Maxwell verstehen

Elektrodynamik mit Python und numerischen Methoden

Dipl. Ing. Sören Gebbert Claude Fable 5

2026-06-20

Inhaltsverzeichnis

Vorwort **1**
Worum es geht . . . 1
Wie man dieses Buch liest . . . 1
Notation . . . 3
Werkzeuge . . . 5

1 Felder sehen **7**
Lernziele . . . 7
1.1 Ein Feld ist eine Zahl an jedem Ort . . . 7
1.2 Ein Feld kann auch ein Pfeil sein . . . 9
1.3 Werkzeug: das Skalarprodukt . . . 12
1.4 Werkzeug: das Kreuzprodukt . . . 12
1.5 Das Feld einer Punktladung . . . 13
1.6 Das Kapitel-Programm: der Dipol . . . 15
Roter Faden . . . 18
Übungen . . . 19
Das Kleingedruckte . . . 22

2 Ableitungen von Feldern **25**
Lernziele . . . 25
2.1 Die Ableitung als Differenz . . . 26
2.2 Der Gradient: bergauf zeigen . . . 28
2.3 Die Divergenz: Quellen aufspüren . . . 30
2.4 Die Rotation: Wirbel aufspüren . . . 31
2.5 Zwei Sätze, die Buchhaltung machen . . . 35
Roter Faden . . . 37
Übungen . . . 37
Das Kleingedruckte . . . 40

3 Die Maxwell-Gleichungen, Term für Term **41**
Lernziele . . . 41
3.1 Die vier Gleichungen auf einen Blick . . . 42
3.2 Gleichung 1: Ladungen sind die Quellen des E-Feldes . . . 42
3.3 Gleichung 2: Es gibt keine magnetischen Ladungen . . . 44
3.4 Gleichung 3: Wandel im Magnetfeld wirbelt das E-Feld . . . 49

3.5 Gleichung 4: Ströme wirbeln das B-Feld — und wandelnde E-Felder auch . . . 52
3.6 Und daraus fällt: das Licht . . . 53
Roter Faden . . . 57
Übungen . . . 57
Das Kleingedruckte . . . 59

4 Wellen und Polarisation **61**
Lernziele . . . 61
4.1 Die ebene Welle lesen . . . 61
4.2 Dieselbe Welle, dreimal gezeichnet — und was davon echt ist . 64
4.3 Quer liegt das Feld: Transversalität . . . 67
4.4 Polarisation: die Schwingungsrichtung ist wählbar . . . 69
4.5 Der Polfilter: ein Skalarprodukt aus Plastik . . . 72
4.6 Die Welle trägt Energie: $\vec{S} = \vec{E} \times \vec{H}$ konkret . . . 73
Roter Faden . . . 76
Übungen . . . 77
Das Kleingedruckte . . . 79

5 Diskretisieren: Maxwell im Daumenkino **81**
Lernziele . . . 81
5.1 Von der Gleichung zur Update-Regel . . . 82
5.2 Der naheliegende Plan — und sein Scheitern . . . 83
5.3 Der bessere Bauplan: doppelt versetzt . . . 87
5.4 Die erste lebende Welle . . . 90
Roter Faden . . . 92
Übungen . . . 92
Das Kleingedruckte . . . 95

6 Das 1D-Wellenlabor **97**
Lernziele . . . 97
6.1 Das Labor . . . 98
6.2 Wände, die schlucken: der Mur-Rand . . . 100
6.3 Senden, ohne zu stören: weiche und harte Quellen . . . 101
6.4 Interferenz nach Plan: die stehende Welle . . . 103
6.5 Verluste: Wellen ausbremsen — und die PML-Idee . . . 105
6.6 Das Spiegelbild vermessen: Fresnel an der Glasgrenze . . . 107
Roter Faden . . . 110
Übungen . . . 110
Das Kleingedruckte . . . 113

7 Explizit oder implizit? Das Update als Matrix **115**
Lernziele . . . 115
7.1 Die Schleife, als Matrix gelesen . . . 116

7.2 Die andere Philosophie: Crank–Nicolson 121
7.3 Wann welches Verfahren? 125
Roter Faden 127
Übungen 127
Das Kleingedruckte 129

8 Stabilität und numerische Dispersion 131
Lernziele 131
8.1 Das Tempolimit, anschaulich 132
8.2 Der Hauptverdächtige: die Zickzackwelle 135
8.3 Unter dem Limit: Wellen im falschen Tempo 138
8.4 Die Dispersionsrelation des Gitters 141
8.5 Messung gegen Theorie 143
8.6 Gitterwahl in der Praxis: Faustregeln 146
8.7 Das Kapitel-Programm 147
Roter Faden 147
Übungen 148

9 FDTD in der Fläche: das volle Yee-Gitter 151
Lernziele 151
9.1 Die halbe Welt genügt: das TMz-Gitter 152
9.2 Das neue Tempolimit 156
9.3 Die Magie wandert in die Diagonale 160
9.4 Offene Ränder, jetzt wirklich: die PML 163
9.5 Ausblick: die dritte Dimension 165
9.6 Das Kapitel-Programm 165
Roter Faden 166
Übungen 166

10 Spektren, Konvergenz, Vertrauen 173
Lernziele 173
10.1 Ein Frequenz-Detektor von Hand 174
10.2 np.fft: derselbe Detektor, nur schnell 176
10.3 Der Chor, sichtbar gemacht 178
10.4 Ein Lauf, alle Resonanzen 179
10.5 Vertrauen ist messbar 183
10.6 Die Checkliste 186
10.7 Das Kapitel-Programm 187
Roter Faden 187
Übungen 188

11 Die Löser-Landschaft 193
Lernziele 193
11.1 Vom Eigenbau zur Werkstatt 194

11.2 Maxwell hat kein Lineal: Meeps Einheiten 195
11.3 Die Ringwelle, zweite Auflage 196
11.4 Querrechnen heißt: der Differenz zuhören 201
11.5 Alte Bekannte I: die PML 203
11.6 Alte Bekannte II: der eingebaute Detektor 206
11.7 Die Landschaft der Verfahren 208
11.8 Die Yagi-Geschichte, zu Ende erzählt 210
11.9 Das Kapitel-Programm 213
Roter Faden 214
Übungen 214
Das Kleingedruckte 217

12 Brechung und Totalreflexion 219
Lernziele 219
12.1 Wellenberge müssen Schritt halten 220
12.2 Der Versuchsaufbau: ein Scheinwerfer im Glas 221
12.3 Wie viel kommt durch? Flux und Fresnel 225
12.4 Der Grenzwinkel: wenn Snellius nicht mehr kann 227
12.5 Das Gespenst über der Grenze 229
12.6 Die Bilanz, die erst beim dritten Monitor schließt 232
12.7 Das Kapitel-Programm 233
Roter Faden 234
Übungen 234
Das Kleingedruckte 237

13 Polarisation und Brewster 239
Lernziele 239
13.1 s und p: die Einfallsebene sortiert das Licht 240
13.2 Fresnel aus der Spiegelformel von Kapitel 6 240
13.3 Die Überraschung im p-Zähler 242
13.4 Die Messung: zwei Kurven, ein Einbruch 243
13.5 Das Foto zum Beweis — und die 90°-Probe 248
13.6 Wozu das gut ist: Brillen und Laser 250
13.7 Das Kapitel-Programm 251
Roter Faden 252
Übungen 252
Das Kleingedruckte 255

14 Interferenz: Doppelspalt und Gitter 257
Lernziele 257
14.1 Der Gangunterschied entscheidet 258
14.2 Zwei Quellen auf der Messbank 259
14.3 Der Doppelspalt: zwei Quellen aus einer Wand 263
14.4 Vom Doppelspalt zum Gitter 266

14.5 Das Gitter als Spektrometer 269
14.6 Das Kapitel-Programm 271
Roter Faden 271
Übungen 272
Das Kleingedruckte 275

15 Resonanz und Güte Q **277**
Lernziele 277
15.1 Der Kamm bekommt zwei Indizes 278
15.2 Die Güte: Schwingungen pro Energievorrat 282
15.3 Eine Linie kann nicht scharf bleiben 283
15.4 Q, dreifach gemessen 285
15.5 Löcher: Verlust durch Strahlung 288
15.6 Das Kapitel-Programm 291
Roter Faden 292
Übungen 292
Das Kleingedruckte 295

16 Dispersion: Phasen- und Gruppengeschwindigkeit **297**
Lernziele 297
16.1 Zwei Wellen genügen: die Schwebung 298
16.2 Ein alter Bekannter: das Gitter ist dispersiv 302
16.3 Warum Glas sortiert: der Lorentz-Oszillator 307
16.4 Das Prisma sortiert andersherum 315
16.5 Freie Elektronen: Drude, Metalle, Ionosphäre 320
16.6 Plasmonik: eine Goldscheibe unter Licht 324
16.7 Das Kapitel-Programm 331
Roter Faden 332
Übungen 332
Das Kleingedruckte 337

17 Tunneln und Cutoff **339**
Lernziele 339
17.1 Eine Brücke für den Saum 340
17.2 Das Rohr und die Klippe 347
17.3 Im Rohr wohnt ein Plasma 352
17.4 Warum die Glasfaser ihr Licht behält 355
17.5 Das Kapitel-Programm 355
Roter Faden 356
Übungen 356
Das Kleingedruckte 363

18 Poynting und Energiefluss **365**
Lernziele 365

18.1 Zwei Töpfe: die Energie im Feld . . . 366
18.2 Die Buchhaltung: der Poynting-Satz . . . 369
18.3 Die Landkarte: wo die Leistung wirklich läuft . . . 374
18.4 Wie schnell reist Energie? . . . 380
18.5 Das Kapitel-Programm . . . 382
Roter Faden . . . 382
Übungen . . . 383
Das Kleingedruckte . . . 388

19 Oszillator und Energie-Umpacken 389
Lernziele . . . 389
19.1 Erst die Begriffe: Was ist was? . . . 390
19.2 Zwei Töpfe werden Bauteile . . . 391
19.3 Das kleinste Yee-Gitter . . . 392
19.4 Drei Schicksale einer Schwingung . . . 398
19.5 Der Hohlraum ist ein Schwingkreis ohne Bauteile . . . 403
19.6 Vom Kraftwerk zur Antenne . . . 407
19.7 Das Kapitel-Programm . . . 409
Roter Faden . . . 409
Übungen . . . 410
Das Kleingedruckte . . . 413

20 Kausalität, Zeitumkehr und Zeitpfeil 415
Lernziele . . . 415
20.1 Ist die Energie das Primäre? . . . 416
20.2 Gleichungen ohne Richtungspfeil . . . 417
20.3 Der Film läuft rückwärts: das 1D-Labor . . . 418
20.4 Der gedämpfte Kreis rückwärts: eine Überraschung . . . 421
20.5 2D: der Absorber gewinnt . . . 424
20.6 Zeitumkehr als Technik . . . 430
20.7 Das Kapitel-Programm . . . 431
Roter Faden . . . 431
Übungen . . . 432
Das Kleingedruckte . . . 439

21 Vom Strom zur Welle 441
Lernziele . . . 441
21.1 Warum strahlt es überhaupt? Der Knick . . . 442
21.2 Die Antenne als Knick-Maschine . . . 446
21.3 Das Feld macht sich selbstständig . . . 451
21.4 Nah und fern — gemessen in 3D . . . 455
21.5 In welche Richtung? Das Strahlungsdiagramm . . . 463
21.6 Wie weit trägt das? Die Reichweitenrechnung . . . 468
21.7 Das Kapitel-Programm . . . 469

Roter Faden 469
Übungen 470
Das Kleingedruckte 477

22 Leitungen, Reflexion und Anpassung 479
Lernziele 479
22.1 Die Leitung: ein Feld in Schaltkreis-Kleidern 480
22.2 Das Leitungs-Labor 482
22.3 Mit der Uhr Fehler orten: TDR 485
22.4 Stehwellen und das SWR 487
22.5 Anpassen: der λ/4-Transformator 490
22.6 Die Lampe: Kirchhoff als Grenzwert 493
22.7 Das Kapitel-Programm 496
Roter Faden 497
Übungen 497
Das Kleingedruckte 502

23 Arrays und Beamforming 505
Lernziele 505
23.1 Umweg = Vorsprung: die Geometrie des Schwenks 506
23.2 Der Array-Faktor: die Zeigersumme bekommt einen Phasenknopf 507
23.3 Maxwell bestätigt: das Feldbild in Meep 511
23.4 Grating Lobes: die Geisterkeulen 517
23.5 Element mal Array: der Faltungssatz der Antennentechnik . . 522
23.6 Taper: die Nebenkeulen zähmen 523
23.7 Scan Loss: der Preis des Schwenkens 527
23.8 Empfang ist Senden rückwärts 532
23.9 Das Kapitel-Programm 532
Roter Faden 533
Übungen 533
Das Kleingedruckte 536

24 Fallstudie Yagi: Entwurf, Validierung, Bauplan 539
Lernziele 539
24.1 Endfire: die Keule längs der Reihe 540
24.2 Woher die Phase kommt: der getriebene Resonator 542
24.3 Die Umschaltkarte: kurz führt, lang wirft zurück 545
24.4 Das Feldbild: die Welle klettert die Leiter entlang 549
24.5 Der Entwurf: Schritt für Schritt zur 5-Element-Yagi 552
24.6 Wie genau ist diese Rechnung? Die Methodengrenze 555
24.7 Bandbreite: schmal — und nach oben kippt sie 556
24.8 Der Bauplan 559
24.9 Das Kapitel-Programm 562
Roter Faden 563

Übungen 563
Das Kleingedruckte 567

25 Die elektrisch kleine Antenne — warum 50 Hz nicht funkt 569
Lernziele 569
25.1 Der Strahlungswiderstand: Funk als Widerstand im Schaltplan 570
25.2 Die 50-Hz-Längenrechnung 574
25.3 Pendeln statt senden 575
25.4 Das erste Gefängnis: der Blindanteil-Abgrund 578
25.5 Das zweite Gefängnis: die Chu-Grenze 578
25.6 Drei Auswege — und was sie wirklich sind 581
25.7 Das Kapitel-Programm 581
Roter Faden 582
Übungen 583
Das Kleingedruckte 586

26 Photonische Kristalle und Bandlücken 589
Lernziele 589
26.1 Viele schwache Echos, ein starker Spiegel 590
26.2 Die Transfer-Matrix: alle Echos auf einmal 596
26.3 Das Spurkriterium: erlaubt oder verboten 598
26.4 Vom Stapel zum Kristall 604
26.5 Der Defekt: Licht um die Ecke 609
26.6 Strukturfarbe: der Falter, nachgerechnet 613
26.7 Das Kapitel-Programm 616
Roter Faden 617
Übungen 617
Das Kleingedruckte 624

27 Ringresonator und Fabry-Pérot 625
Lernziele 625
27.1 Die Echo-Summe: vom Klingeln zum Kamm 626
27.2 Der Kamm im Experiment 631
27.3 Der Ringresonator: Fabry-Pérot ohne Spiegel 637
27.4 Kritische Kopplung: der perfekt schwarze Dip 644
27.5 Der Schritt zum Laser 646
27.6 Das Kapitel-Programm 651
Roter Faden 651
Übungen 652
Das Kleingedruckte 660

28 Metamaterialien und Doppelbrechung 661
Lernziele 661
28.1 Verbotene Vorzeichen? 662

28.2 Die Zutaten: Drähte und Ringe . . . 663
28.3 Das Smith-Experiment: zwei Sperren öffnen ein Fenster . . . 667
28.4 Die Phase läuft rückwärts . . . 671
28.5 Negative Brechung: der Knick auf die falsche Seite . . . 675
28.6 Die Veselago-Flachlinse . . . 678
28.7 Doppelbrechung: ein Material, zwei Indizes . . . 682
28.8 Walk-off: das Doppelbild bei senkrechtem Einfall . . . 685
28.9 Die λ/4-Platte: aus linear wird zirkular . . . 689
28.10 Das Kapitel-Programm . . . 691
Roter Faden . . . 692
Übungen . . . 692
Das Kleingedruckte . . . 700

29 Kramers-Kronig: Kausalität erzwingt Dispersion 703
Lernziele . . . 703
29.1 Die Antwort kommt zu spät . . . 704
29.2 Die halbe Funktion bestimmt die ganze . . . 707
29.3 Das Hauptwert-Integral zähmen . . . 709
29.4 Die Kausalitäts-Probe . . . 711
29.5 Die Absorption ist global festgenagelt . . . 714
29.6 Realwelt-Probe: Meeps Goldkarte . . . 715
29.7 Die Front läuft nie schneller als c . . . 718
29.8 Das Kapitel-Programm . . . 724
Roter Faden . . . 725
Übungen . . . 725
Das Kleingedruckte . . . 731

30 Wo Maxwell endet: Photonen und Wahrscheinlichkeit 735
Lernziele . . . 735
30.1 Der Klick — und warum wir ihn nie bemerken . . . 736
30.2 Die Schwelle sitzt in der Frequenz . . . 737
30.3 Der Doppelspalt, ein Photon nach dem anderen . . . 738
30.4 Die Regel dahinter: Maxwell als Amplituden-Maschine . . . 742
30.5 Schrotrauschen: die Körnigkeit wird messbar . . . 744
30.6 Die Tunnel-Brücke: eine alte Bekannte . . . 745
30.7 Die ehrliche Grenzziehung . . . 747
30.8 Das Kapitel-Programm . . . 747
Roter Faden . . . 748
Übungen . . . 749
Das Kleingedruckte . . . 753

31 Inverses Design: der Computer als Entwerfer 755
Lernziele . . . 755
31.1 Vorwärts, rückwärts, Zielfunktion . . . 756

31.2 Der Computer erfindet den Spiegel 756
31.3 Minimieren statt maximieren: der Antireflex-Belag 760
31.4 Die Landschaft: warum Multistart Pflicht ist 761
31.5 Das Pixel-Puzzle: Topologie-Optimierung im Kleinen 763
31.6 Gradienten, Adjoint und die große Welt 767
31.7 Das Kapitel-Programm 767
Roter Faden 768
Übungen 769
Das Kleingedruckte 775
Schlusswort: die Reise 776

32 Zugabe: Schall — dieselben Gleichungen in deiner Werkstatt 779
Lernziele 779
32.1 Zwei Gleichungen, neu beschriftet 780
32.2 Der Kern, portiert 782
32.3 Die Wellenlängen-Werkstatt-Landkarte 787
32.4 Der phononische Kristall 788
32.5 Vom Gitter zum Bauplan 792
32.6 Schall um die Ecke: der Defekt-Kanal 795
32.7 Das akustische Pixel-Puzzle 800
32.8 Das Kapitel-Programm 803
Roter Faden 804
Übungen 804
Das Kleingedruckte 808

Vorwort

Worum es geht

Vier Gleichungen beschreiben alles, was Licht, Funk, Strom und Magnetismus tun: die **Maxwell-Gleichungen**. Dieses Buch erklärt sie nicht mit Formelseiten, sondern mit **Python**: Jede Größe, jede Gleichung, jedes Phänomen wird sofort numerisch angefasst — Code ausführen, Bild ansehen, verstehen.

Der Weg ist lang, aber er führt weit: von den ersten Pfeilen eines Vektorfeldes (Kapitel 1) über ein selbstgebautes Simulationsprogramm aus dreißig Zeilen NumPy (Teil II) bis zu Antennen, photonischen Kristallen, Metamaterialien und einem Computer, der optische Bauteile selbst entwirft (Teil VIII). Die acht Teile bauen streng aufeinander auf — das Buch ist zum **linearen Lesen** gedacht, jedes Kapitel benutzt, was die früheren aufgebaut haben. Wer am Ende noch nicht genug hat, findet eine Zugabe: dieselben Gleichungen, übersetzt in die Welt des Schalls und gebaut aus Holzdübeln (Kapitel 32).

Du brauchst dafür:

- **Python-Grundkenntnisse** (Schleifen, Funktionen, NumPy-Arrays werden benutzt, aber jede neue Bibliotheksfunktion wird beim ersten Auftreten erklärt),
- **Schulmathematik** (Sinus, Vektoren als Pfeile, Steigung). Alles darüber hinaus — Ableitungen von Feldern, Rotation, komplexe Zahlen — wird in diesem Buch von Grund auf entwickelt, und zwar *numerisch*: Eine Ableitung ist hier zuerst eine Differenz zweier Array-Einträge, dann erst ein Symbol.

Wie man dieses Buch liest

Dieses Buch ist zum **Mitmachen** gebaut, nicht zum Durchblättern. Ein paar wiederkehrende Bausteine solltest du kennen:

- **Vorhersage-Punkte** — vor größeren Programmläufen wirst du gefragt: *„Was erwartest du — und warum?“* Halte kurz inne und lege dich fest, bevor du weiterliest. Dieses Festlegen (auch wenn die Vorhersage danebenliegt — *gerade* dann) ist der wirksamste Lernschritt im Buch.
- **„Naheliegende Vermutung”-Kästchen** — viele Fehlvorstellungen über Felder sind völlig vernünftig und trotzdem falsch. Diese Kästchen nehmen die Vermutung ernst, erklären, warum sie naheliegt — und was stattdessen stimmt.
- **Merkkästen** — die drei bis fünf Sätze, die vom Kapitel bleiben sollen.
- **Roter Faden** — jedes Kapitel sagt am Ende, wo seine Ideen schon vorkamen und wo sie wieder auftauchen.
- **Übungen** — gestuft in *Verstehen* (ohne Rechner beantwortbar), *Verändern* (das Kapitel-Programm gezielt umbauen) und *Übertragen* (etwas Eigenes damit bauen). Musterlösungen sind in der HTML-Fassung aufklappbar.

Dieses Buch gibt es in zwei Fassungen aus *einer* Quelle. Die **HTML-Fassung** ist lebendig: Viele Abbildungen sind in Wahrheit **Animationen** mit Abspielknopf (eine Welle, die wirklich läuft, sagt mehr als jedes Standbild), und am Ende der meisten Kapitel stehen **Spielwiesen** — kleine Programme, die direkt im Browser laufen und die du an Ort und Stelle umschreiben kannst: an einer Schraube drehen, „Run” drücken, das Ergebnis sehen. Die **PDF-Fassung** ist fürs Drucken und Offline-Lesen gedacht; dort werden aus den Animationen Standbild-Streifen, und die Bildunterschrift nennt den Namen der zugehörigen Datei. Wer also die volle Wirkung will, liest am Bildschirm und probiert mit.

Und eine Verabredung zum Code: **Die Code-Zellen eines Kapitels teilen sich ihren Zustand** — wie die Zellen eines Notebooks. Importe, Konstanten und Funktionen gelten ab ihrer ersten Zelle im ganzen Kapitel weiter; `import numpy as np` und `matplotlib` stehen deshalb nur einmal am Kapitelanfang. Damit du nie rätseln musst, wo ein Name herkommt, beginnt jede Zelle, die Definitionen aus früheren Zellen benutzt, mit einem **Herkunfts-Kommentar**:

```
# von oben: simuliere(), gauss(), X   (Abschnitt „Das Labor")
```

Das gilt auch für importierte Naturkonstanten wie `c`, `mu_0` oder `epsilon_0` (sie kommen stets aus `scipy.constants`) — nur die Modul-Kürzel `np` und `plt` werden nicht eigens erwähnt.

Zwischen den Kapiteln wird dagegen **nichts** geteilt — jedes Kapitel startet frisch. Und wer alles an einem Stück sehen will: Jedes Kapitel hat ein **eigenständiges Programm** in `programme/kapNN/` — ein gewöhnliches `.py`-Skript oh-

ne jeden versteckten Zustand, das du ohne das Buch ausführen, lesen und umbauen kannst.

Notation

Diese Symbole tauchen im ganzen Buch auf; jedes wird bei seinem ersten Auftritt ausführlich eingeführt. Die Tabelle ist zum **Zurückblättern** da.

Symbol	Bedeutung	Einheit
$\vec{E}$	elektrisches Feld	V/m
$\vec{H}$, $\vec{B}$	magnetisches Feld / Flussdichte	A/m, T
ρ, $\vec{J}$	Ladungsdichte, Stromdichte	C/m³, A/m²
ε_0, μ_0	elektrische / magnetische Feldkonstante	As/(Vm), Vs/(Am)
c	Lichtgeschwindigkeit, $c = 1/\sqrt{\varepsilon_0 \mu_0}$	m/s
f, ω	Frequenz, Kreisfrequenz $\omega = 2\pi f$	Hz, 1/s
λ, k	Wellenlänge, Wellenzahl $k = 2\pi/\lambda$	m, 1/m
$\vec{S}$	Poynting-Vektor $\vec{S} = \vec{E} \times \vec{H}$	W/m²
$\nabla\cdot$, $\nabla\times$	Divergenz (Quellstärke), Rotation (Wirbelstärke)	—
Δx, Δt	Gitterabstand, Zeitschritt (Numerik)	m, s

Die Einheiten dahinter

Alle Einheiten der Tabelle sind aus diesen Bausteinen zusammengesetzt. Die Alltagsanker geben ein Gefühl für die Größenordnung:

Einheit	Name	misst	Alltagsanker
m	Meter	Länge	—

Einheit	Name	misst	Alltagsanker
s	Sekunde	Zeit	—
kg	Kilogramm	Masse	—
A	Ampere	elektrischen Strom (Ladung pro Sekunde)	Haushaltssicherung 16 A
C	Coulomb	elektrische Ladung; 1 C = 1 As	ein Blitz: einige C
V	Volt	elektrische Spannung — den „Antrieb” pro Ladung	Batterie 1,5 V; Steckdose 230 V
J	Joule	Energie	1 g Schokolade ≈ 20 000 J
W	Watt	Leistung (Energie pro Sekunde); 1 W = 1 J/s	LED-Lampe ≈ 5 W; Wasserkocher ≈ 2000 W
Hz	Hertz	Frequenz (Schwingungen pro Sekunde); 1 Hz = 1/s	Stromnetz 50 Hz; WLAN 2,4 GHz
T	Tesla	magnetische Flussdichte; 1 T = 1 Vs/m²	Erdmagnetfeld ≈ 50 µT; MRT 1,5–3 T

Zusammengesetzte Einheiten liest du wörtlich: **V/m** (elektrisches Feld) ist Spannungsgefälle pro Meter Weg; **A/m²** (Stromdichte) ist Strom pro Querschnittsfläche; **W/m²** (Energiefluss) ist Leistung, die durch einen Quadratmeter Fläche tritt — der Anker dafür: Sonnenschein am Mittag liefert etwa 1000 W/m². Auch die beiden Feldkonstanten entschlüsseln sich so: ε_0 in As/(Vm) verknüpft Ladung mit Spannung pro Länge, μ_0 in Vs/(Am) Strom mit Spannung mal Zeit pro Länge — ihre Zahlenwerte fallen das erste Mal in Kapitel 3 an, wo wir mit ihnen rechnen.

Vorsilben skalieren jede Einheit in Tausenderschritten: µ (Mikro, 10^{-6}), m (Milli, 10^{-3}), k (Kilo, 10^{3}), M (Mega, 10^{6}), G (Giga, 10^{9}) — aus 2 400 000 000 Hz wird so das handliche 2,4 GHz.

Werkzeuge

Alle Programme laufen mit Python 3, NumPy und Matplotlib; ab Teil III kommen die Feldlöser **Meep** (FDTD), **NEC** (Momentenmethode) und **openEMS** (FIT) dazu. Die Einrichtung der Umgebung wird dort erklärt, wo sie erstmals gebraucht wird.

Ein letzter Rat, bevor es losgeht: Lies dieses Buch mit der Tastatur in Reichweite. Die Gleichungen verstehst du nicht, indem du sie liest, sondern indem du sie laufen lässt, an ihnen drehst und dich überraschen lässt, wo deine Vorhersage danebenlag. Genau dafür ist alles hier gebaut. Viel Freude.

1 Felder sehen

Schau auf eine Wetterkarte: An jedem Ort des Landes steht eine Temperatur — und an jedem Ort weht ein Wind mit Richtung und Stärke. Beides sind **Felder**: Die Temperaturkarte ordnet jedem Ort eine *Zahl* zu, die Windkarte jedem Ort einen *Pfeil*. Mehr ist ein Feld nicht.

Das Verblüffende: Das elektrische Feld, mit dem dieses ganze Buch arbeitet, ist genau dasselbe — ein Pfeil an jedem Ort des Raums. Wer eine Windkarte lesen kann, hat das schwerste Konzept der Elektrodynamik schon verstanden. In diesem Kapitel machen wir Felder zu dem, was sie für uns als Programmierer sind: **NumPy-Arrays, die man zeichnen kann.**

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, was ein Skalarfeld und ein Vektorfeld ist — und beide als NumPy-Arrays anlegen,
2. … Felder sichtbar machen (`imshow`/`contour` für Zahlen, `quiver` und `streamplot` für Pfeile) und begründen, welche Darstellung wann passt,
3. … das Skalarprodukt geometrisch deuten („wie viel zeigt $\vec{a}$ in Richtung $\vec{b}$?“) und numerisch nachprüfen,
4. … das Kreuzprodukt geometrisch deuten (senkrechter Pfeil, Rechte-Hand-Regel, Parallelogrammfläche) und mit `np.cross` berechnen,
5. … das Feldbild einer Punktladung und eines Dipols **vorhersagen** und selbst berechnen.

1.1 Ein Feld ist eine Zahl an jedem Ort

Ein **Skalarfeld** ordnet jedem Punkt (x, y) eine Zahl zu — etwa eine Temperatur $T(x, y)$. Für den Computer heißt das: Wir legen ein Gitter aus Punkten über die Fläche und speichern an jedem Gitterpunkt einen Wert. Genau das leistet `np.meshgrid` plus ein 2D-Array:

```python
import numpy as np
import matplotlib.pyplot as plt

# Gitter: 200 × 200 Punkte auf dem Quadrat [-2, 2] × [-2, 2].
# np.linspace(a, b, n): n gleichmäßig verteilte Werte von a bis b.
# np.meshgrid: spannt aus zwei 1D-Achsen das 2D-Gitter auf — x[i, j]
# und y[i, j] sind die Koordinaten des Gitterpunkts in Zeile i, Spalte j
x, y = np.meshgrid(np.linspace(-2, 2, 200), np.linspace(-2, 2, 200))

# Ein erfundenes Temperaturfeld: warme Insel links, kalte Senke rechts.
# np.exp wirkt — wie alle NumPy-Funktionen — elementweise auf das ganze
# Array: eine Zeile Code, 40 000 Rechnungen
T = 20 + 6*np.exp(-((x+1)**2 + y**2)) - 4*np.exp(-((x-1)**2 + (y-0.5)**2))

fig, ax = plt.subplots(figsize=(5.6, 4.4))
bild = ax.imshow(T, extent=(-2, 2, -2, 2), origin="lower", cmap="coolwarm")
linien = ax.contour(x, y, T, colors="k", linewidths=0.5)
ax.clabel(linien, fontsize=7, fmt="%.0f °C")
fig.colorbar(bild, label="Temperatur (°C)")
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("Skalarfeld T(x, y)")
plt.tight_layout(); plt.show()
```

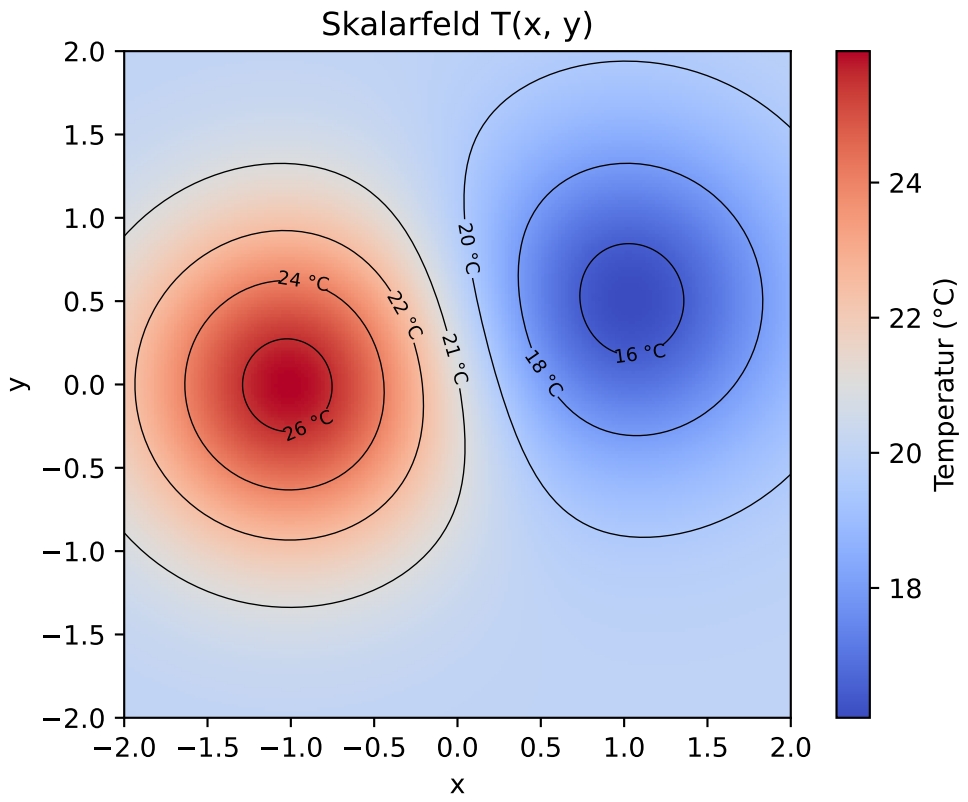


Abbildung 1.1: Ein Skalarfeld: an jedem Ort eine Zahl. Farbe = Temperatur, Linien = Orte gleicher Temperatur (wie Höhenlinien auf der Wanderkarte).

Das Array `T` **ist** das Feld — eine Tabelle mit einer Zahl pro Gitterpunkt. Zwei Darstellungen auf einen Blick: Die *Farbe* zeigt den Wert direkt, die *Konturlinien* verbinden Orte gleichen Werts. Auf solchen Linien wirst du im ganzen Buch herumlaufen — sie heißen später nur anders (Äquipotentiallinien, Wellenfronten).

1.2 Ein Feld kann auch ein Pfeil sein

Ein **Vektorfeld** ordnet jedem Punkt einen *Pfeil* zu: Richtung und Länge. Numerisch sind das einfach **zwei** Arrays — eines für die x-Komponente des Pfeils, eines für die y-Komponente.

Vorhersage-Punkt

Bevor du weiterliest: Wir zeichnen gleich das Vektorfeld mit den Komponenten $u = -y$ und $v = x$, d. h. der Pfeil am Ort (x, y) ist $(-y, x)$. Nimm ein Blatt Papier und trage die Pfeile an den vier Punkten $(1, 0)$, $(0, 1)$, $(-1, 0)$, $(0, -1)$ ein. Was für ein Strömungsmuster entsteht? Lege dich fest.

```
x, y = np.meshgrid(np.linspace(-2, 2, 15), np.linspace(-2, 2, 15))
u = -y      # x-Komponente des Pfeils an jedem Gitterpunkt
v = x       # y-Komponente

fig, ax = plt.subplots(figsize=(4.6, 4.4))
# np.hypot(u, v): elementweise sqrt(u² + v²) — hier die Pfeillänge,
# die als fünftes Argument die Farbe der Pfeile steuert
ax.quiver(x, y, u, v, np.hypot(u, v), cmap="viridis")
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("Vektorfeld (u, v) = (-y, x)")
ax.set_aspect("equal")
plt.tight_layout(); plt.show()
```

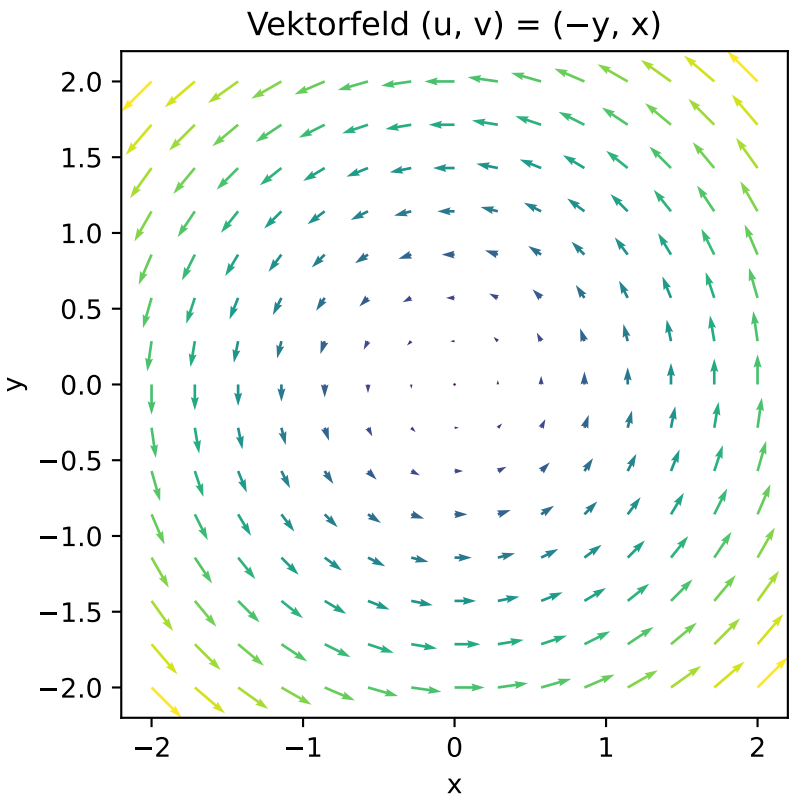


Abbildung 1.2: Ein Vektorfeld: an jedem Ort ein Pfeil. Das Feld (−y, x) ist ein Wirbel gegen den Uhrzeigersinn; außen sind die Pfeile länger.

Ein **Wirbel**: Am Punkt $(1, 0)$ zeigt der Pfeil nach oben $(0, 1)$, am Punkt $(0, 1)$ nach links $(-1, 0)$ — die Strömung kreist gegen den Uhrzeigersinn, außen schneller als innen. Wenn deine Skizze das zeigt: sehr gut. Wenn nicht: Geh die vier Punkte noch einmal durch — der häufigste Stolperstein ist, $(-y, x)$ am Ort einzusetzen, statt es als festen Pfeil zu lesen.

`quiver` („Köcher") zeichnet die rohen Pfeile. Das ist die ehrlichste Darstellung eines Vektorfelds — und unser Arbeitspferd, bis die Pfeile so unterschiedlich lang werden, dass das Bild unleserlich wird. Dann kommt `streamplot` ins Spiel; das siehst du unten beim Dipol.

1.3 Werkzeug: das Skalarprodukt

Bevor wir echte elektrische Felder bauen, brauchen wir zwei Handgriffe für Pfeile. Der erste beantwortet die Frage: **„Wie viel von $\vec{a}$ zeigt in die Richtung von $\vec{b}$?“** Das ist das Skalarprodukt — zwei Pfeile rein, *eine Zahl* raus:

$$\vec{a} \cdot \vec{b} \;=\; a_x b_x + a_y b_y \;=\; |\vec{a}|\,|\vec{b}|\,\cos\alpha.$$

Die erste Form ist die Rechenvorschrift (Komponenten multiplizieren, aufsummieren), die zweite die Bedeutung: Länge von $\vec{a}$ mal Länge von $\vec{b}$ mal „wie parallel sind sie” ($\cos\alpha$, mit dem Winkel α zwischen den Pfeilen). Beide Formen liefern dieselbe Zahl — das prüfen wir nach, statt es zu glauben:

```
a = np.array([3.0, 1.0])        # np.array: macht aus einer Liste einen Vektor
b = np.array([1.0, 2.0])

links  = a @ b                  # der @-Operator: das Skalarprodukt
laenge_a, laenge_b = np.linalg.norm(a), np.linalg.norm(b)   # norm: Länge |a|
alpha  = np.arccos(links / (laenge_a * laenge_b))   # arccos: Kosinus rückwärts
rechts = laenge_a * laenge_b * np.cos(alpha)        # geometrische Form

print(f"Komponentenform:  a·b = {links:.4f}")
# np.degrees: Bogenmaß → Grad (np.radians ginge den umgekehrten Weg)
print(f"geometrische Form:      {rechts:.4f}   (Winkel α = {np.degrees(alpha):.1f}°)")
```

```
Komponentenform:  a·b = 5.0000
geometrische Form:      5.0000   (Winkel α = 45.0°)
```

Drei Fälle solltest du im Schlaf erkennen: Pfeile **parallel** → Skalarprodukt maximal positiv. Pfeile **senkrecht** → exakt null ($\cos 90° = 0$). Pfeile **entgegengesetzt** → negativ. Dieses „null bei senkrecht” wird unser meistbenutzter Test: Ob ein Feld durch eine Fläche *hindurch* zeigt oder nur an ihr *entlangstreicht*, entscheidet in Kapitel 3 ein Skalarprodukt.

1.4 Werkzeug: das Kreuzprodukt

Der zweite Handgriff beantwortet eine andere Frage: **„Welcher Pfeil steht senkrecht auf $\vec{a}$ *und* $\vec{b}$?“** Das Kreuzprodukt $\vec{a} \times \vec{b}$ liefert genau diesen Pfeil

— zwei Pfeile rein, *ein Pfeil* raus. Seine Länge ist die Fläche des von $\vec{a}$ und $\vec{b}$ aufgespannten Parallelogramms, $|\vec{a} \times \vec{b}| = |\vec{a}||\vec{b}| \sin\alpha$, und von den zwei möglichen senkrechten Richtungen gilt die **Rechte-Hand-Regel**: Zeigt der Daumen entlang $\vec{a}$ und der Zeigefinger entlang $\vec{b}$, dann zeigt der Mittelfinger entlang $\vec{a} \times \vec{b}$.

```
a = np.array([2.0, 0.0, 0.0])     # entlang x
b = np.array([1.0, 1.0, 0.0])     # schräg in der xy-Ebene

c = np.cross(a, b)              # np.cross: Kreuzprodukt zweier 3er-Vektoren
print(f"a × b = {c}")
print(f"senkrecht auf a? a·c = {a @ c}    senkrecht auf b? b·c = {b @ c}")

flaeche_geometrisch = (np.linalg.norm(a) * np.linalg.norm(b)
                       * np.sin(np.arccos(a @ b / (np.linalg.norm(a) * np.linalg.norm(b)))))
print(f"|a × b| = {np.linalg.norm(c):.4f}   "
      f"Parallelogrammfläche = {flaeche_geometrisch:.4f}")
```

```
a × b = [0. 0. 2.]
senkrecht auf a? a·c = 0.0    senkrecht auf b? b·c = 0.0
|a × b| = 2.0000   Parallelogrammfläche = 2.0000
```

Beide Skalarprodukte sind null — $\vec{c}$ steht tatsächlich senkrecht auf beiden —, und die Länge stimmt mit der Parallelogrammfläche überein. Beachte: Für das Kreuzprodukt mussten die Pfeile **dreidimensional** werden (dritte Komponente 0); das Ergebnis $(0, 0, 2)$ ragt aus der Papierebene heraus.

Warum dieses Werkzeug? Ein Vorgriff: Die Richtung, in die eine elektromagnetische Welle ihre Energie trägt, ist $\vec{S} = \vec{E} \times \vec{H}$ — ein Kreuzprodukt aus elektrischem und magnetischem Feld. Wenn dir in Kapitel 18 jemand $\vec{E}$ und $\vec{H}$ zeigt, kannst du mit der rechten Hand ablesen, wohin das Licht fließt.

1.5 Das Feld einer Punktladung

Jetzt bauen wir unser erstes *physikalisches* Feld. Eine Punktladung q im Ursprung erzeugt das elektrische Feld

$$\vec{E}(\vec{r}) \;=\; \underbrace{\frac{q}{4\pi\varepsilon_0}}_{\text{Stärke}} \frac{\hat{r}}{r^2} \qquad \text{mit } \hat{r} = \frac{\vec{r}}{r},\; r = |\vec{r}|.$$

In Worten: Der Feldpfeil zeigt **vom Ort der Ladung weg** (Richtung $\hat{r}$) und wird mit dem Quadrat des Abstands schwächer ($1/r^2$). Den konstanten Vorfaktor $q/(4\pi\varepsilon_0)$ setzen wir beim Zeichnen auf 1 — er skaliert nur alle Pfeile gemeinsam (mehr dazu im Kleingedruckten).

```
def punktladungsfeld(q, ort, x, y):
    """E = q · r̂ / r² einer Punktladung am gegebenen Ort (Vorfaktor = 1)."""
    dx, dy = x - ort[0], y - ort[1]
    # np.maximum: elementweises Maximum zweier Arrays/Werte — hebt r
    # überall auf mindestens 1e-9, damit nie durch 0 geteilt wird
    r = np.maximum(np.hypot(dx, dy), 1e-9)
    return q * dx / r**3, q * dy / r**3      # q·r/|r|³  =  q·r̂/r²

# 20 Punkte je Achse: so liegt kein Gitterpunkt exakt auf der Ladung
x, y = np.meshgrid(np.linspace(-2, 2, 20), np.linspace(-2, 2, 20))
ex, ey = punktladungsfeld(+1.0, (0, 0), x, y)
staerke = np.hypot(ex, ey)

fig, ax = plt.subplots(figsize=(5.2, 4.4))
# np.log10: Zehnerlogarithmus (elementweise) — staucht den riesigen
# Wertebereich der Feldstärke auf eine lesbare Farbskala
pfeile = ax.quiver(x, y, ex/staerke, ey/staerke, np.log10(staerke),
                   cmap="viridis", pivot="mid", scale=30)
ax.plot(0, 0, "ro", markersize=10)
fig.colorbar(pfeile, label="log₁₀ |E|  (willk. Einh.)")
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("Punktladung: E zeigt radial nach außen")
ax.set_aspect("equal")
plt.tight_layout(); plt.show()
```

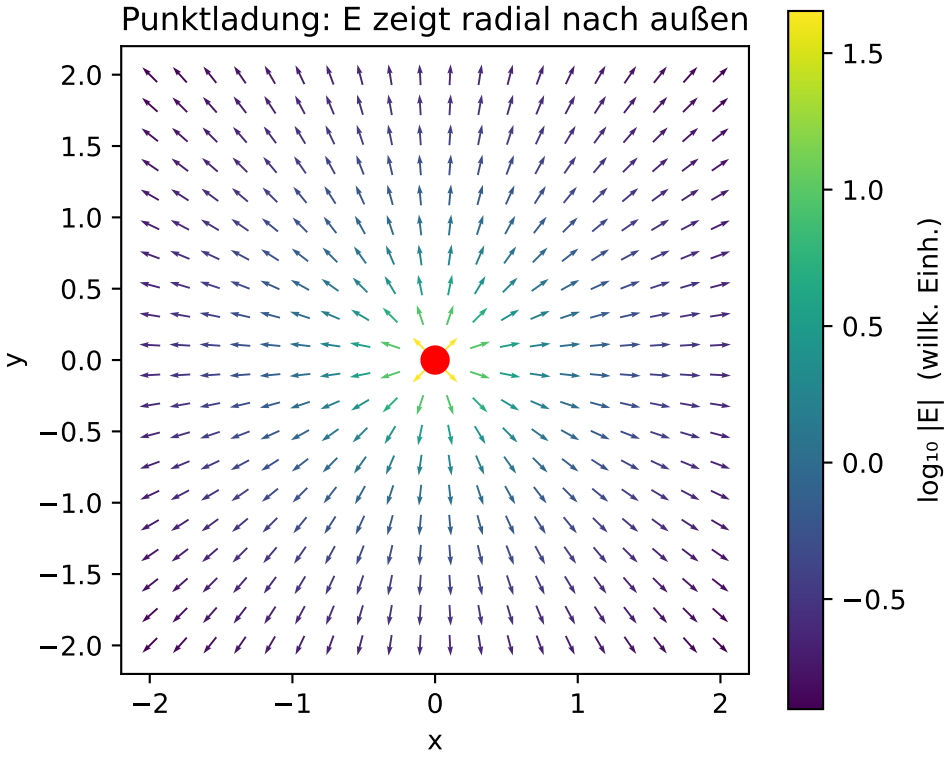


Abbildung 1.3: Das Feld einer positiven Punktladung: alle Pfeile zeigen radial nach außen. Die Pfeile sind auf Einheitslänge gebracht; die Stärke |E| steckt in der Farbe (logarithmisch — von innen nach außen fällt sie um Größenordnungen).

Radiale Pfeile — und das $1/r^2$ steckt in der Farbe: In doppelter Entfernung ist das Feld viermal schwächer. Hier siehst du auch den ersten ehrlichen **Visualisierungs-Trick**: Die Pfeile sind alle gleich lang gezeichnet (`ex/staerke`), sonst würden die riesigen Pfeile nahe der Ladung das Bild erschlagen. Richtung und Stärke sind im Bild getrennt — Pfeil und Farbe.

1.6 Das Kapitel-Programm: der Dipol

Was passiert mit zwei Ladungen? Die Antwort ist das vielleicht wichtigste Prinzip dieses Buchs, und es ist erfreulich schlicht: **Felder addieren sich.** An jedem Ort ist das Gesamtfeld die Pfeilsumme der Einzelfelder (*Superposition*). Eine positive und eine negative Ladung nebeneinander — ein **Dipol** — sind damit zwei Funktionsaufrufe und ein Plus:

Vorhersage-Punkt

Bevor du weiterliest: Skizziere das Linienbild: $+q$ links, $-q$ rechts. Wo starten die Linien, wo enden sie, wie laufen sie dazwischen? Und: In welche Richtung zeigt das Feld auf der Mittelsenkrechten (der senkrechten Linie genau zwischen beiden Ladungen)?

```
# von oben: punktladungsfeld()   (Abschnitt „Das Feld einer Punktladung")
x, y = np.meshgrid(np.linspace(-2, 2, 200), np.linspace(-2, 2, 200))

ex1, ey1 = punktladungsfeld(+1.0, (-0.5, 0.0), x, y)
ex2, ey2 = punktladungsfeld(-1.0, (+0.5, 0.0), x, y)
ex, ey = ex1 + ex2, ey1 + ey2          # Superposition: Felder addieren sich

fig, ax = plt.subplots(figsize=(5.6, 4.6))
strom = ax.streamplot(x, y, ex, ey, color=np.log10(np.hypot(ex, ey)),
                      cmap="viridis", density=1.4, linewidth=0.9)
ax.plot(-0.5, 0, "ro", markersize=10); ax.annotate("+q", (-0.55, 0.12), color="red")
ax.plot(+0.5, 0, "bo", markersize=10); ax.annotate("−q", (0.45, 0.12), color="blue")
fig.colorbar(strom.lines, label="log₁₀ |E|  (willk. Einh.)")
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("Dipol: Superposition zweier Punktladungsfelder")
ax.set_aspect("equal")
plt.tight_layout(); plt.show()
```

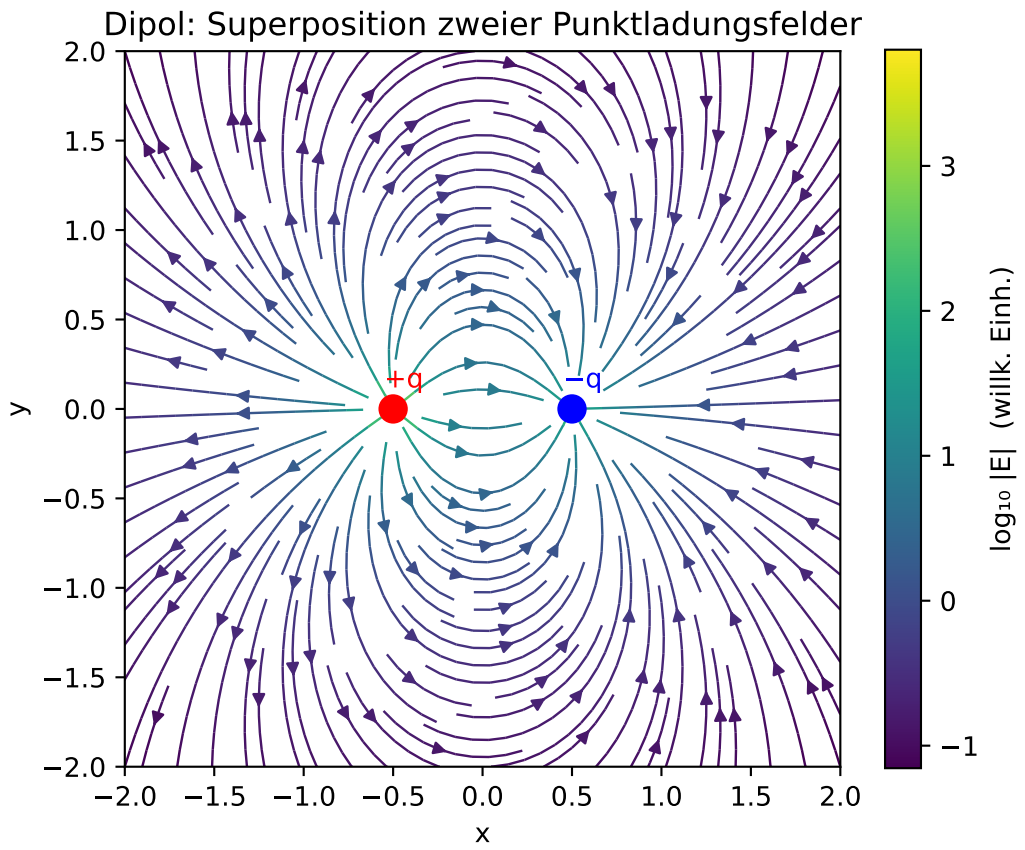


Abbildung 1.4: Der Dipol: Feldlinien laufen von +q (rot) zu −q (blau). Farbe = $\log_{10}|E|$. Auf der Mittelsenkrechten zeigt das Feld exakt in +x-Richtung — von der positiven zur negativen Seite.

`streamplot` zeichnet **Feldlinien**: Kurven, die an jedem Punkt der dortigen Pfeilrichtung folgen. Sie quellen aus $+q$, münden in $-q$, und auf der Mittelsenkrechten laufen sie waagerecht von links nach rechts — dort heben sich die y-Anteile beider Ladungen aus Symmetriegründen exakt auf. Das volle Kapitel-Programm `programme/kap01/kap01_dipolfeld.py` zeichnet beide Bilder und **belegt diese Symmetrie numerisch**: Es wertet das Feld auf der Mittelsenkrechten aus und zeigt, dass die y-Komponente auf Maschinengenauigkeit null ist. Führe es aus und lies den Beleg.

> ⚠ Naheliegende Vermutung
>
> **Vermutung:** *„Feldlinien existieren wirklich — zwischen zwei Linien ist kein Feld.“*

Warum sie naheliegt: Jede Abbildung zeigt einzelne, sauber getrennte Linien mit leerem Raum dazwischen. Eisenfeilspäne am Magneten ordnen sich sogar sichtbar zu Ketten — das sieht nach realen Fäden aus.
Was stattdessen stimmt: Das Feld füllt den Raum **lückenlos**; an *jedem* Punkt steht ein Pfeil. Feldlinien sind ein Zeichenwerkzeug: `streamplot` wählt willkürliche Startpunkte und folgt von dort den Pfeilen — mit dem Parameter `density` kannst du beliebig viele Linien dazwischenlegen, ohne dass sich physikalisch irgendetwas ändert. (Die Feilspäne-Ketten entstehen, weil magnetisierte Späne *einander* anziehen — der Magnet allein hätte sie nie zu Linien sortiert.)

Merkkasten

- Ein **Skalarfeld** ist eine Zahl an jedem Ort, ein **Vektorfeld** ein Pfeil an jedem Ort — numerisch: ein bzw. zwei (drei) Arrays auf einem Gitter.
- **Skalarprodukt** $\vec{a} \cdot \vec{b}$: eine Zahl — „wie viel von $\vec{a}$ zeigt entlang $\vec{b}$?“; null bei senkrecht.
- **Kreuzprodukt** $\vec{a} \times \vec{b}$: ein Pfeil — senkrecht auf beiden (Rechte-Hand-Regel), Länge = Parallelogrammfläche.
- Punktladung: $\vec{E} \propto \hat{r}/r^2$, radial nach außen. **Felder addieren sich** (Superposition) — der Dipol ist nur eine Summe.
- Feldlinien sind ein **Zeichenwerkzeug**; das Feld selbst füllt den Raum lückenlos.

Roter Faden

Alles Weitere im Buch steht auf diesem Kapitel: In **Kapitel 2** fragen wir, wie sich Felder von Ort zu Ort *ändern* — aus Nachbarpfeil minus Pfeil werden Ableitungen, Quellen und Wirbel. In **Kapitel 3** sagen die Maxwell-Gleichungen, *welche* Quellen und Wirbel die Felder $\vec{E}$ und $\vec{B}$ tatsächlich haben — das Skalarprodukt von heute misst dort den Fluss durch Flächen. Und das Kreuzprodukt trägt ab **Kapitel 18** als $\vec{S} = \vec{E} \times \vec{H}$ die Energie jeder Welle durch dieses Buch.

Übungen

Ü 1.1 (Verstehen). Das Skalarprodukt zweier Pfeile ist negativ. Was sagt das über den Winkel zwischen ihnen — und welche Bedeutung hat das Vorzeichen, wenn $\vec{a}$ eine Kraft und $\vec{b}$ ein Weg ist?

> **Musterlösung zu Ü 1.1**
>
> Negativ heißt $\cos\alpha < 0$, also $\alpha > 90°$ — die Pfeile zeigen „eher gegeneinander". Bei Kraft mal Weg ist das Skalarprodukt die verrichtete Arbeit: negativ bedeutet, die Kraft *bremst* die Bewegung, statt sie anzutreiben (z. B. Reibung).

Ü 1.2 (Verstehen). Im Punktladungsbild sind alle Pfeile gleich lang gezeichnet. Woran liest man die Feldstärke ab — und warum wäre das Bild mit „ehrlichen" Pfeillängen unbrauchbar?

> **Musterlösung zu Ü 1.2**
>
> Die Stärke steckt in der Farbe (logarithmisch). Mit echten Längen wäre der Pfeil direkt neben der Ladung wegen $1/r^2$ hunderte Male länger als am Rand — er würde das ganze Bild überdecken, während die äußeren Pfeile zu unsichtbaren Punkten schrumpfen.

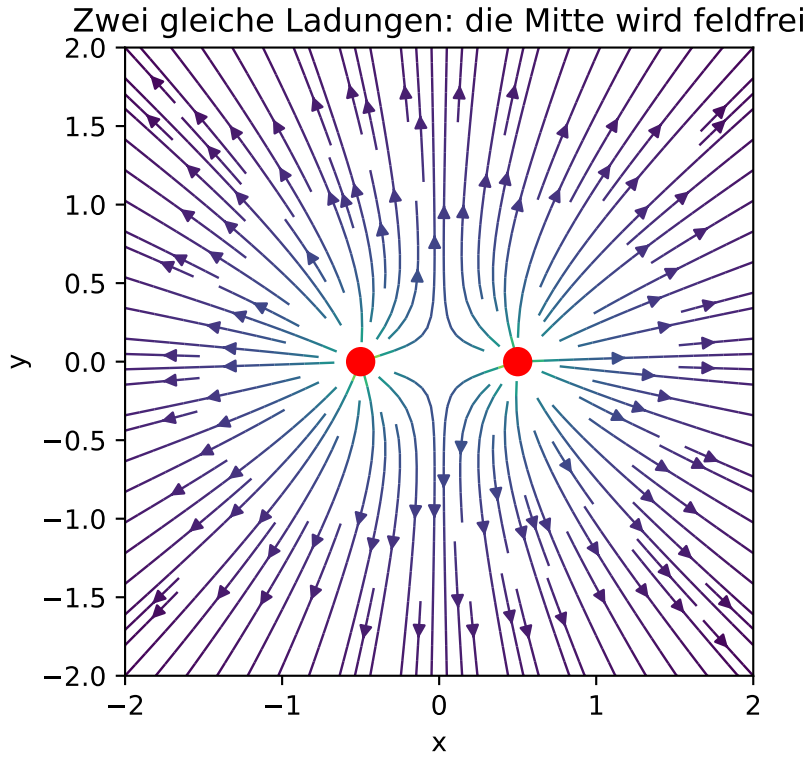
Ü 1.3 (Verändern). Ersetze im Kapitel-Programm die negative Ladung durch eine zweite *positive* ($-1.0 \to +1.0$). Sage vorher: Wie ändert sich das Linienbild, und was passiert mit dem Feld genau in der Mitte zwischen den Ladungen? Dann ausführen und prüfen.

> **Musterlösung zu Ü 1.3**
>
> Die Linien laufen jetzt aus *beiden* Ladungen heraus und weichen einander aus; die Mittelsenkrechte wird zur „Wand", die keine Linie kreuzt. Genau in der Mitte heben sich die beiden Felder vollständig auf: $\vec{E} = 0$ — ein Gleichgewichtspunkt. Die geänderte Zeile und das Ergebnis:

```
# von oben: punktladungsfeld(), x, y   (Dipol-Abschnitt)
ex1, ey1 = punktladungsfeld(+1.0, (-0.5, 0.0), x, y)
ex2, ey2 = punktladungsfeld(+1.0, (+0.5, 0.0), x, y)   # war: -1.0
ex, ey = ex1 + ex2, ey1 + ey2

fig, ax = plt.subplots(figsize=(5.0, 4.2))
ax.streamplot(x, y, ex, ey, color=np.log10(np.hypot(ex, ey)),
              cmap="viridis", density=1.4, linewidth=0.9)
ax.plot([-0.5, 0.5], [0, 0], "ro", markersize=10)
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("Zwei gleiche Ladungen: die Mitte wird feldfrei")
ax.set_aspect("equal"); plt.tight_layout(); plt.show()

e1 = punktladungsfeld(+1.0, (-0.5, 0.0), np.array(0.0), np.array(0.0))
e2 = punktladungsfeld(+1.0, (+0.5, 0.0), np.array(0.0), np.array(0.0))
print(f"E in der Mitte: ({e1[0]+e2[0]:.1f}, {e1[1]+e2[1]:.1f})")
```




```
E in der Mitte: (0.0, 0.0)
```

Der numerische Symmetrie-Beleg im Kapitel-Programm schlägt jetzt übrigens an, denn auf der Mittelsenkrechten verschwindet nicht mehr

nur die y-Komponente, sondern auch die x-Komponente — genau das zeigt der Mittelpunkts-Check.

Ü 1.4 (Übertragen). Baue aus dem Kapitel-Programm einen **Quadrupol**: vier Ladungen $+q, -q, +q, -q$ auf den Ecken eines Quadrats (abwechselnd). Zeichne das Linienbild. Wo ist das Feld null?

Musterlösung zu Ü 1.4

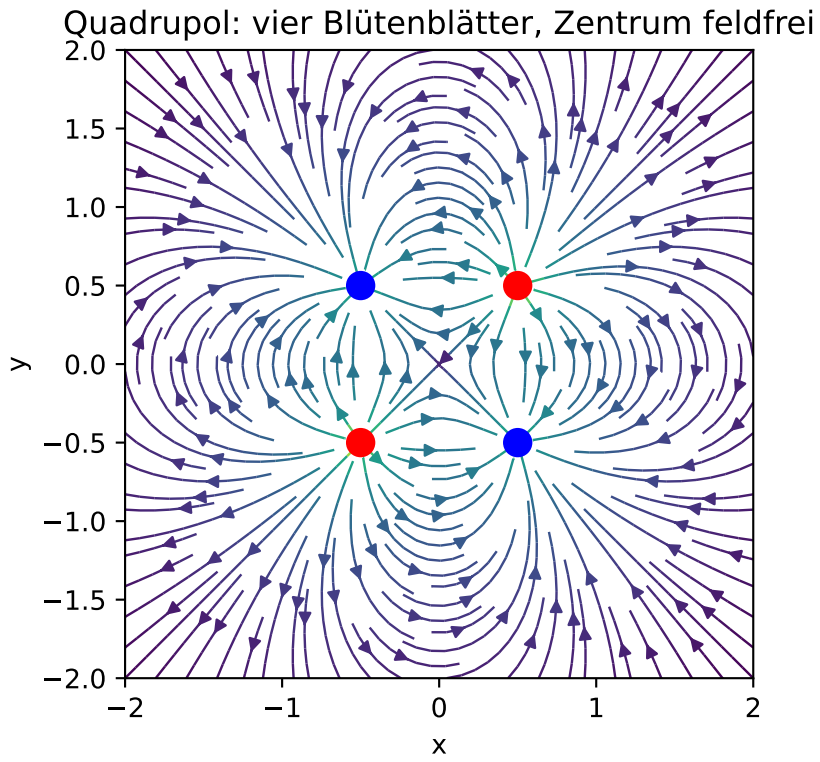
Vier Aufrufe von `punktladungsfeld` und drei Pluszeichen — mehr braucht es dank Superposition nicht:

```
# von oben: punktladungsfeld(), x, y   (Dipol-Abschnitt)
LADUNGEN = [(+1.0, (-0.5, -0.5)), (-1.0, (+0.5, -0.5)),
            (+1.0, (+0.5, +0.5)), (-1.0, (-0.5, +0.5))]

ex = sum(punktladungsfeld(q, ort, x, y)[0] for q, ort in LADUNGEN)
ey = sum(punktladungsfeld(q, ort, x, y)[1] for q, ort in LADUNGEN)

fig, ax = plt.subplots(figsize=(5.0, 4.2))
ax.streamplot(x, y, ex, ey, color=np.log10(np.hypot(ex, ey)),
              cmap="viridis", density=1.4, linewidth=0.9)
for q, (qx, qy) in LADUNGEN:
    ax.plot(qx, qy, "o", color="red" if q > 0 else "blue", markersize=10)
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("Quadrupol: vier Blütenblätter, Zentrum feldfrei")
ax.set_aspect("equal"); plt.tight_layout(); plt.show()

mitte = [punktladungsfeld(q, ort, np.array(0.0), np.array(0.0))
         for q, ort in LADUNGEN]
print(f"E im Zentrum: ({sum(m[0] for m in mitte):.1f},"
      f" {sum(m[1] for m in mitte):.1f})")
```




```
E im Zentrum: (0.0, 0.0)
```

Im Zentrum heben sich aus Symmetriegründen alle vier Beiträge auf: $\vec{E} = 0$. Zusatzbeobachtung für später: Weit weg fällt das Quadrupolfeld noch schneller ab als das Dipolfeld — Stichwort Multipolentwicklung.

Das Kleingedruckte

- **Willkürliche Einheiten:** Der Vorfaktor $q/(4\pi\varepsilon_0)$ skaliert nur alle Pfeile gemeinsam, deshalb durften wir ihn beim Zeichnen auf 1 setzen. Sobald *Zahlenwerte* zählen (Kräfte, Energien, ab Kapitel 3), rechnen wir in SI-Einheiten mit echtem ε_0.
- **2D ist ein Schnitt:** Unsere Bilder zeigen die Ebene $z = 0$ eines dreidimensionalen Feldes. Für die Punktladung mit $1/r^2$ ist das der ehrliche Schnitt durchs Zentrum; ein „echtes 2D-Universum" (Linienladung) hätte $1/r$.
- **Liniendichte mit Vorsicht:** In 3D-Skizzen kodiert die Dichte der Feldlinien die Feldstärke. `streamplot` in der Ebene tut das *nicht* zuverlässig

— darum tragen unsere Linien die Stärke als Farbe.

2 Ableitungen von Feldern

Stell dir vor, du stehst im dichten Nebel irgendwo an einem Berghang. Du willst zum Gipfel, siehst aber keine zehn Meter weit. Was du hast: einen Höhenmesser und deine Beine. Also machst du einen kleinen Schritt nach Osten und liest ab: 30 cm gewonnen. Zurück, ein Schritt nach Norden: 10 cm verloren. Jetzt weißt du alles: Bergauf geht es nach Osten, leicht südlich — und zwar *ohne* je die ganze Landschaft gesehen zu haben.

Genau das ist eine **Ableitung**: Werte an benachbarten Orten vergleichen. Die Mathematik macht die Schritte unendlich klein — der Computer kann das nicht, und das ist die beste Nachricht dieses Buches: Für uns bleibt die Ableitung das, was sie im Nebel war — **eine Differenz zweier Messwerte, geteilt durch die Schrittweite.** Aus diesem einen Handgriff baut dieses Kapitel alle drei „Ableitungen für Felder": Gradient, Divergenz und Rotation. Und in Kapitel 3 wirst du sehen: Die Maxwell-Gleichungen sind nichts als vier Sätze *über genau diese drei Größen*.

Lernziele

Nach diesem Kapitel kannst du …

1. … Ableitungen numerisch als Nachbar-Differenzen berechnen und am log-log-Plot ablesen, wie schnell der Fehler mit der Schrittweite schrumpft,
2. … den **Gradienten** eines Skalarfelds berechnen und deuten: Er zeigt bergauf und steht senkrecht auf den Höhenlinien,
3. … die **Divergenz** als Quellstärke deuten („fließt netto etwas aus dem Kästchen heraus?") und berechnen,
4. … die **Rotation** als Wirbelstärke deuten (Paddelrad-Test) und berechnen — auch dort, wo man keinen Wirbel *sieht*,
5. … die Sätze von **Gauß** und **Stokes** in je einem Satz erklären und numerisch verifizieren.

2.1 Die Ableitung als Differenz

Fangen wir eindimensional an. Die Steigung einer Funktion f am Punkt x schätzt man am fairsten **symmetrisch**: einen halben Schritt zurückblicken, einen halben nach vorn —

$$f'(x) \approx \frac{f(x + \frac{\Delta x}{2}) - f(x - \frac{\Delta x}{2})}{\Delta x}.$$

Das heißt **zentrale Differenz**. Wie gut ist sie? Wir testen sie an $f(x) = \sin(x)$, denn da kennen wir die Antwort exakt: $f'(x) = \cos(x)$.

```
import numpy as np
import matplotlib.pyplot as plt

x0 = 1.0
# np.logspace(-3, 0, 20): 20 logarithmisch verteilte Werte von 10⁻³
# bis 10⁰ — auf der log-Achse gleichmäßig dicht
schrittweiten = np.logspace(-3, 0, 20)
fehler = []
for dx in schrittweiten:
    geschaetzt = (np.sin(x0 + dx/2) - np.sin(x0 - dx/2)) / dx
    fehler.append(abs(geschaetzt - np.cos(x0)))  # Vergleich mit exakt

fig, ax = plt.subplots(figsize=(5.4, 3.6))
ax.loglog(schrittweiten, fehler, "o-", label="zentrale Differenz")
ax.loglog(schrittweiten, 0.04 * schrittweiten**2, "k--",
          label="Erwartung ~ Δx²")
ax.set_xlabel("Schrittweite Δx")
ax.set_ylabel("Fehler |geschätzt − exakt|")
ax.legend()
plt.tight_layout(); plt.show()
```

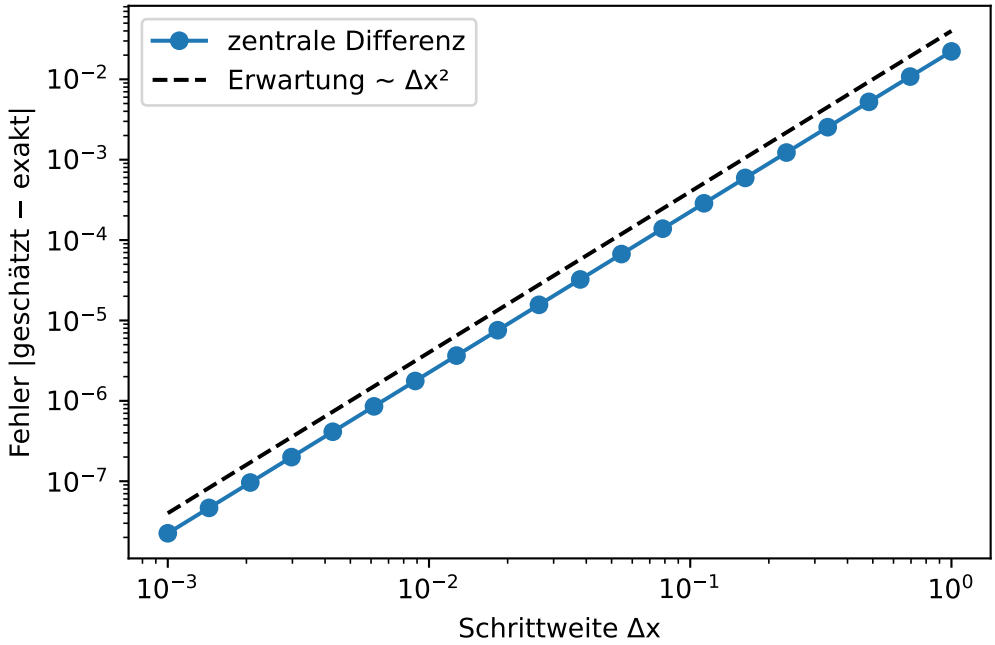


Abbildung 2.1: Der Fehler der zentralen Differenz an f = sin(x), ausgewertet bei x = 1. Gestrichelt die Erwartung ~Δx^2: Halbe Schrittweite ergibt ein Viertel des Fehlers.

Die Messpunkte folgen exakt der gestrichelten Linie: Der Fehler fällt **quadratisch** — Schrittweite halbieren, Fehler viertelt sich. Das ist der Grund, warum wir im ganzen Buch zentrale Differenzen benutzen (die einseitige Variante $\big(f(x+\Delta x)-f(x)\big)/\Delta x$ schafft nur einen Faktor 2). Merke dir dieses Bild: Es ist der **erste Konvergenztest** des Buches, und solche Tests sind unser Wahrheitskriterium — nicht „sieht plausibel aus", sondern „schrumpft mit der erwarteten Ordnung".

Für Felder brauchen wir Ableitungen *nach einer Richtung*: Das Symbol $\partial f/\partial x$ („partielle Ableitung") heißt nur: Steigung in x-Richtung, während y festgehalten wird. Numerisch ist das dieselbe Nachbar-Differenz wie eben — angewendet entlang einer Achse des Arrays. NumPy bringt sie fertig mit: `np.gradient(f, dx, axis=...)`, und im Inneren des Arrays rechnet diese Funktion **genau den zentralen Differenzenquotienten von eben**:

$$\left.\frac{\partial f}{\partial x}\right|_{\text{Punkt } i} \approx \frac{f_{i+1}-f_{i-1}}{2\,\Delta x} \qquad \text{(rechter Nachbar minus linker Nachbar,geteilt durch den doppelten}$$

Wann immer in diesem Buch `np.gradient` auftaucht, steht also nichts Geheimnisvolles dahinter — nur diese eine Zeile Nachbarschafts-Bilanz.

2.2 Der Gradient: bergauf zeigen

Packt man die beiden Steigungen eines Skalarfelds in einen Pfeil, entsteht der **Gradient**:

$$\nabla T = \left(\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}\right).$$

Das Dreieck ∇ („Nabla") liest du am besten als **Pfeil aus Ableitungen**. Der Gradient eines Skalarfelds ist also ein *Vektorfeld* — an jedem Ort der Pfeil, der die Nebel-Frage vom Kapitelanfang beantwortet: In welche Richtung geht es bergauf, und wie steil?

Vorhersage-Punkt

Bevor du weiterliest: Wir zeichnen gleich die Gradientpfeile in die Temperaturlandschaft aus Kapitel 1 (warme Insel links, kalte Senke rechts). In welchem Winkel kreuzen die Pfeile die Konturlinien — und zeigen sie zur warmen Insel hin oder von ihr weg? Lege dich fest.

```
x, y = np.meshgrid(np.linspace(-2, 2, 200), np.linspace(-2, 2, 200))
dx = x[0, 1] - x[0, 0]

# dieselbe Landschaft wie in Kapitel 1
T = 20 + 6*np.exp(-((x+1)**2 + y**2)) - 4*np.exp(-((x-1)**2 + (y-0.5)**2))

dT_dx = np.gradient(T, dx, axis=1)     # Steigung in x-Richtung
dT_dy = np.gradient(T, dx, axis=0)     # Steigung in y-Richtung

fig, ax = plt.subplots(figsize=(5.6, 4.4))
linien = ax.contour(x, y, T, levels=14, cmap="coolwarm")
ax.clabel(linien, fontsize=7, fmt="%.0f")
# slice(None, None, 14) ist der Indexausdruck ::14 als Objekt —
# x[s] greift also jeden 14. Gitterpunkt in beiden Richtungen heraus
s = (slice(None, None, 14), slice(None, None, 14))
ax.quiver(x[s], y[s], dT_dx[s], dT_dy[s], color="k", scale=60)
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("∇T: senkrecht auf den Höhenlinien, bergauf")
```

```
ax.set_aspect("equal")
plt.tight_layout(); plt.show()
```

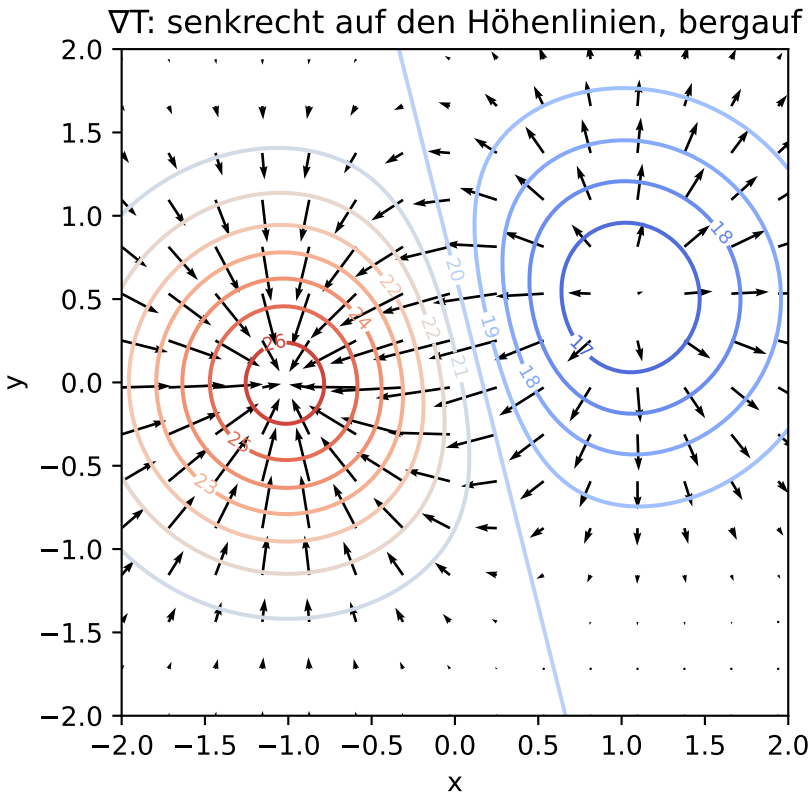


Abbildung 2.2: Der Gradient der Temperaturlandschaft aus Kapitel 1: Jeder Pfeil zeigt bergauf — zur warmen Insel hin, aus der kalten Senke heraus — und kreuzt die Höhenlinien im rechten Winkel.

Beides bestätigt: Die Pfeile stehen **senkrecht** auf den Konturlinien und zeigen **zur warmen Insel hin** — bergauf. Das Senkrecht-Stehen ist kein Zufall, sondern Logik: *Entlang* einer Höhenlinie ändert sich T per Definition nicht, also kann die Änderung nur *quer* dazu liegen. Und die Pfeile sind dort am längsten, wo die Linien am engsten stehen — am steilen Hang.

2.3 Die Divergenz: Quellen aufspüren

Der Gradient fragt ein *Skalarfeld* ab. Die nächsten beiden Werkzeuge fragen ein *Vektorfeld* ab — und zwar mit den zwei Produkten aus Kapitel 1, angewendet auf den Nabla-Pfeil. Das **Skalarprodukt** von ∇ mit dem Feld gibt eine Zahl an jedem Ort:

$$\operatorname{div} \vec{F} \;=\; \nabla \cdot \vec{F} \;=\; \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}.$$

Was misst diese Zahl? Denk dir ein winziges Kästchen um den Ort gelegt und mach Bilanz: **Fließt netto mehr hinaus als hinein?** Wenn ja, muss im Kästchen etwas entspringen — eine **Quelle** (Divergenz positiv). Fließt netto mehr hinein, verschluckt das Kästchen etwas — eine **Senke** (negativ). Geht genau so viel hinaus wie hinein, ist das Feld dort **quellenfrei** (null) — egal wie wild es aussieht.

Halte einen Punkt unbedingt fest, bevor wir weitergehen: **Hinein geht ein Vektorfeld — heraus kommt an jedem Ort eine einzige Zahl.** Das passt zum Skalarprodukt aus Kapitel 1 (zwei Pfeile rein, Zahl raus): $\nabla \cdot \vec{F}$ ist „Ableitungspfeil mal Feldpfeil". Das Ergebnis $\operatorname{div} \vec{F}$ ist deshalb selbst ein **Skalarfeld** — eine Zahl an jedem Ort, wie die Temperaturkarte aus Kapitel 1. Man kann es als Farbbild zeichnen, aber nicht mit Pfeilen.

2.3.1 Eine Divergenz komplett von Hand

Damit das nicht abstrakt bleibt, rechnen wir **einen einzigen Punkt vollständig durch** — mit nichts als den vier Nachbarwerten und dem zentralen Differenzenquotienten. Feld: die Quelle $\vec{F} = (u, v) = (x, y)$. Punkt: $(1{,}0,\ 0{,}5)$. Schrittweite: $h = 0{,}1$.

Für $\partial u/\partial x$ brauchen wir u beim **rechten und linken Nachbarn**, für $\partial v/\partial y$ brauchen wir v beim **oberen und unteren Nachbarn** — vier Zahlen insgesamt:

```
def u(x, y): return x          # x-Komponente des Quellenfelds
def v(x, y): return y          # y-Komponente

px, py, h = 1.0, 0.5, 0.1      # Auswertepunkt und Schrittweite

print(f"u rechts (bei x={px+h}) = {u(px+h, py):.2f}   "
      f"u links (bei x={px-h}) = {u(px-h, py):.2f}")
```

```
print(f"v oben   (bei y={py+h}) = {v(px, py+h):.2f}   "
      f"v unten (bei y={py-h}) = {v(px, py-h):.2f}\n")

du_dx = (u(px+h, py) - u(px-h, py)) / (2*h)
dv_dy = (v(px, py+h) - v(px, py-h)) / (2*h)
print(f"∂u/∂x ≈ (1.10 − 0.90) / 0.2 = {du_dx:.2f}")
print(f"∂v/∂y ≈ (0.60 − 0.40) / 0.2 = {dv_dy:.2f}")
print(f"div F am Punkt ({px}, {py}) = {du_dx:.2f} + {dv_dy:.2f} "
      f"= {du_dx + dv_dy:.2f}")
```

```
u rechts (bei x=1.1) = 1.10   u links (bei x=0.9) = 0.90
v oben   (bei y=0.6) = 0.60   v unten (bei y=0.4) = 0.40

∂u/∂x ≈ (1.10 − 0.90) / 0.2 = 1.00
∂v/∂y ≈ (0.60 − 0.40) / 0.2 = 1.00
div F am Punkt (1.0, 0.5) = 1.00 + 1.00 = 2.00
```

Vier Nachbarwerte angeschaut, zwei Differenzenquotienten gebildet, addiert — **eine Zahl: 2.** Mehr ist eine Divergenz nicht. Wiederhole die Rechnung im Kopf an irgendeinem anderen Punkt: Es kommt wieder 2 heraus (die Differenzen hängen hier nicht vom Ort ab) — das Skalarfeld div $\vec{F}$ ist für dieses Feld also überall konstant gleich 2. Und genau diese Rechnung führt `np.gradient` nachher für alle 40 000 Gitterpunkte gleichzeitig aus, nichts weiter.

2.4 Die Rotation: Wirbel aufspüren

Das **Kreuzprodukt** mit dem Nabla-Pfeil gibt die zweite Frage. In der Ebene bleibt davon eine Zahl übrig (die z-Komponente — der Pfeil, der aus dem Papier ragt):

$$\operatorname{rot}\vec{F}\Big|_z \;=\; (\nabla \times \vec{F})_z \;=\; \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}.$$

Die Anschauung: Halte ein winziges **Paddelrad** in die Strömung. Dreht es sich? Wenn ja, hat das Feld dort Rotation — positiv für Drehung gegen den Uhrzeigersinn. Das Paddelrad reagiert darauf, ob die Strömung auf seiner einen Seite *stärker schiebt* als auf der anderen.

Auch hier gilt: **Vektorfeld rein, an jedem Ort eine Zahl raus** — in der Ebene ist rot $\vec{F}$ ebenfalls ein Skalarfeld. (Im dreidimensionalen Raum wird

daraus ein Pfeil, passend zum Kreuzprodukt; in unserer Ebene bleibt davon nur die Komponente übrig, die aus dem Papier ragt — eine Zahl. Mehr dazu im Kleingedruckten.)

2.4.1 Eine Rotation komplett von Hand

Dieselbe Vier-Nachbarn-Rechnung, jetzt am **Wirbelfeld** $\vec{F} = (u, v) = (-y, x)$ — dem Karussell aus Kapitel 1 —, wieder am Punkt $(1{,}0,\ 0{,}5)$ mit $h = 0{,}1$. Beachte die Überkreuz-Logik der Formel: Für $\partial v/\partial x$ fragen wir die **y-Komponente** beim rechten/linken Nachbarn ab, für $\partial u/\partial y$ die **x-Komponente** beim oberen/unteren:

```
def u(x, y): return -y          # x-Komponente des Wirbelfelds
def v(x, y): return x           # y-Komponente

px, py, h = 1.0, 0.5, 0.1       # derselbe Punkt und Schritt wie eben

dv_dx = (v(px+h, py) - v(px-h, py)) / (2*h)
du_dy = (u(px, py+h) - u(px, py-h)) / (2*h)
print(f"v rechts = {v(px+h, py):+.2f}, v links = {v(px-h, py):+.2f}"
      f"   →  ∂v/∂x ≈ {dv_dx:+.2f}")
print(f"u oben   = {u(px, py+h):+.2f}, u unten = {u(px, py-h):+.2f}"
      f"   →  ∂u/∂y ≈ {du_dy:+.2f}")
print(f"rot F am Punkt ({px}, {py}) = {dv_dx:+.2f} − ({du_dy:+.2f}) "
      f"= {dv_dx - du_dy:+.2f}")
```

```
v rechts = +1.10, v links = +0.90   →  ∂v/∂x ≈ +1.00
u oben   = -0.60, u unten = -0.40   →  ∂u/∂y ≈ -1.00
rot F am Punkt (1.0, 0.5) = +1.00 − (-1.00) = +2.00
```

Wieder kommt **eine einzige Zahl** heraus: $+2$, Drehung gegen den Uhrzeigersinn — das Paddelrad im Karussell dreht sich, wie erwartet. Sieh dir an, *woraus* die 2 entsteht: Beide Differenzen tragen $+1$ bei — die y-Komponente wächst nach rechts ($\partial v/\partial x = +1$), **und** die x-Komponente nimmt nach oben ab ($\partial u/\partial y = -1$, mit dem Minus der Formel also auch $+1$). Die Formel vergleicht stur Nachbarwerte; ob das Feld dabei „rund aussieht", ist ihr gleichgültig — das wird gleich noch wichtig.

Vorhersage-Punkt

Bevor du weiterliest: Drei Felder treten an: die **Quelle** (x, y) (alle Pfeile vom Zentrum weg), der **Wirbel** $(-y, x)$ (das Karussell aus Kapitel 1) und die **Scherung** $(y, 0)$ (alle Pfeile waagerecht, oben nach rechts, unten nach links — wie eine Flussströmung, die am Ufer langsamer ist). Sage für jedes der drei Felder voraus: Divergenz — ja oder nein? Rotation — ja oder nein? Sechs Antworten, dann weiterlesen.

```
# von oben: x, y, dx — das 200er-Gitter aus dem Gradient-Abschnitt
def d_dx(f): return np.gradient(f, dx, axis=1)
def d_dy(f): return np.gradient(f, dx, axis=0)
def divergenz(u, v): return d_dx(u) + d_dy(v)
def rotation(u, v):  return d_dx(v) - d_dy(u)

# np.zeros_like(x): ein Array voller Nullen in der Form von x
felder = [("Quelle (x, y)", x, y),
          ("Wirbel (-y, x)", -y, x),
          ("Scherung (y, 0)", y, np.zeros_like(x))]

fig, achsen = plt.subplots(2, 3, figsize=(9.0, 6.2))
s = (slice(None, None, 20), slice(None, None, 20))
for spalte, (name, u, v) in enumerate(felder):
    for zeile, (wert, art) in enumerate([(divergenz(u, v), "div"),
                                         (rotation(u, v), "rot")]):
        ax = achsen[zeile, spalte]
        bild = ax.imshow(wert, extent=(-2, 2, -2, 2), origin="lower",
                         cmap="RdBu_r", vmin=-2.2, vmax=2.2)
        ax.quiver(x[s], y[s], u[s], v[s], color="k", scale=30, width=0.005)
        ax.set_title(f"{name}: {art} = {wert[130, 130]:+.1f}", fontsize=9)
        ax.set_xticks([]); ax.set_yticks([])
fig.colorbar(bild, ax=list(achsen.flat), shrink=0.75,
             label="div (oben) bzw. rot (unten)")
plt.show()
```

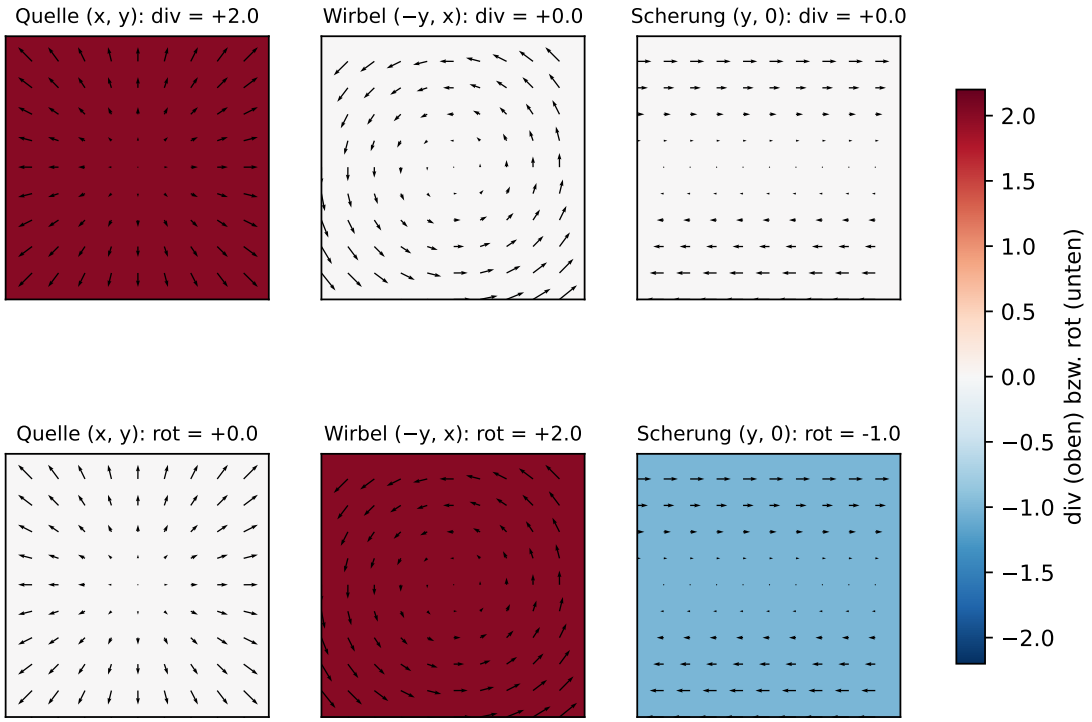


Abbildung 2.3: Drei Felder unter dem Quellen-Mikroskop (oben, Divergenz als Farbe) und dem Wirbel-Mikroskop (unten, Rotation als Farbe). Die Quelle hat nur Divergenz, der Wirbel nur Rotation — und die Scherung hat Rotation, obwohl keine einzige Feldlinie gekrümmt ist.

Eine Lesehilfe, denn in diesen Bildern stecken **zwei Felder übereinander**: Die schwarzen **Pfeile** zeigen die *Eingabe* — das Vektorfeld $\vec{F}$, in jeder Spalte oben und unten dasselbe. Die **Farbe** dahinter zeigt die *Ausgabe* — das Skalarfeld div $\vec{F}$ (obere Zeile) bzw. rot $\vec{F}$ (untere Zeile), also an jedem Ort genau die eine Zahl aus den Handrechnungen von eben, dargestellt wie eine Temperaturkarte: rot positiv, blau negativ, weiß null. Kurz: **Die Pfeile sind die Frage, die Farbe ist die Antwort.**

Und die Antworten: Die Quelle hat Divergenz +2 (genau die Zahl aus unserer Handrechnung — die Farbe bestätigt, dass sie überall gilt) und keine Rotation; der Wirbel hat Rotation +2 und keine Divergenz — so weit, so erwartbar. Die Überraschung ist die **Scherung**: schnurgerade Feldlinien, und trotzdem rot $= -1$. Halte das Paddelrad hinein: Oben schiebt die Strömung nach rechts, unten nach links — es dreht sich (im Uhrzeigersinn, daher negativ), obwohl nirgends etwas „im Kreis fließt". **Rotation sieht man nicht an der Form der Feldlinien, sondern am Quergefälle der Geschwindig-**

keit. Wenn deine sechste Antwort falsch war, bist du in bester Gesellschaft — und um eine wichtige Erfahrung reicher.

⚠ Naheliegende Vermutung

Vermutung: *„Das Coulomb-Feld einer Punktladung hat überall positive Divergenz — die Pfeile fächern doch sichtbar auseinander."*
Warum sie naheliegt: Auseinanderlaufende Pfeile sehen exakt so aus wie das Quellenfeld (x, y) oben links, und das *hat* überall Divergenz.
Was stattdessen stimmt: Die Pfeile fächern auf, ja — aber sie werden mit dem Abstand auch **kürzer**, und zwar genau im richtigen Maß: Die Kästchen-Bilanz geht überall auf null auf, **außer am Ort der Ladung selbst**. Das prüfen wir sofort nach:

```
# von oben: x, y, divergenz()   (Galerie-Abschnitt)
r2 = np.maximum(x**2 + y**2, 1e-12)
ex, ey = x / r2, y / r2          # Coulomb-Feld in der Ebene: r̂/r

div_E = divergenz(ex, ey)
aussen = r2 > 0.5**2             # alles außerhalb von r = 0,5
print(f"|div E| außerhalb der Ladung: höchstens "
      f"{np.max(np.abs(div_E[aussen])):.2e}")
print(f"div E direkt an der Ladung:   {np.max(div_E):.2e}")
```

```
|div E| außerhalb der Ladung: höchstens 1.17e-02
div E direkt an der Ladung:   3.96e+03
```

Fünf Größenordnungen Unterschied: Außerhalb der Ladung ist die Divergenz numerisch null (der winzige Rest ist Diskretisierungsfehler und schrumpft mit feinerem Gitter) — nur an der Ladung selbst explodiert sie. **Quellen des elektrischen Feldes sind allein die Ladungen.** Diesen Satz wirst du in Kapitel 3 wiedersehen: Er ist die erste Maxwell-Gleichung.

2.5 Zwei Sätze, die Buchhaltung machen

Divergenz und Rotation sind *lokale* Größen — ein Wert pro winzigem Kästchen. Zwei berühmte Sätze sagen, was passiert, wenn man viele Kästchen zusammenfasst. Der **Satz von Gauß**:

$$\oint_{\text{Rand}} \vec{F} \cdot \vec{n} \, \mathrm{d}s \;=\; \iint_{\text{Fläche}} \nabla \cdot \vec{F} \, \mathrm{d}A.$$

Links steht der **Fluss**: Wir laufen den Rand einer Fläche ab und summieren, wie viel Feld durch ihn *hindurch* zeigt — das misst das Skalarprodukt mit dem nach außen zeigenden Normalenpfeil $\vec{n}$ (Kapitel 1 lässt grüßen). Rechts steht die Summe aller Quellstärken im Inneren. Der Satz sagt schlicht: **Was netto herauskommt, muss drinnen entspringen.** Bei aneinandergrenzenden Kästchen hebt sich alles Innere weg — was Kästchen A nach rechts verlässt, betritt Kästchen B von links —, übrig bleibt der äußere Rand.

Der **Satz von Stokes** ist derselbe Gedanke für Wirbel: Die **Zirkulation** (wir laufen den Rand *entlang* und summieren, wie viel Feld mit uns mitläuft) ist gleich der Summe aller Wirbelstärken auf der Fläche:

$$\oint_{\text{Rand}} \vec{F} \cdot \mathrm{d}\vec{l} \;=\; \iint_{\text{Fläche}} (\nabla \times \vec{F})_z \, \mathrm{d}A.$$

Glauben müssen wir beides nicht — wir können es nachrechnen. Das Testfeld $\vec{F} = (x^2 y,\; x - y^3)$ hat ortsabhängige Divergenz *und* Rotation, ist also ein ehrlicher Prüfstein:

```
# von oben: x, divergenz(), rotation()   (Galerie-Abschnitt)
U, V = x**2 * y, x - y**3
i = slice(50, 151)              # die Box [-1, 1]² im Gitterausschnitt
rand = x[0, i]                  # Koordinaten entlang einer Boxkante

def flaechenintegral(feld):
    # np.trapezoid(werte, x): numerisches Integral per Trapezregel über
    # Abtastpunkte; zweimal angewendet – erst jede Zeile (axis=1),
    # dann das Ergebnis die Spalte hinunter
    return np.trapezoid(np.trapezoid(feld[i, i], rand, axis=1), rand)

fluss = (np.trapezoid(U[i, 150], rand) - np.trapezoid(U[i, 50], rand)
         + np.trapezoid(V[150, i], rand) - np.trapezoid(V[50, i], rand))
zirkulation = (np.trapezoid(U[50, i], rand) + np.trapezoid(V[i, 150], rand)
               - np.trapezoid(U[150, i], rand) - np.trapezoid(V[i, 50], rand))

print(f"Gauß  :  Fluss        = {fluss:+.5f}   "
      f"Quellsumme  = {flaechenintegral(divergenz(U, V)):+.5f}")
print(f"Stokes:  Zirkulation  = {zirkulation:+.5f}   "
      f"Wirbelsumme = {flaechenintegral(rotation(U, V)):+.5f}")
```

```
Gauß   :  Fluss        = -4.08142   Quellsumme  = -4.08387
Stokes:  Zirkulation  = +2.67929   Wirbelsumme = +2.67929
```

Beide Paare stimmen auf besser als ein Promille überein — der Rest ist Diskretisierungsfehler: verkleinere Δx, und er schrumpft mit der Ordnung aus Abbildung 2.1. Das **Kapitel-Programm** `programme/kap02/kap02_quellen_und_wirbel.py` enthält Galerie und beide Verifikationen in voller Länge, mit feinerem Gitter und `assert`-Schranken — führe es aus und vergleiche mit den analytischen Werten -4 und $8/3$.

Merkkasten

- Numerisch ist jede Ableitung eine **Nachbar-Differenz**; die zentrale Differenz hat Fehler $\sim \Delta x^2$ — und Konvergenztests sind unser Wahrheitskriterium.
- **Gradient** ∇T: Pfeil bergauf, senkrecht auf Höhenlinien (Skalarfeld → Vektorfeld).
- **Divergenz** $\nabla \cdot \vec{F}$: Quellstärke der Kästchen-Bilanz; das Coulomb-Feld ist überall quellenfrei **außer an der Ladung**.
- **Rotation** $\nabla \times \vec{F}$: Paddelrad-Test; sie steckt im Quergefälle, nicht in der Krümmung der Feldlinien.
- **Gauß**: Fluss durch den Rand = Quellen im Inneren. **Stokes**: Zirkulation um den Rand = Wirbel auf der Fläche.

Roter Faden

Die beiden Produkte aus **Kapitel 1** sind hier zu Operatoren geworden: Skalarprodukt mit ∇ → Divergenz, Kreuzprodukt mit ∇ → Rotation; und die Höhenlinien von dort haben jetzt ihren Gegenspieler, den Gradienten. In **Kapitel 3** werden die Maxwell-Gleichungen genau vier Aussagen über $\nabla\cdot$ und $\nabla\times$ von $\vec{E}$ und $\vec{B}$ sein — die Refutation oben war schon die erste. Die Nachbar-Differenzen auf dem Gitter sind ab **Kapitel 5** der Motor des FDTD-Verfahrens, und die Fehlerordnung aus Abbildung 2.1 kehrt dort als Konvergenztest wieder.

Übungen

Ü 2.1 (Verstehen). Das Feld $(x, -y)$ presst von oben und unten zusammen und zieht nach links und rechts auseinander. Divergenz? Rotation? Erst mit

der Kästchen-Bilanz und dem Paddelrad argumentieren, dann mit den Formeln nachrechnen.

Musterlösung zu Ü 2.1

Beides null. Bilanz: Was das Kästchen links/rechts verlässt, kommt oben/unten exakt nach ($\partial u/\partial x = 1$, $\partial v/\partial y = -1$, Summe 0). Paddelrad: Die Strömung ist spiegelsymmetrisch zu beiden Achsen durchs Rad — kein Drehmoment ($\partial v/\partial x = \partial u/\partial y = 0$). Die Nachrechnung auf dem Kapitel-Gitter:

```
# von oben: x, y, divergenz(), rotation()   (Galerie-Abschnitt)
u_press, v_press = x, -y                    # das Feld (x, -y)
print(f"div, größter Betrag: {np.max(np.abs(divergenz(u_press, v_press))):.1e}")
print(f"rot, größter Betrag: {np.max(np.abs(rotation(u_press, v_press))):.1e}")
```

```
div, größter Betrag: 1.1e-14
rot, größter Betrag: 0.0e+00
```

Ein Feld kann also kräftig *verformen*, ohne Quelle oder Wirbel zu sein.

Ü 2.2 (Verstehen). Warum steht der Gradient senkrecht auf den Höhenlinien? Argumentiere ohne Formel, nur mit der Bedeutung der beiden Begriffe.

Musterlösung zu Ü 2.2

Entlang einer Höhenlinie ändert sich der Feldwert nicht — in dieser Richtung ist die Steigung null. Der Gradient zeigt in die Richtung der *größten* Steigung; hätte er eine Komponente entlang der Höhenlinie, würde diese Komponente nichts zur Änderung beitragen und nur Länge verschwenden. Die steilste Richtung ist also die ganz ohne Entlang-Anteil: senkrecht zur Linie.

Ü 2.3 (Verändern). Verschiebe im Kapitel-Programm die Gauß-Box von $[-1, 1]^2$ auf $[0{,}2,\ 1{,}8]^2$ (Indizes anpassen!). Sage vorher, ob Fluss und Quellsumme weiterhin übereinstimmen — und ob sich ihr *Wert* ändert. Dann ausführen.

Musterlösung zu Ü 2.3

Sie stimmen weiterhin überein — der Satz von Gauß gilt für *jede* Box. Der Wert ändert sich aber: Die Divergenz $2xy - 3y^2$ ist ortsabhängig,

eine andere Box fängt andere Quellstärke ein. Auf dem 401er-Gitter des Kapitel-Programms entspricht $[0{,}2,\ 1{,}8]$ den Indizes 220 bis 380:

```
achse4 = np.linspace(-2, 2, 401)
h4 = achse4[1] - achse4[0]
X4, Y4 = np.meshgrid(achse4, achse4)
U4, V4 = X4**2 * Y4, X4 - Y4**3

box = slice(220, 381)                    # x, y ∈ [0,2 ; 1,8]
rand4 = achse4[box]
div4 = np.gradient(U4, h4, axis=1) + np.gradient(V4, h4, axis=0)

quellsumme = np.trapezoid(np.trapezoid(div4[box, box], rand4, axis=1), rand4)
fluss = (np.trapezoid(U4[box, 380], rand4) - np.trapezoid(U4[box, 220], rand4)
         + np.trapezoid(V4[380, box], rand4) - np.trapezoid(V4[220, box], rand4))
print(f"Fluss durch den Rand  : {fluss:+.4f}")
print(f"Divergenz aufsummiert : {quellsumme:+.4f}")
```

```
Fluss durch den Rand  : -4.1984
Divergenz aufsummiert : -4.1988
```

Beide Werte stimmen weiter überein, sind aber andere als bei der zentrierten Box — und negativ, da $-3y^2$ in der oberen Box dominiert.

Ü 2.4 (Übertragen). Untersuche den **Strudel** $\vec{F} = (-y, x)/(x^2 + y^2)$: Berechne seine Rotation numerisch (Vorsicht am Ursprung — schneide ihn aus) und dann die Zirkulation auf einem Kreis oder einer Box um den Ursprung. Du wirst etwas scheinbar Widersprüchliches finden. Wie passt es mit dem Satz von Stokes zusammen?

ℹ Musterlösung zu Ü 2.4

Die Rotation ist **überall null** (außer am Ursprung), die Zirkulation um den Ursprung trotzdem 2π:

```
# von oben: x, y, rotation()   (Galerie-Abschnitt)
r2_st = np.maximum(x**2 + y**2, 1e-12)
u_st, v_st = -y/r2_st, x/r2_st             # der Strudel

rot_st = rotation(u_st, v_st)
aussen = r2_st > 0.5**2                    # Ursprung ausgeschnitten
print(f"max |rot| außerhalb r = 0,5 : {np.max(np.abs(rot_st[aussen])):.1e}")

t = np.linspace(0, 2*np.pi, 2000)          # Kreis mit Radius 1
px_, py_ = np.cos(t), np.sin(t)
ub, vb = -py_/(px_**2 + py_**2), px_/(px_**2 + py_**2)
zirk = np.trapezoid(-ub*np.sin(t) + vb*np.cos(t), t)
print(f"Zirkulation um den Ursprung : {zirk:.6f}   (2π = {2*np.pi:.6f})")
```

```
max |rot| außerhalb r = 0,5 : 1.2e-02
Zirkulation um den Ursprung : 6.283185   (2π = 6.283185)
```

Scheinbar ein Widerspruch zu Stokes. Die Auflösung: Die Fläche im Satz von Stokes enthält den Ursprung, und genau dort ist das Feld singulär — die gesamte Wirbelstärke sitzt als unendlich scharfe Spitze in diesem einen Punkt (wie die Divergenz der Punktladung in der Refutation). Jede Schleife, die den Ursprung *nicht* umschließt, hat Zirkulation null. Merk dir dieses Feld: Es ist das Magnetfeld um einen stromdurchflossenen Draht, und die „Zirkulation = eingeschlossene Stärke”-Regel ist das Ampèresche Gesetz aus Kapitel 3.

Das Kleingedruckte

- **Ränder:** `np.gradient` benutzt am Arrayrand einseitige Differenzen (Fehler $\sim \Delta x$ statt Δx^2). Für unsere Bilder egal — für Präzisionsvergleiche legt man die Auswertebox wie oben ins Innere.
- **Singularitäten:** „Divergenz null außer an der Ladung” heißt mathematisch: Die Quelldichte ist eine Delta-Distribution. Numerisch zeigt sie sich als ein einzelner explodierender Gitterwert, dessen Höhe von Δx abhängt — der *integrierte* Fluss durch eine umschließende Box ist dagegen gitterunabhängig sinnvoll.
- **2D vs. 3D:** In drei Dimensionen ist die Rotation ein voller Vektor mit drei Komponenten; unsere Ebene zeigt nur seine z-Komponente. Divergenz und beide Integralsätze übertragen sich wörtlich (aus der Randkurve wird eine Hüllfläche, aus der Fläche ein Volumen).

3 Die Maxwell-Gleichungen, Term für Term

Alles Elektrische und Magnetische auf der Welt — der Blitz und der Kompass, das Handynetz und das Licht dieser Seite — gehorcht **vier Sätzen, die auf eine Postkarte passen.** James Clerk Maxwell schrieb sie 1865 zusammen, und ihm fiel dabei etwas auf, das bis heute zu den schönsten Überraschungen der Physik gehört: Aus zwei unscheinbaren Konstanten, die man im Labor an Kondensatoren und Spulen misst, fällt eine Geschwindigkeit heraus. *Die* Geschwindigkeit. Am Ende dieses Kapitels rechnest du sie selbst aus — und prüfst nach, dass die Gleichungen jedem Lichtpuls genau dieses eine Tempo *erzwingen*.

Das Beste: Du hast schon alles in der Hand. Die vier Sätze sind genau **zwei Quellen-Aussagen und zwei Wirbel-Aussagen** — und Quellen ($\nabla\cdot$) und Wirbel ($\nabla\times$) sind seit Kapitel 2 deine Werkzeuge.

Lernziele

Nach diesem Kapitel kannst du …

1. … die vier Maxwell-Gleichungen in je einem deutschen Satz wiedergeben und jeden Term benennen,
2. … das Gauß-Gesetz numerisch prüfen: Der Fluss durch eine Hülle ist q/ε_0 — egal welche Form und Größe die Hülle hat,
3. … erklären, warum $\nabla \cdot \vec{B} = 0$ bedeutet, dass Magnetfeldlinien weder Anfang noch Ende haben — und warum ein zersägter Magnet keine einzelnen Pole liefert,
4. … Faraday- und Ampère-Gesetz als Wirbel-Aussagen deuten und beide numerisch verifizieren — inklusive des Verschiebungsstroms,
5. … nachvollziehen, wie aus den zwei Wirbelgleichungen eine Wellengleichung wird, und $c = 1/\sqrt{\varepsilon_0\mu_0}$ selbst ausrechnen.

3.1 Die vier Gleichungen auf einen Blick

Im Vakuum, mit Ladungen und Strömen als Akteuren:

(1)	$\nabla \cdot \vec{E} = \dfrac{\rho}{\varepsilon_0}$	Ladungen sind die Quellen des E-Feldes.
(2)	$\nabla \cdot \vec{B} = 0$	Es gibt keine magnetischen Ladungen.
(3)	$\nabla \times \vec{E} = -\dfrac{\partial \vec{B}}{\partial t}$	Ein sich änderndes B wirbelt E.
(4)	$\nabla \times \vec{B} = \mu_0 \vec{J} + \mu_0 \varepsilon_0 \dfrac{\partial \vec{E}}{\partial t}$	Ströme — und ein sich änderndes E — wirbeln B.

Lies die Bausteine wie Vokabeln: ρ ist die **Ladungsdichte** (Coulomb pro Kubikmeter — wo wie viel Ladung sitzt), $\vec{J}$ die **Stromdichte** (Ampere pro Quadratmeter — wo wie viel Strom fließt). ε_0 und μ_0 sind die zwei **Feldkonstanten** aus der Einheitentabelle im Vorwort; sie legen fest, wie stark eine gegebene Ladung bzw. ein gegebener Strom den Raum mit Feld füllt. Und $\partial/\partial t$ ist die Zeit-Version unserer Nachbar-Differenz: Feld jetzt minus Feld eben, geteilt durch den Zeitschritt.

Zwei Dinge fallen sofort auf. Erstens die Struktur: Gleichung (1) und (2) sagen, **wo Felder entspringen** — Gleichung (3) und (4) sagen, **was Felder umrührt**. Zweitens die gebrochene Symmetrie: Bei $\vec{E}$ steht eine Quelldichte, bei $\vec{B}$ eine Null. Diese Null ist eine der härtesten experimentellen Aussagen der Physik, und wir schauen sie uns gleich genauer an. Jetzt aber: jede Gleichung einzeln auf den Prüfstand.

3.2 Gleichung 1: Ladungen sind die Quellen des E-Feldes

Die punktweise Form kennst du wörtlich aus der Refutation von Kapitel 2: Das Coulomb-Feld ist überall quellenfrei, *außer wo die Ladung sitzt* — genau das sagt $\nabla \cdot \vec{E} = \rho/\varepsilon_0$. Mit dem Satz von Gauß aus Kapitel 2 wird daraus eine Aussage über **Hüllen**: Der Fluss des E-Feldes durch eine geschlossene Hülle ist die Summe der eingeschlossenen Quellstärke,

$$\oint_{\text{Hülle}} \vec{E} \cdot \vec{n}\, \mathrm{d}A \;=\; \frac{q_{\text{innen}}}{\varepsilon_0}.$$

Das ist eine erstaunlich dreiste Behauptung: Die Hülle darf **beliebig groß und beliebig schief** sein, die Ladung darf irgendwo darin sitzen — der Fluss ist immer derselbe Wert q/ε_0. Und sitzt die Ladung außerhalb, ist er exakt null.

> Vorhersage-Punkt
>
> **Bevor du weiterliest:** Wir verdoppeln gleich den Radius der Hülle um eine Ladung. Die Hüllfläche vervierfacht sich — was macht der Fluss? Und warum? (Tipp: Was macht das Feld bei doppeltem Abstand?)

Erstmals rechnen wir in **echten SI-Einheiten**: eine Ladung von $q = 1\,\mathrm{nC}$ (Nanocoulomb — etwa die Reibungsladung eines Luftballons), also Soll-Fluss $q/\varepsilon_0 \approx 112{,}94$ V·m. Wir tasten die Kugeloberfläche mit einem Gitter aus Punkten ab und summieren $\vec{E} \cdot \vec{n}\, \mathrm{d}A$ — das Skalarprodukt aus Kapitel 1 fragt an jedem Punkt: „Wie viel Feld zeigt durch die Hülle *hindurch*?"

```
import numpy as np
from scipy.constants import epsilon_0

Q = 1e-9                                        # 1 nC
print(f"Soll: q/ε₀ = {Q/epsilon_0:.4f} V·m\n")

def fluss_durch_kugel(radius, ladungsort, n=400):
    """ E·n dA über eine Kugel um den Ursprung (abgetastet in θ, φ)."""
    theta, phi = np.linspace(0, np.pi, n), np.linspace(0, 2*np.pi, 2*n)
    # indexing="ij": Achse 0 = θ, Achse 1 = φ — meshgrid würde die
    # beiden sonst in Bild-Konvention (Zeile = y) vertauschen
    TH, PH = np.meshgrid(theta, phi, indexing="ij")
    px = radius*np.sin(TH)*np.cos(PH)          # Punkte auf der Kugel
    py = radius*np.sin(TH)*np.sin(PH)
    pz = radius*np.cos(TH)
    dx, dy, dz = px-ladungsort[0], py-ladungsort[1], pz-ladungsort[2]
    r = np.sqrt(dx**2 + dy**2 + dz**2)
    vor = Q/(4*np.pi*epsilon_0)/r**3           # Coulomb: E = q r̂ /(4πε₀ r²)
    # E·n mal Flächenelement dA = r² sinθ dθ dφ
    integrand = vor*(dx*px + dy*py + dz*pz)/radius * radius**2*np.sin(TH)
    return np.trapezoid(np.trapezoid(integrand, phi, axis=1), theta)

for name, radius, ort in [
```

```
        ("Kugel r = 1 m, Ladung mittig      ", 1.0, (0, 0, 0)),
        ("Kugel r = 1 m, Ladung außermittig ", 1.0, (0.3, 0.2, -0.4)),
        ("Kugel r = 2 m, Ladung außermittig ", 2.0, (0.3, 0.2, -0.4)),
        ("Kugel r = 1 m, Ladung außerhalb!  ", 1.0, (1.5, 0, 0))]:
    print(f"{name}: {fluss_durch_kugel(radius, ort):9.4f} V·m")
```

```
Soll: q/ε₀ = 112.9409 V·m

Kugel r = 1 m, Ladung mittig      :  112.9403 V·m
Kugel r = 1 m, Ladung außermittig :  112.9403 V·m
Kugel r = 2 m, Ladung außermittig :  112.9403 V·m
Kugel r = 1 m, Ladung außerhalb!  :   -0.0001 V·m
```

Viermal dieselbe Frage, dreimal exakt dieselbe Antwort — und einmal null. Die Auflösung der Vorhersage: Bei doppeltem Radius hat die Hülle die vierfache Fläche, aber das Feld ist wegen $1/r^2$ auf ein Viertel gefallen — **das Produkt bleibt.** Das Gauß-Gesetz ist genau diese Buchhaltung in ihrer allgemeinsten Form: Was die Quelle hergibt, muss durch jede Hülle hindurch, die sie umschließt. Das Kapitel-Programm prüft zusätzlich einen **Würfel** — gleiche Ladung, gleicher Fluss, denn auf die Form kommt es nicht an.

3.3 Gleichung 2: Es gibt keine magnetischen Ladungen

Dieselbe Buchhaltung für das Magnetfeld ergibt — null. Immer. Durch jede geschlossene Hülle der Welt tritt netto kein Magnetfeld: $\nabla \cdot \vec{B} = 0$, das Feld $\vec{B}$ ist überall quellenfrei. Niemand hat je ein Teilchen gefunden, aus dem Magnetfeldlinien *entspringen*, so wie das elektrische Feld aus einem Elektron entspringt.

> ⚠ Naheliegende Vermutung
>
> **Vermutung:** *„Doch, gibt es: Der Nordpol eines Magneten ist die Quelle — dort kommen die Feldlinien heraus —, der Südpol ist die Senke."*
> **Warum sie naheliegt:** Von außen betrachtet sieht ein Stabmagnet *exakt* so aus wie das Ladungspaar aus Kapitel 1: Linien quellen am Nordpol, münden am Südpol. Eisenfeilspäne zeichnen genau dieses Bild.
> **Was stattdessen stimmt:** Die Linien hören am Pol nicht auf — sie laufen **im Inneren des Magneten weiter**, vom Süd- zurück zum Nordpol. Jede Magnetfeldlinie ist eine **geschlossene Schleife** ohne Anfang und

> Ende; die „Pole” sind nur die Stellen, wo die Schleifen das Material verlassen. Deshalb scheitert auch jeder Versuch, einen Pol zu isolieren: Zersägst du den Magneten, bekommst du keine getrennten Quell- und Senken-Hälften, sondern **zwei vollständige Magnete** mit je zwei Polen — bis hinunter zum letzten Atom.

Den Unterschied kann man *sehen*. Wir stellen das elektrische Zwei-Ladungs-Feld aus Kapitel 1 neben sein magnetisches Gegenstück: zwei gegenläufige Ströme (eine Leiterschleife, von der Seite betrachtet — Strom kommt links aus dem Bild heraus und geht rechts hinein):

```
import matplotlib.pyplot as plt

x, y = np.meshgrid(np.linspace(-2, 2, 300), np.linspace(-2, 2, 300))

def punktquelle(q, ort):                # E ~ q r̂/r  (Ebene, wie Kap. 1/2)
    dx, dy = x - ort[0], y - ort[1]
    r2 = np.maximum(dx**2 + dy**2, 1e-9)
    return q*dx/r2, q*dy/r2

def drahtfeld(strom, ort):              # B ~ I φ̂/r  (Draht senkrecht zur Ebene)
    dx, dy = x - ort[0], y - ort[1]
    r2 = np.maximum(dx**2 + dy**2, 1e-9)
    return -strom*dy/r2, strom*dx/r2

ex, ey = (a+b for a, b in zip(punktquelle(+1, (-0.7, 0)),
                              punktquelle(-1, (+0.7, 0))))
bx, by = (a+b for a, b in zip(drahtfeld(+1, (-0.7, 0)),
                              drahtfeld(-1, (+0.7, 0))))

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9.2, 4.3))
for ax, (u, v), titel in [(ax1, (ex, ey), "E-Feld zweier Ladungen ±q"),
                          (ax2, (bx, by), "B-Feld zweier Ströme ±I")]:
    ax.streamplot(x, y, u, v, color="tab:blue", density=1.3,
                  linewidth=0.8, arrowsize=1.1)
    ax.plot(-0.7, 0, "o", color="red", markersize=10)
    ax.plot(+0.7, 0, "o", color="blue", markersize=10)
    ax.set_title(titel, fontsize=11)
    ax.set_aspect("equal"); ax.set_xticks([]); ax.set_yticks([])
plt.tight_layout(); plt.show()
```

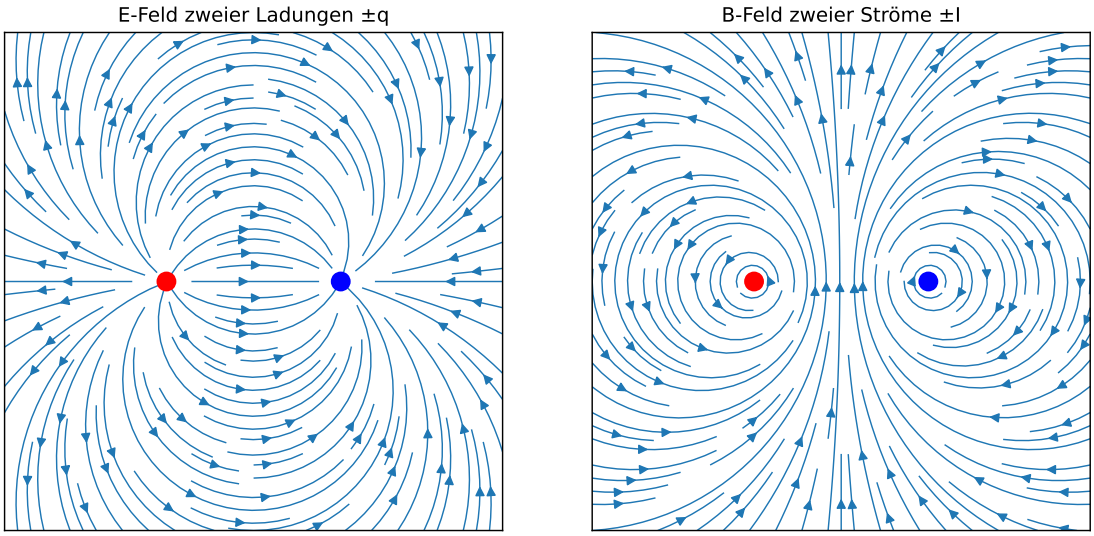


Abbildung 3.1: Gleiche Anordnung, fundamental verschiedene Felder: Elektrische Feldlinien (links) beginnen und enden auf den Ladungen. Magnetfeldlinien (rechts) umkreisen die Ströme und schließen sich — sie haben weder Anfang noch Ende (div B = 0).

Links das vertraute Bild: Linien *starten* auf der positiven und *enden* auf der negativen Ladung. Rechts dagegen startet und endet nichts — jede Linie kreist um einen Strom oder umschlingt beide. Weit weg sehen beide Bilder zum Verwechseln ähnlich aus (deshalb ist die Vermutung oben so naheliegend!), aber an den Quellpunkten selbst trennt sich die Welt: **E entspringt, B umschlingt.**

Wer den Code aufmerksam liest, entdeckt zwischen den beiden Bildern noch eine versteckte Verwandtschaft. Die zwei Feld-Funktionen unterscheiden sich nur in einer Zeile:

```
punktquelle:  ( q·dx/r²,  q·dy/r²)     # zeigt vom Quellpunkt weg
drahtfeld:    (-I·dy/r²,  I·dx/r²)     # zeigt quer dazu
```

Aus dem Pfeil (a, b) wird der Pfeil $(-b, a)$ — und das ist eine Drehung um exakt $90°$, denn das Skalarprodukt der beiden ist $a \cdot (-b) + b \cdot a = 0$, und Skalarprodukt null heißt senkrecht (Kapitel 1). Das B-Feld eines Drahtes ist also das E-Feld einer Punktladung, **an jedem Ort um $90°$ gedreht**. Diese Eigenschaft überlebt sogar das Addieren: Zwei Pfeile erst drehen und dann addieren gibt dasselbe wie erst addieren und dann drehen. Solange die Ströme an denselben Orten sitzen wie die Ladungen (und ihre Stärken im selben Verhältnis stehen), ist deshalb in *jedem* Bildpunkt das Gesamt-B das um

90° gedrehte Gesamt-E — die beiden Linienscharen müssen sich überall im rechten Winkel kreuzen. Das können wir sehen *und* messen: Wir legen beide Bilder übereinander und lassen an jedem Gitterpunkt den Winkel-Test aus Kapitel 1 laufen, den Kosinus aus dem Skalarprodukt. Erwartung: null, überall.

```
# von oben: x, y, ex, ey, bx, by (die Felder aus dem Doppelbild)
fig, ax = plt.subplots(figsize=(5.2, 5.2))
ax.streamplot(x, y, ex, ey, color="tab:blue", density=0.9,
              linewidth=0.8, arrowsize=1.0)
ax.streamplot(x, y, bx, by, color="tab:orange", density=0.9,
              linewidth=0.8, arrowsize=1.0)
ax.plot(-0.7, 0, "o", color="red", markersize=10)
ax.plot(+0.7, 0, "o", color="blue", markersize=10)
ax.set_title("E (blau) und B (orange): überall senkrecht", fontsize=11)
ax.set_aspect("equal"); ax.set_xticks([]); ax.set_yticks([])
plt.tight_layout(); plt.show()

# Winkel-Test (Kapitel 1): cos = (E·B) / (|E|·|B|) an jedem Gitterpunkt;
# der winzige Nenner-Schutz fängt 0/0 an den Quellpunkten selbst ab
betrag = np.maximum(np.hypot(ex, ey) * np.hypot(bx, by), 1e-30)
cos_winkel = (ex * bx + ey * by) / betrag
print(f"größter |cos| zwischen E und B im Bild: "
      f"{np.abs(cos_winkel).max():.1e}   (senkrecht = 0)")
```

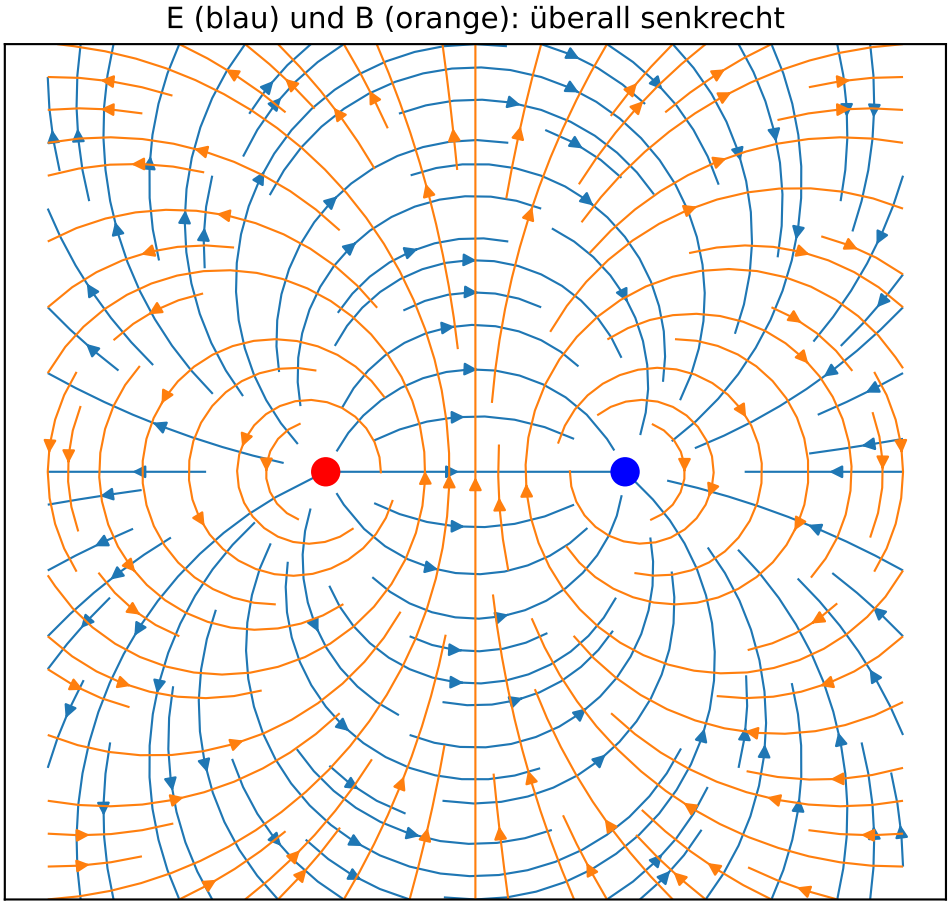


Abbildung 3.2: Beide Teilbilder übereinandergelegt: das E-Feld der Ladungen (blau) und das B-Feld der Ströme (orange) kreuzen sich überall im rechten Winkel — B ist in jedem Punkt das um 90° gedrehte E. Die Messung darunter bestätigt es: Der größte Kosinus eines Kreuzungswinkels im ganzen Bild ist exakt null, nicht bloß ungefähr.

```
größter |cos| zwischen E und B im Bild: 0.0e+00   (senkrecht = 0)
```

Sonst feiern wir in diesem Buch schon Werte um 10^{-16} als „exakt bis auf Rundung" — hier steht eine blanke Null, und auch das hat einen Grund: Die 90°-Drehung vertauscht nur die beiden Komponenten und kippt ein Vorzeichen, dabei wird nichts gerundet. Im Skalarprodukt hebt sich dann Term für Term dasselbe Produkt mit umgekehrtem Vorzeichen weg — das Senkrechtstehen ist keine Näherung, sondern Algebra, und der Computer führt sie fehlerfrei aus. Und diese Drehung ist mehr als ein Kuriosum: Sie ist der geometrische Grund, warum Gleichung 2 eine Null trägt, wo Gleichung 1 ein

ρ hat. Dreht man nämlich ein Feld in jedem Punkt um 90°, tauschen „entspringen" und „umschlingen" die Rollen — aus einem Feld mit Quellen und ohne Wirbel (E der ruhenden Ladungen) wird ein Feld ohne Quellen und mit Wirbeln (B der stetigen Ströme). Aber Vorsicht mit der Verallgemeinerung: Dieses Überall-Senkrecht ist die Signatur *dieser Paarung* — Ströme exakt an den Ladungsorten, in unserer flachen 2D-Welt. Es ist **kein Naturgesetz** „E steht immer senkrecht auf B": Stell den Draht woandershin als die Ladung, und schon kreuzen sich die Linien schief. Bei der Lichtwelle wird uns das Senkrechtstehen wieder begegnen (Kapitel 4) — dort aus einem ganz anderen Grund.

3.4 Gleichung 3: Wandel im Magnetfeld wirbelt das E-Feld

Bis hierher war alles statisch. Gleichung (3) — das **Induktionsgesetz** von Faraday — bringt die Zeit ins Spiel: Wo sich $\vec{B}$ *ändert*, bekommt das elektrische Feld **Rotation**. Das Paddelrad aus Kapitel 2 würde sich drehen — und ein Elektron, das auf einer Kreisbahn sitzt, wird herumgeschoben. Genau davon lebt der Fahrraddynamo (Magnet dreht sich, $\vec{B}$ durch die Spule ändert sich, das gewirbelte E-Feld treibt den Strom) und das Induktionsfeld deines Herdes.

Prüfen wir die Gleichung punktweise — mit den Werkzeugen aus Kapitel 2. Eine Spule mit Radius $R = 0{,}5$ m erzeugt in ihrem Inneren ein Magnetfeld, das wir gleichmäßig anwachsen lassen: $\partial B_z/\partial t = 0{,}5$ T/s, außerhalb null. Das zugehörige induzierte E-Feld wirbelt um die Spulenachse (aus Symmetriegründen auf Kreisen). Faraday verlangt: $(\nabla \times \vec{E})_z$ muss **innerhalb** der Spule exakt $-0{,}5$ sein und **außerhalb** exakt null — obwohl das E-Feld selbst auch außerhalb kräftig ist!

```
R_SPULE, K = 0.5, 0.5                        # Spulenradius (m), dB/dt (T/s)
x, y = np.meshgrid(np.linspace(-1.2, 1.2, 400), np.linspace(-1.2, 1.2, 400))
dx = x[0, 1] - x[0, 0]
r = np.maximum(np.hypot(x, y), 1e-9)

# Induziertes E: innen wächst es mit r, außen fällt es mit 1/r.
# np.where(bedingung, a, b): elementweise a, wo die Bedingung gilt — sonst b
e_phi = np.where(r < R_SPULE, -K*r/2, -K*R_SPULE**2/(2*r))
ex, ey = e_phi*(-y/r), e_phi*(x/r)          # tangential angesetzt

rot_e = np.gradient(ey, dx, axis=1) - np.gradient(ex, dx, axis=0)
```

```
fig, ax = plt.subplots(figsize=(5.6, 4.6))
bild = ax.imshow(rot_e, extent=(-1.2, 1.2, -1.2, 1.2), origin="lower",
                 cmap="RdBu_r", vmin=-0.6, vmax=0.6)
s = (slice(None, None, 25), slice(None, None, 25))
ax.quiver(x[s], y[s], ex[s], ey[s], color="k", scale=3)
ax.add_patch(plt.Circle((0, 0), R_SPULE, fill=False, ls="--", color="k"))
fig.colorbar(bild, label="(∇×E)$_z$  in V/m²")
ax.set_xlabel("x (m)"); ax.set_ylabel("y (m)")
ax.set_title("Induktion: rot E = −∂B/∂t")
ax.set_aspect("equal"); plt.tight_layout(); plt.show()

innen, aussen = r < 0.8*R_SPULE, (r > 1.2*R_SPULE) & (r < 1.0)
print(f"rot E innen : {np.mean(rot_e[innen]):+.4f}   (Soll: −∂B/∂t = {-K})")
print(f"rot E außen : {np.mean(rot_e[aussen]):+.6f}   (Soll: 0)")
```

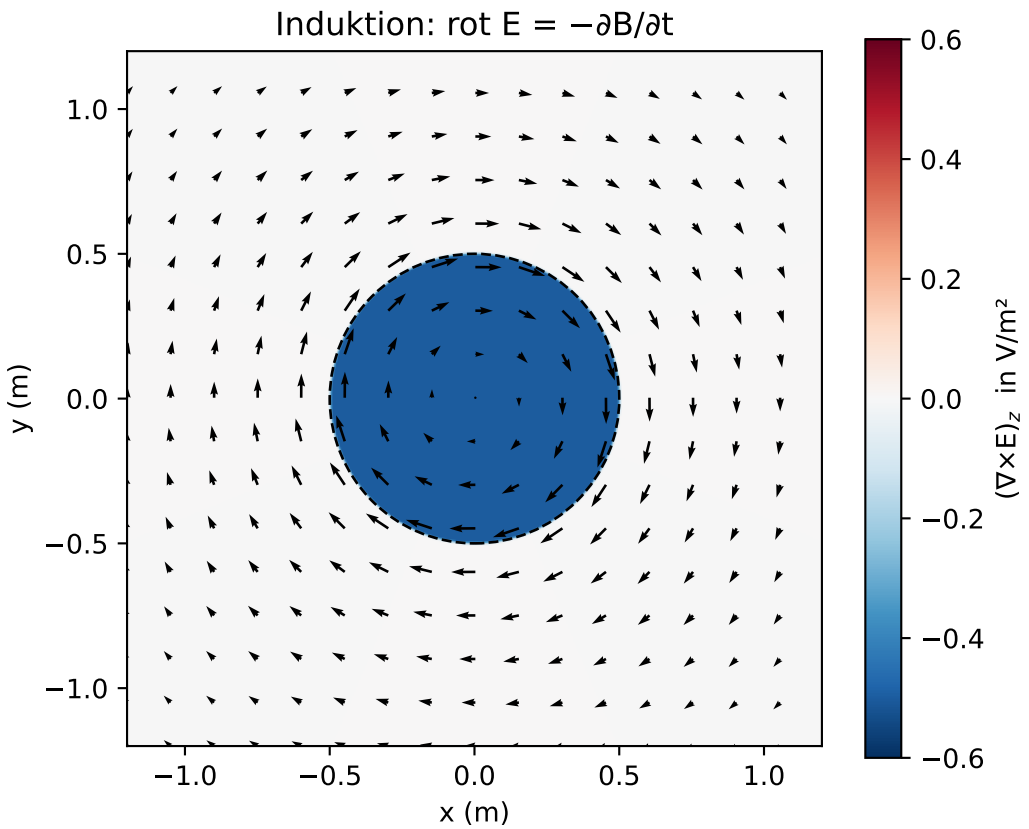


Abbildung 3.3: Punktweise Probe von Faraday: Das induzierte E-Feld (Pfeile) wirbelt auch weit außerhalb der Spule — aber seine Rotation (Farbe) ist nur dort von null verschieden, wo sich B tatsächlich ändert: innerhalb der Spule (gestrichelter Kreis), mit dem Wert −∂B/∂t.

```
rot E innen : -0.5000   (Soll: −∂B/∂t = -0.5)
rot E außen : -0.000000   (Soll: 0)
```

Beide Werte sitzen. Beachte die Pointe am Rand des Bildes: Außerhalb der Spule gibt es ein durchaus starkes, im Kreis laufendes E-Feld — aber **keine Rotation**. Das ist exakt der Strudel aus Übung 2.4: Das Feld fällt mit $1/r$, und das Paddelrad bleibt still. Die ganze Wirbelstärke konzentriert sich dort, wo $\vec{B}$ sich wirklich ändert. Faraday ist eine *lokale* Aussage.

Das Minuszeichen hat übrigens einen Ehrentitel (Lenz'sche Regel) und eine Lebensaufgabe: Es sorgt dafür, dass induzierte Felder ihrer Ursache *entgegen*wirken. In Übung 3.2 überlegst du, was ohne das Minus passieren würde.

3.5 Gleichung 4: Ströme wirbeln das B-Feld — und wandelnde E-Felder auch

Der erste Teil ist das **Ampère-Gesetz**, und du hast es in Übung 2.4 schon in den Händen gehalten: Das Magnetfeld umkreist den Strom, und die Zirkulation entlang einer Schleife misst, wie viel Strom hindurchsticht. In SI:

$$\oint \vec{B} \cdot \mathrm{d}\vec{l} \;=\; \mu_0 \, I_{\text{innen}}.$$

Vorhersage-Punkt

Bevor du weiterliest: Zwei parallele Drähte tragen $+1$ A und -1 A (hin und zurück — ein ganz normales Kabel!). Was misst eine Zirkulationsschleife, die **beide** Drähte umschließt? Und was eine, die daneben liegt und keinen umschließt?

```
from scipy.constants import mu_0

DRAEHTE = [(-0.5, 0.0, +1.0), (+0.5, 0.0, -1.0)]   # (x, y, Strom in A)

def zirkulation(mitte, radius, n=4000):
    """∮ B·dl um einen Kreis, B = Summe der Drahtfelder μ₀I/(2πr)·φ̂"""
    t = np.linspace(0, 2*np.pi, n)
    px, py = mitte[0] + radius*np.cos(t), mitte[1] + radius*np.sin(t)
    bx, by = 0.0, 0.0
    for wx, wy, strom in DRAEHTE:
        dx_, dy_ = px - wx, py - wy
        r2 = dx_**2 + dy_**2
        bx += -mu_0*strom/(2*np.pi) * dy_/r2
        by += +mu_0*strom/(2*np.pi) * dx_/r2
    return np.trapezoid((-bx*np.sin(t) + by*np.cos(t))*radius, t)

for name, mitte, radius in [("um den +1-A-Draht", (-0.5, 0), 0.3),
                            ("um beide Drähte  ", (0, 0),    1.5),
                            ("um keinen Draht  ", (0, 1.5),  0.4)]:
    print(f"Schleife {name}: ∮B·dl = {zirkulation(mitte, radius):+.4e} T·m"
          f"   (μ₀·1A = {mu_0:+.4e})")
```

```
Schleife um den +1-A-Draht: ∮B·dl = +1.2566e-06 T·m   (μ₀·1A = +1.2566e-06)
Schleife um beide Drähte  : ∮B·dl = +2.6470e-23 T·m   (μ₀·1A = +1.2566e-06)
Schleife um keinen Draht  : ∮B·dl = -3.3087e-24 T·m   (μ₀·1A = +1.2566e-06)
```

Um den einzelnen Draht: exakt $\mu_0 \cdot 1\,\mathrm{A}$. Um beide: null — die Ströme heben sich auf, und genau deshalb sind Kabel, in denen Hin- und Rückleiter dicht beieinander liegen, nach außen magnetisch fast unsichtbar. Um keinen: null, egal wie stark das Feld auf der Schleife selbst ist.

Aber das Ampère-Gesetz allein hat ein Loch, und Maxwells Flicken dafür ist der eigentliche Geniestreich. Betrachte einen **Kondensator, der gerade auflädt**: Strom fließt durch den Draht, also wirbelt ein B-Feld um den Draht. Zwischen den Platten aber fließt *kein* Strom — dort wandert keine einzige Ladung über den Spalt. Hört das B-Feld dort schlagartig auf? Maxwells Antwort: Nein — denn zwischen den Platten **wächst das elektrische Feld**, und ein wachsendes E-Feld wirbelt das Magnetfeld genauso wie ein Strom. Das ist der zweite Term, $\mu_0\varepsilon_0\,\partial\vec{E}/\partial t$, der **Verschiebungsstrom**. Ob die Buchhaltung aufgeht, verrät ein Dreizeiler:

```
# von oben: epsilon_0 (Import aus scipy.constants, Gauß-Abschnitt)
I = 1.0                  # Ladestrom (A)
A = 0.01                 # Plattenfläche (m²) — 10 cm × 10 cm
# Plattenkondensator: E = Q/(ε₀A)  →  dE/dt = I/(ε₀A)
dE_dt = I/(epsilon_0*A)
print(f"dE/dt zwischen den Platten : {dE_dt:.3e} V/(m·s)")
print(f"Verschiebungsstrom ε₀·dE/dt·A : {epsilon_0*dE_dt*A:.6f} A")
```

```
dE/dt zwischen den Platten : 1.129e+13 V/(m·s)
Verschiebungsstrom ε₀·dE/dt·A : 1.000000 A
```

Der „Strom aus wachsendem Feld” zwischen den Platten ist **exakt so groß wie der Ladestrom im Draht** — das B-Feld läuft nahtlos über den Spalt hinweg, als wäre der Draht nie unterbrochen. Beachte die Größenordnung: Über zehn *Billionen* Volt pro Meter und Sekunde muss das Feld zulegen, um ein einziges Ampere zu vertreten. Deshalb fiel dieser Term niemandem im Labor auf — Maxwell fand ihn nicht durch Messen, sondern weil die Gleichungen ohne ihn unstimmig sind. Und dieser un-messbar scheinende Term ist der Schlüssel zu allem, was jetzt kommt.

3.6 Und daraus fällt: das Licht

Schau dir die Gleichungen (3) und (4) im leeren Raum an ($\rho = 0$, $\vec{J} = 0$ — keine Ladung weit und breit):

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t}, \qquad \nabla \times \vec{B} = \mu_0 \varepsilon_0 \frac{\partial \vec{E}}{\partial t}.$$

Ein sich änderndes $\vec{B}$ erzeugt ein wirbelndes $\vec{E}$. Dieses $\vec{E}$ ändert sich dabei selbst — und erzeugt (Verschiebungsstrom!) ein wirbelndes $\vec{B}$. Das sich ändert. Und wieder $\vec{E}$ wirbelt. **Die beiden Felder reichen sich gegenseitig weiter, ganz ohne Ladungen und Drähte** — eine Kettenreaktion, die durch den Raum läuft.

Wie schnell? Machen wir es eindimensional konkret: Eine Störung läuft in x-Richtung, E zeigt in y, B in z, beide hängen nur von x und t ab. Dann bleibt von den beiden Rotationen je ein einziger Term übrig, und die Gleichungen werden zum Pingpong-Paar

$$\frac{\partial E}{\partial x} = -\frac{\partial B}{\partial t}, \qquad -\frac{\partial B}{\partial x} = \mu_0 \varepsilon_0 \frac{\partial E}{\partial t}.$$

Leite die erste nach x ab, die zweite nach t, und setze ineinander ein — B fällt heraus:

$$\frac{\partial^2 E}{\partial x^2} = \mu_0 \varepsilon_0 \frac{\partial^2 E}{\partial t^2}.$$

Das ist eine **Wellengleichung**. Sie wird von jedem Profil $f(x - vt)$ gelöst, das unverformt mit dem Tempo v davonläuft — aber nur, wenn $v^2 = 1/(\mu_0 \varepsilon_0)$. Die Gleichungen diktieren das Tempo. Rechnen wir es aus — aus den zwei Konstanten, die mit Licht scheinbar nichts zu tun haben:

Vorhersage-Punkt

Bevor du die Zelle liest: $\varepsilon_0 \approx 8{,}854 \cdot 10^{-12}$ As/(Vm) stammt aus Kondensator-Messungen, $\mu_0 \approx 1{,}257 \cdot 10^{-6}$ Vs/(Am) aus Strom-Kraft-Messungen. Was schätzt du, welche Zahl $1/\sqrt{\varepsilon_0 \mu_0}$ ergibt?

```
# von oben: epsilon_0, mu_0 (Importe aus scipy.constants)
from scipy.constants import c
print(f"1/sqrt(ε₀·μ₀) = {1/np.sqrt(epsilon_0*mu_0):,.0f} m/s")
print(f"Lichtgeschwindigkeit (Messwert) = {c:,.0f} m/s")
```

```
1/sqrt(ε₀·μ₀) = 299,792,458 m/s
Lichtgeschwindigkeit (Messwert) = 299,792,458 m/s
```

Die Lichtgeschwindigkeit. Auf alle Stellen. Zwei Laborkonstanten über Kondensatoren und Spulen, und heraus fällt das Tempo des Lichts — das war Maxwells Moment im Jahr 1865, und seine Schlussfolgerung war so kühn wie unausweichlich: Licht *ist* eine elektromagnetische Welle.

Zum Abschluss drehen wir den Spieß um und lassen die Gleichungen selbst entscheiden. Wir schicken einen Gauß-Puls $E = f(x - vt)$, $B = E/c$ mit *verschiedenen* Tempi v in beide 1D-Gleichungen und messen mit den zentralen Differenzen aus Kapitel 2, wie stark er sie verletzt:

```
X = np.linspace(-10, 10, 2001)
DX = X[1] - X[0]

def residuum(v):
    """Relative Verletzung beider 1D-Gleichungen durch f(x - v·t)."""
    dt = DX/(4*c)
    def e(t): return np.exp(-((X - v*t)/1.0)**2)
    de_dx = np.gradient(e(0.0), DX)
    db_dt = (e(dt) - e(-dt))/(2*dt)/c          # B = E/c
    r1 = np.max(np.abs(de_dx + db_dt))/np.max(np.abs(de_dx))
    db_dx = de_dx/c
    de_dt = (e(dt) - e(-dt))/(2*dt)
    r2 = np.max(np.abs(db_dx + mu_0*epsilon_0*de_dt))/np.max(np.abs(db_dx))
    return max(r1, r2)

tempi = np.linspace(0.8, 1.2, 81)
fig, ax = plt.subplots(figsize=(6.0, 3.6))
ax.semilogy(tempi, [residuum(v*c) for v in tempi], "o-", markersize=3)
ax.axvline(1.0, color="gray", lw=0.8, ls="--")
ax.set_xlabel("Puls-Tempo v / c")
ax.set_ylabel("Verletzung der Gleichungen")
plt.tight_layout(); plt.show()
```

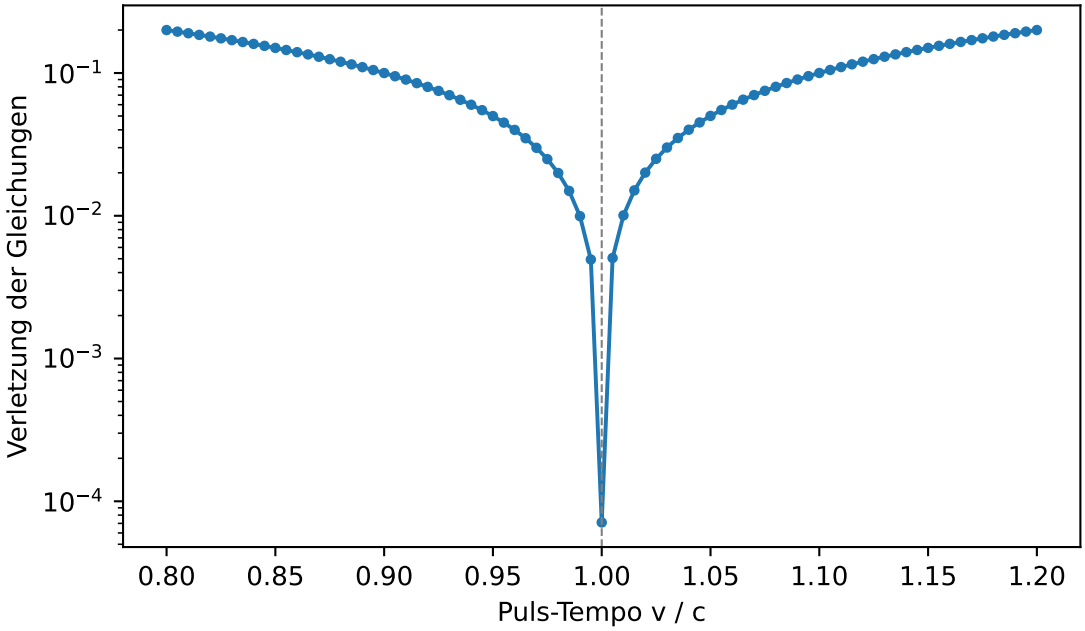


Abbildung 3.4: Der Maxwell-Prüfstand: Ein Puls f(x − v·t) verletzt die beiden Wirbelgleichungen (Residuum, logarithmisch) — außer bei exakt v = c. Die Gleichungen erzwingen das Lichttempo.

Ein scharfer Trichter, drei Größenordnungen tief, mit der Spitze bei exakt $v = c$ (der Bodensatz von $\sim 10^{-4}$ ist unser Diskretisierungsfehler — du weißt seit Kapitel 2, wie er schrumpft). Zu langsam? Verboten. Zu schnell? Verboten. **Von allen denkbaren Tempi lassen die Maxwell-Gleichungen für diese Welle genau eines zu.**

Das **Kapitel-Programm** `programme/kap03/kap03_maxwell_pruefstand.py` bündelt alle Tests dieses Kapitels — Gauß-Hüllen (inklusive Würfel), Ampère-Schleifen und den Tempo-Trichter — mit `assert`-Schranken zum Selbst-Ausführen und Umbauen.

Merkkasten

- Maxwell = **zwei Quellen-Aussagen + zwei Wirbel-Aussagen**: Ladungen sind Quellen von $\vec{E}$; $\vec{B}$ hat keine Quellen; änderndes $\vec{B}$ wirbelt $\vec{E}$ (Induktion); Ströme *und* änderndes $\vec{E}$ (Verschiebungsstrom) wirbeln $\vec{B}$.
- **Gauß-Gesetz praktisch:** Fluss durch jede geschlossene Hülle = $q_{\text{innen}}/\varepsilon_0$ — Form, Größe und Ladungsposition egal.
- **E entspringt, B umschlingt:** Magnetfeldlinien sind geschlossene

Schleifen; ein geteilter Magnet gibt zwei Magnete, nie einen Pol.
- Im leeren Raum reichen sich $\vec{E}$ und $\vec{B}$ gegenseitig weiter: Wellengleichung mit erzwungenem Tempo $c = 1/\sqrt{\varepsilon_0 \mu_0} = 299\,792\,458$ m/s.

Roter Faden

Dieses Kapitel hat eingelöst, was Kapitel 1 und 2 vorbereitet haben: Das Skalarprodukt misst Flüsse durch Hüllen, Divergenz und Rotation *sind* die linke Seite aller vier Gleichungen, die Kapitel-2-Refutation war Gleichung (1), und der Strudel aus Übung 2.4 war das Feld um den Draht in Gleichung (4). Nach vorn: **Kapitel 4** nimmt die ebene Welle auseinander, die hier herausgefallen ist (Polarisation, was schwingt in welche Richtung). Ab **Kapitel 5** bauen wir die zwei Wirbelgleichungen direkt als Nachbar-Differenzen-Schleife nach — das ist FDTD, der Simulationsmotor des restlichen Buchs. Und der Verschiebungsstrom von eben ist dort kein exotischer Korrekturterm, sondern die halbe Update-Regel.

Übungen

Ü 3.1 (Verstehen). Eine geschlossene Hülle enthält keine Ladung, aber direkt neben ihr sitzt eine starke Ladung — das E-Feld auf der Hülle ist überall deutlich von null verschieden. Warum ist der Gesamtfluss trotzdem exakt null?

Musterlösung zu Ü 3.1

Jede Feldlinie der äußeren Ladung, die in die Hülle **eintritt**, muss sie auch wieder **verlassen** — drinnen gibt es ja nichts, wo sie enden könnte. Eintritt zählt negativ ($\vec{E} \cdot \vec{n} < 0$), Austritt positiv; in der Summe hebt sich beides exakt auf. Der Fluss zählt nicht „wie viel Feld ist auf der Hülle", sondern „wie viel entspringt darin".

Ü 3.2 (Verstehen). Stell dir Gleichung (3) mit einem Pluszeichen vor: Das induzierte E-Feld würde den Stromfluss antreiben, der das wachsende B-Feld *verstärkt*. Was würde mit einer Leiterschleife in einem leicht wachsenden Magnetfeld passieren — und warum kann unsere Welt so nicht funktionieren?

Musterlösung zu Ü 3.2

Mit Pluszeichen entstünde ein Teufelskreis: mehr B → mehr induzierter Strom → noch mehr B → … Die Schleife würde sich aus dem Nichts aufschaukeln, Energie aus nichts erzeugen — ein Perpetuum mobile. Das Minuszeichen (Lenz'sche Regel) dreht den Kreis um: Die Induktion *bremst* ihre eigene Ursache, jede Änderung kostet Arbeit, die Energiebilanz geht auf. (In Kapitel 18 wird daraus eine exakte Buchhaltung.)

Ü 3.3 (Verändern). Ändere im Kapitel-Programm den Ansatz des Pulses: Lasse $B = E/c$ unverändert, aber gib dem Puls das Tempo $v = 0{,}9c$ — welcher der beiden `assert`-Tests schlägt zuerst an? Probiere danach $B = E/(0{,}9c)$ *und* $v = 0{,}9c$ zusammen: Lässt sich Maxwell so überlisten?

Musterlösung zu Ü 3.3

Die beiden Varianten, mit getrennt ausgewiesenen Residuen:

```
# von oben: X, DX (Tempo-Trichter-Zelle); c, mu_0, epsilon_0 (Importe)
def residuen(v, b_tempo):
    """Verletzung von Faraday und Ampère für E = f(x-vt), B = E/b_tempo."""
    dt = DX/(4*c)
    def e(t): return np.exp(-((X - v*t)/1.0)**2)
    de_dx = np.gradient(e(0.0), DX)
    de_dt = (e(dt) - e(-dt))/(2*dt)
    faraday = np.max(np.abs(de_dx + de_dt/b_tempo))/np.max(np.abs(de_dx))
    ampere = np.max(np.abs(de_dx/b_tempo + mu_0*epsilon_0*de_dt)) \
        / np.max(np.abs(de_dx/b_tempo))
    return faraday, ampere

for fall, v, bt in [("B = E/c,      v = 0,9c", 0.9*c, c),
                    ("B = E/(0,9c), v = 0,9c", 0.9*c, 0.9*c)]:
    fa, am = residuen(v, bt)
    print(f"{fall}:   Faraday {fa:.3f}   Ampère {am:.3f}")
```

```
B = E/c,      v = 0,9c:   Faraday 0.100   Ampère 0.100
B = E/(0,9c), v = 0,9c:   Faraday 0.000   Ampère 0.190
```

Mit $v = 0{,}9c$ verletzt der Puls **beide** Gleichungen um je 10 % (das Residuum ist $|1-v/c|$) — beide `assert`-Tests schlagen an. Auch der Rettungsversuch scheitert: Mit $B = E/v$ ist Faraday zwar exakt erfüllt (übrig bleibt nur der Diskretisierungsfehler), aber Ampère wird umso stärker verletzt — Residuum $|1 - v^2/c^2| = 19\,\%$. Die beiden Gleichungen zu-

sammen lassen sich nur mit $v = c$ gleichzeitig befriedigen. Genau das zeigt der Trichter in Abbildung 3.4.

Ü 3.4 (Übertragen). Baue den Ampère-Test zu einem **Koaxialkabel** um: Innenleiter mit $+1$ A im Ursprung, Rückstrom -1 A verteilt auf acht Drähte gleichmäßig auf einem Kreis mit Radius $0{,}5$ m. Berechne die Zirkulation für Schleifenradien $0{,}2$ m, $0{,}7$ m und $1{,}5$ m um den Ursprung. Was bedeutet das Ergebnis für Geräte in der Nähe eines solchen Kabels?

Musterlösung zu Ü 3.4

Die Drahtliste wird ersetzt, die `zirkulation`-Funktion bleibt unverändert:

```
# von oben: zirkulation()  (Ampère-Abschnitt; sie liest DRAEHTE neu aus)
winkel8 = np.linspace(0, 2*np.pi, 9)[:-1]          # 8 Positionen
DRAEHTE = [(0.0, 0.0, +1.0)] \
    + [(0.5*np.cos(w), 0.5*np.sin(w), -1.0/8) for w in winkel8]

for radius in (0.2, 0.7, 1.5):
    print(f"Schleifenradius {radius} m: ∮B·dl = "
          f"{zirkulation((0, 0), radius):+.4e} T·m   (μ₀·1A = {mu_0:.4e})")
```

```
Schleifenradius 0.2 m: ∮B·dl = +1.2566e-06 T·m   (μ₀·1A = 1.2566e-06)
Schleifenradius 0.7 m: ∮B·dl = -4.1359e-25 T·m   (μ₀·1A = 1.2566e-06)
Schleifenradius 1.5 m: ∮B·dl = +2.6593e-24 T·m   (μ₀·1A = 1.2566e-06)
```

Radius $0{,}2$ m (nur Innenleiter umschlossen): $\mu_0 \cdot 1$ A. Radius $0{,}7$ m und $1{,}5$ m (alle Ströme umschlossen): null — Hin- und Rückstrom heben sich auf. Außerhalb eines Koaxialkabels verschwindet das Magnetfeld (bei idealer Symmetrie sogar exakt, nicht nur die Zirkulation): Das Kabel stört seine Umgebung nicht und wird umgekehrt kaum gestört — der Grund, warum Antennenleitungen und Messleitungen koaxial gebaut werden.

Das Kleingedruckte

- **Vakuum-Form:** Unsere vier Gleichungen gelten für Ladungen und Ströme im Vakuum (und praktisch in Luft). In Materie kommen die Hilfsfelder $\vec{D}$ und $\vec{H}$ dazu, die Polarisation und Magnetisierung des Materials buchführen — sie betreten das Buch, sobald wir Materialien simulieren (Teil IV/VII).

- **Quasistatische Demos:** Bei der Induktions-Demo haben wir das E-Feld als instantan mitlaufend angesetzt. Das ist erlaubt, solange die Anordnung viel kleiner ist als die Wellenlänge der beteiligten Frequenzen — sonst muss man die endliche Laufzeit mitrechnen, und genau dafür gibt es FDTD (Teil II).
- **Abtasten statt Gittern:** Die Hüllen- und Schleifenintegrale dieses Kapitels tasten *analytisch bekannte* Felder ab. Sobald das Feld selbst nur auf einem Gitter existiert (FDTD), übernehmen die Gitterversionen von div und rot aus Kapitel 2 — die punktweise Induktions-Probe hat das schon vorgemacht.
- **Die Null in Gleichung (2)** ist Messstand, kein Beweis: Die Suche nach magnetischen Monopolen läuft seit Jahrzehnten (bislang ergebnislos), und manche Theorien sagen sie voraus. Fände man einen, bekäme Gleichung (2) eine rechte Seite — und die Gleichungen würden vollkommen symmetrisch.

4 Wellen und Polarisation

Ein Experiment für sofort, Kosten: null. Nimm eine polarisierte Sonnenbrille (fast jede moderne ist es) und schau damit auf dein Handydisplay. Jetzt drehe das Handy langsam um 90 Grad. Das Display wird dunkel — bei manchen Geräten stockfinster, als wäre es aus. Dreh zurück: wieder hell. Zwischen Brille und Display ist nichts als Luft, und trotzdem entscheidet ein *Drehwinkel* darüber, ob Licht ankommt.

Offenbar hat Licht eine eingebaute **Richtung quer zu seinem Weg** — und Displays wie Brillengläser können sie befragen. Diese Richtung heißt **Polarisation**, und um sie zu verstehen, müssen wir die Welle, die in Kapitel 3 aus den Gleichungen gefallen ist, endlich im Detail ansehen: Was genau schwingt da? In welche Richtung? Und was hat das mit dem Drehwinkel der Brille zu tun?

Lernziele

Nach diesem Kapitel kannst du …

1. … die ebene Welle $E_0 \sin(kx - \omega t)$ lesen: was k, ω, λ, T und f bedeuten und wie $c = \lambda f$ daraus folgt,
2. … erklären und numerisch belegen, dass $\vec{E}$, $\vec{B}$ und die Ausbreitungsrichtung paarweise senkrecht stehen,
3. … lineare und zirkulare Polarisation unterscheiden — am 3D-Bild und an der Frontalsicht,
4. … einen Polfilter als **Projektion** verstehen und das Malus-Gesetz $I = I_0 \cos^2 \alpha$ nachrechnen,
5. … mit $\vec{S} = \vec{E} \times \vec{H}$ den Energiefluss einer Welle berechnen — bis zur Feldstärke des Sonnenlichts in V/m.

4.1 Die ebene Welle lesen

Kapitel 3 endete mit der Wellengleichung und ihrer Lösung: ein Profil, das unverformt mit Tempo c davonläuft. Die wichtigste Ausprägung ist die **har-**

monische ebene Welle — ein reiner Sinus:

$$E(x,t) \;=\; E_0\,\sin(kx - \omega t).$$

Drei Stellschrauben, drei Bedeutungen. E_0 ist die **Amplitude** (der Maximalausschlag, in V/m). k ist die **Wellenzahl** — sie sagt, wie schnell die Phase im *Raum* tickt; nach der Strecke $\lambda = 2\pi/k$ (der **Wellenlänge**) wiederholt sich das Muster. Und ω ist die **Kreisfrequenz** — sie sagt, wie schnell die Phase in der *Zeit* tickt; nach der Zeit $T = 2\pi/\omega$ (der **Periode**) ist am festen Ort wieder derselbe Wert da, und $f = 1/T$ zählt die Schwingungen pro Sekunde in Hertz.

```
import numpy as np
import matplotlib.pyplot as plt

E0, LAMBDA, T = 1.0, 2.0, 1.0
k, omega = 2*np.pi/LAMBDA, 2*np.pi/T

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9.2, 3.4))

x = np.linspace(0, 5, 600)
ax1.plot(x, E0*np.sin(k*x), label="t = 0")
ax1.plot(x, E0*np.sin(k*x - omega*T/8), "--", label="t = T/8")
ax1.annotate("", xy=(3.5, 1.12), xytext=(1.5, 1.12),
             arrowprops=dict(arrowstyle="<->"))
ax1.text(2.5, 1.2, "λ", ha="center")
ax1.set_xlabel("Ort x"); ax1.set_ylabel("E(x, t)"); ax1.set_ylim(-1.4, 1.5)
ax1.set_title("Momentbild (Foto)"); ax1.legend(loc="lower left", fontsize=8)

t = np.linspace(0, 2.5, 600)
ax2.plot(t, E0*np.sin(k*1.0 - omega*t), color="tab:green")
ax2.annotate("", xy=(2.25, 1.12), xytext=(1.25, 1.12),
             arrowprops=dict(arrowstyle="<->"))
ax2.text(1.75, 1.2, "T", ha="center")
ax2.set_xlabel("Zeit t  (am festen Ort x = 1)"); ax2.set_ylabel("E(1, t)")
ax2.set_ylim(-1.4, 1.5); ax2.set_title("Zeitverlauf (Messschrieb)")
plt.tight_layout(); plt.show()
```

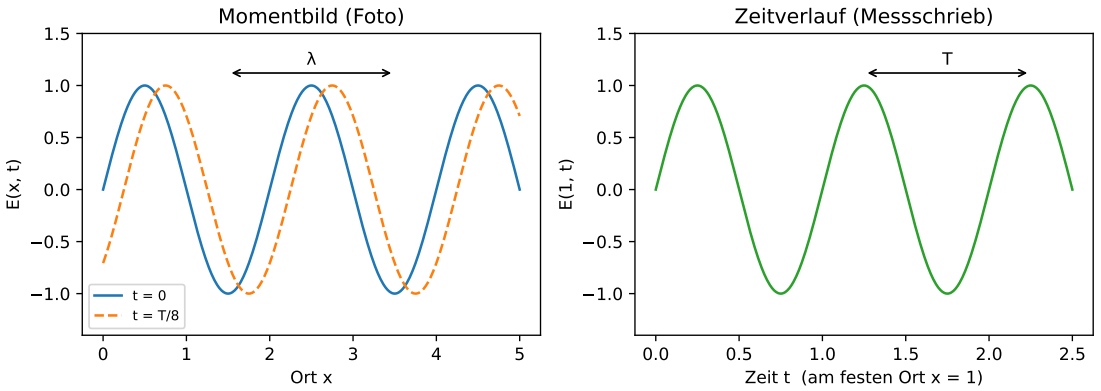


Abbildung 4.1: Zwei Blicke auf dieselbe Welle. Links das Momentbild (Foto zur festen Zeit): Das Muster wiederholt sich nach der Wellenlänge λ, und eine Achtelperiode später (gestrichelt) ist es ein Stück nach rechts gerückt. Rechts der Zeitverlauf an einem festen Ort (Messschrieb): Er wiederholt sich nach der Periode T.

Wie schnell wandert das Muster? Ein Wellenberg sitzt dort, wo das Argument $kx - \omega t$ konstant bleibt — wächst t, muss x mitwachsen, und zwar mit dem Tempo $v = \omega/k$. Mit $k = 2\pi/\lambda$ und $\omega = 2\pi f$ wird daraus die handlichste Formel der Wellenlehre:

$$c = \frac{\omega}{k} = \lambda f \qquad \text{— Wellenlänge mal Frequenz ist das Tempo.}$$

Eine Handrechnung dazu, an einer Zahl, die im Alltag ständig vorbeifliegt: WLAN funkt bei $f = 2{,}4$ GHz $= 2{,}4 \cdot 10^9$ Schwingungen pro Sekunde. Umgestellt ist $\lambda = c/f = (3{,}0 \cdot 10^8 \text{ m/s})/(2{,}4 \cdot 10^9 \text{ 1/s}) = 0{,}125$ m — **12,5 Zentimeter**. Heraus kommt eine Länge, ein ganz gewöhnlicher Zollstock-Wert.

> Vorhersage-Punkt
>
> **Bevor du die Zelle liest:** Das Stromnetz schwingt mit bescheidenen $f = 50$ Hz. Schätze die zugehörige Wellenlänge, bevor du sie berechnest — Zimmerbreite? Stadtlänge? Mehr?

```
from scipy.constants import c
for name, f in [("Stromnetz          ", 50.0),
                ("UKW-Radio          ", 100e6),
                ("WLAN               ", 2.4e9),
```

```
                ("grünes Licht       ", 5.6e14)]:
    lam = c / f
    einheit = f"{lam:,.0f} m" if lam > 1 else f"{lam*1e9:,.0f} nm"
    print(f"{name}: f = {f:>12,.0f} Hz   →   λ = {einheit}")
```

```
Stromnetz          : f =           50 Hz   →   λ = 5,995,849 m
UKW-Radio          : f =  100,000,000 Hz   →   λ = 3 m
WLAN               : f = 2,400,000,000 Hz   →   λ = 124,913,524 nm
grünes Licht       : f = 560,000,000,000,000 Hz   →   λ = 535 nm
```

Die 50-Hz-Welle ist **sechstausend Kilometer** lang — Lissabon bis Moskau. Merke dir diese Zahl: In Kapitel 25 erklärt sie, warum dein Stromkabel zum Glück keine Antenne ist. Und grünes Licht? Ein halber Mikrometer — zweihundertmal feiner als ein Haar. Es ist ein und dieselbe Physik über zwölf Größenordnungen.

> ⚠ Naheliegende Vermutung
>
> **Vermutung:** *„Die Sinuskurve zeigt den Weg des Lichts — es schlängelt sich wellenförmig durch den Raum."*
> **Warum sie naheliegt:** Jede Skizze einer Welle (auch unsere oben) zeichnet eine geschwungene Linie von links nach rechts. Bei einer Wasserwelle bewegt sich die Oberfläche ja wirklich auf und ab — warum nicht auch der Lichtstrahl?
> **Was stattdessen stimmt:** Nichts bewegt sich auf und ab, und der Weg des Lichts ist **schnurgerade**. Die Kurve ist ein *Diagramm*: An jedem Ort x der geraden Strecke trägt sie nach oben ab, **wie stark das elektrische Feld dort gerade ist** — denselben Feldwert, den seit Kapitel 1 ein Pfeil an jedem Ort darstellt. „Es schwingt" heißt: Der Zahlenwert des Feldes am festen Ort pendelt im Takt T — so wie eine Temperatur am festen Ort steigen und fallen kann, ohne dass irgendwo Materie auf- und abhüpft.

4.2 Dieselbe Welle, dreimal gezeichnet — und was davon echt ist

Wenn die Kurve nur ein Diagramm ist: Wie sieht die Welle dann *in echt* aus? Die ehrliche Antwort: **gar nicht.** Ein Feld ist kein Ding mit Aussehen — es ist eine *messbare Eigenschaft des Raums*. Die Messvorschrift kennst du seit Kapitel 1: Setze eine Probeladung an einen Punkt des (schnurgeraden!)

Strahls und miss die Kraft auf sie — Kraft pro Ladung *ist* $\vec{E}$, mit Betrag und Richtung. Läufst du den Strahl entlang und misst an jedem Punkt, bekommst du eine **Zahlenliste**. Alles, was wir zeichnen können, sind verschiedene Auftragungen dieser einen Liste:

```
x = np.linspace(0, 4, 400)
k = 2*np.pi / 2.0                        # Wellenzahl (λ = 2)
E_jetzt = np.sin(k*x)                    # die Messreihe bei t = 0
E_danach = np.sin(k*x - 0.9)             # dieselbe, einen Augenblick später

fig, (ax1, ax2, ax3) = plt.subplots(3, 1, figsize=(7.2, 6.2), sharex=True)

ax1.plot(x, E_jetzt, color="tab:blue")
ax1.axhline(0, color="k", lw=0.5)
ax1.set_ylabel("E (V/m)")
ax1.set_title("1 · Diagramm: der Messwert, nach oben aufgetragen",
              fontsize=10, loc="left")

xq = np.linspace(0, 4, 41)
ax2.quiver(xq, np.zeros_like(xq), np.zeros_like(xq), np.sin(k*xq),
           color="tab:blue", scale_units="y", scale=1, width=0.004)
ax2.axhline(0, color="k", lw=0.8)
ax2.set_ylim(-1.4, 1.4)
ax2.set_ylabel("Kraft auf\nProbeladung")
ax2.set_title("2 · Pfeile: dieselben Werte als Querkraft am Ort",
              fontsize=10, loc="left")

band = np.vstack([E_jetzt, E_danach])   # zwei Zeitpunkte übereinander
ax3.imshow(band, aspect="auto", extent=(0, 4, 0, 2),
           cmap="RdBu_r", vmin=-1, vmax=1)
ax3.set_yticks([1.5, 0.5])
ax3.set_yticklabels(["t = 0", "kurz danach"])
ax3.set_xlabel("Ort entlang des geraden Strahls (m)")
ax3.set_title("3 · Farbband: der Strahl selbst — Farbe = Messwert",
              fontsize=10, loc="left")
plt.tight_layout(); plt.show()
```

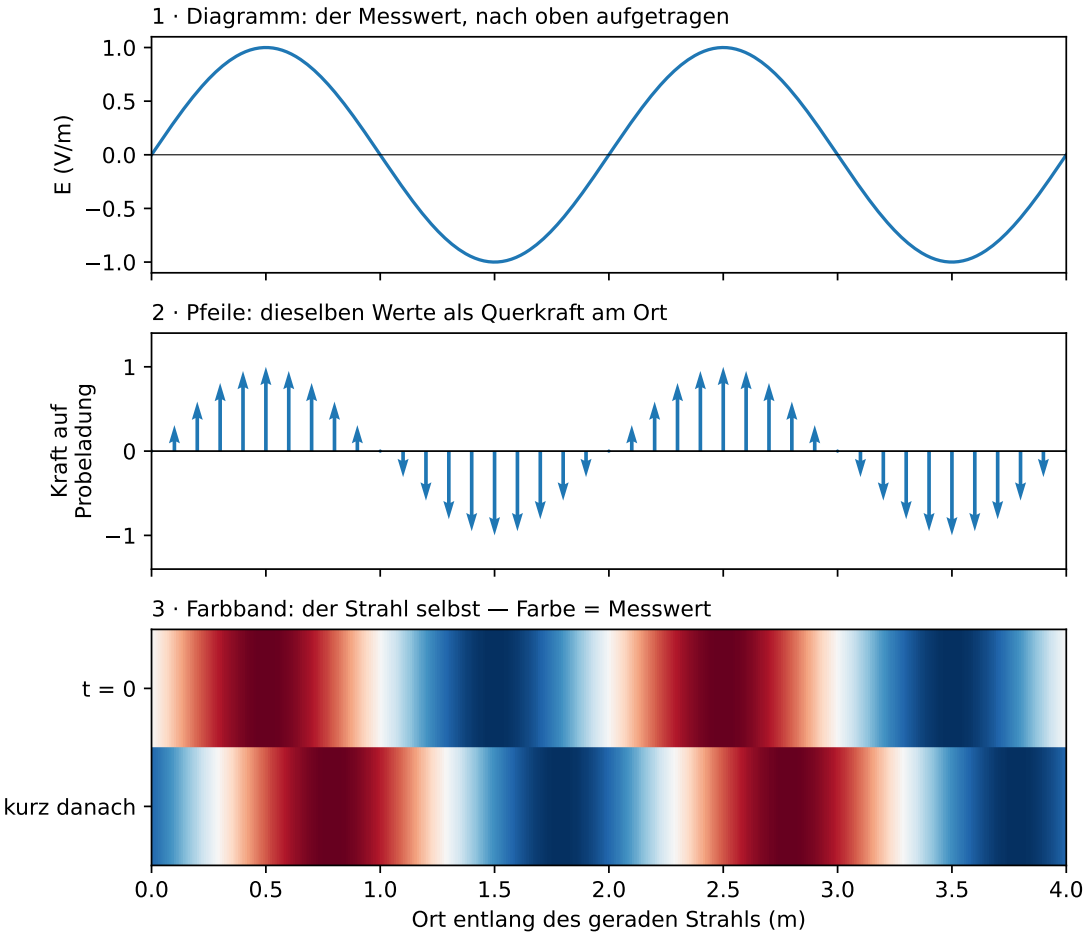


Abbildung 4.2: Ein und dieselbe Messreihe entlang eines geraden Strahls, dreimal aufgetragen. Oben als Diagramm (Messwert nach oben). Mitte als Kraftpfeile am Ort — so würde eine Probeladung an jedem Punkt gezogen. Unten als Farbband: der Strahl selbst, Farbe = Messwert (rot positiv, blau negativ); kurze Zeit später ist das Farbmuster nach rechts gerückt — gewandert ist die Eigenschaft, nicht der Strahl.

Das **Farbband** ist die ehrlichste Darstellung dessen, was räumlich passiert: ein gerader Strahl, an dem sich nichts verbiegt — nur ein Eigenschafts-Muster, das nach rechts wandert wie eine Lichterkette, deren Lämpchen nacheinander schalten. Aber Farbe kann keine **Richtung quer zum Strahl** zeigen, und genau die ist der Inhalt dieses Kapitels. Deshalb braucht es auch die Pfeile — und die sind keine bloße Diagramm-Erfindung, denn die Querrichtung ist **physikalisch real und messbar**:

- Ein Elektron in einer Empfangsantenne **zappelt wirklich quer** — entlang der Pfeile aus Panel 2, im Takt der Welle. Deshalb steht die UKW-Stabantenne aufrecht: ihre Elektronen sollen entlang des Stabes geschoben werden.
- Deine Polbrille reagiert wirklich auf den **Winkel** dieser Richtung — das $\cos^2$, das wir gleich messen, ist Experimentalphysik, kein Zeichen-Artefakt.
- Die **Beträge** tragen reale Energie (am Kapitelende: 868 V/m im Sonnenschein).

Fiktion ist an den Wellendiagrammen also nur eines: die Vorstellung, *das Licht* selbst weiche seitlich aus. Die Querbewegung gehört den **Ladungen**, auf die das Feld trifft — nicht dem Strahl. Mit dieser Übersetzungstafel im Kopf darfst du ab jetzt jede Darstellung lesen, ohne ihr auf den Leim zu gehen: **Kurven und Pfeile, wenn Richtungen und Winkel zählen** (Polarisation, Projektion) — **Farbe, wenn die Verteilung im Raum zählt** (so zeigen ab Kapitel 9 die 2D-Simulationen ihre Felder als Farbbilder).

4.3 Quer liegt das Feld: Transversalität

In welche Richtung zeigt nun der pendelnde Feldpfeil? Die Maxwell-Gleichungen lassen auch hier keine Wahl. Im leeren Raum gilt $\nabla \cdot \vec{E} = 0$ — hätte das Feld eine Komponente *entlang* der Laufrichtung, die wie ein Sinus an- und abschwillt, dann hätte es genau die Quellstärke, die Gleichung (1) ohne Ladungen verbietet. Also bleibt dem Feld nur die **Querrichtung**. Dasselbe gilt für $\vec{B}$, und die beiden Wirbelgleichungen verzahnen die Felder zusätzlich miteinander: Für eine Welle in $+x$-Richtung mit $\vec{E}$ entlang y liegt $\vec{B}$ entlang z, mit dem festen Größenverhältnis $B = E/c$ aus Kapitel 3. Alle drei Richtungen stehen **paarweise senkrecht**:

```
fig = plt.figure(figsize=(7.5, 4.2))
ax = fig.add_subplot(projection="3d")

x = np.linspace(0, 4, 300)
xq = np.linspace(0, 4, 33)                  # Stützstellen für Pfeile
k = 2*np.pi/2.0
ax.plot(x, np.sin(k*x), 0, color="tab:blue", lw=1.5, label="E (in y)")
ax.plot(x, np.zeros_like(x), np.sin(k*x), color="tab:red", lw=1.5,
        label="B (in z)")
ax.quiver(xq, 0, 0, np.zeros_like(xq), np.sin(k*xq), np.zeros_like(xq),
          color="tab:blue", lw=0.8, arrow_length_ratio=0.1)
```

```
ax.quiver(xq, 0, 0, np.zeros_like(xq), np.zeros_like(xq), np.sin(k*xq),
          color="tab:red", lw=0.8, arrow_length_ratio=0.1)
ax.plot(x, np.zeros_like(x), np.zeros_like(x), "k-", lw=0.6)
ax.set_ylim(-1.3, 1.3); ax.set_zlim(-1.3, 1.3)
ax.set_xlabel("x (Laufrichtung)"); ax.set_ylabel("E$_y$")
ax.set_zlabel("B$_z$")
ax.view_init(elev=16, azim=-55); ax.legend(loc="upper left", fontsize=9)
plt.tight_layout(); plt.show()
```

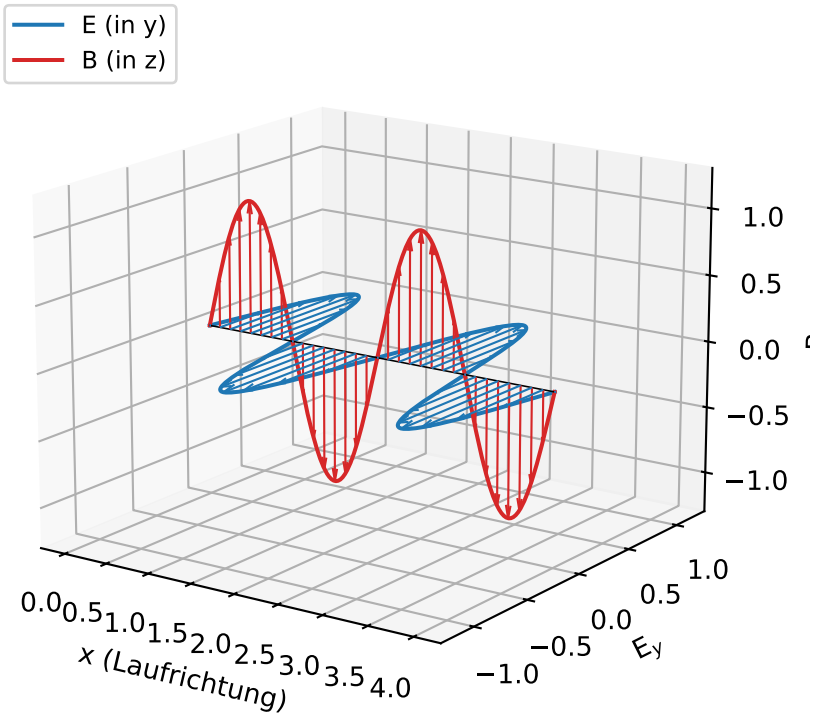


Abbildung 4.3: Momentbild einer ebenen Welle, die nach rechts läuft. Jeder Pfeil ist der Feldwert an einem Punkt der geraden x-Achse: E (blau) schwingt in y, B (rot, in Einheiten von E/c gezeichnet) schwingt in z — beide quer zum Weg, beide im Gleichtakt.

Eine Lesehilfe, denn dieses Bild wird oft missverstanden: Alle Pfeile starten **auf der geraden x-Achse** — der einzigen Linie, auf der hier irgendetwas passiert. Jeder blaue Pfeil ist der E-Feldwert an seinem Fußpunkt (er *zeigt* in y, aber er *sitzt* auf der Achse), jeder rote der B-Feldwert ebendort. Die geschwungenen Kurven verbinden nur die Pfeilspitzen — sie sind das Diagramm aus dem Kästchen oben, in 3D.

Ob die Senkrecht-Behauptung stimmt, klärt das Kreuz- und Skalarprodukt aus Kapitel 1 — an einem konkreten Punkt, mit konkreten Zahlen. Wir nehmen den Ort, an dem die Phase gerade $\pi/3$ beträgt (sin $= 0{,}866$), und setzen $E_0 = 1$ V/m:

```
# von oben: c (Import aus scipy.constants, Wellenlängen-Zelle)
from scipy.constants import mu_0

E = np.array([0.0, 0.866, 0.0])        # V/m   — zeigt in y
B = np.array([0.0, 0.0, 0.866]) / c    # T     — zeigt in z, Betrag E/c
laufrichtung = np.array([1.0, 0.0, 0.0])

print(f"E · B            = {E @ B}   (senkrecht: Skalarprodukt 0)")
print(f"E · Laufrichtung = {E @ laufrichtung}")
print(f"B · Laufrichtung = {B @ laufrichtung}")
print(f"E × B zeigt nach   {np.cross(E, B) / np.linalg.norm(np.cross(E, B))}")
```

```
E · B            = 0.0   (senkrecht: Skalarprodukt 0)
E · Laufrichtung = 0.0
B · Laufrichtung = 0.0
E × B zeigt nach   [1. 0. 0.]
```

Beide Skalarprodukte null, und das Kreuzprodukt $\vec{E} \times \vec{B}$ zeigt exakt in die Laufrichtung — das wird am Kapitelende noch wichtig, denn dort fließt die Energie.

4.4 Polarisation: die Schwingungsrichtung ist wählbar

„$\vec{E}$ liegt quer" lässt der Welle genau **eine Freiheit**: In *welche* Querrichtung zeigt der Pfeil? Diese Wahl heißt Polarisation. Zeigt $\vec{E}$ immer entlang derselben Geraden (etwa stur in y), heißt die Welle **linear polarisiert**. Die Komponenten dürfen sich aber auch abwechseln: E_y und E_z je ein Sinus, um eine Viertelperiode versetzt — dann ist der Pfeil nie null, sondern **dreht sich** mit konstanter Länge im Kreis: **zirkular polarisiert**.

```
fig = plt.figure(figsize=(9.4, 6.6))
x = np.linspace(0, 4, 300)
xq = np.linspace(0, 4, 33)
k = 2*np.pi / 2.0                # Wellenzahl (λ = 2, wie im Kamm-Bild)
```

```
for spalte, (name, fy, fz) in enumerate([
        ("linear", lambda u: np.sin(k*u), lambda u: 0*u),
        ("zirkular", lambda u: np.sin(k*u), lambda u: np.cos(k*u))]):
    ax = fig.add_subplot(2, 2, spalte + 1, projection="3d")
    ax.plot(x, fy(x), fz(x), lw=1.5, color="tab:blue")
    ax.quiver(xq, 0, 0, np.zeros_like(xq), fy(xq), fz(xq),
              color="tab:blue", lw=0.7, arrow_length_ratio=0.1)
    ax.plot(x, 0*x, 0*x, "k-", lw=0.5)
    ax.set_ylim(-1.3, 1.3); ax.set_zlim(-1.3, 1.3)
    ax.set_xlabel("x"); ax.set_ylabel("E$_y$"); ax.set_zlabel("E$_z$")
    ax.set_title(f"{name} (Momentbild)", fontsize=10)
    ax.view_init(elev=16, azim=-55)

    ax2 = fig.add_subplot(2, 2, spalte + 3)
    wt = np.linspace(0, 2*np.pi, 200)
    ax2.plot(np.sin(-wt), 0*wt if spalte == 0 else np.cos(-wt), lw=2)
    ax2.set_xlim(-1.3, 1.3); ax2.set_ylim(-1.3, 1.3)
    ax2.set_xlabel("E$_y$"); ax2.set_ylabel("E$_z$")
    ax2.set_title(f"{name}: Frontalsicht", fontsize=10)
    ax2.set_aspect("equal"); ax2.grid(alpha=0.3)
plt.subplots_adjust(hspace=0.35)
plt.show()
```

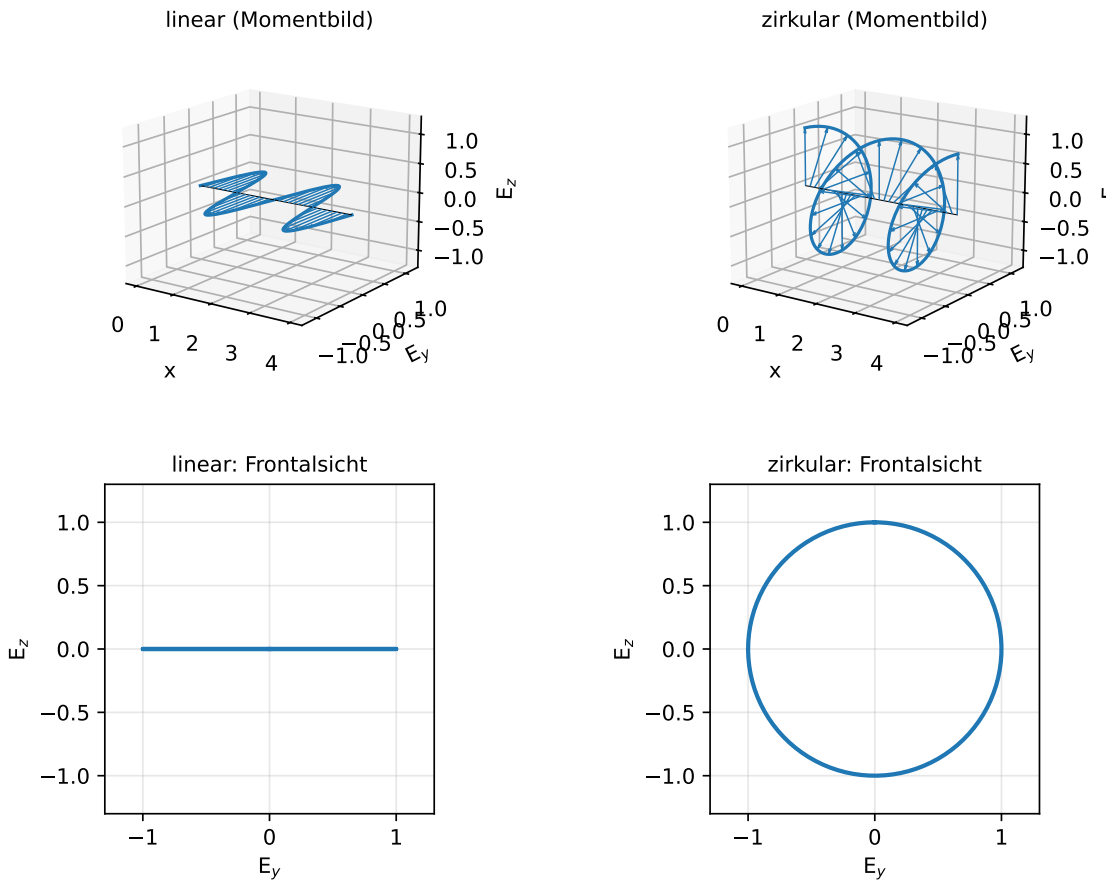


Abbildung 4.4: Links die linear polarisierte Welle: Die E-Pfeile bleiben in einer Ebene, das Momentbild ist ein flaches Band. Rechts die zirkulare: gleiche Amplitude in y und z, um T/4 versetzt — die Pfeilspitze beschreibt eine Schraube. Darunter die Frontalsicht (Blick der Welle entgegen, an einem festen Ort über eine Periode): Strich beziehungsweise Kreis.

Die **Frontalsicht** ist die ehrlichste Darstellung: Sie zeigt, was an einem festen Ort über eine Periode wirklich geschieht. Linear heißt: Der Feldpfeil pendelt auf einem **Strich** (durch null hindurch). Zirkular heißt: Er läuft mit konstanter Länge auf einem **Kreis**. Dazwischen gibt es jede Ellipse — Übung 4.4 baut eine. Sonnenlicht und Glühlampenlicht sind übrigens **unpolarisiert**: ein wildes Gemisch, dessen Schwingungsrichtung millionenfach pro Sekunde zufällig wechselt.

4.5 Der Polfilter: ein Skalarprodukt aus Plastik

Jetzt zum Brillen-Experiment. Ein Polfilter hat eine **Durchlassrichtung** $\hat{d}$: Er lässt von einem ankommenden E-Feld genau den Anteil passieren, der entlang $\hat{d}$ zeigt — und schluckt den Rest. „Der Anteil entlang einer Richtung" ist seit Kapitel 1 ein fester Begriff: die **Projektion**, berechnet mit dem Skalarprodukt. Durchgelassene Amplitude: $E_0 \cos\alpha$, wenn α der Winkel zwischen Polarisation und Durchlassrichtung ist. Die Helligkeit (Intensität) geht mit dem **Quadrat** der Amplitude — warum, klärt der Energie-Abschnitt gleich —, also:

$$I = I_0 \cos^2\alpha \qquad \text{(Malus-Gesetz).}$$

Eine Handrechnung bei $\alpha = 30°$: Durchgelassene Amplitude $\cos 30° = 0{,}866$, Intensität $0{,}866^2 = 0{,}75$ — drei Viertel der Helligkeit. Bei $\alpha = 90°$ (Brille quer zum Display): $\cos 90° = 0$, Intensität null — **stockfinster**, wie beobachtet. Das Display sendet nämlich linear polarisiertes Licht aus (so funktionieren LC-Displays intern), und die Brille fragt mit ihrem Skalarprodukt genau diese Richtung ab.

Und nun die Überraschung, die dieses harmlose $\cos^2$ versteckt:

> Vorhersage-Punkt
>
> **Bevor du die Zelle liest:** Zwei gekreuzte Filter (0° und 90°) lassen nichts durch. Jetzt schieben wir einen **dritten** Filter mit 45° **zwischen** die beiden. Es gibt jetzt mehr lichtschluckendes Material im Weg als vorher. Wird es dunkler, gleich dunkel — oder heller? Lege dich fest, mit Begründung.

```python
def durch_filter(E, winkel_grad):
    """Polfilter = Projektion des E-Vektors auf die Durchlassrichtung."""
    w = np.radians(winkel_grad)         # np.radians: Grad → Bogenmaß
    d = np.array([np.cos(w), np.sin(w)])
    return (E @ d) * d                  # Anteil entlang d, als Vektor

E_start = np.array([1.0, 0.0])          # linear polarisiert, 0°, I = 1

E1 = durch_filter(E_start, 90)
print(f"nur 90°-Filter:          I = {E1 @ E1:.4f}")

E2 = durch_filter(durch_filter(E_start, 45), 90)
print(f"erst 45°, dann 90°:      I = {E2 @ E2:.4f}")
```

```
nur 90°-Filter:          I = 0.0000
erst 45°, dann 90°:      I = 0.2500
```

Heller. Ein Viertel der Helligkeit, wo eben noch absolute Dunkelheit war — durch *Hinzufügen* eines Filters. Der Trick: Der 45°-Filter projiziert das Feld auf seine eigene Richtung und **dreht damit die Polarisation**. Das Licht, das bei ihm herauskommt, schwingt unter 45° — und davon lässt der 90°-Filter wieder den $\cos^2 45° = 0{,}5$-Anteil durch. Zweimal halbiert ergibt ein Viertel. Ein Polfilter ist eben kein Sieb, das Vorhandenes nur ausdünnt — er ist eine Projektion, und Projektionen können die Richtung dessen, was sie durchlassen, neu festlegen. (Übung 4.3 treibt das auf die Spitze: Mit vielen kleinen Drehschritten kommt fast *alles* Licht um die 90°-Ecke.)

4.6 Die Welle trägt Energie: $\vec{S} = \vec{E} \times \vec{H}$ konkret

Dass „Intensität" mit dem Quadrat der Amplitude geht, ist keine Konvention, sondern Physik — und ihr Träger ist das Kreuzprodukt aus Kapitel 1. Der **Poynting-Vektor**

$$\vec{S} \;=\; \vec{E} \times \vec{H} \qquad \text{mit } \vec{H} = \vec{B}/\mu_0 \text{ (im Vakuum)}$$

ist ein Vektorfeld: An jedem Ort zeigt er, **wohin** Feldenergie strömt, und sein Betrag sagt, **wie viel** — in Watt pro Quadratmeter, der Einheit aus der Vorwort-Tabelle. ($\vec{H}$ ist dabei nur das mit $1/\mu_0$ umskalierte $\vec{B}$ — ein eigenes Symbol bekommt es, weil es in Materie später ein Eigenleben entwickelt.)

Für unsere ebene Welle haben wir oben schon nachgerechnet, dass $\vec{E} \times \vec{B}$ in die Laufrichtung zeigt. Setzen wir Zahlen ein — am selben Punkt wie vorhin ($E = 0{,}866$ V/m, $B = E/c$):

$$S \;=\; \frac{E \cdot B}{\mu_0} \;=\; \frac{E^2}{\mu_0 c} \;=\; \frac{0{,}866^2}{1{,}257 \cdot 10^{-6} \cdot 3{,}0 \cdot 10^8} \;\approx\; 2{,}0 \cdot 10^{-3}\,\frac{\mathrm{W}}{\mathrm{m}^2}.$$

Eine Zahl pro Ort und Moment — und weil $S \propto E^2$ ist, steckt hier das Quadrat aus dem Malus-Gesetz. Wie sieht S entlang der ganzen Welle aus?

```
# von oben: mu_0, c (Importe aus scipy.constants)
x = np.linspace(0, 4, 600)
k = 2*np.pi / 2.0                    # Wellenzahl (λ = 2)
E = np.sin(k*x)                      # E_y in V/m  (E₀ = 1)
S = E**2 / (mu_0 * c)                # S_x = E²/(μ₀c) in W/m²

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7.2, 4.6), sharex=True)
ax1.plot(x, E, color="tab:blue")
ax1.axhline(0, color="k", lw=0.5)
ax1.set_ylabel("E (V/m)")
ax2.plot(x, 1000*S, color="tab:orange")
ax2.axhline(0, color="k", lw=0.5)
ax2.set_ylim(-0.3, 2.9)              # Nulllinie sichtbar im Bild
ax2.set_xlabel("Ort x"); ax2.set_ylabel("S (mW/m²)")
plt.tight_layout(); plt.show()

print(f"kleinster Wert von S entlang der Welle: {np.min(S):.1e} W/m²")
print(f"Zeitmittel: S̄ = E₀²/(2μ₀c) = {1.0/(2*mu_0*c)*1000:.3f} mW/m²")
```

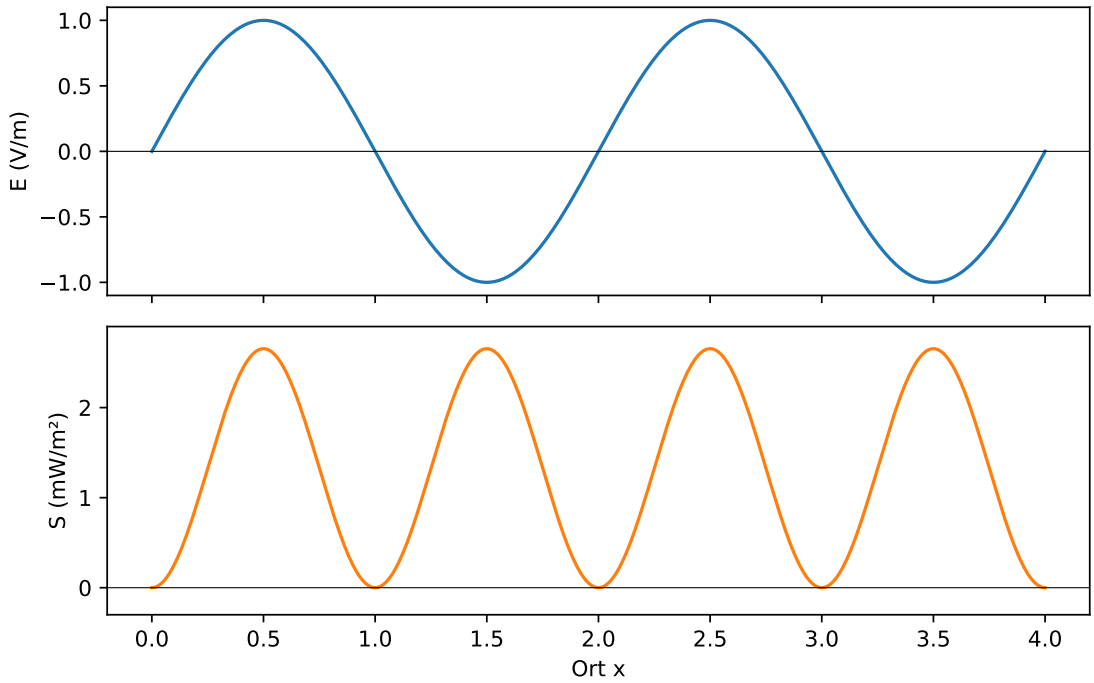


Abbildung 4.5: Oben das Feld E der Welle: Es wechselt das Vorzeichen. Unten der Energiefluss S am selben Ort: Er tut es nie — er pulsiert mit halber Wellenlänge zwischen null und Maximum, bleibt aber überall oberhalb seiner Nulllinie. Energie schwappt nicht zurück.

```
kleinster Wert von S entlang der Welle: 0.0e+00 W/m²
Zeitmittel: S̄ = E₀²/(2μ₀c) = 1.327 mW/m²
```

S wird **nie negativ** — auch wo das Feld gerade „nach unten" zeigt, fließt die Energie vorwärts, denn $\vec{B}$ kippt im selben Moment mit, und das Kreuzprodukt zweier gekippter Pfeile zeigt unverändert nach vorn. Im Zeitmittel bleibt die Hälfte des Spitzenwerts: $\bar{S} = E_0^2/(2\mu_0 c)$.

Diese Formel kann man rückwärts lesen, und das lohnt sich. Die Sonne liefert am Mittag rund $\bar{S} = 1000$ W/m² (der Anker aus dem Vorwort). Welche Feldstärke steckt dahinter?

```
# von oben: mu_0, c (Importe aus scipy.constants)
S_sonne = 1000.0                                    # W/m²
E0 = np.sqrt(2 * mu_0 * c * S_sonne)
print(f"E₀ = sqrt(2·μ₀·c·S̄) = {E0:.0f} V/m")
print(f"B₀ = E₀/c           = {E0/c*1e6:.1f} μT   (Erdmagnetfeld: ≈ 50 μT)")
```

```
E₀ = sqrt(2·μ₀·c·S̄) = 868 V/m
B₀ = E₀/c           = 2.9 μT   (Erdmagnetfeld: ≈ 50 μT)
```

Im hellen Sonnenschein steht an jedem Punkt der Luft ein elektrisches Wechselfeld von fast **900 Volt pro Meter** Spitzenwert — das zugehörige Magnetfeld dagegen ist zwanzigmal *schwächer* als das Erdmagnetfeld. Dieses krasse Ungleichgewicht der Zahlenwerte (Faktor c!) täuscht: Beide Felder tragen exakt gleich viel Energie. Das rechnen wir in Kapitel 18 nach, wenn der Poynting-Vektor sein eigenes Kapitel bekommt.

Das **Kapitel-Programm** `programme/kap04/kap04_polarisation.py` bündelt die Bausteine: Polarisations-Galerie, Malus-Drehkurve gegen die $\cos^2$-Theorie (Abweichung 10^{-16}), Filterketten-Checks und die Sonnenlicht-Rechnung — alles mit `assert`-Schranken.

Merkkasten

- Ebene Welle $E_0 \sin(kx - \omega t)$: k tickt im Raum ($\lambda = 2\pi/k$), ω in der Zeit ($T = 2\pi/\omega$); Tempo $c = \omega/k = \lambda f$.
- Die Sinuskurve ist ein **Diagramm des Feldwerts**, kein Flugweg — Licht läuft schnurgerade.
- $\vec{E} \perp \vec{B} \perp$ Laufrichtung, mit $B = E/c$; **Polarisation** = die Querrichtung, in der $\vec{E}$ schwingt (Strich = linear, Kreis = zirkular).
- Ein **Polfilter projiziert** (Skalarprodukt): $I = I_0 \cos^2 \alpha$ — und weil Projektionen die Polarisation *drehen*, kann ein dritter Filter Licht „zurückholen".
- $\vec{S} = \vec{E} \times \vec{H}$ (W/m²) zeigt bei der ebenen Welle stets vorwärts und pulsiert; Sonnenlicht: $E_0 \approx 870$ V/m.

Roter Faden

Mit diesem Kapitel ist Teil I geschlossen, und seine Werkzeuge haben alle Physik gefunden: Das Skalarprodukt aus **Kapitel 1** steckt im Polfilter, das Kreuzprodukt im Energiefluss; die Quellenfreiheit aus **Kapitel 2/3** hat die Felder in die Querrichtung gezwungen; die Wellengleichung aus **Kapitel 3** lieferte Tempo und Takt. Nach vorn: In **Teil II** (ab Kapitel 5) bauen wir die Wirbelgleichungen als Schleife auf dem Gitter nach — FDTD — und sehen Wellen *entstehen* statt sie anzusetzen. Die Polarisation kehrt in **Kapitel 13** zurück (Brewster-Winkel: warum Polbrillen Spiegelungen schlucken), der Poynting-Vektor in **Kapitel 18**, und die 6000-km-Welle des Stromnetzes in **Kapitel 25**.

Übungen

Ü 4.1 (Verstehen). UKW-Radio sendet um $f = 100$ MHz. Berechne λ — und erkläre damit, warum die klassische Teleskop-Stabantenne ausgezogen etwa 1,5 m lang ist.

> **Musterlösung zu Ü 4.1**
>
> $\lambda = c/f = 3 \cdot 10^8/10^8 = 3$ m:
>
> ```
> # von oben: c (Import aus scipy.constants)
> f_ukw = 100e6
> print(f"λ = c/f = {c/f_ukw:.2f} m   →   λ/2 = {c/f_ukw/2:.2f} m")
> ```
>
> ```
> λ = c/f = 3.00 m   →   λ/2 = 1.50 m
> ```
>
> Antennen arbeiten am besten, wenn ihre Länge zur Welle passt; ein bewährtes Maß ist die **halbe Wellenlänge** — hier 1,5 m. (Warum gerade λ/2 so günstig ist, klärt Teil VI.)

Ü 4.2 (Verstehen). Drehe einen Polfilter um volle 180°. Das Malus-Gesetz sagt $\cos^2 180° = 1$ — alles kommt durch, als hätte man nie gedreht. Aber der durchgelassene Feldvektor zeigt jetzt in die *Gegenrichtung*. Warum sieht man davon nichts?

> **Musterlösung zu Ü 4.2**
>
> Ein um 180° gedrehter E-Vektor ist dieselbe Schwingung mit halbem Perioden-Versatz: Der Pfeil pendelt ohnehin ständig zwischen $+\vec{E}$ und $-\vec{E}$ hin und her. Messbar ist nur die Intensität $\propto E^2$, und $(-E)^2 = E^2$. Die Polarisations-*Richtung* ist darum eine Gerade ohne Pfeilspitze — 0° und 180° sind physikalisch derselbe Filter.

Ü 4.3 (Verändern). Erweitere die Filterkette des Kapitel-Programms: Statt eines 45°-Zwischenfilters nimm N Filter bei $90°/N$, $2 \cdot 90°/N$, … , $90°$. Berechne die durchgelassene Intensität für $N = 2, 4, 8, 16, 90$. Wohin strebt sie — und was bedeutet das anschaulich?

Musterlösung zu Ü 4.3

`E` in einer Schleife durch die gestaffelten Filter schicken:

```
# von oben: durch_filter()   (Polfilter-Abschnitt)
for N in (2, 4, 8, 16, 90):
    E_kette = np.array([1.0, 0.0])
    for i in range(1, N + 1):
        E_kette = durch_filter(E_kette, i * 90 / N)
    print(f"N = {N:2d} Filter:  I = {E_kette @ E_kette:.3f}")
```

```
N =  2 Filter:  I = 0.250
N =  4 Filter:  I = 0.531
N =  8 Filter:  I = 0.733
N = 16 Filter:  I = 0.857
N = 90 Filter:  I = 0.973
```

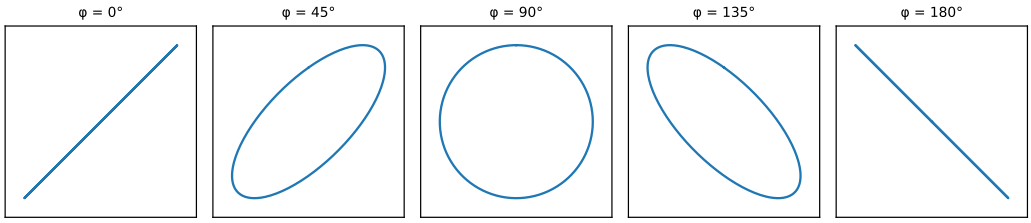
Jeder Schritt dreht nur um den kleinen Winkel $90°/N$ und kostet den Faktor $\cos^2(90°/N)$; insgesamt $I = \cos^{2N}(90°/N)$ — gegen **1** für großes N: Mit genügend sanften Drehschritten lässt sich die Polarisation praktisch verlustfrei um 90° führen. Eine Projektion in vielen kleinen Schritten ist fast eine Drehung.

Ü 4.4 (Übertragen). Baue in der Frontalsicht-Darstellung eine Welle aus $E_y = \sin(\omega t)$ und $E_z = \sin(\omega t + \varphi)$ und zeichne die Spur der Pfeilspitze für $\varphi = 0°, 45°, 90°, 135°, 180°$. Beschreibe, was zwischen Strich und Kreis passiert.

Musterlösung zu Ü 4.4

Fünf `plot(np.sin(wt), np.sin(wt + φ))`-Aufrufe genügen:

```
wt = np.linspace(0, 2*np.pi, 300)
fig, achsen = plt.subplots(1, 5, figsize=(9.6, 2.2))
for ax, phi in zip(achsen, (0, 45, 90, 135, 180)):
    ax.plot(np.sin(wt), np.sin(wt + np.radians(phi)), lw=1.5)
    ax.set_xlim(-1.25, 1.25); ax.set_ylim(-1.25, 1.25)
    ax.set_title(f"φ = {phi}°", fontsize=9)
    ax.set_aspect("equal"); ax.set_xticks([]); ax.set_yticks([])
plt.tight_layout(); plt.show()
```



$\varphi = 0°$: Strich unter 45° (beide Komponenten im Gleichtakt). $\varphi = 90°$: Kreis. Dazwischen $(45°, 135°)$: **Ellipsen**, zunehmend bauchig, mit gekippter Hauptachse — die allgemeine, „elliptische" Polarisation, deren Grenzfälle Strich und Kreis sind. $\varphi = 180°$: wieder ein Strich, nun unter $-45°$.

Das Kleingedruckte

- **Das Lattenzaun-Bild** (Polfilter als Schlitzgitter, durch das nur „passend schwingende" Wellen schlüpfen) trägt für die Mechanik eines Seils — für Licht an realen Drahtgitter-Polarisatoren ist es sogar **verkehrt herum**: Durchgelassen wird die Schwingung *quer* zu den Drähten, denn längs der Drähte kann das Feld Ströme treiben und wird absorbiert.
- $\vec{H}$ **vs.** $\vec{B}$**:** Im Vakuum unterscheiden sich beide nur um den festen Faktor μ_0 — die Unterscheidung wird erst in Materie inhaltlich (Magnetisierung), ab Teil IV/VII.
- **„Intensität"** meint das Zeitmittel von $|\vec{S}|$ — daher der Faktor $\frac{1}{2}$ in $\bar{S} = E_0^2/(2\mu_0 c)$, das Mittel von $\sin^2$.
- **Zirkular gibt es zweimal:** links- und rechtsdrehend, je nach Vorzeichen des Versatzes. Der Drehsinn wird wichtig, wo Licht auf chirale Strukturen trifft (Teil VII) — und bei Satellitenfunk, der zirkulare Polarisation nutzt, weil sie Drehungen der Antenne verzeiht.

5 Diskretisieren: Maxwell im Daumenkino

Ein Daumenkino erzeugt Bewegung aus Standbildern: Jede Seite unterscheidet sich ein kleines bisschen von der vorigen, und beim Durchblättern läuft der Film. Eine Simulation ist ein Daumenkino, das sich **selbst zeichnet**: Aus Bild n berechnet sie Bild $n + 1$ — immer wieder, tausendfach.

In Kapitel 3 haben wir geprüft, ob ein *fertiger* Puls die Maxwell-Gleichungen erfüllt — den Film hatten wir von Hand gezeichnet, die Gleichungen durften nur nicken. Jetzt drehen wir den Spieß um: Die Gleichungen bekommen nur das **erste Bild** (die Felder jetzt) und müssen alle weiteren selbst zeichnen. Das klingt nach einer Kleinigkeit — schließlich sagen die Gleichungen ja gerade, wie sich die Felder ändern. Aber der erste, völlig naheliegende Bauplan scheitert spektakulär, und die Reparatur ist eine der elegantesten Ideen der numerischen Physik. Mit ihr beginnt der Simulationsmotor dieses Buchs: **FDTD** — Finite Differenzen im Zeitbereich.

Lernziele

Nach diesem Kapitel kannst du …

1. … die zwei Wirbelgleichungen in **Update-Regeln** umstellen („morgen = heute + Δt mal Änderungsrate”) und eine solche Regel an Zahlen von Hand durchrechnen,
2. … vorführen, warum der naive Gleichschritt (Euler) für Wellen **explodiert** — und warum kleinere Zeitschritte ihn nicht retten,
3. … das **Leapfrog**-Prinzip erklären: E zu ganzen, B zu halben Zeitschritten — und warum das eine zentrale Differenz ist,
4. … das **Yee-Versatz-Argument** wiedergeben: B sitzt räumlich zwischen den E-Punkten,
5. … den Zehn-Zeilen-Kern einer 1D-Feldsimulation lesen — und ihr Ergebnis gegen die Lichtgeschwindigkeit messen.

5.1 Von der Gleichung zur Update-Regel

Ausgangspunkt ist das 1D-Gleichungspaar aus Kapitel 3, diesmal nach den **Zeitableitungen** aufgelöst:

$$\frac{\partial B}{\partial t} = -\frac{\partial E}{\partial x}, \qquad \frac{\partial E}{\partial t} = -c^2 \frac{\partial B}{\partial x}.$$

So herum gelesen sind das keine Bedingungen mehr, sondern **Kochrezepte**: Das räumliche Gefälle des einen Feldes sagt, wie schnell sich das andere ändert. Und „wie schnell es sich ändert" heißt im Daumenkino: was im nächsten Bild steht. Mit der Nachbar-Differenz aus Kapitel 2 — diesmal in der Zeit — wird aus jeder Gleichung eine **Update-Regel**:

$$E_{\text{neu}} \;=\; E_{\text{alt}} \;+\; \Delta t \cdot \left(-c^2 \frac{\partial B}{\partial x}\right),$$

und genauso für B. Eine Handrechnung an einem konkreten Punkt, mit Gitterabstand $\Delta x = 1$ cm und Zeitschritt $\Delta t = \Delta x / c \approx 33{,}4$ ps (warum genau dieser Zeitschritt eine ausgezeichnete Wahl ist, klärt Kapitel 8):

```
import numpy as np
from scipy.constants import c

DX = 0.01                   # m
DT = DX / c                 # s
print(f"Zeitschritt Δt = {DT*1e12:.2f} ps\n")

E_alt = 100.0               # V/m an unserem Punkt
B_rechts, B_links = 0.2e-6, 0.5e-6        # T, die beiden B-Nachbarn

dB_dx = (B_rechts - B_links) / DX
print(f"∂B/∂x ≈ (0,2 μT − 0,5 μT) / 1 cm = {dB_dx*1e6:+.0f} μT/m")

E_neu = E_alt + DT * (-c**2 * dB_dx)
print(f"E_neu = {E_alt} + Δt·(−c²·∂B/∂x) = {E_neu:.0f} V/m")
```

```
Zeitschritt Δt = 33.36 ps

∂B/∂x ≈ (0,2 μT − 0,5 μT) / 1 cm = -30 μT/m
E_neu = 100.0 + Δt·(−c²·∂B/∂x) = 190 V/m
```

Zwei Nachbarwerte angeschaut, eine Multiplikation — und der Punkt weiß, was er im nächsten Bild anzeigt. Das ganze Verfahren besteht aus nichts anderem, millionenfach wiederholt. Es bleibt nur eine Designfrage offen, und sie hat es in sich: **Wo und wann genau** sollen E und B auf dem Gitter wohnen?

5.2 Der naheliegende Plan — und sein Scheitern

Der erste Entwurf liegt auf der Hand: beide Felder an denselben Gitterpunkten, beide zu denselben Zeiten. Pro Zeitschritt berechnen wir beide räumlichen Ableitungen (zentral, über $2\Delta x$) aus den *alten* Werten und schreiben beide Felder fort. Sauber, symmetrisch, direkt aus dem Lehrbuch der Schrittverfahren — es ist das klassische **Euler-Verfahren**, hier „Gleichschritt" genannt, weil beide Felder im selben Takt marschieren.

Ob dieser Plan taugt, soll ein Experiment entscheiden. Damit der Code gleich nur noch *vorführt*, was hier beschlossen wird, zuerst der vollständige Versuchsaufbau:

Die Bühne. Eine 10 m lange Vakuum-Strecke, zerlegt in $N = 1001$ Gitterpunkte im Abstand $\Delta x = 1$ cm. Der Zeitschritt ist das $\Delta t = \Delta x/c \approx 33$ ps aus der Handrechnung oben; gerechnet werden 220 Schritte, zusammen gut 7 ns.

Die Anfangsbedingung — das erste Daumenkino-Bild: ein ruhender **Gauß-Hügel im E-Feld** bei $x = 5$ m (charakteristische Breite 0,5 m) und $B = 0$ überall. Das ist der einfachste denkbare Start: ein einziges glattes Feldgebilde, keine Quelle, kein Antrieb. Was dieser Anfangszustand *physikalisch* anstellt, heben wir uns für den Schluss des Kapitels auf — für den Test zählt nur, dass beide Verfahren dasselbe glatte Feld weiterrechnen müssen.

Die Randbedingung. Irgendetwas muss an den Enden der Strecke passieren, denn die äußersten Punkte haben nur einen Nachbarn. Wir wählen die einfachste Möglichkeit: Die Update-Zeilen fassen nur die **inneren** Punkte an (im Code am Slice `[1:-1]` zu erkennen), die äußersten E-Werte werden nie verändert und bleiben für immer null. Ein festgenagelter E-Wert wirkt wie eine ideale Metallwand, also wie ein Spiegel (Kapitel 6 führt das als „PEC" richtig ein und spielt damit). Hier sind die Wände reine Kulisse: In 220 Schritten kommt eine Welle höchstens 2,2 m weit, von $x = 5$ m aus erreicht also nichts die Ränder bei 0 und 10 m.

Die Messgröße und das Erfolgskriterium. Wir verfolgen nicht die Feldform, sondern eine einzige Zahl pro Zeitschritt: die **Gesamtenergie** des

Felds. Felder tragen Energie (Kapitel 4); ihre Dichte ist $\frac{\varepsilon_0}{2}E^2$ für das elektrische und $\frac{1}{2\mu_0}B^2$ für das magnetische Feld (beide in J/m³ — vertieft in Kapitel 18). Die Funktion `energie()` summiert beide Anteile über die ganze Strecke. Der Grund für diese Wahl: Energie ist ein **unbestechlicher Schiedsrichter**. Im verlustfreien Vakuum zwischen verlustfreien Spiegeln darf die Gesamtenergie sich nicht ändern — egal, was die Felder im Detail treiben, und ohne dass wir die richtige Lösung kennen müssten. Aufgetragen wird sie relativ zum Anfangswert; die Solllinie ist also eine Waagerechte bei 1, und ein Verfahren, dessen Kurve steigt, erschafft Energie aus dem Nichts — disqualifiziert.

Die Kandidaten. Der Gleichschritt tritt nicht allein an: Als Vergleichsmaßstab läuft eine zweite Variante mit, die E und B räumlich wie zeitlich **versetzt** anordnet (im Code `leapfrog_schritt`; ihr B-Array ist einen Eintrag kürzer, weil die B-Punkte *zwischen* den E-Punkten sitzen). Wie dieser Versatz funktioniert und warum, ist das Thema des nächsten Abschnitts — hier darf die Variante nur schon einmal zeigen, was sie kann.

Vorhersage-Punkt

Bevor du die Zelle liest: Beide Verfahren starten mit demselben Gauß-Hügel, und die Energie-Kurven beider werden über 220 Schritte aufgezeichnet (logarithmische Achse!). Die Solllinie kennst du: konstant bei 1. Was erwartest du für die beiden Kurven — und falls eine abweicht: nach oben oder unten, allmählich oder plötzlich?

```
# von oben: DX, DT; c (Import)   (Handrechnungs-Zelle)
from scipy.constants import epsilon_0, mu_0

N = 1001
X = np.arange(N) * DX     # np.arange(n): die Zahlen 0, 1, …, n-1


def anfangspuls():
    e = np.exp(-((X - 5.0) / 0.5)**2)      # Gauß-Puls im E-Feld
    return e, np.zeros(N - 1)              # B: zwischen den E-Punkten


def energie(e, b):
    # np.sum: summiert alle Array-Elemente zu einer Zahl auf
    return 0.5 * (epsilon_0 * np.sum(e**2) + np.sum(b**2) / mu_0) * DX
```

```python
def euler_schritt(e, b):
    """Gleichschritt: beide Felder am selben Ort, beide aus ALTEN Werten."""
    e_neu, b_neu = e.copy(), b.copy()      # .copy(): echte Kopie statt Verweis
    e_neu[1:-1] = e[1:-1] - c**2 * DT * (b[2:] - b[:-2]) / (2*DX)
    b_neu[1:-1] = b[1:-1] - DT * (e[2:] - e[:-2]) / (2*DX)
    return e_neu, b_neu


def leapfrog_schritt(e, b):
    """Versetzt: B zwischen den E-Punkten, B-Update zwischen den E-Updates."""
    b -= DT / DX * np.diff(e)              # np.diff: e[i+1] - e[i], elementweise
    e[1:-1] -= c**2 * DT / DX * np.diff(b)
    return e, b


e_eu, _ = anfangspuls(); b_eu = np.zeros(N)       # Euler: B am selben Ort
e_lf, b_lf = anfangspuls()
e0 = energie(*anfangspuls())

verlauf_eu, verlauf_lf = [], []
for n in range(220):
    e_eu, b_eu = euler_schritt(e_eu, b_eu)
    e_lf, b_lf = leapfrog_schritt(e_lf, b_lf)
    verlauf_eu.append(energie(e_eu, b_eu[:-1]) / e0)
    verlauf_lf.append(energie(e_lf, b_lf) / e0)

import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(6.4, 3.4))
ax.semilogy(verlauf_eu, label="Euler-Gleichschritt")
ax.semilogy(verlauf_lf, label="Leapfrog (versetzt)")
ax.set_xlabel("Zeitschritt"); ax.set_ylabel("Energie / Anfangsenergie")
ax.legend(); plt.tight_layout(); plt.show()

print(f"Energie nach 220 Schritten — Euler: {verlauf_eu[-1]:.1e},"
      f"  Leapfrog: {verlauf_lf[-1]:.6f}")
```

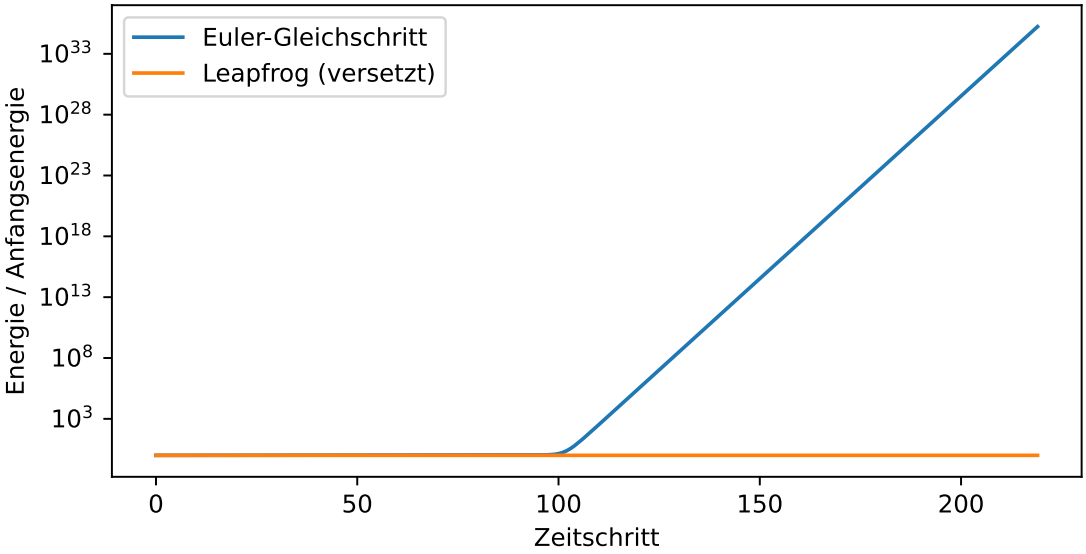


Abbildung 5.1: Gesamtenergie über 220 Zeitschritte, logarithmisch. Der Euler-Gleichschritt explodiert um mehr als 30 Größenordnungen; der Leapfrog auf versetzten Stützstellen hält die Energie konstant. Der Knick bei Schritt ~100 ist kein Zufall — Übung 5.2 berechnet ihn.

```
Energie nach 220 Schritten — Euler: 1.7e+35,  Leapfrog: 1.000100
```

Der Gleichschritt sprengt jede Skala: **35 Größenordnungen** Energiegewinn aus dem Nichts. Das ist kein Programmierfehler und keine Frage der Genauigkeit — das Verfahren selbst ist für Wellen unbrauchbar.

⚠ Naheliegende Vermutung

Vermutung: *„Dann eben kleinere Zeitschritte — kleine Schritte, kleine Fehler, irgendwann stimmt es."*
Warum sie naheliegt: Bei allem bisher in diesem Buch wurde der Fehler kleiner, wenn die Schrittweite schrumpfte — das war geradezu unser Wahrheitskriterium (Kapitel 2).
Was stattdessen stimmt: Der Euler-Gleichschritt **verstärkt jede Schwingung bei jedem Schritt um einen Faktor größer als eins** — egal wie klein Δt ist. Kleinere Schritte verkleinern zwar den Faktor, aber dafür braucht dieselbe Simulationszeit mehr Schritte: Das Produkt bleibt exponentielles Wachstum, nur langsamer. Der Unterschied zwischen „ungenau" und **„instabil"** ist der zentrale Begriff dieses Kapitels: Ungenauigkeit schrumpft mit feinerem Gitter — Instabilität nicht. Sie

> ist ein Konstruktionsfehler, und man behebt sie nicht mit Geduld, sondern mit einem besseren Bauplan.

5.3 Der bessere Bauplan: doppelt versetzt

Die orangefarbene Kurve im Energie-Plot gehört zu diesem besseren Bauplan, und sein Kern ist ein einziger Gedanke: **Alle Differenzen sollen zentral sein** — auch die in der Zeit. Der Euler-Schritt schätzt die Zeitableitung *einseitig* (aus „jetzt" in Richtung „gleich"), und einseitige Differenzen waren schon in Kapitel 2 die schlechtere Wahl. Zentral wäre: Werte *vor und nach* dem Moment der Auswertung. Aber wie soll das gehen, ohne die Zukunft zu kennen?

Mit Versatz. Man lässt die beiden Felder **abwechselnd** springen:

- E lebt zu den **ganzen** Zeitschritten $0,\ 1,\ 2,\ \ldots$
- B lebt zu den **halben** Zeitschritten $\frac{1}{2},\ \frac{3}{2},\ \frac{5}{2},\ \ldots$

Beim Update von $B^{1/2}$ auf $B^{3/2}$ liegt das benutzte E^1 **zeitlich genau in der Mitte** — die Differenz ist zentral, ganz ohne Hellseherei. Und beim Update von E^1 auf E^2 liegt $B^{3/2}$ wieder genau in der Mitte. Die beiden Felder überspringen einander wie beim Bockspringen — daher der Name **Leapfrog**.

Derselbe Trick noch einmal im Raum: Die E-Update-Regel braucht $\partial B/\partial x$ *am Ort des E-Punktes*. Sitzen die B-Punkte **zwischen** den E-Punkten (um eine halbe Zelle versetzt), dann ist $(B_{i+1/2} - B_{i-1/2})/\Delta x$ eine zentrale Differenz direkt am E-Punkt i — mit Spannweite Δx statt $2\Delta x$, also viermal genauer als der Gleichschritt-Stencil. Dieses doppelt versetzte Gitter heißt nach seinem Erfinder **Yee-Gitter** (Kane Yee, 1966), und es ist bis heute das Herz jedes FDTD-Programms:

Im Code ist der ganze Bauplan kürzer als seine Erklärung — das ist der Kern, den du in der Energie-Zelle oben schon als `leapfrog_schritt` benutzt hast:

```
b -= DT / DX * np.diff(e)                  # B springt (halber Takt)
e[1:-1] -= c**2 * DT / DX * np.diff(b)     # dann springt E (ganzer Takt)
```

Diese zwei Zeilen sind die dichtesten des ganzen Buchs — in ihnen stecken die NumPy-Mechanik, das versetzte Gitter und die Randbehandlung gleichzeitig. Deshalb falten wir sie einmal komplett auseinander; dieselbe Rechnung, nur mit benannten Zwischenschritten:

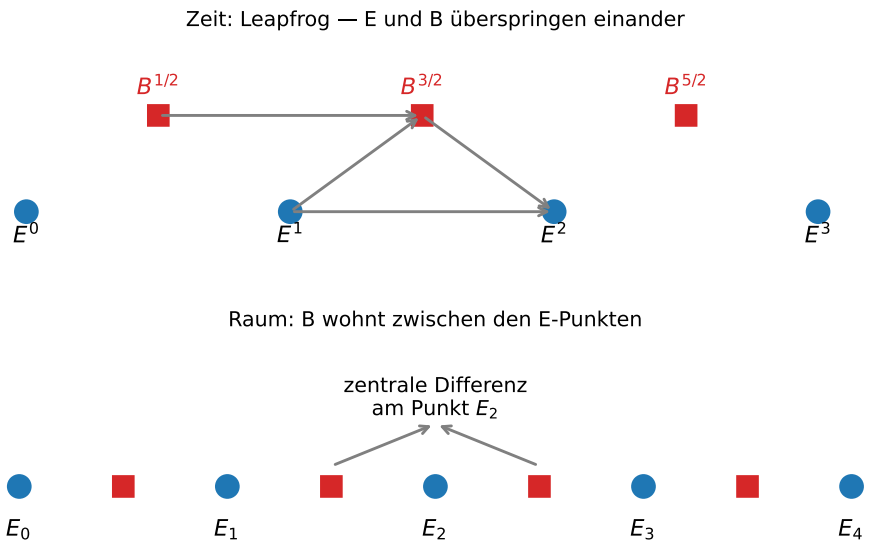


Abbildung 5.2: Das Yee-Schema. Oben der Zeitversatz: E (blau) lebt zu ganzen, B (rot) zu halben Zeitschritten; jeder Pfeil-Dreierpack ist ein Update, der benutzte Querwert liegt immer zeitlich mittig. Unten der Raumversatz: B sitzt zwischen den E-Punkten, die Differenz der beiden B-Nachbarn ist eine zentrale Differenz direkt am E-Punkt.

```
delta_e = np.diff(e)        # Nachbar-Differenzen von e: eine pro Lücke
b = b - DT / DX * delta_e   # B-Update: elementweise, gleiche Länge

delta_b = np.diff(b)        # Nachbar-Differenzen von b
e_innen = e[1:-1]           # die inneren E-Punkte (ohne Rand)
e[1:-1] = e_innen - c**2 * DT / DX * delta_b
```

Was `np.diff` mechanisch tut: nichts als Nachbar-Differenzen. `np.diff([a, b, c, d])` ergibt `[b-a, c-b, d-c]` — aus vier Werten werden **drei** Differenzen, denn es gibt eine pro *Lücke*, nicht eine pro Eintrag; das Ergebnis ist immer um eins kürzer als die Eingabe. Zwei Dinge tut `np.diff` ausdrücklich *nicht*: Es teilt nicht durch Δx (dieser Faktor steckt bei uns im Vorfaktor `DT / DX`), und es ist **kein zentraler Differenzenquotient** — der hieße `np.gradient` (Kapitel 2), behielte die Länge bei und verrechnete übernächste Nachbarn.

Warum die simple Nachbar-Differenz hier trotzdem *zentral* ist, zeigt die Buchhaltung der Plätze. Trag einmal auf, wer wo wohnt und wozu welche Differenz gehört:

```
Ort:          0       Δx      2Δx      3Δx      4Δx
e:           e[0]    e[1]    e[2]     e[3]     e[4]      (N Werte)
b:               b[0]    b[1]     b[2]     b[3]          (N-1 Werte)
np.diff(b):          d[0]     d[1]     d[2]              (N-2 Werte)
gehört zu:           e[1]     e[2]     e[3]
```

Die Differenz `d[0] = b[1] - b[0]` benutzt zwei B-Werte, die **symmetrisch** um den Punkt `e[1]` liegen — je eine halbe Zelle links und rechts. Vom E-Punkt aus gesehen *ist* das die zentrale Differenz aus Abbildung 5.2; die Zentrierung kommt nicht von der Funktion `diff`, sondern von der **Lage der B-Punkte**. Und die Längen gehen wie von selbst auf: `e` hat N Einträge, das dazwischen wohnende `b` hat $N-1$, dessen Differenzen-Array `np.diff(b)` hat $N-2$ — genau einen Wert für jeden *inneren* E-Punkt.

Damit erklärt sich auch das `[1:-1]` (der Slice „alles außer dem ersten und letzten Eintrag", Kapitel 2): Links steht ein Ausschnitt mit $N-2$ Plätzen, rechts ein Ergebnis mit $N-2$ Werten — die Zuweisung schreibt sie elementweise in die inneren Positionen `e[1]` bis `e[N-2]`. Die beiden **Randpunkte rührt die Zeile bewusst nicht an**, und das ist kein NumPy-Zufall, sondern Physik: `e[0]` hat keinen B-Nachbarn links, `e[-1]` keinen rechts — die zentrale Differenz ist dort gar nicht bildbar. Randpunkte brauchen eine eigene Regel (eine *Randbedingung*): In diesem Kapitel bleiben sie einfach null (festgenagelt = Spiegelwand), in Kapitel 6 lernen sie Wellen hinauszulassen. Merke: **Maxwell regiert innen, der Rand wird extra regiert.**

Das Gitter und die Mathematik greifen ineinander wie Zahnräder. Mehr ist FDTD im Kern nicht; alles Weitere (Quellen, Materialien, Ränder) sind Anbauten an diese zwei Zeilen — sie kommen in Kapitel 6.

5.4 Die erste lebende Welle

Zeit für den Moment, auf den Teil I hingearbeitet hat: Wir geben dem Daumenkino ein erstes Bild und lassen die Maxwell-Gleichungen zeichnen. Die Bühne ist dieselbe wie beim Energie-Test (10 m, $\Delta x = 1$ cm, Spiegelränder, Gauß-Hügel bei 5 m, $B = 0$) — nur das Messprogramm ist neu: Wir fotografieren das E-Feld bei Schritt 0, 150 und 300, und ab Schritt 150 notieren wir zusätzlich in jedem Schritt, **wo** das Maximum der rechten Bildhälfte sitzt. Durch diese Ort-über-Zeit-Punkte legen wir am Ende eine Gerade: Ihre Steigung ist die gemessene Ausbreitungsgeschwindigkeit, und die vergleichen wir mit c.

> Vorhersage-Punkt
>
> **Bevor du die Zelle liest:** Das erste Bild ist ein Gauß-Berg im E-Feld — und **B ist überall null**. Es gibt also keine Information darüber, in welche Richtung dieser Puls laufen „soll" (erinnere dich an Kapitel 4: Eine laufende Welle hat $B = E/c$, hier fehlt das B komplett). Was wird passieren? Läuft der Puls nach rechts, nach links, bleibt er stehen, zerfließt er? Lege dich fest.

```
# von oben: anfangspuls(), leapfrog_schritt(), X, N, DT
e, b = anfangspuls()
schnappschuesse = {0: e.copy()}
orte, zeiten = [], []

for n in range(1, 351):
    e, b = leapfrog_schritt(e, b)
    if n in (150, 300):
        schnappschuesse[n] = e.copy()
    if n >= 150:                        # rechter Puls, sauber getrennt
        # np.argmax: der INDEX des größten Werts (nicht der Wert selbst)
        orte.append(X[N//2 + np.argmax(e[N//2:])])
        zeiten.append(n * DT)

fig, ax = plt.subplots(figsize=(7.0, 3.2))
for n, farbe in ((0, "gray"), (150, "tab:blue"), (300, "tab:red")):
    ax.plot(X, schnappschuesse[n], color=farbe, label=f"t = {n*DT*1e9:.1f} ns")
```

```
ax.set_xlabel("x (m)"); ax.set_ylabel("E (V/m)")
ax.legend(); plt.tight_layout(); plt.show()

# np.polyfit(x, y, 1): Gerade durch die Messpunkte; [0] ist die Steigung
tempo = np.polyfit(zeiten, orte, 1)[0]
print(f"gemessenes Tempo: {tempo:,.0f} m/s   (c = {c:,.0f} m/s,"
      f" Abweichung {abs(tempo-c)/c:.0e})")
```

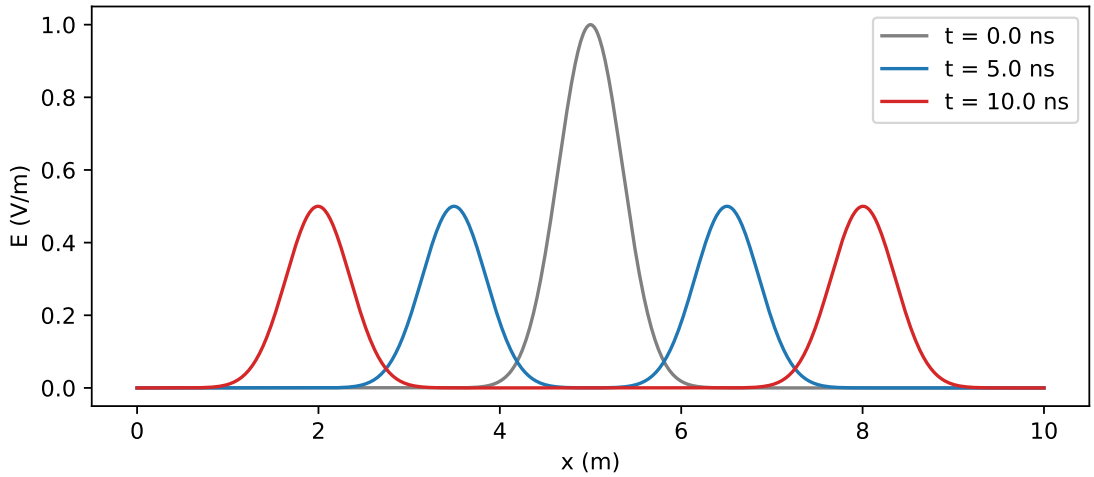


Abbildung 5.3: Ein E-Puls ohne Magnetfeld kann sich nicht entscheiden — und nimmt beide Richtungen: Er zerfällt in zwei Pulse halber Höhe, die mit Lichtgeschwindigkeit auseinanderlaufen.

```
gemessenes Tempo: 299,792,458 m/s   (c = 299,792,458 m/s, Abweichung 4e-16)
```

Der Puls **teilt sich**: zwei Kopien halber Höhe, eine nach links, eine nach rechts. Niemand hat dem Programm das beigebracht — es kennt nur die zwei Update-Zeilen. Die Physik dahinter steckt im Kapitel-4-Befund: Ein nach rechts laufender Puls braucht $B = +E/c$, ein nach links laufender $B = -E/c$. Unser Start (E da, $B = 0$) ist exakt die **Summe aus je einer halben Portion von beidem** — und dank Superposition läuft jede Hälfte ungestört los. Wer einen Puls will, der *nur* nach rechts läuft, muss das passende B mitliefern: Übung 5.4.

Und das Tempo? Die Maximum-Verfolgung liefert die Lichtgeschwindigkeit **auf Maschinengenauigkeit** — nicht ungefähr, sondern exakt. Das ist kein Standard, sondern ein Sonderfall: Beim gewählten Zeitschritt $\Delta t = \Delta x/c$ rückt die Welle pro Schritt exakt eine Zelle weiter, und das Schema transportiert sie fehlerfrei („magischer Zeitschritt" — was bei anderen Zeitschritten

passiert und warum größere sofort explodieren, ist das Thema von Kapitel 8).

Das **Kapitel-Programm** `programme/kap05/kap05_leapfrog_1d.py` fasst alles zusammen — Energievergleich, Pulsteilung, Tempomessung — mit `assert`-Schranken (Euler-Energie > 10^3, Leapfrog-Energie auf 1 % konstant, Tempo auf 10^{-6} genau).

Merkkasten

- Eine Simulation ist ein selbstzeichnendes Daumenkino: **Update-Regel = Gleichung, nach der Zeitableitung aufgelöst**, plus Nachbar-Differenzen.
- **Instabil ≠ ungenau**: Der Euler-Gleichschritt verstärkt Wellen bei *jedem* Δt exponentiell — kleinere Schritte retten ihn nicht.
- **Leapfrog**: E zu ganzen, B zu halben Zeitschritten → die Zeitdifferenz wird zentral, das Verfahren stabil.
- **Yee-Gitter**: B wohnt zwischen den E-Punkten → die Raumdifferenz ist zentral mit Spannweite Δx.
- Der Kern sind **zwei Zeilen NumPy**; ein E-Puls ohne B teilt sich in zwei Hälften ($B = \pm E/c$), die exakt mit c laufen.

Roter Faden

Hier sind drei Fäden zusammengelaufen: die zentralen Differenzen und das „Fehler schrumpft mit Δx^2"-Kriterium aus **Kapitel 2**, die zwei Wirbelgleichungen aus **Kapitel 3** und der $B = E/c$-Zusammenhang aus **Kapitel 4**, der die Pulsteilung erklärt. Nach vorn: **Kapitel 6** baut um den Zehn-Zeilen-Kern ein vollständiges Labor — Quellen, Verluste, Spiegel, Materialwechsel. **Kapitel 7** zeigt, dass unsere Update-Schleife heimlich eine Matrix-Multiplikation ist. Und **Kapitel 8** beantwortet die hier offen gebliebene Frage, welcher Zeitschritt erlaubt ist — und was der „magische" so magisch macht.

Übungen

Ü 5.1 (Verstehen). Die Leapfrog-Zeitdifferenz benutzt nur zwei Werte, $B^{n+1/2}$ und $B^{n-1/2}$ — trotzdem nennt das Kapitel sie „zentral". Um welchen Zeitpunkt ist sie zentriert, und welcher Stelle im Update entspricht das?

Musterlösung zu Ü 5.1

Sie ist um den **ganzen** Zeitschritt n zentriert: $B^{n+1/2}$ liegt eine halbe Schrittweite danach, $B^{n-1/2}$ eine halbe davor — genau wie bei der zentralen Differenz aus Kapitel 2, nur dass die „Nachbarn" hier in der Zeit liegen. Und n ist exakt der Moment, an dem das E-Feld lebt, dessen Update diese Differenz speist. Zentral heißt also nicht „drei Punkte", sondern „symmetrisch um den Auswertepunkt".

Ü 5.2 (Verstehen). Im Energie-Plot bleibt die Euler-Kurve rund 100 Schritte lang scheinbar flach und schießt dann hoch. Die schnellstwachsende Störung wird pro Schritt um den Faktor $\sqrt{2}$ verstärkt und startet als Rundungsrauschen bei etwa 10^{-16}. Berechne, nach wie vielen Schritten sie die Größenordnung 1 erreicht — passt das zum Knick?

Musterlösung zu Ü 5.2

Gesucht ist n mit $10^{-16} \cdot (\sqrt{2})^n = 1$:

```
n_knick = 16 / np.log10(np.sqrt(2))
print(f"n = 16 / log10(√2) ≈ {n_knick:.0f} Schritte")
```

```
n = 16 / log10(√2) ≈ 106 Schritte
```

Etwa **106 Schritte** — genau dort knickt die Kurve hoch. Die Explosion war also von Anfang an da, nur unsichtbar klein: Der glatte Puls selbst wächst kaum, aber das unvermeidliche Rundungsrauschen der Gleitkommazahlen enthält die am schnellsten wachsenden (kurzwelligsten) Störungen, und $\sqrt{2}^n$ holt 16 Größenordnungen in ~100 Schritten auf. Instabilität kündigt sich nicht an — sie war nur noch nicht fällig.

Ü 5.3 (Verändern). Halbiere im Kapitel-Programm den Zeitschritt ($\Delta t = \Delta x/(2c)$, also $S = 0{,}5$) — bei gleicher Simulationsdauer (doppelt so viele Schritte). Sage vorher: Rettet das den Euler-Gleichschritt? Und bleibt der Leapfrog-Puls noch „magisch" perfekt?

Musterlösung zu Ü 5.3

```
# von oben: DX, N, anfangspuls(), energie(), e0; c (Import)
DT2 = DX / (2 * c)                          # S = 0,5

e_eu, _ = anfangspuls(); b_eu = np.zeros(N)
e_lf, b_lf = anfangspuls()
for n in range(440):                        # gleiche Dauer: 2 × 220 Schritte
    e_neu, b_neu = e_eu.copy(), b_eu.copy()
    e_neu[1:-1] = e_eu[1:-1] - c**2*DT2*(b_eu[2:] - b_eu[:-2])/(2*DX)
    b_neu[1:-1] = b_eu[1:-1] - DT2*(e_eu[2:] - e_eu[:-2])/(2*DX)
    e_eu, b_eu = e_neu, b_neu
    b_lf -= DT2/DX * np.diff(e_lf)
    e_lf[1:-1] -= c**2*DT2/DX * np.diff(b_lf)

print(f"Euler   : Energie ×{energie(e_eu, b_eu[:-1])/e0:.1e}")
print(f"Leapfrog: Energie ×{energie(e_lf, b_lf)/e0:.6f},"
      f"  Amplitude rechts: {np.max(e_lf[N//2:]):.4f}")
```

```
Euler   : Energie ×4.7e+11
Leapfrog: Energie ×1.000025,  Amplitude rechts: 0.5000
```

Euler explodiert **weiterhin** (langsamer pro Schritt, aber es sind doppelt so viele — Instabilität bleibt Instabilität). Der Leapfrog bleibt stabil und energietreu, aber die Pulsform ist nicht mehr bitgenau perfekt: Die Amplitude weicht minimal von 0,5 ab, weil bei $S < 1$ verschiedene Wellenlängen *geringfügig* verschieden schnell laufen (numerische Dispersion — das Thema von Kapitel 8). Stabil und brauchbar ist $S = 0{,}5$ trotzdem.

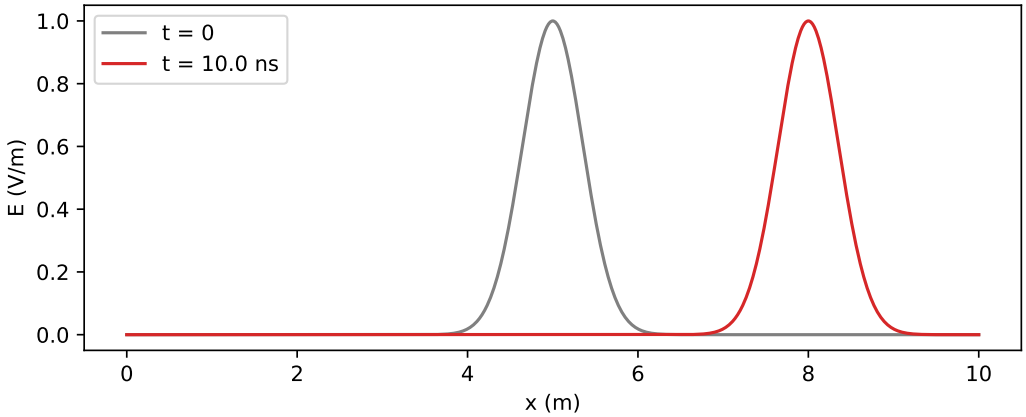
Ü 5.4 (Übertragen). Baue einen Anfangszustand, dessen Puls **nur nach rechts** läuft. (Hinweis aus Kapitel 4: Eine rechtslaufende Welle hat $B = +E/c$ — und denke daran, dass die B-Punkte eine halbe Zelle versetzt sitzen.)

Musterlösung zu Ü 5.4

Das Magnetfeld wird mit derselben Glockenform belegt, ausgewertet an den **B-Positionen** (halbe Zelle versetzt) und durch c geteilt:

```
# von oben: X, DX, DT; c (Import)
x_b = X[:-1] + DX/2                          # die Orte der B-Punkte
e = np.exp(-((X - 5.0)/0.5)**2)
b = np.exp(-((x_b + c*DT/2 - 5.0)/0.5)**2) / c   # B = E/c, rechtslaufend

fig, ax = plt.subplots(figsize=(7.0, 3.0))
ax.plot(X, e, "gray", label="t = 0")
for n in range(1, 301):
    b -= DT/DX * np.diff(e)
    e[1:-1] -= c**2 * DT/DX * np.diff(b)
ax.plot(X, e, "tab:red", label=f"t = {300*DT*1e9:.1f} ns")
ax.set_xlabel("x (m)"); ax.set_ylabel("E (V/m)")
ax.legend(); plt.tight_layout(); plt.show()
print(f"Amplitude: {np.max(e):.4f}   Ort des Maximums: {X[np.argmax(e)]:.2f} m")
```




```
Amplitude: 1.0000   Ort des Maximums: 8.00 m
```

Ein einziger Puls voller Höhe, der bei $x = 8$ m angekommen ist — nichts läuft nach links. (Der kleine Zusatzterm $c\Delta t/2$ im B-Argument berücksichtigt, dass das B-Feld im Leapfrog eine halbe *Zeit*-Stufe vor dem Start lebt; ohne ihn bliebe ein winziger Linksläufer übrig.)

Das Kleingedruckte

- **Die Ränder** unseres Gitters halten $E = 0$ fest — das ist physikalisch eine ideale Metallwand. Sobald ein Puls sie erreicht, wird er reflektiert; in diesem Kapitel haben wir vorher gestoppt. Ränder, die Wellen schlucken statt spiegeln, sind ein eigenes Thema (Kapitel 6 und 9).

- **„B = 0 am Start"** heißt im Leapfrog genau genommen $B(-\Delta t/2) = 0$ — eine halbe Zeitstufe *vor* dem ersten Bild. Dieser Versatz verschiebt die geteilten Pulse um eine halbe Zelle; sichtbar wird das erst bei Präzisionsmessungen (deshalb beginnt die Tempomessung im Programm erst nach der sauberen Trennung).
- **Der „magische" Zeitschritt** $\Delta t = \Delta x/c$ ist eine 1D-Spezialität — im Yee-Gitter in 2D und 3D gibt es ihn nicht, dort ist numerische Dispersion unvermeidlich und will gemessen werden (Kapitel 8/9).
- **Diskrete Energie:** Die hier summierte Energie mischt E und B von leicht verschiedenen Zeitpunkten; deshalb schwankt sie im Promillebereich statt exakt konstant zu sein. Eine exakt erhaltene Energie kann man für das Schema definieren — für unsere Diagnose reicht die einfache Summe.

6 Das 1D-Wellenlabor

Wenn es draußen dunkel ist und drinnen hell, zeigt dir die Fensterscheibe dein Spiegelbild — blass, aber deutlich. Tagsüber ist es auch da, nur überstrahlt. Offenbar wirft Glas einen Teil des Lichts zurück, obwohl es doch durchsichtig ist. *Wie viel* Prozent sind es? Und woher „weiß" das Licht an der Glasoberfläche überhaupt, dass es sich teilen soll?

Am Ende dieses Kapitels hast du die Antwort **selbst gemessen** — in einem Wellenlabor aus etwa vierzig Zeilen Python. Der Zwei-Zeilen-Kern aus Kapitel 5 konnte bisher nur zusehen, wie ein Anfangspuls davonläuft. Ein Labor braucht mehr: eine **Quelle**, die auf Befehl sendet; **Wände**, die Wellen schlucken statt sie zurückzuwerfen; **Materialien**, die dämpfen oder brechen. Genau diese drei Anbauten bauen wir jetzt ein — und mit jedem führen wir ein Experiment durch.

Lernziele

Nach diesem Kapitel kannst du …

1. … eine Quelle einbauen und erklären, warum man ihren Wert **addiert** statt setzt (weiche vs. harte Quelle),
2. … den **Mur-Rand** erklären (bei $S = 1$: der Randpunkt übernimmt den alten Wert des Nachbarn) und seine Restreflexion messen,
3. … mit Spiegel und Dauerquelle eine **stehende Welle** erzeugen und den Knotenabstand $\lambda/2$ nachmessen,
4. … **Verluste** über die Leitfähigkeit σ einbauen (ca/cb-Koeffizienten), das exponentielle Abklingen gegen die Theorie prüfen — und erklären, warum eine *σ-Rampe* kaum reflektiert (die PML-Idee),
5. … Reflexion und Transmission an einer **Glasgrenze** messen und mit den Fresnel-Formeln vergleichen.

6.1 Das Labor

Hier ist das komplette Labor — der Kern aus Kapitel 5, erweitert um Quelle, Material und Ränder. Bevor du den Code liest, der Bauplan in Worten. **Die Bühne** ist für alle Experimente dieses Kapitels dieselbe: eine 12 m lange Strecke aus $N = 1201$ E-Punkten im Abstand $\Delta x = 1$ cm, gerechnet mit dem magischen Zeitschritt $\Delta t = \Delta x/c$ aus Kapitel 5. **Die Stellschrauben** der Funktion `simuliere()` sind die Zutaten der kommenden Abschnitte: eine Quelle (Signalform, Ort, weich oder hart), wahlweise Material (ε_r- und σ-Profile entlang der Strecke) und pro Rand die Wahl zwischen schluckend (`"mur"`) und spiegelnd (`"pec"`, der festgenagelte Rand aus Kapitel 5). **Die Messgeräte:** Über `merke` lassen sich Momentaufnahmen des Feldes zu gewählten Zeitschritten fotografieren, über `sonden` der größte je gesehene Feldbetrag an einzelnen Orten protokollieren, und die `huellkurve` hält für jeden Ort den größten Betrag über die gesamte Laufzeit fest. Lies die Funktion einmal ganz; jede Zutat wird anschließend einzeln vorgeführt:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import c, epsilon_0, mu_0

Z0 = np.sqrt(mu_0 / epsilon_0)         # Wellenwiderstand des Vakuums, ≈ 377 Ω

N = 1201                               # E-Punkte: 12 m bei Δx = 1 cm
DX = 0.01
DT = DX / c                            # S = 1, wie in Kapitel 5
X = np.arange(N) * DX


def simuliere(schritte, signal, quellort=100, hart=False, eps_r=None,
              sigma=None, raender=("mur", "mur"), merke=(), sonden=()):
    """Das 1D-Wellenlabor: Kern aus Kap. 5 + Quelle, Material, Ränder."""
    e, b = np.zeros(N), np.zeros(N - 1)
    # np.ones(n): Array voller Einsen — hier „überall Vakuum"
    eps = np.ones(N) if eps_r is None else eps_r
    sig = np.zeros(N) if sigma is None else sigma
    # Verlust-Koeffizienten: ca staucht das alte Feld bei jedem Schritt,
    # cb wiegt die Nachbar-Differenz (σ = 0 → ca = 1: Kern aus Kap. 5)
    v = sig * DT / (2 * epsilon_0 * eps)
    ca = (1 - v) / (1 + v)
    cb = (c**2 * DT / DX / eps) / (1 + v)

    bilder, pegel = {}, {s: 0.0 for s in sonden}
```

```
    huellkurve = np.zeros(N)
    for n in range(1, schritte + 1):
        b -= DT / DX * np.diff(e)                # B springt (halber Takt)
        alt_links, alt_rechts = e[1], e[-2]      # für die Mur-Ränder
        # E springt — nur innen; Anatomie der Zeile direkt unter der Zelle
        e[1:-1] = ca[1:-1] * e[1:-1] - cb[1:-1] * np.diff(b)
        if hart:
            e[quellort] = signal(n * DT)         # überschreibt!
        else:
            e[quellort] += signal(n * DT)        # addiert
        if raender[0] == "mur":
            e[0] = alt_links                     # Welle tritt links aus
        if raender[1] == "mur":
            e[-1] = alt_rechts                   # … und rechts
        if n in merke:
            bilder[n] = e.copy()
        for s in sonden:
            pegel[s] = max(pegel[s], abs(e[s]))
        huellkurve = np.maximum(huellkurve, np.abs(e))
    return e, bilder, pegel, huellkurve


def gauss(t, t0=4.2e-9, tau=0.7e-9):
    """Glockenpuls; t0 = 6·tau, damit er bei t = 0 praktisch null ist."""
    return np.exp(-((t - t0) / tau)**2)
```

Die E-Update-Zeile ist wieder die dichteste der Funktion — sie ist die Kapitel-5-Zeile, nur mit Material-Koeffizienten davor. Auseinandergefaltet passiert dies:

```
rest    = ca[1:-1] * e[1:-1]      # was vom alten Feld übrig bleibt
antrieb = cb[1:-1] * np.diff(b)   # Schub durch die B-Nachbarn
e[1:-1] = rest - antrieb          # nur die inneren N−2 Punkte
```

Die Längen-Buchhaltung ist die aus Kapitel 5: `e` hat N Einträge, das dazwischen wohnende `b` hat $N-1$, also liefert `np.diff(b)` (Nachbar-Differenzen, eine pro Lücke) genau $N-2$ Werte — einen für jeden inneren E-Punkt, elementweise verheiratet mit den passend zugeschnittenen `ca[1:-1]` und `e[1:-1]`. Im Vakuum ist `ca = 1` und `cb = c²Δt/Δx`, und die Zeile schrumpft exakt auf den Kapitel-5-Kern zusammen. Und wie dort gilt: Die Randpunkte `e[0]` und `e[-1]` lässt die Zeile absichtlich in Ruhe — ihnen fehlt der symmetrische B-Partner, sie werden von den Randzeilen darunter regiert (Mur übernimmt den alten Nachbarwert, PEC lässt die Null stehen).

Drei Dinge sind neu, und jedes bekommt jetzt sein Experiment: die Quellzeile (`+=` oder `=`), die Mur-Randzeilen — und die Koeffizienten `ca`/`cb`, die Material ins Spiel bringen.

6.2 Wände, die schlucken: der Mur-Rand

Die festen Ränder aus Kapitel 5 ($E = 0$) sind Metallwände — jede Welle kommt zurück. Für ein Labor, das den freien Raum nachbilden soll, brauchen wir das Gegenteil: einen Rand, durch den Wellen **hinauslaufen, als ginge das Gitter ewig weiter**. Bei $S = 1$ ist die Lösung verblüffend kurz. Eine linkslaufende Welle rückt pro Zeitschritt exakt eine Zelle nach links — der Randpunkt muss also einfach **den alten Wert seines Nachbarn übernehmen**:

$$E_0^{n+1} \;=\; E_1^n .$$

Das ist die Randbedingung von Mur (in ihrer 1D-Bestform). Ob sie hält, was sie verspricht, sagt der einfachste Test, den das Labor hergibt: Die weiche Standardquelle bei $x = 1$ m gibt den Gauß-Puls ab (er teilt sich in zwei Hälften der Höhe 0,5, eine läuft nach links, eine nach rechts), beide Ränder stehen auf `"mur"`, und wir lassen 2200 Schritte verstreichen — mehr als genug Zeit, dass beide Hälften ihre Ränder erreicht haben. **Erfolgskriterium:** Was danach noch im Gitter steht, wurde reflektiert; bei einem perfekten Rand wäre es exakt null.

```
# von oben: simuliere(), gauss()   (Labor-Zelle)
e, _, _, _ = simuliere(2200, gauss)
print(f"Rest im Gitter, nachdem der Puls beide Ränder passiert hat: "
      f"{np.max(np.abs(e)):.1e}")
```

```
Rest im Gitter, nachdem der Puls beide Ränder passiert hat: 7.1e-16
```

Vom Puls der Höhe 0,5 bleibt nur Maschinenrauschen — der Rand verschluckt die Welle praktisch perfekt. (Dass dieser Trick genau so nur bei $S = 1$ und nur in 1D funktioniert, gehört ins Kleingedruckte; die allgemeine Fassung kommt in Kapitel 9.)

6.3 Senden, ohne zu stören: weiche und harte Quellen

Bisher starteten unsere Wellen als Anfangsbedingung. Eine **Quelle** sendet stattdessen mitten im Lauf: An einem festen Punkt wird das Feld im Takt eines Signals bewegt — wie ein Finger, der auf der Wasseroberfläche tippt. Im Code ist das eine Zeile, aber es gibt zwei Versionen, und der Unterschied ist eine Lehrstunde:

- **weich:** `e[quellort] += signal(t)` — das Signal wird zum Feld **addiert**,
- **hart:** `e[quellort] = signal(t)` — das Signal **überschreibt** das Feld.

Solange nur gesendet wird, sehen beide gleich aus. Der Unterschied zeigt sich erst, wenn eine Welle zur Quelle **zurückkommt**:

> Vorhersage-Punkt
>
> **Bevor du die Zelle liest:** Eine Quelle bei $x = 3$ m sendet einen Puls; rechts am Ende steht eine Metallwand, die ihn zurückschickt. Der Rückläufer erreicht die Quelle, deren Signal längst wieder null ist. Was passiert bei der weichen Quelle (`+= 0` pro Schritt) — und was bei der harten (`= 0` pro Schritt)?

```
# von oben: simuliere(), gauss(), X   (Labor-Zelle)
ERG = {}
for art, hart in (("weich", False), ("hart", True)):
    _, bilder, _, _ = simuliere(2600, gauss, quellort=300, hart=hart,
                                raender=("mur", "pec"), merke=(2500,))
    ERG[art] = bilder[2500]

fig, ax = plt.subplots(figsize=(7.0, 3.0))
ax.plot(X, ERG["weich"], label="weiche Quelle")
ax.plot(X, ERG["hart"], label="harte Quelle")
ax.axvline(3.0, color="gray", lw=0.8, ls=":")
ax.set_xlabel("x (m)"); ax.set_ylabel("E (V/m)")
ax.legend(); plt.tight_layout(); plt.show()

print(f"max|E| nach Quell-Durchgang — weich: "
      f"{np.max(np.abs(ERG['weich'])):.1e},  hart: "
      f"{np.max(np.abs(ERG['hart'])):.2f}")
```

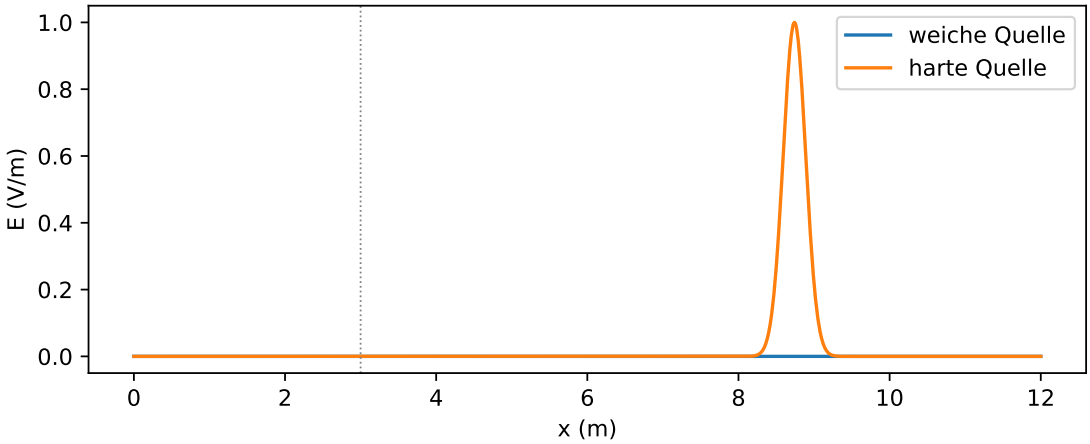


Abbildung 6.1: Schnappschuss, lange nachdem der vom Spiegel zurückgeworfene Puls den Quellort wieder erreicht hat: Bei der weichen Quelle (blau, längst hinausgelaufen) ist das Gitter leer — bei der harten Quelle (orange) wurde der Rückläufer erneut reflektiert und läuft als Geisterpuls wieder nach rechts.

```
max|E| nach Quell-Durchgang — weich: 1.2e-33,  hart: 1.00
```

Naheliegende Vermutung

Vermutung: *„Eine Quelle gibt ihren Wert vor — natürlich überschreibt man das Feld am Quellpunkt.“*
Warum sie naheliegt: So steuert man in jedem Programm eine Variable: Man weist ihr den gewünschten Wert zu. „Hart” wirkt wie die ehrlichere, direktere Quelle.
Was stattdessen stimmt: Wer das Feld überschreibt, erklärt den Quellpunkt zur **Randbedingung**: Dort gilt ab sofort „E ist, was ich sage” — egal was ankommt. Ein zurückkehrender Puls trifft auf einen Punkt, der sich keinen Millimeter bewegen lässt — also auf eine Metallwand, und wird **vollständig reflektiert** (im Bild: der Geisterpuls der harten Quelle). Die weiche Quelle dagegen *addiert* nur ihren Beitrag und lässt alles Übrige ungestört passieren — Superposition statt Diktat. Deshalb gilt: senden mit `+=`.

6.4 Interferenz nach Plan: die stehende Welle

Quelle plus Spiegel ergibt das erste klassische Experiment: Eine **Dauerquelle** (Sinus statt Puls) sendet gegen die Metallwand; hinlaufende und reflektierte Welle überlagern sich. Aus Kapitel 4 kennst du die Zutaten — was dabei herauskommen muss, sagt eine Zeile Trigonometrie: $\sin(kx - \omega t) +$ Rücklauf ergibt ein **ortsfestes Muster mal Takt**, mit Nullstellen (*Knoten*) im Abstand $\lambda/2$ und einem Knoten direkt auf dem Spiegel (dort erzwingt das Metall $E = 0$).

Der Aufbau: Der rechte Rand wird auf `"pec"` gestellt — die Metallwand bei 12 m ist unser Spiegel —, der linke bleibt schluckend. Die Dauerquelle sendet mit $f = 300$ MHz ($\lambda = 1$ m) und sanft anschwellender Amplitude; sie sitzt bei $x = 1{,}13$ m, und die krumme Zahl ist Absicht: Stünde sie exakt auf dem $\lambda/2$-Raster der Wand (etwa bei glatt 1 m, elf Wellenlängen vor dem Spiegel), käme die reflektierte Welle am Quellort genau gegenphasig an und würde links der Quelle alles auslöschen — ein hübscher Spezialfall, der aber das Bild verfälschen würde. Gemessen wird im **eingeschwungenen Zustand**: Nach 2900 Schritten (die Welle ist dann mehrfach hin- und hergelaufen) fotografieren wir das Feld über eine volle Schwingungsperiode — 100 Schnappschüsse — und bilden daraus je Ort den Maximalausschlag, die **Einhüllende**. Sie zeigt auf einen Blick, wo das Feld schwingt und wo dauerhaft Ruhe herrscht.

> Vorhersage-Punkt
>
> **Bevor du die Zelle liest:** In welchem Abstand voneinander erwartest du die Knoten der Einhüllenden — und wo genau liegt der erste, von der Wand aus gezählt?

```
# von oben: simuliere(), X; c (Import)   (Labor-Zelle)
LAMBDA = 1.0
F = c / LAMBDA                                          # ≈ 300 MHz

def sinus(t):
    return np.sin(2*np.pi*F*t) * min(t / 20e-9, 1.0)     # sanfte Rampe

# eingeschwungen nach 2 Hin- und Rückwegen; eine Periode = 100 Schritte
_, bilder, _, _ = simuliere(5000, sinus, quellort=113,
                            raender=("mur", "pec"), merke=range(2901, 3001))
einhuellende = np.max(np.abs(np.array(list(bilder.values()))), axis=0)

fig, ax = plt.subplots(figsize=(7.4, 3.2))
```

```
ax.plot(X, einhuellende, lw=1.2)
for k in range(1, 12):                                  # Theorie: Knoten
    ax.axvline(12.0 - k*LAMBDA/2, color="gray", lw=0.6, ls="--", alpha=0.7)
ax.axvline(1.13, color="tab:red", lw=0.8, ls=":")
ax.annotate("Quelle", (1.18, 1.6), fontsize=8, color="tab:red")
ax.set_xlabel("x (m)"); ax.set_ylabel("max |E| über eine Periode")
plt.tight_layout(); plt.show()
```

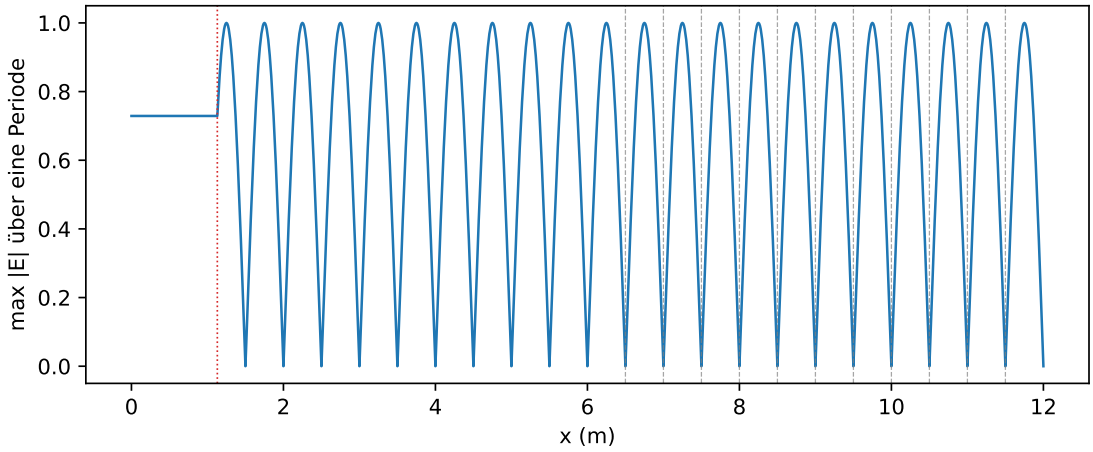


Abbildung 6.2: Einhüllende des Feldes (Maximalausschlag je Ort über eine Schwingungsperiode, im eingeschwungenen Zustand). Zwischen Quelle (1,13 m) und Spiegel (12 m) steht die Welle: Knoten exakt im λ/2-Raster der Theorie (gestrichelte Linien), Bäuche mit doppelter Amplitude dazwischen. Links der Quelle: flache Einhüllende — dort läuft die Welle nur hinaus, nichts steht.

Die Knoten sitzen exakt auf dem $\lambda/2$-Raster — der erste eine halbe Wellenlänge vor der Wand, der „nullte" auf ihr. An den Knoten ist **dauerhaft Ruhe**, obwohl zwei Wellen voller Stärke hindurchlaufen; an den Bäuchen schaukelt sich das Feld auf die **doppelte** Amplitude. Links der Quelle dagegen: glatte Einhüllende, keine Knoten — dort laufen alle Wellenanteile in **dieselbe** Richtung hinaus, und ohne Gegenläufer steht nichts. Dieses Experiment kehrt wieder: als Resonator in Kapitel 15 und als Stehwellenverhältnis auf Leitungen in Kapitel 22.

6.5 Verluste: Wellen ausbremsen — und die PML-Idee

Bisher ist unser Material nur Vakuum. Der erste Fremdstoff ist ein **verlustbehaftetes Medium**: eines mit Leitfähigkeit σ, in dem das E-Feld Ströme treibt, deren Reibung Feldenergie in Wärme verwandelt (das Ohmsche $\vec{J} = \sigma\vec{E}$ wird in Kapitel 18 genauer beleuchtet). In der Update-Regel landet das als zusätzlicher Term, und nach Auflösen stehen die zwei Koeffizienten aus dem Labor:

$$E_{\text{neu}} = c_a\, E_{\text{alt}} - c_b\, \frac{\Delta B}{\Delta x}, \qquad c_a = \frac{1-v}{1+v}, \quad v = \frac{\sigma\,\Delta t}{2\,\varepsilon_0 \varepsilon_r}.$$

$c_a < 1$ staucht das Feld bei jedem Schritt ein wenig — Reibung als Faktor. Eine Handrechnung für $\sigma = 0{,}001$ S/m im Vakuum, mit einer Feinheit, die leicht übersehen wird:

```
# von oben: DT, Z0; epsilon_0 (Import)   (Labor-Zelle)
sigma0 = 0.001
v = sigma0 * DT / (2 * epsilon_0)
ca = (1 - v) / (1 + v)
print(f"v = {v:.2e}  →  ca = {ca:.5f}")
print(f"ca^100  (würde nur E gedämpft)  : {ca**100:.3f}")
print(f"Theorie fürs Abklingen der WELLE: e^(−αL) = "
      f"{np.exp(-sigma0*Z0/2*1.0):.3f}   (α = σZ₀/2, L = 1 m)")
```

```
v = 1.88e-03  →  ca = 0.99624
ca^100  (würde nur E gedämpft)  : 0.686
Theorie fürs Abklingen der WELLE: e^(−αL) = 0.828   (α = σZ₀/2, L = 1 m)
```

Pro Schritt fast nichts — aber Vorsicht: Die Welle klingt **nicht** mit $c_a^{100} = 0{,}686$ pro Meter ab, sondern nur mit $e^{-\alpha L} = 0{,}829$. Der Faktor 2 im Exponenten hat eine schöne physikalische Erklärung: c_a staucht allein das **E-Feld** — doch eine Welle trägt die Hälfte ihrer Energie im **B-Feld** (Kapitel 4), und das lässt die Reibung ungeschoren. Jeder Energieanteil verbringt nur die halbe Zeit „angreifbar" im E-Feld, also wirkt effektiv der halbe Exponent: $\alpha = \sigma Z_0/2$. Ob das stimmt, entscheidet das Experiment — ein 600-MHz-Wellenzug durchquert vier Meter Verlustzone:

```
# von oben: simuliere(), X, sigma0, Z0   (Labor- und Handrechnungs-Zelle)
def traeger(t, f0=600e6, t0=15e-9, tau=2.5e-9):
    return np.sin(2*np.pi*f0*t) * np.exp(-((t - t0)/tau)**2)
```

```
ZONE = (X >= 6.0) & (X <= 10.0)
sigma_sprung = np.where(ZONE, sigma0, 0.0)
_, _, pegel, huelle = simuliere(4200, traeger, sigma=sigma_sprung,
                                sonden=(580, 1020))

fig, ax = plt.subplots(figsize=(7.0, 3.0))
ax.plot(X, huelle, label="Einhüllende (gemessen)")
maske = (X >= 5.8) & (X <= 10.0)
ax.plot(X[maske], pegel[580]*np.exp(-sigma0*Z0/2*(X[maske] - X[580])),
        "k--", label="Theorie e^(−αx)")
ax.axvspan(6, 10, color="orange", alpha=0.15)
ax.set_xlabel("x (m)"); ax.set_ylabel("max |E|")
ax.legend(); plt.tight_layout(); plt.show()

print(f"Dämpfung über die Zone: gemessen {pegel[1020]/pegel[580]:.3f},"
      f"  Theorie {np.exp(-sigma0*Z0/2*4.0):.3f}")
```

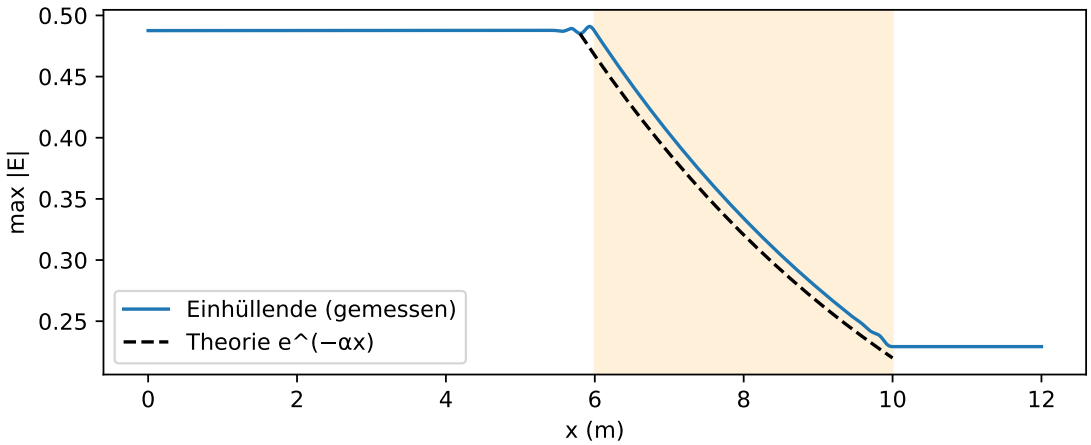


Abbildung 6.3: Einhüllende eines Wellenzugs (600 MHz), der eine 4 m lange Verlustzone (σ = 0,001 S/m, orange) durchquert: exponentielles Abklingen exakt auf der Theorielinie e^(−αx).

```
Dämpfung über die Zone: gemessen 0.472,  Theorie 0.471
```

Aber sieh genau hin: Am **Eintritt** der Zone (bei 6 m) hat die gemessene Einhüllende eine kleine Beule — dort wird ein Teil des Wellenzugs **reflektiert**. Ein Sprung in den Materialeigenschaften wirft immer etwas zurück, auch ein Sprung in σ. Was hilft? Dasselbe wie beim Betreten kalten Wassers: **langsam hineingehen**. Wir ersetzen den Sprung durch eine Rampe gleicher Gesamtdämpfung und messen, was aus der Zone zurückkommt:

```
# von oben: simuliere(), traeger(), ZONE, sigma0, X
sigma_rampe = np.where(ZONE, sigma0 * 2*(X - 6.0)/4.0, 0.0)
echo = {}
for name, sig in (("σ-Sprung", sigma_sprung), ("σ-Rampe ", sigma_rampe)):
    _, bilder, _, _ = simuliere(1400, traeger, sigma=sig, merke=(1280,))
    echo[name] = np.max(np.abs(bilder[1280][:450]))
    print(f"Echo aus der Zone, {name}: {echo[name]:.1e}")
```

```
Echo aus der Zone, σ-Sprung: 3.9e-03
Echo aus der Zone, σ-Rampe : 7.8e-05
```

Die Rampe reflektiert **fünfzigmal weniger** als der Sprung. Genau das ist die Idee der **PML** (*perfectly matched layer*), des Standard-Randes aller professionellen FDTD-Programme: ein Verlustmaterial, dessen Dämpfung sanft von null anwächst — die Welle findet keine Kante, an der sie sich spiegeln könnte, läuft hinein und wird aufgezehrt. Unsere σ-Rampe ist die halbe PML; was ihr zur ganzen fehlt (und warum es erst in 2D nötig wird), steht im Kleingedruckten.

6.6 Das Spiegelbild vermessen: Fresnel an der Glasgrenze

Jetzt zur Fensterscheibe. Glas ist verlustfrei, aber **dichter** als Luft: relative Permittivität $\varepsilon_r = n^2$ mit Brechungsindex $n = 1{,}5$ — im Labor genügt dafür das `eps_r`-Array. Was an der Grenze passieren muss, verraten zwei Stetigkeitsforderungen: Das E-Feld und das B-Feld können an einer (unmagnetischen, ladungsfreien) Grenzfläche nicht springen. Mit den Amplituden 1 (einfallend), r (reflektiert) und t (durchgelassen) heißt das

$$\underbrace{1 + r = t}_{\text{E stetig}}, \qquad \underbrace{1 - r = n\,t}_{\text{B stetig, da } B = nE/c \text{ im Glas}} \qquad \Rightarrow \quad r = \frac{1-n}{1+n}, \quad t = \frac{2}{1+n}.$$

Die Handrechnung für Glas: $r = (1-1{,}5)/(1+1{,}5) = -0{,}2$ und $t = 2/2{,}5 = 0{,}8$. Das Minuszeichen heißt: Der reflektierte Puls kommt **invertiert** zurück. Und die reflektierte *Leistung* ist $r^2 = 0{,}04$.

Vorhersage-Punkt

Bevor du die Zelle liest: $r^2 = 4\,\%$ ist die Theorie. Unser Labor weiß nichts von Fresnel — es kennt nur Nachbar-Differenzen und ein ε_r-Array. Wirst du die $-0{,}2$ im Schnappschuss wiederfinden? Und was erwartest du für Form und Tempo des Pulses *im* Glas?

```
# von oben: simuliere(), gauss(), X   (Labor-Zelle)
N_GLAS = 1.5
eps_glas = np.where(X >= 7.0, N_GLAS**2, 1.0)
_, bilder, _, _ = simuliere(1250, gauss, eps_r=eps_glas, merke=(600, 1124))

einfall = np.max(bilder[600])
r_mess = np.min(bilder[1124][:600]) / einfall
t_mess = np.max(bilder[1124][700:]) / einfall

fig, ax = plt.subplots(figsize=(7.2, 3.2))
ax.plot(X, bilder[600], "gray", label="vorher (einfallend)")
ax.plot(X, bilder[1124], "tab:red", label="nachher")
ax.axvspan(7, 12, color="tab:blue", alpha=0.12)
ax.set_xlabel("x (m)"); ax.set_ylabel("E (V/m)")
ax.legend(); plt.tight_layout(); plt.show()

print(f"gemessen:  r = {r_mess:+.4f}   (Fresnel: {(1-N_GLAS)/(1+N_GLAS):+.1f})")
print(f"           t = {t_mess:+.4f}   (Fresnel: {2/(1+N_GLAS):+.1f})")
print(f"reflektierte Leistung r² = {r_mess**2*100:.1f} %")
```

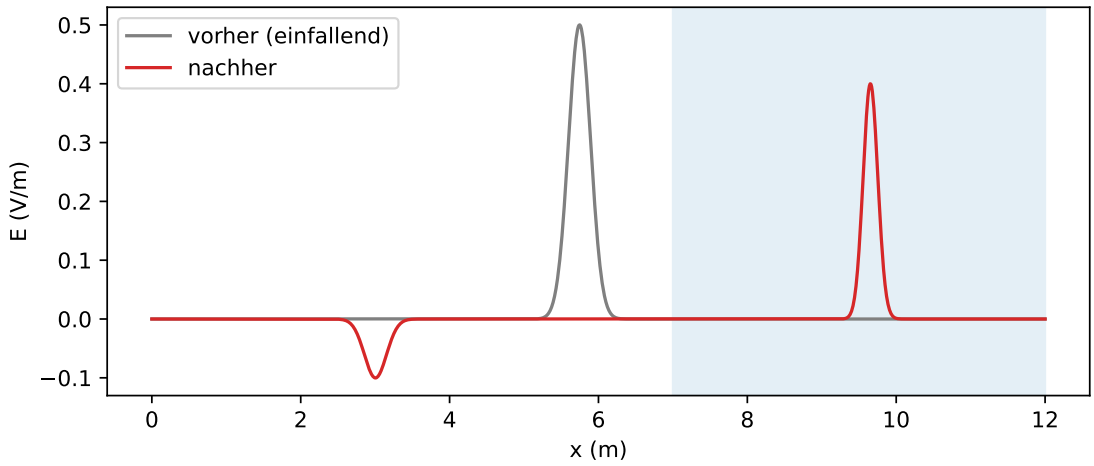


Abbildung 6.4: Ein Puls trifft auf Glas (n = 1,5, blau hinterlegt): Der reflektierte Anteil (links) ist invertiert und hat 20 % der Amplitude; der durchgelassene (rechts) hat 80 % — und ist räumlich gestaucht, weil er im Glas nur noch mit c/n vorankommt.

```
gemessen:  r = -0.2003   (Fresnel: -0.2)
           t = +0.7999   (Fresnel: +0.8)
reflektierte Leistung r² = 4.0 %
```

Das Labor liefert $r = -0{,}2003$ und $t = 0{,}7999$ — Fresnel auf drei Nachkommastellen, aus nichts als Nachbar-Differenzen. **Die 4 % sind dein Spiegelbild in der Fensterscheibe.** (Genau genommen hat die Scheibe zwei Oberflächen, die je 4 % zurückwerfen — was das für Echos bedeutet, untersucht Übung 6.4.) Und der durchgelassene Puls zeigt beide vorhergesagten Effekte: volle Form, aber räumlich um den Faktor n gestaucht, weil er im Glas nur noch c/n schafft.

Das **Kapitel-Programm** `programme/kap06/kap06_wellenlabor_1d.py` enthält das Labor samt aller vier Experimente mit `assert`-Schranken (Mur-Rest $< 10^{-12}$, Dämpfung und Fresnel auf Prozentbruchteile).

Merkkasten

- **Senden mit `+=`**: Die weiche Quelle addiert und lässt Rückläufer passieren; die harte Quelle ist eine Metallwand mit Programm.
- **Mur-Rand bei $S = 1$**: Randpunkt := alter Nachbarwert — die Welle tritt aus, Rest ist Maschinenrauschen.
- **Stehende Welle**: Quelle + Spiegel → Knoten im $\lambda/2$-Raster, der erste auf der Wand; Bäuche mit doppelter Amplitude.

- **Verluste**: $c_a < 1$ staucht das Feld pro Schritt; $c_a^{\text{Schritte}} = e^{-\alpha L}$ mit $\alpha = \sigma Z_0/2$. Materialsprünge reflektieren — **Rampen nicht: die PML-Idee.**
- **Fresnel gemessen**: $r = (1-n)/(1+n)$, an Glas $-0{,}2 \to 4$ % Spiegelbild; transmittierter Puls läuft mit c/n und ist gestaucht.

Roter Faden

Das Labor verbindet fast alles Bisherige: den Kern aus **Kapitel 5**, die stehende Welle, die in **Kapitel 4** als Formel auftauchte (und hier als Messung), die Energiebuchhaltung aus **Kapitel 3** (das Ohmsche σ wird in **Kapitel 18** zur Wärmeleistung $\vec{J} \cdot \vec{E}$). Nach vorn: **Kapitel 7** zeigt, dass unsere Update-Schleife eine Matrix-Multiplikation ist; **Kapitel 8** klärt, welcher Zeitschritt erlaubt ist; in **Kapitel 12** wird aus der 1D-Glasgrenze die 2D-Brechung; und in **Kapitel 22** kehrt Fresnel als Reflexion an Leitungs-Stoßstellen wieder — dort heißt n dann Impedanzverhältnis, und „Anpassung" wird zur Ingenieurskunst.

Übungen

Ü 6.1 (Verstehen). Die harte Quelle reflektiert den Rückläufer *vollständig* — wie eine Metallwand. Eine Metallwand erzwingt $E = 0$; die harte Quelle erzwingt $E = s(t) \neq 0$. Warum reflektiert sie trotzdem genauso total?

> **Musterlösung zu Ü 6.1**
>
> Entscheidend ist nicht, *welcher* Wert erzwungen wird, sondern *dass* einer erzwungen wird: Der Punkt reagiert nicht auf das ankommende Feld. Dank Superposition kann man das Gesamtfeld als „vorgeschriebener Anteil + Abweichung" lesen — und für die **Abweichung** gilt am Quellpunkt exakt die Bedingung $= 0$, also die Metallwand-Bedingung. Der Rückläufer ist eine solche Abweichung und wird daher wie an einer Wand (invertiert und vollständig) reflektiert.

Ü 6.2 (Verstehen). Berechne r und t für den umgekehrten Weg — einen Puls, der **aus dem Glas in die Luft** läuft ($n_1 = 1{,}5 \to n_2 = 1$; allgemein gilt $r = (n_1 - n_2)/(n_1 + n_2)$, $t = 2n_1/(n_1 + n_2)$). Was fällt am Vorzeichen von r auf — und wieso ist $t > 1$ kein Verstoß gegen die Energieerhaltung?

Musterlösung zu Ü 6.2

$r = (1{,}5 - 1)/(2{,}5) = +0{,}2$: gleicher Betrag, aber **nicht invertiert** — die Inversion gibt es nur beim Übergang ins dichtere Medium. Und $t = 3/2{,}5 = 1{,}2 > 1$: Die *Feldamplitude* hinter der Grenze ist größer als davor! Kein Widerspruch: Energie fließt mit $S \propto E^2/Z$, und in der dünneren Luft gehört zur selben Leistung ein größeres E. Die Bilanz stimmt exakt: $r^2 + (1/n)\,t^2 = 0{,}04 + 0{,}96 = 1$. Leistung, nicht Amplitude, ist die Erhaltungsgröße.

Ü 6.3 (Verändern). Miss im Labor die Reflexion an **Wasser** ($n = 1{,}33$). Sage den Wert vorher mit der Fresnel-Formel voraus.

Musterlösung zu Ü 6.3

Vorhersage: $r = (1 - 1{,}33)/(2{,}33) = -0{,}1416$, also $r^2 \approx 2\,\%$. Die Messung — nur eine Zeile anders:

```
# von oben: simuliere(), gauss(), X   (Labor-Zelle)
eps_wasser = np.where(X >= 7.0, 1.33**2, 1.0)
_, bilder, _, _ = simuliere(1250, gauss, eps_r=eps_wasser,
                            merke=(600, 1124))
r_w = np.min(bilder[1124][:600]) / np.max(bilder[600])
print(f"r gemessen: {r_w:+.4f}   (Fresnel: {(1-1.33)/(1+1.33):+.4f})")
```

```
r gemessen: -0.1418   (Fresnel: -0.1416)
```

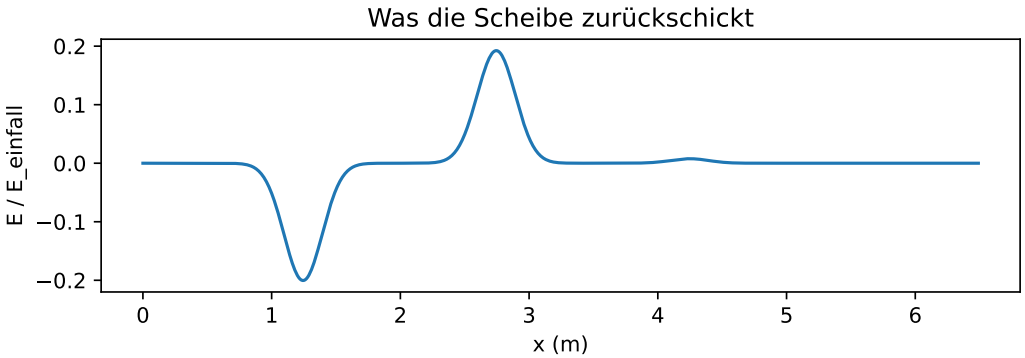
Wasser spiegelt also nur halb so stark wie Glas — deshalb ist das Spiegelbild in einem stillen, dunklen Teich blasser als im Fenster.

Ü 6.4 (Übertragen). Eine echte Fensterscheibe hat **zwei** Oberflächen. Baue eine 50 cm dicke Glasschicht (von 7 m bis 7,5 m) und sieh dir an, was zurückkommt: Es ist nicht *ein* Echo, sondern eine **Echofolge**. Miss die Amplituden der ersten beiden Echos und erkläre sie mit den Ergebnissen von Ü 6.2.

Musterlösung zu Ü 6.4

```
# von oben: simuliere(), gauss(), X, N_GLAS
eps_scheibe = np.where((X >= 7.0) & (X < 7.5), N_GLAS**2, 1.0)
_, bilder, _, _ = simuliere(1350, gauss, eps_r=eps_scheibe,
                            merke=(600, 1300))
spur = bilder[1300][:650] / np.max(bilder[600])

fig, ax = plt.subplots(figsize=(7.0, 2.6))
ax.plot(X[:650], spur)
ax.set_xlabel("x (m)"); ax.set_ylabel("E / E_einfall")
ax.set_title("Was die Scheibe zurückschickt")
plt.tight_layout(); plt.show()
print(f"Echo 1: {np.min(spur):+.4f}    Echo 2: {np.max(spur):+.4f}")
```




```
Echo 1: -0.2003    Echo 2: +0.1923
```

Echo 1 ($-0{,}200$) ist die bekannte Reflexion an der Vorderseite — invertiert. Echo 2 ($+0{,}192$) erzählt seine Reiseroute: hinein ($t = 0{,}8$), an der **Rückseite** reflektiert ($r' = +0{,}2$, nicht invertiert — Ü 6.2!), wieder hinaus ($t' = 1{,}2$, ebenfalls Ü 6.2): $0{,}8 \cdot 0{,}2 \cdot 1{,}2 = +0{,}192$ — exakt der Messwert, inklusive des „unmöglichen" Faktors $t' > 1$. Jeder weitere Umlauf in der Scheibe multipliziert mit $r'^2 = 0{,}04$: Echo 3 wäre $+0{,}0077$, kaum noch sichtbar. Die Echos werden schnell leise, aber sie hören nie ganz auf — dieses unendliche Echospiel heißt **Fabry-Pérot** und bekommt in Kapitel 27 sein eigenes Kapitel, denn richtig eingesetzt macht es aus zwei Glasflächen einen Spiegel, ein Filter oder einen Laser-Resonator.

Das Kleingedruckte

- **Mur allgemein:** Für $S < 1$ lautet der Rand $E_0^{n+1} = E_1^n + \frac{c\Delta t - \Delta x}{c\Delta t + \Delta x}(E_1^{n+1} - E_0^n)$ — bei $S = 1$ kollabiert er auf unsere Kurzform. In 2D/3D bleibt vom Mur-Rand nur eine Näherung, die schräg einfallende Wellen teilweise reflektiert; deshalb dominiert dort die PML.
- **Die ganze PML** dämpft nicht nur über σ, sondern zugleich über einen magnetischen Verlustterm, abgestimmt auf $\sigma_m/\mu_0 = \sigma/\varepsilon_0$ — dann ist die Schicht für *jede* Frequenz und (in 2D/3D) jeden Einfallswinkel reflexionsfrei „angepasst". Unsere Rampe zeigt das Prinzip; die volle Konstruktion kommt in Kapitel 9.
- **Amplituden in verschiedenen Medien** vergleicht man nur über die Leistung $\propto E^2/Z$ (siehe Ü 6.2: $t = 1{,}2$ und trotzdem Energieerhaltung). Systematisch wird diese Buchführung mit dem Impedanzbegriff in Kapitel 22.
- **Harte Quellen haben ihren Platz:** Wo man ein exakt vorgeschriebenes Feldprofil einspeisen will (z. B. als Rand eines Wellenleiters), ist Überschreiben genau richtig — man muss nur wissen, dass man damit eine reflektierende Randbedingung gesetzt hat.

7 Explizit oder implizit? Das Update als Matrix

Wer schon einmal mit einem FEM-Werkzeug, einem Schaltungssimulator wie SPICE oder einem Statik-Programm gearbeitet hat, kennt den heiligen Dreiklang der numerischen Simulation: **Matrix aufstellen, Gleichungssystem lösen, Ergebnis ablesen.** Riesige dünnbesetzte Matrizen, clevere Löser, Speicher als Engpass — so sieht „richtige" Simulation aus.

Nur: In den Kapiteln 5 und 6 haben wir komplette Wellenphysik simuliert, und da war — nichts davon. Keine Matrix, kein Löser, nur zwei Zeilen `np.diff`. Haben wir heimlich doch ein Gleichungssystem gelöst, gut versteckt in NumPy? Oder geht Simulation auch *ohne* — und wenn ja, was unterscheidet dann unsere Methode von denen, die tatsächlich lösen? Dieses Kapitel zieht die Trennlinie scharf: Sie heißt **explizit gegen implizit**, und auf beiden Seiten stehen Verfahren mit sehr verschiedenen Stärken.

Lernziele

Nach diesem Kapitel kannst du …

1. … die Update-Schleife als Matrix-Vektor-**Produkt** $v^{n+1} = A\,v^n$ schreiben und numerisch belegen, dass beides dasselbe ist,
2. … erklären, warum ein explizites Verfahren dabei **kein** Gleichungssystem löst — und woran man eines erkennt, das es tut,
3. … das implizite Crank–Nicolson-Verfahren aufstellen und sehen, wo dort das Gleichungssystem tatsächlich entsteht,
4. … die Eigenschaften beider Welten vorführen: CFL-Explosion bei $S > 1$, unbedingte Stabilität implizit — und warum **stabil nicht genau heißt**,
5. … begründet wählen: Welches Verfahren für welches Problem?

7.1 Die Schleife, als Matrix gelesen

Zuerst eine Vereinfachung, die sich doppelt auszahlt: Wir messen das Magnetfeld ab jetzt in der Größe $u = c \cdot B$ — sie hat dieselbe Einheit wie E (Volt pro Meter; in den Plots von Kapitel 5 haben wir das stillschweigend schon getan). Damit schrumpfen die beiden Update-Zeilen aus Kapitel 5 auf einen einzigen Koeffizienten zusammen — die **Courant-Zahl** $S = c\,\Delta t/\Delta x$:

```
u -= S * np.diff(e)          # der B-Schritt
e[1:-1] -= S * np.diff(u)    # der E-Schritt
```

Beide Zeilen tun dasselbe: Sie bilden Nachbar-Differenzen und ziehen sie gewichtet ab. „Nachbar-Differenzen bilden" ist eine **lineare** Operation — und jede lineare Operation auf einem Vektor lässt sich als Matrix schreiben. Wie diese Matrix aussieht, zeigt am schnellsten ein winziges Gitter:

```
import numpy as np
import matplotlib.pyplot as plt
# scipy.sparse: Matrizen, die nur ihre Nicht-Null-Einträge speichern
import scipy.sparse as sp
from scipy.constants import c

def differenzmatrix(m):
    """(m-1) × m: Zeile i bildet die Nachbar-Differenz f[i+1] - f[i]."""
    # sp.diags(werte, lagen): legt Werte auf Diagonalen — hier -1 auf
    # die Hauptdiagonale (Lage 0) und +1 auf die Nebendiagonale (Lage 1)
    return sp.diags([-np.ones(m - 1), np.ones(m - 1)], [0, 1],
                    shape=(m - 1, m), format="csr")

D6 = differenzmatrix(6)
# .toarray(): macht aus der sparse-Matrix ein gewöhnliches Array
print(D6.toarray())
```

```
[[-1.  1.  0.  0.  0.  0.]
 [ 0. -1.  1.  0.  0.  0.]
 [ 0.  0. -1.  1.  0.  0.]
 [ 0.  0.  0. -1.  1.  0.]
 [ 0.  0.  0.  0. -1.  1.]]
```

Lies eine Zeile von Hand: Zeile 2 hat -1 in Spalte 2 und $+1$ in Spalte 3, also liefert sie $(De)_2 = e_3 - e_2$ — exakt der dritte Eintrag von `np.diff(e)`.

Die ganze Matrix *ist* `np.diff`, aufgeschrieben als Tabelle. Und weil ein kompletter Zeitschritt nur aus solchen Differenzen und Additionen besteht, lässt sich der **gesamte Schritt** als eine Matrix A schreiben, die auf den Zustandsvektor $v = (e_0, \dots, e_{N-1}, u_0, \dots, u_{N-2})$ wirkt:

$$v^{n+1} \;=\; A\,v^n, \qquad A = \begin{pmatrix} I + S^2\,GD & -S\,G \\ -S\,D & I \end{pmatrix},$$

wobei D die Differenzen der e-Werte bildet und G die der u-Werte (mit Nullzeilen am Rand — unsere Metallwände). Der S^2-Block entsteht, weil der E-Schritt das schon aktualisierte u benutzt. Ob die Behauptung stimmt, prüfen wir nicht per Algebra, sondern per Experiment — Matrix gegen Schleife, 100 Schritte:

```
# von oben: differenzmatrix(), sp; c (Import)
N = 1201
DX = 0.01
X = np.arange(N) * DX

def matrizen(s, n=N):
    """Schritt-Matrix A und Crank–Nicolson-Paar (L, R) zur Courant-Zahl s."""
    d = differenzmatrix(n)
    # sp.vstack: stapelt Matrizen übereinander (hier: Nullzeilen als Rand)
    g = sp.vstack([sp.csr_matrix((1, n - 1)), differenzmatrix(n - 1),
                   sp.csr_matrix((1, n - 1))], format="csr")
    # sp.bmat: setzt eine Matrix aus Blöcken zusammen
    a = sp.bmat([[sp.identity(n) + s**2 * (g @ d), -s * g],
                 [-s * d, sp.identity(n - 1)]], format="csr")
    m = sp.bmat([[sp.csr_matrix((n, n)), -s * g],
                 [-s * d, sp.csr_matrix((n - 1, n - 1))]], format="csr")
    i = sp.identity(2 * n - 1, format="csr")
    return a, (i - m / 2).tocsc(), (i + m / 2).tocsr()

def startzustand():
    e = np.exp(-((X - 6.0) / 0.5)**2)
    # np.concatenate: hängt Arrays aneinander — e und u in EINEM Vektor
    return np.concatenate([e, np.zeros(N - 1)])

def leapfrog_schritt(v, s):
    """Der Schleifencode aus Kapitel 5/6, in u = c·B-Schreibweise."""
    e, u = v[:N].copy(), v[N:].copy()
    u -= s * np.diff(e)
    e[1:-1] -= s * np.diff(u)
```

```
    return np.concatenate([e, u])

A, L, R = matrizen(1.0)
v_schleife, v_matrix = startzustand(), startzustand()
for _ in range(100):
    v_schleife = leapfrog_schritt(v_schleife, 1.0)
    v_matrix = A @ v_matrix
print(f"max. Abweichung Schleife vs. Matrix: "
      f"{np.max(np.abs(v_schleife - v_matrix)):.1e}")
print(f"A speichert {A.nnz:,} Einträge — von {(2*N-1)**2:,} möglichen "
      f"({A.nnz/(2*N-1)**2*100:.2f} %)")
```

```
max. Abweichung Schleife vs. Matrix: 4.7e-15
A speichert 9,597 Einträge — von 5,764,801 möglichen (0.17 %)
```

Identisch bis auf Rundungsreste — Schleife und Matrix sind **dasselbe Verfahren in zwei Schreibweisen**. Und die Matrix ist extrem **dünnbesetzt** (englisch *sparse*): Von 5,8 Millionen möglichen Einträgen sind nur 0,17 % von null verschieden, weil jeder Gitterwert nur mit seinen unmittelbaren Nachbarn spricht. So sieht sie aus:

```
# von oben: matrizen()
A_klein, _, _ = matrizen(1.0, n=25)
fig, ax = plt.subplots(figsize=(4.6, 4.6))
ax.spy(A_klein, markersize=3)        # spy: zeichnet die Nicht-Null-Einträge
plt.tight_layout(); plt.show()
```

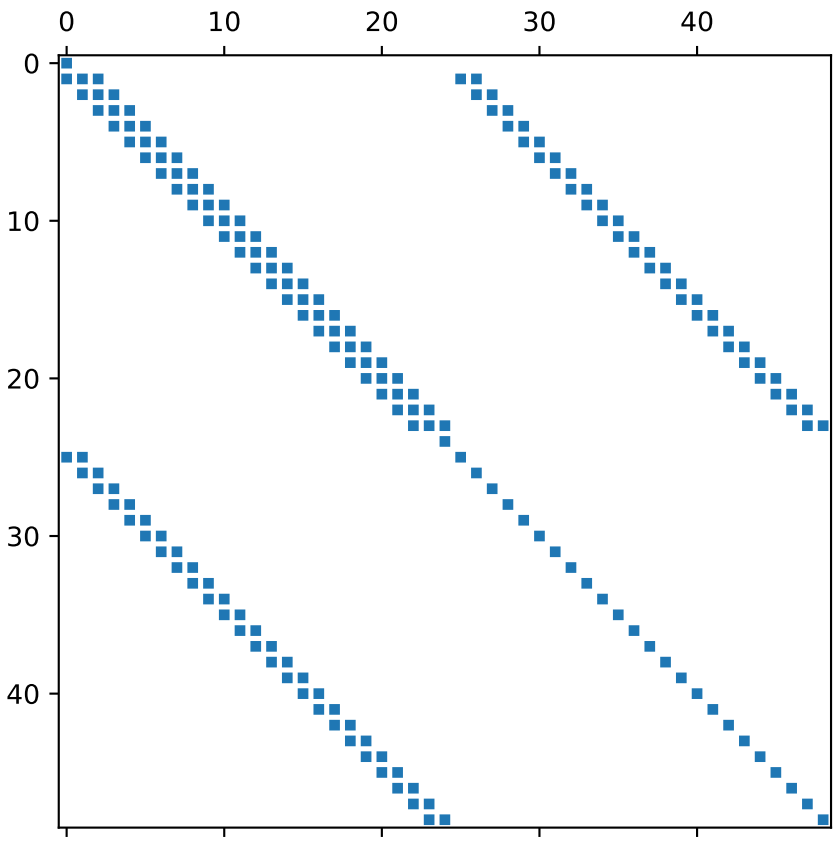


Abbildung 7.1: Die Schritt-Matrix A für ein kleines Gitter mit 25 Punkten (jeder Punkt = ein Nicht-Null-Eintrag). Links oben der e-Block, rechts unten der u-Block; die schmalen Bänder sind die Nachbar-Differenzen — alles andere ist null, denn jeder Punkt spricht nur mit seinen Nachbarn.

⚠ Naheliegende Vermutung

Vermutung: *„Ein Differenzenverfahren stellt aus den Gleichungen ein lineares Gleichungssystem auf und löst es — irgendwo in NumPy steckt also doch ein Löser."*
Warum sie naheliegt: Für weite Teile der Simulationswelt stimmt es. FEM-Programme, Schaltungssimulatoren, Statik-Löser, Strömungslöser — sie alle stellen Systeme auf und lösen sie. Und eine Matrix haben wir ja nun tatsächlich gefunden.
Was stattdessen stimmt: Es kommt darauf an, **auf welcher Seite der Gleichung die Unbekannte steht.** Unser Schritt lautet $v^{n+1} = A\,v^n$ — die Unbekannte v^{n+1} steht *allein* links, rechts stehen nur bekann-

> te alte Werte. Das ist eine **Auswertung** (multiplizieren, fertig), kein Gleichungssystem. Verfahren dieser Bauart heißen **explizit**. Ein Gleichungssystem entsteht erst, wenn die Unbekannte auf *beiden* Seiten auftaucht — etwa $L\,v^{n+1} = R\,v^n$ mit einer Matrix L vor der Unbekannten, die man nicht trivial wegdividieren kann. Solche Verfahren heißen **implizit**, und eines bauen wir jetzt.

7.1.1 „Auswertung" heißt: alle Punkte gleichzeitig

Der Satz „rechts stehen nur bekannte alte Werte" hat eine sehr praktische Konsequenz, und sie erklärt rückwirkend, warum unser Code seit Kapitel 5 so aussieht, wie er aussieht. Wenn jeder neue Wert nur aus *alten* Werten berechnet wird, dann hängt **kein neuer Eintrag von einem anderen neuen Eintrag ab**. Ob du zuerst `e[1]` ausrechnest oder zuerst `e[1000]`, von links nach rechts oder rückwärts — das Ergebnis ist identisch, denn die Punkte reden während des Updates nicht miteinander. Mathematisch werden alle Gitterpunkte **gleichzeitig** aktualisiert; jede Reihenfolge ist nur eine Abarbeitungs-Bequemlichkeit.

Genau deshalb durfte die *Raumschleife* von Anfang an in NumPy verschwinden. Ausgeschrieben wäre ein FDTD-Lauf eine Doppelschleife — außen die Zeit, innen der Ort:

```
for n in range(schritte):          # Zeit:  MUSS nacheinander laufen
    for i in range(1, N - 1):      # Raum:  dürfte auch parallel laufen
        e[i] = e[i] - s * (u[i] - u[i - 1])
```

Die innere Schleife ist reihenfolgefrei, also darf NumPy sie als *eine* Array-Operation ausführen (`np.diff` plus elementweise Verrechnung) — dieselbe Rechnung, nur ohne Python-Umweg pro Punkt; auf einer Grafikkarte würden die Punkte buchstäblich parallel rechnen. Die **äußere Schleife dagegen kann niemand wegoptimieren**: Schritt $n+1$ braucht das Ergebnis von Schritt n — die Zukunft lässt sich nicht parallel zur Gegenwart berechnen. Von der Doppelschleife bleibt darum genau eine echte Schleife übrig, das `for n in range(...)` der Zeit; alles andere ist Matrix mal Vektor.

Eine Sorgfaltsstelle gibt es doch — und sie betrifft nicht die Reihenfolge *innerhalb* eines Arrays, sondern die Frage, **welchen Zeitstand eine Zeile liest**. Vergleiche die beiden Verfahren aus Kapitel 5:

```
def leapfrog_schritt(e, b):        # KEINE Kopien – mit Absicht:
    b -= ... np.diff(e)            # b springt zuerst,
    e[1:-1] -= ... np.diff(b)      # und e liest das FRISCHE b!

def euler_schritt(e, b):           # Kopien – ebenfalls mit Absicht:
    e_neu, b_neu = e.copy(), b.copy()
    e_neu[...] = ...               # beide Zeilen lesen die
    b_neu[...] = ...               # ALTEN Felder e und b
```

Beim Leapfrog ist das Lesen des frisch aktualisierten `b` kein Schlamperei-Risiko, sondern **das Verfahren selbst**: Genau dadurch liegt das benutzte B zeitlich einen halben Schritt vor dem E — der Bockspringer-Versatz aus dem Yee-Schema in Kapitel 5. Der Euler-Gleichschritt dagegen *definiert* sich darüber, dass beide Felder aus demselben alten Zeitstand schreiten — darum die `.copy()`-Zeilen: Ohne sie läse die zweite Zeile das schon aktualisierte Feld der ersten, und aus dem Gleichschritt würde unbemerkt ein halb versetztes Verfahren. Der ganze Unterschied zwischen den beiden Bauplänen steckt also buchstäblich in der Frage „Kopie oder frisches Feld?“. (NumPy nimmt dir dabei eine Falle ab: *Innerhalb* einer Zuweisungszeile wird die komplette rechte Seite ausgewertet, bevor irgendetwas geschrieben wird — halbfertige Mischstände innerhalb eines Arrays kann es gar nicht geben.)

Und damit ist auch klar, was gleich anders wird: Beim **impliziten** Verfahren hängen die neuen Werte *voneinander* ab — der neue `e[5]` steht in derselben Gleichung wie der neue `e[4]` und der neue `e[6]`. Dann gibt es kein „alle gleichzeitig, Reihenfolge egal” mehr, elementweises NumPy hilft nicht weiter, und genau deshalb braucht jeder Zeitschritt jetzt einen Gleichungslöser.

7.2 Die andere Philosophie: Crank–Nicolson

Warum sollte jemand die Unbekannte freiwillig auf beide Seiten stellen? Aus einem ehrenwerten Motiv: **Symmetrie in der Zeit.** Unser expliziter Schritt wertet die Änderungsrate aus den *alten* Feldern aus. Das Verfahren von **Crank–Nicolson** (CN) nimmt stattdessen den Mittelwert aus alter und neuer Änderungsrate — die Trapezregel, angewandt auf die Zeit:

$$v^{n+1} = v^n + \tfrac{1}{2}M\left(v^n + v^{n+1}\right) \quad\Longrightarrow\quad \underbrace{\left(I - \tfrac{M}{2}\right)}_{L} v^{n+1} = \underbrace{\left(I + \tfrac{M}{2}\right)}_{R} v^n,$$

mit M als der Matrix der Änderungsraten (unsere zwei Differenzen-Blöcke mal S). Jetzt steht v^{n+1} wirklich auf beiden Seiten: **Jeder Zeitschritt ist ein Gleichungssystem** mit $2N - 1$ Unbekannten. Der Lohn dieser Mühe ist beachtlich — und ihr Preis auch. Beides zeigt ein Doppel-Experiment:

Vorhersage-Punkt

Bevor du die Zelle liest: Wir lassen erstens den expliziten Schritt mit $S = 1{,}05$ laufen — nur 5 % über dem „magischen" Zeitschritt. Und

```
# von oben: matrizen(), startzustand()
# scipy.sparse.linalg.splu: zerlegt L EINMAL (LU-Faktorisierung);
# danach kostet jedes lu.solve(b) nur noch Vorwärts-/Rückwärts-Einsetzen
from scipy.sparse.linalg import splu

e0 = np.sum(startzustand()**2)

A_schnell, _, _ = matrizen(1.05)
v = startzustand()
verlauf_explizit = []
for _ in range(600):
    v = A_schnell @ v
    verlauf_explizit.append(np.sum(v**2) / e0)
    if verlauf_explizit[-1] > 1e30:
        break

_, L5, R5 = matrizen(5.0)
lu5 = splu(L5)
v = startzustand()
verlauf_cn = []
for _ in range(80):                     # 80 Schritte à S=5 = 400 Zeiteinheiten
    v = lu5.solve(R5 @ v)               # hier wird WIRKLICH gelöst
    verlauf_cn.append(np.sum(v**2) / e0)

fig, ax = plt.subplots(figsize=(6.6, 3.4))
ax.semilogy(np.arange(1, len(verlauf_explizit)+1) * 1.05,
            verlauf_explizit, label="explizit, S = 1,05")
ax.semilogy(np.arange(1, 81) * 5.0, verlauf_cn,
            label="implizit (CN), S = 5")
ax.set_xlabel("Zeit (in Einheiten von Δx/c)")
ax.set_ylabel("Energie / Anfangsenergie")
```

```
ax.legend(); plt.tight_layout(); plt.show()

print(f"explizit S=1,05: Energie ×{verlauf_explizit[-1]:.1e} (Abbruch)")
print(f"CN S=5:          Energie ×{verlauf_cn[-1]:.6f}")
```

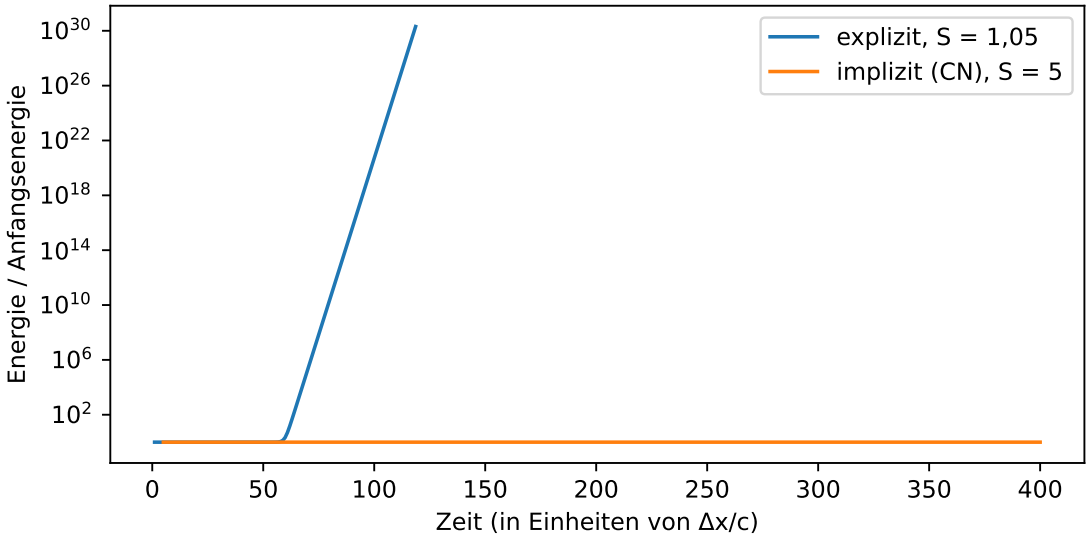


Abbildung 7.2: Die Energie beider Verfahren (logarithmisch, über der physikalischen Zeit). Explizit mit S = 1,05 explodiert nach kurzem Anlauf um dreißig Größenordnungen — Crank–Nicolson läuft mit S = 5 unbeirrt energietreu weiter.

```
explizit S=1,05: Energie ×2.0e+30 (Abbruch)
CN S=5:          Energie ×1.000000
```

Das explizite Verfahren reißt schon bei 5 % Übermut die bekannte Instabilität auf (warum die Grenze exakt bei $S = 1$ liegt, beweist Kapitel 8). Crank–Nicolson dagegen ist **unbedingt stabil**: Es gibt keinen zu großen Zeitschritt, die Energie bleibt auf sechs Nachkommastellen erhalten. Das wirkt wie Zauberei — fünffacher Zeitschritt, perfekte Energie. Zeit für die Ernüchterung:

Vorhersage-Punkt

Bevor du die Zelle liest: Stabil und energieerhaltend — heißt das, das CN-Ergebnis bei $S = 10$ ist auch *richtig*? Wir vergleichen die Pulsform nach derselben physikalischen Laufzeit mit der Leapfrog-Referenz. Lege dich fest.

```
# von oben: startzustand(), leapfrog_schritt(), matrizen(), splu, X, N
v_ref = startzustand()
for _ in range(400):
    v_ref = leapfrog_schritt(v_ref, 1.0)

fig, ax = plt.subplots(figsize=(6.8, 3.2))
ax.plot(X, v_ref[:N], "k", lw=1.8, label="Referenz (Leapfrog, S = 1)")
for s_cn in (2, 10):
    _, L_cn, R_cn = matrizen(float(s_cn))
    lu = splu(L_cn)
    v = startzustand()
    for _ in range(400 // s_cn):
        v = lu.solve(R_cn @ v)
    fehler = np.max(np.abs(v[:N] - v_ref[:N])) / np.max(v_ref[:N])
    ax.plot(X, v[:N], label=f"CN, S = {s_cn}  (Formfehler {fehler*100:.0f} %)")
ax.set_xlim(7, 12)
ax.set_xlabel("x (m)"); ax.set_ylabel("E (V/m)")
ax.legend(fontsize=8); plt.tight_layout(); plt.show()
```

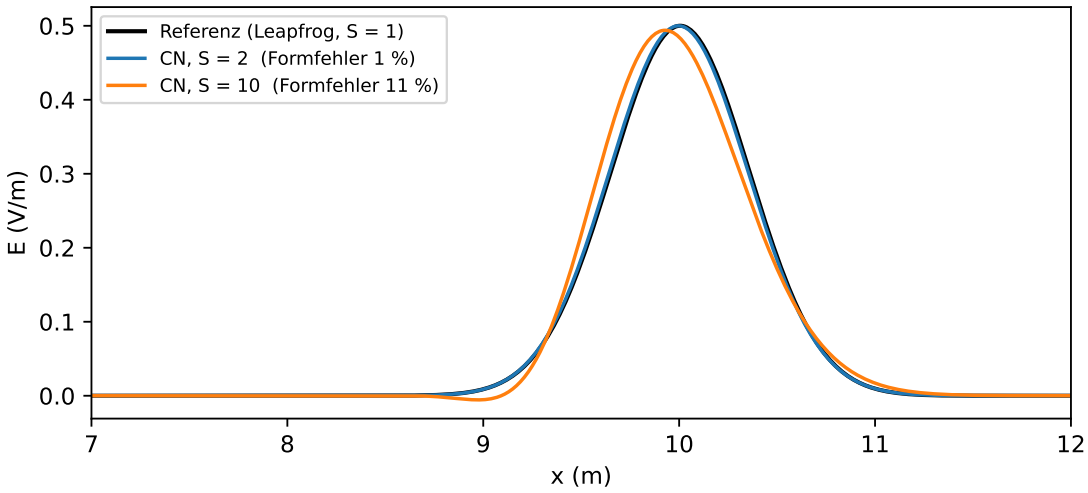


Abbildung 7.3: Der rechte Puls nach gleicher physikalischer Laufzeit: Die Leapfrog-Referenz (schwarz) ist formtreu; Crank–Nicolson bleibt stabil und energietreu, aber mit wachsendem Zeitschritt eilt Welligkeit hinterher und die Form zerfasert — die Energie stimmt, ihre Verteilung nicht mehr.

Stabil heißt nicht genau. Die Energie ist exakt erhalten — aber sie sitzt zunehmend in nachlaufenden Kräuseln statt im Puls: 11 % Formfehler bei

$S = 10$, Tendenz steigend. Das ist kein Bug von CN, sondern das ehrliche Kleingedruckte jeder „unbedingten Stabilität": Der große Zeitschritt *tastet die Schwingung zu grob ab*, und die Phasen der Wellenanteile geraten durcheinander. Wer die Welle auflösen will, braucht kleine Zeitschritte — dann aber kann man sie auch gleich explizit machen.

Bleibt der Preis pro Schritt. Einmal messen:

```
# von oben: matrizen(), splu, startzustand()
# time.perf_counter: präzise Stoppuhr (Sekunden als Gleitkommazahl)
import time

A, L, R = matrizen(1.0)
lu = splu(L)
v = startzustand()
t0 = time.perf_counter()
for _ in range(2000):
    v = A @ v
us_explizit = (time.perf_counter() - t0) / 2000 * 1e6

v = startzustand()
t0 = time.perf_counter()
for _ in range(2000):
    v = lu.solve(R @ v)
us_implizit = (time.perf_counter() - t0) / 2000 * 1e6

print(f"explizit: {us_explizit:5.1f} µs/Schritt   "
      f"implizit: {us_implizit:5.1f} µs/Schritt   "
      f"(Faktor {us_implizit/us_explizit:.1f})")
```

```
explizit:   5.3 µs/Schritt   implizit:  32.3 µs/Schritt   (Faktor 6.1)
```

In 1D ist der Faktor noch zahm, denn unser Gleichungssystem ist ein schmales Band — und die LU-Zerlegung von L fällt nur **einmal** an, danach wird pro Schritt nur eingesetzt. In 3D kippt die Rechnung: Dort hat das System Millionen Unbekannte, die Zerlegung füllt sich mit zusätzlichen Einträgen voll (*fill-in*), und man weicht auf iterative Löser aus, die pro Schritt selbst iterieren müssen. Explizite Verfahren skalieren dagegen einfach weiter: Nachbarn anschauen, addieren — notfalls auf tausend Prozessoren gleichzeitig.

7.3 Wann welches Verfahren?

	explizit (Leapfrog/FDTD)	**implizit** (z. B. Crank–Nicolson)
pro Schritt	Multiplikation — billig, speicherarm	Gleichungssystem **lösen** — teuer
Zeitschritt	begrenzt: $S \leq 1$ (CFL, Kap. 8)	beliebig groß: unbedingt stabil
großer Schritt	explodiert	stabil, aber Form zerfasert
parallelisierbar	hervorragend (nur Nachbarn)	schwieriger (globales System)
zu Hause bei	**Wellen**: Man will die Schwingung ohnehin zeitlich auflösen	**Diffusion/Wärme, steife Probleme**: Die Physik glättet, große Schritte schaden wenig (Ü 7.2)
typische Vertreter	FDTD, FIT (openEMS)	Wärmeleitung, SPICE, FEM-Zeitschritte, ADI-FDTD

Zwei Ergänzungen machen das Bild komplett. Erstens: Es gibt Mischformen — **ADI-FDTD** etwa rechnet implizit nur entlang jeweils einer Raumrichtung und umgeht so die CFL-Grenze, wenn winzige Geometriedetails (eine dünne Schicht, ein feiner Spalt) sonst absurd kleine Zeitschritte erzwingen würden. Zweitens: **Stationäre** Probleme — ein Potentialfeld, eine Antennen-Stromverteilung — haben gar keinen Zeitschritt; sie *sind* von Natur aus ein einziges großes Gleichungssystem. Die Momentenmethode (NEC), mit der wir ab Kapitel 11 Antennen rechnen, gehört in diese Welt: Dort ist „Matrix aufstellen und lösen" keine Wahl, sondern die Definition des Verfahrens.

Das **Kapitel-Programm** `programme/kap07/kap07_explizit_implizit.py` bündelt alle vier Befunde — Identität, CFL-Explosion, CN-Stabilität samt Formfehler, Kostenmessung — mit `assert`-Schranken.

Merkkasten

- Die FDTD-Schleife **ist** ein Matrix-Vektor-Produkt $v^{n+1} = A\,v^n$ mit dünnbesetztem A — eine Auswertung, **kein Gleichungssystem**. Explizit = Unbekannte nur links.
- **Implizit** = Unbekannte auf beiden Seiten ($L\,v^{n+1} = R\,v^n$): jeder Schritt löst ein System — dafür unbedingt stabil.
- **Stabil ≠ genau**: Zu große implizite Schritte tasten die Schwingung zu grob ab; die Energie stimmt, die Form nicht.

- Faustregel: **Wellen explizit** (Zeitauflösung braucht man ohnehin), **Diffusion und steife Probleme implizit**, stationäre Probleme sind immer ein großes System.

Roter Faden

Die zwei Bänder in der spy-Abbildung sind exakt die `np.diff`-Zeilen aus **Kapitel 5/6** — nichts an diesem Kapitel war neue Physik, nur ein neuer Blick. Die Matrix-Sicht trägt ab jetzt: **Kapitel 8** liest aus den Eigenwerten von A ab, warum die CFL-Grenze exakt bei $S = 1$ liegt (Übung 7.4 nimmt den Vorgeschmack). **Kapitel 11** sortiert die Löser-Landschaft — und du wirst die Trennlinie dieses Kapitels dort überall wiedererkennen: Meep und openEMS marschieren explizit, NEC stellt auf und löst.

Übungen

Ü 7.1 (Verstehen). Wenn der explizite Schritt die Matrix A nie braucht — wozu ist es dann trotzdem nützlich, sie aufstellen zu können? Nenne mindestens zwei Gründe.

> **Musterlösung zu Ü 7.1**
>
> Erstens als **Analysewerkzeug**: Eigenschaften des Verfahrens — Stabilität, Erhaltungsgrößen, Fehlerverhalten — sind Eigenschaften der Matrix (ihre Eigenwerte entscheiden über Explodieren oder Erhalten, siehe Ü 7.4 und Kapitel 8). Zweitens als **Verständigungsmittel**: Erst in Matrixform kann man explizite und implizite Verfahren nebeneinanderlegen und sieht, dass sie sich nur darin unterscheiden, auf welcher Seite die Unbekannte steht. Drittens (Bonus) als **Korrektheitsbeweis**: Die Übereinstimmung von Schleife und $A \cdot v$ auf 10^{-15} belegt, dass der Code genau das lineare Verfahren implementiert, das er behauptet.

Ü 7.2 (Verstehen). Für die Wärmeleitungsgleichung (Diffusion) verlangt das explizite Verfahren $\Delta t \leq \Delta x^2/(2D)$ — beachte das **Quadrat**. Warum macht das implizite Verfahren dort so viel attraktiver als bei Wellen?

Musterlösung zu Ü 7.2

Zweimal Quadrat-Pech für explizit: Halbiert man bei Diffusion den Gitterabstand, muss der Zeitschritt auf ein **Viertel** schrumpfen (bei Wellen nur auf die Hälfte) — feine Gitter werden explizit ruinös teuer. Gleichzeitig ist der große implizite Schritt bei Diffusion fast gratis: Die Physik **glättet** alles Schnelle von selbst weg, es gibt keine Schwingungsphasen, die ein grober Schritt verfehlen könnte (genau daran scheiterte CN bei unserer Welle). Implizit passt also zur Diffusion wie explizit zur Welle: Das Verfahren sollte der Physik ähneln, die es rechnet.

Ü 7.3 (Verändern). Ersetze Crank–Nicolson durch **Rückwärts-Euler**: $(I - M)\, v^{n+1} = v^n$ (die Änderungsrate wird *nur* am neuen Zeitpunkt ausgewertet). Lass den Puls mit $S = 2$ über dieselbe Strecke laufen wie in der Genauigkeits-Abbildung. Was passiert mit Amplitude und Energie — und ist das je nützlich?

Musterlösung zu Ü 7.3

```
# von oben: differenzmatrix(), sp, N, splu, startzustand(), e0
def matrix_m(s, n=N):
    d = differenzmatrix(n)
    g = sp.vstack([sp.csr_matrix((1, n - 1)), differenzmatrix(n - 1),
                   sp.csr_matrix((1, n - 1))], format="csr")
    return sp.bmat([[sp.csr_matrix((n, n)), -s * g],
                    [-s * d, sp.csr_matrix((n - 1, n - 1))]], format="csr")

L_be = (sp.identity(2*N - 1, format="csr") - matrix_m(2.0)).tocsc()
lu_be = splu(L_be)
v = startzustand()
for _ in range(200):                        # 200 Schritte à S = 2
    v = lu_be.solve(v)
print(f"Energie: {np.sum(v**2)/e0:.3f}   Amplitude: {np.max(v[:N]):.3f}"
      f"   (Referenz-Amplitude: 0.5)")
```

```
Energie: 0.781   Amplitude: 0.391   (Referenz-Amplitude: 0.5)
```

Der Puls **schrumpft**: gut ein Fünftel der Energie ist verschwunden, die Spitze auf 0,39 gefallen — Rückwärts-Euler ist stabil, aber **numerisch dissipativ**: Es vernichtet Energie, die physikalisch erhalten wäre, am stärksten die feinen Anteile. Für Wellen ist das Gift. Nützlich ist es trotzdem — genau dann, wenn man Dämpfung *möchte*: um störende Transienten wegzubeißen, steife Anteile ruhigzustellen oder ein stationäres

Gleichgewicht schnell zu erreichen. Auch Dissipation ist ein Werkzeug; man muss nur wissen, dass man es in der Hand hält.

Ü 7.4 (Übertragen). Die Stabilität eines linearen Verfahrens steckt in den **Eigenwerten** seiner Schritt-Matrix: Bei jedem Schritt wird jeder Eigenanteil mit seinem λ multipliziert — gibt es ein $|\lambda| > 1$, wächst er exponentiell. Berechne $\max|\lambda|$ von A (kleines Gitter, $n = 80$) für $S = 0{,}5,\ 1{,}0,\ 1{,}05$ und deute das Ergebnis.

Musterlösung zu Ü 7.4

```python
# von oben: matrizen()
for s in (0.5, 1.0, 1.05):
    A80, _, _ = matrizen(s, n=80)
    # np.linalg.eigvals: alle Eigenwerte einer (vollen) Matrix
    lam = np.linalg.eigvals(A80.toarray())
    print(f"S = {s:4}:  max|λ| = {np.max(np.abs(lam)):.6f}")
```

```
S =  0.5:  max|λ| = 1.000000
S =  1.0:  max|λ| = 1.000000
S = 1.05:  max|λ| = 1.874893
```

Bei $S \leq 1$ liegen **alle** Eigenwerte auf dem Einheitskreis ($|\lambda| = 1$): Kein Anteil wächst, keiner stirbt — das Verfahren erhält, was es transportiert. Bei $S = 1{,}05$ springt $\max|\lambda|$ auf 1,87: Der zugehörige Eigenanteil wird pro Schritt fast verdoppelt, und weil Rundungsrauschen jeden Eigenanteil mikroskopisch enthält, explodiert *jede* Simulation — exakt der Mechanismus aus Kapitel 5 (Ü 5.2). Die Stabilitätsfrage ist damit auf eine Eigenwertfrage geschrumpft, und Kapitel 8 beantwortet sie analytisch: Warum kippt $|\lambda|$ genau bei $S = 1$?

Das Kleingedruckte

- **Zeitmessungen** wie unsere µs-Werte hängen von Rechner, NumPy-Version und Tagesform ab — belastbar ist die Größenordnung und der Trend (1D-Band: einstelliger Faktor; 3D: Welten), nicht die Ziffer.
- **CN ist nicht „das" implizite Verfahren**, sondern der symmetrische Vertreter einer Familie (θ-Verfahren): Rückwärts-Euler (Ü 7.3) ist ihr dissipatives Ende, CN das erhaltende. Profi-Löser mischen je nach Bedarf.

- **Die LU-Zerlegung vorab** funktioniert nur, weil unsere Matrix L konstant ist (lineares Material, festes Gitter). Sobald sich Materialeigenschaften mit der Zeit oder dem Feld ändern (nichtlineare Optik, Teil VII-Ausblick), muss implizit pro Schritt neu zerlegt oder iteriert werden — der Kostenvorteil von explizit wächst dann weiter.
- **„Unbedingt stabil" gilt pro Verfahren, nicht pro Problem:** Auch CN kann durch Randbedingungen, variable Koeffizienten oder Kopplungen seine Garantie verlieren. Stabilität prüft man am Gesamtsystem — im Zweifel so, wie wir hier: messen.

8 Stabilität und numerische Dispersion

Zwei Rechnungen aus den letzten Kapiteln sind offen geblieben. In Kapitel 5 lief der Puls nicht *ungefähr* mit Lichtgeschwindigkeit, sondern **exakt** — auf sechzehn Stellen; wir haben das dem „magischen Zeitschritt" $\Delta t = \Delta x / c$ zugeschrieben und die Erklärung vertagt. Und in Übung 7.4 hat NumPy gemessen, dass der größte Eigenwert der Schritt-Matrix bis $S = 1$ stur auf $1{,}000000$ bleibt und bei $S = 1{,}05$ auf $1{,}87$ springt — warum kippt es ausgerechnet bei der Courant-Zahl $S = c\,\Delta t / \Delta x = 1$, und warum so schlagartig?

Dieses Kapitel beantwortet beides mit einer einzigen Formel. Sie ist kurz genug für einen Kaffeebecher-Aufdruck, und sie hat zwei Gesichter: Eines verbietet zu große Zeitschritte (sonst Explosion), das andere bestraft zu grobe Gitter (die Wellen laufen dann zu langsam — und zwar *gemessen* zu langsam). Am Ende weißt du nicht nur, warum Simulationen explodieren, sondern auch, wie fein dein Gitter sein muss, damit du den Ergebnissen trauen darfst.

Lernziele

Nach diesem Kapitel kannst du …

1. … anschaulich begründen, warum $S \leq 1$ sein muss (die Welle darf pro Zeitschritt höchstens eine Zelle weit kommen),
2. … die gefährlichste Gitterwelle — den Zickzack mit $\lambda = 2\Delta x$ — von Hand durch den Zeitschritt schicken und aus ihrer 2×2-Matrix die CFL-Grenze **exakt** bei $S = 1$ herleiten,
3. … numerische Dispersion vorführen und deuten: Kurze Gitterwellen laufen zu langsam, ein scharfer Puls zieht eine Wellenschleppe,
4. … die Dispersionsrelation $\sin(\omega \Delta t/2) = S \sin(k \Delta x/2)$ herleiten und gegen Messungen bei 4, 8 und 16 Punkten pro Wellenlänge prüfen,
5. … erklären, was den magischen Zeitschritt magisch macht — und mit Faustregeln ein Gitter wählen, dem man trauen kann.

8.1 Das Tempolimit, anschaulich

Ein Blick auf die zwei Update-Zeilen aus Kapitel 5 genügt für eine erste, sehr handfeste Beobachtung:

```
u -= S * np.diff(e)          # u sieht nur seine zwei e-Nachbarn
e[1:-1] -= S * np.diff(u)    # e sieht nur seine zwei u-Nachbarn
```

Jeder Gitterpunkt spricht pro Zeitschritt **nur mit seinen direkten Nachbarn**. Eine Information — sagen wir: „hier ist gerade ein Puls losgelaufen" — kommt im Programm also pro Schritt höchstens eine Zelle weit, egal was wir rechnen. Die echte Welle dagegen legt in der Zeit Δt die Strecke $c\,\Delta t$ zurück. Wenn $c\,\Delta t > \Delta x$ ist, also $S > 1$, dann ist die Wahrheit **schneller als das Schema**: Das Programm müsste Feldwerte an Orten ändern, von denen es noch gar nichts wissen kann. Das kann nicht gut gehen — die Bedingung

$$S = \frac{c\,\Delta t}{\Delta x} \leq 1$$

heißt **Courant-Bedingung** (nach Richard Courant, auch CFL-Bedingung nach Courant, Friedrichs und Lewy, 1928 — älter als jeder Computer).

Das Argument ist ehrlich, aber unvollständig. Es sagt, dass es *jenseits* der Grenze nicht gehen kann — aber nicht, **wie** das Versagen aussieht, und auch nicht, ob nicht vielleicht schon bei $S = 0{,}9$ Schluss ist. Dafür müssen wir hinsehen — mit diesem Versuchsaufbau:

Die Bühne ist die vertraute 1D-Strecke: 400 Zellen à 1 cm, also 4 m, mit festen Rändern (PEC, Kapitel 6) — die Funktion `leapfrog` unten ist wörtlich der Kern aus den Kapiteln 5–7 in der $u = c\,B$-Schreibweise, in der S der einzige Koeffizient ist. **Anfangsbedingung:** ein glatter Gauß-Puls im E-Feld in der Mitte bei $x = 2$ m (Breite 12 cm), $u = 0$ — er wird sich wie in Kapitel 5 in zwei Hälften teilen, aber darum geht es diesmal nicht. **Der einzige Streichholz-Moment:** Wir setzen $S = 1{,}05$ — der Zeitschritt ist fünf Prozent größer, als die Courant-Bedingung erlaubt. **Gemessen wird** das Feldbild nach 60 und nach 120 Schritten, einmal im Überblick und einmal unter der Lupe (einzelne Gitterpunkte!), dazu zwei Kennzahlen: der Maximalbetrag des Felds und die *Nachbar-Korrelation* — das mittlere Produkt benachbarter Feldwerte, normiert auf den mittleren quadrierten Wert. Sie ist unser Form-Detektor: $+1$ hieße „Nachbarn sind praktisch gleich" (glattes Feld), -1 hieße „jeder Punkt ist das Negativ seines Nachbarn".

Vorhersage-Punkt

Dass die Simulation bei $S = 1{,}05$ explodiert, wissen wir aus Kapitel 7. Aber **wie** sieht das Feld dabei aus? Bleibt der Puls ein wachsender Puls? Wird das Feld zufälliges Rauschen? Oder etwas Drittes — und wenn ja: mit welcher Form, und welche Nachbar-Korrelation gehört dazu? Lege dich fest, bevor du weiterliest.

```
import numpy as np
import matplotlib.pyplot as plt

DX = 0.01                                   # Gitterabstand (m)

def leapfrog(s, nx, schritte, e0=None, u0=None, quelle=None):
    """Der Kern aus Kapitel 5-7 (u = c·B), Ränder PEC, weiche Quelle bei i=5."""
    e = np.zeros(nx) if e0 is None else e0.copy()
    u = np.zeros(nx - 1) if u0 is None else u0.copy()
    for n in range(schritte):
        u -= s * np.diff(e)
        e[1:-1] -= s * np.diff(u)
        if quelle is not None:
            e[5] += quelle(n)
    return e
```

```
# von oben: leapfrog(), DX
nx = 400
x = np.arange(nx) * DX
e_start = np.exp(-((x - 2.0) / 0.12)**2)        # glatter Puls bei x = 2 m

e_60 = leapfrog(1.05, nx, 60, e0=e_start)
e_120 = leapfrog(1.05, nx, 120, e0=e_start)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7.2, 3.0))
ax1.plot(x, e_start, "gray", lw=1.2, label="Start")
ax1.plot(x, e_60, "tab:red", lw=0.8, label="nach 60 Schritten")
ax1.set_xlim(1.4, 2.6)
ax1.set_xlabel("x (m)"); ax1.set_ylabel("E (V/m)")
ax1.legend(fontsize=8)
ax2.plot(x, e_60, "o-", color="tab:red", ms=3.5, lw=0.7)
ax2.set_xlim(1.94, 2.06)
ax2.set_xlabel("x (m)")
ax2.set_title("Lupe: jeder Punkt ein Gitterwert", fontsize=9)
plt.tight_layout(); plt.show()
```

```
zacken = e_120[150:250]                          # Region um den alten Puls
korrelation = np.mean(zacken[1:] * zacken[:-1]) / np.mean(zacken**2)
print(f"nach 120 Schritten: max|E| = {np.max(np.abs(e_120)):.1e} V/m")
print(f"Nachbar-Korrelation: {korrelation:+.3f}   (perfekter Zickzack: -1)")
```

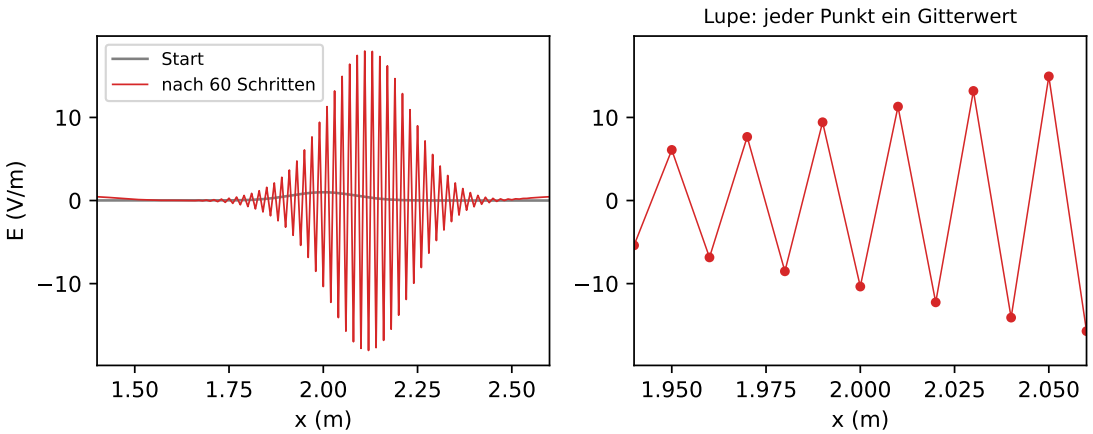


Abbildung 8.1: Ein glatter Puls bei S = 1,05. Links: Nach 60 Schritten ist aus dem Puls (grau) ein schnell oszillierendes Muster geworden, dessen Einhüllende noch an den Puls erinnert. Rechts die Lupe mit den einzelnen Gitterpunkten: Das Feld springt von Punkt zu Punkt zwischen Plus und Minus — ein perfekter Zickzack mit der Wellenlänge 2Δx. Die Linie verbindet nur die Punkte (Lesehilfe: Punkte = die echten Gitterwerte, Linie = Zeichenhilfe).

```
nach 120 Schritten: max|E| = 3.5e+17 V/m
Nachbar-Korrelation: -1.009   (perfekter Zickzack: -1)
```

Weder wachsender Puls noch Zufallsrauschen: Das Feld organisiert sich in die **kürzeste Welle, die das Gitter darstellen kann** — Plus, Minus, Plus, Minus, ein Vorzeichenwechsel pro Zelle, also $\lambda = 2\Delta x$. Der Form-Detektor schlägt voll aus: Die Nachbar-Korrelation liegt bei -1, jeder Punkt ist das Negativ seines Nachbarn. Nach 120 Schritten hat dieser Zickzack die Größenordnung 10^{17} erreicht, während der eigentliche Puls die ganze Zeit brav geblieben ist.

Die Explosion ist also kein allgemeines Chaos, sondern **das Werk einer einzigen Welle**. Das ist eine gute Nachricht: Eine einzelne Welle können wir von Hand durch den Zeitschritt schicken.

8.2 Der Hauptverdächtige: die Zickzackwelle

Die Zickzackwelle hat an den e-Punkten die Werte $E, -E, E, -E, \ldots$ und an den dazwischen sitzenden u-Punkten die Werte $U, -U, U, -U, \ldots$ — zwei Zahlen genügen, um sie komplett zu beschreiben. Was macht `np.diff` mit so einem Muster? Die Handrechnung mit $E = 1$:

```
zick = np.array([1.0, -1.0, 1.0, -1.0, 1.0])
print("Zickzack:    ", zick)
print("np.diff:     ", np.diff(zick))
```

```
Zickzack:     [ 1. -1.  1. -1.  1.]
np.diff:      [-2.  2. -2.  2.]
```

Jede Differenz ist $-1 - 1 = -2$ oder $1 - (-1) = +2$, im Wechsel: **Aus dem Zickzack wird wieder ein Zickzack, nur mit dem Faktor -2.** Die Form bleibt unter dem Update also erhalten; was sich ändert, sind nur die zwei Amplituden E und U. Setzen wir das in die Update-Zeilen ein (`u -= S*np.diff(e)` macht aus dem e-Zickzack der Stärke E einen Beitrag $+2SE$ zum u-Zickzack, und entsprechend für die zweite Zeile, in der bereits das **neue** u steht):

$$U' = U + 2S\,E, \qquad E' = E - 2S\,U' = (1 - 4S^2)\,E - 2S\,U.$$

Ein Zeitschritt ist für diese Welle also eine 2×2-Matrix, die auf das Zahlenpaar (E, U) wirkt:

$$\begin{pmatrix} E' \\ U' \end{pmatrix} = \underbrace{\begin{pmatrix} 1 - 4S^2 & -2S \\ 2S & 1 \end{pmatrix}}_{M(S)} \begin{pmatrix} E \\ U \end{pmatrix}.$$

Ob die Welle wächst, entscheiden die **Eigenwerte** von M — die Streckfaktoren, mit denen wir schon in Übung 7.4 gearbeitet haben: Gibt es ein λ mit $|\lambda| > 1$, wird der zugehörige Anteil bei jedem Schritt um diesen Faktor verstärkt. Für eine 2×2-Matrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ führt die Eigenwert-Bedingung auf eine gewöhnliche quadratische Gleichung:

$$\lambda^2 - T\,\lambda + D = 0, \qquad T = a + d \text{ (die \textbf{Spur})}, \quad D = ad - bc \text{ (die Determinante)}.$$

Beides können wir für $M(S)$ direkt hinschreiben — die Handrechnung, hier für $S = 1{,}05$, jeder Zwischenwert eine nachprüfbare Zahl:

```
S = 1.05
a, b_, c_, d = 1 - 4*S**2, -2*S, 2*S, 1.0
T = a + d                                   # Spur:  -3,41 + 1     = -2,41
D = a*d - b_*c_                             # Det.:  -3,41 + 4,41  =  1
diskriminante = (T/2)**2 - D                # 1,4520 - 1 = 0,4520
lam1 = T/2 + np.sqrt(diskriminante)         # -1,205 + 0,672
lam2 = T/2 - np.sqrt(diskriminante)         # -1,205 - 0,672
print(f"Spur T = {T:.4f},  Determinante D = {D:.4f}")
print(f"Eigenwerte: {lam1:.4f} und {lam2:.4f},  Produkt = {lam1*lam2:.4f}")
```

```
Spur T = -2.4100,  Determinante D = 1.0000
Eigenwerte: -0.5327 und -1.8773,  Produkt = 1.0000
```

Zwei Befunde stecken in diesen Zahlen. Erstens ist die Determinante **exakt 1** — und das gilt für jedes S, denn $D = (1-4S^2)\cdot 1 - (-2S)(2S) = 1-4S^2+4S^2 = 1$. Weil das Produkt der Eigenwerte gleich der Determinante ist, gilt also immer $\lambda_1\lambda_2 = 1$: Was die Matrix in einer Richtung streckt, muss sie in der anderen stauchen. Es gibt nur zwei Möglichkeiten, diese Bilanz zu erfüllen — entweder ein echtes Streck-Stauch-Paar wie hier ($-1{,}877$ und $-0{,}533$), oder beide Eigenwerte haben den Betrag **exakt 1**.

Zweitens: Welcher der beiden Fälle eintritt, entscheidet allein die **Spur**. Die Wurzel $\sqrt{(T/2)^2-1}$ liefert nur dann eine reelle Zahl — also einen echten Streckfaktor —, wenn $|T| \geq 2$ ist. Für $|T| < 2$ gibt es keinen reellen Eigenwert: keinen Streckfaktor, keine Richtung, die einfach nur verstärkt würde. Das Zahlenpaar (E, U) wird dann bei jedem Schritt **gedreht** statt gestreckt — E und U schaukeln ineinander wie Auslenkung und Schwung eines Pendels, der Betrag bleibt. (NumPy bestätigt das: `np.linalg.eigvals` liefert für $S \leq 1$ ein komplexes Zahlenpaar — Drehungen statt Streckungen —, dessen Betrag `np.abs` exakt 1 ist; so kam Übung 7.4 zu ihrer Messung $1{,}000000$.)

Mit $T = 2-4S^2$ wird aus $|T| \leq 2$ die Bedingung $-2 \leq 2-4S^2$, also $S^2 \leq 1$. Da ist sie — **die CFL-Grenze, exakt bei** $S = 1$, nicht ungefähr:

$$\max_S |\lambda| = 1 \ \text{ für } S \leq 1, \qquad |\lambda| = \frac{|T| + \sqrt{T^2-4}}{2} > 1 \ \text{ für } S > 1.$$

```
# von oben: leapfrog() ungenutzt; T-Formel aus dem Text
def verstaerkung(s):
    spur = 2 - 4 * s**2
    if abs(spur) <= 2:
        return 1.0
```

```
    return (abs(spur) + np.sqrt(spur**2 - 4)) / 2

s_achse = np.linspace(0, 1.2, 400)
fig, ax = plt.subplots(figsize=(6.4, 3.2))
ax.plot(s_achse, [verstaerkung(s) for s in s_achse], "tab:blue",
        label="Zickzack-Welle (Spurformel)")
ax.plot([0.5, 1.0, 1.05], [1.0, 1.0, 1.874893], "ko", ms=6,
        label="gemessen in Ü 7.4 (volle Matrix)")
ax.axvline(1.0, color="gray", ls=":", lw=0.8)
ax.set_xlabel("Courant-Zahl S = cΔt/Δx")
ax.set_ylabel("max|λ| pro Zeitschritt")
ax.legend()
plt.tight_layout(); plt.show()
print(f"Spurformel bei S = 1,05: {verstaerkung(1.05):.6f}"
      f"   (Ü 7.4 maß: 1.874893)")
```

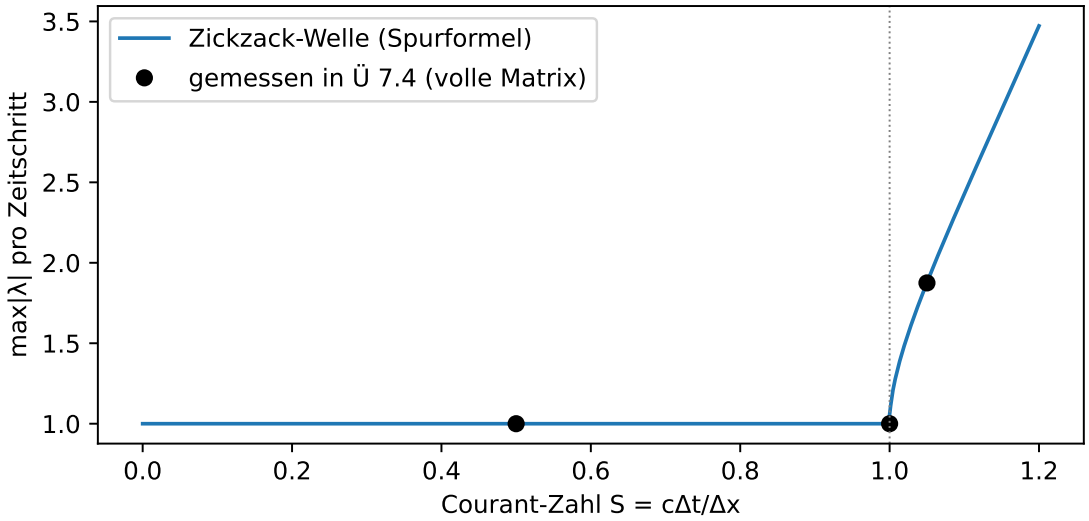


Abbildung 8.2: Der größte Eigenwert-Betrag der Zickzackwelle als Funktion der Courant-Zahl (Linie: die Spurformel). Bis S = 1 liegt er exakt auf 1 (Drehung, nichts wächst); jenseits knickt die Kurve sofort steil nach oben. Die Punkte sind die numerischen Messwerte aus Übung 7.4 an der vollen 159×159-Schritt-Matrix.

```
Spurformel bei S = 1,05: 1.877328   (Ü 7.4 maß: 1.874893)
```

Die Spurformel sagt $1{,}8773$, Übung 7.4 hat an der vollen Matrix $1{,}8749$ gemessen — eine Abweichung von gut einem Promille. Sie ist kein Fehler,

sondern Fensterglas zum Verständnis: Auf dem endlichen Gitter mit seinen zwei Metallwänden passt der perfekte Zickzack nicht ganz hinein, die randnächsten Punkte tanzen leicht aus der Reihe. Je größer das Gitter, desto näher rückt der Messwert an die Formel (bei $n = 320$ Zellen sind es schon $1{,}8772$).

Und warum explodiert dann *jede* Simulation, auch die mit einem glatten, harmlosen Puls? Das hat Übung 5.2 schon beantwortet: Das Rundungsrauschen der Gleitkommazahlen enthält *jede* Welle mikroskopisch — auch den Zickzack, mit einer Amplitude um 10^{-15}. Bei $S = 1{,}05$ wird er pro Schritt ver-$1{,}877$-facht: $10^{-15} \cdot 1{,}877^{120} \approx 7 \cdot 10^{17}$. Gemessen haben wir oben $3{,}5 \cdot 10^{17}$ — die Explosion lief nach Fahrplan.

⚠ Naheliegende Vermutung

„Dann nehme ich zur Sicherheit $S = 0{,}5$. Ein ordentlicher Sicherheitsabstand zur Grenze — und kleinere Zeitschritte sind ja sowieso genauer."
Warum sie naheliegt: Überall sonst stimmt das. In Kapitel 2 wurde jede Ableitung genauer, je kleiner die Schrittweite; Sicherheitsabstand zu einer Abbruchkante ist generell eine vernünftige Idee; und „kleineres Δt = feinere Zeitauflösung = besser" klingt wie ein Naturgesetz.
Was stattdessen stimmt: Stabil ist $S = 0{,}5$ — aber **nicht genauer, sondern ungenauer**. Unterhalb der Grenze laufen die Gitterwellen *zu langsam*, und zwar umso falscher, je kleiner S ist. Der Sicherheitsabstand wird mit verfälschten Geschwindigkeiten bezahlt; in 1D ist die Grenze $S = 1$ selbst der genaueste Punkt. Der Rest des Kapitels führt das vor — und misst es aus.

8.3 Unter dem Limit: Wellen im falschen Tempo

Vorhersage-Punkt

Wir schicken jetzt denselben schmalen Gauß-Puls zweimal über dieselbe Strecke von 12 Metern: einmal mit $S = 1$, einmal mit $S = 0{,}5$ (gleiches Gitter, halber Zeitschritt, doppelt so viele Schritte). Beides ist stabil. Kommt zweimal derselbe Puls an? Falls nein — was genau unterscheidet die beiden?

```
# von oben: leapfrog(), DX
nx = 2000
```

```
x = np.arange(nx) * DX
sigma, x0 = 0.03, 3.0                        # schmaler Puls (3 Zellen breit)

ergebnis = {}
for s in (1.0, 0.5):
    e0 = np.exp(-((x - x0) / sigma)**2)
    # Rechtsläufer wie in Ü 5.4: u = c·B = E an den u-Positionen,
    # eine halbe Zeitstufe vor dem Start ausgewertet
    x_u = x[:-1] + DX / 2
    u0 = np.exp(-((x_u + s * DX / 2 - x0) / sigma)**2)
    ergebnis[s] = leapfrog(s, nx, int(12.0 / DX / s), e0=e0, u0=u0)

fig, ax = plt.subplots(figsize=(7.0, 3.2))
ax.plot(x, np.exp(-((x - x0) / sigma)**2), "gray", lw=1.2, label="Start (3 m)")
ax.plot(x, ergebnis[1.0], "tab:blue", lw=1.2, label="S = 1")
ax.plot(x, ergebnis[0.5], "tab:orange", lw=1.2, label="S = 0,5")
ax.set_xlim(2.0, 16.5)
ax.set_xlabel("x (m)"); ax.set_ylabel("E (V/m)")
ax.legend()
plt.tight_layout(); plt.show()

print(f"Spitze nach 12 m:  S = 1:   {np.max(ergebnis[1.0]):.6f}")
print(f"                   S = 0,5: {np.max(ergebnis[0.5]):.4f}")
print(f"Schleppe (max |E| weit hinter dem Puls, S = 0,5): "
      f"{np.max(np.abs(ergebnis[0.5][700:1400])):.4f}")
```

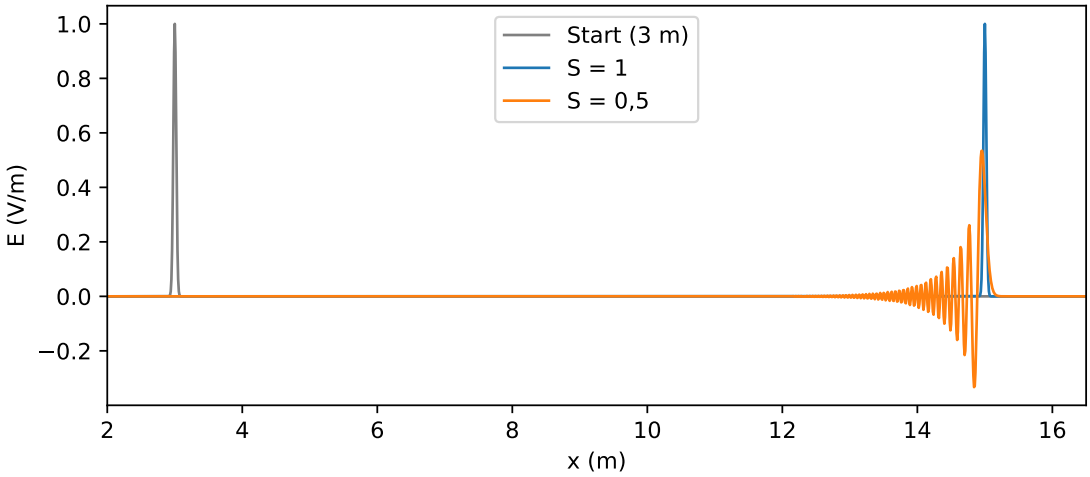


Abbildung 8.3: Derselbe schmale Puls (Start grau bei x = 3 m) nach 12 m Laufstrecke. Mit S = 1 (blau) kommt er in Originalform an — Punkt für Punkt exakt. Mit S = 0,5 (orange) hat er fast die Hälfte seiner Höhe verloren und zieht eine Schleppe aus kurzen Wellen hinter sich her, die nicht mitgekommen sind.

```
Spitze nach 12 m:  S = 1:   1.000000
                   S = 0,5: 0.5343
Schleppe (max |E| weit hinter dem Puls, S = 0,5): 0.0380
```

Mit $S = 1$ kommt der Puls **perfekt** an: Spitze $1{,}000000$, und hinter ihm ist das Feld auf Maschinengenauigkeit null — das ist der magische Zeitschritt aus Kapitel 5, und seine Erklärung sind wir gleich schuldig. Mit $S = 0{,}5$ dagegen ist die Spitze auf $0{,}53$ gesackt, und hinter dem Puls kräuselt eine **Wellenschleppe** her.

Die Schleppe verrät den Mechanismus. Ein schmaler Puls ist — das werden wir in Kapitel 10 mit der Fourier-Analyse präzise machen, die Idee genügt hier — ein Chor aus vielen Sinuswellen verschiedener Wellenlängen, die exakt im Takt starten. Im Vakuum bleibt der Chor für immer im Takt, weil **alle** Wellen mit demselben c laufen. Auf dem Gitter mit $S < 1$ offenbar nicht: Die langen Wellen laufen fast richtig und bilden den (gerundeten) Hauptpuls, die kurzen hinken hinterher und tröpfeln als Schleppe nach. Wenn unterschiedliche Wellenlängen unterschiedlich schnell laufen, heißt das **Dispersion** — hier **numerische Dispersion**, denn sie ist kein Naturphänomen, sondern ein Artefakt des Gitters. (Echte, physikalische Dispersion gibt es auch — Glas macht genau das mit Licht, daher zerlegt ein Prisma Weiß in Farben; davon handelt Kapitel 16. Das Gitter äfft sie nur nach.)

„Kurze Wellen sind zu langsam” ist bisher eine Beobachtung. Jetzt machen wir sie zu einer Formel mit Zahlen, die man messen kann.

8.4 Die Dispersionsrelation des Gitters

Die Zickzack-Rechnung war so bequem, weil die Welle unter dem Update ihre Form behielt und nur Amplituden übrig blieben. Derselbe Trick funktioniert für **jede** Gitterwelle. Wir setzen als Lösungs-Ansatz eine laufende Welle an, wie wir sie seit Kapitel 4 schreiben — an den e-Punkten und den (räumlich wie zeitlich versetzten) u-Punkten jeweils mit eigener Amplitude:

$$e(x,t) = E\ \sin(kx - \omega t), \qquad u(x,t) = U\ \sin(kx - \omega t).$$

Gesucht ist, welche Kombinationen aus Wellenzahl k und Kreisfrequenz ω das Update **zulässt**. Nehmen wir die erste Update-Zeile an einem u-Punkt am Ort x: Sie verknüpft die u-Werte zu den Zeiten $t \pm \Delta t/2$ mit den e-Werten an den Orten $x \pm \Delta x/2$ zur Zeit t:

$$U \sin\!\left(\Phi - \tfrac{\omega\Delta t}{2}\right) - U \sin\!\left(\Phi + \tfrac{\omega\Delta t}{2}\right) = -S\left[E \sin\!\left(\Phi + \tfrac{k\Delta x}{2}\right) - E \sin\!\left(\Phi - \tfrac{k\Delta x}{2}\right)\right],$$

wobei $\Phi = kx - \omega t$ die Phase in der Mitte des Schritts ist. Auf beiden Seiten steht dieselbe Struktur: eine Differenz zweier Sinuswerte, deren Argumente symmetrisch um Φ liegen. Dafür gibt es das passende Werkzeug aus der Schulmathematik, das Additionstheorem $\sin(\Phi \pm \delta) = \sin\Phi\cos\delta \pm \cos\Phi\sin\delta$ — in der Differenz fällt der $\sin\Phi$-Anteil weg:

$$\sin(\Phi + \delta) - \sin(\Phi - \delta) = 2\cos\Phi\ \sin\delta.$$

Damit wird die linke Seite zu $-2U\cos\Phi\ \sin(\omega\Delta t/2)$ und die rechte zu $-2SE\cos\Phi\ \sin(k\Delta x/2)$ — und nun passiert das Schönste an der ganzen Rechnung: $\cos\Phi$ **kürzt sich weg.** Ort und Zeit verschwinden aus der Gleichung; was an *einem* Punkt zu *einem* Zeitpunkt gilt, gilt automatisch überall und immer. Übrig bleibt

$$U\ \sin\frac{\omega\Delta t}{2} = S\,E\ \sin\frac{k\Delta x}{2}.$$

Die zweite Update-Zeile liefert mit derselben Rechnung das Spiegelbild $E\ \sin(\omega\Delta t/2) = S\,U\ \sin(k\Delta x/2)$. Beide zusammen erzwingen $U^2 = E^2$,

also $U = \pm E$ (Plus: Rechtsläufer, Minus: Linksläufer — der $B = \pm E/c$-Befund aus Kapitel 4 und 5, vom Gitter bestätigt), und vor allem die **Dispersionsrelation des Gitters**:

$$\boxed{\sin\frac{\omega\Delta t}{2} = S\,\sin\frac{k\Delta x}{2}}$$

Das ist die versprochene Formel mit den zwei Gesichtern. Sie verknüpft drei reine Zahlen (alles Skalare): den Zeit-Drehwinkel $\omega\Delta t$, den Orts-Drehwinkel $k\Delta x$ und die Courant-Zahl S. Für jede Wellenzahl k, die man hineinsteckt, legt sie fest, mit welcher Frequenz ω — und damit welchem Tempo $v = \omega/k$ — diese Welle auf dem Gitter läuft. Lesen wir sie dreimal:

Lesart 1 — die CFL-Grenze, noch einmal. Die linke Seite ist ein Sinus und kann höchstens 1 sein. Die rechte Seite wird am größten für die kürzeste Welle, den Zickzack ($k\Delta x = \pi$, der Sinus wird 1): Dort verlangt die Gleichung $\sin(\omega\Delta t/2) = S$. Für $S > 1$ gibt es **kein reelles** ω, das das leistet — die Zickzackwelle kann nicht schwingen, ihr bleibt nur das, was wir oben in der 2×2-Rechnung gefunden haben: wachsen. Spurkriterium und Dispersionsrelation sind dieselbe Physik in zwei Sprachen (die Eigenwert-Drehung pro Schritt *ist* der Winkel $\omega\Delta t$).

Lesart 2 — der magische Zeitschritt. Setze $S = 1$: Dann steht da $\sin(\omega\Delta t/2) = \sin(k\Delta x/2)$, also $\omega\Delta t = k\Delta x$ — und nach Division durch Δt: $\omega = k\,\Delta x/\Delta t = ck$. **Jede** Welle, egal wie kurz, läuft exakt mit $v = \omega/k = c$. Keine Dispersion, kein Formverlust, der Chor bleibt im Takt — darum kam der Puls oben unversehrt an, darum maß Kapitel 5 das Tempo auf Maschinengenauigkeit. Anschaulich: Bei $S = 1$ rückt die Welle pro Schritt exakt eine Zelle weiter; das Schema muss nichts interpolieren, es darf durchreichen.

Lesart 3 — Dispersion unterhalb der Grenze. Für $S < 1$ löst man nach ω auf (arcsin macht den Sinus rückgängig) und teilt durch k:

$$\frac{v}{c} = \frac{\omega}{ck} = \frac{2\arcsin\!\left(S\,\sin\frac{k\Delta x}{2}\right)}{S\,k\Delta x}.$$

Weil der Sinus unterwegs „durchhängt" (er wächst langsamer als sein Argument), kommt für jedes $k > 0$ etwas **kleiner als 1** heraus — und zwar umso deutlicher, je größer $k\Delta x$ ist, je weniger Gitterpunkte sich eine Wellenlänge also gönnt. Genau das ist die beobachtete Regel „kurze Wellen hinken".

Wie groß ist der Effekt in Zahlen? Die Handrechnung für eine ordentlich aufgelöste Welle — **8 Punkte pro Wellenlänge** — bei $S = 0{,}5$. Die Quelle gibt

die Frequenz vor, also rechnen wir von ω nach k (gleiche Formel, rückwärts gelesen — jeder Zwischenwert zum Nachrechnen):

```
S = 0.5
w_dt = 2 * np.pi * S / 8        # Zeit-Drehwinkel der Quelle: 0,39270
schritt1 = np.sin(w_dt / 2)     # sin(0,19635)              = 0,19509
schritt2 = schritt1 / S         # durch S                   = 0,39018
# np.arcsin: Umkehrung des Sinus (Bogenmass), Gegenstueck zu np.arccos
k_dx = 2 * np.arcsin(schritt2)  # Orts-Drehwinkel           = 0,80166
print(f"omega*dt = {w_dt:.5f}  ->  k*dx = {k_dx:.5f}")
print(f"Gitter-Wellenlänge: 2*pi/(k*dx) = {2*np.pi/k_dx:.4f} Zellen (statt 8)")
print(f"Tempo: v/c = (omega*dt)/(S*k*dx) = {w_dt/(S*k_dx):.5f}")
```

```
omega*dt = 0.39270  ->  k*dx = 0.80166
Gitter-Wellenlänge: 2*pi/(k*dx) = 7.8378 Zellen (statt 8)
Tempo: v/c = (omega*dt)/(S*k*dx) = 0.97972
```

Die Welle, die physikalisch 8 Zellen lang sein müsste, wird auf dem Gitter auf $7{,}84$ Zellen **gestaucht** und läuft mit $0{,}980\,c$ — zwei Prozent zu langsam. Klingt harmlos; ob es das ist, klärt der Praxis-Abschnitt am Ende.

8.5 Messung gegen Theorie

Eine hergeleitete Formel bekommt bei uns keinen Vertrauensvorschuss (Kapitel 2 und 3 haben es vorgemacht): Wir messen nach. Der Versuchsaufbau kommt aus dem Wellenlabor von Kapitel 6 — eine weiche Quelle speist einen Sinus mit sanft anschwellender Amplitude ein, und wenn der Wellenzug eingeschwungen ist, vermessen wir ihn. Die Quelle diktiert die Frequenz; was sich das Gitter aussucht, ist die **Wellenlänge**. Die messen wir über die Abstände der Nulldurchgänge (linear interpoliert zwischen den zwei Gitterpunkten mit Vorzeichenwechsel, gemittelt über hunderte Durchgänge), denn aus $v = f\lambda$ folgt direkt $v/c = \lambda_{\text{Gitter}}/\lambda_0$.

Vorhersage-Punkt

Drei Messläufe bei $S = 0{,}5$: Wellen mit 4, 8 und 16 Punkten pro Wellenlänge. Die Theoriekurve oben behauptet $v/c = 0{,}901$, $0{,}980$ und $0{,}995$. Bevor du die Messung startest: Welchem der drei Werte traust du am wenigsten — und in welche Richtung würde eine Abweichung zeigen, wenn die Herleitung irgendwo schlampt?

```
# von oben: leapfrog(), DX
def tempo_theorie(s, n_lambda):
    """v/c laut Dispersionsrelation für eine Quelle mit λ₀ = n_lambda·Δx."""
    w_dt = 2 * np.pi * s / n_lambda
    k_dx = 2 * np.arcsin(np.sin(w_dt / 2) / s)
    return w_dt / (s * k_dx)

def tempo_messung(s, n_lambda, nx=2600):
    """Sinusquelle einschwingen lassen, λ über Nulldurchgänge messen."""
    w_dt = 2 * np.pi * s / n_lambda
    anlauf = int(4 * 2 * np.pi / w_dt)             # 4 Perioden sanft anfahren
    quelle = lambda n: min(1.0, n / anlauf)**2 * np.sin(w_dt * n)
    schritte = int((nx - 600) / s)                 # Front bleibt vor dem Rand
    e = leapfrog(s, nx, schritte, quelle=quelle)
    seg = e[400:int(s * schritte) - 600]           # eingeschwungenes Fenster
    # np.sign: Vorzeichen (-1/0/+1); np.flatnonzero: Indizes der Treffer
    i = np.flatnonzero(np.sign(seg[:-1]) * np.sign(seg[1:]) < 0)
    null = i - seg[i] / (seg[i + 1] - seg[i])      # linear interpoliert
    lambda_gitter = 2 * np.mean(np.diff(null))     # in Zellen
    return lambda_gitter / n_lambda                # = v/c

print("Punkte/λ   gemessen   Theorie    Abweichung")
for n_lambda in (4, 8, 16):
    mess = tempo_messung(0.5, n_lambda)
    theo = tempo_theorie(0.5, n_lambda)
    print(f"   {n_lambda:2}      {mess:.4f}     {theo:.4f}    {abs(mess-theo):.1e}")
```

```
Punkte/λ   gemessen   Theorie    Abweichung
    4      0.9013     0.9011    2.4e-04
    8      0.9797     0.9797    1.3e-05
   16      0.9951     0.9951    1.2e-06
```

```
# von oben: tempo_theorie(), tempo_messung()
n_achse = np.linspace(3.0, 24, 300)
fig, ax = plt.subplots(figsize=(6.6, 3.4))
ax.plot(n_achse, [tempo_theorie(0.5, n) for n in n_achse], "tab:blue",
        label="Theorie: Dispersionsrelation (S = 0,5)")
ax.plot([4, 8, 16], [tempo_messung(0.5, n) for n in (4, 8, 16)], "ko",
        ms=7, label="gemessen (Nulldurchgänge)")
ax.axhline(1.0, color="gray", ls=":", lw=0.9, label="Lichtgeschwindigkeit c")
ax.set_xlabel("Punkte pro Wellenlänge")
ax.set_ylabel("v / c")
```

```
ax.legend(loc="lower right")
plt.tight_layout(); plt.show()

print(f"Gegenprobe magischer Zeitschritt — S = 1, 8 Punkte/λ: "
      f"v/c = {tempo_messung(1.0, 8):.6f}")
```

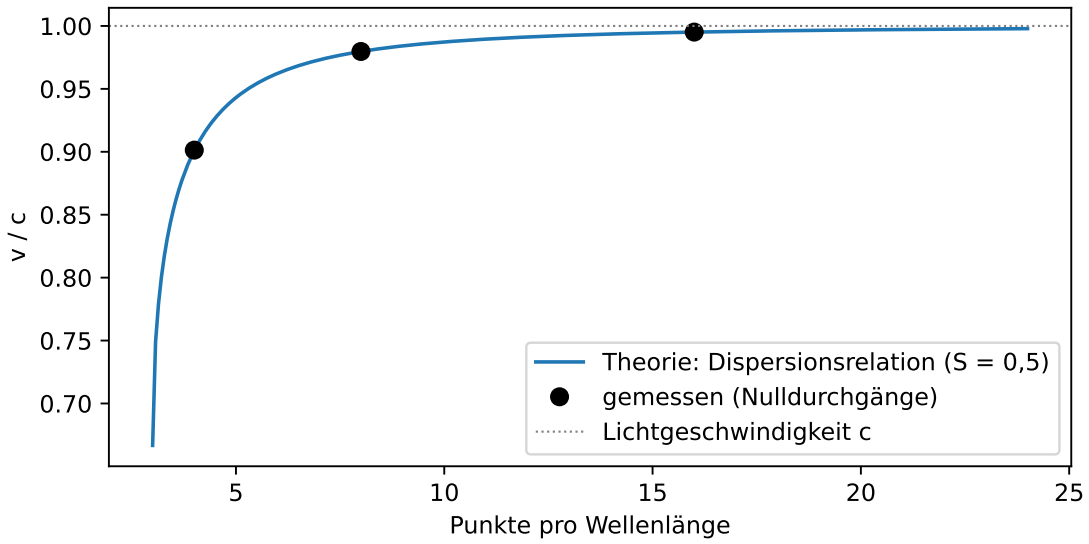


Abbildung 8.4: Wellentempo auf dem Gitter bei S = 0,5 als Funktion der Auflösung. Linie: die Dispersionsrelation; Punkte: die drei unabhängigen Messungen über Nulldurchgänge — sie liegen auf der Kurve (Abweichung unter einem Promille). Gepunktete Linie: das wahre c als eingezeichnete Erwartung. Links endet die Kurve bei 3 Punkten pro Wellenlänge: Noch kürzere Wellen kann eine Quelle bei S = 0,5 gar nicht als laufende Wellen anregen.

```
Gegenprobe magischer Zeitschritt — S = 1, 8 Punkte/λ: v/c = 1.000000
```

Messung und Theorie decken sich auf besser als ein Promille — die Herleitung hält. Und die Gegenprobe bestätigt Lesart 2: Bei $S = 1$ liefert dieselbe Messmaschine $v/c = 1{,}000000$.

Zwei Feinheiten am Rand der Abbildung verdienen einen Satz. Erstens endet die Theoriekurve links bei **3 Punkten pro Wellenlänge**: Für noch kürzere Wellen verlangt die Relation $\sin(k\Delta x/2) = \sin(\omega\Delta t/2)/S > 1$ — unmöglich, das Gitter verweigert den Transport (die Quelle erzeugt dann nur

ein örtlich gefangenes Zappeln). Zweitens fällt der Fehler mit jeder Verdopplung der Auflösung auf etwa ein Viertel — $9{,}9\,\%$, $2{,}0\,\%$, $0{,}49\,\%$. Das ist die Handschrift der **zweiten Ordnung** aus Kapitel 2: FDTD besteht aus zentralen Differenzen, und deren Fehler schrumpft mit dem Quadrat der Schrittweite.

8.6 Gitterwahl in der Praxis: Faustregeln

Wie fein muss das Gitter nun sein? Die ehrliche Antwort: Das hängt nicht nur von der Wellenlänge ab, sondern auch davon, **wie weit die Welle laufen soll** — der Tempofehler wirkt wie eine Uhr, die pro Tag drei Minuten nachgeht: Für eine Verabredung morgen ist das egal, nach einem Monat fehlt eine Stunde. Eine Welle, die mit $v/c = 0{,}98$ über L Wellenlängen läuft, kommt um $L\,(c/v - 1) \approx L \cdot 0{,}02$ Schwingungsperioden zu spät an; nach 50 Wellenlängen ist das eine **ganze Periode** — die Welle ist am Ziel komplett aus dem Takt, jede Interferenz-Vorhersage (Kapitel 14) wäre Unsinn. Daraus ergeben sich die Faustregeln:

Regel	Begründung
Mindestens 10–20 Punkte pro kürzester Wellenlänge	10 Punkte ≈ 1,3 % Tempofehler (bei $S = 0{,}5$) — gut für kurze Läufe
Lange Laufstrecken → feineres Gitter (Fehler wächst linear mit der Strecke)	L Wellenlängen × Tempofehler = Phasenfehler am Ziel (Ü 8.4)
S **so nah an die Stabilitätsgrenze wie erlaubt**	je größer S, desto schwächer die Dispersion; in 1D ist $S = 1$ exakt
Misstrauen bei scharfen Kanten	ein scharfer Puls *enthält* kurze Wellen — sie werden falsch transportiert (die Schleppe!)

Ein Beispiel mit echten Einheiten: WLAN bei $2{,}4$ GHz hat $\lambda = 12{,}5$ cm. „16 Punkte pro Wellenlänge" heißt dann $\Delta x \approx 8$ mm, und $S = 1$ erzwingt $\Delta t = \Delta x/c \approx 26$ ps — die Gitterwahl beginnt **immer** bei der kürzesten Wellenlänge im Problem, nie beim Speicherplatz.

Zum Schluss die Einschränkung, die schon mehrfach anklang: Das Privileg $S = 1$ ist ein **1D-Sonderfall**. In zwei und drei Dimensionen kann eine Welle auch diagonal übers Gitter laufen, und diagonal sind die Gitterpunkte weiter voneinander entfernt als achsenparallel — keine einzelne Zeitschrittwahl passt für beide Richtungen gleichzeitig. Die Stabilitätsgrenze rückt auf $S \leq 1/\sqrt{2}$ (2D) bzw. $1/\sqrt{3}$ (3D), einen magischen Zeitschritt gibt es nicht mehr,

und die Dispersion hängt zusätzlich von der **Laufrichtung** ab. Kapitel 9 führt das vor — die Werkzeuge von hier reichen dafür aus.

8.7 Das Kapitel-Programm

`programme/kap08/kap08_dispersion.py` bündelt alle vier Befunde eigenständig und prüft sie mit `assert`-Schranken: die Zickzack-Eigenwerte gegen die Spurformel (und gegen den Ü-7.4-Messwert), die Explosion als Zickzack (Nachbar-Korrelation $< -0{,}9$, Amplitude $> 10^{12}$), den Pulsvergleich (bei $S = 1$ Spitze auf 10^{-9} exakt, bei $S = 0{,}5$ Spitze $< 0{,}6$ und Schleppe $> 0{,}02$) und die Tempomessung gegen die Dispersionsrelation (Abweichung $< 5 \cdot 10^{-4}$).

Merkkasten

- **Courant-Bedingung** $S = c\,\Delta t/\Delta x \leq 1$: Das Schema reicht Information eine Zelle pro Schritt weiter — die Welle darf nicht schneller sein als der Bote.
- Das Versagen bei $S > 1$ hat eine Form: Die **Zickzackwelle** ($\lambda = 2\Delta x$) bekommt als erste einen reellen Eigenwert $|\lambda| > 1$; das Spurkriterium ihrer 2×2-Matrix legt die Grenze **exakt** auf $S = 1$.
- **Dispersionsrelation des Gitters:** $\sin(\omega\Delta t/2) = S\,\sin(k\Delta x/2)$ — eine Formel, zwei Gesichter: kein reelles ω für $S > 1$ (Explosion), zu kleines ω für $S < 1$ (kurze Wellen hinken — numerische Dispersion).
- **Magischer Zeitschritt:** Bei $S = 1$ folgt $\omega = ck$ für jede Welle — fehlerfreier Transport; ein 1D-Privileg (2D/3D: $S \leq 1/\sqrt{2},\ 1/\sqrt{3}$, keine Magie).
- **Faustregeln:** 10–20 Punkte pro Wellenlänge, mehr bei langen Laufstrecken (Phasenfehler = Strecke × Tempofehler); S nah an die Grenze; Fehler fällt mit $1/N^2$ (zweite Ordnung).

Roter Faden

Drei alte Fäden sind hier zusammengelaufen: Die **Fehler-schrumpft-quadratisch**-Messung aus Kapitel 2 erklärt das $1/N^2$ der Tempotabelle; der **magische Zeitschritt** aus Kapitel 5 ist eingelöst ($S = 1 \Rightarrow \omega = ck$); und die **Eigenwert-Messung** aus Übung 7.4 hat ihre Formel bekommen — der $1{,}87$-Sprung war die Zickzackwelle. Nach vorn: **Kapitel 9** geht in zwei Dimensionen ($S \leq 1/\sqrt{2}$, richtungsabhängige Dispersion), und **Kapitel**

10 macht mit der Fourier-Analyse den „Chor aus Sinuswellen” quantitativ, der hier die Schleppe erklärt hat. **Kapitel 14** (Interferenz) lebt davon, dass Phasen am Ziel stimmen — die Laufstrecken-Faustregel von hier ist dort Geschäftsgrundlage. Und in **Kapitel 16** begegnet uns Dispersion als *Physik*: Glas tut mit Licht, was das Gitter mit unseren Wellen tut — nur dass es dort kein Artefakt ist, sondern Materialeigenschaft.

Übungen

Ü 8.1 (Verstehen). Wie schnell läuft die Zickzackwelle ($\lambda = 2\Delta x$, also $k\Delta x = \pi$) bei $S = 0{,}5$? Rechne von Hand mit der Dispersionsrelation — als Kontrollwerte: $\sin(\pi/2) = 1$ und $\arcsin(1/2) = \pi/6$.

Musterlösung zu Ü 8.1

Die Relation verlangt $\sin(\omega\Delta t/2) = S \cdot \sin(\pi/2) = 0{,}5$, also $\omega\Delta t/2 = \pi/6$ und damit $\omega\Delta t = \pi/3$. Das Tempo:

$$\frac{v}{c} = \frac{\omega\Delta t}{S\,k\Delta x} = \frac{\pi/3}{0{,}5 \cdot \pi} = \frac{2}{3}.$$

Die kürzeste Gitterwelle läuft mit nur **zwei Dritteln der Lichtgeschwindigkeit** — der krasseste Fall der Tempolüge, noch unterhalb des linken Kurvenendes der Messabbildung (dort begann die Kurve erst bei 3 Punkten pro Wellenlänge; der Zickzack hat 2 und existiert nur als Anfangszustand, nicht als anregbare Welle). Die Kontrolle in einer Zeile:

```
print(f"v/c = {2*np.arcsin(0.5)/(0.5*np.pi):.6f}   (2/3 = {2/3:.6f})")
```

```
v/c = 0.666667   (2/3 = 0.666667)
```

Ü 8.2 (Verstehen). Das Vakuum hat keine Dispersion — alle Wellenlängen laufen exakt mit c. Das Gitter hat welche. Was *besitzt* das Gitter, das dem Vakuum fehlt und das diesen Unterschied erzwingt?

Musterlösung zu Ü 8.2

Einen eingebauten Maßstab. Das Gitter hat eine ausgezeichnete Länge (Δx) und eine ausgezeichnete Zeit (Δt); jede Welle wird daran gemessen — genau dafür steht die Größe „Punkte pro Wellenlänge”, und

die Dispersionsrelation enthält k und ω nur in den Kombinationen $k\Delta x$ und $\omega\Delta t$. Eine Welle mit 4 Punkten pro Wellenlänge ist auf dem Gitter etwas *objektiv anderes* als eine mit 16. Das Vakuum dagegen besitzt keinerlei eigene Länge oder Zeit: Es gibt nichts, womit es eine Wellenlänge vergleichen könnte, also kann es Wellen verschiedener Länge nicht verschieden behandeln — gleiches Tempo für alle ist die einzige maßstabsfreie Möglichkeit. (Glas hat wieder einen Maßstab — seine Atome und deren Resonanzen. Darum hat Glas Dispersion; Kapitel 16.)

Ü 8.3 (Verändern). Erweitere die Tempomessung um den Punkt **32 Punkte pro Wellenlänge** und vergleiche den Fehler $1-v/c$ mit dem bei 16 Punkten. Welchen Quotienten erwartest du nach der „zweiten Ordnung" aus Kapitel 2 — und welchen misst du?

Musterlösung zu Ü 8.3

Verdopplung der Auflösung sollte den Fehler **vierteln** (Fehler $\propto 1/N^2$):

```
# von oben: tempo_messung() (Abschnitt „Messung gegen Theorie")
fehler_16 = 1 - tempo_messung(0.5, 16)
fehler_32 = 1 - tempo_messung(0.5, 32)
print(f"Fehler bei 16 Punkten/λ: {fehler_16:.5f}")
print(f"Fehler bei 32 Punkten/λ: {fehler_32:.5f}")
print(f"Quotient: {fehler_16/fehler_32:.2f}   (zweite Ordnung: 4)")
```

```
Fehler bei 16 Punkten/λ: 0.00488
Fehler bei 32 Punkten/λ: 0.00121
Quotient: 4.04   (zweite Ordnung: 4)
```

Gemessen wird ein Quotient von rund **4** — die zweite Ordnung der zentralen Differenzen, jetzt zum dritten Mal im Buch bestätigt (Kapitel 2: einzelne Ableitung; hier: das Wellentempo). Wer den Fehler einer FDTD-Rechnung abschätzen will, kann also schlicht zweimal rechnen: Halbiert sich der Unterschied zur feineren Rechnung auf ein Viertel, ist man im sauberen Konvergenz-Regime.

Ü 8.4 (Übertragen). Du planst eine Simulation, in der ein Funksignal **50 Wellenlängen** weit läuft, und erlaubst am Ziel höchstens **ein Zehntel Periode** Phasenfehler ($S = 0{,}5$). Schreibe einen kleinen Gitterplaner: Wie viele Punkte pro Wellenlänge brauchst du? Reicht die 10-Punkte-Faustregel?

Musterlösung zu Ü 8.4

Eine Welle mit Tempo v braucht für die Strecke $L = 50\lambda$ die Zeit L/v statt L/c — die Verspätung, gemessen in Schwingungsperioden $T = \lambda/c$, ist $50\,(c/v - 1)$. Der Planer probiert Auflösungen durch, bis sie unter $0{,}1$ fällt:

```
# von oben: tempo_theorie() (Abschnitt „Messung gegen Theorie")
for n in range(4, 60):
    v = tempo_theorie(0.5, n)
    verspaetung = 50 * (1 / v - 1)          # in Perioden
    if verspaetung < 0.1:
        print(f"{n} Punkte pro Wellenlänge: Verspätung "
              f"{verspaetung:.3f} Perioden (v/c = {v:.5f})")
        break
```

```
25 Punkte pro Wellenlänge: Verspätung 0.099 Perioden (v/c = 0.99802)
```

Es braucht **25 Punkte pro Wellenlänge** — die 10-Punkte-Faustregel reicht hier *nicht* (sie ergäbe $50 \cdot 0{,}0129 \approx 0{,}64$ Perioden: das Signal käme mehr als eine halbe Schwingung verschoben an, bei Interferenz-Fragen der Unterschied zwischen Verstärkung und Auslöschung). Die Faustregel taugt für kurze Wege; bei langen zählt das Produkt aus Strecke und Tempofehler. Zwei Auswege, beide aus diesem Kapitel: feiner auflösen — oder S an die Grenze schieben (in 1D löst $S = 1$ das Problem exakt; in 2D/3D, wo es keinen magischen Zeitschritt gibt, bleibt nur das feinere Gitter).

9 FDTD in der Fläche: das volle Yee-Gitter

Vier Kapitel lang lebte unsere Welt auf einer Schnur. Das war kein Geiz, sondern Methode — jede Idee (Leapfrog, Quellen, Ränder, Stabilität, Dispersion) ließ sich in 1D sauber sezieren. Aber eine Schnur kennt nur links und rechts. Alles, was Funk und Optik spannend macht, braucht **Richtung und Form**: der Schatten hinter einer Wand, die Keule einer Antenne, die Beugung am Spalt, die Linse. Nichts davon existiert in 1D.

Dieses Kapitel macht den Sprung in die Fläche — und er ist kleiner, als er klingt: Der komplette 2D-Kern sind **drei** `np.diff`-Zeilen statt zwei. Dafür bekommen wir zum ersten Mal Feldbilder im Wortsinn (die Farbdarstellung, die Kapitel 4 versprochen hat), eine neue Stabilitätsgrenze an einer überraschenden Stelle, eine Dispersion, die von der **Laufrichtung** abhängt — und endlich die in Kapitel 6 zugesagte „volle Konstruktion" des reflexionsfreien Randes: die PML.

Lernziele

Nach diesem Kapitel kannst du …

1. … das 2D-Yee-Gitter aufbauen (E_z auf den Kreuzungen, die beiden B-Komponenten auf den Kanten) und den Zeitschritt in drei `np.diff`-Zeilen schreiben,
2. … eine Ringwelle simulieren, als Farbbild lesen und ihr Tempo gegen c messen,
3. … erklären und vorführen, warum die Stabilitätsgrenze in 2D bei $S \leq 1/\sqrt{2}$ liegt — und welche Welle dort kippt (das Schachbrett),
4. … die richtungsabhängige Dispersion messen: Bei $S = 1/\sqrt{2}$ laufen Wellen **diagonal exakt**, entlang der Achsen zu langsam,
5. … offene Ränder bauen: von der σ-Rampe zur Split-Feld-PML, mit gemessener Restreflexion.

9.1 Die halbe Welt genügt: das TMz-Gitter

In 3D haben die Felder sechs Komponenten (E_x, E_y, E_z und B_x, B_y, B_z). Wir betrachten eine Welt, die sich in z-Richtung nicht ändert — alles hängt nur von x und y ab, wie bei einem sehr langen geraden Draht oder einem Wellenbad zwischen zwei Glasplatten. Schreibt man die beiden Wirbelgleichungen aus Kapitel 3 komponentenweise auf und streicht alle z-Ableitungen, zerfallen die sechs Komponenten in **zwei Dreiergruppen, die nichts voneinander wissen**: (E_z, B_x, B_y) und (B_z, E_x, E_y). Es genügt, eine davon zu rechnen — wir nehmen die erste (sie heißt traditionell **TM**, „transversal magnetisch": das E-Feld steht senkrecht auf der Rechenebene, das B-Feld liegt in ihr). Von den Wirbelgleichungen bleibt dann:

$$\frac{\partial E_z}{\partial t} = c^2 \left(\frac{\partial B_y}{\partial x} - \frac{\partial B_x}{\partial y} \right), \qquad \frac{\partial B_x}{\partial t} = -\frac{\partial E_z}{\partial y}, \qquad \frac{\partial B_y}{\partial t} = +\frac{\partial E_z}{\partial x}.$$

Die erste Zeile ist die z-Komponente von „Ampère mit Verschiebungsstrom" — der vertraute Wirbelausdruck $\partial_x B_y - \partial_y B_x$ aus Kapitel 2. Die anderen beiden sind Faraday, Komponente für Komponente.

Wo wohnen die drei Größen auf dem Gitter? In 1D saß B **zwischen** den E-Punkten, damit jede Ableitung eine zentrale Differenz mit Spannweite Δx wird (das Yee-Argument aus Kapitel 5). Dasselbe Prinzip, jetzt auf einem Stadtplan: E_z wohnt auf den **Kreuzungen** (i, j). B_x braucht die y-Ableitung von E_z, also wohnt es auf halbem Weg zwischen zwei Kreuzungen in y-Richtung, bei $(i,\ j{+}\frac{1}{2})$. B_y braucht die x-Ableitung, also sitzt es bei $(i{+}\frac{1}{2},\ j)$. Jede Differenz im Update verbindet damit exakt die zwei nächsten Nachbarn der Zielgröße — alles bleibt zentral, nichts wird übersprungen.

Mit der Abkürzung aus Kapitel 7 — wir rechnen mit $u_x = c\,B_x$ und $u_y = c\,B_y$, die dieselbe Einheit wie E_z haben — bleibt wieder die Courant-Zahl $S = c\,\Delta t/\Delta x$ als einziger Koeffizient (wir setzen $\Delta y = \Delta x$):

```
import numpy as np
import matplotlib.pyplot as plt

DX = 0.01                       # Gitterabstand (m), Δy = Δx
S_2D = 1 / np.sqrt(2)           # warum genau dieser Wert: gleich!

def felder(nx, ny):
    """Leere TMz-Felder: e auf den Kreuzungen, ux/uy auf den Kanten."""
    e = np.zeros((nx, ny))      # E_z bei (i, j)
    ux = np.zeros((nx, ny - 1))  # c·B_x bei (i, j+1/2) — eine Spalte weniger
```

```
    uy = np.zeros((nx - 1, ny))  # c·B_y bei (i+1/2, j) — eine Zeile weniger
    return e, ux, uy

def schritt(e, ux, uy, s):
    """Ein Yee-Zeitschritt in 2D; Ränder bleiben fest (PEC)."""
    # axis-Argument: np.diff bildet Nachbar-Differenzen ENTLANG einer
    # Richtung — axis=0 läuft über i (x), axis=1 über j (y)
    ux -= s * np.diff(e, axis=1)        # Faraday, x-Komponente
    uy += s * np.diff(e, axis=0)        # Faraday, y-Komponente
    e[1:-1, 1:-1] += s * (np.diff(uy, axis=0)[:, 1:-1]
                          - np.diff(ux, axis=1)[1:-1, :])  # Ampère
    return e, ux, uy
```

Das ist der ganze Sprung von 1D nach 2D: Die Update-Zeile für e zieht jetzt **zwei** Differenzen heran (den Wirbel $\partial_x u_y - \partial_y u_x$ statt einer einzelnen Ableitung), und das Magnetfeld hat zwei Komponenten statt einer. Drei Zeilen, keine Matrix, kein Löser — Kapitel 7 lässt grüßen: explizit.

Das erste Experiment soll zeigen, was dieses Gitter mit einer punktförmigen Störung macht. Der Versuchsaufbau: **Die Bühne** ist ein Quadrat von 8 m × 8 m (801×801 Zellen à 1 cm), alle Felder starten auf null; die Ränder sind PEC-Wände, aber weit genug weg, dass im Beobachtungszeitraum nichts dort ankommt. Als Courant-Zahl nehmen wir $S = 1/\sqrt{2}$ — warum das die 2D-Grenze ist, klärt der nächste Abschnitt; hier genügt: Es ist stabil. **Die Anregung:** Statt einer Anfangsbedingung benutzen wir eine **weiche Quelle** im Zentrum (Kapitel 6: aufaddieren statt überschreiben), die genau ein zeitliches Gauß-Päckchen abgibt — ein kurzes „Pling" an einem einzigen Gitterpunkt, danach herrscht dort wieder Ruhe. **Gemessen wird** dreierlei: Schnappschüsse des ganzen Felds bei Schritt 180, 340 und 500; als eingezeichnete Erwartung in jedem Bild ein Kreis um die Quelle mit Radius $c \cdot t$ (mit t ab dem Quellen-Maximum); und das Tempo der Störung mit der Maximum-Verfolgung aus Kapitel 5 — wir notieren laufend den Ort des Wellenkamms auf einem Strahl nach rechts und legen eine Gerade durch die Ort-Zeit-Punkte.

> **Vorhersage-Punkt**
>
> Welches Muster entsteht aus dem „Pling", und mit welchem Tempo bewegt es sich? Und: Welche Farbe hat das Bild **innerhalb** des Musters, dort, wo das Pling längst vorbei ist?

```
# von oben: felder(), schritt(), DX, S_2D
nx = 801
```

```
mitte = nx // 2
def gauss(n, t0=30.0, tau=9.0):
    return np.exp(-((n - t0) / tau)**2)

e, ux, uy = felder(nx, nx)
schnappschuesse, zeiten, kamm_orte = {}, [], []
for n in range(521):
    e, ux, uy = schritt(e, ux, uy, S_2D)
    e[mitte, mitte] += gauss(n)
    if n in (180, 340, 500):
        schnappschuesse[n] = e.copy()
    if n >= 150 and n % 30 == 0:
        zeiten.append(n)
        kamm_orte.append(np.argmax(e[mitte, mitte:]))  # Kamm auf x-Strahl

fig, achsen = plt.subplots(1, 3, figsize=(7.4, 2.7))
ausdehnung = (0, nx * DX, 0, nx * DX)
winkel = np.linspace(0, 2 * np.pi, 200)
for ax, n in zip(achsen, schnappschuesse):
    bild = schnappschuesse[n]
    w = np.abs(bild).max()
    ax.imshow(bild.T, origin="lower", extent=ausdehnung, cmap="RdBu_r",
              vmin=-w / 3, vmax=w / 3)
    radius = S_2D * (n - 30) * DX               # c·t seit dem Quellen-Pling
    ax.plot(mitte * DX + radius * np.cos(winkel),
            mitte * DX + radius * np.sin(winkel), "k:", lw=0.9)
    ax.set_title(f"Schritt {n}", fontsize=9)
    ax.set_xlabel("x (m)")
achsen[0].set_ylabel("y (m)")
plt.tight_layout(); plt.show()

tempo = np.polyfit(zeiten, kamm_orte, 1)[0] / S_2D   # Zellen/Schritt → v/c
print(f"Kamm-Tempo (Maximum-Verfolgung wie in Kapitel 5): {tempo:.4f} c")
```

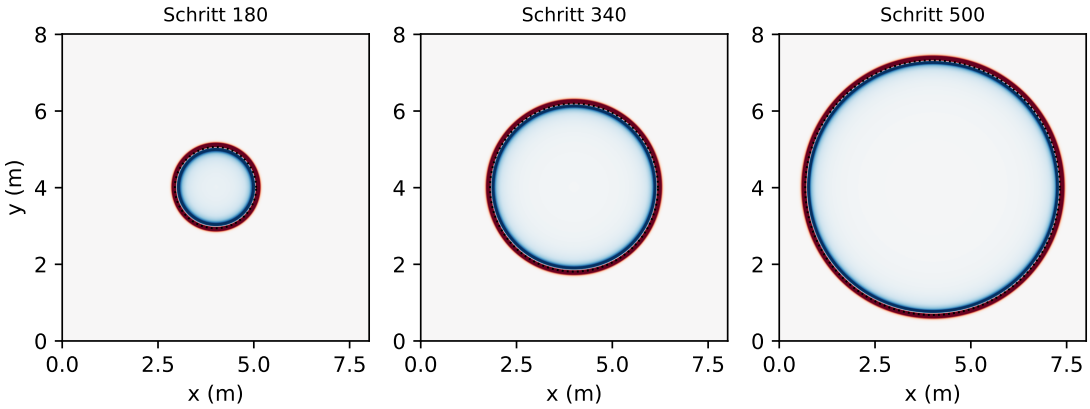


Abbildung 9.1: Die Ringwelle einer Punktquelle zu drei Zeiten — die erste Feld-Fotografie des Buchs (Lesehilfe: Farbe = Momentwert von E_z an jedem Ort; Rot positiv, Blau negativ, Weiß null — dieselbe ehrliche Darstellung wie das Farbband aus Kapitel 4, nur jetzt in echter Fläche). Der gepunktete Kreis ist die eingezeichnete Erwartung: Radius c·t um die Quelle. Der Wellenkamm sitzt in allen drei Bildern auf dem Kreis; innen ist es still.

```
Kamm-Tempo (Maximum-Verfolgung wie in Kapitel 5): 0.9967 c
```

Ein **Ring**, der mit Lichtgeschwindigkeit wächst — die Maximum-Verfolgung liefert $0{,}997\,c$, und der Kamm sitzt in jedem Schnappschuss auf dem $c \cdot t$-Kreis. Niemand hat dem Programm „Ring" beigebracht; es kennt nur Nachbar-Differenzen. Dass aus einem Punkt-Pling ein perfekter Kreis wird, ist die 2D-Ausgabe derselben Physik, die in 1D den Puls in zwei Hälften teilte: Die Störung läuft in **jede** Richtung gleich schnell davon.

Und die Farbe im Innern? Weiß — fast. Hinter dem Ring kehrt das Feld auf nahezu null zurück (hier bleibt ein Nachzügler von etwa einem halben Prozent des Kamms). Das winzige **Nachleuchten** ist kein Programmierfehler, sondern ehrliche 2D-Physik: Unsere Ebene ist ja ein Schnitt durch eine 3D-Welt, in der die „Punktquelle" eine unendlich lange Linie ist — und deren entfernte Abschnitte liefern Beiträge, die später eintreffen. In echtem 3D (Kugelwellen) gäbe es das nicht. Fürs Auge ist es unsichtbar; wir erwähnen es, damit du ihm nicht irgendwann misstraust.

9.2 Das neue Tempolimit

In 1D hieß die Regel $S \leq 1$, und der „magische" Wert $S = 1$ war sogar exakt. Gilt das hier weiter?

> ⚠ Naheliegende Vermutung
>
> **„$S \leq 1$ gilt auch in 2D. Das Informationsargument aus Kapitel 8 stimmt doch weiterhin: Pro Zeitschritt rückt die Information eine Zelle — in x wie in y."**
> *Warum sie naheliegt:* Pro Achse ist das Argument korrekt — die Update-Zeilen verbinden nach wie vor nur direkte Nachbarn, und entlang einer Gitterachse kommt eine Information höchstens eine Zelle pro Schritt voran. Nichts daran wird in 2D falsch.
> *Was stattdessen stimmt:* Der Engpass liegt **diagonal**. Ein Zeitschritt erreicht von einem Punkt aus nur die vier direkten Nachbarn — zusammen eine **Raute** $|\Delta i| + |\Delta j| \leq 1$. Die echte Welle breitet sich aber als Kreis aus, und der größte Kreis, der in diese Raute passt, hat nur den Radius $\Delta x/\sqrt{2}$ (von der Mitte senkrecht zur schrägen Rautenkante). Diagonal schafft das Schema pro Schritt also nur $\Delta x/\sqrt{2}$ — und damit muss $c\,\Delta t \leq \Delta x/\sqrt{2}$ sein:
>
> $$S \leq \frac{1}{\sqrt{2}} \approx 0{,}707.$$
>
> Wie in Kapitel 8 ist das erst ein Plausibilitätsargument. Den scharfen Beweis — und die Form des Versagens — liefert wieder die gefährlichste Gitterwelle.

In Kapitel 8 haben wir die Dispersionsrelation des 1D-Gitters hergeleitet; in der quadrierten Form lautete sie $\sin^2(\omega\Delta t/2) = S^2 \sin^2(k\Delta x/2)$. Führt man dieselbe Rechnung mit dem 2D-Ansatz $\sin(k_x x + k_y y - \omega t)$ durch, passiert genau das, was die dritte Update-Zeile vermuten lässt: Sie enthält **zwei** Differenzen, und jede steuert ihren eigenen Sinus-Term bei — die Beiträge der beiden Richtungen **addieren sich**:

$$\sin^2 \frac{\omega\Delta t}{2} = S^2 \left(\sin^2 \frac{k_x \Delta x}{2} + \sin^2 \frac{k_y \Delta x}{2} \right).$$

(Probe: Für $k_y = 0$ — eine Welle, die exakt in x-Richtung läuft — steht da wieder die 1D-Formel.) Die linke Seite kann höchstens 1 werden. Die rechte wird am größten, wenn **beide** Sinus gleich 1 sind: $k_x \Delta x = k_y \Delta x = \pi$, also

Zickzack in x- **und** y-Richtung zugleich — ein **Schachbrett**, bei dem jeder Gitterwert das Negativ aller vier Nachbarn ist. Für diese Welle verlangt die Relation $\sin(\omega\Delta t/2) = S\sqrt{2}$, und das geht nur reell, solange $S\sqrt{2} \leq 1$. Die Raute hatte recht, und zwar exakt.

Auch die Spur-Rechnung aus Kapitel 8 lässt sich wörtlich übertragen: Für das Schachbrett liefern die beiden Richtungen je einen $-4S^2$-Beitrag zur Spur der Schrittmatrix, $T = 2 - 8S^2$, und die Handrechnung für $S = 0{,}72$ (knapp über der Grenze, denn $8 \cdot 0{,}72^2 = 4{,}1472$):

$$T = -2{,}1472, \qquad |\lambda|_{\max} = \frac{2{,}1472 + \sqrt{2{,}1472^2 - 4}}{2} = \frac{2{,}1472 + 0{,}7813}{2} = 1{,}464.$$

Der Praxistest dazu, wieder mit vollständigem Aufbau: ein kleines Gitter von 101×101 Zellen (mehr braucht es nicht — die Explosion ist lokal), als Anfangsbedingung ein glatter 2D-Gauß-Hügel in der Mitte, PEC-Ränder, $S = 0{,}72$ — nur zwei Prozent über der Grenze $1/\sqrt{2} \approx 0{,}707$. Über 120 Schritte protokollieren wir den Maximalbetrag des Felds (aus seinen letzten 20 Schritten lesen wir den mittleren Wachstumsfaktor pro Schritt ab) und am Ende den Form-Detektor aus Kapitel 8, die Nachbar-Korrelation — jetzt zweimal, einmal für x- und einmal für y-Nachbarn. Danach läuft zur Gegenprobe derselbe Aufbau mit $S = 0{,}70$, knapp *unter* der Grenze.

> **Vorhersage-Punkt**
>
> Kapitel 8 lässt zwei Vorhersagen zu: das **Muster** des Versagens (und damit die beiden Korrelationswerte) und den **Faktor pro Schritt**. Lege dich auf beides fest.

```
# von oben: felder(), schritt()
n_klein = 101
xx, yy = np.meshgrid(np.arange(n_klein), np.arange(n_klein), indexing="ij")
huegel = np.exp(-(((xx - 50)**2 + (yy - 50)**2) / 12.0**2))

e, ux, uy = felder(n_klein, n_klein)
e += huegel
pegel = []
for n in range(120):
    e, ux, uy = schritt(e, ux, uy, 0.72)
    pegel.append(np.abs(e).max())
wachstum = (pegel[-1] / pegel[-21])**(1 / 20)    # mittlerer Faktor, 20 Schritte
```

```
fig, ax = plt.subplots(figsize=(4.6, 4.0))
ax.imshow(e[38:63, 38:63].T, origin="lower", cmap="RdBu_r")
ax.set_xlabel("Zelle i"); ax.set_ylabel("Zelle j")
plt.tight_layout(); plt.show()

brett = e[40:60, 40:60]
print(f"max|E| nach 120 Schritten: {pegel[-1]:.1e}")
print(f"Wachstum pro Schritt: {wachstum:.3f}   (Spurformel: 1.464)")
print(f"Nachbar-Korrelation: x-Richtung "
      f"{np.mean(brett[1:, :] * brett[:-1, :]) / np.mean(brett**2):+.3f},  "
      f"y-Richtung "
      f"{np.mean(brett[:, 1:] * brett[:, :-1]) / np.mean(brett**2):+.3f}")

e2, ux2, uy2 = felder(n_klein, n_klein)
e2 += huegel
for n in range(120):
    e2, ux2, uy2 = schritt(e2, ux2, uy2, 0.70)
print(f"Gegenprobe S = 0,70 (unter der Grenze): "
      f"max|E| = {np.abs(e2).max():.3f} — stabil")
```

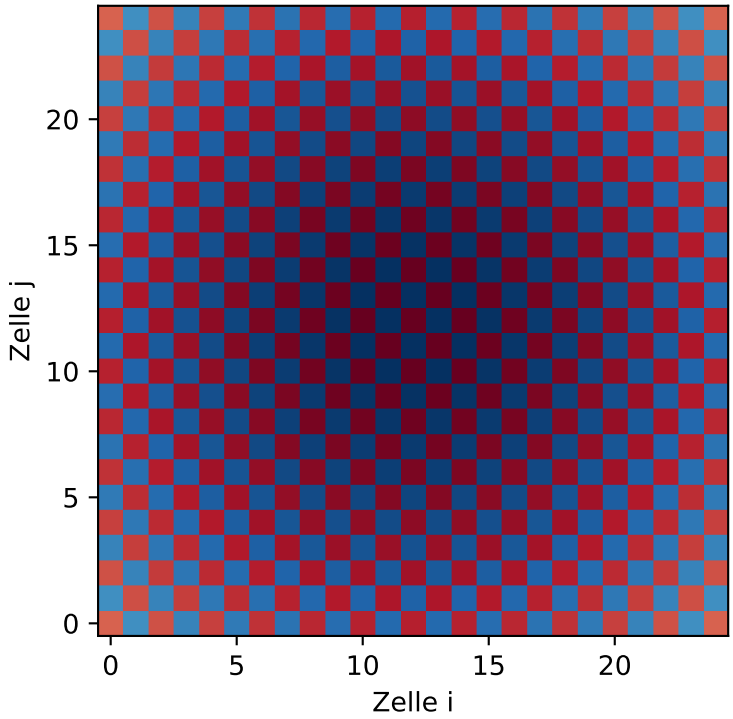


Abbildung 9.2: Das Feld nach 120 Schritten mit S = 0,72, Ausschnitt um das Zentrum (Lesehilfe: jedes Pixel ist ein Gitterwert; Rot positiv, Blau negativ). Aus dem glatten Gauß-Hügel ist ein Schachbrett geworden — der Zickzack aus Kapitel 8 in beiden Richtungen zugleich. Sein gemessener Wachstumsfaktor 1,453 pro Schritt liegt knapp ein Prozent unter der Spurformel (1,464) — die kleine Lücke ist der Randeffekt aus Kapitel 8.

```
max|E| nach 120 Schritten: 5.4e+02
Wachstum pro Schritt: 1.453   (Spurformel: 1.464)
Nachbar-Korrelation: x-Richtung -1.010,  y-Richtung -1.010
Gegenprobe S = 0,70 (unter der Grenze): max|E| = 0.243 — stabil
```

Muster und Faktor, beides nach Fahrplan: Schachbrett (Nachbar-Korrelation ≈ -1 in beiden Richtungen) mit Wachstum $1{,}45$ pro Schritt, und zwei Hundertstel darunter, bei $S = 0{,}70$, passiert — nichts. Die 1D-Werkzeuge aus Kapitel 8 tragen unverändert; nur die Zahl an der Grenze ist neu.

9.3 Die Magie wandert in die Diagonale

In 1D gab es den magischen Zeitschritt: Bei $S = 1$ liefen *alle* Wellen exakt mit c. Diese Tür ist in 2D zu — $S = 1$ ist jenseits der Stabilitätsgrenze. Was passiert am neuen Maximum $S = 1/\sqrt{2}$? Die Dispersionsrelation kennt jetzt nicht nur eine Wellenlänge, sondern auch eine **Laufrichtung**, und die beiden Extremfälle lohnen die Handrechnung.

Entlang einer Achse ($k_y = 0$) gilt die 1D-Formel mit $S = 1/\sqrt{2} < 1$: Dispersion wie in Kapitel 8, kurze Wellen hinken. **Entlang der Diagonalen** ist $k_x = k_y = k/\sqrt{2}$ (der Wellenvektor der Länge k verteilt sich gleichmäßig auf beide Achsen), die beiden gleichen Sinus-Terme addieren sich, und die Relation wird zu

$$\sin\frac{\omega\Delta t}{2} = S\sqrt{2}\,\sin\left(\frac{k\Delta x}{2\sqrt{2}}\right).$$

Jetzt setze $S = 1/\sqrt{2}$ ein: Der Vorfaktor $S\sqrt{2}$ wird **exakt 1**, also $\omega\Delta t = k\Delta x/\sqrt{2}$ — und mit $\Delta t = \Delta x/(c\sqrt{2})$ folgt $\omega = c\,k$ für **jede** Wellenlänge. Das ist Wort für Wort die Rechnung des magischen Zeitschritts aus Kapitel 8, nur dass sie jetzt der Diagonale gehört: **Bei $S = 1/\sqrt{2}$ laufen diagonale Wellen exakt, achsenparallele zu langsam.** Anschaulich passt das zur Raute: Diagonal ist der Schritt des Schemas gerade so groß wie der Weg der Welle — dort wird durchgereicht statt interpoliert.

> **Vorhersage-Punkt**
>
> Wir treiben gleich eine Sinus-Punktquelle mit 5 Punkten pro Wellenlänge bei $S = 1/\sqrt{2}$ und vermessen die Gitterwellenlänge auf zwei Strahlen — exakt mit der Nulldurchgangs-Methode aus Kapitel 8. Die 1D-Formel sagt für die Achse $v/c = 0{,}962$ voraus, die Diagonal-Formel $v/c = 1{,}000$. Vier Prozent Unterschied je nach Richtung: Welche Form hat dann der „Ring" dieser Quelle, wenn er ein Stück gelaufen ist?

```
# von oben: felder(), schritt(), DX, S_2D
def lambda_aus_strahl(strahl, abstand):
    """Wellenlänge über Nulldurchgänge (Methode aus Kapitel 8)."""
    i = np.flatnonzero(np.sign(strahl[:-1]) * np.sign(strahl[1:]) < 0)
    null = i - strahl[i] / (strahl[i + 1] - strahl[i])
    return 2 * np.mean(np.diff(null)) * abstand

nx = 701
```

```
m = nx // 2
n_lambda = 5
w_dt = 2 * np.pi * S_2D / n_lambda
anlauf = int(4 * 2 * np.pi / w_dt)
e, ux, uy = felder(nx, nx)
for n in range(460):
    e, ux, uy = schritt(e, ux, uy, S_2D)
    e[m, m] += min(1.0, n / anlauf)**2 * np.sin(w_dt * n)

a, b = 60, 260                                  # Messfenster (Zellen)
strahl_achse = e[m, m + a:m + b]
strahl_diag = np.array([e[m + k, m + k]         # Schritt entlang i = j:
                        for k in range(int(a / np.sqrt(2)),
                                       int(b / np.sqrt(2)))])
v_achse = lambda_aus_strahl(strahl_achse, 1.0) / n_lambda
v_diag = lambda_aus_strahl(strahl_diag, np.sqrt(2)) / n_lambda

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7.4, 3.4),
                               gridspec_kw={"width_ratios": [1, 1.3]})
ii, jj = np.meshgrid(np.arange(nx), np.arange(nx), indexing="ij")
# Broadcasting/`np.hypot`: Abstand jedes Punkts vom Zentrum
rr = np.hypot(ii - m, jj - m)
bild = e * np.sqrt(rr + 1)                      # 1/√r-Abfall ausgeglichen
w = np.abs(bild[m, m + a:m + b]).max()
ax1.imshow(bild.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w)
ax1.set_xlabel("Zelle i"); ax1.set_ylabel("Zelle j")

abstand = np.arange(a, b) * DX
ax2.plot(abstand, strahl_achse, "tab:blue", lw=1.0, label="Achsenstrahl")
ax2.plot(np.arange(int(a / np.sqrt(2)), int(b / np.sqrt(2)))
         * np.sqrt(2) * DX, strahl_diag, "tab:orange", lw=1.0,
         label="Diagonalstrahl")
ax2.set_xlim(1.2, 1.75)                         # Zoom: Drift der Kämme
ax2.set_xlabel("Abstand von der Quelle (m)"); ax2.set_ylabel("E (V/m)")
ax2.legend(fontsize=8)
plt.tight_layout(); plt.show()

print(f"5 Punkte/λ:  Achse     v/c = {v_achse:.4f}   (1D-Formel: 0.9617)")
print(f"             Diagonale v/c = {v_diag:.4f}   (Diagonal-Formel: 1.0000)")
```

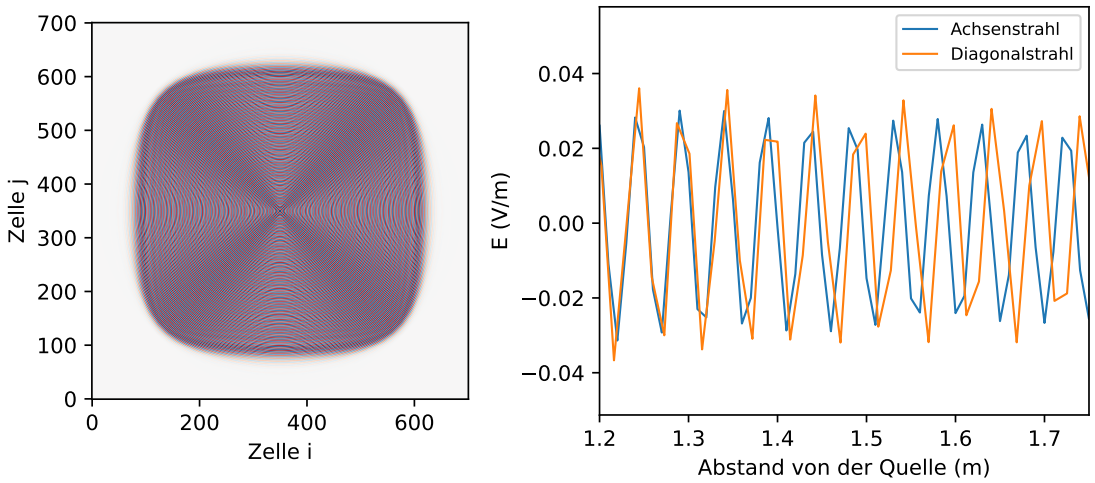


Abbildung 9.3: Links: das Feld der Sinusquelle (5 Punkte pro Wellenlänge, S = 1/√2) — zur Lesbarkeit ist die Farbe mit √r aufgehellt, weil die Ringamplitude nach außen abnimmt (Lesehilfe: Farbe = E_z·√r; die Form zählt, nicht die Helligkeit). Der „Ring" ist ein abgerundetes Quadrat auf der Spitze: diagonal am weitesten, weil die Wellen dort exakt laufen. Rechts der Schnitt durch beide Strahlen: Auf der Diagonale (orange) bleiben die Wellenzüge länger — auf gleicher Strecke sammelt der Achsenstrahl (blau) sichtbar mehr Schwingungen an, seine Wellenlänge ist gestaucht.

```
5 Punkte/λ:  Achse     v/c = 0.9613   (1D-Formel: 0.9617)
             Diagonale v/c = 1.0001   (Diagonal-Formel: 1.0000)
```

Die Messung bestätigt beide Formeln auf drei bis vier Nachkommastellen — und das Farbbild zeigt, was vier Prozent Richtungsunterschied anrichten: Die Wellenfronten bilden kein Kreis mehr, sondern ein **abgerundetes Quadrat auf der Spitze**. Damit ist der Themenfaden aus Kapitel 8 zu Ende erzählt: Numerische Dispersion macht kurze Wellen langsam, und in 2D tut sie es **richtungsabhängig** — ein simulierter Stern funkelt auf groben Gittern viereckig.

Für die Praxis heißt das: In 2D/3D gibt es **keinen** Zeitschritt, der alle Richtungen exakt macht. Die Faustregeln aus Kapitel 8 bleiben — S nah an die Grenze, 10–20 Punkte pro Wellenlänge, mehr bei langen Laufstrecken —, aber das bequeme „in 1D löst $S = 1$ alles" ist Geschichte. Gegen Richtungsfehler hilft nur feiner auflösen (oder klügere Schemata, von denen Kapitel 11 erzählt).

9.4 Offene Ränder, jetzt wirklich: die PML

Bisher enden alle unsere 2D-Gitter in PEC-Wänden: Der Ring läuft hinaus, wird reflektiert und vermüllt die Szene. In 1D half der Mur-Trick (Randpunkt übernimmt den Nachbarwert) — aber er lebte davon, dass die Welle **senkrecht** und **mit bekanntem Tempo** ankommt. In 2D trifft sie unter jedem Winkel ein; ein Patentrezept pro Randpunkt gibt es nicht mehr. Kapitel 6 hat den besseren Weg schon vorgezeichnet: eine **Rampe aus verlustigem Material**, die die Welle schluckt, bevor sie die Wand erreicht — und versprochen, die „volle Konstruktion" in diesem Kapitel nachzuliefern. Wir lösen in zwei Stufen ein und **messen** jede: Ein Ring läuft in den Rand, danach notieren wir, was im Innenbereich an Feld übrig bleibt (relativ zur PEC-Wand als schlechtestem Fall).

Stufe 1, die σ-Rampe aus Kapitel 6: ein 20 Zellen breiter Rahmen, in dem die Leitfähigkeit quadratisch anwächst und nur das E-Update dämpft (die `ca`/`cb`-Koeffizienten aus Kapitel 6, jetzt als 2D-Felder). Das hilft — aber es bleibt ein messbares Echo: Ein nur elektrisch dämpfender Rahmen hat einen anderen Wellenwiderstand als das Vakuum davor (die Welle spürt den Materialwechsel), und schräg einfallende Wellen verbringen lange Wege in der flachen Rampenzone.

Stufe 2, die Split-Feld-PML (Bérenger 1994): zwei Ideen auf einmal. Erstens bekommt der Verlust einen **magnetischen Zwilling** — auch die u-Updates werden gedämpft, abgestimmt im Verhältnis $\sigma_m/\mu_0 = \sigma/\varepsilon_0$, sodass der Wellenwiderstand der Schicht exakt dem des Vakuums gleicht (das fehlende Stück aus Kapitel 6). Zweitens wird E_z buchhalterisch in zwei Teilfelder zerlegt, $E_z = E_{zx} + E_{zy}$: E_{zx} sammelt den $\partial_x u_y$-Anteil des Wirbels, E_{zy} den $\partial_y u_x$-Anteil — und **jedes Teilfeld wird nur von „seinem" σ-Profil gedämpft** (σ_x wächst nur zu den x-Rändern hin, σ_y nur zu den y-Rändern). Eine schräg einlaufende Welle wird so am x-Rand genau in dem Anteil gedämpft, der in x-Richtung läuft, während ihr Tangentialanteil ungestört weiterläuft, statt reflektiert zu werden. Physik ist diese Zerlegung nicht mehr — innerhalb der Schicht gilt nicht mehr Maxwell, sondern ein eigens konstruiertes Gleichungssystem, dessen einziger Daseinszweck das reflexionsfreie Schlucken ist.

```
# von oben: felder(), schritt(), gauss(), S_2D
def restfeld(rand, nx=241, dicke=20, schritte=400, s=S_2D, smax=0.25):
    """Ring läuft in den Rand; Restfeld im Innern nach dem Durchgang."""
    m = nx // 2
    r_e = np.arange(nx, dtype=float)             # σ-Profil: quadratische
    prof_e = (np.maximum(0, (dicke - r_e) / dicke)**2          # Rampe an
              + np.maximum(0, (r_e - (nx - 1 - dicke)) / dicke)**2)  # beiden
```

```
    r_u = np.arange(nx - 1) + 0.5                # ... und an den u-Plätzen
    prof_u = (np.maximum(0, (dicke - r_u) / dicke)**2
              + np.maximum(0, (r_u - (nx - 1 - dicke)) / dicke)**2)

    def koeff(sigma):                            # ca/cb wie in Kapitel 6
        return (1 - sigma) / (1 + sigma), 1 / (1 + sigma)

    if rand == "pml":
        ezx, ezy = np.zeros((nx, nx)), np.zeros((nx, nx))
        ux, uy = np.zeros((nx, nx - 1)), np.zeros((nx - 1, nx))
        # [:, None] (Broadcasting): macht aus dem 1D-Profil eine Spalte,
        # die NumPy über alle Spalten/Zeilen des 2D-Felds ausbreitet
        cax, cbx = [k[:, None] for k in koeff(smax * prof_e)]   # σx(x)
        cay, cby = [k[None, :] for k in koeff(smax * prof_e)]   # σy(y)
        caux, cbux = [k[None, :] for k in koeff(smax * prof_u)]
        cauy, cbuy = [k[:, None] for k in koeff(smax * prof_u)]
        for n in range(schritte):
            e = ezx + ezy                        # das physikalische Feld
            ux = caux * ux - cbux * s * np.diff(e, axis=1)
            uy = cauy * uy + cbuy * s * np.diff(e, axis=0)
            ezx[1:-1, :] = (cax[1:-1] * ezx[1:-1, :]
                            + cbx[1:-1] * s * np.diff(uy, axis=0))
            ezy[:, 1:-1] = (cay[:, 1:-1] * ezy[:, 1:-1]
                            - cby[:, 1:-1] * s * np.diff(ux, axis=1))
            ezx[m, m] += 0.5 * gauss(n)
            ezy[m, m] += 0.5 * gauss(n)
        e = ezx + ezy
    else:                                        # "pec" oder "rampe"
        e, ux, uy = felder(nx, nx)
        sigma = (smax * (prof_e[:, None] + prof_e[None, :])
                 if rand == "rampe" else np.zeros((nx, nx)))
        ca, cb = koeff(sigma)
        for n in range(schritte):
            ux -= s * np.diff(e, axis=1)
            uy += s * np.diff(e, axis=0)
            wirbel = (np.diff(uy, axis=0)[:, 1:-1]
                      - np.diff(ux, axis=1)[1:-1, :])
            e[1:-1, 1:-1] = (ca[1:-1, 1:-1] * e[1:-1, 1:-1]
                             + cb[1:-1, 1:-1] * s * wirbel)
            e[m, m] += gauss(n)
    return np.abs(e[60:-60, 60:-60]).max()

reste = {rand: restfeld(rand) for rand in ("pec", "rampe", "pml")}
```

```
print("Restfeld im Innern, nachdem der Ring den Rand passiert hat:")
for rand, wert in reste.items():
    print(f"  {rand:6}: {wert:.2e}   ({wert / reste['pec'] * 100:7.3f} % "
          f"der PEC-Reflexion)")
```

```
Restfeld im Innern, nachdem der Ring den Rand passiert hat:
  pec   : 1.13e-02   (100.000 % der PEC-Reflexion)
  rampe : 1.68e-03   ( 14.913 % der PEC-Reflexion)
  pml   : 4.40e-05   (  0.391 % der PEC-Reflexion)
```

Die Messleiter: Die σ-Rampe drückt das Echo auf rund **15 %** der PEC-Reflexion, die PML auf **0,4 %** — noch einmal fast vierzigmal leiser, bei gleicher Rahmendicke. Wer mehr will, dreht an Dicke und σ-Profil; in den Werkzeugen von Teil III ist genau diese Schicht (in modernisierter Form) eingebaut, und du weißt jetzt, was sie tut und woran du eine zu dünn geratene erkennst: am Echo.

9.5 Ausblick: die dritte Dimension

Der Schritt von 2D nach 3D bringt keine neue Idee mehr, nur mehr Buchhaltung: Alle sechs Komponenten leben, E wohnt auf den **Kanten** eines Würfelgitters, B auf den **Flächenmitten** — die berühmte **Yee-Zelle**, seit 1966 unverändert das Herz jedes FDTD-Programms. Drei Richtungen addieren drei Sinus-Terme in der Dispersionsrelation, die Grenze rückt auf $S \leq 1/\sqrt{3}$, und Dispersion gibt es entlang dreier Achsen-, Flächen- und Raumdiagonalen-Familien. Der wahre Unterschied ist der Preis: Ein Gitter mit 700 Punkten pro Kante hat in 2D eine halbe Million Zellen, in 3D **343 Millionen** — und weil mit dem Gitter auch die Schrittzahl wächst, skaliert der Aufwand wie N^4. Ab hier lohnt es sich, die Handarbeit an professionelle Löser zu übergeben. Genau dorthin sind wir unterwegs: Kapitel 10 baut zuvor noch die zwei Messinstrumente, ohne die man keinem Löser trauen sollte — Spektren und Konvergenztests.

9.6 Das Kapitel-Programm

`programme/kap09/kap09_yee_2d.py` bündelt alle vier Befunde eigenständig und prüft sie mit `assert`-Schranken: Kamm-Tempo der Ringwelle ($|v/c - 1| < 0{,}01$), Schachbrett-Explosion (Wachstum auf $0{,}02$ an der Spurformel, Korrelation $< -0{,}9$ in beiden Richtungen, Gegenprobe stabil), Richtungsmessung

gegen beide Formeln ($< 2 \cdot 10^{-3}$) und die Rand-Messleiter (Rampe $< 25\,\%$ von PEC, PML $< 5\,\%$ der Rampe).

Merkkasten

- **2D-Yee (TMz):** E_z auf den Kreuzungen, B_x/B_y auf den Kanten dazwischen — jede Differenz bleibt zentral; der ganze Zeitschritt sind drei `np.diff`-Zeilen.
- **Stabilität:** Die Beiträge der Richtungen addieren sich in der Dispersionsrelation; die gefährlichste Welle ist das **Schachbrett**, die Grenze $S \leq 1/\sqrt{2}$ (3D: $1/\sqrt{3}$). Anschaulich: Ein Schritt erreicht nur die Nachbar-**Raute**, diagonal ist sie am engsten.
- **Richtungsabhängige Dispersion:** Bei $S = 1/\sqrt{2}$ laufen diagonale Wellen exakt, achsenparallele zu langsam — Ringe werden auf groben Gittern eckig. Einen für alle Richtungen magischen Zeitschritt gibt es ab 2D nicht.
- **Offene Ränder:** σ-Rampe = gut, **PML** = Rampe plus magnetischer Zwilling ($\sigma_m/\mu_0 = \sigma/\varepsilon_0$) plus Feld-Splitting mit richtungsgebundenen Profilen — gemessen ~40-mal leiser als die Rampe allein.

Roter Faden

Fast alles hier war Wiederverwendung: Der **Yee-Versatz** aus Kapitel 5 wurde vom Stab zum Stadtplan; **Spurkriterium und Dispersionsrelation** aus Kapitel 8 lieferten Schachbrett-Faktor und Diagonal-Magie; die **σ-Rampe** aus Kapitel 6 bekam ihren versprochenen magnetischen Zwilling und wurde zur PML; und die **Farb-Wahlregel** aus Kapitel 4 hat ihre ersten echten Feldbilder eingelöst. Nach vorn: **Kapitel 10** macht aus Zeitsignalen Spektren (der „Chor aus Sinuswellen" wird messbar) und aus Gitterverfeinerung einen Vertrauenstest. **Kapitel 12** schickt Wellen schräg auf Materialgrenzen (Brechung), **Kapitel 14** durch Doppelspalte — beides auf genau dem Gitter, das du in diesem Kapitel gebaut hast.

Übungen

Ü 9.1 (Verstehen). Begründe ohne Rechnung, warum die Stabilitätsgrenze in 3D bei $S \leq 1/\sqrt{3}$ liegt — einmal über die Dispersionsrelation, einmal über das Rauten-Argument. Was ist die gefährlichste Welle in 3D?

Musterlösung zu Ü 9.1

Über die Relation: In 3D addieren sich **drei** $\sin^2$-Terme; im schlimmsten Fall (alle drei Sinus gleich 1, also Zickzack in x, y **und** z — ein dreidimensionales Schachbrett, bei dem jeder Wert das Negativ seiner sechs Nachbarn ist) verlangt die linke Seite $\sin(\omega\Delta t/2) = S\sqrt{3} \leq 1$. **Über die Raute:** Ein Zeitschritt erreicht in 3D nur den Oktaeder $|\Delta i| + |\Delta j| + |\Delta k| \leq 1$; die größte einbeschriebene Kugel hat den Radius $\Delta x/\sqrt{3}$ (Abstand des Mittelpunkts zur schrägen Dreiecksfläche). Pro Dimension kommt ein Summand unter der Wurzel hinzu — die Grenze ist allgemein $1/\sqrt{d}$, weil die Raumdiagonale mit jeder Dimension länger wird, der Ein-Schritt-Horizont des Schemas aber nicht.

Ü 9.2 (Verstehen). In 1D behielt der laufende Puls seine volle Höhe. Die Ringwelle wird dagegen nach außen leiser. Mit welchem Gesetz muss ihre Amplitude abfallen? (Denke an die Energie — sie verteilt sich auf den wachsenden Umfang.) Prüfe deine Antwort an den Kammhöhen der Ring-Simulation.

Musterlösung zu Ü 9.2

Die Energie des Rings verteilt sich auf einen Umfang $2\pi r$, die Energie**dichte** fällt also wie $1/r$ — und weil Energie quadratisch in der Amplitude steckt (Kapitel 4), fällt die **Amplitude wie** $1/\sqrt{r}$. Dann muss $E_{\text{Kamm}} \cdot \sqrt{r}$ konstant sein:

```
# von oben: felder(), schritt(), gauss(), S_2D
nx = 801
m = nx // 2
e, ux, uy = felder(nx, nx)
radien, hoehen = [], []
for n in range(521):
    e, ux, uy = schritt(e, ux, uy, S_2D)
    e[m, m] += gauss(n)
    if n >= 180 and n % 60 == 0:
        strahl = e[m, m:]
        r = np.argmax(strahl)
        radien.append(r); hoehen.append(strahl[r])
produkt = np.array(hoehen) * np.sqrt(np.array(radien))
print("Radius (Zellen):        ", radien)
print("Kammhöhe · sqrt(Radius):", np.round(produkt, 4))
```

```
Radius (Zellen):      [np.int64(109), np.int64(151), np.int64(193), np.int64(235), np.in
Kammhöhe · sqrt(Radius): [0.0973 0.097  0.0961 0.0944 0.0942 0.0933]
```

Das Produkt bleibt nahezu konstant, während der Radius um den Faktor drei wächst — das $1/\sqrt{r}$-Gesetz trägt. (Der leichte Restabfall von wenigen Prozent kommt von der numerischen Dispersion, die den Kamm unterwegs etwas verbreitert.) In 3D verteilt sich die Energie auf Kugel**flächen** $4\pi r^2$, dort fällt die Amplitude wie $1/r$ — das wird in Kapitel 21 zur Reichweitenformel des Funks.

Ü 9.3 (Verändern). Lass die Sinusquelle aus der Richtungsmessung mit nur **4 Punkten pro Wellenlänge** laufen (bei $S = 1/\sqrt{2}$). Berechne zuerst aus den beiden Formeln v/c für Achse und Diagonale — und sieh dir dann die Form der Wellenfronten an.

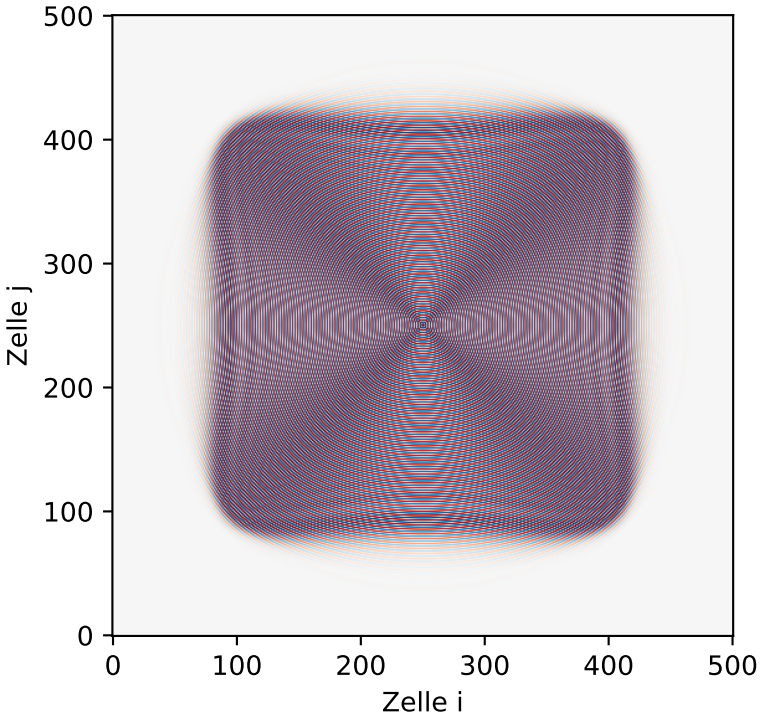
ℹ Musterlösung zu Ü 9.3

```python
# von oben: felder(), schritt(), lambda_aus_strahl(), DX, S_2D
n_lambda = 4
w_dt = 2 * np.pi * S_2D / n_lambda
k_achse = 2 * np.arcsin(np.sin(w_dt / 2) / S_2D)
k_diag = 2 * np.sqrt(2) * np.arcsin(np.sin(w_dt / 2) / (np.sqrt(2) * S_2D))
print(f"Theorie: Achse v/c = {w_dt / (S_2D * k_achse):.4f},   "
      f"Diagonale v/c = {w_dt / (S_2D * k_diag):.4f}")

nx = 501
m = nx // 2
anlauf = int(4 * 2 * np.pi / w_dt)
e, ux, uy = felder(nx, nx)
for n in range(330):
    e, ux, uy = schritt(e, ux, uy, S_2D)
    e[m, m] += min(1.0, n / anlauf)**2 * np.sin(w_dt * n)

fig, ax = plt.subplots(figsize=(4.6, 4.0))
ii, jj = np.meshgrid(np.arange(nx), np.arange(nx), indexing="ij")
bild = e * np.sqrt(np.hypot(ii - m, jj - m) + 1)
w = np.abs(bild[m, m + 50:m + 200]).max()
ax.imshow(bild.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w)
ax.set_xlabel("Zelle i"); ax.set_ylabel("Zelle j")
plt.tight_layout(); plt.show()
```

```
Theorie: Achse v/c = 0.9333,   Diagonale v/c = 1.0000
```



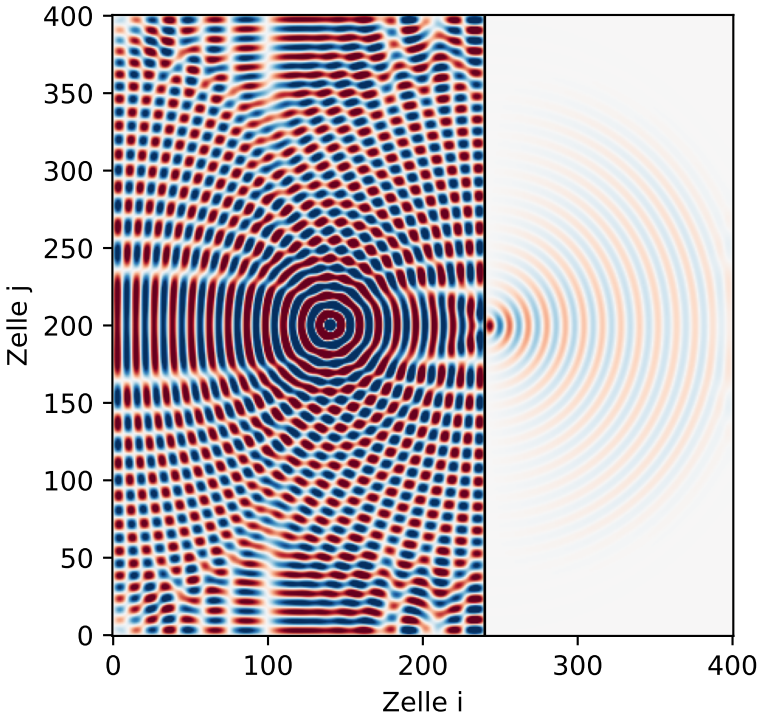
Die Theorie sagt $v/c = 0{,}933$ entlang der Achse gegen exakt $1{,}000$ diagonal — fast sieben Prozent Unterschied, und das Bild zeigt die Quittung: Aus dem Ring ist ein deutliches **Quadrat auf der Spitze** geworden. Vier Punkte pro Wellenlänge sind eben keine Auflösung, sondern eine Karikatur — die 10–20-Punkte-Faustregel aus Kapitel 8 gilt in 2D erst recht.

Ü 9.4 (Übertragen). Baue eine PEC-Wand quer durchs Gitter (eine Zeile, in der E_z nach jedem Schritt auf null gesetzt wird) und lasse darin einen **Spalt** von etwa einer halben Wellenlänge offen. Schicke die Ringwelle einer Sinusquelle von einer Seite darauf. Was erwartest du hinter der Wand — Schatten, Strahl oder etwas Drittes?

Musterlösung zu Ü 9.4

```
# von oben: felder(), schritt(), S_2D
nx = 401
m = nx // 2
n_lambda = 12
w_dt = 2 * np.pi * S_2D / n_lambda
anlauf = int(4 * 2 * np.pi / w_dt)
wand = 240                                  # Wandzeile (i-Index)
spalt = slice(m - 4, m + 4)                 # Öffnung: 8 Zellen ≈ 2/3 λ
e, ux, uy = felder(nx, nx)
for n in range(420):
    e, ux, uy = schritt(e, ux, uy, S_2D)
    e[140, m] += min(1.0, n / anlauf)**2 * np.sin(w_dt * n)
    e[wand, :spalt.start] = 0.0             # PEC-Wand ...
    e[wand, spalt.stop:] = 0.0              # ... mit offenem Spalt

fig, ax = plt.subplots(figsize=(4.6, 4.0))
w = 0.025                                   # harte Sättigung vor der Wand,
ax.imshow(e.T, origin="lower", cmap="RdBu_r",  # damit das leise Feld dahinter
          vmin=-w, vmax=w)                     # sichtbar wird
ax.axvline(wand, color="k", lw=1.0)
ax.set_xlabel("Zelle i"); ax.set_ylabel("Zelle j")
plt.tight_layout(); plt.show()
```



Etwas Drittes: Hinter dem Spalt entsteht **eine neue Ringwelle**, als säße im Spalt selbst eine Punktquelle — die Wellenfronten fächern in den gesamten Halbraum auf, auch in den „Schatten", den ein Lichtstrahl-Bild vorhersagen würde. Ein Spalt, der klein gegen die Wellenlänge ist, *ist* aus Sicht der Welle eine Punktquelle. Genau diese Idee — jede Stelle einer Wellenfront als Quelle einer neuen Elementarwelle — heißt **Huygens-Prinzip**, und sie wird in Kapitel 14 zum Doppelspalt und zur Interferenz ausgebaut: Dort lassen wir zwei solche Spalte gegeneinander antreten.

10 Spektren, Konvergenz, Vertrauen

Schlag eine Glocke mit einem Hammer an. Der Schlag selbst ist stumpf — ein Puls, ein „Wumms” ohne erkennbare Tonhöhe. Und doch antwortet die Glocke mit einem klaren Klang: Sie hat sich aus dem Wumms **ihre eigenen Frequenzen herausgesucht** und alles andere verschluckt. Der stumpfe Schlag muss diese Töne also irgendwie enthalten haben — alle auf einmal.

Genau dieses Prinzip macht aus unseren Simulationen Messgeräte: Ein einziger Puls in einen Resonator, und *eine* Rechnung liefert *alle* seine Resonanzfrequenzen gleichzeitig. Dafür fehlt uns noch das Ohr — ein Werkzeug, das aus einem Zeitsignal die enthaltenen Frequenzen liest. Wir bauen es in diesem Kapitel von Hand (es ist verblüffend wenig dahinter: multiplizieren und mitteln) und lassen es dann von `np.fft` schnell rechnen.

Und dann ist da noch eine Schuld, die dieses Buch seit Kapitel 5 vor sich herschiebt. Wir haben gemessen, geprüft und verglichen — aber die Grundsatzfrage nie systematisch gestellt: **Warum sollte man den Zahlen einer Simulation eigentlich glauben?** Mit dem Spektrum-Werkzeug in der Hand können wir sie beantworten — mit Konvergenztests, dem Vergleich gegen exakte Mathematik und einer Checkliste, die ab Teil III vor jedem Werkzeug-Einsatz steht.

Lernziele

Nach diesem Kapitel kannst du …

1. … einen Frequenz-Detektor selbst bauen (Signal mal Sinus, mitteln) und erklären, warum fremde Frequenzen dabei herausfallen,
2. … Spektren mit `np.fft.rfft` berechnen und lesen — und weißt, was Aufnahmedauer und Abtastschritt über Auflösung und Obergrenze entscheiden,
3. … erklären, warum ein kurzer Puls ein breites Spektrum hat — der „Chor aus Sinuswellen” aus Kapitel 8, jetzt messbar,
4. … einen Resonator mit einem einzigen Pulslauf komplett vermessen und das Ergebnis gegen die Analytik halten,
5. … einen Konvergenztest durchführen und mit der Validierungs-Checkliste begründen, *wann* eine Simulation Vertrauen verdient.

10.1 Ein Frequenz-Detektor von Hand

Gegeben ist eine Messreihe — sagen wir eine Funkantenne, deren Spannung wir 1 µs lang alle 10 ns abgetastet haben, 100 Werte. Wir bauen sie uns selbst, damit wir die Wahrheit kennen: zwei Sinustöne, 3 MHz mit Amplitude 1 und 5 MHz mit Amplitude 0,5; der 3-MHz-Ton bekommt zusätzlich eine Phasenverschiebung, denn eine echte Messung beginnt nie höflich beim Nulldurchgang. Die Frage: Wie findet man die beiden Töne wieder, wenn man die Wahrheit *nicht* kennt?

Die Idee ist ein **Detektor zum Hineinhalten**: Um zu prüfen, ob eine Testfrequenz f im Signal steckt, multipliziere das Signal Punkt für Punkt mit $\sin(2\pi f t)$ — und **mittele** das Produkt über die ganze Aufnahme. Steckt ein Ton der Frequenz f im Signal, dann ist das Produkt im Gleichtakt: positiv mal positiv, negativ mal negativ — der Mittelwert bleibt stehen. Jeder *fremde* Ton gerät gegen den Test-Sinus mal in den Gleichtakt, mal in den Gegentakt, und mittelt sich zu null weg. Das ist keine Hoffnung, sondern Schulmathematik — das Additionstheorem verwandelt jedes Produkt zweier Sinusse in eine Differenz zweier Kosinusse:

$$\sin a \cdot \sin b = \tfrac{1}{2}\big[\cos(a-b) - \cos(a+b)\big],$$

und ein Kosinus, der über volle Perioden gemittelt wird, ergibt null. Nur wenn a und b zur **selben** Frequenz gehören, wird $\cos(a-b)$ zu $\cos(0) = 1$ — und übrig bleibt genau die halbe Amplitude des getroffenen Tons. Die Handrechnung am konkreten Zahlenpaar, jeder Schritt einzeln:

```
import numpy as np
import matplotlib.pyplot as plt

DT_AB = 10e-9                       # Abtastschritt: alle 10 ns ein Messwert
N_AB = 100                          # 100 Werte → Aufnahmedauer 1 µs
t = np.arange(N_AB) * DT_AB

signal = (1.0 * np.sin(2*np.pi * 3e6 * t + 0.7)    # 3 MHz, phasenverschoben
          + 0.5 * np.sin(2*np.pi * 5e6 * t))       # 5 MHz

probe_3 = np.sin(2*np.pi * 3e6 * t)                # Test-Sinus 3 MHz
probe_4 = np.sin(2*np.pi * 4e6 * t)                # Test-Sinus 4 MHz
print(f"Mittel über sin·sin, gleiche Frequenz : "
      f"{np.mean(probe_3 * probe_3):+.4f}   (= 1/2)")
print(f"Mittel über Signal · 3-MHz-Probe      : "
      f"{np.mean(signal * probe_3):+.4f}")
```

```
print(f"Mittel über Signal · 4-MHz-Probe      : "
      f"{np.mean(signal * probe_4):+.4f}   (fremde Frequenz)")
```

```
Mittel über sin·sin, gleiche Frequenz : +0.5000   (= 1/2)
Mittel über Signal · 3-MHz-Probe      : +0.3824
Mittel über Signal · 4-MHz-Probe      : +0.0000   (fremde Frequenz)
```

Die erste Zeile zeigt die Eichung des Detektors: Selbst der Volltreffer liefert nur $\frac{1}{2}$ (das ist das gemittelte $\sin^2$ — die Hälfte der Zeit ist der Sinus klein). Ein Treffer muss also **mal 2** genommen werden. Die zweite Zeile findet vom 3-MHz-Ton nur $0{,}38$ statt der erwarteten $0{,}5$ — die Phasenverschiebung! Der Ton steckt zum Teil „im Kosinus", und ein Sinus-Detektor allein übersieht diesen Anteil. Die Lösung: **zweimal detektieren**, einmal mit Sinus, einmal mit Kosinus, und die beiden Anteile wie die Katheten eines rechtwinkligen Dreiecks zur Gesamtamplitude zusammensetzen:

```
# von oben: signal, t
def detektor(sig, f, t):
    """Amplitude der Frequenz f im Signal: 2·Mittel gegen sin und cos."""
    a = 2 * np.mean(sig * np.sin(2*np.pi * f * t))
    b = 2 * np.mean(sig * np.cos(2*np.pi * f * t))
    return np.hypot(a, b)                # √(a² + b²), wie in Kapitel 1

for f in (2e6, 3e6, 4e6, 5e6, 6e6):
    print(f"Detektor bei {f/1e6:.0f} MHz: {detektor(signal, f, t):.4f}")
```

```
Detektor bei 2 MHz: 0.0000
Detektor bei 3 MHz: 1.0000
Detektor bei 4 MHz: 0.0000
Detektor bei 5 MHz: 0.5000
Detektor bei 6 MHz: 0.0000
```

Da sind sie: $1{,}0$ bei 3 MHz, $0{,}5$ bei 5 MHz, praktisch null überall sonst — Amplituden **und** Verstecke gefunden, ohne die Wahrheit zu kennen. Lässt man den Detektor über ein ganzes Raster von Testfrequenzen laufen, entsteht das **Spektrum** des Signals: eine Funktion, die jeder Frequenz die in ihr steckende Amplitude zuordnet (der Ergebnistyp: ein Array über dem Frequenzraster, eine Zahl pro Testfrequenz).

10.2 np.fft: derselbe Detektor, nur schnell

Unser Detektor rechnet pro Testfrequenz das ganze Signal zweimal durch — für lange Messreihen und viele Frequenzen wird das träge. Die **schnelle Fourier-Transformation** (FFT, nach Jean Baptiste Fourier; der Algorithmus stammt von Cooley und Tukey, 1965) berechnet exakt dieselben Sinus- und Kosinus-Mittel für *alle* Frequenzen des Rasters auf einmal, durch geschicktes Wiederverwenden von Zwischenergebnissen. In NumPy heißt sie für reelle Messreihen `np.fft.rfft`, und es lohnt sich, die Übereinstimmung mit dem Hand-Detektor einmal nachzuprüfen:

```
# von oben: signal, t, detektor(), N_AB, DT_AB
# np.fft.rfft: FFT für reelle Signale – liefert für jede Rasterfrequenz
# das (Kosinus, Sinus)-Anteilspaar, verpackt als komplexe Zahl;
# np.abs holt daraus die Gesamtamplitude (unser np.hypot(a, b))
spektrum_fft = np.abs(np.fft.rfft(signal)) * 2 / N_AB
# np.fft.rfftfreq: das zugehörige Frequenzraster (aus N und Δt)
frequenzen = np.fft.rfftfreq(N_AB, DT_AB)

raster = np.arange(0, 13e6, 1e6)
spektrum_hand = np.array([detektor(signal, f, t) for f in raster])

fig, ax = plt.subplots(figsize=(6.6, 3.2))
ax.plot(frequenzen / 1e6, spektrum_fft, "o", ms=5, label="np.fft.rfft")
ax.plot(raster / 1e6, spektrum_hand, "+", ms=11, mew=1.6,
        label="Detektor von Hand")
ax.set_xlim(-0.3, 12.3)
ax.set_xlabel("Frequenz (MHz)"); ax.set_ylabel("Amplitude (V/m)")
ax.legend()
plt.tight_layout(); plt.show()

vergleich = np.max(np.abs(spektrum_hand
                          - spektrum_fft[:len(raster)]))
print(f"größte Abweichung Hand-Detektor vs. rfft: {vergleich:.1e}")
```

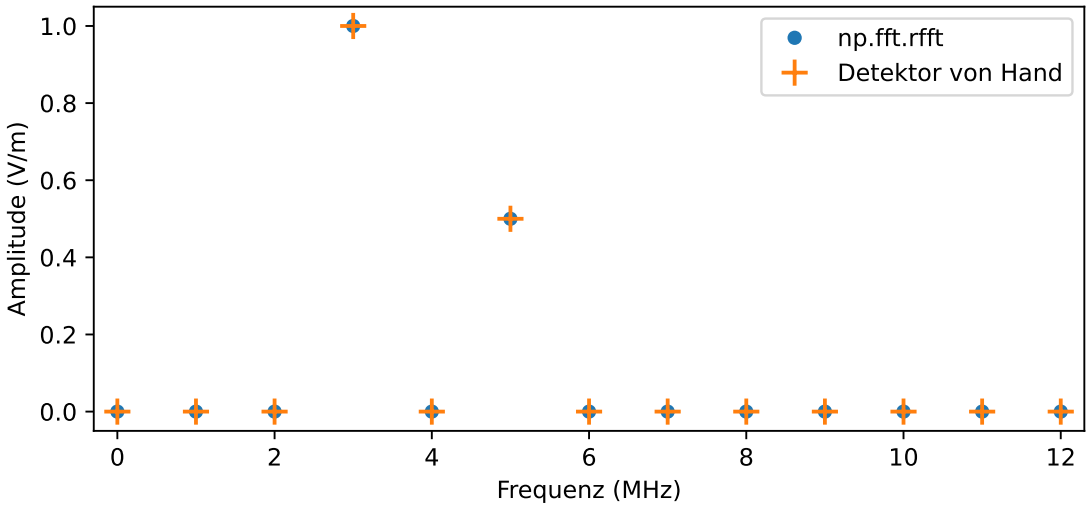


Abbildung 10.1: Das Spektrum des Testsignals, zweimal berechnet: np.fft.rfft (Punkte) und der Hand-Detektor (Kreuze) liegen exakt übereinander — bis auf Rundungsrauschen unterhalb von 10^{-15}. Die beiden versteckten Töne (3 MHz mit Amplitude 1, 5 MHz mit Amplitude 0,5) stehen sauber im sonst leeren Spektrum.

```
größte Abweichung Hand-Detektor vs. rfft: 4.7e-16
```

Der Faktor `2 / N_AB` übersetzt die rfft-Ausgabe in unsere Detektor-Eichung (das Mitteln steckt bei NumPy nicht drin, das Mal-zwei auch nicht). Zwei Eigenschaften des Frequenzrasters solltest du auswendig kennen, denn sie entscheiden über jedes Messdesign:

- **Die Auflösung ist 1/Aufnahmedauer.** Unsere 1 µs lange Messung hat ein Raster im Abstand $\Delta f = 1/(1\,\mu\text{s}) = 1$ MHz — feiner kann diese Aufnahme Frequenzen nicht unterscheiden. Wer zwei nahe Töne trennen will, braucht **länger**, nicht öfter: Erst wenn die Aufnahme so lang ist, dass die beiden Töne darin spürbar verschieden oft schwingen, kann irgendein Verfahren sie auseinanderhalten (Ü 10.3 führt das vor — und räumt mit der verbreiteten Vermutung auf, schnelleres Abtasten würde helfen).
- **Die Obergrenze ist $1/(2\Delta t_{\mathbf{ab}})$.** Mit 10 ns Abtastschritt reicht das Raster bis 50 MHz. Der Grund ist ein alter Bekannter: Die schnellste Schwingung, die eine Abtastreihe darstellen kann, ist die mit **zwei Messwerten pro Periode** — plus, minus, plus, minus. Das ist der Zickzack aus Kapitel 8, diesmal in der Zeit statt im Raum (sein offizieller Name ist **Nyquist-Grenze**; Ü 10.2 zieht die Parallele sauber durch).

10.3 Der Chor, sichtbar gemacht

Kapitel 8 hat behauptet, ein schmaler Puls sei „ein Chor aus vielen Sinuswellen, die exakt im Takt starten" — und die Präzisierung auf dieses Kapitel vertagt. Jetzt ist das eine Messung von drei Zeilen: Wir nehmen zwei Gauß-Pulse verschiedener Breite als Zeitsignale und halten sie in den Detektor.

> **Vorhersage-Punkt**
>
> Zwei Gauß-Pulse, einer 50 ns breit, einer 200 ns: Welcher von beiden braucht den **größeren** Chor — das breitere Spektrum?

Und was wäre demnach das Extrem — welches Zeitsignal bräuchte *alle* Frequenzen gleich stark? Die Messung:

```
# von oben: nichts (eigenständige Messreihe)
t_puls = np.arange(4096) * 1e-9              # 4 µs, abgetastet alle 1 ns
fr = np.fft.rfftfreq(len(t_puls), 1e-9)

fig, ax = plt.subplots(figsize=(6.6, 3.0))
for tau, farbe in ((50e-9, "tab:blue"), (200e-9, "tab:orange")):
    puls = np.exp(-((t_puls - 1e-6) / tau)**2)
    spektrum = np.abs(np.fft.rfft(puls))
    ax.plot(fr / 1e6, spektrum / spektrum[0], color=farbe,
            label=f"Gauß-Puls, τ = {tau*1e9:.0f} ns")
ax.set_xlim(0, 15)
ax.set_xlabel("Frequenz (MHz)"); ax.set_ylabel("Amplitude (normiert)")
ax.legend()
plt.tight_layout(); plt.show()
```

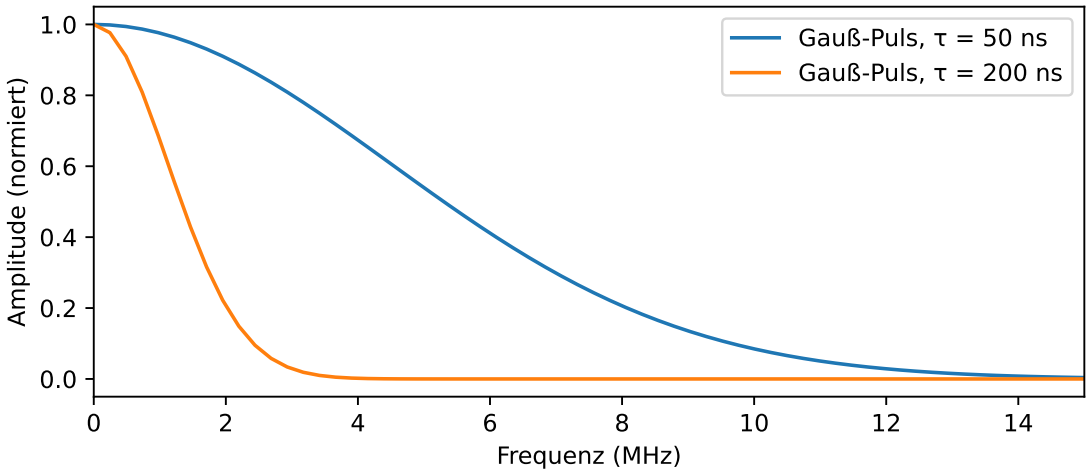


Abbildung 10.2: Die Spektren zweier Gauß-Pulse (auf gleiche Höhe normiert). Der viermal schmalere Puls (blau, 50 ns) braucht den viermal breiteren Chor: Sein Spektrum reicht bis über 10 MHz, das des gemütlichen 200-ns-Pulses (orange) ist schon bei 3 MHz praktisch leer. Je schärfer ein Signal in der Zeit, desto mehr Frequenzen stecken darin.

Der schmale Puls hat das breite Spektrum — Zeitschärfe wird mit Frequenzbreite bezahlt, und im Grenzfall braucht ein unendlich scharfer Knall *alle* Frequenzen. (Dieses Tauschgeschäft ist übrigens dasselbe, das in der Quantenmechanik als Unschärferelation Berühmtheit erlangt hat — dort zwischen Ort und Impuls. Bei uns bleibt es Handwerkszeug.) Damit ist auch die Schleppe aus Kapitel 8 vollständig erklärt: Der schmale Puls *ist* ein breiter Chor, das grobe Gitter lässt dessen hohe Stimmen langsamer laufen, und sie tröpfeln hinterher.

Für die Simulationspraxis ist der Chor aber vor allem eines: ein **Werkzeug**. Wer einen Resonator, ein Filter oder eine Antenne bei tausend Frequenzen kennenlernen will, muss nicht tausendmal simulieren — ein Puls fragt alle tausend gleichzeitig.

10.4 Ein Lauf, alle Resonanzen

Das Glocken-Experiment, jetzt elektromagnetisch. Der Versuchsaufbau: **Die Bühne** ist ein Resonator — die 1D-Strecke aus den Kapiteln 5–8, diesmal $L = 1$ m lang und an **beiden** Enden mit PEC-Spiegeln verschlossen (die stehende Welle aus Kapitel 6 hat es vorgemacht: Zwischen zwei Spiegeln

passen nur Schwingungsmuster, deren Knoten auf die Wände fallen). Die Analytik dazu ist Schulrechnung: In die Länge L passen m Halbwellen, $\lambda_m = 2L/m$, also liegen die Resonanzen am **Kamm**

$$f_m = m \cdot \frac{c}{2L} \approx m \cdot 150 \text{ MHz}, \qquad m = 1, 2, 3, \ldots$$

Das Gitter hat $\Delta x = 2$ cm (51 Punkte) und läuft bewusst mit $S = 0{,}5$ — nicht mit dem magischen Zeitschritt: Wir *wollen* sehen, was die Gitterdispersion aus Kapitel 8 mit den Resonanzen anstellt. **Die Anregung:** der Glockenschlag — eine weiche Quelle bei $x = 0{,}32$ m gibt einen sehr kurzen Gauß-Puls ab ($\tau = 0{,}1$ ns; nach dem Chor-Befund deckt der alle Frequenzen bis über 2,5 GHz ab). Quelle wie **Sonde** (bei $x = 0{,}74$ m) sitzen absichtlich an krummen Orten: Wer auf einem Knoten einer Mode sitzt, kann sie weder anregen noch hören (Ü 10.1 macht daraus ein Experiment). **Gemessen wird** das E-Feld an der Sonde, jeden Zeitschritt, über $2^{17} \approx 131\,000$ Schritte (4,4 µs — lange Aufnahme, feines Raster: $\Delta f = 229$ kHz). Danach eine einzige FFT. **Erwartung:** Peaks auf dem Kamm $m \cdot 150$ MHz — und wer Kapitel 8 ernst nimmt, erwartet die hohen Moden *nicht* ganz dort.

> **Vorhersage-Punkt**
>
> Mode 15 hat $\lambda = 13{,}3$ cm — auf dem 2-cm-Gitter sind das nur 6,7 Punkte pro Wellenlänge. Wird ihr Peak rechts oder links der Kammlinie $15 \cdot f_1$ liegen? Und um grob wie viel Prozent? (Kapitel 8, Faustregel-Tabelle, hilft.)

```
# von oben: nichts Neues nötig — der Resonator wird hier gebaut
from scipy.constants import c

L = 1.0                                     # Resonatorlänge (m)
N_RES = 51                                  # Δx = 2 cm
S = 0.5                                     # bewusst NICHT magisch (Kap. 8)
DT = S * (L / (N_RES - 1)) / c

def resonator_aufnahme(schritte, quellidx=16, sondenidx=37):
    """Glockenschlag in den PEC-Resonator; Sonde zeichnet jeden Schritt auf."""
    e, u = np.zeros(N_RES), np.zeros(N_RES - 1)    # u = c·B (Kap. 7-9)
    tau, t0 = 0.1e-9, 0.6e-9
    # np.empty: Array OHNE Anfangswerte (schneller als zeros) — nur
    # benutzen, wenn ohnehin jeder Eintrag gleich beschrieben wird
    aufnahme = np.empty(schritte)
    for n in range(schritte):
```

```
        u -= S * np.diff(e)
        e[1:-1] -= S * np.diff(u)
        e[quellidx] += np.exp(-((n * DT - t0) / tau)**2)
        aufnahme[n] = e[sondenidx]
    return aufnahme

SCHRITTE = 2**17
aufnahme = resonator_aufnahme(SCHRITTE)
spektrum = np.abs(np.fft.rfft(aufnahme))
fr = np.fft.rfftfreq(SCHRITTE, DT)
f1 = c / (2 * L)

def f_dispersion(m):
    """Kap.-8-Formel: die Gitterfrequenz der Mode m (kΔx = mπ/(N−1))."""
    k_dx = m * np.pi / (N_RES - 1)
    return 2 * np.arcsin(S * np.sin(k_dx / 2)) / (2 * np.pi * DT)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(7.0, 5.2))
ax1.plot(fr / 1e9, spektrum, lw=0.8)
for m in range(1, 17):
    ax1.axvline(m * f1 / 1e9, color="gray", ls=":", lw=0.7)
ax1.set_xlim(0, 2.5)
ax1.set_xlabel("Frequenz (GHz)"); ax1.set_ylabel("|Spektrum|")

ax2.plot(fr / 1e9, spektrum, lw=1.0)
ax2.axvline(15 * f1 / 1e9, color="gray", ls=":", lw=1.2,
            label="Kamm: 15·c/2L")
ax2.axvline(f_dispersion(15) / 1e9, color="tab:red", ls="--", lw=1.2,
            label="Dispersionsformel (Kap. 8)")
ax2.set_xlim(2.10, 2.32); ax2.set_ylim(0, 1.3 * spektrum[
    (fr > 2.1e9) & (fr < 2.32e9)].max())
ax2.set_xlabel("Frequenz (GHz)"); ax2.set_ylabel("|Spektrum|")
ax2.legend(fontsize=8)
plt.tight_layout(); plt.show()

print("Mode   Peak (GHz)   Kamm m·f1   Kap.-8-Formel   Drift")
for m in (1, 5, 10, 15):
    f_theo = f_dispersion(m)
    maske = (fr > f_theo - 0.4 * f1) & (fr < f_theo + 0.4 * f1)
    f_peak = fr[maske][np.argmax(spektrum[maske])]
    print(f" {m:2}    {f_peak/1e9:.5f}      {m*f1/1e9:.5f}      "
          f"{f_theo/1e9:.5f}        {100*(f_peak/(m*f1)-1):+.2f} %")
```

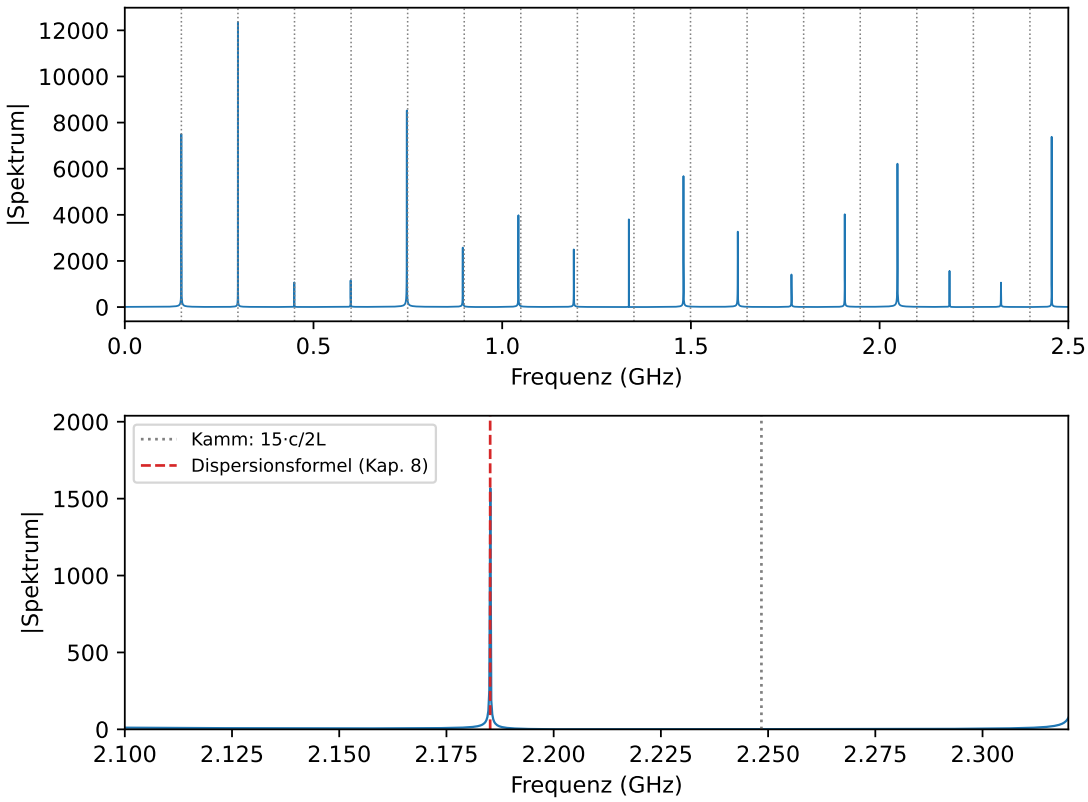


Abbildung 10.3: Oben: das Spektrum der Sonde nach einem einzigen Pulslauf — jeder Peak eine Resonanz; die gepunkteten Linien sind der analytische Kamm m·c/2L. Tiefe Moden sitzen exakt auf ihren Linien, hohe driften zunehmend nach links. Unten der Zoom auf Mode 15: Der Peak liegt 2,8 % unter der Kammlinie (gepunktet) — aber haargenau auf der Vorhersage der Dispersionsformel aus Kapitel 8 (gestrichelt): Diese Mode hat nur 6,7 Gitterpunkte pro Wellenlänge und schwingt deshalb zu langsam.

Mode	Peak (GHz)	Kamm m·f1	Kap.-8-Formel	Drift
1	0.14981	0.14990	0.14988	-0.05 %
5	0.74724	0.74948	0.74717	-0.30 %
10	1.48030	1.49896	1.48036	-1.25 %
15	2.18522	2.24844	2.18519	-2.81 %

Ein einziger Lauf, und der Resonator hat sein komplettes Zeugnis abgelegt: sechzehn Resonanzen in einem Bild. Die tiefen Moden treffen den analytischen Kamm aufs Raster genau — und die hohen erzählen die Geschichte

aus Kapitel 8 weiter: Mode 10 schwingt $1{,}25\,\%$ zu tief, Mode 15 schon $2{,}8\,\%$, denn ihre kurzen Wellen laufen auf dem groben Gitter zu langsam, und wer zu langsam läuft, schwingt zu selten. Das Bemerkenswerte steht in der Tabellenspalte daneben: Die Dispersionsformel aus Kapitel 8 sagt jeden dieser „Fehler" auf weniger als ein Hundertstel Prozent voraus. **Der Fehler ist kein Rauschen — er ist berechenbar.** Genau darauf baut der Rest des Kapitels.

10.5 Vertrauen ist messbar

⚠ Naheliegende Vermutung

„Die Simulation ist stabil durchgelaufen, die Bilder sehen physikalisch vernünftig aus — also kann ich den Zahlen glauben."
Warum sie naheliegt: Instabilität macht sich unübersehbar bemerkbar (Kapitel 7–9: Explosionen um dreißig Größenordnungen), und unsere Bilder *sind* ja oft korrekt — Ringwellen laufen mit c, Knoten sitzen im $\lambda/2$-Raster. Wenn das Grobe stimmt, warum nicht auch das Feine?
Was stattdessen stimmt: Stabil heißt nur „explodiert nicht" (Kapitel 7: stabil ≠ genau), und plausibel heißt nur „qualitativ richtig". Eine Simulation kann beides sein und trotzdem quantitativ wertlos — der Resonator eben war stabil und plausibel und lag bei Mode 15 um 2,8 % daneben; wer danach ein Filter gebaut hätte, hätte Schrott gebaut. Dieses Buch verdankt einer solchen Erfahrung eine seiner Lektionen: Ein zu grob aufgelöstes 2D-Modell einer Yagi-Antenne lieferte wunderschöne, plausible Richtdiagramme — und Elementabstände, die um Dutzende Prozent unbaubar waren. Vertrauen entsteht nicht durch Anschauen, sondern durch **Messen des eigenen Fehlers**. Wie, zeigen die nächsten zwei Experimente.

Werkzeug 1: der Vergleich mit der Analytik. Den haben wir eben schon benutzt — der Modenkamm $f_m = m\,c/2L$ ist exakte Mathematik, und die Abweichung der Simulation davon ist ihr gemessener Fehler. Das Schema ist allgemein: Man sucht sich einen Spezialfall des Problems, für den die exakte Antwort bekannt ist (ein Resonator, eine Kugel, ein unendlicher Draht …), und lässt das Programm gegen die Wahrheit antreten. Besteht es dort, darf man ihm auch dazwischen etwas glauben.

Werkzeug 2: der Konvergenztest. Oft gibt es keine Analytik — genau dafür simuliert man ja. Dann bleibt die Frage: Ändert sich das Ergebnis noch, wenn ich feiner rechne? Der Versuchsaufbau: **Bühne** ist derselbe Resonator, aber als Anfangsbedingung legen wir direkt die **Mode 3** hinein

($E = \sin(3\pi x/L)$, $u = 0$) — sie schwingt dann allein, ohne Chor, und die Sonde an ihrem Bauch bei $L/2$ liefert einen reinen Sinuston. **Gemessen wird** dessen Frequenz, und zwar nicht per FFT (deren Raster wäre hier zu grob), sondern mit der Nulldurchgangs-Methode aus Kapitel 8 — diesmal entlang der Zeitachse: Abstand der Nulldurchgänge, gemittelt über Hunderte Perioden, ergibt die Periodendauer auf sechs Stellen. **Die Leiter:** Wir messen dreimal, mit $\Delta x = 5$ cm, 2,5 cm und 1,25 cm (bei festem $S = 0{,}5$), und vergleichen jeweils mit dem analytischen $f_3 = 3\,c/2L$. **Erfolgskriterium** (Kapitel 2 und Ü 8.3 lassen grüßen): Halbiert sich Δx, muss der Fehler auf ein **Viertel** fallen — das ist die Handschrift der zweiten Ordnung, und nur ein Verfahren, das ihr folgt, ist im sauberen Konvergenz-Regime.

```
# von oben: L, S, f_dispersion()-Idee; c (Import)
def mode3_frequenz(n_punkte, schritte=40000):
    """Mode 3 schwingen lassen; Frequenz über Nulldurchgänge der Sonde."""
    x = np.linspace(0, L, n_punkte)
    e = np.sin(3 * np.pi * x / L)              # die Mode als Anfangsbild
    u = np.zeros(n_punkte - 1)
    dt = S * (L / (n_punkte - 1)) / c
    sonde = np.empty(schritte)
    # np.argmin: der INDEX des kleinsten Werts (Gegenstück zu np.argmax,
    # Kap. 5) — hier: welcher Gitterpunkt liegt am nächsten an L/2?
    bauch = np.argmin(np.abs(x - 0.5))         # Bauch der Mode 3

    for n in range(schritte):
        u -= S * np.diff(e)
        e[1:-1] -= S * np.diff(u)
        sonde[n] = e[bauch]
    i = np.flatnonzero(np.sign(sonde[:-1]) * np.sign(sonde[1:]) < 0)
    null = i - sonde[i] / (sonde[i + 1] - sonde[i])   # wie in Kap. 8
    return 1 / (2 * np.mean(np.diff(null)) * dt)

f3 = 3 * c / (2 * L)
print(f"Mode 3 analytisch: {f3/1e6:.3f} MHz\n")
fehler, gitter = [], []
for n_punkte in (21, 41, 81):
    f_mess = mode3_frequenz(n_punkte)
    dx_cm = 100 * L / (n_punkte - 1)
    fehler.append(1 - f_mess / f3)
    gitter.append(dx_cm)
    print(f"Δx = {dx_cm:5.2f} cm:  gemessen {f_mess/1e6:.4f} MHz,"
          f"  Fehler {100*fehler[-1]:.4f} %")
print(f"\nFehler-Quotienten je Halbierung: "
      f"{fehler[0]/fehler[1]:.2f} und {fehler[1]/fehler[2]:.2f}  (Soll: 4)")
```

```
fig, ax = plt.subplots(figsize=(5.6, 3.2))
ax.loglog(gitter, [100 * f for f in fehler], "o-", ms=7,
          label="gemessener Fehler")
ax.loglog(gitter, [100 * fehler[0] * (g / gitter[0])**2 for g in gitter],
          "k:", label="2. Ordnung (∝ Δx²)")
ax.set_xlabel("Δx (cm)"); ax.set_ylabel("Frequenzfehler (%)")
ax.legend()
plt.tight_layout(); plt.show()
```

```
Mode 3 analytisch: 449.689 MHz

Δx =  5.00 cm:  gemessen 446.5573 MHz,  Fehler 0.6963 %
Δx =  2.50 cm:  gemessen 448.9078 MHz,  Fehler 0.1736 %
Δx =  1.25 cm:  gemessen 449.4936 MHz,  Fehler 0.0434 %

Fehler-Quotienten je Halbierung: 4.01 und 4.00  (Soll: 4)
```

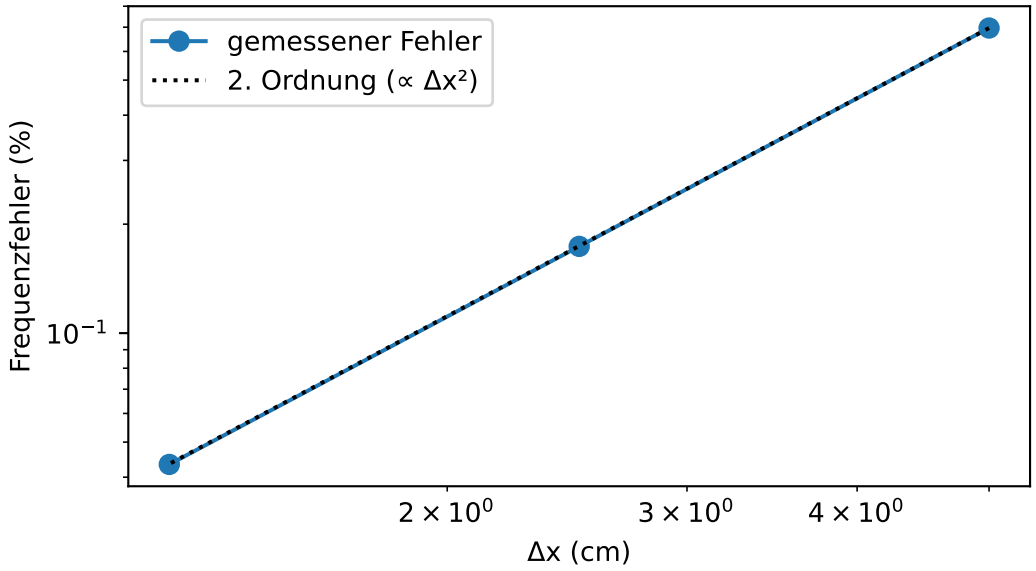


Abbildung 10.4: Die Konvergenzleiter: der Frequenzfehler der Mode 3 gegen die Gitterweite, doppelt-logarithmisch. Die Messpunkte fallen exakt auf die Gerade „Fehler ∝ Δx^2" (gepunktet als eingezeichnete Erwartung): halbe Zelle, ein Viertel Fehler. Wer einen vierten Punkt will, weiß jetzt, wo er liegen wird — das ist der Unterschied zwischen Hoffen und Wissen.

Quotienten von $4{,}01$ und $4{,}00$ — das Verfahren konvergiert lehrbuchhaft mit zweiter Ordnung gegen die Analytik. Und es kommt noch besser: Die gemessenen Fehler ($0{,}696$, $0{,}174$, $0{,}043\,\%$) sind exakt die Werte, die die Dispersionsformel aus Kapitel 8 für diese drei Gitter vorhersagt — auf vier Stellen. Der Fehler dieser Simulation ist damit kein dunkler Rest, sondern eine **verstandene, vorhersagbare, kontrollierbare Größe**. Mehr kann man von einer numerischen Methode nicht verlangen.

Für die Praxis genügt die abgespeckte Form des Tests, und sie ist jede Rechenzeit wert: **Jede wichtige Zahl zweimal rechnen** — einmal normal, einmal mit halbem Δx. Fällt der Unterschied auf etwa ein Viertel des Unterschieds zur nächstgröberen Stufe, ist man im Konvergenz-Regime und kennt obendrein die Größenordnung des Restfehlers. Ändert sich die Zahl dagegen munter weiter, ist das Ergebnis noch keine Physik, sondern Gitterkunst.

10.6 Die Checkliste

Die Vertrauensfrage, als Arbeitsprogramm vor jedem Ernstfall — jede Zeile haben wir inzwischen selbst durchexerziert:

Prüfung	Werkzeug	eingeführt
Bleibt erhalten, was erhalten bleiben muss?	Energie-Schiedsrichter	Kap. 5
Stimmt ein analytisch lösbarer Spezialfall?	z. B. Modenkamm $m\,c/2L$	Kap. 10
Konvergiert das Ergebnis mit 2. Ordnung?	Δx halbieren, Fehler vierteln	Kap. 10
Ist die kürzeste Wellenlänge aufgelöst?	10–20 Punkte pro λ, Laufstrecke zählt	Kap. 8
Stören die Ränder?	Echo messen (Rest nach Durchgang)	Kap. 6/9
Sagt ein unabhängiges Werkzeug dasselbe?	Querrechnen (MoM vs. FDTD …)	Teil III

Die letzte Zeile ist die Königsdisziplin und der eigentliche Grund für Teil III: Wer dasselbe Problem mit zwei grundverschiedenen Methoden rechnet und zweimal dieselbe Antwort bekommt, hat mehr bewiesen als jede Einzelrechnung. Die Yagi-Antenne aus der Refutation oben wurde am Ende genau so

gerettet — entworfen mit der Momentenmethode, gegengerechnet mit einem zweiten Feldlöser, und erst, als beide übereinstimmten, gebaut. Diese Geschichte erzählt Kapitel 11 zu Ende.

10.7 Das Kapitel-Programm

`programme/kap10/kap10_spektren_vertrauen.py` bündelt alle vier Befunde eigenständig und prüft sie mit `assert`-Schranken: Detektor findet $1{,}0/0{,}5$ und schweigt bei 4 MHz ($< 10^{-12}$); Hand-Detektor = rfft ($< 10^{-12}$); Halbwertsbreiten-Verhältnis der Pulsspektren ≈ 4; jeder Resonanz-Peak auf der Dispersionsformel ($< 10^{-3}$ relativ); Konvergenzquotienten in $[3{,}6,\ 4{,}4]$ und Messung = Formel ($< 10^{-4}$).

Merkkasten

- Ein **Spektrum** ist Multiplizieren und Mitteln: $2\langle s{\cdot}\sin\rangle$ und $2\langle s{\cdot}\cos\rangle$ je Testfrequenz; fremde Frequenzen mitteln sich weg (Additionstheorem). `np.fft.rfft` rechnet exakt das — schnell.
- **Auflösung = 1/Aufnahmedauer; Obergrenze = 1/(2·Abtastschritt)** (Nyquist — der Zickzack in der Zeit). Länger messen trennt Frequenzen, schneller abtasten hebt nur die Decke.
- **Schmal in der Zeit = breit im Spektrum.** Darum vermisst *ein* Pulslauf alle Resonanzen zugleich — das Arbeitspferd jedes FDTD-Workflows.
- **Stabil ≠ plausibel ≠ richtig.** Vertrauen kommt aus Messung: Analytik-Vergleich, Konvergenztest (halbes Δx → ein Viertel Fehler), Erhaltungsgrößen, Rand-Echos, Querrechnen — die Checkliste.
- Der Fehler einer guten Simulation ist **berechenbar** statt geheimnisvoll: Unsere Resonanz-Drifts trafen die Kapitel-8-Formel auf vier Stellen.

Roter Faden

Dieses Kapitel hat vor allem Schulden beglichen: Der **Chor** aus Kapitel 8 ist jetzt eine Messung, seine **Dispersionsformel** wurde zum Frequenz-Orakel für ganze Resonatoren, die **Nulldurchgangs-Methode** wanderte vom Raum in die Zeit, der **Zickzack** kehrte als Nyquist-Grenze wieder, und die stehende Welle aus **Kapitel 6** wurde zum Messobjekt. Nach vorn ist der Weg jetzt frei für **Teil III**: Kapitel 11 stellt die professionellen Löser vor —

in deren Inneren du DFT-Monitore, PML-Ränder und Konvergenz-Knöpfe wiedererkennen wirst, denn du hast jedes dieser Teile selbst gebaut. **Kapitel 14** (Interferenz) und **Kapitel 15** (Resonanz und Güte) leben dann ganz von Spektren — dort bekommen die Peaks auch endlich eine Breite, und die erzählt von Verlusten.

Übungen

Ü 10.1 (Verstehen). Im Resonator-Experiment saßen Quelle und Sonde an „krummen" Orten. Sage ohne Rechnung voraus, was im Spektrum passiert, wenn die Quelle stattdessen **exakt in die Mitte** des Resonators gesetzt wird — welche Moden verschwinden, und warum? Prüfe dann nach.

ℹ Musterlösung zu Ü 10.1

Die Mitte ist für jede **gerade** Mode ein *Knoten*: Mode m hat die Form $\sin(m\pi x/L)$, und bei $x = L/2$ ist $\sin(m\pi/2) = 0$ für $m = 2, 4, 6, \ldots$ Eine weiche Quelle addiert ihr Signal an einem Ort — wo eine Mode einen Knoten hat, ist sie taub für jede Anregung (genau wie eine Gitarrensaite, die man am zwölften Bund anschlägt, keine Oktav-Obertöne bekommt). Die geraden Moden müssen also aus dem Spektrum verschwinden:

```
# von oben: resonator_aufnahme(), f_dispersion(), N_RES, DT, f1
aufnahme_mitte = resonator_aufnahme(2**15, quellidx=N_RES // 2)
spektrum_m = np.abs(np.fft.rfft(aufnahme_mitte))
fr_m = np.fft.rfftfreq(2**15, DT)
for m in (1, 2, 3, 4):
    f_theo = f_dispersion(m)
    maske = (fr_m > f_theo - 0.3 * f1) & (fr_m < f_theo + 0.3 * f1)
    print(f"Mode {m}: Peakhöhe {spektrum_m[maske].max():8.1f}")
```

```
Mode 1: Peakhöhe   2400.0
Mode 2: Peakhöhe      4.3
Mode 3: Peakhöhe   2174.6
Mode 4: Peakhöhe      6.3
```

Die Moden 2 und 4 sind um mehr als einen Faktor 100 eingebrochen — übrig bleibt nur, was Rundungsreste anregen. Dieselbe Logik gilt für die **Sonde**: Sie ist blind für Moden, auf deren Knoten sie sitzt. Quelle und Sonde krumm zu platzieren ist also keine Marotte, sondern Messtechnik.

Ü 10.2 (Verstehen). Begründe die Nyquist-Grenze $f_{\text{max}} = 1/(2\Delta t_{\text{ab}})$ mit dem Zickzack-Argument aus Kapitel 8 — und erkläre dann, warum eine Abtastreihe einen Ton *oberhalb* dieser Grenze nicht einfach übersieht, sondern als **falschen tieferen Ton** meldet. (Denkhilfe: Was zeigen die Messwerte, wenn zwischen zwei Abtastungen mehr als eine halbe Schwingung vergeht?)

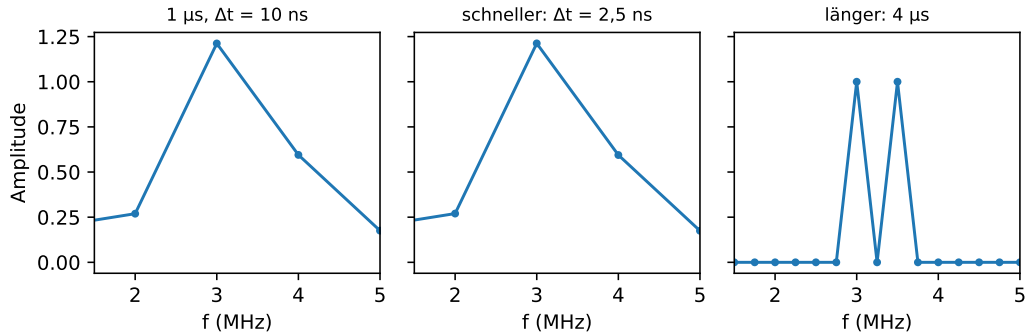
Musterlösung zu Ü 10.2

Das schnellste Muster, das eine Wertereihe im Abstand Δt_{ab} darstellen kann, ist der Wechsel plus–minus–plus–minus: **zwei Werte pro Periode**, also die Periode $2\Delta t_{\text{ab}}$ und damit die Frequenz $1/(2\Delta t_{\text{ab}})$ — wörtlich der Zickzack aus Kapitel 8, nur dass die „Gitterpunkte" jetzt Zeitpunkte sind. Ein schnellerer Ton verschwindet aber nicht: Die Abtastung erwischt von jeder Schwingung nur Momentaufnahmen, und wenn zwischen zwei Aufnahmen fast eine ganze Periode vergeht, sehen die Messwerte aus, als schwänge dort etwas ganz Langsames (derselbe Effekt lässt im Film Wagenräder rückwärts laufen, wenn die Bildrate nicht mitkommt). Der zu schnelle Ton wird also unter falschem Namen einsortiert — **Aliasing**. Darum steht vor jedem ernsthaften Abtasten ein Tiefpassfilter, das alles oberhalb der Nyquist-Grenze entfernt, bevor es lügen kann.

Ü 10.3 (Verändern). Zwei Töne, 3,0 und 3,5 MHz, sollen im Spektrum getrennt werden. Die 1-µs-Aufnahme von oben schafft das nicht ($\Delta f = 1$ MHz). Probiere beide Auswege aus: (a) viermal **schneller abtasten** (gleiche Dauer), (b) viermal **länger aufnehmen** (gleicher Abtastschritt). Welcher hilft?

Musterlösung zu Ü 10.3

```
# von oben: DT_AB, N_AB
fig, achsen = plt.subplots(1, 3, figsize=(7.4, 2.6), sharey=True)
faelle = (("1 µs, Δt = 10 ns", DT_AB, N_AB),
          ("schneller: Δt = 2,5 ns", DT_AB / 4, 4 * N_AB),
          ("länger: 4 µs", DT_AB, 4 * N_AB))
for ax, (titel, dt_ab, n_ab) in zip(achsen, faelle):
    tt = np.arange(n_ab) * dt_ab
    sig = np.sin(2*np.pi * 3.0e6 * tt) + np.sin(2*np.pi * 3.5e6 * tt)
    ax.plot(np.fft.rfftfreq(n_ab, dt_ab) / 1e6,
            np.abs(np.fft.rfft(sig)) * 2 / n_ab, "o-", ms=3)
    ax.set_xlim(1.5, 5); ax.set_title(titel, fontsize=9)
    ax.set_xlabel("f (MHz)")
achsen[0].set_ylabel("Amplitude")
plt.tight_layout(); plt.show()
```



Nur das **längere Aufnehmen** trennt die beiden Töne (rechtes Bild: zwei saubere Spitzen im Abstand 0,5 MHz, denn jetzt ist $\Delta f = 1/4\,\mu\text{s} = 0{,}25$ MHz). Das schnellere Abtasten (Mitte) liefert exakt denselben verschmierten Buckel wie das Original — es verlängert das Frequenzraster nach oben (bis 200 MHz), macht es aber kein bisschen feiner. Auflösung kauft man mit **Zeit**, nicht mit Abtastrate: In 1 µs schwingt der eine Ton 3-mal, der andere 3,5-mal — ein halber Schwingungsunterschied ist schlicht zu wenig Beweismaterial, egal wie dicht man hinschaut.

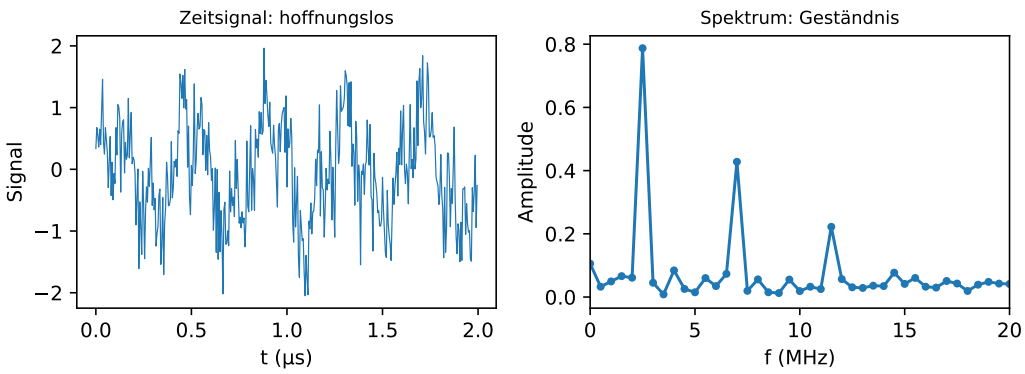
Ü 10.4 (Übertragen). Die Detektivaufgabe: Die Zelle unten baut eine „unbekannte" Messreihe — drei Sinustöne plus kräftiges Rauschen; im Zeitsignal ist mit bloßem Auge nichts zu erkennen. Bestimme Frequenzen und Amplituden der drei Töne.

Musterlösung zu Ü 10.4

```
# np.random.default_rng(7): Zufallsgenerator mit festem Startwert –
# „zufällig", aber bei jedem Lauf reproduzierbar gleich
rng = np.random.default_rng(7)
n_ab, dt_ab = 400, 5e-9                        # 2 µs, alle 5 ns
tt = np.arange(n_ab) * dt_ab
geheim = (0.8 * np.sin(2*np.pi * 2.5e6 * tt)
          + 0.4 * np.sin(2*np.pi * 7.0e6 * tt + 1.0)
          + 0.25 * np.sin(2*np.pi * 11.5e6 * tt)
          # rng.standard_normal(n): n normalverteilte Zufallswerte (σ = 1)
          + 0.5 * rng.standard_normal(n_ab))  # → Rauschen mit σ = 0,5

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(7.4, 2.8))
ax1.plot(tt * 1e6, geheim, lw=0.6)
ax1.set_xlabel("t (µs)"); ax1.set_ylabel("Signal")
ax1.set_title("Zeitsignal: hoffnungslos", fontsize=9)
spektrum_g = np.abs(np.fft.rfft(geheim)) * 2 / n_ab
fr_g = np.fft.rfftfreq(n_ab, dt_ab)
ax2.plot(fr_g / 1e6, spektrum_g, "o-", ms=3)
ax2.set_xlim(0, 20)
ax2.set_xlabel("f (MHz)"); ax2.set_ylabel("Amplitude")
ax2.set_title("Spektrum: Geständnis", fontsize=9)
plt.tight_layout(); plt.show()

for f_ton in (2.5e6, 7.0e6, 11.5e6):
    k = np.argmin(np.abs(fr_g - f_ton))
    print(f"Peak bei {fr_g[k]/1e6:4.1f} MHz: Amplitude {spektrum_g[k]:.3f}")
```




```
Peak bei  2.5 MHz: Amplitude 0.787
Peak bei  7.0 MHz: Amplitude 0.428
```

```
Peak bei 11.5 MHz: Amplitude 0.222
```

Im Zeitsignal: Rauschteppich. Im Spektrum: drei unübersehbare Spitzen bei 2,5, 7 und 11,5 MHz, deren Höhen die wahren Amplituden (0,8/0,4/0,25) bis auf wenige Hundertstel treffen — sogar der leiseste Ton, der im Zeitsignal komplett unter dem Rauschen liegt. Der Trick ist derselbe wie im ganzen Kapitel: Der Detektor *mittelt*, und Rauschen, das mit keiner Testfrequenz im Takt ist, mittelt sich weg, während ein Ton Periode für Periode aufsummiert. Genau deshalb funktioniert Funkempfang unterhalb des Rauschens, GPS zum Beispiel — und genau deshalb ist die FFT das vielleicht meistbenutzte Stück Numerik der Welt.

11 Die Löser-Landschaft

Zehn Kapitel lang war in diesem Buch jede Zeile Physik selbst geschrieben: das Gitter, der Leapfrog, die Quellen, die Ränder, der Frequenz-Detektor. In diesem Kapitel schreiben wir die Ringwelle aus Kapitel 9 noch einmal — diesmal in einer Handvoll Zeilen, denn den Löser stellt jetzt **Meep**, ein professionelles, frei verfügbares FDTD-Programm, das seit zwei Jahrzehnten in Forschung und Industrie im Einsatz ist.

Das ist kein Abschied vom Eigenbau, sondern seine Belohnung. Wer einmal selbst ein Yee-Gitter aufgesetzt hat, betritt Meep nicht als Blackbox, sondern als Werkstatt voller Bekannter: Die Auflösung zählt Punkte pro Wellenlänge (Kapitel 8), die Courant-Zahl steht voreingestellt auf $0{,}5$ (Kapitel 8 und 9), am Rand wartet eine PML (Kapitel 9), und eingebaute DFT-Monitore rechnen exakt unseren Detektor aus Kapitel 10. Jedes dieser Wiedersehen werden wir **nachmessen**, nicht nur behaupten — und dabei die Königsdisziplin der Validierungs-Checkliste einlösen: das **Querrechnen**, dasselbe Problem mit zwei unabhängigen Programmen. Am Ende des Kapitels bekommt außerdem eine Geschichte ihren Schluss, die in Kapitel 10 nur angedeutet wurde: die Yagi-Antenne, die fast falsch gebaut worden wäre.

Lernziele

Nach diesem Kapitel kannst du …

1. … eine Meep-Simulation aufsetzen, laufen lassen und auslesen — und weißt, wo darin Courant-Zahl, weiche Quelle, PML und DFT-Detektor stecken,
2. … Meeps Einheitensystem erklären und umrechnen — und begründen, warum *eine* Rechnung zugleich für Mikrowellen und für Licht gilt (Skaleninvarianz),
3. … zwei unabhängige Programme gegeneinander querrechnen und einer verbleibenden Differenz auf den Grund gehen, statt sie wegzuwinken,
4. … die großen Verfahrensfamilien (FDTD/FIT, Momentenmethode, Finite Elemente, Strahlverfahren) unterscheiden und für ein gegebenes Problem das passende Werkzeug nennen,

5. … an der Yagi-Fallgeschichte erklären, warum ein Werkzeugwechsel manchmal die einzige Rettung ist — und warum erst das Querrechnen einen Entwurf baubar macht.

11.1 Vom Eigenbau zur Werkstatt

Warum überhaupt fremde Software, wenn der eigene Löser läuft? Weil zwischen unserem Lehr-Code und dem Ernstfall ein weites Feld liegt: echte Materialien (Glas, Gold, dispersive Medien), drei Dimensionen, gekrümmte Geometrien, die ein Treppenstufen-Gitter nur grob trifft (Meep glättet sie mit *Subpixel-Mittelung*), Parallelisierung auf viele Prozessorkerne, und — nicht zu unterschätzen — zwanzig Jahre Fehlersuche durch tausende Nutzer. All das kauft man mit einem `import` ein. Der Preis: Man muss dem Programm vertrauen. Genau dafür haben wir in Kapitel 10 die Checkliste gebaut, und genau darum wird dieses Kapitel Meep nicht vorstellen, sondern **vermessen**.

Meep wird am bequemsten über conda installiert (die Python-Distribution Miniforge genügt); ein einziger Befehl legt eine eigene Umgebung samt NumPy und Matplotlib an:

```
conda create -n meep -c conda-forge pymeep
conda activate meep
pip install necpp        # die NEC-Antennen-Engine (für den Yagi-Abschnitt)
```

Alle Code-Zellen dieses Kapitels (und der folgenden) laufen in dieser Umgebung. Die erste Begegnung:

```
import os
# Umgebungsvariable setzen, BEVOR Meep lädt: unterdrückt harmlose
# Hardware-Warnungen mancher MPI-Installationen
os.environ["HWLOC_HIDE_ERRORS"] = "2"

import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)   # Meeps Fortschrittsmeldungen aus – wir messen selbst
```

```
Using MPI version 4.1, 1 processes
```

```
1
```

Die Zeile, die Meep beim Import ausgibt, ist schon die erste Wiedererkennung: Es meldet seine **MPI**-Version — das Programm ist von Haus aus dafür gebaut, eine große Simulation auf viele Prozessorkerne zu verteilen. Für unsere 2D-Experimente reicht ein Kern; ab echten 3D-Rechnungen wird daraus ein entscheidender Vorteil.

11.2 Maxwell hat kein Lineal: Meeps Einheiten

Bevor die erste Simulation läuft, eine Eigenheit, über die jeder Meep-Neuling stolpert: Meep fragt nie nach Metern oder Hertz. Eine Zelle ist „8 mal 8" — acht *was*?

Die Antwort steckt in den Maxwell-Gleichungen selbst. Im Vakuum enthalten sie genau **eine** Naturkonstante, die Geschwindigkeit c — aber keine Länge, keine Zeit, keine Frequenz. Vergrößert man ein Experiment in allen Längen um einen Faktor und streckt die Zeit um denselben Faktor, erfüllen die gestreckten Felder wieder exakt die Maxwell-Gleichungen: Nichts in ihnen kann den Maßstab bemerkt haben. Das heißt aber: **Eine einzige Simulation gilt für alle Maßstäbe gleichzeitig**, solange die Geometrie *in Wellenlängen gemessen* dieselbe bleibt — eine Antenne für 2,4 GHz und ihre tausendfach geschrumpfte Kopie für Licht sind dieselbe Rechnung. Diese **Skaleninvarianz** ist uns übrigens längst vertraut: Schon unser eigener Update-Code kannte weder Δx noch Δt einzeln, sondern nur ihr Verhältnis in der Courant-Zahl S — und Kapitel 8 hat eingeschärft, dass für die Genauigkeit allein *Punkte pro Wellenlänge* zählen.

Meep macht daraus ein Einheitensystem: Es setzt $c = 1$ und überlässt dir die Wahl einer **Bezugslänge** a. Alle Längen sind dann Vielfache von a, alle Zeiten Vielfache von a/c (die Zeit, in der Licht die Strecke a schafft), alle Frequenzen Vielfache von c/a. Daraus folgt die wichtigste Umrechnungsformel der Meep-Welt — eine Frequenz in Meep-Einheiten ist nichts als ein Längenverhältnis:

$$f_{\text{meep}} = \frac{f_{\text{phys}}}{c/a} = \frac{a}{\lambda}.$$

Die Handrechnung dazu, an einem konkreten Zahlenpunkt: Eine Quelle schwinge mit $f_{\text{meep}} = 0{,}5$. Ihre Wellenlänge ist dann $\lambda = a/0{,}5 = 2a$ — zwei Bezugslängen, fertig, mehr sagt die Simulation nicht. Erst *deine* Wahl von a macht Physik daraus. Wählst du $a = 1$ µm (Photonik-Konvention), ist $\lambda = 2$ µm und $f_{\text{phys}} = c/\lambda = (3 \cdot 10^8 \text{ m/s})/(2 \cdot 10^{-6} \text{ m}) \approx 1{,}5 \cdot 10^{14}$ Hz — Infrarotlicht. Wählst du $a = 1$ cm, ist $\lambda = 2$ cm und $f_{\text{phys}} \approx 15$ GHz —

Radartechnik. Das Ergebnis ist in beiden Fällen **derselbe Lauf**; nur das Etikett wechselt:

```
# von oben: nichts (eigenständige Handrechnung); c als Zahl
c = 299792458.0                    # m/s — als Variable, Meep kennt kein SI
f_meep = 0.5

for a, einheit in ((1e-6, "1 µm"), (1e-2, "1 cm")):
    lam = a / f_meep               # λ = a / f_meep, die Formel von oben
    print(f"a = {einheit}:  λ = {lam:.2e} m,"
          f"  f = {c / lam:.3e} Hz")
```

```
a = 1 µm:  λ = 2.00e-06 m,  f = 1.499e+14 Hz
a = 1 cm:  λ = 2.00e-02 m,  f = 1.499e+10 Hz
```

Eine Zahl, zwei Wirklichkeiten — 150 THz Lichttechnik und 15 GHz Radartechnik aus demselben Lauf. Im restlichen Kapitel arbeiten wir direkt in Meep-Einheiten und sparen uns das Etikett; die Übung Ü 11.2 rechnet zur Kontrolle einmal ein WLAN-Problem um.

11.3 Die Ringwelle, zweite Auflage

Jetzt das erste Experiment — und zwar als **Querrechnung** im Sinne der Checkliste aus Kapitel 10: dasselbe Problem, zwei unabhängige Programme, und am Ende wird verglichen. **Die Frage:** Liefert Meep dasselbe Feld wie unser eigener 2D-Code aus Kapitel 9? **Die Bühne:** ein Quadrat von 8×8 Meep-Einheiten, Auflösung 50 Punkte pro Einheit, also $\Delta x = 0{,}02$ und ein Gitter von 400×400 Ez-Punkten; gerechnet wird mit Meeps Standard-Courant-Zahl $S = 0{,}5$ (sicher unter der 2D-Grenze $1/\sqrt{2}$ aus Kapitel 9), also $\Delta t = S \cdot \Delta x = 0{,}01$. **Anfangsbedingung:** alle Felder null; angestoßen wird mit einer *Stromquelle* — Meep addiert in jedem Schritt einen vorgegebenen Strom $J(t)$ auf einen Gitterpunkt, das ist exakt unsere weiche Quelle aus Kapitel 6. Als Zeitverlauf geben wir einen Gauß-Puls vor (Mitte $t_0 = 0{,}6$, Breite $\tau = 0{,}18$). **Randbedingung:** gar keine — lassen wir die PML weg, schließt Meep die Zelle mit PEC-Metallwänden ab, genau wie unser Code. **Gemessen wird** ein Schnappschuss des Ez-Feldes bei $t = 3$, kurz bevor der Ring die Wände erreicht. **Erfolgskriterium:** Die normierten Felder beider Programme unterscheiden sich höchstens um wenige Prozent — bei zwei Programmen, die dasselbe Verfahren auf demselben Gitter rechnen, darf man streng sein.

Ein Aufbaudetail ist begründungspflichtig: Die Quelle sitzt nicht bei $(0, 0)$, sondern beim krummen $(0{,}01,\ 0{,}01)$. Das ist kein Tippfehler, sondern der Yee-Versatz: Meeps Ez-Punkte liegen um einen *halben* Gitterabstand neben den Zellkanten, der Ursprung $(0, 0)$ liegt also *zwischen* vier Ez-Punkten. Wir setzen die Quelle exakt auf den nächstgelegenen Ez-Punkt — sonst verschmiert Meep den Strom auf mehrere Nachbarpunkte, und unser Eigenbau (der die Quelle auf genau einen Punkt setzt) würde Äpfel mit halbverschobenen Birnen vergleichen.

Zuerst die gemeinsame Bühne und unser eigenes Programm — die drei `np.diff`-Zeilen aus Kapitel 9, wörtlich übernommen:

```
# von oben: nichts (Bühne wird hier definiert)
GROESSE = 8.0          # Zellgröße (Meep-Einheiten)
AUFL = 50              # Meeps "resolution": Gitterpunkte pro Einheit
DX = 1.0 / AUFL        # Δx = 0,02
S = 0.5                # Meeps Standard-Courant-Zahl
DT = S * DX            # Δt = 0,01 (c = 1)
T0, TAU = 0.6, 0.18    # Gauß-Puls der Stromquelle
T_BILD = 3.0           # Schnappschuss-Zeit
NX = int(GROESSE * AUFL)                 # 400 Ez-Punkte je Richtung
IQ = NX // 2                             # Quellindex 200 …
XQ = (IQ + 0.5) * DX - GROESSE / 2       # … liegt bei x = +0,01


def quellpuls(t):
    """Der Strom-Zeitverlauf — für beide Programme derselbe."""
    return np.exp(-((t - T0) / TAU) ** 2)


def eigener_lauf(versatz=1.0):
    """Der 2D-Yee-Code aus Kapitel 9 (PEC-Ränder, weiche Quelle).

    versatz: zu welcher Zeit (n + versatz)·Δt der Quellstrom im
    n-ten Schritt ausgewertet wird — Voreinstellung: die Zeit, auf
    der die Felder nach dem Schritt gerade angekommen sind.
    """
    e = np.zeros((NX, NX))
    ux = np.zeros((NX, NX - 1))
    uy = np.zeros((NX - 1, NX))
    for n in range(int(round(T_BILD / DT))):
        ux -= S * np.diff(e, axis=1)
        uy += S * np.diff(e, axis=0)
        e[1:-1, 1:-1] += S * (np.diff(uy, axis=0)[:, 1:-1]
```

```
                                - np.diff(ux, axis=1)[1:-1, :])
        e[IQ, IQ] -= quellpuls((n + versatz) * DT) * DT   # Strom J·Δt
    return e


e_eigen = eigener_lauf()
print(f"eigener Lauf fertig: {int(round(T_BILD / DT))} Schritte,"
      f" max|Ez| = {np.abs(e_eigen).max():.2e}")
```

```
eigener Lauf fertig: 300 Schritte, max|Ez| = 1.07e-04
```

(Das Minuszeichen vor der Quelle übernimmt Meeps Vorzeichenwahl: In der Ampère-Gleichung steht der Strom als $-J$ auf der Seite von $\partial E/\partial t$. Für den Vergleich ist das nur eine Konvention — aber wer querrechnet, muss Konventionen angleichen.)

Und jetzt dasselbe Experiment in Meep. Die gesamte Physik — Gitter, Update-Gleichungen, Quelle, Ränder — steckt in einem `Simulation`-Objekt:

```
# von oben: GROESSE, AUFL, XQ, T0, T_BILD, quellpuls()
quelle = [mp.Source(                        # mp.Source: eine Feldquelle
    mp.CustomSource(quellpuls,              # … mit frei wählbarem Zeitverlauf
                    end_time=10 * T0),      #     (danach gilt die Quelle als aus)
    component=mp.Ez,                        # … speist die Ez-Komponente (TMz!)
    center=mp.Vector3(XQ, XQ))]             # mp.Vector3: ein Ort (x, y[, z])

sim = mp.Simulation(                        # die ganze Simulation als Objekt
    cell_size=mp.Vector3(GROESSE, GROESSE),  # 8 × 8; kein z → Meep rechnet 2D
    resolution=AUFL,                        # Gitterpunkte pro Längeneinheit
    sources=quelle,
    boundary_layers=[])                     # keine Randschichten → PEC-Wände

sim.run(until=T_BILD)                       # laufen lassen bis t = 3

# get_array: das Feld als NumPy-Array herausholen (hier: die ganze Zelle)
ez_meep = sim.get_array(component=mp.Ez, center=mp.Vector3(),
                        size=mp.Vector3(GROESSE, GROESSE))
print(f"Meep fertig: Feld {ez_meep.shape}, max|Ez| = {np.abs(ez_meep).max():.2e}")
```

```
Meep fertig: Feld (400, 400), max|Ez| = 2.65e-01
```

Kein eigener Zeitschritt, keine `diff`-Zeilen — alles, was wir in Kapitel 9 gebaut haben, verbirgt sich hinter `sim.run()`. Aber die Stellschrauben sind dieselben geblieben: `resolution` ist unser „Punkte pro Wellenlänge"-Regler aus Kapitel 8 (nur auf die Bezugslänge statt auf λ bezogen), die Courant-Zahl steht intern auf $0{,}5$ und ließe sich mit `Courant=` verstellen (Ü 11.4 tut das — mit Ansage), und `boundary_layers` entscheidet über das Schicksal der Wellen am Rand.

> **Vorhersage-Punkt**
>
> Zwei Programme, dasselbe Yee-Verfahren, dasselbe Gitter, dieselbe Quelle: Was erwartest du vom Vergleich — (a) Bilder, die sich qualitativ ähneln, (b) Übereinstimmung auf ein paar Prozent oder (c) identische Zahlen bis zur Maschinengenauigkeit?

Festgelegt? Dann lassen wir die beiden Programme nebeneinander antreten — zuerst entscheiden die Bilder, danach die Zahlen:

```
# von oben: e_eigen, ez_meep
fig, achsen = plt.subplots(1, 2, figsize=(7.0, 3.6))
for ax, feld, titel in ((achsen[0], e_eigen, "eigener Kapitel-9-Code"),
                        (achsen[1], ez_meep, "Meep")):
    w = np.abs(feld).max() / 3
    ax.imshow(feld.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w,
              extent=(-4, 4, -4, 4))
    ax.set_title(titel, fontsize=10)
    ax.set_xlabel("x (Meep-Einheiten)")
achsen[0].set_ylabel("y (Meep-Einheiten)")
plt.tight_layout(); plt.show()
```

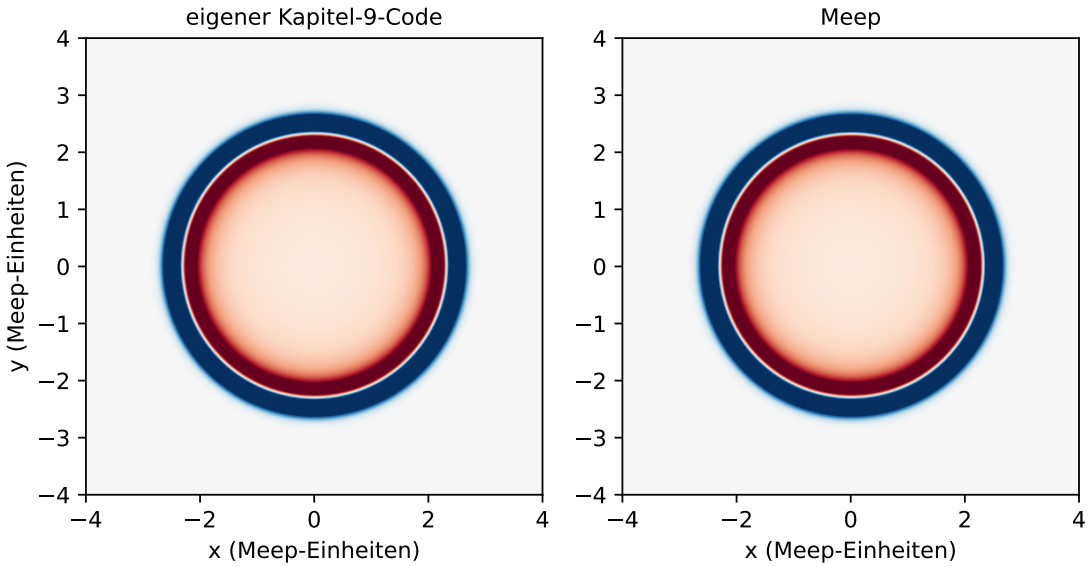


Abbildung 11.1: Dieselbe Ringwelle zweimal (Ez als Farbe, beide auf ihr Maximum normiert): links unser Kapitel-9-Code, rechts Meep — gleiche Quelle, gleiches Gitter, gleiche Courant-Zahl. Mit dem Auge ist kein Unterschied zu finden; auch der helle Innenbereich (das 2D-Nachleuchten aus Kapitel 9) ist in beiden identisch. Ob die Übereinstimmung auch den Zahlen standhält, klärt der nächste Abschnitt.

Die Bilder sind nicht zu unterscheiden — und auch die erste Kontrollzahl passt: Der Wellenkamm sitzt in beiden Läufen am selben Ort.

```
# von oben: e_eigen, ez_meep, NX, IQ, DX, XQ, T_BILD, T0
x_achse = (np.arange(NX) + 0.5) * DX - GROESSE / 2   # Meeps Ez-Gitterachse

schnitt_eigen = e_eigen[IQ, :] / np.abs(e_eigen[IQ, :]).max()
schnitt_meep = ez_meep[IQ, :] / np.abs(ez_meep[IQ, :]).max()

rechts = x_achse > XQ + 1.0                # Kammsuche: rechts, außerhalb
r_eigen = x_achse[rechts][np.argmax(np.abs(schnitt_eigen[rechts]))] - XQ
r_meep = x_achse[rechts][np.argmax(np.abs(schnitt_meep[rechts]))] - XQ
print(f"Kammradius: eigener Code {r_eigen:.2f}, Meep {r_meep:.2f}"
      f"   (zum Vergleich: c·(t − t₀) = {T_BILD - T0:.2f})")
```

```
Kammradius: eigener Code 2.46, Meep 2.46   (zum Vergleich: c·(t − t₀) = 2.40)
```

11.4 Querrechnen heißt: der Differenz zuhören

„Sieht gleich aus" ist beim Querrechnen wertlos — das hat uns die Refutation in Kapitel 10 gelehrt. Also Zahlen: Wir normieren beide Felder auf ihr Maximum und nehmen die größte punktweise Differenz im gesamten 400×400-Feld.

```
# von oben: e_eigen, ez_meep
feld_eigen = e_eigen / np.abs(e_eigen).max()
feld_meep = ez_meep / np.abs(ez_meep).max()
print(f"max |Differenz der normierten Felder| ="
      f" {np.abs(feld_eigen - feld_meep).max() * 100:.2f} %")
```

```
max |Differenz der normierten Felder| = 4.73 %
```

Fast fünf Prozent. Für zwei *verschiedene* Verfahren wäre das ein gutes Ergebnis — für zwei Programme, die **dasselbe Verfahren auf demselben Gitter** rechnen, ist es viel zu viel. Genau hier trennt sich gutes Querrechnen von Augenwischerei: Eine unerklärte Differenz ist kein Schönheitsfehler, sondern eine Botschaft. Irgendetwas machen die beiden Programme verschieden, und wir sollten wissen, was.

Gehen wir die Verdächtigen durch. Das Gitter? Ist mit dem krummen Quellort absichtlich deckungsgleich gemacht. Die Normierung? Beide Felder sind auf ihr Maximum skaliert. Die Quellform? Punkt für Punkt dieselbe Funktion. Bleibt — die Zeit. In unserem Code steckt eine stillschweigende Entscheidung, die wir in Kapitel 9 nie hinterfragt haben: *Zu welchem Zeitpunkt* werten wir den Quellstrom aus? Der n-te Schritt befördert das E-Feld von $n\,\Delta t$ nach $(n+1)\,\Delta t$; unsere Voreinstellung füttert die Quelle mit der Ankunftszeit $(n+1)\,\Delta t$, genauso gut hätten wir die Startzeit $n\,\Delta t$ nehmen können. Aber Kapitel 5 hat uns beigebracht, wie der Leapfrog über solche Fragen denkt: Das E-Update integriert über das Intervall von n bis $n+1$, und was dabei *während* des Intervalls wirkt — wie das B-Feld, das auf den halben Schritten lebt —, gehört in die **Mitte**, auf $(n+\frac{1}{2})\,\Delta t$. Wenn Meep den Strom wie ein sauberes Leapfrog-Schema behandelt, müsste genau diese Wahl die Differenz zusammenfallen lassen. Der Test — alle drei Lesarten gegen denselben Meep-Lauf:

```
# von oben: eigener_lauf(), feld_meep
for versatz, name in ((0.0, "n·Δt        (Startzeit)"),
                      (1.0, "(n+1)·Δt    (Ankunftszeit)"),
                      (0.5, "(n+½)·Δt    (Leapfrog-Mitte)")):
    feld_v = eigener_lauf(versatz)
```

```
    feld_v = feld_v / np.abs(feld_v).max()
    print(f"Quellzeit {name}: "
          f"max|Differenz| = {np.abs(feld_v - feld_meep).max() * 100:.2f} %")
```

```
Quellzeit n·Δt        (Startzeit): max|Differenz| = 4.14 %
Quellzeit (n+1)·Δt    (Ankunftszeit): max|Differenz| = 4.73 %
Quellzeit (n+½)·Δt    (Leapfrog-Mitte): max|Differenz| = 0.62 %
```

Beide „naheliegenden" Wahlen liegen über vier Prozent daneben — die Leapfrog-Mitte trifft auf $0{,}62\,\%$. Meep wertet den Quellstrom also am halben Zeitschritt aus, dort, wo er nach der Logik des Yee-Schemas hingehört. Das ist die schönste Sorte Querrechnen-Ergebnis: Die Differenz war kein Fehler, sondern eine **Lektion** — sie hat uns eine Konvention im Inneren von Meep verraten und nebenbei gezeigt, dass unser eigener Kapitel-9-Code die Quelle einen halben Schritt zu spät gefüttert hat (für alle Messungen dort war das egal: ein gemeinsamer Zeitversatz der ganzen Welle ändert weder Tempo- noch Stabilitätsbefunde).

```
# von oben: eigener_lauf(), schnitt_meep, x_achse, IQ
e_halb = eigener_lauf(0.5)
schnitt_halb = e_halb[IQ, :] / np.abs(e_halb[IQ, :]).max()

fig, ax = plt.subplots(figsize=(6.6, 3.2))
ax.plot(x_achse, schnitt_halb, lw=1.6, label="eigener Code, Quelle bei (n+½)Δt")
ax.plot(x_achse, schnitt_meep, "--", lw=1.2, label="Meep")
ax.plot(x_achse, (schnitt_halb - schnitt_meep) * 10 - 1.5, lw=0.8,
        color="tab:red", label="Differenz × 10 (versetzt)")
ax.axhline(-1.5, color="gray", lw=0.5)
ax.set_xlabel("x (Meep-Einheiten)"); ax.set_ylabel("Ez (normiert)")
ax.legend(fontsize=8, loc="upper right")
plt.tight_layout(); plt.show()
```

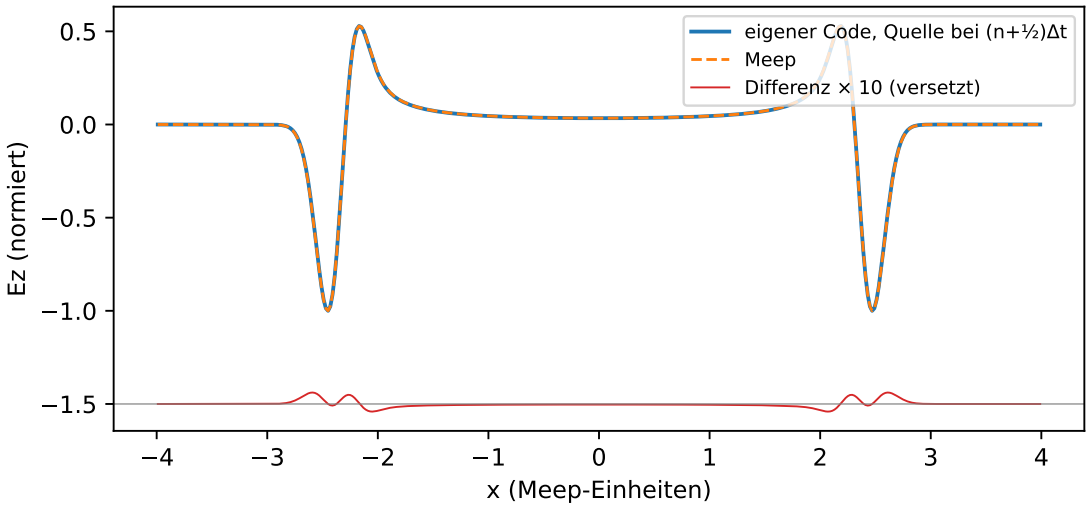


Abbildung 11.2: Schnitt durch die Quelle nach der Quellzeit-Korrektur: eigener Code (durchgezogen) und Meep (gestrichelt) liegen aufeinander; die rote Kurve zeigt die zehnfach vergrößerte Differenz (nach unten versetzt, graue Nulllinie). Die verbliebenen 0,6 % wohnen auf den steilsten Flanken des Ringkamms — wo ein winziger Rest Unterschied in der Quellbehandlung am stärksten durchschlägt.

Woher kommt der letzte halbe Prozentpunkt? Ein weiterer Zeitversatz ist es nicht — verschiebt man die Quellzeit testweise in kleinen Schritten, ist $\frac{1}{2}$ exakt das Minimum. Übrig bleiben feinere Unterschiede in der Quellbehandlung (Meep glättet und integriert Quellen etwas anders als unser nackter Ein-Punkt-Strom), und die schlagen genau dort zu Buche, wo das Feld am steilsten ist. Für die Praxis ist das ein Spitzenwert: Zur Einordnung — als später NEC und openEMS dieselbe Antenne rechneten (Kapitel 11.8), galten zehn Prozent Unterschied im Gewinn als Bestätigung.

11.5 Alte Bekannte I: die PML

Jetzt drehen wir den einen Schalter um, der in Kapitel 9 viel Arbeit war: offene Ränder. Bei uns hieß das Split-Feld-PML — zwei Teilfelder, vier Verlustprofile, sorgfältig abgestimmte Koeffizienten. In Meep heißt es `boundary_layers=[mp.PML(1.0)]`: eine absorbierende Schicht von einer Längeneinheit Dicke, innen an alle Wände gelegt.

Der Versuchsaufbau für die Messung: **Frage** — wie viel reflektiert Meeps PML? **Bühne, Quelle:** wie im Ringexperiment. **Ablauf:** Wir lassen länger laufen, bis $t = 7$; der Ring (Radius $c \cdot (t - t_0) = 6{,}4$) ist dann längst in den Rändern (bei $4{,}0$) verschwunden. **Gemessen wird** das Restfeld im zentralen 3×3-Quadrat, relativ zur Ringamplitude aus dem ersten Lauf. **Erfolgskriterium:** deutlich unter den $\sim 15\,\%$ der σ-Rampe aus Kapitel 9 — die PML soll ihren Namen verdienen. Und weil wir inzwischen misstrauische Messtechniker sind, bauen wir eine **Referenz** ein: denselben Lauf in einer doppelt so großen Zelle (16×16, ohne PML). Dort sind die Wände so weit weg, dass bis $t = 7$ keine Reflexion ins Zentrum zurückkehrt — dieses Feld zeigt die *reine Physik*, und die punktweise Differenz zum PML-Lauf isoliert exakt das, was der Rand verschuldet.

Vorhersage-Punkt

Erinnerung an Kapitel 9: Nach dem Ringdurchgang blieb die Mitte nicht leer — die 2D-Linienquelle leuchtet nach. Wenn wir gleich Restfeld im

```
# von oben: GROESSE, AUFL, XQ, T0, NX, quellpuls()
def meep_lauf(groesse, pml, courant=0.5, geometrie=(), bis=3.0):
    """Eine Meep-Simulation wie oben; gibt das Ez-Feld bei t=bis zurück."""
    q = [mp.Source(mp.CustomSource(quellpuls, end_time=10 * T0),
                   component=mp.Ez, center=mp.Vector3(XQ, XQ))]
    sim = mp.Simulation(cell_size=mp.Vector3(groesse, groesse),
                        resolution=AUFL, sources=q, Courant=courant,
                        geometry=list(geometrie),       # Objekte in der Zelle
                        boundary_layers=[mp.PML(1.0)] if pml else [])
    sim.run(until=bis)
    return sim.get_array(component=mp.Ez, center=mp.Vector3(),
                         size=mp.Vector3(groesse, groesse))


ez_pml = meep_lauf(GROESSE, pml=True, bis=7.0)
ez_pec7 = meep_lauf(GROESSE, pml=False, bis=7.0)       # PEC zum Kontrast
ez_gross = meep_lauf(2 * GROESSE, pml=False, bis=7.0)  # die Referenzbox
ez_ref = ez_gross[NX // 2:NX // 2 + NX, NX // 2:NX // 2 + NX]  # innerer Teil

ring_amp = np.abs(ez_meep).max()         # Ringamplitude bei t = 3 (von oben)
innen = slice(NX // 2 - 75, NX // 2 + 75)              # zentrale 3 × 3
kern = slice(75, NX - 75)                # alles außerhalb der PML-Schicht
rest_pml = np.abs(ez_pml[innen, innen]).max()
```

```
nachleuchten = np.abs(ez_ref[innen, innen]).max()
reflexion = np.abs(ez_pml[kern, kern] - ez_ref[kern, kern]).max()
print("nach dem Ringdurchgang, relativ zur Ringamplitude:")
print(f"  Restfeld im Zentrum (PEC-Wände): "
      f"{np.abs(ez_pec7[innen, innen]).max() / ring_amp * 100:5.1f} %")
print(f"  Restfeld im Zentrum (Meep-PML) :  {rest_pml / ring_amp * 100:.2f} %")
print(f"  reine Physik (Referenzbox)     :  "
      f"{nachleuchten / ring_amp * 100:.2f} %")
print(f"  echte PML-Reflexion (Differenz):  {reflexion / ring_amp:.1e}")
```

```
nach dem Ringdurchgang, relativ zur Ringamplitude:
  Restfeld im Zentrum (PEC-Wände):  55.9 %
  Restfeld im Zentrum (Meep-PML) :  0.56 %
  reine Physik (Referenzbox)     :  0.56 %
  echte PML-Reflexion (Differenz):  7.0e-07
```

Drei Lehren auf einmal. Erstens: Mit PEC-Wänden bleibt erwartungsgemäß alles im Kasten — gut die Hälfte der Ringamplitude schwappt als Echo durcheinander. Zweitens: Hinter Meeps PML bleiben $0{,}56\,\%$ übrig — aber die Referenzbox entlarvt, *was* da übrig bleibt: Es ist haargenau das **2D-Nachleuchten** aus Kapitel 9, die ganz normale Physik einer Linienquelle, kein Randeffekt. Wer dem Vorhersage-Punkt aufgesessen ist und es der PML angekreidet hätte, hat soeben am eigenen Leib erfahren, warum Messmethoden geprüft gehören. Drittens, und das ist die eigentliche Zahl: Die **echte Reflexion** von Meeps PML — die Differenz zwischen PML-Lauf und Referenz — liegt bei $7 \cdot 10^{-7}$ der Ringamplitude. Unter einem **Millionstel**. Unsere selbstgebaute Split-Feld-PML aus Kapitel 9 war stolz auf ihre $0{,}4\,\%$; Meeps PML (eine weiterentwickelte Variante mit gestreckten Koordinaten und optimiertem Profil) spielt mehrere Ligen darüber. Genau für solche Unterschiede bezahlt man gern mit einem `import`.

```
# von oben: ez_pec7, ez_pml, ring_amp, kern
fig, achsen = plt.subplots(1, 3, figsize=(9.6, 3.4))
felder = ((ez_pec7, ring_amp / 3, "PEC-Wände, t = 7"),
          (ez_pml, ring_amp / 3, "Meep-PML, t = 7"),
          (ez_pml, np.abs(ez_pml[kern, kern]).max(),
           "Meep-PML, Farbskala aufs Restfeld"))
for ax, (feld, w, titel) in zip(achsen, felder):
    ax.imshow(feld.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w,
              extent=(-4, 4, -4, 4))
    ax.set_title(titel, fontsize=9)
    ax.set_xlabel("x (Meep-Einheiten)")
```

```
achsen[0].set_ylabel("y (Meep-Einheiten)")
plt.tight_layout(); plt.show()
```

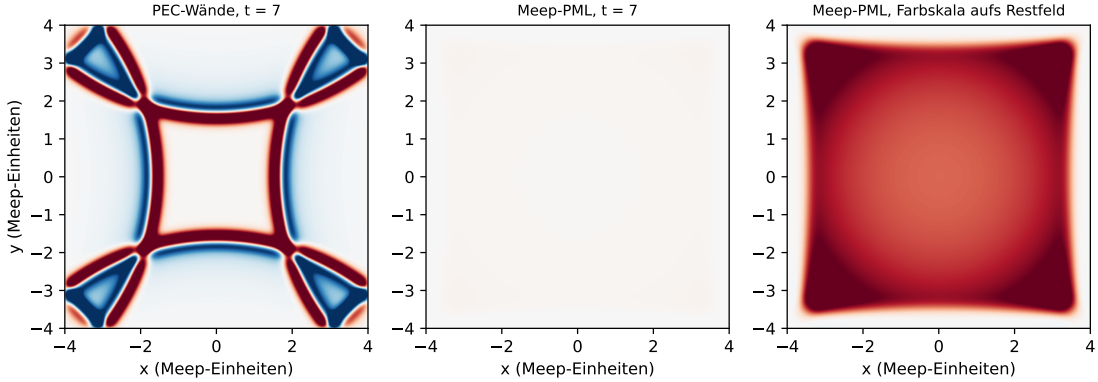


Abbildung 11.3: Die Zelle bei t = 7, nachdem der Ring die Ränder erreicht hat: Links mit PEC-Wänden (Farbskala des Ringexperiments) — die Zelle ist voller Echos. Mitte: mit Meeps PML, gleiche Skala — leer. Rechts dasselbe PML-Feld, Farbskala um den Faktor ≈ 140 auf das Restfeld gespreizt: kein Echo-Muster, sondern ein flacher, durchweg positiver Schimmer, der nach außen sanft zunimmt — das 2D-Nachleuchten der Linienquelle aus Kapitel 9 (nahe am verschwundenen Ring ist es am jüngsten und stärksten). Die echte PML-Reflexion ($7{\cdot}10^{-7}$) wäre auch auf dieser Skala unsichtbar.

11.6 Alte Bekannte II: der eingebaute Detektor

In Kapitel 10 haben wir Spektren so gemessen: Zeitreihe an einer Sonde aufzeichnen, am Ende `rfft`. Meep kann das eleganter — ein **DFT-Monitor** rechnet das Multiplizieren-und-Mitteln unseres Detektors *während des Laufs* mit, für jede gewünschte Frequenz (deshalb braucht er die Zeitreihe nie zu speichern; bei großen 3D-Feldern ist das der einzige gangbare Weg). Die Behauptung „das ist exakt unser Detektor" ist natürlich messpflichtig.

Der Aufbau: Ringexperiment mit PML, eine Sonde eine Längeneinheit rechts der Quelle. Dort lassen wir **beide** Messgeräte gleichzeitig laufen — Meeps DFT-Monitor für 14 Testfrequenzen zwischen $0{,}3$ und $1{,}6$, und parallel zeichnen wir die Zeitreihe selbst auf und halten hinterher den Kapitel-10-Detektor ($2\langle s \cdot \sin\rangle$, $2\langle s \cdot \cos\rangle$, `np.hypot`) an dieselben Frequenzen. Aufnahmedauer $T = 12$. **Erfolgskriterium:** Beide Spektren stimmen

bis auf einen konstanten Eichfaktor überein — konstant heißt: dasselbe Verhältnis bei allen 14 Frequenzen, sonst wäre es keine Eichung, sondern eine Abweichung.

```
# von oben: GROESSE, AUFL, XQ, T0, DT, quellpuls()
T_AUF = 12.0
frequenzen = np.linspace(0.3, 1.6, 14)
q = [mp.Source(mp.CustomSource(quellpuls, end_time=10 * T0),
               component=mp.Ez, center=mp.Vector3(XQ, XQ))]
sim = mp.Simulation(cell_size=mp.Vector3(GROESSE, GROESSE), resolution=AUFL,
                    sources=q, boundary_layers=[mp.PML(1.0)])

sonde_ort = mp.Vector3(XQ + 1.0, XQ)
# add_dft_fields: der eingebaute Detektor – sammelt Ez bei den
# Testfrequenzen auf einem kleinen Fleck um die Sonde
dft = sim.add_dft_fields([mp.Ez], frequenzen.tolist(), center=sonde_ort,
                         size=mp.Vector3(2 * DX, 2 * DX))

reihe = []                                  # unsere eigene Zeitreihe
# mp.at_every(Δt, f): ruft f in jedem Zeitschritt auf;
# get_field_point: ein Feldwert an einem Ort
sim.run(mp.at_every(DT, lambda s: reihe.append(
    s.get_field_point(mp.Ez, sonde_ort).real)), until=T_AUF)


def dft_wert(k):
    """|Ez(f_k)| am Sondenort (Mitte des Monitor-Flecks)."""
    # np.atleast_2d: macht aus einem Skalar/Vektor notfalls eine
    # 2D-Matrix – der Monitor-Fleck kann je nach Größe beides liefern
    a = np.atleast_2d(sim.get_dft_array(dft, mp.Ez, k))
    return np.abs(a[a.shape[0] // 2, a.shape[1] // 2])


spektrum_meep = np.array([dft_wert(k) for k in range(len(frequenzen))])
reihe = np.array(reihe)
t = np.arange(len(reihe)) * DT
spektrum_hand = np.array(                   # der Detektor aus Kapitel 10
    [np.hypot(2 * np.mean(reihe * np.sin(2 * np.pi * f * t)),
              2 * np.mean(reihe * np.cos(2 * np.pi * f * t)))
     for f in frequenzen])

verhaeltnis = spektrum_meep / spektrum_hand
print(f"Verhältnis Meep-Monitor / Hand-Detektor über 14 Frequenzen:")
print(f"  {verhaeltnis.mean():.4f} ± {verhaeltnis.std():.1e}")
```

```
print(f"  zum Vergleich, T/(2·√(2π)) = "
      f"{T_AUF / (2 * np.sqrt(2 * np.pi)):.4f}")
```

```
Verhältnis Meep-Monitor / Hand-Detektor über 14 Frequenzen:
  2.3937 ± 2.7e-15
  zum Vergleich, T/(2·√(2π)) = 2.3937
```

Das Verhältnis ist konstant bis auf $3 \cdot 10^{-15}$ — Maschinengenauigkeit, wie beim `rfft`-Vergleich in Kapitel 10: Meeps Monitor **ist** unser Detektor. Und selbst der Eichfaktor lässt sich entzaubern: Meep summiert $s(t)\, e^{i\omega t}\, \Delta t/\sqrt{2\pi}$ statt zu mitteln und mit 2 zu multiplizieren — das Verhältnis der beiden Konventionen ist $T/(2\sqrt{2\pi}) = 2{,}3937$, auf alle vier Stellen genau der gemessene Wert. Wo in Kapitel 10 der Faktor $2/N$ stand, steht hier eben $\Delta t/\sqrt{2\pi}$; Eichungen sind Verabredungen, keine Physik.

Damit ist die Werkstatt-Tour komplett: Quelle, Courant, Gitter, PML, Detektor — wir haben jedes Bauteil von Meep an unserem Eigenbau gemessen und verstehen, was es tut. Zeit für den Blick über Meep hinaus.

11.7 Die Landschaft der Verfahren

⚠ Naheliegende Vermutung

„Es muss doch ein bestes Simulationsprogramm geben — das nimmt man dann für alles."
Warum sie naheliegt: Bei Alltagssoftware funktioniert das ja: Ein guter Browser zeigt jede Webseite, eine gute Tabellenkalkulation rechnet jede Tabelle. Warum sollte ein „guter Feldlöser" nicht jedes Feld lösen?
Was stattdessen stimmt: Hinter jedem Löser steht eine Entscheidung, **was diskretisiert wird** — das Raumvolumen, nur die Metalloberflächen oder gleich nur Strahlwege. Diese Entscheidung macht ihn für eine Klasse von Problemen brillant und für andere prinzipiell ungeeignet; kein Programmieraufwand der Welt ändert das. Meep löst ein Volumengitter und scheitert darum ehrlich an einem 1 mm dünnen Antennendraht (das Gitter müsste den Draht auflösen — bei 12 cm Wellenlänge ein Auflösungs-Albtraum). NEC diskretisiert nur die Drähte selbst und erledigt dieselbe Antenne in Millisekunden — kann dafür aber mit einem Klotz aus Glas praktisch nichts anfangen. Werkzeugwahl ist Teil der Physik, nicht der Geschmackssache.

Die großen Familien, sortiert nach dem, was sie diskretisieren:

Tabelle 11.1: Die großen Verfahrensfamilien

Familie	rechnet auf	ideal für	schwach bei	freie Software
FDTD, Zeitbereich	Volumen + Zeit	Wellen unterwegs; Breitband (ein Puls, alle Frequenzen — Kap. 10); Photonik; dispersive Stoffe	dünne Drähte; el. riesige Objekte; viel Leerraum	**Meep**; **openEMS** (FIT, mit Dünndraht-Modell)
MoM, Momenten-Methode	Drähte und Leiter-Flächen	Drahtantennen; Abstrahlung im Freiraum; Impedanzen	große Volumen aus Dielektrikum	**NEC2** (`necpp`, 4nec2, xnec2c)
FEM, meist Frequenz-Bereich	Volumen, freies Netz	komplexe 3D-Geometrie; Resonatoren; eine Frequenz je Lauf	sehr breitbandig; sehr groß	FreeFEM, Elmer (komm.: HFSS, COMSOL)
Strahlen (PO/GO/UTD)	Strahlwege	elektrisch Riesiges (Radarecho, Funkfeld einer Stadt)	Details in Wellenlängen-Größe	(in kommerziellen Hybriden, z. B. FEKO)

Die Faustregel zum Mitnehmen knüpft an das Leitmotiv aus Kapitel 8 an — es zählen nur **Verhältnisse zur Wellenlänge**: Strukturen *um* λ herum (Photonik, Resonatoren, Metamaterialien) → Volumengitter, also FDTD/FEM. Dünne Leiter (Radius $\ll \lambda$) in viel Freiraum → MoM. Objekte $\gg \lambda$ → Strahlverfahren. Und quer durch alle Familien gilt: **scipy** ist das Werkzeug *zwischen* den Werkzeugen — seine dünn besetzten Matrizen und Löser (Kapitel 7), die FFT (Kapitel 10) und seine ODE-Integratoren stecken als Bausteine in halb der Landschaft, und für ein maßgeschneidertes Problem (unser nächstes Kapitel rechnet Brechung wieder zu Fuß) bleiben sie die erste Wahl.

Noch eine Grenze der Landkarte: Sie gilt vom Mikrowellen- bis ins ultraviolette Regime. Bei 50 Hz (Wellenlänge 6000 km!) simuliert niemand Wellenausbreitung — dort regieren Schaltkreis- und Magnetostatik-Werkzeuge; und jenseits des UV übernimmt die Quantenphysik (Kapitel 30 kommt darauf zurück).

11.8 Die Yagi-Geschichte, zu Ende erzählt

Kapitel 10 hat sie als Warnung zitiert, jetzt bekommt sie ihren Schluss — die Geschichte, der dieses Buch seine Querrechnen-Moral verdankt. Sie handelt von einer **Yagi-Uda-Antenne** (der klassischen Dachantennen-Bauform: ein gespeister Dipol, dahinter ein Reflektorstab, davor mehrere „Direktoren", die die Abstrahlung bündeln), die für 2,4 GHz entworfen und wirklich gebaut werden sollte.

Akt 1 — das falsche Werkzeug. Der erste Versuch lief in Meep, versteht sich: ein 3D-Modell aus dünnen Metallstäben. Es scheiterte nicht an einem Bug, sondern an der Refutation von oben — 1 mm Drahtradius bei 122 mm Wellenlänge sprengt jedes bezahlbare Volumengitter; die Resonanzlängen kamen je nach Auflösung anders heraus. Der Rückzug auf ein **2D-Modell** (Metallstreifen statt Drähte) brachte wunderschöne Bilder: eine saubere Keule, +11 dB Vor/Rück-Verhältnis. Nur: In 2D liegt die Resonanzlänge der Elemente bei $\sim 0{,}32\,\lambda$ — ein Artefakt der Flächenwelt. Wer diese Maße in Kupfer gesägt hätte, hätte Schrott gebaut. Das 2D-Modell war *qualitativ richtig* (so funktioniert eine Yagi!) und *quantitativ unbaubar* — im Rückblick die wertvollste Einsicht der ganzen Saga.

Akt 2 — die beschämend späte Recherche. Dem 2D-Modell ging wochenlanges Herumprobieren am 3D-Modell voraus — Maße tunen, Auflösung erhöhen, wieder tunen; die Keule zeigte hartnäckig nach hinten. Die Lösung stand längst in der Fachliteratur: Yagis rechnet man in der Lehre als 2D-Modell *mit eben jener kurzen Resonanzlänge*, und baubare Entwürfe macht man mit der Momentenmethode. Seither gilt im Buch wie im Labor die Regel: **Bei etablierten Problemen zuerst die Literatur durchsuchen, dann rechnen.** Raten ist die teuerste aller Methoden.

Akt 3 — das richtige Werkzeug. Für dünne Drähte im Freiraum ist die Momentenmethode gemacht: NEC zerlegt jeden Draht in Stromsegmente und löst direkt nach den Strömen — kein Volumengitter, kein Drahtradius-Albtraum (der Radius geht als Parameter in die Gleichungen ein, nicht ins Gitter). Der Versuchsaufbau des folgenden Laufs: **fünf parallele Drahtdipole** (Reflektor, gespeistes Element, drei Direktoren) mit den Maßen des finalen Entwurfs — Längen um $0{,}43$–$0{,}50\,\lambda$, *nicht* $0{,}32$! —, Drahtradius 1 mm, frei im Raum bei 2,45 GHz. **Gemessen werden** das Richtdiagramm in der Horizontalebene, der Gewinn vorn und hinten und die Eingangsimpedanz am Speisepunkt (die entscheidet, ob sich die Antenne überhaupt an ein 50-Ω-Kabel anschließen lässt — Kapitel 22 widmet sich genau dieser Frage). **Erfolgskriterium:** ein Vor/Rück-Verhältnis deutlich über 10 dB und eine Impedanz ohne großen Blindanteil (Resonanz).

```
# von oben: c (der Zahlenwert aus dem Einheiten-Abschnitt)
import necpp

F_MHZ = 2450.0
LAMBDA = c / (F_MHZ * 1e6)                # ≈ 0,122 m
# (Name, Position auf dem Träger in λ, Elementlänge in λ)
YAGI = [("Reflektor",  0.00, 0.500),
        ("Driven",     0.20, 0.473),     # das gespeiste Element
        ("Direktor 1", 0.35, 0.440),
        ("Direktor 2", 0.55, 0.440),
        ("Direktor 3", 0.75, 0.430)]

ctx = necpp.nec_create()                  # eine NEC-Rechnung anlegen
for i, (_, xpos, laenge) in enumerate(YAGI):
    x, halb = xpos * LAMBDA, laenge * LAMBDA / 2
    # nec_wire: ein gerader Draht, in 11 Stromsegmente zerlegt
    # (Anfangs- und Endpunkt in m, letzter Geometrieparameter: Radius 1 mm)
    necpp.nec_wire(ctx, i + 1, 11, x, -halb, 0, x, halb, 0, 0.001, 1.0, 1.0)
necpp.nec_geometry_complete(ctx, 0)       # 0 = Freiraum, kein Erdboden
necpp.nec_fr_card(ctx, 0, 1, F_MHZ, 0)    # die Rechenfrequenz
necpp.nec_ex_card(ctx, 0, 2, 6, 0, 1.0, 0, 0, 0, 0, 0)   # 1 V auf die
#                        ↑ Draht 2 (Driven), Segment 6 (die Mitte)
necpp.nec_rp_card(ctx, 0, 1, 73, 0, 5, 0, 0,             # Richtdiagramm:
                  90.0, 0.0, 0.0, 5.0, 0.0, 0.0)         # Horizont, 5°-Raster
azimut = np.array([necpp.nec_gain(ctx, 0, 0, j) for j in range(73)])
z_re = necpp.nec_impedance_real(ctx, 0)
z_im = necpp.nec_impedance_imag(ctx, 0)
necpp.nec_delete(ctx)

print(f"Gewinn vorwärts : {azimut[0]:5.1f} dBi")
print(f"Gewinn rückwärts: {azimut[36]:5.1f} dBi"
      f"   → Vor/Rück {azimut[0] - azimut[36]:.1f} dB")
print(f"Eingangsimpedanz: {z_re:.1f} {z_im:+.1f}j Ω")
```

```
Gewinn vorwärts :   8.9 dBi
Gewinn rückwärts:  -5.8 dBi   → Vor/Rück 14.6 dB
Eingangsimpedanz: 19.0 +1.2j Ω
```

Millisekunden statt Minuten — und alle Erfolgskriterien erfüllt: fast 9 dBi Gewinn, 14,6 dB Vor/Rück, und der Blindanteil der Impedanz ist praktisch null, die Antenne ist resonant. (Die 19 Ω Wirkanteil sind für Yagis dieser Bauart typisch und verlangen ein Anpassglied ans 50-Ω-Kabel — das Thema von Kapitel 22.)

```
# von oben: azimut
phi = np.radians(np.arange(73) * 5.0)
fig = plt.figure(figsize=(4.6, 4.6))
ax = fig.add_subplot(projection="polar")   # Polardiagramm: Winkel + Radius
ax.plot(phi, np.maximum(azimut, -25))      # Nebenzipfel bei −25 dBi deckeln
ax.set_rmin(-25); ax.set_rmax(10)          # radiale Achse: −25 … +10 dBi
ax.set_title("Yagi-Richtdiagramm (dBi), NEC", fontsize=10)
plt.tight_layout(); plt.show()
```

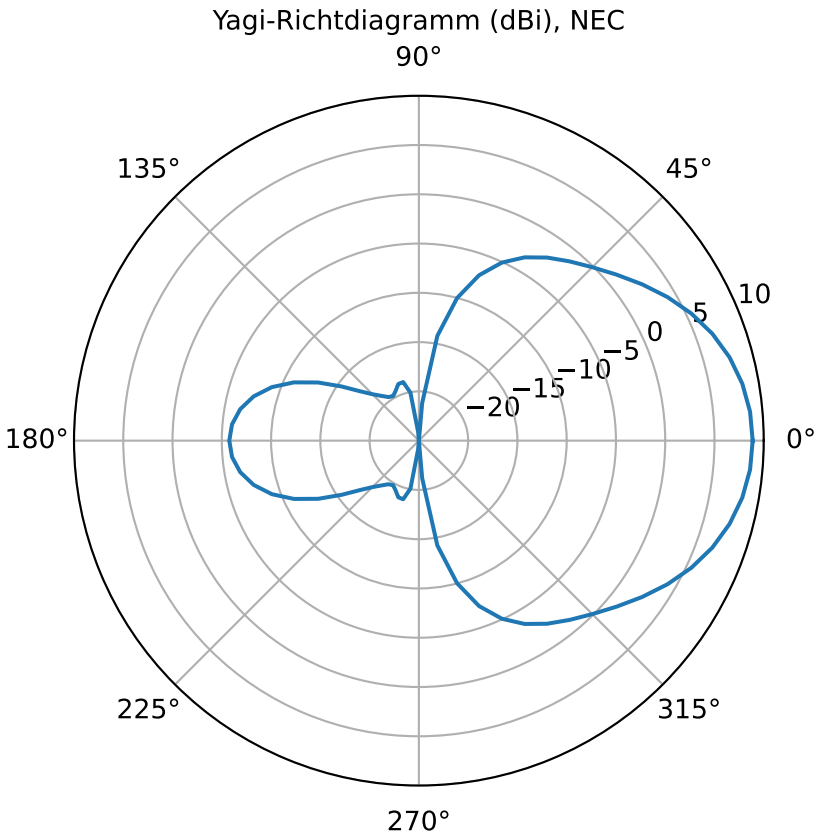


Abbildung 11.4: Das NEC-Richtdiagramm der 2,4-GHz-Yagi in der Horizontalebene (Gewinn in dBi, radiale Achse bei −25 dBi gedeckelt): eine kräftige Keule nach vorn (0°, knapp 9 dBi), ein kleiner Rückzipfel (180°, −6 dBi). Genau dieses Diagramm — mit einem zweiten, unabhängigen Verfahren bestätigt — machte den Entwurf baureif.

Akt 4 — das Querrechnen vor dem Bauen. Einem einzigen Programm glaubt man keinen Bauplan. Die Gegenrechnung übernahm **openEMS**

— wieder ein Zeitbereichs-Volumenverfahren wie Meep, aber mit einem Dünndraht-Zusatzmodell, das Drähte unterhalb der Gitterauflösung korrekt behandelt (es läuft in einer eigenen conda-Umgebung und braucht Minuten statt Millisekunden, deshalb zitieren wir hier sein Ergebnis): Gewinn ≈ 10 dBi, Vor/Rück ≈ 12 dB, Hauptkeule nach vorn. Momentenmethode und Volumen-FDTD — zwei Verfahren, die mathematisch nichts voneinander wissen — stimmen auf etwa ein Dezibel überein. *Erst damit* war der Entwurf baubar; die Antenne wurde anschließend tatsächlich aus Kupferstäben gebaut.

Die Moral, in drei Zeilen Checkliste: Das Werkzeug muss zur Geometrie passen (Verhältnisse zur Wellenlänge!). Ein vereinfachtes Modell kann qualitativ glänzen und quantitativ unbaubar sein. Und gebaut wird erst, wenn zwei unabhängige Verfahren dasselbe sagen — Querrechnen ist kein Misstrauen gegen Programme, sondern Respekt vor dem Lötkolben.

11.9 Das Kapitel-Programm

`programme/kap11/kap11_loeser_landschaft.py` bündelt die vier Messungen des Kapitels eigenständig und sichert sie mit `assert`-Schranken: Ringwellen-Querrechnung (naive Quellzeiten $> 3\,\%$, Leapfrog-Mitte $< 1\,\%$, Kammradien gleich), PML (Restfeld = Nachleuchten auf $10\,\%$, echte Reflexion $< 2 \cdot 10^{-6}$), DFT-Monitor (Verhältnis konstant auf 10^{-9}, Eichfaktor = $T/(2\sqrt{2\pi})$ auf 10^{-6}) und NEC-Yagi (Gewinn, Vor/Rück und Impedanz in engen Fenstern).

Merkkasten

- **Meep ist unser Buch in Bibliotheksform**: `resolution` = Punkte pro Wellenlänge (Kap. 8), Courant-Zahl $0{,}5$ (Kap. 8/9), Stromquelle = weiche Quelle (Kap. 6), `mp.PML` (Kap. 9, nur tausendfach besser: Reflexion $< 10^{-6}$), DFT-Monitor = Kap.-10-Detektor (gemessen: konstanter Eichfaktor, Rest 10^{-15}).
- **Meep-Einheiten:** $c = 1$, Bezugslänge a frei; $f_{\text{meep}} = a/\lambda$. Maxwell ist skaleninvariant — ein Lauf gilt für alle Maßstäbe, vom Radar bis zum Licht.
- **Querrechnen heißt zuhören:** Eine unerklärte Differenz ist eine Botschaft. Unsere 4,7 % entpuppten sich als Meeps Quellstrom am halben Zeitschritt — der Yee-Versatz, einmal mehr.
- **Was ein Löser diskretisiert, bestimmt, was er kann:** Volumen (FDTD/FEM) für Strukturen um λ, Drähte (MoM) für dünne Leiter, Strahlen für das elektrisch Riesige. Ein „bestes Programm für

alles” gibt es prinzipbedingt nicht.
- **Die Yagi-Moral:** Literatur vor dem Raten, Werkzeug zur Geometrie, und gebaut wird erst nach bestandener Querrechnung.

Roter Faden

Dieses Kapitel war die Ernte von Teil II: Jedes Bauteil, das wir in den Kapiteln 5–10 selbst geschraubt haben — Yee-Gitter und Leapfrog (**Kap. 5**), weiche Quellen (**Kap. 6**), Courant-Zahl und Punkte-pro-λ (**Kap. 8**), PML und Nachleuchten (**Kap. 9**), Spektral-Detektor und Querrechnen-Checkliste (**Kap. 10**) — hängt in Meep am vertrauten Haken, und wir haben jeden Haken einzeln nachgewogen. Sogar die Yagi-Warnung aus Kapitel 10 hat jetzt Anfang, Mitte und versöhnliches Ende. Nach vorn öffnet sich Teil IV: Ab **Kapitel 12** (Brechung und Totalreflexion) ist Meep eingearbeitetes Werkzeug statt Untersuchungsobjekt — und die Wahlregel aus Kapitel 4 („Farbe für räumliche Verteilung”) bekommt mit `get_array` ihren Dauerauftrag. **Kapitel 22** löst das 19-Ω-Versprechen der Yagi ein (Leitungen und Anpassung), und **Kapitel 24** baut sie — mit Stückliste.

Übungen

Ü 11.1 (Verstehen). Wähle für jedes Problem das Verfahren (FDTD, MoM, FEM oder Strahlverfahren) und begründe mit einer Zeile: (a) eine UKW-Drahtantenne (145 MHz, 2 mm Drahtdurchmesser) auf dem Hausdach; (b) die Antireflex-Beschichtung einer Solarzelle über das ganze Sonnenspektrum; (c) ein Mikrowellen-Keramikfilter im Metallgehäuse, das bei genau 5,8 GHz arbeiten soll; (d) das Radarecho eines Flugzeugs bei 10 GHz.

ℹ Musterlösung zu Ü 11.1

(a) MoM (NEC): dünner Draht (2 mm $\ll \lambda \approx$ 2 m) in viel Freiraum — das Bilderbuchproblem der Momentenmethode; ein Volumengitter müsste Millimeter auflösen und Meter umfassen. **(b) FDTD (Meep):** breitbandig (ein Puls, alle Frequenzen, Kapitel 10), dispersive dünne Schichten, Strukturgrößen um die Wellenlänge — Meeps Heimspiel. **(c) FEM:** eine einzige Arbeitsfrequenz, komplexe 3D-Geometrie aus Dielektrikum und Metall, gesucht ist eine scharfe Resonanz — das ist der Frequenzbereichs-Fall. (openEMS/FDTD ginge auch, müsste aber

das lange Ausklingen des Resonators aussitzen.) **(d) Strahlverfahren (PO/UTD):** das Flugzeug misst hunderte Wellenlängen ($\lambda = 3$ cm) — jedes Gitter wäre astronomisch; asymptotische Verfahren sind hier nicht Notlösung, sondern die einzige Lösung.

Ü 11.2 (Verstehen). Eine WLAN-Simulation bei $f = 2{,}45$ GHz soll in Meep laufen; als Bezugslänge wird $a = 1$ cm gewählt. Berechne von Hand (und prüfe mit drei Zeilen Code): Welche Meep-Frequenz f_{meep} gehört dazu? Und welche `resolution` brauchst du mindestens, damit das Gitter 20 Punkte pro Wellenlänge hat (Faustregel aus Kapitel 8)?

Musterlösung zu Ü 11.2

Die Wellenlänge ist $\lambda = c/f = 0{,}1224$ m $= 12{,}24$ cm $= 12{,}24\,a$. Also $f_{\text{meep}} = a/\lambda = 1/12{,}24 = 0{,}0817$. Die Auflösung zählt Punkte pro Bezugslänge; 20 Punkte pro Wellenlänge verlangen resolution $= 20/\lambda_{\text{meep}} = 20/12{,}24 \approx 1{,}6$, aufgerundet also `resolution=2` (mehr schadet nie — Kapitel 8). Dass hier eine *Zwei* reicht, wo unser Ringexperiment 50 brauchte, liegt nur an der Wahl von a: Die Wellenlänge ist zwölf Bezugslängen groß.

```
# von oben: c
a = 0.01
lam = c / 2.45e9                    # Wellenlänge in m
f_meep = a / lam
print(f"λ = {lam / a:.2f}·a  →  f_meep = {f_meep:.4f}")
print(f"resolution ≥ {20 * f_meep:.2f}  → gewählt: 2")
```

```
λ = 12.24·a  →  f_meep = 0.0817
resolution ≥ 1.63  → gewählt: 2
```

Ü 11.3 (Verändern). Stelle der Ringwelle ein Hindernis in den Weg: Fülle mit `geometry` die rechte Zellenhälfte mit Glas ($\varepsilon = 4$, also Brechungsindex $n = 2$ — Kapitel 4). Miss im Schnappschuss bei $t = 2{,}6$ den Kammradius links (Vakuum) und rechts (Glas). Sage vorher: Welches Verhältnis erwartest du?

Musterlösung zu Ü 11.3

Im Glas läuft die Welle mit $c/n = c/2$ — der rechte Kamm sollte also bei der **halben** Strecke stehen, Verhältnis 2. Die Messung:

```
# von oben: meep_lauf(), GROESSE, NX, IQ, DX, x_achse, XQ
# mp.Block: ein Quader; mp.Medium: ein Material (hier ε = 4)
glas = mp.Block(size=mp.Vector3(GROESSE / 2, GROESSE, mp.inf),
                center=mp.Vector3(GROESSE / 4, 0),
                material=mp.Medium(epsilon=4))
ez_glas = meep_lauf(GROESSE, pml=True, geometrie=[glas], bis=2.6)

schnitt = ez_glas[:, IQ]                        # Schnitt entlang x durch die Quelle
nach_rechts = schnitt[IQ + 30:]                 # Kammsuche jenseits der Quelle
nach_links = schnitt[:IQ - 29][::-1]
r_glas = (np.argmax(np.abs(nach_rechts)) + 30) * DX
r_vakuum = (np.argmax(np.abs(nach_links)) + 30) * DX
print(f"Kammradius Vakuum {r_vakuum:.2f}, Glas {r_glas:.2f}"
      f"  →  Verhältnis {r_vakuum / r_glas:.2f}   (Theorie: n = 2)")
```

```
Kammradius Vakuum 2.06, Glas 1.02  →  Verhältnis 2.02   (Theorie: n = 2)
```

Gemessen: $2{,}0$ — und ganz nebenbei steckt im Glas-Lauf schon das nächste Kapitel: An der Grenzfläche wird ein Teil des Rings reflektiert, und die Wellenlänge im Glas ist halbiert. Genau dort machen wir in Kapitel 12 weiter.

Ü 11.4 (Übertragen). Meep lässt dich die Courant-Zahl verstellen. Setze sie auf $0{,}8$ — das liegt über der 2D-Grenze $1/\sqrt{2} \approx 0{,}707$ aus Kapitel 9. Sage vorher, was passieren wird *und wie das explodierte Feld aussehen muss*; prüfe beides nach (die Nachbar-Korrelation aus Kapitel 8/9 überführt den Täter).

ℹ Musterlösung zu Ü 11.4

Vorhersage: Instabilität — und zwar getragen vom **Schachbrett**, der Zickzack-Welle in beiden Richtungen zugleich (Kapitel 9); seine Nachbar-Korrelation muss in x **und** y nahe -1 liegen.

```
# von oben: meep_lauf(), GROESSE, NX
ez_boom = meep_lauf(GROESSE, pml=False, courant=0.8, bis=3.0)
print(f"max|Ez| nach t = 3:  {np.abs(ez_boom).max():.1e}")

b = ez_boom[NX // 2 - 50:NX // 2 + 50, NX // 2 - 50:NX // 2 + 50]
korr_x = np.mean(b[1:, :] * b[:-1, :]) / np.mean(b ** 2)
korr_y = np.mean(b[:, 1:] * b[:, :-1]) / np.mean(b ** 2)
print(f"Nachbar-Korrelation: x {korr_x:+.3f}, y {korr_y:+.3f}")
```

```
max|Ez| nach t = 3:  1.7e+71
Nachbar-Korrelation: x -0.998, y -0.998
```

Siebzig Größenordnungen in drei Zeiteinheiten, Korrelation $-0{,}998$ in beiden Richtungen: dasselbe Schachbrett wie in unserem Eigenbau — natürlich, denn es ist dasselbe Verfahren. Profi-Software schützt nicht vor der CFL-Bedingung; sie wählt nur die Voreinstellung ($0{,}5$) so, dass man nicht aus Versehen darüber stolpert. (Wer mag, halbiere stattdessen `resolution` im Ringexperiment und beobachte die Dispersions-Schleppe aus Kapitel 8 — auch sie wandert selbstverständlich mit um.)

Das Kleingedruckte

Warum FEM-Löser Gleichungssysteme lösen — und Meep einfach marschiert. Tabelle 11.1 sortiert die Verfahren danach, *was* sie diskretisieren. Quer dazu verläuft die explizit/implizit-Trennlinie aus Kapitel 7, und dass sie verläuft, wie sie verläuft, hat einen benennbaren Grund. Finite Differenzen sind eine *Kollokations*-Methode: Die Gleichung soll **punktweise an den Gitterpunkten** gelten, vor der Unbekannten steht nichts — der Zeitschritt ist eine Auswertung, FDTD (und das verwandte FIT von openEMS) marschiert. FEM dagegen baut die Lösung aus **überlappenden Ansatzfunktionen** über den Netzelementen auf und verlangt die Gleichung nur *im gewichteten Mittel* (das Prinzip heißt Galerkin). Der Preis steht nach der Raum-Diskretisierung links vor der Zeitableitung: eine sogenannte **Massenmatrix** M — die Buchhaltung darüber, wie stark sich benachbarte Ansatzfunktionen überlappen —, sodass die halbdiskrete Gleichung $M\,\dot{u} = K\,u$ lautet. Weil M nicht diagonal ist, kostet schon der *explizit gemeinte* Zeitschritt pro Schritt die Lösung von $M\,x = b$: Das „alle Punkte gleichzeitig, Reihenfolge egal" aus Kapitel 7 ist konstruktionsbedingt verbaut. In der Praxis kommt ein zweiter Grund dazu: Die großen kommerziellen FEM-Feldlöser (HFSS, COMSOL) arbeiten meist gleich im **Frequenzbereich** — eine Frequenz, ein stationäres Problem, und das *ist* per Definition ein Gleichungssystem; dort gibt es gar keinen Zeitschritt, der explizit sein könnte. Und wer im Zeitbereich ohnehin pro Schritt löst, nimmt meist gleich das unbedingt stabile implizite Verfahren samt großem Δt mit — exakt die Kapitel-7-Abwägung.

Das Schlupfloch: explizites FEM gibt es doch. Zwingend ist das Gleichungssystem nämlich nicht. Beim **Mass Lumping** nähert man M durch eine Diagonalmatrix (geschickte Quadratur kann das sogar exakt einrichten) — dann ist M^{-1} geschenkt und der Schritt wieder eine Auswertung. Und die

elegante moderne Lösung heißt **Discontinuous Galerkin** (DGTD): Die Ansatzfunktionen leben strikt *pro Element*, ohne Stetigkeitszwang zum Nachbarn; gekoppelt wird über Flüsse an den Elementgrenzen, wie bei Finite-Volumen-Verfahren. Dadurch zerfällt M in kleine, unabhängige Blöcke je Element, die man einmal vorab invertiert — danach marschiert das Verfahren voll explizit, auf unstrukturierten Netzen und mit hoher Ordnung: FEM-Geometrieflexibilität mit FDTD-Marschverhalten. Der Merksatz: **Explizit geht überall dort, wo nach der Raum-Diskretisierung nichts Nichttriviales vor der Unbekannten steht** — finite Differenzen haben das gratis, FIT/finite Volumen auch, FEM muss es sich mit Lumping oder DG erarbeiten. Und umgekehrt gibt es implizites FDTD (ADI, Kapitel 7): Die Trennlinie explizit/implizit ist eine Eigenschaft der Matrizen, nicht der Verfahrensfamilie.

12 Brechung und Totalreflexion

Ein Strohhalm im Wasserglas sieht geknickt aus — das weiß jedes Kind, und kaum jemand findet es noch erstaunlich. Dabei versteckt sich gleich hinter dem Knick ein Effekt, der es verdient: Schaut man **aus dem Wasser heraus** flach genug nach oben, kommt das Licht gar nicht mehr hinaus. Es wird vollständig zurückgeworfen — von einer Grenzfläche, an der nichts ist außer Wasser und Luft. Ein Spiegel ohne Spiegel, und zwar ein perfekter: Gewöhnliche Metallspiegel schlucken ein paar Prozent pro Reflexion, die **Totalreflexion** schluckt nichts. Genau deshalb kann eine Glasfaser Licht über tausende Kilometer tragen — das Licht prallt millionenfach an der Faserwand ab und verliert dabei: nichts.

In Übung 11.3 haben wir das Werkzeug dafür schon in der Hand gehabt — ein Glasblock im Meep-Gitter, eine Welle, die hineinlief und langsamer wurde. In diesem Kapitel lassen wir die Welle **schräg** auf die Grenze treffen und vermessen alles, was dann passiert: den Knick (mit einer Messmethode, die das Brechungsgesetz gleich mitbeweist), den reflektierten Anteil (auf drei Nachkommastellen gegen die Theorie), den Grenzwinkel — und das Gespenst, das bei Totalreflexion jenseits der Grenze zurückbleibt.

Lernziele

Nach diesem Kapitel kannst du …

1. … das Brechungsgesetz aus einer einzigen Idee herleiten — die Wellenberge müssen an der Grenze Schritt halten — und es als Erhaltung der Raumfrequenz k_x im Feldbild nachmessen,
2. … den reflektierten Energieanteil einer Grenzfläche messen (Flux-Monitore plus Referenzlauf) und gegen die Fresnel-Formel halten,
3. … den Grenzwinkel berechnen und erklären, warum es ihn nur auf dem Weg vom dichten ins dünne Medium gibt,
4. … das evaneszente Feld hinter einer total reflektierenden Grenze messen (Eindringtiefe, „zu schnelles” k_x) und begründen, warum dort eine laufende Welle verboten ist,
5. … eine Energiebilanz so lange verfolgen, bis sie schließt — auch wenn die fehlende Energie seitwärts entwischt.

12.1 Wellenberge müssen Schritt halten

Bevor irgendein Programm läuft, die zentrale Überlegung — sie braucht nur ein Lineal. Eine ebene Welle läuft schräg im Glas nach oben, unter dem Winkel θ_1 zur Senkrechten auf die Grenzfläche (so werden Winkel in der Optik immer gemessen: zum **Lot**, nicht zur Fläche). Ihre Wellenberge sind parallele Geraden im Abstand $\lambda_1 = \lambda_0/n_1$ — im Glas ist die Welle ja langsamer und gestaucht (Kapitel 4: $n = \sqrt{\varepsilon_r}$, in Übung 11.3 gemessen). Wo ein Wellenberg die Grenzlinie trifft, hinterlässt er dort einen „Fußabdruck", und die Geometrie sagt sofort, in welchem Abstand diese Fußabdrücke aufeinanderfolgen:

$$d = \frac{\lambda_1}{\sin\theta_1}.$$

(Je steiler die Welle einfällt, desto weiter rücken die Fußabdrücke auseinander; bei senkrechtem Einfall, $\theta_1 = 0$, trifft ein Berg die ganze Grenze gleichzeitig — d wird unendlich.)

Oberhalb der Grenze muss die durchgelassene Welle **dieselben Fußabdrücke** bedienen: Die Felder direkt unter und direkt über der Grenze hängen über die Maxwell-Gleichungen zusammen, ein Wellenberg unten erzwingt einen Wellenberg oben, im Takt, Punkt für Punkt. Also gilt mit den Größen des oberen Mediums

$$\frac{\lambda_2}{\sin\theta_2} = \frac{\lambda_1}{\sin\theta_1},$$

und wenn man $\lambda = \lambda_0/n$ einsetzt und umstellt, steht das **Brechungsgesetz von Snellius** da:

$$n_1 \sin\theta_1 = n_2 \sin\theta_2.$$

Die Handrechnung am konkreten Zahlenpunkt, die uns gleich als Messvorhersage dient: Glas ($n_1 = 1{,}5$), Einfall $\theta_1 = 30°$, oben Vakuum ($n_2 = 1$). Dann ist $\sin\theta_2 = 1{,}5 \cdot \sin 30° = 1{,}5 \cdot 0{,}5 = 0{,}75$, also $\theta_2 = \arcsin(0{,}75) = 48{,}59°$ — der Strahl knickt beim Übergang ins dünnere Medium **vom Lot weg**, von $30°$ auf fast $49°$.

Für die Messung lohnt ein zweiter Blick auf die Fußabdrücke: Ihr Kehrwert $k_x = 1/d = \sin\theta/\lambda$ ist eine **Raumfrequenz** — wie viele Wellenberge pro Längeneinheit die Grenze entlangwandern (der Ergebnistyp: eine Zahl, Einheit $1/\lambda_0$). Das Schritthalten heißt dann schlicht: k_x **ist auf beiden Seiten**

der Grenze gleich. In dieser Form ist Snellius direkt messbar, denn Raumfrequenzen können wir seit Kapitel 8 bestimmen — mit der Nulldurchgangs-Methode, nur diesmal entlang einer Linie *parallel zur Grenze*.

12.2 Der Versuchsaufbau: ein Scheinwerfer im Glas

Die Frage: Knickt die Welle wirklich um die vorhergesagten $48{,}59°$ — und ist k_x beidseits der Grenze gleich? **Die Bühne:** eine Zelle von 24×16 Meep-Einheiten; wir wählen die Frequenz $f_{\text{meep}} = 1$, sodass eine Längeneinheit genau eine Vakuum-Wellenlänge ist (Kapitel 11) — Auflösung 24 Punkte pro λ_0. Die untere Halbebene ($y < 0$) ist Glas ($n = 1{,}5$), oben Vakuum, ringsum PML. **Die Quelle:** Ein einzelner Lichtstrahl statt einer raumfüllenden Welle — dafür hat Meep die `GaussianBeamSource`: eine Quellfläche, auf der Meep die Felder eines **Gaußstrahls** vorschreibt, also eines Wellenbündels mit glockenförmigem Querschnittsprofil, das eine Richtung (`beam_kdir`) und eine Taillenbreite (`beam_w0`) hat. Wir stellen den Scheinwerfer ins Glas (Quellfläche bei $y = -4$) und zielen so, dass der Strahl die Grenze immer im Ursprung $(0, 0)$ trifft — dazu muss die Quelle um $4 \tan \theta_1$ seitlich versetzt stehen, das übernimmt eine Formel. Die Taille $w_0 = 3\lambda$ ist ein Kompromiss, den wir später noch bezahlen werden: schmal genug, dass man im Bild einen Strahl *sieht*, breit genug, dass das Bündel winkelscharf bleibt (mehr dazu bei der Energiebilanz). **Anfangsbedingung:** Felder null; die Quelle schwingt als Dauerstrich (`ContinuousSource`), bis das Bild eingeschwungen ist ($t = 50$). **Gemessen wird** der Ez-Schnappschuss; die Auswertung holt k_x aus je einer Gitterzeile knapp unter ($y = -0{,}5$) und knapp über ($y = +0{,}5$) der Grenze. **Erfolgskriterium:** beide k_x gleich $0{,}75$ (auf rund ein Prozent), Brechungswinkel $48{,}6°$ (auf ein halbes Grad).

```
import os
# vor dem Meep-Import: harmlose MPI-Hardware-Warnungen unterdrücken (Kap. 11)
os.environ["HWLOC_HIDE_ERRORS"] = "2"

import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)

N_GLAS = 1.5
AUFL = 24              # Punkte pro Längeneinheit = pro λ0 (f_meep = 1)
SX, SY = 24.0, 16.0
DX = 1.0 / AUFL
```

```
F = 1.0
W0 = 3.0                # Strahltaille (in λ0)
Y_QUELLE = -4.0


def simulation(theta_grad, zeitverlauf, mit_grenze=True, von_oben=False):
    """Gaußstrahl, der die Grenze (y = 0) immer bei (0,0) trifft."""
    th = np.radians(theta_grad)
    y_q = -Y_QUELLE if von_oben else Y_QUELLE
    kdir = mp.Vector3(np.sin(th), -np.cos(th) if von_oben else np.cos(th))
    if mit_grenze:
        geometrie = [mp.Block(size=mp.Vector3(mp.inf, SY),
                              center=mp.Vector3(0, -SY / 2),
                              # mp.Medium(index=...): Material über den
                              # Brechungsindex statt über ε angeben
                              material=mp.Medium(index=N_GLAS))]
    else:                 # Referenzlauf: dasselbe Glas, aber ohne Grenze
        geometrie = [mp.Block(size=mp.Vector3(mp.inf, mp.inf, mp.inf),
                              material=mp.Medium(index=N_GLAS))]
    quelle = mp.GaussianBeamSource(   # der "Scheinwerfer"
        zeitverlauf,
        center=mp.Vector3(-abs(y_q) * np.tan(th), y_q),  # Ziel: (0,0)
        size=mp.Vector3(10, 0),       # Quellfläche, auf der Meep den
        component=mp.Ez,              #   Strahl vorschreibt
        beam_kdir=kdir,               # Strahlrichtung
        beam_w0=W0,                   # Taillenbreite
        beam_x0=mp.Vector3(0, 0),     # Ort der Taille: der Auftreffpunkt
        beam_E0=mp.Vector3(0, 0, 1))  # E zeigt aus der Ebene (TMz, Kap. 9)
    return mp.Simulation(cell_size=mp.Vector3(SX, SY), resolution=AUFL,
                         boundary_layers=[mp.PML(1.0)], sources=[quelle],
                         geometry=geometrie)


def feldbild(theta_grad, von_oben=False):
    """Dauerstrich einschwingen lassen, Ez-Schnappschuss zurückgeben."""
    # mp.ContinuousSource: Dauerstrich-Sinus (width: sanfte Anlaufzeit)
    sim = simulation(theta_grad, mp.ContinuousSource(F, width=2),
                     von_oben=von_oben)
    sim.run(until=50.0)
    return sim.get_array(component=mp.Ez, center=mp.Vector3(),
                         size=mp.Vector3(SX, SY))
```

```
ez30 = feldbild(30.0)
print(f"eingeschwungen; Feld {ez30.shape},"
      f" Grenzwinkel-Vorgriff: arcsin(1/1,5) = "
      f"{np.degrees(np.arcsin(1 / N_GLAS)):.1f}°")
```

```
Using MPI version 4.1, 1 processes
eingeschwungen; Feld (576, 384), Grenzwinkel-Vorgriff: arcsin(1/1,5) = 41.8°
```

Vorhersage-Punkt

Der Strahl kommt mit 30° zum Lot von unten. Skizziere, bevor du weiterblätterst: In welche Richtung knickt er beim Übergang ins Vakuum — zum Lot hin oder vom Lot weg? Und wird außer dem geknickten Strahl noch etwas anderes im Bild sein?

Die Antwort steckt in der Handrechnung von oben — $\sin\theta_2$ ist *größer* als $\sin\theta_1$, der Strahl flieht das Lot. Und weil schon der senkrechte Übergang Glas→Luft in Kapitel 6 ein 4-%-Spiegelbild lieferte, sollte auch hier ein reflektierter Strahl zurücklaufen. Das Bild, mit allen drei Erwartungen als Linien eingezeichnet:

```
# von oben: ez30, SX, SY, N_GLAS
fig, ax = plt.subplots(figsize=(7.0, 4.6))
w = np.abs(ez30).max() / 2
ax.imshow(ez30.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w,
          extent=(-SX / 2, SX / 2, -SY / 2, SY / 2))
ax.axhline(0, color="black", lw=1.2)

th1 = np.radians(30.0)
th2 = np.arcsin(N_GLAS * np.sin(th1))          # Snellius-Erwartung
ax.plot([-7 * np.tan(th1), 0], [-7, 0], "k--", lw=1.0)   # einfallend
ax.plot([0, 7 * np.tan(th1)], [0, -7], "k--", lw=1.0)    # reflektiert
ax.plot([0, 7.5 * np.tan(th2)], [0, 7.5], "k--", lw=1.0) # gebrochen
ax.text(-9.5, -7.3, "einfallend, 30°"); ax.text(3.2, -7.3, "reflektiert")
ax.text(5.0, 5.6, "gebrochen, 48,6°")
ax.text(-11.5, 6.8, "Vakuum n = 1"); ax.text(-11.5, -5.4, "Glas n = 1,5")
ax.set_xlabel("x (λ)"); ax.set_ylabel("y (λ)")
plt.tight_layout(); plt.show()
```

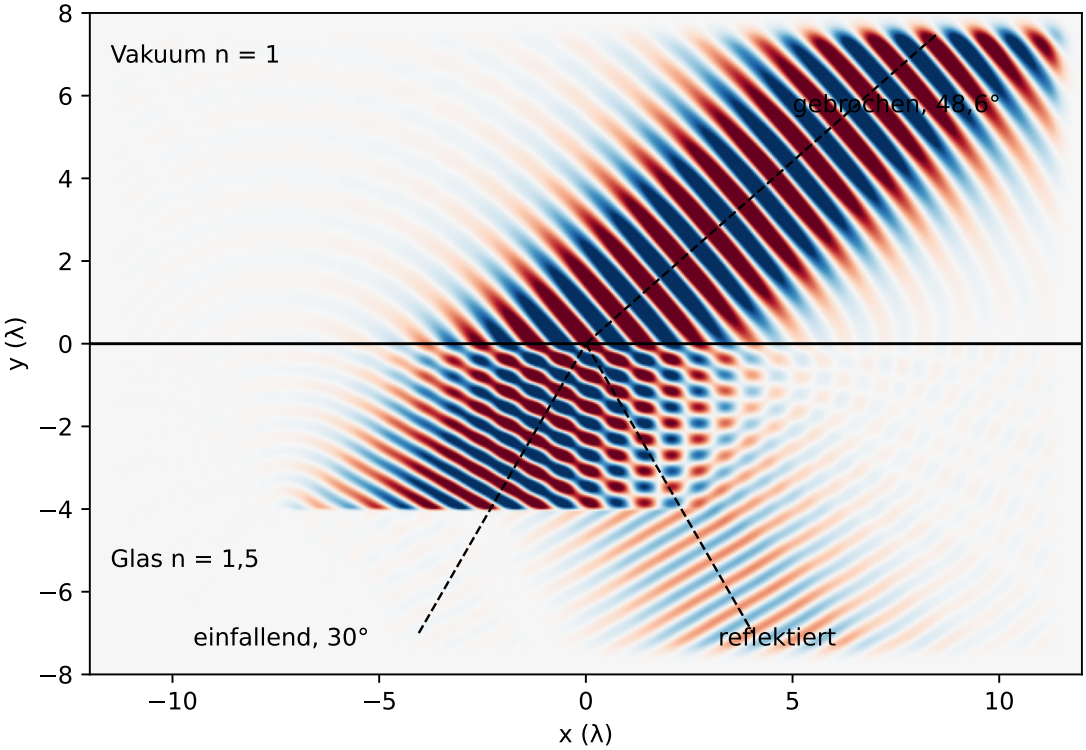


Abbildung 12.1: Der Gaußstrahl trifft mit 30° von unten (Glas, n = 1,5) auf die Grenze zum Vakuum (schwarze Linie). Farbe = Ez; die gestrichelten Geraden sind die Erwartung: einfallender Strahl (30°), reflektierter Strahl (30° zur anderen Seite) und der gebrochene Strahl unter arcsin(1,5·sin 30°) = 48,6° — die Welle folgt allen dreien. Im Glas ist die Wellenlänge sichtbar kürzer (n = 1,5); das Streifenmuster links unten ist die Interferenz aus einfallendem und reflektiertem Strahl.

Alle drei Strahlen liegen auf ihren Erwartungslinien. Aber „liegt drauf" ist ein Urteil per Augenmaß, und seit Kapitel 10 geben wir uns damit nicht mehr zufrieden. Die Messung: Wir legen je eine Messlinie parallel zur Grenze — eine Gitterzeile im Glas ($y = -0{,}5$), eine im Vakuum ($y = +0{,}5$) — und bestimmen auf beiden die Raumfrequenz k_x über die Nulldurchgänge, wörtlich mit der Methode aus Kapitel 8:

```
# von oben: ez30, SX, SY, AUFL, DX
def idx(x, y):
    """Gitterindizes zum Ort (x, y)."""
    return int(round((x + SX / 2) * AUFL)), int(round((y + SY / 2) * AUFL))
```

```python
def kx_aus_zeile(ez, y, x0=-4.0, x1=4.0):
    """Raumfrequenz entlang x auf Höhe y (Nulldurchgänge, Kapitel 8)."""
    i0, _ = idx(x0, 0); i1, _ = idx(x1, 0); _, j = idx(0, y)
    zeile = ez[i0:i1, j]
    s = np.flatnonzero(np.sign(zeile[:-1]) * np.sign(zeile[1:]) < 0)
    null = s - zeile[s] / (zeile[s + 1] - zeile[s])   # interpolierte Nullen
    return 1.0 / (2 * np.mean(np.diff(null)) * DX)    # kx = 1 / λx


kx_glas = kx_aus_zeile(ez30, -0.5)
kx_vakuum = kx_aus_zeile(ez30, +0.5)
print(f"kx im Glas  : {kx_glas:.4f}  (Theorie n·sin30° = {N_GLAS * 0.5:.4f})")
print(f"kx im Vakuum: {kx_vakuum:.4f}")
print(f"θ₂ = arcsin(kx·λ0) = {np.degrees(np.arcsin(kx_vakuum)):.2f}°"
      f"   (Snellius: {np.degrees(np.arcsin(N_GLAS * 0.5)):.2f}°)")
```

```
kx im Glas  : 0.7570  (Theorie n·sin30° = 0.7500)
kx im Vakuum: 0.7504
θ₂ = arcsin(kx·λ0) = 48.63°   (Snellius: 48.59°)
```

Beide Seiten tragen dasselbe $k_x = 0{,}75$ — die Fußabdrücke halten Schritt, und weil im Vakuum $|k| = 1/\lambda_0 = 1$ fest steht, folgt der Brechungswinkel aus $\sin\theta_2 = k_x\lambda_0$ allein: $48{,}6°$, auf vier Hundertstel Grad an der Handrechnung. Merke die Logik: Wir haben **keinen Winkel gemessen**, sondern eine Raumfrequenz — Snellius ist kein Gesetz über Strahlen, sondern über Wellenberge, die an einer Linie Schritt halten. Die Strahlen-Form ist nur seine Schatten-Geometrie.

12.3 Wie viel kommt durch? Flux und Fresnel

Das Feldbild zeigt einen reflektierten Strahl — wie viel Energie trägt er? Für den senkrechten Einfall kennen wir die Antwort aus Kapitel 6: Der Spiegelfaktor war $r = (Z_2 - Z_1)/(Z_2 + Z_1) = 0{,}2$, also $4\,\%$ der Energie. Die Verallgemeinerung auf schrägen Einfall heißt **Fresnel-Formel**; für unsere Polarisation (E zeigt aus der Ebene, „senkrecht" oder *s*-polarisiert) lautet sie

$$r_s = \frac{n_1\cos\theta_1 - n_2\cos\theta_2}{n_1\cos\theta_1 + n_2\cos\theta_2}, \qquad R = r_s^2.$$

Ihre Herleitung verschieben wir auf Kapitel 13 — dort wird sie zusammen mit ihrer Schwester für die zweite Polarisation gebraucht (die eine echte Überraschung bereithält). Hier prüfen wir sie erst einmal an zwei Punkten. Der Anker: Bei $\theta_1 = 0$ sind beide Kosinus 1, und es bleibt $(n_1 - n_2)/(n_1 + n_2) = 0{,}5/2{,}5 = 0{,}2$ — exakt der Kapitel-6-Wert, die Formel kennt ihren Spezialfall. Die Handrechnung für 30°: $n_1 \cos\theta_1 = 1{,}5 \cdot 0{,}866 = 1{,}299$ und $n_2 \cos\theta_2 = \cos 48{,}59° = 0{,}661$, also $r_s = 0{,}638/1{,}960 = 0{,}325$ und $R = 0{,}325^2 = 0{,}106$ — schräg verspiegelt Glas zweieinhalbmal stärker als senkrecht.

Gemessen wird mit dem Werkzeug aus Kapitel 11: **Flux-Monitore** (der DFT-Detektor in seiner Energiestrom-Form — er sammelt $E \times H$ durch eine Linie, frequenzaufgelöst) und die **Referenzlauf-Methode**. Der Aufbau: ein Monitor unterhalb der Grenze ($y = -2$), einer oberhalb ($y = +2$), jeweils 20 Einheiten breit. Der Referenzlauf füllt die ganze Zelle mit Glas — kein Übergang, der Monitor unten sieht die volle einfallende Leistung P_{ein}. Im Hauptlauf läuft durch den unteren Monitor einfallende *minus* reflektierte Leistung, also ist $R = (P_{\text{ein}} - P_{\text{unten}})/P_{\text{ein}}$; oben kommt $T = P_{\text{oben}}/P_{\text{ein}}$ heraus. Eine Feinheit noch: Für saubere Energiezahlen nehmen wir statt des Dauerstrichs einen **schmalbandigen Puls** (`GaussianSource`) und lassen laufen, bis alle Felder durch die Monitore hindurch sind — dann ist die Flux-Summe vollständig, ohne Einschwing-Schmutz. **Erfolgskriterium:** $R = 0{,}106$ auf ein Prozentpunkt-Niveau, und die Bilanz $R + T = 1$ (Checkliste, Zeile „Erhaltung").

```
# von oben: simulation(), F, N_GLAS
def fluss_bilanz(theta_grad, mit_seite=False):
    """R und T (und ggf. seitlicher Abfluss) per Puls + Referenzlauf."""
    werte = {}
    for mit_grenze in (False, True):
        # mp.GaussianSource: Puls um f = F, schmalbandig (fwidth klein)
        sim = simulation(theta_grad, mp.GaussianSource(F, fwidth=F / 8),
                         mit_grenze=mit_grenze)
        # add_flux(f, df, n, Region): Energiestrom-Monitor bei Frequenz f;
        # mp.FluxRegion: die Messlinie (Vorzeichen: Fluss in +Richtung)
        unten = sim.add_flux(F, 0, 1, mp.FluxRegion(
            center=mp.Vector3(0, -2), size=mp.Vector3(20, 0)))
        oben = sim.add_flux(F, 0, 1, mp.FluxRegion(
            center=mp.Vector3(0, +2), size=mp.Vector3(20, 0)))
        seite = sim.add_flux(F, 0, 1, mp.FluxRegion(
            center=mp.Vector3(10, 0), size=mp.Vector3(0, 4),
            direction=mp.X))              # Messlinie senkrecht: Fluss in +x
        # until_after_sources: weiterlaufen, bis n Zeiteinheiten NACH
        # Quellende vergangen sind — alle Pulse sind dann durch
```

```
        sim.run(until_after_sources=80)
        werte[mit_grenze] = (mp.get_fluxes(unten)[0],   # die Messwerte
                             mp.get_fluxes(oben)[0],
                             mp.get_fluxes(seite)[0])
    p_ein = werte[False][0]
    r = (p_ein - werte[True][0]) / p_ein
    t = werte[True][1] / p_ein
    s = werte[True][2] / p_ein
    return (r, t, s) if mit_seite else (r, t)


R30, T30 = fluss_bilanz(30.0)
print(f"30°:  R = {R30:.4f}   (Fresnel-s: 0,1058)")
print(f"      T = {T30:.4f}")
print(f"      R + T = {R30 + T30:.4f}")
```

```
30°:  R = 0.1061   (Fresnel-s: 0,1058)
      T = 0.8936
      R + T = 0.9997
```

Treffer auf drei Nachkommastellen, und die Energiebilanz schließt auf ein Promille. Damit ist das Brechungs-Inventar komplett: Richtung (Snellius, gemessen), Anteile (Fresnel, gemessen), Erhaltung (geprüft). Jetzt drehen wir den Winkel auf.

12.4 Der Grenzwinkel: wenn Snellius nicht mehr kann

Snellius verlangt $\sin\theta_2 = 1{,}5\,\sin\theta_1$ — und $\sin\theta_2$ kann nicht größer als 1 werden. Die Gleichung hat nur eine Lösung, solange

$$\sin\theta_1 \;\le\; \frac{n_2}{n_1} = \frac{1}{1{,}5} \qquad\Longrightarrow\qquad \theta_1 \;\le\; \theta_c = \arcsin\frac{1}{1{,}5} = 41{,}81^\circ.$$

Bei $41{,}8^\circ$ Einfall verlässt der gebrochene Strahl die Grenze streifend ($\theta_2 = 90^\circ$); darüber gibt es **keinen** Brechungswinkel mehr. Schon die Richtung des Arguments zeigt: Das kann nur passieren, wenn $n_1 > n_2$ ist — vom dichten ins dünne Medium. Umgekehrt, von dünn nach dicht, ist $\sin\theta_2 = \sin\theta_1/1{,}5$ immer kleiner als 1; von oben kommt man *immer* ins Glas hinein (Ü 12.4 prüft das nach). Und dieselbe Arcussinus-Sackgasse ist uns schon einmal begegnet: In Kapitel 8 hatte die Gitter-Dispersionsrelation unterhalb

von drei Punkten pro Wellenlänge keine reelle Lösung mehr — was die Mathematik dort verweigerte, war ebenfalls eine laufende Welle. Was stattdessen passiert, sehen wir gleich; erst die Vorhersage.

Vorhersage-Punkt

Wir fahren den Scheinwerfer von $30°$ auf $55°$ — deutlich über den Grenzwinkel. Was zeigt das Feldbild oberhalb der Grenze: exakt nichts, ein Stück gebrochener Strahl, das an der Grenze „klebt", oder etwas Drittes? Und was erwartest du für den reflektierten Strahl im Glas?

Der Lauf ist eine Zeile — gleicher Aufbau, anderer Winkel:

```
# von oben: feldbild(), SX, SY
ez55 = feldbild(55.0)

fig, ax = plt.subplots(figsize=(7.0, 4.6))
w = np.abs(ez55).max() / 2
ax.imshow(ez55.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w,
          extent=(-SX / 2, SX / 2, -SY / 2, SY / 2))
ax.axhline(0, color="black", lw=1.2)
th = np.radians(55.0)
ax.plot([-7 * np.tan(th), 0], [-7, 0], "k--", lw=1.0)
ax.plot([0, 7 * np.tan(th)], [0, -7], "k--", lw=1.0)
ax.text(-10.5, -7.3, "einfallend, 55°"); ax.text(6.0, -7.3, "reflektiert")
ax.text(-11.5, 6.8, "Vakuum: kein Strahl")
ax.set_xlabel("x (λ)"); ax.set_ylabel("y (λ)")
plt.tight_layout(); plt.show()
```

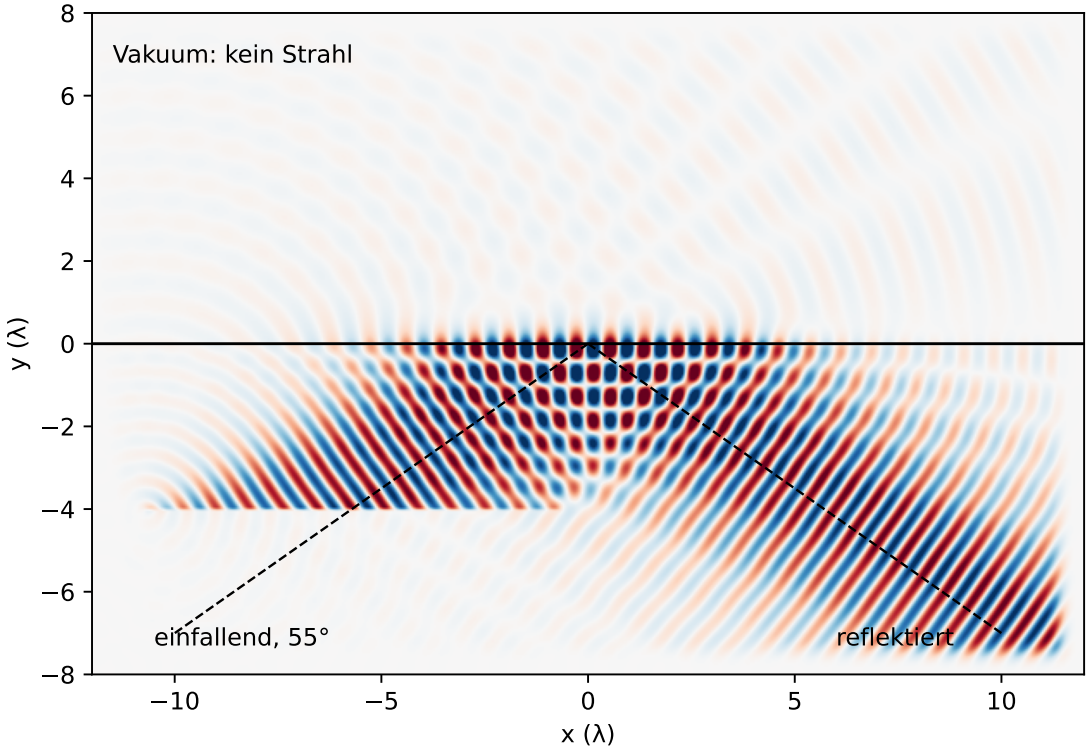


Abbildung 12.2: Derselbe Aufbau bei 55° Einfall, deutlich über dem Grenzwinkel von 41,8° (gestrichelt: einfallende und reflektierte Erwartungslinie). Oben existiert kein gebrochener Strahl mehr — der Strahl wird vollständig zurückgeworfen. Das kräftige Streifenmuster im Glas ist die Interferenz aus einfallendem und reflektiertem Strahl (stehende Welle quer zur Grenze, Kapitel 6); direkt über der Grenze bleibt ein hauchdünner heller Saum zurück, dem wir gleich nachgehen.

12.5 Das Gespenst über der Grenze

⚠ Naheliegende Vermutung

„Totalreflexion heißt total: Jenseits der Grenze ist das Feld exakt null."
Warum sie naheliegt: Der Name sagt es doch — *alles* wird reflektiert, die Energiebilanz wird gleich $R \approx 100\,\%$ bestätigen. Wo nichts hinfließt,

sollte nichts sein.
Was stattdessen stimmt: Direkt über der Grenze lebt ein schmaler, exponentiell abklingender Feldsaum — das **evaneszente Feld**. Es ist kein Rechenartefakt, sondern messbare Physik mit einer messbaren Eindringtiefe (gleich: knapp ein Viertel der Wellenlänge), es wandert mit dem Strahl die Grenze entlang und transportiert dabei sogar Energie *seitwärts* — nur eben nicht *hinaus*. Die Felder können an der Grenze gar nicht schlagartig auf null springen (die Maxwell-Gleichungen verknüpfen beide Seiten); sie können oberhalb nur möglichst schnell *verschwinden lernen*. In Kapitel 17 wird dieser Saum zur Hauptfigur: Hält man ein zweites Glas dicht genug daran, springt das Licht hindurch — es tunnelt.

Warum *kann* dort keine laufende Welle sein? Das Schritthalte- Argument beantwortet auch das — und jetzt zahlt sich die k_x-Lesart von Snellius aus. Die Fußabdrücke des $55°$-Strahls wandern mit $k_x = n_1 \sin 55°/\lambda_0 = 1{,}23/\lambda_0$ die Grenze entlang. Eine freie Vakuumwelle hat aber insgesamt nur $|k| = 1/\lambda_0$, und ihre Komponenten müssen $k_x^2 + k_y^2 = k_0^2$ erfüllen — schon die x-Komponente allein wäre hier größer als der ganze erlaubte Betrag! Für $k_y^2 = k_0^2 - k_x^2 < 0$ gibt es keine reelle Wellenzahl nach oben; der Ausweg der Physik ist ein Feld, das in y nicht schwingt, sondern **exponentiell abklingt**, $E \propto e^{-\kappa y}$, mit (in Kreiszahl-Schreibweise, daher das 2π)

$$\kappa = \frac{2\pi}{\lambda_0}\sqrt{(n_1 \sin\theta_1)^2 - n_2^2} \overset{55°}{=} 4{,}49/\lambda_0, \qquad \delta = \frac{1}{\kappa} = 0{,}22\,\lambda_0.$$

Beides ist nachprüfbar, und zwar im Feld, das wir schon haben. Erst das „zu schnelle" Muster: k_x auf einer Linie knapp über der Grenze —

```
# von oben: ez55, kx_aus_zeile(), N_GLAS
kx_saum = kx_aus_zeile(ez55, +0.15)
print(f"kx im Saum: {kx_saum:.3f}"
      f"   (n·sin55° = {N_GLAS * np.sin(np.radians(55)):.3f};"
      f" freie Vakuumwelle: maximal 1,000)")
```

```
kx im Saum: 1.215   (n·sin55° = 1.229; freie Vakuumwelle: maximal 1,000)
```

Da steht ein Muster mit $k_x = 1{,}2$ im Vakuum — schneller, als jede freie Welle dort laufen dürfte. Es kann nur als gebundener Saum existieren. Jetzt sein Abklingprofil: Wir mitteln E_z^2 entlang x (über den Auftreffbereich) für jede Gitterhöhe y über der Grenze und tragen den Pegel logarithmisch auf — ein Exponentialabfall muss dann eine Gerade mit Steigung $-\kappa$ sein, und die Theoriegerade zeichnen wir dazu:

```
# von oben: ez55, idx(), DX, AUFL, SX, SY
i0, _ = idx(-3, 0); i1, _ = idx(3, 0); _, j0 = idx(0, 0)
hoehen, pegel = [], []
for j in range(j0 + 1, j0 + int(0.6 * AUFL)):
    hoehen.append((j + 0.5) * DX - SY / 2)
    pegel.append(np.sqrt(np.mean(ez55[i0:i1, j] ** 2)))
hoehen = np.array(hoehen); pegel = np.array(pegel)

fit = (hoehen > 0.05) & (hoehen < 0.42)
kappa = -np.polyfit(hoehen[fit], np.log(pegel[fit]), 1)[0]
kappa_theorie = 2 * np.pi * np.sqrt((N_GLAS * np.sin(np.radians(55)))**2 - 1)
print(f"κ gemessen: {kappa:.2f}   Theorie: {kappa_theorie:.2f}"
      f"   → δ = {1 / kappa:.3f} λ (Theorie {1 / kappa_theorie:.3f} λ)")

fig, achsen = plt.subplots(1, 2, figsize=(7.4, 3.2))
ax = achsen[0]
a0, _ = idx(-4, 0); a1, _ = idx(8, 0); _, b0 = idx(0, -2); _, b1 = idx(0, 2)
w = np.abs(ez55).max() / 6
ax.imshow(ez55[a0:a1, b0:b1].T, origin="lower", cmap="RdBu_r",
          vmin=-w, vmax=w, extent=(-4, 8, -2, 2))
ax.axhline(0, color="black", lw=1.0)
ax.set_xlabel("x (λ)"); ax.set_ylabel("y (λ)")
ax.set_title("Zoom auf die Grenze", fontsize=10)

ax = achsen[1]
ax.semilogy(hoehen, pegel / pegel[0], "o", ms=4, label="Messung")
ax.semilogy(hoehen, np.exp(-kappa_theorie * (hoehen - hoehen[0])), "k--",
            lw=1.0, label="Theorie $e^{-\\kappa y}$")
ax.set_xlabel("Höhe y über der Grenze (λ)")
ax.set_ylabel("Feldpegel (normiert)")
ax.set_title("Abklingen des Saums", fontsize=10)
ax.legend(fontsize=8)
plt.tight_layout(); plt.show()
```

```
κ gemessen: 4.30   Theorie: 4.49   → δ = 0.233 λ (Theorie 0.223 λ)
```

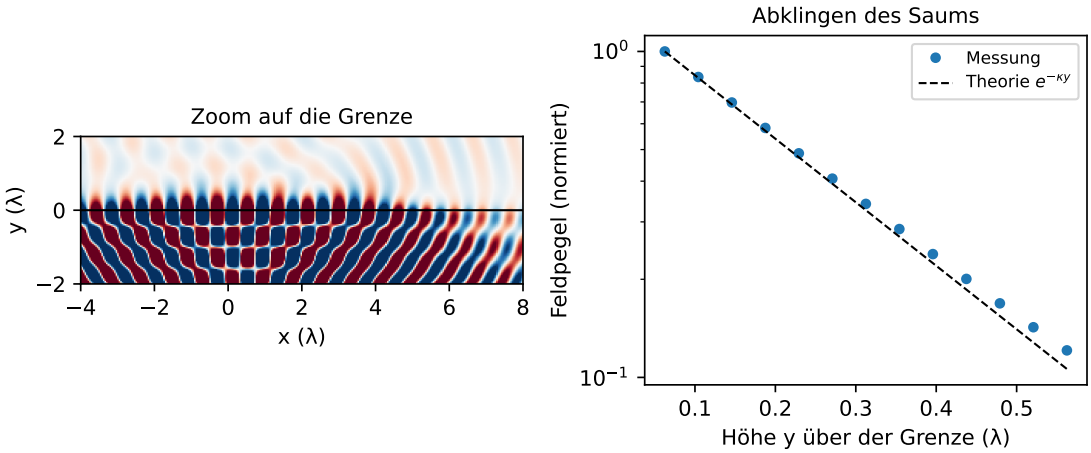


Abbildung 12.3: Links: Zoom auf die Grenzregion des 55°-Laufs (Farbskala um den Faktor 6 gespreizt) — der evaneszente Saum klebt als dünne Leuchtspur auf der Grenze. Rechts: sein Feldpegel (Wurzel aus dem x-Mittel von E_z^2, ein Punkt pro Gitterhöhe) über der Höhe y, logarithmisch aufgetragen; die gestrichelte Gerade ist die Theorie e^(−κy) mit κ = 4,49 — kein Fit, sondern Vorhersage. Der gemessene Abfall folgt ihr über eine Größenordnung; die Eindringtiefe ist knapp ein Viertel der Wellenlänge.

Die Punkte liegen auf der Vorhersage-Geraden: $\kappa = 4{,}30$ gemessen gegen $4{,}49$ berechnet, die Eindringtiefe trifft auf ein Hundertstel der Wellenlänge. Das Gespenst ist vermessen.

12.6 Die Bilanz, die erst beim dritten Monitor schließt

Bleibt die Kernbehauptung: *total* reflektiert. Die Flux-Maschinerie von oben liefert das in einer Zeile — und wer den Seitenmonitor im Code bemerkt hat, ahnt, warum wir ihn eingebaut haben:

```
# von oben: fluss_bilanz()
R55, T55, S55 = fluss_bilanz(55.0, mit_seite=True)
print(f"55°:  reflektiert      R = {R55:.4f}")
print(f"      oben hinaus      T = {T55:.4f}")
print(f"      seitlich (Saum)  S = {S55:.4f}")
print(f"      Summe              = {R55 + T55 + S55:.4f}")
```

```
55°:  reflektiert      R = 0.9889
      oben hinaus      T = 0.0040
      seitlich (Saum)  S = 0.0076
      Summe              = 1.0005
```

Auf den ersten Blick eine Enttäuschung: $R = 98{,}9\,\%$ — was ist das für eine „Total"-Reflexion? Aber die Checkliste sagt nicht „glaube der Theorie", sie sagt „schließe die Bilanz". Die fehlenden 1,1 % gehen an zwei adressierbare Posten. Erstens: $0{,}4\,\%$ verlassen die Zelle tatsächlich nach oben — unser Strahl ist kein Winkel, sondern ein **Winkelbündel**. Eine Taille von $w_0 = 3\lambda$ erkauft man mit etwa $\pm 6°$ Streuung um die $55°$ (das ist dieselbe Reziprozität wie in Kapitel 10: schmal im Raum = breit im Spektrum, hier im *Winkel*-Spektrum), und die äußersten Flügel des Bündels liegen nahe genug am Grenzwinkel, um teilweise zu entkommen. Zweitens: $0{,}8\,\%$ fließen **seitwärts** aus dem Messfenster — der Saum wandert mit k_x die Grenze entlang und trägt dabei Energie parallel zur Grenzfläche davon (außerdem tritt der reflektierte Strahl ein Stückchen seitlich versetzt aus, wo der Saum ihn zurückspeist — das Kleingedruckte kennt den Namen dazu). Mit allen drei Posten: $98{,}9 + 0{,}4 + 0{,}8 = 100{,}0\,\%$. Für die ideale, unendlich ausgedehnte ebene Welle sagt die Theorie exakt $R = 1$; unsere Messung bestätigt das so gut, wie ein endlicher Strahl es zulässt — und sie hat uns nebenbei gezwungen, dem Saum beim Energietragen zuzusehen.

12.7 Das Kapitel-Programm

`programme/kap12/kap12_brechung_totalreflexion.py` bündelt alle vier Befunde eigenständig und sichert sie mit `assert`-Schranken: k_x-Erhaltung und θ_2 (auf $0{,}5°$), $R(30°)$ gegen Fresnel (auf $0{,}01$) samt $R + T = 1$, Saum-$k_x > 1{,}1$ und κ auf $8\,\%$, Totalreflexions-Bilanz auf $2\,\%$ — dazu die Übungs-Befunde (Frequenz-Skalierung des Saums, Umkehrlauf).

Merkkasten

- **Snellius ist Schritthalten:** Die Wellenberge beider Seiten teilen sich die Fußabdrücke auf der Grenze — k_x ist erhalten, $n_1 \sin\theta_1 = n_2 \sin\theta_2$ ist nur die Strahlen-Übersetzung davon.
- **Beim Übergang ins dünnere Medium flieht der Strahl das Lot** — bis es bei $\theta_c = \arcsin(n_2/n_1)$ keinen Ausweg mehr gibt: Totalreflexion, der perfekte Spiegel ohne Spiegel. Nur vom dichten ins dünne Medium!
- **Hinter der total reflektierenden Grenze ist nicht nichts:** Ein

evaneszenter Saum klingt mit $\kappa = (2\pi/\lambda_0)\sqrt{(n_1 \sin\theta_1)^2 - n_2^2}$ ab (gemessen), trägt sein „zu schnelles" $k_x > k_0$ (gemessen) und schiebt Energie seitwärts, nie hinaus.
- **Energiebilanzen schließen, nicht abnicken:** R, T und der seitliche Abfluss summieren sich auf 100 % — gefunden mit Flux-Monitoren plus Referenzlauf (Kapitel 11).
- Ein Strahl ist ein **Winkelbündel**: Taille w_0 kostet Winkelschärfe $\sim \lambda/w_0$ — Kapitel-10-Reziprozität im Winkelraum.

Roter Faden

Das Kapitel hat von fast jedem Vorgänger geerbt: Von **Kapitel 4** den Brechungsindex $n = \sqrt{\varepsilon_r}$, von **Kapitel 6** den 4-%-Anker für Fresnel, von **Kapitel 8** die Nulldurchgangs-Messung *und* die Arcussinus-Sackgasse (Cutoff dort, Grenzwinkel hier — zweimal dieselbe Mathematik, zweimal stirbt eine laufende Welle), von **Kapitel 10** die Schmal-breit-Reziprozität (diesmal im Winkelraum), von **Kapitel 11** Flux-Monitore und die Referenzlauf-Methode, und Übung 11.3 war der Prolog. Nach vorn: **Kapitel 13** holt die Fresnel-Herleitung nach und lässt die zweite Polarisation ihren Brewster-Trick zeigen; **Kapitel 17** baut aus dem evaneszenten Saum eine Tunnelbrücke (FTIR) und erklärt nebenbei, warum die Glasfaser ihr Licht behält — der perfekte Spiegel aus diesem Kapitel ist ihr Betriebsgeheimnis.

Übungen

Ü 12.1 (Verstehen). Ein Speerfischer sieht einen Fisch schräg vor sich unter der Wasseroberfläche ($n_{\text{Wasser}} = 1{,}33$). Begründe mit dem Schritthalte-Argument (nicht mit auswendig gelerntem „zum Lot hin/vom Lot weg"): Knickt der Sehstrahl beim Übergang Wasser→Luft zum Lot oder vom Lot? Erscheint der Fisch dem Fischer dadurch **höher** oder **tiefer**, als er wirklich steht — und muss er beim Wurf darüber oder darunter zielen?

Musterlösung zu Ü 12.1

Im Wasser ist die Wellenlänge kürzer ($\lambda_0/1{,}33$). Damit die Fußabdrücke auf der Oberfläche Schritt halten ($\lambda/\sin\theta$ beidseits gleich), muss der Sinus in der Luft *größer* sein — das Licht vom Fisch verlässt das Wasser **vom Lot weggeknickt**, läuft also flacher zum Auge, als die gerade

Verbindung es täte. Der Fischer verlängert den ankommenden Strahl gedanklich geradlinig zurück und sieht den Fisch deshalb **zu hoch** (näher an der Oberfläche). Wer den Speer dorthin wirft, wo er den Fisch sieht, wirft zu hoch — gezielt wird **darunter**. (Dass der Speer auf seinem Weg ins Wasser selbst gebrochen würde, kompensiert das nicht: Der massive Speer fliegt seine ballistische Bahn — gebrochen wird nur das Licht.)

Ü 12.2 (Verstehen). Berechne von Hand die Grenzwinkel für Wasser→Luft ($n = 1{,}33$) und Diamant→Luft ($n = 2{,}42$); prüfe mit drei Zeilen Code. Erkläre dann mit den beiden Zahlen, warum ein geschliffener Brillant so auffällig funkelt — und warum derselbe Schliff aus Fensterglas ($n = 1{,}5$) vergleichsweise matt bliebe.

Musterlösung zu Ü 12.2

$\theta_c = \arcsin(1/n)$: für Wasser $\arcsin(0{,}752) = 48{,}8°$, für Diamant $\arcsin(0{,}413) = 24{,}4°$ (Glas: 41,8°).

```
# von oben: nichts (eigenständige Handrechnungs-Kontrolle)
for name, n in (("Wasser", 1.33), ("Glas", 1.5), ("Diamant", 2.42)):
    print(f"{name:8}: Grenzwinkel {np.degrees(np.arcsin(1 / n)):.1f}°")
```

```
Wasser  : Grenzwinkel 48.8°
Glas    : Grenzwinkel 41.8°
Diamant : Grenzwinkel 24.4°
```

Im Diamanten ist schon jeder Strahl total gefangen, der die Facette flacher als $24{,}4°$ zum Lot trifft — ein riesiger Winkelbereich. Ein gut geschliffener Brillant ist deshalb eine Falle aus perfekten (verlustfreien!) Spiegeln: Licht, das oben hineinfällt, wird an den Unterseiten-Facetten fast immer total reflektiert, springt mehrfach durch den Stein und verlässt ihn gebündelt wieder nach oben — es *blitzt*. Beim Glas-Imitat (Grenzwinkel $41{,}8°$) verfehlen viele dieser Strahlen die Totalreflexions-Bedingung, sickern durch die Unterseite hinaus, und der Stein wirkt stumpf. (Der zweite Diamant-Trumpf, die starke Farbzerlegung, braucht Dispersion — Kapitel 16.)

Ü 12.3 (Verändern). Der Saum hat bei $f_{\text{meep}} = 1$ die Eindringtiefe $\delta \approx 0{,}22$ Einheiten. Sage mit der κ-Formel voraus, was bei **halber Frequenz** ($f_{\text{meep}} = 0{,}5$, also doppelter Wellenlänge) passiert, und miss nach (gleicher Aufbau, `ContinuousSource(0.5, width=2)`, Profil wie im Kapitel — Fit-Fenster mitskalieren!). Warum ist das nur eine Umformulierung der Skaleninvarianz

aus Kapitel 11?

Musterlösung zu Ü 12.3

$\kappa \propto 1/\lambda_0$, also $\delta \propto \lambda_0$: Halbe Frequenz heißt doppelte Eindringtiefe, $0{,}45$ Einheiten. Das *muss* so sein, denn Maxwell kennt keine eigene Länge (Kapitel 11) — der Saum kann sich nur an der Wellenlänge messen, und „$\delta = 0{,}22\,\lambda_0$" ist die einzig mögliche Form des Ergebnisses. Die Messung:

```
# von oben: simulation(), idx(), DX, AUFL, SX, SY, kappa (Messwert f=1)
sim = simulation(55.0, mp.ContinuousSource(0.5, width=2))
sim.run(until=50.0)
ez_f05 = sim.get_array(component=mp.Ez, center=mp.Vector3(),
                       size=mp.Vector3(SX, SY))
i0, _ = idx(-3, 0); i1, _ = idx(3, 0); _, j0 = idx(0, 0)
h2, p2 = [], []
for j in range(j0 + 1, j0 + int(1.1 * AUFL)):     # Fenster: doppelt so hoch
    h2.append((j + 0.5) * DX - SY / 2)
    p2.append(np.sqrt(np.mean(ez_f05[i0:i1, j] ** 2)))
h2 = np.array(h2); p2 = np.array(p2)
m2 = (h2 > 0.05) & (h2 < 0.84)
kappa_f05 = -np.polyfit(h2[m2], np.log(p2[m2]), 1)[0]
print(f"κ(f=0,5) = {kappa_f05:.2f}  →  δ = {1 / kappa_f05:.2f}")
print(f"Verhältnis δ(f=0,5) / δ(f=1) = {kappa / kappa_f05:.2f}   (Theorie: 2)")
```

```
κ(f=0,5) = 1.89  →  δ = 0.53
Verhältnis δ(f=0,5) / δ(f=1) = 2.27   (Theorie: 2)
```

Gemessen rund 2,3 statt glatt 2 — kein Grund zur Panik, sondern ein bekannter Täter: Bei halber Frequenz ist die Taille w_0 nur noch anderthalb Wellenlängen breit, das Winkelbündel also doppelt so breit, und seine flachen Flügel (kleineres κ) strecken den gemessenen Abfall. Wer es genauer will, muss w_0 mitverdoppeln — und braucht dann eine größere Zelle: Auch Messaufbauten skalieren mit λ.

Ü 12.4 (Übertragen). Dreh den Spieß um: Lass den $55°$-Strahl **von oben** (aus dem Vakuum) auf das Glas fallen — `feldbild(55.0, von_oben=True)` baut das. Sage vorher: Gibt es jetzt Totalreflexion? In welche Richtung knickt der Strahl, und unter welchem Winkel läuft er im Glas? Miss den Winkel mit der k_x-Methode des Kapitels nach (Messlinie im Glas; bedenke, dass dort $|k| = n/\lambda_0$ gilt).

Musterlösung zu Ü 12.4

Keine Totalreflexion — abwärts (dünn → dicht) ist $\sin\theta_2 = \sin 55°/1{,}5 = 0{,}546$ immer lösbar: $\theta_2 = 33{,}1°$, der Strahl knickt **zum Lot hin**. Der Grenzwinkel ist eine Einbahnstraßen-Schranke. Die Messung — k_x ist weiterhin erhalten, nur ist der Gesamtbetrag im Glas $|k| = n/\lambda_0 = 1{,}5$, also $\sin\theta_2 = k_x/(nk_0)$:

```
# von oben: feldbild(), kx_aus_zeile(), N_GLAS
ez_oben = feldbild(55.0, von_oben=True)
kx_glas2 = kx_aus_zeile(ez_oben, -0.5)
print(f"kx im Glas: {kx_glas2:.4f}   (Theorie sin55° = "
      f"{np.sin(np.radians(55)):.4f})")
print(f"θ₂ = arcsin(kx/(n·k0)) = "
      f"{np.degrees(np.arcsin(kx_glas2 / N_GLAS)):.2f}°"
      f"   (Snellius: 33,10°)")
```

```
kx im Glas: 0.8119   (Theorie sin55° = 0.8192)
θ₂ = arcsin(kx/(n·k0)) = 32.77°   (Snellius: 33,10°)
```

Auch hier hält k_x Schritt — dieselben Fußabdrücke, nur diesmal von der schnellen auf die langsame Seite übergeben. Dass ein Teil reflektiert wird, bleibt übrigens auch abwärts wahr (rund 14 % bei diesem Winkel); nur das *Total* ist der Einbahnstraße vorbehalten.

Das Kleingedruckte

Der seitlich versetzte Wiederaustritt des reflektierten Strahls hat einen Namen: **Goos-Hänchen-Verschiebung** — der Strahl taucht ein Stück weit als Saum in das dünnere Medium ein, läuft dort ein Stückchen mit und tritt um etwa eine halbe Wellenlänge verschoben wieder aus. Für unsere Bilanz war er Teil des seitlichen Postens; messen lässt er sich erst mit deutlich schmaleren Strahlen, als unsere Zelle erlaubt. Und wer beim „perfekten Spiegel" an die PML aus Kapitel 9 gedacht hat: Genau andersherum — die PML ist der perfekte *Schlucker* und braucht dafür einen unphysikalischen Material-Trick, die Totalreflexion ist der perfekte *Rückwerfer* und braucht gar nichts. Echte Glasfasern verlieren übrigens doch ein wenig Licht — nicht an der Totalreflexion (die ist wirklich verlustfrei), sondern an Streuung und Restabsorption im Glas selbst; deshalb wird ihr Werkstoff so rein gekocht, dass man durch einen kilometerdicken Block noch hindurchsehen könnte.

13 Polarisation und Brewster

Setz eine Polarisations-Sonnenbrille auf und schau auf einen See: Die grellen Reflexe auf dem Wasser sind weg, man sieht hinein statt darauf. Jetzt neige den Kopf um neunzig Grad — die Reflexe sind wieder da. Die Brille hat also nicht einfach „Helligkeit geschluckt"; sie hat eine **Schwingungsrichtung** aussortiert, und das Spiegelbild auf dem Wasser steckt offenbar fast vollständig in einer einzigen solchen Richtung. Wie kommt ein simpler Wasser-Reflex zu einer so einseitigen Polarisation?

Die Antwort ist das fehlende Stück aus Kapitel 12. Dort haben wir die Fresnel-Formel benutzt, aber nicht hergeleitet — und nur eine der beiden. In diesem Kapitel holen wir beides nach: Die Herleitung kostet mit dem Schritthalte-Wissen aus Kapitel 12 und der Spiegelformel aus Kapitel 6 nur wenige Zeilen, und die zweite Polarisation versteckt darin eine echte Überraschung — einen Einfallswinkel, bei dem eine Glasfläche **aufhört zu spiegeln**. Vollständig. Wir rechnen ihn aus, messen ihn nach und sehen am Ende, warum die Anglerbrille funktioniert.

Lernziele

Nach diesem Kapitel kannst du …

1. … die Einfallsebene sowie s- und p-Polarisation erklären — und weißt, welche davon unsere 2D-Simulationen bisher stillschweigend gerechnet haben,
2. … beide Fresnel-Formeln aus der Spiegelformel von Kapitel 6 herleiten (Stichwort Schräg-Wellenwiderstand) und an Zahlenpunkten auswerten,
3. … den Brewster-Winkel berechnen ($\tan\theta_B = n_2/n_1$), geometrisch deuten (reflektiert $\perp$ gebrochen) und physikalisch begründen (ein Dipol strahlt nicht längs seiner Achse),
4. … $R(\theta)$-Kurven für beide Polarisationen messen und erklären, warum der Brewster-Reflex in der Messung klein, aber nicht exakt null ist,
5. … erklären, warum Reflexe von Wasser und Glas polarisiert sind und wie Polfilter-Brille und Brewster-Fenster das ausnutzen.

13.1 s und p: die Einfallsebene sortiert das Licht

Sobald Licht *schräg* auf eine Grenzfläche trifft, gibt es eine ausgezeichnete Ebene: die **Einfallsebene**, aufgespannt vom einfallenden Strahl und dem Lot auf die Grenze. In unseren 2D-Bildern *ist* die Zeichenebene die Einfallsebene. Die Polarisation des Lichts — die Querrichtung, in der $\vec{E}$ schwingt (Kapitel 4) — kann nun zwei Grundstellungen einnehmen, und jede beliebige Polarisation ist eine Mischung aus beiden:

- **s-Polarisation**: $\vec{E}$ steht **senkrecht** auf der Einfallsebene (zeigt also aus unserem Bild heraus — das deutsche „senkrecht" hat dem Fall weltweit den Buchstaben gegeben).
- **p-Polarisation**: $\vec{E}$ liegt **parallel zur Einfallsebene** (im Bild: schwingt schräg in der Zeichenebene, stets quer zur Laufrichtung — Transversalität, Kapitel 4).

Hier lohnt ein ehrlicher Rückblick: Unser gesamtes 2D-Leben seit Kapitel 9 — das TMz-Gitter mit E_z als Hauptfeld, die Ringwellen, das komplette Kapitel 12 — war **s-Polarisation**. Die Wahl fiel damals aus Bequemlichkeit (ein skalares E_z statt zweier Komponenten); jetzt wird sie inhaltlich: Es gibt eine zweite Hälfte der Welt. Bei der p-Polarisation tauschen die Felder die Rollen — dort zeigt das **Magnetfeld** aus der Ebene (H_z), während $\vec{E}$ in der Ebene liegt (das TEz-Gitter, der Zwilling aus dem Kapitel-9-Ausblick). Für Meep heißt das nur: Wir kippen den Polarisationsvektor der Quelle, und als Bildgröße zeigen wir H_z statt E_z.

13.2 Fresnel aus der Spiegelformel von Kapitel 6

Kapitel 6 hat für senkrechten Einfall hergeleitet:

$$r = \frac{Z_2 - Z_1}{Z_2 + Z_1}, \qquad Z = \frac{Z_0}{n}$$

— der Spiegelfaktor folgt aus zwei Stetigkeitsbedingungen an der Grenze (E und H tangential stetig) plus der Wellenbeziehung $H = E/Z$. Beim schrägen Einfall gelten **dieselben zwei Bedingungen** — stetig sein müssen die Feldkomponenten *parallel zur Grenzfläche*. Neu ist nur ein Geometriefaktor, und der hängt von der Polarisation ab:

- **s-Fall:** $\vec{E}$ (aus der Ebene) liegt schon komplett parallel zur Grenze; vom $\vec{H}$-Feld, das quer zum Strahl steht, liegt nur der Anteil $H\cos\theta$ in der Grenzfläche. Das Verhältnis der *tangentialen* Felder ist also nicht Z, sondern $E/(H\cos\theta) = Z/\cos\theta$.
- **p-Fall:** Jetzt liegt $\vec{H}$ (aus der Ebene) komplett parallel, und von $\vec{E}$ zählt nur $E\cos\theta$ — das tangentiale Verhältnis ist $Z\cos\theta$.

Mehr passiert nicht: In der Kapitel-6-Formel ersetzt man Z durch diesen **Schräg-Wellenwiderstand** Z_{eff}, jede Seite mit ihrem eigenen Winkel (θ_1 unten, θ_2 oben, verknüpft über Snellius). Einsetzen von $Z = Z_0/n$ und Kürzen liefert die beiden **Fresnel-Formeln**:

$$r_s = \frac{n_1\cos\theta_1 - n_2\cos\theta_2}{n_1\cos\theta_1 + n_2\cos\theta_2}, \qquad r_p = \frac{n_1\cos\theta_2 - n_2\cos\theta_1}{n_1\cos\theta_2 + n_2\cos\theta_1}, \qquad R = r^2.$$

Gleiche Bauform, aber die Winkel sitzen über Kreuz — das wird gleich den ganzen Unterschied machen. Erst die Kontrollen, je eine Handrechnung pro Formel (Glas $n_1 = 1{,}5$ → Vakuum, Einfall 30°, also $\theta_2 = 48{,}59°$ aus Kapitel 12):

- r_s: Zähler $1{,}5 \cdot 0{,}866 - 1 \cdot 0{,}661 = 0{,}638$, Nenner $1{,}960$, also $r_s = 0{,}325$, $R_s = 10{,}6\,\%$ — exakt die Zahl, die Kapitel 12 *gemessen* hat. Und bei $\theta_1 = 0$ kollabieren beide Formeln zu $(n_1 - n_2)/(n_1 + n_2) = 0{,}2$, dem Kapitel-6-Wert.
- r_p: Zähler $1{,}5 \cdot 0{,}661 - 1 \cdot 0{,}866 = 0{,}992 - 0{,}866 = 0{,}126$, Nenner $1{,}858$, also $r_p = 0{,}068$ und $R_p = 0{,}46\,\%$. Derselbe Winkel, dasselbe Glas — aber **dreiundzwanzigmal weniger** Reflexion als im s-Fall, nur weil das Feld anders herum schwingt.

```
import numpy as np

N_GLAS = 1.5

def fresnel(theta_grad, pol, n1=N_GLAS, n2=1.0):
    """Reflexionsgrad R für s- oder p-Polarisation (Einfall in Medium 1)."""
    t1 = np.radians(theta_grad)
    t2 = np.arcsin(n1 * np.sin(t1) / n2)        # Snellius (Kap. 12)
    if pol == "s":
        r = ((n1 * np.cos(t1) - n2 * np.cos(t2))
             / (n1 * np.cos(t1) + n2 * np.cos(t2)))
    else:
        r = ((n1 * np.cos(t2) - n2 * np.cos(t1))
```

```
            / (n1 * np.cos(t2) + n2 * np.cos(t1)))
    return r ** 2

for pol in ("s", "p"):
    print(f"R_{pol}(0°) = {fresnel(1e-9, pol):.4f}   "
          f"R_{pol}(30°) = {fresnel(30, pol):.4f}")
```

```
R_s(0°) = 0.0400   R_s(30°) = 0.1058
R_p(0°) = 0.0400   R_p(30°) = 0.0046
```

Beide Handrechnungen bestätigt — und der Kapitel-6-Wert $R(0°) = 4\,\%$ gleich mit.

13.3 Die Überraschung im p-Zähler

> ⚠ Naheliegende Vermutung
>
> **„Eine Grenzfläche spiegelt immer. Mal stärker, mal schwächer — aber ein bisschen Reflex bleibt bei jedem Winkel."**
> *Warum sie naheliegt:* Bisher stimmte es ausnahmslos. Senkrechter Einfall: 4 %. Schräg (s, 30°): 10,6 %. Über dem Grenzwinkel: 100 %. Jede Glasscheibe, jede Pfütze, jedes Schaufenster spiegelt — die Erfahrung kennt keine reflexfreie Grenzfläche.
> *Was stattdessen stimmt:* Für die p-Polarisation existiert genau ein Einfallswinkel, bei dem die Reflexion **vollständig verschwindet** — nicht klein wird: verschwindet. Eine Glasfläche, unter diesem Winkel mit p-Licht beleuchtet, ist ein perfektes Anti-Spiegel-Fenster. Die Erfahrung übersieht das nur, weil Alltagslicht beide Polarisationen mischt und die s-Hälfte brav weiterspiegelt.

Der Verdächtige ist der Zähler von r_p. Beim s-Zähler $n_1 \cos\theta_1 - n_2 \cos\theta_2$ kann (auf dem Weg Glas → Vakuum) nie null herauskommen: $n_1 > n_2$ *und* $\cos\theta_1 > \cos\theta_2$ — beide Faktoren der linken Seite gewinnen. Beim p-Zähler $n_1 \cos\theta_2 - n_2 \cos\theta_1$ aber ziehen die Faktoren gegeneinander, und tatsächlich wechselt er das Vorzeichen: Unsere Handrechnung gab bei 30° noch $+0{,}126$, bei $\theta_1 \to 0$ ist er $n_1 - n_2 < 0$... Moment — er ist negativ bei 0° und positiv bei 30°? Dann liegt **dazwischen eine Nullstelle.**

Wo genau, verrät ein geometrischer Ansatz. Probiere den Sonderfall $\theta_1 + \theta_2 = 90°$ — gebrochener und reflektierter Strahl stünden dann senkrecht

aufeinander (der reflektierte läuft unter θ_1 nach unten, der gebrochene unter θ_2 nach oben; Winkelsumme am Lot: $180° - \theta_1 - \theta_2 = 90°$). In diesem Fall ist $\cos\theta_2 = \cos(90° - \theta_1) = \sin\theta_1$, und der p-Zähler wird zu $n_1 \sin\theta_1 - n_2 \cos\theta_1$ — das ist null, wenn

$$\tan\theta_B = \frac{n_2}{n_1}.$$

Das ist der **Brewster-Winkel**. (Dass dieser Winkel die 90°-Bedingung auch wirklich erfüllt, prüft Snellius: $n_1 \sin\theta_B = n_2 \cos\theta_B = n_2 \sin\theta_2$ verlangt $\theta_2 = 90° - \theta_B$. □) Die Handrechnung für unsere Bühne, Glas → Vakuum: $\tan\theta_B = 1/1{,}5$, also $\theta_B = \arctan(0{,}667) = 33{,}69°$ — und der zugehörige Brechungswinkel ist $90° - 33{,}69° = 56{,}31°$. Beides sind Messvorhersagen.

Warum ausgerechnet $90°$? Hier kommt die Physik hinter der Algebra, und sie ist das schönste Argument des Kapitels. Die reflektierte Welle fällt ja nicht vom Himmel: Sie wird von den **schwingenden Ladungen der Materie** abgestrahlt, kleinen Dipolen, die das durchgehende Feld im Takt schüttelt (so entsteht „Reflexion" mikroskopisch). Ein schwingender Dipol strahlt aber nicht in alle Richtungen gleich — **längs seiner eigenen Achse strahlt er gar nicht** (in Achsrichtung „sieht" man die Querbewegung der Ladung nicht; Kapitel 21 misst diese Keule aus). Bei p-Polarisation schwingen die Dipole parallel zum $\vec{E}$ der gebrochenen Welle, also senkrecht zum gebrochenen Strahl — und genau am Brewster-Winkel zeigt die Richtung, in die der Reflex laufen müsste, exakt **entlang dieser Dipolachse**. Es gibt schlicht niemanden, der in diese Richtung strahlen könnte. Für s-Polarisation schwingen die Dipole dagegen aus der Ebene heraus — senkrecht zu *jeder* Strahlrichtung in der Ebene, deshalb hat der s-Fall nie eine Sendepause und nie einen Brewster-Winkel.

13.4 Die Messung: zwei Kurven, ein Einbruch

Der Versuchsaufbau, in einem Absatz, denn er ist der von Kapitel 12 mit zwei Drehknöpfen mehr: **Bühne** 24×12, Glas unten, Auflösung 20 Punkte pro λ_0; der Gaußstrahl ($w_0 = 3$) zielt von unten auf $(0, 0)$, als schmalbandiger Puls für saubere Flux-Bilanzen. **Neu 1:** Die Polarisation wird über den Feldvektor der Quelle gewählt — `beam_E0` $= (0, 0, 1)$ ist s; für p kippen wir ihn in die Ebene, senkrecht zur Laufrichtung. **Neu 2:** Gemessen wird $R(\theta)$ für **beide** Polarisationen über je sieben Winkel von $10°$ bis $35°$ — oberhalb wäre der Grenzwinkel $41{,}8°$ zu nahe: Die Winkel-Flügel des Bündels (Kapitel 12!) gerieten in die Totalreflexion und höben die Kurve. **Wichtig:** Der Referenzlauf (alles Glas) wird **für jede Polarisation einzeln** gefahren — die

eingespeiste Leistung unterscheidet sich zwischen s und p um gut ein Prozent, und bei einem erwarteten R_p unter einem Prozent wäre eine geteilte Referenz ein versteckter Systematikfehler genau auf Signalgröße. **Erfolgskriterium:** Beide Kurven auf den Fresnel-Linien (bis 30° auf wenige Promille absolut); am Brewster-Winkel bricht R_p um mehr als eine Größenordnung unter R_s ein.

```
import os
os.environ["HWLOC_HIDE_ERRORS"] = "2"   # wie in Kap. 11/12

import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)

AUFL = 20
SX, SY = 24.0, 12.0
DX = 1.0 / AUFL
F = 1.0
W0 = 3.0
Y_QUELLE = -3.5
# np.arctan: Umkehrung des Tangens (Gegenstück zu np.arcsin, Kap. 8)
TH_BREWSTER = np.degrees(np.arctan(1.0 / N_GLAS))
print(f"Brewster-Vorhersage: {TH_BREWSTER:.2f}°")

def simulation(theta_grad, pol, zeitverlauf, mit_grenze=True,
               von_oben=False):
    """Gaußstrahl auf (0,0) wie in Kap. 12; pol: "s" (Ez) oder "p" (Hz)."""
    th = np.radians(theta_grad)
    vz = -1.0 if von_oben else 1.0
    kdir = mp.Vector3(np.sin(th), vz * np.cos(th))
    e0 = (mp.Vector3(0, 0, 1) if pol == "s"            # s: E aus der Ebene
          else mp.Vector3(vz * np.cos(th), -np.sin(th), 0))  # p: E in der
    if mit_grenze:                                     #   Ebene, ⊥ kdir
        geometrie = [mp.Block(size=mp.Vector3(mp.inf, SY),
                              center=mp.Vector3(0, -SY / 2),
                              material=mp.Medium(index=N_GLAS))]
    elif von_oben:
        geometrie = []                  # Referenz von oben: alles Vakuum
    else:
        geometrie = [mp.Block(size=mp.Vector3(mp.inf, mp.inf, mp.inf),
                              material=mp.Medium(index=N_GLAS))]
    y_q = -Y_QUELLE if von_oben else Y_QUELLE
    quelle = mp.GaussianBeamSource(
```

```
        zeitverlauf, center=mp.Vector3(-abs(y_q) * np.tan(th), y_q),
        size=mp.Vector3(10, 0),
        component=mp.ALL_COMPONENTS,    # Polarisation steckt in beam_E0
        beam_kdir=kdir, beam_w0=W0, beam_x0=mp.Vector3(0, 0), beam_E0=e0)
    return mp.Simulation(cell_size=mp.Vector3(SX, SY), resolution=AUFL,
                         boundary_layers=[mp.PML(1.0)], sources=[quelle],
                         geometry=geometrie)


def reflexionsgrad(theta_grad, pol, von_oben=False):
    """R per Puls + Referenzlauf (die Kapitel-12-Methode)."""
    y_mon = +1.8 if von_oben else -1.8
    werte = {}
    for mit_grenze in (False, True):
        sim = simulation(theta_grad, pol, mp.GaussianSource(F, fwidth=F / 8),
                         mit_grenze=mit_grenze, von_oben=von_oben)
        monitor = sim.add_flux(F, 0, 1, mp.FluxRegion(
            center=mp.Vector3(0, y_mon), size=mp.Vector3(20, 0)))
        sim.run(until_after_sources=50)
        flux = mp.get_fluxes(monitor)[0]
        werte[mit_grenze] = -flux if von_oben else flux   # +: bergauf zählt
    return (werte[False] - werte[True]) / werte[False]
```

```
Using MPI version 4.1, 1 processes
Brewster-Vorhersage: 33.69°
```

Vorhersage-Punkt

Skizziere beide Kurven $R_s(\theta)$ und $R_p(\theta)$ von $10°$ bis $35°$, bevor du die Messung startest: Wo starten sie bei kleinem Winkel (Kapitel 6 verrät es), welche steigt, welche fällt — und was passiert bei $33{,}7°$?

Die Messung dauert einen Kaffee lang (28 Meep-Läufe — zwei Polarisationen, sieben Winkel, je Haupt- und Referenzlauf):

```
# von oben: fresnel(), reflexionsgrad(), TH_BREWSTER
WINKEL = np.array([10.0, 15.0, 20.0, 25.0, 30.0, TH_BREWSTER, 35.0])
gemessen = {}
for pol in ("s", "p"):
    gemessen[pol] = np.array([reflexionsgrad(th, pol) for th in WINKEL])

fig, ax = plt.subplots(figsize=(6.6, 3.8))
```

```
th_fein = np.linspace(5, 40, 200)
for pol, farbe in (("s", "tab:blue"), ("p", "tab:orange")):
    ax.plot(th_fein, [fresnel(t, pol) for t in th_fein], color=farbe,
            lw=1.2, label=f"Fresnel {pol}")
    ax.plot(WINKEL, gemessen[pol], "o", color=farbe, ms=5,
            label=f"Messung {pol}")
ax.axvline(TH_BREWSTER, color="gray", ls=":", lw=1.0)
ax.annotate("Brewster\n33,7°", (TH_BREWSTER, 0.27), ha="center", fontsize=9)
ax.set_xlabel("Einfallswinkel θ₁ (°)")
ax.set_ylabel("Reflexionsgrad R")
ax.set_ylim(0, 0.45)
ax.legend(fontsize=8, loc="upper left")
plt.tight_layout(); plt.show()

i_b = np.argmin(np.abs(WINKEL - TH_BREWSTER))
print(f"am Brewster-Winkel: R_s = {gemessen['s'][i_b]:.4f},"
      f"  R_p = {gemessen['p'][i_b]:.4f}"
      f"  (Faktor {gemessen['s'][i_b] / gemessen['p'][i_b]:.0f})")
```

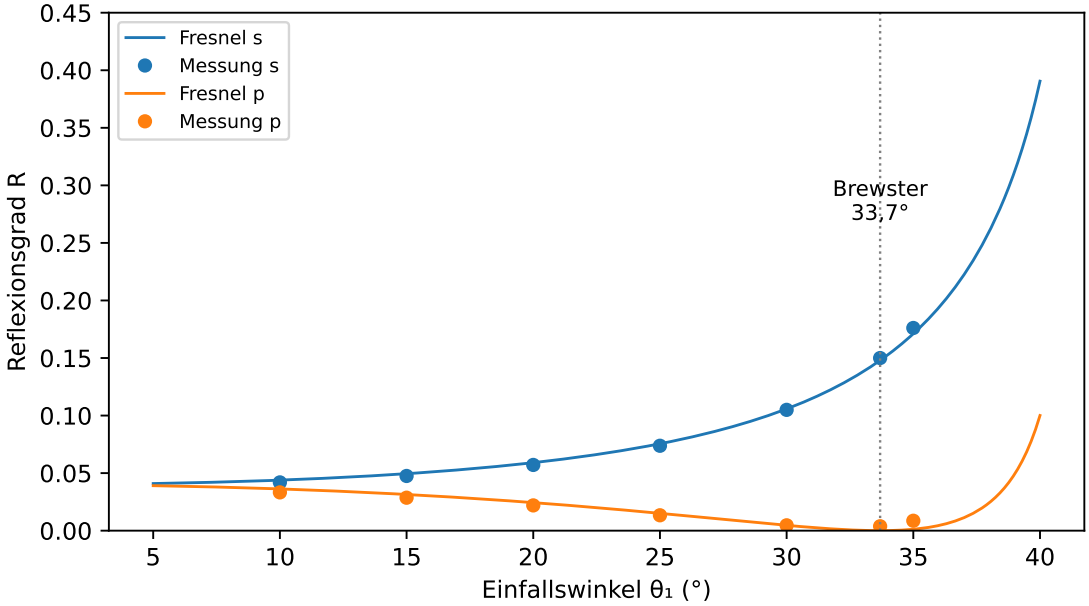


Abbildung 13.1: Der Reflexionsgrad der Glas-Vakuum-Grenze für beide Polarisationen, von 10° bis 35°: Linien = Fresnel-Formeln, Punkte = Meep-Messung (Flux + Referenzlauf). Beide starten bei den 4 % des senkrechten Einfalls (Kapitel 6). Die s-Kurve (blau) steigt monoton — sie kennt keinen Brewster-Winkel. Die p-Kurve (orange) fällt und geht bei 33,7° (gepunktete Linie) durch ihre Nullstelle; der gemessene Restwert von 0,4 % stammt vom Winkelbündel des Strahls. Oberhalb von 35° rasen beide Theoriekurven auf die Totalreflexion bei 41,8° zu — gemessen wird dort nicht mehr, weil die Winkel-Flügel des Bündels in die Totalreflexion gerieten.

```
am Brewster-Winkel: R_s = 0.1500,  R_p = 0.0038  (Faktor 39)
```

Beide Kurven liegen auf ihren Fresnel-Linien, und am Brewster-Winkel klafft zwischen den Polarisationen ein **Faktor vierzig**: $15\,\%$ gegen $0{,}4\,\%$. Dass die p-Messung nicht exakt null zeigt, ist kein Schönheitsfehler der Theorie, sondern — wie schon bei der 98,9-%-Totalreflexion in Kapitel 12 — die Winkelbreite des Bündels: R_p ist nur *exakt am* Brewster-Winkel exakt null und wächst quadratisch daneben; ein $\pm 6°$-Bündel mittelt darüber und sammelt die Reste der Flügel ein. Die ideale ebene Welle träfe die Null exakt.

13.5 Das Foto zum Beweis — und die 90°-Probe

Zahlen überzeugen den Verstand, Bilder das Bauchgefühl. Derselbe Aufbau noch einmal als Dauerstrich-Schnappschuss, beide Polarisationen exakt am Brewster-Winkel — beim p-Lauf zeigt das Bild das Feld H_z (das skalare Feld der p-Welt):

```
# von oben: simulation(), TH_BREWSTER, SX, SY, F
felder = {}
for pol, komponente in (("s", mp.Ez), ("p", mp.Hz)):
    sim = simulation(TH_BREWSTER, pol, mp.ContinuousSource(F, width=2))
    sim.run(until=45.0)
    felder[pol] = sim.get_array(component=komponente, center=mp.Vector3(),
                                size=mp.Vector3(SX, SY))

fig, achsen = plt.subplots(2, 1, figsize=(7.0, 5.4))
th_b = np.radians(TH_BREWSTER)
th_2 = np.radians(90.0 - TH_BREWSTER)
for ax, pol, name in ((achsen[0], "s", "s-Polarisation (Ez): Reflex da"),
                      (achsen[1], "p", "p-Polarisation (Hz): Reflex weg")):
    feld = felder[pol]
    w = np.abs(feld).max() / 2.5
    ax.imshow(feld.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w,
              extent=(-SX / 2, SX / 2, -SY / 2, SY / 2))
    ax.axhline(0, color="black", lw=0.9)
    ax.plot([-5.5 * np.tan(th_b), 0], [-5.5, 0], "k--", lw=0.9)
    ax.plot([0, 5.5 * np.tan(th_2)], [0, 5.5], "k--", lw=0.9)
    if pol == "s":
        ax.plot([0, 5.5 * np.tan(th_b)], [0, -5.5], "k--", lw=0.9)
    ax.set_title(name, fontsize=10)
    ax.set_ylabel("y (λ)")
achsen[1].set_xlabel("x (λ)")
plt.tight_layout(); plt.show()
```

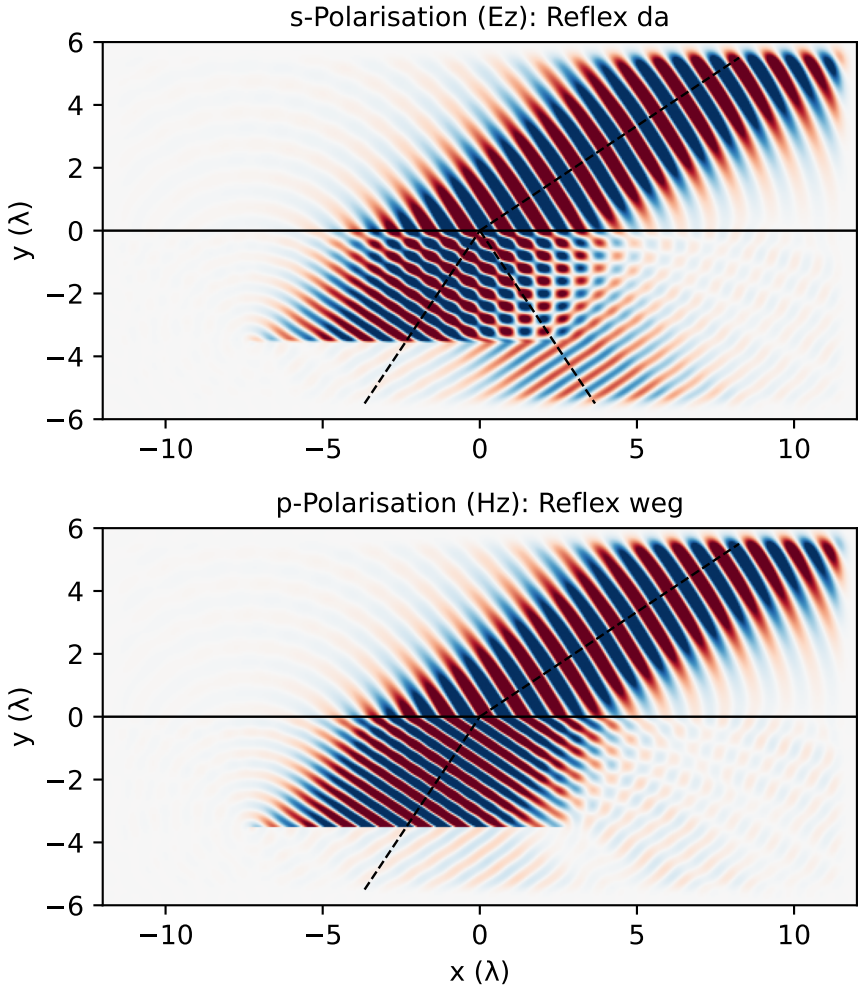


Abbildung 13.2: Beide Polarisationen am Brewster-Winkel 33,7° (oben Ez der s-Welle, unten Hz der p-Welle; gestrichelt die Erwartungslinien für einfallenden, gebrochenen — und beim s-Bild reflektierten — Strahl). Oben das gewohnte Bild: einfallender, gebrochener und ein deutlicher reflektierter Strahl samt Interferenz-Streifen im Glas. Unten fehlt der reflektierte Arm vollständig — die Grenzfläche spiegelt nicht, das V aus Einfall und Reflex ist zur einfachen Knicklinie geworden. Reflektierte Richtung und gebrochener Strahl stünden hier senkrecht aufeinander: Der Reflex müsste exakt längs der Dipolachsen laufen, und dorthin strahlt niemand.

Und weil zu jedem schönen Bild eine Zahl gehört: Die 90°-Behauptung ist messbar. Der gebrochene Strahl im p-Bild trägt sein k_x (Kapitel 12), und

daraus folgt sein Winkel — der zusammen mit dem Brewster-Winkel $90°$ ergeben muss:

```
# von oben: felder, SX, SY, AUFL, DX, TH_BREWSTER
def idx(x, y):
    return int(round((x + SX / 2) * AUFL)), int(round((y + SY / 2) * AUFL))


def kx_aus_zeile(feld, y, x0=-4.0, x1=4.0):
    """Nulldurchgangs-Methode, wörtlich aus Kapitel 12."""
    i0, _ = idx(x0, 0); i1, _ = idx(x1, 0); _, j = idx(0, y)
    zeile = feld[i0:i1, j]
    s = np.flatnonzero(np.sign(zeile[:-1]) * np.sign(zeile[1:]) < 0)
    null = s - zeile[s] / (zeile[s + 1] - zeile[s])
    return 1.0 / (2 * np.mean(np.diff(null)) * DX)


kx_oben = kx_aus_zeile(felder["p"], +0.5)
theta2 = np.degrees(np.arcsin(min(1.0, kx_oben)))
print(f"kx über der Grenze: {kx_oben:.4f}  →  θ₂ = {theta2:.2f}°")
print(f"θ_B + θ₂ = {TH_BREWSTER + theta2:.2f}°   (Erwartung: 90°)")
```

```
kx über der Grenze: 0.8328  →  θ₂ = 56.39°
θ_B + θ₂ = 90.08°   (Erwartung: 90°)
```

$90{,}1°$ — die Geometrie hinter dem Dipol-Argument, im eigenen Feldbild nachgemessen.

13.6 Wozu das gut ist: Brillen und Laser

Die Anglerbrille. Sonnenlicht ist unpolarisiert — eine wild wechselnde Mischung aus s und p zu gleichen Teilen. Der Wasser-Reflex aber behandelt die Hälften völlig verschieden: Nahe am Brewster-Winkel (für Wasser, $n = 1{,}33$, liegt er bei $53°$ — ein sehr alltäglicher Blickwinkel auf einen See) wird die p-Hälfte kaum bis gar nicht reflektiert, die s-Hälfte kräftig. Das Spiegelbild auf dem Wasser besteht also fast nur aus s-Licht — und „s" heißt hier: $\vec{E}$ **horizontal** (senkrecht zur Einfallsebene, die das Lot enthält). Die Polfilter-Brille sperrt schlicht horizontal schwingendes Licht (ein Skalarprodukt aus Plastik, Kapitel 4), löscht damit gezielt den Reflex und lässt das unpolarisierte Licht aus der Tiefe halbiert passieren. Neigt man den Kopf um $90°$, sperrt

die Brille die falsche Richtung — der Reflex ist zurück. Übung 13.3 macht aus dieser Geschichte eine Messkurve.

Das Brewster-Fenster. Die Umkehrung desselben Tricks: Wer in einem Laser Licht tausendfach durch ein Fenster schicken muss, kann selbst 4 % Verlust pro Durchgang nicht bezahlen. Ein im Brewster-Winkel schräg gestelltes Fenster ist für p-Licht **verlustfrei** — und weil die s-Hälfte bei jedem Umlauf 15 % Reflexionsverlust erleidet, gewinnt im Wettbewerb um die Verstärkung die p-Welle: Der Laserstrahl kommt von selbst sauber polarisiert heraus. Dieselbe Physik, die der Angler trägt, steckt gekippt in fast jedem Gaslaser.

13.7 Das Kapitel-Programm

`programme/kap13/kap13_polarisation_brewster.py` bündelt die Befunde eigenständig und sichert sie mit `assert`-Schranken: beide $R(\theta)$-Kurven bis $30°$ auf $0{,}005$ absolut an Fresnel, Brewster-Punkt $R_p < 0{,}006$ bei Faktor > 25 zu R_s, 90°-Probe auf $0{,}5°$, und der Umkehrlauf von oben (Ü 13.4) mit $R_p = 0{,}002$ gegen $R_s = 0{,}142$.

Merkkasten

- **s** = $\vec{E}$ senkrecht zur Einfallsebene (unsere TMz-Welt seit Kapitel 9), **p** = $\vec{E}$ in der Einfallsebene (H_z-Welt). Jede Polarisation ist eine Mischung aus beiden.
- **Fresnel = Kapitel-6-Formel mit Schräg-Wellenwiderstand:** s: $Z/\cos\theta$, p: $Z\cos\theta$ — mehr Herleitung braucht es nicht.
- **Brewster:** $\tan\theta_B = n_2/n_1$ (Glas → Luft $33{,}7°$, Luft → Glas $56{,}3°$ — komplementär). Dort: reflektiert ⊥ gebrochen, der Reflex müsste längs der Dipolachsen laufen — und Dipole strahlen dorthin nicht. Nur p hat das; s-Dipole zeigen aus der Ebene und senden immer.
- **Reflexe sind polarisiert:** nahe Brewster fast rein s (horizontal) — deshalb funktioniert die Polfilter-Brille, und deshalb polarisiert sich ein Laser am Brewster-Fenster selbst.
- Ein gemessener **Brewster-Restwert** ($0{,}4\,\%$ statt null) ist Bündel-Physik, kein Theoriefehler — dieselbe Lektion wie bei der 98,9-%-Totalreflexion.

Roter Faden

Mit diesem Kapitel ist die Polarisation komplett: **Kapitel 4** gab ihr Richtung und Malus-Projektion, hier bekam sie ihre Grenzflächen-Physik — und das Versprechen aus dem Kapitel-4-Polfilter („ein Skalarprodukt aus Plastik") zahlt sich in einer Brille aus. Die Herleitung lebte von **Kapitel 6** (Spiegelformel) und **Kapitel 12** (Schritthalten, k_x-Messung, Flux samt Referenzlauf, Bündel-Flügel); die stillschweigende TMz-Wahl aus **Kapitel 9** wurde endlich zur bewussten Entscheidung. Nach vorn: Das Dipol-Argument bekommt in **Kapitel 21** seine eigene Bühne (die Strahlungskeule des Dipols, vermessen), das Impedanz-Denken kehrt in **Kapitel 22** als Leitungstheorie wieder, und **Kapitel 28** braucht für die Doppelbrechung genau das hier gelegte Fundament — Materialien, in denen n selbst von der Polarisation abhängt.

Übungen

Ü 13.1 (Verstehen). Ein Fotograf will das Spiegelbild auf einem Schaufenster loswerden, ein anderer das Glitzern auf nassem Asphalt. Erkläre für beide: In welcher Richtung muss die Durchlassachse des Polfilters stehen — und warum hilft der Filter beim Schaufenster nur richtig, wenn der Fotograf *schräg* zur Scheibe steht?

> **Musterlösung zu Ü 13.1**
>
> Reflexe sind s-dominiert, und s heißt: $\vec{E}$ senkrecht zur Einfallsebene. Beim nassen Asphalt (waagerechte Fläche, Lot senkrecht) ist die Einfallsebene vertikal — s ist **horizontal**, der Filter muss also vertikal durchlassen (wie die Anglerbrille). Beim Schaufenster steht die Fläche senkrecht; blickt der Fotograf schräg von der Seite darauf, ist die Einfallsebene horizontal, der Reflex schwingt **vertikal**, und die Durchlassachse muss horizontal stehen — um $90°$ anders als auf der Straße. Steht der Fotograf dagegen frontal vor der Scheibe ($\theta_1 \approx 0$), gibt es keine ausgezeichnete Einfallsebene: Der Reflex ist unpolarisiert, $R_s = R_p = 4\,\%$, und kein Filter der Welt kann ihn auslöschen — er kann nur die Hälfte schlucken. Der Polfilter wirkt umso besser, je näher der Blickwinkel am Brewster-Winkel liegt ($56°$ zum Lot bei Glas).

Ü 13.2 (Verstehen). Berechne von Hand die Brewster-Winkel für den Blick **aus der Luft** auf Wasser ($n = 1{,}33$), Glas ($1{,}5$) und Diamant ($2{,}42$); prüfe mit drei Zeilen Code. Berechne dann für Glas zusätzlich den Brewster-Winkel der **Gegenrichtung** (Glas → Luft) und addiere beide. Zufall?

Musterlösung zu Ü 13.2

$\theta_B = \arctan(n_2/n_1)$, von Luft aus also $\arctan(n)$: Wasser $53{,}1°$, Glas $56{,}3°$, Diamant $67{,}5°$. Gegenrichtung Glas → Luft: $\arctan(1/1{,}5) = 33{,}7°$ — und $56{,}3° + 33{,}7° = 90°$. Kein Zufall: Der Brewster-Winkel der einen Seite ist der Brechungswinkel der anderen (die 90°-Geometrie ist symmetrisch — derselbe Strahlenknick, nur rückwärts durchlaufen, und Lichtwege sind umkehrbar). Anders als der Grenzwinkel ist Brewster also **keine Einbahnstraße**: Es gibt ihn in beide Richtungen, nur an komplementären Winkeln.

```
# von oben: N_GLAS
for name, n in (("Wasser", 1.33), ("Glas", N_GLAS), ("Diamant", 2.42)):
    print(f"Luft → {name:8}: θ_B = {np.degrees(np.arctan(n)):.1f}°")
print(f"Glas → Luft     : θ_B = {np.degrees(np.arctan(1 / N_GLAS)):.1f}°"
      f"   (Summe mit Luft → Glas: 90°)")
```

```
Luft → Wasser  : θ_B = 53.1°
Luft → Glas    : θ_B = 56.3°
Luft → Diamant : θ_B = 67.5°
Glas → Luft     : θ_B = 33.7°   (Summe mit Luft → Glas: 90°)
```

Ü 13.3 (Verändern). Unpolarisiertes Licht ist eine 50:50-Mischung: $R_{\text{unpol}} = (R_s + R_p)/2$. Der **Polarisationsgrad** des Reflexes, $P = (R_s - R_p)/(R_s + R_p)$, misst, wie einseitig er ist (0 = unpolarisiert, 1 = perfekt s-polarisiert). Berechne $P(\theta)$ aus den schon gemessenen Kurven des Kapitels — keine neue Simulation nötig — und beantworte: Wie nah am Brewster-Winkel muss der Angler-Blick liegen, damit die Brille mehr als $90\,\%$ des Reflexes erwischen kann?

Musterlösung zu Ü 13.3

```
# von oben: WINKEL, gemessen (die R(θ)-Messwerte des Kapitels)
P = (gemessen["s"] - gemessen["p"]) / (gemessen["s"] + gemessen["p"])
for th, p_grad in zip(WINKEL, P):
    print(f"θ = {th:5.2f}°:  P = {p_grad:.2f}")
```

```
θ = 10.00°:  P = 0.12
θ = 15.00°:  P = 0.25
θ = 20.00°:  P = 0.44
θ = 25.00°:  P = 0.69
θ = 30.00°:  P = 0.92
```

```
θ = 33.69°:  P = 0.95
θ = 35.00°:  P = 0.91
```

Schon bei 10° ist der Reflex zu gut 10 % einseitig, ab etwa 30° zu über 90 %, am Brewster-Winkel zu 95 % (ideal: 100 % — der Rest ist wieder

i Musterlösung zu Ü 13.4

Ü 13.4 (Bündel). Die gute Nachricht für den Angler: Das Fenster ist **breit**. R fällt am Brewster-Winkel nur langsam ab, R_p in Polarisationsrichtung quadratisch klein bleibt. Wie groß werden R_p und R_s unweit von 53°? Bei 55,31° oben und 53,69° unten sind R_s und R_p gut 95 %, R_s etwa 95 % — denn 55,31° oben und 53,69° unten sind über Snellius **dasselbe Strahlenpaar**, und beim Medientausch geht der Medien-Faktor $r_s \to 1/r_s$, also R_s unverändert.

```python
# von oben: reflexionsgrad(), TH_BREWSTER
th_oben = np.degrees(np.arctan(N_GLAS))
rp = reflexionsgrad(th_oben, "p", von_oben=True)
rs = reflexionsgrad(th_oben, "s", von_oben=True)
print(f"von oben bei {th_oben:.2f}°:  R_p = {rp:.4f},  R_s = {rs:.4f}")
```

```
von oben bei 56.31°:  R_p = 0.0020,  R_s = 0.1418
```

Gemessen: $R_p = 0{,}2\,\%$ (Bündel-Rest), $R_s = 14\,\%$ — beide Vorhersagen treffen. Bemerkenswert daran: Von oben gibt es **keinen Grenzwinkel** (Kapitel 12, Einbahnstraße), aber sehr wohl einen Brewster-Winkel. Die beiden Phänomene sind unabhängig — Totalreflexion braucht das dichtere Ausgangsmedium, Brewster nur eine Grenzfläche und die richtige Schwingungsrichtung.

Das Kleingedruckte

Drei Ehrlichkeiten zum Schluss. Erstens: Auf **Metallen** funktioniert der Polfilter-Trick schlecht — ihre Reflexion entsteht nicht durch brave Dipole in einem Dielektrikum, sondern durch frei verschiebbare Leitungselektronen; es gibt nur ein flaches „Pseudo-Brewster"-Minimum, keine Null. Der Chrom-Glanz einer Stoßstange bleibt durch jede Anglerbrille sichtbar — ein schöner Feldtest, *welche* Sorte Spiegel man vor sich hat. Zweitens: Genau am Brewster-Winkel ist die durchgelassene p-Welle zwar verlustfrei, aber der **Strahl versetzt sich** beim Durchgang durch eine endliche dicke Platte seitlich — wer mit Brewster-Fenstern baut, richtet danach neu aus. Drittens: Unsere 2D-Welt zeigt s und p als getrennte Universen (E_z- und H_z-Gitter koppeln nicht). In 3D mischt eine schräg stehende Polarisation beide Anteile, und die Grenzfläche zerlegt sie — nichts daran ist neu, es ist Vektorzerlegung plus die zwei Kurven dieses Kapitels.

14 Interferenz: Doppelspalt und Gitter

Halte eine CD schräg ins Licht: Sie schillert in allen Regenbogenfarben — dabei ist die Scheibe selbst farblos. Kein Farbstoff, kein Filter, nur Millionen feiner Datenrillen im Abstand von anderthalb Mikrometern. Irgendetwas an dieser regelmäßigen Struktur *sortiert* weißes Licht nach Wellenlängen, und zwar so präzise, dass Astronomen mit demselben Prinzip die chemische Zusammensetzung von Sternen bestimmen.

Dieses Etwas ist **Interferenz** — die Kunst der Wellen, sich gegenseitig auszulöschen oder zu verstärken, je nachdem, ob sie im Takt oder im Gegentakt ankommen. Wir kennen den Mechanismus seit Kapitel 1 unter dem nüchternen Namen Superposition: Felder addieren sich. In diesem Kapitel lassen wir diese harmlose Addition ihr berühmtestes Schauspiel aufführen — vom Streifenmuster zweier Quellen über den Doppelspalt bis zum Gitter, das am Ende tatsächlich zwei „Farben" in zwei Richtungen sortiert. Unterwegs misst jede Behauptung nach, und ein neues Meep-Werkzeug schenkt uns den unendlich weit entfernten Schirm.

Lernziele

Nach diesem Kapitel kannst du …

1. … das Streifenmuster zweier Quellen aus dem **Gangunterschied** vorhersagen ($d \sin\theta = m\lambda$) und erklären, wohin an den dunklen Streifen die Energie verschwindet (nirgendwohin — sie ist umverteilt),
2. … den Doppelspalt auf zwei Huygens-Quellen zurückführen und die Einzelspalt-Hüllkurve im gemessenen Profil nachweisen,
3. … begründen und messen, warum ein Gitter mit N Spalten N^2-fach hellere und $1/N$-fach schmalere Maxima wirft,
4. … mit Meeps `near2far` das Fernfeld berechnen — und weißt, wann ein „Schirm" zu nah ist, um die Wahrheit zu zeigen,
5. … erklären, wie ein Gitter Farben sortiert ($\sin\theta_m = m\lambda/d$), warum Rot stärker abgelenkt wird als Blau — und warum eine CD schillert.

14.1 Der Gangunterschied entscheidet

Zwei Quellen schwingen im selben Takt (dieselbe Frequenz, fester Phasenbezug — die Fachsprache nennt das *kohärent*; Ü 14.1 klärt, warum das im Wellenbad leicht und mit Taschenlampen aussichtslos ist). Ein Beobachter steht weit weg, in einer Richtung θ schräg zur Mittelachse. Die Welle der ferneren Quelle muss dann ein Stück weiter laufen als die der näheren — um den **Gangunterschied**, und der ist reine Geometrie (dasselbe Dreieck wie beim Schritthalten in Kapitel 12):

$$\Delta = d\,\sin\theta,$$

mit d dem Quellabstand. Ist Δ ein ganzes Vielfaches der Wellenlänge, $\Delta = m\lambda$, kommen beide Wellen im Takt an: Berg trifft Berg, die Amplituden addieren sich — **konstruktive Interferenz**, ein heller Streifen der Ordnung m. Liegt Δ genau dazwischen, $\Delta = (m + \frac{1}{2})\lambda$, trifft Berg auf Tal: Auslöschung, dunkel. Die Form des Übergangs liefert das Additionstheorem (das dritte Mal in diesem Buch, nach den Kapiteln 8 und 10): Zwei gleich starke Schwingungen mit Phasenversatz $\varphi = 2\pi\Delta/\lambda$ addieren sich zu einer Schwingung der Amplitude $2\cos(\varphi/2)$ — die Intensität (das mittlere Feldquadrat, Kapitel 4) folgt also

$$I(\theta) \;\propto\; \cos^2\!\left(\frac{\pi\, d \sin\theta}{\lambda}\right).$$

Die Handrechnung für unsere Bühne ($d = 4\lambda$): Der erste helle Streifen neben der Mitte ($m = 1$) liegt bei $\sin\theta = \lambda/d = 0{,}25$, also $\theta = 14{,}5°$; der erste dunkle bei $\sin\theta = 0{,}125$, also $7{,}2°$. Auf einem Schirm in Abstand $L = 15$ (dazu gleich) erwarten wir den m=1-Streifen bei $y \approx L\tan\theta = 3{,}9$.

> ⚠ Naheliegende Vermutung
>
> **„Licht plus Licht gibt helleres Licht. Zwei Lampen sind heller als eine — immer.”**
> *Warum sie naheliegt:* Im Alltag stimmt es ausnahmslos; zwei Glühbirnen haben noch nie einen dunklen Fleck erzeugt. Und Energieerhaltung scheint es zu verlangen: Zwei Quellen speisen doppelte Leistung ein, also muss es überall heller werden.
> *Was stattdessen stimmt:* **Felder** addieren sich, nicht Helligkeiten. Wo zwei gleich starke Wellen im Gegentakt ankommen, ist es *dunkel* — Licht plus Licht gleich Finsternis. Die Energie fehlt dort wirklich, verletzt ist trotzdem nichts: An den hellen Streifen ist es dafür **viermal**

so hell wie mit einer Quelle (Amplitude verdoppelt, Intensität vervierfacht), und über den Schirm gemittelt bleibt exakt das Doppelte übrig. Interferenz ist ein Umverteiler, kein Schöpfer. Alle drei Zahlen — 4, 0 und 2 — messen wir gleich nach. (Der Alltag sieht davon nichts, weil Glühbirnen nicht kohärent sind; Ü 14.1.)

14.2 Zwei Quellen auf der Messbank

Der Versuchsaufbau: **Bühne** — eine 24×24-Zelle ($\lambda_0 = 1$, also $f_{\text{meep}} = 1$; Auflösung 16 Punkte pro λ), ringsum PML. **Quellen:** zwei gleichphasige Ez-Punktquellen (Dauerstrich) bei $x = -6$, $y = \pm 2$ — Abstand $d = 4\lambda$. **Gemessen wird** auf einem „Schirm": einer senkrechten Linie bei $x = +9$ ($L = 15$ hinter den Quellen), knapp vor der PML. Als Messgröße dient die **Intensität**, das Zeitmittel von E_z^2 — praktisch: einschwingen lassen, dann E_z über eine volle Periode in 16 Schnappschüssen mitteln. **Erwartung** als einzeichenbare Theorielinie: nicht nur die cos²-Form, sondern die ehrliche Version mit allem, was wir wissen — die Amplitude jeder 2D-Quelle fällt wie $1/\sqrt{r}$ (Kapitel 9), also

$$I(y) \;\propto\; \frac{1/2}{r_1} + \frac{1/2}{r_2} + \frac{\cos\bigl(2\pi(r_2 - r_1)/\lambda\bigr)}{\sqrt{r_1 r_2}},$$

worin $r_{1,2}$ die exakten Abstände des Schirmpunkts zu beiden Quellen sind (die ersten beiden Terme sind die Einzelhelligkeiten, der dritte ist der Interferenzterm — er pendelt mit dem Gangunterschied $r_2 - r_1$ zwischen Verstärken und Löschen). **Erfolgskriterium:** Messprofil auf der Theorielinie (rms unter 5 %), und die 4/0/2-Zahlen aus der Refutation.

```
import os
os.environ["HWLOC_HIDE_ERRORS"] = "2"   # wie in Kap. 11–13

import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)

AUFL = 16
SX, SY = 24.0, 24.0
F = 1.0                 # λ0 = 1
```

```
X_Q, X_S = -6.0, 9.0    # Quellen- und Schirmposition
L = X_S - X_Q
SCHIRM = 22.0           # Länge der Schirmlinie
D2 = 4.0                # Quellabstand


def punktquellen(ys, amplituden=None):
    """Gleichgetaktete Ez-Punktquellen bei x = X_Q, Höhen ys."""
    return [mp.Source(mp.ContinuousSource(F, width=2), component=mp.Ez,
                      center=mp.Vector3(X_Q, y),
                      amplitude=1.0 if amplituden is None else amplituden[i])
            for i, y in enumerate(ys)]


def schirm_intensitaet(quellen, geometrie=(), vollbild=False):
    """Einschwingen, dann Ez² über eine Periode mitteln."""
    sim = mp.Simulation(cell_size=mp.Vector3(SX, SY), resolution=AUFL,
                        boundary_layers=[mp.PML(1.0)], sources=quellen,
                        geometry=list(geometrie))
    sim.run(until=45.0)
    acc = []
    def sammle(s):
        if vollbild:
            acc.append(s.get_array(component=mp.Ez, center=mp.Vector3(),
                                   size=mp.Vector3(SX, SY)))
        else:
            acc.append(s.get_array(component=mp.Ez, center=mp.Vector3(X_S, 0),
                                   size=mp.Vector3(0, SCHIRM)))
    sim.run(mp.at_every(1.0 / F / 16, sammle), until=1.0 / F)
    I = np.mean(np.square(acc), axis=0)
    if vollbild:
        return I
    return np.linspace(-SCHIRM / 2, SCHIRM / 2, len(I)), I


karte_zwei = schirm_intensitaet(punktquellen((-D2 / 2, D2 / 2)),
                                vollbild=True)
y, I_zwei = schirm_intensitaet(punktquellen((-D2 / 2, D2 / 2)))
print(f"eingeschwungen; Schirmprofil mit {len(y)} Punkten")
```

```
Using MPI version 4.1, 1 processes
eingeschwungen; Schirmprofil mit 354 Punkten
```

Vorhersage-Punkt

Bevor das Bild erscheint: Ist es auf der Mittellinie ($y = 0$, gleich weit von beiden Quellen) hell oder dunkel? Und auf wie viele helle Streifen insgesamt tippst du auf dem Schirm von $y = -9$ bis $+9$? (Die Handrechnung oben kennt die Antwort.)

Gleicher Weg heißt Gangunterschied null heißt: im Takt — die Mitte ist hell. Und da der Schirmrand bei $y = 9$ unter $\theta = 31°$ erscheint ($\sin\theta = 0{,}51$, also bis $m = \pm 2$): fünf Streifen. Das Bild dazu:

```
# von oben: karte_zwei, y, I_zwei, L, D2, SX, SY, X_S
r1 = np.hypot(L, y - D2 / 2)            # exakte Wege zu beiden Quellen
r2 = np.hypot(L, y + D2 / 2)
drei_term = (0.5 / r1 + 0.5 / r2
             + np.cos(2 * np.pi * F * (r2 - r1)) / np.sqrt(r1 * r2))

fig, achsen = plt.subplots(1, 2, figsize=(7.4, 3.4),
                           gridspec_kw={"width_ratios": [1.15, 1]})
ax = achsen[0]
ax.imshow(np.sqrt(karte_zwei).T, origin="lower", cmap="inferno",
          extent=(-SX / 2, SX / 2, -SY / 2, SY / 2))
ax.axvline(X_S, color="white", ls="--", lw=0.9)
ax.set_xlabel("x (λ)"); ax.set_ylabel("y (λ)")
ax.set_title("zwei Quellen, d = 4λ", fontsize=10)

ax = achsen[1]
fenster = np.abs(y) < 9
ax.plot(y, I_zwei / I_zwei[fenster].max(), "o", ms=2.6, label="Messung")
ax.plot(y, drei_term / drei_term[fenster].max(), "k-", lw=1.0,
        label="Theorie (3 Terme)")
ax.set_xlim(-9, 9); ax.set_ylim(0, 1.05)
ax.set_xlabel("y am Schirm (λ)"); ax.set_ylabel("I (normiert)")
ax.set_title("Schirmprofil", fontsize=10)
ax.legend(fontsize=8)
plt.tight_layout(); plt.show()
```

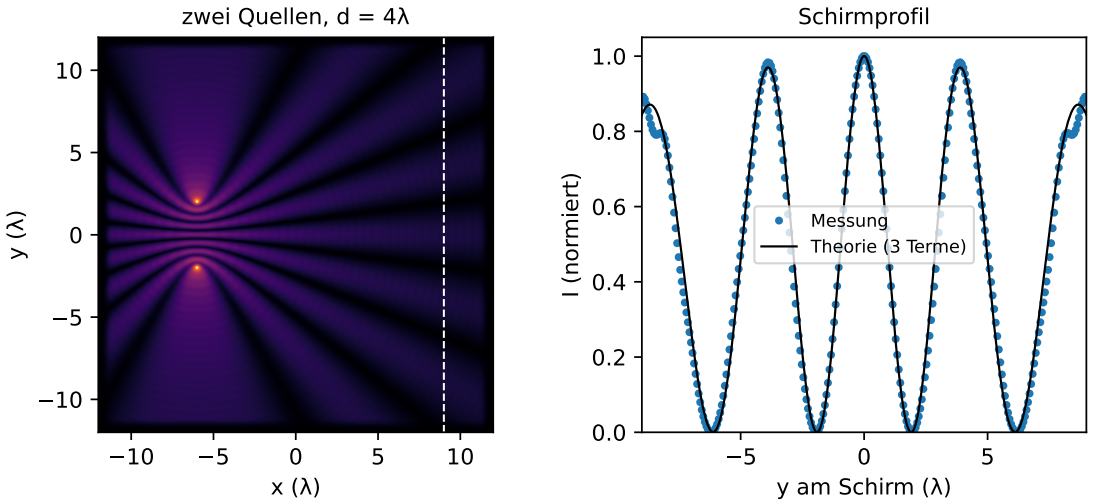


Abbildung 14.1: Links: die Intensität (Zeitmittel von Ez^2, Wurzel-Farbskala) zweier gleichphasiger Quellen — aus dem Überlappgebiet wachsen helle Strahlen, getrennt von dunklen Auslöschungs-Schneisen; die gestrichelte Linie ist der Schirm. Rechts: das Profil auf dem Schirm (Punkte) liegt auf der Drei-Term-Theorielinie mit den exakten Abständen r_1, r_2 und den $1/\sqrt{r}$-Amplituden aus Kapitel 9 (rms-Abweichung 3 %): fünf helle Streifen, die Mitte am hellsten, die Minima praktisch schwarz.

Fünf Streifen, Mitte hell, Messpunkte auf der Linie. Jetzt die drei Zahlen aus der Refutation — dafür braucht es noch den Referenzlauf mit *einer* Quelle:

```
# von oben: punktquellen(), schirm_intensitaet(), I_zwei, y, fenster
_, I_eine = schirm_intensitaet(punktquellen((0.0,)))
mitte = len(y) // 2
print(f"hellster Streifen : {I_zwei[mitte] / I_eine[mitte]:.2f}-mal"
      f" eine Quelle   (Theorie: 4)")
print(f"dunkelster Punkt  : {I_zwei.min() / I_zwei.max():.1e}"
      f" vom Maximum    (Theorie: 0)")
print(f"Schirm-Mittelwert : {I_zwei[fenster].mean() / I_eine[fenster].mean():.2f}-mal"
      f" eine Quelle   (Theorie: 2)")
```

```
hellster Streifen : 3.98-mal eine Quelle   (Theorie: 4)
dunkelster Punkt  : 3.9e-04 vom Maximum    (Theorie: 0)
Schirm-Mittelwert : 2.04-mal eine Quelle   (Theorie: 2)
```

Viermal, (praktisch) null, zweimal — die Energie der zweiten Quelle ist vollständig da, nur eben von den dunklen in die hellen Streifen geschoben. Wer übrigens grübelt, *wie* die Quellen das „wissen”: gar nicht. Jede sendet stur ihre Ringwelle; das Muster entsteht erst in der Addition am Messort.

14.3 Der Doppelspalt: zwei Quellen aus einer Wand

Historisch lief das Experiment andersherum — Thomas Young hatte 1801 keine zwei kohärenten Quellen, sondern *eine* Welle und eine Wand mit zwei Schlitzen. Dass das auf dasselbe hinausläuft, ist die Huygens-Idee aus der Kapitel-9-Übung: Jeder Spalt, klein gegen die Wellenlänge, wirkt als eigene Punktquelle — und beide werden von derselben ankommenden Welle angestoßen, sind also automatisch kohärent. Der Aufbau: dieselbe Bühne, aber statt der zwei Quellen eine **ebene Welle** (eine breite Linienquelle am linken Rand) und bei $x = -6$ eine PEC-Wand mit zwei Spalten der Breite $w = 0{,}8\lambda$ im Abstand $d = 4\lambda$ — gebaut mit einem Stanz-Trick: erst die durchgehende Metallwand, dann zwei Luftblöcke darüber (spätere Objekte überschreiben frühere). **Gemessen wird** wie eben; **Erwartung:** dasselbe Streifenmuster wie bei zwei Quellen. Ob das ganz aufgeht, zeigt der Vergleich:

```
# von oben: schirm_intensitaet(), y, fenster, drei_term, X_Q, D2, SY
W_SPALT, DICKE = 0.8, 0.4
# mp.metal: idealer Leiter (das PEC aus Kap. 6), mp.air: Luft/Vakuum
wand = [mp.Block(size=mp.Vector3(DICKE, mp.inf, mp.inf),
                 center=mp.Vector3(X_Q, 0), material=mp.metal),
        mp.Block(size=mp.Vector3(DICKE, W_SPALT, mp.inf),    # „ausstanzen":
                 center=mp.Vector3(X_Q, +D2 / 2), material=mp.air),
        mp.Block(size=mp.Vector3(DICKE, W_SPALT, mp.inf),    # Luft ersetzt
                 center=mp.Vector3(X_Q, -D2 / 2), material=mp.air)]  # Metall
ebene_welle = [mp.Source(mp.ContinuousSource(F, width=2), component=mp.Ez,
                         center=mp.Vector3(-10.5, 0),
                         size=mp.Vector3(0, SY - 2))]

_, I_spalt = schirm_intensitaet(ebene_welle, wand)
karte_spalt = schirm_intensitaet(ebene_welle, wand, vollbild=True)

d_naiv = np.abs(I_spalt / I_spalt[fenster].max()
                - drei_term / drei_term[fenster].max())[fenster]
print(f"Doppelspalt vs. Zweiquellen-Theorie:"
      f" max. Abweichung {d_naiv.max() * 100:.0f} %")
```

```
Doppelspalt vs. Zweiquellen-Theorie: max. Abweichung 42 %
```

Autsch — die Streifen sind da, aber bis zu 42 % daneben? Bevor jemand Meep verdächtigt: Querrechnen heißt der Differenz zuhören (Kapitel 11). Ein Blick auf das Profil verrät, *wo* sie wohnt — die **äußeren** Streifen sind zu schwach, und zwar systematisch mit dem Winkel. Das ist kein Fehler, sondern die Physik des einzelnen Spalts: $w = 0{,}8\lambda$ ist eben *nicht* klein gegen λ. Ein Spalt endlicher Breite ist selbst eine kleine Kette von Huygens-Quellen, deren Beiträge unter schrägen Winkeln gegeneinander laufen — er strahlt nach vorn am stärksten und zu den Seiten schwächer. Das Fernfeld dieser Kette ist die **Einzelspalt-Hüllkurve**

$$I_{\text{Spalt}}(\theta) \;\propto\; \left(\frac{\sin x}{x}\right)^2, \qquad x = \frac{\pi w \sin\theta}{\lambda},$$

mit der ersten Nullstelle bei $w \sin\theta = \lambda$ (die Mini-Kette löscht sich aus, wenn über die Spaltbreite genau eine Wellenlänge Gangunterschied liegt — bei uns erst bei $\sin\theta = 1{,}25$, also nie ganz: die Hülle drückt nur, sie löscht nicht). Das Doppelspalt-Muster ist das Produkt aus beidem: Zweiquellen-Streifen mal Hüllkurve.

```
# von oben: karte_spalt, I_spalt, y, fenster, drei_term, W_SPALT, L
sin_theta = y / np.hypot(L, y)
# np.sinc(x) = sin(πx)/(πx) — NumPy steckt das π selbst hinein
huelle = np.sinc(W_SPALT * sin_theta / 1.0) ** 2
theorie_spalt = drei_term * huelle

fig, achsen = plt.subplots(1, 2, figsize=(7.4, 3.4),
                           gridspec_kw={"width_ratios": [1.15, 1]})
ax = achsen[0]
ax.imshow(np.sqrt(karte_spalt).T, origin="lower", cmap="inferno",
          extent=(-SX / 2, SX / 2, -SY / 2, SY / 2))
ax.axvline(X_S, color="white", ls="--", lw=0.9)
ax.set_xlabel("x (λ)"); ax.set_ylabel("y (λ)")
ax.set_title("Doppelspalt (w = 0,8λ)", fontsize=10)

ax = achsen[1]
ax.plot(y, I_spalt / I_spalt[fenster].max(), "o", ms=2.6, label="Messung")
ax.plot(y, theorie_spalt / theorie_spalt[fenster].max(), "k-", lw=1.0,
        label="Zweiquellen × Hülle")
ax.plot(y, huelle / huelle.max(), ":", color="gray", lw=1.0,
        label="Hüllkurve")
ax.set_xlim(-9, 9); ax.set_ylim(0, 1.05)
ax.set_xlabel("y am Schirm (λ)"); ax.set_ylabel("I (normiert)")
ax.set_title("Profil mit Hüllkurve", fontsize=10)
```

```
ax.legend(fontsize=8)
plt.tight_layout(); plt.show()

d_mit = np.abs(I_spalt / I_spalt[fenster].max()
               - theorie_spalt / theorie_spalt[fenster].max())[fenster]
print(f"mit Hüllkurve: max {d_mit.max() * 100:.0f} %,"
      f" rms {np.sqrt((d_mit**2).mean()) * 100:.0f} %")
```

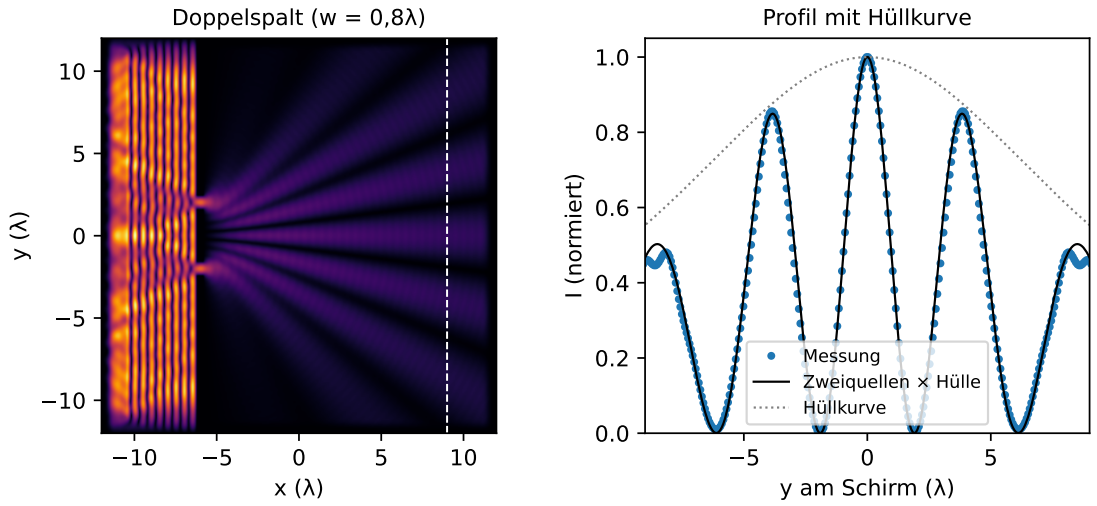


Abbildung 14.2: Links: der Doppelspalt im Intensitätsbild — die ebene Welle von links wird von der Wand (bei x = −6) bis auf zwei Spalte blockiert, dahinter fächern dieselben Strahlen auf wie bei zwei Punktquellen. Rechts: das Schirmprofil (Punkte) gegen die Zweiquellen-Theorie mal Einzelspalt-Hüllkurve (Linie; die Hüllkurve selbst gepunktet): Die Streifen LIEGEN exakt wie bei zwei Quellen, ihre Helligkeit folgt der Hülle — Abweichung nur noch 2 % rms statt 42 % ohne Hülle.

```
mit Hüllkurve: max 6 %, rms 2 %
```

Die 42 % sind auf 2 % rms zusammengefallen — die Hüllkurve war keine Ausrede, sondern messbare Notwendigkeit. Merkenswert ist die Arbeitsteilung: Der **Abstand** d der Spalte bestimmt, *wo* die Streifen liegen; die **Breite** w jedes Spalts bestimmt, *wie weit hinaus* sie hell bleiben.

14.4 Vom Doppelspalt zum Gitter

Was passiert mit mehr Spalten — sagen wir N gleichphasigen Quellen im Abstand d? Die hellen Streifen können nur dort bleiben, wo **alle** N Beiträge im Takt sind, und das sind exakt die alten Orte $d \sin\theta = m\lambda$ (wenn Nachbarn im Takt sind, sind es alle). Aber zwei Dinge ändern sich drastisch, und beide folgen aus einem Bild: Stelle die N Beiträge als N Uhrzeiger dar, deren Richtung die Ankunftsphase zeigt. Im Hauptmaximum zeigen alle in dieselbe Richtung — Gesamtamplitude N, Intensität N^2. Verlässt man das Maximum nur ein wenig, verdrehen sich die Zeiger gegeneinander; **vollständig dunkel** wird es, sobald sie gleichmäßig über den ganzen Kreis verteilt sind und sich zu null schließen — und das passiert schon, wenn zwischen Nachbarn der Gangunterschied λ/N liegt (der ganze Zeigerfächer spannt dann genau eine Umdrehung auf). Das Hauptmaximum endet also bei $\sin\theta = \lambda/(Nd)$ statt bei $\lambda/(2d)$: N**-mal schmaler**. Mehr Spalte machen die Streifen nicht zahlreicher, sondern schärfer und heller — aus dem sanften $\cos^2$ wird ein Kamm aus Nadeln.

Nur: Womit messen? Hier wartet eine Falle, in die wir beim Schreiben dieses Kapitels prompt selbst getappt sind. Acht Quellen im Abstand 2λ sind ein 14λ breites Instrument, und ein scharfer Beugungskamm bildet sich erst dort aus, wo alle N Teilwellen *als ebene Front* ankommen — die Faustformel verlangt Schirmabstände groß gegen $(\text{Breite})^2/\lambda \approx 200$. Unser Schirm bei $L = 15$ steht also mitten im Nahfeld-Gewühl; sein Profil zeigt dort schlicht nicht die Gitterformel (wer mag, probiere es — das Hauptmaximum *verbreitert* sich scheinbar mit N, weil die äußeren Quellen schon geometrisch weiter weg sind). Was wir bräuchten, ist ein unendlich ferner Schirm. Den hat Meep im Regal: `add_near2far` zeichnet die Felder auf einer Linie hinter den Quellen auf (frequenzaufgelöst — es ist derselbe DFT-Monitor aus Kapitel 11 in neuer Rolle) und rechnet daraus *nach* dem Lauf das Feld an beliebig fernen Punkten aus. Genau dieses Werkzeug vermisst in Teil VI die Strahlungsdiagramme von Antennen; hier ist seine Premiere. Der Aufbau der Gitterserie: $N = 2, 4, 8$ Punktquellen ($d = 2\lambda$), als **Puls** angeregt (für saubere DFT-Fernfelder, wie bei den Flux-Messungen seit Kapitel 12), Aufzeichnungslinie bei $x = -2$, ausgewertet auf einem Bogen im Abstand 1000. **Erfolgskriterium:** zentrale Höhe $\propto N^2$ (Faktor 4 je Verdopplung), Halbwertsbreite $\propto 1/N$ (Faktor 2).

```
# von oben: X_Q, SX, SY, AUFL
D_GITTER = 2.0
R_FERN = 1000.0


def fernfeld(n_quellen, f_mitte, f_breite, n_freq):
```

```
    """Fernfeld-Winkelprofile |Ez|² einer Reihe von Puls-Punktquellen."""
    ys = (np.arange(n_quellen) - (n_quellen - 1) / 2) * D_GITTER
    quellen = [mp.Source(mp.GaussianSource(1.1, fwidth=0.5), component=mp.Ez,
                         center=mp.Vector3(X_Q, yq)) for yq in ys]
    sim = mp.Simulation(cell_size=mp.Vector3(SX, SY), resolution=AUFL,
                        boundary_layers=[mp.PML(1.0)], sources=quellen)
    # add_near2far: zeichnet die Felder auf der Linie auf (DFT) und kann
    # daraus später das Feld an beliebig fernen Punkten ausrechnen;
    # Frequenzraster wie bei add_flux: n_freq Punkte um f_mitte ± f_breite/2
    nahfeld = sim.add_near2far(f_mitte, f_breite, n_freq,
                               mp.Near2FarRegion(center=mp.Vector3(-2.0, 0),
                                                 size=mp.Vector3(0, SY - 2)))
    sim.run(until_after_sources=60)
    winkel = np.linspace(-60, 60, 481)
    profile = np.zeros((n_freq, len(winkel)))
    for i, th in enumerate(np.radians(winkel)):
        punkt = mp.Vector3(X_Q + R_FERN * np.cos(th), R_FERN * np.sin(th))
        ff = sim.get_farfield(nahfeld, punkt)   # 6 Feldkomponenten je
        for k in range(n_freq):                 # Frequenz, Ez ist Nr. 2
            profile[k, i] = abs(ff[6 * k + 2]) ** 2
    return winkel, profile


gitter = {}
for n in (2, 4, 8):
    w, p = fernfeld(n, 1.0, 0, 1)
    gitter[n] = p[0]

fig, ax = plt.subplots(figsize=(6.8, 3.4))
for n, farbe in ((2, "tab:gray"), (4, "tab:orange"), (8, "tab:blue")):
    ax.plot(w, gitter[n] / gitter[8].max(), color=farbe, lw=1.2,
            label=f"N = {n}")
ax.set_xlim(-45, 45)
ax.set_xlabel("Winkel (°)"); ax.set_ylabel("I (auf N = 8 normiert)")
ax.legend(fontsize=9)
plt.tight_layout(); plt.show()
```

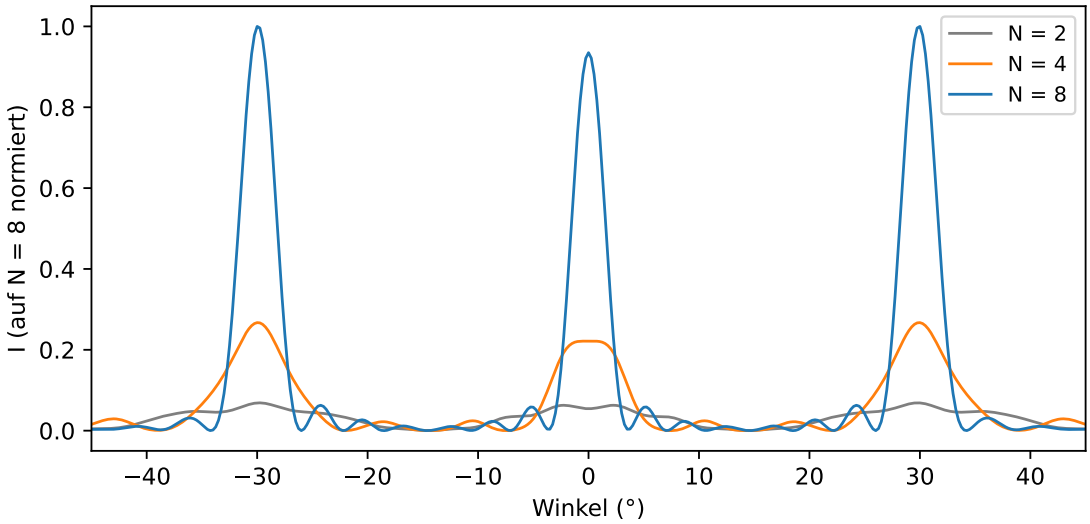


Abbildung 14.3: Das Fernfeld (near2far, Abstand 1000) der Quellreihen mit N = 2, 4 und 8 (d = 2λ), alle auf das N = 8-Maximum normiert: Die Hauptmaxima bleiben bei 0° und ±30° (d·sinθ = mλ) liegen, aber sie wachsen mit jeder Verdopplung um den Faktor vier und werden halb so breit — aus den sanften Zweiquellen-Hügeln (grau) wird der Nadelkamm des Gitters (blau). Zwischen den Nadeln liegen die schwachen Nebenmaxima der Zeiger-Zwischenstellungen.

Die Messzahlen zur Vorhersage:

```
# von oben: gitter, w
def halbwertsbreite(w, p):
    j = np.argmin(np.abs(w))
    halb = p[j] / 2
    rechts = j + np.argmax(p[j:] < halb)
    links = j - np.argmax(p[:j + 1][::-1] < halb)
    return w[rechts] - w[links]


h = {n: gitter[n][np.argmin(np.abs(w))] for n in (2, 4, 8)}
b = {n: halbwertsbreite(w, gitter[n]) for n in (2, 4, 8)}
print(f"zentrale Höhe   : N=2→4: ×{h[4]/h[2]:.2f},  N=4→8: ×{h[8]/h[4]:.2f}"
      f"   (Theorie: ×4)")
print(f"Halbwertsbreite : {b[2]:.1f}° / {b[4]:.1f}° / {b[8]:.1f}°"
      f"  → Verhältnisse {b[2]/b[4]:.2f} und {b[4]/b[8]:.2f}  (Theorie: ≈2)")
```

```
zentrale Höhe   : N=2→4: ×4.08,  N=4→8: ×4.22   (Theorie: ×4)
Halbwertsbreite : 17.0° / 7.5° / 3.5°  → Verhältnisse 2.27 und 2.14  (Theorie: ≈2)
```

N^2 und $1/N$, beide getroffen. Diese Schärfung ist der ganze Daseinszweck des Gitters: Eine Nadel, deren Position von λ abhängt und deren Breite mit jedem weiteren Spalt schrumpft, ist ein **Messgerät für Wellenlängen** — bei einem echten Gitter mit zehntausenden Rillen werden aus unseren Nadeln Skalpelle, die Spektrallinien auf sechs Stellen trennen.

14.5 Das Gitter als Spektrometer

Zeit für den Höhepunkt: zwei „Farben" gleichzeitig. Die Position der ersten Ordnung hängt über $\sin\theta_1 = \lambda/d$ von der Wellenlänge ab — verschiedene Farben landen auf verschiedenen Winkeln, das Gitter *ist* ein Spektrometer.

> Vorhersage-Punkt
>
> Wir schicken zwei Frequenzen durch das N=8-Gitter: $f = 1$ („rot", $\lambda = 1$) und $f = 1{,}25$ („blau", $\lambda = 0{,}8$). Welche Farbe wird **stärker** abgelenkt? Wer beim Glasprisma aufgepasst hat, tippe trotzdem nicht vorschnell — und rechne mit $\sin\theta_1 = \lambda/d$ kurz nach.

Beim Prisma gewinnt Blau (dazu mehr in Kapitel 16) — beim Gitter ist es umgekehrt: Die *größere* Wellenlänge braucht den größeren Winkel für denselben Gangunterschied, **Rot wird stärker abgelenkt** ($\sin\theta = 1/2 \to 30{,}0°$ gegen $0{,}8/2 \to 23{,}6°$). Und die Messung kostet dank Kapitel 10 nur *einen* Lauf: Der Anregungspuls enthält beide Frequenzen sowieso (schmal in der Zeit, breit im Spektrum), und der near2far-Monitor sortiert sie — wir bestellen einfach beide Frequenzen auf seinem Raster:

```python
# von oben: fernfeld(), D_GITTER
# Frequenzraster f_mitte ± f_breite/2 → genau (1,0 und 1,25)
w, p2 = fernfeld(8, 1.125, 0.25, 2)

fig, ax = plt.subplots(figsize=(6.8, 3.2))
for k, (f, farbe, name) in enumerate(((1.0, "tab:red", "f = 1 („rot“)"),
                                      (1.25, "tab:blue", "f = 1,25 („blau“)"))):
    ax.plot(w, p2[k] / p2[k].max(), color=farbe, lw=1.3, label=name)
    ax.axvline(np.degrees(np.arcsin((1 / f) / D_GITTER)), color=farbe,
               ls=":", lw=1.0)
    maske = (w > 10) & (w < 50)
    print(f"f = {f}: 1. Ordnung gemessen bei"
```

```
            f" {w[maske][np.argmax(p2[k][maske])]:.2f}°"
            f"  (arcsin(λ/d) = {np.degrees(np.arcsin((1/f)/D_GITTER)):.2f}°)")
ax.set_xlim(0, 50)
ax.set_xlabel("Winkel (°)"); ax.set_ylabel("I (je Farbe normiert)")
ax.legend(fontsize=9)
plt.tight_layout(); plt.show()
```

```
f = 1.0: 1. Ordnung gemessen bei 30.00°  (arcsin(λ/d) = 30.00°)
f = 1.25: 1. Ordnung gemessen bei 23.50°  (arcsin(λ/d) = 23.58°)
```

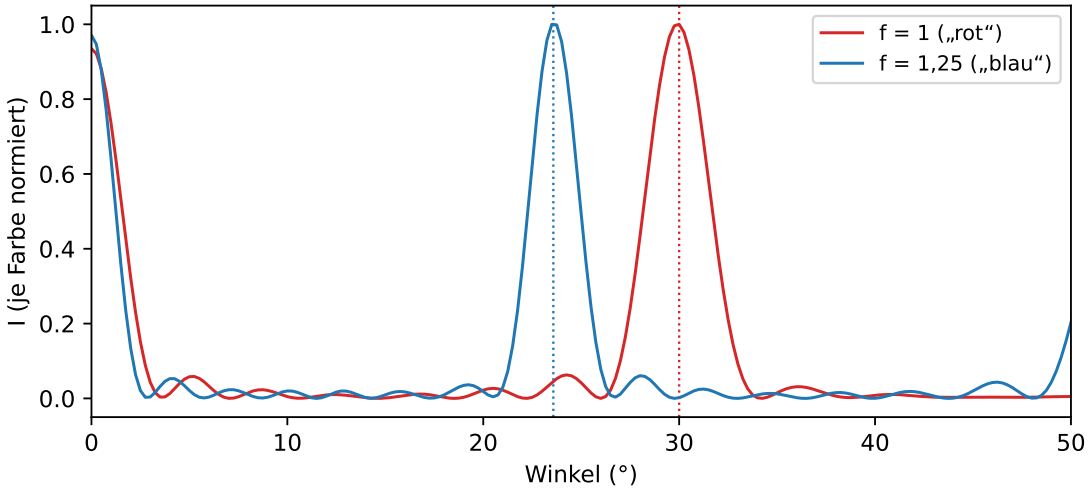


Abbildung 14.4: Ein einziger Pulslauf durch das N = 8-Gitter, zwei DFT-Fernfelder: Die erste Beugungsordnung von „Rot“ (f = 1, λ = 1) liegt bei 30,0°, die von „Blau“ (f = 1,25, λ = 0,8) bei 23,5° — jeweils auf der gepunkteten Theorielinie arcsin(λ/d). Das Gitter sortiert Farben, und zwar andersherum als ein Prisma: Die längere Wellenlänge wird stärker abgelenkt. (Profile je Farbe normiert; der gemeinsame Gipfel am linken Rand ist die nullte Ordnung, in der alle Farben zusammenfallen.)

Zwei Farben, zwei Nadeln, beide auf der Theorielinie — das CD-Rätsel vom Kapitelanfang ist damit Mechanik: Die Datenrillen sind ein Reflexionsgitter mit $d = 1{,}6$ µm, jede Lichtwellenlänge bekommt ihre eigene Richtung, und beim Kippen der Scheibe wandert ein anderer Teil des sortierten Regenbogens ins Auge (Ü 14.2 rechnet die Winkel aus). Astronomen treiben es nur präziser: Gitterspektrografen zerlegen Sternenlicht so fein, dass chemische

Fingerabdrücke und Dopplerverschiebungen ablesbar werden — alles Anwendungen von $d \sin\theta = m\lambda$.

14.6 Das Kapitel-Programm

`programme/kap14/kap14_interferenz_gitter.py` bündelt alle Befunde eigenständig, mit `assert`-Schranken: Zweiquellen-Profil auf der Drei-Term-Theorie (rms < 5 %), die 4/0/2-Zahlen, Doppelspalt ohne Hülle > 20 % daneben und mit Hülle < 10 % (rms < 4 %), Gitterhöhen $\propto N^2$ und Breiten $\propto 1/N$ in ±15-%-Fenstern, Spektrometer-Winkel auf 1°, dazu die Übungs-Befunde (gegenphasig, $d < \lambda$).

Merkkasten

- **Felder addieren sich, nicht Helligkeiten:** hell ist, wo der Gangunterschied $d \sin\theta$ ein Vielfaches von λ ist — dort viermal so hell wie eine Quelle; dazwischen dunkel; im Mittel zweimal. Interferenz verteilt um, sie erschafft nichts.
- **Spalte sind Quellen** (Huygens): Der Spalt-*Abstand* setzt die Streifen-Orte, die Spalt-*Breite* die Hüllkurve ($\sin x/x^2$, erste Null bei $w \sin\theta = \lambda$).
- **Gitter = Schärfemaschine:** N Spalte machen die Maxima N^2-fach hell und N-fach schmal (Zeigerbild: dunkel, sobald die N Zeiger den Kreis schließen). Orte bleiben $d \sin\theta = m\lambda$.
- **Gitter sortieren Farben** — Rot stärker als Blau (umgekehrt zum Prisma). Eine CD ist ein Gitter mit $d = 1{,}6$ µm.
- **Schirme können zu nah sein:** Beugungsmuster gelten im Fernfeld ($L \gg \text{Breite}^2/\lambda$); Meeps `near2far` liefert den unendlich fernen Schirm aus einem Nahfeld-Lauf.

Roter Faden

Die Superposition aus **Kapitel 1** hat hier ihren großen Auftritt gehabt, choreografiert von der Gangunterschieds-Geometrie aus **Kapitel 12** und vermessen mit dem Intensitätsbegriff aus **Kapitel 4**; die $1/\sqrt{r}$-Amplituden (**Kapitel 9**) steckten in der Theorielinie, die Huygens-Übung von dort wurde zum Doppelspalt befördert, und „schmal in der Zeit, breit im Spektrum" (**Kapitel 10**) lieferte beide Spektrometer-Farben in einem Puls. Nach vorn: **Kapitel 16** stellt das Gitter dem Prisma gegenüber (Dispersion — dort klärt sich, warum das Prisma andersherum sortiert), **Kapitel 21/23** übernehmen

`near2far` und bauen aus unserer Quellenreihe ein **Antennen-Array** — dass ein Phased Array und ein Beugungsgitter dasselbe Rechenwerk sind, ist dann keine Überraschung mehr —, und **Kapitel 26** lässt Gitter aus ganzen Kristallebenen interferieren (photonische Bandlücken).

Übungen

Ü 14.1 (Verstehen). Zwei Taschenlampen nebeneinander werfen keine Interferenzstreifen an die Wand — zwei Lautsprecher mit demselben Sinussignal erzeugen aber sehr wohl hörbare „taube Stellen" im Raum. Erkläre den Unterschied mit dem Begriff der Kohärenz. (Denkhilfe: Was müsste zwischen den Feldern beider Quellen über die ganze Messdauer konstant bleiben, damit sich ein *ortsfestes* Muster ausbildet — und was macht das Licht einer Glühwendel alle paar Nanosekunden?)

Musterlösung zu Ü 14.1

Ein ortsfestes Streifenmuster braucht einen **festen Phasenbezug**: Der Gangunterschied entscheidet nur dann dauerhaft über hell und dunkel, wenn die beiden Quellen ihren Takt zueinander nie wechseln. Unsere Meep-Quellen (und die zwei Lautsprecher am selben Generator) sind perfekt kohärent — ein Muster, für immer. Eine Glühwendel dagegen sendet das Gemeinschaftswerk unzähliger unabhängiger Atome, deren Phasenlage sich nach jedem Emissionsakt — alle paar Nanosekunden — zufällig neu würfelt; zwei *getrennte* Lampen würfeln zudem unabhängig. Das Streifenmuster existiert in jedem Nanosekunden-Augenblick durchaus, springt aber milliardenfach pro Sekunde an neue Orte; Auge und Kamera sehen nur den Mittelwert, und der ist strukturlos „zweimal so hell" (das Schirm-Mittel aus der Refutation!). Young löste das Problem 1801 genial sparsam: *eine* Quelle, per Doppelspalt in zwei Kopien geteilt — Kopien desselben Würfelwurfs sind zueinander immer kohärent.

Ü 14.2 (Verstehen). Die Datenrillen einer CD liegen $1{,}6$ µm auseinander. Berechne von Hand (und prüfe mit drei Zeilen Code) die Winkel der ersten Beugungsordnung für blaues (450 nm), grünes (532 nm) und rotes (650 nm) Licht — bei senkrechtem Lichteinfall. Welche Farbe liegt außen im CD-Regenbogen? Und warum zeigt eine DVD ($d = 0{,}74$ µm) ihre Farben unter steileren Winkeln?

Musterlösung zu Ü 14.2

$\sin\theta_1 = \lambda/d$: blau $0{,}281 \to 16{,}3°$, grün $0{,}333 \to 19{,}4°$, rot $0{,}406 \to 24{,}0°$ — Rot außen, wie bei unserem Zwei-Farben-Experiment. Die DVD packt ihre Spuren mehr als doppelt so eng; kleineres d heißt größeres λ/d, also wandern alle Ordnungen nach außen (rot: $\sin\theta = 0{,}88 \to 61{,}5°$), und die zweite Ordnung passt gar nicht mehr hinein.

```
# von oben: nichts (eigenständige Handrechnungs-Kontrolle)
for d_um in (1.6, 0.74):
    winkel = [np.degrees(np.arcsin(lam / (d_um * 1000)))
              for lam in (450, 532, 650)]
    print(f"d = {d_um} µm:  blau {winkel[0]:.1f}°,"
          f" grün {winkel[1]:.1f}°, rot {winkel[2]:.1f}°")
```

```
d = 1.6 µm:  blau 16.3°, grün 19.4°, rot 24.0°
d = 0.74 µm:  blau 37.5°, grün 46.0°, rot 61.4°
```

Ü 14.3 (Verändern). Drehe eine der beiden Quellen aus dem ersten Experiment um — `amplituden=(1.0, -1.0)` lässt sie im Gegentakt schwingen. Sage vorher: Was wird aus dem hellen Streifen in der Mitte, und wohin rücken die Maxima? Miss nach.

Musterlösung zu Ü 14.3

Der eingebaute Gegentakt wirkt wie ein zusätzlicher halber Wellenlängen-Gangunterschied: Überall, wo es hell war, wird es dunkel und umgekehrt — die Mitte (Gangunterschied null, Gegentakt bleibt Gegentakt) wird zum **dunklen** Streifen, die Maxima rücken auf die alten Minima bei $d\sin\theta = (m + \frac{1}{2})\lambda$:

```
# von oben: punktquellen(), schirm_intensitaet(), I_zwei, y, mitte
_, I_gegen = schirm_intensitaet(punktquellen((-D2 / 2, D2 / 2),
                                             amplituden=(1.0, -1.0)))
print(f"Mitte: {I_gegen[mitte] / I_zwei[mitte]:.1e} der gleichphasigen")
print(f"hellster Streifen jetzt bei y = {y[np.argmax(I_gegen)]:.1f}"
      f"   (dort lag vorher das erste Minimum, y ≈ ±1,9)")
```

```
Mitte: 6.0e-04 der gleichphasigen
hellster Streifen jetzt bei y = -2.0   (dort lag vorher das erste Minimum, y ≈ ±1,9)
```

Das Muster ist exakt invertiert — die Phase der Quellen ist ein ebenso scharfes Steuer wie ihre Position. Genau dieser Hebel (Phasen statt

Musterlösung zu Ü 14.4

Ü 14.4 (Übertragen). Rücke die beiden Quellen auf $d = 0{,}8\lambda$ zusammen — das gibt es nicht. Es bleibt allein die nullte Ordnung: **ein** breiter Hügel, nirgends mehr vollständige Auslöschung (der Gangunterschied erreicht $\lambda/2$ erst jenseits des Schirmrandes!) lenkt in Kapitel 23 die Keule eines Phased-Array-Radars, ganz ohne bewegliche Teile.

```
# von oben: punktquellen(), schirm_intensitaet(), y, fenster, I_zwei, SCHIRM
def maxima_zaehlen(I):
    """Lokale Maxima über 30 % des Größten (Mindestabstand 1 λ)."""
    schritt = int(len(I) / SCHIRM)
    gefunden = []
    for i in range(schritt, len(I) - schritt):
        if abs(y[i]) < 9 and I[i] > 0.3 * I[fenster].max() \
                and I[i] == I[i - schritt:i + schritt + 1].max() \
                and (not gefunden or i - gefunden[-1] >= schritt):
            gefunden.append(i)
    return len(gefunden)


_, I_eng = schirm_intensitaet(punktquellen((-0.4, 0.4)))
print(f"helle Streifen: d = 0,8λ → {maxima_zaehlen(I_eng)},"
      f"   d = 4λ → {maxima_zaehlen(I_zwei)}")
```

```
helle Streifen: d = 0,8λ → 1,   d = 4λ → 5
```

Für Antennen-Arrays (Kapitel 23) ist das die halbe Miete: Dort *will* man genau eine Hauptkeule und keine Kopien davon in andere Richtungen (sie hießen dort „Grating Lobes" — wörtlich: Gitterkeulen). Die Designregel lautet deshalb: Elementabstand unter einer Wellenlänge halten — dieselbe Mathematik, die hier die Streifen abschaltet, schaltet dort die Geisterkeulen ab.

Das Kleingedruckte

Unsere „Farben” $f = 1$ und $f = 1{,}25$ liegen eine Viertel Oktave auseinander; echtes Weißlicht füllt das Intervall kontinuierlich, und jede Ordnung wird zum durchgehenden Spektralband (deshalb überlappen sich bei Gittern die hohen Ordnungen: das Rot der zweiten erreicht das Blau der dritten). Die Fraunhofer-Bedingung $L \gg B^2/\lambda$ verdient ihren eigenen Respekt: Für ein 10-cm-Gitter im sichtbaren Licht liegt der „ferne” Schirm jenseits von zwanzig Kilometern — reale Spektrometer falten den Weg deshalb mit einer Sammellinse, die das Fernfeld in ihre Brennebene holt (optisch exakt das, was `near2far` numerisch tut). Und wer den Doppelspalt mit *einzelnen Photonen* betreibt, betritt das vielleicht tiefste Experiment der Physik — Kapitel 30 wagt den Blick.

15 Resonanz und Güte Q

Fahre mit einem feuchten Finger über den Rand eines Weinglases, bis es singt, und höre dann zu: Der Ton steht noch sekundenlang im Raum. Drei Sekunden Nachklang bei tausend Schwingungen pro Sekunde — das Glas schwingt **dreitausendmal**, bevor der Ton merklich verebbt. Ein Plastikbecher schafft keine einzige hörbare Schwingung. Offenbar gibt es eine Kennzahl, die „gute" von „schlechten" Schwingern trennt — wie viele Schwingungen ein Resonator durchhält, bevor seine Energie zur Neige geht. Sie heißt **Güte** und trägt den Buchstaben Q.

Dieses Kapitel schuldet das Buch seinen Spektren: In Kapitel 10 standen die Resonanzen als Peaks im Kamm, und der Rote Faden versprach, dass diese Peaks „endlich eine Breite bekommen — und die erzählt von Verlusten". Genau das passiert jetzt. Wir bauen den Kapitel-10-Resonator in die Fläche um (sein Kamm bekommt zwei Indizes), geben ihm dann Verluste und messen Q auf **drei unabhängige Arten** — am Abklingen, an der Linienbreite und mit einem neuen Meep-Messgerät namens Harminv, das in Kapitel 11 schon angekündigt war. Alle drei müssen dasselbe sagen, und es gibt eine exakte Theoriezahl, an der sich alle messen lassen müssen.

Lernziele

Nach diesem Kapitel kannst du …

1. … die Eigenfrequenzen eines rechteckigen Hohlraums berechnen (f_{mn}-Formel) und seine Modenbilder lesen (Bäuche und Knotenlinien),
2. … die Güte Q definieren (Energie pro Verlustrate) und das exponentielle Ausschwingen $e^{-\omega t/2Q}$ herleiten,
3. … Q auf drei Arten messen — Abkling-Fit, Linienbreite $\Delta f = f/Q$, Harminv — und erklären, warum alle drei dasselbe messen,
4. … begründen, warum eine endlich lange Schwingung *keine* beliebig scharfe Frequenz haben kann (die Kapitel-10-Reziprozität, quantitativ),
5. … Strahlungsverluste einschätzen: warum ein Loch in der Wand Q um Größenordnungen senkt — und warum die Gittertür des Mikrowellenherds trotzdem dicht ist.

15.1 Der Kamm bekommt zwei Indizes

Der Resonator aus Kapitel 10 war eine Strecke zwischen zwei Spiegeln: Hineinpassen musste eine ganze Zahl von Halbwellen, $f_m = m{\cdot}c/2L$. In der Fläche gilt dasselbe Spiel zweimal — ein rechteckiger Kasten der Kantenlängen $L_x \times L_y$ mit Metallwänden verlangt stehende Wellen in *beiden* Richtungen: $k_x = m\pi/L_x$ **und** $k_y = n\pi/L_y$. Den Rest erledigt der Satz des Pythagoras im k-Raum ($|k|^2 = k_x^2 + k_y^2$, Kapitel 9) zusammen mit $f = c|k|/2\pi$:

$$f_{mn} \;=\; \frac{c}{2}\sqrt{\left(\frac{m}{L_x}\right)^2 + \left(\frac{n}{L_y}\right)^2}, \qquad m, n = 1, 2, 3, \ldots$$

Die Handrechnung für unsere Bühne, einen 4×3-Kasten (Meep-Einheiten, $c = 1$): Die Grundmode TM_{11} liegt bei $f_{11} = \frac{1}{2}\sqrt{1/16 + 1/9} = 0{,}2083$, darüber $\mathrm{TM}_{21} = 0{,}3005$, $\mathrm{TM}_{12} = 0{,}3560$ und $\mathrm{TM}_{22} = 2f_{11} = 0{,}4167$ — kein gleichmäßiger Kamm mehr wie in 1D, sondern ein unregelmäßiges Linienmuster, weil zwei Zahlenpaare hineinregieren.

Das Experiment dazu kennt seine Teile längst: **Bühne** — eine Meep-Zelle, die *nur* aus dem 4×3-Kasten besteht; ganz ohne PML sind Meeps Zellränder PEC-Metallwände (Kapitel 11), der Kasten baut sich also von selbst. Auflösung 24. **Anregung:** der Glockenschlag aus Kapitel 10 — ein Gauß-Puls (um $f = 0{,}3$, Bandbreite $0{,}36$) an einem absichtlich unsymmetrischen Punkt $(0{,}7,\ 0{,}45)$, damit keine Mode auf ihrem Knoten erwischt wird und stumm bleibt (die Lektion aus Ü 10.1). **Gemessen wird** an einer ebenso unsymmetrischen Sonde $(-0{,}6, -0{,}8)$: jede Zeitschritt-Probe über 2000 Zeiteinheiten, dann `rfft` — Werkzeugkette wörtlich aus Kapitel 10. **Erfolgskriterium:** die vier berechneten Linien auf einem Promille.

```
import os
os.environ["HWLOC_HIDE_ERRORS"] = "2"   # wie in Kap. 11–14

import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)

LX, LY = 4.0, 3.0
AUFL = 24
DT = 0.5 / AUFL                # Meeps Zeitschritt (Courant 0,5, Kap. 11)
FCEN, DF = 0.30, 0.36
```

```
QUELLE = mp.Vector3(0.7, 0.45)
SONDE = mp.Vector3(-0.6, -0.8)


def f_theorie(m, n):
    return 0.5 * np.sqrt((m / LX) ** 2 + (n / LY) ** 2)


def kasten(sigma_d=0.0):
    """Der 4×3-PEC-Hohlraum; optional verlustbehaftet gefüllt."""
    geometrie = []
    if sigma_d > 0:
        # mp.Medium(D_conductivity=σ): Material mit Stromverlusten –
        # das σ aus Kapitel 6, in Meeps Einheiten
        geometrie = [mp.Block(size=mp.Vector3(mp.inf, mp.inf, mp.inf),
                              material=mp.Medium(D_conductivity=sigma_d))]
    quelle = [mp.Source(mp.GaussianSource(FCEN, fwidth=DF), component=mp.Ez,
                        center=QUELLE)]
    return mp.Simulation(cell_size=mp.Vector3(LX, LY), resolution=AUFL,
                         sources=quelle, geometry=geometrie)


def sondenlauf(sigma_d, dauer):
    """Puls anregen, Ez an der Sonde jeden Zeitschritt aufzeichnen."""
    sim = kasten(sigma_d)
    reihe = []
    sim.run(mp.at_every(DT, lambda s: reihe.append(
        s.get_field_point(mp.Ez, SONDE).real)), until=dauer)
    return np.array(reihe)


reihe0 = sondenlauf(0.0, 2000.0)
print(f"{len(reihe0)} Messwerte aufgenommen")
```

```
Using MPI version 4.1, 1 processes
96000 Messwerte aufgenommen
```

```
# von oben: reihe0, DT, f_theorie()
spektrum0 = np.abs(np.fft.rfft(reihe0))
fr = np.fft.rfftfreq(len(reihe0), DT)

MODEN = ((1, 1), (2, 1), (1, 2), (2, 2))
```

```
fig, ax = plt.subplots(figsize=(6.6, 3.0))
zeige = (fr > 0.15) & (fr < 0.45)
ax.plot(fr[zeige], spektrum0[zeige] / spektrum0[zeige].max(), lw=1.0)
for m, n in MODEN:
    ax.axvline(f_theorie(m, n), color="gray", ls=":", lw=0.9)
    ax.annotate(f"TM{m}{n}", (f_theorie(m, n), 1.03), ha="center", fontsize=8)
ax.set_ylim(0, 1.12)
ax.set_xlabel("Frequenz (c/a)"); ax.set_ylabel("|Spektrum| (normiert)")
plt.tight_layout(); plt.show()

for m, n in MODEN:
    ft = f_theorie(m, n)
    maske = (fr > ft - 0.01) & (fr < ft + 0.01)
    fp = fr[maske][np.argmax(spektrum0[maske])]
    print(f"TM{m}{n}: gemessen {fp:.4f} | Theorie {ft:.4f}"
          f"  ({(fp - ft) / ft * 100:+.2f} %)")
```

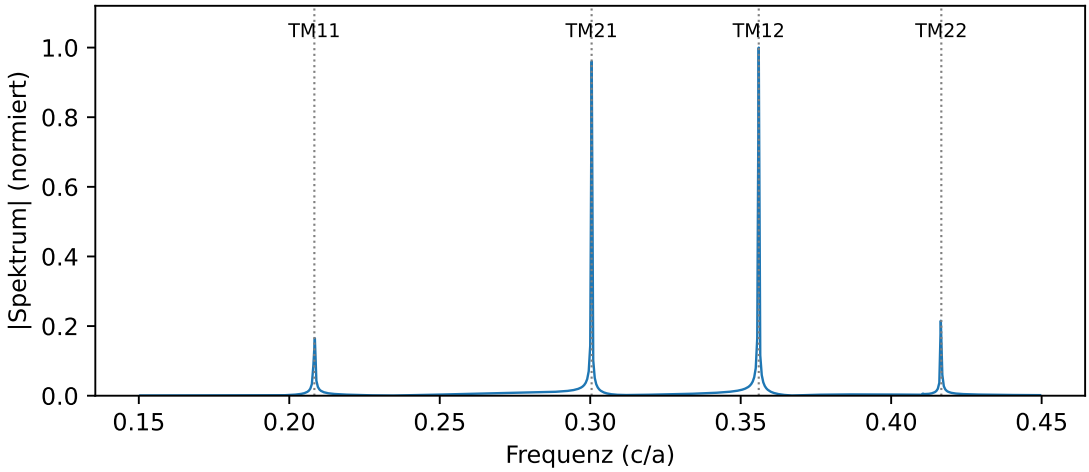


Abbildung 15.1: Das Sondenspektrum des verlustfreien 4×3-Hohlraums nach einem einzigen Puls (Werkzeugkette aus Kapitel 10): Die Resonanzen stehen exakt auf den gepunkteten Theorielinien der f(m,n)-Formel — ein unregelmäßiger 2D-Kamm statt der gleichmäßigen 1D-Leiter aus Kapitel 10, weil jetzt zwei Modenzahlen mitspielen. Die Linien sind nadelscharf: Ohne Verluste klingt hier nichts ab, und die Breite ist allein durch die Aufnahmedauer begrenzt.

```
TM11: gemessen 0.2085 | Theorie 0.2083  (+0.08 %)
TM21: gemessen 0.3005 | Theorie 0.3005  (+0.01 %)
```

```
TM12: gemessen 0.3560 | Theorie 0.3560  (-0.00 %)
TM22: gemessen 0.4165 | Theorie 0.4167  (-0.04 %)
```

Und so sehen die Schwingungsmuster hinter den Linien aus — die DFT-Felder (Kapitel 11) bei den vier Eigenfrequenzen, aus *einem* weiteren Pulslauf:

```
# von oben: kasten(), f_theorie(), MODEN, LX, LY
sim = kasten()
dfts = [sim.add_dft_fields([mp.Ez], [f_theorie(m, n)],
                           center=mp.Vector3(), size=mp.Vector3(LX, LY))
        for m, n in MODEN]
sim.run(until=2000.0)

fig, achsen = plt.subplots(2, 2, figsize=(7.0, 5.0))
for ax, (m, n), dft in zip(achsen.flat, MODEN, dfts):
    feld = np.real(sim.get_dft_array(dft, mp.Ez, 0))
    w = np.abs(feld).max()
    ax.imshow(feld.T, origin="lower", cmap="RdBu_r", vmin=-w, vmax=w,
              extent=(-LX / 2, LX / 2, -LY / 2, LY / 2))
    ax.set_title(f"TM{m}{n}   f = {f_theorie(m, n):.3f}", fontsize=9)
    ax.set_xlabel("x"); ax.set_ylabel("y")
plt.tight_layout(); plt.show()
```

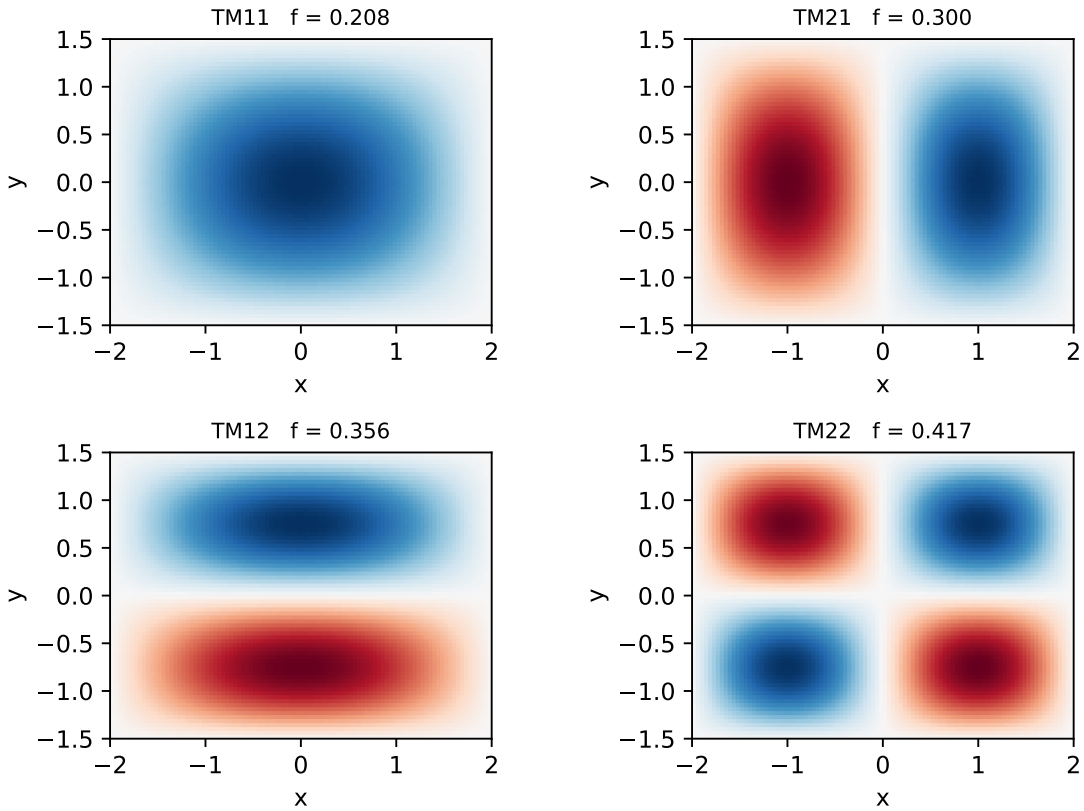


Abbildung 15.2: Die Feldmuster der vier tiefsten Moden (DFT-Felder bei den Eigenfrequenzen, Ez als Farbe): TM-mn hat m Bäuche in x und n Bäuche in y, getrennt von Knotenlinien, auf denen das Feld dauerhaft null bleibt — die Gitarrensaiten-Muster aus Kapitel 10, in die Fläche übersetzt. Die Knotenlinien sind keine Deko: Wo eine Mode einen Knoten hat, ist sie taub und stumm (Ü 10.1) — Übung 15.4 macht daraus ein Verlust-Experiment.

15.2 Die Güte: Schwingungen pro Energievorrat

Jetzt bekommt der Resonator Verluste — und wir brauchen die Kennzahl vom Kapitelanfang. Die **Güte** vergleicht, was ein Resonator *hat*, mit dem, was er *verliert*:

$$Q \;=\; \omega \cdot \frac{\text{gespeicherte Energie } W}{\text{Verlustleistung } P},$$

mit $\omega = 2\pi f$. Die Schreibweise mit ω macht Q dimensionslos und gibt ihr die anschauliche Lesart: Q ist (bis auf den Faktor 2π) die **Zahl der Schwingungen, nach der der Energievorrat zur Neige geht**. Denn aus der Definition folgt sofort eine Differentialgleichung — die Energie verliert pro Zeit $P = \omega W/Q$, also $W(t) = W_0\, e^{-\omega t/Q}$, und die Feldamplitude (Energie $\propto$ Amplitude², Kapitel 5) klingt mit der halben Rate:

$$E(t) \;\propto\; e^{-\omega t/2Q} \cos(2\pi f t).$$

Das Weinglas vom Anfang: drei Sekunden bis $1/e$ bei 1 kHz sind $Q = \pi f \tau \approx 10^4$ (Ü 15.1 rechnet nach).

Als Verlustquelle nehmen wir zunächst die ehrlichste: ein schwach leitfähiges Material im ganzen Kasten — das σ aus Kapitel 6, das dem E-Feld Ströme entzieht. In Meep heißt es `D_conductivity`, und seine Konvention messen wir gleich selbst nach, statt sie zu glauben. Die Theorie dazu ist ein Einzeiler: Wenn die Amplitude mit der Rate $\sigma_D/2$ fällt (das wird die Messung zeigen), dann ist $\omega/2Q = \sigma_D/2$, also

$$Q \;=\; \frac{\omega}{\sigma_D} \;=\; \frac{2\pi f}{\sigma_D}$$

— eine exakte Zielscheibe für alles, was folgt. Für $\sigma_D = 0{,}02$ und die TM_{11} ($f = 0{,}2083$): $Q = 65{,}4$.

15.3 Eine Linie kann nicht scharf bleiben

⚠ Naheliegende Vermutung

„Verluste machen die Schwingung leiser, aber nicht anders: Die Resonanzfrequenz bleibt ein scharfer, exakter Wert — man muss nur genau genug messen.”
Warum sie naheliegt: Die Frequenz steckt doch in der Geometrie (f_{mn}-Formel), und an der ändert ein bisschen Dämpfung nichts. Eine gedämpfte Gitarrensaite klingt ja auch nicht *falsch*, nur kürzer.
Was stattdessen stimmt: Eine Schwingung, die nach endlicher Zeit verstummt, **hat keine exakt scharfe Frequenz** — nicht aus Messfehlergründen, sondern grundsätzlich: Kapitel 10 hat gezeigt, dass kurz in der Zeit breit im Spektrum heißt, und ein abklingender Ton ist „kurz in der Zeit”. Seine Spektrallinie trägt eine echte, untilgbare Breite

$$\Delta f = \frac{f}{Q}$$

(Halbwertsbreite der Leistung). Je verlustärmer, desto schärfer — unendlich scharf wäre erst die unendlich lange Schwingung. Diese Breite ist keine Schwäche, sondern eine **Messgröße**: Wer die Linienbreite abliest, kennt die Verluste, ohne je das Abklingen gesehen zu haben.

Bevor wir das an Meep prüfen, eichen wir die Messroutine nach Kapitel-10-Sitte an einem Signal, dessen Wahrheit wir kennen — einer künstlichen gedämpften Schwingung mit exakt den Werten, die das Experiment gleich liefern soll ($f = 0{,}2083$, Abklingzeit der Amplitude $\tau = 2/\sigma_D = 100$, also Soll-$Q = \pi f \tau = 65{,}4$):

```
# von oben: DT (Zeitraster wie im Experiment)
def fwhm_guete(reihe, dt, f_nahe):
    """Q aus der Halbwertsbreite des Leistungs-Peaks nahe f_nahe."""
    leistung = np.abs(np.fft.rfft(reihe)) ** 2
    fr = np.fft.rfftfreq(len(reihe), dt)
    idx = np.flatnonzero((fr > f_nahe - 0.02) & (fr < f_nahe + 0.02))
    i0 = idx[np.argmax(leistung[idx])]
    halb = leistung[i0] / 2
    r = i0
    while leistung[r] > halb:                # rechte Halbwerts-Flanke …
        r += 1
    l = i0
    while leistung[l] > halb:                # … und linke
        l -= 1
    fr_r = fr[r-1] + (fr[r] - fr[r-1]) * (leistung[r-1] - halb) \
        / (leistung[r-1] - leistung[r])      # linear interpoliert
    fr_l = fr[l+1] - (fr[l+1] - fr[l]) * (leistung[l+1] - halb) \
        / (leistung[l+1] - leistung[l])
    return fr[i0] / (fr_r - fr_l)


t_test = np.arange(int(1200 / DT)) * DT
kunst = np.exp(-t_test / 100) * np.sin(2 * np.pi * 0.2083 * t_test)
print(f"Eichung am künstlichen Signal: Q = {fwhm_guete(kunst, DT, 0.2083):.1f}"
      f"   (Soll: π·f·τ = {np.pi * 0.2083 * 100:.1f})")
```

```
Eichung am künstlichen Signal: Q = 64.9   (Soll: π·f·τ = 65.4)
```

Die Routine misst richtig. Jetzt das echte Experiment.

15.4 Q, dreifach gemessen

Der Aufbau: derselbe Kasten, gefüllt mit $\sigma_D = 0{,}02$, derselbe Puls, dieselbe Sonde, 1200 Zeiteinheiten Aufnahme. **Gemessen wird dreimal unabhängig:** (1) im Zeitbereich — die Einhüllende des abklingenden Signals (Blockmaxima, wie die Einhüllenden in Kapitel 6) auf der log-Skala linear gefittet; (2) im Frequenzbereich — die geeichte Halbwertsbreiten-Routine; (3) mit **Harminv**, Meeps eingebautem Modenanalysator: Er zerlegt eine *kurze* abklingende Zeitreihe direkt in gedämpfte Sinusschwingungen und liefert für jede gefundene Mode Frequenz, Q und Amplitude — gewissermaßen ein Spektrometer für Ungeduldige, das statt langer FFT-Aufnahmen ein Modell anpasst (300 Zeiteinheiten genügen ihm). **Erfolgskriterium:** Alle drei treffen die Theoriezahl $2\pi f/\sigma_D = 65{,}4$ auf wenige Prozent — drei Messgeräte, die einander nie gesehen haben.

```
# von oben: sondenlauf(), DT, f_theorie(), fwhm_guete()
SIGMA = 0.02
reihe = sondenlauf(SIGMA, 1200.0)
t = np.arange(len(reihe)) * DT

blk = int(10.0 / DT)                          # Einhüllende: Blockmaxima
n_blk = len(reihe) // blk
env_t = np.array([t[i * blk + np.argmax(np.abs(reihe[i*blk:(i+1)*blk]))]
                  for i in range(n_blk)])
env_a = np.array([np.abs(reihe[i*blk:(i+1)*blk]).max() for i in range(n_blk)])
fit = (env_t > 100) & (env_t < 900)           # Anfang (Puls) und Ende
rate = -np.polyfit(env_t[fit], np.log(env_a[fit]), 1)[0]   # (Rauschen) meiden
f11 = f_theorie(1, 1)
q_zeit = np.pi * f11 / rate                   # Amplitude ~ e^(−ωt/2Q)
q_breite = fwhm_guete(reihe, DT, f11)

fig, achsen = plt.subplots(1, 2, figsize=(7.4, 3.2))
ax = achsen[0]
ax.plot(t[::40], np.abs(reihe[::40]), lw=0.3, color="tab:blue")
ax.plot(env_t, env_a, "o", ms=2.5, color="tab:orange")
ax.plot(env_t, env_a[0] * np.exp(-SIGMA / 2 * (env_t - env_t[0])), "k--",
        lw=1.0, label="Theorie $e^{-\\sigma t/2}$")
ax.set_yscale("log")
ax.set_xlabel("Zeit (a/c)"); ax.set_ylabel("|Ez| an der Sonde")
ax.legend(fontsize=8)

ax = achsen[1]
leistung = np.abs(np.fft.rfft(reihe)) ** 2
```

```
fr2 = np.fft.rfftfreq(len(reihe), DT)
zoom = (fr2 > f11 - 0.012) & (fr2 < f11 + 0.012)
ax.plot(fr2[zoom], leistung[zoom] / leistung[zoom].max(), "o-", ms=3, lw=0.8)
ax.axhline(0.5, color="gray", ls=":", lw=0.9)
ax.set_xlabel("Frequenz (c/a)"); ax.set_ylabel("Leistung (normiert)")
plt.tight_layout(); plt.show()

print(f"Abklingrate: {rate:.5f}   (σ_D/2 = {SIGMA / 2:.5f})")
print(f"Q aus dem Abklingen     : {q_zeit:.1f}")
print(f"Q aus der Linienbreite  : {q_breite:.1f}")
```

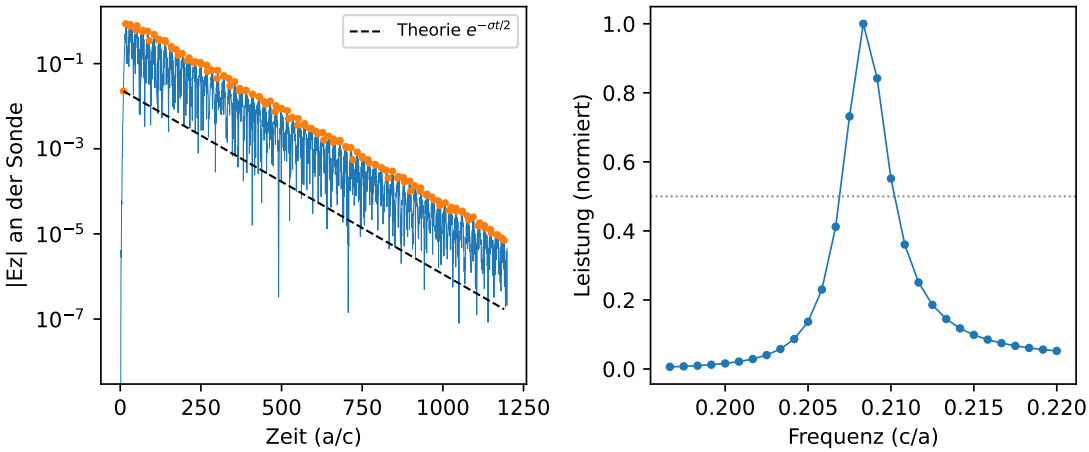


Abbildung 15.3: Links: das Sondensignal des gefüllten Resonators (blau, ausgedünnt) mit seiner Einhüllenden (orange Punkte) auf logarithmischer Skala — der exponentielle Abfall ist eine Gerade, und sie liegt auf der Theoriegeraden e^(−σt/2) (gestrichelt): Die gemessene Rate 0,01001 bestätigt Meeps σ_D-Konvention (σ_D/2 = 0,01) und liefert Q = 65,4. Rechts: der Zoom auf die TM11-Linie im Leistungsspektrum desselben Signals — aus der Nadel von eben ist eine Glockenkurve mit messbarer Halbwertsbreite geworden (graue Linie bei 50 %), und Δf = f/Q liefert Q = 62,6.

```
Abklingrate: 0.01001   (σ_D/2 = 0.01000)
Q aus dem Abklingen     : 65.4
Q aus der Linienbreite  : 62.6
```

Zwei Messungen stehen, die dritte übernimmt das neue Gerät:

```
# von oben: kasten(), SONDE, FCEN, DF, SIGMA, f11
sim = kasten(SIGMA)
# mp.Harminv(Komponente, Ort, fcen, df): Modenanalysator — zerlegt das
# Signal am Ort in gedämpfte Sinusse und liefert (freq, Q, amp) je Mode
harminv = mp.Harminv(mp.Ez, SONDE, FCEN, DF)
# mp.after_sources(h): h erst NACH dem Quellpuls mitlaufen lassen —
# Harminv soll das freie Ausschwingen sehen, nicht die Anregung
sim.run(mp.after_sources(harminv), until_after_sources=300)

moden_h = [md for md in harminv.modes if abs(md.amp) > 1e-4 and md.freq > 0.1]
q_harminv = min(moden_h, key=lambda md: abs(md.freq - f11)).Q
print(f"Q aus Harminv           : {q_harminv:.1f}")
print(f"Q aus der Theorie 2πf/σ : {2 * np.pi * f11 / SIGMA:.1f}")
```

```
Q aus Harminv           : 65.2
Q aus der Theorie 2πf/σ : 65.4
```

$65{,}4$ — $62{,}6$ — $65{,}2$ gegen die Theoriezahl $65{,}4$: Drei grundverschiedene Auswertungen desselben physikalischen Sachverhalts, höchste Abweichung gut vier Prozent (die Linienbreite leidet am gröbsten unter dem endlichen Frequenzraster $1/T$). Das ist die Checkliste in Reinform — und nebenbei der Beleg, dass Abklingzeit und Linienbreite wirklich zwei Seiten derselben Münze sind.

Harminv hat aber noch mehr gesehen — es findet **alle** Moden im Anregungsband, jede mit eigenem Q. Und weil $Q = 2\pi f/\sigma_D$ proportional zu f wächst (gleiche Verlustrate, aber mehr Schwingungen pro Zeit), liefert ein einziger Lauf gleich einen Vier-Punkte-Test der Theorie:

```
# von oben: moden_h, SIGMA
print("Mode      f        Q (Harminv)   2πf/σ_D   Abweichung")
for md in sorted(moden_h, key=lambda md: md.freq):
    soll = 2 * np.pi * md.freq / SIGMA
    print(f"       {md.freq:.4f}   {md.Q:8.1f}    {soll:8.1f}"
          f"    {(md.Q - soll) / soll * 100:+.2f} %")
```

```
Mode      f        Q (Harminv)   2πf/σ_D   Abweichung
       0.2083       65.2        65.4    -0.41 %
       0.3004       94.4        94.4    +0.04 %
       0.3559      111.8       111.8    +0.02 %
       0.4166      130.7       130.9    -0.16 %
```

Vier Moden, vier Güten, alle auf der Geraden — unter einem halben Prozent. Wer so etwas misst, darf dem Werkzeug ab jetzt auch dort vertrauen, wo es keine Theoriezahl gibt. Gut, denn genau dorthin gehen wir jetzt.

15.5 Löcher: Verlust durch Strahlung

Reale Resonatoren verlieren selten durch ihr Füllmaterial — sie verlieren durch ihre **Öffnungen**: Energie strahlt davon. Das ist kein Defekt, sondern meist der Zweck (irgendwo muss das Signal ja hinein und heraus — jeder Laser braucht einen teildurchlässigen Spiegel). Der Aufbau: Der 4×3-Kasten wird jetzt aus PEC-Blockwänden (Dicke $0{,}2$) in eine größere, offene Zelle mit PML gebaut — wie die Doppelspalt-Wand in Kapitel 14, nur als geschlossener Rahmen; in die rechte Wand stanzen wir ein Loch der Breite w. Eine Ehrlichkeit vorweg: Die diskretisierte Blockwand sitzt effektiv etwa einen Gitterpixel weiter außen, die TM_{11} rutscht dadurch um gut zwei Prozent auf $f \approx 0{,}203$ — für die Q-Frage ist das egal, wir suchen Harminv-Moden einfach dort. **Gemessen wird** Q (Harminv) für Lochbreiten von $0{,}4$ bis $1{,}6$ — zur Einordnung: Die Wellenlänge der Mode ist $\lambda = 1/f \approx 4{,}9$, das kleinste Loch misst also rund $\lambda/12$.

> Vorhersage-Punkt
>
> Der geschlossene Kasten hatte (numerisch) Q jenseits von 10^4. Was tut ein Loch von $\lambda/12$ in der Wand — halbiert es die Güte, zehntelt es sie, oder ändert es fast nichts? Und wie stark fällt Q danach pro Verdopplung der Lochbreite?

Vier Messläufe, je 400 Zeiteinheiten Ausschwingen — und ein Schnappschuss des Lecks für das Auge:

```
# von oben: LX, LY, AUFL, FCEN, DF, QUELLE, SONDE
WAND = 0.2


def loch_resonator(w_loch):
    """Der Kasten als PEC-Rahmen in offener Zelle; Loch rechts."""
    geometrie = [
        mp.Block(size=mp.Vector3(WAND, LY + 2 * WAND, mp.inf),
                 center=mp.Vector3(+(LX + WAND) / 2, 0), material=mp.metal),
        mp.Block(size=mp.Vector3(WAND, LY + 2 * WAND, mp.inf),
                 center=mp.Vector3(-(LX + WAND) / 2, 0), material=mp.metal),
        mp.Block(size=mp.Vector3(LX + 2 * WAND, WAND, mp.inf),
```

```
                 center=mp.Vector3(0, +(LY + WAND) / 2), material=mp.metal),
        mp.Block(size=mp.Vector3(LX + 2 * WAND, WAND, mp.inf),
                 center=mp.Vector3(0, -(LY + WAND) / 2), material=mp.metal),
    ]
    if w_loch > 0:                            # Stanz-Trick aus Kap. 14
        geometrie.append(mp.Block(size=mp.Vector3(WAND, w_loch, mp.inf),
                                  center=mp.Vector3(+(LX + WAND) / 2, 0),
                                  material=mp.air))
    quelle = [mp.Source(mp.GaussianSource(FCEN, fwidth=DF), component=mp.Ez,
                        center=QUELLE)]
    return mp.Simulation(cell_size=mp.Vector3(LX + 4, LY + 4),
                         resolution=AUFL, sources=quelle,
                         geometry=geometrie, boundary_layers=[mp.PML(1.0)])


q_loch = {}
for w in (0.4, 0.8, 1.2, 1.6):
    sim = loch_resonator(w)
    h = mp.Harminv(mp.Ez, SONDE, FCEN, DF)
    sim.run(mp.after_sources(h), until_after_sources=400)
    kandidaten = [md for md in h.modes if abs(md.freq - 0.203) < 0.02]
    q_loch[w] = abs(max(kandidaten, key=lambda md: abs(md.amp)).Q)

sim_bild = loch_resonator(1.2)
sim_bild.run(until=120.0)
feld = sim_bild.get_array(component=mp.Ez, center=mp.Vector3(),
                          size=mp.Vector3(LX + 4, LY + 4))

fig, achsen = plt.subplots(1, 2, figsize=(7.4, 3.3),
                           gridspec_kw={"width_ratios": [1.25, 1]})
ax = achsen[0]
w_skala = np.abs(feld).max() / 12
ax.imshow(feld.T, origin="lower", cmap="RdBu_r",
          vmin=-w_skala, vmax=w_skala,
          extent=(-(LX + 4) / 2, (LX + 4) / 2, -(LY + 4) / 2, (LY + 4) / 2))
ax.set_xlabel("x"); ax.set_ylabel("y")
ax.set_title("das Leck bei w = 1,2", fontsize=10)

ax = achsen[1]
ax.bar([str(w) for w in q_loch], list(q_loch.values()), color="tab:blue")
ax.set_yscale("log")
ax.set_xlabel("Lochbreite w"); ax.set_ylabel("Q (Harminv)")
ax.set_title("Q gegen das Loch", fontsize=10)
```

```
plt.tight_layout(); plt.show()

for w, q in q_loch.items():
    print(f"w = {w}:  Q = {q:.0f}")
```

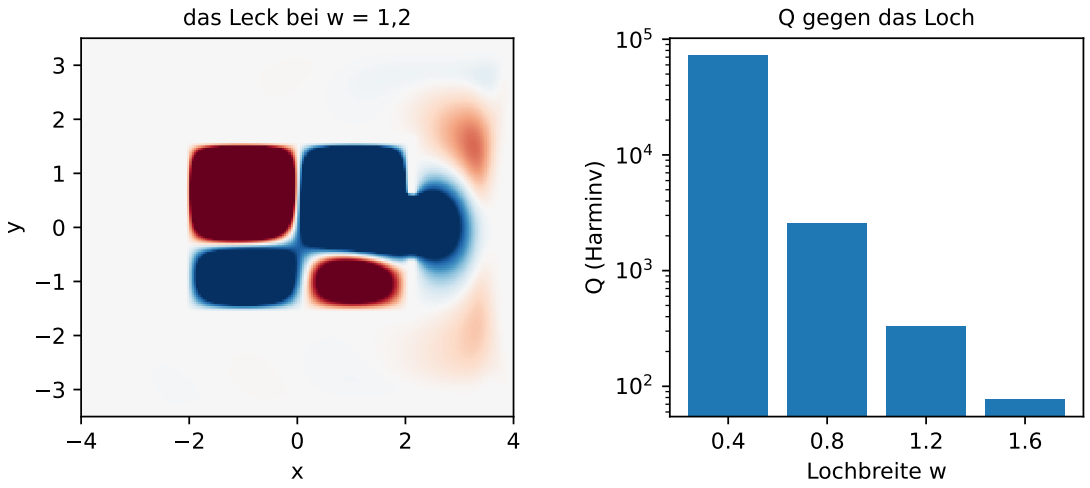


Abbildung 15.4: Links: Schnappschuss des ausschwingenden Resonators mit Loch (w = 1,2) — innen schwingt noch die vom Puls angeregte Modenmischung, aus dem Loch tropft bei jeder Schwingung ein Wellenzug davon (Farbskala gespreizt, damit das schwache Leck sichtbar wird). Rechts: die Güte über der Lochbreite (Harminv, log-Skala): Von w = 0,8 bis 1,6 stürzt Q um anderthalb Größenordnungen — aber das kleine Loch (w = 0,4 ≈ λ/12) lässt Q bei zigtausend, praktisch dicht. Kleine Löcher sperren die Welle evaneszent (Kapitel 12), genau deshalb darf die Mikrowellenherd-Tür ein Lochgitter tragen.

```
w = 0.4:  Q = 72479
w = 0.8:  Q = 2580
w = 1.2:  Q = 332
w = 1.6:  Q = 77
```

Von $w = 0{,}8$ bis $1{,}6$ stürzt die Güte von 2600 auf 77 — das Loch ist der Verlustkanal, und jede Verbreiterung kostet brutal. Die eigentliche Überraschung ist aber das *kleine* Loch: Bei $w = 0{,}4$ bleibt Q bei über siebzigtausend, vom geschlossenen Kasten im Rahmen unserer Messzeit nicht zu unterscheiden. Warum sperrt ein offenes Loch? Weil der Schlitz

für die Welle ein winziger Wellenleiter ist, und in einen PEC-Kanal der Breite w passt keine laufende Ez-Welle mit $\lambda > 2w$ (das Feld müsste an beiden Kanten null sein und dazwischen schwingen — dieselbe Arcussinus-Sackgasse wie beim Grenzwinkel in Kapitel 12). Unterhalb dieses **Cutoffs** dringt das Feld nur evaneszent in den Schlitz ein und klingt dort ab; unsere Mode mit $\lambda = 4{,}9$ bräuchte $w > 2{,}45$ für freie Durchfahrt. Genau darum darfst du durch die Gittertür eines Mikrowellenherds hindurchschauen: Die Millimeter-Löcher sind für sichtbares Licht ($\lambda \sim 0{,}5$ µm) Scheunentore, für die 12-cm-Mikrowelle aber tief unter Cutoff. Kapitel 17 macht aus diesem Nebensatz ein eigenes Kapitel.

15.6 Das Kapitel-Programm

`programme/kap15/kap15_resonanz_guete.py` bündelt alle Befunde eigenständig, mit `assert`-Schranken: Kammfrequenzen auf $0{,}2\,\%$, Abklingrate $= \sigma_D/2$ auf $5\,\%$, alle drei Q-Messungen binnen $10\,\%$ an $2\pi f/\sigma_D$, die vier Harminv-Moden auf $2\,\%$ an der Q-Geraden, und die Loch-Hierarchie ($Q_{0,4} > 20\,Q_{0,8} > \ldots$).

Merkkasten

- **Hohlraum-Moden:** $f_{mn} = \frac{c}{2}\sqrt{(m/L_x)^2 + (n/L_y)^2}$ — der Kapitel-10-Kamm mit zwei Indizes; Modenbilder mit Bäuchen und Knotenlinien.
- **Güte** $Q = \omega W/P$: Amplitude klingt mit $e^{-\omega t/2Q}$, Energie mit $e^{-\omega t/Q}$. Material mit σ_D: exakt $Q = 2\pi f/\sigma_D$ (Konvention selbst nachgemessen: Rate $= \sigma_D/2$).
- **Endlich lang = endlich scharf:** Die Linie trägt die Breite $\Delta f = f/Q$ — Abklingzeit und Linienbreite sind dieselbe Information (Kapitel-10-Reziprozität, quantitativ). Drei Messwege, ein Ergebnis: 65,4 / 62,6 / 65,2 gegen 65,4.
- **Harminv** misst f und Q aller Moden aus einer kurzen Zeitreihe — das Spektrometer für Ungeduldige (und für hohe Q, denen keine FFT-Aufnahme lang genug wäre).
- **Löcher strahlen** — Q fällt um Größenordnungen mit der Lochbreite; aber unterhalb des Cutoffs ($\lambda > 2w$) sperrt der Schlitz evaneszent. Mikrowellentür-Prinzip.

Roter Faden

Die Glocke aus **Kapitel 10** hat jetzt ihre Kennzahl, ihr Kamm seine zweite Dimension und ihre Peaks die versprochene Breite — geliefert von der Reziprozität, die dort „Chor" hieß. Das σ aus **Kapitel 6** wurde vom Störenfried zum geeichten Werkzeug, die DFT-Felder aus **Kapitel 11** malten die Modenbilder, und das evaneszente Sperren aus **Kapitel 12** erklärt nebenbei die Mikrowellentür. Nach vorn: **Kapitel 17** nimmt sich den Cutoff als Hauptdarsteller vor (Hohlleiter und Tunneln), **Kapitel 19** überträgt Q auf den schwingenden Stromkreis, und in **Kapitel 27** wird der teildurchlässige Resonator zur Hauptfigur — Fabry-Pérot und der Schritt zum Laser leben exakt von dem Kompromiss, den unser Koppel-Loch vorgeführt hat: Ohne Leck kein Nutzsignal, mit Leck kein hohes Q.

Übungen

Ü 15.1 (Verstehen). Das Weinglas vom Kapitelanfang: $f = 1000$ Hz, die Amplitude braucht $\tau = 3$ s bis auf $1/e$. Berechne Q von Hand (aus $e^{-\omega t/2Q}$ folgt $Q = \pi f \tau$). Wie lange müsste ein Resonator mit dem Q unserer TM_{11} ($Q = 65$) bei 1000 Hz nachklingen — und warum klingt ein Plastikbecher gar nicht?

> **Musterlösung zu Ü 15.1**
>
> $Q = \pi f \tau = \pi \cdot 1000 \cdot 3 \approx 9400$ — das Weinglas ist ein Resonator mit der Güte zehntausend (Glas ist steif und innerlich verlustarm). Mit $Q = 65$ wäre $\tau = Q/(\pi f) = 65/(\pi \cdot 1000) \approx 21$ ms — ein dumpfer „Tock" statt eines Tons. Der Plastikbecher liegt noch darunter: Weiche Polymere wandeln Schwingungsenergie in innere Reibung (Wärme) um, Q einstellig, die Schwingung ist nach wenigen Perioden verzehrt — und eine Linie mit $\Delta f = f/Q \sim f/5$ ist so breit, dass von „Tonhöhe" keine Rede mehr sein kann. Genau deshalb *singt* Glas und *klackt* Plastik.

Ü 15.2 (Verstehen). Im Mikrowellenherd (2,45 GHz) gibt es notorisch heiße und kalte Stellen, und ein Drehteller. Erkläre beides mit den Modenbildern dieses Kapitels — und rechne nach: Legt man eine Tafel Schokolade hinein (ohne Teller!), schmelzen Flecken im Abstand von etwa 6 cm. Welche Naturkonstante kann man daraus bestimmen?

Musterlösung zu Ü 15.2

Die Garkammer ist ein Hohlraumresonator, und das Feld darin ist eine Überlagerung von Moden wie in unseren Bildern — mit Bäuchen (heiß) und Knotenlinien (kalt, dort steht das Feld dauerhaft auf null). Der Drehteller fährt das Essen durch Bäuche *und* Knoten, damit es im Mittel überall gart. Die Schmelzflecken der Schokolade markieren die Bäuche einer stehenden Welle; benachbarte Bäuche liegen $\lambda/2$ auseinander (Kapitel 6), also $\lambda \approx 12{,}2$ cm — und mit der aufgedruckten Frequenz folgt $c = \lambda f \approx 0{,}122 \cdot 2{,}45 \cdot 10^9 \approx 3{,}0 \cdot 10^8$ m/s: **die Lichtgeschwindigkeit, gemessen mit Schokolade.** (Im Detail mitteln reale Herde über mehrere Moden und verstimmen sich durch das Gargut — die 6 cm sind erstaunlich robust, aber kein Präzisionsexperiment.)

Ü 15.3 (Verändern). Verdopple und halbiere σ_D (0,01 / 0,02 / 0,04) und miss jeweils mit Harminv das Q der **stärksten** Mode (das ist hier die TM_{12} — sie liegt der Quelle am günstigsten). Sage vorher, was herauskommen muss, und prüfe gegen $Q = 2\pi f/\sigma_D$.

Musterlösung zu Ü 15.3

$Q \propto 1/\sigma_D$: doppelte Verluste, halbe Güte — für die TM_{12} ($f = 0{,}356$) erwartet also etwa $224/112/56$:

```
# von oben: kasten(), SONDE, FCEN, DF
for sig in (0.01, 0.02, 0.04):
    sim = kasten(sig)
    h = mp.Harminv(mp.Ez, SONDE, FCEN, DF)
    sim.run(mp.after_sources(h), until_after_sources=300)
    md = max((m for m in h.modes if m.freq > 0.1),
             key=lambda m: abs(m.amp))
    print(f"σ_D = {sig}:  f = {md.freq:.4f},  Q = {md.Q:6.1f}"
          f"   (Theorie {2 * np.pi * md.freq / sig:6.1f})")
```

```
σ_D = 0.01:  f = 0.3559,  Q =  223.7   (Theorie  223.6)
σ_D = 0.02:  f = 0.3559,  Q =  111.8   (Theorie  111.8)
σ_D = 0.04:  f = 0.3561,  Q =   52.5   (Theorie   55.9)
```

Die Halbierung pro Verdopplung sitzt — und der letzte Wert verrät nebenbei die Grenze des Messgeräts: Bei $\sigma_D = 0{,}04$ ist das Signal nach den 300 Zeiteinheiten der Analyse um e^{-6} gefallen, Harminv fittet zunehmend am Restrauschen und liegt sechs Prozent unter der Theorie (wer die Analysezeit verkürzt, trifft besser). Noch deutlicher trifft es die

leise TM_{11}: Sie taucht bei $\sigma_D = 0{,}04$ in der Modenliste gar nicht mehr auf — schwach angeregte, stark gedämpfte Moden verliert Harminv zuerst. Der praktische Umkehrschluss bleibt: Wer Q misst, kennt die Verluste seines Materials — die Linienbreite ist ein Materialprüfgerät.

Ü 15.4 (Übertragen). Versetze das Loch ($w = 1{,}2$) von der rechten Wand in die **Mitte der oberen Wand** (bei $x = 0$) und lies mit Harminv die Güten der TM_{11} (bei $\approx 0{,}20$) und der TM_{21} (bei $\approx 0{,}29$) ab. Wirf vorher einen Blick auf die beiden Modenbilder: Welche der beiden Moden wird das Loch kaum bemerken — und warum?

Musterlösung zu Ü 15.4

Die TM_{21} hat bei $x = 0$ ihre **Knotenlinie** — genau dort, wo das Loch sitzt, ist ihr Feld ohnehin null, es gibt nichts, was hinausstrahlen könnte. Die TM_{11} dagegen hat bei $x = 0$ ihren Bauch direkt an der oberen Wand:

```
# von oben: LX, LY, WAND, AUFL, FCEN, DF, QUELLE, SONDE, loch_resonator-Idee
geometrie = [
    mp.Block(size=mp.Vector3(WAND, LY + 2 * WAND, mp.inf),
             center=mp.Vector3(+(LX + WAND) / 2, 0), material=mp.metal),
    mp.Block(size=mp.Vector3(WAND, LY + 2 * WAND, mp.inf),
             center=mp.Vector3(-(LX + WAND) / 2, 0), material=mp.metal),
    mp.Block(size=mp.Vector3(LX + 2 * WAND, WAND, mp.inf),
             center=mp.Vector3(0, +(LY + WAND) / 2), material=mp.metal),
    mp.Block(size=mp.Vector3(LX + 2 * WAND, WAND, mp.inf),
             center=mp.Vector3(0, -(LY + WAND) / 2), material=mp.metal),
    mp.Block(size=mp.Vector3(1.2, WAND, mp.inf),        # Loch OBEN Mitte
             center=mp.Vector3(0, +(LY + WAND) / 2), material=mp.air)]
sim = mp.Simulation(cell_size=mp.Vector3(LX + 4, LY + 4), resolution=AUFL,
                    sources=[mp.Source(mp.GaussianSource(FCEN, fwidth=DF),
                                       component=mp.Ez, center=QUELLE)],
                    geometry=geometrie, boundary_layers=[mp.PML(1.0)])
h = mp.Harminv(mp.Ez, SONDE, FCEN, DF)
sim.run(mp.after_sources(h), until_after_sources=400)
for ziel, name in ((0.203, "TM11"), (0.293, "TM21")):
    md = min((m for m in h.modes if abs(m.amp) > 1e-3),
             key=lambda m: abs(m.freq - ziel))
    print(f"{name}: f = {md.freq:.4f}, |Q| = {abs(md.Q):.0f}")
```

```
TM11: f = 0.1970, |Q| = 204
TM21: f = 0.2933, |Q| = 46386
```

Die TM_{11} bricht auf Q in der Größenordnung 200 ein, die TM_{21} bleibt bei $|Q| > 10^4$ — **ein Resonator verliert nur dort, wo seine Mode wohnt.** (Bei derart verlustarmen Moden kann Harminv sogar ein sinnloses Vorzeichen liefern — lies $|Q|$ schlicht als „zu groß für 400 Zeiteinheiten Messzeit".) Dieses Prinzip ist ein Designwerkzeug: Sensoren koppeln dort an, wo die Mode stark ist; Störquellen versteckt man auf ihren Knoten — und ein selektives Loch kann aus einem Resonator einen Modenfilter machen, der eine Mode entlässt und die andere hütet.

Das Kleingedruckte

Drei Randnotizen. Erstens: Die Linienform unserer Resonanz ist eine **Lorentz-Kurve** — sie fällt zu den Flanken viel träger als eine Gauß-Glocke, weshalb sich nahe Resonanzen schon „anschmecken", lange bevor man sie trifft (Kapitel 16 begegnet derselben Kurve als Materialantwort wieder). Zweitens: Wer den Resonator nicht anschlägt, sondern mit einem Dauerton der Frequenz f **treibt**, misst als Antwortamplitude über f exakt dieselbe Lorentz-Linie — Linienbreite und Resonanzkurve der erzwungenen Schwingung sind ein und dasselbe Objekt; unser Puls hat nur (Kapitel 10!) alle Treibfrequenzen gleichzeitig abgefragt. Drittens: Reale Hohlraumwände aus echtem Metall sind keine perfekten Spiegel; ihre Restverluste begrenzen Mikrowellen-Resonatoren auf $Q \sim 10^4$, supraleitende Kavitäten schaffen 10^{10} — dort schwingt ein Feld *sekundenlang*, das beste Weinglas der Physik.

16 Dispersion: Phasen- und Gruppengeschwindigkeit

Ein Wintersturm tobt über dem Südpazifik und wühlt das Meer auf. Drei Tage später, viertausend Kilometer entfernt, läuft an einem kalifornischen Strand eine lange, gleichmäßige Dünung ein — und die Surfer dort wissen etwas Bemerkenswertes: Die ersten Wellen, die ankommen, sind immer die **langen**; die kurzen, kabbeligen folgen Stunden bis Tage später. Der Ozean hat das Wellengewirr des Sturms unterwegs **nach Wellenlänge sortiert**, ganz von selbst. Das geht nur, wenn lange Wasserwellen schneller laufen als kurze — wenn also die Geschwindigkeit einer Welle von ihrer Wellenlänge abhängt. Genau das heißt **Dispersion**.

Dasselbe Sortieren, millionenfach schneller, passiert in jedem Glasprisma: Weißes Licht geht hinein, ein Regenbogen kommt heraus, weil Glas verschiedene Frequenzen verschieden schnell laufen lässt. Kapitel 14 ist auf diesem Punkt etwas schuldig geblieben — das Beugungsgitter sortierte Farben, aber *andersherum* als ein Prisma, und die Auflösung wurde auf dieses Kapitel vertagt. Vorher müssen wir allerdings eine unbequeme Entdeckung machen: Sobald ein Medium dispersiv ist, hat eine Welle nicht mehr *eine* Geschwindigkeit, sondern **zwei** — die Wellenberge laufen mit dem einen Tempo, das Wellenpaket (und mit ihm Energie und Botschaft) mit einem anderen. Am Ende des Kapitels wird eines dieser beiden Tempi in einem unserer Experimente **schneller als das Licht im Vakuum** sein, mit Messprotokoll — und Einstein wird sich trotzdem nicht im Grab umdrehen müssen.

Lernziele

Nach diesem Kapitel kannst du …

1. … Phasengeschwindigkeit $v_p = \omega/k$ und Gruppengeschwindigkeit $v_g = d\omega/dk$ unterscheiden, aus einer Dispersionsrelation berechnen und beide in einer Simulation messen,
2. … erklären, warum Glas dispersiv ist (der Lorentz-Oszillator: gebundene Elektronen sind Resonatoren aus Kapitel 15 — und ihre Antwort hängt von der Treibfrequenz ab),

3. … die Prismen-Ablenkung mit der Snellius-Kette ausrechnen und begründen, warum das Prisma Blau stärker ablenkt, das Gitter aber Rot,
4. … das Drude-Modell freier Elektronen anwenden: warum Metalle spiegeln, warum die Ionosphäre Kurzwelle um den Globus trägt — und warum $v_p > c$ kein Widerspruch zur Relativität ist,
5. … die Plasmonen-Resonanz einer Gold-Nanostruktur messen und erklären, warum Goldrubinglas rot ist.

16.1 Zwei Wellen genügen: die Schwebung

Was ist überhaupt „die Geschwindigkeit" einer Welle? Für eine einzelne, unendlich lange Sinuswelle ist die Antwort eindeutig: Ein Wellenberg sitzt dort, wo die Phase $kx - \omega t$ einen festen Wert hat; wandert die Zeit um Δt weiter, muss x um $(\omega/k)\,\Delta t$ nachrücken. Das Tempo der Berge ist also

$$v_p \;=\; \frac{\omega}{k} \qquad \text{(Phasengeschwindigkeit).}$$

Aber eine unendliche Sinuswelle trägt keine Botschaft — sie sieht heute aus wie gestern. Wer ein Signal schicken will, muss der Welle eine *Form* aufprägen: einen Anfang, ein Ende, ein Paket. Und die billigste Form entsteht, wenn man nur **zwei** Sinuswellen mit leicht verschiedenem k überlagert. Das Additionstheorem — es trug schon den Doppelspalt in Kapitel 14 — verwandelt die Summe in ein Produkt:

$$\cos(k_1 x - \omega_1 t) + \cos(k_2 x - \omega_2 t) \;=\; 2\,\underbrace{\cos\!\big(\bar{k}x - \bar{\omega}t\big)}_{\text{Träger}} \cdot \underbrace{\cos\!\Big(\tfrac{\Delta k}{2}x - \tfrac{\Delta\omega}{2}t\Big)}_{\text{Einhüllende}}$$

mit den Mittelwerten $\bar{k} = (k_1 + k_2)/2$, $\bar{\omega} = (\omega_1 + \omega_2)/2$ und den Differenzen $\Delta k = k_2 - k_1$, $\Delta\omega = \omega_2 - \omega_1$. Ein schneller Träger, moduliert von einer langsamen, breiten Einhüllenden — eine **Schwebung**, in Wellengruppen zerschnitten. Und jetzt das Entscheidende: Träger und Einhüllende sind selbst wieder von der Form $\cos(Kx - \Omega t)$, jede mit ihrem eigenen Tempo. Der Träger läuft mit $\bar{\omega}/\bar{k} \approx v_p$. Die Einhüllende aber läuft mit

$$v_g \;=\; \frac{\Delta\omega}{\Delta k} \;\xrightarrow{\Delta k \to 0}\; \frac{d\omega}{dk} \qquad \text{(Gruppengeschwindigkeit)}$$

— der *Steigung* der Dispersionsrelation $\omega(k)$, nicht ihrem Verhältnis ω/k. Solange $\omega(k)$ eine Ursprungsgerade ist (Vakuum: $\omega = ck$), sind Steigung und Verhältnis dasselbe und beide Tempi gleich c. Sobald die Kurve krümmt, gehen sie auseinander.

Die Handrechnung dazu, mit der Dispersionsrelation der Tiefwasserwellen vom Kapitelanfang, $\omega = \sqrt{gk}$ (wir setzen $g = 1$, Spielzeug-Einheiten): $k_1 = 1$ gibt $\omega_1 = 1$; $k_2 = 1{,}2$ gibt $\omega_2 = \sqrt{1{,}2} = 1{,}0954$. Träger: $\bar{k} = 1{,}1$, $\bar{\omega} = 1{,}0477$, also $v_p = 1{,}0477/1{,}1 = 0{,}9525$. Einhüllende: $\Delta\omega/\Delta k = 0{,}0954/0{,}2 = 0{,}4772$ — fast genau die **Hälfte**. Das ist kein Zufall dieser Zahlen: Die Ableitung von $\omega = \sqrt{gk}$ ist $d\omega/dk = \frac{1}{2}\sqrt{g/k} = \omega/2k$, exakt die halbe Phasengeschwindigkeit. Beide Tempi sind hier Skalare (Zahlen, keine Vektoren — wir messen entlang einer Richtung).

Das Experiment dazu braucht kein Gitter und keinen Löser — die Summe zweier Kosinus können wir direkt auswerten. **Frage:** Laufen Berge und Gruppen wirklich mit $0{,}9525$ bzw. $0{,}4772$? **Bühne:** die x-Achse von -100 bis 200, fein abgetastet; die Zeit ist ein Parameter der Formel, es gibt keine Update-Schleife. **Gemessen wird** zweierlei: ein Wellenberg, der bei $t = 0$ im Zentrum der Gruppe um $x = 0$ sitzt, wird in kleinen Zeitschritten verfolgt (in jedem Schritt suchen wir das lokale Maximum nahe der alten Position); und der **Energie-Schwerpunkt** der Gruppe, $\sum x\,u^2 / \sum u^2$ in einem mitwandernden Fenster — denn die Energie sitzt im Quadrat des Feldes (Kapitel 4), und „wo das Paket ist" heißt „wo seine Energie ist". **Erfolgskriterium:** beide Messwerte auf etwa ein Prozent an den Sollwerten.

```
import numpy as np
import matplotlib.pyplot as plt

def omega_von_k(k):
    """Tiefwasserwellen: ω = √(gk) mit g = 1."""
    return np.sqrt(k)

K1, K2 = 1.0, 1.2
W1, W2 = omega_von_k(K1), omega_von_k(K2)
VP_SOLL = (W1 + W2) / 2 / ((K1 + K2) / 2)    # Träger: 0,9525
VG_SOLL = (W2 - W1) / (K2 - K1)              # Einhüllende: 0,4772

x = np.linspace(-100, 200, 6000)

def schwebung(t):
    return np.cos(K1 * x - W1 * t) + np.cos(K2 * x - W2 * t)

def gruppen_schwerpunkt(t, mitte, halbbreite=14):
```

```
    """Energie-Schwerpunkt der Gruppe im Fenster um `mitte`."""
    u = schwebung(t)
    m = (x > mitte - halbbreite) & (x < mitte + halbbreite)
    return np.sum(x[m] * u[m]**2) / np.sum(u[m]**2)

# Gruppe verfolgen: Schwerpunkt bei t = 0 und t = 20
s0 = gruppen_schwerpunkt(0.0, 0.0)
s1 = gruppen_schwerpunkt(20.0, VG_SOLL * 20)
vg_mess = (s1 - s0) / 20

# einen Berg verfolgen: kleine Schritte, Maximum nahe der alten Position
berg, dt = 0.0, 0.05
for n in range(200):                          # bis t = 10
    u = schwebung((n + 1) * dt)
    m = (x > berg - 2) & (x < berg + 2)
    berg = x[m][np.argmax(u[m])]
vp_mess = berg / 10

print(f"Berge:   v_p = {vp_mess:.4f}   (Theorie {VP_SOLL:.4f})")
print(f"Gruppe:  v_g = {vg_mess:.4f}   (Theorie {VG_SOLL:.4f})")
print(f"Verhältnis v_g/v_p = {vg_mess/vp_mess:.3f}   (Theorie 0,5)")
```

```
Berge:   v_p = 0.9468   (Theorie 0.9525)
Gruppe:  v_g = 0.4768   (Theorie 0.4772)
Verhältnis v_g/v_p = 0.504   (Theorie 0,5)
```

Beide Tempi getroffen (der Berg liegt ein halbes Prozent daneben — er verformt sich leicht, während er durch die Gruppe wandert; gleich sehen wir, warum er das tut). Dasselbe als Bild:

```
# von oben: schwebung(), gruppen_schwerpunkt(), K1, K2, W1, W2,
#           VP_SOLL, VG_SOLL, x, berg (Bergposition bei t = 10)
fig, achsen = plt.subplots(2, 1, figsize=(8, 4.6), sharex=True)
for achse, t in zip(achsen, (0.0, 20.0)):
    u = schwebung(t)
    huelle = 2 * np.abs(np.cos(0.5 * (K2 - K1) * x
                               - 0.5 * (W2 - W1) * t))
    achse.plot(x, u, "C0", lw=0.8)
    achse.plot(x, huelle, "k--", lw=1.0, label="Einhüllende")
    achse.plot(x, -huelle, "k--", lw=1.0)
    sp = gruppen_schwerpunkt(t, VG_SOLL * t)
    achse.plot(sp, 2.25, "o", color="C3", clip_on=False)
```

```
    achse.axvline(VG_SOLL * t, color="C3", ls=":", lw=1)
    achse.set(ylabel="u", ylim=(-2.4, 2.4), xlim=(-40, 60))
    achse.annotate(f"t = {t:.0f}", (0.02, 0.82),
                   xycoords="axes fraction")
# der verfolgte Berg: bei t = 0 in der Gruppenmitte, bei t = 20 vorn
achsen[0].plot(0, 2.0, "v", color="C2", ms=8, clip_on=False)
achsen[1].plot(VP_SOLL * 20, 2.0, "v", color="C2", ms=8,
               clip_on=False, label="verfolgter Berg (Soll)")
achsen[1].set_xlabel("x")
achsen[1].legend(loc="lower right", fontsize=8)
fig.tight_layout()
plt.show()
```

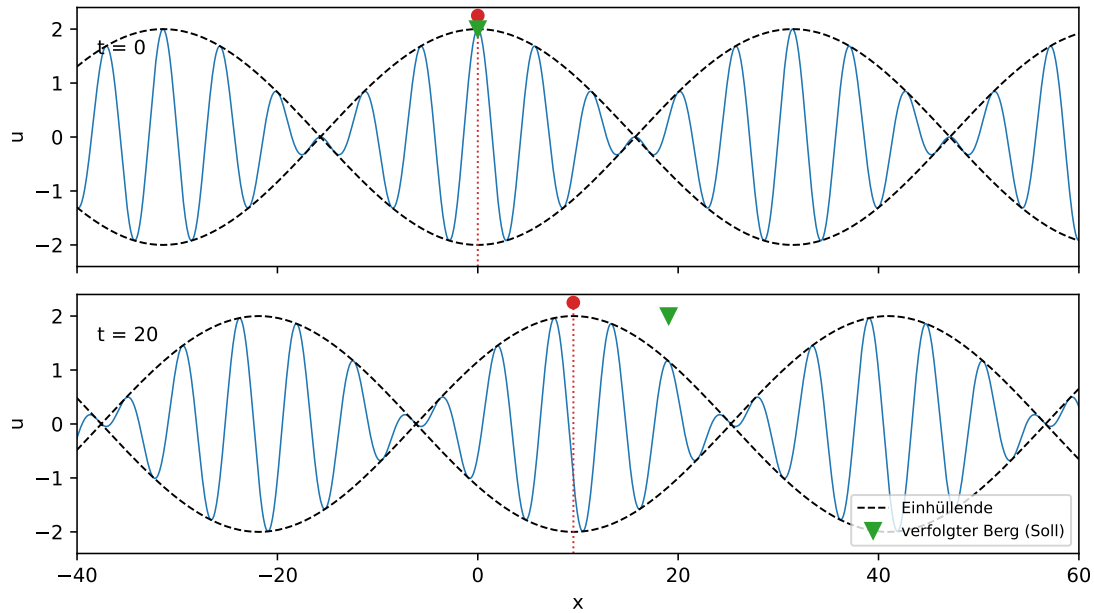


Abbildung 16.1: Schwebung zweier Tiefwasserwellen ($\omega = \sqrt{k}$) bei $t = 0$ (oben) und $t = 20$ (unten). Die gestrichelte Einhüllende wandert mit $v_g = 0{,}477$ (runder Marker und gepunktete Soll-Linie), der verfolgte Wellenberg mit $v_p = 0{,}952$ (Dreieck) — doppelt so schnell. Der Berg, der anfangs in der Gruppenmitte saß, ist bei $t = 20$ fast an deren Vorderkante angekommen: Berge wandern durch die Gruppe nach vorn und verebben dort.

Im unteren Panel steckt eine Beobachtung, die gleich noch wichtig wird: Der markierte Berg ist gegenüber der Gruppenmitte **nach vorn** gerückt.

Weil hier $v_p > v_g$ gilt, werden Berge am hinteren Rand der Gruppe „geboren", wandern durch sie hindurch und verebben an der Vorderkante — am Strand kann man genau das beobachten, wenn ein Wellen-Set einläuft: Die vorderste Welle des Sets verschwindet scheinbar, während hinten eine neue auftaucht. Kein Berg überlebt den Weg; nur das *Paket* kommt an. Und mit ihm die Energie: Unser Schwerpunkt-Detektor hat gemessen, dass das Energiequadrat u^2 mit v_g reist, nicht mit v_p. Wer eine Botschaft schickt — einen Puls, ein Morsezeichen, ein Bit — schickt eine Einhüllende. **Information und Energie reisen mit der Gruppengeschwindigkeit.**

Merkkasten: zwei Geschwindigkeiten

- $v_p = \omega/k$ — das Tempo der Wellenberge (der Phase).
- $v_g = d\omega/dk$ — das Tempo des Pakets: Steigung der Dispersionsrelation, nicht ihr Verhältnis.
- Energie und Information reisen mit v_g.
- Ohne Dispersion ($\omega = ck$) fallen beide zusammen; jede Krümmung von $\omega(k)$ treibt sie auseinander.

16.2 Ein alter Bekannter: das Gitter ist dispersiv

Wir besitzen seit Kapitel 8 ein Medium mit *exakt bekannter*, krummer Dispersionsrelation — das FDTD-Gitter selbst. Dort stand:

$$\sin\left(\frac{\omega\,\Delta t}{2}\right) \;=\; S\cdot\sin\left(\frac{k\,\Delta x}{2}\right),$$

und die Folge war, dass grob aufgelöste Wellen zu langsam laufen („numerische Dispersion"). Damals haben wir die Phasengeschwindigkeit über Nulldurchgänge gemessen. Jetzt, mit dem Begriffspaar aus dem letzten Abschnitt, können wir mehr: Die **Ableitung** der Dispersionsrelation nach k liefert (Kettenregel links und rechts, dann nach $d\omega/dk$ auflösen)

$$v_g \;=\; \frac{d\omega}{dk} \;=\; c\,\frac{\cos(k\Delta x/2)}{\cos(\omega\Delta t/2)},$$

eine Vorhersage für das Tempo eines ganzen *Pakets* auf dem Gitter. Die Handrechnung für $S = 0{,}5$ und eine Trägerwelle mit 5 Punkten pro Wellenlänge ($k\Delta x = 2\pi/5$, also $k\Delta x/2 = 0{,}6283$): Aus der Dispersionsrelation

folgt $\sin(\omega\Delta t/2) = 0{,}5 \cdot \sin(0{,}6283) = 0{,}2939$, also $\omega\Delta t/2 = 0{,}2985$. Damit $v_p = \omega/k = 0{,}9495\,c$ (die Berge sind 5 % zu langsam, wie in Kapitel 8) — aber $v_g = c \cdot \cos(0{,}6283)/\cos(0{,}2985) = c \cdot 0{,}8090/0{,}9558 = 0{,}8464\,c$: Das **Paket** ist sogar **15 %** zu langsam. Die Gruppengeschwindigkeit reagiert empfindlicher auf die Gitterkrümmung als die Phase — sie ist ja die Ableitung, und Ableitungen verstärken Dellen.

Das Experiment: das 1D-Wellenlabor aus Kapitel 6, absichtlich grob betrieben. **Frage:** Laufen Berge mit $0{,}9495\,c$ und Paket mit $0{,}8464\,c$? **Bühne:** 2600 Gitterpunkte, $\Delta x = 1$, $S = 0{,}5$ (also $\Delta t = 0{,}5$, $c = 1$); an beiden Enden Mur-Ränder aus Kapitel 6, damit nichts reflektiert. **Anregung:** eine weiche Quelle bei Punkt 200 sendet einen Sinus der Gitterfrequenz $\omega = 1{,}1938$ (genau die Frequenz, die zum 5-Punkte-Träger gehört), moduliert mit einer Gauß-Einhüllenden (Breite 40 Zeiteinheiten um $t = 150$) — ein Wellenpaket, schmalbandig genug, dass „die" Gruppengeschwindigkeit wohldefiniert ist (Kapitel 10: kurz = breitbandig, wir wählen bewusst *lang*). Die weiche Quelle strahlt nach beiden Seiten; das linkslaufende Zwillingspaket verschwindet im linken Mur-Rand. **Gemessen wird** dreierlei: ω aus den Nulldurchgängen einer Sonden-Zeitreihe und k aus den Nulldurchgängen eines Schnappschusses (beides Werkzeuge aus Kapitel 8/12) für $v_p = \omega/k$; die Bahn des Energie-Schwerpunkts $\sum x\,u^2/\sum u^2$ über die Zeit; und als Gegenprobe die Ankunftszeiten des Pakets an zwei Sonden (Punkte 900 und 1500). **Erfolgskriterium:** alle drei Messwerte auf ein Prozent an der Gittertheorie.

> **Vorhersage-Punkt**
>
> Im x-t-Diagramm des Laufs (Ort waagerecht, Zeit senkrecht aufwärts) erscheint das Paket als schräger Streifen. Die einzelnen Wellenberge darin sind feine Streifen mit *eigener* Steigung. Welche der beiden Spuren verläuft steiler — die der Berge oder die des Pakets? (Steiler heißt hier: mehr Zeit pro Strecke, also *langsamer*.)

Die Antwort steht in der Rechnung oben: Das Paket ist mit $0{,}85\,c$ das langsamere, seine Spur verläuft steiler; die Berge (flachere Streifen, $0{,}95\,c$) durchqueren das Paket von hinten nach vorn — wie am Strand. Der Lauf dazu:

```
# von oben: np, plt
C, DX, S = 1.0, 1.0, 0.5
DT = S * DX / C
N_GITTER, N_SCHRITTE = 2600, 4400
K_TRAEGER = 2 * np.pi / 5                    # 5 Punkte pro Wellenlänge
W_TRAEGER = (2 / DT) * np.arcsin(S * np.sin(K_TRAEGER * DX / 2))
VP_GITTER = W_TRAEGER / K_TRAEGER            # 0,9495 c
VG_GITTER = C * np.cos(K_TRAEGER * DX / 2) / np.cos(W_TRAEGER * DT / 2)
```

```
QUELLE, T0, SIG = 200, 150.0, 40.0
SONDE1, SONDE2 = 900, 1500
MUR = (S - 1) / (S + 1)                      # Mur-Rand (Kapitel 6)

u = np.zeros(N_GITTER)
B = np.zeros(N_GITTER - 1)
reihe1, reihe2, bahn, xt_fein = [], [], [], []
for n in range(N_SCHRITTE):
    t = n * DT
    B += (DT / DX) * np.diff(u)
    u_l, u_r, u0, uN = u[1], u[-2], u[0], u[-1]
    u[1:-1] += (C**2 * DT / DX) * np.diff(B)
    u[0] = u_l + MUR * (u[1] - u0)           # Mur links
    u[-1] = u_r + MUR * (u[-2] - uN)         # Mur rechts
    u[QUELLE] += 0.1 * np.sin(W_TRAEGER * t) * np.exp(-((t - T0) / SIG)**2)
    reihe1.append(u[SONDE1])
    reihe2.append(u[SONDE2])
    if t > T0 + 4 * SIG and n % 40 == 0:     # Schwerpunktbahn
        e = u**2
        e[:QUELLE + 80] = 0.0                # Quellgegend ausblenden
        bahn.append((t, np.sum(np.arange(N_GITTER) * e) / np.sum(e)))
    if 1200 <= n < 1900 and n % 2 == 0:      # feines x-t-Fenster
        xt_fein.append(u[600:1000].copy())

reihe1, reihe2 = np.array(reihe1), np.array(reihe2)
zeiten = np.arange(N_SCHRITTE) * DT
bahn = np.array(bahn)
m = bahn[:, 0] > 600
vg_bahn = np.polyfit(bahn[m, 0], bahn[m, 1], 1)[0]

fig, (links, rechts) = plt.subplots(
    1, 2, figsize=(9, 4.2), gridspec_kw={"width_ratios": [1.15, 1]})
xt = np.array(xt_fein)
links.imshow(xt, origin="lower", aspect="auto", cmap="RdBu_r",
             extent=[600, 1000, 600, 950],
             vmin=-np.abs(xt).max(), vmax=np.abs(xt).max())
x0, t0 = 200 + VG_GITTER * (775 - 150), 775  # Paketmitte bei t = 775
for v, stil, name in ((VP_GITTER, "k--", "Berge: $v_p$ = 0,950 c"),
                      (VG_GITTER, "k-", "Paket: $v_g$ = 0,846 c")):
    links.plot([x0 - v * 175, x0 + v * 175], [t0 - 175, t0 + 175],
               stil, lw=1.4, label=name)
# eng zoomen, damit die einzelnen Bergspuren im Druck auflösbar sind
links.set(xlim=(680, 830), ylim=(700, 860), xlabel="Gitterpunkt",
```

```
          ylabel="Zeit", title="Paketstreifen mit Bergspuren")
links.legend(loc="upper left", fontsize=8)

rechts.plot(bahn[m, 0], bahn[m, 1], "o", ms=3, label="Schwerpunkt")
rechts.plot(bahn[m, 0], 200 + VG_GITTER * (bahn[m, 0] - 150), "k-",
            lw=1, label="Theorie: $x = x_0 + v_g t$")
rechts.set(xlabel="Zeit", ylabel="Gitterpunkt",
           title="Bahn des Energie-Schwerpunkts")
rechts.legend(fontsize=8)
fig.tight_layout()
plt.show()
```

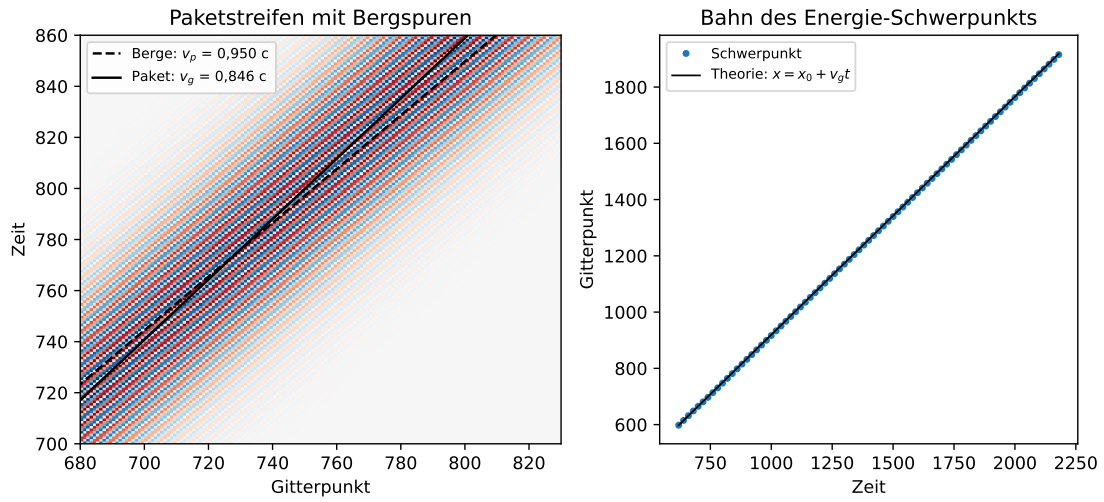


Abbildung 16.2: Links: x-t-Diagramm eines Ausschnitts (Farbe = u). Das Paket läuft als breiter Streifen entlang der durchgezogenen v_g-Theorielinie (0,846 c); die feinen Bergspuren darin sind flacher und folgen der gestrichelten v_p-Linie (0,950 c) — jeder Berg wandert im Paket nach vorn. Rechts: die Bahn des Energie-Schwerpunkts (Punkte) auf der v_g-Theoriegeraden.

Im linken Bild ist die Vorhersage mit bloßem Auge zu prüfen: Die feinen Bergspuren sind tatsächlich *flacher* als der Paketstreifen — jeder Berg überholt das Paket. Nun die Zahlen:

```
# von oben: reihe1, zeiten, u, bahn, vg_bahn, QUELLE, SIG,
#           SONDE1, SONDE2, VP_GITTER, VG_GITTER, N_GITTER
def ankunftszeit(reihe, zeiten, sigma):
```

```
    """Zeitlicher Energie-Schwerpunkt, im 2. Durchgang gefenstert."""
    e = reihe**2
    t_roh = np.sum(zeiten * e) / np.sum(e)
    m = np.abs(zeiten - t_roh) < 3.5 * sigma
    return np.sum(zeiten[m] * e[m]) / np.sum(e[m])

def omega_aus_zeitreihe(reihe, zeiten, sigma):
    """Mittlere Periode aus den Nulldurchgängen ums Paketzentrum."""
    e = reihe**2
    tm = np.sum(zeiten * e) / np.sum(e)
    m = np.abs(zeiten - tm) < 2.0 * sigma
    r, z = reihe[m], zeiten[m]
    w = np.where(np.diff(np.sign(r)) > 0)[0]
    tn = [z[i] - r[i] * (z[i + 1] - z[i]) / (r[i + 1] - r[i]) for i in w]
    return 2 * np.pi / np.mean(np.diff(tn))

def k_aus_schnappschuss(u, mitte, halbbreite=40):
    """Mittlere Wellenlänge aus den Nulldurchgängen ums Paket."""
    f = u[mitte - halbbreite:mitte + halbbreite]
    xs = np.arange(mitte - halbbreite, mitte + halbbreite, dtype=float)
    w = np.where(np.diff(np.sign(f)) > 0)[0]
    xn = [xs[i] - f[i] / (f[i + 1] - f[i]) for i in w]
    return 2 * np.pi / np.mean(np.diff(xn))

w_mess = omega_aus_zeitreihe(reihe1, zeiten, SIG)
e = u**2
e[:QUELLE + 80] = 0.0
mitte = int(np.sum(np.arange(N_GITTER) * e) / np.sum(e))
k_mess = k_aus_schnappschuss(u, mitte)
t1 = ankunftszeit(reihe1, zeiten, SIG)
t2 = ankunftszeit(reihe2, zeiten, SIG)

print(f"Berge:  v_p = ω/k = {w_mess:.4f}/{k_mess:.4f} "
      f"= {w_mess/k_mess:.4f} c   (Theorie {VP_GITTER:.4f})")
print(f"Paket:  v_g (Schwerpunktbahn)  = {vg_bahn:.4f} c   "
      f"(Theorie {VG_GITTER:.4f})")
print(f"Paket:  v_g (Ankunftszeiten)   = "
      f"{(SONDE2 - SONDE1)/(t2 - t1):.4f} c")
```

```
Berge:  v_p = ω/k = 1.1958/1.2611 = 0.9482 c   (Theorie 0.9495)
Paket:  v_g (Schwerpunktbahn)  = 0.8455 c   (Theorie 0.8464)
Paket:  v_g (Ankunftszeiten)   = 0.8460 c
```

Drei Messungen, drei Treffer auf etwa ein Promille. Das Gitter *ist* ein dispersives Medium — nur eben ein künstliches, dessen $\omega(k)$ wir aus dem Algorithmus hergeleitet haben statt aus der Natur. Merke dir die beiden Formeln dieses Abschnitts gut: Die Gegenprobe „messen, was die Gittertheorie vorhersagt" wird uns gleich noch einmal retten, wenn künstliche und echte Dispersion im selben Experiment auftreten.

16.3 Warum Glas sortiert: der Lorentz-Oszillator

Woher kommt *echte* Dispersion? Kapitel 8 hat es angedeutet: Das Vakuum kann Wellen nicht nach Länge behandeln, weil es keinen eigenen Maßstab besitzt — Glas aber hat einen. Seine Elektronen sind an die Atome **gebunden** wie Massen an Federn, und ein Masse-Feder-System ist ein Resonator mit einer Eigenfrequenz f_0 (Kapitel 15 hat gerade ausführlich vermessen, was Resonatoren tun). Fährt eine Lichtwelle der Frequenz f hindurch, dann *treibt* ihr E-Feld diese Elektronen — eine erzwungene Schwingung. Und die Antwort eines getriebenen Resonators hängt von der Treibfrequenz ab: weit unterhalb von f_0 schwingen die Elektronen brav im Takt mit; je näher f an die Resonanz rückt, desto heftiger fällt die Antwort aus. Die mitschwingenden Elektronen sind ihrerseits winzige Antennen, deren Strahlung sich der Welle überlagert und sie verlangsamt — **stärkere Antwort heißt größeres** n. Das Ergebnis dieser Überlegung (die Rechnung ist die einer getriebenen Feder und steht im Kleingedruckten) heißt **Lorentz-Modell**:

$$\varepsilon(f) \;=\; 1 + \frac{\sigma f_0^2}{f_0^2 - f^2}, \qquad n(f) = \sqrt{\varepsilon(f)},$$

wobei σ die Stärke der Materialantwort misst (wie viele Elektronen mitschwingen) und f_0 die Resonanzfrequenz der Federn ist. Genau in dieser Form — als `LorentzianSusceptibility` — nimmt Meep dispersive Materialien entgegen; die Dämpfung γ, die Kapitel 15 zur Lorentz-*Linie* gehörte, lassen wir hier null (unser Glas soll nicht schlucken, nur sortieren).

Wir bauen uns ein Buchglas: $f_0 = 2$ und $\sigma = 1{,}25$. Die Handrechnung zeigt, dass das vernünftige Werte sind: Bei $f \to 0$ ist $\varepsilon = 1 + 1{,}25 = 2{,}25$, also $n = 1{,}5$ — exakt das Glas, das uns seit Kapitel 6 begleitet. Bei $f = 0{,}8$ (nennen wir es **Rot**) ist $\varepsilon = 1 + 1{,}25 \cdot 4/(4 - 0{,}64) = 2{,}4881$, $n = 1{,}5774$. Bei $f = 1{,}2$ (**Blau**) ist $\varepsilon = 1 + 5/2{,}56 = 2{,}9531$, $n = 1{,}7185$. Der Brechungsindex **wächst** mit der Frequenz, solange wir unterhalb der Resonanz bleiben — das heißt **normale Dispersion**, und es ist der Normalfall durchsichtiger Stoffe: Ihre Elektronen-Resonanzen liegen im Ultravioletten, das sichtbare Licht treibt sie alle von unten.

Für die Gruppengeschwindigkeit brauchen wir die Steigung. Mit $k = 2\pi f\, n(f)/c$ liefert die Produktregel $dk/df = (2\pi/c)\,(n + f\, dn/df)$, also

$$v_g = \frac{c}{n_g}, \qquad n_g = n + f\,\frac{dn}{df}$$

— der **Gruppenindex** n_g: das gewohnte n plus einen Zuschlag für die Steigung. Handrechnung für unser Glas: Bei $f = 0{,}8$ ist $dn/df = 0{,}2246$ (man kann numerisch ableiten), also $n_g = 1{,}5774 + 0{,}8 \cdot 0{,}2246 = 1{,}757$ und $v_g = 0{,}5691\,c$ gegenüber $v_p = c/n = 0{,}6340\,c$. Bei $f = 1{,}2$: $n_g = 2{,}358$, $v_g = 0{,}4241\,c$ gegenüber $v_p = 0{,}5819\,c$. Zwei Vorhersagen fürs Protokoll: **Blau ist in beiden Tempi das langsamere Licht** — und der Abstand zwischen v_p und v_g ist bei Blau größer, weil die $n(f)$-Kurve dort steiler ist. So sehen Modell und Kurven aus:

```
# von oben: np, plt
import os
os.environ["HWLOC_HIDE_ERRORS"] = "2"   # vor dem Meep-Import (Kapitel 11)
import meep as mp

mp.verbosity(0)                          # Fortschrittsmeldungen stumm

F0_GLAS, ST_GLAS = 2.0, 1.25
# ein dispersives Meep-Material: epsilon=1 plus Lorentz-Term
# (frequency = f0, sigma = Stärke, gamma = Dämpfung — hier 0)
GLAS = mp.Medium(epsilon=1, E_susceptibilities=[
    mp.LorentzianSusceptibility(frequency=F0_GLAS, gamma=0.0,
                                sigma=ST_GLAS)])

def eps_glas(f):
    return 1 + ST_GLAS * F0_GLAS**2 / (F0_GLAS**2 - f**2)

def n_glas(f):
    return np.sqrt(eps_glas(f))

def n_gruppe(f, df=1e-6):
    return n_glas(f) + f * (n_glas(f + df) - n_glas(f - df)) / (2 * df)

fig, (links, rechts) = plt.subplots(1, 2, figsize=(9, 3.6))
f1 = np.linspace(0.01, 1.93, 400)
f2 = np.linspace(2.07, 4.5, 400)
links.plot(f1, eps_glas(f1), "C0")
```

```
links.plot(f2, eps_glas(f2), "C0")
links.axvspan(2.0, 3.0, color="0.85")        # ε < 0: verbotenes Fenster
links.axhline(0, color="k", lw=0.6)
links.axvline(2.0, color="k", ls=":", lw=1)
links.annotate("$f_0$", (2.02, 18))
links.annotate("$\\varepsilon < 0$", (2.35, 8))
links.set(xlabel="Frequenz f", ylabel="$\\varepsilon(f)$",
          ylim=(-12, 24), title="Materialantwort mit Resonanz")

fz = np.linspace(0.01, 1.55, 300)
rechts.plot(fz, n_glas(fz), "C0", label="n(f)")
rechts.plot(fz, [n_gruppe(f) for f in fz], "C0--",
            label="$n_g = n + f\\,dn/df$")
for f, farbe, name in ((0.8, "tab:red", "Rot"),
                       (1.2, "tab:blue", "Blau")):
    rechts.plot(f, n_glas(f), "o", color=farbe)
    rechts.annotate(name, (f, n_glas(f)),
                    textcoords="offset points", xytext=(8, -4),
                    color=farbe)
# Meeps eigene Auskunft über das Material als Gegenprobe
f_probe = np.array([0.4, 0.8, 1.2, 1.5])
eps_meep = [GLAS.epsilon(f)[0][0].real for f in f_probe]
rechts.plot(f_probe, np.sqrt(eps_meep), "x", color="k", ms=7,
            label="Meep: Medium.epsilon(f)")
rechts.set(xlabel="Frequenz f", ylabel="Brechungsindex",
           ylim=(1.3, 4.2), title="n und $n_g$ im Transparenzbereich")
rechts.legend(fontsize=8)
fig.tight_layout()
plt.show()
```

```
Using MPI version 4.1, 1 processes
```

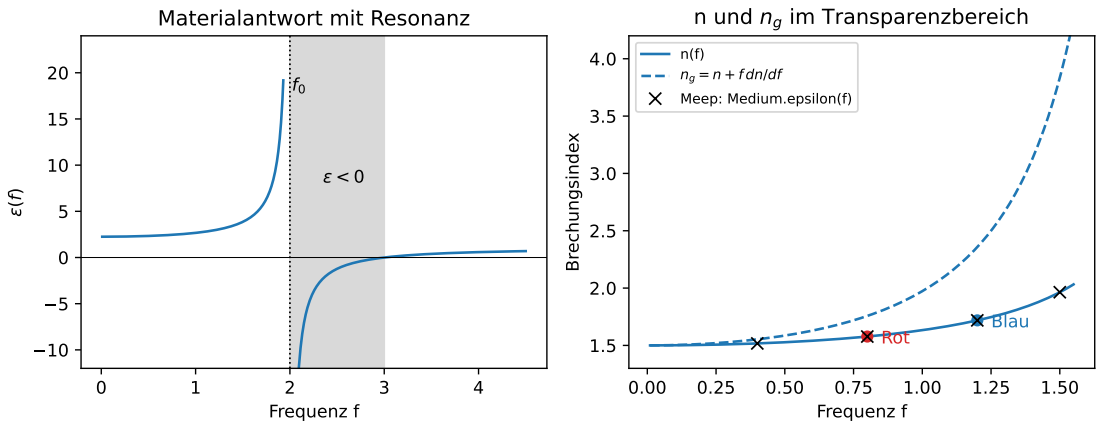


Abbildung 16.3: Das Lorentz-Glas (f_0 = 2, σ = 1,25). Links: ε(f) schießt an der Resonanz in die Höhe, dahinter folgt ein Fenster mit ε < 0 (grau), in dem keine Welle laufen kann. Rechts: Zoom in den transparenten Bereich — Brechungsindex n (durchgezogen) und Gruppenindex n_g (gestrichelt) wachsen mit f; die Punkte markieren unser Rot (f = 0,8) und Blau (f = 1,2). Die Kreuze sind Meeps eigene Auskunft über das Material — sie liegen exakt auf unserer Formel.

Zwei Dinge am linken Panel verdienen einen zweiten Blick. Erstens: Genau *an* der Resonanz explodiert die Antwort — unsere ungedämpfte Formel hat dort einen Pol (echte Materialien glätten ihn mit $\gamma > 0$ zur Lorentz-Linie aus Kapitel 15 und **absorbieren** dort: Resonanz heißt schlucken, nicht durchlassen). Zweitens: Direkt oberhalb von f_0 ist ε **negativ** — dort gibt es kein reelles n, keine laufende Welle, nur evaneszentes Abklingen wie hinter der Totalreflexionsgrenze in Kapitel 12. Dieses graue Fenster ist eine Miniaturausgabe dessen, was uns im Drude-Abschnitt und in Kapitel 26 (Bandlücken) wieder begegnet: **Frequenzbereiche, in denen ein Medium schlicht „nein" sagt.**

16.3.1 Das Wettrennen

Jetzt wird gemessen. **Frage:** Treffen die gemessenen v_g und v_p unsere vier Vorhersagen — und gewinnt Rot das Rennen? **Bühne:** ein Streifen von 52×2 Meep-Einheiten, ganz mit Lorentz-Glas gefüllt (`default_material`); PML nur links und rechts. Neu ist die Zutat `k_point = (0,0,0)`: Sie erklärt den oberen und unteren Zellrand für **periodisch** — die Zelle wird zum Schlauch zusammengeklebt, und eine Quelllinie über die volle Höhe erzeugt eine saubere

ebene Welle, als wäre der Streifen unendlich breit. **Anregung:** ein Gauß-Puls (`GaussianSource`, Bandbreite $f/12$ — wieder bewusst schmalbandig) auf einer Linie bei $x = -22$. **Gemessen wird** an zwei Sonden bei $x = -15$ und $x = +10$ (Abstand 25): die Ankunftszeit des Pakets je Sonde als zeitlicher Energie-Schwerpunkt (dasselbe Werkzeug wie eben) — daraus v_g; und an einem **engen Sondenpaar** (Abstand $0{,}15$, weniger als eine halbe Glas-Wellenlänge) die Phase des Trägers per Hand-Detektor aus Kapitel 10 (Zeitreihe mal cos/sin, `arctan2`): Die Phasendifferenz ist $k \cdot 0{,}15$, daraus $v_p = \omega/k$. **Erfolgskriterium:** ein erster Lauf mit Auflösung 20 — und wir prüfen *ehrlich*, wie nah wir der Materialtheorie kommen.

```
# von oben: mp, np, GLAS, ankunftszeit() (Abschnitt 16.2)
DT_SONDE = 0.05
ABSTAND_WEIT = 25.0                          # Sonde 1 -> Sonde 2
ABSTAND_ENG = 0.15                           # Phasen-Paar

def streifenlauf(medium, f, dauer, aufloesung):
    """Puls durch den periodischen Streifen; vier Sonden-Zeitreihen."""
    quelle = [mp.Source(mp.GaussianSource(f, fwidth=f / 12),
                        component=mp.Ez, center=mp.Vector3(-22, 0),
                        size=mp.Vector3(0, 2))]
    sim = mp.Simulation(cell_size=mp.Vector3(52, 2),
                        resolution=aufloesung, sources=quelle,
                        default_material=medium,   # füllt die ganze Zelle
                        boundary_layers=[mp.PML(2.0, direction=mp.X)],
                        k_point=mp.Vector3())   # oben/unten periodisch
    sonden = [mp.Vector3(-15, 0), mp.Vector3(10, 0),
              mp.Vector3(10, 0), mp.Vector3(10 + ABSTAND_ENG, 0)]
    reihen = [[] for _ in sonden]

    def messen(sim):
        for r, p in zip(reihen, sonden):
            r.append(sim.get_field_point(mp.Ez, p).real)

    sim.run(mp.at_every(DT_SONDE, messen), until=dauer)
    return [np.array(r) for r in reihen]

def detektor_phase(reihe, zeiten, f):
    """Kapitel-10-Detektor: Phase der Frequenz f in der Zeitreihe."""
    co = np.sum(reihe * np.cos(2 * np.pi * f * zeiten))
    si = np.sum(reihe * np.sin(2 * np.pi * f * zeiten))
    # np.arctan2(y, x): Vier-Quadranten-Arcustangens — liefert den
    # Winkel des Punkts (x, y) im vollen Kreis (-pi bis pi), weil er
    # beide Vorzeichen kennt (arctan(y/x) allein kann das nicht)
```

```
    return np.arctan2(si, co)

def tempi_messen(medium, f, dauer, aufloesung):
    """v_g aus Ankunftszeiten, v_p aus der Phasendifferenz."""
    reihen = streifenlauf(medium, f, dauer, aufloesung)
    zeiten = np.arange(len(reihen[0])) * DT_SONDE
    sigma = (12 / f) / 2                    # grobe Paketbreite
    t1 = ankunftszeit(reihen[0], zeiten, sigma)
    t2 = ankunftszeit(reihen[1], zeiten, sigma)
    dphi = (detektor_phase(reihen[3], zeiten, f)
            - detektor_phase(reihen[2], zeiten, f)) % (2 * np.pi)
    vp = 2 * np.pi * f * ABSTAND_ENG / dphi
    return ABSTAND_WEIT / (t2 - t1), vp, (reihen, zeiten, t1, t2)

lauf_rot = tempi_messen(GLAS, 0.8, 180, 20)
lauf_blau = tempi_messen(GLAS, 1.2, 200, 20)
for name, f, lauf in (("Rot ", 0.8, lauf_rot), ("Blau", 1.2, lauf_blau)):
    vg, vp = lauf[0], lauf[1]
    vg_soll, vp_soll = 1 / n_gruppe(f), 1 / n_glas(f)
    print(f"{name} f = {f}:  v_g = {vg:.4f} c  (Material {vg_soll:.4f}, "
          f"{(vg/vg_soll - 1)*100:+.1f} %)   v_p = {vp:.4f} c  "
          f"(Material {vp_soll:.4f}, {(vp/vp_soll - 1)*100:+.1f} %)")
```

```
Rot  f = 0.8:  v_g = 0.5591 c  (Material 0.5691, -1.8 %)  v_p = 0.6302 c  (Material 0.6340, -0.6
Blau f = 1.2:  v_g = 0.4038 c  (Material 0.4241, -4.8 %)  v_p = 0.5726 c  (Material 0.5819, -1.
```

Rot gewinnt das Rennen, qualitativ stimmt alles — aber quantitativ sollte uns das Ergebnis stören: Blau verfehlt die v_g-Theorie um fast **fünf Prozent**, weit mehr, als unsere Messwerkzeuge an Streuung hergeben. Bevor man in so einem Moment am Lorentz-Modell zweifelt, lohnt der Blick auf die Bühne: Wer ist hier eigentlich *noch* dispersiv? Das Gitter. Auflösung 20 bedeutet für Blau im Glas eine Wellenlänge von $\lambda = 1/(1{,}2 \cdot 1{,}7185) = 0{,}485$, also nur **9,7 Gitterpunkte pro Wellenlänge** — grob genug, dass die Gitterformel aus dem letzten Abschnitt mitspielt. Rechnen wir ihre Vorhersage aus: Beide Dispersionen wirken in guter Näherung *nacheinander* (das Material verlangsamt auf c/n_g, das Gitter multipliziert seinen cos / cos-Faktor obendrauf):

```
# von oben: np, n_glas(), n_gruppe()
def gitterfaktor(f, aufloesung, courant=0.5):
    """Kapitel-8-Korrektur für v_g im Medium mit Index n(f)."""
    dx = 1 / aufloesung
```

```
    dt = courant * dx
    k_med = 2 * np.pi * f * n_glas(f)        # Wellenzahl im Glas
    return (np.cos(k_med * dx / 2)
            / np.cos(2 * np.pi * f * dt / 2))

for name, f, lauf in (("Rot ", 0.8, lauf_rot), ("Blau", 1.2, lauf_blau)):
    kombiniert = gitterfaktor(f, 20) / n_gruppe(f)
    print(f"{name}: Material x Gitter = {kombiniert:.4f} c   "
          f"gemessen: {lauf[0]:.4f} c")
```

```
Rot : Material x Gitter = 0.5591 c   gemessen: 0.5591 c
Blau: Material x Gitter = 0.4039 c   gemessen: 0.4038 c
```

Die kombinierte Vorhersage trifft die Messung auf **vier Stellen** — die fehlenden fünf Prozent waren bis auf den letzten Krümel numerische Dispersion. Das ist die wichtigste Methodenlektion dieses Kapitels: **Physikalische und numerische Dispersion addieren sich im selben Experiment**, und wer die zweite nicht kennt (oder nicht beziffern kann), misst die erste falsch und merkt es nicht. Wir können sie beziffern — und wir können sie kleiner machen: Auflösung verdoppeln, Gegenprobe (die Konvergenzleiter aus Kapitel 10):

```
# von oben: tempi_messen(), gitterfaktor(), n_gruppe(), n_glas(), GLAS
lauf_rot40 = tempi_messen(GLAS, 0.8, 180, 40)
lauf_blau40 = tempi_messen(GLAS, 1.2, 200, 40)
for name, f, lauf in (("Rot ", 0.8, lauf_rot40),
                      ("Blau", 1.2, lauf_blau40)):
    vg, vp = lauf[0], lauf[1]
    print(f"{name}: v_g = {vg:.4f} c  (Material {1/n_gruppe(f):.4f}, "
          f"kombiniert {gitterfaktor(f, 40)/n_gruppe(f):.4f})   "
          f"v_p = {vp:.4f} c  (Material {1/n_glas(f):.4f})")
```

```
Rot : v_g = 0.5666 c  (Material 0.5691, kombiniert 0.5666)  v_p = 0.6330 c  (Material 0.6340)
Blau: v_g = 0.4190 c  (Material 0.4241, kombiniert 0.4190)  v_p = 0.5796 c  (Material 0.5819)
```

Bei Auflösung 40 schrumpft der Gitteranteil auf rund ein Prozent (der Faktor steckt im Quadrat der Punktdichte, Kapitel 8), und wieder liegt die Messung auf der kombinierten Theorie. Das Wettrennen im Bild — Rot braucht für die 25 Einheiten zwischen den Sonden 44 Zeiteinheiten, Blau 60:

```
# von oben: lauf_rot40, lauf_blau40, plt, np
fig, achsen = plt.subplots(2, 1, figsize=(8.5, 4.6), sharex=True)
for achse, name, farbe, lauf in (
        (achsen[0], "Rot (f = 0,8)", "tab:red", lauf_rot40),
        (achsen[1], "Blau (f = 1,2)", "tab:blue", lauf_blau40)):
    (reihen, zeiten, t1, t2) = lauf[2]
    for reihe, stil in ((reihen[0], 0.45), (reihen[1], 1.0)):
        achse.plot(zeiten, reihe / np.abs(reihe).max(), color=farbe,
                   alpha=stil, lw=0.7)
    for t in (t1, t2):
        achse.axvline(t, color="k", ls=":", lw=1)
    achse.annotate("", xy=(t2, 1.25), xytext=(t1, 1.25),
                   arrowprops=dict(arrowstyle="->"),
                   annotation_clip=False)
    achse.annotate(f"Laufzeit {t2 - t1:.1f}",
                   ((t1 + t2) / 2, 1.32), ha="center",
                   annotation_clip=False)
    achse.set(ylabel="$E_z$ (norm.)", ylim=(-1.15, 1.15))
    achse.annotate(name, (0.015, 0.8), xycoords="axes fraction",
                   color=farbe)
achsen[1].set_xlabel("Zeit")
fig.tight_layout()
plt.show()
```

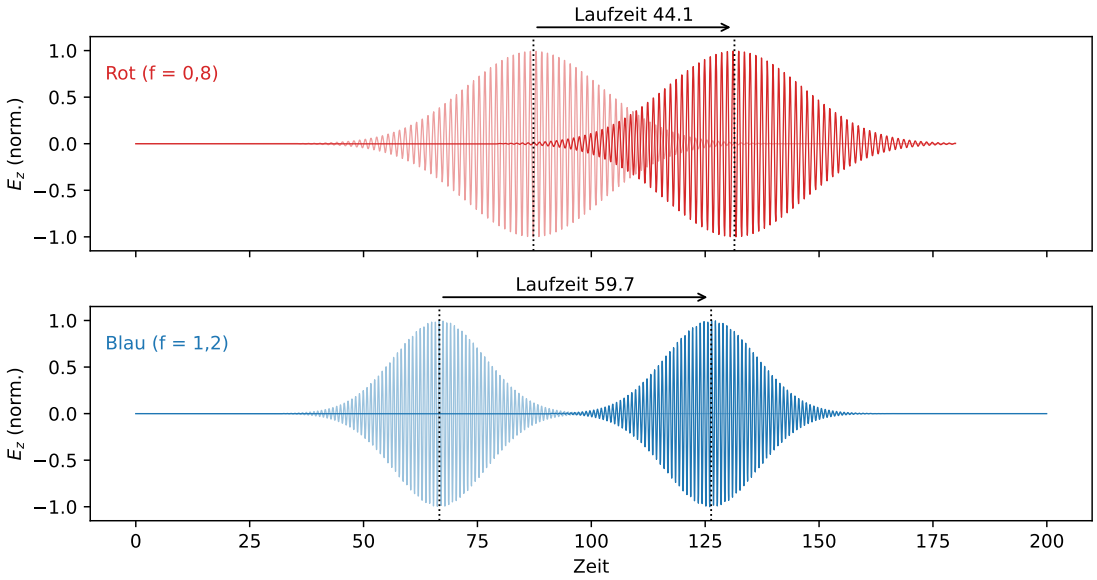


Abbildung 16.4: Das Wettrennen durchs Lorentz-Glas (Auflösung 40): die Zeitreihen beider Sonden (links x = −15, rechts x = +10) für Rot (oben) und Blau (unten), Amplituden normiert. Die senkrechten Linien markieren die gemessenen Ankunftszeiten (Energie-Schwerpunkte); die Pfeile zeigen die daraus folgende Laufzeit für 25 Einheiten Strecke. Rot braucht 44,1 Zeiteinheiten, Blau 59,7 — Blau ist das langsame Licht im Glas. Zusätzlich sieht man Blau stärker zerfließen: seine Bandbreite erwischt ein steileres Stück der n(f)-Kurve.

Damit ist das Versprechen aus Kapitel 15 eingelöst: Die Lorentz-Kurve, die dort die *Linienform* eines Resonators war, ist hier als **Materialantwort** wieder aufgetaucht — Glas ist ein Chor aus Elektronen-Resonatoren, und $n(f)$ ist seine Resonanzkurve, von weit unten angespielt.

16.4 Das Prisma sortiert andersherum

Jetzt zum Kapitel-14-Versprechen. Ein Prisma lenkt Licht zweimal über Snellius ab — einmal beim Eintritt, einmal beim Austritt — und weil n von der Frequenz abhängt, hängt der gesamte Ablenkwinkel δ von der Farbe ab. Die Kette für ein Prisma mit Apexwinkel A (Einfallswinkel θ_1 auf die erste Fläche, alle Winkel zu den Flächennormalen):

$$\theta_2 = \arcsin\frac{\sin\theta_1}{n}, \qquad \theta_3 = A - \theta_2, \qquad \theta_4 = \arcsin\left(n\sin\theta_3\right), \qquad \delta = \theta_1 + \theta_4 - A.$$

Die Handrechnung für unser Buchglas, $A = 40°$ und $\theta_1 = 32{,}65°$ (gleich wird klar, warum gerade dieser Wert): **Rot** ($n = 1{,}5774$): $\theta_2 = \arcsin(0{,}5395/1{,}5774) = 20{,}0°$, $\theta_3 = 20{,}0°$, $\theta_4 = \arcsin(1{,}5774 \cdot 0{,}342) = 32{,}65°$, also $\delta_{\text{rot}} = 32{,}65 + 32{,}65 - 40 = \mathbf{25{,}30}°$ — der Strahlengang ist spiegelsymmetrisch ($\theta_2 = \theta_3$), das Prisma steht in **Minimalablenkung**, so justiert man es auch im Labor. **Blau** ($n = 1{,}7185$): $\theta_2 = 18{,}30°$, $\theta_3 = 21{,}70°$, $\theta_4 = 39{,}45°$, $\delta_{\text{blau}} = \mathbf{32{,}11}°$. Blau wird um fast sieben Grad stärker abgelenkt — größeres n, stärkerer Knick. Beim Gitter war es umgekehrt: $\sin\theta = \lambda/d$ wächst mit der **Wellenlänge**, dort gewann Rot. Gitter sortieren nach Geometrie, Prismen nach Materialantwort — und die läuft in die Gegenrichtung.

Das Experiment übernimmt das Spektrometer-Rezept aus Kapitel 14: **ein** breitbandiger Puls, zwei DFT-Fernfelder. **Frage:** Landen die beiden Farben bei $25{,}30°$ und $32{,}11°$ unter dem Horizont? **Bühne:** eine 30×28-Zelle, darin das Prisma als `mp.Prism` (drei Eckpunkte, Material: unser Lorentz-Glas). Wir **drehen das Prisma** um $-12{,}65°$, so dass der *horizontale* Strahl die linke Fläche unter $\theta_1 = 32{,}65°$ trifft — wie auf einem Prismentisch. (Der Strahl bleibt aus gutem Grund horizontal: Eine *gekippte* Breitband-Strahlquelle hat in Meep eine frequenzabhängige Richtung — Details im Kleingedruckten; den Treffpunkt legen wir etwas unter die Flächenmitte, damit das Strahlbündel nicht am Apex anstößt.) **Anregung:** `GaussianBeamSource` mit Pulszeitverlauf um $f = 1{,}0$, Bandbreite $0{,}7$ — deckt Rot und Blau zugleich ab; Strahltaille 3. **Gemessen wird** mit `add_near2far` auf einer Linie hinter dem Prisma, ausgewertet bei exakt $f = 0{,}8$ und $1{,}2$ (Rasterregel aus Kapitel 14: zwei Frequenzen liegen bei $f_{\text{cen}} \pm df/2$, also $f_{\text{cen}} = 1{,}0$, $df = 0{,}4$); das Fernfeld wird auf einem Bogen mit Radius 1000 abgetastet, der Gipfel mit Parabel-Feinschliff (Kapitel 10) geortet. **Erfolgskriterium:** beide Gipfel auf ein halbes Grad.

```
# von oben: mp, np, plt, GLAS, n_glas()
A_GRAD, L_FLAECHE = 40.0, 14.0

def ablenkung_theorie(f, theta1_grad):
    """Snellius-Kette durchs Prisma (Winkel in Grad)."""
    n = n_glas(f)
    # np.deg2rad/np.rad2deg: Grad <-> Bogenmaß (wie np.radians/
    # np.degrees aus Kapitel 1/4, nur als sprechendes Paar)
    t1 = np.deg2rad(theta1_grad)
    t2 = np.arcsin(np.sin(t1) / n)
```

```
    t3 = np.deg2rad(A_GRAD) - t2
    t4 = np.arcsin(n * np.sin(t3))
    return theta1_grad + np.rad2deg(t4) - A_GRAD

THETA1 = np.rad2deg(np.arcsin(n_glas(0.8)
                              * np.sin(np.deg2rad(A_GRAD / 2))))
DELTA_ROT = ablenkung_theorie(0.8, THETA1)    # 25,30°
DELTA_BLAU = ablenkung_theorie(1.2, THETA1)   # 32,11°

# Prisma-Ecken: Apex oben, dann um -(theta1 - A/2) gedreht
halber_apex = np.deg2rad(A_GRAD / 2)
ecken0 = [np.array([0.0, 0.0]),
          L_FLAECHE * np.array([-np.sin(halber_apex),
                                -np.cos(halber_apex)]),
          L_FLAECHE * np.array([+np.sin(halber_apex),
                                -np.cos(halber_apex)])]
beta = -np.deg2rad(THETA1 - A_GRAD / 2)
drehung = np.array([[np.cos(beta), -np.sin(beta)],
                    [np.sin(beta), np.cos(beta)]])
schwerpunkt0 = sum(ecken0) / 3
# @ zwischen Matrix und Vektor: Matrixprodukt (dreht den Vektor) —
# dieselbe Operation wie das Skalarprodukt aus Kapitel 1, eine Stufe größer
ECKEN = [drehung @ (e - schwerpunkt0) + np.array([1.0, 2.5])
         for e in ecken0]
TREFFPUNKT = ECKEN[0] + 0.60 * (ECKEN[1] - ECKEN[0])

# mp.Prism: beliebiges Vieleck aus Eckpunkten (height=inf: 2D-Profil)
prisma = mp.Prism(vertices=[mp.Vector3(*e) for e in ECKEN],
                  height=mp.inf, material=GLAS)
quelle = [mp.GaussianBeamSource(
    mp.GaussianSource(1.0, fwidth=0.7, is_integrated=True),
    center=mp.Vector3(TREFFPUNKT[0] - 8.0, TREFFPUNKT[1]),
    size=mp.Vector3(0, 8), beam_x0=mp.Vector3(0, 0),
    beam_kdir=mp.Vector3(1, 0), beam_w0=3.0,
    beam_E0=mp.Vector3(0, 0, 1), component=mp.ALL_COMPONENTS)]
sim = mp.Simulation(cell_size=mp.Vector3(30, 28), resolution=20,
                    sources=quelle, geometry=[prisma],
                    boundary_layers=[mp.PML(1.5)])
n2f = sim.add_near2far(1.0, 0.4, 2, mp.Near2FarRegion(
    center=mp.Vector3(9.0, -0.5), size=mp.Vector3(0, 24), weight=+1))

spur = {}
```

```
def aufsummieren(sim):
    ez2 = sim.get_array(component=mp.Ez).T ** 2
    spur["I"] = spur.get("I", 0) + ez2

sim.run(mp.at_every(0.5, aufsummieren), until_after_sources=80)

R_FERN = 1000.0
winkel = np.linspace(-50, 10, 400)
I_fern = np.zeros((2, len(winkel)))
for j, w in enumerate(winkel):
    wr = np.deg2rad(w)
    ff = sim.get_farfield(n2f, mp.Vector3(R_FERN * np.cos(wr),
                                          R_FERN * np.sin(wr)))
    for kf in range(2):
        I_fern[kf, j] = abs(ff[6 * kf + 2])**2   # E_z der Frequenz kf

def peakwinkel(I):
    j = np.argmax(I)
    p = np.polyfit(winkel[j - 4:j + 5], I[j - 4:j + 5], 2)
    return p[1] / (2 * p[0])                 # Grad unter dem Horizont

W_ROT, W_BLAU = peakwinkel(I_fern[0]), peakwinkel(I_fern[1])

fig, (links, rechts) = plt.subplots(
    1, 2, figsize=(9.5, 4.0), gridspec_kw={"width_ratios": [1.2, 1]})
helligkeit = np.sqrt(spur["I"])              # Wurzel: Spur sichtbarer
links.imshow(helligkeit, origin="lower", cmap="inferno",
             extent=[-15, 15, -14, 14],
             vmax=0.5 * helligkeit.max())
umriss = ECKEN + [ECKEN[0]]
links.plot([e[0] for e in umriss], [e[1] for e in umriss], "w-", lw=1)
austritt = np.array([4.3, 0.6])              # ungefährer Austrittspunkt
for delta, farbe in ((DELTA_ROT, "tab:red"), (DELTA_BLAU, "tab:blue")):
    d = np.deg2rad(delta)
    links.plot([austritt[0], austritt[0] + 12 * np.cos(d)],
               [austritt[1], austritt[1] - 12 * np.sin(d)],
               ls="--", color=farbe, lw=1.4)
links.set(xlim=(-15, 15), ylim=(-14, 14), xlabel="x", ylabel="y",
          title="Lichtspur mit Theoriestrahlen")

for kf, (farbe, name, soll, mess) in enumerate(
        (("tab:red", "Rot (f = 0,8)", DELTA_ROT, W_ROT),
         ("tab:blue", "Blau (f = 1,2)", DELTA_BLAU, W_BLAU))):
```

```
    rechts.plot(-winkel, I_fern[kf] / I_fern[kf].max(), color=farbe,
                label=f"{name}: {mess:.2f}°")
    rechts.axvline(soll, color=farbe, ls=":", lw=1.2)
rechts.set(xlabel="Winkel unter dem Horizont (Grad)",
           ylabel="Fernfeld-Intensität (norm.)", xlim=(10, 45),
           title="Ein Puls, zwei Farben (Soll gepunktet)")
rechts.legend(fontsize=8)
fig.tight_layout()
plt.show()
print(f"Gemessen: Rot {W_ROT:.2f}° (Soll {DELTA_ROT:.2f}°),  "
      f"Blau {W_BLAU:.2f}° (Soll {DELTA_BLAU:.2f}°)")
```

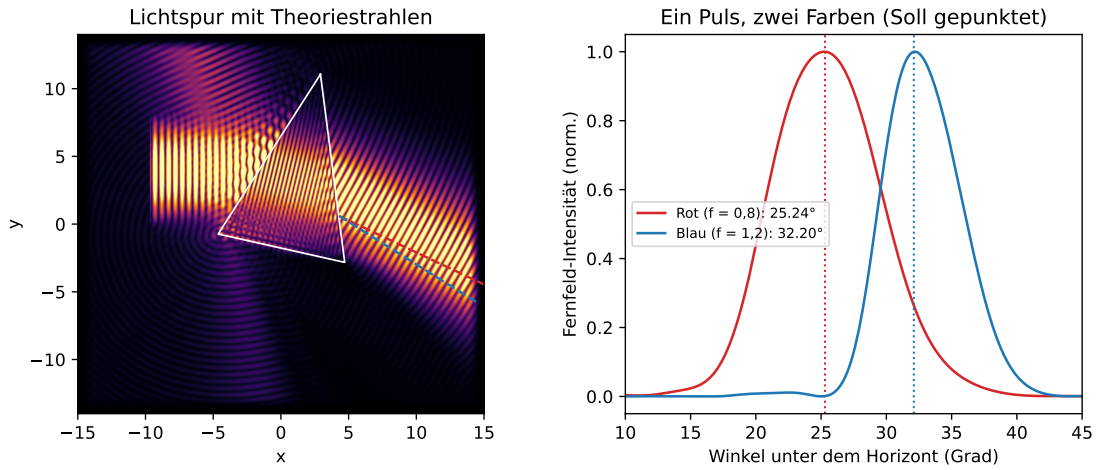


Abbildung 16.5: Links: die „Lichtspur“ des Pulses (über den ganzen Lauf aufsummiertes E_z²); das gedrehte Prisma (Umriss) knickt den horizontalen Strahl nach unten, die gestrichelten Theoriestrahlen zeigen die Sollrichtungen für Rot (25,3°) und Blau (32,1°) — der helle Fächer hinter dem Prisma liegt genau zwischen ihnen. Rechts: die beiden DFT-Fernfelder aus demselben Lauf, normiert; die gepunkteten Senkrechten sind die Snellius-Vorhersagen. Gemessen: Rot 25,24°, Blau 32,20° — das Prisma lenkt Blau stärker ab, genau andersherum als das Gitter aus Kapitel 14.

```
Gemessen: Rot 25.24° (Soll 25.30°),  Blau 32.20° (Soll 32.11°)
```

Beide Gipfel sitzen auf einem Zehntelgrad. Damit ist auch das Diamant-Versprechen aus Kapitel 12 fällig: Der zweite Trumpf des Brillanten — sein „Feuer”, das farbige Aufblitzen — ist genau diese Sortierung. Diamant hat

nicht nur ein großes n ($\approx 2{,}42$), sondern auch eine steile $n(f)$-Kurve; jede Facette wirkt als Mini-Prisma und zieht Weiß zu Spektren auseinander. Und der Regenbogen ist dasselbe Experiment mit Wassertropfen als Prismenersatz: Brechung hinein, Spiegelung an der Rückwand, Brechung heraus — Blau stets stärker geknickt.

Eine Gegenüberstellung als Gedankenstütze: Das **Gitter** sortiert über $d \sin\theta = m\lambda$ — reine Geometrie, Wellenlänge gegen Spaltabstand, **Rot** (großes λ) wird stärker abgelenkt. Das **Prisma** sortiert über $n(f)$ — Materialantwort, Resonatoren von unten angespielt, **Blau** (näher an f_0) wird stärker abgelenkt. Wer beide Muster kennt, liest an einem Spektrum sofort ab, welches Instrument es erzeugt hat.

16.5 Freie Elektronen: Drude, Metalle, Ionosphäre

Was passiert mit dem Lorentz-Modell, wenn man die **Feder durchschneidet**? In einem Metall sind die Leitungselektronen nicht an Atome gebunden — sie sind frei, ihre Rückstellkraft ist null, ihre „Resonanzfrequenz" ist $f_0 = 0$. Setzt man das in die Lorentz-Formel ein (mit $\sigma f_0^2 \to f_p^2$ als endlichem Grenzwert), bleibt das **Drude-Modell**:

$$\varepsilon(f) = 1 - \frac{f_p^2}{f^2},$$

wobei f_p — die **Plasmafrequenz** — die Eigenfrequenz ist, mit der die freie Elektronenwolke als Ganzes gegen die Ionen schwappt; sie wächst mit der Elektronendichte. Diese eine Zeile erklärt erstaunlich viel. **Unterhalb** von f_p ist $\varepsilon < 0$: kein reelles n, keine laufende Welle — die Welle dringt nur evaneszent ein (der Saum aus Kapitel 12) und wird vollständig reflektiert. *Darum spiegeln Metalle:* Für sichtbares Licht liegt f_p typischer Metalle im Ultravioletten, alles Sichtbare ist „unterhalb" und prallt ab. Und darum trägt die **Ionosphäre** Kurzwellenfunk um den Globus: Sie ist ein dünnes Plasma mit f_p von einigen Megahertz — Kurzwelle darunter wird himmelwärts gespiegelt wie von einem Metallblech, UKW und Satellitenfunk darüber fliegen ungehindert hindurch (Ü 16.4 misst diese Klippe nach). **Oberhalb** von f_p wird das Medium durchsichtig, aber mit einer Kuriosität: ε liegt zwischen 0 und 1, also ist $n < 1$ — und damit $v_p = c/n > c$. Die Handrechnung bei $f = 1{,}25\, f_p$: $\varepsilon = 1 - 1/1{,}5625 = 0{,}36$, $n = 0{,}6$, $v_p = 1{,}667\, c$. Zeit für das Kästchen.

⚠ Naheliegende Vermutung

„Nichts bewegt sich schneller als das Licht im Vakuum — also können auch Wellenberge in einem Medium höchstens mit c laufen; ein Medium mit $n < 1$ wäre ein Widerspruch zur Relativitätstheorie."
Warum sie naheliegt: „Nichts ist schneller als c" ist eines der bestbestätigten Prinzipien der Physik, und ein Wellenberg ist doch sichtbar ein „Etwas", das sich bewegt.
Was stattdessen stimmt: Die Relativität deckelt den Transport von **Energie und Information** — und beides reist, wie dieses Kapitel gemessen hat, mit der *Gruppen*geschwindigkeit. Ein Wellenberg dagegen trägt keine Botschaft: Er ist ein Wiederholungsmuster in einem Zug, der schon da ist — wie der Lichtfleck eines schwenkenden Leuchtturms, der über eine ferne Wolkenwand huschen kann, so schnell er will, ohne dass irgendetwas diese Strecke *durchquert*. Im Drude-Medium gilt sogar exakt $v_p \cdot v_g = c^2$ (nachzurechnen aus $n_g = 1/n$): Je schneller die Berge, desto langsamer das Paket. Die Messung folgt sofort — und die Vorderkante des Pulses, die *Front*, bleibt dabei immer unter c (das vertiefen Kapitel 20 und 29).

Gemessen wird mit exakt dem Werkzeug des Wettrennens — derselbe Streifen, dasselbe Sondenquartett, nur das Medium ist jetzt ein Drude-Material (Meep: `DrudeSusceptibility` mit $f_p = 1$) und der Träger liegt bei $f = 1{,}25$, sicher oberhalb der Plasmafrequenz. Erfolgskriterium: v_p deutlich über c und das Produkt $v_p \cdot v_g$ auf zwei Prozent an c^2. Zusätzlich zeichnen wir diesmal das ganze x-t-Diagramm des Streifens auf, denn dieses Bild muss man gesehen haben:

```
# von oben: mp, np, plt, DT_SONDE, ankunftszeit(), detektor_phase(),
#           ABSTAND_WEIT, ABSTAND_ENG
DRUDE = mp.Medium(epsilon=1, E_susceptibilities=[
    mp.DrudeSusceptibility(frequency=1.0, gamma=1e-6, sigma=1.0)])
F_DRUDE = 1.25
N_DRUDE = np.sqrt(1 - 1 / F_DRUDE**2)        # 0,6

quelle = [mp.Source(mp.GaussianSource(F_DRUDE, fwidth=F_DRUDE / 12),
                    component=mp.Ez, center=mp.Vector3(-22, 0),
                    size=mp.Vector3(0, 2))]
sim = mp.Simulation(cell_size=mp.Vector3(52, 2), resolution=40,
                    sources=quelle, default_material=DRUDE,
                    boundary_layers=[mp.PML(2.0, direction=mp.X)],
                    k_point=mp.Vector3())
```

```
sonden = [mp.Vector3(-15, 0), mp.Vector3(10, 0),
          mp.Vector3(10, 0), mp.Vector3(10 + ABSTAND_ENG, 0)]
reihen = [[] for _ in sonden]
xt_streifen = []

def messen(sim):
    for r, p in zip(reihen, sonden):
        r.append(sim.get_field_point(mp.Ez, p).real)
    if 40 <= sim.meep_time() <= 120:
        xt_streifen.append(sim.get_array(
            center=mp.Vector3(0, 0), size=mp.Vector3(52, 0),
            component=mp.Ez))

sim.run(mp.at_every(DT_SONDE, messen), until=170)

reihen = [np.array(r) for r in reihen]
zeiten = np.arange(len(reihen[0])) * DT_SONDE
sigma = (12 / F_DRUDE) / 2
t1 = ankunftszeit(reihen[0], zeiten, sigma)
t2 = ankunftszeit(reihen[1], zeiten, sigma)
vg_drude = ABSTAND_WEIT / (t2 - t1)
dphi = (detektor_phase(reihen[3], zeiten, F_DRUDE)
        - detektor_phase(reihen[2], zeiten, F_DRUDE)) % (2 * np.pi)
vp_drude = 2 * np.pi * F_DRUDE * ABSTAND_ENG / dphi

fig, achse = plt.subplots(figsize=(7.5, 4.6))
xt = np.array(xt_streifen)
achse.imshow(xt, origin="lower", aspect="auto", cmap="RdBu_r",
             extent=[-26, 26, 40, 120],
             vmin=-np.abs(xt).max(), vmax=np.abs(xt).max())
t0 = 80.0                                    # Anker: gemessene Paketbahn
x0 = 10.0 - vg_drude * (t2 - t0)
for v, stil, farbe, name in (
        (vp_drude, "--", "k", f"Berge: $v_p$ = {vp_drude:.2f} c"),
        (1.0, ":", "0.25", "Vakuumlicht: c"),
        (vg_drude, "-", "k", f"Paket: $v_g$ = {vg_drude:.2f} c")):
    achse.plot([x0 - v * 38, x0 + v * 38], [t0 - 38, t0 + 38],
               ls=stil, color=farbe, lw=1.4, label=name)
# eng zoomen, damit die Bergspuren im Druck auflösbar bleiben
achse.set(xlim=(-12, 20), ylim=(64, 96), xlabel="x", ylabel="Zeit",
          title="Drude-Medium: Berge schneller als c, Paket nicht")
achse.legend(loc="lower right", fontsize=8)
fig.tight_layout()
```

```
plt.show()
print(f"v_p = {vp_drude:.4f} c   (Theorie {1/N_DRUDE:.4f} c)")
print(f"v_g = {vg_drude:.4f} c   (Theorie {N_DRUDE:.4f} c)")
print(f"Produkt v_p·v_g = {vp_drude * vg_drude:.4f} c²")
```

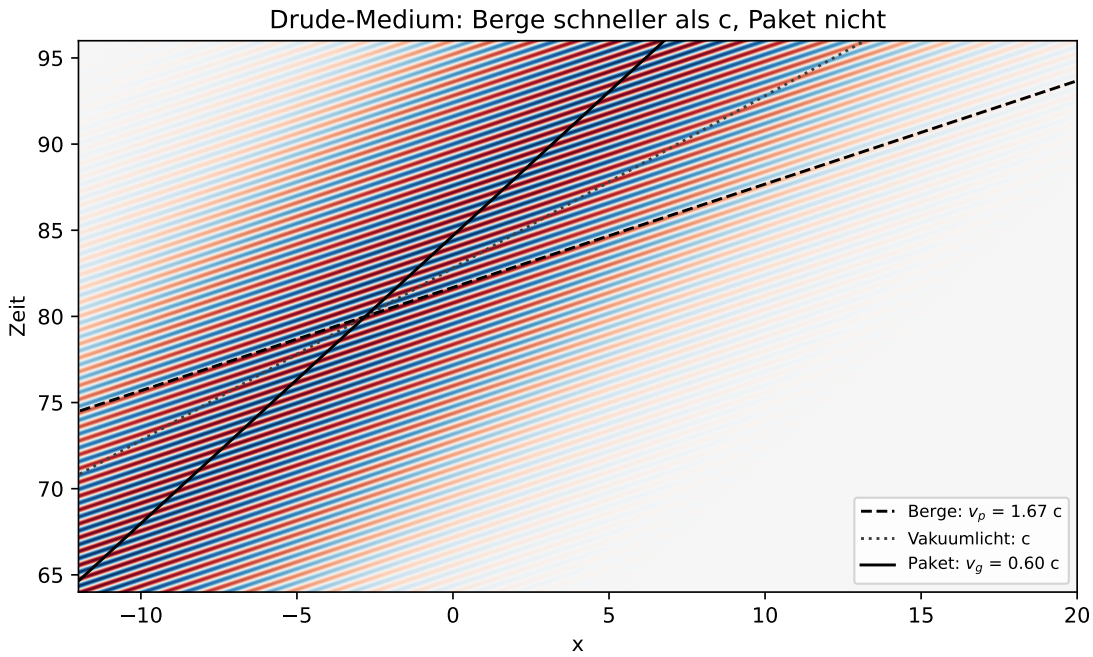


Abbildung 16.6: Ein Puls (f = 1,25 f_p) im Drude-Medium als x-t-Diagramm (Farbe = E_z entlang der Streifenachse); alle drei Theorielinien sind zum Vergleich durch denselben Punkt der gemessenen Paketbahn gelegt. Das Paket folgt der durchgezogenen v_g-Linie (0,60 c) und bleibt brav hinter der gepunkteten c-Linie zurück. Die feinen Bergspuren darin sind dagegen flacher als die c-Linie: Sie folgen der gestrichelten v_p-Linie mit 1,67 c — schneller als Vakuumlicht. Kein Widerspruch: Jeder einzelne Berg entsteht hinten im Paket, rast nach vorn und stirbt an der Front; keiner überholt sie je.

```
v_p = 1.6676 c   (Theorie 1.6667 c)
v_g = 0.5986 c   (Theorie 0.6000 c)
Produkt v_p·v_g = 0.9983 c²
```

Da steht es im Messprotokoll: Wellenberge mit $1{,}67\,c$, auf ein Promille an der Theorie — und gleichzeitig ein Paket, das mit $0{,}60\,c$ gemächlich unter

der c-Linie bleibt, Produkt c^2 auf zwei Promille. Die Berge im x-t-Diagramm zeigen das Leuchtturm-Argument grafisch: Jede Bergspur beginnt an der Hinterkante des Paketstreifens und endet an dessen Vorderkante. Schneller als c ist hier nur ein *Muster*, nie ein *Bote*.

16.6 Plasmonik: eine Goldscheibe unter Licht

Zum Finale tauschen wir das Modellmaterial gegen **echtes Gold**. Meep bringt eine Materialbibliothek mit (`meep.materials`), in der gemessene Metalle als Drude-Lorentz-Summen hinterlegt sind — Drude für die freien Elektronen, plus Lorentz-Terme für gebundene („Interband"-)Übergänge, gefittet an Messdaten. Die Bibliothek rechnet in Mikrometern: Wir setzen also $a = 1\,\mu\text{m}$ (Kapitel 11: eine Längenskala wählen, alles andere folgt), dann ist $f_{\text{meep}} = 1/\lambda[\mu\text{m}]$ und sichtbares Licht läuft von $f = 1{,}1$ (900 nm) bis $2{,}5$ (400 nm). Erst die Auskunft des Materials selbst:

```
# von oben: np, plt, mp
from meep.materials import Au

lam_nm = np.linspace(400, 900, 200)
eps_au = np.array([Au.epsilon(1 / (l / 1000))[0][0]
                   for l in lam_nm])

fig, (oben, unten) = plt.subplots(2, 1, figsize=(7, 4.4), sharex=True)
oben.plot(lam_nm, eps_au.real, "C0")
oben.axhline(0, color="k", ls="--", lw=0.8)
oben.set(ylabel="Re $\\varepsilon$", title="Gold: $\\varepsilon(\\lambda)$")
unten.plot(lam_nm, eps_au.imag, "C1")
unten.axhline(0, color="k", lw=0.6)
unten.set(xlabel="Wellenlänge (nm)", ylabel="Im $\\varepsilon$",
          ylim=(0, 6))
fig.tight_layout()
plt.show()
```

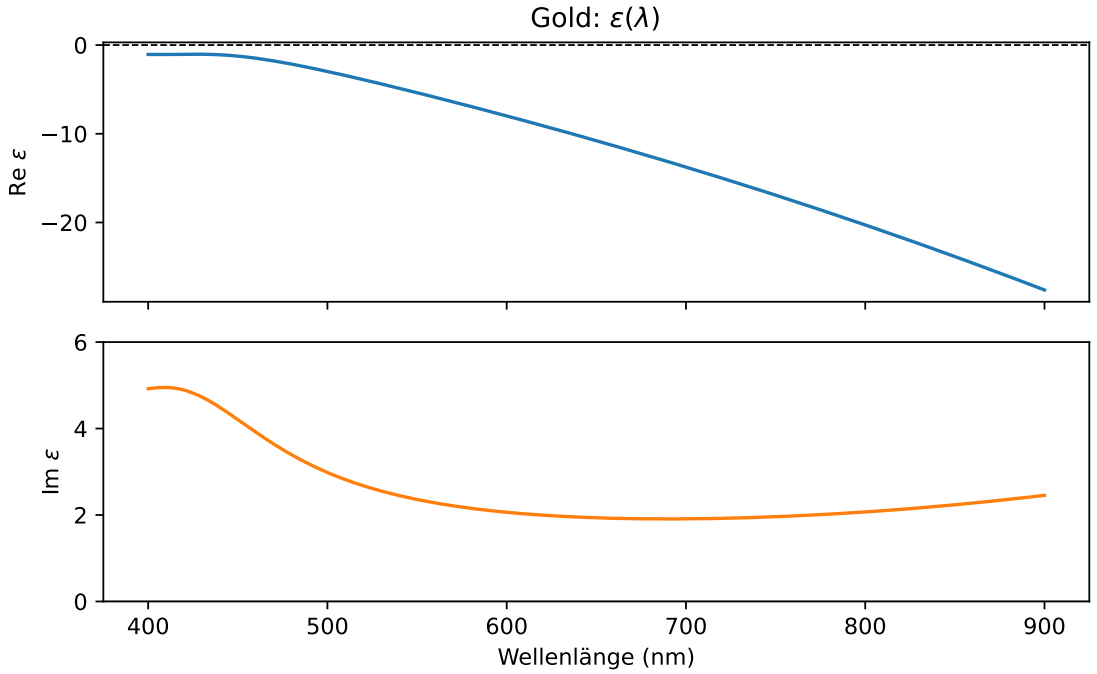


Abbildung 16.7: Die Permittivität von Gold aus Meeps Materialbibliothek (Drude-Lorentz-Fit an Messdaten), über der Vakuum-Wellenlänge. Oben: Re ε ist im gesamten Sichtbaren negativ (gestrichelt: die Null) — Gold ist überall „unterhalb der Plasmafrequenz“, darum spiegelt es. Unten: Im ε > 0 beschreibt Absorption; der Anstieg im Blau-Grünen stammt von den gebundenen Interband-Elektronen — sie schlucken Blau, weshalb der Spiegel goldgelb statt silbrig glänzt.

Ein massiver Goldspiegel wirft das Licht also zurück. Etwas ganz anderes passiert, wenn das Gold **kleiner als die Wellenlänge** wird — eine Scheibe von 100 nm Durchmesser, ein Fünftel der Lichtwellenlänge. Dann kann die Elektronenwolke der Scheibe *als Ganzes* gegen die Ionen schwingen: Das Licht schiebt sie nach links, die Oberflächenladungen an den Rändern ziehen sie zurück — eine Rückstellkraft, eine Eigenfrequenz, ein **Resonator** (Kapitel 15, nur dass hier Elektronen schwappen statt Felder im Hohlraum). Diese Schwingung heißt **lokalisiertes Oberflächenplasmon**, und wo Resonanz ist, ist starke Antwort: Bei der Resonanzfrequenz streut und schluckt die Scheibe Licht weit über ihre Größe hinaus.

Zwei Dinge brauchen wir fürs Experiment. Erstens die richtige **Polarisation**: Das E-Feld muss die Elektronen *quer über die Scheibe* schieben, damit Randladungen entstehen — also E in der Ebene, das ist die H_z-Welt aus Kapitel 13 (Quellkomponente `mp.Hz`; was mit der E_z-Polarisation passiert, klärt

Ü 16.5). Zweitens eine **Theorielinie**: Für den Kreiszylinder existiert die exakte Streulösung als Reihe aus Besselfunktionen — das zweidimensionale Gegenstück zur Mie-Theorie der Kugel von 1908. Wir tippen sie aus der Literatur ab (Bohren/Huffman, Kapitel 8; `scipy.special` liefert die Besselfunktionen `jv` und die Hankelfunktionen `hankel1`) und füttern sie mit demselben komplexen $\varepsilon(\lambda)$ aus Meeps Bibliothek — Theorie und Simulation nutzen exakt dasselbe Material:

```
# von oben: np, Au
# scipy.special: Sammlung „besonderer" Funktionen der mathematischen
# Physik; jv = Besselfunktion J_n, hankel1 = Hankelfunktion H_n^(1)
# (auslaufende Zylinderwelle – das 2D-Gegenstück zu e^{ikr}/r)
from scipy.special import jv, hankel1

def mie_zylinder(lam, radius, polarisation="Hz"):
    """Exakter Streuquerschnitt (pro Länge) eines Kreiszylinders
    bei senkrechtem Einfall (Bohren/Huffman, Kap. 8).
    'Hz': H parallel zur Achse (E in der Ebene) – Plasmonen-Fall;
    'Ez': E parallel zur Achse."""
    m = np.sqrt(Au.epsilon(1 / lam)[0][0])   # komplexer Index
    x = 2 * np.pi * radius / lam
    eta = m * x

    def dj(n, z):                            # Ableitungen über
        return 0.5 * (jv(n - 1, z) - jv(n + 1, z))   # Rekursionsformel

    def dh(n, z):
        return 0.5 * (hankel1(n - 1, z) - hankel1(n + 1, z))

    summe = 0.0
    for n in range(0, 16):
        if polarisation == "Hz":
            za = dj(n, eta) * jv(n, x) - m * jv(n, eta) * dj(n, x)
            ne = dj(n, eta) * hankel1(n, x) - m * jv(n, eta) * dh(n, x)
        else:
            za = m * dj(n, eta) * jv(n, x) - jv(n, eta) * dj(n, x)
            ne = m * dj(n, eta) * hankel1(n, x) - jv(n, eta) * dh(n, x)
        summe += (1 if n == 0 else 2) * abs(za / ne)**2
    return (2 / x) * summe * 2 * radius
```

Gemessen wird der **Streuquerschnitt** $\sigma(\lambda)$: wie viel Leistung die Scheibe aus einer ebenen Welle herausstreut, geteilt durch die einfallende Intensität — eine Fläche (in 2D: eine Länge), die man direkt mit der Scheibengröße vergleichen kann. Das Rezept braucht **zwei Läufe**, eine neue Tech-

nik: Erst ein **Normierungslauf** ohne Scheibe — er misst die einfallende Intensität und merkt sich die Felder auf einem Kasten aus vier Flussmonitoren um den (künftigen) Ort der Scheibe (`get_flux_data`). Dann der **Streulauf** mit Scheibe: Von seinen Monitoren ziehen wir die gemerkten einfallenden Felder ab (`load_minus_flux_data`) — was übrig bleibt, ist das *reine Streufeld*, und der Netto-Fluss aus dem Kasten heraus ist die gestreute Leistung. **Bühne:** $1{,}6 \times 1{,}6\,\mu\text{m}$, Auflösung 200 (die Scheibe ist nur 20 Zellen breit — Nanophotonik zwingt zu feinen Gittern), ebene Welle als Linienquelle mit `is_integrated=True` (nötig, weil die Linie bis in die PML reicht). Eine Vorsichtsmaßnahme übernehmen wir aus der Meep-Praxis: Bei dispersiven Bibliotheks-Metallen kann der Standard-Zeitschritt instabil werden, weil deren Fit-Terme sehr hohe Resonanzfrequenzen enthalten (die Stabilitätsgrenze aus Kapitel 8 galt nur für frequenz*un*abhängige Materialien); üblich ist `Courant = 0.2` — wir zahlen die 2,5-fachen Zeitschritte als Versicherungsprämie. **Erfolgskriterium:** Messpunkte auf der exakten Zylinder-Theorie.

Vor dem Lauf der Vorhersage-Moment:

> **Vorhersage-Punkt**
>
> Gold glänzt goldgelb — bei welcher Farbe erwartest du den stärksten Streupeak der 100-nm-Scheibe: bei Gold-Gelb (~580 nm), bei Rot, oder woanders?

Die Antwort gibt der Lauf; sie hängt nicht am Glanz des Spiegels, sondern an der Eigenfrequenz der schwappenden Elektronenwolke:

```
# von oben: mp, np, plt, Au, mie_zylinder()
R_SCHEIBE = 0.05                              # Radius 50 nm
F_GOLD = 1 / 0.55                             # Zentrum ~550 nm
DF_GOLD = 1 / 0.4 - 1 / 0.9                   # 400 bis 900 nm
NF_GOLD = 40

def goldaufbau(mit_scheibe, aufloesung, komponente=mp.Hz):
    # mp.Cylinder: Kreis(scheibe) – in 2D einfach ein runder Fleck
    geometrie = ([mp.Cylinder(radius=R_SCHEIBE, material=Au)]
                 if mit_scheibe else [])
    quelle = [mp.Source(
        mp.GaussianSource(F_GOLD, fwidth=DF_GOLD, is_integrated=True),
        component=komponente, center=mp.Vector3(-0.6, 0),
        size=mp.Vector3(0, 1.6))]
    return mp.Simulation(cell_size=mp.Vector3(1.6, 1.6),
                         resolution=aufloesung, sources=quelle,
                         geometry=geometrie,
```

```
                        boundary_layers=[mp.PML(0.4)], Courant=0.2)

def messkasten(sim):
    """Vier Flussmonitore als Kasten um die Scheibe (außen positiv)."""
    seiten = [sim.add_flux(F_GOLD, DF_GOLD, NF_GOLD, mp.FluxRegion(
        center=mp.Vector3(sx * 0.1, 0), size=mp.Vector3(0, 0.2),
        direction=mp.X, weight=sx)) for sx in (-1, +1)]
    seiten += [sim.add_flux(F_GOLD, DF_GOLD, NF_GOLD, mp.FluxRegion(
        center=mp.Vector3(0, sy * 0.1), size=mp.Vector3(0.2, 0),
        direction=mp.Y, weight=sy)) for sy in (-1, +1)]
    return seiten

def streuquerschnitt(aufloesung, komponente=mp.Hz):
    sim = goldaufbau(False, aufloesung, komponente)   # 1: Normierung
    kasten = messkasten(sim)
    einfall = sim.add_flux(F_GOLD, DF_GOLD, NF_GOLD, mp.FluxRegion(
        center=mp.Vector3(-0.1, 0), size=mp.Vector3(0, 0.2),
        direction=mp.X))
    sim.run(until_after_sources=30)
    norm_daten = [sim.get_flux_data(k) for k in kasten]
    P_einfall = np.array(mp.get_fluxes(einfall))
    freqs = np.array(mp.get_flux_freqs(einfall))

    sim = goldaufbau(True, aufloesung, komponente)    # 2: Streuung
    kasten = messkasten(sim)
    for k, daten in zip(kasten, norm_daten):
        sim.load_minus_flux_data(k, daten)   # Einfallsfeld abziehen
    sim.run(until_after_sources=30)
    P_streu = sum(np.array(mp.get_fluxes(k)) for k in kasten)
    return 1 / freqs, P_streu / (P_einfall / 0.2)    # λ, σ

lam200, sigma200 = streuquerschnitt(200)
lam300, sigma300 = streuquerschnitt(300)
lam_fein = np.linspace(0.40, 0.90, 200)
sigma_mie = np.array([mie_zylinder(l, R_SCHEIBE) for l in lam_fein])

fig, achse = plt.subplots(figsize=(7, 4.2))
achse.axvspan(495, 570, color="green", alpha=0.10)
achse.plot(lam_fein * 1000, sigma_mie * 1000, "k-", lw=1.2,
           label="Zylinder-Streutheorie (exakt)")
achse.plot(lam200 * 1000, sigma200 * 1000, "o", ms=4, mfc="none",
           color="goldenrod", label="Meep, Auflösung 200")
achse.plot(lam300 * 1000, sigma300 * 1000, "o", ms=4,
```

```
            color="darkgoldenrod", label="Meep, Auflösung 300")
achse.set(xlabel="Wellenlänge (nm)",
          ylabel="Streuquerschnitt (nm pro nm Länge)",
          title="100-nm-Goldscheibe: Plasmonen-Resonanz im Grünen")
achse.legend(fontsize=8)
fig.tight_layout()
plt.show()

i_peak = np.argmax(sigma300)
for name, lam, sig in (("Auflösung 200", lam200, sigma200),
                       ("Auflösung 300", lam300, sigma300)):
    mie_dort = np.array([mie_zylinder(l, R_SCHEIBE) for l in lam])
    abw = np.abs(sig / mie_dort - 1)
    print(f"{name}: Abweichung von der Theorie im Mittel "
          f"{np.mean(abw)*100:.1f} %, maximal {np.max(abw)*100:.1f} %")
print(f"Peak bei λ = {lam300[i_peak]*1000:.0f} nm — im Grünen")
```

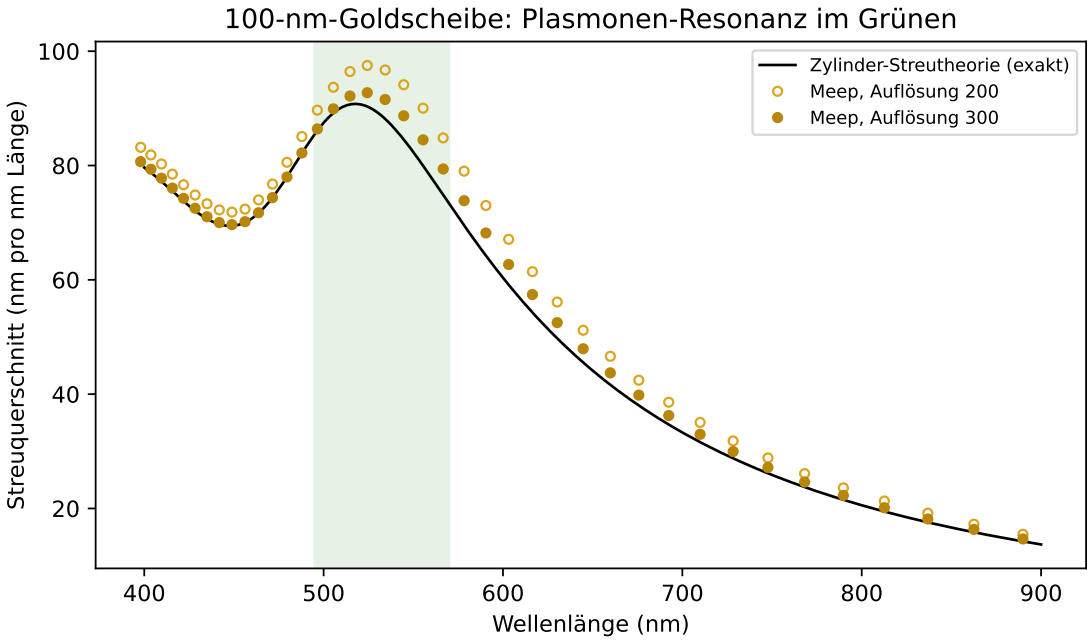


Abbildung 16.8: Streuquerschnitt der 100-nm-Goldscheibe über der Wellenlänge: Meep-Messpunkte bei Auflösung 200 (offene Kreise) und 300 (gefüllte), dazu die exakte Zylinder-Streutheorie (Linie) mit demselben $\varepsilon(\lambda)$. Der Peak liegt bei ~520 nm — im Grünen (Farbband), nicht beim Gold-Gelb des Spiegels. Die 200er-Punkte liegen systematisch ein paar Prozent über der Linie: Treppenstufen-Artefakte am runden Metallrand, dort wo die Plasmonen-Ladungen sitzen; mit Auflösung 300 rücken die Punkte auf die Linie (Konvergenz, Kapitel 10).

```
Auflösung 200: Abweichung von der Theorie im Mittel 8.1 %, maximal 13.7 %
Auflösung 300: Abweichung von der Theorie im Mittel 2.8 %, maximal 6.2 %
Peak bei λ = 524 nm – im Grünen
```

Der Peak liegt bei **~520 nm, im Grünen** — nicht beim Gelb des Goldglanzes. Eine Scheibe, die Grün heraussiebt, lässt von weißem Licht vor allem **Rot** übrig: Genau das ist das Geheimnis von **Goldrubinglas**, dem tiefroten Kirchenfenster- und Pokalglas, das Glasmacher seit Jahrhunderten herstellen, indem sie winzigste Goldtröpfchen im Glas ausfällen — Nano-Optik aus der Werkstatt, Jahrhunderte vor dem Wort „Nano". (Das spektakulärste Stück, der römische Lycurgus-Becher, wirkt im Auflicht grünlich und im Durchlicht rot.) Dass ein und dasselbe Metall als Spiegel gelb und als Nanoteilchen rot erscheint, liegt an der Resonanz: Der Spiegel reflektiert breitbandig (minus das von Interband-Übergängen geschluckte Blau), die Schei-

be aber *wählt* ihre Farbe über die Eigenfrequenz der Elektronenwolke — ein Resonator, dessen Resonanz man über Größe, Form und Umgebung verstimmen kann. Auf diesem Regler beruht ein ganzes Forschungsfeld, die **Plasmonik**: Biosensoren, die anschlagen, wenn Moleküle die Resonanz verschieben, Nanoantennen, die Licht auf Molekülgröße konzentrieren — und die LSPR-Schnelltests der Medizin (der rötliche Streifen mancher Teststreifen *ist* kolloidales Gold).

16.7 Das Kapitel-Programm

Das Programm `kap16_dispersion.py` (im Buchpaket unter `programme/kap16/`) fasst alle sechs Experimente zusammen: Schwebung, Gitterpaket, Lorentz-Wettrennen samt Auflösungs-Gegenprobe, Prisma, Drude-Messung und Goldscheiben-Spektrum. Jede Behauptung dieses Kapitels steht dort als `assert`; der Lauf braucht etwa zwei Minuten und endet mit „Alle Tests bestanden."

Merkkasten

- **Dispersion** = Wellentempo hängt von der Frequenz ab. Dann trennen sich $v_p = \omega/k$ (Berge) und $v_g = d\omega/dk$ (Paket, Energie, Information).
- **Glas**: gebundene Elektronen = getriebene Resonatoren (Lorentz). Unterhalb der Resonanz wächst n mit f — Blau ist das langsame Licht und wird im Prisma stärker geknickt (das Gitter knickt Rot stärker: Geometrie statt Material).
- **Metall/Plasma**: freie Elektronen (Drude), $\varepsilon = 1 - f_p^2/f^2$. Unter f_p wird gespiegelt (Metallglanz, Ionosphäre), über f_p ist $n < 1$ und $v_p > c$ — erlaubt, weil Berge keine Boten sind ($v_p v_g = c^2$, die Front bleibt unter c).
- **Numerische und physikalische Dispersion addieren sich** — die Kapitel-8-Formel beziffert den Gitteranteil, die Konvergenzleiter schrumpft ihn.
- **Plasmonen**: In Nanometall schwappt die Elektronenwolke als Resonator; eine 100-nm-Goldscheibe streut deshalb Grün am stärksten — darum ist Goldrubinglas rot.

Roter Faden

Rückwärts: Die **Schwebung** kam mit dem Additionstheorem zustande, das schon Stehwellen (Kapitel 6) und Doppelspalt (Kapitel 14) baute; „Energie sitzt in u^2" stammt aus **Kapitel 4**. Die **Gitterdispersion** aus **Kapitel 8** wurde hier zweimal eingelöst: als erstes messbares dispersives Medium — und als Störgröße, die man beziffern muss, bevor man Materialdispersion misst. Der **Hand-Detektor aus Kapitel 10** hat die Phasenmessung übernommen, **Kapitel 12** lieferte Snellius und den evaneszenten Saum (der im Drude-Medium unter f_p und im $\varepsilon < 0$-Fenster des Lorentz-Glases wiederkehrte), **Kapitel 13** die Einsicht, dass Polarisation entscheidet (die Plasmonen-Scheibe braucht E quer), und **Kapitel 14** das Spektrometer-Rezept „ein Puls, zwei DFT-Fernfelder" fürs Prisma — dessen umgekehrte Sortierung damit erklärt ist. Die Resonatoren aus **Kapitel 15** wurden zu Elektronen an Federn: Die Lorentz-Linie ist jetzt Materialantwort, das Plasmon ihr schwappendes Extrem. Vorwärts: **Kapitel 17** schickt den evaneszenten Saum durch einen Spalt (Tunneln) und findet die Plasma-Klippe als Hohlleiter-Cutoff wieder; **Kapitel 20** verfolgt das Front-Argument bis zur Kausalität; **Kapitel 26** baut aus periodischen Strukturen Dispersion nach Maß (Bandlücken = verbotene Fenster wie unser graues $\varepsilon < 0$-Band); und **Kapitel 29** zeigt das Tiefste an alledem: Kausalität *erzwingt* Dispersion — ein Medium, das alle Farben gleich behandelt und trotzdem absorbiert, kann es nicht geben.

Übungen

Ü 16.1 (Verstehen). Tiefwasserwellen gehorchen $\omega = \sqrt{gk}$. Zeige von Hand, dass $v_g = v_p/2$ gilt. Dann zum Sturm vom Kapitelanfang: Am Strand kommt zuerst Dünung mit 16 s Periode an; 24 Stunden später ist die Periode auf 12 s gefallen. Wie weit war der Sturm entfernt? (Tipp: Eine Welle der Frequenz f braucht für die Strecke D die Zeit $t = D/v_g(f)$ — leite daraus her, wie schnell die beobachtete Frequenz pro Tag wächst.)

> **Musterlösung zu Ü 16.1**
>
> $v_p = \omega/k = \sqrt{g/k}$ und $v_g = d\omega/dk = \frac{1}{2}\sqrt{g/k} = v_p/2$ — die Wurzel halbiert die Steigung. Mit $\omega = 2\pi f$ und $k = \omega^2/g$ wird $v_g = g/(4\pi f)$: *niedrige* Frequenzen reisen schneller, darum kommt die lange Dünung zuerst. Eine Welle der Frequenz f braucht $t = D/v_g = 4\pi f D/g$; nach t Stunden kommt also gerade die Frequenz $f(t) = g\,t/(4\pi D)$ an — sie wächst **linear** mit der Zeit, und die Steigung verrät die Distanz:

$D = g/(4\pi\, df/dt)$.

```
g = 9.81
df = 1/12 - 1/16                    # Hz Zuwachs ...
dt = 24 * 3600                      # ... in einem Tag
D = g / (4 * np.pi * df / dt)
print(f"Sturmentfernung D = {D/1000:.0f} km")
print(f"v_g bei T = 16 s: {g/(4*np.pi/16):.1f} m/s "
      f"-> Laufzeit {D/(g/(4*np.pi/16))/3600:.0f} h")
```

```
Sturmentfernung D = 3238 km
v_g bei T = 16 s: 12.5 m/s -> Laufzeit 72 h
```

Rund 3200 km — Ozeanografen nutzen genau dieses Verfahren seit den 1940ern, um Sturmgebiete ohne Satelliten zu orten. Die Dünung lief dafür etwa drei Tage.

Ü 16.2 (Verstehen). Berechne v_p und v_g auf dem FDTD-Gitter für 4 Punkte pro Wellenlänge bei $S = 0{,}5$ — und begründe ohne Rechnung, was bei $S = 1$ herauskäme.

Musterlösung zu Ü 16.2

$k\Delta x/2 = \pi/4$: $\sin(\omega\Delta t/2) = 0{,}5 \cdot 0{,}7071 = 0{,}3536$, also $\omega\Delta t/2 = 0{,}3614$. Damit $v_p = \omega/k = (2 \cdot 0{,}3614/0{,}5)/(\pi/2) = 0{,}9202\,c$ und $v_g = c\cos(\pi/4)/\cos(0{,}3614) = 0{,}7071/0{,}9354 = 0{,}7560\,c$ — bei 4 Punkten pro Wellenlänge schleicht das Paket schon mit Dreiviertel-Tempo.

```
S, kdx2 = 0.5, np.pi / 4
wdt2 = np.arcsin(S * np.sin(kdx2))
print(f"v_p = {(2*wdt2/S)/(np.pi/2):.4f} c,  "
      f"v_g = {np.cos(kdx2)/np.cos(wdt2):.4f} c")
```

```
v_p = 0.9202 c,  v_g = 0.7559 c
```

Bei $S = 1$ (der magische Zeitschritt, Kapitel 8) wird die Dispersionsrelation zur Geraden $\omega = ck$: keine Krümmung, $v_p = v_g = c$ für *alle* Wellenlängen — auch für 4 Punkte pro Welle. Dispersion braucht eine krumme $\omega(k)$-Kurve; der magische Schritt bügelt sie glatt.

Ü 16.3 (Verändern). Rücke den Träger im Lorentz-Glas näher an die Resonanz: $f = 1{,}5$ (statt 1,2). Berechne erst n, n_g und die erwartete v_g (Materi-

altheorie und kombinierte Theorie mit Gitterfaktor bei Auflösung 40), dann miss nach. Was fällt am Paket auf?

Musterlösung zu Ü 16.3

```
# von oben: tempi_messen(), gitterfaktor(), n_glas(), n_gruppe(), GLAS
f = 1.5
print(f"n = {n_glas(f):.4f}, n_g = {n_gruppe(f):.4f}")
print(f"v_g Material   = {1/n_gruppe(f):.4f} c")
print(f"v_g kombiniert = {gitterfaktor(f, 40)/n_gruppe(f):.4f} c")
vg, vp, _ = tempi_messen(GLAS, f, 280, 40)
print(f"v_g gemessen   = {vg:.4f} c    (v_p = {vp:.4f} c)")
```

```
n = 1.9640, n_g = 3.8344
v_g Material   = 0.2608 c
v_g kombiniert = 0.2543 c
v_g gemessen   = 0.2544 c    (v_p = 0.5051 c)
```

Nahe der Resonanz wird die $n(f)$-Kurve steil: $n_g = 3{,}83$ ist fast doppelt so groß wie $n = 1{,}96$ — das Paket kriecht mit gut einem Viertel von c, während die Berge noch halbes c laufen, und die Messung liegt wieder auf der kombinierten Theorie. Außerdem zerfließt das Paket sichtbar stärker als bei $f = 1{,}2$: Seine Bandbreite erwischt einen Bereich, in dem v_g selbst stark variiert (Gruppengeschwindigkeits-Dispersion, siehe Kleingedrucktes). Treibt man das Spiel weiter — sehr nah an die Resonanz, mit schwacher Dämpfung — entsteht „slow light": Im Labor wurden Lichtpulse so auf Fahrradtempo abgebremst.

Ü 16.4 (Verändern). Die Ionosphäre als Streifenexperiment: Lege eine Drude-Schicht der Dicke 2 ($f_p = 1$) quer in einen Vakuum-Streifen und schicke je einen schmalbandigen Puls bei $f = 0{,}7$ und $f = 1{,}25$ hindurch. Miss die Transmission (Fluss hinter der Schicht, normiert per Referenzlauf ohne Schicht — das Rezept aus Kapitel 12). Erkläre das Ergebnis mit Kurzwellenfunk und Satellitenfunk.

Musterlösung zu Ü 16.4

```
# von oben: mp, np, DRUDE
def schichtlauf(f, mit_schicht):
    quelle = [mp.Source(mp.GaussianSource(f, fwidth=f / 8),
                        component=mp.Ez, center=mp.Vector3(-18, 0),
                        size=mp.Vector3(0, 2))]
    geometrie = ([mp.Block(size=mp.Vector3(2, mp.inf, mp.inf),
                           material=DRUDE)] if mit_schicht else [])
    sim = mp.Simulation(cell_size=mp.Vector3(44, 2), resolution=40,
                        sources=quelle, geometry=geometrie,
                        boundary_layers=[mp.PML(2.0, direction=mp.X)],
                        k_point=mp.Vector3())
    dahinter = sim.add_flux(f, f / 8, 1, mp.FluxRegion(
        center=mp.Vector3(10, 0), size=mp.Vector3(0, 2),
        direction=mp.X))
    sim.run(until_after_sources=120)
    return mp.get_fluxes(dahinter)[0]

for f in (0.7, 1.25):
    T = schichtlauf(f, True) / schichtlauf(f, False)
    print(f"f = {f}: Transmission = {T:.2e}")
```

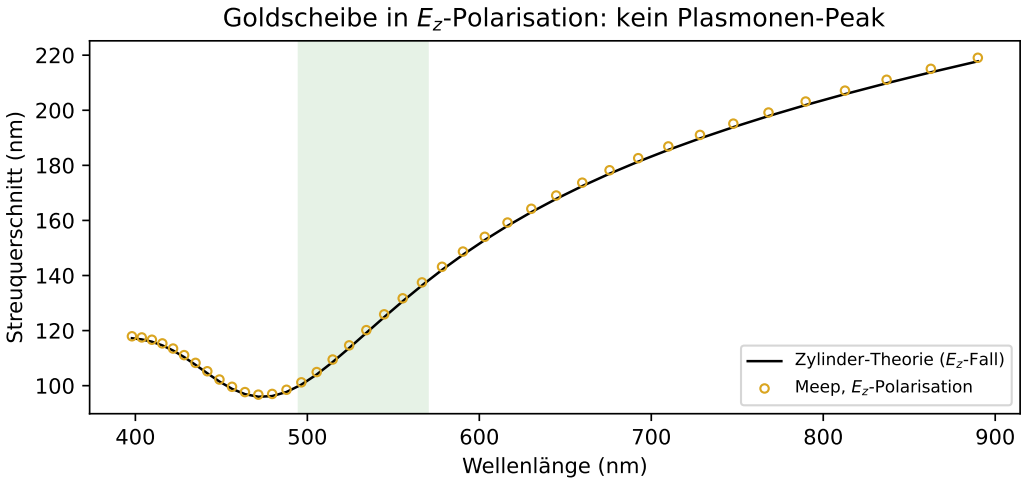
```
f = 0.7: Transmission = 5.16e-08
f = 1.25: Transmission = 9.96e-01
```

Unter der Plasmafrequenz kommt praktisch **nichts** durch (Transmission $\sim 10^{-7}$ — die Welle dringt nur evaneszent ein, zwei Einheiten Schichtdicke sind viele Abklinglängen, Kapitel 12), darüber fast **alles** ($\approx 0{,}996$; der kleine Rest sind Fresnel-Reflexe an den beiden Schichtgrenzen). Genau diese Klippe nutzt der Funk: Kurzwelle (unter dem f_p der Ionosphäre, einige MHz) wird am „Plasmaspiegel" zur Erde zurückgeworfen und springt so um den Globus; Satellitenfunk im GHz-Bereich liegt weit über f_p und durchquert die Schicht, als wäre sie nicht da. Nachts sinkt die Elektronendichte und mit ihr f_p — darum tragen Mittelwellensender nachts weiter.

Ü 16.5 (Übertragen). Miss das Streuspektrum der Goldscheibe noch einmal, aber mit E_z-Polarisation (E-Feld parallel zur Scheibenachse). Vergleiche mit der passenden Theorielinie (`mie_zylinder(..., polarisation="Ez")`). Warum verschwindet der grüne Peak — und warum wächst die Streuung jetzt zu langen Wellenlängen hin?

Musterlösung zu Ü 16.5

```
# von oben: streuquerschnitt(), mie_zylinder(), mp, np, plt
lam_ez, sigma_ez = streuquerschnitt(200, komponente=mp.Ez)
mie_ez = np.array([mie_zylinder(l, 0.05, polarisation="Ez")
                   for l in lam_ez])
fig, achse = plt.subplots(figsize=(7, 3.4))
achse.axvspan(495, 570, color="green", alpha=0.10)
achse.plot(lam_ez * 1000, mie_ez * 1000, "k-", lw=1.2,
           label="Zylinder-Theorie ($E_z$-Fall)")
achse.plot(lam_ez * 1000, sigma_ez * 1000, "o", ms=4, mfc="none",
           color="goldenrod", label="Meep, $E_z$-Polarisation")
achse.set(xlabel="Wellenlänge (nm)", ylabel="Streuquerschnitt (nm)",
          title="Goldscheibe in $E_z$-Polarisation: kein Plasmonen-Peak")
achse.legend(fontsize=8)
fig.tight_layout()
plt.show()
abw = np.abs(sigma_ez / mie_ez - 1)
print(f"Abweichung von der Theorie: im Mittel {np.mean(abw)*100:.1f} %")
```




```
Abweichung von der Theorie: im Mittel 0.7 %
```

Kein Peak im Grünen — die Kurve steigt einfach zu langen Wellen hin. In E_z-Polarisation schiebt das Feld die Elektronen **längs** der (in 2D unendlich langen) Scheibenachse: Da gibt es keine Ränder, keine Oberflächenladungen, keine Rückstellkraft — also keinen Resonator und kein Plasmon (Kapitel 13 lässt grüßen: Polarisation entscheidet, welche Physik überhaupt stattfindet). Was bleibt, ist die Scheibe als dünner leitender **Draht**, in dem das Feld frei Strom treibt — ein winziger Spiegel, der

lange Wellen immer effektiver zurückwirft. Bemerkenswert am Rande: Hier liegen schon die 200er-Messpunkte auf etwa ein Prozent an der Theorie — die Treppenstufen am Rand stören kaum noch, weil die empfindliche Physik (die Randladungen des Plasmons) in dieser Polarisation gar nicht existiert.

Das Kleingedruckte

Fünf Randnotizen. **Erstens:** Die Lorentz-Formel folgt aus Newton für ein Federelektron, $m\ddot{x} = -m\omega_0^2 x + qE$: Für $E \propto e^{-i\omega t}$ ist die Auslenkung $x = qE/[m(\omega_0^2 - \omega^2)]$, die Polarisation $P = Nqx$ liefert $\varepsilon = 1 + Nq^2/[\varepsilon_0 m(\omega_0^2 - \omega^2)]$ — unsere Formel mit $\sigma f_0^2 = Nq^2/(4\pi^2\varepsilon_0 m)$. Echte Gläser haben mehrere Resonanzen; die Praxisformel der Optikindustrie (Sellmeier) ist schlicht eine Summe solcher Terme. **Zweitens:** Zwischen Unter- und Oberseite einer Resonanz fällt n mit f („anomale Dispersion") — dort wird auch kräftig absorbiert, und die Gruppengeschwindigkeit verliert ihre einfache Bedeutung als Energietempo; die saubere Schranke, die immer gilt, ist die **Frontgeschwindigkeit** $\leq c$ (Kapitel 20/29). **Drittens:** Hängt v_g selbst von f ab (das tut es fast immer), zerfließen Pulse unterwegs — **Gruppengeschwindigkeits-Dispersion**. Für Glasfaser-Strecken ist sie der begrenzende Gegner: Ein Bit, das zerfließt, frisst seinen Nachbarn. Die Gegenmittel — Betrieb nahe 1310 nm, wo die GVD der Quarzfaser durch null geht, oder Kompensationsfasern mit gegenläufiger Dispersion — sind angewandte $n_g(f)$-Kurven. **Viertens**, der versprochene Werkstatt-Fallstrick: Eine `GaussianBeamSource` prägt der Quelllinie ein festes räumliches Phasenmuster auf. Kippt man den Strahl, gehört zu diesem Muster für jede Frequenz eine *andere* Richtung ($\sin\theta \propto 1/f$) — ein breitbandiger Puls fächert dann nach Farben auf, bevor er das Prisma überhaupt erreicht. Deshalb blieb unser Strahl horizontal (Phasenmuster flach, Richtung frequenzunabhängig) und das Prisma wurde gedreht. **Fünftens:** Die Plasmonen-Resonanz der Scheibe kann man quasistatisch abschätzen (Resonanz des kleinen Zylinders bei $\mathrm{Re}\,\varepsilon = -1$, der Kugel bei -2) — für Gold landet das bei ~430 nm statt der gemessenen ~520 nm: Bei 100 nm Durchmesser ist die Scheibe nicht mehr „klein gegen λ" (Verzögerungs-Rotverschiebung), und die Interband-Elektronen dämpfen das blaue Ende. Die ausgetippte Zylinder-Reihe kennt all das; die Faustformel liefert nur noch die Richtung „blau-grün". Solche ehrlichen Abweichungen zwischen Faustformel und exakter Theorie sind in der Plasmonik Alltag.

17 Tunneln und Cutoff

Lege einen Finger auf ein Glas Wasser und schau von schräg unten durch die Wand: Wo die Haut das Glas berührt, siehst du die Fingerlinien dunkel und gestochen scharf — obwohl an dieser Stelle eigentlich **Totalreflexion** herrschen müsste, die das Licht vollständig im Glas hält. Genau dieser Effekt steckt in vielen Fingerabdruck-Scannern: Eine Glasplatte führt Licht per Totalreflexion, und überall dort, wo eine Hautleiste *nahe genug* an die Oberfläche kommt, springt das Licht aus dem Glas in den Finger und fehlt anschließend im reflektierten Bild. Nahe genug — nicht: berührt. Das Licht überwindet einen Luftspalt, durch den es laut Kapitel 12 gar nicht hindurchkönnen dürfte.

Dieses Kapitel löst gleich drei alte Versprechen ein. Kapitel 12 hatte den **evaneszenten Saum** vermessen — das exponentiell abklingende Feld jenseits der totalreflektierenden Grenze — und angekündigt, dass er hier zur Hauptfigur wird: Hält man ihm ein zweites Glas hin, *tunnelt* das Licht. Kapitel 15 fand, dass ein kleines Loch im Resonator praktisch dicht ist (die Mikrowellentür), und vertagte das Warum auf dieses Kapitel: den **Cutoff** — unterhalb einer Grenzfrequenz trägt ein Rohr schlicht keine Welle. Und Kapitel 16 maß im Drude-Medium Wellenberge, die schneller als c laufen; dieselbe Physik wohnt, wie sich zeigen wird, in jedem Stück Metallrohr. Tunneln und Cutoff sind dabei kein Themenpaar aus Verlegenheit: Es ist **dieselbe Mathematik** — ein Frequenzbereich, in dem die Welle nicht laufen kann, ein exponentielles $e^{-\kappa x}$, und die Frage, wie viel davon am anderen Ende noch übrig ist.

Lernziele

Nach diesem Kapitel kannst du …

1. … die frustrierte Totalreflexion (FTIR) erklären und quantitativ vorhersagen: Transmission durch einen Luftspalt nach der sinh-Formel, über drei Dekaden gemessen,
2. … begründen, warum ein Strahl ein Winkelbündel ist und wann das die Messung dominiert (die Kapitel-12-Lektion, quantitativ gemacht),

3. ... die Grenzfrequenz eines Hohlleiters $f_c = m/(2w)$ herleiten und die Cutoff-Klippe messen — samt der Mikrowellentür-Bilanz aus Kapitel 15,
4. ... die Hohlleiter-Dispersion $\omega^2 = \omega_c^2 + c^2k^2$ herleiten, ihre Verwandtschaft mit dem Drude-Plasma aus Kapitel 16 erkennen und $v_p \cdot v_g = c^2$ im Rohr messen,
5. ... erklären, warum die Glasfaser ihr Licht behält — und unter welchen Umständen doch nicht.

17.1 Eine Brücke für den Saum

Zuerst die Erinnerung mit Zahlen. Trifft Licht aus Glas ($n = 1{,}5$) unter $\theta > \theta_c = 41{,}8°$ auf die Grenze zu Luft, wird es total reflektiert; jenseits der Grenze lebt nur der Saum $e^{-\kappa y}$ mit (Kapitel 12)

$$\kappa = \frac{2\pi f}{c}\sqrt{n^2 \sin^2\theta - 1}.$$

Die Handrechnung für unsere Bühne — $f = 1$, $\theta = 50°$: $n \sin\theta = 1{,}5 \cdot 0{,}766 = 1{,}149$, also $\kappa = 2\pi\sqrt{1{,}320 - 1} = 2\pi \cdot 0{,}566 = 3{,}555$ und Eindringtiefe $\delta = 1/\kappa = 0{,}281$ — gut eine Viertelwellenlänge. In dieser dünnen Schicht ist Feld, und Kapitel 12 hat gemessen, dass es dort sogar Energie *seitwärts* transportiert. Es fehlt ihm nur ein Abnehmer.

Den liefern wir jetzt: ein **zweites Glas**, im Abstand d parallel über dem ersten. Liegt d in Reichweite des Saums, findet das evaneszente Feld an der zweiten Grenze wieder ein Medium, in dem es als *laufende* Welle weiter kann — und ein Teil der Leistung tut genau das. Der Effekt heißt **frustrierte Totalreflexion** (FTIR): Die Totalreflexion wird um ihren hundertprozentigen Anspruch gebracht. Die Theorie dazu ist ein Randbedingungs-Puzzle wie die Spiegelformel aus Kapitel 6, nur mit zwei Grenzen und dem Saum dazwischen; das Ergebnis für unsere Polarisation (E_z, also s) lautet

$$T(d) = \frac{1}{1 + V\,\sinh^2(\kappa d)}, \qquad V = \frac{(k_z^2 + \kappa^2)^2}{4\,k_z^2\,\kappa^2},$$

mit $k_z = (2\pi f/c)\, n \cos\theta$, der senkrechten Wellenzahl im Glas. Zwei Proben aufs Exempel: Für $d \to 0$ ist $\sinh 0 = 0$, also $T = 1$ — kein Spalt, keine Reflexion, das Glas ist durchgehend. Und für $\kappa d \gtrsim 1$ wächst $\sinh^2(\kappa d) \approx e^{2\kappa d}/4$, also fällt $T \propto e^{-2\kappa d}$ — *exponentiell*, mit dem doppelten κ, weil Leistung das Quadrat der Amplitude ist. Die Handrechnung an einem Punkt: $k_z = 2\pi \cdot 1{,}5 \cdot \cos 50° = 6{,}058$, $V = (36{,}7 + 12{,}6)^2/(4 \cdot 36{,}7 \cdot 12{,}6) = 1{,}31$; bei

$d = 0{,}4$ ist $\sinh(1{,}422) = 1{,}95$ und damit $T = 1/(1 + 1{,}31 \cdot 3{,}81) = 0{,}167$ — ein Sechstel der Leistung springt über einen Spalt von $0{,}4$ Wellenlängen.

> ⚠ Naheliegende Vermutung
>
> **„Total heißt total: Solange das zweite Glas die Oberfläche nicht berührt, bleibt die Reflexion bei 100 % — Luftspalt ist Luftspalt."**
> *Warum sie naheliegt:* Kapitel 12 hat die Totalreflexion als perfekten Spiegel gemessen (Reflexionsgrad 0,989 plus Bündel-Krümel) — und ein Spalt, durch den keine laufende Welle passt, sieht nach einer absoluten Barriere aus.
> *Was stattdessen stimmt:* Der Saum *ist* Feld jenseits der Grenze — nur eben abklingendes. Kommt die zweite Grenze in seine Reichweite (ein paar Zehntel Wellenlängen), trägt er Leistung hinüber: bei $d = 0{,}4\,\lambda$ schon ein Sechstel. „Total" gilt nur, wenn der Saum *nichts findet* — die Messung folgt sofort, und die Empfindlichkeit der Sache (exponentiell in d!) ist kein Ärgernis, sondern die Grundlage von Fingerabdruck-Scannern und einstellbaren Strahlteilern.

Das Experiment: die Kapitel-12-Bühne mit zweitem Stockwerk. **Frage:** Folgt die gemessene Transmission $T(d)$ der sinh-Formel? **Bühne:** eine 24×16-Zelle, Auflösung 24; Glas füllt alles unterhalb von $y = 0$ und alles oberhalb von $y = d$ — dazwischen der Luftspalt. **Anregung:** der Gauß-Strahl aus Kapitel 12 (`GaussianBeamSource`, Taille $w_0 = 3$, schmalbandiger Puls um $f = 1$), aus dem unteren Glas auf den Punkt $(0, 0)$ gezielt, Einfallswinkel $50°$ — acht Grad über dem Grenzwinkel, damit auch die Flügel des Winkelbündels (±4°, dazu gleich mehr) totalreflektiert werden. **Gemessen wird** der Fluss durch eine Linie oben im zweiten Glas (`add_flux`, `until_after_sources` — das Rezept aus Kapitel 12); als Referenz dient der Lauf mit $d = 0$, also durchgehendes Glas: $T(d) = P(d)/P(0)$. **Erfolgskriterium:** die Punkte auf der Theorielinie über mehrere Dekaden.

Zuerst ein einzelner Lauf mit $d = 0{,}6$, von dem wir uns auch die Lichtspur (das aufsummierte E_z^2, das Rezept aus Kapitel 16) zeigen lassen:

```
import os
os.environ["HWLOC_HIDE_ERRORS"] = "2"   # vor dem Meep-Import (Kapitel 11)
import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)                          # Fortschrittsmeldungen stumm

N_GLAS = 1.5
```

```
F0 = 1.0
W0 = 3.0
AUFL = 24
GLAS = mp.Medium(index=N_GLAS)
ZELLE = mp.Vector3(24, 16)

def ftir_lauf(theta_grad, d, mit_spur=False):
    """Ein FTIR-Lauf; liefert Fluss oberhalb (und ggf. die Lichtspur)."""
    geometrie = [mp.Block(size=mp.Vector3(mp.inf, 8, mp.inf),
                          center=mp.Vector3(0, -4), material=GLAS),
                 mp.Block(size=mp.Vector3(mp.inf, 8 - d, mp.inf),
                          center=mp.Vector3(0, d + (8 - d) / 2),
                          material=GLAS)]
    th = np.deg2rad(theta_grad)
    quelle = [mp.GaussianBeamSource(
        mp.GaussianSource(F0, fwidth=F0 / 8),
        center=mp.Vector3(-4.0 * np.tan(th), -4.0),   # zielt auf (0, 0)
        size=mp.Vector3(10, 0), beam_x0=mp.Vector3(0, 0),
        beam_kdir=mp.Vector3(np.sin(th), np.cos(th)), beam_w0=W0,
        beam_E0=mp.Vector3(0, 0, 1), component=mp.ALL_COMPONENTS)]
    sim = mp.Simulation(cell_size=ZELLE, resolution=AUFL,
                        sources=quelle, geometry=geometrie,
                        boundary_layers=[mp.PML(1.5)])
    oben = sim.add_flux(F0, F0 / 8, 1, mp.FluxRegion(
        center=mp.Vector3(3.0, 4.0), size=mp.Vector3(16, 0),
        direction=mp.Y))
    spur = {}
    def aufsummieren(sim):
        ez2 = sim.get_array(component=mp.Ez).T ** 2
        spur["I"] = spur.get("I", 0) + ez2
    schritte = [mp.at_every(0.5, aufsummieren)] if mit_spur else []
    sim.run(*schritte, until_after_sources=40)
    if mit_spur:
        return mp.get_fluxes(oben)[0], spur["I"]
    return mp.get_fluxes(oben)[0]

P_d06, spur = ftir_lauf(50.0, 0.6, mit_spur=True)

fig, achse = plt.subplots(figsize=(7, 4.4))
achse.imshow(np.sqrt(spur), origin="lower", cmap="inferno",
             extent=[-12, 12, -8, 8], vmax=0.5 * np.sqrt(spur).max())
for y in (0.0, 0.6):
    achse.axhline(y, color="w", lw=0.8)
```

```
achse.annotate("Glas", (-11.5, -7), color="w")
achse.annotate("Glas", (-11.5, 6.6), color="w")
achse.annotate("Luftspalt d = 0,6", (-11.5, 1.1), color="w", fontsize=8)
achse.set(xlim=(-12, 12), ylim=(-8, 8), xlabel="x", ylabel="y",
          title="Das Licht springt über den Spalt")
fig.tight_layout()
plt.show()
```

```
Using MPI version 4.1, 1 processes
```

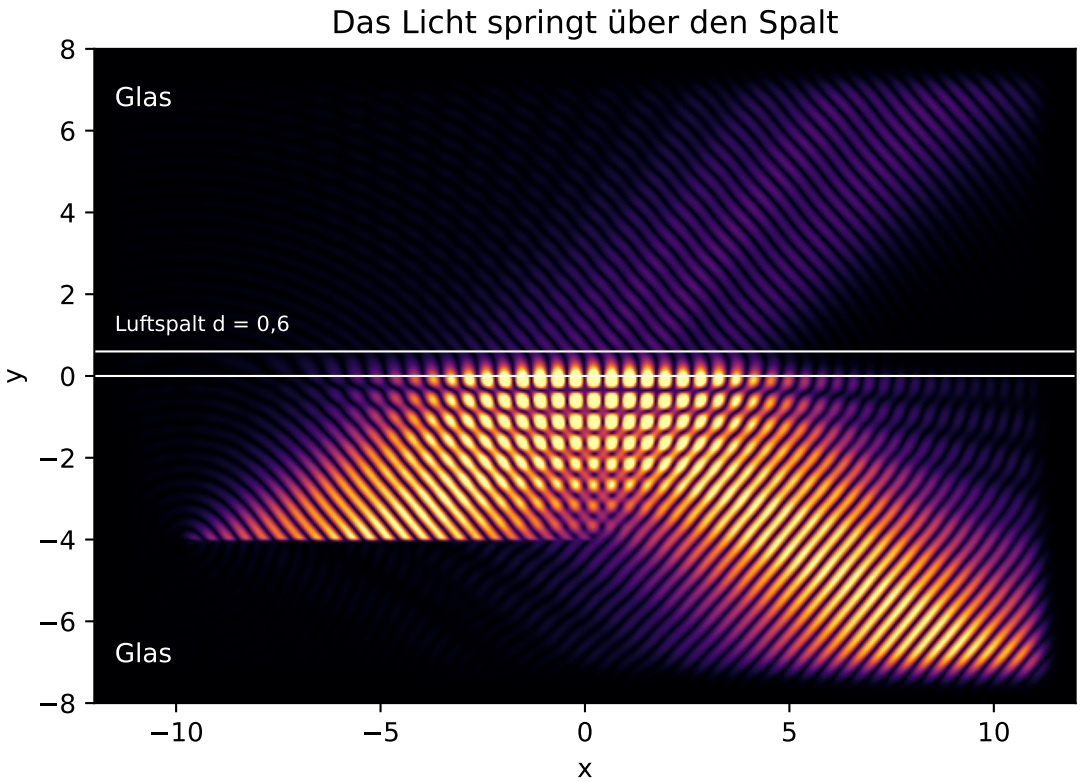


Abbildung 17.1: Lichtspur eines FTIR-Laufs mit Spalt d = 0,6 (aufsummiertes E_z², Wurzeldarstellung; die weißen Linien markieren die beiden Glasoberflächen). Der Strahl kommt von links unten, der helle reflektierte Arm läuft nach rechts unten ab — aber über den Luftspalt hinweg setzt ein zweiter, schwächerer Strahl im oberen Glas an und läuft parallel weiter: das getunnelte Licht. Im Spalt selbst liegt nur der schmale helle Saum.

Da ist die Brücke.

Bevor die Messreihe läuft, der Vorhersage-Moment:

> **Vorhersage-Punkt**
>
> Wir messen gleich T für Spaltbreiten von $0{,}2$ bis $1{,}2$. Wenn man den Spalt von $d = 0{,}4$ auf $d = 0{,}8$ **verdoppelt** — fällt die Transmission dann ungefähr auf die Hälfte, auf ein Viertel, oder auf etwas ganz anderes?

Die Formel hat die Antwort schon verraten: Im Exponentialbereich kostet *jede zusätzliche Strecke* δ denselben *Faktor* — $T \propto e^{-2\kappa d}$ ist kein Verhältnis-, sondern ein Faktorengesetz. Verdopplung von $0{,}4$ auf $0{,}8$ heißt $e^{-2\cdot 3{,}555\cdot 0{,}4} \approx 1/17$ zusätzlich (die sinh-Korrektur mildert das auf etwa $1/11$). Jetzt die Reihe:

```python
# von oben: ftir_lauf(), np
def theorie_T(theta_grad, d):
    """FTIR-Transmission, ebene Welle, s-Polarisation."""
    th = np.deg2rad(theta_grad)
    k = 2 * np.pi * F0
    arg = (N_GLAS * np.sin(th))**2 - 1
    if arg <= 0:
        return 1.0          # unterhalb des Grenzwinkels: durchlässig
    kz = k * N_GLAS * np.cos(th)
    kap = k * np.sqrt(arg)
    V = (kz**2 + kap**2)**2 / (4 * kz**2 * kap**2)
    return 1 / (1 + V * np.sinh(kap * d)**2)

P_ref = ftir_lauf(50.0, 0.0)             # Referenz: durchgehendes Glas
T_mess = {0.6: P_d06 / P_ref}            # der Lauf von eben
for d in (0.2, 0.4, 0.8, 1.0, 1.2):
    T_mess[d] = ftir_lauf(50.0, d) / P_ref
for d in sorted(T_mess):
    print(f"d = {d}:  T = {T_mess[d]:.4f}   "
          f"(ebene Welle: {theorie_T(50.0, d):.4f})")
print(f"\nSpaltverdopplung 0,4 -> 0,8: Faktor "
      f"{T_mess[0.4] / T_mess[0.8]:.1f}")
```

```
d = 0.2:  T = 0.5682   (ebene Welle: 0.5608)
d = 0.4:  T = 0.1781   (ebene Welle: 0.1666)
d = 0.6:  T = 0.0516   (ebene Welle: 0.0421)
d = 0.8:  T = 0.0167   (ebene Welle: 0.0103)
d = 1.0:  T = 0.0070   (ebene Welle: 0.0025)
```

```
d = 1.2:  T = 0.0040   (ebene Welle: 0.0006)

Spaltverdopplung 0,4 -> 0,8: Faktor 10.7
```

Die Verdopplungsantwort: Faktor elf, nicht zwei — exponentielle Gesetze bestrafen lineare Intuition. Aber die Tabelle enthält noch eine zweite Geschichte, und die kennen wir aus Kapitel 12: Bei kleinen Spalten passt die Ebene-Welle-Formel auf wenige Prozent, doch je größer d, desto weiter liegt die Messung **über** der Theorie — bei $d = 1{,}2$ um mehr als Faktor sechs. Wieder ist der Strahl schuld, genauer: dass er keiner ist. Ein Gauß-Strahl der Taille $w_0 = 3$ ist ein **Winkelbündel** mit etwa ±4° Streuung um die Strahlrichtung, und κ hängt empfindlich vom Winkel ab: Die flacheren Komponenten des Bündels (näher an θ_c) haben ein kleineres κ und tunneln *viel* besser. Bei großem d, wo jede Dekade κd zählt, dominieren diese Flügel die Transmission — das Bündel-Mittel liegt weit über dem Wert des Mittelwinkels. Das können wir ausrechnen, denn das Winkelspektrum des Gauß-Strahls ist selbst eine Gauß-Glocke (Breite $\propto 1/w_0$ — schmaler Strahl, breites Bündel: die Kapitel-10-Reziprozität im Raum):

```python
# von oben: theorie_T(), T_mess, N_GLAS, F0, W0, np, plt
def theorie_T_buendel(theta_grad, d):
    """sinh-Formel, gemittelt über das Gauß-Winkelbündel des Strahls."""
    k_med = 2 * np.pi * F0 * N_GLAS
    kx0 = k_med * np.sin(np.deg2rad(theta_grad))
    thetas = np.deg2rad(np.linspace(theta_grad - 12,
                                    theta_grad + 12, 241))
    kx = k_med * np.sin(thetas)
    gewicht = np.exp(-((kx - kx0)**2) * (W0**2) / 2)   # Winkelspektrum
    T = np.array([theorie_T(np.rad2deg(t), d) for t in thetas])
    return np.sum(gewicht * T) / np.sum(gewicht)

fig, achse = plt.subplots(figsize=(7, 4.2))
d_fein = np.linspace(0.05, 1.3, 120)
achse.semilogy(d_fein, [theorie_T(50.0, d) for d in d_fein], "k--",
               lw=1, label="ebene Welle: sinh-Formel")
achse.semilogy(d_fein, [theorie_T_buendel(50.0, d) for d in d_fein],
               "k-", lw=1.2, label="über das Winkelbündel gemittelt")
achse.semilogy(sorted(T_mess), [T_mess[d] for d in sorted(T_mess)],
               "o", color="C0", label="Meep")
achse.set(xlabel="Spaltbreite d (Wellenlängen)",
          ylabel="Transmission T",
          title="Tunneln über den Luftspalt: T(d)")
achse.legend(fontsize=9)
```

```
fig.tight_layout()
plt.show()
for d in (0.8, 1.0, 1.2):
    print(f"d = {d}: Messung/Bündel = "
          f"{T_mess[d] / theorie_T_buendel(50.0, d):.3f},  "
          f"Messung/ebene Welle = {T_mess[d] / theorie_T(50.0, d):.2f}")
```

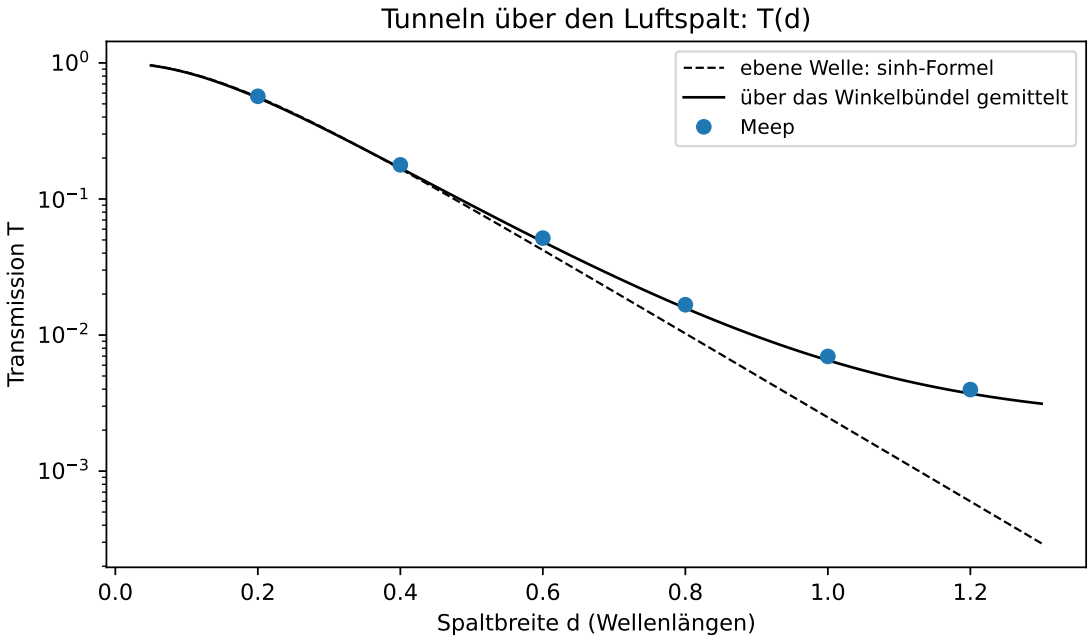


Abbildung 17.2: FTIR-Transmission über der Spaltbreite (halblogarithmisch, drei Dekaden): Meep-Messpunkte, dazu die sinh-Formel für die ebene Welle (gestrichelt) und ihr Mittel über das Gauß-Winkelbündel des Strahls (durchgezogen). Bei kleinen Spalten fallen beide Linien zusammen; ab d ≈ 0,8 tunneln praktisch nur noch die flachen Flügel des Bündels, und die Messung folgt der Bündel-Linie — bei d = 1,2 liegt sie Faktor 6 über der ebenen Welle.

```
d = 0.8: Messung/Bündel = 1.059,  Messung/ebene Welle = 1.63
d = 1.0: Messung/Bündel = 1.067,  Messung/ebene Welle = 2.80
d = 1.2: Messung/Bündel = 1.070,  Messung/ebene Welle = 6.63
```

Alle Punkte liegen auf wenige Prozent an der Bündel-Linie — über drei Dekaden. Die Lektion in einem Satz: **Exponentielle Effekte verstärken kleine Unterschiede im Exponenten zu großen Faktoren im Ergebnis** — wer

mit Strahlen statt ebenen Wellen misst, muss das Bündel mitrechnen, sonst „widerlegt" die eigene Messung eine korrekte Formel.

Was FTIR praktisch kann, folgt direkt aus der Kurve. Der **Fingerabdruck-Scanner** vom Kapitelanfang liest die Exponentialempfindlichkeit ab: Hautleisten (Abstand ~0) koppeln das Licht fast vollständig aus, Hauttäler (einige Mikrometer Luft, also mehrere δ) fast gar nicht — der Kontrast ist enorm, ganz ohne Linsenoptik an der Grenzfläche. Ein **einstellbarer Strahlteiler** ist ein Prismenpaar mit Stellschraube: Der Spaltabstand regelt stufenlos, welcher Anteil geradeaus tunnelt und welcher reflektiert wird — vom Spiegel ($d \gg \delta$) bis zur Fensterscheibe ($d \to 0$). Und wer die sinh-Formel schon einmal gesehen hat, hatte vermutlich ein Quantenmechanik-Buch in der Hand: Die Transmission eines Teilchens durch eine Potentialbarriere hat **exakt dieselbe Gestalt** — $\left[1 + V \sinh^2(\kappa d)\right]^{-1}$, nur dass κ dort aus der Schrödinger- statt der Wellengleichung kommt. FTIR ist das klassische Wellenexperiment zum Quantentunneln; was die Elektrodynamik hier mit Feldern vorführt, macht die Quantenmechanik mit Aufenthaltswahrscheinlichkeiten (Kapitel 30 nimmt diesen Faden bei $|E|^2$ wieder auf).

17.2 Das Rohr und die Klippe

Jetzt zur Mikrowellentür. Kapitel 15 hat gemessen, dass ein Loch der Breite $0{,}4$ (ein Zwölftel der Wellenlänge) den Resonator praktisch dicht lässt, und als Grund den „Wellenleiter unter Cutoff" genannt. Zeit, das Rohr selbst zu bauen. Ein **Hohlleiter** ist in unserer 2D-Welt ein Kanal zwischen zwei Metallwänden im Abstand w: quer zur Laufrichtung muss das Feld an beiden Wänden null sein — eine stehende Welle mit m Bäuchen, exakt wie im Kapitel-15-Kasten, nur dass die *zweite* Richtung offen bleibt und die Welle dort laufen darf. In der $f(m,n)$-Formel aus Kapitel 15 heißt das: Die offene Richtung trägt keinen festen Index mehr, ihr k darf beliebig klein werden — übrig bleibt als Minimum die reine Querschwingung,

$$f_c \;=\; \frac{m\,c}{2w},$$

die **Grenzfrequenz** (englisch *cutoff*). Unterhalb von f_c passt nicht einmal mehr die Querschwingung in den Kanal: Es gibt kein reelles k mehr, das Feld kann nur noch evaneszent abklingen — mit

$$\kappa \;=\; \frac{2\pi}{c}\sqrt{f_c^2 - f^2},$$

dem exakten Gegenstück zur FTIR-Formel (dort machte der zu große Querimpuls k_x die Welle evaneszent, hier die zu enge Geometrie). Die Handrechnung für unser Experiment: breites Rohr $w = 4$, also $f_c = 0{,}125$; Engstelle $w = 2$, also $f_c = 0{,}25$. Eine Welle mit $f = 0{,}2$ läuft im breiten Rohr bequem ($0{,}2 > 0{,}125$), ist aber in der Engstelle unter Cutoff: $\kappa = 2\pi\sqrt{0{,}0625 - 0{,}04} = 2\pi \cdot 0{,}15 = 0{,}9425$. Ein Stück Engstelle der Länge L sollte also $T \approx e^{-2\kappa L}$ durchlassen — das ist die Mikrowellentür als Labormodell: Das Loch in der Tür ist ein *kurzes Stück Rohr unter Cutoff*.

Das Experiment nutzt einen Trick aus Kapitel 15: **Meeps Zellrand ist von sich aus eine Metallwand.** Wir bauen also gar keine Rohrwände — die Zelle 36×4 *ist* das breite Rohr (PML nur links und rechts, in Laufrichtung); nur die Engstelle besteht aus zwei Metallblöcken, die den Kanal in der Mitte auf $w = 2$ verengen. **Anregung:** eine Quelllinie quer über das Rohr (gleichmäßiges E_z — das überlappt mit der $m = 1$-Mode, nicht mit $m = 2$), schmalbandiger Puls um $f = 0{,}2$. **Gemessen wird** der Fluss hinter der Engstelle, normiert auf den Referenzlauf ohne Engstelle (das FTIR-Rezept von eben). **Erfolgskriterium:** $\ln T$ fällt linear in L mit Steigung -2κ. Auch hier zuerst die Lichtspur eines Laufs:

```
# von oben: mp, np, plt
W_BREIT, W_ENG = 4.0, 2.0
F_ROHR = 0.2
KAPPA_ROHR = 2 * np.pi * np.sqrt(0.25**2 - F_ROHR**2)

def engstelle(L_eng):
    if L_eng == 0:
        return []
    return [mp.Block(size=mp.Vector3(L_eng, (W_BREIT - W_ENG) / 2,
                                     mp.inf),
                     center=mp.Vector3(0, s * (W_ENG / 2
                                       + (W_BREIT - W_ENG) / 4)),
                     material=mp.metal) for s in (+1, -1)]

def rohr_fluss(fcen, df, nfreq, L_eng, dauer=300, mit_spur=False):
    """Zellrand = Rohrwand (Kapitel-15-Trick); PML nur in x."""
    quelle = [mp.Source(mp.GaussianSource(fcen, fwidth=df),
                        component=mp.Ez, center=mp.Vector3(-13, 0),
                        size=mp.Vector3(0, W_BREIT))]
    sim = mp.Simulation(cell_size=mp.Vector3(36, W_BREIT),
                        resolution=20, sources=quelle,
                        geometry=engstelle(L_eng),
                        boundary_layers=[mp.PML(2.0, direction=mp.X)])
    hinten = sim.add_flux(fcen, df, nfreq, mp.FluxRegion(
```

```
        center=mp.Vector3(10, 0), size=mp.Vector3(0, W_BREIT),
        direction=mp.X))
    spur = {}
    def aufsummieren(sim):
        ez2 = sim.get_array(component=mp.Ez).T ** 2
        spur["I"] = spur.get("I", 0) + ez2
    schritte = [mp.at_every(0.5, aufsummieren)] if mit_spur else []
    sim.run(*schritte, until_after_sources=dauer)
    fluesse = np.array(mp.get_fluxes(hinten))
    freqs = np.array(mp.get_flux_freqs(hinten))
    return (fluesse, freqs, spur["I"]) if mit_spur else (fluesse, freqs)

P_rohr_ref, _ = rohr_fluss(F_ROHR, F_ROHR / 8, 1, 0.0)
T_rohr = {}
for L in (0.5, 1.0, 3.0):
    P, _ = rohr_fluss(F_ROHR, F_ROHR / 8, 1, L)
    T_rohr[L] = P[0] / P_rohr_ref[0]
P_L2, _, spur_rohr = rohr_fluss(F_ROHR, F_ROHR / 8, 1, 2.0,
                                mit_spur=True)
T_rohr[2.0] = P_L2[0] / P_rohr_ref[0]

Ls = np.array(sorted(T_rohr))
steigung = np.polyfit(Ls[1:], np.log([T_rohr[L] for L in Ls[1:]]), 1)[0]

fig, (links, rechts) = plt.subplots(
    1, 2, figsize=(9.5, 3.8), gridspec_kw={"width_ratios": [1.5, 1]})
links.imshow(np.sqrt(spur_rohr), origin="lower", cmap="inferno",
             extent=[-18, 18, -2, 2], aspect="auto",
             vmax=0.6 * np.sqrt(spur_rohr).max())
for s in (+1, -1):
    y0, y1 = s * 1.0, s * 2.0
    links.plot([-1, 1, 1, -1, -1], [y0, y0, y1, y1, y0], "w-", lw=0.8)
links.set(xlim=(-14, 14), xlabel="x", ylabel="y",
          title="Engstelle unter Cutoff (L = 2)")

L_fein = np.linspace(0.3, 3.2, 50)
C_fit = np.exp(np.mean([np.log(T_rohr[L]) + 2 * KAPPA_ROHR * L
                        for L in Ls[1:]]))
rechts.semilogy(L_fein, C_fit * np.exp(-2 * KAPPA_ROHR * L_fein), "k-",
                lw=1, label="C·e$^{-2\\kappa L}$ (κ aus Theorie)")
rechts.semilogy(list(T_rohr), list(T_rohr.values()), "s", color="C1",
                label="Meep")
rechts.set(xlabel="Länge L der Engstelle", ylabel="Transmission T",
```

```
            title="T(L): exponentiell dicht")
rechts.legend(fontsize=8)
fig.tight_layout()
plt.show()
print(f"Fit-Steigung (L >= 1): {steigung:.4f}   "
      f"(Theorie -2κ = {-2 * KAPPA_ROHR:.4f})")
for L in sorted(T_rohr):
    print(f"L = {L}: T = {T_rohr[L]:.4g}")
```

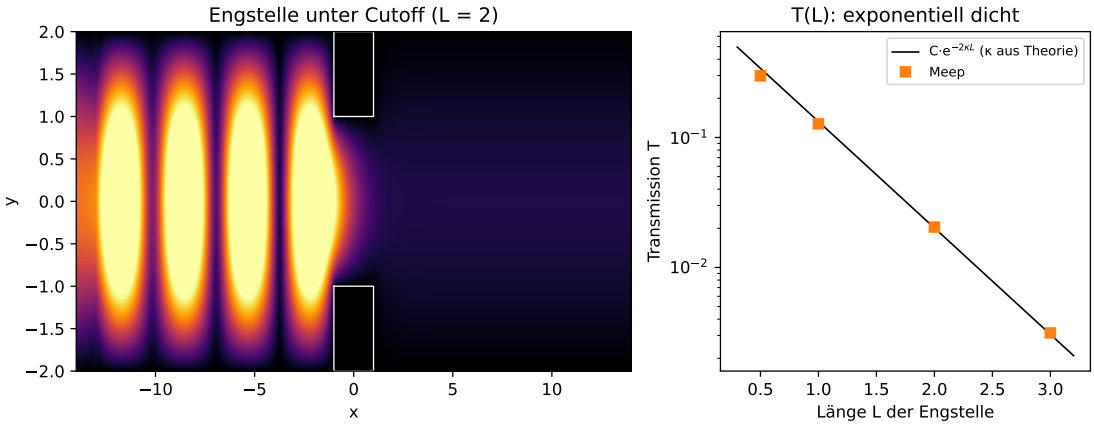


Abbildung 17.3: Links: Lichtspur im Rohr mit Engstelle (L = 2, f = 0,2; Wurzeldarstellung, Engstellen-Blöcke weiß umrandet). Im breiten Rohr steht die m = 1-Mode (ein Bauch in der Mitte, null an den Wänden); in der Engstelle bricht die Helligkeit sichtbar zusammen — das Feld klingt dort exponentiell ab statt zu laufen — und rechts tritt nur ein schwacher Rest aus. Rechts: T(L) halblogarithmisch; die Messpunkte fallen auf einer Geraden mit der Theorie-Steigung −2κ (Linie: C·e^(−2κL) mit angepasstem C).

```
Fit-Steigung (L >= 1): -1.8519   (Theorie -2κ = -1.8850)
L = 0.5: T = 0.2979
L = 1.0: T = 0.1273
L = 2.0: T = 0.02044
L = 3.0: T = 0.003136
```

Die Steigung trifft -2κ auf unter zwei Prozent — das Rohr unter Cutoff ist ein Tunnelspalt in Blech. Jetzt die **Mikrowellentür-Bilanz**, die Kapitel 15 schuldig blieb: Das $w = 0{,}4$-Loch von dort hat $f_c = 1/(2 \cdot 0{,}4) = 1{,}25$, die Mode lag bei $f = 0{,}203$ — *sechsfach* unter Cutoff. Macht $\kappa =$

$2\pi\sqrt{1{,}25^2 - 0{,}203^2} = 7{,}75$ und durch die Wandtiefe $0{,}2$ den Leistungsfaktor $e^{-2\cdot 7{,}75\cdot 0{,}2} = e^{-3{,}1} \approx 0{,}045$ — und das ist nur die Durchgangsdämpfung; dazu kommt, dass die große 4,9-Wellenlängen-Mode an das winzige Loch kaum ankoppelt (deshalb maß Kapitel 15 sogar $Q \approx 70\,000$ statt „nur" zwanzigmal schlechter). Beim echten Mikrowellenherd sind die Verhältnisse noch krasser — das rechnet Ü 17.1 nach.

Was passiert *an* der Grenzfrequenz? Wir fahren mit einem breitbandigen Puls über die Klippe — ein Lauf, vierzig Frequenzen, geteilt durch die Referenz (das Spektrometer-Denken aus Kapitel 10/14):

```
# von oben: rohr_fluss(), KAPPA_ROHR, np, plt
P_sweep_ref, freqs_sweep = rohr_fluss(0.25, 0.2, 40, 0.0, dauer=400)
P_sweep, _ = rohr_fluss(0.25, 0.2, 40, 2.0, dauer=400)
T_sweep = P_sweep / P_sweep_ref

fig, achse = plt.subplots(figsize=(7, 4.0))
achse.semilogy(freqs_sweep, T_sweep, "o-", ms=3, color="C2",
               label="Meep (ein Puls + Referenz)")
theo = [np.exp(-2 * 2 * np.pi
               * np.sqrt(max(0.25**2 - f**2, 0)) * 2.0)
        for f in freqs_sweep]
achse.semilogy(freqs_sweep, theo, "k--", lw=1,
               label="e$^{-2\\kappa(f)\\,L}$ (parameterfrei)")
achse.axvline(0.25, color="k", ls=":", lw=1)
achse.annotate("$f_c$", (0.253, 3e-3))
achse.set(xlabel="Frequenz f", ylabel="Transmission T",
          ylim=(2e-4, 2),
          title="Die Cutoff-Klippe (Engstelle L = 2)")
achse.legend(fontsize=9)
fig.tight_layout()
plt.show()
unter = T_sweep[freqs_sweep < 0.17].max()
ueber = T_sweep[freqs_sweep > 0.30].min()
print(f"Unter der Klippe (f < 0,17): T < {unter:.2g};  "
      f"über der Klippe (f > 0,30): T > {ueber:.2g}")
```

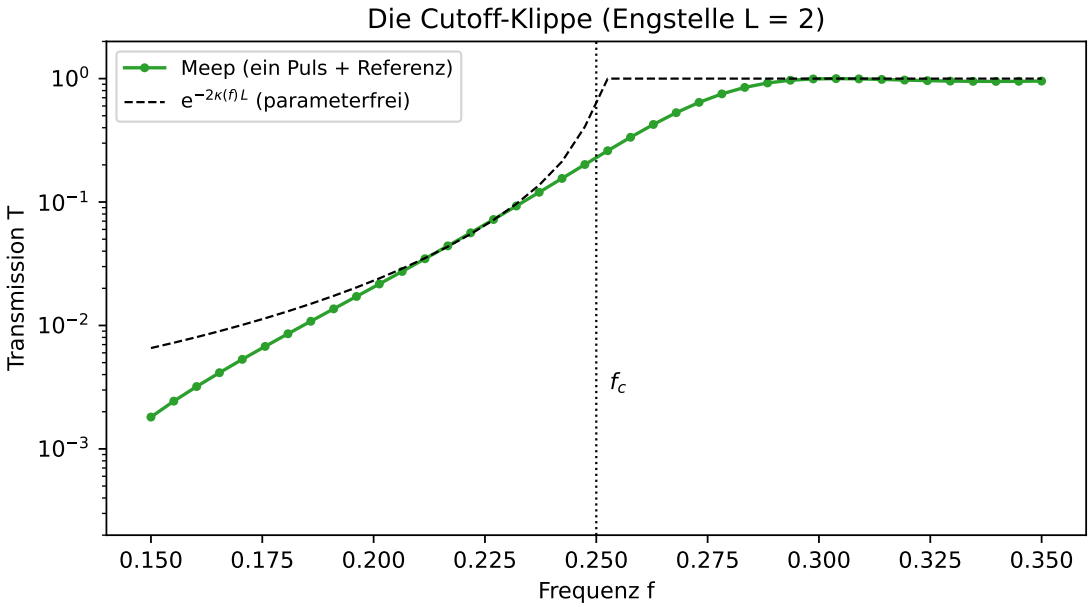


Abbildung 17.4: Transmission durch die Engstelle (L = 2) über der Frequenz, gemessen mit einem breitbandigen Puls und Referenzlauf. Unterhalb der Grenzfrequenz f_c = 0,25 (gepunktete Senkrechte) sperrt das Rohr exponentiell — die gestrichelte Linie ist e^(−2κ(f)L) ohne jeden Fitparameter —, oberhalb läuft die Welle praktisch ungehindert. Die Klippe ist steil, aber bei endlicher Rohrlänge nicht senkrecht: Genau bei f_c ist κ = 0, und ein kurzes Stück lässt dort die Hälfte durch. (Das Frequenzraster endet bei 0,35 — darüber meldet sich die m = 3-Mode des breiten Rohrs mit ihrer eigenen Klippe; siehe Kleingedrucktes.)

```
Unter der Klippe (f < 0,17): T < 0.0041;  über der Klippe (f > 0,30): T > 0.95
```

Drei Dekaden Sperrung auf der einen Seite, freie Fahrt auf der anderen, und der Übergang folgt der parameterfreien $\kappa(f)$-Kurve: Ein Stück Rohr ist ein **Hochpassfilter mit Exponentialflanke** — je länger das Stück, desto steiler (Ü 17.3 misst das nach).

17.3 Im Rohr wohnt ein Plasma

Oberhalb der Klippe läuft die Welle — aber wie schnell? Die Antwort steckt im Pythagoras aus Kapitel 9: Im Rohr setzt sich der Wellenvektor aus dem

festen Quer-Anteil $k_y = \pi m/w$ (die stehende Welle zwischen den Wänden) und dem freien Längs-Anteil k zusammen, $|k|^2 = (\pi m/w)^2 + k^2$, und mit $\omega = c\,|k|$ wird daraus

$$\omega^2 \;=\; \omega_c^2 + c^2 k^2.$$

Diese Formel sollte ein Déjà-vu auslösen: Es ist **exakt die Dispersionsrelation des Drude-Plasmas** aus Kapitel 16 — dort stand ω_p (die Eigenfrequenz der schwappenden Elektronenwolke), hier steht ω_c (die Eigenfrequenz der Querschwingung). Ein Metallrohr verhält sich für Wellen *genau* wie ein Plasma, nur dass die Grenzfrequenz aus der Geometrie kommt statt aus der Elektronendichte. Alles, was Kapitel 16 über das Drude-Medium hergeleitet hat, gilt wörtlich weiter: unterhalb evaneszent (das war unsere Klippe), oberhalb transparent mit $n_{\text{eff}} = \sqrt{1-(f_c/f)^2} < 1$ — also Phasengeschwindigkeit **über** c, Gruppengeschwindigkeit darunter, Produkt exakt c^2. Die Handrechnung für $f = 1{,}2\,f_c$: $\sqrt{1-1/1{,}44} = 0{,}5528$, also $v_g = 0{,}553\,c$ und $v_p = 1{,}809\,c$.

> Vorhersage-Punkt
>
> Gleich schicken wir ein Wellenpaket mit $f = 1{,}2\,f_c$ durch ein langes Rohr und messen mit den Kapitel-16-Werkzeugen. Kommt das Paket früher oder später an als ein gleich gestartetes Paket im freien Raum — und was machen die Wellenberge *im* Paket?

Das Rohr für diese Messung machen wir uns so schmal wie möglich, damit es billig bleibt: eine Zelle 120×1 — der Zellrand ist die Rohrwand, also $w = 1$ und $f_c = 0{,}5$; der Träger liegt bei $f = 0{,}6$. Zwei Sonden im Abstand 60 messen die Paket-Ankunftszeiten (Energie-Schwerpunkt), ein enges Sondenpaar (Abstand $0{,}5 < \lambda_{\text{Rohr}}/2$) die Trägerphase — wörtlich das Wettrennen-Rezept aus Kapitel 16. Erfolgskriterium: beide Tempi auf ein Prozent, Produkt c^2:

```
# von oben: mp, np
F_VG, FC_ENG = 0.6, 0.5
quelle = [mp.Source(mp.GaussianSource(F_VG, fwidth=F_VG / 12),
                    component=mp.Ez, center=mp.Vector3(-45, 0),
                    size=mp.Vector3(0, 1.0))]
sim = mp.Simulation(cell_size=mp.Vector3(120, 1.0), resolution=20,
                    sources=quelle,
                    boundary_layers=[mp.PML(3.0, direction=mp.X)])
SONDEN = [mp.Vector3(-30, 0.18), mp.Vector3(30, 0.18),
          mp.Vector3(30, 0.18), mp.Vector3(30.5, 0.18)]
```

```
reihen = [[] for _ in SONDEN]
DT_S = 0.05

def messen(sim):
    for r, p in zip(reihen, SONDEN):
        r.append(sim.get_field_point(mp.Ez, p).real)

sim.run(mp.at_every(DT_S, messen), until=520)
reihen = [np.array(r) for r in reihen]
zeiten = np.arange(len(reihen[0])) * DT_S

def ankunftszeit(reihe, sigma):
    """Zeitlicher Energie-Schwerpunkt, gefenstert (Kapitel 16)."""
    e = reihe**2
    t_roh = np.sum(zeiten * e) / np.sum(e)
    m = np.abs(zeiten - t_roh) < 3.5 * sigma
    return np.sum(zeiten[m] * e[m]) / np.sum(e[m])

def detektor_phase(reihe, f):
    """Kapitel-10-Detektor: Phase der Frequenz f."""
    co = np.sum(reihe * np.cos(2 * np.pi * f * zeiten))
    si = np.sum(reihe * np.sin(2 * np.pi * f * zeiten))
    return np.arctan2(si, co)

sig = (12 / F_VG) / 2
t1 = ankunftszeit(reihen[0], sig)
t2 = ankunftszeit(reihen[1], sig)
vg_rohr = 60.0 / (t2 - t1)
dphi = (detektor_phase(reihen[3], F_VG)
        - detektor_phase(reihen[2], F_VG)) % (2 * np.pi)
vp_rohr = 2 * np.pi * F_VG * 0.5 / dphi
wurzel = np.sqrt(1 - (FC_ENG / F_VG)**2)
print(f"Paket:  v_g = {vg_rohr:.4f} c   (Theorie {wurzel:.4f} c)")
print(f"Berge:  v_p = {vp_rohr:.4f} c   (Theorie {1/wurzel:.4f} c)")
print(f"Produkt v_p·v_g = {vp_rohr * vg_rohr:.4f} c²")
```

```
Paket:  v_g = 0.5525 c   (Theorie 0.5528 c)
Berge:  v_p = 1.8062 c   (Theorie 1.8091 c)
Produkt v_p·v_g = 0.9979 c²
```

Das Paket kommt *später* an als im freien Raum — es kriecht mit $0{,}55\,c$, weil die Welle effektiv im Zickzack zwischen den Wänden läuft und nur die Längskomponente vorankommt. Die Berge dagegen rasen mit $1{,}81\,c$ durchs Paket:

dieselbe Leuchtturm-Physik wie im Drude-Medium, diesmal in einem Stück Blech, das man kaufen kann. Wer je mit Satelliten-, Radar- oder Beschleunigertechnik zu tun hat, rechnet täglich mit diesen beiden Geschwindigkeiten — die Signallaufzeit eines Hohlleiters bestimmt v_g, seine Wellenlänge im Rohr ($\lambda_{\text{Rohr}} = v_p/f$, hier $3{,}0$ statt $1{,}67$ im Freien) bestimmt v_p.

17.4 Warum die Glasfaser ihr Licht behält

Damit ist auch die Kapitel-12-Zusage fällig — und sie braucht keinen neuen Lauf mehr, nur die Summe dieses Kapitels. Eine Glasfaser führt ihr Licht per Totalreflexion: Der Kern ist optisch dichter als der Mantel, flache Strahlen werden an der Grenze total reflektiert, kilometerlang, milliardenfach. Nach diesem Kapitel weiß man, wie bemerkenswert das ist: *Jede* dieser Reflexionen hat einen evaneszenten Saum, der ein Stück weit aus dem Kern hinausreicht — und **tunneln würde, sobald er etwas findet**. Eine Berührung, ein Fettfilm, ein zweites Stück Glas in Saumweite: Licht koppelt aus, die Faser „leckt" (genau so funktionieren FTIR-Sensoren und abgegriffene Fasern). Die Lösung der Faserbauer: Der Saum bekommt *garantiert* nichts zu fassen. Der Kern (wenige bis ~50 µm) steckt in einem dicken Glasmantel (125 µm) — der Saum reicht nur Bruchteile einer Wellenlänge weit hinein, dahinter kommt Schutzschicht. Die Totalreflexion bleibt, was sie verspricht, weil man dem Tunneln nichts zum Hinüberspringen anbietet. Und der Richtkoppler in Ü 17.4 zeigt die Kehrseite als Werkzeug: Legt man *absichtlich* eine zweite Faser in den Saum, pendelt das Licht kontrolliert hinüber — so werden Glasfasersignale verzweigt, ganz ohne die Faser zu öffnen.

17.5 Das Kapitel-Programm

Das Programm `kap17_tunneln_cutoff.py` (im Buchpaket unter `programme/kap17/`) bündelt alle Messungen: FTIR-Reihe mit Bündel-Theorie, Engstellen-Reihe, Klippen-Sweep, Rohr-Tempi und den Richtkoppler aus Ü 17.4. Jede Behauptung steht als `assert`; der Lauf braucht knapp eine Minute und endet mit „Alle Tests bestanden."

Merkkasten

- **Evaneszent heißt nicht leer**: Der Saum trägt Feld, und findet er in Reichweite ($\sim \delta = 1/\kappa$) ein Medium, tunnelt Leistung hinüber — $T \approx e^{-2\kappa d}$, exponentiell empfindlich am Abstand.

- **Strahlen sind Winkelbündel**: Bei exponentiellen Effekten gewinnen die Flügel des Bündels; wer sie ignoriert, misst „Widersprüche" zur korrekten Formel.
- **Cutoff**: Ein Kanal der Breite w trägt unterhalb von $f_c = mc/2w$ keine laufende Welle — ein Loch ist ein kurzes Rohr unter Cutoff, darum ist die Mikrowellentür dicht.
- **Rohr = Plasma aus Geometrie**: $\omega^2 = \omega_c^2 + c^2k^2$, also $v_p > c > v_g$ und $v_p \cdot v_g = c^2$ — wörtlich die Drude-Formeln aus Kapitel 16.
- Tunneln (FTIR) und Cutoff sind dieselbe Mathematik: ein verbotener Bereich, ein κ, ein Exponentialgesetz — und die Quantenmechanik benutzt exakt dieselbe sinh-Formel.

Roter Faden

Rückwärts: Der Saum samt κ-Formel stammt aus **Kapitel 12** (dort vermessen, hier angezapft), die Flux-und-Referenz-Methode aus **Kapitel 11/12**, das Winkelbündel-Argument aus **Kapitel 12/13** — hier wurde es erstmals *quantitativ* zur Theorielinie. **Kapitel 15** lieferte die stehenden Querwellen (f_c ist ein halber Kapitel-15-Kasten), den Zellrand-als-Metallwand-Trick und die offene Mikrowellentür-Rechnung. Aus **Kapitel 16** kommen das komplette Tempo-Messbesteck (Energie-Schwerpunkt, Sondenpaar, Hand-Detektor aus **Kapitel 10**) und die Drude-Form der Dispersionsrelation — das Rohr ist ihr Geometrie-Zwilling. Vorwärts: **Kapitel 20** stellt die Frage, die hier offen bleiben musste — wie schnell ist eigentlich *Tunneln*, und was schützt die Kausalität? **Kapitel 22** bringt mit Leitungen die Gegenspieler der Hohlleiter, **Kapitel 26** baut Cutoffs aus Periodik (Bandlücken: „verbotene Bereiche nach Maß"), und **Kapitel 27** macht aus dem Richtkoppler der Übung 17.4 Ringresonatoren und Filter. **Kapitel 30** schließlich löst die Quanten-Anleihe ein: Dieselbe sinh-Formel, aber $|E|^2$ ist dann eine Wahrscheinlichkeit (F16).

Übungen

Ü 17.1 (Verstehen). Die Mikrowellentür in SI-Einheiten: Der Herd arbeitet bei $f = 2{,}45$ GHz, die Löcher im Türgitter haben $w \approx 2$ mm, das Blech ist $d \approx 1$ mm dick. Berechne die Grenzfrequenz eines Lochs (als Schlitz-Modell $f_c = c/2w$), das κ bei Betriebsfrequenz und den Durchgangsfaktor $e^{-2\kappa d}$. Warum siehst du trotzdem deinen Teller durchs Gitter?

Musterlösung zu Ü 17.1

```
# von oben: np
c0 = 3e8
f, w, d = 2.45e9, 2e-3, 1e-3
f_c = c0 / (2 * w)
kappa = (2 * np.pi / c0) * np.sqrt(f_c**2 - f**2)
print(f"f_c = {f_c/1e9:.0f} GHz  (Betrieb: {f/1e9:.2f} GHz "
      f"-> Faktor {f_c/f:.0f} unter Cutoff)")
print(f"kappa = {kappa:.0f} 1/m  -> e^(-2 kappa d) = "
      f"{np.exp(-2*kappa*d):.3f} pro {d*1000:.0f} mm Blech")
```

```
f_c = 75 GHz  (Betrieb: 2.45 GHz -> Faktor 31 unter Cutoff)
kappa = 1570 1/m  -> e^(-2 kappa d) = 0.043 pro 1 mm Blech
```

Das Loch liegt **dreißigfach** unter Cutoff. Der reine Durchgangsfaktor durch 1 mm Blech ist zwar „nur" $\approx 0{,}04$ — aber wie in Kapitel 15 gemessen, ist die eigentliche Sperre die **Ankopplung**: Eine 12-cm-Welle übergibt an ein 2-mm-Loch fast nichts (und das Ausgekoppelte muss auf der anderen Seite wieder an die Freiraumwelle ankoppeln). Beides zusammen drückt die Leckage um viele Größenordnungen unter die Grenzwerte; unsere Schlitz-Rechnung ist die ehrliche Obergrenze für den Rohranteil. Sichtbares Licht dagegen hat $\lambda \approx 0{,}5$ µm: Für $f \gg f_c$ ist das Loch ein viertausend Wellenlängen breites Scheunentor — deshalb siehst du den Teller, während die Mikrowelle eingesperrt bleibt. Struktur gegen Wellenlänge entscheidet, nicht „Loch oder kein Loch" (das Kapitel-15-Motiv).

Ü 17.2 (Verändern). Der Strahlteiler-Regler: Miss die FTIR-Transmission bei festem Spalt $d = 0{,}6$ für $\theta = 45°, 50°, 55°, 60°$ (je mit eigener Referenz!). Erkläre den Trend mit der κ-Formel — und warum bei 45° auch die Bündel-Mittelung sichtbar ungenauer wird.

Musterlösung zu Ü 17.2

```
# von oben: ftir_lauf(), theorie_T(), theorie_T_buendel(), np
for th in (45, 50, 55, 60):
    T = ftir_lauf(th, 0.6) / ftir_lauf(th, 0.0)
    print(f"theta = {th}°: T = {T:.4f}   "
          f"(eben {theorie_T(th, 0.6):.4f}, "
          f"Bündel {theorie_T_buendel(th, 0.6):.4f})")
```

```
theta = 45°: T = 0.1197   (eben 0.1037, Bündel 0.2052)
theta = 50°: T = 0.0516   (eben 0.0421, Bündel 0.0485)
theta = 55°: T = 0.0240   (eben 0.0176, Bündel 0.0197)
theta = 60°: T = 0.0135   (eben 0.0076, Bündel 0.0085)
```

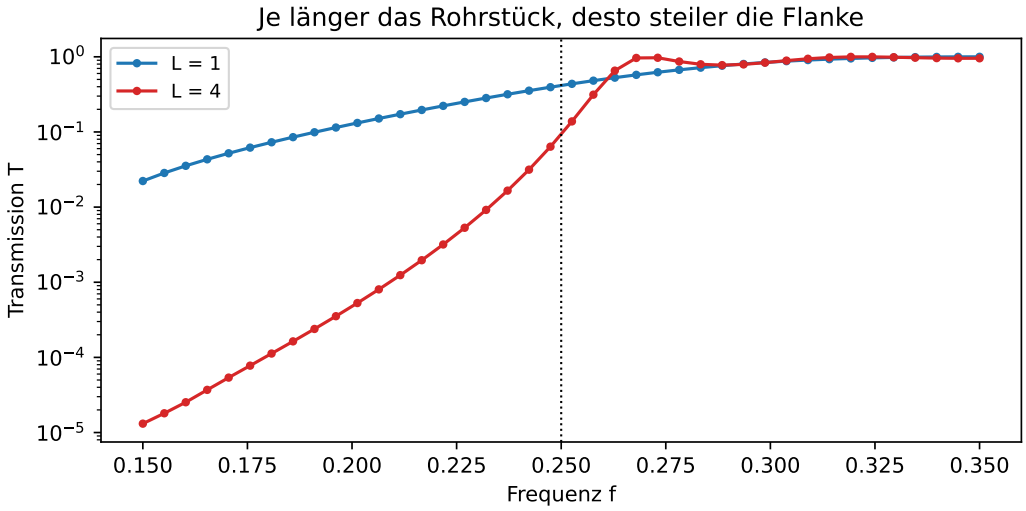
Je steiler der Einfall, desto größer $n \sin\theta$, desto größer κ — desto kürzer die Brücke: T fällt von 45° nach 60° um fast eine Größenordnung. Ein Prismenpaar hat also **zwei** Regler: den Spalt (exponentiell) und den Winkel (über κ). Bei $45°$ liegt die Messung allerdings deutlich *unter* der Bündel-Linie: Der Mittelwinkel ist nur noch $3°$ vom Grenzwinkel entfernt, die unteren Bündel-Flügel geraten *unter* θ_c — und für sie ist unsere Bündel-Formel zu grob (sie setzt dort pauschal $T = 1$, real gelten die Fresnel-Formeln aus Kapitel 13 mit Doppelgrenzfläche). Nahe an einer Klippe braucht auch die Mittelung feineres Werkzeug — dieselbe Demut, die Kapitel 12 an den Bündel-Flügeln gelernt hat.

Ü 17.3 (Verändern). Die weiche Klippe: Wiederhole den $T(f)$-Sweep aus dem Kapitel für Engstellen der Länge $L = 1$ und $L = 4$. Wie ändert sich die Flanke unterhalb von f_c — und um welchen Faktor unterscheiden sich die beiden Kurven bei $f = 0{,}2$? (Vorhersage per κ-Formel, dann messen.)

Musterlösung zu Ü 17.3

Vorhersage: Unterhalb der Klippe gilt $T \approx Ce^{-2\kappa(f)L}$ — die *Steigung im Logarithmus* wächst proportional zu L. Bei $f = 0{,}2$ ist $\kappa = 0{,}9425$, also Verhältnis $T_{L=1}/T_{L=4} \approx e^{2\kappa\cdot 3} = e^{5{,}65} \approx 285$.

```python
# von oben: rohr_fluss(), np, plt
P_ref_u, freqs_u = rohr_fluss(0.25, 0.2, 40, 0.0, dauer=400)
fig, achse = plt.subplots(figsize=(7, 3.6))
T_bei_02 = {}
for L, farbe in ((1.0, "C0"), (4.0, "C3")):
    P_L, _ = rohr_fluss(0.25, 0.2, 40, L, dauer=400)
    T_L = P_L / P_ref_u
    achse.semilogy(freqs_u, T_L, "o-", ms=3, color=farbe,
                   label=f"L = {L:.0f}")
    j = np.argmin(np.abs(freqs_u - 0.2))
    T_bei_02[L] = T_L[j]
achse.axvline(0.25, color="k", ls=":", lw=1)
achse.set(xlabel="Frequenz f", ylabel="Transmission T",
          title="Je länger das Rohrstück, desto steiler die Flanke")
achse.legend(fontsize=9)
fig.tight_layout()
plt.show()
print(f"T(f = 0,2):  L=1: {T_bei_02[1.0]:.3g},  L=4: "
      f"{T_bei_02[4.0]:.3g},  Verhältnis = "
      f"{T_bei_02[1.0] / T_bei_02[4.0]:.0f}  (Vorhersage ~285)")
```




```
T(f = 0,2):  L=1: 0.132,  L=4: 0.000529,  Verhältnis = 249  (Vorhersage ~285)
```

Ein kurzes Rohrstück ist ein matschiger Filter, ein langes ein scharfer: Die Flankensteilheit ist schlicht $2\kappa(f)L$ im Exponenten. Genau deshalb sind Hohlleiter-Filter und Tür-Bleche *dick* ausgeführt — und genau deshalb war die Klippe im Kapitel (L = 2) steil, aber nicht senkrecht.

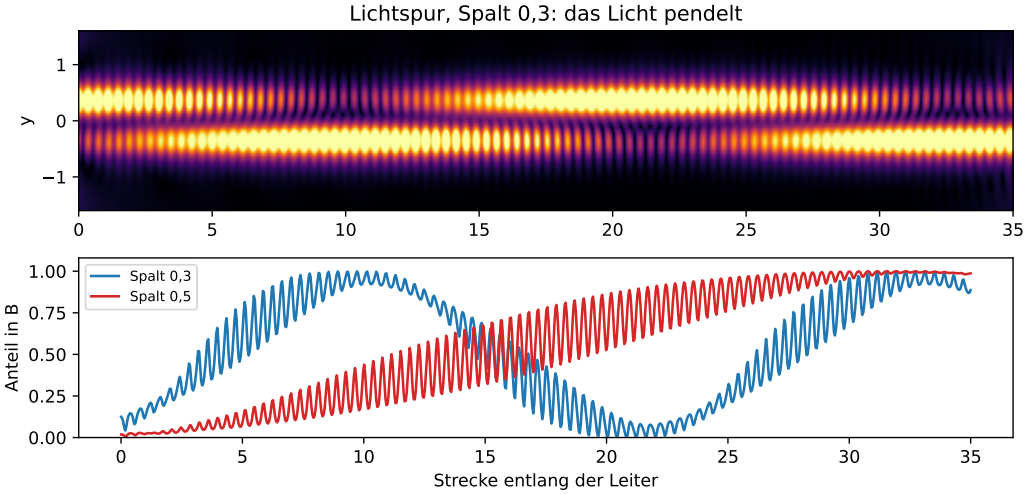
Ü 17.4 (Übertragen). Der Richtkoppler: Lege zwei Glasstreifen (Breite $0{,}4$, $n = 1{,}5$) parallel mit Spalt g und speise Licht ($f = 1$, Dauerton) in nur *einen* ein. Miss den Leistungsanteil im zweiten Streifen entlang der Strecke für $g = 0{,}3$ und $g = 0{,}5$. Was passiert — und was hat das mit der Schwebung aus Kapitel 16 zu tun?

Musterlösung zu Ü 17.4

```
# von oben: mp, np, plt, GLAS
W_S = 0.4

def koppler(spalt):
    y_a, y_b = +(spalt + W_S) / 2, -(spalt + W_S) / 2
    geom = [mp.Block(size=mp.Vector3(40, W_S, mp.inf),
                     center=mp.Vector3(0, y), material=GLAS)
            for y in (y_a, y_b)]
    quelle = [mp.Source(mp.ContinuousSource(1.0, is_integrated=True),
                        component=mp.Ez, center=mp.Vector3(-18, y_a),
                        size=mp.Vector3(0, W_S))]
    sim = mp.Simulation(cell_size=mp.Vector3(44, 6), resolution=24,
                        sources=quelle, geometry=geom,
                        boundary_layers=[mp.PML(2.0)])
    sim.run(until=120)
    summe = None
    for _ in range(10):                  # eine Periode mitteln
        sim.run(until=0.1)
        ez = sim.get_array(component=mp.Ez)
        summe = ez**2 if summe is None else summe + ez**2
    nx, ny = summe.shape
    xs = np.linspace(-22, 22, nx)
    ys = np.linspace(-3, 3, ny)
    ja, jb = (np.argmin(np.abs(ys - y)) for y in (y_a, y_b))
    P_a = summe[:, ja - 2:ja + 3].sum(axis=1)
    P_b = summe[:, jb - 2:jb + 3].sum(axis=1)
    m = (xs > -16) & (xs < 19)
    return xs[m] + 16, P_b[m] / (P_a[m] + P_b[m]), summe

fig, (oben, unten) = plt.subplots(2, 1, figsize=(8.5, 4.2))
strecke03, anteil03, bild = koppler(0.3)
strecke05, anteil05, _ = koppler(0.5)
oben.imshow(np.sqrt(bild.T), origin="lower", cmap="inferno",
            extent=[-6, 38, -3, 3], aspect="auto",   # x als Strecke
            vmax=0.6 * np.sqrt(bild).max())
oben.set(xlim=(0, 35), ylim=(-1.6, 1.6), ylabel="y",
         title="Lichtspur, Spalt 0,3: das Licht pendelt")
for strecke, anteil, farbe, name in (
        (strecke03, anteil03, "C0", "Spalt 0,3"),
        (strecke05, anteil05, "C3", "Spalt 0,5")):
    unten.plot(strecke, anteil, color=farbe, label=name)
unten.set(xlabel="Strecke entlang der Leiter",
          ylabel="Anteil in B", ylim=(0, 1.08))
unten.legend(fontsize=8, loc="upper left")
fig.tight_layout()
plt.show()
for name, strecke, anteil in (("0,3", strecke03, anteil03),
                              ("0,5", strecke05, anteil05)):
    L_halb = strecke[np.argmax(anteil > 0.5)] - strecke[0]
    print(f"Spalt {name}: halbe Übergabe nach {L_halb:.1f}, "
          f"maximaler B-Anteil {anteil.max():.2f}")
```




```
Spalt 0,3: halbe Übergabe nach 3.2, maximaler B-Anteil 1.00
Spalt 0,5: halbe Übergabe nach 11.1, maximaler B-Anteil 1.00
```

Das Licht **pendelt vollständig** zwischen den Streifen hin und her — beim engen Spalt mehrmals auf der Strecke, beim weiten einmal: Die Übergabelänge wächst steil (exponentiell, denn gekoppelt wird über den Saum!) mit dem Spalt. Der Mechanismus ist die Kapitel-16-Schwebung in neuem Gewand: Das Doppelsystem hat zwei Eigenwellen — eine symmetrische und eine antisymmetrische Supermode — mit *leicht verschiedenen* Ausbreitungskonstanten. Speist man nur Streifen A, regt man beide gleich an, und ihre Überlagerung wandert mit der Differenz der beiden Wellenzahlen periodisch von A nach B und zurück: eine Schwebung im Raum statt in der Zeit. (Die feine Welligkeit auf den Kurven ist eine stehende Restwelle von den Streifenenden.) Technisch ist das der **Richtkoppler** — Glasfasersignale werden genau so verzweigt, und Kapitel 27 koppelt damit Ringresonatoren an.

Das Kleingedruckte

Vier Randnotizen. **Erstens:** Wie *schnell* ist Tunneln? Die Frage ist berühmt und heikel: Naive „Tunnelzeiten" (etwa der Vergleich der Paket-Gipfel vor und hinter der Barriere) können scheinbar Überlichtgeschwindigkeit ergeben (Hartman-Effekt) — aufgelöst wird das wie in Kapitel 16: Der durchgelassene Puls ist ein *umgeformter Vorderteil* des einlaufenden, keine Botschaft überholt die Front. Kapitel 20 nimmt sich Kausalität und Zeitpfeil grundsätzlich vor. **Zweitens:** Unser Klippen-Sweep endete bei $f = 0{,}35$,

weil bei $0{,}375$ die $m = 3$-Mode des breiten Rohrs ihre eigene Klippe hat (die gleichmäßige Quelllinie regt $m = 1, 3, 5, \ldots$ an; $m = 2$ bleibt aus Symmetriegründen stumm). Oberhalb wird die Transmission zur Verhandlungssache *zwischen* Moden — Hohlleiter betreibt man deshalb gern „einmodig", im Band zwischen $f_{c,1}$ und der nächsten nutzbaren Klippe. **Drittens:** Die FTIR-Formel galt für unsere s-Polarisation; für p-Polarisation (Kapitel 13) ändert sich der Vorfaktor V (und nahe am Brewster-Winkel wird die Geschichte reicher) — der Exponent $e^{-2\kappa d}$ bleibt. **Viertens:** Für Löcher, die klein gegen die Wellenlänge sind, kommt zur Cutoff-Dämpfung die miserable **Ankopplung** (in der Streutheorie: Bethes Lochblenden-Theorie, Streuung $\propto (w/\lambda)^4$ und schlimmer) — unsere Rohr-Rechnung ist deshalb stets die freundliche Obergrenze der Leckage; real ist es weniger. Gut für Mikrowellentüren, schlecht für jeden, der durch ein Schlüsselloch WLAN empfangen will.

18 Poynting und Energiefluss

Du legst den Lichtschalter um, und die Lampe ist an — „sofort”, über drei Meter Kabel. Jetzt die unbequeme Zahl dazu: Die Elektronen, die im Kupferdraht den Strom bilden, **kriechen**. Bei zehn Ampere durch eine gewöhnliche 1,5-mm²-Leitung beträgt ihre Driftgeschwindigkeit etwa einen halben **Millimeter pro Sekunde** — für die drei Meter bis zur Lampe bräuchte ein Elektron anderthalb Stunden. Bei Wechselstrom kommen sie überhaupt nie an: Sie zittern nur wenige Mikrometer um ihren Platz. Irgendetwas anderes muss die Energie zur Lampe bringen, und zwar mit nahezu Lichtgeschwindigkeit. Dieses Kapitel zeigt, *wer* — und vor allem: *auf welchem Weg*. Die Antwort („im Feld neben dem Draht, nicht im Draht”) gehört zu den Einsichten der Elektrodynamik, die auch erfahrene Praktiker noch verblüffen.

Dafür sammeln wir Schulden ein. Kapitel 1 hat beim Kreuzprodukt versprochen, dass $\vec{S} = \vec{E} \times \vec{H}$ einmal „die Energie jeder Funkwelle” tragen wird; Kapitel 3 schuldet uns die „exakte Buchhaltung” der Feldenergie; Kapitel 4 behauptete, dass elektrischer und magnetischer Topf einer Lichtwelle *exakt gleich viel* Energie enthalten (obwohl die Zahlenwerte um den Faktor c auseinanderliegen); Kapitel 6 hat die Verlustleistung $\vec{J} \cdot \vec{E}$ angekündigt; und Kapitel 16/17 haben wiederholt behauptet, Energie reise mit der Gruppengeschwindigkeit — gemessen haben wir bisher nur, dass *Pakete* das tun. Alles davon wird jetzt eingelöst, und fast alles davon wird *gemessen*.

Lernziele

Nach diesem Kapitel kannst du …

1. … die Energiedichten $u_E = \frac{\varepsilon_0}{2} E^2$ und $u_B = \frac{1}{2\mu_0} B^2$ berechnen und zeigen, dass eine laufende Welle beide Töpfe exakt gleich füllt — eine stehende Welle dagegen zwischen ihnen pendelt,
2. … den Poynting-Satz $\partial_t u + \nabla \cdot \vec{S} = -\vec{J} \cdot \vec{E}$ herleiten und als Buchhaltung im FDTD-Lauf verifizieren — im Vakuum auf Maschinengenauigkeit,
3. … die $\vec{S}$-Karte einer Leitung lesen: Die Leistung fließt im Feld zwischen den Leitern; Verluste sind der kleine Anteil, der seitlich ins Metall kippt,

4. ... das Lampen-Paradox auflösen (Driftgeschwindigkeit ausrechnen — und erklären, warum sie nicht zählt),
5. ... die Energie-Transportgeschwindigkeit $v_E = \langle S\rangle/\langle u\rangle$ messen und zeigen, dass sie v_g ist, nicht v_p.

18.1 Zwei Töpfe: die Energie im Feld

Die Zutaten stehen seit Teil I bereit. Ein elektrisches Feld speichert pro Volumen die Energie $u_E = \frac{\varepsilon_0}{2}E^2$, ein magnetisches $u_B = \frac{1}{2\mu_0}B^2$ — beides Skalarfelder (an jedem Ort eine Zahl, Einheit J/m³). Jetzt die Kapitel-4-Schuld, mit den Zahlen von dort: Sonnenlicht am Boden bringt rund $I = 1000$ W/m², das gehört (gleich rechnen wir nach, warum) zu einem E-Feld mit Spitzenwert $E_0 \approx 868$ V/m — und zu einem B-Feld von nur $B_0 = E_0/c \approx 2{,}9$ µT, zwanzigmal schwächer als das Erdmagnetfeld. Sieht aus, als wäre das B-Feld eine Randerscheinung. Die Energierechnung sagt das Gegenteil:

$$u_B = \frac{B_0^2}{2\mu_0} = \frac{(E_0/c)^2}{2\mu_0} = \frac{E_0^2}{2\mu_0 c^2} = \frac{\varepsilon_0}{2}E_0^2 = u_E,$$

denn $c^2 = 1/\varepsilon_0\mu_0$ (Kapitel 3) — der „Faktor c Unterschied" der Feldstärken wird von den ungleichen Vorfaktoren exakt geschluckt. Mit Zahlen: $u_E = \frac{1}{2}\cdot 8{,}85\cdot 10^{-12}\cdot 868^2 = 3{,}3$ µJ/m³ und $u_B = (2{,}89\cdot 10^{-6})^2/(2\cdot 1{,}257\cdot 10^{-6}) = 3{,}3$ µJ/m³. **Beide Töpfe sind gleich voll** — bei jeder laufenden Welle, denn dort gilt punktweise $B = E/c$ (Kapitel 4).

Bei einer *stehenden* Welle gilt das nicht, und das macht sie zum perfekten Testfall:

> Vorhersage-Punkt
>
> Eine stehende Welle schwingt im Takt: Zweimal pro Periode ist das E-Feld **überall gleichzeitig null**. Die Welle hat aber ständig dieselbe Gesamtenergie. Wo steckt sie in diesem Moment — und wo steckt sie eine Viertelperiode später?

Das Experiment dazu im 1D-Labor aus Kapitel 6 (zwei Läufe, ein Bild). **Frage:** Füllt der laufende Puls beide Töpfe gleich — und pendelt die stehende Welle zwischen ihnen? **Bühne:** das vertraute 12-m-Gitter, $\Delta x = 1$ cm, $S = 1$. **Anfangsbedingungen:** Lauf 1 startet einen rechtslaufenden Gauß-Puls (E und B gemeinsam gesetzt, $B = -E/c$ — so läuft er nur nach rechts, Kapitel 5); Lauf 2 startet die Grundmode des geschlossenen Kastens ($E =$

$\sin(\pi x/L)$, $B = 0$, PEC-Ränder) und wir fotografieren u_E und u_B bei $t = 0$ und eine Viertelperiode später. **Erfolgskriterium:** Lauf 1: $u_B/u_E = 1$ im Puls; Lauf 2: die Energie wechselt vollständig den Topf.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.constants import c as C
from scipy.constants import epsilon_0, mu_0

N, DX = 1201, 0.01
DT = DX / C                                # S = 1 (Kapitel 5)
X = np.arange(N) * DX
L = (N - 1) * DX

def schritt(E, B, pec=False):
    """Ein Leapfrog-Schritt; Mur- oder PEC-Ränder (Kapitel 5/6)."""
    B += (DT / DX) * np.diff(E)
    E_alt = E.copy()
    E[1:-1] += (C**2 * DT / DX) * np.diff(B)
    if not pec:
        E[0], E[-1] = E_alt[1], E_alt[-2]  # Mur bei S = 1
    return E_alt

# Lauf 1: rechtslaufender Puls (B = -E/c am Halbpunkt, Kapitel 5)
E1 = np.exp(-((X - 3.0) / 0.3)**2)
B1 = -0.5 * (E1[:-1] + E1[1:]) / C
for n in range(200):                       # Puls wandert 3 m -> 5 m
    schritt(E1, B1)

# Lauf 2: stehende Grundmode im PEC-Kasten
E2 = np.sin(np.pi * X / L)
B2 = np.zeros(N - 1)
T_periode = 2 * L / C                      # Grundmode: λ = 2L
u_paare = [(0.5 * epsilon_0 * E2**2, 0.5 / mu_0 * B2**2)]   # t = 0
for n in range(int(T_periode / 4 / DT)):   # eine Viertelperiode weiter
    schritt(E2, B2, pec=True)
u_paare.append((0.5 * epsilon_0 * E2**2, 0.5 / mu_0 * B2**2))

fig, (links, rechts) = plt.subplots(1, 2, figsize=(9, 3.6))
links.plot(X, 0.5 * epsilon_0 * E1**2 * 1e12, "C0",
           label="$u_E$")
links.plot(X[:-1] + DX / 2, 0.5 / mu_0 * B1**2 * 1e12, "k:",
           lw=2, label="$u_B$")
links.set(xlim=(4.0, 6.0), xlabel="x (m)",
```

```
          ylabel="Energiedichte (µJ/m³ je Amplitude 1 V/m)",
          title="laufender Puls: beide Töpfe gleich")
links.legend(fontsize=9)
for (u_e, u_b), stil, name in ((u_paare[0], "-", "t = 0"),
                               (u_paare[1], "--", "t = T/4")):
    rechts.plot(X, u_e * 1e12, "C0" + stil, label=f"$u_E$ ({name})")
    rechts.plot(X[:-1] + DX / 2, u_b * 1e12, "C3" + stil,
                label=f"$u_B$ ({name})")
rechts.set(xlabel="x (m)", ylabel="Energiedichte (µJ/m³)",
           title="stehende Welle: die Energie pendelt")
rechts.legend(fontsize=8)
fig.tight_layout()
plt.show()
u_e1 = 0.5 * epsilon_0 * E1**2
u_b1 = 0.5 / mu_0 * B1**2
m = u_e1[:-1] > 0.01 * u_e1.max()
print(f"laufender Puls: u_B/u_E im Puls = "
      f"{(u_b1[m] / u_e1[:-1][m]).mean():.4f}")
print(f"stehende Welle: E-Topf bei t = 0: "
      f"{np.sum(u_paare[0][0]):.3e},  bei T/4: "
      f"{np.sum(u_paare[1][0]):.3e}  (alles umgezogen)")
```

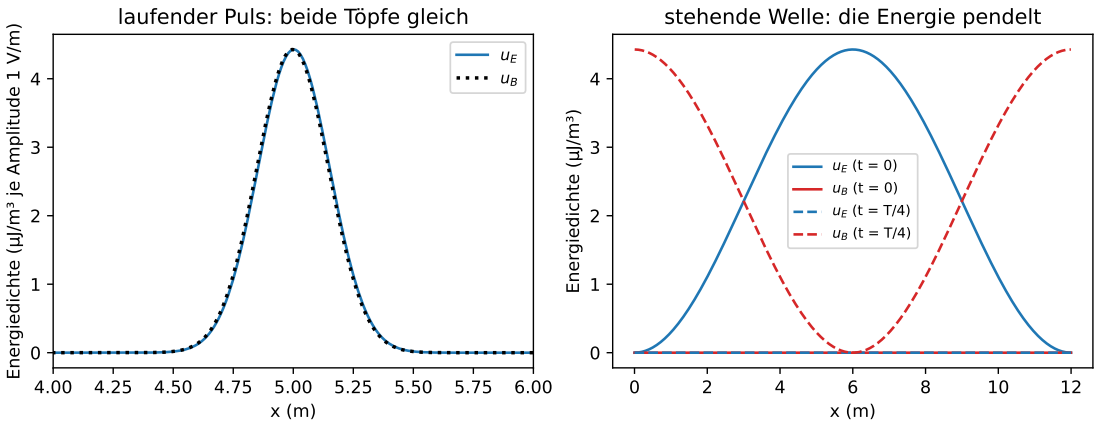


Abbildung 18.1: Links: Beim rechtslaufenden Puls (nach 200 Schritten) liegen u_E (Linie) und u_B (gepunktet) deckungsgleich übereinander — eine laufende Welle füllt beide Töpfe exakt gleich. Rechts: Die stehende Grundmode des geschlossenen Kastens bei t = 0 (alles im E-Topf, ein Bauch in der Mitte) und eine Viertelperiode später (alles in den B-Topf umgezogen, mit Bäuchen an den Wänden — B ist dort am stärksten, wo E einen Knoten hat). Die Energie schwappt zweimal pro Periode vollständig um.

```
laufender Puls: u_B/u_E im Puls = 1.0057
stehende Welle: E-Topf bei t = 0: 2.656e-09,  bei T/4: 4.551e-15  (alles umgezogen)
```

Die Antwort auf den Vorhersage-Punkt steht im rechten Bild: Im E-Nulldurchgang ist die Energie **komplett im Magnetfeld** — und zwar räumlich umsortiert, denn die B-Bäuche sitzen an den E-Knoten. Die stehende Welle ist ein Pendel: E-Topf und B-Topf tauschen die Energie zweimal pro Periode vollständig aus, wie Spannkraft und Bewegung beim Federpendel. (Kapitel 19 baut auf genau diesem Bild den Schwingkreis.)

18.2 Die Buchhaltung: der Poynting-Satz

Energie sitzt also im Feld — aber sie *bleibt* nicht sitzen. Wie reist sie? Die Antwort folgt aus den Maxwell-Gleichungen mit nichts als der Produktregel; in 1D (Felder $E_z(x)$, $B_y(x)$, unsere Laborwelt) passt die Rechnung auf wenige Zeilen. Die Zeitableitung der Energiedichte ist

$$\frac{\partial u}{\partial t} = \varepsilon_0 E \frac{\partial E}{\partial t} + \frac{1}{\mu_0} B \frac{\partial B}{\partial t},$$

und für beide Zeitableitungen haben wir seit Kapitel 5 Maxwell-Ersatz: $\varepsilon_0 \partial_t E = -\partial_x B/\mu_0 - J$ und $\partial_t B = -\partial_x E$ (Vorzeichen in unserer 1D-Konvention). Einsetzen, Produktregel rückwärts lesen ($E\,\partial_x B + B\,\partial_x E = \partial_x(EB)$), fertig:

$$\boxed{\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{E\,B}{\mu_0}\right) = -\,J\,E}$$

Das ist der **Poynting-Satz**, und er hat exakt die Gestalt einer Kontinuitätsgleichung: Was an Energiedichte verschwindet, ist entweder als Strom $S = EB/\mu_0$ (allgemein: $\vec{S} = \vec{E} \times \vec{H}$, das Kreuzprodukt aus Kapitel 1) **weitergeflossen** oder als $J\,E$ an Ladungen **verarbeitet** worden — beim Ohmschen $J = \sigma E$ heißt das: zu Wärme geworden, σE^2, die Wärmeleistung aus dem Kapitel-6-Versprechen. Es gibt kein viertes Schicksal. Energie verschwindet nie, sie hat immer einen Weg oder einen Empfänger — das ist die „exakte Buchhaltung", die Kapitel 3 angekündigt hat.

Wie exakt? Das lässt sich auf dem Gitter wörtlich prüfen — und dabei wartet eine Pointe. **Frage:** Ist die Gesamtenergie eines geschlossenen, verlustfreien Kastens konstant? **Bühne:** das 1D-Gitter mit PEC-Enden (nichts entkommt), darin ein ruhender Anfangspuls, 4000 Schritte. **Gemessen wird** die Gesamtenergie $U(t)$ — auf zwei Arten: einmal „naiv" als $\sum_i \frac{\varepsilon_0}{2}(E_i^n)^2\,\Delta x + \sum_j \frac{1}{2\mu_0}(B_j^{n+1/2})^2\,\Delta x$, und einmal mit einem kleinen, gleich zu rechtfertigenden Kniff. **Erfolgskriterium:** konstant — aber wie konstant?

```
# von oben: np, plt, N, DX, DT, X, epsilon_0, mu_0, C
E = np.exp(-((X - 3.0) / 0.3)**2)
B = np.zeros(N - 1)
U_naiv, U_versetzt = [], []
for n in range(4000):
    B += (DT / DX) * np.diff(E)
    E_alt = E.copy()
    E[1:-1] += (C**2 * DT / DX) * np.diff(B)    # PEC: Ränder bleiben 0
    u_b = 0.5 / mu_0 * np.sum(B**2) * DX
    U_naiv.append(0.5 * epsilon_0 * np.sum(E**2) * DX + u_b)
    U_versetzt.append(0.5 * epsilon_0 * np.sum(E_alt * E) * DX + u_b)
U_naiv, U_versetzt = np.array(U_naiv), np.array(U_versetzt)
```

```
fig, achse = plt.subplots(figsize=(7, 3.6))
t_ns = np.arange(4000) * DT * 1e9
achse.plot(t_ns, U_naiv / U_naiv.mean(), "C0", lw=0.7,
           label="naiv: $\\Sigma\\,(E^n)^2$")
achse.plot(t_ns, U_versetzt / U_versetzt.mean(), "k", lw=1.4,
           label="versetzt: $\\Sigma\\,E^n E^{n+1}$")
achse.set(xlabel="Zeit (ns)", ylabel="U / Mittelwert",
          title="Energie im geschlossenen Kasten: zwei Buchungsarten")
achse.legend(fontsize=9)
fig.tight_layout()
plt.show()
print(f"naiv:     relative Schwankung "
      f"{(U_naiv.max() - U_naiv.min()) / U_naiv.mean():.2e}")
print(f"versetzt: relative Schwankung "
      f"{(U_versetzt.max() - U_versetzt.min()) / U_versetzt.mean():.2e}")
```

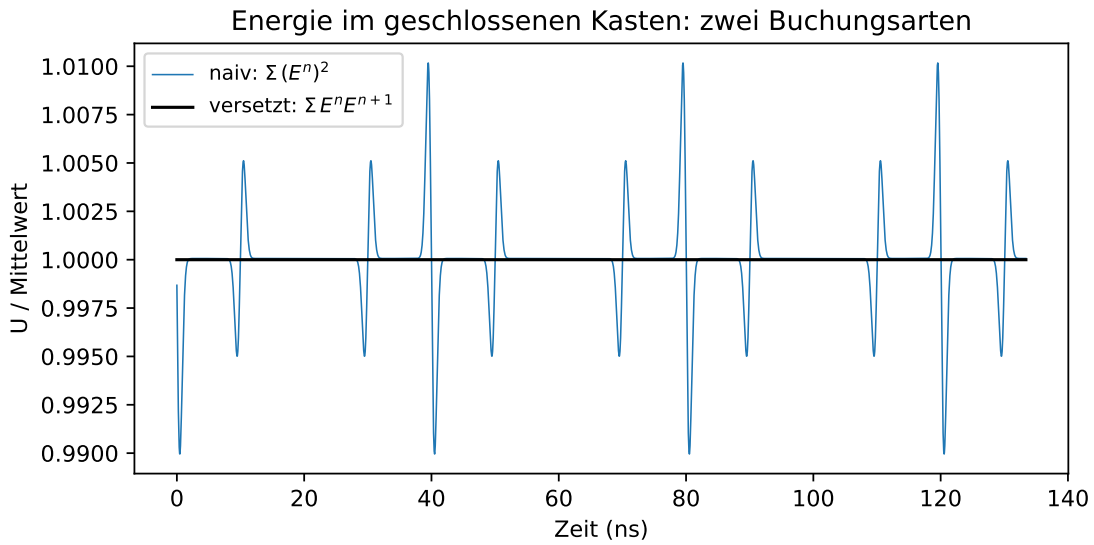


Abbildung 18.2: Gesamtenergie im geschlossenen, verlustfreien 1D-Kasten über 4000 Schritte. Die naive Summe (blau) zittert um etwa zwei Prozent — am stärksten immer dann, wenn der Puls gerade an einer Wand reflektiert. Die Yee-versetzte Energie (schwarz), bei der der E-Topf als Produkt aufeinanderfolgender Zeitschritte $E^{n \cdot E}(n+1)$ gebucht wird, ist konstant auf zwölf Stellen: Die diskrete Buchhaltung schließt exakt — man muss nur zeitversetzt buchen, wie das Gitter selbst.

```
naiv:     relative Schwankung 2.02e-02
```

```
versetzt: relative Schwankung 8.96e-13
```

Die naive Summe schwankt um zwei Prozent — kein Programmierfehler, sondern der **Yee-Versatz** (Kapitel 5), der hier zum dritten Mal zuschlägt: E^n und $B^{n+1/2}$ leben einen *halben Zeitschritt* auseinander, ihre Quadrate gehören gar nicht zum selben Moment. Das Produkt $E^n E^{n+1}$ dagegen sitzt zeitlich genau bei $n + \frac{1}{2}$ — beim B-Feld — und mit dieser Buchung ist U konstant auf **zwölf Stellen**, Maschinengenauigkeit. (Meep wertete aus demselben Grund die Quelle am Halbschritt aus, Kapitel 11; wer auf dem Yee-Gitter Energie bilanzieren will, muss yee-konform buchen.)

Jetzt die volle Bilanz mit allen drei Termen. **Frage:** Gilt $\Delta U/\Delta t$ = Fluss durch den Rand − Wärmeleistung Schritt für Schritt? **Bühne:** offenes Gitter (Mur-Ränder), der Puls startet links, läuft durch ein **Kontrollvolumen** $[5\,\mathrm{m}, 9\,\mathrm{m}]$, in dem von $x = 6$ bis 8 m eine Verlustzone liegt ($\sigma = 0{,}02$ S/m, das Kapitel-6-Rezept). **Gemessen wird** in jedem Schritt: $\Delta U/\Delta t$ im Volumen (versetzt gebucht), der Poynting-Fluss $S = EB/\mu_0$ an den beiden Rändern des Volumens (an den B-Halbpunkten, zeitlich gemittelt — yee-konform) und die Wärmeleistung $\sum \sigma\, E^n E^{n+1}\, \Delta x$. **Erfolgskriterium:** Der Rest (Summe aller drei) ist klein gegen jeden Einzelterm.

```python
# von oben: np, N, DX, DT, X, epsilon_0, mu_0, C
def kontrollbox(DX_, schritte, sig_wert):
    """Bilanz dU/dt = Fluss - P_Wärme über das Volumen [5 m, 9 m]."""
    N_ = int(12.0 / DX_) + 1
    DT_ = DX_ / C
    X_ = np.arange(N_) * DX_
    E = np.exp(-((X_ - 3.0) / 0.3)**2)
    B = np.zeros(N_ - 1)
    sigma = np.zeros(N_)
    sigma[(X_ > 6.0) & (X_ < 8.0)] = sig_wert
    s2 = sigma * DT_ / (2 * epsilon_0)
    ca = (1 - s2) / (1 + s2)                # Verlust-Update (Kapitel 6)
    cb = (C**2 * DT_ / DX_) / (1 + s2)
    A, Bx = int(5.0 / DX_), int(9.0 / DX_)
    reste, terme = [], []
    U_alt = None
    for n in range(schritte):
        B_alt = B.copy()
        B += (DT_ / DX_) * np.diff(E)
        E_alt = E.copy()
        E[1:-1] = ca[1:-1] * E[1:-1] + cb[1:-1] * np.diff(B)
        E[0], E[-1] = E_alt[1], E_alt[-2]
        U = (0.5 * epsilon_0 * np.sum(E_alt[A:Bx + 1] * E[A:Bx + 1]) * DX_
```

```
            + 0.5 / mu_0 * np.sum(B[A:Bx]**2) * DX_)
        fluss = (0.5 / mu_0) * (E_alt[Bx] * (B_alt[Bx] + B[Bx])
                                - E_alt[A] * (B_alt[A - 1] + B[A - 1]))
        p_ohm = np.sum(sigma[A:Bx + 1] * E_alt[A:Bx + 1]
                       * E[A:Bx + 1]) * DX_
        if U_alt is not None:
            reste.append(abs((U - U_alt) / DT_ - fluss + p_ohm))
            terme.append(max(abs((U - U_alt) / DT_), abs(fluss),
                             abs(p_ohm), 1e-30))
        U_alt = U
    return max(reste) / max(terme)

print(f"Vakuum-Volumen:  relativer Bilanzrest = "
      f"{kontrollbox(0.01, 3000, 0.0):.1e}")
print(f"mit σ-Zone:      relativer Bilanzrest = "
      f"{kontrollbox(0.01, 3000, 0.02):.1e}")
print(f"mit σ-Zone, Δx/2: relativer Bilanzrest = "
      f"{kontrollbox(0.005, 6000, 0.02):.1e}")
```

```
Vakuum-Volumen:  relativer Bilanzrest = 6.0e-13
mit σ-Zone:      relativer Bilanzrest = 9.1e-03
mit σ-Zone, Δx/2: relativer Bilanzrest = 4.6e-03
```

Im Vakuum schließt die Bilanz auf 10^{-13} — der diskrete Poynting-Satz ist auf dem Yee-Gitter keine Näherung, sondern eine **Identität**, wenn man Fluss und Energie konsistent versetzt bucht. Mit der σ-Zone bleibt ein Rest von einem knappen Prozent, und die Gegenprobe entlarvt ihn als Diskretisierung: Halbiertes Δx halbiert den Rest. Die Physik dahinter ist dieselbe wie immer — die Buchhaltung selbst hat keine Löcher.

Merkkasten: die Energie-Buchhaltung

- $u_E = \frac{\varepsilon_0}{2} E^2$, $u_B = \frac{1}{2\mu_0} B^2$; laufende Welle: beide Töpfe exakt gleich, stehende Welle: Pendel.
- $\partial_t u + \nabla \cdot \vec{S} = -\vec{J} \cdot \vec{E}$ mit $\vec{S} = \vec{E} \times \vec{H}$: Energie fließt ($\vec{S}$) oder wird verarbeitet ($\vec{J} \cdot \vec{E}$) — nichts verschwindet.
- $\vec{S}$ ist eine Stromdichte (W/m²); Leistung in Watt wird daraus durch Flächenintegrale — genau das tun Flussmonitore.

18.3 Die Landkarte: wo die Leistung wirklich läuft

Jetzt zum Lichtschalter. Wir bauen das einfachste Modell eines Stromkabels, das unsere 2D-Welt hergibt: zwei parallele Metallplatten („Hin- und Rückleiter") mit einem Spalt dazwischen, gespeist von einer Wechselquelle — eine **Parallelplatten-Leitung**. Vorher das Kästchen, denn hier wohnt eine der hartnäckigsten Fehlvorstellungen der Elektrotechnik:

> ⚠ Naheliegende Vermutung
>
> **„Die Energie fließt im Draht wie Wasser im Rohr — der Strom trägt sie, die Elektronen bringen sie mit."**
> *Warum sie naheliegt:* Wir sagen „Strom fließt", wir rechnen $P = U \cdot I$ mit Größen *des Drahtes*, und ohne Draht geht die Lampe nun mal nicht.
> *Was stattdessen stimmt:* Die Elektronen driften Millimeter pro Sekunde — als Energie-Boten unbrauchbar (die Rechnung folgt). Die Energie reist im **Feld zwischen und neben den Leitern**, mit nahezu Lichtgeschwindigkeit; der Draht führt das Feld, wie eine Schiene den Zug führt, ohne selbst zu fahren. Die $\vec{S}$-Karte, die wir gleich messen, zeigt es schwarz auf weiß: Im Spalt strömt die volle Leistung, im (idealen) Metall exakt nichts — und bei einem *realen* Leiter kippt nur ein kleiner Pfeil-Anteil seitlich ins Metall hinein: Das ist genau die Verlustleistung $\vec{J} \cdot \vec{E}$, die den Draht warm macht. Der Draht bekommt von der Energie nur das, was er *verheizt*.

Das Experiment. **Frage:** Wo läuft die Leistung — und kommt sie vollständig an? **Bühne:** eine 34×6-Zelle, zwei Platten der Dicke $0{,}4$ im Abstand $1{,}0$ (erst ideales Metall `mp.metal`, dann echtes), PML ringsum. Die Welle braucht ein E-Feld *quer* zwischen den Platten — das ist die H_z-Polarisation aus Kapitel 13 (für E_z wären die Platten ein Rohr unter Cutoff, Kapitel 17!); so geführt läuft zwischen den Platten eine kabeltypische **TEM-Welle** ohne Grenzfrequenz. **Anregung:** eine H_z-Quelllinie im Spalt bei $x = -12$, Dauerton $f = 0{,}5$. **Gemessen wird** das Periodenmittel von $\vec{S} = (E_y H_z, -E_x H_z)$ und u auf dem ganzen Gitter (zehn Schnappschüsse über eine Periode — das Kapitel-14-Rezept); außerdem, in einem zweiten Lauf mit schmalbandigem Puls, der Fluss durch fünf Querschnitte (Kapitel-12-Methode). **Erfolgskriterium:** P an allen Schnitten gleich; außerhalb der Platten nichts; $v_E = \langle S \rangle / \langle u \rangle = c$ (TEM-Welle im Vakuum-Spalt).

```
# von oben: np, plt
import os
os.environ["HWLOC_HIDE_ERRORS"] = "2"   # vor dem Meep-Import (Kapitel 11)
import meep as mp
```

```
mp.verbosity(0)                          # Fortschrittsmeldungen stumm

F0 = 0.5
SPALT, DICKE = 1.0, 0.4

def leitung_aufbau(material, quelltyp, aufloesung=24):
    geom = [mp.Block(size=mp.Vector3(mp.inf, DICKE, mp.inf),
                     center=mp.Vector3(0, s * (SPALT + DICKE) / 2),
                     material=material) for s in (+1, -1)]
    quelle = [mp.Source(quelltyp, component=mp.Hz,
                        center=mp.Vector3(-12, 0),
                        size=mp.Vector3(0, SPALT))]
    return mp.Simulation(cell_size=mp.Vector3(34, 6),
                         resolution=aufloesung, sources=quelle,
                         geometry=geom,
                         boundary_layers=[mp.PML(2.0)])

def periodenmittel(sim, dauer=120):
    """Einschwingen, dann ⟨Sx⟩, ⟨Sy⟩, ⟨u⟩ über eine Periode mitteln."""
    sim.run(until=dauer)
    sx = sy = u = None
    for k in range(10):
        sim.run(until=(1 / F0) / 10)
        ex = sim.get_array(component=mp.Ex).T
        ey = sim.get_array(component=mp.Ey).T
        hz = sim.get_array(component=mp.Hz).T
        sxk, syk = ey * hz, -ex * hz     # S = E × H in der Hz-Welt
        uk = 0.5 * (ex**2 + ey**2 + hz**2)
        sx = sxk if sx is None else sx + sxk
        sy = syk if sy is None else sy + syk
        u = uk if u is None else u + uk
    return sx / 10, sy / 10, u / 10

karten = {}
for name, material in (("PEC", mp.metal),
                       ("verlustbehaftet",
                        mp.Medium(D_conductivity=50.0))):
    sim = leitung_aufbau(material,
                         mp.ContinuousSource(F0, is_integrated=True))
    karten[name] = periodenmittel(sim)

ny, nx = karten["PEC"][0].shape
ys = np.linspace(-3, 3, ny)
```

```
xs = np.linspace(-17, 17, nx)

fig, achsen = plt.subplots(2, 1, figsize=(8, 5.2), sharex=True)
for achse, name in zip(achsen, ("PEC", "verlustbehaftet")):
    sx, sy, u = karten[name]
    betrag = np.hypot(sx, sy)
    achse.imshow(betrag, origin="lower", cmap="magma",
                 extent=[-17, 17, -3, 3], aspect="auto",
                 vmax=0.7 * betrag.max())
    # Pfeile nur, wo wirklich etwas fließt (sonst Punktrauschen),
    # und schwarz — auf dem hellen Spalt gut sichtbar
    schwach = betrag < 0.02 * betrag.max()
    sx_p = np.where(schwach, np.nan, sx)
    sy_p = np.where(schwach, np.nan, sy)
    s = 5                                # jeder 5. Punkt: Pfeilgitter
    achse.quiver(xs[::s], ys[::s], sx_p[::s, ::s], sy_p[::s, ::s],
                 color="k", scale=betrag.max() * 22, width=0.0035)
    for vz in (+1, -1):
        achse.axhline(vz * SPALT / 2, color="w", lw=0.6)
        achse.axhline(vz * (SPALT / 2 + DICKE), color="w", lw=0.6)
    achse.set(xlim=(-11.5, 2), ylim=(-1.45, 1.45), ylabel="y",
              title=f"Platten: {name}")
achsen[1].set_xlabel("x")
fig.tight_layout()
plt.show()

sx, sy, u = karten["PEC"]
spalt_m = np.abs(ys) < 0.5
zone = np.abs(xs) < 6
# np.ix_: spannt aus zwei 1D-Masken das Rechteck-Gitter auf, so dass
# arr[np.ix_(zeilen, spalten)] den Block "Zeilen x Spalten" ausschneidet
v_E = sx[np.ix_(spalt_m, zone)].mean() / u[np.ix_(spalt_m, zone)].mean()
schirm = (np.hypot(sx, sy)[np.abs(ys) > 1.0, :].max()
          / np.hypot(sx, sy)[np.ix_(spalt_m, zone)].max())
sx_v, sy_v, _ = karten["verlustbehaftet"]
j_rand = np.argmin(np.abs(ys - 0.48))
kipp = np.rad2deg(np.arctan2(sy_v[j_rand, zone].mean(),
                             sx_v[j_rand, zone].mean()))
print(f"PEC: v_E im Spalt = {v_E:.4f} c;  |S| außen/innen = {schirm:.0e}")
print(f"Verlustplatten: Kippwinkel an der Plattenkante = {kipp:+.1f}°")
```

```
Using MPI version 4.1, 1 processes
```

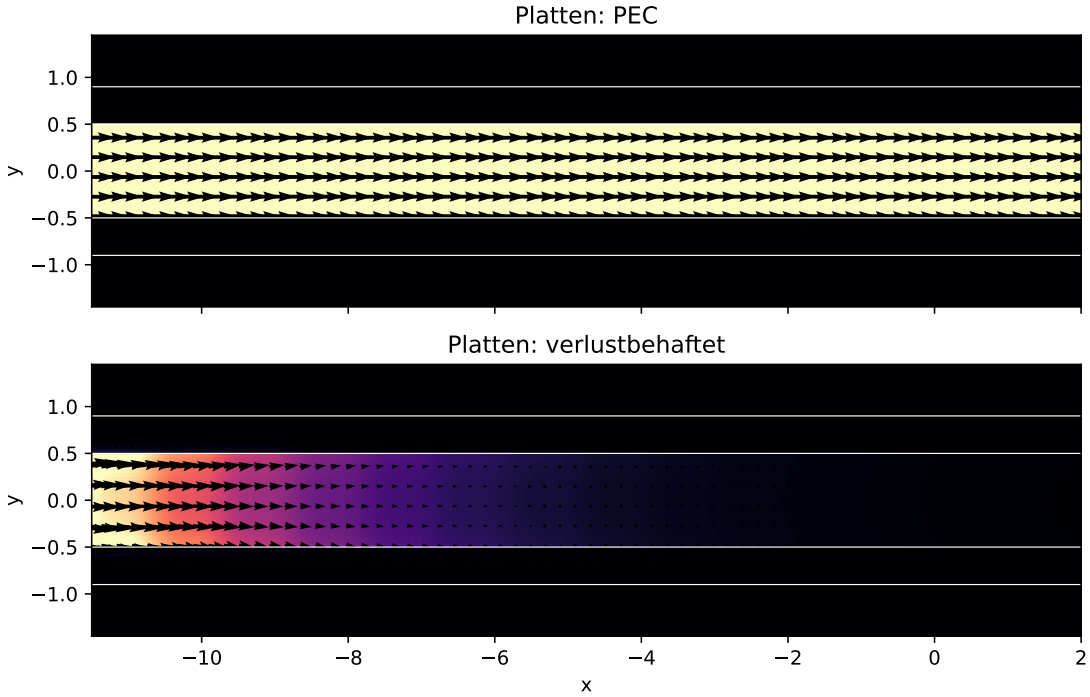


Abbildung 18.3: Die ⟨S⟩-Karte der Parallelplatten-Leitung (Lesehilfe: Farbe = Betrag der Energiestromdichte, Pfeile = ihre Richtung und Stärke; die weißen Linien sind die Plattenkanten, die Quelle sitzt links außerhalb des Ausschnitts). Oben ideales Metall: Die gesamte Leistung strömt im Spalt nach rechts — in den Platten und hinter ihnen ist exakt nichts (gemessen $< 10^{-100}$ des Spaltwerts). Unten Platten aus verlustbehaftetem Material: Die Strömung wird nach rechts hin schwächer (das Kabel „verliert“), und an den Plattenkanten kippen die Pfeile um etwa 9° INS Metall — dieser hineingekippte Anteil ist die Verlustleistung, die das Blech heizt.

```
PEC: v_E im Spalt = 0.9995 c;  |S| außen/innen = 7e-277
Verlustplatten: Kippwinkel an der Plattenkante = +9.1°
```

Die Karte spricht für sich: Im PEC-Fall läuft **alles** im Spalt (außen steht eine Null mit über hundert führenden Nullen — das ideale Kabel schirmt perfekt), und die Energie reist dort mit $v_E = c$. Der Draht trägt nichts; er *lenkt*. Die Puls-Gegenprobe mit fünf Flussmonitoren bestätigt die verlustfreie Lieferung:

```
# von oben: leitung_aufbau(), mp, np, F0, SPALT, DICKE
sim = leitung_aufbau(mp.metal, mp.GaussianSource(F0, fwidth=F0 / 8))
schnitte = [sim.add_flux(F0, F0 / 8, 1, mp.FluxRegion(
    center=mp.Vector3(x, 0), size=mp.Vector3(0, SPALT + 2 * DICKE),
    direction=mp.X)) for x in (-8, -4, 0, 4, 8)]
sim.run(until_after_sources=60)
P = np.array([mp.get_fluxes(s)[0] for s in schnitte])
print("P an den Schnitten x = -8, -4, 0, 4, 8 (ideale Platten):")
print(f"  {P.round(4)}   →  max. Abweichung "
      f"{np.abs(P / P[0] - 1).max():.1e}")
```

```
P an den Schnitten x = -8, -4, 0, 4, 8 (ideale Platten):
  [15.6292 15.6292 15.6292 15.6292 15.6292]   →  max. Abweichung 1.2e-06
```

Im unteren Kartenpanel — Platten aus schlechtem Metall — wird das Kabel ehrlich: Die Strömung verarmt nach rechts, und an den Plattenkanten kippen die Pfeile messbar (9°) **ins Blech**. Dort gilt jetzt das dritte Glied der Buchhaltung: Was hineinkippt, wird zu $\vec{J} \cdot \vec{E}$, zu Wärme. Das prüfen wir quantitativ nach — mit einer Eichung vorweg, ganz im Kapitel-15-Stil: Im *homogen* verlustigen Streifen (wo die Bilanz trivial stimmen muss) vergleichen wir die Flussmonitor-Leistung mit dem Volumenintegral $\sigma \sum |\hat{E}|^2/2$ aus den DFT-Feldern (Kapitel 15) und finden den Normierungsfaktor von Meeps DFT-Konvention:

```
# von oben: mp, np, F0, leitung_aufbau(), SPALT, DICKE
def eichlauf():
    """Homogen verlustiger Streifen: (P1−P2) / (σ·Σ|Ê|²/2) = ?"""
    sim = mp.Simulation(
        cell_size=mp.Vector3(48, 2), resolution=20,
        sources=[mp.Source(mp.GaussianSource(F0, fwidth=F0 / 8),
                           component=mp.Hz, center=mp.Vector3(-18, 0),
                           size=mp.Vector3(0, 2))],
        default_material=mp.Medium(D_conductivity=0.1),
        boundary_layers=[mp.PML(3.0, direction=mp.X)],
        k_point=mp.Vector3())
    quer = [sim.add_flux(F0, F0 / 8, 1, mp.FluxRegion(
        center=mp.Vector3(x, 0), size=mp.Vector3(0, 2),
        direction=mp.X)) for x in (-5.0, 5.0)]
    dft = sim.add_dft_fields([mp.Ex, mp.Ey], F0, 0, 1,
                             center=mp.Vector3(0, 0),
                             size=mp.Vector3(10, 2))
    sim.run(until_after_sources=120)
```

```
    P1, P2 = (mp.get_fluxes(q)[0] for q in quer)
    e2 = sum(np.abs(sim.get_dft_array(dft, f, 0))**2
             for f in (mp.Ex, mp.Ey))
    nx_, ny_ = e2.shape
    return (P1 - P2) / (0.1 * 0.5 * e2.sum() * (10 / nx_) * (2 / ny_))

faktor = eichlauf()
print(f"Eichfaktor = {faktor:.4f}  →  Joule-Integral = σ·Σ|Ê|² "
      f"(Meeps Flüsse tragen den Faktor ½ nicht)")

def leitungsbilanz(aufloesung, sd=20.0):
    """P(x₁) − P(x₂) gegen ∫J·E in den Platten dazwischen."""
    sim = leitung_aufbau(mp.Medium(D_conductivity=sd),
                         mp.GaussianSource(F0, fwidth=F0 / 8),
                         aufloesung)
    X1, X2 = -10.0, -6.0
    quer = [sim.add_flux(F0, F0 / 8, 1, mp.FluxRegion(
        center=mp.Vector3(x, 0), size=mp.Vector3(0, 5.5),
        direction=mp.X)) for x in (X1, X2)]
    H = SPALT + 2 * DICKE
    dft = sim.add_dft_fields([mp.Ex, mp.Ey], F0, 0, 1,
                             center=mp.Vector3((X1 + X2) / 2, 0),
                             size=mp.Vector3(X2 - X1, H))
    sim.run(until_after_sources=80)
    P1, P2 = (mp.get_fluxes(q)[0] for q in quer)
    e2 = sum(np.abs(sim.get_dft_array(dft, f, 0))**2
             for f in (mp.Ex, mp.Ey))
    nx_, ny_ = e2.shape
    ys_ = np.linspace(-H / 2, H / 2, ny_)
    platte = np.abs(ys_) > SPALT / 2
    joule = sd * e2[:, platte].sum() * ((X2 - X1) / nx_) * (H / ny_)
    return P1 - P2, joule

for aufl in (24, 48):
    diff, joule = leitungsbilanz(aufl)
    print(f"Auflösung {aufl}: P(x₁)−P(x₂) = {diff:.3f},  "
          f"∫J·E in den Platten = {joule:.3f}  "
          f"(Fehler {abs(diff - joule) / diff * 100:.0f} %)")
```

```
Eichfaktor = 2.0092  →  Joule-Integral = σ·Σ|Ê|² (Meeps Flüsse tragen den Faktor ½ nicht)
Auflösung 24: P(x₁)−P(x₂) = 3.676,  ∫J·E in den Platten = 2.804  (Fehler 24 %)
Auflösung 48: P(x₁)−P(x₂) = 3.452,  ∫J·E in den Platten = 3.291  (Fehler 5 %)
```

Bei Auflösung 48 schließt auch diese Bilanz auf wenige Prozent — die fehlende Leistung der Leitung steckt nachweislich als Wärme im Blech. Dass die Auflösung 24 noch 24 % danebenliegt, ist selbst eine Lektion: Die Verluste spielen sich in einer dünnen **Skin-Schicht** unter der Metalloberfläche ab (hier ~0,2 Einheiten tief), und wer sie bilanzieren will, muss sie auflösen. Unser „schlechtes Metall" ($\sigma_D = 20$ statt Kupfer-Astronomie) ist übrigens Absicht: Bei echtem Kupfer wäre die Skin-Schicht so dünn und der Verlust so winzig, dass im kleinen Laborgebiet schlicht nichts zu sehen wäre — die Physik ist dieselbe, nur der Maßstab nicht.

Bleibt die Drift-Rechnung, die das Kästchen versprochen hat. In Kupfer stehen $n \approx 8{,}5 \cdot 10^{28}$ Leitungselektronen pro Kubikmeter bereit. Der Strom ist $I = n\,q\,A\,v_d$ — Ladungsdichte mal Querschnitt mal Tempo. Auflösen nach v_d für $I = 10$ A durch $A = 1{,}5$ mm²:

$$v_d = \frac{I}{n\,q\,A} = \frac{10}{8{,}5 \cdot 10^{28} \cdot 1{,}6 \cdot 10^{-19} \cdot 1{,}5 \cdot 10^{-6}} \approx 0{,}5\ \frac{\text{mm}}{\text{s}}.$$

Ein halber Millimeter pro Sekunde — und bei 50-Hz-Wechselstrom pendeln die Elektronen damit nur etwa zwei Mikrometer hin und her, ein Vierzigstel Haardurchmesser (Ü 18.2 rechnet beides nach). Die Lampe geht trotzdem „sofort" an, weil das **Feld** im Spalt zwischen den Leitern mit nahezu c anrückt — unsere Karte hat es gemessen — und *überall im Stromkreis gleichzeitig* die dort wartenden Elektronen anschiebt. Die Elektronen sind nicht die Boten der Energie; sie sind das Medium, an dem das Feld vor Ort angreift. Dasselbe Bild gilt vom Koaxialkabel (die Leistung fließt im Isolator zwischen Innen- und Außenleiter — im Querschnitt ein leuchtender Ring, im Kupfer null) bis zur Hochspannungstrasse: Der Energiestrom eines Stromnetzes läuft durch die *Luft zwischen den Seilen*. Der Unterschied zur Glasfaser (Kapitel 17) ist nur der Führungsmechanismus — Randbedingung am Metall hier, Totalreflexion dort; die Energie wohnt in beiden Fällen im Feld.

18.4 Wie schnell reist Energie?

Kapitel 16 und 17 haben behauptet, Energie reise mit v_g — gemessen wurde aber stets das Tempo von *Paketen* (Energie-Schwerpunkten). Mit $\vec{S}$ und u lässt sich der Satz endlich direkt prüfen, denn der Quotient

$$v_E = \frac{\langle S \rangle}{\langle u \rangle}$$

ist wörtlich „transportierte Leistung pro vorrätiger Energie" — eine Geschwindigkeit, die keinerlei Paket braucht; ein Dauerton genügt. In der TEM-Leitung eben kam $v_E = 0{,}999\,c$ heraus. Die spannende Bühne ist aber das **Rohr aus Kapitel 17** mit seinen zwei Tempi ($v_g = 0{,}553\,c$, $v_p = 1{,}81\,c$ bei $f = 1{,}2\,f_c$): Welchem folgt die Energie? **Bühne:** dieselbe 120×1-Zelle wie dort (Zellrand = Rohrwand), Dauerton $f = 0{,}6$, eingeschwungen; **gemessen wird** das Periodenmittel von $S_x = -E_z H_y$ und u über das mittlere Rohrdrittel. **Erfolgskriterium:** v_E trifft eine der beiden Kapitel-17-Zahlen — die Theorie sagt, welche:

```
# von oben: mp, np
sim = mp.Simulation(
    cell_size=mp.Vector3(120, 1.0), resolution=20,
    sources=[mp.Source(mp.ContinuousSource(0.6, is_integrated=True),
                       component=mp.Ez, center=mp.Vector3(-45, 0),
                       size=mp.Vector3(0, 1.0))],
    boundary_layers=[mp.PML(3.0, direction=mp.X)])
sim.run(until=400)
sx = u = None
for k in range(10):
    sim.run(until=(1 / 0.6) / 10)
    ez = sim.get_array(component=mp.Ez).T
    hx = sim.get_array(component=mp.Hx).T
    hy = sim.get_array(component=mp.Hy).T
    sxk = -ez * hy                        # S_x = −E_z·H_y (Ez-Welt)
    uk = 0.5 * (ez**2 + hx**2 + hy**2)
    sx = sxk if sx is None else sx + sxk
    u = uk if u is None else u + uk
xs_r = np.linspace(-60, 60, sx.shape[1])
zone_r = np.abs(xs_r) < 25
v_E_rohr = sx[:, zone_r].mean() / u[:, zone_r].mean()
vg = np.sqrt(1 - (0.5 / 0.6)**2)
print(f"v_E = {v_E_rohr:.4f} c   −   v_g = {vg:.4f} c,  "
      f"v_p = {1/vg:.4f} c")
```

```
v_E = 0.5518 c   −   v_g = 0.5528 c,  v_p = 1.8091 c
```

$v_E = v_g$, auf zwei Promille — und meilenweit von v_p entfernt. Die Behauptung aus zwei Kapiteln ist damit keine Paket-Anekdote mehr, sondern eine Eigenschaft des *eingeschwungenen Feldes selbst*: Auch im Dauerbetrieb, ganz ohne Einhüllende, transportiert das Rohr seine Energie mit der Gruppengeschwindigkeit. (Anschaulich: Die Welle läuft im Zickzack zwischen den Wänden; u ist überall, aber nur die Längskomponente von $\vec{S}$ kommt voran.)

Wo $v_p > c$ war, ist v_E brav unter c geblieben — die Energie hält sich an die Regeln, die Berge durften nur so tun.

18.5 Das Kapitel-Programm

Das Programm `kap18_poynting.py` (im Buchpaket unter `programme/kap18/`) bündelt alles: Töpfe, naive vs. versetzte Buchhaltung, Kontrollvolumen-Bilanz samt Konvergenz, Leitungs-Karten und -Flüsse, Joule-Bilanz mit Eichung, v_E im Rohr und die Ringwellen-Bilanz aus Ü 18.3. Jede Behauptung steht als `assert`; der Lauf braucht etwa eine Minute und endet mit „Alle Tests bestanden."

Merkkasten

- **Die Energie reist im Feld, nicht im Draht.** Der Leiter führt; was er von der Leistung abbekommt, ist genau sein Verlust ($\vec{J} \cdot \vec{E}$ — die Pfeile, die ins Metall kippen).
- Elektronen **driften** (~mm/s) — die Lampe geht an, weil das Feld mit $\sim c$ kommt und überall zugleich anschiebt.
- Auf dem Yee-Gitter ist der Poynting-Satz eine **exakte Identität** — wenn man zeitversetzt bucht ($E^n E^{n+1}$).
- $v_E = \langle S \rangle / \langle u \rangle = v_g$ — gemessen in Leitung ($= c$) und Rohr ($= 0{,}55\,c$). Energie folgt der Gruppe, nie den Bergen.

Roter Faden

Rückwärts: Das **Kreuzprodukt aus Kapitel 1** ist mit $\vec{S} = \vec{E} \times \vec{H}$ endgültig Physik geworden, die **Lenz-Bilanz aus Kapitel 3** eine Identität auf dreizehn Stellen, die **Kapitel-4-Zahlen** (868 V/m Sonnenlicht) haben ihre Energierechnung bekommen, das **σ aus Kapitel 6** seine Wärmeleistung, der **Yee-Versatz aus Kapitel 5** seinen dritten Auftritt (diesmal in der Energie selbst), und die **Rohre/Streifen aus Kapitel 16/17** lieferten die Bühnen für $v_E = v_g$. Vorwärts: **Kapitel 19** lässt die Energie im Schwingkreis pendeln (das stehende-Welle-Pendel als Bauteil), **Kapitel 20** fragt nach der Richtung der Zeit in dieser Buchhaltung, **Kapitel 21** misst, wie eine Antenne ihre Leistung ins Fernfeld drückt (mit genau diesen Flussmonitoren), **Kapitel 22** bringt $P = U \cdot I$ und die Feldsicht zusammen (Leitungen, 50 Ω), und **Kapitel 25** beantwortet mit $\vec{S}$ die Steckdosenfrage, warum 50 Hz nicht funkt.

Übungen

Ü 18.1 (Verstehen). Sonnenlicht, komplett durchgerechnet: Aus $I = 1000$ W/m² folgen E_0, B_0, $\langle u_E \rangle$ und $\langle u_B \rangle$ (Zeitmittel!). Zeige außerdem: Der **Strahlungsdruck** auf eine absorbierende Fläche ist $p = I/c$ (Tipp aus dem Kleingedruckten: $\vec{S}/c^2$ ist Impulsdichte — Impuls pro Zeit und Fläche ist Druck). Wie viel ist das in Pascal — und warum spürt man die Sonne als Wärme, aber nicht als Schub?

Musterlösung zu Ü 18.1

```
# von oben: np, epsilon_0, mu_0, C
I = 1000.0
E0 = np.sqrt(2 * mu_0 * C * I)           # aus I = E0²/(2 μ0 c)
B0 = E0 / C
print(f"E0 = {E0:.0f} V/m,  B0 = {B0*1e6:.2f} μT")
print(f"<u_E> = {epsilon_0 * E0**2 / 4 * 1e6:.2f} μJ/m³,  "
      f"<u_B> = {B0**2 / (4 * mu_0) * 1e6:.2f} μJ/m³")
print(f"Strahlungsdruck p = I/c = {I / C * 1e6:.1f} μPa")
```

```
E0 = 868 V/m,  B0 = 2.90 μT
<u_E> = 1.67 μJ/m³,  <u_B> = 1.67 μJ/m³
Strahlungsdruck p = I/c = 3.3 μPa
```

Die Zeitmittel tragen den Faktor $\frac{1}{2}$ aus $\langle \cos^2 \rangle$ (Kapitel 10): $\langle u_E \rangle = \varepsilon_0 E_0^2/4 = 1{,}7$ µJ/m³, exakt gleich $\langle u_B \rangle$. Der Strahlungsdruck $p = I/c \approx 3{,}3$ µPa ist dreißig Milliarden Mal kleiner als der Luftdruck — die Energie kommt mit voller Kraft (1 kW auf den Quadratmeter Haut!), der Impuls aber durch c geteilt: Licht trägt viel Energie pro Schwung. Messbar ist der Schub trotzdem, wo nichts stört und Fläche riesig wird: Kometenschweife zeigen von der Sonne weg, und Sonnensegel (IKAROS, 2010) fliegen tatsächlich damit.

Ü 18.2 (Verstehen). Rechne die Driftgeschwindigkeit für deine eigene Zimmerbeleuchtung nach (sagen wir 60 W an 230 V durch 1,5-mm²-Kupfer) und die Zitterweite bei 50 Hz. Formuliere dann in zwei Sätzen, warum die Lampe „sofort" angeht.

Musterlösung zu Ü 18.2

```
# von oben: np
n_e, q, A = 8.5e28, 1.6e-19, 1.5e-6
I = 60 / 230                                # 0,26 A
v_d = I / (n_e * q * A)
print(f"I = {I:.2f} A  →  v_d = {v_d*1e6:.1f} µm/s")
omega = 2 * np.pi * 50
print(f"Zitterweite bei 50 Hz: ±{v_d*np.sqrt(2)/omega*1e9:.0f} nm")
```

```
I = 0.26 A  →  v_d = 12.8 µm/s
Zitterweite bei 50 Hz: ±58 nm
```

Knapp 13 µm/s Drift — und im Wechselstrom pendeln die Elektronen um ganze **±60 Nanometer**, weniger als eine Lichtwellenlänge. Warum die Lampe sofort angeht: Beim Schalterschluss breitet sich das *Feld* zwischen den Leitern mit nahezu Lichtgeschwindigkeit aus und schiebt **überall im Stromkreis gleichzeitig** die dort bereits wartenden Elektronen an. Die Energie kommt aus dem Feld neben dem Draht — die Elektronen vor Ort müssen sie nur noch in der Lampe abliefern.

Ü 18.3 (Verändern). Die Ringwelle aus Kapitel 9, jetzt als Leistungsbilanz: Miss in einem 2D-Lauf mit Dauerton-Punktquelle das Periodenmittel von $\vec{S}$ und bilde das Kreisintegral $P(r) = \oint \langle S_r \rangle \, \mathrm{d}s$ für mehrere Radien. Was muss herauskommen — und was heißt das für die $1/\sqrt{r}$-Amplitude von damals?

Musterlösung zu Ü 18.3

```
# von oben: mp, np
sim = mp.Simulation(cell_size=mp.Vector3(20, 20), resolution=20,
                    sources=[mp.Source(mp.ContinuousSource(1.0),
                                       component=mp.Ez,
                                       center=mp.Vector3())],
                    boundary_layers=[mp.PML(1.5)])
sim.run(until=60)
sx = sy = None
for k in range(10):
    sim.run(until=0.1)
    ez = sim.get_array(component=mp.Ez).T
    hx = sim.get_array(component=mp.Hx).T
    hy = sim.get_array(component=mp.Hy).T
    sxk, syk = -ez * hy, ez * hx
    sx = sxk if sx is None else sx + sxk
    sy = syk if sy is None else sy + syk
sx, sy = sx / 10, sy / 10
xs = np.linspace(-10, 10, sx.shape[1])
ys = np.linspace(-10, 10, sx.shape[0])
for r in (3.0, 5.0, 7.0):
    phis = np.linspace(0, 2 * np.pi, 361)[:-1]
    s_r = []
    for ph in phis:
        i = np.argmin(np.abs(xs - r * np.cos(ph)))
        j = np.argmin(np.abs(ys - r * np.sin(ph)))
        s_r.append(sx[j, i] * np.cos(ph) + sy[j, i] * np.sin(ph))
    print(f"r = {r}:  P = 2πr·⟨S_r⟩ = {np.mean(s_r)*2*np.pi*r:.4f}")
```

```
r = 3.0:  P = 2πr·⟨S_r⟩ = 0.7143
r = 5.0:  P = 2πr·⟨S_r⟩ = 0.7180
r = 7.0:  P = 2πr·⟨S_r⟩ = 0.7158
```

$P(r)$ ist konstant (hier auf ein halbes Prozent): Was die Quelle abgibt, muss durch jeden Kreis vollständig hindurch — Energieerhaltung als Messwert. Genau daraus folgt rückwärts die Amplitudenregel aus Kapitel 9: $\langle S_r \rangle \propto E^2$ muss wie $1/(2\pi r)$ fallen, also $E \propto 1/\sqrt{r}$. Die „geometrische Verdünnung" von damals war nie etwas anderes als diese Bilanz.

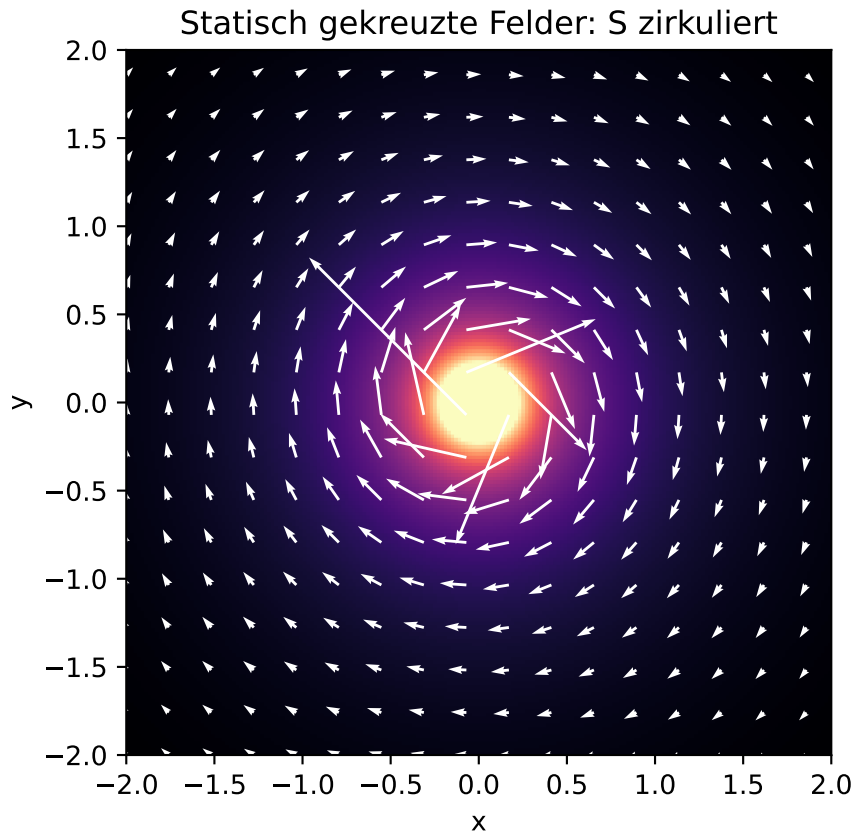
Ü 18.4 (Übertragen). Die statische Kuriosität aus dem Kleingedruckten, selbst kartiert: Lege das 2D-Feld einer Punktladung ($\vec{E}$ radial, $\propto 1/r$) über

ein homogenes H_z-Feld und berechne $\vec{S} = (E_y H_z, -E_x H_z)$ als NumPy-Karte. Was zeigt die Karte — und was sagt $\nabla \cdot \vec{S}$ (mit `np.gradient`, Kapitel 2) dazu?

Musterlösung zu Ü 18.4

```
# von oben: np, plt
xg, yg = np.meshgrid(np.linspace(-2, 2, 200),
                     np.linspace(-2, 2, 200))
r2 = xg**2 + yg**2 + 0.05**2
Ex, Ey = xg / r2, yg / r2                # Punktladung (2D): E ∝ r̂/r
Hz = 1.0                                 # homogenes Magnetfeld
Sx, Sy = Ey * Hz, -Ex * Hz
div_S = (np.gradient(Sx, axis=1) / (xg[0, 1] - xg[0, 0])
         + np.gradient(Sy, axis=0) / (yg[1, 0] - yg[0, 0]))

fig, achse = plt.subplots(figsize=(5.5, 4.6))
betrag = np.hypot(Sx, Sy)
# np.percentile(a, 99): der Wert, unter dem 99 % der Daten liegen —
# robuste Farbskala, die der 1/r²-Spitze an der Ladung nicht aufsitzt
achse.imshow(betrag, origin="lower", extent=[-2, 2, -2, 2],
             cmap="magma", vmax=np.percentile(betrag, 99))
s = 12
achse.quiver(xg[::s, ::s], yg[::s, ::s], Sx[::s, ::s], Sy[::s, ::s],
             color="w", width=0.004)
achse.set(xlabel="x", ylabel="y",
          title="Statisch gekreuzte Felder: S zirkuliert")
fig.tight_layout()
plt.show()
print(f"max |∇·S| / max |S|/Δr = "
      f"{np.abs(div_S[5:-5, 5:-5]).max() / (betrag.max() / 0.02):.1e}")
```




```
max |∇·S| / max |S|/Δr = 1.1e-02
```

Die Pfeile (Lesehilfe: Farbe = $|\vec{S}|$, Pfeile = Richtung) laufen im **Kreis um die Ladung** — obwohl beide Felder vollkommen statisch sind, behauptet $\vec{E} \times \vec{H}$ einen Energiestrom. Der Widerspruch löst sich über die Buchhaltung: $\nabla \cdot \vec{S} = 0$ — numerisch bleibt etwa ein Prozent der natürlichen Vergleichsskala übrig, der übliche Diskretisierungsrest zentraler Differenzen am $1/r$-Feld (Kapitel 2), keine echte Quelle. Die Strömung hat keine Quellen und keine Senken, nirgends sammelt sich Energie an, nirgends fehlt welche. Nur **Bilanzen** von $\vec{S}$ sind Physik; die Stromlinien selbst darf man nicht naiver lesen, als der Poynting-Satz es deckt. (Und doch nimmt die Natur diese Zirkulation ernst: Sie trägt einen messbaren Drehimpuls — Feynmans rotierende Scheibe.)

Das Kleingedruckte

Vier Randnotizen. **Erstens:** Der Poynting-Satz legt $\vec{S}$ nur über seine Divergenz fest — punktweise darf man zu $\vec{E} \times \vec{H}$ beliebige quellenfreie Wirbelfelder addieren, ohne dass sich eine Bilanz ändert. $\vec{E} \times \vec{H}$ ist die Standard-Wahl, und sie ist die physikalisch richtige: $\vec{S}/c^2$ ist zugleich die **Impulsdichte** des Feldes (daher der Strahlungsdruck aus Ü 18.1), und nur diese Wahl ist mit der Relativitätstheorie konsistent. **Zweitens:** In dispersiven Medien (Kapitel 16) ist die Energiedichte *nicht* einfach $\varepsilon E^2/2$ — ein Teil der Energie steckt in den mitschwingenden Elektronen, und die korrekte Formel (nach Brillouin) enthält $\partial(\omega\varepsilon)/\partial\omega$. Erst damit gilt auch dort $v_E = v_g$; unsere Messungen haben das Problem umgangen, indem Leitung und Rohr im Vakuum geführt wurden. **Drittens:** Meeps DFT-Konvention war eine kleine Falle: Flussmonitore liefern $\int \mathrm{Re}\,(\hat{E} \times \hat{H}^*)$ *ohne* den Faktor $\frac{1}{2}$ — wer DFT-Feldbeträge gegen Flüsse bilanziert, eicht das Verhältnis besser einmal am homogenen Fall, wie wir es getan haben (Faktor 2,01 statt 1). **Viertens:** Beim echten Kabel bei 50 Hz ist die Skin-Schicht im Kupfer etwa 9 mm dick, der Energiestrom im Draht selbst trotzdem winzig gegen den im Feld — und die berühmte Frage, „wie schnell die Lampe wirklich angeht", hat 2021 als *Veritasium*-Video für Furore gesorgt: Die saubere Antwort ist genau unsere Karte ($\vec{S}$ kommt teils auf direktem Luftweg von der Batterie zur Lampe, der volle eingeschwungene Strom folgt mit der Leitungslaufzeit — Kapitel 22).

19 Oszillator und Energie-Umpacken

Auf der Anzeige eines Radios steht „UKW 100,6 MHz". Aus der Steckdose, die das Radio speist, kommen 50 Hz. Zwischen den beiden Zahlen liegt ein Faktor von zwei Millionen — und irgendwo im Gerät muss er entstehen. Wie macht man aus dem trägen Wechselstrom des Kraftwerks die flinken hundert Millionen Schwingungen pro Sekunde, die eine Antenne braucht? Die naheliegende Antwort („man transformiert die Frequenz hoch, so wie ein Trafo die Spannung hochtransformiert") ist falsch, und zwar auf eine lehrreiche Art: Sie verwechselt die *Energie* mit dem *Muster*, das man ihr aufprägt. Dieses Kapitel sortiert die Begriffe, baut das schwingfähigste aller Schaltbilder — den LC-Schwingkreis — und zeigt, dass seine Frequenz aus genau dem Vorgang folgt, den Kapitel 18 vorbereitet hat: aus dem **Umpacken von Energie** zwischen einem elektrischen und einem magnetischen Topf.

Kapitel 18 hat es wörtlich angekündigt: Die stehende Welle im geschlossenen Kasten schaufelt ihre Energie zweimal pro Periode vollständig vom E-Topf in den B-Topf und zurück — „Kapitel 19 baut auf genau diesem Bild den Schwingkreis." Heute lösen wir das ein. Unterwegs treffen wir alte Bekannte in neuer Rolle: den Leapfrog aus Kapitel 5 (diesmal mit nur *zwei* Zahlen statt zwei Feldern), die Stabilitätsgrenze aus Kapitel 8 (diesmal in null Raumdimensionen), die ca/cb-Verlustkoeffizienten aus Kapitel 6, die Güte Q aus Kapitel 15 — und am Ende den Hohlraumresonator selbst, der sich als Schwingkreis ohne Bauteile entpuppt.

Lernziele

Nach diesem Kapitel kannst du …

1. … Energie, Leistung, Strom, Spannung, Frequenz und Signal sauber auseinanderhalten — und erklären, warum „Strom" nicht die Energie ist und Energie keine Frequenz hat,
2. … die Schwingfrequenz $\omega_0 = 1/\sqrt{LC}$ aus der Energieerhaltung herleiten (nicht auswendig lernen),
3. … den Schwingkreis als kleinstes Yee-Gitter simulieren: Leapfrog mit versetzter Energie-Buchhaltung auf Maschinengenauigkeit, Eulers exponentielles Aufblähen, Stabilitätsgrenze $\omega_0 \Delta t = 2$,

4. … die drei Schicksale einer Schwingung unterscheiden (ideal, gedämpft, angetrieben), $Q = \omega_0 L/R$ messen und erklären, warum ein echter Oszillator eine **Amplitudenregelung** braucht,
5. … einen Hohlraumresonator als verteilten Schwingkreis lesen: Die Energie pendelt mit $2f_0$ zwischen E- und B-Topf.

19.1 Erst die Begriffe: Was ist was?

Viele Knoten in diesem Themenkreis lösen sich, sobald die Begriffe sauber getrennt sind. Die Alltagssprache hilft dabei nicht — wir „verbrauchen Strom" und zahlen die „Stromrechnung", obwohl kein einziges Elektron im Kraftwerk bleibt. Ein Wasserwerk-Bild sortiert die Rollen:

Tabelle 19.1: Sechs Begriffe, die die Alltagssprache verrührt.

Begriff	Was es physikalisch ist	Wasserwerk-Bild
Energie	die „Ware", die gespeichert und transportiert wird (Joule)	das Wasser
Leistung	die *Lieferrate* der Energie (Watt = Joule pro Sekunde)	Liter pro Sekunde
Strom	bewegte Ladung — ein *Träger*, nicht die Energie selbst (Ampere)	die Strömung im Rohr
Spannung	der „Druck", der die Ladungen treibt (Volt)	der Wasserdruck
Frequenz	Eigenschaft des *zeitlichen Musters* — nicht der Energie (Hertz)	wie oft die Pumpe pro Sekunde umschaltet
Signal	Information, die dem Energiefluss aufgeprägt ist	Morsezeichen mit dem Wasserhahn

Zwei Zeilen dieser Tabelle hat Kapitel 18 bereits mit Messungen unterfüttert: Die Energie reist im *Feld* (nicht im Draht), und der Strom ist nur der Träger, an dem das Feld angreift — die Elektronen selbst kriechen mit Mikrometern pro Sekunde. Neu und zentral für dieses Kapitel ist die **Frequenz-Zeile**: Gleichstrom ist ein vollkommen gleichmäßiger Energiefluss *ohne* Frequenz, und trotzdem kann man aus ihm jede beliebige Schwingung gewinnen. Die Frequenz steckt nicht in der Energie — sie entsteht erst, wenn man die Energie in etwas **Schwingfähiges** füllt, und sie ist eine Eigenschaft *dieser Struktur*. Was so ein Schwingfähiges ausmacht, ist die nächste Frage.

19.2 Zwei Töpfe werden Bauteile

Die stehende Welle aus Kapitel 18 hatte alles, was eine Schwingung braucht: zwei Energietöpfe (E-Feld und B-Feld) und einen Mechanismus, der den Inhalt verlustfrei hin- und herschaufelt (die Maxwell-Gleichungen). Die Elektrotechnik verpackt diese beiden Töpfe seit 150 Jahren in zwei Bauteile:

- Der **Kondensator** C ist ein eingesperrtes E-Feld: zwei Platten, dazwischen das Feld der Ladung q, die auf ihnen sitzt. Seine Spannung ist $u_C = q/C$ (das *definiert* die Kapazität: viel Ladung pro Volt = großes C), seine gespeicherte Energie

  $$E_C = \frac{q^2}{2C}.$$

  Das ist wörtlich die Energiedichte $\frac{\varepsilon_0}{2}E^2$ aus Kapitel 18, aufintegriert über das Plattenvolumen.
- Die **Spule** L ist ein eingesperrtes B-Feld: eine Drahtwicklung, durch die der Strom i sein Magnetfeld aufbaut. Ändert sich der Strom, wehrt sich die Spule mit der Induktionsspannung $u_L = L\,\mathrm{d}i/\mathrm{d}t$ — das ist das Induktionsgesetz aus Kapitel 3, eingedampft auf ein Bauteil (die Lenz-Regel steckt im Vorzeichen: die Spule stemmt sich gegen jede Änderung). Ihre Energie ist

  $$E_L = \frac{L\,i^2}{2},$$

  das aufintegrierte $\frac{1}{2\mu_0}B^2$.

Jetzt verbinden wir beide zu einem Ring: Kondensator an Spule, sonst nichts. Auf dem Kondensator sitze die Ladung q, durch die Spule fließe der Strom $i = \mathrm{d}q/\mathrm{d}t$ (was vom Kondensator abfließt, fließt durch die Spule). Statt wie üblich die Maschenregel zu bemühen, nehmen wir den Weg, der zum Kapitel gehört: **die Energieerhaltung selbst als Bewegungsgesetz.** Im verlustfreien Ring muss die Gesamtenergie konstant sein:

$$\frac{\mathrm{d}}{\mathrm{d}t}\left[\frac{q^2}{2C} + \frac{L\,i^2}{2}\right] = 0 \quad\Longrightarrow\quad \frac{q}{C}\,i + L\,i\,\frac{\mathrm{d}i}{\mathrm{d}t} = 0.$$

Den gemeinsamen Faktor i kürzen (solange Strom fließt), und übrig bleibt

$$L\,\frac{\mathrm{d}i}{\mathrm{d}t} = -\frac{q}{C} \qquad \text{mit} \qquad \frac{\mathrm{d}q}{\mathrm{d}t} = i.$$

Das ist Wort für Wort die Feder-Masse-Gleichung der Mechanik: $m\,\dot{v} = -k\,x$ mit $\dot{x} = v$. Die Spule spielt die träge Masse (sie wehrt sich gegen Stromänderung), der Kondensator die gespannte Feder (er drückt die Ladung zurück). Und wie bei der Feder folgt die Schwingung $q(t) = q_0 \cos(\omega_0 t)$ mit

$$\boxed{\omega_0 = \frac{1}{\sqrt{LC}}, \qquad f_0 = \frac{1}{2\pi\sqrt{LC}}}$$

— nachrechnen: $\ddot{q} = -q/(LC)$, und $\cos(\omega_0 t)$ erfüllt das genau für $\omega_0^2 = 1/LC$. **Die Frequenz folgt allein aus den beiden Topfgrößen** L und C — nicht aus der Energiemenge (die setzt nur die Amplitude) und schon gar nicht aus der Frequenz irgendeiner Quelle.

Wie groß sind solche Bauteile im Radio? Für UKW bei $f_0 = 100$ MHz wähle eine handelsübliche kleine Spule mit $L = 0{,}1$ µH. Dann muss

$$C = \frac{1}{(2\pi f_0)^2\,L} = \frac{1}{(6{,}28\cdot 10^8)^2\cdot 10^{-7}\,\mathrm{H}} \approx 25\ \mathrm{pF}$$

sein — 25 Picofarad, ein winziger Keramikkondensator. Beides sind Centartikel: Die hundert Millionen Schwingungen pro Sekunde kosten keine Hightech, nur die richtige Paarung zweier Töpfe. (Wer am Drehkondensator eines alten Radios dreht, ändert genau dieses C — Ü 19.1 rechnet die Mittelwellen-Skala nach.)

19.3 Das kleinste Yee-Gitter

Jetzt ins Labor — und das Labor ist diesmal verblüffend klein. Unser FDTD-Verfahren aus Kapitel 5 brauchte zwei *Felder* auf einem Gitter; der Schwingkreis braucht zwei *Zahlen*: q und i. Null Raumdimensionen, das ganze System ein Punkt. Aber die Struktur der Gleichungen ist dieselbe — q ändert sich durch i, i ändert sich durch q —, also übertragen wir auch das Erfolgsrezept: **versetzte Zeiten.** Die Ladung lebt zu ganzen Zeitschritten q^n, der Strom zu halben $i^{n+1/2}$, jeder macht seinen Schritt mit dem aktuellsten Wert des anderen:

$$i^{n+1/2} = i^{n-1/2} - \frac{\Delta t}{L}\,\frac{q^n}{C}, \qquad q^{n+1} = q^n + \Delta t\, i^{n+1/2}.$$

Das ist der Leapfrog aus Kapitel 5 in seiner kleinstmöglichen Form — und der Kondensator-Strom-Tanz ist das E-B-Schachbrett des Yee-Gitters, auf einen einzigen Punkt zusammengezogen.

In Kapitel 5 stand neben dem Leapfrog der naive Euler-Schritt, bei dem *beide* Größen aus den *alten* Werten schreiten — und der dort das Feld explodieren ließ. Hier ist das System so klein, dass wir dem Versagen erstmals komplett mit Papier und Bleistift beikommen. Vorher du:

> Vorhersage-Punkt
>
> Wir simulieren den idealen Schwingkreis ($L = C = 1$, also $\omega_0 = 1$, Periode $T_0 = 2\pi$) über 30 Perioden, einmal mit Euler, einmal mit Leapfrog, Schrittweite $\Delta t = 0{,}05$ (126 Schritte pro Periode — üppig). Was macht die Schwingungsamplitude in den beiden Läufen: bleibt sie, schrumpft sie, wächst sie? Und falls sie sich ändert — wie schnell?

Die Handrechnung dazu ist zwei Zeilen lang und einer der schönsten Momente dieses Buchs. Euler macht (mit $L = C = 1$) aus dem Paar (q, i) den neuen Zustand $q' = q + \Delta t\, i$ und $i' = i - \Delta t\, q$. Berechne die neue Energie:

$$q'^2 + i'^2 = (q + \Delta t\, i)^2 + (i - \Delta t\, q)^2 = q^2 + i^2 + \Delta t^2 (q^2 + i^2) = \left(1 + \Delta t^2\right)\left(q^2 + i^2\right)$$

— die gemischten Terme $\pm 2\Delta t\, qi$ heben sich exakt weg, der quadratische Rest bleibt. **Euler multipliziert die Energie in jedem Schritt mit exakt $1 + (\omega_0 \Delta t)^2$.** Kein Rauschen, kein Zufall: ein eingebautes exponentielles Aufblähen, das mit kleinerem Δt langsamer wird, aber nie verschwindet. Genau das war in Kapitel 5 die Explosion — dort konnten wir sie nur vorführen, hier können wir sie *ausrechnen*.

Nun das Experiment. **Frage:** Packt der Schwingkreis seine Energie wirklich nur um — und hält unser Leapfrog diese Buchhaltung, während Euler sie aufbläht? **Bühne:** keine — nur die zwei Zahlen q und i mit $L = C = 1$. **Anfangsbedingungen:** $q_0 = 1$ (Kondensator voll geladen), $i = 0$ (noch kein Strom) — alles startet im E-Topf. **Messgrößen:** $E_C = q^2/2C$, $E_L = Li^2/2$ und ihre Summe, über drei Perioden; dazu die Energieverläufe beider Verfahren über 30 Perioden. **Erfolgskriterium:** Beim Leapfrog bleibt die *versetzte* Summe (gleich mehr dazu) auf Maschinengenauigkeit konstant; Eulers Energie folgt der eben hergeleiteten $\left(1 + (\omega_0 \Delta t)^2\right)^n$-Kurve.

```
import numpy as np
import matplotlib.pyplot as plt
```

```
L, C = 1.0, 1.0                  # Eigenkreisfrequenz w0 = 1/sqrt(LC) = 1
OMEGA0 = 1.0 / np.sqrt(L * C)
T0 = 2 * np.pi / OMEGA0          # Periode

def leapfrog(R=0.0, dt=0.05, T=200.0, q0=1.0):
    """q zu ganzen, i zu halben Zeiten; R semi-implizit (Kapitel 6)."""
    s = R * dt / (2 * L)
    ca, cb = (1 - s) / (1 + s), (dt / L) / (1 + s)
    q, i, waerme = q0, 0.0, 0.0
    qs, is_, EC, EL, U, W = [], [], [], [], [], []
    for n in range(int(T / dt)):
        i_alt = i
        i = ca * i - cb * q / C
        q_alt = q
        q = q + dt * i
        waerme += R * ((i_alt + i) / 2) ** 2 * dt   # zeitzentriert
        qs.append(q); is_.append(i)
        EC.append(q**2 / (2 * C)); EL.append(L * i**2 / 2)
        U.append(q_alt * q / (2 * C) + L * i**2 / 2)  # versetzt!
        W.append(waerme)
    return [np.array(a) for a in (qs, is_, EC, EL, U, W)]

DT = 0.05
q, i, EC, EL, U, _ = leapfrog(dt=DT, T=3 * T0)
t = np.arange(1, len(q) + 1) * DT

fig, ax = plt.subplots(figsize=(8, 3.4))
ax.plot(t / T0, EC, color="crimson", lw=2, label="$E_C$ (E-Feld)")
ax.plot(t / T0, EL, color="royalblue", lw=2, label="$E_L$ (B-Feld)")
ax.plot(t / T0, EC + EL, color="black", lw=2, label="Summe")
ax.plot(t / T0, 0.5 * np.cos(OMEGA0 * t) ** 2, "k--", lw=0.9,
        label="Theorie $\\cos^2$, $\\sin^2$")
ax.plot(t / T0, 0.5 * np.sin(OMEGA0 * t) ** 2, "k--", lw=0.9)
ax.set_xlabel("Zeit in Perioden")
ax.set_ylabel("Energie")
ax.legend(fontsize=9, ncol=2)
plt.show()

schwankung = np.ptp(U) / U[0]    # np.ptp: Spannweite (max - min)
print(f"Versetzte Energie qⁿqⁿ⁺¹/2C + Li²/2: "
      f"Schwankung {schwankung:.1e}")
```

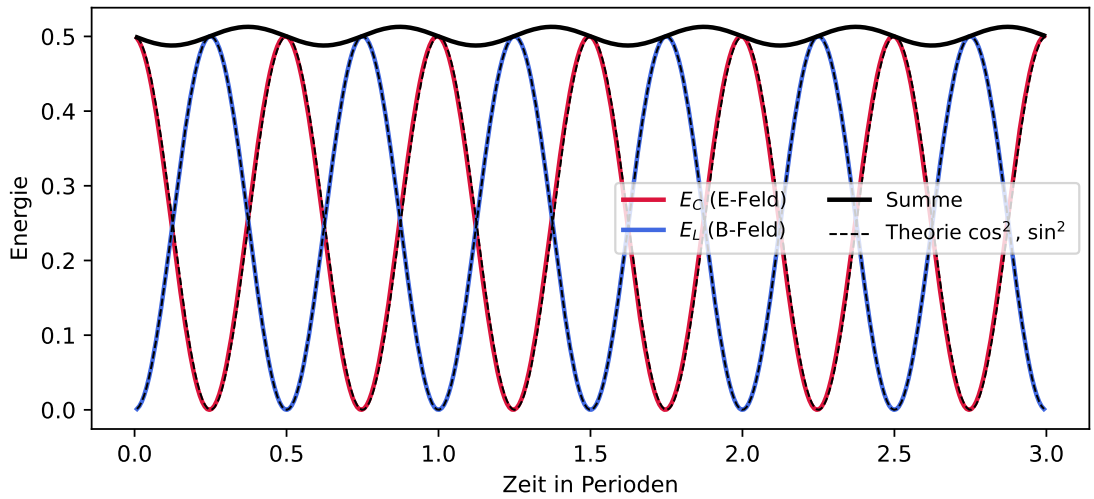


Abbildung 19.1: Das Energie-Pendel des idealen Schwingkreises (Leapfrog, Δt = 0,05): Die Energie wandert zweimal pro Periode vollständig vom Kondensator (E-Feld, rot) in die Spule (B-Feld, blau) und zurück — exakt auf den gestrichelten Theoriekurven cos² und sin². Die Summe (schwarz) ist eine Gerade: Es ist immer dieselbe Energie, nur umgepackt. Das Bild ist das Schaltkreis-Double der stehenden Welle aus Kapitel 18.

```
Versetzte Energie qⁿqⁿ⁺¹/2C + Li²/2: Schwankung 1.8e-15
```

Das Pendel läuft punktgenau auf den $\cos^2$/$\sin^2$-Kurven — und die ausgedruckte Zahl darunter ist die eigentliche Sensation: Schwankung um wenige 10^{-15} — Maschinengenauigkeit. Aber Vorsicht, hier steckt das Kapitel-18-Kleingedruckte drin: Konstant auf diesem Niveau ist nicht die naive Summe $q^2/2C + Li^2/2$ (die zittert leicht, weil q und i zu *verschiedenen* Zeiten leben), sondern die **versetzt gebuchte** Energie

$$U^{n+1/2} = \frac{q^n\, q^{n+1}}{2C} + \frac{L\left(i^{n+1/2}\right)^2}{2}$$

— das Produkt zweier benachbarter q-Werte schiebt den E-Topf rechnerisch auf den Halbschritt, auf dem der Strom ohnehin wohnt. In Kapitel 18 war das die Identität $\frac{\varepsilon_0}{2}\sum E^n E^{n+1}$; hier ist es dieselbe Pointe mit einem einzigen Summanden. Der Yee-Versatz ist kein Implementierungsdetail, sondern der Grund, warum dieses Verfahren Energie *exakt* buchhalten kann.

Jetzt die Gegenprobe mit Euler — und mit der Stabilitätsgrenze, die das Leapfrog-Verfahren auch in null Dimensionen hat.

```
# von oben: np, plt, leapfrog, L, C, OMEGA0, T0, DT

def euler(dt=DT, T=30 * T0, q0=1.0):
    """Beide Updates aus den ALTEN Werten — Kapitel 5 lässt grüßen."""
    q, i = q0, 0.0
    U = []
    for n in range(int(T / dt)):
        q, i = q + dt * i, i - dt * q / (L * C)
        U.append(q**2 / (2 * C) + L * i**2 / 2)
    return np.array(U)

U_eu = euler()
_, _, _, _, U_lf, _ = leapfrog(dt=DT, T=30 * T0)
t30 = np.arange(1, len(U_eu) + 1) * DT / T0
theorie = 0.5 * (1 + (OMEGA0 * DT) ** 2) ** np.arange(1, len(U_eu) + 1)

fig, ax = plt.subplots(figsize=(8, 3.2))
ax.semilogy(t30, U_eu, color="crimson", lw=2, label="Euler")
ax.semilogy(t30, theorie, "k--", lw=1.2,
            label="Theorie $(1+(\\omega_0\\Delta t)^2)^n$")
ax.semilogy(t30, U_lf, color="royalblue", lw=2,
            label="Leapfrog (versetzt gebucht)")
ax.set_xlabel("Zeit in Perioden")
ax.set_ylabel("Energie")
ax.legend(fontsize=9)
plt.show()

print(f"Euler nach 30 Perioden: Energie x{U_eu[-1] / 0.5:.0f} "
      f"(Theorie x{theorie[-1] / 0.5:.0f})")
for dt_test in (1.99, 2.01):
    q_t, *_ = leapfrog(dt=dt_test, T=400.0)
    print(f"Leapfrog mit ω₀Δt = {dt_test}: "
          f"max|q| = {np.abs(q_t).max():.0e}")
```

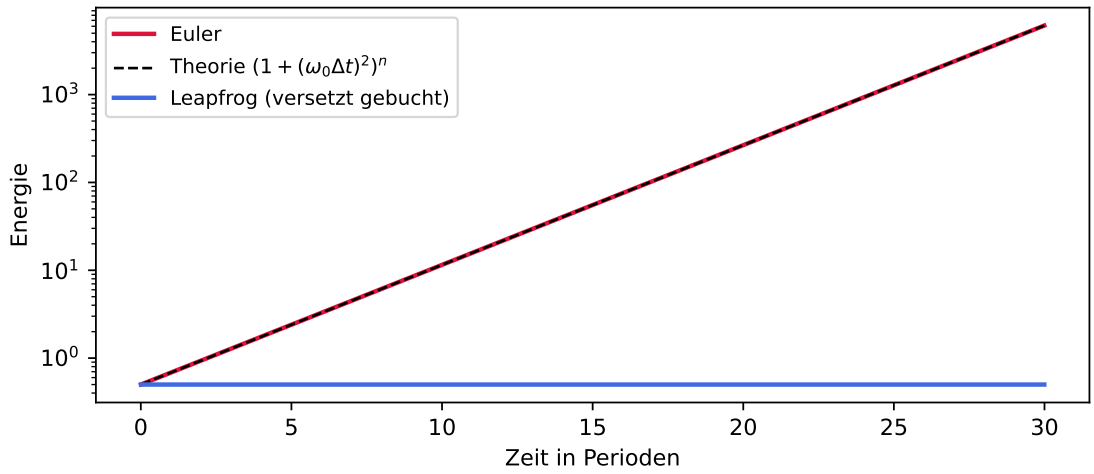


Abbildung 19.2: Energie über 30 Perioden, logarithmisch. Eulers Energie (rot) wächst exakt auf der hergeleiteten Theoriegeraden $(1+(\omega_0\Delta t)^2)^n$ (schwarz gestrichelt) — nach 30 Perioden ist sie um den Faktor 12 000 aufgebläht. Der Leapfrog (blau) hält seine versetzte Energie über dieselbe Strecke auf vierzehn Stellen konstant; im Bild ist seine Linie schlicht waagerecht.

```
Euler nach 30 Perioden: Energie x12219 (Theorie x12219)
Leapfrog mit ω₀Δt = 1.99: max|q| = 1e+01
Leapfrog mit ω₀Δt = 2.01: max|q| = 1e+18
```

Drei Befunde in einem Bild samt Ausdruck. Erstens: Euler liegt *deckungsgleich* auf der von Hand hergeleiteten Geraden — die Explosion ist Algebra, keine Laune. Zweitens: Der Leapfrog hält. Drittens, aus den letzten beiden Zeilen: Auch der Leapfrog hat seine Grenze, und sie liegt **exakt bei** $\omega_0\Delta t = 2$ — bei $1{,}99$ bleibt die Schwingung endlich, bei $2{,}01$ explodiert sie um achtzehn Größenordnungen. Das ist die CFL-Bedingung aus Kapitel 8 in ihrer nacktesten Form: Dort lautete sie $S \leq 1$ und besagte, dass auch die *schnellste* Gitterwelle — die mit $\omega_{\text{max}}\Delta t = 2$ — noch sauber abgetastet werden muss. Ein räumliches Gitter ist ein Chor aus Schwingkreisen verschiedener Frequenzen, und der Zeitschritt muss den flinksten von ihnen aushalten. Hier gibt es nur einen, also sieht man die Grenze pur.

Bevor wir weiterziehen, die Pflicht-Gegenprobe mit fertigem Werkzeug (Kapitel 11: Vertraue keinem einzelnen Löser): scipy bringt mit `solve_ivp` einen erwachsenen Integrator für gewöhnliche Differentialgleichungen mit — adaptive Schrittweite, eingebaute Fehlerkontrolle.

```
# von oben: np, leapfrog, L, C, T0, DT
from scipy.integrate import solve_ivp   # scipy-ODE-Löser (Kapitel 11)

# Zustand y = [q, i]; Rückgabe: die Zeitableitungen [dq/dt, di/dt]
sol = solve_ivp(lambda t, y: [y[1], -y[0] / (L * C)],
                [0, 30 * T0], [1.0, 0.0],
                rtol=1e-10, atol=1e-12,    # Toleranzen eng anziehen
                dense_output=True)          # Loesung als Funktion
q_lf, *_ = leapfrog(dt=DT, T=30 * T0)
t_lf = np.arange(1, len(q_lf) + 1) * DT
abw = np.abs(q_lf - sol.sol(t_lf)[0]).max()
print(f"Leapfrog vs. solve_ivp über 30 Perioden: "
      f"max. Abweichung {abw:.3f}")
```

```
Leapfrog vs. solve_ivp über 30 Perioden: max. Abweichung 0.044
```

Gut vier Prozent Abweichung — und die sind lehrreich, nicht peinlich. Zwei Effekte stecken darin: Der Leapfrog schwingt minimal zu langsam (seine Gitterfrequenz ist $\omega_{\text{num}} = \frac{2}{\Delta t}\arcsin(\omega_0 \Delta t/2)$, das ist die Dispersionsformel aus Kapitel 8 ohne den Ortsterm — über 30 Perioden läuft eine kleine Phasendifferenz auf), und unser Start setzt $i = 0$ zur Zeit $t = -\Delta t/2$ statt exakt zu $t = 0$ — der halbe Schritt Versatz, der in Kapitel 11 schon Meeps Quellen um 4,7 % verschoben hat. Beide Effekte schrumpfen mit Δt. Die *Energie* hält der Leapfrog trotzdem exakt: Er verteilt seinen Fehler in die Phase, nie in die Amplitude — für lange Läufe ist das Gold wert (das Kleingedruckte verrät den Fachbegriff dafür).

19.4 Drei Schicksale einer Schwingung

Der ideale Kreis schwingt ewig. Echte Spulen haben Drahtwiderstand, echte Kondensatoren Leckströme — zusammen ein Serienwiderstand R, an dem die Leistung $R\,i^2$ in Wärme fällt (das $\vec{J} \cdot \vec{E}$ aus Kapitel 18, auf ein Bauteil eingedampft). Die Bewegungsgleichung bekommt den Reibungsterm $L\,\dot{i} = -q/C - R\,i$, und numerisch behandeln wir ihn genau wie das σ in Kapitel 6: semi-implizit, mit den ca/cb-Koeffizienten (der Code oben in `leapfrog()` kann das längst — wir haben den Parameter `R` bisher nur nicht benutzt).

Auch die Wärme buchen wir mit der Sorgfalt aus Kapitel 18: zeitzentriert. Der Strom lebt auf Halbschritten; die in einem ganzen Schritt verheizte Energie ist $R\left(\frac{i_{\text{alt}}+i_{\text{neu}}}{2}\right)^2 \Delta t$ — das arithmetische Mittel schiebt i^2 auf den ganzen

Schritt. Mit dieser Paarung (und nur mit ihr) muss die Bilanz „EM-Energie + kumulierte Wärme = konstant" *exakt* aufgehen.

Frage: Wohin geht die Energie des gedämpften Kreises, und wie schnell? **Bühne und Anfangsbedingungen:** wie eben, dazu $R = 0{,}12$ (in unseren Einheiten; das gibt gut acht Schwingungen, bis die Energie auf 1/e fällt). **Messgrößen:** $E_C + E_L$, kumulierte Wärme, ihre Summe; dazu der Logarithmus der EM-Energie für den Abkling-Fit (die Methode aus Kapitel 15). **Erfolgskriterium:** Summe konstant auf Maschinengenauigkeit; Abklingrate R/L, also $Q = \omega_0/\text{Rate} = \omega_0 L/R = 8{,}33$.

```
# von oben: np, plt, leapfrog, L, C, OMEGA0

R = 0.12
q, i, EC, EL, U, W = leapfrog(R=R, dt=0.01, T=120.0)
t = np.arange(1, len(q) + 1) * 0.01

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9, 3.4))
ax1.plot(t, U, color="black", lw=1.5, label="EM-Energie")
ax1.plot(t, W, color="darkorange", ls="--", lw=1.5,
         label="Wärme (kumuliert)")
ax1.plot(t, U + W, color="seagreen", ls=":", lw=2.2, label="Summe")
ax1.set_xlabel("Zeit")
ax1.set_ylabel("Energie")
ax1.legend(fontsize=9)

ax2.semilogy(t, U, color="black", lw=1.2, label="EM-Energie")
ax2.semilogy(t, U[0] * np.exp(-R / L * (t - t[0])), "k--", lw=1.2,
             label="Theorie $e^{-Rt/L}$")
ax2.set_xlabel("Zeit")
ax2.set_ylabel("Energie (log)")
ax2.legend(fontsize=9)
plt.show()

bilanz = np.ptp(U + W) / U[0]
rate = -np.polyfit(t, np.log(U), 1)[0]
print(f"EM + Wärme: Schwankung {bilanz:.1e}")
print(f"Abklingrate {rate:.4f} (Theorie R/L = {R / L}) -> "
      f"Q = ω₀/Rate = {OMEGA0 / rate:.2f} "
      f"(Theorie ω₀L/R = {OMEGA0 * L / R:.2f})")
```

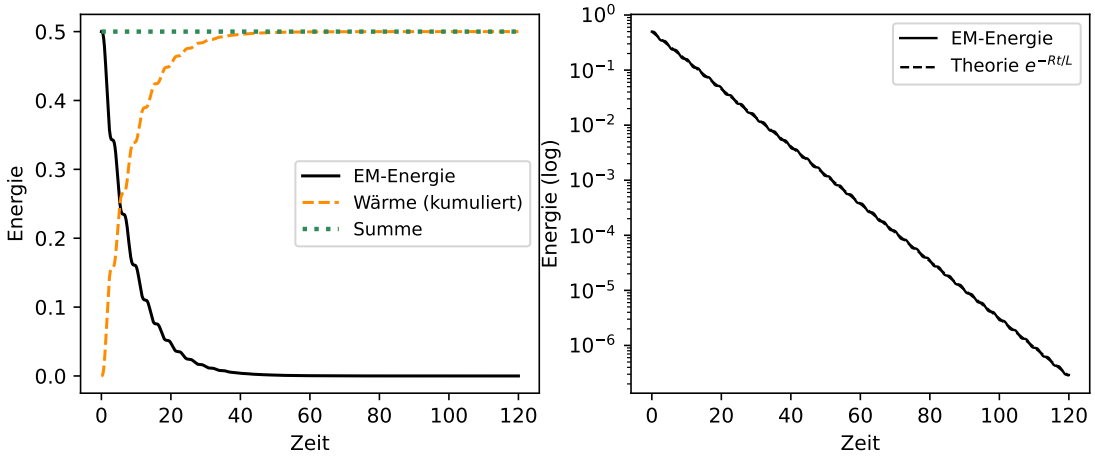


Abbildung 19.3: Der gedämpfte Schwingkreis (R = 0,12). Links: Die EM-Energie (schwarz) fällt, die kumulierte Wärme (orange, gestrichelt) steigt spiegelbildlich — ihre Summe (grün gepunktet) ist eine perfekte Waagerechte: Energie verschwindet nicht, sie wechselt nur das Konto. Rechts: dieselbe EM-Energie logarithmisch; sie folgt der Theoriegeraden e^(−Rt/L) (gestrichelt) — aus der Steigung folgt Q = 8,33, exakt ω_0L/R. Die kleine Treppe um die Gerade ist echte Physik: Wärme entsteht nur, wenn Strom fließt (R·i^2), und der ist zweimal pro Periode null — die Energie fällt stoßweise.

```
EM + Wärme: Schwankung 5.8e-14
Abklingrate 0.1200 (Theorie R/L = 0.12) -> Q = ω₀/Rate = 8.33 (Theorie ω₀L/R = 8.33)
```

Die Bilanz schließt auf 10^{-13} — das diskrete Double des Energiesatzes, zum dritten Mal nach Kapitel 18 (Vakuum und σ-Zone), diesmal im Schaltkreis. Und die Güte ist exakt die aus Kapitel 15: Dort war $Q = \omega W/P$ definiert und über $2\pi f/\sigma_D$ gemessen; hier ist P die Wärmeleistung und es kommt $Q = \omega_0 L/R$ heraus — gleiche Definition, gleiches Abklinggesetz $e^{-\omega_0 t/Q}$, nur der Verlustmechanismus hat einen anderen Namen. Ein Schwingkreis mit $Q = 8$ ist übrigens ein schlechter: Radio-Schwingkreise erreichen 100, Quarze hunderttausend — je höher Q, desto schärfer die Frequenz (Kapitel 15: $\Delta f = f/Q$) und desto besser der Takt.

Womit wir beim dritten Schicksal wären. Gedämpft stirbt die Schwingung — ein Sender braucht sie aber *dauerhaft*. Die Lösung: eine **Rückkopplung**, die im Takt der Schwingung Energie nachschiebt, wie man eine Schaukel im richtigen Moment anstößt. Elektrisch wirkt so eine mitschwingende Quelle wie ein *negativer Widerstand* $-k$: Statt $R\,i^2$ zu entnehmen, speist sie $k\,i^2$

ein. Für $k = R$ heben sich Verlust und Nachschub exakt auf, und der Kreis schwingt ewig — auf dem Papier.

In Wirklichkeit ist $k = R$ eine **Messerschneide**: Ist die Rückkopplung ein Haar zu schwach, klingt die Schwingung ab; ein Haar zu stark, wächst sie über alle Grenzen. Kein Bauteil der Welt trifft und hält eine solche Gleichheit. Jeder echte Oszillator löst das mit einem Trick, der so robust ist, dass man ihn leicht übersieht: Die Rückkopplung ist **bei kleinen Amplituden zu stark und sättigt bei großen** — zum Beispiel $k_{\text{eff}} = k\,(1 - (q/q_s)^2)$ mit einer Sättigungsskala q_s. Kleine Schwingungen werden angefacht ($k_{\text{eff}} > R$), große gebremst ($k_{\text{eff}} < R$), und dazwischen liegt genau eine Amplitude, bei der sich Zufuhr und Verlust über die Periode die Waage halten. Diese Gleichung hat einen berühmten Namen — **van-der-Pol-Oszillator** — und ihr Verhalten einen schönen Fachbegriff: einen *Grenzzyklus*, eine Schwingung, die ihre Amplitude selbst findet und festhält.

Frage: Stellt sich die Amplitude wirklich von selbst ein — unabhängig davon, wie der Kreis startet? **Bühne:** der Kreis von oben mit $R = 0{,}12$, Rückkopplung $k = 0{,}3$ und $q_s = 1$; gelöst mit `solve_ivp` (die Gleichung ist nichtlinear — genau der Fall, für den man einen adaptiven Löser dabei hat). **Anfangsbedingungen:** einmal ein Hauch von nichts ($q_0 = 0{,}001$ — etwa das thermische Rauschen, aus dem jeder reale Oszillator anläuft), einmal viel zu viel ($q_0 = 3$). **Messgrößen:** $q(t)$ und die Endamplitude beider Läufe. **Erfolgskriterium:** Beide laufen auf dieselbe Amplitude; die Störungstheorie (Kleingedrucktes) sagt $A = 2q_s\sqrt{(k-R)/k} = 1{,}549$ voraus.

```
# von oben: np, plt, solve_ivp, L, C, R

K, QS = 0.30, 1.0

def vdp(t, y):
    q, i = y
    daempfung = R - K * (1 - (q / QS) ** 2)   # saettigende Rueckkopplung
    return [i, (-daempfung * i - q / C) / L]

A_theo = 2 * QS * np.sqrt((K - R) / K)
t_aus = np.linspace(0, 150, 4000)
fig, ax = plt.subplots(figsize=(8, 3.4))
for q0, farbe, lab in ((1e-3, "purple", "Start $q_0 = 0{,}001$"),
                       (3.0, "0.6", "Start $q_0 = 3$")):
    s = solve_ivp(vdp, [0, 150], [q0, 0.0], rtol=1e-9, atol=1e-12,
                  dense_output=True, max_step=0.05)
    ax.plot(t_aus, s.sol(t_aus)[0], color=farbe, lw=0.8, label=lab)
ax.axhline(A_theo, color="black", ls="--", lw=1.2,
```

```
            label="Theorie $\\pm 2 q_s\\sqrt{(k-R)/k}$")
ax.axhline(-A_theo, color="black", ls="--", lw=1.2)
ax.set_xlabel("Zeit")
ax.set_ylabel("q(t)")
ax.legend(fontsize=9, loc="lower right")
plt.show()

tt = np.linspace(300, 400, 4000)
for q0 in (1e-3, 3.0):
    s = solve_ivp(vdp, [0, 400], [q0, 0.0], rtol=1e-9, atol=1e-12,
                  dense_output=True, max_step=0.05)
    print(f"Start q0 = {q0}: Endamplitude "
          f"{np.abs(s.sol(tt)[0]).max():.4f}  "
          f"(Theorie {A_theo:.4f})")
```

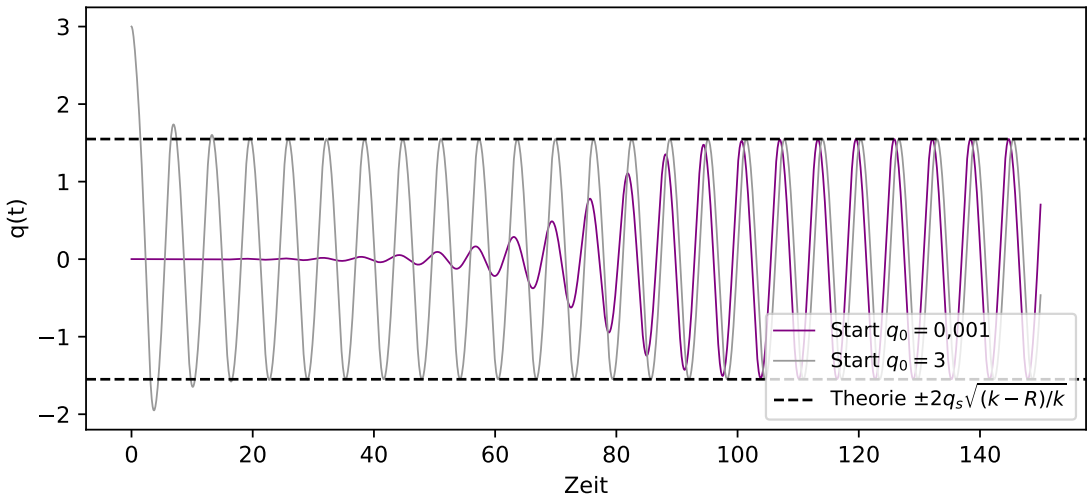


Abbildung 19.4: Der van-der-Pol-Oszillator findet seine Amplitude selbst. Violett: Anlauf aus $q_0 = 0{,}001$ — das „Rauschen“ wird exponentiell angefacht, bis die Sättigung greift. Grau: Start mit viel zu großem $q_0 = 3$ — die Schwingung wird auf denselben Wert heruntergebremst. Beide enden exakt auf dem theoretischen Grenzzyklus $\pm 2q_s\cdot\sqrt{(k-R)/k} = \pm 1{,}549$ (gestrichelte Linien). Die Amplitude ist eine Eigenschaft der Schaltung, nicht der Vorgeschichte — wie die Frequenz.

```
Start q0 = 0.001: Endamplitude 1.5495  (Theorie 1.5492)
Start q0 = 3.0: Endamplitude 1.5495  (Theorie 1.5492)
```

Beide Startwerte enden auf der vierten Nachkommastelle bei derselben Amplitude. Das ist der Kern jedes Oszillators, vom Quarz in der Armbanduhr bis zum Sender im WLAN-Router: ein schwingfähiges Paar Töpfe (das die **Frequenz** setzt), ein Verlust, eine übersteuerte Rückkopplung mit Sättigung (die die **Amplitude** setzt) — und eine Energiequelle ohne jede Frequenz, meist schlichter Gleichstrom. Die Pendeluhr ist dasselbe Prinzip aus Messing: Das Pendel ist der Schwingkreis, das aufgezogene Gewicht die DC-Quelle, und die *Hemmung* — der Anker, der das Zahnrad genau zweimal pro Periode freigibt — ist die getaktete Rückkopplung (Ü 19.4 baut sie nach). Damit lässt sich die Analogie-Landkarte zeichnen, die alle Bilder dieses Kapitels nebeneinanderlegt:

Tabelle 19.2: Ein Bild für alles: dieselben sechs Rollen in drei Kostümen.

Schwingkreis	Feder und Masse	Pendeluhr
Kondensator C (E-Feld-Energie)	gespannte Feder (Lageenergie)	Pendel am Umkehrpunkt
Spule L (B-Feld-Energie)	bewegte Masse (Schwung)	Pendel im Tiefpunkt (volle Fahrt)
Widerstand R (→ Wärme)	Reibung	Luft- und Lagerreibung
Rückkopplung mit Sättigung	Anstoßen im richtigen Moment	Hemmung (Escapement)
DC-Quelle (Netzteil)	dein Arm beim Anschaukeln	das aufgezogene Gewicht
$f = 1/(2\pi\sqrt{LC})$	f aus Federhärte und Masse	f aus der Pendellänge

19.5 Der Hohlraum ist ein Schwingkreis ohne Bauteile

Jetzt schließt sich der Kreis zu Kapitel 15 und 18. Der Resonatorkasten aus Kapitel 15 — vier mal drei Einheiten, ideal leitende Wände — hat Eigenfrequenzen, Güten, Abklingkurven: alles, was ein Schwingkreis hat. Kein Wunder, behaupten wir jetzt: **Er ist einer.** Seine Grundmode hat ihr E-Feld als Bauch in der Kastenmitte (ein verteilter Kondensator) und ihr B-Feld als Wirbel darum herum (eine verteilte Spule); die Maxwell-Gleichungen verdrahten beide. Wenn das stimmt, muss die Energie der Mode genauso zwischen E-Topf und B-Topf pendeln wie in Abbildung 19.1. Meep kann beide Töpfe direkt wiegen — und bevor wir es tun, die Pflichtfrage:

Vorhersage-Punkt

Die Grundmode TM_{11} schwingt mit $f_{11} = 0{,}2083$ (Kapitel-15-Formel). Mit welcher Frequenz pendelt die **Energie** zwischen E-Topf und B-Topf: mit f_{11}, mit $f_{11}/2$ — oder mit $2f_{11}$? (Denk an Abbildung 19.1: Wie oft pro Periode ist der Kondensator leer?)

Das Experiment dazu. **Frage:** Pendelt die Modenenergie vollständig und mit welcher Frequenz — und klingt sie mit Verlusten so ab, wie es der Kapitel-15-Anker verlangt? **Bühne:** der Kasten aus Kapitel 15 (4×3, Auflösung 24, Zellrand = PEC-Wand). **Anfangs- und Randbedingungen:** Ein schmalbandiger Gauß-Puls bei f_{11} (Bandbreite $f_{11}/8$, damit keine Nachbarmode mitschwingt) lädt die Grundmode; nach Quellende läuft der Kasten frei. **Messgrößen:** die elektrische und die magnetische Gesamtenergie im Kasten, alle zwei Zeitschritte; danach derselbe Lauf mit Materialverlusten $\sigma_D = 0{,}02$ (die Konvention, die wir in Kapitel 15 selbst vermessen haben: Felder klingen mit Rate $\sigma_D/2$). **Erfolgskriterium:** Pendeln mit $2f_{11} = 0{,}4167$, Topfwechsel vollständig (0 bis 100 %), Summe konstant; im Verlustlauf fällt die *Energie* mit Rate σ_D (doppelte Feldrate, weil Energie quadratisch in den Feldern ist).

```
# von oben: np, plt
import meep as mp

mp.verbosity(0)

LX, LY = 4.0, 3.0                        # der Kasten aus Kapitel 15
AUFL = 24
DT_MEEP = 0.5 / AUFL
F11 = 0.5 * np.sqrt((1 / LX) ** 2 + (1 / LY) ** 2)
# mp.Volume: ein benanntes Raumgebiet (hier: der ganze Kasten)
KASTEN = mp.Volume(center=mp.Vector3(), size=mp.Vector3(LX, LY))

def hohlraum(sigma_d=0.0, dauer=120.0):
    geometrie = []
    if sigma_d > 0:
        geometrie = [mp.Block(size=mp.Vector3(mp.inf, mp.inf, mp.inf),
                              material=mp.Medium(D_conductivity=sigma_d))]
    quelle = [mp.Source(mp.GaussianSource(F11, fwidth=F11 / 8),
                        component=mp.Ez, center=mp.Vector3(0.3, 0.2))]
    sim = mp.Simulation(cell_size=mp.Vector3(LX, LY), resolution=AUFL,
                        sources=quelle, geometry=geometrie)
    t, uE, uB = [], [], []
```

```
    def messen(s):
        t.append(s.meep_time())
        # Meep wiegt die Toepfe direkt: Integral der Energiedichte
        uE.append(s.electric_energy_in_box(box=KASTEN))
        uB.append(s.magnetic_energy_in_box(box=KASTEN))

    # synchronized_magnetic: schiebt B fuer die Messung auf den
    # E-Zeitpunkt — sonst zittert die Summe (der Yee-Versatz!)
    sim.run(mp.after_sources(mp.synchronized_magnetic(
        mp.at_every(2 * DT_MEEP, messen))), until_after_sources=dauer)
    return np.array(t), np.array(uE), np.array(uB)

t_h, uE_h, uB_h = hohlraum()
summe = uE_h + uB_h
t_rel = t_h - t_h[0]

SD = 0.02
t_v, uE_v, uB_v = hohlraum(sigma_d=SD, dauer=200.0)
summe_v = uE_v + uB_v
t_vrel = t_v - t_v[0]

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9, 3.4))
ax1.plot(t_rel, uE_h / summe.mean(), color="crimson", lw=1.4,
         label="E-Topf")
ax1.plot(t_rel, uB_h / summe.mean(), color="royalblue", lw=1.4,
         label="B-Topf")
ax1.plot(t_rel, summe / summe.mean(), color="black", lw=1.6,
         label="Summe")
ax1.set_xlim(0, 12)
ax1.set_xlabel("Zeit nach Quellende")
ax1.set_ylabel("Energieanteil")
ax1.legend(fontsize=9, loc="center right")

ax2.semilogy(t_vrel, summe_v / summe_v[0], color="black", lw=1.4,
             label="Gesamtenergie")
ax2.semilogy(t_vrel, np.exp(-SD * t_vrel), "k--", lw=1.2,
             label="Theorie $e^{-\\sigma_D t}$")
ax2.set_xlabel("Zeit nach Quellende")
ax2.set_ylabel("Energie (normiert, log)")
ax2.legend(fontsize=9)
plt.show()

sp = np.abs(np.fft.rfft(uE_h - uE_h.mean()))
```

```
fr = np.fft.rfftfreq(len(t_h), t_h[1] - t_h[0])
print(f"Pendel-Frequenz {fr[sp.argmax()]:.4f}  "
      f"(2 f11 = {2 * F11:.4f})")
print(f"E-Topf: {uE_h.min() / summe.mean() * 100:.1f} % bis "
      f"{uE_h.max() / summe.mean() * 100:.1f} % der Gesamtenergie; "
      f"Summe konstant auf {np.ptp(summe) / summe.mean() * 100:.1f} %")
rate = -np.polyfit(t_v, np.log(summe_v), 1)[0]
print(f"Verlustlauf: Abklingrate {rate:.4f}  (σ_D = {SD}) -> "
      f"Q = 2πf/σ_D = {2 * np.pi * F11 / SD:.1f}")
```

```
Using MPI version 4.1, 1 processes
```

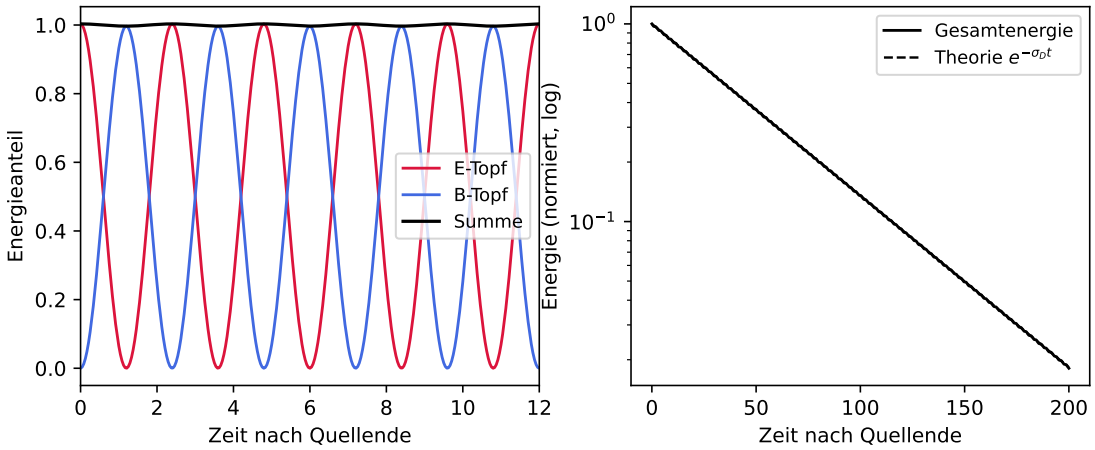


Abbildung 19.5: Der Hohlraum als Schwingkreis. Links: Nach dem Quellende pendelt die Energie der Grundmode vollständig zwischen E-Topf (rot) und B-Topf (blau) — gemessen mit Frequenz 0,4165, exakt das erwartete $2 \cdot f_{11} = 0{,}4167$. Die Summe (schwarz) bleibt waagerecht: derselbe Befund wie im Bauteil-Schwingkreis (Abbildung 19.1), nur dass L und C hier über den Raum verschmiert sind. Rechts: Mit Materialverlusten σ_D = 0,02 fällt die Gesamtenergie logarithmisch exakt auf der Theoriegeraden e^(−σ_D·t) — die Feldrate σ_D/2 aus Kapitel 15, verdoppelt fürs Quadrat.

```
Pendel-Frequenz 0.4165  (2 f11 = 0.4167)
E-Topf: 0.0 % bis 100.3 % der Gesamtenergie; Summe konstant auf 0.7 %
Verlustlauf: Abklingrate 0.0200  (σ_D = 0.02) -> Q = 2πf/σ_D = 65.4
```

Alle drei Vorhersagen treffen ein. Die Energie pendelt mit $2f_{11}$ — zweimal pro Feldperiode ist der E-Topf leer, genau wie der Kondensator in Abbildung 19.1 und die stehende Welle in Kapitel 18 (die Energie ist quadratisch im Feld, darum verdoppelt sich ihre Frequenz). Der Topfwechsel ist vollständig: 0 bis 100 Prozent. Und der Verlustlauf liefert die Rate σ_D auf vier Stellen — was $Q = 2\pi f_{11}/\sigma_D = 65{,}4$ ergibt, exakt die Zahl, die Kapitel 15 für dieselbe Mode mit drei unabhängigen Methoden gemessen hat. Damals war $Q = 2\pi f/\sigma_D$ ein Anker aus Abklingkurve, Linienbreite und Harminv; heute wissen wir, *was* da abklingt: die pendelnde Energie eines Schwingkreises, dessen L und C über den Kasten verschmiert sind.

Dass die schwarze Summenlinie links bei genauem Hinsehen um ein halbes Prozent kräuselt, ist kein Messfehler, sondern ein alter Bekannter: Meeps Energiezähler bucht naiv — E und B leben auf versetzten Zeiten, und `synchronized_magnetic` interpoliert das B-Feld nur näherungsweise auf den E-Zeitpunkt (ohne diese Hilfe wäre das Kräuseln viermal größer; probiere es aus). Die exakt versetzte Buchhaltung aus Abbildung 19.1 und Kapitel 18 bekommt man nur, wenn man selbst Buch führt.

Der Blick nach vorn: Ein Kasten hat nicht *eine* Mode, sondern abzählbar viele (Kapitel 15 hat vier vermessen) — **ein Hohlraum ist ein ganzes Sortiment von Schwingkreisen**, jede Mode mit eigenem f und eigenem Q. Genau so werden wir in Kapitel 27 den Fabry-Pérot-Resonator und den Ringresonator lesen, und der Laser ist am Ende ein solcher Schwingkreis mit einer Rückkopplung, die nicht aus einem Transistor kommt, sondern aus stimuliertem Licht.

19.6 Vom Kraftwerk zur Antenne

Jetzt lässt sich die Eingangsfrage vollständig beantworten — und zuerst räumen wir die naheliegende falsche Antwort ab:

> ⚠ Naheliegende Vermutung
>
> **„Die 50 Hz aus der Steckdose werden im Sender zu 100 MHz hochtransformiert — so wie ein Trafo die Spannung hochtransformiert."**
> *Warum sie naheliegt:* Ein Trafo macht aus 230 V mühelos 10 000 V; dass ein ähnliches Bauteil aus 50 Hz eben 100 MHz macht, klingt nur konsequent. Und irgendwoher müssen die 100 MHz ja kommen.
> *Was stattdessen stimmt:* Ein Transformator ändert *nie* die Frequenz — er kann es nicht, denn die Frequenz ist kein Pegel, den man übersetzen

könnte, sondern das zeitliche Muster des Signals selbst (und ein Trafo ist ein lineares Bauteil: Was mit 50 Hz hineinschwingt, kommt mit 50 Hz heraus). **Energie hat keine Frequenz.** Die 100 MHz werden im Sender *neu erzeugt*: von einem Oszillator, dessen schwingfähige Struktur — ein LC-Kreis oder ein Quarz — genau diese Eigenfrequenz hat. Die Steckdose liefert nur die Ware Energie; das Muster prägt ihr erst der Oszillator auf.

Der vollständige Weg der Energie von der Steckdose bis zur Funkwelle hat damit vier Stationen:

Tabelle 19.3: Von 50 Hz zu 100 MHz: Die Frequenz wird nicht gewandelt, sondern an Station 2 neu geboren.

Station	was mit der Energie passiert	Frequenz danach
Netzteil	richtet die 50 Hz zu Gleichstrom	keine (DC)
Oszillator (LC oder Quarz)	füllt sie in eine schwingfähige Struktur	$f_0 = \frac{1}{2\pi\sqrt{LC}}$ — neu erzeugt
Verstärker	macht die Schwingung kräftig (mehr DC-Energie ins selbe Muster)	unverändert
Antenne	übergibt sie als Welle ans Feld	unverändert

Im Alltag ist die schwingfähige Struktur meist kein LC-Paar aus Einzelteilen, sondern ein **Schwingquarz**: ein Kristallplättchen, das mechanisch schwingt wie eine winzige Stimmgabel und dessen Piezoeffekt die Schwingung elektrisch ankoppelt. Sein Vorteil ist das Q: Wo ein LC-Kreis bei ein paar hundert endet, erreicht ein Quarz 10^4 bis 10^6 — entsprechend schärfer (Kapitel 15: $\Delta f = f/Q$) und stabiler ist sein Takt. Der Quarz in jeder Armbanduhr schwingt bei 32 768 Hz, und warum ausgerechnet dort, klärt Ü 19.1. Die Rückkopplung übernimmt ein Transistor: ein Ventil, das den großen DC-Energiefluss des Netzteils im Takt der Schwingung öffnet — die Hemmung der Pendeluhr, in Silizium gegossen, samt Sättigung als Amplitudenregler.

Und damit ist auch klar, was die `Source` in unseren Meep-Läufen die ganze Zeit *war*: die Abstraktion genau dieser Kette. „Hier sitzt ein Sender mit Frequenz f" — Netzteil, Oszillator und Verstärker in einer Programmzeile. Wie die Antenne als letzte Station die Schwingung tatsächlich vom Draht ins Feld ablöst, ist die Geschichte von Teil VI (Kapitel 21); und warum man dafür nicht einfach die 50 Hz direkt auf einen Draht geben kann, beantwortet Kapitel 25 mit einer einzigen Längenrechnung.

Bevor das Kapitel-Programm alles bündelt, der Kern dieses Kapitels in einem Satz: Wer Energie in zwei Formen speichern und verlustarm umpacken

kann, schwingt — mit der Frequenz seiner Struktur, nicht der seiner Quelle.

Merkkasten

- **Energie hat keine Frequenz.** Die Frequenz ist Eigenschaft einer schwingfähigen Struktur ($\omega_0 = 1/\sqrt{LC}$) und wird im Oszillator neu erzeugt — nie „hochtransformiert".
- **Schwingen = Energie-Umpacken** zwischen zwei Töpfen: Kondensator ↔ Spule, E-Feld ↔ B-Feld, Feder ↔ Masse. Die Energie pendelt mit $2f_0$, weil sie quadratisch im Feld ist.
- Der **Leapfrog** ist auch in null Dimensionen der richtige Integrator: versetzte Energie exakt konstant, Stabilität bis $\omega_0\Delta t = 2$; Euler bläht jede Schwingung um exakt $1 + (\omega_0\Delta t)^2$ pro Schritt auf.
- Ein **Oszillator** = Schwingkreis (Frequenz) + Verlust + sättigende Rückkopplung (Amplitude) + DC-Quelle (Energie). Das Schicksal ohne Rückkopplung: $Q = \omega_0 L/R$ Schwingungen, dann ist Schluss.
- Ein **Hohlraum ist ein Schwingkreis ohne Bauteile** — jede Mode ein eigenes LC-Paar mit eigenem Q.

19.7 Das Kapitel-Programm

Das Programm `kap19_oszillator.py` (im Buchpaket unter `programme/kap19/`) bündelt alles: Leapfrog gegen Euler samt Theoriefaktor und Stabilitätsgrenze, die solve_ivp-Gegenprobe, den gedämpften Kreis mit exakter Wärmebilanz und Q-Fit, den van-der-Pol-Grenzzyklus von beiden Seiten, das Hohlraum-Pendel mit Verlustlauf und den Hemmungs-Stoß aus Ü 19.4. Jede Behauptung steht als `assert`; der Lauf braucht eine knappe halbe Minute und endet mit „Alle Tests bestanden."

Roter Faden

Rückwärts: Das **Induktionsgesetz aus Kapitel 3** ist zur Spule geschrumpft, der **Leapfrog aus Kapitel 5** hat seine kleinste Bühne bekommen (und Eulers Explosion endlich ihre Zwei-Zeilen-Herleitung), die **ca/cb-Koeffizienten aus Kapitel 6** dämpfen jetzt Schaltkreise, die **CFL-Grenze aus Kapitel 8** zeigt sich als $\omega_0\Delta t \leq 2$ pur, das **scipy aus Kapitel 11** hat seinen ODE-Auftritt, die **Güte aus Kapitel 15** ihre Schaltkreis-Form $Q = \omega_0 L/R$ (und der σ_D-Anker seine Energie-Lesart), und das **Energie-Pendel aus Kapitel 18** ist zum Bauteilpaar geworden. Vorwärts:

Kapitel 20 fragt, ob man die gedämpfte Schwingung rückwärts rechnen kann (die Wärme wird zum Problem), **Kapitel 21** löst die Schwingung von der Schaltung und schickt sie als Welle davon — die Meep-`Source` bleibt dabei unsere Sender-Abstraktion —, **Kapitel 22** verbindet Schaltkreis- und Feldsicht zur Leitungstheorie (50 Ω!), und **Kapitel 27** liest Fabry-Pérot- und Ringresonatoren als das, was der Hohlraum heute geworden ist: Schwingkreise für Licht.

Übungen

Ü 19.1 (Verstehen). Zwei Frequenz-Steckbriefe: (a) Der Quarz jeder Armbanduhr schwingt bei exakt 32 768 Hz. Was ist an dieser Zahl so besonders, dass die Uhrenindustrie sie seit 1970 verwendet? (Tipp: Wie macht man aus der Schwingung ein Ein-Sekunden-Ticken, wenn die einzige billige Digitalschaltung ein Frequenz*halbierer* ist?) (b) Ein Mittelwellenradio (530 kHz bis 1600 kHz) stimmt mit einem Drehkondensator ab, der sich von 500 pF auf 50 pF drehen lässt. Welche Spule L braucht der Schwingkreis — und passt der Drehbereich zum Frequenzband?

Musterlösung zu Ü 19.1

```python
# von oben: np
print(f"(a) 32768 = 2^{int(np.log2(32768))}")
L_mw = 1 / ((2 * np.pi * 530e3) ** 2 * 500e-12)
print(f"(b) L = {L_mw * 1e6:.0f} µH")
for C_pf in (500, 50):
    f = 1 / (2 * np.pi * np.sqrt(L_mw * C_pf * 1e-12))
    print(f"    C = {C_pf} pF  ->  f = {f / 1e3:.0f} kHz")
```

```
(a) 32768 = 2^15
(b) L = 180 µH
    C = 500 pF  ->  f = 530 kHz
    C = 50 pF  ->  f = 1676 kHz
```

(a) $32\,768 = 2^{15}$: Fünfzehn hintereinandergeschaltete Halbierer-Stufen machen daraus exakt 1 Hz — das Sekunden-Ticken. Jede andere Frequenz bräuchte Teiler durch krumme Zahlen, die Zweierpotenz braucht nur das billigste Digitalbauteil der Welt, fünfzehnmal. (b) Aus $f = 1/(2\pi\sqrt{LC})$ am unteren Bandende folgt $L \approx 180$ µH; weil $f \propto 1/\sqrt{C}$ ist, überstreicht der Faktor 10 im Kondensator den Faktor $\sqrt{10} \approx 3{,}2$ in der Frequenz — gemessen 530 bis 1676 kHz, das Band passt. Genau so

fühlte sich Radio-Abstimmen ein Jahrhundert lang an: Man verdrehte ein C und wanderte damit die $1/\sqrt{LC}$-Skala entlang.

Ü 19.2 (Verstehen/Verändern). Eulers Aufblähfaktor $1 + (\omega_0 \Delta t)^2$ pro Schritt sieht harmlos aus, wenn Δt klein ist. Zeige numerisch, dass er es nicht ist: Miss das Energiewachstum über 10 Perioden für $\Delta t = 0{,}1$, $0{,}05$, $0{,}02$ und $0{,}01$ und vergleiche mit der Näherung $e^{T\Delta t}$ (mit $T = 20\pi$; sie folgt aus $\ln(1 + x) \approx x$). Wie klein müsste Δt sein, damit die Energie über 10 Perioden um höchstens 1 % wächst — und wie viele Schritte wären das?

Musterlösung zu Ü 19.2

```python
# von oben: np, euler, T0
T = 10 * T0
for dt in (0.1, 0.05, 0.02, 0.01):
    U = euler(dt=dt, T=T)
    print(f"dt = {dt}: Energie x{U[-1] / 0.5:7.2f}   "
          f"(Näherung e^(T·dt) = {np.exp(T * dt):7.2f})")
dt_1prozent = np.log(1.01) / T
print(f"Für 1 %: dt = {dt_1prozent:.1e}  ->  "
      f"{T / dt_1prozent:.0f} Schritte für 10 Perioden")
```

```
dt = 0.1: Energie x 517.40   (Näherung e^(T·dt) =  535.49)
dt = 0.05: Energie x  23.01   (Näherung e^(T·dt) =   23.14)
dt = 0.02: Energie x   3.51   (Näherung e^(T·dt) =    3.51)
dt = 0.01: Energie x   1.87   (Näherung e^(T·dt) =    1.87)
Für 1 %: dt = 1.6e-04  ->  396755 Schritte für 10 Perioden
```

Die Näherung trifft gut (sie liegt stets leicht drüber, weil $\ln(1+x) < x$). Das Wachstum pro *Zeiteinheit* ist $e^{\omega_0^2 \Delta t}$ — es schrumpft nur linear mit Δt. Für 1 % über 10 Perioden braucht Euler $\Delta t \approx 1{,}6 \cdot 10^{-4}$, also rund **400 000 Schritte** — während der Leapfrog mit $\Delta t = 0{,}05$ (1257 Schritte) die Energie auf *fünfzehn Dezimalstellen* hält. Der Unterschied ist nicht Fleiß, sondern Struktur: Euler hat das Aufblähen eingebaut, der Leapfrog die Erhaltung.

Ü 19.3 (Verändern). Die Q-Gerade des Schwingkreises: Lass den gedämpften Kreis für $R = 0{,}05$, $0{,}1$, $0{,}2$, $0{,}4$ und $0{,}8$ laufen, miss jeweils die Abklingrate der Energie per Log-Fit und trage $Q = \omega_0/\text{Rate}$ gegen $\omega_0 L/R$ auf. Liegt alles auf der Geraden — auch beim „schlechten" Kreis mit Q nahe 1, wo die Schwingung nach einer Umdrehung fast verhungert? (Kapitel 15 hat dieselbe Messung am Hohlraum gemacht — dort war σ_D der Drehknopf.)

Musterlösung zu Ü 19.3

```
# von oben: np, leapfrog, OMEGA0, L
print(" R      Q gemessen   Q = w0 L/R")
for R_test in (0.05, 0.1, 0.2, 0.4, 0.8):
    *_, U_t, _ = leapfrog(R=R_test, dt=0.005, T=80.0)
    t_t = np.arange(1, len(U_t) + 1) * 0.005
    rate = -np.polyfit(t_t, np.log(U_t), 1)[0]
    print(f"{R_test:5.2f}   {OMEGA0 / rate:9.2f}   "
          f"{OMEGA0 * L / R_test:9.2f}")
```

```
 R      Q gemessen   Q = w0 L/R
 0.05       20.00       20.00
 0.10       10.00       10.00
 0.20        5.00        5.00
 0.40        2.50        2.50
 0.80        1.25        1.25
```

Alle fünf Punkte sitzen auf der Geraden, selbst $Q = 1{,}25$ — die Energie-Abklingrate ist *exakt* R/L, unabhängig davon, wie stark die Dämpfung die Schwingung schon verformt (erst nahe am Kriechfall, $R = 2\sqrt{L/C}$, bräche die Auswertung zusammen, weil nichts mehr schwingt). Die Messkette ist dieselbe wie am Kapitel-15-Hohlraum: anregen, loslassen, Logarithmus fitten — nur der Verlust-Drehknopf heißt hier R statt σ_D.

Ü 19.4 (Übertragen). Die Pendeluhr im Computer: Ersetze die kontinuierliche Rückkopplung des van-der-Pol-Kreises durch eine **Hemmung** — einen festen Stromstoß $\Delta i = 0{,}05$, der genau einmal pro Periode wirkt (im Moment des Nulldurchgangs von q, wenn $|i|$ maximal ist, in Bewegungsrichtung). Starte den gedämpften Kreis ($R = 0{,}12$) mit kleiner Amplitude und beobachte $|i|$ unmittelbar nach jedem Stoß. Auf welchen Wert stellt sich die Schwingung ein? Leite das Gleichgewicht her: Zwischen zwei Stößen fällt die Stromamplitude um den Faktor $e^{-RT_0/2L}$, der Stoß hebt sie um Δi.

Musterlösung zu Ü 19.4

```
# von oben: np, L, C, T0
R_u, DI, dt = 0.12, 0.05, 0.005
s = R_u * dt / (2 * L)
ca, cb = (1 - s) / (1 + s), (dt / L) / (1 + s)
q, i = 0.2, 0.0
amps = []
for n in range(int(600 / dt)):
    i = ca * i - cb * q / C
    q_alt, q = q, q + dt * i
    if q_alt > 0 >= q:            # Nulldurchgang abwärts
        i -= DI                   # Stoß in Bewegungsrichtung
        amps.append(abs(i))
i_stat = DI / (1 - np.exp(-R_u * T0 / (2 * L)))
print(f"|i| nach Stoß: erst {amps[0]:.3f}, zuletzt {amps[-1]:.4f}")
print(f"Gleichgewicht Δi/(1-e^(-R T₀/2L)) = {i_stat:.4f}")
```

```
|i| nach Stoß: erst 0.231, zuletzt 0.1589
Gleichgewicht Δi/(1-e^(-R T₀/2L)) = 0.1592
```

Im Gleichgewicht muss der Stoß genau das ersetzen, was eine Periode Dämpfung wegnimmt: $i_\infty(1 - e^{-RT_0/2L}) = \Delta i$, also $i_\infty = \Delta i/(1 - e^{-RT_0/2L}) = 0{,}159$ — die Messung trifft das auf drei Stellen. Wie beim van der Pol findet die Amplitude sich selbst, aber der Mechanismus ist *getaktet* statt kontinuierlich: zu kleine Schwingung → der relative Verlust pro Periode ist kleiner als der Stoß → sie wächst (und umgekehrt). Genau so hält die Hemmung einer Pendeluhr ihr Pendel seit dem 17. Jahrhundert auf Amplitude — und genau so arbeitet jeder getaktete Treiber, vom Schaukel-Anschubser bis zur Motorzündung.

Das Kleingedruckte

Warum der Leapfrog die Energie hält: Symplektik. Dass der Leapfrog die (versetzte) Energie exakt erhält, ist kein Zufall, sondern Familieneigenschaft: Er gehört zu den *symplektischen* Integratoren, die die geometrische Struktur der Mechanik (die Erhaltung von Phasenraumvolumen) ins Diskrete retten. Solche Verfahren lösen nicht die exakte Gleichung näherungsweise, sondern eine *minimal verschobene* Gleichung **exakt** — unsere versetzte Energie ist genau die Erhaltungsgröße dieser Schattengleichung. Deshalb rechnet die Himmelsmechanik Planetenbahnen über Jahrmillionen mit

Leapfrog-Verwandten („Störmer–Verlet”), und deshalb driftet FDTD auch nach Millionen Zeitschritten nicht in der Amplitude. Der Preis ist der Phasenfehler aus dem solve_ivp-Vergleich — Energie exakt, Tempo leicht daneben: für Stabilität fast immer der bessere Tausch.

Die van-der-Pol-Amplitude. Die Behauptung $A = 2q_s\sqrt{(k-R)/k}$ folgt mit dem Mittelungstrick aus Kapitel 10: Für eine fast sinusförmige Schwingung $q = A\cos\omega_0 t$ ist die mittlere Verlustleistung $\langle (R - k + kq^2/q_s^2)\, i^2\rangle$. Mit $\langle i^2\rangle = A^2\omega_0^2/2$ und $\langle q^2 i^2\rangle = A^4\omega_0^2/8$ (das Produkt $\cos^2\sin^2$ mittelt auf $1/8$) verschwindet sie genau bei $A^2 = 4q_s^2(k-R)/k$. Die Messung in Abbildung 19.4 trifft das auf vier Stellen, weil $k - R$ klein ist; bei brutal übersteuerter Rückkopplung wird die Schwingung merklich unsinusförmig und die Formel zur Näherung — der van der Pol wird dann zum „Relaxationsoszillator” mit Sägezahn-Charakter.

Meeps Energie-Zähler und der Yee-Versatz. `electric_energy_in_box` und `magnetic_energy_in_box` integrieren $\frac{1}{2}\vec{E}\cdot\vec{D}$ bzw. $\frac{1}{2}\vec{H}\cdot\vec{B}$ über das Gebiet — zum *aktuellen* Zeitpunkt, und der ist für E und B verschieden (Kapitel 5). Ohne Gegenmaßnahme zittert die Summe deshalb um ein paar Prozent; `synchronized_magnetic` lässt Meep das B-Feld für die Dauer der Messung um den halben Schritt mitteln und drückt das Zittern in unserem Lauf von 2,8 % auf 0,65 %. Ganz verschwinden kann es so nicht — die Mittelung ist zweiter Ordnung, keine exakte Buchhaltung. Wer es exakt braucht, bucht versetzt ($E^n E^{n+1}$), wie in Kapitel 18 von Hand.

Reale Oszillatoren. Zwischen unserem van der Pol und einem WLAN-Sender liegen drei Ingenieursschichten, die alle dieselbe Physik verfeinern: Der **Quarz** ersetzt das LC-Paar durch einen mechanischen Resonator mit $Q \sim 10^5$ (schärferer Takt, denn das Phasenrauschen eines Oszillators sinkt mit $1/Q^2$); die **PLL** (Phasenregelschleife) vervielfacht eine Quarz-Referenzfrequenz auf die Zielfrequenz — auch sie transformiert keine Frequenz „hoch”, sondern *vergleicht* und *regelt* einen zweiten Oszillator; und die **Amplitudenregelung** ist statt einer rohen Sättigung oft eine eigene Regelschleife, die die Verzerrung kleinhält. Das Grundgerüst bleibt in jedem Fall das von Tabelle 19.2: Töpfe, Verlust, getaktete Zufuhr.

20 Kausalität, Zeitumkehr und Zeitpfeil

Man erkennt in Sekunden, wenn ein Film rückwärts läuft: Scherben springen vom Boden auf und fügen sich zum Glas, Rauch kriecht in den Schornstein zurück, Wellen laufen auf den geworfenen Stein zu. Nichts davon verletzt ein einziges Naturgesetz der Mechanik oder der Elektrodynamik — und genau das ist das Rätsel. Die Maxwell-Gleichungen, mit denen wir seit neunzehn Kapiteln rechnen, enthalten keine eingebaute Zeitrichtung; unser FDTD-Verfahren tut dagegen nichts anderes, als aus der Vergangenheit die Zukunft zu bauen, Schritt für Schritt, niemals umgekehrt. Wer hat recht? In diesem Kapitel spulen wir den Film tatsächlich zurück — wir rechnen Maxwell *rückwärts*, erst am Schwingkreis aus Kapitel 19, dann auf dem 2D-Gitter aus Kapitel 9 — und finden eine Antwort, die präziser ist als beide Alternativen: Die Richtung der Zeit steckt nicht in den Gleichungen. Sie steckt darin, was wir *wissen können*.

Vorher räumen wir die letzte Grundsatzfrage von Teil V ab. Zwei Kapitel lang haben wir Energie gewogen, verfolgt und umgepackt — da liegt ein Verdacht nahe: Wenn $\vec{S}$ zeigt, wohin die Energie fließt, ist dann vielleicht *die Energie* das Eigentliche, und die Felder sind nur ihre Beschreibung? Die Frage klingt philosophisch, hat aber eine harte physikalische Antwort, und sie hängt — das wird sich zeigen — direkt mit der Zeitpfeil-Frage zusammen: Beide Male geht es darum, was der *Zustand* der Welt ist und was nur Buchhaltung über ihn.

Lernziele

Nach diesem Kapitel kannst du …

1. … mit vier unabhängigen Argumenten begründen, warum die Felder der Zustand der Elektrodynamik sind und die Energie ihre Buchhaltung — nicht umgekehrt,
2. … die Zeitumkehr-Invarianz der Maxwell-Gleichungen von Hand zeigen ($t \to -t$, $\vec{B} \to -\vec{B}$) und erklären, wo die Kausalität stattdessen sitzt (in der Wahl der *retardierten* Lösungen),

3. … einen FDTD-Lauf wörtlich rückwärts rechnen — als Symmetrie (B-Feld umdrehen) und als algebraische Inversion — und vorhersagen, wann das gelingt,
4. … den **Informations-Horizont** eines gedämpften Systems berechnen: Eine absolute Unkenntnis δ wächst rückwärts wie $\delta\, e^{Rt/2L}$, die Vergangenheit endet bei $T^* = (2L/R)\ln(1/\delta)$,
5. … den Zweiten Hauptsatz in seiner numerischen Gestalt erklären: mikroskopisch umkehrbar, praktisch unumkehrbar — und benennen, welche drei Feinde jede Zeitumkehr begrenzen.

20.1 Ist die Energie das Primäre?

Erst der Verdacht, dann die Prüfung:

> ⚠ Naheliegende Vermutung
>
> **„Die fließende Energie ist das Eigentliche — die Felder sind nur die Größen, mit denen wir ihre Strömung beschreiben."**
> *Warum sie naheliegt:* Energie ist das, was ankommt, was man bezahlt, was die Lampe leuchten lässt. Die Felder dagegen sind unsichtbare Rechengrößen — und $\vec{S}$ aus Kapitel 18 sah verdächtig danach aus, als wäre der Energiestrom das eigentliche Geschehen.
> *Was stattdessen stimmt:* Es ist genau umgekehrt, aus vier unabhängigen, harten Gründen (unten ausgeführt): Die Felder sind der **Zustand** der Welt, die Energie ist eine — großartige, erhaltene — **Buchhaltungsgröße** über diesen Zustand. Feynman sagt es wörtlich: Die Physik weiß nicht, was Energie „ist"; sie weiß nur, dass diese eine Zahl bei allem Umpacken gleich bleibt.

Die vier Gründe, jeder für sich schon ausreichend:

1. **Zu wenig Information.** An jedem Punkt sind $(\vec{E}, \vec{B})$ sechs Zahlen, $(u, \vec{S})$ nur vier. Dreh die Polarisationsebene einer Welle (Kapitel 4): u und $\vec{S}$ ändern sich überhaupt nicht — die Felder schon, und der Unterschied ist *messbar* (jede Polfilter-Brille tut es, Kapitel 13). Aus dem Energiestrom lassen sich die Felder nicht rekonstruieren; ein Schatten kann den Körper nicht erzeugen.
2. **Die Natur addiert Felder, nicht Energien.** An den dunklen Streifen des Doppelspalts (Kapitel 14) gilt $E_1 + E_2 = 0$ — gemessen: $4 \cdot 10^{-4}$ der Einzelhelligkeit. Zwei Energieströme können sich aber nur addieren,

nie auslöschen, denn Energie hat kein Vorzeichen. Interferenz funktioniert nur, weil die *Felder* (mit Phase und Vorzeichen) die Grundgröße sind; die Energie folgt quadratisch aus ihnen.

3. **Keine geschlossene Dynamik.** Maxwell macht aus den Feldern von heute die Felder von morgen — $(\vec{E}, \vec{B})$ ist ein vollständiger Zustand, und FDTD führt das seit Kapitel 5 jeden Zeitschritt vor. Für $(u, \vec{S})$ gibt es keine solche Gleichung: Aus dem heutigen Energiefluss allein folgt der morgige nicht. Das Primäre einer Theorie muss aber das sein, was die Zukunft festlegt.
4. **Der „Fluss" hängt vom Beobachter ab.** Ein statisches Feld hat für dich $\vec{S} = 0$ — für jemanden, der vorbeifährt, nicht. Eine Substanz, deren Fließen davon abhängt, wer hinschaut, taugt nicht als Urgrund.

Die Rangordnung der Theorie läuft also in eine Richtung: **Quellen** (Ladungen, Ströme) erzeugen **Felder** (den Zustand), und aus den Feldern folgt die **Energie samt Bilanz**. Zwei Körner Wahrheit stecken trotzdem in der Intuition: Das Wort „Anregung" ist goldrichtig — nur das Substrat ist ein anderes, als der Verdacht meinte (in der Quantenfeldtheorie ist ein Photon eine Anregung des *Feldes*, und hf ist eine Eigenschaft dieser Anregung, Kapitel 30). Und es gibt eine Theorie, in der Energie- und Impulsströme wirklich etwas erzeugen — die Gravitation. Nur die elektromagnetischen Felder erzeugen sie nicht; *deren* Quellen sind Ladungen und Ströme.

20.2 Gleichungen ohne Richtungspfeil

Nun zur Zeit. Nimm die beiden 1D-Gleichungen, mit denen alles begann (Kapitel 5):

$$\frac{\partial B}{\partial t} = -\frac{\partial E}{\partial x}, \qquad \frac{\partial E}{\partial t} = -c^2 \frac{\partial B}{\partial x}.$$

Lass den Film rückwärts laufen, das heißt: Ersetze $t \to -t$. Jede Zeitableitung wechselt das Vorzeichen, die Ortsableitungen bleiben — beide Gleichungen scheinen zu kippen. Aber jetzt der Trick: Drehe *zusätzlich das Magnetfeld um*, $B \to -B$ (und lass E, wie es ist). In der ersten Gleichung kippen dann *beide* Seiten (links das B, rechts nichts — aber die Zeitableitung hatte schon gekippt), in der zweiten ebenfalls (links nur die Zeitableitung, rechts das B): Beide Vorzeichenwechsel heben sich paarweise auf, und **der rückwärts laufende Film mit umgedrehtem Magnetfeld erfüllt exakt dieselben Gleichungen**. Das gilt genauso in 3D für die vollen Maxwell-Gleichungen. Physikalisch ist das B-Umdrehen plausibel: B entsteht aus

Strömen (Kapitel 3), und im Rückwärtsfilm fließen alle Ströme andersherum. Zwei Größen aus Kapitel 18 sortieren sich dann von selbst: Der Poynting-Vektor $\vec{S} = \vec{E} \times \vec{H}$ dreht um (alle Energieströme fließen rückwärts — gut so), die Energiedichte $u \propto E^2 + B^2$ bleibt unverändert (Quadrate vergessen Vorzeichen).

Wenn aber die einlaufende, sich auf die Quelle zusammenziehende Welle genauso „legal" ist wie die auslaufende — warum sehen wir draußen immer nur die auslaufende? An dieser Stelle wird oft der Fehler gemacht, die Antwort in der Gleichung zu suchen. **Gleichungen sind Relationen ohne Richtungspfeil:** $\nabla \times \vec{H} = \vec{J} + \partial \vec{D}/\partial t$ sagt nicht, wer wen verursacht — nur, dass beide Seiten zu jeder Zeit gleich sind. Die Kausalität ist eine **zusätzliche physikalische Aussage**, und sie steckt in der *Wahl der Lösungen*: Von den mathematisch erlaubten realisiert die Natur die **retardierten** — die Felder *jetzt* sind durch Ladungen und Ströme zu *früheren* Zeiten bestimmt, verzögert um die Lichtlaufzeit (explizit ausgeschrieben heißen diese Lösungen Jefimenko-Gleichungen). Die einlaufenden Geschwister, die *avancierten* Lösungen, sind nicht verboten — sie verlangen nur absurd verabredete Anfangsbedingungen: Ein perfekt auf die Quelle zulaufender Wellenring müsste am Anfang der Zeit überall draußen exakt präpariert worden sein.

Und unser Handwerkszeug? **FDTD ist genau diese Sichtweise als Algorithmus.** Anfangszustand plus Zeitschritte: Die Zukunft wird Schritt für Schritt aus der Vergangenheit berechnet, nie umgekehrt — die Kausalität steckt bei uns nicht in den Update-Formeln (die sind so richtungslos wie Maxwell selbst, gleich beweisen wir es), sondern in der Entscheidung, bei $t = 0$ mit einem *bekannten Anfang* loszurechnen.

20.3 Der Film läuft rückwärts: das 1D-Labor

Die Behauptung „richtungslos" ist überprüfbar, und zwar wörtlich. Wir nehmen das älteste Experiment des Buchs — den Gauß-Puls aus Kapitel 5, der sich in zwei Hälften teilt — und drehen mitten im Lauf das Magnetfeld um. Kein Code läuft „rückwärts", wir rechnen stur vorwärts weiter; nur B bekommt ein Minuszeichen. Vorher du:

Vorhersage-Punkt

Der Puls ist nach 120 Schritten längst in zwei auseinanderlaufende Hälften zerfallen (Kapitel 5). Jetzt ersetzen wir $B \to -B$ und machen 120 ganz normale Leapfrog-Schritte weiter. Was passiert: Laufen die Hälften weiter auseinander? Frieren sie ein? Oder laufen sie zurück — und

> falls ja, wie exakt setzt sich der Anfangspuls wieder zusammen?

Das Experiment dazu. **Frage:** Ist der diskrete Leapfrog zeitumkehrinvariant wie seine Gleichungen? **Bühne:** das 1D-Gitter aus Kapitel 5 (400 Zellen, $S = 1$, PEC-Ränder, in Gittereinheiten $c = 1$, $\Delta x = \Delta t = 1$). **Anfangsbedingungen:** Gauß-Puls in E, $B = 0$ — er teilt sich in zwei Hälften (das wissen wir seit Kapitel 5). **Eingriff:** nach 120 Schritten $B \to -B$; danach 120 weitere *Vorwärts*-Schritte, allerdings mit getauschter Update-Reihenfolge (erst E, dann B — warum, klärt der Text nach dem Bild). **Messgröße:** $\max|E - E_0|$ am Ende. **Erfolgskriterium:** Maschinengenauigkeit.

```
import numpy as np
import matplotlib.pyplot as plt

N1 = 400
X1 = np.arange(N1)
E = np.exp(-((X1 - 200) / 12.0) ** 2)
E0 = E.copy()
B = np.zeros(N1 - 1)

for k in range(120):                    # vorwärts: erst B, dann E
    B += np.diff(E)
    E[1:-1] += np.diff(B)
E_mitte = E.copy()

B *= -1                                 # ZEITUMKEHR: B -> -B, E bleibt
for k in range(120):                    # weiter vorwärts, Reihenfolge getauscht
    E[1:-1] += np.diff(B)
    B += np.diff(E)

fig, axes = plt.subplots(1, 3, figsize=(9, 3.0), sharey=True)
for ax, feld, titel in ((axes[0], E0, "$t = 0$: der Anfangspuls"),
                        (axes[1], E_mitte, "$t = 120$: zweigeteilt,\njetzt $B \\to -B$"),
                        (axes[2], E, "$t = 240$: wieder beisammen")):
    ax.plot(X1, feld, color="royalblue", lw=1.5)
    ax.set_title(titel, fontsize=10)
    ax.set_xlabel("Zelle")
axes[2].plot(X1, E0, ":", color="black", lw=1.5, label="Anfangspuls $E_0$")
axes[2].legend(fontsize=8)
axes[0].set_ylabel("E")
plt.show()
```

```
print(f"max|E - E0| nach Hin- und Rückfilm: {np.abs(E - E0).max():.1e}")
```

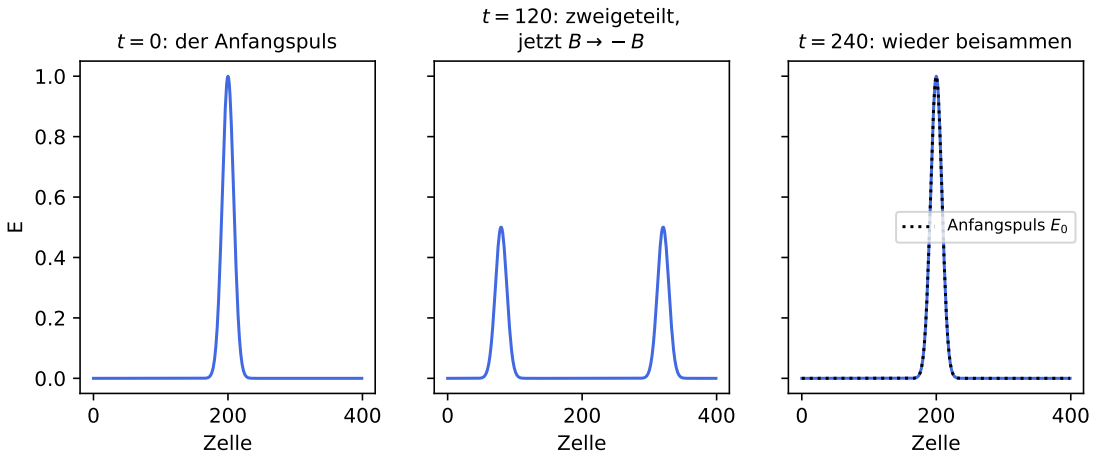


Abbildung 20.1: Zeitumkehr als Symmetrie, kein Trick: Links der Anfangspuls, Mitte der Zustand nach 120 Schritten (zwei Hälften, halbe Höhe — der Moment, in dem B umgedreht wird), rechts der Zustand nach 120 weiteren ganz normalen Vorwärts-Schritten (durchgezogen) über dem Anfangspuls (gepunktet): deckungsgleich auf $2{,}5 \cdot 10^{-16}$. Der Leapfrog kennt keine Zeitrichtung — mit umgedrehtem B ist der Rückwärtsfilm einfach eine weitere Lösung.

```
max|E - E0| nach Hin- und Rückfilm: 2.5e-16
```

Die Hälften kehren um, laufen aufeinander zu und setzen sich zum Original zusammen — auf sechzehn Stellen. Es lohnt sich, kurz innezuhalten: Wir haben *nichts invertiert*, keinen einzigen Schritt zurückgenommen. Der Rückwärtsfilm ist selbst ein stinknormaler Vorwärtslauf, nur mit gespiegeltem Magnetfeld — die Symmetrie der Gleichungen, vom Gitter geerbt. Das einzige Zugeständnis ans Diskrete ist die getauschte Update-Reihenfolge, und sie ist ein alter Bekannter: B lebt auf *halben* Zeitschritten (Kapitel 5). Der Zustand „E jetzt, B eine halbe Stufe *später*" wird unter Zeitumkehr zum Zustand „E jetzt, B eine halbe Stufe *früher*" — und ein Lauf, der mit dem E-Update beginnt, ist exakt der Leapfrog für diese gespiegelte Staffelung. (Wer den Tausch weglässt, bekommt den Anfangspuls trotzdem zurück, nur um einen Zeitschritt verrutscht — ein Fehler in der zweiten Nachkommastelle statt der sechzehnten. Der Yee-Versatz hat damit seinen vierten Auftritt im Buch.)

20.4 Der gedämpfte Kreis rückwärts: eine Überraschung

Verlustfrei ist die Zeitumkehr also makellos. Jetzt das Gegenteil von makellos: der gedämpfte Schwingkreis aus Kapitel 19, dessen Energie unwiederbringlich zu Wärme wurde — das Paradebeispiel eines Vorgangs mit Zeitrichtung. Sein B-Flip-Double gibt es nicht: Ein Medium mit Verlusten bleibt unter Zeitumkehr ein Medium mit Verlusten, der zurücklaufende Film zeigt eine Schwingung, die von selbst *anwächst* — dafür müsste der Widerstand zum **Verstärker** werden. Genau das können wir im Computer aber erzwingen, indem wir die Leapfrog-Schritte *algebraisch auflösen*: Aus $i_{\text{neu}} = c_a\, i_{\text{alt}} - c_b\, q/C$ wird rückwärts $i_{\text{alt}} = (i_{\text{neu}} + c_b\, q/C)/c_a$ — eine **Division durch** $c_a < 1$, der Verstärker in einer Codezeile. Jeder Rückwärtsschritt multipliziert dabei auch jeden Fehler mit $1/c_a$. Über die Laufzeit T summiert sich das zu $e^{RT/2L}$ — bei $T = 700$ (das sind 111 Schwingungen, die Amplitude ist auf $e^{-42} \approx 10^{-19}$ gefallen) zu einem Verstärkungsfaktor von 10^{18}. Vorher du:

> **Vorhersage-Punkt**
>
> Wir lassen den Kreis ($R = 0{,}12$, also $Q = 8{,}3$) bis $T = 700$ abklingen und lösen dann alle 70 000 Schritte rückwärts auf. Die Rundungsfehler des Computers ($\sim 10^{-16}$) werden dabei um $e^{RT/2L} \approx 10^{18}$ verstärkt. Kommt der Anfangszustand $q_0 = 1$ wieder heraus — oder Rauschen von der Größe 10^2?

Die Rechnung ist drei Zeilen Code, und ihr Ergebnis hat uns beim Schreiben dieses Kapitels selbst überrascht:

```
# von oben: np
L, C, R, DT0 = 1.0, 1.0, 0.12, 0.01

def kreis_hin_zurueck(R_, T, delta=0.0):
    """Kapitel-19-Leapfrog vorwärts, Störung δ am Umkehrpunkt,
    dann alle Schritte algebraisch exakt rückwärts auflösen."""
    s = R_ * DT0 / (2 * L)
    ca, cb = (1 - s) / (1 + s), (DT0 / L) / (1 + s)
    q, i = 1.0, 0.0
    for k in range(int(T / DT0)):     # vorwärts (Kapitel 19)
        i = ca * i - cb * q / C
        q = q + DT0 * i
    q += delta                        # absolute Unkenntnis (erstmal 0)
    for k in range(int(T / DT0)):     # rückwärts auflösen
```

```
        q = q - DT0 * i
        i = (i + cb * q / C) / ca     # Division durch ca: der Verstärker
    return abs(q - 1.0)               # Abstand zum wahren Anfang

print(f"verlustfrei, T = 700: Rest {kreis_hin_zurueck(0.0, 700.0):.1e}")
print(f"GEDÄMPFT,    T = 700: Rest {kreis_hin_zurueck(R, 700.0):.1e}")
```

```
verlustfrei, T = 700: Rest 7.2e-15
GEDÄMPFT,    T = 700: Rest 6.9e-15
```

Die Rückrechnung klappt. Trotz Dämpfung, trotz 10^{18}-facher Fehlerverstärkung steht der Anfangszustand wieder da, auf vierzehn Stellen. Wie kann das sein? Weil die Verstärkung nicht auf festes Rauschen wirkt, sondern auf die Rundungsfehler des Gleitkomma-Formats — und das speichert *relativ* genau: Auch die auf 10^{-19} abgeklungene Schwingung trägt noch ihre vollen sechzehn gültigen Ziffern (ein `float` reicht hinunter bis 10^{-308}). Der Rundungsfehler ist an jeder Stelle $\sim 10^{-16}$ *der aktuellen Amplitude*; rückwärts wächst er genau in dem Maß, in dem die Amplitude selbst wieder anwächst — die relative Genauigkeit bleibt konstant. Die Lektion ist fundamental: **Dämpfung vernichtet keine Information.** Sie macht das Signal nur *leise*. Wer beliebig genau messen kann, kann die Vergangenheit eines gedämpften Systems vollständig rekonstruieren.

Niemand kann beliebig genau messen. Und damit kommt der eigentliche Gegenspieler ins Spiel: nicht der Verlust, sondern die **Unkenntnis**. Wir stellen sie mit einem einzigen Eingriff nach — am Umkehrpunkt addieren wir auf q ein winziges, *absolutes* $\delta = 10^{-15}$ (ein Staubkorn: als hätten wir den Endzustand „nur" auf fünfzehn Stellen gemessen) und lösen dann wie eben exakt rückwärts auf. Dieses δ ist von der Schwingung entkoppelt, es klingt rückwärts nicht ab — es wird nur verstärkt, pro Zeitspanne t um $e^{Rt/2L}$. Die Vorhersage ist also messerscharf: Rest $\approx \delta\, e^{RT/2L}$, und die Vergangenheit endet am **Informations-Horizont**

$$T^* = \frac{2L}{R} \ln \frac{1}{\delta} \qquad \text{(hier: } T^* = \tfrac{2}{0{,}12} \ln 10^{15} \approx 576\text{).}$$

```
# von oben: np, plt, kreis_hin_zurueck, L, R

DELTA = 1e-15
T_reihe = np.array([100, 200, 300, 400, 500, 600], float)
rest = np.array([kreis_hin_zurueck(R, T, DELTA) for T in T_reihe])
T_stern = 2 * L / R * np.log(1 / DELTA)
```

```
fig, ax = plt.subplots(figsize=(7, 3.4))
ax.semilogy(T_reihe, rest, "o", color="crimson", ms=7, label="gemessen")
tt = np.linspace(50, 650, 200)
ax.semilogy(tt, DELTA * np.exp(R * tt / (2 * L)), "k--",
            label="Theorie $\\delta\\,e^{RT/2L}$")
ax.axvline(T_stern, color="0.5", ls=":", label="$T^* = (2L/R)\\ln(1/\\delta)$")
ax.set_xlabel("Laufzeit T bis zur Umkehr")
ax.set_ylabel("Rest bei $t = 0$")
ax.legend(fontsize=9)
plt.show()

for T, rst in zip(T_reihe, rest):
    print(f"T = {T:3.0f}: Rest {rst:.2e}   "
          f"(Theorie {DELTA * np.exp(R * T / (2 * L)):.2e})")
```

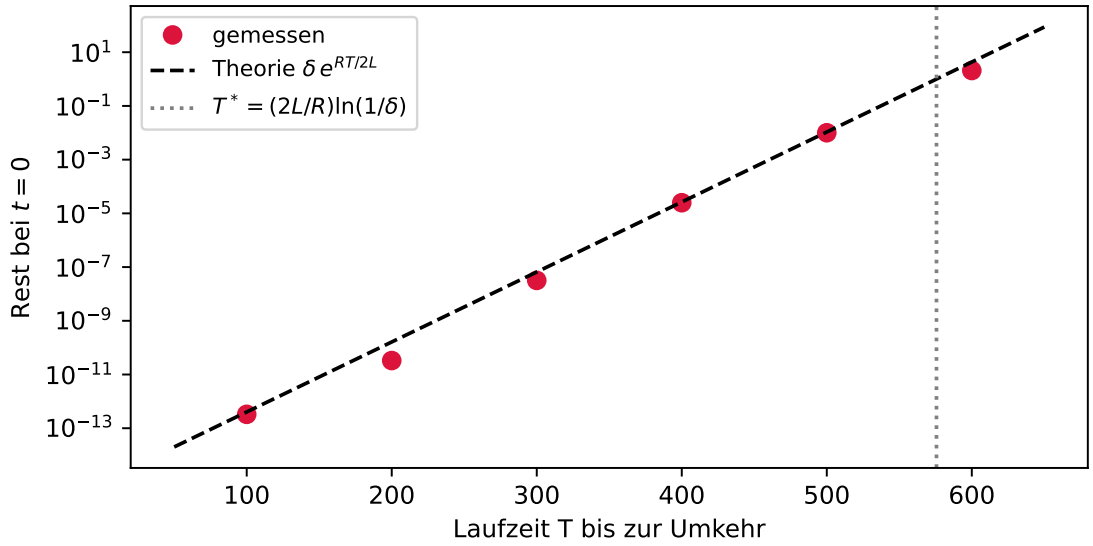


Abbildung 20.2: Der Informations-Horizont: Rest der Rückrechnung bei t = 0, wenn der Umkehrzustand um ein absolutes δ = 10^{-15} gestört wurde, über der Laufzeit T (Punkte), dazu die parameterfreie Theoriegerade δ·e^(RT/2L). Die Steigung stimmt exakt — jede Zeiteinheit Dämpfung verstärkt die Unkenntnis rückwärts um denselben Faktor, um den sie das Signal vorwärts leiser macht. An der gepunkteten Linie (T* ≈ 576, das sind 92 Schwingungen) erreicht der Fehler die Größe des Signals selbst: Dahinter ist die Vergangenheit nicht mehr rekonstruierbar — nicht weil die Gleichungen es verbieten, sondern weil unser Wissen endet.

```
T = 100: Rest 3.29e-13   (Theorie 4.03e-13)
T = 200: Rest 3.31e-11   (Theorie 1.63e-10)
T = 300: Rest 3.19e-08   (Theorie 6.57e-08)
T = 400: Rest 2.49e-05   (Theorie 2.65e-05)
T = 500: Rest 1.00e-02   (Theorie 1.07e-02)
T = 600: Rest 2.08e+00   (Theorie 4.31e+00)
```

Die Punkte liegen auf der Theoriegeraden (der Vorfaktor wackelt um einen Faktor 2 — er hängt davon ab, in welche Schwingungsphase das δ fällt; die *Steigung* ist exakt). Bei $T = 600$, knapp hinter dem Horizont, ist der „rekonstruierte Anfang" um den Faktor 2 falsch — Schrott. Das Bemerkenswerte an der Horizont-Formel: δ steht im *Logarithmus*. Zehntausendmal genauer messen schiebt den Horizont nur um $(2L/R)\ln 10^4$, also gut ein Drittel, hinaus — die Vergangenheit gedämpfter Systeme ist exponentiell teuer. Genau dieselbe Mathematik begrenzt übrigens die Wettervorhersage (dort wächst die Unkenntnis *vorwärts*, weil die Strömung chaotisch ist) — und Ü 20.2 rechnet aus, was ein realistisches Messgerät mit sechs Stellen vom Horizont übrig lässt.

20.5 2D: der Absorber gewinnt

Jetzt dasselbe Spiel auf dem Yee-Gitter aus Kapitel 9 — und damit die Begegnung mit dem Zweiten Hauptsatz in seiner ehrlichsten Form. Vorab die Vermutung, die sich nach dem 0D-Erfolg geradezu aufdrängt:

⚠ Naheliegende Vermutung

„Verluste machen die Gleichungen unumkehrbar — deshalb scheitert die Zeitumkehr an einem Absorber."
Warum sie naheliegt: Wärme kommt nicht zurück; was der Absorber geschluckt hat, ist weg. Irgendwo muss die Unumkehrbarkeit ja in der Mathematik stecken.
Was stattdessen stimmt: Die Gleichungen — auch die mit Verlusten — bleiben **algebraisch exakt umkehrbar**, und der gedämpfte Schwingkreis hat es eben sogar numerisch vorgeführt. Was die Rückrechnung im großen Feld zerstört, ist nicht die Algebra, sondern die **Information**: Der Absorber sitzt in einem Meer aus Feldwerten voller Größe, deren Rundungsreste ($\sim 10^{-16}$ *der großen Nachbarwerte*) bei jedem Schritt über die Rotation in ihn hineinmischen. Das ist genau das absolute δ von eben — nur dass es niemand von Hand einstreuen muss: **Die vielen Freiheitsgrade der Umgebung liefern die Unkenntnis**

> **frei Haus.** Mikroskopisch umkehrbar, praktisch unumkehrbar — der Zweite Hauptsatz handelt nicht von Gleichungen, sondern von Wissen.

Das Experiment, in zwei Läufen. **Frage:** Lässt sich eine 2D-Welle zur Quelle zurückrechnen — und was ändert ein Absorber daran? **Bühne:** das 300×300-Gitter aus Kapitel 9 ($S = 0{,}5 < 1/\sqrt{2}$, PEC-Ränder, Gittereinheiten); in Lauf B steht rechts der Quelle ein Absorberbalken (16×180 Zellen, $c_a = 0{,}8$ — pro Rückwärtsschritt also Faktor $1{,}25$). **Ablauf:** Eine Punktquelle in der Mitte sendet einen Wellenzug ($\lambda = 20$ Zellen, 260 Schritte vorwärts); dann werden alle Schritte algebraisch rückwärts aufgelöst, die Quelle wird dabei zeitrichtig wieder *abgezogen*. Unterwegs vergleichen wir mit gemerkten Zwischenständen des Hinwegs. **Messgrößen:** das Restfeld bei $t = 0$ (Soll: leer!) und die Fehlerkurve über den Rückwärtsschritten. **Erfolgskriterium:** Lauf A Maschinenrest; Lauf B sagt der Kasten oben voraus.

```
# von oben: np, plt

NX = NY = 300
DT = 0.5
N_SCHRITTE = 260
QX, QY = NX // 2, NY // 2
PERIODE, N0, BREITE = 40, 50, 14          # Wellenlänge λ = 20 Zellen
AX0, AX1, AY0, AY1 = 200, 216, 60, 240    # der Absorberbalken (Lauf B)

def quellsignal(n):
    return (np.sin(2 * np.pi * n / PERIODE)
            * np.exp(-(((n - N0) / BREITE) ** 2)))

def rot_h(Hx, Hy):
    rot = np.zeros((NX, NY))
    rot[1:-1, 1:-1] = (Hy[1:-1, 1:-1] - Hy[:-2, 1:-1]) - \
                      (Hx[1:-1, 1:-1] - Hx[1:-1, :-2])
    return rot

def hin_und_zurueck_2d(V):
    ca = np.ones((NX, NY))
    cb = np.full((NX, NY), DT)
    if V > 0:
        ca[AX0:AX1, AY0:AY1] = (1 - V) / (1 + V)
        cb[AX0:AX1, AY0:AY1] = DT / (1 + V)
    Ez = np.zeros((NX, NY)); Hx = np.zeros((NX, NY)); Hy = np.zeros((NX, NY))
    checks = {}
    for n in range(N_SCHRITTE):               # ---- vorwärts ----
```

```
        Hx[:, :-1] -= DT * (Ez[:, 1:] - Ez[:, :-1])
        Hy[:-1, :] += DT * (Ez[1:, :] - Ez[:-1, :])
        Ez[:] = ca * Ez + cb * rot_h(Hx, Hy)
        Ez[QX, QY] += quellsignal(n)
        if n % 10 == 0:
            checks[n] = Ez.copy()             # Zwischenstände merken
    bild_ende = Ez.copy()
    f_t, f_w = [], []
    for n in range(N_SCHRITTE - 1, -1, -1):   # ---- rückwärts auflösen ----
        Ez[QX, QY] -= quellsignal(n)          # Quelle zeitrichtig abziehen
        Ez[:] = (Ez - cb * rot_h(Hx, Hy)) / ca
        Hx[:, :-1] += DT * (Ez[:, 1:] - Ez[:, :-1])
        Hy[:-1, :] -= DT * (Ez[1:, :] - Ez[:-1, :])
        if (n - 1) in checks:
            f_t.append(n - 1)
            f_w.append(np.max(np.abs(Ez - checks[n - 1])))
    return (np.max(np.abs(Ez)), np.array(f_t[::-1]),
            np.array(f_w[::-1]), bild_ende, Ez.copy())

rest_a, ft_a, fw_a, bild_a, anfang_a = hin_und_zurueck_2d(0.0)
rest_b, ft_b, fw_b, bild_b, anfang_b = hin_und_zurueck_2d(0.111)

fig, ax = plt.subplots(2, 2, figsize=(9, 8.4))
VMAX = 0.06                               # eng, damit der Ring satt druckt
def zeig(a, feld, titel, skala=VMAX, balken=False):
    a.imshow(feld.T, origin="lower", cmap="RdBu_r", vmin=-skala, vmax=skala)
    a.plot(QX, QY, "k+", ms=8)
    a.set_title(titel, fontsize=10)
    a.set_xticks([]); a.set_yticks([])
    if balken:
        a.plot([AX0, AX1, AX1, AX0, AX0], [AY0, AY0, AY1, AY1, AY0],
               color="green", lw=1.5)

zeig(ax[0, 0], bild_a, "A vorwärts, $t = T$: Welle ausgelaufen")
zeig(ax[0, 1], anfang_a, f"A rückwärts bei $t = 0$: leer\n(Rest {rest_a:.0e})")
zeig(ax[1, 0], bild_b, "B vorwärts, $t = T$: Balken wirft Schatten",
     balken=True)
zeig(ax[1, 1], anfang_b, f"B rückwärts bei $t = 0$: Rauschexplosion\n"
     f"(max {rest_b:.0f} — eigene Skala!)", skala=rest_b, balken=True)
plt.show()

print(f"Lauf A: Restfeld {rest_a:.1e}   Lauf B: Restfeld {rest_b:.1e}")
```

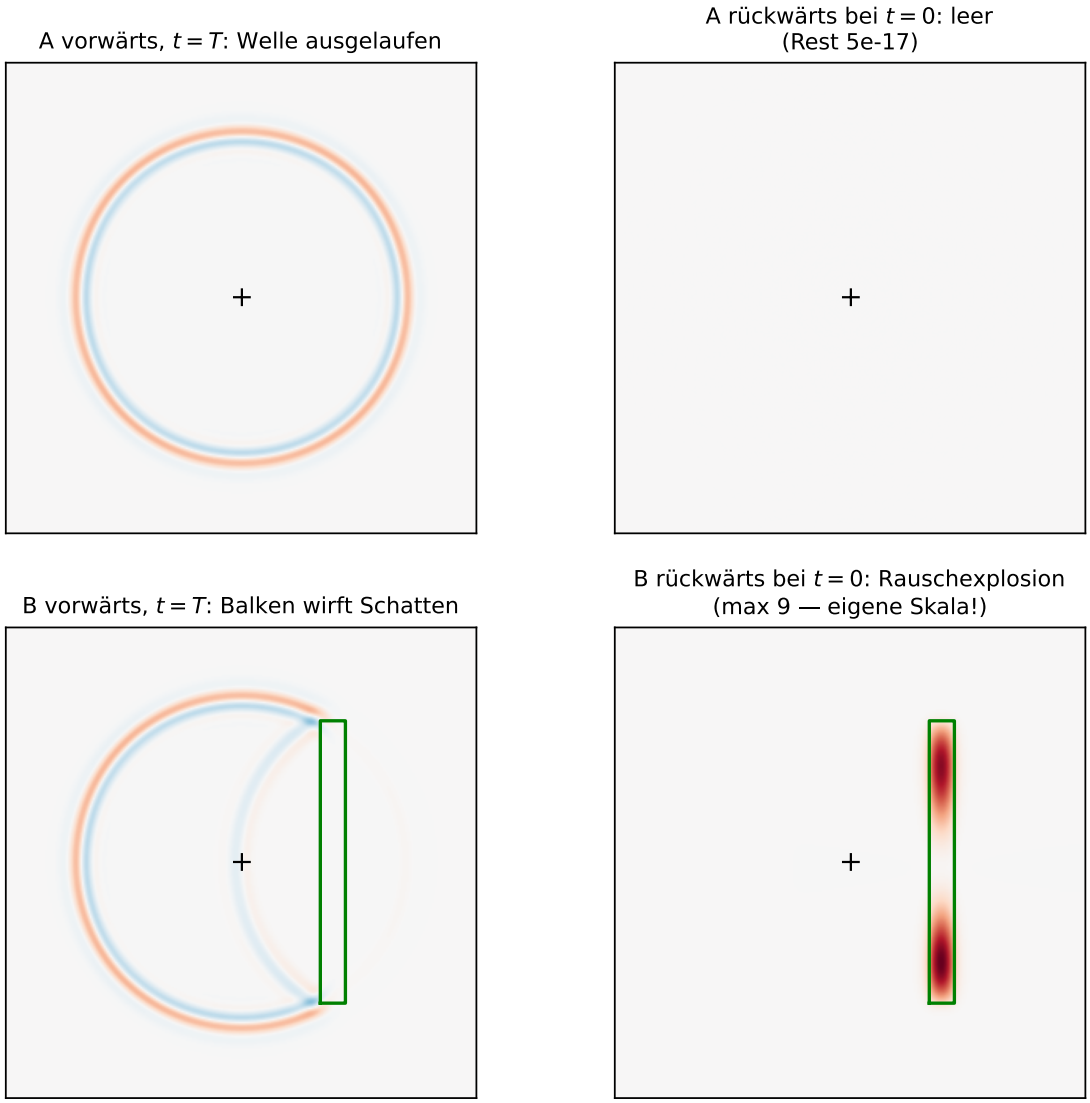


Abbildung 20.3: Zeitumkehr in der Fläche. Oben Lauf A (verlustfrei): Der ausgelaufene Wellenring (links) läuft rückwärts vollständig in die Quelle zurück — bei t = 0 (rechts) ist das Gitter bis auf $5{\cdot}10^{-17}$ leer, das Bild zeigt buchstäblich nichts. Unten Lauf B (Absorberbalken, grün): Vorwärts schluckt der Balken einen Teil des Rings (links, der Schatten rechts vom Balken); rückwärts explodiert das Rauschen im Balken, und der „rekonstruierte Anfang“ (rechts, eigene Farbskala!) ist ein Artefakt von der Größe 9 — achtzigmal lauter als die ursprüngliche Welle und ohne jede Ähnlichkeit mit dem wahren Anfang (leeres Gitter).

```
Lauf A: Restfeld 4.7e-17   Lauf B: Restfeld 9.2e+00
```

Lauf A ist die Flächen-Fassung des B-Flip-Wunders: 260 Schritte hin, 260 algebraisch zurück, und das Gitter ist wieder so leer, wie es angefangen hat — der Wellenzug verschwindet rückstandsfrei in der Quelle (dass das Restfeld sogar *kleiner* als beim Schwingkreis ausfällt, liegt daran, dass im Vakuum-Teil gar nicht dividiert wird: $c_a = 1$).

Lauf B dagegen liefert als „Anfang" ein Feld, das es nie gab: laut, auf den Balken konzentriert, physikalisch bedeutungslos. Die Fehlerkurve zeigt, wie es dazu kommt:

```
# von oben: np, plt, ft_a, fw_a, ft_b, fw_b, N_SCHRITTE

rs_a = N_SCHRITTE - 1 - ft_a              # Nr. des Rückwärtsschritts
rs_b = N_SCHRITTE - 1 - ft_b

fig, ax = plt.subplots(figsize=(7, 3.4))
ax.semilogy(rs_a, np.maximum(fw_a, 1e-18), "o-", ms=4,
            color="royalblue", label="A verlustfrei")
ax.semilogy(rs_b, np.maximum(fw_b, 1e-18), "s-", ms=4,
            color="crimson", label="B mit Absorber")
ax.semilogy(rs_b, 1e-17 * (1.25) ** rs_b, "k--", lw=1.2,
            label="Maximalsteigung $(1/c_a)^n$")
ax.set_xlabel("Rückwärtsschritt n")
ax.set_ylabel("max. Fehler")
ax.legend(fontsize=9)
plt.show()

raten = np.polyfit(rs_b[2:], np.log(fw_b[2:]), 1)[0]
print(f"Lauf B: gemessene Fehlerrate {raten:.3f} pro Schritt "
      f"(Maximalsteigung ln(1/ca) = {np.log(1.25):.3f})")
```

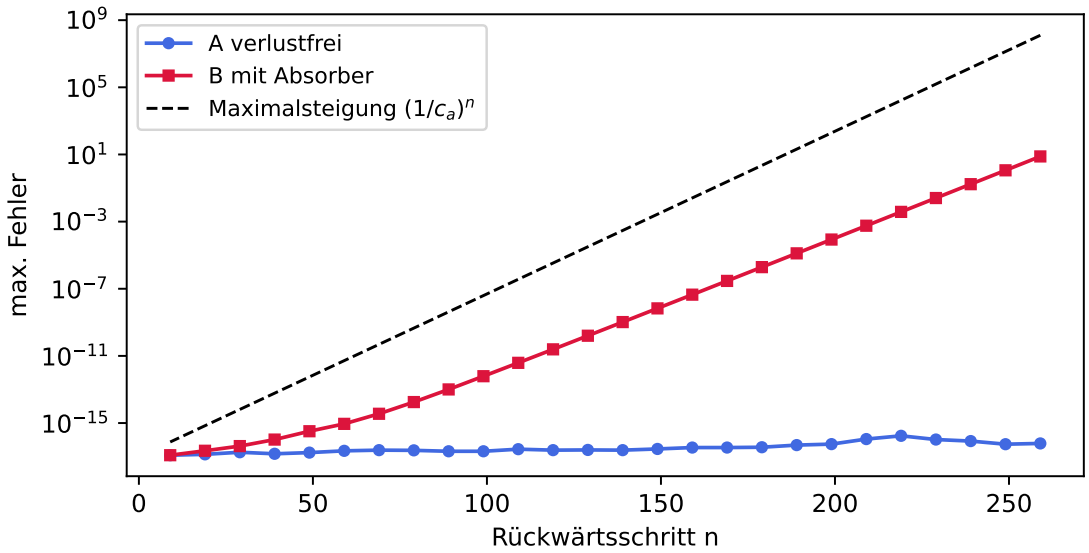


Abbildung 20.4: Der Rekonstruktionsfehler (Abstand zu den gemerkten Zwischenständen des Hinwegs) über den Rückwärtsschritten. Lauf A (blau) bleibt auf Maschinenniveau — die Zeitumkehr ist exakt. In Lauf B (rot) wächst der Fehler exponentiell; die gestrichelte Gerade ist die lokale Maximalsteigung $(1/c_a)^n = 1{,}25^n$ des Absorbers. Die gemessene Steigung bleibt darunter (Faktor e^0,17 statt e^0,22 pro Schritt), weil das verstärkte Rauschen ständig aus dem Balken herausstrahlt und im Vakuum nicht weiterwächst — mit stärkerem Absorber rückt die Messung an die Grenze heran (Ü 20.3).

```
Lauf B: gemessene Fehlerrate 0.170 pro Schritt (Maximalsteigung ln(1/ca) = 0.223)
```

Das ist der **Zweite Hauptsatz in Zahlen**. Vorwärts schiebt der Absorber die Feldinformation in Freiheitsgrade, die unser Gitter gar nicht mitführt (echte Wärme: ungezählte Molekülbewegungen — unser σ ist ja nur die Pauschalbuchung dafür, Kapitel 6); zurückbleiben Zahlen, deren letzte Stellen von den groben Nachbarn verrauscht sind. Und rückwärts müsste genau aus diesen letzten Stellen das Verschluckte wiederauferstehen — die Division durch c_a versucht es redlich und verstärkt eben alles, Signal wie Unkenntnis. Mikroskopisch ist jede Kollision umkehrbar; praktisch wächst die kleinste Unkenntnis exponentiell, bis sie das Signal frisst. Boltzmanns Antwort auf das Umkehr-Paradoxon, auf einem 300×300-Gitter nachgestellt.

Damit lassen sich die **drei Feinde der Zeitumkehr** benennen, denen jedes reale Rückwärtsrechnen begegnet:

1. **Verluste** — eben vorgeführt: nicht weil sie die Gleichungen brechen, sondern weil sie Information in unzugängliche Freiheitsgrade verschieben und die Rückverstärkung jede Unkenntnis explodieren lässt.
2. **Die PML** (Kapitel 9) — eine *eingebaute* Informationsvernichtungsmaschine: Was sie geschluckt hat, ist weg; rückwärts müsste sie eine perfekt abgestimmte Verstärkerschicht sein, und ihr Rauschen explodiert wie eben der Balken, nur um Größenordnungen heftiger.
3. **Evaneszente Wellen** (Kapitel 12) — Details der Quelle, die feiner sind als etwa $\lambda/2$, klingen exponentiell ab und sind schon wenige Wellenlängen entfernt unlesbar: Die Vergangenheit ist aus der Ferne nur **beugungsbegrenzt scharf** rekonstruierbar. (Ü 20.4 misst genau diese Grenze.)

20.6 Zeitumkehr als Technik

Wie viel muss man eigentlich wissen, um die Vergangenheit zu rechnen? Weniger, als man denkt — aber vollständig: Maxwell ist von *erster* Ordnung in der Zeit, also legt **ein einziger kompletter Schnappschuss** ($\vec{E}$ und $\vec{H}$ überall im Raum zu *einem* Zeitpunkt) die gesamte Vergangenheit und Zukunft fest — keine Anfangsgeschwindigkeiten, kein Anlauf (genau deshalb genügte unseren Simulationen seit Kapitel 5 ein Anfangszustand). Alternativ genügt das Feld auf einer **geschlossenen Fläche, über die Zeit aufgezeichnet** — man muss nicht überall messen, nur lückenlos um das Geschehen herum und lange genug.

Diese zweite Variante ist keine Theorie-Fußnote, sondern Alltagstechnik: Der **Zeitumkehrspiegel** (Mathias Fink, ab den 1990ern) zeichnet eine Welle auf einem Wandlerring auf und sendet sie zeitgespiegelt wieder aus — worauf sie sich, durch alle Streuung hindurch, von selbst auf ihre ursprüngliche Quelle fokussiert. So zertrümmert man Nierensteine durch inhomogenes Gewebe und schickt Ultraschall durch den Schädel; die Seismologie lokalisiert Bebenherde, indem sie die Signale aller Stationen rückwärts in ein Erdmodell propagiert; und Meeps Adjoint-Löser (Kapitel 31) macht aus der Rückwärts-Simulation ein Entwurfswerkzeug. In Ü 20.4 baust du den Spiegel selbst — 72 Aufnahmepunkte auf einem Ring genügen, und der Fokus wird so scharf, wie Feind Nummer drei es erlaubt: eine halbe Wellenlänge.

Ein Faden bleibt für später gespannt: Wenn Kausalität bedeutet, dass die Antwort eines Materials nie *vor* der Ursache kommen darf, dann hat das überraschende Konsequenzen für ganz gewöhnliche Optik — aus „keine Wirkung vor der Ursache” folgt zwingend, dass jedes durchsichtige Material Dispersion *haben muss*. Diese Brücke (sie heißt Kramers-Kronig) schlägt Kapitel 29.

Bevor das Kapitel-Programm alles bündelt, der Teil-V-Bogen in einem Satz: Felder sind der Zustand, Energie ist seine Buchhaltung, und die Richtung der Zeit steht auf dem Konto der Buchhalterin — sie misst, was wir noch wissen.

Merkkasten

- **Felder sind der Zustand, Energie ist Buchhaltung** — vier Gründe: zu wenig Information (Polarisation!), Felder addieren sich (Interferenz!), keine eigene Dynamik, beobachterabhängig.
- **Maxwell hat keine Zeitrichtung**: $t \to -t$ und $\vec{B} \to -\vec{B}$ ist wieder eine Lösung ($\vec{S}$ dreht um, u bleibt). Kausalität = Wahl der *retardierten* Lösungen — FDTD ist diese Wahl als Algorithmus.
- **Dämpfung vernichtet keine Information**, sie macht das Signal leise: Der gedämpfte Kreis ließ sich über 111 Schwingungen exakt zurückrechnen.
- Eine **absolute** Unkenntnis δ wächst rückwärts wie $\delta\, e^{Rt/2L}$ → Informations-Horizont $T^* = (2L/R)\ln(1/\delta)$. In ausgedehnten Systemen liefert die Umgebung das δ von selbst — das ist der Zweite Hauptsatz: **Der Zeitpfeil steckt darin, was wir wissen können.**
- Ein **Schnappschuss von $\vec{E}$ und $\vec{H}$** legt alles fest (Maxwell ist 1. Ordnung); ein aufgezeichneter Rand genügt auch → Zeitumkehrspiegel, beugungsbegrenzt auf $\sim \lambda/2$.

20.7 Das Kapitel-Programm

Das Programm `kap20_zeitumkehr.py` (im Buchpaket unter `programme/kap20/`) bündelt alles: den 1D-B-Flip, den gedämpften Kreis rückwärts (mit und ohne Störung, samt Horizont-Reihe), die beiden 2D-Läufe mit Fehlerkurven, die Absorber-Stärke-Reihe aus Ü 20.3 und den Zeitumkehrspiegel aus Ü 20.4. Jede Behauptung steht als `assert`; der Lauf braucht etwa eine Minute und endet mit „Alle Tests bestanden.”

Roter Faden

Rückwärts: Der **Puls aus Kapitel 5** hat sich wieder zusammengesetzt (und der Yee-Halbschritt hatte seinen vierten Auftritt), das **2D-Gitter aus Kapitel 9** lief erstmals rückwärts (und die PML von dort entpuppte sich als Informationsvernichter), der **evaneszente Saum aus Kapitel 12** wurde zur

Schärfegrenze der Vergangenheit, die **Interferenz aus Kapitel 14** lieferte das stärkste Argument gegen das Energie-Primat, der **gedämpfte Kreis aus Kapitel 19** bekam seine Zeitpfeil-Pointe, und der **Poynting-Vektor aus Kapitel 18** dreht unter Zeitumkehr brav um. Vorwärts: **Kapitel 21** kehrt zur Vorwärtsrichtung zurück und lässt Antennen strahlen (retardiert!), **Kapitel 29** macht aus der Kausalität ein quantitatives Werkzeug (Kramers-Kronig: Durchsichtigkeit erzwingt Dispersion), **Kapitel 30** fragt, was vom Determinismus des Schnappschusses an der Quantengrenze übrig bleibt, und **Kapitel 31** nutzt die Rückwärts-Simulation als Entwurfswerkzeug (Adjoint).

Übungen

Ü 20.1 (Verstehen). Zeitumkehr von Hand: (a) Zeige an den beiden 1D-Gleichungen explizit, dass mit $E'(x,t) = E(x,-t)$ und $B'(x,t) = -B(x,-t)$ wieder eine Lösung vorliegt (Kettenregel für die innere Ableitung $-t$ genügt). (b) Was wird unter dieser Spiegelung aus $\vec{S} = \vec{E} \times \vec{H}$ und aus u? Prüfe beides numerisch am 1D-Lauf aus Abbildung 20.1: Vergleiche $E \cdot B$ und $E^2 + c^2 B^2$ unmittelbar vor und nach dem B-Flip. (c) Welche „Filme" verraten ihre Richtung: eine ebene Welle im Vakuum, die zerspringende Tasse, der gedämpfte Schwingkreis? Und warum ist Letzterer trotzdem mikroskopisch umkehrbar?

Musterlösung zu Ü 20.1

(a) Einsetzen mit Kettenregel: $\partial_t B' = -\partial_t[B(x,-t)] = +(\partial_t B)(x,-t)$ — das innere Minus der Zeitspiegelung und das äußere Minus der B-Umkehr heben sich; ebenso in der zweiten Gleichung, wo rechts $\partial_x B' = -(\partial_x B)(x,-t)$ das Minus der links stehenden Zeitableitung von E auffängt. Beide Gleichungen bestehen. Numerisch zu (b):

```python
# von oben: np — der 1D-Aufbau aus fig-bflip, am Umkehrpunkt
E_u = np.exp(-((np.arange(400) - 200) / 12.0) ** 2)
B_u = np.zeros(399)
for k in range(120):
    B_u += np.diff(E_u)
    E_u[1:-1] += np.diff(B_u)
S_vor = E_u[:-1] * B_u                  # ~ Poynting (c = 1)
u_vor = E_u**2 + np.concatenate([B_u, [0]])**2
B_u *= -1                               # der Flip
S_nach = E_u[:-1] * B_u
u_nach = E_u**2 + np.concatenate([B_u, [0]])**2
print(f"S: max|S_nach + S_vor| = {np.abs(S_nach + S_vor).max():.1e}"
      f"  (S dreht exakt um)")
print(f"u: max|u_nach - u_vor| = {np.abs(u_nach - u_vor).max():.1e}"
      f"  (u bleibt exakt)")
```

```
S: max|S_nach + S_vor| = 0.0e+00  (S dreht exakt um)
u: max|u_nach - u_vor| = 0.0e+00  (u bleibt exakt)
```

(c) Die ebene Welle verrät nichts — rückwärts ist sie einfach eine Welle in Gegenrichtung, beide Filme sind gleichberechtigte Lösungen. Tasse und gedämpfter Kreis verraten sich sofort: Energie sammelt sich nie von selbst aus tausend Splittern bzw. aus der Wärme wieder ein. *Mikroskopisch* ist trotzdem beides umkehrbar — jede einzelne Molekülkollision, jeder Feldschritt gehorcht richtungslosen Gleichungen. Was die Richtung erzeugt, ist die Information: Der Rückwärtsfilm verlangt absurd fein verabredete Anfangsbedingungen (jeder Splitter, jede Molekülschwingung exakt richtig), und die kleinste Abweichung davon wächst exponentiell — wie in Abbildung 20.2 gemessen.

Ü 20.2 (Verstehen/Verändern). Der Horizont eines ehrlichen Messgeräts: Wiederhole die Horizont-Reihe aus Abbildung 20.2 mit $\delta = 10^{-6}$ (sechs gültige Stellen — ein sehr gutes Voltmeter). Bei welchem T^* endet die Rekonstruktion jetzt, in Schwingungsperioden ($T_0 = 2\pi$) ausgedrückt? Und andersherum: Wie genau müsste man messen, um 50 Schwingungen weit zurückzusehen?

Musterlösung zu Ü 20.2

```
# von oben: np, kreis_hin_zurueck, L, R
d6 = 1e-6
for T in (100.0, 200.0, 250.0, 300.0):
    print(f"T = {T:3.0f}: Rest {kreis_hin_zurueck(R, T, d6):.2e}")
T6 = 2 * L / R * np.log(1 / d6)
print(f"T* = {T6:.0f}  =  {T6 / (2 * np.pi):.0f} Schwingungen")
delta_50 = np.exp(-50 * 2 * np.pi * R / (2 * L))
print(f"Für 50 Schwingungen: δ ≲ {delta_50:.0e}")
```

```
T = 100: Rest 3.20e-04
T = 200: Rest 3.31e-02
T = 250: Rest 4.92e-01
T = 300: Rest 3.19e+01
T* = 230  =  37 Schwingungen
Für 50 Schwingungen: δ ≲ 7e-09
```

Mit sechs Stellen endet die Vergangenheit nach $T^* = (2L/R)\ln 10^6 \approx 230$, also nach **37 Schwingungen** — die Messreihe bestätigt es: Zwischen $T = 200$ und $T = 250$ erreicht der Fehler die Größe des Signals, bei $T = 300$ ist nichts mehr zu retten. Für 50 Schwingungen bräuchte man $\delta \lesssim 7 \cdot 10^{-9}$, also neun gültige Stellen — pro gewünschter Schwingung Vergangenheit kostet es den Faktor $e^{2\pi R/2L} \approx 1{,}46$, etwa eine sechstel Dekade Messgenauigkeit. Information über die Vergangenheit gedämpfter Systeme wird exponentiell teuer — das ist der Kern des Zeitpfeils.

Ü 20.3 (Verändern). Die Stärke des Feindes: Lass den 2D-Doppellauf für drei Absorberstärken laufen ($V = 0{,}05$, $0{,}111$, $0{,}2$, also $c_a = 0{,}905$, $0{,}8$, $0{,}667$) und bestimme jeweils das Restfeld und die Fehlerrate pro Rückwärtsschritt (Log-Fit wie immer). Vergleiche mit der lokalen Maximalsteigung $\ln(1/c_a)$. Warum bleibt die gemessene Rate darunter — und warum rückt sie mit wachsendem V näher heran?

Musterlösung zu Ü 20.3

```
# von oben: np, hin_und_zurueck_2d, N_SCHRITTE
print(" V      Restfeld    Rate/Schritt   ln(1/ca)   Verhältnis")
for V in (0.05, 0.111, 0.2):
    rst, ft, fw, _, _ = hin_und_zurueck_2d(V)
    rs = N_SCHRITTE - 1 - ft
    rate = np.polyfit(rs[2:], np.log(fw[2:]), 1)[0]
    grenze = np.log((1 + V) / (1 - V))
    print(f"{V:5.3f}   {rst:9.1e}   {rate:8.3f}     {grenze:8.3f}"
          f"   {rate / grenze:6.2f}")
```

```
 V      Restfeld    Rate/Schritt   ln(1/ca)   Verhältnis
0.050     2.2e-13      0.039        0.100     0.39
0.111     9.2e+00      0.170        0.223     0.76
0.200     1.1e+23      0.375        0.405     0.92
```

Schon $V = 0{,}05$ verstärkt den Rundungsrest in 260 Schritten um zehn Größenordnungen (bleibt aber unter der Hörschwelle); $V = 0{,}2$ produziert ein Restfeld von 10^{23}. Die gemessene Rate liegt unter $\ln(1/c_a)$, weil der Balken kein geschlossener Kasten ist: Das in ihm verstärkte Rauschen strahlt ständig in das umgebende Vakuum ab, wo es nicht weiterwächst — die globale Rate ist ein Kompromiss aus Verstärkung drinnen und Verdünnung draußen. Je stärker der Absorber, desto mehr dominiert die Verstärkung das Rennen gegen die Abstrahlung, und das Verhältnis klettert von 0,39 über 0,76 auf 0,92 Richtung 1.

Ü 20.4 (Übertragen). Baue den Zeitumkehrspiegel: Zeichne im verlustfreien 2D-Lauf das Feld E_z an 72 Punkten eines Rings (Radius 60 Zellen um die Quelle) über 300 Schritte auf. Starte dann ein *leeres* Gitter und speise an denselben Punkten die Aufzeichnung **zeitgespiegelt** als weiche Quellen ein. Was entsteht im Ringinneren — wo, wann, und wie scharf? Vergleiche die Fokusbreite mit $\lambda/2$ und erkläre den Zusammenhang mit Feind Nummer drei.

Musterlösung zu Ü 20.4

```python
# von oben: np, plt, quellsignal, rot_h, NX, NY, DT, QX, QY, PERIODE
N_AUF, RAD, N_PKT = 300, 60, 72
w = 2 * np.pi * np.arange(N_PKT) / N_PKT
# np.round + .astype(int): auf ganze Gitterindizes runden
RX = np.round(QX + RAD * np.cos(w)).astype(int)
RY = np.round(QY + RAD * np.sin(w)).astype(int)

def vakuum_schritt(Ez, Hx, Hy):
    Hx[:, :-1] -= DT * (Ez[:, 1:] - Ez[:, :-1])
    Hy[:-1, :] += DT * (Ez[1:, :] - Ez[:-1, :])
    Ez[:] = Ez + DT * rot_h(Hx, Hy)

# Aufnahme: normaler Quelllauf, Ring hört mit
Ez = np.zeros((NX, NY)); Hx = np.zeros((NX, NY)); Hy = np.zeros((NX, NY))
aufnahme = np.zeros((N_AUF, N_PKT))
for n in range(N_AUF):
    vakuum_schritt(Ez, Hx, Hy)
    Ez[QX, QY] += quellsignal(n)
    aufnahme[n] = Ez[RX, RY]

# Wiedergabe: leeres Gitter, Ring sendet die Aufnahme rückwärts
Ez = np.zeros((NX, NY)); Hx = np.zeros((NX, NY)); Hy = np.zeros((NX, NY))
max_amp, max_n, fokus = 0.0, 0, None
for n in range(N_AUF):
    vakuum_schritt(Ez, Hx, Hy)
    Ez[RX, RY] += aufnahme[N_AUF - 1 - n]    # zeitgespiegelt!
    if abs(Ez[QX, QY]) > max_amp:
        max_amp, max_n, fokus = abs(Ez[QX, QY]), n, Ez.copy()

fig, ax = plt.subplots(figsize=(5.2, 5.2))
sk = np.abs(fokus).max()
ax.imshow(fokus.T, origin="lower", cmap="RdBu_r", vmin=-sk, vmax=sk)
ax.plot(RX, RY, ".", color="green", ms=3)
ax.plot(QX, QY, "k+", ms=10)
ax.set_xticks([]); ax.set_yticks([])
plt.show()

schnitt = np.abs(fokus[QX - 30:QX + 31, QY])
halb = np.flatnonzero(schnitt > schnitt.max() / 2)
print(f"Fokus bei Schritt {max_n} (Spiegelzeit der Emission: "
      f"{N_AUF - 1 - 50}); FWHM {halb[-1] - halb[0]} Zellen "
      f"(λ/2 = {PERIODE * DT / 2:.0f} Zellen)")
```

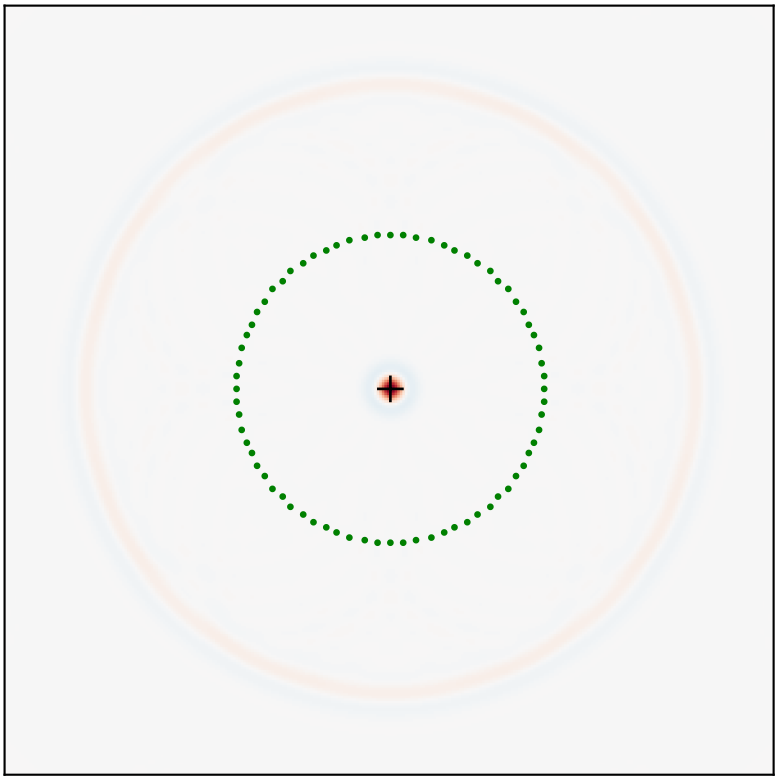


Abbildung 20.5: Der Mini-Zeitumkehrspiegel im Moment des Fokus: Die von 72 Ringpunkten (grün) zeitgespiegelt wieder ausgesandte Welle ist nach innen gelaufen und sammelt sich von selbst am alten Quellort — exakt zur Spiegelzeit der Emission. Der Fokus ist 6 Zellen breit (λ = 20 Zellen): Schärfer als etwa eine halbe Wellenlänge geht es nicht, denn die evaneszenten Details der Punktquelle waren am Ring längst unmessbar klein — die Vergangenheit ist nur beugungsbegrenzt scharf.

```
Fokus bei Schritt 249 (Spiegelzeit der Emission: 249); FWHM 6 Zellen (λ/2 = 10 Zellen)
```

Im Inneren entsteht eine perfekt einlaufende Kreiswelle, die sich **exakt zur Spiegelzeit der Emission** (Schritt 249) am alten Quellort sammelt — obwohl keiner der 72 Lautsprecher etwas von den anderen „weiß"; die Aufzeichnung enthält alle nötigen Laufzeit- und Phasenbeziehungen von selbst. Der Fokus ist 6 Zellen breit, von der Größenordnung $\lambda/2$: Die Punktquelle (eine Zelle!) kommt nicht zurück, denn ihre feineren Details steckten in evaneszenten Anteilen, die den Ring nie messbar erreicht haben — Feind Nummer drei, jetzt als Messwert. Genau dieses

> Prinzip fokussiert in der Medizin Stoßwellen durch inhomogenes Gewebe auf Nierensteine: Der Spiegel braucht kein Modell des Mediums, er spielt nur das rückwärts, was er gehört hat.

Das Kleingedruckte

Loschmidt gegen Boltzmann. Unser Absorber-Experiment ist die Gitterfassung eines Streits von 1876: Loschmidt hielt Boltzmann entgegen, aus umkehrbaren Gleichungen könne unmöglich ein unumkehrbares Naturgesetz (Entropiezunahme) folgen — man drehe einfach alle Geschwindigkeiten um. Boltzmanns Antwort ist im Kern unsere Fehlerkurve: Die Umkehrung müsste *exakt* sein, und jede noch so kleine Abweichung wächst in Systemen mit vielen Freiheitsgraden exponentiell. Der Zweite Hauptsatz ist kein Bewegungsgesetz, sondern eine Aussage über Wahrscheinlichkeit und Wissen — was sich auch daran zeigt, dass *Spin-Echos* in der Kernspinresonanz das Loschmidt-Umdrehen für kleine, gut isolierte Systeme tatsächlich vorführen (dort heißt der B-Flip 180°-Puls).

Warum $e^{RT/2L}$ und nicht $e^{RT/L}$? Die *Energie* des Kreises fällt mit $e^{-Rt/L}$ (Kapitel 19), die *Amplituden* q und i fallen mit der halben Rate $e^{-Rt/2L}$ — und rückwärts verstärkt werden Zustandsgrößen, nicht Energien. Eine Störung δ in q wächst daher mit $e^{+Rt/2L}$. Wer die Horizont-Formel auf die Energie bezieht, bekommt denselben Horizont: $\delta^2 e^{RT/L} \sim 1$ ist dieselbe Bedingung wie $\delta\, e^{RT/2L} \sim 1$.

Avancierte Lösungen sind nicht nutzlos. Die „verbotenen” einlaufenden Lösungen sind dieselben Funktionen, die der Zeitumkehrspiegel *praktisch herstellt* — man muss sie nur aktiv präparieren, statt auf ihr spontanes Auftreten zu warten. Auch die Theorie verwendet sie: Wheeler und Feynman haben 1945 eine Elektrodynamik aus halb retardierten, halb avancierten Feldern konstruiert (der Strahlungswiderstand kommt dort von den Absorbern des restlichen Universums). Sie blieb eine Kuriosität, aber eine lehrreiche: Die Retardierung ist eine Eigenschaft unserer *Welt* (ihrer Anfangsbedingung), nicht der Elektrodynamik.

Präzisions-Grenzen des Schnappschuss-Determinismus. „Ein Schnappschuss legt alles fest” gilt im Rahmen der klassischen Theorie. Real begrenzen ihn dieselben drei Feinde plus einer: Auch das *Aufzeichnen* selbst stört. An der Quantengrenze (Kapitel 30) wird daraus ein prinzipielles Verbot — den vollständigen Feldzustand eines einzelnen Photons kann man nicht kopieren. Der klassische Determinismus dieses Kapitels ist also

eine Idealisierung — eine, die für Radiowellen, Ultraschall und Seismik hervorragend trägt.

21 Vom Strom zur Welle

Die Kette aus Kapitel 19 endete an einer Programmzeile. Steckdose, Netzteil, Oszillator, Verstärker — und dann stand da: „Die Meep-`Source` ist die Abstraktion genau dieser Kette.” Eine Programmzeile sagt „hier sitzt ein Sender”, und die Welle läuft. Aber *wie*? Zwischen dem Wechselstrom, der in einem Stück Draht hin- und herschwappt, und der Funkwelle, die hundert Meter weiter ein Radio erreicht, liegt der eigentliche Zaubertrick der ganzen Funktechnik: Das Feld muss sich **vom Draht lösen**. Heinrich Hertz hat diesen Trick 1886 in einem Hörsaal in Karlsruhe zum ersten Mal vorgeführt — mit einer Funkenstrecke als Sender und einem Drahtring als Empfänger, ein paar Meter weiter. Auf die Frage nach dem Nutzen soll er geantwortet haben, das Ganze habe vermutlich keinen. Dein Telefon führt das Gegenargument gerade in der Hosentasche.

In diesem Kapitel eröffnen wir Teil VI und bauen die Brücke in drei Bögen: *Warum* strahlt eine beschleunigte Ladung überhaupt (ein Geometrie-Argument, das ohne eine einzige neue Formel das $1/r$-Gesetz liefert)? *Wie* macht eine Antenne daraus eine Maschine (der Strom im Draht ist eine alte Bekannte: eine stehende Welle)? Und *was* kommt draußen an — wie teilt sich die Welt um die Antenne in ein gebundenes Nahfeld und ein frei laufendes Fernfeld, und in welche Richtungen geht die Leistung? Dafür verlassen wir zum ersten Mal im Buch die Fläche: Die Potenzgesetze der Abstrahlung sind 3D-Physik, und wir rechnen sie in 3D nach.

Lernziele

Nach diesem Kapitel kannst du …

1. … mit dem Knick-Argument erklären, warum genau *beschleunigte* Ladungen strahlen — und aus der Knick-Geometrie herleiten, dass das Strahlungsfeld wie $1/r$ abfällt und wie $\sin\theta$ von der Richtung abhängt,
2. … die Stromverteilung auf einem Halbwellendipol als stehende Welle erklären (Bauch am Speisepunkt, Knoten an den offenen Enden) und im Feldbild die Hertz’sche Ablösung der Feldlinien-Schleifen erkennen,

3. … Nahfeld und Fernfeld quantitativ unterscheiden: die drei Stockwerke der Hertz-Formel ($1/r^3$, $1/r^2$, $1/r$), die Grenze $r = \lambda/2\pi$ und das Verhältnis von Blind- zu Wirkleistung $1/(kr)^3$,
4. … ein Strahlungsdiagramm messen (Near-to-Far in 3D) und lesen, die Direktivität daraus integrieren und in dBi umrechnen,
5. … aus Sendeleistung, Direktivität und Abstand die Leistungsdichte und Feldstärke am Empfangsort abschätzen (das $1/r^2$-Gesetz des Funks).

```
import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)        # Meeps Fortschrittsmeldungen aus (Kap. 16)
```

```
Using MPI version 4.1, 1 processes
```

```
1
```

21.1 Warum strahlt es überhaupt? Der Knick

Eine ruhende Ladung hat ein radiales Feld, das wie $1/r^2$ abfällt — das Coulomb-Feld aus Kapitel 1 und 3. Eine *gleichförmig* bewegte Ladung trägt dieses Feld einfach mit sich; aus ihrer Sicht ruht sie ja. In beiden Fällen wird nichts abgestrahlt: $1/r^2$ im Feld heißt $1/r^4$ in der Energiedichte, und das ist, über immer größere Kugelflächen $4\pi r^2$ integriert, immer weniger — die Energie bleibt bei der Ladung.

Interessant wird es beim **Beschleunigen**, und der Grund steckt in einer Einsicht aus Kapitel 20: Feldnachrichten reisen mit c. Was eine Ladung *jetzt* tut, weiß das Feld im Abstand r erst nach der Laufzeit r/c — das Feld dort draußen zeigt den **retardierten** Zustand der Quelle. Nun lassen wir eine Ladung, die lange geruht hat, zur Zeit $t = 0$ einen kurzen Stoß der Dauer τ erleiden, nach dem sie mit der Geschwindigkeit v weiterdriftet. Dann zerfällt die Welt zur Zeit t in drei Zonen:

- **Außen** ($r > ct$): Die Nachricht vom Stoß ist noch nicht angekommen. Die Feldlinien zeigen radial auf den *alten* Ruheplatz — als wäre nichts geschehen.
- **Innen** ($r < c(t - \tau)$): Hier ist das Feld längst auf dem neuesten Stand; die Linien zeigen radial auf die *neue*, driftende Position.

- **Dazwischen** eine Kugelschale der Dicke $c\tau$, die mit Lichtgeschwindigkeit nach außen läuft. Feldlinien reißen nicht ab (es gibt keine Ladung, an der sie enden könnten — Gauß, Kapitel 3), also müssen sie innerhalb der Schale die innere und die äußere Zone **verbinden**. Diese Verbindungsstücke sind geknickt — und der Knick ist quer zur Ausbreitungsrichtung.

> Vorhersage-Punkt
>
> Die Ladung driftet nach rechts. Welche Feldlinien bekommen den stärksten Knick — die nach rechts und links (in Bewegungsrichtung) oder die nach oben und unten (quer dazu)? Und was bedeutet das für die Richtung, in die am stärksten „gesendet" wird?

Das Bild dazu können wir mit reiner Geometrie zeichnen — keine Maxwell-Maschinerie, nur die drei Zonen:

```
C = 1.0      # Lichttempo
V = 0.25     # Drifttempo der Ladung nach dem Stoß
TAU = 0.6    # Dauer des Stoßes

def ort_der_ladung(t):
    """Ruhe bis t=0, gleichmäßige Beschleunigung bis TAU, dann Drift."""
    if t <= 0:
        return 0.0
    if t <= TAU:
        return 0.5 * (V / TAU) * t**2
    return V * (t - TAU / 2)

def zeichne_knick(ax, t, n_linien=24):
    r_aussen = C * t                        # Nachricht „Stoß beginnt"
    r_innen = max(C * (t - TAU), 0.0)       # Nachricht „Stoß ist vorbei"
    x_q = ort_der_ladung(t)
    winkel = np.linspace(0, 2 * np.pi, n_linien, endpoint=False)
    for w in winkel:
        ex, ey = np.cos(w), np.sin(w)
        # innerer Ast: Strahl von der NEUEN Position bis zur inneren
        # Schalenwand (Schnitt Strahl/Kreis um den Stoßort, exakt)
        unter_wurzel = (x_q * ex)**2 - x_q**2 + r_innen**2
        s_ende = -x_q * ex + np.sqrt(max(unter_wurzel, 0.0))
        s_in = np.linspace(0.25, max(s_ende, 0.25), 20)
        ax.plot(x_q + s_in * ex, s_in * ey, "b-", lw=0.8)
        # äußerer Ast: Strahl vom ALTEN Ruheplatz, ab der Schale
        s_out = np.linspace(r_aussen, 9.5, 20)
```

```
        ax.plot(s_out * ex, s_out * ey, "b-", lw=0.8)
        # der Knick: das Verbindungsstück durch die Schale
        ax.plot([x_q + s_in[-1] * ex, r_aussen * ex],
                [s_in[-1] * ey, r_aussen * ey], "r-", lw=1.6)
    for r, stil in ((r_innen, ":"), (r_aussen, "--")):
        ax.add_patch(plt.Circle((0, 0), r, fill=False, color="gray",
                                ls=stil, lw=0.8))
    ax.plot(x_q, 0, "ko", ms=6)
    ax.set_xlim(-8.4, 8.4); ax.set_ylim(-8.4, 8.4)
    ax.set_aspect("equal")
    ax.set_xticks([]); ax.set_yticks([])
    ax.set_title(f"t = {t:.1f}")

fig, achsen = plt.subplots(1, 3, figsize=(7.5, 2.9))
for ax, t in zip(achsen, (2.0, 4.5, 7.5)):
    zeichne_knick(ax, t)
plt.show()
```

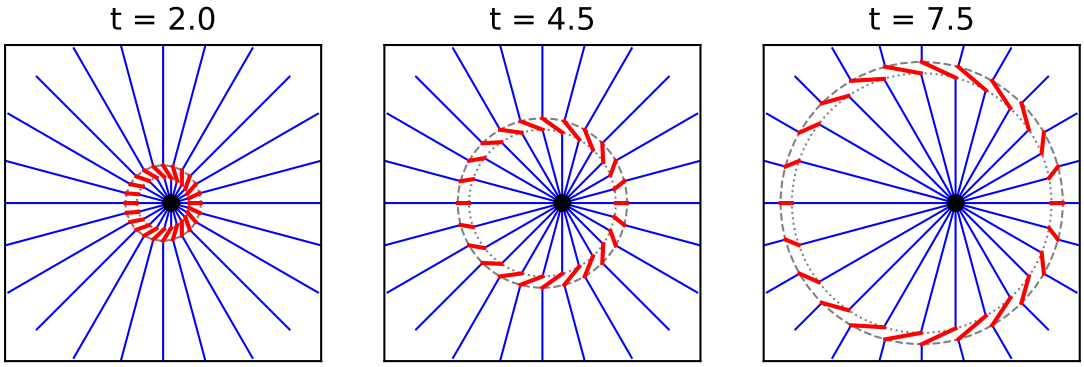


Abbildung 21.1: Die Feldlinien einer Ladung (schwarzer Punkt), die bei t = 0 aus der Ruhe angestoßen wurde und seither nach rechts driftet, zu drei Zeiten. Außen (jenseits des gestrichelten Kreises bei r = ct) zeigen die Linien noch auf den alten Ruheplatz, innen schon auf die neue Position. Dazwischen läuft eine Schale der Dicke $c\tau$ mit Lichtgeschwindigkeit nach außen, in der jede Feldlinie einen Knick (rot) trägt — quer zur Laufrichtung, am stärksten senkrecht zur Bewegung, exakt null in Bewegungsrichtung.

Die Vorhersage lässt sich direkt ablesen: In Bewegungsrichtung (nach rechts und links) liegen alter und neuer Strahl auf derselben Geraden — **kein Knick**. Quer zur Bewegung (oben und unten) ist der Versatz zwischen

den Zonen am größten — **der stärkste Knick**. Eine schwingende Ladung sendet quer zu ihrer Bewegung am stärksten und längs ihrer Bewegung gar nicht. Das ist genau das Dipol-Argument, mit dem Kapitel 13 den Brewster-Winkel erklärt hat — und wir werden es am Ende dieses Kapitels als „Donut"-Strahlungsdiagramm vermessen.

21.1.1 Ein Strahlensatz liefert das $1/r$-Gesetz

Der Knick kann mehr als ein hübsches Bild sein — er trägt eine quantitative Botschaft. Schau auf eine Feldlinie, die senkrecht zur Bewegung läuft (im Bild: nach oben), zur späten Zeit t. Ihr Knickstück überbrückt **quer** den Versatz zwischen innerer und äußerer Zone: Die innere Zone ist um den Drift $v\,t$ verschoben, also ist die Querkomponente des Knicks proportional zu $v\,t$ (und allgemein, bei Beobachtung unter dem Winkel θ zur Bewegungsachse, zu $v\,t\sin\theta$ — nur die Querkomponente des Versatzes knickt die Linie). **Radial** ist das Knickstück genau die Schalendicke $c\,\tau$ lang. Das Verhältnis von Quer-Feld zu Radial-Feld ist also pure Geometrie:

$$\frac{E_\perp}{E_r} = \frac{v\,t\,\sin\theta}{c\,\tau}.$$

Jetzt zahlen sich zwei alte Bekannte aus. Erstens: Das radiale Feld ist das Coulomb-Feld, $E_r \propto 1/r^2$. Zweitens: Die Schale ist zur Zeit t am Radius $r = c\,t$ angekommen, also ist $t = r/c$. Einsetzen:

$$E_\perp = E_r \cdot \frac{v\,(r/c)\sin\theta}{c\,\tau} \propto \frac{1}{r^2}\cdot\frac{v\,r\sin\theta}{c^2\tau} = \frac{(v/\tau)\,\sin\theta}{c^2}\cdot\frac{1}{r},$$

und v/τ ist die **Beschleunigung** a während des Stoßes. Das Querfeld in der Schale fällt nur wie $1/r$ — eine ganze Potenz langsamer als jedes statische Feld. Seine Energiedichte fällt wie $1/r^2$, und über die Kugelfläche $4\pi r^2$ integriert bleibt eine **konstante** Energie in der Schale, egal wie weit sie schon gelaufen ist: Diese Energie kommt nie zur Ladung zurück. *Das* ist Strahlung. Drei Befunde aus einem Strahlensatz: Strahlung braucht Beschleunigung ($E_\perp \propto a$), sie fällt wie $1/r$, und sie geht wie $\sin\theta$ bevorzugt quer.

Eine Handrechnung am dritten Bild von Abbildung 32.5, damit eine Zahl herauskommt: Dort ist $v = 0{,}25\,c$, $\tau = 0{,}6$ und $t = 7{,}5$. Senkrecht nach oben ($\sin\theta = 1$) erwarten wir $E_\perp/E_r = 0{,}25\cdot 7{,}5/0{,}6 = 3{,}1$ — das Knickstück sollte gut dreimal so weit quer laufen wie radial. Miss im Bild nach: Der rote Strich oben ist tatsächlich fast quer gestellt, rund dreimal länger

als die Schalendicke. Je später man schaut, desto querer der Knick — der Strahlungsanteil gewinnt mit wachsendem r gegen das Coulomb-Feld, das ist der Kern des $1/r$-Arguments.

Eine Antenne ist nun nichts anderes als eine **Maschine für periodische Knicke**: Der Wechselstrom beschleunigt die Ladungen im Draht im Takt der Frequenz hin und her, jede Halbschwingung schickt eine neue Knick-Schale los, und die Schalen hintereinander *sind* die Welle — Wellenlänge $\lambda = c/f$, wie seit Kapitel 4. Schauen wir der Maschine bei der Arbeit zu.

21.2 Die Antenne als Knick-Maschine

Wir bauen den Klassiker: einen **Halbwellendipol** — zwei Metallstäbe in einer Linie, zusammen etwa $\lambda/2$ lang, in der Mitte von einem Wechselstrom gespeist. Die Geschichte spielt im UKW-Band: $f = 100$ MHz, also $\lambda = 3$ m und eine Antenne von etwa $1{,}5$ m — die Maße eines Dachantennen-Stabs.

Frage: Wie verteilt sich der Strom auf dem Draht, und wie löst sich das Feld ab? **Bühne:** eine 2D-Zelle von $8\lambda \times 8\lambda$ (24 m × 24 m), Auflösung 48 Punkte pro λ, rundum 1 λ PML; wir bleiben für dieses Feldlinien-Bild bewusst in 2D, weil sich E-Feldlinien nur dort direkt zeichnen lassen (das E-Feld liegt in der Ebene, H_z steht senkrecht — die TE-Polarisation, das Gegenstück zur TMz-Welt aus Kapitel 9). **Aufbau:** zwei Metallarme entlang y (Gesamtlänge $0{,}48\lambda$, Drahtbreite $0{,}06\lambda$) mit einem Speisespalt von $0{,}0625\lambda$ in der Mitte, darin eine weiche E_y-Dauerstrichquelle ($f = 1$ in Meep-Einheiten, sanft eingeschaltet über eine Periode — Kapitel 6). Die beiden Detailwahlen gehören zusammen: Der Speisespalt muss **mehrere Gitterzellen** breit sein (hier drei). Wäre er schmaler als eine Zelle, würde das Gitter ihn beim Rastern der Metallblöcke einfach zuschütten — die Quelle säße *im* Metall, und der Kurzschluss würgt die Felder um viele Größenordnungen ab. Tückisch daran: Normierte Bilder sähen weiterhin plausibel aus, nur die absoluten Feldstärken verraten den Fehler. **Messgrößen:** (1) die Feldlinien von $\vec{E}$ nach dem Einschwingen, (2) der Strom im Draht über den Sprung von H_z quer über das Blech (Durchflutungssatz aus Kapitel 3: ein Umlauf eng um den Leiter misst den eingeschlossenen Strom). **Erfolgskriterium:** geschlossene Feldlinien-Schleifen, die sich im Abstand $\lambda/2$ ablösen, und eine Stromverteilung mit Bauch in der Mitte.

Vor dem Lauf der Vorhersage-Punkt zum Strom:

Vorhersage-Punkt

Am offenen Drahtende kann keine Ladung hinaus — der *Leitungsstrom* muss dort null sein. Was heißt das für die Stromverteilung auf dem Draht? Du hast dieses Muster schon einmal gesehen: in Kapitel 6, als eine Welle zwischen zwei Spiegeln gefangen war. Wo liegt der Bauch, wo liegen die Knoten?

Dann lassen wir die Maschine anlaufen — erst einschwingen, dann eine volle Periode dicht fotografieren:

```
A_METER = 3.0            # 1 Meep-Längeneinheit = lambda = 3 m (UKW)
DIPOL_L = 0.48           # Gesamtlänge in lambda (knapp lambda/2)
SPALT = 0.0625           # Speisespalt — exakt 3 Gitterzellen breit!
DRAHT_B = 0.06           # Drahtbreite — dick genug fürs Gitter
AUFL = 48

arm = DIPOL_L / 2 - SPALT / 2
y_arm = SPALT / 2 + arm / 2
zwei_arme = [
    mp.Block(size=mp.Vector3(DRAHT_B, arm, mp.inf),
             center=mp.Vector3(0, +y_arm), material=mp.metal),
    mp.Block(size=mp.Vector3(DRAHT_B, arm, mp.inf),
             center=mp.Vector3(0, -y_arm), material=mp.metal),
]
sim = mp.Simulation(
    cell_size=mp.Vector3(8.0, 8.0),
    boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.ContinuousSource(frequency=1.0, width=1.0),
                       component=mp.Ey, center=mp.Vector3())],
    resolution=AUFL, geometry=zwei_arme)

filmbilder = {"Ex": [], "Ey": [], "Hz": []}

def merke(s):
    filmbilder["Ex"].append(s.get_array(component=mp.Ex))
    filmbilder["Ey"].append(s.get_array(component=mp.Ey))
    filmbilder["Hz"].append(s.get_array(component=mp.Hz))

sim.run(until=20.0)                                  # einschwingen
sim.run(mp.at_every(1.0 / 16.0, merke), until=1.0)   # 1 Periode filmen
Ex_t = np.array(filmbilder["Ex"])[:16]
Ey_t = np.array(filmbilder["Ey"])[:16]
```

```
Hz_t = np.array(filmbilder["Hz"])[:16]
nt, nx, ny = Ex_t.shape
x_m = np.linspace(-4, 4, nx) * A_METER       # Achsen in Metern
y_m = np.linspace(-4, 4, ny) * A_METER
print(f"{nt} Filmbilder über exakt eine Periode, Gitter {nx} x {ny}")
```

```
16 Filmbilder über exakt eine Periode, Gitter 384 x 384
```

Die Strommessung: Wir lesen H_z eine Gitterzelle links und rechts vom Blech ab. Der Durchflutungssatz sagt — entzerrt, wie es sich gehört: Ein enger Umlauf um den Leiter sammelt $\oint \vec{H} \cdot d\vec{l}$; in unserer 2D-Welt hat $\vec{H}$ nur die z-Komponente, also bleibt vom Umlauf genau die Differenz der beiden Ablesungen übrig. Sie ist der Strom (pro Meter Tiefe), der zwischen den beiden Ablesepunkten hindurchfließt:

```
# von oben: Ex_t, Ey_t, Hz_t, x_m, y_m, AUFL, DRAHT_B, DIPOL_L, A_METER
L_m = DIPOL_L * A_METER

# --- Strom: Hz-Sprung eine Zelle neben dem Blech, Amplitude über
#     die aufgezeichnete Periode ---
abstand = DRAHT_B / 2 + 1.0 / AUFL
ix_links = np.argmin(np.abs(x_m / A_METER + abstand))
ix_rechts = np.argmin(np.abs(x_m / A_METER - abstand))
sprung = Hz_t[:, ix_links, :] - Hz_t[:, ix_rechts, :]   # (Zeit, y)
strom = np.max(np.abs(sprung), axis=0)                  # Amplitude je y
im_draht = np.abs(y_m / A_METER) <= DIPOL_L / 2
strom = strom / strom[im_draht].max()

fig, (ax_f, ax_s) = plt.subplots(
    1, 2, figsize=(7.5, 4.0), gridspec_kw={"width_ratios": [1.35, 1]})

# --- links: Feldlinien über Feldstärke ---
i_bild = nt - 1
betrag = np.hypot(Ex_t[i_bild], Ey_t[i_bild])
skala = np.percentile(betrag, 70) + 1e-12
gestaucht = np.arcsinh(betrag / skala)     # arcsinh-Lupe (Kap. 9)
ax_f.imshow(gestaucht.T, origin="lower", cmap="inferno",
            extent=[x_m.min(), x_m.max(), y_m.min(), y_m.max()],
            vmin=0, vmax=np.percentile(gestaucht, 99.5))
with np.errstate(invalid="ignore"):
    ax_f.streamplot(x_m, y_m, Ex_t[i_bild].T, Ey_t[i_bild].T,
                    color="white", density=1.1, linewidth=0.5,
```

```
                    arrowsize=0.6)
ax_f.plot([0, 0], [SPALT / 2 * A_METER, L_m / 2], "c-", lw=3.5)
ax_f.plot([0, 0], [-L_m / 2, -SPALT / 2 * A_METER], "c-", lw=3.5)
ax_f.set_xlabel("x (m)"); ax_f.set_ylabel("y (m)")
ax_f.set_title("E-Feldlinien lösen sich ab")

# --- rechts: Stromprofil ---
ax_s.plot(np.where(im_draht, strom, np.nan), y_m, "b-", lw=2.5,
          label="Messung ($H_z$-Sprung)")
ideal = np.where(im_draht, np.abs(np.cos(np.pi * y_m / L_m)), np.nan)
ax_s.plot(ideal, y_m, "r--", lw=1.5, label="Ideal: dünner 3D-Draht")
ax_s.axhline(+L_m / 2, color="gray", lw=0.8, ls=":")
ax_s.axhline(-L_m / 2, color="gray", lw=0.8, ls=":")
ax_s.set_ylim(-1.3 * L_m / 2, 1.3 * L_m / 2); ax_s.set_xlim(0, 1.08)
ax_s.set_xlabel("Stromamplitude (normiert)"); ax_s.set_ylabel("y (m)")
ax_s.set_title("Strom im Draht")
ax_s.legend(fontsize=8, loc="center right"); ax_s.grid(alpha=0.3)
fig.tight_layout()
plt.show()
```

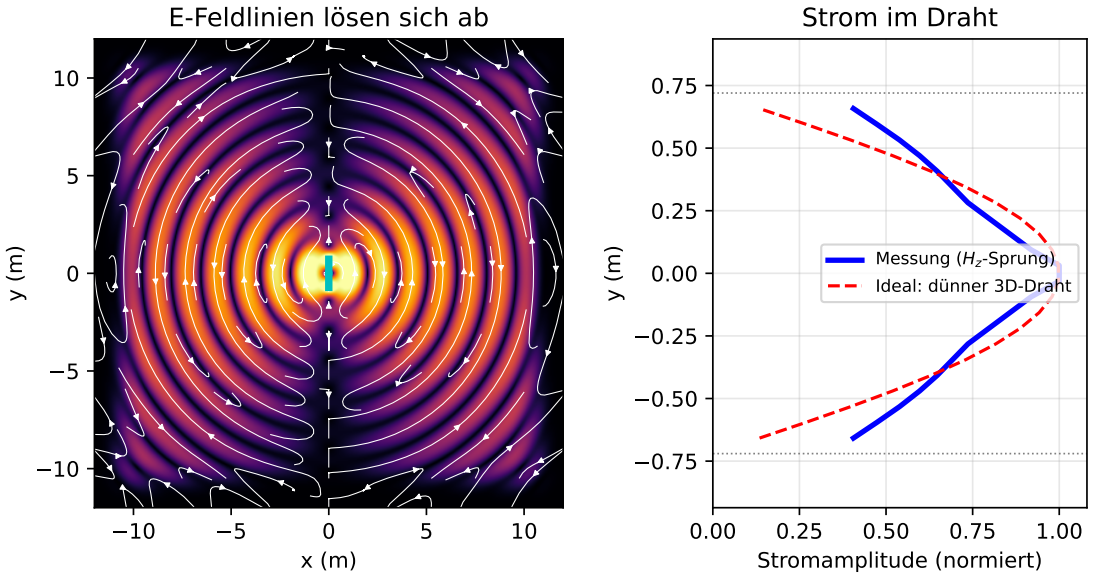


Abbildung 21.2: Links: das eingeschwungene Feld des 100-MHz-Dipols (Farbfläche: Feldstärke, logarithmisch gestaucht; weiße Linien: E-Feldlinien — Lesehilfe: Linien = Richtung, Farbe = Stärke). Nahe der Antenne spannen sich die Linien noch von Arm zu Arm, weiter draußen sind sie zu geschlossenen Schleifen abgeschnürt, die im Abstand λ/2 = 1,5 m aufeinanderfolgen. Rechts: die Stromamplitude auf dem Draht — ein Bauch am Speisepunkt, abfallend zu den Enden, dazu die ideale Dünndraht-Stehwelle als gestrichelte Erwartung. Warum unsere Messkurve die Null am Ende nicht erreicht, klärt der Text.

Die Vorhersage zum Strom trifft im Kern zu: **Der Strom auf der Antenne ist eine stehende Welle.** Der Speisepunkt sitzt im Bauch, zu den Enden hin fällt die Amplitude — dieselbe Physik wie die gefangene Welle in Kapitel 6, nur dass hier die *Stromwelle* im Draht am offenen Ende reflektiert wird (wo der Leitungsstrom nicht weiterkann), statt der Feldwelle am Spiegel. Genau deshalb ist $\lambda/2$ die natürliche Länge: Sie lässt einen vollen Bauch zwischen zwei Knoten genau auf die Antenne passen, der Draht schwingt in seiner Grundmode — die Antenne ist ein **Resonator** (Kapitel 15 lässt grüßen), den man in der Mitte anstößt.

Zwei Ehrlichkeiten zur rechten Kurve. Erstens erreicht unsere Messung am Drahtende nicht die Null, und das hat einen physikalisch lehrreichen Grund: Die Messschleife umfasst neben dem Leitungsstrom auch den **Verschiebungsstrom** (Kapitel 3!). An den Enden sammelt sich im Takt die Ladung,

ihr quellendes E-Feld führt den Strom als $\varepsilon_0 \partial \vec{E}/\partial t$ im Feld weiter — der *Gesamtstrom* reißt am Drahtende gar nicht ab, nur der Leitungsanteil. Zweitens ist die gestrichelte cos-Kurve die Theorie für einen **dünnen Draht in 3D**; unser 2D-„Draht" ist in Wahrheit ein unendlich tiefes Blech, das deutlich anders belastet ist. Das ist dieselbe Lektion, die in Kapitel 11 die 2D-Yagi unbaubar machte — 2D taugt für das Prinzip, nicht für Drahtmaße. Wer exakte Drahtströme braucht, nimmt das Werkzeug mit Dünndraht-Modell: NEC, so wie es Kapitel 24 beim Yagi-Bauplan tun wird.

21.3 Das Feld macht sich selbstständig

Bevor wir messen, wie weit die Welle trägt, räumen wir die zentrale Fehlvorstellung ab:

> ⚠ Naheliegende Vermutung
>
> **„Das Feld gehört zu seiner Quelle. Schaltet man den Sender aus, bricht auch das Feld zusammen — so wie das Feld im Kondensator verschwindet, wenn man ihn entlädt."**
> *Warum sie naheliegt:* In der Statik stimmt sie. Jedes Feld in Teil I hing an seinen Ladungen und Strömen; nimmt man die Quelle weg, ist auch das Feld weg. Und der Sprachgebrauch — der Strom „erzeugt" das Feld — klingt nach einer Abhängigkeit, die ständigen Nachschub braucht.
> *Was stattdessen stimmt:* Feldnachrichten reisen mit c (Kapitel 20). Was die Quelle *jetzt* tut, erreicht eine abgelöste Feldschleife im Abstand r erst nach r/c — und holt sie nie ein, denn die Schleife flieht selbst mit c. Draußen tragen sich die Felder gegenseitig: Das wandernde $\vec{B}$ induziert $\vec{E}$, das wandernde $\vec{E}$ baut per Verschiebungsstrom $\vec{B}$ — die zwei dynamischen Maxwell-Gleichungen aus Kapitel 3 brauchen keine Quelle mehr, das war der Kern der Wellengleichung in Kapitel 4. Einmal abgelöst, ist die Welle **autark**.

Der Beweis ist ein Experiment, das die Behauptung wörtlich nimmt. **Frage:** Läuft die Welle weiter, wenn der Sender nachweislich schweigt? **Bühne:** dieselbe 2D-Welt, aber größer ($12\lambda \times 12\lambda$), damit der Ring nach dem Verstummen noch Strecke vor sich hat; ein kurzer Dipol ($0{,}3\lambda$ — er klingt nach dem Puls kaum nach). **Quelle:** ein Gauß-Puls um $f = 1$ mit `cutoff=3` — Meep schaltet die Quelle danach **exakt ab**, ab $t = 2 \cdot 3/1{,}1 \approx 5{,}5$ Perioden ist jeder Quellstrom identisch null. **Messgrößen:** Schnappschüsse des Felds und die Position des Ringmaximums über die Zeit. **Erfolgskriterium:** Der

Ring läuft nach $t = 5{,}5$ unverändert weiter, sein Radius wächst mit Tempo c.

```
KURZ_L = 0.3
arm_k = KURZ_L / 2 - SPALT / 2
y_arm_k = SPALT / 2 + arm_k / 2
sim_puls = mp.Simulation(
    cell_size=mp.Vector3(12.0, 12.0),
    boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(
        # cutoff=3: nach 3 Gauß-Breiten beidseits wird die Quelle
        # exakt null — danach fließt garantiert kein Quellstrom mehr
        mp.GaussianSource(frequency=1.0, fwidth=1.1, cutoff=3),
        component=mp.Ey, center=mp.Vector3())],
    resolution=AUFL,
    geometry=[
        mp.Block(size=mp.Vector3(DRAHT_B, arm_k, mp.inf),
                 center=mp.Vector3(0, +y_arm_k), material=mp.metal),
        mp.Block(size=mp.Vector3(DRAHT_B, arm_k, mp.inf),
                 center=mp.Vector3(0, -y_arm_k), material=mp.metal)])

puls = {"Ex": [], "Ey": [], "Hz": [], "t": []}

def merke_puls(s):
    puls["Ex"].append(s.get_array(component=mp.Ex))
    puls["Ey"].append(s.get_array(component=mp.Ey))
    puls["Hz"].append(s.get_array(component=mp.Hz))
    puls["t"].append(s.meep_time())   # aktuelle Simulationszeit

sim_puls.run(mp.at_every(0.25, merke_puls), until=10.5)
Ex_p = np.array(puls["Ex"]); Ey_p = np.array(puls["Ey"])
Hz_p = np.array(puls["Hz"]); t_p = np.array(puls["t"])
T_AUS = 2 * 3 / 1.1
print(f"{len(t_p)} Schnappschüsse; Quelle exakt aus ab t = {T_AUS:.2f}")
# Gesundheits-Check (siehe Bühnenbeschreibung oben): die ABSOLUTE
# Spitzenamplitude — säße die Quelle im zugerasterten Spalt, stünde
# hier nur Rundungsrauschen (~1e-19)
print(f"Spitzenamplitude des Laufs: max|Ey| = {np.abs(Ey_p).max():.3f}")
```

```
42 Schnappschüsse; Quelle exakt aus ab t = 5.45
Spitzenamplitude des Laufs: max|Ey| = 20.946
```

```
# von oben: Ex_p, Ey_p, t_p, T_AUS (Lauf oben); Gitter 12 lambda
x_p = np.linspace(-6, 6, Ex_p.shape[1])
# Ringposition: Maximum des Feldbetrags entlang der +x-Achse (y = 0)
iy0 = Ex_p.shape[2] // 2
profil = np.hypot(Ex_p[:, :, iy0], Ey_p[:, :, iy0])[:, x_p > 0.5]
r_ring = x_p[x_p > 0.5][np.argmax(profil, axis=1)]

print("   t    Ringradius   (Quelle aus ab t = 5,45)")
for tz in (6.0, 6.5, 7.0, 7.5, 8.0):
    i = np.argmin(np.abs(t_p - tz))
    print(f"  {t_p[i]:4.1f}   {r_ring[i]:5.2f} λ")
# Fit nur bis t = 8: danach erreicht der Ring die PML und die
# Maximum-Suche rastet auf den schwächeren 2D-Nachlauf ein
spaet = (t_p >= 5.5) & (t_p <= 8.0)
tempo = np.polyfit(t_p[spaet], r_ring[spaet], 1)[0]
print(f"\nTempo des Rings nach dem Verstummen: {tempo:.3f} c")
```

```
   t    Ringradius   (Quelle aus ab t = 5,45)
   6.0    2.87 λ
   6.5    3.37 λ
   7.0    3.87 λ
   7.5    4.37 λ
   8.0    4.87 λ

Tempo des Rings nach dem Verstummen: 1.002 c
```

Der Ring weiß nichts vom Schicksal seiner Quelle: Lange nachdem der letzte Quellstrom geflossen ist, wächst sein Radius stur mit einer Wellenlänge pro Periode — Tempo c bis auf rund ein Prozent (der Rest: numerische Dispersion aus Kapitel 8 plus die Rasterung der Maximum-Suche aufs Gitter). Dazu das Bild — und der Energieblick von Kapitel 18: Wo strömt die Leistung?

```
# von oben: Ex_p, Ey_p, Hz_p, t_p, x_p, KURZ_L, SPALT, A_METER
i8 = np.argmin(np.abs(t_p - 8.0))
y_p = x_p
Sx = Ey_p[i8] * Hz_p[i8]          # Poynting in 2D-TE (Kap. 18):
Sy = -Ex_p[i8] * Hz_p[i8]         # Sx = Ey·Hz, Sy = −Ex·Hz
fig, (ax_e, ax_p2) = plt.subplots(1, 2, figsize=(7.5, 3.9))

betrag_e = np.hypot(Ex_p[i8], Ey_p[i8])
sk_e = np.percentile(betrag_e, 70) + 1e-12
ge = np.arcsinh(betrag_e / sk_e)
```

```
ax_e.imshow(ge.T, origin="lower", cmap="inferno",
            extent=[-6, 6, -6, 6], vmin=0,
            vmax=np.percentile(ge, 99.5))
with np.errstate(invalid="ignore"):
    ax_e.streamplot(x_p, y_p, Ex_p[i8].T, Ey_p[i8].T, color="white",
                    density=1.1, linewidth=0.5, arrowsize=0.6)
ax_e.set_title("E-Feld: autarke Schleifen")
ax_e.set_xlabel("x (λ)"); ax_e.set_ylabel("y (λ)")

betrag_s = np.hypot(Sx, Sy)
sk_s = np.percentile(betrag_s[betrag_s > 0], 70) + 1e-12
gs = np.arcsinh(betrag_s / sk_s)
ax_p2.imshow(gs.T, origin="lower", cmap="viridis",
             extent=[-6, 6, -6, 6], vmin=0,
             vmax=np.percentile(gs, 99.5))
with np.errstate(invalid="ignore"):
    ax_p2.streamplot(x_p, y_p, Sx.T, Sy.T, color="white",
                     density=1.1, linewidth=0.5, arrowsize=0.6)
ax_p2.set_title(r"$\vec S = \vec E \times \vec H$: Energie radial nach außen")
ax_p2.set_xlabel("x (λ)"); ax_p2.set_ylabel("y (λ)")
for ax in (ax_e, ax_p2):
    ax.plot([0, 0], [SPALT / 2, KURZ_L / 2], "c-", lw=2.5)
    ax.plot([0, 0], [-KURZ_L / 2, -SPALT / 2], "c-", lw=2.5)
fig.tight_layout()
plt.show()
```

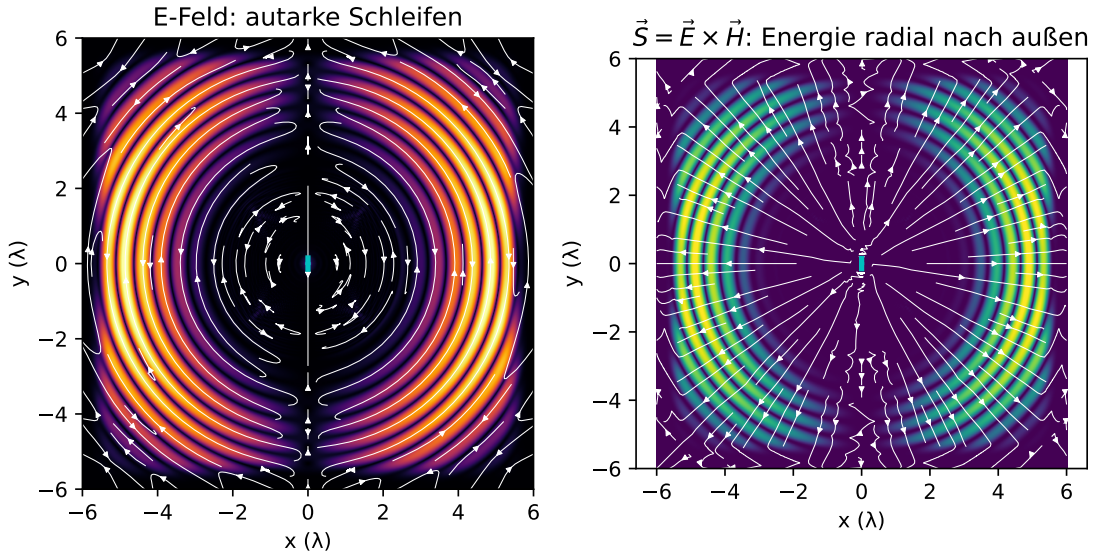


Abbildung 21.3: Der abgelöste Einzelpuls bei t = 8 — die Quelle ist seit zweieinhalb Perioden exakt still. Links das E-Feld mit Feldlinien: eine geschlossene, quellenfreie Schleifenstruktur, kein Faden führt mehr zur Antenne. Rechts der Poynting-Vektor (Farbfläche: Betrag, Linien: Flussrichtung): Die Energie strömt radial nach außen, mit dem Ring. Dass hinter dem Hauptring schwächere Ringe nachlaufen, ist eine 2D-Eigenheit (das Huygens-Prinzip gilt scharf nur in 3D); in 3D wäre der Puls solo unterwegs.

21.4 Nah und fern — gemessen in 3D

Wie stark ist das Feld in welchem Abstand? Für diese Frage reicht 2D nicht mehr: Dort verteilen sich Wellen auf wachsende Kreis*umfänge* und fallen wie $1/\sqrt{r}$ (gemessen in Kapitel 9). Die Funkwelt ist dreidimensional — Kugel*flächen* $4\pi r^2$, Strahlungsfeld $\propto 1/r$, genau wie es das Knick-Argument verlangt. Also rechnen wir jetzt zum ersten Mal im Buch **echtes 3D**.

Die Theorie dazu ist über 130 Jahre alt: Für den **Hertz'schen Dipol** — ein punktförmiges Stromelement, das mit f schwingt — lässt sich Maxwell exakt lösen. Quer zur Dipolachse (äquatorial) hat der Betrag des E-Felds die Form

$$|E|(r) \;\propto\; \sqrt{\left(\tfrac{1}{kr} - \tfrac{1}{(kr)^3}\right)^2 + \tfrac{1}{(kr)^4}}\,, \qquad k = \frac{2\pi}{\lambda},$$

ein Haus mit **drei Stockwerken**: Ganz nah dominiert $1/(kr)^3$ — das ist das *statische* Dipolfeld aus Kapitel 3, das im Takt mitschwingt. Ganz fern bleibt $1/(kr)$ — die Strahlung aus dem Knick-Argument. Dazwischen vermittelt der $1/(kr)^2$-Term. Die Stockwerke wechseln dort, wo $kr \approx 1$ ist, also bei

$$r = \frac{\lambda}{2\pi} \approx 0{,}16\,\lambda$$

— *der* Trennradius zwischen Nahfeld und Fernfeld. **Längs** der Achse dagegen fehlt der $1/(kr)$-Term komplett (kein Knick in Bewegungsrichtung!); dort fällt das Feld auch in der Ferne wie $1/r^2$ — die Antenne strahlt längs ihrer Achse *nichts* ab, wie es Kapitel 13 fürs Dipol-Argument schon benutzt hat.

Frage: Treffen unsere Messpunkte beide Kurven — $1/r$ quer, $1/r^2$ längs? **Bühne:** eine 3D-Zelle $6\lambda \times 6\lambda \times 6\lambda$ mit 1 λ PML, Auflösung 40. Das ist die bisher teuerste Zelle des Buchs (rund drei Minuten Rechenzeit); zwei **Spiegelsymmetrien** machen sie bezahlbar. **Quelle:** eine E_z-Punktquelle — sie ist der Hertz-Dipol in Reinform (ein Stromelement von der Größe einer Gitterzelle), gepulst um $f = 1$, gelegt exakt auf einen E_z-Gitterpunkt. **Messgrößen:** die DFT-Amplitude von E_z bei $f = 1$ (Kapitel 15) auf zwei Messlinien — äquatorial entlang x und axial entlang z —, dazu (für später) eine geschlossene Near-to-Far-Box. **Erfolgskriterium:** Beide Messreihen liegen auf ihrer Hertz-Kurve; im Fernbereich Steigung -1 bzw. -2 im log-log-Bild.

Die WLAN-Story dazu: In Meep-Einheiten ist $\lambda = 1$; lies die Zelle als 2,4-GHz-Welt, dann ist $\lambda = 12{,}5$ cm und die Zelle ein 75-cm-Würfel um einen winzigen Sender.

```
AUFL_3D = 40
z_q = 0.5 / AUFL_3D     # Quelle exakt auf einen Ez-Gitterpunkt
                        # (Ez wohnt auf halben z-Schritten – Yee!)
sim3 = mp.Simulation(
    # drei Komponenten in cell_size: die erste echte 3D-Zelle des Buchs
    cell_size=mp.Vector3(6, 6, 6),
    boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.GaussianSource(frequency=1.0, fwidth=0.3),
                       component=mp.Ez, center=mp.Vector3(0, 0, z_q))],
    resolution=AUFL_3D,
    # mp.Mirror: Aufbau und Quelle sind spiegelsymmetrisch in x und y –
    # Meep rechnet nur ein Viertel des Volumens, Ergebnis identisch
    symmetries=[mp.Mirror(mp.X), mp.Mirror(mp.Y)])
```

```
linie_eq = mp.Volume(center=mp.Vector3(0.95, 0, z_q),
                     size=mp.Vector3(1.9, 0, 0))
linie_ax = mp.Volume(center=mp.Vector3(0, 0, z_q + 0.95),
                     size=mp.Vector3(0, 0, 1.9))
# äquatorial nehmen wir Ez UND Hy auf — aus dem Paar wird gleich
# noch der komplexe Poynting-Vektor
dft_eq = sim3.add_dft_fields([mp.Ez, mp.Hy], 1.0, 0, 1, where=linie_eq)
dft_ax = sim3.add_dft_fields([mp.Ez], 1.0, 0, 1, where=linie_ax)

# Near-to-Far-Box (Kap. 14, jetzt geschlossen in 3D: sechs Flächen;
# weight=-1, wo die Außennormale in negative Achsrichtung zeigt) —
# wir nehmen sie im selben Lauf fürs Richtdiagramm weiter unten auf
b = 1.6
def n2f_box(s):
    return s.add_near2far(
        1.0, 0, 1,
        mp.Near2FarRegion(mp.Vector3(+b, 0, 0), size=mp.Vector3(0, 2*b, 2*b)),
        mp.Near2FarRegion(mp.Vector3(-b, 0, 0), size=mp.Vector3(0, 2*b, 2*b), weight=-1),
        mp.Near2FarRegion(mp.Vector3(0, +b, 0), size=mp.Vector3(2*b, 0, 2*b)),
        mp.Near2FarRegion(mp.Vector3(0, -b, 0), size=mp.Vector3(2*b, 0, 2*b), weight=-1),
        mp.Near2FarRegion(mp.Vector3(0, 0, +b), size=mp.Vector3(2*b, 2*b, 0)),
        mp.Near2FarRegion(mp.Vector3(0, 0, -b), size=mp.Vector3(2*b, 2*b, 0), weight=-1))

n2f_hertz = n2f_box(sim3)

# stop_when_fields_decayed: läuft, bis das Feld am Kontrollpunkt auf
# 1e-5 des Maximums gefallen ist — der Puls ist dann fertig abgestrahlt
sim3.run(until_after_sources=mp.stop_when_fields_decayed(
    5, mp.Ez, mp.Vector3(0.5, 0, 0), 1e-5))
print("3D-Lauf fertig.")
```

```
3D-Lauf fertig.
```

```
# von oben: sim3, dft_eq, dft_ax (3D-Lauf)
k = 2 * np.pi
ez_eq = np.abs(sim3.get_dft_array(dft_eq, mp.Ez, 0))
ez_ax = np.abs(sim3.get_dft_array(dft_ax, mp.Ez, 0))
r_lin = np.linspace(0, 1.9, len(ez_eq))
messbar = (r_lin >= 0.2) & (r_lin <= 1.85)
r_mess = r_lin[messbar]; kr = k * r_mess

hertz_eq = lambda kr: np.sqrt((1/kr - 1/kr**3)**2 + 1/kr**4)
```

```
hertz_ax = lambda kr: np.sqrt(1/kr**4 + 1/kr**6)
fern = kr > 6
A_eq = np.mean(ez_eq[messbar][fern] / hertz_eq(kr[fern]))
A_ax = np.mean(ez_ax[messbar][fern] / hertz_ax(kr[fern]))

fig, ax = plt.subplots(figsize=(7.0, 4.4))
r_theo = np.linspace(0.04, 1.9, 400); kr_t = k * r_theo
ax.loglog(r_theo, A_eq * hertz_eq(kr_t), "b-", lw=1.3,
          label="Hertz-Formel, äquatorial")
ax.loglog(r_theo, A_ax * hertz_ax(kr_t), "g-", lw=1.3,
          label="Hertz-Formel, axial")
ax.loglog(r_mess[::3], ez_eq[messbar][::3], "bo", ms=4.5, mfc="none",
          label="Meep, äquatorial")
ax.loglog(r_mess[::3], ez_ax[messbar][::3], "gs", ms=4.5, mfc="none",
          label="Meep, axial")
ax.loglog(r_theo, A_eq / kr_t, "k--", lw=0.9, label=r"$\propto 1/r$")
ax.loglog(r_theo, A_ax / kr_t**2, "k:", lw=1.1, label=r"$\propto 1/r^2$")
ax.axvline(1 / k, color="orange", lw=1.4)
ax.text(1 / k * 1.07, ax.get_ylim()[0] * 1.5, r"$r = \lambda/2\pi$",
        color="orange", rotation=90, va="bottom")
ax.set_xlabel("Abstand r (λ)"); ax.set_ylabel("|E| (relativ)")
ax.set_title("Drei Stockwerke: Nah- und Fernfeld des Hertz-Dipols")
ax.legend(fontsize=8); ax.grid(alpha=0.3, which="both")
plt.show()

steigung_eq = np.polyfit(np.log(r_mess[fern]),
                         np.log(ez_eq[messbar][fern]), 1)[0]
steigung_ax = np.polyfit(np.log(r_mess[fern]),
                         np.log(ez_ax[messbar][fern]), 1)[0]
print(f"log-log-Steigung im Fernbereich: äquatorial {steigung_eq:.3f} "
      f"(Soll −1), axial {steigung_ax:.3f} (Soll −2)")
```

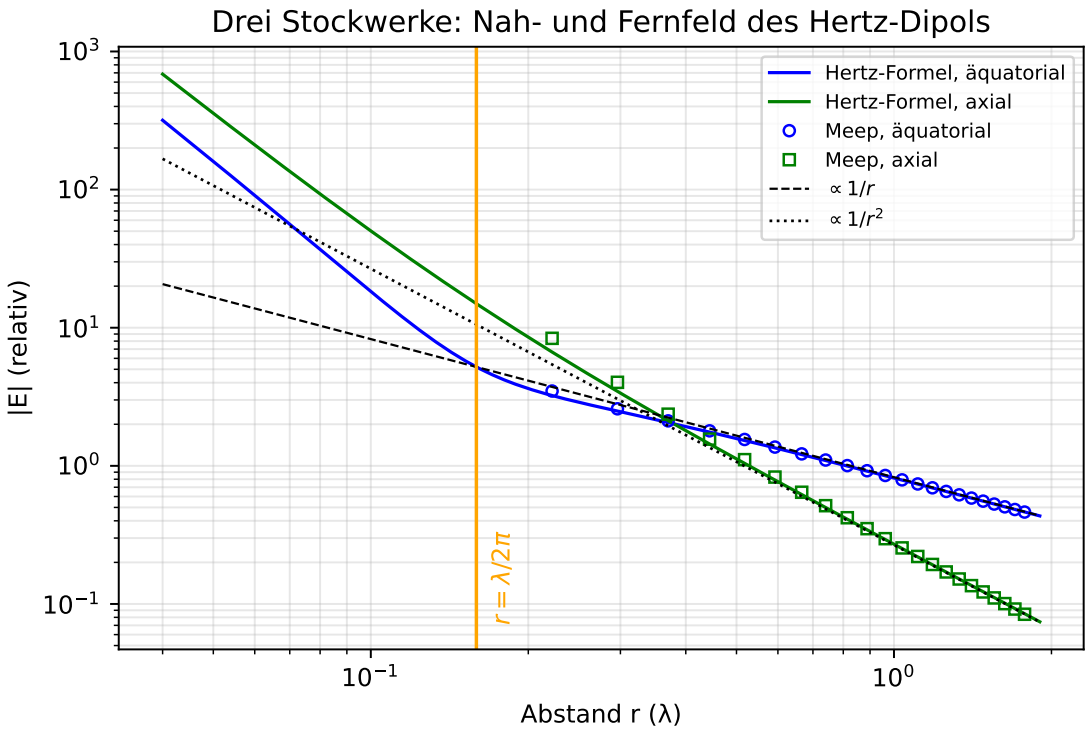


Abbildung 21.4: |E|(r) des Hertz-Dipols im log-log-Bild, äquatorial (blau) und axial (grün), je mit der exakten Hertz-Kurve als Linie und den Messpunkten aus dem 3D-Lauf. Die gestrichelte 1/r-Asymptote trägt die äquatoriale Kurve in der Ferne, die gepunktete $1/r^2$-Asymptote die axiale — längs der Achse gibt es keinen Strahlungsterm. Die orangefarbene Senkrechte markiert die Nahfeld-Fernfeld-Grenze r = λ/2π, an der die Stockwerke wechseln. Messpunkte erst ab r = 0,2 λ: näher an der Quelle ist die eine Gitterzelle der Punktquelle keine Punktquelle mehr.

```
log-log-Steigung im Fernbereich: äquatorial -0.996 (Soll −1), axial -2.053 (Soll −2)
```

Beide Messreihen liegen auf ihren Kurven, die Steigungen im Fernbereich treffen -1 und -2 — das Knick-Argument und die 130 Jahre alte Formel bestehen den Gittertest. Was die Grenze $\lambda/2\pi$ *praktisch* bedeutet, zeigt eine Handrechnung quer durch die Frequenzen, alle mit derselben Formel $r = \lambda/2\pi = c/(2\pi f)$:

Anwendung	f	λ	Nahfeldgrenze $\lambda/2\pi$
WLAN	2,45 GHz	12,2 cm	1,9 cm
UKW-Radio	100 MHz	3 m	48 cm
Stromnetz	50 Hz	6000 km	955 km

Die letzte Zeile ist die wichtigste: Beim 50-Hz-Netz liegt die Nahfeldgrenze bei fast tausend Kilometern. **Alles, was du im Alltag an 50-Hz-Feldern erlebst — jede Leitung, jeder Trafo — ist tiefstes Nahfeld**: gebundenes Feld, das pendelt, statt zu reisen. Warum ein Stromnetz trotzdem (verschwindend wenig) strahlt und warum man dafür quantitativ mehr braucht als diese Tabelle, klärt Kapitel 25.

21.4.1 Pendeln oder reisen: Blind- und Wirkleistung

„Gebundenes Feld, das pendelt" kann man messen. Die DFT-Amplituden sind komplexe Zahlen (Kapitel 15) — sie tragen die Phasenlage. Daraus baut man den **komplexen Poynting-Vektor** $S_r = -\frac{1}{2}E_\theta H_\varphi^*$: Sein Realteil ist die *Wirkleistung*, die im Zeitmittel tatsächlich nach außen fließt; sein Imaginärteil ist die *Blindleistung* — Energie, die zweimal pro Periode hinaus- und wieder hineinschwappt, ohne je anzukommen (das Wort kennst du aus dem Schwingkreis-Umfeld von Kapitel 19; hier siehst du das Feld-Original). Für den Hertz-Dipol sagt die Theorie ein verblüffend schlichtes Verhältnis voraus:

$$\frac{\text{Blind}}{\text{Wirk}} = \frac{\operatorname{Im} S_r}{\operatorname{Re} S_r} = \frac{1}{(kr)^3}.$$

Bei $r = \lambda/2\pi$ (also $kr = 1$) pendelt **genauso viel, wie fließt** — die Nahfeldgrenze ist keine Konvention, sondern der Ort dieses Gleichstands.

```
# von oben: sim3, dft_eq, r_lin, k (3D-Lauf + Profil)
ez_c = sim3.get_dft_array(dft_eq, mp.Ez, 0)
hy_c = sim3.get_dft_array(dft_eq, mp.Hy, 0)
# np.conj: konjugiert komplex (Vorzeichen des Imaginärteils kippen) —
# E·H* trennt Wirkanteil (Re) und Blindanteil (Im)
S_radial = -0.5 * ez_c * np.conj(hy_c)
# fürs Verhältnis dürfen die Punkte weiter nach innen als beim
# Profil — die Quellzellen-Abweichung ist hier Teil der Geschichte
nah_dran = (r_lin >= 0.13) & (r_lin <= 1.85)
r_bw = r_lin[nah_dran]
```

```
blind_zu_wirk = (np.imag(S_radial[nah_dran]) /
                 np.real(S_radial[nah_dran]))

fig, ax = plt.subplots(figsize=(6.8, 4.2))
zeig = blind_zu_wirk > 0
ax.loglog(r_bw[zeig][::2], blind_zu_wirk[zeig][::2], "o",
          ms=4.5, mfc="none", color="purple", label="Meep (3D-Lauf)")
r_t = np.linspace(0.12, 1.9, 300)
ax.loglog(r_t, 1 / (k * r_t)**3, "k-", lw=1.2,
          label=r"Theorie $1/(kr)^3$")
ax.plot(1 / k, 1.0, "x", color="orange", ms=11, mew=2.5,
        label=r"$r = \lambda/2\pi$: Blind = Wirk")
ax.set_xlabel("Abstand r (λ)")
ax.set_ylabel("Blindleistung / Wirkleistung")
ax.set_title("Nah pendelt die Energie, fern reist sie")
ax.legend(); ax.grid(alpha=0.3, which="both")
plt.show()

# blind_zu_wirk fällt monoton mit r — der gemessene Gleichstand
# (Verhältnis = 1) per Interpolation, np.interp braucht Aufsteigendes
r_gleich = np.interp(1.0, blind_zu_wirk[::-1], r_bw[::-1])
print(f"Gleichstand Blind = Wirk gemessen bei r = {r_gleich:.3f} λ "
      f"(Theorie: λ/2π = {1/k:.3f} λ)")
```

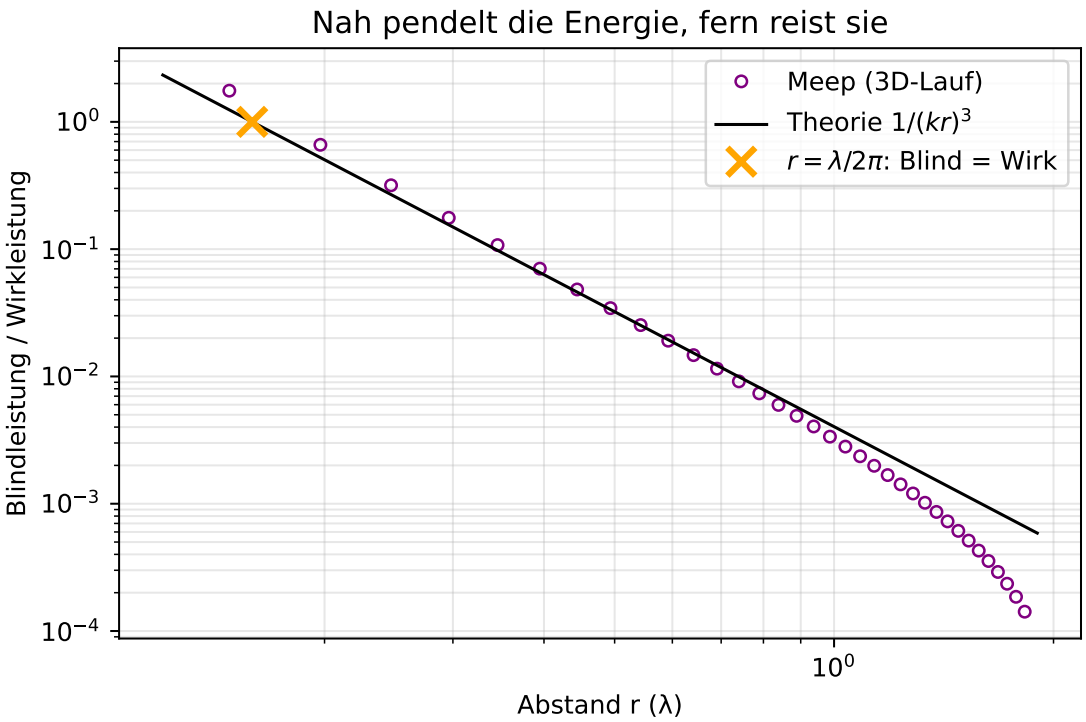


Abbildung 21.5: Das Verhältnis von Blind- zu Wirkleistung über dem Abstand, aus den komplexen DFT-Feldern des 3D-Laufs (Punkte), gegen die Theorie 1/(kr)³ (Linie). Nahe der Antenne überwiegt das Pendeln um ein Vielfaches, am Theorie-Punkt r = λ/2π herrscht Gleichstand (orangefarbenes Kreuz), weit draußen ist praktisch alle Leistung Wirkleistung auf dem Weg nach draußen. Die Messpunkte links von 0,2 λ heben sich sichtbar über die Kurve — so nah an der Quelle ist die eine Gitterzelle keine Punktquelle mehr; ganz rechts streuen sie, weil der winzige Imaginärteil im numerischen Rauschen versinkt.

```
Gleichstand Blind = Wirk gemessen bei r = 0.175 λ (Theorie: λ/2π = 0.159 λ)
```

Damit ist das Begriffspaar geerdet: **Nahfeld** heißt „die Blindleistung dominiert" — Energie schwappt zwischen Antenne und Umgebung, wie zwischen den Töpfen des Schwingkreises. **Fernfeld** heißt „die Wirkleistung dominiert" — die Energie ist unterwegs und kommt nicht wieder. Eine NFC-Karte und eine Ladespule arbeiten im Nahfeld (deshalb bricht die Verbindung nach Zentimetern ab: $1/r^3$-Felder!), dein WLAN im Fernfeld.

21.5 In welche Richtung? Das Strahlungsdiagramm

Bleibt die Richtungsfrage. Das Knick-Argument verspricht $E_\perp \propto \sin\theta$, also Strahlungsleistung $\propto \sin^2\theta$: null längs der Achse, maximal quer, dazwischen ein glatter Übergang — in 3D um die Achse rotiert ein **Donut**. Das prüfen wir mit dem Werkzeug aus Kapitel 14: Die Near-to-Far-Box, die im 3D-Lauf schon mitgelaufen ist, liefert das Fernfeld in jeder Wunschrichtung; wir tasten einen Kreis in der xz-Ebene ab (der „E-Ebene": sie enthält die Dipolachse) und tragen die Leistungsdichte über dem Winkel θ zur Achse auf.

Und weil eine echte Antenne kein Punkt ist, lassen wir den **Halbwellendipol in 3D** gegen den Hertz-Dipol antreten — die Bsp.-7-Geometrie aus dem Playground: zwei Metallzylinder, Länge zusammen $0{,}5\lambda$, Drahtradius $0{,}04\lambda$. Für ihn kennt die Lehrbuch-Theorie (dünner Draht, cos-Strom) die E-Ebenen-Form $\left[\cos(\frac{\pi}{2}\cos\theta)/\sin\theta\right]^2$ — eine etwas schlankere Acht als $\sin^2\theta$.

> Vorhersage-Punkt
>
> Der Halbwellendipol ist $\lambda/2$ groß statt punktförmig — sein Strom ist über den Draht verteilt. Bündelt das die Strahlung *stärker* oder *schwächer* als beim Punkt-Dipol? Überlege mit Kapitel 14: Was machte dort eine *breitere* Quelle (der Einzelspalt) mit ihrem Beugungsbild?

Der zweite 3D-Lauf ist deutlich billiger als der erste (gröberes Gitter reicht fürs Fernfeld — die Box mittelt ohnehin):

```
# von oben: n2f_box(), n2f_hertz, sim3 (3D-Lauf des Hertz-Dipols)
L3, SPALT3, R_DRAHT = 0.5, 0.08, 0.04
arm3 = L3 / 2 - SPALT3 / 2
zc3 = SPALT3 / 2 + arm3 / 2
sim4 = mp.Simulation(
    cell_size=mp.Vector3(6, 6, 6), boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.GaussianSource(frequency=1.0, fwidth=0.3),
                       component=mp.Ez, center=mp.Vector3())],
    resolution=20,
    geometry=[mp.Cylinder(radius=R_DRAHT, height=arm3,
                          center=mp.Vector3(0, 0, +zc3),
                          material=mp.metal),
              mp.Cylinder(radius=R_DRAHT, height=arm3,
                          center=mp.Vector3(0, 0, -zc3),
                          material=mp.metal)],
```

```
    symmetries=[mp.Mirror(mp.X), mp.Mirror(mp.Y)])
n2f_l2 = n2f_box(sim4)
sim4.run(until_after_sources=mp.stop_when_fields_decayed(
    5, mp.Ez, mp.Vector3(), 1e-4))

def e_ebene(s, n2f, n_winkel=181):
    """Fernfeld-Leistungsdichte |S|(θ) auf einem Kreis in der xz-Ebene."""
    theta = np.linspace(0.001, np.pi - 0.001, n_winkel)
    S = np.empty(n_winkel)
    for i, t in enumerate(theta):
        ff = s.get_farfield(n2f, 1000.0 * mp.Vector3(np.sin(t), 0,
                                                     np.cos(t)))
        Ex, Ey, Ez, Hx, Hy, Hz = ff
        # zeitgemittelter Poynting-Vektor S = ½ Re(E × H*) (Kap. 18)
        Sx = 0.5 * np.real(Ey * np.conj(Hz) - Ez * np.conj(Hy))
        Sy = 0.5 * np.real(Ez * np.conj(Hx) - Ex * np.conj(Hz))
        Sz = 0.5 * np.real(Ex * np.conj(Hy) - Ey * np.conj(Hx))
        S[i] = np.sqrt(Sx**2 + Sy**2 + Sz**2)
    return theta, S

theta_h, S_hertz = e_ebene(sim3, n2f_hertz)
theta_l, S_l2 = e_ebene(sim4, n2f_l2)
print("Fernfeld auf je 181 Richtungen ausgewertet.")
```

```
Fernfeld auf je 181 Richtungen ausgewertet.
```

```
# von oben: theta_h, S_hertz, theta_l, S_l2
fig = plt.figure(figsize=(6.4, 5.4))
ax = fig.add_subplot(projection="polar")        # Polarbild (Kap. 11)
voll = np.concatenate([theta_h, 2 * np.pi - theta_h[::-1]])

def rundum(S):
    Sn = S / S.max()
    return np.concatenate([Sn, Sn[::-1]])

with np.errstate(divide="ignore", invalid="ignore"):
    theo_l2 = (np.cos(np.pi / 2 * np.cos(theta_l)) /
               np.sin(theta_l))**2
theo_l2 = theo_l2 / np.nanmax(theo_l2)

ax.plot(voll, rundum(S_l2), "b-", lw=2, label="Meep: λ/2-Dipol")
ax.plot(voll, np.concatenate([theo_l2, theo_l2[::-1]]), "r--", lw=1.3,
```

```
        label="Theorie λ/2-Dipol")
ax.plot(voll, rundum(S_hertz), "g-", lw=1.6, label="Meep: Hertz-Dipol")
ax.plot(voll, rundum(np.sin(theta_h)**2), "k:", lw=1.3,
        label=r"$\sin^2\theta$ (Knick-Argument)")
ax.set_theta_zero_location("N")
ax.set_title("E-Ebene: die „Acht“ — nichts längs der Achse", pad=18)
ax.legend(loc="lower right", bbox_to_anchor=(1.12, -0.14), fontsize=8)
plt.show()

abw = np.abs(S_hertz / S_hertz.max() - np.sin(theta_h)**2).max()
print(f"Hertz-Dipol: größte Abweichung von sin²θ = {abw:.3f}")
```

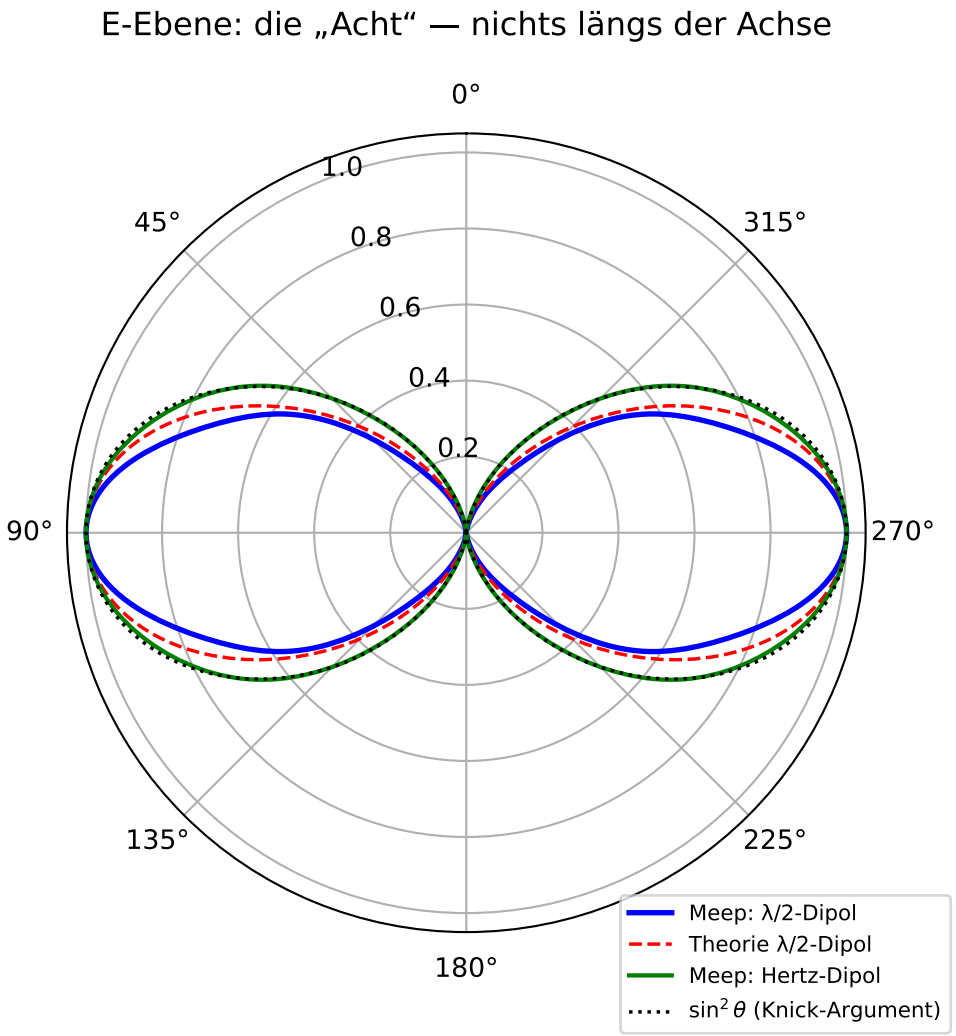


Abbildung 21.6: Das Strahlungsdiagramm in der E-Ebene (Polardiagramm; 0° = Antennenachse, die Antenne steht senkrecht im Bild). Der Hertz-Dipol (grün) trifft die $\sin^2\theta$-Vorhersage des Knick-Arguments fast perfekt, der Halbwellendipol (blau) liegt auf der schlankeren λ/2-Theoriekurve (rot gestrichelt). Beide strahlen exakt nichts längs der Achse. In der H-Ebene (quer zur Antenne, hier nicht gezeigt) sind beide Diagramme perfekte Kreise — rundum gleich stark.

```
Hertz-Dipol: größte Abweichung von sin²θ = 0.012
```

Die Antwort auf den Vorhersage-Punkt: Die ausgedehnte Antenne bündelt **stärker** — ihre Acht ist schlanker. Wie in Kapitel 14 gilt: je größer die strah-

lende Struktur (in Wellenlängen), desto schärfer die Richtwirkung. Beim Sprung von „Punkt" auf $\lambda/2$ ist der Effekt noch mild; Übung 21.3 dreht die Länge weiter auf und findet Überraschendes.

21.5.1 Eine Zahl für die Bündelung: Direktivität und dBi

Um Antennen zu vergleichen, presst man das ganze Diagramm in eine Zahl: die **Direktivität** D — wie viel stärker strahlt die Antenne in ihre beste Richtung als eine (hypothetische) Antenne, die dieselbe Leistung gleichmäßig in alle Richtungen verteilt?

$$D = \frac{4\pi\, S_{\max}}{\oint S\, d\Omega} = \frac{S_{\max}}{\langle S \rangle_{\text{Kugel}}}.$$

Das Integral über alle Richtungen haben wir komplett gemessen — unsere Diagramme sind rotationssymmetrisch um die Achse, also reicht ein gewichtetes Integral über θ:

```
# von oben: theta_h, S_hertz, theta_l, S_l2
def direktivitaet(theta, S):
    """D = 4π·S_max / ∮S dΩ; rotationssymmetrisch um die z-Achse:
    dΩ = 2π·sin(θ)dθ."""
    gesamt = 2 * np.pi * np.trapezoid(S * np.sin(theta), theta)
    return 4 * np.pi * S.max() / gesamt

D_hertz = direktivitaet(theta_h, S_hertz)
D_l2 = direktivitaet(theta_l, S_l2)
print(f"Hertz-Dipol: D = {D_hertz:.3f}  → {10*np.log10(D_hertz):.2f} dBi"
      "   (Theorie: 1,50 = 1,76 dBi)")
print(f"λ/2-Dipol:   D = {D_l2:.3f}  → {10*np.log10(D_l2):.2f} dBi"
      "   (Theorie dünner Draht: 1,64 = 2,15 dBi)")
```

```
Hertz-Dipol: D = 1.507  → 1.78 dBi   (Theorie: 1,50 = 1,76 dBi)
λ/2-Dipol:   D = 1.745  → 2.42 dBi   (Theorie dünner Draht: 1,64 = 2,15 dBi)
```

Der Hertz-Dipol landet praktisch exakt auf der Theorie. Der Halbwellendipol liegt ein paar Prozent über seinem Lehrbuchwert 1,64 — kein Messfehler, sondern Modell-Ehrlichkeit: Unser Meep-Draht ist mit $0{,}04\lambda$ Radius *dick* (reale Antennendrähte: ein Zehntel davon) und hat einen breiten Speisespalt; seine Stromverteilung ist nicht das ideale cos-Profil. Dass Meep dicke

Stäbe gut kann und dünne Drähte schlecht, ist seit der Yagi-Fallgeschichte (Kapitel 11) ein vertrautes Thema.

Die Logarithmus-Schreibweise $10 \log_{10} D$ mit der Einheit **dBi** („Dezibel über isotrop") ist die Verkehrssprache der Antennentechnik — und sie löst rückwirkend ein Rätsel aus Kapitel 11: Die NEC-Yagi hatte „8,9 dBi". Jetzt kannst du das lesen: $10^{0,89} \approx 7{,}8$-mal stärker in Hauptrichtung als der isotrope Vergleichsstrahler, also noch $7{,}8/1{,}64 \approx 4{,}8$-mal stärker als der Dipol, aus dem sie gebaut ist.

Antenne	D	dBi
isotroper Strahler (Rechengröße)	1	0
Hertz-Dipol (punktförmig)	1,5	1,76
Halbwellendipol	1,64	2,15
Yagi aus Kapitel 11 (NEC)	$\approx 7{,}8$	8,9

21.6 Wie weit trägt das? Die Reichweitenrechnung

Zum Schluss lösen wir das Versprechen aus Kapitel 9 ein: das $1/r$-Gesetz als **Reichweitenformel des Funks**. Im Fernfeld fällt E wie $1/r$, die Leistungsdichte $S \propto E^2$ also wie $1/r^2$ — dieselbe Leistung verteilt sich auf immer größere Kugelflächen. Mit Sendeleistung P und Direktivität D kommt in Hauptrichtung im Abstand r an:

$$S(r) \;=\; \frac{P \cdot D}{4\pi r^2}, \qquad E_0 \;=\; \sqrt{2\, Z_0\, S} \quad \text{mit } Z_0 = 377\ \Omega \text{ (Kapitel 6).}$$

Die Handrechnung zum WLAN-Router: $P = 100$ mW, Dipolantenne ($D = 1{,}64$), Abstand $r = 50$ m (Nachbarhaus). Erst die Leistungsdichte: $S = 0{,}1 \cdot 1{,}64/(4\pi \cdot 50^2) = 5{,}2\ \mu$W/m². Dann das Feld: $E_0 = \sqrt{2 \cdot 377 \cdot 5{,}2 \cdot 10^{-6}} = 63$ mV/m. Zum Vergleich: Sonnenlicht bringt 1000 W/m² (Kapitel 18) — das WLAN-Signal im Nachbarhaus ist **zweihundert Millionen Mal** dünner, und doch liest es dein Laptop mühelos, weil Empfänger Feldstärken bis hinunter zu Mikrovolt pro Meter verarbeiten. Funkstrecken funktionieren nicht, weil viel Leistung ankommt, sondern weil fast nichts schon reicht — und weil im Fernfeld eben nur $1/r^2$ verloren geht, nicht $e^{-r/\delta}$ wie in einem verlustbehafteten Medium (Kapitel 6) oder $1/r^6$ wie bei den gebundenen Nahfeldern.

21.7 Das Kapitel-Programm

`programme/kap21/kap21_strom_zur_welle.py` bündelt die vier Experimente eigenständig: den 2D-Halbwellendipol mit der gemessenen Strom-Stehwelle (Bauch am Speisepunkt, cos-Theorielinie samt der ehrlichen Blechstreifen-Abweichung), den 3D-Hertz-Dipol, dessen $|E|(r)$-Profil über anderthalb Dekaden auf der Hertz-Formel liegt (Fernast $1/r$ äquatorial, $1/r^2$ axial, Knie bei $\lambda/2\pi$), das Blind-zu-Wirk-Verhältnis $\mathrm{Im}(S)/\mathrm{Re}(S) = 1/(kr)^3$ aus dem komplexen Poynting-Vektor (die Nahfeld-Fernfeld-Grenze bei $\lambda/2\pi$) und die near2far-Richtdiagramme (Hertz $\sin^2\theta$ mit $D = 1{,}53$; Halbwellendipol $D = 1{,}74 \approx 2{,}4$ dBi). Laufzeit etwa vier Minuten — der 3D-Hertz-Lauf mit Auflösung 40 dominiert.

Merkkasten

- **Strahlung braucht Beschleunigung.** Eine gestoßene Ladung hinterlässt eine mit c laufende Knick-Schale; reine Geometrie liefert $E_\perp \propto a \sin\theta/r$ — einmal $1/r$ statt $1/r^2$, und die Schalenenergie kehrt nie zurück.
- **Die Antenne ist ein Resonator:** Der Strom im Halbwellendipol ist eine stehende Welle (Bauch am Speisepunkt, Knoten an den offenen Enden), das Feld schnürt sich jede Halbschwingung als geschlossene Schleife ab.
- **Einmal abgelöst, ist die Welle autark** — die Quelle kann sie weder zurückrufen noch löschen; ihre Nachricht hinkt mit c hinterher.
- **Nahfeld bis** $r \approx \lambda/2\pi$ (Blindleistung pendelt, Felder $\propto 1/r^3$), **Fernfeld** darüber (Wirkleistung reist, $E \propto 1/r$, Blind/Wirk $= 1/(kr)^3$). Längs der Dipolachse wird nichts abgestrahlt: Diagramm $\sin^2\theta$ (Punkt-Dipol).
- **Direktivität** $D = 4\pi S_{\max}/\oint S\,d\Omega$, als $10\log_{10} D$ in **dBi**; Leistungsdichte am Empfänger $S = PD/4\pi r^2$.

Roter Faden

Dieses Kapitel hat geerntet, was lange gesät war: Das **statische Dipolfeld aus Kapitel 3** wurde zum $1/r^3$-Erdgeschoss der Hertz-Formel, die **Retardierung aus Kapitel 20** zum Motor des Knick-Arguments, die **stehende Welle aus Kapitel 6** zur Stromverteilung im Draht, der **Verschiebungsstrom aus Kapitel 3** zur Erklärung, warum der Antennenstrom am

Drahtende nicht endet, sondern ins Feld wechselt. Der **Poynting-Vektor aus Kapitel 18** bekam seine komplexe Schwester (Blind- und Wirkleistung), das **Near-to-Far aus Kapitel 14** seine geschlossene 3D-Box, das **Dipol-Argument aus Kapitel 13** seine vermessene Keule, und die **2D-Ringwelle aus Kapitel 9** ihren 3D-Gegenspieler mit echtem $1/r$. Nach vorn: **Kapitel 22** fragt, wie die Leistung verlustfrei vom Sender zum Speisepunkt kommt (Leitungen, Reflexion, 50 Ω), **Kapitel 23** schaltet viele Dipole zum schwenkbaren Array zusammen, **Kapitel 24** baut aus passiven Nachbarstäben die Yagi mit Bauplan, und **Kapitel 25** rechnet aus, warum die 955-km-Zeile der Nahfeld-Tabelle das 50-Hz-Netz endgültig vom Funken freispricht.

Übungen

Ü 21.1 (Verstehen). (a) Ein Gleichstrom fließt durch einen langen geraden Draht. Die Elektronen *bewegen* sich — warum strahlt der Draht trotzdem nicht? Argumentiere mit dem Knick. (b) Und warum strahlt derselbe Draht doch ein winziges bisschen, wenn man den Strom einschaltet? (c) Berechne die Nahfeldgrenze $\lambda/2\pi$ für dein Induktionskochfeld (arbeitet bei etwa 25 kHz) und entscheide: Ist der Topf im Nah- oder Fernfeld?

Musterlösung zu Ü 21.1

(a) Gleichförmig driftende Ladungen tragen ihr Feld mit — ohne Beschleunigung kein Knick, keine Schale, kein $1/r$-Anteil. (Genauer mit Kapitel 2: Ein *konstanter* Strom macht ein *konstantes* Magnetfeld, $\partial\vec{B}/\partial t = 0$, also wird kein E-Feld induziert — die Selbsterhaltungs-Kette der Welle startet nie.)

(b) Beim Einschalten werden die Ladungen beschleunigt — genau einmal läuft eine Knick-Schale davon. Jeder Schaltvorgang funkt; deshalb knackt ein altes Radio, wenn der Kühlschrank anspringt.

(c) Der Code rechnet:

```
C_LICHT = 3e8
f = 25e3
grenze = C_LICHT / (2 * np.pi * f)
print(f"lambda = {C_LICHT/f/1000:.0f} km, "
      f"Nahfeldgrenze = {grenze/1000:.1f} km")
```

```
lambda = 12 km, Nahfeldgrenze = 1.9 km
```

Die Nahfeldgrenze liegt bei fast zwei Kilometern — der Topf (20 cm) sitzt im *tiefsten* Nahfeld. Induktives Kochen und induktives Laden sind

reine Nahfeld-Kopplungen: gebundenes Feld, keine Abstrahlung in nennenswerter Menge.

Ü 21.2 (Verstehen/Rechnen). (a) Warum ist das Strahlungsdiagramm in der H-Ebene (quer zur Antenne) ein perfekter Kreis? Ein Satz mit dem Knick-Argument genügt. (b) Ein UKW-Sender strahlt $P = 100$ kW über eine Antenne mit $D = 5$ ab. Wie groß sind Leistungsdichte und Feldstärke E_0 in 50 km Entfernung — und wie viele Größenordnungen liegt das über der Empfindlichkeit eines Autoradios (≈ 1 μV/m)?

Musterlösung zu Ü 21.2

(a) Quer zur Achse gesehen ist die Querbewegung der Ladungen aus jeder Richtung gleich gut sichtbar ($\theta = 90°$ rundum) — der Knick ist in der ganzen Ebene gleich stark, das Diagramm ein Kreis.

(b)

```
Z0 = 377.0
P, D, r = 100e3, 5.0, 50e3
S = P * D / (4 * np.pi * r**2)
E0 = np.sqrt(2 * Z0 * S)
print(f"S = {S*1e6:.0f} µW/m², E0 = {E0*1000:.0f} mV/m")
print(f"über Autoradio-Schwelle: Faktor {E0/1e-6:.0e}")
```

```
S = 16 µW/m², E0 = 110 mV/m
über Autoradio-Schwelle: Faktor 1e+05
```

Rund 16 μW/m² und 110 mV/m — gut **fünf Größenordnungen** über der Schwelle. Deshalb klingt UKW auch hinter Bergen und in Tiefgaragen noch: Es ist Reserve für viele Reflexions- und Beugungsverluste da.

Ü 21.3 (Verändern). Verlängere den 3D-Dipol im Richtdiagramm-Lauf auf $L = 1{,}5\,\lambda$ (passe `arm3` und `zc3` an; die Near-to-Far-Box umschließt mit $b = 1{,}6$ auch diesen Draht noch). Sage mit der Spielwiesen-Formel vorher, wie das Diagramm aussieht, und vergleiche dann Meep gegen die Theoriekurve. Was ist aus der „Acht" geworden — und warum ist „länger = stärker gebündelt" ab $L > \lambda$ falsch?

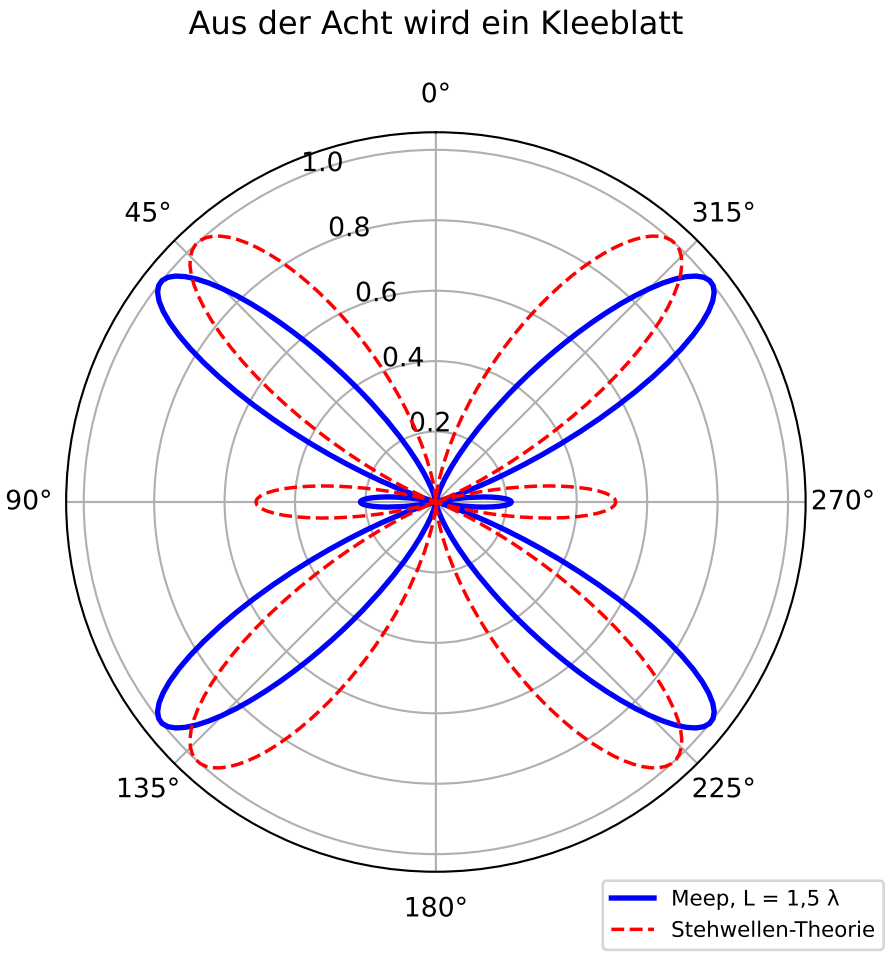
Musterlösung zu Ü 21.3

Auf einem $1{,}5\lambda$-Draht passen Stromabschnitte mit entgegengesetztem Vorzeichen (die Stehwelle wechselt alle $\lambda/2$ das Vorzeichen). Die Teil-

ströme interferieren wie gegenphasige Quellen (Ü 14.4): Quer zur Achse löschen sie sich teilweise aus, schräg addieren sie sich — die Acht zerfällt in ein **Kleeblatt** mit schrägen Hauptkeulen.

```
# von oben: n2f_box(), e_ebene(), direktivitaet() (Kapiteltext)
L_LANG = 1.5
arm_u = L_LANG / 2 - SPALT3 / 2
zc_u = SPALT3 / 2 + arm_u / 2
sim_u = mp.Simulation(
    cell_size=mp.Vector3(6, 6, 6), boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.GaussianSource(frequency=1.0, fwidth=0.3),
                       component=mp.Ez, center=mp.Vector3())],
    resolution=20,
    geometry=[mp.Cylinder(radius=R_DRAHT, height=arm_u,
                          center=mp.Vector3(0, 0, +zc_u),
                          material=mp.metal),
              mp.Cylinder(radius=R_DRAHT, height=arm_u,
                          center=mp.Vector3(0, 0, -zc_u),
                          material=mp.metal)],
    symmetries=[mp.Mirror(mp.X), mp.Mirror(mp.Y)])
n2f_u = n2f_box(sim_u)
sim_u.run(until_after_sources=mp.stop_when_fields_decayed(
    5, mp.Ez, mp.Vector3(0.3, 0, 0.5), 1e-4))
theta_u, S_u = e_ebene(sim_u, n2f_u)

kL2 = np.pi * L_LANG
theo_u = ((np.cos(kL2 * np.cos(theta_u)) - np.cos(kL2)) /
          np.sin(theta_u))**2
fig = plt.figure(figsize=(6.0, 5.0))
ax = fig.add_subplot(projection="polar")
voll = np.concatenate([theta_u, 2 * np.pi - theta_u[::-1]])
Sn = S_u / S_u.max(); Tn = theo_u / theo_u.max()
ax.plot(voll, np.concatenate([Sn, Sn[::-1]]), "b-", lw=2,
        label="Meep, L = 1,5 λ")
ax.plot(voll, np.concatenate([Tn, Tn[::-1]]), "r--", lw=1.3,
        label="Stehwellen-Theorie")
ax.set_theta_zero_location("N")
ax.set_title("Aus der Acht wird ein Kleeblatt", pad=18)
ax.legend(loc="lower right", bbox_to_anchor=(1.12, -0.12), fontsize=8)
plt.show()
i_max = np.argmax(S_u[:len(S_u)//2])
print(f"Hauptkeule bei θ ≈ {np.degrees(theta_u[i_max]):.0f}° "
      "zur Achse (Theorie: 42,6°)")
```




```
Hauptkeule bei θ ≈ 51° zur Achse (Theorie: 42,6°)
```

Meep bestätigt das Kleeblatt — vier Keulen, Nullstellen längs der Achse und ein Einbruch quer dazu. Die gemessene Hauptkeule liegt allerdings rund acht Grad steiler als die Dünndraht-Theorie: wieder der dicke Stab, dessen Stromverteilung nicht das ideale Stehwellen-Profil ist (wie schon bei der Direktivität des Halbwellendipols). „Mehr Draht" bündelt jedenfalls nur, solange der Strom überall gleichphasig schwingt — darüber zerlegt Interferenz die Keule. Wer wirklich bündeln will, schaltet viele *getrennt gespeiste* Dipole zusammen: Kapitel 23.

Ü 21.4 (Übertragen). Die Antenne im Auto(dach) ist oft nur *ein* Stab — ein **Monopol** über einer Metallfläche. Simuliere in der 2D-Welt des Kapitels einen einzelnen $0{,}22\lambda$-Arm senkrecht auf einem PEC-Boden (ein Metallblock als Boden, Quelle im Fußpunkt) und vergleiche das Feldbild der oberen

Halbebene mit dem des Dipols. Erkläre den Befund mit einem Spiegelbild-Argument — und damit, warum die Karosserie zur Antenne gehört.

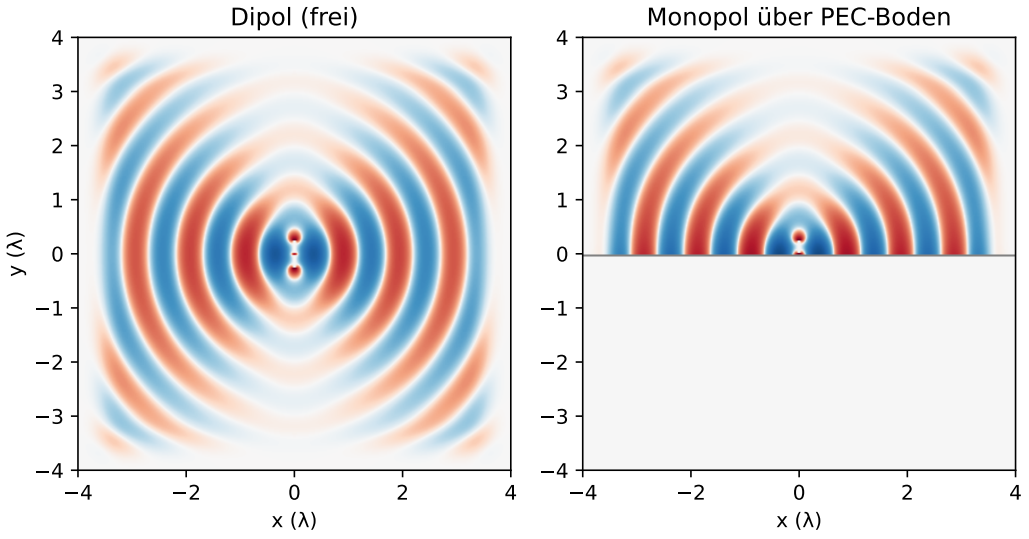
Musterlösung zu Ü 21.4

Der PEC-Boden erzwingt $E_{\parallel} = 0$ — dieselbe Randbedingung, die ein **Spiegelbild** des Arms samt *gespiegeltem* Strom erfüllen würde (Kapitel 6: die PEC-Wand als Spiegel). Für die obere Halbebene ist der Monopol über Blech daher exakt äquivalent zum vollen Dipol; der Boden liefert den fehlenden Arm gratis.

```
# von oben: DRAHT_B, SPALT, AUFL, Ey_t (Kapiteltext, 2D-Dipol-Lauf)
boden_und_arm = [
    mp.Block(size=mp.Vector3(mp.inf, 2.0, mp.inf),
             center=mp.Vector3(0, -1.0 - SPALT / 2),
             material=mp.metal),
    mp.Block(size=mp.Vector3(DRAHT_B, 0.22, mp.inf),
             center=mp.Vector3(0, SPALT / 2 + 0.11),
             material=mp.metal),
]
sim_m = mp.Simulation(
    cell_size=mp.Vector3(8.0, 8.0), boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.ContinuousSource(frequency=1.0, width=1.0),
                       component=mp.Ey, center=mp.Vector3())],
    resolution=AUFL, geometry=boden_und_arm)
mono_bilder = []
sim_m.run(until=20.0)
sim_m.run(mp.at_every(1.0 / 16.0,
                      lambda s: mono_bilder.append(
                          s.get_array(component=mp.Ey))), until=1.0)
ey_mono = np.array(mono_bilder)[:16]

fig, (a1, a2) = plt.subplots(1, 2, figsize=(7.3, 3.7))
x8 = np.linspace(-4, 4, ey_mono.shape[1])
# Schnappschüsse zur gleichen Quellenphase (letztes Bild beider Läufe)
for ax, feld, titel in ((a1, Ey_t[15], "Dipol (frei)"),
                        (a2, ey_mono[15], "Monopol über PEC-Boden")):
    g = np.arcsinh(feld / (np.abs(feld).max() / 30))
    ax.imshow(g.T, origin="lower", cmap="RdBu",
              extent=[-4, 4, -4, 4],
              vmin=-np.abs(g).max(), vmax=np.abs(g).max())
    ax.set_title(titel); ax.set_xlabel("x (λ)")
a2.axhline(-SPALT / 2, color="gray", lw=1.0)   # Bodenoberkante
a1.set_ylabel("y (λ)")
fig.tight_layout()
plt.show()

# quantitativ und phasenunabhängig: Amplituden-Profile (Maximum über
# die Periode) auf der Linie y = 1 oberhalb des Bodens
iy1 = np.argmin(np.abs(x8 - 1.0))
p_d = np.max(np.abs(Ey_t[:, :, iy1]), axis=0)
p_m = np.max(np.abs(ey_mono[:, :, iy1]), axis=0)
p_d, p_m = p_d / p_d.max(), p_m / p_m.max()
print("mittlere Amplituden-Abweichung auf der Linie y = 1: "
      f"{np.mean(np.abs(p_d - p_m)):.3f} (normiert)")
```




```
mittlere Amplituden-Abweichung auf der Linie y = 1: 0.016 (normiert)
```

Oberhalb des Bodens stimmen die Feldbilder bis auf wenige Prozent überein (unterhalb ist beim Monopol naturgemäß nichts). Praktisch heißt das: Autodach, Blechgehäuse oder die Massefläche einer Platine sind **die halbe Antenne**. Ein $\lambda/4$-Stab über gutem Blech funkt wie ein $\lambda/2$-Dipol — und ein schlechter Massekontakt ist der häufigste Grund, warum eine Antenne „nicht geht".

Das Kleingedruckte

Warum die Punktquelle ein perfekter Hertz-Dipol ist. Meeps E_z-Punktquelle ist ein erzwungener Strom J_z in einer einzigen Gitterzelle — buchstäblich ein „Stromelement $I\,dl$", das Objekt, für das die Hertz-Lösung gilt. Deshalb trifft sie $\sin^2\theta$ und $D = 1{,}5$ so präzise, während der ausgedehnte Stab-Dipol von seiner Lehrbuchkurve leicht abweicht: Nicht die Theorie ist unscharf, sondern „dünner Draht mit cos-Strom" ist für dicke Stäbe das falsche Modell. Erst nahe der Quelle (innerhalb weniger Zellen) endet die Punkt-Idylle — eine Zelle ist dort eben doch kein Punkt, weshalb unsere Profil-Messpunkte erst bei $0{,}2\lambda$ beginnen.

Der Strahlungswiderstand wartet in Kapitel 25. Von der Schaltungsseite gesehen „verbraucht" die Antenne Leistung wie ein Widerstand — die abgestrahlte Leistung pro Speisestrom-Quadrat heißt *Strahlungswiderstand*

(beim $\lambda/2$-Dipol etwa $73\ \Omega$, eine der berühmtesten Zahlen der Funktechnik und der halbe Grund für „$50\ \Omega$“, Kapitel 22). Für die elektrisch kleine Antenne bricht er wie $(l/\lambda)^2$ ein — das ist dann das quantitative Ende der 50-Hz-Funk-Frage.

Hertz' Funkenstrecke. Hertz erzeugte seine Hochfrequenz 1886 mit Funkenentladungen — jeder Funke ein Stromstoß, also genau der „Knick” vom Kapitelanfang, nur tausendfach wiederholt. Sein Empfänger war ein Drahtring mit winziger Funkenstrecke, abgelesen im Dunkeln mit der Lupe. Die Messungen (stehende Wellen im Hörsaal, Reflexion, Polarisation, Brechung an einem Asphalt-Prisma) bestätigten Maxwells Theorie zwanzig Jahre nach ihrer Veröffentlichung — und Marconi machte aus dem „nutzlosen” Hörsaalversuch binnen zehn Jahren die Funktelegrafie.

22 Leitungen, Reflexion und Anpassung

Kapitel 18 hat eine Frage absichtlich offen gelassen. Eine Batterie, ein Schalter, eine Lampe — aber das Kabel dazwischen ist ein Ungetüm: Es läuft erst einen halben Kilometer nach rechts, dann einen halben Kilometer zurück. Lampe und Batterie stehen dabei direkt nebeneinander auf demselben Tisch. Du legst den Schalter um. Wann leuchtet die Lampe — nach der Laufzeit durchs lange Kabel? Sofort? Oder irgendetwas dazwischen? Die Frage hat 2021 als Veritasium-Video Millionen Zuschauer gespalten, und die ehrliche Antwort ist keine der griffigen Parolen, sondern eine *Kurve*: die Spannung an der Lampe als Funktion der Zeit. Am Ende dieses Kapitels rechnest du diese Kurve selbst — mit einem Simulator aus zwanzig Zeilen NumPy — und sie wird verblüffender aussehen, als beide Lager es sich vorgestellt haben.

Unterwegs sammelt das Kapitel ein, was seit Teil II überall angeschrieben steht und nie erklärt wurde: **50 Ω**. Die Zahl steht auf jedem Funkgerät, jedem Oszilloskop-Eingang, jedem Antennenkabel — und die Yagi aus Kapitel 11 hatte ausgerechnet 19 Ω am Speisepunkt, was der Bauplan mit einem „Anpassglied" kurierte, dessen Theorie wir schuldig blieben. Dieses Kapitel zahlt die Schuld: Es verbindet die Schaltkreis-Sprache aus Kapitel 19 (Spannung, Strom, Widerstand) mit der Feld-Sprache aus Kapitel 4 und 6 (Welle, Wellenwiderstand, Reflexion) — und zeigt, dass die Spiegelformel aus Kapitel 6 wortwörtlich auch für Kabel gilt.

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum ein Kabel eine *Leitung* ist: Die Telegrafengleichungen sind das 1D-Maxwell aus Kapitel 5 in Schaltkreis-Kleidern, mit Wellenwiderstand $Z_0 = \sqrt{L'/C'}$ und Tempo $v = 1/\sqrt{L'C'}$ — und du kannst Z_0 eines Koaxkabels aus seiner Geometrie ausrechnen,
2. … die Reflexion am Leitungsende mit der Spiegelformel $r = (R_L - Z_0)/(R_L + Z_0)$ vorhersagen und die drei Grenzfälle (offen, kurzgeschlossen, angepasst) im Zeitbereich deuten,

3. … mit einem Puls und einer Uhr Kabelfehler orten (Zeitbereichs-Reflektometrie) und aus dem Stehwellenmuster das SWR ablesen,
4. … einen $\lambda/4$-Transformator dimensionieren ($Z = \sqrt{Z_0 R_L}$), seine Schmalbandigkeit erklären und die 19-Ω-Yagi ans 50-Ω-Kabel anpassen,
5. … die Lampenfrage quantitativ beantworten: Die Spannung an der Last wächst als Reflexions-Treppe, und der Kirchhoff-Stromkreis ist ihr Grenzwert.

```
import numpy as np
import matplotlib.pyplot as plt
```

22.1 Die Leitung: ein Feld in Schaltkreis-Kleidern

Was ist ein Kabel, physikalisch? Kapitel 18 hat es mit der $\vec{S}$-Karte beantwortet: Bei einer Zweidraht- oder Koaxialleitung fließt die Energie **im Feld zwischen den Leitern**, nicht im Metall — das Metall führt das Feld nur, wie ein Flussbett das Wasser. Zwischen den Leitern steht ein E-Feld (quer) und um die Leiter ein B-Feld (ringsum), und beide wandern als Welle die Leitung entlang.

Nun könnte man jede Leitung als 3D-Feldproblem rechnen — aber für eine Welle, die brav längs läuft, ist das Verschwendung. Quer zur Leitung sieht das Feld an jeder Stelle x gleich *geformt* aus, nur seine *Stärke* ändert sich mit x und t. Also genügen zwei Zahlen pro Querschnitt, und es sind alte Bekannte aus Kapitel 19:

- die **Spannung** $U(x,t)$ — das E-Feld quer integriert (welchen Schubs pro Ladung das Querfeld liefert),
- der **Strom** $I(x,t)$ — das B-Feld einmal um den Innenleiter herum aufgelaufen (Durchflutungssatz, Kapitel 3).

Die Maxwell-Gleichungen für das geführte Feld werden dann zu zwei Gleichungen für U und I, den **Telegrafengleichungen**:

$$\frac{\partial U}{\partial x} = -L' \frac{\partial I}{\partial t}, \qquad \frac{\partial I}{\partial x} = -C' \frac{\partial U}{\partial t}.$$

Dabei ist C' die Kapazität pro Meter (wie viel Ladung die Leiter pro Volt Querspannung speichern) und L' die Induktivität pro Meter (wie viel Fluss der Strom um sich wickelt). Und jetzt schau genau hin — diese Struktur kennst du **Zeichen für Zeichen** aus Kapitel 4 und 5:

1D-Maxwell (Kap. 4/5)	Leitung	Bedeutung
E	U	die „elektrische" Größe
B	I	die „magnetische" Größe
μ_0	L'	Trägheit der magnetischen Seite
ε_0	C'	Speicherfähigkeit der elektrischen Seite
$c = 1/\sqrt{\mu_0\varepsilon_0}$	$v = 1/\sqrt{L'C'}$	Wellentempo
$Z_0 = \sqrt{\mu_0/\varepsilon_0} = 377\ \Omega$	$Z_0 = \sqrt{L'/C'}$	Wellenwiderstand

Alles, was wir seit Kapitel 5 über 1D-Wellen wissen, gilt deshalb wortwörtlich für Kabel: Es laufen Wellen mit dem Tempo v in beide Richtungen, auf einer laufenden Welle sind beide Größen fest verkoppelt ($U = Z_0 I$, das Gegenstück zu $E = cB$), und an jeder Stoßstelle zweier Wellenwiderstände wird mit der **Spiegelformel aus Kapitel 6** reflektiert. Der Wellenwiderstand des Vakuums, der uns seit Kapitel 6 begleitet, war nie eine Exotin der Feldtheorie — er ist dasselbe Konzept, das auf jedem Kabel als Aufdruck steht.

22.1.1 Warum ausgerechnet 50 Ω? Eine Handrechnung

Für ein Koaxkabel (Innenleiter-Durchmesser d, Schirm-Innendurchmesser D, dazwischen Kunststoff mit ε_r) liefern die Feldbilder aus Teil I die beiden Beläge: Das radiale E-Feld einer Linienladung und das Ring-B-Feld eines Stroms, jeweils integriert, ergeben

$$C' = \frac{2\pi\varepsilon_0\varepsilon_r}{\ln(D/d)}, \qquad L' = \frac{\mu_0}{2\pi}\ln(D/d) \quad\Longrightarrow\quad Z_0 = \frac{377\ \Omega}{2\pi\sqrt{\varepsilon_r}}\ln\frac{D}{d} \approx \frac{60\ \Omega}{\sqrt{\varepsilon_r}}\ln\frac{D}{d}.$$

Die Zahlen fürs Standard-Laborkabel RG-58: Innenleiter $d = 0{,}9$ mm, Schirm innen $D = 2{,}95$ mm, Polyethylen mit $\varepsilon_r = 2{,}25$. Erst der Logarithmus: $\ln(2{,}95/0{,}9) = \ln(3{,}28) = 1{,}19$. Dann der Vorfaktor: $60/\sqrt{2{,}25} = 60/1{,}5 = 40\ \Omega$. Zusammen $Z_0 = 40 \cdot 1{,}19 = 47{,}5\ \Omega$ — das ist die 50 mit Fertigungstoleranz. Und das Tempo? $v = c/\sqrt{\varepsilon_r} = c/1{,}5 = 2\cdot 10^8$ m/s — auf Datenblättern steht das als „Verkürzungsfaktor 0,66". Eine Welle braucht für 100 m Kabel also 500 ns. Diese zwei Zahlen — $Z_0 = 50\ \Omega$,

$v = 2 \cdot 10^8$ m/s — sind die Bühne des ganzen Kapitels. (Warum die Industrie gerade bei 50 gelandet ist und das Fernsehen bei 75, erzählt das Kleingedruckte.)

Bevor wir bauen, die Fehlvorstellung, um die sich alles dreht:

⚠ Naheliegende Vermutung

„Ein Kabel ist ein Verbindungsstrich im Schaltplan. Sobald der Stromkreis geschlossen ist, gelten Ohm und Kirchhoff — überall und sofort."
Warum sie naheliegt: Im Alltag stimmt sie immer. Jede Schaltung, die du je gesteckt hast, gehorchte den Kirchhoff-Regeln, und der Schaltplan kennt für ein Kabel kein Symbol — es *ist* der Strich zwischen den Bauteilen.
Was stattdessen stimmt: Der Strich ist eine Leitung mit Laufzeit. Was du an einem Ende tust, erfährt das andere Ende frühestens nach l/v — bis dahin kann dort kein Ohmsches Gesetz „wissen", was passiert ist. Kirchhoff gilt erst, wenn die Leitung kürzer ist als alles, was sich währenddessen ändert: $l \ll \lambda$ — wieder das Leitmotiv des Buchs, dass nur Verhältnisse zur Wellenlänge zählen. Bei 50 Hz ist $\lambda = 6000$ km, da ist jedes Hauskabel ein Strich. Bei einer 1-ns-Signalflanke ist die „Wellenlänge" 20 cm — da ist schon das USB-Kabel eine Leitung. Dieses Kapitel lebt in der Welt, in der der Strich Laufzeit hat; am Ende sehen wir zu, wie Kirchhoff aus ihr *herauswächst*.

22.2 Das Leitungs-Labor

Wir bauen den Simulator — und weil die Telegrafengleichungen das 1D-Maxwell in neuen Kleidern sind, ist er ein alter Bekannter: das Wellenlabor aus Kapitel 6, mit U statt E und I statt B. Neu sind allein die **Enden**. In Kapitel 6 endete die Welt an Spiegeln oder im Mur-Nichts; eine Leitung endet in *Bauteilen* — links eine Quelle mit Innenwiderstand, rechts eine Last.

Frage: Reproduziert der Simulator die Spiegelformel an einer ohmschen Last? **Bühne:** 100 m RG-58 ($Z_0 = 50\ \Omega$, $v = 2 \cdot 10^8$ m/s), aufgeteilt in 1001 U-Punkte im Abstand $\Delta x = 10$ cm, dazwischen 1000 I-Punkte (der Yee-Versatz aus Kapitel 5); Zeitschritt am magischen Punkt $S = 1$, also $\Delta t = \Delta x/v = 0{,}5$ ns. **Quelle:** links eine *Thevenin-Quelle* — ideale Spannungsquelle $U_q(t)$ hinter $R_q = 50\ \Omega$ (so sieht jeder Laborgenerator aus; und weil $R_q = Z_0$, schluckt sie zurücklaufende Echos, statt sie wieder loszuschicken). **Last:** rechts ein Widerstand R_L, den wir durchprobieren. **Messgrö-**

ßen: $U(t)$ an einer Sonde bei $x = 30$ m — erst läuft der Hinpuls vorbei, später das Echo; das Amplitudenverhältnis (mit Vorzeichen) ist r. **Erfolgskriterium:** $r = (R_L - Z_0)/(R_L + Z_0)$ für alle Lasten.

Vorher der Vorhersage-Punkt zu den zwei Extremfällen:

> Vorhersage-Punkt
>
> Am **offenen** Ende kann kein Strom fließen, am **kurzgeschlossenen** Ende keine Spannung anliegen. Welche Vorzeichen haben die beiden Echos eines positiven Spannungspulses? Und welche Bauteile aus Kapitel 6 spielten exakt dieselben Rollen — für das E-Feld?

Der Kern des Simulators in zwei Zeilen plus zwei Rändern:

```
N = 1001                    # U-Punkte; I lebt auf den 1000 Lücken
DX = 0.1                    # m  → 100 m Leitung
V_TEMPO = 2.0e8             # m/s (Koax: Verkürzungsfaktor 0,66)
DT = DX / V_TEMPO           # S = 1: der magische Zeitschritt (Kap. 5/8)
Z0 = 50.0                   # Ohm
X = np.arange(N) * DX

def leitung(schritte, signal, r_quelle, r_last, z_profil=None,
            sonden=(), merke=(), huelle_ab=None):
    """Telegrafen-FDTD: U auf ganzen, I auf halben Gitterpunkten."""
    # np.full(n, wert): Array voller gleicher Werte – überall Z0,
    # außer ein eigenes Profil ist gegeben (für Stoßstellen später)
    z = np.full(N, Z0) if z_profil is None else z_profil
    c_strich = 1.0 / (z * V_TEMPO)         # C' auf den U-Punkten
    l_strich = 0.5 * (z[1:] + z[:-1]) / V_TEMPO   # L' auf den I-Punkten
    U, I = np.zeros(N), np.zeros(N - 1)
    a_q = 2 * DT / (c_strich[0] * DX)      # Randfaktoren: dazu gleich
    a_l = 2 * DT / (c_strich[-1] * DX)     # mehr im Text
    pegel = {s: [] for s in sonden}
    bilder, huelle = {}, np.zeros(N)
    for n in range(1, schritte + 1):
        # innen: wortwörtlich der Kapitel-5-Kern (E→U, B→I)
        I -= DT / (l_strich * DX) * np.diff(U)
        U[1:-1] -= DT / (c_strich[1:-1] * DX) * np.diff(I)
        # links: Thevenin-Quelle (halbe Zelle, semi-implizit)
        u_q = signal(n * DT)
        U[0] = ((1 - a_q / (2 * r_quelle)) * U[0]
                + a_q * (u_q / r_quelle - I[0])) / (1 + a_q / (2 * r_quelle))
        # rechts: ohmsche Last (halbe Zelle, semi-implizit)
```

```
        if r_last == np.inf:                # offenes Ende: R → ∞
            U[-1] = U[-1] + a_l * I[-1]
        else:
            U[-1] = ((1 - a_l / (2 * r_last)) * U[-1]
                     + a_l * I[-1]) / (1 + a_l / (2 * r_last))
        for s in sonden:
            pegel[s].append(U[s])
        if n in merke:
            bilder[n] = U.copy()
        if huelle_ab is not None and n >= huelle_ab:
            huelle = np.maximum(huelle, np.abs(U))
    return U, I, pegel, bilder, huelle

print(f"Labor bereit: {N*DX:.0f} m, Δt = {DT*1e9:.1f} ns, "
      f"Laufzeit einfach = {N*DX/V_TEMPO*1e9:.0f} ns")
```

```
Labor bereit: 100 m, Δt = 0.5 ns, Laufzeit einfach = 500 ns
```

Die zwei Innen-Zeilen sind der Kapitel-5-Leapfrog mit neuen Namen — die brauchen keine neue Anatomie. Die **Ränder** schon, denn dort steckt die neue Physik. Auseinandergefaltet für die Last (die Quelle funktioniert spiegelbildlich): Der letzte U-Punkt vertritt nur eine **halbe Gitterzelle** — rechts von ihm ist keine Leitung mehr, sondern das Bauteil. Diese halbe Zelle ist ein kleiner Kondensator mit der Ladungsbilanz

$$\underbrace{C' \frac{\Delta x}{2} \frac{dU_{\text{Ende}}}{dt}}_{\text{halbe Zelle lädt sich}} = \underbrace{I_{\text{letzte}}}_{\text{was die Leitung anliefert}} - \underbrace{U_{\text{Ende}}/R_L}_{\text{was die Last abzweigt}} ,$$

nichts als Knotenregel plus Ohm. Daraus wird die Code-Zeile, indem man dU/dt als Differenzenquotient schreibt und — der Trick aus der Wärmebilanz von Kapitel 19 — U_{Ende} im Ohm-Term **zeitzentriert** mittelt (halb alter, halb neuer Wert): Das sortiert sich zu dem Bruch mit den $(1 \mp a/2R)$-Faktoren im Code und bleibt auch für sehr kleine R_L stabil. Der Faktor $a = 2\Delta t/(C'\Delta x)$ ist dabei nur die Abkürzung für „Zeitschritt geteilt durch halbe Zellkapazität".

Jetzt die Eichung — fünf Lasten, eine Formel:

```
# von oben: leitung(), DT, Z0 (Leitungs-Labor)
T_PULS = 60e-9
puls = lambda t: np.exp(-((t - T_PULS) / 12e-9)**2)
```

```
SONDE = 300                                # Sonde bei x = 30 m

print("Last            r gemessen   Spiegelformel")
for name, r_l in (("angepasst (50)", 50.0), ("offen      ", np.inf),
                  ("kurz       ", 1e-9), ("200 Ω      ", 200.0),
                  ("19 Ω (Yagi)", 19.0)):
    _, _, pegel, _, _ = leitung(13000, puls, 50.0, r_l, sonden=(SONDE,))
    u = np.array(pegel[SONDE])
    t = np.arange(1, 13001) * DT
    hin = u[t < 400e-9]                    # Hinlauf passiert die Sonde
    echo = u[(t > 700e-9) & (t < 1500e-9)] # das Echo (hin + zurück)
    r_mess = echo[np.argmax(np.abs(echo))] / hin[np.argmax(np.abs(hin))]
    r_theo = (0.0 if r_l == 50 else
              1.0 if r_l == np.inf else (r_l - Z0) / (r_l + Z0))
    print(f"{name}      {r_mess:+.5f}      {r_theo:+.5f}")
```

```
Last            r gemessen   Spiegelformel
angepasst (50)      -0.00000      +0.00000
offen            +1.00000      +1.00000
kurz             -1.00000      -1.00000
200 Ω            +0.60000      +0.60000
19 Ω (Yagi)      -0.44928      -0.44928
```

Alle fünf exakt, auf fünf Nachkommastellen. Die Vorhersage: Das **offene Ende** wirft den Puls mit $+1$ zurück (die Ladung staut sich, die Spannung verdoppelt sich kurzzeitig am Ende) — für die Spannung ist es das, was die PEC-Wand in Kapitel 6 fürs E-Feld *nicht* war: Dort erzwang der Spiegel $E = 0$ und warf mit -1 zurück. Genau andersherum der **Kurzschluss**: Er erzwingt $U = 0$ und spiegelt mit -1 — der Kurzschluss ist die „PEC-Wand der Spannungswelt". Und die **19-Ω-Yagi** am 50-Ω-Kabel wirft $r = -0{,}449$ zurück: fast die Hälfte der Spannungsamplitude läuft rückwärts. Was das für die Leistung heißt, klärt der übernächste Abschnitt — erst machen wir aus dem Echo ein Werkzeug.

22.3 Mit der Uhr Fehler orten: TDR

Dreh die Logik um: Wenn jede Störstelle ein Echo wirft, dann verrät das Echo die Störstelle. Puls hineinschicken, auf die Uhr schauen — aus der Echo-Verzögerung t_E folgt der Ort $x = v\,(t_E - t_0)/2$ (hin *und* zurück!), aus dem Vorzeichen die Art des Fehlers. Das heißt **Zeitbereichs-Reflektometrie** (TDR), und es ist Alltagstechnik: Jeder Ethernet-Chip ortet

so Kabelbrüche, Energieversorger finden damit Fehler in kilometerlangen Erdkabeln, ohne zu graben.

Frage: Finden wir einen Kabelbruch auf 100 m blind auf den Meter genau? **Bühne:** das Labor von oben, aber das Kabel ist bei $x = 62$ m gebrochen — Leiter durchtrennt, dahinter kommt nichts mehr an; wir modellieren den Bruch als Sprung auf ein extrem hochohmiges Reststück (50 kΩ — praktisch offen). **Messgröße:** $U(t)$ an einer Sonde direkt am Anfang. **Erfolgskriterium:** Echo-Vorzeichen positiv (offenes Ende!), Ort aus der Laufzeit stimmt.

```
# von oben: leitung(), puls, T_PULS, DT, V_TEMPO, N, Z0, X
z_bruch = np.full(N, Z0)
z_bruch[620:] = 50000.0          # Bruch bei x = 62 m: dahinter „offen"
_, _, pegel_tdr, _, _ = leitung(13000, puls, 50.0, 50000.0,
                                z_profil=z_bruch, sonden=(5,))
u_tdr = np.array(pegel_tdr[5])
t = np.arange(1, 13001) * DT

i_echo = np.argmax(np.abs(u_tdr[t > 300e-9]))
t_echo = t[t > 300e-9][i_echo]
ort = V_TEMPO * (t_echo - T_PULS) / 2

fig, ax = plt.subplots(figsize=(7.0, 3.4))
ax.plot(t * 1e9, u_tdr, "b-", lw=1.3)
ax.axvline(T_PULS * 1e9, color="gray", ls=":", lw=0.9)
ax.axvline(t_echo * 1e9, color="r", ls="--", lw=1)
ax.set_xlim(0, 1500)
ax.set_xlabel("t (ns)"); ax.set_ylabel("U am Kabelanfang (V)")
ax.set_title(f"Echo nach {1e9*(t_echo - T_PULS):.0f} ns "
             f"→ Bruch bei x = {ort:.1f} m")
ax.grid(alpha=0.3)
plt.show()
```

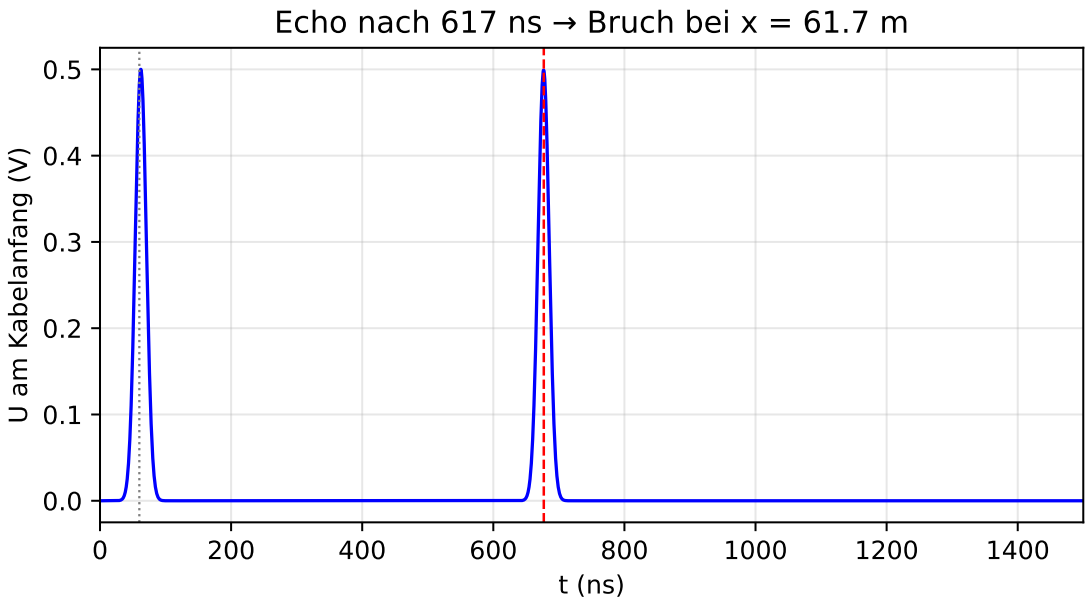


Abbildung 22.1: TDR in der Praxis: Bei t = 60 ns geht der Puls hinein (grau gepunktet), bei t = 680 ns kommt sein Echo zurück (rot gestrichelt) — positiv, also ein offenes Ende. Die Laufzeit-Rechnung v·(t_E − t_0)/2 ortet den Bruch bei 62 m, auf die Gitterzelle genau. Ein intaktes, angepasst abgeschlossenes Kabel hätte gar kein Echo geliefert.

Das positive Echo nach 620 ns sagt: offenes Ende bei $2 \cdot 10^8 \cdot 620\,\mathrm{ns}/2 = 62$ m. Kein Aufschrauben, kein Graben — ein Puls und eine Uhr.

22.4 Stehwellen und das SWR

Der Puls war die Diagnose — jetzt der Normalbetrieb: ein Sender, der **dauerhaft** sendet, auf ein Kabel mit fehlangepasster Last. Hinlaufende und reflektierte Welle überlagern sich dann permanent, und das gibt das Muster aus Kapitel 6: eine **teilstehende Welle**. Ihre Hüllkurve wogt zwischen $U_{\text{max}} = (1 + |r|)\, U_{\text{hin}}$ (wo sich beide addieren) und $U_{\text{min}} = (1 - |r|)\, U_{\text{hin}}$ (wo sie gegeneinander arbeiten). Das Verhältnis

$$\text{SWR} = \frac{U_{\text{max}}}{U_{\text{min}}} = \frac{1 + |r|}{1 - |r|}$$

heißt **Stehwellenverhältnis** und ist *die* Kennzahl der Funkpraxis: Sie ist mit einer simplen Spannungsmessung längs der Leitung bestimmbar, ohne Phasen, ohne Oszilloskop — deshalb zeigt jedes Antennenmessgerät SWR an. Perfekte Anpassung: SWR = 1. Totalreflexion: SWR = ∞.

Frage: Welches SWR macht die 19-Ω-Yagi auf dem 50-Ω-Kabel — und stimmt die Bauplan-Zahl? **Bühne:** das Labor, Dauerstrich mit $f_0 = 20$ MHz (Wellenlänge auf dem Kabel: $v/f_0 = 10$ m — zehn Bäuche auf 100 m), sanft angefahren; Last 19 Ω. **Messgröße:** die $|U|$-Hüllkurve über die letzte Periode, daraus $U_{\text{max}}/U_{\text{min}}$ (Ränder ausgenommen). **Erfolgskriterium:** Für ohmsche Lasten $R_L < Z_0$ ist $|r| = (Z_0 - R_L)/(Z_0 + R_L)$ und damit SWR $= Z_0/R_L = 50/19 = 2{,}63$.

```
# von oben: leitung(), DT, Z0, X, N
F0 = 20e6
def sinus(t):
    return min(t / 100e-9, 1.0) * np.sin(2 * np.pi * F0 * t)

_, _, _, _, huelle = leitung(24000, sinus, 50.0, 19.0, huelle_ab=23500)
innen = slice(100, 900)
swr = huelle[innen].max() / huelle[innen].min()

fig, ax = plt.subplots(figsize=(7.0, 3.4))
ax.plot(X, huelle, "b-", lw=1.4)
ax.axhline(huelle[innen].max(), color="r", ls="--", lw=0.9)
ax.axhline(huelle[innen].min(), color="r", ls="--", lw=0.9)
ax.set_xlabel("x (m)"); ax.set_ylabel("|U|-Hüllkurve (V)")
ax.set_title(f"SWR gemessen: {swr:.3f}   (Theorie 50/19 = {50/19:.3f})")
ax.grid(alpha=0.3)
plt.show()

r19 = (19 - Z0) / (19 + Z0)
print(f"|r| = {abs(r19):.3f} → Wirkleistung in die Last: "
      f"1 − |r|² = {1 - r19**2:.3f}")
```

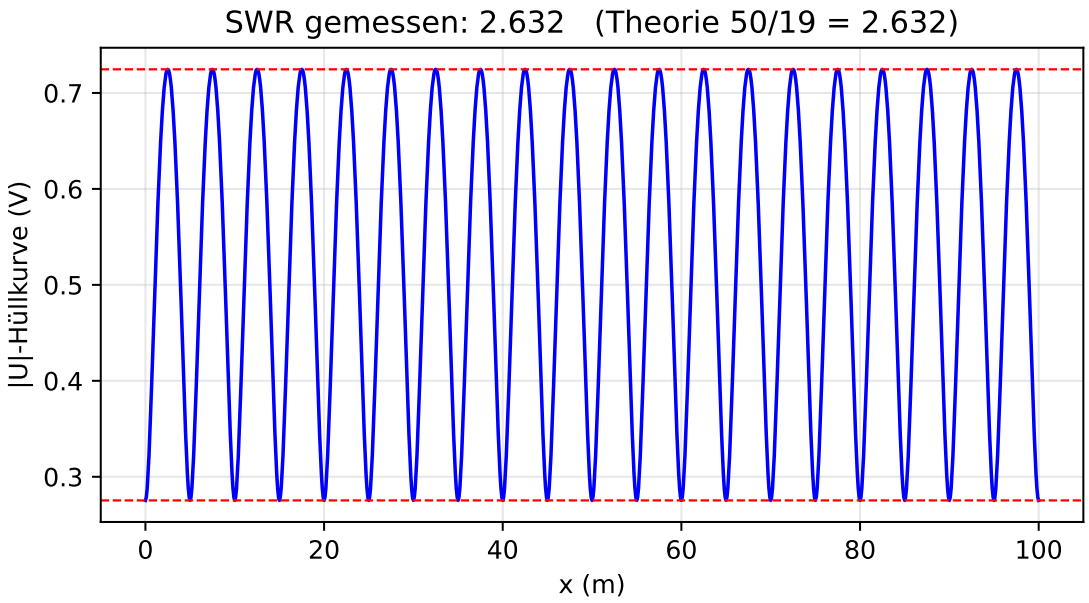


Abbildung 22.2: Die Hüllkurve der Spannung auf dem Kabel mit 19-Ω-Last im Dauerstrich: eine teilstehende Welle mit Bäuchen und flachen Tälern im λ/2-Abstand von 5 m. Das Verhältnis der gestrichelten Linien ist das SWR = 2,63 = exakt 50/19 — die Zahl, die der Yagi-Bauplan aus Kapitel 11 für den Direktanschluss nennt. Direkt an der Last (x = 100 m) liegt ein Spannungs-Tal: Die 19 Ω sind kleiner als Z_0, die Last drückt die Spannung.

```
|r| = 0.449 → Wirkleistung in die Last: 1 − |r|² = 0.798
```

Zwei Lesarten dieser Zahl. Erstens die **Leistungsbilanz**: Reflektiert wird der Anteil $|r|^2 = 0{,}20$ der Leistung — **80 % kommen trotzdem an**. Genau deshalb sagt der Yagi-Bauplan, der Direktanschluss „funktioniert zum Experimentieren": Ein SWR von 2,6 klingt dramatisch, kostet aber nur ein Fünftel der Leistung (etwa 1 dB). Wofür sich die Anpass-Mühe wirklich lohnt: Senderendstufen mögen die rücklaufende Welle nicht (sie heizt die Endtransistoren), und auf langen Kabeln läuft das Echo doppelt durch die Kabeldämpfung. Zweitens die **Topologie**: An der Last selbst sitzt ein Spannungs-*Tal*, denn $R_L < Z_0$ zieht viel Strom bei wenig Spannung — eine Last mit $R_L > Z_0$ hätte dort einen Bauch. Auch das Stehwellenmuster trägt also Information über die Last, nicht nur ihre Stärke.

22.5 Anpassen: der λ/4-Transformator

Wie wird man die Reflexion los, ohne die Last zu ändern? Die elegante Antwort ist ein Stück Leitung mit einem *dritten* Wellenwiderstand — und die Idee dahinter kennst du aus Kapitel 14: **Interferenz**. Schaltet man zwischen das 50-Ω-Kabel und die 19-Ω-Last ein Zwischenstück, gibt es *zwei* Stoßstellen und damit *zwei* Echos. Ist das Zwischenstück genau $\lambda/4$ lang, sammelt das zweite Echo auf seinem Umweg (hin und zurück durchs Stück) einen Gangunterschied von $\lambda/2$ — es kommt **gegenphasig** zum ersten zurück. Und sorgt man zusätzlich dafür, dass beide Echos gleich *stark* sind, löschen sie sich vollständig: Dafür müssen die Sprungverhältnisse an beiden Stößen gleich sein, $Z_T/Z_0 = R_L/Z_T$, also

$$Z_T \;=\; \sqrt{Z_0\,R_L} \;=\; \sqrt{50\cdot 19}\,\Omega \;=\; 30{,}8\,\Omega.$$

Das geometrische Mittel — die Anpassung „auf halbem logarithmischem Weg". Ob es funktioniert, prüfen wir breitbandig mit dem Trick aus Kapitel 10: **ein** Puls, dessen Spektrum alle Frequenzen um f_0 enthält, dann Hinlauf und Echo getrennt fourier-transformiert — der Quotient ist $|r|(f)$ für alle Frequenzen aus einem einzigen Lauf.

> **Vorhersage-Punkt**
>
> Die $\lambda/4$-Bedingung gilt *bei einer Frequenz*: Für $f_0 = 20$ MHz ist das Stück $\lambda/4 = 2{,}5$ m lang. Was passiert bei 10 MHz, wo dasselbe Stück nur $\lambda/8$ ist — und bei 40 MHz ($\lambda/2$)? Skizziere $|r|(f)$, bevor du weiterliest.

```
# von oben: leitung(), DT, Z0, X, N, F0, SONDE
LAMBDA = V_TEMPO / F0                        # 10 m auf dem Kabel
z_trafo = np.full(N, Z0)
im_stueck = (X >= 70.0) & (X < 70.0 + LAMBDA / 4)
z_trafo[im_stueck] = np.sqrt(50.0 * 19.0)    # 30,8 Ω, 2,5 m lang
z_trafo[X >= 70.0 + LAMBDA / 4] = 19.0       # dahinter 19-Ω-Welt

def traeger_puls(t):
    return np.exp(-((t - 80e-9) / 15e-9)**2) * np.sin(2 * np.pi * F0 * t)

_, _, pegel_tr, _, _ = leitung(16000, traeger_puls, 50.0, 19.0,
                               z_profil=z_trafo, sonden=(SONDE,))
u_tr = np.array(pegel_tr[SONDE])
t16 = np.arange(1, 16001) * DT
```

```
hinlauf = np.where(t16 <= 500e-9, u_tr, 0.0)   # erst kommt der Hinlauf,
ruecklauf = np.where(t16 > 500e-9, u_tr, 0.0)  # ab 500 ns nur noch Echo
freq = np.fft.rfftfreq(len(u_tr), DT)
r_von_f = np.abs(np.fft.rfft(ruecklauf)) / np.abs(np.fft.rfft(hinlauf))

fig, ax = plt.subplots(figsize=(7.0, 3.6))
fenster = (freq > 8e6) & (freq < 32e6)
ax.plot(freq[fenster] / 1e6, r_von_f[fenster], "b-", lw=1.8,
        label="mit λ/4-Transformator")
ax.axhline(abs(r19), color="r", ls="--", lw=1.2,
           label=f"ohne Anpassung: |r| = {abs(r19):.2f}")
ax.axvline(20, color="gray", ls=":", lw=0.9)
ax.set_xlabel("f (MHz)"); ax.set_ylabel("|r|")
ax.set_title("Anpassung perfekt — bei genau einer Frequenz")
ax.legend(); ax.grid(alpha=0.3)
plt.show()

i20 = np.argmin(np.abs(freq - F0))
print(f"|r|(20 MHz) = {r_von_f[i20]:.4f}")
```

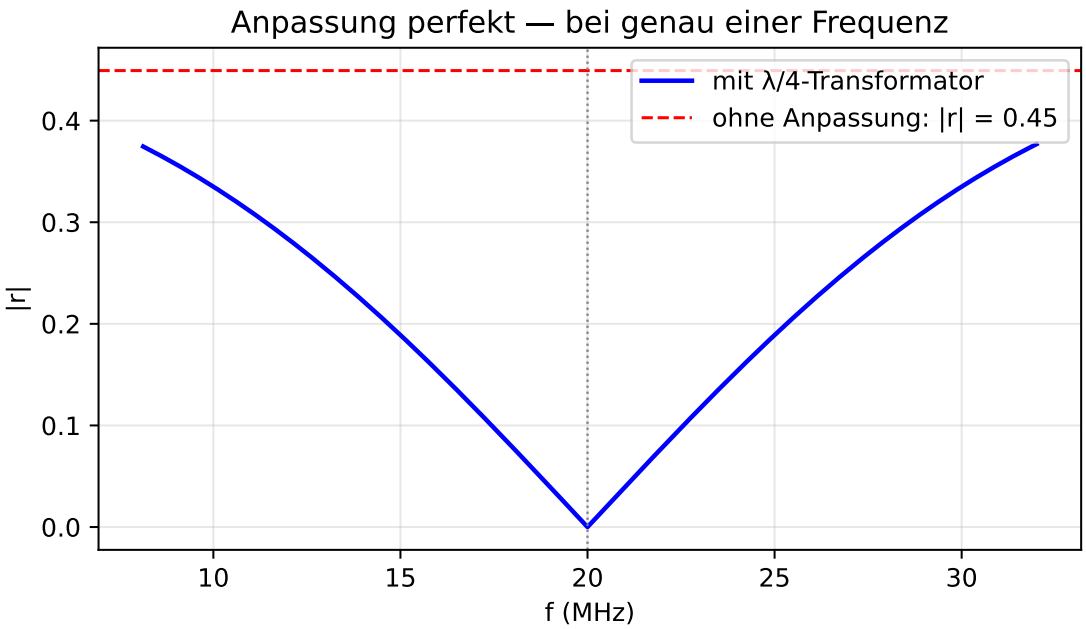


Abbildung 22.3: Der Reflexionsfaktor über der Frequenz, gemessen mit einem einzigen Puls (FFT von Hinlauf und Echo, Kapitel-10-Methode). Ohne Anpassung läge |r| konstant bei 0,45 (rot gestrichelt). Das 30,8-Ω-Stück von 2,5 m Länge drückt die Reflexion bei f_0 = 20 MHz auf exakt null — aber nur dort: Schon ±5 MHz daneben ist die Hälfte des Effekts verloren. Der λ/4-Transformator ist ein Präzisionswerkzeug für eine Frequenz, kein Breitbandheilmittel.

```
|r|(20 MHz) = 0.0000
```

Die V-Kurve beantwortet den Vorhersage-Punkt: Bei 20 MHz verschwindet die Reflexion **vollständig** ($|r| = 0{,}0000$ — Interferenz ist kein Näherungstrick), zu beiden Seiten wächst sie wieder, und bei 40 MHz ($\lambda/2$-Stück: die Echos kommen *gleichphasig* zurück) wäre die Anpassung komplett wirkungslos. Dieses Muster solltest du wiedererkennen: Es ist die Physik der **Antireflex-Beschichtung** auf Brillengläsern und Kameraobjektiven — eine $\lambda/4$-Schicht mit dem geometrischen Mittel der Brechungsindizes, perfekt im Grünen, schwächer am Rand des Spektrums (deshalb schimmern vergütete Linsen violett: Rot und Blau überleben). Kapitel 27 baut daraus mehrlagige Spiegel und Filter. Und die Querverbindung nach hinten: Die σ-**Rampe aus Kapitel 6** — sanft statt schlagartig wechseln — ist die *breitbandige* Schwester dieser Idee; Übung 22.3 lässt beide gegeneinander antreten.

22.6 Die Lampe: Kirchhoff als Grenzwert

Jetzt die Eingangsfrage. Wir bauen sie im Labor nach, mit einer ehrlichen Vereinfachung: Unser Simulator kennt nur die Leitung selbst (das TEM-Feld *zwischen* den Leitern — die zusätzliche direkte Funk-Kopplung zwischen Batterie und Lampe über die Luft ist Kapitel-21-Physik und kommt am Ende zu Wort). **Bühne:** Batterie $U_q = 1$ V mit Innenwiderstand $R_q = 25\ \Omega$, Schalter mit sanfter Flanke (20 ns — kein realer Schalter springt unendlich schnell, und ein Ein-Schritt-Sprung würde die Nyquist-Zickzack-Mode des Gitters anregen, Kapitel 8/10), dann 100 m Kabel, am Ende die „Lampe" als 200-Ω-Widerstand. Beachte: **Quelle und Last sind beide fehlangepasst** ($25 \neq 50 \neq 200$) — Echos werden also an *beiden* Enden weitergereicht. **Messgröße:** $U(t)$ an der Lampe. **Erfolgskriterium:** Kirchhoff am Ende — $U_\infty = U_q R_L/(R_L + R_q) = 200/225 = 0{,}889$ V.

> Vorhersage-Punkt
>
> Drei Kandidaten für die Lampenspannung kurz nach dem ersten Eintreffen der Welle (t knapp über 500 ns): (a) noch null, (b) schon der Kirchhoff-Wert 0,889 V, (c) etwas anderes — größer? kleiner? Rechne mit, was die Welle *mitbringen* kann: Sie startet als Spannungsteiler aus R_q und Z_0 (nicht R_L! — von der Last weiß sie noch nichts) und wird an der Lampe mit $1 + r_L$ „empfangen".

```
# von oben: leitung(), DT, Z0, N
R_Q, R_LAMPE = 25.0, 200.0
T_AN = 20e-9
def schalter(t):
    # sin²-Rampe: startet exakt bei 0, glatt — dann konstant 1
    return np.sin(np.pi * t / (2 * T_AN))**2 if t < T_AN else 1.0

_, _, pegel_l, _, _ = leitung(30000, schalter, R_Q, R_LAMPE,
                              sonden=(N - 1,))
u_lampe = np.array(pegel_l[N - 1])
t30 = np.arange(1, 30001) * DT

# die Reflexions-Treppe von Hand: Startwelle = Spannungsteiler aus
# R_q und Z0; jeder Umlauf multipliziert mit r_Lampe·r_Quelle
r_q = (R_Q - Z0) / (R_Q + Z0)          # = −1/3
r_l = (R_LAMPE - Z0) / (R_LAMPE + Z0)  # = +0,6
u_start = Z0 / (R_Q + Z0)              # = 2/3 V
T_LAUF = N * DX / V_TEMPO              # 500 ns
stufen_t, stufen_u = [T_LAUF], [0.0]
```

```
for k in range(8):
    u_k = u_start * (1 + r_l) * sum((r_l * r_q)**j for j in range(k + 1))
    stufen_t += [(2 * k + 1) * T_LAUF, (2 * k + 3) * T_LAUF]
    stufen_u += [u_k, u_k]

fig, ax = plt.subplots(figsize=(7.2, 3.8))
ax.plot(t30 * 1e6, u_lampe, "b-", lw=1.6, label="Simulation: U an der Lampe")
ax.plot(np.array(stufen_t) * 1e6, stufen_u, "r--", lw=1.1,
        label="Reflexions-Treppe (Theorie)")
ax.axhline(R_LAMPE / (R_LAMPE + R_Q), color="gray", ls=":",
           label="Kirchhoff: $R_L/(R_L+R_q)$ = 0,889 V")
ax.set_xlim(0, 8); ax.set_ylim(-0.05, 1.18)
ax.set_xlabel("t (µs)"); ax.set_ylabel("U (V)")
ax.set_title("Die Lampe sieht eine Treppe — Kirchhoff ist ihr Grenzwert")
ax.legend(fontsize=8, loc="center right")
ax.grid(alpha=0.3)
plt.show()

print("Stufe  Simulation  geometrische Reihe")
for k, t_test in enumerate((0.7e-6, 1.7e-6, 2.7e-6, 3.7e-6)):
    theo = u_start * (1 + r_l) * sum((r_l * r_q)**j for j in range(k + 1))
    i = np.argmin(np.abs(t30 - t_test))
    print(f"  {k+1}     {u_lampe[i]:.4f}      {theo:.4f}")
print(f"Endwert {u_lampe[-1]:.4f}   Kirchhoff {R_LAMPE/(R_LAMPE+R_Q):.4f}")
```

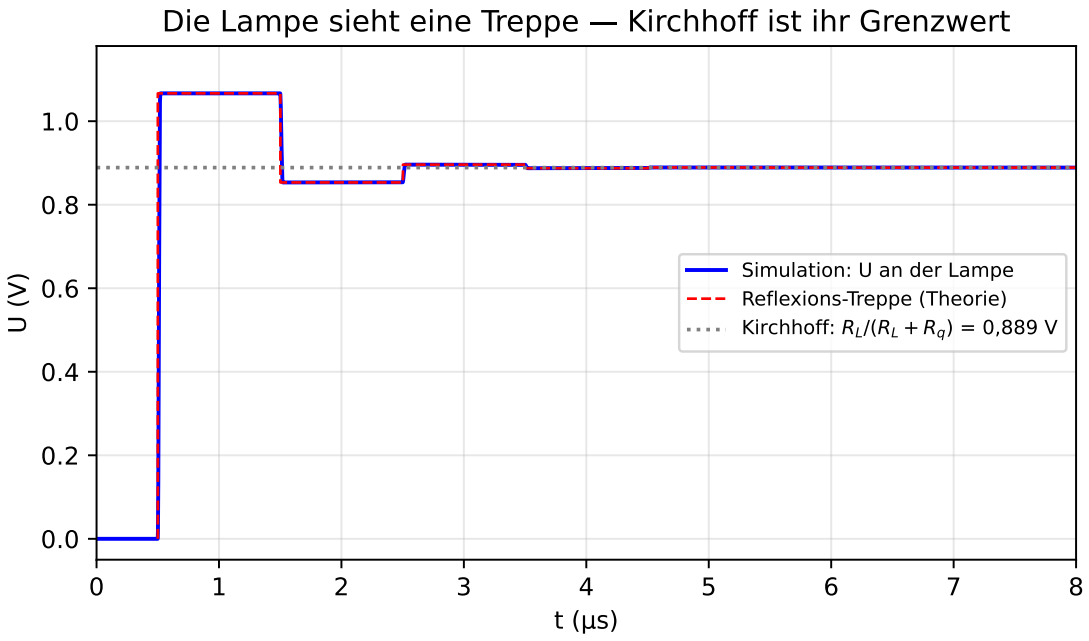


Abbildung 22.4: Die Antwort auf die Lampenfrage: Die Spannung an der Lampe (blau) bleibt exakt eine Laufzeit lang null, springt dann auf 1,07 V — ÜBER den Endwert! — und nähert sich in einer Treppe mit 1 µs Stufenbreite (ein Echo-Umlauf) dem Kirchhoff-Wert 0,889 V (grau gepunktet). Jede Stufe liegt exakt auf der Reflexions-Theorie (rot gestrichelt): die geometrische Reihe der an beiden Enden hin- und herreflektierten Teilwellen. Der vertraute Stromkreis aus Kapitel 19 ist der Grenzwert dieser Treppe.

```
Stufe  Simulation  geometrische Reihe
  1     1.0667      1.0667
  2     0.8533      0.8533
  3     0.8960      0.8960
  4     0.8875      0.8875
Endwert 0.8889   Kirchhoff 0.8889
```

Die Auflösung des Vorhersage-Punkts ist Kandidat (c) — und zwar **größer**: Die erste Stufe ist $u_{\text{start}}(1 + r_L) = \frac{2}{3} \cdot 1{,}6 = 1{,}067$ V, *über* dem Endwert. Eine Laufzeit lang weiß die Welle nichts von der 200-Ω-Lampe; sie startet mit dem Spannungsteiler aus R_q und dem **Wellenwiderstand** — das Kabel verhält sich in der ersten halben Mikrosekunde exakt wie ein 50-Ω-Widerstand, obwohl nirgends einer verbaut ist. Erst die Echos handeln dann schrittweise aus, was Kirchhoff „schon immer wusste”: Stufe um Stufe, jede

um den Umlauffaktor $r_L\, r_q = -0{,}2$ kleiner und abwechselnd über und unter dem Ziel, kollabiert die Wellenlösung auf die Stromkreislösung. **Ohm und Kirchhoff sind keine Grundgesetze neben Maxwell — sie sind das, was von Maxwell übrig bleibt, wenn alle Echos verklungen sind.** Für die 100-m-Leitung dauert das ein paar Mikrosekunden; für die 30-cm-Strippen auf deinem Steckbrett ein paar Nanosekunden — deshalb merkt der Alltag nichts davon.

Und die Veritasium-Pointe mit der Lampe **1 m neben** der Batterie? In unserem Leitungsmodell bleibt die Lampe exakt eine Kabel-Laufzeit dunkel, egal wie nah sie steht. Im realen Experiment kommt hinzu, was Kapitel 21 vermessen hat: Der Stromstoß beim Einschalten ist ein *Knick* — Batterieschleife und Lampenschleife sind auch zwei winzige Antennen, und ein sehr kleiner Teil der Energie springt nach $1\,\mathrm{m}/c \approx 3$ ns direkt durch die Luft hinüber. „Die Lampe leuchtet sofort ein bisschen" und „der volle Strom braucht die Leitung" sind beide wahr — es ist dieselbe Treppe, nur mit einem winzigen Funken-Vorboten auf Stufe null.

22.7 Das Kapitel-Programm

`programme/kap22/kap22_leitungen.py` bündelt alle Befunde eigenständig, jeden gegen die Theorie geprüft: die Eichung der Reflexionsfaktoren auf die Spiegelformel (offen $+1$, kurz -1, angepasst 0, 19 Ω $\to$ $-0{,}449$ wie bei der Yagi), die TDR-Ortung des Kabelbruchs auf die Gitterzelle genau, das Stehwellenverhältnis SWR $= 2{,}632 = 50/19$ mit 80 % Wirkleistung, den $\lambda/4$-Transformator mit $|r|(f_0) = 0$ (und seiner schmalbandigen V-Kurve) und die Lampen-Treppe, deren Stufen auf fünf Stellen auf der geometrischen Reihe gegen den Kirchhoff-Endwert $0{,}889$ laufen — die erste Stufe $1{,}067$ darüber. Reines NumPy, kein Meep; Laufzeit wenige Sekunden.

Merkkasten

- **Jedes Kabel ist eine Leitung:** Die Telegrafengleichungen sind 1D-Maxwell in U/I-Sprache, mit $Z_0 = \sqrt{L'/C'}$ (Koax: $\approx 60/\sqrt{\varepsilon_r} \cdot \ln(D/d)$ Ω) und $v = 1/\sqrt{L'C'}$. „Verbindungsstrich" ist der Grenzfall $l \ll \lambda$.
- **Reflexion am Ende = Spiegelformel:** $r = (R_L - Z_0)/(R_L + Z_0)$; offen $+1$, kurz -1, angepasst 0. Reflektierte Leistung $|r|^2$, angekommene $1 - |r|^2$.
- **TDR:** Echo-Laufzeit ortet die Störstelle ($x = v\,\Delta t/2$), das Vorzeichen nennt ihre Art. **SWR** $= (1 + |r|)/(1 - |r|)$ — für ohmsche

Lasten unter Z_0 einfach Z_0/R_L.

- **λ/4-Transformator:** $Z_T = \sqrt{Z_0 R_L}$ löscht die Reflexion durch Interferenz der beiden Stoß-Echos — exakt, aber nur bei f_0 (dieselbe Physik wie die Antireflex-Schicht auf Brillengläsern).
- **Kirchhoff ist der Grenzwert:** Nach einem Schaltvorgang nähert sich die Lastspannung dem Stromkreiswert als Reflexions-Treppe (Umlauffaktor $r_L r_q$); die erste Stufe sieht nur R_q und Z_0 — das Kabel wirkt anfangs als Widerstand von Z_0 Ohm.

Roter Faden

Rückwärts: Die **Spiegelformel aus Kapitel 6** wurde zur Last-Reflexion (und der Kurzschluss zur PEC-Wand der Spannungswelt), der **Kapitel-5-Leapfrog** läuft hier wortwörtlich in neuen Variablen, die **Interferenz aus Kapitel 14** löscht im λ/4-Stück die Reflexion, die **FFT-Methode aus Kapitel 10** misst $|r|(f)$ aus einem einzigen Puls, die **zeitzentrierte Bilanz aus Kapitel 19** stabilisiert die Last-Randbedingung, und die **Lampenfrage aus Kapitel 18** hat ihre Kurve bekommen — mit dem **Knick aus Kapitel 21** als Erklärung für den winzigen Luftweg-Vorboten. Der **Yagi-Bauplan aus Kapitel 11** ist um seine Theorie ergänzt: 19 Ω heißt SWR 2,6, heißt 80 % ohne und 100 % mit λ/4-Anpassung. Nach vorn: **Kapitel 23** schaltet viele angepasste Antennen zum schwenkbaren Array, **Kapitel 24** baut die Yagi zu Ende (NEC-Maße, Anpassglied, Balun), **Kapitel 25** fragt, was passiert, wenn die Antenne selbst viel kleiner als λ ist (ihr Strahlungswiderstand stürzt ab — und 50-Hz-Funk stirbt endgültig), und **Kapitel 27** stapelt λ/4-Schichten zu Spiegeln, Filtern und Resonatoren.

Übungen

Ü 22.1 (Verstehen). (a) Fernseh- und Satellitenkabel haben $Z_0 = 75\ \Omega$ bei $\varepsilon_r = 1{,}4$ (geschäumtes Dielektrikum). Welches Durchmesserverhältnis D/d braucht das? (b) Dein USB-Kabel ist 1 m lang, die Signalflanken dauern 0,1 ns. Ist das Kabel ein „Verbindungsstrich" oder eine Leitung? Rechne die Flankenlänge auf dem Kabel aus ($v \approx 2 \cdot 10^8$ m/s) und vergleiche.

Musterlösung zu Ü 22.1

```
# (a) Z0 = 60/sqrt(eps_r) · ln(D/d)  →  D/d = exp(Z0·sqrt(eps_r)/60)
verhaeltnis = np.exp(75 * np.sqrt(1.4) / 60)
print(f"(a) D/d = {verhaeltnis:.2f}")
# (b) Flankenlänge = v · Anstiegszeit
flanke = 2e8 * 0.1e-9
print(f"(b) Flankenlänge = {flanke*100:.0f} cm auf 100 cm Kabel")
```

```
(a) D/d = 4.39
(b) Flankenlänge = 2 cm auf 100 cm Kabel
```

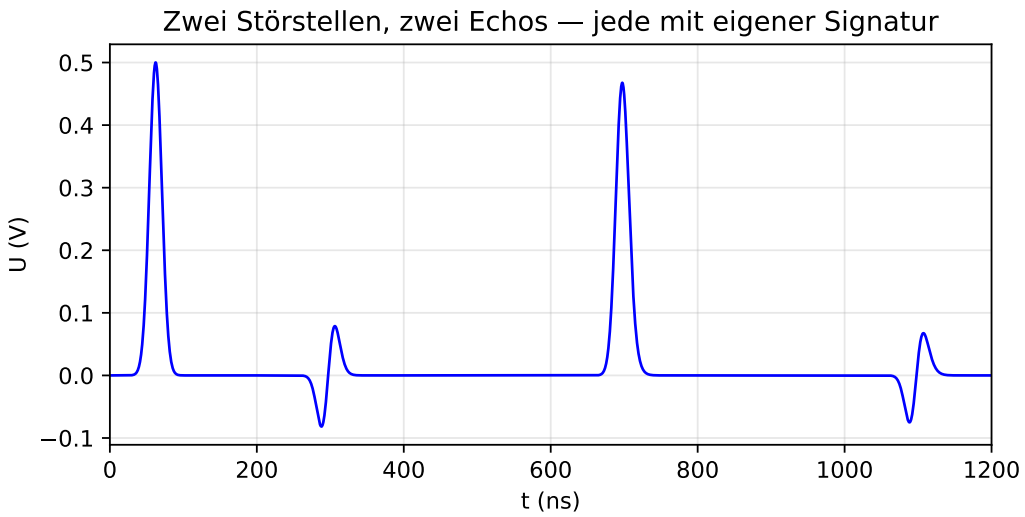
(a) $D/d \approx 4{,}4$ — deshalb sind 75-Ω-Kabel bei gleichem Innenleiter dicker als 50-Ω-Kabel. (b) Die Flanke ist nur 2 cm lang — auf einem 1-m-Kabel ist sie eine *Welle*, die fünfzigmal hineinpasst. USB ist eine Leitung durch und durch: Ohne durchgehend kontrollierte 90 Ω (differentiell) und Terminierung käme an den Echos kein Bit mehr vorbei. Der „Verbindungsstrich" endet heute nicht bei Funkfrequenzen, sondern bei jeder schnellen Digitalflanke.

Ü 22.2 (Verändern). Ein Kabel liefert *zwei* Echos: ein schwaches negatives nach 300 ns und ein starkes positives nach 700 ns. Stelle die Situation im TDR-Labor nach (Tipp: eine gequetschte Stelle hat lokal *kleineres* Z_0) und erkläre beide Echos. Warum ist das zweite Echo nicht einfach $r = +1$?

Musterlösung zu Ü 22.2

Negativ nach 300 ns: eine Störstelle mit $Z < Z_0$ bei $x = v \cdot (300\,\text{ns} - t_0)/2$... wir rechnen mit dem Puls bei $t_0 = 60$ ns: $x = 2 \cdot 10^8 \cdot 240\,\text{ns}/2 = 24$ m. Positiv nach 700 ns: offenes Ende bei 64 m.

```
# von oben: leitung(), puls, T_PULS, DT, V_TEMPO, N, Z0, X
z_doppel = np.full(N, Z0)
z_doppel[235:245] = 30.0        # Quetschung: 1 m mit 30 Ω bei x = 24 m
z_doppel[640:] = 50000.0        # Bruch (offen) bei x = 64 m
_, _, pegel_d, _, _ = leitung(13000, puls, 50.0, 50000.0,
                              z_profil=z_doppel, sonden=(5,))
t = np.arange(1, 13001) * DT
fig, ax = plt.subplots(figsize=(7.0, 3.2))
ax.plot(t * 1e9, pegel_d[5], "b-", lw=1.2)
ax.set_xlim(0, 1200); ax.set_xlabel("t (ns)"); ax.set_ylabel("U (V)")
ax.set_title("Zwei Störstellen, zwei Echos — jede mit eigener Signatur")
ax.grid(alpha=0.3)
plt.show()
```

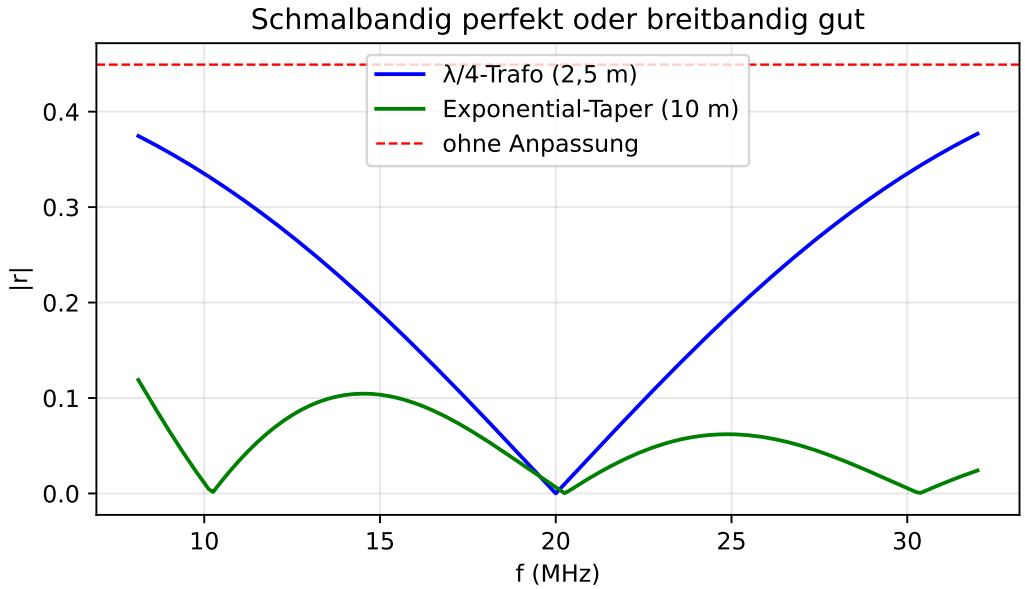


Die Quetschung wirft mit $r = (30-50)/(30+50) = -0{,}25$ ein negatives Teilecho und lässt den Rest *durch* — der dann am Bruch mit $+1$ reflektiert wird. Das zweite Echo ist trotzdem kleiner als der volle Puls, denn es musste **zweimal durch die Quetschung**: hinein mit der Spannungstransmission $1+r$, heraus mit $1-r$, zusammen $1-r^2 \approx 0{,}94$ — und an jedem Quetschungsrand zweigt zusätzlich ein Mini-Mehrfachecho ab. TDR-Profis lesen aus solchen Echo-Ketten ganze Kabeltrassen wie ein Protokoll.

Ü 22.3 (Verändern). Der λ/4-Trafo ist schmalbandig. Ersetze das 2,5-m-Stück durch einen **Taper**: $Z(x)$ läuft über 10 m *exponentiell* (also linear im Logarithmus) von 50 Ω auf 19 Ω. Miss $|r|(f)$ mit der Puls-FFT-Methode und vergleiche mit der V-Kurve des λ/4-Trafos. Welche Kapitel-6-Idee ist der Taper?

Musterlösung zu Ü 22.3

```
# von oben: leitung(), traeger_puls, r_von_f, freq, fenster,
#           DT, Z0, X, N, SONDE, r19
z_taper = np.full(N, Z0)
im_taper = (X >= 70.0) & (X < 80.0)
# np.geomspace(a, b, n): n Punkte von a nach b, logarithmisch
# gleichverteilt — genau „exponentiell von 50 auf 19"
z_taper[im_taper] = np.geomspace(50.0, 19.0, im_taper.sum())
z_taper[X >= 80.0] = 19.0
_, _, pegel_tp, _, _ = leitung(16000, traeger_puls, 50.0, 19.0,
                               z_profil=z_taper, sonden=(SONDE,))
u_tp = np.array(pegel_tp[SONDE])
t16 = np.arange(1, 16001) * DT
r_taper = (np.abs(np.fft.rfft(np.where(t16 > 500e-9, u_tp, 0.0)))
           / np.abs(np.fft.rfft(np.where(t16 <= 500e-9, u_tp, 0.0))))
fig, ax = plt.subplots(figsize=(7.0, 3.6))
ax.plot(freq[fenster] / 1e6, r_von_f[fenster], "b-", lw=1.6,
        label="λ/4-Trafo (2,5 m)")
ax.plot(freq[fenster] / 1e6, r_taper[fenster], "g-", lw=1.6,
        label="Exponential-Taper (10 m)")
ax.axhline(abs(r19), color="r", ls="--", lw=1, label="ohne Anpassung")
ax.set_xlabel("f (MHz)"); ax.set_ylabel("|r|")
ax.legend(); ax.grid(alpha=0.3)
ax.set_title("Schmalbandig perfekt oder breitbandig gut")
plt.show()
```



Der Vergleich zeigt zwei Philosophien: Der λ/4-Trafo glänzt mit seiner exakten Null, aber nur bei f_0. Der Taper hält $|r|$ über das **ganze Band** klein (mit eigenen Interferenz-Minima, wo seine 10 m gerade passend ausgehen) — und seine Einhüllende sinkt nach oben hin weiter, denn je kürzer λ, desto „sanfter" wirken die 10 m. Das ist wortwörtlich die **σ-Rampe aus Kapitel 6** und die Grundidee der PML: Übergänge glätten statt verstecken. Faustregel der HF-Technik: Eine Frequenz → λ/4-Trafo; ein Frequenz*band* → Taper (oder mehrere λ/4-Stufen — Kapitel 27).

Ü 22.4 (Übertragen). Die Yagi ans Kabel, komplett: Hänge die 19-Ω-Last über einen korrekt dimensionierten λ/4-Transformator ans 50-Ω-Kabel und miss die **Leistung in der Last** im Dauerstrich bei f_0 — einmal direkt angeschlossen, einmal angepasst. (Leistung an der Last: $P = U_L \cdot I_L$, im Zeitmittel über eine Periode.) Bestätige 80 % gegen ≈ 100 %.

Musterlösung zu Ü 22.4

Wir messen P direkt am Lastwiderstand als $\langle U^2 \rangle / R_L$ über die letzte Periode und beziehen sie auf die verfügbare Leistung der Quelle ($P_{\max} = U_q^2/(8R_q)$ bei $R_q = Z_0$ — das Maximum, das eine 50-Ω-Quelle überhaupt abgeben kann):

```python
# von oben: leitung(), sinus, z_trafo, DT, Z0, N, F0
def lastleistung(z_profil):
    _, _, pegel, _, _ = leitung(24000, sinus, 50.0, 19.0,
                                z_profil=z_profil, sonden=(N - 1,))
    u = np.array(pegel[N - 1])[-1000:]        # letzte 20 Perioden
    return np.mean(u**2) / 19.0

p_max = 1.0**2 / (8 * 50.0)                   # verfügbare Leistung
p_direkt = lastleistung(None)
p_trafo = lastleistung(z_trafo)
print(f"direkt:    P/P_max = {p_direkt / p_max:.3f}   (Theorie 0,798)")
print(f"mit Trafo: P/P_max = {p_trafo / p_max:.3f}   (Theorie 1,000)")
```

```
direkt:    P/P_max = 0.799   (Theorie 0,798)
mit Trafo: P/P_max = 1.001   (Theorie 1,000)
```

Der Transformator holt die fehlenden 20 % — aus 1 dB Fehlanpassungsverlust wird Anpassung auf Messgenauigkeit. Genau dieses Spiel treibt der Bauplan in Kapitel 24 in echt: Hairpin und Gamma-Glied sind nichts als kompakte Bauformen derselben Impedanz-Übersetzung.

Das Kleingedruckte

Warum 50 und warum 75? Für Koaxkabel mit Luftfüllung hat die Spannungsfestigkeit ihr Optimum bei etwa 30 Ω, die geringste Dämpfung liegt bei etwa 77 Ω — 50 ist der historische Kompromiss aus der Radartechnik der 1930er. Das Fernsehen, dem Dämpfung wichtiger war als Leistung, blieb bei 75 Ω. Beide Zahlen sind also Ingenieurskompromisse, keine Naturkonstanten — im Gegensatz zu den 377 Ω des Vakuums, die seit Kapitel 6 feststehen.

Balun und Mantelwellen. Unser 1D-Modell kennt nur die TEM-Welle *zwischen* den Leitern. Ein reales Koaxkabel hat noch einen dritten Leiter: die *Außenseite* des Schirms. Wird ein unsymmetrisches Kabel direkt an eine symmetrische Antenne (Dipol!) geschraubt, fließt ein Teil des Antennenstroms außen auf dem Mantel zurück — der Mantel strahlt mit und verbiegt das Richtdiagramm. Dagegen hilft ein **Balun** (*bal*anced–*un*balanced): bei 2,4 GHz z. B. die λ/4-Mantelhülse aus dem Yagi-Bauplan (§5) — die übrigens nichts anderes ist als ein λ/4-Stück Leitung, das für Mantelströme ein offenes Ende in einen Kurzschluss hochtransformiert. Auch das ist die Mathematik dieses Kapitels, nur auf den dritten Leiter angewandt.

Verluste. Unsere Leitung war verlustfrei; reale Kabel dämpfen (Skin-Effekt im Kupfer, Polarisationsverluste im Dielektrikum — beides wächst mit der Frequenz: RG-58 schluckt bei 2,4 GHz rund 1 dB *pro Meter*!). In den Telegrafengleichungen ergänzen dafür zwei weitere Beläge (R' längs, G' quer) genau die Rolle, die σ in Kapitel 6 spielte. Und wer Lasten anpassen muss, die nicht rein ohmsch sind (die Yagi hatte $19 + \mathrm{j}1\ \Omega$!), benutzt dafür seit 1939 eine geniale grafische Karte — das **Smith-Diagramm**. Es hier einzuführen würde ein eigenes Kapitel kosten; für unsere Zwecke reicht: Auch komplexe Lasten sind mit Leitungsstücken und der Mathematik dieses Kapitels anpassbar.

23 Arrays und Beamforming

Auf jedem Flughafen dreht sich eine Radarantenne — eine ganze Mechanik aus Motor, Lagern und Getriebe, nur damit der Strahl einmal pro Umlauf in jede Richtung zeigt. Ein Starlink-Terminal auf dem Wohnmobildach hat dasselbe Problem in schwieriger: Es muss einen Satelliten verfolgen, der in 550 km Höhe mit 27 000 km/h über den Himmel zieht. Trotzdem liegt die Schüssel flach, starr und lautlos da — kein Motor, kein Gelenk, nichts bewegt sich. Und sie hält den Satelliten trotzdem im Strahl, wechselt sogar alle paar Minuten in Sekundenbruchteilen zum nächsten.

Wie lenkt man einen Funkstrahl, ohne irgendetwas zu bewegen? Die Antwort steckt in einer Übung, die du vielleicht schon gerechnet hast: In Übung 14.3 hat das *Umpolen einer einzigen Quelle* das gesamte Interferenzmuster invertiert — hell wurde dunkel, dunkel wurde hell. Die Phase einer Quelle ist ein ebenso scharfes Steuer wie ihr Ort. Dieses Kapitel macht aus dem Trick ein Werkzeug: viele gleiche Strahler in einer Reihe, jeder mit einem eigenen **Phasenknopf** — ein *Phased Array*. Der Strahl folgt dann nicht der Geometrie, sondern dem **Zündplan**.

Lernziele

Nach diesem Kapitel kannst du …

1. … mit „Umweg = Vorsprung” erklären und ausrechnen, wohin die Keule einer Strahlerreihe kippt: $\sin\theta_0 = \Delta\Phi/(k\,d)$,
2. … den **Array-Faktor** als Zeigersumme hinschreiben, in NumPy auswerten und zeigen, dass $N = 2$ exakt der Doppelspalt aus Kapitel 14 ist,
3. … **Grating Lobes** vorhersagen (Geisterkeulen bei $\sin\theta = \sin\theta_0 \pm m\lambda/d$) und die Designregel für den Elementabstand herleiten,
4. … mit dem **Faltungssatz** Gesamtdiagramme zerlegen (Element × Array-Faktor) und mit **Amplituden-Tapern** Nebenkeulen gegen Keulenbreite eintauschen — und du weißt, warum das exakt die Fensterfunktionen der Spektralanalyse sind,
5. … **Scan Loss** (die cos-Regel der projizierten Apertur) und **Empfangs-Beamforming** (Reziprozität, Null-Steuern) erklären.

```
import numpy as np
import matplotlib.pyplot as plt
```

23.1 Umweg = Vorsprung: die Geometrie des Schwenks

Stell dir zwei Strahler im Abstand d übereinander vor, beide senden dieselbe Frequenz. Ein weit entfernter Beobachter steht in Richtung θ schräg oberhalb der Mittelsenkrechten (der „Breitseite" der Reihe). Für ihn ist der obere Strahler ein Stück näher — die Welle von unten muss einen **Umweg** laufen, und ein Blick auf das rechtwinklige Dreieck aus Abstand d und Blickrichtung zeigt: Der Umweg ist genau die Projektion

$$\Delta s = d\,\sin\theta.$$

Senden beide Strahler **gleichphasig**, addieren sich ihre Wellen dort konstruktiv, wo der Umweg verschwindet — die Keule zeigt zur Breitseite, $\theta = 0$. Das ist der Doppelspalt aus Kapitel 14, rückwärts gelesen: nicht „wo wird es hell¿', sondern „wohin schicken wir das Hell?".

Jetzt kommt der Phasenknopf. Lassen wir den *oberen* Strahler um die Phase $\Delta\Phi$ **später** zünden, dann gleicht sein Zeitrückstand den Wegvorsprung genau dort aus, wo

$$k\,d\,\sin\theta_0 = \Delta\Phi \quad\Longrightarrow\quad \sin\theta_0 = \frac{\Delta\Phi}{k\,d}, \qquad k = \frac{2\pi}{\lambda}.$$

In dieser Richtung kommen beide Wellenberge wieder gleichzeitig an — die Keule ist zur Seite der *später* zündenden Quelle gekippt. **Umweg = Vorsprung**: Der geometrische Umweg wird durch den elektrischen Vorsprung bezahlt, und der Zündplan bestimmt die Richtung.

Eine Handrechnung mit Zahlen, wie immer. Acht Strahler im Abstand $d = \lambda/2$, jeder zündet $90°$ nach seinem unteren Nachbarn:

$$\sin\theta_0 = \frac{\Delta\Phi}{k\,d} = \frac{\pi/2}{(2\pi/\lambda)\cdot(\lambda/2)} = \frac{\pi/2}{\pi} = 0{,}5 \quad\Longrightarrow\quad \theta_0 = 30°.$$

Heraus kommt ein Winkel, ein Skalar. Dreht man den Phasenknopf auf $135°$, wandert die Keule auf $\arcsin(0{,}75) = 48{,}6°$ — elektronisch, in Nanosekunden, ohne dass sich ein Gramm Metall bewegt.

Das anschaulichste Bild für den Mechanismus ist die **Stadion-La-Ola**: Jeder Strahler „steht auf", kurz nachdem sein Nachbar aufgestanden ist. Die Wellenberge aller Strahler liegen dann auf einer **gekippten Linie** — und diese gekippte Linie *ist* die neue Wellenfront. Sie läuft senkrecht zu sich selbst davon, also in die Richtung θ_0.

Naheliegende Vermutung: „Antennen strahlen, wohin sie zeigen"

Naheliegende Vermutung: Eine Antenne strahlt dorthin, wohin sie zeigt. Wer die Richtung ändern will, muss die Antenne drehen — wie eine Taschenlampe.
Warum sie naheliegt: Bei allem, was wir aus dem Alltag kennen, bestimmt die Geometrie die Richtung: Taschenlampe, Parabolspiegel, Satellitenschüssel. Auch das Flughafen-Radar dreht sich ja tatsächlich.
Was stattdessen stimmt: Die Richtung der Abstrahlung steckt in den **relativen Phasen** der strahlenden Ströme. Eine starre Reihe gleicher Strahler plus ein einstellbarer Zündplan genügt: $\sin\theta_0 = \Delta\Phi/(k\,d)$. Die Geometrie legt nur fest, *welche* Richtungen erreichbar sind und wie scharf die Keule wird — *wohin* sie zeigt, entscheidet die Elektronik. Genau deshalb kann ein flaches, unbewegliches Starlink-Panel einen rasenden Satelliten verfolgen.

23.2 Der Array-Faktor: die Zeigersumme bekommt einen Phasenknopf

Wie hell ist es in einer beliebigen Richtung θ? In Kapitel 14 haben wir dafür Zeiger addiert: Jede Quelle trägt im Fernfeld einen Zeiger bei, dessen Drehwinkel ihr Gangunterschied bestimmt. Neu ist nur, dass jetzt auch der Zündplan am Drehwinkel mitdreht. Quelle n (von unten gezählt) hat gegenüber Quelle 0 den Wegvorsprung $n\,d\sin\theta$ (Phase $+n\,kd\sin\theta$) und die Zündverspätung $n\,\Delta\Phi$ (Phase $-n\,\Delta\Phi$). Mit der Amplituden-„Belegung" a_n (vorerst alle 1) lautet die Zeigersumme

$$F(\theta) \;=\; \sum_{n=0}^{N-1} a_n\, e^{\mathrm{i}\,n\,(kd\sin\theta\,-\,\Delta\Phi)}, \qquad \text{Leistung} \;\propto\; |F(\theta)|^2.$$

$|F|^2$ heißt **Array-Faktor** (genauer: Leistungs-Array-Faktor) — „Faktor", weil er später mit dem Diagramm des Einzelelements multipliziert wird; für unsere in der Ebene rundum gleich strahlenden Punktquellen ist er schon das

ganze Diagramm. Das ist ein Skalarfeld über dem Winkel: Für jede Richtung θ kommt eine Zahl heraus.

In NumPy ist die Summe vier Zeilen — und weil sie das Arbeitstier des ganzen Kapitels wird, falten wir sie einmal auseinander:

```
# von oben: nichts — k = 2π/λ mit λ = 1 (Meep-Konvention, Kap. 11)
k = 2 * np.pi


def array_faktor(theta, belegung, d, dphi_grad=0.0, normiert=True):
    """Leistungs-Array-Faktor |Σ a_n·exp(i·n·(kd·sinθ − ΔΦ))|².

    Konvention: Element n zündet um n·ΔΦ später als Element 0
    (die La-Ola läuft nach oben) → die Keule kippt nach +θ.
    """
    n = np.arange(len(belegung))                    # 0, 1, …, N−1
    # Drehwinkel pro Element-Schritt, für JEDE Richtung theta:
    psi = k * d * np.sin(theta) - np.deg2rad(dphi_grad)
    # Zeiger e^{i·n·ψ}: Zeilen = Richtungen, Spalten = Elemente
    # (Broadcasting [:, None]/[None, :] wie in Kap. 9)
    zeiger = np.asarray(belegung)[None, :] * np.exp(
        1j * n[None, :] * psi[:, None])
    F = zeiger.sum(axis=1)                          # die Zeigersumme
    P = np.abs(F) ** 2                              # Leistung
    return P / P.max() if normiert else P
```

Die Längen-Buchhaltung: `theta` hat M Richtungen, `belegung` N Elemente. `psi` hat M Einträge (ein Drehwinkel pro Richtung), die Zeiger-Matrix $M \times N$ (jede Richtung bekommt alle N Zeiger), und `zeiger.sum(axis=1)` addiert pro Zeile die N Zeiger — übrig bleiben M komplexe Summen, eine pro Richtung.

Bevor wir damit schwenken, die Pflichtprüfung am Bekannten: Für $N = 2$ und $\Delta\Phi = 0$ muss exakt das Doppelspalt-cos^2 aus Kapitel 14 herauskommen, $\cos^2(\pi d \sin\theta/\lambda)$ — zwei Zeiger der Länge 1 addieren sich je nach Zwischenwinkel:

```
# von oben: array_faktor(), k
theta = np.linspace(-np.pi / 2, np.pi / 2, 100001)

P_zwei = array_faktor(theta, [1, 1], d=0.5)
doppelspalt = np.cos(np.pi * 0.5 * np.sin(theta)) ** 2
```

```
print(f"N = 2 gegen das Kap.-14-cos²: max. Abweichung "
      f"{np.abs(P_zwei - doppelspalt).max():.1e}")
```

```
N = 2 gegen das Kap.-14-cos²: max. Abweichung 4.4e-16
```

Maschinengenauigkeit — der Array-Faktor *ist* die Doppelspaltformel, verallgemeinert auf N Quellen und einen Phasenknopf.

23.2.1 Mehr Elemente, schärfere Keule

Vorhersage-Punkt (PRIMM): Wir verdoppeln gleich die Elementzahl von 4 auf 8 auf 16 (Abstand fest $d = \lambda/2$). Was passiert mit der Breite der Hauptkeule — halbiert sie sich, viertelt sie sich, oder bleibt sie gleich? Denk an das Beugungsgitter aus Kapitel 14: Dort machten N Spalte die Maxima N-fach schmal.

Die Theorie gibt die Antwort exakt für die **erste Nullstelle** neben der Hauptkeule: Die Zeigersumme wird null, sobald die N Zeiger den Kreis einmal schließen (das Zeigerbild aus Kapitel 14), also bei $N\,kd\sin\theta = 2\pi$ oder

$$\sin\theta_{\text{Null}} = \frac{\lambda}{N\,d}.$$

Die Keulenbreite fällt wie $1/N$ — im Nenner steht $N\,d$, die **Gesamtlänge des Arrays in Wellenlängen**. Es ist dieselbe Regel wie überall in diesem Buch, von der Spaltbreite (Kapitel 14) bis zum dicken Dipol (Kapitel 21): Große Strukturen bündeln scharf.

```
# von oben: array_faktor(), theta
fig, ax = plt.subplots(figsize=(7.0, 4.2))
for N, farbe in ((4, "tab:blue"), (8, "tab:green"), (16, "tab:red")):
    P = array_faktor(theta, np.ones(N), d=0.5)
    ax.plot(np.degrees(theta), 10 * np.log10(np.maximum(P, 1e-5)),
            color=farbe, lw=1.4, label=f"N = {N}")
    null = np.degrees(np.arcsin(1 / (N * 0.5)))
    ax.axvline(+null, color=farbe, ls="--", lw=0.9, alpha=0.7)
    ax.axvline(-null, color=farbe, ls="--", lw=0.9, alpha=0.7)
ax.set_xlim(-90, 90)
ax.set_ylim(-40, 2)
ax.set_xlabel("Winkel θ zur Breitseite (°)")
ax.set_ylabel("Array-Faktor (dB)")
```

```
ax.set_title("Verdopplung der Elemente halbiert die Keule "
             "(gestrichelt: Theorie-Nullstellen)")
ax.legend()
plt.show()

P_8 = array_faktor(theta, np.ones(8), d=0.5)
i_min = np.flatnonzero((P_8[1:-1] < P_8[:-2]) & (P_8[1:-1] < P_8[2:])) + 1
nullen = np.degrees(theta[i_min])
print(f"N = 8: erste Nullstelle gemessen {nullen[nullen > 0][0]:.2f}°, "
      f"Theorie {np.degrees(np.arcsin(1 / 4)):.2f}°")
```

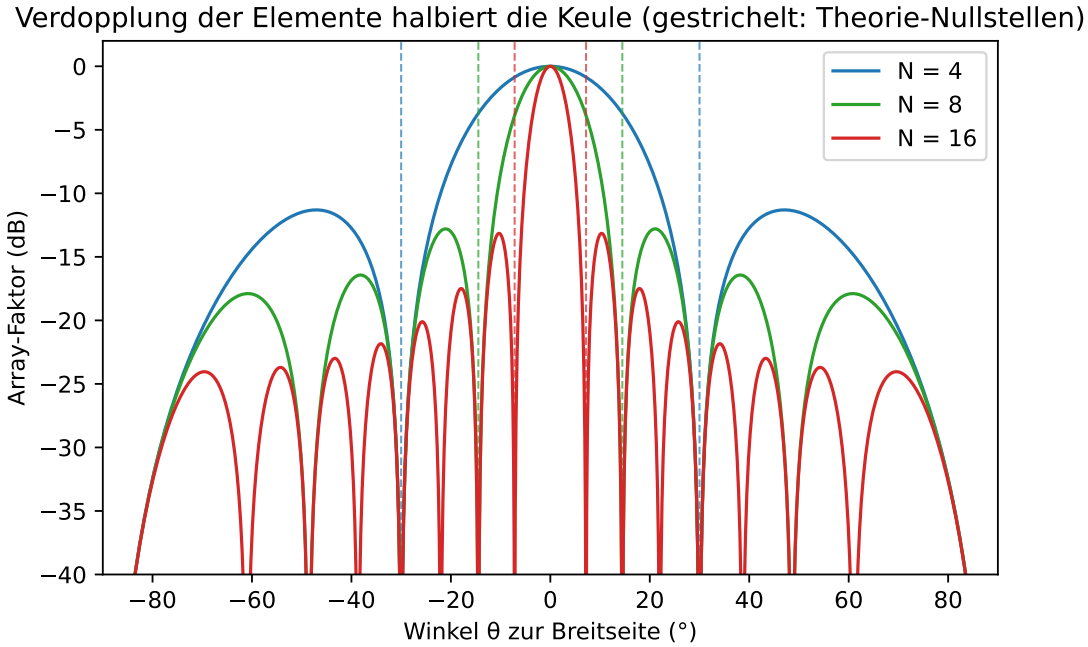


Abbildung 23.1: Der Array-Faktor für 4, 8 und 16 Elemente (d = λ/2, ΔΦ = 0) in Dezibel. Die gestrichelten Senkrechten markieren die theoretische erste Nullstelle sin θ = λ/(N·d) — mit jeder Verdopplung der Elementzahl halbiert sich die Keulenbreite. Die Nebenkeulen bleiben dabei hartnäckig auf etwa −13 dB stehen; um sie kümmert sich der Taper-Abschnitt.

```
N = 8: erste Nullstelle gemessen 14.48°, Theorie 14.48°
```

Die Antwort auf den Vorhersage-Punkt: Sie **halbiert** sich — die Nullstelle rückt von $\arcsin(1/2) = 30°$ ($N = 4$) auf $14{,}48°$ ($N = 8$) auf $7{,}18°$ ($N = 16$), und die Messung trifft die Theorie auf zwei Nachkommastellen.

23.2.2 Der Schwenk, vermessen

Jetzt drehen wir den Phasenknopf. Erfolgskriterium: Das Maximum des Array-Faktors (per `np.argmax`) liegt für jede Einstellung auf $\arcsin(\Delta\Phi/(kd))$:

```
# von oben: array_faktor(), theta
print("ΔΦ      Keule (argmax)   Theorie arcsin(ΔΦ/kd)")
for dphi in (0, 45, 90, 135):
    P = array_faktor(theta, np.ones(8), d=0.5, dphi_grad=dphi)
    mess = np.degrees(theta[np.argmax(P)])
    theo = np.degrees(np.arcsin(np.deg2rad(dphi) / (k * 0.5)))
    print(f"{dphi:3d}°    {mess:+7.2f}°        {theo:+7.2f}°")
```

```
ΔΦ      Keule (argmax)   Theorie arcsin(ΔΦ/kd)
  0°      -0.00°          +0.00°
 45°     +14.48°         +14.48°
 90°     +30.00°         +30.00°
135°     +48.59°         +48.59°
```

Vier Einstellungen, vier Treffer auf zwei Nachkommastellen — die Handrechnung von oben ($90° \to 30°$, $135° \to 48{,}59°$) steht in der Tabelle.

23.3 Maxwell bestätigt: das Feldbild in Meep

Der Array-Faktor ist Fernfeld-Geometrie — Zeiger, Umwege, Phasen. Aber rechnet das volle Maxwell-System wirklich so? Wir prüfen es in der 2D-Welt aus Kapitel 9, mit Meep als Arbeitspferd (Kapitel 11).

Der Versuchsaufbau. **Frage:** Kippen die Wellenfronten einer phasengestaffelt gespeisten Quellreihe wirklich um $\theta_0 = \arcsin(\Delta\Phi/(kd))$, und folgt die Strahlungskeule? **Bühne:** eine $16\lambda \times 16\lambda$ große 2D-Zelle, ringsum 1 λ PML; acht E_z-Punktquellen im Abstand $d = \lambda/2$ entlang y, als Reihe bei $x = -5\lambda$. **Anfangs-/Randbedingungen:** Feld startet bei null, die Quellen senden Dauerstrich ($f = 1$), der Lauf endet bei $t = 40$ — die Zelle ist dann längst eingeschwungen (Laufzeit quer: 16 Zeiteinheiten). **Messgrößen:** das komplexe E_z-Feld der ganzen Zelle; daraus das Momentanbild $\mathrm{Re}(E_z)$ und die zeitgemittelte Intensität $|E_z|^2$. **Erfolgskriterium:** Front- und Keulenrichtung liegen auf der eingezeichneten Theorielinie, und als absoluter Gesundheitscheck (die Lektion aus Kapitel 21!) muss max $|E_z|$ in der Größenordnung 1 liegen — nicht 10^{-19}.

Zwei Meep-Bausteine sind neu. Erstens `force_complex_fields=True`: Meep rechnet normalerweise mit reellen Feldern; mit komplexen Feldern liefert *ein* eingeschwungener Dauerstrich-Lauf gleich beides — Momentanbild (Realteil) und Zeitmittel ($|E_z|^2$), ganz ohne Periodenmittelung wie in Kapitel 18. Zweitens bekommt `mp.Source` eine **komplexe Amplitude**: Der Faktor $e^{\mathrm{i}n\Delta\Phi}$ ist mit Meeps $e^{-\mathrm{i}\omega t}$-Zeitkonvention genau die Zündverspätung um $n\,\Delta\Phi$ — das Zeitsignal der Quelle n ist $\cos(\omega t - n\Delta\Phi)$, unser Zündplan.

```
# von oben: k; ab hier Meep (Kap. 11): λ = 1, f = 1
import meep as mp
mp.verbosity(0)

N_EL, D_EL = 8, 0.5
SX = SY = 16
X_ARRAY = -5.0


def cw_feld(dphi_grad):
    """Eingeschwungener Dauerstrich-Lauf; liefert das komplexe Ez."""
    y0 = -(N_EL - 1) * D_EL / 2
    quellen = [
        mp.Source(mp.ContinuousSource(frequency=1.0), component=mp.Ez,
                  center=mp.Vector3(X_ARRAY, y0 + n * D_EL),
                  # komplexe Amplitude = Phasenknopf: Element n
                  # zündet um n·ΔΦ später (Meep-Konvention e^{-iωt})
                  amplitude=np.exp(1j * np.deg2rad(dphi_grad) * n))
        for n in range(N_EL)]
    sim = mp.Simulation(cell_size=mp.Vector3(SX, SY),
                        boundary_layers=[mp.PML(1.0)],
                        sources=quellen, resolution=16,
                        # komplexe Felder: Momentanbild UND |Ez|²
                        # aus einem einzigen Lauf
                        force_complex_fields=True)
    sim.run(until=40)
    return sim.get_array(center=mp.Vector3(), size=mp.Vector3(SX, SY),
                         component=mp.Ez)


PHASEN = list(range(0, 136, 15))          # 0°, 15°, …, 135°
felder = {p: cw_feld(p) for p in PHASEN}
```

```
# von oben: felder, PHASEN
print(f"Gesundheitscheck: max|Ez| (ΔΦ = 90°) = "
      f"{np.abs(felder[90]).max():.3f}")
```

```
Gesundheitscheck: max|Ez| (ΔΦ = 90°) = 2.128
```

Die Amplitude liegt bei 2 — die Quellen speisen wirklich ins Feld (nach dem Speisespalt-Debakel aus Kapitel 21 prüfen wir das grundsätzlich, *bevor* wir hübsche normierte Bilder ansehen).

```
# von oben: felder, N_EL, D_EL, SX, SY, X_ARRAY, k
ext = (-SX / 2, SX / 2, -SY / 2, SY / 2)
y_q = (np.arange(N_EL) - (N_EL - 1) / 2) * D_EL


def keulenlinie(ax, dphi_grad, farbe="k"):
    th0 = np.arcsin(np.deg2rad(dphi_grad) / (k * D_EL))
    ax.plot([X_ARRAY, X_ARRAY + 11.5 * np.cos(th0)],
            [0, 11.5 * np.sin(th0)], farbe + "--", lw=1.2, alpha=0.85)


fig, ax = plt.subplots(2, 3, figsize=(7.6, 5.3))
for j, p in enumerate((0, 90, 135)):
    ez = felder[p]
    re = np.real(ez).T                    # Meep-Array: (x, y) → Bild (y, x)
    inten = (np.abs(ez) ** 2).T
    th0 = np.degrees(np.arcsin(np.deg2rad(p) / (k * D_EL)))

    a = ax[0, j]
    a.imshow(re, origin="lower", extent=ext, cmap="RdBu_r",
             vmin=-np.abs(re).max() * 0.35, vmax=np.abs(re).max() * 0.35)
    a.plot([X_ARRAY] * N_EL, y_q, "k.", ms=3)
    keulenlinie(a, p)
    a.set_title(f"Re(Ez) — ΔΦ = {p}°", fontsize=9)
    a.set_xticks([]); a.set_yticks([])

    a = ax[1, j]
    a.imshow(inten / inten.max(), origin="lower", extent=ext,
             cmap="inferno", vmin=0, vmax=0.25)
    a.plot([X_ARRAY] * N_EL, y_q, "w.", ms=3)
    keulenlinie(a, p, "w")
    a.set_title(f"|Ez|² — Keule bei {th0:.0f}°", fontsize=9)
    a.set_xticks([]); a.set_yticks([])
fig.suptitle("Acht Quellen, drei Zündpläne: die Keule folgt dem "
             "Phasenknopf", fontsize=11)
```

```
fig.tight_layout()
plt.show()
```

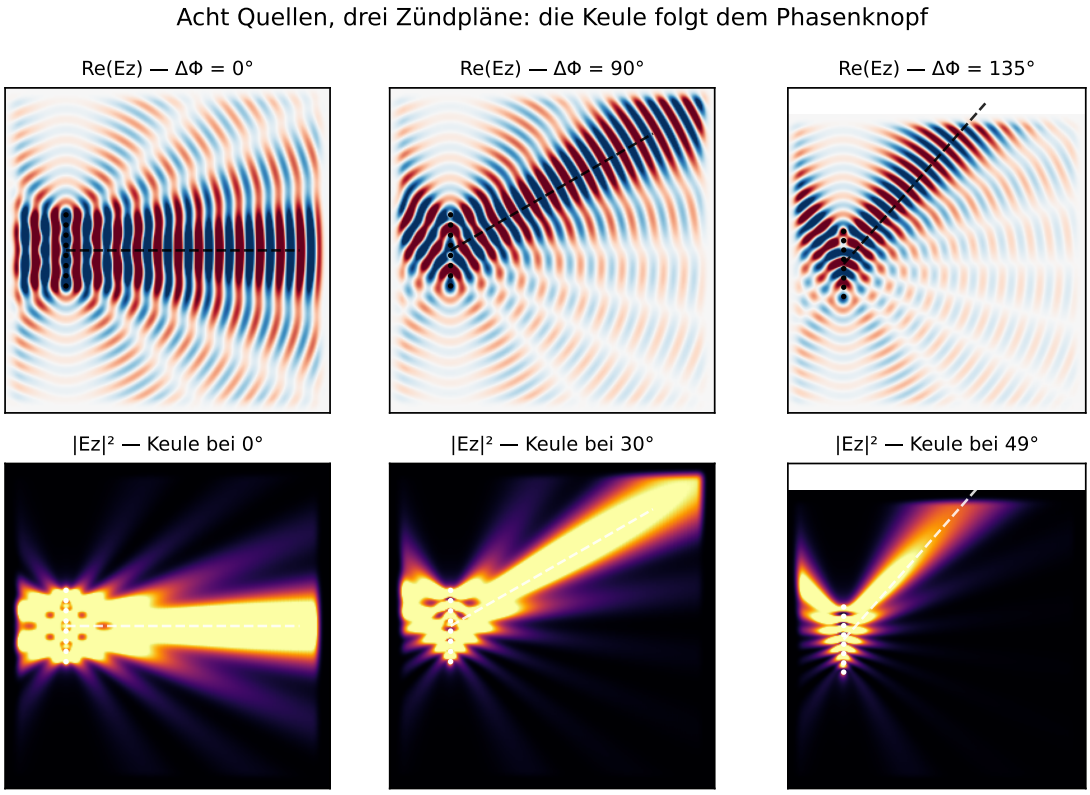


Abbildung 23.2: Dieselbe Strahlerreihe, drei Zündpläne — oben das Momentanfeld Re(Ez) (rot/blau = Berg/Tal), unten die zeitgemittelte Intensität $|Ez|^2$ desselben Laufs. Bei $\Delta\Phi = 0$ laufen die Fronten parallel zur Reihe und die Keule zeigt zur Breitseite; bei 90° und 135° kippen die Fronten, und die Keule folgt exakt der gestrichelten Theorielinie $\arcsin(\Delta\Phi/kd)$ (30° bzw. 48,6°). Nichts an der Geometrie hat sich geändert — nur der Zündplan.

Oben sieht man den La-Ola-Mechanismus leibhaftig: Die Wellenberge liegen auf gekippten Linien, und je weiter der Phasenknopf gedreht wird, desto steiler die Front. Unten bestätigt die Intensität, dass die Energie der gekippten Front folgt — die Keule liegt auf der Theorielinie. Auffällig auch die *Verbreiterung* der Keule beim Schwenken (vergleiche 135° mit 0°): Das ist der **Scan Loss**, den wir weiter unten vermessen.

23.3.1 Die Nagelprobe: Fernfeld gegen Array-Faktor

Die Feldbilder überzeugen das Auge — jetzt die Zahlen. Der Versuchsaufbau: **Frage:** Ist das Maxwell-Richtdiagramm der Quellreihe wirklich der Array-Faktor, Punkt für Punkt? **Bühne:** eine kompakte $4\lambda \times 8\lambda$-Zelle (1 λ PML), dieselben acht Quellen mit $\Delta\Phi = 90°$ — diesmal als Gauß-**Puls** um $f = 1$, denn das near2far-Werkzeug aus Kapitel 14 sammelt sein Fernfeld am liebsten aus einem abklingenden Lauf. **Randbedingungen:** Die near2far-Box (vier Kanten, Außennormalen-Gewichte wie bei der 3D-Box in Kapitel 21) umschließt die Reihe vollständig und bleibt aus der PML. **Messgrößen:** $|\vec{S}|(\theta)$ auf 179 Richtungen in 1000 λ Abstand, normiert. **Erfolgskriterium:** rms-Abweichung vom Array-Faktor unter 0,01 — und die Keule bei $+30°$.

```
# von oben: N_EL, D_EL (8, 0.5), array_faktor(), k
y0 = -(N_EL - 1) * D_EL / 2
quellen_puls = [
    mp.Source(mp.GaussianSource(frequency=1.0, fwidth=0.3),
              component=mp.Ez, center=mp.Vector3(0, y0 + n * D_EL),
              amplitude=np.exp(1j * np.deg2rad(90) * n))
    for n in range(N_EL)]
sim_ff = mp.Simulation(cell_size=mp.Vector3(4, 8),
                       boundary_layers=[mp.PML(1.0)],
                       sources=quellen_puls, resolution=16,
                       force_complex_fields=True)
bx, by = 0.6, 2.4
n2f = sim_ff.add_near2far(
    1.0, 0, 1,
    mp.Near2FarRegion(mp.Vector3(+bx, 0), size=mp.Vector3(0, 2 * by)),
    mp.Near2FarRegion(mp.Vector3(-bx, 0), size=mp.Vector3(0, 2 * by),
                      weight=-1),
    mp.Near2FarRegion(mp.Vector3(0, +by), size=mp.Vector3(2 * bx, 0)),
    mp.Near2FarRegion(mp.Vector3(0, -by), size=mp.Vector3(2 * bx, 0),
                      weight=-1))
sim_ff.run(until_after_sources=mp.stop_when_fields_decayed(
    5, mp.Ez, mp.Vector3(0.5, 0), 1e-5))

th_ff = np.radians(np.linspace(-89, 89, 179))
S_ff = np.empty(len(th_ff))
for i, t in enumerate(th_ff):
    Ex, Ey, Ez, Hx, Hy, Hz = sim_ff.get_farfield(
        n2f, 1000.0 * mp.Vector3(np.cos(t), np.sin(t)))
    # zeitgemittelter Poynting-Vektor S = ½ Re(E × H*) (Kap. 18/21)
    Sx = 0.5 * np.real(Ey * np.conj(Hz) - Ez * np.conj(Hy))
```

```
    Sy = 0.5 * np.real(Ez * np.conj(Hx) - Ex * np.conj(Hz))
    S_ff[i] = np.hypot(Sx, Sy)
S_ff = S_ff / S_ff.max()
```

```
# von oben: th_ff, S_ff, array_faktor()
AF = array_faktor(th_ff, np.ones(N_EL), D_EL, dphi_grad=90)

fig, ax = plt.subplots(figsize=(7.0, 4.2))
ax.plot(np.degrees(th_ff), 10 * np.log10(np.maximum(AF, 1e-5)),
        "r-", lw=1.6, label="Array-Faktor (Zeigersumme)")
ax.plot(np.degrees(th_ff), 10 * np.log10(np.maximum(S_ff, 1e-5)),
        "bo", ms=3.5, label="Meep: near2far-Fernfeld")
ax.axvline(30, color="k", ls=":", lw=1)
ax.set_xlim(-90, 90)
ax.set_ylim(-40, 2)
ax.set_xlabel("Winkel θ zur Breitseite (°)")
ax.set_ylabel("Leistung, normiert (dB)")
ax.set_title("Maxwell gegen Zeigersumme: dasselbe Diagramm")
ax.legend(loc="lower left")
plt.show()

rms = np.sqrt(np.mean((S_ff - AF) ** 2))
print(f"rms-Abweichung Meep ↔ Array-Faktor: {rms:.4f}")
print(f"Meep-Keule bei {np.degrees(th_ff[np.argmax(S_ff)]):+.1f}°  "
      "(Theorie +30°)")
```

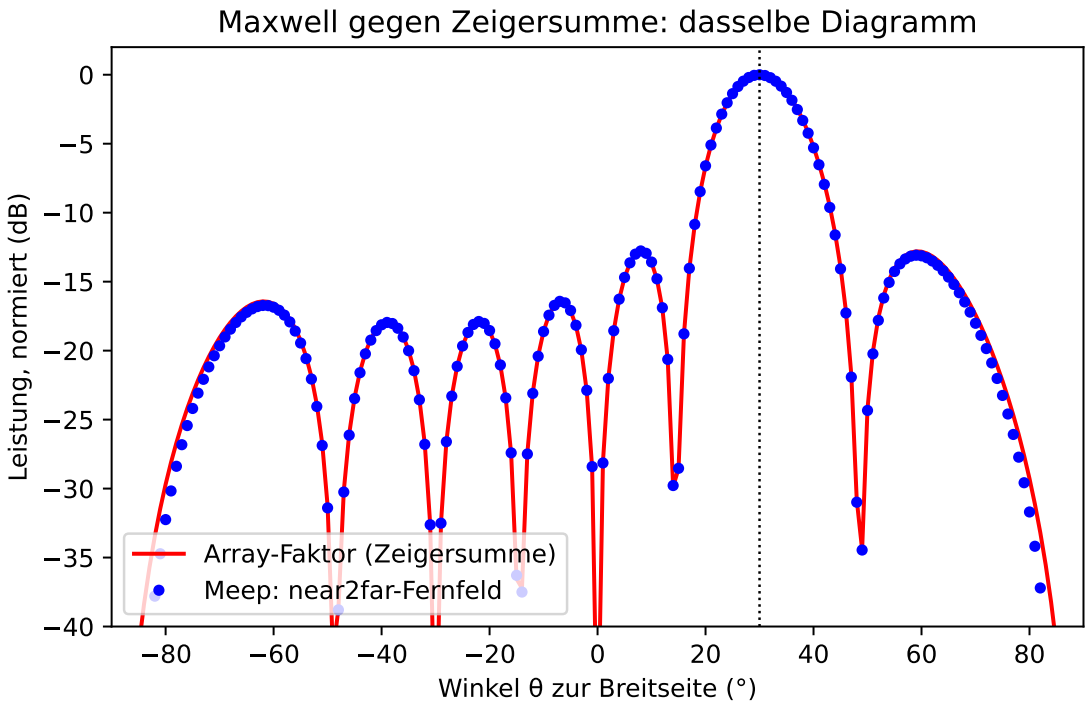


Abbildung 23.3: Das Maxwell-Fernfeld (blaue Punkte: Meep near2far, ein Pulslauf) gegen die Zeigersumme (rote Linie: Array-Faktor) für acht Quellen mit $\Delta\Phi = 90°$, in Dezibel. Beide Kurven liegen aufeinander — Hauptkeule bei +30°, alle Nebenkeulen an Ort und Pegel getroffen. Das volle Maxwell-System und die Fernfeld-Geometrie aus vier Zeilen NumPy sind dasselbe Rechenwerk.

```
rms-Abweichung Meep ↔ Array-Faktor: 0.0018
Meep-Keule bei +30.0°  (Theorie +30°)
```

Eine rms-Abweichung von 0,002 bei normierten Kurven: Das Maxwell-Fernfeld und die Zeigersumme sind deckungsgleich. Für Punktquellen (in der 2D-Ebene rundum gleich strahlend) ist der Array-Faktor also nicht Näherung, sondern das exakte Richtdiagramm — und wir dürfen die schnelle NumPy-Formel mit gutem Gewissen für alles Weitere benutzen.

23.4 Grating Lobes: die Geisterkeulen

Vorhersage-Punkt (PRIMM): Bisher stand der Elementabstand stur auf $d = \lambda/2$. Was passiert, wenn wir die acht Quellen auseinanderziehen, auf

$d = 1{,}5\,\lambda$ — wird die Keule einfach schmaler (das Array ist ja jetzt dreimal so lang), oder geht etwas schief? Übung 14.4 hat die Antwort schon einmal von der anderen Seite gestreift.

Die Zeigersumme verrät es: Alle N Zeiger stehen nicht nur dann parallel, wenn der Drehwinkel pro Element null ist, sondern auch, wenn er ein **ganzes Vielfaches von** 2π beträgt. Konstruktive Überlagerung also überall dort, wo

$$kd\sin\theta - \Delta\Phi = 2\pi m \quad\Longrightarrow\quad \sin\theta = \sin\theta_0 + m\,\frac{\lambda}{d}, \qquad m = 0, \pm 1, \pm 2, \ldots$$

Das ist wortwörtlich die Gitterformel aus Kapitel 14 — dort *wollten* wir die hohen Ordnungen (sie sortierten die Farben), hier sind sie ein Ärgernis: Jede Lösung mit $m \neq 0$ ist eine **Grating Lobe** („Gitterkeule"), eine Geisterkeule in voller Stärke, die dieselbe Leistung in eine ungewollte Richtung schickt. Ein Radar mit Grating Lobes sieht Geisterechos; ein Starlink-Panel würde fremde Satelliten stören. Für $d \leq \lambda/2$ (und Schwenk bis zum Horizont) hat $\sin\theta = \sin\theta_0 \pm \lambda/d$ keine Lösung — deshalb unser braver Standardabstand.

```
# von oben: array_faktor(), theta
fig, ax = plt.subplots(figsize=(7.0, 4.2))
for d, farbe in ((0.5, "tab:blue"), (1.5, "tab:red")):
    P = array_faktor(theta, np.ones(8), d=d)
    ax.plot(np.degrees(theta), 10 * np.log10(np.maximum(P, 1e-5)),
            color=farbe, lw=1.4, label=f"d = {d} λ")
for w in (-41.8, 41.8):
    ax.axvline(w, color="tab:red", ls=":", lw=1.1)
ax.set_xlim(-90, 90)
ax.set_ylim(-40, 2)
ax.set_xlabel("Winkel θ zur Breitseite (°)")
ax.set_ylabel("Array-Faktor (dB)")
ax.set_title("d über λ: Geisterkeulen bei sin θ = ±λ/d")
ax.legend()
plt.show()
```

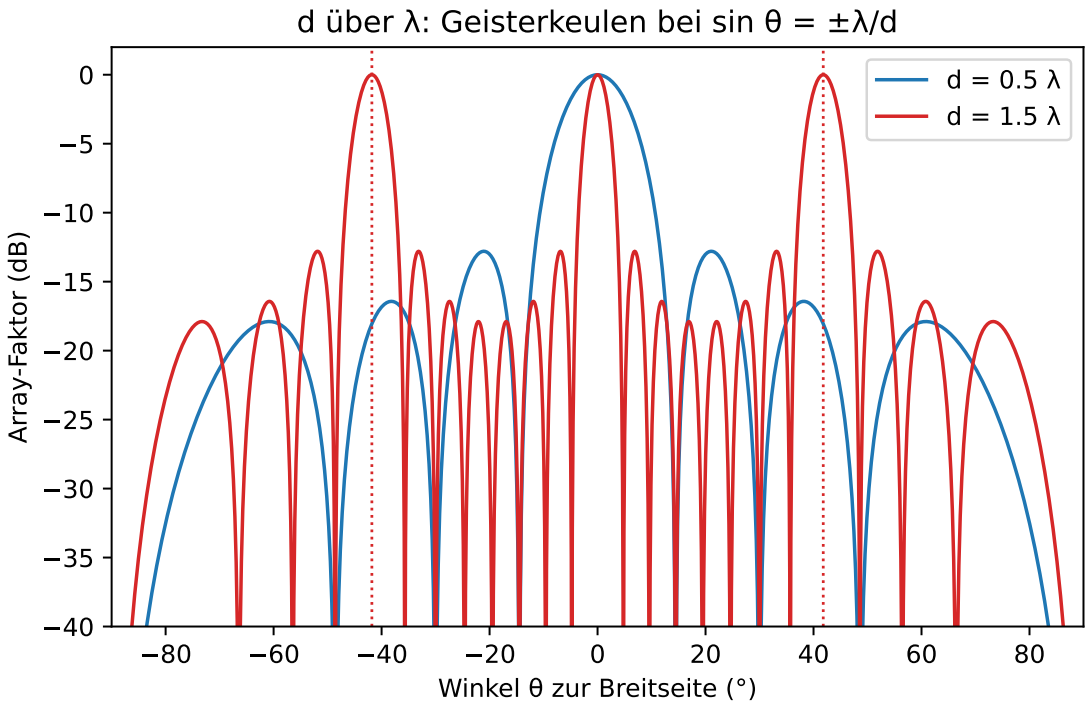


Abbildung 23.4: Acht Elemente ohne Schwenk ($\Delta\Phi = 0$), nur der Abstand wächst: Bei $d = \lambda/2$ (blau) gibt es eine einzige Hauptkeule; bei $d = 1{,}5\ \lambda$ (rot) tauchen Geisterkeulen in voller Höhe bei $\sin\theta = \pm\lambda/d$, also ±41,8° auf (gepunktete Theorie-Senkrechten). Die Hauptkeule selbst ist schmaler geworden — das längere Array bündelt schärfer —, aber der Gewinn ist wertlos, wenn zwei Geister dieselbe Leistung woandershin schicken.

Die Antwort auf den Vorhersage-Punkt: *beides*. Die Hauptkeule wird schmaler — und bekommt zwei Geschwister in voller Höhe. Dass Maxwell das genauso sieht, zeigt das Feldbild — sechs weit auseinandergezogene Quellen, und die Zelle füllt sich mit drei Strahlbündeln:

```
# von oben: cw_feld() — Achtung, wir verstellen kurz N_EL und D_EL
N_EL, D_EL = 6, 1.5
ez_g = cw_feld(0)
print(f"Gesundheitscheck: max|Ez| = {np.abs(ez_g).max():.3f}")
y_q6 = (np.arange(N_EL) - (N_EL - 1) / 2) * D_EL

fig, ax = plt.subplots(figsize=(5.6, 5.2))
inten = (np.abs(ez_g) ** 2).T
```

```
ax.imshow(inten / inten.max(), origin="lower", extent=ext,
          cmap="inferno", vmin=0, vmax=0.25)
ax.plot([X_ARRAY] * N_EL, y_q6, "w.", ms=4)
for vorz in (-1, 0, +1):
    th_g = vorz * np.arcsin(1 / D_EL)
    ax.plot([X_ARRAY, X_ARRAY + 11.5 * np.cos(th_g)],
            [0, 11.5 * np.sin(th_g)], "w--", lw=1.1, alpha=0.85)
ax.set_xticks([]); ax.set_yticks([])
ax.set_title("d = 1,5 λ: ein Strahl bestellt, drei geliefert")
plt.show()
N_EL, D_EL = 8, 0.5
```

```
Gesundheitscheck: max|Ez| = 1.714
```

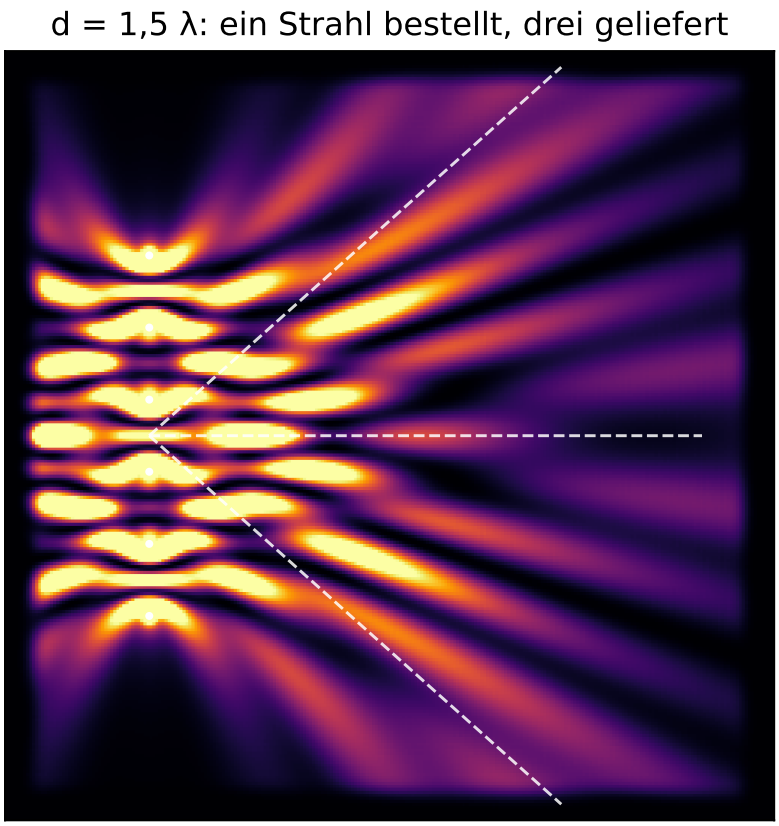


Abbildung 23.5: Das Meep-Feldbild zur Geisterkeulen-Formel: sechs Quellen im Abstand d = 1,5 λ, gleichphasig gespeist (ΔΦ = 0). Die zeitgemittelte Intensität zeigt drei Strahlbündel — die gewollte Hauptkeule zur Breitseite und die beiden Grating Lobes auf den gestrichelten ±41,8°-Theorielinien. Das Array verschenkt zwei Drittel seiner gebündelten Leistung an die Geister.

Beim **Schwenken** verschärft sich die Lage. Die Geister sitzen bei $\sin\theta = \sin\theta_0 \pm \lambda/d$; schwenkt man die Hauptkeule auf θ_0, rückt der Geist auf der Gegenseite näher an den sichtbaren Bereich ($|\sin\theta| \leq 1$) heran. Er bleibt genau dann draußen, wenn

$$\frac{d}{\lambda} < \frac{1}{1 + |\sin\theta_0|}.$$

Wer bis $\pm 90°$ schwenken will, landet wieder bei $d < \lambda/2$; wer nur $\pm 30°$ braucht, darf sich $d < 2\lambda/3$ leisten — mehr Abstand heißt weniger Elemente (und Elektronik!) für dieselbe Arraylänge, deshalb wird hier in der Praxis

gefeilscht. Übung 23.2 lässt dich die Regel an einem konkreten Fall nachmessen.

23.5 Element mal Array: der Faltungssatz der Antennentechnik

Echte Arrays bestehen nicht aus isotropen Punktquellen, sondern aus Dipolen, Patches, Hörnern — Elementen mit eigenem Richtdiagramm. Wie kombiniert sich das Element-Diagramm mit dem Array-Faktor? Kapitel 14 hat die Antwort für den Doppelspalt vorgemacht: Dort war das Muster „Quellen-Streifen × Spaltbreiten-Hüllkurve". Allgemein gilt die **Produktregel**

$$P_{\text{gesamt}}(\theta) = P_{\text{Element}}(\theta) \cdot |F_{\text{Array}}(\theta)|^2,$$

solange alle Elemente gleich sind und gleich ausgerichtet stehen. Warum, zeigt eine hübsche Umformung: Ein „Element mit Struktur" ist selbst eine kleine Quellgruppe — und die Belegung des Gesamtsystems entsteht, indem man die Element-Belegung an jede Array-Position **stempelt**. Dieses Stempeln ist mathematisch eine **Faltung** der beiden Belegungen, und gefaltete Belegungen haben *multiplizierte* Diagramme — der **Faltungssatz**, alter Bekannter aus der Signalverarbeitung.

Das lässt sich in drei Zeilen *exakt* nachprüfen. Als „Element" nehmen wir das kleinste interessante: zwei gegenphasige Quellen (`[1, -1]` — die Übung 14.3!), als Array acht uniforme Plätze:

```
# von oben: array_faktor(), theta
element = np.array([1.0, -1.0])      # gegenphasiges Quellenpaar
gruppe = np.ones(8)                  # 8 Array-Plätze, uniform
# np.convolve: diskrete Faltung — "stempelt" das erste Array an
# jede Position des zweiten und summiert Überlappendes
gesamt = np.convolve(element, gruppe)
print(f"Gesamtbelegung ({len(gesamt)} Elemente): {gesamt}")

links = array_faktor(theta, gesamt, d=0.5, normiert=False)
rechts = (array_faktor(theta, element, d=0.5, normiert=False)
          * array_faktor(theta, gruppe, d=0.5, normiert=False))
print(f"Faltungssatz | Produkt − Direktrechnung |: "
      f"{np.abs(links - rechts).max() / links.max():.1e}")
```

```
Gesamtbelegung (9 Elemente): [ 1.  0.  0.  0.  0.  0.  0.  0. -1.]
Faltungssatz | Produkt − Direktrechnung |: 3.7e-15
```

Maschinengenauigkeit — Diagramme multiplizieren sich, wenn Belegungen sich falten. Die Gesamtbelegung liest sich dabei fast von selbst: Das Quellenpaar an Platz 0 bis 7 gestempelt gibt `[1, 0, 0, …, 0, -1]` — innen löschen sich die Stempel aus, übrig bleiben die Ränder. Merke die Richtung der Regel: **Ortsraum gefaltet = Winkelraum multipliziert.**

Und damit fällt ein zweiter Groschen. Die Zeigersumme $F = \sum_n a_n e^{in\psi}$ ist — Buchstabe für Buchstabe — die **diskrete Fouriertransformation der Belegung** a_n, ausgewertet am Winkel $\psi = kd \sin\theta - \Delta\Phi$. Das Richtdiagramm eines Arrays ist das *Spektrum* seiner Belegung:

```
# von oben: nichts Neues — np.fft.fft ist die komplexe Voll-DFT
# (Kap. 10 nutzte rfft für reelle Signale; fft nimmt auch komplexe
# Eingaben und liefert alle M Frequenzen). Die 8er-Belegung wird
# auf 4096 Punkte mit Nullen aufgefüllt ("zero-padding"): mehr
# Auswertepunkte derselben Summe, keine neue Information.
M = 4096
F_dft = np.fft.fft(np.ones(8), M)
psi_dft = 2 * np.pi * np.arange(M) / M
zeiger = np.sum(np.exp(-1j * np.arange(8)[None, :]
                       * psi_dft[:, None]), axis=1)
print(f"DFT gegen Zeigersumme: max. Abweichung "
      f"{np.abs(np.abs(F_dft)**2 - np.abs(zeiger)**2).max():.1e}")
```

```
DFT gegen Zeigersumme: max. Abweichung 1.3e-13
```

Wieder Maschinengenauigkeit. Diese Identität ist mehr als eine Kuriosität: Sie verdrahtet das Antennenkapitel mit Kapitel 10 — und sie macht die nächste Frage trivial.

23.6 Taper: die Nebenkeulen zähmen

In Abbildung 23.1 blieben die Nebenkeulen stur bei etwa -13 dB stehen, egal wie viele Elemente wir spendierten. Für ein Radar sind -13 dB viel: Ein starkes Echo in einer Nebenkeule ist von einem schwachen in der Hauptkeule nicht zu unterscheiden. Woher kommt die Zahl?

Aus Kapitel 10, jetzt rückwärts gelesen. Eine uniforme Belegung ist ein **Rechteckfenster** — und ihr Spektrum ist die sinc-Funktion mit ihrer ersten Nebenkeule bei $-13{,}3$ dB (für endliches N etwas darüber; gleich gemessen). In Kapitel 10 hieß das Problem „Leck-Effekt", und die Lösung hieß **Fensterfunktion**: die Belegung zu den Rändern hin sanft ausblenden. In

der Antennenwelt heißt dieselbe Lösung **Amplituden-Taper** — die äußeren Elemente schwächer speisen.

Der Faltungssatz liefert den schönsten Einstieg: Falte das Rechteck mit sich selbst, und du bekommst ein **Dreieck** (`np.convolve(np.ones(8), np.ones(8))` ergibt 1, 2, …, 8, …, 2, 1). Gefaltete Belegung = multipliziertes Diagramm — das Dreieck-Diagramm ist das *Quadrat* des Rechteck-Diagramms, also muss jede Nebenkeule in Dezibel exakt **doppelt** so tief liegen. Und am Ende dieser Logik wartet die **binomische Belegung** (die Zeile aus dem Pascal-Dreieck): das Rechteck `[1, 1]` sieben Mal mit sich selbst gefaltet, das Diagramm also die siebte Potenz des Zweierzeigers — und der hat als einziger *gar keine* Nebenkeulen.

```
# von oben: array_faktor(), theta, P_8
import math


def nebenkeule_db(P):
    """Pegel der höchsten Nebenkeule (lokale Maxima unter 0,99)."""
    i = np.flatnonzero((P[1:-1] > P[:-2]) & (P[1:-1] > P[2:])) + 1
    pegel = [p for p in sorted(P[i], reverse=True) if 1e-9 < p < 0.99]
    return 10 * np.log10(pegel[0]) if pegel else -np.inf


dreieck = np.convolve(np.ones(8), np.ones(8))
# math.comb: Binomialkoeffizient "7 über n" — die Pascal-Zeile
binom = np.array([math.comb(7, n) for n in range(8)], float)

nk_uni = nebenkeule_db(P_8)
nk_dreieck = nebenkeule_db(array_faktor(theta, dreieck, d=0.5))
print(f"Nebenkeule uniform (N = 8):  {nk_uni:.2f} dB  "
      "(sinc-Grenzwert −13,26)")
print(f"Nebenkeule Dreieck (N = 15): {nk_dreieck:.2f} dB  "
      f"(2 × uniform = {2 * nk_uni:.2f})")
print(f"binomisch {binom}: Nebenkeulen unter −90 dB? "
      f"{nebenkeule_db(array_faktor(theta, binom, d=0.5)) < -90}")
```

```
Nebenkeule uniform (N = 8):  -12.80 dB  (sinc-Grenzwert −13,26)
Nebenkeule Dreieck (N = 15): -25.59 dB  (2 × uniform = -25.59)
binomisch [ 1.  7. 21. 35. 35. 21.  7.  1.]: Nebenkeulen unter −90 dB? True
```

Die Dreieck-Vorhersage trifft auf zwei Nachkommastellen: $-25{,}59 = 2 \cdot (-12{,}80)$ dB. Kein Fit, keine Näherung — der Faltungssatz in Aktion. Aber Ruhe hat ihren Preis:

```
# von oben: array_faktor(), theta, dreieck, binom


def fwhm_grad(P):
    i = np.flatnonzero(P >= 0.5)
    return np.degrees(theta[i[-1]] - theta[i[0]])


fig, ax = plt.subplots(figsize=(7.0, 4.2))
for beleg, name, farbe in ((np.ones(8), "uniform", "tab:blue"),
                           (dreieck, "Dreieck", "tab:orange"),
                           (binom, "binomisch", "tab:green")):
    P = array_faktor(theta, beleg, d=0.5)
    ax.plot(np.degrees(theta), 10 * np.log10(np.maximum(P, 1e-7)),
            color=farbe, lw=1.4,
            label=f"{name} (FWHM {fwhm_grad(P):.1f}°)")
ax.axhline(-12.8, color="tab:blue", ls=":", lw=1)
ax.set_xlim(-90, 90)
ax.set_ylim(-65, 2)
ax.set_xlabel("Winkel θ zur Breitseite (°)")
ax.set_ylabel("Array-Faktor (dB)")
ax.set_title("Taper: Nebenkeulen gegen Keulenbreite tauschen")
ax.legend(fontsize=9)
plt.show()
```

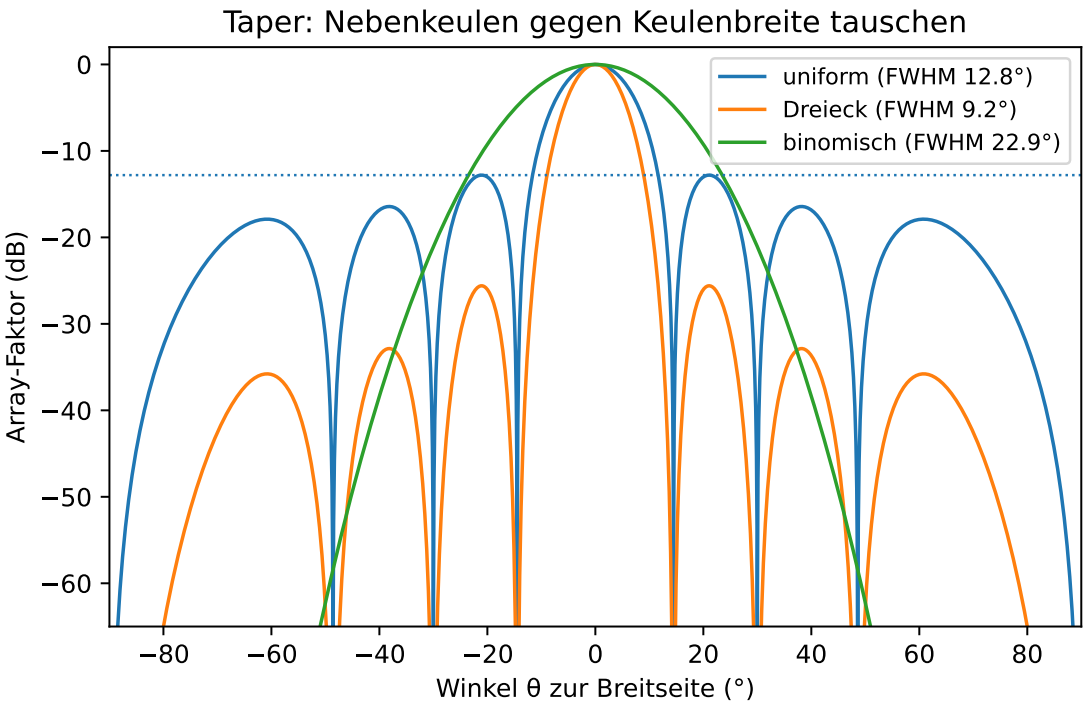


Abbildung 23.6: Drei Belegungen, drei Diagramme ($d = \lambda/2$; uniform und binomisch mit $N = 8$, das gefaltete Dreieck hat $N = 15$ — sein Vorteil ist trotz der fast doppelten Arraylänge deutlich): Die uniforme Belegung (blau) hat die schmalste Hauptkeule, aber −12,8-dB-Nebenkeulen. Das Dreieck (orange) verdoppelt die Nebenkeulen-Tiefe in Dezibel exakt (−25,6 dB), die binomische Belegung (grün) hat gar keine mehr — bezahlt aber mit der breitesten Hauptkeule. Nebenkeulen lassen sich gegen Keulenbreite eintauschen, nie abschaffen.

Die uniforme Belegung bündelt am schärfsten (sie nutzt jede Quelle voll aus), die binomische ist vollkommen nebenkeulenfrei und fast doppelt so breit. Praktische Arrays (Radar: Taylor-Belegung; Spektralanalyse: Hann-Fenster) liegen dazwischen — es ist in beiden Welten derselbe Kompromiss, weil es dieselbe Mathematik ist. Wer in Kapitel 22 den Exponential-Taper gegen Reflexionen gesehen hat, erkennt auch ihn wieder: sanfte Übergänge statt harter Kanten, dort längs der Leitung, hier quer über die Apertur.

23.7 Scan Loss: der Preis des Schwenkens

Zurück zu den Feldbildern in Abbildung 23.2: Beim Schwenk auf $48{,}6°$ war die Keule sichtbar *breiter* als bei der Breitseite. Der Grund ist Geometrie — von schräg gesehen **erscheint das Array verkürzt**. Wirksam ist nur die Projektion der Apertur auf die Blickrichtung, $Nd\cos\theta_0$, und mit der Apertur schrumpft die Bündelung:

$$D(\theta_0) \;\approx\; D(0)\,\cos\theta_0 .$$

Das ist der **Scan Loss**. Wir messen ihn am Array-Faktor: Die Direktivität in 2D ist (ganz wie in Kapitel 21, nur mit Kreis- statt Kugelintegral) $D = 2\pi P_{\max}/\oint P\,\mathrm{d}\varphi$ — das Integral läuft über den vollen Kreis, denn unsere Quellreihe strahlt nach vorn *und* hinten je eine Keule (das kürzt sich im Verhältnis $D(\theta_0)/D(0)$ heraus):

```
# von oben: array_faktor(), k
theta_voll = np.linspace(-np.pi, np.pi, 400001)


def direktivitaet_2d(dphi_grad):
    P = array_faktor(theta_voll, np.ones(8), d=0.5,
                     dphi_grad=dphi_grad, normiert=False)
    return 2 * np.pi * P.max() / np.trapezoid(P, theta_voll)


D_breitseite = direktivitaet_2d(0)
schwenk = np.arange(0, 66, 5)
D_rel = np.array([direktivitaet_2d(np.degrees(k * 0.5 *
                                              np.sin(np.radians(t))))
                  for t in schwenk]) / D_breitseite

fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.plot(schwenk, np.cos(np.radians(schwenk)), "-", color="0.55",
        lw=1.6, label="cos θ₀ (projizierte Apertur)")
ax.plot(schwenk, D_rel, "bo", ms=5, label="Messung am Array-Faktor")
ax.set_xlabel("Schwenkwinkel θ₀ (°)")
ax.set_ylabel("D(θ₀) / D(0)")
ax.set_title(f"Scan Loss: D(0) = {D_breitseite:.1f}, und schräg "
             "wird das Array kleiner")
ax.legend()
plt.show()
```

```
print(f"θ₀ = 40°: D/D₀ = {D_rel[schwenk == 40][0]:.3f}   "
      f"(cos 40° = {np.cos(np.radians(40)):.3f})")
```

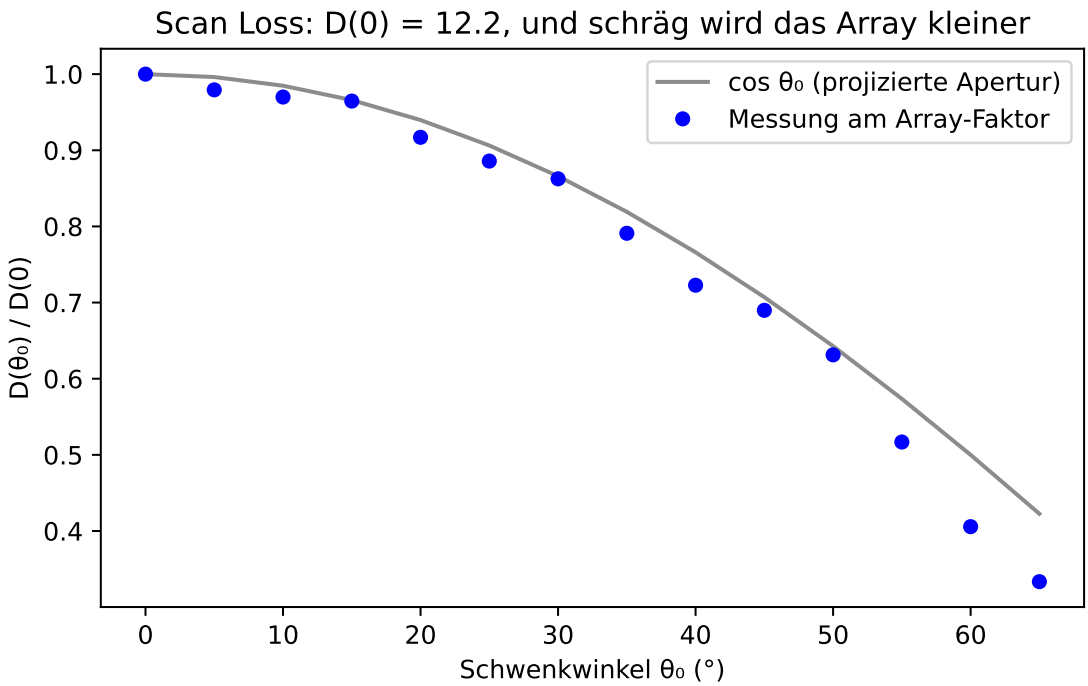


Abbildung 23.7: Scan Loss am Array-Faktor (N = 8, d = λ/2): die relative Direktivität $D(\theta_0)/D(0)$ für Schwenkwinkel bis 65° (blaue Punkte) gegen die cos-θ_0-Regel der projizierten Apertur (graue Linie). Bis etwa 45° folgt die Messung der Regel; darüber fällt sie schneller, weil die verbreiterte Keule zusätzlich am Rand des sichtbaren Bereichs „anstößt“. Schwenken kostet — ein Phased Array, das bis 60° schauen soll, muss diesen Verlust im Leistungsbudget einplanen.

```
θ₀ = 40°: D/D₀ = 0.723   (cos 40° = 0.766)
```

23.7.1 Echte Dipole: NEC rechnet die Verkopplung mit

Punktquellen kennen keine Nachbarn. Echte Dipole schon: Jeder steht im Nahfeld der anderen, ihre Ströme beeinflussen sich gegenseitig (**Verkopplung**), und beim Schwenken ändert sich diese Beeinflussung mit. Für Drahtantennen ist NEC das Werkzeug der Wahl (Kapitel 11; Drahtantennen kann Meep nicht, das war die Yagi-Lektion).

Der Versuchsaufbau. **Frage:** Folgt auch eine Reihe *echter* Dipole dem Zündplan, und was kostet der Schwenk wirklich? **Bühne:** sechs Halbwellendipole ($0{,}47\,\lambda$, Drahtradius 1 mm) im Abstand $d = \lambda/2$ bei 2450 MHz — die Maße der Yagi-Werkstatt aus Kapitel 11. **Messgrößen:** das Azimut-Gewinndiagramm in dBi für fünf Zündpläne; Keulenrichtung (Maximum im vorderen Halbraum) und Spitzengewinn. **Erfolgskriterium:** Keule auf $\arcsin(\Delta\Phi/(kd))$ innerhalb der 1°-Rasterung; der Gewinnabfall sollte *mindestens* der cos-Regel folgen. Eine Konventionsfalle vorweg: NEC stammt aus der Elektrotechnik und rechnet mit $e^{+\mathrm{j}\omega t}$ (Meep: $e^{-\mathrm{i}\omega t}$) — „später zünden" heißt dort **negative** Phase, unser Schwenk nach $+\theta$ braucht also $\Delta\Phi < 0$ auf der `ex_card`.

```
# von oben: k; NEC wie in Kap. 11 (necpp, SI-Einheiten!)
import necpp


def nec_array_diagramm(dphi_grad, n_dipole=6, d_lambda=0.5,
                       f_mhz=2450.0):
    """Azimut-Gewinn (dBi) einer phasengespeisten Dipolreihe."""
    lam = 299792458.0 / (f_mhz * 1e6)
    ctx = necpp.nec_create()
    for i in range(n_dipole):
        y = (i - (n_dipole - 1) / 2) * d_lambda * lam
        halb = 0.47 * lam / 2
        necpp.nec_wire(ctx, i + 1, 11, 0, y, -halb, 0, y, halb,
                       0.001, 1.0, 1.0)
    necpp.nec_geometry_complete(ctx, 0)
    necpp.nec_fr_card(ctx, 0, 1, f_mhz, 0)
    for i in range(n_dipole):
        ph = np.radians(dphi_grad * i)
        # Speisespannung mit Phase n·ΔΦ (Real-/Imaginärteil)
        necpp.nec_ex_card(ctx, 0, i + 1, 6, 0,
                          np.cos(ph), np.sin(ph), 0, 0, 0, 0)
    npts = 361
    necpp.nec_rp_card(ctx, 0, 1, npts, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                      360.0 / (npts - 1), 0.0, 0.0)
    gewinn = np.array([necpp.nec_gain(ctx, 0, 0, j)
                       for j in range(npts)])
    necpp.nec_delete(ctx)
    return np.linspace(0, 360, npts), gewinn


print("ΔΦ        Keule    Theorie    Gewinn     Scan Loss   cos-Regel")
g_breitseite = None
```

```
for dphi in (0, -30, -60, -90, -120):
    phi_nec, g = nec_array_diagramm(dphi)
    vorn = (phi_nec <= 90) | (phi_nec >= 270)     # vorderer Halbraum
    i_peak = np.flatnonzero(vorn)[np.argmax(g[vorn])]
    keule = phi_nec[i_peak] if phi_nec[i_peak] <= 180 \
        else phi_nec[i_peak] - 360
    theo = np.degrees(np.arcsin(-np.radians(dphi) / (k * 0.5)))
    if g_breitseite is None:
        g_breitseite = g.max()
    cos_db = 10 * np.log10(np.cos(np.radians(theo)))
    print(f"{dphi:+5d}°   {keule:+6.1f}°  {theo:+7.2f}°   "
          f"{g.max():5.2f} dBi   {g.max() - g_breitseite:+5.2f} dB"
          f"   {cos_db:+5.2f} dB")
```

```
ΔΦ        Keule    Theorie    Gewinn     Scan Loss   cos-Regel
   +0°     +0.0°    -0.00°   11.05 dBi   +0.00 dB   +0.00 dB
  -30°     +9.0°    +9.59°   10.93 dBi   -0.12 dB   -0.06 dB
  -60°    +19.0°   +19.47°   10.67 dBi   -0.38 dB   -0.26 dB
  -90°    +29.0°   +30.00°   10.18 dBi   -0.87 dB   -0.62 dB
 -120°    +40.0°   +41.81°    9.62 dBi   -1.43 dB   -1.28 dB
```

```
# von oben: nec_array_diagramm(), k
fig = plt.figure(figsize=(6.2, 5.6))
ax = fig.add_subplot(projection="polar")
for dphi, farbe in ((0, "tab:blue"), (-60, "tab:green"),
                    (-120, "tab:red")):
    phi_nec, g = nec_array_diagramm(dphi)
    theo = np.degrees(np.arcsin(-np.radians(dphi) / (k * 0.5)))
    ax.plot(np.radians(phi_nec), np.clip(g - g.max(), -30, 0),
            color=farbe, lw=1.6,
            label=f"ΔΦ = {dphi}° → {theo:.0f}° ({g.max():.1f} dBi)")
ax.set_theta_zero_location("E")
ax.set_rticks([-30, -20, -10, -3, 0])
ax.set_rlabel_position(135)
ax.set_title("NEC: sechs echte Dipole folgen dem Zündplan", pad=18)
ax.legend(loc="lower left", bbox_to_anchor=(-0.14, -0.12), fontsize=8)
plt.show()
```

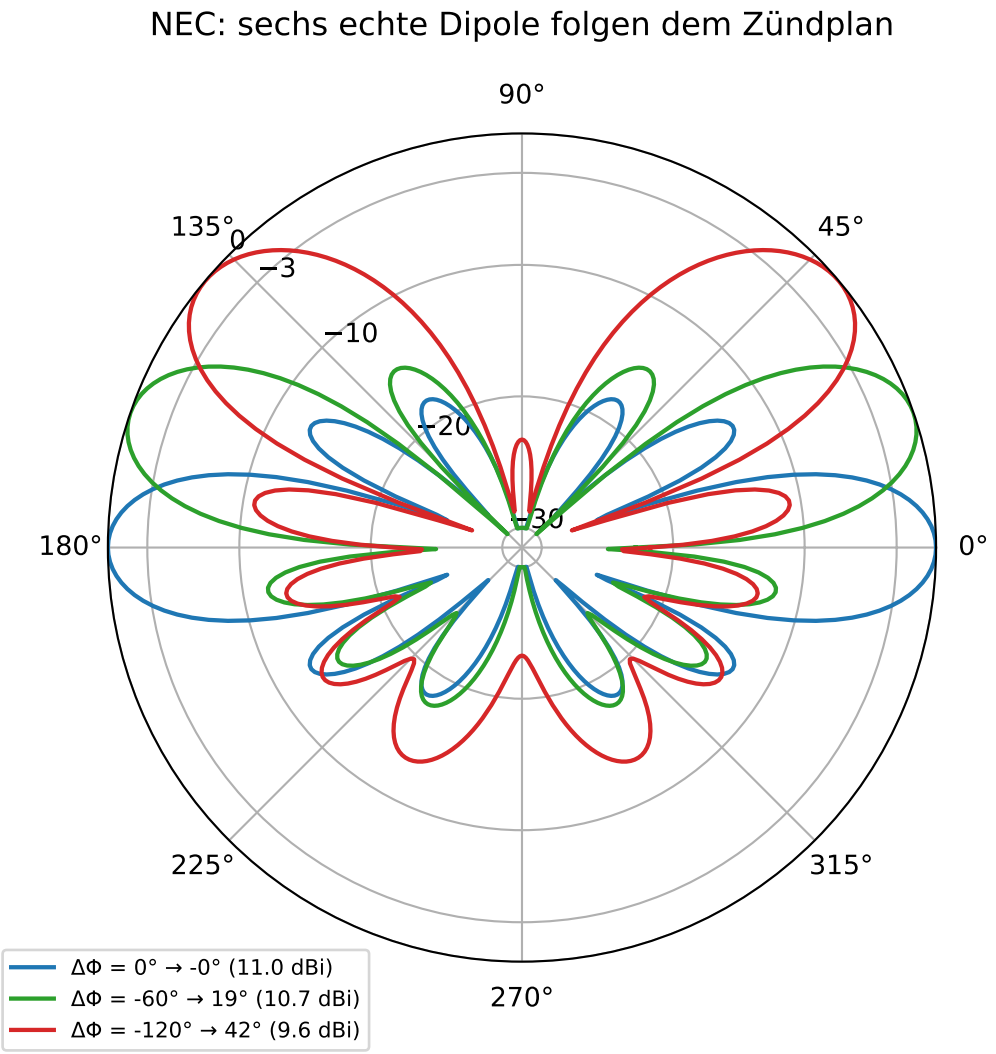


Abbildung 23.8: Drei Zündpläne der 6-Dipol-Reihe als NEC-Azimutdiagramm (Polarbild, Gewinn in dBi relativ zum jeweiligen Maximum, 30-dB-Dynamik; 0° = Breitseite). Die Keule wandert von 0° über 19° nach 42° — echte Drähte folgen demselben Umweg-gleich-Vorsprung-Gesetz wie die Punktquellen. Der Spitzengewinn fällt dabei von 11,05 auf 9,62 dBi; die Rückseite spiegelt das Bild, weil eine Dipolreihe ohne Reflektor nicht zwischen vorn und hinten unterscheiden kann.

Drei Befunde. Erstens: Die Keule sitzt für alle Zündpläne auf der Theorie (innerhalb der 1°-Rasterung — bei $-120°$ liegt das Maximum bei $40°$ statt $41{,}8°$, weil die beim Schwenken breit gewordene Keule auf dem Gradraster abge-

tastet wird). Zweitens: Der Scan Loss ist real — $-1{,}43$ dB bei $41{,}8°$. Drittens, und das ist die ehrliche Pointe: Er ist *größer* als die cos-Regel ($-1{,}28$ dB). Die Differenz ist die **Verkopplung** — die Dipole stehen einander im Nahfeld, und dieser Effekt steckt in keiner Zeigersumme. Genau dafür braucht man Werkzeuge wie NEC: Der Array-Faktor liefert das Design, der Feldlöser die Wahrheit (die Validierungs-Checkliste aus Kapitel 10 lässt grüßen).

23.8 Empfang ist Senden rückwärts

Bisher hat das Array gesendet. Die **Reziprozität** der Maxwell-Gleichungen (sie steckt letztlich in der Zeitumkehr-Symmetrie aus Kapitel 20) garantiert: Dasselbe Array mit demselben Zündplan **empfängt** auch am besten aus der Richtung θ_0 — man verzögert die N Empfangssignale elektronisch um genau die Zündplan-Zeiten und addiert sie. Signale aus θ_0 addieren sich dann phasenrichtig (N-fache Amplitude, N^2-fache Leistung), Signale aus anderen Richtungen verzettelt die Zeigersumme. Das Array „lauscht" in eine wählbare Richtung — und kann mehr: Mit N Phasenknöpfen lassen sich nicht nur Keulen formen, sondern auch gezielt **Nullstellen** auf Störer legen (Übung 23.4 baut so einen Störblocker). 5G-Masten, WLAN-Router mit mehreren Antennen und das Starlink-Terminal tun beides zugleich: senden und lauschen, jeweils mit eigenem Zündplan.

23.9 Das Kapitel-Programm

`programme/kap23/kap23_arrays_beamforming.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die $N = 2$-Doppelspalt-Identität und die Schwenkreihe (auf $0{,}05°$), Faltungssatz und DFT-Identität (Maschinengenauigkeit), die Taper-Kette uniform/Dreieck/binomisch ($-12{,}80$ dB, exakt verdoppelt, nebenkeulenfrei), Grating Lobes bei $d = 1{,}5\,\lambda$ samt Schwenk-Designregel, die Scan-Loss-Reihe, den Meep-Doppelcheck (Feldbild-Gesundheitswert und near2far gegen Array-Faktor, rms < 0,01), die NEC-Schwenkreihe (Keulen auf 2°) und den Null-Steuer-Trick (Störer < -60 dB). Läuft in einer guten Minute.

Merkkasten

- **Umweg = Vorsprung:** Die Keule einer Strahlerreihe zeigt dorthin, wo der Wegunterschied den Zündplan ausgleicht — $\sin\theta_0 = \Delta\Phi/(kd)$. Richtung ist Phasensache, keine Geometriesache.
- **Der Array-Faktor ist die DFT der Belegung**, ausgewertet bei

$\psi = kd\sin\theta - \Delta\Phi$; $N = 2$ ist der Doppelspalt. Arraylänge Nd setzt die Keulenbreite ($\sin\theta_{\text{Null}} = \lambda/(Nd)$).

- **Grating Lobes:** Geisterkeulen bei $\sin\theta = \sin\theta_0 \pm m\lambda/d$; fernhalten mit $d/\lambda < 1/(1 + |\sin\theta_0|)$ — voll schwenkbar heißt $d \leq \lambda/2$.
- **Faltungssatz:** Belegungen falten sich, Diagramme multiplizieren sich (Element × Array-Faktor). Taper = Fensterfunktion: Nebenkeulen gegen Keulenbreite, nie umsonst (uniform -13 dB, Dreieck doppelt so tief, binomisch keulenfrei).
- **Schwenken kostet** (Scan Loss $\approx \cos\theta_0$ aus der projizierten Apertur, real mehr durch Verkopplung), und **Empfang = Senden rückwärts** (Reziprozität): verzögern + addieren, Keulen *und* Nullen steuerbar.

Roter Faden

Dieses Kapitel ist die Ernte von Teil IV: Der **Doppelspalt und das Zeigerbild aus Kapitel 14** wurden rückwärts gelesen (die Übungen 14.3/14.4 — Phasen-Hebel und $d < \lambda$-Regel — waren die Vorboten), die **DFT aus Kapitel 10** entpuppte sich als Diagramm-Maschine samt Fenster-Kompromiss, das **near2far-Fernfeld und die Direktivität aus Kapitel 21** lieferten das Messgeschirr, und die **Reziprozität** lehnt sich an die Zeitumkehr aus Kapitel 20. Auch **Kapitel 22** wirkt mit: Jedes der N Elemente braucht seine eigene angepasste Speiseleitung — ein Phased Array ist immer auch ein Leitungs-Kunstwerk. Nach vorn: **Kapitel 24** baut den Gegenentwurf — die Yagi erzeugt ihre Richtwirkung mit *passiven* Elementen, deren „Zündplan" die Verkopplung selbst übernimmt; **Kapitel 26** lässt aus Quellreihen ganze Kristallgitter werden (Bragg statt Grating); und das **inverse Design in Kapitel 31** wird Belegungen nicht mehr wählen, sondern *optimieren* lassen.

Übungen

Ü 23.1 (Verstehen). Ein 5G-Mast soll seine Keule um $25°$ aus der Breitseite schwenken (Elementabstand $d = \lambda/2$). Berechne von Hand den nötigen Phasenversatz $\Delta\Phi$ zwischen Nachbarelementen — und erkläre in zwei Sätzen, warum derselbe Trick mit einer *einzigen* Antenne prinzipiell nicht funktioniert.

Musterlösung zu Ü 23.1

$\Delta\Phi = kd \sin\theta_0 = 2\pi \cdot \frac{1}{2} \cdot \sin 25° = \pi \cdot 0{,}4226 = 1{,}328 \text{ rad} = 76{,}1°$:

```
# von oben: k (= 2π, λ = 1)
dphi = np.degrees(k * 0.5 * np.sin(np.radians(25)))
print(f"ΔΦ = {dphi:.1f}° pro Element")
```

```
ΔΦ = 76.1° pro Element
```

Mit einer einzigen Antenne gibt es nichts, *wogegen* man eine Phase verstellen könnte: Richtwirkung entsteht aus Interferenz **mehrerer** Teilwellen mit definierten Phasenbeziehungen. Eine Einzelantenne kann ihr Diagramm nur über ihre Geometrie ändern — und die müsste man eben drehen.

Ü 23.2 (Verändern). Ein Entwickler wählt $d = 0{,}7\,\lambda$ (spart Elemente) und will bis $\theta_0 = 50°$ schwenken. Prüfe zuerst mit der Designregel $d/\lambda < 1/(1+|\sin\theta_0|)$, ob das gut geht, und sieh dir dann das Diagramm mit `array_faktor` an: Wo sitzt die Geisterkeule, und wie stark ist sie? Bis zu welchem Schwenkwinkel wäre $d = 0{,}7\,\lambda$ in Ordnung?

Musterlösung zu Ü 23.2

Die Regel verlangt $d/\lambda < 1/(1+\sin 50°) = 0{,}566$ — mit $0{,}7$ klar verletzt. Die Geisterkeule sitzt bei $\sin\theta = \sin 50° - 1/0{,}7 = -0{,}663$, also $-41{,}5°$, und zwar in **voller Höhe** (0 dB — sie ist keine Nebenkeule, sondern eine zweite Hauptkeule). Gut geht es, solange $\sin\theta_0 < 1/0{,}7 - 1 = 0{,}4286$, also bis $\theta_0 = 25{,}4°$:

```
# von oben: array_faktor(), theta, k
dphi_50 = np.degrees(k * 0.7 * np.sin(np.radians(50)))
P = array_faktor(theta, np.ones(8), d=0.7, dphi_grad=dphi_50)
i_max = np.flatnonzero((P[1:-1] > P[:-2]) & (P[1:-1] > P[2:])) + 1
gross = i_max[P[i_max] > 0.5]
print(f"Keulen über −3 dB bei {np.round(np.degrees(theta[gross]), 1)}°")
print(f"max. sauberer Schwenk: "
      f"{np.degrees(np.arcsin(1 / 0.7 - 1)):.1f}°")
```

```
Keulen über −3 dB bei [-41.5  50. ]°
max. sauberer Schwenk: 25.4°
```

Das gesparte Material bezahlt man also mit einem geschrumpften

Schwenkbereich — genau dieser Handel wird bei jedem realen Array ausgefochten.

Ü 23.3 (Verändern). Die binomische Belegung hat gar keine Nebenkeulen — warum baut dann nicht jeder Radarbauer binomisch? Miss den Preis nach: Vergleiche für $N = 8$, $d = \lambda/2$ die Halbwertsbreite (FWHM) *und* die 2D-Direktivität der uniformen und der binomischen Belegung (nimm die `direktivitaet_2d`-Logik aus dem Scan-Loss-Abschnitt und gib ihr eine Belegung mit).

Musterlösung zu Ü 23.3

```python
# von oben: array_faktor(), theta, theta_voll, binom, fwhm_grad()


def D_2d(belegung):
    P = array_faktor(theta_voll, belegung, d=0.5, normiert=False)
    return 2 * np.pi * P.max() / np.trapezoid(P, theta_voll)


for beleg, name in ((np.ones(8), "uniform"), (binom, "binomisch")):
    P = array_faktor(theta, beleg, d=0.5)
    print(f"{name:10s}: FWHM = {fwhm_grad(P):4.1f}°,   "
          f"D_2D = {D_2d(beleg):5.2f}")
```

```
uniform   : FWHM = 12.8°,   D_2D = 12.16
binomisch : FWHM = 22.9°,   D_2D =  7.39
```

Die binomische Keule ist fast doppelt so breit, und die Direktivität fällt von $12{,}2$ auf $7{,}4$ — fast die Hälfte der Bündelung ist weg. Nebenkeulenfreiheit ist der teuerste Punkt auf der Tauschkurve; reale Entwürfe (Taylor-, Tschebyscheff-Belegungen) kaufen „tief genug" statt „ganz weg". Außerdem verlangt die binomische Belegung Speiseamplituden über einen Faktor 35 — auch das will erst einmal sauber gebaut sein (Kapitel 22 lässt grüßen).

Ü 23.4 (Übertragen). Ein GPS-Empfänger mit 8-Element-Array empfängt sein Nutzsignal aus $0°$ — und einen Störsender aus $40°$. Baue den Störblocker: Berechne zuerst, wie stark die uniforme Belegung (Keule auf $0°$) den Störer von sich aus dämpft. Lege dann eine **Nullstelle** auf $40°$, indem du vom uniformen Gewichtsvektor seine Projektion auf den Störer-Steering-Vektor $v_n = e^{ink d \sin 40°}$ abziehst — und miss, was das den Empfang aus $0°$ kostet. (Empfangs-Beamforming: Die „Belegung" sind jetzt die Gewichte,

mit denen die N Empfangssignale addiert werden — konjugiert, weil Empfangen der zeitumgekehrte Sendefall ist.)

Musterlösung zu Ü 23.4

```
# von oben: array_faktor(), theta, k
n8 = np.arange(8)
v_stoer = np.exp(1j * n8 * k * 0.5 * np.sin(np.radians(40)))
w_uni = np.ones(8, complex)
# np.vdot: Skalarprodukt, das das ERSTE Argument konjugiert —
# für komplexe Vektoren das richtige „wie viel a steckt in b"
# Projektion abziehen: w ⊥ v_stoer ⇒ Summe über den Störer = 0
w_null = w_uni - (np.vdot(v_stoer, w_uni)
                  / np.vdot(v_stoer, v_stoer)) * v_stoer

i_40 = np.argmin(np.abs(theta - np.radians(40)))
for w, name in ((w_uni, "uniform"), (w_null, "mit Null")):
    P = array_faktor(theta, np.conj(w), d=0.5, normiert=False)
    print(f"{name:9s}: Störer (40°) {10 * np.log10(P[i_40] / P.max()):8.1f} dB"
          f"   Maximum bei {np.degrees(theta[np.argmax(P)]):+5.1f}°"
          f"   Pegel dort {10 * np.log10(P.max() / 8**2):+6.2f} dB")
```

```
uniform  : Störer (40°)   -16.8 dB  Maximum bei  -0.0°  Pegel dort  +0.00 dB
mit Null : Störer (40°)  -110.4 dB  Maximum bei  -0.1°  Pegel dort  -0.18 dB
```

Die uniforme Belegung dämpft den Störer nur um knapp 17 dB (er sitzt zufällig nahe einer Nebenkeule). Die Projektion drückt ihn um über 100 dB — praktisch auf null —, während die Hauptkeule auf $0°$ stehen bleibt und nur $0{,}18$ dB verliert. Acht Phasenknöpfe, ein Skalarprodukt: mehr braucht ein adaptiver Störblocker im Kern nicht. Genau so (nur mit laufend nachgeführten Gewichten) schützen sich GPS-Empfänger in Flugzeugen gegen Jammer.

Das Kleingedruckte

Verkopplung und aktive Impedanz. Der NEC-Befund kratzte nur an der Oberfläche: Im dichten Array ändert sich die Eingangsimpedanz jedes Elements mit dem Schwenkwinkel („aktive Impedanz”), weil die Nachbarn mit wechselnder Phase in den Speisepunkt zurückstrahlen. Bei manchen Winkeln kann ein Array dadurch fast vollständig reflektieren („Scan Blindness”)

— die Anpasskunst aus Kapitel 22 muss dann über den ganzen Schwenkbereich funktionieren.

Fläche statt Reihe. Reale Panels sind 2D-Gitter (das Starlink-Terminal: rund 1500 Patch-Elemente). Die gute Nachricht steckt im Faltungssatz: Eine Rechteck-Fläche ist die Faltung aus Zeile und Spalte, ihr Diagramm also das Produkt zweier Linien-Array-Faktoren — alles in diesem Kapitel gilt pro Achse weiter.

Phase ist nicht Laufzeit. Unser Phasenknopf stellt *Phasen* bei einer Frequenz. Breitbandige Signale brauchen echte **Zeit**-Verzögerungen, sonst schielt die Keule pro Frequenz woandershin („Beam Squint" — die Gitterformel hängt ja an λ). Große Radare mischen deshalb beides: grobe echte Verzögerung, feine Phase.

Digital wird es mächtig. Wer hinter jedem Element einen eigenen Empfänger samt Analog-Digital-Wandler hat, kann die Gewichte in Software setzen — und zwar *viele Sätze gleichzeitig*: mehrere Keulen, mehrere Nullen, pro Nutzer ein eigener Strahl (Massive MIMO im 5G-Netz). Aus demselben Datensatz lassen sich nachträglich beliebige Richtungen formen — Beamforming wird vom Hardware- zum Rechenproblem.

Und die Konvention. NEC ($e^{+\mathrm{j}\omega t}$, Elektrotechnik) und Meep ($e^{-\mathrm{i}\omega t}$, Physik) sind sich übers Vorzeichen der Phase uneins — dieselbe Zündverspätung heißt einmal $-\Delta\Phi$, einmal $+\Delta\Phi$. Das ist keine Physik, nur Buchhaltung; aber wer Phasen zwischen Werkzeugen überträgt, prüft das Vorzeichen am besten an einem Fall mit bekannter Antwort — so wie unsere 30°-Handrechnung.

24 Fallstudie Yagi: Entwurf, Validierung, Bauplan

Das Starlink-Panel aus Kapitel 23 bündelt seinen Strahl mit rund 1500 gespeisten Elementen, ebenso vielen Phasenstellern und einem Rechner, der den Zündplan tausendmal pro Sekunde nachführt. Auf Millionen Dächern steht seit siebzig Jahren eine Antenne, die fast dasselbe leistet — mit **einem einzigen Kabel**: die Yagi-Uda-Antenne, die klassische Fernseh- und WLAN-Richtantenne. Ein gespeister Dipol, dahinter ein Stab, davor ein paar kürzere Stäbe. Die Stäbe hängen an nichts; sie sind einfach Metall im Raum. Und trotzdem wirft die Antenne fast ihre gesamte Leistung nach vorn.

Wie bringt man tote Metallstäbe dazu, im Takt mitzusenden — und zwar mit genau den *richtigen* Phasen? Die Antwort macht aus diesem Kapitel das Gegenstück zu Kapitel 23: Die Yagi ist **der Zündplan ohne Kabel**. Und weil ihre Erfindung so gut dokumentiert ist wie ihre Fallstricke (Kapitel 11 hat die Geschichte unserer eigenen Irrwege erzählt), eignet sie sich wie kein zweites Gerät für eine komplette Fallstudie: vom Wirkprinzip über den rechnerischen Entwurf und seine Vertrauensgrenzen bis zur Stückliste mit Millimetermaßen, die du wirklich bauen kannst.

Lernziele

Nach diesem Kapitel kannst du …

1. … das **Endfire-Rezept** aus der Kapitel-23-Werkzeugkiste herleiten (zwei Quellen, $d = \lambda/4$, $\Delta\Phi = 90°$ → Kardioide mit exakter Rückwärts-Null),
2. … erklären, woher ein *passives* Element seine Phase nimmt: Es ist ein getriebener Resonator — länger als die Resonanz heißt induktiv (Reflektor), kürzer heißt kapazitiv (Direktor) — und du kannst den Umschlag in NEC nachmessen,
3. … eine Mehrelement-Yagi **schrittweise entwerfen** (Aufbau- Reihe, Direktorenkette nachstimmen) und die Faustregeln begründen,

4. … die **Vertrauensgrenzen** der Rechnung benennen (Segmentierung mit Gültigkeitsfenster, robuste vs. empfindliche Größen, Querrechnung) und die Konsequenz für den Bau ziehen (Abgleich einplanen),
5. … Bandbreite und SWR der Yagi messen, ihr abruptes Kippen oberhalb der Entwurfsfrequenz erklären und die Anpass-Optionen aus Kapitel 22 zuordnen.

```
import numpy as np
import matplotlib.pyplot as plt
```

24.1 Endfire: die Keule längs der Reihe

Kapitel 23 hat die Keule quer zur Strahlerreihe geschwenkt („Breitseite"). Eine Yagi strahlt aber **längs** ihres Booms — in der Arraysprache: *Endfire*. Auch dafür liefert „Umweg = Vorsprung" das Rezept, und es ist so hübsch, dass es eine Handrechnung verdient.

Nimm zwei Strahler im Abstand $d = \lambda/4$ hintereinander, und lass den **vorderen um** $\Delta\Phi = 90°$ **später** zünden. Nach vorn ist der vordere Strahler um $\lambda/4$ voraus — sein Zeitrückstand von einer Viertelperiode gleicht das exakt aus: volle Verstärkung. Nach hinten addieren sich dagegen Wegrückstand ($\lambda/4$) und Zeitrückstand (nochmal $\lambda/4$) zu einer **halben Wellenlänge**: exakte Auslöschung. Eine Richtung bekommt alles, die Gegenrichtung nichts:

```
# von oben: nichts — zwei Zeiger genügen (Kap.-23-Rechenwerk)
phi = np.linspace(0, 2 * np.pi, 3600, endpoint=False)
kd = 2 * np.pi * 0.25                  # Abstand λ/4
# Element 2 sitzt vorn und zündet 90° später:
P = np.abs(1 + np.exp(1j * (kd * np.cos(phi) - np.pi / 2))) ** 2
print(f"vorn (0°):    {P[0]:.3f}   (Maximum 4 = doppelte Amplitude)")
print(f"hinten (180°): {P[1800]:.2e}  — exakte Null")
```

```
vorn (0°):    4.000   (Maximum 4 = doppelte Amplitude)
hinten (180°): 1.50e-32  — exakte Null
```

```
# von oben: phi, P
fig = plt.figure(figsize=(5.2, 4.8))
ax = fig.add_subplot(projection="polar")
ax.plot(phi, P / P.max(), "b-", lw=2)
ax.set_theta_zero_location("E")
ax.set_title("Endfire-Paar: die Kardioide\n(0° = vorn, Strahlrichtung)",
```

```
              fontsize=10)
plt.show()
```

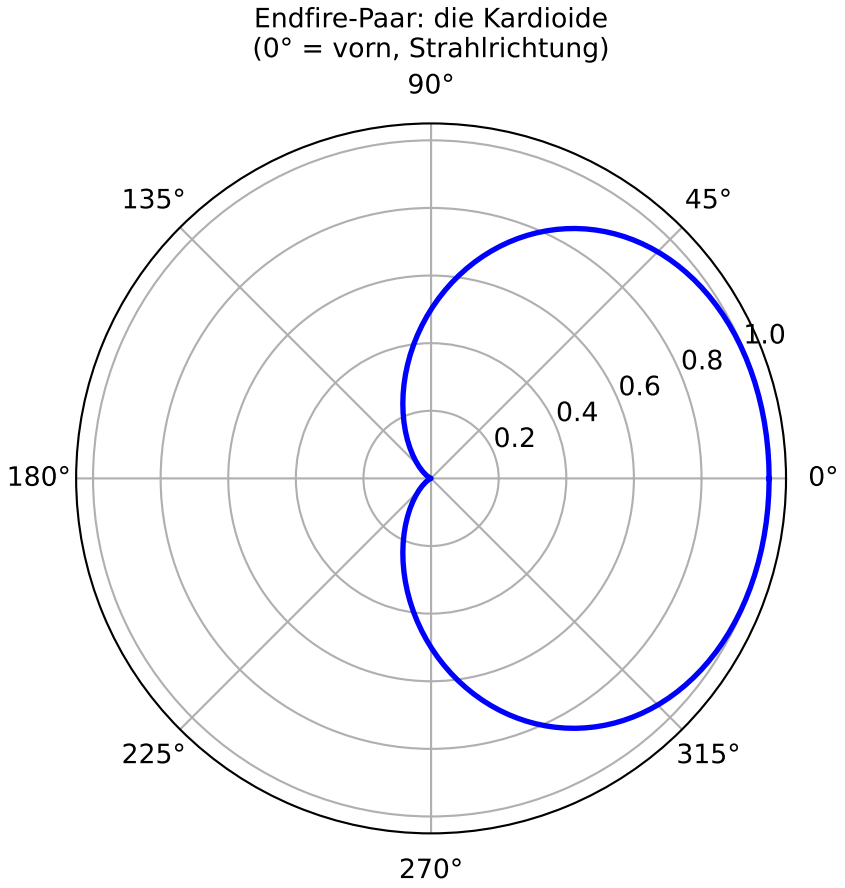


Abbildung 24.1: Das Richtdiagramm des Endfire-Paars (zwei Strahler, d = λ/4, der vordere zündet 90° später): eine Kardioide — volle Verstärkung nach vorn (0°), exakte Auslöschung nach hinten (180°), und dazwischen ein weicher Übergang ohne Nebenkeulen. Zwei Quellen und ein Phasenknopf genügen für perfekte Rückwärts-Unterdrückung.

Genau dieses Rezept steckt im Herzen jeder Yagi. Aber als Phased Array gebaut bräuchte es zwei Speiseleitungen, einen Leistungsteiler und einen 90°-Phasenschieber — bei 2,4 GHz alles Präzisionsarbeit (Kapitel 22 lässt grüßen). Die geniale Abkürzung von Shintaro Uda und Hidetsugu Yagi (1926): **Die zweite Quelle gibt es gratis.** Man stellt einfach einen unbeschalteten Metallstab in das Feld des gespeisten Dipols.

Naheliegende Vermutung: „Ein passives Element steht nur im Weg“

Naheliegende Vermutung: Ein Metallstab ohne Anschluss kann nicht senden — höchstens abschatten. Die passiven Elemente einer Yagi wirken wie Zaunlatten: Sie blockieren die Strahlung nach hinten und lassen sie nach vorn durch.
Warum sie naheliegt: Im Alltag *ist* Metall ein Hindernis — ein Blech vor einer Lampe wirft Schatten. Und „senden” verbinden wir mit einem Anschluss: kein Kabel, kein Signal.
Was stattdessen stimmt: Ein Metallstab im Wechselfeld **muss** senden. Das Feld des gespeisten Dipols treibt die Leitungselektronen im Stab auf und ab — und beschleunigte Ladungen strahlen, das ist das Knick-Argument aus Kapitel 21. Jedes passive Element ist ein vollwertiger Zweitsender; nur seinen Zündplan stellt kein Kabel ein, sondern die **Physik des getriebenen Resonators**: Die Verstimmung des Stabes legt die Phase seines Stromes fest. Mit Abschattung hat das nichts zu tun — ein einzelner 2-mm-Stab ist für eine 122-mm-Welle fast unsichtbar. Erst die *Interferenz* von gespeistem und wiederausgesandtem Feld formt die Keule.

24.2 Woher die Phase kommt: der getriebene Resonator

Der Stab ist ein Resonator — das wissen wir seit Kapitel 21: Ein Dipol der Länge $\approx \lambda/2$ trägt eine Strom-Stehwelle, wie ein Schwingkreis aus verteiltem L und C (Kapitel 19). Das Feld des gespeisten Elements wirkt auf ihn wie der Antrieb auf einen Oszillator. Und für getriebene Oszillatoren kennen wir die Regel aus Kapitel 15 und 19: **Die Phase der Antwort hängt an der Verstimmung.**

- Ist der Stab **länger** als seine Resonanzlänge, reagiert er **induktiv** — sein Strom eilt der antreibenden Spannung *nach*.
- Ist er **kürzer**, reagiert er **kapazitiv** — der Strom eilt *vor*.
- Genau auf Resonanz ist die Antwort am stärksten und in Phase.

Ein paar Prozent Längenunterschied verschieben die Stromphase um Jahrzehnte von Grad — das ist der eingebaute Phasenknopf. Zusammen mit der Laufzeit über den Elementabstand entsteht so (in grober Näherung) das Endfire-Paar von oben: Der Reflektor hinter dem Dipol ist *zu lang* geraten, der Direktor davor *zu kurz* — beide Verstimmungen drehen die Phase jeweils in die Richtung, die die Abstrahlung nach vorn sortiert.

Wie groß die Effekte wirklich sind, rechnet kein Daumenargument — dafür holen wir NEC aus Kapitel 11 zurück, das Werkzeug für Drahtantennen. Zuerst die Resonanzlänge selbst. Der Versuchsaufbau: **Frage:** Bei welcher Länge ist ein einzelner Drahtdipol (Radius 1 mm, frei im Raum, 2450 MHz) resonant — und wie reagiert er daneben? **Messgröße:** die Eingangsimpedanz $Z = R + \mathrm{j}X$ für Längen von $0{,}42$ bis $0{,}52\,\lambda$; die Resonanz ist der Nulldurchgang des Blindanteils X. **Erfolgskriterium:** ein sauberer Vorzeichenwechsel von kapazitiv ($X < 0$) zu induktiv ($X > 0$), deutlich unterhalb der idealen $0{,}5\,\lambda$ (der Endeffekt — die Dickdraht-Lektion aus Kapitel 21 in Zahlen).

```
# von oben: nichts Neues — NEC-Bausteine wie in Kap. 11
import necpp

C_LICHT = 299792458.0
F_MHZ = 2450.0
LAM = C_LICHT / (F_MHZ * 1e6)          # ≈ 0,122 m
R_DRAHT = 0.001                        # 1 mm Drahtradius
SEG = 11                               # Segmente pro Element


def dipol_z(l_lambda, seg=SEG):
    """Eingangsimpedanz eines mittengespeisten Einzeldipols."""
    ctx = necpp.nec_create()
    halb = l_lambda * LAM / 2
    necpp.nec_wire(ctx, 1, seg, 0, -halb, 0, 0, halb, 0,
                   R_DRAHT, 1.0, 1.0)
    necpp.nec_geometry_complete(ctx, 0)
    necpp.nec_fr_card(ctx, 0, 1, F_MHZ, 0)
    necpp.nec_ex_card(ctx, 0, 1, (seg + 1) // 2, 0, 1.0, 0, 0, 0, 0, 0)
    # ein Mini-Richtdiagramm (3 Punkte) stößt die Rechnung an
    necpp.nec_rp_card(ctx, 0, 1, 3, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                      180.0, 0.0, 0.0)
    z = complex(necpp.nec_impedance_real(ctx, 0),
                necpp.nec_impedance_imag(ctx, 0))
    necpp.nec_delete(ctx)
    return z


laengen = np.arange(0.42, 0.521, 0.01)
z_reihe = np.array([dipol_z(L) for L in laengen])

# Nulldurchgang von X linear interpolieren
X = z_reihe.imag
```

```
i = np.flatnonzero(np.diff(np.sign(X)))[0]
L_res = laengen[i] - X[i] * 0.01 / (X[i + 1] - X[i])
print(f"Resonanz: X = 0 bei L = {L_res:.4f} λ "
      f"(ideal dünner Draht: 0,5 λ)")
```

```
Resonanz: X = 0 bei L = 0.4522 λ (ideal dünner Draht: 0,5 λ)
```

```
# von oben: laengen, z_reihe, L_res
fig, (a1, a2) = plt.subplots(1, 2, figsize=(7.4, 3.6))
a1.plot(laengen, z_reihe.imag, "bo-")
a1.axhline(0, color="0.6", lw=1)
a1.axvline(L_res, color="tab:red", ls="--", lw=1.2,
           label=f"Resonanz {L_res:.3f} λ")
a1.set_xlabel("Dipollänge (λ)")
a1.set_ylabel("Blindanteil X (Ω)")
a1.set_title("kapazitiv ↔ induktiv", fontsize=10)
a1.legend(fontsize=8)

phase = np.degrees(np.arctan2(-z_reihe.imag, z_reihe.real))
a2.plot(laengen, phase, "go-")
a2.axhline(0, color="0.6", lw=1)
a2.axvline(L_res, color="tab:red", ls="--", lw=1.2)
a2.set_xlabel("Dipollänge (λ)")
a2.set_ylabel("Stromphase (°)")
a2.set_title("der eingebaute Phasenknopf", fontsize=10)
fig.tight_layout()
plt.show()
```

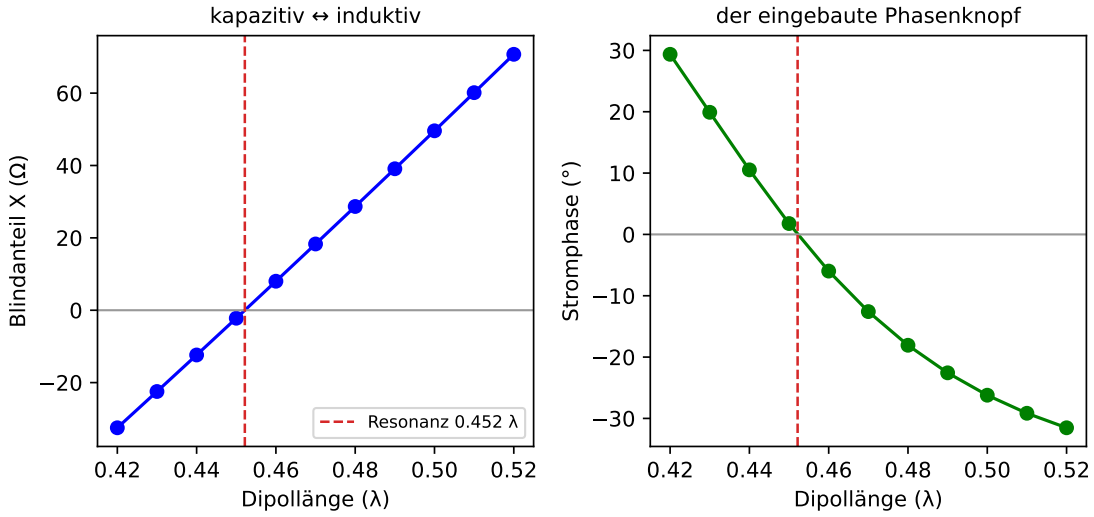


Abbildung 24.2: Der Einzeldipol als Bauteil (NEC, Radius 1 mm, 2450 MHz): Links der Blindanteil X seiner Eingangsimpedanz über der Länge — unterhalb von 0,452 λ kapazitiv (X < 0), darüber induktiv, mit fast perfekt geradem Durchgang. Rechts die daraus folgende Phasenverschiebung des Stroms, arctan(−X/R): Wenige Hundertstel λ Längenänderung drehen die Phase um Jahrzehnte von Grad. Das ist der Phasenknopf, den die Yagi über die Sägelänge ihrer Stäbe bedient.

Zwei Befunde nimmt der Entwurf gleich mit. Erstens liegt die Resonanz bei $0{,}452\,\lambda$, fast zehn Prozent unter dem Ideal $\lambda/2$ — der **Endeffekt**: An den Stabenden staut sich Ladung, das wirkt wie eine Zusatzkapazität und verkürzt die Resonanzlänge, und zwar umso stärker, je dicker der Draht (unsere 1-mm-Stäbe sind elektrisch „dick"). Wer einen Dipol nach der reinen $\lambda/2$-Formel sägt, baut ihn zu lang. Zweitens: Die Phasenkurve rechts ist steil — der Knopf ist empfindlich. Das erklärt vorab, warum Yagis auf Millimeterfehler reagieren werden.

24.3 Die Umschaltkarte: kurz führt, lang wirft zurück

Jetzt das Kernexperiment des Kapitels. Der Versuchsaufbau: **Frage:** Macht die Verstimmung den Parasiten wirklich zum Direktor oder Reflektor — und wo genau schaltet es um? **Bühne:** der gespeiste Dipol ($0{,}473\,\lambda$) bei $x = 0$; ein passiver Stab im Abstand $0{,}15\,\lambda$ **hinter** ihm (bei $-x$); „vorn" ist $+x$. **Messgrößen:** der Gewinn nach vorn und nach hinten, während die Parasitlänge von $0{,}40$ bis $0{,}56\,\lambda$ wächst (NEC kann uns die Ströme im Stab nicht

direkt verraten — das Fernfeld ist die ehrliche Messgröße). **Erfolgskriterium:** ein Vorzeichenwechsel der Differenz „vorn minus hinten" in der Nähe der eben gemessenen Resonanzlänge.

Vorhersage-Punkt (PRIMM): Der Stab steht *hinter* dem Dipol. Wenn er kurz (kapazitiv) ist — zeigt die Keule dann nach vorn oder nach hinten? Beachte: „Direktor" heißt, die Keule zeigt *zum* Stab hin.

```
# von oben: dipol_z()-Bausteine, LAM, R_DRAHT, SEG, F_MHZ, necpp


def yagi(design, f_mhz=F_MHZ, seg=SEG, npts=73, driven_tag=2):
    """Drahtreihe (Position in λ, Länge in λ) → (Azimut-dBi, Z).

    Element Nummer driven_tag wird in der Mitte gespeist, alle
    anderen sind blanke Stäbe. Azimut: 0° = +x (vorn).
    """
    ctx = necpp.nec_create()
    for i, (xpos, laenge) in enumerate(design):
        x, halb = xpos * LAM, laenge * LAM / 2
        necpp.nec_wire(ctx, i + 1, seg, x, -halb, 0, x, halb, 0,
                       R_DRAHT, 1.0, 1.0)
    necpp.nec_geometry_complete(ctx, 0)
    necpp.nec_fr_card(ctx, 0, 1, float(f_mhz), 0)
    necpp.nec_ex_card(ctx, 0, driven_tag, (seg + 1) // 2, 0,
                      1.0, 0, 0, 0, 0, 0)
    necpp.nec_rp_card(ctx, 0, 1, npts, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                      360.0 / (npts - 1), 0.0, 0.0)
    az = np.array([necpp.nec_gain(ctx, 0, 0, j) for j in range(npts)])
    z = complex(necpp.nec_impedance_real(ctx, 0),
                necpp.nec_impedance_imag(ctx, 0))
    necpp.nec_delete(ctx)
    return az, z


ABSTAND = 0.15
l_parasit = np.arange(0.40, 0.561, 0.01)
vorn_minus_hinten = []
for L in l_parasit:
    az, _ = yagi([(0.0, 0.473), (-ABSTAND, L)], npts=3, driven_tag=1)
    vorn_minus_hinten.append(az[0] - az[1])    # 0° minus 180°
vorn_minus_hinten = np.array(vorn_minus_hinten)

i = np.flatnonzero(np.diff(np.sign(vorn_minus_hinten)))[0]
```

```
L_um = l_parasit[i] - vorn_minus_hinten[i] * 0.01 \
    / (vorn_minus_hinten[i + 1] - vorn_minus_hinten[i])
print(f"Umschlag bei L = {L_um:.3f} λ "
      f"(Einzelresonanz: {L_res:.3f} λ)")
```

```
Umschlag bei L = 0.437 λ (Einzelresonanz: 0.452 λ)
```

```
# von oben: l_parasit, vorn_minus_hinten, L_res, L_um
fig, ax = plt.subplots(figsize=(7.0, 4.2))
ax.plot(l_parasit, vorn_minus_hinten, "bo-", lw=1.4)
ax.axhline(0, color="0.6", lw=1)
ax.axvline(L_res, color="tab:red", ls="--", lw=1.2,
           label=f"Einzelresonanz {L_res:.3f} λ")
ax.annotate("kurz = kapazitiv:\nKeule ZUM Stab (Direktor)",
            xy=(0.405, -6), fontsize=9, color="tab:blue")
ax.annotate("lang = induktiv:\nKeule WEG vom Stab (Reflektor)",
            xy=(0.49, 3), fontsize=9, color="tab:blue")
ax.set_xlabel("Parasitlänge (λ)")
ax.set_ylabel("Gewinn vorn − hinten (dB)")
ax.set_title("Ein Stab, zwei Berufe — die Sägelänge entscheidet")
ax.legend(loc="lower right", fontsize=9)
plt.show()
```

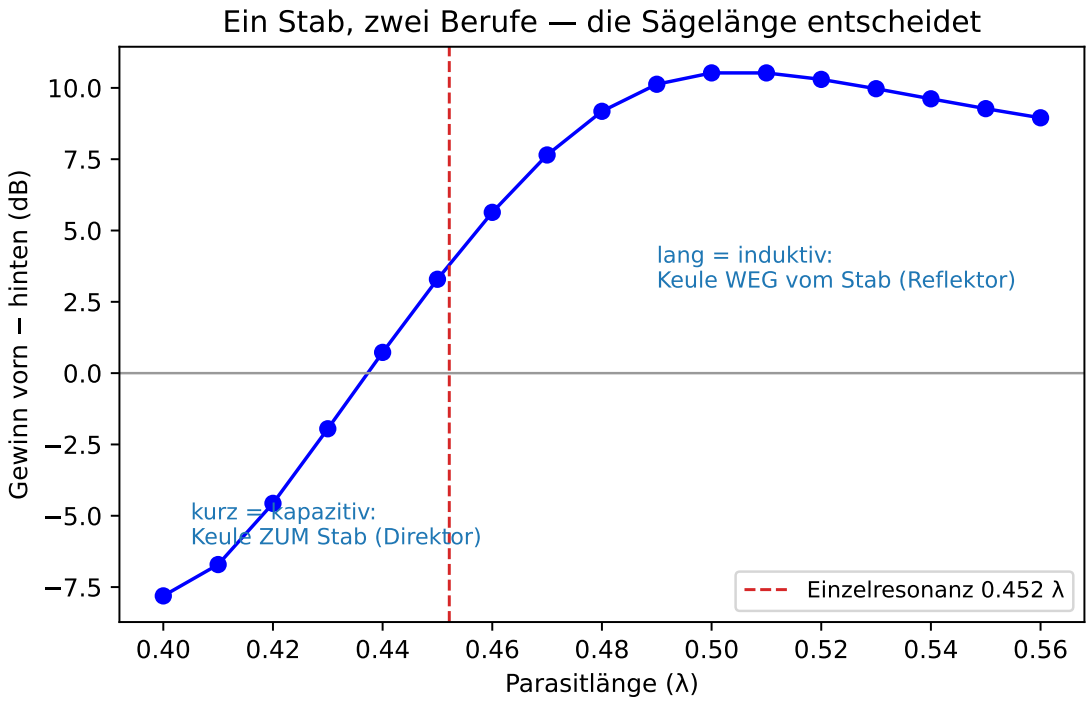


Abbildung 24.3: Die Umschaltkarte des passiven Elements (NEC: gespeister Dipol plus ein Parasit 0,15 λ dahinter): aufgetragen ist der Gewinn nach vorn minus nach hinten über der Parasitlänge. Kurze Stäbe (kapazitiv) ziehen die Keule zu sich — der Stab steht hinten, also zeigt sie nach hinten (negative Werte): er wirkt als Direktor. Lange Stäbe (induktiv) werfen sie von sich weg: Reflektor, mit gut +10 dB Vor/Rück im Maximum. Der Umschlag (0,437 λ) liegt nahe der Einzelresonanz (rot gestrichelt, 0,452 λ); den kleinen Versatz verursacht die Verkopplung der beiden Stäbe, die die wirksame Resonanz verschiebt.

Die Antwort auf den Vorhersage-Punkt: Der kurze Stab hinter dem Dipol zieht die Keule **nach hinten**, zu sich hin — er wirkt als Direktor; erst der lange Stab wirft sie nach vorn. Damit ist die Bauanleitung der Yagi vollständig begründet: **hinter** den Dipol gehört ein *langer* Stab (Reflektor), **vor** ihn *kurze* (Direktoren). Und der Merksatz des Experiments: *Kurz führt, lang wirft zurück.* Sind versehentlich *alle* Stäbe zu lang, wirken alle als Reflektoren — die Keule kippt nach hinten (Übung 24.1 zeigt, dass genau das auch beim Betrieb auf zu hoher Frequenz passiert).

24.4 Das Feldbild: die Welle klettert die Leiter entlang

Bevor wir entwerfen, wollen wir den Mechanismus einmal *sehen*. Dafür wechseln wir kurz zu Meep in die 2D-Welt — mit der Warnung aus der Kapitel-11-Saga im Gepäck: Das 2D-Modell (Metallstreifen statt Drähte) zeigt das Prinzip wunderbar, aber seine Resonanzlänge liegt bei $\sim 0{,}32\,\lambda$ — ein Artefakt der Flächenwelt. **Diese Maße sind nicht baubar**; gesägt wird später nach NEC.

Der Versuchsaufbau: **Frage:** Bündelt die Fünf-Streifen-Anordnung das Feld sichtbar nach vorn, und wie sieht die laufende Welle zwischen den Direktoren aus? **Bühne:** eine $14\lambda \times 8\lambda$-Zelle (1 λ PML); fünf Metallstreifen (Breite $0{,}03\,\lambda$) quer zur x-Achse: Reflektor ($0{,}37\,\lambda$ lang) bei $x = -0{,}2$, gespeister Streifen ($0{,}32\,\lambda$, mit Speisespalt von 3 Gitterzellen — die Kapitel-21-Regel) bei $x = 0$, drei Direktoren ($0{,}27\,\lambda$) davor. **Quelle:** Dauerstrich E_y im Spalt, komplexe Felder für Momentanbild und Film aus einem Lauf; die y-Spiegelsymmetrie (Phase -1 für E_y, Kapitel 21) halbiert die Rechenzeit. **Messgrößen:** das H_z-Feld (die natürliche Skalargröße dieser Polarisation) und als absoluter Gesundheitswert $\max|H_z|$. **Erfolgskriterium:** sichtbare Bündelung nach vorn; quantitativ wird gleich danach das Fernfeld geprüft.

```
# von oben: nichts Neues — Meep-Bausteine aus Kap. 11/21/23
import meep as mp
mp.verbosity(0)

W_2D, GAP_2D, RES_2D = 0.03, 0.075, 40
# (x-Position, Länge, gespeist?) — 2D-Lern-Maße, NICHT baubar!
ELEMENTE_2D = [(-0.20, 0.37, False), (0.00, 0.32, True),
               (0.28, 0.27, False), (0.56, 0.27, False),
               (0.84, 0.27, False)]


def geometrie_2d():
    g = []
    for x, laenge, gespeist in ELEMENTE_2D:
        if gespeist:
            arm = laenge / 2 - GAP_2D / 2
            yc = GAP_2D / 2 + arm / 2
            for vorz in (+1, -1):
                g.append(mp.Block(size=mp.Vector3(W_2D, arm, mp.inf),
                                  center=mp.Vector3(x, vorz * yc),
                                  material=mp.metal))
```

```
        else:
            g.append(mp.Block(size=mp.Vector3(W_2D, laenge, mp.inf),
                              center=mp.Vector3(x, 0),
                              material=mp.metal))
    return g


sim2d = mp.Simulation(
    cell_size=mp.Vector3(14, 8), boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.ContinuousSource(frequency=1.0, width=1.0),
                       component=mp.Ey, center=mp.Vector3())],
    resolution=RES_2D, geometry=geometrie_2d(),
    symmetries=[mp.Mirror(mp.Y, phase=-1)],
    force_complex_fields=True)
sim2d.run(until=30)
hz = sim2d.get_array(center=mp.Vector3(), size=mp.Vector3(14, 8),
                     component=mp.Hz)
```

```
# von oben: hz
print(f"Gesundheitscheck: max|Hz| = {np.abs(hz).max():.2f}")
```

```
Gesundheitscheck: max|Hz| = 12.56
```

```
# von oben: hz, ELEMENTE_2D
fig, ax = plt.subplots(figsize=(7.4, 4.3))
re = np.real(hz).T
grenze = np.abs(re).max() * 0.35
ax.imshow(re, origin="lower", extent=(-7, 7, -4, 4), cmap="RdBu_r",
          vmin=-grenze, vmax=grenze)
for x, laenge, _ in ELEMENTE_2D:
    ax.plot([x, x], [-laenge / 2, laenge / 2], "k-", lw=2.5)
ax.annotate("Keule →", xy=(5.6, 3.2), fontsize=11)
ax.set_xlabel("x in λ (Reflektor links, Direktoren rechts)")
ax.set_ylabel("y in λ")
ax.set_title("Fünf Streifen, ein Kabel: das Feld wird nach vorn "
             "sortiert")
plt.show()
```

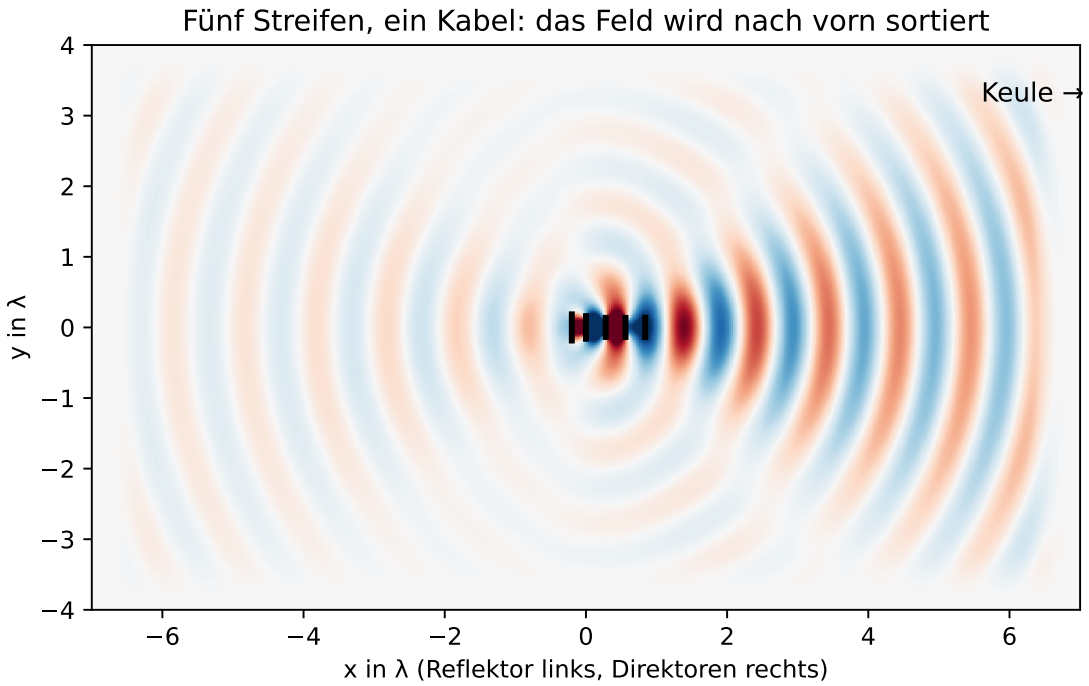


Abbildung 24.4: Das Momentanfeld Re(Hz) der 2D-Yagi im eingeschwungenen Zustand (schwarze Striche: die fünf Elemente; der gespeiste Streifen ist der zweite von links). Nach vorn (rechts) laufen kräftige, fast ebene Wellenfronten, nach hinten bleibt nur ein schwacher Rest — fünf Metallstreifen formen aus der Rundum-Abstrahlung des Dipols eine Keule. Zwischen den Direktoren ist das Feld am stärksten: Die Kette führt die Welle wie eine Leiter.

Quantitativ bestätigt das Fernfeld die Bündelung — ein Puls-Lauf mit der near2far-Box (Kapitel 14/23), Erfolgsmaß ist das Vor/Rück-Verhältnis:

```
# von oben: geometrie_2d(), RES_2D
sim2d = mp.Simulation(
    cell_size=mp.Vector3(14, 8), boundary_layers=[mp.PML(1.0)],
    sources=[mp.Source(mp.GaussianSource(frequency=1.0, fwidth=0.3),
                       component=mp.Ey, center=mp.Vector3())],
    resolution=RES_2D, geometry=geometrie_2d(),
    symmetries=[mp.Mirror(mp.Y, phase=-1)])
b = 1.3
n2f_2d = sim2d.add_near2far(
    1.0, 0, 1,
    mp.Near2FarRegion(mp.Vector3(+b, 0), size=mp.Vector3(0, 2 * b)),
```

```
    mp.Near2FarRegion(mp.Vector3(-b, 0), size=mp.Vector3(0, 2 * b),
                      weight=-1),
    mp.Near2FarRegion(mp.Vector3(0, +b), size=mp.Vector3(2 * b, 0)),
    mp.Near2FarRegion(mp.Vector3(0, -b), size=mp.Vector3(2 * b, 0),
                      weight=-1))
sim2d.run(until_after_sources=mp.stop_when_fields_decayed(
    5, mp.Ey, mp.Vector3(), 1e-4))


def S_fern(dx, dy):
    Ex, Ey, Ez, Hx, Hy, Hz = sim2d.get_farfield(
        n2f_2d, mp.Vector3(1000.0 * dx, 1000.0 * dy))
    Sx = 0.5 * np.real(Ey * np.conj(Hz) - Ez * np.conj(Hy))
    Sy = 0.5 * np.real(Ez * np.conj(Hx) - Ex * np.conj(Hz))
    return np.hypot(Sx, Sy)


fb_2d = 10 * np.log10(S_fern(1, 0) / S_fern(-1, 0))
print(f"2D-Yagi: Vor/Rück-Verhältnis = {fb_2d:+.1f} dB")
```

```
2D-Yagi: Vor/Rück-Verhältnis = +9.8 dB
```

Knapp zehn Dezibel mit fünf Streifen und einem Kabel — das Prinzip steht. Jetzt der echte Entwurf.

24.5 Der Entwurf: Schritt für Schritt zur 5-Element-Yagi

Zurück zu NEC und den echten Drähten. Wir bauen die 2,4-GHz-Yagi des Bauplans Element für Element auf und messen nach jedem Schritt drei Größen: Gewinn nach vorn, Vor/Rück und die Eingangsimpedanz (sie entscheidet über den Kabelanschluss, Kapitel 22). Der Versuchsaufbau steckt in der schon definierten `yagi()`-Funktion; das Ziel-Design — Längen zwischen $0{,}43$ und $0{,}50\,\lambda$, Drahtradius 1 mm — stammt aus dem BAUPLAN des Repos und wurde dort gegen openEMS quergerechnet. **Erfolgskriterium der Reihe:** Am Ende stehen die Kapitel-11-Werte (knapp 9 dBi, Vor/Rück ≈ 15 dB, $Z \approx 19\ \Omega$ resonant).

```
# von oben: yagi()
BASIS = [(0.00, 0.500),     # Reflektor: LANG (induktiv)
         (0.20, 0.473),     # Driven (gespeist)
         (0.35, 0.440),     # Direktoren: KURZ (kapazitiv)
         (0.55, 0.440),
         (0.75, 0.430)]

print("Schritt           Gewinn   Vor/Rück   Impedanz")
stufen = [("nur Driven", [BASIS[1]], 1),
          ("+ Reflektor", BASIS[:2], 2),
          ("+ Direktor 1", BASIS[:3], 2),
          ("+ Direktor 2", BASIS[:4], 2),
          ("+ Direktor 3", BASIS, 2)]
gewinne = []
for name, design, tag in stufen:
    az, z = yagi(design, driven_tag=tag)
    gewinne.append(az[0])
    print(f"{name:14s}   {az[0]:5.2f} dBi  {az[0] - az[36]:+6.2f} dB"
          f"   {z.real:5.1f} {z.imag:+6.1f}j Ω")
```

```
Schritt           Gewinn   Vor/Rück   Impedanz
nur Driven        2.16 dBi   +0.00 dB    83.7  +21.4j Ω
+ Reflektor       5.97 dBi  +10.11 dB    82.9  +62.2j Ω
+ Direktor 1      9.23 dBi   +7.85 dB    10.2  +42.6j Ω
+ Direktor 2      9.76 dBi  +10.17 dB     4.3  +21.8j Ω
+ Direktor 3      8.88 dBi  +14.64 dB    19.0   +1.2j Ω
```

Diese Tabelle ist der ganze Yagi-Entwurf im Zeitraffer, und sie erzählt drei Geschichten:

- **Der Reflektor** bringt den ersten großen Sprung: +3,8 dB und zehn Dezibel Vor/Rück — das Endfire-Paar vom Kapitelanfang, in Hardware.
- **Die Direktoren** heben den Gewinn weiter (auf fast 10 dBi), aber sieh dir die Impedanzspalte an: Jedes neue Element wirkt auf das gespeiste zurück und drückt dessen Wirkwiderstand in den Keller ($83 \to 10 \to 4$ Ω!) und den Blindanteil ins Induktive. Eine Yagi ist ein *verkoppeltes System* — nichts lässt sich isoliert einstellen.
- **Direktor 3 ist das Abstimm-Element:** Er ist bewusst kürzer gewählt ($0{,}430$ statt $0{,}440\ \lambda$) und *kostet* sogar fast ein Dezibel Gewinn — aber er holt das Vor/Rück-Verhältnis auf $+14{,}6$ dB und vor allem die Impedanz auf $19 + 1{,}2$j Ω: praktisch reiner Wirkwiderstand, die Antenne ist resonant und anpassbar. Ein Yagi-Entwurf optimiert nie nur den Gewinn, sondern immer das Dreieck **Gewinn / Vor-Rück / Impedanz**.

24.5.1 Mehr Direktoren, mehr Gewinn?

Vorhersage-Punkt (PRIMM): Der Bauplan verspricht in §7 „jedes weitere Element etwa +1 dB". Wir hängen jetzt stumpf weitere $0{,}435\,\lambda$-Direktoren im $0{,}2\,\lambda$-Raster an. Steigt der Gewinn um +1 dB pro Stück?

```
# von oben: yagi(), BASIS
print("Direktoren   Boom(λ)   Gewinn")
for n_dir in range(1, 7):
    design = BASIS[:2] + [(0.35 + 0.20 * k, 0.435)
                          for k in range(n_dir)]
    az, _ = yagi(design)
    print(f"   {n_dir}          {design[-1][0]:.2f}     "
          f"{az[0]:5.2f} dBi")
```

```
Direktoren   Boom(λ)   Gewinn
   1          0.35      9.18 dBi
   2          0.55      9.88 dBi
   3          0.75      9.56 dBi
   4          0.95      8.20 dBi
   5          1.15      7.36 dBi
   6          1.35      5.80 dBi
```

Er steigt nicht — er **bricht ein**, ab dem dritten Klon geht es steil bergab. Die Erklärung liefert das Feldbild von oben: Die Direktorenkette *führt* eine Welle, die von Stab zu Stab weitergereicht wird. Das funktioniert nur, solange jeder Stab mit der passenden Phase weiterreicht — die Kette ist ein Wellenleiter, und ihre „Sprosse" müssen zum Tempo der geführten Welle passen. Stumpf geklonte Glieder geraten außer Tritt, die Phasenfehler summieren sich entlang der Kette. Wer länger bauen will, muss **nachstimmen** — und dann geht es tatsächlich weiter aufwärts:

```
# von oben: yagi(), BASIS
az, _ = yagi(BASIS[:2] + [(0.35 + 0.30 * k, 0.40) for k in range(6)])
print(f"6 Direktoren, nachgestimmt (0,40 λ kurz, 0,3-λ-Raster): "
      f"{az[0]:.2f} dBi bei Boom 1,85 λ")
```

```
6 Direktoren, nachgestimmt (0,40 λ kurz, 0,3-λ-Raster): 12.77 dBi bei Boom 1,85 λ
```

Fast 13 dBi — vier Dezibel mehr als das 5-Element-Design, für den zweieinhalbfachen Boom. Die Faustregel der Yagi-Welt lautet denn auch nicht „pro

Element", sondern: **Der Gewinn wächst mit der Boom-Länge** (grob logarithmisch), und lange Ketten brauchen kürzere, weiter auseinandergezogene Glieder. Professionelle Langyagi-Entwürfe (etwa die DL6WU-Reihen) variieren Länge *und* Abstand jedes einzelnen Direktors.

24.6 Wie genau ist diese Rechnung? Die Methodengrenze

Kapitel 10 verlangt vor dem Vertrauen den **Konvergenztest**: Diskretisierung verfeinern, prüfen, ob die Antwort stehen bleibt. Die Diskretisierung der Momentenmethode ist die Zahl der Stromsegmente pro Draht. Also verfeinern wir — von 5 bis 41 Segmenten — und tragen mit, wie lang ein Segment dabei im Verhältnis zum Drahtradius wird:

```
# von oben: yagi(), BASIS, LAM
print("SEG   Segmentlänge      Gewinn    Impedanz")
for seg in (5, 7, 9, 11, 15, 21, 41):
    az, z = yagi(BASIS, seg=seg)
    sl_mm = 0.473 * LAM * 1000 / seg
    print(f"{seg:3d}   {sl_mm:5.2f} mm = {sl_mm:4.1f} R   "
          f"{az[0]:5.2f} dBi   {z.real:5.1f} {z.imag:+6.1f}j Ω")
```

```
SEG   Segmentlänge      Gewinn    Impedanz
  5   11.58 mm = 11.6 R    9.48 dBi    56.9   +3.6j Ω
  7    8.27 mm =  8.3 R    9.21 dBi    35.1   -5.1j Ω
  9    6.43 mm =  6.4 R    9.02 dBi    24.2   -2.0j Ω
 11    5.26 mm =  5.3 R    8.88 dBi    19.0   +1.2j Ω
 15    3.86 mm =  3.9 R    8.67 dBi    14.2   +5.8j Ω
 21    2.76 mm =  2.8 R    8.41 dBi    10.7  +10.5j Ω
 41    1.41 mm =  1.4 R    7.45 dBi     5.7  +21.9j Ω
```

Das ist **keine** Konvergenz — und die Tabelle ist damit lehrreicher als jede brave. Zwei Dinge passieren hier gleichzeitig. Erstens reagieren die Größen verschieden empfindlich: Der **Gewinn** bewegt sich um ein gutes halbes Dezibel, solange die Segmentierung vernünftig ist — das Fernfeld ist eine gutmütige, über die ganze Struktur gemittelte Größe. Die **Impedanz** dagegen hängt am Strom *in einem einzigen Segment* (dem Speisesegment) und wandert um einen Faktor drei. Zweitens — und das ist der eigentliche Punkt — hat das Verfeinern hier eine **Grenze**: NEC modelliert Drähte mit der Dünnsegment-Näherung, die Segmente verlangt, die deutlich länger sind als der Drahtradius (Faustregel: Segmentlänge $\gtrsim 4$ Radien). Bei SEG = 41

ist ein Segment nur noch 1,4 Radien lang — die Rechnung verlässt das Gültigkeitsfenster ihrer eigenen Näherung, und „feiner" wird ab da *falscher*, nicht besser.

> **Konvergenztest heißt nicht: blind verfeinern**
>
> Die Kapitel-10-Checkliste gilt — aber jedes Verfahren hat ein **Gültigkeitsfenster seiner Modellannahmen**, und Verfeinern hilft nur innerhalb davon. Bei FDTD hieß die Grenze „Strukturen unterhalb der Gitterzelle" (der Speisespalt aus Kapitel 21!), bei der Momentenmethode heißt sie „Segmente nicht kürzer als ein paar Drahtradien". Wo das Fenster eng wird, ersetzt man den Konvergenztest durch zwei andere Vertrauensquellen: die **Querrechnung mit einem unabhängigen Verfahren** — für diese Yagi hat das openEMS (Volumen-FDTD mit Dünndraht-Modell, eigene Rechenumgebung; Kapitel 11) erledigt: ≈ 10 dBi, Vor/Rück ≈ 12 dB, Hauptkeule vorn — und die **Messung am gebauten Gerät**.

Für den Bauplan heißt das konkret: Den *Fernfeld-Versprechen* (Keule, ≈ 9 dBi, Vor/Rück) darf man glauben — zwei Verfahren bestätigen sie. Die *Impedanz* ist eine Größenordnungsangabe („um 20 Ω, jedenfalls weit unter 50"): Anpassung ist sicher nötig, ihr Feinabgleich gehört ans gebaute Gerät. Genau deshalb sieht der Bauplan einen verschiebbaren Halter und eine SWR-Messung vor — nicht als Misstrauen gegen die Simulation, sondern als ihr ehrliches Gegenstück.

24.7 Bandbreite: schmal — und nach oben kippt sie

Eine Antenne aus lauter Resonatoren ist zwangsläufig schmalbandig (endlich scharf = endlich lang, Kapitel 15). Wie schmal, zeigt ein Frequenz-Sweep: Wir rechnen das volle Design von 2300 bis 2600 MHz durch und tragen das SWR am 50-Ω-Kabel auf — mit der **komplexen** Reflexionsformel aus Kapitel 22, $r = (Z - Z_0)/(Z + Z_0)$, denn abseits der Resonanz hat Z einen kräftigen Blindanteil:

```
# von oben: yagi(), BASIS
frequenzen = np.arange(2300, 2601, 25)
swr, gewinn_f = [], []
for f in frequenzen:
    az, z = yagi(BASIS, f_mhz=float(f))
    r = (z - 50) / (z + 50)            # komplexes r (Kap. 22)
    swr.append((1 + abs(r)) / (1 - abs(r)))
```

```
    gewinn_f.append(az[0])
swr, gewinn_f = np.array(swr), np.array(gewinn_f)
i0 = np.argmin(np.abs(frequenzen - 2450))
print(f"SWR bei 2450 MHz: {swr[i0]:.2f}  "
      f"(die 2,63 = 50/19 aus Kapitel 22)")
print(f"bei 2500 MHz: SWR {swr[np.argmin(np.abs(frequenzen - 2500))]:.0f},"
      f" Gewinn {gewinn_f[np.argmin(np.abs(frequenzen - 2500))]:+.1f} dBi")
```

```
SWR bei 2450 MHz: 2.63  (die 2,63 = 50/19 aus Kapitel 22)
bei 2500 MHz: SWR 38, Gewinn -7.2 dBi
```

```
# von oben: frequenzen, swr, gewinn_f
fig, (a1, a2) = plt.subplots(2, 1, figsize=(7.0, 5.2), sharex=True)
# semilogy: logarithmische y-Achse — das SWR springt über Dekaden
a1.semilogy(frequenzen, swr, "b.-")
a1.axvspan(2400, 2483, color="tab:green", alpha=0.15,
           label="WLAN-Band")
a1.axhline(2.63, color="0.6", ls=":", lw=1)
a1.set_ylabel("SWR am 50-Ω-Kabel")
a1.set_title("Schmalbandig — und oberhalb der Entwurfsfrequenz "
             "kippt sie")
a1.legend(fontsize=9)
a2.plot(frequenzen, gewinn_f, "g.-")
a2.axvspan(2400, 2483, color="tab:green", alpha=0.15)
a2.axhline(0, color="0.6", lw=1)
a2.set_xlabel("Frequenz (MHz)")
a2.set_ylabel("Gewinn vorn (dBi)")
fig.tight_layout()
plt.show()
```

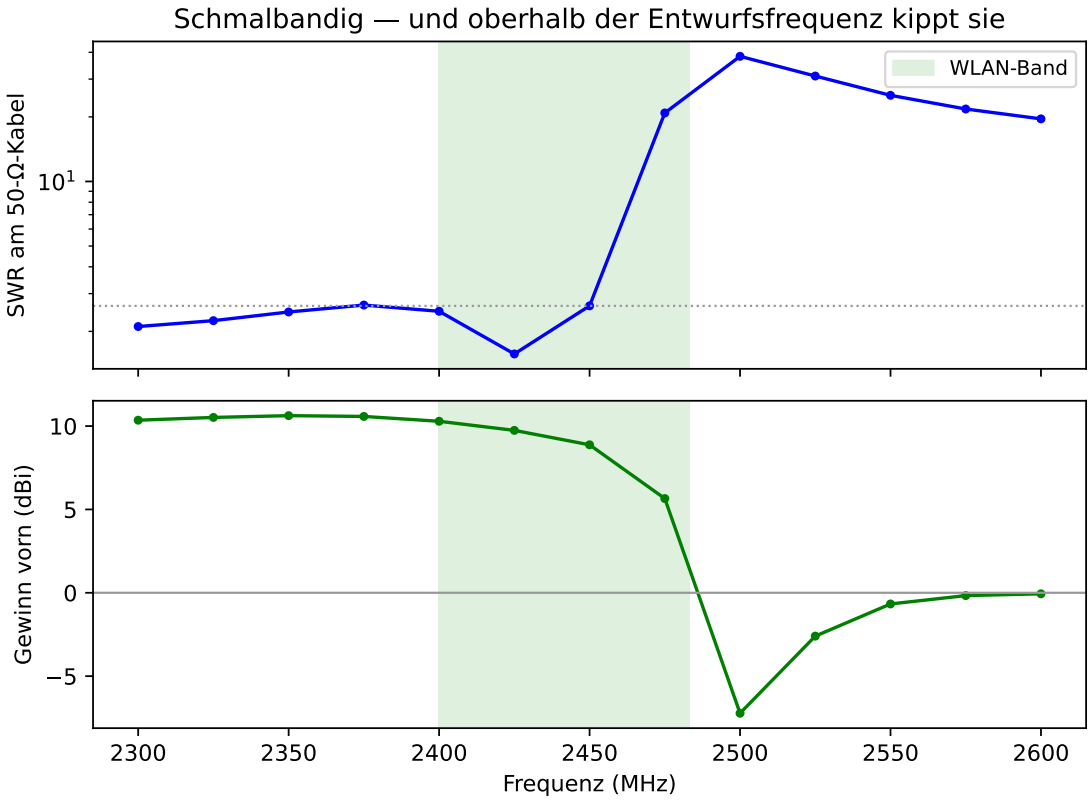


Abbildung 24.5: Der Frequenz-Sweep der 5-Element-Yagi: oben das SWR am 50-Ω-Kabel (logarithmisch; grün hinterlegt das 2,4-GHz-WLAN-Band), unten der Gewinn nach vorn. Unterhalb der Entwurfsfrequenz bleibt die Antenne gutmütig — oberhalb kippt sie abrupt: Bei 2500 MHz ist das SWR auf 38 explodiert und der „Gewinn“ nach vorn negativ, die Keule zeigt nach hinten. Der Grund steht in der Umschaltkarte: Steigt die Frequenz, wächst die elektrische Länge aller Stäbe — die Direktoren rutschen über die Resonanz und werden zu Reflektoren.

Die Asymmetrie ist kein Zufall, sondern die Umschaltkarte in Aktion: Ob ein Stab Direktor oder Reflektor ist, entscheidet seine Länge **in Wellenlängen**. Höhere Frequenz = kürzeres λ = alle Stäbe elektrisch länger; schon zwei Prozent reichen, und die Direktoren rutschen über die Resonanz auf die Reflektor-Seite — die Yagi schießt nach hinten. Tiefere Frequenzen schieben alle Stäbe tiefer in den Direktor-/Kapazitiv-Bereich, das verschmerzt sie deutlich länger. Merke für den Bau: **Im Zweifel lieber etwas zu kurz sägen als zu lang** — und nach dem gleichen Argument kippt eine zu lang geratene Yagi komplett (Übung 24.1).

24.7.1 Anpassung: die 19 Ω ans 50-Ω-Kabel

Bleibt der Anschluss. 19 Ω an 50 Ω heißt SWR 2,6 — es funkt (80 % der Leistung kommen an, Kapitel 22), aber gut ist es nicht. Der Bauplan nennt drei Praxislösungen, die sich alle mit Kapitel-22-Vokabular lesen lassen:

- **Hairpin-Anpassung:** Das gespeiste Element wird 1–2 mm *gekürzt* (wird kapazitiv — die Umschaltkarte als Stellschraube!), dann kompensiert eine kleine Drahtschleife parallel zum Speisepunkt als Induktivität: zusammen ein L-Glied, das 19 Ω auf 50 Ω hebt.
- **Gamma-Anpassung:** Das Element bleibt ein durchgehender Stab (kein Speisespalt!); ein kurzer Parallelstab mit Serienkondensator greift die Spannung ein Stück neben der Mitte ab — dort, wo die Strom-Stehwelle (Kapitel 21) den passenden Impedanzpegel bietet. Bonus: braucht keinen Balun.
- **Faltdipol:** Das gespeiste Element als geschlossene Schleife vervierfacht die Impedanz (19 → 76 Ω) — passend fürs 75-Ω-Fernsehkabel aus Übung 22.1.

Dazu gehört die **Symmetrierung**: Der Dipol ist symmetrisch, das Koaxkabel nicht — ohne Gegenmaßnahme fließt ein Strom außen auf dem Kabelschirm mit, und der Mantel strahlt (Kapitel 22, Kleingedrucktes). Bei 2,4 GHz hilft eine **λ/4-Mantelhülse**: ein etwa 31 mm langes Metallröhrchen über dem Kabel, am antennenfernen Ende mit dem Schirm verlötet. Das ist wörtlich ein Kapitel-22-Trick — eine am Ende kurzgeschlossene $\lambda/4$-Leitung (zwischen Hülse und Schirm) transformiert ihren Kurzschluss in einen Leerlauf am offenen Ende und sperrt so den Mantelstrom.

24.8 Der Bauplan

Damit ist alles begründet, was im Bauplan steht. Die Stückliste kommt direkt aus dem Design — Millimeter, zum Sägen:

```
# von oben: BASIS, LAM
namen = ["Reflektor", "Driven", "Direktor 1", "Direktor 2",
         "Direktor 3"]
print(f"f = 2450 MHz, λ = {LAM * 1000:.2f} mm — Kupferstab Ø 2 mm")
print("Element       Länge      Boom-Position")
for name, (xpos, laenge) in zip(namen, BASIS):
    print(f"{name:11s}  {laenge * LAM * 1000:5.1f} mm    "
          f"{xpos * LAM * 1000:5.1f} mm")
```

```
f = 2450 MHz, λ = 122.36 mm — Kupferstab Ø 2 mm
Element       Länge      Boom-Position
Reflektor     61.2 mm      0.0 mm
Driven        57.9 mm     24.5 mm
Direktor 1    53.8 mm     42.8 mm
Direktor 2    53.8 mm     67.3 mm
Direktor 3    52.6 mm     91.8 mm
```

```python
# von oben: BASIS, LAM, namen
fig, ax = plt.subplots(figsize=(7.2, 3.4))
for name, (xpos, laenge) in zip(namen, BASIS):
    x_mm = xpos * LAM * 1000
    l_mm = laenge * LAM * 1000
    farbe = "tab:blue" if name == "Driven" else "k"
    ax.plot([x_mm, x_mm], [-l_mm / 2, l_mm / 2], color=farbe, lw=3.5)
    ax.annotate(f"{l_mm:.1f}", xy=(x_mm, l_mm / 2 + 2), ha="center",
                fontsize=8)
    ax.annotate(f"{x_mm:.1f}", xy=(x_mm, -l_mm / 2 - 5), ha="center",
                fontsize=8, color="0.4")
ax.plot([-6, 98], [0, 0], color="0.7", lw=5, zorder=0)   # der Boom
ax.annotate("Strahlrichtung →", xy=(70, 25), fontsize=11)
ax.annotate("Speisespalt + Kabel", xy=(24.5, -38),
            ha="center", fontsize=8, color="tab:blue")
ax.set_xlim(-15, 110)
ax.set_ylim(-45, 42)
ax.set_aspect("equal")
ax.set_xlabel("Boom-Position (mm); Zahlen über den Stäben: Länge (mm)")
ax.set_yticks([])
ax.set_title("Bauskizze: fünf Stäbe, ein Kabel, 8,9 dBi")
plt.show()
```

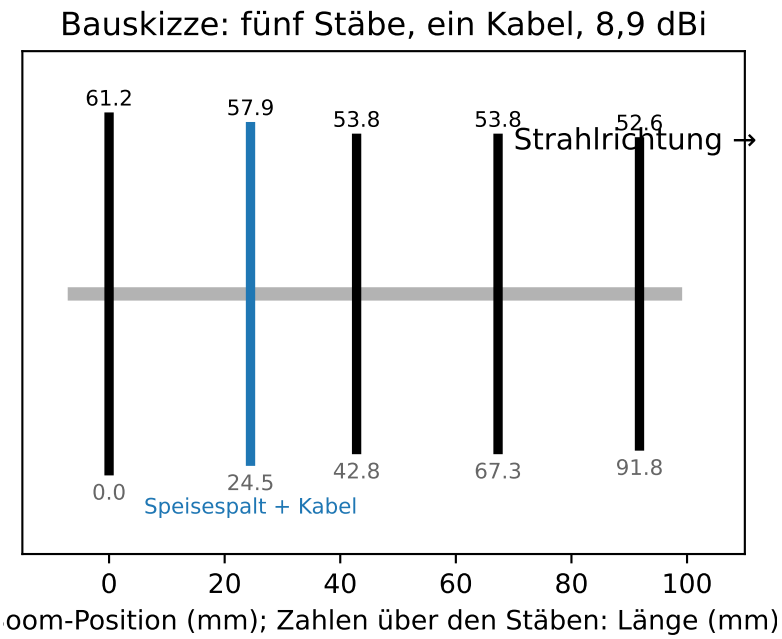


Abbildung 24.6: Die Bauskizze der 2,4-GHz-Yagi in Originalproportionen (Maße in Millimetern): fünf Kupferstäbe Ø 2 mm auf einem rund 10 cm langen Boom, nur das zweite Element (Driven, blau) bekommt einen Speisespalt und das Kabel. Der Boom selbst ist unkritisch — ein 3D-gedruckter Kunststoffsteg ist für 2,4 GHz praktisch unsichtbar (ε_r ≈ 2,7, dünn gegen λ) und verstimmt anders als ein Metallboom nichts.

Zum fertigen Gerät fehlen drei Handgriffe, alle aus dem Bauplan des Repos (`BAUPLAN_yagi_2.4GHz.md`):

1. **Boom drucken:** ein flacher PLA/PETG-Steg mit Ø-2-mm-Bohrungen an den fünf Positionen; für das gespeiste Element zwei Bohrungen mit 2–3 mm Mittelspalt. Profi-Tipp aus der Methodengrenzen-Diskussion: einen kleinen verschiebbaren Halter (±2 mm) vorsehen — der Feinabgleich ist eingeplant, nicht eingestanden.
2. **Abgleichen:** Resonanz und SWR mit einem NanoVNA messen (für das 2,4-GHz-Band gibt es sie günstig); erst die Driven-Länge auf Resonanz trimmen, dann die Anpassung auf minimales SWR.
3. **Richtwirkung testen:** die Signalstärke (RSSI) eines WLAN-Zugangspunkts in Strahlrichtung gegen die Rückrichtung vergleichen — der Unterschied sollte um die 10–15 dB liegen, das gemessene Vor/Rück-Verhältnis.

Und damit ist die Fallstudie geschlossen, deren Anfang Kapitel 11 erzählt

hat: Prinzip im 2D-Feldbild, Entwurf mit dem Werkzeug, das zur Geometrie passt, Vertrauen aus Konvergenz-*Verständnis* plus unabhängiger Querrechnung — und ein Gerät, das man am Ende in die Hand nehmen kann.

24.9 Das Kapitel-Programm

`programme/kap24/kap24_fallstudie_yagi.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die exakte Rückwärts-Null der Kardioide ($< 10^{-20}$), die Einzeldipol-Resonanz ($0{,}44 < L < 0{,}47\,\lambda$), die Umschaltkarte (klare Vorzeichen an beiden Enden), die Aufbau-Reihe (Endwerte in engen Fenstern um 8,9 dBi / 14,6 dB / 19 Ω), den Einbruch der naiven Direktorenklone und die nachgestimmten $>$ 12 dBi, den Frequenz-Kipp (SWR > 20 und negativer Gewinn bei 2500 MHz) und das 2D-Vor/Rück ($+9{,}8$ dB). Läuft in wenigen Sekunden.

Merkkasten

- **Endfire-Rezept:** zwei Strahler, $d = \lambda/4$, der vordere $90°$ später → volle Verstärkung nach vorn, exakte Null nach hinten (Kardioide). Die Yagi baut dieses Paar — ohne zweites Kabel.
- **Passiv heißt nicht stumm:** Ein Metallstab im Wechselfeld sendet zwangsläufig; seine Phase stellt die **Verstimmung** (getriebener Resonator): *kurz führt, lang wirft zurück* — Direktoren kürzer und vorn, Reflektor länger und hinten.
- **Yagi-Entwurf ist Systemabstimmung:** Jedes Element verstimmt alle anderen (Impedanz!); optimiert wird das Dreieck Gewinn / Vor-Rück / Impedanz. Mehr Gewinn kommt aus mehr **Boom-Länge** mit nachgestimmter Kette, nicht aus geklonten Direktoren.
- **Methodengrenzen kennen:** Verfeinern hilft nur im Gültigkeitsfenster der Modellannahmen (MoM: Segmentlänge $\gtrsim 4$ Drahtradien). Fernfeldgrößen sind robust, die Speisepunkt-Impedanz ist empfindlich — Querrechnung und Abgleich am Gerät gehören zum Entwurf.
- **Yagis sind schmalbandig und kippen nach oben** (Direktoren werden elektrisch zu lang → Reflektoren). Beim Bau gilt: lieber knapp zu kurz als zu lang; Anpassung (Hairpin/Gamma/ Faltdipol) und λ/4-Mantelhülse sind Kapitel-22-Technik.

Roter Faden

Dieses Kapitel hat die Fäden von drei Teilen des Buchs verknotet: das Zeiger-Rechenwerk und „Umweg = Vorsprung” aus **Kapitel 14/23** (Endfire statt Breitseite), den getriebenen Resonator samt Phasengang aus **Kapitel 15/19**, das Knick-Argument („beschleunigte Ladung strahlt — auch ohne Kabel”) aus **Kapitel 21**, die Spiegelformel, das SWR und den λ/4-Trick aus **Kapitel 22** und die Validierungs-Checkliste aus **Kapitel 10** — verschärft um die Lektion, dass jedes Verfahren ein Gültigkeitsfenster hat (**Kapitel 11**: die Saga; hier: ihre Vermessung). Nach vorn: **Kapitel 25** fragt, warum die 50-Hz-Steckdose mit denselben Drahtlängen *nicht* funkt (die elektrisch kleine Antenne), und die Wellenleiter-Natur der Direktorenkette kehrt in **Kapitel 26/27** wieder, wenn periodische Strukturen Wellen führen und sperren.

Übungen

Ü 24.1 (Verstehen). Ein Bastler sägt alle fünf Stäbe versehentlich 4 % zu lang. Was passiert mit der Keule — wird sie nur etwas schwächer? Begründe mit der Umschaltkarte, und stütze die Antwort mit dem Frequenz-Sweep: Welcher Messpunkt dort entspricht „alle Stäbe 2 % zu lang”?

> ℹ Musterlösung zu Ü 24.1
>
> Zu lange Stäbe sind elektrisch in Richtung „induktiv” verschoben. Die Direktoren liegen im Entwurf nur wenige Prozent unter der Resonanz — 4 % Zugabe schiebt sie über den Umschlagpunkt: Aus Direktoren werden Reflektoren, und die Keule **kippt nach hinten**, sie wird nicht bloß schwächer. Im Frequenz-Sweep steckt dasselbe Experiment, denn es zählt nur die Länge in Wellenlängen: Betrieb bei 2500 MHz statt 2450 macht alle Stäbe elektrisch $2500/2450 - 1 \approx 2\,\%$ länger — und genau dort bricht der Sweep zusammen (SWR 38, Gewinn nach vorn negativ). Eine Yagi verzeiht „zu kurz” viel eher als „zu lang”.

Ü 24.2 (Verändern). Baue die 7-Element-Yagi: Hänge an das `BASIS`-Design zwei weitere Direktoren im $0{,}25\,\lambda$- Abstand an und suche ihre beste gemeinsame Länge zwischen $0{,}39$ und $0{,}44\,\lambda$ (in 0,01er-Schritten). Wie viel Gewinn bringt der längere Boom — und was macht das Vor/Rück- Verhältnis?

Musterlösung zu Ü 24.2

```
# von oben: yagi(), BASIS
print("L_dir    Gewinn    Vor/Rück")
for L_dir in np.arange(0.39, 0.441, 0.01):
    design = BASIS + [(0.75 + 0.25 * (k + 1), L_dir)
                      for k in range(2)]
    az, _ = yagi(design)
    print(f"{L_dir:.2f}   {az[0]:6.2f} dBi   {az[0] - az[36]:+6.2f} dB")
```

```
L_dir    Gewinn    Vor/Rück
0.39    10.10 dBi   +16.87 dB
0.40    10.09 dBi   +17.48 dB
0.41     9.99 dBi   +19.73 dB
0.42     9.61 dBi   +33.07 dB
0.43     7.98 dBi   +13.07 dB
0.44     0.83 dBi    -2.38 dB
```

Die zwei Zusatzdirektoren wollen *kürzer* sein als die Bestandskette (das Optimum liegt hier um $0{,}40$–$0{,}42\,\lambda$) und bringen dann rund ein bis anderthalb Dezibel — weniger als „+1 dB pro Element", der Zuwachs nimmt mit der Kettenlänge ab. Das Vor/Rück-Verhältnis verschiebt sich dabei je nach Länge spürbar: Wer es maximal will, optimiert wieder das ganze Dreieck, nicht eine Zahl. (Und die Impedanz hat sich natürlich auch wieder verschoben — nachprüfen!)

Ü 24.3 (Übertragen). Skaliere die Yagi auf 433 MHz (das 70-cm-Funkband): Alle Längen und Abstände wachsen um den Faktor $2450/433 \approx 5{,}66$ — aber dein Stabmaterial bleibt 2 mm dick. Maxwell ist skaleninvariant (Kapitel 11): Ist die skalierte Antenne trotzdem exakt gleichwertig? Prüfe mit `dipol_z` bei 433 MHz, ob ein auf $0{,}452\,\lambda$ geschnittener Einzeldipol dort noch resonant ist, und erkläre den Unterschied.

Musterlösung zu Ü 24.3

```
# von oben: dipol_z()-Bauweise, necpp – eigene Version für 433 MHz
F2 = 433.0
LAM2 = C_LICHT / (F2 * 1e6)


def dipol_z_433(l_lambda):
    ctx = necpp.nec_create()
    halb = l_lambda * LAM2 / 2
    necpp.nec_wire(ctx, 1, SEG, 0, -halb, 0, 0, halb, 0,
                   0.001, 1.0, 1.0)          # Radius bleibt 1 mm!
    necpp.nec_geometry_complete(ctx, 0)
    necpp.nec_fr_card(ctx, 0, 1, F2, 0)
    necpp.nec_ex_card(ctx, 0, 1, (SEG + 1) // 2, 0, 1.0, 0, 0, 0, 0, 0)
    necpp.nec_rp_card(ctx, 0, 1, 3, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                      180.0, 0.0, 0.0)
    z = complex(necpp.nec_impedance_real(ctx, 0),
                necpp.nec_impedance_imag(ctx, 0))
    necpp.nec_delete(ctx)
    return z


print(f"bei 2450 MHz: Z(0,452 λ) = {dipol_z(0.452):.1f} Ω")
print(f"bei  433 MHz: Z(0,452 λ) = {dipol_z_433(0.452):.1f} Ω")
for L in (0.46, 0.47, 0.48):
    print(f"bei  433 MHz: Z({L} λ) = {dipol_z_433(L):.1f} Ω")
```

```
bei 2450 MHz: Z(0,452 λ) = 72.4-0.2j Ω
bei  433 MHz: Z(0,452 λ) = 63.1-35.2j Ω
bei  433 MHz: Z(0.46 λ) = 66.3-21.4j Ω
bei  433 MHz: Z(0.47 λ) = 70.6-4.2j Ω
bei  433 MHz: Z(0.48 λ) = 75.2+13.1j Ω
```

Nicht exakt gleichwertig: Bei 433 MHz ist der $0{,}452\,\lambda$-Dipol noch deutlich **kapazitiv** — die Resonanz liegt jetzt höher, um $0{,}47\,\lambda$. Skaleninvarianz gilt nur, wenn **alles** mitskaliert, auch der Drahtradius; unser 2-mm-Stab ist bei 433 MHz (λ = 69 cm) elektrisch viel *dünner* als bei 2,4 GHz, und dünnere Dipole haben weniger Endeffekt-Verkürzung (näher an $0{,}5\,\lambda$). Wer skaliert, muss also nachstimmen — alle Elementlängen ein bis zwei Prozent strecken — oder gleich einen 11-mm-Stab nehmen. Genau deshalb geben seriöse Baupläne immer den Drahtdurchmesser mit an.

Ü 24.4 (Übertragen). Schließe den Kreis zu Kapitel 23: Baue das Endfire-Paar vom Kapitelanfang als *gespeistes* Phased Array in NEC — zwei resonante Dipole ($0{,}452\,\lambda$) im Abstand $\lambda/4$, der vordere mit $-90°$ Speisephase (NEC zählt die Phase in $e^{+j\omega t}$-Konvention, Kapitel 23). Miss das Vor/Rück-Verhältnis und vergleiche mit der parasitären Lösung (Driven + Reflektor aus der Aufbau-Reihe): Wer unterdrückt rückwärts besser, und warum erreicht keiner die exakte Kardioiden-Null?

Musterlösung zu Ü 24.4

```
# von oben: yagi()-Bauweise, necpp, LAM, R_DRAHT, SEG
ctx = necpp.nec_create()
for i, x in enumerate((0.0, 0.25)):
    halb = 0.452 * LAM / 2
    necpp.nec_wire(ctx, i + 1, SEG, x * LAM, -halb, 0,
                   x * LAM, halb, 0, R_DRAHT, 1.0, 1.0)
necpp.nec_geometry_complete(ctx, 0)
necpp.nec_fr_card(ctx, 0, 1, F_MHZ, 0)
necpp.nec_ex_card(ctx, 0, 1, (SEG + 1) // 2, 0, 1.0, 0.0, 0, 0, 0, 0)
necpp.nec_ex_card(ctx, 0, 2, (SEG + 1) // 2, 0,
                  np.cos(-np.pi / 2), np.sin(-np.pi / 2), 0, 0, 0, 0)
necpp.nec_rp_card(ctx, 0, 1, 3, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                  180.0, 0.0, 0.0)
g_v = necpp.nec_gain(ctx, 0, 0, 0)
g_h = necpp.nec_gain(ctx, 0, 0, 1)
necpp.nec_delete(ctx)
print(f"gespeistes Endfire-Paar: vorn {g_v:.2f} dBi, "
      f"hinten {g_h:.2f} dBi → F/B {g_v - g_h:+.1f} dB")

az, _ = yagi(BASIS[:2])
print(f"parasitäres Paar (Driven + Reflektor): "
      f"F/B {az[0] - az[36]:+.1f} dB")
```

```
gespeistes Endfire-Paar: vorn 5.64 dBi, hinten 2.25 dBi → F/B +3.4 dB
parasitäres Paar (Driven + Reflektor): F/B +10.1 dB
```

Das gespeiste Paar gewinnt deutlich (um die 20 dB Vor/Rück gegen gut 10 dB parasitär) — sein Phasenknopf ist exakt, während der parasitäre Strom in Phase *und* Stärke vom Verkopplungs-Kompromiss diktiert wird. Die ideale Kardioiden-Null erreicht aber keiner von beiden: Die Herleitung galt für zwei *Punktquellen mit gleich starken Strömen* — echte Dipole sind ausgedehnt, und schon ein kleiner Amplitudenunterschied lässt die Auslöschung unvollständig (zwei Zeiger löschen sich

nur bei exakt gleicher Länge aus). Der Yagi-Trick bleibt trotzdem der Gewinner der Praxis: null zusätzliche Speisetechnik für immer noch über 10 dB Rückwärts-Unterdrückung — und mit der Direktorenkette skaliert er, was das gespeiste Paar nicht kann.

Das Kleingedruckte

Die Verkopplung formal. Was wir über Fernfelder gemessen haben, fasst die Antennentheorie in Gegenimpedanzen: Für zwei Antennen gilt $U_1 = Z_{11}I_1 + Z_{12}I_2$ (und symmetrisch für 2). Beim Parasiten ist $U_2 = 0$ (kurzgeschlossen), also $I_2 = -Z_{21}I_1/Z_{22}$ — Betrag *und* Phase des parasitären Stroms folgen aus Eigen- (Z_{22}, die Verstimmung) und Gegenimpedanz (Z_{21}, der Abstand). Unsere Umschaltkarte ist die Fernfeld-Spur genau dieser Formel.

Moderne Entwürfe überlassen das Dreieck Gewinn/Vor-Rück/Impedanz Optimierern (genetische Algorithmen, Partikelschwärme), die NEC tausendfach aufrufen — die Fitnessfunktion ist die Kunst. Kapitel 31 nimmt den Faden auf (inverses Design).

Warum kürzer besser führt: Dass die nachgestimmte Kette ihre Direktoren *kürzer* will, hat einen ehrwürdigen Namen — die **Hansen-Woodyard-Bedingung** (1938): Eine Endfire-Reihe erreicht ihre maximale Direktivität nicht im exakten Synchrontakt mit der Lichtgeschwindigkeit, sondern mit einer gezielten Extra-Nacheilung von insgesamt etwa π über die Kettenlänge. Physikalisch: Die Direktorenkette führt eine Oberflächenwelle, die etwas *langsamer* als c läuft — und kürzere (kapazitive) Glieder stellen genau diese Verzögerung ein. Zu viel davon kippt den Effekt wieder.

Lange Yagis werden zunehmend toleranzempfindlich: Je schärfer die geführte Welle auf die Kette abgestimmt ist, desto kleiner die Bandbreite und desto härter schlagen Fertigungsfehler durch — die 12,8-dBi-Kette von oben ist spürbar zickiger als das gutmütige 5-Element-Design. Auch das Wetter redet mit: Ein Regen- oder Eisfilm verstimmt die Stäbe (Wasser hat $\varepsilon_r \approx 80$, Kapitel 1!) — Langyagi-Entwürfe rechnen Reserve ein.

Der Boom. Unsere Rechnung lässt die Stäbe frei schweben; ein Kunststoffboom ändert daran fast nichts. Ein **Metallboom** dagegen, durch den die Elemente hindurchgehen, verkürzt sie elektrisch — Praxis-Korrekturformeln (nach DL6WU) schlagen je nach Boomdurchmesser einige Prozent Länge auf. Wer Metall baut, korrigiert oder isoliert.

Und die Geschichte: Shintaro Uda publizierte die Antenne 1926 auf Japanisch, sein Professor Hidetsugu Yagi machte sie auf Englisch bekannt — weshalb die Welt bis heute meist nur den halben Namen nennt. Ihre erste große Bühne hatte die „Yagi" ausgerechnet im Zweiten Weltkrieg als Radarantenne — auf allen Seiten.

25 Die elektrisch kleine Antenne — warum 50 Hz nicht funkt

Durch deine Küche fließen kräftigere Wechselströme als durch manchen Rundfunksender: Ein Herd zieht 16 Ampere, fünfzigmal pro Sekunde hin und her. Und seit Kapitel 21 wissen wir: Beschleunigte Ladung strahlt — *jeder* Draht ist eine Antenne. Wechselstrom ist nichts als pausenlos beschleunigte Ladung. Müsste der Äther dann nicht voller 50-Hz-Sender sein, jedes Haus ein Rundfunkturm, jedes Verlängerungskabel ein Störsender für den ganzen Kontinent?

Ist er nicht — und Kapitel 19 hat versprochen, dass die Begründung am Ende **eine einzige Längenrechnung** ist. Dieses Kapitel löst das Versprechen ein und schließt damit Teil VI: Es vermisst, wie dramatisch eine Antenne einbricht, die klein gegen ihre Wellenlänge ist — und warum sich dieser Einbruch weder mit Leistung noch mit Trickschaltungen wegkaufen lässt. Unterwegs fällt Licht auf eine Technik, die das vermeintliche Versagen absichtlich ausnutzt: die Karte, mit der du morgens die Bürotür öffnest.

Lernziele

Nach diesem Kapitel kannst du …

1. … den **Strahlungswiderstand** R_s erklären ($P = \frac{1}{2}\hat{I}^2 R_s$) und wissen, warum er für kurze Dipole wie $R_s = 20\pi^2(l/\lambda)^2$ einbricht — und du hast die Formel mit NEC über eine Dekade nachgemessen,
2. … die 50-Hz-Längenrechnung führen: Wirkungsgrad $\eta = R_s/(R_s + R_V)$ eines Netzkabels ($\approx 5 \cdot 10^{-9}$) und die Doppelleitungs-Auslöschung obendrauf,
3. … erklären, warum das Nahfeld einer kleinen Antenne **pendelt statt sendet** (Blindleistung, Kapitel 21) — und warum der Brumm im Verstärker trotzdem real ist,
4. … das doppelte Gefängnis kleiner Antennen benennen: der **Blindanteil-Abgrund** (Verlängerungsspulen verheizen den Rest) und die **Chu-Grenze** (winzige Bandbreite),

5. … die Auswege einordnen: Empfangen ist nicht Senden, NFC ist absichtliches Nahfeld, und Längstwellenfunk braucht Antennen von der Größe eines Dorfes.

```
import numpy as np
import matplotlib.pyplot as plt
```

25.1 Der Strahlungswiderstand: Funk als Widerstand im Schaltplan

Kapitel 21 hat gemessen, *dass* ein Dipol strahlt, und Kapitel 22, wie man Leistung zu ihm hinbringt. Jetzt verbinden wir beides mit einer Buchhalter-Größe. Vom Speisepunkt aus gesehen tut eine Antenne dasselbe wie ein Widerstand: Sie nimmt Wirkleistung auf und gibt sie nie zurück — nur verheizt sie sie nicht, sondern schickt sie als Welle davon. Man definiert deshalb den **Strahlungswiderstand**

$$P_{\text{abgestrahlt}} \;=\; \tfrac{1}{2}\,\hat{I}^2\,R_s,$$

mit $\hat{I}$ als Stromamplitude am Speisepunkt. R_s ist keine Materialeigenschaft, sondern reine Geometrie — beim $\lambda/2$-Dipol sind es die berühmten $\approx 73\ \Omega$ (der Wirkanteil, den unsere NEC-Läufe in Kapitel 24 immer mitgeliefert haben).

Was passiert, wenn die Antenne *kurz* gegen die Wellenlänge wird? Das Knick-Argument aus Kapitel 21 gibt die Antwort her: Das Strahlungsfeld ist proportional zur beschleunigten Ladungsmenge — für einen Stromfaden also zum Produkt aus Strom und Länge, $\hat{I}\cdot l$, und zur Frequenz. In Wellenlängen-Sprache: Das Feld trägt einen Faktor l/λ, die Leistung dessen Quadrat. Für den mittengespeisten kurzen Dipol (auf dem der Strom dreieckig von der Mitte zu den Enden abfällt — die Enden sind Sackgassen, Kapitel 21) lautet das Ergebnis der Lehrbuchrechnung kompakt:

$$R_s \;=\; 20\pi^2\left(\frac{l}{\lambda}\right)^2 \qquad (l \ll \lambda).$$

Die Handrechnung dazu: Ein Dipol von einem Zehntel der Wellenlänge hat

$$R_s = 20\pi^2\cdot 0{,}1^2 = 20\cdot 9{,}87\cdot 0{,}01 \approx 1{,}97\ \Omega$$

— von 73 Ω auf zwei Ohm, obwohl die Antenne nur um den Faktor fünf geschrumpft ist. **Halbe Länge heißt ein Viertel des Strahlungswiderstands.** Heraus kommt ein Skalar in Ohm, und er ist der Wechselkurs zwischen Strom und Funkleistung.

25.1.1 Die NEC-Messreihe

Formeln prüfen wir nach — der Werkzeugkasten steht seit Kapitel 24 bereit. Der Versuchsaufbau: **Frage:** Folgt $R_s(l/\lambda)$ wirklich der $20\pi^2(l/\lambda)^2$-Parabel, und was macht der Blindanteil? **Bühne:** ein einzelner mittengespeister Dipol in NEC bei 2450 MHz, Längen von $0{,}04$ bis $0{,}4\,\lambda$. Eine Lehre aus Kapitel 24 fließt direkt ein: Bei $l/\lambda = 0{,}04$ ist ein Segment nur $0{,}44$ mm lang — mit dem 1-mm-Draht von dort läge die Rechnung außerhalb des MoM-Gültigkeitsfensters (Segmentlänge $\gtrsim 4$ Radien). Also **Drahtradius 0,1 mm**; damit bleibt die ganze Reihe im grünen Bereich. **Messgrößen:** $Z = R + \mathrm{j}X$ am Speisepunkt. **Erfolgskriterium:** R auf der Theorie-Parabel (log-log-Steigung 2) für $l/\lambda \leq 0{,}15$; für längere Dipole darf die Kurzdipol-Formel ehrlich abweichen.

Vorhersage-Punkt (PRIMM): Der Blindanteil X — wird er für kürzere Dipole größer oder kleiner? Denk an Kapitel 24: Ein Dipol *unterhalb* seiner Resonanzlänge ist kapazitiv, und ein kurzer Stab ist ein winziger Kondensator …

```
# von oben: nichts Neues — NEC-Bausteine aus Kap. 11/24
import necpp

C_LICHT = 299792458.0
F_MHZ = 2450.0
LAM = C_LICHT / (F_MHZ * 1e6)
R_DRAHT = 0.0001                 # 0,1 mm — Kap.-24-Segmentregel!
SEG = 11


def dipol_z(l_lambda):
    """Eingangsimpedanz eines mittengespeisten Einzeldipols."""
    ctx = necpp.nec_create()
    halb = l_lambda * LAM / 2
    necpp.nec_wire(ctx, 1, SEG, 0, -halb, 0, 0, halb, 0,
                   R_DRAHT, 1.0, 1.0)
    necpp.nec_geometry_complete(ctx, 0)
    necpp.nec_fr_card(ctx, 0, 1, F_MHZ, 0)
    necpp.nec_ex_card(ctx, 0, 1, (SEG + 1) // 2, 0, 1.0, 0, 0, 0, 0, 0)
```

```
    necpp.nec_rp_card(ctx, 0, 1, 3, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                      180.0, 0.0, 0.0)
    z = complex(necpp.nec_impedance_real(ctx, 0),
                necpp.nec_impedance_imag(ctx, 0))
    necpp.nec_delete(ctx)
    return z


ll = np.array([0.04, 0.06, 0.08, 0.10, 0.15, 0.20, 0.30, 0.40])
z_reihe = np.array([dipol_z(L) for L in ll])

kurz = ll <= 0.15
steigung = np.polyfit(np.log(ll[kurz]),
                      np.log(z_reihe.real[kurz]), 1)[0]
print(f"log-log-Steigung für l/λ ≤ 0,15: {steigung:.3f} (Theorie 2)")
print(f"l/λ = 0,10: R = {z_reihe.real[3]:.2f} Ω "
      f"(Formel: {20 * np.pi**2 * 0.01:.2f}),  "
      f"X = {z_reihe.imag[3]:+.0f} Ω")
print(f"l/λ = 0,04: R = {z_reihe.real[0]:.3f} Ω,  "
      f"X = {z_reihe.imag[0]:+.0f} Ω")
```

```
log-log-Steigung für l/λ ≤ 0,15: 2.075 (Theorie 2)
l/λ = 0,10: R = 2.06 Ω (Formel: 1.97),  X = -1201 Ω
l/λ = 0,04: R = 0.308 Ω,  X = -2165 Ω
```

```
# von oben: ll, z_reihe
fig, (a1, a2) = plt.subplots(1, 2, figsize=(7.4, 3.6))
a1.loglog(ll, z_reihe.real, "bo", label="NEC-Messung")
lfein = np.linspace(0.03, 0.45, 200)
a1.loglog(lfein, 20 * np.pi**2 * lfein**2, "r-", lw=1.3,
          label=r"$20\pi^2(l/\lambda)^2$")
a1.set_xlabel("Dipollänge l/λ")
a1.set_ylabel(r"$R_s$ (Ω)")
a1.set_title("der quadratische Einbruch", fontsize=10)
a1.legend(fontsize=8)

a2.semilogx(ll, -z_reihe.imag, "go-")
a2.set_xlabel("Dipollänge l/λ")
a2.set_ylabel("kapazitiver Blindwiderstand −X (Ω)")
a2.set_title("der Blindanteil-Abgrund", fontsize=10)
fig.tight_layout()
plt.show()
```

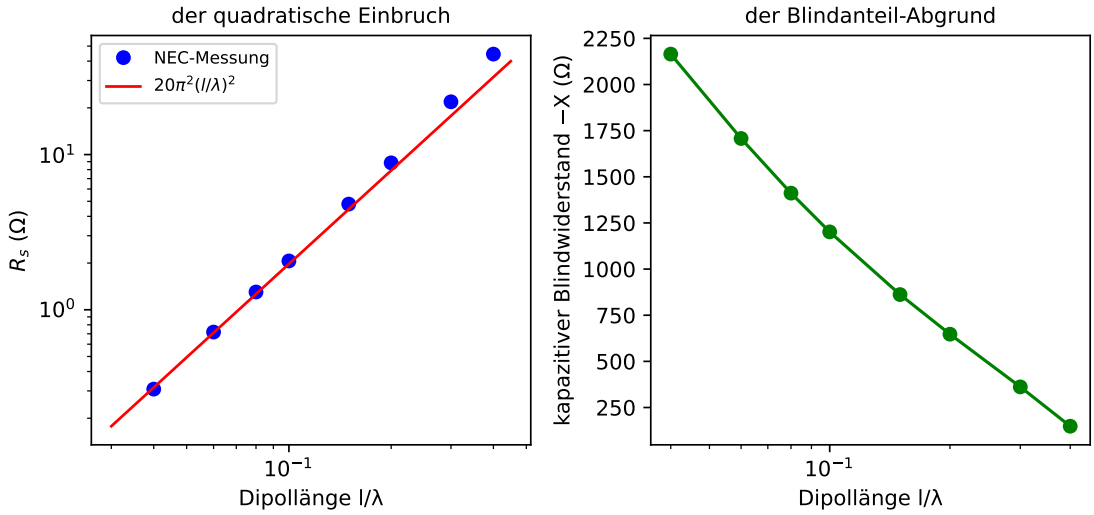


Abbildung 25.1: Die NEC-Messreihe am kurzen Dipol (Draht 0,1 mm): Links der Strahlungswiderstand im log-log-Bild — die Messpunkte liegen für l/λ ≤ 0,15 auf der 20π²(l/λ)²-Parabel (rote Linie, Steigung 2); zu größeren Längen hin wächst R schneller, die Kurzdipol-Formel ist dort ehrlich nur noch Asymptote. Rechts der Blindanteil: Der kurze Dipol ist ein winziger Kondensator, und sein kapazitiver Blindwiderstand explodiert zu kurzen Längen hin auf über zweitausend Ohm — gegen zwei Ohm Strahlungswiderstand. Dieser Abgrund, nicht R_s allein, ist das eigentliche Anpassungs-Drama.

Die Antwort auf den Vorhersage-Punkt: Der Blindanteil **explodiert** zu kurzen Dipolen hin — bei $l/\lambda = 0{,}04$ stehen $-2165\ \Omega$ Kondensator-Blindwiderstand gegen $0{,}31\ \Omega$ Strahlungswiderstand. Eine kleine Antenne ist elektrisch fast *nur* Kondensator mit einem Hauch Funk darin. Beide Panels zusammen sind die Anklageschrift; das Urteil spricht jetzt die Längenrechnung.

Naheliegende Vermutung: „Reicht die Leistung nicht, nimmt man einen stärkeren Verstärker“

Naheliegende Vermutung: Eine Antenne ist Draht mit Strom darin — strahlt sie zu leise, dreht man eben die Leistung auf. Notfalls funkt auch die Steckdose, man muss nur kräftig genug einspeisen.
Warum sie naheliegt: Bei allem anderen im Funkzimmer hilft mehr Leistung wirklich; und $P = \frac{1}{2}\hat{I}^2 R_s$ sieht aus, als müsste man nur $\hat{I}$ erhöhen.

Was stattdessen stimmt: Der Strom fließt durch R_s **und** durch den Verlustwiderstand R_V des Drahtes — die Leistung teilt sich im Verhältnis $R_s : R_V$. Bricht R_s quadratisch ein, geht praktisch alles in die Drahtheizung: Beim Netzkabel kommen von einem Watt nur Nanowatt-Bruchteile in den Äther (gleich ausgerechnet). Mehr Leistung heizt proportional mehr — das *Verhältnis* bleibt hoffnungslos. Und selbst wer verlustfrei speisen könnte, säße im doppelten Gefängnis aus Blindanteil-Abgrund und Chu-Grenze (unten): Klein gegen λ ist keine Leistungsfrage, sondern Geometrie.

25.2 Die 50-Hz-Längenrechnung

Jetzt das Versprechen aus Kapitel 19. Wir nehmen das schlimmste anzunehmende Szenario: zehn Meter Netzkabel, *absichtlich* als Antenne missbraucht — ein offener Draht, mittig gespeist, voller Herdstrom (16 A effektiv). Die Rechnung läuft in fünf Zeilen:

```
# von oben: C_LICHT
lam_50 = C_LICHT / 50                        # Wellenlänge bei 50 Hz
l_kabel = 10.0
R_s50 = 20 * np.pi**2 * (l_kabel / lam_50)**2

rho_cu, A_kabel = 1.7e-8, 1.5e-6             # Kupfer, 1,5 mm²
R_v = rho_cu * l_kabel / A_kabel             # Drahtwiderstand
eta = R_s50 / (R_s50 + R_v)

print(f"λ(50 Hz) = {lam_50 / 1000:.0f} km → l/λ = "
      f"{l_kabel / lam_50:.2e}")
print(f"R_s = {R_s50:.2e} Ω   gegen   R_V = {R_v:.3f} Ω")
print(f"Wirkungsgrad η = R_s/(R_s + R_V) = {eta:.1e}")
print(f"16-A-Herd: P_funk = I² · R_s = {16**2 * R_s50:.2e} W")
```

```
λ(50 Hz) = 5996 km → l/λ = 1.67e-06
R_s = 5.49e-10 Ω   gegen   R_V = 0.113 Ω
Wirkungsgrad η = R_s/(R_s + R_V) = 4.8e-09
16-A-Herd: P_funk = I² · R_s = 1.41e-07 W
```

(Eine Sorgfaltsfrage am Rande: Gilt bei 50 Hz überhaupt der Gleichstrom-Drahtwiderstand? Ja — die Skin-Tiefe, ab Kapitel 18 ein Bekannter, beträgt

bei 50 Hz in Kupfer 9,3 mm und ist damit viel größer als der Drahtradius; der Strom nutzt den vollen Querschnitt.)

Das Ergebnis verdient ein langsames Aussprechen: Der Strahlungswiderstand von zehn Metern Draht bei 50 Hz beträgt **ein halbes Nano-Ohm**. Der Wirkungsgrad ist fünf *Milliardstel*. Ein voll aufgedrehter Küchenherd, dessen Zuleitung wir mutwillig zur Antenne aufgebogen haben, sendet **0,14 Mikrowatt** — weniger als ein Fitnessarmband. Die 955-km- Nahfeldgrenze aus Kapitel 21 sagte qualitativ „alles Nahfeld"; die $(l/\lambda)^2$-Formel macht daraus eine Zahl, gegen die es keine Berufung gibt.

Und die Wirklichkeit ist noch gnadenloser, denn echte Kabel sind **Doppelleitungen**: Hin- und Rückleiter tragen entgegengesetzte Ströme wenige Millimeter voneinander. In der Sprache von Kapitel 23 ist das die Belegung $[1, -1]$ mit $d \ll \lambda$ — der Faltungssatz multipliziert das Diagramm des Paars an alles dran, und dessen Wert ist winzig:

```
# von oben: lam_50, R_s50
kd = 2 * np.pi * 0.005 / lam_50      # Aderabstand 5 mm
daempfung = (kd / 2)**2              # |1 - e^{ikd}|²/4 ≈ (kd/2)²
print(f"Doppelleitungs-Faktor (kd/2)² = {daempfung:.1e}")
print(f"P_funk des Herds mit echtem Kabel: "
      f"{16**2 * R_s50 * daempfung:.1e} W")
```

```
Doppelleitungs-Faktor (kd/2)² = 6.9e-18
P_funk des Herds mit echtem Kabel: 9.6e-25 W
```

10^{-24} Watt. Zum Vergleich: Das ist die Leistung, die ein einzelnes WLAN-Photon … nein, für diese Zahl gibt es keinen hilfreichen Vergleich mehr. Das Stromnetz funkt nicht — nicht, weil es zu schwach wäre, sondern weil es, gemessen an seiner Wellenlänge, **zu klein** ist. Sechstausend Kilometer Welle gegen zehn Meter Draht: Das ist dasselbe Verhältnis wie eine 2,4-GHz-WLAN-Welle gegen eine Antenne von der Größe eines Virus.

25.3 Pendeln statt senden

Wohin geht dann die Feldenergie, die ein 50-Hz-Kabel zweifellos um sich herum aufbaut? Kapitel 21 hat es am Hertz-Dipol gemessen: Im Nahfeld dominiert die **Blindleistung** — das Verhältnis Blind zu Wirk wächst wie $1/(kr)^3$. Die Energie schwappt im Takt der Schwingung vom Draht ins Feld und wieder zurück, wie Wasser in einer schwappenden Wanne; nur ein Riesel-Anteil

entkommt als Welle. Mit den analytischen Hertz-Dipol-Feldern (genau denen, deren Beträge Kapitel 21 über anderthalb Dekaden bestätigt hat) lässt sich das Schwappen direkt ansehen — als Zeitverlauf der radialen Energiestromdichte $S_r(t)$:

```
# von oben: nichts Neues — die komplexen Hertz-Felder (Kap. 21),
# dimensionslos in u = kr, hier bei θ = 90° (Äquator)


def hertz_eth_hph(u):
    """E_theta und H_phi des Hertz-Dipols (komplex, normiert)."""
    eiu = np.exp(1j * u)
    H = (1 / u - 1j / u**2) * eiu
    E = (1 / u - 1j / u**2 - 1 / u**3) * eiu
    return E, H


t = np.linspace(0, 4 * np.pi, 400)          # zwei Perioden
fig, ax = plt.subplots(figsize=(7.0, 4.0))
for u, farbe, name in ((0.3, "tab:blue", "kr = 0,3 (Nahfeld)"),
                       (5.0, "tab:red", "kr = 5 (Fernfeld)")):
    E, H = hertz_eth_hph(u)
    S_r = np.real(E * np.exp(-1j * t)) * np.real(H * np.exp(-1j * t))
    ax.plot(t / (2 * np.pi), S_r / np.abs(S_r).max(), color=farbe,
            lw=1.6, label=name)
    ax.axhline(S_r.mean() / np.abs(S_r).max(), color=farbe, ls="--",
               lw=0.9, alpha=0.7)
ax.axhline(0, color="0.5", lw=1)
ax.set_xlabel("Zeit (Perioden)")
ax.set_ylabel(r"$S_r(t)$, normiert")
ax.set_title("Nahfeld pendelt (S_r wird negativ), Fernfeld fließt")
ax.legend(fontsize=9)
plt.show()

E, H = hertz_eth_hph(0.3)
S = E * np.conj(H)
print(f"kr = 0,3: |Blind/Wirk| = {abs(S.imag / S.real):.1f}"
      f"   (Theorie 1/(kr)³ = {1 / 0.3**3:.1f})")
```

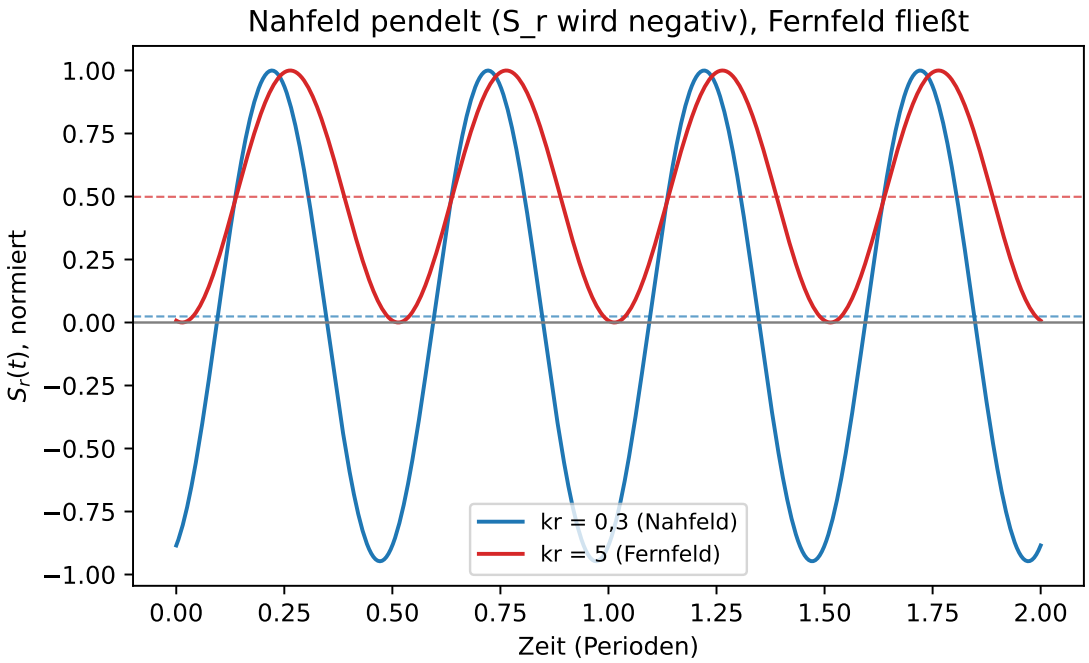


Abbildung 25.2: Der radiale Energiestrom S_r(t) des Hertz-Dipols über zwei Schwingungsperioden, normiert auf den jeweiligen Maximalwert: Im tiefen Nahfeld (kr = 0,3, blau) pendelt die Energie — fast eine halbe Periode lang ist S_r negativ, das Feld zahlt zurück, was es eben erst bekommen hat; netto bleibt fast nichts (gestrichelte Mittelwert-Linie). Im Fernfeld (kr = 5, rot) fließt die Energie stetig nach außen, S_r wird nie negativ. Ein 50-Hz-Kabel lebt bei kr ~ 10^{-5} — sein Feld ist reines Schwappen.

```
kr = 0,3: |Blind/Wirk| = 37.0   (Theorie 1/(kr)³ = 37.0)
```

Die blaue Kurve wird fast so tief negativ, wie sie positiv wird — netto bleibt bei $kr = 0{,}3$ nur ein Bruchteil; das Blind-zu-Wirk-Verhältnis trifft die $1/(kr)^3$-Formel aus Kapitel 21 exakt. Genau dieses Schwappen *ist* übrigens der 50-Hz-Brumm, den eine Gitarre am Verstärker einfängt: kein empfangener Funk, sondern direktes Mitbaden im pendelnden Nahfeld — Übung 25.1 zieht die Grenze sauber.

25.4 Das erste Gefängnis: der Blindanteil-Abgrund

Zurück zur Anpassungsfrage. Wer die kleine Antenne trotzdem betreiben will, muss ihre riesige Kapazität wegstimmen — eine **Verlängerungsspule** in Serie liefert $+\mathrm{j}|X|$ gegen das $-\mathrm{j}|X|$ des Stabs (das L-Glied-Denken aus Kapitel 22/24). Auf dem Papier kostet das nichts; real hat jede Spule Drahtverluste, zusammengefasst in ihrer **Güte** $Q_L = |X_L|/R_{\text{Spule}}$ (die Bauteil-Schwester der Resonator-Güte aus Kapitel 15). Eine solide Luftspule schafft $Q_L \approx 100$ — und damit wird die Rechnung bitter:

```
# von oben: dipol_z()
z = dipol_z(0.10)
R_spule = abs(z.imag) / 100             # Q_L = 100
eta_gesamt = z.real / (z.real + R_spule)
print(f"l/λ = 0,1:  Z = {z.real:.2f} {z.imag:+.0f}j Ω")
print(f"Spule kompensiert {abs(z.imag):.0f} Ω und verheizt dabei "
      f"{R_spule:.1f} Ω")
print(f"η = R_s/(R_s + R_Spule) = {eta_gesamt:.2f}")
```

```
l/λ = 0,1:  Z = 2.06 -1201j Ω
Spule kompensiert 1201 Ω und verheizt dabei 12.0 Ω
η = R_s/(R_s + R_Spule) = 0.15
```

Die Spule, die den $\lambda/10$-Dipol überhaupt erst anschließbar macht, frisst **85 % der Leistung**. Je kürzer die Antenne, desto größer $|X|$, desto mehr verheizt die Kompensation — der Blindanteil-Abgrund bestraft das Schrumpfen doppelt. (Genau deshalb fühlen sich die Gummiwendel-Antennen an Handfunkgeräten so an, wie sie sich anfühlen: Die Wendel *ist* die Verlängerungsspule, über die Länge verteilt — ein erzwungener Kompromiss, kein Wunder.)

25.5 Das zweite Gefängnis: die Chu-Grenze

Angenommen, jemand baut die Wunderspule mit $Q_L = 10\,000$ aus Supraleiter — wäre die kleine Antenne dann gerettet? Nein, und das Verbot ist diesmal fundamental. Die kompensierte kleine Antenne ist ein **Resonator** (Kapitel 15/19): viel pendelnde Nahfeld-Energie, wenig entkommende. Das Verhältnis ist aber genau die Definition der Güte Q — und Kapitel 15 hat eingebrannt, was hohe Güte für das Spektrum heißt: **endlich scharf**, Bandbreite $\Delta f = f/Q$.

L. J. Chu hat 1948 gezeigt, dass dieses Q eine *untere* Schranke hat, die nur von der elektrischen Größe der Antenne abhängt — vom Radius a der kleinsten Kugel, die sie umschließt, gemessen in $ka = 2\pi a/\lambda$:

$$Q \;\geq\; \frac{1}{(ka)^3} + \frac{1}{ka}.$$

Die Idee dahinter ist Kapitel-21-Stoff: Das $1/(kr)^3$- Pendelfeld *gehört* zu jeder kleinen Quelle; unterhalb von $r = a$ kann keine Bauform es abschütteln, und sein Energievorrat wächst zwingend wie $1/(ka)^3$. Kleine Antenne heißt hohes Q heißt **schmale Bandbreite** — egal wie genial die Konstruktion:

```
# von oben: C_LICHT


def chu_q(ka):
    return 1 / ka**3 + 1 / ka


beispiele = [("WLAN-Stummel", 2.45e9, 0.03),
             ("Autoschlüssel", 433e6, 0.02),
             ("NFC-Karte", 13.56e6, 0.04),
             ("MW-Ferritstab", 1e6, 0.05)]

fig, ax = plt.subplots(figsize=(7.0, 4.2))
ka_fein = np.logspace(-3.2, 0.6, 300)
ax.loglog(ka_fein, chu_q(ka_fein), "b-", lw=1.6,
          label="Chu: Q ≥ 1/(ka)³ + 1/(ka)")
for name, f, a in beispiele:
    ka = 2 * np.pi * f / C_LICHT * a
    ax.plot(ka, chu_q(ka), "ro", ms=6)
    ax.annotate(f" {name}", xy=(ka, chu_q(ka)), fontsize=9)
    print(f"{name:14s}: ka = {ka:7.4f} → Q ≥ {chu_q(ka):14.1f}"
          f" → Bandbreite ≤ {100 / chu_q(ka):9.4f} %")
ax.set_xlabel("elektrische Größe ka = 2πa/λ")
ax.set_ylabel("kleinstmögliches Q")
ax.set_title("Das Bandbreiten-Gefängnis: klein gegen λ heißt schmal")
ax.legend(fontsize=9)
plt.show()
```

```
WLAN-Stummel  : ka =  1.5404 → Q ≥            0.9 → Bandbreite ≤  108.3743 %
Autoschlüssel : ka =  0.1815 → Q ≥          172.8 → Bandbreite ≤    0.5788 %
```

```
NFC-Karte     : ka =  0.0114 → Q ≥       680800.1 → Bandbreite ≤    0.0001 %
MW-Ferritstab : ka =  0.0010 → Q ≥    868986376.0 → Bandbreite ≤    0.0000 %
```

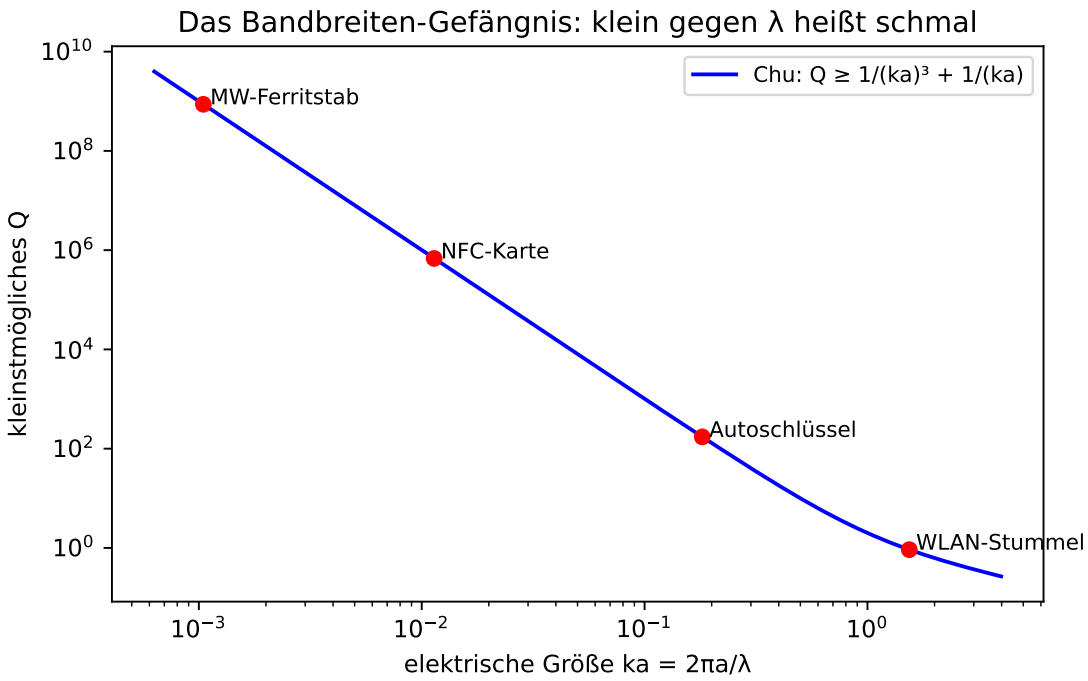


Abbildung 25.3: Die Chu-Grenze: das kleinstmögliche Q (und damit die größtmögliche relative Bandbreite ~1/Q) über der elektrischen Größe ka der Antenne. Reale Funkgeräte (rote Punkte) drängen sich am Knie der Kurve: Der WLAN-Stummel (ka ≈ 1,5) ist frei, der Autoschlüssel (ka ≈ 0,18) muss mit unter einem Prozent Bandbreite leben — und der Mittelwellen-Ferritstab (ka ≈ 0,001) dürfte als Sender nur noch Millihertz übertragen. Als Empfänger funktioniert er trotzdem; warum, klärt der nächste Abschnitt.

Der Autoschlüssel bei 433 MHz passt mit seinen $0{,}6\,\%$ maximaler Bandbreite gerade noch durch (sein Signal ist langsam — wenige Kilobit pro Sekunde reichen zum Aufsperren). Der Mittelwellen-Ferritstab dürfte als *Sender* nur noch Bruchteile von Hertz übertragen: ein Morsezeichen pro Tag. Damit ist auch der letzte Fluchtweg der 50-Hz-Funkidee versperrt — selbst die verlustfreie, perfekt angepasste Zehn-Meter-Antenne hätte bei $ka \approx 10^{-5}$ ein Mindest-Q von 10^{15}: Sie könnte ihr Signal um *Femtohertz* modulieren.

25.6 Drei Auswege — und was sie wirklich sind

Ausweg 1: Empfangen ist nicht Senden. Das Mittelwellenradio im Oldtimer empfängt mit einem Meter Stab bei $\lambda = 300$ m, also $l/\lambda = 0{,}003$ und $\eta \sim 10^{-4}$ — und funktioniert. Der Trick ist keiner: Beim *Empfang* zählt nicht der Wirkungsgrad, sondern das Verhältnis von Signal zu Rauschen, und auf Mittelwelle kommt das Rauschen nicht aus dem Empfänger, sondern **vom Himmel** (Gewitter rund um den Globus, atmosphärisches Knistern — zigtausendfach stärker als das Eigenrauschen der Elektronik). Eine ineffiziente Antenne schwächt Signal *und* Himmelsrauschen um denselben Faktor: Das Verhältnis bleibt, solange das geschwächte Rauschen immer noch über dem Empfängerrauschen liegt. Die Reziprozität aus Kapitel 23 ist also nicht verletzt — die Antenne ist in beide Richtungen gleich schlecht; nur ist Schlechtsein beim Empfang oft egal. (Genau deshalb ist der „Ferritstab" im Radio kein Skandal, als Sender aber unbrauchbar.)

Ausweg 2: das Nahfeld zum Zweck erklären. Die NFC-Karte an der Bürotür arbeitet bei 13,56 MHz — $\lambda/2\pi = 3{,}5$ m, die Reichweite aber nur Zentimeter. NFC findet *vollständig im Pendelfeld* statt: Lesegerät- und Kartenspule bilden einen **Transformator durch Luft** (Kapitel 19); Energie schwappt induktiv hinüber, statt als Welle davonzulaufen. Der lächerliche Strahlungswiderstand der kleinen Schleife ($5 \cdot 10^{-7}\ \Omega$ — für Schleifen gilt sogar eine vierte Potenz, $R_s \propto (C/\lambda)^4$) ist hier ein **Feature**: Was nicht strahlt, ist drei Meter weiter unhörbar — gut für die Batterie und für die Abhörsicherheit. Das vermeintliche Versagen der kleinen Antenne, als Geschäftsmodell.

Ausweg 3: rohe Größe. Wer wirklich bei Kilohertz *senden* muss — Längstwellenfunk erreicht getauchte U-Boote, weil nur solche Wellen ins Meerwasser eindringen —, baut Antennen von der Größe einer Ortschaft. Der schwedische Sender Grimeton (17,2 kHz, seit 1924 in Betrieb, heute Weltkulturerbe) spannt seine Drähte über sechs 127-Meter-Türme und zwei Kilometer Länge — und erreicht trotzdem nur einstellige Prozente Wirkungsgrad bei zweihundert Kilowatt Maschinenleistung (Übung 25.4 rechnet das Modell). Die $(l/\lambda)^2$-Formel verhandelt nicht; man kann sie nur mit Beton und Stahl ein Stück hinaufklettern.

25.7 Das Kapitel-Programm

`programme/kap25/kap25_kleine_antenne.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die NEC-Reihe (Steigung 2 ± 0,2, Formeltreue ≤ 9 % für $l/\lambda \leq 0{,}15$, Blindabgrund < $-2000\ \Omega$), die komplette 50-Hz-Längenrechnung samt Doppelleitungsfaktor ($< 10^{-23}$ W), die

Spulen-Rechnung (η zwischen 10 und 20 %), die Chu-Werte der vier Geräte und die exakte Konsistenz der Hertz-Formeln ($\langle S_r \rangle (kr)^2$ konstant auf 10^{-12}, Blind/Wirk $= 1/(kr)^3$, Pendel-Nachweis bei $kr = 0{,}3$). Läuft in Sekunden.

Merkkasten

- **Strahlungswiderstand:** $P = \frac{1}{2}\hat{I}^2 R_s$ — Funk als Widerstand im Schaltplan ($\lambda/2$-Dipol: 73 Ω). Für kurze Dipole bricht er ein: $R_s = 20\pi^2 (l/\lambda)^2$ — halbe Länge, ein Viertel Widerstand.
- **Die 50-Hz-Längenrechnung:** 10 m Draht bei $\lambda = 6000$ km → $R_s \approx 5 \cdot 10^{-10}$ Ω, $\eta \approx 5 \cdot 10^{-9}$; als Doppelleitung nochmals $(kd/2)^2 \approx 10^{-17}$ obendrauf. Das Netz funkt nicht — sein Feld **pendelt** ($S_r(t)$ wird im Nahfeld negativ; Blind/Wirk $= 1/(kr)^3$).
- **Doppeltes Gefängnis:** Der kapazitive Blindanteil explodiert (-2165 Ω bei $l = 0{,}04\,\lambda$) und die Verlängerungsspule verheizt den Rest; und selbst verlustfrei gilt **Chu**: $Q \geq 1/(ka)^3 + 1/(ka)$ — kleine Antenne heißt schmale Bandbreite, fundamental.
- **Auswege mit Augenmaß:** Empfang verzeiht Ineffizienz (Himmelsrauschen dominiert), NFC nutzt das Pendelfeld absichtlich (Trafo durch Luft), Längstwellen-Sender kaufen Prozente mit Bauwerken.
- Eine Antenne ist klein oder groß **nur im Verhältnis zur Wellenlänge** — das Leitmotiv dieses Buchs, hier als Geldfrage.

Roter Faden

Mit diesem Kapitel schließt sich Teil VI, und fast jeder Faden lief hier zusammen: das **Knick-Argument und die Nahfeld-Tabelle aus Kapitel 21** (die 955-km-Zeile hat jetzt ihre Zahl), die **Güte aus Kapitel 15** und der **Resonator aus Kapitel 19** (Chu ist ihre Verschwörung gegen kleine Antennen), **Anpassung und Verluste aus Kapitel 22/24** (die Verlängerungsspule als bittere L-Glied-Pointe), der **Faltungssatz aus Kapitel 23** (die Doppelleitung als $[1, -1]$-Belegung) und die **Skin-Tiefe aus Kapitel 18**. Nach vorn: Teil VII verlässt die Antennen und fragt, was passiert, wenn *periodische Strukturen* Wellen führen, sperren und falten — **Kapitel 26** beginnt mit dem photonischen Kristall; und das pendelnde Nahfeld kehrt dort als evaneszente Blochwelle in der Bandlücke wieder.

Übungen

Ü 25.1 (Verstehen). Eine E-Gitarre brummt am Verstärker mit 50 Hz, sobald sie in der Nähe eines Netzkabels liegt — dabei haben wir gerade bewiesen, dass das Netz nicht funkt (10^{-24} W!). Löse den scheinbaren Widerspruch auf: Was genau fängt der Tonabnehmer ein, und warum hilft schon ein Meter Abstand so deutlich?

Musterlösung zu Ü 25.1

Der Tonabnehmer empfängt keine *Welle*, er badet im **pendelnden Nahfeld**: Das 50-Hz-Magnetfeld des Kabels durchsetzt seine Spule und induziert direkt eine Spannung — derselbe Mechanismus wie bei NFC, nur ungewollt (ein Transformator durch Luft, Kapitel 19). Mit „Funk" hat das nichts zu tun; die Energie kehrt fast vollständig ins Kabel zurück ($S_r(t)$ pendelt, Abbildung 25.2). Deshalb hilft Abstand so drastisch: Das quasistatische Nahfeld eines Leiterpaars fällt mit $1/r^2$ bis $1/r^3$ (Kapitel 21 — die steilen Stockwerke *unterhalb* der $1/r$-Welle), während eine echte Funkwelle nur mit $1/r$ abklingt. Ein Meter statt zehn Zentimeter heißt hundert- bis tausendfach weniger Brumm — bei einem Sender wären es nur zehnfach.

Ü 25.2 (Verändern). Miss die R_s-Reihe mit einem zehnmal dickeren Draht (Radius 1 mm statt 0,1 mm) für $l/\lambda = 0{,}06$ bis $0{,}15$ nach. Sage vorher: Ändert sich der *Strahlungswiderstand*? Und der *Blindanteil*? (Denk daran, was R_s bestimmt — und was einen Kondensator.) Prüfe auch, ob die Reihe noch im MoM-Gültigkeitsfenster aus Kapitel 24 liegt.

Musterlösung zu Ü 25.2

```
# von oben: dipol_z()-Bauweise, necpp, LAM, SEG


def dipol_z_dick(l_lambda, radius=0.001):
    ctx = necpp.nec_create()
    halb = l_lambda * LAM / 2
    necpp.nec_wire(ctx, 1, SEG, 0, -halb, 0, 0, halb, 0,
                   radius, 1.0, 1.0)
    necpp.nec_geometry_complete(ctx, 0)
    necpp.nec_fr_card(ctx, 0, 1, F_MHZ, 0)
    necpp.nec_ex_card(ctx, 0, 1, (SEG + 1) // 2, 0,
                      1.0, 0, 0, 0, 0, 0)
    necpp.nec_rp_card(ctx, 0, 1, 3, 0, 5, 0, 0, 90.0, 0.0, 0.0,
                      180.0, 0.0, 0.0)
    z = complex(necpp.nec_impedance_real(ctx, 0),
                necpp.nec_impedance_imag(ctx, 0))
    necpp.nec_delete(ctx)
    return z


print("l/λ     dünn (0,1 mm)        dick (1 mm)        Seg/R")
for L in (0.06, 0.10, 0.15):
    zd = dipol_z(L)
    zk = dipol_z_dick(L)
    segr = L * LAM * 1000 / SEG / 1.0
    print(f"{L:.2f}  {zd.real:6.2f} {zd.imag:+7.0f}j   "
          f"{zk.real:6.2f} {zk.imag:+7.0f}j    {segr:4.1f}")
```

```
l/λ     dünn (0,1 mm)        dick (1 mm)        Seg/R
0.06    0.72   -1708j     0.11    -169j     0.7
0.10    2.06   -1201j     1.14    -284j     1.1
0.15    4.80    -862j     3.76    -280j     1.7
```

Der Strahlungswiderstand bleibt (im Rahmen weniger Prozent) gleich — er hängt am Produkt Strom × Länge, nicht am Drahtquerschnitt. Der **Blindanteil** dagegen schrumpft beim dicken Draht deutlich: Ein dicker Stab ist ein größerer Kondensator (mehr Oberfläche), also weniger kapazitiver Blindwiderstand — die Verlängerungsspule hat weniger zu kompensieren und verheizt weniger. Dicke Kurzantennen sind deshalb effizienter anzupassen; auf die Gültigkeit muss man nur bei den kürzesten Längen achten ($l/\lambda = 0{,}06$: Segmentlänge $\approx 2{,}4$ Radien —

schon grenzwertig, die 0,06er-Zeile des dicken Drahts ist mit Vorsicht zu lesen).

Ü 25.3 (Verstehen/Übertragen). Ein Mittelwellensender (1 MHz) speist 100 kW in seinen 100-m-Sendemast; dein Autoradio empfängt ihn mit einem 1-m-Stab. Berechne die l/λ-Werte beider Antennen und ihre Strahlungswiderstände nach der Kurzdipol-Formel. Warum braucht die *Sende*-Seite den großen Mast, während die *Empfangs*-Seite mit dem Stummel davonkommt?

Musterlösung zu Ü 25.3

```
# von oben: nichts (reine Handrechnungs-Kontrolle)
lam_mw = 299792458.0 / 1e6
for name, l in (("Sendemast 100 m", 100.0), ("Autostab 1 m", 1.0)):
    print(f"{name}: l/λ = {l / lam_mw:.4f}, "
          f"R_s = {20 * np.pi**2 * (l / lam_mw)**2 * 1000:.3f} mΩ "
          f"(Dipol-Näherung)")
```

```
Sendemast 100 m: l/λ = 0.3336, R_s = 21962.832 mΩ (Dipol-Näherung)
Autostab 1 m: l/λ = 0.0033, R_s = 2.196 mΩ (Dipol-Näherung)
```

Der Mast kommt auf $l/\lambda = 0{,}33$ — gar nicht mehr „elektrisch klein" (real als Monopol über leitendem Boden gerechnet, mit ordentlichem R_s um Dutzende Ohm): Beim **Senden** zählt jedes Prozent Wirkungsgrad, denn verlorene Kilowatt sind Heizkosten und Reichweite. Der Autostab liegt bei $l/\lambda = 0{,}003$ und $R_s \approx 2$ mΩ — als Sender hoffnungslos, als **Empfänger** unproblematisch: Auf Mittelwelle dominiert das atmosphärische Rauschen aus der Antenne das Eigenrauschen des Empfängers um Größenordnungen, und eine schwache Antenne schwächt Signal und Himmelsrauschen gleichermaßen — das Signal-zu-Rausch-Verhältnis bleibt, bis das Antennensignal unter das Elektronikrauschen fällt. Senden ist ein Energieproblem, Empfangen ein Rauschproblem.

Ü 25.4 (Übertragen). Der Längstwellensender Grimeton (17,2 kHz, $\lambda \approx 17{,}4$ km) nutzt als strahlendes Element im Wesentlichen die **vertikalen** 127-m-Speiseleitungen seiner Antenne (die 2 km horizontaler Draht wirken vor allem als kapazitive „Dachlast", die den Strom in den Vertikalteilen groß und gleichmäßig macht — wie ein verbreitertes Dreieck). Modelliere die Anlage grob als Monopol über perfekter Erde: h/λ? Für einen *dachbelasteten* Monopol gilt $R_s \approx 40\pi^2(h/\lambda)^2$ (gleichmäßiger Strom statt Dreieck, Monopol statt Dipol). Wie groß ist R_s — und welcher Wirkungsgrad folgt, wenn Erdungs- und Spulenverluste zusammen etwa $1{,}5\ \Omega$ betragen?

Musterlösung zu Ü 25.4

```
# von oben: nichts (Handrechnungs-Kontrolle)
lam_g = 299792458.0 / 17.2e3
h = 127.0
R_s_g = 40 * np.pi**2 * (h / lam_g)**2
print(f"h/λ = {h / lam_g:.4f}")
print(f"R_s ≈ {R_s_g:.2f} Ω")
print(f"η = {R_s_g / (R_s_g + 1.5):.2f}")
```

```
h/λ = 0.0073
R_s ≈ 0.02 Ω
η = 0.01
```

Rund $0{,}02$ Ohm Strahlungswiderstand gegen anderthalb Ohm Verluste: η im Bereich weniger Prozent — von den 200 kW der Maschine erreichen nur einige Kilowatt den Äther, und das galt 1924 als Triumph (es reichte für Telegramme nach New York). Die historischen Berichte nennen Wirkungsgrade derselben Größenordnung. Wer die Zahlen variiert, sieht den Hebel: Doppelte Masthöhe vervierfacht R_s — deshalb die 127-Meter-Türme, deshalb die Dachlast (ohne sie fiele der Strom dreieckig ab und R_s noch einmal um den Faktor 4). Kleiner-als-λ bleibt auch mit Weltkulturerbe-Aufwand ein teures Geschäft.

Das Kleingedruckte

Chu, etwas genauer. Chus Beweis zerlegt das Feld außerhalb der umschließenden Kugel in Kugelwellen-Moden und zeigt: Jede Mode unterhalb von $ka \approx 1$ speichert zwingend viel mehr Energie, als sie abstrahlt — das beste Q bekommt, wer nur die niedrigste Mode anregt, und genau dafür gilt die Formel. Spätere Verfeinerungen (Harrington, McLean) ändern Faktoren, nicht das Fazit. Die Schranke gilt für *passive, lineare* Antennen; aktiv nachgestimmte Systeme (der Empfänger springt mit der Resonanz dem Signal hinterher) können sie scheinbar umgehen — bezahlt mit Elektronik, Energie und Rauschen.

Das zweite verbotene Paradies: Superdirektivität. Auf dem Papier kann eine *beliebig kleine* Anordnung beliebig scharf bündeln — die Kapitel-23-Mathematik verbietet es nicht (Belegungen mit wild alternierenden Vorzeichen). Real explodieren dabei Ströme, Verluste und Toleranzen exponentiell; es ist dieselbe Sorte fundamentaler Strafe wie Chu, nur für die Richt-

wirkung. Wer beides zusammen liest, versteht, warum Antennentechnik ein Handwerk der ehrlichen Kompromisse ist.

ELF: Funken bei 76 Hertz. Es ging übrigens doch — einmal. Die US-Marine betrieb bis 2004 das „Project ELF": Sendeanlagen bei 76 Hz ($\lambda \approx 4000$ km!), deren „Antenne" aus **135 Kilometern** Überlandleitung bestand, geerdet über granitnem Untergrund, sodass der Rückstrom durch Kilometer Erdtiefe floss — die wirksame Schleifenfläche machte den (l/λ)-Faktor gerade erträglich. Megawatt hinein, *einstellige Watt* hinaus, Datenrate: Minuten pro Buchstabengruppe — genug, um U-Boote in aller Welt zum Auftauchen für richtigen Funk zu rufen. Die 50-Hz-Funkfrage hat also eine ehrliche Antwort: Man *kann* — mit der Infrastruktur eines Bundesstaats, für drei Zeichen pro Minute.

Und warum Trafos brummen dürfen: Alles, was das Stromnetz an Feld erzeugt, bleibt pendelndes Nahfeld — es koppelt in Gitarren, Messleitungen und schlecht geschirmte Verstärker, aber es *reist* nicht. Elektromagnetische Verträglichkeit (EMV) bei 50 Hz ist deshalb ein Nahfeld-Handwerk: Abstand, Verdrillen (die $[1, -1]$-Auslöschung im Kleinen!), Schirmung — kein Funkproblem.

26 Photonische Kristalle und Bandlücken

Der Morpho-Falter aus Südamerika leuchtet in einem Blau, das kein Farbstoff der Welt hinbekommt — metallisch, fast grell, über hundert Meter sichtbar. Das Verblüffende: In seinen Flügeln steckt **kein blaues Pigment**. Zerreibt man eine Flügelschuppe, bleibt unscheinbares Braun zurück. Die Farbe sitzt nicht in der Chemie, sondern in der *Geometrie*: Die Schuppen tragen Lamellen aus Chitin, gestapelt im Abstand von etwa einem Zehntel Mikrometer. Eine Struktur, die sich im Takt der Lichtwellenlänge wiederholt, wirft eine Farbe komplett zurück und lässt die anderen passieren.

Dieselbe Idee, nur präziser gefertigt, sitzt in jedem Laser: Seine Endspiegel müssen besser reflektieren als jedes polierte Metall — und bestehen ausgerechnet aus *durchsichtigen* Glasschichten. In diesem Kapitel bauen wir solche Strukturen nach: erst den Schichtstapel, der zum nahezu perfekten Spiegel wird, dann den zweidimensionalen Kristall aus dielektrischen Stäbchen, der ein ganzes Frequenzband in **allen** Richtungen sperrt — eine **photonische Bandlücke**. Und zum Schluss nutzen wir das Verbot konstruktiv: Ein fehlender Stab wird zur Lichtleitung, die um eine 90°-Ecke führt, ohne dass nennenswert etwas verloren geht.

Damit eröffnet dieses Kapitel den Teil VII des Buchs: Es geht nicht mehr darum, Wellen zu *erzeugen* (Teil VI), sondern darum, was **periodische Strukturen** mit ihnen anstellen.

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum viele schwache Echos einen fast perfekten Spiegel ergeben (λ/4-Bedingung: alle Echos in Phase) — und du hast es in Meep nachgemessen ($T = 10^{-3}$ bei acht Schichtpaaren),
2. … die **Transfer-Matrix** eines Schichtstapels aufstellen (eine 2×2-Matrix pro Schicht, Matrixprodukt für den Stapel) und damit $T(f)$ exakt vorhersagen,

3. … das **Spurkriterium** $|\mathrm{Tr}\, M| \leq 2$ anwenden: erlaubte Bänder und verbotene Lücken im Banddiagramm $\omega(K)$ — und du erkennst darin die Mathematik der Stabilitätsanalyse aus Kapitel 8 wieder,
4. … begründen, warum ein Stapel nur senkrecht sperrt, ein 2D-Kristall aber alle Richtungen — und hast die volle Bandlücke des Stäbchengitters als Schnittmenge zweier Stoppbänder gemessen (0,32–0,44 in Einheiten fa/c),
5. … Defekte als Werkzeug einsetzen: die fehlende Stabreihe als Wellenleiter, der Licht verlustarm um die Ecke führt, und den fehlenden Einzelstab als Resonator,
6. … Strukturfarben (Morpho, Opal) quantitativ erklären — inklusive des Schillerns beim Schrägblick.

26.1 Viele schwache Echos, ein starker Spiegel

Fangen wir bei einer alten Bekannten an: der Grenzfläche. Seit Kapitel 6 wissen wir, was eine einzelne Glasoberfläche mit einer senkrecht auftreffenden Welle macht — sie wirft den Bruchteil

$$r = \frac{n_1 - n_2}{n_1 + n_2}$$

der Amplitude zurück. Für Luft auf Glas ($n = 1{,}5$) sind das $r = -0{,}2$, also magere 4 % der *Leistung* — das schwache Spiegelbild im Schaufenster. Eine einzelne Grenzfläche ist ein erbärmlicher Spiegel.

Aber niemand zwingt uns, bei einer zu bleiben. Wir stapeln zwei Materialien im Wechsel — in diesem Kapitel durchgehend $n_1 = 1{,}5$ (Glas) und $n_2 = 2{,}5$ (ein hochbrechendes Material wie Titandioxid) — und fragen: Was machen die *vielen* Echos miteinander? Kapitel 14 hat die Antwort im Prinzip schon gegeben: Es kommt nicht auf die Zahl der Echos an, sondern auf ihre **Phasen**. Und an den Phasen können wir drehen, denn sie hängen an den Schichtdicken.

Der Trick heißt **λ/4-Stapel** und ist die Umkehrung eines Bausteins aus Kapitel 22: Dort haben wir mit dem λ/4-Transformator Echos gegeneinander *ausgelöscht* (Antireflex); jetzt schalten wir sie *in Reihe*. Zwei Zutaten greifen ineinander:

- **Die Vorzeichen wechseln mit.** An jedem Übergang niedrig → hoch ist $r < 0$, an jedem Übergang hoch → niedrig ist $r > 0$. Benachbarte Grenzflächen reflektieren also gegenphasig — *bevor* wir den Laufweg einrechnen.

- **Der Laufweg dreht die Phase zurück.** Macht jede Schicht (optisch) genau eine Viertelwellenlänge dick, $n_j d_j = \lambda_0/4$, dann sammelt das Echo von der nächsten Grenzfläche eine Hin-und-zurück-Verzögerung von $\lambda_0/2$ ein — eine halbe Periode, Faktor -1.

Minus mal minus: Beide Effekte zusammen bringen **alle** Echos exakt in Phase. Jede weitere Grenzfläche zahlt auf dasselbe Konto ein. Rechnen wir das Konto kurz von Hand an: Bei $n_1 = 1{,}5$ und $n_2 = 2{,}5$ ist jedes Einzelecho

$$|r| = \frac{2{,}5 - 1{,}5}{2{,}5 + 1{,}5} = \frac{1}{4},$$

und acht Schichtpaare haben 16 Grenzflächen. Würden sich die Amplituden einfach aufsummieren, stünden $16 \cdot 0{,}25 = 4$ auf dem Zettel — das Vierfache der einlaufenden Welle, also offensichtlicher Unsinn. Die naive Buchhaltung bricht zusammen, weil jedes Echo unterwegs an allen anderen Grenzflächen erneut gestreut wird. Was wirklich herauskommt, muss ein Experiment zeigen (und danach eine ehrliche Theorie, die alle Mehrfach-Echos mitzählt). Eines ist aber jetzt schon klar: Die Echos verstärken sich, bis der Nachschub versiegt.

Naheliegende Vermutung: „Aus durchsichtigen Zutaten kann man keinen Spiegel bauen“

Warum sie naheliegt: Glas lässt 96 % durch. Wer durchsichtige Schichten stapelt, stapelt scheinbar Durchsichtigkeit — selbst zwanzig Flächen à 4 % „sammeln” nach dieser Logik höchstens die Hälfte des Lichts ein, und real sieht man durch zwanzig Glasscheiben ja auch hindurch.
Was stattdessen stimmt: Reflexionen addieren sich nicht wie Prozente auf einer Rechnung, sondern wie **Zeiger mit Phase** (Kapitel 14). Zwanzig zufällig dicke Scheiben löschen sich teilweise aus — zwanzig *λ/4-gestimmte* Schichten ziehen am selben Strang. Wir messen gleich: Schon acht Paare aus Glas und Titandioxid reflektieren 99,9 % — besser als poliertes Silber. Der Unterschied zwischen „durchsichtig” und „Spiegel” ist keine Materialeigenschaft, sondern eine Frage der Anordnung.

26.1.1 Der Stapel im Experiment

Das Experiment dazu, zum ersten Mal in einer **1D-Zelle**: Unsere Welt ist eine einzige Achse, denn Stapel und Welle hängen nur von z ab — Meep rech-

net dann nur diese Achse durch (`dimensions=1`), und ein kompletter Frequenz-Sweep dauert Sekunden. **Frage:** Wie viel Leistung lässt der Stapel bei welcher Frequenz durch? **Bühne:** acht Schichtpaare $n_1 = 1{,}5/n_2 = 2{,}5$ in λ/4-Dicken, Periode $a = d_1 + d_2 = 1$; daraus folgt die Design-Frequenz $f_0 = 0{,}2667$ (in Meep-Einheiten fa/c). Links und rechts Luft, an beiden Enden PML. **Anfangs-/Randbedingungen:** ein Gauß-Puls (Kapitel 10: ein Lauf, alle Frequenzen) startet vor dem Stapel. **Messgrößen:** der Leistungsfluss hinter dem Stapel, geteilt durch einen Referenzlauf ohne Stapel — exakt die $T(f)$-Methodik der Kapitel 12/17. **Erfolgskriterium:** Bei f_0 müssen die Echos maximal sperren; wie breit das Sperrgebiet ist, wissen wir noch nicht — das *ist* die Messung.

Vorhersage-Punkt (PRIMM): Die λ/4-Bedingung gilt exakt nur bei f_0. Erwartest du eine scharfe Sperr-*Nadel* bei f_0 — oder ein breites Sperr-*Band*? Was passiert weit unterhalb, bei $f \to 0$, wo der ganze Stapel viel dünner ist als eine Wellenlänge?

```
import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)

# --- der Lambda/4-Stapel: Periode a = 1 ---
N1, N2 = 1.5, 2.5
LAM0 = 4.0 / (1.0 / N1 + 1.0 / N2)   # n1*d1 = n2*d2 = lam0/4 und d1+d2 = 1
F0 = 1.0 / LAM0                      # Design-Frequenz 0,2667
D1 = LAM0 / (4.0 * N1)               # 0,625
D2 = LAM0 / (4.0 * N2)               # 0,375
N_PAARE = 8

AUFL = 40
DPML, LUFT = 2.0, 3.0
L_STAPEL = N_PAARE * (D1 + D2)
LZ = 2 * DPML + 2 * LUFT + L_STAPEL
FCEN, DF = 0.27, 0.42                # Puls deckt 0,06 bis 0,48 ab
NFREQ = 401


def stapel_geometrie(n_paare=N_PAARE):
    """Bloecke fuer n_paare Schichtpaare, zentriert um z = 0."""
    geom = []
    z0 = -n_paare * (D1 + D2) / 2
    for i in range(n_paare):
```

```
        za = z0 + i * (D1 + D2)
        geom.append(mp.Block(size=mp.Vector3(mp.inf, mp.inf, D1),
                             center=mp.Vector3(0, 0, za + D1 / 2),
                             material=mp.Medium(index=N1)))
        geom.append(mp.Block(size=mp.Vector3(mp.inf, mp.inf, D2),
                             center=mp.Vector3(0, 0, za + D1 + D2 / 2),
                             material=mp.Medium(index=N2)))
    return geom


def lauf_1d(geom):
    """T(f)-Messlauf: Puls rein, Fluss hinter dem Stapel."""
    sim = mp.Simulation(
        cell_size=mp.Vector3(0, 0, LZ),
        dimensions=1,                # nur die z-Achse wird gerechnet
        resolution=AUFL,
        boundary_layers=[mp.PML(DPML)],
        geometry=geom,
        sources=[mp.Source(mp.GaussianSource(FCEN, fwidth=DF),
                           component=mp.Ex,
                           center=mp.Vector3(0, 0, -LZ / 2 + DPML + 0.5))])
    z_mess = LZ / 2 - DPML - 0.5
    tr = sim.add_flux(FCEN, DF, NFREQ,
                      mp.FluxRegion(center=mp.Vector3(0, 0, z_mess)))
    sim.run(until_after_sources=mp.stop_when_fields_decayed(
        50, mp.Ex, mp.Vector3(0, 0, z_mess), 1e-9))
    return np.array(mp.get_flux_freqs(tr)), np.array(mp.get_fluxes(tr))


fs, P_ref = lauf_1d([])                       # Referenz: leere Zelle
_, P_stapel = lauf_1d(stapel_geometrie())
T_meep = P_stapel / P_ref

i0 = np.argmin(np.abs(fs - F0))
print(f"T bei f0 = {F0:.4f}:  {T_meep[i0]:.2e}")
print(f"entspricht R = {1 - T_meep[i0]:.4f}")
```

```
Using MPI version 4.1, 1 processes
T bei f0 = 0.2667:  1.14e-03
entspricht R = 0.9989
```

99,9 % Reflexion aus Glas und Titandioxid — und der Plot zeigt, dass die Antwort auf den Vorhersage-Punkt „Band” lautet:

```
# von oben: fs, T_meep, F0
fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.semilogy(fs, T_meep, "C0-", lw=1.5, label="Meep, 8 Paare")
ax.axvline(F0, color="0.4", ls="--", lw=1.2,
           label=f"Design-Frequenz $f_0$ = {F0:.3f}")
ax.set_xlim(0.05, 0.48)
ax.set_ylim(1e-4, 2)
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$")
ax.set_title("Die Klippe: ein verbotenes Frequenzband")
ax.legend(loc="lower right", fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

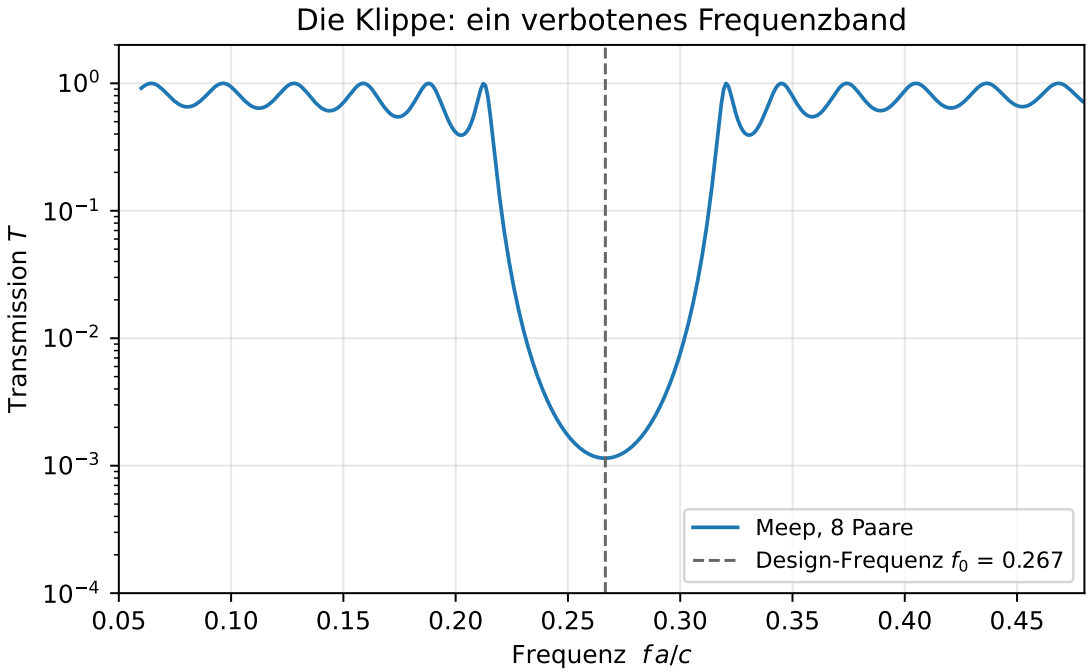


Abbildung 26.1: Transmission durch den 8-Paar-Stapel: kein Nadelstich, sondern eine breite Klippe rund um die Design-Frequenz (gestrichelt). Weit unterhalb ist der Stapel unsichtbar ($T \to 1$): Dort ist er dünn gegen die Wellenlänge — die Kapitel-25-Erfahrung, von der anderen Seite gesehen.

Das ist dieselbe Sorte Klippe wie beim Hohlleiter unterhalb seiner Grenzfrequenz (Kapitel 17) — nur dass hier *mitten im Spektrum* ein Band ausgestanzt

ist: zwischen $f \approx 0{,}22$ und $f \approx 0{,}31$ kommt praktisch nichts durch. Ein solches Band heißt **Stoppband** — und wir werden gleich verstehen, dass es der eindimensionale Vorbote einer photonischen Bandlücke ist.

Wie tief der Boden der Klippe liegt, bestimmt die Zahl der Schichtpaare. Gleicher Aufbau, gleiche Messgröße, nur N variiert — die Frage: Wie skaliert $T(f_0)$ mit N?

Vorhersage-Punkt (PRIMM): Halbiert jedes zusätzliche Paar die Durchlässigkeit? Viertelt es sie? Tipp: Kapitel 17 — wenn in einem verbotenen Bereich „nichts laufen kann", welche Sorte Feld lebt dann dort, und wie hängt es von der Eindringtiefe ab?

```
# von oben: lauf_1d(), stapel_geometrie(), fs, P_ref, F0
NS = np.array([2, 4, 6, 8, 10, 12])
T_N = []
for n in NS:
    _, P_n = lauf_1d(stapel_geometrie(n))
    T_N.append((P_n / P_ref)[np.argmin(np.abs(fs - F0))])
T_N = np.array(T_N)

for n, t in zip(NS, T_N):
    print(f"N = {n:2d} Paare:  T(f0) = {t:.3e}")
faktor_pro_paar = np.exp(np.polyfit(NS, np.log(T_N), 1)[0])
print(f"\nAbfall pro Schichtpaar: Faktor {faktor_pro_paar:.3f}")
```

```
N =  2 Paare:  T(f0) = 4.074e-01
N =  4 Paare:  T(f0) = 6.543e-02
N =  6 Paare:  T(f0) = 8.761e-03
N =  8 Paare:  T(f0) = 1.144e-03
N = 10 Paare:  T(f0) = 1.489e-04
N = 12 Paare:  T(f0) = 1.935e-05

Abfall pro Schichtpaar: Faktor 0.368
```

Jedes Paar drückt die Transmission um denselben **Faktor 0,36** — exponentieller Abfall, das Markenzeichen evaneszenter Felder seit Kapitel 12. Im Stoppband *läuft* die Welle nicht durch den Stapel, sie *sickert* hinein und klingt von Paar zu Paar ab; was am Ende noch übrig ist, tunnelt hinaus (Kapitel 17, exakt dieselbe Physik). Merke dir die 0,36 — in Kürze sagt eine Theorie diesen Faktor ohne ein einziges Experiment voraus. Woher sie kommt, zeigt am schnellsten ein Blick auf das Feld selbst.

26.2 Die Transfer-Matrix: alle Echos auf einmal

Die naive Echo-Summe ist vorhin an den Mehrfachreflexionen gescheitert. Die ehrliche Rechnung gibt es trotzdem — und sie ist überraschend billig. Die Idee: Wir fragen nicht „welche Echos entstehen wo?“, sondern **„was macht eine Schicht mit den Feldern an ihrem Rand?“** In einer Schicht leben genau zwei Unbekannte (eine hinlaufende und eine rücklaufende Teilwelle), und E und H an der Hinterkante hängen *linear* an denen der Vorderkante. Lineare Abbildung von zwei Zahlen auf zwei Zahlen — das ist eine 2×2-Matrix:

$$\begin{pmatrix} E \\ Z_0 H \end{pmatrix}_{\text{hinten}} = \underbrace{\begin{pmatrix} \cos\delta & \frac{\mathrm{i}}{n}\sin\delta \\ \mathrm{i}\, n \sin\delta & \cos\delta \end{pmatrix}}_{M_{\text{Schicht}}} \begin{pmatrix} E \\ Z_0 H \end{pmatrix}_{\text{vorne}}, \qquad \delta = \frac{2\pi f}{c}\, n\, d.$$

Dabei ist δ schlicht die Phasendicke der Schicht (wie viele Radiant die Welle beim Durchqueren sammelt) und n steuert das Verhältnis von E und H — der Wellenwiderstand aus Kapitel 6, in Matrixform gegossen. Der ganze Stapel ist dann das **Matrixprodukt** seiner Schichten, in Laufrichtung von vorn nach hinten — dieselbe Buchhaltungs-Eleganz wie beim Update-Operator in Kapitel 7, und alle Mehrfach-Echos sind automatisch mitgezählt, weil die Matrix die *vollständige* Feldbeziehung transportiert. Aus der Gesamtmatrix fallen r und t mit zwei Zeilen Algebra heraus (Anschluss an die Halbräume; die Formel steht im Code).

Der Versuchsaufbau der Nagelprobe: **Frage:** Trifft die Transfer-Matrix die Meep-Messung — über den ganzen Sweep, parameterfrei? **Bühne:** dieselben acht Paare, dieselben Frequenzen; die Theorie läuft in NumPy in Millisekunden. **Messgrößen:** $T_{\text{TMM}}(f)$ gegen $T_{\text{Meep}}(f)$, als Differenzmaß der rms-Abstand. **Erfolgskriterium:** Abweichung im Prozentbereich (Meep diskretisiert mit 40 Punkten pro a — kleine Restfehler sind ehrlich, Kapitel 8).

```
# von oben: N1, N2, D1, D2, N_PAARE, fs, T_meep


def schichtmatrix(n, d, f):
    """Charakteristische 2x2-Matrix einer Schicht (senkrechter Einfall)."""
    delta = 2.0 * np.pi * f * n * d
    return np.array([[np.cos(delta), 1j * np.sin(delta) / n],
                     [1j * n * np.sin(delta), np.cos(delta)]])


def kette_rt(matrizen):
```

```
    """r und t der Schichtkette zwischen zwei Luft-Halbraeumen."""
    M = np.eye(2, dtype=complex)
    for m in matrizen:
        M = M @ m                       # Matrixprodukt: Schicht fuer Schicht
    nenner = (M[0, 0] + M[0, 1]) + (M[1, 0] + M[1, 1])
    r = ((M[0, 0] + M[0, 1]) - (M[1, 0] + M[1, 1])) / nenner
    t = 2.0 / nenner
    return r, t


def stapel_rt(f, n_paare=N_PAARE):
    mats = [m for _ in range(n_paare)
            for m in (schichtmatrix(N1, D1, f), schichtmatrix(N2, D2, f))]
    return kette_rt(mats)


T_tmm = np.array([np.abs(stapel_rt(f)[1]) ** 2 for f in fs])
R_tmm = np.array([np.abs(stapel_rt(f)[0]) ** 2 for f in fs])

print(f"Energie-Check:  max |R + T - 1| = {np.max(np.abs(R_tmm + T_tmm - 1)):.1e}")
fenster = (fs > 0.10) & (fs < 0.44)
rms = np.sqrt(np.mean((T_meep[fenster] - T_tmm[fenster]) ** 2))
print(f"Meep vs. Transfer-Matrix:  rms = {rms:.4f}")
print(f"T(f0):  Meep {T_meep[np.argmin(np.abs(fs - F0))]:.2e}, "
      f"TMM {np.abs(stapel_rt(F0)[1])**2:.2e}")
```

```
Energie-Check:  max |R + T - 1| = 2.7e-15
Meep vs. Transfer-Matrix:  rms = 0.0074
T(f0):  Meep 1.14e-03, TMM 1.13e-03
```

```
# von oben: fs, T_meep, T_tmm, F0, fenster
fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.semilogy(fs, T_tmm, "C1-", lw=1.6, label="Transfer-Matrix (Theorie)")
ax.semilogy(fs[fenster][::4], T_meep[fenster][::4], "C0o", ms=4,
            mfc="none", label="Meep (jeder 4. Messpunkt)")
ax.axvline(F0, color="0.4", ls="--", lw=1)
ax.set_xlim(0.05, 0.48)
ax.set_ylim(1e-4, 2)
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$")
ax.set_title("Acht Paare, zwei Rechenwege, eine Kurve")
ax.legend(loc="lower right", fontsize=9)
```

```
ax.grid(alpha=0.3)
plt.show()
```

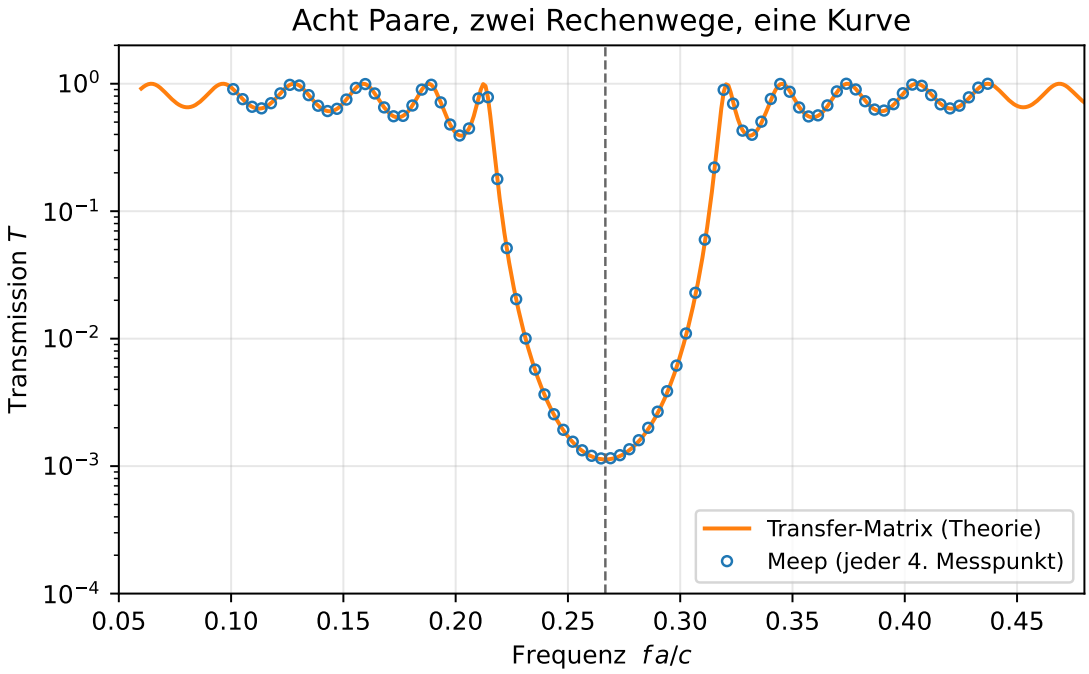


Abbildung 26.2: Die Nagelprobe: Meep-Messpunkte auf der Transfer-Matrix-Linie — rms unter einem Prozent, und am Boden der Klippe (10^{-3}) stimmen beide auf zwei Stellen überein. Zwei völlig verschiedene Rechenwege (Zeitschritte im Gitter gegen 2×2-Matrixalgebra), ein Ergebnis: die Validierungs-Checkliste aus Kapitel 10 in Aktion.

26.3 Das Spurkriterium: erlaubt oder verboten

Die Transfer-Matrix kann mehr, als $T(f)$ ausrechnen — sie beantwortet die Grundsatzfrage dieses Kapitels: **Für welche Frequenzen kann in einem *unendlichen* Stapel überhaupt eine Welle laufen?** Dazu brauchen wir nur die Matrix M *einer* Periode (eine n_1- plus eine n_2-Schicht).

In einem perfekt periodischen Medium muss sich eine laufende Welle nach jeder Periode bis auf einen Phasenfaktor wiederholen:

$$\begin{pmatrix} E \\ Z_0 H \end{pmatrix}_{z+a} = \mathrm{e}^{\mathrm{i}Ka} \begin{pmatrix} E \\ Z_0 H \end{pmatrix}_z .$$

Eine solche Lösung heißt **Blochwelle**, und K ist ihre Wellenzahl — das periodische Gegenstück zum k der ebenen Welle. Der Vergleich mit der Periodenmatrix sagt: $\mathrm{e}^{\mathrm{i}Ka}$ muss ein **Eigenwert von** M sein. Für unsere 2×2-Matrix ist das eine Rechnung von drei Zeilen: Die beiden Eigenwerte $\lambda_\pm$ erfüllen $\lambda_+\lambda_- = \det M = 1$ (verlustfreie Schicht: was reingeht, kommt raus) und $\lambda_+ + \lambda_- = \mathrm{Tr}\, M$. Daraus folgt

$$\cos(Ka) = \frac{\mathrm{Tr}\, M}{2},$$

und jetzt entscheidet eine einzige Zahl über das Schicksal der Welle:

- $|\mathrm{Tr}\, M| \leq 2$: Es gibt ein reelles K — die Blochwelle **läuft**. Erlaubtes Band.
- $|\mathrm{Tr}\, M| > 2$: Der Kosinus müsste größer als eins sein. Das geht nur mit **komplexem** K — die „Welle” klingt räumlich exponentiell ab. Verbotenes Band: die **Bandlücke**.

Falls dir dieses Entweder-oder bekannt vorkommt: Es ist **wörtlich die Mathematik aus Kapitel 8**. Dort haben wir die Zickzack-Mode des Zeitschritt-Operators untersucht und gefunden: $|\mathrm{Tr}| \leq 2$ heißt stabile Schwingung, $|\mathrm{Tr}| > 2$ heißt exponentielle Explosion — die CFL-Grenze. Hier läuft dieselbe Eigenwert-Analyse über eine Periode im **Raum** statt über einen Schritt in der **Zeit**: Statt „explodieren” heißt das Urteil nun „abklingen”. Eine Formel, zwei Physiken.

Der Versuchsaufbau: **Frage:** Wie sieht die Dispersionsrelation $f(K)$ des unendlichen Stapels aus — und wo liegt die Lücke? **Bühne:** Theorieseite: die Spurformel, ausgewertet über ein feines Frequenzraster. Messseite: **eine einzige Periode** in Meep mit Bloch-periodischen Rändern — der `k_point` aus Kapitel 16 gibt das K vor, und Harminv (Kapitel 15) lauscht, welche Eigenfrequenzen in der Zelle leben. Fünf K-Werte, fünf Kurzläufe. **Erfolgskriterium:** Die Meep-Punkte liegen auf dem Spurformel-Band; am Zonenrand ($Ka/2\pi = 0{,}5$) müssen exakt die beiden Kanten unserer gemessenen Klippe herauskommen.

```
# von oben: schichtmatrix(), N1, N2, D1, D2, F0



def spur_halbe(f):
```

```
    """Tr(M)/2 der Periodenmatrix — die Schicksalszahl."""
    M = schichtmatrix(N1, D1, f) @ schichtmatrix(N2, D2, f)
    return float(np.real(M[0, 0] + M[1, 1]) / 2.0)


# Band aus der Spurformel: zu jedem f das K (nan in der Luecke)
f_band = np.linspace(1e-3, 0.5, 2000)
spur = np.array([spur_halbe(f) for f in f_band])
K_band = np.where(np.abs(spur) <= 1,
                  np.arccos(np.where(np.abs(spur) <= 1, spur, 0))
                  / (2 * np.pi),
                  np.nan)

# Lueckenkanten: wo |Tr/2| die 1 kreuzt
verboten = np.abs(spur) > 1
i = np.argmin(np.abs(f_band - F0))
i_lo, i_hi = i, i
while verboten[i_lo - 1]:
    i_lo -= 1
while verboten[i_hi + 1]:
    i_hi += 1
F_LO, F_HI = f_band[i_lo], f_band[i_hi]
print(f"Spurformel: Luecke {F_LO:.4f} .. {F_HI:.4f}")
print(f"Tr/2 bei f0: {spur_halbe(F0):.4f}  "
      f"(= -(n1/n2 + n2/n1)/2 = {-(N1/N2 + N2/N1)/2:.4f})")

bw = (F_HI - F_LO) / F0
bw_formel = (4 / np.pi) * np.arcsin((N2 - N1) / (N2 + N1))
print(f"relative Lueckenbreite: {bw:.4f}  "
      f"(Lehrbuchformel (4/pi)*arcsin(dn/sn): {bw_formel:.4f})")


def meep_moden_bei_k(k):
    """Eigenfrequenzen einer Bloch-periodischen Einheitszelle."""
    geom = [mp.Block(size=mp.Vector3(mp.inf, mp.inf, D1),
                     center=mp.Vector3(0, 0, -0.5 + D1 / 2),
                     material=mp.Medium(index=N1)),
            mp.Block(size=mp.Vector3(mp.inf, mp.inf, D2),
                     center=mp.Vector3(0, 0, -0.5 + D1 + D2 / 2),
                     material=mp.Medium(index=N2))]
    sim = mp.Simulation(
        cell_size=mp.Vector3(0, 0, 1), dimensions=1, resolution=64,
        geometry=geom,
```

```
        k_point=mp.Vector3(0, 0, k),     # Bloch-Phase pro Periode
        sources=[mp.Source(mp.GaussianSource(0.3, fwidth=0.6),
                           component=mp.Ex,
                           center=mp.Vector3(0, 0, -0.5 + 0.123))])
    h = mp.Harminv(mp.Ex, mp.Vector3(0, 0, 0.171), 0.3, 0.6)
    sim.run(mp.after_sources(h), until_after_sources=300)
    return sorted(m.freq for m in h.modes if m.freq > 0.01 and abs(m.Q) > 50)


K_MESS = [0.1, 0.2, 0.3, 0.4, 0.5]
punkte = [(k, f) for k in K_MESS for f in meep_moden_bei_k(k) if f < 0.48]
print("\nMeep-Moden (k, f):",
      ", ".join(f"({k:.1f}, {f:.4f})" for k, f in punkte))
rand = meep_moden_bei_k(0.5)
print(f"Zonenrand K·a/2π = 0,5:  f = {rand[0]:.4f} und {rand[1]:.4f}  "
      f"— die Klippenkanten!")
```

```
Spurformel: Luecke 0.2239 .. 0.3095
Tr/2 bei f0: -1.1333  (= -(n1/n2 + n2/n1)/2 = -1.1333)
relative Lueckenbreite: 0.3211  (Lehrbuchformel (4/pi)*arcsin(dn/sn): 0.3217)

Meep-Moden (k, f): (0.1, 0.0516), (0.2, 0.1028), (0.2, 0.4304), (0.3, 0.1528), (0.3, 0.3804), (0.4, 0.19
Zonenrand K·a/2π = 0,5:  f = 0.2238 und 0.3095  — die Klippenkanten!
```

```
# von oben: K_band, f_band, punkte, F_LO, F_HI, N1, N2, D1, D2
n_misch = np.sqrt(N1**2 * D1 + N2**2 * D2)   # gemitteltes Epsilon
K_gerade = np.linspace(0, 0.5, 50)

fig, ax = plt.subplots(figsize=(6.4, 4.6))
ax.plot(K_band, f_band, "C1.", ms=2.5, label="Spurformel  cos(Ka) = Tr/2")
ax.plot([p[0] for p in punkte], [p[1] for p in punkte], "C0o", ms=7,
        mfc="none", mew=1.6, label="Meep (k_point + Harminv)")
ax.plot(K_gerade, K_gerade / n_misch, "0.4", ls="--", lw=1.2,
        label=f"Gerade  f = K/(2π·n_eff),  n_eff = {n_misch:.2f}")
ax.axhspan(F_LO, F_HI, color="gold", alpha=0.35, label="Bandlücke")
ax.set_xlim(0, 0.52)
ax.set_ylim(0, 0.5)
ax.set_xlabel("Blochzahl  $K a / 2\\pi$")
ax.set_ylabel("Frequenz  $f\\,a/c$")
ax.set_title("$\\omega(K)$ mit Lücke: das Banddiagramm")
ax.legend(fontsize=8, loc="upper left")
ax.grid(alpha=0.3)
plt.show()
```

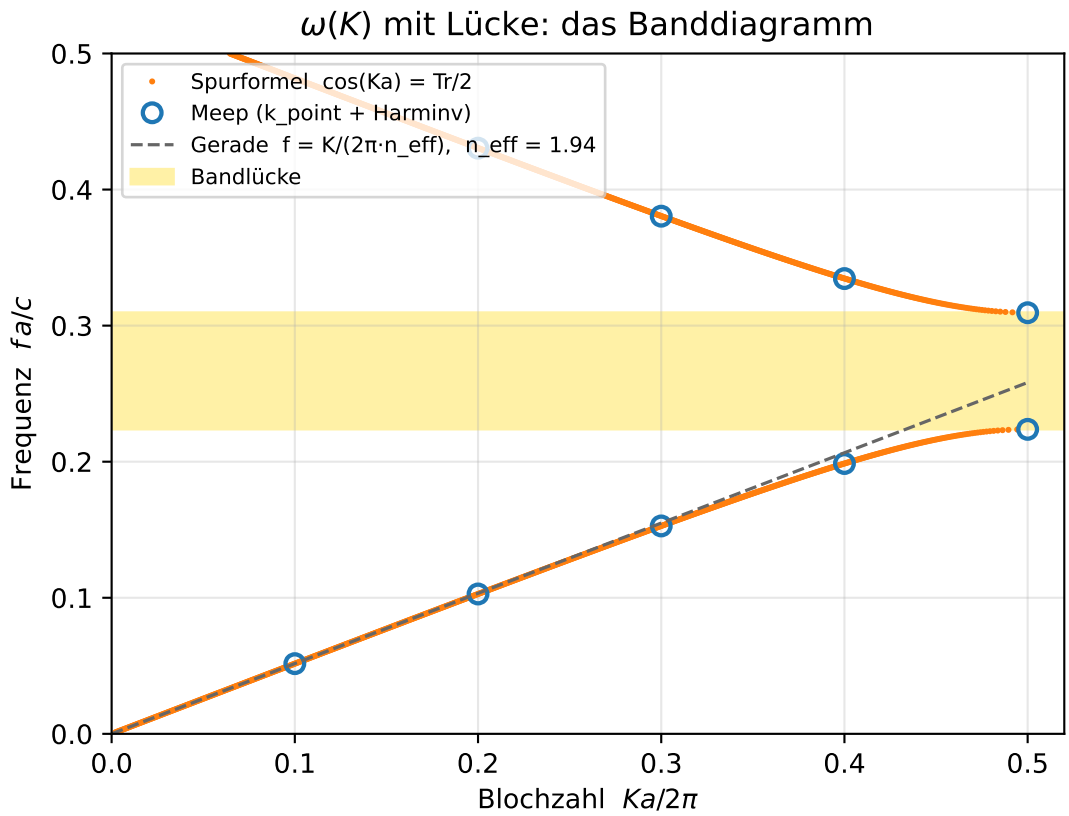


Abbildung 26.3: Das Banddiagramm des unendlichen Stapels: Die Spurformel (orange) reißt bei der Bandlücke (gold) auf, die Meep-Messpunkte (Kreise) sitzen auf den Bändern. Gestrichelt die Erwartung ohne Streuung: eine Gerade mit dem Misch-Index — unten folgt ihr das Band, am Zonenrand biegt es ab und spaltet auf. Genau wie die Gitterdispersion in Kapitel 8, nur dass dort das numerische Gitter der „Kristall“ war.

Lies das Diagramm wie die Dispersionsdiagramme aus Kapitel 16: Bei kleinen Frequenzen merkt die Welle von den Schichten nur den Durchschnitt — sie läuft mit dem Misch-Index auf der Geraden. Je näher das Band dem Zonenrand kommt, desto stärker arbeiten die Bragg-Echos dagegen: Das Band wird flach (Gruppengeschwindigkeit $\to 0$ — *slow light*, die stehende Welle am Rand), dann reißt die Kurve auf. In der Lücke gibt es schlicht **keinen Platz im Diagramm**: keine Frequenz, kein K, keine laufende Welle.

Und in der Lücke selbst? Da wird K komplex, und sein Imaginärteil ist messbar — er steht schon in unserer N-Reihe. Aus $|\cos(Ka)| > 1$ folgt $\mathrm{Im}(Ka) = \operatorname{arccosh}|\mathrm{Tr}/2|$, und die Leistung fällt pro Periode um $\mathrm{e}^{-2\,\mathrm{Im}(Ka)}$:

```
# von oben: NS, T_N, F0, spur_halbe()
im_Ka = np.arccosh(abs(spur_halbe(F0)))   # arccosh: Umkehrung von cosh
print(f"Im(Ka) = arccosh({abs(spur_halbe(F0)):.4f}) = {im_Ka:.4f}")
print(f"Leistungsfaktor pro Paar: e^(-2*{im_Ka:.4f}) = {np.exp(-2*im_Ka):.4f}")
print(f"gemessen (N-Reihe): {faktor_pro_paar:.4f}")

fig, ax = plt.subplots(figsize=(6.4, 3.8))
ax.semilogy(NS, T_N, "C0o", ms=7, label="Meep-Messung $T(f_0)$")
n_fein = np.linspace(2, 12, 50)
ax.semilogy(n_fein, T_N[0] * np.exp(-2 * im_Ka * (n_fein - NS[0])),
            "C1-", lw=1.5,
            label="Spurformel: Faktor $\\mathrm{e}^{-2\\,\\mathrm{Im}(Ka)}$ pro Paar")
ax.set_xlabel("Zahl der Schichtpaare $N$")
ax.set_ylabel("$T(f_0)$")
ax.set_title("Evaneszenter Abfall: Messung trifft Spurformel")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

```
Im(Ka) = arccosh(1.1333) = 0.5108
Leistungsfaktor pro Paar: e^(-2*0.5108) = 0.3600
gemessen (N-Reihe): 0.3675
```

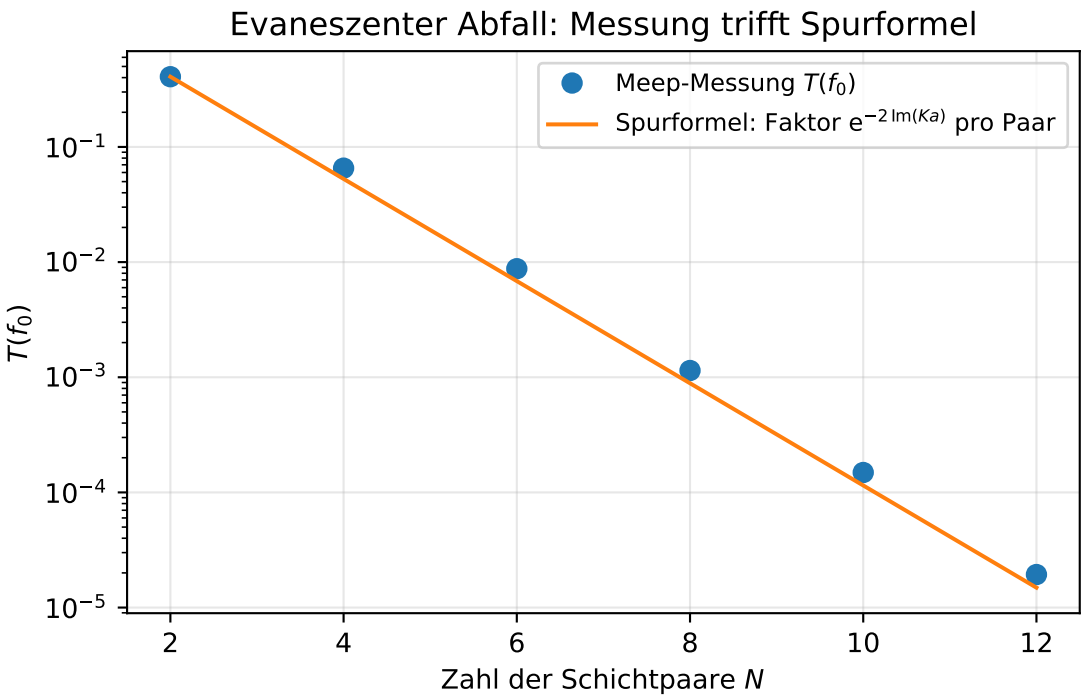


Abbildung 26.4: Die N-Reihe aus dem ersten Experiment (Punkte), daneben die parameterfreie Vorhersage der Spurformel (Linie): T fällt pro Schichtpaar um e^(−2·arccosh|Tr/2|) = 0,36. Die Stabilitätsmathematik aus Kapitel 8 sagt die Tunneldämpfung des Spiegels exakt voraus.

Damit ist der Kreis geschlossen: Klippe, exponentieller Abfall und Banddiagramm sind drei Gesichter derselben Zahl Tr $M/2$.

26.4 Vom Stapel zum Kristall

Der Stapel ist ein **eindimensionaler** photonischer Kristall — und er hat eine Schwachstelle, die jeder kennt, der einmal eine Seifenblase oder eben einen Morpho-Falter gedreht hat: Sein Sperrband gilt nur für *eine* Richtung. Bei schrägem Einfall verkürzt sich der wirksame Laufweg zwischen den Ebenen, die Bragg-Bedingung rutscht zu anderen Frequenzen — die Farbe *schillert*. Für einen Falter ist das eine Zierde. Wer aber Licht **einsperren** will (dazu gleich mehr), für den ist es ein Leck: Was die Wand senkrecht nicht passieren darf, nimmt eben den schrägen Ausgang.

Die Lösung: Periodik nicht in einer Richtung, sondern in allen. Unser Versuchsobjekt ist der Klassiker der 2D-Photonik — ein **Quadratgitter aus dielektrischen Stäbchen** (Aluminiumoxid, $\varepsilon = 8{,}9$, Radius $0{,}2\,a$) in Luft, Gitterkonstante $a = 1$. Die Polarisation passt von selbst: Unsere 2D-Welt rechnet seit Kapitel 9 mit E_z senkrecht zur Ebene, und genau für diese Polarisation hat das Stäbchengitter seine berühmte Lücke. Aus der Vogelperspektive von Kapitel 23 ist der Kristall übrigens ein alter Bekannter: ein **Array passiver Streuer** — Tausende winziger Antennen, deren Echos sich phasenrichtig sortieren, wie bei der Yagi-Direktorenkette in Kapitel 24. Nur dass hier niemand speist: Das Gitter antwortet nur.

Der Versuchsaufbau: **Frage:** Hat auch das Stäbchengitter ein Stoppband — und bei welchen Frequenzen? **Bühne:** acht Stabreihen quer zur Laufrichtung; quer dazu machen periodische Ränder (der `k_point=mp.Vector3()` aus Kapitel 16) das Gitter unendlich breit — wir simulieren eine einzige Gitterzeile. PML nur in Laufrichtung. **Anfangsbedingungen:** breiter Gauß-Puls als ebene Welle. **Messgrößen:** $T(f)$ gegen einen Leerlauf, wie beim Stapel. **Erfolgskriterium:** eine Klippe mit sauberem Boden ($T \ll 10^{-2}$); die Lage vergleichen wir anschließend mit dem Literaturwert.

```
# von oben: DPML, LUFT, FCEN/DF-Logik wie beim Stapel
EPS_STAB = 8.9          # Aluminiumoxid
R_STAB = 0.2            # Stabradius in Einheiten von a
FCEN2, DF2 = 0.35, 0.5
NFREQ2 = 401


def sweep_kristall(n_reihen=8, breite=1.0, dx=1.0, versetzt=False):
    """T(f) durch n_reihen Stabreihen; quer periodisch (Breite = Periode)."""
    tiefe = n_reihen * dx
    lx = 2 * DPML + 2 * LUFT + tiefe

    def geometrie(mit):
        if not mit:
            return []
        geom = []
        x0 = -tiefe / 2 + dx / 2
        for i in range(n_reihen):
            y0 = breite / 2 if (versetzt and i % 2 == 1) else 0.0
            geom.append(mp.Cylinder(radius=R_STAB,
                                    center=mp.Vector3(x0 + i * dx, y0),
                                    material=mp.Medium(epsilon=EPS_STAB)))
            if versetzt:
                # Periodenpartner, damit die versetzte Reihe am
```

```
                # y-Rand vollstaendig ist
                geom.append(mp.Cylinder(radius=R_STAB,
                                        center=mp.Vector3(x0 + i * dx,
                                                          y0 - breite),
                                        material=mp.Medium(epsilon=EPS_STAB)))
        return geom

    laeufe = []
    for mit in (False, True):
        sim = mp.Simulation(
            cell_size=mp.Vector3(lx, breite), resolution=24,
            boundary_layers=[mp.PML(DPML, direction=mp.X)],
            geometry=geometrie(mit),
            k_point=mp.Vector3(),        # periodische Raender in y
            sources=[mp.Source(mp.GaussianSource(FCEN2, fwidth=DF2),
                               component=mp.Ez,
                               center=mp.Vector3(-lx / 2 + DPML + 0.5, 0),
                               size=mp.Vector3(0, breite))])
        x_mess = lx / 2 - DPML - 0.5
        tr = sim.add_flux(FCEN2, DF2, NFREQ2,
                          mp.FluxRegion(center=mp.Vector3(x_mess, 0),
                                        size=mp.Vector3(0, breite)))
        sim.run(until_after_sources=mp.stop_when_fields_decayed(
            50, mp.Ez, mp.Vector3(x_mess, 0), 1e-8))
        laeufe.append((np.array(mp.get_flux_freqs(tr)),
                       np.array(mp.get_fluxes(tr))))
    (f2, p_ref), (_, p_kr) = laeufe
    return f2, p_kr / p_ref


fs_x, T_x = sweep_kristall()


def stoppband(f_arr, T_arr, schwelle=1e-2):
    drin = (f_arr > 0.15) & (f_arr < 0.55) & (T_arr < schwelle)
    return f_arr[drin][0], f_arr[drin][-1]


sb_x = stoppband(fs_x, T_x)
print(f"Stoppband senkrecht zu den Reihen: {sb_x[0]:.3f} .. {sb_x[1]:.3f}")
print(f"T am Boden: {T_x[(fs_x > 0.30) & (fs_x < 0.42)].min():.1e}")
```

```
Stoppband senkrecht zu den Reihen: 0.277 .. 0.440
```

```
T am Boden: 3.8e-06
```

Eine prächtige Klippe — Stoppband von 0,28 bis 0,44. Aber Vorsicht: Genau hier lauert der Denkfehler des 1D-Stapels.

Vorhersage-Punkt (PRIMM): Wir wiederholen die Messung **diagonal** — die Welle läuft jetzt entlang der Gitterdiagonale auf dieselben Stäbe zu. Bekommt sie dasselbe Stoppband zu spüren? Bevor du „klar, gleiches Gitter" sagst: Der Abstand der Stabreihen, die die Welle nacheinander quert, ist diagonal nicht mehr a, sondern $a/\sqrt{2}$ — und die Bragg-Frequenz hängt am Reihenabstand …

```
# von oben: sweep_kristall(), stoppband()
# Diagonale = um 45 Grad gedrehtes Gitter: Querperiode sqrt(2)·a,
# Reihenabstand sqrt(2)/2·a, jede zweite Reihe um die halbe
# Querperiode versetzt. 12 Reihen ≈ gleiche Kristalltiefe.
fs_m, T_m = sweep_kristall(n_reihen=12, breite=np.sqrt(2.0),
                           dx=np.sqrt(2.0) / 2, versetzt=True)
sb_m = stoppband(fs_m, T_m)
print(f"Stoppband diagonal: {sb_m[0]:.3f} .. {sb_m[1]:.3f}")

luecke = (max(sb_x[0], sb_m[0]), min(sb_x[1], sb_m[1]))
print(f"Schnittmenge beider Richtungen: {luecke[0]:.3f} .. {luecke[1]:.3f}")
print("Literatur (Joannopoulos et al.): 0.32 .. 0.44")
```

```
Stoppband diagonal: 0.323 .. 0.545
Schnittmenge beider Richtungen: 0.323 .. 0.440
Literatur (Joannopoulos et al.): 0.32 .. 0.44
```

```
# von oben: fs_x, T_x, fs_m, T_m, luecke
fig, ax = plt.subplots(figsize=(7.0, 4.2))
ax.semilogy(fs_x, np.maximum(T_x, 1e-14), "C0-", lw=1.4,
            label="senkrecht (Reihenabstand $a$)")
ax.semilogy(fs_m, np.maximum(T_m, 1e-14), "C3-", lw=1.4,
            label="diagonal (Reihenabstand $a/\\sqrt{2}$)")
ax.axvspan(luecke[0], luecke[1], color="gold", alpha=0.4,
           label="volle Bandlücke (Schnittmenge)")
ax.axvline(0.32, color="0.4", ls="--", lw=1)
ax.axvline(0.44, color="0.4", ls="--", lw=1)
ax.set_xlim(0.15, 0.55)
ax.set_ylim(1e-13, 3)
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$")
```

```
ax.set_title("Erst die Schnittmenge macht den Kristall")
ax.legend(fontsize=8, loc="lower left")
ax.grid(alpha=0.3)
plt.show()
```

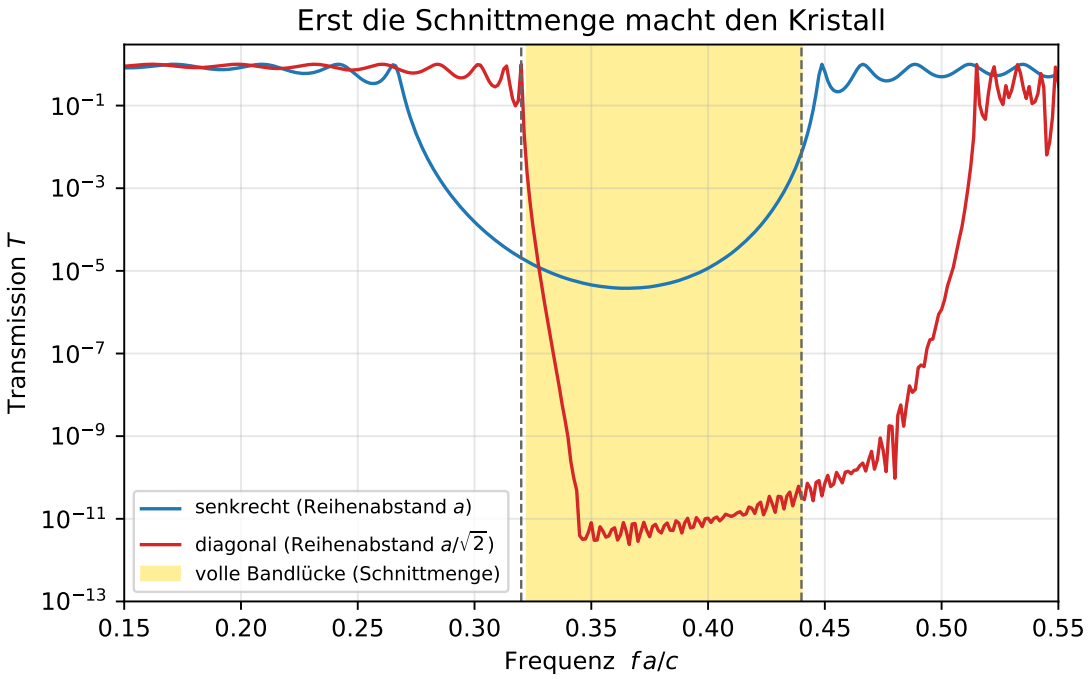


Abbildung 26.5: Zwei Einfallsrichtungen, zwei verschiedene Stoppbänder: senkrecht (blau) sperrt ab 0,28, diagonal (rot) erst ab 0,32 — dafür bis weit über 0,5 hinaus. Nur die Schnittmenge (gold) ist in beiden Richtungen verboten: die echte photonische Bandlücke, 0,32 bis 0,44 — auf der Literaturlage (gestrichelte Kanten).

Die beiden Richtungen sperren verschieden — wie vom Reihenabstand-Argument vorhergesagt beginnt das diagonale Stoppband später (kürzerer Abstand → höhere Bragg-Frequenz). Aber sie **überlappen**: Zwischen 0,32 und 0,44 kommt die Welle weder geradeaus noch schräg durch — und eine genauere Analyse über *alle* Richtungen (das Kleingedruckte) bestätigt, dass dieses Fenster komplett dicht ist. Das ist der entscheidende Unterschied zwischen einem Spiegel und einem Kristall: Eine **vollständige photonische Bandlücke** ist ein Frequenzbereich, in dem es im Inneren des Materials *überhaupt keine* laufende Lichtwelle gibt. Licht dieser Frequenzen kann im Kristall schlicht nicht existieren — egal woher es

kommt.

26.5 Der Defekt: Licht um die Ecke

Wozu ist ein Material gut, in dem Licht nicht existieren kann? Zum **Einsperren** — und das ist die eigentliche Pointe des Kapitels. Bohren wir in den perfekten Kristall einen Fehler: Wir nehmen eine komplette Stabreihe heraus. In dieser Gasse ist Platz für Licht (sie ist ja leer), aber ihre Wände sind für Lückenfrequenzen undurchdringlich — und zwar nicht wegen Reflexion an einer Grenzfläche, die bei schrägem Auftreffen schwächelt, sondern weil es **im Wandmaterial keine Zustände gibt**. Das Licht *muss* in der Gasse bleiben. Ein Wellenleiter, dessen Wände aus einem Verbot bestehen.

Der härteste Test für dieses Prinzip ist eine Kurve, an der jeder klassische Leiter scheitert: die scharfe **90°-Ecke**. Eine Glasfaser (Totalreflexion, Kapitel 12) verliert ihr Licht schon bei sanften Biegungen — der Auftreffwinkel rutscht unter den Grenzwinkel. Hier kann das nicht passieren: Es gibt keinen Winkel, unter dem die Wand durchlässig würde.

Der Versuchsaufbau: **Frage:** Wie viel Leistung übersteht die scharfe Ecke? **Bühne:** ein endlicher Kristall aus 17 × 17 Stäben; einmal fehlt die mittlere Zeile (gerader Leiter, Referenz), einmal fehlen die linke halbe Zeile und die obere halbe Spalte (L-förmiger Leiter mit 90°-Knick). **Anfangsbedingungen:** ein Gauß-Puls bei $f = 0{,}38$ (mitten in der Lücke), gestartet *im* Leiter nahe dem linken Rand. **Messgrößen:** der Leistungsfluss kurz vor dem Leiterende — beim geraden Leiter rechts, beim Knick oben; beide Wege sind gleich lang, das Verhältnis $T_{\text{Knick}}/T_{\text{gerade}}$ isoliert den Preis der Ecke. **Erfolgskriterium:** ehrlich offen — klassische Intuition sagt „eine scharfe Ecke kostet fast alles".

Vorhersage-Punkt (PRIMM): Dein Tipp für $T_{\text{Knick}}/T_{\text{gerade}}$ bei $f = 0{,}38$: eher 1 %, 10 % oder 90 %?

```
# von oben: EPS_STAB, R_STAB, DPML
NK = 8                    # Staebe von -NK..NK -> 17x17
L_KRIST = 2 * NK + 1
LX_WL = L_KRIST + 2 * 2.0 + 2 * DPML
F_LEIT, DF_LEIT = 0.38, 0.10


def staebe(form):
    """Kristall mit Leiter-Gasse: 'gerade' oder 'knick' (L-Form)."""
    geom = []
```

```
    for i in range(-NK, NK + 1):
        for j in range(-NK, NK + 1):
            if form == "gerade" and j == 0:
                continue
            if form == "knick" and ((j == 0 and i <= 0) or
                                    (i == 0 and j >= 0)):
                continue
            geom.append(mp.Cylinder(radius=R_STAB, center=mp.Vector3(i, j),
                                    material=mp.Medium(epsilon=EPS_STAB)))
    return geom


def lauf_leiter(form):
    sim = mp.Simulation(
        cell_size=mp.Vector3(LX_WL, LX_WL), resolution=16,
        boundary_layers=[mp.PML(DPML)], geometry=staebe(form),
        sources=[mp.Source(mp.GaussianSource(F_LEIT, fwidth=DF_LEIT),
                           component=mp.Ez,
                           center=mp.Vector3(-NK + 0.5, 0))])
    if form == "gerade":
        fr = mp.FluxRegion(center=mp.Vector3(NK - 0.5, 0),
                           size=mp.Vector3(0, 1))
        sonde = mp.Vector3(NK - 0.5, 0)
    else:
        fr = mp.FluxRegion(center=mp.Vector3(0, NK - 0.5),
                           size=mp.Vector3(1, 0))
        sonde = mp.Vector3(0, NK - 0.5)
    tr = sim.add_flux(F_LEIT, DF_LEIT, 201, fr)
    sim.run(until_after_sources=mp.stop_when_fields_decayed(
        50, mp.Ez, sonde, 1e-7))
    return np.array(mp.get_flux_freqs(tr)), np.array(mp.get_fluxes(tr))


fs_l, P_gerade = lauf_leiter("gerade")
_, P_knick = lauf_leiter("knick")
T_ecke = P_knick / P_gerade
print(f"T_Knick / T_gerade bei f = 0,38:  "
      f"{T_ecke[np.argmin(np.abs(fs_l - 0.38))]:.3f}")
```

```
T_Knick / T_gerade bei f = 0,38:  1.010
```

Die Ecke kostet nichts. Das Verhältnis liegt bei eins (die Restwelligkeit kommt von Echos an den Leiter-Enden, Fabry-Pérot lässt grüßen — Kapitel

27 nimmt sich genau solche Resonanzen vor). Zum Vergleich die klassische Welt: Ein scharfer 90°-Knick in einer Glasfaser wirft praktisch alles weg. So sieht der Sieg über die Geometrie aus:

```
# von oben: staebe(), LX_WL, L_KRIST, F_LEIT, NK
sim_cw = mp.Simulation(
    cell_size=mp.Vector3(LX_WL, LX_WL), resolution=16,
    boundary_layers=[mp.PML(DPML)], geometry=staebe("knick"),
    force_complex_fields=True,
    sources=[mp.Source(mp.ContinuousSource(F_LEIT),
                       component=mp.Ez, center=mp.Vector3(-NK + 0.5, 0))])
sim_cw.run(until=150)
ez_knick = sim_cw.get_array(component=mp.Ez, center=mp.Vector3(),
                            size=mp.Vector3(L_KRIST + 2, L_KRIST + 2))
print(f"Gesundheitswert: max|Ez| = {np.abs(ez_knick).max():.2f}")

seite = L_KRIST + 2
betrag = np.abs(ez_knick).T   # Transponieren: Bildzeilen = y
fig, ax = plt.subplots(figsize=(6.6, 6.2))
ax.imshow(betrag, origin="lower", cmap="inferno",
          extent=[-seite / 2, seite / 2, -seite / 2, seite / 2],
          vmax=np.percentile(betrag, 99.5))
for i in range(-NK, NK + 1):
    for j in range(-NK, NK + 1):
        if (j == 0 and i <= 0) or (i == 0 and j >= 0):
            continue
        ax.add_patch(plt.Circle((i, j), R_STAB, fill=False,
                                color="w", lw=0.4, alpha=0.55))
ax.set_xlabel("x/a")
ax.set_ylabel("y/a")
ax.set_title("Licht fährt um die Ecke")
plt.show()
```

```
Gesundheitswert: max|Ez| = 1.28
```

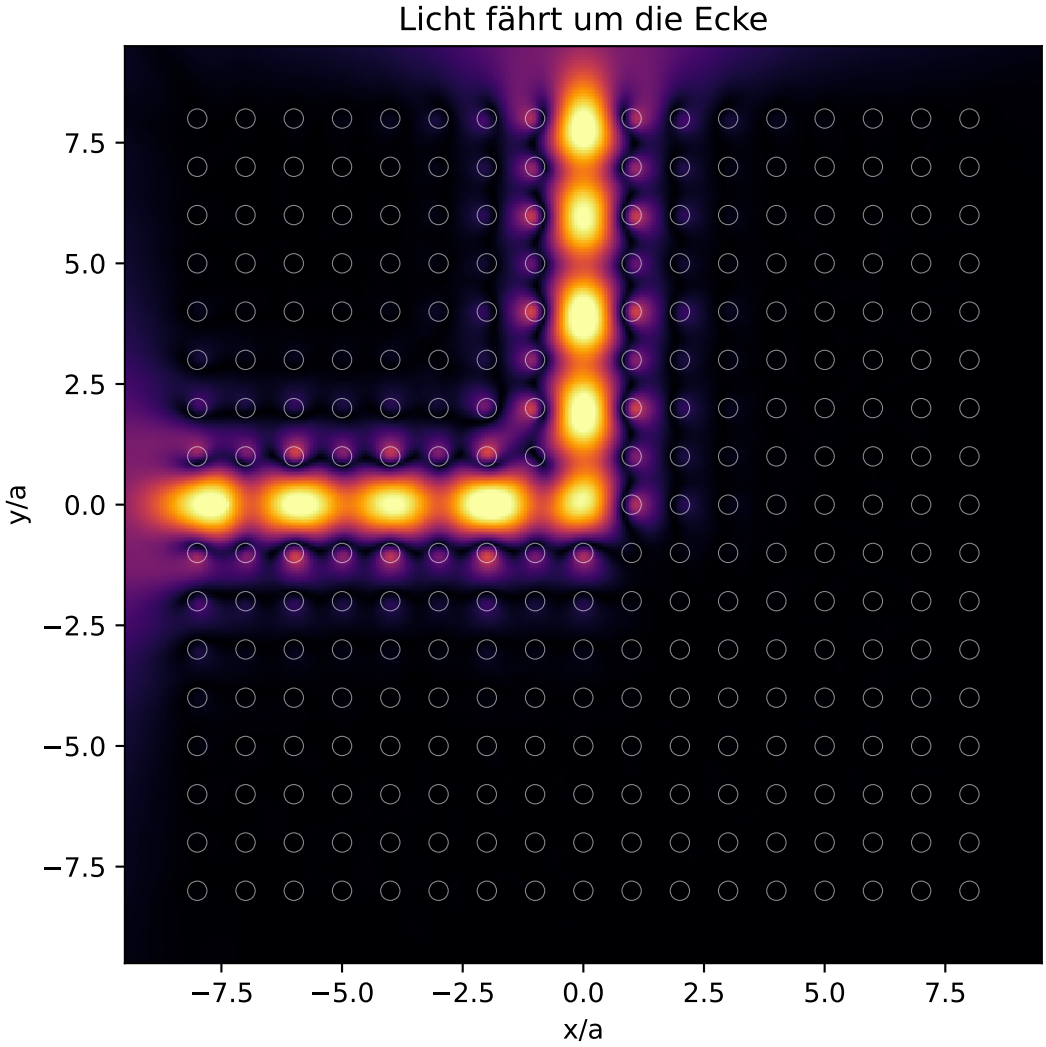


Abbildung 26.6: |E_z| im eingeschwungenen Zustand (Dauerstrich f = 0,38, helle Farbe = viel Feld; Kreise = Stäbe): Das Licht läuft von links in der Gasse, biegt scharf nach oben ab und kommt oben an — die Erwartung „T ≈ 1“ aus der Flussmessung, hier als Bild. In den Kristall dringt es nur ein bis zwei Reihen weit ein (evaneszent, wie im Stapel), und vor dem Knick steht die Stehwellen-Riffelung des leichten Echos.

Solche Knicke sind der Grund, warum photonische Kristalle als „Leiterplatten für Licht” gehandelt werden: Auf einem Chip lassen sich Lichtwege damit auf Wellenlängen-Skala falten — wo eine Glasfaser Zentimeter an Biegeradius bräuchte.

26.6 Strukturfarbe: der Falter, nachgerechnet

Zurück zum Anfang — jetzt mit Werkzeug. Die Lamellen des Morpho-Falters sind ein Bragg-Stapel aus Chitin ($n \approx 1{,}56$) und Luft, beide Schichten etwa 90 nm dick. Welche Farbe wirft er zurück? Die Bragg-Bedingung (alle Echos in Phase: ein Umlauf durch eine Periode = eine ganze Wellenlänge) liefert die Mitte des Sperrbands: $\lambda_{\max} = 2(n_1 d_1 + n_2 d_2)$. Beim Schrägblick unter dem Winkel θ zählt nur noch die Wegkomponente senkrecht zum Stapel — in jeder Schicht ersetzt $d_j \cos\theta_j$ (mit Snellius für θ_j) die volle Dicke, und $\lambda_{\max}$ rutscht nach **blau-violett**:

```
# von oben: schichtmatrix(), kette_rt()
N_CHITIN, D_CHITIN = 1.56, 90e-9
N_LUFT, D_LUFT = 1.0, 90e-9
N_LAMELLEN = 8


def morpho_R(lam, theta_grad):
    """Reflexion des Chitin-Luft-Stapels bei Blickwinkel theta (s-Pol.)."""
    th = np.radians(theta_grad)
    # Snellius in jeder Schicht; bei s-Polarisation geht n*cos(theta_j)
    # als effektiver Index in die Schichtmatrix ein
    c_ch = np.sqrt(1 - (np.sin(th) / N_CHITIN) ** 2)
    c_lu = np.sqrt(1 - np.sin(th) ** 2)
    f = 1.0 / lam                      # c = 1: Frequenz in 1/m
    mats = []
    for _ in range(N_LAMELLEN):
        mats.append(schichtmatrix(N_CHITIN * c_ch, D_CHITIN, f))
        mats.append(schichtmatrix(N_LUFT * c_lu, D_LUFT, f))
    # Anschluss an Luft ebenfalls unter dem Winkel
    M = np.eye(2, dtype=complex)
    for m in mats:
        M = M @ m
    p = c_lu                           # Luft-Halbraum, s-Polarisation
    nenner = (M[0, 0] + M[0, 1] * p) * p + (M[1, 0] + M[1, 1] * p)
    r = ((M[0, 0] + M[0, 1] * p) * p - (M[1, 0] + M[1, 1] * p)) / nenner
    return np.abs(r) ** 2


lams = np.linspace(300e-9, 700e-9, 400)
fig, ax = plt.subplots(figsize=(7.0, 4.0))
for th, farbe in [(0, "C0"), (45, "C4")]:
    R_m = np.array([morpho_R(l, th) for l in lams])
```

```
    ax.plot(lams * 1e9, R_m, farbe, lw=1.6, label=f"Blickwinkel {th}°")
    th_r = np.radians(th)
    lam_bragg = 2 * (N_CHITIN * D_CHITIN
                     * np.sqrt(1 - (np.sin(th_r) / N_CHITIN) ** 2)
                     + D_LUFT * np.sqrt(1 - np.sin(th_r) ** 2))
    ax.axvline(lam_bragg * 1e9, color=farbe, ls="--", lw=1.1)
ax.axvspan(380, 490, color="royalblue", alpha=0.10)
ax.text(437, 1.04, "blau", color="royalblue", ha="center", fontsize=9)
ax.set_xlabel("Wellenlänge in nm")
ax.set_ylabel("Reflexion $R$")
ax.set_ylim(0, 1.12)
ax.set_title("Morpho-Blau: Farbe aus Geometrie")
ax.legend(fontsize=9, loc="center right")
ax.grid(alpha=0.3)
plt.show()
```

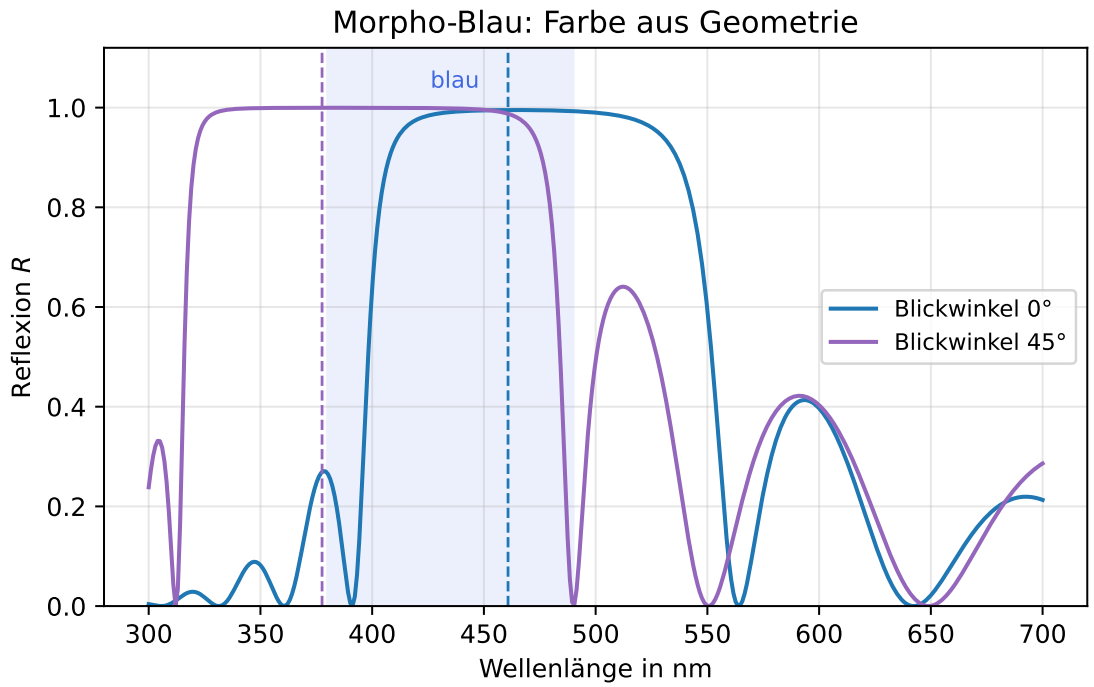


Abbildung 26.7: Das Reflexionsspektrum des Morpho-Modells (Transfer-Matrix, 8 Lamellen Chitin/Luft à 90 nm): senkrecht betrachtet liegt das Maximum um 460 nm (sattes Blau), bei 45°-Blick wandert es auf etwa 380 nm Richtung Violett — das Schillern. Senkrechte Striche: die Bragg-Handformel 2·Σ n·d·cosθ (461 bzw. 378 nm) sitzt jeweils im Sperrband-Maximum.

Kein Pigment, keine Absorption — die Farbe entsteht, weil ein Frequenzband nicht *hinein* darf und deshalb zurückkommt. Dasselbe Prinzip färbt Opale (dort sind es gestapelte Silikat-Kügelchen — ein 3D-Kristall, dessen Lücke je nach Blickrichtung andere Farben trifft: das Opalfeuer), Pfauenfedern und manche Käferpanzer. Und am anderen Ende der Präzisionsskala: Die Endspiegel von Laser-Resonatoren sind λ/4-Stapel mit $R > 99{,}999\,\%$ — Werte, an die kein Metallspiegel herankommt, denn Metall *absorbiert* immer ein paar Prozent (Kapitel 16), während im Dielektrikum nichts verloren geht, sondern nur umgelenkt wird. Auch **photonische Glasfasern** gehören in diese Familie: ein Kristall aus Luftlöchern, der das Licht in einem **hohlen** Kern führt — Lichtleitung in Luft, wo es weder Materialdämpfung noch nennenswerte Dispersion gibt; mit Totalreflexion (Kapitel 12) wäre das unmöglich, denn die braucht innen das *dichtere* Medium.

Bleibt der Name: Warum „Kristall"? Weil die Mathematik geliehen ist. Elek-

tronen in einem Halbleiter laufen als Quantenwellen (Kapitel 30) durch das periodische Potential der Atomrümpfe — und bekommen aus exakt derselben Bloch-Analyse ihre **elektronische Bandlücke**: den Energiebereich ohne erlaubte Zustände, der Halbleiter von Metallen unterscheidet und auf dem die gesamte Elektronik steht. Die Photonik hat sich diese fünfzig Jahre ältere Theorie ausgeliehen und „Schrödinger" durch „Maxwell" ersetzt; der photonische Kristall ist der Halbleiter des Lichts. Was dort Dotierung und Störstellen sind, waren hier unsere fehlenden Stäbe.

26.7 Das Kapitel-Programm

`programme/kap26/kap26_photonische_kristalle.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die Transfer-Matrix-Eigenschaften ($R + T = 1$ auf Maschinengenauigkeit, Lückenbreite auf der arcsin-Formel, $\mathrm{Tr}/2$ bei f_0 exakt $-(n_1/n_2 + n_2/n_1)/2$), den Meep-1D-Sweep auf der TMM-Linie (rms < 0,02), die N-Reihe auf der $\mathrm{e}^{-2\,\mathrm{Im}(Ka)}$-Geraden, das Banddiagramm (Meep-Punkte auf dem TMM-Band, Abweichung $< 10^{-3}$; Zonenrand = Klippenkanten), beide 2D-Sweeps samt Schnittmengen-Lücke 0,32–0,44, den Knick ($T \approx 1$), die Punktdefekt-Kavität (kommt in Übung 26.4) und die Morpho-Zahlen. Läuft in wenigen Minuten.

Damit ist der Werkzeugkasten des Kapitels beisammen — Zeit für das Fazit.

Merkkasten

- **λ/4-Stapel:** Vorzeichenwechsel der Reflexion plus λ/2-Umweg bringen *alle* Echos in Phase — acht Glas/Titandioxid-Paare reflektieren 99,9 %. Durchsichtig ist keine Materialeigenschaft, sondern eine Frage der Anordnung.
- **Transfer-Matrix:** eine 2×2-Matrix pro Schicht, der Stapel ist ihr Produkt — alle Mehrfach-Echos inklusive. Exakt, parameterfrei, Millisekunden.
- **Spurkriterium:** $\cos(Ka) = \mathrm{Tr}\,M/2$. Bei $|\mathrm{Tr}| \leq 2$ läuft die Blochwelle (Band), bei $|\mathrm{Tr}| > 2$ klingt sie ab (Lücke) — dieselbe Mathematik wie die CFL-Stabilität aus Kapitel 8, nur im Raum statt in der Zeit. Abfallrate: $\mathrm{Im}(Ka) = \mathrm{arccosh}\,|\mathrm{Tr}/2|$.
- **Bandlücke = Schnittmenge:** Ein Stapel sperrt nur senkrecht (deshalb schillert der Falter). Erst wenn die Stoppbänder *aller* Richtungen überlappen, ist es eine vollständige photonische Bandlücke (Stäbchengitter: 0,32–0,44 bei $\varepsilon = 8{,}9$, $r = 0{,}2\,a$).
- **Defekte sind das Werkzeug:** Eine fehlende Stabreihe führt Licht

um die scharfe 90°-Ecke ($T \approx 1$), denn ihre Wände haben keine Zustände, in die etwas lecken könnte; ein fehlender Einzelstab sperrt es ein.

Roter Faden

Dieses Kapitel hat viele alte Fäden zu einem neuen Stoff verwebt: die **Grenzflächen-Reflexion aus Kapitel 6** und den **λ/4-Trafo aus Kapitel 22** (vom Echo-Löscher zum Echo-Sammler), die **Interferenz-Buchhaltung aus Kapitel 14** (dort haben wir versprochen, dass die Gitterformel wiederkommt — als Bragg-Reflexion ganzer Kristallebenen), das **Spurkriterium aus Kapitel 8** (von der Zeit in den Raum gewendet), **k-Punkt und Dispersionsdiagramm aus Kapitel 16**, das **Harminv-Ohr aus Kapitel 15**, die **evaneszenten Felder aus Kapitel 12/17** (in der Lücke wohnt dieselbe Feldsorte wie hinter der Totalreflexion — und wie im Pendelfeld von Kapitel 25) und die **Streuer-Arrays aus Kapitel 23/24** (der Kristall als Antennengruppe, die niemand speist). Nach vorn: **Kapitel 27** holt sich die Defektidee zurück und baut aus Spiegeln Resonatoren — Fabry-Pérot, Ringresonator und der Schritt zum Laser; die Transmissionsnadel aus Übung 26.3 ist davon der Prolog.

Übungen

Ü 26.1 — Das Grün des Käfers (Verstehen). Ein Prachtkäfer erzeugt sein Grün ($\lambda \approx 530$ nm) mit demselben Chitin/Luft-Stapel wie der Morpho ($n = 1{,}56$, gleich dicke Schichten). (a) Wie dick müssen seine Lamellen sein? (b) Ein Regentropfen benetzt den Panzer und ersetzt die Luft in den Spalten durch Wasser ($n = 1{,}33$). Was passiert mit der Farbe — Lage *und* Stärke? Erst überlegen (zwei Effekte!), dann rechnen.

ℹ Musterlösung zu Ü 26.1

(a) Aus $\lambda_{\text{max}} = 2(n_1 d + n_2 d)$ folgt $d = \lambda_{\text{max}}/(2(n_1 + n_2))$ — die Bragg-Bedingung nach d aufgelöst:

```
lam_gruen = 530e-9
d_kaefer = lam_gruen / (2 * (1.56 + 1.0))
print(f"Lamellendicke: {d_kaefer*1e9:.0f} nm  (Morpho: 90 nm)")
```

```
Lamellendicke: 104 nm  (Morpho: 90 nm)
```

Grün braucht etwa 104 nm — nur 14 nm mehr als das Morpho-Blau. So fein arbeitet die Evolution an diesem Regler.

(b) Zwei Effekte, beide gegen die Farbe: Die optische Dicke der Spalten steigt (n: 1,0 → 1,33), also rutscht $\lambda_{\text{max}} = 2d(n_1 + n_2)$ **nach rot**; und der Kontrast n_2/n_1 fällt von 1,56 auf 1,17, also wird die Lücke schmal und die Reflexion schwach (das arcsin-Argument schrumpft von 0,22 auf 0,08):

```python
lam_nass = 2 * d_kaefer * (1.56 + 1.33)
breite_trocken = (4 / np.pi) * np.arcsin((1.56 - 1.0) / (1.56 + 1.0))
breite_nass = (4 / np.pi) * np.arcsin((1.56 - 1.33) / (1.56 + 1.33))
print(f"nass: lambda_max = {lam_nass*1e9:.0f} nm (statt 530)")
print(f"relative Lueckenbreite: {breite_trocken:.3f} -> {breite_nass:.3f}")
```

```
nass: lambda_max = 598 nm (statt 530)
relative Lueckenbreite: 0.281 -> 0.101
```

Der nasse Käfer wird rötlich-fahl — und genau das beobachtet man: Strukturfarben *erlöschen* bei Benetzung und kommen beim Trocknen wieder. Ein Pigment könnte das nicht; es ist der schnellste Feldtest auf Strukturfarbe.

Ü 26.2 — Der Kontrast-Regler (Verändern). Miss mit der Transfer-Matrix die relative Lückenbreite für $n_2 = 1{,}7$ bis $4{,}0$ (bei $n_1 = 1{,}5$, jeweils λ/4-Dicken) und lege die Formel $(4/\pi)\arcsin\big((n_2 - n_1)/(n_2 + n_1)\big)$ darüber. Wie groß muss der Kontrast sein, damit die Lücke relativ breiter ist als die einer Oktave (Faktor 2 in der Frequenz, also Breite/Mitte = 2/3)?

Musterlösung zu Ü 26.2

```
# von oben: schichtmatrix(), N1


def lueckenbreite(n2):
    lam0 = 4.0 / (1.0 / N1 + 1.0 / n2)
    f0 = 1.0 / lam0
    d1 = lam0 / (4.0 * N1)
    d2 = lam0 / (4.0 * n2)

    def tr2(f):
        M = schichtmatrix(N1, d1, f) @ schichtmatrix(n2, d2, f)
        return abs(float(np.real(M[0, 0] + M[1, 1]) / 2))

    f_raster = np.linspace(0.5 * f0, 1.5 * f0, 2001)
    drin = np.array([tr2(f) > 1 for f in f_raster])
    return (f_raster[drin][-1] - f_raster[drin][0]) / f0


n2_reihe = np.linspace(1.7, 4.0, 12)
gemessen = np.array([lueckenbreite(n2) for n2 in n2_reihe])
formel = (4 / np.pi) * np.arcsin((n2_reihe - N1) / (n2_reihe + N1))

fig, ax = plt.subplots(figsize=(6.4, 3.8))
ax.plot(n2_reihe / N1, gemessen, "C0o", ms=6, label="Transfer-Matrix")
ax.plot(n2_reihe / N1, formel, "C1-", lw=1.5, label="arcsin-Formel")
ax.axhline(2 / 3, color="0.4", ls="--", lw=1, label="eine Oktave (2/3)")
ax.set_xlabel("Kontrast $n_2 / n_1$")
ax.set_ylabel("Lückenbreite / Mittenfrequenz")
ax.set_title("Die Lücke wächst mit dem Kontrast")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()

n2_oktave = N1 * (1 + np.sin(np.pi / 6)) / (1 - np.sin(np.pi / 6))
print(f"Oktave: arcsin-Argument = sin(pi/6) = 0.5 "
      f"-> n2 = 3*n1 = {n2_oktave:.1f}")
```

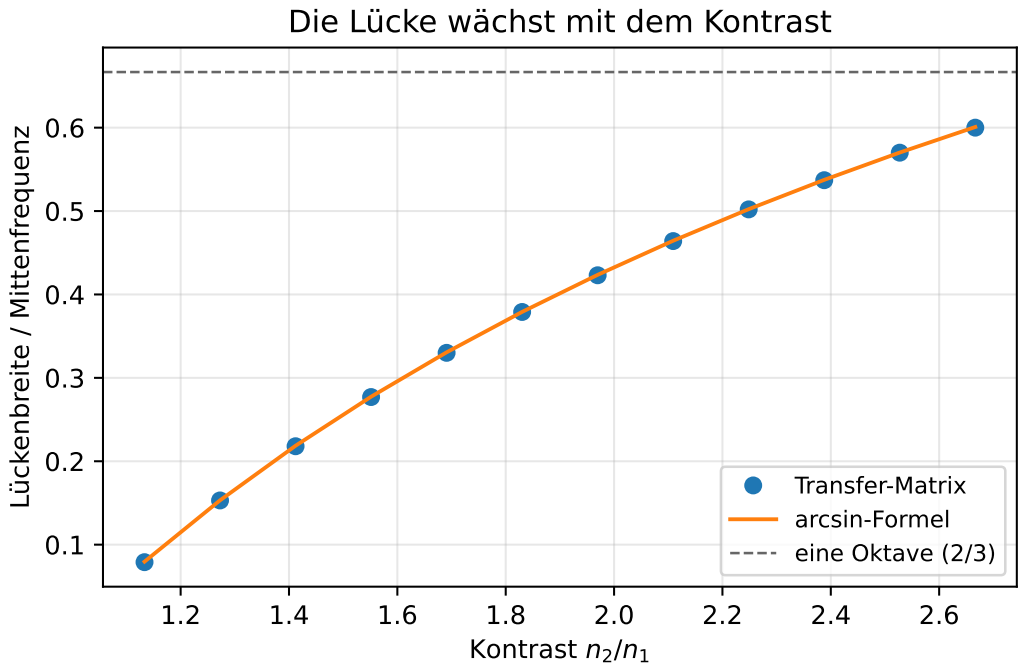


Abbildung 26.8: Gemessene Lückenbreite (Punkte) auf der arcsin-Formel (Linie): Der Kontrast ist der einzige Regler — und für eine Oktave Sperrbreite braucht es $n_2/n_1 \approx 5$, mehr als jedes gängige Schichtpaar hergibt.

```
Oktave: arcsin-Argument = sin(pi/6) = 0.5 -> n2 = 3*n1 = 4.5
```

Messung und Formel decken sich. Für die Oktave muss $(n_2 - n_1)/(n_2 + n_1) = \sin(\pi/6) = 1/2$ sein, also $n_2 = 3n_1 = 4{,}5$ — jenseits gängiger transparenter Materialien. Breitband-Verspiegelung löst man deshalb anders: mehrere Stapel mit gestaffelten f_0 hintereinander (Chirp).

Ü 26.3 — Die Nadel im Spiegel (Verändern). Baue mit der Transfer-Matrix eine **Defektschicht** in den Stapel: vier λ/4-Paare, dann eine n_1-Schicht **doppelter** Dicke (eine λ/2-Schicht), dann vier Paare gespiegelt. Berechne $T(f)$ innerhalb der Lücke. Vorhersage: Die Lücke war eben noch hermetisch dicht — was kann eine einzige falsche Schicht schon ausrichten? Miss dann nach: Lage, Höhe und Breite (FWHM → $Q = f/\Delta f$, Kapitel 15) der Überraschung; wiederhole mit fünf und sechs Paaren pro Seite.

Musterlösung zu Ü 26.3

```
# von oben: schichtmatrix(), kette_rt(), N1, N2, D1, D2, F0, F_LO, F_HI


def kavitaet_T(f, n_spiegel):
    mats = []
    for _ in range(n_spiegel):                      # linker Spiegel
        mats += [schichtmatrix(N1, D1, f), schichtmatrix(N2, D2, f)]
    mats += [schichtmatrix(N1, 2 * D1, f)]          # lambda/2-Defekt
    for _ in range(n_spiegel):                      # rechter Spiegel
        mats += [schichtmatrix(N2, D2, f), schichtmatrix(N1, D1, f)]
    return np.abs(kette_rt(mats)[1]) ** 2


f_nadel = np.linspace(F_LO, F_HI, 8001)
fig, ax = plt.subplots(figsize=(7.0, 4.0))
for n_sp, farbe in [(4, "C0"), (5, "C1"), (6, "C3")]:
    T_kav = np.array([kavitaet_T(f, n_sp) for f in f_nadel])
    i_pk = np.argmax(T_kav)
    ueber = T_kav > T_kav[i_pk] / 2
    fwhm = f_nadel[ueber][-1] - f_nadel[ueber][0]
    ax.semilogy(f_nadel, T_kav, farbe, lw=1.3,
                label=f"{n_sp} Paare/Seite:  T = {T_kav[i_pk]:.3f}, "
                      f"Q = {f_nadel[i_pk]/fwhm:.0f}")
ax.axvline(F0, color="0.4", ls="--", lw=1)
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$")
ax.set_ylim(1e-6, 3)
ax.set_title("Eine λ/2-Schicht bohrt ein Loch in die Lücke")
ax.legend(fontsize=9, loc="lower right")
ax.grid(alpha=0.3)
plt.show()
```

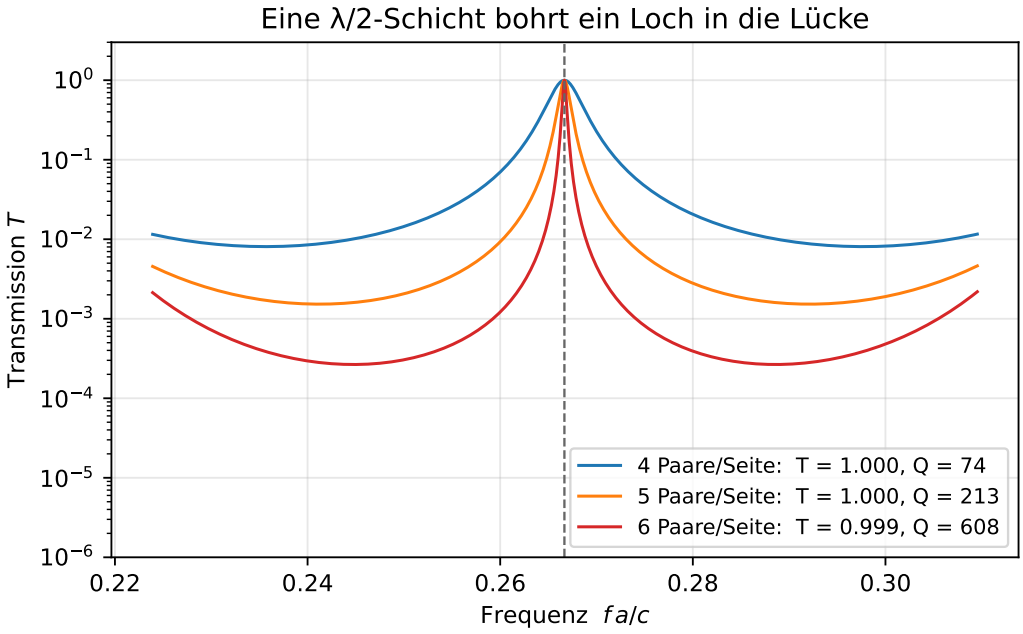


Abbildung 26.9: Die Transmissionsnadel der λ/2-Defektschicht: mitten in der hermetischen Lücke öffnet sich ein nadelfeines Fenster mit T = 1 — ein Fabry-Pérot-Resonator, dessen Spiegel Bragg-Stapel sind. Pro zusätzlichem Spiegelpaar wird die Nadel um den Faktor 1/0,36 ≈ 3 schmaler (Q: 74 → 213 → 608).

Die „eine falsche Schicht" richtet alles aus: Bei exakt f_0 steigt T auf **1,000** — der Stapel, der eben noch 10^{-3} durchließ, wird für genau eine Frequenz perfekt durchsichtig. Die λ/2-Schicht ist ein Resonator zwischen zwei Bragg-Spiegeln; auf Resonanz baut sich das Feld in der Kavität so weit auf, dass die durchgelassene Welle die reflektierte exakt auslöscht (dieselbe Logik wie beim λ/4-Trafo aus Kapitel 22, nur mit gespeichertem Feld als Vermittler). Jedes weitere Spiegelpaar verdreifacht Q — der Faktor ist unser alter Bekannter $1/0{,}36$ aus der Spurformel. Kapitel 27 beginnt genau hier.

Ü 26.4 — Wände aus Nichts (Übertragen). Entferne statt einer Reihe nur den **mittleren Stab** eines Stäbchen-Kristalls und miss mit Harminv (Kapitel 15), was in diesem Punktdefekt wohnt: Resonanzfrequenz und Güte für Kristalle aus 7×7 und 9×9 Stäben. Vorhersage: In welche Richtung muss f fallen (denk an die Lücke), und was erwartest du beim Schritt 7×7 → 9×9 für Q — Verdopplung oder mehr?

Musterlösung zu Ü 26.4

```python
# von oben: EPS_STAB, R_STAB


def punktdefekt(nk):
    geom = [mp.Cylinder(radius=R_STAB, center=mp.Vector3(i, j),
                        material=mp.Medium(epsilon=EPS_STAB))
            for i in range(-nk, nk + 1) for j in range(-nk, nk + 1)
            if not (i == 0 and j == 0)]
    L = 2 * nk + 1
    sim = mp.Simulation(
        cell_size=mp.Vector3(L + 6, L + 6), resolution=16,
        boundary_layers=[mp.PML(2.0)], geometry=geom,
        sources=[mp.Source(mp.GaussianSource(0.38, fwidth=0.10),
                           component=mp.Ez,
                           center=mp.Vector3(0.123, 0.061))])
    h = mp.Harminv(mp.Ez, mp.Vector3(0.1, 0.05), 0.38, 0.10)
    sim.run(mp.after_sources(h), until_after_sources=400)
    beste = max(h.modes, key=lambda m: abs(m.Q))
    return beste.freq, abs(beste.Q)


for nk in (3, 4):
    f_def, q_def = punktdefekt(nk)
    print(f"{2*nk+1}x{2*nk+1}-Kristall:  f = {f_def:.4f},  Q = {q_def:.0f}")
```

```
7x7-Kristall:  f = 0.3945,  Q = 728
9x9-Kristall:  f = 0.3944,  Q = 4227
```

Die Mode liegt bei $f = 0{,}394$ — **mitten in der Lücke** (0,32–0,44), wie es sein muss: Nur dort sind die Wände dicht. Und Q wächst beim Schritt 7×7 → 9×9 um den Faktor **sechs**, nicht zwei: Der einzige Verlustkanal ist das evaneszente Durchtunneln der endlichen Kristallwand, und das fällt (wie beim Stapel) **exponentiell** mit jeder Stabreihe. Vergleiche Kapitel 15: Dort lebte der Resonator in Metallwänden und verlor durchs Koppel-Loch — hier sind die Wände „Nichts mit Verbot", und man stellt die Güte ein, indem man Reihen hinzufügt. Genau solche Punktdefekt-Kavitäten (in Silizium, bei 1550 nm) erreichen $Q > 10^6$ auf Mikrometer-Volumen — Bauteile für Filter, Sensoren und Laser auf dem Chip.

Das Kleingedruckte

Zwei Richtungen sind nicht alle. Unsere Schnittmengen-Messung hat die Γ–X- und die Γ–M-Richtung geprüft (so heißen die Wege vom Zonenzentrum zur Kante bzw. zur Ecke der **Brillouin-Zone**, des Einheitsquadrats im K-Raum). Dass dazwischen nichts Schlimmeres passiert, ist beim Quadratgitter Erfahrungstatsache, aber kein Naturgesetz — die saubere Methode rechnet das Eigenwertproblem für *alle* $\vec{K}$ entlang des Zonenrands (Werkzeuge wie MPB, der Frequenzbereichs-Bruder von Meep, machen genau das; die Literaturkanten 0,32/0,44 stammen daher). Unsere Transfer-Matrix ist der 1D-Spezialfall dieser Bloch-Rechnung.

Polarisation entscheidet. Die Lücke des Stäbchengitters gilt für E_z (TM). Für die andere Polarisation (TE) hat dasselbe Gitter **keine** vollständige Lücke — dafür hat sie das „Negativ": ein Lochgitter in einer dielektrischen Platte. Faustregel aus der Literatur: isolierte Dielektrikum-Inseln begünstigen TM-Lücken, verbundene Netze TE-Lücken. Wer beide will, kombiniert beides (z. B. Wabengitter) — erst dann sperrt der Kristall *jedes* Licht seiner Lücke.

Von 2D nach 3D. Eine vollständige 3D-Bandlücke (alle Richtungen, alle Polarisationen) ist deutlich schwerer zu bekommen: Der Opal schafft es trotz seiner hübschen Farben nicht (seine Pseudolücke ist richtungsabhängig — daher das Farbenspiel), der **invertierte Opal** (Luftkugeln in Titandioxid) und die „Holzstapel"-Struktur (*woodpile*) schaffen es. Vorgeschlagen wurde das Konzept 1987 unabhängig von Eli Yablonovitch (Ziel: spontane Emission unterdrücken — eine Lichtquelle in der Lücke *kann* nicht abstrahlen!) und Sajeev John (Ziel: Licht-Lokalisierung); die erste künstliche 3D-Lücke (1991, „Yablonovite") wurde in Plexiglas gebohrt und im Mikrowellenband vermessen — Skaleninvarianz (Kapitel 11) macht Mikrowellen-Modelle zu ehrlichen Prototypen der Photonik.

Die Analogie trägt weit. Bloch entwickelte die Bandtheorie 1928 für Elektronen in Kristallen; „Bänder", „Lücke", „Zustandsdichte", „Defektzustand", „Dotierung" — das gesamte Vokabular dieses Kapitels stammt aus der Festkörperphysik. Der Unterschied: Photonen wechselwirken nicht miteinander und folgen exakt den linearen Maxwell-Gleichungen — der photonische Kristall ist insofern das *sauberere* Lehrbuchbeispiel der Bandtheorie als der echte Halbleiter mit seinen Elektron-Elektron-Effekten. Kapitel 30 kommt auf diese Brücke zurück, dann von der Quantenseite.

27 Ringresonator und Fabry-Pérot

Am Ende von Kapitel 26 stand eine Kuriosität: Eine einzige λ/2-Schicht zwischen zwei fast perfekten Spiegeln ließ bei genau einer Frequenz **alles** Licht hindurch — $T = 1{,}000$ durch eine Wand, die sonst 99,9 % zurückwirft. Was dort als Nadel in der Bandlücke auftauchte, ist eines der ältesten und nützlichsten Bauteile der Optik: das **Fabry-Pérot- Interferometer**, zwei Spiegel im Abstand L. Es steckt im Etalon des Spektroskopikers, in jeder Laserdiode — und in den vier Kilometer langen Armen von LIGO, wo es Licht hunderte Male hin- und herfaltet, bis eine Längenänderung von einem Zehntausendstel Protonendurchmesser messbar wird.

In diesem Kapitel verstehen wir den Resonator erst als **Echo-Summe** (eine geometrische Reihe, die wir seit der Lampen-Treppe in Kapitel 22 im Griff haben), messen dann seinen Frequenz-Kamm in Meep nach — und biegen ihn anschließend zum Kreis: Der **Ringresonator** ist ein Fabry-Pérot ohne Spiegel und das Arbeitspferd der Silizium-Photonik. Zum Schluss tun wir, was seit Kapitel 19 in der Luft liegt: Wir füllen den Resonator mit Verstärkung und sehen einen **Laser** an seiner Schwelle anspringen.

Lernziele

Nach diesem Kapitel kannst du …

1. … die Transmission eines Fabry-Pérot als **geometrische Reihe der Umlauf-Echos** herleiten und weißt, warum auf Resonanz $T = 1$ wird, obwohl jeder Spiegel fast alles reflektiert,
2. … den Frequenz-Kamm vermessen: **freier Spektralbereich** FSR $\approx 1/(2L)$, **Finesse** $\pi\sqrt{R}/(1-R)$, Güte Q — und du kennst die Feldüberhöhung in der Kavität,
3. … erklären, wie ein **Ringresonator** an einen Wellenleiter koppelt (evaneszent, Kapitel 17), warum im Bus Dips entstehen und was die azimutale Modenzahl ist,
4. … **kritische Kopplung** erkennen: Der Transmissions-Dip wird genau dann perfekt schwarz, wenn Koppel- und Eigenverlust gleichziehen — die Impedanzanpassung der Photonik,

5. … die **Laser-Schwelle** quantitativ vorhersagen: Verstärkungsrate = Verlustrate des kalten Resonators — und du weißt, warum erst die Sättigung aus einem Verstärker einen stabilen Oszillator macht (Kapitel 19).

27.1 Die Echo-Summe: vom Klingeln zum Kamm

Was macht ein Puls zwischen zwei Teilspiegeln? Er klingelt. Genau das haben wir in Übung 6.4 gemessen: Zwischen zwei Glasgrenzen hin- und herlaufende Echos, jedes um den Reflexionsfaktor leiser. Und in Kapitel 22 hat uns die Lampen-Treppe gezeigt, wie man mit so etwas rechnet: Die Echos bilden eine **geometrische Reihe**, und geometrische Reihen kann man aufsummieren.

Nehmen wir zwei identische Teilspiegel (Amplituden r fürs Reflektieren, t fürs Durchlassen) im Abstand L. Eine Welle, die ganz hindurch will, hat unendlich viele Wege: direkt durch beide Spiegel; oder drinnen einmal extra hin und zurück; oder zweimal; oder … Jeder zusätzliche Umlauf multipliziert die Amplitude mit demselben Faktor — zweimal reflektieren, zweimal die Strecke L:

$$t_{\text{ges}} = t^2\,\mathrm{e}^{-\mathrm{i}\varphi} \sum_{m=0}^{\infty} \left(r^2 \mathrm{e}^{-2\mathrm{i}\varphi}\right)^m = \frac{t^2\,\mathrm{e}^{-\mathrm{i}\varphi}}{1 - r^2\,\mathrm{e}^{-2\mathrm{i}\varphi}}, \qquad \varphi = \frac{2\pi f}{c}\,L.$$

Ein Wort zur **Phasenuhr**, bevor wir rechnen — hier lauert dieselbe Falle wie zwischen Meep und NEC in Kapitel 23 (unsere Regel von dort: Konventionen prüft man an einem Fall mit bekannter Antwort). Unsere Transfer-Matrix aus Kapitel 26 hat eine eingebaute Vorzeichenwahl, und der Testfall ist eine schlichte Leerstrecke: Ihre Matrix liefert $t = \mathrm{e}^{-\mathrm{i}\delta}$ — die Laufphase zählt in dieser Konvention **negativ**. Also muss auch der Umweg in unserer Reihe als $\mathrm{e}^{-\mathrm{i}\varphi}$ auftreten, sonst landen die Resonanzen an den falschen Stellen.

Als Spiegel nehmen wir das einfachste, was das Regal hergibt: je **eine** λ/4-Schicht mit $n = 3$ (bei der Bezugsfrequenz $f_0 = 0{,}25$), Kavität $L = 5$ Luft dazwischen. Eine solche Einzelschicht reflektiert

$$R = \left(\frac{1 - n^2}{1 + n^2}\right)^2 = \left(\frac{-8}{10}\right)^2 = 0{,}64.$$

Vorhersage-Punkt (PRIMM): Beide Spiegel werfen 64 % der Leistung zurück. Wie viel kommt auf Resonanz — wenn alle Echos im Takt sind — durch

das Paar? Höchstens $0{,}36^2 \approx 13\,\%$? Mehr? Rechne nicht, rate — und sieh dann zu, wie die Reihe konvergiert.

```
import numpy as np
import matplotlib.pyplot as plt

# --- Transfer-Matrix-Werkzeug (wie Kap. 26) ---
F0, N_SP = 0.25, 3.0
D_SP = 1.0 / (4 * N_SP * F0)     # lambda/4-Spiegelschicht
L = 5.0                          # Kavitaetslaenge (Luft)


def schichtmatrix(n, d, f):
    """Charakteristische 2x2-Matrix einer Schicht (Kap. 26)."""
    delta = 2.0 * np.pi * f * n * d
    return np.array([[np.cos(delta), 1j * np.sin(delta) / n],
                     [1j * n * np.sin(delta), np.cos(delta)]])


def rt_von(M):
    """komplexe r und t einer Kette zwischen Luft-Halbraeumen."""
    nenner = (M[0, 0] + M[0, 1]) + (M[1, 0] + M[1, 1])
    r = ((M[0, 0] + M[0, 1]) - (M[1, 0] + M[1, 1])) / nenner
    t = 2.0 / nenner
    return r, t


# Phasenuhr-Probe: Leerstrecke der Dicke d hat t = e^(-i*delta)
_, t_vak = rt_von(schichtmatrix(1.0, 1.0, 0.25))
print(f"Leerstrecke: t = {t_vak:.4f}  (e^(-i pi/2) = {np.exp(-1j*np.pi/2):.4f})")

r_sp, t_sp = rt_von(schichtmatrix(N_SP, D_SP, F0))
print(f"Einzelspiegel: R = {np.abs(r_sp)**2:.4f}")

# Echo-Summe gegen die exakte Transfer-Matrix des Gesamtsystems
def fp_t_tmm(f):
    M = schichtmatrix(N_SP, D_SP, f) @ schichtmatrix(1.0, L, f) \
        @ schichtmatrix(N_SP, D_SP, f)
    return rt_von(M)[1]


F_RES = 0.2946                     # eine Resonanz (gleich gemessen)
r, t = rt_von(schichtmatrix(N_SP, D_SP, F_RES))
phi = 2 * np.pi * F_RES * L
```

```
print("\nEcho-Summe auf Resonanz:")
for n_um in [1, 2, 5, 10, 30]:
    s = sum((r**2 * np.exp(-2j * phi))**m for m in range(n_um))
    print(f"  {n_um:3d} Umläufe: T = {np.abs(t**2 * np.exp(-1j*phi) * s)**2:.4f}")
t_exakt = t**2 * np.exp(-1j * phi) / (1 - r**2 * np.exp(-2j * phi))
print(f"  geschlossene Reihe: T = {np.abs(t_exakt)**2:.6f}")
print(f"  Transfer-Matrix:    T = {np.abs(fp_t_tmm(F_RES))**2:.6f}")
print(f"  Differenz: {abs(np.abs(t_exakt)**2 - np.abs(fp_t_tmm(F_RES))**2):.1e}")
```

```
Leerstrecke: t = 0.0000-1.0000j  (e^(-i pi/2) = 0.0000-1.0000j)
Einzelspiegel: R = 0.6400

Echo-Summe auf Resonanz:
    1 Umläufe: T = 0.1433
    2 Umläufe: T = 0.3767
    5 Umläufe: T = 0.8232
   10 Umläufe: T = 0.9829
   30 Umläufe: T = 1.0000
  geschlossene Reihe: T = 0.999968
  Transfer-Matrix:    T = 0.999968
  Differenz: 1.6e-15
```

Hundert Prozent — durch zwei 64-%-Spiegel. Das erste Echo allein bringt nur 13 %, aber die Nachzügler stapeln sich phasenrichtig obendrauf, und die geschlossene Reihe stimmt mit der Transfer-Matrix auf Maschinengenauigkeit überein (sie *ist* dieselbe Physik, nur anders sortiert). Der Mechanismus ist derselbe wie bei der λ/2-Nadel aus Übung 26.3: Auf Resonanz baut sich in der Kavität so viel Feld auf, dass dessen Leck durch den Eingangsspiegel die direkte Reflexion **exakt auslöscht** — destruktive Interferenz rückwärts heißt konstruktive vorwärts. Ein Resonator ist ein Anti-Spiegel für seine Lieblingsfrequenz.

Aus der Reihe folgt mit $R = |r|^2$ sofort die berühmte **Airy-Formel** (Nenner ausmultiplizieren):

$$T(f) = \frac{(1-R)^2}{1 + R^2 - 2R\cos\delta}, \qquad \delta = \frac{4\pi f L}{c} - 2\arg(r),$$

und mit ihr der ganze Kamm: Maxima ($T = 1$), wann immer die Umlaufphase δ ein Vielfaches von 2π trifft.

```
# von oben: schichtmatrix(), rt_von(), fp_t_tmm(), N_SP, D_SP, L
fs = np.linspace(0.15, 0.35, 2001)
T_tmm = np.array([np.abs(fp_t_tmm(f))**2 for f in fs])


def airy_T(f):
    r, _ = rt_von(schichtmatrix(N_SP, D_SP, f))
    R = np.abs(r)**2
    delta = 4 * np.pi * f * L - 2 * np.angle(r)
    return (1 - R)**2 / (1 + R**2 - 2 * R * np.cos(delta))


T_airy = np.array([airy_T(f) for f in fs])
print(f"max |Airy - TMM| = {np.max(np.abs(T_airy - T_tmm)):.1e}")

# Kamm vermessen: Resonanzlagen, FSR, Halbwertsbreite -> Finesse
from scipy.signal import argrelmax   # lokale Maxima eines Arrays
i_res = [i for i in argrelmax(T_tmm)[0] if T_tmm[i] > 0.9]
f_res = fs[i_res]
fsr = float(np.diff(f_res).mean())
ip = i_res[-1]
ueber = T_tmm > 0.5
lo = hi = ip
while ueber[lo - 1]:
    lo -= 1
while ueber[hi + 1]:
    hi += 1
fwhm = fs[hi] - fs[lo]
R_SP = np.abs(rt_von(schichtmatrix(N_SP, D_SP, F0))[0])**2
print(f"Resonanzen: {np.round(f_res, 4)}")
print(f"FSR = {fsr:.4f}   (1/(2L) = {1/(2*L):.4f}  ->  L_eff = {1/(2*fsr):.2f})")
print(f"Finesse = FSR/FWHM = {fsr/fwhm:.2f}   "
      f"(ideal pi*sqrt(R)/(1-R) = {np.pi*np.sqrt(R_SP)/(1-R_SP):.2f})")

fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.plot(fs, T_tmm, "C1-", lw=1.5, label="Transfer-Matrix")
ax.plot(fs[::25], T_airy[::25], "C0o", ms=4, mfc="none", label="Airy-Formel")
for k in range(-1, 3):
    ax.axvline(0.25 + k * 0.1, color="0.55", ls="--", lw=0.9)
ax.axvline(0.25, color="0.55", ls="--", lw=0.9,
           label="naive Kammteilung 1/(2L)")
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$")
```

```
ax.set_title("Zwei Spiegel, ein Kamm")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

```
max |Airy - TMM| = 3.0e-15
Resonanzen: [0.2054 0.2946]
FSR = 0.0892   (1/(2L) = 0.1000  ->  L_eff = 5.61)
Finesse = FSR/FWHM = 6.56   (ideal pi*sqrt(R)/(1-R) = 6.98)
```

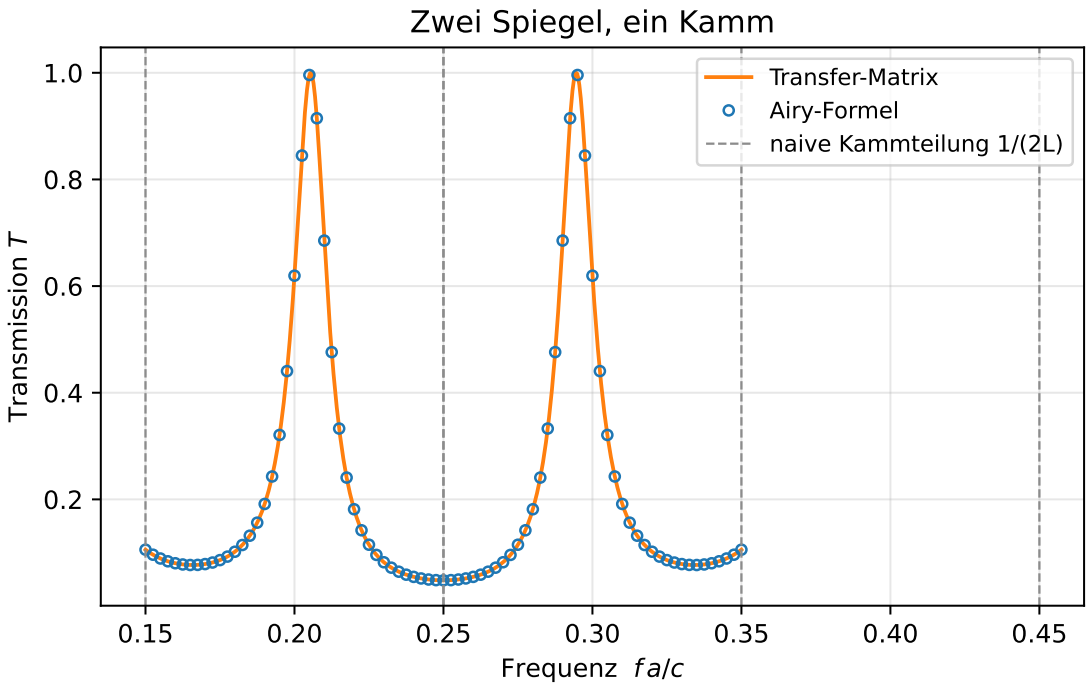


Abbildung 27.1: Der Fabry-Pérot-Kamm aus der Transfer-Matrix (Linie) mit der Airy-Formel (Punkte — deckungsgleich): Resonanzen mit T = 1 im Abstand des freien Spektralbereichs, dazwischen sperrt das Spiegelpaar besser als ein Einzelspiegel (T unter 0,1). Gestrichelt die naive Kammteilung 1/(2L): Der echte Kamm ist enger — die Welle wohnt ein Stück weit im Spiegel.

Zwei Messwerte verdienen einen zweiten Blick:

- **Der freie Spektralbereich** (FSR), der Abstand der Kammzähne, ist die Umlauf-Frequenz des Resonators: FSR $= c/(2L)$ — ein Echo pro

Umlauf, eine Resonanz pro aufgesammelter Vollphase. Gemessen: 0,0892 statt 0,1. Kein Fehler, sondern Kapitel 12/26: Die Welle dringt beim Reflektieren ein Stück in den Spiegel ein (ihre Reflexionsphase $\arg r$ dreht sich mit f), der Resonator ist *optisch länger* als gebaut — $L_{\text{eff}} = 5{,}61$ statt 5.

- **Die Finesse** $\mathcal{F} = \text{FSR}/\text{FWHM}$ zählt, wie viele Linienbreiten zwischen zwei Zähne passen — die Trennschärfe des Kamms. Ideal wäre $\pi\sqrt{R}/(1-R) = 6{,}98$; unsere realen (dispersiven) Spiegel liefern 6,6.

27.2 Der Kamm im Experiment

Jetzt Meep — derselbe 1D-Aufbau wie beim Bragg-Stapel in Kapitel 26. **Frage:** Liegt der gemessene Kamm auf der Theorielinie, inklusive der $T = 1$-Spitzen? **Bühne:** 1D-Zelle (`dimensions=1`), zwei $n = 3$-Schichten der Dicke $1/3$ im Abstand 5, beidseits Luft und PML. **Anfangsbedingungen:** breiter Gauß-Puls vor dem ersten Spiegel. **Messgrößen:** Fluss hinter dem zweiten Spiegel, normiert auf den Leerlauf. **Erfolgskriterium:** rms-Abstand zur Transfer-Matrix im Prozentbereich.

```
# von oben: N_SP, D_SP, L, fp_t_tmm()
import meep as mp

mp.verbosity(0)

DPML, LUFT = 2.0, 3.0
LZ = 2 * DPML + 2 * LUFT + 2 * D_SP + L
AUFL = 40


def spiegel_geom():
    zi = -(L / 2 + D_SP / 2)
    return [mp.Block(size=mp.Vector3(mp.inf, mp.inf, D_SP),
                     center=mp.Vector3(0, 0, s * zi),
                     material=mp.Medium(index=N_SP)) for s in (-1, 1)]


def sweep_1d(geom):
    sim = mp.Simulation(
        cell_size=mp.Vector3(0, 0, LZ), dimensions=1, resolution=AUFL,
        boundary_layers=[mp.PML(DPML)], geometry=geom,
        sources=[mp.Source(mp.GaussianSource(0.25, fwidth=0.25),
                           component=mp.Ex,
```

```
                            center=mp.Vector3(0, 0, -LZ / 2 + DPML + 0.5))])
    z_m = LZ / 2 - DPML - 0.5
    tr = sim.add_flux(0.25, 0.25, 801,
                      mp.FluxRegion(center=mp.Vector3(0, 0, z_m)))
    sim.run(until_after_sources=mp.stop_when_fields_decayed(
        50, mp.Ex, mp.Vector3(0, 0, z_m), 1e-9))
    return np.array(mp.get_flux_freqs(tr)), np.array(mp.get_fluxes(tr))


fs_m, P_ref = sweep_1d([])
_, P_fp = sweep_1d(spiegel_geom())
T_meep = P_fp / P_ref
T_theo = np.array([np.abs(fp_t_tmm(f))**2 for f in fs_m])
fen = (fs_m > 0.15) & (fs_m < 0.35)
print(f"rms(Meep - TMM) = {np.sqrt(np.mean((T_meep[fen]-T_theo[fen])**2)):.4f}")
```

```
Using MPI version 4.1, 1 processes
rms(Meep - TMM) = 0.0014
```

```
# von oben: fs_m, T_meep, T_theo, fen, fs, T_tmm
fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.plot(fs, T_tmm, "C1-", lw=1.5, label="Transfer-Matrix")
ax.plot(fs_m[fen][::10], T_meep[fen][::10], "C0o", ms=4.5, mfc="none",
        label="Meep 1D (jeder 10. Punkt)")
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$")
ax.set_title("Der Kamm besteht die Nagelprobe")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

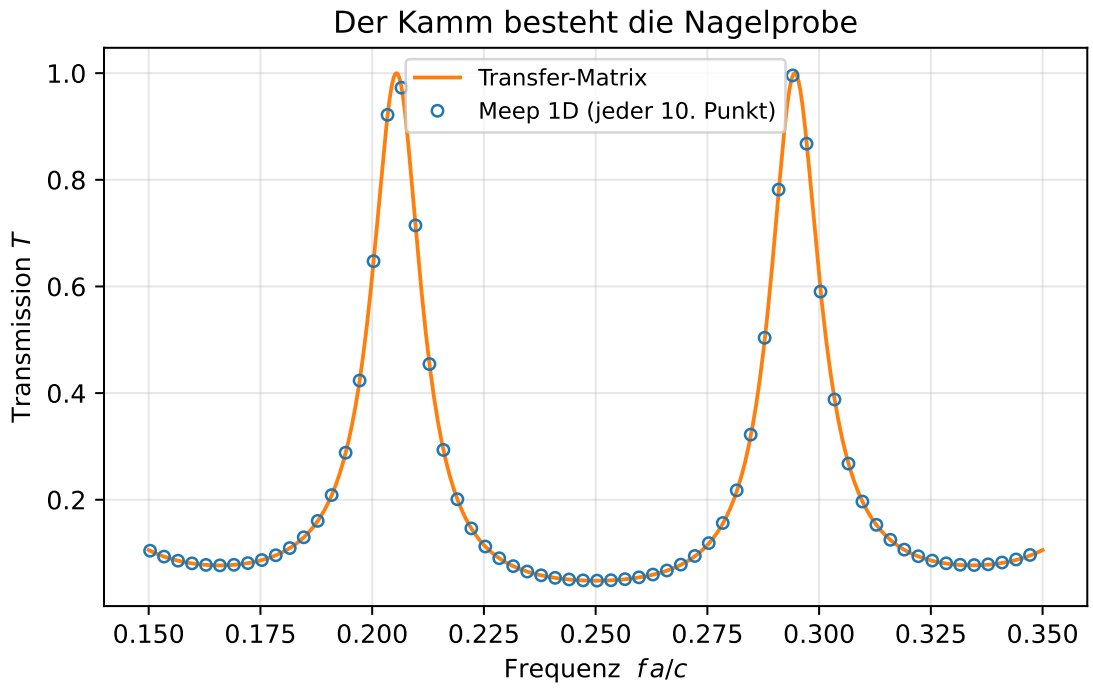


Abbildung 27.2: Meep-Messpunkte auf der Transfer-Matrix-Linie: Der Kamm stimmt über den ganzen Sweep, die Spitzen erreichen T = 1,000 — durch zwei 64-%-Spiegel. Dieselbe Doppelprobe wie in Kapitel 26, jetzt für den Resonator.

27.2.1 Güte und Klingeldauer

Der Kamm verrät über seine Linienbreite die Güte $Q = f/\Delta f \approx 22$ — und Kapitel 15 verlangt die Gegenprobe über das **Abklingen**: Ein Resonator mit Güte Q klingelt $Q/2\pi$ Schwingungen lang, sein Feld fällt wie $e^{-\omega t/2Q}$. Der Versuchsaufbau: **Frage:** Liefert das Ausklingen dasselbe Q wie die Linienbreite? **Bühne:** dieselbe Zelle; ein kurzer Puls stößt die Mode bei $f = 0{,}2946$ *in* der Kavität an, danach lauschen wir an einer Sonde. **Messgröße:** die Amplitude der Resonanzschwingung über der Zeit — gemessen mit einem **gleitenden Lock-in**: In Fenstern von vier Perioden multiplizieren wir das Signal mit $e^{-2\pi i f_{\mathrm{res}} t}$ und mitteln — der Frequenz-Detektor aus Kapitel 10, nur eben Fenster für Fenster. Das hört ausschließlich auf die Mode und ignoriert alles andere; warum uns das gleich noch den Hals rettet, zeigt der Laser-Abschnitt. **Erfolgskriterium:** ln-Amplitude auf einer Geraden, Q aus der Steigung gleich dem Kamm-Q.

Nebenbefund fürs Werkstattbuch: Das bequeme `mp.Harminv` aus Kapitel 15 ist hier nutzlos — es verwirft Moden mit $Q < 50$ grundsätzlich als numerischen Müll. Für unser $Q \approx 22$ heißt es selber fitten.

```
# von oben: spiegel_geom(), LZ, DPML, AUFL, L
F_RES = 0.2946


def lockin_rate(sig=0.0, dauer=250):
    """Klingelrate der Resonanzmode (Feld), per gleitendem Lock-in.

    sig > 0 fuellt die Kavitaet mit Verstaerkung (kommt im
    Laser-Abschnitt) — die Messroutine bleibt dieselbe.
    """
    geom = spiegel_geom()
    if sig > 0:
        geom = geom + [mp.Block(size=mp.Vector3(mp.inf, mp.inf, L),
                                center=mp.Vector3(0, 0, 0),
                                material=mp.Medium(D_conductivity=-sig))]
    sim = mp.Simulation(
        cell_size=mp.Vector3(0, 0, LZ), dimensions=1, resolution=AUFL,
        boundary_layers=[mp.PML(DPML)], geometry=geom,
        sources=[mp.Source(mp.GaussianSource(F_RES, fwidth=0.05),
                           component=mp.Ex, center=mp.Vector3(0, 0, 0.31))])
    ts, es = [], []

    def messen(sim):
        ts.append(sim.meep_time())
        es.append(np.real(sim.get_field_point(mp.Ex, mp.Vector3(0, 0, 0.17))))

    sim.run(mp.after_sources(mp.at_every(0.25, messen)),
            until_after_sources=dauer)
    t, e = np.array(ts), np.array(es)
    n_f = int(4 / F_RES / 0.25)        # Fenster: vier Perioden
    mitten, amps = [], []
    for i0 in range(0, len(t) - n_f, n_f // 2):
        tw, ew = t[i0:i0 + n_f], e[i0:i0 + n_f]
        z = np.sum(ew * np.exp(-2j * np.pi * F_RES * tw))
        mitten.append(tw.mean())
        amps.append(2 * abs(z) / n_f)
    mitten, amps = np.array(mitten), np.array(amps)
    m = (amps > 1e-10) & (amps < 1e6)
    return np.polyfit(mitten[m], np.log(amps[m]), 1)[0]
```

```
rate_kalt = lockin_rate()
Q_abkling = 2 * np.pi * F_RES / (-2 * rate_kalt)
print(f"Klingelrate: {rate_kalt:.5f} pro Zeiteinheit")
print(f"Q aus dem Abklingen: {Q_abkling:.1f}   (aus der Linienbreite: 21,7)")
```

```
Klingelrate: -0.04236 pro Zeiteinheit
Q aus dem Abklingen: 21.9   (aus der Linienbreite: 21,7)
```

Beide Wege, ein Q — die Kapitel-15-Doppelmessung funktioniert auch hier. Und drinnen? Auf Resonanz speichert die Kavität Feld, bis die Verluste mit dem Nachschub gleichziehen. Wie viel, sagt die Echo-Reihe: Die umlaufende Amplitude ist um $\sqrt{1-R}/(1-R)$ überhöht, und weil sich drinnen Hin- und Rückwelle zur Stehwelle addieren, kommt an den Bäuchen noch der Faktor $(1+\sqrt{R})$ dazu — macht $(1+\sqrt{R})\sqrt{1-R}/(1-R) = 3{,}0$ für unsere Spiegel:

```
# von oben: spiegel_geom(), LZ, DPML, AUFL, L, D_SP, F_RES, R_SP


def cw_profil(f):
    """Eingeschwungenes |Ex|(z) bei Dauerstrich-Anregung von links."""
    sim = mp.Simulation(
        cell_size=mp.Vector3(0, 0, LZ), dimensions=1, resolution=AUFL,
        boundary_layers=[mp.PML(DPML)], geometry=spiegel_geom(),
        force_complex_fields=True,
        sources=[mp.Source(mp.ContinuousSource(f), component=mp.Ex,
                           center=mp.Vector3(0, 0, -LZ / 2 + DPML + 0.5))])
    sim.run(until=400)
    ex = sim.get_array(component=mp.Ex, center=mp.Vector3(),
                       size=mp.Vector3(0, 0, LZ - 2 * DPML))
    z = np.linspace(-(LZ / 2 - DPML), LZ / 2 - DPML, len(ex))
    return z, ex


z_p, ex_on = cw_profil(F_RES)
_, ex_off = cw_profil(0.25)
innen = np.abs(ex_on)[np.abs(z_p) < L / 2 - 0.2].max()
davor = np.abs(ex_on)[(z_p < -L / 2 - D_SP - 0.3) & (z_p > -L / 2 - 2.5)].max()
ue_theo = (1 + np.sqrt(R_SP)) * np.sqrt(1 - R_SP) / (1 - R_SP)
print(f"Überhöhung auf Resonanz: {innen / davor:.2f}  (Theorie {ue_theo:.2f})")
innen_off = np.abs(ex_off)[np.abs(z_p) < L / 2 - 0.2].max()
print(f"zwischen den Resonanzen: innen nur noch "
      f"{innen_off / np.abs(ex_off).max():.2f} der Außenamplitude")
```

```
Überhöhung auf Resonanz: 2.86  (Theorie 3.00)
zwischen den Resonanzen: innen nur noch 0.33 der Außenamplitude
```

```python
# von oben: z_p, ex_on, ex_off, L, D_SP
fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.axvspan(-L / 2 - D_SP, -L / 2, color="0.82")
ax.axvspan(L / 2, L / 2 + D_SP, color="0.82")
ax.plot(z_p, np.abs(ex_on), "C0-", lw=1.6,
        label=f"auf Resonanz (f = {F_RES})")
ax.plot(z_p, np.abs(ex_off), "C1-", lw=1.4, label="daneben (f = 0,25)")
ax.set_xlabel("z (graue Balken: die Spiegel)")
ax.set_ylabel("$|E_x|$")
ax.set_title("Die Kavität als Feldspeicher")
ax.legend(fontsize=9, loc="upper right")
ax.grid(alpha=0.3)
plt.show()
```

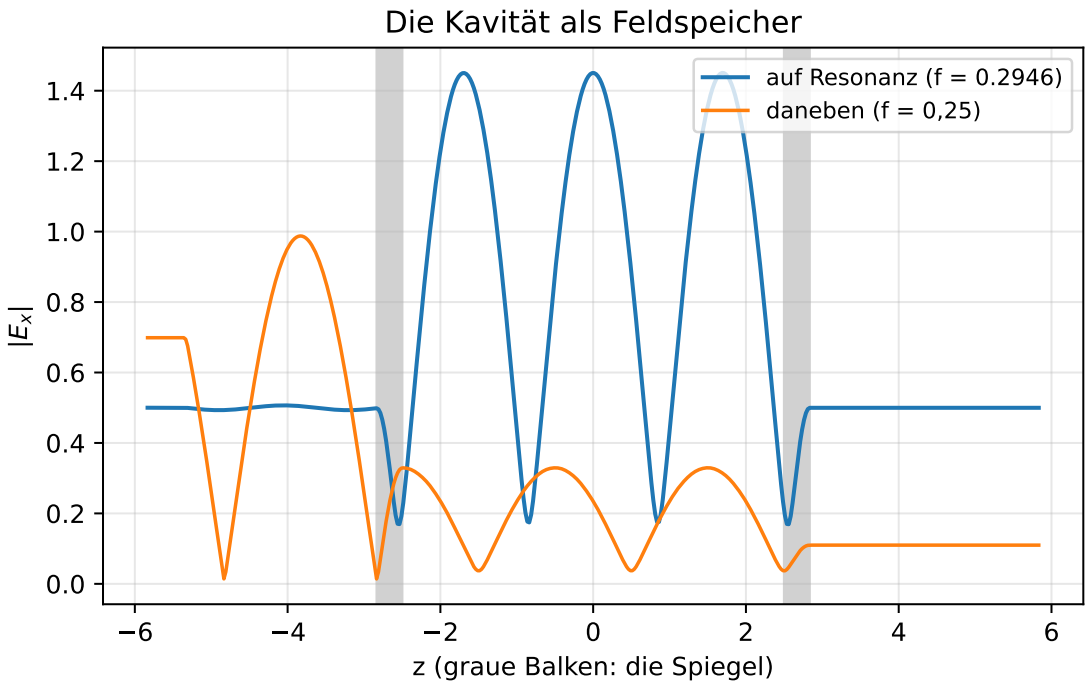


Abbildung 27.3: |E_x| entlang der Achse im eingeschwungenen Zustand (Spiegel grau): Auf Resonanz (blau) staut sich in der Kavität das Dreifache der Einfallsamplitude, und vor dem Resonator ist die Stehwellen-Riffelung fast verschwunden — er schluckt alles (T = 1). Zwischen den Resonanzen (orange) bleibt die Kavität fast leer, und draußen steht das volle Echo-Muster.

Diese Feldüberhöhung ist der eigentliche Beruf des Fabry-Pérot: LIGO speichert in seinen Arm-Kavitäten hunderte Kilowatt zirkulierender Lichtleistung aus wenigen Watt Laser — jede Faltung erhöht die Empfindlichkeit, als wäre der Arm hunderte Male so lang.

27.3 Der Ringresonator: Fabry-Pérot ohne Spiegel

Spiegel sind in der Optik Luxus — auf einem Photonik-Chip erst recht. Der elegante Ausweg: **gar keine Spiegel**. Biege einen Wellenleiter zum Kreis, und die Welle beißt sich selbst in den Schwanz. Resonanz herrscht, wann immer der Umfang eine ganze Zahl von Wellenlängen trägt — dann verstärkt sich die umlaufende Welle bei jedem Umlauf selbst, exakt wie zwischen zwei Spiegeln, nur ohne Umkehren. Die Rolle der Spiegelverluste übernimmt die

Krümmung: Eine geführte Welle in der Kurve verliert ständig ein wenig Licht nach außen (die Glasfaser lässt grüßen — und Kapitel 12: Totalreflexion ist nur an der *ebenen* Grenzfläche perfekt).

Der Versuchsaufbau: **Frage:** Welche Moden lebt ein dielektrischer Ring, und wie gut hält er sie? **Bühne:** 2D, ein Ring mit Innenradius 1, Leiterbreite 1, $n = 3{,}4$ (Silizium-Welt) in Luft; rundum PML. **Anfangsbedingungen:** ein Puls-Anstoß im Ringleiter. **Messgrößen:** `Harminv` (Kapitel 15) lauscht die Eigenfrequenzen samt Q heraus — hier funktioniert es, die Güten sind hoch genug. **Erfolgskriterium:** ein Frequenz-Kamm; und ein klarer Trend des Q über die Frequenz, den der nächste Absatz erklären soll.

Vorhersage-Punkt (PRIMM): Verliert die *hochfrequente* oder die *tieffrequente* Mode mehr Licht in der Kurve? Denk an Kapitel 17: Wie weit ragt der evaneszente Saum einer schwach geführten (= tieffrequenten) Welle aus dem Leiter hinaus?

```
# von oben: DPML
N_WG, W, R_IN = 3.4, 1.0, 1.0


def ring_moden(r_in, fcen=0.15, df=0.12):
    pad = 4.0
    s = 2 * (r_in + W + pad + DPML)
    geom = [mp.Cylinder(radius=r_in + W, material=mp.Medium(index=N_WG)),
            mp.Cylinder(radius=r_in, material=mp.air)]
    sim = mp.Simulation(
        cell_size=mp.Vector3(s, s), resolution=16,
        boundary_layers=[mp.PML(DPML)], geometry=geom,
        sources=[mp.Source(mp.GaussianSource(fcen, fwidth=df),
                           component=mp.Ez,
                           center=mp.Vector3(r_in + 0.1, 0.02))])
    h = mp.Harminv(mp.Ez, mp.Vector3(r_in + 0.12, -0.03), fcen, df)
    sim.run(mp.after_sources(h), until_after_sources=400)
    return sorted((m.freq, abs(m.Q)) for m in h.modes
                  if m.freq > 0.05 and abs(m.Q) > 20)


moden = ring_moden(R_IN)
for f_m, q_m in moden:
    print(f"f = {f_m:.4f}   Q = {q_m:7.0f}")
```

```
f = 0.1181   Q =      76
f = 0.1473   Q =     337
```

```
f = 0.1756   Q =    2069
f = 0.2036   Q =    5015
```

Ein Kamm — der FSR von etwa 0,028 ist wieder die Umlauf-Frequenz, diesmal über den Umfang statt über $2L$. Und das Q explodiert nach oben: von 76 auf 5000 innerhalb von vier Zähnen. Die Antwort auf den Vorhersage-Punkt: Die **tieffrequente** Mode leidet — ihr evaneszenter Saum ragt weit aus dem Leiter, und was weit draußen mitlaufen soll, müsste in der Kurve schneller als das Licht unterwegs sein. Diesen Teil verliert der Ring unweigerlich. Höhere Frequenz = engerer Saum = exponentiell dichter eingesperrt; die Biege-Übung am Kapitelende macht daraus eine Messreihe.

27.3.1 Der Ring am Bus

Ein einsamer Ring nützt niemandem — man muss Licht hinein- und herausbekommen, **ohne ihn zu berühren**. Das Werkzeug dafür haben wir in Übung 17.4 vermessen: Liegen zwei Leiter nah beieinander, hüpft Leistung durch die evaneszenten Säume hinüber (der Richtkoppler — dort als Vorgriff auf genau dieses Kapitel angekündigt). Also legen wir einen geraden **Bus-Wellenleiter** im Abstand g am Ring vorbei und schicken einen Puls hindurch. **Frage:** Was sieht der Bus von den Ringmoden? **Messgröße:** $T(f)$ am Bus-Ausgang, normiert auf den Lauf ohne Ring. **Erfolgskriterium:** Auffälligkeiten genau an den Harminv-Frequenzen von oben.

```python
# von oben: N_WG, W, R_IN, DPML, moden


def bus_sweep(spalt, mit_ring, fcen, df):
    pad = 3.0
    sx = 2 * (R_IN + W + pad + DPML)
    y_bus = -(R_IN + W + spalt + W / 2)
    geom = [mp.Block(size=mp.Vector3(mp.inf, W, mp.inf),
                     center=mp.Vector3(0, y_bus),
                     material=mp.Medium(index=N_WG))]
    if mit_ring:
        geom += [mp.Cylinder(radius=R_IN + W, material=mp.Medium(index=N_WG)),
                 mp.Cylinder(radius=R_IN, material=mp.air)]
    sim = mp.Simulation(
        cell_size=mp.Vector3(sx, sx), resolution=16,
        boundary_layers=[mp.PML(DPML)], geometry=geom,
        sources=[mp.Source(mp.GaussianSource(fcen, fwidth=df),
                           component=mp.Ez,
```

```
                            center=mp.Vector3(-sx / 2 + DPML + 0.5, y_bus),
                            size=mp.Vector3(0, 2 * W))])
    x_m = sx / 2 - DPML - 0.5
    tr = sim.add_flux(fcen, df, 401,
                      mp.FluxRegion(center=mp.Vector3(x_m, y_bus),
                                    size=mp.Vector3(0, 2 * W)))
    sim.run(until_after_sources=mp.stop_when_fields_decayed(
        100, mp.Ez, mp.Vector3(x_m, y_bus), 1e-8))
    return np.array(mp.get_flux_freqs(tr)), np.array(mp.get_fluxes(tr))


fs_r, P0r = bus_sweep(0.2, False, 0.15, 0.10)
_, P1r = bus_sweep(0.2, True, 0.15, 0.10)
T_ring = P1r / P0r

from scipy.signal import argrelmin   # lokale Minima
dips = [i for i in argrelmin(T_ring)[0]
        if T_ring[i] < 0.7 and 0.11 < fs_r[i] < 0.19]
print("Dips bei:", np.round(fs_r[dips], 4))
print("Ringmoden:", np.round([m[0] for m in moden[:3]], 4))
```

```
Dips bei: [0.1212 0.1485 0.1755]
Ringmoden: [0.1181 0.1473 0.1756]
```

```
# von oben: fs_r, T_ring, moden
fig, ax = plt.subplots(figsize=(7.0, 4.0))
ax.plot(fs_r, T_ring, "C0-", lw=1.5)
for f_m, _ in moden[:3]:
    ax.axvline(f_m, color="0.55", ls=":", lw=1.2)
ax.set_xlabel("Frequenz  $f\\,a/c$")
ax.set_ylabel("Transmission $T$ im Bus")
ax.set_title("Der Ring nascht am Bus — aber nur auf Resonanz")
ax.set_xlim(0.10, 0.20)
ax.set_ylim(0, 1.05)
ax.grid(alpha=0.3)
plt.show()
```

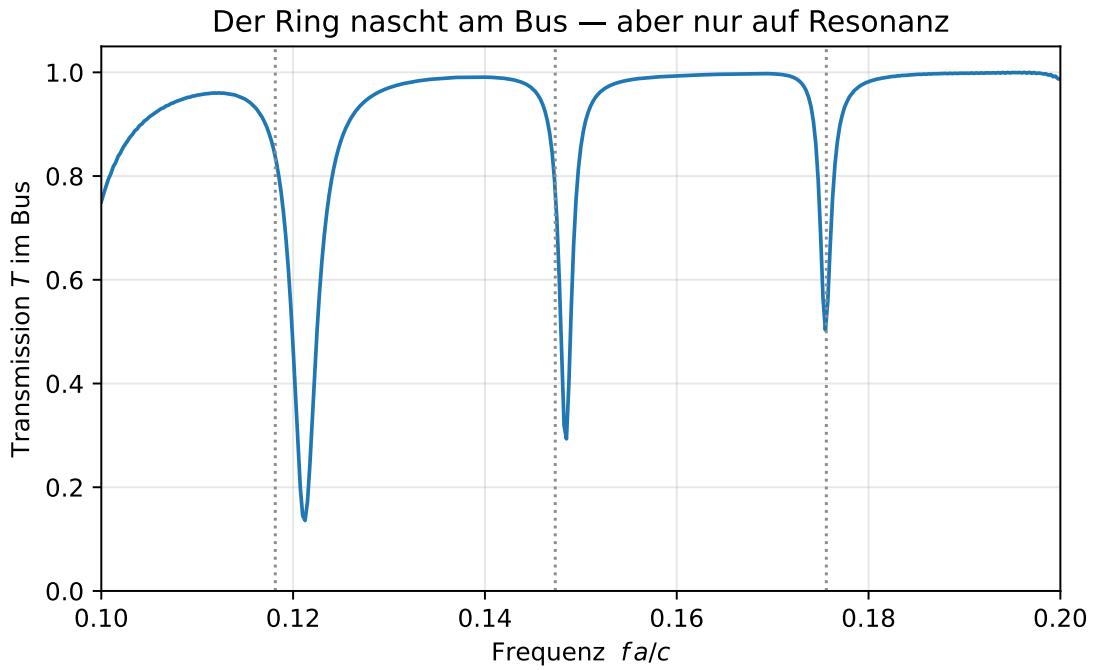


Abbildung 27.4: Transmission im Bus-Wellenleiter: scharfe Dips exakt an den Ringmoden (gepunktete Linien aus der Harminv-Messung). Bei diesen Frequenzen saugt der Ring die Leistung aus dem Bus und verstrahlt sie über seinen Biegeverlust — alle anderen Frequenzen laufen ungerührt vorbei. Ein Kerbfilter, periodisch wie der Resonator selbst.

Was im Ring selbst los ist, zeigt der Dauerstrich-Blick auf den mittleren Dip — und beantwortet nebenbei, welche azimutale Modenzahl m (wie viele Wellenlängen auf dem Umfang?) dort schwingt. Statt Bäuche zu zählen, lassen wir die Phase zählen: Auf einem Kreis durch den Ringleiter muss $\arg E_z$ insgesamt m volle Umdrehungen machen — `np.unwrap` (setzt Phasensprünge von $\pm\pi$ zu einer stetigen Kurve zusammen) macht daraus eine einzige Zahl.

```
# von oben: N_WG, W, R_IN, DPML
SPALT = 0.2
pad = 3.0
sx_cw = 2 * (R_IN + W + pad + DPML)
y_bus = -(R_IN + W + SPALT + W / 2)
geom_cw = [mp.Block(size=mp.Vector3(mp.inf, W, mp.inf),
                    center=mp.Vector3(0, y_bus),
                    material=mp.Medium(index=N_WG)),
```

```
            mp.Cylinder(radius=R_IN + W, material=mp.Medium(index=N_WG)),
            mp.Cylinder(radius=R_IN, material=mp.air)]
sim_cw = mp.Simulation(
    cell_size=mp.Vector3(sx_cw, sx_cw), resolution=16,
    boundary_layers=[mp.PML(DPML)], geometry=geom_cw,
    force_complex_fields=True,
    sources=[mp.Source(mp.ContinuousSource(0.1485), component=mp.Ez,
                       center=mp.Vector3(-sx_cw / 2 + DPML + 0.5, y_bus),
                       size=mp.Vector3(0, 2 * W))])
sim_cw.run(until=600)
seite_cw = 2 * (R_IN + W) + 2.5
ez_ring = sim_cw.get_array(component=mp.Ez,
                           center=mp.Vector3(0, y_bus / 2),
                           size=mp.Vector3(seite_cw, seite_cw + abs(y_bus)))
print(f"Gesundheitswert: max|Ez| = {np.abs(ez_ring).max():.2f}")

# Phasenwindung auf dem Kreis durch die Leitermitte
nx, ny = ez_ring.shape
xs = np.linspace(-seite_cw / 2, seite_cw / 2, nx)
ys = np.linspace(y_bus / 2 - (seite_cw + abs(y_bus)) / 2,
                 y_bus / 2 + (seite_cw + abs(y_bus)) / 2, ny)
rm = R_IN + W / 2
winkel = np.linspace(0, 2 * np.pi, 720, endpoint=False)
ring_werte = np.array([
    ez_ring[np.argmin(np.abs(xs - rm * np.cos(p))),
            np.argmin(np.abs(ys - rm * np.sin(p)))] for p in winkel])
phase = np.unwrap(np.angle(ring_werte))   # Phasenspruenge glaetten
print(f"azimutale Modenzahl m = {(phase[-1] - phase[0]) / (2 * np.pi):.2f}")

fig, ax = plt.subplots(figsize=(6.4, 6.6))
ax.imshow(np.abs(ez_ring).T, origin="lower", cmap="inferno",
          extent=[xs[0], xs[-1], ys[0], ys[-1]],
          vmax=np.percentile(np.abs(ez_ring), 99.5))
ax.add_patch(plt.Circle((0, 0), R_IN + W, fill=False, color="w",
                        lw=0.5, alpha=0.6))
ax.add_patch(plt.Circle((0, 0), R_IN, fill=False, color="w",
                        lw=0.5, alpha=0.6))
ax.set_xlabel("x/a")
ax.set_ylabel("y/a")
ax.set_title("Die umlaufende Mode (m = 4)")
plt.show()
```

```
Gesundheitswert: max|Ez| = 0.71
```

```
azimutale Modenzahl m = 4.00
```

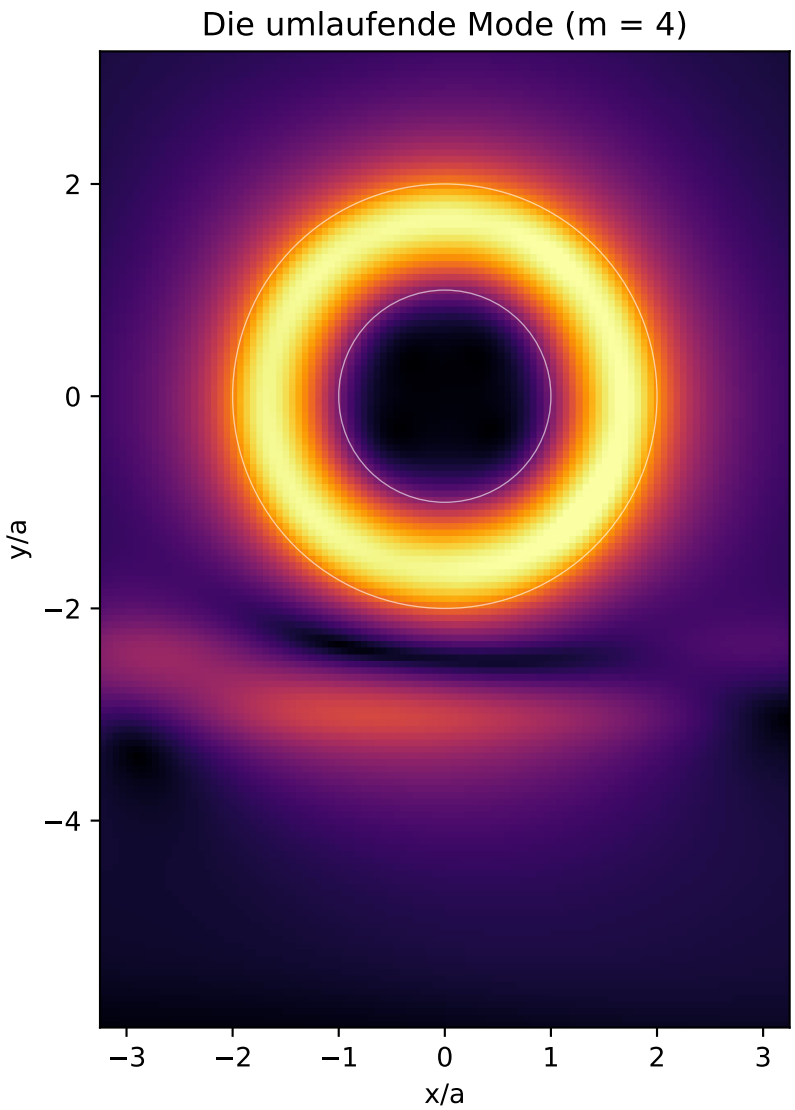


Abbildung 27.5: |E_z| im eingeschwungenen Zustand bei f = 0,1485 (Dauerstrich im Bus, der unten im Bild liegt): Der Ring leuchtet auf ganzer Länge fast gleichmäßig — das Kennzeichen einer laufenden Welle (eine Stehwelle hätte dunkle Knoten). Die Phasenmessung entlang des Rings ergibt m = 4,00: exakt vier Wellenlängen auf dem Umfang.

27.4 Kritische Kopplung: der perfekt schwarze Dip

Der Dip bei 0,1485 erreichte $T \approx 0{,}29$ — ordentlich, aber kein schwarzes Loch. Liegt es am Abstand? Rücken wir den Bus doch näher an den Ring, dann koppelt mehr.

Vorhersage-Punkt (PRIMM): Wir fahren den Spalt g von 0,2 bis 1,3 durch. Wird der Dip mit *kleinerem* Spalt (stärkerer Kopplung) monoton tiefer — je näher, desto schwärzer?

```
# von oben: bus_sweep()
SPALTE = [0.2, 0.4, 0.7, 1.0, 1.3]
T_min_g = []
for g in SPALTE:
    fg, p0 = bus_sweep(g, False, 0.149, 0.02)
    _, p1 = bus_sweep(g, True, 0.149, 0.02)
    T_min_g.append(float((p1 / p0).min()))
    print(f"Spalt g = {g}:  T_min = {T_min_g[-1]:.4f}")
```

```
Spalt g = 0.2:  T_min = 0.2859
Spalt g = 0.4:  T_min = 0.0565
Spalt g = 0.7:  T_min = 0.3255
Spalt g = 1.0:  T_min = 0.7651
Spalt g = 1.3:  T_min = 0.9399
```

```
# von oben: SPALTE, T_min_g
fig, ax = plt.subplots(figsize=(6.6, 4.0))
ax.plot(SPALTE, T_min_g, "C0o-", ms=7, lw=1.4)
ax.axvline(0.4, color="0.4", ls="--", lw=1.2)
ax.annotate("kritisch", (0.4, 0.5), xytext=(0.55, 0.55), fontsize=10,
            arrowprops=dict(arrowstyle="->", color="0.3"))
ax.text(0.21, 0.32, "überkoppelt", fontsize=9, color="0.35")
ax.text(0.95, 0.6, "unterkoppelt", fontsize=9, color="0.35")
ax.set_xlabel("Koppelspalt $g$")
ax.set_ylabel("Dip-Tiefe $T_{\\min}$")
ax.set_title("Kritische Kopplung: das V der Anpassung")
ax.set_ylim(0, 1)
ax.grid(alpha=0.3)
plt.show()
```

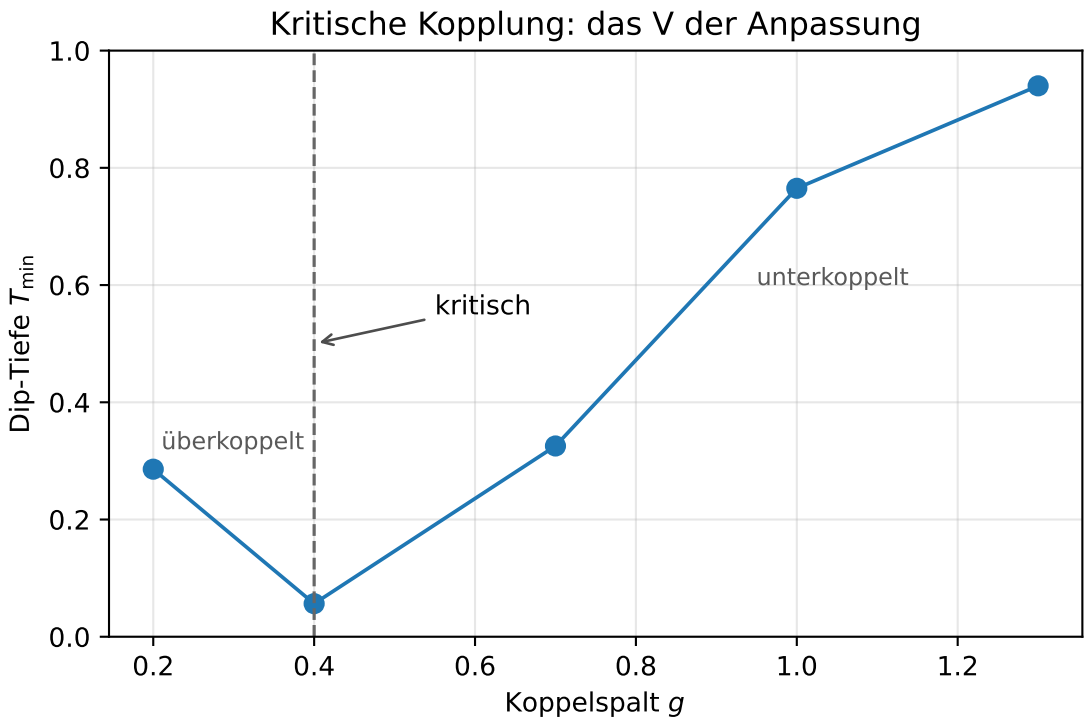


Abbildung 27.6: Die Tiefe des Transmissions-Dips über dem Koppelspalt: kein monotoner Verlauf, sondern ein V mit Minimum bei g = 0,4 (gestrichelt) — dort schluckt der Ring praktisch alles (T = 0,06). Links davon ist die Kopplung stärker als der Ringverlust (überkoppelt), rechts schwächer (unterkoppelt); ganz rechts merkt der Bus den Ring kaum noch.

Nein — die Kurve ist ein **V**. Der Dip wird erst tiefer, erreicht bei $g = 0{,}4$ fast die Null und füllt sich dann wieder. Dahinter steckt ein Wettkampf zweier Verlustkanäle: Der Ring verliert Licht durch seine **Krümmung** (Eigenverlust, fest) und zurück in den **Bus** (Koppelverlust, über g einstellbar). Was der Bus hinter dem Ring sieht, ist die Interferenz der direkt durchgelaufenen Welle mit dem Anteil, der den Ring besucht hat — und diese beiden löschen sich **exakt** aus, wenn beide Kanäle gleich stark sind: **kritische Kopplung**. Das ist Leistungsanpassung, Kapitel 22, nur in der Optik: Quelle und Verbraucher (hier: Biegeverlust) müssen zueinander passen, sonst bleibt Echo übrig. Und es ist derselbe Mechanismus, mit dem in Kapitel 15 das Koppel-Loch die Güte des Hohlraums regelte — dort haben wir den Resonator be- und entlüftet, hier stimmen wir ihn auf seinen eigenen Verlust ab.

Genau auf diesem V steht die Silizium-Photonik: Ein Ring nahe der kritischen Kopplung ist ein extrem empfindlicher Schalter. Verschiebt man sei-

ne Resonanz nur um eine Linienbreite — etwa indem Ladungsträger den Brechungsindex minimal ändern —, springt T am Arbeitspunkt zwischen fast 0 und fast 1: ein **Ring-Modulator**, das Bauteil, das in Rechenzentren elektrische Bits auf Licht prägt. Dieselbe Empfindlichkeit macht den Ring zum **Biosensor**: Ein Protein, das auf der Oberfläche andockt, verstimmt die Resonanz messbar.

27.5 Der Schritt zum Laser

Ein Resonator speichert Licht und verliert es langsam wieder — Rate $\omega/2Q$, gemessen. Was, wenn wir den Verlust nicht nur ausgleichen, sondern überbieten? Seit Kapitel 19 wissen wir, was dann passieren *muss*: Ein schwingungsfähiges System mit Energiezufuhr, die schneller nachschiebt als die Dämpfung zehrt, ist ein **selbsterregter Oszillator** — dort war es der van-der-Pol-Kreis an der Messerschneide $k = R$. Ein Laser ist exakt das: Resonator (haben wir) plus verstärkendes Medium (kommt jetzt).

Verstärkung bekommt Meep verblüffend billig: Kapitel 15 hat vermessen, dass `D_conductivity` $= \sigma$ das Feld mit der Rate $\sigma/2$ **dämpft** — also verstärkt ein *negatives* σ mit derselben Rate. Erst die Eichung im homogenen Medium (Versuchsaufbau: periodische 1D-Zelle, laufende Welle, ein Anstoß; Erfolgskriterium: Wachstumsrate exakt $\sigma/2$):

```
# von oben: AUFL
sim_h = mp.Simulation(
    cell_size=mp.Vector3(0, 0, 4), dimensions=1, resolution=AUFL,
    k_point=mp.Vector3(0, 0, 0.25),
    default_material=mp.Medium(D_conductivity=-0.05),
    sources=[mp.Source(mp.GaussianSource(0.25, fwidth=0.1),
                       component=mp.Ex, center=mp.Vector3(0, 0, 0.3))])
werte_h = []
sim_h.run(mp.after_sources(mp.at_every(0.5, lambda s: werte_h.append(
    (s.meep_time(), abs(s.get_field_point(mp.Ex, mp.Vector3(0, 0, 1.1))))))),
    until_after_sources=150)
t_h = np.array([w[0] for w in werte_h])
a_h = np.array([w[1] for w in werte_h])
m_h = a_h > 1e-12
print(f"Wachstumsrate: {np.polyfit(t_h[m_h], np.log(a_h[m_h]), 1)[0]:.5f}"
      f"   (sigma/2 = 0.02500)")
```

```
Wachstumsrate: 0.02500   (sigma/2 = 0.02500)
```

Exakt. Jetzt das eigentliche Experiment. **Frage:** Bei welchem σ springt der Resonator an? Die Vorhersage steht schon da: Die kalte Klingelrate war $-0{,}0424$ — Verstärkung gewinnt, sobald $\sigma/2$ das übertrifft, also bei $\sigma_{\text{th}} \approx 0{,}085$. **Bühne:** die FP-Kavität, gefüllt mit Gain-Medium; ein winziger Anstoß; dann messen wir die Lock-in-Rate der Mode für eine σ-Reihe. **Erfolgskriterium:** Raten auf einer Geraden, Nulldurchgang an der Vorhersage.

Und hier rettet uns der Lock-in tatsächlich den Hals (Werkstatt-Notiz): Die Gauß-Stromquelle hinterlässt in 1D ein winziges **statisches Restfeld** (ihr Strom-Integral ist nicht exakt null). Ein normaler Resonatorlauf merkt das nie — aber ein Gain-Medium verstärkt auch Gleichfelder, ohne jeden Spiegelverlust, und nach hundert Zeiteinheiten dominiert der wachsende DC-Sockel jede naive Hüllkurvenmessung. Der Lock-in auf f_{res} ist dafür taub.

```
# von oben: lockin_rate(), rate_kalt
sig_naiv = -2 * rate_kalt
SIGS = np.array([0.0, 0.5, 0.8, 1.0, 1.2, 1.5]) * sig_naiv
raten = np.array([rate_kalt] + [lockin_rate(s) for s in SIGS[1:]])
for s_v, r_v in zip(SIGS, raten):
    print(f"sigma = {s_v:.4f}:  Rate = {r_v:+.5f}")

iw = int(np.flatnonzero(raten > 0)[0])
sig_th = SIGS[iw - 1] - raten[iw - 1] * (SIGS[iw] - SIGS[iw - 1]) \
    / (raten[iw] - raten[iw - 1])
print(f"\nSchwelle (Nulldurchgang): sigma_th = {sig_th:.4f}")
print(f"Vorhersage 2 x kalte Rate: {sig_naiv:.4f}  "
      f"(Abweichung {100 * (sig_th / sig_naiv - 1):+.1f} %)")
```

```
sigma = 0.0000:  Rate = -0.04236
sigma = 0.0424:  Rate = -0.02314
sigma = 0.0678:  Rate = -0.01128
sigma = 0.0847:  Rate = -0.00290
sigma = 0.1017:  Rate = +0.00609
sigma = 0.1271:  Rate = +0.02465

Schwelle (Nulldurchgang): sigma_th = 0.0902
Vorhersage 2 x kalte Rate: 0.0847  (Abweichung +6.4 %)
```

```
# von oben: SIGS, raten, sig_th, sig_naiv, rate_kalt
fig, ax = plt.subplots(figsize=(6.6, 4.0))
ax.plot(SIGS, raten, "C0o", ms=7, label="Meep (Lock-in-Rate)")
sgrid = np.linspace(0, SIGS[-1], 50)
ax.plot(sgrid, rate_kalt + sgrid / 2, "0.4", ls=":", lw=1.3,
```

```
        label="Referenz: kalte Rate + $\\sigma/2$")
ax.axvline(sig_naiv, color="0.4", ls="--", lw=1.2,
           label="2 × kalte Verlustrate")
ax.axhline(0, color="0.6", lw=1)
ax.plot([sig_th], [0], "C3*", ms=14, label=f"Schwelle {sig_th:.4f}")
ax.set_xlabel("Verstärkung $\\sigma$")
ax.set_ylabel("Wachstumsrate des Feldes")
ax.set_title("Der Laser springt am Verlustgleichstand an")
ax.legend(fontsize=8, loc="upper left")
ax.grid(alpha=0.3)
plt.show()
```

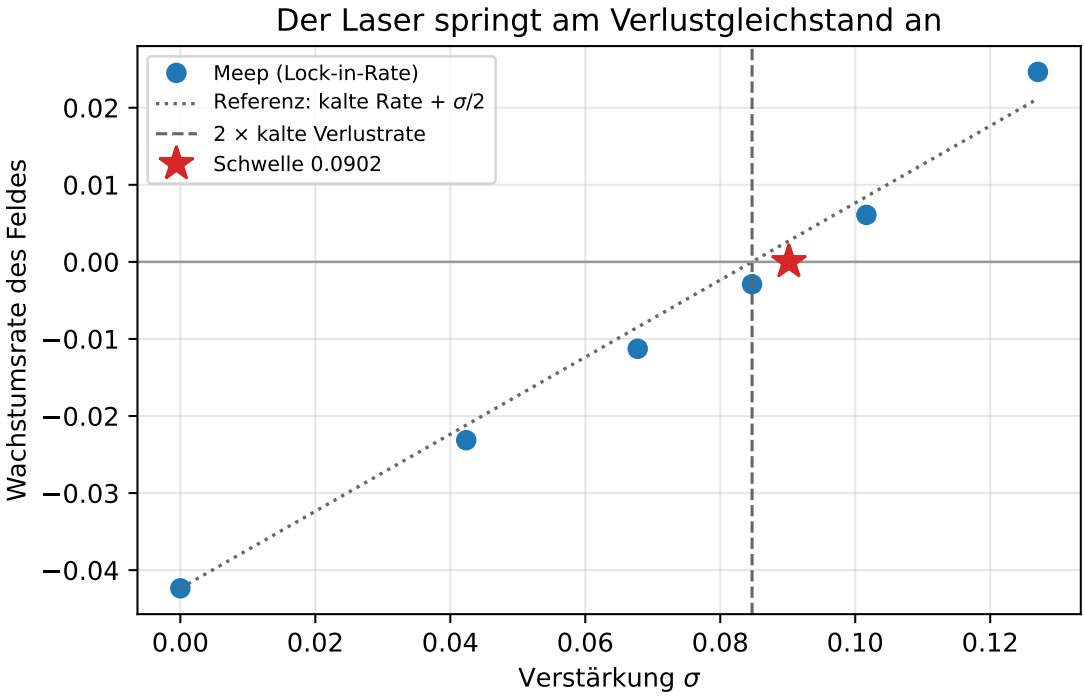


Abbildung 27.7: Die Wachstumsrate der Resonatormode über der Verstärkung: unterhalb der Schwelle klingt jeder Anstoß ab, oberhalb schaukelt er sich exponentiell auf. Der Nulldurchgang liegt sechs Prozent über der einfachen Vorhersage „2 × kalte Verlustrate“ (gestrichelt) — die Mode wohnt zu einem Teil in den Spiegeln, wo nichts verstärkt. Die Steigungs-Referenz σ/2 (gepunktet) trifft den Verlauf nahe der Schwelle.

Oberhalb der Schwelle wächst das Feld in unserer Simulation **unbegrenzt**

— exponentiell, für immer. Ein echter Laser tut das offensichtlich nicht, und was fehlt, kennt Kapitel 19 genau: die **Sättigung**. Reale Verstärkung speist sich aus angeregten Atomen, und je stärker das Feld, desto schneller räumt es sie ab — die Verstärkung sinkt mit der Intensität, $G = G_0/(1 + I/I_{\text{sat}})$. Genau wie beim van-der-Pol-Oszillator stabilisiert das die Amplitude: Das Feld wächst, bis die gesättigte Verstärkung die Verluste *exakt* deckt, und bleibt dort — der Grenzzyklus. Ein Umlauf-Modell mit dieser einen Zutat liefert die berühmteste Kurve der Laserphysik:

```
# Umlauf-Modell: pro Umlauf Verstaerkung (gesaettigt), dann Verlust
def stationaere_intensitaet(G0, verlust=0.2, n_umlauf=4000):
    I = 1e-9                        # winziger Anstoss ("Rauschen")
    for _ in range(n_umlauf):
        I = I * (1 + G0 / (1 + I)) * (1 - verlust)
    return I


G_reihe = np.linspace(0.05, 0.6, 23)
I_stat = np.array([stationaere_intensitaet(G) for G in G_reihe])
V = 0.2
G_schwelle = V / (1 - V)
I_theo = np.maximum(0.0, G_reihe * (1 - V) / V - 1)

fig, ax = plt.subplots(figsize=(6.6, 4.0))
ax.plot(G_reihe, I_stat, "C0o", ms=6, label="Umlauf-Modell (4000 Umläufe)")
ax.plot(G_reihe, I_theo, "C1-", lw=1.5, label="stationäre Lösung")
ax.axvline(G_schwelle, color="0.4", ls="--", lw=1.2,
           label=f"Schwelle $G_0$ = {G_schwelle:.2f}")
ax.set_xlabel("Kleinsignal-Verstärkung $G_0$ pro Umlauf")
ax.set_ylabel("stationäre Intensität $I/I_{sat}$")
ax.set_title("Die Laser-Kennlinie: aus — Knick — an")
ax.legend(fontsize=9, loc="upper left")
ax.grid(alpha=0.3)
plt.show()
```

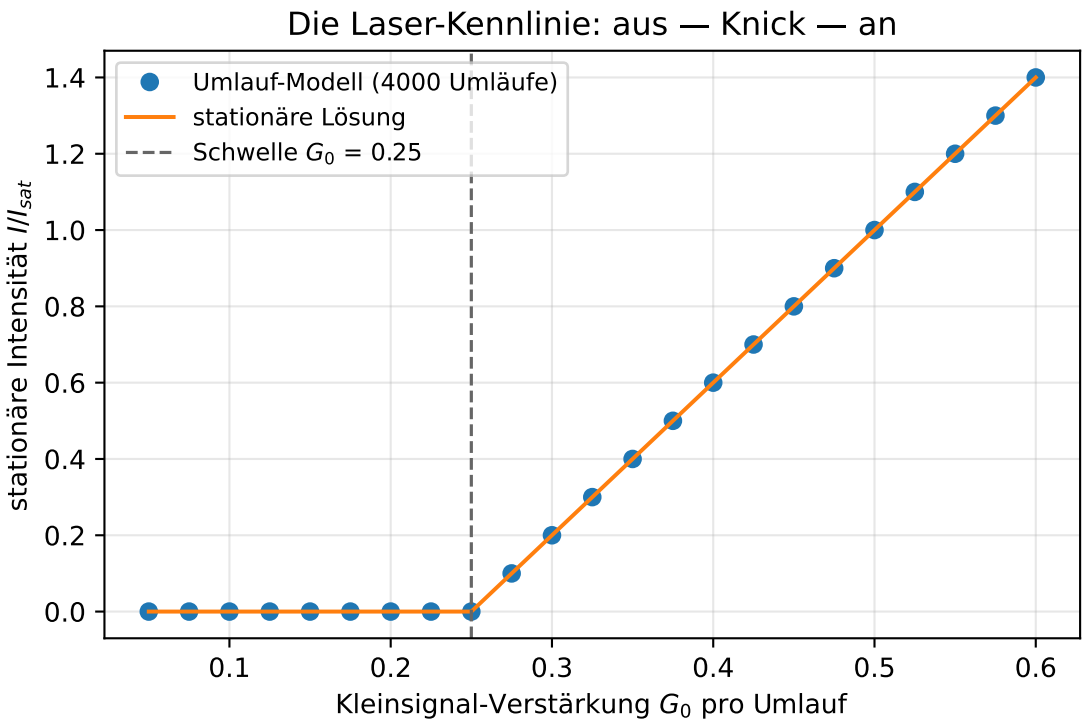


Abbildung 27.8: Die Laser-Kennlinie aus dem Umlauf-Modell (Punkte): Unterhalb der Schwelle bleibt die stationäre Intensität null — alles Klingeln stirbt aus. An der Schwelle (Verstärkung = Verlust) knickt die Kurve scharf ab und steigt dann linear: Jedes zusätzliche Quäntchen Pumpleistung wird zu Licht. Die Linie ist die stationäre Lösung $G_0/(1+I) \cdot (1-V) = V$ von Hand.

Mehr als dieses Umlauf-Modell und die Schwellen-Messung soll der Ausblick nicht behaupten: Ein echter Laser hat Atome mit Niveaus, spektral geformte Verstärkung und Rauschen, das die allererste Welle liefert. Aber die Architektur steht komplett im Buch: **Resonator (Kapitel 15/27) + gesättigte Verstärkung (Kapitel 19) = Oszillator für Licht.** Und der Frequenz-Kamm erklärt eine letzte Alltagsbeobachtung: Ein Laser schwingt nicht auf „einer Farbe", sondern auf den Kammzähnen, die in sein Verstärkungsfenster fallen — bei einem Helium-Neon-Rohr sind das typisch zwei, drei Längsmoden (Übung 27.1).

27.6 Das Kapitel-Programm

`programme/kap27/kap27_ringresonator_fabryperot.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die Phasenuhr-Probe und Echo-Summe == TMM == Airy (je 10^{-15}), Kamm (FSR 0,0892, $L_{\text{eff}} = 5{,}61$, Finesse 6,6), den Meep-Sweep (rms 0,0014), das doppelte Q (21,9/21,7), die Feldüberhöhung (2,86 vs. 3,0), die homogene Gain-Eichung ($\sigma/2$ exakt), die Laser-Schwelle (±10 % am Verlustgleichstand), die Ringmoden samt monotonem Biege-Q, die Bus-Dips auf den Modenfrequenzen, das V der kritischen Kopplung und den Kennlinien-Knick. Läuft in wenigen Minuten.

Merkkasten

- **Fabry-Pérot = Echo-Summe:** geometrische Reihe der Umläufe; auf Resonanz löscht das Kavitätsleck die Reflexion aus → $T = 1$ durch zwei 64-%-Spiegel. Kamm mit FSR $= c/2L_{\text{eff}}$ (die Welle wohnt ein Stück im Spiegel), Trennschärfe = Finesse $\pi\sqrt{R}/(1-R)$.
- **Im Resonator staut sich Feld:** Überhöhung $(1+\sqrt{R})\sqrt{1-R}/(1-R)$ — der Grund, warum LIGO und Laserkavitäten falten statt verlängern.
- **Ring = Fabry-Pérot ohne Spiegel:** Resonanz, wenn der Umfang m Wellenlängen trägt; Verlustkanal ist die Krümmung (Biege-Q wächst exponentiell mit Radius und Frequenz); Ankopplung evaneszent über den Kapitel-17-Richtkoppler.
- **Kritische Kopplung:** Der Bus-Dip wird genau dann schwarz, wenn Koppelverlust = Eigenverlust — Leistungsanpassung (Kapitel 22) in der Photonik; Arbeitspunkt von Modulatoren und Sensoren.
- **Laser = Resonator + gesättigte Verstärkung:** Schwelle exakt am Verlustgleichstand (gemessen); die Sättigung macht aus exponentiellem Wachstum einen Grenzzyklus (Kapitel 19) — daher der Knick der Kennlinie.

Roter Faden

Dieses Kapitel ist ein Sammelbecken: die **Echo-Geometrie aus Kapitel 6/22** (geometrische Reihe — von der Lampen-Treppe zum Airy-Kamm), die **Transfer-Matrix und Defektnadel aus Kapitel 26** (Übung 26.3 war dieses Kapitel im Kleinformat), **Güte und Harminv aus Kapitel 15** (Q dreifach, Koppel-Loch → kritische Kopplung), der **Richtkoppler aus Kapitel**

17 (Zusage eingelöst: er speist jetzt Ringe), die **Anpassung aus Kapitel 22** (kritische Kopplung als optisches SWR = 1) und der **van-der-Pol-Oszillator aus Kapitel 19** (Schwelle, Sättigung, Grenzzyklus — jetzt mit Licht). Nach vorn: **Kapitel 28** fragt, was passiert, wenn man Materialien selbst zu Schaltungen macht — Metamaterialien, negativer Brechungsindex und Doppelbrechung; der Ringresonator kehrt dort als Baustein zurückgerollter Wellen nicht wieder, wohl aber die Frage, wie weit man ε und μ verbiegen darf.

Übungen

Ü 27.1 — Kammzählen im Gerätepark (Verstehen). Rechne den freien Spektralbereich dreier realer Resonatoren aus (FSR $= c/(2nL)$): (a) eine Laserdiode ($L = 0{,}3$ mm, $n = 3{,}5$), (b) ein Helium-Neon-Laser ($L = 30$ cm, $n \approx 1$) — wie viele Längsmoden passen in sein 1,5 GHz breites Verstärkungsfenster? (c) ein LIGO-Arm ($L = 4$ km). Was bedeutet das winzige FSR von (c) praktisch?

Musterlösung zu Ü 27.1

```python
C = 299792458.0
for name, L_r, n_r in [("Laserdiode", 0.3e-3, 3.5),
                       ("HeNe-Rohr", 0.30, 1.0),
                       ("LIGO-Arm", 4000.0, 1.0)]:
    fsr = C / (2 * n_r * L_r)
    print(f"{name:11s}: FSR = {fsr:.3e} Hz")
print(f"\nHeNe: {1.5e9 / (C / (2 * 0.30)):.1f} Moden im Gain-Fenster")
```

```
Laserdiode : FSR = 1.428e+11 Hz
HeNe-Rohr  : FSR = 4.997e+08 Hz
LIGO-Arm   : FSR = 3.747e+04 Hz

HeNe: 3.0 Moden im Gain-Fenster
```

(a) Die Laserdiode hat 143 GHz Modenabstand — ihr Verstärkungsfenster ist zwar Terahertz breit, aber mit Tricks (Gitter, DFB) zwingt man sie auf einen einzigen Zahn. (b) Beim HeNe passen **drei** Moden ins Fenster: Ein gewöhnlicher Laborlaser schwingt auf zwei bis drei Frequenzen gleichzeitig, die um 500 MHz auseinanderliegen — sichtbar als Schwebung auf einer schnellen Photodiode.

(b) LIGOs 37,5 kHz bedeuten: Der Kamm ist so fein, dass der Laser

aktiv auf einem Zahn *festgenagelt* werden muss (Pound-Drever-Hall-Regelung) — die Kavität ist dort Lineal, nicht Filter.

Ü 27.2 — Die Finesse-Reihe (Verändern). Variiere die Spiegeldicke ($0{,}3$ bis $1{,}0 \times \lambda/4$) und miss mit der Transfer-Matrix Finesse und Spitzenhöhe des Kamms. Lege $\pi\sqrt{R}/(1-R)$ als Theorielinie darüber (R jeweils des Einzelspiegels bei f_0). Wo weicht die Messung ab, und in welche Richtung?

Musterlösung zu Ü 27.2

```python
# von oben: schichtmatrix(), rt_von(), F0, N_SP, D_SP, L


def kamm_finesse(dicke_faktor):
    d = dicke_faktor * D_SP

    def T_f(f):
        M = schichtmatrix(N_SP, d, f) @ schichtmatrix(1.0, L, f) \
            @ schichtmatrix(N_SP, d, f)
        return np.abs(rt_von(M)[1])**2

    f_raster = np.linspace(0.15, 0.35, 4001)
    Tk = np.array([T_f(f) for f in f_raster])
    iks = [i for i in argrelmax(Tk)[0] if Tk[i] > 0.5]
    # FWHM an der Resonanz nächst f0 messen — nicht am Rasterrand!
    ip = iks[int(np.argmin(np.abs(f_raster[iks] - F0)))]
    ueber = Tk > Tk[ip] / 2
    lo = hi = ip
    while lo > 0 and ueber[lo - 1]:
        lo -= 1
    while hi < len(Tk) - 1 and ueber[hi + 1]:
        hi += 1
    fsr_l = float(np.diff(f_raster[iks]).mean())
    return fsr_l / (f_raster[hi] - f_raster[lo])


fakt = np.array([0.3, 0.45, 0.6, 0.8, 1.0])
fin_mess = [kamm_finesse(fk) for fk in fakt]
R_reihe = np.array([np.abs(rt_von(schichtmatrix(N_SP, fk * D_SP, F0))[0])**2
                    for fk in fakt])
fin_theo = np.pi * np.sqrt(R_reihe) / (1 - R_reihe)

fig, ax = plt.subplots(figsize=(6.4, 3.8))
ax.plot(R_reihe, fin_mess, "C0o", ms=7, label="Transfer-Matrix-Kamm")
ax.plot(R_reihe, fin_theo, "C1-", lw=1.5,
        label="ideal: $\\pi\\sqrt{R}/(1-R)$")
ax.set_xlabel("Spiegelreflexion $R$ (bei $f_0$)")
ax.set_ylabel("Finesse")
ax.set_title("Trennschärfe wächst mit dem Spiegel")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

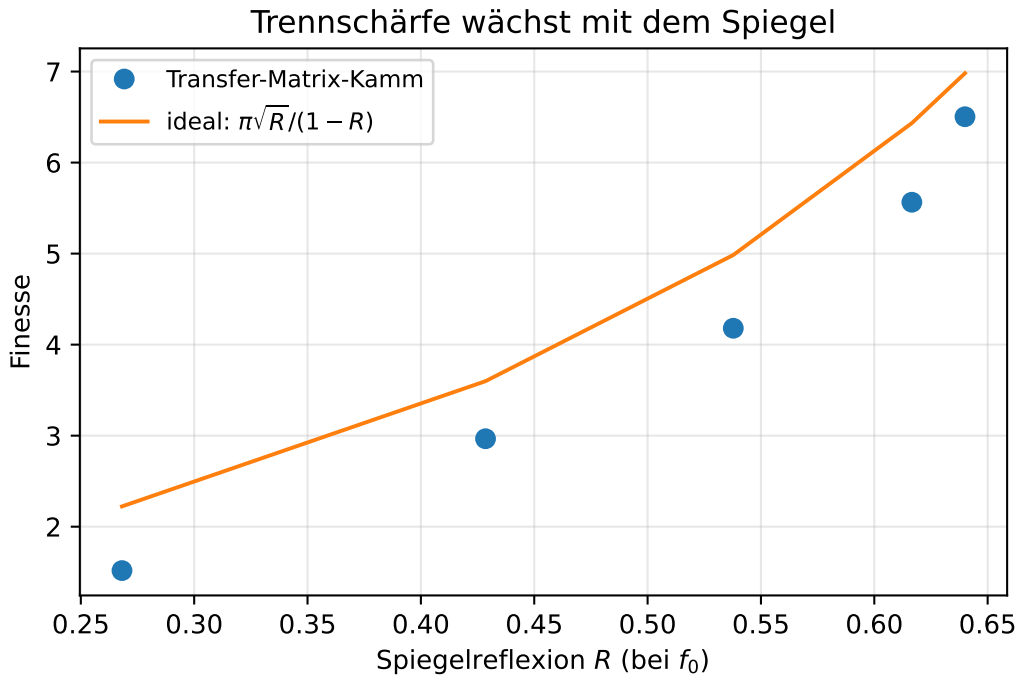


Abbildung 27.9: Gemessene Finesse (Punkte) gegen die Idealformel (Linie): Die Messung liegt durchweg leicht darunter, und der Abstand wächst mit R. Grund ist die Frequenzabhängigkeit der realen Spiegel — über die (schmaler werdende) Linie ist R nicht konstant und die Reflexionsphase dreht sich mit; die Idealformel kennt beides nicht.

Die Messpunkte liegen systematisch **unter** der Idealkurve (bei $R = 0{,}64$: 6,5 statt 7,0). Die Formel nimmt an, dass R und die Spiegelphase über die Resonanzlinie konstant sind — echte (dispersive) Spiegel verschmieren die Linie zusätzlich. Die Lehre ist dieselbe wie bei der Dickdraht-Antenne in Kapitel 21: Idealformeln sind Leitplanken, die Bauteile dahinter haben Charakter.

Ü 27.3 — Die Kanalweiche (Übertragen). Lege einen **zweiten** Bus-Wellenleiter oben an den Ring (gleicher Spalt 0,2) und miss beide Ausgänge: die Transmission im unteren Bus und die Leistung, die im oberen Bus herauskommt (**Add-Drop-Filter**). Zwei Vorhersagen, bevor du startest: Was passiert auf Resonanz mit der Leistung — und aus *welchem Ende* des oberen Leiters kommt sie heraus? (Verfolge den Drehsinn des Kreisverkehrs: Die Bus-Welle läuft nach rechts und gibt ihn dem Ring vor.)

Musterlösung zu Ü 27.3

```
# von oben: N_WG, W, R_IN, DPML, bus_sweep()
SPALT3 = 0.2
pad3 = 3.0
sx3 = 2 * (R_IN + W + pad3 + DPML)
y_unten = -(R_IN + W + SPALT3 + W / 2)
y_oben = +(R_IN + W + SPALT3 + W / 2)
geom3 = [mp.Block(size=mp.Vector3(mp.inf, W, mp.inf),
                  center=mp.Vector3(0, y_unten),
                  material=mp.Medium(index=N_WG)),
         mp.Block(size=mp.Vector3(mp.inf, W, mp.inf),
                  center=mp.Vector3(0, y_oben),
                  material=mp.Medium(index=N_WG)),
         mp.Cylinder(radius=R_IN + W, material=mp.Medium(index=N_WG)),
         mp.Cylinder(radius=R_IN, material=mp.air)]
sim3 = mp.Simulation(
    cell_size=mp.Vector3(sx3, sx3), resolution=16,
    boundary_layers=[mp.PML(DPML)], geometry=geom3,
    sources=[mp.Source(mp.GaussianSource(0.149, fwidth=0.02),
                       component=mp.Ez,
                       center=mp.Vector3(-sx3 / 2 + DPML + 0.5, y_unten),
                       size=mp.Vector3(0, 2 * W))])
x_m3 = sx3 / 2 - DPML - 0.5
tr_durch = sim3.add_flux(0.149, 0.02, 401,
                         mp.FluxRegion(center=mp.Vector3(x_m3, y_unten),
                                       size=mp.Vector3(0, 2 * W)))
# Drop-Port LINKS oben: weight=-1 zaehlt den Fluss nach -x positiv
tr_drop = sim3.add_flux(0.149, 0.02, 401,
                        mp.FluxRegion(center=mp.Vector3(-x_m3, y_oben),
                                      size=mp.Vector3(0, 2 * W), weight=-1))
sim3.run(until_after_sources=mp.stop_when_fields_decayed(
    100, mp.Ez, mp.Vector3(x_m3, y_unten), 1e-8))

fs3 = np.array(mp.get_flux_freqs(tr_durch))
_, P0_3 = bus_sweep(SPALT3, False, 0.149, 0.02)
T_durch = np.array(mp.get_fluxes(tr_durch)) / P0_3
T_drop = np.array(mp.get_fluxes(tr_drop)) / P0_3
i3 = np.argmin(T_durch)
print(f"auf Resonanz (f = {fs3[i3]:.4f}):  "
      f"T_durch = {T_durch[i3]:.3f},  T_drop = {T_drop[i3]:.3f}")
print(f"daneben (f = {fs3[0]:.4f}):       "
      f"T_durch = {T_durch[0]:.3f},  T_drop = {T_drop[0]:.4f}")
```

```
auf Resonanz (f = 0.1494):  T_durch = 0.039,  T_drop = 0.733
daneben (f = 0.1390):       T_durch = 0.966,  T_drop = 0.0273
```

Auf Resonanz wechselt fast alles die Spur: 73 % der Leistung verlassen den **oberen** Leiter, im unteren bleiben 4 % — daneben läuft alles ungerührt unten durch. Und die Richtung: Die Bus-Welle läuft nach $+x$ und prägt dem Ring einen Drehsinn auf (gegen den Uhrzeigersinn); oben angekommen läuft die umlaufende Welle nach $-x$, also koppelt sie in den oberen Leiter **nach links** aus — der Drop-Port liegt diagonal gegenüber dem Eingang. Genau so sortieren WDM-Filter im Glasfasernetz einzelne Farbkanäle aus einem Bündel heraus: ein Ring pro Kanal, jeder auf seine Frequenz gestimmt.

Ü 27.4 — Der Preis der Kurve (Übertragen). Miss die Ringmoden für drei Ringgrößen ($r_{\text{innen}} = 0{,}6$, $1{,}0$ und $1{,}5$) und verfolge jeweils die Mode, die am nächsten an $f = 0{,}147$ liegt. Vorhersage: Wie ändert sich ihr Q — linear mit dem Radius oder schneller? Und was sagt das über das Knicken von Glasfasern?

Musterlösung zu Ü 27.4

```python
# von oben: ring_moden()
print("Mode nächst f = 0,147 je Ringgröße:")
for r_in in [0.6, 1.0, 1.5]:
    mm = ring_moden(r_in, 0.15, 0.1)
    f_n, q_n = min(mm, key=lambda m: abs(m[0] - 0.147))
    print(f"  r_innen = {r_in}:  f = {f_n:.4f},  Q = {q_n:.0f}")
```

```
Mode nächst f = 0,147 je Ringgröße:
  r_innen = 0.6:  f = 0.1463,  Q = 84
  r_innen = 1.0:  f = 0.1473,  Q = 344
  r_innen = 1.5:  f = 0.1420,  Q = 1006
```

$Q = 84 \to 344 \to 1006$: grob **Verdreifachung je halbe Radiuseinheit** — exponentiell, nicht linear. Der Biegeverlust ist ein Tunnelprozess (der evaneszente Saum muss „hinter" den Radius gelangen, ab dem er nicht mehr mitkommt), und Tunnelraten hängen exponentiell von der Barriere ab — Kapitel 17, zum dritten Mal. Für die Glasfaser heißt das: Ein sanfter Bogen (Zentimeter) kostet praktisch nichts, ein scharfer Knick (Millimeter) wirft das Licht fast vollständig hinaus. Deshalb schreiben Faserhersteller Mindest-Biegeradien vor — und deshalb führte in Kapitel 26 nur die Bandlücke, nicht die Totalreflexion, das Licht um die scharfe Ecke.

Das Kleingedruckte

LIGO im Fabry-Pérot-Vokabular. Die 4-km-Arme sind Kavitäten mit Finesse ≈ 450: Das Licht läuft effektiv ~280-mal hin und her, die zirkulierende Leistung erreicht Hunderte Kilowatt, und die Resonanzbedingung verwandelt eine Armlängenänderung von 10^{-19} m in eine messbare Phasendrehung. Dazu kommt „Power Recycling": Auch der Rest des Interferometers sitzt in einem Resonator, der das ungenutzte Licht zurückwirft — Feldüberhöhung im Quadrat.

Echte Lasersimulation. Unser lineares Gain kennt weder Sättigung noch Spektrum. Meep kann beides: Das `MultilevelAtom`-Material koppelt die Felder an Besetzungszahlen von Atomniveaus (Ratengleichungen) — damit schwingt eine Simulation wirklich an, sättigt und zeigt Modenkonkurrenz. Der Preis sind mehrere zusätzliche Parameter (Lebensdauern, Dipolstärken, Pumprate) und deutlich längere Läufe; für das Schwellen-Argument dieses Kapitels wäre es Overkill.

Gires-Tournois und der verstimmte Spiegel. Setzt man hinter die Kavität einen *perfekten* Spiegel, kommt alles Licht zurück ($|r| = 1$ immer) — aber die **Phase** dreht sich nahe der Resonanz rasend schnell. Solche Gires-Tournois-Etalons sind reine Phasen-Bauteile, mit denen man Pulse staucht (Dispersionskompensation): Der Resonator als Verzögerungsleitung, Kapitel 16 lässt grüßen.

Frequenzkämme wörtlich. Ein Laser, der *viele* Längsmoden phasenstarr koppelt (Modenkopplung), sendet einen Kamm aus Tausenden exakt äquidistanten Linien — das Lineal, mit dem man heute optische Frequenzen zählt (Nobelpreis 2005, Hänsch/Hall). Der Zahnabstand ist nichts anderes als unser FSR.

28 Metamaterialien und Doppelbrechung

1968 stellte der sowjetische Physiker Victor Veselago eine Frage, die sich auf einer Briefmarke formulieren lässt: *Was wäre, wenn ein Material gleichzeitig negatives ε und negatives μ hätte?* Seine Antwort auf dem Papier war verblüffend — Wellen liefen darin rückwärts, Brechung würde auf die falsche Seite knicken, eine flache Platte würde zur Linse. Und dann passierte: nichts. 32 Jahre lang blieb die Arbeit eine Kuriosität, denn kein bekanntes Material der Welt hat beides zugleich. Erst im Jahr 2000 baute die Gruppe um David Smith in San Diego ein „Material", das es doch kann — zusammengesetzt aus Kupferdrähten und kleinen Ring-Schwingkreisen, beides Zentimeter groß. Der Trick: Wenn die Bausteine viel kleiner sind als die Wellenlänge, fragt die Welle nicht, woraus ein Medium besteht. Sie spürt nur gemittelte Antworten — ein effektives ε, ein effektives μ. Solche maßgeschneiderten Kunst-Medien heißen **Metamaterialien**.

Das ist die Gegenrichtung zu Kapitel 26: Dort waren die Bausteine *vergleichbar* mit der Wellenlänge ($a \sim \lambda$), und alles drehte sich um Interferenz — Bandlücken, Bragg-Spiegel. Hier sind die Bausteine *viel kleiner* ($a \ll \lambda$), und die Welle sieht ein homogenes Medium mit Eigenschaften, die kein Naturstoff liefert. In diesem Kapitel bauen wir Veselagos Medium in Meep nach und messen alle drei Vorhersagen. Danach drehen wir den Spieß um und schauen auf Materialien, die *zwei verschiedene* Brechungsindizes gleichzeitig haben — je nachdem, wie das Licht polarisiert ist: **Doppelbrechung**, vom Kalkspat-Doppelbild bis zur λ/4-Platte, die aus linearem Licht das zirkulare macht, das wir in Kapitel 4 versprochen haben.

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum $\varepsilon < 0$ **allein** eine Welle sperrt (Kapitel 16), $\varepsilon < 0$ **und** $\mu < 0$ sie aber wieder laufen lassen — und warum dann $n = -\sqrt{\varepsilon\mu}$ die richtige Wahl ist (Energie vorwärts, Phase rückwärts),

2. … das historische Smith-Experiment nachbauen: Drähte allein sperren, Ringe allein sperren, beide zusammen öffnen ein **Durchlassfenster** — der Fingerabdruck des negativen Index,
3. … negative Brechung und die **Veselago-Flachlinse** simulieren und weißt, woran reale Metamaterialien kranken (Verluste wohnen an der Resonanz, Kapitel 15),
4. … **anisotrope Materialien** (ε als Tensor) in Meep ansetzen und Doppelbrechung messen: zwei Polarisationen, zwei Brechungswinkel, zwei Indizes,
5. … das Kalkspat-**Doppelbild** erklären (Walk-off: der außerordentliche Strahl wandert seitlich, obwohl er senkrecht einfällt) und eine **λ/4-Platte** dimensionieren, die linear polarisiertes Licht zirkular macht.

28.1 Verbotene Vorzeichen?

Fangen wir mit dem an, was wir schon wissen. Die Wellengleichung aus Kapitel 4 verknüpft die Wellenzahl mit den Materialkonstanten:

$$k^2 = \omega^2 \varepsilon_0 \mu_0 \, \varepsilon\mu \quad \Longrightarrow \quad n^2 = \varepsilon\mu.$$

Ist **nur** ε negativ, wird $k^2 < 0$ — die Wellenzahl ist imaginär, die Welle klingt exponentiell ab, statt zu laufen. Genau das haben wir zweimal gemessen: an der Ionosphären-Klippe in Kapitel 16 (Drude-Medium unter der Plasmafrequenz) und im Rohr unter dem Cutoff in Kapitel 17. Ein Medium mit $\varepsilon < 0$ ist ein Spiegel, kein Leiter.

Aber jetzt Veselagos Schachzug: Sind ε **und** μ negativ, ist ihr Produkt wieder positiv — $k^2 > 0$, die Welle läuft! Und hier sitzt die Falle, die sich als nächstes aufdrängt:

Naheliegende Vermutung: „n = √((−1)·(−1)) = +1 — also verhält sich das Medium wie Vakuum“

Warum sie naheliegt: $n^2 = \varepsilon\mu = (-1)(-1) = 1$, und die Schulregel sagt: Wurzel ziehen, fertig, $n = 1$. Dann wäre das Doppel-Minus-Medium von Vakuum nicht zu unterscheiden — Veselagos Frage wäre langweilig.
Was stattdessen stimmt: Die Wurzel hat zwei Zweige, und Maxwell wählt für uns. Schreibe die beiden Rotationsgleichungen für eine ebene Welle (Kapitel 4): $\vec{k} \times \vec{E} = \omega\mu_0\mu\vec{H}$ und $\vec{k} \times \vec{H} = -\omega\varepsilon_0\varepsilon\vec{E}$. Bei positiven Materialkonstanten bilden $\vec{E}, \vec{H}, \vec{k}$ ein **rechtshändiges** Dreibein — und

der Energiefluss $\vec{S} = \vec{E} \times \vec{H}$ zeigt entlang $\vec{k}$. Drehst du die Vorzeichen von ε *und* μ um, klappen beide Gleichungen um: Das Dreibein wird **linkshändig**, $\vec{S}$ zeigt jetzt *entgegen* $\vec{k}$. Die Energie fließt von der Quelle weg (das muss sie), also zeigt die Phasenausbreitung **zur Quelle hin**. Genau das bedeutet $n = -1$: Snellius gilt weiter, aber mit negativem Winkel — und die Phase läuft rückwärts. Veselago nannte solche Stoffe darum „linkshändige Medien".

Eine Sorge müssen wir noch ausräumen, bevor wir bauen: Darf es ein Medium mit $\varepsilon = \mu = -1$ überhaupt geben? Die Energiedichte aus Kapitel 18, $u = \frac{1}{2}\varepsilon_0\varepsilon E^2 + \frac{1}{2}\mu_0\mu H^2$, wäre damit **negativ** — ein Feld, das Energie erzeugt, wenn man es anfacht. Das wäre tatsächlich Unsinn. Die Auflösung: Diese Energieformel gilt nur für Konstanten ohne Frequenzabhängigkeit. In einem **dispersiven** Medium gilt stattdessen $u = \frac{1}{2}\varepsilon_0 \frac{\mathrm{d}(\omega\varepsilon)}{\mathrm{d}\omega} E^2 + \ldots$ — und diese Ableitungen sind bei unseren Drude- und Lorentz-Modellen positiv, auch wo ε und μ selbst negativ sind. Merksatz: **Negative Materialkonstanten gibt es nur mit Dispersion.** Ein $n = -1$ bei *allen* Frequenzen wäre unphysikalisch; ein $n = -1$ bei *einer* Frequenz ist erlaubt. Warum Kausalität und Dispersion so eng verheiratet sind, ist das Thema von Kapitel 29 (Kramers-Kronig).

28.2 Die Zutaten: Drähte und Ringe

Woher nehmen, wenn die Natur es nicht liefert? Smiths Antwort von 2000 besteht aus zwei Bausteinen, die wir beide schon kennen:

Baustein 1 — das Drahtgitter (macht $\varepsilon < 0$). Ein Gitter aus dünnen Metalldrähten, Abstand viel kleiner als die Wellenlänge, wirkt auf eine Welle mit $\vec{E}$ parallel zu den Drähten wie ein **verdünntes Plasma**: Die Elektronen können den Drähten entlang schwingen, aber die Induktivität der dünnen Drähte (Kapitel 22) macht sie träge. Das Ergebnis ist genau die Drude-Antwort aus Kapitel 16,

$$\varepsilon(f) = 1 - \frac{f_p^2}{f^2 + \mathrm{i}\gamma f},$$

nur dass die „Plasmafrequenz" f_p jetzt keine Materialkonstante ist, sondern eine **Stellschraube**: Drahtabstand und Drahtdicke legen sie fest. Unterhalb von f_p ist $\varepsilon < 0$ — die Ionosphäre aus dem Baumarkt. (In Übung 28.3 baust du dieses Gitter aus echten PEC-Stäbchen und misst seine Klippe.)

Baustein 2 — der Split-Ring (macht $\mu < 0$). Ein aufgeschnittener Metallring ist ein **Schwingkreis** im Sinn von Kapitel 19: Der Ring ist die Induktivität L, der Schlitz der Kondensator C. Das Magnetfeld der Welle durchsetzt den Ring und treibt einen Kreisstrom — ein getriebener Resonator (Kapitel 15). Und jetzt kommt der Phasensprung an der Resonanz ins Spiel, den wir dort gemessen haben: **Knapp oberhalb** der Ringresonanz f_0 antwortet der Strom *gegenphasig* — der Ring baut ein Magnetfeld auf, das dem antreibenden Feld entgegensteht, und zwar stärker, als das Vakuum mitmacht. Die gemittelte Antwort ist ein Lorentz-Term im μ:

$$\mu(f) = 1 + \frac{\sigma_R f_0^2}{f_0^2 - f^2 - \mathrm{i}\gamma f},$$

der in einem Band knapp über f_0 negativ wird. Anders als beim Draht (Drude: träge Elektronen, $\varepsilon < 0$ für alles unter f_p) ist $\mu < 0$ also ein **Resonanz-Phänomen** — es lebt nur in einem schmalen Band, und es wohnt zwangsläufig dort, wo auch die Verluste wohnen (Kapitel 15: an der Resonanz wird absorbiert). Das wird uns noch beschäftigen.

Wir setzen beide Modelle in Meep an. Neu ist nur eines: Dispersion im μ schreibt man als `H_susceptibilities` — das exakte Gegenstück zu den `E_susceptibilities` aus Kapitel 16.

Zur Wahl der Zahlen: Wir arbeiten bei der Betriebsfrequenz $f_{\mathrm{op}} = 1$ (also $\lambda = 1$; Skaleninvarianz wie immer seit Kapitel 11 — dieselbe Rechnung gilt für 10 GHz wie für 300 THz). Die Drude-Stärke wählen wir so, dass $\varepsilon(1) = -1$ wird, die Ringresonanz legen wir auf $f_0 = 0{,}8$ mit einer Stärke, die $\mu(1) = -1$ liefert. Kleine Verluste $\gamma = 0{,}01$ sind Absicht: Sie machen das Modell ehrlich (echte Metamaterialien sind verlustig) — und sie halten die Simulation stabil.

```
import numpy as np
import matplotlib.pyplot as plt
import meep as mp

mp.verbosity(0)

F_OP = 1.0                      # Betriebsfrequenz (lambda = 1)
GAMMA = 1e-2                    # Verluste der Bausteine
F0_RING, S_RING = 0.8, 1.125    # Ringresonanz und -stärke

# Drähte: Drude-epsilon (Kapitel 16). Meeps Konvention:
# eps(f) = 1 - sigma·f_ref²/(f² + i·gamma·f) — mit f_ref = F_OP und
# sigma = 2 wird eps(F_OP) = 1 - 2 = -1, exakt.
```

```
drude_e = mp.DrudeSusceptibility(frequency=F_OP, gamma=GAMMA, sigma=2.0)

# Ringe: Lorentz-mu (Kapitel-19-Schwingkreis als Materialantwort):
# mu(f) = 1 + sigma·f0²/(f0² − f² − i·gamma·f)
lorentz_h = mp.LorentzianSusceptibility(frequency=F0_RING, gamma=GAMMA,
                                        sigma=S_RING)

MAT_DRAEHTE = mp.Medium(epsilon=1, E_susceptibilities=[drude_e])
MAT_RINGE = mp.Medium(epsilon=1, mu=1, H_susceptibilities=[lorentz_h])
MAT_NIM = mp.Medium(epsilon=1, E_susceptibilities=[drude_e],
                    mu=1, H_susceptibilities=[lorentz_h])


def eps_von(f):
    """Drude-Antwort der Drähte (komplex)."""
    return 1.0 - 2.0 * F_OP**2 / (f**2 + 1j * GAMMA * f)


def mu_von(f):
    """Lorentz-Antwort der Ringe (komplex)."""
    return 1.0 + S_RING * F0_RING**2 / (F0_RING**2 - f**2 - 1j * GAMMA * f)


print(f"eps({F_OP}) = {eps_von(F_OP):.4f}")
print(f"mu({F_OP})  = {mu_von(F_OP):.4f}")
```

```
Using MPI version 4.1, 1 processes
eps(1.0) = -0.9998+0.0200j
mu(1.0)  = -0.9985+0.0555j
```

Beide Antworten stehen bei $f = 1$ auf -1 (plus kleine Imaginärteile — die Verluste). Bevor wir messen, lohnt der Blick auf die ganzen Kurven, denn sie legen die Bühne fest:

```
fs = np.linspace(0.3, 1.7, 600)
F_P = np.sqrt(2) * F_OP                          # eps-Nulldurchgang
F_MU_OBEN = F0_RING * np.sqrt(1 + S_RING)        # mu-Nulldurchgang

fig, (a1, a2) = plt.subplots(1, 2, figsize=(9.6, 3.6), sharex=True)
a1.plot(fs, eps_von(fs).real, "C0-", lw=1.6)
a1.axhline(0, color="k", lw=0.6, ls="--")
a1.axvline(F_P, color="C0", lw=0.8, ls=":")
```

```
a1.annotate(r"$f_p$", (F_P, 1.5), color="C0")
a1.set_ylim(-6, 3)
a1.set_xlabel("Frequenz f")
a1.set_ylabel(r"$\mathrm{Re}\,\varepsilon$")
a1.set_title("Drähte: Drude")
a2.plot(fs, mu_von(fs).real, "C3-", lw=1.6)
a2.axhline(0, color="k", lw=0.6, ls="--")
a2.axvspan(F0_RING, F_MU_OBEN, color="gold", alpha=0.25)
a2.annotate(r"$\mu < 0$", (0.93, 2.0), color="C3")
a2.set_ylim(-6, 6)
a2.set_xlabel("Frequenz f")
a2.set_ylabel(r"$\mathrm{Re}\,\mu$")
a2.set_title("Ringe: Lorentz (Resonanz bei 0,8)")
for a in (a1, a2):
    a.grid(alpha=0.3)
fig.tight_layout()
plt.show()
```

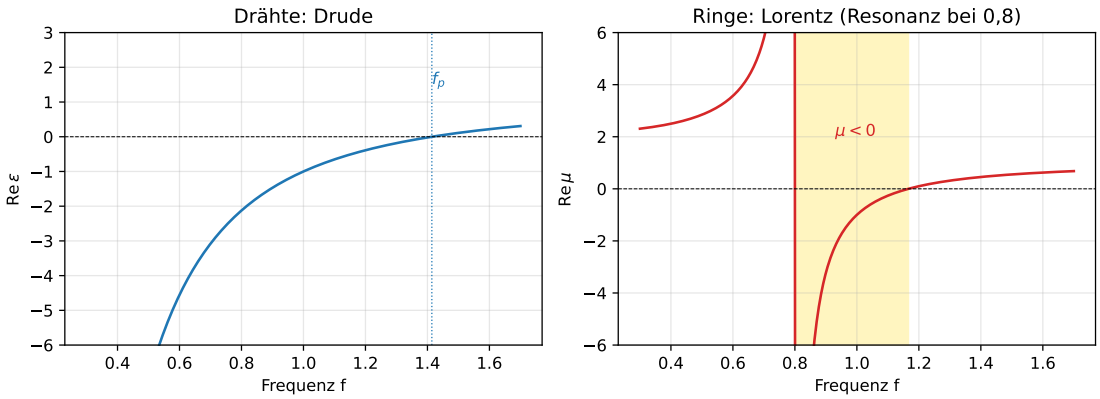


Abbildung 28.1: Die beiden Zutaten über der Frequenz. Links die Drude-Antwort der Drähte: ε < 0 für alles unterhalb von f_p = 1,41. Rechts die Lorentz-Antwort der Ringe: μ < 0 nur im Resonanzband 0,8 bis 1,17 (gold). Nur im Überlapp (beide Kurven unter der Nulllinie, gestrichelt) sind beide Vorzeichen negativ — dort, und nur dort, kann die Welle wieder laufen.

Lies die Karte: Die Drähte sperren (allein) das ganze Band bis $f_p = 1{,}41$. Die Ringe sperren (allein) ihr Resonanzband $0{,}8$ bis $1{,}17$. Das Ringband liegt **vollständig im Drahtband** — und genau dort sind beide Vorzeichen gleichzeitig negativ.

28.3 Das Smith-Experiment: zwei Sperren öffnen ein Fenster

Versuchsaufbau. *Frage:* Was lässt ein Slab aus Drahtgitter, Ringgitter oder beidem zusammen durch? *Bühne:* eine 1D-Zelle (`dimensions=1`, wie der λ/4-Stapel in Kapitel 26), in der Mitte ein Slab der Dicke $d = 2$ aus dem jeweiligen Material, beidseitig 3 Einheiten Luft und 2 Einheiten PML. *Anregung:* ein breitbandiger Gauß-Puls (Mitte $f = 1$, Breite $1{,}4$ — deckt $0{,}3$ bis $1{,}7$ ab). *Messgröße:* die Transmission $T(f)$ als Fluss-Verhältnis mit Referenzlauf, exakt das Rezept aus Kapitel 12. *Erfolgskriterium:* Die drei Kurven müssen auf der parameterfreien Transfer-Matrix-Linie liegen, die wir gleich danach aus Kapitel 26 übernehmen.

Und jetzt du — der Vorhersagepunkt dieses Kapitels:

> Vorhersage (PRIMM)
>
> Die Drähte allein sperren bei $f = 1$ praktisch alles ($\varepsilon = -1$, $\mu = +1$ → evaneszent, Kapitel 16). Die Ringe allein sperren bei $f = 1$ ebenfalls praktisch alles ($\varepsilon = +1$, $\mu = -1$ → dieselbe Mathematik, nur mit getauschten Rollen). Was misst du, wenn **beide Gitter zusammen** im Slab stehen — zwei Sperren hintereinander? (a) Noch dichter — Sperren addieren sich. (b) Genau so dicht. (c) Der Slab wird durchlässig. Entscheide dich, *bevor* du weiterliest — und notiere in einem Satz, warum.

```
D_SLAB = 2.0                # Slabdicke
AUFL_1D = 64
DPML_1D, PAD_1D = 2.0, 3.0
SZ_1D = 2 * (DPML_1D + PAD_1D) + D_SLAB


def sweep_1d(material, fcen=1.0, df=1.4, nfreq=281):
    """T(f) durch den Slab: Lauf mit Material / Leerlauf (Kap. 12)."""
    quelle = [mp.Source(mp.GaussianSource(fcen, fwidth=df), component=mp.Ex,
                        center=mp.Vector3(0, 0, -SZ_1D / 2 + DPML_1D + 1.0))]
    z_mess = SZ_1D / 2 - DPML_1D - 1.0
    leistungen = []
    for geometrie in ([], [mp.Block(size=mp.Vector3(mp.inf, mp.inf, D_SLAB),
                                    center=mp.Vector3(), material=material)]):
        sim = mp.Simulation(cell_size=mp.Vector3(0, 0, SZ_1D),
                            resolution=AUFL_1D, dimensions=1, Courant=0.25,
                            boundary_layers=[mp.PML(DPML_1D)],
```

```
                                geometry=geometrie, sources=quelle)
        tr = sim.add_flux(fcen, df, nfreq,
                          mp.FluxRegion(center=mp.Vector3(0, 0, z_mess)))
        sim.run(until_after_sources=mp.stop_when_fields_decayed(
            50, mp.Ex, mp.Vector3(0, 0, z_mess), 1e-9))
        leistungen.append(np.array(mp.get_fluxes(tr)))
    fs = np.array(mp.get_flux_freqs(tr))
    return fs, leistungen[1] / leistungen[0]


fs_1d, T_draehte = sweep_1d(MAT_DRAEHTE)
_, T_ringe = sweep_1d(MAT_RINGE)
_, T_beide = sweep_1d(MAT_NIM)

i_op = np.argmin(np.abs(fs_1d - F_OP))
print(f"T(f = 1): Drähte allein {T_draehte[i_op]:.5f}, "
      f"Ringe allein {T_ringe[i_op]:.5f}, beide {T_beide[i_op]:.4f}")
```

```
T(f = 1): Drähte allein 0.00000, Ringe allein 0.00000, beide 0.3829
```

Antwort (c): **Zwei Sperren hintereinander sind durchlässig.** Jede allein blockt auf 10^{-4}, beide zusammen lassen 38 % durch. Wer hier (a) getippt hat, hat in Alltagslogik gedacht — zwei Mauern sind dicker als eine. Aber die Mauern sperren aus *entgegengesetzten* Gründen ($\varepsilon < 0$ bzw. $\mu < 0$), und zusammen heben sich die Gründe auf: $k^2 = \omega^2 \varepsilon \mu > 0$, die Welle läuft. Genau mit dieser Messung — Fenster nur, wenn *beide* Gitter montiert sind — hat Smith 2000 den negativen Index nachgewiesen.

Jetzt die Theorielinie. Die Transfer-Matrix aus Kapitel 26 kann das alles parameterfrei, wir müssen ihr nur beibringen, mit komplexem n **und** komplexem Wellenwiderstand $Z = \sqrt{\mu/\varepsilon}$ zu rechnen (bisher war stillschweigend $\mu = 1$). Dabei fällt die Vorzeichenwahl, über die wir oben geredet haben — und sie folgt zwei physikalischen Geboten:

```
def schicht_T(fs, eps_funktion, mu_funktion, d):
    """Transmission einer Schicht — Kap.-26-Matrix mit komplexem n, Z.

    Die Wurzel-Zweige wählt die Physik:
      Im(n) >= 0  — die Welle klingt in Laufrichtung ab (Kausalität:
                    ein passives Medium verstärkt nicht);
      Re(Z) >= 0  — die Energie fließt vorwärts.
    Sind eps und mu beide negativ, landet Re(n) damit von selbst
```

```
    im Negativen: n = −1 ist keine Setzung, sondern Konsequenz.
    """
    Ts = []
    for f in fs:
        eps, mu = eps_funktion(f), mu_funktion(f)
        n = np.sqrt(eps * mu)
        if n.imag < 0:
            n = -n                        # Zweig mit Im(n) >= 0
        Z = np.sqrt(mu / eps)
        if Z.real < 0:
            Z = -Z                        # Zweig mit Re(Z) >= 0
        delta = 2 * np.pi * f * n * d     # Laufphase (e^{-i delta}, Kap. 26)
        M = np.array([[np.cos(delta), -1j * Z * np.sin(delta)],
                      [-1j * np.sin(delta) / Z, np.cos(delta)]])
        t = 2.0 / ((M[0, 0] + M[0, 1]) + (M[1, 0] + M[1, 1]))
        Ts.append(abs(t)**2)
    return np.array(Ts)


eins = lambda f: 1.0 + 0j
T_tmm_draehte = schicht_T(fs_1d, eps_von, eins, D_SLAB)
T_tmm_ringe = schicht_T(fs_1d, eins, mu_von, D_SLAB)
T_tmm_beide = schicht_T(fs_1d, eps_von, mu_von, D_SLAB)

maske = (fs_1d > 0.4) & (fs_1d < 1.6)
for name, T, Tt in [("Drähte", T_draehte, T_tmm_draehte),
                    ("Ringe", T_ringe, T_tmm_ringe),
                    ("beide", T_beide, T_tmm_beide)]:
    rms = np.sqrt(np.mean((T[maske] - Tt[maske])**2))
    print(f"rms(Meep − Matrix), {name}: {rms:.4f}")
```

```
rms(Meep − Matrix), Drähte: 0.0040
rms(Meep − Matrix), Ringe: 0.0008
rms(Meep − Matrix), beide: 0.0054
```

```
fig, ax = plt.subplots(figsize=(7.2, 4.2))
for T, Tt, farbe, name in [(T_draehte, T_tmm_draehte, "C0", "Drähte allein"),
                           (T_ringe, T_tmm_ringe, "C2", "Ringe allein"),
                           (T_beide, T_tmm_beide, "C3", "beide zusammen")]:
    ax.plot(fs_1d, T, ".", ms=3.5, color=farbe, label=f"Meep: {name}")
    ax.plot(fs_1d, Tt, "-", lw=1.1, color=farbe, alpha=0.55)
ax.axvspan(F0_RING, F_MU_OBEN, color="gold", alpha=0.2,
```

```
            label=r"$\mu < 0$ (Ring-Band)")
ax.axvline(F_P, color="C0", lw=0.8, ls=":")
ax.set_xlim(0.3, 1.7)
ax.set_ylim(-0.03, 1.05)
ax.set_xlabel("Frequenz f")
ax.set_ylabel("Transmission T")
ax.set_title("Zwei Sperren öffnen ein Fenster")
ax.legend(fontsize=8, loc="upper left")
ax.grid(alpha=0.3)
plt.show()
```

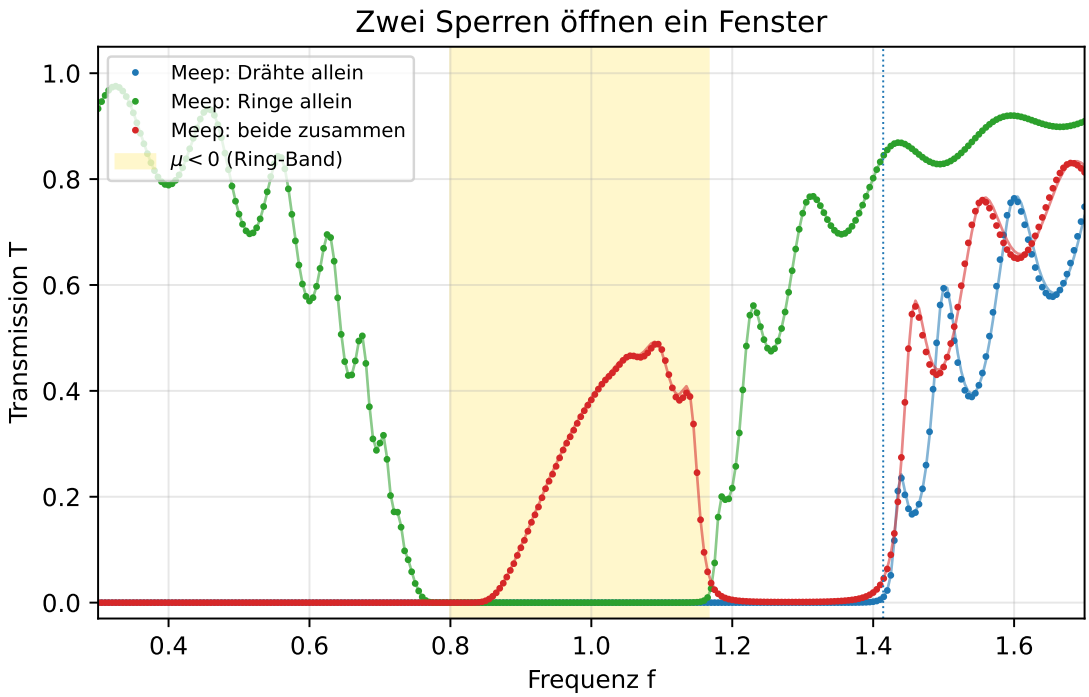


Abbildung 28.2: Das Smith-Experiment in 1D: Transmission durch den Slab für Drähte allein (blau), Ringe allein (grün) und beide zusammen (rot); Punkte sind Meep, Linien die Transfer-Matrix mit komplexem n und Z. Das goldene Band markiert μ < 0. Jedes Gitter allein sperrt dort — beide zusammen öffnen genau dort das Durchlassfenster des negativen Index.

Meep liegt auf der Matrix-Linie (rms unter 0,006 über den ganzen Sweep). Zwei Dinge verdient die Kurve noch:

Erstens, die weichen Flanken. Unterhalb von $f_0 = 0{,}8$ ist nur ε negativ

— verboten — und doch sickert bei $0{,}5$ sichtbar etwas durch. Das ist kein Fehler, sondern Kapitel 17: Der Slab ist nur 2λ dick, und evaneszente Wellen **tunneln** durch dünne verbotene Zonen. Verbotene Bänder haben bei endlicher Dicke immer weiche Ränder.

Zweitens, das Fenster ist nicht klar, sondern milchig: $T = 0{,}38$ statt $1{,}0$, obwohl die Anpassung bei $f = 1$ perfekt ist ($Z = \sqrt{\mu/\varepsilon} = 1$ — wie Vakuum, nichts wird reflektiert!). Der Fehlbetrag ist **reine Absorption**, und wir können das quantitativ festnageln: Aus dem komplexen n folgt die Lambert-Beer-Dämpfung (Kapitel 6) über die Slabdicke:

```
n_op = np.sqrt(eps_von(F_OP) * mu_von(F_OP))
if n_op.imag < 0:
    n_op = -n_op
# Leistung faellt wie e^(-2 Im(k) d) mit k = 2 pi f n
T_beer = np.exp(-2 * (2 * np.pi * F_OP * n_op.imag) * D_SLAB)
print(f"n(1) = {n_op:.4f}")
print(f"Lambert-Beer aus Im(n): T = {T_beer:.3f}  "
      f"(gemessen: {T_beide[i_op]:.3f})")
```

```
n(1) = -0.9993+0.0378j
Lambert-Beer aus Im(n): T = 0.387  (gemessen: 0.383)
```

$\mathrm{Im}(n) = 0{,}038$ erklärt das Fenster vollständig. Und dieses $\mathrm{Im}(n)$ ist kein Schönheitsfehler unserer Parameter, sondern Systemzwang: $\mu < 0$ gibt es nur **nahe der Ringresonanz** (Lorentz!), und an der Resonanz wohnen die Verluste (Kapitel 15). Reale Metamaterialien im Optischen verlieren pro Wellenlänge oft mehr, als unsere $\gamma = 0{,}01$ es tun — das ist bis heute ihr wundester Punkt.

28.4 Die Phase läuft rückwärts

Bis hierher könnte ein Skeptiker sagen: „Schönes Fenster — aber woher weißt du, dass n darin *negativ* ist und nicht einfach irgendein gewöhnlicher Wert?" Gute Frage. Der Fingerabdruck von $n < 0$ ist die **rückwärts laufende Phase**, und die können wir direkt messen.

Versuchsaufbau. *Frage:* In welche Richtung wächst die Phase der Welle vor, im und hinter dem Slab? *Bühne:* die 1D-Zelle von oben, etwas verlängert, Slab $d = 2$ aus dem Doppel-Material in der Mitte. *Anregung:* Dauerstrich bei $f = 1$ mit `force_complex_fields=True` — dann trägt jedes $E_x(z)$ seine Phase mit sich (Kapitel 15). *Messgröße:* das Phasenprofil $\varphi(z) = \arg E_x(z)$,

mit `np.unwrap` zu einer stetigen Kurve zusammengesetzt (das Werkzeug aus Kapitel 27, das dort die Umlaufmode gezählt hat); seine Steigung ist die lokale Wellenzahl. *Erfolgskriterium:* Steigung $+k_0$ in der Luft — und wenn $n = -1$ stimmt, exakt $-k_0$ im Slab.

```
SZ_CW = 2 * (DPML_1D + 5.5) + D_SLAB
sim = mp.Simulation(cell_size=mp.Vector3(0, 0, SZ_CW), resolution=AUFL_1D,
                    dimensions=1, Courant=0.25, force_complex_fields=True,
                    boundary_layers=[mp.PML(DPML_1D)],
                    geometry=[mp.Block(size=mp.Vector3(mp.inf, mp.inf, D_SLAB),
                                       center=mp.Vector3(), material=MAT_NIM)],
                    sources=[mp.Source(mp.ContinuousSource(F_OP),
                                       component=mp.Ex,
                                       center=mp.Vector3(0, 0, -SZ_CW / 2
                                                         + DPML_1D + 1.0))])
sim.run(until=400)

ex_cw = sim.get_array(component=mp.Ex, center=mp.Vector3(),
                      size=mp.Vector3(0, 0, SZ_CW - 2 * DPML_1D))
zs_cw = np.linspace(-(SZ_CW / 2 - DPML_1D), SZ_CW / 2 - DPML_1D, len(ex_cw))
phase_cw = np.unwrap(np.angle(ex_cw))
k0 = 2 * np.pi * F_OP

steigungen = {}
for name, z1, z2 in [("davor", -4.5, -1.5), ("im Slab", -0.8, 0.8),
                     ("dahinter", 1.5, 4.5)]:
    m = (zs_cw > z1) & (zs_cw < z2)
    steigungen[name] = np.polyfit(zs_cw[m], phase_cw[m], 1)[0] / k0
    print(f"Phasensteigung {name:9s}: {steigungen[name]:+.4f} · k0")
```

```
Phasensteigung davor    : +1.0015 · k0
Phasensteigung im Slab  : -1.0007 · k0
Phasensteigung dahinter : +1.0016 · k0
```

```
fig, ax = plt.subplots(figsize=(7.2, 4.0))
ax.axvspan(-D_SLAB / 2, D_SLAB / 2, color="0.85")
ax.plot(zs_cw, phase_cw / k0, "C0-", lw=1.6, label="gemessen")
for z_start, ph_start, stg in [(-4.5, None, 1), (1.5, None, 1)]:
    m0 = np.argmin(np.abs(zs_cw - z_start))
    zz = np.linspace(z_start, z_start + 3, 10)
    ax.plot(zz, (phase_cw[m0] + (zz - z_start) * k0) / k0, "k--", lw=0.9,
            alpha=0.6)
ax.plot([], [], "k--", lw=0.9, alpha=0.6, label="Steigung +k0 (Luft)")
```

```
ax.set_xlabel("z (grau: der Slab)")
ax.set_ylabel(r"Phase $\varphi(z)\,/\,k_0$")
ax.set_title(f"Steigung im Slab: {steigungen['im Slab']:+.3f} · k0")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

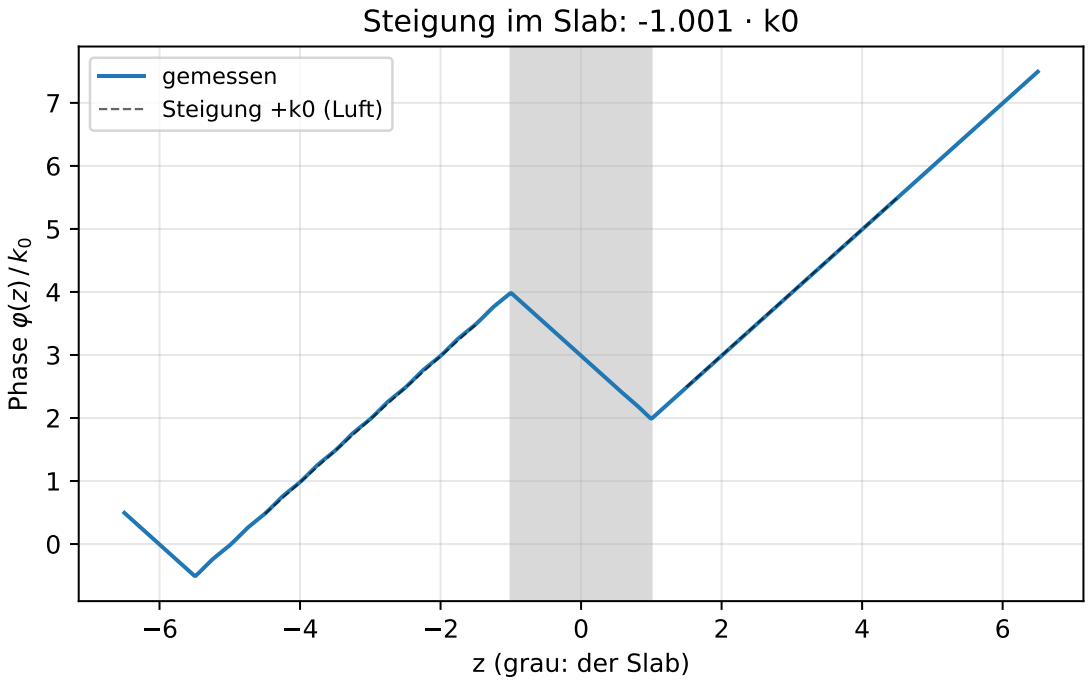


Abbildung 28.3: Das Phasenprofil der Dauerstrich-Welle (f = 1) über den Ort. In der Luft steigt die Phase mit $+k_0$ (gestrichelte Referenzgeraden) — im Slab (grau) fällt sie mit exakt $-k_0$. Die Welle läuft als Ganzes nach rechts, aber ihre Phase läuft im Material nach links: n = −1, wörtlich gemessen. (Der kleine Knick ganz links liegt vor der Quelle bei z = −5,5: Dort läuft die Welle nach links, ihre Phase fällt also ebenfalls.)

Da steht es: $+1{,}00\,k_0$ davor, $-1{,}00\,k_0$ drin, $+1{,}00\,k_0$ dahinter. Im Slab sind die Wellenberge rückwärts unterwegs.

Und die Energie? Die läuft vorwärts — sonst käme hinten nichts an. Wie schnell, sagt uns ein Puls. Seine Gruppengeschwindigkeit können wir sogar von Hand vorhersagen, mit der Gruppenindex-Formel aus Kapitel 16, $n_g = n + f\,\mathrm{d}n/\mathrm{d}f$. Die Ableitung entzerren wir wie üblich Schritt für Schritt: Aus $n^2 = \varepsilon\mu$ folgt mit der Produktregel

$$2n\,\frac{\mathrm{d}n}{\mathrm{d}f} = \varepsilon\,\frac{\mathrm{d}\mu}{\mathrm{d}f} + \mu\,\frac{\mathrm{d}\varepsilon}{\mathrm{d}f}.$$

Bei $f = 1$ ist $\varepsilon = \mu = n = -1$, und die beiden Steigungen liest man aus den Modellformeln ab: $\mathrm{d}\varepsilon/\mathrm{d}f = 4$ (Drude) und $\mathrm{d}\mu/\mathrm{d}f = 11{,}1$ (Lorentz). Also

$$\frac{\mathrm{d}n}{\mathrm{d}f} = \frac{(-1)(11{,}1) + (-1)(4)}{2\cdot(-1)} = +7{,}56, \qquad n_g = -1 + 7{,}56 = 6{,}56.$$

Zwei Dinge stehen in dieser kleinen Rechnung: Erstens ist n_g **positiv** — die Energie läuft vorwärts, obwohl n negativ ist. Zweitens ist n_g **groß**: Der Puls kriecht mit $c/6{,}56 \approx 0{,}15\,c$. Beides prüft die Messung — wir schicken einen schmalbandigen Puls durch einen dickeren Slab ($d = 3$) und stoppen die Zeit zwischen zwei Sonden:

```
D_DICK = 3.0
sz_p = 2 * (DPML_1D + 5.0) + D_DICK
sim = mp.Simulation(cell_size=mp.Vector3(0, 0, sz_p), resolution=AUFL_1D,
                    dimensions=1, Courant=0.25,
                    boundary_layers=[mp.PML(DPML_1D)],
                    geometry=[mp.Block(size=mp.Vector3(mp.inf, mp.inf, D_DICK),
                                       center=mp.Vector3(), material=MAT_NIM)],
                    sources=[mp.Source(mp.GaussianSource(F_OP, fwidth=0.08),
                                       component=mp.Ex,
                                       center=mp.Vector3(0, 0, -sz_p / 2
                                                         + DPML_1D + 0.5))])
vorn, hinten, zeiten = [], [], []


def messe_laufzeit(s):
    zeiten.append(s.meep_time())
    vorn.append(abs(s.get_field_point(
        mp.Ex, mp.Vector3(0, 0, -D_DICK / 2 - 1.0))))
    hinten.append(abs(s.get_field_point(
        mp.Ex, mp.Vector3(0, 0, D_DICK / 2 + 1.0))))


sim.run(mp.at_every(0.5, messe_laufzeit), until=120)
zeiten = np.array(zeiten)
dt_puls = zeiten[np.argmax(hinten)] - zeiten[np.argmax(vorn)]
v_g = D_DICK / (dt_puls - 2.0)     # 2,0 = Vakuumweg zwischen den Sonden
```

```
print(f"Laufzeit Sonde→Sonde: {dt_puls:.1f}; "
      f"v_g im Slab = {v_g:.3f} c (Vorhersage 1/6,56 = {1 / 6.56:.3f} c)")
```

```
Laufzeit Sonde→Sonde: 21.5; v_g im Slab = 0.154 c (Vorhersage 1/6,56 = 0.152 c)
```

$v_g = 0{,}154\,c$ gemessen, $0{,}152\,c$ vorhergesagt. **Energie vorwärts (langsam), Phase rückwärts** — beides gleichzeitig, beides quantitativ.

> Naheliegende Vermutung: „Phase rückwärts — das ist doch Zeitumkehr und verletzt die Kausalität“
>
> **Warum sie naheliegt:** Rückwärtslaufende Wellenberge sehen aus wie ein rückwärts abgespielter Film — und der verstößt gegen den Zeitpfeil aus Kapitel 20.
> **Was stattdessen stimmt:** Transportiert wird nur, was Energie und Information trägt — und beides reist mit der *Gruppe*, vorwärts und (hier) gemächlich mit $0{,}15\,c$. Die Phasenberge sind keine Boten; das haben wir schon in Kapitel 16 gemessen, als $v_p = 1{,}67\,c$ über der Plasmafrequenz auch niemandem Überlicht beschert hat. Ein negatives v_p ist genauso harmlos wie ein überlichtschnelles. Der Film läuft vorwärts — nur das Riffelmuster darauf bewegt sich gegen den Strom, wie das Profil einer Bohrschnecke, die sich dreht: Das Muster wandert nach oben, das Bohrmehl nach unten.

28.5 Negative Brechung: der Knick auf die falsche Seite

Jetzt in die Fläche. Snellius (Kapitel 12) mit $n = -1$ sagt: $\sin\theta_2 = \sin\theta_1/n = -\sin\theta_1$ — der gebrochene Strahl liegt auf der **gleichen Seite der Normalen** wie der einfallende. Das klingt harmlos und sieht verstörend aus. Wir messen es.

Versuchsaufbau. *Frage:* Auf welche Seite knickt ein schräger Strahl in einer Platte aus dem Doppel-Material — und wie groß ist der Versatz nach dem Durchgang? *Bühne:* eine 2D-Zelle (20×12, PML ringsum), darin horizontal eine Platte der Dicke $t = 3$; einmal aus gewöhnlichem Glas ($n = 1{,}5$) als Kontrolle, einmal aus dem Material des Smith-Fensters. *Anregung:* ein Gauß-Bündel (`GaussianBeamSource`, Taille $1{,}5$, Dauerstrich $f = 1$), das unter $30°$ von oben einfällt. *Messgröße:* der Schwerpunkt des Strahls direkt über dem Eintritt und direkt unter dem Austritt; die Differenz ist der seitliche Versatz über die Platte. *Erwartung:* Glas bricht zur Normalen hin, $\theta_2 =$

$19{,}5°$, Versatz $+t\tan 19{,}5° = +1{,}06$; bei $n = -1$ gilt $\theta_2 = -30°$, Versatz $-t\tan 30° = -1{,}73$ — anderes Vorzeichen, und betragsmäßig sogar größer als der Einfallswinkel-Versatz von Glas.

Ein Bau-Detail ist diesmal Pflichtlektüre — es hat diese Simulation drei Anläufe gekostet:

Werkstattkasten: das μ-dispersive Material darf die PML nicht berühren

Der erste Versuch dieser Simulation — Platte quer durch die ganze Zelle, wie wir es seit Kapitel 12 immer bauen — starb nach wenigen hundert Schritten mit Meeps Fehlermeldung *„fields are NaN or Inf"*. Die Detektivarbeit (Bausteine einzeln testen: nur-ε stabil, nur-μ explodiert, homogene Zelle mit periodischen Rändern stabil) zeigte: Schuld war der **Überlapp von Material und PML**. Meeps FAQ nennt den Fall ausdrücklich — in Medien, die Rückwärtswellen tragen, dreht die PML-Streckung ihre Dämpfung in **Verstärkung** um; auch der einfachere `mp.Absorber` hilft hier nicht. In 1D war der Slab nie in der PML, darum fiel es erst jetzt auf. Die Regel für alle NIM-Läufe in diesem Kapitel: **Blöcke endlich bauen, mit Luftpuffer zur PML** — was am Plattenrand an Beugung entsteht, ist der ehrliche Preis (echte Proben sind auch endlich).

```
AUFL_2D = 32
MAT_GLAS = mp.Medium(index=1.5)


def brechung_slab(material, until=150):
    """Gauß-Bündel, 30° Einfall, Platte t = 3 — komplexes CW-Feld."""
    t, dpml = 3.0, 1.0
    sx, sy = 20.0, 12.0
    theta = np.radians(30)
    quelle = [mp.GaussianBeamSource(
        mp.ContinuousSource(F_OP),
        center=mp.Vector3(-2.5, sy / 2 - dpml - 0.3),
        size=mp.Vector3(8.0, 0), beam_x0=mp.Vector3(0, 0),
        beam_kdir=mp.Vector3(np.sin(theta), -np.cos(theta)),
        beam_w0=1.5, beam_E0=mp.Vector3(0, 0, 1))]
    block = mp.Block(size=mp.Vector3(sx - 2 * dpml - 2.0, t, mp.inf),
                     center=mp.Vector3(), material=material)
    sim = mp.Simulation(cell_size=mp.Vector3(sx, sy), resolution=AUFL_2D,
                        Courant=0.25, force_complex_fields=True,
                        boundary_layers=[mp.PML(dpml)],
                        geometry=[block], sources=quelle)
```

```
    sim.run(until=until)
    ez = sim.get_array(component=mp.Ez, center=mp.Vector3(),
                       size=mp.Vector3(sx - 2 * dpml, sy - 2 * dpml))
    return ez, sx - 2 * dpml, sy - 2 * dpml


def strahl_versatz(feld, bx, by, t=3.0):
    """Strahlschwerpunkt unter dem Austritt minus über dem Eintritt."""
    nx, ny = feld.shape
    xs = np.linspace(-bx / 2, bx / 2, nx)
    ys = np.linspace(-by / 2, by / 2, ny)
    intensitaet = np.abs(feld)**2

    def schwerpunkt(y0):
        iy = np.argmin(np.abs(ys - y0))
        zeile = intensitaet[:, iy]
        return np.sum(xs * zeile) / np.sum(zeile)

    return schwerpunkt(-t / 2 - 0.2) - schwerpunkt(+t / 2 + 0.2)


ez_glas, bx_b, by_b = brechung_slab(MAT_GLAS)
ez_nim, _, _ = brechung_slab(MAT_NIM)
v_glas = strahl_versatz(ez_glas, bx_b, by_b)
v_nim = strahl_versatz(ez_nim, bx_b, by_b)
print(f"Versatz Glas: {v_glas:+.3f}  (Theorie +1,06)")
print(f"Versatz NIM:  {v_nim:+.3f}  (Theorie −1,73)")
```

```
Versatz Glas: +1.092  (Theorie +1,06)
Versatz NIM:  -1.558  (Theorie −1,73)
```

```
fig, achsen = plt.subplots(1, 2, figsize=(10.5, 4.4), sharey=True)
for ax, feld, name, versatz in [(achsen[0], ez_glas, "Glas n = +1,5", v_glas),
                                (achsen[1], ez_nim, "Metamaterial n = −1",
                                 v_nim)]:
    ax.imshow(np.abs(feld).T, origin="lower", cmap="inferno",
              extent=[-bx_b / 2, bx_b / 2, -by_b / 2, by_b / 2],
              vmax=0.85 * np.abs(ez_glas).max())
    ax.axhline(-1.5, color="w", lw=0.7)
    ax.axhline(1.5, color="w", lw=0.7)
    ax.annotate("", xy=(-1.0, 1.5), xytext=(-2.5, 4.2),
                arrowprops=dict(color="w", arrowstyle="->", lw=1.2))
```

```
    ax.set_title(f"{name}:  Versatz {versatz:+.2f}")
    ax.set_xlabel("x")
achsen[0].set_ylabel("y")
fig.tight_layout()
plt.show()
```

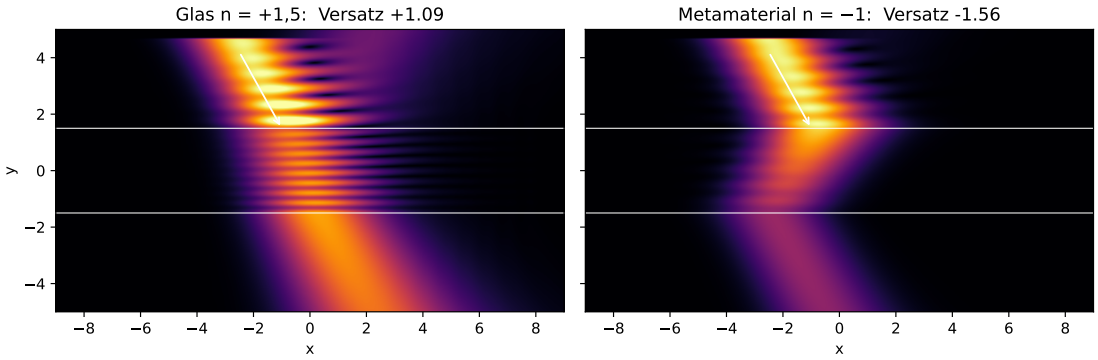


Abbildung 28.4: Dasselbe Bündel, zwei Platten: links Glas (n = 1,5), rechts das Metamaterial (n = −1). Gezeigt ist |Ez| (Momentaufnahme des Dauerstrichs), die weißen Linien markieren die Platte, der Pfeil den einfallenden Strahl. Im Glas knickt der Strahl zur Normalen, im Metamaterial auf die andere Seite — der Austrittsversatz wechselt das Vorzeichen (gemessen +1,09 bzw. −1,56; die Theoriewerte +1,06 und −1,73 gelten für den unendlich breiten Strahl).

Im Glasbild läuft der Strahl wie seit Kapitel 12 gewohnt: hinein, zur Normalen geknickt, hinaus, Versatz nach rechts. Im Metamaterial-Bild knickt derselbe Strahl am Eintritt **nach links zurück** — er kreuzt die eigene Einfallsrichtung — und tritt um $-1{,}56$ versetzt aus (der Theoriewert $-1{,}73$ gilt für unendlich breite Strahlen; unser Bündel ist nur $1{,}5\lambda$ schmal und verliert etwas an die Beugung an den Plattenkanten). Ein Vorzeichen als Messwert — genau wie es 2001 im Mikrowellenlabor gemessen wurde, am Prisma statt an der Platte.

28.6 Die Veselago-Flachlinse

Veselagos hübscheste Folgerung kommt ganz ohne Krümmung aus. Eine Sammellinse funktioniert, weil gekrümmtes Glas schräge Strahlen zur Achse zurückbiegt. Bei $n = -1$ biegt schon die **ebene** Grenzfläche jeden Strahl

auf die andere Seite der Normalen — eine flache Platte sammelt von selbst: Strahlen, die von einem Punkt vor der Platte auseinanderlaufen, werden am Eintritt zurückgefaltet, kreuzen sich **in** der Platte und nach dem Austritt **noch einmal**. Eine Platte der Dicke t bildet einen Punkt im Abstand d_1 davor in einen Punkt im Abstand $t - d_1$ dahinter ab. Keine Brennweite, keine Achse, kein Schleifen — jede Punktquelle vor der Platte bekommt ihr Bild dahinter.

> Vorhersage (PRIMM)
>
> Unsere Platte ist $t = 2$ dick, die Punktquelle sitzt $d_1 = 1$ vor ihrer Vorderseite. Die Vorderseite liegt bei $y = -1$, die Rückseite bei $y = +1$. Bei welcher y-Koordinate erwartest du das Bild? Und: Was erwartest du für eine Quelle, die *weiter* als t vor der Platte sitzt ($d_1 > t$)?

Versuchsaufbau. *Frage:* Wo bündelt die flache Platte das Licht der Punktquelle? *Bühne:* 2D-Zelle 14×10, Platte $t = 2$ (endlich breit, Luftpuffer zur PML — der Werkstattkasten gilt weiter). *Anregung:* Punktquelle (Dauerstrich $f = 1$) bei $y = -2$, also $d_1 = 1$ vor der Platte. *Messgrößen:* die Karte $|E_z|$, das Intensitätsprofil auf der Achse hinter der Platte (Lage des Maximums) und die Halbwertsbreite des Fokus quer zur Achse. *Erfolgskriterium:* Bild bei $y = +1 + (t - d_1) = +2$, Fokusbreite nahe der Beugungsgrenze $\sim \lambda/2$.

```
T_LINSE, D1 = 2.0, 1.0


def flachlinse(material=None, until=200):
    dpml = 1.0
    sx, sy = 14.0, 10.0
    block = mp.Block(size=mp.Vector3(sx - 2 * dpml - 2.0, T_LINSE, mp.inf),
                     center=mp.Vector3(),
                     material=material if material else MAT_NIM)
    sim = mp.Simulation(cell_size=mp.Vector3(sx, sy), resolution=AUFL_2D,
                        Courant=0.25, force_complex_fields=True,
                        boundary_layers=[mp.PML(dpml)],
                        geometry=[block],
                        sources=[mp.Source(mp.ContinuousSource(F_OP),
                                           component=mp.Ez,
                                           center=mp.Vector3(0, -T_LINSE / 2
                                                             - D1))])
    sim.run(until=until)
    ez = sim.get_array(component=mp.Ez, center=mp.Vector3(),
                       size=mp.Vector3(sx - 2 * dpml, sy - 2 * dpml))
```

```
    return ez, sx - 2 * dpml, sy - 2 * dpml


ez_linse, bx_l, by_l = flachlinse()
nx_l, ny_l = ez_linse.shape
xs_l = np.linspace(-bx_l / 2, bx_l / 2, nx_l)
ys_l = np.linspace(-by_l / 2, by_l / 2, ny_l)
int_linse = np.abs(ez_linse)**2

# Fokus: Intensitätsmaximum auf der Achse, oberhalb der Platte
ix0 = np.argmin(np.abs(xs_l))
profil_achse = np.where(ys_l > 1.2, int_linse[ix0, :], 0)
iy_fokus = np.argmax(profil_achse)
y_fokus = ys_l[iy_fokus]

# Quer-Halbwertsbreite am Fokus (Randschutz wie in Kap. 27 gelernt)
quer = int_linse[:, iy_fokus] / int_linse[:, iy_fokus].max()
i_max = int(np.argmax(quer))
lo, hi = i_max, i_max
while lo > 0 and quer[lo - 1] >= 0.5:
    lo -= 1
while hi < len(quer) - 1 and quer[hi + 1] >= 0.5:
    hi += 1
fwhm_fokus = xs_l[hi] - xs_l[lo]
print(f"Fokus auf der Achse bei y = {y_fokus:+.3f}  (Theorie +2,0)")
print(f"Quer-FWHM des Fokus: {fwhm_fokus:.3f}  (lambda/2 = 0,5)")
```

```
Fokus auf der Achse bei y = +1.914  (Theorie +2,0)
Quer-FWHM des Fokus: 0.405  (lambda/2 = 0,5)
```

```
fig, ax = plt.subplots(figsize=(7.0, 5.0))
ax.imshow(np.abs(ez_linse).T, origin="lower", cmap="inferno",
          extent=[-bx_l / 2, bx_l / 2, -by_l / 2, by_l / 2],
          vmax=0.6 * np.abs(ez_linse).max())
ax.axhline(-T_LINSE / 2, color="w", lw=0.7)
ax.axhline(T_LINSE / 2, color="w", lw=0.7)
ax.plot(0, -T_LINSE / 2 - D1, "wo", ms=7, mfc="none", mew=1.4)
ax.plot(0, y_fokus, "w+", ms=12, mew=1.6)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_title("Eine flache Platte als Linse")
plt.show()
```

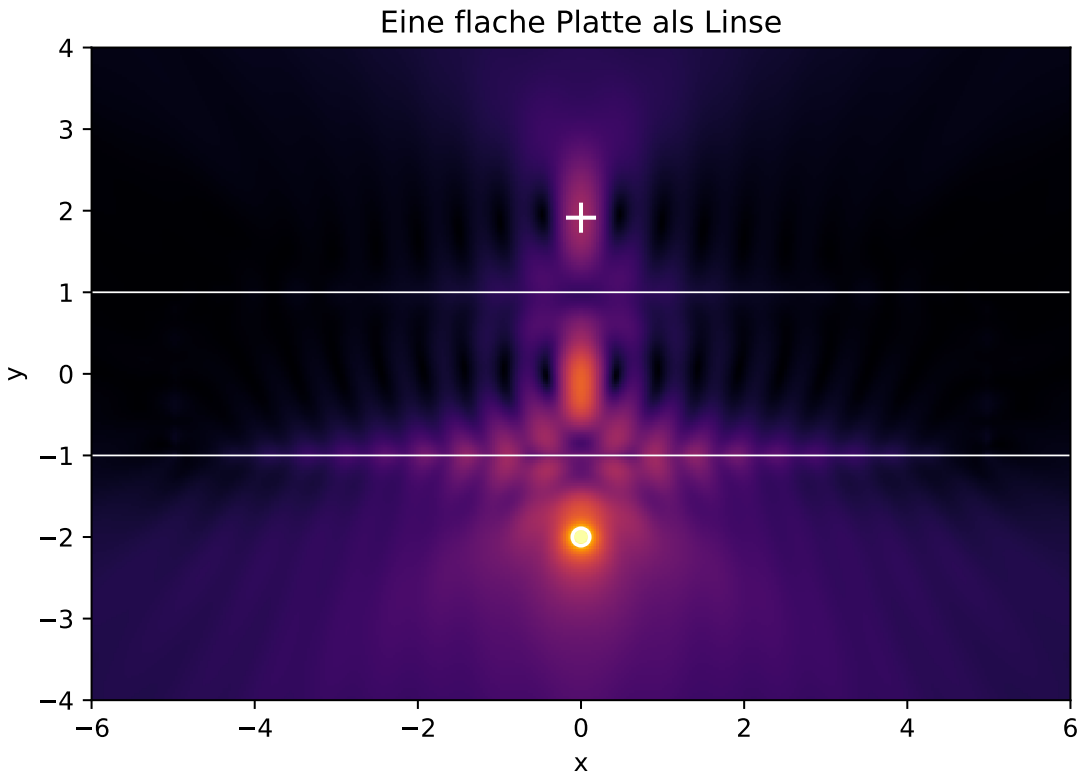


Abbildung 28.5: Die Flachlinse bei der Arbeit: |Ez|-Karte der Punktquelle (weißer Kreis bei y = −2) vor der Metamaterial-Platte (weiße Linien, t = 2). Das Licht wird in der Platte einmal und dahinter ein zweites Mal gebündelt; das weiße Kreuz markiert das gemessene Bild bei y = +1,91 ≈ t − d_1 hinter der Austrittsfläche — ohne jede Krümmung.

Das Bild sitzt bei $y = +1{,}91$ — neun Hundertstel vor dem Ideal, der Tribut an die Verluste — und sein Fokus ist mit FWHM $= 0{,}41$ etwa beugungsbegrenzt scharf. In der Platte selbst siehst du das **innere** Bild: die erste Kreuzung der zurückgefalteten Strahlen, bei Tiefe d_1 unter der Eintrittsfläche.

Eine Randnotiz mit großer Geschichte: Im Jahr 2000 zeigte John Pendry, dass die ideale, verlustfreie Veselago-Platte sogar die **evaneszenten** Felder der Quelle (Kapitel 12) wieder aufrichten würde — sie wäre eine „perfekte Linse" mit Auflösung *unterhalb* der Beugungsgrenze. Unser gemessenes FWHM von $0{,}41 < \lambda/2$ deutet diesen Effekt bereits an. Warum daraus trotzdem kein Wunder-Mikroskop wurde, steht im Kleingedruckten — die Kurzfassung hast du in diesem Kapitel schon zweimal gelesen: Verluste wohnen an der Resonanz. Was aus den Verlusten folgt, misst du selbst in Übung 28.4.

28.7 Doppelbrechung: ein Material, zwei Indizes

Bisher haben wir an den *Vorzeichen* von ε gedreht. Die zweite Hälfte des Kapitels dreht an etwas anderem: an der **Richtung**. Alle unsere Materialien waren bisher isotrop — derselbe ε-Wert, egal wie das Feld orientiert ist. Aber viele Kristalle sind anisotrop: Ihre Atome sitzen in Reihen und Schichten, und ein $\vec{E}$-Feld längs der Schichten polarisiert sie leichter als eines quer dazu. Dann ist ε keine Zahl mehr, sondern ein **Tensor** — für uns ganz praktisch: drei verschiedene Werte für die drei Feldrichtungen,

$$\varepsilon = \begin{pmatrix} \varepsilon_x & 0 & 0 \\ 0 & \varepsilon_y & 0 \\ 0 & 0 & \varepsilon_z \end{pmatrix},$$

solange wir die Koordinatenachsen auf die Kristallachsen legen. Die Konsequenz steht sofort da: Eine Welle, deren $\vec{E}$ in z zeigt, spürt ε_z und läuft mit $n_z = \sqrt{\varepsilon_z}$; eine mit $\vec{E}$ in x spürt ε_x. **Ein Material, zwei Brechungsindizes — sortiert nach Polarisation.** Das heißt Doppelbrechung, und unser Modellkristall ist der Klassiker: Kalkspat (Calcit), mit $n_o = 1{,}658$ („ordentlich", für $\vec{E}$ senkrecht zur besonderen Kristallachse) und $n_e = 1{,}486$ („außerordentlich", für $\vec{E}$ parallel dazu) — eine der größten Doppelbrechungen unter den häufigen Mineralen.

Unsere 2D-Welt ist dafür wie geschaffen, denn sie hat von Natur aus zwei getrennte Polarisationen (Kapitel 13): die E_z-Welle (das Feld zeigt aus der Ebene) und die H_z-Welle (das Feld liegt in der Ebene). Legen wir die besondere Kristallachse in z-Richtung, dann gilt $\varepsilon_x = \varepsilon_y = n_o^2$ und $\varepsilon_z = n_e^2$ — die E_z-Welle ist der außerordentliche Strahl, die H_z-Welle der ordentliche.

Versuchsaufbau. *Frage:* Brechen sich die beiden Polarisationen am selben Kristall verschieden? *Bühne:* 2D-Zelle 22×14, ein Kalkspat-Block der Dicke $t = 6$ quer durch die Zelle (`epsilon_diag` — ein gewöhnliches, verlustfreies Material, das auch in der PML brav bleibt). *Anregung:* dasselbe Gauß-Bündel wie oben, Einfall $45°$, einmal als E_z-, einmal als H_z-Welle. *Messgröße:* diesmal nicht der Strahlschwerpunkt, sondern **der Wellenvektor selbst** — wir schneiden die komplexe Feldkarte im Kristall aus und holen uns per 2D-FFT den dominanten $\vec{k}$: seine Richtung ist der Brechungswinkel, sein Betrag (geteilt durch f) der Index. Eine Messung, zwei Ergebnisse. *Erwartung (Snellius):* $\theta_e = \arcsin(\sin 45°/1{,}486) = 28{,}4°$ und $\theta_o = \arcsin(\sin 45°/1{,}658) = 25{,}2°$.

Warum nicht wieder der Schwerpunkt? Werkstatt-Erfahrung aus dem Aufbau dieses Kapitels: Beim $45°$-Einfall ist die interne Reflexion der E_z-Welle

(s-Polarisation, Kapitel 13) kräftig genug, dass ihr Interferenzmuster den Schwerpunkt verfälscht — die H_z-Welle läuft dagegen nahe am Brewster-Winkel und ist sauber. Die FFT trennt die Laufrichtungen von selbst und ist gegen solche Überlagerungen immun; das Hanning-Fenster und das Zero-Padding kennen wir sinngemäß aus den Spektren von Kapitel 10, nur eben in zwei Dimensionen.

```
N_O, N_E = 1.658, 1.486
MAT_KALKSPAT = mp.Medium(epsilon_diag=mp.Vector3(N_O**2, N_O**2, N_E**2))
TH_EIN = np.radians(45)
AUFL_DB = 48                        # feiner: wir messen |k| auf Prozente


def brechung_aniso(polarisation, until=150):
    """45°-Bündel auf den Kalkspat-Block; liefert die Feldkarte IM Kristall."""
    t, dpml = 6.0, 1.0
    sx, sy = 22.0, 14.0
    e0 = (mp.Vector3(0, 0, 1) if polarisation == "Ez"
          else mp.Vector3(np.cos(TH_EIN), np.sin(TH_EIN), 0))
    quelle = [mp.GaussianBeamSource(
        mp.ContinuousSource(F_OP),
        center=mp.Vector3(-5.0, sy / 2 - dpml - 0.3),
        size=mp.Vector3(8.0, 0), beam_x0=mp.Vector3(0, 0),
        beam_kdir=mp.Vector3(np.sin(TH_EIN), -np.cos(TH_EIN)),
        beam_w0=1.5, beam_E0=e0)]
    sim = mp.Simulation(cell_size=mp.Vector3(sx, sy), resolution=AUFL_DB,
                        Courant=0.25, force_complex_fields=True,
                        boundary_layers=[mp.PML(dpml)],
                        geometry=[mp.Block(size=mp.Vector3(mp.inf, t, mp.inf),
                                           center=mp.Vector3(),
                                           material=MAT_KALKSPAT)],
                        sources=quelle)
    sim.run(until=until)
    komponente = mp.Ez if polarisation == "Ez" else mp.Hz
    return sim.get_array(component=komponente, center=mp.Vector3(0, 0),
                         size=mp.Vector3(16.0, 5.0))


def k_dominant(feld, bx=16.0, by=5.0):
    """Dominanter k-Vektor einer komplexen Feldkarte per 2D-FFT.

    Zutaten: Hanning-Fenster gegen die harten Ausschnittsränder
    (Kap. 10), Zero-Padding ×4 für ein feines k-Raster, danach
    Schwerpunkt der 3×3-Umgebung des Spektral-Maximums.
```

```
    """
    nx, ny = feld.shape
    # np.hanning: das Glockenfenster aus der Kap.-10-Werkstatt, hier in
    # beiden Richtungen als äußeres Produkt über die Karte gelegt
    gefenstert = feld * np.hanning(nx)[:, None] * np.hanning(ny)[None, :]
    # np.fft.fft2 = DFT in beiden Achsen zugleich (Karte -> k-Raum);
    # np.fft.fftshift rückt die Nullfrequenz vom Rand in die Mitte,
    # damit kx/ky wie gewohnt von negativ nach positiv laufen
    spektrum = np.fft.fftshift(np.fft.fft2(gefenstert, s=(4 * nx, 4 * ny)))
    kx = np.fft.fftshift(np.fft.fftfreq(4 * nx, d=bx / nx))
    ky = np.fft.fftshift(np.fft.fftfreq(4 * ny, d=by / ny))
    # argmax liefert den Index im flachgeklopften Array –
    # np.unravel_index übersetzt ihn zurück in das (i, j)-Paar
    i, j = np.unravel_index(np.argmax(np.abs(spektrum)), spektrum.shape)
    umgebung = np.abs(spektrum[i - 1:i + 2, j - 1:j + 2])
    di = (umgebung * np.array([-1, 0, 1])[:, None]).sum() / umgebung.sum()
    dj = (umgebung * np.array([-1, 0, 1])[None, :]).sum() / umgebung.sum()
    return (kx[i] + di * (kx[1] - kx[0]), ky[j] + dj * (ky[1] - ky[0]))


felder_db = {}
for pol, n_soll in [("Ez", N_E), ("Hz", N_O)]:
    feld = brechung_aniso(pol)
    felder_db[pol] = feld
    kxp, kyp = k_dominant(feld)
    winkel = np.degrees(np.arctan2(abs(kxp), abs(kyp)))
    n_mess = np.hypot(kxp, kyp) / F_OP
    soll = np.degrees(np.arcsin(np.sin(TH_EIN) / n_soll))
    print(f"{pol}-Welle: Winkel {winkel:5.2f}° (Snellius {soll:5.2f}°),  "
          f"n = {n_mess:.3f} (Soll {n_soll})")
```

```
Ez-Welle: Winkel 28.39° (Snellius 28.41°),  n = 1.479 (Soll 1.486)
Hz-Welle: Winkel 25.59° (Snellius 25.24°),  n = 1.664 (Soll 1.658)
```

```
fig, achsen = plt.subplots(1, 2, figsize=(10.5, 3.6), sharey=True)
for ax, pol, name, n_soll in [(achsen[0], "Ez", "außerordentlich (n_e)", N_E),
                              (achsen[1], "Hz", "ordentlich (n_o)", N_O)]:
    feld = felder_db[pol]
    ax.imshow(np.real(feld).T, origin="lower", cmap="RdBu",
              extent=[-8, 8, -2.5, 2.5], aspect="auto")
    th = np.arcsin(np.sin(TH_EIN) / n_soll)
    zz = np.linspace(-2.5, 2.5, 10)
```

```
    versatz_strahl = -2.0          # Eintrittspunkt des Bündels (x bei y=2,5)
    ax.plot(versatz_strahl + (2.5 - zz) * np.tan(th), zz, "k--", lw=1.2)
    ax.set_title(f"{pol}-Welle: {name}")
    ax.set_xlabel("x")
achsen[0].set_ylabel("y")
fig.tight_layout()
plt.show()
```

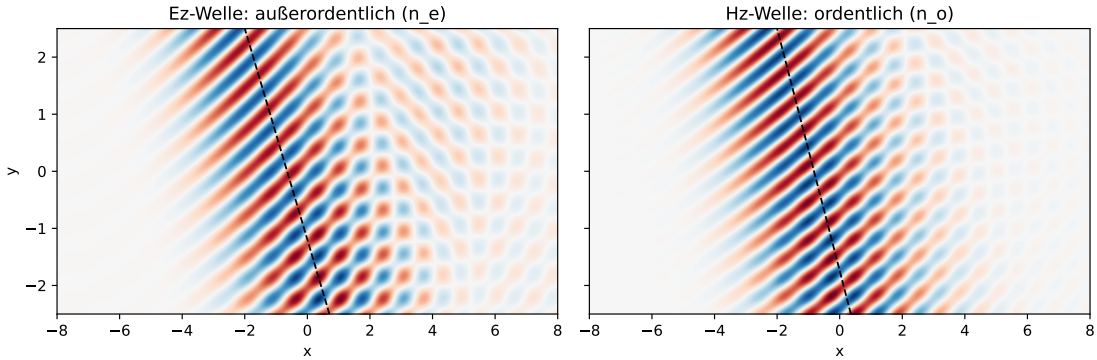


Abbildung 28.6: Ein Kristall, zwei Brechungswinkel: die Feldkarte im Kalkspat (Ausschnitt im Block) für die Ez-Welle (links, außerordentlich — sie spürt n_e = 1,486) und die Hz-Welle (rechts, ordentlich — sie spürt n_o = 1,658). Die gestrichelten Linien zeigen die jeweilige Snellius-Richtung; die per 2D-FFT gemessenen Wellenvektoren treffen Winkel und Index auf unter ein Prozent.

Zwei Polarisationen, zwei Winkel, zwei Indizes — gemessen auf unter ein Prozent. Schickst du **unpolarisiertes** Licht (eine Mischung aus beiden, Kapitel 13) durch den Kristall, trennt es sich in zwei Strahlen. Das ist die halbe Erklärung des berühmten Kalkspat-Doppelbilds. Die andere Hälfte ist verblüffender — und kommt jetzt.

28.8 Walk-off: das Doppelbild bei senkrechtem Einfall

Wer einen Kalkspat-Kristall auf eine Buchseite legt, sieht die Schrift doppelt — **auch wenn er senkrecht hindurchschaut**. Senkrechter Einfall heißt aber $\theta_1 = 0$, und Snellius liefert dann $\theta_2 = 0$ für *jeden* Index. Woher kommt das zweite Bild?

Der Punkt: Auf der Buchseite liegt die Kristallachse im Allgemeinen **schräg** zur Blickrichtung. Dann ist der ε-Tensor in unseren Koordinaten nicht mehr diagonal — er bekommt Außerdiagonal-Einträge, und die verkoppeln die Feldrichtungen: D_x hängt dann auch von E_y ab. Für die außerordentliche Welle bedeutet das, dass **Wellenfronten und Energiefluss nicht mehr parallel laufen**. Die Fronten laufen weiter geradeaus (der $\vec{k}$-Vektor gehorcht der Grenzflächen-Logik aus Kapitel 12), aber der Poynting-Vektor (Kapitel 18!) kippt um den „Walk-off"-Winkel ρ zur Seite. Der ordentliche Strahl kennt das Problem nicht — sein $\vec{E}$ zeigt aus der Ebene, für ihn ist der Kristall isotrop.

Für die Achse unter $45°$ liefert die Kristalloptik (wir zitieren sie hier, statt sie herzuleiten)

$$\tan\rho = \frac{n(45°)^2}{2}\left(\frac{1}{n_e^2} - \frac{1}{n_o^2}\right)\sin(2\cdot 45°), \qquad \frac{1}{n(45°)^2} = \frac{1/2}{n_o^2} + \frac{1/2}{n_e^2},$$

was für Kalkspat $\rho = 6{,}23°$ ergibt — über eine Kristalldicke von $t = 6$ also $6\cdot\tan\rho = 0{,}65$ seitlicher Versatz.

Versuchsaufbau. *Frage:* Wandert der außerordentliche Strahl seitlich, obwohl er senkrecht einfällt — und der ordentliche nicht? *Bühne:* 2D-Zelle 14×14, Kalkspat-Block $t = 6$, dessen Achse um $45°$ in der Ebene gedreht ist. Den gedrehten Tensor bauen wir von Hand: Die Drehung mischt ε_o und ε_e zu $\varepsilon_{xx} = \varepsilon_{yy} = (\varepsilon_o + \varepsilon_e)/2$ und erzeugt den Koppelterm $\varepsilon_{xy} = (\varepsilon_o - \varepsilon_e)/2$ — in Meep ist das ein `epsilon_offdiag`. *Anregung:* Gauß-Bündel senkrecht von oben, einmal E_z (ordentlich: spürt nur das ungedrehte $\varepsilon_{zz} = \varepsilon_o$), einmal H_z (außerordentlich: spürt den gedrehten Tensor in der Ebene). *Messgröße:* der seitliche Strahlversatz über den Block — jetzt darf es wieder der Schwerpunkt sein, denn bei senkrechtem Einfall ist die Reflexion winzig. *Erfolgskriterium:* E_z-Versatz $= 0$, H_z-Versatz $= 0{,}65$.

```
EPS_O, EPS_E = N_O**2, N_E**2
eps_mittel = (EPS_O + EPS_E) / 2
eps_koppel = (EPS_O - EPS_E) / 2
MAT_GEDREHT = mp.Medium(epsilon_diag=mp.Vector3(eps_mittel, eps_mittel,
                                                EPS_O),
                        epsilon_offdiag=mp.Vector3(eps_koppel, 0, 0))
T_WALK = 6.0
n_45 = 1 / np.sqrt(0.5 / N_O**2 + 0.5 / N_E**2)
tan_rho = (n_45**2 / 2) * (1 / N_E**2 - 1 / N_O**2)
print(f"Theorie: rho = {np.degrees(np.arctan(tan_rho)):.2f}°, "
```

```
      f"Versatz über t = {T_WALK}: {T_WALK * tan_rho:.3f}")


def walkoff(polarisation, until=150):
    dpml = 1.0
    sx, sy = 14.0, 14.0
    e0 = mp.Vector3(0, 0, 1) if polarisation == "Ez" else mp.Vector3(1, 0, 0)
    quelle = [mp.GaussianBeamSource(
        mp.ContinuousSource(F_OP),
        center=mp.Vector3(0, sy / 2 - dpml - 0.3),
        size=mp.Vector3(10.0, 0), beam_x0=mp.Vector3(0, 0),
        beam_kdir=mp.Vector3(0, -1), beam_w0=1.5, beam_E0=e0)]
    sim = mp.Simulation(cell_size=mp.Vector3(sx, sy), resolution=AUFL_2D,
                        Courant=0.25, force_complex_fields=True,
                        boundary_layers=[mp.PML(dpml)],
                        geometry=[mp.Block(size=mp.Vector3(mp.inf, T_WALK,
                                                           mp.inf),
                                           center=mp.Vector3(),
                                           material=MAT_GEDREHT)],
                        sources=quelle)
    sim.run(until=until)
    komponente = mp.Ez if polarisation == "Ez" else mp.Hz
    feld = sim.get_array(component=komponente, center=mp.Vector3(),
                         size=mp.Vector3(sx - 2 * dpml, sy - 2 * dpml))
    return feld, sx - 2 * dpml, sy - 2 * dpml


feld_o, bx_w, by_w = walkoff("Ez")
feld_e, _, _ = walkoff("Hz")
v_ord = strahl_versatz(feld_o, bx_w, by_w, t=T_WALK)
v_ausser = strahl_versatz(feld_e, bx_w, by_w, t=T_WALK)
print(f"Versatz ordentlich (Ez):        {v_ord:+.3f}")
print(f"Versatz außerordentlich (Hz):   {v_ausser:+.3f}")
```

```
Theorie: rho = 6.23°, Versatz über t = 6.0: 0.655
Versatz ordentlich (Ez):        -0.000
Versatz außerordentlich (Hz):   -0.653
```

```
fig, achsen = plt.subplots(1, 2, figsize=(10.0, 4.6), sharey=True)
for ax, feld, name, versatz in [(achsen[0], feld_o, "ordentlich (Ez)", v_ord),
                                (achsen[1], feld_e, "außerordentlich (Hz)",
                                 v_ausser)]:
```

```
    ax.imshow(np.abs(feld).T, origin="lower", cmap="inferno",
              extent=[-bx_w / 2, bx_w / 2, -by_w / 2, by_w / 2],
              vmax=0.8 * np.abs(feld_o).max())
    ax.axhline(-T_WALK / 2, color="w", lw=0.7)
    ax.axhline(T_WALK / 2, color="w", lw=0.7)
    ax.annotate("", xy=(1.6, 1.6), xytext=(-1.6, -1.6),
                arrowprops=dict(color="w", arrowstyle="<->", lw=1.0,
                                alpha=0.7))
    ax.set_title(f"{name}: Versatz {versatz:+.2f}")
    ax.set_xlabel("x")
achsen[0].set_ylabel("y")
fig.tight_layout()
plt.show()
```

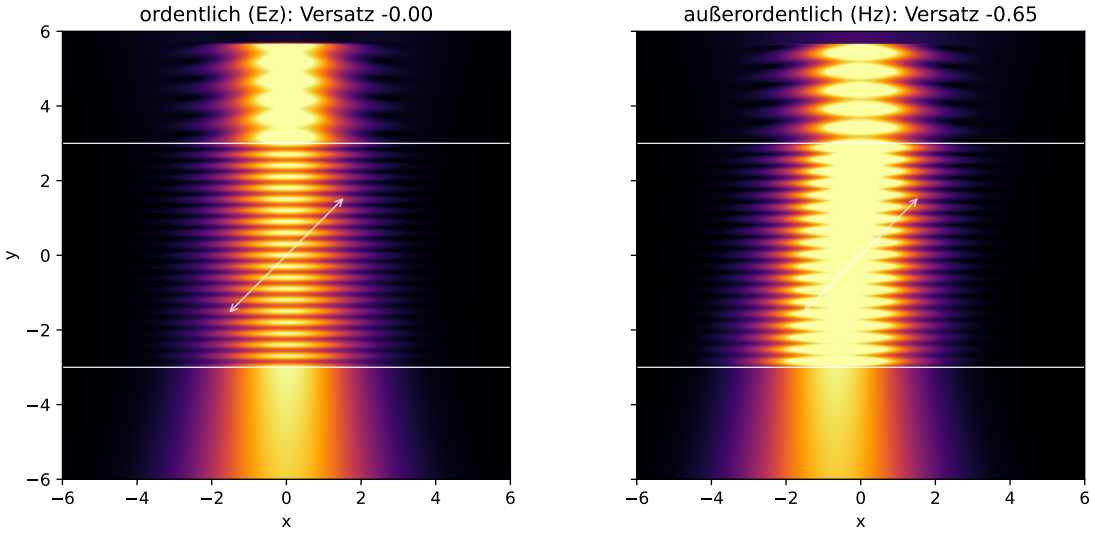


Abbildung 28.7: Das Doppelbild im Experiment: dasselbe Bündel fällt senkrecht auf den Kalkspat mit 45°-Achse (Doppelpfeil). Links die ordentliche Welle (Ez): sie läuft exakt gerade durch (Versatz 0,000). Rechts die außerordentliche (Hz): ihre Wellenfronten bleiben waagerecht, aber die Energie wandert um 6,2° zur Seite — unten tritt der Strahl 0,65 versetzt aus. Unpolarisiertes Licht enthält beide Anteile: zwei Bilder.

Der ordentliche Strahl: $0{,}000$. Der außerordentliche: $-0{,}65$ — Betrag exakt auf der Theorie (das Vorzeichen sagt nur, zu welcher Seite die Achse kippt). **Senkrechter Einfall, seitlicher Auszug**: Die Wellenfronten laufen stur geradeaus, aber die Energie schert aus — Poynting und $\vec{k}$ sind im anisotropen

Kristall keine Parallelen mehr. Beim NIM zeigten sie in *entgegengesetzte* Richtungen, hier in *verschiedene* — zwei Kapitel-18-Pointen in einem Kapitel. Dreht man den Kristall auf der Buchseite, rotiert das zweite Bild ums erste: Die Walk-off-Richtung folgt der Achse.

28.9 Die λ/4-Platte: aus linear wird zirkular

Zum Schluss lösen wir das älteste offene Versprechen dieses Buchs ein. In Kapitel 4 stand: Überlagert man zwei senkrecht zueinander polarisierte Wellen mit **90° Phasenverzug**, rotiert der Summenvektor — zirkulare Polarisation. Nur: Wie *macht* man die 90°? Jetzt haben wir das Werkzeug. In einem doppelbrechenden Plättchen laufen die beiden Feldkomponenten verschieden schnell — die längs der „schnellen" Achse (n_s) eilt der längs der „langsamen" (n_l) davon. Der Phasenverzug wächst mit der Dicke:

$$\Delta\varphi = 2\pi f\, d\,(n_l - n_s) \overset{!}{=} \frac{\pi}{2} \quad\Longrightarrow\quad d_{\lambda/4} = \frac{\lambda}{4\,(n_l - n_s)}.$$

Mit $n_s = 1{,}5$ und $n_l = 1{,}6$ (ein künstlicher Kristall mit ordentlich Kontrast — echter Quarz hat nur $\Delta n = 0{,}009$ und braucht entsprechend dickere oder trickreichere Platten) wird $d = \lambda/0{,}4/1 = 2{,}5$.

Versuchsaufbau. *Frage:* Macht das Plättchen aus 45°-linearem Licht zirkulares? *Bühne:* Eigentlich ist das ein 1D-Problem — aber Meeps `dimensions=1` kennt nur eine einzige E-Komponente, und wir brauchen zwei (Werkstatt-Notiz). Der Ausweg: eine schmale 2D-Zelle (Breite $0{,}25$) mit periodischem Rand (`k_point = 0`), die Welle läuft in y. Die beiden Polarisationen sind dann E_z und E_x — und der Kristalltensor $\mathrm{diag}(n_s^2,\ \cdot\ , n_l^2)$ gibt jeder ihren Index (ε_y spielt bei senkrechtem Durchlauf keine Rolle). *Anregung:* beide Komponenten gleich stark (das ist die 45°-Linearpolarisation), Dauerstrich $f = 1$, `force_complex_fields` — wir wollen die Phasen. *Messgröße:* die komplexen Amplituden E_x und E_z an einem Punkt hinter dem Plättchen: ihr Phasenunterschied und ihr Betragsverhältnis. *Erfolgskriterium:* $90°$ (modulo Vorzeichen — welcher von beiden eilt, hängt davon ab, welche Achse die schnelle ist) und Verhältnis nahe 1.

```
N_SCHNELL, N_LANGSAM = 1.5, 1.6
D_VIERTEL = 1 / (4 * F_OP * (N_LANGSAM - N_SCHNELL))
MAT_PLATTE = mp.Medium(epsilon_diag=mp.Vector3(N_SCHNELL**2, 1.0,
                                               N_LANGSAM**2))
```

```
def wellenplatte(dicke, until=300):
    """Quasi-1D: schmale 2D-Zelle, k_point=0, Welle läuft in y."""
    dpml, pad = 2.0, 4.0
    sy = 2 * (dpml + pad) + dicke
    sx = 0.25
    quellen = [mp.Source(mp.ContinuousSource(F_OP), component=c,
                         center=mp.Vector3(0, -sy / 2 + dpml + 1.0),
                         size=mp.Vector3(sx, 0))
               for c in (mp.Ez, mp.Ex)]      # 45°: beide gleich stark
    sim = mp.Simulation(cell_size=mp.Vector3(sx, sy), resolution=64,
                        Courant=0.25, force_complex_fields=True,
                        k_point=mp.Vector3(),
                        boundary_layers=[mp.PML(dpml, direction=mp.Y)],
                        geometry=[mp.Block(size=mp.Vector3(mp.inf, dicke,
                                                           mp.inf),
                                           center=mp.Vector3(),
                                           material=MAT_PLATTE)],
                        sources=quellen)
    sim.run(until=until)
    messpunkt = mp.Vector3(0, sy / 2 - dpml - 1.0)
    vor_punkt = mp.Vector3(0, -dicke / 2 - 0.5)
    return ((complex(sim.get_field_point(mp.Ex, messpunkt)),
             complex(sim.get_field_point(mp.Ez, messpunkt))),
            (complex(sim.get_field_point(mp.Ex, vor_punkt)),
             complex(sim.get_field_point(mp.Ez, vor_punkt))))


(ex_nach, ez_nach), (ex_vor, ez_vor) = wellenplatte(D_VIERTEL)
verzug = np.degrees(np.angle(ex_nach) - np.angle(ez_nach)) % 360
print(f"d = {D_VIERTEL}:")
print(f"  Phasenverzug Ex gegen Ez: {verzug:.1f}°  (Soll 270° ≡ −90°)")
print(f"  Betragsverhältnis |Ex|/|Ez|: {abs(ex_nach) / abs(ez_nach):.3f}")
```

```
d = 2.499999999999998:
  Phasenverzug Ex gegen Ez: 269.5°  (Soll 270° ≡ −90°)
  Betragsverhältnis |Ex|/|Ez|: 0.924
```

Der Verzug trifft die Viertelwelle (es sind $-90°$, weil bei uns die E_x-Achse die schnelle ist und ihre Phase *weniger* nachläuft). Das Betragsverhältnis ist $0{,}92$ statt $1{,}00$ — die beiden Achsen sehen leicht verschiedene Fresnel-Verluste und Mehrfachechos im Plättchen (Kapitel 27 lässt grüßen: Jedes

Plättchen ist auch ein schwaches Etalon; reale Wellenplatten werden deshalb entspiegelt). Eine $0{,}92$-Ellipse ist von einem Kreis mit bloßem Auge kaum zu unterscheiden.

Was haben wir gebaut? Vor dem Plättchen schwingen E_x und E_z im Takt — der Summenvektor pendelt auf einer 45°-Linie. Dahinter ist E_x um eine Viertelperiode verschoben: Wenn E_z maximal ist, geht E_x gerade durch null und umgekehrt — der Summenvektor **rotiert**. Das ist zirkulare Polarisation, das Versprechen aus Kapitel 4, eingelöst mit einem Stück anisotropem Material.

Wozu das alles? Zirkular polarisiertes Licht steckt in 3D-Kinobrillen (links- und rechtsdrehend für die beiden Augen — und es bleibt zirkular, wenn man den Kopf neigt, anders als bei linearen Filtern), in CD/DVD-Laufwerken (der Isolator aus Polarisator + λ/4-Platte schützt die Laserdiode vor ihrem eigenen Reflex) und in jedem LCD-Bildschirm — dessen Flüssigkristalle sind nichts anderes als **schaltbare Wellenplatten**, deren Verzug eine Spannung einstellt (mehr im Kleingedruckten).

28.10 Das Kapitel-Programm

`programme/kap28/kap28_metamaterial_doppelbrechung.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die Eichung $\varepsilon(1) = \mu(1) = -1$, die Smith-Dreierreihe samt Transfer-Matrix-Vergleich und Lambert-Beer-Gegenprobe, das Phasenprofil ($\pm 1{,}00\, k_0$), die Gruppenlaufzeit gegen das handgerechnete n_g, den Vorzeichenwechsel des Brechungsversatzes, die Flachlinse (Fokuslage und FWHM), die Doppelbrechungs-Winkel und -Indizes per 2D-FFT, den Walk-off (ordentlich exakt null), beide Wellenplatten und die Drahtgitter-Klippen. Läuft in wenigen Minuten.

Merkkasten

- **Metamaterial ≠ Kristall:** Bausteine $a \ll \lambda$ wirken als homogenes Medium (effektives ε, μ); Bausteine $a \sim \lambda$ machen Interferenz (Kapitel 26).
- **Ein negatives Vorzeichen sperrt, zwei öffnen:** $\varepsilon < 0$ allein → evaneszent (Kapitel 16/17); $\varepsilon, \mu < 0$ → Welle läuft mit $n = -\sqrt{\varepsilon\mu}$. Energie vorwärts, Phase rückwärts ($\vec{S}$ antiparallel $\vec{k}$); Brechung knickt auf die falsche Seite, eine flache Platte wird zur Linse (Bild bei $t - d_1$).
- **Negative Konstanten gibt es nur mit Dispersion** — und $\mu < 0$ nur nahe einer Resonanz (Split-Ring = Kapitel-19-Schwingkreis).

Darum sind Verluste der Systempreis der Metamaterialien (Kapitel 15: die Absorption wohnt an der Resonanz).

- **Anisotropie = ein Index pro Feldrichtung:** ε als Tensor (`epsilon_diag`/`offdiag`). Doppelbrechung sortiert nach Polarisation (n_o/n_e); bei schräger Achse scheren Energiefluss und Wellenvektor auseinander (Walk-off → Doppelbild).
- **Wellenplatten sind Phasen-Werkzeuge:** $d_{\lambda/4} = \lambda/(4\,\Delta n)$ macht aus 45°-linear zirkular (Kapitel-4-Zusage); die λ/2-Platte spiegelt die Polarisationsrichtung.
- **Werkstatt:** μ-dispersive Materialien dürfen weder PML noch Absorber berühren (Rückwärtswellen drehen die Dämpfung um) — Blöcke endlich bauen, Luftpuffer lassen.

Roter Faden

Dieses Kapitel erntet quer durchs Buch: Die **Drude-Klippe aus Kapitel 16** wird zur Stellschraube (Drahtgitter), der **Schwingkreis aus Kapitel 19** wird zum Materialparameter (Split-Ring macht $\mu(f)$ zur Lorentz-Linie aus **Kapitel 15** — mitsamt ihrer Verluste), **Snellius aus Kapitel 12** bekommt ein Vorzeichen, der **Poynting-Vektor aus Kapitel 18** trennt sich vom Wellenvektor (antiparallel im NIM, schräg im Kristall), das **Tunneln aus Kapitel 17** erklärt die weichen Flanken des Smith-Fensters, und die **Polarisation aus Kapitel 4/13** wird endlich zum Werkzeug: Die λ/4-Platte löst die älteste offene Zusage des Buchs ein. Nach vorn: Zweimal haben wir uns in diesem Kapitel auf „das gibt es nur mit Dispersion" berufen — **Kapitel 29** zeigt, dass dahinter ein Satz steckt: Kausalität *erzwingt* Dispersion (Kramers-Kronig), und Real- und Imaginärteil von $\varepsilon(f)$ sind keine unabhängigen Größen.

Übungen

Ü 28.1 — Die Flachlinse vermessen (Verstehen). Eine Veselago-Platte ($n = -1$) ist $t = 4$ cm dick. (a) Eine Punktquelle sitzt $d_1 = 1{,}5$ cm vor der Platte — wo liegen das innere und das äußere Bild? Konstruiere beide mit dem Snellius-Knick $\theta_2 = -\theta_1$ (eine Skizze mit zwei, drei Strahlen genügt). (b) Was ändert sich bei $d_1 = 3$ cm? (c) Was passiert für $d_1 = 5$ cm — und was sagt das über den Unterschied zwischen dieser „Linse" und einer Glaslinse mit Brennweite?

Musterlösung zu Ü 28.1

Bei $n = -1$ wird jeder Strahl an beiden Flächen exakt gespiegelt gebrochen ($\theta_2 = -\theta_1$): Ein Strahl, der unter dem Winkel θ von der Quelle wegläuft, läuft in der Platte unter $-\theta$ weiter und hinter ihr wieder unter θ. Die Geometrie ist dadurch ein reines Dreisatz-Problem: Alle Strahlen kreuzen die Achse erst bei der Tiefe d_1 **in** der Platte (inneres Bild), dann im Abstand $t - d_1$ **hinter** ihr.

```
import numpy as np
import matplotlib.pyplot as plt

T_P, D1_U = 4.0, 1.5
fig, ax = plt.subplots(figsize=(7.0, 3.6))
ax.axvspan(0, T_P, color="0.88")
for steigung in [0.5, 0.25, -0.25, -0.5]:
    # Quelle bei x = −d1; Knick an x = 0, Gegenknick in der Platte
    x_pfad = [-D1_U, 0, T_P, T_P + (T_P - D1_U)]
    y_pfad = [0, steigung * D1_U,
              steigung * D1_U - steigung * T_P,
              0]
    ax.plot(x_pfad, y_pfad, "C0-", lw=1.2)
ax.plot(-D1_U, 0, "ko", ms=6)
ax.plot(D1_U, 0, "C3x", ms=8)
ax.plot(T_P + (T_P - D1_U), 0, "C3x", ms=8)
ax.annotate("Quelle", (-D1_U, 0.15), ha="center")
ax.annotate("inneres Bild", (D1_U, -0.45), ha="center", color="C3")
ax.annotate("äußeres Bild", (T_P + (T_P - D1_U), 0.3), ha="center",
            color="C3")
ax.set_xlabel("Achse (grau: die Platte, t = 4)")
ax.set_ylabel("Querrichtung")
ax.set_ylim(-1.4, 1.4)
ax.grid(alpha=0.3)
plt.show()
print(f"(a) d1 = 1,5: inneres Bild {D1_U} cm tief, "
      f"äußeres {T_P - D1_U} cm hinter der Platte")
print(f"(b) d1 = 3,0: inneres Bild 3 cm tief, äußeres {T_P - 3.0} cm dahinter")
```

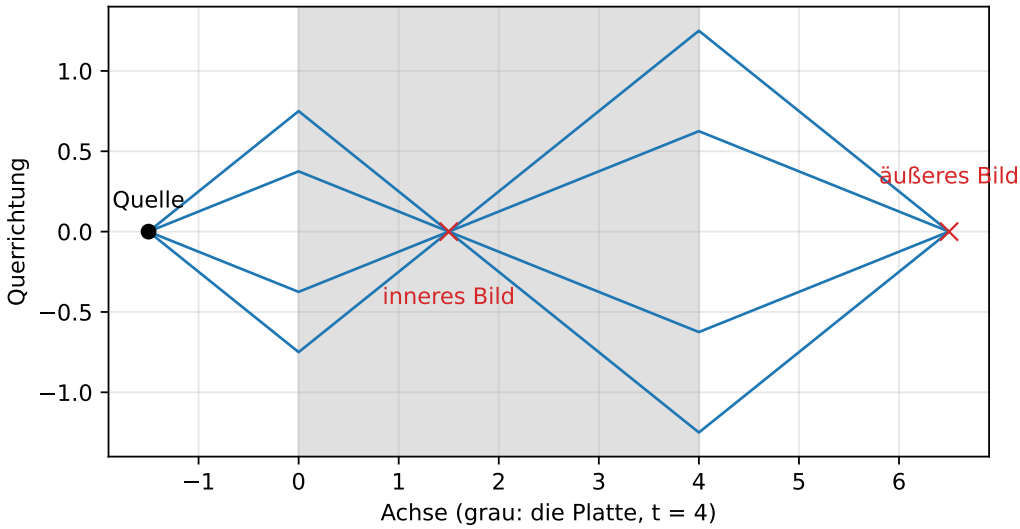


Abbildung 28.8: Strahlkonstruktion zu Ü 28.1 (a): Jeder Strahl knickt an beiden Flächen auf die andere Seite der Normalen. Inneres Bild bei Tiefe $d_1 = 1{,}5$ in der Platte, äußeres bei $t - d_1 = 2{,}5$ dahinter.

```
(a) d1 = 1,5: inneres Bild 1.5 cm tief, äußeres 2.5 cm hinter der Platte
(b) d1 = 3,0: inneres Bild 3 cm tief, äußeres 1.0 cm dahinter
```

(c) Für $d_1 = 5 > t$ gibt es **kein Bild mehr**: Die Strahlen knicken zwar, kreuzen die Achse aber schon nicht mehr innerhalb der Platte (d_1 passt nicht hinein) — und hinter der Platte laufen sie auseinander wie zuvor. Die Veselago-Platte hat also **keine Brennweite**: Sie kann nur abbilden, was näher als t vor ihr liegt, und sie kann parallele Strahlen (Quelle im Unendlichen) überhaupt nicht bündeln. Eine Glaslinse bildet jede Objektweite ab — die Flachlinse verschiebt jedes Nahfeld-Bild nur um die feste Strecke $2t$ nach hinten ($d_1 +$ Bildabstand $= d_1 + t - d_1 + t$... besser gemerkt: Quelle→Bild ist immer $2t$).

Ü 28.2 — Die λ/2-Platte (Verändern). Verdopple im Wellenplatten-Aufbau die Dicke auf $d = 5$. (a) Sage den Phasenverzug vorher. (b) Miss ihn nach. (c) Die Eingangswelle war 45°-linear ($E_x = E_z$). Was für eine Polarisation kommt heraus? Zeichne die E-Vektor-Bahn aus den gemessenen Amplituden und bestimme ihren Winkel — welche Operation führt die λ/2-Platte an der Polarisationsrichtung aus?

Musterlösung zu Ü 28.2

(a) Doppelte Dicke, doppelter Verzug: $180°$.

```
(ex_h, ez_h), _ = wellenplatte(2 * D_VIERTEL)
verzug_h = np.degrees(np.angle(ex_h) - np.angle(ez_h)) % 360
print(f"(b) Verzug = {verzug_h:.1f}°  (Soll 180°), "
      f"|Ex|/|Ez| = {abs(ex_h) / abs(ez_h):.4f}")

# (c) E-Vektor-Bahn und ihr Hauptachsen-Winkel
fein = np.linspace(0, 2 * np.pi, 400)
bahn_x = np.real(ex_h * np.exp(-1j * fein))
bahn_z = np.real(ez_h * np.exp(-1j * fein))
i_spitze = np.argmax(np.hypot(bahn_x, bahn_z))
winkel_bahn = np.degrees(np.arctan2(bahn_x[i_spitze], bahn_z[i_spitze]))
print(f"(c) Bahn-Winkel = {winkel_bahn:+.1f}° gegen die z-Achse "
      f"(Eingang: +45°)")
```

```
(b) Verzug = 179.4°  (Soll 180°), |Ex|/|Ez| = 1.0000
(c) Bahn-Winkel = -45.0° gegen die z-Achse (Eingang: +45°)
```

(c) Heraus kommt wieder **lineares** Licht (das Betragsverhältnis ist 1,000 und die Bahn eine Linie), aber die Schwingungsrichtung liegt bei $-45°$: Die λ/2-Platte **spiegelt die Polarisationsrichtung an ihren Achsen** — aus $+45°$ wird $-45°$, allgemein aus dem Winkel α der Winkel $-\alpha$. Deshalb ist sie das Standard-Werkzeug, um Polarisationen zu *drehen* (Platte um $\beta/2$ drehen → Polarisation dreht um β), etwa vor polarisierenden Strahlteilern.

Ü 28.3 — Das Drahtgitter in echt (Übertragen). Im Kapitel haben wir den Drähten einfach eine Drude-Formel *zugeschrieben*. Baue das Metamaterial-Atom jetzt wirklich: eine einzelne Reihe-um-Reihe-Zelle ($s_y = a = 0{,}2$, periodischer Rand mit `k_point = 0`) mit zehn PEC-Stäbchen (`mp.metal`) im Abstand $a = 0{,}2$ quer zum Strahl, Radius $r = 0{,}01$ — das Gitter ist also viel feiner als die Wellenlänge ($a/\lambda = 0{,}2$ bei $f = 1$). (a) Miss $T(f)$ von $0{,}1$ bis $2{,}5$ und finde die Klippe. (b) Wiederhole mit $r = 0{,}02$: Wandert die Klippe nach oben oder unten? Begründe mit dem Induktivitäts-Argument aus Kapitel 22. (c) Die Pendry-Faustformel $f_p \approx \sqrt{2\pi/\ln(a/r)}\,/\,(2\pi a)$ liefert $1{,}15$ bzw. $1{,}3$ — du wirst deutlich höhere Klippen messen. Sieh dir a/λ *an der Klippe* an: In welchem Regime sind wir dort, und welches Kapitel ist dann zuständig?

Musterlösung zu Ü 28.3

```
A_G, N_REIHEN = 0.2, 10


def drahtgitter_sweep(radius, fcen=1.3, df=2.4, nfreq=361):
    dpml, pad = 1.0, 2.0
    breite = N_REIHEN * A_G
    sx = 2 * (dpml + pad) + breite
    sy = A_G
    leistungen = []
    for mit_gitter in (False, True):
        geometrie = [mp.Cylinder(radius=radius,
                                 center=mp.Vector3(-breite / 2 + A_G / 2
                                                   + i * A_G, 0),
                                 material=mp.metal)
                     for i in range(N_REIHEN)] if mit_gitter else []
        quelle = [mp.Source(mp.GaussianSource(fcen, fwidth=df),
                            component=mp.Ez,
                            center=mp.Vector3(-sx / 2 + dpml + 0.5, 0),
                            size=mp.Vector3(0, sy))]
        sim = mp.Simulation(cell_size=mp.Vector3(sx, sy), resolution=200,
                            k_point=mp.Vector3(),
                            boundary_layers=[mp.PML(dpml, direction=mp.X)],
                            geometry=geometrie, sources=quelle)
        x_mess = sx / 2 - dpml - 0.5
        tr = sim.add_flux(fcen, df, nfreq,
                          mp.FluxRegion(center=mp.Vector3(x_mess, 0),
                                        size=mp.Vector3(0, sy)))
        sim.run(until_after_sources=mp.stop_when_fields_decayed(
            50, mp.Ez, mp.Vector3(x_mess, 0), 1e-9))
        leistungen.append(np.array(mp.get_fluxes(tr)))
    fs = np.array(mp.get_flux_freqs(tr))
    return fs, leistungen[1] / leistungen[0]


fig, ax = plt.subplots(figsize=(7.0, 3.8))
for radius, farbe in [(0.01, "C0"), (0.02, "C3")]:
    fs_d, T_d = drahtgitter_sweep(radius)
    ueber = T_d > 0.5
    klippe = fs_d[np.argmax(ueber)] if ueber.any() else float("nan")
    pendry = np.sqrt(2 * np.pi / np.log(A_G / radius)) / (2 * np.pi * A_G)
    ax.plot(fs_d, T_d, "-", color=farbe,
            label=f"r = {radius}: Klippe {klippe:.2f} (Pendry {pendry:.2f})")
    ax.axvline(klippe, color=farbe, lw=0.8, ls=":")
    print(f"r = {radius}: Klippe (T = 0,5) bei f = {klippe:.3f}, "
          f"Pendry-Formel {pendry:.2f}, a/lambda an der Klippe: "
          f"{A_G * klippe:.2f}")
ax.set_xlabel("Frequenz f")
ax.set_ylabel("Transmission T")
ax.set_title("Zehn Reihen dünner Stäbchen = ein Plasma")
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.show()
```

```
r = 0.01: Klippe (T = 0,5) bei f = 1.447, Pendry-Formel 1.15, a/lambda an der Klippe: 0.29
r = 0.02: Klippe (T = 0,5) bei f = 1.813, Pendry-Formel 1.31, a/lambda an der Klippe: 0.36
```

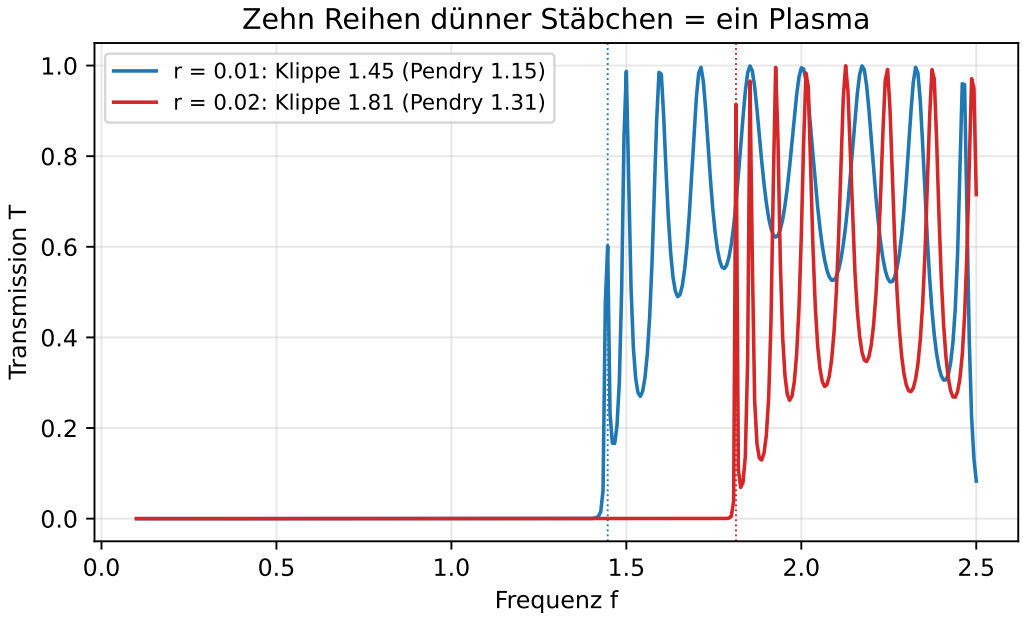


Abbildung 28.9: Das Drahtgitter als Plasma: Transmission durch zehn Stäbchen-Reihen für zwei Drahtradien. Unterhalb der Klippe ist das Gitter dicht wie die Ionosphäre für Kurzwelle — obwohl es zu über 99 % aus Luft besteht. Dickere Drähte verschieben die Klippe nach oben.

(a), (b) Die Klippe liegt für $r = 0{,}01$ bei $f \approx 1{,}45$ und wandert für $r = 0{,}02$ auf $\approx 1{,}8$ — **dickere Drähte sperren bis höher hinauf.** Das Induktivitäts-Argument: f_p ist die Frequenz, bei der die träge Strom-Antwort des Gitters die Feld-Antwort gerade noch einholt. Die Trägheit kommt hier nicht von der Elektronenmasse, sondern von der **Induktivität** der dünnen Drähte (Kapitel 22: $L' \propto \ln(a/r)$) — dünnere Drähte sind induktiver, also träger, also sperrt ihr Gitter nur bis zu einer *tieferen* Frequenz.
(c) An der Klippe ist $a/\lambda = 0{,}29$ bzw. $0{,}36$ — das ist **kein** sauberes Metamaterial-Regime mehr ($a \ll \lambda$ verlangt eher $a/\lambda \lesssim 0{,}1$), sondern der Vorhof von Kapitel 26: Bragg-Interferenz zwischen den Reihen mischt mit, und die hübsche Pendry-Formel (hergeleitet für $a/\lambda \to 0$) wird zur Größenordnungs-Schätzung. Merke: Ob etwas „Medium" oder „Kristall" ist, entscheidet nicht das Bauteil, sondern das Verhältnis a/λ

— ein und dasselbe Drahtgitter ist bei tiefen Frequenzen ein Plasma und bei hohen ein photonischer Kristall.

Ü 28.4 — Die Achillesferse (Übertragen). Die Flachlinse aus dem Kapitel lief mit $\gamma = 0{,}01$. Reale Metamaterialien — besonders optische — sind verlustreicher. Wiederhole den Linsenlauf mit $\gamma = 0{,}05$ und $\gamma = 0{,}15$ (in *beiden* Bausteinen). (a) Sage qualitativ vorher, was mit dem Fokus passiert. (b) Miss Fokuslage und Spitzenintensität. (c) Erkläre mit Kapitel 15, warum man dieses Problem nicht einfach „wegoptimieren" kann.

Musterlösung zu Ü 28.4

```
def nim_mit(gamma):
    return mp.Medium(
        epsilon=1,
        E_susceptibilities=[mp.DrudeSusceptibility(frequency=F_OP,
                                                   gamma=gamma, sigma=2.0)],
        mu=1,
        H_susceptibilities=[mp.LorentzianSusceptibility(frequency=F0_RING,
                                                        gamma=gamma,
                                                        sigma=S_RING)])


print("gamma   Fokuslage   Spitzenintensität")
for gamma_u in [0.01, 0.05, 0.15]:
    ez_u, bx_u, by_u = flachlinse(material=nim_mit(gamma_u))
    nx_u, ny_u = ez_u.shape
    xs_u = np.linspace(-bx_u / 2, bx_u / 2, nx_u)
    ys_u = np.linspace(-by_u / 2, by_u / 2, ny_u)
    int_u = np.abs(ez_u)**2
    profil_u = np.where(ys_u > 1.2, int_u[np.argmin(np.abs(xs_u)), :], 0)
    iy_u = np.argmax(profil_u)
    print(f"{gamma_u:5.2f}   y = {ys_u[iy_u]:+.2f}    {profil_u[iy_u]:.4f}")
```

```
gamma   Fokuslage   Spitzenintensität
 0.01   y = +1.91    0.4449
 0.05   y = +1.76    0.0041
 0.15   y = +1.29    0.0000
```

(a), (b) Der Fokus **stirbt dramatisch**: Schon $\gamma = 0{,}05$ kostet einen Faktor 100 an Intensität (von $0{,}44$ auf $0{,}004$), bei $0{,}15$ bleibt praktisch nichts — und was bleibt, rückt zur Platte hin (das „Bild" ist dann nur noch das durchsickernde Nahfeld der Plattenrückseite). Zum Vergleich:

$\gamma = 0{,}05$ heißt im Lorentz-Modell immer noch ein Resonator-Q von $f_0/\gamma = 16$ — kein schlechter Schwingkreis.
(c) Das Kapitel-15-Argument: $\mu < 0$ existiert **nur nahe der Ringresonanz**, und ein getriebener Resonator setzt genau dort am meisten Leistung um — Verlust und Funktion wohnen an derselben Adresse, man kann nicht das eine behalten und das andere räumen. Bei Mikrowellen (Kupferringe, hohes Q) geht es leidlich; im Sichtbaren müssen die „Ringe" aus Plasmonik-Metallen gebaut werden (Kapitel 16: Gold mit seinen Drude-Verlusten), und dort hat genau diese Rechnung der Pendry-Superlinse die Flügel gestutzt. Merksatz fürs Berufsleben: Wenn ein Wundermaterial eine Resonanz braucht, frage zuerst nach dem Imaginärteil.

Das Kleingedruckte

Die perfekte Linse und ihr Ende. Pendrys Rechnung von 2000 zeigte: Die verlustfreie $n = -1$-Platte verstärkt evaneszente Wellen exponentiell wieder auf — die Information über Sub-Wellenlängen-Details, die nach Kapitel 12 im Abstand $\sim \lambda$ verloren geht, käme im Bild vollständig an („perfect lens", Auflösung unbegrenzt). Der Haken steckt im selben Exponenten: Schon winzige Verluste kappen die Verstärkung der feinsten Details zuerst, und mit realistischen $\mathrm{Im}(\varepsilon)$ bleibt von „perfekt" ein Faktor zwei, drei unter der Beugungsgrenze — im Nahfeld, als „Superlinse" aus einem schlichten Silberfilm sogar experimentell bestätigt (2005). Unser FWHM von $0{,}41\lambda$ statt $0{,}5\lambda$ ist ein matter Abglanz dieses Effekts.

Transformationsoptik und Tarnkappen. Die Metamaterial-Idee in voller Größe: Weil Maxwell unter Koordinatentransformationen forminvariant ist, kann man eine *gedachte* Verbiegung des Raums (etwa: „alle Strahlen fließen um diese Kugel herum") in ein *Rezept* für ortsabhängige ε- und μ-Tensoren übersetzen. Die Mikrowellen-„Tarnkappe" von 2006 (Schurig/Smith) war genau das — ein Ring aus Split-Ring-Zellen mit radial verlaufenden Materialprofilen. Die Grenzen sind dieselben wie immer in diesem Kapitel: schmalbandig (Dispersion!) und verlustig.

Flüssigkristalle: die schaltbare Wellenplatte. Die stäbchenförmigen Moleküle eines nematischen Flüssigkristalls sind anisotrop polarisierbar (Δn bis $0{,}3$ — mehr als Kalkspat) und lassen sich mit ein paar Volt ausrichten. Ein LCD-Pixel ist eine Zelle, deren Verzug die Spannung zwischen „λ/2-Platte" (Licht passiert den gekreuzten Ausgangspolarisator) und „neutral" (Licht wird geschluckt) durchstimmt. Dass dein Monitor unter polarisierter Sonnenbrille seltsam aussieht, ist angewandte Doppelbrechung.

Chiralität und Bianisotropie. Unsere Tensoren waren symmetrisch und reell. Materialien mit Schraubenstruktur (Zuckerlösung, manche Metamaterial-Zellen) koppeln $\vec{E}$ an $\vec{B}$ — sie drehen die Polarisationsebene kontinuierlich (optische Aktivität) oder unterscheiden links- und rechtszirkular in der Absorption. Der Saccharimeter des Zuckersieders ist Polarisationsoptik von 1840, die bis heute in jeder Zuckerfabrik steht.

Negativer Index ohne Ringe. Photonische Kristalle (Kapitel 26) können nahe der Bandkante effektiv $n < 0$ zeigen — über Interferenz statt über Resonanzen, mit geringeren Verlusten, aber nur in schmalen Bändern und Winkelbereichen. Auch hier gilt: Die Natur gibt das Vorzeichen nicht her, man muss es ihr abringen.

29 Kramers-Kronig: Kausalität erzwingt Dispersion

Dieses Buch hat zwei offene Rechnungen. In Kapitel 16 haben wir Wellenberge gemessen, die mit $1{,}67\,c$ durch ein Drude-Medium liefen — und beruhigt: Die *Front* eines Signals sei trotzdem nie schneller als c, das zeigen wir später. In Kapitel 28 haben wir uns zweimal darauf berufen, dass es negative Materialkonstanten „nur mit Dispersion" gibt — auch das ohne Beweis. Beide Schulden begleicht dieses Kapitel, und zwar mit einem einzigen Gedanken, den wir seit Kapitel 20 besitzen: **Materie kann nicht antworten, bevor das Feld da ist.**

Aus diesem harmlosen Satz folgt Erstaunliches. Der niederländische Physiker Hendrik Kramers und sein deutscher Kollege Ralph Kronig zeigten 1926/27: Wenn die Antwort eines Materials kausal ist, dann sind sein Realteil (die Brechung) und sein Imaginärteil (die Absorption) **keine unabhängigen Größen**. Wer die Absorption bei *allen* Frequenzen kennt, kann die Brechzahl bei *jeder* Frequenz ausrechnen — ohne ein einziges weiteres Experiment. Ein Material kann nicht erst beim Brechen „ja" und beim Schlucken „nein" sagen; beides ist dieselbe Auskunft, einmal als Echo und einmal als Verspätung gelesen. In diesem Kapitel bauen wir diese Rechnung nach: erst am Lorentz-Modell aus Kapitel 16 (maschinen-genau), dann an Meeps echter Gold-Karte — und zum Schluss messen wir im selbstgebauten FDTD, dass die Signalfront tatsächlich nie schneller als c ist, selbst wenn die Berge das Licht überholen.

Lernziele

Nach diesem Kapitel kannst du …

1. … die **Antwortfunktion** $\chi(t)$ als Gedächtnis der Materie lesen: Die Polarisation ist eine Faltung über die Feld-Vergangenheit, und $\chi(t)$ des Lorentz-Glases ist wörtlich die gedämpfte Schwingung aus Kapitel 19,
2. … erklären, warum Kausalität Real- und Imaginärteil von $\chi(\omega)$ verkettet — mit nichts als der Zerlegung in gerade und ungerade Anteile,

3. … das **Kramers-Kronig-Integral** numerisch ausführen (Hauptwert per Subtraktions-Trick) und am Lorentz-Modell auf zehn Stellen verifizieren,
4. … die **Kausalitäts-Probe** führen: $\chi(t)$ per inverser FFT ausrechnen und prüfen, ob vor $t = 0$ wirklich nichts kommt — und du weißt, warum ein perfekt durchsichtiges Glas mit $n \neq 1$ daran scheitert,
5. … erklären, warum das KK-Integral das *ganze* Spektrum braucht (Meeps Gold-Karte: oben abschneiden verzeiht, unten nicht),
6. … messen, dass die **Signalfront** ein Medium ohne jede Verzögerung durchquert — auch bei $v_p = 1{,}67\,c$.

29.1 Die Antwort kommt zu spät

Bisher haben wir Materialantworten im Frequenzbild geschrieben: $P(\omega) = \varepsilon_0\,\chi(\omega)\,E(\omega)$ — bei jeder Frequenz ein eigener Proportionalitätsfaktor. Im Zeitbild steckt hinter so einer frequenzweisen Multiplikation immer eine **Faltung** (das ist der Faltungssatz aus Kapitel 23, hier rückwärts gelesen):

$$P(t) = \varepsilon_0 \int_0^\infty \chi(\tau)\,E(t-\tau)\,\mathrm{d}\tau.$$

Die Gleichung erzählt eine kleine Geschichte: Die Polarisation **jetzt** (t) entsteht aus dem Feld von **vorhin** ($t-\tau$), gewichtet mit einer Funktion $\chi(\tau)$, die angibt, wie stark sich die Materie an ein Feld erinnert, das vor der Zeitspanne τ anlag. $\chi(t)$ heißt darum **Antwortfunktion** — das Gedächtnis des Materials. Und die untere Integralgrenze 0 ist keine Bequemlichkeit, sondern Physik: Bei negativem τ stünde dort das Feld der **Zukunft**. Kausalität (Kapitel 20) heißt genau

$$\chi(t) = 0 \quad \text{für } t < 0.$$

Wie sieht so ein Gedächtnis aus? Für das Lorentz-Glas aus Kapitel 16 können wir es erraten, ohne zu rechnen: Dort hängt an jedem Atom ein gedämpfter Oszillator ($\ddot{P} + \gamma\dot{P} + \omega_0^2 P = \omega_p^2\,E$, das ist Kapitel 19 in Materialform). Gibt man so einem Oszillator einen einzigen Stoß — ein Delta-Blitz von E —, dann schwingt er gedämpft aus. Die Antwortfunktion *ist* diese Ausschwingkurve:

$$\chi(t) = \frac{\omega_p^2}{\nu}\,\mathrm{e}^{-\gamma t/2}\,\sin(\nu t) \quad \text{für } t > 0, \qquad \nu = \sqrt{\omega_0^2 - \gamma^2/4}.$$

Das prüfen wir jetzt nach — und bauen uns dabei gleich das Werkzeug für den Rest des Kapitels. **Versuchsaufbau:** Die Frage ist, ob das frequenzweise $\chi(\omega) = \omega_p^2/(\omega_0^2 - \omega^2 - \mathrm{i}\gamma\omega)$ aus Kapitel 16 im Zeitbild wirklich diese Ausschwingkurve ergibt — und ob vor $t = 0$ wirklich *nichts* kommt. Die Bühne ist ein reines NumPy-Rechenblatt: Wir werten $\chi(\omega)$ auf einem feinen, symmetrischen Frequenzgitter aus (± 200, 2^{18} Punkte) und übersetzen mit der inversen Fourier-Transformation $\chi(t) = \frac{1}{2\pi}\int \chi(\omega)\,\mathrm{e}^{-\mathrm{i}\omega t}\,\mathrm{d}\omega$ ins Zeitbild. Gemessen werden zwei Zahlen: der größte χ-Wert **vor** $t = 0$ (Erfolg: praktisch null) und die größte Abweichung von der Kapitel-19-Formel **nach** $t = 0$ (Erfolg: ebenso).

```
import matplotlib.pyplot as plt
import numpy as np

# das Lorentz-Glas aus Kapitel 16: Resonanz, Daempfung, Staerke
W0, GAMMA, WP = 1.0, 0.1, 1.0


def chi_lorentz(w):
    """χ(ω) des Lorentz-Oszillators (Zeitfaktor e^{-iωt})."""
    return WP**2 / (W0**2 - w**2 - 1j * GAMMA * w)


# symmetrisches Frequenzgitter: negative UND positive Frequenzen,
# denn das Zeitsignal entsteht aus beiden Haelften gemeinsam
NT = 2**18
W_FFT = 200.0
w_fft = np.linspace(-W_FFT, W_FFT, NT, endpoint=False)
dw = w_fft[1] - w_fft[0]


def antwortfunktion(chi_w):
    """χ(t) = (1/2π) ∫ χ(ω) e^{-iωt} dω, Zeitachse aufsteigend sortiert."""
    # np.fft.ifftshift: Gegenstueck zu fftshift aus Kap. 28 — schiebt die
    # Nullfrequenz von der Gittermitte an den Anfang, wie die FFT es will
    chi_t = np.fft.fft(np.fft.ifftshift(chi_w)) * dw / (2 * np.pi)
    # np.fft.fftfreq: die VOLLE Frequenzachse zur FFT (positive und
    # negative Werte) — rfftfreq aus Kap. 10 war nur die halbe
    ts = np.fft.fftfreq(NT, d=dw / (2 * np.pi))
    # np.argsort liefert die Indizes, die ein Array sortieren wuerden —
    # damit bringen wir Zeitachse und Werte gemeinsam in Reihenfolge
    ordnung = np.argsort(ts)
    return ts[ordnung], chi_t[ordnung]
```

```
ts, chi_t = antwortfunktion(chi_lorentz(w_fft))
chi_t = chi_t.real          # Imaginaerteil ~ 1e-12 (Realitaets-Check)

vorher = np.abs(chi_t[ts < -0.5]).max() / np.abs(chi_t).max()
nu = np.sqrt(W0**2 - GAMMA**2 / 4)
maske = (ts > 0.5) & (ts < 40.0)
formel = (WP**2 / nu) * np.exp(-GAMMA * ts[maske] / 2) * np.sin(nu * ts[maske])
abw = np.abs(chi_t[maske] - formel).max() / np.abs(chi_t).max()
print(f"vor t=0:   |χ| höchstens {vorher:.1e} des Maximums")
print(f"nach t=0:  Abweichung von der Kap.-19-Formel höchstens {abw:.1e}")
```

```
vor t=0:   |χ| höchstens 3.5e-07 des Maximums
nach t=0:  Abweichung von der Kap.-19-Formel höchstens 3.3e-07
```

Beide Zahlen liegen bei 10^{-7} — das ist die Güte des endlichen Frequenzfensters, nicht der Physik. Das Gedächtnis des Lorentz-Glases ist exakt die gedämpfte Schwingung aus Kapitel 19, und vor dem Stoß ist es leer:

```
zoom = (ts > -15) & (ts < 50)
fig, ax = plt.subplots(figsize=(7.0, 3.6))
ax.plot(ts[zoom], chi_t[zoom], "C0-", lw=1.4, label="χ(t) aus iFFT")
ax.plot(ts[maske][::1500], formel[::1500], "ko", ms=4,
        label="Kap.-19-Formel (Stichproben)")
ax.axvline(0, color="k", lw=0.8)
ax.axhline(0, color="0.8", lw=0.6)
ax.annotate("hier wäre die Zukunft —\nund hier ist nichts",
            xy=(-8, 0.4), fontsize=9, ha="center")
ax.set_xlabel("Zeit t seit dem Feld-Blitz")
ax.set_ylabel("χ(t)")
ax.set_title("Die Materie antwortet mit Gedächtnis — aber nie im Voraus")
ax.legend(fontsize=9)
plt.show()
```

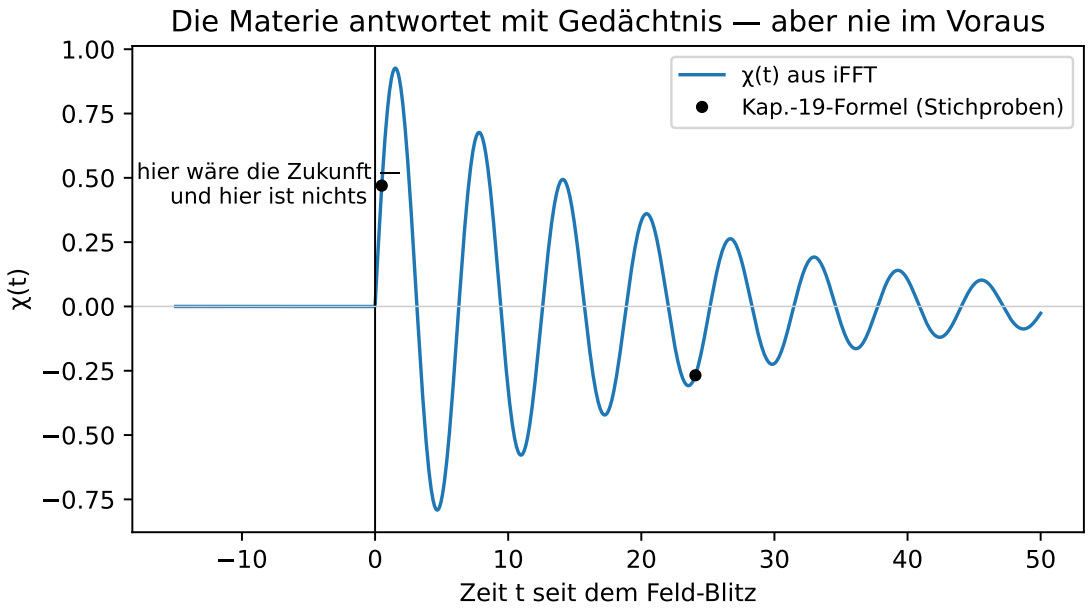


Abbildung 29.1: Das Gedächtnis des Lorentz-Glases: die Antwortfunktion χ(t) aus der inversen FFT von χ(ω) (blau) und die Kapitel-19-Ausschwingformel (ω_p²/ν)·e^(−γt/2)·sin(νt) als schwarze Punkte — sie liegen exakt aufeinander. Links der senkrechten Linie (t < 0, vor dem Feld-Blitz) ist die Antwort null auf sieben Stellen: Das ist die Kausalität, aus der dieses Kapitel alles Weitere zieht.

29.2 Die halbe Funktion bestimmt die ganze

Jetzt kommt der zentrale Gedanke des Kapitels — und er braucht keine höhere Mathematik, nur einen Trick, der dir in Kapitel 10 schon einmal begegnet ist: Jede Funktion lässt sich in einen **geraden** und einen **ungeraden** Anteil zerlegen,

$$\chi(t) = \underbrace{\frac{\chi(t)+\chi(-t)}{2}}_{\chi_g(t)\ \text{(gerade)}} + \underbrace{\frac{\chi(t)-\chi(-t)}{2}}_{\chi_u(t)\ \text{(ungerade)}}.$$

Für eine *beliebige* Funktion sind χ_g und χ_u unabhängig voneinander. Für eine **kausale** Funktion nicht! Denn für $t < 0$ ist $\chi(t) = 0$, also $\chi_g(t) =$

$\chi(-t)/2$ und $\chi_u(t) = -\chi(-t)/2$ — die beiden Anteile sind dort exakt entgegengesetzt. Und für $t > 0$ sind sie exakt gleich (beide $\chi(t)/2$). Kompakt:

$$\chi_u(t) = \operatorname{sgn}(t) \cdot \chi_g(t).$$

Wer den geraden Anteil kennt, kennt also automatisch den ungeraden — **die halbe Funktion bestimmt die ganze**. Das prüfen wir sofort am Lorentz-Gedächtnis von oben nach:

```
# symmetrisches Zeitgitter um 0, Werte vom FFT-Gitter uebernommen
t_sym = np.linspace(-40, 40, 8001)
# np.interp: lineare Interpolation — liest chi_t an den neuen Stellen ab
c_sym = np.interp(t_sym, ts, chi_t)
gerade = (c_sym + c_sym[::-1]) / 2      # [::-1] spiegelt: χ(−t)
ungerade = (c_sym - c_sym[::-1]) / 2
probe = np.abs(ungerade - np.sign(t_sym) * gerade).max() / np.abs(c_sym).max()
print(f"χ_u = sgn(t)·χ_g erfüllt bis auf {probe:.1e}")
```

```
χ_u = sgn(t)·χ_g erfüllt bis auf 5.0e-04
```

Warum ist das der Schlüssel? Erinnere dich an Kapitel 10: Bei der Fourier-Transformation gehört der **gerade** Anteil einer reellen Funktion zum **Realteil** des Spektrums, der **ungerade** zum **Imaginärteil**. Die Kette schließt sich:

$$\operatorname{Re}\chi(\omega) \overset{\text{FT}}{\longleftrightarrow} \chi_g(t) \overset{\cdot\operatorname{sgn}(t)}{\longrightarrow} \chi_u(t) \overset{\text{FT}}{\longleftrightarrow} \mathrm{i}\operatorname{Im}\chi(\omega).$$

Aus $\operatorname{Im}\chi$ lässt sich χ_u bauen, daraus per sgn-Multiplikation χ_g, daraus $\operatorname{Re}\chi$ — **Real- und Imaginärteil einer kausalen Antwort sind zwei Lesarten derselben Information.** Führt man die drei Pfeile formal aus (die sgn-Multiplikation wird im Frequenzraum zu einer Faltung mit $1/\omega$ — die Rechnung steht im Kleingedruckten), entsteht die berühmte **Kramers-Kronig-Relation**:

$$\operatorname{Re}\chi(\omega) = \frac{2}{\pi}\,\mathcal{P}\!\int_0^\infty \frac{\omega' \operatorname{Im}\chi(\omega')}{\omega'^2 - \omega^2}\,\mathrm{d}\omega'.$$

Lies sie ruhig zweimal: Links steht die *Brechung* bei einer Frequenz, rechts ein Integral über die *Absorption* bei **allen** Frequenzen. Das $\mathcal{P}$ („Hauptwert") warnt vor der Stelle $\omega' = \omega$, an der der Nenner null wird — dazu gleich. Die

Relation funktioniert auch umgekehrt (Re rein, Im raus, mit einem Minuszeichen); wir arbeiten mit der Richtung, die Messpraxis hat: Absorption ist meist leichter zu messen als Brechung.

29.3 Das Hauptwert-Integral zähmen

Ein Integral, dessen Integrand bei $\omega' = \omega$ durch null teilt, sieht nach Ärger aus. Der **Hauptwert** rettet es: Man nähert sich der heiklen Stelle von beiden Seiten gleich schnell, und weil der Integrand links und rechts davon entgegengesetzte Vorzeichen hat, heben sich die beiden Unendlichkeiten weg. Numerisch gibt es dafür einen Trick, der so einfach ist, dass er fast verdächtig wirkt — der **Subtraktions-Trick**. Wir schreiben den Zähler als $h(\omega') = \omega' \operatorname{Im}\chi(\omega')$ und ziehen seinen Wert an der heiklen Stelle ab:

$$\mathcal{P}\!\int \frac{h(\omega')}{\omega'^2-\omega^2}\,\mathrm{d}\omega' = \int \frac{h(\omega')-h(\omega)}{\omega'^2-\omega^2}\,\mathrm{d}\omega' \;+\; h(\omega)\,\mathcal{P}\!\int \frac{\mathrm{d}\omega'}{\omega'^2-\omega^2}.$$

Der erste Integrand ist **nirgends mehr singulär**: An der Stelle $\omega' = \omega$ wird er zu „$0/0$" mit dem harmlosen Grenzwert $h'(\omega)/(2\omega)$ — eine normale Zahl, die `np.trapezoid` problemlos verdaut. Und das zweite Integral enthält gar kein h mehr; es ist eine Schulbuch-Stammfunktion (ln-Ausdruck), die wir von Hand hinschreiben. **Versuchsaufbau:** Frage — rekonstruiert das KK-Integral aus dem Imaginärteil des Lorentz-χ den Realteil? Bühne: Integrationsgitter von fast 0 bis $W = 400$ mit 800 000 Punkten; ausgewertet wird an 60 Teststellen zwischen $\omega = 0{,}05$ und 3 (bewusst um einen halben Gitterabstand versetzt, damit keine Teststelle exakt auf einem Integrationspunkt liegt — sonst wird der „glatte" Quotient dort numerisch doch wieder wacklig). Messgröße: die größte Abweichung vom exakten Re χ. Erfolgskriterium: nahe Maschinengenauigkeit — die Relation ist ja exakt.

```
def kk_re(w_aus, ws, im_funktion):
    """Re χ an den Stellen w_aus aus Im χ (Kramers-Kronig).

    ws: Integrationsgitter (beginnt nahe 0); im_funktion(ω) liefert
    Im χ. Der Wert an der heiklen Stelle wird exakt ausgewertet."""
    h = ws * im_funktion(ws)              # Zaehler des Integranden
    re = np.empty(len(w_aus))
    a, b = ws[0], ws[-1]
    for k, w in enumerate(w_aus):
        hw = w * im_funktion(w)           # h an der heiklen Stelle
        glatt = (h - hw) / (ws**2 - w**2)     # nirgends mehr singulaer
```

```
        # Hauptwert des Rest-Integrals ∫ dω′/(ω′²−ω²) von a bis b,
        # von Hand integriert (Partialbruch → Logarithmus):
        rest = hw / (2 * w) * np.log(abs((b - w) * (a + w)
                                         / ((b + w) * (a - w))))
        re[k] = (2 / np.pi) * (np.trapezoid(glatt, ws) + rest)
    return re


ws = np.linspace(1e-6, 400.0, 800_000)
w_test = np.linspace(0.05, 3.0, 60) + (ws[1] - ws[0]) / 2
re_kk = kk_re(w_test, ws, lambda w: chi_lorentz(w).imag)
fehler = np.abs(re_kk - chi_lorentz(w_test).real).max()
print(f"größter Fehler der Rekonstruktion: {fehler:.1e}")
```

```
größter Fehler der Rekonstruktion: 3.3e-10
```

Zehn Stellen. Aus der Absorptionskurve allein entsteht die komplette Brechungskurve — inklusive des anomalen Einbruchs an der Resonanz und der Flügel weit draußen:

```
w_fein = np.linspace(0.01, 3.0, 600)
fig, ax = plt.subplots(figsize=(7.0, 3.8))
ax.plot(w_fein, chi_lorentz(w_fein).real, "C0-", lw=1.4,
        label="Re χ (exakt)")
ax.plot(w_fein, chi_lorentz(w_fein).imag, "C3-", lw=1.4,
        label="Im χ (die Eingabe)")
ax.plot(w_test, re_kk, "ko", ms=3.5, label="Re χ aus KK(Im χ)")
ax.axvline(W0, color="0.75", lw=0.7)
ax.axhline(0, color="0.85", lw=0.6)
ax.set_xlabel("Frequenz ω/ω₀")
ax.set_ylabel("χ")
ax.set_title(f"Im rein, Re raus — Fehler {fehler:.0e}")
ax.legend(fontsize=9)
plt.show()
```

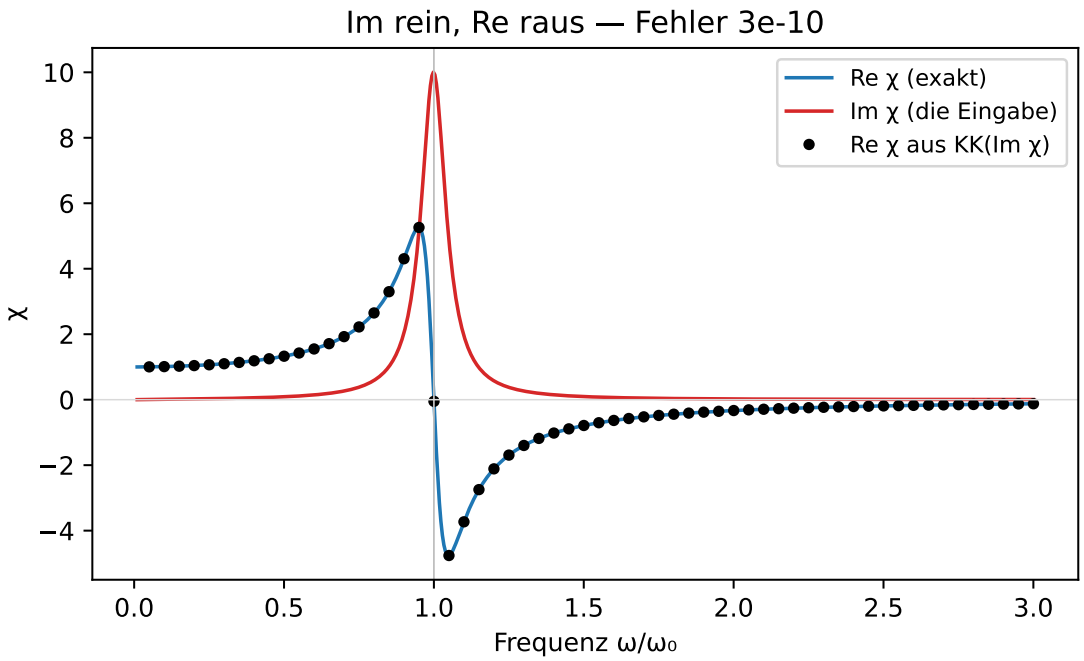


Abbildung 29.2: Kramers-Kronig bei der Arbeit: Aus dem Imaginärteil des Lorentz-χ (rot, die Absorptionslinie — das ist die EINZIGE Eingabe) rekonstruiert das Hauptwert-Integral den Realteil an 60 Teststellen (schwarze Punkte). Die blaue Linie ist der exakte Realteil: Die Punkte liegen mit einem größten Fehler von $3{\cdot}10^{-10}$ darauf — Brechung und Absorption sind eine einzige Information.

29.4 Die Kausalitäts-Probe

Die Relation kann mehr als rekonstruieren: Sie kann **Materialien verbieten**. Wenn Re und Im zusammengehören, dann kann man nicht beliebige Wunschpaare hinschreiben — und das prüfen wir jetzt an einem Wunschtraum der Optik.

Vorhersage (PRIMM)

Wir nehmen das Lorentz-χ und **löschen den Imaginärteil**: $\chi_{\text{Spiel}}(\omega) = \operatorname{Re}\chi(\omega)$, also volle Dispersion, aber null Absorption bei allen Frequenzen — ein perfekt durchsichtiges dispersives Glas. Dann rechnen wir wie in Abbildung 29.1 die Antwortfunktion $\chi(t)$ aus. Was kommt heraus

> — (a) dieselbe Ausschwingkurve wie vorher, (b) eine etwas verformte, aber immer noch kausale Kurve, oder (c) etwas grundsätzlich Kaputtes?

Die Maschine dafür haben wir schon — ein Aufruf genügt:

```
ts2, spiel_t = antwortfunktion(chi_lorentz(w_fft).real + 0j)
spiel_t = spiel_t.real
verletzung = np.abs(spiel_t[ts2 < -0.5]).max() / np.abs(spiel_t).max()
print(f"Spiel-χ: |χ(t<0)| erreicht {verletzung:.2f} des Maximums")
```

```
Spiel-χ: |χ(t<0)| erreicht 1.00 des Maximums
```

Antwort (c), und zwar maximal kaputt: Die Antwortfunktion wird **exakt gerade** — das halbe Echo kommt **vor dem Knall**. Kein Wunder: Ein reiner Realteil ist eine gerade Funktion von ω, und die inverse FFT einer geraden reellen Funktion ist zwangsläufig gerade in t. Unser Wunschglas müsste anfangen zu schwingen, *bevor* der Feld-Blitz eintrifft:

```
zoom = (ts > -40) & (ts < 40)
fig, ax = plt.subplots(figsize=(7.0, 3.8))
ax.plot(ts[zoom], chi_t[zoom], "C0-", lw=1.2,
        label="Lorentz-χ(t): kausal")
ax.plot(ts2[zoom], spiel_t[zoom], "C3-", lw=1.2, alpha=0.85,
        label="Spiel-χ(t): Im = 0 erzwungen")
ax.axvline(0, color="k", lw=0.8)
ax.set_xlabel("Zeit t seit dem Feld-Blitz")
ax.set_ylabel("χ(t)")
ax.set_title("Absorption gelöscht — das halbe Echo kommt vor dem Knall")
ax.legend(fontsize=9, loc="upper left")
plt.show()
```

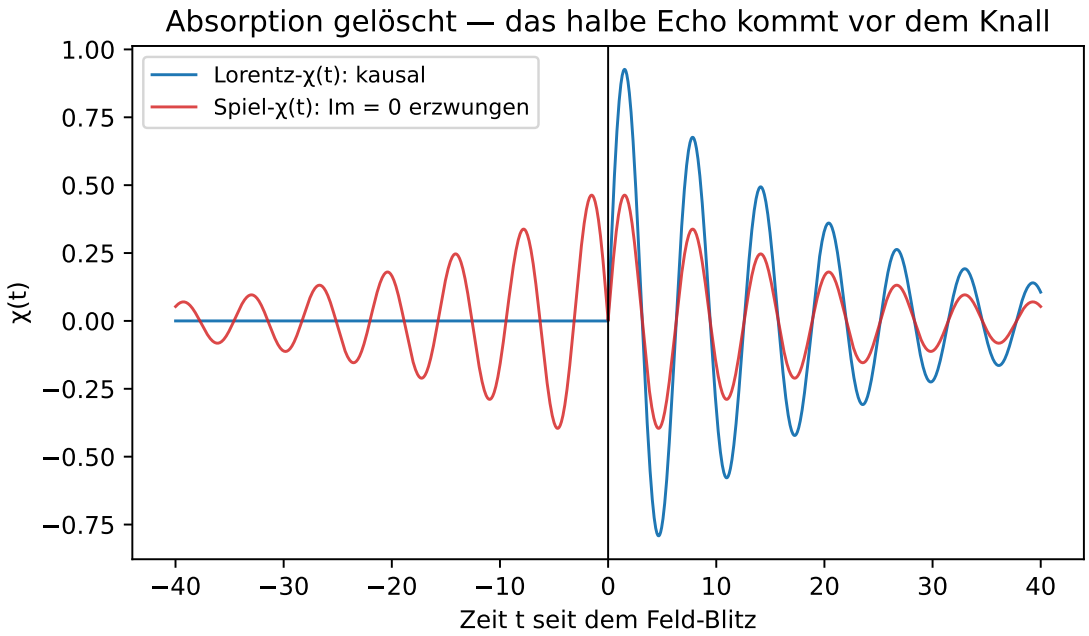


Abbildung 29.3: Die Kausalitäts-Probe: das ehrliche Lorentz-χ(t) (blau) beginnt exakt bei t = 0. Das Spiel-Material mit gelöschter Absorption (rot) hat dieselbe Dispersion — aber seine Antwortfunktion ist spiegelsymmetrisch: Sie schwingt schon VOR dem Blitz (links der schwarzen Linie), mit voller halber Amplitude. Ein dispersives Material ohne jede Absorption müsste hellsehen.

Naheliegende Vermutung: „Ein perfekt durchsichtiges Glas mit n ≠ 1 bei allen Frequenzen ist nur eine Frage der Materialkunst“

Warum sie naheliegt: Fensterglas wirkt im Sichtbaren verlustfrei und hat trotzdem $n = 1{,}5$ — warum sollte man das nicht auf alle Frequenzen ausdehnen können, wenn man nur das richtige Material fände?
Was stattdessen stimmt: Abbildung 29.3 ist der Unmöglichkeitsbeweis. $n \neq 1$ heißt $\chi \neq 0$, und ein χ, dessen Imaginärteil *überall* null ist, hat eine zeitlich gerade Antwortfunktion — es müsste antworten, bevor gefragt wird. Echtes Glas trickst nicht, es *verlagert*: Seine Absorption wohnt im Ultravioletten und Infraroten (Übung 29.1 rechnet nach, dass die BK7-Resonanzen bei 77 nm, 141 nm und 10 µm sitzen — alle außerhalb des Sichtbaren). Das $n = 1{,}5$ im Sichtbaren ist das Kramers-Kronig-Echo dieser unsichtbaren Absorptionslinien. Durchsichtig *in einem Fenster* ist erlaubt — durchsichtig *überall* nicht. Und nebenbei ist

damit auch der Kapitel-28-Kredit eingelöst: $\chi \neq 0$ erzwingt Absorption irgendwo, Absorption erzwingt über KK Frequenzabhängigkeit — **negative Konstanten ohne Dispersion kann es nicht geben.**

29.5 Die Absorption ist global festgenagelt

Aus der Verkettung folgt noch eine Zugabe, die wir gleich für Gold brauchen werden. Weit oberhalb *aller* Resonanzen können die Elektronen dem Feld nur noch als freie Teilchen folgen; jedes Material wird dort zum Drude-Plasma mit $\chi \to -\omega_p^2/\omega^2$ (Kapitel 16). Setzt man dieses Verhalten in die KK-Relation ein, fällt für $\omega \to \infty$ heraus:

$$\int_0^\infty \omega' \,\mathrm{Im}\,\chi(\omega')\,\mathrm{d}\omega' = \frac{\pi}{2}\,\omega_p^2.$$

Das ist die **f-Summenregel**: Die Gesamtfläche unter der (gewichteten) Absorptionskurve ist durch die Elektronendichte festgelegt — egal, wie sich die Absorption auf Linien verteilt. Ein Material kann seine Absorption **verschieben** (in den UV-Bereich, wie Glas), aber nicht **löschen**. Die Probe am Lorentz-Modell:

```
integral = np.trapezoid(ws * chi_lorentz(ws).imag, ws)
# was jenseits von W = 400 fehlt, von Hand: Im χ → γω_p²/ω´³,
# also ∫_W^∞ ω´·Im χ dω´ ≈ γω_p²/W
auslaeufer = GAMMA * WP**2 / 400.0
print(f"Integral bis 400:   {integral:.6f}")
print(f"+ Ausläufer-Restwert {auslaeufer:.6f}  →  "
      f"{integral + auslaeufer:.6f}")
print(f"Sollwert π/2·ω_p² =  {np.pi / 2 * WP**2:.6f}")
```

```
Integral bis 400:   1.570546
+ Ausläufer-Restwert 0.000250  →  1.570796
Sollwert π/2·ω_p² =  1.570796
```

Hübsches Detail: Das numerische Integral verfehlt $\pi/2$ um exakt den von Hand gerechneten Ausläufer $\gamma\omega_p^2/W$ — die Abweichung ist keine Ungenauigkeit, sondern der ehrlich fehlende Schwanz. Merke dir diesen Schwanz; beim Gold gleich entscheidet sein Gegenstück am *unteren* Rand über Erfolg und Desaster.

29.6 Realwelt-Probe: Meeps Goldkarte

Genug Modell — jetzt echte Materialdaten. `meep.materials.Au` ist Meeps Gold: ein Drude-Term plus fünf Lorentz-Linien, angepasst an Messdaten (gültig zwischen $0{,}2$ und $12{,}4$ µm Wellenlänge; die Suszeptibilitäten kennen wir seit Kapitel 16). Wenn unser Satz stimmt, muss diese Karte Kramers-Kronig erfüllen — sie ist ja aus kausalen Bausteinen zusammengesetzt. Aber sie gibt uns auch Gelegenheit für eine ehrliche Lektion: Was passiert, wenn man — wie jedes echte Labor — die Absorption nur in einem **Messfenster** kennt? **Versuchsaufbau:** Frage — rekonstruiert das KK-Integral $\mathrm{Re}\,\varepsilon$ von Gold im Sichtbaren (a) aus dem fast vollständigen Spektrum (f' von nahe 0 bis 1000, in Meep-Einheiten $f = 1/\lambda$ in µm) und (b) nur aus dem Fit-Fenster $0{,}08$ bis 5? Messgröße: die größte Abweichung von der Karte an elf Punkten zwischen $\lambda = 400$ und 800 nm. Erwartung: (a) sehr gut, (b) deutlich schlechter — wie deutlich, ist die Lektion.

```
import meep as mp
from meep.materials import Au


def eps_gold(f):
    """ε(f) aus Meeps Au-Karte, als analytische Drude-Lorentz-Summe.

    (Au.epsilon() selbst verweigert Frequenzen außerhalb des
    Fit-Fensters — die Summenformel ist überall definiert.)"""
    # np.asarray: macht aus Skalar oder Liste ein Array, ohne Kopie,
    # wenn es schon eines ist — so schluckt die Funktion beides
    f = np.asarray(f, float)
    eps = np.full(f.shape, complex(Au.epsilon_diag.x))
    for s in Au.E_susceptibilities:
        f0, g, sig = s.frequency, s.gamma, s.sigma_diag.x
        if isinstance(s, mp.DrudeSusceptibility):
            eps += -sig * f0**2 / (f**2 + 1j * f * g)
        else:
            eps += sig * f0**2 / (f0**2 - f**2 - 1j * f * g)
    return eps


f_seh = np.linspace(1.25, 2.5, 11)         # sichtbar: λ = 800 … 400 nm
eps_karte = eps_gold(f_seh)
print(f"Gold bei λ = 800 nm: ε = {eps_karte[0]:.2f}")

# (a) fast das ganze Spektrum — nahe 0 logarithmisch verdichtet!
```

```
# np.unique: sortiert und entfernt Doppelte — naehte der Teilgitter
fs_breit = np.unique(np.concatenate([
    np.geomspace(1e-6, 0.5, 100_000),
    np.linspace(0.5, 20.0, 400_000),
    np.geomspace(20.0, 1000.0, 60_000)]))
re_breit = kk_re(f_seh, fs_breit, lambda f: eps_gold(f).imag) + 1.0
print(f"(a) KK mit f′ bis 1000:    größte Abw. "
      f"{np.abs(re_breit - eps_karte.real).max():.1e}")

# (b) nur das Fit-Fenster
fs_fenster = np.linspace(0.0807, 5.0, 200_000)
re_fenster = kk_re(f_seh, fs_fenster, lambda f: eps_gold(f).imag) + 1.0
print(f"(b) KK nur Fenster 0,08–5: größte Abw. "
      f"{np.abs(re_fenster - eps_karte.real).max():.1f}  (!)")
```

```
Using MPI version 4.1, 1 processes
Gold bei λ = 800 nm: ε = -20.28+2.07j
(a) KK mit f′ bis 1000:    größte Abw. 3.8e-04
(b) KK nur Fenster 0,08–5: größte Abw. 15.7  (!)
```

Mit dem ganzen Spektrum trifft das Integral die Karte auf vier Nachkommastellen — **reale Materialdaten erfüllen Kramers-Kronig**, so wie es der Satz verlangt. Mit dem Fenster allein liegt es um **15** daneben — bei einem Sollwert von -20 ist das kein Messfehler mehr, sondern ein anderes Material:

```
fig, ax = plt.subplots(figsize=(7.0, 3.8))
ax.plot(f_seh, eps_karte.real, "C0-", lw=1.6, label="Re ε (Goldkarte)")
ax.plot(f_seh, re_breit, "ko", ms=4.5, label="KK, f′ bis 1000")
ax.plot(f_seh, re_fenster, "C3s", ms=4.5, label="KK, nur Fenster 0,08–5")
ax.axhline(0, color="0.85", lw=0.6)
ax.set_xlabel("f = 1/λ [1/µm]   (sichtbares Band)")
ax.set_ylabel("Re ε")
ax.set_title("Reale Daten erfüllen KK — aber nur das ganze Konto stimmt")
ax.legend(fontsize=9)
plt.show()
```

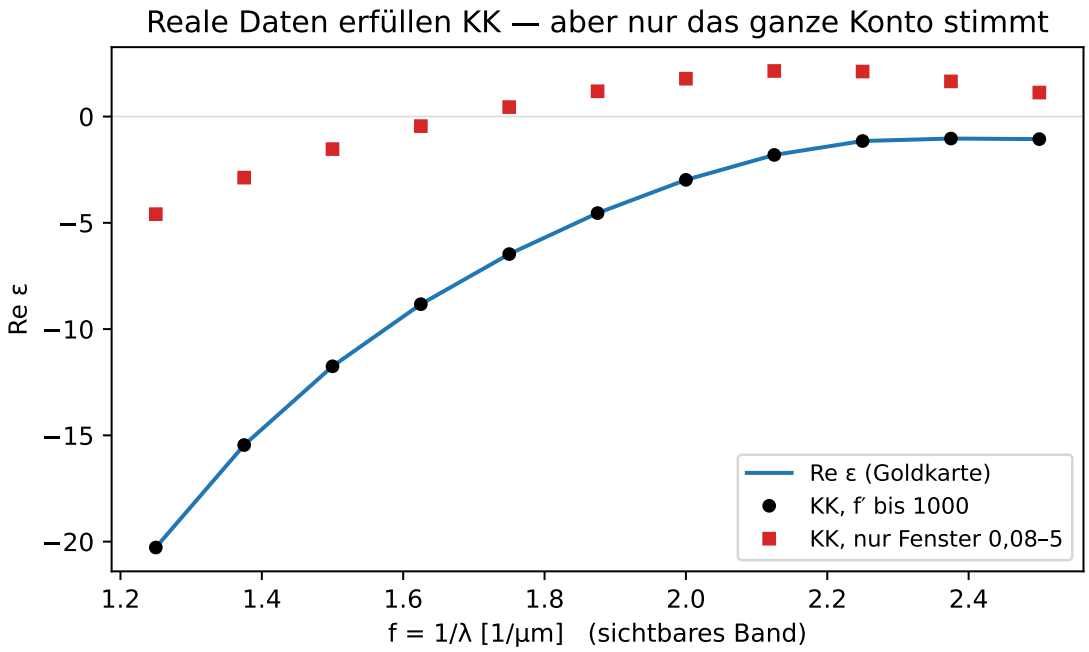


Abbildung 29.4: Gold im Sichtbaren: Re ε aus Meeps Materialkarte (blaue Linie) gegen die KK-Rekonstruktion aus Im ε. Mit Integrationsbereich bis f′ = 1000 (schwarze Punkte) liegt die Rekonstruktion exakt auf der Karte (Abweichung $4{\cdot}10^{-4}$). Nur mit dem Messfenster 0,08–5 (rote Quadrate) fehlt vor allem der Drude-Anteil unterhalb des Fensters — und die Rekonstruktion verliert über 15 Einheiten. Das KK-Integral ist ein Konto: Es stimmt nur, wenn alle Posten gebucht sind.

Warum ist das Fenster so katastrophal, wo doch unser Lorentz-Integral oben bei $W = 400$ enden durfte, ohne dass auch nur eine der zehn Stellen verloren ging? Schau, *wo* das Fenster schneidet.

Werkstattkasten: oben abschneiden verzeiht — unten nicht

Der Integrand der KK-Relation ist $h(\omega') = \omega' \operatorname{Im}\chi(\omega')$. **Am oberen Ende** fällt h bei jedem realen Material wie $1/\omega'^2$; der abgeschnittene Schwanz kostet einen Fehler $\propto 1/W^3$ (Übung 29.4 misst genau dieses Gesetz) — darum durften wir beim Lorentz-Modell sorglos bei $W = 400$ aufhören. **Am unteren Ende** ist Gold aber ein Leiter: Der Drude-Term hat $\operatorname{Im}\chi \propto 1/\omega'$, also läuft h auf das *Plateau* $h(0) = \sigma f_0^2/\gamma \approx 1241$ — ein riesiger Buckel der Breite $\gamma_D \approx 0{,}04$, und das Messfenster ab $0{,}08$ schneidet ihn komplett weg. Dieses Loch kostet linear ($\propto a$ bei

unterer Grenze a) mit dem Riesen-Vorfaktor ω_p^2/γ. Praktische Folgen: (1) Beim numerischen KK-Integral für Leiter das Gitter bis fast 0 führen und dort verdichten (unser `geomspace`-Stück — ein erster Versuch mit Untergrenze 10^{-3} lag um $0{,}4$ daneben); (2) im Labor, wo man unterhalb des Messfensters schlicht keine Daten *hat*, rettet man sich mit verankerten Varianten der Relation („subtraktive KK", Kleingedrucktes).

29.7 Die Front läuft nie schneller als c

Bleibt die älteste Schuld: Kapitel 16 hat im Drude-Medium $v_p = 1{,}67\,c$ **gemessen** — Wellenberge, schneller als das Licht. Die beruhigende Auskunft, ein *Signal* sei trotzdem nie schneller als c, blieb unbewiesen. Der Beweis-Gedanke gehört in dieses Kapitel, denn er ist derselbe wie bisher: Das Medium ist ein Haufen Oszillatoren mit Gedächtnis. Wenn die allererste Feldfront eintrifft, haben die Oszillatoren noch nichts gespürt — sie *können* noch nicht mitspielen. Die Front sieht das Medium schlicht nicht; sie läuft mit Vakuum-c, und erst das Nachfolgende wird gebremst, gebrochen, verformt.

Das messen wir jetzt — und zwar nicht in Meep, sondern im selbstgebauten 1D-FDTD aus Kapitel 5/6, erweitert um genau eine Idee: **An jeden Gitterpunkt im Medium hängen wir einen Kapitel-19-Oszillator.** Das ist keine Metapher, sondern der Algorithmus (Meep macht es interessanterweise genauso; das Verfahren heißt ADE, *auxiliary differential equation*). Drei Bausteine:

1. **Maxwell läuft über D:** Statt E direkt updaten wir $D = E + P$ mit der gewohnten Nachbar-Differenz-Zeile aus Kapitel 5.
2. **Der Oszillator liefert P:** $\ddot{P} + \gamma\dot{P} + \omega_0^2 P = \omega_p^2 E$, diskretisiert mit zentrierten Differenzen — exakt das Kapitel-19-Schema. Aufgelöst nach dem neuen Wert: $P^{n+1} = k_a P^n + k_b P^{n-1} + k_c E^n$ mit den drei Koeffizienten aus der Zelle unten (Zähler $2 - \omega_0^2 \Delta t^2$ bzw. $-(1 - \gamma\Delta t/2)$ und $\omega_p^2 \Delta t^2$, gemeinsamer Nenner $1 + \gamma\Delta t/2$ — dieselbe Bauform wie beim gedämpften Schwingkreis).
3. **Der Material-Schritt:** $E^{n+1} = D^{n+1} - P^{n+1}$. Im Vakuum ist $P = 0$ und alles kollabiert auf das alte Kapitel-5-Update.

Versuchsaufbau: Die Bühne ist eine Zeile aus 3200 Zellen ($\Delta x = 1$, $c = 1$, $S = \Delta t = 0{,}5$); bei $x = 20$ sitzt eine weiche (additive) CW-Quelle, die bei $t = 0$ **abrupt** einsetzt — der Einschalt-Knick ist hier kein Unfall wie in Kapitel 22, sondern das Messobjekt: Er *ist* die Front. Ab $x = 400$ beginnt das Medium. Drei Messgrößen: (1) der Brechungsindex aus der Wellenlänge

im eingeschwungenen Medium (dominante Raumfrequenz per FFT — die Nulldurchgangs-Methode stolpert hier über Restwelligkeits-Doppelnullen); (2) die Gruppengeschwindigkeit aus der Schwerpunkt-Laufzeit eines schmalbandigen Gauß-Pulses zwischen zwei Detektoren; (3) die **Front-Verzögerung** — dazu gleich mehr. Erfolgskriterium: n und v_g treffen die Kapitel-16-Formeln; die Front-Verzögerung ist null.

```
DT = 0.5            # S = 0,5 – mit S = 1 ist das ADE-Schema instabil
NX = 3200
X_Q, X_MED = 20, 400
F_TR = 0.05         # Traegerfrequenz: λ = 20 Zellen
W_TR = 2 * np.pi * F_TR


def fdtd_lauf(w0, gamma, wp2, nt, medium=True, quelle=None):
    """1D-FDTD mit Lorentz-Medium ab X_MED; liefert E(t, x)."""
    e = np.zeros(NX); b = np.zeros(NX - 1); d = np.zeros(NX)
    p = np.zeros(NX); p_alt = np.zeros(NX)
    med = np.zeros(NX, bool)
    if medium:
        med[X_MED:] = True
    nenner = 1 + gamma * DT / 2          # Kap.-19-Koeffizienten
    ka = (2 - w0**2 * DT**2) / nenner
    kb = -(1 - gamma * DT / 2) / nenner
    kc = wp2 * DT**2 / nenner
    if quelle is None:
        quelle = lambda t: np.sin(W_TR * t)
    serie = np.empty((nt, NX))
    for n in range(nt):
        d[1:-1] += DT * np.diff(b)       # Maxwell: D-Update (Kap. 5)
        p_neu = np.where(med, ka * p + kb * p_alt + kc * e, 0.0)
        p_alt, p = p, p_neu              # Oszillator-Update (Kap. 19)
        e = d - p                        # Material-Schritt: E = D – P
        e[X_Q] += quelle(n * DT)         # weiche Quelle, abrupt ab t = 0
        e[0] = e[-1] = 0.0
        b += DT * np.diff(e)             # Maxwell: B-Update
        serie[n] = e
    return serie


def k_dominant(profil):
    """dominante Raumfrequenz: FFT + Hanning + Zero-Padding + Parabel."""
    s = (profil - profil.mean()) * np.hanning(len(profil))
    sp = np.abs(np.fft.rfft(s, n=8 * len(profil)))
```

```
    fr = np.fft.rfftfreq(8 * len(profil), d=1.0)
    i = np.argmax(sp)
    a, m, c = sp[i - 1], sp[i], sp[i + 1]
    return fr[i] + 0.5 * (a - c) / (a - 2 * m + c) * (fr[1] - fr[0])


def schwerpunkt_zeit(signal, nt):
    s = signal**2
    return (np.arange(nt) * DT * s).sum() / s.sum()
```

Zuerst das Lorentz-Glas: Resonanz bei $f_0 = 0{,}15$, Stärke so gewählt, dass $n(f_{\mathrm{Tr}}) = 1{,}5$ — unser Standardglas, aber diesmal ehrlich dispersiv:

```
W0_G = 2 * np.pi * 0.15
WP2_G = 1.25 * (W0_G**2 - W_TR**2)       # ⇒ n² = 1 + … = 2,25
cw = fdtd_lauf(W0_G, 0.0, WP2_G, 7000)
n_glas = k_dominant(cw[-1, 500:1100]) / F_TR
print(f"Glas: n = {n_glas:.3f}  (Soll 1,500; Rest ist Gitterdispersion)")

t0, tau = 1300.0, 400.0                  # schmalbandiger Gauß-Puls
puls = lambda t: np.sin(W_TR * t) * np.exp(-((t - t0) / tau) ** 2)
pm = fdtd_lauf(W0_G, 0.0, WP2_G, 10000, quelle=puls)
v_g = 800 / (schwerpunkt_zeit(pm[:, 1600], 10000)
             - schwerpunkt_zeit(pm[:, 800], 10000))
dn_dw = WP2_G * W_TR / ((W0_G**2 - W_TR**2)**2 * 1.5)
print(f"      v_g = {v_g:.3f} c  (Soll 1/n_g = {1/(1.5 + W_TR*dn_dw):.3f} c)")
del pm                                    # Speicher-Hygiene (256 MB)
```

```
Glas: n = 1.512  (Soll 1,500; Rest ist Gitterdispersion)
      v_g = 0.608 c  (Soll 1/n_g = 0.623 c)
```

Index und Gruppentempo treffen ihre Theoriewerte auf Prozent-Niveau (den Rest teilen sich die Gitterdispersion aus Kapitel 8 und die Verformung des Pulses durch die Frequenzabhängigkeit von v_g selbst). Jetzt die Front. Die naive Messung — „wann überschreitet $|E|$ am Detektor eine Schwelle?" — hat einen Haken, der ehrlich benannt gehört: Unser Gitter mit $S = 0{,}5$ transportiert Information mit bis zu **einer Zelle pro Zeitschritt**, also $2c$; winzige numerische Vorläufer (bis $4 \cdot 10^{-3}$ der Amplitude) lecken über die c-Linie, ganz ohne Physik. Der saubere Ausweg ist ein **Differenz-Experiment**: Wir fahren denselben Lauf zweimal — einmal mit, einmal ohne Medium — und

fragen, **wann sich die beiden Läufe am Detektor zum ersten Mal unterscheiden**. Alles, was das Gitter an Artefakten produziert, steckt in beiden Läufen gleich und kürzt sich; übrig bleibt allein die Wirkung des Mediums. Kann das Medium die Front bremsen (oder schieben), dann trennen sich die Läufe später (oder früher) als die Vakuum-Front-Ankunft.

Vorhersage (PRIMM)

Jetzt das Drude-Medium aus Kapitel 16, Arbeitspunkt oberhalb der Plasmafrequenz: $n = 0{,}6$, die Berge laufen mit $v_p = 1{,}67\,c$. Der Differenz-Detektor steht 1000 Zellen tief im Medium. Trennen sich Medium-Lauf und Vakuum-Lauf dort (a) *früher* als die Vakuum-Front (das Medium „schiebt" — Information mit Überlicht), (b) genau *mit* der Vakuum-Front, oder (c) *später* (das Medium bremst auch die Front)? Und beim Glas aus dem letzten Lauf — gilt dort dieselbe Antwort?

```
def front_verzug(w0, wp2):
    """Differenz-Experiment: Wann trennt das Medium die Läufe?"""
    med = fdtd_lauf(w0, 0.0, wp2, 7000)
    vak = fdtd_lauf(w0, 0.0, wp2, 7000, medium=False)
    amax = np.abs(med).max()
    xd = 1400
    differenz = med[:, xd] - vak[:, xd]
    t_vak = np.argmax(np.abs(vak[:, xd]) > 1e-6 * amax)
    t_med = np.argmax(np.abs(differenz) > 1e-6 * amax)
    return (t_med - t_vak) * DT, med


verzug_glas, _ = front_verzug(W0_G, WP2_G)
print(f"Glas (n = 1,5):  Front-Verzug {verzug_glas:.1f} Zeiteinheiten")

WP2_D = 0.64 * W_TR**2                    # Drude: n² = 1 − ω_p²/ω² = 0,36
verzug_drude, cw_drude = front_verzug(0.0, WP2_D)
n_drude = k_dominant(cw_drude[-1, 500:1100]) / F_TR
pm = fdtd_lauf(0.0, 0.0, WP2_D, 10000, quelle=puls)
v_g_d = 800 / (schwerpunkt_zeit(pm[:, 1600], 10000)
               - schwerpunkt_zeit(pm[:, 800], 10000))
del pm
print(f"Drude:  n = {n_drude:.3f} → v_p = {1/n_drude:.2f} c;  "
      f"v_g = {v_g_d:.3f} c")
print(f"        Front-Verzug {verzug_drude:.1f} Zeiteinheiten")
```

```
Glas (n = 1,5):  Front-Verzug 0.0 Zeiteinheiten
```

```
Drude:  n = 0.599 → v_p = 1.67 c;  v_g = 0.609 c
        Front-Verzug 0.0 Zeiteinheiten
```

Antwort (b), in beiden Medien: **Front-Verzug null.** Im Glas braucht der Puls-*Schwerpunkt* gut 700 Zeiteinheiten länger als durchs Vakuum — die allererste Regung kommt keinen einzigen Zeitschritt später an. Und im Drude-Medium laufen die Berge mit $1{,}67\,c$, die Energie kriecht mit $0{,}6\,c$ — und die Front hält exakt Vakuum-Tempo. Drei Geschwindigkeiten in einem Lauf, und nur eine davon ist eine Verkehrsregel:

```
fig, ax = plt.subplots(figsize=(7.0, 4.4))
schritte = np.arange(0, 7000, 14)
xt = np.log10(np.abs(cw_drude[schritte, :2400]) + 1e-12)
ax.imshow(xt, origin="lower", aspect="auto", cmap="inferno",
          extent=[0, 2400, 0, 7000 * DT], vmin=-8, vmax=0)
tt = np.linspace(0, 7000 * DT, 50)
ax.plot(X_Q + tt, tt, "w--", lw=1.2, label="Front-Soll: x = x₀ + c·t")
tg = tt[tt > X_MED - X_Q]                # v_g-Linie erst ab dem Medium
ax.plot(X_MED + (tg - (X_MED - X_Q)) * 0.6, tg, "c-", lw=1.2,
        label="Energie: v_g = 0,6 c")
ax.axvline(X_MED, color="w", lw=0.6)
ax.set_xlim(0, 2400)
ax.set_ylim(0, 7000 * DT)
ax.set_xlabel("x [Zellen]   (Medium ab x = 400)")
ax.set_ylabel("Zeit t")
ax.set_title("Berge mit 1,67 c — die Front bleibt auf der c-Linie")
ax.legend(fontsize=9, loc="upper left")
plt.show()
```

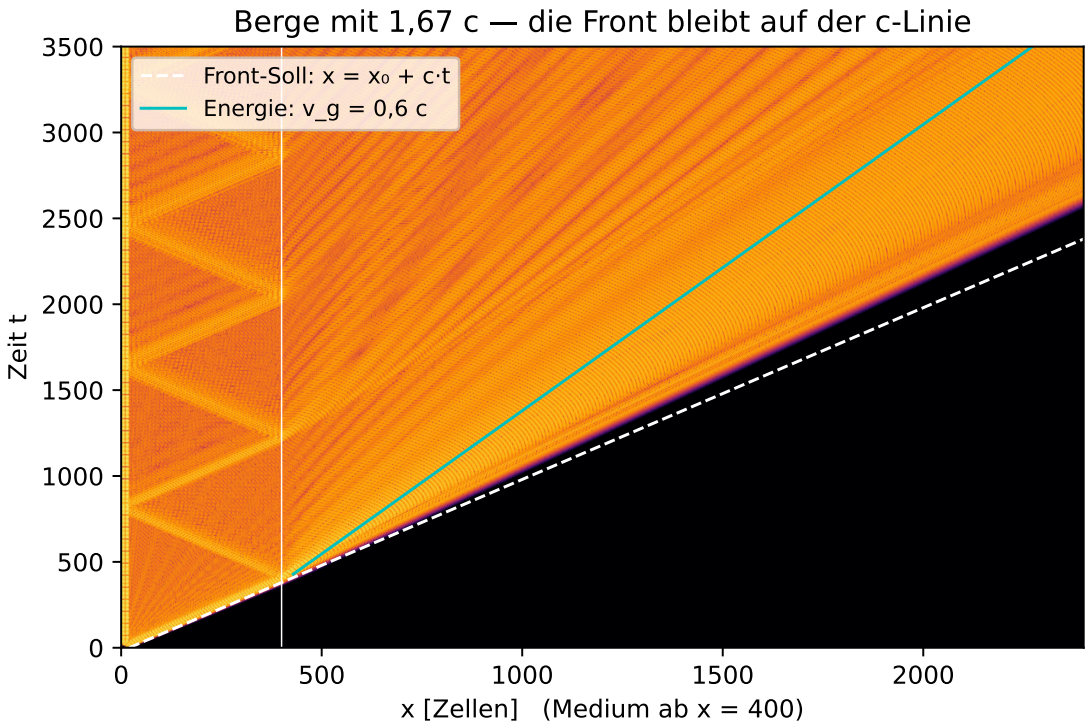


Abbildung 29.5: Das Front-Rennen im Drude-Medium als Weg-Zeit-Bild (Farbskala: log|E|; Medium rechts der weißen Linie bei x = 400). Die gestrichelte weiße Gerade ist die Vakuum-Lichtlinie $x = x_0 + ct$: Die Feldkante klebt exakt auf ihr, im Vakuum wie im Medium. Die cyanfarbene Linie ist die Gruppengeschwindigkeit 0,6 c — auf ihr wandert die Hauptenergie. Und die Wellenberge im Medium (die feinen Streifen) sind FLACHER geneigt als die c-Linie: Sie laufen mit 1,67 c, von der Front geboren und am v_g-Rücken sterbend. Schneller als Licht sind nur die Muster — nicht die Botschaft. (Das Dreiecksmuster links stammt von der Teilreflexion an der Mediumgrenze, die zwischen dem linken Zellrand und dem Medium hin- und herläuft.)

Das ist Kramers-Kronig im Zeitbild, mit denselben Worten wie am Kapitelanfang: Die Oszillatoren des Mediums haben von der Welle noch nichts gespürt, wenn die Front eintrifft — *für die Front ist jedes Medium Vakuum*. Erst hinter ihr baut sich (per Faltung, Abbildung 29.1) die Materialantwort auf, und alles Spätere wird verformt: Berge dürfen dabei beliebig schnell *erscheinen*, denn ein Wellenberg trägt keine Neuigkeit — er ist Wiederholung. Eine Botschaft steckt erst in einer *Änderung* des Musters, und jede Änderung ist eine kleine Front. Sommerfeld und Brillouin haben diese Analyse

1914 mit Papier und Bleistift durchgezogen, als Antwort auf genau unsere Kapitel-16-Provokation — ihre „Vorläufer" findest du im Kleingedruckten.

29.8 Das Kapitel-Programm

`programme/kap29/kap29_kramers_kronig.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die KK-Rekonstruktion am Lorentz-Modell (maschinen-nah), die iFFT-Kausalitätsprobe samt Kapitel-19-Formelvergleich und Spiel-ε, die sgn-Identität, die f-Summenregel mit Ausläufer-Korrektur, die Gold-Karte (breites Integral gegen Messfenster), beide Front-Experimente (Glas und Drude: n, v_g, Front-Verzug null) und den BK7-Wert aus Übung 29.1. Läuft in wenigen Sekunden — Meep wird darin nur einmal kurz nach seiner Gold-Karte gefragt, gerechnet wird alles selbst.

Merkkasten

- **Antwortfunktion:** $P(t)$ ist eine Faltung der Feld-Vergangenheit mit dem Gedächtnis $\chi(t)$; Kausalität heißt $\chi(t < 0) = 0$. Das Gedächtnis des Lorentz-Glases ist die gedämpfte Kapitel-19-Schwingung.
- **Kramers-Kronig:** Kausal ⇒ $\chi_u = \text{sgn}\cdot\chi_g$ ⇒ Re und Im von $\chi(\omega)$ bestimmen einander (Hauptwert-Integral). Dispersion ohne Absorption (irgendwo) ist unmöglich — ein „überall durchsichtiges" Glas mit $n \neq 1$ müsste vor der Ursache antworten.
- **Numerik:** Hauptwert per Subtraktions-Trick (Wert an der heiklen Stelle abziehen, Rest analytisch); Kausalitäts-Probe per iFFT. Das Integral braucht das ganze Spektrum: oben abschneiden kostet $\propto 1/W^3$, ein Loch unten (Drude-Buckel der Leiter!) kostet linear mit Vorfaktor ω_p^2/γ.
- **Summenregel:** $\int \omega \,\text{Im}\,\chi \,\mathrm{d}\omega = \pi\omega_p^2/2$ — Absorption lässt sich verschieben, nicht löschen.
- **Drei Tempi, eine Regel:** Berge laufen mit v_p (gern $> c$), Energie mit v_g, aber jede *Änderung* läuft als Front mit exakt c — die Oszillatoren des Mediums können die erste Nachricht nicht vorwegnehmen.
- **Werkstatt:** Kausalitäts-Aussagen im FDTD als **Differenz-Experiment** anlegen (Medium minus Vakuum) — das $S < 1$-Gitter trägt selbst numerische Vorläufer über die c-Linie, die sich im Paarvergleich kürzen.

Roter Faden

Mit diesem Kapitel schließt Teil VII — und er schließt sich als Bogen: Kapitel 26 baute aus Interferenz Spiegel, Kapitel 27 daraus Resonatoren, Kapitel 28 aus Resonatoren Materialien nach Maß — und Kapitel 29 liefert das Gesetz, das all diesen Antworten zugrunde liegt. Die **Kausalitäts-Wahl aus Kapitel 20** wird quantitativ: Das Gedächtnis $\chi(t)$ ist die **Kapitel-19-Schwingung**, seine Frequenz-Doppelgestalt verkettet die **Lorentz-Linien aus Kapitel 15/16** mit ihren eigenen Verlusten, der **Faltungssatz aus Kapitel 23** übersetzt zwischen beiden Bildern, und die **FFT aus Kapitel 10** ist das Messgerät. Die Kapitel-16-Schuld ($v_p > c$) und die Kapitel-28-Schuld (Dispersionspflicht) sind beglichen. Nach vorn: **Kapitel 30** verlässt den sicheren Boden — wir fragen, wo Maxwell *endet*: beim Photon, beim Doppelspalt mit einzelnen Lichtteilchen, bei $|E|^2$ als Wahrscheinlichkeit. Die Front-Regel dieses Kapitels übersteht übrigens auch diesen Umzug: Auch Quanten tragen keine Botschaft schneller als c.

Übungen

Ü 29.1 — Warum Fensterglas durchsichtig ist (Verstehen). Die Brechzahl von BK7-Glas wird in Datenblättern mit der **Sellmeier-Formel** angegeben: $n^2(\lambda) = 1 + \sum_k B_k \lambda^2/(\lambda^2 - C_k)$ mit $B = (1{,}0396;\ 0{,}2318;\ 1{,}0105)$ und $C = (0{,}0060;\ 0{,}0200;\ 103{,}56)$ µm^2. (a) Zeige durch Vergleich mit dem Lorentz-χ dieses Kapitels: Sellmeier ist eine Summe von Lorentz-Linien im Grenzfall $\gamma \to 0$ — und $\sqrt{C_k}$ sind die Resonanz-Wellenlängen. Wo wohnen die drei Resonanzen von BK7? (b) Berechne n bei der Natrium-d-Linie ($587{,}6$ nm). (c) Warum darf die Formel die Verluste weglassen, obwohl dieses Kapitel gezeigt hat, dass es ohne Absorption keine Dispersion gibt?

Musterlösung zu Ü 29.1

(a) Schreibe das Lorentz-χ auf Wellenlängen um ($\omega \propto 1/\lambda$) und setze $\gamma = 0$:

$$\chi = \frac{\omega_p^2}{\omega_0^2 - \omega^2} = \frac{\omega_p^2}{\omega_0^2} \cdot \frac{\lambda^2}{\lambda^2 - \lambda_0^2},$$

also genau ein Sellmeier-Term mit $B = \omega_p^2/\omega_0^2$ und $C = \lambda_0^2$. Die Resonanzen von BK7:

```
B = [1.03961212, 0.231792344, 1.01046945]
C = [0.00600069867, 0.0200179144, 103.560653]      # µm²
print("Resonanzen:", [f"{np.sqrt(c)*1000:.0f} nm" for c in C])

lam = np.geomspace(0.05, 20.0, 2000)               # µm
n2 = 1 + sum(b * lam**2 / (lam**2 - c) for b, c in zip(B, C))
lam_d = 0.5876
n_d = np.sqrt(1 + sum(b * lam_d**2 / (lam_d**2 - c)
                      for b, c in zip(B, C)))
print(f"n(587,6 nm) = {n_d:.4f}")

sichtbar = (lam > 0.38) & (lam < 0.78)
fig, ax = plt.subplots(figsize=(7.0, 3.4))
ax.semilogx(lam[n2 > 0], np.sqrt(n2[n2 > 0]), "C0-", lw=1.4)
ax.axvspan(0.38, 0.78, color="gold", alpha=0.2)
for c in C:
    ax.axvline(np.sqrt(c), color="C3", ls="--", lw=0.8)
ax.plot(lam_d, n_d, "ko", ms=5)
ax.set_ylim(1.3, 1.8)
ax.set_xlabel("Wellenlänge λ [µm]")
ax.set_ylabel("n")
ax.set_title("BK7: n im Sichtbaren = das Echo dreier ferner Resonanzen")
plt.show()
```

```
Resonanzen: ['77 nm', '141 nm', '10176 nm']
n(587,6 nm) = 1.5168
```

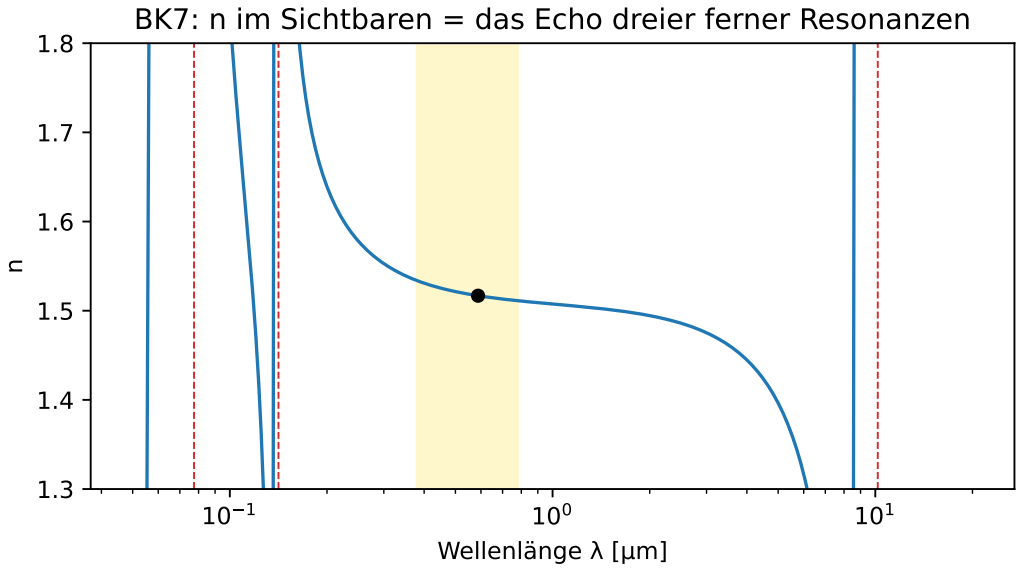


Abbildung 29.6: BK7 im Sichtbaren (gold hinterlegt): Die Sellmeier-Kurve fällt sanft von Blau nach Rot (normale Dispersion) und trifft bei 587,6 nm den Datenblattwert 1,5168 (Punkt). Die drei Resonanzen (rot gestrichelt bei 77 nm, 141 nm und 10,2 µm) liegen alle außerhalb — das Sichtbare lebt auf den ruhigen KK-Flügeln dieser unsichtbaren Absorber.

(b) $n(587{,}6\text{ nm}) = 1{,}5168$ — exakt der Datenblattwert. (c) Die Formel gilt *nur im Transparenzfenster*, weit weg von allen Resonanzen. Dort ist $\operatorname{Im}\chi$ tatsächlich praktisch null — erlaubt ist das, weil die Absorption anderswo wohnt (Kapitel-Refutation: verboten ist nur *überall* durchsichtig). Und Übung 29.2 zeigt: Weit weg von der Linie hängt $\operatorname{Re}\chi$ gar nicht von γ ab — darum darf Sellmeier die Breiten unterschlagen, solange die Lagen und Stärken stimmen.

Ü 29.2 — Zwei Linien, eine Regel (Verändern). Ergänze das Lorentz-χ des Kapitels um eine zweite Linie bei $\omega_0 = 2$ ($\gamma = 0{,}15$, Stärke $0{,}5$). (a) Funktioniert die KK-Rekonstruktion unverändert? Warum? (b) Sweepe die Breite der ersten Linie ($\gamma = 0{,}3/0{,}1/0{,}03$) bei fester Stärke ω_p^2: Wie ändert sich der Im-Gipfel, wie der Realteil weit weg von der Linie (etwa bei $\omega = 0{,}5$)?

Musterlösung zu Ü 29.2

```
def chi2(w):
    return (chi_lorentz(w)
            + 0.5 / (4.0 - w**2 - 1j * 0.15 * w))

re2 = kk_re(w_test, ws, lambda w: chi2(w).imag)
print(f"(a) zwei Linien: KK-Fehler {np.abs(re2 - chi2(w_test).real).max():.1e}")

for g in (0.3, 0.1, 0.03):
    c = 1.0 / (1.0 - 0.5**2 - 1j * g * 0.5)
    print(f"(b) γ = {g:4.2f}: Im-Gipfel ≈ {1/g:5.1f},  "
          f"Re χ(0,5) = {c.real:.4f}")
```

```
(a) zwei Linien: KK-Fehler 5.8e-10
(b) γ = 0.30: Im-Gipfel ≈   3.3,  Re χ(0,5) = 1.2821
(b) γ = 0.10: Im-Gipfel ≈  10.0,  Re χ(0,5) = 1.3274
(b) γ = 0.03: Im-Gipfel ≈  33.3,  Re χ(0,5) = 1.3328
```

(a) Der Fehler bleibt bei 10^{-10}: Sowohl die Fourier-Transformation als auch das KK-Integral sind **linear** — die Relation gilt für jede Summe kausaler Antworten, also für beliebig viele Linien (genau deshalb funktionierte sie auch für Meeps sechsteilige Gold-Karte).

(b) Der Im-Gipfel wächst wie $1/\gamma$ (die Fläche ist durch die Summenregel festgenagelt: schmaler = höher), aber $\mathrm{Re}\,\chi(0{,}5)$ ändert sich nur um wenige Prozent — weit weg von der Linie zählt allein $\omega_p^2/(\omega_0^2 - \omega^2)$, die Breite fällt heraus. Die Brechzahl im Transparenzfenster merkt sich nur **Lage und Stärke** der fernen Absorber, nicht ihre Form — die Rechtfertigung der Sellmeier-Formel aus Ü 29.1 (c).

Ü 29.3 — Der Hilbert-Schnellweg (Übertragen). Die sgn-Verkettung aus diesem Kapitel heißt in der Signalverarbeitung **Hilbert-Transformation**, und SciPy hat sie eingebaut: `scipy.signal.hilbert(x)` liefert zu einem reellen Signal das „analytische Signal" $x + \mathrm{i}\,H[x]$. (a) Finde durch Probieren am Lorentz-Modell heraus, wie $\mathrm{Re}\,\chi$ aus `hilbert(Im χ)` zu gewinnen ist (Vorzeichen!). (b) Vergleiche Genauigkeit und Rechenzeit mit unserem `kk_re`. (c) Was braucht der Hilbert-Weg, was unser Integral nicht braucht?

Musterlösung zu Ü 29.3

```
import time
# scipy.signal.hilbert: analytisches Signal x + i·H[x] per FFT —
# H ist die Hilbert-Transformation (Faltung mit 1/(πt))
from scipy.signal import hilbert

t1 = time.perf_counter()
re_hil = -np.imag(hilbert(chi_lorentz(w_fft).imag))
dauer_hil = time.perf_counter() - t1
innen = np.abs(w_fft) < 5.0
print(f"(a) Re χ = −Im(hilbert(Im χ)); Abw. "
      f"{np.abs(re_hil[innen] - chi_lorentz(w_fft[innen]).real).max():.1e}")

t1 = time.perf_counter()
kk_re(w_test, ws, lambda w: chi_lorentz(w).imag)
dauer_kk = time.perf_counter() - t1
print(f"(b) hilbert: {dauer_hil*1000:.0f} ms für {NT} Stellen — "
      f"kk_re: {dauer_kk*1000:.0f} ms für {len(w_test)}")
```

```
(a) Re χ = −Im(hilbert(Im χ)); Abw. 2.1e-05
(b) hilbert: 8 ms für 262144 Stellen — kk_re: 164 ms für 60
```

(a) Mit unserer Konvention gilt $\mathrm{Re}\,\chi = -\mathrm{Im}(\texttt{hilbert}(\mathrm{Im}\,\chi))$ — das Minus steckt in SciPys Definition von H. (b) Der FFT-Weg liefert *alle* 2^{18} Stellen schneller, als unser Integral 60 Stellen schafft — pro Stelle ist er um Größenordnungen voraus. (c) Den Preis zahlt man an den Voraussetzungen: `hilbert` braucht das Signal auf einem **gleichmäßigen Gitter über die ganze Achse** (positive *und* negative Frequenzen) und behandelt es als periodisch — Abschneide-Artefakte wohnen an den Rändern. Unser `kk_re` nimmt jedes krumme Gitter (die `geomspace`-Verdichtung am Gold-Buckel!) und wertet gezielt einzelne Stellen aus. Werkzeugwahl wie immer: Massenware per FFT, Maßarbeit per Integral.

Ü 29.4 — Die Abschneide-Gesetze (Übertragen). Der Werkstattkasten behauptet zwei Fehlergesetze für das KK-Integral. Miss beide am Modell nach: (a) Obere Grenze W sweepen (Lorentz-Modell, Auswertung bei $\omega = 0{,}5$): Bestätige Fehler $\approx 2\gamma\omega_p^2/(3\pi W^3)$. (b) Untere Grenze a sweepen (Drude-Modell $\chi = -\omega_p^2/(\omega^2 + \mathrm{i}\gamma\omega)$ mit $\gamma = 0{,}05$, Auswertung bei $\omega = 1$): Bestätige Fehler $\approx (2/\pi)(\omega_p^2/\gamma)\,a$. Warum ist (b) die Gold-Lektion in Formelgestalt?

Musterlösung zu Ü 29.4

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(8.5, 3.4))

w = 0.5
Ws = np.array([2.0, 5.0, 10.0, 20.0, 50.0, 100.0])
f_oben = []
for W in Ws:
    g = np.linspace(1e-6, W, int(4000 * W))
    f_oben.append(abs(kk_re([w + (g[1] - g[0]) / 2], g,
                            lambda x: chi_lorentz(x).imag)[0]
                      - chi_lorentz(w + (g[1] - g[0]) / 2).real))
ax1.loglog(Ws, f_oben, "ko", ms=5, label="gemessen")
ax1.loglog(Ws, 2 * GAMMA * WP**2 / (3 * np.pi * Ws**3), "C0-",
           lw=1.2, label="2γω_p²/(3πW³)")
ax1.set_xlabel("obere Grenze W")
ax1.set_ylabel("Fehler in Re χ(0,5)")
ax1.set_title("oben: ∝ 1/W³ — verzeihlich")
ax1.legend(fontsize=8)

G_D = 0.05
def chi_drude(x):
    return -WP**2 / (x**2 + 1j * G_D * x)
afs = np.array([1e-4, 1e-3, 1e-2, 5e-2])
f_unten = []
for a in afs:
    g = np.unique(np.concatenate([np.geomspace(a, 1.0, 100_000),
                                  np.linspace(1.0, 200.0, 200_000)]))
    f_unten.append(abs(kk_re([1.0 + 2.5e-4], g,
                             lambda x: chi_drude(x).imag)[0]
                       - chi_drude(1.0 + 2.5e-4).real))
ax2.loglog(afs, f_unten, "ko", ms=5, label="gemessen")
ax2.loglog(afs, 2 / np.pi * WP**2 / G_D * afs, "C3-", lw=1.2,
           label="(2/π)(ω_p²/γ)·a")
ax2.set_xlabel("untere Grenze a (Drude)")
ax2.set_title("unten: ∝ a·ω_p²/γ — fatal")
ax2.legend(fontsize=8)
fig.tight_layout()
plt.show()
```

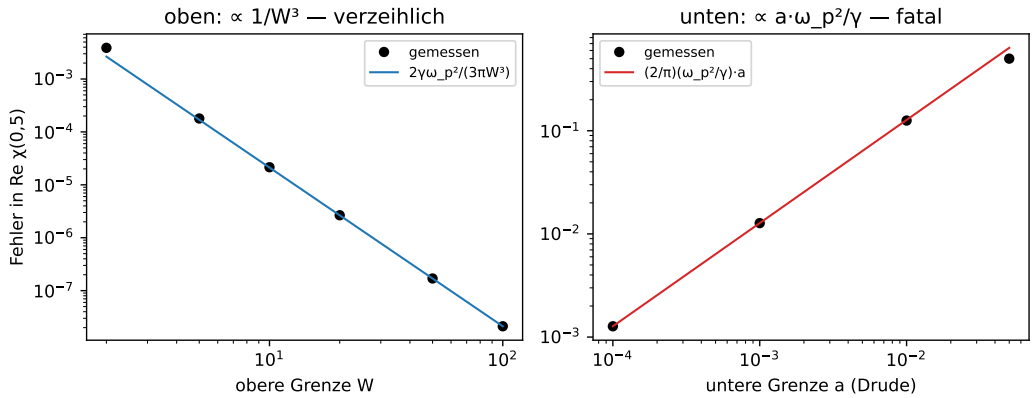


Abbildung 29.7: Die beiden Abschneide-Gesetze im Doppel-Log: Links kostet die obere Grenze W mit der Potenz −3 (Messpunkte auf der Theorielinie 2γω_p²/(3πW³)) — bei W = 20 ist der Fehler schon unter 10^{-5}. Rechts kostet das Loch [0, a] beim Drude-Modell nur linear, aber mit dem Riesen-Vorfaktor ω_p²/γ: Schon a = 0,01 (ein winziges Loch) verfälscht Re χ um 0,13. Gold im Messfenster scheiterte rechts, nicht links.

Beide Messreihen liegen auf ihren Theorielinien. (b) *ist* die Gold-Lektion: Das Messfenster begann bei $0{,}08$ — beim Drude-Anteil von Gold ($\sigma f_0^2/\gamma \approx 1241$) ein Loch mit vierstelligem Vorfaktor. Oben abzuschneiden hätte fast nichts gekostet; das Loch unten kostete 15 Einheiten von 20. Wer KK auf Leiter anwendet, braucht den Gleichstrom-Rand — oder die verankerten Varianten aus dem Kleingedruckten.

Das Kleingedruckte

Kramers, Kronig und die Röntgenröhre. Ralph Kronig (1926) und Hendrik Kramers (1927) fanden die Relation unabhängig, beim Studium der Röntgen-Dispersion. Der mathematische Unterbau (für Leser mit Funktionentheorie): Kausalität macht $\chi(\omega)$ in der oberen komplexen Halbebene **analytisch** — das Integral $\int_0^\infty \chi(t)\,\mathrm{e}^{\mathrm{i}\omega t}\,\mathrm{d}t$ konvergiert dort sogar besser als auf der reellen Achse, weil $\operatorname{Im}\omega > 0$ einen zusätzlichen Dämpfungsfaktor liefert. Eine analytische Funktion ist durch ihre Werte auf dem Rand bestimmt (Cauchy-Integral um die geschlitzte Halbebene), und genau dieses Randintegral *ist* die KK-Relation. Unser sgn-Weg ist die fouriertransformierte Fassung derselben Aussage: $\mathcal{F}[\mathrm{sgn}](\omega) \propto 1/\omega$, die Multiplikati-

on im Zeitbild wird zur $1/\omega$-Faltung im Frequenzbild — und diese Faltung ist das Hauptwert-Integral. Der Ehrenname des Satzes in der Mathematik: Titchmarsh-Theorem.

Sommerfeld, Brillouin und die Vorläufer. Als die Frage „$v_p > c$ — bricht da die Relativität?" um 1910 ernsthaft diskutiert wurde, rechneten Arnold Sommerfeld und Léon Brillouin (1914) die Front-Ausbreitung in einem Lorentz-Medium analytisch durch. Ergebnis: Die Front läuft exakt mit c, und hinter ihr kommen zwei leise „Boten", bevor das Hauptsignal eintrifft — der **Sommerfeld-Vorläufer** (die höchsten Frequenzen, für die das Medium durchsichtig wie Vakuum ist; in unserem Differenz-Experiment stecken sie im allerersten Anstieg der Differenz) und der **Brillouin-Vorläufer** (die Quasi-Statik). Beide sind im 21. Jahrhundert sauber gemessen worden — auch in Medien mit $v_g > c$ („superluminale" Pulse aus invertierten Atomen): Die Pulsspitze mag früher ankommen, die Front nie. Information reist mit der Front.

Subtraktive Kramers-Kronig. Das Laborproblem aus Abbildung 29.4 — Absorption nur im Messfenster bekannt — hat einen Standard-Ausweg: Kennt man $\operatorname{Re}\chi(\omega_1)$ an *einer* Ankerfrequenz aus einer unabhängigen Messung, kann man die Relation für die **Differenz** $\operatorname{Re}\chi(\omega) - \operatorname{Re}\chi(\omega_1)$ schreiben; der unbekannte Rest außerhalb des Fensters fällt dann teilweise heraus, weil beide Auswertepunkte denselben fernen Schwanz sehen. Mit mehreren Ankern („multiplicatively subtractive KK") wird das Verfahren zum Arbeitstier der Ellipsometrie und Reflexions-Spektroskopie: Aus $R(\omega)$ über ein breites Band plus wenige Ankerpunkte entstehen komplette n-und-k-Tabellen — fast alle optischen Konstanten in Datenbanken sind so entstanden, auch die Messdaten hinter Meeps Gold-Fit.

Die Summenregel zählt Elektronen. In der Quantenfassung (Thomas-Reiche-Kuhn-Summenregel) steht rechts von $\int \omega \operatorname{Im}\chi \,\mathrm{d}\omega = \pi\omega_p^2/2$ die Gesamtzahl der Elektronen pro Volumen — daher der Name *f-Summenregel*: Die „Oszillatorstärken" f_k aller Übergänge summieren sich auf die Elektronenzahl. Praktiker prüfen damit Messreihen: Integriert die gemessene Absorption auf zu wenig Elektronen, fehlt ein Band — die Summenregel ist die Bilanzprüfung der Spektroskopie.

Kausalitätsprüfung jenseits der Optik. Die Relation gilt für *jede* lineare kausale Antwort: mechanische Nachgiebigkeit, akustische Impedanz, elektrische Netzwerke. In der Hochfrequenztechnik wird sie rückwärts als Lügendetektor benutzt: Gemessene oder simulierte S-Parameter einer Leitung (Kapitel 22), deren Real- und Imaginärteil die Hilbert-Verkettung verletzen, sind physikalisch unmöglich — Standardtest in jedem Schaltungssimulator, bevor ein Modell in die Zeitbereichs-Simulation darf. Auch unser FDTD lebt davon: Meeps Materialfits (wie die Gold-Karte) sind als Summen kausaler

Drude-Lorentz-Terme konstruiert, damit der Zeitschritt-Algorithmus nie in die Zukunft greifen muss.

30 Wo Maxwell endet: Photonen und Wahrscheinlichkeit

Neunundzwanzig Kapitel lang war Licht ein Feld. Wir haben es diskretisiert, durch Spalte geschickt, in Resonatoren gesperrt, an Antennen abgestrahlt und rückwärts laufen lassen — und jede einzelne Messung hat die Feldbeschreibung bestätigt. Dieses Kapitel erzählt, wo sie endet. Denn dreht man die Leistung einer Lichtquelle weit genug herunter und hängt einen empfindlichen Detektor dahinter, dann passiert etwas, das kein kontinuierliches Feld tun dürfte: Es **klickt**. Licht kommt in einzelnen, unteilbaren Treffern an — Photonen, jedes mit der Energie $E = hf$. Zwischen zwei Klicks: nichts.

Das klingt nach Abrissbirne für alles Bisherige — und ist das Gegenteil. Die Pointe dieses Kapitels, vorweggenommen: Die Maxwell-Rechnung bleibt **exakt richtig**, sie wechselt nur den Beruf. Was wir 29 Kapitel lang als Helligkeit $|E|^2$ ausgerechnet haben, ist — Photon für Photon gelesen — die **Wahrscheinlichkeitsverteilung** für den Ort des nächsten Treffers. Wir werden das nicht behaupten, sondern nachspielen: ein Doppelspalt, ein Zufallsgenerator, und das Streifenmuster aus Kapitel 14 entsteht aus einzelnen Punkten neu. Vorher klären wir, warum uns die Körnigkeit des Lichts im Alltag nie auffällt, und warum eine simple Frequenz-Schwelle (der photoelektrische Effekt) das Feldbild in Erklärungsnot bringt. Am Ende lösen wir die älteste Zusage aus Kapitel 17 ein — die Tunnel-Formel der Quantenmechanik ist ein alter Bekannter — und ziehen eine ehrliche Grenze: was Maxwell nicht kann, und was trotzdem für immer bleibt.

Lernziele

Nach diesem Kapitel kannst du …

1. … mit $E = hf$ **Photonen-Budgets** rechnen und weißt, warum ein WLAN-Router niemals körnig funkt ($6 \cdot 10^{22}$ Photonen pro Sekunde), ein Röntgengerät aber sehr wohl einzelne Quanten zählt,
2. … den **photoelektrischen Effekt** erklären: Die Schwelle sitzt in der *Frequenz*, nicht in der *Helligkeit* — rotes Licht löst aus Cäsium kein Elektron, egal wie hell es ist,

3. … den **Einzelphotonen-Doppelspalt** als Monte-Carlo-Experiment nachbauen: einzelne Treffer ziehen, das Histogramm konvergiert mit $1/\sqrt{N}$ gegen das Maxwell-Muster,
4. … erklären, warum beim „Welcher-Weg"-Experiment die Streifen verschwinden: **Amplituden interferieren, Wahrscheinlichkeiten nicht** ($P \neq P_1 + P_2$),
5. … **Schrotrauschen** als $\sqrt{N}$-Gesetz messen und weißt, warum Nachtaufnahmen körnig sind,
6. … die **Tunnel-Brücke** schlagen: Die Schrödinger-Gleichung liefert für die Rechteckbarriere dieselbe $\sinh^2$-Formel wie unser FTIR-Experiment aus Kapitel 17 — und du kannst ausrechnen, warum ein Rastertunnelmikroskop einzelne Atome sieht.

30.1 Der Klick — und warum wir ihn nie bemerken

Max Planck führte die Größe h im Jahr 1900 als Verzweiflungstat ein (um das Spektrum glühender Körper zu retten), Albert Einstein nahm sie 1905 ernst: Licht der Frequenz f tauscht Energie nur in unteilbaren Portionen

$$E_{\text{Photon}} = hf = \frac{hc}{\lambda}, \qquad h = 6{,}626 \cdot 10^{-34}\ \text{Js}$$

aus. Wenn das stimmt, müsste Licht körnig sein wie Regen auf einem Blechdach. Warum haben wir in 29 Kapiteln nichts davon gemerkt? Rechnen wir nach — drei Sender, die wir aus diesem Buch kennen:

```
import matplotlib.pyplot as plt
import numpy as np

H = 6.62607015e-34       # Plancksches Wirkungsquantum [Js]
C0 = 2.99792458e8        # Lichtgeschwindigkeit [m/s]
EV = 1.602176634e-19     # 1 Elektronvolt [J]


def photonen_pro_sekunde(P, lam=None, f=None):
    """Leistung P in Photonenzahl pro Sekunde umrechnen."""
    e_photon = H * (C0 / lam if lam else f)
    return e_photon, P / e_photon


for name, P, lam, f in [
```

```
        ("Laserpointer 1 mW, 532 nm  ", 1e-3, 532e-9, None),
        ("WLAN-Router 100 mW, 2,45 GHz", 0.1, None, 2.45e9),
        ("UKW-Sender 100 kW, 100 MHz ", 100e3, None, 100e6)]:
    e, n = photonen_pro_sekunde(P, lam, f)
    print(f"{name}: {e / EV:9.3g} eV  →  {n:8.2g} Photonen/s")
```

```
Laserpointer 1 mW, 532 nm  :      2.33 eV  →   2.7e+15 Photonen/s
WLAN-Router 100 mW, 2,45 GHz:  1.01e-05 eV  →   6.2e+22 Photonen/s
UKW-Sender 100 kW, 100 MHz :  4.14e-07 eV  →   1.5e+30 Photonen/s
```

Ein grüner Laserpointer feuert fast drei *Billiarden* Photonen pro Sekunde ab; selbst in einer Nanosekunde sind es noch Millionen. Der WLAN-Router liegt um sieben weitere Größenordnungen darüber, der UKW-Sender bei 10^{30} — das ist der Grund, warum die gesamte Funktechnik der Kapitel 21 bis 25 nie ein Photon zu Gesicht bekommt: Bei solchen Strömen ist die Körnigkeit so unsichtbar wie einzelne Wassermoleküle in einem Fluss. Maxwell ist die Beschreibung des Flusses — und sie ist dort vollständig. Die Klicks tauchen erst auf, wenn man entweder die Leistung absurd klein macht (gleich am Doppelspalt) oder die Photonenenergie groß: Ein Röntgenquant trägt $50\,000$ eV, da zählt jeder Treffer einzeln (Übung 30.1).

30.2 Die Schwelle sitzt in der Frequenz

Dass die Portionen real sind und nicht bloß Buchhaltungstrick, zeigt das älteste Experiment dieser Geschichte: Licht fällt auf eine Metallplatte und schlägt Elektronen heraus — der **photoelektrische Effekt**. Im Feldbild wäre die Sache klar: Mehr Helligkeit heißt mehr $|E|^2$, heißt mehr Energie pro Sekunde, heißt schnellere Elektronen.

> Vorhersage (PRIMM)
>
> Eine Cäsium-Platte (Austrittsarbeit $W = 2{,}14$ eV) wird mit rotem Licht ($\lambda = 700$ nm) beleuchtet. Es kommen keine Elektronen. Jetzt machen wir die Lampe **hundertmal heller**. Was passiert — (a) jetzt kommen Elektronen, (b) es kommen immer noch keine, aber bei hundertfacher Wartezeit schon, oder (c) es kommen nie welche, egal wie hell und wie lange?

Die Antwort des Experiments ist (c) — und sie hat Einstein den Nobelpreis eingebracht. Im Photonenbild ist sie zwingend: Ein Elektron schluckt immer genau *ein* Photon. Reicht dessen Portion hf nicht über die Austrittsarbeit W,

hilft auch ein Trommelfeuer aus zu kleinen Portionen nichts — hundertmal heller heißt hundertmal *mehr* Photonen, nicht *größere*:

```
W_CS, W_CU = 2.14, 4.65      # Austrittsarbeiten Cäsium/Kupfer [eV]
for lam, name in [(550e-9, "grün  550 nm"), (700e-9, "rot   700 nm")]:
    e = H * C0 / lam / EV
    urteil = "Elektronen!" if e > W_CS else "nichts – egal wie hell"
    print(f"{name}: {e:.3f} eV  gegen W_Cs = {W_CS} eV  →  {urteil}")
print(f"Grenzwellenlänge Cäsium: hc/W = {H*C0/(W_CS*EV)*1e9:.0f} nm")
print(f"Grenzwellenlänge Kupfer: hc/W = {H*C0/(W_CU*EV)*1e9:.0f} nm")
```

```
grün  550 nm: 2.254 eV  gegen W_Cs = 2.14 eV  →  Elektronen!
rot   700 nm: 1.771 eV  gegen W_Cs = 2.14 eV  →  nichts – egal wie hell
Grenzwellenlänge Cäsium: hc/W = 579 nm
Grenzwellenlänge Kupfer: hc/W = 267 nm
```

Grünes Licht (2,25 eV) liegt knapp über der Cäsium-Schwelle, rotes (1,77 eV) knapp darunter — die Grenze verläuft mitten durchs sichtbare Spektrum, bei 579 nm. Kupfer braucht mit 4,65 eV hartes Ultraviolett. Und die Helligkeit? Sie bestimmt nur, *wie viele* Elektronen pro Sekunde kommen — nie, *ob* und *wie schnell*. Die Überschuss-Energie $hf - W$ steckt in der Geschwindigkeit des Elektrons, und genau das macht den Effekt zum Messgerät: Trägt man die Bremsspannung, die die schnellsten Elektronen gerade noch stoppt, gegen die Frequenz auf, entsteht eine Gerade mit der Steigung h/e. Robert Millikan hat auf diese Weise h vermessen — in der erklärten Absicht, Einstein zu *widerlegen* (mehr dazu im Kleingedruckten); Übung 30.2 spielt seine Auswertung mit `np.polyfit` nach.

30.3 Der Doppelspalt, ein Photon nach dem anderen

Jetzt das Experiment, an dem sich entscheidet, ob das Feldbild oder das Teilchenbild gewinnt — und die Antwort ist berühmt geworden: beide. Wir nehmen den Doppelspalt aus Kapitel 14 und drehen die Quelle so leise, dass immer nur **ein einziges Photon** unterwegs ist. G. I. Taylor hat das 1909 wirklich getan (mit einer Gasflamme hinter Rußglas und drei Monaten Belichtungszeit).

Naheliegende Vermutung: „Mit einzelnen Photonen verschwinden die Streifen — ein Photon kann ja nur durch einen Spalt gehen“

Warum sie naheliegt: Interferenz hieß 29 Kapitel lang: Zwei Wellenzüge — einer pro Spalt — überlagern sich am Schirm. Ein einzelnes, unteilbares Photon hat aber scheinbar keinen Partner; es sollte sich für einen Spalt *entscheiden* müssen, und übrig bliebe die strukturlose Summe zweier Einzelspalt-Flecken.
Was stattdessen stimmt: Die Streifen bleiben — in Taylors Experiment, in Tonomuras berühmter Elektronen-Fassung von 1989, in jedem modernen Praktikum. Jedes Photon landet als *ein* Punkt an *einem* Ort (das ist der Teilchen-Anteil), aber die **Wahrscheinlichkeit**, wo, folgt exakt dem Interferenzmuster der Maxwell-Welle (das ist der Wellen-Anteil). Dirac hat es auf die kürzeste Formel gebracht: Jedes Photon interferiert *mit sich selbst*. Die Welle, die durch beide Spalte läuft, ist keine Woge aus vielen Photonen — sie ist die Amplituden-Buchhaltung des einzelnen. Erst eine Messung, *welchen* Spalt das Photon nahm, zerstört das Muster — dazu gleich mehr.

Das können wir nachspielen, und zwar quantitativ. **Versuchsaufbau:** Die Frage ist, ob aus einzeln gezogenen Zufallstreffern wirklich das Kapitel-14-Muster entsteht — und wie schnell. Die Bühne sind Praktikums-Maße: roter Laser ($\lambda = 650$ nm), Spaltabstand $d = 0{,}25$ mm, Spaltbreite $a = 0{,}05$ mm, Schirm in $L = 2$ m Entfernung; das Maxwell-Muster ist die Kapitel-14-Formel Streifen × Hülle mit dem Streifenabstand $\lambda L/d = 5{,}2$ mm. Daraus machen wir eine Wahrscheinlichkeitsverteilung (normieren auf Summe 1) und lassen den Zufallsgenerator aus Kapitel 10 einzelne Treffer ziehen. Gemessen wird der rms-Abstand zwischen Treffer-Histogramm und $|E|^2$ für $N = 10^2$ bis 10^6. Erfolgskriterium: Das Histogramm konvergiert gegen das Muster, und der Fehler fällt wie $1/\sqrt{N}$ — das Standard-Gesetz jeder Zählstatistik.

```
LAM = 650e-9                    # roter Laser
D_SPALT = 0.25e-3               # Spaltabstand
A_SPALT = 0.05e-3               # Spaltbreite
L_SCHIRM = 2.0                  # Abstand zum Schirm
x = np.linspace(-0.03, 0.03, 1200)        # Schirmkoordinate [m]
sint = x / np.hypot(x, L_SCHIRM)          # sin(theta) zu jedem x


def intensitaet(sint, beide=True):
    """Kap.-14-Doppelspalt: Hülle (Spaltbreite) × Streifen (Abstand)."""
    huelle = np.sinc(A_SPALT * sint / LAM) ** 2
```

```
    if beide:
        return huelle * np.cos(np.pi * D_SPALT * sint / LAM) ** 2
    return huelle


p_doppel = intensitaet(sint)
p_doppel = p_doppel / p_doppel.sum()      # |E|² → Wahrscheinlichkeit

rng = np.random.default_rng(30)           # feste Saat: reproduzierbar
fehler = []
Ns = np.array([100, 1000, 10_000, 100_000, 1_000_000])
hists = {}
for N in Ns:
    # rng.choice: zieht N Indizes, jeder mit seiner Wahrscheinlichkeit
    # aus p= — das ist unser "Photon trifft den Schirm"
    treffer = rng.choice(len(x), size=N, p=p_doppel)
    # np.bincount: zählt, wie oft jeder Index vorkommt (das Histogramm)
    hist = np.bincount(treffer, minlength=len(x)) / N
    hists[N] = hist
    fehler.append(np.sqrt(((hist - p_doppel) ** 2).mean()))
steigung_mc = np.polyfit(np.log(Ns), np.log(fehler), 1)[0]
print(f"Konvergenz: rms ∝ N^{steigung_mc:.2f}  (Zählstatistik: −0,5)")
```

```
Konvergenz: rms ∝ N^-0.51  (Zählstatistik: −0,5)
```

Bevor du das Bild ansiehst, die PRIMM-Frage im Kleinen: *Wo landet das allererste Photon?* Antwort: irgendwo — einzeln betrachtet ist jeder Treffer strukturlos zufällig, nur die exakten Nullstellen des Musters sind tabu. Die Ordnung wohnt nicht im einzelnen Punkt, sondern in der Statistik:

```
fig, achsen = plt.subplots(1, 3, figsize=(9.5, 3.2), sharex=True)

punkte = rng.choice(len(x), size=200, p=p_doppel)
# rng.uniform: gleichverteilte Zufallszahlen — hier nur als Streu-Höhe,
# damit die 200 Schirm-Punkte optisch nicht aufeinanderliegen
achsen[0].plot(x[punkte] * 1e3, rng.uniform(0, 1, 200), "k.", ms=2.5)
achsen[0].set_title("N = 200: Punkte", fontsize=10)
achsen[0].set_ylabel("Schirm (Streu-Höhe)")

for ax, N in [(achsen[1], 10_000), (achsen[2], 1_000_000)]:
    ax.plot(x * 1e3, hists[N] / p_doppel.max(), "C0-", lw=0.9,
            label=f"Histogramm N = {N:,}".replace(",", " "))
```

```
    ax.plot(x * 1e3, p_doppel / p_doppel.max(), "C3-", lw=1.1,
            label="|E|² (Maxwell)")
    ax.set_title(f"N = {N:,}".replace(",", " "), fontsize=10)
    ax.legend(fontsize=7)
for ax in achsen:
    ax.set_xlabel("x [mm]")
plt.tight_layout()
plt.show()
```

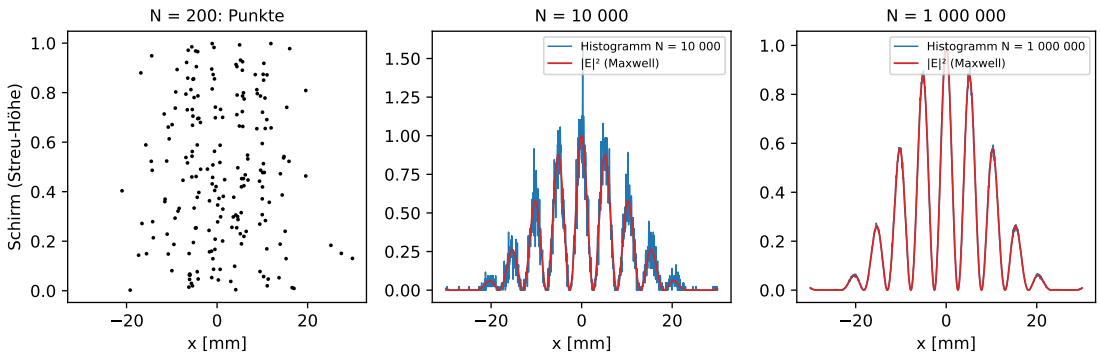


Abbildung 30.1: Der Doppelspalt, Photon für Photon: Bei N = 200 Treffern (links, jeder Punkt ein Photon; die Höhe ist gewürfelt, damit die Punkte nicht aufeinanderliegen) ist kaum Ordnung zu ahnen. Bei N = 10 000 (Mitte, Histogramm) treten die Streifen hervor, bei N = 1 000 000 (rechts) liegt das Histogramm deckungsgleich auf der roten Maxwell-Kurve $|E|^2$ aus Kapitel 14. Die klassische Intensität ist die Wahrscheinlichkeitsverteilung des einzelnen Treffers.

Und die Konvergenz folgt dem Wurzel-Gesetz auf zwei Nachkommastellen — wer das Muster doppelt so glatt will, braucht viermal so viele Photonen:

```
fig, ax = plt.subplots(figsize=(6.4, 3.4))
ax.loglog(Ns, fehler, "ko", ms=5, label="rms gemessen")
ax.loglog(Ns, fehler[0] * (Ns / Ns[0]) ** -0.5, "C0-", lw=1.2,
          label="Theorie: ∝ 1/√N")
ax.set_xlabel("Photonenzahl N")
ax.set_ylabel("rms(Histogramm − |E|²)")
ax.set_title(f"gemessene Steigung: {steigung_mc:.2f}")
ax.legend(fontsize=9)
plt.show()
```

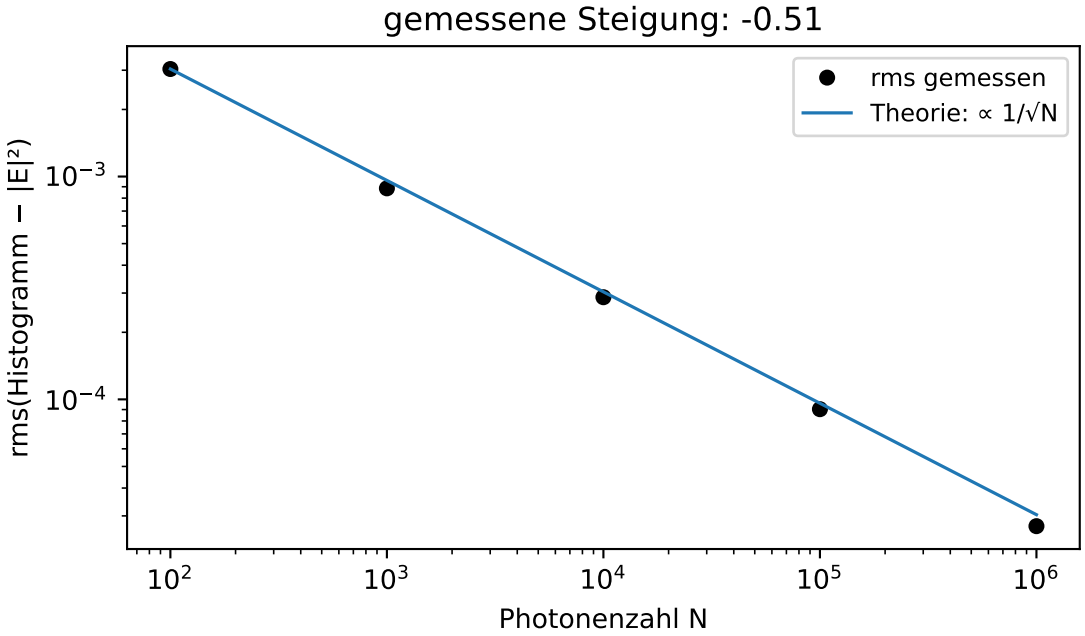


Abbildung 30.2: Das Wurzel-Gesetz der Zählstatistik: Der rms-Abstand zwischen Treffer-Histogramm und Maxwell-Muster fällt über vier Dekaden Photonenzahl exakt auf der 1/√N-Geraden (gemessene Steigung −0,51). Dasselbe Gesetz regiert jede Nachtaufnahme, jede Strahlentherapie-Dosis und jedes Radioteleskop-Bild.

30.4 Die Regel dahinter: Maxwell als Amplituden-Maschine

Halten wir fest, was hier passiert ist — denn es ist der Kern des ganzen Kapitels. Wir haben **nichts neu gerechnet**: Die Kurve, gegen die die Treffer konvergieren, ist Zeile für Zeile die Doppelspalt-Formel aus Kapitel 14. Neu ist allein die *Lesart*:

$$\underbrace{|E(x)|^2}_{\text{Kap. 1–29: Helligkeit}} = \underbrace{p(x)}_{\text{Kap. 30: Wahrscheinlichkeit pro Photon}} .$$

Maxwell ist — in der Sprache der Quantenphysik — die **Amplituden-Maschine** des Lichts: Die Felder E und B sind die Wahrscheinlichkeits*amplituden* des einzelnen Photons, ihre Interferenz, Brechung, Beugung und Resonanz sind exakt die Amplituden-Spielregeln, und erst das

Betragsquadrat wird zur messbaren Zählrate. Darum war alles aus Teil IV bis VII automatisch richtig — und darum bekommt auch die schönste Pointe aus Kapitel 14 ein Quanten-Echo: Dort hieß es „Licht plus Licht ist nicht heller — die Energie wird umverteilt". Jetzt heißt es: Die *Wahrscheinlichkeit* wird umverteilt. Ein Photon mehr im Maximum ist ein Photon weniger im Minimum; die Gesamtzahl stimmt immer.

Dass sich hinter dem harmlosen Wort „Lesart" ein echter Unterschied verbirgt, zeigt das Gegenexperiment. Was passiert, wenn wir *wissen*, durch welchen Spalt jedes Photon geht — etwa indem wir abwechselnd einen Spalt zukleben? Dann gibt es nichts mehr zu interferieren: Jedes Photon hat nur noch *eine* Amplitude, und die Wahrscheinlichkeiten der beiden Wege addieren sich klassisch:

```
p_einzel = intensitaet(sint, beide=False)
p_summe = p_einzel / p_einzel.sum()   # P₁ + P₂: nur die Hülle

zentral = np.abs(x) < 0.005
def sichtbarkeit(p):
    """Streifen-Sichtbarkeit (max−min)/(max+min) im Zentrum."""
    return ((p[zentral].max() - p[zentral].min())
            / (p[zentral].max() + p[zentral].min()))


print(f"Sichtbarkeit, beide Spalte offen:   {sichtbarkeit(p_doppel):.3f}")
print(f"Sichtbarkeit mit Welcher-Weg-Wissen: {sichtbarkeit(p_summe):.3f}")
print(f"am Hauptmaximum: p_beide / (p₁+p₂) = "
      f"{p_doppel.max() / p_summe[np.argmax(p_doppel)]:.2f}")
```

```
Sichtbarkeit, beide Spalte offen:   1.000
Sichtbarkeit mit Welcher-Weg-Wissen: 0.061
am Hauptmaximum: p_beide / (p₁+p₂) = 2.00
```

```
fig, ax = plt.subplots(figsize=(7.0, 3.6))
ax.plot(x * 1e3, p_doppel / p_doppel.max(), "C0-", lw=1.3,
        label="beide offen: |A₁ + A₂|²")
ax.plot(x * 1e3, p_summe / p_doppel.max(), "C3-", lw=1.3,
        label="Welcher-Weg: P₁ + P₂")
ax.set_xlabel("x [mm]")
ax.set_ylabel("Wahrscheinlichkeit (normiert)")
ax.set_title("Amplituden interferieren — Wahrscheinlichkeiten nicht")
ax.legend(fontsize=9)
plt.show()
```

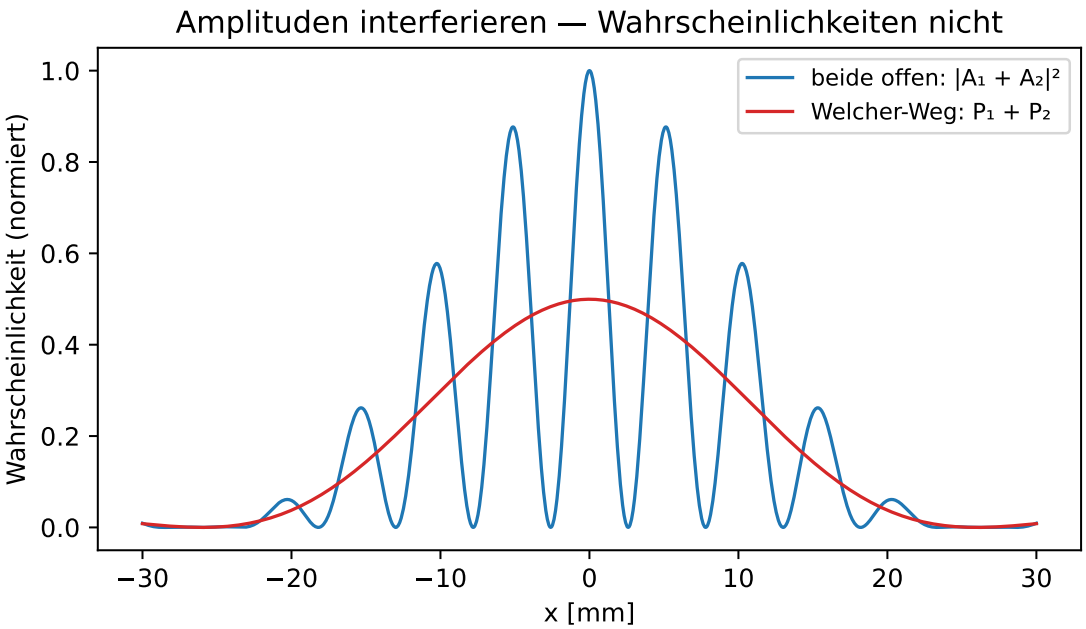


Abbildung 30.3: Der Unterschied zwischen Amplituden- und Wahrscheinlichkeits-Addition: Mit beiden Spalten offen (blau) interferieren die Amplituden — Streifen mit Sichtbarkeit 1,000, am Hauptmaximum doppelt so hoch wie die rote Kurve. Weiß man dagegen, welchen Weg jedes Photon nahm (rot: die Summe zweier Einzelspalt-Verteilungen), addieren sich nur noch Wahrscheinlichkeiten — die Hülle bleibt, die Streifen sind weg (Sichtbarkeit 0,061). $P \neq P_1 + P_2$: Das ist der messbare Kern der Quantenmechanik.

Das ist keine Fußnote, sondern *die* Bruchkante zwischen klassischer Statistik und Quantenwelt: Ein klassisches Teilchen hätte immer einen Weg, und $P = P_1 + P_2$ wäre ein Naturgesetz. Das Photon hat stattdessen Amplituden — und unser 29-Kapitel-Werkzeugkasten ist genau deshalb so wertvoll, weil er diese Amplituden exakt ausrechnet. Übung 30.3 baut das Welcher-Weg-Experiment als Monte-Carlo nach.

30.5 Schrotrauschen: die Körnigkeit wird messbar

Wenn Photonen würfeln, muss jede Lichtmessung *rauschen* — nicht weil das Messgerät schlecht wäre, sondern prinzipiell. Die Statistik dahinter kennen wir aus Kapitel 10 (der Zufallsgenerator) in neuer Rolle: Die Zahl der Treffer

in einem festen Zeitfenster folgt der **Poisson-Verteilung**, und deren Markenzeichen ist $\sigma = \sqrt{N}$ — bei im Mittel N Photonen pro Pixel schwankt die Zählung um $\sqrt{N}$. Das Signal-zu-Rausch-Verhältnis wächst also nur mit der Wurzel:

$$\mathrm{SNR} = \frac{N}{\sqrt{N}} = \sqrt{N}.$$

Versuchsaufbau: Wir „fotografieren" unser Doppelspalt-Muster mit einer simulierten Kamera: Jeder Pixel erwartet $N \cdot |E|^2/|E|^2_{\max}$ Photonen und zieht seine tatsächliche Zählung aus der Poisson-Verteilung (`rng.poisson`). Gemessen wird das SNR am hellsten Pixel über 200 Aufnahmen, bei Erwartungswert 1 und 100. Erfolg: SNR = 1 bzw. 10 — das Wurzel-Gesetz.

```
I2d = intensitaet(sint)[None, :] * np.ones((40, 1))   # Streifen-"Foto"
for n_pix in (1.0, 100.0):
    erwartung = n_pix * I2d / I2d.max()
    # rng.poisson: zieht ganzzahlige Zaehlungen mit dem gegebenen
    # Erwartungswert — das Photonen-Wuerfeln jedes Pixels
    bilder = rng.poisson(erwartung[None, :, :] * np.ones((200, 1, 1)))
    hell = erwartung == n_pix
    snr = bilder[:, hell].mean() / bilder[:, hell].std()
    print(f"Erwartung {n_pix:4g} Photonen/Pixel → SNR = {snr:5.2f}  "
          f"(√N = {np.sqrt(n_pix):.1f})")
```

```
Erwartung    1 Photonen/Pixel → SNR =  1.01  (√N = 1.0)
Erwartung  100 Photonen/Pixel → SNR = 10.04  (√N = 10.0)
```

Genau deshalb sind Nachtaufnahmen körnig, rauschen Röntgenbilder bei niedriger Dosis, und genau deshalb belichten Astronomen stundenlang: Wer das Rauschen halbieren will, muss **viermal** so viele Photonen sammeln — dasselbe $1/\sqrt{N}$ wie bei unserer Doppelspalt-Konvergenz, denn es ist dieselbe Statistik.

30.6 Die Tunnel-Brücke: eine alte Bekannte

Bleibt eine Zusage aus Kapitel 17 offen. Dort haben wir Licht durch einen „verbotenen" Luftspalt tunneln lassen (FTIR) und die Transmission über drei Dekaden auf einer sinh-Formel wiedergefunden:

$$T_{\text{FTIR}} = \frac{1}{1 + C \cdot \sinh^2(\kappa d)}.$$

Die Quantenmechanik beschreibt ein Elektron mit der Schrödinger-Gleichung — mathematisch eine Wellengleichung für die Amplitude ψ, deren Betragsquadrat (wie eben bei unserem Photon!) die Aufenthalts-Wahrscheinlichkeit ist. Lässt man dieses Elektron mit Energie E gegen eine Energie-Wand der Höhe $V > E$ laufen, liefert dieselbe Rechnung wie in Kapitel 17 (Welle, evaneszenter Saum, Anschlussbedingungen):

$$T_{\text{QM}} = \frac{1}{1 + \dfrac{V^2}{4E(V-E)} \sinh^2(\kappa d)}, \qquad \kappa = \frac{\sqrt{2m(V-E)}}{\hbar}.$$

Dieselbe Bauform, Term für Term — nur der Vorfaktor C und das Rezept für κ sind material- bzw. teilchenspezifisch. Das ist kein Zufall, sondern derselbe Mechanismus: Eine Wellen-Amplitude klingt in einer verbotenen Zone exponentiell ab, und was am anderen Ende noch übrig ist, läuft weiter. Mit Zahlen für ein Elektron:

```
HQUER = 1.054571817e-34
ME = 9.1093837015e-31

E_EL, V_B = 1.0 * EV, 2.0 * EV       # Elektron 1 eV, Wand 2 eV
kappa = np.sqrt(2 * ME * (V_B - E_EL)) / HQUER
d = 0.5e-9
T = 1 / (1 + V_B**2 * np.sinh(kappa * d)**2 / (4 * E_EL * (V_B - E_EL)))
print(f"κ = {kappa*1e-9:.2f} pro Nanometer")
print(f"T(d = 0,5 nm) = {T:.4f}")
print(f"0,1 nm näher dran → Strom × e^(2κ·0,1 nm) = "
      f"{np.exp(2 * kappa * 0.1e-9):.2f}")
```

```
κ = 5.12 pro Nanometer
T(d = 0,5 nm) = 0.0235
0,1 nm näher dran → Strom × e^(2κ·0,1 nm) = 2.79
```

$\kappa = 5{,}1$ pro Nanometer — die verbotene Zone des Elektrons ist Bruchteile eines Nanometers tief, eine Million Mal kürzer als beim Licht in Kapitel 17. Genau diese Winzigkeit ist zur Messtechnik geworden: Das **Rastertunnelmikroskop** fährt eine Metallspitze im Abstand unter einem Nanometer über eine Oberfläche und misst den Tunnelstrom. Ein Zehntel Nanometer

näher — etwa ein halber Atomdurchmesser — und der Strom springt um fast den Faktor drei (bei realistischen Austrittsarbeiten um eine ganze Größenordnung, Übung 30.4). Darum sieht ein STM einzelne Atome, und darum genügt ihm als „Linse" das unterste Atom seiner Spitze. Die Mathematik dahinter hast du in Kapitel 17 an einem Glasprisma gelernt.

30.7 Die ehrliche Grenzziehung

Was kann Maxwell nun *nicht* — und was bleibt? Die Grenze verläuft schärfer, als man nach diesem Kapitel fürchten könnte. Maxwell **fehlt**: der einzelne Klick (die Theorie kennt nur kontinuierliche Felder, keine Detektions- Ereignisse), die Zählstatistik jenseits von Poisson (echte Einzelphotonen-Quellen zeigen *Antibunching* — nach einem Klick folgt garantiert eine Pause; das kann keine klassische Welle, siehe Kleingedrucktes) und die Verschränkung mehrerer Photonen (die Korrelationen, auf denen Quantenkryptografie und Quantencomputer beruhen). Wann immer einzelne Quanten *als einzelne* zählen, braucht es die Quantenelektrodynamik.

Maxwell **bleibt**: alles andere — und „alles andere" ist fast alles. Jede Antenne, jeder Wellenleiter, jede Faser, jede Linse, jeder Resonator, jedes Radar und jedes WLAN-Modul arbeitet mit Photonenströmen von 10^{15} aufwärts, wo die Körnigkeit im Wurzel-Gesetz versinkt. Und selbst dort, wo gezählt wird, rechnet die Quantenoptik die *Amplituden* nach wie vor mit den Maxwell-Gleichungen aus — unser Buch war, ohne es zu sagen, von Anfang an auch ein Buch über die Wellenseite der Quantenwelt. Die Front-Regel aus Kapitel 29 übersteht den Umzug übrigens unbeschadet: Auch verschränkte Photonen tragen keine Botschaft schneller als c.

30.8 Das Kapitel-Programm

`programme/kap30/kap30_photonen.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die drei Photonen-Budgets, die Photoeffekt-Schwellen und Grenzwellenlängen, die Millikan-Gerade (h auf wenige Prozent aus fünf verrauschten Messpunkten), den Monte-Carlo-Doppelspalt samt $N^{-0,5}$-Konvergenz, das Welcher-Weg-Experiment (Sichtbarkeit 1,000 → 0,06, Faktor 2 am Hauptmaximum), die Poisson-SNR und die Tunnel-Zahlen. Der Zufall läuft mit fester Saat — jeder Lauf liefert exakt dieselben Treffer. Reines NumPy, kein Meep, Laufzeit wenige Sekunden.

Merkkasten

- **Photonen:** Licht tauscht Energie in Portionen $E = hf$. Funk: 10^{-5} eV und 10^{22} Photonen/s (nie körnig); Licht: ~2 eV und 10^{15}/s (fast nie); Röntgen: 10^4 eV (jeder Treffer zählt).
- **Photoeffekt:** Die Schwelle sitzt in der Frequenz ($hf > W$), die Helligkeit liefert nur die Rate. Die Stoppspannungs-Gerade hat die Steigung h/e (Millikan).
- **|E|² ist Wahrscheinlichkeit:** Jedes Photon landet einzeln und zufällig; das Histogramm konvergiert mit $1/\sqrt{N}$ gegen die Maxwell-Intensität. Maxwell ist die Amplituden-Maschine des Photons.
- **Amplituden interferieren, Wahrscheinlichkeiten nicht:** Welcher-Weg-Wissen ersetzt $|A_1 + A_2|^2$ durch $P_1 + P_2$ — die Streifen verschwinden.
- **Schrotrauschen:** Poisson-Statistik, SNR $= \sqrt{N}$ — Rauschen halbieren kostet viermal so viele Photonen.
- **Tunnel-Brücke:** Schrödingers Rechteckbarriere hat dieselbe $\sinh^2$-Formel wie FTIR (Kap. 17); $\kappa \approx 5$–11/nm macht das STM atomar empfindlich.
- **Grenze:** Maxwell fehlt der Klick, das Antibunching und die Verschränkung — für alles mit vielen Photonen (und für alle Amplituden) ist er vollständig.

Roter Faden

Dieses Kapitel hat kein einziges neues Werkzeug gebraucht — nur neue Augen: Der **Doppelspalt aus Kapitel 14** wurde zur Wahrscheinlichkeitsverteilung (und seine „Energie wird umverteilt"-Pointe zur Wahrscheinlichkeits-Umverteilung), der **Zufallsgenerator aus Kapitel 10** zum Photonenwürfel, die **sinh-Formel aus Kapitel 17** zur Schrödinger-Brücke, das $|E|^2$ **aus Kapitel 21** zur Zählrate, und die **Front-Regel aus Kapitel 29** gilt auch für Quanten. Die Maxwell-Gleichungen, mit denen dieses Buch in Kapitel 3 begann, stehen damit an ihrem ehrlichen Platz: nicht widerlegt, sondern *eingebettet* — als exakte Amplituden-Theorie des Lichts, klassisch gelesen eine Feldtheorie, quantisch gelesen die halbe Quantenelektrodynamik. Nach vorn bleibt ein letztes Kapitel: **Kapitel 31** dreht den Spieß ein finales Mal um — statt eine gegebene Struktur zu simulieren, lassen wir den Computer selbst *entwerfen* (inverses Design) und ziehen das Schlusswort dieses Buchs.

Übungen

Ü 30.1 — Wo wird Licht körnig? (Verstehen). Rechne die Photonen-Budgets: (a) ein Handy beim Telefonieren ($0{,}2$ W bei $1{,}8$ GHz), (b) ein Mikrowellenherd (800 W bei $2{,}45$ GHz), (c) eine Röntgenröhre, deren Nutzstrahl 1 mW bei 50 keV Photonenenergie trägt. Wie viele Photonen pro Sekunde sind das jeweils — und in welchem Fall „sieht" ein Detektor einzelne Treffer?

Musterlösung zu Ü 30.1

```python
for name, P, e_ph in [
        ("Handy 0,2 W / 1,8 GHz   ", 0.2, H * 1.8e9),
        ("Mikrowelle 800 W        ", 800.0, H * 2.45e9),
        ("Röntgen 1 mW / 50 keV   ", 1e-3, 50e3 * EV)]:
    print(f"{name}: {e_ph/EV:8.2g} eV  →  {P/e_ph:8.2g} Photonen/s")
```

```
Handy 0,2 W / 1,8 GHz   :  7.4e-06 eV  →   1.7e+23 Photonen/s
Mikrowelle 800 W        :    1e-05 eV  →   4.9e+26 Photonen/s
Röntgen 1 mW / 50 keV   :    5e+04 eV  →   1.2e+11 Photonen/s
```

Handy und Mikrowelle liegen bei 10^{23} bis 10^{26} Photonen pro Sekunde — die Körnigkeit ist hoffnungslos unsichtbar (und nebenbei: ein 10^{-5}-eV-Photon kann kein Molekül ionisieren, egal wie viele es sind — die Photoeffekt-Logik gilt auch hier). Die Röntgenröhre dagegen liefert nur $1{,}2 \cdot 10^{11}$ Treffer pro Sekunde, jeder mit der 25 000-fachen Energie eines sichtbaren Photons: Medizinische Detektoren zählen sie tatsächlich einzeln, und die Dosis-Statistik folgt dem $\sqrt{N}$-Gesetz dieses Kapitels.

Ü 30.2 — Millikans Gerade (Verändern). Eine Cäsium-Fotozelle wird mit fünf Quecksilber-Linien beleuchtet (365, 405, 436, 546, 578 nm); gemessen werden die Stoppspannungen (mit ±10 mV Messrauschen). Erzeuge die Messreihe synthetisch mit $U = hf/e - W$, fitte die Gerade mit `np.polyfit` und gewinne h und W zurück. Wie genau wird h — und welche der fünf Linien trägt gar keine Information mehr?

Musterlösung zu Ü 30.2

```
rng_ue = np.random.default_rng(16)
lams = np.array([365.0, 405.0, 436.0, 546.0, 578.0]) * 1e-9
fs = C0 / lams
u_ideal = H * fs / EV - W_CS
nutzbar = u_ideal > 0.05         # deutlich über dem 10-mV-Rauschen
print("ideale Stoppspannungen [V]:",
      np.round(u_ideal, 3), "— nutzbar:", nutzbar)

# rng.normal: Gauß-Zufallszahlen mit Mittel 0 und Breite 10 mV —
# das Messrauschen des Voltmeters
u_mess = u_ideal[nutzbar] + rng_ue.normal(0, 0.01, nutzbar.sum())
steig, achse = np.polyfit(fs[nutzbar], u_mess, 1)
print(f"h aus dem Fit: {steig*EV:.3e} Js "
      f"(Abweichung {abs(steig*EV - H)/H*100:.1f} %)")
print(f"W aus dem Achsenabschnitt: {-achse:.2f} eV (Soll {W_CS})")

fig, ax = plt.subplots(figsize=(6.4, 3.4))
ax.plot(fs[nutzbar] / 1e14, u_mess, "ko", ms=6, label="Messpunkte")
ff = np.linspace(fs.min(), fs.max(), 50)
ax.plot(ff / 1e14, steig * ff + achse, "C0-", lw=1.2,
        label="Fit: Steigung = h/e")
ax.axhline(0, color="0.8", lw=0.6)
ax.set_xlabel("Frequenz [10¹⁴ Hz]")
ax.set_ylabel("Stoppspannung [V]")
ax.set_title("die Steigung dieser Geraden ist eine Naturkonstante")
ax.legend(fontsize=9)
plt.show()
```

```
ideale Stoppspannungen [V]: [1.257 0.921 0.704 0.131 0.005] —
nutzbar: [ True True True True False]
h aus dem Fit: 6.539e-34 Js (Abweichung 1.3 %)
W aus dem Achsenabschnitt: 2.10 eV (Soll 2.14)
```

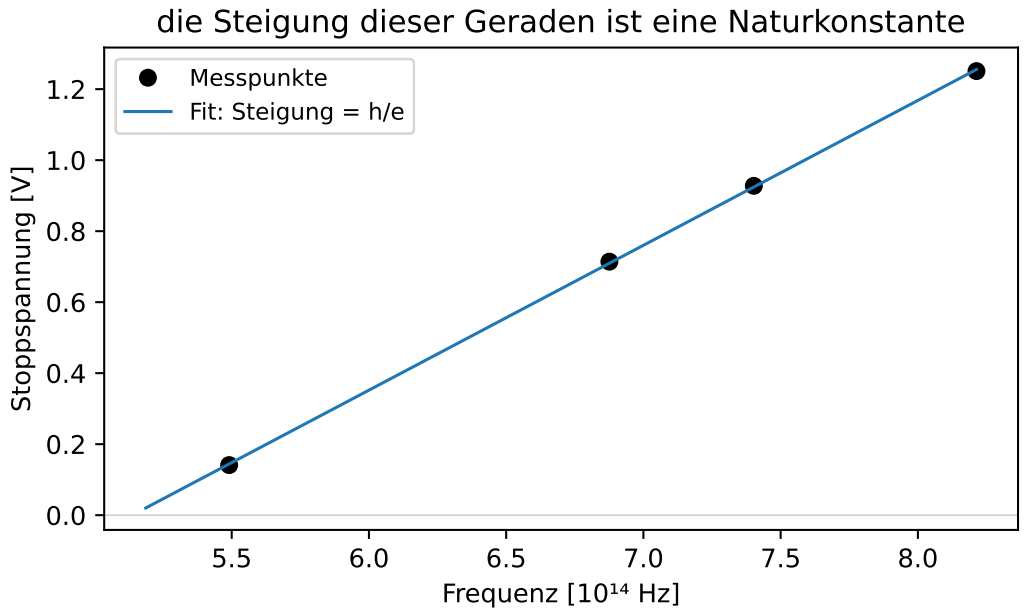


Abbildung 30.4: Millikans Auswertung im Nachbau: Stoppspannung gegen Frequenz für vier Quecksilber-Linien. Die gelbe 578-nm-Linie (2,145 eV) liegt praktisch AUF der Cäsium-Schwelle — ihre Stoppspannung von 5 mV versinkt im 10-mV-Messrauschen und fällt aus dem Fit. Die gefittete Gerade liefert h aus ihrer Steigung auf gut ein Prozent und die Austrittsarbeit aus dem Achsenabschnitt; ihre Nullstelle ist die Grenzfrequenz.

Die gelbe 578-nm-Linie liegt mit $2{,}145$ eV praktisch exakt auf der Cäsium-Schwelle (kein Zufall: Die *Grenzwellenlänge* war 579 nm) — die Handvoll Millivolt Stoppspannung, die sie liefert, ist vom Messrauschen nicht zu unterscheiden, und ein Punkt mit Rauschen statt Information würde den Fit nur verschmutzen. Aus den vier übrigen Punkten kommt h auf gut ein Prozent zurück. Millikan brauchte zehn Jahre Präzisionsarbeit für denselben Plot — und nannte Einsteins Deutung, die er damit bestätigte, noch im Ergebnis-Papier „reckless".

Ü 30.3 — Welcher-Weg als Monte-Carlo (Übertragen). Baue das Doppelspalt-Monte-Carlo so um, dass für jedes Photon zuerst per Münzwurf ein Spalt gewählt und dann der Treffer aus der *Einzelspalt*-Verteilung dieses Spalts gezogen wird (die beiden Einzelspalt-Muster sind gegeneinander um d verschoben — bei unseren Maßen fast deckungsgleich). Vorhersage: halb so viele Streifen, schwächere Streifen — oder keine? Miss

die Sichtbarkeit.

Musterlösung zu Ü 30.3

```
rng_ue3 = np.random.default_rng(23)
N = 200_000
# Einzelspalt-Verteilungen, je um ±d/2 verschoben
p_links = intensitaet((x + D_SPALT / 2) / np.hypot(x, L_SCHIRM),
                      beide=False)
p_rechts = intensitaet((x - D_SPALT / 2) / np.hypot(x, L_SCHIRM),
                       beide=False)
p_links, p_rechts = p_links / p_links.sum(), p_rechts / p_rechts.sum()

muenze = rng_ue3.integers(0, 2, N)     # rng.integers: ganzzahlig 0/1
treffer = np.where(muenze == 0,
                   rng_ue3.choice(len(x), size=N, p=p_links),
                   rng_ue3.choice(len(x), size=N, p=p_rechts))
hist = np.bincount(treffer, minlength=len(x)) / N

print(f"Sichtbarkeit Welcher-Weg-MC: "
      f"{sichtbarkeit(hist):.3f}  (beide offen: 1,000)")
```

```
Sichtbarkeit Welcher-Weg-MC: 0.141  (beide offen: 1,000)
```

Keine Streifen — die Sichtbarkeit fällt auf das Niveau des Zählrauschens. Sobald *im Experiment festgelegt* ist (und sei es nur im Prinzip auslesbar), welchen Weg das Photon nahm, addieren sich Wahrscheinlichkeiten statt Amplituden. Es braucht dafür keinen „Beobachter" und keine Störung — der Münzwurf in unserem Code *ist* die Welcher-Weg-Information. Die moderne Fassung dieses Gedankens (Quantenradierer, Delayed Choice) steht im Kleingedruckten.

Ü 30.4 — Das STM rechnet sich selbst (Übertragen). Zwischen Metallspitze und Probe wirkt die Austrittsarbeit als Tunnelwand: $V - E \approx W \approx 4{,}5$ eV. (a) Berechne κ. (b) Um welchen Faktor ändert sich der Tunnelstrom ($\propto e^{-2\kappa d}$), wenn sich der Abstand um 0,1 nm ändert — etwa einen halben Atomdurchmesser? (c) Die Spitze ist nie perfekt: Ihr unterstes Atom ragt vielleicht 0,25 nm über die nächsten hinaus. Welchen Anteil des Gesamtstroms trägt es — und warum „sieht" das STM deshalb atomar scharf?

Musterlösung zu Ü 30.4

```
W_METALL = 4.5 * EV
kap = np.sqrt(2 * ME * W_METALL) / HQUER
print(f"(a) κ = {kap*1e-9:.1f} pro Nanometer")
print(f"(b) 0,1 nm: Faktor e^(2κ·0,1nm) = {np.exp(2*kap*0.1e-9):.1f}")
print(f"(c) 0,25 nm Vorsprung: Nachbar-Atome tragen nur "
      f"1/{np.exp(2*kap*0.25e-9):.0f} des Spitzenatom-Stroms")
```

```
(a) κ = 10.9 pro Nanometer
(b) 0,1 nm: Faktor e^(2κ·0,1nm) = 8.8
(c) 0,25 nm Vorsprung: Nachbar-Atome tragen nur 1/229 des Spitzenatom-Stroms
```

(a) $\kappa \approx 10{,}9$/nm — doppelt so steil wie im 1-eV-Beispiel des Kapitels. (b) Ein Zehntel Nanometer ändert den Strom um fast eine Größenordnung (Faktor 8,8) — darum regelt ein STM seine Höhe auf Pikometer genau, indem es den Strom konstant hält. (c) Schon 0,25 nm Vorsprung lassen das unterste Atom den 230-fachen Strom seiner Nachbarn tragen: Über 99 % des Signals fließen durch ein einziges Atom. Die „Linse" des STM ist keine Optik, sondern die Exponentialfunktion — dieselbe, die in Kapitel 17 den Lichtsaum hinter dem Prisma beschrieb.

Das Kleingedruckte

Taylor 1909, Tonomura 1989. Geoffrey Ingram Taylor — damals Student, später einer der großen Strömungsphysiker — schwächte das Licht einer Gasflamme mit Rußglas so weit ab, dass im Apparat im Mittel weniger als ein Photon unterwegs war, und belichtete eine Fotoplatte hinter einer Nadel monatelang: Die Beugungsstreifen blieben in voller Schärfe. Akira Tonomura wiederholte das Experiment 1989 mit einzelnen *Elektronen* im Elektronenmikroskop und filmte, wie aus einzelnen Einschlägen über Stunden das Interferenzmuster wächst — die Aufnahmen sind im Netz frei zugänglich und sehen exakt aus wie unsere **?@fig-treffer-Reihe**, Bild für Bild. Die *Physics World* kürte das Experiment 2002 zum „schönsten Experiment der Physik".

Einstein 1905, Millikan 1916. Einsteins Lichtquanten-Arbeit (für die er 1921 den Nobelpreis erhielt — nicht für die Relativitätstheorie) sagte die Stoppspannungs-Gerade quantitativ voraus. Robert Millikan hielt die Lichtquanten für Unsinn und baute ein Jahrzehnt lang immer bessere Apparaturen, um die Gerade zu *widerlegen* — er fand sie mit h auf ein halbes Prozent

genau bestätigt und beschrieb Einsteins Hypothese noch in der Veröffentlichung der Bestätigung als „reckless". Auch das ist Wissenschaft: Das Experiment gewinnt gegen die Überzeugung des Experimentators, und ein guter Messwert schlägt eine gute Meinung.

Was ein Photon nicht ist. Die Kugel-Vorstellung („kleines Lichtteilchen fliegt auf einer Bahn") führt sofort in Widersprüche — welchen Spalt nähme die Kugel? In der Quantenelektrodynamik ist das Photon eine *Anregungsstufe einer Feldmode*: Erst löst man die Maxwell-Gleichungen (das haben wir 29 Kapitel lang getan), die Lösungen heißen Moden — und dann kann jede Mode nur ganzzahlig viele Energiequanten hf tragen. „Teilchen" ist daran nur die Unteilbarkeit der Portion beim Zählen. Der harte experimentelle Beleg, dass Poisson-Würfeln nicht alles ist, heißt **Antibunching**: Ein einzelnes angeregtes Atom sendet sein Photon und braucht dann eine Pause — zwei Klicks unmittelbar nacheinander sind *unmöglich*, während jede klassische Welle beliebig dicht klicken ließe. Gemessen wird das über die Intensitäts-Korrelation $g^{(2)}(0) < 1$ (Hanbury Brown/Twiss-Aufbau); es ist der Goldstandard-Test für Einzelphotonen-Quellen der Quantenkryptografie.

Das Auge als Photonenzähler. Selig Hecht, Simon Shlaer und Maurice Pirenne zeigten 1942 mit dunkeladaptierten Versuchspersonen: Für eine bewusste Lichtwahrnehmung genügen etwa fünf bis sieben Photonen auf der Netzhaut — einzelne Stäbchenzellen antworten nachweislich schon auf *ein* Photon. Das nächtliche „Grieseln" beim Sternegucken ist zum Teil echtes Schrotrauschen: Bei einem Stern an der Sehgrenze treffen nur einige hundert Photonen pro Sekunde die Pupille, und das $\sqrt{N}$-Gesetz dieses Kapitels wird unmittelbar erlebbar.

Welcher Weg, modern. Die Komplementarität braucht keinen störenden Messfühler: Im *Quantenradierer*- Experiment wird die Weg-Information in die Polarisation des Photons geschrieben (eine λ/4-Platte pro Spalt — unsere Kapitel-28-Bekannte!) — die Streifen verschwinden, obwohl niemand misst. Löscht man die Markierung hinter dem Spalt wieder, kehren die Streifen zurück; in der *Delayed-Choice*-Variante fällt diese Entscheidung erst, nachdem das Photon die Spalte längst passiert hat. Die Buchhaltung stimmt trotzdem immer: Sichtbarkeit der Streifen und Verfügbarkeit der Weg-Information sind komplementär — je mehr vom einen, desto weniger vom anderen, quantitativ als Ungleichung fassbar.

31 Inverses Design: der Computer als Entwerfer

In Kapitel 24 haben wir eine Antenne entworfen — und zwar *wir*: Wir kannten die Faustregeln (kurz führt, lang wirft zurück), wir haben Parameter durchgefahren, nachgestimmt, quergerechnet. Der Computer war dabei unser Messlabor, aber der Entwurf kam aus unserem Kopf. Dieses letzte Kapitel dreht den Spieß um. Wir sagen dem Computer nur noch, **was** das Bauteil können soll — „reflektiere diese Frequenz so stark wie möglich", „lenke das Licht dorthin" — und lassen ihn selbst herausfinden, **wie** es aussehen muss. Das heißt **inverses Design**: Statt von der Struktur zur Wirkung zu rechnen (vorwärts, wie in 30 Kapiteln), suchen wir die Struktur zur gewünschten Wirkung.

Das ist keine Spielerei, sondern der Stand der Technik: Die kompaktesten optischen Strahlteiler, Wellenlängen- Sortierer und Modenwandler der Photonik stammen heute aus Optimierungs-Schleifen — winzige Pixel-Labyrinthe, auf die kein Mensch gekommen wäre, ein Vielfaches kleiner als ihre von Hand entworfenen Vorgänger. In diesem Kapitel bauen wir die Idee mit unseren eigenen Werkzeugen nach: Ein Optimierer aus SciPy wird den **Bragg-Spiegel aus Kapitel 26 neu erfinden** (und dabei einen Fehler korrigieren, den wir ihm absichtlich unterschieben), ein simples Probier-Verfahren wird in unserem Kapitel-9-FDTD-Gitter einen **Strahl-Umlenker aus 36 Pixeln** zusammenpuzzeln — und am Ende steht, wie es sich für ein letztes Kapitel gehört, ein Schlusswort.

Lernziele

Nach diesem Kapitel kannst du …

1. … den Unterschied zwischen **Vorwärts-Simulation** und **inversem Design** erklären und eine **Zielfunktion** formulieren,
2. … den **Fluch der Dimensionen** beziffern: warum Durchprobieren bei acht Parametern schon Jahrhunderte kostet und ein Optimierer mit rund tausend Auswertungen auskommt,

3. … ein Design-Problem mit `scipy.optimize` lösen und — wichtiger — das Ergebnis **physikalisch lesen** (der Optimierer entdeckt die λ/4-Regel, ohne sie zu kennen),
4. … erklären, warum **lokale Optima** Multistart erzwingen (die Zielfunktions-Landschaft hat viele Gipfel),
5. … ein **Topologie-Puzzle** per Hillclimbing lösen und weißt, warum invers entworfene Bauteile aussehen „wie gewachsen”,
6. … skizzieren, wie das **Adjoint-Verfahren** alle Ableitungen aus nur zwei Simulationsläufen gewinnt — und warum das die Topologie-Optimierung erst möglich macht.

31.1 Vorwärts, rückwärts, Zielfunktion

Alles, was wir in diesem Buch gebaut haben, war eine **Vorwärts-Maschine**: Struktur rein (Geometrie, Materialien), Wirkung raus (Reflexion, Richtdiagramm, Fokus). Inverses Design braucht genau eine Zutat mehr — eine einzige Zahl, die sagt, wie *gut* eine Wirkung ist: die **Zielfunktion**. Aus „reflektiere möglichst stark bei f_0” wird $Z = R(f_0)$, aus „entspiegele das ganze sichtbare Band” wird $Z = -\overline{R}_{400-700\,\mathrm{nm}}$ (das Minus, weil Optimierer traditionell *minimieren*). Dann lautet die Aufgabe: Finde die Parameter, die Z optimal machen. Die Vorwärts-Maschine wird dabei vom Hauptdarsteller zum Hilfsarbeiter — sie wird hunderte Male aufgerufen, einmal pro Design-Kandidat.

Warum nicht einfach alles durchprobieren, wie beim Yagi-Sweep in Kapitel 24? Dort waren es *zwei* Parameter. Die Kosten des Durchprobierens wachsen aber **exponentiell** mit der Parameterzahl: acht Schichtdicken à zehn Testwerte sind schon 10^8 Simulationen, die 36 Ja/Nein-Pixel unseres Puzzles gleich $2^{36} \approx 7 \cdot 10^{10}$ — bei einer Zehntelsekunde pro Lauf über 200 Jahre (Übung 31.1 rechnet nach). Das ist der **Fluch der Dimensionen**, und er ist der Grund, warum inverses Design ein eigenes Handwerk ist: Ein Optimierer tastet sich mit *gezielten* Auswertungen durch den Riesenraum — wir werden gleich sehen, dass er mit etwa tausend Aufrufen auskommt, wo der Sweep 10^8 bräuchte.

31.2 Der Computer erfindet den Spiegel

Unser erstes Design-Problem ist mit Absicht eines, dessen Lösung wir längst kennen — so können wir dem Optimierer auf die Finger schauen. Aus Kapitel 26 wissen wir: Der beste Spiegel aus zwei abwechselnden Glassorten ist der

λ/4-Stapel — jede Schicht exakt eine viertel Wellenlänge dick (im jeweiligen Material), damit alle Teilreflexe in Phase aufeinanderfallen.

> Vorhersage (PRIMM)
>
> Wir geben dem Optimierer acht Schichten (Materialfolge hoch/niedrig fest: $n = 2{,}4$ und $1{,}5$, wie in Kapitel 26), aber **alle acht Dicken frei**, mit zufälligen Startwerten. Die Zielfunktion ist schlicht: maximiere $R(f_0)$. Was findet er — (a) irgendeinen krummen Kompromiss, schlechter als der λ/4-Stapel, (b) ziemlich genau den λ/4-Stapel, oder (c) etwas *Besseres* als unseren Plan?

Versuchsaufbau: Die Vorwärts-Maschine ist die Transfer-Matrix aus Kapitel 26 (acht Zeilen NumPy, unten als `tmm_r`). Der Optimierer ist `scipy.optimize.minimize` mit dem Nelder-Mead-Verfahren — ein robuster Kletterer, der keine Ableitungen braucht, sondern ein „Simplex" aus Testpunkten durch den Parameterraum wälzt. Weil solche Kletterer in Hügellandschaften am nächstgelegenen Gipfel hängen bleiben können, starten wir **sechsmal von zufälligen Punkten** (Multistart). Gemessen werden: das erreichte $R(f_0)$ jedes Starts, die Zahl der Funktionsauswertungen — und dann sehen wir uns die gefundenen Dicken genau an. Erfolgskriterium: Die Physik muss im Ergebnis wiedererkennbar sein.

```
import matplotlib.pyplot as plt
import numpy as np
# scipy.optimize.minimize: der Allzweck-Optimierer — sucht das Minimum
# einer Funktion vieler Variablen; method= waehlt das Verfahren
from scipy.optimize import minimize

N_HI, N_LO = 2.4, 1.5
F0 = 1.0                                # Designfrequenz (λ0 = 1)


def tmm_r(ns, ds, f, n_sub=1.0):
    """Reflexion |r|² eines Schichtstapels (Kap.-26-Transfer-Matrix)."""
    M = np.eye(2, dtype=complex)
    for n, d in zip(ns, ds):
        delta = 2 * np.pi * f * n * d
        M = M @ np.array([[np.cos(delta), -1j * np.sin(delta) / n],
                          [-1j * n * np.sin(delta), np.cos(delta)]])
    r = ((M[0, 0] + M[0, 1] * n_sub - M[1, 0] - M[1, 1] * n_sub)
         / (M[0, 0] + M[0, 1] * n_sub + M[1, 0] + M[1, 1] * n_sub))
    return abs(r) ** 2
```

```
NS_BRAGG = np.array([N_HI, N_LO] * 4)   # Materialfolge fest
zaehler = {"n": 0}                      # zaehlt die Vorwaerts-Laeufe


def kosten(ds):
    zaehler["n"] += 1
    return -tmm_r(NS_BRAGG, ds, F0)     # Minus: minimize soll maximieren


rng = np.random.default_rng(31)
laeufe = []
for start in range(6):
    d0 = rng.uniform(0.02, 0.4, 8)      # zufaellige Startdicken
    zaehler["n"] = 0
    res = minimize(kosten, d0, method="Nelder-Mead",
                   options={"maxiter": 4000, "xatol": 1e-6, "fatol": 1e-9})
    laeufe.append((-res.fun, res.x.copy(), zaehler["n"]))
    print(f"Start {start}: R = {-res.fun:.4f} "
          f"nach {zaehler['n']:4d} Auswertungen")
```

```
Start 0: R = 0.9595 nach 1500 Auswertungen
Start 1: R = 0.9595 nach 1144 Auswertungen
Start 2: R = 0.9595 nach  964 Auswertungen
Start 3: R = 0.9595 nach  798 Auswertungen
Start 4: R = 0.9595 nach  895 Auswertungen
Start 5: R = 0.9595 nach 1095 Auswertungen
```

Alle sechs Starts landen beim selben Wert — das Problem ist gutmütiger als befürchtet — und brauchen dafür jeweils um die tausend Auswertungen statt 10^8. Aber *was* haben sie gefunden? Schauen wir die Dicken des besten Laufs an. Eine Feinheit vorweg: In der Transfer-Matrix wirken zwei Dicken identisch, wenn sie sich um eine **halbe Wellenlänge im Material** unterscheiden (eine volle Extra-Hin-und-Her-Phase von 2π) — physikalisch vergleichbar sind also die Dicken *modulo* $\lambda/2n$:

```
r_best, d_best, _ = max(laeufe, key=lambda l: l[0])
soll = 1.0 / (4 * NS_BRAGG)              # die λ/4n-Regel aus Kap. 26
rest = np.mod(d_best, 1.0 / (2 * NS_BRAGG))
print("gefundene Dicken:", np.round(d_best, 4))
print("modulo λ/2n:     ", np.round(rest, 4))
print("λ/4n-Regel:      ", np.round(soll, 4))
```

```
gefundene Dicken: [ 0.3125  0.1667  0.3125  0.5    0.3125  0.1667  0.1042 -0.   ]
modulo λ/2n:      [0.1042 0.1667 0.1042 0.1667 0.1042 0.1667 0.1042 0.3333]
λ/4n-Regel:       [0.1042 0.1667 0.1042 0.1667 0.1042 0.1667 0.1042 0.1667]
```

Sieben der acht Dicken sind — modulo halber Wellenlänge — **exakt λ/4n**, auf vier Nachkommastellen. Der Optimierer hat die Kapitel-26-Regel wiederentdeckt, ohne je von Interferenz gehört zu haben; manche Schichten baut er als gleichwertige $3\lambda/4$-Fassung. Und die achte Schicht? Sie ist auf **null kollabiert**. Das ist kein Unfall, sondern Antwort (c) der Vorhersage:

```python
ds7 = [1 / (4 * n) for n in [N_HI, N_LO] * 3 + [N_HI]]
r7 = tmm_r([N_HI, N_LO] * 3 + [N_HI], ds7, F0)
r8 = tmm_r(NS_BRAGG, 1 / (4 * NS_BRAGG), F0)
print(f"unser 8-Schichten-Plan (endet niedrigbrechend): R = {r8:.4f}")
print(f"7 Schichten, endet hochbrechend:                R = {r7:.4f}")
print(f"der Optimierer fand:                            R = {r_best:.4f}")
```

```
unser 8-Schichten-Plan (endet niedrigbrechend): R = 0.9111
7 Schichten, endet hochbrechend:                R = 0.9595
der Optimierer fand:                            R = 0.9595
```

Unser Bauplan hatte einen versteckten Fehler: Die Folge [hoch, niedrig] × 4 endet mit einer *niedrigbrechenden* Schicht zur Luft hin — und die schwächt den Spiegel, statt ihn zu stärken (ein guter Stapel endet hochbrechend; das steht so in jedem Optik-Lehrbuch, aber wir hatten es ihm nicht verraten). Der Optimierer hat die nutzlose Schicht kurzerhand **wegoptimiert** und aus unserem schlechten 8-Schicht-Plan den besten 7-Schicht-Spiegel gemacht: $0{,}9595$ statt $0{,}9111$.

Naheliegende Vermutung: „Der Computer findet nur, was man hineinsteckt“

Warum sie naheliegt: Wir haben die Materialfolge, die Schichtzahl und die Zielfunktion vorgegeben — wo soll da Raum für eine eigene „Idee” des Optimierers sein? Er dreht doch nur an den Knöpfen, die wir ihm hinhalten.
Was stattdessen stimmt: Innerhalb des Suchraums findet der Optimierer auch das, woran *wir* nicht gedacht haben — hier die Korrektur unseres Endschicht-Fehlers, in der Forschungspraxis ganze Bauformen, auf die kein Entwerfer gekommen ist (die Pixel-Labyrinthe der invers entworfenen Photonik). Die ehrliche Fassung der Vermutung lautet anders: Der Computer findet nur, was die **Zielfunktion belohnt**. *Das* ist die wahre Verantwortung des Design-Ingenieurs — wer „maximiere R

bei genau f_0" bestellt, bekommt unter Umständen einen schmalbandigen, fertigungsempfindlichen Exoten (Übung 31.2 und 31.4 führen das vor). Die Kunst wandert vom Zeichenbrett in die Formulierung des Ziels.

31.3 Minimieren statt maximieren: der Antireflex-Belag

Dasselbe Werkzeug, umgekehrtes Ziel: Eine Glasoberfläche (Kapitel 6: $R = 4$ %) soll über das *ganze sichtbare Band* möglichst wenig reflektieren — die Entspiegelung jeder Brille und jedes Objektivs. Jetzt darf der Optimierer neben zwei Dicken auch zwei **Brechzahlen** wählen (im realistischen Bereich 1,35 bis 2,4):

```
N_GLAS = 1.5
fs_band = np.linspace(1 / 0.7, 1 / 0.4, 40)    # 400 … 700 nm


def r_mittel(ns, ds):
    return np.mean([tmm_r(ns, ds, f, n_sub=N_GLAS) for f in fs_band])


res_ar = minimize(lambda p: r_mittel(p[:2], p[2:]), [1.6, 1.9, 0.1, 0.1],
                  method="Nelder-Mead",
                  bounds=[(1.35, 2.4), (1.35, 2.4),
                          (0.02, 0.3), (0.02, 0.3)],
                  options={"maxiter": 3000})
print(f"nacktes Glas:            R-Mittel = {r_mittel([], []):.4f}")
print(f"klassische MgF2-Schicht: R-Mittel = "
      f"{r_mittel([1.38], [0.55 / (4 * 1.38)]):.4f}")
print(f"2-Schicht-Design:        R-Mittel = {res_ar.fun:.5f}")
print("Parameter (n1, n2, d1, d2):", np.round(res_ar.x, 3))
```

```
nacktes Glas:            R-Mittel = 0.0400
klassische MgF2-Schicht: R-Mittel = 0.0164
2-Schicht-Design:        R-Mittel = 0.00405
Parameter (n1, n2, d1, d2): [1.35  1.8   0.094 0.141]
```

Von 4 Prozent auf 0,4 Promille im Bandmittel — Faktor vier unter der klassischen Einfachschicht aus Magnesiumfluorid. Auch hier lohnt das Lesen

des Ergebnisses: Die erste Brechzahl klebt am unteren Rand des Erlaubten (1,35 — der Optimierer hätte gern noch weniger, aber so niedrigbrechende feste Materialien gibt es kaum; genau deshalb ist $n = 1{,}38$ von MgF_2 in der Optik so wertvoll). Reale Objektiv-Vergütungen treiben das Spiel mit sechs bis acht Schichten weiter — entworfen mit genau dieser Methode.

31.4 Die Landschaft: warum Multistart Pflicht ist

Dass alle sechs Bragg-Starts denselben Gipfel fanden, war Glück des Problems, nicht Eigenschaft des Verfahrens. Um zu sehen, womit Optimierer wirklich kämpfen, zeichnen wir die Zielfunktion einmal *vollständig* — das geht nur, wenn man sie auf zwei Parameter eindampft (genau deshalb ist diese Karte ein Luxus, den es ab drei Parametern nicht mehr gibt). Zwei Schichten, beide Dicken frei:

```
d1s = np.linspace(0.02, 0.45, 90)
d2s = np.linspace(0.02, 0.45, 90)
karte = np.array([[tmm_r([N_HI, N_LO], [d1, d2], F0) for d1 in d1s]
                  for d2 in d2s])
# scipy.ndimage.maximum_filter: ersetzt jeden Punkt durch das Maximum
# seiner Umgebung — wo der Wert gleich bleibt, sitzt ein lokaler Gipfel
from scipy.ndimage import maximum_filter
gipfel = (karte == maximum_filter(karte, size=9)) & (karte > 0.3)
gj, gi = np.where(gipfel)

fig, ax = plt.subplots(figsize=(6.2, 4.6))
im = ax.imshow(karte, origin="lower", cmap="viridis",
               extent=[d1s[0], d1s[-1], d2s[0], d2s[-1]])
ax.plot(d1s[gi], d2s[gj], "r^", ms=8, mfc="none",
        label=f"{gipfel.sum()} lokale Gipfel")
ax.set_xlabel("Dicke d1 (hochbrechend)")
ax.set_ylabel("Dicke d2 (niedrigbrechend)")
ax.set_title("schon zwei Parameter ergeben vier Gipfel")
ax.legend(fontsize=9)
plt.colorbar(im, ax=ax, label="R(f0)")
plt.show()
```

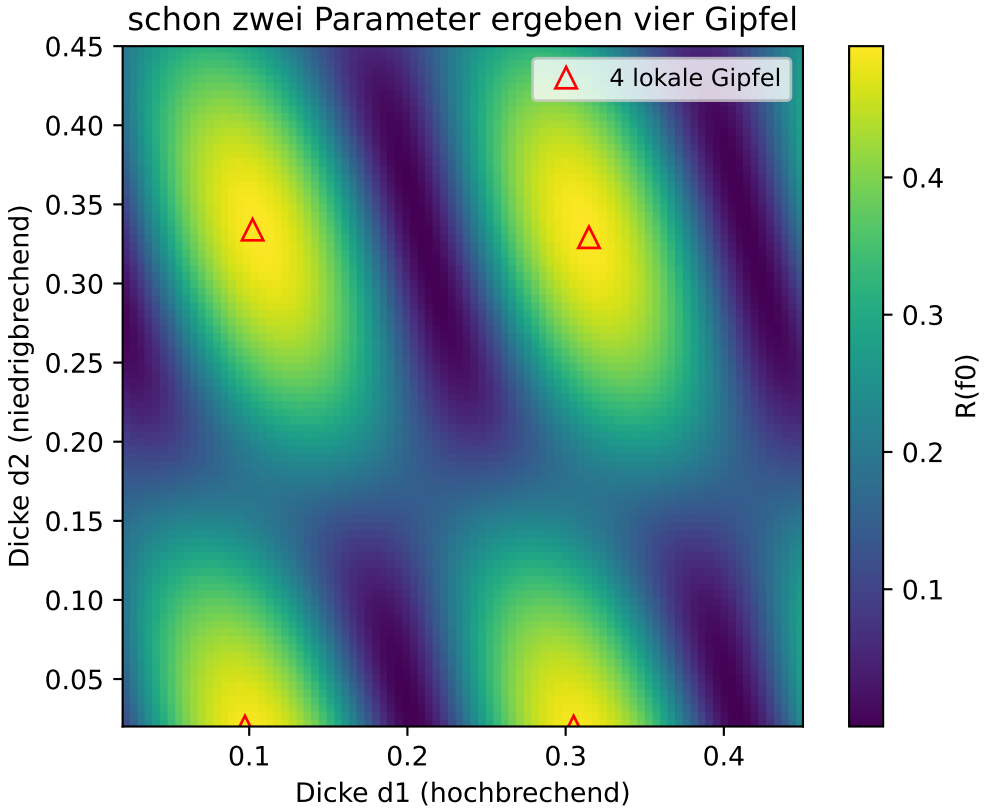


Abbildung 31.1: Die Zielfunktions-Landschaft $R(d_1, d_2)$ für einen Zwei-Schicht-Spiegel: vier getrennte Gipfel (Dreiecke), getrennt durch tiefe Täler. Ein Bergsteiger-Verfahren landet auf dem Gipfel, in dessen Einzugsgebiet es startet — nur Multistart (oder Vorwissen) findet verlässlich den besten. Und die Gipfel-Positionen erzählen schon die Kapitel-Pointe: Die hochbrechende Dicke d_1 steht auf $\lambda/4n$ (0,104) oder der gleichwertigen $3\lambda/4$-Fassung (0,313) — die niedrigbrechende Schicht dagegen stellt sich UNSICHTBAR ($d_2 \approx 0$ oder $\lambda/2n = 0{,}333$, eine volle Halbwellen-Schicht wirkt wie keine). Schon die Landschaft weiß: Eine niedrigbrechende Schicht zur Luft hin schadet dem Spiegel.

Vier Gipfel bei zwei Parametern — und die Gipfelzahl wächst mit jeder Dimension weiter. Ein lokales Verfahren (Nelder-Mead, Gradientenabstieg) klettert stur bergauf und endet auf dem Gipfel seines Startgebiets, der keineswegs der höchste sein muss. Die Standard-Gegenmittel hast du eben schon benutzt: **Multistart** (mehrfach zufällig starten, bestes Ergebnis nehmen) und **Vorwissen** (ein physikalisch motivierter Startpunkt — etwa „un-

gefähr λ/4" — startet meist im richtigen Tal).

31.5 Das Pixel-Puzzle: Topologie-Optimierung im Kleinen

Bisher hat der Optimierer an *Maßen* gedreht — die Bauform (ein Schichtstapel) stand fest. Die Königsklasse des inversen Designs geht einen Schritt weiter: Sie lässt auch die **Form** frei, indem sie den Bauraum in Pixel zerlegt und für jedes einzelne fragt: Material oder Luft? Das heißt **Topologie-Optimierung**, und wir bauen sie jetzt im Kleinformat — mit dem 2D-FDTD-Gitter aus Kapitel 9 als Vorwärts-Maschine, komplett selbst geschrieben.

Versuchsaufbau: Die Frage: Kann ein dummes Probier-Verfahren aus 36 Ja/Nein-Pixeln ein optisches Bauteil zusammensetzen? Die Bühne: ein 150 × 110-Gitter (TMz wie in Kapitel 9, σ-Rampen als Rand), links eine CW-Linienquelle ($\lambda = 20$ Zellen), in der Mitte die **Design-Region** — 6 × 6 Pixel à 4 × 4 Zellen, jedes entweder Luft ($\varepsilon = 1$) oder Material ($\varepsilon = 6$). Die Zielfunktion: die aufsummierte Leistung $\sum E_z^2$ an einem Zielpunkt, der **schräg versetzt** hinter der Region liegt — geradeaus durchfliegen gilt nicht, das Licht muss umgelenkt werden. Der Optimierer ist diesmal absichtlich der einfachste der Welt: **Bit-Flip-Bergsteigen** — wirf eine Münze, welcher Pixel kippt, behalte den Flip, wenn das Ziel steigt, sonst nimm ihn zurück; 250 Versuche. Gemessen wird gegen zwei Referenzen: die leere Region und das beste von 20 Zufallsmustern. Erfolgskriterium: deutlich besser als beide.

```
NX, NY = 150, 110
DT = 0.5
F_TR = 1 / 20.0
NT = 500
RAND = 10
X_Q = 18
PIX, NPX, NPY = 4, 6, 6
X0, Y0 = 60, 43                      # Ecke der Design-Region
ZIEL = (130, 88)                     # der Zielpunkt: schräg oben rechts
EPS_HI = 6.0

# Daempfungs-Rampe an allen Raendern (die σ-Rampe aus Kapitel 9)
sig = np.zeros((NX, NY))
for i in range(RAND):
    s = 0.35 * ((RAND - i) / RAND) ** 2
    sig[i, :] = np.maximum(sig[i, :], s)
    sig[NX - 1 - i, :] = np.maximum(sig[NX - 1 - i, :], s)
```

```
    sig[:, i] = np.maximum(sig[:, i], s)
    sig[:, NY - 1 - i] = np.maximum(sig[:, NY - 1 - i], s)
daempf = (1 - sig * DT / 2) / (1 + sig * DT / 2)


def bewerte(pixel, feld_zurueck=False):
    """Ein 2D-FDTD-Lauf für ein Pixelmuster → Leistung am Zielpunkt."""
    eps = np.ones((NX, NY))
    for i in range(NPX):
        for j in range(NPY):
            if pixel[i, j]:
                eps[X0 + i * PIX:X0 + (i + 1) * PIX,
                    Y0 + j * PIX:Y0 + (j + 1) * PIX] = EPS_HI
    ez = np.zeros((NX, NY))
    bx = np.zeros((NX, NY - 1))
    by = np.zeros((NX - 1, NY))
    summe = 0.0
    betrag = np.zeros((NX, NY))
    for n in range(NT):
        rot_b = np.zeros((NX, NY))
        rot_b[1:-1, 1:-1] = ((by[1:, 1:-1] - by[:-1, 1:-1])
                             - (bx[1:-1, 1:] - bx[1:-1, :-1]))
        ez = daempf * ez + DT * rot_b / eps
        ez[X_Q, RAND + 5:NY - RAND - 5] += np.sin(2 * np.pi * F_TR * n * DT)
        bx -= DT * np.diff(ez, axis=1)
        by += DT * np.diff(ez, axis=0)
        if n > 150:                          # erst einschwingen lassen
            summe += ez[ZIEL] ** 2
            if feld_zurueck:
                betrag = np.maximum(betrag, np.abs(ez))
    return (summe, betrag) if feld_zurueck else summe


leer = bewerte(np.zeros((NPX, NPY), bool))
pixel = np.zeros((NPX, NPY), bool)
wert = leer
verlauf = [wert]
flips = []                                   # Protokoll fuer den Film
for it in range(250):
    i, j = rng.integers(0, NPX), rng.integers(0, NPY)
    pixel[i, j] = ~pixel[i, j]
    neu = bewerte(pixel)
    if neu > wert:
```

```
        wert = neu
        flips.append((it, pixel.copy(), wert))
    else:
        pixel[i, j] = ~pixel[i, j]           # Flip zuruecknehmen
    verlauf.append(wert)

# Referenz: 20 zufaellige Muster
# rng.random: gleichverteilte Zahlen in [0,1) — mit "> 0.5" wird daraus
# ein zufaelliges Ja/Nein-Muster
zufall_best = max(bewerte(rng.random((NPX, NPY)) > 0.5) for _ in range(20))
print(f"leer:           {leer:6.1f}")
print(f"Zufall (beste von 20): {zufall_best:6.1f}")
print(f"Bergsteigen:    {wert:6.1f}  → Faktor {wert / leer:.1f} gegen leer")
```

```
leer:             39.7
Zufall (beste von 20):  133.7
Bergsteigen:     235.8  → Faktor 5.9 gegen leer
```

250 dumme Münzwürfe schlagen das beste von zwanzig Zufallsmustern deutlich — nicht weil ein einzelner Versuch klug wäre, sondern weil **behalten und verwerfen** Information akkumuliert. So sieht das gefundene Bauteil bei der Arbeit aus:

```
_, feldkarte = bewerte(pixel, feld_zurueck=True)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9.2, 3.6),
                               width_ratios=[1, 1.3])
ax1.plot(verlauf, "C0-", lw=1.3)
ax1.axhline(leer, color="0.7", lw=0.9, label="leere Region")
ax1.axhline(zufall_best, color="C3", ls="--", lw=0.9,
            label="bestes Zufallsmuster")
ax1.set_xlabel("Bit-Flip-Versuche")
ax1.set_ylabel("Leistung am Zielpunkt")
ax1.legend(fontsize=8)
ax1.set_title(f"Faktor {wert / leer:.1f} in 250 Versuchen")

ax2.imshow(feldkarte.T, origin="lower", cmap="inferno",
           vmax=0.7 * feldkarte.max())
# plt.Rectangle + add_patch: zeichnet ein Rechteck als Overlay
ax2.add_patch(plt.Rectangle((X0, Y0), NPX * PIX, NPY * PIX, fill=False,
                            edgecolor="w", lw=0.9))
for i in range(NPX):
    for j in range(NPY):
```

```
        if pixel[i, j]:
            ax2.add_patch(plt.Rectangle((X0 + i * PIX, Y0 + j * PIX),
                                        PIX, PIX, facecolor="none",
                                        edgecolor="c", lw=0.7))
ax2.plot(*ZIEL, "w+", ms=11)
ax2.set_xlabel("x")
ax2.set_ylabel("y")
ax2.set_title("der gefundene Umlenker (max |Ez|)")
plt.tight_layout()
plt.show()
```

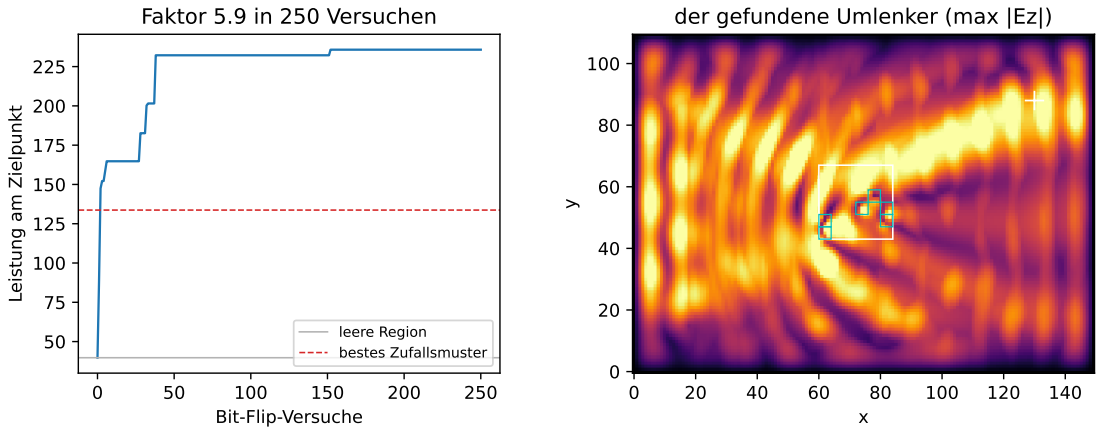


Abbildung 31.2: Das Pixel-Puzzle: Links der Optimierungs-Verlauf — jede Treppenstufe ist ein behaltener Bit-Flip; die graue Linie ist die leere Design-Region, die rote das beste von 20 Zufallsmustern. Rechts das gefundene Bauteil bei der Arbeit (Karte: max |Ez|): Die Welle kommt von links, die cyan umrandeten Pixel sind Material ($\varepsilon = 6$), und am weißen Kreuz — dem Zielpunkt, schräg versetzt — sammelt sich das Licht. Das Muster wirkt zusammengewürfelt, lenkt aber nachweislich Faktor 6 mehr Leistung dorthin als die leere Region: invers entworfene Bauteile sehen aus „wie gewachsen“, nicht wie konstruiert.

Halte einen Moment inne bei diesem Bild: Niemand hat dieses Muster entworfen. Es ist *entstanden* — aus einer Zielvorgabe, einer Vorwärts-Maschine und 250 Münzwürfen. Es nutzt vermutlich eine Mischung aus Brechung, Streuung und Interferenz, die sich gegen unsere Intuition sperrt — und genau das ist typisch: Invers entworfene Photonik sieht aus wie Flechtwerk oder Korallen, nicht wie Linsen und Spiegel. Lesbarkeit war nie das Ziel;

nur die Zielfunktion zählt.

31.6 Gradienten, Adjoint und die große Welt

Unser Bergsteigen ist ehrlich, aber verschwenderisch: Jeder einzelne Münzwurf kostet eine volle Simulation, und bei den Millionen Pixeln echter Topologie-Optimierung ist das hoffnungslos. Die Profis gehen mit **Gradienten** bergauf — sie wissen für *jeden* Pixel gleichzeitig, ob mehr oder weniger Material das Ziel verbessert. Naiv gemessen (jeden Parameter einzeln anstupsen, Differenzenquotient) kostet der Gradient $N + 1$ Simulationen pro Schritt — bei N Pixeln wieder hoffnungslos.

Der Ausweg ist ein Satz, der zu den elegantesten der numerischen Physik gehört: das **Adjoint-Verfahren**. Es liefert den vollständigen Gradienten — alle N Ableitungen — aus genau **zwei** Simulationen: dem normalen Vorwärts-Lauf und einem zweiten Lauf, bei dem eine Ersatzquelle *am Zielpunkt* sitzt und das Feld rückwärts durch dieselbe Struktur schickt. Aus dem Überlapp beider Felder an jedem Ort folgt, wie empfindlich das Ziel auf Material genau dort reagiert. Dass „rückwärts durch dieselbe Struktur" überhaupt wohldefiniert ist, verdankt sich der Zeitumkehr-Symmetrie der Maxwell-Gleichungen aus Kapitel 20 — die dort eine philosophische Pointe war und hier zum Arbeitstier wird. Zwei Läufe statt $N + 1$: Erst dieser Faktor macht Design-Räume mit Millionen Freiheitsgraden begehbar, und jedes ernsthafte Photonik-Designwerkzeug (auch Meep bringt ein Adjoint-Modul mit, siehe Kleingedrucktes) hat ihn eingebaut.

Damit ist die Landkarte komplett: **Zielfunktion** formulieren, **Vorwärts-Maschine** wählen (TMM, FDTD, NEC — was zur Physik passt), **Optimierer** dazu (gradientenfrei für wenige Parameter, Adjoint-Gradienten für viele), **Multistart** gegen lokale Gipfel, und am Ende — das zeigt Übung 31.4 — ein **Robustheits-Check** gegen Fertigungstoleranzen, denn der nominell beste Entwurf ist nicht immer der gutmütigste. Der Mensch ist dabei nicht arbeitslos geworden; seine Arbeit ist gewandert: vom Zeichnen der Struktur zum Formulieren des Ziels und zum kritischen Lesen des Ergebnisses. Beides erfordert genau das, was dieses Buch aufgebaut hat — physikalisches Urteilsvermögen.

31.7 Das Kapitel-Programm

`programme/kap31/kap31_inverses_design.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die Bragg-Erfindung (sechs Starts, λ/4-Regel

modulo λ/2n, kollabierte Endschicht, 7-gegen-8-Schichten-Kontrolle), das Antireflex-Design, die Landschafts-Karte mit ihren vier Gipfeln, das Pixel-Puzzle (Faktor > 4 gegen leer, besser als 20 Zufallsmuster) und den Robustheits-Check aus Übung 31.4. Reines NumPy/SciPy mit fester Zufalls-Saat; Laufzeit etwa eine Minute.

Merkkasten

- **Inverses Design:** Ziel statt Struktur vorgeben — eine Zielfunktion macht aus „gut" eine Zahl, der Optimierer ruft die Vorwärts-Maschine als Unterprogramm.
- **Fluch der Dimensionen:** Sweep-Kosten wachsen exponentiell (10^8 für 8 Parameter à 10 Werte); Optimierer kommen mit ~10^3 gezielten Auswertungen aus.
- **Ergebnisse lesen:** Der Optimierer entdeckt physikalische Regeln (λ/4) und Plan-Fehler (nutzlose Endschicht) — aber er belohnt nur, was die Zielfunktion bestellt. Die Verantwortung wandert in die Ziel-Formulierung.
- **Lokale Gipfel:** Schon zwei Parameter ergeben vier Maxima → Multistart oder physikalisch motivierte Startpunkte.
- **Topologie-Optimierung:** Bauraum in Pixel zerlegen; selbst Bit-Flip-Bergsteigen findet funktionierende Bauteile („wie gewachsen"). Skalierbar wird das erst durch das **Adjoint-Verfahren**: alle N Ableitungen aus zwei Läufen (vorwärts + Ersatzquelle am Ziel rückwärts — die Kapitel-20-Zeitumkehr als Werkzeug).
- **Robustheit:** Designs auf flachen Maxima (Bragg) verzeihen Fertigungsfehler; Designs auf Nullstellen (Antireflex) leben von der Präzision — nominal bester ≠ bester.

Roter Faden

Das letzte Kapitel erntet das ganze Buch: Die **Transfer-Matrix aus Kapitel 26** und das **2D-Yee-Gitter aus Kapitel 9** wurden als Vorwärts-Maschinen eingespannt, der **Zufallsgenerator aus Kapitel 10/30** wurde zum Optimierer, die **λ/4-Physik aus Kapitel 22/26** kam als *Prüfstein* zurück (der Optimierer musste sie wiederfinden), die **Zeitumkehr aus Kapitel 20** wurde im Adjoint-Verfahren zum Rechenwerkzeug, und der **Yagi-Entwurf von Hand aus Kapitel 24** bekam sein Gegenstück: den Entwurf aus der Zielfunktion. Damit schließt sich der Kreis dieses Buchs — vom Sehen der Felder über das Verstehen und Nachbauen bis zu dem Punkt, an dem der Computer selbst zu entwerfen beginnt und unsere Aufgabe das Urteilen wird.

Übungen

Ü 31.1 — Der Fluch, beziffert (Verstehen). (a) Unser Pixel-Puzzle hat 36 binäre Pixel und braucht 0,1 s pro Bewertung. Wie lange dauert das vollständige Durchprobieren aller Muster? (b) Wie viele Bewertungen hat unser Bergsteigen gebraucht — und welchen Anteil des Suchraums hat es damit gesehen? (c) Acht Schichtdicken à 100 Testwerte, 1 ms pro TMM-Lauf: Sweep-Dauer?

Musterlösung zu Ü 31.1

```
sekunden = 2.0**36 * 0.1
print(f"(a) 2^36 Muster × 0,1 s = {sekunden:.2g} s "
      f"= {sekunden / 3.15e7:.0f} Jahre")
print(f"(b) 250 Bewertungen = {250 / 2.0**36:.1e} des Suchraums")
print(f"(c) 100^8 × 1 ms = {100.0**8 * 1e-3 / 3.15e7:.1e} Jahre")
```

```
(a) 2^36 Muster × 0,1 s = 6.9e+09 s = 218 Jahre
(b) 250 Bewertungen = 3.6e-09 des Suchraums
(c) 100^8 × 1 ms = 3.2e+05 Jahre
```

(a) Rund 220 Jahre — für ein Spielzeugproblem mit 36 Pixeln. (b) Das Bergsteigen sah vier *Milliardstel* des Suchraums und fand trotzdem ein Bauteil mit Faktor 6 — Optimierung ist keine Abkürzung durchs Probieren, sondern ein anderes Spiel. (c) $3 \cdot 10^5$ Jahre: Der „harmlose" Acht-Parameter-Sweep ist längst jenseits jeder Rechenzeit. Genau deshalb war der Kapitel-24-Sweep auf zwei Parameter beschränkt — und genau hier übernehmen Optimierer.

Ü 31.2 — Bestelle breit (Verändern). Der Kapitel-Spiegel wurde nur bei f_0 bestellt. Ändere die Zielfunktion auf das *mittlere* R über das Band $f = 0{,}85$ bis $1{,}15$ und optimiere erneut (Multistart!). (a) Wie ändern sich die gefundenen Dicken? (b) Vergleiche $R(f_0)$ und die Bandbreite beider Designs — was hat der Breitband-Spiegel bezahlt?

Musterlösung zu Ü 31.2

```
fs_breit = np.linspace(0.85, 1.15, 21)


def kosten_band(ds):
    return -np.mean([tmm_r(NS_BRAGG, ds, f) for f in fs_breit])


best_band = None
for s in range(4):
    res = minimize(kosten_band, rng.uniform(0.02, 0.4, 8),
                   method="Nelder-Mead", options={"maxiter": 4000})
    if best_band is None or res.fun < best_band.fun:
        best_band = res
d_band = best_band.x
print(f"(a) Banddicken mod λ/2n: "
      f"{np.round(np.mod(d_band, 1 / (2 * NS_BRAGG)), 3)}")
print(f"    (λ/4n wäre:          {np.round(1 / (4 * NS_BRAGG), 3)})")
print(f"(b) R(f0): schmal {tmm_r(NS_BRAGG, d_best, F0):.4f}, "
      f"breit {tmm_r(NS_BRAGG, d_band, F0):.4f}")
print(f"    R-Mittel im Band: schmal "
      f"{-kosten_band(d_best):.4f}, breit {-best_band.fun:.4f}")

fs_plot = np.linspace(0.6, 1.4, 300)
fig, ax = plt.subplots(figsize=(6.6, 3.4))
ax.plot(fs_plot, [tmm_r(NS_BRAGG, d_best, f) for f in fs_plot], "C0-",
        lw=1.3, label="Ziel: nur f0")
ax.plot(fs_plot, [tmm_r(NS_BRAGG, d_band, f) for f in fs_plot], "C1-",
        lw=1.3, label="Ziel: Band 0,85–1,15")
ax.axvspan(0.85, 1.15, color="0.9")
ax.set_xlabel("Frequenz f")
ax.set_ylabel("R")
ax.set_title("die Zielfunktion ist die Bestellung")
ax.legend(fontsize=9)
plt.show()
```

```
(a) Banddicken mod λ/2n: [0.012 0.157 0.104 0.166 0.104 0.167 0.105 0.   ]
    (λ/4n wäre:          [0.104 0.167 0.104 0.167 0.104 0.167 0.104 0.167])
(b) R(f0): schmal 0.9595, breit 0.7988
    R-Mittel im Band: schmal 0.6591, breit 0.8221
```

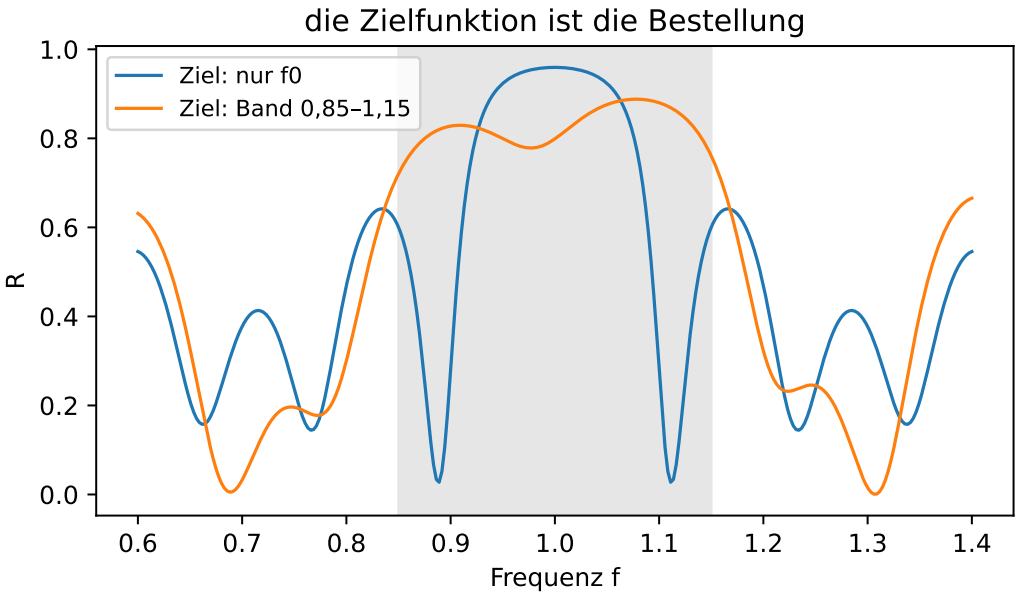


Abbildung 31.3: Schmal bestellt gegen breit bestellt: Das nur bei f_0 optimierte Design (blau) erreicht dort die höchste Reflexion (0,96), fällt aber außerhalb schnell ab. Das auf das Band 0,85–1,15 optimierte Design (orange, Band grau hinterlegt) hält die Reflexion über das ganze Band oben — und bezahlt dafür deutlich am Gipfel (0,80 bei f_0). Die Zielfunktion ist die Bestellung: Wer nur einen Punkt bestellt, bekommt einen Spezialisten.

(a) Die inneren Dicken bleiben nahe der λ/4-Familie, aber gegeneinander **verstimmt**, und die Randschichten scheren ganz aus (eine kollabiert wieder auf null) — der Optimierer spannt mit der Verstimmung das Band auf (dasselbe Prinzip treiben „gechirpte" Spiegel für ultrakurze Laserpulse auf die Spitze, deren Schichtdicken systematisch anwachsen). (b) Der Preis ist deutlich: Am Gipfel f_0 fällt R von $0{,}96$ auf $0{,}80$ — dafür steigt das Band-Mittel von $0{,}66$ auf $0{,}82$. Ein klassischer Zielkonflikt: Beide Designs sind „optimal" — für verschiedene Bestellungen.

Ü 31.3 — Der Strahlteiler (Übertragen). Ändere im Pixel-Puzzle die Zielfunktion: Statt eines Zielpunkts gibt es jetzt **zwei** (z. B. schräg oben *und* schräg unten, symmetrisch), und das Ziel ist das *Minimum* der beiden Leistungen (damit der Optimierer keinen Punkt bevorzugt). Lass das Bergsteigen laufen. Teilt das gefundene Muster den Strahl?

Musterlösung zu Ü 31.3

```
ZIEL_B = (130, 22)                  # zweiter Zielpunkt: schräg unten


def bewerte2(pixel):
    eps = np.ones((NX, NY))
    for i in range(NPX):
        for j in range(NPY):
            if pixel[i, j]:
                eps[X0 + i * PIX:X0 + (i + 1) * PIX,
                    Y0 + j * PIX:Y0 + (j + 1) * PIX] = EPS_HI
    ez = np.zeros((NX, NY))
    bx = np.zeros((NX, NY - 1))
    by = np.zeros((NX - 1, NY))
    s_a = s_b = 0.0
    for n in range(NT):
        rot_b = np.zeros((NX, NY))
        rot_b[1:-1, 1:-1] = ((by[1:, 1:-1] - by[:-1, 1:-1])
                             - (bx[1:-1, 1:] - bx[1:-1, :-1]))
        ez = daempf * ez + DT * rot_b / eps
        ez[X_Q, RAND + 5:NY - RAND - 5] += np.sin(2 * np.pi * F_TR * n * DT)
        bx -= DT * np.diff(ez, axis=1)
        by += DT * np.diff(ez, axis=0)
        if n > 150:
            s_a += ez[ZIEL] ** 2
            s_b += ez[ZIEL_B] ** 2
    return min(s_a, s_b), s_a, s_b


pix2 = np.zeros((NPX, NPY), bool)
w2 = bewerte2(pix2)[0]
for it in range(250):
    i, j = rng.integers(0, NPX), rng.integers(0, NPY)
    pix2[i, j] = ~pix2[i, j]
    neu = bewerte2(pix2)[0]
    if neu > w2:
        w2 = neu
    else:
        pix2[i, j] = ~pix2[i, j]
_, s_a, s_b = bewerte2(pix2)
print(f"Leistung oben {s_a:.1f} / unten {s_b:.1f} "
      f"(leer war oben {leer:.1f})")
print(f"min(beide) = {w2:.1f} — gegen leer: Faktor {w2 / leer:.1f}")
```

```
Leistung oben 138.2 / unten 139.8 (leer war oben 39.7)
min(beide) = 138.2 — gegen leer: Faktor 3.5
```

Das Min-Ziel zwingt den Optimierer zur Fairness: Er kann nicht einen Punkt mästen und den anderen verhungern lassen — heraus kommt ein Muster, das beide Zielpunkte über das Leer-Niveau hebt, ein **Strahlteiler aus Münzwürfen**. (Die erreichten Werte liegen unter denen des Einzelziel-Umlenkers — die Leistung wird ja geteilt.) Genau mit solchen Mehrziel-Funktionen — oft als Minimum oder gewichtete Summe — entstehen die realen Wellenlängen-Sortierer der Photonik.

Ü 31.4 — Der Robustheits-Check (Übertragen). Kein Fertigungsprozess trifft Dicken exakt. Verrausche (a) den gefundenen 7-Schicht-Bragg und (b) das Antireflex-Design aus diesem Kapitel mit ±5 % gleichverteiltem Dickenfehler (Monte-Carlo, 300 Züge wie in Kapitel 30) und vergleiche Mittelwert und Streuung der Güte mit dem Nominalwert. Welches Design ist gutmütiger — und warum?

Musterlösung zu Ü 31.4

```python
ns7 = [N_HI, N_LO] * 3 + [N_HI]
mc_bragg = [tmm_r(ns7, np.array(ds7) * rng.uniform(0.95, 1.05, 7), F0)
            for _ in range(300)]
p_ar = res_ar.x
mc_ar = [r_mittel(p_ar[:2], p_ar[2:] * rng.uniform(0.95, 1.05, 2))
         for _ in range(300)]
print(f"(a) Bragg:  nominal R = {r7:.4f}, "
      f"verrauscht {np.mean(mc_bragg):.4f} ± {np.std(mc_bragg):.4f}")
print(f"(b) AR:     nominal R = {res_ar.fun * 100:.3f} %, "
      f"verrauscht {np.mean(mc_ar) * 100:.3f} % "
      f"± {np.std(mc_ar) * 100:.3f} %")
```

```
(a) Bragg:  nominal R = 0.9595, verrauscht 0.9589 ± 0.0003
(b) AR:     nominal R = 0.405 %, verrauscht 0.435 % ± 0.021 %
```

Der Bragg verliert durch ±5 % Dickenfehler praktisch nichts (−0,0006 im Mittel): Sein Optimum ist ein **flaches Maximum** — die erste Ableitung ist dort null, kleine Fehler wirken nur quadratisch *nach unten von einem hohen Wert*. Das Antireflex-Design verschlechtert sich relativ um rund acht Prozent: Seine Güte ist eine **Nullstelle** der Reflexion, und von null aus gibt es nur eine Richtung — jede Abweichung kostet sofort. Die allgemeine Lektion: Wer invers entwirft, sollte die Toleranz-Statistik **in die Zielfunktion holen** (etwa das mittlere R über das

> Fertigungs-Ensemble minimieren statt das nominale) — die Profis nennen das *robust design*, und unsere Kapitel-30-Monte-Carlo-Technik ist das Werkzeug dazu.

Das Kleingedruckte

Meeps Adjoint-Modul und die Werkzeug-Welt. Meep bringt mit `meep.adjoint` eine vollwertige Topologie-Optimierung mit: Design-Regionen mit kontinuierlichem ε pro Pixel, automatische Gradienten über das Adjoint-Verfahren (intern braucht es die Pakete `autograd` und `nlopt`, die in unserer Umgebung nicht installiert sind — daher die Eigenbau-Route dieses Kapitels). Die kommerziellen Photonik-Werkzeuge (Lumerical, Tidy3D) haben Adjoint ebenso eingebaut wie die Mechanik-Welt ihre Topologie-Löser. Wer den Einstieg sucht: Die Meep-Dokumentation enthält durchgerechnete Adjoint-Beispiele (Modenwandler, Linsen, Demultiplexer) in genau der Python-Sprache, die du aus diesem Buch kennst.

Eine kurze Geschichte der gewachsenen Bauteile. Die Topologie-Optimierung stammt aus der Mechanik: Martin Bendsøe und Ole Sigmund entwickelten in den 1990ern die Verfahren, mit denen heute Flugzeug-Halterungen und Hüftimplantate „wachsen" — Material nur dort, wo Lastpfade es brauchen, knochenartig verästelt. In die Photonik kam die Methode um 2004 (Jensen/Sigmund) und wurde durch die Gruppe um Jelena Vučković in Stanford berühmt: Ihr invers entworfener Wellenlängen-Demultiplexer von 2015 — ein Quadrat von 2,8 µm Kantenlänge, das zwei Farben in zwei Wellenleiter sortiert — wurde zur Ikone des Felds und ist um Größenordnungen kleiner als jede klassische Lösung. Heute kommen Beschleuniger-auf-dem-Chip-Strukturen, Freiform-Metaoberflächen und ganze Kamera-Objektive aus solchen Schleifen.

Das Adjoint-Verfahren, eine Stufe genauer. Die saubere Herleitung läuft über Lagrange-Multiplikatoren: Man hängt die Maxwell-Gleichungen als Nebenbedingung an die Zielfunktion und erhält ein „adjungiertes" Feld, das von einer aus der Zielfunktion abgeleiteten Quelle erzeugt wird — sitzt das Ziel an einem Punkt, ist die Adjoint-Quelle ein Dipol an genau diesem Punkt. Der Gradient bezüglich $\varepsilon(\vec{r})$ ist dann (bis auf Konstanten) das Produkt aus Vorwärtsfeld und Adjoint-Feld am selben Ort. Dass der adjungierte Maxwell-Operator bis auf Zeitumkehr derselbe ist wie der ursprüngliche (Reziprozität!), macht den zweiten Lauf zu einer gewöhnlichen FDTD-Simulation — deshalb bekommt man das Verfahren in jeden bestehenden Löser eingebaut. Dieselbe Mathematik trägt übrigens das Training neuronaler Netze: „Backpropagation" ist das Adjoint-Verfahren der Lernmaschinen, und die

Zeitumkehr-Läufe aus Kapitel 20 waren näher an der modernen KI, als es damals aussah.

Fertigung als Nebenbedingung. Reale invers entworfene Bauteile müssen herstellbar sein: minimale Strukturgrößen (eine Lithografie-Maschine kann keine 10-nm-Inseln drucken), Verbote freischwebender Material-Inseln, Binärisierung (am Ende muss jeder Pixel Material *oder* Luft sein, nicht 37 % dazwischen). Die Praxis erzwingt das über Filter (das Design wird vor der Bewertung geglättet) und Projektionen (weiche Übergänge werden im Lauf der Optimierung zunehmend hart geschaltet) — und über genau die Robustheits-Ziele aus Übung 31.4, oft als Optimierung über drei Kopien gleichzeitig: nominal, überätzt, unterätzt.

Schlusswort: die Reise

Auf der ersten Seite dieses Buchs stand ein Pfeilbild — ein elektrisches Feld, das man ansehen kann. Dreißig Kapitel später hat ein Computer eigenständig ein optisches Bauteil entworfen, und dazwischen lag ein einziger durchgehender Weg: vier Gleichungen. Alles, was uns begegnet ist — die Stehwelle im 1D-Labor und der Regenbogen im Prisma, der Knick in der Antennenleitung und das Doppelbild im Kalkspat, das Tunneln durchs Verbotene und das Photon, das würfelt —, war Maxwell, nur immer wieder anders befragt. Wenn dieses Buch eine einzige Überzeugung hinterlässt, dann diese: Die Elektrodynamik ist kein Katalog von Phänomenen, sondern *ein* Gedanke mit unerschöpflichen Konsequenzen.

Das zweite, was bleiben soll, ist eine Arbeitsweise. Wir haben in jedem Kapitel dasselbe Ritual geübt: erst **vorhersagen**, dann **messen**, dann die **Abweichung ernst nehmen** — denn fast immer wohnte dort die eigentliche Physik (der Sockel an der Dipol-Speisung, die fehlenden 0,8 % in der Energiebilanz, die Etalon-Echos der Wellenplatte). Dazu die Handwerksregeln, die sich durch alle Werkstattkästen zogen: Traue keiner Simulation ohne Theorielinie im Bild, keiner einzelnen Auflösung, keinem Ergebnis, das du nicht auf einem zweiten Weg bestätigen kannst — und wenn ein etabliertes Problem zickt, lies nach, bevor du rätst. Diese Regeln sind nicht an Meep gebunden und nicht an die Elektrodynamik; sie sind das übertragbare Kapital dieses Buchs, und sie gelten in jedem Feld, in dem Rechnungen Wirklichkeit vorhersagen sollen.

Und der Eigenbau? Wir haben den Leapfrog-Kern in Kapitel 5 aus sechs Zeilen NumPy gebaut und hätten ab Kapitel 11 bequem alles Meep überlassen können. Dass wir es nie ganz getan haben, war Absicht: Jedes Mal, wenn etwas Rätselhaftes geschah — Felder explodierten, Phasen liefen rückwärts,

eine PML verstärkte statt zu dämpfen —, war es der Blick in den selbstgebauten Kern, der das Rätsel löste. Werkzeuge, deren Inneres man einmal selbst verdrahtet hat, bleiben durchschaubar, wenn sie sich seltsam verhalten. Das gilt für FDTD wie für alles andere, was du in deinem Werkzeugkasten trägst.

Wie weiter? Am besten mit einer eigenen Frage — der Playground neben diesem Buch ist genau dafür da, und jedes „was passiert, wenn …" ist ein legitimer Forschungsauftrag. Wer tiefer in die Theorie will, findet in Griffiths' *Introduction to Electrodynamics* den nächsten Schritt und in den *Feynman Lectures* (Band II, frei im Netz) die schönste Physik-Prosa, die es gibt; für die Numerik ist Taflove/Hagness die FDTD-Bibel, für photonische Kristalle Joannopoulos (frei im Netz), für Wellen und Antennen Orfanidis (ebenfalls frei). Und die Werkzeuge dieses Buchs — Meep, NEC, NumPy — sind quelloffen und werden von Gemeinschaften gepflegt, denen man beitreten kann: Fragen stellen, Beispiele teilen, Fehler melden. Wissenschaft ist ein Mannschaftssport.

James Clerk Maxwell hat seine Gleichungen 1865 veröffentlicht, zwischen Pferdekutschen und Gaslaternen, und niemand — er selbst eingeschlossen — konnte ahnen, dass darin Radio und Radar, Glasfaser und WLAN, Laser und Kernspintomograf schon enthalten waren. Richard Feynman urteilte hundert Jahre später, vom amerikanischen Bürgerkrieg werde neben dieser Entdeckung wenig übrig bleiben — „the most significant event of the 19th century". Die Gleichungen sind heute 160 Jahre alt und keinen Tag müde; du hast in diesem Buch gesehen, dass vier Zeilen Vektoranalysis genügen, um Licht zu brechen, zu speichern, rückwärts laufen zu lassen und von einem Computer verbauen zu lassen, der sie als Zielfunktion liest.

Was du damit baust, ist neu. Viel Freude dabei.

(Und wer nach dem Schlussakkord noch nicht gehen mag: Es gibt eine Zugabe. Kapitel 32 nimmt die Gleichungen mit in die Werkstatt — als Schall.)

32 Zugabe: Schall — dieselben Gleichungen in deiner Werkstatt

Das Buch ist zu Ende — das Schlusswort ist gesprochen. Aber wie bei einem guten Konzert gibt es eine Zugabe, und sie beginnt mit einem Geständnis: Du hast in 31 Kapiteln gar kein „Elektromagnetik-Programm" gebaut. Du hast einen **Wellenlöser** gebaut. Der Beweis dauert etwa zehn Zeilen: Wir nehmen den 2D-FDTD-Kern aus Kapitel 9, beschriften die Variablen um — aus dem Feld E_z wird der Schalldruck p, aus den Magnetfeldern wird die Luftbewegung — und plötzlich rechnet dasselbe Programm **Akustik**. Alles, was du gelernt hast, reist mit: Impedanz und Reflexion, Interferenz, Bandlücken, sogar das inverse Design aus Kapitel 31.

Warum sich diese Zugabe lohnt, ist schnell gesagt: Bei Schall im Kilohertz-Bereich liegen die Wellenlängen zwischen Zentimetern und Metern — genau im Arbeitsbereich von Bohrmaschine, CNC-Fräse und 3D-Drucker. Und das Messgerät besitzt du längst: Lautsprecher, Mikrofon und die FFT aus Kapitel 10. Schall ist damit das einzige Wellengebiet dieses Buchs, in dem du den vollen Kreis **vorhersagen → simulieren → bauen → selbst messen** zu Hause schließen kannst. Bei der Yagi aus Kapitel 24 musstest du der WLAN-Anzeige glauben; den phononischen Kristall, den wir gleich entwerfen — ein Stäbchengitter aus Rundholz, das einen ganzen Frequenzbereich des Hörschalls aussperrt —, kannst du bohren, besingen und sein Loch im Spektrum mit eigenen Ohren und eigenem Mikrofon nachweisen.

Lernziele

Nach diesem Kapitel kannst du …

1. … die **Akustik-Gleichungen** (Newton plus Massenerhaltung) als Strukturzwilling der Maxwell-Paare erkennen und die Übersetzungstabelle $E \leftrightarrow p$, $\varepsilon \leftrightarrow 1/\kappa$, $\mu \leftrightarrow \rho$ anwenden,
2. … den 2D-FDTD-Kern aus Kapitel 9 in wenigen Zeilen auf Schall **portieren** und weißt, wie ein schallharter Streuer ins Gitter kommt (und warum am harten Rand der Druck *nicht* flippt — anders als E am PEC),

3. … mit der **Impedanz** $Z = \rho c$ Reflexionen an Materialgrenzen berechnen (warum Holz für Schall ein besserer Spiegel ist als Silber für Licht — und wozu Ultraschall-Gel gut ist),
4. … die **Wellenlängen-Werkstatt-Landkarte** aufstellen: welche Frequenzen sich mit CNC, 3D-Druck, Holz und Aluminium bauen *und* mit Hobbymitteln messen lassen,
5. … einen **phononischen Kristall** entwerfen (Bandlücke, N-Reihen-Gesetz, Defekt-Kanal) und seinen Bauplan samt Messprotokoll ableiten,
6. … die Kapitel-31-Methode auf Schall **übertragen** (ein Bit-Flip-Optimierer puzzelt einen Schall-Umlenker zusammen).

32.1 Zwei Gleichungen, neu beschriftet

Schall in Luft ist eine winzige Störung: Der Luftdruck weicht um den **Schalldruck** p vom Ruhewert ab (bei Zimmerlautstärke um etwa ein Millionstel), und die Luft selbst bewegt sich mit der **Schnelle** $\vec{v}$ ein paar Mikrometer hin und her. Zwei Gesetze koppeln die beiden Größen. Erstens Newton: Ein Druck*gefälle* schiebt die Luft an,

$$\rho \frac{\partial \vec{v}}{\partial t} = -\nabla p.$$

Zweitens die Massenerhaltung: Wo Luft zusammenströmt (negative Divergenz der Schnelle), steigt der Druck,

$$\frac{\partial p}{\partial t} = -\kappa \, \nabla \cdot \vec{v},$$

mit der Dichte ρ und dem Kompressionsmodul κ (wie steif sich die Luft gegen Zusammendrücken wehrt). Zwei Größen, erste Zeitableitungen, jede treibt die andere über Raumableitungen — das ist exakt der Bauplan unseres 1D-FDTD-Paars aus Kapitel 6. Die Übersetzungstabelle:

Tabelle 32.1: Die Übersetzungstabelle. Die letzten beiden Zeilen sind die interessantesten — dazu gleich mehr.

Elektromagnetik	Akustik
E (Feld)	p (Schalldruck)
H bzw. B	$\vec{v}$ (Schnelle)
ε (Permittivität)	$1/\kappa$ (Kompressibilität)

Elektromagnetik	Akustik
μ (Permeabilität)	ρ (Dichte)
$c = 1/\sqrt{\varepsilon\mu}$	$c = \sqrt{\kappa/\rho}$
Wellenwiderstand $Z = \sqrt{\mu/\varepsilon}$	Impedanz $Z = \rho c$
PEC ($E = 0$)	offenes Rohrende ($p = 0$)
—	schallharte Wand ($v_\perp = 0$)

Ein Unterschied versteckt sich nicht in der Tabelle, sondern in der Geometrie: p ist eine **Zahl**, kein Vektor. Schall in Luft ist eine Longitudinalwelle — es gibt keine Polarisation, und alles aus Kapitel 4 und Kapitel 13 über Schwingungsrichtungen bleibt ohne Gegenstück. Dafür passt die skalare 2D-Welt, in der wir seit Kapitel 9 rechnen (E_z, H_x, H_y), strukturell perfekt: $(E_z, H_x, H_y) \to (p, v_x, v_y)$.

Die Impedanz-Zeile der Tabelle lohnt eine Handrechnung, denn an ihr hängt die ganze Werkstatt-Tauglichkeit. Die Spiegelformel aus Kapitel 6 gilt wörtlich weiter: $r = (Z_2 - Z_1)/(Z_2 + Z_1)$.

```
import numpy as np

C_LUFT = 343.0                  # m/s bei 20 Grad
Z_LUFT = 1.2 * C_LUFT           # rho = 1,2 kg/m^3
print(f"Luft: Z = {Z_LUFT:.0f} kg/(m^2 s)")
for name, rho, c in (("Wasser", 1000.0, 1480.0),
                     ("Holz (Buche, laengs)", 700.0, 4000.0),
                     ("Aluminium", 2700.0, 6320.0),
                     ("PLA (3D-Druck)", 1240.0, 2200.0)):
    Z = rho * c
    R = ((Z - Z_LUFT) / (Z + Z_LUFT)) ** 2
    print(f"{name}: Z = {Z:.2e} -> R an Luft = {R*100:.2f} %")
```

```
Luft: Z = 412 kg/(m^2 s)
Wasser: Z = 1.48e+06 -> R an Luft = 99.89 %
Holz (Buche, laengs): Z = 2.80e+06 -> R an Luft = 99.94 %
Aluminium: Z = 1.71e+07 -> R an Luft = 99.99 %
PLA (3D-Druck): Z = 2.73e+06 -> R an Luft = 99.94 %
```

Holz reflektiert 99,94 Prozent der Schallleistung, Aluminium 99,99 — zum Vergleich: Ein Silberspiegel reflektiert sichtbares Licht zu etwa 98 Prozent. **Für Schall ist ein Besenstiel ein besserer Spiegel, als es Silber für Licht ist.** Der Grund ist das Impedanzverhältnis: Luft zu Holz liegt bei fast

1:7000, Luft zu Glas (optisch) nur bei 1:1,5. Genau deshalb dürfen wir gleich alle festen Materialien — Holz, Aluminium, PLA — als **ideal schallhart** nähern: Die Luftbewegung an ihrer Oberfläche ist praktisch null, $v_\perp = 0$. Das ist für die Akustik, was der PEC für unsere Antennen war. Und es erklärt nebenbei, warum beim Ultraschall am Arzt ein Gel aufgetragen wird: Schon eine hauchdünne Luftschicht zwischen Schallkopf und Haut wäre eine 99,9-Prozent-Spiegelwand (Übung 32.1 rechnet nach).

32.2 Der Kern, portiert

Versuchsaufbau: Wir bauen den Kapitel-9-Kern als Akustik-Fassung neu auf — gleiche Bühne wie immer: ein Rechteckgebiet, eine σ-Dämpfungsrampe an allen Rändern (diesmal *dick*, dazu unten mehr), eine weiche Quelle, die sin-Schwingungen oder Gauß-Pulse in den Druck drückt, und „Mikrofone", die p an gewählten Punkten mitschreiben. Neu ist ein einziges Argument: ein Bool-Feld `hart`, das markiert, welche Zellen aus festem Material bestehen. Auf allen Kanten harter Zellen wird die Schnelle nach jedem Update auf null gesetzt — Luft strömt nicht in den Stab.

Die drei tragenden Zeilen, entzerrt — du erkennst jede aus Kapitel 5 und Kapitel 9 wieder, nur die Schilder sind neu:

- `vx -= DT * np.diff(p, axis=0)` — Newton. `np.diff` bildet die Nachbar-Differenz *pro Lücke*: p lebt auf den Zellen (Länge N), v_x auf den Kanten dazwischen (Länge $N-1$) — der vertraute Yee-Versatz, nur dass jetzt der *Druckunterschied zwischen zwei Zellen* die Luft in der Kante dazwischen anschiebt.
- `vx[m_vx] = 0.0` — die Materialregel. `m_vx` markiert jede Kante, die eine harte Zelle berührt (`hart[1:,:] | hart[:-1,:]` — *oder*-Verknüpfung der beiden Nachbarzellen jeder Kante).
- `p = daempf * p - DT * div` — Massenerhaltung mit Randdämpfung: Die Divergenz sammelt ein, wie viel Luft pro Zelle zusammenströmt; `daempf` ist die $(1-\sigma\Delta t/2)$- Rampe aus Kapitel 9, unverändert.

```
DT = 0.5                        # Courant S = 0,5 < 1/sqrt(2)


def daempfung(nx, ny, rand, s_max=0.3):
    """Sigma-Rampe an allen Raendern (kubisch und DICK)."""
    sig = np.zeros((nx, ny))
    for i in range(rand):
        s = s_max * ((rand - i) / rand) ** 3
```

```
        sig[i, :] = np.maximum(sig[i, :], s)
        sig[nx-1-i, :] = np.maximum(sig[nx-1-i, :], s)
        sig[:, i] = np.maximum(sig[:, i], s)
        sig[:, ny-1-i] = np.maximum(sig[:, ny-1-i], s)
    return (1 - sig * DT / 2) / (1 + sig * DT / 2)


def schall_lauf(hart, daempf, q_pos, f_tr, nt, tau=None, t0=0.0,
                mikros=(), schnappschuesse=()):
    """2D-Akustik-FDTD: p auf den Zellen, vx/vy auf den Kanten."""
    nx, ny = hart.shape
    m_vx = hart[1:, :] | hart[:-1, :]   # Kanten an harten Zellen
    m_vy = hart[:, 1:] | hart[:, :-1]
    p = np.zeros((nx, ny))
    vx = np.zeros((nx-1, ny))
    vy = np.zeros((nx, ny-1))
    reihen = np.zeros((nt, len(mikros)))
    bilder = {}
    for n in range(nt):
        # Newton: die Schnelle folgt dem Druckgefaelle (rho = 1)
        vx -= DT * np.diff(p, axis=0)
        vy -= DT * np.diff(p, axis=1)
        vx[m_vx] = 0.0              # Luft kommt nicht in den Stab
        vy[m_vy] = 0.0
        # Massenerhaltung: der Druck folgt der Kompression (kappa = 1)
        div = np.zeros((nx, ny))
        div[1:-1, 1:-1] = ((vx[1:, 1:-1] - vx[:-1, 1:-1])
                           + (vy[1:-1, 1:] - vy[1:-1, :-1]))
        p = daempf * p - DT * div
        t = n * DT
        anregung = np.sin(2 * np.pi * f_tr * t)
        if tau is not None:         # Gauss-Puls statt Dauerton
            anregung *= np.exp(-((t - t0) / tau) ** 2)
        p[q_pos] += anregung        # weiche Quelle (Kapitel 6)
        for k, (mi, mj) in enumerate(mikros):
            reihen[n, k] = p[mi, mj]
        if n in schnappschuesse:
            bilder[n] = p.copy()
    return reihen, p, bilder
```

Mehr ist es nicht — vergleiche mit dem TMz-Kern aus Kapitel 9: gleiche Schleifenstruktur, gleiche `np.diff`-Buchhaltung, gleiche Rampe. Bevor wir ihm trauen, zwei Proben. Die erste ist Pflicht seit Kapitel 10: *Läuft ein Puls*

mit der richtigen Geschwindigkeit? In unseren Einheiten (Zellgröße 1, $\rho = 1$, $\kappa = 1$) muss $c = \sqrt{\kappa/\rho} = 1$ herauskommen.

```
NXc, NYc = 420, 232
RANDc = 40
daempf_c = daempfung(NXc, NYc, RANDc)
fronten = {}
for nt_stop in (200, 380):
    _, p_end, _ = schall_lauf(np.zeros((NXc, NYc), bool), daempf_c,
                              (60, NYc // 2), 0.035, nt_stop,
                              tau=12.0, t0=60.0)
    profil = np.abs(p_end[60:, NYc // 2])
    fronten[nt_stop] = np.max(np.nonzero(profil
                                         > 0.02 * profil.max())[0])
c_mess = (fronten[380] - fronten[200]) / ((380 - 200) * DT)
print(f"Front-Differenzmessung: c = {c_mess:.3f} (Soll: 1)")
```

```
Front-Differenzmessung: c = 1.000 (Soll: 1)
```

Vorhersage (PRIMM)

Die zweite Probe ist die interessantere. In Kapitel 6 hast du gesehen, dass ein Puls am PEC-Spiegel **mit umgekehrtem Vorzeichen** zurückkommt (E muss am Leiter null sein). Gleich schicken wir einen Druckpuls gegen eine schallharte Wand — und an ein „offenes Rohrende", an dem wir $p = 0$ festhalten. Kommt das Echo jeweils aufrecht oder gespiegelt zurück? Notiere beide Antworten, bevor du weiterliest.

```
import matplotlib.pyplot as plt

N1, NT1 = 400, 700
p1 = np.exp(-((np.arange(N1) - 120) / 8.0) ** 2)
start = p1.copy()
v1 = np.zeros(N1 - 1)
hart1 = np.zeros(N1, bool)
hart1[300:] = True              # rechts: schallharte Wand
m_v1 = hart1[1:] | hart1[:-1]
for n in range(NT1):
    v1 -= DT * np.diff(p1)
    v1[m_v1] = 0.0
    p1[1:-1] -= DT * np.diff(v1)   # p[0] bleibt 0: offenes Ende
```

```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 4.6), sharex=True)
ax1.plot(start, "C0")
ax1.axvspan(300, N1, color="0.75")
ax1.set_ylabel("p (Start)")
ax1.set_ylim(-0.7, 1.1)
ax2.plot(p1, "C0")
ax2.axvspan(300, N1, color="0.75")
ax2.axhline(0.5, ls=":", color="C2", lw=0.8)
ax2.axhline(-0.5, ls=":", color="C3", lw=0.8)
ax2.annotate("von der harten Wand:\naufrecht (+0,5)", xy=(130, 0.5),
             xytext=(160, 0.75), arrowprops=dict(arrowstyle="->"),
             fontsize=9)
ax2.annotate("vom offenen Ende:\ngespiegelt (-0,5)", xy=(230, -0.5),
             xytext=(40, -0.55), arrowprops=dict(arrowstyle="->"),
             fontsize=9)
ax2.set_ylabel("p (nach den Echos)")
ax2.set_ylim(-0.7, 1.1)
ax2.set_xlabel("Zelle")
plt.tight_layout()
plt.show()
spitze = p1[:300].max()
print(f"Echo von der Wand: {spitze:+.3f}   "
      f"Echo vom offenen Ende: {p1[:300].min():+.3f}")
```

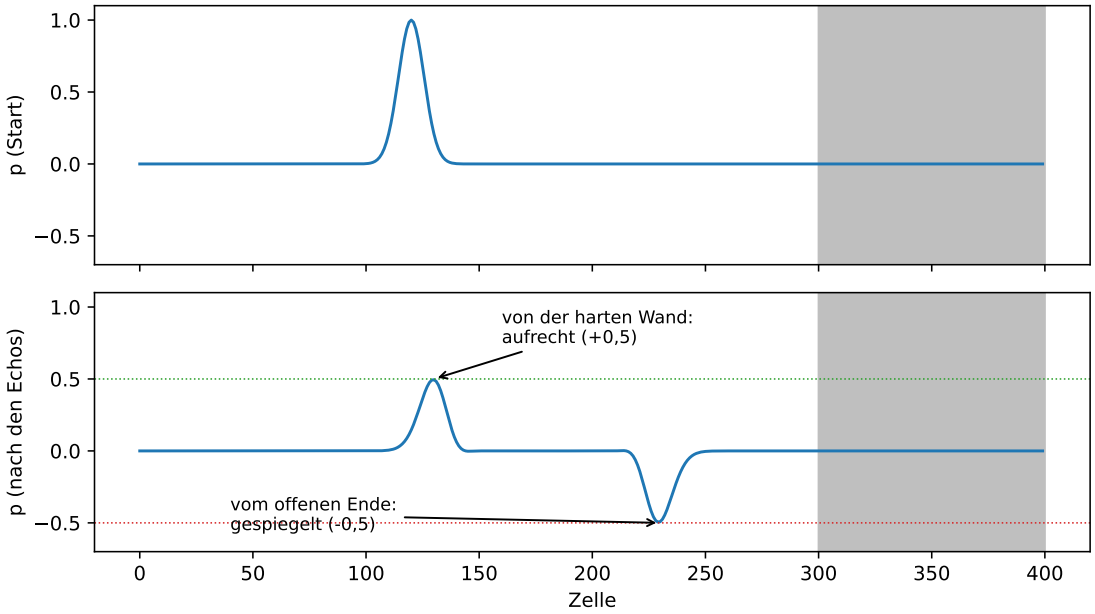


Abbildung 32.1: Die Vorzeichen-Probe in 1D: Ein Druckpuls (Höhe 1) zerfällt in zwei halbe Pulse (Kapitel 5). Der rechte trifft die schallharte Wand und kommt AUFRECHT zurück (+0,5): Am Stillstand der Luft staut sich der Druck — die Wand ist ein Druckbauch. Der linke trifft das offene Ende (p = 0 festgehalten) und kommt GESPIEGELT zurück (−0,5): Das offene Rohrende ist ein Druckknoten. Genau umgekehrt zum E-Feld am PEC — die Übersetzungstabelle vertauscht die Rollen der Randbedingungen.

```
Echo von der Wand: +0.495   Echo vom offenen Ende: -0.495
```

Der Druck flippt an der Wand **nicht** — die Wand ist ein Druck*bauch*, kein Knoten. Wer schon einmal bemerkt hat, wie laut die eigene Stimme in einer gefliesten Duschecke klingt, hat diesen Druckbauch gehört. Geflippt wird stattdessen am *offenen* Ende — dort, wo der Druck auf den Außendruck festgenagelt ist. Die Übersetzungstabelle vertauscht also die Rollen: Die harte Wand verhält sich wie der *magnetische* Spiegel der EM-Welt, das offene Rohr wie der PEC. Für Orgelpfeifen und Flaschen-Resonanzen ist das die halbe Physik — für uns ist es die Warnung, beim Übersetzen nicht nur Variablen, sondern auch Randbedingungen zu übersetzen.

32.3 Die Wellenlängen-Werkstatt-Landkarte

Naheliegende Vermutung: „Je kürzer die Wellenlänge, desto schwieriger wird das Bauen“

Warum sie naheliegt: Kleinere Strukturen verlangen feinere Werkzeuge — Mikrochips brauchen Reinräume, optische Gitter Nanometer-Maschinen. Also scheint die Faustregel zu gelten: Lange Wellen leicht, kurze Wellen schwer.
Was stattdessen stimmt: Für den Hobbybereich ist nicht die *Fertigung* der Engpass, sondern die **Messtechnik**. Eine 3-cm-Mikrowellenlinse (10 GHz) fräst jede CNC mühelos — aber das Messgerät dafür (ein Netzwerkanalysator) kostet mehr als die Fräse. Beim Schall dagegen liegt *beides* im Hobbybudget: Die Wellenlängen des Hörschalls (1 kHz: 34 cm, 10 kHz: 3,4 cm) treffen genau die Skala von Bohrmaschine, Fräse und 3D-Drucker, und das komplette Messlabor — Lautsprecher, Mikrofon, FFT-Software aus Kapitel 10 — kostet etwa so viel wie das Holz.

Konkret, erst die EM-Seite (mit $\lambda = c/f$):

Tabelle 32.2: EM-Wellen und die Werkstatt: ab 10 GHz wird die Fertigung einfach — und die Messung zum Engpass.

Frequenz	λ	bauen mit	messen mit
2,4 GHz (WLAN)	12,5 cm	Alu-Rohr, Draht (Kapitel 24)	WLAN-Anzeige (grob)
10 GHz	3 cm	CNC, 3D-Druck (PLA-Linsen)	VNA/SDR — **teuer**
100 GHz+	< 3 mm	Feinmechanik	Laborgerät
Licht	0,5 µm	gar nicht (kaufen)	Photodiode

Und die Schall-Seite ($c = 343$ m/s — das macht die Wellen bei gleicher Frequenz fast eine Million Mal kürzer):

Tabelle 32.3: Schall und die Werkstatt: vom Hörbereich bis ~15 kHz ist alles mit Bordmitteln baubar UND messbar.

Frequenz	λ	bauen mit	messen mit
100 Hz	3,4 m	Zimmerwände	Mikrofon

Frequenz	λ	bauen mit	messen mit
1 kHz	34 cm	Holz, Säge	Mikrofon
3 kHz	11 cm	**Rundholz-Gitter (dieses Kapitel)**	Mikrofon
10 kHz	3,4 cm	3D-Druck	Mikrofon (96 kHz)
40 kHz (Ultraschall)	8,6 mm	3D-Druck, fein	Transducer + schneller ADC

Die Zeile bei 3 kHz ist unser Ziel: Wellenlänge 11 cm, Strukturen von einigen Zentimetern — Baumarktware. Dass ein Mikrofon plus Soundkarte ein vollwertiges Messgerät ist, hast du in Kapitel 10 implizit schon gelernt: Ein Sweep durch alle Frequenzen, eine FFT der Aufnahme, und du hältst ein Spektrum in der Hand. Genau so werden wir gleich die Bandlücke nachweisen.

32.4 Der phononische Kristall

In Kapitel 26 hat ein periodischer Stapel durchsichtiger Schichten Licht perfekt gespiegelt — die Bandlücke. Das akustische Gegenstück heißt **phononischer Kristall**, und seine einfachste baubare Form ist ein Gitter aus runden Stäben in Luft: Holzdübel in ein Brett gesteckt, fertig. Jede Stabreihe reflektiert ein wenig; liegen die Reihen im richtigen Abstand a, addieren sich die Echos konstruktiv (die Bragg-Bedingung aus Kapitel 26: $\lambda \approx 2a$, also $f \approx c/2a$), und der Kristall sperrt.

Versuchsaufbau: Quadratgitter mit Gitterkonstante $a = 16$ Zellen, Stabradius $r = 0{,}35\,a$ (Füllfaktor $\pi \cdot 0{,}35^2 \approx 0{,}38$), 9 Säulen quer, 1 bis 5 Reihen in Laufrichtung. Links eine breite Quellzeile, die einen Gauß-Puls (Mittenfrequenz 0,035, Band etwa 0,012 bis 0,06) abstrahlt; rechts eine ganze **Mikrofon-Zeile**, deren Leistungsspektren wir summieren — ein einzelner Messpunkt säße womöglich in einem Beugungs-Minimum. Transmission $T(f)$ = Spektrum mit Kristall geteilt durch Spektrum ohne (zwei Läufe, wie bei jeder Referenzmessung seit Kapitel 12). Die Dämpfungsrampe ist mit 40 Zellen ungewohnt dick: Die längsten Wellen im Band messen 80 Zellen, und eine dünne Rampe wirft sie zurück ins Messgebiet (das Kleingedruckte erzählt, wie sich dieser Fehler bemerkbar machte).

Vorhersage (PRIMM)

Die Bragg-Peilung sagt: Lücke um $f \approx c/2a$, also $f \cdot a/c \approx 0{,}5$. Zwei Fragen vor dem Lauf: (1) Eine einzelne Stabreihe lässt bei dieser Frequenz noch knapp die Hälfte durch. Wie tief drücken fünf Reihen die Transmission — auf ein Viertel? Ein Zehntel? Weniger? (2) Was erwartest du *weit unterhalb* der Lücke, wo die Welle zehnmal länger ist als ein Stab dick?

```
A_GIT = 16
R_STAB = 0.35 * A_GIT
N_SAEULEN = 9
X_KRIST, X_Q, X_MIK = 160, 60, 330
NT_SPEK = 8000
ii_c, jj_c = np.meshgrid(np.arange(NXc), np.arange(NYc),
                         indexing="ij")


def kristall_bauen(n_reihen):
    hart = np.zeros((NXc, NYc), bool)
    y0 = (NYc - N_SAEULEN * A_GIT) / 2
    for r in range(n_reihen):
        for s in range(N_SAEULEN):
            cx = X_KRIST + (r + 0.5) * A_GIT
            cy = y0 + (s + 0.5) * A_GIT
            hart |= (ii_c - cx) ** 2 + (jj_c - cy) ** 2 < R_STAB ** 2
    return hart


quelle_c = (X_Q, slice(RANDc + 8, NYc - RANDc - 8))
mik_zeile = [(X_MIK, j) for j in range(RANDc + 12, NYc - RANDc - 12)]


def leistungsspektrum(hart):
    reihen, _, _ = schall_lauf(hart, daempf_c, quelle_c, 0.035,
                               NT_SPEK, tau=12.0, t0=60.0,
                               mikros=mik_zeile)
    return np.sum(np.abs(np.fft.rfft(reihen, axis=0)) ** 2, axis=1)


fs = np.fft.rfftfreq(NT_SPEK, DT)
spek_frei = leistungsspektrum(np.zeros((NXc, NYc), bool))
band = ((fs > 0.012) & (fs < 0.06)
```

```
        & (spek_frei > 0.01 * spek_frei.max()))
fb = fs[band]
T_leiter = {}
for n_r in (1, 2, 3, 5):
    T_leiter[n_r] = (leistungsspektrum(kristall_bauen(n_r))[band]
                     / spek_frei[band])

# Lueckenkanten der ersten Luecke (T < 0,05) suchen
T5 = T_leiter[5]
i_tief = np.argmin(np.where(fb < 0.045, T5, 1.0))
i = i_tief
while T5[i] < 0.05:
    i -= 1
f_unten = fb[i + 1]
i = i_tief
while T5[i] < 0.05:
    i += 1
f_oben = fb[i - 1]

fig, ax = plt.subplots(figsize=(8.4, 4.6))
for n_r, farbe in ((1, "C1"), (2, "C2"), (3, "C3"), (5, "C0")):
    ax.semilogy(fb * A_GIT, T_leiter[n_r], color=farbe,
                label=f"{n_r} Reihe{'n' if n_r > 1 else ''}")
ax.axvspan(f_unten * A_GIT, f_oben * A_GIT, color="gold", alpha=0.25)
ax.axvline(0.5, ls="--", color="0.5", lw=1)
ax.text(0.505, 1.2, "Bragg-Peilung c/2a", fontsize=8, color="0.4")
ax.set_xlabel("f·a/c (dimensionslos)")
ax.set_ylabel("Transmission T")
ax.set_ylim(1e-3, 3)
ax.legend(fontsize=9)
ax.grid(alpha=0.3)
plt.tight_layout()
plt.show()
print(f"Luecke (T < 0,05): f*a/c = {f_unten*A_GIT:.3f} "
      f"bis {f_oben*A_GIT:.3f}")
for n_r in (1, 2, 3, 5):
    i = np.argmin(np.abs(fb - 0.030))
    print(f"  {n_r} Reihen: T(f*a/c = 0,48) = {T_leiter[n_r][i]:.3f}")
```

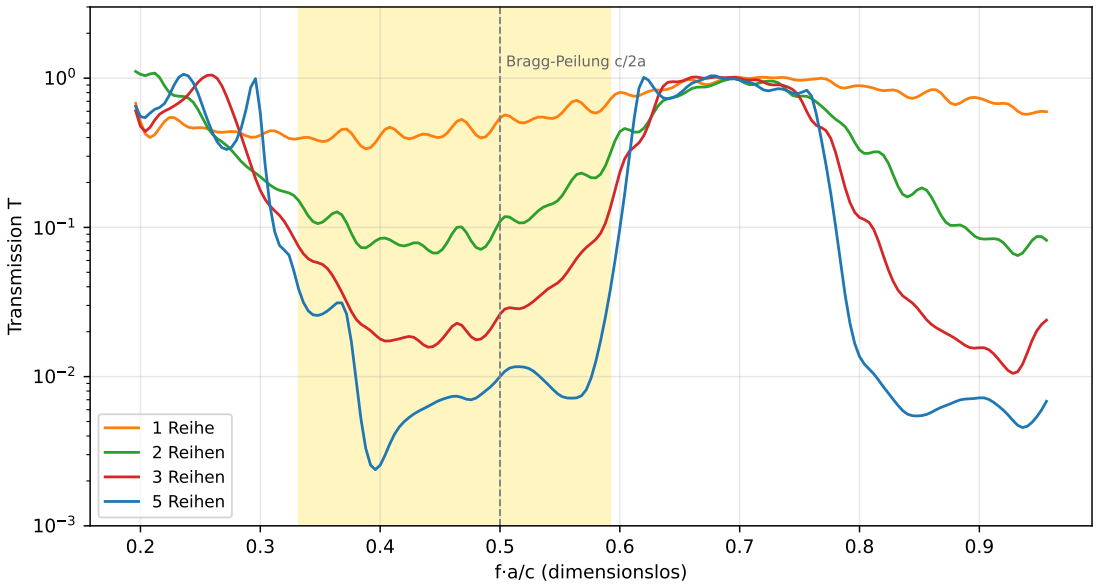


Abbildung 32.2: Das Transmissionsspektrum des Stab-Kristalls für 1, 2, 3 und 5 Reihen (Leistung, logarithmisch). Links der Lücke (f·a/c < 0,3) ist das Gitter fast durchsichtig — die langen Wellen laufen um die Stäbe herum. In der Bandlücke (gold, T < 0,05 von f·a/c = 0,33 bis 0,59) drückt JEDE zusätzliche Reihe die Transmission um etwa den Faktor 4: bei f·a/c = 0,48 von 0,42 (eine Reihe) über 0,073 und 0,018 auf 0,007 (fünf Reihen) — das exponentielle N-Reihen-Gesetz aus Kapitel 26. Die gestrichelte Bragg-Peilung c/2a liegt IN der Lücke, trifft aber nicht ihre Mitte: Der 2D-Kristall ist kein 1D-Schichtstapel. Oberhalb des Durchlassfensters um 0,65 beginnt die zweite Lücke.

```
Luecke (T < 0,05): f*a/c = 0.332 bis 0.592
  1 Reihen: T(f*a/c = 0,48) = 0.416
  2 Reihen: T(f*a/c = 0,48) = 0.073
  3 Reihen: T(f*a/c = 0,48) = 0.018
  5 Reihen: T(f*a/c = 0,48) = 0.007
```

Drei Lesefrüchte. Erstens das **N-Reihen-Gesetz**: 0,42 → 0,073 → 0,018 → 0,007 — jede Reihe drückt um etwa den Faktor 4, eine Gerade im Log-Plot. Das ist exakt das Bild der Bragg-Spiegel-Reihe aus Kapitel 26: In der Lücke fällt die Welle exponentiell, wie im Tunnel-Kapitel Kapitel 17. Wer also tiefer sperren will, baut nicht *dickere* Stäbe, sondern *mehr Reihen*. Zweitens: Weit unterhalb der Lücke ist das Gitter fast durchsichtig ($T \approx 1$) — die 80-Zellen-Welle umfließt die 5-Zellen-Stäbe, sie *sieht* nur ein leicht zäheres effektives

Medium. Ein Lattenzaun hält den Verkehrslärm nicht auf. Drittens liegt die Bragg-Peilung zwar *in* der Lücke, aber nicht in ihrer Mitte — der 2D-Kristall mit dicken Stäben ist eben kein 1D-Stapel; die Peilung taugt zum Zielen, die Kanten liefert die Messung.

32.5 Vom Gitter zum Bauplan

Bis hier war alles dimensionslos — mit Absicht, denn die **Skaleninvarianz** aus Kapitel 11 gilt für Schall genauso: Der Lauf oben beschreibt *jeden* Stab-Kristall mit $r/a = 0{,}35$; erst die Wahl von a in Metern klebt Etiketten auf die Achsen. Die Umrechnung ist eine Zeile: $f_{\mathrm{Hz}} = (f \cdot a/c) \cdot c_{\mathrm{Luft}}/a_{\mathrm{real}}$.

```
A_REAL = 0.050                  # Raster 50 mm
D_STAB = 0.035                  # Buche-Rundstab 35 mm (= 0,7 a)
print(f"r/a = {D_STAB/2/A_REAL:.2f}  (Simulation: 0,35)")
print(f"Luecke: {f_unten*A_GIT*C_LUFT/A_REAL:.0f} bis "
      f"{f_oben*A_GIT*C_LUFT/A_REAL:.0f} Hz")
```

```
r/a = 0.35  (Simulation: 0,35)
Luecke: 2278 bis 4061 Hz
```

Die Lücke landet bei **2,3 bis 4,1 Kilohertz** — mitten im empfindlichsten Bereich des Gehörs und bequem im Arbeitsbereich jedes Lautsprechers und Mikrofons. Damit ist der Bauplan festgelegt:

```
fig, ax = plt.subplots(figsize=(6.4, 6.4))
for i in range(9):
    for j in range(9):
        ax.add_patch(plt.Circle((50 + i * 50, 50 + j * 50), 17.5,
                                facecolor="burlywood",
                                edgecolor="saddlebrown"))
ax.add_patch(plt.Rectangle((0, 0), 500, 500, fill=False, lw=1.5))
ax.annotate("", xy=(100, 22), xytext=(50, 22),
            arrowprops=dict(arrowstyle="<->"))
ax.text(75, 6, "a = 50 mm", ha="center", fontsize=9)
ax.annotate("", xy=(67.5, 100), xytext=(32.5, 100),
            arrowprops=dict(arrowstyle="<->"))
ax.text(50, 121, "d = 35 mm", ha="center", fontsize=9)
ax.text(250, 520, "Brett >= 450 x 450 mm, 81 Bohrungen",
        ha="center", fontsize=10)
ax.set_xlim(-20, 520)
```

```
ax.set_ylim(-20, 545)
ax.set_aspect("equal")
ax.axis("off")
plt.tight_layout()
plt.show()
```

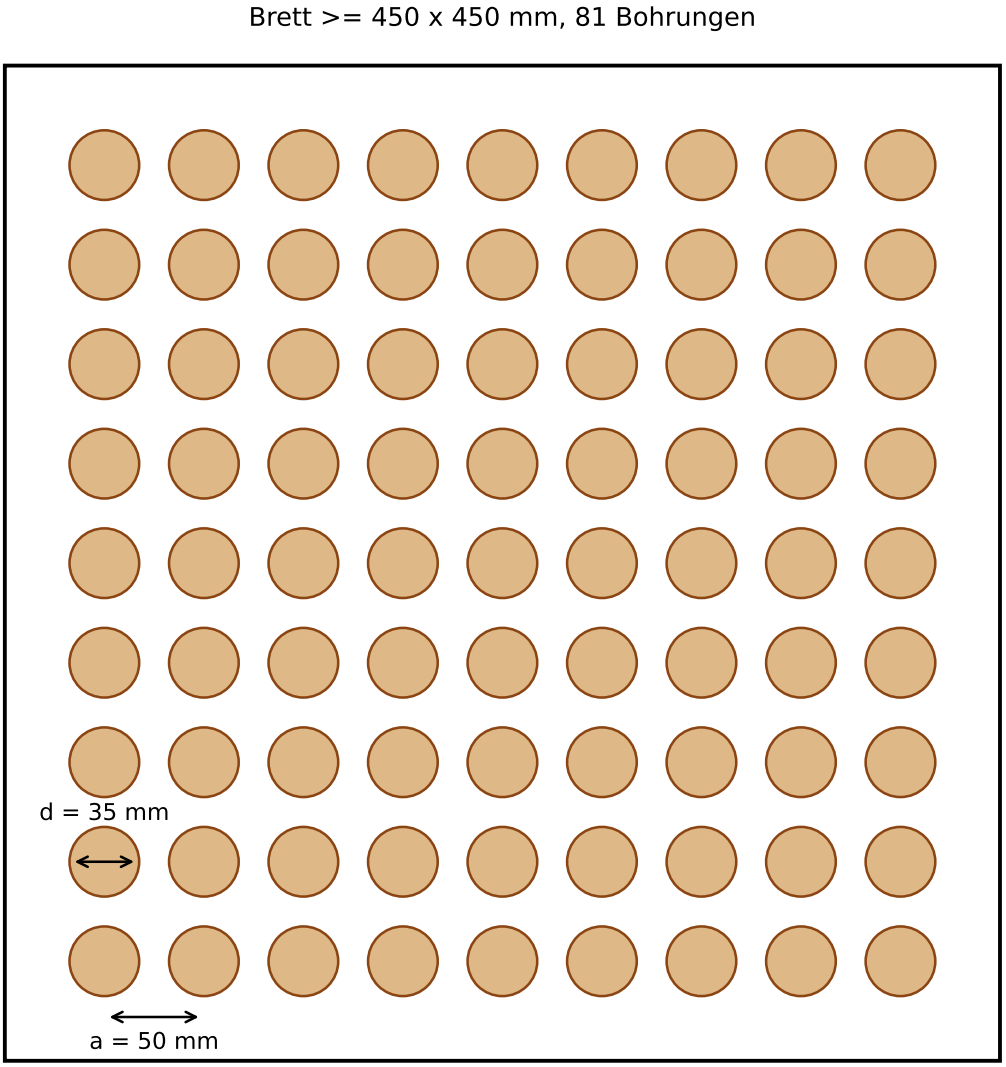


Abbildung 32.3: Die Bohrschablone: 9 × 9 Buche-Rundstäbe (Durchmesser 35 mm, Länge ≥ 30 cm) im Quadratraster a = 50 mm, gesteckt in ein Brett ab 45 × 45 cm — 81 Bohrungen, mit CNC oder Bohrständer eine Stunde Arbeit. Vorhersage der Simulation: Transmissions-Einbruch von etwa 2,3 bis 4,1 kHz, um Faktor 100 oder mehr (fünf Reihen in jeder Richtung — das Gitter wirkt in beiden Achsen). Wer nur die Lücke hören will, dem genügen schon 5 × 9 Stäbe.

Und so wird gemessen — das **Messprotokoll**, Schritt für Schritt wie bei

jeder Referenzmessung dieses Buchs:

1. **Aufbau:** Lautsprecher und Mikrofon etwa einen Meter auseinander auf Stabhöhe (Mitte des Gitters), dazwischen später der Kristall. Beides auf weichem Untergrund (Handtuch), damit der Tisch nicht mitspielt.
2. **Referenzlauf:** Einen langsamen Sinus-Sweep von 1 bis 6 kHz abspielen (20 Sekunden; jede Tongenerator-App kann das) und aufnehmen. FFT der Aufnahme — das ist $S_{\text{ohne}}(f)$, dein „Lauf ohne Kristall".
3. **Messlauf:** Kristall dazwischenstellen, exakt gleiche Aufnahme wiederholen: $S_{\text{mit}}(f)$.
4. **Auswertung:** $T(f) = S_{\text{mit}}/S_{\text{ohne}}$ — dieselbe Division wie in jeder Simulation seit Kapitel 12. In der Lücke sollte T um 20 Dezibel oder mehr einbrechen.
5. **Gegenprobe** (die Methode aus Kapitel 10): Messung wiederholen, ohne irgendetwas zu ändern — die Differenz zweier identischer Läufe zeigt dir dein Messrauschen. Erst was deutlich darüber liegt, ist Signal.

Das Wohnzimmer ist kein reflexionsfreier Raum: Echos von Wänden und Tisch wellen das Spektrum (das Kleingedruckte verrät den Profi-Trick dagegen). Aber ein Faktor-100-Einbruch über eine Oktave ist robust gegen alle Raum-Wellen — genau deshalb haben wir einen Kristall mit *tiefer* Lücke entworfen und nicht einen mit eleganter schmaler.

32.6 Schall um die Ecke: der Defekt-Kanal

Die Bandlücke sperrt — richtig spannend wird sie, wenn man sie gezielt *durchlöchert*. Wie in Kapitel 26 (Punktdefekt-Kavität) gilt: Ein Defekt im Kristall ist der einzige Ort, an dem eine Lücken-Frequenz existieren darf. Entfernen wir eine ganze **Stab-Zeile**, entsteht ein Kanal, in dem der Schall laufen kann — ringsum von Bandlücken-Wänden eingesperrt, die kein Loch und keinen Spalt haben (anders als jeder reale Flur).

Versuchsaufbau: Ein größerer Kristall (11 Reihen, 15 Säulen — die äußeren Säulen reichen bis in die Dämpfungsrampe, damit nichts außen herumläuft), Dauerton-Quelle dicht vor dem Kanalmund, Messpunkt in einer Pore kurz vor dem Kanalende. Eine Feinheit, die uns beim Entwickeln zuerst entgangen ist: Quelle und Messpunkte müssen an Stellen liegen, die in *beiden* Konfigurationen (mit und ohne Kanal) freie Luft sind — wer den Vergleichspunkt dorthin legt, wo im intakten Kristall ein Stab steht, vergleicht mit einer Zelle, in der der Druck schlicht für immer null bleibt. Die Frequenz wählen wir bei $f \cdot a/c = 0{,}53$, mitten im **Leitungsband** des Kanals: Auch der Kanal hat nämlich ein eigenes Durchlassband innerhalb der Lücke — er ist

ein periodisch beladener Wellenleiter, kein leeres Rohr (ein Puls-Vorablauf, wie wir ihn für das Spektrum oben benutzt haben, findet es).

```
RANDk, PUFk = 40, 24
NREI, NSAE = 11, 15
NXk = 2 * (RANDk + PUFk) + NREI * A_GIT
NYk = 16 + NSAE * A_GIT
X_K, Y_K = RANDk + PUFk, 8
S_KAN = 7
Y_MITTE = Y_K + int((S_KAN + 0.5) * A_GIT)
daempf_k = daempfung(NXk, NYk, RANDk)
ii_k, jj_k = np.meshgrid(np.arange(NXk), np.arange(NYk),
                         indexing="ij")


def kristall_kanal(fehlend=()):
    hart = np.zeros((NXk, NYk), bool)
    for r in range(NREI):
        for s in range(NSAE):
            if (r, s) in fehlend:
                continue
            cx = X_K + (r + 0.5) * A_GIT
            cy = Y_K + (s + 0.5) * A_GIT
            hart |= ((ii_k - cx) ** 2 + (jj_k - cy) ** 2
                     < R_STAB ** 2)
    return hart


quelle_k = (X_K - 4, slice(Y_MITTE - 4, Y_MITTE + 5))
M_END = (X_K + 9 * A_GIT, Y_MITTE)        # Pore: immer freie Luft
M_KNICK = (X_K + int(6.5 * A_GIT), Y_K + 12 * A_GIT)
kanal_gerade = [(r, S_KAN) for r in range(NREI)]
kanal_knick = ([(r, S_KAN) for r in range(7)]
               + [(6, s) for s in range(S_KAN, NSAE)])
NT_CW = 5000


def dauerton(fehlend, mess, f_cw):
    reihen, p_end, _ = schall_lauf(kristall_kanal(fehlend), daempf_k,
                                   quelle_k, f_cw, NT_CW,
                                   mikros=[mess])
    return np.sum(reihen[NT_CW * 2 // 3:, 0] ** 2), p_end
```

```
F_KANAL = 0.033
e_gerade, p_gerade = dauerton(kanal_gerade, M_END, F_KANAL)
e_intakt, p_intakt = dauerton([], M_END, F_KANAL)

fig, achsen = plt.subplots(1, 2, figsize=(9.6, 4.2))
for ax, p_b, fehlend, tit in (
        (achsen[0], p_gerade, kanal_gerade,
         f"mit Kanal: Faktor {e_gerade/e_intakt:.0f}"),
        (achsen[1], p_intakt, [], "intakt: fast nichts kommt an")):
    b = p_b.copy()
    b[kristall_kanal(fehlend)] = np.nan
    vm = np.nanpercentile(np.abs(b), 99.3)
    ax.imshow(b.T, origin="lower", cmap="RdBu", vmin=-vm, vmax=vm)
    ax.plot(*M_END, "k+", ms=11)
    ax.set_title(tit, fontsize=10)
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
print(f"Kanalende, mit Kanal gegen intakt: "
      f"Faktor {e_gerade/e_intakt:.0f}")
```

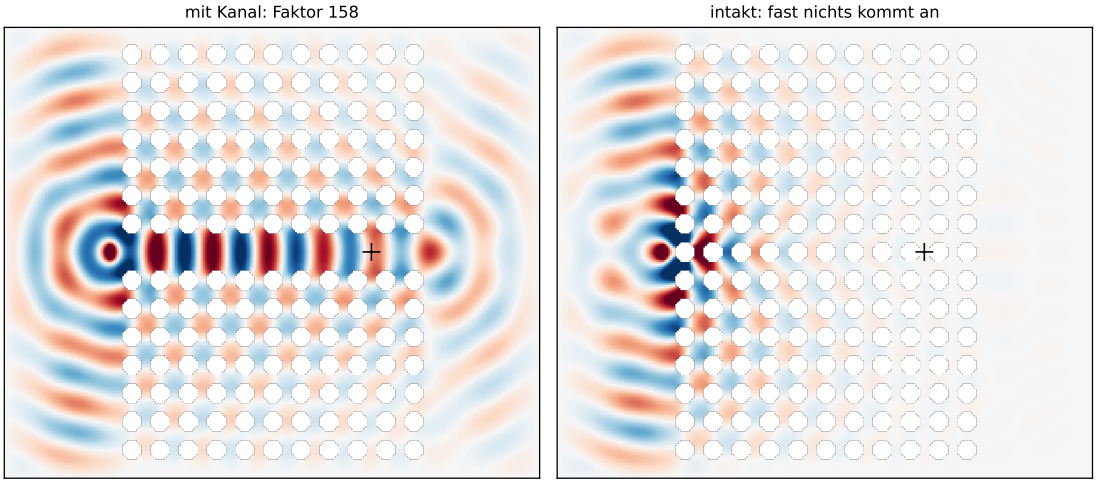


Abbildung 32.4: Eine fehlende Stab-Zeile wird zur Schall-Leitung (Dauerton mitten in der Bandlücke, f·a/c = 0,53). Links: Mit Kanal läuft der Schall als eingesperrte Welle bis zum Messpunkt (Kreuz) — am Kanalende kommt 158-mal mehr Leistung an als im intakten Kristall (rechts), wo nach wenigen Reihen praktisch nichts mehr lebt. Die Bandlücke wirkt dabei als perfekte Kanalwand: Sie hat keine Fugen, durch die der Schall seitlich entweichen könnte.

```
Kanalende, mit Kanal gegen intakt: Faktor 158
```

Faktor 158 — die fehlende Zeile leitet. Auch das ist baubar: Beim Bestücken des Bretts eine Dübelreihe weglassen, fertig ist der Schall-Flur. Jetzt der Ehrgeiz: Wir lassen den Kanal um **90 Grad abbiegen** (halbe Strecke geradeaus, dann seitlich hinaus). In Kapitel 26 hat der EM-Knick verlustfrei funktioniert — eine berühmte Eigenschaft photonischer Wellenleiter. Gilt das auch hier?

```
F_KNICK = 0.034
e_knick, p_knick = dauerton(kanal_knick, M_KNICK, F_KNICK)
e_ecke, p_ecke = dauerton(kanal_knick + [(5, 8)], M_KNICK, F_KNICK)
e_seite, _ = dauerton([], M_KNICK, F_KNICK)

fig, achsen = plt.subplots(1, 2, figsize=(9.6, 4.2))
for ax, p_b, fehlend, tit in (
        (achsen[0], p_knick, kanal_knick,
         f"Knick pur: Faktor {e_knick/e_seite:.0f}"),
        (achsen[1], p_ecke, kanal_knick + [(5, 8)],
```

```
        f"Innenecke geoeffnet: Faktor {e_ecke/e_seite:.0f}")):
    b = p_b.copy()
    b[kristall_kanal(fehlend)] = np.nan
    vm = np.nanpercentile(np.abs(b), 99.3)
    ax.imshow(b.T, origin="lower", cmap="RdBu", vmin=-vm, vmax=vm)
    ax.plot(*M_KNICK, "k+", ms=11)
    ax.set_title(tit, fontsize=10)
    ax.set_xticks([])
    ax.set_yticks([])
plt.tight_layout()
plt.show()
print(f"Knick pur: Faktor {e_knick/e_seite:.0f}, "
      f"Innenecke geoeffnet: Faktor {e_ecke/e_seite:.0f}")
```

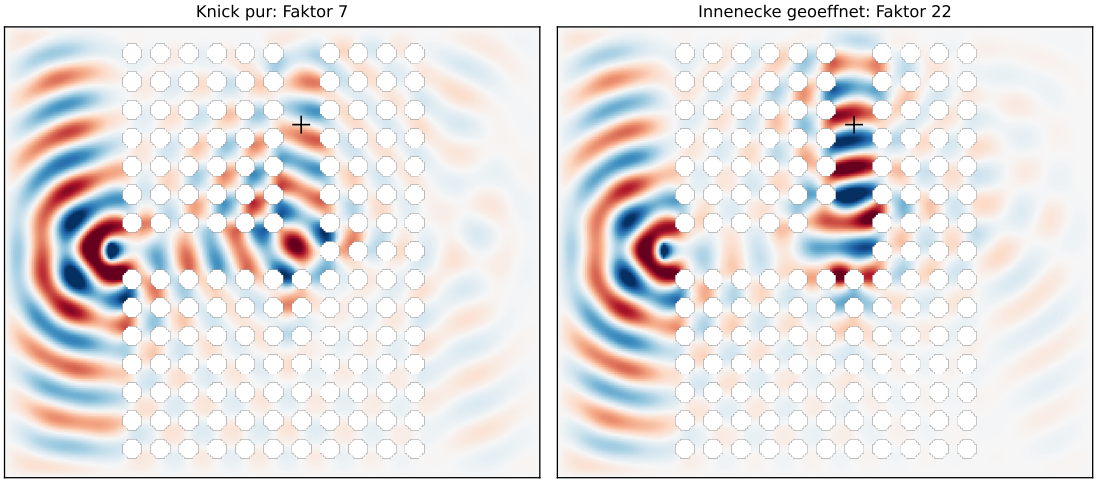


Abbildung 32.5: Der 90-Grad-Knick ist wählerischer als die Gerade (Dauerton bei f·a/c = 0,54). Links der pure Knick: Am Seitenausgang (Kreuz) kommt nur Faktor 7 gegenüber intakt an — die Ecke wirft den größten Teil zurück. Rechts derselbe Knick, nachdem EIN einziger Stab an der Innenecke zusätzlich entfernt wurde: Faktor 22, das Dreifache. Die Kapitel-26-Erfahrung „der Knick kostet nichts“ war also kein Naturgesetz, sondern das Ergebnis eines gut abgestimmten Designs — Ecken sind ein eigenes Handwerk, und Übung 32.4 überlässt es dem Optimierer.

```
Knick pur: Faktor 7, Innenecke geoeffnet: Faktor 22
```

Nein — die Ecke reflektiert kräftig (Faktor 7 statt 158). Aber ein einziger

zusätzlich entfernter Stab an der *Innenecke* verdreifacht die Ausbeute: Die Kurve wird innen aufgeweitet, die Welle bekommt Platz zum Abbiegen. In der Photonik ist genau dieses Ecken-Tuning ein Standard-Handgriff. Merke: Defekt-Bauteile (Kanäle, Kurven, Verzweigungen) haben eigene Resonanzen und eigene Bänder — wer sie entwirft, misst nach. Oder lässt entwerfen: In Übung 32.4 darf der Kapitel-31-Optimierer die Ecke selbst kurieren — und findet den Innenecken-Trick ohne je ein Photonik-Lehrbuch gesehen zu haben.

Eine Begriffsfalle noch, weil sie naheliegt: Unser Kanal ist **kein Hohlleiter** im Sinn von Kapitel 17. Der EM-Hohlleiter (PEC-Wände, $E = 0$) hat eine Cutoff-Frequenz, unter der nichts läuft. Ein *schallharter* Kanal hat keinen Cutoff — an harten Wänden darf der Druck ja gerade maximal sein (die Vorzeichen-Probe von oben!), und eine ebene Welle läuft durch jeden noch so engen harten Schlauch. Was unseren Kanal nach unten begrenzt, ist nicht die Kanalbreite, sondern die Bandlücke seiner Wände.

32.7 Das akustische Pixel-Puzzle

Zum Schluss schließt sich der Kreis zu Kapitel 31: Wenn der Kern derselbe ist, muss auch das **inverse Design** wörtlich übertragbar sein. Wir stellen dem Optimierer dieselbe Aufgabe wie dort — lenke die Welle zu einem schräg versetzten Zielpunkt —, nur dass die 36 Pixel diesmal nicht aus $\varepsilon = 6$ bestehen, sondern **schallhart** sind: Jedes gesetzte Pixel ist ein kleiner Klotz, das gesamte Suchgebiet ein 3D-Druckteil von wenigen Zentimetern.

Versuchsaufbau: Wörtlich Kapitel 31 — Design-Region aus 6 × 6 Pixeln à 4 × 4 Zellen, eine Dauerton-Quellzeile links, Zielpunkt schräg rechts oben, Zielgröße die dort aufsummierte Druckleistung. Optimierer: Bit-Flip-Bergsteigen, 250 Münzwürfe, behalte jede Verbesserung. Ein Lauf dauert 0,04 Sekunden — der ganze Entwurf gut zehn Sekunden.

```
NXp, NYp = 150, 110
RANDp = 10
daempf_p = daempfung(NXp, NYp, RANDp, s_max=0.35)
PIX, NPX, NPY = 4, 6, 6
X0, Y0 = 60, 43
ZIEL = (130, 88)
quelle_p = (18, slice(RANDp + 5, NYp - RANDp - 5))


def bewerte(pixel, feld_zurueck=False):
    hart = np.zeros((NXp, NYp), bool)
```

```
    for i in range(NPX):
        for j in range(NPY):
            if pixel[i, j]:
                hart[X0+i*PIX:X0+(i+1)*PIX,
                     Y0+j*PIX:Y0+(j+1)*PIX] = True
    reihen, p_end, bilder = schall_lauf(hart, daempf_p, quelle_p,
                                        1 / 20.0, 500,
                                        mikros=[ZIEL],
                                        schnappschuesse=(range(150, 500)
                                            if feld_zurueck else ()))
    summe = np.sum(reihen[151:, 0] ** 2)
    if feld_zurueck:
        karte = np.max([np.abs(b) for b in bilder.values()], axis=0)
        return summe, karte
    return summe


leer = bewerte(np.zeros((NPX, NPY), bool))
rng = np.random.default_rng(32)
pixel = np.zeros((NPX, NPY), bool)
wert = leer
verlauf = [wert]
for it in range(250):
    i, j = rng.integers(0, NPX), rng.integers(0, NPY)
    pixel[i, j] = ~pixel[i, j]
    neu = bewerte(pixel)
    if neu > wert:
        wert = neu
    else:
        pixel[i, j] = ~pixel[i, j]
    verlauf.append(wert)
zufall = [bewerte(rng.random((NPX, NPY)) > 0.5) for _ in range(20)]
_, karte = bewerte(pixel, feld_zurueck=True)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(9.6, 4.0))
ax1.plot(verlauf)
ax1.axhline(leer, ls="--", color="gray", label="leere Region")
ax1.axhline(max(zufall), ls=":", color="C3",
            label="bestes Zufallsmuster")
ax1.set_xlabel("Flip-Versuche")
ax1.set_ylabel("Leistung am Ziel")
ax1.legend(fontsize=8)
ax2.imshow(karte.T, origin="lower", cmap="inferno")
```

```
for i in range(NPX):
    for j in range(NPY):
        if pixel[i, j]:
            ax2.add_patch(plt.Rectangle((X0 + i*PIX, Y0 + j*PIX),
                                        PIX, PIX, facecolor="none",
                                        edgecolor="c", lw=0.8))
ax2.plot(*ZIEL, "w+", ms=11)
ax2.set_xticks([])
ax2.set_yticks([])
plt.tight_layout()
plt.show()
print(f"leer {leer:.1f} -> optimiert {wert:.1f} "
      f"(Faktor {wert/leer:.1f}); "
      f"bestes Zufallsmuster {max(zufall):.1f}")
```

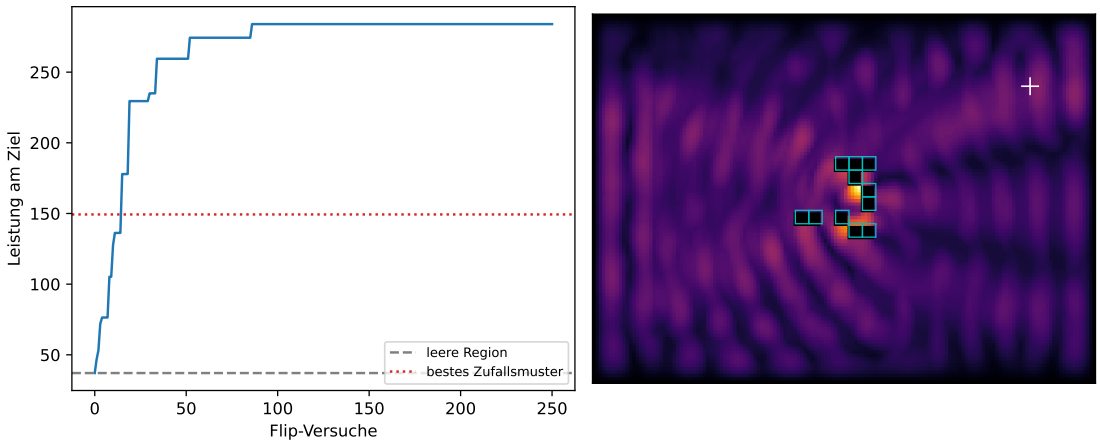


Abbildung 32.6: Das Pixel-Puzzle, akustische Fassung: Links der Optimierungs-Verlauf — von 37 (leere Region) auf 284, Faktor 7,7; das beste von 20 Zufallsmustern (rot) bleibt bei 149. Rechts der gefundene Umlenker bei der Arbeit (Karte: max |p|): Die schallharten Pixel (cyan) wirken zusammengewürfelt — kein Mensch hätte dieses Muster gezeichnet —, lenken aber nachweislich Faktor 7,7 mehr Leistung zum Kreuz als die leere Region. Dasselbe Verfahren, derselbe Kern, anderes Etikett — und das Ergebnis wäre als Zentimeter-Klötzchen druckbar.

```
leer 37.1 -> optimiert 284.0 (Faktor 7.7); bestes Zufallsmuster 149.3
```

Faktor 7,7 — sogar etwas mehr als die elektromagnetische Fassung in Kapitel 31 (dort 5,9), denn schallharte Pixel streuen stärker als $\varepsilon = 6$. Damit ist

der Bogen geschlossen: Die Methode des letzten Kapitels — Ziel formulieren, Vorwärtsmaschine einspannen, Optimierer würfeln lassen — funktioniert unverändert in einer anderen Physik. Das ist keine Schall-Besonderheit, sondern die eigentliche Lehre der Zugabe: **Wer einen Wellenlöser und eine Zielfunktion hat, kann entwerfen — egal, welche Welle.**

32.8 Das Kapitel-Programm

`programme/kap32/kap32_akustik.py` bündelt alle Befunde eigenständig und mit `assert`-Schranken: die Port-Probe ($c = 1{,}000$), das Transmissionsspektrum mit N-Reihen-Leiter und Lückenkanten samt Werkstatt-Umrechnung, den geraden Kanal und den Knick mit Innenecken-Kur, das akustische Pixel-Puzzle (Faktor > 4 gegen leer, besser als 20 Zufallsmuster) und die Impedanz-Tabelle. Reines NumPy mit fester Zufalls-Saat; Laufzeit etwa zwei Minuten.

Merkkasten

- **Übersetzungstabelle:** $E \to p$, $H \to v$, $\varepsilon \to 1/\kappa$, $\mu \to \rho$, $Z = \rho c$ — die Maxwell-Werkzeugkiste rechnet Akustik, weil die Gleichungsstruktur identisch ist. Aber: Randbedingungen mitübersetzen! Harte Wand = Druckbauch (kein Flip), offenes Ende = Druckknoten (Flip).
- **Schallhart:** Holz, Alu, PLA reflektieren > 99,9 % — feste Körper sind für Luftschall, was PEC für Mikrowellen ist. Im Gitter: $v = 0$ auf allen Kanten harter Zellen.
- **Werkstatt-Landkarte:** Beim Schall im kHz-Bereich treffen sich baubare Wellenlänge (cm bis dm) und kostenloses Messgerät (Mikrofon + FFT) — der volle Kreis vorhersagen → simulieren → bauen → messen wird zu Hause schließbar.
- **Phononischer Kristall:** Stabgitter mit $r/a = 0{,}35$ sperrt $f \cdot a/c \approx 0{,}33$ bis $0{,}59$; jede Reihe drückt T um ≈ Faktor 4 (N-Reihen-Gesetz). Bei $a = 50$ mm: Lücke 2,3 bis 4,1 kHz.
- **Defekte sind Bauteile:** Eine fehlende Stab-Zeile leitet (Faktor 158); Ecken reflektieren und brauchen Tuning — Defekt-Bauteile haben eigene Bänder, also: nachmessen.
- **Methoden reisen:** Bit-Flip-Bergsteigen + Akustik-Kern = druckbarer Schall-Umlenker. Wellenlöser + Zielfunktion = Entwurfsmaschine, unabhängig von der Wellensorte.

Roter Faden

Die Zugabe erntet quer durchs Buch: Der **Kern aus Kapitel 9** wurde wortwörtlich übernommen (nur die Schilder getauscht), die **Spiegelformel aus Kapitel 6** lieferte mit $Z = \rho c$ die Schallhart-Näherung, die **Messmethodik aus Kapitel 10** (Sweep, FFT, Referenzlauf, Rausch-Gegenprobe) wurde zum Wohnzimmer-Protokoll, die **Bandlücken-Physik aus Kapitel 26** (Bragg-Bedingung, N-Reihen-Gesetz, Defekt-Logik) fand ihr akustisches Double, die **Skaleninvarianz aus Kapitel 11** klebte die Hertz-Etiketten auf, und das **inverse Design aus Kapitel 31** lief unverändert auf der neuen Physik. Dass all das funktioniert, ist die letzte und vielleicht wichtigste Lektion des Buchs: Du hast nicht 31 Kapitel Elektromagnetik gelernt, sondern 31 Kapitel *Wellen, Numerik und Messkultur* — Maxwell war der Lehrmeister, aber das Gelernte gehört dir überall.

Übungen

Ü 32.1 — Gel, Taucher, Stäbe (Verstehen). (a) Körpergewebe hat etwa $Z \approx 1{,}6 \cdot 10^6$ kg/(m²s). Welcher Anteil der Leistung eines Ultraschallkopfs käme durch eine dünne Luftschicht ins Gewebe — und warum löst das Gel das Problem? (b) Jemand ruft einem Taucher vom Boot aus etwas zu. Wie viel Rufleistung erreicht sein Ohr unter Wasser? (c) Begründe mit den Zahlen aus dem Impedanz-Block, warum wir Holzstäbe als ideal schallhart rechnen durften.

Musterlösung zu Ü 32.1

```python
Z_GEWEBE = 1.6e6
T_gel = 1 - ((Z_GEWEBE - Z_LUFT) / (Z_GEWEBE + Z_LUFT)) ** 2
Z_WASSER = 1000.0 * 1480.0
T_wasser = 1 - ((Z_WASSER - Z_LUFT) / (Z_WASSER + Z_LUFT)) ** 2
print(f"(a) durch Luft ins Gewebe: T = {T_gel*100:.2f} %")
print(f"(b) Luft -> Wasser: T = {T_wasser*100:.2f} %")
```

```
(a) durch Luft ins Gewebe: T = 0.10 %
(b) Luft -> Wasser: T = 0.11 %
```

(a) Nur etwa 0,1 Prozent — eine Luftschicht von Bruchteilen eines Millimeters genügt für Totalblockade, denn die Grenzen Luft–Gewebe wirken doppelt (hinein und am Schallkopf zurück). Das Gel (Z nahe Wasser/Gewebe) ersetzt die Luftschicht durch einen fast stufenlo-

sen Übergang — dieselbe Logik wie die Anpassung in Kapitel 22, nur dass hier *Weglassen* der Fehlstelle genügt. (b) Ebenfalls ~0,1 Prozent: Die Wasseroberfläche ist von oben ein fast perfekter Spiegel — unter Wasser klingt die Welt über Wasser darum so dumpf und fern. (c) Holz steht mit $Z = 2{,}8 \cdot 10^6$ noch über Gewebe und Wasser: 99,94 Prozent Reflexion. Der Fehler der Ideal-hart-Näherung (0,06 Prozent pro Stabkontakt) liegt weit unter jeder anderen Unsicherheit des Modells.

Ü 32.2 — Die Füllfaktor-Reihe (Verändern). Miss das Transmissionsspektrum (5 Reihen) für $r/a = 0{,}25$, $0{,}30$ und $0{,}35$. Wie ändern sich Breite und Tiefe der Lücke? Formuliere eine Faustregel für den Werkstatt-Einsatz.

Musterlösung zu Ü 32.2

```
for r_a in (0.25, 0.30, 0.35):
    R_test = r_a * A_GIT
    hart = np.zeros((NXc, NYc), bool)
    y0 = (NYc - N_SAEULEN * A_GIT) / 2
    for r in range(5):
        for s in range(N_SAEULEN):
            cx = X_KRIST + (r + 0.5) * A_GIT
            cy = y0 + (s + 0.5) * A_GIT
            hart |= ((ii_c - cx) ** 2 + (jj_c - cy) ** 2
                     < R_test ** 2)
    T = leistungsspektrum(hart)[band] / spek_frei[band]
    it = np.argmin(np.where(fb < 0.045, T, 1.0))
    i = it
    while T[i] < 0.05:
        i -= 1
    fu = fb[i + 1]
    i = it
    while T[i] < 0.05:
        i += 1
    fo = fb[i - 1]
    print(f"r/a = {r_a}: T_min = {T[it]:.4f}, Luecke "
          f"f*a/c = {fu*A_GIT:.2f}..{fo*A_GIT:.2f} "
          f"(Breite {(fo-fu)*A_GIT:.2f})")
```

```
r/a = 0.25: T_min = 0.0159, Luecke f*a/c = 0.42..0.54 (Breite 0.12)
r/a = 0.3: T_min = 0.0055, Luecke f*a/c = 0.37..0.57 (Breite 0.20)
```

```
r/a = 0.35: T_min = 0.0024, Luecke f*a/c = 0.33..0.59 (Breite 0.26)
```

Dickere Stäbe wirken doppelt: Die Lücke wird *breiter* (Breite 0,12 → 0,20 → 0,26 — mehr als verdoppelt) und *tiefer* (T_{min} 0,015 → 0,005 → 0,002), und ihre Unterkante rutscht nach unten (die Welle wird im engeren Porenraum effektiv langsamer). Faustregel für die Werkstatt: **So dicke Stäbe wie beschaffbar** — die Lücke verzeiht dann auch Bohr-Toleranzen; mit $d/a \lesssim 0{,}5$ wird die Lücke schmal und kippelig. Nach oben begrenzt die Nachbarschaft: Ab $d/a \approx 0{,}9$ berühren sich die Stäbe fast und aus dem Gitter wird eine Wand mit Schlitzen.

Ü 32.3 — Dasselbe Gitter bei 40 kHz (Übertragen). Dein 3D-Drucker soll den Kristall für Ultraschall (40 kHz) drucken. (a) Welche Gitterkonstante und welcher Stabdurchmesser folgen aus der Skaleninvarianz? Ist das druckbar? (b) Kann deine 96-kHz-Soundkarte die Messung übernehmen? (c) Wie viele neue Simulationen brauchst du für den Entwurf?

Musterlösung zu Ü 32.3

```python
f_mitte = (f_unten + f_oben) / 2 * A_GIT      # f*a/c der Lueckenmitte
a_us = f_mitte * C_LUFT / 40e3
print(f"f*a/c (Mitte) = {f_mitte:.2f}")
print(f"(a) a = {a_us*1000:.1f} mm, d = {0.7*a_us*1000:.1f} mm")
```

```
f*a/c (Mitte) = 0.46
(a) a = 4.0 mm, d = 2.8 mm
```

(a) $a \approx 4$ mm mit $d \approx 2{,}8$ mm — für einen 0,4-mm-Düsendrucker problemlos; das ganze 9 × 9-Gitter misst keine 4 Zentimeter. (b) Knapp nein: Nyquist erlaubt bei 96 kHz zwar theoretisch 48 kHz, aber gewöhnliche Mikrofone und Lautsprecher geben weit oberhalb von 20 kHz fast nichts mehr ab beziehungsweise auf — du bräuchtest 40-kHz- Ultraschallkapseln (Centartikel, in jedem Park-Sensor) und einen schnelleren Analog-Digital-Wandler oder ein Oszilloskop. Die Messkette, nicht die Mathematik, setzt die Grenze — die Landkarten-Lektion dieses Kapitels. (c) **Null.** Die Simulation war dimensionslos; nur die Etiketten ändern sich — die Skaleninvarianz aus Kapitel 11 in ihrer ganzen Pracht.

Ü 32.4 — Der Ecken-Doktor (Übertragen). Lass den Kapitel-31-Optimierer die Knick-Ecke kurieren: Die neun Stäbe der 3 × 3-Region innen an der Ecke (Reihen 3–5, Säulen 8–10) dürfen einzeln verschwinden. Optimiere mit Bit-Flip-Bergsteigen (30 Flips, Start: alle Stäbe vorhanden)

die Leistung am Seitenausgang. Findet er den Innenecken-Trick?

Musterlösung zu Ü 32.4

```
ECKE = [(r, s) for r in (3, 4, 5) for s in (8, 9, 10)]


def knick_wert(frei_bits):
    fehlend = kanal_knick + [e for e, b in zip(ECKE, frei_bits)
                             if b]
    return dauerton(fehlend, M_KNICK, F_KNICK)[0]


bits = np.zeros(9, bool)
wert_k = knick_wert(bits)
print(f"Start (Knick pur): Faktor {wert_k/e_seite:.0f}")
rng_e = np.random.default_rng(7)
for it in range(30):
    k = rng_e.integers(0, 9)
    bits[k] = ~bits[k]
    neu = knick_wert(bits)
    if neu > wert_k:
        wert_k = neu
        frei = [e for e, b in zip(ECKE, bits) if b]
        print(f"  Flip {it}: entfernt {frei} -> "
              f"Faktor {wert_k/e_seite:.0f}")
    else:
        bits[k] = ~bits[k]
print(f"Ende: Faktor {wert_k/e_seite:.0f}")
```

```
Start (Knick pur): Faktor 7
  Flip 2: entfernt [(5, 8)] -> Faktor 22
  Flip 4: entfernt [(4, 10), (5, 8)] -> Faktor 31
Ende: Faktor 31
```

Ja — und zwar als *ersten* erfolgreichen Zug: Flip 2 entfernt genau den Innenecken-Stab (5, 8) und springt von Faktor 7 auf 22; Flip 4 legt mit (4, 10) nach (Faktor 31), danach ist der kleine Suchraum ausgereizt. Der Optimierer hat den Photonik-Standard-Handgriff in Sekunden wiederentdeckt — wie beim Bragg-Spiegel in Kapitel 31 gilt: Er kennt keine Regeln, aber er findet, was die Zielfunktion belohnt. Wer mag, vergrößert die Region auf 4×4 — der Suchraum wächst auf 2^{16}, das Bergsteigen bleibt bei ein paar Dutzend Läufen.

Das Kleingedruckte

Die Skulptur, die zuerst sperrte. Die erste experimentell vermessene akustische Bandlücke steckte in einem Kunstwerk: *Órgano*, eine minimalistische Skulptur von Eusebio Sempere im Museumspark von Madrid — Edelstahlrohre, 2,9 cm dick, im 10-cm-Raster. Martínez-Sala und Kollegen vermaßen sie 1995 (*Nature* 378, 241) und fanden den vorhergesagten Transmissions-Einbruch um 1,7 kHz. Unser Rundholz-Gitter ist also in bester Gesellschaft — und die Originalarbeit eine schöne Bestätigung dafür, dass die Hobby-Messung funktioniert: Auch dort stand die Skulptur unter freiem Himmel, nicht im Labor.

Festkörper: „Maxwell plus mehr". Unsere Übersetzungstabelle gilt für Schall in Fluiden (Luft, Wasser). *In* Festkörpern wird es nicht einfacher, sondern reicher: Dort gibt es zusätzlich **Scherwellen** mit eigener, langsamerer Geschwindigkeit, und an jeder Grenzfläche wandeln Druck- und Scherwellen ineinander um (Modenkonversion). Die Elastodynamik hat darum *mehr* Feldkomponenten als Maxwell, nicht weniger. Für unser Gitter ist das egal — der Schall bleibt in der Luft, die Stäbe sind nur harte Ränder —, aber wer Ultraschall *durch* Bauteile schicken will (Werkstoffprüfung), braucht das volle Programm.

Akustische Metamaterialien und schwebende Tröpfchen. Auch Block D reist mit: Es gibt akustische Metamaterialien mit negativer effektiver Dichte oder Kompressibilität (Resonatoren statt Drähte und Ringe), akustische Tarnkappen-Prototypen und Labyrinth-Strukturen, die Schall auf Bruchteile seiner Wellenlänge falten. Und die Levitation gibt es wörtlich: Ein Raster aus 40-kHz-Kapseln (das *TinyLev*-Design ist quelloffen und 3D-druckbar) hält Styroporkugeln und Wassertröpfchen in den Druckknoten einer Stehwelle schweben — Strahlungsdruck wie in Kapitel 18, nur in Luft statt Licht.

Messen im Wohnzimmer, ehrlich. Der Raum reflektiert, und $T(f)$ aus zwei Sweeps wellt sich entsprechend. Der Profi-Trick heißt **Zeitfensterung (Gating)**: Statt eines Sweeps nimmt man die Impulsantwort auf (Händeklatschen genügt zur Probe) und schneidet im Zeitsignal alles ab, was nach dem Direktschall eintrifft — Wandechos brauchen ein paar Millisekunden länger, und nach der FFT des gefensterten Signals ist der Raum fast verschwunden. Das ist exakt die Fenster-Logik aus Kapitel 10, nur auf Echos statt auf Spektren angewandt. Für unseren Faktor-100-Einbruch ist sie Kür, nicht Pflicht; wer aber die Lückenkanten auf Prozent genau mit der Simulation vergleichen will, kommt an ihr nicht vorbei.

Und damit ist die Zugabe gespielt. Das Brett liegt im Baumarkt, der Bohrplan in Abbildung 32.3 — was du daraus baust und misst, ist dein erstes eigenes Wellen-Experiment nach diesem Buch. Viel Freude dabei.