FEM verstehen

**Wärme, Verformung und die Methode der finiten Elemente —
Schritt für Schritt mit Python**

Dipl. Ing. Sören Gebbert Claude Fable 5

2026-07-23

Inhaltsverzeichnis

Vorwort **1**
Worum es geht 1
Der rote Faden 2
Wie man dieses Buch liest 2
Danksagung und Werkstattbericht 3

I. Teil I — Ankommen **5**

1. Der Träger im Feuer **7**
Lernziele 9
1.1. Das Finale zum Anfassen 10
1.2. Was ist eigentlich eine Simulation? 18
1.3. Wo die finiten Elemente überall stecken 20
1.4. Die fünf Bausteine und die Landkarte des Buches 21
1.5. Woher die Methode kommt 23
Roter Faden 23
Übungen 24
Das Kleingedruckte 25

2. Python von Null **27**
Lernziele 27
2.1. Variablen: Dingen einen Namen geben 28
2.2. Sauber drucken mit f-Strings 29
2.3. Listen: eine Zahl für jeden Knoten 30
2.4. Schleifen: jeden Knoten der Reihe nach 33
2.5. Entscheidungen treffen mit `if` 37
2.6. Funktionen: dem Programm eigene Wörter beibringen 38
2.7. Verschachtelte Listen: Tabellen und Matrizen 40
2.8. Die interaktive Einheit: deine Temperaturtabelle 42
2.9. Das Kapitel-Programm 44
Roter Faden 45
Übungen 45
Das Kleingedruckte 47

II. Teil II — Die FEM-Maschinerie ohne Analysis **49**

3. Die Federkette **51**
Lernziele . . . 52
3.1. Eine einzelne Feder . . . 53
3.2. Das Kräftegleichgewicht an einem Knoten . . . 54
3.3. Der Steckbrief der Feder: die 2×2-Elementmatrix . . . 55
3.4. Assemblierung: aus drei Steckbriefen ein System . . . 57
3.5. Die Randbedingung: das Auflager einbauen . . . 62
3.6. Von Hand lösen . . . 63
3.7. Die interaktive Einheit: der Matrix-Drucker . . . 64
3.8. Das Kapitel-Programm . . . 68
3.9. Woher die Methode kommt . . . 68
Roter Faden . . . 70
Übungen . . . 71
Das Kleingedruckte . . . 72

4. Gleichungssysteme lösen **75**
Lernziele . . . 75
4.1. Was „lösen" überhaupt heißt . . . 76
4.2. Gauß-Elimination: erst leerräumen, dann ernten . . . 78
4.3. Warum die FEM-Matrix fast nur aus Nullen besteht . . . 83
4.4. Der Thomas-Algorithmus: Gauß, der nur das Band anfasst . . . 84
4.5. Gauß-Seidel: die Lösung pendelt sich ein . . . 87
4.6. Was kostet das alles? . . . 92
4.7. Die interaktive Einheit: der gläserne Löser . . . 93
4.8. Das Kapitel-Programm . . . 96
Roter Faden . . . 97
Übungen . . . 98
Das Kleingedruckte . . . 99

III. Teil III — Der Balken in 1D: von der Physik zur FEM **101**

5. Steigung und Fläche **103**
Lernziele . . . 105
Erste Frage: Wie steil? . . . 106
5.1. Die Steigung zwischen zwei Punkten . . . 106
5.2. Das Steigungsdreieck schrumpfen lassen . . . 109
5.3. Der Sonderfall, um den sich alles dreht: die Gerade . . . 112
Zweite Frage: Wie viel? . . . 113
5.4. Die Fläche unter der Kurve . . . 113
5.5. Wieder die Gerade: das Trapez ist exakt . . . 118
5.6. Alles nur Geraden — der Brückenschlag zu Kapitel 7 . . . 119

5.7. Die interaktive Einheit: das Steigungs- und Flächenlabor . . . 121
5.8. Das Kapitel-Programm . . . 124
Roter Faden . . . 124
Übungen . . . 125
Das Kleingedruckte . . . 126

6. Die Wärmeleitungsgleichung 129
Lernziele . . . 129
6.1. Fouriers Gesetz: das Gefälle treibt den Strom . . . 130
6.2. Die Bilanz am Trägerstück . . . 136
6.3. Lösen durch Probieren: die Gerade . . . 139
6.4. Die interaktive Einheit: der Bilanz-Prüfer . . . 141
6.5. Die Bilanz als Bewegung . . . 145
6.6. Das Kapitel-Programm . . . 146
Roter Faden . . . 147
Übungen . . . 148
Das Kleingedruckte . . . 149

7. Die schwache Form: von der Gleichung zur Federkette 151
Lernziele . . . 152
7.1. Schritt 1 — Der Ansatz: Zuständigkeitsgebiete . . . 153
7.2. Schritt 2 — Einsetzen und scheitern . . . 157
7.3. Schritt 3 — Gewichtete Reste: die Bilanz im Mittel . . . 159
7.4. Schritt 4 — Umverteilung: die Steigungslast auf beide Schultern 161
7.5. Schritt 5 — Die Elementmatrix, und der Gänsehaut-Moment . 163
7.6. Schritt 6 — Assemblieren, einspannen, lösen . . . 166
7.7. Die interaktive Einheit: die Wärme-Federkette . . . 170
7.8. Das Kapitel-Programm . . . 174
7.9. Rückblick auf die Landkarte . . . 174
7.10 Die Namensschilder . . . 175
Roter Faden . . . 177
Übungen . . . 177
Das Kleingedruckte . . . 179

8. Randbedingungen 181
Lernziele . . . 182
8.1. Drei Ränder, drei Geschichten . . . 183
8.2. Der geparkte Randterm — jetzt abgeholt . . . 185
8.3. Neumann konkret: der Brandherd mit 200 Watt . . . 186
8.4. Dirichlet konkret: die Temperatur festhalten . . . 189
8.5. Wenn man nichts vorgibt: die Isolierung von selbst . . . 192
8.6. Der schwimmende Träger . . . 194
8.7. Umschalten: das Profil klappt um . . . 197
8.8. Die interaktive Einheit: der Rand-Schalter . . . 198

8.9. Das Kapitel-Programm . . . 203
8.10 Die Randbedingungs-Werkstatt . . . 204
8.11 Die Namensschilder . . . 204
Roter Faden . . . 206
Übungen . . . 206
Das Kleingedruckte . . . 208

9. Jetzt mit Zeit **211**
Lernziele . . . 212
9.1. Das Konto ändert sich jetzt doch . . . 213
9.2. Von der Wärmekapazität zur Massenmatrix . . . 215
9.3. Der explizite Euler-Schritt . . . 217
9.4. Das Experiment: den Zeitschritt vergrößern . . . 220
9.5. Die implizite Rettung . . . 222
9.6. Zwanzig Stunden — und der Brandschutz rechnet in Minuten 225
9.7. Die interaktive Einheit: die Zeitmaschine . . . 227
9.8. Das Kapitel-Programm . . . 231
Rückblick auf die Landkarte . . . 231
Roter Faden . . . 232
Übungen . . . 233
Das Kleingedruckte . . . 234

IV. Teil IV — In die Fläche **235**

10. In die Fläche: Dreiecke **237**
Lernziele . . . 238
10.1 Der Träger als Fläche: die Geometrie festlegen . . . 239
10.2 Vom Streckenanteil zum Flächenanteil . . . 241
10.3 Drei Punkte zum Anfassen: Ecke, Kante, Mitte . . . 244
10.4 Das Zelt: die Ansatzfunktion in der Fläche . . . 246
10.5 Das Netz als Bauplan . . . 248
10.6 Die Elementmatrix: der 3×3-Steckbrief . . . 249
10.7 Assemblieren, einspannen, lösen . . . 251
10.8 Die interaktive Einheit: das Dreiecks-Labor . . . 256
10.9 Das Kapitel-Programm . . . 263
10.10 Die Namensschilder . . . 263
Roter Faden . . . 265
Übungen . . . 265
Das Kleingedruckte . . . 267

11. Rechtecke **269**
Lernziele . . . 270
11.1 Bilinear: zweimal interpolieren statt einmal . . . 270

11.2 Die vier Ansatzfunktionen: zwei Hütchen im Produkt 273
11.3 Der Sattel: warum das Innere sich krümmt 274
11.4 Die Elementmatrix: vier mal vier, aus 1D-Bausteinen 277
11.5 Der Rechteck-Träger: gleiche Lösung, anderes System 280
11.6 Die interaktive Einheit: das Rechteck-Labor 284
11.7 Das Kapitel-Programm 291
11.8 Ausblick und Kleingedrucktes 292
11.9 Die Namensschilder 294
Roter Faden 295
Übungen 295

12. Traue keinem bunten Bild **297**
Lernziele 298
12.1 Die drei Fragen — und zwei Wörter, die man nicht verwechseln darf 299
12.2 Prüffall 1 — die Gerade: der Nulltest 300
12.3 Prüffall 2 — die Parabel: der erste echte Diskretisierungsfehler 301
12.4 Dasselbe in der Fläche — und eine Falle beim Testen 303
12.5 Prüffall 3 — die Abkühlkurve gegen eine exakte Reihe 306
12.6 Die Energiebilanz — der Wachhund, der immer mitläuft 308
12.7 Die Fehler-Sprechstunde: die fünf Klassiker an ihren Symptomen 309
12.8 Die interaktive Einheit: die Prüfstation 312
12.9 So machen es die Großen 317
12.10 Die Auflösung des Aufhängers 318
Roter Faden 319
Übungen 320
Das Kleingedruckte 321

V. Teil V — Der Balken verformt sich **323**

13. Der Träger dehnt sich: 1D-Thermomechanik **325**
Lernziele 326
13.1 Schritt 1 — Dehnung: Verformung ohne Einheit 327
13.2 Schritt 2 — Spannung: Kraft, auf die Fläche verteilt 328
13.3 Schritt 3 — Hooke als Material-Feder: dieselbe Formelgestalt 329
13.4 Schritt 4 — Wärmedehnung: der Träger will länger werden . . 332
13.5 Schritt 5 — Die Kernformel: Spannung aus verhinderter Dehnung 333
13.6 Schritt 6 — FEM-Übersetzung: die thermische Kraft 335
13.7 Schritt 7 — Der gekoppelte Träger: das Temperaturfeld als Last 338
13.8 Schritt 8 — Der Nulltest, gebaut 342
13.9 Zwei Filme: Aufheizen und der Grenzfall-Umschalter 343
13.10 Die interaktive Einheit: die Dehnbank 345
13.11 Das Kapitel-Programm 352

Roter Faden . . . 352
Übungen . . . 353
Das Kleingedruckte . . . 355

14. Der Träger verbiegt sich: 2D-Thermomechanik **357**
Lernziele . . . 358
14.1 Schritt 1 — Zwei Zahlen pro Knoten . . . 359
14.2 Schritt 2 — Die drei Dehnungen: ein Messprotokoll der Verformung . . . 361
14.3 Schritt 3 — Die Querkontraktion: das seitliche Ausweichen . . . 362
14.4 Schritt 4 — Vom Dreieck zum Steckbrief: die 6×6-Matrix . . . 365
14.5 Schritt 5 — Die thermische Kraft in der Fläche . . . 367
14.6 Schritt 6 — Lagerung, und was ohne sie geschieht . . . 368
14.7 Schritt 7 — Das Finale: der Träger krümmt sich . . . 370
14.8 Zwei Filme: die drei Dehnungen und das Krümmen . . . 374
14.9 Die interaktive Einheit: die Biegebank . . . 376
14.10 Das Kapitel-Programm . . . 384
Roter Faden . . . 385
Übungen . . . 385
Das Kleingedruckte . . . 388

15. Das große Finale **389**
Lernziele . . . 389
15.1 Die Werksbesichtigung: das Programm als Fließband . . . 390
15.2 Die Blackbox aus Kapitel 1 — jetzt mit offenem Motorraum . . . 394
15.3 Die Experiment-Werkstatt . . . 396
15.4 Das Finale in Bewegung . . . 404
15.5 Die interaktive Einheit: dein Finale . . . 405
15.6 Die Landkarte, alle Häkchen gesetzt . . . 415
15.7 Grenzen und Ausblick: die Landkarte des Weiterlernens . . . 416
Schlusswort . . . 419
Roter Faden . . . 419
Übungen . . . 420
Das Kleingedruckte . . . 422

VI. Teil VI — Vertiefung: Die Methode wandert (Biologie) **423**

16. Der Knochen als Träger: FEM in der Biologie **425**
Lernziele . . . 426
16.1 Der Transfer, ausdrücklich . . . 427
16.2 Die 2D-Idealisierung, ehrlich . . . 428
16.3 Das ortsabhängige Material — die neue Idee . . . 428
16.4 Assemblieren mit variablem E . . . 432

16.5 Lasten, Lager — und die Rechnung . . . 433
16.6 Probier es aus: die Knochenwerkstatt . . . 439
16.7 Ausblick: Warum Knochen sich umbauen . . . 447
Roter Faden . . . 450
Übungen . . . 450
Das Kleingedruckte . . . 451

17 Warum Knochen sich umbauen: FEM mit Rückkopplung **453**
Lernziele . . . 453
17.1 Viele statische Rechnungen — keine biologische Uhr . . . 454
17.2 Ein Dreieck speichert Verformungsenergie . . . 455
17.3 Energie je Masse wird zum Reiz . . . 456
17.4 Der schlechte erste Versuch: jedes Element hört nur sich selbst . . . 457
17.5 Die Nachbarschaft hört mit . . . 457
17.6 Aufbau, Abbau und die Totzone . . . 458
17.7 Die vollständige Schleife . . . 459
17.8 Wann ist die Schleife fertig? . . . 462
17.9 Gegen eine unabhängige Rechnung geprüft . . . 463
17.10 Probier es aus: die Remodeling-Werkstatt . . . 463
17.11 Übungen . . . 474
17.12 Was das Modell zeigt — und was nicht . . . 475

Anhang **477**

Glossar **477**
A . . . 477
B . . . 478
D . . . 479
E . . . 481
F . . . 482
G . . . 483
H . . . 484
I . . . 485
I . . . 486
K . . . 486
L . . . 487
M . . . 488
N . . . 488
P . . . 489
Q . . . 490
R . . . 491
S . . . 492
T . . . 494

V 495
W 496
Z 497

Vorwort

Worum es geht

Ein Stahlträger, der linke Auflagerpunkt in der kühlen Außenwand, spannt sich unter einer Betondecke. Eines Tages bricht unter seinem rechten Ende ein Brand aus. Die heißen Brandgase lecken an der Unterseite, die Wärme kriecht Stunde um Stunde am Träger entlang zum kalten Auflager, und über die Höhe des Trägers baut sich ein Temperaturgefälle auf, unter dem er sich zu krümmen beginnt. Woher weiß der Computer, wie warm es in der Mitte des Trägers ist — dort, wo niemand misst? Und woher kennt er die Verformung, lange bevor der Träger je einem Feuer ausgesetzt war?

Die Antwort heißt **Finite-Elemente-Methode (FEM)** — das Rechenverfahren hinter praktisch jeder Ingenieursimulation: Karosserien im Crashtest, Brücken unter Verkehrslast, Handygehäuse beim Sturz, Grundwasser unter einer Baugrube. Dieses Buch erklärt, was dabei unter der Haube passiert — so gründlich, dass du am Ende **jede Zeile eines vollständigen FEM-Programms verstehst**, und so behutsam, dass du dafür **nicht mehr als Schulmathematik** mitbringen musst.

Alles, was darüber hinausgeht, entwickelt das Buch selbst, und zwar genau dann, wenn der Träger es braucht: die Steigung und die Fläche unter einer Kurve (mehr „Analysis" kommt nicht vor), lineare Gleichungssysteme und ihre Löser, Ansatzfunktionen, Randbedingungen, Zeitschritte — bis hin zur Frage, warum sich ein von unten beheizter Träger krümmt wie ein Bimetallstreifen.

Auch **Programmieren wird nicht vorausgesetzt**: Kapitel 2 führt genau so viel Python ein, wie das Buch braucht. Und Python spielt hier eine klar umrissene Rolle — es ist **nie die Erklärung, sondern immer die Unterstützung**. Erklärt wird in Prosa und mit Grafiken; der Code macht die Mathematik anschließend anfassbar: Er druckt das Gleichungssystem zum Anschauen, zeichnet das Temperaturprofil, lässt dich am Regler drehen. Der gesamte Lehrcode ist bewusst schlicht gehalten — reines Python ohne Zusatzbibliotheken, ohne Tricks, mit deutschen, sprechenden Namen — damit du in jedem Moment siehst, dass dort nichts weiter passiert als Listen, Schleifen und Additionen.

Der rote Faden

Das ganze Buch begleitet **einen einzigen Träger** durch **ein einziges Ereignis** — den Brand unter seinem rechten Ende. In Kapitel 1 spielst du mit dem fertigen Simulationsprogramm wie mit einer Blackbox. Dann wird die Box Kapitel für Kapitel geöffnet: Eine Kette aus Federn liefert die komplette FEM-Maschinerie mit reiner Schulalgebra (Kapitel 3), Löser machen große Gleichungssysteme bezwingbar (Kapitel 4), die Wärmeleitung füllt die Maschinerie mit Physik und verfolgt, wie die Hitze der Länge nach zum Auflager wandert (Kapitel 5–9), Dreiecke und Rechtecke tragen die Rechnung in die Fläche und lösen das Gefälle über die Höhe des Trägers auf (Kapitel 10–12), und die Thermomechanik lässt den Träger sich endlich verformen — erst dehnen, dann krümmen (Kapitel 13–14). Im Finale (Kapitel 15) steht dasselbe Programm wie am Anfang vor dir — nur dass du jetzt jede Zeile kennst.

Die Kapitel bauen streng aufeinander auf; das Buch ist zum **linearen Lesen** gedacht.

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 Simulation sind völlig vernünftig und trotzdem falsch („der Computer löst die Gleichung exakt”, „je feiner das Netz, desto richtiger”). Diese Kästchen nehmen die Vermutung ernst, erklären, warum sie naheliegt — und was stattdessen stimmt.
- **Interaktive Programme** — in der HTML-Fassung läuft der Python-Code direkt in deinem Browser: verändern, ausführen, beobachten. Jedes Kapitel endet mit einem kleinen Programm, in dem du genau das ausprobierst, was du gerade gelernt hast.
- **Echte Zahlen** — jedes wichtige Gleichungssystem wird mindestens einmal vollständig hingeschrieben, sodass du jede Zahl nachrechnen kannst.
- **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* (ein Programm gezielt umbauen) und *Übertragen* (die Idee auf ein neues Problem anwenden), alle mit Musterlösungen.

Danksagung und Werkstattbericht

Dieses Buch entsteht im Duett von Mensch und KI: Konzept, fachliche Leitplanken und Qualitätsmaßstab von Sören Gebbert; Ausarbeitung, Programme und Grafiken in enger Zusammenarbeit mit Claude (Anthropic). Strategie und Kapitelpläne sind offen dokumentiert und werden mit dem Buch weiterentwickelt.

Teil I.

Teil I — Ankommen

1. Der Träger im Feuer

Stell dir ein Bürogeschoss vor, kurz nach Feierabend. Unter der massiven Betondecke spannt sich ein Stahlträger von Wand zu Wand, einen Meter lang, eine Handbreit hoch. Sein linkes Ende steckt tief im Auflager der kühlen Außenwand, die es bei rund 20 °C festhält. Und dann, eines Abends, bricht unter seinem rechten Ende ein Feuer aus — ein Papierstapel, ein Regal, ein schwelender Verteilerkasten. Heiße Brandgase lecken von unten an den Träger. Die Wärme kriecht ins Metall: Zuerst wird nur das rechte Ende heiß — bei einem heftigen Brand beginnt es sogar dunkelrot zu glühen —, und Stunde um Stunde schiebt sich die Hitze weiter nach links, dem kalten Auflager entgegen. Weil die Flammen den Träger von unten heizen, die Betondecke ihn aber von oben kühl hält, steht über seine Höhe ein Temperaturgefälle — und darunter beginnt er sich zu krümmen, leise, aber unaufhaltsam, wie ein Bimetallstreifen im Toaster.

Jetzt die Frage, um die es in diesem ganzen Buch geht: **Woher weiß ein Computer, wie warm der Träger in der Mitte ist — dort, wo niemand ein Thermometer hält?** Und woher kennt er die Verbiegung, bevor überhaupt jemand den Träger eingebaut und das Feuer entfacht hat?

Die kurze Antwort passt in einen Satz: Er zerlegt den Träger in viele kleine Stücke, stellt für jedes Stück eine einfache Beziehung auf und löst alle gemeinsam. Diese Stücke heißen **finite Elemente** — endliche, handliche Elemente statt des unendlich fein verschmierten echten Materials. Der ganze Rest des Buches macht nichts anderes, als diesen einen Satz mit Leben zu füllen.

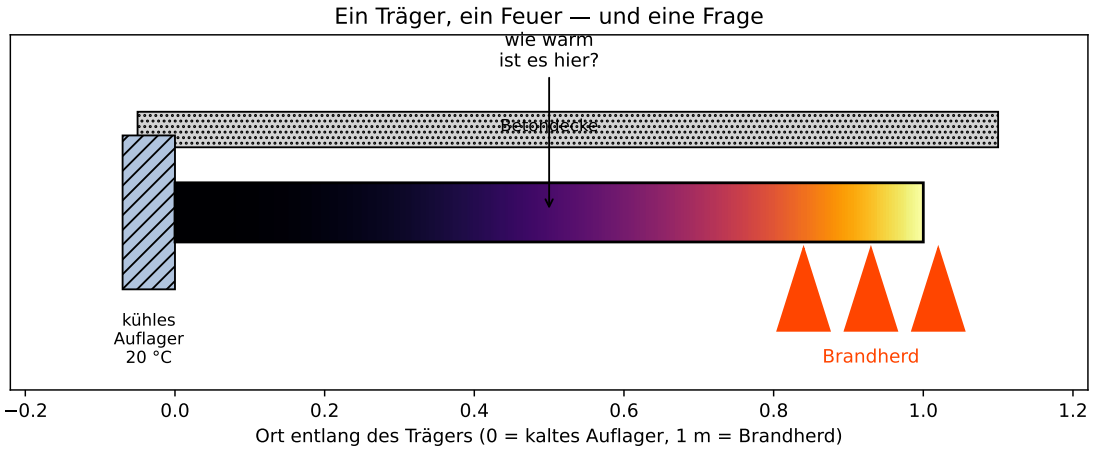

Abbildung 1.1.: Die Buchszene: der Stahlträger, links im kühlen Auflager der Außenwand (20 °C), unter der Betondecke, rechts unten der Brandherd. Die Farben sind der Blick einer Wärmebildkamera — heiß leuchtet hell, kalt bleibt dunkel. Die Frage des Buches steht in der Mitte: dort misst niemand, und trotzdem wollen wir es wissen.

Was man hier sieht: die Szene, die uns fünfzehn Kapitel lang begleitet. Der Farbverlauf ist kein Schmuck, sondern die eigentliche Unbekannte — die Temperatur an jedem Ort, so wie eine Wärmebildkamera sie sähe. Am linken Ende kennen wir sie (das Auflager hält 20 °C), am rechten gibt der Brandherd sie vor; dazwischen soll der Computer sie ausrechnen. Genau dieselbe Szene taucht in jedem Kapitel wieder auf, nur jedes Mal mit etwas mehr geöffneter Motorhaube.

i Warum ein voller Rechteckquerschnitt und kein Doppel-T?

Echte Stahlträger haben fast nie einen vollen rechteckigen Querschnitt. Sie sind **I-Profile** (im Querschnitt wie ein liegendes H): zwei breite Gurte oben und unten, dazwischen ein dünner Steg. Das spart Stahl, ohne viel Tragfähigkeit zu verlieren. Für dieses Buch nehmen wir stattdessen einen **massiven Rechteckquerschnitt**, 0,1 m × 0,1 m. Das ist eine bewusste Vereinfachung: Sie hält die Geometrie und alle Rechnungen sauber, ohne dass am Kern der Methode etwas verloren ginge. Der Weg vom Rechteck zum echten Profil ist danach nur noch Formsache — dieselbe FEM, nur mit einer verwickelteren Form.

Lernziele

Nach diesem Kapitel kannst du …

1. … in eigenen Worten **erklären**, was eine Simulation ist und warum man die Wirklichkeit dafür in kleine Stücke zerlegt,
2. … an drei Beispielen **benennen**, wo die Finite-Elemente-Methode heute steckt (Fahrzeugbau, Elektronik, Brandschutz),
3. … die **fünf Bausteine** jeder FEM-Rechnung aufzählen — Geometrie und Netz, Physik, Randbedingungen, Gleichungssystem, Löser,
4. … am Finale-Programm **vorhersagen**, was ein feineres Netz oder heißere Brandgase qualitativ ändern — und es ausprobieren.

⚠ Naheliegende Vermutung

Vermutung: *„Der Computer löst die Wärmegleichung exakt — er ist ja ein Computer, und Computer rechnen genau.”*
Warum sie naheliegt: Ein Taschenrechner liefert $7 \times 8 = 56$ auf Knopfdruck und ohne Fehler. Warum sollte er bei einem heißen Träger plötzlich nur schätzen?
Was stattdessen stimmt: Der Computer löst gar nicht das echte Problem, sondern ein **Ersatzproblem** aus endlich vielen Zahlen — den Temperaturen an einigen ausgewählten Punkten. Zwischen diesen Punkten rät er (auf sehr kluge Weise). Die ganze Kunst der Methode besteht darin, das Ersatzproblem so zu bauen, dass es der Wirklichkeit nahekommt. „Finite Elemente” ist genau der Name für diese Kunst. Dass ein Ersatzproblem trotzdem verblüffend genau sein kann — und woran man das erkennt — ist eines der wichtigsten Themen des Buches (Kapitel 12).

ℹ Ehrlich gesagt: Stahl brennt nicht — er glüht und wird weich

Der Titel dieses Kapitels ist mit Bedacht gewählt, aber er darf nicht missverstanden werden. **Stahl verbrennt nicht** wie Holz oder Papier, und er schmilzt auch nicht so bald: Baustahl wird erst bei rund 1500 °C flüssig. Was ihm im Brand gefährlich wird, setzt lange vor dem Schmelzen ein und ganz unauffällig — er **verliert seine Festigkeit**. Schon bei etwa 600 °C trägt Stahl nur noch ungefähr die **Hälfte** der Last, die er bei Zimmertemperatur aushält (CEN (Europäisches Komitee für Normung) 2005). Ein Träger versagt im Brand also nicht, weil er wegschmilzt, sondern weil er weich wird und unter seiner Last nachgibt.
Sichtbar wird die Hitze am **Glühen**. Erhitzt man Stahl im Dunkeln, be-

ginnt er ab etwa 550 °C matt zu leuchten, und die Farbe verrät die Temperatur:

Farbe	ungefähre Temperatur
beginnende Dunkelrotglut	~550 °C
Kirschrot	~800 °C
Hellrot bis Orange	~1000 °C
Gelb	~1200 °C
Weiß, dann flüssig	~1500 °C

Deshalb sprechen wir vom „glühenden" Träger nur dort, wo er es wirklich tut — jenseits von rund 550 °C. Und deshalb umhüllt man tragende Stahlteile im Bauwerk mit **Brandschutzplatten** oder dämmenden Anstrichen: Sie verzögern das Aufheizen, sodass der Träger im Ernstfall lange genug kühl und damit tragfähig bleibt. Genau diese Frage — *wie lange hält der Träger?* — treibt das ganze Buch an.

1.1. Das Finale zum Anfassen

Bevor wir irgendetwas erklären, darfst du mit dem **fertigen Ergebnis** spielen. Das folgende Programm ist im Kern dasselbe, das am Ende des Buches steht (3) — nur dass dort die Motorhaube offen ist und du jede Zeile kennst. Hier ist es eine **Blackbox** mit drei Reglern: wie heiß die Brandgase sind, wie fein der Träger zerlegt wird und wie lange das Feuer schon brennt.

Zuerst schauen wir dem Vorgang beim Ablaufen zu. Die folgende Darstellung rechnet die Minuten durch: Man sieht die Wärme vom Brandherd (rechts) ins kühl gehaltene Metall wandern und den Träger sich dabei krümmen (die Verbiegung ist stark überhöht dargestellt, sonst sähe man sie kaum).

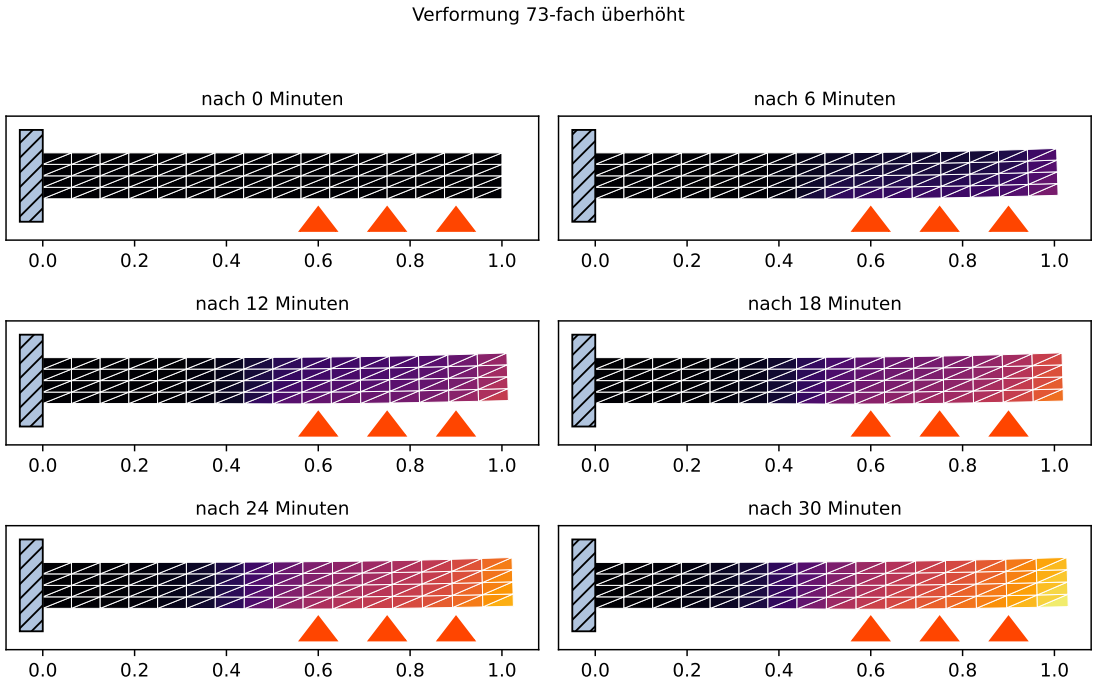

Abbildung 1.2.: Sechs Momentaufnahmen desselben Vorgangs (in der HTML-Fassung eine Animation): Die Wärme kriecht von den Brandgasen ins Metall, der Träger — eine Fläche aus Dreiecken — wölbt sich nach oben. Verformung stark überhöht (Faktor im Bild).

Was man sieht: Nach wenigen Minuten ist erst die äußerste rechte Ecke warm, der ganze Rest liegt noch kalt im Auflager. Stahl leitet Wärme nämlich eher zäh — und so reicht die spürbare Hitze selbst nach einer halben Stunde erst gut eine Handbreit, rund 16 cm, in den Träger hinein. Die Wärme *kriecht*. Die Verbiegung folgt ihr auf dem Fuß: Nur wo es heiß ist, krümmt sich das Metall. Beides zusammen — die kriechende Wärme und die daraus folgende Verformung — ist das Ziel des Buches.

Vorhersage-Punkt

Bevor du das Programm laufen lässt: Gleich darfst du an der Temperatur der Brandgase drehen. Angenommen, du machst sie doppelt so heiß — wird sich der Träger dann **doppelt so stark** verbiegen, oder anders? Lege dich fest, bevor du es ausprobierst.

Und nun die Blackbox selbst. In der HTML-Fassung läuft der folgende Code direkt in deinem Browser — und zwar keine vereinfachte Vorschau, sondern

die **echte Rechnung des großen Finales** (3) in kompakter Form: Der Träger ist eine Fläche aus kleinen **Dreiecken**, die Brandgase wärmen ihn von unten rechts, die Betondecke kühlt ihn von oben, und aus dem Temperaturfeld folgt die Verformung. Ändere oben die drei Werte, drücke auf Ausführen und schau, was passiert — den „Maschinenraum" darunter musst du jetzt noch nicht verstehen, den öffnet das Buch Kapitel für Kapitel.

```
#| autorun: true
# ====== Die drei Regler — hier darfst du drehen ======
BRANDGAS_TEMPERATUR_GRAD = 500.0  # wie heiß die Brandgase sind (200 bis 800)
NETZ = "mittel"                   # "grob", "mittel" oder "fein"
ZEIT_MINUTEN = 30.0               # wie lange das Feuer schon brennt
# =====================================================
# Der Maschinenraum darunter ist die NumPy-Kurzfassung des
# Finale-Programms aus Kapitel 15 (fem_balken.py): Der Träger ist eine
# echte Fläche aus Dreiecken, gerechnet werden die volle
# 2D-Wärmeleitung der Brand-Szene und daraus die Verformung. Das
# Rechenpaket NumPy dient nur als schneller Rechenknecht für den
# Browser — das Buch baut dieselbe Maschine in reinem Python nach,
# Kapitel für Kapitel.

import numpy as np

LAENGE, HOEHE, TIEFE = 1.0, 0.1, 0.1                        # Meter
LEITFAEHIGKEIT, DICHTE, KAPAZITAET = 50.0, 7850.0, 460.0   # Baustahl S235
E_MODUL, QUERKONTRAKTION, AUSDEHNUNG = 210e9, 0.30, 12e-6
UEBERGANG, LUFT_GRAD, BRANDHERD_AB = 25.0, 20.0, 0.5
ZEITSCHRITTE = 10
NETZDICHTEN = {"grob": (8, 2), "mittel": (16, 4), "fein": (32, 8)}


def baue_netz(nx, ny):
    """Knoten und Dreiecke eines strukturierten Netzes (Kapitel 10)."""
    x_werte = np.linspace(0.0, LAENGE, nx + 1)
    y_werte = np.linspace(0.0, HOEHE, ny + 1)
    gitter_x, gitter_y = np.meshgrid(x_werte, y_werte)
    knoten = np.column_stack([gitter_x.ravel(), gitter_y.ravel()])
    dreiecke = []
    for zeile in range(ny):
        for spalte in range(nx):
            ecke = zeile * (nx + 1) + spalte
            dreiecke.append((ecke, ecke + 1, ecke + nx + 2))
            dreiecke.append((ecke, ecke + nx + 2, ecke + nx + 1))
    return knoten, np.array(dreiecke)
```

```
def zeltsteigungen(ecken):
    """Steigungen und Fläche eines Dreiecks (Kapitel 10)."""
    x, y = ecken[:, 0], ecken[:, 1]
    b = np.array([y[1] - y[2], y[2] - y[0], y[0] - y[1]])
    c = np.array([x[2] - x[1], x[0] - x[2], x[1] - x[0]])
    doppelte_flaeche = x[0] * b[0] + x[1] * b[1] + x[2] * b[2]
    return b / doppelte_flaeche, c / doppelte_flaeche, doppelte_flaeche / 2.0


def waermefeld(t_gas, minuten, nx, ny):
    """Transiente 2D-Wärmeleitung der Brand-Szene (Kapitel 6 bis 11)."""
    knoten, dreiecke = baue_netz(nx, ny)
    n = len(knoten)
    leitmatrix = np.zeros((n, n))
    speicher = np.zeros(n)
    for drei in dreiecke:
        b, c, flaeche = zeltsteigungen(knoten[drei])
        beitrag = LEITFAEHIGKEIT * TIEFE * flaeche * (np.outer(b, b)
                                                      + np.outer(c, c))
        leitmatrix[np.ix_(drei, drei)] += beitrag
        speicher[drei] += DICHTE * KAPAZITAET * TIEFE * flaeche / 3.0

    # Ränder der Brand-Szene (Kapitel 8/15): links kühles Auflager
    # (20 °C fest), oben Betondecke und unten links kühle Luft, unten
    # rechts und an der Stirn heiße Brandgase (Wärmeübergang).
    unten = np.arange(nx + 1)
    oben = ny * (nx + 1) + np.arange(nx + 1)
    links = np.arange(ny + 1) * (nx + 1)
    rechts = links + nx
    stueck_x = LAENGE / nx * TIEFE
    stueck_y = HOEHE / ny * TIEFE

    def randstuecke(anzahl, stueck):
        werte = np.full(anzahl, stueck)
        werte[0] = werte[-1] = stueck / 2.0
        return werte

    uebergang_diagonale = np.zeros(n)
    uebergang_last = np.zeros(n)
    for kante, flaechen, umgebung in [
            (oben, randstuecke(nx + 1, stueck_x), np.full(nx + 1, LUFT_GRAD)),
            (rechts, randstuecke(ny + 1, stueck_y), np.full(ny + 1, t_gas)),
```

```
            (unten, randstuecke(nx + 1, stueck_x),
             np.where(knoten[unten, 0] >= BRANDHERD_AB, t_gas, LUFT_GRAD))]:
        uebergang_diagonale[kante] += UEBERGANG * flaechen
        uebergang_last[kante] += UEBERGANG * flaechen * umgebung

    fest = np.zeros(n, dtype=bool)
    fest[links] = True
    frei = ~fest

    temperaturen = np.full(n, LUFT_GRAD)
    dt = minuten * 60.0 / ZEITSCHRITTE
    system = leitmatrix.copy()
    system[np.arange(n), np.arange(n)] += speicher / dt + uebergang_diagonale
    system_frei = system[np.ix_(frei, frei)]
    system_rand = system[np.ix_(frei, fest)]
    for schritt in range(ZEITSCHRITTE):
        rechte_seite = speicher / dt * temperaturen + uebergang_last
        b_frei = rechte_seite[frei] - system_rand @ temperaturen[fest]
        temperaturen[frei] = np.linalg.solve(system_frei, b_frei)
    return knoten, dreiecke, temperaturen


def verformung(knoten, dreiecke, temperaturen):
    """2D-Thermoelastik: Verschiebungen aus der Wärme (Kapitel 13/14)."""
    n = len(knoten)
    steifigkeit = np.zeros((2 * n, 2 * n))
    kraefte = np.zeros(2 * n)
    faktor = E_MODUL / (1.0 - QUERKONTRAKTION**2)
    material = faktor * np.array(
        [[1.0, QUERKONTRAKTION, 0.0],
         [QUERKONTRAKTION, 1.0, 0.0],
         [0.0, 0.0, (1.0 - QUERKONTRAKTION) / 2.0]])
    for drei in dreiecke:
        b, c, flaeche = zeltsteigungen(knoten[drei])
        b_tabelle = np.zeros((3, 6))
        b_tabelle[0, 0::2] = b
        b_tabelle[1, 1::2] = c
        b_tabelle[2, 0::2] = c
        b_tabelle[2, 1::2] = b
        element = b_tabelle.T @ material @ b_tabelle * TIEFE * flaeche
        delta_t = temperaturen[drei].mean() - LUFT_GRAD
        vordehnung = np.array([AUSDEHNUNG * delta_t,
                               AUSDEHNUNG * delta_t, 0.0])
```

```
        element_kraefte = (b_tabelle.T @ (material @ vordehnung)
                           * TIEFE * flaeche)
        dof = np.empty(6, dtype=int)
        dof[0::2] = 2 * drei
        dof[1::2] = 2 * drei + 1
        steifigkeit[np.ix_(dof, dof)] += element
        kraefte[dof] += element_kraefte

    # Lager: die linke Kante steckt fest im Auflager (Kapitel 14).
    fest = np.zeros(2 * n, dtype=bool)
    linke_kante = np.where(np.isclose(knoten[:, 0], 0.0))[0]
    fest[2 * linke_kante] = True
    fest[2 * linke_kante + 1] = True
    frei = ~fest
    verschiebungen = np.zeros(2 * n)
    verschiebungen[frei] = np.linalg.solve(
        steifigkeit[np.ix_(frei, frei)], kraefte[frei])
    return verschiebungen


nx, ny = NETZDICHTEN[NETZ]
knoten, dreiecke, temperaturen = waermefeld(
    BRANDGAS_TEMPERATUR_GRAD, ZEIT_MINUTEN, nx, ny)
verschiebungen = verformung(knoten, dreiecke, temperaturen)

hebung = verschiebungen[1::2][np.isclose(knoten[:, 0], LAENGE)].max()
mitte = temperaturen[np.argmin((knoten[:, 0] - 0.5)**2
                               + (knoten[:, 1] - 0.05)**2)]
print("Netz: %s (%d Knoten, %d Dreiecke), Brandgase: %.0f °C, Zeit: %.0f min"
      % (NETZ, len(knoten), len(dreiecke), BRANDGAS_TEMPERATUR_GRAD,
         ZEIT_MINUTEN))
print("Temperatur in der Trägermitte:  %.0f °C" % mitte)
print("Heißester Punkt (unten rechts): %.0f °C" % temperaturen.max())
print("Hebung der freien Spitze:       %.2f mm (nach oben)"
      % (hebung * 1000.0))

# --- Abbildung ---
import matplotlib.pyplot as plt
import matplotlib.tri as mtri

betrag = np.abs(verschiebungen)
ueber = 0.3 * HOEHE / betrag.max() if betrag.max() > 0 else 1.0
vx = knoten[:, 0] + ueber * verschiebungen[0::2]
```

```
vy = knoten[:, 1] + ueber * verschiebungen[1::2]
netz_verformt = mtri.Triangulation(vx, vy, dreiecke)

fig, (ax_flaeche, ax_profil) = plt.subplots(
    2, 1, figsize=(8.5, 6.0), gridspec_kw={"height_ratios": [3, 2]})
bild = ax_flaeche.tripcolor(netz_verformt, temperaturen, cmap="inferno",
                            vmin=LUFT_GRAD, vmax=temperaturen.max())
ax_flaeche.triplot(netz_verformt, color="white", lw=0.4, alpha=0.5)
ax_flaeche.add_patch(plt.Rectangle((0.0, 0.0), LAENGE, HOEHE, fill=False,
                                   edgecolor="0.4", ls="--", lw=1.0))
ax_flaeche.add_patch(plt.Rectangle((-0.05, -0.05), 0.05, HOEHE + 0.10,
                                   hatch="///", facecolor="lightsteelblue",
                                   edgecolor="black"))
ax_flaeche.add_patch(plt.Rectangle((-0.05, HOEHE + 0.025), LAENGE + 0.10,
                                   0.028, facecolor="0.82", edgecolor="0.5"))
for flamme in (0.60, 0.75, 0.90):
    ax_flaeche.fill([flamme - 0.04, flamme, flamme + 0.04],
                    [-0.075, -0.018, -0.075], color="orangered")
ax_flaeche.set_xlim(-0.08, LAENGE + 0.08)
ax_flaeche.set_ylim(-0.10, HOEHE + 0.09)
ax_flaeche.set_aspect("equal")
ax_flaeche.set_yticks([])
ax_flaeche.set_title("Der Träger als Fläche aus %d Dreiecken – "
                     "Verformung %.0f-fach überhöht"
                     % (len(dreiecke), ueber))
fig.colorbar(bild, ax=ax_flaeche, pad=0.02, shrink=0.85,
             label="Temperatur (°C)")

unterkante = np.isclose(knoten[:, 1], 0.0)
oberkante = np.isclose(knoten[:, 1], HOEHE)
ax_profil.plot(knoten[unterkante, 0], temperaturen[unterkante], "o-",
               color="tab:red", ms=3, label="Unterkante (Brandseite)")
ax_profil.plot(knoten[oberkante, 0], temperaturen[oberkante], "s-",
               color="tab:blue", ms=3, label="Oberkante (Betondecke)")
ax_profil.set_xlabel("Ort entlang des Trägers (m)")
ax_profil.set_ylabel("Temperatur (°C)")
ax_profil.set_title("Wie weit die Wärme gekrochen ist – unten heißer als oben")
ax_profil.legend()
ax_profil.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()
```

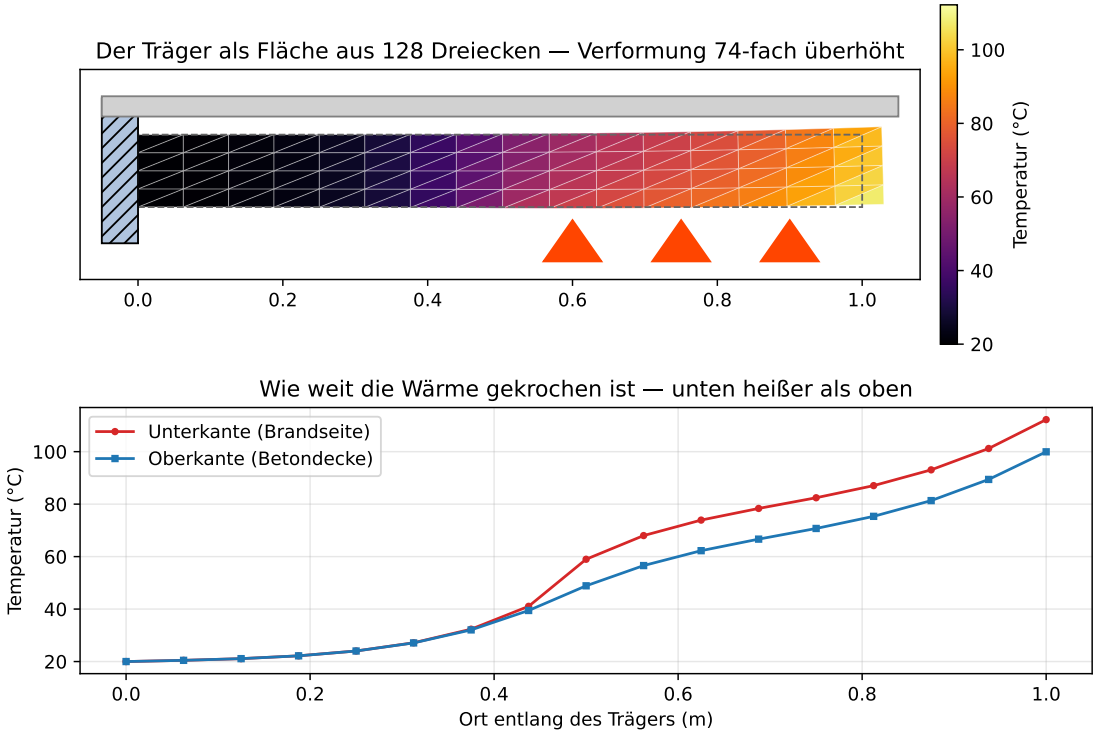

Abbildung 1.3.: Vorgerenderte Fassung der Blackbox (Brandgase 500 °C, mittleres Netz, 30 Minuten): oben das Temperaturfeld auf dem überhöht verformten Dreiecksnetz — derselbe Träger wie im großen Finale (Kapitel 15) —, unten die Temperatur längs der Unter- und der Oberkante. In der HTML-Fassung ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst.

Zur Vorhersage von eben: Heißere Brandgase geben im gleichen Takt mehr Durchbiegung — genauer: Die Verformung wächst hier proportional zur **Übertemperatur** der Brandgase, also zu ihrem Abstand von den 20 °C des kalten Auflagers. Verdoppelst du diesen Abstand, verdoppelt sich auch die Verbiegung. Das ist kein Zufall und keine Selbstverständlichkeit; warum es so ist (und wann es *nicht* mehr stimmt), klärt sich erst in Kapitel 13. Fürs Erste zählt nur: Du hast einen Regler bewegt und eine Wirkung gesehen. Genau dieses Ausprobieren ist der rote Faden.

> **Ehrlich gesagt: was hier rechnet — und was nicht**
>
> Diese Blackbox ist keine Attrappe: Sie rechnet die volle 2D-Finite-Elemente-Kette des großen Finales (3) — Wärmeleitung über die Fläche,

Brandgase und Betondecke als Wärmeübergang am Rand, und daraus die Verformung. Weil die Flammen den Träger von unten heizen und die Decke ihn von oben kühl hält, ist seine Unterseite wärmer als seine Oberseite; die warme Unterseite dehnt sich stärker, und der Träger wölbt sich nach oben — wie ein Bimetallstreifen. Das ist genau die Physik, die die Kapitel 13 und 14 entwickeln.
Zwei Dinge sind trotzdem ehrlich zu sagen. Erstens rechnet der Maschinenraum hier mit dem fertigen Rechenpaket **NumPy** — als flinkem Rechenknecht für deinen Browser. Verstehen kann man daran nichts; darum baut das Buch dieselbe Maschine Kapitel für Kapitel in reinem Python nach, bis du in 3 jede Zeile kennst. Zweitens bleibt es eine grobe Rechnung auf einem kleinen Netz — wie fein das Netz sein muss, damit man den Zahlen trauen darf, ist die Kernfrage von Kapitel 12.

1.2. Was ist eigentlich eine Simulation?

Eine Simulation lässt ein Stück Wirklichkeit im Rechner ablaufen, bevor (oder ohne dass) es in echt passiert. Das Kunststück dabei ist immer dasselbe und besteht aus drei Schritten:

1. **Zerlegen.** Die Wirklichkeit ist unendlich fein — jeder noch so kleine Punkt des Trägers hat seine eigene Temperatur. Damit kann ein Rechner nicht umgehen; er kennt nur endlich viele Zahlen. Also teilen wir den Träger in eine überschaubare Zahl kleiner **Stücke** und merken uns die Temperatur nur an deren Ecken, den **Knoten**.
2. **Lokal rechnen.** Für ein einzelnes kleines Stück ist die Physik einfach: Wärme fließt von warm nach kalt, und wie viel, hängt nur von den paar Nachbarknoten ab. Diese einfache lokale Beziehung schreibt man für jedes Stück getrennt auf — immer nach demselben Bauplan.
3. **Zusammensetzen.** Die vielen kleinen lokalen Beziehungen teilen sich ihre Knoten und ergeben zusammengefügt ein großes **Gleichungssystem**. Löst man es, kennt man die Temperatur an allen Knoten auf einen Schlag.

Diese Denkweise ist nicht neu und nicht auf Träger beschränkt. Eine **Wettervorhersage** arbeitet genauso: Sie legt ein Gitter über die Landkarte, kennt für jede Gitterzelle Temperatur, Druck und Wind, und rechnet für jede Zelle aus, was ihre Nachbarn ihr im nächsten Zeitschritt zuschieben. Kein Meteorologe löst die Wirklichkeit „exakt" — er löst ein Ersatzproblem auf einem Gitter. Feineres Gitter, schärfere Vorhersage; gröberes Gitter, schnellere Rechnung. Genau dieser Kompromiss begegnet uns beim Träger wieder.

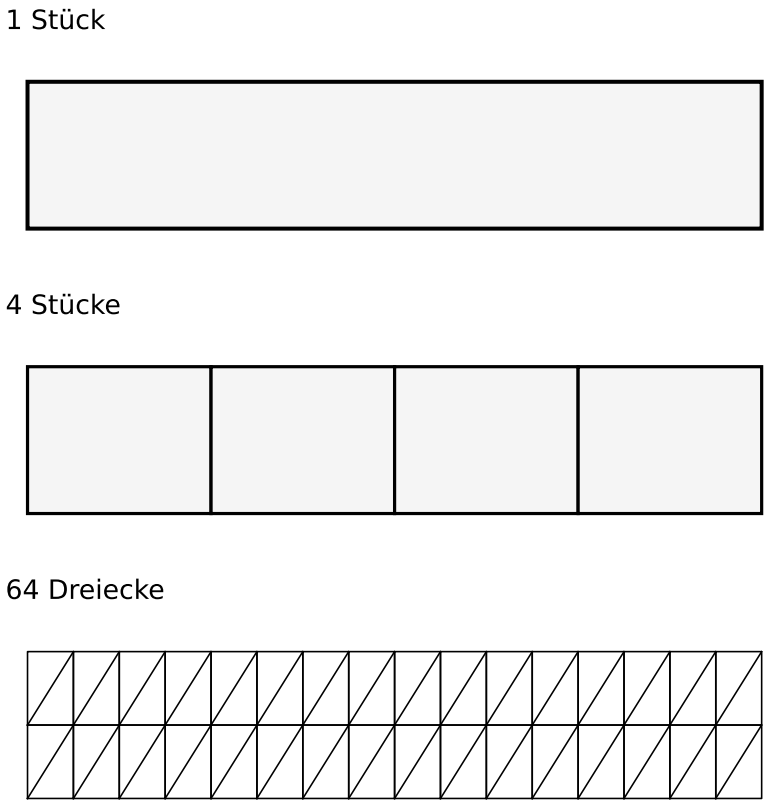

Abbildung 1.4.: Derselbe Träger, drei Mal zerlegt: als ein einziges Stück (viel zu grob), in vier Stücke (die Idee wird sichtbar) und in 64 Dreiecke (so fein, dass die Form kaum noch leidet). Je mehr Stücke, desto näher am echten Träger — und desto mehr zu rechnen.

Was man hier sieht: Ein einziges Stück kann den Temperaturverlauf nicht abbilden — es kennt ja nur seine Ecken. Mit vier Stücken erahnt man die Kurve schon, mit vielen Dreiecken schmiegt sich das Netz eng an die Wirklichkeit. Das ist die zentrale Stellschraube jeder Simulation, und du hast sie eben am Regler „NETZ” selbst bedient.

Dass ein feineres Netz die Lösung glatter und näher an der Wirklichkeit macht, sieht man am Temperaturprofil unseres Trägers unmittelbar:

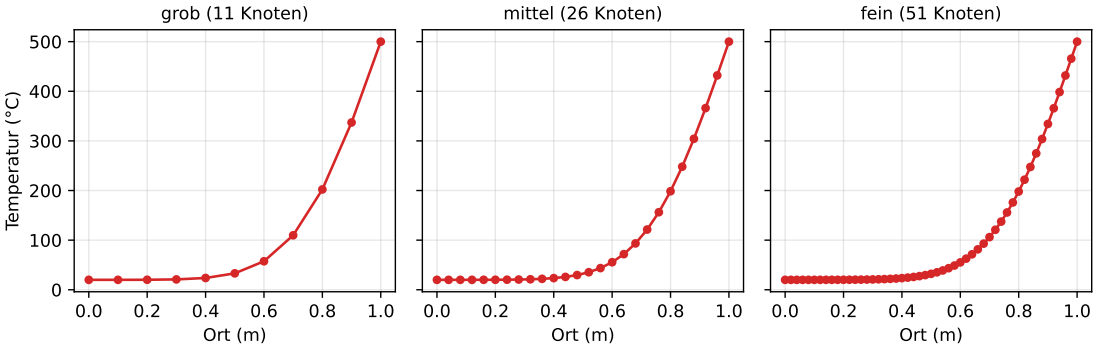

Abbildung 1.5.: Dasselbe Wärmeproblem mit grobem, mittlerem und feinem Netz gerechnet (Brandgase 500 °C, 30 Minuten). Die groben Punkte treffen die kriechende Wärmefront nur eckig; je mehr Knoten, desto glatter und vertrauenswürdiger die Kurve.

Was man hier sieht: Alle drei Kurven erzählen dieselbe Geschichte — rechts heiß, links kalt —, aber das grobe Netz zeichnet sie mit wenigen groben Ecken, das feine mit einer glatten Linie. Weil Stahl die Wärme nur zäh weiterreicht, drängt sich der ganze Anstieg im rechten Viertel zusammen; genau dort trennt sich grob von fein. Für ein erstes Gefühl reicht grob; wer der Zahl in der Mitte trauen will, braucht fein. Wann „fein genug" erreicht ist, ist eine eigene Kunst (Kapitel 12).

1.3. Wo die finiten Elemente überall stecken

Der Träger im Feuer ist ein Lehrbeispiel — aber die Methode dahinter ist eine der meistgenutzten Rechentechniken der Ingenieurwelt. Drei Beispiele:

Das Auto. Wenn ein neues Automodell in den Crashtest geht, ist der teuerste Teil längst gelaufen — im Rechner. Seit den 1990er Jahren ist die Crash-Simulation Industriestandard: Ein komplettes Fahrzeugmodell besteht heute aus mehreren Millionen finiten Elementen, bei Spitzenmodellen sind es rund zehn Millionen, und ein Aufprall wird in winzigen Zeitschritten durchgerechnet. So sieht man, wo sich die Karosserie falten muss, damit die Insassen geschützt bleiben — bevor das erste Blech gebogen ist.

Das Handy. Warum überlebt das eine Gehäuse den Sturz auf den Gehweg und das andere nicht? Bevor ein Smartphone gebaut wird, lässt man es im Rechner tausendfach fallen. Aus verschiedenen Höhen, auf verschiedene

Ecken, nach genormten Falltests: Die Simulation zeigt, wo im Glas die Spannung sich staut und wo eine Platine reißt. Ein paar Zehntelmillimeter mehr Rahmen an der richtigen Stelle — im Rechner gefunden, nicht im Reklamationslager.

Der Brandschutz. Genau unsere Szene ist ein eigener Ingenieurberuf. Bevor ein Stahltragwerk gebaut wird, muss jemand nachweisen, dass es im Brandfall lange genug steht, damit alle Menschen hinauskommen und die Feuerwehr löschen kann. Dieser **Brandschutznachweis** — oft mit finiten Elementen gerechnet — verfolgt, wie schnell sich die Träger erwärmen und wann sie ihre Tragfähigkeit verlieren. Das Ergebnis steckt in Kürzeln, die man an jeder Baustelle findet: **R30**, **R60**, **R90** stehen für *Widerstand* (englisch *resistance*) über 30, 60 oder 90 Minuten. Ein Träger der Klasse R90 muss unter Last und Feuer mindestens anderthalb Stunden durchhalten. Ob eine Beplankung dick genug für R90 ist oder nicht, entscheidet oft genau so eine Rechnung, wie wir sie in diesem Buch aufbauen.

Allen dreien ist gemeinsam: Ein reales Objekt wird in finite Elemente zerlegt, für jedes Element gilt eine einfache Physik, und aus den vielen kleinen Beziehungen wird ein großes Gleichungssystem, das der Computer löst. Wer diesen Kern am Träger verstanden hat, versteht ihn überall.

1.4. Die fünf Bausteine und die Landkarte des Buches

Jede FEM-Rechnung — ob Träger, Auto oder Brücke — steht auf denselben **fünf Bausteinen**:

1. **Geometrie und Netz** — die Form des Objekts und ihre Zerlegung in Elemente.
2. **Physik** — das Naturgesetz, das in jedem Element gilt (bei uns: die Wärmeleitung, später die Verformung).
3. **Randbedingungen** — was am Rand vorgegeben ist (links das kühle Auflager mit 20 °C, rechts die heißen Brandgase).
4. **Gleichungssystem** — die vielen lokalen Beziehungen, zu einem großen System zusammengesetzt.
5. **Löser** — das Verfahren, das dieses Gleichungssystem knackt.

Diese fünf Bausteine sind zugleich die Landkarte des Buches: Jedes kommende Kapitel füllt einen von ihnen mit Leben. Abbildung 1.6 zeigt, welches Kapitel welchen Baustein liefert.

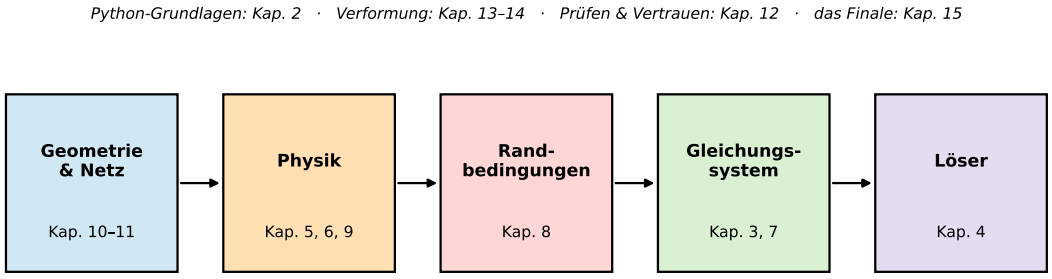

Abbildung 1.6.: Die Landkarte des Buches: die fünf Bausteine jeder FEM-Rechnung und die Kapitel, die sie liefern. Diese Grafik kehrt am Ende jedes Teils wieder und markiert dann, wie weit die Reise gediehen ist.

Was man hier sieht: die Reiseroute. Von links nach rechts entsteht eine FEM-Rechnung; von Kapitel zu Kapitel füllen wir die Kästen. Auffällig: Der Baustein „Gleichungssystem" kommt schon in Kapitel 3 — lange vor der Physik. Das ist Absicht. Wir bauen die komplette FEM-Maschinerie zuerst an einer **Kette aus Federn**, wo reine Schulalgebra genügt, und lassen die schwierigere Physik erst später einziehen.

Ein einziger Ausblick auf die Physik sei erlaubt, mehr Formel kommt in diesem Kapitel nicht vor. Das Naturgesetz hinter dem wandernden Farbband heißt Fouriersches Gesetz und sagt schlicht, wie stark die Wärme fließt:

$$\text{Wärmestrom} \;=\; -\lambda \cdot (\text{Temperaturgefälle}).$$

Das λ ist die Wärmeleitfähigkeit (bei unserem Stahl $50\ \mathrm{W/(m\,K)}$), das Gefälle misst, wie schnell sich die Temperatur mit dem Ort ändert. Fällt die Temperatur zum Beispiel um $280\ \mathrm{K}$ über den ganzen Meter — von 300 °C am beflammten Ende auf 20 °C am Auflager —, so ist das Gefälle $280\ \mathrm{K/m}$ und der Wärmestrom $50 \cdot 280 = 14\,000\ \mathrm{W/m^2}$, also 14 Kilowatt pro Quadratmeter. Durch die Querschnittsfläche unseres Trägers ($0{,}01\ \mathrm{m^2}$) fließen damit rund $140\ \mathrm{W}$. Das Minuszeichen sagt nur: Die Wärme fließt bergab, vom Heißen zum Kalten. Diesen einen Satz zu einer vollständigen Träger-Rechnung auszubauen, ist die Aufgabe von Kapitel 5 bis Kapitel 9.

1.5. Woher die Methode kommt

Die Finite-Elemente-Methode ist keine Erfindung des Computerzeitalters allein, sondern eine Antwort auf ein sehr konkretes Problem: **den Flugzeugflügel der 1950er Jahre**. Schnellere, dünnere Tragflächen ließen sich mit den Formeln der klassischen Statik nicht mehr zuverlässig berechnen — ihre Form war zu unregelmäßig. Ingenieure bei Boeing und an der Universität Berkeley, unter ihnen M. J. Turner und Ray Clough, kamen Mitte der 1950er auf denselben Gedanken wie wir eben: Zerlege den komplizierten Flügel in viele kleine, einfache Dreiecke, für die man rechnen *kann*, und setze das Ganze wieder zusammen (Turner u. a. 1956). Den Namen „finite element" prägte Clough wenig später (Clough 1960).

Aus dieser Not entstanden fast alle Begriffe, die uns begegnen werden — Steifigkeit, Element, Assemblierung. Warum sie gerade so heißen und wie aus der Idee eine Methode wurde, erzählt Kapitel 3 ausführlich. Für jetzt genügt: Die Methode wurde erfunden, weil jemand ein reales Ding ausrechnen musste, das zu kompliziert für eine einzige Formel war. Genau davor stehen wir mit dem Träger im Feuer.

Merkkasten

- Eine Simulation **zerlegt** die Wirklichkeit in kleine Stücke, rechnet **lokal** und **setzt** die Ergebnisse zu einem großen Gleichungssystem zusammen.
- Der Computer löst nicht das echte Problem, sondern ein **Ersatzproblem** aus endlich vielen Zahlen — die Kunst ist, es der Wirklichkeit nahe zu bauen.
- Die Stücke heißen **finite Elemente**; ein **feineres Netz** ist genauer, aber teurer.
- Jede FEM-Rechnung steht auf fünf Bausteinen: **Geometrie und Netz, Physik, Randbedingungen, Gleichungssystem, Löser** — die Landkarte des Buches.

Roter Faden

Dieses Kapitel hat die Reise nur gezeigt, nicht angetreten. Alles, was du hier als Blackbox erlebt hast, wird geöffnet: Das **Netz** und seine Elemente bauen wir ab Kapitel 3 und in der Fläche ab 1; die **Physik** der Wärme kommt in Kapitel 5 bis Kapitel 9; die **Randbedingungen** — das kühle Auflager und die Brandgase — bekommen in Kapitel 8 ihren Auftritt; das **Gleichungssystem** und sein **Löser** sind das Thema von Kapitel 3 und 2; und warum sich

der Träger überhaupt **verbiegt**, klärt sich in Kapitel 13 und Kapitel 14. Im Finale (3) steht dann genau dieses Programm wieder vor dir — mit offener Motorhaube.

Übungen

Ü 1.1 (Verstehen). Ordne die fünf Bausteine einer **Brückensimulation** zu: Was ist hier jeweils (a) die Geometrie und das Netz, (b) die Physik, (c) die Randbedingungen, (d) das Gleichungssystem, (e) der Löser? Beantworte es in eigenen Worten, ohne Rechner.

> **i** Musterlösung zu Ü 1.1
>
> Eine mögliche Zuordnung: (a) **Geometrie und Netz** — die Form der Brücke (Träger, Pfeiler, Fahrbahn), zerlegt in viele kleine Elemente; (b) **Physik** — die Gesetze der Statik, die in jedem Element Kräfte und Verformung verknüpfen; (c) **Randbedingungen** — die Brücke ist an den Pfeilern im Boden verankert (dort keine Bewegung), und obendrauf drücken Verkehr und Wind; (d) **Gleichungssystem** — alle lokalen Kraft-Verformungs- Beziehungen zusammengesetzt; (e) **Löser** — das Rechenverfahren, das ausrechnet, wie stark sich jeder Knoten verschiebt. Dieselben fünf Bausteine wie beim Träger, nur größer.

Ü 1.2 (Verändern). Führe in der Blackbox oben **zwei Experimente** durch und notiere jeweils deine Beobachtung. (a) Lass NETZ und ZEIT fest und stelle die Brandgase nacheinander auf 300, 500 und 800 °C. Wie ändern sich die Mittentemperatur und die Hebung der Spitze? (b) Lass die Brandgase und ZEIT fest und schalte NETZ von “grob” über “mittel” auf “fein”. Ändert sich die Hebung stark? Und die Temperaturkurven?

> **i** Musterlösung zu Ü 1.2
>
> **(a)** Mit heißeren Brandgasen steigen beide Werte, und zwar im gleichen Takt: Verdoppelt man die Übertemperatur der Brandgase (also den Abstand zu 20 °C), verdoppeln sich auch Mittentemperatur-Anstieg und Hebung. Von 300 auf 800 °C wächst die Hebung von rund 0,09 auf rund 0,24 mm — auf knapp das Dreifache, genau wie die Übertemperatur von 280 auf 780 K. Das liegt daran, dass in dieser Rechnung alles **linear** von der Temperatur abhängt — eine bewusste Vereinfachung, deren Grenzen Kapitel 13 bespricht.
> **(b)** Überraschung: Die Hebung wächst beim Verfeinern deutlich — von

rund 0,07 mm (“grob”) über 0,15 mm (“mittel”) auf 0,20 mm (“fein”). Das Netz ist also kein harmloser Schönheitsregler, sondern verändert das Ergebnis: Grobe Dreiecksnetze sind bei Biegung zu steif und nähern sich der wahren Verformung **von unten**. Auch die Temperaturkurven werden feiner gezeichnet, ändern sich aber kaum noch — die Wärme ist gutmütiger als die Verformung. Welche Zahl man am Ende glauben darf und wie man das prüft, ist so wichtig, dass ihr Kapitel 12 ganz gewidmet ist (und Kapitel 14 erklärt die zu steifen Dreiecke).

Ü 1.3 (Übertragen). Denk dir ein eigenes Simulationsproblem aus deinem Alltag aus — eine Bratpfanne auf der Herdplatte, das Eis in einem Getränk, die Wärmedämmung einer Hauswand. Zerlege es in die fünf Bausteine: Was ist die Geometrie, was die Physik, was sind die Randbedingungen? Du musst nichts ausrechnen, nur benennen.

Musterlösung zu Ü 1.3 (Beispiel Bratpfanne)

Geometrie und Netz: die Pfanne (Boden und Griff), zerlegt in kleine Stücke. **Physik:** Wärmeleitung im Metall, wie beim Träger. **Randbedingungen:** Der Boden liegt auf der heißen Herdplatte (dort eine hohe Temperatur vorgegeben), das Griffende hängt in der Zimmerluft. **Gleichungssystem und Löser:** wie gehabt — alle lokalen Beziehungen zusammensetzen und lösen. Ergebnis: wie warm der Griff wird, ohne ihn anzufassen. Es ist im Kern dieselbe Rechnung wie beim Träger im Feuer.

Das Kleingedruckte

Dieses Kapitel hat bewusst mehr versprochen als bewiesen. Das Finale-Programm ist eine **Vorschau**: Sein Wärmeteil rechnet echt, aber grob; sein Biege-Teil schätzt die Verformung mit einer stark vereinfachten Formel, die nur ein Gefühl vermitteln soll. Auch die Annahme über den Temperaturunterschied zwischen Unter- und Oberseite und die lineare Kopplung von Wärme und Verformung sind Näherungen, die wir erst ab Kapitel 13 durch die echte Rechnung ersetzen. Ebenso ist der massive Rechteckquerschnitt eine Vereinfachung gegenüber dem echten I-Profil, und wir behandeln alle Materialkennwerte als temperaturunabhängig — obwohl Stahl in Wahrheit mit steigender Hitze weicher wird, wie der Kasten weiter oben schon andeutet. Nichts davon musst du jetzt glauben — du sollst es am Ende des Buches selbst herleiten und einordnen können. Bis dahin gilt die einfachste aller Regeln: erst ausprobieren, dann verstehen.

2. Python von Null

In Kapitel 1 hast du an einer Blackbox gedreht: drei Regler für die Brandgase, das Netz und die Zeit, und darunter ein „Maschinenraum", den du noch nicht verstehen musstest (er kürzt mit dem Rechenpaket NumPy ab, damit dein Browser flott rechnet). Das Buch baut genau diese Maschine nun in reinem Python nach — ein Programm von ein paar hundert Zeilen, das im großen Finale (3) komplett vor dir liegt. Und jetzt kommt das Versprechen, das dir die Angst nehmen soll: **Jede einzelne dieser Zeilen benutzt nur, was du in diesem Kapitel lernst. Nicht mehr.** Keine geheime Zutat, kein höheres Programmierstudium — Variablen, Listen, Schleifen, ein paar Funktionen. Das ist alles.

Dieses Kapitel ist deshalb kein Programmierkurs, sondern ein Werkzeugkasten. Wir bringen genau so viel Python bei, wie das Buch braucht — und keine Fingerübung ist umsonst: **Jedes Beispiel rechnet schon etwas am Träger aus.** Wir zählen keine erfundenen Zahlen zusammen, sondern die Temperaturen an den fünf Knoten unseres Stahlträgers. So lernst du das Werkzeug immer an dem Ding kennen, für das du es später brauchst.

Lernziele

Nach diesem Kapitel kannst du …

1. … **Variablen**, Zahlen und Texte benutzen und mit `print` formatiert ausgeben — eine saubere Temperaturtabelle drucken,
2. … **Listen** anlegen, lesen und verändern — als Bild für „die Temperatur an jedem Knoten",
3. … mit **`for`-Schleifen** Summen bilden und alle Knoten ablaufen,
4. … mit **`if`** Fälle unterscheiden — etwa Randknoten von inneren Knoten,
5. … kleine **Funktionen** mit Docstring schreiben und aufrufen,
6. … eine **verschachtelte Liste** als Tabelle lesen und ausdrucken — die Form, in der später die Gleichungssysteme stehen.

⚠ Naheliegende Vermutung

Vermutung: *„Um Simulationen zu programmieren, muss man ein Mathe- oder Informatik-Profi sein."*
Warum sie naheliegt: Das fertige Programm aus Kapitel 1 sieht kompliziert aus, und „Finite-Elemente-Methode" klingt nach einem Ingenieurstudium. Wer noch nie programmiert hat, vermutet dahinter Zauberei.
Was stattdessen stimmt: Der Kern unseres FEM-Programms besteht aus **Listen, Schleifen und Additionen** — genau den Werkzeugen dieses Kapitels. Die Klugheit der Methode steckt nicht im Code, sondern in der *Idee*, den Träger in Stücke zu zerlegen. Ist die Idee erst da, ist der Code erstaunlich schlicht. Das ganze Buch ist der Beweis: Am Ende liest du jede Zeile des Finales — und wunderst dich, wie wenig dahintersteckt.

2.1. Variablen: Dingen einen Namen geben

Ein Programm rechnet mit Werten — mit der Länge des Trägers, mit einer Temperatur, mit einer Anzahl von Knoten. Damit man diese Werte nicht jedes Mal neu hinschreiben muss und damit man sie später wiedererkennt, gibt man ihnen einen **Namen**. Ein solcher benannter Wert heißt **Variable**.

Stell dir ein beschriftetes Marmeladenglas vor: Außen klebt ein Schild („Erdbeere"), innen ist der Inhalt. Die Variable ist das Schild, der Wert ist der Inhalt. Und so schreibt man das in Python auf:

```
laenge_m = 1.0
hoehe_m = 0.1
temperatur_links_grad = 20.0
temperatur_rechts_grad = 300.0
```

Das Gleichheitszeichen ist hier **kein** „ist gleich" wie in der Mathematik. Es bedeutet: *„Nimm den Wert rechts und häng ihm das Schild links an."* Ab jetzt steht der Name `laenge_m` überall für die Zahl `1.0`. Wir haben damit die Daten unseres Trägers benannt — links kühl bei 20 °C im Auflager, ein Meter lang, eine Handbreit (0,1 m) hoch. Das rechte Ende, das der Brandherd aufheizt, setzen wir auf 300 °C: die Temperatur in der **Frühphase** des Feuers und zugleich der Standardwert des **Handbeispiels**, mit dem das Buch alle Rechnungen vorführt (in der Blackbox aus Kapitel 1 durftest du die Brandgase ja beliebig heiß drehen).

Zwei Sorten von Werten begegnen uns dabei: **Zahlen** wie `1.0` oder `20.0`, mit denen man rechnet, und **Text** wie `"Knoten"`, der immer in Anführungszeichen steht und einfach das ist, was dort geschrieben steht. Die Regel, deutsche sprechende Namen zu verwenden (`laenge_m` statt `L`), ist Absicht: Der Code soll sich lesen wie eine Beschreibung dessen, was er tut. Nur eine Kleinigkeit: In Python-Namen dürfen keine Umlaute stehen, darum schreiben wir `laenge` statt „länge" und `waermster` statt „wärmster".

Mit Variablen rechnet man wie mit den Zahlen selbst. Unser Träger hat fünf Knoten (das Standard-Beispiel des Buches); der **Abstand** zwischen zwei benachbarten Knoten ergibt sich aus Länge und Knotenzahl:

```
knotenzahl = 5
abstand_m = laenge_m / (knotenzahl - 1)
print(abstand_m)
```

```
0.25
```

Warum `knotenzahl - 1`? Fünf Knoten spannen **vier** gleich lange Abschnitte auf — wie fünf Zaunpfähle vier Zaunfelder ergeben. Ein Meter auf vier Felder verteilt macht 0,25 m pro Feld. Genau diese 0,25 m — der Knotenabstand h des Handbeispiels — wird uns durch das ganze Buch begleiten.

Das Wörtchen `print` in der letzten Zeile ist der Befehl, mit dem ein Programm etwas „laut ausspricht", also auf den Bildschirm schreibt. Ohne `print` rechnet das Programm zwar, sagt aber nichts. Und weil sauberes Ausdrucken in diesem Buch ein wichtiges Werkzeug ist — wir werden ganze Gleichungssysteme als Text drucken —, schauen wir uns `print` genauer an.

2.2. Sauber drucken mit f-Strings

Ein nacktes `print(abstand_m)` schreibt bloß `0.25` hin, ohne zu sagen, was das ist. Schöner wäre ein ganzer Satz mit der Zahl mittendrin. Dafür gibt es den **f-String** — man kann ihn sich als *Lückentext* vorstellen: ein Satz mit Lücken, in die Werte eingesetzt werden.

Man setzt ein kleines `f` direkt vor das Anführungszeichen. Das `f` signalisiert: In diesem Text stecken Lücken in geschweiften Klammern `{…}`, und in jede Lücke kommt der Wert, der drinsteht.

```
knoten_nummer = 3
temperatur_grad = 160.0
print(f"Knoten {knoten_nummer} ist {temperatur_grad} Grad warm.")
```

```
Knoten 3 ist 160.0 Grad warm.
```

Python ersetzt `{knoten_nummer}` durch `3` und `{temperatur_grad}` durch `160.0` und druckt den fertigen Satz. Der Lückentext bleibt lesbar, und die Zahlen stehen an ihrem Platz.

Für Tabellen brauchen wir noch eine Kleinigkeit: Zahlen sollen sauber **untereinander** stehen. Dafür schreibt man hinter die Variable in der Lücke einen Doppelpunkt und ein Format. `{temperatur_grad:6.1f}` heißt: „sechs Zeichen breit, eine Nachkommastelle". Die feste Breite sorgt dafür, dass alle Zahlen rechtsbündig in einer Spalte landen — auch wenn die eine dreistellig und die andere zweistellig ist.

```
print(f"{20.0:6.1f}")
print(f"{160.0:6.1f}")
print(f"{300.0:6.1f}")
```

```
  20.0
 160.0
 300.0
```

Die drei Zahlen stehen ordentlich untereinander, weil jede genau sechs Zeichen breit gedruckt wird. Mehr Formatierung brauchen wir im ganzen Buch nicht — dieses eine Muster `:6.1f` reicht für alle unsere Tabellen.

2.3. Listen: eine Zahl für jeden Knoten

Jetzt kommt das wichtigste Werkzeug des Kapitels. Bisher hatte jede Variable **einen** Wert. Aber unser Träger hat fünf Knoten, und jeder hat seine eigene Temperatur. Fünf einzelne Variablen anzulegen (`temperatur0`, `temperatur1`, …) wäre unhandlich — und bei einem feinen Netz mit hundert Knoten unmöglich.

Die Lösung ist die **Liste**: eine geordnete Reihe von Werten unter einem einzigen Namen. Das Bild dazu ist ein **Eierkarton mit nummerierten Mulden** — ein Behälter, viele Fächer, jedes Fach hält einen Wert. Man schreibt die Werte in eckigen Klammern, durch Kommas getrennt:

```
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]
print(temperaturen)
```

```
[20.0, 20.0, 20.0, 20.0, 300.0]
```

Diese fünf Zahlen sind der **Anfangszustand** unseres Trägers, genau der Moment, in dem Kapitel 1 die Simulation startet: Der Brandherd hat das rechte Ende schon auf 300 °C gebracht — die Frühphase des Feuers —, der ganze Rest liegt noch kühl bei 20 °C, festgehalten vom Auflager. Ein Wert pro Knoten, fünf Knoten, fünf Zahlen in einer Liste. Genau das meint der Fachbegriff **diskretisieren**: die durchgehende Temperatur des echten Trägers durch endlich viele Zahlen an endlich vielen Knoten ersetzen. Die Liste *ist* der diskretisierte Träger.

Abbildung 2.1 macht das sichtbar — es ist die wichtigste Grafik dieses Kapitels, präg sie dir ein:

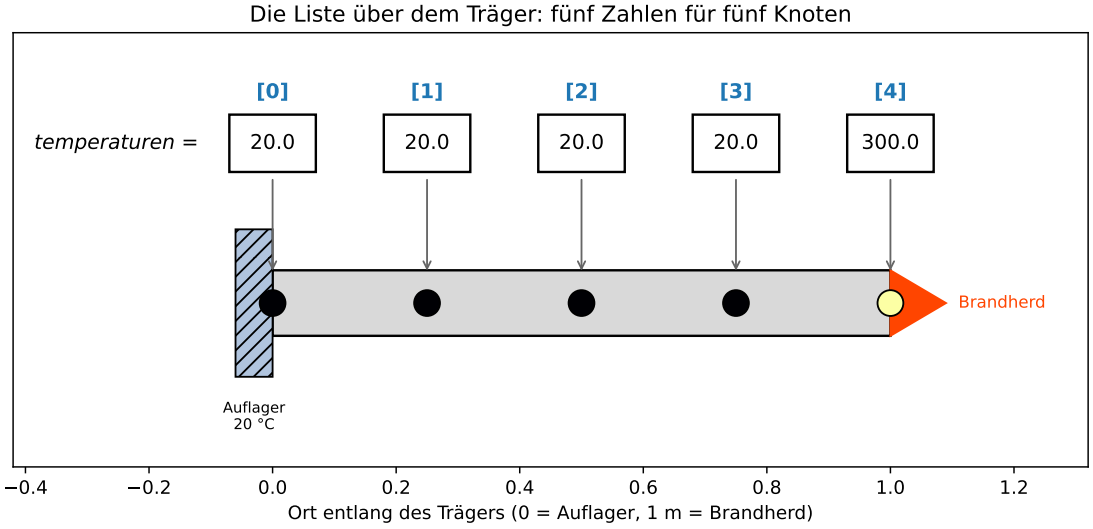

Abbildung 2.1.: Die Liste über dem Träger. Jeder der fünf Listeneinträge gehört zu genau einem Knoten des Trägers: der Wert im Kästchen ist die Temperatur, die eckige Zahl darüber ist der Index. Der linke Rand steckt im kühlen Auflager (20 °C), rechts sitzt der Brandherd (300 °C). Die Farbe der Knotenpunkte zeigt die Temperatur (inferno-Skala wie in Kapitel 1).

Was man hier sieht: Die Liste und der Träger sind dasselbe, nur zwei Mal gezeichnet. Jedes Kästchen sitzt über genau einem Knoten; sein Inhalt ist dessen Temperatur. Die blaue Zahl in eckigen Klammern darüber ist der **Index** — die Nummer des Fachs. Und hier die eine Überraschung, die jeden Anfänger stolpern lässt: **Python zählt ab null.** Der erste Eintrag ist `[0]`, nicht `[1]`.

Das ist weniger seltsam, als es klingt. In einem deutschen Haus wohnt man im **Erdgeschoss**, dann kommt der erste Stock, der zweite und so weiter — das Erdgeschoss ist nicht „der erste Stock", sondern die Nummer null der Etagen. Genauso ist es bei der Liste: Der linke Knoten im Auflager ist Fach `[0]`, der Knoten am Brandherd ist Fach `[4]`. Fünf Fächer, nummeriert von 0 bis 4.

Einen einzelnen Wert holt man mit seinem Index in eckigen Klammern heraus:

```
print(temperaturen[0])
print(temperaturen[4])
```

```
20.0
300.0
```

`temperaturen[0]` ist die 20,0 am kühlen Auflager, `temperaturen[4]` die 300,0 am Brandherd. Wie viele Fächer die Liste hat, verrät `len` (von englisch *length*, Länge):

```
print(len(temperaturen))
```

```
5
```

Fünf Fächer — und weil sie ab 0 nummeriert sind, ist das letzte Fach `temperaturen[4]`, also `temperaturen[len(temperaturen) - 1]`. Diese Zeile werden wir noch oft brauchen: „der letzte Knoten".

Eine Liste ist nicht in Stein gemeißelt — man kann einen Eintrag **überschreiben**, so wie man ein Ei aus einer Mulde nimmt und ein anderes hineinlegt. Sagen wir, der Brand frisst weiter und das rechte Ende steigt auf 350 °C:

```
temperaturen[4] = 350.0
print(temperaturen)
temperaturen[4] = 300.0   # wieder zurueck auf den Buchwert
```

```
[20.0, 20.0, 20.0, 20.0, 350.0]
```

Später, wenn die Wärme durch den Träger wandert, tut unser Programm genau das in jedem Zeitschritt: Es überschreibt die Temperaturen an den inneren Knoten mit ihren neuen Werten. Eine Liste, die sich Schritt für Schritt ändert — mehr passiert bei der ganzen Wärmesimulation nicht.

2.4. Schleifen: jeden Knoten der Reihe nach

Wir haben fünf Temperaturen und wollen etwas mit **jeder** tun — sie drucken, aufsummieren, vergleichen. Das von Hand fünfmal (oder bei feinem Netz hundertmal) hinzuschreiben, wäre mühsam und fehleranfällig. Dafür gibt es die **Schleife**: eine Anweisung „tu das Folgende mit jedem Eintrag der Reihe nach". Das Bild dazu: Du gehst ein Wohnhaus ab und leerst der Reihe nach jeden Briefkasten — dieselbe Handlung, ein Fach nach dem anderen.

In Python heißt diese Schleife `for`. Die einfachste Form nimmt sich direkt jeden Wert der Liste vor:

```
for temperatur in temperaturen:
    print(f"{temperatur:6.1f} °C")
```

```
  20.0 °C
  20.0 °C
  20.0 °C
  20.0 °C
 300.0 °C
```

Man liest das fast wie einen deutschen Satz: *„Für* jede `temperatur` *in* `temperaturen`: drucke sie." Der Name `temperatur` (ohne s) ist eine Variable, die die Schleife bei jedem Durchgang mit dem nächsten Wert füllt: erst 20,0, dann wieder 20,0, … und zuletzt 300,0. Wichtig ist die **Einrückung**: Alles, was eingerückt unter dem `for` steht, gehört zur Schleife und wird bei jedem Durchgang wiederholt. Die Einrückung ist keine Verzierung, sondern sagt Python, *was* wiederholt werden soll.

Abbildung 2.2 zeigt, wie die Schleife über den Träger läuft:

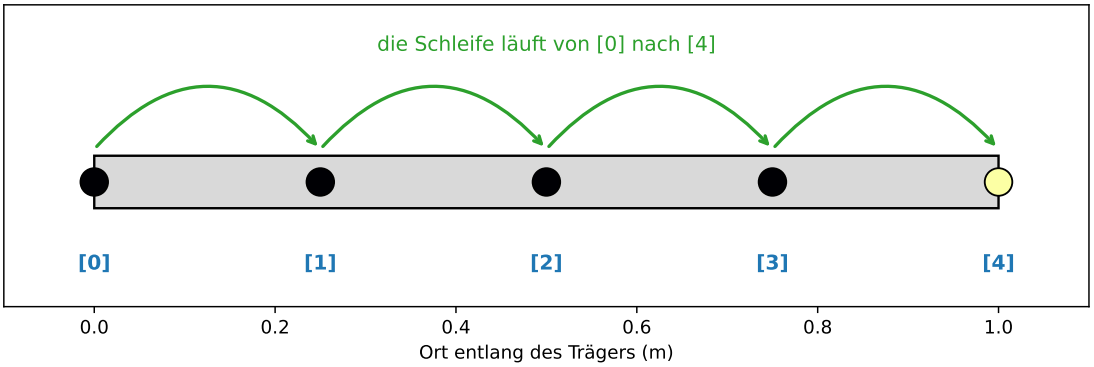

Abbildung 2.2.: Die Schleife läuft die Knoten von links nach rechts ab, ein Fach nach dem anderen — von Index [0] am Auflager bis [4] am Brandherd. Kein abstraktes Flussdiagramm, sondern die Bewegung direkt auf dem Träger.

Was man hier sieht: Die Schleife ist kein Sprung ins Ungewisse, sondern ein geordneter Gang von Knoten zu Knoten. Jeder grüne Bogen ist ein Schleifendurchgang. Am Ende, bei `[4]`, ist die Schleife fertig — sie hat jeden Knoten genau einmal besucht.

2.4.1. Eine laufende Summe

Der häufigste Grund für eine Schleife in diesem Buch ist das **Aufsummieren**. Um den Mittelwert der fünf Temperaturen zu bekommen, müssen wir sie erst alle zusammenzählen. Der Trick heißt **laufende Summe**: ein Zähler, der bei null beginnt und bei jedem Knoten um dessen Temperatur wächst.

Bevor wir das laufen lassen, eine Vorhersage:

> Vorhersage-Punkt
>
> **Bevor du weiterliest:** Die fünf Temperaturen sind `[20, 20, 20, 20, 300]`. Was ist ihre Summe, und was ihr Mittelwert (Summe geteilt durch 5)? Rechne es im Kopf und leg dich fest, bevor das Programm antwortet.

```
summe = 0.0
for temperatur in temperaturen:
    summe = summe + temperatur

mittelwert = summe / len(temperaturen)
```

```
print(f"Summe:      {summe:6.1f}")
print(f"Mittelwert: {mittelwert:6.1f}")
```

```
Summe:       380.0
Mittelwert:   76.0
```

Die Zeile `summe = summe + temperatur` sieht mathematisch unmöglich aus — eine Zahl kann nicht sich selbst plus etwas sein. Aber das Gleichheitszeichen ist ja der „Nimm-rechts, häng-links-an"-Befehl von vorhin: *Nimm den alten Wert von `summe`, addiere die aktuelle Temperatur, und mach das Ergebnis zum neuen `summe`.* Der Zähler klettert so von 0 über 20, 40, 60, 80 auf 380 — und 380 geteilt durch 5 ergibt einen Mittelwert von 76,0 °C. Lag deine Vorhersage daneben? Der große Sprung am Schluss ist die 300 am Brandherd, die den Schnitt hochzieht, obwohl vier von fünf Knoten noch kalt sind.

Diese laufende Summe ist übrigens ein alter Bekannter aus der Mathematik: Das große Summenzeichen Σ, dem wir in späteren Kapiteln oft begegnen, ist nichts anderes als genau diese Schleife, nur auf Papier geschrieben. „$\sum_i T_i$" heißt wörtlich: „lauf mit i über alle Knoten und zähl die T_i zusammen" — Zeile für Zeile dasselbe wie unsere `for`-Schleife.

Die folgende Darstellung zeigt die laufende Summe beim Hochzählen — in der HTML-Fassung als Animation, im Druck als Bildreihe:

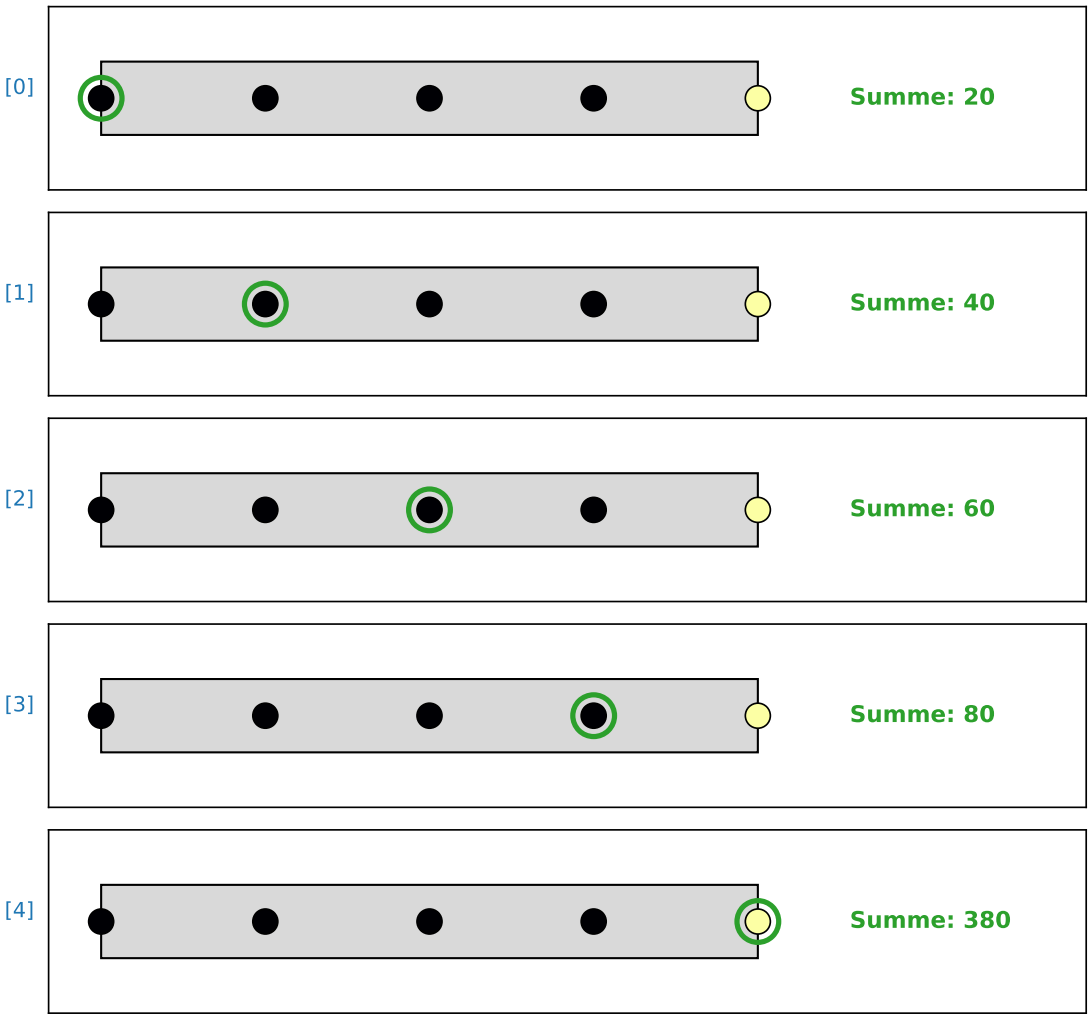

Abbildung 2.3.: Dieselbe Schleife als Bildreihe (in der HTML-Fassung eine Animation): Von oben nach unten wandert der grüne Ring über die Knoten, rechts steht die laufende Summe nach jedem Schritt.

Was man hier sieht: Die Summe wächst in gleichmäßigen Schritten von 20 durch die vier kalten Knoten — und schnellt beim letzten, dem Knoten am Brandherd, von 80 auf 380 hoch. Eine einzige heiße Zahl unter vier kalten prägt das Ergebnis. Genau dieses Gefühl, wie eine Summe sich aus ihren Beiträgen aufbaut, brauchst du, wenn wir später ganze Gleichungssysteme aufaddieren.

2.4.2. Alle Knoten mit ihrem Index

Manchmal genügt der Wert allein nicht — wir brauchen auch die **Nummer** des Knotens, etwa um seinen Ort auf dem Träger auszurechnen (Ort = Index mal Abstand). Dann läuft die Schleife nicht über die Werte, sondern über die **Indizes** 0, 1, 2, 3, 4. Die liefert `range(5)`: „die Zahlen von 0 bis unter 5". Allgemein schreibt man `range(len(temperaturen))`, damit es auch bei mehr Knoten stimmt.

Und damit drucken wir endlich die vollständige Temperaturtabelle — die Zusammenführung von Liste, Schleife und f-String:

```
print("Knoten | Ort (m) | Temperatur")
print("-------+---------+-----------")
for index in range(len(temperaturen)):
    ort_m = index * abstand_m
    print(f"  [{index}]  |  {ort_m:4.2f}   | {temperaturen[index]:6.1f} °C")
```

```
Knoten | Ort (m) | Temperatur
-------+---------+-----------
  [0]  |  0.00   |   20.0 °C
  [1]  |  0.25   |   20.0 °C
  [2]  |  0.50   |   20.0 °C
  [3]  |  0.75   |   20.0 °C
  [4]  |  1.00   |  300.0 °C
```

Innerhalb der Schleife holt `temperaturen[index]` den Wert am aktuellen Knoten, und `index * abstand_m` rechnet dessen Ort aus: Knoten `[0]` sitzt bei 0,00 m, Knoten `[2]` bei 0,50 m, Knoten `[4]` bei 1,00 m — am Brandherd-Ende. So eine Tabelle ist in diesem Buch kein Nebenprodukt, sondern ein Hauptwerkzeug: Wir werden Temperaturen, Kräfte und ganze Matrizen als solche Textblöcke drucken und ihnen ansehen, ob die Rechnung stimmt.

2.5. Entscheidungen treffen mit `if`

Bisher tat die Schleife mit jedem Knoten dasselbe. Oft aber müssen wir **Fälle unterscheiden**: Ein Knoten am Rand des Trägers wird anders behandelt als einer in der Mitte. Der linke Randknoten steckt im Auflager (seine Temperatur ist fest vorgegeben), der rechte am Brandherd — nur die **inneren** Knoten rechnet die Simulation überhaupt aus. Diese Unterscheidung trifft das Schlüsselwort **`if`** (englisch für „falls").

Das Bild dazu ist eine **Weggabelung**: Trifft eine Bedingung zu, geht es den einen Weg (`if`), sonst den anderen (`else`, „sonst"). Ein Knoten ist ein Randknoten, wenn sein Index der erste (`0`) oder der letzte (`len(temperaturen) - 1`) ist:

```
for index in range(len(temperaturen)):
    if index == 0 or index == len(temperaturen) - 1:
        rolle = "Randknoten"
    else:
        rolle = "innerer Knoten"
    print(f"[{index}] {temperaturen[index]:6.1f} °C  ({rolle})")
```

```
[0]   20.0 °C  (Randknoten)
[1]   20.0 °C  (innerer Knoten)
[2]   20.0 °C  (innerer Knoten)
[3]   20.0 °C  (innerer Knoten)
[4]  300.0 °C  (Randknoten)
```

Zwei Dinge sind hier neu. Erstens das **doppelte** Gleichheitszeichen `==`: Es *fragt* „sind die beiden gleich¿', während das einfache `=` etwas zuweist. Der Unterschied ist wichtig — `=` befiehlt, `==` fragt. Zweitens das `or` („oder"): Die Bedingung ist erfüllt, wenn *entweder* der Index 0 *oder* der letzte ist. So bekommen genau die beiden Enden das Schild „Randknoten", die drei mittleren das Schild „innerer Knoten".

Diese Sortierung nach Rand und Innerem ist kein Selbstzweck. Sie ist das Herz der **Randbedingungen**, denen Kapitel 8 ein ganzes Kapitel widmet: An den Randknoten *wissen* wir die Temperatur schon (Auflager und Brandherd), an den inneren müssen wir sie ausrechnen. Genau so wird unser FEM-Programm die Knoten behandeln.

2.6. Funktionen: dem Programm eigene Wörter beibringen

Den Mittelwert einer Liste haben wir oben mit einer Schleife ausgerechnet. Wollen wir das an mehreren Stellen tun, müssten wir die Schleife jedes Mal neu hinschreiben. Besser ist es, sie **einmal** unter einem Namen aufzuschreiben und diesen Namen dann beliebig oft zu benutzen. So ein benannter Rechenweg heißt **Funktion**.

Eine Funktion ist wie ein **Rezept mit einem Namen**: Vorne kommen die Zutaten hinein (die *Argumente*), hinten kommt ein Ergebnis heraus (das,

was hinter `return` steht). Einmal aufgeschrieben, kann man es immer wieder „kochen", mit verschiedenen Zutaten. Man schreibt eine Funktion mit dem Schlüsselwort `def` (von englisch *define*, festlegen):

```
def mittelwert(werte):
    """Berechnet den Mittelwert einer Liste von Zahlen.

    werte: eine Liste mit mindestens einer Zahl.
    Rueckgabe: die Summe aller Eintraege geteilt durch ihre Anzahl.
    """
    summe = 0.0
    for wert in werte:
        summe = summe + wert
    return summe / len(werte)
```

Die erste eingerückte Zeile in dreifachen Anführungszeichen ist der **Docstring** — eine Kurzbeschreibung im Kopf des Rezepts, die sagt, was die Funktion tut und was hinein- und herauskommt. In diesem Buch bekommt **jede** Funktion einen Docstring; er ist die Bedienungsanleitung für den nächsten Leser (oft dich selbst in ein paar Wochen).

Aufgeschrieben ist das Rezept noch nichts wert — man muss es **aufrufen**, also die Zutaten hineingeben:

```
print(mittelwert(temperaturen))
print(mittelwert([100.0, 200.0, 300.0]))
```

```
76.0
200.0
```

Beim ersten Aufruf geben wir unsere Träger-Temperaturen hinein und bekommen die vertrauten 76,0 zurück. Beim zweiten geben wir eine ganz andere Liste hinein und bekommen 200,0 — dieselbe Funktion, andere Zutaten. Das ist die ganze Kraft der Funktion: Einmal richtig gebaut, arbeitet sie für jede Liste.

Funktionen sind die **Kapitel eines Programms**. So wie dieses Buch die große Aufgabe „FEM verstehen" in Kapitel zerlegt, zerlegt ein gutes Programm seine Arbeit in Funktionen mit sprechenden Namen: `baue_elementmatrix`, `loese_gleichungssystem`, `mittelwert`. Wer die Namen liest, versteht das Programm, ohne jede Zeile zu lesen — und genau so werden wir ab Kapitel 7 die Bausteine unseres FEM-Programms bauen, eine Funktion nach der anderen.

2.7. Verschachtelte Listen: Tabellen und Matrizen

Ein letztes Werkzeug, und es folgt unmittelbar aus dem, was du schon kennst. Eine Liste kann Zahlen enthalten — aber sie kann genauso gut *andere Listen* enthalten. Eine **Liste von Listen** ist eine **Tabelle**: eine Liste von Zeilen, und jede Zeile ist selbst eine Liste von Zellen.

Wir schreiben eine kleine 3×3-Tabelle auf. Die Zahlen darin sind mit Bedacht gewählt — es ist genau die Form, in der ab Kapitel 3 unsere Gleichungssysteme stehen werden:

```
tabelle = [
    [ 2, -1,  0],
    [-1,  2, -1],
    [ 0, -1,  2],
]

print(tabelle[0])
print(tabelle[0][1])
```

```
[2, -1, 0]
-1
```

`tabelle[0]` holt die **erste Zeile** heraus — sie ist selbst eine Liste, `[2, -1, 0]`. Und `tabelle[0][1]` geht einen Schritt weiter: erst die erste Zeile, davon das Fach `[1]`, also die `-1`. Zwei Indizes hintereinander, wie **Zeile und Spalte** in einem Rechenblatt: erst runter zur Zeile, dann rüber zur Spalte. Abbildung 2.4 zeigt das Gitter mit seinen Indizes:

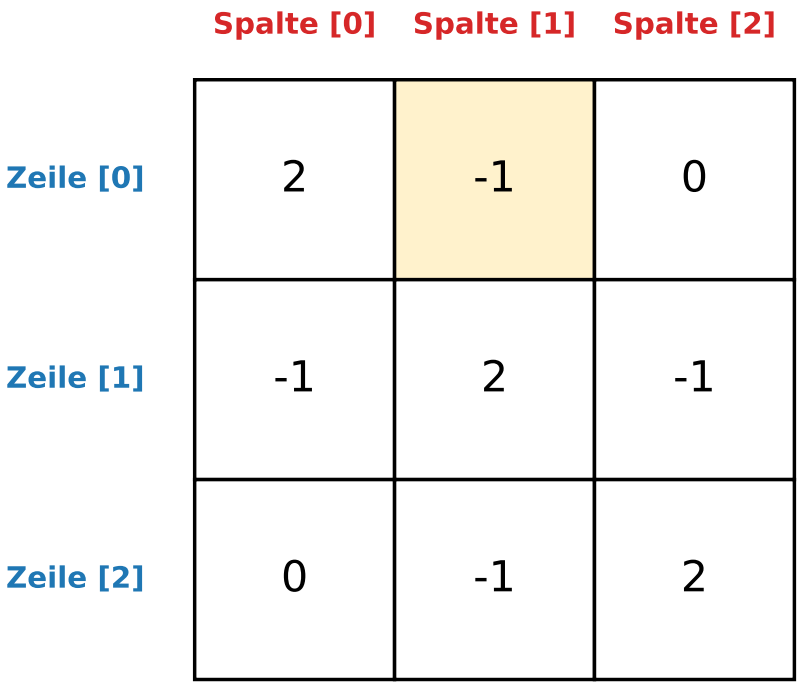

Abbildung 2.4.: Die verschachtelte Liste als beschriftetes Gitter. Ein Eintrag wird durch zwei Indizes angesprochen: erst die Zeile (blau, tabelle[Zeile]), dann die Spalte (rot). tabelle[0][1] ist der Eintrag in Zeile [0], Spalte [1] — hier die -1.

Was man hier sieht: Ein Zahlengitter, in dem jede Zelle über zwei Indizes erreichbar ist — Zeile und Spalte. Das gelb markierte Feld ist `tabelle[0][1]`. Diese Bauform, eine Liste von Zeilen, ist alles, was man braucht, um eine **Matrix** im Rechner abzulegen. Und die konkreten Zahlen hier — 2 auf der Mittellinie, -1 daneben, 0 in den Ecken — sind kein Zufall: Es ist die Gestalt, die das Gleichungssystem einer Federkette in Kapitel 3 annimmt. Jede Zahl wird dort sagen, wie stark zwei Knoten zusammenhängen; die Nullen in den Ecken heißen „diese beiden Knoten spüren einander gar nicht".

Eine Tabelle druckt man mit **zwei** ineinandergeschachtelten Schleifen: die äußere läuft über die Zeilen, die innere innerhalb einer Zeile über die Spalten.

```
for zeile in tabelle:
    for zahl in zeile:
        print(f"{zahl:4d}", end="")
    print()
```

```
   2  -1   0
  -1   2  -1
   0  -1   2
```

Zwei Kleinigkeiten sind neu. Das Format `:4d` heißt „vier Zeichen breit, ganze Zahl" (`d` statt `f`, weil hier keine Nachkommastellen stehen) — es sorgt wieder für saubere Spalten. Und `end=""` am Ende von `print` bedeutet: „mach danach *keine* neue Zeile auf". Normalerweise springt `print` nach jeder Ausgabe in die nächste Zeile; hier aber wollen wir die Zahlen einer Zeile nebeneinander haben. Das leere `print()` nach der inneren Schleife schließt die Zeile dann ab. So erscheint das Gitter als sauberer Textblock — die Bandstruktur der Matrix wird buchstäblich sichtbar, und genau so werden wir ab Kapitel 3 unsere Gleichungssysteme zu Gesicht bekommen.

2.8. Die interaktive Einheit: deine Temperaturtabelle

Jetzt bist du dran. Die folgenden Zellen laufen in der HTML-Fassung **direkt in deinem Browser** — du kannst die Zahlen ändern, auf Ausführen drücken und sofort sehen, was herauskommt. Wir bauen die Temperaturtabelle Stück für Stück aus, genau mit den Werkzeugen dieses Kapitels. Vor jeder Zelle steht eine Vorhersage-Frage: Leg dich fest, *bevor* du ausführst — so merkst du sofort, ob du die Idee schon hast.

Wir starten mit der nackten Liste und drucken jeden Knoten mit seinem Index.

> Vorhersage-Punkt
>
> **Bevor du ausführst:** Welche Zeile wird als **letzte** gedruckt — und welche Zahl steht darin?

```
#| autorun: true
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]

for index in range(len(temperaturen)):
    print(f"Knoten [{index}]: {temperaturen[index]:6.1f} °C")
```

Die erste Modify-Stufe: **Mittelwert**. Häng ans Ende eine laufende Summe und teile durch die Anzahl.

> Vorhersage-Punkt
>
> **Bevor du ausführst:** Welche Zahl erscheint als Mittelwert? (Denk an die vier kalten Knoten und den einen heißen.)

```
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]

summe = 0.0
for temperatur in temperaturen:
    summe = summe + temperatur

print(f"Mittelwert: {summe / len(temperaturen):.1f} °C")
```

Die zweite Stufe: **Maximum**. Wir merken uns den bisher größten Wert und ersetzen ihn, sobald ein größerer auftaucht — genau die Logik, mit der man in einem Stapel Zettel den mit der höchsten Zahl heraussucht.

> Vorhersage-Punkt
>
> **Bevor du ausführst:** Welchen Wert wird das Programm als größten melden?

```
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]

groesster = temperaturen[0]
for temperatur in temperaturen:
    if temperatur > groesster:
        groesster = temperatur

print(f"Wärmster Wert: {groesster:.1f} °C")
```

Die dritte Stufe: **schönere Formatierung** — die vollständige Tabelle mit Ort und Temperatur, ordentlich in Spalten. Probier ruhig aus, den Brandherd (`temperaturen[4]`) höher zu stellen oder das Auflager (`temperaturen[0]`) zu ändern, und schau, wie sich die Tabelle mitbewegt.

> Vorhersage-Punkt
>
> **Bevor du ausführst:** In welcher Zeile steht der Ort `1.00` — und welche Temperatur daneben?

```
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]
abstand_m = 1.0 / (len(temperaturen) - 1)

print("Knoten | Ort (m) | Temperatur")
print("-------+---------+-----------")
for index in range(len(temperaturen)):
    ort_m = index * abstand_m
    print(f"  [{index}]  |  {ort_m:4.2f}   | {temperaturen[index]:6.1f} °C")
```

Alles, was du hier ausprobiert hast — Liste, Schleife, `if`, f-String — steckt später Zeile für Zeile im FEM-Programm. Du hast eben seinen Wortschatz gelernt.

2.9. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript zu diesem Kapitel liegt in `programme/kap02/kap02_temperaturtabelle.py`. Es fasst alles zusammen: Es legt die Knotenliste an, druckt die formatierte Tabelle, berechnet mit Funktionen den Mittelwert und findet den wärmsten Knoten — reine Textausgabe, kein Bild. Sein Kern sind genau die Funktionen und Schleifen dieses Kapitels; wer das Kapitel gelesen hat, versteht jede Zeile. Führe es mit `python kap02_temperaturtabelle.py` aus und vergleiche seine Ausgabe mit den Tabellen oben.

Merkkasten

- Eine **Variable** ist ein Name für einen Wert; `=` heißt „nimm rechts, häng links an”, nicht „ist gleich”.
- Eine **Liste** ist eine Zahl pro Knoten — der diskretisierte Träger. Sie zählt **ab 0**: der erste Eintrag ist `[0]`, der letzte `[len(...) - 1]`.
- Eine **`for`-Schleife** tut mit jedem Knoten dasselbe; die **laufende Summe** (`summe = summe + wert`) ist das Σ der Mathematik in Code.
- **`if/else`** unterscheidet Fälle (Rand oder innen); `==` *fragt*, `=` *weist zu*.
- Eine **Funktion** (`def`, mit Docstring) ist ein benanntes Rezept, das man einmal schreibt und beliebig oft aufruft — die Kapitel des Programms.
- Eine **verschachtelte Liste** ist eine Tabelle (Liste von Zeilen) — die Form jeder Matrix.

Roter Faden

Zurück: Die drei Regler, an denen du in Kapitel 1 gedreht hast, sind nichts anderes als Variablen wie `BRANDGAS_TEMPERATUR_GRAD` — du hast dort schon Werte gesetzt, ohne es Programmieren zu nennen.

Vor: Die **Liste** wird zum ständigen Begleiter — sie hält überall die Knotenwerte, ob Temperatur, Verschiebung oder Kraft. Die **verschachtelte Liste** wird ab Kapitel 3 zur Matrix, dem Herzstück jedes Gleichungssystems; ihr strukturiertes Ausdrucken (2) macht die Bandstruktur sichtbar. Und die **Funktionen** werden ab Kapitel 7 zu den Bausteinen des FEM-Programms: `baue_elementmatrix`, `loese_gleichungssystem`, ein Rezept nach dem anderen, bis im Finale (3) alle zusammenspielen.

Übungen

Ü 2.1 (Verstehen). Lies die folgenden drei Stücke, ohne sie auszuführen, und sag jeweils vorher, was gedruckt wird.

(a)

```
werte = [3.0, 7.0, 2.0]
print(werte[0] + werte[2])
```

(b)

```
summe = 0.0
for zahl in [10.0, 20.0, 30.0]:
    summe = summe + zahl
print(summe)
```

(c)

```
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]
print(len(temperaturen))
print(temperaturen[len(temperaturen) - 1])
```

> **Musterlösung zu Ü 2.1**
>
> (a) **5.0** — `werte[0]` ist 3,0 und `werte[2]` ist 2,0 (das *dritte* Fach, denn gezählt wird ab 0); zusammen 5,0.

(b) **60.0** — die laufende Summe zählt 10, dann 30, dann 60 hoch.

(c) Erst **5** (die Liste hat fünf Fächer), dann **300.0** — das letzte Fach `temperaturen[4]`, der Knoten am Brandherd.

Ü 2.2 (Verändern). Erweitere die Temperaturtabelle aus der interaktiven Einheit um eine vierte Spalte **„Abstand zum Brandherd"**. Der Brandherd sitzt am rechten Ende (Ort 1,00 m), der Abstand eines Knotens ist also `1.0 - ort_m`. Welche Zahl steht in dieser Spalte bei Knoten `[0]`, welche bei Knoten `[4]`?

Musterlösung zu Ü 2.2

In der Schleife eine Zeile ergänzen und die neue Spalte drucken:

```
for index in range(len(temperaturen)):
    ort_m = index * abstand_m
    abstand_brandherd_m = 1.0 - ort_m
    print(f"  [{index}]  |  {ort_m:4.2f}   | {temperaturen[index]:6.1f} °C"
          f" | {abstand_brandherd_m:5.2f} m")
```

Bei Knoten `[0]` (am Auflager) steht **1,00 m**, bei Knoten `[4]` (am Brandherd) **0,00 m** — der Knoten am Brandherd hat den Abstand null zu sich selbst.

Ü 2.3 (Übertragen). Schreibe eine Funktion `waermster_knoten(temperaturen)`, die **beides** zurückgibt: den Index des wärmsten Knotens *und* seinen Wert. (Tipp: Eine Funktion kann mit `return index, wert` zwei Dinge auf einmal zurückgeben.) Rufe sie mit der Trägerliste auf und drucke ein Ergebnis wie „Wärmster Knoten: [4] mit 300.0 °C". Das vollständige Lösungsskript liegt in `loesungen/kap02_ue3.py`.

Musterlösung zu Ü 2.3

```python
def waermster_knoten(temperaturen):
    """Sucht den waermsten Knoten einer Temperaturliste.

    temperaturen: Liste der Knotentemperaturen.
    Rueckgabe: Index des waermsten Knotens und seine Temperatur.
    """
    waermster_index = 0
    for index in range(len(temperaturen)):
        if temperaturen[index] > temperaturen[waermster_index]:
            waermster_index = index
    return waermster_index, temperaturen[waermster_index]


temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]
index, wert = waermster_knoten(temperaturen)
print(f"Wärmster Knoten: [{index}] mit {wert:.1f} °C")
```

Ausgabe: `Wärmster Knoten: [4] mit 300.0 °C`. Die Funktion merkt sich den bisher wärmsten Index und ersetzt ihn, sobald ein größerer Wert auftaucht — dieselbe Logik wie beim Maximum in der interaktiven Einheit, nur gibt sie zusätzlich den Ort zurück.

Das Kleingedruckte

Dieses Kapitel hat Python bewusst **klein** gehalten. Es gibt vieles, was die Sprache noch kann — und was du hier absichtlich *nicht* gesehen hast, weil das Buch es nicht braucht: keine „Klassen", keine „Dictionaries", keine trickreichen Einzeiler, kein NumPy. Wir haben `for`-Schleifen benutzt, aber keine `while`-Schleifen; wir haben Listen von Hand gefüllt, statt sie mit „Comprehensions" in einer Zeile zu erzeugen. Das ist Absicht und folgt der Regel des ganzen Buches: **reines, ehrliches Python**, in dem du jede Zeile lesen kannst und die Mathematik eins zu eins wiederfindest (die Schleife *ist* die Summe, die Liste *ist* der Vektor, die verschachtelte Liste *ist* die Matrix). Erfahrene Programmierer würden manches kürzer schreiben — kürzer ist hier aber nicht das Ziel. Verständlich ist das Ziel. Mit genau diesem Werkzeugkasten bauen wir ab Kapitel 3 die komplette FEM-Maschinerie.

Teil II.

Teil II — Die FEM-Maschinerie ohne Analysis

3. Die Federkette

Häng eine Reihe gleicher Federn hintereinander — wie die Federn in einem alten Bettgestell oder die Glieder einer gedehnten Fahrradkette. Halte das linke Ende fest und zieh am rechten. Was passiert? Das rechte Glied wandert am weitesten, das nächste etwas weniger, und ganz links, am Auflager, bewegt sich nichts. Jedes Glied *spürt* den Zug am Ende, aber jedes verschiebt sich anders weit.

Jetzt die Frage: **Wie rechnet man aus, wie weit sich jedes einzelne Glied verschiebt?** Nicht nur das erste und das letzte, sondern jedes dazwischen.

1952 stand ein Team bei Boeing vor genau dieser Frage — nur ging es nicht um Federn, sondern um einen Flugzeugflügel, und nicht um Verschiebung von Hand, sondern um Hunderte gekoppelte Bauteile. Ihre Antwort benutzt heute jedes FEM-Programm der Welt, und sie ist erstaunlich einfach. In diesem Kapitel bauen wir sie an der Federkette komplett nach — mit nichts als Schulalgebra. Keine Ableitung, kein Integral, keine einzige krumme Formel.

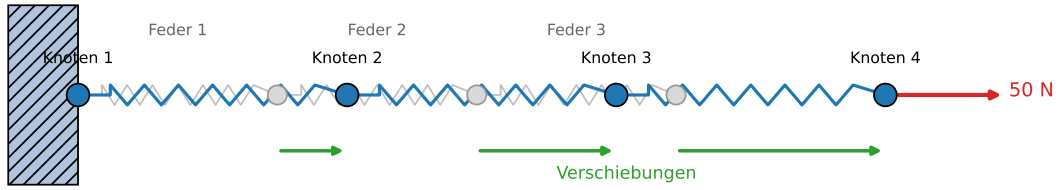

Abbildung 3.1.: Die Federkette des Kapitels: drei gleiche Federn, vier Knoten. Knoten 1 steckt fest im Auflager (links, schraffiert), am Knoten 4 zieht eine Kraft von 50 N nach rechts. Die blassen Umrisse zeigen die Ruhelage, die kräftigen Federn den gezogenen Zustand; die grünen Pfeile sind die Verschiebungen, die wir ausrechnen wollen. Sie wachsen von links nach rechts — je weiter vom Auflager, desto weiter der Weg.

Was man hier sieht: unser Rechenobjekt für dieses ganze Kapitel. Drei Federn, vier Knoten, links fest, rechts gezogen. Bekannt ist die Kraft (50 N)

und dass Knoten 1 sich nicht bewegt; gesucht sind die drei grünen Pfeile — die Verschiebungen der Knoten 2, 3 und 4. Am Ende des Kapitels wirst du sie von Hand ausrechnen können, und du wirst dabei genau die Maschinerie benutzt haben, die auch im Flugzeugflügel steckt.

Lernziele

Nach diesem Kapitel kannst du …

1. … für eine einzelne Feder ihre **2×2-Elementmatrix** aus dem Hookeschen Gesetz aufschreiben,
2. … drei Federn zu einem **4×4-Gleichungssystem** zusammensetzen (*assemblieren*) und erklären, warum sich Nachbar-Federn auf der Diagonale addieren,
3. … eine feste Einspannung als **Randbedingung** ins System einbauen,
4. … das entstehende System **von Hand** auf die freien Knoten reduzieren und lösen,
5. … erklären, **woher** die FEM historisch kommt und was „Element” im Flugzeugbau der 1950er bedeutete.

⚠ Naheliegende Vermutung

Vermutung: *„Je mehr Federn ich anhänge, desto grundlegend komplizierter wird die Rechnung. Bei zwanzig Federn brauche ich eine ganz andere, viel schwierigere Methode als bei zwei.”*
Warum sie naheliegt: Zwanzig gekoppelte Bauteile, jedes zieht am nächsten — das klingt nach einem hoffnungslos verhedderten Knäuel aus Wechselwirkungen, das mit jeder neuen Feder unübersichtlicher wird.
Was stattdessen stimmt: Jede Feder liefert **denselben** winzigen Baustein — eine 2×2-Tabelle mit vier Zahlen. Mehr Federn heißt nur: mehr gleiche Bausteine, an der richtigen Stelle aufaddiert. Der Rechenweg für zwanzig Federn ist Zeile für Zeile derselbe wie für zwei, nur öfter wiederholt. Genau diese stumpfe Gleichförmigkeit ist es, die die FEM computertauglich macht: Ein Computer tut gern tausendmal dasselbe. Wir zeigen das in diesem Kapitel Schritt für Schritt.

3.1. Eine einzelne Feder

Fangen wir so klein an wie möglich: mit **einer** Feder. Sie hat zwei Enden, und an jedem Ende sitzt ein Knoten. Der linke Knoten hat die Verschiebung u_1, der rechte die Verschiebung u_2 — beide gemessen von der Ruhelage aus, in Millimetern. Eine positive Verschiebung heißt „nach rechts gewandert".

Was die Feder tut, hängt nur von *einer* Größe ab: wie weit sie gedehnt ist. Und gedehnt ist sie um den Unterschied der beiden Enden, $u_2 - u_1$. Verschieben sich beide Enden gleich weit, wandert die Feder als Ganzes, ohne sich zu dehnen — dann ist $u_2 - u_1 = 0$, und sie ist kräftefrei.

Das Kraftgesetz einer Feder ist das **Hookesche Gesetz**, und es ist so einfach, wie ein Naturgesetz nur sein kann: Die Kraft ist proportional zur Dehnung.

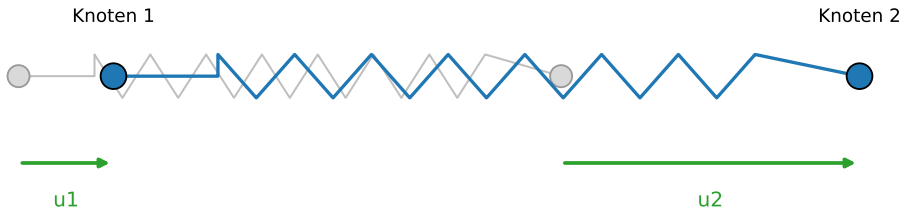

Abbildung 3.2.: Eine einzelne Feder mit ihren beiden Knoten. Links die Ruhelage (blass), rechts der gedehnte Zustand (blau). Der linke Knoten ist um u1 gewandert, der rechte um u2; die Dehnung der Feder ist der Unterschied u2 minus u1. Genau um diesen Unterschied ist die Feder länger geworden.

Was man hier sieht: Nur der *Unterschied* der beiden Verschiebungen dehnt die Feder. Der linke Knoten ist ein Stück nach rechts gewandert (u_1), der rechte weiter (u_2); die Feder ist um $u_2 - u_1$ länger geworden. Verschöben sich beide gleich weit, bliebe die Feder gleich lang.

In Formeln: Zieht man am rechten Knoten, so ist die Kraft F, mit der man ziehen muss, um ihn um $u_2 - u_1$ weiter als den linken zu bringen,

$$F \;=\; k \cdot (u_2 - u_1),$$

wobei k die **Steifigkeit** der Feder ist — die Zahl, die sagt, wie störrisch sie sich der Dehnung widersetzt. Eine steife Feder hat ein großes k, eine weiche

ein kleines. Für unsere Federn nehmen wir durchweg $k = 100$ N/mm: Um eine Feder um einen Millimeter zu dehnen, braucht man 100 Newton. Dehnt man sie um $0{,}5$ mm, so ist die Kraft $100 \cdot 0{,}5 = 50$ N. Genau diese Zahl wird gleich wiederkehren.

3.2. Das Kräftegleichgewicht an einem Knoten

Eine einzelne Feder ist noch keine Kette. Der interessante Ort ist ein Knoten, an dem **zwei** Federn zusammentreffen — sagen wir Knoten 2, zwischen Feder 1 (links) und Feder 2 (rechts). Dieser Knoten wird von beiden Seiten gezogen, und die entscheidende Regel der ganzen Statik lautet: **Ein Knoten, der in Ruhe ist, steht unter ausgeglichenen Kräften.** Alles, was von links zieht, wird von rechts genau aufgewogen. Sonst würde er sich ja bewegen.

Schauen wir uns diese Kräfte einzeln an. Feder 1 hängt zwischen Knoten 1 und 2; sie ist um $u_2 - u_1$ gedehnt und zieht darum an Knoten 2 mit der Kraft $k\,(u_2 - u_1)$ **nach links zurück** (eine gedehnte Feder zieht ihre Enden zueinander). Feder 2 hängt zwischen Knoten 2 und 3; sie ist um $u_3 - u_2$ gedehnt und zieht an Knoten 2 mit $k\,(u_3 - u_2)$ **nach rechts**.

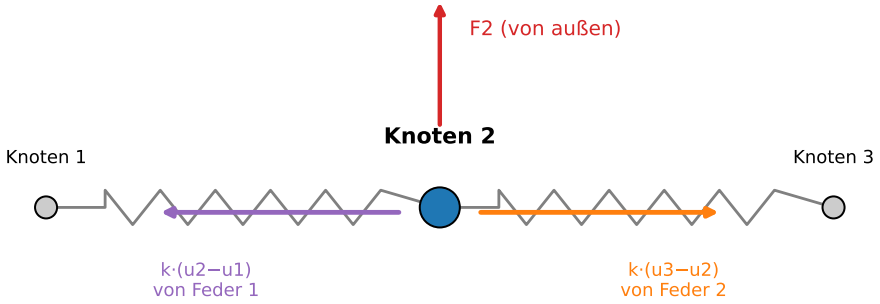

Abbildung 3.3.: Der Mittelknoten 2 freigeschnitten: Feder 1 (links) zieht ihn mit k·(u2−u1) nach links zurück, Feder 2 (rechts) zieht ihn mit k·(u3−u2) nach rechts, und von außen wirkt eine Knotenkraft F2. Ist der Knoten in Ruhe, heben sich alle drei genau auf. Die Pfeillängen sind schematisch.

Was man hier sieht: drei Kräfte an einem Knoten. Zwei kommen von den Federn (lila und orange), eine kommt von außen (rot) — etwa der Zug, den

wir aufbringen. Die Statik verlangt, dass sie sich aufheben. Schreiben wir das hin: Die äußere Kraft F_2 muss tragen, was die beiden Federn dem Knoten antun. Sortiert man die Feder-Kräfte danach, welche Verschiebung sie enthalten, ergibt sich

$$F_2 \;=\; -\,k\,u_1 \;+\; 2k\,u_2 \;-\; k\,u_3.$$

Hier steht schon fast die ganze FEM. Lies die Gleichung als Satz: Die Kraft am Knoten 2 setzt sich zusammen aus einem Beitrag seines linken Nachbarn ($-k\,u_1$), einem doppelten Beitrag von sich selbst ($+2k\,u_2$ — weil an Knoten 2 *zwei* Federn ziehen) und einem Beitrag seines rechten Nachbarn ($-k\,u_3$). Die $2k$ auf dem eigenen Platz ist kein Zufall, sondern die Handschrift eines Knotens, der zwischen zwei Federn hängt. Merke sie dir — sie ist es, die gleich in der großen Tabelle auf der Diagonale auftaucht.

3.3. Der Steckbrief der Feder: die 2×2-Elementmatrix

Bevor wir die ganze Kette bauen, halten wir fest, was *eine* Feder für ihre zwei Knoten bedeutet — ihren **Steckbrief**. Eine Feder verknüpft zwei Knoten, und zu jedem Knoten gehört eine Gleichung: die Kraft an diesem Knoten. Für die Feder zwischen Knoten 1 und 2 sind das genau zwei Zeilen, F_1 und F_2:

$$\begin{aligned} F_1 &= \;\; k\,u_1 - k\,u_2, \\ F_2 &= -k\,u_1 + k\,u_2. \end{aligned}$$

Die erste Zeile sagt: Zieht der rechte Knoten weg (u_2 groß), so wird der linke Knoten nach rechts gezogen — mit umgekehrtem Vorzeichen. Die zweite Zeile ist ihr Spiegelbild. Beide zusammen sind bloß das Hookesche Gesetz $F = k\,(u_2 - u_1)$, einmal von jedem Knoten aus betrachtet.

Diese zwei Gleichungen kann man **ordentlich als Tabelle** aufschreiben — genau so, wie wir in Kapitel 2 verschachtelte Listen als Zahlengitter gedruckt haben. Wir notieren nur noch die Vorfaktoren der Verschiebungen, in einem Quadrat aus vier Zahlen:

$$\mathbf{K}_{\text{Feder}} \;=\; \begin{pmatrix} k & -k \\ -k & k \end{pmatrix} \;=\; \begin{pmatrix} 100 & -100 \\ -100 & 100 \end{pmatrix}.$$

Das ist die **Elementmatrix** einer Feder — ihr vollständiger Steckbrief. Das Wort *Matrix* soll dich nicht schrecken: Es ist nichts als ein ordentlich aufgeschriebenes Gleichungssystem, eine verschachtelte Liste mit zwei Zeilen

und zwei Spalten, wie du sie längst kennst. Jede Zeile gehört zu einem Knoten, jede Spalte auch.

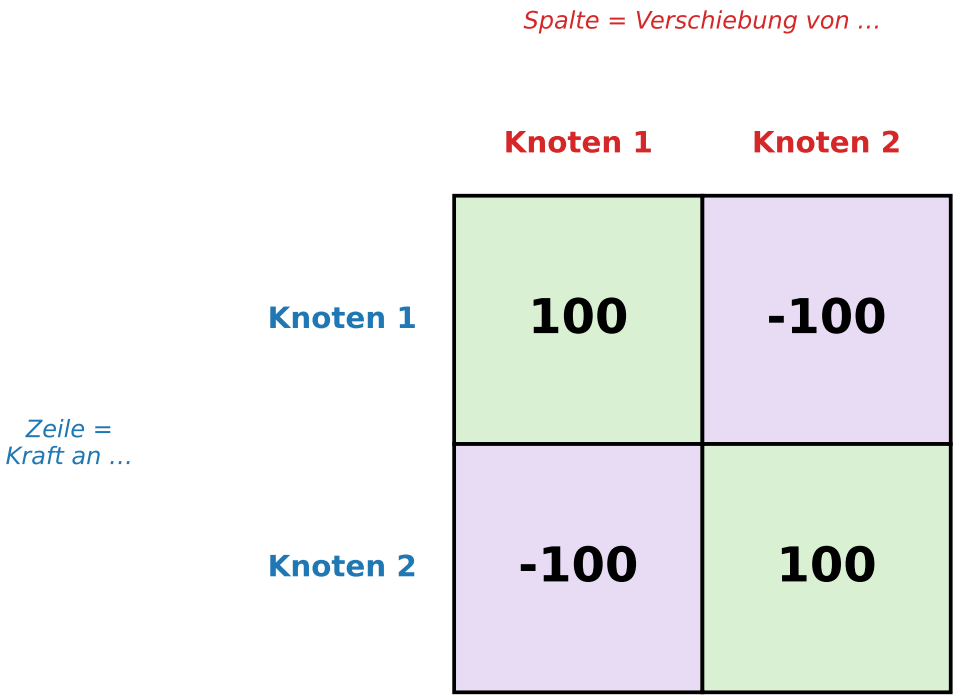

Abbildung 3.4.: Der Steckbrief einer Feder als 2×2-Tabelle. Zeilen und Spalten gehören zu den beiden Knoten der Feder. Die beiden Diagonalfelder (grün, +k) sagen: „Ziehe ich an einem Knoten, wehrt sich die Feder mit k.“ Die beiden Kreuzfelder (violett, −k) sagen: „Bewegt sich der eine Knoten, zieht er den anderen mit.“ Für unsere Federn ist k = 100 N/mm.

Was man hier sieht: vier Zahlen, die alles über eine Feder sagen. Auf der Diagonale steht zweimal $+k$ (grün): Das ist der Widerstand, den ein Knoten spürt, wenn *er selbst* gezogen wird. Auf den beiden Kreuzfeldern steht $-k$ (violett): So stark zieht ein Knoten seinen Partner mit. Dass die Zeilensummen null sind ($100 - 100 = 0$), ist kein Zufall — es heißt: Verschiebt man beide Knoten gleich weit, entsteht keine Kraft. Die Feder merkt nur Dehnung, nicht Wanderung.

3.4. Assemblierung: aus drei Steckbriefen ein System

Jetzt kommt der Kern des Kapitels und der Kern der ganzen FEM. Wir haben drei Federn, also drei gleiche Steckbriefe. Wir haben vier Knoten. Wie wird aus den drei kleinen 2×2-Tabellen **eine** große Tabelle für alle vier Knoten?

Die Antwort heißt **Assemblierung** (vom französischen *assembler*, zusammensetzen — ein Wort aus dem Maschinenbau, wo man Bauteile zu einer Baugruppe zusammenfügt). Die Idee ist verblüffend mechanisch: Wir legen eine leere 4×4-Tabelle an — vier Knoten, also vier Zeilen und vier Spalten — und **schieben jeden Steckbrief an die Stelle seiner beiden Knoten hinein**. Feder 1 gehört zu den Knoten 1 und 2, ihr Steckbrief kommt also in die Felder der Zeilen und Spalten 1 und 2. Feder 2 gehört zu den Knoten 2 und 3, Feder 3 zu den Knoten 3 und 4.

Das Entscheidende passiert dort, wo sich zwei Federn einen Knoten **teilen**. Knoten 2 gehört zu Feder 1 *und* zu Feder 2. Beide wollen etwas in das Diagonalfeld von Knoten 2 schreiben — jede ihr $+k$. Und was tun zwei Beiträge am selben Ort? Sie **addieren sich**. Aus k und k wird $2k$. Genau das ist die $2k$, die wir schon beim Kräftegleichgewicht gesehen haben: Sie entsteht, weil an einem inneren Knoten zwei Federn zerren, und ihre beiden Steckbriefe überlappen sich genau auf seinem Diagonalfeld.

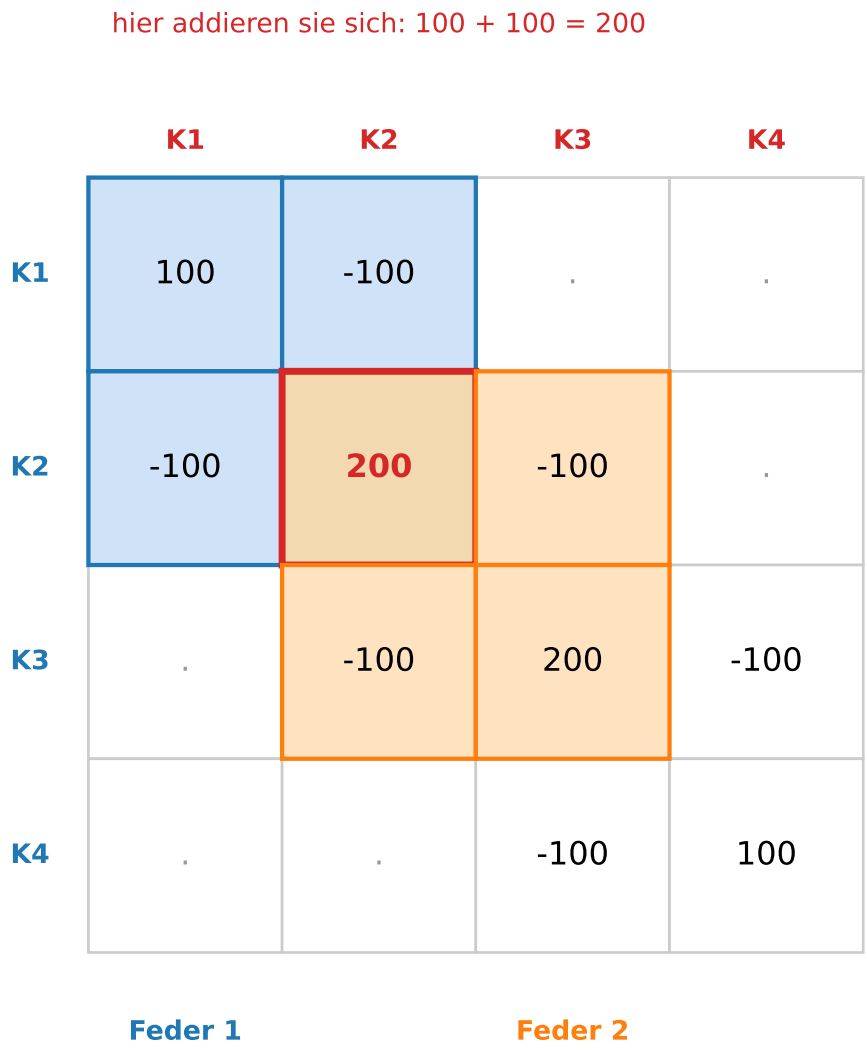

Abbildung 3.5.: Assemblierung als Schiebespiel: Der Steckbrief von Feder 1 (blau) rutscht in die Felder der Knoten 1–2, der von Feder 2 (orange) in die Felder der Knoten 2–3. Auf dem Diagonalfeld von Knoten 2 (rot umrandet) treffen beide aufeinander — dort addieren sich ihre Beiträge zu 100 + 100 = 200. Feder 3 (hier weggelassen) macht dasselbe bei Knoten 3–4.

Was man hier sieht: Die beiden 2×2-Steckbriefe überlappen sich in genau *einem* Feld — dem Diagonalfeld des gemeinsamen Knotens 2 (rot umrandet). Nur dort addieren sich Zahlen; überall sonst schreibt jede Feder in ihr

eigenes, leeres Feld. Die Ecken oben rechts und unten links bleiben leer (Punkte): Knoten 1 und Knoten 4 teilen sich keine Feder, also spüren sie einander nicht. Diese Leere ist kein Makel, sondern die **Bandstruktur** — sie macht das System später schnell lösbar.

3.4.1. Die Assemblierung im Ablauf

Weil das Ineinanderschieben der Bausteine der wichtigste Vorgang des Kapitels ist, lohnt es sich, ihm zuzusehen. Die folgende Animation baut die 4×4-Tabelle Feder für Feder auf: Jeder 2×2-Steckbrief fliegt an seinen Platz, und auf den geteilten Diagonalfeldern siehst du die Zahlen von 100 auf 200 hochspringen, wenn die zweite Feder ihren Beitrag dazulegt.

Leer

.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.

Feder 1 (K1–K2)

100	-100	.	.
-100	100	.	.
.	.	.	.
.	.	.	.

Feder 2 (K2–K3)

100	-100	.	.
-100	**200**	-100	.
.	-100	100	.
.	.	.	.

Feder 3 (K3–K4)

100	-100	.	.
-100	200	-100	.
.	-100	**200**	-100
.	.	-100	100

Abbildung 3.6.: Dieselbe Assemblierung als Bildreihe (in der HTML-Fassung eine Animation): von der leeren Tabelle über Feder 1 und Feder 2 bis zur vollständigen 4×4-Matrix. Beachte, wie die Diagonalfelder der Knoten 2 und 3 beim Einbau der zweiten anliegenden Feder von 100 auf 200 wachsen.

Was man sieht: Aus dem Nichts wächst Zeile für Zeile ein Gleichungssystem. Jede Feder liefert denselben Baustein; nur ihr *Platz* in der großen Tabelle ist ein anderer. Und weil benachbarte Federn sich einen Knoten teilen,

überlappen ihre Bausteine genau auf einem Diagonalfeld — dort, und nur dort, springt die Zahl auf 200. Nach drei Federn steht die vollständige Systemmatrix.

3.4.2. Das vollständige 4×4-System

Setzen wir alles zusammen. Vier Knoten ergeben vier Gleichungen, eine Kraft-Gleichung pro Knoten. Ordentlich als Matrix aufgeschrieben — Zeile für Knoten, Spalte für Verschiebung — lautet das System der Federkette:

$$\begin{pmatrix} 100 & -100 & 0 & 0 \\ -100 & 200 & -100 & 0 \\ 0 & -100 & 200 & -100 \\ 0 & 0 & -100 & 100 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} F_1 \\ 0 \\ 0 \\ 50 \end{pmatrix}$$

Gehen wir jede Zahl durch, denn hier ist nichts geheimnisvoll:

- **Zeile 1** gehört zu Knoten 1, an dem nur Feder 1 hängt: 100 auf der Diagonale, -100 zum Nachbarn Knoten 2, sonst nichts.
- **Zeile 2** gehört zu Knoten 2, dem inneren Knoten zwischen Feder 1 und Feder 2: die berühmte $200 = 100 + 100$ auf der Diagonale (die Überlappung!), links und rechts je -100 zu den Nachbarn.
- **Zeile 3** ist das Spiegelbild von Zeile 2, einen Knoten weiter — wieder 200 in der Mitte.
- **Zeile 4** gehört zu Knoten 4, an dem nur Feder 3 hängt: 100 auf der Diagonale, -100 zum Nachbarn Knoten 3.

Auf der rechten Seite steht, welche Kraft von außen an jedem Knoten zieht: An Knoten 4 sind es unsere 50 N, an den inneren Knoten 2 und 3 zieht niemand von außen (deshalb null), und an Knoten 1 steht F_1 — die noch unbekannte **Auflagerkraft**, die das Auflager aufbringt, um den Knoten festzuhalten.

> **Der Vorgriff aus Kapitel 2 — eingelöst**
>
> Erinnerst du dich an die 3×3-Tabelle aus Kapitel 2, die wir „mit Bedacht gewählt" nannten?
>
> $$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$
>
> Genau diese Gestalt steckt in unserem System. Teilt man die ganze 4×4-Matrix durch $k = 100$, bleibt das reine Muster übrig:

$$\begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{pmatrix}.$$

Die 2 auf der Diagonale ist ein Knoten mit zwei Federn, die -1 daneben ist die Kopplung zweier Nachbarn, die 0 (bzw. der Punkt) heißt „diese beiden spüren einander nicht" — Wort für Wort das Versprechen aus Kapitel 2. Die exakte 3×3-Tabelle mit lauter Zweien auf der Diagonale bekommst du, wenn *alle* gezeigten Knoten innere Knoten sind, also die Kette an beiden Enden festsitzt. Genau das baust du in Übung 3.3.

3.5. Die Randbedingung: das Auflager einbauen

Das System hat noch einen Haken: Auf der linken Seite steht die unbekannte Verschiebung u_1, auf der rechten die unbekannte Auflagerkraft F_1 — zwei Unbekannte in einer Gleichung, und das durch alle Zeilen. So ist das System nicht lösbar. Schlimmer noch: Ohne das Auflager könnte die ganze Kette als Ganzes davonschweben, ohne sich zu dehnen — es gäbe unendlich viele Lösungen.

Was rettet uns? Eine Information, die wir noch gar nicht benutzt haben: **Knoten 1 sitzt fest im Auflager, er verschiebt sich nicht.** Also ist $u_1 = 0$. Das ist eine **Randbedingung** — ein am Rand vorgegebener Wert. (Warum das genügt und was Randbedingungen sonst noch können, ist so wichtig, dass Kapitel 8 ein ganzes Kapitel daraus macht.)

Was tut $u_1 = 0$ mit dem System? In jeder Zeile wird der erste Summand $100 \cdot u_1$ oder $-100 \cdot u_1$ zu null — die ganze **erste Spalte** fällt weg. Und die **erste Zeile** brauchen wir nicht mehr: Sie würde nur die Auflagerkraft F_1 ausrechnen, die uns gerade nicht interessiert. Wir **streichen** also Zeile 1 und Spalte 1. Übrig bleibt ein kleineres, sauber lösbares System für die drei freien Knoten 2, 3 und 4:

$$\begin{pmatrix} 200 & -100 & 0 \\ -100 & 200 & -100 \\ 0 & -100 & 100 \end{pmatrix} \begin{pmatrix} u_2 \\ u_3 \\ u_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 50 \end{pmatrix}$$

Drei Gleichungen, drei Unbekannte, keine Auflagerkraft mehr. Dieses System können wir von Hand lösen.

3.6. Von Hand lösen

Vorhersage-Punkt

Bevor du weiterrechnest: Die drei Federn sind gleich steif, und weil sie hintereinanderhängen, zieht die Kraft von 50 N durch alle drei hindurch — wie ein Seil, an dem drei gleiche Gummibänder in Reihe hängen. Was glaubst du: Verschiebt sich Knoten 4 (am Zug) gleich weit wie Knoten 2 (nah am Auflager), oder weiter, oder weniger weit? Und um welche Beträge? Leg dich fest.

Wir lösen das reduzierte System durch **Einsetzen** — schrittweise eine Unbekannte durch die anderen ausdrücken. Schreiben wir die drei Zeilen als Gleichungen (die Einheiten sind N/mm mal mm = N, wir lassen sie beim Rechnen weg):

$$\begin{aligned} \text{(I)} \quad & 200\,u_2 - 100\,u_3 = 0, \\ \text{(II)} \quad & -100\,u_2 + 200\,u_3 - 100\,u_4 = 0, \\ \text{(III)} \quad & -100\,u_3 + 100\,u_4 = 50. \end{aligned}$$

Aus Gleichung (I) folgt sofort $200\,u_2 = 100\,u_3$, also $u_3 = 2\,u_2$. Der mittlere Knoten wandert doppelt so weit wie der erste freie — das klingt schon nach der Kette.

Aus Gleichung (III) folgt $100\,u_4 = 50 + 100\,u_3$, also $u_4 = u_3 + 0{,}5$. Der letzte Knoten liegt genau einen halben Millimeter vor dem mittleren.

Jetzt setzen wir beides in Gleichung (II) ein. Mit $u_3 = 2\,u_2$ und $u_4 = u_3 + 0{,}5 = 2\,u_2 + 0{,}5$ wird aus (II):

$$-100\,u_2 + 200\,(2\,u_2) - 100\,(2\,u_2 + 0{,}5) = 0.$$

Ausmultiplizieren: $-100\,u_2 + 400\,u_2 - 200\,u_2 - 50 = 0$, also $100\,u_2 - 50 = 0$ und damit

$$u_2 = 0{,}5 \text{ mm}.$$

Rückwärts eingesetzt: $u_3 = 2 \cdot 0{,}5 = 1{,}0$ mm und $u_4 = 1{,}0 + 0{,}5 = 1{,}5$ mm. Die ganze Lösung lautet also

$$u_1 = 0, \quad u_2 = 0{,}5 \text{ mm}, \quad u_3 = 1{,}0 \text{ mm}, \quad u_4 = 1{,}5 \text{ mm}.$$

3.6.1. Was die Zahlen bedeuten

Schau dir die Verschiebungen an: $0 — 0{,}5 — 1{,}0 — 1{,}5$. Von Knoten zu Knoten wächst der Weg um immer denselben Betrag, $0{,}5$ mm. Das ist kein Zufall, und es bestätigt deine Vorhersage (oder korrigiert sie): Jede der drei Federn ist um genau $0{,}5$ mm gedehnt, keine mehr als die andere.

Warum muss das so sein? Weil die Federn **hintereinander** hängen und die Kraft durch alle gleich hindurchgeht. Am Knoten 4 ziehen wir mit 50 N; dieselben 50 N muss Feder 3 tragen, sonst wäre Knoten 4 nicht im Gleichgewicht. Feder 3 gibt die 50 N an Knoten 3 weiter, der sie an Feder 2 reicht, und so fort — **jede Feder trägt dieselbe Kraft von 50 N**. Und eine Feder, die 50 N trägt, ist bei $k = 100$ N/mm um $50/100 = 0{,}5$ mm gedehnt. Drei gleiche Dehnungen hintereinander ergeben die Treppe $0{,}5 — 1{,}0 — 1{,}5$.

Merke dir dieses Wechselspiel: Die Matrix-Maschinerie (assemblieren, Randbedingung, lösen) und die anschauliche Physik (gleiche Kraft, gleiche Dehnung) liefern **dasselbe** Ergebnis. Die Maschinerie ist nicht klüger als die Physik — sie ist nur so stur und gleichförmig, dass ein Computer sie auch bei tausend Federn ausführen kann, wo die Anschauung längst kapituliert.

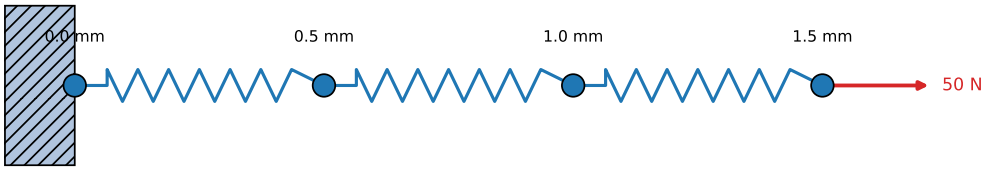

Abbildung 3.7.: Die gelöste Kette (in der HTML-Fassung eine Animation, die den Zug wachsen lässt): Endzustand mit den Verschiebungen 0 / 0,5 / 1,0 / 1,5 mm. Jede Feder ist um denselben halben Millimeter gedehnt; die Verschiebung ist optisch überhöht.

Was man sieht: die Kette in ihrer gedehnten Ruhelage. Die Verschiebungen bilden eine gleichmäßige Treppe — der sichtbare Beweis, dass jede Feder dieselbe Last trägt und dieselbe Strecke nachgibt.

3.7. Die interaktive Einheit: der Matrix-Drucker

Die Assemblierung von Hand ist bei drei Federn noch überschaubar. Aber die eigentliche Botschaft des Kapitels ist ja: **Der Rechenweg bleibt gleich,**

egal wie lang die Kette wird. Das probierst du jetzt selbst aus. Das folgende Programm baut die Systemmatrix für eine Kette aus beliebig vielen Federn und **druckt sie als Textblock** — die Nullen als Punkte, damit die Bandstruktur buchstäblich ins Auge springt.

Der Code ist genau die Papier-Assemblierung in Python: Erst eine leere Tabelle aus Nullen, dann für jede Feder ihre vier Beiträge ($+k$ auf die zwei Diagonalfelder, $-k$ auf die zwei Kreuzfelder) an die Stelle ihrer beiden Knoten addieren — dieselbe Doppelschleife, die du in Kapitel 2 gelernt hast.

Vorhersage-Punkt

Bevor du ausführst: Stell die Zahl der Federn auf 5. Dann hat die Kette 6 Knoten, die Matrix ist also 6×6 — das sind 36 Felder. Wie viele davon sind **nicht** null? (Tipp: Denk an die Bandstruktur — jeder Knoten hängt nur mit sich selbst und seinen unmittelbaren Nachbarn zusammen.) Leg dich auf eine Zahl fest.

```
#| autorun: true
# Der Matrix-Drucker. Aendere STEIFIGKEITEN: eine Zahl je Feder.
# Aus n Federn werden n+1 Knoten und eine (n+1)x(n+1)-Systemmatrix.
STEIFIGKEITEN = [100.0, 100.0, 100.0, 100.0, 100.0]   # fuenf gleiche Federn


def baue_systemmatrix(steifigkeiten):
    """Assembliert die Steifigkeitsmatrix einer Federkette.

    steifigkeiten: Liste der Federsteifigkeiten (eine Zahl je Feder).
    Rueckgabe: die Systemmatrix als verschachtelte Liste.
    """
    federzahl = len(steifigkeiten)
    knotenzahl = federzahl + 1

    # Leere Matrix aus lauter Nullen.
    matrix = []
    for zeile in range(knotenzahl):
        neue_zeile = []
        for spalte in range(knotenzahl):
            neue_zeile.append(0.0)
        matrix.append(neue_zeile)

    # Jede Feder ihren 2x2-Baustein an ihren beiden Knoten aufaddieren.
    for feder in range(federzahl):
        steifigkeit = steifigkeiten[feder]
```

```
        links = feder
        rechts = feder + 1
        matrix[links][links] = matrix[links][links] + steifigkeit
        matrix[links][rechts] = matrix[links][rechts] - steifigkeit
        matrix[rechts][links] = matrix[rechts][links] - steifigkeit
        matrix[rechts][rechts] = matrix[rechts][rechts] + steifigkeit
    return matrix


def drucke_matrix(matrix):
    """Druckt die Matrix als Textblock; Nullen werden zu Punkten."""
    nicht_null = 0
    for zeile in matrix:
        for wert in zeile:
            if wert == 0.0:
                print(f"{'.':>8}", end="")
            else:
                nicht_null = nicht_null + 1
                print(f"{wert:8.0f}", end="")
        print()
    print()
    print(f"{len(matrix)}x{len(matrix)}-Matrix, davon {nicht_null} Zahlen "
          f"nicht null.")


matrix = baue_systemmatrix(STEIFIGKEITEN)
drucke_matrix(matrix)

# --- Abbildung ---
import matplotlib.pyplot as plt

groesse = len(matrix)
fig, ax = plt.subplots(figsize=(4.6, 4.6))
for r in range(groesse):
    for c in range(groesse):
        gefuellt = matrix[r][c] != 0.0
        ax.add_patch(plt.Rectangle((c, groesse - 1 - r), 1, 1,
                                   facecolor="tab:blue" if gefuellt else "white",
                                   edgecolor="0.8", lw=0.8))
ax.set_xlim(0, groesse)
ax.set_ylim(0, groesse)
ax.set_aspect("equal")
ax.set_title("Belegte Felder (blau) – die Bandstruktur")
```

```
ax.axis("off")
plt.tight_layout()
plt.show()
```

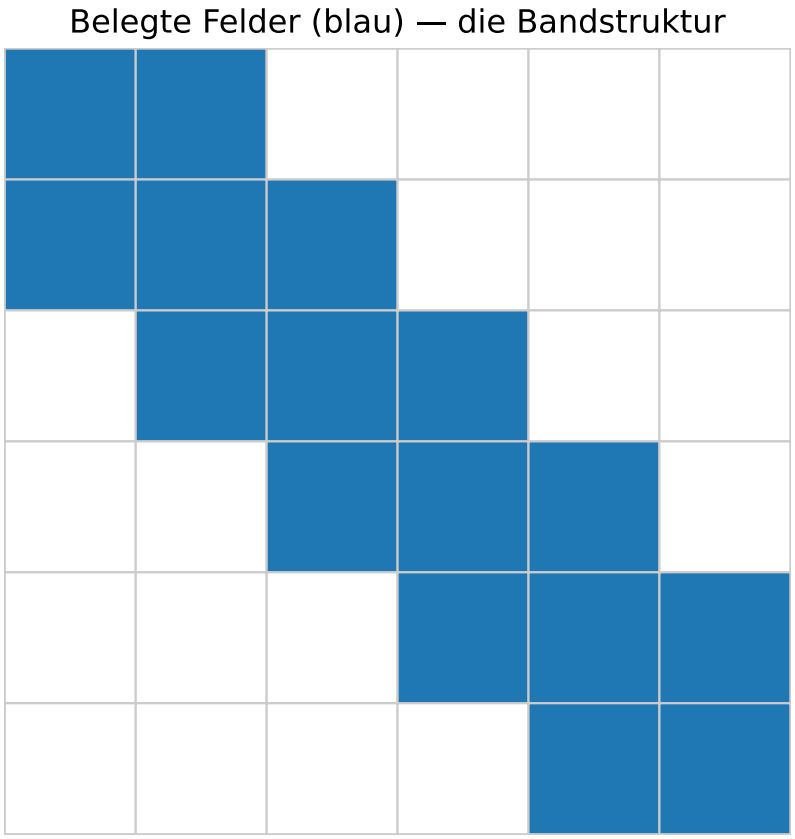

Abbildung 3.8.: Vorgerenderte Fassung des Matrix-Druckers für fünf Federn (6×6-Matrix). Blau sind die belegten Felder: die Diagonale und je ein Feld links und rechts davon — 16 von 36. Alles andere ist leer. In der HTML-Fassung ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst.

Was man sieht: Bei fünf Federn sind von den 36 Feldern nur **16** nicht null — die Diagonale (6 Felder) und je ein Feld links und rechts davon (zweimal 5). Lag deine Vorhersage richtig? Alles andere ist leer. Und je länger die Kette, desto krasser das Verhältnis: Bei 100 Federn hat die 101×101-Matrix über 10.000 Felder, aber nur rund 300 sind belegt. Diese **dünne Besetzung** ist ein Geschenk — 2 zeigt, wie ein Löser sie ausnutzt und deshalb auch riesige Ketten in Sekunden schafft. Probier ruhig aus, einzelne Steifigkeiten zu

ändern oder Federn dazuzuhängen: Die Bandstruktur bleibt, nur die Zahlen darin ändern sich.

3.8. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript zu diesem Kapitel liegt in `programme/kap03/kap03_federkette.py`. Es führt die drei Schritte dieses Kapitels an einem Stück vor: Es **assembliert** die 4×4-Systemmatrix (mit demselben `baue_systemmatrix` wie oben, das als wiederverwendbarer Baustein in `programme/gemeinsam/matrix_drucker.py` liegt — 2 wird ihn importieren), baut die **Randbedingung** ein, **löst** die Kette und macht zum Schluss die Probe: Es rechnet aus den Verschiebungen die Knotenkräfte zurück und prüft, ob am festen Knoten die Auflagerkraft und am Zugende die 50 N herauskommen. Führe es mit `python kap03_federkette.py` aus und vergleiche seine Ausgabe mit unserer Handrechnung.

Zum Lösen nutzt das Skript bewusst **nicht** einen allgemeinen Gleichungslöser — der ist das Thema von 2. Es verwendet stattdessen die anschauliche Physik der Reihenschaltung: Jede Feder trägt dieselbe Kraft, ihre Dehnung ist Kraft geteilt durch Steifigkeit, und die Verschiebung eines Knotens ist die Summe der Dehnungen bis zu ihm. So bleibt das Kapitel bei reiner Schulalgebra, und der maschinelle Löser bekommt im nächsten Kapitel seinen großen Auftritt.

3.9. Woher die Methode kommt

Jetzt, wo du die Maschinerie selbst gebaut hast, lohnt der Blick zurück auf ihre Erfinder — denn fast jeder Begriff dieses Kapitels stammt aus einer sehr konkreten Notlage der frühen 1950er Jahre.

> **i** Geschichte: Vom Flugzeugflügel zur Weltmethode
>
> Nach dem Zweiten Weltkrieg wurden Flugzeuge schneller, und ihre Flügel wurden dünner und pfeilförmiger — besonders die **Delta-Flügel** der neuen Düsenjets. Solche Flügel ließen sich mit den klassischen Formeln der Statik nicht mehr zuverlässig berechnen: Ihre Form war zu unregelmäßig, und bei hoher Geschwindigkeit drohte **Flattern**, ein gefährliches Aufschaukeln.
> Bei **Boeing** in Seattle leitete der Ingenieur **M. J. (Jon) Turner** die Gruppe für Strukturdynamik. In den Sommern **1952 und 1953** holte er sich

zwei Gäste dazu: **Ray W. Clough** von der Universität Berkeley und **Harold Martin** von der Universität Washington. Ihre Idee, um den komplizierten Flügel berechenbar zu machen, ist genau die dieses Kapitels: **Zerlege den Flügel in viele kleine, einfache Stücke — Dreiecke und Rechtecke —, für die man rechnen kann, schreibe für jedes seinen Steckbrief (seine Steifigkeitsmatrix) auf und setze alles wieder zusammen.** 1956 veröffentlichten Turner, Clough, Martin und L. J. Topp das Verfahren (Turner u. a. 1956); man nannte es damals *direct stiffness method*, Methode der direkten Steifigkeit. Daher kommen unsere Wörter **Steifigkeit**, **Element** und **Assemblierung** — es war die Sprache des Maschinen- und Flugzeugbaus.
Etwa zur selben Zeit entwickelte in Europa **John Argyris** — an der Universität Stuttgart und in London — die energetische Theorie dahinter (Argyris 1954--1955). Den einprägsamen Namen aber prägte **Ray Clough**: **1960** stellte er auf einer Tagung in Pittsburgh seine Arbeit unter dem Titel „The Finite Element Method in Plane Stress Analysis" vor (Clough 1960) — und aus der *direct stiffness method* wurde die **Finite-Elemente-Methode**. Von da an ging es schnell: Ende der 1960er gab die **NASA** das große, frei verfügbare Programm **NASTRAN** in Auftrag (1968 fertiggestellt), und was als Rettung für den Flugzeugflügel begann, rechnet heute Autos, Brücken, Handys — und Stahlträger im Feuer.

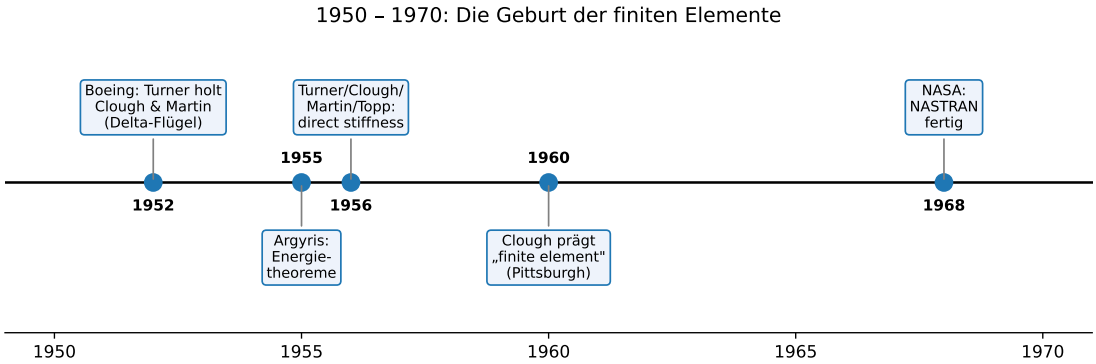

Abbildung 3.9.: Zwei Jahrzehnte von der Notlage zur Weltmethode: von Turners Boeing-Team am Delta-Flügel (1952) über das Gründungspapier (1956) und Cloughs Namensprägung „finite element" (1960) bis zum NASA-Programm NASTRAN (1968), das die Methode allen zugänglich machte.

Was man hier sieht: In keinen zwei Jahrzehnten wuchs aus einer Flügel-Notlage eine Methode, die heute überall steckt. Bemerkenswert ist das Tem-

po zwischen Idee (1952), erstem Papier (1956) und dem Namen, der blieb (1960) — und dass die Begriffe, mit denen wir eben gerechnet haben, genau von hier stammen.

Merkkasten

- Eine **Feder** verknüpft zwei Knoten; ihr Steckbrief ist die **2×2-Elementmatrix** $\begin{pmatrix} k & -k \\ -k & k \end{pmatrix}$ — nur ein ordentlich aufgeschriebenes Gleichungssystem.
- **Assemblieren** heißt: jede Elementmatrix an die Stelle ihrer Knoten in die große Systemmatrix addieren. Wo sich Federn einen Knoten teilen, **addieren** sich ihre Beiträge auf der Diagonale ($k + k = 2k$).
- Ein fester Knoten ist eine **Randbedingung** ($u = 0$): Man **streicht** seine Zeile und Spalte und erhält ein kleineres, lösbares System.
- In einer Reihenschaltung trägt **jede Feder dieselbe Kraft**; gleiche Federn dehnen sich darum gleich weit — die Verschiebungen wachsen als gleichmäßige Treppe.
- Die Systemmatrix ist **dünn besetzt** (Bandstruktur): fast nur Nullen, die belegten Zahlen drängen sich um die Diagonale.
- Die Begriffe **Steifigkeit, Element, Assemblierung** stammen aus dem Flugzeugbau der 1950er; **Clough** prägte 1960 den Namen „finite element".

Roter Faden

Zurück: Die verschachtelte Liste aus Kapitel 2 ist jetzt eine **Matrix** geworden — genau die 3×3-Tabelle, die wir dort „mit Bedacht gewählt" nannten, ist die Gestalt der Federketten-Systemmatrix. Die Doppelschleife, die dort ein Zahlengitter druckte, druckt hier die Systemmatrix; die laufende Summe steckt in der Assemblierung („addiere den Beitrag an die richtige Stelle").

Vor: Das 4×4-System haben wir hier von Hand gelöst; 2 löst es **maschinell** und nutzt dabei die Bandstruktur aus. Und dann der große Bogen: Genau dieselbe Maschinerie — assemblieren, Randbedingung, lösen — rechnet ab Kapitel 7 die **Wärme** im Träger. Denn:

> **Ersetze „Verschiebung" durch „Temperatur" und „Kraft" durch „Wärmestrom" — dieselbe Maschinerie rechnet Wärme.**

Das ist der Plan der nächsten Kapitel. Die Federkette war nie nur eine Feder; sie war die FEM im Kleinen.

Übungen

Ü 3.1 (Verstehen). Eine Kette aus **zwei** gleichen Federn ($k = 100$ N/mm) hat drei Knoten. Schreibe ihre 3×3-Systemmatrix von Hand auf, indem du die zwei Steckbriefe assemblierst. Welche Zahl steht auf dem mittleren Diagonalfeld (Knoten 2), und warum? Kontrolliere dein Ergebnis, indem du im Matrix-Drucker `STEIFIGKEITEN = [100.0, 100.0]` setzt.

> **i** Musterlösung zu Ü 3.1
>
> Feder 1 belegt die Knoten 1–2, Feder 2 die Knoten 2–3. Assembliert:
>
> $$\begin{pmatrix} 100 & -100 & 0 \\ -100 & 200 & -100 \\ 0 & -100 & 100 \end{pmatrix}$$
>
> Auf dem mittleren Diagonalfeld steht **200**, weil Knoten 2 der einzige innere Knoten ist: An ihm hängen beide Federn, und ihre Beiträge ($100 + 100$) addieren sich genau dort. Die beiden äußeren Knoten tragen je nur eine Feder, darum steht bei ihnen 100. Die Ecken oben rechts und unten links sind null — Knoten 1 und 3 teilen sich keine Feder.

Ü 3.2 (Verändern). Nimm wieder die drei Federn aus dem Kapitel, aber mach die **mittlere** zehnmal steifer: $k_2 = 1000$ N/mm statt 100. Am Ende ziehen weiter 50 N. **Sage zuerst vorher:** Welche der drei Federn dehnt sich jetzt am wenigsten, und wie weit kommt das freie Ende insgesamt — weiter oder weniger weit als die 1,5 mm von vorhin? Rechne dann nach (das Skript `loesungen/kap03_ue2.py` macht es vor).

> **i** Musterlösung zu Ü 3.2
>
> Auch jetzt trägt jede Feder dieselbe Kraft von 50 N (Reihenschaltung, das ändert die Steifigkeit nicht). Aber die steife Mittelfeder gibt unter 50 N nur um $50/1000 = 0{,}05$ mm nach — ein Zehntel der weichen Federn. Die beiden weichen Federn dehnen sich weiter je 0,5 mm. Die Verschiebungen sind also $0 - 0{,}5 - 0{,}55 - 1{,}05$ mm: Das freie Ende kommt nur noch **1,05 mm** weit statt 1,5 mm. Merksatz: **Eine steife Feder verschiebt sich wenig** — sie „schluckt" kaum etwas von der Bewegung. Das ist dasselbe Prinzip, das später einen gut wärmeleiten-

den Trägerabschnitt kaum Temperaturgefälle aufbauen lässt.

Ü 3.3 (Übertragen). Was ändert sich, wenn die Kette an **beiden** Enden festsitzt? Nimm eine Kette aus **vier** gleichen Federn (der Einfachheit halber $k = 1$), fünf Knoten, und halte Knoten 1 **und** Knoten 5 fest. Schreibe die volle 5×5-Systemmatrix auf und streiche die Zeilen und Spalten **beider** fester Knoten. Was bleibt übrig — und woran erinnert es dich? (Das Skript `loesungen/kap03_ue3.py` baut es Schritt für Schritt.)

i Musterlösung zu Ü 3.3

Die volle Matrix (mit $k = 1$) ist

$$\begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & 1 \end{pmatrix}.$$

Streicht man Zeile und Spalte 1 (Knoten 1 fest) **und** Zeile und Spalte 5 (Knoten 5 fest), bleibt für die drei inneren Knoten 2, 3, 4 genau

$$\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

übrig — die 3×3-Tabelle aus Kapitel 2, jetzt vollständig eingelöst. Der Unterschied zum Kapitel-Beispiel: Weil jetzt **beide** Enden fest sind, trägt jeder verbleibende Knoten zwei Federn, und alle Diagonaleinträge sind 2 (nicht mehr die 1 am freien Ende). Physikalisch tauchen jetzt **zwei** Haltekräfte auf, eine an jedem Ende — dass Randbedingungen so viel mitbestimmen, ist der Grund, warum Kapitel 8 ihnen ein ganzes Kapitel widmet.

Das Kleingedruckte

Ein paar ehrliche Feinheiten, abseits des Hauptpfads. **Erstens:** Wir haben so getan, als wäre eine Feder eine perfekt lineare Sache — doppelte Kraft, doppelte Dehnung, immer. Echte Federn und echte Werkstoffe halten sich nur in einem gewissen Bereich daran; überdehnt man sie, wird es krumm. Das ganze Buch bewegt sich bewusst im linearen Bereich (Strategie-Vereinfachung Nr. 3). **Zweitens:** Wir haben die erste Zeile des Systems einfach gestrichen

und die Auflagerkraft F_1 nicht ausgerechnet. Man *kann* sie hinterher bestimmen (unser Kapitel-Programm tut es in der Probe: es kommen -50 N heraus, das Auflager zieht mit 50 N dagegen) — nur für die Verschiebungen braucht man sie nicht. **Drittens:** „Matrix" haben wir hier strikt als „ordentlich aufgeschriebenes Gleichungssystem" behandelt, ohne Matrizenrechnung. Das reicht für das ganze Buch; die üblichen Rechenregeln für Matrizen (das „Malnehmen" zweier Matrizen) brauchen wir nie. Wer sie kennt, darf sie mitdenken — wer nicht, verpasst nichts.

4. Gleichungssysteme lösen

Am Ende von Kapitel 3 stand ein Gleichungssystem an der Tafel — das reduzierte 3×3-System der Federkette. Wir haben es von Hand gelöst, durch geschicktes Einsetzen, und drei Zahlen geerntet: $u_2 = 0{,}5$, $u_3 = 1{,}0$, $u_4 = 1{,}5$ mm. Das war überschaubar, weil es nur drei Unbekannte waren.

Aber die Botschaft der Federkette war ja gerade: **Der Weg bleibt gleich, egal wie lang die Kette wird.** Das Netz, mit dem wir am Ende dieses Buches den Träger im Brandfall durchrechnen, hat rund 400 Unbekannte. Ein Handy-Crashtest bei einem Hersteller hat leicht zehn Millionen. Niemand setzt zehn Millionen Gleichungen von Hand ineinander ein. Die Frage dieses Kapitels lautet darum:

Wie bringt man einer Maschine bei, ein Gleichungssystem zu lösen — und warum dauert das bei zehn Millionen Unbekannten nicht Jahre?

Wir bauen dazu drei Löser, die den Rest des Buches antreiben werden. Alle drei bekommen zur Prüfung dasselbe reduzierte Federketten-System vorgelegt, und alle drei müssen am Ende dieselben drei Zahlen liefern wie unsere Handrechnung: $0{,}5$, $1{,}0$, $1{,}5$ mm. Die Handrechnung aus Kapitel 3 ist von jetzt an unsere **Probe**.

Lernziele

Nach diesem Kapitel kannst du …

1. … ein 3×3-System mit **Gauß-Elimination** von Hand lösen — leerräumen, dann rückwärts einsetzen,
2. … erklären, warum die Federketten-Matrix fast nur Nullen enthält (**dünn besetzt**, **Bandmatrix**) und warum das Rechenzeit spart,
3. … den **Thomas-Algorithmus** als „Gauß, der nur das Band anfasst" anwenden,
4. … **Gauß-Seidel** als Entspannungsverfahren erklären und beobachten, wie sich die Lösung Runde um Runde einpendelt,
5. … abschätzen, warum volle Elimination bei großen Systemen unbezahlbar wird — mit konkreten Zahlen, ohne Fachbegriffe.

⚠ Naheliegende Vermutung

Vermutung: *„Doppelt so viele Unbekannte — doppelt so lange Rechenzeit. Zehnmal so viele, zehnmal so lange."*
Warum sie naheliegt: Das ist die Erfahrung aus dem Alltag: doppelt so viele Kartoffeln schälen dauert doppelt so lange. Rechenarbeit müsste sich doch genauso verhalten.
Was stattdessen stimmt: Bei der vollen Gauß-Elimination **verachtfacht** sich die Arbeit, wenn sich die Zahl der Unbekannten nur verdoppelt — nicht mal zwei, sondern mal zwei hoch drei. Von 10 auf 20 Unbekannte ist es kaum zu merken; von einer Million auf zwei Millionen wird aus einem Tag mehr als eine Woche. Genau deshalb reicht es nicht, *irgend* einen Löser zu haben. Der Ausweg ist, die vielen Nullen der FEM-Matrix auszunutzen: Beim Thomas-Algorithmus ist doppelt so groß dann wirklich nur doppelt so teuer. Diesen Unterschied — Fluch und Rettung — rechnen wir in diesem Kapitel mit echten Zahlen aus.

4.1. Was „lösen" überhaupt heißt

Bevor wir eine Maschine losschicken, halten wir fest, was sie eigentlich suchen soll. Ein Gleichungssystem zu lösen heißt: **Zahlen finden, die alle Gleichungen gleichzeitig erfüllen.** Nicht die eine oder die andere — alle auf einmal.

Am kleinsten Fall sieht man es am besten. Nimm zwei Gleichungen mit zwei Unbekannten x und y:

$$\begin{aligned} x + y &= 3, \\ x - y &= 1. \end{aligned}$$

Jede dieser Gleichungen beschreibt eine **Gerade** in einem x-y-Diagramm — eine ganze Straße von Punkten, die sie erfüllen. Die erste erfüllen alle Punkte mit $x + y = 3$, die zweite alle mit $x - y = 1$. Gesucht ist der *eine* Punkt, der auf **beiden** Geraden liegt: ihr Schnittpunkt.

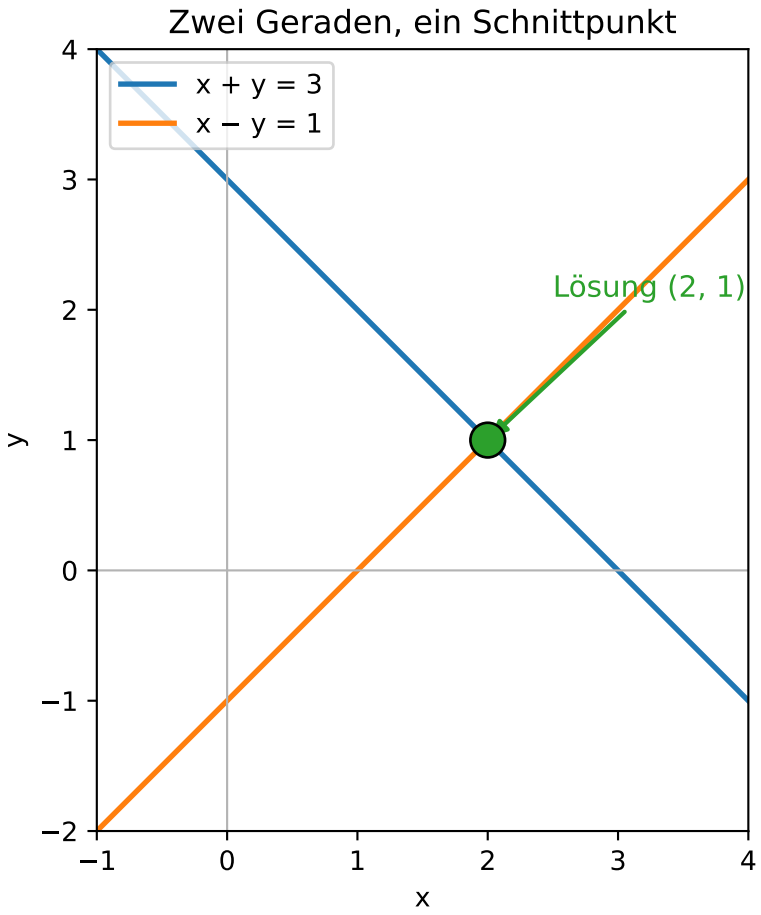

Abbildung 4.1.: Lösen heißt schneiden: Jede der beiden Gleichungen ist eine Gerade. Die erste (blau) sammelt alle Punkte mit x + y = 3, die zweite (orange) alle mit x − y = 1. Nur ein einziger Punkt liegt auf beiden — der Schnittpunkt bei x = 2, y = 1. Das ist die Lösung des Systems.

Was man hier sieht: Zwei Geraden, ein Schnittpunkt. Am Punkt $(2, 1)$ gilt $2 + 1 = 3$ *und* $2 - 1 = 1$ — beide Gleichungen stimmen zugleich. Setz die Zahlen ein und probiere es nach. Genau das ist eine Lösung: ein Satz von Werten, an dem alle Gleichungen aufgehen.

Bei drei Unbekannten — wie im reduzierten Federketten-System — wären es drei „Flächen” im Raum, die sich in einem Punkt treffen; bei vieren kann man es nicht mehr zeichnen. Deshalb **rechnen** wir statt zu zeichnen, und zwar so stur und schematisch, dass es auch eine Maschine tun kann. Das erste Verfahren dafür ist über zweihundert Jahre alt und trägt den Namen des Mannes, der es in eine feste Form goss: Carl Friedrich Gauß.

4.2. Gauß-Elimination: erst leerräumen, dann ernten

Die Idee der Gauß-Elimination ist bestechend einfach. Ein System mit drei Unbekannten ist schwer, weil in jeder Zeile alle drei zugleich vorkommen. Wäre in der letzten Zeile nur noch **eine** Unbekannte, könnten wir sie sofort ausrechnen. Stünden in der vorletzten nur noch **zwei**, davon eine schon bekannt, ginge auch die. Und so weiter nach oben.

Das Verfahren stellt genau das her. Es hat zwei Phasen:

1. **Leerräumen** (Vorwärtselimination): Wir machen mit geschickten Zeilen-Kombinationen alle Zahlen **unter** der Diagonale zu null. Übrig bleibt eine Treppe — eine *Dreiecksform*, in der die letzte Zeile nur noch eine Unbekannte enthält.
2. **Ernten** (Rückwärtseinsetzen): Von unten nach oben rechnen wir eine Unbekannte nach der anderen aus, jede aus den schon bekannten darüber.

Führen wir es an unserem reduzierten Federketten-System vor. Es lautet, Zeile für Zeile als Gleichung geschrieben (Einheiten N und mm lassen wir beim Rechnen weg):

$$\begin{aligned}
&(\mathrm{I}) && 200\,u_2 && -100\,u_3 && && = && 0,\\
&(\mathrm{II}) && -100\,u_2 && +200\,u_3 && -100\,u_4 && = && 0,\\
&(\mathrm{III}) && && -100\,u_3 && +100\,u_4 && = && 50.
\end{aligned}$$

Vorhersage-Punkt

Bevor du weiterrechnest: Wir kennen die Antwort schon aus Kapitel 3 (0,5, 1,0, 1,5). Diesmal geht es um den *Weg*. Der erste Schritt wird die -100 am Anfang von Zeile (II) beseitigen, indem er ein Vielfaches von Zeile (I) dazuzählt. Welches Vielfache? Überlege: Was muss man mit den 200 aus Zeile (I) tun, damit gerade $+100$ herauskommt, das die -100 ausgleicht? Leg dich fest.

4.2.1. Phase 1, Schritt 1: die erste Spalte leerräumen

Wir wollen in Zeile (II) das $-100\,u_2$ loswerden. In Zeile (I) steht an derselben Stelle $200\,u_2$. Addieren wir die **Hälfte** von Zeile (I) zu Zeile (II), so wird aus $-100 + \frac{1}{2}\cdot 200 = -100 + 100 = 0$ — genau null. Die halbe Zeile (I) ist unser Werkzeug (das Vielfache aus dem Vorhersage-Punkt: der Faktor $\frac{1}{2}$).

Rechnen wir Zeile (II) $+\,\frac{1}{2}\cdot$ Zeile (I) Glied für Glied aus:

$$\begin{aligned}
u_2\text{-Glied:}&\quad -100+\tfrac{1}{2}\cdot 200 = 0,\\
u_3\text{-Glied:}&\quad 200+\tfrac{1}{2}\cdot(-100) = 150,\\
u_4\text{-Glied:}&\quad -100+\tfrac{1}{2}\cdot 0 = -100,\\
\text{rechte Seite:}&\quad 0+\tfrac{1}{2}\cdot 0 = 0.
\end{aligned}$$

Zeile (III) hat an der ersten Stelle schon eine Null — hier ist nichts zu tun. Nach dem ersten Schritt steht also:

$$\begin{array}{rrrrcr}
(\mathrm{I}) & 200\,u_2 & -100\,u_3 & & = & 0,\\
(\mathrm{II}') & & 150\,u_3 & -100\,u_4 & = & 0,\\
(\mathrm{III}) & & -100\,u_3 & +100\,u_4 & = & 50.
\end{array}$$

Die erste Spalte ist unter der Diagonale leer. u_2 kommt nur noch in der ersten Zeile vor.

4.2.2. Phase 1, Schritt 2: die zweite Spalte leerräumen

Jetzt dasselbe eine Etage tiefer: In Zeile (III) soll das $-100\,u_3$ verschwinden. Darüber, in Zeile (II'), steht jetzt $150\,u_3$. Diesmal brauchen wir nicht die Hälfte, sondern $\frac{100}{150} = \frac{2}{3}$ von Zeile (II'), damit $-100+\frac{2}{3}\cdot 150 = -100+100 = 0$ wird.

Zeile (III) $+\,\frac{2}{3}\cdot$ Zeile (II'):

$$\begin{aligned}
u_3\text{-Glied:}&\quad -100+\tfrac{2}{3}\cdot 150 = 0,\\
u_4\text{-Glied:}&\quad 100+\tfrac{2}{3}\cdot(-100) = 100-66{,}67 = 33{,}33,\\
\text{rechte Seite:}&\quad 50+\tfrac{2}{3}\cdot 0 = 50.
\end{aligned}$$

Das Ergebnis ist die ersehnte Dreiecksform:

$$\begin{array}{rrrrcr}
(\mathrm{I}) & 200\,u_2 & -100\,u_3 & & = & 0,\\
(\mathrm{II}') & & 150\,u_3 & -100\,u_4 & = & 0,\\
(\mathrm{III}') & & & \frac{100}{3}\,u_4 & = & 50.
\end{array}$$

Unter der Diagonale steht nur noch null. Die letzte Zeile enthält nur eine einzige Unbekannte. Von hier an ist es leicht.

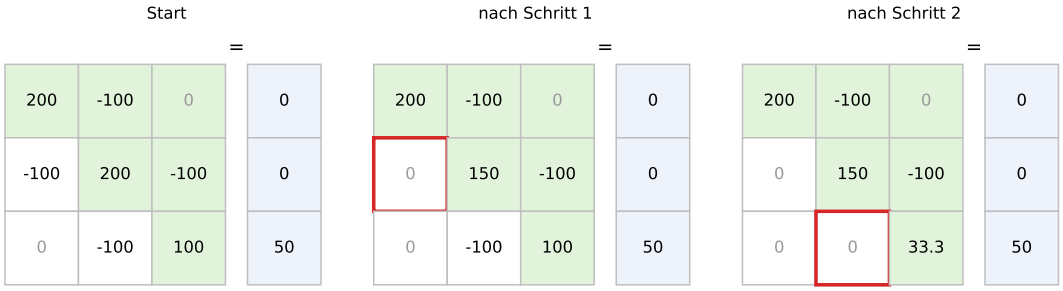

Abbildung 4.2.: Die Vorwärtselimination als Schrittbild. Links das volle System, in der Mitte nach dem ersten Schritt (die erste Spalte unter der Diagonale ist geräumt), rechts nach dem zweiten (auch die zweite Spalte). Die grün schraffierte Treppe ist die Dreiecksform: unter ihr steht nur noch null. Die rot umrandeten Felder sind die, die im jeweiligen Schritt gerade zu null gemacht wurden.

Was man hier sieht: Wie das System von links nach rechts „leergeräumt" wird. In jedem Schritt fällt ein Feld unter der Diagonale auf null (rot umrandet), bis nur noch die grüne Dreieckstreppe Zahlen trägt. Diese Form ist das Ziel der ersten Phase — an ihr kann man von unten nach oben ernten.

4.2.3. Phase 2: die Ernte (Rückwärtseinsetzen)

Jetzt kommt der schöne Teil. Wir lesen die Dreiecksform von **unten nach oben**.

Aus Zeile (III') — der einzelnen Unbekannten — folgt sofort

$$\tfrac{100}{3}\,u_4 = 50 \quad\Longrightarrow\quad u_4 = \frac{50}{100/3} = \frac{50\cdot 3}{100} = 1{,}5.$$

In Zeile (II') ist jetzt nur noch u_3 unbekannt, denn u_4 kennen wir:

$$150\,u_3 - 100\,u_4 = 0 \quad\Longrightarrow\quad u_3 = \frac{100\cdot 1{,}5}{150} = \frac{150}{150} = 1{,}0.$$

In Zeile (I) schließlich ist nur noch u_2 offen:

$$200\,u_2 - 100\,u_3 = 0 \quad\Longrightarrow\quad u_2 = \frac{100\cdot 1{,}0}{200} = 0{,}5.$$

Geerntet: $u_2 = 0{,}5$, $u_3 = 1{,}0$, $u_4 = 1{,}5$ mm — Zahl für Zahl dieselbe Lösung wie die Handrechnung aus Kapitel 3. Die Probe stimmt. Wir haben sie diesmal nicht durch pfiffiges Einsetzen gefunden, sondern durch ein stures Schema, das man Zeile für Zeile abarbeitet, ohne nachzudenken — und genau darum kann es eine Maschine.

4.2.4. Dasselbe in Python

Die Prosa hat erklärt, was passiert; jetzt macht der Code die Schritte anfassbar. Das folgende Programm ist die Vorwärtselimination in reinem Python — dieselbe Doppelschleife, die wir eben auf Papier ausgeführt haben, mit denselben Namen: für jede Spalte den Pivot nehmen, darunter jede Zeile um ein Vielfaches der Pivotzeile verringern. Nach jedem Schritt druckt es die Matrix, damit du das Leerräumen mitverfolgen kannst (die Nullen als Punkte, wie beim Matrix-Drucker aus Kapitel 3).

```
def zeige_elimination(matrix, rechte_seite):
    """Fuehrt die Vorwaertselimination aus und druckt jeden Schritt.

    matrix: die Systemmatrix als verschachtelte Liste.
    rechte_seite: Liste der rechten Seiten.
    Rueckgabe: nichts; die Funktion druckt nur mit.

    Reine Vorfuehrung: Spalte fuer Spalte wird unter der Diagonale
    leergeraeumt, nach jeder Spalte wird die Matrix gezeigt.
    """
    n = len(matrix)
    # Auf Kopien arbeiten, damit das Original erhalten bleibt.
    werk = []
    for zeile in matrix:
        werk.append(list(zeile))
    seite = list(rechte_seite)

    for spalte in range(n):
        pivot = werk[spalte][spalte]
        for zeile in range(spalte + 1, n):
            faktor = werk[zeile][spalte] / pivot
            for k in range(spalte, n):
                werk[zeile][k] = werk[zeile][k] - faktor * werk[spalte][k]
            seite[zeile] = seite[zeile] - faktor * seite[spalte]
        print(f"Nach dem Leerraeumen von Spalte {spalte + 1}:")
        drucke(werk, seite)
```

```
def drucke(matrix, seite):
    """Druckt Matrix und rechte Seite; Nullen werden zu Punkten."""
    for zeile in range(len(matrix)):
        for wert in matrix[zeile]:
            if abs(wert) < 0.0001:
                print(f"{'.':>8}", end="")
            else:
                print(f"{wert:8.2f}", end="")
        print(f"   | {seite[zeile]:7.2f}")
    print()


system = [[200.0, -100.0, 0.0],
          [-100.0, 200.0, -100.0],
          [0.0, -100.0, 100.0]]
kraefte = [0.0, 0.0, 50.0]
zeige_elimination(system, kraefte)
```

```
Nach dem Leerraeumen von Spalte 1:
  200.00 -100.00       .   |    0.00
       .  150.00 -100.00   |    0.00
       . -100.00  100.00   |   50.00

Nach dem Leerraeumen von Spalte 2:
  200.00 -100.00       .   |    0.00
       .  150.00 -100.00   |    0.00
       .       .   33.33   |   50.00

Nach dem Leerraeumen von Spalte 3:
  200.00 -100.00       .   |    0.00
       .  150.00 -100.00   |    0.00
       .       .   33.33   |   50.00
```

Interpretation der Ausgabe: Genau unsere zwei Papier-Schritte. Nach Spalte 1 ist die erste Spalte unter der Diagonale leer und in der Mitte steht die 150; nach Spalte 2 auch die zweite, und unten rechts steht die 33,33. Der Code hat nichts anderes getan als wir — nur schneller und ohne sich zu verrechnen. Das Rückwärtseinsetzen und die drei Verfahren dieses Kapitels stecken gebündelt im wiederverwendbaren Modul `programme/gemeinsam/loeser.py`, das die späteren Kapitel importieren.

4.3. Warum die FEM-Matrix fast nur aus Nullen besteht

Die Gauß-Elimination funktioniert für **jedes** lösbare System. Aber sie ist verschwenderisch, wenn das System viele Nullen enthält — und die FEM-Matrizen enthalten fast nur Nullen. Erinnern wir uns an den Matrix-Drucker aus Kapitel 3: Bei fünf Federn waren von den 36 Feldern der 6×6-Matrix nur 16 belegt, der Rest war leer. Drucken wir zum Vergleich die Matrix einer längeren Kette und daneben, wie eine „volle" Matrix gleicher Größe aussähe.

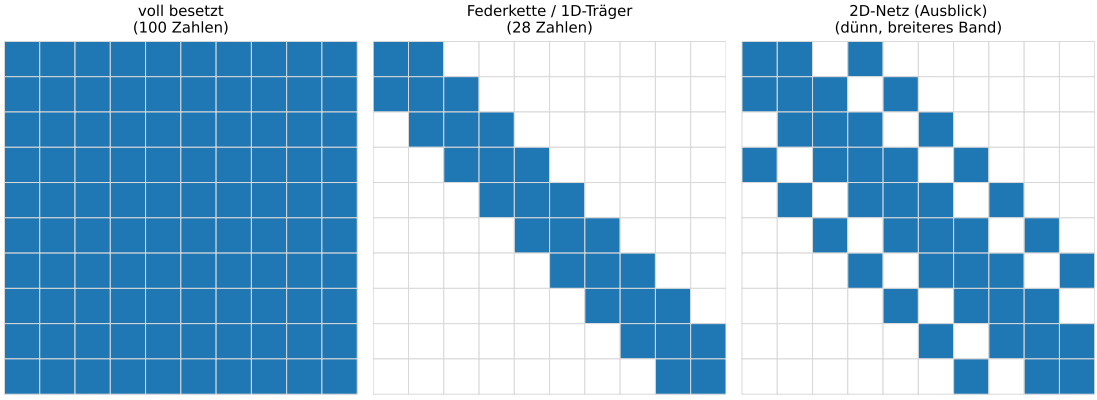

Abbildung 4.3.: Drei Besetzungsmuster im Vergleich (blau = hier steht eine Zahl, weiß = null). Links eine voll besetzte 10×10-Matrix: jedes Feld trägt eine Zahl. In der Mitte die Federkette (tridiagonal): nur die Diagonale und ihre beiden Nachbarn — ein schmales Band. Rechts, als Ausblick, eine 2D-Netzmatrix: auch dünn besetzt, aber mit einem breiteren Band und zwei zusätzlichen Nebenbändern. Je dünner das Band, desto billiger die Lösung.

Was man hier sieht: Dieselbe Größe, drei Welten. Die volle Matrix (links) hat in jedem der 100 Felder eine Zahl. Die Federkette (Mitte) hat nur 28 — die Diagonale und je eine Nebendiagonale; alles andere ist leer. Man nennt eine solche Matrix **dünn besetzt**: Der allergrößte Teil ihrer Felder ist null. Und weil die wenigen Zahlen sich als schmales **Band** um die Diagonale drängen, heißt sie eine **Bandmatrix**. Beides hat denselben physikalischen Grund, den wir in Kapitel 3 gesehen haben: Jeder Knoten hängt nur mit seinen **unmittelbaren Nachbarn** zusammen, mit keinem Knoten weiter weg. Rechts deutet sich an, wie es in 2D aussieht (ab 1): immer noch dünn, aber mit einem etwas breiteren Band.

Diese Leere ist bares Geld. Ein Löser, der die Nullen erst gar nicht ansieht, spart fast die gesamte Arbeit. Genau das tut der nächste Löser.

4.4. Der Thomas-Algorithmus: Gauß, der nur das Band anfasst

Schau noch einmal auf unsere Gauß-Elimination von eben. Beim Leerräumen der ersten Spalte mussten wir nur Zeile (II) anfassen — Zeile (III) hatte an der ersten Stelle ohnehin schon eine Null, dort war nichts zu eliminieren. Und beim Leerräumen der zweiten Spalte gab es überhaupt nur eine Zeile darunter. Die vielen Nullen haben uns Arbeit geschenkt, *ohne dass wir es geplant hatten*.

Der **Thomas-Algorithmus** macht aus diesem Geschenk ein System. Er ist nichts anderes als die Gauß-Elimination, aber er weiß von vornherein, dass die Matrix tridiagonal ist — dass also außerhalb des schmalen Bandes nur Nullen stehen. Darum fasst er in jeder Zeile nur drei Zahlen an: die auf der Diagonale und ihre beiden Nachbarn. Alles andere überspringt er, weil es sowieso null ist und null bleibt.

Man beschreibt ein tridiagonales System darum gar nicht mehr als volles Quadrat, sondern nur durch seine drei Diagonalen — drei Listen:

- die **untere** Nebendiagonale (die Zahlen links der Diagonale),
- die **Hauptdiagonale**,
- die **obere** Nebendiagonale (die Zahlen rechts der Diagonale).

Für unser reduziertes Federketten-System sind das:

$$\text{unten} = (\,-100,\,-100\,), \quad \text{Diagonale} = (\,200,\,200,\,100\,), \quad \text{oben} = (\,-100,\,-100\,).$$

Und jetzt kommt das Schöne: Wenn der Thomas-Algorithmus dieses Band abarbeitet, tut er **genau dieselben Rechnungen** wie unsere Gauß-Handrechnung vorhin — dieselben Faktoren $\frac{1}{2}$ und $\frac{2}{3}$, dieselben Zwischenzahlen 150 und $\frac{100}{3}$. Er läuft einmal von oben nach unten durch das Band (die Vorwärtswelle, das Leerräumen) und einmal von unten nach oben zurück (die Rückwärtswelle, das Ernten). Nur berührt er nie ein Feld außerhalb des Bandes. Bei drei Unbekannten fällt der Unterschied nicht auf; bei tausend spart er fast die gesamte Arbeit.

> **Geschichte: der Namensgeber des Thomas-Algorithmus**
>
> Der Algorithmus trägt den Namen von **Llewellyn Hilleth Thomas** (1903–1992), einem britischen Physiker. Bekannt wurde er zunächst durch einen ganz anderen Beitrag — die „Thomas-Präzession" in der Atomphysik der 1920er Jahre. Das Verfahren zum schnellen Lösen tridiagonaler Systeme beschrieb er **1949** in einem Bericht des Watson Scientific Computing Laboratory an der Columbia University (Thomas 1949). Dass ausgerechnet solche Systeme so oft auftauchen — bei Wärmeleitung, bei Schwingungen, überall dort, wo nur Nachbarn miteinander reden —, macht seinen kleinen, schnellen Algorithmus bis heute zu einem der meistbenutzten der Numerik.

Den Thomas-Löser haben wir als wiederverwendbaren Baustein in `programme/gemeinsam/loeser.py` abgelegt. Sein Kern ist kurz genug, um ihn ganz zu zeigen — die Vorwärtswelle, die das Band leerräumt, und die Rückwärtswelle, die erntet:

```
def thomas(unter, diagonale, ober, rechte_seite):
    """Loest ein tridiagonales System, ohne die Nullen anzufassen.

    unter: Nebendiagonale links (unter[0] wird nicht benutzt).
    diagonale: die Hauptdiagonale (n Zahlen).
    ober: Nebendiagonale rechts (ober[n-1] wird nicht benutzt).
    rechte_seite: die n rechten Seiten.
    Rueckgabe: Liste der n Loesungswerte.
    """
    n = len(diagonale)
    ober_neu = list(ober)
    seite_neu = list(rechte_seite)

    # Vorwaertswelle: von oben nach unten das Band leerraeumen.
    ober_neu[0] = ober[0] / diagonale[0]
    seite_neu[0] = rechte_seite[0] / diagonale[0]
    for i in range(1, n):
        nenner = diagonale[i] - unter[i] * ober_neu[i - 1]
        if i < n - 1:
            ober_neu[i] = ober[i] / nenner
        seite_neu[i] = (rechte_seite[i] - unter[i] * seite_neu[i - 1]) / nenner

    # Rueckwaertswelle: von unten nach oben ernten.
    loesung = [0.0] * n
    loesung[n - 1] = seite_neu[n - 1]
    for i in range(n - 2, -1, -1):
```

```
        loesung[i] = seite_neu[i] - ober_neu[i] * loesung[i + 1]
    return loesung


unter = [0.0, -100.0, -100.0]     # unter[0] gibt es nicht
diagonale = [200.0, 200.0, 100.0]
ober = [-100.0, -100.0, 0.0]      # ober[2] gibt es nicht
rechte_seite = [0.0, 0.0, 50.0]

loesung = thomas(unter, diagonale, ober, rechte_seite)
gerundet = []
for wert in loesung:
    gerundet.append(round(wert, 3))
print("Thomas-Lösung:", gerundet, "mm")
```

```
Thomas-Lösung: [0.5, 1.0, 1.5] mm
```

Interpretation der Ausgabe: $0{,}5$, $1{,}0$, $1{,}5$ — wieder dieselbe Probe. Der Thomas-Löser hat nie ein Feld außerhalb des Bandes berührt und liefert doch aufs Tausendstel dasselbe wie die volle Gauß-Elimination.

4.4.1. Der Thomas-Lauf als Welle durch den Träger

Weil der Thomas-Algorithmus das Band strikt der Reihe nach abarbeitet, kann man ihm zusehen wie einer Welle, die durch den Träger läuft. Die folgende Animation zeigt es an einer längeren Kette: Erst wandert die Vorwärtswelle von links nach rechts und räumt Knoten für Knoten leer (das Leerräumen), dann kehrt die Rückwärtswelle von rechts nach links zurück und trägt Knoten für Knoten den fertigen Verschiebungswert ein (das Ernten).

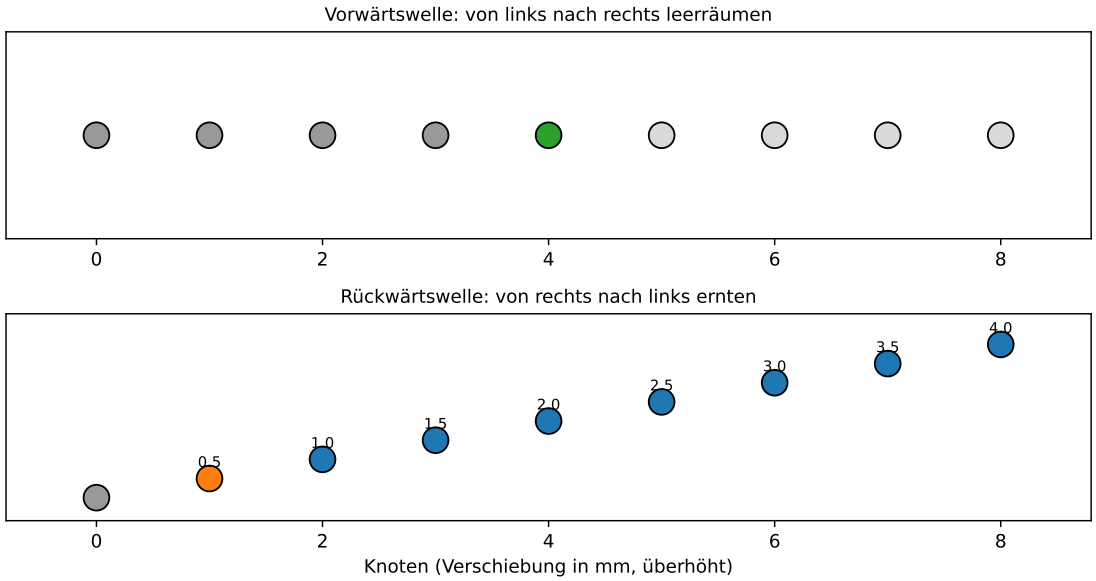

Abbildung 4.4.: Der Thomas-Lauf als Bildreihe (in der HTML-Fassung eine Animation). Oben die Vorwärtswelle, die die Knoten von links nach rechts leerräumt; unten die Rückwärtswelle, die von rechts nach links die Verschiebungen einträgt, bis die vollständige Treppe 0 / 0,5 / … / 4,0 mm steht.

Was man sieht: Der Algorithmus berührt jeden Knoten genau zweimal — einmal auf dem Hinweg (leerräumen), einmal auf dem Rückweg (ernten). Zwei Durchläufe durch die Kette, mehr nicht. Verdoppelt man die Kettenlänge, verdoppelt sich schlicht die Länge der beiden Wege: doppelt so viele Knoten, doppelt so viel Arbeit. Genau das Versprechen aus dem Vermutungs-Kasten.

4.5. Gauß-Seidel: die Lösung pendelt sich ein

Die beiden ersten Verfahren rechnen die Lösung in einem Zug aus — sie räumen leer und ernten, fertig. Das dritte Verfahren geht ganz anders vor, und es lohnt sich, weil es später die großen 2D-Netze bändigt und weil es ein Bild liefert, das den Kern der ganzen Physik trifft.

Die Idee heißt **Entspannung** (englisch *relaxation*). Stell dir vor, jeder Knoten wüsste seine eigene Gleichung — aber nur seine eigene. Knoten 3 etwa weiß: „Meine Verschiebung sollte in der Mitte zwischen meinen beiden Nachbarn liegen.” Er schaut also auf seine Nachbarn und rückt sich dorthin, wohin seine Gleichung ihn haben will. Dann macht es Knoten 4 genauso,

dann Knoten 2, und dann wieder von vorn. Kein Knoten sieht das ganze System; jeder korrigiert nur sich selbst, immer wieder. Und erstaunlicherweise pendelt sich die ganze Kette dabei in die richtige Lösung ein.

Machen wir das konkret. Wir stellen jede Zeile des reduzierten Systems nach *ihrer eigenen* Unbekannten um — nach der Zahl auf der Diagonale:

$$
\begin{aligned}
u_2 &= \tfrac{1}{200}\,(0 + 100\,u_3) = \tfrac{1}{2}\,u_3,\\
u_3 &= \tfrac{1}{200}\,(0 + 100\,u_2 + 100\,u_4) = \tfrac{1}{2}\,(u_2 + u_4),\\
u_4 &= \tfrac{1}{100}\,(50 + 100\,u_3) = 0{,}5 + u_3.
\end{aligned}
$$

Lies die mittlere Zeile als Satz: *„u_3 ist der Mittelwert seiner beiden Nachbarn.“* Das ist die Entspannungsregel in Reinform. Jetzt raten wir einen Startwert — sagen wir, wir wissen nichts und setzen **alle Verschiebungen auf 0** — und wenden die drei Regeln immer wieder an. Dabei benutzen wir jeden neuen Wert sofort weiter, sobald wir ihn haben (das ist der Unterschied zwischen Gauß-Seidel und seinem etwas langsameren Bruder, dem Jacobi-Verfahren).

Vorhersage-Punkt

Bevor du die Tabelle liest: Wir starten bei $0/0/0$, die richtige Lösung ist $0{,}5/1{,}0/1{,}5$. In der ersten Runde ändert sich nur u_4 (weil nur seine Gleichung eine Zahl von außen, die $0{,}5$, mitbringt) — die beiden anderen bleiben zunächst bei null. Was glaubst du: Sind wir nach 5 Runden schon nah dran, oder braucht es eher 20? Und rückt die Lösung von unten heran (immer zu klein) oder pendelt sie über das Ziel hinaus? Leg dich fest.

Rechnen wir die ersten Runden von Hand mit (jede Zahl aus der Regel darüber, immer mit den frischesten Nachbarwerten):

Tabelle 4.1.: Die ersten fünf Gauß-Seidel-Runden am reduzierten Federketten-System. Alle Werte kriechen von unten auf ihre Zielwerte 0,5 / 1,0 / 1,5 zu; die „größte Änderung” wird von Runde zu Runde kleiner.

Runde	u_2	u_3	u_4	größte Änderung
Start	0	0	0	—
1	0	0	0,500	0,500
2	0	0,250	0,750	0,250
3	0,125	0,438	0,938	0,188
4	0,219	0,578	1,078	0,141

Runde	u_2	u_3	u_4	größte Änderung
5	0,289	0,684	1,184	0,105

Man sieht es in Tabelle 4.1: Die Werte nähern sich **von unten** und pendeln nicht über das Ziel hinaus — jede Runde ein Stück näher, aber die Schritte werden kleiner. Nach fünf Runden sind wir noch nicht da (u_2 steht erst bei $0{,}29$ statt $0{,}5$). Das ist typisch: Gauß-Seidel braucht viele kleine Runden, wo Gauß und Thomas in einem Zug fertig sind. Sein Vorteil liegt woanders — bei den großen 2D-Netzen, wo eine volle Elimination zu teuer wird.

4.5.1. Woher weiß man, wann man aufhören darf?

Anders als Gauß hat Gauß-Seidel kein natürliches Ende — man könnte ewig weiterrechnen und käme der Lösung immer noch ein Härchen näher. Man braucht also ein **Abbruchkriterium**: eine Regel, die sagt „gut genug ist gut genug". Die einfachste und ehrlichste ist die letzte Spalte von Tabelle 4.1: Wenn sich in einer ganzen Runde **kein** Wert mehr um mehr als eine kleine Schwelle ändert — etwa um weniger als $0{,}001$ mm —, dann bewegt sich nichts mehr nennenswert, und wir hören auf. Man verlangt also nicht die *perfekte* Lösung (die man ohne Gauß gar nicht kennt), sondern eine, die sich nicht mehr merklich verbessert.

4.5.2. Das Einpendeln als Animation

Die Tabelle zeigt Zahlen; das Bild zeigt die Bewegung. Die folgende Animation lässt die Federkette Runde um Runde entspannen. Sie startet als flache Linie bei null (alle Verschiebungen falsch) und schwingt sich in die gleichmäßige Treppe $0/0{,}5/1{,}0/1{,}5$ ein — genau die Lösung, die wir in Kapitel 3 von Hand gefunden haben.

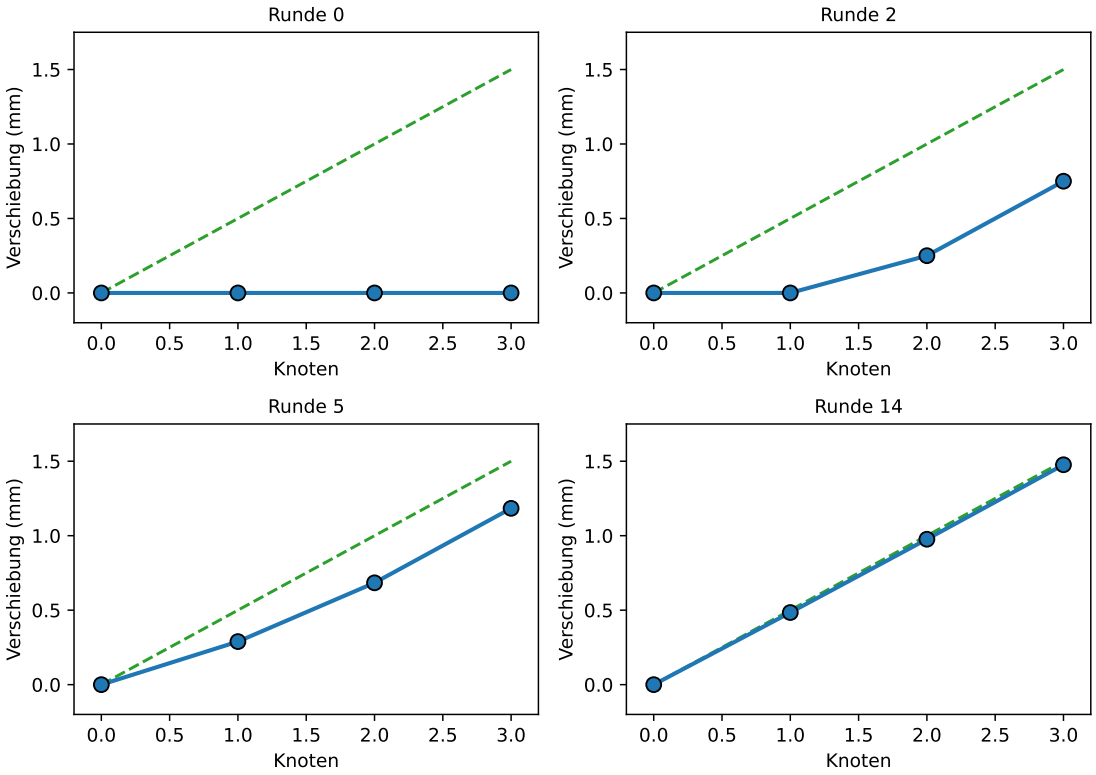

Abbildung 4.5.: Das Einpendeln als Bildreihe (in der HTML-Fassung eine Animation): das Verschiebungsprofil in den Runden 0, 2, 5 und 14. Von der flachen Null-Linie schwingt es sich stetig in die Ziel-Treppe 0 / 0,5 / 1,0 / 1,5 mm (grün gestrichelt) ein.

Was man sieht: Aus der falschen flachen Linie wächst Runde um Runde die richtige Treppe. Das ist mehr als eine Rechenspielerei — es ist ein Bild für etwas Physikalisches. Später, wenn dieselbe Sorte Gleichungssystem nicht Verschiebungen, sondern **Temperaturen** trägt (ab Kapitel 7), beschreibt genau dieses Einpendeln, wie sich der Träger im Brandfall auf sein endgültiges Temperaturprofil einschwingt. Die Entspannungsregel „jeder Knoten ist der Mittelwert seiner Nachbarn" ist dann kein Rechentrick mehr, sondern die Physik selbst. Doch das ist die Geschichte von Teil III — hier bleibt es bei der Federkette.

4.5.3. Wie schnell wird der Fehler kleiner?

Man möchte wissen, wie viele Runden Gauß-Seidel braucht. Tragen wir dazu den Fehler — die größte Änderung pro Runde aus Tabelle 4.1 — über

der Rundenzahl auf. Weil der Fehler in jeder Runde um denselben *Bruchteil* schrumpft (grob um ein Viertel), fällt er anfangs schnell und dann immer zäher; eine gewöhnliche Achse würde das schlecht zeigen. Darum benutzen wir für den Fehler eine besondere Achsenteilung.

Was ist eine logarithmische Achse?

Auf einer **gewöhnlichen** Achse haben gleiche *Abstände* gleiche *Differenzen*: von 1 bis 2 ist es genauso weit wie von 100 bis 101. Auf einer **logarithmischen** Achse haben gleiche Abstände gleiche *Faktoren*: von 1 bis 10 ist es genauso weit wie von 10 bis 100 oder von 0,01 bis 0,1 — jeder gleiche Schritt bedeutet „mal zehn". Das ist genau richtig für etwas, das in jeder Runde denselben Bruchteil wegnimmt: Auf der Log-Achse wird aus dem zäh abfallenden Bogen eine **gerade Linie**, und ihre Steilheit sagt unmittelbar, wie schnell das Verfahren ist. Fällt die Gerade steil, halbiert sich der Fehler in wenigen Runden; fällt sie flach, braucht es viele.

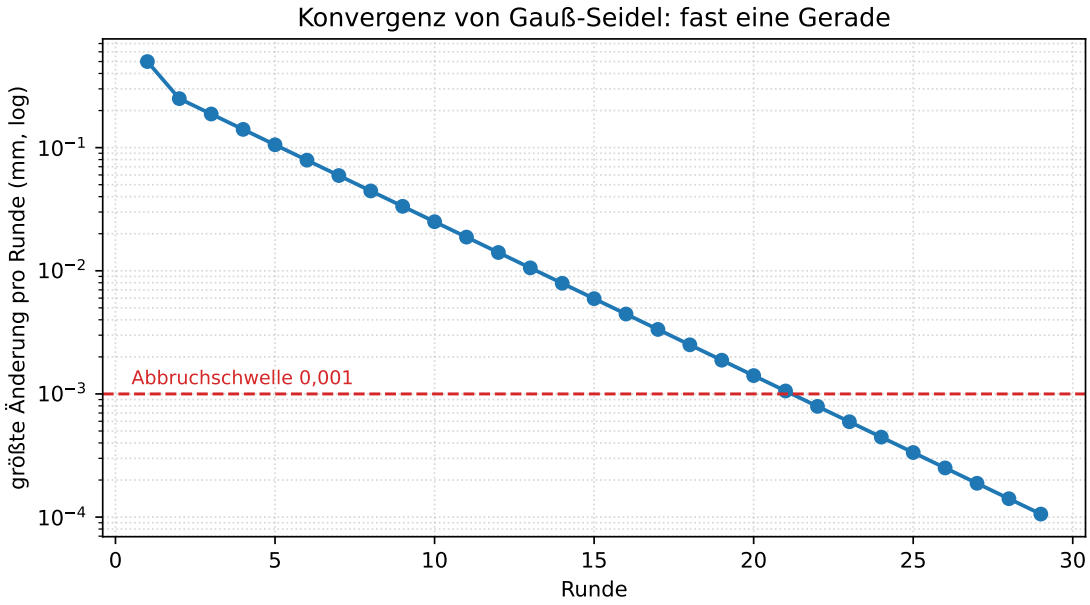

Abbildung 4.6.: Der Gauß-Seidel-Fehler (größte Änderung pro Runde) über der Rundenzahl, mit logarithmisch geteilter senkrechter Achse. Weil jede Runde denselben Bruchteil des Fehlers wegnimmt, liegen die Punkte fast auf einer Geraden — die Signatur eines Verfahrens mit gleichmäßiger Konvergenz. Die gestrichelte Linie markiert die Abbruchschwelle 0,001; sie wird nach 22 Runden unterschritten.

Was man hier sieht: Auf der logarithmischen Achse liegen die Punkte nahezu auf einer Geraden, die stetig fällt. Das ist die Handschrift eines Verfahrens, das den Fehler Runde um Runde um denselben Bruchteil verkleinert. Wo die Punkte die rote Schwelle $0{,}001$ unterschreiten — nach 22 Runden —, gilt die Lösung als gut genug. Für dieses winzige System sind 22 Runden viel (Thomas war in einem Durchlauf fertig); der Lohn kommt erst bei großen 2D-Netzen.

4.6. Was kostet das alles?

Jetzt können wir die Frage aus dem Vermutungs-Kasten mit Zahlen beantworten. „Kosten" meint hier die Anzahl der einfachen Rechenschritte (eine Multiplikation und eine Subtraktion), die ein Löser braucht — denn die Rechenzeit hängt fast nur daran.

Für die **volle Gauß-Elimination** muss man beim Leerräumen für jede der n Spalten alle darunter liegenden Zeilen anfassen, und in jeder Zeile alle Einträge. Das sind grob $n \times n \times n$, also n^3 Schritte — genauer etwa ein Drittel davon, $n^3/3$. Für den **Thomas-Algorithmus** dagegen ist in jeder Zeile nur das schmale Band zu bearbeiten, immer gleich viel Arbeit pro Zeile; das ergibt grob $8\,n$ Schritte — proportional zu n, nicht zu n^3. Setzen wir Zahlen ein:

Tabelle 4.2.: Rechenschritte für volle Gauß-Elimination gegenüber dem Thomas-Löser. Bei jeder Verzehnfachung der Unbekannten wächst die volle Elimination um das Tausendfache (zehn hoch drei), der Thomas-Löser nur um das Zehnfache.

Unbekannte n	volle Elimination ($\approx n^3/3$)	Thomas ($\approx 8n$)
10	≈ 330	80
100	≈ 330 000	800
1 000	≈ 330 Millionen	8 000
1 000 000 (Crashtest)	$\approx 3 \cdot 10^{17}$	8 Millionen

Tabelle 4.2 macht den Vermutungs-Kasten greifbar. Übersetzen wir die beiden Extremzeilen in Zeit — für einen Rechner, der eine Milliarde einfache Schritte pro Sekunde schafft (eine vorsichtige Annahme für einen heutigen Computer):

- Bei **1 000** Unbekannten: volle Elimination rund $0{,}3$ Sekunden, Thomas ein Wimpernschlag von acht Millionstel Sekunden. Beides schnell — hier merkt man den Unterschied kaum.
- Bei **einer Million** Unbekannten (Größenordnung eines echten Crashtests): volle Elimination rund $3 \cdot 10^{17}$ Schritte — das wären über **zehn Jahre** Rechnen. Der Thomas-Löser bräuchte 8 Millionen Schritte, also **acht Tausendstelsekunden**.

Zehn Jahre gegen einen Wimpernschlag — das ist der ganze Unterschied zwischen „unmöglich" und „geht sofort", und er kommt allein daher, dass der eine Löser die vielen Nullen ausnutzt und der andere nicht. Deshalb ist die Bandstruktur aus Kapitel 3 kein hübsches Detail, sondern die Bedingung dafür, dass große Simulationen überhaupt existieren.

> **i** Das Kleingedruckte: Warum wir nie umsortieren müssen
>
> In Lehrbüchern zur Gauß-Elimination liest man oft von **Pivotisierung** — dem Vertauschen von Zeilen, damit auf der Diagonale keine Null (oder keine zu kleine Zahl) steht, durch die man dann nicht teilen könnte. Wir haben das im ganzen Kapitel nie gebraucht, und das ist kein Glück, sondern hat einen Grund: Die FEM-Matrizen dieses Buches sind **diagonaldominant** — auf der Diagonale steht immer die betragsgrößte Zahl der Zeile (bei der Federkette die 200 gegen zwei -100 daneben). Das kommt direkt aus der Physik: Ein Knoten reagiert auf seine eigene Bewegung stärker als auf die jedes einzelnen Nachbarn. Solche Systeme kann man stur von oben nach unten eliminieren, ohne je umzusortieren — und Gauß-Seidel ist bei ihnen sogar garantiert gutmütig, es pendelt sich immer ein. Wann genau ein Verfahren konvergiert, ist eine eigene Wissenschaft, die wir hier nicht betreten; für unsere gutartigen Matrizen dürfen wir uns auf sie verlassen.

4.7. Die interaktive Einheit: der gläserne Löser

Jetzt bist du dran. Das folgende Programm ist ein **gläserner Löser** — er rechnet nicht nur, sondern zeigt bei jedem Schritt sein Inneres. Der erste Teil führt die Gauß-Elimination vor und druckt die Matrix nach jedem geräumten Spalten-Schritt (die Nullen als Punkte). Du kannst die Systemgröße zwischen 3 und 6 einstellen — das Programm baut dann eine Federketten-Matrix dieser Größe und räumt sie vor deinen Augen leer.

Der Code ist genau die Papier-Elimination von oben, in Python gegossen: dieselbe Doppelschleife, dieselben Namen (`pivot`, `faktor`, leerräumen).

Vorhersage-Punkt

Bevor du ausführst: Stell `GROESSE = 4` ein. Wie viele Eliminationsschritte (geräumte Spalten) wird es geben, bis die Dreiecksform steht? Und im zweiten Teil, dem Gauß-Seidel: Wie viele Runden braucht es wohl, bis sich kein Wert mehr um mehr als $0{,}001$ ändert — mehr oder weniger als die 22 aus dem kleinen Beispiel? Leg dich fest, dann probiere es.

```
#| autorun: true
# Der glaeserne Loeser. Aendere GROESSE (3 bis 6) und sieh zu, wie die
# Gauss-Elimination die Matrix Spalte fuer Spalte leerraeumt.
GROESSE = 4

def baue_federkette(groesse):
    """Baut eine reduzierte Federketten-Matrix (tridiagonal) dieser Groesse."""
    matrix = []
    for zeile in range(groesse):
        neue_zeile = []
        for spalte in range(groesse):
            neue_zeile.append(0.0)
        matrix.append(neue_zeile)
    for i in range(groesse):
        matrix[i][i] = 200.0
        if i > 0:
            matrix[i][i - 1] = -100.0
        if i < groesse - 1:
            matrix[i][i + 1] = -100.0
    matrix[groesse - 1][groesse - 1] = 100.0   # freies Ende: nur eine Feder
    rechte_seite = [0.0] * groesse
    rechte_seite[groesse - 1] = 50.0
    return matrix, rechte_seite


def drucke(matrix, seite):
    """Druckt Matrix und rechte Seite; Nullen werden zu Punkten."""
    for zeile in range(len(matrix)):
        for wert in matrix[zeile]:
            if abs(wert) < 0.0001:
                print(f"{'.':>8}", end="")
            else:
                print(f"{wert:8.2f}", end="")
        print(f"   | {seite[zeile]:7.2f}")
    print()
```

```
def gauss_zeige(matrix, rechte_seite):
    """Vorwaertselimination mit Ausgabe nach jedem Schritt; erntet dann."""
    n = len(matrix)
    werk = []
    for zeile in matrix:
        werk.append(list(zeile))
    seite = list(rechte_seite)
    print("Startsystem:")
    drucke(werk, seite)
    for spalte in range(n):
        pivot = werk[spalte][spalte]
        for zeile in range(spalte + 1, n):
            faktor = werk[zeile][spalte] / pivot
            for k in range(spalte, n):
                werk[zeile][k] = werk[zeile][k] - faktor * werk[spalte][k]
            seite[zeile] = seite[zeile] - faktor * seite[spalte]
        print(f"Nach dem Leerraeumen von Spalte {spalte + 1}:")
        drucke(werk, seite)
    # Ernten (Rueckwaertseinsetzen).
    loesung = [0.0] * n
    for zeile in range(n - 1, -1, -1):
        summe = seite[zeile]
        for k in range(zeile + 1, n):
            summe = summe - werk[zeile][k] * loesung[k]
        loesung[zeile] = summe / werk[zeile][zeile]
    return loesung


matrix, rechte_seite = baue_federkette(GROESSE)
loesung = gauss_zeige(matrix, rechte_seite)
gerundet = []
for wert in loesung:
    gerundet.append(round(wert, 3))
print("Geerntete Loesung:", gerundet, "mm")

# --- Abbildung ---
import matplotlib.pyplot as plt
knoten = list(range(GROESSE + 1))
verschiebung = [0.0] + loesung
fig, ax = plt.subplots(figsize=(7.0, 3.0))
ax.plot(knoten, verschiebung, "o-", lw=2.0, ms=9, markeredgecolor="black")
ax.set_xlabel("Knoten")
```

```
ax.set_ylabel("Verschiebung (mm)")
ax.set_title("Gelöste Federkette (%d freie Knoten)" % GROESSE)
plt.tight_layout()
plt.show()
```

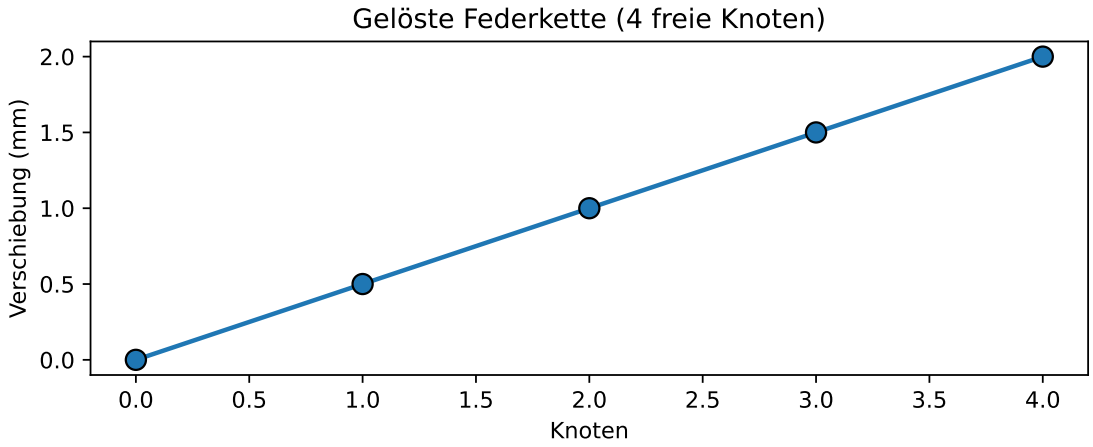

Abbildung 4.7.: Vorgerenderte Fassung des gläsernen Lösers für vier freie Knoten: die gelöste Federkette als gleichmäßig ansteigende Treppe. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst — stell die Systemgröße um und sieh zu, wie sich die Treppe verlängert.

Was man sieht: Für vier freie Knoten steht die Treppe $0{,}5$, $1{,}0$, $1{,}5$, $2{,}0$ mm — vier gleiche Federn, jede um denselben halben Millimeter gedehnt. In der Textausgabe darüber siehst du, wie die Matrix Spalte für Spalte leergeräumt wird, bis die Dreiecksform steht. Probiere andere Größen und beobachte, wie viele Schritte das Leerräumen jeweils braucht — bei n freien Knoten sind es $n - 1$ geräumte Spalten.

4.8. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript zu diesem Kapitel liegt in `programme/kap04/kap04_federkette_maschinell.py`. Es ist die im Buch lange versprochene **Verheiratung** der beiden Bausteine: Der Matrix-Drucker aus Kapitel 3 (`baue_systemmatrix`) baut die Systemmatrix, der Löser aus diesem Kapitel (`thomas`) knackt sie. Zur Kontrolle löst es dasselbe System noch ein zweites Mal mit der vollen `gauss_elimination` — beide müssen aufs Tausendstel übereinstimmen.

Als Aufgabe nimmt es eine Kette aus **acht** Federn — mehr, als man bequem von Hand rechnet. Es assembliert die 9×9-Systemmatrix, druckt ihre Bandstruktur, baut die Randbedingung ein (Knoten 1 fest), liest die drei Diagonalen ab und löst. Heraus kommt die erwartete Treppe in $0{,}5$-mm-Schritten bis zu $4{,}0$ mm am freien Ende. So schließt sich der Bogen von Kapitel 3 (assemblieren) zu 2 (lösen): Die FEM-Maschinerie steht jetzt vollständig — für die Federkette. Ab Kapitel 7 trägt dieselbe Maschinerie Wärme.

Merkkasten

- Ein Gleichungssystem **lösen** heißt: Zahlen finden, die alle Gleichungen zugleich erfüllen — im 2×2-Fall der Schnittpunkt zweier Geraden.
- **Gauß-Elimination**: erst unter der Diagonale **leerräumen** (Dreiecksform), dann von unten nach oben **ernten** (Rückwärtseinsetzen). Funktioniert immer, kostet aber rund $n^3/3$ Schritte.
- FEM-Matrizen sind **dünn besetzt** (Bandmatrizen): fast nur Nullen, weil jeder Knoten nur mit seinen Nachbarn zusammenhängt.
- Der **Thomas-Algorithmus** ist die Gauß-Elimination, die nur das Band anfasst — eine Vorwärts- und eine Rückwärtswelle, Kosten nur $\approx 8n$.
- **Gauß-Seidel** entspannt: Jeder Knoten rückt sich wiederholt in den Mittelwert seiner Nachbarn; die Lösung pendelt sich ein. Abbruch, wenn sich nichts mehr merklich ändert.
- Die vielen Nullen auszunutzen ist kein Luxus: Bei einer Million Unbekannten entscheidet es über **zehn Jahre gegen einen Wimpernschlag**.

Roter Faden

Zurück: Das 4×4-System der Federkette aus Kapitel 3 haben wir dort von Hand gelöst und das reduzierte 3×3-System als Probe behalten. Dieses Kapitel hat es **maschinell** gelöst — dreimal, mit Gauß, Thomas und Gauß-Seidel, und jedes Mal kamen die vertrauten $0{,}5/1{,}0/1{,}5$ mm heraus. Die Bandstruktur, die Kapitel 3 als hübsches Muster einführte, wurde hier zur Rechenersparnis.

Vor: Der Thomas-Löser wird ab Kapitel 7 den 1D-Träger lösen — dann tragen die Gleichungen Temperaturen statt Verschiebungen, aber die Matrix ist dieselbe tridiagonale Bandmatrix. Das Einpendeln von Gauß-Seidel wird in Kapitel 9 zur physikalischen Anschauung, wie sich der Träger zeitlich auf sein Temperaturprofil einschwingt, und in 1 bändigt Gauß-Seidel die

größeren 2D-Netze. Und die Kostenrechnung dieses Kapitels ist der Grund, warum das interaktive Finale in 3 seine Netze deckelt: Reines Python im Browser verträgt nur so viele Unbekannte, wie ein Bandlöser in Sekunden schafft.

Übungen

Ü 4.1 (Verstehen). Löse das folgende 3×3-System von Hand mit Gauß-Elimination — erst leerräumen, dann ernten:

$$\begin{aligned} 2x &+ y &&= 4,\\ x &+ 3y &+ z &= 7,\\ &y &+ 2z &= 5. \end{aligned}$$

Schreibe jeden Schritt hin und prüfe dein Ergebnis anschließend mit dem gläsernen Löser, indem du das System dort einträgst.

> **i** Musterlösung zu Ü 4.1
>
> **Leerräumen, Spalte 1:** Zeile 2 hat vorne eine 1, Zeile 1 eine 2. Ziehe die Hälfte von Zeile 1 von Zeile 2 ab (Faktor $\frac{1}{2}$): Zeile 2 wird $0\,x + (3 - \frac{1}{2})\,y + z = 7 - 2$, also $2{,}5\,y + z = 5$. Zeile 3 hat vorne schon eine 0.
> **Leerräumen, Spalte 2:** Unter der $2{,}5$ in Zeile 2 steht in Zeile 3 eine 1. Ziehe $\frac{1}{2{,}5} = 0{,}4$ von Zeile 2 von Zeile 3 ab: Zeile 3 wird $(1 - 0{,}4)·? $ … konkret y-Glied $1 - 0{,}4 \cdot 2{,}5 = 0$, z-Glied $2 - 0{,}4 \cdot 1 = 1{,}6$, rechte Seite $5 - 0{,}4 \cdot 5 = 3$. Also $1{,}6\,z = 3$.
> **Ernten:** $z = 3 / 1{,}6 = 1{,}875$; dann $2{,}5\,y = 5 - z = 3{,}125$, also $y = 1{,}25$; dann $2x = 4 - y = 2{,}75$, also $x = 1{,}375$. Probe durch Einsetzen in die erste Zeile: $2 \cdot 1{,}375 + 1{,}25 = 4$. ∎

Ü 4.2 (Verändern). Gauß-Seidel braucht am reduzierten Federketten-System vom Nullstart aus 22 Runden. Füttere es stattdessen mit einem **besseren Startwert** — einem linear ansteigenden Profil, das die richtige *Form* schon erahnt (etwa $0{,}4$, $0{,}8$, $1{,}2$). **Sage zuerst vorher:** Spart das viele Runden oder wenige? Rechne dann nach; das Skript `loesungen/kap04_ue2.py` vergleicht beide Startwerte.

> **Musterlösung zu Ü 4.2**
>
> Der lineare Startwert $0{,}4/0{,}8/1{,}2$ braucht nur **16** Runden statt 22 — sechs gespart. Der Grund: Die wahre Lösung ist eine gleichmäßige Treppe, also linear ansteigend. Der Nullstart hat von dieser Form gar nichts und muss sie erst aufbauen; das lineare Profil hat die Form schon und muss nur noch die Höhe nachjustieren. Beide landen bei denselben $0{,}5/1{,}0/1{,}5$ mm — ein guter Startwert ändert das Tempo, nie das Ziel. (Das ist der Grund, warum man in der Praxis eine grobe Näherung als Startwert verwendet, wo man eine hat — etwa die Lösung des vorigen Zeitschritts.)

Ü 4.3 (Übertragen). Verheirate den Matrix-Drucker aus Kapitel 3 mit dem Thomas-Löser aus diesem Kapitel und löse eine Kette aus **acht** gleichen Federn maschinell: assemblieren, Randbedingung einbauen, die drei Diagonalen ablesen, lösen. Welche Verschiebung hat das freie Ende? Das Skript `loesungen/kap04_ue3.py` baut es Schritt für Schritt (und das Kapitel-Programm `programme/kap04/` zeigt die ausführliche Fassung).

> **Musterlösung zu Ü 4.3**
>
> Acht gleiche Federn, jede unter der durchgereichten Kraft von 50 N um $0{,}5$ mm gedehnt: Die Knoten stehen bei $0{,}5$, $1{,}0$, $1{,}5$, …, $4{,}0$ mm. Das freie Ende (Knoten 9) kommt genau $8 \cdot 0{,}5 = 4{,}0$ mm weit. Der Thomas-Löser liefert das, ohne je ein Feld außerhalb des schmalen Bandes anzufassen — und die Gauß-Elimination als Kontrolle im Kapitel-Programm bestätigt es aufs Tausendstel.

Das Kleingedruckte

Ein paar ehrliche Feinheiten abseits des Hauptpfads. **Erstens:** Wir haben Gauß-Seidel nur an einem gutmütigen System gezeigt und behauptet, es pendle sich „immer" ein. Das gilt für die diagonaldominanten Matrizen dieses Buches; für beliebige Systeme kann Gauß-Seidel auch davonlaufen statt zu konvergieren. Wann genau es funktioniert, ist eine eigene Theorie, die wir bewusst nicht betreten — unsere FEM-Matrizen sind stets auf der sicheren Seite. **Zweitens:** Die Kostenformeln $n^3/3$ und $8n$ sind gerundete Faustzahlen; die genaue Schrittzahl hängt von Feinheiten der Umsetzung ab. Für die Größenordnung — und darum geht es hier — sind sie belastbar. **Drittens:** Es gibt weit mehr Löser als diese drei (Cholesky, konjugierte Gradienten, Mehrgitter …), und schnellere für riesige Netze. Wir nehmen die drei, weil

man sie ganz durchschaut und weil sie den Rest des Buches tragen. Wer später ein großes Profi-Programm benutzt, wird andere Namen lesen — aber das Grundprinzip, die vielen Nullen auszunutzen, steckt in ihnen allen.

Teil III.

Teil III — Der Balken in 1D: von der Physik zur FEM

5. Steigung und Fläche

Bis hierher war der Träger eine **Kette**: Knoten, die an ihren Nachbarn ziehen, Federn, die sich addieren, ein Gleichungssystem, das wir lösen. Kein einziges Mal mussten wir dabei etwas ableiten oder integrieren — reine Schulalgebra, wie versprochen. Jetzt, zu Beginn des dritten Teils, kommt zum ersten Mal die *Physik der Wärme* ins Spiel, und mit ihr zwei Fragen, die uns von nun an bei jedem Temperaturprofil begegnen werden.

Schau dir die Momentaufnahme unseres Trägers an: dieselbe Szene wie in Kapitel 1, der Brandherd rechts unten, die Wärme unterwegs nach links dem kühlen Auflager entgegen — eingefroren eine halbe Stunde nach dem Ausbruch des Feuers. Und das Erste, was auffällt: Nach dreißig Minuten liegt die linke Hälfte des Trägers noch fast so kalt da wie zu Beginn. Die Hitze ist erst gut eine Handbreit weit ins Metall gekrochen; die Mitte weiß noch nichts vom Feuer.

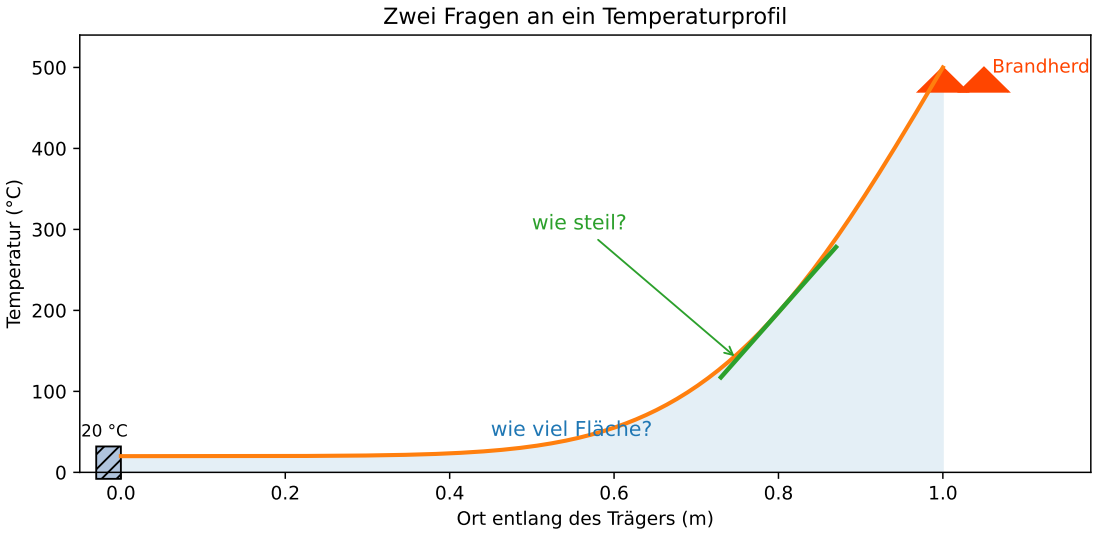

Abbildung 5.1.: Das Temperaturprofil des Trägers nach 30 Minuten (die Momentaufnahme aus Kapitel 1, hier als glatte Kurve). Links hält das kühle Auflager 20 °C, rechts sitzt der Brandherd (500 °C); dazwischen liegt der Träger über die linke Hälfte noch fast kalt und steigt erst nahe dem Feuer steil an. Zwei Fragen stellt uns dieser Verlauf ab jetzt immer wieder: Wie steil ist er nahe am Brandherd (grüner Pfeil)? Und wie viel Fläche liegt unter ihm (blau)?

Was man hier sieht: kein gleichmäßiges Gefälle, sondern eine Kurve, die über die linke Hälfte fast flach bei 20 °C liegt und erst zum Brandherd hin steil emporschießt. Das ist der Fingerabdruck von Stahl: Die Wärme braucht Zeit, um vorzudringen — nach einer halben Stunde reicht ihr spürbarer Einfluss nur etwa eine Handbreit weit ins Metall. Zwei Fragen drängen sich auf. Die erste: **Wie steil** steigt die Temperatur zum Brandherd hin? Denn ein steiles Gefälle treibt viel Wärme — das ist der Motor des Wärmestroms, den Kapitel 6 aufbaut. Die zweite: **Wie viel Fläche** liegt unter der Kurve? Denn diese Fläche ist ein Maß für die Wärme, die insgesamt im Träger steckt — und genau solche Flächen wird Kapitel 7 für jedes Element ausrechnen.

Steigung und Fläche: **mehr Analysis braucht dieses Buch nicht.** Und das Beste kommt gleich zu Beginn — beide lassen sich für Geraden mit reiner Schulgeometrie *exakt* bestimmen, mit dem Dreieck und dem Trapez. Weil unsere FEM fast nur mit Geraden rechnet, ist das keine Nebensache, sondern der ganze Trick.

Die kalte Mitte — warum Feuerwiderstand in Minuten zählt

Nach dreißig Minuten steht die Mitte des Trägers bei $x = 0{,}5$ m gerade einmal bei rund 32 °C — kaum mehr als handwarm, obwohl am rechten Ende 500 °C herrschen. Das ist kein Rechenfehler, sondern die Trägheit des Stahls: In der Zeit t wandert der spürbare Wärmeeinfluss nur etwa $\sqrt{a \cdot t}$ weit ins Metall, mit der **Temperaturleitfähigkeit** $a = 1{,}385 \cdot 10^{-5}\ \mathrm{m^2/s}$ des Stahls. Nach einer halben Stunde sind das $\sqrt{1{,}385 \cdot 10^{-5} \cdot 1800} \approx 0{,}16$ m — rund 16 Zentimeter. Genau deshalb misst man Feuerwiderstand in **Minuten** — R30, R60, R90 —: Es ist eine Frage der Zeit, wie weit die Hitze bis zum tragenden Kern und bis zum Auflager vordringt. Für dieses Kapitel heißt das: In der Mitte ist noch fast nichts los. Wollen wir sehen, wie *steil* die Temperatur ansteigt, müssen wir dorthin schauen, wo etwas geschieht — nahe an den Brandherd, zu $x = 0{,}8$ m.

Lernziele

Nach diesem Kapitel kannst du …

1. … die **Steigung** eines Temperaturprofils an einer Stelle aus zwei nahen Punkten berechnen (den Differenzenquotienten $\Delta T / \Delta x$) und in °C/m angeben,
2. … **erklären**, was beim Schrumpfen des Abstands h passiert und was die „Ableitung" $\mathrm{d}T/\mathrm{d}x$ dann bedeutet,
3. … eine **Fläche** unter einer Kurve mit Rechteckstreifen annähern und die Näherung durch mehr Streifen verbessern,
4. … für Geraden und Stücke aus Geraden Steigung und Fläche **exakt** angeben — mehr braucht die FEM dieses Buches nicht,
5. … das Symbol $\int$ lesen als „Summe aus unendlich schmalen Streifen".

Naheliegende Vermutung

Vermutung: *„Für die Finite-Elemente-Methode braucht man das ganze Analysis-Studium: Ableitungsregeln, Integraltabellen, Grenzwerte. Ohne das kommt man an eine Wärmerechnung gar nicht heran."*
Warum sie naheliegt: Ableiten und Integrieren gelten als der schwere Stoff der Oberstufe und der ersten Semester. Und die FEM rechnet doch Wärmeleitung aus — die steht in den Lehrbüchern voller Integrale.
Was stattdessen stimmt: Die schweren Werkzeuge stecken in der

Methode, nicht im Anwender. Unsere Ansatzfunktionen sind **Geraden** (Kapitel 7). Die Steigung einer Geraden ist ein Dreieck (Höhe durch Breite), die Fläche unter einer Geraden ist ein Trapez — beides Schulgeometrie, beides *exakt*, ganz ohne Grenzwert. Der ganze „unendlich klein"-Apparat wird genau einmal gebraucht, um zu *begründen*, warum die Geradenrechnung erlaubt ist; danach rechnest du nur noch mit Dreiecken und Trapezen. Dieses Kapitel zeigt beide Seiten: erst, was „unendlich klein" anschaulich heißt, dann, wie wenig davon beim Rechnen übrig bleibt.

Erste Frage: Wie steil?

5.1. Die Steigung zwischen zwei Punkten

Fangen wir handfest an. „Wie steil steigt die Temperatur?" heißt: Um wie viel Grad ändert sie sich, wenn ich ein Stück am Träger weitergehe? Das ist genau die Frage nach einer **Steigung** — dieselbe, die du von einer Straße kennst: „zehn Prozent Steigung" heißt zehn Meter Höhe auf hundert Meter Weg, also Höhenunterschied geteilt durch zurückgelegte Strecke.

Beim Träger ist die „Höhe" die Temperatur und der „Weg" der Ort. Weil in der Mitte noch fast nichts passiert, gehen wir nahe an den Brandherd. Nimm die Stelle $x = 0{,}8$ m und den Punkt ganz rechts am Brandherd bei $x = 1{,}0$ m. Der Fühler zeigt dort

$$T(0{,}8) = 197{,}77\ °\mathrm{C}, \qquad T(1{,}0) = 500{,}00\ °\mathrm{C}.$$

Die Verbindungslinie dieser beiden Punkte nennt man eine **Sekante** (von lateinisch *secare*, schneiden — sie *schneidet* die Kurve in zwei Punkten). Ihre Steigung ist der Temperaturunterschied geteilt durch den Ortsunterschied:

$$\text{Steigung} = \frac{\Delta T}{\Delta x} = \frac{T(1{,}0) - T(0{,}8)}{1{,}0 - 0{,}8} = \frac{500{,}00 - 197{,}77}{0{,}2} = \frac{302{,}23}{0{,}2} = 1511{,}2\,\frac{°\mathrm{C}}{\mathrm{m}}.$$

Das Zeichen Δ (großes griechisches Delta) heißt schlicht „Unterschied": ΔT ist der Temperaturunterschied, Δx der Ortsunterschied. Und die **Einheit** verrät sofort, was die Zahl bedeutet: °C pro Meter. Über diese zwanzig Zentimeter steigt die Temperatur im Schnitt um rund 1500 Grad je Meter —

das ganze Gefälle drängt sich in den schmalen Streifen nahe dem Brandherd. Abbildung 5.2 zeichnet die Sekante ein.

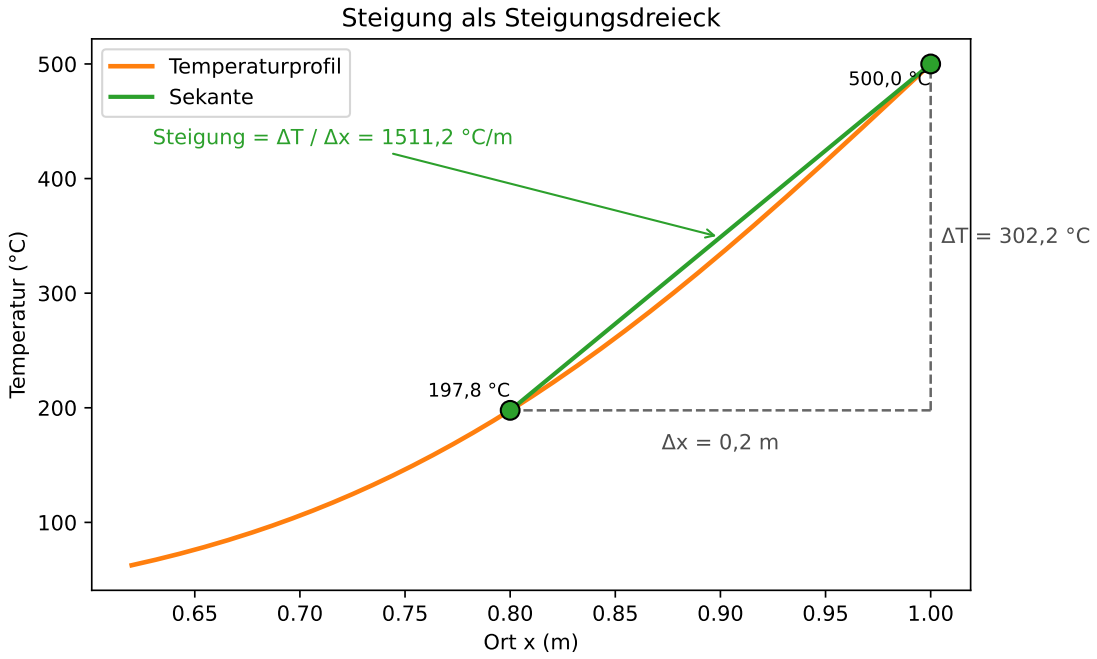

Abbildung 5.2.: Die Sekante durch die Punkte bei x = 0,8 m (197,8 °C) und x = 1,0 m am Brandherd (500 °C). Das Steigungsdreieck darunter zeigt Δx = 0,2 m (waagerecht) und ΔT = 302,2 °C (senkrecht); die Steigung ist ihr Verhältnis, 1511,2 °C/m. Die Sekante ist flacher als die Kurve dicht am Brandherd und steiler als links davon — sie mittelt über das Stück.

Was man hier sieht: ein rechtwinkliges **Steigungsdreieck**. Die waagerechte Kathete ist $\Delta x = 0{,}2$ m, die senkrechte $\Delta T = 302{,}2$ °C, und die Steigung der Sekante ist ihr Verhältnis. Das ist reine Schulgeometrie — kein Grenzwert, keine Ableitungsregel. Genau dieses Dreieck werden wir gleich immer kleiner machen.

Bevor wir das tun, gießen wir die Rechnung in eine kleine Funktion. Erst brauchen wir aber das Profil selbst als etwas, das wir befragen können — eine Funktion, die zu jedem Ort die Temperatur zurückgibt. Wir haben die 30-Minuten-Momentaufnahme aus Kapitel 1 dafür in die Funktion `temperatur` gegossen. Wie sie im Inneren rechnet, ist hier **Nebensache** — die Formel dahinter begründen Kapitel 6 und Kapitel 12; für uns ist sie einfach ein Fühler, den wir an jede Stelle des Trägers halten können.

5. Steigung und Fläche

```
import math


def temperatur(ort):
    """Temperatur des Trägers am gegebenen Ort, 30 Minuten nach dem Ausbruch.

    ort: Stelle auf dem Träger, 0 (kühles Auflager) bis 1 (Brandherd), in Metern.
    Rueckgabe: Temperatur an dieser Stelle in Grad Celsius.

    Das ist die Momentaufnahme aus Kapitel 1 als glatte Kurve. Woher die
    Formel kommt, entwickeln die Kapitel 6 und 12; hier ist sie nur unser
    Messobjekt.
    """
    leitfaehigkeit = 50.0 / (7850.0 * 460.0)
    zeit = 30.0 * 60.0
    wert = 20.0 + 480.0 * ort
    for n in range(1, 6):
        vorzeichen = (-1.0) ** n
        beitrag = 960.0 * vorzeichen / (n * math.pi)
        abklingen = math.exp(-leitfaehigkeit * (n * math.pi) ** 2 * zeit)
        wert = wert + beitrag * math.sin(n * math.pi * ort) * abklingen
    return wert


def steigung_zwischen(ort_links, ort_rechts):
    """Steigung der Sekante durch zwei Punkte des Profils (in Grad C pro Meter).

    ort_links, ort_rechts: die beiden Orte in Metern.
    Rueckgabe: Temperaturunterschied geteilt durch Ortsunterschied.
    """
    delta_temperatur = temperatur(ort_rechts) - temperatur(ort_links)
    delta_ort = ort_rechts - ort_links
    return delta_temperatur / delta_ort


print("T(0,8 m) = %.2f °C" % temperatur(0.8))
print("T(1,0 m) = %.2f °C" % temperatur(1.0))
print("Sekantensteigung 0,8 → 1,0 m: %.1f °C/m" % steigung_zwischen(0.8, 1.0))
```

```
T(0,8 m) = 197.77 °C
T(1,0 m) = 500.00 °C
Sekantensteigung 0,8 → 1,0 m: 1511.2 °C/m
```

Interpretation der Ausgabe: dieselben Zahlen wie in der Handrechnung — 197,77 °C, 500,00 °C und daraus die Steigung 1511,2 °C/m. Die Funktion `steigung_zwischen` tut nichts anderes als das Steigungsdreieck: oben der Unterschied, unten der Abstand, dann teilen. Sie ist unser Werkzeug für das nächste Experiment.

5.2. Das Steigungsdreieck schrumpfen lassen

Die Sekante über zwanzig Zentimeter hat einen Schönheitsfehler: Sie mittelt über ein ganzes Stück Kurve. Links von $1{,}0$ m ist die Kurve flacher, ganz rechts am Brandherd steiler; die Sekante mischt beides. Wir wollten aber wissen, wie steil es **genau an der Stelle** $0{,}8$ m ist — nicht im Mittel über zwanzig Zentimeter.

Der Ausweg ist so naheliegend wie folgenreich: Wir lassen den rechten Punkt **näher heranrücken**. Statt bis $1{,}0$ gehen wir nur bis $0{,}9$, dann bis $0{,}81$, dann bis $0{,}801$. Der Abstand — wir nennen ihn ab jetzt h — wird immer kleiner, und die Sekante schmiegt sich immer enger an die Kurve bei $0{,}8$ m an.

> Vorhersage-Punkt
>
> **Bevor du die Tabelle liest:** Gleich rechnet das Programm die Sekantensteigung bei $x = 0{,}8$ m für immer kleinere Abstände $h = 0{,}2/0{,}1/0{,}01/0{,}001$ m aus. Was glaubst du: Werden die vier Zahlen wild durcheinanderspringen, oder laufen sie auf einen festen Wert zu? Und wenn sie zulaufen — von oben (immer kleiner werdend) oder von unten? Leg dich fest.

```
def steigungstabelle(ort, abstaende):
    """Druckt die Sekantensteigung an einem Ort fuer mehrere Abstaende h.

    ort: die feste Stelle, an der wir die Steigung wissen wollen (in Metern).
    abstaende: Liste der Abstaende h zum rechten Nachbarpunkt.
    Rueckgabe: nichts; die Funktion druckt eine Tabelle.
    """
    print("   h (m)  | T(x)      | T(x+h)    | Steigung (°C/m)")
    print("  --------+-----------+-----------+----------------")
    for h in abstaende:
        steigung = steigung_zwischen(ort, ort + h)
        print("  %6.3f  | %8.3f  | %8.3f  | %13.4f"
              % (h, temperatur(ort), temperatur(ort + h), steigung))
```

```
steigungstabelle(0.8, [0.2, 0.1, 0.01, 0.001])
```

```
   h (m)   | T(x)      | T(x+h)    | Steigung (°C/m)
  ---------+-----------+-----------+----------------
   0.200   |  197.770  |  500.000  |     1511.1501
   0.100   |  197.770  |  334.037  |     1362.6659
   0.010   |  197.770  |  209.485  |     1171.5196
   0.001   |  197.770  |  198.921  |     1150.8614
```

Interpretation der Ausgabe: Die vier Steigungen springen nicht, sie **laufen zu** — von oben: $1511{,}2 \to 1362{,}7 \to 1171{,}5 \to 1150{,}9$ °C/m. Je kleiner h, desto weniger ändert sich die Zahl noch. Sie nähert sich einem Grenzwert von rund 1149 °C/m. Sie kommen von oben, weil die Kurve nach rechts zum Brandherd hin steiler wird: Ein weiter entfernter rechter Punkt liegt schon im steileren Teil und zieht die Sekante nach oben. Rückt er heran, verschwindet dieser Effekt.

Diesen Zielwert, dem die Sekantensteigungen zulaufen, nennt man die **Ableitung** des Profils an der Stelle $0{,}8$ m. Er ist die Steigung der Kurve *genau dort* — die Steilheit an einem einzigen Punkt. Weil er aus dem Differenzenquotienten $\Delta T/\Delta x$ entsteht, wenn man Δx „zu Ende schrumpfen" lässt, schreibt man ihn mit einem kleinen d statt des großen Δ:

$$\left.\frac{\mathrm{d}T}{\mathrm{d}x}\right|_{x=0{,}8\,\mathrm{m}} \approx 1149\ \frac{^\circ\mathrm{C}}{\mathrm{m}}.$$

Lies $\mathrm{d}T/\mathrm{d}x$ als „$\Delta T/\Delta x$, zu Ende gedacht": dasselbe Steigungsdreieck, nur so klein geschrumpft, dass es die Kurve genau an einer Stelle berührt. Das kleine d ist ein Δ, das man unendlich klein gemacht hat. Mehr steckt hinter dem berühmten Symbol nicht — kein Zauber, nur ein zu Ende gedachtes Dreieck. Die **Einheit bleibt dieselbe** wie bei der Sekante: °C pro Meter. Das ist wichtig, denn diese Steilheit ist es, die den Wärmestrom antreibt (Kapitel 6): Wo $\mathrm{d}T/\mathrm{d}x$ groß ist, fließt viel Wärme.

Die Sekante, die sich beim Schrumpfen an die Kurve anschmiegt, hat einen eigenen Namen: Im Grenzfall berührt sie die Kurve nur noch in dem einen Punkt, statt sie zu schneiden — dann heißt sie **Tangente** (von lateinisch *tangere*, berühren). Die Ableitung ist die Steigung dieser Tangente. Abbildung 5.3 zeigt die Verwandlung in drei Bildern.

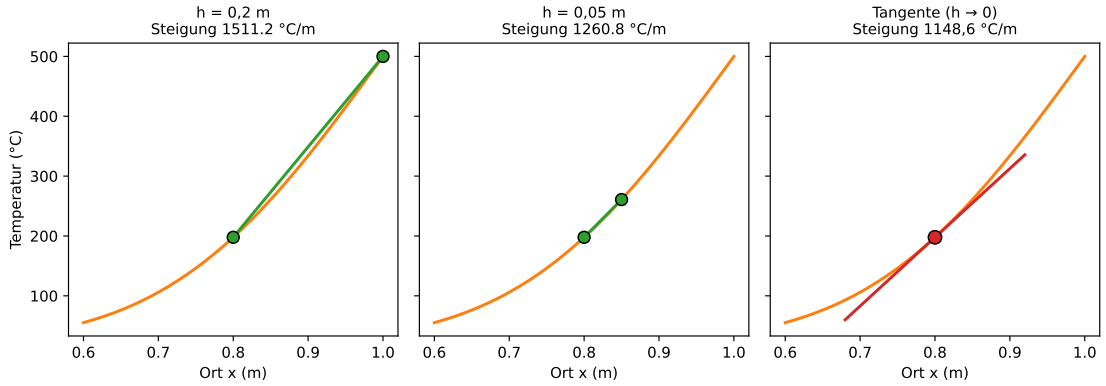

Abbildung 5.3.: Von der Sekante zur Tangente. Links ein großes h = 0,2 m: die Sekante (grün) schneidet die Kurve weit auseinander, Steigung 1511,2 °C/m. Mitte ein kleines h = 0,05 m: die Sekante liegt schon enger an, 1260,8 °C/m. Rechts das „zu Ende gedachte“ h: die Tangente berührt die Kurve nur noch im Punkt 0,8 m, Steigung 1148,6 °C/m — die Ableitung.

Was man hier sieht: Wie die grüne Sekante beim Schrumpfen des Abstands h in die rote Tangente kippt. Von links nach rechts wird die Gerade immer flacher und legt sich immer enger an die Kurve — bis sie sie im letzten Bild nur noch berührt. Der Steigungswert unter jedem Bild wandert dabei von 1511,2 über 1260,8 auf 1148,6 °C/m: genau die Zahlen aus unserer Tabelle.

Diese Drehung von der Sekante in die Tangente ist die eigentliche Bewegung hinter dem Wort „Ableitung”. Die folgende Darstellung (Animation in der HTML-Fassung, Bildreihe im Druck) zeigt sie als durchgehenden Vorgang, mit mitlaufendem Steigungswert.

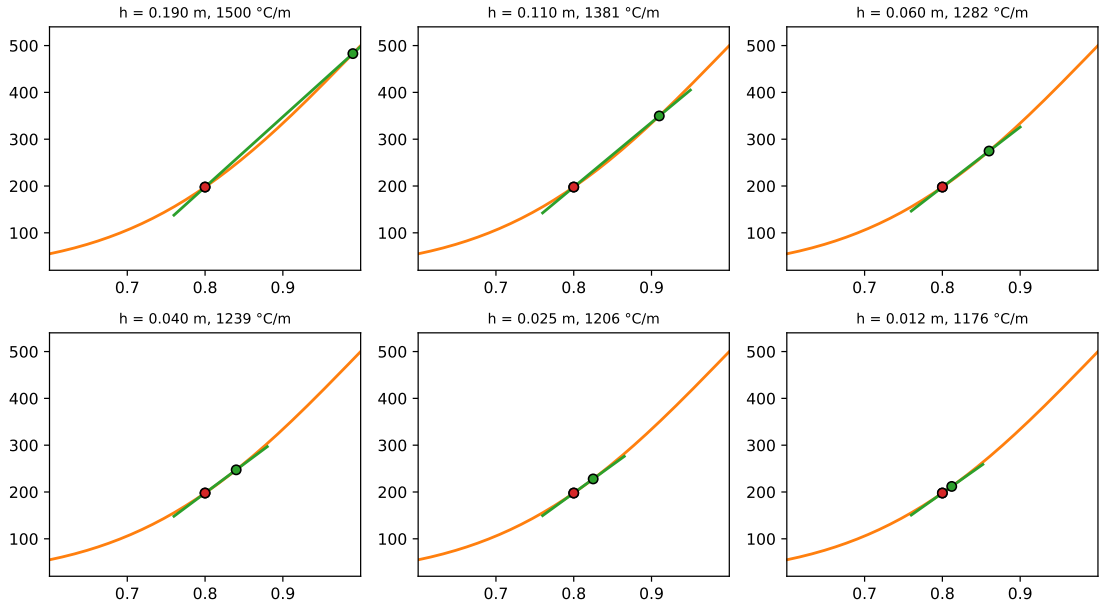

Abbildung 5.4.: Die Sekante dreht sich in die Tangente (in der HTML-Fassung eine Animation): sechs Momentaufnahmen mit schrumpfendem h. Von h = 0,19 m bis h = 0,012 m kippt die grüne Sekante immer flacher, und ihr Steigungswert sinkt von rund 1500 auf rund 1176 °C/m — dem Wert am Punkt selbst zu (rund 1149 °C/m).

Was man sieht: ein und dieselbe Bewegung, ob als Film oder als Bildreihe. Der rechte Punkt wandert heran, die Sekante kippt flacher, der Steigungswert sinkt und kommt bei rund 1149 °C/m zur Ruhe. „Ableiten" ist genau diese Bewegung — mehr nicht.

5.3. Der Sonderfall, um den sich alles dreht: die Gerade

Jetzt kommt die frohe Botschaft, die dem ganzen Kapitel den Titel gibt. Bei unserer krummen Kurve mussten wir h schrumpfen lassen, weil die Steigung von Ort zu Ort *anders* ist — flach über der kalten Mitte, steil am Brandherd. Aber schau, was bei einer **Geraden** passiert.

Denk dir das Feuer über viele Stunden ununterbrochen weiterbrennen, bis sich nichts mehr ändert: Dann ist das Profil kein krummer Verlauf mehr, sondern eine perfekte Gerade vom kühlen Auflager (20 °C) zum Brandherd

(500 °C). Ihre Steigung ist überall dieselbe. Egal, ob du das Steigungsdreieck über den ganzen Meter legst oder über einen Millimeter — du bekommst immer

$$\frac{\Delta T}{\Delta x} = \frac{500 - 20}{1{,}0} = 480\ \frac{^\circ\mathrm{C}}{\mathrm{m}}.$$

Bei einer Geraden ist die Sekante über jedes Stück schon die Tangente. Es gibt **nichts zu schrumpfen**, keinen Grenzwert zu bilden, keine Zahlen, die zulaufen — die Steigung steht mit einem einzigen Dreieck exakt fest. $\Delta T/\Delta x$ und $\mathrm{d}T/\mathrm{d}x$ sind hier dasselbe.

> Merksatz
>
> **Unsere FEM rechnet fast nur mit Geraden.** Zwischen zwei Knoten nähern wir jedes Temperaturprofil durch ein gerades Stück an (Kapitel 7). Und für gerade Stücke ist die Steigung Schulgeometrie: Höhe des Dreiecks durch seine Breite, $\Delta T/\Delta x$, exakt und ohne Grenzwert. Der ganze Schrumpf-Apparat aus diesem Abschnitt wird nur *einmal* gebraucht — um zu verstehen, was „Steigung an einem Punkt" überhaupt heißt. Beim eigentlichen Rechnen bleibt das Dreieck.

Das ist kein Taschenspielertrick, sondern die tragende Idee der Methode: Statt die krumme Kurve überall exakt abzuleiten, zerlegen wir sie in kurze gerade Stücke und leiten *die* ab — was für Geraden trivial ist. Je kürzer die Stücke (je feiner das Netz), desto besser trifft die Stückelung die echte Kurve. Genau diesen Handel — Krummes durch viele Gerade ersetzen — hast du in Kapitel 1 schon am Regler „NETZ" gesehen.

Zweite Frage: Wie viel?

5.4. Die Fläche unter der Kurve

Die zweite Frage des Kapitels klingt zunächst ganz anders als die erste, führt aber zu demselben Werkzeugkasten. Sie lautet: **Wie viel Wärme steckt insgesamt im Träger?**

Warum hat das mit einer *Fläche* zu tun? Denk an die Liste aus Kapitel 2: An jedem Knoten sitzt eine Temperatur. Die gespeicherte Wärme eines kleinen Trägerstücks ist umso größer, je heißer es ist und je länger das Stück. Heiß mal lang — Höhe mal Breite — ist aber genau eine **Fläche** unter dem Temperaturprofil. Ein hoher, breiter Abschnitt der Kurve steht für viel Wärme, ein

niedriger, schmaler für wenig. Die gesamte Wärme ist die gesamte Fläche unter der Kurve.

Nur: Wie misst man die Fläche unter einer *krummen* Kurve? Ein Rechteck kann man (Länge mal Breite), ein Dreieck auch — aber diese gebogene Fläche hat keine Schulformel. Der Ausweg ist derselbe Gedanke wie beim Netz: **Zerlege das Schwierige in viele einfache Stücke.** Wir stellen unter die Kurve lauter schmale **Rechteckstreifen**, deren Fläche wir einzeln ausrechnen können, und addieren sie.

Vorhersage-Punkt

Bevor du weiterliest: Gleich legen wir 4, dann 8, dann 64 Rechteckstreifen unter die Kurve. Jeder Streifen ist so hoch wie die Kurve an seinem linken Rand. Weil die Kurve ansteigt, ist jeder Streifen ein bisschen zu niedrig — die Streifensumme wird die wahre Fläche also **unterschätzen**. Was glaubst du: Kommt man mit 64 Streifen schon nah an den wahren Wert (rund 105 °C·m), oder bleibt ein großer Fehler? Leg dich fest.

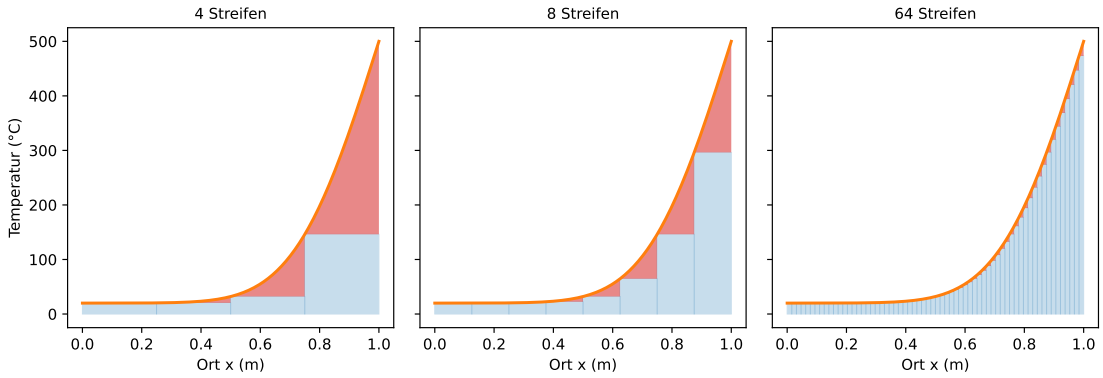

Abbildung 5.5.: Dieselbe Fläche unter dem Profil, mit 4, 8 und 64 Rechteckstreifen angenähert. Jeder Streifen ist so hoch wie die Kurve an seinem linken Rand. Der rot schraffierte Rest ist das, was die Streifen übersehen — die kleinen Dreiecke zwischen Streifenoberkante und Kurve, gebündelt dort, wo die Kurve zum Brandherd hin steil wird. Mit mehr Streifen schrumpft dieser Rest sichtbar zusammen.

Was man hier sieht: Mit 4 Streifen sind die roten Fehlerdreiecke am Brandherd noch dick — die Treppe passt schlecht unter die steile Kurve. Mit 8 Streifen ist der rote Rest schon halbiert, mit 64 Streifen kaum noch zu se-

hen: Die Treppe schmiegt sich fast an die Kurve. Mehr Streifen heißt weniger übersehene Fläche. Setzen wir Zahlen dahinter.

```
def flaeche_mit_streifen(anzahl):
    """Naehert die Flaeche unter dem Profil mit Rechteckstreifen an.

    anzahl: wie viele gleich breite Streifen wir von 0 bis 1 Meter legen.
    Rueckgabe: die aufsummierte Streifenflaeche (in Grad C mal Meter).

    Jeder Streifen ist so hoch wie die Kurve an seinem linken Rand; seine
    Flaeche ist Hoehe mal Breite. Wir summieren sie der Reihe nach auf -- die
    laufende Summe aus Kapitel 2.
    """
    breite = 1.0 / anzahl
    summe = 0.0
    for i in range(anzahl):
        ort_links = i * breite
        hoehe = temperatur(ort_links)
        summe = summe + hoehe * breite
    return summe


print(" Streifen | Fläche (°C·m) | Fehler zur wahren Fläche")
print(" ---------+---------------+-------------------------")
for anzahl in [4, 8, 64]:
    flaeche = flaeche_mit_streifen(anzahl)
    print("   %4d   |    %8.2f   |    %7.2f"
          % (anzahl, flaeche, 105.51 - flaeche))
```

```
 Streifen | Fläche (°C·m) | Fehler zur wahren Fläche
 ---------+---------------+-------------------------
      4   |       54.65   |      50.86
      8   |       77.75   |      27.76
     64   |      101.79   |       3.72
```

Interpretation der Ausgabe: Die Streifensummen kriechen von unten an die wahre Fläche heran: $54{,}6 \to 77{,}8 \to 101{,}8$ °C · m. Mit 4 Streifen fehlen noch fast 51 °C·m, mit 8 nur noch gut halb so viel, mit 64 nur noch $3{,}7$ °C · m — rund dreieinhalb Prozent. Jede Verdopplung der Streifenzahl halbiert grob den Fehler. Genau wie beim Schrumpfen von h läuft eine Reihe von Näherungen auf einen festen Wert zu; diesmal ist es die **Fläche**, der wahre Wert liegt bei rund $105{,}5$ °C · m.

Diesen Grenzwert — die Fläche, der die immer feineren Streifensummen zulaufen — nennt man das **Integral** des Profils. Und wie die Ableitung ein zugeschrumpftes Δ bekam, bekommt das Integral ein eigenes Zeichen für die zugeschrumpfte Summe:

$$\int_0^{1\,\mathrm{m}} T(x)\,\mathrm{d}x \;\approx\; 105{,}5\ °\mathrm{C}\cdot\mathrm{m}.$$

Das langgezogene $\int$ ist nichts als ein in die Länge gezogenes **S** für „Summe” — die Summe der Streifen, wenn man sie unendlich schmal werden lässt. Das $\mathrm{d}x$ dahinter ist die **Breite** eines solchen unendlich schmalen Streifens (dasselbe $\mathrm{d}x$ wie in der Ableitung, wieder ein zugeschrumpftes Δx). Und $T(x)$ ist seine **Höhe**. Zusammengelesen heißt $\int_0^{1\,\mathrm{m}} T(x)\,\mathrm{d}x$ also wörtlich:

> „Summiere über alle Orte von 0 bis 1 Meter die Höhe $T(x)$ mal die schmale Breite $\mathrm{d}x$.”

Es ist die `for`-Schleife aus `flaeche_mit_streifen`, nur auf Papier geschrieben und mit unendlich vielen, unendlich schmalen Streifen — so wie das Summenzeichen Σ aus Kapitel 2 die Schleife des Aufsummierens war. Auch die **Einheit** liest man direkt ab: Höhe mal Breite, also °C mal Meter, $°\mathrm{C}\cdot\mathrm{m}$.

Die folgende Darstellung zeigt das Zulaufen als Bewegung (in der HTML-Fassung animiert): Die Streifen werden schmaler, und die rote Restfläche schmilzt weg.

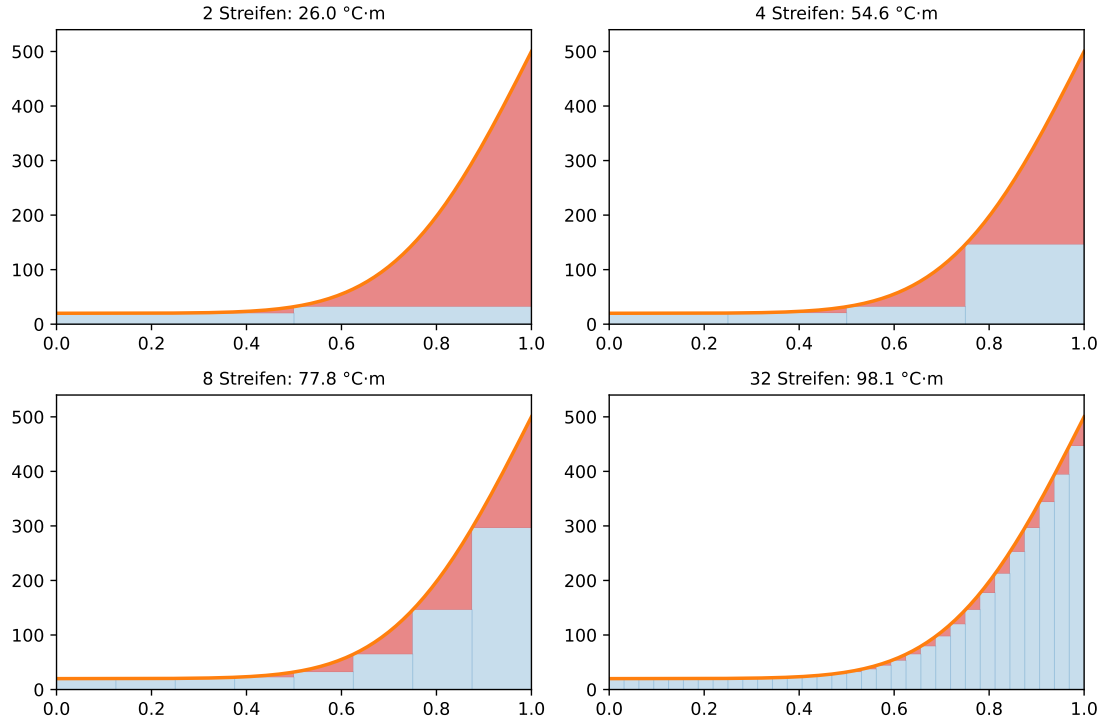

Abbildung 5.6.: Die schmelzende Restfläche (in der HTML-Fassung eine Animation): 2, 4, 8 und 32 Streifen. Von Bild zu Bild füllt die blaue Treppe die Fläche vollständiger, der rote Rest schrumpft, und die Streifensumme klettert an die wahre Fläche von 105,5 °C·m heran.

Was man sieht: Der rote Rest ist der Fehler, und er schmilzt mit jeder Verfeinerung. Bei 64 Streifen ist er fast weg. Das ist dieselbe Geschichte wie beim Netz aus Kapitel 1: Feiner zerlegen bringt näher an die Wirklichkeit — nur geht es hier nicht um die Form des Trägers, sondern um die Fläche unter seinem Temperaturprofil.

Von der Fläche zur Wärmemenge

Die Fläche $\int T\,\mathrm{d}x$ hat die krumme Einheit °C·m. Zur echten **Wärmemenge** in Joule wird sie durch die Materialdaten des Trägers. Man zählt dazu die Fläche über der 20-°C-Linie (die *zusätzliche* Wärme, die der Brand gebracht hat):

$$\int_0^{1\,\mathrm{m}} \big(T(x) - 20\,^\circ\mathrm{C}\big)\,\mathrm{d}x \;\approx\; 85{,}5\ ^\circ\mathrm{C}\cdot\mathrm{m}.$$

Diese Fläche multipliziert man mit der Dichte ρ, der Wärmekapazität c und der Querschnittsfläche A des Trägers. Für Stahl sind das $\rho = 7850\ \mathrm{kg/m^3}$ und $c = 460\ \mathrm{J/(kg\,K)}$, zusammen $\rho c = 3{,}611\ \mathrm{MJ/(m^3\,K)}$; bei einem quadratischen Querschnitt von $0{,}1\ \mathrm{m} \times 0{,}1\ \mathrm{m}$ ist $A = 0{,}01\ \mathrm{m^2}$. Verfolge die Einheiten Schritt für Schritt — sie kürzen sich sauber zu Joule:

$$\underbrace{\mathrm{°C \cdot m}}_{\text{Fläche}} \cdot \underbrace{\frac{\mathrm{kg}}{\mathrm{m^3}}}_{\rho} \cdot \underbrace{\frac{\mathrm{J}}{\mathrm{kg \cdot K}}}_{c} \cdot \underbrace{\mathrm{m^2}}_{A} = \mathrm{J}$$

(die Grad-Differenz °C zählt als Kelvin K und kürzt sich weg). Mit Zahlen:

$$Q = \rho\, c\, A \int (T - 20)\,\mathrm{d}x = 7850 \cdot 460 \cdot 0{,}01 \cdot 85{,}5 \approx 3{,}1\ \mathrm{MJ}.$$

Gut drei Millionen Joule zusätzliche Wärme stecken nach einer halben Stunde im Träger — herausgelesen allein aus der Fläche unter der Kurve. Genau solche Flächen wird Kapitel 7 für jedes einzelne Element ausrechnen.

5.5. Wieder die Gerade: das Trapez ist exakt

Auch bei der Fläche kommt die Erleichterung, sobald das Profil ein **gerades Stück** ist. Unter einer Geraden brauchen wir keine Streifen zu stapeln und keinen Grenzwert zu bilden — die Fläche unter einer Geraden ist ein **Trapez**, und dessen Fläche kennt die Schulgeometrie exakt:

$$\text{Trapezfläche} = \frac{T_{\text{links}} + T_{\text{rechts}}}{2} \cdot \text{Breite}.$$

In Worten: die **mittlere Höhe** der beiden Ränder, mal die Breite. Nimm ein gerades Stück Profil, das über eine Breite von $0{,}25\ \mathrm{m}$ von $100\ °\mathrm{C}$ auf $200\ °\mathrm{C}$ steigt. Seine Fläche ist

$$\frac{100 + 200}{2} \cdot 0{,}25 = 150 \cdot 0{,}25 = 37{,}5\ °\mathrm{C \cdot m}$$

— eine einzige Multiplikation, kein Streifenstapel, kein Fehler. Abbildung 5.7 stellt beide Wege nebeneinander.

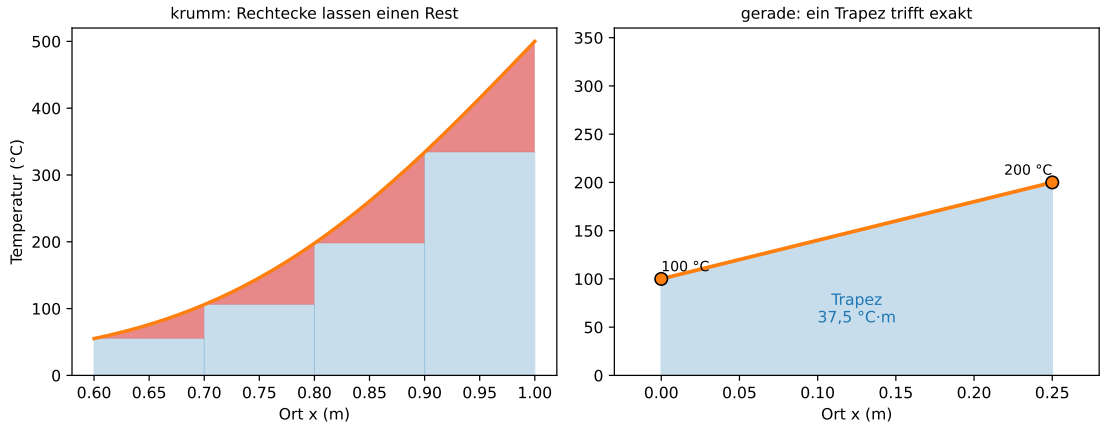

Abbildung 5.7.: Rechteckstreifen gegen Trapez. Links unter der krummen Kurve: Rechtecke lassen immer einen roten Rest, den erst unendlich viele Streifen schließen. Rechts unter einem geraden Stück: Ein einziges Trapez (mittlere Höhe mal Breite) trifft die Fläche exakt — kein Rest, kein Grenzwert. Deshalb ist die Gerade für die FEM Gold wert.

Was man hier sieht: links der mühsame Weg (viele Streifen, immer ein roter Rest), rechts der bequeme (ein Trapez, kein Rest). Sobald das Profil gerade ist, verschwindet die ganze Streifen-Schrumpferei — die Fläche steht mit einer Multiplikation fest. Das ist der zweite Grund, warum unsere FEM so gern mit Geraden rechnet: Nicht nur die Steigung, auch die Fläche wird dann Schulgeometrie.

5.6. Alles nur Geraden — der Brückenschlag zu Kapitel 7

Fügen wir beide Erkenntnisse zusammen und werfen einen ersten Blick auf das, was Kapitel 7 im Großen tun wird. Dort wird das Temperaturprofil nicht als eine krumme Kurve behandelt, sondern als **Streckenzug** aus geraden Stücken zwischen den Knoten — als wäre die Kurve mit dem Lineal von Knoten zu Knoten nachgezogen. Abbildung 5.8 zeigt so ein Profil, hütchenförmig (es steigt und fällt wieder, ein Vorgriff auf die „Hütchenfunktionen” aus Kapitel 7).

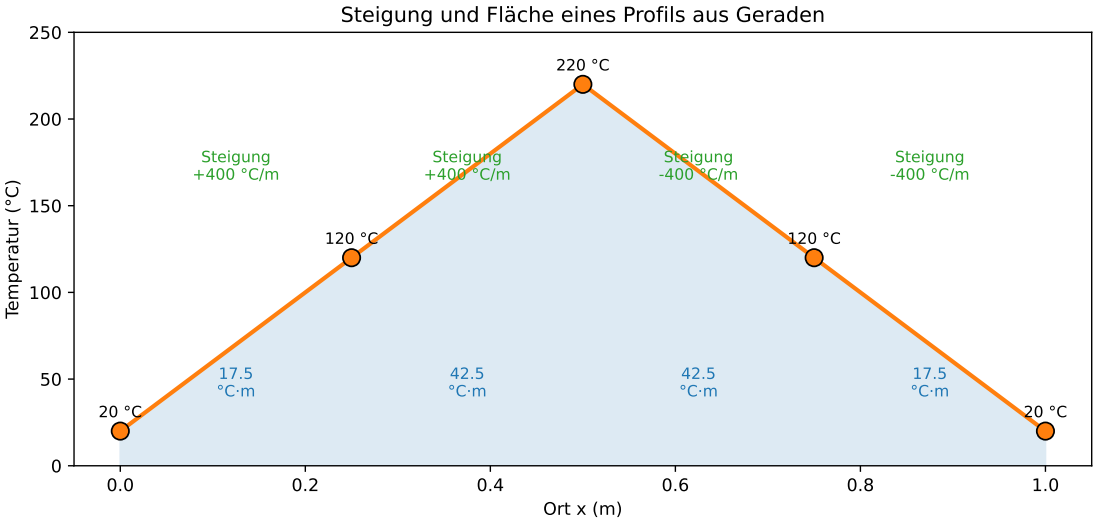

Abbildung 5.8.: Ein Profil aus lauter Geraden (Vorgriff auf Kapitel 7). Fünf Knoten bei 0 / 0,25 / 0,5 / 0,75 / 1 m, dazwischen gerade Stücke. Über jedem Abschnitt steht seine exakte Steigung (grün, ΔT/Δx) und seine exakte Trapezfläche (blau). Beides ist Schulgeometrie — kein Grenzwert nötig, weil jedes Stück eine Gerade ist.

Was man hier sieht: vier gerade Stücke, und für jedes ist beides sofort ablesbar. Die **Steigungen** sind $+400, +400, -400, -400$ °C/m — das Hütchen steigt mit 400 °C/m an und fällt mit 400 °C/m wieder ab, jede Steigung ein bloßes $\Delta T/\Delta x$. Die **Trapezflächen** sind $17{,}5$, $42{,}5$, $42{,}5$, $17{,}5$ °C · m; zusammen macht das 120 °C · m unter dem ganzen Hütchen. Keine Sekante wurde geschrumpft, kein Streifen gestapelt — jede Zahl ist Dreieck oder Trapez. **Das ist die ganze Analysis, die die FEM dieses Buches braucht.** In Kapitel 7 bekommt jedes dieser Geradenstücke seinen Namen (Element) und seine Steigung ihren Platz im Gleichungssystem.

Steigung und Fläche gehören zusammen

Eine Beobachtung zum Schluss, die uns später noch dient. Steigung und Fläche sind keine zwei fremden Dinge — sie sind **Umkehrungen** voneinander. Schau auf das Hütchen: Wenn du die Steigung $+400$ °C/m des ersten Stücks mit seiner Breite $0{,}25$ m multiplizierst, bekommst du $+100$ °C — genau den Temperaturanstieg von 20 auf 120 °C über dieses Stück zurück. Die **Fläche unter dem Steigungs-Verlauf** baut den **Temperaturunterschied** wieder auf. Ableiten und Integrieren machen einander rückgängig: Das eine misst, wie schnell sich etwas ändert, das

andere summiert die Änderungen wieder zusammen. Diese Verwandtschaft trägt einen großen Namen (den „Hauptsatz”), den wir hier nicht brauchen und nicht beweisen — für uns genügt die Beobachtung am Bild.

5.7. Die interaktive Einheit: das Steigungs- und Flächenlabor

Jetzt bist du dran. Das folgende Programm ist das **Steigungs- und Flächenlabor**: ein festes Profil (unsere 30-Minuten-Kurve), an dem du beide Werkzeuge dieses Kapitels selbst bedienst. Du drehst an zwei Reglern — dem Sekantenabstand `H_SEKANTE` und der Streifenzahl `STREIFENZAHL` — und siehst, wie sich Steigung und Fläche verändern. Der Rechenkern ist genau die Tabelle aus den beiden Abschnitten oben, nur automatisch: eine kurze Schleife für die Fläche, eine Zeile für die Steigung.

Vorhersage-Punkt

Bevor du ausführst: Der Regler `H_SEKANTE` steht auf $0{,}001$ m. Halbiere ihn im Kopf auf $0{,}0005$ m. Ändert sich die angezeigte Steigung dann noch um mehr als ein paar °C/m, oder steht der Vorkommawert praktisch fest? Und andersherum: Wenn du von $0{,}2$ auf $0{,}1$ m halbierst — springt die Steigung da nur um ein paar °C/m oder um über hundert? Leg dich für beide fest, dann probiere es.

```
#| autorun: true
# Steigungs- und Flaechenlabor. Dreh an den beiden Reglern und sieh zu, wie
# sich Steigung (an der Stelle 0,8 m) und Flaeche (unter dem ganzen Profil)
# aendern.
H_SEKANTE = 0.001        # Abstand zum rechten Nachbarpunkt (Meter)
STREIFENZAHL = 8         # wie viele Rechteckstreifen die Flaeche annaehern
STELLE = 0.8             # an welcher Stelle wir die Steigung messen (Meter)

import math


def temperatur(ort):
    """Temperatur des Trägers am Ort (Meter), 30 Minuten nach dem Ausbruch."""
    leitfaehigkeit = 50.0 / (7850.0 * 460.0)
    zeit = 30.0 * 60.0
```

```
    wert = 20.0 + 480.0 * ort
    for n in range(1, 6):
        vorzeichen = (-1.0) ** n
        beitrag = 960.0 * vorzeichen / (n * math.pi)
        abklingen = math.exp(-leitfaehigkeit * (n * math.pi) ** 2 * zeit)
        wert = wert + beitrag * math.sin(n * math.pi * ort) * abklingen
    return wert


def steigung_bei(stelle, h):
    """Sekantensteigung an einer Stelle, mit Abstand h nach rechts (Grad C/m)."""
    return (temperatur(stelle + h) - temperatur(stelle)) / h


def flaeche_mit_streifen(anzahl):
    """Rechteck-Naeherung der Flaeche unter dem Profil (Grad C mal Meter)."""
    breite = 1.0 / anzahl
    summe = 0.0
    for i in range(anzahl):
        summe = summe + temperatur(i * breite) * breite
    return summe


steigung = steigung_bei(STELLE, H_SEKANTE)
flaeche = flaeche_mit_streifen(STREIFENZAHL)
print("Steigung bei %.2f m (h = %.4f m): %8.3f °C/m" % (STELLE, H_SEKANTE, steigung))
print("Fläche mit %d Streifen:            %8.3f °C·m" % (STREIFENZAHL, flaeche))
print("(wahre Steigung ≈ 1149 °C/m, wahre Fläche ≈ 105,5 °C·m)")

# --- Abbildung ---
import matplotlib.pyplot as plt
orte = []
werte = []
schritte = 200
for i in range(schritte + 1):
    ort = i / schritte
    orte.append(ort)
    werte.append(temperatur(ort))

fig, ax = plt.subplots(figsize=(8.0, 3.4))
breite = 1.0 / STREIFENZAHL
for i in range(STREIFENZAHL):
    xl = i * breite
```

```
    ax.add_patch(plt.Rectangle((xl, 0), breite, temperatur(xl),
                               facecolor="tab:blue", alpha=0.22,
                               edgecolor="tab:blue", lw=0.5))
ax.plot(orte, werte, color="tab:orange", lw=2.2)
ax.plot([STELLE, STELLE + H_SEKANTE],
        [temperatur(STELLE), temperatur(STELLE + H_SEKANTE)],
        "o-", color="tab:green", lw=2.0, ms=7, markeredgecolor="black")
ax.set_xlabel("Ort x (m)")
ax.set_ylabel("Temperatur (°C)")
ax.set_title("Steigung (grün) und Fläche (blau) im Labor")
plt.tight_layout()
plt.show()
```

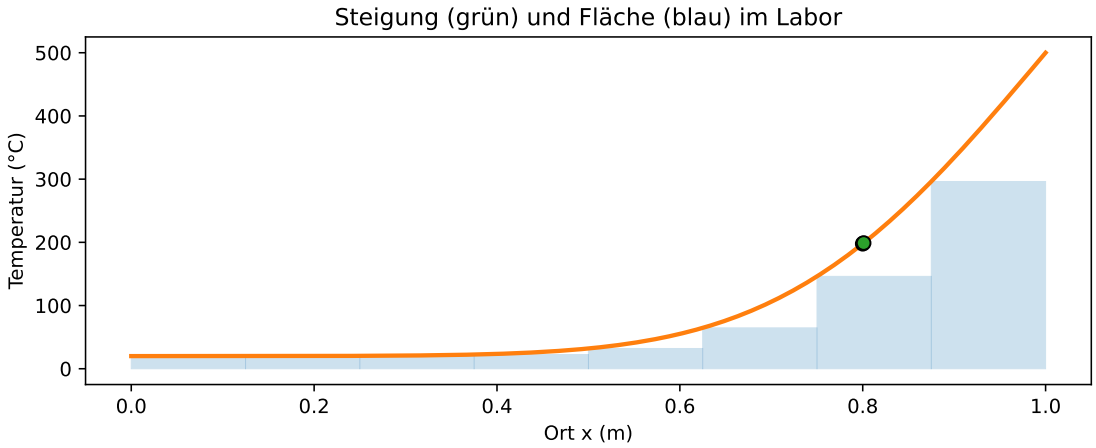

Abbildung 5.9.: Vorgerenderte Fassung des Steigungs- und Flächenlabors (Sekantenabstand 0,001 m, 8 Streifen). Die grüne Sekante misst die Steigung bei 0,8 m, die blaue Treppe nähert die Fläche an. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst.

Was man sieht: Mit `H_SEKANTE = 0,001` zeigt das Labor eine Steigung von $1150{,}9$ °C/m — schon dicht an der wahren $1148{,}6$. Halbierst du h auf $0{,}0005$, wandert die Anzeige nur noch um gut ein Grad (auf $1149{,}7$); der Vorkommawert steht längst fest. Von $0{,}2$ auf $0{,}1$ dagegen sprang sie noch um fast 150 °C/m. Das ist das „Zulaufen" aus dem ersten Teil, jetzt zum Selbstdrehen. Bei der Fläche gilt dasselbe: Verdopple `STREIFENZAHL`, und der blaue Treppenrand schmiegt sich enger an die Kurve, die Zahl kriecht näher an 105,5 °C·m.

5.8. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript zu diesem Kapitel liegt in `programme/kap05/kap05_steigung_und_flaeche.py`. Es ist das Labor als Kommandozeilen-Werkzeug: Es druckt die volle Schrumpf-Tabelle der Sekantensteigung (von $h = 0{,}2$ bis $0{,}0001$ m), die Streifen-Tabelle der Fläche (von 2 bis 512 Streifen) und rechnet die gespeicherte Wärme in Joule aus. Zum Schluss speichert es eine Abbildung des Profils mit eingezeichneter Sekante und Streifen als PNG. Sein Kern sind genau die Funktionen dieses Kapitels — `temperatur`, `steigung_zwischen`, `flaeche_mit_streifen` —, keine Zeile mehr.

Merkkasten

- Die **Steigung** eines Profils an einer Stelle ist $\Delta T/\Delta x$ — Temperaturunterschied durch Ortsunterschied, in °C/m (das **Steigungsdreieck**).
- Lässt man den Abstand h auf null **schrumpfen**, läuft die Sekantensteigung auf einen Wert zu: die **Ableitung** $\mathrm{d}T/\mathrm{d}x$ — „Δ, zu Ende gedacht". Sie treibt den Wärmestrom (Kapitel 6).
- Die **Fläche** unter dem Profil nähert man mit **Rechteckstreifen** an; mehr Streifen, kleinerer Fehler. Der Grenzwert ist das **Integral** $\int T\,\mathrm{d}x$ — ein langgezogenes S für „Summe unendlich schmaler Streifen", in °C·m. Sie misst die gespeicherte Wärme (Kapitel 7).
- Für **Geraden** ist beides Schulgeometrie und **exakt**: Steigung = Dreieck ($\Delta T/\Delta x$), Fläche = Trapez (mittlere Höhe mal Breite). Kein Grenzwert nötig.
- **Unsere FEM rechnet fast nur mit Geraden** — darum ist genau das die ganze Analysis, die dieses Buch braucht.

Roter Faden

Zurück: Die Tabellen dieses Kapitels sind `for`-Schleifen aus Kapitel 2 — die **laufende Summe** der Streifenflächen ist dasselbe Aufaddieren, das dort den Mittelwert bildete, und das mathematische Summenzeichen Σ und das Integralzeichen $\int$ sind beide nur diese Schleife auf Papier. Die Idee „Krummes durch viele einfache Stücke ersetzen" kennst du schon vom **Netz** aus Kapitel 1.

Vor: Die **Steigung** $\mathrm{d}T/\mathrm{d}x$ wird in Kapitel 6 zum Motor des Wärmestroms (Fouriers Gesetz: viel Gefälle, viel Fluss). Die **Trapez- und Dreiecksflächen** unter den Geradenstücken werden in Kapitel 7 zu den Elementinte-

gralen der Hütchenfunktionen — die vier Steigungen des Hütchens aus Abbildung 5.8 landen dort im Gleichungssystem. Und die **Streifenidee** kehrt in 1 wieder, wenn wir Flächenanteile in Dreiecken aufteilen.

Übungen

Ü 5.1 (Verstehen). Ein gerades Stück Temperaturprofil steigt über eine Breite von $0{,}4$ m von 60 °C auf 260 °C. Rechne **ohne Rechner** aus: (a) die Steigung dieses Stücks in °C/m, (b) die Fläche unter ihm in °C·m. Kontrolliere danach im Labor, indem du dir ein entsprechend steiles gerades Profil vorstellst.

> **i** Musterlösung zu Ü 5.1
>
> (a) Steigung $= \Delta T / \Delta x = (260 - 60)/0{,}4 = 200/0{,}4 = 500$ °C/m. Weil das Stück gerade ist, ist das exakt — kein Schrumpfen nötig.
>
> (b) Trapezfläche = mittlere Höhe mal Breite $= \frac{60+260}{2} \cdot 0{,}4 = 160 \cdot 0{,}4 = 64$ °C · m. Auch das ist exakt, weil die Fläche unter einer Geraden ein Trapez ist.

Ü 5.2 (Verändern). Stelle im Labor ein **steileres** Erlebnis her, indem du im Kopf durchspielst, was passiert, wenn der Brand heißer wäre (das Profil also überall höher liegt und zum Brandherd hin steiler ansteigt). **Sage zuerst vorher:** Wird sich dann eher die *Steigung* bei 0,8 m stark ändern oder eher die *Fläche* — oder beides? Verändere dann `STREIFENZAHL` von 8 auf 64 und prüfe, ob die Fläche näher an ihren wahren Wert rückt.

> **i** Musterlösung zu Ü 5.2
>
> Ein heißerer Brand hebt das Profil überall an **und** macht es zum Brandherd hin steiler — darum wachsen **beide** Größen: Die Steigung bei 0,8 m wird größer (die Kurve steigt steiler zum Brandherd hin an), und die Fläche wird größer (mehr Höhe unter der ganzen Kurve, also mehr gespeicherte Wärme). Das Verfeinern von 8 auf 64 Streifen ändert am *wahren* Wert der Fläche nichts — es bringt die **Näherung** nur näher an ihn heran: Der Fehler schrumpft von rund 28 auf unter 4 °C·m. Mehr Streifen verbessern die Messung, nicht die Sache.

Ü 5.3 (Übertragen). Das ist die Übung, die Kapitel 7 unmittelbar braucht. Der Träger hat fünf Knoten mit den Temperaturen $[\,20,\ 20,\ 32,\ 146,\ 500\,]$ °C (aus unserer Kurve abgelesen und gerundet) an den Orten $[\,0,\ 0{,}25,\ 0{,}5,\ 0{,}75,\ 1{,}0\,]$ m.

Schreibe eine Funktion `elementsteigungen(temperaturen, abstand)`, die aus den fünf Knotentemperaturen die **vier Elementsteigungen** berechnet (je die Steigung des geraden Stücks zwischen zwei Nachbarknoten) und als Liste zurückgibt. Das vollständige Lösungsskript liegt in `loesungen/kap05_ue3.py`.

i Musterlösung zu Ü 5.3

```python
def elementsteigungen(temperaturen, abstand):
    """Die Steigung jedes geraden Stuecks zwischen zwei Nachbarknoten.

    temperaturen: Liste der Knotentemperaturen (Grad C).
    abstand: gleicher Knotenabstand h (Meter).
    Rueckgabe: Liste der Steigungen (Grad C/m), eine pro Element.
    """
    steigungen = []
    for i in range(len(temperaturen) - 1):
        delta_temperatur = temperaturen[i + 1] - temperaturen[i]
        steigungen.append(delta_temperatur / abstand)
    return steigungen


temperaturen = [20.0, 20.0, 32.0, 146.0, 500.0]
for steigung in elementsteigungen(temperaturen, 0.25):
    print("%.0f °C/m" % steigung)
```

Ausgabe: 0, 48, 456, 1416 °C/m. Jede Zahl ist ein bloßes $\Delta T / \Delta x$ — die Steigung eines geraden Stücks, exakt und ohne Grenzwert. Die Steigungen werden nach rechts hin dramatisch größer: Die ersten beiden Stücke über der kalten Mitte sind fast flach — die Mitte weiß noch nichts vom Feuer —, das letzte direkt am Brandherd ist mit über 1400 °C/m extrem steil. Genau diese vier Elementsteigungen füttert Kapitel 7 in das Gleichungssystem des Trägers; hier hast du sie mit reiner Schulgeometrie selbst erzeugt.

Das Kleingedruckte

Dieses Kapitel hat die Analysis bewusst **klein** gehalten — nur so viel, wie die zwei Fragen des Trägers verlangen. Was du hier absichtlich *nicht* gesehen hast: keine **Ableitungsregeln** (Produkt-, Ketten-, Quotientenregel), keine **Stammfunktionen** und Integraltabellen, keinen förmlichen **Grenzwert-Begriff** und keinen Beweis des „Hauptsatzes", der Ableiten und Integrieren

verbindet. All das gibt es, all das füllt dicke Bücher — und all das brauchen wir nicht, weil unsere Ansatzfunktionen Geraden sind und für Geraden Dreieck und Trapez genügen.

Zwei ehrliche Feinheiten noch. **Erstens:** Die glatte Kurve `temperatur`, an der wir gemessen haben, ist die *exakte* 30-Minuten-Momentaufnahme des Trägers (dieselbe Physik wie in Kapitel 1, nur nicht grob mit 26 Knoten gerechnet, sondern als geschlossene Formel). Woher diese Formel kommt, ist Stoff von Kapitel 6 und Kapitel 12; hier war sie nur ein Messobjekt, an dem wir Steigung und Fläche üben konnten. **Zweitens:** Die „wahren" Werte $1148{,}6\ °\mathrm{C/m}$ und $105{,}5\ °\mathrm{C \cdot m}$, denen unsere Näherungen zuliefen, haben wir mit sehr feinen Rechnungen bestimmt — im wirklichen Leben kennt man sie oft nicht vorab, und dann ist die Frage „fein genug?" eine Kunst für sich. Ihr ist Kapitel 12 gewidmet („Traue keinem bunten Bild").

6. Die Wärmeleitungsgleichung

Stell dir dein Girokonto vor. Sein Stand ändert sich nie aus dem Nichts — er ändert sich nur durch **Zugänge und Abgänge**: Gehalt kommt rein, Miete geht raus. Steht am Monatsende dieselbe Zahl da wie am Anfang, dann muss über den Monat genau so viel reingekommen sein, wie rausgegangen ist. Keine Buchung ist verschwunden; die Bilanz geht auf.

Ein Trägerstück ist ein solches Konto, nur dass darauf **Wärme** liegt. Wärme fließt vom Brandherd herein, Wärme fließt zum kühlen Auflager hin wieder hinaus. Und wenn sich am Träger nichts mehr ändert — wenn er *eingeschwungen* ist, wie es in Kapitel 1 hieß —, dann muss durch jedes noch so kleine Stück gleich viel Wärme hinein- wie herausfließen. Sonst würde sich das Stück ja aufheizen oder abkühlen, und genau das tut es im eingeschwungenen Zustand nicht mehr.

Diese schlichte Buchhaltung, aufgeschrieben für *jedes* Stück des Trägers gleichzeitig, ist die **Wärmeleitungsgleichung** — die erste Differentialgleichung dieses Buches. Sie sieht am Ende gelehrt aus, aber sie sagt nichts Komplizierteres als: „rein gleich raus, überall". Dieses Kapitel baut sie Stück für Stück aus dieser einen Idee auf. Und es beantwortet endlich die Frage aus dem allerersten Kapitel: Warum ist das eingeschwungene Temperaturprofil eine **Gerade** — und die Mitte exakt 160 °C?

Lernziele

Nach diesem Kapitel kannst du …

1. … **Fouriers Gesetz** $q = -\lambda \cdot \mathrm{d}T/\mathrm{d}x$ in Worten erklären (Wärme fließt bergab, das Gefälle treibt den Strom) und mit Einheiten zu einer Zahl in Watt ausrechnen,
2. … die **Energiebilanz** an einem Trägerstück aufstellen — rein minus raus ist im Gleichgewicht null,
3. … daraus die **stationäre Wärmeleitungsgleichung** gewinnen und jeden ihrer Bausteine physikalisch benennen,

4. … begründen, **warum** das eingeschwungene Profil im homogenen Träger eine Gerade sein muss (und mit dem Bilanz-Prüfer selbst nachweisen, dass nur die Gerade die Bilanz erfüllt),
5. … erklären, was eine **Differentialgleichung** von einer gewöhnlichen Gleichung unterscheidet: Gesucht ist eine ganze Funktion, keine einzelne Zahl.

⚠ Naheliegende Vermutung

Vermutung: *„Direkt neben dem Brandherd (300 °C) ist der Träger fast überall heiß — die Wärme staut sich dort, und erst ganz links wird es schlagartig kühl.“*
Warum sie naheliegt: Am heißen Ende steckt die Wärme ja hinein, und Wärme „braucht Zeit“, um sich zu verteilen. Es liegt nahe, sich einen heißen Klumpen am Brandherd vorzustellen, der nach links hin ausfranst.
Was stattdessen stimmt: Im eingeschwungenen Zustand ist das Profil eine **perfekte Gerade** von 20 °C bis 300 °C — die Mitte liegt exakt bei 160 °C, kein Stau, keine Beule. Der Grund ist die Bilanz: Jeder Stau würde an einer Stelle ein steileres Gefälle erzeugen, und ein steileres Gefälle treibt sofort mehr Wärme ab (Fouriers Gesetz), bis der Stau wieder verschwunden ist. Das rechnen wir in diesem Kapitel nach — und in Kapitel 12 messen wir es am fertigen Programm nach.

6.1. Fouriers Gesetz: das Gefälle treibt den Strom

Fang mit einer Alltagserfahrung an. Du hältst einen Metalllöffel in eine heiße Suppe. Nach einer Weile wird das Griffende warm — Wärme ist durch das Metall gewandert, von heiß (in der Suppe) nach kalt (in deiner Hand). Zwei Dinge weißt du aus Erfahrung, ohne je eine Formel gesehen zu haben:

- **Je größer der Temperaturunterschied, desto stärker der Wärmestrom.** Ein Löffel in kochender Suppe wird schneller heiß als einer in lauwarmem Tee.
- **Je nach Material fließt die Wärme leicht oder schwer.** Der Metalllöffel wird heiß; ein Holzlöffel im selben Topf bleibt am Griff angenehm kühl.

Genau diese beiden Erfahrungen fasst das **Fouriersche Gesetz** in eine Formel. Es ist das Naturgesetz hinter der wandernden Wärme aus Kapitel 1, und es sagt, wie stark die Wärme an einer Stelle fließt:

$$\underbrace{q}_{\text{Wärmestromdichte}} = -\lambda \cdot \underbrace{\frac{\mathrm{d}T}{\mathrm{d}x}}_{\text{Temperaturgefälle}} .$$

Lies das Stück für Stück. Das $\mathrm{d}T/\mathrm{d}x$ ist die **Steigung** des Temperaturprofils — genau die Ableitung, die wir in Kapitel 5 mit dem Steigungsdreieck erarbeitet haben: um wie viel Grad die Temperatur pro Meter fällt oder steigt. Das ist der erste Erfahrungssatz: Ein steiles Gefälle (große Steigung) treibt viel Wärme. Der Buchstabe λ (griechisch „Lambda") ist die **Wärmeleitfähigkeit** des Materials — das ist der zweite Erfahrungssatz: Metall leitet gut (großes λ), Holz schlecht (kleines λ). Und das Minuszeichen — dazu gleich mehr — sagt nur, in welche Richtung die Wärme fließt: bergab, vom Warmen zum Kalten.

Die Größe q links heißt **Wärmestromdichte**: die Wärme, die pro Sekunde durch einen Quadratmeter Querschnittsfläche strömt, gemessen in Watt pro Quadratmeter ($\mathrm{W/m^2}$). Ein Watt ist ein Joule pro Sekunde — es ist also ein *Strom*, ein Durchsatz, nicht eine gespeicherte Menge. Das ist der Unterschied zur Fläche unter der Kurve aus Kapitel 5: Die maß die *gespeicherte* Wärme (Joule); q misst den *fließenden* Nachschub (Joule pro Sekunde).

6.1.1. Jedes Material hat sein λ

Die Wärmeleitfähigkeit λ ist eine Materialeigenschaft, so wie die Dichte oder die Farbe. Man schlägt sie in einer Tabelle nach. Abbildung 6.1 stellt ein paar Werte nebeneinander — und die Spannweite ist gewaltig.

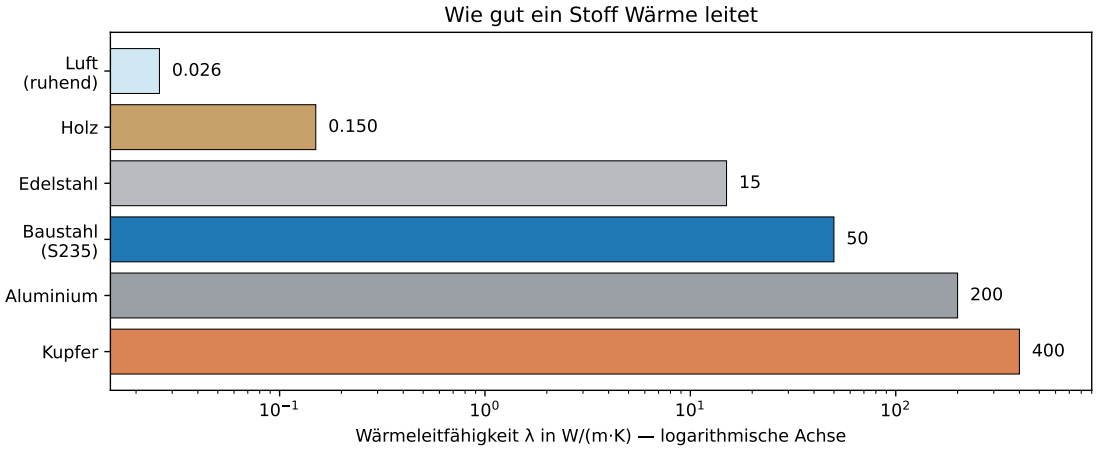

Abbildung 6.1.: Wärmeleitfähigkeit λ einiger Stoffe, in W/(m·K), auf einer logarithmischen Achse (jeder Schritt nach rechts ist Faktor zehn). Von Luft bis Kupfer liegen mehr als vier Zehnerpotenzen — Kupfer leitet Wärme rund fünfzehntausendmal besser als ruhende Luft. Unser Baustahl (50, hervorgehoben) ist ein guter Leiter im mittleren Feld; Aluminium leitet noch viermal besser, der Edelstahl der Übung 6.3 dagegen mehr als dreimal schlechter. Holz und Luft sind Dämmstoffe. Genau deshalb ist der Kochtopf aus Metall und sein Griff aus Holz.

Was man hier sieht: Die Achse ist logarithmisch — sonst würde man neben der 400 von Kupfer die 0,026 von Luft gar nicht mehr erkennen. Metalle (Edelstahl, Baustahl, Aluminium, Kupfer) leiten hundert- bis zehntausendmal besser als Holz oder Luft. Unser Träger aus **Baustahl** mit $\lambda = 50\ \mathrm{W/(m\,K)}$ sitzt im mittleren Feld: ein guter, aber kein überragender Wärmeleiter. Auffällig ist, wie weit die beiden Stähle auseinanderliegen — der rostfreie Edelstahl leitet mit etwa 15 mehr als dreimal schlechter als unser Baustahl; diesen Unterschied nutzen wir in Übung 6.3 aus. Dass ausgerechnet ruhende Luft so schlecht leitet, ist übrigens der ganze Trick jeder Dämmung — von der Daunenjacke bis zur Doppelverglasung: Sie sperren Luft in kleine Kammern und nutzen ihr winziges λ.

> **Die Einheit von λ, kurz erklärt**
>
> Warum $\mathrm{W/(m\,K)}$? Die Einheit fällt aus Fouriers Gesetz heraus. Links steht q in $\mathrm{W/m^2}$, rechts das Gefälle $\mathrm{d}T/\mathrm{d}x$ in Kelvin pro Meter (K/m). Damit die Gleichung aufgeht, muss λ die Einheit tragen, die aus K/m

wieder $\mathrm{W/m^2}$ macht:

$$\frac{\mathrm{W}}{\mathrm{m}^2} = \lambda \cdot \frac{\mathrm{K}}{\mathrm{m}} \quad \Longrightarrow \quad \lambda = \frac{\mathrm{W}}{\mathrm{m}^2} \cdot \frac{\mathrm{m}}{\mathrm{K}} = \frac{\mathrm{W}}{\mathrm{m\,K}}.$$

(Ein Temperatur*unterschied* in Grad Celsius ist derselbe Zahlenwert wie in Kelvin — eine Differenz von 10 °C ist eine Differenz von 10 K —, deshalb stehen im Gefälle wahlweise °C/m oder K/m.)

6.1.2. Das Minuszeichen heißt „bergab"

Über kein Zeichen dieses Gesetzes stolpern Anfänger so oft wie über das Minus. Es ist aber ganz harmlos: Es sorgt nur dafür, dass die Wärme in die **richtige Richtung** fließt. Abbildung 6.2 macht es sichtbar.

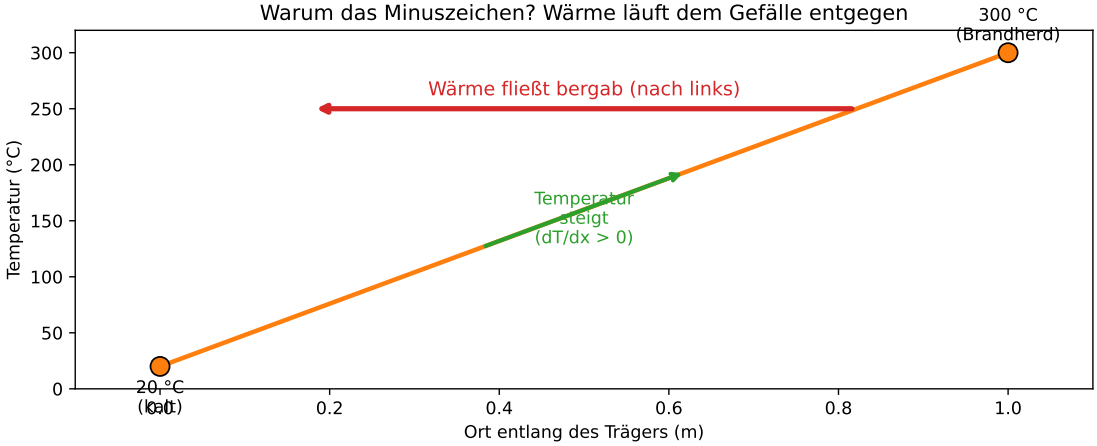

Abbildung 6.2.: Das Minuszeichen als Richtungspfeil. Das Temperaturprofil steigt nach rechts an (dorthin sitzt der Brandherd), die Steigung dT/dx ist also positiv. Der Wärmestrom aber zeigt nach links, vom Heißen zum Kalten — er ist dem Gefälle entgegengesetzt. Genau das leistet das Minuszeichen: Aus der positiven Steigung wird ein nach links gerichteter (negativer) Strom. Wärme fließt bergab, wie Wasser.

Was man hier sieht: Zwei Pfeile, die in verschiedene Richtungen zeigen. Der grüne folgt der Temperatur *bergauf* nach rechts zum Brandherd — die Steigung $\mathrm{d}T/\mathrm{d}x$ ist positiv. Der rote Wärmestrom aber läuft nach *links*, weg vom Brandherd, zum kühlen Auflager. Strom und Gefälle zeigen entgegengesetzt, und das Minuszeichen dreht genau das eine ins andere um: Aus der

positiven Steigung macht es einen nach links gerichteten Strom. Merke dir das Bild vom Wasser: Es fließt von selbst bergab, nie bergauf. So auch die Wärme.

6.1.3. Der Wärmestrom durch unseren Träger — eine anfassbare Zahl

Jetzt setzen wir zum ersten Mal echte Zahlen ein — das Standard-Handbeispiel dieses Buches im **eingeschwungenen** Zustand: links steckt der Träger im kühlen Auflager und wird dort bei $T_{\text{links}} = 20\ °\text{C}$ gehalten, rechts hält der Brandherd das Ende auf $T_{\text{rechts}} = 300\ °\text{C}$. Der Träger ist $L = 1\ \text{m}$ lang, aus Baustahl ($\lambda = 50\ \text{W}/(\text{m}\,\text{K})$), und sein Querschnitt ist quadratisch: **0,1 m hoch und 0,1 m breit**, also eine Fläche von

$$A \;=\; 0{,}1\ \text{m} \times 0{,}1\ \text{m} \;=\; 0{,}01\ \text{m}^2.$$

(Dasselbe A hat schon Kapitel 5 benutzt, als es aus der Fläche unter der Kurve die gespeicherte Wärme in Joule ausrechnete — der Träger bleibt derselbe.)

Weil das Profil eingeschwungen eine Gerade ist (das *beweisen* wir gleich, hier nehmen wir es vorweg), ist das Gefälle überall dasselbe. Wir bekommen es mit dem Steigungsdreieck aus Kapitel 5 — Höhe durch Breite, exakt und ohne Grenzwert:

$$\frac{\mathrm{d}T}{\mathrm{d}x} \;=\; \frac{T_{\text{rechts}} - T_{\text{links}}}{L} \;=\; \frac{300 - 20}{1{,}0} \;=\; 280\ \frac{\text{K}}{\text{m}}.$$

Die Temperatur fällt (von rechts gesehen) um 280 Grad je Meter. Fouriers Gesetz macht daraus die Wärmestromdichte — wir rechnen mit dem Betrag und merken uns die Richtung („nach links") getrennt:

$$q \;=\; \lambda \cdot \frac{\mathrm{d}T}{\mathrm{d}x} \;=\; 50 \cdot 280 \;=\; 14\,000\ \frac{\text{W}}{\text{m}^2}.$$

Das ist die Dichte — pro Quadratmeter, also 14 Kilowatt durch jeden Quadratmeter Querschnitt. Unser Träger ist aber viel dünner als ein Quadratmeter; durch seinen Querschnitt $A = 0{,}01\ \text{m}^2$ fließt entsprechend weniger:

$$I \;=\; q \cdot A \;=\; 14\,000 \cdot 0{,}01 \;=\; 140\ \text{W}.$$

140 Watt — das ist eine Zahl, die man anfassen kann. So viel Wärme strömt Sekunde für Sekunde durch den Träger, vom Brandherd zum kühlen Auflager. Zum Vergleich: So viel Leistung gibt eine helle, altmodische Glühbirne ab, oder ungefähr so viel Wärme, wie ein Mensch bei zügiger körperlicher Arbeit an seine Umgebung abgibt. Der ganze Träger schaufelt also ununterbrochen die Wärmeleistung eines arbeitenden Menschen vom heißen zum kühlen Ende weiter — und weil er eingeschwungen ist, kommt am Auflager exakt so viel an, wie am Brandherd hineingeht. Sonst würde er sich ja erwärmen.

Gießen wir Fouriers Gesetz in reines Python — zwei kurze Funktionen, die genau die Handrechnung nachvollziehen:

```
WAERMELEITFAEHIGKEIT = 50.0      # Baustahl S235, W/(m K)
QUERSCHNITT_M2 = 0.1 * 0.1       # 0,1 m x 0,1 m = 0,01 m^2


def waermestromdichte(gefaelle):
    """Fouriers Gesetz: Waermestromdichte q aus dem Temperaturgefaelle.

    gefaelle: Temperaturunterschied pro Meter (K/m), als Betrag bergab.
    Rueckgabe: Waerme pro Sekunde und Quadratmeter (W/m^2).
    """
    return WAERMELEITFAEHIGKEIT * gefaelle


def waermestrom(gefaelle, flaeche):
    """Gesamter Waermestrom durch eine Querschnittsflaeche (Watt).

    gefaelle: Temperaturgefaelle (K/m).
    flaeche: Querschnittsflaeche des Traegers (m^2).
    Rueckgabe: Waermestrom in Watt (Joule pro Sekunde).
    """
    return waermestromdichte(gefaelle) * flaeche


gefaelle = (300.0 - 20.0) / 1.0
print("Temperaturgefaelle: %.0f K/m" % gefaelle)
print("Waermestromdichte q: %.0f W/m^2" % waermestromdichte(gefaelle))
print("Waermestrom I durch den Traeger: %.0f W" % waermestrom(gefaelle, QUERSCHNITT_M2))
```

```
Temperaturgefaelle: 280 K/m
Waermestromdichte q: 14000 W/m^2
Waermestrom I durch den Traeger: 140 W
```

Interpretation der Ausgabe: dieselben drei Zahlen wie in der Handrechnung — Gefälle 280 K/m, Dichte 14 000 W/m², Strom 140 W. Die Funktion `waermestrom` tut nichts anderes, als das Gefälle mit λ und der Fläche zu multiplizieren; Fouriers Gesetz ist eine einzige Multiplikation. Merke dir diese 140 W — sie kehren gleich in der Bilanz wieder.

> **Das Auflager als Wärmesenke**
>
> Wir tun im ganzen Kapitel so, als bliebe das linke Ende des Trägers unerschütterlich bei 20 °C — egal, wie lange der Brand schon lodert. Ist das ehrlich? Ja, und zwar aus einem handfesten Grund. Das linke Ende steckt tief im **Auflager**, in der massiven, kühlen Außenwand aus Mauerwerk und Beton. Diese Wand ist ein gewaltiger Wärmespeicher: Tonnen von Material, die bei Raumtemperatur liegen und über eine große Fläche mit dem übrigen Bauwerk und der Außenluft verbunden sind.
> Die 140 W, die der Träger ankarrt, sind für eine solche Masse eine Lächerlichkeit — sie fließen in die Wand hinein und verlaufen sich darin, ohne sie merklich zu erwärmen; das Auflager schluckt diesen Strom mühelos. Deshalb dürfen wir das kühle Ende als **festgehaltene Temperatur** behandeln: 20 °C, komme, was wolle. Eine so vorgeschriebene Temperatur an einem Rand heißt später **Dirichlet-Randbedingung** (Kapitel 8) — und das Auflager ist ihr ehrlichstes Beispiel: eine **Wärmesenke**, die nimmt, ohne selbst warm zu werden. (Erst über sehr lange Zeit — Stunden, viele Brandlasten — würde auch das Auflager langsam anziehen; für unser eingeschwungenes Handbeispiel bleibt es kühl.)

6.2. Die Bilanz am Trägerstück

Jetzt kommt die eigentliche Idee des Kapitels: die **Buchhaltung**. Wir schneiden aus dem Träger ein Stück heraus und führen für dieses Stück Kontobuch über die Wärme — was hereinkommt, was hinausgeht.

Nimm dazu unseren Träger mit den fünf Knoten aus Kapitel 1 und Kapitel 5: an den Orten $0,\ 0{,}25,\ 0{,}5,\ 0{,}75,\ 1{,}0$ m, mit dem Abstand $h = 0{,}25$ m. Betrachte den mittleren Knoten (bei $0{,}5$ m) und schneide ein Stück um ihn herum frei — von seinem linken bis zu seinem rechten Nachbarn. Abbildung 6.3 zeigt diesen Freischnitt: das ist **die** Schlüsselgrafik dieses Kapitels.

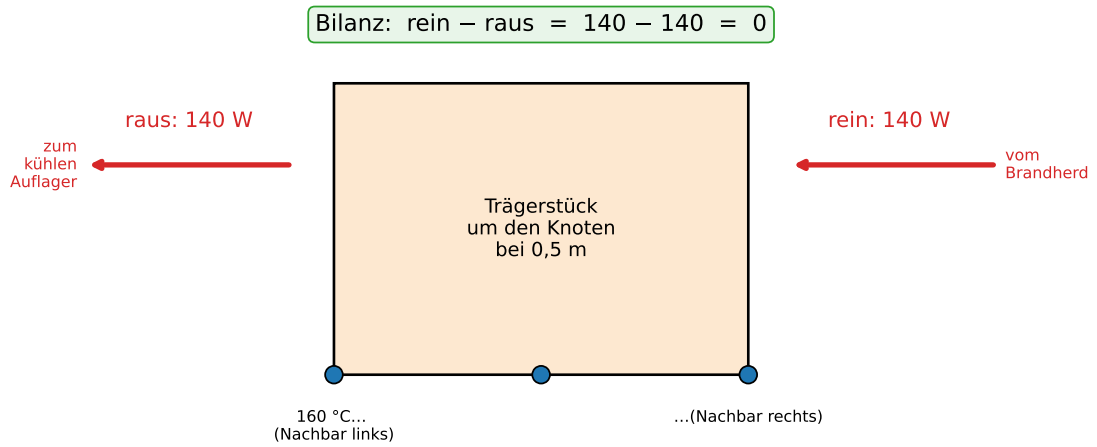

Abbildung 6.3.: Ein Trägerstück um den mittleren Knoten (0,5 m), freigeschnitten. Von rechts (der heißen Seite) strömt Wärme herein, nach links (zur kühlen Seite) strömt sie hinaus. Auf der eingeschwungenen Geraden ist beides gleich groß — 140 W rein, 140 W raus. Das Konto ist ausgeglichen: Der Knoten heizt sich nicht auf. Genau diese Bilanz, für jedes Stück gefordert, ist die Wärmeleitungsgleichung.

Was man hier sieht: ein Konto mit einer Buchung rein und einer raus. Von rechts, der heißen Seite, strömen 140 W in das Stück hinein; nach links, zur kühlen Seite, strömen 140 W wieder hinaus. Beide sind gleich groß, also ist die Bilanz null — das Stück sammelt keine Wärme an und verliert keine, seine Temperatur bleibt stehen. Das ist die Bedingung des eingeschwungenen Zustands, auf ein einzelnes Trägerstück heruntergebrochen.

Woher kommen die beiden 140 W? Aus Fouriers Gesetz, angewandt auf die beiden Ränder des Stücks. Der **hereinströmende** Strom ist der durch das rechte Element (zwischen dem Knoten und seinem rechten Nachbarn), der **hinausströmende** der durch das linke Element. Jeder ist λ mal Gefälle mal Fläche — und weil auf der Geraden das Gefälle überall dasselbe ist (280 K/m), sind beide Ströme gleich 140 W. Genau das prüfen wir gleich mit dem Programm für *beliebige* Profile nach.

Warum gerade der eingeschwungene Zustand?

Solange sich der Träger noch aufheizt (die transiente Phase aus Kapitel 1), stimmt die Bilanz *nicht*: Dann kommt an manchen Stücken mehr rein als raus, und die Differenz heizt das Stück auf — es lädt sein Konto

auf. Dieses Aufladen ist ein eigener Term (der **Speicherterm**), und ihn nehmen wir uns erst in Kapitel 9 vor („Jetzt mit Zeit"). In diesem Kapitel bleiben wir beim eingeschwungenen Zustand, wo sich nichts mehr ändert und die Bilanz sauber auf null aufgeht. Ein schweres Ding nach dem anderen.

6.2.1. Das Stück schrumpfen lassen

Bisher war unser Stück $0{,}5$ m breit (von 0,25 bis 0,75 m). Aber die Bilanz muss für *jedes* Stück gelten, egal wie breit — und egal, *wo* wir schneiden. Also machen wir dasselbe, was in Kapitel 5 aus der Sekante die Tangente gemacht hat: Wir lassen die Breite des Stücks **schrumpfen**. Wir denken das Steigungsdreieck zu Ende.

Nenn die Breite des Stücks Δx. Die Bilanz sagt: Der Wärmestrom, der links hineingeht, ist gleich dem, der rechts herauskommt —

$$\underbrace{q(x)}_{\text{Strom bei } x} \;=\; \underbrace{q(x+\Delta x)}_{\text{Strom bei } x+\Delta x}\,.$$

In Worten: Die Wärmestromdichte q hat am linken Rand des Stücks denselben Wert wie am rechten Rand. Das heißt aber nichts anderes, als dass sich q über das Stück **nicht ändert**. Und wenn das für jedes noch so schmale Stück an jeder Stelle gilt, dann ändert sich q nirgends — q ist entlang des ganzen Trägers **konstant**. Mit dem Steigungs-Werkzeug aus Kapitel 5 geschrieben: Die *Steigung* von q ist überall null,

$$\frac{\mathrm{d}q}{\mathrm{d}x} \;=\; 0.$$

Das ist die Bilanz „zu Ende gedacht" — dasselbe Muster wie beim Schrumpfen des Sekantenabstands, nur diesmal auf den Wärmestrom angewandt. Jetzt setzen wir noch Fouriers Gesetz ein, $q = -\lambda\,\mathrm{d}T/\mathrm{d}x$, und bekommen die **stationäre Wärmeleitungsgleichung**:

$$\boxed{\frac{\mathrm{d}}{\mathrm{d}x}\left(\lambda\,\frac{\mathrm{d}T}{\mathrm{d}x}\right) \;=\; 0}$$

(Das Minuszeichen aus Fouriers Gesetz fällt weg, weil null mal minus eins immer noch null ist.) Übersetzen wir jeden Baustein einzeln zurück ins Anschauliche:

- $\frac{\mathrm{d}T}{\mathrm{d}x}$ ist das **Temperaturgefälle** — wie steil das Profil an einer Stelle fällt (das Steigungsdreieck aus Kapitel 5).
- $\lambda \frac{\mathrm{d}T}{\mathrm{d}x}$ ist (bis aufs Vorzeichen) der **Wärmestrom** an dieser Stelle — Fouriers Gesetz.
- $\frac{\mathrm{d}}{\mathrm{d}x}(\dots)$ fragt, **wie sich dieser Strom entlang des Trägers ändert** — die Steigung des Stroms.
- $= 0$ verlangt, dass sich der Strom **nirgends ändert** — genau die Bilanz „rein gleich raus" an jedem Punkt.

Die ganze Gleichung sagt also wörtlich: *„Der Wärmestrom ist überall gleich."* Mehr steckt nicht dahinter. Sie sieht nur deshalb ehrfurchtgebietend aus, weil zwei Steigungen ineinander verschachtelt sind — erst die des Profils (macht den Strom), dann die des Stroms (die null sein soll).

Was ist hier eigentlich gesucht?

Halt kurz inne und vergleiche mit den Gleichungen aus Kapitel 3 und 2. Dort war die Lösung eine **Zahl** (oder eine Handvoll Zahlen: die Knotenwerte). Hier ist die Lösung eine **ganze Funktion** $T(x)$ — eine Temperatur an *jedem* Ort des Trägers, unendlich viele Werte auf einmal. Genau das macht den Unterschied zwischen einer gewöhnlichen Gleichung und einer **Differentialgleichung**. Eine gewöhnliche Gleichung wie $2u = 6$ sucht eine Zahl ($u = 3$). Eine Differentialgleichung stellt eine Bedingung an die *Steigungen* einer gesuchten Funktion und fragt: **Welche Funktion erfüllt diese Bedingung überall?** Das Wort verrät es: „Differential" von der Ableitung (dem geschrumpften Δ), „Gleichung" wie immer. Es ist eine Gleichung, in der Ableitungen vorkommen — und gesucht ist die Funktion, die sie wahr macht.

6.3. Lösen durch Probieren: die Gerade

Wie löst man eine Differentialgleichung? Es gibt ganze Bücher voller Methoden dafür — und wir brauchen keine einzige davon. Für unsere Gleichung genügt **Probieren**: Wir raten eine Funktion, setzen sie ein und schauen, ob die Bilanz aufgeht. Das ist erlaubt und sogar ehrlich — wenn eine geratene Funktion die Gleichung *und* die Randwerte erfüllt, ist sie die Lösung.

Unsere Vermutung liegt nach allem, was wir gesehen haben, auf der Hand: eine **Gerade**. Probieren wir eine Gerade von 20 °C links auf 300 °C rechts. Für eine Gerade ist das Gefälle $\mathrm{d}T/\mathrm{d}x$ überall dasselbe (280 K/m, Kapitel 5).

Dann ist auch der Wärmestrom $\lambda\,\mathrm{d}T/\mathrm{d}x$ überall derselbe (140 W durch den Querschnitt). Und wenn der Strom überall derselbe ist, dann ändert er sich nirgends —

$$\frac{\mathrm{d}}{\mathrm{d}x}(\text{konstanter Strom}) \;=\; 0.$$

Die Bilanz geht auf, an jeder Stelle. **Die Gerade erfüllt die Wärmeleitungsgleichung.** Und die beiden Randwerte — 20 °C erzwungen links, 300 °C erzwungen rechts — legen fest, *welche* Gerade es ist: die eine, die genau durch diese beiden Punkte läuft. Damit ist die Lösung eindeutig.

Jetzt können wir die Vermutung vom Kapitelanfang endgültig widerlegen.

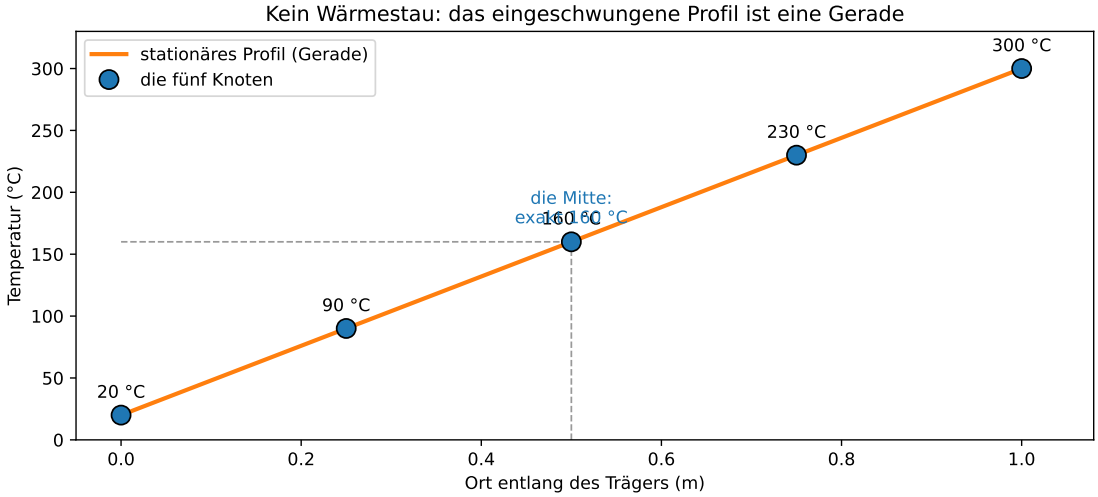

Abbildung 6.4.: Die eingeschwungene Lösung: eine Gerade von 20 °C auf 300 °C. Kein Wärmestau am Brandherd, keine Beule — die Mitte liegt exakt bei 160 °C, genau zwischen kalt und heiß. Bei einem Viertel der Länge sind es 90 °C, bei drei Vierteln 230 °C. Das ist die Antwort auf die Frage aus Kapitel 1, wie warm der Träger in der Mitte ist.

Was man hier sieht: die widerlegte Vermutung. Kein heißer Klumpen am Brandherd, keine Beule — eine schnurgerade Linie von 20 °C bis 300 °C. Die Mitte liegt exakt bei $\frac{20+300}{2} = 160$ °C, genau in der Mitte zwischen den beiden Enden. Der Grund ist die Bilanz: Ein Wärmestau würde die Kurve an einer Stelle nach oben beulen, dort das Gefälle links und rechts verschieden machen — und dann wäre der Strom links anders als rechts, die Bilanz verletzt. Der einzige Verlauf, bei dem der Strom überall gleich ist, ist die

Gerade. So beantwortet die Buchhaltung die Frage aus Kapitel 1, wie warm der Träger in der Mitte ist: **160 °C, kein Zufall.**

6.4. Die interaktive Einheit: der Bilanz-Prüfer

Bisher hast du uns geglaubt, dass nur die Gerade die Bilanz erfüllt. Jetzt prüfst du es selbst nach. Das folgende Programm ist der **Bilanz-Prüfer**: Du gibst ihm ein Kandidaten-Profil vor — eine Gerade, einen Knick oder eine Parabel —, und es rechnet an jedem inneren Knoten aus, wie viel Wärme hereinkommt und wie viel hinausgeht. Es druckt eine Tabelle `Knoten | rein | raus | Bilanz`. Ein Profil *besteht* die Prüfung nur, wenn die Bilanz an **jedem** inneren Knoten null ist.

Der Rechenkern ist genau die Bilanz von oben, in reines Python gegossen: Der Strom durch ein Element ist λ mal Gefälle (aus Kapitel 5: Differenz durch Abstand) mal Fläche. „Rein" ist der Strom durch das rechte Element (von der heißen Seite), „raus" der durch das linke Element (zur kühlen Seite).

> Vorhersage-Punkt
>
> **Bevor du ausführst:** Drei Kandidaten treten an — eine **Gerade**, ein **Knick** (zwei gerade Stücke mit einem scharfen Bug in der Mitte) und eine nach oben gewölbte **Parabel**. Alle drei starten bei 20 °C links und enden bei 300 °C rechts; sie unterscheiden sich nur dazwischen. Welche besteht die Bilanzprüfung an *allen* inneren Knoten? Nur die Gerade? Auch der Knick? Auch die Parabel? Leg dich fest, dann führe die Zelle aus (sie startet mit der Geraden — ändere `PROFIL` auf `knick` oder `parabel`).

```
#| autorun: true
# Der Bilanz-Pruefer. Waehle unten ein Profil (PROFIL = gerade / knick /
# parabel) und sieh nach, ob an jedem inneren Knoten gleich viel Waerme
# reinkommt wie rausgeht. Nur dann ist die Waermeleitungsgleichung erfuellt.
WAERMELEITFAEHIGKEIT = 50.0      # Baustahl S235, W/(m K)
QUERSCHNITT_M2 = 0.01            # 0,1 m x 0,1 m
KNOTENABSTAND = 0.25             # Abstand h zwischen zwei Knoten (Meter)


def gerade():
    """Gleichmaessig ansteigende Gerade von 20 auf 300 Grad."""
    return [20.0, 90.0, 160.0, 230.0, 300.0]
```

```
def knick():
    """Zwei gerade Stuecke mit einem scharfen Knick in der Mitte."""
    return [20.0, 60.0, 100.0, 200.0, 300.0]


def parabel():
    """Nach oben gewoelbtes Profil (wie mit einem Heizdraht im Traeger)."""
    return [20.0, 120.0, 200.0, 260.0, 300.0]


def strom_durch_element(temperatur_links, temperatur_rechts):
    """Waermestrom (Watt) durch ein gerades Stueck zwischen zwei Knoten.

    Betrag in Flussrichtung: lambda mal Gefaelle mal Querschnittsflaeche.
    Das Gefaelle ist die Steigung aus Kapitel 5 -- Differenz durch Abstand.
    """
    gefaelle = (temperatur_rechts - temperatur_links) / KNOTENABSTAND
    return WAERMELEITFAEHIGKEIT * gefaelle * QUERSCHNITT_M2


def pruefe_bilanz(temperaturen):
    """Prueft an jedem inneren Knoten die Bilanz rein - raus und druckt sie.

    temperaturen: Liste der Knotentemperaturen (Grad C).
    Rueckgabe: Liste der Bilanzwerte an den inneren Knoten (Watt).

    rein  = Strom durch das rechte Element (von der heissen Seite herein),
    raus  = Strom durch das linke Element  (zur kuehlen Seite hinaus).
    Bilanz = rein - raus; null heisst: der Knoten heizt sich nicht auf.
    """
    print("Knoten |   rein   |   raus   |  Bilanz")
    print("-------+----------+----------+---------")
    bilanzen = []
    for i in range(1, len(temperaturen) - 1):
        rein = strom_durch_element(temperaturen[i], temperaturen[i + 1])
        raus = strom_durch_element(temperaturen[i - 1], temperaturen[i])
        bilanz = rein - raus
        bilanzen.append(bilanz)
        print("   %d   | %7.1f  | %7.1f  | %7.1f" % (i + 1, rein, raus, bilanz))
    return bilanzen


PROFIL = gerade                 # <-- hier gerade / knick / parabel eintragen
```

```
temperaturen = PROFIL()
print("Profil:", PROFIL.__name__, "->", temperaturen)
print()
bilanzen = pruefe_bilanz(temperaturen)
besteht = True
for b in bilanzen:
    if abs(b) > 0.001:
        besteht = False
print()
if besteht:
    print("BESTANDEN: rein = raus an jedem Knoten. Die Bilanz geht auf.")
else:
    print("DURCHGEFALLEN: an mindestens einem Knoten kommt mehr rein als raus.")

# --- Abbildung ---
import matplotlib.pyplot as plt
orte = [0.0, 0.25, 0.5, 0.75, 1.0]
fig, ax = plt.subplots(figsize=(8.5, 3.6))
ax.plot(orte, temperaturen, "-", color="tab:orange", lw=2.2, zorder=2)
# innere Knoten gruen (Bilanz ok) oder rot (verletzt) faerben
farben_innen = []
for b in bilanzen:
    if abs(b) < 0.001:
        farben_innen.append("tab:green")
    else:
        farben_innen.append("tab:red")
ax.plot(orte[0], temperaturen[0], "o", color="0.5", ms=11,
        markeredgecolor="black", zorder=3)
ax.plot(orte[-1], temperaturen[-1], "o", color="0.5", ms=11,
        markeredgecolor="black", zorder=3)
for k in range(1, 4):
    ax.plot(orte[k], temperaturen[k], "o", color=farben_innen[k - 1], ms=13,
            markeredgecolor="black", zorder=4)
    ax.annotate("%+.0f W" % bilanzen[k - 1], xy=(orte[k], temperaturen[k]),
                xytext=(0, 12), textcoords="offset points", ha="center",
                fontsize=10, color=farben_innen[k - 1])
ax.set_xlabel("Ort entlang des Trägers (m)")
ax.set_ylabel("Temperatur (°C)")
ax.set_title("Bilanz-Prüfer: grüner Knoten = ausgeglichen, roter = verletzt")
plt.tight_layout()
plt.show()
```

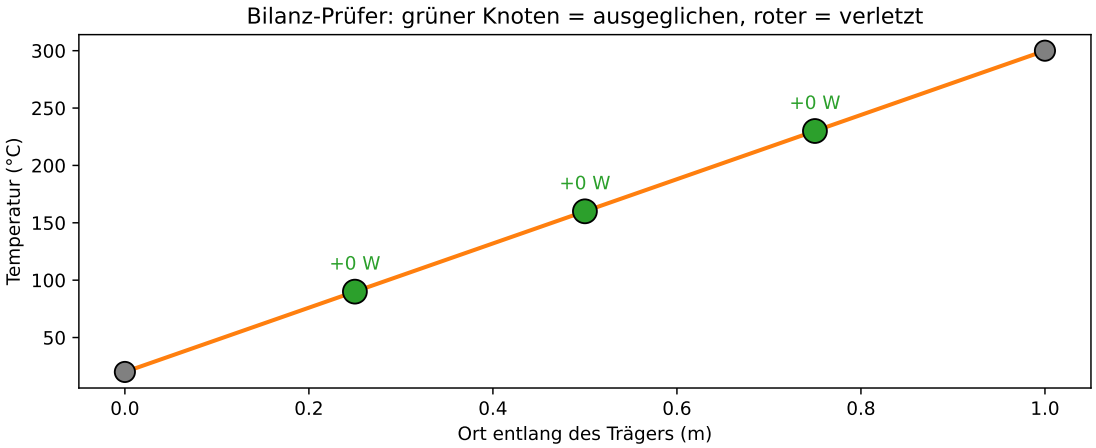

Abbildung 6.5.: Vorgerenderte Fassung des Bilanz-Prüfers mit der Geraden: alle drei inneren Knoten sind grün, die Bilanz ist überall 0 W. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst — trag `PROFIL = knick` oder `PROFIL = parabel` ein und sieh die Knoten rot werden.

Was man sieht: Mit der **Geraden** ist die Bilanz an jedem inneren Knoten genau null — 140 W kommen rein, 140 W gehen raus, überall. Alle drei Knoten leuchten grün, das Profil besteht. Trägst du `PROFIL = knick` ein, wird der mittlere Knoten rot: Dort, wo die Linie knickt, kommen plötzlich 200 W rein, aber nur 80 W gehen raus — eine Bilanz von $+120$ W, die den Knoten aufheizen würde. Bei der **Parabel** sind *alle* inneren Knoten rot, jeder mit einer Bilanz von -40 W: An jeder Stelle strömt mehr hinaus, als hereinkommt — der Fehlbetrag müsste von irgendwoher kommen. Nur die Gerade hält der Prüfung an allen Knoten stand. Du hast damit selbst nachgewiesen, was oben behauptet wurde — bevor irgendwer es bewiesen hat.

Wärmequelle im Träger — der Heizdraht

Warum fällt die Parabel an *jedem* inneren Knoten mit demselben Wert durch? Weil sie die Lösung eines *anderen* Problems ist. Ziehen wir einen **Heizdraht** durch den Träger, der überall gleichmäßig Wärme *erzeugt* (statt sie nur durchzuleiten), dann darf die Bilanz nicht mehr aufgehen: An jedem Stück muss genau so viel mehr hinausströmen, wie der Draht dort einspeist. Die rechte Seite der Gleichung ist dann nicht null, sondern die eingespeiste Leistung:

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\lambda\,\frac{\mathrm{d}T}{\mathrm{d}x}\right) \;=\; -\,\dot{q}_{\text{Quelle}}.$$

Und die Lösung ist dann keine Gerade mehr, sondern genau eine **Parabel** — die Kurve wölbt sich **nach oben**, liegt also überall über der geraden Verbindung von 20 auf 300 °C, denn jeder Punkt bekommt zusätzlich zur durchgeleiteten noch die selbst erzeugte Wärme. In der Mitte ist die Aufwölbung am größten: Unser Parabelprofil steht dort bei 200 °C, ganze 40 Grad über den 160 °C der Geraden. Und die -40 W, mit denen jeder innere Knoten durchfällt, sind genau die Leistung, die der Draht dort einspeist: An jedem Knoten strömen 40 W mehr hinaus, als hereinkommen, und diese 40 W liefert der Draht. Rechnet man das auf den ganzen Meter hoch — drei innere Knoten, je 40 W, dazu die Randanteile —, speist der Draht rund $160\ \mathrm{W/m}$ ein. Diese Wärmequelle bauen wir in Kapitel 7 mit der FEM und stellen sie in Kapitel 12 auf den Prüfstand — hier ist sie nur ein Ausblick.

Dieselbe Gleichung, ganz woanders: das Grundwasser

Die Bilanzidee ist nicht auf Wärme beschränkt. Strömt Grundwasser durch Sand, gilt das **Darcysche Gesetz** — Wort für Wort dasselbe wie Fouriers Gesetz, nur mit anderen Namen: Der Wasserfluss ist proportional zum Gefälle der Wasserhöhe, $q = -K\,\mathrm{d}h/\mathrm{d}x$, mit der Durchlässigkeit K an der Stelle von λ und der Wasserhöhe h an der Stelle der Temperatur T. Und dieselbe Bilanz („rein gleich raus”) führt auf dieselbe Gleichung. Wer die Wärmeleitung im Träger verstanden hat, versteht die stationäre Grundwasserströmung gratis mit — und Dutzende weitere Vorgänge, von der elektrischen Leitung bis zur Diffusion. Das ist die stille Macht dieser einen Gleichung.

6.5. Die Bilanz als Bewegung

Bisher war die Bilanz ein Standbild. Aber „rein gleich raus” ist im Kern etwas **Fließendes**. Die folgende Darstellung (Animation in der HTML-Fassung) zeigt den Wärmestrom als Strom kleiner Teilchen, die durch das Trägerstück wandern — erst im Gleichgewicht, dann gestört.

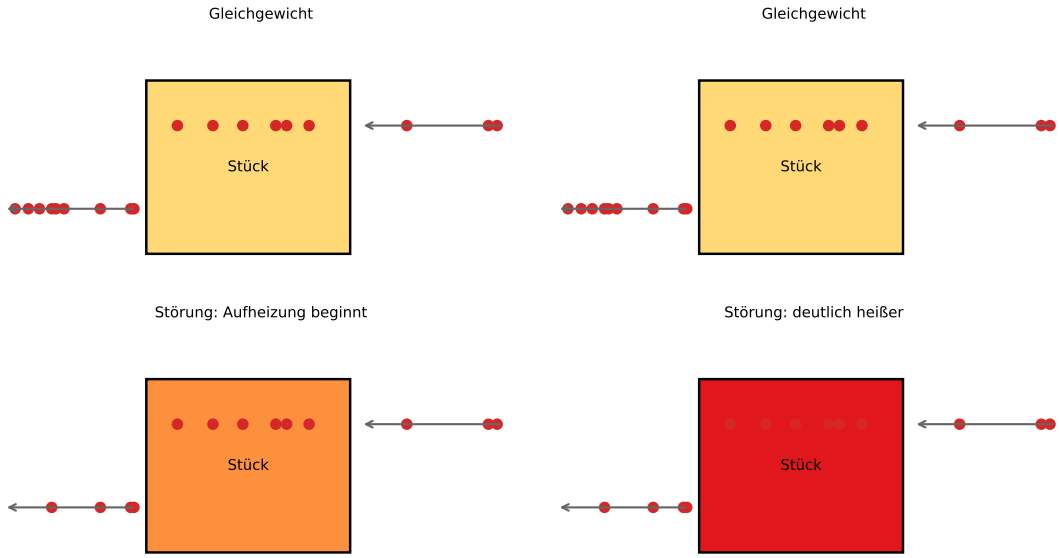

Abbildung 6.6.: Die Bilanz als Teilchenstrom (in der HTML-Fassung eine Animation): vier Momentaufnahmen. Oben im Gleichgewicht — gleich viele Teilchen rechts herein wie links hinaus, das Stück behält seine Farbe. Unten die Störung — es kommen mehr rein als raus, der Überschuss staut sich, das Stück heizt sich auf (Farbe wandert ins Rote). Das Aufheizen ist der Speicherterm von Kapitel 9.

Was man sieht: In der ersten Hälfte fließen gleich viele Teilchen rechts herein wie links hinaus — das Konto ist ausgeglichen, das Stück behält seine Temperatur. Dann wird der Abfluss gedrosselt: Es kommt mehr rein als raus, der Überschuss staut sich, und das Stück heizt sich auf (die Farbe wandert ins Rote). Genau dieses Aufheizen — die *Differenz* zwischen rein und raus, die sich in Temperatur verwandelt — ist der Speicherterm, den Kapitel 9 dazunimmt. Im eingeschwungenen Zustand dieses Kapitels ist die Differenz immer null.

6.6. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript liegt in `programme/kap06/kap06_bilanzprue`
Es ist der Bilanz-Prüfer als Kommandozeilen-Werkzeug: Es rechnet zuerst den Wärmestrom durch den eingeschwungenen Träger aus (die 140 W), prüft dann alle drei Kandidaten — Gerade, Knick, Parabel — an jedem inneren Knoten durch und druckt für jeden die Bilanztabelle mit dem

Urteil „bestanden" oder „durchgefallen". Zum Schluss speichert es eine Abbildung der drei Profile mit ihren grünen und roten Knoten als PNG. Sein Kern sind genau die Funktionen dieses Kapitels — `strom_durch_element` und `pruefe_bilanz` —, keine Zeile mehr.

Merkkasten

- **Fouriers Gesetz:** Der Wärmestrom ist $q = -\lambda\,\mathrm{d}T/\mathrm{d}x$ — das **Gefälle treibt den Strom**, λ misst, wie gut das Material leitet, das Minuszeichen heißt „bergab" (vom Warmen zum Kalten).
- Durch unseren eingeschwungenen Träger fließen **140 W** — die Wärme einer hellen Glühbirne, ununterbrochen vom Brandherd zum kühlen Auflager.
- Die **Bilanz** an jedem Stück lautet: rein minus raus ist im Gleichgewicht null. Zu Ende gedacht (Stück auf null geschrumpft) wird daraus die **stationäre Wärmeleitungsgleichung** $\frac{\mathrm{d}}{\mathrm{d}x}(\lambda\,\frac{\mathrm{d}T}{\mathrm{d}x}) = 0$ — „der Wärmestrom ist überall gleich".
- Eine **Differentialgleichung** sucht keine Zahl, sondern eine **ganze Funktion** $T(x)$: die Funktion, deren Steigungen die Bilanz überall erfüllen.
- Im homogenen Träger ist diese Funktion eine **Gerade** — kein Wärmestau, die Mitte exakt in der Mitte. Der Bilanz-Prüfer zeigt: nur die Gerade besteht an allen Knoten.

Roter Faden

Zurück: Die **Steigung** $\mathrm{d}T/\mathrm{d}x$ aus Kapitel 5 ist hier zum Motor des Wärmestroms geworden — Fouriers Gesetz macht aus dem Steigungsdreieck eine Zahl in Watt. Das „zu Ende gedachte" Schrumpfen des Steigungsdreiecks (Kapitel 5) hat aus der Bilanz am endlichen Stück die Differentialgleichung gemacht. Und der Wärmestrom durch ein Element ist λ mal Differenz durch Abstand — dieselbe Differenz-durch-Abstand, die in Kapitel 5 die Elementsteigungen waren.

Vor: Diese Differentialgleichung *raten* konnten wir nur, weil der Träger gutmütig ist — homogen, gerade, in 1D. Bei Materialwechseln (der Knick der Übungsaufgabe), Wärmequellen (die Parabel), krummen Geometrien oder in der Fläche geht das Raten nicht mehr. Kapitel 7 zeigt, wie man dieselbe Gleichung in die **Federketten-Maschinerie** aus Kapitel 3 übersetzt und damit *jedes* Profil ausrechnet — der eigentliche Auftritt der FEM. Kapitel 8 macht die Randbedingungen ernst, Kapitel 9 fügt den Speicherterm (die Zeit) hinzu, und die Gerade wie die Parabel werden in Kapitel 12 zu Prüffällen, an

denen wir die FEM verifizieren.

Übungen

Ü 6.1 (Verstehen). Der Träger wird auf beiden Seiten heißer gehalten: links 50 °C, rechts 250 °C, sonst alles unverändert ($L = 1$ m, $\lambda = 50$ W/(m K), $A = 0{,}01$ m^2). Rechne **ohne Rechner** aus: (a) das Temperaturgefälle in K/m, (b) die Wärmestromdichte q in W/m^2, (c) den Wärmestrom I durch den Träger in Watt.

> **i** Musterlösung zu Ü 6.1
>
> (a) Gefälle $= (250 - 50)/1{,}0 = 200$ K/m. (b) $q = \lambda \cdot 200 = 50 \cdot 200 = 10\,000$ W/m^2. (c) $I = q \cdot A = 10\,000 \cdot 0{,}01 = 100$ W. Der kleinere Temperaturunterschied (200 statt 280 K) treibt einen kleineren Strom (100 statt 140 W) — das Gefälle treibt den Strom.

Ü 6.2 (Verändern). Stelle dir vor, du **verdoppelst** im Bilanz-Prüfer die Wärmeleitfähigkeit auf `WAERMELEITFAEHIGKEIT = 100.0` (statt 50) — der Träger bestünde dann aus einem doppelt so gut leitenden Werkstoff. Die Randwerte bleiben 20 und 300 °C. **Sage zuerst vorher:** Ändert sich am eingeschwungenen *Profil* etwas (an den Knotentemperaturen)? Und ändert sich am *Wärmestrom*? Prüfe deine Vorhersage, indem du im Kapitel-Programm oder im Prüfer λ verdoppelst. Achtung — das ist eine beliebte Falle!

> **i** Musterlösung zu Ü 6.2
>
> Das **Profil bleibt exakt gleich** — dieselbe Gerade 20 → 300 °C, Mitte 160 °C. Denn die Bilanzbedingung „rein = raus" heißt slope-links = slope-rechts, und λ kürzt sich dabei heraus (es steht auf beiden Seiten der Bilanz). Ein besserer Leiter ändert nicht, *wie* die Temperatur verteilt ist, solange die Enden festgehalten werden. Der **Wärmestrom verdoppelt sich** aber: $I = \lambda \cdot \text{Gefälle} \cdot A = 100 \cdot 280 \cdot 0{,}01 = 280$ W statt 140 W. Ein doppelt so guter Leiter schaufelt bei gleichem Gefälle doppelt so viel Wärme durch. Die Falle ist, „bessere Leitung" mit „andere Temperaturen" zu verwechseln — im eingeschwungenen Zustand mit festen Randwerten stimmt das nicht.

Ü 6.3 (Übertragen). In den Baustahlträger ist ein Stück aus **rostfreiem Edelstahl** eingeschweißt — etwa dort, wo eine korrosionsfeste Verbindung sitzt. Rechne mit zwei Hälften: die linke (0 bis 0,5 m) bleibt Baustahl ($\lambda = 50$), die rechte (0,5 bis 1,0 m) ist Edelstahl ($\lambda = 15$). Randwerte wie

immer: links 20 °C am Auflager, rechts 300 °C am Brandherd. **Skizziere das Profil zuerst von Hand.** Überlege: Durch beide Hälften fließt derselbe Wärmestrom (rein = raus, sonst wäre es nicht eingeschwungen). Wo das Material schlechter leitet (der Edelstahl), muss das Gefälle *steiler* sein, um denselben Strom zu treiben — das Profil bekommt einen **Knick** an der Materialgrenze. Rechne die Übergangstemperatur bei 0,5 m aus und prüfe alles mit dem Skript `loesungen/kap06_ue3.py`.

Musterlösung zu Ü 6.3

Weil derselbe Strom durch beide Hälften fließt, gilt $\lambda_{\text{Bau}} \cdot \text{Gefälle}_{\text{Bau}} = \lambda_{\text{Edel}} \cdot \text{Gefälle}_{\text{Edel}}$. Also verhalten sich die Gefälle umgekehrt wie die Leitfähigkeiten:

$$\frac{\text{Gefälle}_{\text{Edel}}}{\text{Gefälle}_{\text{Bau}}} = \frac{\lambda_{\text{Bau}}}{\lambda_{\text{Edel}}} = \frac{50}{15} = \frac{10}{3} \approx 3{,}3.$$

Der Edelstahl hat das **gut dreifache Gefälle**. Beide Hälften sind gleich lang, also verteilt sich der ganze Temperatursprung von 280 K im Verhältnis $15 : 50$ (also $3 : 10$) auf Baustahl und Edelstahl: Über dem Baustahl fallen $280 \cdot \frac{3}{13} \approx 64{,}6$ K ab, über dem Edelstahl $280 \cdot \frac{10}{13} \approx 215{,}4$ K. Die **Übergangstemperatur** bei 0,5 m ist also

$$20 + 64{,}6 \approx 84{,}6\ ^\circ\text{C}.$$

Der gut leitende Baustahl bleibt vergleichsweise kühl (20 → 84,6 °C über die ganze linke Hälfte), im schlechter leitenden Edelstahl schießt die Temperatur dann steil hoch (84,6 → 300 °C). Der Knick sitzt genau an der Materialgrenze. Prüft man die Bilanz mit dem *jeweiligen* λ jeder Hälfte, geht sie an jedem Knoten auf (überall fließen dieselben rund 64,6 W) — der Knick ist kein Widerspruch zur Bilanz, sondern ihre Folge. Das vollständige Skript `loesungen/kap06_ue3.py` rechnet das Profil $[20,\ 52{,}3,\ 84{,}6,\ 192{,}3,\ 300]$ °C aus und weist die Stromkonstanz nach. Denselben Zwei-Material-Träger nimmt sich Kapitel 7 noch einmal vor — dort löst ihn die FEM-Maschinerie ganz ohne die Handüberlegung von hier (Ü 7.2).

Das Kleingedruckte

Drei ehrliche Feinheiten zum Schluss.

Erstens: Wir haben die Differentialgleichung durch *Raten* gelöst — eine Gerade eingesetzt und gesehen, dass sie passt. Das ist völlig legitim, aber

es funktioniert nur, weil unser Träger so gutmütig ist. Schon der Knick aus Übung 6.3 (zwei Materialien) und die Parabel (Wärmequelle) zeigen, dass das Profil auch ganz anders aussehen kann. Bei krummen Geometrien oder in der Fläche (1) gibt es überhaupt keine einfache Funktion mehr zu raten. Dafür wurde die FEM erfunden — Kapitel 7 baut sie.

Zweitens: Wir sind durchweg beim **eingeschwungenen** Zustand geblieben. Die transienten Momentaufnahmen aus Kapitel 1 und Kapitel 5 (die krumme 30-Minuten-Kurve beim stärker aufgeflammten Brandherd) gehorchen einer *größeren* Gleichung, die den Speicherterm — das Aufheizen aus der Animation — enthält. Sie ist das Thema von Kapitel 9. Die Sinusreihen-Formel, mit der Kapitel 5 seine „Fühler"-Kurve berechnet hat, ist die exakte Lösung genau dieser größeren Gleichung; woher sie kommt, klärt sich dort und in Kapitel 12.

Drittens: Wir haben so getan, als flösse die Wärme nur *längs* durch den Träger und nirgends an die umgebende Luft verloren. Das ist eine bewusste Vereinfachung (ein perfekt gedämmter Träger). In Wirklichkeit gibt der Träger auch seitlich Wärme ab; das würde einen weiteren Term hinzufügen und das Profil leicht nach unten biegen. Für das Verständnis der Bilanz ändert das nichts — es kommt nur ein weiterer Zu-/Abgang aufs Konto.

7. Die schwache Form: von der Gleichung zur Federkette

Zwei Bilder liegen jetzt auf dem Tisch, und sie scheinen nichts miteinander zu tun zu haben. Das eine ist die **Federkette** aus Kapitel 3: eine Handvoll Knoten, an denen Federn ziehen, zusammengefasst zu einer Tabelle aus Zahlen, die wir in 2 im Handumdrehen gelöst haben. Das andere ist die **Wärmeleitungsgleichung** aus Kapitel 6: eine einzige Gleichung, die nicht für fünf Knoten gilt, sondern für *unendlich viele Punkte gleichzeitig* — und deren Lösung keine Handvoll Zahlen ist, sondern eine ganze Kurve.

Dieses Kapitel schlägt die Brücke zwischen den beiden. Am Ende wirst du sehen: Die Wärmeleitung im Träger *ist* buchstäblich eine Federkette. Zwischen den Knoten sitzen **Wärme-Federn**, und ihre Steifigkeit hat sogar einen konkreten Wert — $\lambda A/h$, aus Wärmeleitfähigkeit, Querschnitt und Knotenabstand. Ist die Brücke einmal geschlagen, rechnet die Federketten-Maschinerie aus Kapitel 3 und 2 die Wärme genauso, wie sie die Verschiebung gerechnet hat: assemblieren, Randwerte einbauen, lösen.

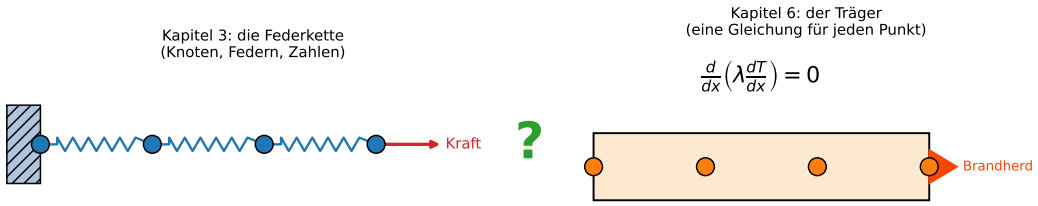

Abbildung 7.1.: Die Brücke dieses Kapitels. Links die Federkette aus Kapitel 3: Knoten, Federn, eine Zugkraft — gelöst mit reiner Schulalgebra. Rechts der Träger aus Kapitel 6: dieselben fünf Knoten, aber dazwischen die Wärmeleitungsgleichung, die für jeden Punkt gilt. Die Frage des Kapitels steht in der Mitte: Wie wird aus der Gleichung rechts die Kette links?

Was man hier sieht: links das Vertraute, rechts das scheinbar Fremde, in der Mitte das Fragezeichen dieses Kapitels. Die Leitfrage stellen wir bewusst **rückwärts**: nicht „was ist das für ein Verfahren?“, sondern ganz hand-

fest — *„Wie komme ich von der Gleichung rechts zur Kette links?“* Sieben Schritte weit ist der Weg, und jeder einzelne benutzt nur Werkzeuge, die du schon hast: die Federketten-Maschinerie (Kapitel 3), den Löser (2), Steigung und Fläche (Kapitel 5), die Bilanz (Kapitel 6).

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum man die Temperatur zwischen den Knoten durch **Geradenstücke** annähert, und wie **Hütchenfunktionen** das aufschreiben,
2. … das Prinzip der **gewichteten Reste** in einem Satz sagen — die Bilanz muss nicht überall exakt stimmen, aber im gewichteten Mittel um jeden Knoten,
3. … erklären, welches Problem die **Umverteilung** (partielle Integration) löst: Unsere Geradenstücke haben Knicke, zweimal ableiten geht nicht,
4. … die **2×2-Elementmatrix der Wärmeleitung** $\frac{\lambda A}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ selbst ausrechnen — nur mit Steigungen und Flächen aus Kapitel 5,
5. … das **5×5-System** des Trägers assemblieren und mit dem Thomas-Löser aus 2 lösen — und begründen, warum die FEM-Lösung hier die exakte Gerade trifft.

⚠ Naheliegende Vermutung

Vermutung: *„Eine Näherung aus groben Geradenstücken kann nie exakt sein. Wenn ich die krumme Wirklichkeit durch ein paar gerade Linien ersetze, handle ich mir immer einen Fehler ein.“*
Warum sie naheliegt: Das ganze Kapitel handelt vom Annähern — die glatte Temperatur wird durch einen eckigen Streckenzug ersetzt. Eckig statt glatt klingt nach Kompromiss, nach „ungefähr”, nach unvermeidlichem Fehler.
Was stattdessen stimmt: Für unseren homogenen Träger trifft die FEM die Lösung an den Knoten **exakt** — auf die Nachkommastelle. Der Grund ist verblüffend: Die wahre Lösung *ist* selbst eine Gerade (Kapitel 6), und einen Geradenzug kann man mit Geradenstücken perfekt nachzeichnen. Der Fehler lebt nur dort, wo die Wirklichkeit **krummer** ist als der Ansatz — bei Wärmequellen, Materialsprüngen, in der Fläche. Wie groß er dann ist und wie man ihn misst, vermessen wir in Kapitel 12. Hier erleben wir zuerst den schönen Sonderfall, in dem die grobe Näherung gar keine Näherung ist.

7.1. Schritt 1 — Der Ansatz: Zuständigkeitsgebiete

Fangen wir mit der einzigen Entscheidung an, die die ganze Methode trägt: **Wie beschreiben wir die Temperatur im Träger mit nur fünf Zahlen?** Wir kennen (oder suchen) die Temperatur an fünf Knoten — bei $0,\ 0{,}25,\ 0{,}5,\ 0{,}75,\ 1{,}0$ m. Aber die Temperatur gibt es an *jeder* Stelle dazwischen, nicht nur an den Knoten. Was setzen wir zwischen die Knoten?

Die Antwort haben wir in Kapitel 5 schon vorbereitet: **Zwischen zwei Knoten spannen wir eine Gerade.** Der Temperaturverlauf wird ein **Streckenzug** — als hätte jemand die Knotenwerte mit dem Lineal verbunden. Fünf Zahlen, vier gerade Stücke, fertig ist die ganze Kurve. Und weil jedes Stück eine Gerade ist, sind Steigung und Fläche darauf exakte Schulgeometrie (Dreieck und Trapez, Kapitel 5) — kein Grenzwert, kein Fehler beim Rechnen.

Jetzt kommt der Trick, der diesen Streckenzug in eine saubere Formel gießt. Wir fragen: **Welchen Beitrag leistet ein einzelner Knoten zur ganzen Kurve?** Denk an Knoten 3 in der Mitte. Sein Wert bestimmt die Kurve *ganz* an seinem eigenen Ort, ein bisschen weniger kurz davor und dahinter, und gar nicht mehr bei seinen Nachbarn. Sein Einfluss ist ein **Hütchen** (oder ein Zelt): Über dem Knoten steht es auf der Höhe 1, zu beiden Nachbarn fällt es geradlinig auf 0 ab, und weiter draußen ist es null. Abbildung 7.2 zeigt alle fünf.

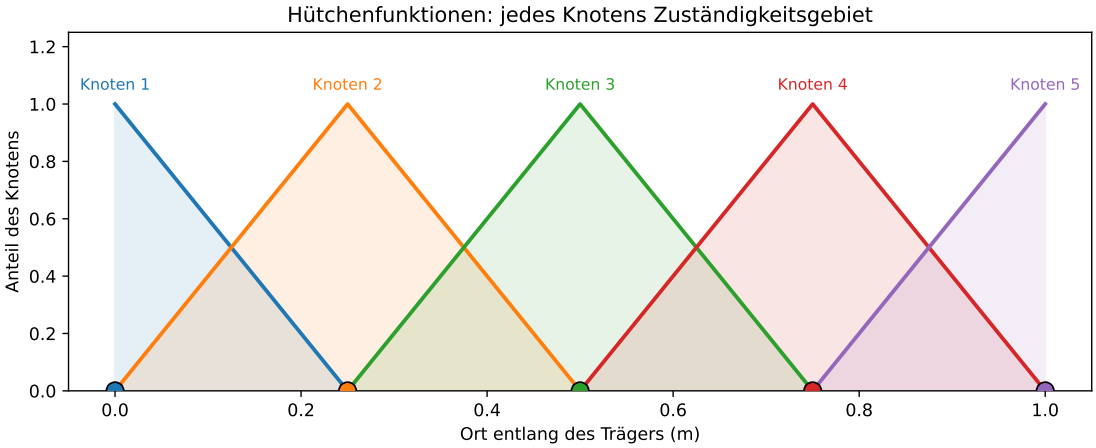

Abbildung 7.2.: Die fünf Hütchenfunktionen über dem Träger — je eine pro Knoten. Jedes Hütchen steht über seinem eigenen Knoten auf 1 und fällt zu den Nachbarn geradlinig auf 0. Das ist das „Zuständigkeitsgebiet" des Knotens: Nur dort, wo sein Hütchen von null verschieden ist, hat er ein Wort mitzureden. An jeder Stelle überlappen sich höchstens zwei Hütchen — darum wird jeder Punkt von höchstens zwei Knoten „regiert".

Was man hier sieht: fünf Hütchen, für jeden Knoten eines. Über seinem eigenen Ort trägt ein Knoten voll (Höhe 1), bei den Nachbarn gar nicht (Höhe 0). Wichtig ist die **Überlappung**: An jeder Stelle des Trägers sind höchstens zwei Hütchen von null verschieden — das linke und das rechte des Abschnitts, in dem man steht. Jeder Punkt wird also von höchstens zwei Knoten „regiert", und ihre beiden Hütchen sagen, zu welchen Anteilen. Genau das ist **Interpolation**: die Temperatur an einer Zwischenstelle als Mischung der beiden zuständigen Knotenwerte.

Diese Beschreibung lässt sich Wort für Wort in reines Python gießen. Ein Hütchen ist eine denkbar einfache Funktion — 1 am eigenen Knoten, linear fallend, 0 ab dem Nachbarn:

```
def huetchenwert(knoten_nummer, ort, knotenabstand):
    """Wert der Huetchenfunktion eines Knotens an einer Stelle.

    knoten_nummer: welcher Knoten (0, 1, 2, ...).
    ort: die Stelle auf dem Träger (Meter).
    knotenabstand: der gleiche Abstand h zwischen zwei Knoten (Meter).
    Rueckgabe: der Anteil des Knotens an dieser Stelle (zwischen 0 und 1).
```

```
    Das Huetchen steht ueber dem eigenen Knoten auf 1 und faellt zu beiden
    Nachbarn geradlinig auf 0; ab dem Nachbarn ist es null.
    """
    knoten_ort = knoten_nummer * knotenabstand
    abstand = abs(ort - knoten_ort)
    if abstand >= knotenabstand:
        return 0.0
    return 1.0 - abstand / knotenabstand


def interpoliere(temperaturen, ort, knotenabstand):
    """Temperatur an einer beliebigen Stelle aus den Knotenwerten.

    temperaturen: Liste der Knotentemperaturen (Grad Celsius).
    ort: die Stelle auf dem Träger (Meter).
    knotenabstand: der gleiche Abstand h zwischen zwei Knoten (Meter).
    Rueckgabe: die interpolierte Temperatur an dieser Stelle.

    Die Kurve ist die Summe: jeder Knotenwert mal seinem Huetchen. Weil an
    jeder Stelle nur zwei Huetchen ungleich null sind, mischt die Summe
    genau die zwei zustaendigen Knotenwerte.
    """
    wert = 0.0
    for knoten in range(len(temperaturen)):
        wert = wert + temperaturen[knoten] * huetchenwert(knoten, ort,
                                                          knotenabstand)
    return wert


# Ein Beispielprofil an den fuenf Knoten und ein paar Zwischenstellen.
KNOTENTEMPERATUREN = [20.0, 90.0, 160.0, 230.0, 300.0]
KNOTENABSTAND = 0.25

print("Ort (m) | interpolierte Temperatur")
print("--------+-------------------------")
for schritt in range(9):
    ort = schritt * 0.125
    print("  %.3f | %7.1f °C" % (ort, interpoliere(KNOTENTEMPERATUREN, ort,
                                                   KNOTENABSTAND)))
```

```
Ort (m) | interpolierte Temperatur
--------+-------------------------
  0.000 |    20.0 °C
```

```
0.125 |    55.0 °C
0.250 |    90.0 °C
0.375 |   125.0 °C
0.500 |   160.0 °C
0.625 |   195.0 °C
0.750 |   230.0 °C
0.875 |   265.0 °C
1.000 |   300.0 °C
```

Interpretation der Ausgabe: An den Knoten (0, 0,25, 0,5, …) kommen genau die Knotenwerte heraus — 20, 90, 160, … —, weil dort das eigene Hütchen 1 ist und alle anderen 0 sind. Genau *zwischen* zwei Knoten (etwa bei 0,125 m: 55 °C) liegt der Mittelwert der beiden Nachbarn, weil sich dort ihre beiden Hütchen zu je einem halben Anteil mischen. Die Zeile `temperaturen[knoten] * huetchenwert(...)`, aufsummiert, ist die ganze Interpolation — und sie ist buchstäblich eine **gewichtete Summe** wie der Mittelwert aus Kapitel 2, nur dass die Gewichte die Hütchen sind.

7.1.1. Fünf Zahlen bestimmen die ganze Kurve

Der Streckenzug hat eine Eigenschaft, die man gesehen haben muss: Verschiebt man einen einzigen Knotenwert, folgt die Kurve — aber nur in der Umgebung dieses Knotens, weil sein Hütchen weiter draußen null ist. Die folgende Animation zieht die Knotenwerte auf und ab; achte darauf, wie der Geradenzug gehorcht und wie ruhig die entfernten Abschnitte bleiben.

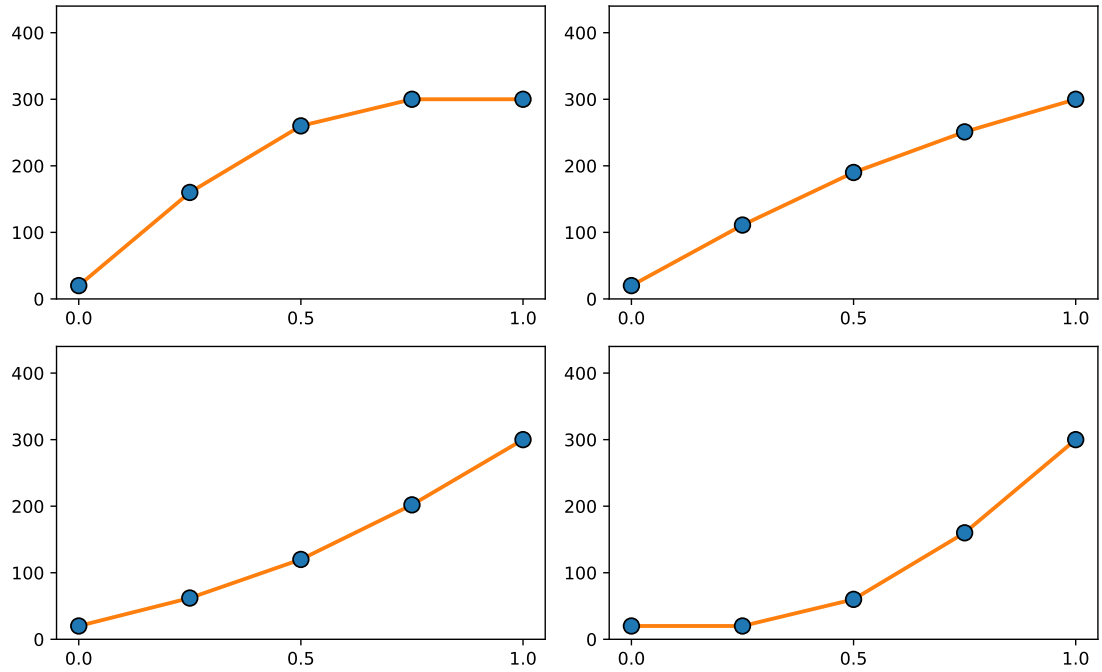

Abbildung 7.3.: Fünf Zahlen bestimmen die Kurve (in der HTML-Fassung eine Animation): vier Stellungen der Knotenwerte. Jedes Mal folgt der Streckenzug den Punkten; die Ränder bleiben fest. Verschiebt sich ein Knoten, verbiegt sich die Kurve nur in seiner Nachbarschaft.

Was man sieht: Der Geradenzug ist völlig durch die fünf Knotenwerte festgelegt — kein Freiheitsgrad mehr dazwischen. Das ist die Kernidee des Ansatzes: Aus der unendlich langen Wunschliste „Temperatur an *jedem* Punkt" haben wir **fünf Stellschrauben** gemacht. Die Kunst des restlichen Kapitels ist, die richtigen fünf Zahlen zu finden.

7.2. Schritt 2 — Einsetzen und scheitern

Wir haben einen Ansatz mit fünf Stellschrauben. Der nächste Gedanke liegt auf der Hand: **Setz den Streckenzug in die Wärmeleitungsgleichung ein und schau, welche Knotenwerte sie erfüllen.** Die Gleichung aus Kapitel 6 lautet

$$\frac{\mathrm{d}}{\mathrm{d}x}\left(\lambda\,\frac{\mathrm{d}T}{\mathrm{d}x}\right) = 0.$$

Sie verlangt zwei Ableitungen hintereinander: erst die Steigung des Profils $\mathrm{d}T/\mathrm{d}x$, dann die Steigung *davon*. Und genau hier laufen wir gegen eine Wand. Sehen wir uns an, was mit den Ableitungen unseres Streckenzugs passiert.

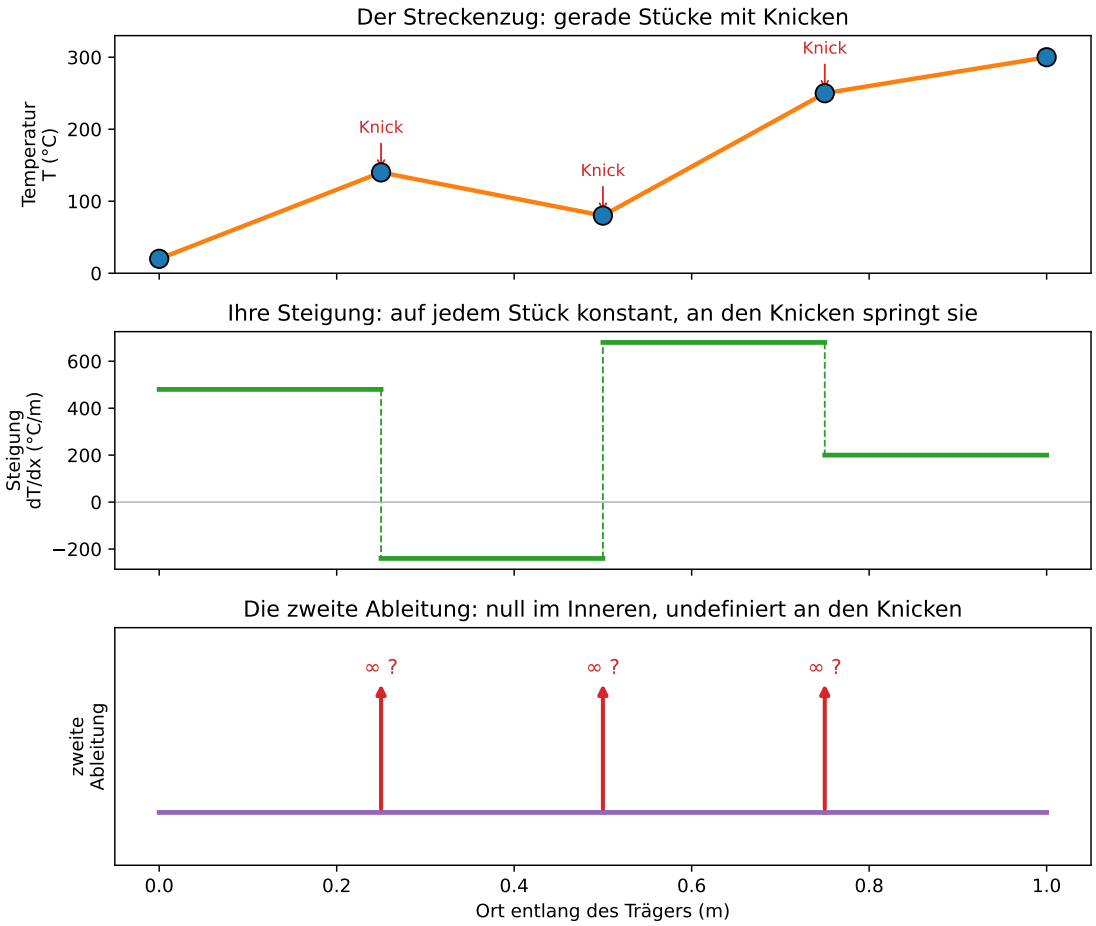

Abbildung 7.4.: Warum das Einsetzen scheitert. Oben ein Streckenzug (irgendein Kandidatenprofil) mit scharfen Knicken an den Knoten. In der Mitte seine Steigung dT/dx: auf jedem geraden Stück konstant, aber an den Knicken springt sie schlagartig — eine Treppe mit senkrechten Stufen. Unten die zweite Ableitung: innerhalb der Stücke null, aber an den Knicken müsste sie unendlich groß werden (die roten Blitze). Dort, wo die Wärmeleitungsgleichung ihre zweite Ableitung braucht, existiert sie nicht.

Was man hier sieht: eine ehrliche Krise. Der Streckenzug (oben) hat an jedem Knoten einen **Knick**. Seine Steigung (Mitte) ist auf jedem geraden

Stück eine brave Konstante — aber an den Knicken *springt* sie schlagartig von einem Wert auf den nächsten, ohne Übergang. Und die zweite Ableitung (unten), die ja die Steigung *der Steigung* misst, ist innerhalb der Stücke null (eine Konstante hat keine Steigung), müsste aber an den Sprungstellen unendlich groß werden: eine endliche Stufe in null Breite. Sie **existiert dort nicht**.

Das ist der Haken. Die Wärmeleitungsgleichung fordert die zweite Ableitung an *jeder* Stelle — auch an den Knoten. Aber genau an den Knoten, wo unsere Stellschrauben sitzen, stirbt die zweite Ableitung. Wir können den Streckenzug schlicht nicht in die Gleichung einsetzen. Der direkte Weg ist versperrt.

> **Warum wir den Ansatz trotzdem behalten**
>
> Man könnte in Versuchung geraten, den Streckenzug wegzuwerfen und eine glatte Kurve zu suchen, die man beliebig oft ableiten kann. Aber der Streckenzug ist ein Geschenk: Auf ihm sind Steigung und Fläche exakte Schulgeometrie (Kapitel 5), und mit fünf Zahlen ist er computertauglich einfach. Statt den guten Ansatz aufzugeben, ändern wir lieber die *Frage*. Der nächste Schritt formuliert die Wärmeleitungsgleichung so um, dass sie mit der einen Ableitung auskommt, die unser Streckenzug hat — und die zweite gar nicht mehr braucht. Aus der Not wird die Methode geboren.

7.3. Schritt 3 — Gewichtete Reste: die Bilanz im Mittel

Kehren wir zurück zum Ursprung der Gleichung. Sie war nie ein Selbstzweck — sie war die **Bilanz** aus Kapitel 6: An jedem Punkt soll gleich viel Wärme herein- wie hinausströmen (rein = raus). Die Forderung „an *jedem* Punkt" ist das Problem: Es gibt unendlich viele Punkte, aber wir haben nur fünf Stellschrauben. Fünf Zahlen können unmöglich unendlich viele Bedingungen erfüllen.

Was tut man, wenn man eine Bedingung nicht überall erzwingen kann? Man prüft sie **stichprobenartig, aber klug**. Nicht an einzelnen Punkten (das wäre Willkür), sondern als **gewichtetes Mittel**: Man fragt nicht „stimmt die Bilanz genau hier¿', sondern „stimmt die Bilanz *im Mittel um diesen Knoten herum*?". Und man stellt genau **eine solche Frage pro Knoten** — so viele Fragen, wie es Stellschrauben gibt. Fünf Knoten, fünf Bilanzfragen, fünf Gleichungen: Das passt genau zu einem lösbaren System.

Die entscheidende Wahl ist, *mit welchem Gewicht* man mittelt. Und hier kommt die eleganteste Idee der ganzen Methode: **Wir nehmen als Gewicht dasselbe Hütchen.** Das Hütchen von Knoten 3 ist ja gerade sein Zuständigkeitsgebiet — es ist groß an seinem Ort und null bei den Nachbarn. Als Gewicht benutzt, „befühlt" es die Bilanz genau dort, wo dieser Knoten etwas zu sagen hat, und ignoriert den Rest des Trägers.

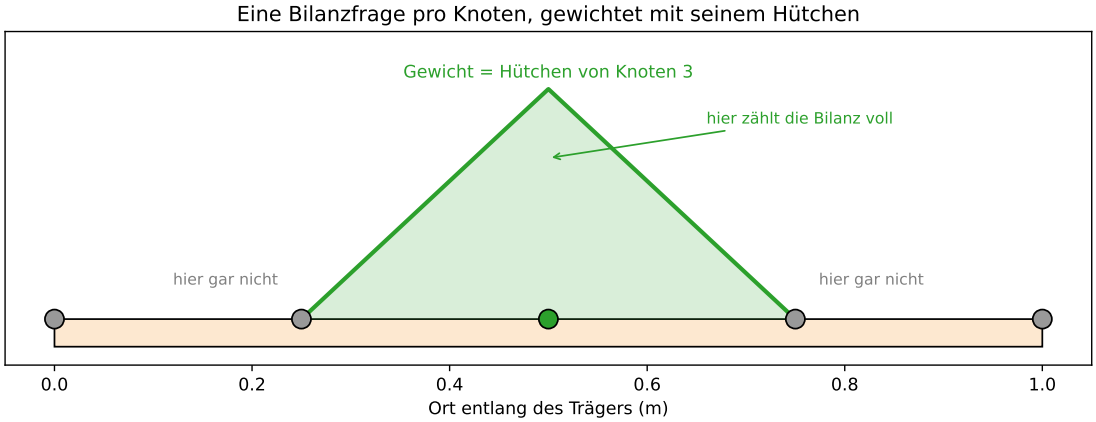

Abbildung 7.5.: Das Hütchen als Gewicht. Für die Bilanzfrage an Knoten 3 (grün) legen wir sein eigenes Hütchen als Wichtung über den Träger. Wo das Hütchen hoch ist (am Knoten), zählt die Bilanz voll; wo es abfällt (zu den Nachbarn), immer weniger; außerhalb (grau) gar nicht. So wird aus „Bilanz genau am Punkt" ein gewichtetes Mittel um den Knoten — eine Frage, die unser Streckenzug beantworten kann.

Was man hier sieht: die Bilanzfrage des mittleren Knotens, gewichtet mit seinem eigenen Hütchen. Die Bilanz wird nicht überall gleich streng genommen: Direkt am Knoten zählt sie voll, zu den Nachbarn hin immer weniger, außerhalb gar nicht. Jeder Knoten stellt so *seine* lokale Bilanzfrage — „stimmt die Wärmebilanz in meinem Einzugsbereich?" —, und die fünf Antworten sind die fünf Gleichungen des Systems. Aufgeschrieben lautet die Frage für Knoten i: die mit dem Hütchen N_i gewichtete Bilanz über den ganzen Träger soll null sein,

$$\int_0^L \underbrace{\frac{\mathrm{d}}{\mathrm{d}x}\left(\lambda \frac{\mathrm{d}T}{\mathrm{d}x}\right)}_{\text{Bilanz an der Stelle}} \cdot \underbrace{N_i(x)}_{\text{Gewicht}} \, \mathrm{d}x \;=\; 0.$$

Das lange $\int$ ist die Summe aus Kapitel 5 — hier summiert es die gewichtete Bilanz über alle Orte. Nur: In der Klammer steht immer noch die **zweite**

Ableitung, die unser Streckenzug nicht hat. Diesen letzten Stein räumt der nächste Schritt weg.

7.4. Schritt 4 — Umverteilung: die Steigungslast auf beide Schultern

Jetzt kommt der technische Kunstgriff des Kapitels — und er ist harmloser, als sein späterer Name („partielle Integration") klingt. In einer Dimension ist er nichts als eine **Umverteilung** von Arbeit.

Das Problem: In dem Integral hängen *beide* Ableitungen an der Temperatur T — erst die Steigung, dann die Steigung davon. Auf T lastet die ganze „Steigungslast", und die zweite davon kann unser Streckenzug nicht tragen. Das Gewicht N_i dagegen steht unbeteiligt daneben und wird gar nicht abgeleitet.

Die Umverteilung nimmt T eine der beiden Ableitungen ab und legt sie dem Gewicht auf die Schulter: **Statt T zweimal abzuleiten, leiten wir T einmal ab und das Gewicht einmal.** Danach trägt jeder nur noch eine Steigung — und *eine* Steigung hat unser Streckenzug, das ist ja gerade seine brave Treppe aus Abbildung 7.4. Die unmögliche zweite Ableitung verschwindet.

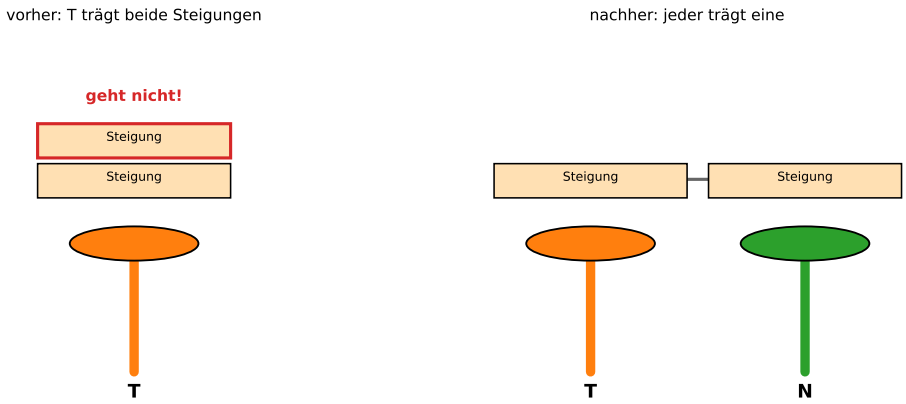

Abbildung 7.6.: Die Umverteilung als Lastentausch. Links trägt die Temperatur T beide Steigungen allein (die zweite kann der Streckenzug nicht stemmen — rotes Kreuz). Rechts nach der Umverteilung: T trägt eine Steigung, das Gewicht N die andere. Jeder trägt nur noch eine — und eine Steigung hat der Streckenzug. Die Last ist auf zwei Schultern verteilt.

Was man hier sieht: links die Überlastung — die Temperatur soll zwei Steigungen zugleich tragen, die obere geht nicht (rotes Kreuz). Rechts der Ausgleich: eine Steigung bleibt bei T, die andere wandert zum Gewicht N. Beide tragen jetzt eine tragbare Last. Genau das leistet die Umverteilung im Integral. Aus der gewichteten Bilanz wird

$$\int_0^L \frac{\mathrm{d}}{\mathrm{d}x}\left(\lambda \frac{\mathrm{d}T}{\mathrm{d}x}\right) N_i \,\mathrm{d}x \;=\; \underbrace{\left[N_i\,\lambda \frac{\mathrm{d}T}{\mathrm{d}x}\right]_0^L}_{\text{Randterm — geparkt}} \;-\; \int_0^L \lambda \,\frac{\mathrm{d}N_i}{\mathrm{d}x}\,\frac{\mathrm{d}T}{\mathrm{d}x}\,\mathrm{d}x.$$

Lies das rechte Integral: Jetzt steht dort $\frac{\mathrm{d}N_i}{\mathrm{d}x} \cdot \frac{\mathrm{d}T}{\mathrm{d}x}$ — die Steigung des Gewichts mal der Steigung der Temperatur, **beide nur einfach abgeleitet**. Kein zweites Mal, kein Sterben an den Knicken. Die Umverteilung hat die Gleichung genau auf das Werkzeug zugeschnitten, das unser Streckenzug bietet.

> **Der Randterm — hier geparkt, in Kapitel 8 abgeholt**
>
> Der eckige Klammerausdruck $[N_i\,\lambda\,\mathrm{d}T/\mathrm{d}x]_0^L$ entsteht bei der Umverteilung immer — er zählt, was an den beiden **Enden** des Trägers ($x = 0$ und $x = L$) passiert. Für einen **inneren** Knoten ist er harmlos: Dessen Hütchen N_i ist an beiden Trägerenden null (es reicht ja nur bis zu den Nachbarn), also ist die ganze Klammer null, und der Term verschwindet. Nur für die beiden **Randknoten** bleibt er stehen — und dort steckt genau die Information über den Wärmestrom, der am Rand ein- oder austritt. Das ist der Stoff von Kapitel 8 („Randbedingungen"). Wir zeigen den Term hier ehrlich und **parken** ihn: Für die inneren Knoten, um die es in diesem Kapitel geht, ist er null.

Setzt man den Randterm für die inneren Knoten auf null, bleibt die schlichte Forderung

$$\int_0^L \lambda \,\frac{\mathrm{d}N_i}{\mathrm{d}x}\,\frac{\mathrm{d}T}{\mathrm{d}x}\,\mathrm{d}x \;=\; 0.$$

Und jetzt kommt eine schöne Überraschung. Rechnet man dieses Integral für das Hütchen eines inneren Knotens aus (das tun wir gleich, Schritt für Schritt), so ergibt sich Zeile für Zeile *genau* die Bilanzgleichung aus Kapitel 6: „Was durch das linke Element hereinfließt, gleicht, was durch das

rechte hinausfließt.” Der ganze Apparat — Ansatz, gewichtete Reste, Umverteilung — hat die Bilanz nicht ersetzt, sondern sie **systematisch hergeleitet**, wo wir sie in Kapitel 6 nur raten konnten. Der Bilanz-Prüfer von damals hat schon genau diese Gleichung geprüft.

7.5. Schritt 5 — Die Elementmatrix, und der Gänsehaut-Moment

Jetzt rechnen wir das Integral tatsächlich aus — und zwar so, wie es die FEM immer tut: **ein Element nach dem anderen.** Wir zoomen auf ein einzelnes Element, das zwischen einem linken Knoten a und einem rechten Knoten b liegt, im Abstand $h = 0{,}25$ m. Auf diesem Element leben nur zwei Hütchen: das von a (fällt über das Element von 1 auf 0) und das von b (steigt von 0 auf 1). Alle anderen Hütchen sind hier null.

Ihre **Steigungen** sind das Herz der Rechnung, und sie sind wieder reine Schulgeometrie (Kapitel 5):

- Das Hütchen von a fällt über die Breite h um 1: Steigung $\dfrac{0-1}{h} = -\dfrac{1}{h} = -\dfrac{1}{0{,}25} = -4\ \frac{1}{\mathrm{m}}$.
- Das Hütchen von b steigt über die Breite h um 1: Steigung $\dfrac{1-0}{h} = +\dfrac{1}{h} = +4\ \frac{1}{\mathrm{m}}$.

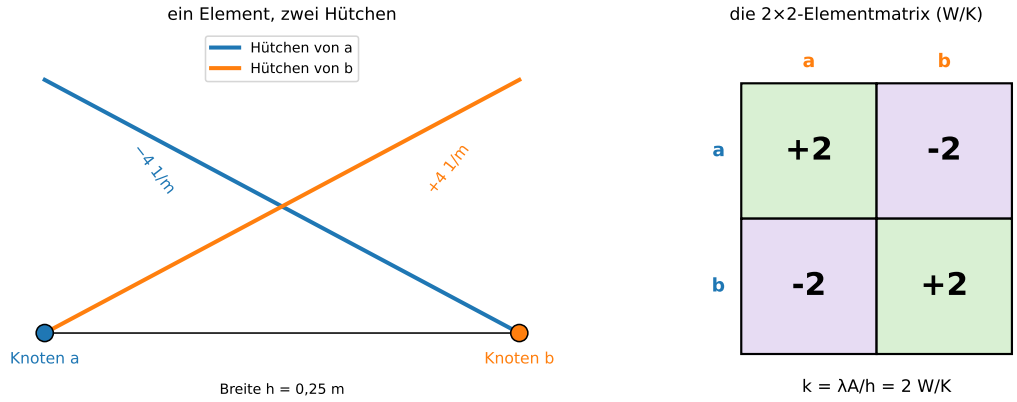

Abbildung 7.7.: Das Element-Rechenblatt. Auf einem Element (Breite h = 0,25 m) leben zwei Hütchen: das des linken Knotens a fällt von 1 auf 0 (Steigung −4 1/m), das des rechten Knotens b steigt von 0 auf 1 (Steigung +4 1/m). Aus diesen beiden Steigungen und der Breite baut sich die 2×2-Elementmatrix unten auf — jeder Eintrag ist Steigung × Steigung × λA × Breite.

Was man hier sieht: links die zwei Hütchen mit ihren Steigungen $\mp 4\ \frac{1}{\mathrm{m}}$, rechts das Ergebnis, das wir jetzt Eintrag für Eintrag ausrechnen. Jeder Eintrag der 2×2-Matrix ist das Integral $\int \lambda A \frac{\mathrm{d}N_r}{\mathrm{d}x}\frac{\mathrm{d}N_c}{\mathrm{d}x}\,\mathrm{d}x$ über das Element — und weil beide Steigungen über das ganze Element **konstant** sind, ist auch ihr Produkt konstant. Das Integral einer Konstanten ist aber die einfachste Fläche überhaupt: **Rechteck = Höhe mal Breite** (Kapitel 5, ganz ohne Streifenstapel). Höhe ist hier $\lambda A \cdot (\text{Steigung} \times \text{Steigung})$, Breite ist die Elementlänge h.

Rechnen wir alle vier Einträge mit $\lambda = 50\ \mathrm{W/(m\,K)}$, $A = 0{,}01\ \mathrm{m}^2$ (also $\lambda A = 0{,}5\ \mathrm{W\,m/K}$) und $h = 0{,}25\ \mathrm{m}$:

$$K_{aa} = \lambda A \cdot \left(-\tfrac{1}{h}\right)\left(-\tfrac{1}{h}\right) \cdot h = 0{,}5 \cdot (-4)(-4) \cdot 0{,}25 = 0{,}5 \cdot 16 \cdot 0{,}25 = +2\ \tfrac{\mathrm{W}}{\mathrm{K}},$$

$$K_{ab} = \lambda A \cdot \left(-\tfrac{1}{h}\right)\left(+\tfrac{1}{h}\right) \cdot h = 0{,}5 \cdot (-4)(+4) \cdot 0{,}25 = 0{,}5 \cdot (-16) \cdot 0{,}25 = -2\ \tfrac{\mathrm{W}}{\mathrm{K}},$$

$$K_{ba} = \lambda A \cdot \left(+\tfrac{1}{h}\right)\left(-\tfrac{1}{h}\right) \cdot h = -2\ \tfrac{\mathrm{W}}{\mathrm{K}},$$

$$K_{bb} = \lambda A \cdot \left(+\tfrac{1}{h}\right)\left(+\tfrac{1}{h}\right) \cdot h = +2\ \tfrac{\mathrm{W}}{\mathrm{K}}.$$

Alle vier Zahlen zusammengeschrieben ergeben die **Elementmatrix der Wärmeleitung**:

$$\mathbf{K}_{\text{Element}} = \frac{\lambda A}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 2 & -2 \\ -2 & 2 \end{pmatrix} \tfrac{\text{W}}{\text{K}}, \qquad k = \frac{\lambda A}{h} = \frac{50 \cdot 0{,}01}{0{,}25} = 2\,\tfrac{\text{W}}{\text{K}}.$$

Und jetzt halt inne und sieh genau hin.

Der Gänsehaut-Moment

Vergleiche die eben ausgerechnete Wärme-Elementmatrix mit dem Steckbrief der **Feder** aus Kapitel 3:

$$\underbrace{\begin{pmatrix} k & -k \\ -k & k \end{pmatrix}}_{\text{Feder (Kap. 3)}} \qquad \text{gegen} \qquad \underbrace{\frac{\lambda A}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}}_{\text{Wärme (Kap. 7)}}$$

Es ist dieselbe Matrix. Zahl für Zahl, Vorzeichen für Vorzeichen. Die Wärmeleitung in einem Trägerstück ist eine **Feder** — nur heißt ihre Steifigkeit nicht mehr „Federhärte", sondern $k = \lambda A/h$: viel Wärmeleitfähigkeit, dicker Querschnitt und kurzer Abstand machen eine „steife" Wärme-Feder. Zwischen unseren fünf Knoten sitzen also vier Wärme-Federn mit $k = 2$ W/K, und der ganze Rest des Kapitels ist Kapitel 3, noch einmal gespielt — mit Temperatur statt Verschiebung und Wärmestrom statt Kraft.

Diese Verwandtschaft ist kein Zufall und keine bloße Analogie — es ist buchstäblich dasselbe Gleichungssystem. Abbildung 7.8 stellt die beiden Ketten nebeneinander.

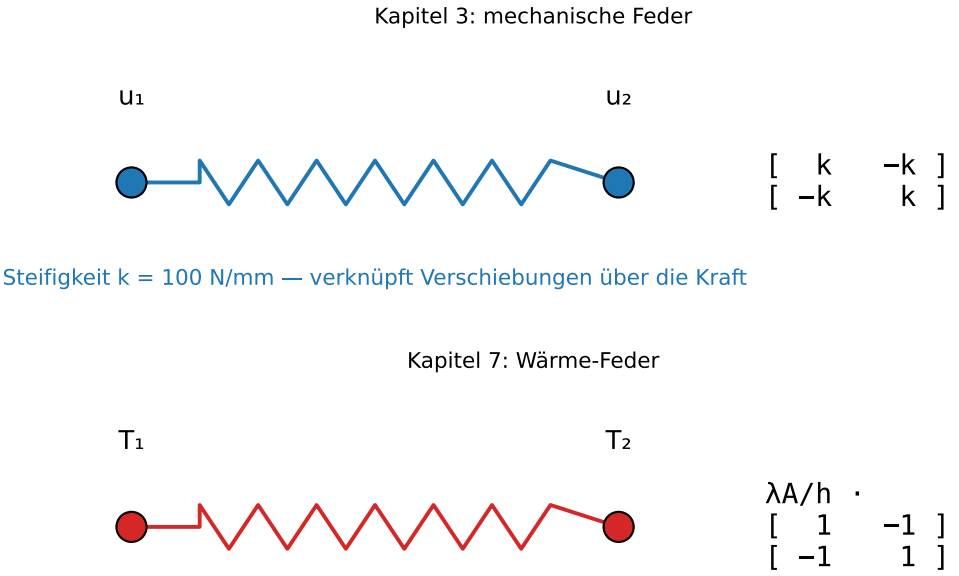

Abbildung 7.8.: Federkette und Wärmekette sind dieselbe Kette. Oben die mechanische Feder: Steifigkeit k, sie verknüpft Verschiebungen u über die Kraft. Unten die Wärme-Feder: Steifigkeit k = λA/h, sie verknüpft Temperaturen T über den Wärmestrom. Beide haben denselben 2×2-Steckbrief; nur die Namen der Größen sind ausgetauscht.

Was man hier sieht: oben und unten dieselbe Feder, nur anders beschriftet. Die mechanische verknüpft **Verschiebungen** u über die Kraft, die thermische verknüpft **Temperaturen** T über den Wärmestrom — aber ihr Steckbrief ist identisch. Damit ist die Brücke aus dem Aufhänger geschlagen: Ab hier rechnet die Federketten-Maschinerie die Wärme.

7.6. Schritt 6 — Assemblieren, einspannen, lösen

Vier Wärme-Federn mit $k = 2$ W/K, fünf Knoten — das ist genau die Ausgangslage von Kapitel 3, nur mit neuen Zahlen. Also spielen wir dasselbe **Schiebespiel**: Jeder Element-Steckbrief rutscht an die Stelle seiner beiden Knoten in die große 5×5-Tabelle, und wo sich zwei Federn einen Knoten teilen, addieren sich ihre k auf der Diagonale zu $2k = 4$ W/K. Die folgende Animation ist bewusst dieselbe wie in Kapitel 3 — nur stehen jetzt 2 und 4

und −2 in den Feldern statt 100 und 200 und −100. Das Wiedererkennen ist die halbe Botschaft.

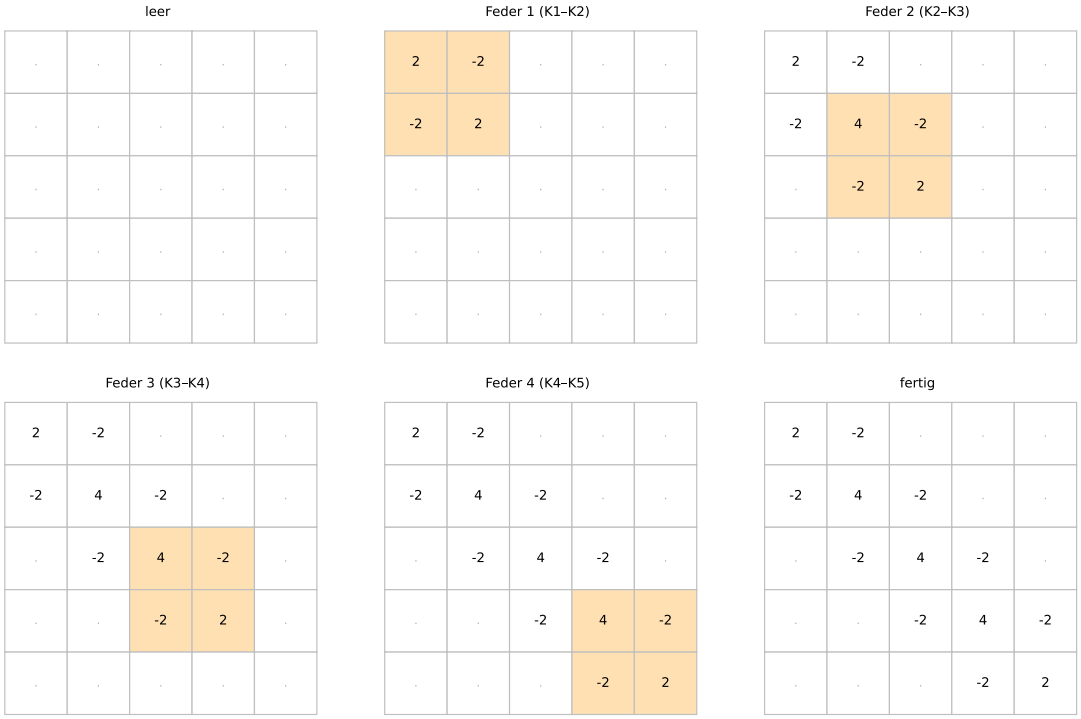

Abbildung 7.9.: Assemblierung der Wärmekette als Bildreihe (in der HTML-Fassung eine Animation): von der leeren 5×5-Tabelle über die vier Wärme-Federn zur vollständigen Systemmatrix. Auf den geteilten Diagonalfeldern wächst die Zahl von 2 auf 4, wenn die zweite anliegende Feder dazukommt — dasselbe Muster wie bei den Federn in Kapitel 3.

Was man sieht: dasselbe Wachsen wie bei den Federn, mit den Wärmezahlen. Nach vier Wärme-Federn steht die vollständige Systemmatrix — und sie trägt genau die Bandstruktur, die wir schon kennen: die Diagonale und je ein Feld links und rechts, alles andere leer.

7.6.1. Das vollständige 5×5-System

Schreiben wir es einmal ganz hin. Fünf Knoten, fünf Kraft-Gleichungen — Verzeihung, **Wärmestrom-Gleichungen**, eine pro Knoten. Auf der linken Seite die assemblierte Matrix (in W/K), auf der rechten die von außen zugeführten Wärmeströme (in W):

$$\begin{pmatrix} 2 & -2 & 0 & 0 & 0 \\ -2 & 4 & -2 & 0 & 0 \\ 0 & -2 & 4 & -2 & 0 \\ 0 & 0 & -2 & 4 & -2 \\ 0 & 0 & 0 & -2 & 2 \end{pmatrix} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \end{pmatrix} = \begin{pmatrix} Q_1 \\ 0 \\ 0 \\ 0 \\ Q_5 \end{pmatrix}$$

Die Diagonale $2/4/4/4/2$ erzählt dieselbe Geschichte wie bei den Federn: Die inneren Knoten 2, 3, 4 tragen je **zwei** Wärme-Federn (darum $4 = 2+2$), die Randknoten 1 und 5 nur eine (darum 2). Auf der rechten Seite steht bei den inneren Knoten **null** — dort speist niemand von außen Wärme ein (keine Quelle, das ist die reine Leitung). Bei den Randknoten stehen Q_1 und Q_5: die noch unbekannten Wärmeströme, die das kühle Auflager und der Brandherd abführen und zuführen — das genaue Gegenstück zur unbekannten Auflagerkraft F_1 der Federkette.

7.6.2. Die Randwerte einbauen

Wie bei der Feder ist das System so noch nicht lösbar: An den Rändern stehen unbekannte Ströme, und die Temperatur könnte als Ganzes nach oben oder unten rutschen. Was fehlt, ist die Information, die wir noch nicht benutzt haben: **Knoten 1 wird auf** $20\ °\mathrm{C}$ **gehalten, Knoten 5 auf** $300\ °\mathrm{C}$**.** Das sind die beiden Randwerte.

Wir bauen sie mit dem **Streich-Rezept** aus Kapitel 3 ein: Die beiden Randknoten sind bekannt, also streichen wir ihre Zeilen und Spalten und lösen nur noch für die drei inneren Knoten. Ein feiner Unterschied zur Feder: Dort war der feste Wert *null* ($u_1 = 0$), und eine gestrichene Spalte verschwand spurlos. Hier sind die Werte **nicht** null (20 und 300) — darum hinterlässt die gestrichene Spalte einen Beitrag auf der rechten Seite: der bekannte Randwert mal seinem Spalteneintrag wandert dorthin. Aus Zeile 2 zum Beispiel wird mit $T_1 = 20$:

$$-2 \cdot \underbrace{20}_{T_1} + 4\,T_2 - 2\,T_3 = 0 \quad \Longrightarrow \quad 4\,T_2 - 2\,T_3 = 2 \cdot 20 = 40.$$

Ebenso wandert bei Zeile 4 der Term $-2 \cdot 300 = -600$ nach rechts und wird zu $+600$. Übrig bleibt ein sauberes 3×3-System für die drei inneren Knoten:

$$\begin{pmatrix} 4 & -2 & 0 \\ -2 & 4 & -2 \\ 0 & -2 & 4 \end{pmatrix} \begin{pmatrix} T_2 \\ T_3 \\ T_4 \end{pmatrix} = \begin{pmatrix} 40 \\ 0 \\ 600 \end{pmatrix}$$

(*Warum* das Streichen und Verschieben genau die richtige Behandlung fester Randwerte ist — und was man mit Rändern noch alles anstellen kann —, ist so wichtig, dass Kapitel 8 ein ganzes Kapitel daraus macht. Hier benutzen wir das Rezept, die Begründung kommt dort.)

> Vorhersage-Punkt
>
> **Bevor wir lösen:** In Kapitel 6 haben wir mit reiner Bilanz-Physik begründet, dass das eingeschwungene Profil eine **Gerade** von 20 auf 300 °C ist — mit den Knotenwerten 20 / 90 / 160 / 230 / 300 °C. Gleich löst der Thomas-Algorithmus das 3×3-System der FEM. Was glaubst du: Trifft die FEM diese Gerade — und wenn ja, *ungefähr* oder ganz **genau**? Leg dich fest.

7.6.3. Lösen mit dem Thomas-Algorithmus

Das reduzierte System ist tridiagonal — genau die Bandform, für die wir in 2 den **Thomas-Algorithmus** gebaut haben. Er läuft als Vorwärts- und Rückwärtswelle einmal durch das Band. Wir füttern ihn mit den drei Diagonalen und der rechten Seite:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from loeser import thomas

# Das reduzierte 3x3-System der inneren Knoten (in W/K bzw. W).
unter = [0.0, -2.0, -2.0]        # Nebendiagonale links (unter[0] unbenutzt)
diagonale = [4.0, 4.0, 4.0]      # Hauptdiagonale
ober = [-2.0, -2.0, 0.0]         # Nebendiagonale rechts (ober[2] unbenutzt)
rechte_seite = [40.0, 0.0, 600.0]

innere = thomas(unter, diagonale, ober, rechte_seite)
temperaturen = [20.0] + innere + [300.0]

print("Knoten |   Ort   | FEM-Temperatur | exakte Gerade")
print("-------+---------+----------------+--------------")
for i in range(5):
    gerade = 20.0 + (300.0 - 20.0) * i / 4.0
    print("  %d    | %.2f m |   %6.1f °C    |   %6.1f °C"
          % (i + 1, i * 0.25, temperaturen[i], gerade))
```

```
Knoten |   Ort   | FEM-Temperatur | exakte Gerade
-------+---------+----------------+--------------
  1    | 0.00 m |     20.0 °C    |     20.0 °C
  2    | 0.25 m |     90.0 °C    |     90.0 °C
  3    | 0.50 m |    160.0 °C    |    160.0 °C
  4    | 0.75 m |    230.0 °C    |    230.0 °C
  5    | 1.00 m |    300.0 °C    |    300.0 °C
```

Interpretation der Ausgabe: $20/90/160/230/300$ °C — die FEM trifft die Gerade aus Kapitel 6 nicht ungefähr, sondern **exakt**, Knoten für Knoten. Damit ist die Vermutung vom Kapitelanfang widerlegt.

> ⚠ Vermutung aufgelöst
>
> **Die Vermutung war:** „Eine Näherung aus groben Geradenstücken kann nie exakt sein." **Und doch trifft die FEM hier die Wahrheit auf die Nachkommastelle.** Der Grund: Die wahre Lösung *ist* selbst eine Gerade (Kapitel 6), und eine Gerade lässt sich durch Geradenstücke perfekt darstellen — der Streckenzug *ist* dann die exakte Kurve, keine Näherung. Egal, ob wir 5 oder 500 Knoten nehmen: An den Knoten stimmt es immer exakt. Der Fehler taucht erst auf, wenn die Wirklichkeit **krummer** ist als unser Geradenzug — bei der Wärmequelle (Übung 7.3), beim Materialsprung (Übung 7.2), in der Fläche (1). Wie groß er dann ist, misst Kapitel 12.

Und die Physik dahinter ist dieselbe wie bei der Federkette. Rechne aus den Knotentemperaturen den Wärmestrom durch jedes Element zurück ($k \cdot$ Temperatursprung): Durch **jede** der vier Wärme-Federn fließen $2 \cdot 70 = 140$ W — exakt die 140 W aus Kapitel 6. So wie in Kapitel 3 jede mechanische Feder dieselbe Kraft von 50 N trug und sich gleich weit dehnte, trägt hier jede Wärme-Feder denselben Strom von 140 W und lässt die Temperatur um dieselben 70 K steigen. Die Treppe $20/90/160/230/300$ ist die Wärme-Version der Verschiebungstreppe $0/0{,}5/1{,}0/1{,}5$.

7.7. Die interaktive Einheit: die Wärme-Federkette

Jetzt bist du dran. Das folgende Programm ist die **Wärme-Federkette** als Ganzes: Du gibst die Zahl der Knoten vor (zwischen 3 und 20), und es baut den Element-Steckbrief, assembliert das System, baut die Randwerte ein und löst mit dem Thomas-Algorithmus — alle Schritte dieses Kapitels an einem Stück. Jede Codezeile trägt die Nummer des Papier-Schritts, dem sie entspricht.

Vorhersage-Punkt

Bevor du ausführst: Das Programm startet mit 5 Knoten. Ändere `KNOTENZAHL` gleich auf 20 und lass es erneut laufen. Was glaubst du: Ändern sich die Temperaturen **an den Knoten**, wenn du von 5 auf 20 Knoten verfeinerst — werden sie „genauer"? Oder bleiben die Knotenwerte, wo sie sind? Leg dich fest, bevor du es probierst.

```
#| autorun: true
# Die Waerme-Federkette. Aendere KNOTENZAHL (3 bis 20) und sieh zu.
KNOTENZAHL = 5

WAERMELEITFAEHIGKEIT = 50.0    # Baustahl S235, W/(m K)
QUERSCHNITT_M2 = 0.01          # 0,1 m x 0,1 m
TRAEGERLAENGE_M = 1.0
TEMPERATUR_LINKS = 20.0        # Auflager links, Grad Celsius
TEMPERATUR_RECHTS = 300.0      # Brandherd rechts, Grad Celsius


def thomas(unter, diagonale, ober, rechte_seite):
    """Loest ein tridiagonales System (Bandloeser aus Kapitel 4)."""
    n = len(diagonale)
    ober_neu = list(ober)
    seite_neu = list(rechte_seite)
    ober_neu[0] = ober[0] / diagonale[0]
    seite_neu[0] = rechte_seite[0] / diagonale[0]
    for i in range(1, n):
        nenner = diagonale[i] - unter[i] * ober_neu[i - 1]
        if i < n - 1:
            ober_neu[i] = ober[i] / nenner
        seite_neu[i] = (rechte_seite[i] - unter[i] * seite_neu[i - 1]) / nenner
    loesung = [0.0] * n
    loesung[n - 1] = seite_neu[n - 1]
    for i in range(n - 2, -1, -1):
        loesung[i] = seite_neu[i] - ober_neu[i] * loesung[i + 1]
    return loesung


elementzahl = KNOTENZAHL - 1
h = TRAEGERLAENGE_M / elementzahl

# Schritt 5: der Element-Steckbrief -- k = lambda*A/h.
k = WAERMELEITFAEHIGKEIT * QUERSCHNITT_M2 / h
```

```
# Schritt 6: assemblieren -- leere Matrix, dann jede Feder ihr +-k.
matrix = []
for zeile in range(KNOTENZAHL):
    matrix.append([0.0] * KNOTENZAHL)
for element in range(elementzahl):
    links = element
    rechts = element + 1
    matrix[links][links] = matrix[links][links] + k
    matrix[links][rechts] = matrix[links][rechts] - k
    matrix[rechts][links] = matrix[rechts][links] - k
    matrix[rechts][rechts] = matrix[rechts][rechts] + k

# Schritt 6: Randwerte einbauen (Streich-Rezept). Innere Knoten 1..n-2.
volle_seite = [0.0] * KNOTENZAHL
unter = []
diagonale = []
ober = []
seite = []
for i in range(1, KNOTENZAHL - 1):
    diagonale.append(matrix[i][i])
    unter.append(matrix[i][i - 1] if i - 1 >= 1 else 0.0)
    ober.append(matrix[i][i + 1] if i + 1 <= KNOTENZAHL - 2 else 0.0)
    beitrag = matrix[i][0] * TEMPERATUR_LINKS \
        + matrix[i][KNOTENZAHL - 1] * TEMPERATUR_RECHTS
    seite.append(volle_seite[i] - beitrag)

# Schritt 6: loesen (Thomas) und Raender wieder anfuegen.
innere = thomas(unter, diagonale, ober, seite)
temperaturen = [TEMPERATUR_LINKS] + innere + [TEMPERATUR_RECHTS]

print("Element-Steifigkeit k = lambda*A/h = %.2f W/K  (h = %.3f m)" % (k, h))
print()
print("Knoten |   Ort   | Temperatur | exakte Gerade")
print("-------+---------+------------+--------------")
for i in range(KNOTENZAHL):
    gerade = TEMPERATUR_LINKS + (TEMPERATUR_RECHTS - TEMPERATUR_LINKS) * i / elementzahl
    print("  %2d   | %.3f m | %7.1f °C | %7.1f °C" % (i + 1, i * h, temperaturen[i], gera

# --- Abbildung ---
import matplotlib.pyplot as plt
orte = []
for i in range(KNOTENZAHL):
    orte.append(i * h)
```

```
fig, ax = plt.subplots(figsize=(8.0, 3.6))
ax.plot([0, TRAEGERLAENGE_M], [TEMPERATUR_LINKS, TEMPERATUR_RECHTS], "-",
        color="tab:orange", lw=2.0, label="exakte Gerade (Kapitel 6)")
ax.plot(orte, temperaturen, "o", color="tab:blue", ms=9,
        markeredgecolor="black", label="FEM-Knotenwerte")
ax.set_xlabel("Ort entlang des Trägers (m)")
ax.set_ylabel("Temperatur (°C)")
ax.set_title("Die Wärme-Federkette mit %d Knoten" % KNOTENZAHL)
ax.legend(loc="upper left")
plt.tight_layout()
plt.show()
```

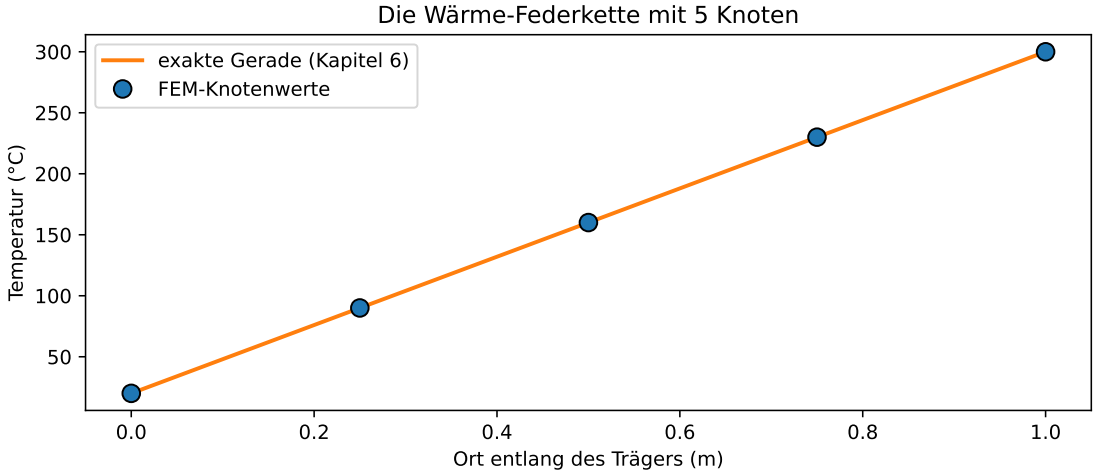

Abbildung 7.10.: Vorgerenderte Fassung der Wärme-Federkette mit 5 Knoten: die FEM-Knotenwerte (Punkte) liegen exakt auf der Geraden aus Kapitel 6. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst — stell KNOTENZAHL auf 20 und sieh, dass die Punkte auf derselben Geraden bleiben.

Was man sieht: Mit 5 Knoten liegen die Punkte auf der Geraden. Stellst du `KNOTENZAHL = 20`, ändert sich die Element-Steifigkeit k (sie wächst, weil h kleiner wird) und damit jede Zahl in der Matrix — aber die **Temperaturen an den Knoten bleiben exakt auf derselben Geraden.** Genau das war die Pointe des Vorhersage-Punkts: Verfeinern macht die Lösung hier nicht „genauer", denn sie war schon exakt. Fünf Knoten genügen für die Gerade; mehr Knoten braucht man erst, wenn die Kurve sich biegt.

7.8. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript liegt in `programme/kap07/kap07_waerme_fem`
Es ist der **Kern des späteren Finales**: dieselben vier Funktionen, die das ganze restliche Buch nur noch größer macht. `baue_elementmatrix` liefert den Steckbrief $\lambda A/h\,[[1,-1],[-1,1]]$, `assembliere` schiebt die Bausteine zusammen (mit dem Matrix-Drucker aus Kapitel 3 für die Bandstruktur), `baue_randwerte_ein` setzt die festen Temperaturen nach dem Streich-Rezept ein, und gelöst wird mit dem importierten Thomas-Löser aus 2. Das Skript rechnet das 5-Knoten-Handbeispiel vor, weist nach, dass Thomas und die volle Gauß-Elimination dasselbe liefern, und zeigt, dass durch jede Wärme-Feder dieselben 140 W fließen. Führe es mit `python kap07_waerme_fem.py` aus.

7.9. Rückblick auf die Landkarte

Erinnerst du dich an die **Landkarte** aus Kapitel 1 — die fünf Bausteine jeder FEM-Rechnung? In diesem Kapitel sind gleich mehrere Kästen mit Leben gefüllt worden. Die **Physik** (die Wärmeleitungsgleichung, Kapitel 6) ist über Ansatz, gewichtete Reste und Umverteilung in ein **Gleichungssystem** übersetzt worden; assembliert wie in Kapitel 3; die **Randbedingungen** sind (vorläufig) eingebaut; und der **Löser** aus 2 hat es geknackt.

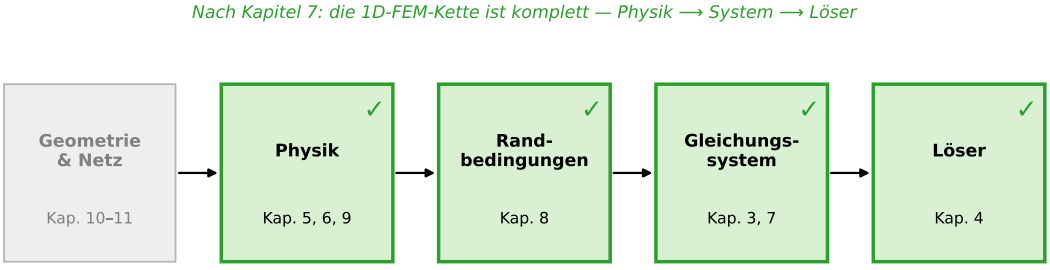

Abbildung 7.11.: Die Landkarte aus Kapitel 1, mit dem Stand nach Kapitel 7. Grün und mit Häkchen: die Bausteine, die jetzt gefüllt sind — Physik, Randbedingungen (vorläufig), Gleichungssystem und Löser. Noch offen (blass): Geometrie und Netz für die Fläche, das kommt in Teil IV. In der 1D-Welt ist die FEM-Kette damit komplett.

Was man hier sieht: Vier der fünf Kästen tragen jetzt ein Häkchen. In einer Dimension ist die FEM damit vollständig: Wir können die Physik eines Trägers in ein Gleichungssystem gießen und es lösen. Offen bleibt vor allem der erste Kasten — **Geometrie und Netz** —, sobald der Träger die Linie verlässt und in die *Fläche* geht. Dann werden aus den Hütchen auf der Linie **Zelte über Dreiecken** (1) — aber das Rezept bleibt: Steckbrief, assemblieren, Randwerte, lösen.

FDM: fast dieselbe Matrix, ein anderer Weg dorthin

Es gibt einen zweiten, älteren Weg, die Wärmeleitungsgleichung in ein Gleichungssystem zu verwandeln: die **Finite-Differenzen-Methode** (FDM). Sie geht *nicht* über Hütchen und gewichtete Reste, sondern ersetzt die Ableitungen direkt durch Steigungsdreiecke aus Kapitel 5 — die zweite Ableitung an einem Knoten als „Steigung rechts minus Steigung links, geteilt durch h“. Rechnet man das für unseren gleichmäßigen Träger aus, kommt für jeden inneren Knoten die Zeile $-T_{i-1} + 2\,T_i - T_{i+1} = 0$ heraus — **dasselbe Muster** wie unsere assemblierte Matrix (die $4 = 2 \cdot 2$ auf der Diagonale, die -2 daneben, alles durch k geteilt). Auf dem gleichmäßigen 1D-Gitter liefern FDM und FEM also fast Zahl für Zahl dasselbe. Der Unterschied zeigt sich erst, wenn die Ränder kniffliger werden (Kapitel 8) und wenn der Träger in die Fläche geht: Dort spielt die FEM ihre Stärke aus — beliebige Netze und Ränder, die sich „von selbst” ergeben. In 1D auf gleichem Gitter sind die beiden Methoden fast Zwillinge.

7.10. Die Namensschilder

Wir haben die ganze Methode gebaut, ohne ihre Fachwörter zu benutzen — mit Absicht. Jetzt, wo die Sache steht, kleben wir die Namensschilder an. Keiner dieser Begriffe ist mehr als ein Etikett für etwas, das du schon verstanden hast.

Die Fachwörter, jetzt nachgereicht

- **Ansatzfunktion** (auch *Formfunktion*): unser **Hütchen** N_i. Der „Ansatz” ist die Entscheidung, die Lösung als Summe $T(x) = \sum_j T_j N_j(x)$ anzusetzen — Knotenwerte mal Hütchen.
- **Schwache Form**: die Gleichung *nach* der Umverteilung, in der nur noch *einfache* Ableitungen vorkommen. „Schwach”, weil sie weniger verlangt als die ursprüngliche („starke”) Gleichung — statt

einer zweiten Ableitung überall nur noch ein gewichtetes Mittel. Genau deshalb kommt der eckige Streckenzug mit ihr zurecht.

- **Methode der gewichteten Reste**: die Idee, die Gleichung nicht überall, sondern nur im gewichteten Mittel zu erfüllen. Der „Rest" (oder das *Residuum*) ist der Fehlbetrag der Gleichung; wir zwingen ihn, im Mittel gegen jedes Gewicht null zu sein.
- **Testfunktion** (oder *Gewichtsfunktion*): das Gewicht, mit dem gemittelt wird. Weil wir dafür *dieselben* Hütchen nehmen wie für den Ansatz, trägt das Verfahren einen eigenen Namen: **Galerkin-Verfahren** (nach dem russischen Ingenieur Boris Galjorkin, 1871–1945, der es 1915 an Stäben und Platten entwickelte (Galerkin 1915)). „Ansatz und Test aus demselben Baukasten" — das ist die Galerkin-Wahl.
- **Partielle Integration**: der mathematische Name für unsere **Umverteilung** — die Regel, die eine Ableitung von einem Faktor auf den anderen schiebt und dabei den geparkten Randterm erzeugt.
- **Elementintegral / Elementmatrix**: das über *ein* Element gerechnete Integral $\int \lambda A\, N_r' N_c'\, \mathrm{d}x$ und die 2×2-Tabelle, die daraus entsteht — der Steckbrief, den wir in Schritt 5 von Hand ausgerechnet haben.

Aus der assemblierten Systemmatrix wird, ganz wie bei den Federn, die **Steifigkeitsmatrix** genannt — bei der Wärme auch **Leitfähigkeitsmatrix**. Alle diese Wörter stehen im Glossar; hier haben wir zuerst die Sache gebaut und dann das Schild geklebt.

Merkkasten

- Der **Ansatz** beschreibt die Temperatur als **Streckenzug**: fünf Knotenwerte, dazwischen Geraden. Aufgeschrieben mit **Hütchenfunktionen** — jedes Knotens Zuständigkeitsgebiet.
- Der Streckenzug hat **Knicke**, seine zweite Ableitung stirbt — man kann ihn nicht direkt in die Wärmeleitungsgleichung einsetzen.
- Ausweg: **gewichtete Reste** (Bilanz nur im Mittel, ein Gewicht = Hütchen pro Knoten) plus **Umverteilung** (partielle Integration), die T eine Ableitung abnimmt. Danach braucht man nur noch *eine* Steigung — die hat der Streckenzug.
- Die **Elementmatrix der Wärmeleitung** ist $\frac{\lambda A}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$ **— Zahl für Zahl die Federmatrix** aus Kapitel 3, mit $k = \lambda A/h = 2\ \mathrm{W/K}$.
- **Assemblieren, Randwerte einbauen, mit Thomas lösen** liefert $20/90/160/230/300\ °\mathrm{C}$ — die exakte Gerade aus Kapitel 6. Durch

jede Wärme-Feder fließen dieselben 140 W.
- Für den homogenen Träger ist die grobe Näherung **keine Näherung**: Weil die Wahrheit selbst eine Gerade ist, trifft die FEM sie an den Knoten exakt.

Roter Faden

Zurück: Dieses Kapitel war der Knoten, in dem alle Fäden zusammenliefen. Der **Ansatz** aus Geradenstücken ist die Streckenzug-Idee aus Kapitel 5; die **Steigungen** der Hütchen ($\pm 1/h$) und die **Rechteckflächen** der konstanten Integranden sind Schulgeometrie von dort. Die **Elementmatrix**, das **Assemblieren** und das **Streich-Rezept** für Randwerte sind die Federketten-Maschinerie aus Kapitel 3; gelöst hat der **Thomas-Algorithmus** aus 2. Und die **Bilanz** aus Kapitel 6 ist genau das, was die gewichteten Reste systematisch herleiten — die 140 W und die Gerade kehren Zahl für Zahl wieder.

Vor: Der **Randterm**, den wir geparkt haben, wird in Kapitel 8 abgeholt — dort entscheidet sich, was „fest gehalten", „beflammt" oder „isoliert" im Gleichungssystem bedeutet. Der **Zeitterm** (das Aufheizen aus der Animation in Kapitel 6) kommt in Kapitel 9 dazu und bringt die **Massenmatrix**, die nach derselben Halbe-Halbe-Logik gebaut wird wie die Wärmequelle in Übung 7.3. Die **Hütchen auf der Linie** werden in 1 zu **Zelten über Dreiecken**, wenn der Träger in die Fläche geht. Und die **Exaktheits-Frage** — wie groß der Fehler ist, wenn die Wahrheit krumm ist — steht im Zentrum von Kapitel 12.

Übungen

Ü 7.1 (Verstehen). Rechne die Elementmatrix für ein **Edelstahl-Element** von Hand aus: Länge $h = 0{,}5$ m, Wärmeleitfähigkeit $\lambda = 15\ \mathrm{W/(m\,K)}$, gleicher Querschnitt $A = 0{,}01\ \mathrm{m}^2$. Wie groß ist die Wärme-Steifigkeit $k = \lambda A/h$, und wie sieht die 2×2-Matrix aus? Vergleiche mit einer Baustahl-Feder des Trägers ($k = 2\ \mathrm{W/K}$): steifer oder weicher?

Musterlösung zu Ü 7.1

$$k = \frac{\lambda A}{h} = \frac{15 \cdot 0{,}01}{0{,}5} = \frac{0{,}15}{0{,}5} = 0{,}3\ \tfrac{\mathrm{W}}{\mathrm{K}}, \qquad \mathbf{K}_{\text{Element}} = 0{,}3 \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}\ \tfrac{\mathrm{W}}{\mathrm{K}}.$$

Mit $k = 0{,}3$ W/K ist diese Wärme-Feder **rund sechseinhalbmal weicher** als eine Baustahl-Feder des Trägers ($k = 2$). Zwei Gründe wirken zusammen: Edelstahl leitet gut dreimal schlechter als Baustahl ($\lambda = 15$ statt 50), und das doppelt so lange Element ($h = 0{,}5$ statt $0{,}25$ m) halbiert k noch einmal. Eine „weiche" Wärme-Feder trägt unter demselben Strom ein größeres Temperaturgefälle: Sie ist ein schlechter Leiter, genau wie es sein soll.

Ü 7.2 (Verändern). Ersetze im 5-Knoten-Träger die **beiden rechten Elemente** (zwischen den Knoten 3 und 5) — die dem Brandherd zugewandte Hälfte — durch **rostfreien Edelstahl** ($\lambda = 15$); die beiden linken Elemente bleiben Baustahl ($\lambda = 50$). Randwerte wie immer, 20 und 300 °C. **Sage zuerst vorher:** Wo im Profil entsteht ein Knick, und in welche Richtung? Assembliere dann von Hand oder mit dem Skript und deute das Ergebnis. Das Lösungsskript `loesungen/kap07_ue2.py` rechnet es vor.

i Musterlösung zu Ü 7.2

Die Wärme-Steifigkeiten sind jetzt $2,\ 2,\ 0{,}6,\ 0{,}6$ W/K (die beiden Edelstahl-Elemente haben $k = 15 \cdot 0{,}01/0{,}25 = 0{,}6$). Assembliert und gelöst ergibt sich das Profil

$$[\,20,\ 52{,}3,\ 84{,}6,\ 192{,}3,\ 300\,]\ °\text{C}.$$

Durch **alle** Elemente fließt derselbe Strom von rund $64{,}6$ W — es ist eine Reihenschaltung, genau wie bei den Federn. Die Edelstahl-Elemente leiten gut dreimal schlechter ($k = 0{,}6$ statt 2), tragen also unter demselben Strom je das gut dreifache Gefälle: über die kühlere Baustahl-Hälfte fallen zusammen nur $64{,}6$ K ab (20 → 84,6 °C), über die heiße Edelstahl-Hälfte dagegen $215{,}4$ K (84,6 → 300 °C). Genau an der Materialgrenze bei Knoten 3 **knickt** das Profil scharf nach oben. Das ist der Knick aus Übung 6.3 — dort noch von Hand über die Stromkonstanz erschlossen (Reihenwiderstand $4{,}33$ K/W, Übergangstemperatur 84,6 °C), hier von der FEM-Maschinerie ausgerechnet. (Dass der Strom hier nur 64,6 W beträgt und nicht die 140 W des reinen Baustahl-Trägers, liegt am zusätzlichen Widerstand des Edelstahls: Die Kette ist als Ganzes „weicher" geworden.)

Ü 7.3 (Übertragen). Bau eine **Wärmequelle** ein — einen Heizdraht, der über die ganze Länge gleichmäßig $P = 160$ W/m erzeugt. Auf ein Element der Länge $h = 0{,}25$ m entfallen damit $P \cdot h = 40$ W. Das **Rezept** für die rechte Seite: Verteile die Leistung jedes Elements *hälftig* auf seine beiden Knoten (je 20 W). Ein innerer Knoten sammelt so von seinen zwei Elemen-

ten 40 W ein; diese Knotenleistungen sind die rechte Seite (bisher war sie null). Löse und vergleiche mit der Geraden. Das Skript `loesungen/kap07_ue3.py` macht es vor.

Musterlösung zu Ü 7.3

Die rechte Seite ist jetzt $[\,20,\ 40,\ 40,\ 40,\ 20\,]$ W (die Randwerte 20/300 °C werden ohnehin festgehalten, ihre 20 W spielen keine Rolle). Assembliert und gelöst ergibt sich

$$[\,20,\ 120,\ 200,\ 260,\ 300\,]\ °\mathrm{C}.$$

Das ist keine Gerade mehr! Verglichen mit der quellfreien Geraden $[\,20, 90, 160, 230, 300\,]$ wölbt sich das Profil zu einer **Parabel** über die Gerade — in der Mitte am stärksten ($+40$ °C bei Knoten 3), zu den festgehaltenen Rändern hin auf null. So sieht es der Kasten in Kapitel 6 vorher: Eine Wärmequelle macht aus der Geraden eine gewölbte Kurve, weil die in der Mitte erzeugte Wärme nach beiden Seiten den weitesten Weg zum Rand hat und sich dort staut.

> **Warum genau hälftig auf die zwei Knoten?**
>
> Das Hütchen eines Knotens fällt über sein Element von 1 auf 0 — im Mittel „gehört" ihm also die *halbe* Elementlänge und damit die halbe darin erzeugte Leistung. Rechnet man das Elementintegral $\int P\,N_i\,\mathrm{d}x$ sauber aus (eine Dreiecksfläche aus Kapitel 5: halbe Höhe mal Breite), kommt exakt „die Hälfte an jeden Knoten" heraus. Genau dieselbe Halbe-Halbe-Logik trifft dich in Kapitel 9 wieder, wenn die **Massenmatrix** den Zeitterm auf die Knoten verteilt.

Das Kleingedruckte

Drei ehrliche Feinheiten zum Schluss.

Erstens: Wir haben die Umverteilung (partielle Integration) benutzt, aber nicht bewiesen. Sie ist eine Rechenregel der Analysis, die immer gilt; wir haben sie hier nur *angewandt* und an einem Elementpaar plausibel gemacht, statt sie herzuleiten. Wer die Regel kennt, erkennt sie wieder; wer nicht, darf ihr vertrauen — das Ergebnis (die Elementmatrix) haben wir mit reiner Schulgeometrie nachgerechnet, ganz ohne die Regel.

Zweitens: Der **Randterm** ist echt und wichtig, wir haben ihn nur für die inneren Knoten auf null gesetzt. Für die Randknoten trägt er die Information

über den ein- und austretenden Wärmestrom — und genau daraus werden in Kapitel 8 die verschiedenen Arten von Randbedingungen. Wir haben ihn also nicht unter den Teppich gekehrt, sondern für ein eigenes Kapitel aufgehoben.

Drittens: Die schöne **Exaktheit** dieses Kapitels — FEM trifft die Wahrheit auf die Nachkommastelle — ist der gutmütige Sonderfall, kein Naturgesetz. Sie gilt, weil die wahre Lösung selbst im Ansatzraum liegt (eine Gerade unter Geradenstücken). Sobald die Wahrheit krumm ist (Wärmequelle, Materialsprung, gebogene Geometrie), trifft die FEM nur noch *näherungsweise*, und die Näherung wird mit feinerem Netz besser. Diese ehrliche Fehler-Frage — „wie genau ist mein buntes Bild wirklich?" — ist so zentral, dass ihr Kapitel 12 gewidmet ist.

8. Randbedingungen

Zwei Simulationen desselben Trägers liegen nebeneinander auf dem Bildschirm. Gleicher Stahl, gleiche fünf Knoten, derselbe Brandherd, der von rechts unten Wärme in das Ende drückt — und doch zeigen sie völlig verschiedene Temperaturen. In der einen bleibt das linke Ende am kühlen Auflager auf 20 °C, und die Temperatur steigt zum Brandherd hin gleichmäßig bis auf 420 °C an. In der anderen zieht der ganze Träger auf annähernd dieselbe hohe Temperatur hoch, obwohl derselbe Brandherd brennt. Woran liegt das? An einer einzigen Kleinigkeit am *linken* Ende: einmal steckt es in der kühlen Außenwand, die es auf 20 °C hält, einmal ist es rundum brandschutzbeplankt und lässt keine Wärme mehr abfließen. Die Gleichung im Inneren war Zeichen für Zeichen dieselbe. **Der Rand hat entschieden.**

Genau hier liegt die stille Gefahr jeder Simulation. Was in der Mitte des Trägers passiert, haben wir in Kapitel 7 bis zur letzten Nachkommastelle verstanden. Aber das Gleichungssystem von dort war **noch nicht lösbar** — es fehlte die Information, was an den beiden *Enden* geschieht. Diese Information heißt **Randbedingung**, und wer sie falsch setzt, rechnet eine andere Welt aus, ohne es zu merken. Randbedingungen sind der häufigste Anwenderfehler in der ganzen Ingenieurspraxis. Dieses Kapitel macht sie zur bewussten Entscheidung — und löst dabei den Randterm ein, den Kapitel 7 ausdrücklich geparkt hat.

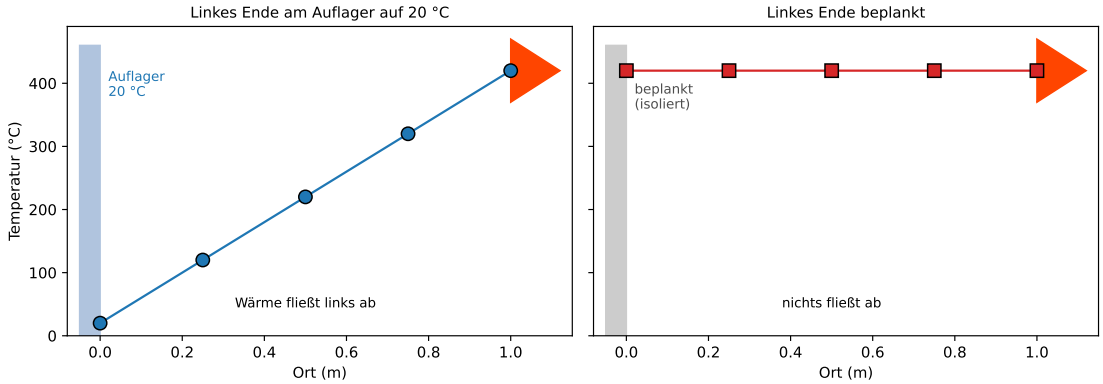

Abbildung 8.1.: Derselbe Träger, derselbe Brandherd rechts — nur der linke Rand ist verschieden. Links (blau) hält das kühle Auflager das linke Ende auf 20 °C, die Wärme fließt dort in die Außenwand ab, das Profil steigt gleichmäßig bis 420 °C an. Rechts (rot) ist das linke Ende beplankt und lässt nichts mehr durch, der Brandherd heizt einen rundum abgeschlossenen Träger — er läuft überall auf dieselbe hohe Temperatur zu (hier idealisiert flach). Zwei Welten aus einem einzigen Unterschied am Rand.

Was man hier sieht: links und rechts derselbe Träger mit demselben Brandherd — aber weil das linke Ende einmal Wärme ins kühle Auflager abführt und einmal nicht, stellt sich ein ganz anderes Profil ein. Die eine Simulation beschreibt einen Träger mit kühler Wärmesenke, die andere einen rundum eingepackten. Nur der Rand unterscheidet sie. (Die rechte Kurve ist hier idealisiert flach; wie es wirklich zugeht, klärt dieses Kapitel Schritt für Schritt.)

Lernziele

Nach diesem Kapitel kannst du …

1. … die drei Grundtypen benennen und physikalisch unterscheiden: Temperatur vorgeben (**Dirichlet**), Wärmestrom vorgeben (**Neumann**), Übergang an die Umgebung (**Robin**) — jeweils mit einem Alltagsbeispiel,
2. … eine vorgegebene Temperatur ins 5×5-System einbauen, und zwar auf **zwei** Wegen (Zeile ersetzen und Streichen), und begründen, warum beide dieselbe Lösung liefern,

3. … einen vorgegebenen Wärmestrom als **eine einzige Zahl auf der rechten Seite** einbauen und erklären, warum er *nicht* in die Matrix wandert,
4. … erklären, warum „gar nichts vorgeben" automatisch „perfekt beplankt" bedeutet — die berühmte **natürliche** Randbedingung,
5. … diagnostizieren, warum ein Träger ohne jede Temperaturvorgabe **unlösbar** ist (das System „schwimmt").

⚠ Naheliegende Vermutung

Vermutung: *„Wenn ich am rechten Rand nichts vorschreibe, lässt das Programm dort eben alles offen — der Träger kann sich am freien Ende verhalten, wie er will."*
Warum sie naheliegt: „Nichts vorgeben" klingt nach „keine Einschränkung", nach maximaler Freiheit. Man erwartet, dass ein Ende, über das man nichts gesagt hat, sich irgendwie neutral einstellt.
Was stattdessen stimmt: „Nichts vorschreiben" heißt in der FEM etwas sehr Bestimmtes — nämlich **Randterm = 0**, also *kein Wärmestrom* über diesen Rand, also **perfekt beplankt**. Das ist keine Freiheit, sondern eine knallharte Annahme: eine lückenlose Brandschutzbeplankung, durch die keine Wärme mehr entweicht. Die FEM hat diese Standardannahme fest eingebaut, und wer sie nicht kennt, dämmt sein Bauteil aus Versehen. Wir demonstrieren das in diesem Kapitel an echten Zahlen — der Träger läuft dann überall auf dieselbe Temperatur.

8.1. Drei Ränder, drei Geschichten

Bevor wir irgendeine Zahl anfassen, klären wir die Physik. Am Ende eines Trägers kann die Wirklichkeit dreierlei tun, und jede Möglichkeit hat einen Namen und ein Alltagsbild. Abbildung 8.2 stellt sie nebeneinander.

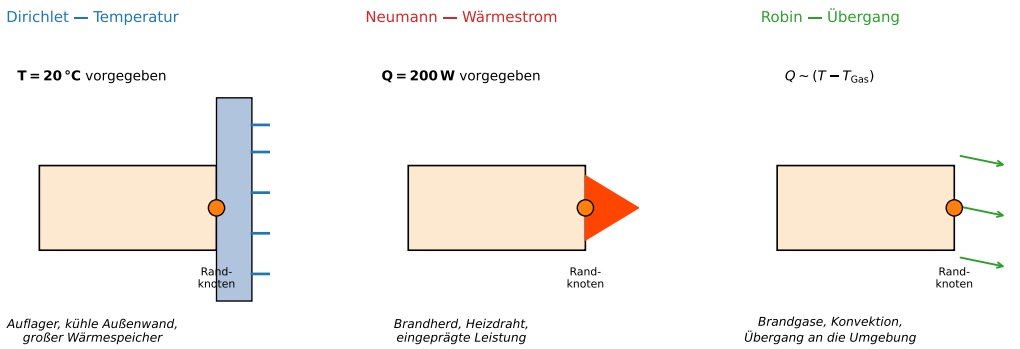

Abbildung 8.2.: Die drei Randtypen als Bildergeschichte. Links: Das kühle Auflager hält das Ende auf einer festen Temperatur, egal wie viel Wärme dafür fließen muss (Dirichlet). Mitte: Der Brandherd pumpt eine feste Leistung hinein, egal wie heiß das Ende schon ist (Neumann). Rechts: Vorbeiströmende Brandgase tauschen Wärme umso stärker aus, je größer der Temperaturunterschied zum Ende ist (Robin). Vorgegeben ist jeweils das Fettgedruckte.

Was man hier sieht: drei Enden, drei Vorgaben. Das **kühle Auflager** (links) erzwingt eine **Temperatur** und nimmt dafür so viel Wärme auf, wie nötig — die massive Außenwand ist so groß, dass sie nicht merklich wärmer wird. Der **Brandherd** (Mitte) erzwingt einen **Wärmestrom** — er gibt seine Leistung ab, ganz gleich, wie heiß das Ende schon ist. Die **Brandgase** (rechts) sind ein Zwitter: Sie tauschen Wärme aus, aber *umso stärker, je größer der Unterschied* zwischen Gas und Ende ist — vorgegeben ist hier weder Temperatur noch Strom, sondern der **Zusammenhang** zwischen beiden.

Die ersten beiden Typen sind das Handwerkszeug dieses Kapitels; der dritte ist wichtig genug für einen eigenen Kasten, kommt aber erst in Kapitel 14 und 3 richtig zum Einsatz.

Der dritte Typ: Robin, der realistische Übergang

Der **Robin-Rand** (auch *Übergangs-* oder *Konvektionsrandbedingung*) beschreibt, was in der Wirklichkeit am häufigsten passiert: Ein Bauteil steht in Gas anderer Temperatur, und über die Oberfläche tauschen beide Wärme aus — **proportional zur Temperaturdifferenz**. Sind die Brandgase $600\,°\mathrm{C}$ heiß und das Ende erst $200\,°\mathrm{C}$, treibt die Differenz von $400\,\mathrm{K}$ den Wärmestrom hinein; hat das Ende schon $550\,°\mathrm{C}$ erreicht, ist die Differenz kleiner und der Strom schwächer. In Formel:

der übertragene Strom ist $Q = \alpha_{\mathrm{c}}\, A\,(T_{\mathrm{Gas}} - T)$, mit dem **Wärmeübergangskoeffizienten** α_{c}; für Brandgase rechnet die Norm mit rund $\alpha_{\mathrm{c}} = 25\,\mathrm{W/(m^2\,K)}$.
Robin ist eine **Mischung** aus den beiden Grundtypen: Für sehr großes α_{c} (perfekter Kontakt) presst er das Ende auf T_{Gas} — das ist Dirichlet. Für $\alpha_{\mathrm{c}} = 0$ (kein Übergang) fließt gar nichts — das ist der isolierte Neumann-Rand mit Strom null. Alle realistischen Übergänge liegen dazwischen. Weil Robin einen Anteil enthält, der *von der gesuchten Temperatur selbst* abhängt, wandert er — anders als der reine Brandherd — teils in die **Matrix** und teils auf die rechte Seite. Das rechnen wir in 3 aus, wenn aus dem idealen Brandherd die echten Brandgase werden.

8.2. Der geparkte Randterm — jetzt abgeholt

Erinnere dich an die Umverteilung aus Kapitel 7 (die partielle Integration). Beim Umsortieren des Integrals blieb ein Ausdruck übrig, den wir ausdrücklich zur Seite gelegt haben — der **Randterm**:

$$\Big[\, N_i\, \lambda A\, \frac{\mathrm{d}T}{\mathrm{d}x} \,\Big]_0^L .$$

Für die *inneren* Knoten war er null, weil ihr Hütchen N_i an beiden Trägerenden null ist — deshalb durften wir ihn in Kapitel 7 getrost parken. Jetzt holen wir ihn für die **Randknoten** ab und sehen genau hin, was er bedeutet. Der Faktor $-\lambda A\,\frac{\mathrm{d}T}{\mathrm{d}x}$ ist nach dem Fourierschen Gesetz aus Kapitel 6 nichts anderes als der **Wärmestrom** Q, der an dieser Stelle fließt. Der Randterm ist also, Wort für Wort:

Der Wärmestrom, der am Ende des Trägers ein- oder austritt.

Das ist der ganze Zauber. Der Term, den die Mathematik beim Umsortieren wie von selbst ausgespuckt hat, ist genau die Stelle, an der die **Flüsse am Rand** ins System eintreten. Man muss ihn nicht künstlich hinzufügen — er **steht schon da**, in der schwachen Form, seit Kapitel 7. Darum heißen Randbedingungen, die einen **Fluss** vorgeben, in der FEM **natürliche Randbedingungen**: Sie ergeben sich natürlich aus der Herleitung, ganz ohne Extra-Handgriff.

Und der Sonderfall, der die Vermutung vom Kapitelanfang auflöst, liegt jetzt offen zutage: Setzt man den Randterm auf **null**, sagt man „am Rand fließt kein Wärmestrom" — das Ende ist **perfekt isoliert**. Wer also an einem Rand

gar nichts tut, lässt den Term auf seinem geparkten Wert null stehen — und dämmt das Ende, ohne es zu wollen. Genau das rechnen wir gleich nach.

Merke: die zwei Familien

- **Fluss vorgeben (Neumann):** Der Randterm bekommt einen Wert. Er sitzt auf der **rechten Seite** des Gleichungssystems — bei dem Knoten, an dem der Strom eintritt. „Natürliche" Randbedingung, weil schon in der schwachen Form enthalten. Nichts tun = Fluss null = isoliert.
- **Temperatur vorgeben (Dirichlet):** Der Knotenwert selbst wird festgeschrieben. Das ist ein **Eingriff in die Unbekannten** und muss von Hand ins System eingebaut werden (Streichen oder Zeile ersetzen). „Wesentliche" Randbedingung, weil man sie der schwachen Form *aufzwingen* muss.

8.3. Neumann konkret: der Brandherd mit 200 Watt

Fangen wir mit der natürlichen, der einfacheren Familie an. Der Träger steckt links im kühlen Auflager auf $20\,^{\circ}\mathrm{C}$ (das ist Dirichlet, dazu gleich), rechts drückt der Brandherd **200 Watt** in den Randknoten 5. Wie baut man diese 200 W ein?

Die Antwort ist verblüffend schlicht: Der eingespeiste Strom ist der Randterm, und der Randterm sitzt auf der rechten Seite bei seinem Knoten. Also schreiben wir eine **einzige Zahl** in die rechte Seite — die 200 bei Knoten 5 — und rühren die Matrix **nicht an**. Abbildung 8.3 zeigt das System davor und danach.

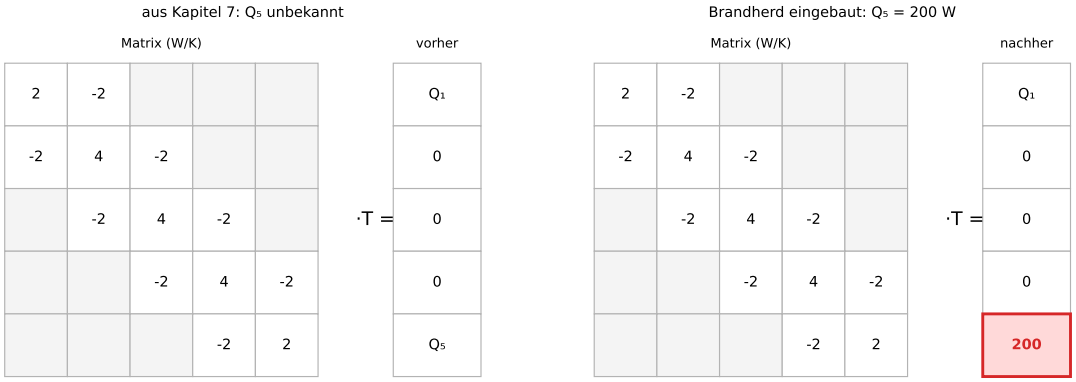

Abbildung 8.3.: Neumann einbauen heißt: eine Zahl auf der rechten Seite. Links das assemblierte System aus Kapitel 7 mit der unbekannten Randleistung Q_5 bei Knoten 5. Rechts nach dem Einbau des Brandherds: aus dem unbekannten Q_5 wird die vorgegebene 200 W (rot). Die Systemmatrix (die 2/4/4/4/2-Bandstruktur links) bleibt Zeichen für Zeichen unverändert — der Fluss wandert nicht in die Matrix.

Was man hier sieht: Der einzige Unterschied zwischen „vorher" und „nachher" ist eine Zahl in der letzten Zeile der rechten Seite — aus dem unbekannten Q_5 wird die vorgegebene 200. Die Matrix ist beide Male identisch. Genau das meint „der Fluss wandert nicht in die Matrix": Eine Neumann-Randbedingung ist ein Eintrag auf der rechten Seite, sonst nichts.

Bleibt der linke Rand. Er wird auf $20\,°\mathrm{C}$ gehalten — das ist eine **Dirichlet**-Bedingung, die wir gleich im Detail behandeln; hier bauen wir sie schon einmal mit dem Streich-Rezept aus Kapitel 7 ein (Knoten 1 streichen, seinen Wert auf die rechte Seite schieben). Übrig bleibt ein 4×4-System für die Knoten 2 bis 5. Lassen wir das Programm es hinschreiben und lösen:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "kap08"))
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from kap08_randbedingungen import (baue_system, baue_rand_ein,
                                   loese_mit_raendern)

# Träger mit 5 Knoten, leeres System.
matrix, rechte_seite, feste_temperaturen, k = baue_system(5)

# Links 20 °C halten (Dirichlet), rechts Brandherd mit 200 W (Neumann).
```

```
baue_rand_ein(rechte_seite, feste_temperaturen, "temperatur", 20.0, "links")
baue_rand_ein(rechte_seite, feste_temperaturen, "strom", 200.0, "rechts")

temperaturen, lage = loese_mit_raendern(
    matrix, rechte_seite, feste_temperaturen, [0.0, 0.0, 0.0, 0.0, 0.0])

print("Rechte Seite (Watt je Knoten):", rechte_seite)
print("Lage der Lösung:", lage)
print()
print("Knoten |   Ort  | Temperatur | Sprung zum Vorgänger")
print("-------+--------+------------+---------------------")
vorher = temperaturen[0]
for i in range(5):
    sprung = temperaturen[i] - vorher
    print("  %d    | %.2f m |  %6.1f °C |   %+6.1f K"
          % (i + 1, i * 0.25, temperaturen[i], sprung))
    vorher = temperaturen[i]
```

```
Rechte Seite (Watt je Knoten): [0.0, 0.0, 0.0, 0.0, 200.0]
Lage der Lösung: eindeutig

Knoten |   Ort  | Temperatur | Sprung zum Vorgänger
-------+--------+------------+---------------------
  1    | 0.00 m |    20.0 °C |     +0.0 K
  2    | 0.25 m |   120.0 °C |   +100.0 K
  3    | 0.50 m |   220.0 °C |   +100.0 K
  4    | 0.75 m |   320.0 °C |   +100.0 K
  5    | 1.00 m |   420.0 °C |   +100.0 K
```

Interpretation der Ausgabe: Die Lösung ist $20/120/220/320/420\,°\mathrm{C}$ — und der Sprung von Knoten zu Knoten ist **jedes Mal dieselbe** $100\,\mathrm{K}$. Das ist kein Zufall, sondern reine Physik: Weil im Träger keine Wärme erzeugt oder verschluckt wird, fließt durch **jedes** Element derselbe Strom von 200 W (rein beim Brandherd rechts, raus ins Auflager links). Und derselbe Strom durch dieselbe Wärme-Feder $k = 2\,\mathrm{W/K}$ macht denselben Temperatursprung: $\Delta T = Q/k = 200/2 = 100\,\mathrm{K}$ pro Element. Fünf Knoten, vier gleiche Stufen — das Profil ist wieder eine Gerade, nur dass diesmal der **Brandherd** ihre Steigung bestimmt, nicht ein zweites Auflager.

Vorhersage-Punkt

Bevor du weiterliest: Das Auflager links hält 20 °C, der Brandherd rechts speist 200 W ein. Wie groß ist der Wärmestrom, den das Auflager links aufnehmen muss, damit der Träger im eingeschwungenen Zustand bleibt? Und in welche Richtung fließt er? Leg dich fest, bevor du weiterliest.

Die Antwort steckt in der Bilanz: Was rechts hineinfließt, muss links wieder heraus, sonst würde sich der Träger immer weiter aufheizen. Das Auflager nimmt also genau die **200 W** wieder auf, die der Brandherd einspeist — es ist die Senke, die den stationären Zustand überhaupt erst möglich macht. Diese verborgene Rolle des gehaltenen Rands ist gleich der Schlüssel zum „schwimmenden Träger".

8.4. Dirichlet konkret: die Temperatur festhalten

Jetzt die wesentliche Randbedingung. Das Auflager presst das linke Ende auf $T_1 = 20\,°\mathrm{C}$ — eine **feste Temperatur**, eine der gesuchten Unbekannten wird schlicht vorgeschrieben. Anders als der Brandherd lässt sich das nicht durch einen Eintrag auf der rechten Seite erledigen; wir müssen in die Gleichungen eingreifen. Dafür gibt es zwei Rezepte, und sie führen beweisbar zum selben Ergebnis.

Rezept A — die Zeile ersetzen. Knoten 1 hat eine eigene Zeile im 5×5-System (seine Wärmestrom-Bilanz). Aber diese Bilanz interessiert uns gar nicht — wir *wissen* ja, dass $T_1 = 20$ ist. Also werfen wir die ganze Zeile weg und schreiben an ihre Stelle die denkbar einfachste Gleichung, die genau das sagt:

$$1 \cdot T_1 + 0 \cdot T_2 + 0 \cdot T_3 + 0 \cdot T_4 + 0 \cdot T_5 = 20.$$

Die erste Zeile der Matrix wird also zu $[\,1\;0\;0\;0\;0\,]$, die erste Zahl der rechten Seite zu 20. Der Rest des Systems bleibt stehen. Löst man es, kommt $T_1 = 20$ heraus (das erzwingt die neue Zeile), und die übrigen Knoten ordnen sich darum herum ein.

Rezept B — streichen. Das ist das Rezept aus Kapitel 3 und Kapitel 7. Weil T_1 bekannt ist, brauchen wir seine Zeile nicht (sie ergäbe nur die Auflagerleistung Q_1, die uns nicht interessiert), und in allen anderen Zeilen ist der Term $-2\,T_1$ eine **bekannte** Zahl, die auf die rechte Seite wandert. Wir **streichen** Zeile und Spalte 1 und lösen nur noch für die vier freien Knoten. Aus Zeile 2 wird zum Beispiel

$$-2 \cdot \underbrace{20}_{T_1} + 4\,T_2 - 2\,T_3 = 0 \quad \Longrightarrow \quad 4\,T_2 - 2\,T_3 = 40.$$

Abbildung 8.4 stellt beide Rezepte nebeneinander.

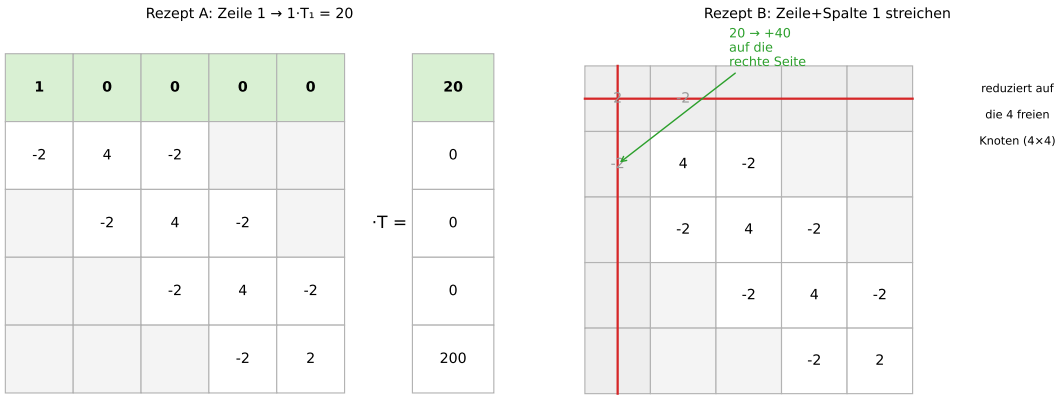

Abbildung 8.4.: Zwei Rezepte für dieselbe feste Temperatur $T_1 = 20$ °C. Links Rezept A: Die erste Zeile wird durch die triviale Gleichung $1 \cdot T_1 = 20$ ersetzt (grün), der Rest bleibt. Rechts Rezept B: Zeile und Spalte 1 werden gestrichen (grau durchgestrichen), der bekannte Wert 20 wandert als $-(-2) \cdot 20 = 40$ auf die rechte Seite (grün). Rezept A behält die 5×5-Größe, Rezept B schrumpft auf 4×4 — aber die Temperaturen, die herauskommen, sind Zahl für Zahl identisch.

Was man hier sieht: Zwei Wege, dieselbe Vorgabe einzubauen. Rezept A lässt das System 5×5 groß und opfert eine Zeile an die triviale Gleichung; Rezept B macht es kleiner, indem es den bekannten Knoten ganz herausnimmt und seinen Beitrag nach rechts schiebt. Beweisen wir, dass beide dieselbe Lösung liefern — nicht durch Zureden, sondern durch Nachrechnen:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from loeser import gauss_elimination


def K5():
    """Die assemblierte 5x5-Systemmatrix des Trägers (k = 2 W/K)."""
    k = 2.0
    matrix = []
```

```
    for zeile in range(5):
        matrix.append([0.0, 0.0, 0.0, 0.0, 0.0])
    for element in range(4):
        links = element
        rechts = element + 1
        matrix[links][links] = matrix[links][links] + k
        matrix[links][rechts] = matrix[links][rechts] - k
        matrix[rechts][links] = matrix[rechts][links] - k
        matrix[rechts][rechts] = matrix[rechts][rechts] + k
    return matrix


# Rezept A: Zeile 0 durch 1*T1 = 20 ersetzen, volles 5x5 lösen.
matrix_a = K5()
seite_a = [0.0, 0.0, 0.0, 0.0, 200.0]
matrix_a[0] = [1.0, 0.0, 0.0, 0.0, 0.0]
seite_a[0] = 20.0
loesung_a = gauss_elimination(matrix_a, seite_a)

# Rezept B: Zeile+Spalte 0 streichen, 4x4 lösen, dann T1 = 20 anfügen.
matrix = K5()
seite = [0.0, 0.0, 0.0, 0.0, 200.0]
frei = [1, 2, 3, 4]
kleine_matrix = []
kleine_seite = []
for i in frei:
    neue_zeile = []
    for j in frei:
        neue_zeile.append(matrix[i][j])
    kleine_matrix.append(neue_zeile)
    beitrag = matrix[i][0] * 20.0     # der bekannte Randwert wandert nach rechts
    kleine_seite.append(seite[i] - beitrag)
innere = gauss_elimination(kleine_matrix, kleine_seite)
loesung_b = [20.0] + innere

zeile_a = ""
zeile_b = ""
for i in range(5):
    zeile_a = zeile_a + ("%7.1f" % loesung_a[i])
    zeile_b = zeile_b + ("%7.1f" % loesung_b[i])
print("Rezept A (Zeile ersetzen):", zeile_a, "°C")
print("Rezept B (streichen):     ", zeile_b, "°C")
gleich = True
```

```
for a, b in zip(loesung_a, loesung_b):
    if abs(a - b) > 1e-9:
        gleich = False
print("Beide identisch:", gleich)
```

```
Rezept A (Zeile ersetzen):    20.0  120.0  220.0  320.0  420.0 °C
Rezept B (streichen):         20.0  120.0  220.0  320.0  420.0 °C
Beide identisch: True
```

Interpretation der Ausgabe: Zahl für Zahl dasselbe Profil — $20/120/220/320/420\,°\mathrm{C}$. Rezept A und Rezept B sind zwei Schreibweisen desselben Eingriffs: Beide sagen „T_1 ist 20", nur einmal als zusätzliche Zeile im großen System, einmal durch Herausnehmen des Knotens. Welches man nimmt, ist Geschmack (und Bequemlichkeit des Programmierers) — nie eine Frage der Physik.

> **Was Profi-Programme tun: der Strafterm**
>
> Große FEM-Pakete streichen ungern Zeilen (das verschiebt alle Indizes und zerstört die schöne Bandstruktur). Sie benutzen oft einen dritten Trick, den **Strafterm** (englisch *penalty*): Man addiert auf das Diagonalfeld des festen Knotens eine **riesige** Zahl P (etwa 10^{12}) und auf die rechte Seite $P \cdot 20$. Dann lautet die Zeile ungefähr $P \cdot T_1 \approx P \cdot 20$, und weil P alles andere in der Zeile erdrückt, kommt praktisch $T_1 = 20$ heraus — ohne dass man die Zeile wirklich entfernt. Es ist ein „so-tun-als-ob-Auflager", so steif, dass der Knoten quasi festgeklebt ist. Der Preis ist ein winziger Fehler (das Auflager ist nicht *unendlich* steif) und eine schlechter konditionierte Matrix. Wir bleiben in diesem Buch beim sauberen Streichen; den Strafterm musst du nur wiedererkennen, wenn du ihn in fremdem Code siehst.

8.5. Wenn man nichts vorgibt: die Isolierung von selbst

Jetzt lösen wir die Vermutung vom Kapitelanfang auf. Wir halten den linken Rand auf $20\,°\mathrm{C}$, es gibt **keine** Wärmequelle im Träger — und am rechten Rand geben wir **nichts** vor. Keine Temperatur, keinen Brandherd, nichts. Nach der Vermutung müsste das rechte Ende „frei" sein und irgendeinen Wert annehmen. Schauen wir nach:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "kap08"))
from kap08_randbedingungen import (baue_system, baue_rand_ein,
                                   loese_mit_raendern)

matrix, rechte_seite, feste_temperaturen, k = baue_system(5)

# Links 20 °C halten. Rechts: NICHTS vorgeben (isoliert).
baue_rand_ein(rechte_seite, feste_temperaturen, "temperatur", 20.0, "links")
baue_rand_ein(rechte_seite, feste_temperaturen, "isoliert", 0.0, "rechts")

temperaturen, lage = loese_mit_raendern(
    matrix, rechte_seite, feste_temperaturen, [0.0, 0.0, 0.0, 0.0, 0.0])

zeile = ""
for t in temperaturen:
    zeile = zeile + ("%6.1f °C" % t)
print("Rechte Seite (Watt je Knoten):", rechte_seite)
print("Temperaturen:", zeile)
```

```
Rechte Seite (Watt je Knoten): [0.0, 0.0, 0.0, 0.0, 0.0]
Temperaturen:   20.0 °C  20.0 °C  20.0 °C  20.0 °C  20.0 °C
```

Interpretation der Ausgabe: $20/20/20/20/20\,°C$ — der ganze Träger steht auf 20 °C. Kein „freies" Ende, das sich beliebig einstellt, sondern eine kerzengerade, flache Linie. Das ist die **natürliche Randbedingung** in Aktion: „Rechts nichts vorgeben" hieß „Randterm null" hieß „kein Wärmestrom über das rechte Ende". Und wenn rechts kein Strom austritt und es keine Quelle gibt, dann kann durch den ganzen Träger kein Strom fließen (in Reihe geschaltet trägt jedes Element denselben Strom, und der ist am rechten Ende null). Ohne Strom kein Temperaturgefälle — also überall dieselbe Temperatur wie am gehaltenen Ende: 20 °C.

Der Träger ist rechts **perfekt beplankt**, ohne dass wir das Wort „isoliert" je getippt hätten. Es war die Voreinstellung. Genau davor warnt die Vermutung: Wer ein Ende „vergisst", baut keine offene Grenze, sondern eine lückenlose Beplankung — und wundert sich, warum die Wärme sich staut.

8.6. Der schwimmende Träger

Bisher hatten wir immer *mindestens einen* gehaltenen Rand — das Auflager auf 20 °C. Was passiert, wenn wir es weglassen? Nehmen wir den Brandherd von vorhin (200 W rechts hinein) und ersetzen das Auflager links durch einen zweiten Fluss: 200 W treten links wieder aus (in eine Wärmesenke, die diesmal aber keine feste Temperatur erzwingt, sondern nur den Strom abnimmt). Rein gleich raus, die Bilanz stimmt — und doch geht etwas schief.

Vorhersage-Punkt

Bevor du das Programm laufen lässt: Wir geben nur noch Flüsse vor (200 W rein, 200 W raus), keine einzige Temperatur. Die Wärmemenge stimmt, es fließt so viel heraus wie hinein. Glaubst du, das Programm findet **eine** eindeutige Temperaturverteilung — oder fehlt ihm etwas? Und wenn ja: was? Leg dich fest.

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "kap08"))
from kap08_randbedingungen import (baue_system, baue_rand_ein,
                                   loese_mit_raendern)
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from loeser import gauss_seidel

matrix, rechte_seite, feste_temperaturen, k = baue_system(5)

# Nur Flüsse: 200 W rein rechts, 200 W raus links. KEINE feste Temperatur.
baue_rand_ein(rechte_seite, feste_temperaturen, "strom", -200.0, "links")
baue_rand_ein(rechte_seite, feste_temperaturen, "strom", 200.0, "rechts")

# Zwei Gauß-Seidel-Läufe mit verschiedenen Startprofilen.
aus_null = gauss_seidel(matrix, rechte_seite, [0.0, 0.0, 0.0, 0.0, 0.0],
                        1e-8, 400)
aus_hundert = gauss_seidel(matrix, rechte_seite,
                           [100.0, 100.0, 100.0, 100.0, 100.0], 1e-8, 400)

zeile_null = ""
zeile_hundert = ""
for i in range(5):
    zeile_null = zeile_null + ("%7.1f" % aus_null[i])
    zeile_hundert = zeile_hundert + ("%7.1f" % aus_hundert[i])
print("Start bei 0:   ", zeile_null)
```

```
print("Start bei 100: ", zeile_hundert)
print()
sprung_null = ""
sprung_hundert = ""
for i in range(4):
    sprung_null = sprung_null + ("%7.1f" % (aus_null[i + 1] - aus_null[i]))
    sprung_hundert = sprung_hundert + ("%7.1f" % (aus_hundert[i + 1] - aus_hundert[i]))
print("Sprung je Element, Start 0:  ", sprung_null)
print("Sprung je Element, Start 100:", sprung_hundert)
```

```
Start bei 0:     -250.0 -150.0  -50.0   50.0  150.0
Start bei 100:   -150.0  -50.0   50.0  150.0  250.0

Sprung je Element, Start 0:     100.0  100.0  100.0  100.0
Sprung je Element, Start 100:   100.0  100.0  100.0  100.0
```

Interpretation der Ausgabe: Zwei Läufe, zwei **verschiedene** Profile — aber sieh auf die Sprünge: Sie sind in beiden Läufen identisch $100\,\mathrm{K}$ pro Element. Die **Form** steht fest (dieselbe Steigung wie beim Neumann-Träger, denn derselbe Brandherd treibt dieselben 200 W durch dieselben Federn), aber die **Höhe** ist völlig willkürlich — sie hängt allein davon ab, wo Gauß-Seidel zu raten begonnen hat. Das System hat nicht *eine* Lösung, sondern **unendlich viele**, die sich nur um eine additive Konstante unterscheiden.

Physikalisch ist das sonnenklar: Wir haben gesagt, *wie steil* das Profil ist (die Flüsse bestimmen die Steigung), aber niemand hat gesagt, *wo* es liegt — kein einziger Punkt ist auf einen Temperaturwert festgenagelt. Ist das linke Ende bei 20 °C oder bei 120 °C? Beide Profile erfüllen jede Gleichung und jede Bilanz gleich gut. Der Träger **schwimmt**: Man kann ihn als Ganzes nach oben oder unten schieben, ohne dass sich an den Wärmeströmen etwas ändert. Abbildung 8.5 zeigt drei dieser gleichberechtigten Lösungen.

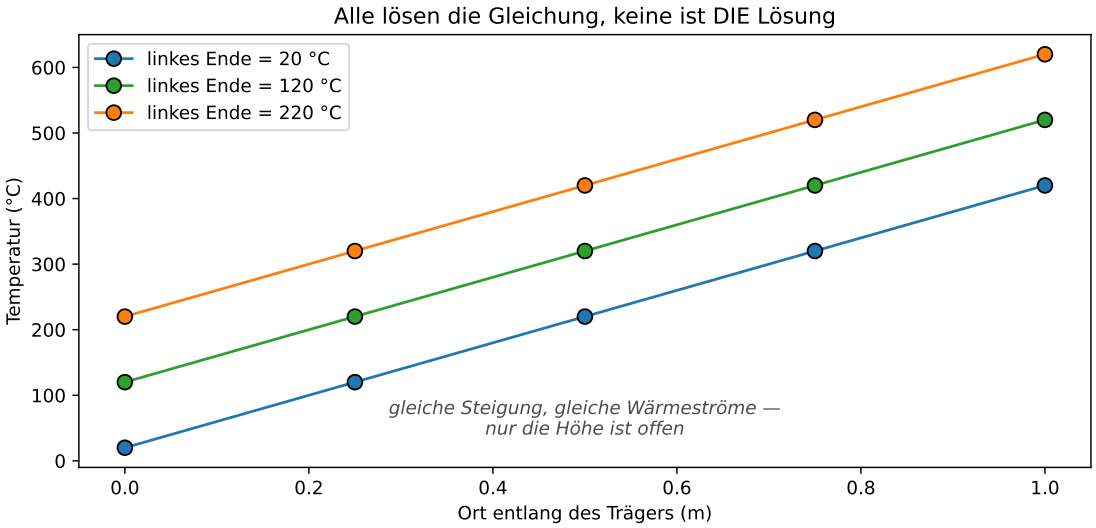

Abbildung 8.5.: Der schwimmende Träger: dieselbe Profilform auf drei Höhen. Nur Flüsse sind vorgegeben (200 W rein rechts, 200 W raus links), keine Temperatur — deshalb steht die Steigung (100 K pro Element) fest, aber die Lage ist frei. Alle drei Geraden lösen das Gleichungssystem exakt und tragen denselben Wärmestrom; keine von ihnen ist DIE Lösung. Erst eine einzige vorgegebene Temperatur nagelt eine davon fest.

Was man hier sieht: drei parallele Profile, jedes eine gültige Lösung. Sie unterscheiden sich nur um eine konstante Verschiebung nach oben. Genau diese Vieldeutigkeit meint der Satz „das System ist nur bis auf eine Konstante bestimmt". Der Ausweg ist so klein wie entscheidend: **Eine einzige Dirichlet-Bedingung** — irgendein Punkt auf einen festen Wert — sticht eine der Geraden heraus und macht die Lösung eindeutig. Das Auflager aus Kapitel 7 tat genau das.

Und es gibt einen noch schlimmeren Verwandten, den man ehrlich benennen muss. Bisher stimmte wenigstens die Bilanz (200 rein, 200 raus). Was, wenn beide Enden Wärme **hineinpumpen** — links 200 W *und* rechts 200 W, ohne Senke? Dann sammelt sich Wärme im Träger an, ohne je abzufließen; im eingeschwungenen Zustand müsste die Temperatur unendlich hoch stehen. Es gibt **gar keine** stationäre Lösung mehr, und Gauß-Seidel zeigt das ehrlich, indem es Runde um Runde davontreibt:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "kap08"))
from kap08_randbedingungen import baue_system, baue_rand_ein
```

```
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from loeser import gauss_seidel

matrix, rechte_seite, feste_temperaturen, k = baue_system(5)

# Beidseitig 200 W HINEIN, keine Senke -> keine stationäre Lösung.
baue_rand_ein(rechte_seite, feste_temperaturen, "strom", 200.0, "links")
baue_rand_ein(rechte_seite, feste_temperaturen, "strom", 200.0, "rechts")

print("Netto zugeführte Leistung:", sum(rechte_seite), "W (sollte 0 sein!)")
print()
print("Runden | Temperatur an Knoten 3")
print("-------+-----------------------")
for runden in (5, 20, 100, 400):
    naeherung = gauss_seidel(matrix, rechte_seite,
                             [0.0, 0.0, 0.0, 0.0, 0.0], 0.0, runden)
    print("  %3d  |  %8.0f °C" % (runden, naeherung[2]))
```

```
Netto zugeführte Leistung: 400.0 W (sollte 0 sein!)

Runden | Temperatur an Knoten 3
-------+-----------------------
    5  |       200 °C
   20  |       950 °C
  100  |      4950 °C
  400  |     19950 °C
```

Interpretation der Ausgabe: Die Temperatur wächst mit jeder Runde weiter, ohne sich je einzupendeln — 400 W fließen hinein und nichts heraus, also gibt es keinen Zustand, in dem sich nichts mehr ändert. Das ist der Unterschied zwischen „bis auf eine Konstante offen" (Bilanz stimmt, nur die Höhe fehlt) und „gar nicht lösbar" (die Bilanz stimmt nicht einmal). Beide Male ist die Ursache dieselbe: **zu wenig festgelegt am Rand.** Merke dir das Bild vom schwimmenden Träger — es kehrt in Kapitel 14 wieder, wenn ein mechanischer Träger *ohne Lagerung* auf genau dieselbe Weise davonschwebt (Starrkörpermoden).

8.7. Umschalten: das Profil klappt um

Die drei Randtypen am rechten Ende erzeugen drei ganz verschiedene Profile — bei linkem Auflager auf 20 °C. Halten wir sie nebeneinander: rechts

gehalten auf 300 °C ergibt die vertraute Gerade auf 300; rechts **beflammt** mit 200 W ergibt die steilere Gerade hinauf auf 420; rechts **isoliert** ergibt die waagerechte Linie auf 20. Die folgende Darstellung schaltet zwischen den dreien um — sieh zu, wie das Profil jedes Mal umklappt.

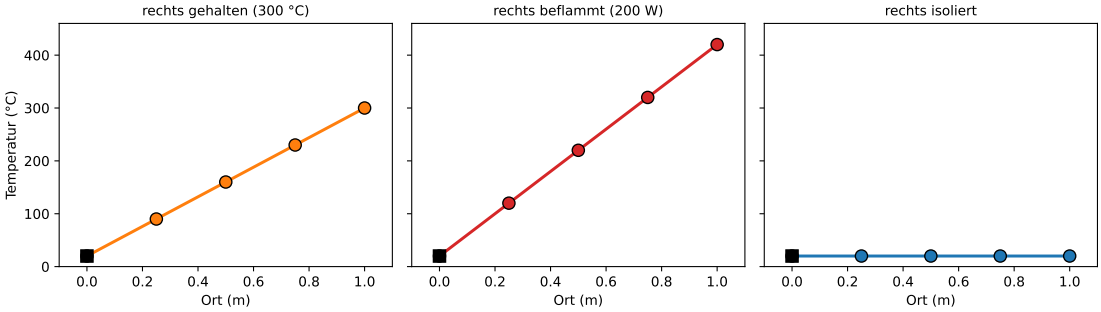

Abbildung 8.6.: Umschalten der rechten Randbedingung als Bildreihe (in der HTML-Fassung eine Animation): links immer das Auflager auf 20 °C. Rechts gehalten (300 °C) gibt die mittlere Gerade, rechts beflammt (200 W) die steilere hinauf auf 420, rechts isoliert die waagerechte Linie. Derselbe Träger, drei Ränder, drei Profile.

Was man sieht: Drei stationäre Zustände, erzeugt durch nichts als den Wechsel der rechten Randbedingung. Der gehaltene Rand zwingt das Ende auf 300 °C; der Brandherd lässt es auf den Wert steigen, der gerade 200 W abführt (420 °C, sogar höher als der gehaltene Rand); die Beplankung lässt keinerlei Wärme durch, und der Träger gleicht sich vollständig dem gehaltenen linken Ende an. Wer die Randbedingung wechselt, wechselt das Ergebnis — dramatischer als jede Änderung im Inneren.

8.8. Die interaktive Einheit: der Rand-Schalter

Jetzt bist du am Zug. Das folgende Programm ist der **Rand-Schalter**: Du wählst für das linke *und* das rechte Ende je einen Randtyp (`"temperatur"`, `"strom"` oder `"isoliert"`) und einen Zahlenwert. Das Programm baut die Randbedingungen ins 5-Knoten-System ein, druckt die geänderte rechte Seite und die feste-Temperatur-Liste, löst und zeigt das Profil. Auch der ungültige Fall (beidseitig nur Fluss) ist zugelassen — dann erlebst du selbst, wie der Träger schwimmt oder davontreibt, und bekommst die Erklärung dazu.

Vorhersage-Punkt

Bevor du ausführst: Der Schalter startet mit links `"temperatur"` 20 °C und rechts `"strom"` 200 W (die beflammte Gerade auf 420 °C). Stell rechts auf `"isoliert"` um (Zeile `RECHTS_TYP = "isoliert"`). Welche Temperatur zeigt Knoten 5 dann — 420 °C, 20 °C oder etwas dazwischen? Leg dich fest, bevor du es probierst.

```
#| autorun: true
# --- Der Rand-Schalter: probiere die Kombinationen aus ---
# Erlaubte Typen: "temperatur" (Grad C), "strom" (Watt), "isoliert".
LINKS_TYP = "temperatur"
LINKS_WERT = 20.0
RECHTS_TYP = "strom"
RECHTS_WERT = 200.0

WAERMELEITFAEHIGKEIT = 50.0
QUERSCHNITT_M2 = 0.01
TRAEGERLAENGE_M = 1.0
KNOTENZAHL = 5


def gauss_elimination(matrix, rechte_seite):
    """Loest ein volles Gleichungssystem (Kapitel 4)."""
    n = len(matrix)
    werk = []
    for zeile in matrix:
        werk.append(list(zeile))
    seite = list(rechte_seite)
    for spalte in range(n):
        pivot = werk[spalte][spalte]
        for zeile in range(spalte + 1, n):
            faktor = werk[zeile][spalte] / pivot
            for s in range(spalte, n):
                werk[zeile][s] = werk[zeile][s] - faktor * werk[spalte][s]
            seite[zeile] = seite[zeile] - faktor * seite[spalte]
    loesung = [0.0] * n
    for zeile in range(n - 1, -1, -1):
        summe = seite[zeile]
        for s in range(zeile + 1, n):
            summe = summe - werk[zeile][s] * loesung[s]
        loesung[zeile] = summe / werk[zeile][zeile]
    return loesung
```

```
def gauss_seidel(matrix, rechte_seite, start, runden):
    """Naeherungsloeser (Kapitel 4), feste Rundenzahl -- zeigt das Schwimmen."""
    n = len(matrix)
    wert = list(start)
    for runde in range(runden):
        for i in range(n):
            summe = rechte_seite[i]
            for j in range(n):
                if j != i:
                    summe = summe - matrix[i][j] * wert[j]
            wert[i] = summe / matrix[i][i]
    return wert


def baue_system():
    """Assembliert das 5x5-System des Trägers (k = lambda*A/h, Kapitel 7)."""
    elementzahl = KNOTENZAHL - 1
    h = TRAEGERLAENGE_M / elementzahl
    k = WAERMELEITFAEHIGKEIT * QUERSCHNITT_M2 / h
    matrix = []
    for zeile in range(KNOTENZAHL):
        matrix.append([0.0] * KNOTENZAHL)
    for element in range(elementzahl):
        links = element
        rechts = element + 1
        matrix[links][links] = matrix[links][links] + k
        matrix[links][rechts] = matrix[links][rechts] - k
        matrix[rechts][links] = matrix[rechts][links] - k
        matrix[rechts][rechts] = matrix[rechts][rechts] + k
    return matrix


def baue_rand_ein(rechte_seite, feste, typ, wert, knoten):
    """Baut eine Randbedingung an einem Knoten ein (der Baukasten aus Kap. 8)."""
    if typ == "temperatur":
        feste[knoten] = wert
    elif typ == "strom":
        rechte_seite[knoten] = rechte_seite[knoten] + wert
    # "isoliert": nichts tun (Randterm = 0)


matrix = baue_system()
```

```
rechte_seite = [0.0] * KNOTENZAHL
feste = [None] * KNOTENZAHL
baue_rand_ein(rechte_seite, feste, LINKS_TYP, LINKS_WERT, 0)
baue_rand_ein(rechte_seite, feste, RECHTS_TYP, RECHTS_WERT, KNOTENZAHL - 1)

# Das modifizierte System zeigen.
print("Rechte Seite (Watt):     ", rechte_seite)
zeile = "Feste Temperaturen (°C): ["
for i in range(KNOTENZAHL):
    if feste[i] is None:
        zeile = zeile + " frei "
    else:
        zeile = zeile + ("%4.0f " % feste[i])
print(zeile + "]")
print()

# Sind Knoten festgehalten? Dann streichen und direkt loesen.
frei = []
for i in range(KNOTENZAHL):
    if feste[i] is None:
        frei.append(i)

abstand_m = TRAEGERLAENGE_M / (KNOTENZAHL - 1)
orte = []
for i in range(KNOTENZAHL):
    orte.append(i * abstand_m)

if len(frei) < KNOTENZAHL:
    kleine_matrix = []
    kleine_seite = []
    for i in frei:
        neue_zeile = []
        for j in frei:
            neue_zeile.append(matrix[i][j])
        kleine_matrix.append(neue_zeile)
        beitrag = 0.0
        for j in range(KNOTENZAHL):
            if feste[j] is not None:
                beitrag = beitrag + matrix[i][j] * feste[j]
        kleine_seite.append(rechte_seite[i] - beitrag)
    innere = gauss_elimination(kleine_matrix, kleine_seite)
    temperaturen = list(feste)
    for platz in range(len(frei)):
```

```
        temperaturen[frei[platz]] = innere[platz]
    print("Eindeutige Lösung:")
    for i in range(KNOTENZAHL):
        print("  Knoten %d (%.2f m): %7.1f °C" % (i + 1, orte[i],
                                                  temperaturen[i]))
else:
    bilanz = 0.0
    for w in rechte_seite:
        bilanz = bilanz + w
    temperaturen = gauss_seidel(matrix, rechte_seite, [0.0] * KNOTENZAHL, 400)
    print("KEINE feste Temperatur -> der Träger schwimmt!")
    if abs(bilanz) < 1e-6:
        print("Die Flüsse sind ausgeglichen (Netto %.0f W): die FORM steht," % bilanz)
        print("aber die HÖHE ist willkürlich. Gauß-Seidel liefert eine von")
        print("unendlich vielen Lösungen (hier aus Start 0):")
    else:
        print("Die Flüsse sind NICHT ausgeglichen (Netto %.0f W): stationär" % bilanz)
        print("gibt es gar keine Lösung, die Temperatur treibt davon. Was")
        print("Gauß-Seidel nach 400 Runden zeigt, ist kein Ergebnis, sondern")
        print("eine Momentaufnahme des Davonlaufens:")
    for i in range(KNOTENZAHL):
        print("  Knoten %d (%.2f m): %9.1f °C" % (i + 1, orte[i],
                                                  temperaturen[i]))

# --- Abbildung ---
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(8.0, 3.6))
ax.plot(orte, temperaturen, "o-", color="tab:red", ms=9,
        markeredgecolor="black")
ax.set_xlabel("Ort entlang des Trägers (m)")
ax.set_ylabel("Temperatur (°C)")
ax.set_title("Dein Rand-Schalter: %s | %s" % (LINKS_TYP, RECHTS_TYP))
plt.tight_layout()
plt.show()
```

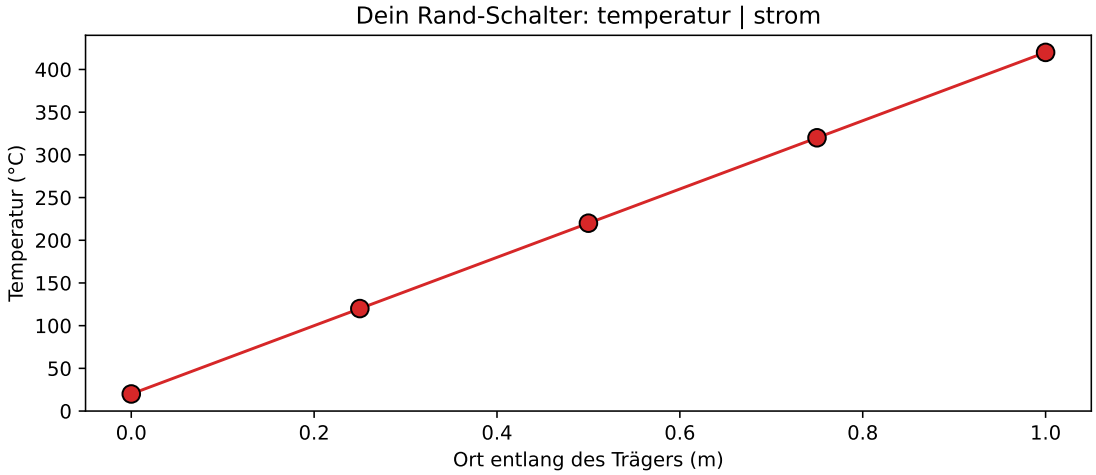

Abbildung 8.7.: Vorgerenderte Fassung des Rand-Schalters mit der Startbelegung (links 20 °C gehalten, rechts 200 W beflammt): die Gerade steigt auf 420 °C. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Zelle darüber ausführst — stell rechts auf “isoliert” und sieh, wie das Profil auf die flache 20-°C-Linie zusammenfällt.

Was man sieht: Mit der Startbelegung steigt die Gerade auf 420 °C. Stellst du `RECHTS_TYP = "isoliert"`, fällt das ganze Profil auf die flache Linie bei 20 °C zusammen — Knoten 5 zeigt dann 20 °C, nicht 420. Das war die Pointe des Vorhersage-Punkts: Ein isoliertes Ende führt keine Wärme ab, also staut sich nichts, und bei fehlender Quelle gleicht sich alles dem gehaltenen linken Ende an. Probier auch links und rechts *beide* auf `"strom"` — dann siehst du den schwimmenden Träger (bei ausgeglichenen Werten) oder das Davontreiben (bei zu viel Zustrom) aus dem vorigen Abschnitt selbst.

8.9. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript liegt in `programme/kap08/kap08_randbedingungen.py`. Es erweitert die vier Funktionen aus Kapitel 7 um einen **Randbedingungs-Baukasten**: `baue_rand_ein(...)` setzt eine einzelne Randbedingung (Temperatur, Strom oder isoliert) an ein Trägerende, `reduziere(...)` streicht alle festgehaltenen Knoten auf einen Schlag, und `loese_mit_raendern(...)` fädelt die Lösung ein — mit mindestens einer festen Temperatur direkt über die Gauß-Elimination, ohne jede feste Temperatur über Gauß-Seidel, das dann das Schwimmen und Davontreiben sichtbar macht. Das Skript rechnet alle fünf Szenarien dieses Kapitels vor (beidseitig gehalten, beflammt, isoliert,

schwimmend, unlösbar) und speichert die Umschalt-Abbildung. Führe es mit `python kap08_randbedingungen.py` aus. `baue_elementmatrix` und `assembliere` sind bewusst Zeile für Zeile aus dem Kapitel-7-Programm einkopiert (mit Quellhinweis), damit das Skript ohne Pfad-Bastelei über Kapitelgrenzen läuft.

8.10. Die Randbedingungs-Werkstatt

Randbedingungen sind der häufigste Fehler in der Praxis — also machen wir aus der Erfahrung dieses Kapitels eine feste Prüfroutine. Diese Checkliste geht man vor *jeder* Simulation durch; sie kehrt in Kapitel 12 (Verifikation) und 3 (das große Finale) wieder.

Werkstatt: der Randbedingungs-Review

1. **Hat jeder Rand eine Bedingung?** Gehe jedes Ende (in 2D: jede Randkante) durch. Ein Rand, über den du *nichts* gesagt hast, ist nicht „offen” — er ist **isoliert** (Fluss null). War das Absicht?
2. **Ist mindestens eine Temperatur (Dirichlet) vorgegeben?** Ohne einen einzigen festgehaltenen Wert **schwimmt** das System — es ist nur bis auf eine Konstante bestimmt. Reine Fluss-Ränder genügen nie allein.
3. **Stimmt die Bilanz bei reinen Fluss-Rändern?** Wenn du (fast) nur Flüsse vorgibst: Summiert sich die eingespeiste Leistung zu null? Wenn mehr hinein- als hinausfließt, gibt es **gar keinen** stationären Zustand.
4. **Passt der Typ zur Physik?** Kühles Auflager und Thermostat → Temperatur (Dirichlet). Brandherd und Heizdraht → Strom (Neumann). Übergang an Brandgase oder Luft → Übergang (Robin). Ein Verwechseln von „Temperatur halten” und „Leistung einspeisen” ist der klassische Anfängerfehler.
5. **Sind die Werte plausibel und richtig herum?** Vorzeichen des Stroms (rein oder raus?), Einheit (°C, nicht K, wo das Programm es erwartet), Größenordnung. Ein Strom mit falschem Vorzeichen heizt, wo er kühlen soll.

8.11. Die Namensschilder

Wir haben die Sache gebaut; jetzt kleben wir die Namen an. Alle drei Grundtypen tragen den Namen eines Mathematikers des 19. Jahrhunderts.

Die Fachwörter, jetzt nachgereicht

- **Dirichlet-Randbedingung** (wesentliche Randbedingung): der **Wert** wird vorgegeben — beim Träger die feste Temperatur. Nach **Peter Gustav Lejeune Dirichlet** (1805–1859). Man muss sie dem System *aufzwingen* (Streichen oder Zeile ersetzen), darum „wesentlich".
- **Neumann-Randbedingung** (natürliche Randbedingung): die **Steigung** bzw. der **Fluss** wird vorgegeben — beim Träger der Wärmestrom. Nach **Carl Gottfried Neumann** (1832–1925). Sie steckt schon im Randterm der schwachen Form, ergibt sich also *natürlich*.
- **Robin-Randbedingung** (Übergangs-, Konvektionsrandbedingung): eine **Mischung** — der Fluss hängt linear von der Temperatur ab. Nach **Victor Gustave Robin** (1855–1897).
- **Natürlich / wesentlich:** natürlich = ohne Extra-Handgriff aus der schwachen Form (Neumann); wesentlich = muss aufgezwungen werden (Dirichlet).
- **Isoliert:** der Sonderfall des Neumann-Rands mit Fluss null — die eingebaute Standardannahme der FEM.

Merkkasten

- Der in Kapitel 7 geparkte **Randterm** ist der **Wärmestrom an den Enden** — dort wohnen die Flussrandbedingungen.
- **Neumann** (Fluss vorgeben): eine Zahl auf die **rechte Seite**, Matrix bleibt unberührt. Brandherd mit 200 W → Profil $20/120/220/320/420\,°\mathrm{C}$, gleicher Sprung $Q/k = 100\,\mathrm{K}$ pro Element.
- **Dirichlet** (Temperatur vorgeben): **Zeile ersetzen** oder **streichen** — beide liefern identisch $T_1 = 20\,°\mathrm{C}$. Profis benutzen oft den **Straftherm**.
- **Nichts vorgeben = perfekt beplankt** (Randterm null). Links 20 °C, rechts nichts, keine Quelle → überall 20 °C.
- Ohne **eine** feste Temperatur **schwimmt** der Träger (Lösung nur bis auf eine Konstante); bei unausgeglichenen Flüssen gibt es gar keine stationäre Lösung.

Roter Faden

Zurück: Dieses Kapitel hat den in Kapitel 7 geparkten **Randterm** eingelöst — er ist der Wärmestrom an den Enden, und aus ihm werden die Neumann-Randbedingungen. Das **Streich-Rezept** für feste Werte stammt aus Kapitel 3 (Feder, $u_1 = 0$) und Kapitel 7 (Temperatur, nicht null); hier haben wir es um das zweite Rezept (Zeile ersetzen) und den Blick auf den Strafterm ergänzt. Die **Gerade** des beflammten Trägers ist wieder die aus Kapitel 6, nur mit dem Brandherd statt des zweiten Auflagers als Steigungsgeber; der Sprung Q/k pro Element ist die Reihenschaltung aus Kapitel 3 und der Wärmestrom-Rechnung aus Kapitel 7. Gelöst wurde mit der **Gauß-Elimination** und dem **Gauß-Seidel-Verfahren** aus 2.

Vor: Der **Robin-Rand** wird in Kapitel 14 und 3 der realistische Brandgas-Übergang, wenn aus der idealen Leistungseinspeisung die Kühlung an strömende Gase wird. Die **Randbedingungs-Werkstatt** kehrt in Kapitel 12 (Verifikation) und 3 (Finale) wieder. Der **schwimmende Träger** taucht in Kapitel 14 als mechanischer Träger *ohne Lagerung* auf — dann schwebt nicht die Temperatur, sondern das ganze Bauteil davon (Starrkörpermoden). Und in Kapitel 9 kommt die **Zeit** dazu: Dann entscheidet der Rand nicht nur über das eingeschwungene Profil, sondern über den ganzen Weg dorthin.

Übungen

Ü 8.1 (Verstehen). Ordne die drei Randtypen — Dirichlet (Temperatur), Neumann (Fluss), Robin (Übergang) — den drei Alltagssituationen zu und begründe:

(a) Ein **Topf steht auf der Herdplatte**: Die Platte gibt eine eingestellte Leistung ab. Welcher Typ ist der Boden des Topfes? (b) Eine gefüllte **Thermoskanne**: Wie ist ihre Wand ideal modelliert? (c) Ein **Heizkörper im Zimmer** gibt Wärme an die Raumluft ab, umso mehr, je heißer er gegenüber der Luft ist. Welcher Typ ist seine Oberfläche?

> **i** Musterlösung zu Ü 8.1
>
> (a) **Herdplatte → Neumann.** Die Platte prägt eine **Leistung** ein (die Wattzahl der Stufe), unabhängig davon, wie heiß der Topfboden schon ist — genau der Brandherd dieses Kapitels. (b) **Thermoskanne → isoliert** (Neumann mit Fluss null). Ihre doppelwandige Vakuumwand lässt idealerweise **keinen Wärmestrom** durch — das ist die natürliche Randbedingung als Konstruktionsziel, das-

selbe Prinzip wie eine lückenlose Beplankung. (c) **Heizkörper → Robin.** Er gibt Wärme **proportional zur Übertemperatur** an die Luft ab: Ein sehr heißer Heizkörper in kühlem Zimmer kühlt schnell, ein lauwarmer kaum — Fluss hängt von der Temperatur ab, das ist der Übergang. (Ein *Thermostat*, der eine feste Raumtemperatur erzwingt, wäre dagegen Dirichlet.)

Ü 8.2 (Verändern). Der Träger links auf 20 °C, rechts der Brandherd. **Sage zuerst vorher:** Wenn du die Brandherd-Leistung von 200 W auf **400 W** verdoppelst — verdoppelt sich die Endtemperatur an Knoten 5 (von 420 °C auf 840 °C)? Rechne es dann mit dem Skript `loesungen/kap08_ue2.py` nach und deute das Ergebnis.

i Musterlösung zu Ü 8.2

Nein — nicht die Endtemperatur, sondern die Übertemperatur verdoppelt sich. Bei 400 W ist der Sprung je Element $\Delta T = Q/k = 400/2 = 200\,\mathrm{K}$ (doppelt so viel wie die 100 K bei 200 W), das Profil wird

$$[\,20,\ 220,\ 420,\ 620,\ 820\,]\ ^\circ\mathrm{C}.$$

Knoten 5 steigt von 420 auf 820 °C — das ist **nicht** das Doppelte von 420 (das wären 840). Verdoppelt hat sich die **Übertemperatur** über dem gehaltenen Rand: von $420 - 20 = 400\,\mathrm{K}$ auf $820 - 20 = 800\,\mathrm{K}$. Das Auflager hält den Nullpunkt bei 20 °C fest; verdoppeln lässt sich nur, was *über* ihm liegt. Das ist genau der Denkfehler, vor dem die Übung warnt — absolute Temperaturen sind keine Vielfachen voneinander, nur die Abstände vom festgehaltenen Bezugspunkt.
Und diesmal steht hinter der Zahl echte Physik: Mit 820 °C ist das rechte Ende in der **Kirschrotglut** angekommen (die Glühfarben-Tafel aus Kapitel 1 setzt dunkelrot ab rund 550 °C an, kirschrot bei etwa 800 °C) — der Träger **glüht** hier wirklich sichtbar. Das ist keine Effekthascherei, sondern die ernste Seite des Brandschutzes: Bei solchen Temperaturen hat Baustahl längst einen großen Teil seiner Festigkeit verloren (bei 600 °C etwa die Hälfte). Genau darum wird ein Träger im Brandfall nicht durch Schmelzen gefährlich — das käme erst bei rund 1500 °C —, sondern weil er seine Tragkraft verliert, lange bevor er glüht.

Ü 8.3 (Übertragen). Kombiniere zwei Bausteine: die **beidseitige Dirichlet-Bedingung** (links 20 °C, rechts 300 °C) aus diesem Kapitel *und* die **Wärmequelle** aus Übung 7.3 (ein Heizdraht mit $P = 160\,\mathrm{W/m}$, also 40 W je Element, hälftig auf die Knoten verteilt). Baue erst die Quelle auf die rechte Seite, dann die festen Ränder ein. Welches Profil erwartest

du — die Gerade, die Parabel aus 7.3, oder etwas Neues? Rechne mit `loesungen/kap08_ue3.py`.

> **Musterlösung zu Ü 8.3**
>
> Es kommt genau die **Parabel aus Übung 7.3** heraus:
>
> $$[20,\ 120,\ 200,\ 260,\ 300]\ °\mathrm{C}.$$
>
> Die festen Ränder halten die Enden auf 20 und 300 °C (wie die reine Gerade), und die Wärmequelle wölbt das Profil dazwischen über die Gerade auf — in der Mitte am stärksten ($+40$ K bei Knoten 3). Die Lehre: **Rand und rechte Seite werden unabhängig voneinander eingebaut und addieren sich.** Die Quelle geht auf die rechte Seite (Kapitel 7), die festen Temperaturen per Streich-Rezept (Kapitel 8) — die Reihenfolge ist gleichgültig, das Ergebnis dasselbe. Randbedingungen und Belastung sind zwei getrennte Regler am selben System.

Das Kleingedruckte

Drei ehrliche Feinheiten zum Schluss.

Erstens: Wir haben den Neumann-Rand als *eingeprägte Leistung in Watt* behandelt — eine Zahl auf der rechten Seite. In der Literatur wird die Neumann-Bedingung oft als vorgegebene **Wärmestromdichte** q (in W/m²) formuliert; die Leistung ist dann $Q = q \cdot A$ mit der Randfläche A. Für unseren 1D-Träger mit festem Querschnitt ist das nur ein Umrechnungsfaktor, und wir bleiben bei der anschaulicheren Wattzahl. In 2D (1) muss man sorgfältiger sein, weil sich die Leistung über die Länge der Randkante verteilt.

Zweitens: Der **Robin-Rand** ist hier nur beschrieben, nicht ausgerechnet. Sein Einbau ist etwas kniffliger, weil ein Teil von ihm (der temperaturabhängige Anteil) in die **Matrix** wandert — er verändert die Diagonale des Randknotens. Das lohnt sich erst, wenn wir den realistischen Brandgas-Übergang wirklich brauchen (3). Bis dahin genügen die beiden Grundtypen für alles.

Drittens: Alle Randbedingungen dieses Kapitels waren **zeitlich konstant** — das Auflager hält *immer* 20 °C, der Brandherd gibt *immer* 200 W. In Wirklichkeit ändern sich Ränder oft mit der Zeit (der Brandherd flammt auf, das Auflager erwärmt sich). Zeitabhängige Randbedingungen sind der natürliche nächste Schritt, sobald Kapitel 9 die **Zeit** ins Spiel bringt; das

Rezept bleibt dasselbe, nur werden die Randwerte in jedem Zeitschritt neu gesetzt.

9. Jetzt mit Zeit

Bisher stand unser Träger immer schon fertig aufgeheizt vor uns. Wir haben die Gerade berechnet, auf der sich die Temperaturen im eingeschwungenen Zustand einpendeln — 20 °C am kühlen Auflager links, 300 °C beim Brandherd rechts, sauber dazwischen gestaffelt. Aber die entscheidende Frage des Brandschutzes haben wir dabei übersprungen: *Wie lange dauert es überhaupt, bis es so weit ist?* Wie viele Minuten kriecht die Hitze nach links, ehe sie auch das ferne Ende im Auflager erreicht?

Ab jetzt tickt die Uhr. In Kapitel 1 lief diese Simulation schon einmal vor deinen Augen ab — die Wärmefront, die vom Brandherd ins Stahlprofil kriecht —, aber ihr Maschinenraum war eine Blackbox. Dieses Kapitel baut genau diesen Maschinenraum. Und es hält ein Versprechen ein, das dort schon im Code steckte: die kleine Schleife mit den „expliziten Zeitschritten". Am Ende dieses Kapitels weißt du, warum sie so aussieht, wie sie aussieht — und warum sie, einen Tick zu mutig eingestellt, mit einem Knall explodiert.

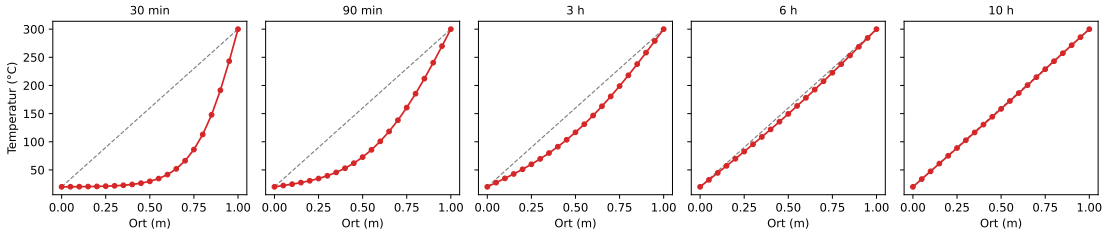

Abbildung 9.1.: Die wandernde Wärmefront als Bildreihe (in der HTML-Fassung eine Animation): fünf Momentaufnahmen über zehn Stunden. Die Front schiebt sich vom Brandherd rechts nach links, bis die eingeschwungene Gerade (gestrichelt) erreicht ist. Zeitraffer — die Angabe ist die echte verstrichene Zeit. Nach 30 und 90 Minuten (den ersten beiden Bildern) ist erst das rechte Ende warm; das Auflager links liegt noch kalt.

Was man hier sieht: einen Träger, der nicht auf einen Schlag heiß ist, sondern Minute um Minute wärmer wird. Anfangs steht nur das rechte Ende beim Brandherd, der Rest liegt kalt im Auflager. Dann frisst sich die Wärme

nach links, erst steil, dann immer träger, bis sich das Profil an die vertraute Gerade aus Kapitel 6 anschmiegt und stehen bleibt. Wichtig: Nach 30 und 90 Minuten — den ersten beiden Bildern der Reihe — ist erst das rechte Drittel warm; das Auflager bleibt lange kalt. Diese ganze Bewegung — den Weg zur Geraden — wollen wir jetzt selbst rechnen.

Lernziele

Nach diesem Kapitel kannst du …

1. … den **Speicherterm** $\rho c\,\partial T/\partial t$ physikalisch erklären (die Bilanz aus Kapitel 6, bei der das Konto sich jetzt doch ändert) und die Rolle der volumetrischen Wärmekapazität ρc benennen,
2. … erklären, was die **Massenmatrix** speichert und was **Lumping** damit tut — die Wärmekapazität eines Elements auf seine beiden Knoten verteilen,
3. … einen **expliziten Euler-Schritt** von Hand rechnen — neuer Wert = alter Wert + Zeitschritt · momentane Änderungsrate — am 5-Knoten-Träger mit echten Zahlen,
4. … das **Stabilitätslimit** erklären und als Faustregel formulieren: halbe Maschenweite erzwingt ein Viertel des Zeitschritts, sonst explodiert die Rechnung,
5. … den **impliziten Schritt** erklären — alle neuen Werte gleichzeitig aus einem Gleichungssystem bestimmen — und begründen, wann sich sein Mehraufwand lohnt.

⚠ Naheliegende Vermutung

Vermutung: *„Der Zeitschritt ist wie die Netzfeinheit nur eine Frage der Genauigkeit. Wähle ich ihn grob, wird das Ergebnis eben etwas ungenauer — aber es bleibt ein Ergebnis.”*
Warum sie naheliegt: Bei allem, was wir bisher genähert haben — die Fläche unter einer Kurve, das Temperaturprofil zwischen den Knoten — , galt genau das: grob = ungenau, fein = genau. Ein sanfter Regler, kein Abgrund.
Was stattdessen stimmt: Beim expliziten Zeitschritt gibt es eine **scharfe Grenze**. Diesseits davon wird ein größerer Schritt tatsächlich nur etwas ungenauer. Aber einen Tick jenseits der Grenze wird die Rechnung nicht ungenau, sondern sie **explodiert**: Die Temperaturen fangen an zu zappeln, das Zappeln schaukelt sich von Schritt zu Schritt auf, und nach wenigen Sekunden stehen an den Knoten Millionen Grad — mal plus, mal minus. Ein zu großer Schritt überzieht ein Konto, und

die Überziehung wächst sich zur Katastrophe aus. Das rechnen und erleben wir in diesem Kapitel.

9.1. Das Konto ändert sich jetzt doch

In Kapitel 6 war ein Trägerstück ein Girokonto für Wärme: vom Brandherd strömen 140 W herein, zum kühlen Auflager fließen 140 W hinaus, die Bilanz geht auf null auf, und die Temperatur des Stücks bleibt stehen. Das war die Bedingung des eingeschwungenen Zustands — und genau deshalb galt sie **erst am Ende** des Aufheizens.

Solange der Träger noch kalt ist, stimmt die Bilanz eben *nicht*. An einem Stück nahe dem Brandherd strömt mehr Wärme herein, als nach links weiterfließt. Wohin geht der Überschuss? Er bleibt im Stück und **heizt es auf**. Das ist keine Nebensache, sondern der ganze Vorgang: Die *Differenz* zwischen Zufluss und Abfluss ist genau das, was die Temperatur des Stücks von Sekunde zu Sekunde ändert.

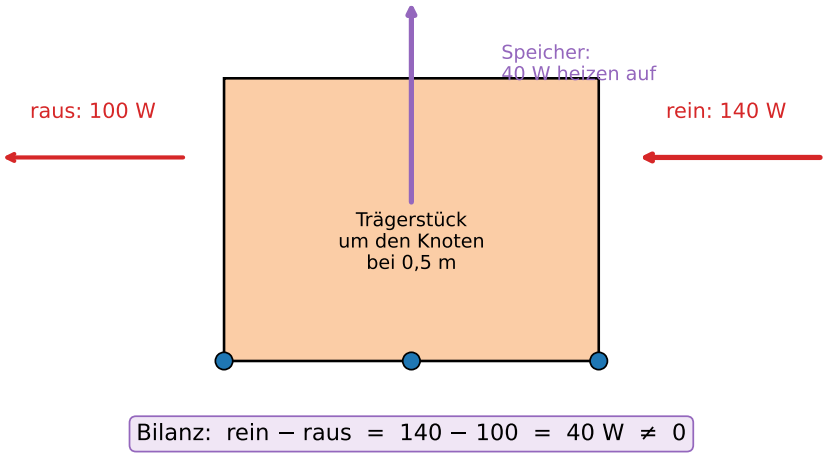

Abbildung 9.2.: Dieselbe Bilanz wie in Kapitel 6, aber im Aufheizen: Vom Brandherd strömen 140 W herein, nach links zum kühlen Auflager fließen erst 100 W ab. Die Differenz von 40 W verschwindet nicht — sie lädt das Stück auf (der lila Pfeil nach oben) und hebt seine Temperatur. Dieser Speicherpfeil ist der neue Term dieses Kapitels.

Was man hier sieht: dasselbe Konto wie in Kapitel 6, aber mitten im Aufheizen. Diesmal ist der Abfluss kleiner als der Zufluss, und die Differenz — hier 40 W — ist nicht verloren. Sie steckt als **gespeicherte Wärme** im Stück und treibt seine Temperatur nach oben (der lila Pfeil). Genau diesen Pfeil hat Kapitel 6 bewusst weggelassen, um „ein schweres Ding nach dem anderen" zu behandeln. Jetzt holen wir ihn.

9.1.1. Wie viel Wärme steckt in einem Grad?

Wie stark ein bisschen überschüssige Wärme die Temperatur hebt, hängt vom Material ab. Zwei Stoffgrößen des Baustahls bestimmen das: die **Dichte** $\rho = 7850\ \mathrm{kg/m^3}$ (wie viel Masse in einem Kubikmeter steckt) und die **spezifische Wärmekapazität** $c = 460\ \mathrm{J/(kg\,K)}$ (wie viel Wärme ein Kilogramm um ein Grad erwärmt). Multipliziert man beide, bekommt man die **volumetrische Wärmekapazität**:

$$\rho\, c \;=\; 7850 \cdot 460 \;=\; 3\,611\,000\ \frac{\mathrm{J}}{\mathrm{m^3\,K}} \;=\; 3{,}61\ \frac{\mathrm{MJ}}{\mathrm{m^3\,K}}.$$

Lies das so: Um einen Kubikmeter Stahl um ein einziges Grad zu erwärmen, muss man 3,61 Millionen Joule hineinstecken. Das ist der Preis der Trägheit — und der Grund, warum der Träger sich Zeit lässt, während der Brand unter ihm längst lodert.

In der Sprache der Wärmeleitungsgleichung tritt dieser Speicher als neuer Term links neben die alte Bilanz. Statt „Zufluss minus Abfluss = 0" heißt es jetzt „Zufluss minus Abfluss = das, was gespeichert wird":

$$\underbrace{\rho\, c\, \frac{\partial T}{\partial t}}_{\text{Speicher (neu)}} \quad = \quad \underbrace{\frac{\partial}{\partial x}\left(\lambda\, \frac{\partial T}{\partial x}\right)}_{\text{Zufluss − Abfluss (Kapitel 6)}} \quad .$$

Das Zeichen $\partial T/\partial t$ ist die **Änderungsrate der Temperatur über der Zeit** — Grad pro Sekunde, genau die Steigung aus Kapitel 5, nur dass die Waagerechte jetzt nicht der Ort x ist, sondern die Zeit t. Rechts steht buchstäblich die Bilanz aus Kapitel 6. Im eingeschwungenen Zustand ist die linke Seite null (nichts ändert sich mehr), und wir sind zurück bei der alten Geraden. Alles Neue an diesem Kapitel steckt in dem einen linken Term.

9.2. Von der Wärmekapazität zur Massenmatrix

Den Zufluss-minus-Abfluss-Term haben wir in Kapitel 7 in die Maschinerie übersetzt: Er wurde zur **Steifigkeitsmatrix** mit der Wärme-Steifigkeit $k = \lambda A/h = 2\ \mathrm{W/K}$ zwischen je zwei Knoten. Mit dem neuen Speicherterm machen wir jetzt genau dasselbe Spiel — dieselbe schwache Form, dieselben Hütchenfunktionen N_i aus Kapitel 7.

Setzt man den Ansatz $T(x,t) = \sum_j T_j(t)\, N_j(x)$ (Knotenwerte mal Hütchen, jetzt mit der Zeit veränderliche Knotenwerte) in den Speicherterm ein und gewichtet wie immer mit dem Hütchen N_i, entsteht aus $\int \rho c\, N_i\, \partial T/\partial t \,\mathrm{d}x$ eine Summe: Jeder Knoten j trägt mit seiner Änderungsrate $\mathrm{d}T_j/\mathrm{d}t$ bei, gewichtet mit dem Überlapp der beiden Hütchen. Diese Gewichte bilden eine neue Matrix — die **Massenmatrix M**. Für ein einzelnes Element sieht sie so aus:

Ergebnis-Kasten: die konsistente Elementmassenmatrix

$$\mathbf{M}_{\text{Element}} \;=\; \frac{\rho\, c\, A\, h}{6}\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

Sie wiegt ab, wie stark die Temperaturänderung an einem Knoten in die Bilanz seines Nachbarn hineinwirkt. Weil sich benachbarte Hütchen überlappen, sind auch die Nebeneinträge (die 1en) ungleich null: Die Änderungsraten benachbarter Knoten sind **gekoppelt**. Diese exakt integrierte Form heißt **konsistente** Massenmatrix. Wo die 6 und die $2, 1$ herkommen, steht im Kleingedruckten am Kapitelende — für den Weg durch dieses Kapitel brauchen wir sie nicht.

Der Vorfaktor ist die gesamte **Wärmekapazität eines Elements**: Masse mal spezifische Wärme, also $\rho\, c\, A\, h$. Mit den Zahlen unseres 5-Knoten-Trägers ($A = 0{,}01\ \mathrm{m}^2$, $h = 0{,}25\ \mathrm{m}$):

$$\rho\, c\, A\, h \;=\; 3\,611\,000 \cdot 0{,}01 \cdot 0{,}25 \;=\; 9027{,}5\ \frac{\mathrm{J}}{\mathrm{K}}.$$

Ein Elementstück des Trägers braucht also 9027,5 Joule, um ein Grad wärmer zu werden. Diese Zahl trägt das ganze Kapitel.

9.2.1. Lumping: jeder Knoten bekommt seinen Eimer

Die konsistente Massenmatrix hat einen Haken: Ihre Nebeneinträge koppeln die Knoten. Wollen wir gleich einen Zeitschritt machen, müssten wir

wegen dieser Kopplung selbst bei der einfachsten Rechnung ein Gleichungssystem lösen — der Vorteil eines „billigen" expliziten Schritts wäre dahin.

Es gibt einen ehrlichen Ausweg, und er hat einen anschaulichen Namen: **Lumping**, vom englischen *lump*, der Klumpen. Statt die Speicherwirkung über das Element zu verschmieren, klumpen wir sie an den Knoten zusammen. Jedes Element gibt seine Kapazität $\rho c A h$ je zur Hälfte an seine beiden Knoten ab — genau die **Halbe-Halbe-Logik**, mit der schon in Kapitel 7 die Wärmequelle auf die Knoten verteilt wurde. Das Bild dazu: An jedem Knoten hängt ein **Eimer**, in dem sich seine Wärme sammelt.

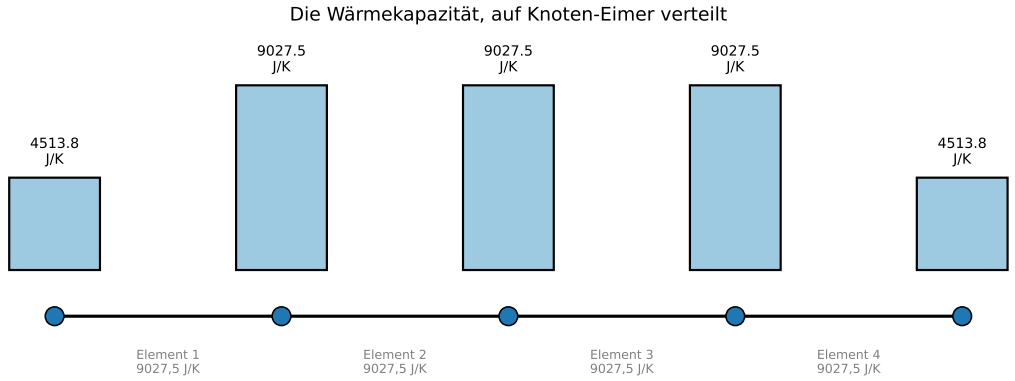

Abbildung 9.3.: Lumping am 5-Knoten-Träger: Jedes der vier Elemente (Kapazität 9027,5 J/K) gibt die Hälfte an jeden seiner beiden Knoten ab. Innere Knoten bekommen von zwei Elementen je 4513,75 J/K — zusammen ein voller Eimer von 9027,5 J/K. Die beiden Randknoten grenzen nur an ein Element und tragen den halben Eimer, 4513,75 J/K.

Was man hier sieht: fünf Eimer, aber nicht gleich groß. Die drei inneren Knoten grenzen an zwei Elemente und sammeln von jedem die Hälfte ein — $4513{,}75 + 4513{,}75 = 9027{,}5\ \mathrm{J/K}$, ein voller Eimer. Die beiden Randknoten sitzen am Ende und grenzen nur an ein einziges Element; ihr Eimer ist halb so groß, $4513{,}75\ \mathrm{J/K}$. Wichtig: Die Gesamtsumme bleibt erhalten — nichts geht verloren, die Wärme wird nur anders verteilt. Aus der Massenmatrix ist eine reine Diagonale geworden, ein Eimer je Knoten, keine Kopplung mehr. Genau das brauchen wir für den nächsten Schritt.

9.3. Der explizite Euler-Schritt

Jetzt haben wir alle Teile beisammen. Die Bilanz an einem inneren Knoten lautet in Worten: *Die gespeicherte Wärme ändert sich so schnell, wie netto Wärme hineinfließt.* In Zahlen — Eimer mal Änderungsrate = Zufluss minus Abfluss:

$$C_i \cdot \frac{\mathrm{d}T_i}{\mathrm{d}t} \;=\; k\,(T_{i-1} - 2\,T_i + T_{i+1}).$$

Links steht der Eimer C_i (für innere Knoten $9027{,}5\ \mathrm{J/K}$) mal der Änderungsrate. Rechts steht die Bilanz aus Kapitel 6 und Kapitel 7, ausgedrückt durch die drei Knoten: Der Knoten schaut nach links und rechts, und was seine beiden Nachbarn zusammen mehr haben als das Doppelte von ihm, treibt ihn. Das Minus-Doppelte ist genau die $-2, +4, -2$-Zeile der Steifigkeitsmatrix.

Der Trick, aus dieser Beziehung eine Vorhersage zu machen, ist der **explizite Euler-Schritt**, benannt nach Leonhard Euler (Euler 1768). Er ist bestechend einfach: Ersetze die Änderungsrate durch „Wie viel ändert sich pro Zeitschritt Δt" und löse nach dem neuen Wert auf:

$$T_i^{\text{neu}} \;=\; T_i^{\text{alt}} \;+\; \frac{\Delta t}{C_i}\,k\,(T_{i-1} - 2\,T_i + T_{i+1}).$$

In Worten, und das ist der ganze Satz, den man sich merken muss:

> Neuer Wert = alter Wert + Zeitschritt · momentane Änderungsrate.

Jeder Knoten schaut auf seine zwei alten Nachbarwerte, rechnet seine Änderungsrate aus und macht einen Schritt in diese Richtung. Kein Gleichungssystem, keine Kopplung — nur eine Division je Knoten. Genau das macht das Lumping so wertvoll.

Vorhersage-Punkt

Bevor du weiterliest: Gleich rechnen wir den Träger von Hand los. Der Startzustand ist die Liste `[20, 20, 20, 20, 300]` aus Kapitel 2 — alles kalt, nur das rechte Ende beim Brandherd. Wir wählen $\Delta t = 600$ s (zehn Minuten). Welcher der drei inneren Knoten (2, 3 oder 4) ändert sich im allerersten Schritt als Einziger? Und warum die anderen beiden (noch) nicht? Leg dich fest.

9.3.1. Drei Schritte von Hand

Wir rechnen mit $\Delta t = 600$ s, $k = 2$ W/K und dem inneren Eimer $C = 9027{,}5$ J/K. Der Vorfaktor ist für alle inneren Knoten gleich:

$$\frac{\Delta t}{C}\,k \;=\; \frac{600}{9027{,}5}\cdot 2 \;\approx\; 0{,}1329.$$

Die Randknoten 1 (20 °C, Auflager) und 5 (300 °C, Brandherd) sind festgehalten — Dirichlet, wie in Kapitel 8 — und ändern sich nie. Nur die Knoten 2, 3 und 4 rechnen. Nehmen wir den ersten Schritt für Knoten 4 (rechts, neben dem Brandherd):

$$T_4^{\text{neu}} = 20 + 0{,}1329 \cdot (\underbrace{20}_{T_3} - 2\cdot 20 + \underbrace{300}_{T_5}) = 20 + 0{,}1329 \cdot 280 = 57{,}2\ ^\circ\text{C}.$$

Für Knoten 2 und 3 dagegen sind im ersten Schritt beide Nachbarn noch bei 20 °C, die Klammer wird $20-40+20 = 0$ — sie ändern sich nicht. Die Wärme muss erst bei ihnen ankommen. Genau das war die Vorhersage-Frage.

Explizite Handrechnung: die Wärmefront kriecht nach links

Zeitpunkt	Knoten 1	Knoten 2	Knoten 3	Knoten 4	Knoten 5
Start (0 s)	20.0	20.0	20.0	20.0	300.0
nach 600 s	20.0	20.0	20.0	57.2	300.0
nach 1200 s	20.0	20.0	24.9	84.5	300.0
nach 1800 s	20.0	20.7	32.2	105.3	300.0

Abbildung 9.4.: Die ersten drei expliziten Zeitschritte von Hand, Δt = 600 s. Rot markiert ist in jedem Schritt, welche Knoten sich neu bewegt haben. Die Wärme kriecht vom Knoten 4 (am Brandherd) Schritt für Schritt nach links — nach einer halben Stunde hat sie gerade eben Knoten 2 erreicht.

Was man hier sieht: die Front als Zahlenwelle. Im ersten Schritt bewegt sich nur Knoten 4. Im zweiten hat die Wärme Knoten 3 erreicht, im dritten auch Knoten 2. Die beiden grauen Randspalten stehen fest. Jede Zahl in

dieser Tabelle ist mit dem Taschenrechner nachprüfbar — es steckt kein einziger versteckter Schritt darin.

Und genau diese Tabelle ist eine `for`-Schleife. Der Kern des expliziten Schritts sind acht Zeilen reines Python — eine je Nachbar-Blick, so wie in der Handrechnung:

```
def expliziter_schritt(temperaturen, kapazitaeten, k, dt):
    """Ein expliziter Euler-Zeitschritt am 1D-Träger.

    temperaturen: Liste der jetzigen Knotentemperaturen (Grad Celsius).
    kapazitaeten: gelumpte Knotenkapazitaeten C_i (J/K).
    k: Waerme-Steifigkeit lambda*A/h (W/K).
    dt: Zeitschritt (Sekunden).
    Rueckgabe: neue Temperaturliste; die Randknoten bleiben fest.
    """
    neu = list(temperaturen)
    for i in range(1, len(temperaturen) - 1):
        zufluss = k * (temperaturen[i - 1]
                       - 2.0 * temperaturen[i]
                       + temperaturen[i + 1])
        neu[i] = temperaturen[i] + dt / kapazitaeten[i] * zufluss
    return neu


# Die Handrechnung nachfahren: Start [20, 20, 20, 20, 300], drei Schritte.
kapazitaeten = [4513.75, 9027.5, 9027.5, 9027.5, 4513.75]
temperaturen = [20.0, 20.0, 20.0, 20.0, 300.0]
for schritt in range(3):
    temperaturen = expliziter_schritt(temperaturen, kapazitaeten, 2.0, 600.0)
    print("nach %4d s:" % ((schritt + 1) * 600), end=" ")
    for wert in temperaturen:
        print("%8.3f" % wert, end="")
    print()
```

```
nach  600 s:   20.000  20.000  20.000  57.220 300.000
nach 1200 s:   20.000  20.000  24.947  84.544 300.000
nach 1800 s:   20.000  20.658  32.212 105.262 300.000
```

Interpretation: Zeile für Zeile dieselben Zahlen wie in der Tabelle — `57.220`, dann `24.947` und `84.544`, dann `20.658`, `32.212` und `105.262`. Die `for`-Schleife über `i` ist der Blick jedes Knotens auf seine Nachbarn `i-1` und `i+1`; die Division durch `kapazitaeten[i]` ist der Eimer. Mehr passiert bei einem expliziten Zeitschritt nicht.

9.4. Das Experiment: den Zeitschritt vergrößern

Zehn Minuten pro Handrechnung sind mühsam. Bis der Träger eingeschwungen ist, vergehen viele Stunden (dazu gleich mehr) — das sind Tausende von 600-Sekunden-Schritten. Der naheliegende Wunsch: den Zeitschritt größer machen, dann sind es weniger Schritte. Probieren wir es. Was passiert bei $\Delta t = 1800$ s (einer halben Stunde), und was bei $\Delta t = 3000$ s (fünfzig Minuten)?

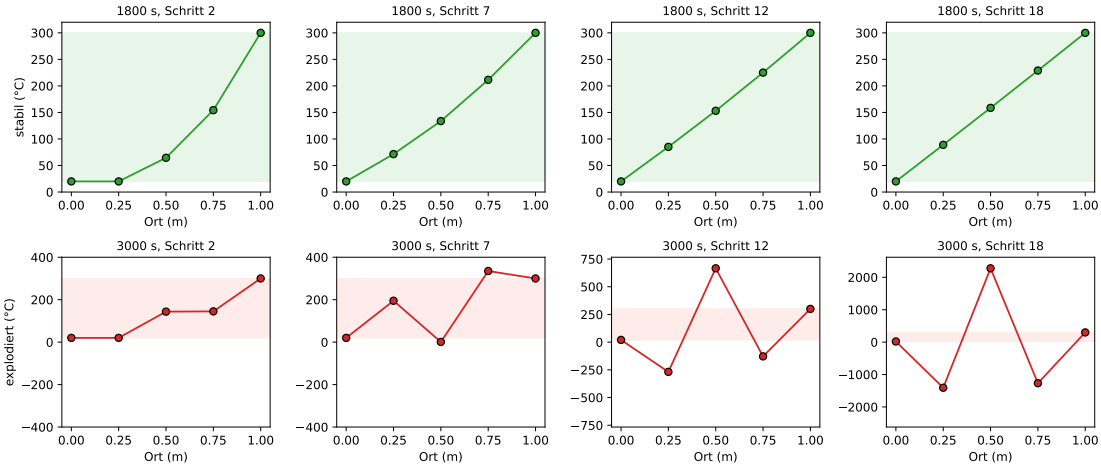

Abbildung 9.5.: Die Explosion als Bildreihe (in der HTML-Fassung eine Animation). Obere Reihe Δt = 1800 s: die Front wandert brav ein und bleibt im grünen Band zwischen 20 und 300 °C. Untere Reihe Δt = 3000 s: erst kleine Ausschläge, dann Überschwinger über 300 und unter 0 — beachte die wachsende y-Achse — bis die Skala sprengt.

Was man hier sieht: zwei Rechnungen mit demselben Programm, demselben Träger, demselben Startzustand — nur der Zeitschritt ist anders. Links mit $\Delta t = 1800$ s läuft alles ruhig ein. Rechts mit $\Delta t = 3000$ s beginnt es harmlos, dann fangen die inneren Knoten an, gegeneinander zu schwingen: einer schießt über 300 °C hinaus, sein Nachbar fällt unter null, im nächsten Schritt ist es umgekehrt, nur heftiger. Nach wenigen Schritten stehen dort Hunderte, bald Tausende Grad — mit wechselndem Vorzeichen, und es wächst weiter. Physikalisch ist das Unsinn: Nichts am Träger kann kälter als das Auflager oder heißer als der Brandherd werden. Es ist kein Messfehler und keine Ungenauigkeit, sondern ein **Versagen des Verfahrens**.

Zappeln ist nicht gleich Zappeln

In 2 hat die Lösung beim Gauß-Seidel-Verfahren auch „gezappelt" — sie pendelte sich Runde um Runde in die richtige Antwort ein. Das war *erwünscht*: der Lösungsweg selbst. Hier ist das Zappeln das genaue Gegenteil — es ist ein Verfahrensfehler, der sich aufschaukelt und von der Wahrheit *wegläuft*. Beim Einpendeln werden die Ausschläge kleiner, bei der Explosion größer. Dieselbe Zickzack-Optik, entgegengesetzte Bedeutung.

9.4.1. Die scharfe Grenze und die Viertel-Regel

Warum kippt es genau zwischen 1800 und 3000 Sekunden? Der explizite Schritt korrigiert jeden Knoten um einen Bruchteil seines Rückstands zu den Nachbarn. Ist dieser Bruchteil zu groß, schießt die Korrektur über das Ziel hinaus — und weil im nächsten Schritt dasselbe mit umgekehrtem Vorzeichen passiert, wächst der Überschuss. Die Grenze, ab der das losgeht, lässt sich als **Faustregel** hinschreiben:

$$\Delta t_{\max} \;=\; \frac{C_{\text{innen}}}{2\,k} \;=\; \frac{9027{,}5}{2 \cdot 2} \;=\; \frac{9027{,}5}{4} \;\approx\; 2257\ \text{s}.$$

Das sind rund 38 Minuten. Unsere 1800 s (eine halbe Stunde) liegen darunter (stabil), die 3000 s (fünfzig Minuten) darüber (Explosion) — genau wie beobachtet. Dieselbe Grenze lässt sich auch mit den Materialgrößen und der Maschenweite schreiben, und in dieser Form zeigt sie das Entscheidende:

$$\Delta t_{\max} \;=\; \frac{h^2\,\rho\,c}{2\,\lambda} \;=\; \frac{0{,}25^2 \cdot 3\,611\,000}{2 \cdot 50} \;\approx\; 2257\ \text{s}.$$

Das h^2 ist der Haken. Die Maschenweite steht **im Quadrat**. Halbiert man sie — verdoppelt also die Knotenzahl, um ein feineres Bild zu bekommen —, dann viertelt sich der erlaubte Zeitschritt. Rechnen wir es für $h = 0{,}125$ m (9 statt 5 Knoten) nach:

$$\Delta t_{\max} \;=\; \frac{0{,}125^2 \cdot 3\,611\,000}{2 \cdot 50} \;\approx\; 564\ \text{s} \qquad (\text{ein Viertel von } 2257\ \text{s}).$$

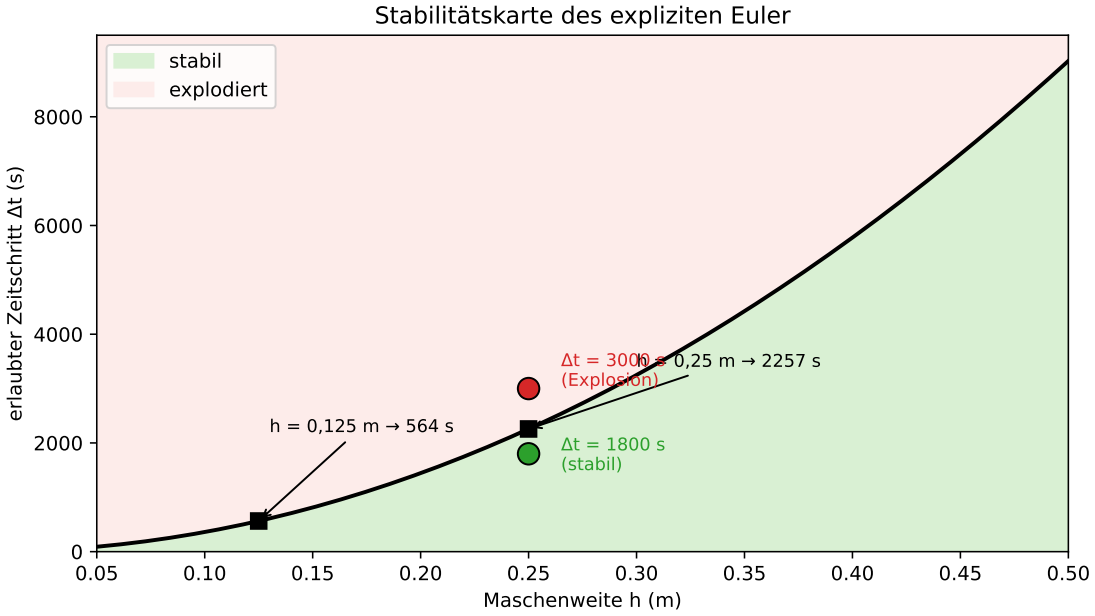

Abbildung 9.6.: Die Stabilitätskarte des expliziten Euler: erlaubter Zeitschritt gegen Maschenweite. Unter der Kurve (grün) ist der Schritt stabil, darüber (rot) explodiert er. Die Kurve fällt mit dem Quadrat der Maschenweite. Eingezeichnet: die beiden Experimentpunkte bei h = 0,25 m — 1800 s liegt sicher im Grünen, 3000 s im Roten.

Was man hier sieht: eine Landkarte mit einer grünen und einer roten Zone, getrennt durch die Faustregel-Kurve. Wer ein feineres Netz will, rutscht auf der Kurve nach links — und der erlaubte Zeitschritt stürzt überproportional ab. Ein doppelt so feines Netz braucht nicht doppelt, sondern viermal so viele Zeitschritte, und jeder ist auch noch teurer. Feine explizite Rechnungen werden so schnell unbezahlbar. Das ist der Grund, aus dem es das nächste Verfahren gibt.

9.5. Die implizite Rettung

Der explizite Schritt fragt: „Wie schnell ändern sich die Dinge **jetzt**?" und schreibt diese Rate für den ganzen Schritt fest. Das ist der Fehler: Am Anfang des Schritts ist die Rate am größten, und wenn der Schritt zu lang ist, überschießt sie. Der **implizite Euler-Schritt** stellt die Frage anders: „Welche neuen Werte passen so zusammen, dass die Änderungs- rate **am Ende** des Schritts stimmt?"

Das klingt nach einer Henne-Ei-Frage — die neue Rate hängt von den neuen Werten ab, die wir doch erst suchen. Und genau das ist der Punkt: Man kann die Knoten nicht mehr einzeln nacheinander ausrechnen, sondern nur noch **alle gleichzeitig**. Für jeden inneren Knoten i lautet die Forderung

$$\left(\frac{C_i}{\Delta t} + 2k\right) T_i^{\text{neu}} \;-\; k\,T_{i-1}^{\text{neu}} \;-\; k\,T_{i+1}^{\text{neu}} \;=\; \frac{C_i}{\Delta t}\,T_i^{\text{alt}}.$$

Jede Zeile koppelt einen Knoten an seine zwei Nachbarn — das ist ein **tridiagonales Gleichungssystem**, dieselbe Bandstruktur wie beim stationären Träger in Kapitel 7. Und dafür haben wir längst das perfekte Werkzeug: den **Thomas-Algorithmus** aus 2. Ein impliziter Zeitschritt ist also ein Thomas-Lauf. Er kostet mehr als die simple Division des expliziten Schritts — aber er hat eine Eigenschaft, die alles aufwiegt: Er ist bei **jedem** Zeitschritt stabil. Kein Δt ist zu groß; die Explosion kann nicht passieren.

> ⚠ Vermutung aufgelöst
>
> Die Vermutung vom Kapitelanfang — „der Zeitschritt ist nur eine Frage der Genauigkeit" — ist beim **expliziten** Verfahren schlicht falsch: Jenseits der scharfen Grenze explodiert es. Beim **impliziten** Verfahren wird sie dagegen fast wahr: Stabil ist es immer, und ein größerer Schritt macht das Ergebnis nur ungenauer, nicht kaputt. „Fast", weil ungenau hier etwas Bestimmtes heißt — die Front verschmiert. Das schauen wir uns als Letztes an.

9.5.1. Stabil heißt nicht genau

Man könnte nun übermütig werden: Wenn implizit immer stabil ist, nehme ich eben einen riesigen Zeitschritt und bin in einem Sprung fertig. Stabil bleibt es — aber genau bleibt es nicht. Grobe Schritte **verschmieren die Front**: Die scharfe Kante zwischen warm und kalt, die durch den Träger wandert, wird weichgezeichnet.

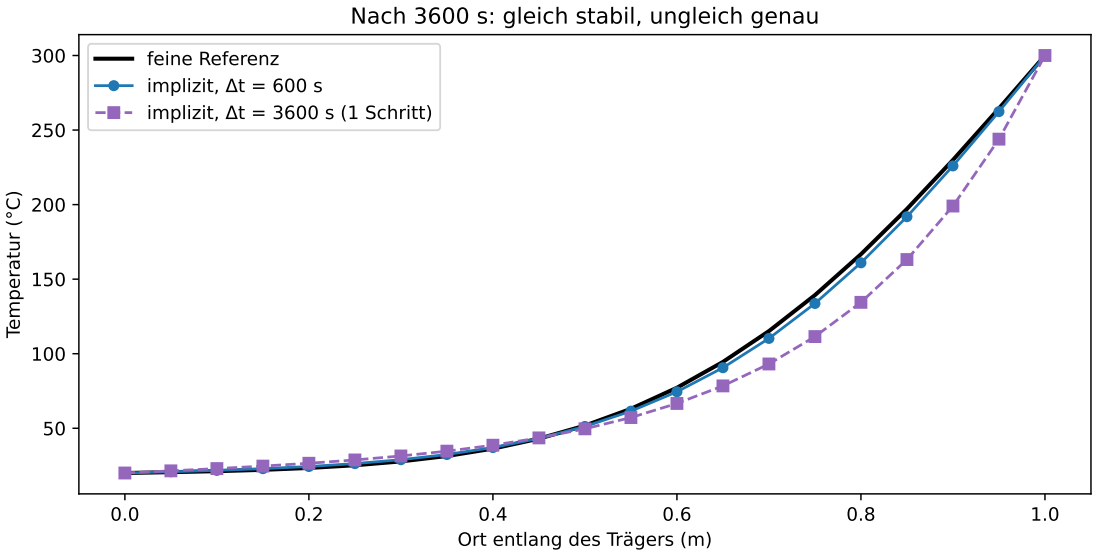

Abbildung 9.7.: Genauigkeit ist nicht Stabilität. Alle drei Kurven sind nach einer Stunde (3600 s) auf demselben 21-Knoten-Träger gerechnet und alle drei stabil. Die feine Referenz (schwarz, winzige Schritte) zeigt die echte, noch recht scharfe Front. Der implizite Lauf mit mittlerem Δt (blau) trifft sie gut; der eine Riesenschritt (lila) hat die Front verschmiert — dieselbe Endzeit, dieselbe Stabilität, weniger Genauigkeit.

Was man hier sieht: drei stabile Rechnungen, ein und dieselbe Endzeit. Die feine Referenz zeigt die echte Front — rechts schon deutlich warm, links noch kalt, mit einem klaren Übergang. Der mittlere Zeitschritt (blau) liegt fast auf ihr. Der eine Riesenschritt (lila) dagegen hat die Front verschmiert: Er unterschätzt die Wärme rechts und schmiert sie zu weit nach links. Kein Absturz, kein Zappeln — nur eben ungenau. Genau davor warnt die aufgelöste Vermutung: Stabilität ist geschenkt, Genauigkeit nicht.

9.5.2. Explizit oder implizit — was nehmen?

	explizit	implizit
Kosten je Schritt	eine Division je Knoten	ein Gleichungssystem (Thomas)
Zeitschritt	an die Maschenweite gefesselt (h^2)	frei wählbar
Stabilität	nur unter Δt_{max}	immer stabil
Gefahr bei groß Δt	Explosion	nur Front verschmiert
gut, wenn …	Netz grob, Schritt klein gewollt	Netz fein oder große Schritte gewollt

Abbildung 9.8.: Die Entscheidungstabelle: explizit gegen implizit. Der explizite Schritt ist billig, aber sein Zeitschritt ist an die Netzfeinheit gefesselt. Der implizite Schritt kostet pro Schritt ein Gleichungssystem, darf dafür aber so große Schritte machen, wie es die gewünschte Genauigkeit erlaubt.

Was man hier sieht: kein klarer Sieger, sondern ein Tausch. Wer ein grobes Netz hat und ohnehin kleine Schritte will, fährt mit dem billigen expliziten Verfahren gut. Wer ein feines Netz braucht (und darum an die h^2-Fessel gerät) oder große Schritte machen möchte, nimmt das implizite: Sein Mehraufwand je Schritt zahlt sich aus, weil er unvergleichlich weniger Schritte braucht. Genau deshalb rechnen die meisten professionellen Programme transiente Wärme **implizit** — die Stabilität ist ihnen den Preis wert.

9.6. Zwanzig Stunden — und der Brandschutz rechnet in Minuten

Eine Zahl fehlt noch, und sie erklärt, warum in jeder Animation dieses Kapitels „Zeitraffer” steht. Wie lange dauert das Aufheizen überhaupt? Die Antwort steckt in einer einzigen Kombination der Trägergrößen, der **Zeitkonstante**:

$$\tau \;=\; \frac{L^2\,\rho\,c}{\lambda} \;=\; \frac{1{,}0^2 \cdot 3\,611\,000}{50} \;=\; 72\,220\ \mathrm{s} \;\approx\; 20\ \text{Stunden}.$$

Ehrlich gerechnet: die Einheiten

L^2 hat die Einheit m^2, ρc ist $\mathrm{J/(m^3\,K)}$, λ ist $\mathrm{W/(m\,K)} = \mathrm{J/(s\,m\,K)}$. Alles zusammengesetzt kürzen sich Meter, Joule und Kelvin weg, und übrig bleiben **Sekunden** — wie es sich für eine Zeit gehört. Das ist keine erfundene Formel, sondern die natürliche Zeitskala, die aus der Wärmeleitungsgleichung selbst fällt.

Rund zwanzig Stunden also, bis unser Stahlträger über seine ganze Länge eingeschwungen ist. In Echtzeit wäre das eine sehr langweilige Animation. Darum lief der Film am Kapitelanfang im Zeitraffer — zehn Stunden in wenigen Sekunden, mit der eingeblendeten Uhr, die die echten Stunden zählt, ganz wie die Zeitraffer-Uhr aus Kapitel 1. Ehrlich beschriftet ist der Zeitraffer kein Trick, sondern eine Notwendigkeit.

Und genau hier steckt die eigentliche Pointe für den Brandschutz — sie verlangt einen eigenen Kasten.

R30, R60, R90 — Brandschutz rechnet in Minuten

Bauteile werden nach ihrer **Feuerwiderstandsdauer** eingeteilt: **R30**, **R60**, **R90** heißt, das Bauteil hält im Normbrand 30, 60 oder 90 **Minuten**, bevor es seine Tragfähigkeit verliert (CEN (Europäisches Komitee für Normung) 2005). Minuten — nicht Stunden. Wie weit ist die Wärme in dieser Zeit überhaupt in den Träger vorgedrungen? Das schätzt die **Eindringtiefe** $\sqrt{a\,t}$ mit der Temperaturleitfähigkeit $a = \lambda/(\rho c) = 1{,}385 \cdot 10^{-5}\ \mathrm{m^2/s}$:

$$\sqrt{a \cdot 30\ \mathrm{min}} = \sqrt{1{,}385 \cdot 10^{-5} \cdot 1800} \ \approx\ 0{,}16\ \mathrm{m}\ =\ 16\ \mathrm{cm}.$$

Nach einer halben Stunde ist die Hitze also erst rund 16 cm weit gekrochen, nach 90 Minuten (R90) rund 27 cm — der ganze Träger ist einen Meter lang. Das kühle Auflager am fernen Ende bleibt in der gesamten brandschutzrelevanten Zeit kalt. Die Zeitkonstante $\tau \approx 20\ \mathrm{h}$ sagt, wann der *ganze* Träger durchgewärmt wäre; der Brandschutz aber entscheidet sich in den ersten **Minuten**, nahe dem Brandherd. Beide Uhren gehören zusammen — und beide muss man kennen.

Ein Vorgriff: warum später 1D genügt

Dieselbe Zeitkonstante lässt sich auch über die **Höhe** des Trägers bilden. Er ist nur $H = 0{,}1$ m hoch, und H geht im Quadrat ein:

$$\tau_{\text{Höhe}} = \frac{H^2\,\rho\,c}{\lambda} = \frac{0{,}1^2 \cdot 3\,611\,000}{50} \approx 722\ \text{s} \approx 12\ \text{min}.$$

Quer über die Höhe gleicht sich ein Temperaturunterschied also in etwa 12 Minuten aus — rund hundertmal schneller als über die Länge (20 Stunden). Genau deshalb dürfen wir die Wärme im Träger überhaupt als **1D**-Problem entlang der Länge rechnen: quer ist der Träger praktisch immer schon ausgeglichen. Diesen Gedanken nehmen wir in 1 wieder auf, wenn wir in die Fläche gehen.

9.7. Die interaktive Einheit: die Zeitmaschine

Jetzt drehst du selbst an der Uhr. Die **Zeitmaschine** rechnet den transienten Träger mit beiden Verfahren; du stellst Zeitschritt, Knotenzahl und Verfahren ein und siehst zweierlei: das Temperaturprofil am Ende und die Temperatur des Mittenknotens über der ganzen Zeit. Der Thomas-Löser aus 2 ist unten hineinkopiert, weil der Browser nicht aus den Kapitel-Ordnern importieren kann.

Vorhersage-Punkt

Bevor du ausführst: Die Zeitmaschine startet mit 5 Knoten, explizit und $\Delta t = 1800$ s — sauber unter der Grenze von 2257 s, also stabil. Ändere jetzt in der ersten Zeile `ZEITSCHRITT_S = 1800.0` auf `3000.0` und führe erneut aus. Was zeigt die Kurve des Mittenknotens dann — läuft sie glatt auf ihren Endwert, oder tut sie etwas anderes? Leg dich fest, bevor du es probierst.

```
#| autorun: true
# ====== Die Zeitmaschine – dreh an den Reglern ======
ZEITSCHRITT_S = 1800.0       # der Zeitschritt Delta t in Sekunden
KNOTENZAHL = 5               # Zahl der Knoten (hoechstens 50)
VERFAHREN = "explizit"       # "explizit" oder "implizit"
GESAMTZEIT_MIN = 600.0       # simulierte Dauer in Minuten
# ====================================================

# Der Traeger des Buches (Baustahl S235)
WAERMELEITFAEHIGKEIT = 50.0
QUERSCHNITT_M2 = 0.01
TRAEGERLAENGE_M = 1.0
DICHTE = 7850.0
```

```
WAERMEKAPAZITAET = 460.0
TEMPERATUR_LINKS = 20.0
TEMPERATUR_RECHTS = 300.0


def thomas(unter, diagonale, ober, rechte_seite):
    """Loest ein tridiagonales System (Thomas-Loeser aus Kapitel 4)."""
    n = len(diagonale)
    ober_neu = list(ober)
    seite_neu = list(rechte_seite)
    ober_neu[0] = ober[0] / diagonale[0]
    seite_neu[0] = rechte_seite[0] / diagonale[0]
    for i in range(1, n):
        nenner = diagonale[i] - unter[i] * ober_neu[i - 1]
        if i < n - 1:
            ober_neu[i] = ober[i] / nenner
        seite_neu[i] = (rechte_seite[i] - unter[i] * seite_neu[i - 1]) / nenner
    loesung = [0.0] * n
    loesung[n - 1] = seite_neu[n - 1]
    for i in range(n - 2, -1, -1):
        loesung[i] = seite_neu[i] - ober_neu[i] * loesung[i + 1]
    return loesung


# Waerme-Steifigkeit und gelumpte Eimer aus den Traegergroessen.
elementlaenge = TRAEGERLAENGE_M / (KNOTENZAHL - 1)
k = WAERMELEITFAEHIGKEIT * QUERSCHNITT_M2 / elementlaenge
elementkapazitaet = DICHTE * WAERMEKAPAZITAET * QUERSCHNITT_M2 * elementlaenge
kapazitaeten = []
for knoten in range(KNOTENZAHL):
    kapazitaeten.append(elementkapazitaet)
kapazitaeten[0] = elementkapazitaet / 2.0
kapazitaeten[KNOTENZAHL - 1] = elementkapazitaet / 2.0

# Startzustand: alles kalt, nur das rechte Ende beim Brandherd.
temperaturen = []
for knoten in range(KNOTENZAHL):
    temperaturen.append(TEMPERATUR_LINKS)
temperaturen[KNOTENZAHL - 1] = TEMPERATUR_RECHTS


def expliziter_schritt(werte):
    """Ein expliziter Euler-Schritt: neuer Wert aus der jetzigen Rate."""
```

```
    neu = list(werte)
    for i in range(1, KNOTENZAHL - 1):
        zufluss = k * (werte[i - 1] - 2.0 * werte[i] + werte[i + 1])
        neu[i] = werte[i] + ZEITSCHRITT_S / kapazitaeten[i] * zufluss
    return neu


def impliziter_schritt(werte):
    """Ein impliziter Euler-Schritt: ein tridiagonales System, mit Thomas geloest."""
    innere = KNOTENZAHL - 2
    unter = [0.0] * innere
    diagonale = [0.0] * innere
    ober = [0.0] * innere
    rechte = [0.0] * innere
    for platz in range(innere):
        i = platz + 1
        traegheit = kapazitaeten[i] / ZEITSCHRITT_S
        diagonale[platz] = traegheit + 2.0 * k
        if platz > 0:
            unter[platz] = -k
        if platz < innere - 1:
            ober[platz] = -k
        rechte[platz] = traegheit * werte[i]
    rechte[0] = rechte[0] + k * werte[0]
    rechte[innere - 1] = rechte[innere - 1] + k * werte[KNOTENZAHL - 1]
    innen = thomas(unter, diagonale, ober, rechte)
    return [werte[0]] + innen + [werte[KNOTENZAHL - 1]]


# Die Zeit durchlaufen und den Mittenknoten mitschreiben.
gesamtzeit = GESAMTZEIT_MIN * 60.0
schrittzahl = int(gesamtzeit / ZEITSCHRITT_S)
mitte = KNOTENZAHL // 2
mittenverlauf = [temperaturen[mitte]]
zeitpunkte = [0.0]
for schritt in range(schrittzahl):
    if VERFAHREN == "implizit":
        temperaturen = impliziter_schritt(temperaturen)
    else:
        temperaturen = expliziter_schritt(temperaturen)
    mittenverlauf.append(temperaturen[mitte])
    zeitpunkte.append((schritt + 1) * ZEITSCHRITT_S / 60.0)
```

```
print("Verfahren:", VERFAHREN, " Zeitschritt:", ZEITSCHRITT_S, "s")
print("Mittenknoten am Ende: %.1f Grad" % temperaturen[mitte])

# --- Abbildung ---
import matplotlib.pyplot as plt
orte = [i * elementlaenge for i in range(KNOTENZAHL)]
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10.0, 3.6))
ax1.plot(orte, temperaturen, "o-", color="tab:red", markeredgecolor="black")
ax1.set_xlabel("Ort (m)")
ax1.set_ylabel("Temperatur (°C)")
ax1.set_title("Profil am Ende")
ax2.plot(zeitpunkte, mittenverlauf, "-", color="tab:blue")
ax2.set_xlabel("Zeit (min)")
ax2.set_ylabel("Mittenknoten (°C)")
ax2.set_title("Die Mitte über der Zeit")
plt.tight_layout()
plt.show()
```

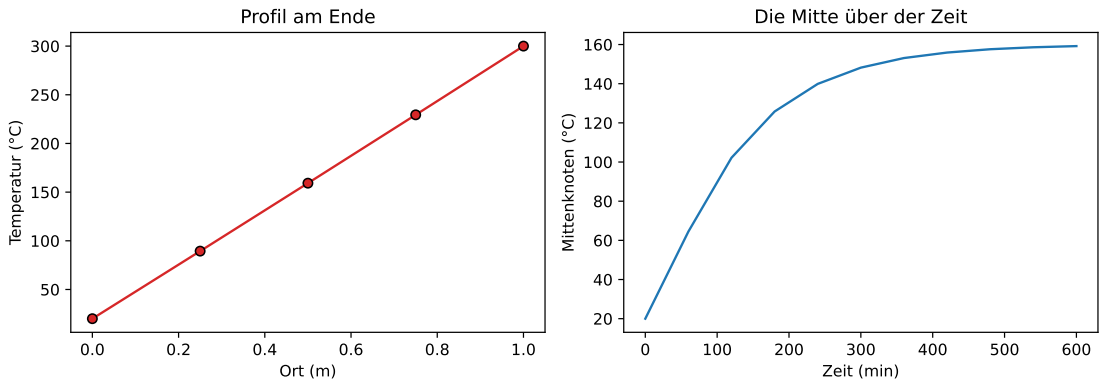

Abbildung 9.9.: Vorgerenderte Fassung der Zeitmaschine mit der Startbelegung (5 Knoten, explizit, Δt = 1800 s): links das eingeschwungene Profil, rechts der Mittenknoten, der glatt auf seine 160 °C zuläuft. Im Browser wird dieses Bild durch dein eigenes Ergebnis ersetzt.

Was man hier sieht: links das vertraute Endprofil, die Gerade von 20 auf 300 °C, mit den rund 160 °C in der Mitte; rechts der Weg dorthin — der Mittenknoten steigt erst zögernd (die Wärme ist noch nicht da), dann zügig, dann immer flacher, bis er sich bei 160 °C anlegt. Stell in der ersten Zeile `ZEITSCHRITT_S = 3000.0` ein, und diese glatte Kurve wird zur zackigen Explosion. Schalte danach auf `VERFAHREN = "implizit"` um — und selbst mit $\Delta t = 3000$ s ist wieder Ruhe.

9.8. Das Kapitel-Programm

Das vollständige, eigenständige Programm zu diesem Kapitel liegt in `programme/kap09/kap09_transiente_fem.py`. Es baut die gelumpten Kapazitäten, bietet `ein_expliziter_schritt` und `ein_impliziter_schritt` (letzterer importiert den Thomas-Löser aus `programme/gemeinsam/loeser.py`) und rechnet beim Aufruf die Handrechnung, die Explosion und die Frontverschmierung nach. Die im Kapitel gezeigten Zahlen stammen alle aus diesem Programm.

Merkkasten

- Der **Speicherterm** $\rho c\,\partial T/\partial t$ ist die Bilanz aus Kapitel 6, bei der die Differenz zwischen Zu- und Abfluss nicht mehr null ist, sondern das Stück aufheizt. $\rho c = 3{,}61\ \mathrm{MJ/(m^3\,K)}$ ist der Preis der Trägheit.
- Der Speicherterm wird zur **Massenmatrix**. **Lumping** macht sie diagonal: Jeder Knoten bekommt seinen Eimer — innen 9027,5 J/K, am Rand 4513,75 J/K.
- **Expliziter Euler:** neuer Wert = alter Wert + Zeitschritt · Änderungs- rate. Billig (eine Division je Knoten), aber nur unter $\Delta t_{\max} = h^2\rho c/(2\lambda) \approx 2257$ s (rund 38 min) stabil — darüber **explodiert** die Rechnung.
- **Halbe Maschenweite → ein Viertel des Zeitschritts** (das h^2).
- **Impliziter Euler:** alle neuen Werte zugleich aus einem tridiagonalen System (Thomas). Immer stabil, aber grobe Schritte verschmieren die Front. Stabilität ≠ Genauigkeit.

Rückblick auf die Landkarte

Mit diesem Kapitel endet Teil III — und der Baustein **Physik** unserer Landkarte ist damit vollständig. Wir sind von der Steigung und der Fläche (Kapitel 5) über die Bilanz (Kapitel 6) und die schwache Form (Kapitel 7) bis zu den Randbedingungen (Kapitel 8) gekommen, und jetzt tickt auch die Uhr (dieses Kapitel). Der Träger kann stationär *und* transient gerechnet werden.

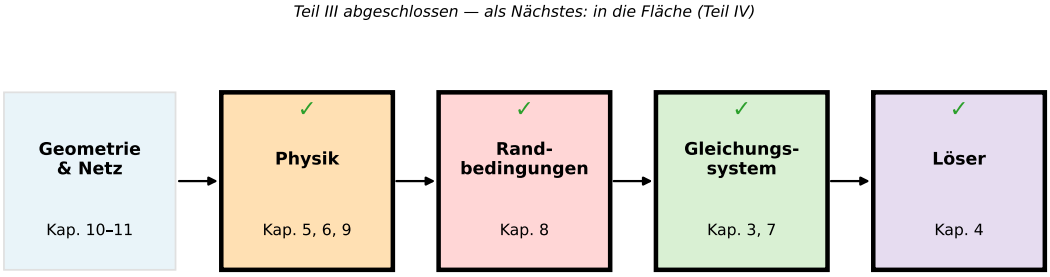

Abbildung 9.10.: Die Landkarte des Buches am Ende von Teil III: Der Baustein „Physik" ist jetzt komplett ausgefüllt (kräftig hervorgehoben) — stationär und transient. Der Löser (Kap. 4) steht ebenfalls. Was noch fehlt, ist die Fläche: Geometrie und Netz jenseits der 1D-Kette (Teil IV).

Was man hier sieht: dieselbe Landkarte wie in Kapitel 1, aber vier von fünf Kästen tragen jetzt ihr Häkchen. Nur der erste — Geometrie und Netz — ist noch blass: Bisher war unser „Netz" eine simple Kette aus fünf Knoten auf einer Linie. Teil IV bricht in die Fläche auf, mit Dreiecken und Rechtecken, und füllt auch diesen letzten Kasten.

Roter Faden

Zurück: Der Speicherterm ist die Bilanz aus Kapitel 6, diesmal mit der Differenz, die aufheizt. Die Massenmatrix wird nach derselben Halbe-Halbe-Logik gebaut wie die Wärmequelle in Kapitel 7. Der implizite Schritt ist ein tridiagonales System und damit ein Fall für den Thomas-Algorithmus aus 2. Die festen Randtemperaturen behandeln wir wie in Kapitel 8. Und die explizite Zeitschleife aus der Blackbox von Kapitel 1 ist jetzt kein Geheimnis mehr.

Vor: Lumping und Zeitschritte laufen in 2D unverändert weiter (1 und 3). Die Explosion ist der Grund, aus dem das große Finale (3) implizit rechnet. Und die Frontverschmierung wird in Kapitel 12 zu einem Prüffall: Woher weiß man, ob ein „schön aussehendes" Ergebnis nicht bloß eine grob verschmierte Näherung ist?

Übungen

Ü 9.1 (Verstehen). Führe die Handrechnung aus Abbildung 9.4 um einen **vierten** Schritt fort (von „nach 1800 s" auf „nach 2400 s"), mit denselben Zahlen $\Delta t = 600$ s, $k = 2$ W/K, $C = 9027{,}5$ J/K. Rechne die drei inneren Knoten von Hand und prüfe mit der Zeitmaschine (5 Knoten, explizit, $\Delta t = 600$, 4 Schritte).

> **Musterlösung zu Ü 9.1**
>
> Vorfaktor wie gehabt $\frac{600}{9027{,}5} \cdot 2 \approx 0{,}1329$. Aus dem Zustand nach 1800 s, $[20;\ 20{,}658;\ 32{,}212;\ 105{,}262;\ 300]$:
>
> - Knoten 2: $20{,}658 + 0{,}1329 \cdot (20 - 2 \cdot 20{,}658 + 32{,}212) = 22{,}106$ °C,
> - Knoten 3: $32{,}212 + 0{,}1329 \cdot (20{,}658 - 2 \cdot 32{,}212 + 105{,}262) = 40{,}386$ °C,
> - Knoten 4: $105{,}262 + 0{,}1329 \cdot (32{,}212 - 2 \cdot 105{,}262 + 300) = 121{,}438$ °C.
>
> Ergebnis nach 2400 s: $[20;\ 22{,}11;\ 40{,}39;\ 121{,}44;\ 300]$. Die Wärme kriecht weiter nach links; Knoten 4 hat gerade die 120 °C überschritten.

Ü 9.2 (Verändern). Verdopple die Zahl der Elemente (5 → 9 Knoten, h von 0,25 auf 0,125 m) und kreise mit der Zeitmaschine den größten noch stabilen Zeitschritt experimentell ein: Erhöhe Δt Schritt für Schritt, bis die Kurve des Mittenknotens explodiert. Vergleiche mit der Faustregel. Stimmt die Viertel-Regel?

> **Musterlösung zu Ü 9.2**
>
> Die Faustregel liefert für 9 Knoten $\Delta t_{\max} = h^2 \rho c / (2\lambda) = 0{,}125^2 \cdot 3\,611\,000/100 \approx 564$ s — ein Viertel der 2257 s des 5-Knoten-Trägers, weil h im Quadrat steht. Experimentell bleibt der Lauf bis knapp über diese Grenze ruhig und kippt kurz darüber. Das Lösungsskript `loesungen/kap09_ue2.py` kreist beide Grenzen ein und bestätigt das Verhältnis $4 : 1$.

Ü 9.3 (Übertragen). Tausche das Material von Baustahl auf **Aluminium** ($\lambda = 200$ W/(m K), $\rho = 2700$ kg/m^3, $c = 900$ J/(kg K)). Rechne **zuerst** von Hand die Zeitkonstante $\tau = L^2 \rho c / \lambda$ und die Faustregel $\Delta t_{\max}$ aus, **dann** simuliere. Wird ein Aluminiumträger schneller oder langsamer warm — und darf der explizite Schritt größer oder kleiner sein als bei Stahl?

> **Musterlösung zu Ü 9.3**
>
> $\rho c = 2700 \cdot 900 = 2{,}43\ \mathrm{MJ/(m^3\,K)}$, also $\tau = 1 \cdot 2\,430\,000/200 = 12\,150$ s $\approx 3{,}4$ Stunden — fast sechsmal flinker als Stahl (Aluminium leitet die Wärme viel besser). Die Faustregel: $\Delta t_{\max} = 0{,}25^2 \cdot 2\,430\,000/(2 \cdot 200) \approx 380$ s, also fast sechsmal *kleiner* als bei Stahl — das große λ steht im Nenner. Beides folgt aus derselben Ursache: Aluminium leitet gut, also flitzt die Wärme schnell und muss in kleineren Zeitsprüngen verfolgt werden. Das Lösungsskript `loesungen/kap09_ue3.py` rechnet und simuliert das.

Das Kleingedruckte

Woher die konsistente Massenmatrix kommt. Die 6 und die $2, 1$ im Ergebnis-Kasten sind die exakten Flächen unter den Produkten zweier Hütchen. Über ein Element der Länge h ist $\int N_i N_i \,\mathrm{d}x = h/3$ (die Fläche unter einem quadratischen Hütchenquadrat) und $\int N_i N_j \,\mathrm{d}x = h/6$ für Nachbarn. Mal $\rho c A$ ergibt das $\frac{\rho c A h}{6}\left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$. Das Lumping addiert je Zeile zusammen und schreibt die Summe auf die Diagonale — massenerhaltend, wie die Eimer zeigen. Wer die Flächen selbst nachrechnen will, findet das Werkzeug in Kapitel 5.

Was dieses Kapitel bewusst weglässt. Es gibt ein Verfahren genau zwischen explizit und implizit — das **Crank-Nicolson-Verfahren**, das die Änderungsrate zur Hälfte am Anfang und zur Hälfte am Ende ansetzt und dadurch genauer wird. Es ist der übliche Kompromiss in der Praxis; wir erwähnen es nur. Ebenso haben wir die Stabilitätsgrenze als **Faustregel** benutzt, nicht als exakten Beweis. Für die kleine 5-Knoten-Kette liegt die echte Grenze etwas höher (rund 2640 s, also gut 44 Minuten) als die 2257 s (rund 38 Minuten) der Faustregel — die Faustregel ist bewusst auf der sicheren Seite. Wer es genau wissen will, braucht die sogenannte von-Neumann-Stabilitätsanalyse; sie steht nicht auf dem Weg dieses Buches. Auch die feine Frage, ob die *konsistente* Massenmatrix bei sehr kleinen Zeitschritten leicht überschwingt, lassen wir beim Eimerbild bewenden: Gelumpt ist robust, und robust genügt uns.

Teil IV.

Teil IV — In die Fläche

10. In die Fläche: Dreiecke

Ein Mosaik pflastert jede Wand — krumme Bögen, Ecken, Löcher —, indem es kleine Steinchen dicht an dicht legt. Ein Stadiondach, eine geodätische Kuppel, das Drahtgittermodell eines Kinofilmgesichts: Schaut man genau hin, sind sie aus **Dreiecken** gebaut. Das ist kein Zufall, sondern der Grund, warum Dreiecke das Lieblingswerkzeug der Finite-Elemente-Methode sind: Mit genug Dreiecken lässt sich *jede* Form auslegen.

Bis hierher lebte unser Träger auf einer **Linie** — fünf Knoten, vier Elemente, eine Zahl pro Ort. In diesem Kapitel verlässt er die Linie und wird zur **Fläche**. Und das Beste kommt gleich vorweg: Es ist dasselbe Spiel. Alles, was du seit Kapitel 3 gelernt hast — Steckbrief, Assemblieren, Randwerte, Lösen —, funktioniert unverändert weiter. Es ändert sich nur eine einzige Frage: **Wie interpoliert man auf einem Dreieck?**

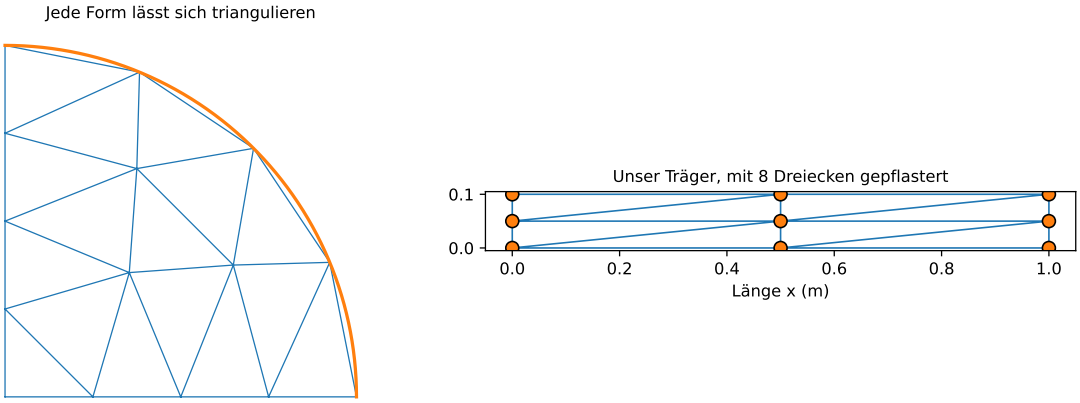

Abbildung 10.1.: Warum Dreiecke. Links eine krumme Form (ein Viertelkreis), mit Dreiecken ausgelegt — die Ränder werden treppchenweise erfasst, mit mehr Dreiecken beliebig genau. Rechts derselbe Gedanke für unseren Träger: das grobe 8-Dreiecke-Netz dieses Kapitels. Drei Punkte legen immer eine Ebene fest, darum ist das Dreieck der einfachste Flächen-Baustein, den es gibt.

Was man hier sieht: links eine gekrümmte Form, aus lauter Dreiecken

gelegt — je feiner die Dreiecke, desto glatter der Rand. Rechts der Träger dieses Kapitels: neun Knoten, acht Dreiecke, klein genug, dass wir sein Gleichungssystem gleich noch von Hand anschauen. Beide Male gilt: Drei Punkte spannen genau eine Ebene auf, und über einer Ebene lässt sich kinderleicht interpolieren. Das ist der ganze Trick.

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum in der Fläche **Dreiecke** die natürlichen Bausteine sind — jede Form lässt sich triangulieren, und drei Ecken legen eine Ebene fest,
2. … einen Punkt im Dreieck durch seine drei **Flächenanteile** beschreiben und damit zwischen drei Eckwerten interpolieren,
3. … die Ansatzfunktion eines Knotens als **Zelt über seinen Dreiecken** beschreiben — die flächenhafte Fortsetzung des Hütchens aus Kapitel 7,
4. … ein Netz als **Knotenliste plus Dreiecksliste** lesen und die **3×3-Elementmatrix** eines Dreiecks aus den Steigungen seiner Zeltflächen aufstellen,
5. … am 8-Dreiecke-Träger die **Assemblierung** nachvollziehen, das System mit dem Löser aus 2 knacken und die Bandstruktur deuten.

⚠ Naheliegende Vermutung

Vermutung: *„Zwei Dimensionen sind grundsätzlich komplizierter als eine. Für die Fläche braucht man ganz neue Mathematik — die Werkzeuge aus dem 1D-Teil kann ich vergessen."*
Warum sie naheliegt: Eine Fläche hat mehr Richtungen, mehr Nachbarn, mehr zu bedenken als eine Linie. Es *fühlt* sich nach einem großen Sprung an, und in vielen Lehrbüchern beginnt hier ein neues, dickeres Kapitel voller Indizes.
Was stattdessen stimmt: Die Rezeptur ist Wort für Wort dieselbe wie in 1D — Ansatz, Elementmatrix, Assemblierung, Randwerte, Lösen. Es kommt genau **ein** neuer Baustein hinzu: Interpolation mit **Flächenanteilen** statt Streckenanteilen. Eine Idee, nicht zehn. Wer Kapitel 7 verstanden hat, hat den Berg schon bestiegen; dieses Kapitel ist der Spaziergang auf dem Kamm.

10.1. Der Träger als Fläche: die Geometrie festlegen

Bevor wir rechnen, klären wir, *was* die Fläche eigentlich zeigt. Unser Träger ist ein Quader: 1 m lang, 0,1 m hoch, 0,1 m tief. In 1D haben wir ihn auf seine **Länge** zusammengedrückt und Höhe und Tiefe in der Querschnittsfläche $A = 0{,}01$ m^2 versteckt. Jetzt lösen wir eine der beiden versteckten Richtungen auf: Wir schauen von der **Seite** auf den Träger.

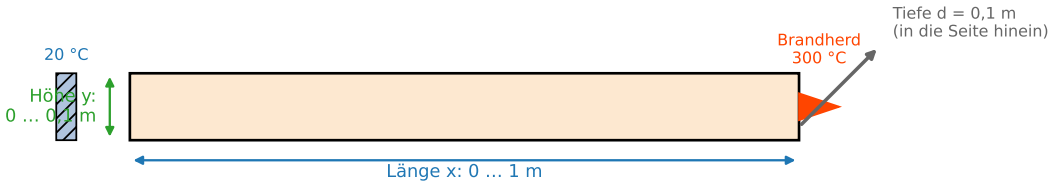

Abbildung 10.2.: Die Geometrie-Konvention des ganzen restlichen Buches. Wir sehen den Träger von der Seite: x ist die Länge (0 bis 1 m), y die Höhe (0 bis 0,1 m). Die dritte Richtung, die Tiefe d = 0,1 m, steht senkrecht auf dem Papier (grauer Pfeil). Höhe mal Tiefe ergibt weiterhin die 1D-Querschnittsfläche A = 0,1 m · 0,1 m = 0,01 m².

Was man hier sieht: den Träger von der Seite. Die **Länge** x läuft von links (kühles Auflager, 20 °C) nach rechts (Brandherd, 300 °C), die **Höhe** y von unten nach oben. Die **Tiefe** $d = 0{,}1$ m zeigt in die Papierebene hinein — sie bleibt in diesem Buch immer gleich. Damit stimmt alles mit dem 1D-Teil zusammen: Höhe mal Tiefe ist dieselbe Querschnittsfläche wie in 1D, $0{,}1 \cdot 0{,}1 = 0{,}01$ m$^2 = A$.

Merke: die Geometrie-Konvention

Das 2D-Modell ist die **Seitenansicht** des Trägers:

- x = Länge, von 0 bis 1 m,
- y = Höhe, von 0 bis 0,1 m,
- Tiefe d = 0,1 m (senkrecht zur Zeichenebene, immer konstant).

Wo in 1D die ganze Querschnittsfläche $A = 0{,}01$ m^2 als ein Faktor auftrat, tritt in 2D nur noch die **Tiefe** $d = 0{,}1$ m auf — die Höhe ist jetzt eine echte Richtung des Netzes und keine versteckte Zahl mehr. Diese Konvention gilt bis zum Finale (Kapitel 14, 3).

Jetzt, wo die Höhe eine echte Richtung des Netzes wird, dürfen wir eine Frage einlösen, die im 1D-Teil offengeblieben war: Warum durften wir den Träger überhaupt als bloße Linie behandeln? In Kapitel 9 stand die Antwort schon als Vorgriff bereit — ohne ist sie, und mit dem 2D-Modell können wir sie gleich nachprüfen.

Warum durften wir 1D rechnen?

Der ganze dritte Teil hat den Träger als **Linie** gerechnet — eine Temperatur je Ort *entlang der Länge*, die Höhe zu einer bloßen Zahl in der Querschnittsfläche eingedampft. Das war kein Trick, sondern durfte sein, und der Grund steckt in den **Zeitkonstanten** aus Kapitel 9.
Ein Temperaturunterschied gleicht sich über eine Strecke ℓ in etwa der Zeit $\tau = \ell^2 \rho c/\lambda$ aus — die Strecke geht **im Quadrat** ein. Über die volle Länge $L = 1$ m des Trägers ist das

$$\tau_{\text{längs}} = \frac{L^2\,\rho\,c}{\lambda} = \frac{1^2 \cdot 7850 \cdot 460}{50} \approx 72\,220 \text{ s} \approx 20 \text{ h},$$

über die kleine Höhe $H = 0{,}1$ m dagegen

$$\tau_{\text{Höhe}} = \frac{H^2\,\rho\,c}{\lambda} = \frac{0{,}1^2 \cdot 7850 \cdot 460}{50} \approx 722 \text{ s} \approx 12 \text{ min}.$$

Das Verhältnis ist genau

$$\frac{\tau_{\text{längs}}}{\tau_{\text{Höhe}}} = \left(\frac{L}{H}\right)^2 = \left(\frac{1}{0{,}1}\right)^2 = 100:$$

Quer über die Höhe gleicht sich die Temperatur rund hundertmal schneller aus als längs. Ehe sich über die Länge etwas rührt, ist der Träger über seine Höhe längst durchgewärmt — auf jedem senkrechten Schnitt herrscht praktisch eine einzige Temperatur. Genau das rechtfertigt die 1D-Sicht: eine Zahl je Längsstelle genügt.
Und jetzt, mit dem Netz in der Fläche, müssen wir es nicht mehr glauben — wir können es **nachprüfen**. Gleich wird die Rechnung mit gleichem Rand links und rechts von selbst ein Feld liefern, das über die Höhe keinen Unterschied macht. Erst wenn die Brandgase die Höhe *ungleich* belasten — von unten heiß, von oben kühl —, wird der Höhenunterschied wichtig; das ist die Szene, die den Träger im Finale krümmt (Kapitel 14).

10.2. Vom Streckenanteil zum Flächenanteil

Erinnern wir uns, wie Interpolation in 1D funktioniert hat. Zwischen zwei Knoten teilt ein Punkt die Strecke in **zwei Anteile**: Liegt er bei einem Viertel des Weges, gehört er zu drei Vierteln dem linken und zu einem Viertel dem rechten Knoten. Genau das haben die Hütchen aus Kapitel 7 aufgeschrieben — der Anteil eines Knotens *ist* das Gewicht seines Wertes.

In der Fläche ist ein Punkt nicht mehr durch *eine* Zahl (den Streckenanteil) festgelegt, sondern von **drei** Ecken umgeben. Der Trick: Verbinde den Punkt P mit den drei Ecken. Das zerlegt das Dreieck in **drei Teildreiecke**, und die drei **Flächenanteile** übernehmen exakt die Rolle der Streckenanteile.

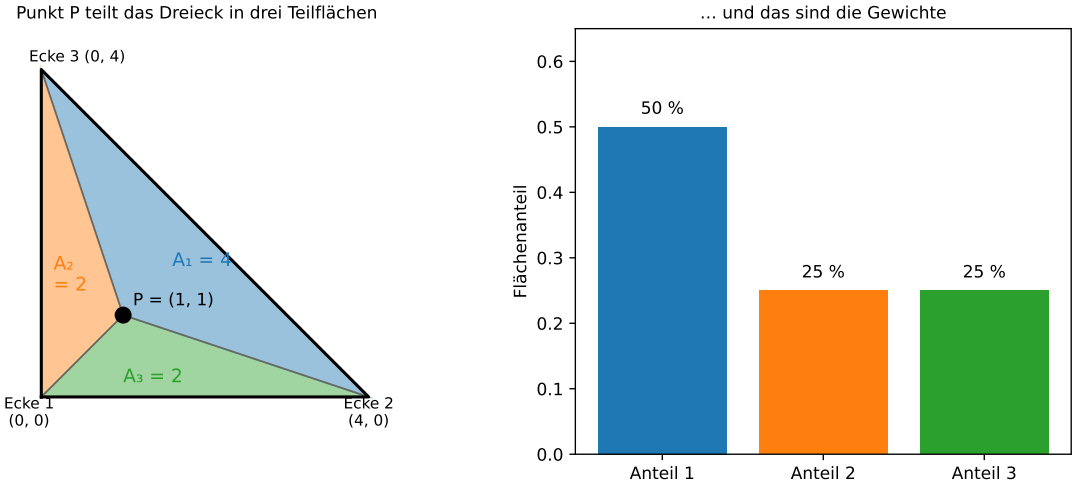

Abbildung 10.3.: Der einzige neue Baustein des Kapitels. Verbindet man den Punkt P mit den drei Ecken, zerfällt das Dreieck in drei farbige Teildreiecke. Der Anteil einer Ecke ist die Fläche des Teildreiecks GEGENÜBER von ihr, geteilt durch die Gesamtfläche. Rechts die drei Anteile als Balken — sie summieren sich immer zu 100 %.

Was man hier sieht: Der Punkt P zerteilt das Dreieck in drei Teildreiecke, jedes in einer Farbe. Die **Überkreuz-Logik** ist wichtig: Der Anteil, der zu Ecke 1 gehört, ist die Fläche des Teildreiecks **gegenüber** von Ecke 1 (das blaue, das die Ecke 1 gar nicht berührt). Warum gegenüber? Wenn P direkt auf Ecke 1 säße, wäre das gegenüberliegende Teildreieck das ganze Dreieck — Anteil 1 wäre 100 %. Wandert P weg von Ecke 1, schrumpft dieses Teildreieck, und der Anteil sinkt.

10.2.1. Rechnen mit echten Zahlen

Die Flächen berechnen wir mit einem Schulrezept, das ganz ohne Vektorprodukt auskommt — der **halben Kreuzdifferenz**. Für ein Dreieck mit den Ecken (x_1, y_1), (x_2, y_2), (x_3, y_3):

$$\text{Fläche} = \tfrac{1}{2}\,\big|\,(x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1)\,\big|.$$

In Worten: Spanne von der ersten Ecke aus die beiden Kanten auf, bilde „erste Kante breit mal zweite Kante hoch, minus über Kreuz", und nimm die Hälfte des Betrags. Für unser Dreieck mit $V_1 = (0,0)$, $V_2 = (4,0)$, $V_3 = (0,4)$:

$$A = \tfrac{1}{2}\,\big|(4-0)(4-0) - (0-0)(0-0)\big| = \tfrac{1}{2} \cdot 16 = 8.$$

Jetzt die drei Teildreiecke mit $P = (1,1)$, jedes mit derselben Formel:

$$\begin{aligned}
A_1 &= \text{Fläche}(P, V_2, V_3) = \tfrac{1}{2}\big|(4-1)(4-1) - (0-1)(0-1)\big| = \tfrac{1}{2}\,|9-1| = 4,\\
A_2 &= \text{Fläche}(P, V_3, V_1) = \tfrac{1}{2}\big|(0-1)(0-1) - (0-1)(4-1)\big| = \tfrac{1}{2}\,|1+3| = 2,\\
A_3 &= \text{Fläche}(P, V_1, V_2) = \tfrac{1}{2}\big|(0-1)(0-1) - (4-1)(0-1)\big| = \tfrac{1}{2}\,|1+3| = 2.
\end{aligned}$$

Die drei Flächenanteile sind die Teilflächen, geteilt durch die Gesamtfläche:

$$\lambda_1 = \frac{A_1}{A} = \frac{4}{8} = 0{,}5, \qquad \lambda_2 = \frac{A_2}{A} = \frac{2}{8} = 0{,}25, \qquad \lambda_3 = \frac{A_3}{A} = \frac{2}{8} = 0{,}25.$$

Die **Probe** ist immer dieselbe: $0{,}5+0{,}25+0{,}25 = 1$ — die Anteile summieren sich zu 100 %. (Ich schreibe die Anteile hier $\lambda_1, \lambda_2, \lambda_3$; das ist nur ein Name, kein neuer Stoff.)

Und nun der Lohn: **Interpolieren** ist genau dieselbe gewichtete Summe wie in 1D, nur mit drei Gewichten. Kennen wir die Temperaturen an den drei Ecken — sagen wir $T_1 = 20$ °C, $T_2 = 300$ °C, $T_3 = 140$ °C —, dann ist die Temperatur bei P:

$$T(P) = \lambda_1 T_1 + \lambda_2 T_2 + \lambda_3 T_3 = 0{,}5 \cdot 20 + 0{,}25 \cdot 300 + 0{,}25 \cdot 140 = 120\ °\text{C}.$$

Genau diese Rechnung gießen wir in wenige Zeilen reines Python. Der Code sagt Wort für Wort, was oben auf Papier stand — Fläche als halbe Kreuzdifferenz, Anteile als Verhältnis, Interpolation als gewichtete Summe:

```
def dreiecksflaeche(p1, p2, p3):
    """Flaeche eines Dreiecks aus drei Ecken (halbe Kreuzdifferenz)."""
    kreuz = (p2[0] - p1[0]) * (p3[1] - p1[1]) \
        - (p3[0] - p1[0]) * (p2[1] - p1[1])
    if kreuz < 0.0:
        kreuz = -kreuz
    return kreuz / 2.0


def flaechenanteile(ecken, punkt):
    """Die drei Flaechenanteile eines Punktes im Dreieck (Summe 1)."""
    gesamt = dreiecksflaeche(ecken[0], ecken[1], ecken[2])
    anteil_1 = dreiecksflaeche(punkt, ecken[1], ecken[2]) / gesamt
    anteil_2 = dreiecksflaeche(punkt, ecken[2], ecken[0]) / gesamt
    anteil_3 = dreiecksflaeche(punkt, ecken[0], ecken[1]) / gesamt
    return [anteil_1, anteil_2, anteil_3]


def interpoliere(ecken, eckwerte, punkt):
    """Wert an einem Punkt aus den drei Eckwerten, gewichtet mit den Anteilen."""
    anteile = flaechenanteile(ecken, punkt)
    wert = 0.0
    for ecke in range(3):
        wert = wert + eckwerte[ecke] * anteile[ecke]
    return wert


ECKEN = [(0.0, 0.0), (4.0, 0.0), (0.0, 4.0)]
ECKWERTE = [20.0, 300.0, 140.0]
PUNKT = (1.0, 1.0)

anteile = flaechenanteile(ECKEN, PUNKT)
print("Flaechenanteile:")
for ecke in range(3):
    print("  Ecke", ecke + 1, ":", round(anteile[ecke], 3))
print("Summe der Anteile:", round(anteile[0] + anteile[1] + anteile[2], 3))
print("Interpolierte Temperatur bei P:", interpoliere(ECKEN, ECKWERTE, PUNKT),
      "Grad")
```

```
Flaechenanteile:
```

```
  Ecke 1 : 0.5
  Ecke 2 : 0.25
  Ecke 3 : 0.25
Summe der Anteile: 1.0
Interpolierte Temperatur bei P: 120.0 Grad
```

Interpretation der Ausgabe: Genau die Zahlen der Handrechnung — $(0{,}5;\ 0{,}25;\ 0{,}25)$, Summe 1, und 120 °C bei P. Der Code hat nichts Neues getan, er hat die Papierrechnung nur nachgesprochen. Das ist der einzige neue Baustein des ganzen Kapitels; alles Weitere kennst du schon.

10.3. Drei Punkte zum Anfassen: Ecke, Kante, Mitte

Damit die Flächenanteile Anschauung bekommen, schauen wir uns drei besondere Lagen von P an. Sie zeigen, warum die Anteile genau die richtigen Gewichte für die Interpolation sind.

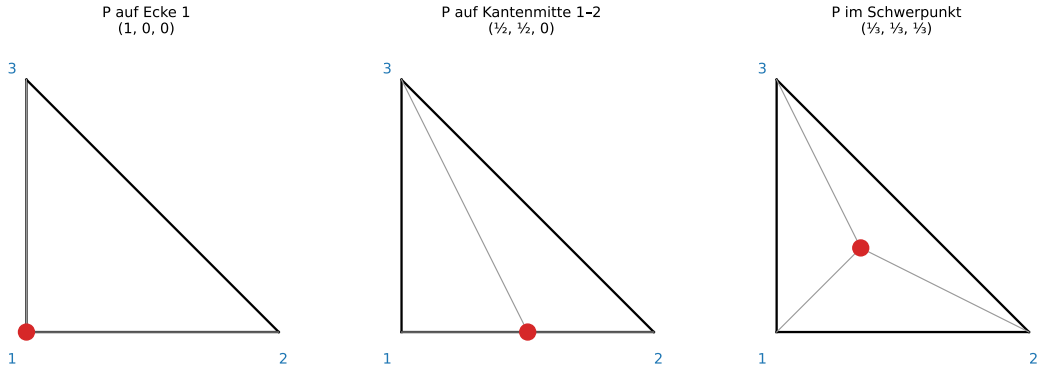

Abbildung 10.4.: Drei Lagen des Punktes P und ihre Anteil-Tripel. Auf einer Ecke gehört alles dieser Ecke (1, 0, 0). Auf der Mitte einer Kante teilen sich die beiden Kantenecken hälftig, die Gegenecke zählt gar nicht (½, ½, 0). Im Schwerpunkt sind alle drei gleich (⅓, ⅓, ⅓). Das mittlere Bild ist der Schlüssel zur Kantenkompatibilität.

Was man hier sieht: drei sprechende Fälle.

- **Auf Ecke 1** ist der Anteil $(1, 0, 0)$: Die Interpolation gibt genau T_1 zurück. Das ist die entscheidende Eigenschaft jeder Ansatzfunktion — an ihrem eigenen Knoten voll, an den anderen null. Es ist dieselbe Regel wie beim Hütchen (Kapitel 7).

- **Auf der Kantenmitte 1–2** ist der Anteil $(\frac{1}{2}, \frac{1}{2}, 0)$: Die Gegenecke 3 zählt **gar nicht**. Der Wert dort ist der schlichte Mittelwert von T_1 und T_2.
- **Im Schwerpunkt** ist alles gleich, $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ — der Mittelwert aller drei Ecken.

Der mittlere Fall verdient besondere Aufmerksamkeit, denn er löst ein Problem, das man leicht übersieht: Wie passen **benachbarte** Dreiecke zusammen?

Kantenkompatibilität: warum das Netz nahtlos ist

Zwei Dreiecke, die sich eine Kante teilen, haben auf dieser Kante **dieselben zwei Eckknoten** — und die Gegenecke des jeweils anderen Dreiecks zählt auf der Kante nicht (Anteil 0). Deshalb hängt die Temperatur **entlang der gemeinsamen Kante nur von den beiden geteilten Knotenwerten ab** — für beide Dreiecke gleich.
Die Folge: Zwischen zwei Dreiecken gibt es **keinen Sprung** und **keine Überlappung**. Der Streckenzug von Kapitel 7 wird in der Fläche zu einer zusammenhängenden, geknickten „Zeltlandschaft” ohne Risse. Genau das brauchen wir, damit die vielen kleinen Dreiecke ein sinnvolles Ganzes ergeben — und darauf baut Kapitel 11 auf, wenn Rechtecke die Dreiecke ablösen.

10.3.1. Der Punkt wandert: die Teilflächen atmen

Wenn P durch das Dreieck wandert, ändern sich die drei Teilflächen laufend — die eine wächst, während die andere schrumpft —, ihre Summe bleibt aber immer die volle Dreiecksfläche. Und der interpolierte Wert läuft mit. Die folgende Animation zeigt das: P zieht seine Bahn, die drei Teildreiecke atmen, und die Temperatur bei P wandert zwischen den Eckwerten hin und her.

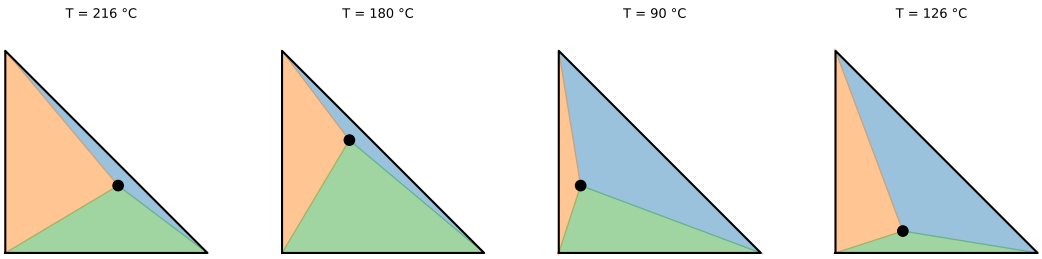

Abbildung 10.5.: Der wandernde Punkt als Bildreihe (in der HTML-Fassung eine Animation): vier Lagen von P auf seiner Schleife. Jedes Mal andere Teilflächen, andere Anteile, ein anderer interpolierter Wert — aber die Summe der Anteile ist stets 1.

Was man sieht: Die Teilflächen sind kommunizierende Röhren — was die eine gewinnt, verliert eine andere, die Summe bleibt konstant. Deshalb summieren sich die Anteile immer zu 1, und der interpolierte Wert bleibt stets zwischen dem kleinsten und größten Eckwert. Springt P nahe an eine Ecke, dominiert deren Anteil und die Temperatur nähert sich deren Eckwert.

10.4. Das Zelt: die Ansatzfunktion in der Fläche

Jetzt der Brückenschlag zu Kapitel 7. Dort war die Ansatzfunktion eines Knotens ein **Hütchen**: über dem eigenen Knoten auf 1, zu den Nachbarn geradlinig auf 0. In der Fläche wird daraus ein **Zelt**: Betrachte den Anteil λ_k eines Knotens **als Funktion des Ortes** — er ist 1 am eigenen Knoten und fällt über jedes anliegende Dreieck geradlinig auf 0 an den anderen Ecken. Setzt man die schrägen Dreiecksflächen aller Dreiecke zusammen, die an diesem Knoten hängen, entsteht ein Zeltdach.

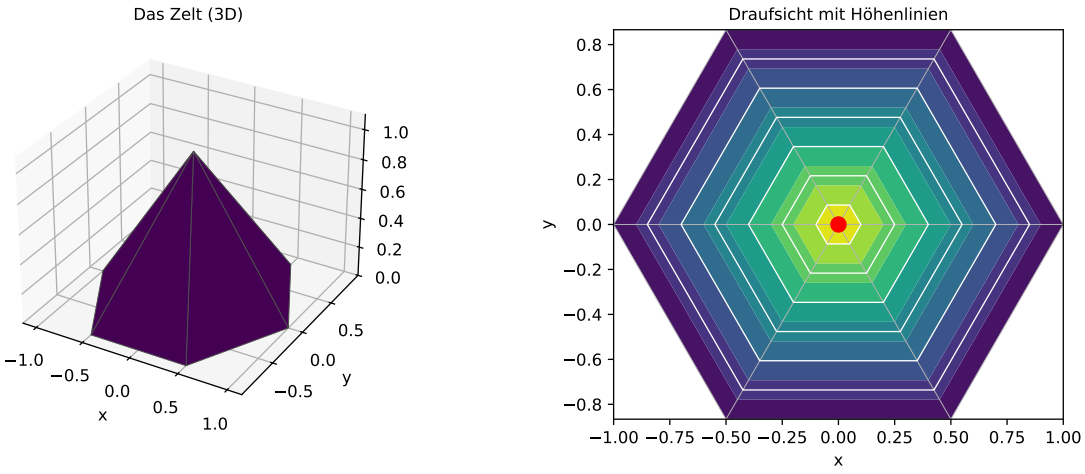

Abbildung 10.6.: Das Zelt eines Knotens über seinen sechs Dreiecken — die flächenhafte Fortsetzung des Hütchens aus Kapitel 7. Links die 3D-Ansicht: über dem eigenen Knoten steht das Zelt auf der Höhe 1, an allen Nachbarknoten auf 0, dazwischen jede Dreiecksfläche eben (schräg). Rechts die Draufsicht mit Höhenlinien: sechseckige Ringe, die zum Zentrum hin auf 1 steigen.

Was man hier sieht: links das Zelt in 3D — ein spitzes Dach, das über dem Mittelknoten auf 1 steht und zu allen sechs Nachbarn auf 0 abfällt; jede einzelne Dreiecksfläche ist eben (das garantieren die drei Ecken, die eine Ebene festlegen). Rechts dieselbe Funktion von oben, mit Höhenlinien: konzentrische sechseckige Ringe, die zum roten Mittelknoten hin auf 1 steigen. Genau wie in 1D überlappen sich nur die Zelte benachbarter Knoten, und an jedem Punkt der Fläche summieren sich alle Zelte zu 1.

Das Hütchen, eine Dimension höher

Schneidet man das Zelt mit einer senkrechten Ebene durch den Mittelknoten, erhält man **exakt** das Hütchen aus Kapitel 7: 1 in der Mitte, geradlinig auf 0 zu beiden Seiten. Das Zelt *ist* das Hütchen, in die zweite Dimension gezogen. Und weil jede Zeltfläche eben ist, hat sie über jedem Dreieck eine **konstante Steigung** — in x-Richtung eine, in y-Richtung eine. Das ist das flächenhafte Gegenstück zu den $\pm 1/h$ des Hütchens, und es macht die Elementmatrix gleich genauso einfach wie in 1D.

10.5. Das Netz als Bauplan

Bevor wir rechnen, brauchen wir eine Beschreibung des Netzes, die ein Programm lesen kann. Sie besteht aus **zwei Listen** — genau den Datenstrukturen aus Kapitel 2:

- die **Knotenliste**: für jeden Knoten seine Koordinaten (x, y),
- die **Dreiecksliste**: für jedes Dreieck die drei Knotennummern seiner Ecken.

Mehr ist ein Netz nicht. Man liest es wie einen Bauplan: Die Dreiecksliste sagt, *welche* Knoten ein Dreieck bilden, die Knotenliste sagt, *wo* diese Knoten liegen. Abbildung 10.7 zeigt beide Listen neben dem gezeichneten Netz.

```
Knotenliste (Nr: x, y):
  1: (0.00, 0.00)
  2: (0.50, 0.00)
  3: (1.00, 0.00)
  4: (0.00, 0.05)
  5: (0.50, 0.05)
  6: (1.00, 0.05)
  7: (0.00, 0.10)
  8: (0.50, 0.10)
  9: (1.00, 0.10)

Dreiecksliste (Ecken):
  T1: 1 – 2 – 5
  T2: 1 – 5 – 4
  T3: 2 – 3 – 6
  T4: 2 – 6 – 5
  T5: 4 – 5 – 8
  T6: 4 – 8 – 7
  T7: 5 – 6 – 9
  T8: 5 – 9 – 8
```

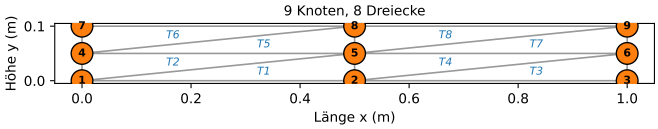

Abbildung 10.7.: Das Netz des Trägers als Bauplan. Links die beiden Listen — neun Knoten mit Koordinaten, acht Dreiecke mit je drei Knotennummern. Rechts das gezeichnete Netz mit nummerierten Knoten (orange) und Dreiecken (grau, eingekreist). Die Knoten sind von unten nach oben, die Dreiecke zellenweise durchnummeriert; jede Gitterzelle ist von unten links nach oben rechts geteilt.

Was man hier sieht: links der komplette Bauplan in zwei Listen, rechts das Netz. Die neun Knoten sitzen auf einem 3×3-Gitter ($x = 0/0{,}5/1$ m, $y = 0/0{,}05/0{,}1$ m), von unten nach oben nummeriert. Knoten 5 sitzt genau in der Mitte des Trägers. Jede der vier Gitterzellen ist diagonal in zwei Dreiecke geteilt — macht acht. Diese beiden Listen sind alles, was das Programm über die Geometrie wissen muss.

10.6. Die Elementmatrix: der 3×3-Steckbrief

Jetzt der Schritt, der in 1D die 2×2-Federmatrix lieferte — in 2D wird daraus eine **3×3-Matrix**, weil ein Dreieck drei Knoten hat. Und das Rezept ist dasselbe wie in Kapitel 7: Jeder Eintrag ist **Steigung mal Steigung**, mal Material, mal Fläche. Nur gibt es jetzt zwei Steigungen pro Zeltfläche (eine in x, eine in y), und wir zählen ihre Produkte zusammen:

$$K_{ij} = \lambda \cdot d \cdot A \cdot \Big(\underbrace{\tfrac{\partial N_i}{\partial x}\tfrac{\partial N_j}{\partial x}}_{\text{Steigungen in } x} + \underbrace{\tfrac{\partial N_i}{\partial y}\tfrac{\partial N_j}{\partial y}}_{\text{Steigungen in } y} \Big).$$

In Worten: **Leitfähigkeit λ mal Tiefe d mal Dreiecksfläche A mal (Steigung mal Steigung, in beiden Richtungen zusammengezählt).** Weil die Steigungen über das ganze Dreieck konstant sind, ist der Klammerausdruck konstant — und das Elementintegral wieder nur „Wert mal Fläche", genau wie in 1D. Die Steigungen der Zeltfläche von Knoten i bekommt man aus den Ecken:

$$\frac{\partial N_i}{\partial x} = \frac{y_j - y_k}{2A}, \qquad \frac{\partial N_i}{\partial y} = \frac{x_k - x_j}{2A},$$

wobei j, k die beiden *anderen* Ecken sind und $2A$ die doppelte Fläche. Das sieht abstrakt aus — machen wir es an einem echten Dreieck des Trägers konkret.

10.6.1. Ein Dreieck von Hand

Wir nehmen das rechtwinklige Dreieck **T1** aus dem Netz: die Knoten 1, 2, 5 mit den Ecken $(0,0)$, $(0{,}5,0)$, $(0{,}5,0{,}05)$. Rechtwinklig heißt: Die Zahlen werden freundlich. Zuerst die doppelte Fläche und die Fläche selbst:

$$2A = 0{,}025\ \text{m}^2, \qquad A = 0{,}0125\ \text{m}^2$$

(ein rechtwinkliges Dreieck mit Katheten $0{,}5$ und $0{,}05$: Fläche $\tfrac{1}{2}\cdot 0{,}5 \cdot 0{,}05 = 0{,}0125$). Nun die Steigungen der drei Zeltflächen — je eine in x, eine in y, in $1/\text{m}$:

Knoten	$\partial N/\partial x$	$\partial N/\partial y$
1 (0, 0)	-2	0

Knoten	$\partial N/\partial x$	$\partial N/\partial y$
2 (0,5, 0)	$+2$	-20
5 (0,5, 0,05)	0	$+20$

Die Steigungen summieren sich spaltenweise zu null (das müssen sie, weil die drei Zelte zusammen überall die konstante Höhe 1 ergeben). Die y-Steigungen sind mit ± 20 groß, weil das Dreieck nur $0{,}05$ m hoch ist — auf kurzer Höhe muss das Zelt steil abfallen. Der gemeinsame Vorfaktor ist

$$\lambda \cdot d \cdot A = 50 \cdot 0{,}1 \cdot 0{,}0125 = 0{,}0625 \ \tfrac{\mathrm{W}}{\mathrm{K}}.$$

Jetzt jeder Eintrag als „Steigung mal Steigung, mal $0{,}0625$“. Zwei Beispiele, den Rest macht das Programm:

$$K_{11} = 0{,}0625 \cdot \big((-2)(-2) + (0)(0)\big) = 0{,}0625 \cdot 4 = 0{,}25,$$
$$K_{25} = 0{,}0625 \cdot \big((+2)(0) + (-20)(+20)\big) = 0{,}0625 \cdot (-400) = -25.$$

Alle neun zusammengeschrieben ergeben den Steckbrief von T1 (in W/K):

$$\mathbf{K}_{\mathrm{T1}} = \begin{pmatrix} 0{,}25 & -0{,}25 & 0 \\ -0{,}25 & 25{,}25 & -25 \\ 0 & -25 & 25 \end{pmatrix} \qquad \text{Zeilen 1, 2, 5}$$

Und hier die vertraute **Probe**: Jede Zeile summiert sich zu null ($0{,}25 - 0{,}25 + 0 = 0$, $-0{,}25 + 25{,}25 - 25 = 0$, $0 - 25 + 25 = 0$). Das ist dasselbe Zeichen wie bei der Feder in Kapitel 3 — verschiebt man alle drei Knoten um dasselbe Stück (oder erhitzt sie gleich), passiert nichts, also müssen die Zeilen sich wegheben. Lassen wir das Programm dieselbe Matrix ausrechnen, damit die Handrechnung belegt ist:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from dreiecke import baue_dreieck_matrix

ecken_t1 = [(0.0, 0.0), (0.5, 0.0), (0.5, 0.05)]
steckbrief = baue_dreieck_matrix(ecken_t1, 50.0, 0.1)

print("Steckbrief von T1 (Knoten 1, 2, 5), in W/K:")
for zeile in steckbrief:
```

```
    zeilensumme = zeile[0] + zeile[1] + zeile[2]
    print("  [ %7.2f %7.2f %7.2f ]   Zeilensumme = %.2f"
          % (zeile[0], zeile[1], zeile[2], zeilensumme))
```

```
Steckbrief von T1 (Knoten 1, 2, 5), in W/K:
  [    0.25   -0.25    0.00 ]   Zeilensumme = 0.00
  [   -0.25   25.25  -25.00 ]   Zeilensumme = 0.00
  [    0.00  -25.00   25.00 ]   Zeilensumme = 0.00
```

Interpretation der Ausgabe: Zahl für Zahl die Matrix der Handrechnung, und jede Zeilensumme ist null. Die Handrechnung ist damit belegt. Ein neuer Elementtyp, dieselbe Prüfregel wie seit Kapitel 3.

10.7. Assemblieren, einspannen, lösen

Ab hier ist es reine Wiederholung — das **vierte** Mal in diesem Buch, dass wir Element-Steckbriefe zu einem System zusammenschieben. Jedes der acht Dreiecke legt seinen 3×3-Steckbrief an die Stelle seiner drei Knoten in die große 9×9-Matrix, und wo Dreiecke sich einen Knoten teilen, addieren sich ihre Beiträge auf dessen Diagonalfeld. Die folgende Animation pflastert den Träger Dreieck für Dreieck und lässt das System mitwachsen.

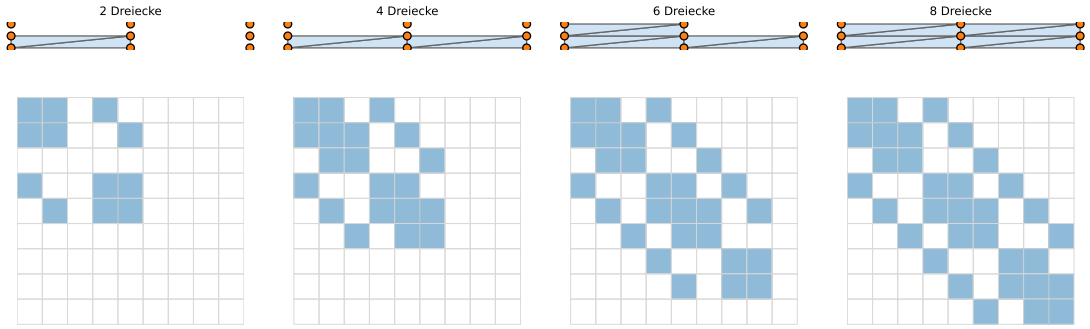

Abbildung 10.8.: Die Assemblierung als Bildreihe (in der HTML-Fassung eine Animation): der Träger nach 2, 4, 6 und 8 Dreiecken, darunter jeweils die belegten Felder der 9×9-Matrix. Das Band wird mit jedem Dreieck breiter besetzt, bleibt aber immer dicht um die Diagonale.

Was man sieht: Der Träger füllt sich mit Dreiecken, und im gleichen Takt füllt sich die Matrix. Anders als in 1D, wo jeder Knoten nur zwei Nachbarn

hatte (Tridiagonale), hängt ein Knoten hier mit mehreren zusammen — auch schräg über die Netzkanten. Das Band ist deshalb **breiter** als in 1D, bleibt aber ein Band: weit weg von der Diagonale ist alles leer.

10.7.1. Das System, ausgeschrieben

Schreiben wir die assemblierte Matrix als Textblock — die **Matrix-Drucker**-Konvention aus Kapitel 3, Nullen als Punkte. Daneben zum Vergleich das tridiagonale 1D-System aus 2/Kapitel 7.

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from dreiecke import assembliere_2d
from matrix_drucker import drucke_matrix

knoten = [(0.0, 0.0), (0.5, 0.0), (1.0, 0.0),
          (0.0, 0.05), (0.5, 0.05), (1.0, 0.05),
          (0.0, 0.10), (0.5, 0.10), (1.0, 0.10)]
dreiecke = [(0, 1, 4), (0, 4, 3), (1, 2, 5), (1, 5, 4),
            (3, 4, 7), (3, 7, 6), (4, 5, 8), (4, 8, 7)]

matrix = assembliere_2d(knoten, dreiecke, 50.0, 0.1)
print("Systemmatrix des 8-Dreiecke-Trägers (9x9, W/K), Nullen als Punkte:")
print()
drucke_matrix(matrix)
```

```
Systemmatrix des 8-Dreiecke-Trägers (9x9, W/K), Nullen als Punkte:

     25      -0       .     -25       .       .       .       .       .
     -0      50      -0       .     -50       .       .       .       .
      .      -0      25       .       .     -25       .       .       .
    -25       .       .      50      -0       .     -25       .       .
      .     -50       .      -0     101      -0       .     -50       .
      .       .     -25       .      -0      50       .       .     -25
      .       .       .     -25       .       .      25      -0       .
      .       .       .       .     -50       .      -0      50      -0
      .       .       .       .       .     -25       .      -0      25
```

Interpretation der Ausgabe: ein breiteres Band als die 1D-Tridiagonale. Der Matrix-Drucker rundet auf ganze W/K. Auf der Diagonale stehen die großen Zahlen (gerundet 25, 50 und 101; in Wahrheit 25,25 und 50,5), daneben

die starke Kopplung *quer* über die kurze Höhe (-25 und -50). Die Kopplung *längs* des Trägers ist dagegen so schwach — nur $-0{,}25$ bzw. $-0{,}5\ \mathrm{W/K}$ —, dass der ganzzahlige Drucker sie als „-0" zeigt: winzig, aber eben nicht null (einen Punkt druckt er nur, wo gar keine Kopplung ist). Die lang gestreckten, flachen Dreiecke koppeln über die kurze Höhe viel stärker als über die lange Länge — die Geometrie steht der Matrix ins Gesicht geschrieben. Abbildung 10.9 stellt beide Bandstrukturen nebeneinander.

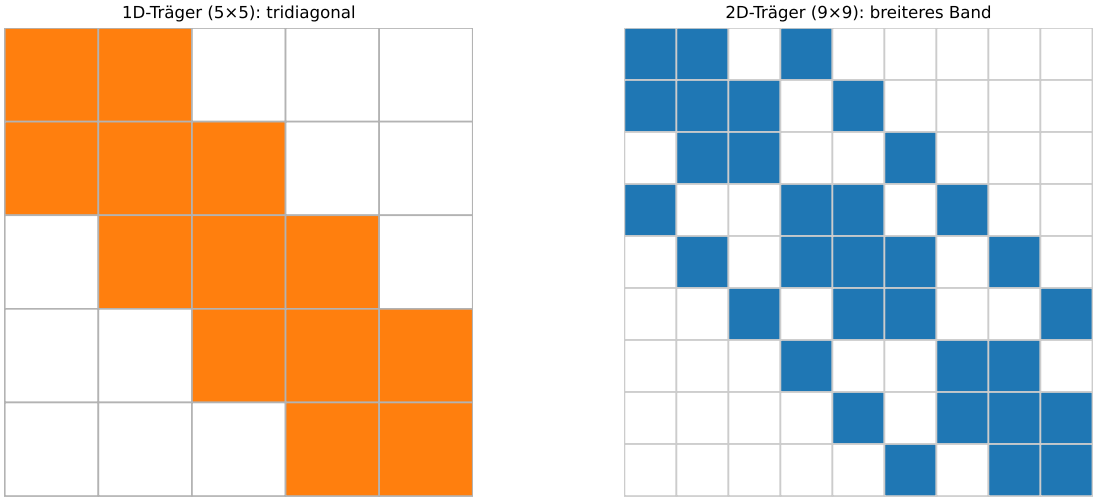

Abbildung 10.9.: Zwei Bandstrukturen. Links das 1D-System aus Kapitel 7 (5×5, tridiagonal): nur die Diagonale und je ein Nachbar. Rechts das 2D-System dieses Kapitels (9×9): dasselbe Prinzip — belegt nur nahe der Diagonale —, aber ein breiteres Band, weil jeder Knoten in der Fläche mehr Nachbarn hat. Beide sind dünn besetzt, beide löst derselbe Löser aus Kapitel 4.

Was man hier sieht: links das schmale 1D-Band, rechts das breitere 2D-Band — aber beide sind **dünn besetzt**, beide haben ihre Zahlen dicht um die Diagonale. Der Unterschied ist nur die Bandbreite, nicht das Prinzip. Und weil beide dünn besetzt sind, löst sie derselbe Löser aus 2.

10.7.2. Randwerte und Lösung

Die linke Kante (Knoten 1, 4, 7) halten wir auf 20 °C, die rechte (Knoten 3, 6, 9) auf 300 °C — das **Streich-Rezept** aus Kapitel 8, jetzt auf sechs statt zwei Randknoten angewandt. Übrig bleiben die drei freien Knoten der Mittelspalte (2, 5, 8). Gelöst wird mit **Gauß-Seidel** aus 2.

Vorhersage-Punkt

Bevor wir lösen: Knoten 5 sitzt genau in der Trägermitte. Links wird der Träger auf 20 °C gehalten, rechts auf 300 °C. Welche Temperatur erwartest du an Knoten 5? Und was ist mit den anderen beiden Mittelknoten (2 unten, 8 oben) — gleiche Temperatur oder anders? Leg dich fest.

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from dreiecke import assembliere_2d, reduziere_system, setze_ein
from loeser import gauss_seidel

knoten = [(0.0, 0.0), (0.5, 0.0), (1.0, 0.0),
          (0.0, 0.05), (0.5, 0.05), (1.0, 0.05),
          (0.0, 0.10), (0.5, 0.10), (1.0, 0.10)]
dreiecke = [(0, 1, 4), (0, 4, 3), (1, 2, 5), (1, 5, 4),
            (3, 4, 7), (3, 7, 6), (4, 5, 8), (4, 8, 7)]

matrix = assembliere_2d(knoten, dreiecke, 50.0, 0.1)

# Randwerte: linke Kante 20, rechte 300, Mittelspalte frei (None).
randwerte = [20.0, None, 300.0, 20.0, None, 300.0, 20.0, None, 300.0]
kleine_matrix, kleine_seite, frei = reduziere_system(
    matrix, [0.0] * 9, randwerte)

startwerte = [0.0] * len(frei)
loesung_frei = gauss_seidel(kleine_matrix, kleine_seite, startwerte,
                            0.00000001, 2000)
temperaturen = setze_ein(loesung_frei, frei, randwerte)

frei_namen = []
for k in frei:
    frei_namen.append(k + 1)
print("Freie Knoten (Mittelspalte):", frei_namen)
print()
print("Knoten |   Ort (x, y)    | Temperatur")
print("-------+-----------------+-----------")
for i in range(9):
    x, y = knoten[i]
    print("   %d   | (%.2f, %.3f) m | %6.1f Grad" % (i + 1, x, y, temperaturen[i]))
```

```
Freie Knoten (Mittelspalte): [2, 5, 8]

Knoten |   Ort (x, y)    | Temperatur
-------+-----------------+-----------
   1   | (0.00, 0.000) m |   20.0 Grad
   2   | (0.50, 0.000) m |  160.0 Grad
   3   | (1.00, 0.000) m |  300.0 Grad
   4   | (0.00, 0.050) m |   20.0 Grad
   5   | (0.50, 0.050) m |  160.0 Grad
   6   | (1.00, 0.050) m |  300.0 Grad
   7   | (0.00, 0.100) m |   20.0 Grad
   8   | (0.50, 0.100) m |  160.0 Grad
   9   | (1.00, 0.100) m |  300.0 Grad
```

Interpretation der Ausgabe: Alle drei Mittelknoten stehen auf **160 °C** — genau in der Mitte zwischen 20 und 300, und oben wie unten gleich. Lag deine Vorhersage richtig? Das ist die Pointe des Kapitels:

> ⚠ Vermutung aufgelöst
>
> **Die Vermutung war:** „2D braucht ganz neue Mathematik." **Und heraus kommt:** dieselbe Gerade wie in 1D. Bei diesem Rand (links überall kühl am Auflager, rechts überall heiß beim Brandherd) hängt die Temperatur gar nicht von der Höhe y ab — sie wächst nur mit der Länge x, exakt wie beim 1D-Träger aus Kapitel 7. Die Mittelspalte liegt bei $x = 0{,}5$, also bei 160 °C. **Hier ist 2D noch 1D in Verkleidung.** Genau das haben wir im Kasten „Warum durften wir 1D rechnen?" vorausgesagt: Solange die Höhe gleichmäßig belastet wird, verrät sie nichts Neues. Das ändert sich, sobald die Wärme nicht mehr von der Seite, sondern von **unten** kommt: Treffen die Brandgase die Unterkante und hält die Betondecke die Oberseite kühl, dann kippt das Gefälle in die **Höhe** — ein Höhenunterschied, der kein Streifenbild mehr ist und der den Träger später (Kapitel 14, 3) krümmt. Genau das rechnest du in Übung 10.2 nach.

Das Temperatur-Farbbild macht die Streifen sichtbar: waagerechte Bänder, kein Unterschied zwischen oben und unten.

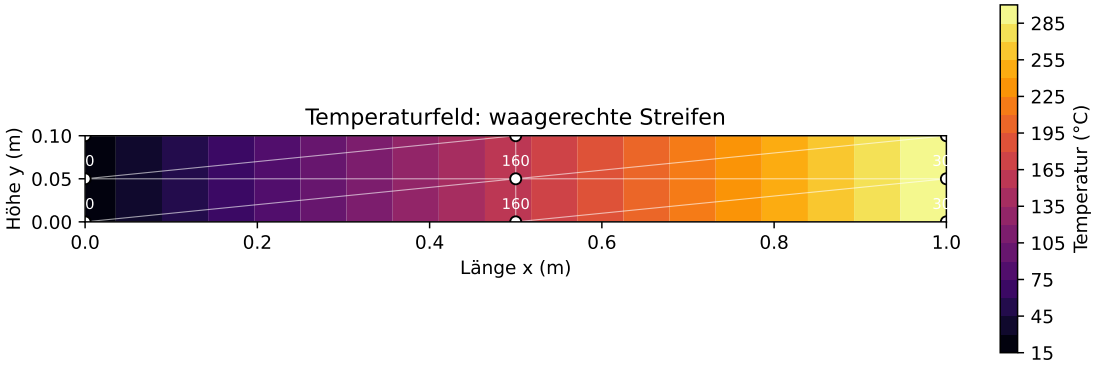

Abbildung 10.10.: Das Temperaturfeld des Trägers als Farbbild. Die Farbe wächst gleichmäßig von 20 °C links (dunkel) auf 300 °C rechts (hell); die Höhenlinien sind senkrecht — bei gleichem Rand hängt die Temperatur nur von der Länge ab, nicht von der Höhe. Das ist das bunte Bild zu den 160 °C der Mittelspalte.

Was man hier sieht: senkrechte Farbübergänge, waagerechte Streifen. Die Temperatur klettert gleichmäßig von links (20 °C) nach rechts (300 °C), und auf jeder senkrechten Linie ist sie konstant. Das bestätigt die 160 °C und die Pointe: Bei symmetrischem Rand ist das Flächenfeld nur die in die Breite gezogene 1D-Gerade. In Übung 10.2 kippen die Streifen: Trifft das Brandgas die Unterkante, läuft das Gefälle über die Höhe.

10.8. Die interaktive Einheit: das Dreiecks-Labor

Jetzt bist du dran. Das Dreiecks-Labor hat zwei Teile. **Teil 1** lässt dich einen Punkt in ein Dreieck setzen und liest die drei Flächenanteile samt interpoliertem Wert ab. **Teil 2** ist der ganze 8-Dreiecke-Träger: Randwerte setzen, System drucken, mit Gauß-Seidel lösen, Farbbild zeichnen.

10.8.1. Teil 1 — Flächenanteile eines Punktes

Setze `PUNKT` an eine Stelle im Dreieck und sieh, wie sich die drei Anteile und der interpolierte Wert ändern. Probiere eine Ecke ($(0, 0)$), eine Kantenmitte ($(2, 0)$) und den Schwerpunkt (etwa $(1{,}33,\ 1{,}33)$).

```
#| autorun: true
# Setze den Punkt und die drei Eckwerte -- probiere Ecke, Kante, Mitte.
PUNKT = (1.0, 1.0)
ECKEN = [(0.0, 0.0), (4.0, 0.0), (0.0, 4.0)]
ECKWERTE = [20.0, 300.0, 140.0]   # Temperaturen an den drei Ecken


def dreiecksflaeche(p1, p2, p3):
    """Flaeche eines Dreiecks (halbe Kreuzdifferenz)."""
    kreuz = (p2[0] - p1[0]) * (p3[1] - p1[1]) \
        - (p3[0] - p1[0]) * (p2[1] - p1[1])
    if kreuz < 0.0:
        kreuz = -kreuz
    return kreuz / 2.0


def flaechenanteile(ecken, punkt):
    """Die drei Flaechenanteile eines Punktes (Summe 1)."""
    gesamt = dreiecksflaeche(ecken[0], ecken[1], ecken[2])
    anteil_1 = dreiecksflaeche(punkt, ecken[1], ecken[2]) / gesamt
    anteil_2 = dreiecksflaeche(punkt, ecken[2], ecken[0]) / gesamt
    anteil_3 = dreiecksflaeche(punkt, ecken[0], ecken[1]) / gesamt
    return [anteil_1, anteil_2, anteil_3]


anteile = flaechenanteile(ECKEN, PUNKT)
wert = 0.0
for ecke in range(3):
    wert = wert + ECKWERTE[ecke] * anteile[ecke]

print("Punkt:", PUNKT)
print("Flaechenanteile:")
for ecke in range(3):
    print("  Ecke %d: %.3f  (Eckwert %.0f Grad)"
          % (ecke + 1, anteile[ecke], ECKWERTE[ecke]))
print("Summe der Anteile:", round(anteile[0] + anteile[1] + anteile[2], 3))
print("Interpolierte Temperatur bei P: %.1f Grad" % wert)

# --- Abbildung ---
import matplotlib.pyplot as plt
V1, V2, V3 = ECKEN
fig, ax = plt.subplots(figsize=(5.5, 5.0))
ax.fill([PUNKT[0], V2[0], V3[0]], [PUNKT[1], V2[1], V3[1]], color="tab:blue", alpha=0.45)
```

```
ax.fill([PUNKT[0], V3[0], V1[0]], [PUNKT[1], V3[1], V1[1]], color="tab:orange", alpha=0.4
ax.fill([PUNKT[0], V1[0], V2[0]], [PUNKT[1], V1[1], V2[1]], color="tab:green", alpha=0.45
ax.plot([V1[0], V2[0], V3[0], V1[0]], [V1[1], V2[1], V3[1], V1[1]], "k-", lw=1.8)
ax.plot(PUNKT[0], PUNKT[1], "o", color="black", ms=10)
ax.set_title("T bei P = %.0f Grad" % wert)
ax.set_aspect("equal")
ax.axis("off")
plt.tight_layout()
plt.show()
```

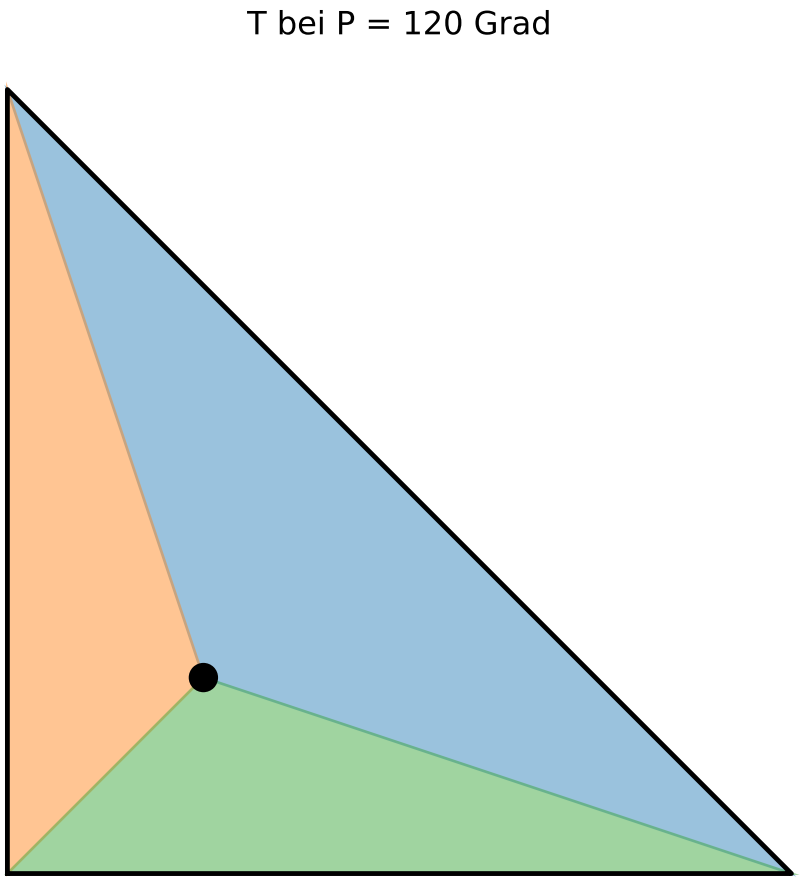

Abbildung 10.11.: Vorgerenderte Fassung von Teil 1 des Dreiecks-Labors: Punkt P = (1, 1) mit den Anteilen (½, ¼, ¼) und T = 120 °C. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du den Punkt verschiebst.

10.8.2. Teil 2 — der 8-Dreiecke-Träger

Setze die beiden Randtemperaturen `LINKS` und `RECHTS`, und das Labor baut das System, druckt es, löst mit Gauß-Seidel und zeichnet das Farbbild. Der Code ist Zeile für Zeile das, was oben im Kapitel stand.

```
#| autorun: true
# Setze die Randtemperaturen (linke und rechte Kante) und sieh das Feld.
LINKS = 20.0
RECHTS = 300.0

KNOTEN = [(0.0, 0.0), (0.5, 0.0), (1.0, 0.0),
          (0.0, 0.05), (0.5, 0.05), (1.0, 0.05),
          (0.0, 0.10), (0.5, 0.10), (1.0, 0.10)]
DREIECKE = [(0, 1, 4), (0, 4, 3), (1, 2, 5), (1, 5, 4),
            (3, 4, 7), (3, 7, 6), (4, 5, 8), (4, 8, 7)]


def baue_dreieck_matrix(ecken):
    """Der 3x3-Steckbrief eines Dreiecks (Steigung mal Steigung mal Flaeche)."""
    (xa, ya) = ecken[0]
    (xb, yb) = ecken[1]
    (xc, yc) = ecken[2]
    zwei_flaeche = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    flaeche = zwei_flaeche
    if flaeche < 0.0:
        flaeche = -flaeche
    flaeche = flaeche / 2.0
    steigung_x = [(yb - yc) / zwei_flaeche, (yc - ya) / zwei_flaeche,
                  (ya - yb) / zwei_flaeche]
    steigung_y = [(xc - xb) / zwei_flaeche, (xa - xc) / zwei_flaeche,
                  (xb - xa) / zwei_flaeche]
    matrix = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
    for i in range(3):
        for j in range(3):
            skalar = steigung_x[i] * steigung_x[j] + steigung_y[i] * steigung_y[j]
            matrix[i][j] = 50.0 * 0.1 * flaeche * skalar
    return matrix


def assembliere(knoten, dreiecke):
    """Schiebt alle 3x3-Steckbriefe in die grosse Systemmatrix."""
    n = len(knoten)
```

```
    matrix = []
    for zeile in range(n):
        neue_zeile = []
        for spalte in range(n):
            neue_zeile.append(0.0)
        matrix.append(neue_zeile)
    for dreieck in dreiecke:
        ecken = [knoten[dreieck[0]], knoten[dreieck[1]], knoten[dreieck[2]]]
        steckbrief = baue_dreieck_matrix(ecken)
        for a in range(3):
            for b in range(3):
                i = dreieck[a]
                j = dreieck[b]
                matrix[i][j] = matrix[i][j] + steckbrief[a][b]
    return matrix


def gauss_seidel(matrix, rechte_seite, startwerte, toleranz, max_runden):
    """Loest das System naeherungsweise (Entspannungsverfahren aus Kapitel 4)."""
    n = len(matrix)
    wert = list(startwerte)
    for runde in range(max_runden):
        groesste = 0.0
        for i in range(n):
            summe = rechte_seite[i]
            for j in range(n):
                if j != i:
                    summe = summe - matrix[i][j] * wert[j]
            neu = summe / matrix[i][i]
            aenderung = neu - wert[i]
            if aenderung < 0.0:
                aenderung = -aenderung
            if aenderung > groesste:
                groesste = aenderung
            wert[i] = neu
        if groesste < toleranz:
            break
    return wert


# Randwerte: linke Kante (x = 0) auf LINKS, rechte Kante (x = 1) auf RECHTS.
randwerte = []
for (x, y) in KNOTEN:
```

```
    if x == 0.0:
        randwerte.append(LINKS)
    elif x == 1.0:
        randwerte.append(RECHTS)
    else:
        randwerte.append(None)

matrix = assembliere(KNOTEN, DREIECKE)

# Streich-Rezept: nur die freien Knoten behalten, feste auf die rechte Seite.
frei = []
for knoten in range(len(KNOTEN)):
    if randwerte[knoten] is None:
        frei.append(knoten)

kleine_matrix = []
kleine_seite = []
for i in frei:
    neue_zeile = []
    for j in frei:
        neue_zeile.append(matrix[i][j])
    kleine_matrix.append(neue_zeile)
    beitrag = 0.0
    for j in range(len(KNOTEN)):
        if randwerte[j] is not None:
            beitrag = beitrag + matrix[i][j] * randwerte[j]
    kleine_seite.append(-beitrag)

frei_namen = []
for k in frei:
    frei_namen.append(k + 1)
print("Reduziertes System der freien Knoten", frei_namen, ":")
for zeile, seite in zip(kleine_matrix, kleine_seite):
    print("  [ %7.1f %7.1f %7.1f ] = %8.1f"
          % (zeile[0], zeile[1], zeile[2], seite))

startwerte = []
for k in range(len(frei)):
    startwerte.append(0.0)
loesung_frei = gauss_seidel(kleine_matrix, kleine_seite, startwerte,
                            0.00000001, 2000)

temperaturen = list(randwerte)
```

```
for k in range(len(frei)):
    temperaturen[frei[k]] = loesung_frei[k]

print()
print("Temperaturen an den 9 Knoten:")
for i in range(len(KNOTEN)):
    print("  Knoten %d: %.1f Grad" % (i + 1, temperaturen[i]))

# --- Abbildung ---
import matplotlib.pyplot as plt
import matplotlib.tri as mtri
xs = [p[0] for p in KNOTEN]
ys = [p[1] for p in KNOTEN]
triang = mtri.Triangulation(xs, ys, DREIECKE)
fig, ax = plt.subplots(figsize=(9.0, 3.0))
bild = ax.tricontourf(triang, temperaturen, levels=20, cmap="inferno")
ax.triplot(triang, color="white", lw=0.6, alpha=0.6)
fig.colorbar(bild, ax=ax, label="Temperatur (Grad)")
ax.set_xlabel("Laenge x (m)")
ax.set_ylabel("Hoehe y (m)")
ax.set_title("Temperaturfeld des Trägers")
ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

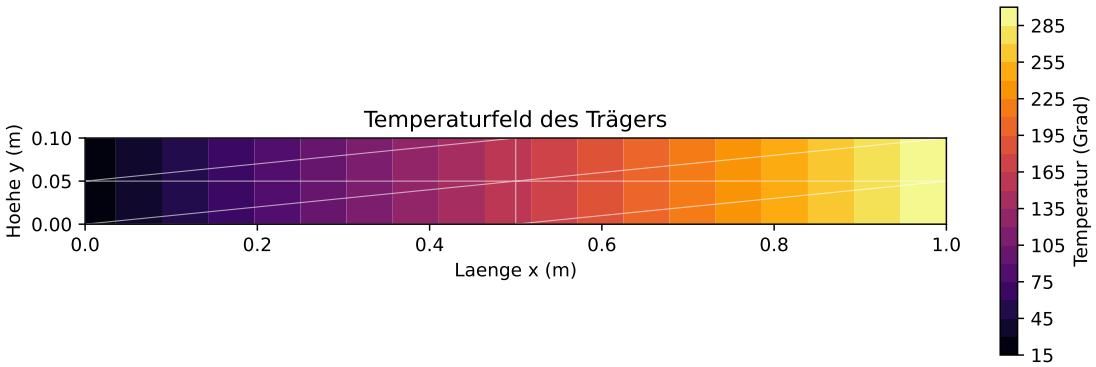

Abbildung 10.12.: Vorgerenderte Fassung von Teil 2: das Temperaturfeld bei 20 °C links und 300 °C rechts — waagerechte Streifen, Mittelspalte 160 °C. Im Browser ersetzt dein eigenes Ergebnis dieses Bild, sobald du die Randwerte änderst.

Was man sieht: Bei gleichem Rand links und rechts bleibt das Feld streifig

(Mittelspalte 160 °C). Setze in Teil 2 einmal `LINKS = 300` und `RECHTS = 20` — die Streifen drehen sich um. Erst Übung 10.2 (Brandgas an der Unterkante) lässt das Gefälle über die Höhe kippen und bringt so die zweite Dimension ins Spiel.

10.9. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript liegt in `programme/kap10/kap10_dreiecke_fem.py`. Es baut das 8-Dreiecke-Netz, rechnet den Steckbrief von T1 vor, druckt die assemblierte Systemmatrix, baut die Randwerte ein und löst mit Gauß-Seidel — dieselben Schritte wie im Kapitel, an einem Stück. Die 2D-Bausteine selbst (`baue_dreieck_matrix`, `assembliere_2d`, `reduziere_system`, …) liegen im gemeinsamen Modul `programme/gemeinsam/dreiecke.py`, das auch Kapitel 12 und die Verformung in Kapitel 14 wiederverwenden. Führe das Skript mit `python kap10_dreiecke_fem.py` aus.

10.10. Die Namensschilder

Wie in Kapitel 7 haben wir die Sache zuerst gebaut und die Fachwörter weggelassen. Jetzt die Schilder.

Die Fachwörter, jetzt nachgereicht

- **Baryzentrische Koordinaten**: das Fachwort für unsere **Flächenanteile** $\lambda_1, \lambda_2, \lambda_3$. „Baryzentrum" ist griechisch-lateinisch für **Schwerpunkt** (*barys* = schwer). Die Idee stammt von **August Ferdinand Möbius** (1827): Man beschreibt einen Punkt durch die Gewichte, die man auf die drei Ecken legen müsste, damit ihr Schwerpunkt genau dort liegt. Diese Gewichte *sind* die Flächenanteile.
- **P1-Dreieck** (lineares Dreieckselement): das Dreieck mit einem Knoten je Ecke und **linearer** Interpolation dazwischen — das „P1" steht für Polynome vom Grad 1 (eben, keine Krümmung). Es ist das flächenhafte Gegenstück zum linearen Stab-Element aus Kapitel 7.
- **Zeltfunktion** (Ansatz-/Formfunktion in 2D): die Funktion $N_k = \lambda_k$, über dem eigenen Knoten 1, an allen anderen 0 — das **Zelt**, das wir gezeichnet haben. Es ist das Hütchen aus Kapitel 7, eine Dimension höher.
- **Triangulierung**: die Zerlegung einer Fläche in Dreiecke — der Bauplan aus Knoten- und Dreiecksliste. Das zugehörige **Netz** ist

das flächenhafte Gegenstück zur Knotenkette in 1D.

Alle diese Wörter stehen im Glossar; hier haben wir zuerst die Sache gebaut und dann das Schild geklebt.

Woher kommen die Dreiecke? Ein Namedrop.

Wir haben das Netz von Hand hingeschrieben — neun Knoten, acht Dreiecke. Für echte, krumme Formen übernimmt das ein **Netzgenerator**. Das bekannteste Verfahren ist die **Delaunay-Triangulierung** (nach Boris Delaunay, 1934): Sie verbindet die Punkte so, dass die Dreiecke möglichst „gutmütig" werden — keine extrem spitzen Nadeln, sondern nach Möglichkeit gedrungene Dreiecke. Wie das genau geht, ist ein Kapitel für sich und nicht unser Thema; wir schreiben unsere kleinen Netze weiter von Hand. Merke nur: Ein Programm kann die Dreiecksliste aus einer Punktwolke selbst erzeugen, und Delaunay ist der Klassiker dafür.

Merkkasten

- Der einzige neue Baustein in der Fläche ist die **Interpolation mit Flächenanteilen**: Punkt P mit den Ecken verbinden, die drei Teildreiecke messen (halbe Kreuzdifferenz), durch die Gesamtfläche teilen. Die Anteile summieren sich zu 1.
- Interpolieren heißt: jeder Eckwert mal seinem Anteil, aufsummiert — dieselbe gewichtete Summe wie in 1D, nur mit drei Gewichten.
- Die Ansatzfunktion eines Knotens ist ein **Zelt** über seinen Dreiecken — das Hütchen aus Kapitel 7, eine Dimension höher. Jede Zeltfläche ist eben, also hat sie über jedem Dreieck **konstante Steigungen** (eine in x, eine in y).
- Ein **Netz** ist eine Knotenliste plus eine Dreiecksliste — ein Bauplan.
- Die **Elementmatrix** ist jetzt 3×3: $K_{ij} = \lambda\, d\, A\, (\partial_x N_i\, \partial_x N_j + \partial_y N_i\, \partial_y N_j)$. Zeilensummen null, wie bei der Feder.
- **Assemblieren, Randwerte, lösen** liefert für den symmetrischen Träger die Mittelspalte auf 160 °C: 2D ist hier noch **1D in Verkleidung**.

Roter Faden

Zurück: Der ganze Rechenweg ist die Wiederkehr von Kapitel 7 — nur der Interpolationsbaustein wechselte von Streckenanteilen (Hütchen) zu **Flächenanteilen** (Zelt). Die **Elementmatrix** aus Steigung mal Steigung, die **Assemblierung** durch Aufaddieren, das **Streich-Rezept** für Randwerte (Kapitel 8) und der **Gauß-Seidel-Löser** (2) sind alle unverändert übernommen. Selbst die Probe „Zeilensumme null" ist dieselbe wie bei der Feder (Kapitel 3).

Vor: In Kapitel 11 lösen **Rechtecke** die Dreiecke ab — mit bilinearer statt linearer Interpolation, und wir vergleichen beide. Kapitel 12 fragt, wie *genau* diese groben Netze eigentlich sind (Netzkonvergenz), denn hier war die Lösung nur darum exakt, weil sie zufällig eine Gerade war. Und dasselbe Netz, das hier die Temperatur trug, trägt in Kapitel 14 die **Verschiebungen** des sich verbiegenden Trägers — dann sitzen an jedem Knoten zwei Zahlen statt einer.

Übungen

Ü 10.1 (Verstehen). Gegeben das Dreieck mit den Ecken $V_1 = (0,0)$, $V_2 = (6,0)$, $V_3 = (0,3)$ und der Punkt $P = (2,1)$. Berechne von Hand die drei Flächenanteile (halbe Kreuzdifferenz) und prüfe die Summe. Kontrolliere danach im Dreiecks-Labor (Teil 1).

Musterlösung zu Ü 10.1

Gesamtfläche: $A = \frac{1}{2}\,|(6-0)(3-0) - (0-0)(0-0)| = \frac{1}{2}\cdot 18 = 9$.

- $A_1 = \text{Fläche}(P, V_2, V_3) = \frac{1}{2}\,|(6-2)(3-1) - (0-2)(0-1)| = \frac{1}{2}\,|8-2| = 3 \to \lambda_1 = 3/9 = \frac{1}{3}$.
- $A_2 = \text{Fläche}(P, V_3, V_1) = \frac{1}{2}\,|(0-2)(0-1) - (0-2)(3-1)| = \frac{1}{2}\,|2+4| = 3 \to \lambda_2 = 3/9 = \frac{1}{3}$.
- $A_3 = \text{Fläche}(P, V_1, V_2) = \frac{1}{2}\,|(0-2)(0-1) - (6-2)(0-1)| = \frac{1}{2}\,|2+4| = 3 \to \lambda_3 = 3/9 = \frac{1}{3}$.

Probe: $\frac{1}{3}+\frac{1}{3}+\frac{1}{3} = 1$. □ Der Punkt $P = (2,1)$ ist der **Schwerpunkt** des Dreiecks (Mittel der drei Ecken: $((0+6+0)/3,\ (0+0+3)/3) = (2,1)$) — deshalb alle Anteile gleich.

Ü 10.2 (Verändern). Jetzt kommt die Wärme nicht mehr von der Seite, sondern von **unten** — die Brandszene des Buches. Beflamme die **Unterkante** des Trägers: Halte die untere Kante (Knoten 1, 2, 3) auf 300 °C (die Brandgase) und die obere Kante (Knoten 7, 8, 9) auf 20 °C (die kühle Betondecke drückt von oben); die Mittelreihe (Knoten 4, 5, 6) bleibt frei. Sage vorher: Bleibt die Temperatur über die Höhe gleich wie im Kapitel, oder entsteht ein Gefälle von unten nach oben? Löse und deute das Farbbild. Das Skript `loesungen/kap10_ue2.py` rechnet es vor.

Musterlösung zu Ü 10.2

Jetzt sind Unter- und Oberkante festgehalten, die Mittelreihe ist frei. Aus der Symmetrie stellt sich die Mittelreihe genau dazwischen ein: alle drei Knoten (4, 5, 6) auf **160 °C**. Das Feld ist dasselbe wie im Kapitel — nur um eine Vierteldrehung gekippt: Vorher war die Temperatur auf jeder senkrechten Linie konstant und wuchs über die Länge; jetzt ist sie auf jeder waagerechten Linie konstant und fällt über die **Höhe**, von 300 °C an der beflammten Unterkante auf 20 °C an der kühlen Oberkante. Und genau das ist die Vorschau auf das Finale. Ein solches Höhengefälle — heiße Unterseite, kühle Oberseite — dehnt den unteren Rand stärker als den oberen und **krümmt** den Träger nach oben (thermisches Verbiegen, Kapitel 14, 3). Die Wärmerechnung liefert hier das Gefälle; die Verformungsrechnung macht später die Biegung daraus. Das Feld, auf dem der Träger sich verbiegt, steht damit schon.

Ü 10.3 (Übertragen). Schreibe ein eigenes **L-förmiges** Mini-Netz aus vier Dreiecken als Knoten- und Dreiecksliste auf (ein 2×2-Quadrat ohne die rechte obere Zelle — sechs Ecken, vier Dreiecke) und schicke es durch die Maschinerie: linke Kante 20 °C, rechte Kante 300 °C. Wo stellt sich die einspringende Ecke ein? Das Skript `loesungen/kap10_ue3.py` zeigt eine Musterlösung.

Musterlösung zu Ü 10.3

Ein möglicher Bauplan (gegen den Uhrzeigersinn nummeriert):

Knoten	1	2	3	4	5	6
(x, y)	(0,0)	(2,0)	(2,1)	(1,1)	(1,2)	(0,2)

Dreiecke: 1–2–4, 1–4–6, 2–3–4, 4–5–6 (Flächen $1, 1, \frac{1}{2}, \frac{1}{2}$, Summe 3 — das L aus drei Einheitsquadraten). Hält man die linke Kante (Knoten 1, 6) auf 20 °C und die rechte (Knoten 2, 3) auf 300 °C, stellt sich die einspringende Ecke (Knoten 4) auf rund 122 °C ein — näher an der kalten Seite, weil sie an das große kalte Gebiet links grenzt. Die entscheidende Lehre: Der einzige Aufwand war, die zwei Listen richtig aufzuschreiben; die Maschinerie lief unverändert.

Das Kleingedruckte

Drei ehrliche Feinheiten.

Erstens — dünne Dreiecke. Unsere Träger-Dreiecke sind mit $0{,}5$ m $\times$ $0{,}05$ m ziemlich langgestreckt (Verhältnis 10 zu 1). Das sieht man den großen Zahlen (25,25, 25) in der Elementmatrix an: Die steilen y-Steigungen entstehen aus der kurzen Höhe. Für unser gutmütiges, gleichmäßiges Problem ist das kein Problem, aber als Faustregel gilt: **Vermeide Nadeln.** Sehr spitze Dreiecke verschlechtern die Genauigkeit und machen das Gleichungssystem schwerer lösbar. Netzgeneratoren (Delaunay) achten deshalb auf gedrungene Dreiecke. Wie man Netzqualität misst, gehört zu Kapitel 12.

Zweitens — nur lineare Elemente. Wir benutzen ausschließlich **P1-Dreiecke** mit gerader Interpolation. Über einem Dreieck ist das Feld deshalb immer eine Ebene, sein Gradient konstant. Elemente höherer Ordnung (gekrümmte Ansätze) können mit weniger Dreiecken mehr, sind aber ein Thema für sich und kommen in diesem Buch nur als Ausblick vor.

Drittens — die Exaktheit ist wieder geliehen. Dass die Mittelspalte *exakt* 160 °C traf, liegt wie in Kapitel 7 daran, dass die wahre Lösung selbst eine Ebene ist (linear in x, unabhängig von y) — und eine Ebene stellen die P1-Dreiecke exakt dar. Sobald die Wahrheit krumm ist (eine einspringende Ecke, ein Loch, ein runder Rand), trifft die FEM nur noch näherungsweise, und man braucht mehr Dreiecke. Wie viel genauer ein feineres Netz wird — und wie man das ehrlich misst —, ist das Thema von Kapitel 12.

11. Rechtecke

Ein Fliesenleger, der ein rechteckiges Bad auslegt, greift nicht zu Dreiecken. Er nimmt **Rechtecke** — sie passen ohne Verschnitt, Reihe an Reihe, und man sieht sofort, welche Fliese wohin gehört. Für unseren Träger, der selbst ein braves Rechteck ist, drängt sich dieselbe Idee auf: Warum ihn mühsam in acht Dreiecke zerschneiden, wenn vier Rechtecke ihn genauso bedecken?

In 1 haben wir den Träger zur Fläche gemacht und mit Dreiecken gepflastert. Dieses Kapitel legt die **zweite Elementfamilie** daneben: das Rechteck. Und die spannende Frage ist nicht, ob es funktioniert — das tut es, mit derselben Maschinerie wie immer —, sondern *worin es sich unterscheidet*. Denn ein Rechteck ist **nicht** einfach „zwei Dreiecke im Sparpaket". Es interpoliert anders, und dieser Unterschied ist die eigentliche Lehre des Kapitels.

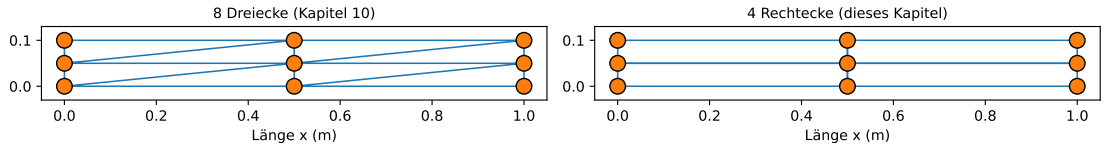

Abbildung 11.1.: Derselbe Träger, zwei Pflasterungen. Links das 8-Dreiecke-Netz aus Kapitel 10, rechts das 4-Rechtecke-Netz dieses Kapitels — beide auf demselben 3×3-Knotengitter (neun orange Knoten). Rechts sind es weniger Elemente, keine Diagonalen, und jede Zelle ist so regelmäßig wie der Träger selbst. Ob das auch die bessere Wahl ist, klärt dieses Kapitel.

Was man hier sieht: links das vertraute Dreiecksnetz, rechts das neue Rechtecknetz — dieselben neun Knoten, aber vier Rechtecke statt acht Dreiecke. Weniger Elemente, kein Diagonalen-Wirrwarr, alles rechtwinklig. Das sieht nach der aufgeräumteren Wahl aus. Ob sie es *ist*, hängt davon ab, wie ein Rechteck zwischen seinen vier Ecken interpoliert — und genau da beginnt die Geschichte.

Lernziele

Nach diesem Kapitel kannst du …

1. … auf einem Rechteck aus **vier Eckwerten** interpolieren, indem du erst in x und dann in y interpolierst („zweimal Strecke statt einmal Fläche"),
2. … erklären, warum die bilineare Fläche im Inneren **gekrümmt** ist (ein Sattel!), obwohl sie auf jeder Kante eine Gerade bleibt — und warum Rechtecke und Dreiecke deshalb **kantenkompatibel** sind,
3. … die **vier Ansatzfunktionen** des Rechtecks als Produkt zweier Hütchen hinzeichnen und ihre Eigenschaften prüfen,
4. … die **4×4-Elementmatrix** eines Rechtecks als Summe von Produkten aus 1D-Integralen aufstellen und den Träger mit vier Rechtecken vernetzen,
5. … das Rechteck-System mit dem Dreieck-System vergleichen (gleiche Lösung, **neun statt fünf** Kopplungen pro innerem Knoten) und in einem Satz sagen, was „isoparametrisch" leistet und warum dieses Buch es nicht braucht.

⚠ Naheliegende Vermutung

Vermutung: *„Ein Rechteck ist einfach zwei zusammengeklebte Dreiecke — dieselbe Interpolation, nur weniger Verwaltung."*
Warum sie naheliegt: Man kann jedes Rechteck mit einer Diagonale in zwei Dreiecke zerschneiden. Die Knoten sind dieselben, die Fläche ist dieselbe — also, so der Gedanke, muss auch das Ergebnis dasselbe sein. Das Rechteck wäre nur eine bequemere Verpackung für zwei Dreiecke.
Was stattdessen stimmt: Die bilineare Fläche über vier Ecken ist im Inneren **gekrümmt** — ein Sattel. Die zwei Dreiecke dagegen wären zwei **ebene Platten mit einem Knick** entlang der Diagonale. Und schlimmer: Je nachdem, **welche** der beiden Diagonalen man wählt, entsteht ein *anderer* Knick und ein *anderer* Wert im Inneren. Das Rechteck kennt diese Diagonalen-Willkür nicht — es hat gar keine Diagonale. Wie groß der Unterschied wird, rechnen wir in diesem Kapitel aus (er kann 280 Grad betragen).

11.1. Bilinear: zweimal interpolieren statt einmal

Fangen wir mit dem einen neuen Baustein an — dem Rest des Kapitels sieht man schon an, dass er die alte Maschinerie nicht anrührt. In 1 zerlegte ein

Punkt das Dreieck in drei Flächenanteile. Beim Rechteck gehen wir einen anderen, noch einfacheren Weg: Wir interpolieren **zweimal hintereinander die gute alte Strecke** aus Kapitel 7.

Stell dir ein quadratisches Blech vor, an dessen vier Ecken vier Temperaturfühler kleben: unten links 20 °C, unten rechts 300 °C, oben rechts 100 °C, oben links 40 °C. Gesucht ist die Temperatur an einer Stelle mittendrin. Das Rezept in zwei Schritten:

1. **In x-Richtung** interpolieren, und zwar auf der **Unterkante** und auf der **Oberkante** getrennt. Das gibt zwei Zwischenwerte — einen unten, einen oben, beide an der gewünschten x-Stelle.
2. **In y-Richtung** zwischen diesen beiden Zwischenwerten interpolieren. Fertig.

Nehmen wir den **Mittelpunkt** des Blechs (halbe Breite, halbe Höhe). Schritt eins auf der Unterkante: die Mitte zwischen 20 und 300 ist $20 + \frac{1}{2}\,(300 - 20) = 160$ °C. Auf der Oberkante: die Mitte zwischen 40 und 100 ist $40 + \frac{1}{2}\,(100 - 40) = 70$ °C. Schritt zwei, in der Höhe zwischen diesen beiden:

$$T_{\text{Mitte}} = 160 + \tfrac{1}{2}\,(70 - 160) = 160 - 45 = 115\ \text{°C}.$$

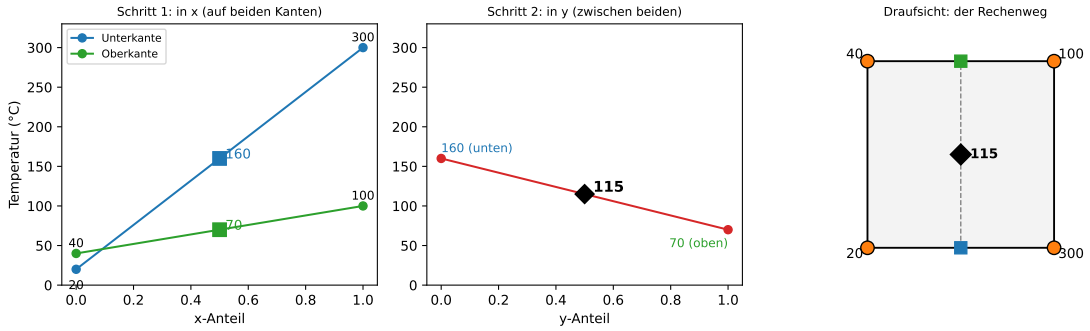

Abbildung 11.2.: Bilinear ist zweimal linear. Schritt 1 (links): auf der Unterkante zwischen 20 und 300 interpolieren (Mitte 160) und auf der Oberkante zwischen 40 und 100 (Mitte 70). Schritt 2 (Mitte): senkrecht zwischen diesen beiden Zwischenwerten interpolieren — 160 und 70 ergeben in der Blechmitte 115 °C. Rechts das Ganze noch einmal als Draufsicht mit dem Rechenweg eingezeichnet. Die drei Interpolationen sind je eine schlichte Strecke wie in Kapitel 7.

Was man hier sieht: drei ganz gewöhnliche Strecken-Interpolationen. Zwei waagerechte (auf Unter- und Oberkante, Schritt 1) und eine senk-

rechte dazwischen (Schritt 2). Kein neues Werkzeug — nur das Hütchen aus Kapitel 7, dreimal angewandt. Deshalb der Name **bilinear**: *bi* für „zweimal", *linear* für „geradlinig".

Diese drei Zeilen Handrechnung gießen wir in Python. Der Code sagt Wort für Wort, was oben stand — erst die beiden Kanten, dann dazwischen:

```
def bilineare_interpolation(eckwerte, anteil_x, anteil_y):
    """Wert an einer Stelle im Rechteck -- bilinear, also zweimal linear.

    eckwerte: [unten_links, unten_rechts, oben_rechts, oben_links].
    anteil_x: 0 = linke Kante, 1 = rechte Kante.
    anteil_y: 0 = untere Kante, 1 = obere Kante.
    """
    unten_links, unten_rechts, oben_rechts, oben_links = eckwerte

    # Schritt 1: Unter- und Oberkante in x-Richtung.
    wert_unterkante = unten_links + anteil_x * (unten_rechts - unten_links)
    wert_oberkante = oben_links + anteil_x * (oben_rechts - oben_links)

    # Schritt 2: zwischen beiden in y-Richtung.
    return wert_unterkante + anteil_y * (wert_oberkante - wert_unterkante)


ECKWERTE = [20.0, 300.0, 100.0, 40.0]   # unten links / rechts, oben rechts / links

print("Mitte  (x=0,50, y=0,50):", bilineare_interpolation(ECKWERTE, 0.5, 0.5), "°C")
print("Viertel(x=0,25, y=0,50):", bilineare_interpolation(ECKWERTE, 0.25, 0.5), "°C")
print("Mittelwert der vier Ecken:", sum(ECKWERTE) / 4.0, "°C")
```

```
Mitte  (x=0,50, y=0,50): 115.0 °C
Viertel(x=0,25, y=0,50): 72.5 °C
Mittelwert der vier Ecken: 115.0 °C
```

Interpretation der Ausgabe: In der Mitte kommen die 115 °C der Handrechnung heraus — und, ein hübscher Zufall, das ist genau der **Mittelwert der vier Ecken** ($ (20+300+100+40)/4 = 115$). Das gilt aber **nur im Zentrum**; schon beim Viertelpunkt (72,5 °C) stimmt es nicht mehr mit dem Eckmittel überein. Die „Mitte = Mittel der Ecken"-Regel merken wir uns trotzdem — beim Sattel gleich wird sie zum Kronzeugen.

11.2. Die vier Ansatzfunktionen: zwei Hütchen im Produkt

In Kapitel 7 war die Ansatzfunktion eines 1D-Knotens ein **Hütchen**: 1 am eigenen Knoten, geradlinig auf 0 zu den Nachbarn. In 1 wurde daraus das **Zelt** über Dreiecken. Beim Rechteck ist die Bauidee noch direkter: **Multipliziere ein Hütchen in x mit einem Hütchen in y.**

Ein Rechteck hat vier Ecken, also vier Ansatzfunktionen. Nimm die Ecke unten links. Ihr x-Hütchen ist 1 an der linken Kante und fällt nach rechts auf 0; ihr y-Hütchen ist 1 an der unteren Kante und fällt nach oben auf 0. Das **Produkt** der beiden ist 1 genau in der Ecke unten links und 0 an den drei anderen Ecken — die Ansatzfunktion dieser Ecke. Ebenso für die drei übrigen.

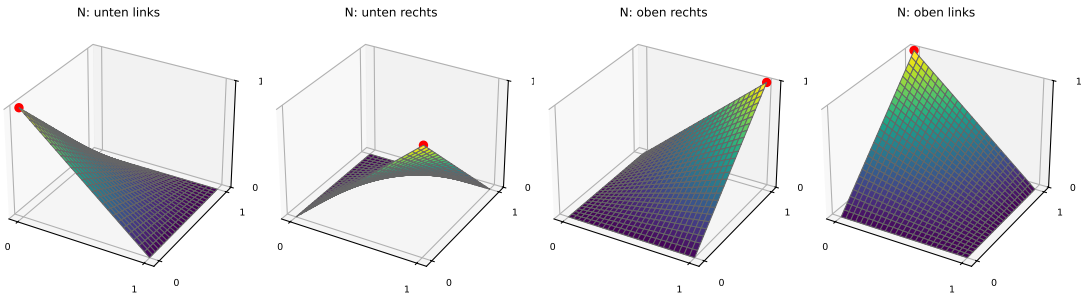

Abbildung 11.3.: Die vier Ansatzfunktionen des Rechtecks, jede über dem Einheitsquadrat. Jede steht auf Höhe 1 über ihrer eigenen Ecke (rot) und fällt zu den anderen drei Ecken auf 0. Entlang jeder Kante läuft sie geradlinig (eine Gerade), im Inneren aber ist sie leicht gewölbt — eine Sattelfläche, weil sie das Produkt zweier Hütchen ist. Genau diese Wölbung fehlt dem ebenen Dreiecks-Zelt aus Kapitel 10.

Was man hier sieht: vier Dächer, jedes über einer anderen Ecke auf Höhe 1, zu den drei anderen Ecken auf 0. Anders als das Dreiecks-Zelt (eine ebene Fläche) ist jedes dieser Dächer im Inneren **leicht gewölbt** — an den Rändern straff, in der Mitte durchgesackt. Diese Wölbung stammt vom Produkt: Eine Größe mal einer anderen, beide veränderlich, ergibt eben keine Ebene mehr.

Prüfen wir die Ansatzfunktionen mit derselben Checkliste wie in 1:

- **Ecke exklusiv.** Jede Funktion ist 1 an ihrer eigenen Ecke, 0 an den drei anderen (rote Punkte). Beim Interpolieren gibt jede Ecke also genau ihren eigenen Wert zurück — wie es sein muss.

- **Summe 1.** An jeder Stelle addieren sich die vier Funktionen zu 1 ($(1-s)(1-t) + s(1-t) + st + (1-s)t = 1$, einfach ausmultipliziert). Deshalb bleibt der interpolierte Wert immer zwischen dem kleinsten und größten Eckwert.
- **Kante linear.** Auf jeder Kante ist entweder x oder y konstant. Dann wird aus dem Produkt „Hütchen mal Hütchen" ein einzelnes Hütchen mal einer Zahl — eine schlichte **Gerade**. Und diese Gerade hängt nur von den **beiden Knoten dieser Kante** ab, nicht von den anderen zwei.

Der letzte Punkt ist der wichtigste, denn er ist genau die Eigenschaft, die in 1 die Dreiecke nahtlos zusammengehalten hat:

Kantenkompatibel — sogar mit Dreiecken

Auf jeder Rechteckkante ist der Verlauf **linear** und hängt nur von den zwei Knoten dieser Kante ab. Das ist wortgleich die Kantenkompatibilität der Dreiecke aus 1: Zwei Nachbarelemente, die sich eine Kante teilen, interpolieren dort dieselben zwei Knotenwerte — kein Sprung, keine Überlappung.
Die Folge ist bemerkenswert: Weil **beide** Familien auf ihren Kanten linear sind, darf man Rechtecke und Dreiecke sogar im **selben Netz mischen** — Rechtecke im regelmäßigen Inneren, Dreiecke am schrägen Rand. An der gemeinsamen Kante passen sie exakt zusammen (Übung 11.3 rechnet das nach).

11.3. Der Sattel: warum das Innere sich krümmt

Jetzt zur Pointe des Kapitels. Auf den Kanten ist die bilineare Fläche eine Gerade — das haben wir gerade gesehen. Aber im **Inneren**? Multipliziert man die vier Ansatzfunktionen aus, bleibt ein einziger Term übrig, den ein Dreieck nicht hat: ein Glied, in dem x **und** y zusammen vorkommen, das **gemischte Glied** $x \cdot y$. Und dieses Glied krümmt die Fläche.

Anschaulich ist das die Form, die ein an vier verschieden hoch aufgehängtes Tuch annimmt: In der einen Diagonalrichtung hängt es durch, in der anderen wölbt es sich auf — ein **Sattel** (dieselbe Form wie ein Kartoffelchip oder ein Pass zwischen zwei Berggipfeln). Am deutlichsten wird der Sattel, wenn man die Eckwerte **über Kreuz** wählt: zwei gegenüberliegende Ecken kalt, die anderen beiden heiß.

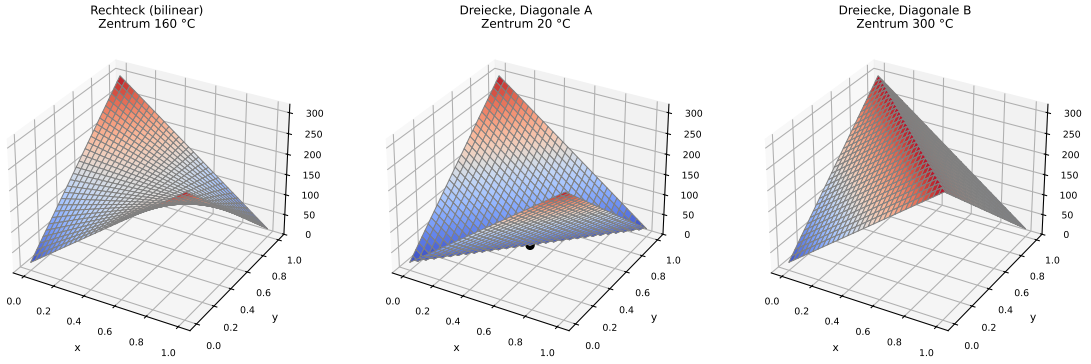

Abbildung 11.4.: Die Schlüsselgrafik. Über denselben vier Eckwerten 20/300/20/300 (über Kreuz) drei Flächen. Links das Rechteck: eine glatte, gekrümmte Sattelfläche, im Zentrum 160 °C. Mitte und rechts die zwei Dreiecksvarianten — je nachdem, welche Diagonale man schneidet, zwei ebene Platten mit Knick. Schneidet man entlang der beiden kalten Ecken (Mitte), ist das Zentrum 20 °C; entlang der beiden heißen (rechts) 300 °C. Ein und dieselben Ecken, drei völlig verschiedene Innenwerte.

Was man hier sieht: dreimal dieselben vier Ecken (20/300/20/300), drei grundverschiedene Flächen. Das Rechteck (links) spannt eine glatte, gekrümmte Sattelfläche; ihr Zentrum liegt bei **160 °C** — dem Mittel der vier Ecken, wie wir es oben schon gelernt haben. Die beiden Dreiecksvarianten (Mitte, rechts) sind dagegen zwei ebene Platten mit einem scharfen Knick entlang der Diagonale, und ihr Zentrum hängt davon ab, **welche** Diagonale man geschnitten hat.

Rechnen wir die drei Zentrumswerte nach. Das Zentrum liegt genau auf **beiden** Diagonalen. Schneidet man entlang **Diagonale A** (sie verbindet die beiden kalten 20-°C-Ecken), so ist das Zentrum der Mittelpunkt dieser Diagonale, also $(20 + 20)/2 = 20$ °C. Schneidet man entlang **Diagonale B** (die beiden heißen 300-°C-Ecken), ist es $(300 + 300)/2 = 300$ °C. Das Rechteck macht keinen Schnitt und landet bei 160 °C — exakt zwischen beiden.

> ⚠ Die Diagonalen-Willkür
>
> **Die Vermutung war:** „Rechteck = zwei Dreiecke, dieselbe Interpolation.” **Und heraus kommt:** ein Unterschied von bis zu **280 Grad** im selben Punkt.

Bei den Eckwerten 20/300/20/300 zeigt das Zentrum

20 °C (Diagonale A), 300 °C (Diagonale B), 160 °C (Rechteck).

Die zwei Dreiecksvarianten liegen 280 Grad auseinander — allein durch die Wahl der Schnittdiagonale, einer reinen Netz-Entscheidung ohne physikalischen Grund. Das ist die **Diagonalen-Willkür**: Das Dreiecksnetz muss sich für eine Diagonale entscheiden und schleppt diese Willkür in die Lösung. Das Rechteck hat keine Diagonale und keine Willkür — sein bilinearer Wert liegt neutral in der Mitte. (Bei sehr feinen Netzen schrumpft der Unterschied, weil die Zellen klein werden und die Eckwerte kaum noch über Kreuz stehen; Kapitel 12 misst das.)

11.3.1. Krümmung aus lauter Geraden

Wie kann eine Fläche gekrümmt sein, wenn sie auf jeder Kante — und sogar auf jeder waagerechten und senkrechten Linie im Inneren — eine Gerade ist? Die folgende Animation zeigt es: Eine waagerechte Scanlinie wandert von unten nach oben über das Rechteck. Auf jeder Höhe ist das Temperaturprofil in x eine **Gerade** — aber ihre beiden Endpunkte (die Werte an linker und rechter Kante) wandern mit der Höhe, und deshalb **kippt und verschiebt** sich diese Gerade laufend. Stapelt man alle diese Geraden übereinander, entsteht die gekrümmte Sattelfläche: **Krümmung aus lauter Geraden**.

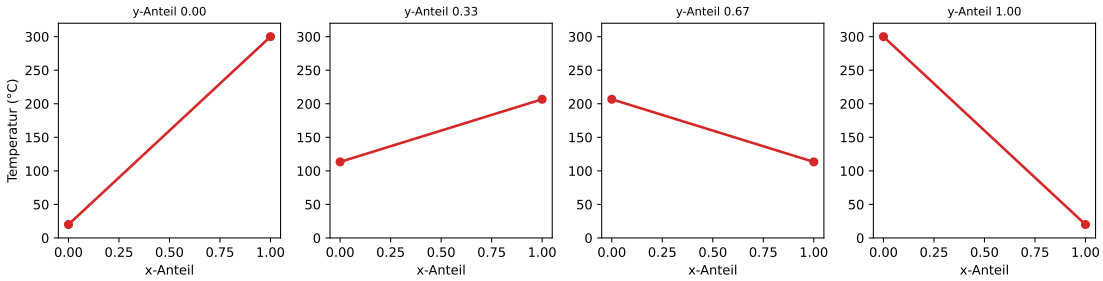

Abbildung 11.5.: Der Interpolations-Scanner als Bildreihe (in der HTML-Fassung eine Animation): das x-Profil auf vier Höhen (y-Anteil 0, ⅓, ⅔, 1). Jedes einzelne Profil ist eine Gerade, aber ihre Neigung dreht sich stetig — unten fällt sie (300 nach 20 … Moment, von 20 auf 300), oben kippt sie andersherum. Die vier Geraden zusammen deuten die gekrümmte Sattelfläche an.

Was man sieht: Jedes einzelne Profil ist kerzengerade — und trotzdem ist die Fläche, die alle Profile zusammen aufspannen, gekrümmt. Der Grund ist der gemischte $x\,y$-Term: Weil die Endpunkte der Geraden mit der Höhe wandern, dreht sich die Gerade, und eine sich drehende Gerade überstreicht eine gewölbte Fläche. Das Dreieck kann das nicht — seine Ansatzfläche ist eine starre Ebene ohne diesen Term.

11.4. Die Elementmatrix: vier mal vier, aus 1D-Bausteinen

Jetzt der Schritt, der bei der Feder die 2×2-Matrix und beim Dreieck die 3×3-Matrix lieferte: Ein Rechteck hat vier Knoten, also wird sein Steckbrief eine **4×4-Matrix**. Und weil wir uns strikt auf **achsparallele** Rechtecke beschränken (das ist die bewusste Vereinfachung dieses Buches, Strategie §7), bleibt das Ausrechnen erstaunlich einfach: Jede Ansatzfunktion ist ein Produkt „Hütchen in x mal Hütchen in y“, und deshalb zerfällt jedes Elementintegral in ein **Produkt zweier 1D-Integrale** — beide kennen wir schon.

Wir brauchen genau zwei 1D-Bausteine, beide aus früheren Kapiteln:

- **Steigung mal Steigung** auf einer Strecke der Länge L — die 1D-Matrix aus Kapitel 7: auf der Diagonale $+1/L$, daneben $-1/L$.
- **Hütchen-Überlapp** auf einer Strecke der Länge L — die Massenmatrix aus Kapitel 9: $\frac{L}{6}\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}$.

Jeder Eintrag der 4×4-Matrix ist die Summe von **zwei** Produkten dieser Bausteine:

$$K_{ij} = \lambda\, d\,\Big(\underbrace{[\text{Steigung} \times \text{Steigung in } x]\cdot[\text{Hütchen-Überlapp in } y]}_{\text{der } x\text{-Beitrag}} + \underbrace{[\text{Hütchen-Überlapp in } x]\cdot[\text{St}}_{\text{der } y\text{-Beit}}$$

In Worten: In der einen Richtung liefern die **Steigungen** den Beitrag (dort zählt, wie steil die Ansatzfunktion abfällt), in der anderen der **Überlapp** der Hütchen (dort zählt, wie sehr sich die Funktionen überdecken) — und das für x und y addiert. Das ist die ganze Rechnung. Machen wir eine Zeile von Hand.

11.4.1. Eine Zeile von Hand

Wir nehmen ein Träger-Rechteck: Breite $a = 0{,}5$ m (Länge einer Zelle), Höhe $b = 0{,}05$ m. Die Knoten gegen den Uhrzeigersinn: **1** = unten links, **2** = unten rechts, **3** = oben rechts, **4** = oben links. Wir rechnen die **erste Zeile** aus — die Kopplungen von Knoten 1 zu allen vieren. Der Vorfaktor ist $\lambda\, d = 50 \cdot 0{,}1 = 5\ \frac{\mathrm{W}}{\mathrm{m\,K}} \cdot \mathrm{m}$.

Die 1D-Bausteine für $a = 0{,}5$ und $b = 0{,}05$: Steigung mal Steigung ist $\pm 1/a = \pm 2$ bzw. $\pm 1/b = \pm 20$; der Hütchen-Überlapp ist auf der Diagonale $2a/6 = a/3$ bzw. $b/3$ und daneben $a/6$ bzw. $b/6$. Damit die vier Einträge der ersten Zeile (Knoten 1 ist in x das **linke**, in y das **untere** Hütchen):

$$
\begin{aligned}
K_{11} &= 5\left(\tfrac{1}{a}\cdot\tfrac{b}{3} + \tfrac{a}{3}\cdot\tfrac{1}{b}\right) = 5\,(0{,}0333 + 3{,}3333) = 16{,}83,\\
K_{12} &= 5\Big(\underbrace{-\tfrac{1}{a}}_{x\text{-Steig.}}\cdot\tfrac{b}{3} + \tfrac{a}{6}\cdot\underbrace{\tfrac{1}{b}}_{y\text{-Steig.}}\Big) = 5\,(-0{,}0333 + 1{,}6667) = +8{,}17,\\
K_{13} &= 5\left(-\tfrac{1}{a}\cdot\tfrac{b}{6} - \tfrac{a}{6}\cdot\tfrac{1}{b}\right) = 5\,(-0{,}0167 - 1{,}6667) = -8{,}42,\\
K_{14} &= 5\left(\tfrac{1}{a}\cdot\tfrac{b}{6} - \tfrac{a}{3}\cdot\tfrac{1}{b}\right) = 5\,(0{,}0167 - 3{,}3333) = -16{,}58.
\end{aligned}
$$

Die **Probe** ist dieselbe wie immer: $16{,}83 + 8{,}17 - 8{,}42 - 16{,}58 = 0$ — die Zeile summiert sich zu null, weil ein gleichmäßiges Aufheizen aller vier Knoten keinen Wärmestrom erzeugen darf (wie bei Feder und Dreieck). Alle sechzehn Einträge zusammen ergeben den Steckbrief. Lassen wir das Programm ihn ausrechnen und die Probe machen:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from rechtecke import baue_rechteck_matrix

# Träger-Rechteck 0,5 m x 0,05 m, Knoten 1-2-3-4 gegen den Uhrzeigersinn.
ecken = [(0.0, 0.0), (0.5, 0.0), (0.5, 0.05), (0.0, 0.05)]
steckbrief = baue_rechteck_matrix(ecken, 50.0, 0.1)

print("4x4-Steckbrief des Träger-Rechtecks (W/K):")
for zeile in steckbrief:
    summe = zeile[0] + zeile[1] + zeile[2] + zeile[3]
    print("  [ %7.2f %7.2f %7.2f %7.2f ]   Zeilensumme = %.2f"
          % (zeile[0], zeile[1], zeile[2], zeile[3], summe))
```

```
4x4-Steckbrief des Träger-Rechtecks (W/K):
  [   16.83    8.17   -8.42  -16.58 ]   Zeilensumme = 0.00
  [    8.17   16.83  -16.58   -8.42 ]   Zeilensumme = 0.00
  [   -8.42  -16.58   16.83    8.17 ]   Zeilensumme = -0.00
  [  -16.58   -8.42    8.17   16.83 ]   Zeilensumme = -0.00
```

Interpretation der Ausgabe: Zahl für Zahl die Handrechnung, und jede Zeilensumme null. Zwei Dinge fallen auf, und beide kommen vom **Nadel-Zuschnitt** des Rechtecks (10-mal so lang wie hoch):

- Die Kopplung **quer über die kurze Höhe** ($K_{14} = -16{,}58$, zu Knoten 4 direkt darüber) ist mit Abstand die stärkste — genau wie bei den Dreiecken in 1 die -25 die kurze Höhe überbrückte. Kurze Strecke, steile Ansatzfunktion, große Zahl.
- Die Kopplung **längs der Länge** ($K_{12} = +8{,}17$, zu Knoten 2 daneben) ist sogar **positiv**. Das ist ungewöhnlich — normalerweise sind Nachbarkopplungen negativ. Es ist ein Warnzeichen des extremen Seitenverhältnisses: Bei so einer Nadel überwiegt der y-Beitrag den x-Beitrag so stark, dass das Vorzeichen kippt. Für unser gutmütiges Problem schadet das nicht (die Lösung stimmt), aber es ist ein erster Hinweis, dass entartete Elemente ihre Tücken haben — mehr dazu im Kleingedruckten.

Der Rechteck-Steckbrief für beliebiges Seitenverhältnis

Für ein Rechteck mit Breite a und Höhe b hat der Q1-Steckbrief eine feste Gestalt, in der nur die beiden Seitenverhältnisse $\frac{b}{a}$ (Längs-Anteil) und $\frac{a}{b}$ (Quer-Anteil) auftauchen. Für ein **Quadrat** ($a = b$) wird er besonders schön:

$$\mathbf{K}_{\text{Quadrat}} = \frac{\lambda\, d}{6} \begin{pmatrix} 4 & -1 & -2 & -1 \\ -1 & 4 & -1 & -2 \\ -2 & -1 & 4 & -1 \\ -1 & -2 & -1 & 4 \end{pmatrix}.$$

Auf der Diagonale die 4, zu den beiden Kantennachbarn -1, und zur **diagonal gegenüberliegenden** Ecke -2. Diese -2 ist der entscheidende Unterschied zum Dreieck: Das Rechteck koppelt jeden Knoten auch mit der **Eck-Diagonale**. Genau daher kommt der Sattel — und gleich der 9-Punkt-Stern.

11.5. Der Rechteck-Träger: gleiche Lösung, anderes System

Jetzt vernetzen wir den ganzen Träger mit Rechtecken und vergleichen ihn mit dem Dreiecksnetz aus 1 — auf **demselben** 3×3-Knotengitter. Der Bauplan ist dieselbe Knotenliste wie in 1, nur bleibt jede der vier Gitterzellen jetzt **ein Rechteck**, statt in zwei Dreiecke geteilt zu werden.

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from rechtecke import assembliere_rechtecke
from matrix_drucker import drucke_matrix

knoten = [(0.0, 0.0), (0.5, 0.0), (1.0, 0.0),
          (0.0, 0.05), (0.5, 0.05), (1.0, 0.05),
          (0.0, 0.10), (0.5, 0.10), (1.0, 0.10)]
rechtecke = [(0, 1, 4, 3), (1, 2, 5, 4), (3, 4, 7, 6), (4, 5, 8, 7)]

matrix = assembliere_rechtecke(knoten, rechtecke, 50.0, 0.1)
print("Systemmatrix des 4-Rechtecke-Trägers (9x9, W/K), Nullen als Punkte:")
print()
drucke_matrix(matrix)
```

```
Systemmatrix des 4-Rechtecke-Trägers (9x9, W/K), Nullen als Punkte:

     17       8       .     -17      -8       .       .       .       .
      8      34       8      -8     -33      -8       .       .       .
      .       8      17       .      -8     -17       .       .       .
    -17      -8       .      34      16       .     -17      -8       .
     -8     -33      -8      16      67      16      -8     -33      -8
      .      -8     -17       .      16      34       .      -8     -17
      .       .       .     -17      -8       .      17       8       .
      .       .       .      -8     -33      -8       8      34       8
      .       .       .       .      -8     -17       .       8      17
```

Interpretation der Ausgabe: ein deutlich **volleres** Band als beim Dreiecksnetz. Vergleiche die mittlere Zeile (Knoten 5, die Trägermitte): Beim Dreiecksnetz aus 1 hatte sie nur fünf Einträge (Knoten 5 und seine vier direkten Kantennachbarn), alle Diagonalfelder waren leer. Hier ist die Zeile von Knoten 5 **komplett voll** — Knoten 5 koppelt mit **allen acht** Nachbarn,

auch den vier Eck-Diagonalen. Abbildung 11.6 stellt beide Kopplungsmuster nebeneinander.

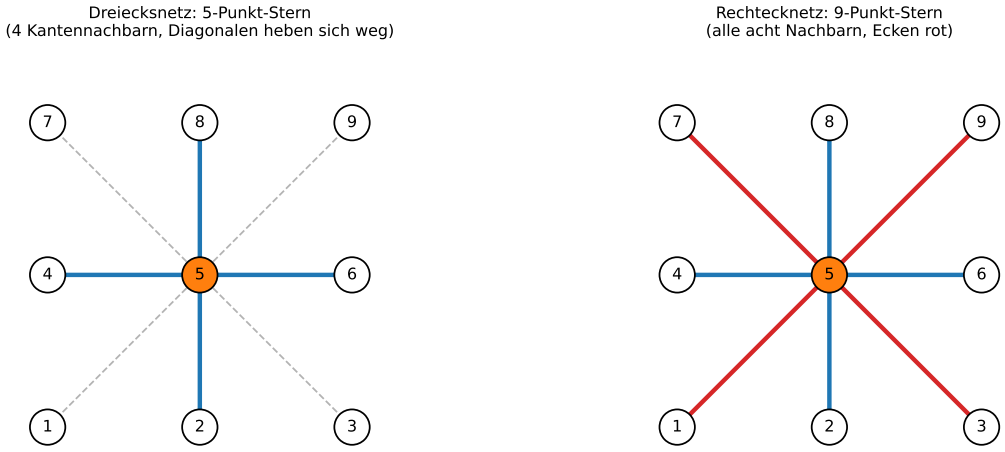

Abbildung 11.6.: Zwei Kopplungsmuster für denselben Mittelknoten. Links das Dreiecksnetz (Kapitel 10): der innere Knoten 5 koppelt nur mit seinen vier direkten Kantennachbarn — ein 5-Punkt-Stern. Die Kopplungen zu den vier Eck-Diagonalen heben sich exakt weg (Diagonalen als graue gestrichelte Linien, aber keine Verbindung). Rechts das Rechtecknetz: derselbe Knoten koppelt mit allen acht Nachbarn, Ecken eingeschlossen — ein 9-Punkt-Stern. Die vier Eck-Kopplungen (rot) sind der Unterschied.

Was man hier sieht: links koppelt der Mittelknoten mit vier Nachbarn, rechts mit acht. Der Unterschied sind die vier **Eck-Kopplungen** (rot) — genau die Diagonalnachbarn, die der $x\,y$-Term des Rechtecks ins Spiel bringt und die dem ebenen Dreieck fehlen. Beim Dreiecksnetz auf diesem regelmäßigen Gitter heben sich die Diagonalbeiträge sogar **exakt weg** (deshalb der schlanke 5-Punkt-Stern); beim Rechteck bleiben sie stehen. Mehr Kopplungen heißt: ein etwas volleres Gleichungssystem, das aber derselbe Löser aus 2 knackt.

Vorhersage-Punkt

Bevor wir lösen: Wir halten wie in 1 die linke Kante auf 20 °C und die rechte auf 300 °C. Das Dreiecksnetz lieferte damals die Mittelspalte auf exakt 160 °C. Jetzt rechnet das **Rechteck**netz dasselbe Problem. Erwartest du wieder 160 °C — oder muss durch die andere Interpolati-

> on und die neun Kopplungen etwas anderes herauskommen? Leg dich fest.

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from rechtecke import assembliere_rechtecke, reduziere_system, setze_ein
from loeser import gauss_seidel

knoten = [(0.0, 0.0), (0.5, 0.0), (1.0, 0.0),
          (0.0, 0.05), (0.5, 0.05), (1.0, 0.05),
          (0.0, 0.10), (0.5, 0.10), (1.0, 0.10)]
rechtecke = [(0, 1, 4, 3), (1, 2, 5, 4), (3, 4, 7, 6), (4, 5, 8, 7)]

matrix = assembliere_rechtecke(knoten, rechtecke, 50.0, 0.1)
randwerte = [20.0, None, 300.0, 20.0, None, 300.0, 20.0, None, 300.0]
kleine_matrix, kleine_seite, frei = reduziere_system(matrix, [0.0] * 9, randwerte)
loesung_frei = gauss_seidel(kleine_matrix, kleine_seite, [0.0] * len(frei),
                            0.00000001, 5000)
temperaturen = setze_ein(loesung_frei, frei, randwerte)

print("Knoten |   Ort (x, y)    | Temperatur")
print("-------+-----------------+-----------")
for i in range(9):
    x, y = knoten[i]
    print("   %d   | (%.2f, %.3f) m | %6.1f °C" % (i + 1, x, y, temperaturen[i]))
```

```
Knoten |   Ort (x, y)    | Temperatur
-------+-----------------+-----------
   1   | (0.00, 0.000) m |   20.0 °C
   2   | (0.50, 0.000) m |  160.0 °C
   3   | (1.00, 0.000) m |  300.0 °C
   4   | (0.00, 0.050) m |   20.0 °C
   5   | (0.50, 0.050) m |  160.0 °C
   6   | (1.00, 0.050) m |  300.0 °C
   7   | (0.00, 0.100) m |   20.0 °C
   8   | (0.50, 0.100) m |  160.0 °C
   9   | (1.00, 0.100) m |  300.0 °C
```

Interpretation der Ausgabe: wieder **160 °C** in der ganzen Mittelspalte — Zahl für Zahl dasselbe Ergebnis wie beim Dreiecksnetz. Das ist die Auflösung:

⚠ Vermutung aufgelöst

Gleiche Lösung, verschiedenes System. Bei diesem symmetrischen Rand (links überall kalt, rechts überall heiß) ist die wahre Temperatur eine **Gerade in** x, unabhängig von der Höhe. Und eine Gerade in x stellen Dreiecke **und** Rechtecke exakt dar — beim Rechteck ist auf jeder waagerechten Linie das Profil ja gerade, und der krümmende $x\,y$-Term wird nur gebraucht, wenn die Ecken über Kreuz stehen. Deshalb dieselben 160 °C.
Verschieden ist nur der **Weg dorthin**: neun Kopplungen statt fünf, ein volleres Band, ein anderer Steckbrief. Bei der geraden Lösung sieht man den Unterschied im Ergebnis nicht — aber sobald die Wahrheit *krumm* wird (eine beflammte Ecke, ein Loch, ein runder Rand), interpolieren Rechteck und Dreieck verschieden, und dann zählt, welche Familie die Wirklichkeit besser trifft. Das ist die Frage von Kapitel 12.

Das Temperatur-Farbbild bestätigt die Streifen — wie bei den Dreiecken hängt die Temperatur nur von der Länge ab, nicht von der Höhe.

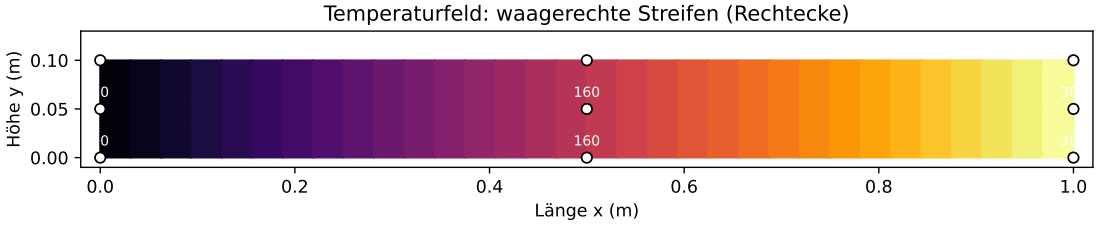

Abbildung 11.7.: Das Temperaturfeld des Rechteck-Trägers. Die Farbe wächst gleichmäßig von 20 °C links auf 300 °C rechts, die Höhenlinien stehen senkrecht — waagerechte Streifen, Mittelspalte 160 °C. Ununterscheidbar vom Dreiecks-Farbbild in Kapitel 10, obwohl das System darunter ein anderes ist.

Was man hier sieht: senkrechte Farbübergänge, waagerechte Streifen — dasselbe Bild wie in 1. Die Rechteck-Zellen sind bilinear gefärbt, aber bei diesem geraden Feld sieht man die Krümmung nirgends, weil es keine gibt. Erst ein unsymmetrischer Rand (eine beflammte Ecke) würde Rechteck und Dreieck sichtbar auseinandertreiben.

11.6. Die interaktive Einheit: das Rechteck-Labor

Jetzt bist du dran. Das Rechteck-Labor hat zwei Teile. **Teil 1** lässt dich vier Eckwerte und einen Punkt setzen und vergleicht den bilinearen Wert direkt mit **beiden** Dreiecks-Antworten (je eine pro Diagonale). **Teil 2** ist der ganze Träger mit einem **Umschalter** zwischen Dreiecken und Rechtecken auf demselben Randwertproblem.

11.6.1. Teil 1 — bilinear gegen beide Dreiecke

> Vorhersage-Punkt
>
> **Bevor du es laufen lässt:** Setze die Eckwerte über Kreuz (20/300/20/300) und den Punkt ins Zentrum. Der bilineare Wert wird 160 °C sein. Und die beiden Dreiecks-Antworten? Liegt der bilineare Wert **über** beiden, **unter** beiden oder **zwischen** ihnen? Leg dich fest, bevor du die Zahlen siehst.

Setze `ECKWERTE` (gegen den Uhrzeigersinn ab links unten) und den `PUNKT` (Anteile in x und y, je 0 bis 1). Probiere die Kreuz-Belegung, dann eine gleichmäßig steigende — wann sind sich alle drei Antworten einig?

```
#| autorun: true
# Vier Eckwerte (unten links / unten rechts / oben rechts / oben links)
# und ein Punkt als Anteile in x und y (0 = Kante links/unten, 1 = rechts/oben).
ECKWERTE = [20.0, 300.0, 20.0, 300.0]   # ueber Kreuz -> maximaler Sattel
PUNKT_X = 0.5
PUNKT_Y = 0.5


def bilinear(eckwerte, sx, sy):
    """Bilinearer Wert: erst in x auf beiden Kanten, dann in y."""
    ul, ur, orr, ol = eckwerte
    unten = ul + sx * (ur - ul)
    oben = ol + sx * (orr - ol)
    return unten + sy * (oben - unten)


def dreieck_wert(eckwerte, sx, sy, diagonale):
    """Wert auf zwei Dreiecken, je nach Schnittdiagonale (A oder B).

    Diagonale A schneidet von unten links nach oben rechts,
```

```
    Diagonale B von unten rechts nach oben links.
    """
    ul, ur, orr, ol = eckwerte
    if diagonale == "A":
        if sy <= sx:                        # unteres Dreieck: ul, ur, or
            return ul + (ur - ul) * sx + (orr - ur) * sy
        return ul + (orr - ol) * sx + (ol - ul) * sy   # oberes: ul, or, ol
    else:
        if sy <= 1.0 - sx:                  # unteres Dreieck: ul, ur, ol
            return ul + (ur - ul) * sx + (ol - ul) * sy
        return ur + (orr - ur) * sy + (ol - orr) * (1.0 - sx)  # oberes: ur, or, ol


wert_bilinear = bilinear(ECKWERTE, PUNKT_X, PUNKT_Y)
wert_a = dreieck_wert(ECKWERTE, PUNKT_X, PUNKT_Y, "A")
wert_b = dreieck_wert(ECKWERTE, PUNKT_X, PUNKT_Y, "B")

print("Eckwerte:", ECKWERTE)
print("Punkt (x-Anteil, y-Anteil):", PUNKT_X, PUNKT_Y)
print()
print("Rechteck (bilinear)        : %6.1f Grad" % wert_bilinear)
print("Dreiecke, Diagonale A      : %6.1f Grad" % wert_a)
print("Dreiecke, Diagonale B      : %6.1f Grad" % wert_b)
print()
print("Spanne zwischen den Dreiecken: %.1f Grad" % abs(wert_a - wert_b))

# --- Abbildung ---
import matplotlib.pyplot as plt
namen = ["Rechteck\n(bilinear)", "Dreieck A", "Dreieck B"]
werte = [wert_bilinear, wert_a, wert_b]
farben = ["tab:green", "tab:blue", "tab:orange"]
fig, ax = plt.subplots(figsize=(5.5, 4.0))
ax.bar(namen, werte, color=farben)
for i, w in enumerate(werte):
    ax.text(i, w + 3, "%.0f" % w, ha="center", fontsize=11)
ax.set_ylabel("Temperatur am Punkt (Grad)")
ax.set_title("Bilinear gegen beide Dreiecks-Varianten")
plt.tight_layout()
plt.show()
```

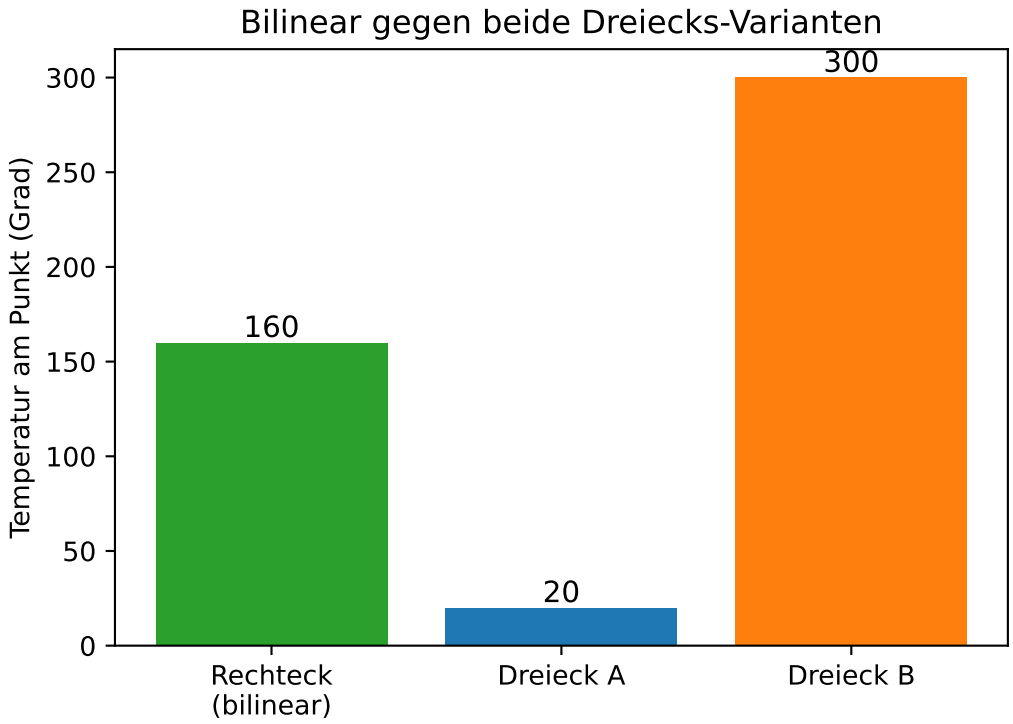

Abbildung 11.8.: Vorgerenderte Fassung von Teil 1: über Kreuz (20/300/20/300) im Zentrum liefert das Rechteck 160 °C, Dreieck A 20 °C, Dreieck B 300 °C. Der bilineare Wert liegt exakt ZWISCHEN den beiden Dreiecks-Antworten. Im Browser ersetzt dein eigenes Ergebnis dieses Bild.

Die Antwort auf die Vorhersage: Der bilineare Wert liegt **zwischen** den beiden Dreiecks-Antworten — im Kreuz-Fall genau in der Mitte (160 zwischen 20 und 300). Das Rechteck mittelt gerade das aus, worüber die zwei Dreiecke sich streiten.

11.6.2. Teil 2 — der Träger mit Elementtyp-Umschalter

Jetzt der ganze Träger. Setze `ELEMENTTYP` auf `"dreiecke"` oder `"rechtecke"` und die Randtemperaturen — das Labor baut das passende System, löst es und zählt die Kopplungen des Mittelknotens. Achte darauf: Die **Lösung** ändert sich beim Umschalten kaum, die **Kopplungszahl** springt von 5 auf 9.

```
#| autorun: true
# Umschalter: "dreiecke" (Kapitel 10) oder "rechtecke" (dieses Kapitel).
```

```
ELEMENTTYP = "rechtecke"
LINKS = 20.0
RECHTS = 300.0

KNOTEN = [(0.0, 0.0), (0.5, 0.0), (1.0, 0.0),
          (0.0, 0.05), (0.5, 0.05), (1.0, 0.05),
          (0.0, 0.10), (0.5, 0.10), (1.0, 0.10)]
DREIECKE = [(0, 1, 4), (0, 4, 3), (1, 2, 5), (1, 5, 4),
            (3, 4, 7), (3, 7, 6), (4, 5, 8), (4, 8, 7)]
RECHTECKE = [(0, 1, 4, 3), (1, 2, 5, 4), (3, 4, 7, 6), (4, 5, 8, 7)]


def dreieck_matrix(ecken):
    """3x3-Steckbrief eines P1-Dreiecks (Steigung mal Steigung mal Flaeche)."""
    (xa, ya), (xb, yb), (xc, yc) = ecken
    zwei = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    flaeche = abs(zwei) / 2.0
    steig_x = [(yb - yc) / zwei, (yc - ya) / zwei, (ya - yb) / zwei]
    steig_y = [(xc - xb) / zwei, (xa - xc) / zwei, (xb - xa) / zwei]
    matrix = [[0.0, 0.0, 0.0], [0.0, 0.0, 0.0], [0.0, 0.0, 0.0]]
    for i in range(3):
        for j in range(3):
            matrix[i][j] = 50.0 * 0.1 * flaeche * (steig_x[i] * steig_x[j]
                                                   + steig_y[i] * steig_y[j])
    return matrix


def rechteck_matrix(ecken):
    """4x4-Steckbrief eines Q1-Rechtecks (1D-Bausteine im Produkt)."""
    a = ecken[1][0] - ecken[0][0]
    b = ecken[3][1] - ecken[0][1]
    steif_x = [[1.0 / a, -1.0 / a], [-1.0 / a, 1.0 / a]]
    steif_y = [[1.0 / b, -1.0 / b], [-1.0 / b, 1.0 / b]]
    masse_x = [[2.0 * a / 6.0, a / 6.0], [a / 6.0, 2.0 * a / 6.0]]
    masse_y = [[2.0 * b / 6.0, b / 6.0], [b / 6.0, 2.0 * b / 6.0]]
    hut_x = [0, 1, 1, 0]
    hut_y = [0, 0, 1, 1]
    matrix = [[0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0],
              [0.0, 0.0, 0.0, 0.0], [0.0, 0.0, 0.0, 0.0]]
    for i in range(4):
        for j in range(4):
            teil_x = steif_x[hut_x[i]][hut_x[j]] * masse_y[hut_y[i]][hut_y[j]]
            teil_y = masse_x[hut_x[i]][hut_x[j]] * steif_y[hut_y[i]][hut_y[j]]
```

```
            matrix[i][j] = 50.0 * 0.1 * (teil_x + teil_y)
    return matrix


def gauss_seidel(matrix, seite, start, toleranz, runden):
    """Loest das System naeherungsweise (aus Kapitel 4)."""
    n = len(matrix)
    wert = list(start)
    for runde in range(runden):
        groesste = 0.0
        for i in range(n):
            summe = seite[i]
            for j in range(n):
                if j != i:
                    summe = summe - matrix[i][j] * wert[j]
            neu = summe / matrix[i][i]
            aenderung = neu - wert[i]
            if aenderung < 0.0:
                aenderung = -aenderung
            if aenderung > groesste:
                groesste = aenderung
            wert[i] = neu
        if groesste < toleranz:
            break
    return wert


# System je nach Elementtyp assemblieren.
matrix = []
for zeile in range(9):
    neue_zeile = []
    for spalte in range(9):
        neue_zeile.append(0.0)
    matrix.append(neue_zeile)
if ELEMENTTYP == "dreiecke":
    for tri in DREIECKE:
        ecken = [KNOTEN[tri[0]], KNOTEN[tri[1]], KNOTEN[tri[2]]]
        klein = dreieck_matrix(ecken)
        for a in range(3):
            for b in range(3):
                matrix[tri[a]][tri[b]] = matrix[tri[a]][tri[b]] + klein[a][b]
else:
    for rec in RECHTECKE:
```

```
        ecken = [KNOTEN[rec[0]], KNOTEN[rec[1]], KNOTEN[rec[2]], KNOTEN[rec[3]]]
        klein = rechteck_matrix(ecken)
        for a in range(4):
            for b in range(4):
                matrix[rec[a]][rec[b]] = matrix[rec[a]][rec[b]] + klein[a][b]

# Kopplungen des Mittelknotens (Nummer 5, Index 4) zaehlen.
kopplungen = 0
for wert in matrix[4]:
    if abs(wert) > 1e-9:
        kopplungen = kopplungen + 1

# Randwerte einbauen (Streich-Rezept) und loesen.
randwerte = []
for (x, y) in KNOTEN:
    if x == 0.0:
        randwerte.append(LINKS)
    elif x == 1.0:
        randwerte.append(RECHTS)
    else:
        randwerte.append(None)

frei = []
for k in range(9):
    if randwerte[k] is None:
        frei.append(k)
kleine_matrix = []
kleine_seite = []
for i in frei:
    zeile = []
    for j in frei:
        zeile.append(matrix[i][j])
    kleine_matrix.append(zeile)
    beitrag = 0.0
    for j in range(9):
        if randwerte[j] is not None:
            beitrag = beitrag + matrix[i][j] * randwerte[j]
    kleine_seite.append(-beitrag)

loesung_frei = gauss_seidel(kleine_matrix, kleine_seite, [0.0] * len(frei),
                            0.00000001, 5000)
temperaturen = list(randwerte)
for k in range(len(frei)):
```

```
    temperaturen[frei[k]] = loesung_frei[k]

print("Elementtyp:", ELEMENTTYP)
print("Kopplungen des Mittelknotens 5 (inkl. sich selbst):", kopplungen)
print()
print("Temperaturen der Mittelspalte (Knoten 2, 5, 8):")
print("  %.1f  %.1f  %.1f  Grad" % (temperaturen[1], temperaturen[4], temperaturen[7]))

# --- Abbildung ---
import matplotlib.pyplot as plt
xs = [p[0] for p in KNOTEN]
ys = [p[1] for p in KNOTEN]
fig, ax = plt.subplots(figsize=(8.5, 3.0))
streu = ax.scatter(xs, ys, c=temperaturen, cmap="inferno", s=260,
                   edgecolors="black", vmin=20, vmax=300)
for i in range(9):
    ax.annotate("%.0f" % temperaturen[i], (xs[i], ys[i]),
                textcoords="offset points", xytext=(0, 9), ha="center",
                fontsize=8)
fig.colorbar(streu, ax=ax, label="Temperatur (Grad)")
ax.set_title("Träger, Elementtyp: " + ELEMENTTYP)
ax.set_xlabel("Laenge x (m)")
ax.set_ylabel("Hoehe y (m)")
ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

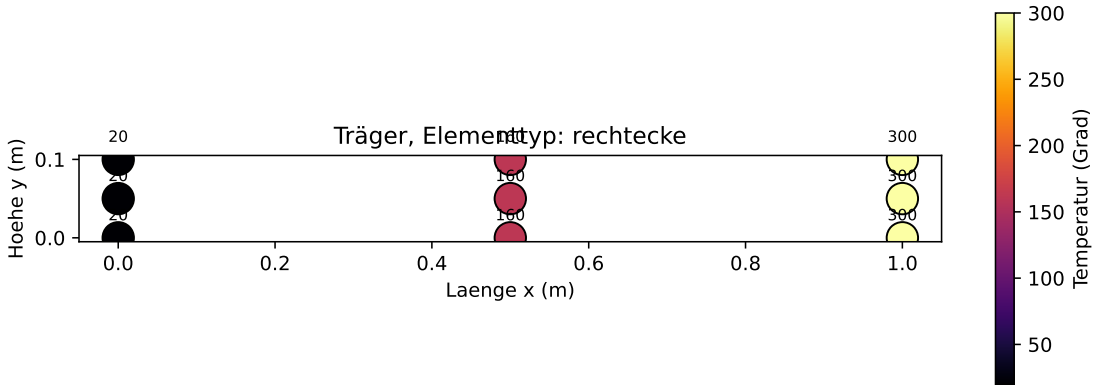

Abbildung 11.9.: Vorgerenderte Fassung von Teil 2: der Träger mit Rechtecken, Rand 20/300, Mittelspalte 160 °C, Mittelknoten mit 9 Kopplungen. Schaltet man im Browser auf „dreiecke", bleibt die Lösung 160 °C, aber die Kopplungszahl fällt auf 5.

Was man sieht: Schalte zwischen `"dreiecke"` und `"rechtecke"` hin und her — die Mittelspalte bleibt auf 160 °C, aber die Kopplungszahl springt zwischen 5 und 9. Genau das ist die Botschaft: **gleiche Lösung, verschiedenes System.** Im Finale (3) sitzt derselbe Umschalter, dann darfst du für ein echtes Problem entscheiden, welche Familie du nimmst.

11.7. Das Kapitel-Programm

Das vollständige, eigenständig lauffähige Skript liegt in `programme/kap11/kap11_rechtecke_fem.py`. Es baut das 4-Rechtecke-Netz, rechnet den Steckbrief eines Träger-Rechtecks vor, druckt die assemblierte 9×9-Matrix, zählt die Kopplungen im Vergleich zum Dreiecksnetz und löst das Randwertproblem — dieselben Schritte wie im Kapitel, an einem Stück. Die Rechteck-Bausteine (`bilineare_interpolation`, `baue_rechteck_matrix`, `assembliere_rechtecke`) liegen im gemeinsamen Modul `programme/gemeinsam/rechtecke.py`, das Kapitel 12 und das Finale 3 über den Elementtyp-Umschalter mitbenutzen. Das Streich-Rezept für Randwerte teilt es sich mit dem Dreiecks-Modul — es ist ja elementtyp-unabhängig.

11.8. Ausblick und Kleingedrucktes

11.8.1. Was Profi-Programme mit schiefen Vierecken tun

Wir haben uns strikt auf **achsparallele** Rechtecke beschränkt. Das war kein Zufall, sondern die tragende Vereinfachung dieses Kapitels — und sie hat einen Preis, den wir ehrlich benennen.

Panorama: isoparametrische Elemente

Ein echtes Bauteil hat krumme Ränder und schiefe Ecken; da passt kein achsparalleles Rechteck mehr. Profi-Programme (OGS, Abaqus, ANSYS & Co.) benutzen deshalb **schiefe Vierecke** — und einen eleganten Trick, um sie trotzdem zu berechnen:

1. Man rechnet nie auf dem schiefen Viereck selbst, sondern auf einem **Referenzquadrat** mit den bequemen Koordinaten von -1 bis $+1$. Dort sehen die vier Ansatzfunktionen genauso aus wie bei uns.
2. Dieselben vier bilinearen Funktionen bilden das Referenzquadrat auf das schiefe Viereck ab — die Geometrie wird mit **demselben** Ansatz beschrieben wie die Lösung. Daher der Name **isoparametrisch** (griechisch *isos* = gleich): gleiche Parameter für Form und Feld.
3. Beim Umrechnen der Integrale vom Quadrat aufs Viereck taucht ein Dehnungsfaktor auf, die **Jacobi-Determinante**, und die Integrale löst man nicht mehr von Hand, sondern mit **Gauß-Quadratur** (ein paar geschickt gewählte Stützstellen, wie die Streifen aus Kapitel 5, nur cleverer).

Für unseren rechteckigen Träger ist all das **unnötig** — die Jacobi-Determinante wäre eine Konstante, die Gauß-Quadratur exakt, und wir bekämen genau die Matrix, die wir oben von Hand gerechnet haben. Für krumme Ränder aber ist es **unverzichtbar**. Es ist der Standardweg jedes ernsthaften FEM-Codes und ein schönes Thema für ein weiterführendes Buch. Merke nur: *dieselben vier Hütchen-Produkte* stecken auch dort drin — wir haben den Kern schon in der Hand.

11.8.2. Lumping: der Eimer, geviertelt

In Kapitel 9 haben wir die **Massenmatrix** (den Speicherterm der Zeit) durch **Lumping** auf die Diagonale geklumpt: Jeder Knoten bekam seinen eigenen „Eimer" Wärmekapazität. Beim 1D-Träger teilte sich jedes Element hälftig

auf seine zwei Knoten auf. Beim Rechteck ist es genauso einfach, nur durch vier:

Q1-Lumping — jeder Ecke ein Viertel

Die verteilte Wärmekapazität eines Rechtecks wird beim Lumping zu **gleichen Teilen auf seine vier Ecken** geschoben — ein Viertel je Ecke. Für eine Trägerzelle ($0{,}5\ \text{m} \times 0{,}05\ \text{m} \times 0{,}1\ \text{m}$ Tiefe) ist die Zellkapazität

$$\rho\, c \cdot V = 7850 \cdot 460 \cdot (0{,}5 \cdot 0{,}05 \cdot 0{,}1) = 9027{,}5\ \tfrac{\text{J}}{\text{K}},$$

also bekommt jede der vier Ecken $9027{,}5/4 = 2256{,}875\ \frac{\text{J}}{\text{K}}$. Ein innerer Knoten, an dem vier Rechtecke zusammenstoßen, sammelt viermal ein Viertel — eine volle Zellkapazität. Das ist dasselbe Rezept wie in Kapitel 9 (Gesamtmasse bleibt erhalten, nur die Verteilung wird diagonal), nur eben durch vier statt durch zwei geteilt. Damit wird der explizite Zeitschritt auch beim Rechteck-Träger eine simple Division je Knoten.

11.8.3. Drei ehrliche Feinheiten

Erstens — die Nadel rächt sich langsam. Unsere Träger-Rechtecke sind mit 10 : 1 stark langgestreckt, und das trieb die Elementmatrix zu extremen Zahlen (bis zur positiven Nachbarkopplung $K_{12} = +8{,}17$). Für unser gerades, gutmütiges Feld schadet das nichts. Aber wie bei den „Nadel"-Dreiecken in 1 gilt: Sehr langgestreckte Elemente verschlechtern bei krummen Lösungen die Genauigkeit. Faustregel wie dort — je gedrungener, desto besser; Kapitel 12 misst es.

Zweitens — nur bilineare Elemente. Wir benutzen ausschließlich das **Q1-Rechteck** mit vier Ecken und bilinearer Interpolation. Elemente höherer Ordnung (mit Zusatzknoten auf den Kanten, gekrümmte Ansätze) können mit weniger Elementen mehr, sind aber ein Thema für sich und kommen hier nur als Ausblick vor.

Drittens — die Exaktheit ist wieder geliehen. Dass die Mittelspalte *exakt* 160 °C traf, liegt wie in Kapitel 7 und 1 daran, dass die wahre Lösung selbst eine Gerade in x ist — und die stellt das Q1-Rechteck exakt dar. Sobald die Wahrheit krumm ist (eine beflammte Ecke, ein runder Rand), trifft die FEM nur näherungsweise, und dann zeigt sich, ob Rechteck oder Dreieck näher an der Wirklichkeit liegt. Genau das ist die Frage von Kapitel 12.

11.9. Die Namensschilder

Wie in 1 haben wir zuerst die Sache gebaut und die Fachwörter weggelassen. Jetzt die Schilder.

Die Fachwörter, jetzt nachgereicht

- **Bilineare Interpolation**: das „zweimal linear" — erst in x, dann in y. *Bi* (lateinisch „zwei") plus *linear*: in jeder der beiden Richtungen einzeln geradlinig, im Inneren durch das gemischte $x\,y$-Glied gekrümmt.
- **Q1-Rechteck** (bilineares Viereckselement): das Rechteck mit einem Knoten je Ecke und bilinearer Interpolation. Das „Q" steht für *quadrilateral* (Viereck), die „1" für Grad 1 in jeder Richtung. Es ist das Gegenstück zum **P1-Dreieck** aus 1.
- **Sattelfläche** (hyperbolisches Paraboloid): die gekrümmte Form der bilinearen Ansatzfläche — in der einen Diagonale durchhängend, in der anderen aufgewölbt, wie ein Reitsattel oder Kartoffelchip. Sie entsteht allein aus dem $x\,y$-Glied.
- **Isoparametrisch**: der Trick der Profi-Programme, schiefe Vierecke mit *denselben* Ansatzfunktionen zu berechnen, mit denen sie das Feld darstellen (nur als Ausblick — dieses Buch braucht ihn nicht).

Alle diese Wörter stehen im Glossar.

Merkkasten

- **Bilinear = zweimal linear**: erst auf Unter- und Oberkante in x interpolieren, dann dazwischen in y. Drei Strecken-Interpolationen, kein neues Werkzeug.
- Die vier Ansatzfunktionen sind **Produkte zweier Hütchen** (eins in x, eins in y). Auf jeder Kante linear (darum **kantenkompatibel**, sogar mit Dreiecken), im Inneren gekrümmt (der **Sattel**, vom $x\,y$-Glied).
- Ein Dreieck-Paar hat eine **Diagonalen-Willkür** (bis 280 Grad Unterschied im Zentrum); das Rechteck kennt sie nicht.
- Die **Elementmatrix** ist 4×4 und zerfällt in Produkte von 1D-Bausteinen: Steigung mal Steigung in der einen Richtung, Hütchen-Überlapp in der anderen. Zeilensummen null, wie immer.
- Der Rechteck-Träger liefert dieselben **160 °C** wie der Dreieck-Träger, aber mit **9 statt 5** Kopplungen pro innerem Knoten (der 9-Punkt-Stern, weil die Eck-Diagonalen mitkoppeln).

Roter Faden

Zurück: Die bilineare Interpolation ist dreimal das **Hütchen** aus Kapitel 7; die Elementmatrix baut auf den 1D-Bausteinen „Steigung mal Steigung" (Kapitel 7) und „Hütchen-Überlapp" (die Massenmatrix aus Kapitel 9) auf, beide im Produkt. Die **Kantenkompatibilität** ist wortgleich die aus 1, und die 160 °C-Lösung ist dieselbe wie dort — nur das System darunter hat sich geändert. **Assemblieren, Streich-Rezept (Kapitel 8) und Gauß-Seidel (2)** liefen unverändert weiter.

Vor: Kapitel 12 fragt, welche Familie — Dreieck oder Rechteck — die Wirklichkeit *genauer* trifft, wenn die Lösung nicht mehr zufällig gerade ist (Netzkonvergenz), und wie man das ehrlich misst. Und im Finale 3 sitzt der **Elementtyp-Umschalter** aus dem Rechteck-Labor: Dann entscheidest du für ein echtes Problem, ob du mit Dreiecken oder Rechtecken pflasterst.

Übungen

Ü 11.1 (Verstehen). Ein Rechteck hat die Eckwerte unten links 100 °C, unten rechts 200 °C, oben rechts 400 °C, oben links 300 °C. Berechne von Hand mit dem Zwei-Schritt-Rezept die Temperatur am Punkt bei x-Anteil $\frac{1}{2}$ und y-Anteil $\frac{1}{4}$. Kontrolliere im Rechteck-Labor (Teil 1).

> **Musterlösung zu Ü 11.1**
>
> Schritt 1, in x bei Anteil $\frac{1}{2}$: Unterkante $100 + \frac{1}{2}(200 - 100) = 150$ °C, Oberkante $300 + \frac{1}{2}(400 - 300) = 350$ °C. Schritt 2, in y bei Anteil $\frac{1}{4}$: $150 + \frac{1}{4}(350 - 150) = 150 + 50 = 200$ °C. Kontrolle im Labor mit `ECKWERTE = [100, 200, 400, 300]`, `PUNKT_X = 0.5`, `PUNKT_Y = 0.25` → 200 °C.

Ü 11.2 (Verändern). Suche die vier Eckwerte (nur Werte 20 °C oder 300 °C erlaubt), die den Unterschied zwischen Rechteck und Dreiecken im Zentrum **maximal** machen. Bei welcher Belegung ist er am größten, und wie groß wird er? Das Skript `loesungen/kap11_ue2.py` probiert es durch.

> **Musterlösung zu Ü 11.2**
>
> Der Sattel ist am stärksten, wenn diagonal gegenüberliegende Ecken **gleich** sind und die beiden Diagonalen so verschieden wie möglich — die Belegung **über Kreuz**: unten links 20, unten rechts 300, oben

rechts 20, oben links 300 (oder umgekehrt). Dann liefert Diagonale A (die beiden 20-Ecken verbindend) im Zentrum 20 °C, Diagonale B 300 °C, das Rechteck 160 °C — eine Spanne von **280 Grad** zwischen den Dreiecksvarianten. Der Grund: Die Spanne ist genau $|e|/2$ mit dem Verwindungsterm e = unten links − unten rechts + oben rechts − oben links, und der wird über Kreuz betragsmäßig maximal ($|-560|$). Das Skript bestätigt die Tabelle.

Ü 11.3 (Übertragen). Skizziere von Hand ein gemischtes Netz für einen Träger mit **schrägem** rechten Ende: Rechtecke im regelmäßigen Inneren, Dreiecke am schrägen Rand. Begründe, warum an der Grenze zwischen einem Rechteck und einem angesetzten Dreieck kein Sprung entsteht. Das Skript `loesungen/kap11_ue3.py` prüft die Kantenkompatibilität numerisch nach.

i Musterlösung zu Ü 11.3

Skizze: zwei bis drei achsparallele Rechtecke bilden den geraden Teil des Trägers; das schräge Ende wird mit Dreiecken aufgefüllt, deren Grundkante genau eine Rechteckkante ist und deren Spitzen zum schrägen Rand zeigen. **Warum es passt:** Auf der gemeinsamen Kante ist der Verlauf für **beide** Elementtypen linear und hängt nur von den zwei geteilten Knotenwerten ab — beim Rechteck, weil dort x (oder y) konstant ist und der $x\,y$-Term wegfällt; beim Dreieck, weil die dritte Ecke (die Spitze) auf der Kante den Anteil null hat. Beide interpolieren also dieselbe Gerade zwischen denselben zwei Werten — kein Sprung, keine Überlappung. Das Skript `kap11_ue3.py` legt ein Rechteck und ein Nachbardreieck an eine gemeinsame Kante und zeigt, dass die Kantenwerte auf jede Nachkommastelle übereinstimmen (größte Differenz: null).

12. Traue keinem bunten Bild

Zwei Ingenieurbüros rechnen denselben Träger: links steckt sein Ende im kühlen 20-°C-Auflager, rechts heizt der Brandherd auf 300 °C. Beide liefern einen sauberen, professionell aussehenden Bericht ab — ein Farbbild des Temperaturverlaufs, glatt von Blau nach Rot, dazu die Kennzahl, auf die es dem Prüfstatiker ankommt: die Wärmeleistung, die der Träger vom heißen zum kühlen Ende durchreicht. Büro A schreibt **140 W**. Büro B schreibt **280 W** — doppelt so viel. Beide Bilder sehen gleich vertrauenswürdig aus.

Eines der beiden ist um den Faktor zwei falsch. Vom bloßen Ansehen: nicht zu entscheiden. Und — das ist der eigentliche Schrecken — die beiden Farbbilder des Temperaturverlaufs sind sogar *identisch*. Der Fehler versteckt sich in einer Zahl, die das schöne Bild gar nicht zeigt.

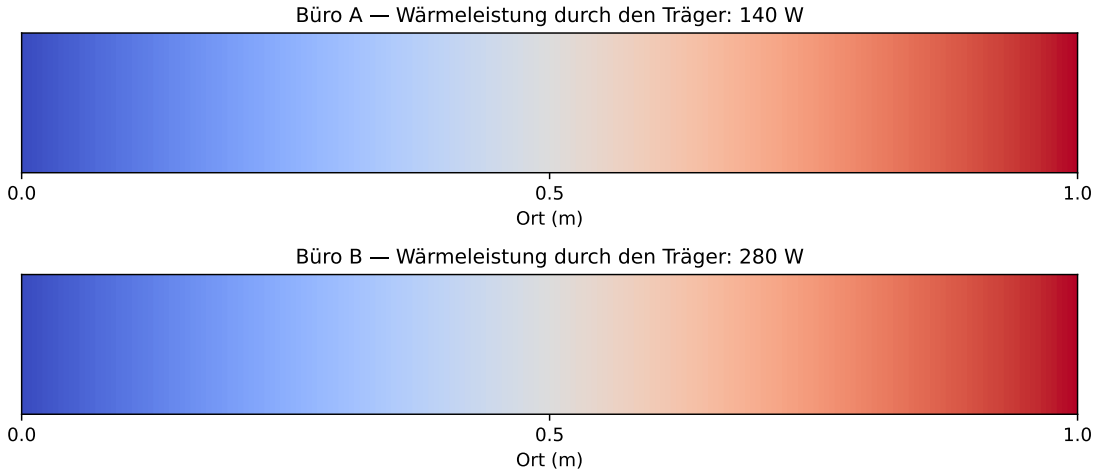

Abbildung 12.1.: Das Duell der zwei Berichte. Beide zeigen dasselbe Temperaturbild des Trägers — dieselbe Farbskala, dieselbe Gerade von 20 °C (blau, links) nach 300 °C (rot, rechts). Nur die Kennzahl darunter unterscheidet sich: 140 W gegen 280 W. Welcher Bericht ist falsch? Am Bild ist es nicht zu sehen — das ist die Lektion des Kapitels. Die Auflösung steht am Kapitelende.

Was man hier sieht: zwei Bilder, die man nicht auseinanderhalten kann,

und zwei Zahlen, die sich um den Faktor zwei unterscheiden. Ein FEM-Programm rechnet *immer* ein hübsches Bild aus — auch wenn es sich verrechnet hat. Der Merksatz dieses Kapitels lautet darum:

> **Bunt ist kein Beweis.**

Dieses Kapitel gibt dir die Werkzeuge, in fünf Minuten zu entscheiden, welches der beiden Büros recht hat — und allgemeiner: jeder Simulation die drei Fragen zu stellen, die ein Profi ihr stellt, bevor er ihr glaubt.

Lernziele

Nach diesem Kapitel kannst du …

1. … **Verifikation** („löse ich die Gleichungen richtig¿‘) und **Validierung** („löse ich die richtigen Gleichungen?”) auseinanderhalten,
2. … drei analytische **Prüffälle** des Trägers einsetzen — die Gerade als **Nulltest**, die Parabel mit Wärmequelle und die transiente Abkühlkurve — und aus einer Abweichung auf ihre Ursache schließen,
3. … eine **Konvergenzstudie** durchführen: den Fehler bei h, $h/2$, $h/4$ messen und am **Viertelungs-Rhythmus** erkennen, dass die Rechnung sauber konvergiert,
4. … die **Energiebilanz** als Alltagstest nutzen (rein − raus = gespeichert, auf 1 % genau?),
5. … die fünf häufigsten Anfängerfehler an ihren Symptomen erkennen und mit der **Simulations-Checkliste** systematisch ausschließen.

⚠ Naheliegende Vermutung

Vermutung: *„Je feiner das Netz, desto richtiger — irgendwann stimmt es dann von selbst.”*
Warum sie naheliegt: In Kapitel 1 haben wir es ja selbst gesehen: mehr Knoten bilden die Wirklichkeit genauer ab. Und tatsächlich schrumpft der Fehler mit jedem Verfeinern.
Was stattdessen stimmt: Verfeinern heilt **nur den einen** Fehler, der vom groben Netz kommt — den **Diskretisierungsfehler**. Eine vergessene Randbedingung, ein Einheitenfehler, ein vertauschtes Vorzeichen oder ein Programmierfehler bleiben auf jedem Netz falsch. Schlimmer noch: Auf einem feinen Netz sieht das falsche Ergebnis *glatter und vertrauenswürdiger* aus als auf einem groben. Ein feines Netz macht ein falsches Bild nicht richtig, sondern nur schöner. Genau deshalb braucht es Prüffälle mit *bekannter* Antwort.

12.1. Die drei Fragen — und zwei Wörter, die man nicht verwechseln darf

Bevor ein Profi einer Simulation glaubt, stellt er ihr drei Fragen:

1. **Stimmt das Programm?** Rechnet es die Gleichungen, die es zu rechnen vorgibt, überhaupt richtig — ohne Programmier- oder Einheitenfehler?
2. **Reicht das Netz?** Ist der Diskretisierungsfehler klein genug für den Zweck?
3. **Passen die Randbedingungen zur Wirklichkeit?** Bildet das Modell die echte Situation ab?

Die ersten beiden Fragen prüft man **rechnerisch**, ganz ohne Labor — das ist die **Verifikation**, das Thema dieses Kapitels. Die dritte führt irgendwann ins Labor, zu echten Messungen — das ist die **Validierung**.

Verifikation und Validierung — zwei Fragen, nicht eine

Zwei Wörter, die im Alltag oft durcheinandergehen und die man als Simulationsingenieur sauber trennt (Roache 1998):

- **Verifikation:** *„Löse ich die Gleichungen richtig?“* — Rechnet das Programm die *gewählte* Mathematik korrekt? Werkzeuge: analytische Prüffälle, Netzkonvergenz, Bilanzen. Braucht **kein** Labor, nur bekannte Vergleichslösungen. **Das kann dieses Buch vollständig.**
- **Validierung:** *„Löse ich die richtigen Gleichungen?“* — Bildet das *Modell* die Wirklichkeit ab? Werkzeug: der Vergleich mit **Messungen** an echten Bauteilen. Deckt auf, was keine noch so saubere Rechnung findet — z. B. dass wir die Wärmeabgabe an die Luft weggelassen haben, oder dass Stahl bei 300 °C anders leitet als bei 20 °C.

Merkregel: **Erst verifizieren, dann validieren.** Ein Programm mit einem Rechenfehler gegen eine Messung zu halten, verwirrt nur — man weiß dann nicht, ob das Modell oder der Code danebenliegt. Dieses Kapitel verifiziert; die Validierung bleibt sein ehrlicher Ausblick (siehe *Das Kleingedruckte*).

12.2. Prüffall 1 — die Gerade: der Nulltest

Der erste Prüffall ist der schärfste, und wir haben ihn längst in der Hand. In Kapitel 6 haben wir bewiesen: Ohne Wärmequelle ist das eingeschwungene Temperaturprofil des homogenen Trägers **exakt eine Gerade** — von den 20 °C links geradlinig auf die 300 °C rechts. Die Formel dazu kennt jeder Schüler:

$$T(x) \;=\; 20\,^{\circ}\mathrm{C} \;+\; \frac{x}{L}\,(300-20)\,\mathrm{K}, \qquad T(0{,}5\,\mathrm{m}) = 20 + \tfrac{1}{2}\cdot 280 = 160\,^{\circ}\mathrm{C}.$$

Das Besondere: Die linearen Elemente dieses Buches stellen eine Gerade **exakt** dar (ihre Hütchen interpolieren geradlinig, Kapitel 7). Zwischen der exakten Lösung und dem, was die FEM darstellen *kann*, klafft hier also **keine Lücke** — kein Diskretisierungsfehler, auf keinem Netz. Die FEM *muss* die Gerade an jedem Knoten und an jedem Punkt dazwischen punktgenau treffen. Das macht die Gerade zum **Nulltest**:

> **Der Nulltest — das schärfste Werkzeug**
>
> Ein Nulltest ist ein Prüffall, dessen exakte Lösung die FEM **ohne jeden Diskretisierungsfehler** treffen muss. Läuft er durch: schön. Zeigt er auch nur ein hundertstel Grad Abweichung, ist das **garantiert** ein Programm- oder Randbedingungsfehler — es gibt keine gutmütige Ausrede „das Netz war zu grob". Deshalb ist der Nulltest das erste, was man rechnet, wenn ein neues Programm läuft.

Wir lassen die FEM aus Kapitel 7 den quellfreien Träger auf drei Netzen rechnen und vergleichen jeden Knoten mit der Geraden:

```
def gerade(ort, links_grad, rechts_grad, laenge):
    """Die exakte Gerade zwischen den beiden Randtemperaturen."""
    return links_grad + ort / laenge * (rechts_grad - links_grad)


def groesste_abweichung(knotentemperaturen, links_grad, rechts_grad, laenge):
    """Die groesste Abweichung der FEM-Knoten von der exakten Geraden (Grad)."""
    knotenzahl = len(knotentemperaturen)
    elementlaenge = laenge / (knotenzahl - 1)
    schlimmste = 0.0
    for knoten in range(knotenzahl):
        ort = knoten * elementlaenge
```

```
        abweichung = knotentemperaturen[knoten] - gerade(
            ort, links_grad, rechts_grad, laenge)
        if abweichung < 0.0:
            abweichung = -abweichung
        if abweichung > schlimmste:
            schlimmste = abweichung
    return schlimmste


# Die FEM-Knotenwerte des quellfreien Traegers (aus dem Kapitel-Programm) fuer
# 5, 9 und 17 Knoten -- hier zur Anschauung schon eingesetzt:
fem_5 = [20.0, 90.0, 160.0, 230.0, 300.0]
print("Nulltest -- FEM gegen die exakte Gerade:")
print("  5 Knoten: groesste Abweichung =",
      "%.2e Grad" % groesste_abweichung(fem_5, 20.0, 300.0, 1.0))
```

```
Nulltest -- FEM gegen die exakte Gerade:
  5 Knoten: groesste Abweichung = 0.00e+00 Grad
```

Interpretation: null (bis auf die letzte Stelle, die der Computer beim Rechnen mit Kommazahlen ohnehin verwackelt). Auf 9 und 17 Knoten sieht es genauso aus — der Nulltest ist auf *jedem* Netz exakt. Büro A und Büro B aus dem Aufhänger würden diesen Test übrigens **beide** bestehen; merke dir das für die Auflösung.

12.3. Prüffall 2 — die Parabel: der erste echte Diskretisierungsfehler

Jetzt bauen wir einen Fehler ein, den die FEM *nicht* mehr wegzaubern kann. Wir stecken den Heizdraht aus Kapitel 6 in den Träger: eine gleichmäßig über die Länge verteilte Wärmequelle, insgesamt $Q = 160\,\mathrm{W}$ (das ist die Aufgabe aus Übung 7.3). Die Bilanz geht dann nicht mehr mit einer Geraden auf; die exakte Lösung ist eine **Parabel** über der Geraden:

$$T(x) \;=\; \underbrace{20 + \tfrac{x}{L}\cdot 280}_{\text{Gerade}} \;+\; \frac{q}{2\lambda A}\,x\,(L-x), \qquad q = \frac{Q}{L} = 160\ \tfrac{\mathrm{W}}{\mathrm{m}}.$$

Der Vorfaktor mit echten Zahlen — Baustahl $\lambda = 50\ \mathrm{W/(m\,K)}$, Querschnitt $A = 0{,}01\ \mathrm{m}^2$:

$$\frac{q}{2\lambda A} = \frac{160}{2 \cdot 50 \cdot 0{,}01} = 160\ \tfrac{^\circ\mathrm{C}}{\mathrm{m}^2}, \qquad \text{Höcker in der Mitte} = 160{\cdot}0{,}5{\cdot}0{,}5 = 40\,^\circ\mathrm{C}.$$

In der Mitte wölbt sich die Parabel also genau **40 °C** über die Gerade — auf 200 °C statt 160 °C. Das ist dieselbe Parabel, die der Kasten in Kapitel 6 angekündigt und die Übung 7.3 ausgerechnet hat. Jetzt ist sie unsere geprüfte Referenz.

Und nun kommt das Verblüffende. Wir lassen die FEM die Parabel auf 5, 9 und 17 Knoten rechnen und messen den Fehler auf **zwei** Arten: einmal nur *an den Knoten*, einmal *zwischen* den Knoten (dort, wo der FEM-Streckenzug die gebogene Kurve geradlinig abkürzt).

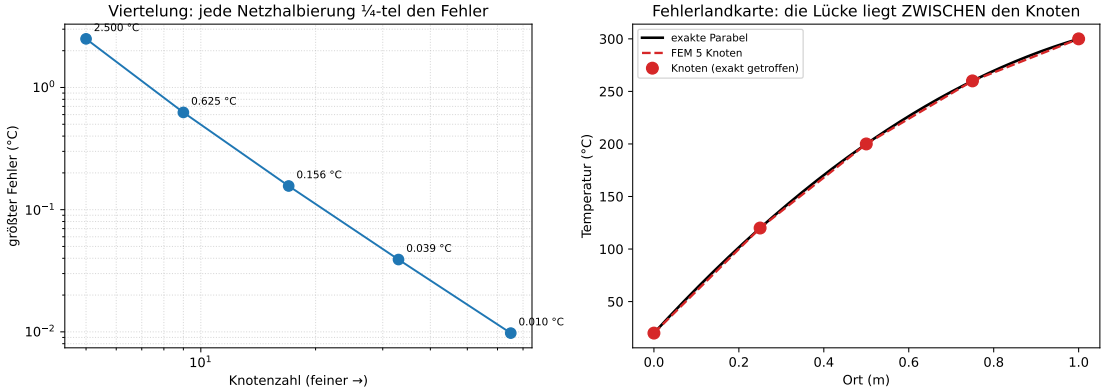

Abbildung 12.2.: Links der Konvergenzplot: der größte Zwischen-Knoten-Fehler der Parabel über der Knotenzahl (beide Achsen logarithmisch — gleiche Faktoren werden zu gleichen Abständen). Von Netz zu Netz viertelt sich der Fehler: 2,50 → 0,625 → 0,156 → 0,039 °C. Rechts die Fehlerlandkarte für 5 Knoten: Die FEM (rot gestrichelt) trifft jeden Knoten (rote Punkte) exakt auf der schwarzen Parabel, schneidet aber zwischen den Knoten die Kurve ab — die rote Fläche ist der ganze Fehler.

Interpretation: Zwei Überraschungen. Erstens: **An den Knoten ist die FEM sogar exakt** — die roten Punkte liegen punktgenau auf der schwarzen Parabel (der Knotenfehler ist so klein wie beim Nulltest). Das ist eine Besonderheit der 1D-Rechnung. Zweitens, und das ist die eigentliche Lektion: Der Fehler steckt ganz **zwischen** den Knoten, wo der gerade FEM-Streckenzug die Kurve der Parabel abkürzt. Genau dort greift ein buntes Bild, wenn es

zwischen den Knoten interpoliert. Und dieser Zwischen-Knoten-Fehler **viertelt sich** bei jeder Netzhalbierung: $2{,}50 \to 0{,}625 \to 0{,}156 \to 0{,}039\,°\mathrm{C}$.

> **Der Viertelungs-Rhythmus (2. Ordnung), ohne Angst vor dem Wort**
>
> Halbiert man die Maschenweite h, so viertelt sich der Fehler. Warum *viertelt*, nicht *halbiert*? Weil der Fehler eines geraden Stücks über einer Kurve mit dem **Quadrat** der Stücklänge wächst (h^2) — halb so lang heißt $(\frac{1}{2})^2 = \frac{1}{4}$ des Fehlers. Das nennt man **zweite Ordnung** oder **Konvergenzordnung 2**. Der Viertelungs-Rhythmus ist der praktische Test dafür: Rechne dieselbe Aufgabe auf h, $h/2$, $h/4$ und schau, ob sich der Fehler brav viertelt. Tut er das *nicht* (viertelt zu wenig oder gar nicht), stimmt etwas nicht — dann ist der Fehler kein Netzfehler, sondern ein echter Bug. Der Log-Plot links macht den Rhythmus sichtbar: Weil beide Achsen gleiche Faktoren zu gleichen Abständen stauchen, liegen die Punkte auf einer **Geraden** — je steiler, desto höher die Ordnung.

12.4. Dasselbe in der Fläche — und eine Falle beim Testen

In 1 und Kapitel 11 haben wir zwei Elementfamilien gebaut, Dreiecke und Rechtecke. Welche ist genauer? Der naheliegende Plan: dieselbe Parabel in 2D rechnen und die Fehler vergleichen. Wir legen den Träger als Fläche an (oben und unten isoliert, also nur in x-Richtung ein Verlauf) und lassen beide Familien auf immer feineren Netzen los.

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap12"))
from kap12_pruefstation import loese_2d_parabel

print("2D-Parabel – Knotenfehler von Dreiecken und Rechtecken:")
print("  nx | Dreieck      | Rechteck")
print("  ---+--------------+-------------")
for nx in (4, 8, 16):
    _, _, fd = loese_2d_parabel(nx, 1, "dreieck")
    _, _, fr = loese_2d_parabel(nx, 1, "rechteck")
    print("  %2d | %.2e   | %.2e" % (nx, fd, fr))
```

```
2D-Parabel – Knotenfehler von Dreiecken und Rechtecken:
  nx | Dreieck      | Rechteck
  ---+--------------+-------------
```

```
  4 | 5.68e-14   | 1.14e-13
  8 | 3.98e-13   | 4.83e-13
 16 | 9.09e-13   | 4.26e-13
```

Interpretation: null und wieder null — **beide** Familien treffen die Parabel an jedem Knoten exakt, auf jedem Netz. Heißt das, Dreiecke und Rechtecke sind gleich gut? **Nein** — es heißt, dass dieser Test sie gar nicht auseinanderhalten *kann*. Die Parabel krümmt sich nur in einer Richtung; in dieser einen Richtung sind beide Elemente geradlinig und darum an den Knoten exakt. Ein Prüffall testet nur, was er auch beansprucht.

Vorhersage-Punkt

Bevor du weiterliest: Wir bauen jetzt einen Prüffall, der in *beide* Richtungen krümmt — einen glatten Wärmehügel. Jetzt *muss* jedes Element die Krümmung mit geraden bzw. sattelförmigen Flächen annähern; ein echter Fehler entsteht. Was glaubst du: Viertelt er sich auch hier (zweite Ordnung)? Und gewinnt das Dreieck oder das Rechteck? Leg dich fest.

Der neue Prüffall ist ein quadratischer Fleck Blech, an allen vier Rändern auf 20 °C gehalten, mit einer Quelle, die in der Mitte am stärksten heizt. Seine exakte Lösung ist eine glatte Kuppel, $20 + 40\sin(\pi x)\sin(\pi y)$ — ein „Parabel-Hügel", der in x *und* y krümmt.

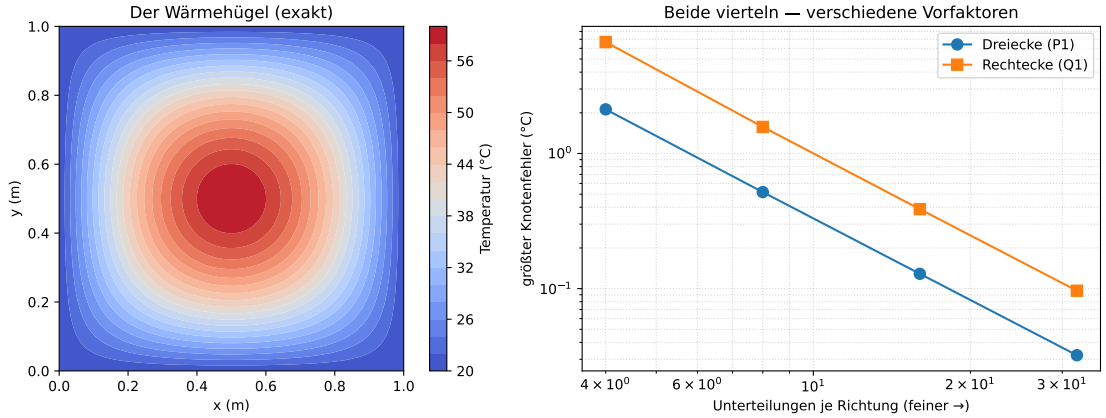

Abbildung 12.3.: Links der Wärmehügel als Farbbild (exakte Lösung, Peak 60 °C in der Mitte). Rechts die Konvergenz beider Familien: Dreiecke (blau) und Rechtecke (orange) vierteln beide ihren Fehler bei jeder Verfeinerung (zweite Ordnung), aber mit verschiedenem Vorfaktor — hier liegen die Dreiecke etwas günstiger. Ein echter 2D-Test trennt die Familien, die entartete Parabel konnte es nicht.

Interpretation: Jetzt entsteht ein echter Fehler, und **beide** Familien zähmen ihn im Viertel-Takt (zweite Ordnung — die Log-Geraden sind gleich steil). Sie unterscheiden sich nur im **Vorfaktor**: Auf demselben Netz sind die Dreiecke hier rund dreimal genauer als die Rechtecke (2,1 gegen 6,7 °C beim gröbsten Netz). Welche Familie „gewinnt", hängt von der Form der Lösung ab — Kapitel 11 hat schon angedeutet, dass es keinen generellen Sieger gibt. Wichtiger als der Sieger ist die Lehre über das *Testen*: Ein Prüffall, den zwei verschiedene Verfahren *identisch* bestehen, vergleicht sie nicht — er ist zu einfach. Man muss den Test so wählen, dass er das beansprucht, was man wissen will.

Die Verfeinerung lässt sich auch als Film ansehen: Netz und Fehler schmelzen gemeinsam im Viertel-Takt.

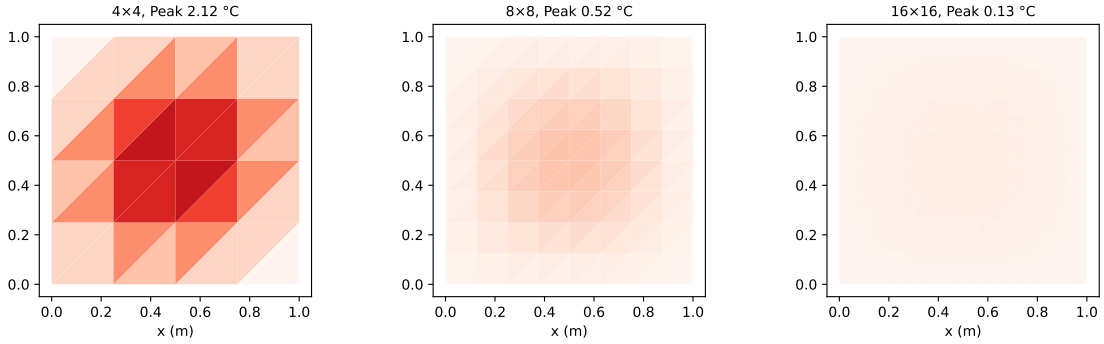

Abbildung 12.4.: Netzverfeinerung des Wärmehügels als Bildreihe (in der HTML-Fassung eine Animation): die Fehlerbilder für 4×4, 8×8 und 16×16 Dreiecke bei fester Farbskala. Der Höchstfehler viertelt sich von Bild zu Bild (2,12 → 0,52 → 0,13 °C).

12.5. Prüffall 3 — die Abkühlkurve gegen eine exakte Reihe

Die ersten beiden Prüffälle waren stationär. Jetzt bekommt die Zeit ihre Prüfung. Der Träger startet kalt (überall 20 °C), das rechte Ende springt auf die Temperatur des Brandherds, und wir sehen der Wärme beim Hereinwandern zu (Kapitel 9). Gibt es dafür eine *exakte* Vergleichslösung? Ja — und wir sind ihr längst begegnet.

Woher kam die Fühler-Kurve aus Kapitel 5?

In Kapitel 5 hatten wir eine glatte Funktion `temperatur(ort)` als „Messfühler" benutzt — die 30-Minuten-Momentaufnahme des Trägers, an der wir Steigung und Fläche geübt haben. Damals stand dort: *„Woher die Formel kommt, entwickeln die Kapitel 6 und 12."* Hier ist die Einlösung. Jene Kurve war die **exakte** Lösung der Wärmeleitungsgleichung — die stationäre Gerade plus eine abklingende Summe von Sinuswellen (eine **Fourier-Reihe**):

$$T(x,t) \;=\; \underbrace{20 + \tfrac{x}{L}\,\Delta T}_{\text{Endzustand}} \;+\; \sum_{n=1}^{\infty} \frac{2\,\Delta T\,(-1)^n}{n\pi}\,\sin\!\left(\frac{n\pi x}{L}\right) e^{-a\,(n\pi/L)^2\,t}, \qquad a = \frac{\lambda}{\rho c}.$$

Man muss diese Formel **nicht herleiten** — sie ist ein geprüftes Werkzeug, kein Lehrstoff dieses Buches. Zwei Dinge genügen zum Verständnis: Jede Sinuswelle klingt mit der Zeit ab (der Faktor $e^{-\cdots t}$), und die

langsamste Welle ($n = 1$) bestimmt, wie lange es dauert — ihre Abklingzeit ist $\tau_1 = \tau/\pi^2 \approx 7320\,\text{s}$, mit der Zeitkonstante $\tau = L^2\rho c/\lambda \approx 72\,220\,\text{s}$ (rund 20 Stunden) aus Kapitel 9. Genau diese Reihe ist ab jetzt unsere Referenz. Wir prüfen, ob die transiente FEM gegen sie **konvergiert**.

Wir lassen die implizite FEM aus Kapitel 9 bis $t = 1800\,\text{s}$ (30 Minuten) rechnen und vergleichen mit der Reihe — einmal bei Netzverfeinerung, einmal bei Zeitschritt-Verfeinerung:

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap12"))
from kap12_pruefstation import transienter_fehler

print("Transient gegen die Reihe (t = 1800 s, rechtes Ende 500 °C):")
print()
print("Netz verfeinern (dt = 1 s fest):")
for kn in (5, 9, 17):
    print("  %2d Knoten: max Fehler = %.3f °C" % (kn, transienter_fehler(kn, 1.0, 1800.0, 500.0)))
print()
print("Zeitschritt verfeinern (33 Knoten fest):")
vorher = None
for dt in (200.0, 100.0, 50.0, 25.0):
    f = transienter_fehler(33, dt, 1800.0, 500.0)
    verh = "" if vorher is None else "  (%.2f-fach kleiner)" % (vorher / f)
    print("  dt = %6.1f s: max Fehler = %5.3f °C%s" % (dt, f, verh))
    vorher = f
```

```
Transient gegen die Reihe (t = 1800 s, rechtes Ende 500 °C):

Netz verfeinern (dt = 1 s fest):
   5 Knoten: max Fehler = 11.328 °C
   9 Knoten: max Fehler = 3.865 °C
  17 Knoten: max Fehler = 1.022 °C

Zeitschritt verfeinern (33 Knoten fest):
  dt =  200.0 s: max Fehler = 7.414 °C
  dt =  100.0 s: max Fehler = 3.751 °C  (1.98-fach kleiner)
  dt =   50.0 s: max Fehler = 1.917 °C  (1.96-fach kleiner)
  dt =   25.0 s: max Fehler = 1.001 °C  (1.91-fach kleiner)
```

Interpretation: Die FEM konvergiert sauber gegen die exakte Reihe — die „Fühler-Kurve" aus Kapitel 5 ist also wirklich die richtige Antwort, und die

FEM trifft sie im Grenzfall. Aber die beiden Verfeinerungen verhalten sich **unterschiedlich**: Das Netz strebt die Viertelung an (Raum, zweite Ordnung) — und die Tabelle ist dabei ehrlicher als jede Faustregel: erst 2,9-fach, dann 3,8-fach kleiner. Auf groben Netzen ist der Viertelungs-Rhythmus noch nicht ganz eingerastet, mit jeder Verfeinerung rückt er näher an die 4 heran (genau das meint „asymptotisch"). Der Zeitschritt aber **halbiert** den Fehler nur (jede Halbierung von Δt halbiert ihn — die zweite Tabelle zeigt es fast aufs Komma).

Zeit ist gröber als Raum

Der implizite Euler ist nur **erster Ordnung**: Halbierst du Δt, halbierst du den Zeitfehler — nicht mehr. Der Raum ist bei linearen Elementen zweiter Ordnung (viertelt), die Zeit nur erster (halbiert). Wer also die Zeit genauso genau haben will wie den Raum, muss viel kleinere Zeitschritte machen, als das Netz an Feinheit nahelegt. Es gibt bessere Zeitschrittverfahren — das **Crank-Nicolson**-Verfahren etwa mittelt alten und neuen Zeitpunkt und ist zweiter Ordnung — aber das ist ein anderes Kapitel. Für uns zählt die ehrliche Faustregel: **Zeit ist gröber als Raum**, also Zeitschritt nicht vergessen, wenn man verfeinert (Übung 12.2).

12.6. Die Energiebilanz — der Wachhund, der immer mitläuft

Prüffälle mit bekannter Lösung sind Gold, aber es gibt sie nicht für jede echte Aufgabe. Ein Test läuft dagegen *immer* mit, ganz ohne Vergleichslösung: die **Energiebilanz**. Die Physik erlaubt keine Wärme aus dem Nichts. Über jeden Zeitraum muss darum aufgehen:

$$\underbrace{\text{hereingeflossen}}_{\text{heißes Ende}} = \underbrace{\text{gespeichert}}_{\text{Anstieg}\times\rho c A} + \underbrace{\text{hinausgeflossen}}_{\text{kaltes Ende}} \,(+\,\text{erzeugt}).$$

Das rechnet man im Programm einfach mit: an jedem Rand den Wärmestrom mal Zeitschritt aufsummieren, am Ende die gespeicherte Wärme aus den Temperaturanstiegen bilden. Geht die Bilanz nicht auf 1 % genau auf, ist etwas faul — eine Wärmequelle, die man vergessen hat, ein Rand, über den heimlich Wärme entweicht, ein Vorzeichen verkehrt herum.

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap12"))
from kap12_pruefstation import energiebilanz

herein, rechte, rel = energiebilanz(17, 30.0, 3600.0, 300.0)
print("Energiebilanz des transienten Traegers (17 Knoten, dt = 30 s, 1 Stunde):")
print("  hereingeflossen      = %6.0f kJ" % (herein / 1000.0))
print("  gespeichert + hinaus = %6.0f kJ" % (rechte / 1000.0))
print("  relativer Bilanzfehler = %.3f %%   (Toleranz: 1 %%)" % (rel * 100.0))
```

```
Energiebilanz des transienten Traegers (17 Knoten, dt = 30 s, 1 Stunde):
  hereingeflossen      =   2241 kJ
  gespeichert + hinaus =   2241 kJ
  relativer Bilanzfehler = 0.000 %   (Toleranz: 1 %)
```

Interpretation: Die Bilanz schließt praktisch perfekt — jedes Joule, das rechts hereinkommt, findet sich als gespeicherte oder links abgeflossene Wärme wieder. Das ist kein Beweis, dass *alles* stimmt (die Bilanz merkt manche Fehler nicht, siehe Aufhänger-Auflösung), aber ein **Ausschlusskriterium**: Schließt sie nicht, hast du garantiert einen Fehler. Als Wachhund gehört sie in jedes ernsthafte Programm.

12.7. Die Fehler-Sprechstunde: die fünf Klassiker an ihren Symptomen

Die häufigsten Anfängerfehler hinterlassen typische Symptombilder. Wer sie kennt, diagnostiziert in Sekunden. Hier die Galerie der fünf Klassiker — je mit Symptom und Diagnose-Rezept. Klapp jede Diagnose erst auf, wenn du selbst geraten hast.

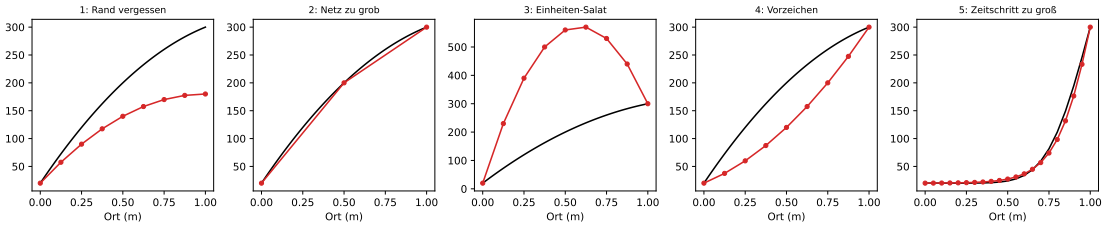

Abbildung 12.5.: Die Symptom-Galerie der fünf Klassiker, jeweils als Temperaturprofil des Trägers (schwarz die korrekte Parabel-Lösung, rot der fehlerhafte Lauf). Von links: vergessene Randbedingung (rechtes Ende läuft weg), zu grobes Netz (Knoten stimmen, Form zu kantig), Einheiten-Salat (Höcker zehnfach), Vorzeichenfehler (Kurve hängt durch statt aufzuwölben), zu großer Zeitschritt (Front verschmiert, transient). Jedes Bild ist bunt und plausibel — und trotzdem falsch.

Klassiker 1 — die vergessene Randbedingung

Symptom: Ein Rand zeigt nicht den Wert, den er zeigen soll — das rechte Ende „läuft weg" statt bei 300 °C zu stehen; oft läuft die halbe Rechnung auf einen unsinnigen Wert. **Ursache:** Über einen Rand wurde *nichts* gesagt — und in der FEM heißt „nichts sagen" **isoliert** (Fluss null), nicht „offen" (Kapitel 8). **Diagnose-Rezept:** Punkt 1 der Checkliste — jede Randkante durchgehen: Hat sie eine *gewollte* Bedingung? Der Nullfeld deckt das sofort auf, weil die Gerade dann nicht mehr stimmt.

Klassiker 2 — das zu grobe Netz

Symptom: der heimtückischste. In 1D sehen die **Knoten sogar richtig** aus (die Mitte zeigt korrekt 200 °C!) — nur zwischen den Knoten ist die Form zu kantig. Ein Farbbild, das nur die Knoten zeigt, verrät nichts. **Ursache:** Der Diskretisierungsfehler ist noch zu groß für den Zweck. **Diagnose-Rezept:** die **Konvergenzprobe**. Rechne dieselbe Aufgabe doppelt so fein. Ändert sich das Ergebnis noch merklich, war das Netz zu grob; viertelt sich die Änderung brav, kannst du abschätzen, wie fein es sein muss. Niemals einem *einzigen* Netz trauen.

Klassiker 3 — der Einheiten-Salat

Symptom: die Größenordnung stimmt nicht — der Höcker ist zehnmal zu hoch (560 °C statt 200 °C in der Mitte), oder die Temperaturen sind absurd. **Ursache:** eine Materialzahl in der falschen Einheit — λ in W/(cm·K) statt W/(m·K), Länge in mm statt m, Grad statt Kelvin, wo das Programm es anders erwartet. **Diagnose-Rezept:** Checkliste Punkt 5. Rechne *eine* Kennzahl von Hand grob nach (den Höcker $qL^2/(8\lambda A)$, den Wärmestrom $\lambda A\,\Delta T/L$) und vergleiche die Größenordnung. Einheiten konsequent mitschreiben — der teuerste Fehler der Raumfahrt war ein Einheiten-Salat.

Klassiker 4 — das falsche Vorzeichen

Symptom: die Kurve krümmt sich falsch herum — sie **hängt durch** statt sich aufzuwölben (Mitte 120 °C statt 200 °C). **Ursache:** ein vertauschtes Vorzeichen — der Heizdraht kühlt statt zu heizen, oder ein Wärmestrom fließt rein, wo er raus soll. **Diagnose-Rezept:** Checkliste Punkt 5 (Vorzeichen des Stroms) und die **Energiebilanz**. Frag dich: In welche Richtung *muss* die Wärme fließen? Eine Quelle wölbt die Kurve *über* die Gerade, eine Senke *unter* sie.

Klassiker 5 — der zu große Zeitschritt

Symptom (transient): die Wärmefront ist **verschmiert** — sie ist weiter ins Kalte gewandert und flacher, als sie sein dürfte. (Beim *expliziten* Euler wäre das Symptom drastischer: wachsende Zickzack-Ausschläge, die Explosion aus Kapitel 9.) **Ursache:** Δt zu groß — der implizite Euler bleibt zwar stabil, aber ungenau (erste Ordnung). **Diagnose-Rezept:** Zeitschritt halbieren und schauen, ob sich die Lösung noch merklich ändert — dieselbe Konvergenzprobe wie beim Netz, nur in der Zeit (Übung 12.2). *Zeit ist gröber als Raum.*

Diese Erfahrung gießen wir — wie schon die Randbedingungen in Kapitel 8 — in eine feste Prüfroutine. Sie erweitert die dortige Randbedingungs-Checkliste zur **Simulations-Checkliste des ganzen Buches**; sie kehrt im Finale (3) neben den Reglern wieder.

Die Simulations-Checkliste des Buches

Vor *jeder* Simulation — und vor jedem Glauben an ein buntes Bild:

1. **Randbedingungen** (die Checkliste aus Kapitel 8): Hat jeder Rand

eine *gewollte* Bedingung? Mindestens eine Temperatur (Dirichlet) vorgegeben, sonst schwimmt das System? Passt der Typ zur Physik (Thermostat → Dirichlet, Brandherd → Neumann)? Vorzeichen und Einheiten der Randwerte richtig?
2. **Nulltest:** Gibt es einen Fall mit exakt bekannter Lösung, den das Programm punktgenau treffen *muss* (die Gerade, freie Dehnung)? Läuft er durch?
3. **Konvergenz:** Auf zwei Netzen (h und $h/2$) gerechnet — viertelt sich die Änderung? Falls transient: auch Δt halbiert?
4. **Energiebilanz:** Geht rein $=$ gespeichert $+$ raus $(+$ erzeugt$)$ auf 1 % genau auf?
5. **Plausibilität von Hand:** Stimmen Größenordnung, Vorzeichen und Richtung *einer* nachgerechneten Kennzahl (Höcker, Wärmestrom, Zeitkonstante)?

Erst wenn alle fünf Haken sitzen, ist das bunte Bild ein Beweis — vorher ist es nur bunt.

12.8. Die interaktive Einheit: die Prüfstation

Jetzt bedienst du die Prüfstation selbst. Du wählst einen Prüffall, ein Netz und einen Elementtyp; die Station rechnet und zeigt dir Fehlertabelle und Bilanzprotokoll. Und sie hat einen **Sabotage-Modus**: Stell `SABOTAGE` auf 1 bis 5, dann baut sie *heimlich* einen Fehler ein — die vier stationären Klassiker (1 = Rand vergessen, 2 = Netz zu grob, 3 = Einheiten-Salat, 4 = Vorzeichen) und als fünften den gemeinen Aufhänger-Fehler. Du spielst Detektiv: Nutze die Checkliste, sag *vorher*, welcher Fehler es ist, und klapp dann die Auflösung auf. (Den fünften Galerie-Klassiker, den zu großen Zeitschritt, jagst du in Übung 12.2 — er zeigt sich nur im transienten Fall.)

Vorhersage-Punkt (das Detektivspiel)

Bevor du ausführst: Die Station startet mit `SABOTAGE = 5` — dem gemeinsten Fall. Die Temperaturgerade wird *völlig normal* aussehen, der Nulltest wird *bestehen*. Trotzdem stimmt etwas nicht. Was für ein Fehler versteckt sich so gut, dass nur *eine* der fünf Prüfungen der Checkliste ihn fängt — und welche Prüfung ist es? Leg dich fest, dann führe aus.

```
#| autorun: true
# ====== Die Prüfstation — dreh an den Reglern ======
```

```
PRUEFFALL = "parabel"   # "gerade" (Nulltest), "parabel" oder "abkuehlung"
KNOTENZAHL = 5          # 5, 9, 17, 33 ...
SABOTAGE = 5            # 0 = sauber; 1..5 = ein heimlicher Klassiker
# ======================================================

WAERMELEITFAEHIGKEIT = 50.0
QUERSCHNITT = 0.01
LAENGE = 1.0
LINKS = 20.0
RECHTS = 300.0
QUELLE_GESAMT = 160.0


def thomas_loeser(unter, diagonale, ober, rechte_seite):
    """Tridiagonaler Loeser (Thomas) -- reines Python, aus Kapitel 4."""
    n = len(diagonale)
    ober_neu = list(ober)
    seite_neu = list(rechte_seite)
    ober_neu[0] = ober[0] / diagonale[0]
    seite_neu[0] = rechte_seite[0] / diagonale[0]
    for i in range(1, n):
        nenner = diagonale[i] - unter[i] * ober_neu[i - 1]
        if i < n - 1:
            ober_neu[i] = ober[i] / nenner
        seite_neu[i] = (rechte_seite[i] - unter[i] * seite_neu[i - 1]) / nenner
    loesung = [0.0] * n
    loesung[n - 1] = seite_neu[n - 1]
    for i in range(n - 2, -1, -1):
        loesung[i] = seite_neu[i] - ober_neu[i] * loesung[i + 1]
    return loesung


def loese_stationaer(knotenzahl, quelle_gesamt, leitfaehigkeit, rechts_fest):
    """Stationaerer Traeger; rechts_fest=False laesst das rechte Ende isoliert."""
    elementzahl = knotenzahl - 1
    h = LAENGE / elementzahl
    k = leitfaehigkeit * QUERSCHNITT / h
    quelle_je_element = quelle_gesamt / elementzahl
    knotenquelle = [0.0] * knotenzahl
    for element in range(elementzahl):
        knotenquelle[element] = knotenquelle[element] + quelle_je_element / 2.0
        knotenquelle[element + 1] = knotenquelle[element + 1] + quelle_je_element / 2.0
```

```
    if rechts_fest:
        frei = list(range(1, knotenzahl - 1))
    else:
        frei = list(range(1, knotenzahl))    # rechtes Ende ist frei (isoliert)
    n = len(frei)
    unter = [0.0] * n
    diagonale = [0.0] * n
    ober = [0.0] * n
    seite = [0.0] * n
    for platz in range(n):
        knoten = frei[platz]
        # Diagonaleintrag: zwei Nachbarn, am freien rechten Ende nur einer.
        if knoten == knotenzahl - 1:
            diagonale[platz] = k
        else:
            diagonale[platz] = 2.0 * k
        if platz > 0:
            unter[platz] = -k
        if platz < n - 1:
            ober[platz] = -k
        seite[platz] = knotenquelle[knoten]
        if knoten - 1 == 0:
            seite[platz] = seite[platz] + k * LINKS
        if rechts_fest and knoten + 1 == knotenzahl - 1:
            seite[platz] = seite[platz] + k * RECHTS
    innen = thomas_loeser(unter, diagonale, ober, seite)
    if rechts_fest:
        return [LINKS] + innen + [RECHTS]
    return [LINKS] + innen


def exakte_parabel(ort):
    """Die exakte Parabel als Referenz."""
    hoecker = (QUELLE_GESAMT / LAENGE) / (2.0 * WAERMELEITFAEHIGKEIT * QUERSCHNITT)
    return LINKS + ort / LAENGE * (RECHTS - LINKS) + hoecker * ort * (LAENGE - ort)


def waermestrom(leitfaehigkeit):
    """Der stationaere Waermestrom durch den quellfreien Traeger (Watt)."""
    return leitfaehigkeit * QUERSCHNITT * (RECHTS - LINKS) / LAENGE


# --- die gewaehlte Aufgabe samt heimlicher Sabotage aufbauen ---
```

```
quelle = 0.0 if PRUEFFALL == "gerade" else QUELLE_GESAMT
leitfaehigkeit = WAERMELEITFAEHIGKEIT
rechts_fest = True
hinweis = "sauber (kein Fehler)."

if SABOTAGE == 1:
    rechts_fest = False
    hinweis = "Klassiker 1: rechte Randbedingung vergessen (isoliert)."
elif SABOTAGE == 2:
    KNOTENZAHL = 3
    hinweis = "Klassiker 2: Netz zu grob -- Knoten ok, Form kantig."
elif SABOTAGE == 3:
    leitfaehigkeit = 5.0
    hinweis = "Klassiker 3: Einheiten-Salat -- lambda zehnfach zu klein."
elif SABOTAGE == 4:
    quelle = -QUELLE_GESAMT
    hinweis = "Klassiker 4: Vorzeichen der Quelle vertauscht."
elif SABOTAGE == 5:
    leitfaehigkeit = 100.0
    quelle = 0.0
    hinweis = "Der Aufhaenger-Fehler: lambda doppelt, quellfrei -- unsichtbar am Bild, nur der Waerm

temperaturen = loese_stationaer(KNOTENZAHL, quelle, leitfaehigkeit, rechts_fest)

print("Pruefstation:", PRUEFFALL, "|", len(temperaturen), "Knoten | SABOTAGE =", SABOTAGE)
print("Knoten |  Ort   | FEM      | exakte Parabel | Abweichung")
print("-------+--------+----------+----------------+-----------")
h = LAENGE / (len(temperaturen) - 1)
groesste = 0.0
for i in range(len(temperaturen)):
    ort = i * h
    exakt = exakte_parabel(ort)
    d = temperaturen[i] - exakt
    if d < 0.0:
        d = -d
    if d > groesste:
        groesste = d
    print("  %2d   | %.3f m | %7.1f  |    %7.1f     |  %6.1f" % (i + 1, ort, temperaturen[i], exakt,
print()
print("Groesste Abweichung von der Parabel:", "%.2f Grad" % groesste)
print("Waermestrom (lambda A dT/L):", "%.0f W" % waermestrom(leitfaehigkeit),
      "(richtig waeren 140 W)")
print("Diagnose (erst selbst raten!):", hinweis)
```

```
# --- Abbildung ---
import matplotlib.pyplot as plt
xs = [i / 100.0 for i in range(101)]
orte = [i * h for i in range(len(temperaturen))]
fig, ax = plt.subplots(figsize=(7.0, 3.4))
ax.plot(xs, [exakte_parabel(x) for x in xs], "-", color="black", label="exakte Parabel")
ax.plot(orte, temperaturen, "o-", color="tab:red", label="FEM (evtl. sabotiert)")
ax.set_xlabel("Ort (m)")
ax.set_ylabel("Temperatur (°C)")
ax.set_title("Die Prüfstation")
ax.legend()
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt

def exakte_parabel(ort):
    hoecker = (160.0 / 1.0) / (2.0 * 50.0 * 0.01)
    return 20.0 + ort * 280.0 + hoecker * ort * (1.0 - ort)

xs = [i / 100.0 for i in range(101)]
orte = [i * 0.25 for i in range(5)]
gerade = [20.0 + o * 280.0 for o in orte]
fig, ax = plt.subplots(figsize=(7.0, 3.4))
ax.plot(xs, [exakte_parabel(x) for x in xs], "-", color="black", label="exakte Parabel")
ax.plot(orte, gerade, "o-", color="tab:red", label="FEM (SABOTAGE = 5)")
ax.set_xlabel("Ort (m)")
ax.set_ylabel("Temperatur (°C)")
ax.set_title("Die Prüfstation")
ax.legend()
plt.tight_layout()
plt.show()
```

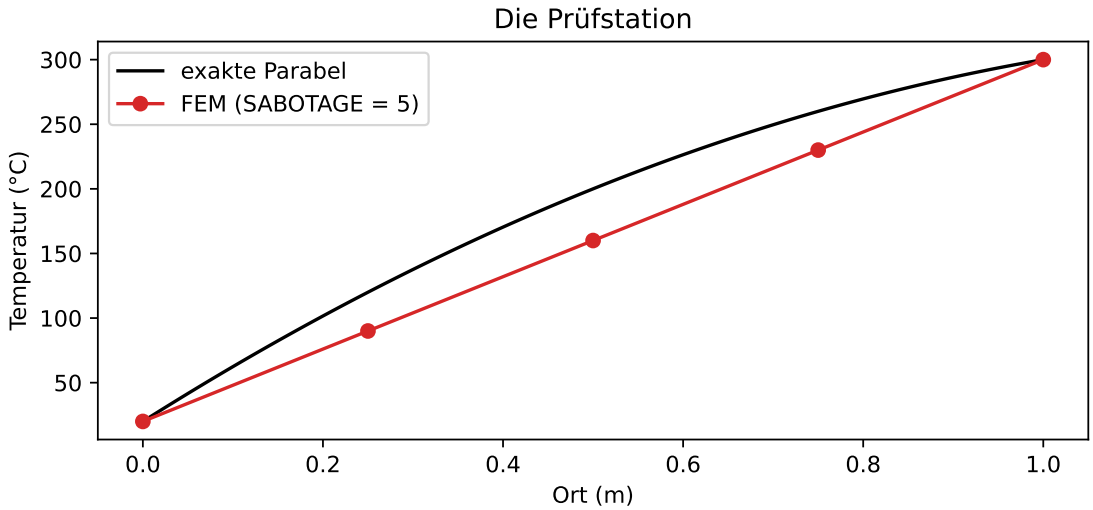

Abbildung 12.6.: Vorgerenderte Fassung der Prüfstation mit dem Startfall SABOTAGE = 5: Die rote FEM-Kurve liegt genau auf der Geraden 20 → 300 (die Parabel-Referenz gehört zum Quellfall und dient hier nur als Maßstab). Am Temperaturbild ist nichts zu sehen — der Faktor-2-Fehler im λ verrät sich erst am Wärmestrom (280 W statt 140 W). Im Browser ersetzt dein eigener Lauf dieses Bild.

12.9. So machen es die Großen

Diese Kultur — nichts glauben, was nicht gegen eine bekannte Lösung geprüft ist — ist keine Schrulle dieses Buches, sondern der Alltag ernsthafter Simulationssoftware. Das quelloffene Programm **OpenGeoSys**, mit dem der Playground dieses Buches arbeitet (Kolditz u. a. 2012), pflegt eine große Sammlung von **Benchmarks**, die bei *jeder* Codeänderung automatisch nachgerechnet werden. Einer davon ist der **Thiem-Brunnentest**: Pumpt man aus einem Brunnen im Grundwasser, senkt sich der Wasserspiegel zu einem Trichter, dessen genaue Form Günther Thiem schon 1906 als exakte Formel angab (Thiem 1906). Die Software rechnet denselben Trichter numerisch und legt ihn über die analytische Kurve — genau unser Nulltest-Gedanke, nur mit Wasser statt Wärme (im Playground unter `sim_thiem/`). Wer so prüft, darf seinem Programm auch dort trauen, wo es keine Formel mehr gibt.

12.10. Die Auflösung des Aufhängers

Zurück zu den zwei Berichten. Büro A sagt 140 W, Büro B sagt 280 W — und die Temperaturbilder sind identisch. Jetzt haben wir die Werkzeuge. Der Fehler: Büro B hat sich bei der Wärmeleitfähigkeit um den Faktor zwei vertan — ein klassischer Einheiten-Salat, $\lambda = 100$ statt $50\ \mathrm{W/(m\,K)}$.

Warum ist das am Bild nicht zu sehen? Weil im quellfreien Träger die Temperaturgerade **gar nicht von λ abhängt** — die Wärmeleitfähigkeit kürzt sich aus der stationären Gleichung heraus. Ob λ nun 50 oder 100 ist: Die Lösung ist dieselbe Gerade 20 → 300. Darum bestehen **beide** Büros den **Nulltest** mit Bravour — das schärfste Werkzeug ist hier *blind*.

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap12"))
from kap12_pruefstation import loese_stationaer, waermestrom_durch_traeger

def als_text(temperaturen):
    """Rundet eine Temperaturliste zu lesbarem Text (ohne Comprehension)."""
    stuecke = []
    for wert in temperaturen:
        stuecke.append("%.1f" % wert)
    return "  ".join(stuecke)


richtig = loese_stationaer(5, 0.0, leitfaehigkeit=50.0)
falsch = loese_stationaer(5, 0.0, leitfaehigkeit=100.0)
print("Temperaturgerade bei lambda = 50:", als_text(richtig))
print("Temperaturgerade bei lambda = 100:", als_text(falsch))
print("-> identisch: der Nulltest schweigt.")
print()
print("Waermestrom lambda =  50 W/(m K):", "%.0f W" % waermestrom_durch_traeger(50.0))
print("Waermestrom lambda = 100 W/(m K):", "%.0f W" % waermestrom_durch_traeger(100.0))
```

```
Temperaturgerade bei lambda = 50: 20.0  90.0  160.0  230.0  300.0
Temperaturgerade bei lambda = 100: 20.0  90.0  160.0  230.0  300.0
-> identisch: der Nulltest schweigt.

Waermestrom lambda =  50 W/(m K): 140 W
Waermestrom lambda = 100 W/(m K): 280 W
```

Interpretation: Der Fehler steckt nicht in der Temperatur, sondern im **Wärmestrom** — und der *hängt* von λ ab: $Q = \lambda A\,\Delta T/L = 50{\cdot}0{,}01{\cdot}280/1 =$

$140\,\mathrm{W}$ mit dem richtigen λ, und glatt das Doppelte, **280 W**, mit dem falschen. Die eine Kennzahl, die das schöne Bild nicht zeigt, ist genau die, in der der Fehler wohnt. Büro A hat recht. Und weil man den wahren Wärmestrom aus dem Brandherd unabhängig kennt (oder von Hand nachrechnet), fällt Büro B beim Punkt 5 der Checkliste sofort durch — die Plausibilität von Hand.

Das ist die Quintessenz des Kapitels: Ein FEM-Programm liefert immer ein plausibles, buntes Bild. Ob es *stimmt*, entscheidet nicht das Bild, sondern die Prüfung — Nulltest, Konvergenz, Bilanz, Wärmestrom, Handrechnung. **Bunt ist kein Beweis.**

Merkkasten

- **Verifikation** (rechne ich richtig?) prüft man rechnerisch — mit Prüffällen, Konvergenz und Bilanzen; **Validierung** (rechne ich das Richtige?) braucht Messungen. Erst das eine, dann das andere.
- Der **Nulltest** (die Gerade) muss die FEM *exakt* treffen — jede Abweichung ist ein Bug, kein Netzfehler. Sein blinder Fleck: Fehler, die die exakte Lösung nicht verändern.
- Bei der **Parabel** viertelt sich der Fehler mit jeder Netzhalbierung (zweite Ordnung); in 1D sind die Knoten sogar exakt, der Fehler lebt *dazwischen*.
- **Zeit ist gröber als Raum:** der implizite Euler halbiert den Fehler mit Δt nur (erste Ordnung) — Zeitschritt beim Verfeinern nicht vergessen.
- Die **Energiebilanz** läuft immer mit: rein $=$ gespeichert $+$ raus, auf 1 % genau.
- **Bunt ist kein Beweis.** Erst wenn die fünf Haken der Simulations-Checkliste sitzen, glaubt man dem Bild.

Roter Faden

Wo kam das schon vor, wo kommt es wieder? — Dieses Kapitel hat die Versprechen eingelöst, die der ganze Teil III und IV gestreut hat: die **Gerade** aus Kapitel 6 wird zum Nulltest, die **Parabel** aus Kapitel 6 und Übung 7.3 zum Diskretisierungs-Prüffall, die **Fühler-Kurve** aus Kapitel 5 entpuppt sich als die exakte Reihe, und die **Randbedingungs-Checkliste** aus Kapitel 8 wächst zur Simulations-Checkliste. Die **Elementfamilien** aus 1 und Kapitel 11 treten gegeneinander an. — Nach vorn: Der **Nulltest** eröffnet Kapitel 13, wo freie Wärmedehnung *Spannung null* ergeben muss (Übung 12.3 entwirft ihn schon); und die **Simulations-Checkliste** hängt im Finale (3) neben den Reglern, wenn du selbst am vollständigen FEM-Programm

drehst.

Übungen

Ü 12.1 (Verstehen). Sieh dir den Konvergenzplot (Abbildung 12.2) an. Der Zwischen-Knoten-Fehler der Parabel beträgt bei 17 Knoten 0,156 °C, bei 33 Knoten 0,039 °C. Du brauchst für deine Anwendung eine Genauigkeit von **0,1 °C**. Wie viele Knoten reichen — und wie viele sind es ungefähr, wenn du den Viertelungs-Rhythmus zum Abschätzen nutzt?

> **Musterlösung zu Ü 12.1**
>
> 17 Knoten (0,156 °C) reichen **nicht** ganz, 33 Knoten (0,039 °C) reichen locker — also liegt die Grenze dazwischen. Genauer: Der Fehler wächst mit h^2, also Fehler $\approx 40 \cdot h^2$ (in °C, mit h in Metern). Für 0,1 °C: $h^2 = 0{,}1/40 = 0{,}0025$, also $h = 0{,}05\,\text{m}$. Das sind $L/h = 20$ Elemente, mithin **21 Knoten**. Wer nur die tabellierten Netze 5/9/17/33 zur Hand hat, nimmt sicherheitshalber die 33 — lieber etwas zu fein als zu grob. Die Lehre: Man liest die nötige Feinheit direkt aus der Konvergenzkurve ab, statt zu raten.

Ü 12.2 (Verändern). In der Prüfstation (bzw. im Skript `loesungen/kap12_ue2.py`) spielst du Zeitschritt und Maschenweite gegeneinander aus. Verfeinere beim transienten Fall **nur das Netz** und lass $\Delta t = 200\,\text{s}$ fest. Was passiert mit dem Fehler ab einer gewissen Knotenzahl — und warum? Wann lohnt feineres Netz nichts mehr?

> **Musterlösung zu Ü 12.2**
>
> Der Gesamtfehler ist ungefähr die Summe aus einem Raum-Anteil (viertelt sich mit h) und einem Zeit-Anteil (halbiert sich mit Δt). Verfeinert man nur das Netz, fällt der Fehler zunächst — bis er auf dem **Zeit-Anteil** aufsitzt und sich durch noch mehr Knoten nicht weiter drücken lässt: Bei $\Delta t = 200\,\text{s}$ bleibt er ab etwa 17 Knoten bei rund 7,4 °C stehen, ganz gleich, wie fein das Netz noch wird (Skript `kap12_ue2.py`, Teil a). Erst ein kleineres Δt senkt ihn weiter (Teil b, brave Halbierung). Merke: Feineres Netz und kleineres Δt müssen **zusammen** gehen — wer nur eines verfeinert, läuft in den Boden des anderen. *Zeit ist gröber als Raum.*

Ü 12.3 (Übertragen). Entwirf einen **Nulltest für die Thermomechanik** (Vorgriff auf Kapitel 13). Ein frei aufgehängter Träger (nirgends festge-

klemmt) wird gleichmäßig erwärmt. Er dehnt sich aus — aber welche **Spannung** muss dabei im Inneren herrschen? Beschreibe das Prüf-Protokoll: Was ist der bekannte Sollwert, und was schließt du aus einer Abweichung?

> **i** Musterlösung zu Ü 12.3
>
> **Sollwert:** Spannung **exakt null**, überall. Ein frei beweglicher Körper, der sich gleichmäßig erwärmt, dehnt sich einfach ungehindert aus — er darf länger werden, ohne dass irgendwo eine Kraft entsteht. Spannung baut sich erst auf, wenn die Ausdehnung *behindert* wird (eingeklemmte Enden). **Protokoll:** (1) Träger frei lagern (keine Einspannung, aber gerade so viele Lager, dass er nicht davonschwebt — statisch bestimmt). (2) Gleichmäßige Temperaturerhöhung ΔT aufbringen. (3) Die berechnete Spannung an allen Knoten ablesen. (4) **Erwartung:** null (bis auf Rechenungenauigkeit). Jede von null verschiedene Spannung ist ein **Bug** — meist eine versehentlich zu starre Lagerung, ein falscher Ausdehnungskoeffizient oder ein Vorzeichenfehler in der Kopplung Temperatur → Dehnung. Das ist derselbe Nulltest-Gedanke wie bei der Geraden: eine Aufgabe mit exakt bekannter, im Ansatzraum liegender Lösung, an der ein Fehler *garantiert* auffällt. Kapitel 13 baut ihn wirklich.

Das Kleingedruckte

Dieses Kapitel **verifiziert**, es **validiert** nicht. Wir haben die FEM gegen exakte Lösungen *derselben* Gleichung gehalten — aber nie gefragt, ob diese Gleichung die Wirklichkeit trifft. Der echte Stahlträger gibt Wärme an die Luft ab (das haben wir weggelassen), sein λ ändert sich mit der Temperatur (wir halten es konstant), und bei mehreren hundert Grad spielen Effekte hinein — allen voran der Festigkeitsverlust des Stahls —, die unser Modell gar nicht kennt. Solche Fragen beantwortet nur die **Validierung** — der Vergleich mit einer echten Messung. Sie ist der ehrliche nächste Schritt, den dieses Buch nicht mehr geht.

Zwei weitere Grenzen: Wir haben **keine Fehlerabschätzungs-Theorie** getrieben (keine Normen, keine Beweise — nur den handfesten Viertelungs-Rhythmus), und wir haben die Netze **von Hand** verfeinert. Professionelle Programme verfeinern **adaptiv**: Sie schätzen selbst, *wo* der Fehler groß ist (etwa an der wandernden Front), und setzen genau dort mehr Knoten. Das ist eine eigene Kunst — für uns bleibt es ein Namedrop am Wegesrand.

Teil V.

Teil V — Der Balken verformt sich

13. Der Träger dehnt sich: 1D-Thermomechanik

Zwölf Kapitel lang haben wir eine einzige Frage an den Träger gestellt: *Wie warm ist er, wo?* Wir haben das Temperaturfeld erobert — stationär (Kapitel 6), mit der schwachen Form gelöst (Kapitel 7), über die Zeit verfolgt (Kapitel 9) und schließlich geprüft (Kapitel 12). Aber ganz am Anfang, in Kapitel 1, hat die Blackbox noch etwas anderes gezeigt: Der heiße Träger wurde nicht nur bunt — er wurde **länger**. Dieses Versprechen lösen wir jetzt ein.

Warum ruhen die schweren Enden großer Brücken auf Rollen, und warum lagen in alten Bahngleisen alle paar Meter Stoßlücken? Weil Metall, das man erwärmt, sich ausdehnen *will* — und weil es, wenn man es nicht lässt, mit ungeheurer Kraft zurückdrückt. Unser Stahlträger, um 100 Grad erwärmt, möchte 1,2 mm pro Meter länger werden. Lässt man ihn gewähren, passiert nichts Dramatisches: Er wird eben ein bisschen länger. Steckt er aber mit beiden Enden fest in den Wänden, kann er nicht länger werden — und drückt dann mit rund zweieinhalb **Meganewton** gegen seine Auflager, der Gewichtskraft von etwa 257 Tonnen. Genau das ist ein gefürchteter Vorgang im Brandfall: Ein sich ausdehnender Stahlträger kann die Wände, in denen er steckt, nach außen drücken. In diesem Kapitel rechnest du beide Fälle aus, und du wirst sehen: Es ist wieder die Federkette aus Kapitel 3, nur dass die Feder diesmal ein Stück Stahl ist.

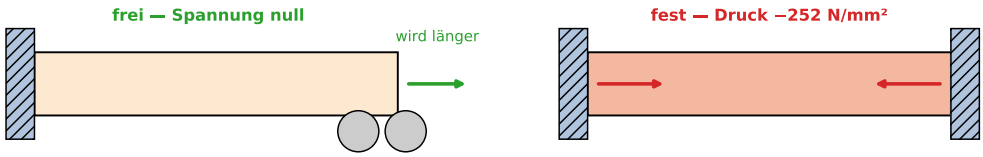

Abbildung 13.1.: Zwei Wege, mit der Wärmedehnung umzugehen. Links das Rollenlager einer Brücke: Ein Ende darf sich frei verschieben, der Träger wird bei Hitze länger — und bleibt spannungsfrei. Rechts der beidseitig eingespannte Träger: Er kann nicht länger werden, also staut sich die verhinderte Dehnung als gewaltige Druckspannung im Inneren. Genau diese beiden Fälle — frei und fest — sind die Grenzfälle des ganzen Kapitels.

Was man hier sieht: dieselbe Erwärmung, zwei Lagerungen, zwei völlig verschiedene Ergebnisse. Links darf sich der Träger ausdehnen und trägt danach *keine* Spannung; rechts ist ihm die Dehnung verwehrt, und genau diese verhinderte Dehnung wird zur Druckspannung. Der ganze Rest des Kapitels baut die Werkzeuge, um beide Zahlen — die Verlängerung links, die Spannung rechts — selbst auszurechnen.

Lernziele

Nach diesem Kapitel kannst du …

1. … **Dehnung** ε (Längenänderung geteilt durch Länge, dimensionslos) und **Spannung** σ (Kraft geteilt durch Fläche, in N/mm²) definieren und über das Hookesche Gesetz $\sigma = E\,\varepsilon$ verknüpfen — die Feder aus Kapitel 3, jetzt pro Material statt pro Bauteil,
2. … die **Wärmedehnung** $\alpha\,\Delta T$ berechnen (Stahl, 100 K → 1,2 mm pro Meter) und von der elastischen Dehnung unterscheiden,
3. … die **Kernformel** $\sigma = E\,(\varepsilon - \alpha\,\Delta T)$ in Worten erklären: Spannung entsteht nur aus dem *verhinderten* Teil der Dehnung,
4. … die beiden Grenzfälle rechnen — frei gelagert (Dehnung ja, Spannung null) und beidseitig eingespannt (Dehnung null, $\sigma = -E\,\alpha\,\Delta T$, beim um 100 K erwärmten Stahlträger satte $-252\ \mathrm{N/mm^2}$ Druck),
5. … das **FEM-Stab-System** aufstellen: dieselbe Matrix wie die Federkette, aber mit $k = E\,A/h$ und einer neuen rechten Seite aus **thermischen Kräften** — und die Verformung eines gekoppelten 5-Knoten-Trägers ausrechnen und deuten.

⚠ Naheliegende Vermutung

Vermutung: *„Wo Wärme ist, ist Spannung. Ein heißer Träger steht unter Spannung — je heißer, desto mehr."*
Warum sie naheliegt: Wärme und Kraft fühlen sich verwandt an. Ein heißer Topf, ein berstendes Glas unter kochendem Wasser, verzogene Bratbleche — überall scheint Hitze unmittelbar Spannung zu erzeugen. Und die Aufhänger-Zahl (257 Tonnen!) bestärkt den Eindruck.
Was stattdessen stimmt: Ein frei gelagerter, gleichmäßig erwärmter Träger dehnt sich **völlig spannungsfrei** aus. Er wird länger, und dabei entsteht *nirgends* eine Kraft. Spannung baut sich erst auf, wenn die Dehnung **verhindert** wird — durch ein festes Auflager oder dadurch, dass verschieden heiße Teile aneinanderhängen und sich gegenseitig festhalten. Der Merksatz, den dieses Kapitel an vielen Stellen verankert, lautet:

Spannung = E × verhinderte Dehnung.

Nicht die Wärme macht die Spannung, sondern der *Widerstand* gegen die Ausdehnung. Das berstende Glas zerspringt, weil seine heiße Innenseite sich ausdehnen will und die kalte Außenseite sie festhält — verhinderte Dehnung, nicht Hitze an sich.

13.1. Schritt 1 — Dehnung: Verformung ohne Einheit

Fangen wir beim Einfachsten an: beim Ziehen an einem Stab. Hängt man an einen Draht ein Gewicht, wird er ein Stückchen länger. Aber „ein Stückchen" ist keine brauchbare Angabe — ein Millimeter Verlängerung bedeutet bei einem Fingerhut-kurzen Draht etwas ganz anderes als bei einem Kilometer Stahlseil. Was zählt, ist die Verlängerung **im Verhältnis zur Länge**. Genau das ist die **Dehnung**:

$$\varepsilon = \frac{\Delta L}{L} = \frac{\text{Längenänderung}}{\text{ursprüngliche Länge}}.$$

Weil oben und unten dieselbe Einheit steht (Meter durch Meter), ist die Dehnung eine **reine Zahl ohne Einheit**. Ein Beispiel mit unserem Träger: Ziehen wir den 1 Meter langen Stahlträger um $0{,}5$ mm länger, so ist

$$\varepsilon = \frac{0{,}5\ \text{mm}}{1000\ \text{mm}} = 0{,}0005 = 0{,}05\,\% = 500 \cdot 10^{-6}.$$

Weil die Zahlen so klein sind, schreibt man sie in der Praxis oft in **Mikrostrain** (10^{-6}): $500\ \mu\varepsilon$. Ob als Bruch, als Prozent oder in Mikrostrain — es ist dieselbe dimensionslose Zahl. Merke sie dir als „so viel Weglänge pro Länge": Dehnung ist keine Strecke, sondern ein Streckungsfaktor.

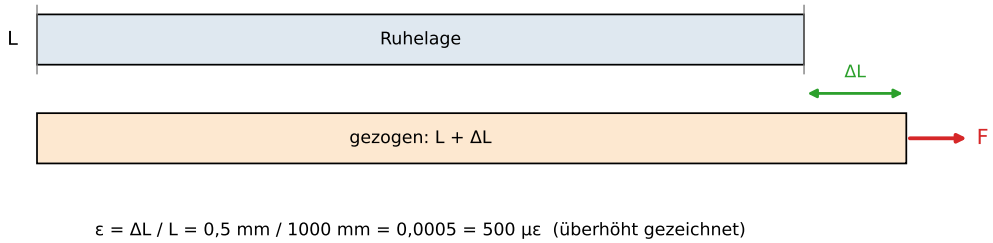

Abbildung 13.2.: Dehnung ist Längenänderung geteilt durch Länge. Oben der Träger in Ruhe (Länge L = 1 m), unten um ΔL = 0,5 mm gezogen (stark überhöht dargestellt). Die Dehnung ε = ΔL/L = 0,0005 ist eine reine Zahl — dieselbe Zahl gälte für einen 2 m langen Träger, der um 1 mm gezogen wird. Sie misst die Streckung, nicht die Strecke.

Was man hier sieht: oben der ruhende Träger, unten der gezogene. Die grüne Marke ΔL ist stark überhöht — in Wahrheit wären $0{,}5$ mm auf dieser Skala unsichtbar. Die Dehnung fasst beide Zustände in einer einzigen Zahl zusammen. Und das Schöne: Dieselbe Dehnung entsteht auf zwei ganz verschiedene Arten — durch **Ziehen** (wie hier) oder durch **Erwärmen** (gleich in Schritt 4). Die Dehnung unterscheidet nicht, woher sie kommt; das wird der Schlüssel zur Kernformel.

13.2. Schritt 2 — Spannung: Kraft, auf die Fläche verteilt

Die Kraft, mit der man zieht, sagt für sich genommen wenig über die Belastung des Materials. Zwei Newton an einem Haardraht zerreißen ihn; zwei Newton an einem Schiffstau spürt man kaum. Was das Material wirklich belastet, ist die Kraft **pro Querschnittsfläche** — die **Spannung**:

$$\sigma = \frac{F}{A} = \frac{\text{Kraft}}{\text{Querschnittsfläche}}.$$

Die Einheit ist Kraft durch Fläche, also Newton pro Quadratmeter — das Pascal (Pa). Weil ein Pascal winzig ist (ein Blatt Papier drückt mit einigen Pascal auf den Tisch), rechnet die Mechanik in **Newton pro Quadratmillimeter**: $1\ \mathrm{N/mm^2} = 10^6\ \mathrm{Pa} = 1\ \mathrm{MPa}$. Unser Träger hat den Querschnitt $A = 0{,}1\ \mathrm{m} \times 0{,}1\ \mathrm{m} = 0{,}01\ \mathrm{m^2} = 10\,000\ \mathrm{mm^2}$. Um in ihm die Spannung $252\ \mathrm{N/mm^2}$ zu erzeugen, braucht es demnach

$$F = \sigma \cdot A = 252\ \tfrac{\mathrm{N}}{\mathrm{mm^2}} \cdot 10\,000\ \mathrm{mm^2} = 2\,520\,000\ \mathrm{N} = 2{,}52\ \mathrm{MN}.$$

Das ist die 257-Tonnen-Kraft aus dem Aufhänger. Sie steht schon fest, bevor wir überhaupt zur FEM kommen — sie folgt allein aus Spannung mal Fläche.

Stolperstein: N/mm² und GPa nicht verwechseln

Hier lauert der häufigste Zahlenfehler der Strukturmechanik. Es kursieren zwei Einheiten, die sich um den Faktor **tausend** unterscheiden:

- **Spannungen** notiert man in $\mathrm{N/mm^2}$ (= MPa). Typische Werte: einige zehn bis einige hundert $\mathrm{N/mm^2}$.
- **Steifigkeiten** (der E-Modul, gleich in Schritt 3) notiert man in **Gigapascal** (GPa). $1\ \mathrm{GPa} = 1000\ \mathrm{N/mm^2}$.

Baustahl hat den E-Modul $E = 210\ \mathrm{GPa} = 210\,000\ \mathrm{N/mm^2}$. Verwechselt man die beiden, rechnet man um den Faktor 1000 daneben — aus $252\ \mathrm{N/mm^2}$ würden $0{,}252\ \mathrm{GPa}$, und wer das mit einer Spannung in $\mathrm{N/mm^2}$ vergleicht, zieht falsche Schlüsse. Faustregel: **E-Moduln in GPa, Spannungen in N/mm², und** $1\ \mathrm{GPa} = 1000\ \mathrm{N/mm^2}$**.**

13.3. Schritt 3 — Hooke als Material-Feder: dieselbe Formelgestalt

Dehnung und Spannung hängen zusammen, und zwar auf die einfachste denkbare Weise: proportional. Doppelte Dehnung, doppelte Spannung. Diese Proportionalität ist das **Hookesche Gesetz** — dieselbe Idee wie die Feder aus Kapitel 3 ($F = k\,x$), nur eine Ebene tiefer, im Material selbst:

$$\sigma = E \cdot \varepsilon.$$

Die Proportionalitätszahl E heißt **Elastizitätsmodul** oder kurz E-Modul. Sie ist die „Steifigkeit des Materials": Wie viel Spannung braucht es für eine bestimmte Dehnung? Ein großes E bedeutet ein steifes Material (Stahl, 210 GPa), ein kleineres ein nachgiebigeres (Aluminium, 70 GPa). Anders als die Federhärte k aus Kapitel 3, die vom ganzen Bauteil abhing, ist E eine **reine Materialeigenschaft** — derselbe Baustahl hat immer 210 GPa, ob als Träger oder als Draht.

> **Namensschild: Thomas Young**
>
> Im Englischen heißt der E-Modul *Young's modulus*, nach dem britischen Universalgelehrten **Thomas Young** (1773–1829) — Arzt, Physiker und einer der Entzifferer der ägyptischen Hieroglyphen. Young fasste um 1807 die Steifigkeit eines Materials als eigene Kennzahl, unabhängig von der Bauteilform. Die deutsche Bezeichnung „Elastizitätsmodul" betont dieselbe Sache: das Maß (*modulus*) für die Elastizität eines Stoffes.

Jetzt bauen wir aus dem Materialgesetz wieder eine **Feder** für ein Trägerstück — und hier passiert etwas, das dir bekannt vorkommen wird. Ein Stück Träger der Länge h mit Querschnitt A: Zieht man seine Enden um $u_2 - u_1$ auseinander, ist die Dehnung $\varepsilon = (u_2 - u_1)/h$, die Spannung $\sigma = E\,\varepsilon$, und die Kraft $F = \sigma\,A = E\,A\,(u_2 - u_1)/h$. Das ist wieder $F = k\,(u_2 - u_1)$ — mit der Steifigkeit

$$k \;=\; \frac{E\,A}{h} \;=\; \frac{210 \cdot 10^9\ \text{Pa} \cdot 0{,}01\ \text{m}^2}{0{,}25\ \text{m}} \;=\; 8{,}4 \cdot 10^9\ \tfrac{\text{N}}{\text{m}}.$$

Diese Zahl ist gewaltig. $8{,}4 \cdot 10^9$ N/m heißt: Um das Trägerstück um einen **tausendstel Millimeter** (ein Mikrometer) zu stauchen, braucht es $8{,}4 \cdot 10^9 \cdot 10^{-6} = 8400$ N $= 8{,}4$ kN — das Gewicht eines Kleinwagens, für einen Mikrometer. Stahl ist eine extrem steife Feder.

Und nun halte inne und vergleiche $k = E\,A/h$ mit der **Wärme-Feder** aus Kapitel 7: dort war $k = \lambda\,A/h$. **Es ist wörtlich dieselbe Formelgestalt** — Materialzahl mal Fläche durch Länge. In Kapitel 7 war es ein Gänsehaut-Moment, dass Wärmeleitung eine Feder ist; hier schließt sich der Kreis. Die Mechanik *ist* die ursprüngliche Feder, und die Wärme hat nur ihre Gestalt geliehen. Das ist kein Zufall, sondern eine tiefe Verwandtschaft, die wir jetzt zur wichtigsten Grafik des Kapitels machen.

	WÄRME (Kap. 6–9)	MECHANIK (Kap. 13)
gesuchte Größe (pro Knoten)	Temperatur T	Verschiebung u
ihr Gefälle / Anstieg	Temperaturgefälle dT/dx	Dehnung ε = du/dx
innere „Fluss"-Größe	Wärmestrom(dichte) q	Spannung σ
Materialgesetz	q = −λ · dT/dx (Fourier)	σ = E · ε (Hooke)
Materialzahl	Wärmeleitfähigkeit λ = 50	E-Modul E = 210 GPa
Element-Steifigkeit	k = λ·A / h	k = E·A / h
Quelle / Last (rechte Seite)	Wärmequelle im Element	thermische Kraft E·A·α·ΔT
fester Wert am Rand	vorgegebene Temperatur	festes Auflager (u vorgegeben)
vorgegebener Fluss am Rand	aufgeprägter Wärmestrom	aufgeprägte Kraft

Abbildung 13.3.: Die Analogie-Tafel Wärme ↔ Mechanik. Zeile für Zeile dieselbe Maschine mit ausgetauschten Namen: Die gesuchte Größe ist einmal die Temperatur, einmal die Verschiebung; ihr Gefälle ist einmal das Temperaturgefälle, einmal die Dehnung; die Materialzahl ist λ = 50 W/(m·K) bzw. E = 210 GPa; die Element-Steifigkeit ist in beiden Fällen (Materialzahl)·A/h. Diese Tafel kehrt in Kapitel 14 und 15 wieder — sie ist der Grund, warum wir die ganze FEM-Maschinerie nur einmal bauen mussten.

Was man hier sieht: die beiden Physiken des Buches, Zeile für Zeile nebeneinander. Lies eine beliebige Zeile von links nach rechts — es ist stets dieselbe Rolle, einmal für die Wärme, einmal für die Mechanik besetzt. Die gesuchte Größe (Temperatur ↔ Verschiebung), ihr Gefälle (Temperaturgefälle ↔ Dehnung), die innere Flussgröße (Wärmestrom ↔ Spannung), die Materialzahl ($\lambda = 50\ \mathrm{W/(m\,K)} \leftrightarrow E = 210$ GPa) und ganz zentral die Element-Steifigkeit ($\lambda A/h \leftrightarrow EA/h$). Weil die rechte Spalte dieselbe Maschine ist wie die linke, können wir die gesamte FEM-Maschinerie aus Kapitel 3 und Kapitel 7 wiederverwenden. Nur zwei Zeilen sind wirklich neu und brauchen noch Arbeit: die **thermische Kraft** auf der rechten Seite (Schritt 6) und der Weg von der Wärme *in* die Mechanik (Schritt 4).

13.4. Schritt 4 — Wärmedehnung: der Träger will länger werden

Bis hier kam die Dehnung vom Ziehen. Aber es gibt einen zweiten Weg, einen Träger zu dehnen, ganz ohne Kraft: **erwärmen**. Die Atome eines erwärmten Metalls schwingen heftiger und beanspruchen im Mittel mehr Platz — das Material wird in alle Richtungen größer. Wie viel, sagt der **Wärmeausdehnungs- koeffizient** α: Pro Grad Temperaturerhöhung dehnt sich das Material um α (wieder eine Dehnung pro Kelvin). Die **freie Wärmedehnung** ist

$$\varepsilon_{\text{therm}} \;=\; \alpha \cdot \Delta T .$$

Für unseren Baustahl ist $\alpha \;=\; 12 \cdot 10^{-6}\ \frac{1}{\text{K}}$. Erwärmt man den Träger um $\Delta T = 100\ \text{K}$, will er sich um

$$\varepsilon_{\text{therm}} \;=\; 12 \cdot 10^{-6}\ \tfrac{1}{\text{K}} \cdot 100\ \text{K} \;=\; 1{,}2 \cdot 10^{-3} \;=\; 1200\ \mu\varepsilon$$

dehnen — auf einem Meter Länge also $\Delta L = \varepsilon_{\text{therm}} \cdot L = 1{,}2 \cdot 10^{-3} \cdot 1\ \text{m} = 1{,}2\ \text{mm}$. Das ist die Zahl aus dem Aufhänger. Zum Vergleich verschiedene Materialien, jeweils für $100\ \text{K}$ auf einem Meter:

```
def waermedehnung_mm_pro_meter(alpha_pro_kelvin, delta_t_kelvin):
    """Freie Waermedehnung in mm pro Meter Traegerlaenge.

    alpha_pro_kelvin: Waermeausdehnungskoeffizient (1/K).
    delta_t_kelvin: Temperaturerhoehung (K).
    Rueckgabe: Laengenaenderung in mm je Meter Ausgangslaenge.
    """
    dehnung = alpha_pro_kelvin * delta_t_kelvin
    return dehnung * 1000.0


MATERIALIEN = [
    ("Stahl", 12e-6),
    ("Aluminium", 23e-6),
    ("Fensterglas", 9e-6),
]
DELTA_T = 100.0

print("Material    | alpha (1/K) | Dehnung pro 100 K | mm pro Meter")
```

```
print("------------+-------------+-------------------+-------------")
for name, alpha in MATERIALIEN:
    dehnung = alpha * DELTA_T
    print("%-11s | %10.1e | %14.4f %% | %8.2f mm" % (
        name, alpha, dehnung * 100.0, waermedehnung_mm_pro_meter(alpha, DELTA_T)))
```

```
Material    | alpha (1/K) | Dehnung pro 100 K | mm pro Meter
------------+-------------+-------------------+-------------
Stahl       |    1.2e-05 |         0.1200 % |     1.20 mm
Aluminium   |    2.3e-05 |         0.2300 % |     2.30 mm
Fensterglas |    9.0e-06 |         0.0900 % |     0.90 mm
```

Interpretation der Ausgabe: Unser Stahl dehnt sich pro 100 K um gut einen Millimeter je Meter — deutlich weniger als Aluminium (fast doppelt so viel) und etwas mehr als Glas. Zwei Dinge zum Merken: Die Wärmedehnung ist *dieselbe Art Zahl* wie die Zug-Dehnung aus Schritt 1 (dimensionslos), und sie entsteht **ohne jede Kraft** — ein frei liegender Träger dehnt sich einfach, spannungsfrei.

> **Warum Stahlbeton funktioniert**
>
> Beton und Stahl haben fast denselben Wärmeausdehnungskoeffizienten (Beton etwa $10 \cdot 10^{-6}$, Baustahl $12 \cdot 10^{-6}$ $\frac{1}{\mathrm{K}}$). Deshalb dehnen sich Bewehrungsstahl und umgebender Beton bei Temperatur- schwankungen fast gleich stark — sie halten sich *nicht* gegenseitig fest, und es entsteht kaum thermische Spannung an ihrer Grenzfläche. Wären die Werte weit auseinander, würde jeder heiße Sommertag den Verbund lockern. Stahlbeton ist ein Glücksfall der Materialwahl — und ein Vorgeschmack auf die Nahtstellen-Spannung, die uns in Übung 13.3 beschäftigt.

13.5. Schritt 5 — Die Kernformel: Spannung aus verhinderter Dehnung

Jetzt kommen die beiden Dehnungsarten zusammen, und daraus wird die wichtigste Formel des ganzen Teils V. Die Überlegung geht in zwei Schritten — man kann sie sich als kleine Geschichte erzählen:

1. **Erst frei dehnen lassen.** Wir erwärmen den Träger um ΔT und lassen ihn zunächst völlig ungehindert. Er dehnt sich um $\alpha\,\Delta T$ — ganz ohne Kraft, ohne Spannung.

2. **Dann auf Sollmaß zurückzwingen.** Jetzt schauen wir, wie lang der Träger *tatsächlich* sein darf. Diese wirkliche, von außen erlaubte Dehnung nennen wir ε (die Gesamtdehnung). Ist sie kleiner als die freie Wärmedehnung, müssen wir den Träger um die Differenz $\varepsilon - \alpha\,\Delta T$ **zurückstauchen** — und *dieses* Zurückzwingen kostet Kraft, erzeugt also Spannung.

Nur der zweite Teil zählt für die Spannung. Die Spannung folgt Hooke, aber nicht aus der ganzen Dehnung, sondern aus dem **verhinderten Teil**:

$$\boxed{\sigma \;=\; E\,(\varepsilon - \alpha\,\Delta T)}$$

In Worten — der Merksatz von oben, jetzt als Formel: **Spannung ist E mal verhinderte Dehnung.** Von der ganzen Dehnung ε wird die freie Wärmedehnung $\alpha\,\Delta T$ abgezogen; was übrig bleibt, ist der Teil, den das Material gegen seinen Willen mitmachen musste, und nur der spannt es.

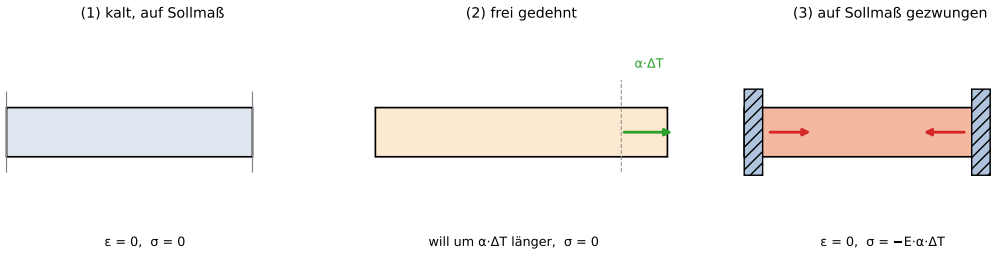

Abbildung 13.4.: Die Kernformel als Bildgeschichte in drei Akten. (1) Ausgang: der kalte Träger auf Sollmaß. (2) Frei dehnen lassen: erwärmt will er sich um α·ΔT verlängern — spannungsfrei. (3) Auf Sollmaß zurückzwingen: erlaubt das Auflager keine Verlängerung (ε = 0), muss der Träger um die volle Wärmedehnung zurückgestaucht werden — diese verhinderte Dehnung wird zur Druckspannung σ = −E·α·ΔT. Nur der zurückgezwungene Teil spannt.

Was man hier sieht: die Formel als Ablauf. Akt 2 ist spannungsfrei — freie Dehnung kostet nichts. Erst Akt 3, das Zurückzwingen auf das erlaubte Maß, erzeugt die Kraft. Jetzt lesen wir die beiden **Grenzfälle** direkt aus der Formel ab; sie sind die Eckpfeiler des Kapitels.

Grenzfall A — frei gelagert (Spannung null). Der Träger darf sich ausdehnen, so viel er will. Dann ist die erlaubte Gesamtdehnung genau die freie Wärmedehnung, $\varepsilon = \alpha\,\Delta T$, und die Klammer wird null:

$$\sigma \;=\; E\,(\alpha\,\Delta T - \alpha\,\Delta T) \;=\; 0.$$

Der Träger wird um $\Delta L = \alpha\,\Delta T \cdot L = 1{,}2$ mm länger und trägt **keine Spannung**. Genau das ist die Vermutung des Kapitels, widerlegt: Wärme allein macht keine Spannung.

Grenzfall B — beidseitig eingespannt (Dehnung null). Jetzt halten zwei feste Auflager den Träger auf seiner kalten Länge fest. Er *darf* sich nicht dehnen, also ist die Gesamtdehnung $\varepsilon = 0$, und die volle Wärmedehnung ist verhindert:

$$\sigma \;=\; E\,(0 - \alpha\,\Delta T) \;=\; -E\,\alpha\,\Delta T \;=\; -210\,000\,\tfrac{\mathrm{N}}{\mathrm{mm}^2}\cdot 12\cdot 10^{-6}\,\tfrac{1}{\mathrm{K}}\cdot 100\,\mathrm{K} \;=\; -252\,\tfrac{\mathrm{N}}{\mathrm{mm}^2}.$$

Das Minuszeichen bedeutet **Druck** (der Träger drückt nach außen gegen die Auflager). Und hier kommt die Pointe dieses Kapitels: $-252\ \mathrm{N/mm^2}$ liegt **über der Streckgrenze von S235** ($235\ \mathrm{N/mm^2}$) — der Spannung, ab der Baustahl sich bleibend verformt statt elastisch zurückzufedern. Der beidseitig eingespannte, um 100 K erwärmte Träger *überschreitet* diese Grenze also bereits: Er **fließt**, noch bevor der Brand richtig in Fahrt kommt. Was das für unsere lineare Rechnung bedeutet — und warum echte Brandbemessung deshalb ganz anders vorgeht — klären wir ehrlich im Kleingedruckten am Kapitelende.

13.6. Schritt 6 — FEM-Übersetzung: die thermische Kraft

Wir haben alles, um die FEM aufzustellen. Die linke Seite ist geschenkt: Aus der Analogie-Tafel wissen wir, dass die Elementmatrix eines Stab-Elements dieselbe ist wie die der Federkette, nur mit $k = E\,A/h$:

$$\mathbf{K}_{\text{Element}} \;=\; \frac{E\,A}{h}\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}.$$

Bleibt die **rechte Seite** — und die ist der einzige wirklich neue Baustein des Kapitels. Wie kommt die Wärmedehnung ins Gleichungssystem? Über die Geschichte aus Schritt 5: Ein erwärmtes Element *will* sich um $\alpha\,\Delta T$ dehnen. Halten wir es (gedanklich) zunächst auf Länge, entsteht darin die Druckspannung $\sigma = -E\,\alpha\,\Delta T$, und diese Spannung drückt mit der Kraft $F = \sigma\,A = -E\,A\,\alpha\,\Delta T$ auf die Enden. Um das Element *wirklich* frei zu

lassen, müssen wir diese Haltekraft wieder wegnehmen — wir bringen also am Element ein Kraftpaar auf, das seine Enden **auseinander** drückt:

$$\mathbf{f}_{\text{Element}} \;=\; E\,A\,\alpha\,\Delta T\begin{pmatrix}-1\\+1\end{pmatrix} \qquad \text{(links nach außen, rechts nach außen).}$$

Das ist die **thermische Kraft**. ΔT ist dabei die mittlere Übertemperatur des Elements gegenüber der spannungsfreien Referenz. Rechnen wir sie für ein Element von Hand aus — nehmen wir das erste Element des linearen Profils $20 \to 300$ °C aus Kapitel 7. Seine beiden Knoten liegen bei 20 °C und 90 °C, im Mittel also 55 °C, das sind $\Delta T = 35$ K über der Referenz 20 °C:

$$F \;=\; E\,A\,\alpha\,\Delta T \;=\; 210\cdot 10^{9}\cdot 0{,}01\cdot 12\cdot 10^{-6}\cdot 35 \;=\; 882\,000\ \text{N} \;=\; 882\ \text{kN}.$$

Dieses Kraftpaar (-882 kN am linken, $+882$ kN am rechten Knoten) trägt das Element in die rechte Seite ein. Bei mehreren Elementen assembliert man diese Kräfte genau wie in Kapitel 3 die Federkräfte: An einem inneren Knoten stoßen zwei Elemente zusammen, ihre thermischen Kräfte zeigen gegeneinander, und nur ihre **Differenz** bleibt stehen. Wo alle Elemente gleich warm sind, hebt sich im Inneren alles auf — nur an den freien Enden bleibt etwas übrig. Genau deshalb ist der gleichmäßig erwärmte freie Träger spannungsfrei.

> **Freiheitsgrad — jetzt fällig**
>
> In Kapitel 3 haben wir das Wort noch vertagt; hier wird es gebraucht. Ein **Freiheitsgrad** ist eine einzelne unbekannte Zahl, die das Gleichungssystem bestimmen muss — eine „Stellschraube" der Lösung. Beim 1D-Träger hat jeder Knoten **einen** Freiheitsgrad: seine Verschiebung u in Längsrichtung (bei der Wärme war es die Temperatur). Fünf Knoten, fünf Freiheitsgrade, ein 5×5-System. Der Begriff wird ab Kapitel 14 wichtig, denn dort bekommt jeder Knoten in der Fläche **zwei** Freiheitsgrade — eine Verschiebung nach rechts *und* eine nach oben —, und die Systeme wachsen entsprechend.

Die vier FEM-Bausteine dieses Kapitels — Matrix bauen, thermische Kräfte, lösen mit Lagerung, Spannungen zurückrechnen — stecken im gemeinsamen Modul `programme/gemeinsam/stab.py`, das die Kapitel 14 und 15 erweitern werden. Hier ist das Herzstück, die Übersetzung der Element-Übertemperaturen in Knotenkräfte, als reines Python:

```
def thermische_kraefte(e_modul, flaeche, ausdehnung, delta_t_elemente):
    """Uebersetzt die Element-Uebertemperaturen in Knotenkraefte.

    e_modul: Elastizitaetsmodul E (Pascal).
    flaeche: Querschnittsflaeche A (Quadratmeter).
    ausdehnung: Waermeausdehnungskoeffizient alpha (1/Kelvin).
    delta_t_elemente: Uebertemperatur je Element (Kelvin), gegenueber der
        spannungsfreien Referenz. Ein Wert pro Element.
    Rueckgabe: Liste der Knotenkraefte (Newton) -- die rechte Seite.

    Jedes erwaermte Element drueckt seine beiden Knoten mit E*A*alpha*dT
    auseinander: den linken nach links (minus), den rechten nach rechts (plus).
    """
    knotenzahl = len(delta_t_elemente) + 1
    kraefte = []
    for knoten in range(knotenzahl):
        kraefte.append(0.0)

    for element in range(len(delta_t_elemente)):
        kraft = e_modul * flaeche * ausdehnung * delta_t_elemente[element]
        kraefte[element] = kraefte[element] - kraft
        kraefte[element + 1] = kraefte[element + 1] + kraft
    return kraefte


# Ein Element von Hand nachrechnen: erstes Element des linearen Profils.
E_MODUL = 210e9
FLAECHE = 0.01
AUSDEHNUNG = 12e-6
kraefte = thermische_kraefte(E_MODUL, FLAECHE, AUSDEHNUNG, [35.0])
print("thermisches Kraftpaar bei dT = 35 K:")
print("  linker Knoten :", kraefte[0] / 1000.0, "kN")
print("  rechter Knoten:", kraefte[1] / 1000.0, "kN")
```

```
thermisches Kraftpaar bei dT = 35 K:
  linker Knoten : -882.0 kN
  rechter Knoten: 882.0 kN
```

Interpretation der Ausgabe: Genau die ± 882 kN von Hand. Der linke Knoten wird nach links gedrückt, der rechte nach rechts — das Element stemmt sich auseinander. Diese Funktion ist die einzige Zeile Physik, die die Mechanik über die Wärme hinaus braucht; alles andere ist die alte Federketten-Maschinerie.

13.7. Schritt 7 — Der gekoppelte Träger: das Temperaturfeld als Last

Jetzt fügen wir alles zusammen und rechnen den **gekoppelten** Fall: Das Temperaturfeld aus Kapitel 7 und Kapitel 9 wird zur Last für die Mechanik. Das ist die **Einweg-Kopplung**, die erste echte Multiphysik des Buches. Sie läuft als Fließband in eine Richtung:

Abbildung 13.5.: Die Einweg-Kopplung als Fließband. Das Temperaturfeld (aus Kapitel 7/9) wird elementweise gemittelt, daraus werden die thermischen Kräfte, die ins Stab-Gleichungssystem gehen; dessen Lösung sind die Verschiebungen, aus denen man Dehnung und Spannung zurückrechnet. Der Pfeil zeigt nur nach rechts: Die Temperatur treibt die Verformung, aber die Verformung wirkt nicht auf die Temperatur zurück (siehe „Einbahnstraße").

Was man hier sieht: die Rechenkette von links nach rechts. Wir starten mit dem bekannten Temperaturprofil, mitteln es je Element zur Übertemperatur, machen daraus die thermischen Kräfte, lösen das Stab-System und rechnen aus den Verschiebungen Dehnung und Spannung zurück. Nehmen wir das lineare Profil $20 \;\to\; 300$ °C mit den fünf Knotenwerten $20, 90, 160, 230, 300$ °C. Die vier Element-Mittel, jeweils minus 20 °C Referenz, sind

$$\Delta T_1 = 35, \quad \Delta T_2 = 105, \quad \Delta T_3 = 175, \quad \Delta T_4 = 245 \text{ K}.$$

Damit rechnen wir zwei Lagerungen durch — links fest / rechts frei und beidseitig fest — und lesen aus den Verschiebungen die Spannung ab. Der folgende Lauf benutzt die Bausteine aus `stab.py`:

```
import os
import sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from stab import (baue_stab_matrix, thermische_kraefte, loese_mit_lagerung,
```

```
                  spannungen_zurueckrechnen)

E_MODUL = 210e9
FLAECHE = 0.01
AUSDEHNUNG = 12e-6
KNOTEN_X = [0.0, 0.25, 0.5, 0.75, 1.0]
DELTA_T = [35.0, 105.0, 175.0, 245.0]

matrix = baue_stab_matrix(E_MODUL, FLAECHE, KNOTEN_X)
kraefte = thermische_kraefte(E_MODUL, FLAECHE, AUSDEHNUNG, DELTA_T)

# Links fest, rechts frei.
u_frei = loese_mit_lagerung(matrix, kraefte, {0: 0.0})
sig_frei = spannungen_zurueckrechnen(u_frei, KNOTEN_X, E_MODUL, AUSDEHNUNG, DELTA_T)

# Beidseitig fest.
u_fest = loese_mit_lagerung(matrix, kraefte, {0: 0.0, 4: 0.0})
sig_fest = spannungen_zurueckrechnen(u_fest, KNOTEN_X, E_MODUL, AUSDEHNUNG, DELTA_T)

def als_mm(werte_in_meter):
    """Liste von Meter-Werten als mm-Text (drei Nachkommastellen)."""
    texte = []
    for wert in werte_in_meter:
        texte.append("%.3f" % (wert * 1000.0))
    return texte


def als_nmm2(werte_in_pascal):
    """Liste von Pascal-Werten als N/mm2-Text (1 N/mm2 = 1e6 Pa)."""
    texte = []
    for wert in werte_in_pascal:
        texte.append("%.1f" % (wert / 1e6))
    return texte


print("links fest / rechts frei:")
print("  u (mm)      :", als_mm(u_frei))
print("  sigma(N/mm2):", als_nmm2(sig_frei))
print("beidseitig fest:")
print("  u (mm)      :", als_mm(u_fest))
print("  sigma(N/mm2):", als_nmm2(sig_fest))
```

```
links fest / rechts frei:
```

```
  u (mm)      : ['0.000', '0.105', '0.420', '0.945', '1.680']
  sigma(N/mm2): ['0.0', '0.0', '0.0', '0.0']
beidseitig fest:
  u (mm)      : ['0.000', '-0.315', '-0.420', '-0.315', '0.000']
  sigma(N/mm2): ['-352.8', '-352.8', '-352.8', '-352.8']
```

Interpretation der Ausgabe — hier steckt der Aha-Moment. Zwei Dinge fallen auf:

- **Frei gelagert ist die Spannung überall null**, obwohl der Träger höchst ungleichmäßig erwärmt ist (von 20 auf 300 °C)! Die Verschiebung wächst kumulativ an — jedes Element legt seine eigene Wärmedehnung obendrauf, bis das freie Ende bei $1{,}68$ mm steht. Aber weil sich jedes Element frei dehnen darf, ist nirgends Dehnung *verhindert* — also nirgends Spannung. Das ist die überraschende Lehre: **Auch eine ungleichmäßige Erwärmung erzeugt in einem frei gelagerten 1D-Träger keine Spannung.** (In 2D wird das anders — dort können sich verschieden heiße Fasern gegenseitig festhalten; das ist der Stoff von Kapitel 14.)
- **Beidseitig fest ist die Spannung überall gleich**, nämlich $-352{,}8\ \mathrm{N/mm^2}$ — und zwar *konstant* über den ganzen Träger, obwohl das linke Element nur um 35 K, das rechte um 245 K erwärmt ist. Das ist verblüffend, aber zwingend: In einem Stab ohne Längslasten dazwischen muss die **Längskraft überall gleich** sein (was am linken Ende hineingeht, muss am rechten wieder heraus — dieselbe „rein = raus"-Bilanz wie bei der Wärme in Kapitel 6). Konstante Kraft bei konstantem Querschnitt heißt konstante Spannung. Der Zahlenwert ist $\sigma = -E\,\alpha\,\Delta T_{\mathrm{mittel}}$ mit dem Träger-Mittel $\Delta T_{\mathrm{mittel}} = (35 + 105 + 175 + 245)/4 = 140$ K, also $-210\,000 \cdot 12 \cdot 10^{-6} \cdot 140 = -352{,}8\ \mathrm{N/mm^2}$. (Auch dieser Wert liegt weit über der Streckgrenze — die Rechnung ist eine obere Abschätzung, siehe Kleingedrucktes.)

Sehen wir uns das im Zusammenhang an: drei Felder übereinander — Temperatur, Verschiebung, Spannung.

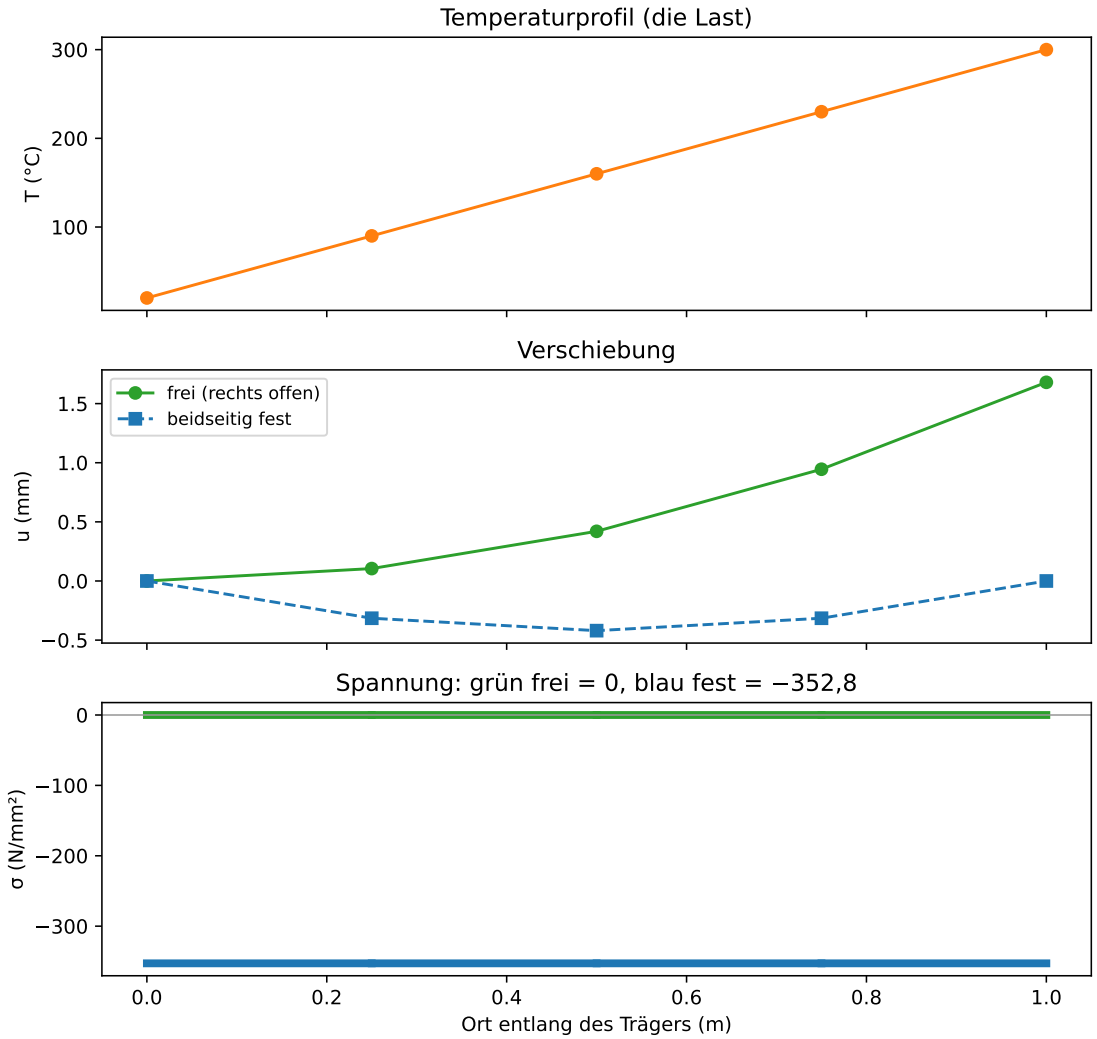

Abbildung 13.6.: Zum ersten Mal drei Felder im Zusammenhang. Oben das Temperaturprofil (die Last, für beide Fälle gleich). Mitte die Verschiebung: frei (grün) wächst sie kumulativ bis 1,68 mm; fest (blau) bleibt sie an beiden Enden null, im Inneren weicht der Träger leicht aus (bis −0,42 mm). Unten die Spannung: frei überall null (grün, auf der Nulllinie), fest überall konstant −352,8 N/mm² (blau). Dieselbe Erwärmung, zwei Lagerungen, gegensätzliche Antworten — das ist die ganze Botschaft des Kapitels.

Was man hier sieht: die Kopplung auf einen Blick. Das obere Feld (Temperatur) ist die Eingabe, identisch für beide Fälle. Das mittlere Feld zeigt, wie verschieden die beiden Lagerungen antworten: Der freie Träger (grün)

wächst kumulativ, der feste (blau) kann nirgends wachsen und weicht stattdessen im Inneren leicht aus. Das untere Feld ist die Pointe: Die Spannung des freien Trägers liegt platt auf der Null, die des festen liegt platt auf $-352{,}8$ — beide konstant, aber aus entgegengesetzten Gründen.

13.8. Schritt 8 — Der Nulltest, gebaut

In Kapitel 12 haben wir einen **Nulltest für die Thermomechanik** *entworfen* (Übung 12.3): Ein frei gelagerter, gleichmäßig erwärmter Träger muss Spannung **exakt null** liefern, auf jedem Netz, weil seine wahre Lösung im Ansatzraum liegt und keinen Diskretisierungsfehler zulässt. Jetzt, mit der fertigen Mechanik, **bauen** wir ihn — der erste Anwendungsfall unserer Prüfkultur auf die neue Physik. Das Protokoll ist genau das aus Übung 12.3: frei lagern, gleichmäßig ΔT aufbringen, Spannung an allen Elementen ablesen, Erwartung null.

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from stab import (baue_stab_matrix, thermische_kraefte, loese_mit_lagerung,
                  spannungen_zurueckrechnen)

E_MODUL = 210e9
FLAECHE = 0.01
AUSDEHNUNG = 12e-6
KNOTEN_X = [0.0, 0.25, 0.5, 0.75, 1.0]
DELTA_T_GLEICH = [100.0, 100.0, 100.0, 100.0]   # ueberall +100 K, frei

matrix = baue_stab_matrix(E_MODUL, FLAECHE, KNOTEN_X)
kraefte = thermische_kraefte(E_MODUL, FLAECHE, AUSDEHNUNG, DELTA_T_GLEICH)
u = loese_mit_lagerung(matrix, kraefte, {0: 0.0})
sigma = spannungen_zurueckrechnen(u, KNOTEN_X, E_MODUL, AUSDEHNUNG, DELTA_T_GLEICH)

groesste_spannung = 0.0
for wert in sigma:
    if abs(wert) > groesste_spannung:
        groesste_spannung = abs(wert)

print("NULLTEST -- frei gelagert, gleichmaessig +100 K")
print("-----------------------------------------------")
print("freies Ende u:      %.4f mm" % (u[-1] * 1000.0))
print("erwartete Dehnung:  %.4f mm  (alpha*dT*L)" % (AUSDEHNUNG * 100 * 1000))
print("groesste |sigma|:   %.3e N/mm2" % (groesste_spannung / 1e6))
```

```
if groesste_spannung / 1e6 < 1e-6:
    print("Ergebnis: BESTANDEN -- Spannung null auf Maschinengenauigkeit.")
else:
    print("Ergebnis: DURCHGEFALLEN -- da steckt ein Fehler.")
```

```
NULLTEST -- frei gelagert, gleichmaessig +100 K
------------------------------------------------
freies Ende u:      1.2000 mm
erwartete Dehnung:  1.2000 mm  (alpha*dT*L)
groesste |sigma|:   9.107e-14 N/mm2
Ergebnis: BESTANDEN -- Spannung null auf Maschinengenauigkeit.
```

Interpretation der Ausgabe: Das freie Ende steht bei $1{,}2$ mm (genau $\alpha\,\Delta T\,L$), und die größte Spannung im ganzen Träger ist null — auf Maschinengenauigkeit. Was stehen bleibt, ist ein winziger Rechenrest von der Größe 10^{-13} N/mm^2, milliardenfach kleiner als jede echte Spannung; er liegt weit unter der Bestehensschwelle und zählt als sauberes Null. Damit ist die Mechanik verifiziert: Baute man versehentlich eine zu starre Lagerung ein, ein falsches Vorzeichen in der Kopplung oder ein falsches α, würde hier *garantiert* eine Spannung ungleich null auftauchen. Der Nulltest ist der erste Griff, mit dem man ein neues Thermomechanik-Programm prüft — bevor man ihm irgendeine bunte Verformung glaubt.

13.9. Zwei Filme: Aufheizen und der Grenzfall-Umschalter

Der stationäre Fall ist gerechnet — jetzt lassen wir ihn laufen. Der erste Film nimmt die wandernde Wärmefront aus Kapitel 9 und heftet ihr die Verformung an: Während die Wärme vom Brandherd rechts ins Metall kriecht, wächst das freie Ende live mit.

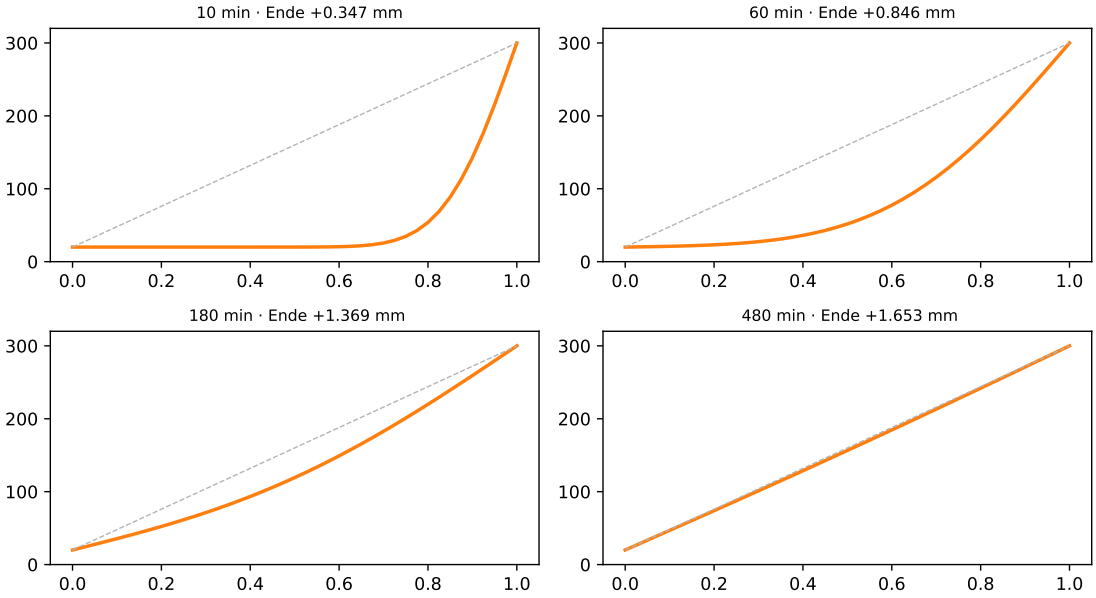

Abbildung 13.7.: Der Aufheiz-Film als Bildreihe (in der HTML-Fassung eine Animation): vier Momentaufnahmen. Die Wärmefront (oben) schiebt sich nach links, das freie Ende (Trägerlänge unten, überhöht) wächst mit, bis der eingeschwungene Zustand erreicht ist.

Was man sieht: Die Front kriecht nach links (Kapitel 9), und das freie Ende wächst genau in dem Maß, wie sich die *gesamte* aufgenommene Wärme im Träger summiert. Am Anfang ist nur die rechte Spitze warm, das Ende wächst kaum; je weiter die Front vordringt, desto länger wird der Träger, bis der stationäre Zustand erreicht ist. Wichtig: Der Überhöhungsfaktor ist eingeblendet — die echte Dehnung ist Bruchteile eines Millimeters, sonst sähe man nichts.

Der zweite Film stellt die beiden Grenzfälle direkt gegeneinander: dieselbe gleichmäßige Erwärmung, einmal frei, einmal fest. Man sieht, wie die beiden Träger die Rollen tauschen — der eine dehnt sich (und bleibt spannungsfrei), der andere bleibt starr (und lädt sich mit Druckspannung auf).

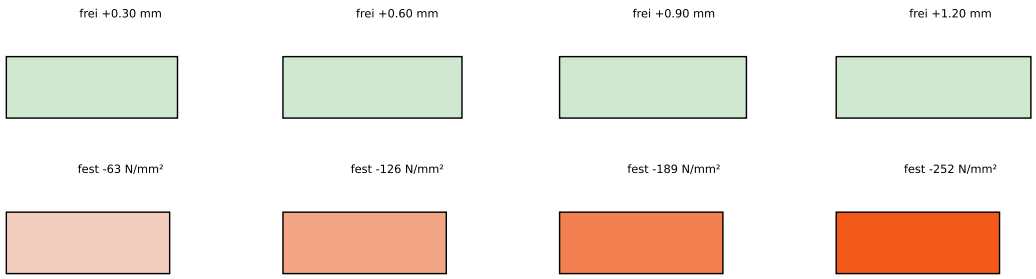

Abbildung 13.8.: Der Grenzfall-Umschalter als Bildreihe (in der HTML-Fassung eine Animation): die Erwärmung steigt in vier Stufen. Oben der freie Träger (wird länger, grün, $\sigma = 0$), unten der feste (bleibt auf Länge, färbt sich rot, σ bis -252 N/mm²).

Was man sieht: die beiden Grenzfälle als Bewegung. Oben wächst der freie Träger sichtbar (überhöht), bleibt aber blass — keine Spannung. Unten rührt sich der feste Träger nicht, färbt sich aber mit steigender Temperatur immer tiefer rot: Die verhinderte Dehnung staut sich als Druckspannung, bis bei 100 K die -252 N/mm^2 erreicht sind. Dehnungsträger und Spannungsträger — zwei Antworten auf dieselbe Wärme, je nach Lagerung.

13.10. Die interaktive Einheit: die Dehnbank

Jetzt bist du dran. Die **Dehnbank** rechnet den Träger für jede Kombination aus Lagerung, Temperaturprofil und Material und zeigt den Dreifachplot $T(x)$, $u(x)$, $\sigma(x)$. Bevor du sie laufen lässt, eine Vorhersage:

> Vorhersage-Punkt
>
> **Bevor du das Programm startest:** Der Träger ist **beidseitig fest** eingespannt und wird **gleichmäßig um 100 K** erwärmt. Wo ist die Druckspannung am größten — in der **Mitte**, an den **Rändern**, oder **überall gleich**? Leg dich fest, bevor du weiterliest.

```
#| autorun: true
# Die Dehnbank -- probiere die drei Regler aus:
#   LAGERUNG = "frei" / "einseitig" / "beidseitig"
#   PROFIL   = "gleichmaessig" / "linear" / "momentaufnahme"
#   MATERIAL = "stahl" / "alu"
```

```
LAGERUNG = "beidseitig"
PROFIL = "gleichmaessig"
MATERIAL = "stahl"

import math

FLAECHE = 0.01
KNOTEN_X = [0.0, 0.25, 0.5, 0.75, 1.0]
REFERENZ = 20.0
STOFFE = {"stahl": (210e9, 12e-6), "alu": (70e9, 23e-6)}


def knotentemperaturen(profil):
    """Temperatur an den fuenf Knoten (Grad C) fuer eines der drei Profile."""
    werte = []
    if profil == "gleichmaessig":
        for k in range(len(KNOTEN_X)):
            werte.append(REFERENZ + 100.0)
    elif profil == "linear":
        for k in range(len(KNOTEN_X)):
            werte.append(REFERENZ + 280.0 * KNOTEN_X[k] / KNOTEN_X[-1])
    else:  # momentaufnahme: transiente Fuehler-Kurve nach 30 min (Kapitel 5/9)
        a = 50.0 / (7850.0 * 460.0)
        for k in range(len(KNOTEN_X)):
            x = KNOTEN_X[k]
            wert = REFERENZ + 280.0 * x / KNOTEN_X[-1]
            for n in range(1, 40):
                koeff = 560.0 * ((-1.0) ** n) / (n * math.pi)
                abkling = math.exp(-((n * math.pi) ** 2) * a * 1800.0)
                wert = wert + koeff * math.sin(n * math.pi * x) * abkling
            werte.append(wert)
    return werte


def element_delta_t(profil):
    """Uebertemperatur je Element = Mittel der Knoten minus Referenz (K)."""
    knoten = knotentemperaturen(profil)
    werte = []
    for e in range(len(KNOTEN_X) - 1):
        werte.append((knoten[e] + knoten[e + 1]) / 2.0 - REFERENZ)
    return werte
```

```
def rechne(lagerung, profil, material):
    """Loest den Lastfall und gibt Verschiebungen (m) und Spannungen (Pa)."""
    e_modul, alpha = STOFFE[material]
    delta_t = element_delta_t(profil)
    n = len(KNOTEN_X)
    # Systemmatrix (Federkette mit k = E*A/h) und thermische Kraefte.
    matrix = []
    for i in range(n):
        zeile = []
        for j in range(n):
            zeile.append(0.0)
        matrix.append(zeile)
    kraefte = []
    for i in range(n):
        kraefte.append(0.0)
    for e in range(n - 1):
        h = KNOTEN_X[e + 1] - KNOTEN_X[e]
        k = e_modul * FLAECHE / h
        matrix[e][e] += k
        matrix[e][e + 1] -= k
        matrix[e + 1][e] -= k
        matrix[e + 1][e + 1] += k
        kraft = e_modul * FLAECHE * alpha * delta_t[e]
        kraefte[e] -= kraft
        kraefte[e + 1] += kraft
    # Lager festlegen.
    if lagerung == "beidseitig":
        lager = {0: 0.0, n - 1: 0.0}
    else:
        lager = {0: 0.0}
    # Reduziertes System der freien Knoten loesen (kleine Gauss-Elimination).
    frei = []
    for i in range(n):
        if i not in lager:
            frei.append(i)
    red = []
    for i in frei:
        zeile = []
        for j in frei:
            zeile.append(matrix[i][j])
        red.append(zeile)
    rs = []
    for i in frei:
```

```
        s = kraefte[i]
        for j in lager:
            s = s - matrix[i][j] * lager[j]
        rs.append(s)
    xf = loese_gauss(red, rs)
    u = []
    for i in range(n):
        u.append(0.0)
    for pos in range(len(frei)):
        u[frei[pos]] = xf[pos]
    for i in lager:
        u[i] = lager[i]
    sigma = []
    for e in range(n - 1):
        h = KNOTEN_X[e + 1] - KNOTEN_X[e]
        eps = (u[e + 1] - u[e]) / h
        sigma.append(e_modul * (eps - alpha * delta_t[e]))
    return u, sigma


def loese_gauss(matrix, seite):
    """Kleine Gauss-Elimination fuer das reduzierte System."""
    n = len(matrix)
    werk = []
    for zeile in matrix:
        werk.append(list(zeile))
    s = list(seite)
    for sp in range(n):
        piv = werk[sp][sp]
        for z in range(sp + 1, n):
            f = werk[z][sp] / piv
            for c in range(sp, n):
                werk[z][c] = werk[z][c] - f * werk[sp][c]
            s[z] = s[z] - f * s[sp]
    x = []
    for i in range(n):
        x.append(0.0)
    for z in range(n - 1, -1, -1):
        summe = s[z]
        for c in range(z + 1, n):
            summe = summe - werk[z][c] * x[c]
        x[z] = summe / werk[z][z]
    return x
```

```
temp = knotentemperaturen(PROFIL)
u, sigma = rechne(LAGERUNG, PROFIL, MATERIAL)
print("Lagerung=%s  Profil=%s  Material=%s" % (LAGERUNG, PROFIL, MATERIAL))
u_texte = []
for wert in u:
    u_texte.append("%.3f" % (wert * 1000.0))
sig_texte = []
for wert in sigma:
    sig_texte.append("%.1f" % (wert / 1e6))
print("Knoten-u (mm) :", u_texte)
print("Element-sigma :", sig_texte, "N/mm2")

# --- Abbildung ---
import matplotlib.pyplot as plt
fig, ax = plt.subplots(3, 1, figsize=(7.5, 7.0), sharex=True)
ax[0].plot(KNOTEN_X, temp, "o-", color="tab:orange")
ax[0].set_ylabel("T (°C)"); ax[0].set_title("Temperaturprofil")
ax[1].plot(KNOTEN_X, [x * 1000.0 for x in u], "o-", color="tab:blue")
ax[1].set_ylabel("u (mm)"); ax[1].set_title("Verschiebung")
for e in range(len(KNOTEN_X) - 1):
    ax[2].plot([KNOTEN_X[e], KNOTEN_X[e + 1]], [sigma[e] / 1e6] * 2,
               color="tab:red", lw=4)
ax[2].axhline(0, color="0.6", lw=0.8)
ax[2].set_ylabel("σ (N/mm²)"); ax[2].set_title("Spannung")
ax[2].set_xlabel("Ort (m)")
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt

KNOTEN_X = [0.0, 0.25, 0.5, 0.75, 1.0]
temp = [120.0] * 5
u = [0.0] * 5
sigma = [-252.0] * 4

fig, ax = plt.subplots(3, 1, figsize=(7.5, 7.0), sharex=True)
ax[0].plot(KNOTEN_X, temp, "o-", color="tab:orange")
ax[0].set_ylabel("T (°C)"); ax[0].set_title("Temperaturprofil")
ax[1].plot(KNOTEN_X, u, "o-", color="tab:blue")
ax[1].set_ylabel("u (mm)"); ax[1].set_title("Verschiebung")
for e in range(4):
```

```
    ax[2].plot([KNOTEN_X[e], KNOTEN_X[e + 1]], [sigma[e]] * 2,
               color="tab:red", lw=4)
ax[2].axhline(0, color="0.6", lw=0.8)
ax[2].set_ylabel("σ (N/mm²)"); ax[2].set_title("Spannung")
ax[2].set_xlabel("Ort (m)")
plt.tight_layout()
plt.show()
```

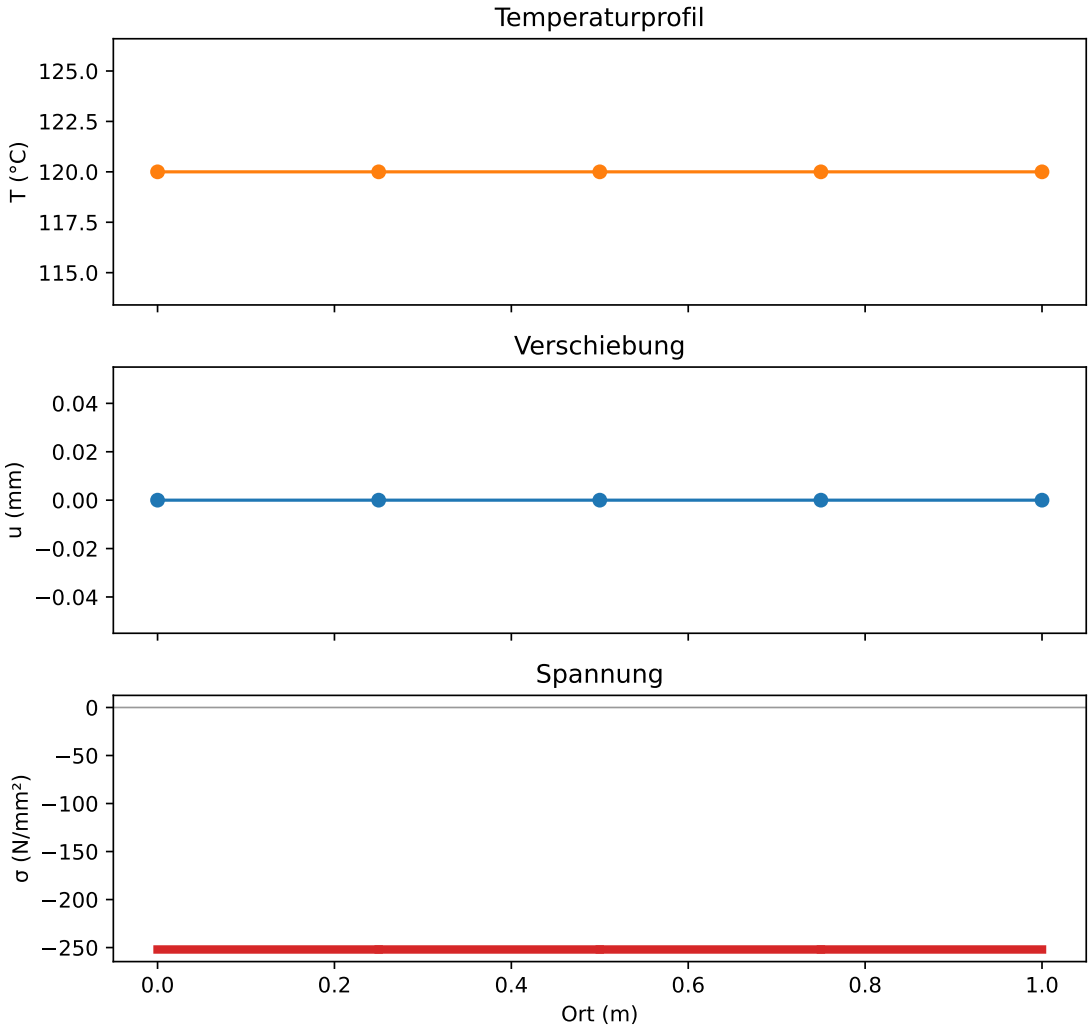

Abbildung 13.9.: Vorgerenderte Fassung der Dehnbank (beidseitig fest, gleichmäßig +100 K, Stahl): Das Temperaturprofil ist flach bei 120 °C, die Verschiebung ist an beiden Enden null (und dazwischen ebenfalls, weil gleichmäßig warm), die Spannung ist überall konstant −252 N/mm². Im Browser wird dieses Bild durch dein Ergebnis ersetzt.

Auflösung des Vorhersage-Punkts: Die Druckspannung ist **überall gleich** — konstant $-252\ \mathrm{N/mm^2}$ über den ganzen Träger, kein Maximum in der Mitte, keine Spitze am Rand. Das ist derselbe Grund wie beim linearen Profil: Die Längskraft muss im Stab überall gleich sein, also auch die Spannung. Viele tippen auf „Mitte" oder „Rand" — die Wärme fühlt sich lokal an. Aber die Spannung verteilt sich völlig gleichmäßig.

Probiere jetzt die anderen Regler: Schalte auf `"frei"` (Spannung fällt auf null), tausche `"stahl"` gegen `"alu"` (die Spannung *fällt* auf $-161\ \mathrm{N/mm^2}$, obwohl Aluminiums α größer ist — warum, klärt Übung 13.2) oder wähle das `"momentaufnahme"`-Profil, die transiente Fühler-Kurve nach 30 Minuten.

13.11. Das Kapitel-Programm

Das vollständige, eigenständige Programm zu diesem Kapitel ist `programme/kap13/kap13_d`
Es bündelt die Bausteine aus `programme/gemeinsam/stab.py` (Matrix, thermische Kräfte, Lösen mit Lagerung, Spannungen zurückrechnen), rechnet den gekoppelten 5-Knoten-Fall in beiden Lagerungen, führt das **Nulltest-Protokoll** aus und speichert den Dreifachplot. Die interaktive Dehnbank oben ist seine im Browser lauffähige Kurzfassung.

Merkkasten

- **Dehnung** $\varepsilon = \Delta L/L$ ist dimensionslos; **Spannung** $\sigma = F/A$ misst man in N/mm² (= MPa), den **E-Modul** in GPa ($1\ \mathrm{GPa} = 1000\ \mathrm{N/mm^2}$).
- **Hooke:** $\sigma = E\,\varepsilon$ — die Feder aus Kapitel 3, pro Material. Die Stab-Steifigkeit $k = E\,A/h$ hat *dieselbe Formelgestalt* wie die Wärme-Feder $\lambda\,A/h$ (Kapitel 7).
- **Kernformel:** $\sigma = E\,(\varepsilon - \alpha\,\Delta T)$ — Spannung entsteht nur aus der **verhinderten** Dehnung, nicht aus der Wärme an sich.
- **Grenzfälle:** frei → $\sigma = 0$ (der Träger dehnt sich, auch ungleichmäßig erwärmt); beidseitig fest → $\sigma = -E\,\alpha\,\Delta T_{\mathrm{mittel}}$, konstant über den ganzen Träger.
- **FEM:** dieselbe Matrix wie die Federkette, neue rechte Seite aus thermischen Kräften $\pm E\,A\,\alpha\,\Delta T$. Der **Nulltest** (frei + gleichmäßig warm → $\sigma = 0$) prüft das Programm.

Roter Faden

Wo kam das schon vor, wo kommt es wieder? — **Zurück:** Die Federkette (Kapitel 3) ist endlich wirklich Mechanik geworden; ihre Steifigkeit k ist jetzt $E\,A/h$. Die Elementrechnung von Kapitel 7 lieferte dieselbe Matrixgestalt für die Wärme; die Analogie-Tafel führt beide zusammen. Das Temperaturfeld aus Kapitel 7 und Kapitel 9 wird hier zur Last. Der Nulltest aus Kapitel 12 (dort in Übung 12.3 entworfen) ist jetzt gebaut und bestanden. — **Nach vorn:** Kapitel 14 hebt alles in die Fläche — dort verbiegt sich der

Träger endlich (die zweite Dimension macht aus dem Längenunterschied eine Krümmung, siehe Übung 13.3), und die Analogie-Tafel wächst um die Querkontraktion. 3 baut daraus das interaktive Finale.

Übungen

Ü 13.1 (Verstehen). Eine Bahnschiene aus Stahl ($E = 210$ GPa, $\alpha = 12 \cdot 10^{-6}\ \frac{1}{\mathrm{K}}$) wird im Sommer um $\Delta T = 40$ K wärmer als bei der Verlegung. Sie ist lückenlos verschweißt und kann sich nicht ausdehnen. Welche Spannung entsteht? Ist es Zug oder Druck? Rechne ohne Computer.

> **Musterlösung zu Ü 13.1**
>
> Beidseitig gehindert, also der Grenzfall B: $\varepsilon = 0$ und
>
> $$\sigma = -E\,\alpha\,\Delta T = -210\,000\ \tfrac{\mathrm{N}}{\mathrm{mm}^2} \cdot 12 \cdot 10^{-6}\ \tfrac{1}{\mathrm{K}} \cdot 40\ \mathrm{K} = -100{,}8\ \tfrac{\mathrm{N}}{\mathrm{mm}^2}.$$
>
> Rund $-101\ \mathrm{N/mm^2}$ **Druck** (Minuszeichen). Zum Vergleich: Die Streckgrenze von Schienenstahl liegt bei einigen hundert $\mathrm{N/mm^2}$, die Schiene fließt bei diesen 40 K also nicht — aber ein schlanker Druckstab kann *ausknicken*, und genau das ist die Gefahr: die **Gleisverwerfung**, bei der sich das erhitzte Gleis seitlich ausbeult. Deshalb verschweißt man Schienen bei einer festgelegten **Neutraltemperatur** (bei europäischen Bahnen meist etwa 20–27 °C) und hält sie mit einem massiven **Schotterbett** und schweren Schwellen fest, damit die Druckkräfte nicht zum seitlichen Ausweichen führen (VDI / ingenieur.de 2023). Frühere Stoßlücken waren die einfachere Antwort: Man ließ die Schiene sich frei dehnen — Grenzfall A, Spannung null, dafür das rhythmische „Tackern" der Räder über die Lücken.

Ü 13.2 (Verändern). Ersetze in der Dehnbank Stahl durch Aluminium (`MATERIAL = "alu"`), bei gleichmäßiger Erwärmung um 100 K. Was ändert sich am freien Ende (Verschiebung), was an der Spannung des beidseitig festen Trägers? Aluminium hat ein *größeres* α, aber ein *kleineres* E als Stahl — wer gewinnt bei der Einspann-Spannung? Lösungsskript: `loesungen/kap13_ue2.py`.

Musterlösung zu Ü 13.2

E und α wirken **gegeneinander**. Freies Ende: Aluminium dehnt sich um $\Delta L = \alpha\,\Delta T\,L = 23 \cdot 10^{-6} \cdot 100 \cdot 1 = 2{,}3$ mm — fast doppelt so viel wie Stahl (1,2 mm), weil α größer ist. Beidseitig fest aber steht Aluminium unter *niedrigerer* Druckspannung:

$$\sigma_{Alu} = -E\alpha\Delta T = -70\,000 \cdot 23 \cdot 10^{-6} \cdot 100 = -161\ \frac{\text{N}}{\text{mm}^2}$$

gegen -252 N/mm² bei Stahl. Der Grund: Für die Einspann-Spannung zählt nur das Produkt $E\alpha$ [illegible] ist mit -161 N/mm² übrigens [illegible] am unteren Rand der Streckgrenzen gängiger Konstruktionslegierungen (grob 160 bis 240 N/mm²) (Aluminium-Werkstoffdaten 2024), während der Stahl seine Streckgrenze schon klar überschritten hat.

Musterlösung zu Ü 13.3

In dieser **Reihenschaltung: nein**, die Spannung ist in beiden Stücken null. Weil das rechte Ende frei ist, kann sich jedes Element ungehindert dehnen: das Alu-Stück [illegible] ($2300\ \mu\varepsilon$), das Stahl-Stück weniger ($1200\ \mu\varepsilon$) [illegible] **Die Verschiebungen** unterscheiden sich [illegible] **Spannungen** [illegible] Bimetall-Effekt [illegible] entsteht erst, wenn die beiden Materialien **nebeneinander** liegen. Dann zwingt der Längenunterschied den Verbund zur Biegung, und *das* erzeugt Spannung. Das braucht die zweite Dimension und ist der Cliffhanger auf Kapitel 14.

Ü 13.3 (Übertragen) Ein Bimetall-Vorbote. Verschweiße zwei gleich lange Elemente hintereinander zu einem Stab — das linke aus Aluminium, das rechte aus Stahl — und erwärme beide gleichmäßig um 100 K, links fest, rechts frei. Baue Matrix und thermische Kräfte mit *elementweisem* Material (jedes Element sein eigenes E und α). Entsteht an der Naht zwischen Alu und Stahl eine Spannung? Lösungsskript: loesungen/kap13_ue3.py

Das Kleingedruckte

Zwei ehrliche Einschränkungen begleiten dieses Kapitel.

Einbahnstraße. Wir haben nur *eine* Richtung der Kopplung gerechnet: Die Temperatur erzeugt eine Verformung, aber die Verformung wirkt **nicht** auf die Temperatur zurück (die minimale Erwärmung durch das Stauchen selbst, die sogenannte thermoelastische Dämpfung, lassen wir weg). Das ist die bewusste Vereinfachung der **Einweg-Kopplung** aus der Buchstrategie: erst das Wärmeproblem lösen, dann sein Ergebnis als feste Last in die Mechanik reichen. Für unseren Träger ist der Rückkanal winzig und die Vereinfachung sauber; echte gekoppelte Probleme (die vollen thermo-hydro-mechanischen Prozesse) müssen beide Richtungen zugleich iterieren.

Ehrlich gerechnet: unser eingespannter Träger fließt schon

Unser ganzes Modell ist **linear elastisch**: Spannung und Dehnung bleiben proportional, und lässt die Last nach, federt der Träger exakt zurück. Das gilt nur bis zur **Streckgrenze** — bei Baustahl S235 sind das $235\ \mathrm{N/mm^2}$. Unsere $-252\ \mathrm{N/mm^2}$ beim beidseitig eingespannten, um 100 K erwärmten Träger liegen bereits *darüber*: Der Stahl beginnt zu **fließen**, verformt sich also bleibend, und $\sigma = E\,\varepsilon$ stimmt nicht mehr. Ab da ist unsere Zahl eine **obere Abschätzung** — in Wirklichkeit begrenzt das Fließen die Spannung ungefähr auf die Streckgrenze und verteilt die überzählige Last um.
Im Brandfall wird es noch ernster, aber nicht dramatischer, sondern gerade darum so sorgfältig geregelt: Stahl **verliert bei Hitze an Festigkeit**. Schon bei etwa 600 °C ist die Streckgrenze auf rund die Hälfte gefallen (dieselbe Kurve, die in Kapitel 1 die Brandschutzbeplankung begründet) — die Grenze, die wir eben überschritten haben, sinkt beim Aufheizen also selbst weiter ab. Genau deshalb rechnet die **Heißbemessung** von Tragwerken *nicht* linear elastisch, sondern mit temperaturabhängigen Festigkeiten: Der Eurocode 3, Teil 1-2 (EN 1993-1-2) (CEN (Europäisches Komitee für Normung) 2005) legt fest, wie stark die Streckgrenze bei jeder Temperatur abzumindern ist, und daraus folgen die Feuerwiderstandsklassen R30/R60/R90. Was jenseits der Streckgrenze passiert — **Plastizität** — ist eine eigene, nichtlineare Welt, die dieses Buch nicht mehr betritt. Wir bleiben im elastischen Bereich und merken uns ehrlich: Für den eingespannten Träger im Brand liefert unsere lineare Rechnung die *Warnung*, nicht die letzte Zahl.

Und schließlich, wie stets in Teil V: Der 1D-Stab kann nur **länger oder kürzer** werden — keine Biegung, keine Querkontraktion, keine seitliche Ausbeulung. Genau eine neue Größe nach der anderen. Die Fläche, in der sich

der Träger endlich *verbiegt,* beginnt im nächsten Kapitel.

14. Der Träger verbiegt sich: 2D-Thermomechanik

Am Ende von Kapitel 13 blieb eine Frage offen. Wir hatten zwei Stücke Metall hintereinandergeschweißt — links Aluminium, rechts Stahl — und beide erwärmt. Das Aluminium wollte sich weiter dehnen als der Stahl, aber weil sie *hintereinander* lagen, durfte jedes für sich wachsen: keine Spannung, nur verschieden weit gewanderte Knoten. Der Cliffhanger war: Was, wenn ein Längenunterschied nicht ausweichen darf, weil zwei verschieden lange Fasern **nebeneinander** liegen und untrennbar aneinanderhängen? Dann hält die eine die andere fest, der Überschuss an Länge kann nirgends hin — und der Verbund **krümmt sich**. Das ist das Bimetall, das jeden Thermostat, jeden alten Blinkgeber schaltet.

Genau diese Lage stellt der Brand unter unserem Träger her. Die Flammen und heißen Brandgase treffen die **Unterseite**; die Betondecke darüber hält die **Oberseite** kühl. So steht dauerhaft ein Temperaturgefälle über die Höhe: unten heiß, oben kalt. Der Träger ist damit ein *kontinuierliches* Bimetall aus einem einzigen Stück Stahl — die heiße untere Faser will länger sein als die kalte obere, beide sitzen untrennbar aneinander, und der ganze Träger wölbt sich zu einem Bogen, vom Brand weg nach oben. Ingenieure kennen das aus Brandversuchen unter dem Namen **thermal bowing** (Usmani u. a. 2001). Es ist der Moment, auf den das Buch seit Kapitel 1 zuläuft: Dort zeigte die Blackbox einen Träger, der sich unter Hitze verzog, und wir haben es geglaubt, ohne es zu verstehen. Jetzt rechnet dein Programm die Krümmung selbst aus — und wir prüfen sie gegen eine Handformel, die auf einen Bierdeckel passt.

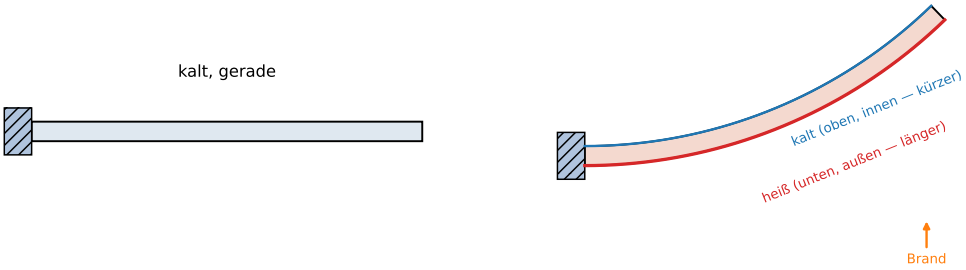

Abbildung 14.1.: Das kontinuierliche Bimetall. Links der kalte Träger, gerade. Rechts derselbe Träger mit einem Temperaturgefälle über die Höhe: die untere Faser (heiß, vom Brand) will länger sein als die obere (kalt, unter der Betondecke), aber beide hängen zusammen — also weicht der Längenunterschied in eine Krümmung aus. Der Träger wölbt sich zu einem Kreisbogen nach oben, vom Brand weg, die heiße Unterseite außen. Genau diese Krümmung rechnet das Kapitel aus.

Was man hier sieht: links der kalte Träger, rechts der vom Brand einseitig erwärmte. Weil die untere Faser heißer ist, *will* sie länger werden als die obere. Da beide Fasern untrennbar am selben Stück Stahl hängen, kann der Längenunterschied nicht als Verschiebung ausweichen wie in Kapitel 13 — er weicht in eine **Krümmung** aus. Der Träger wölbt sich in einen Kreisbogen nach oben, weg vom Brand, die heiße Unterseite außen. Der ganze Rest des Kapitels baut die Werkzeuge, um den Radius dieses Bogens auszurechnen.

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum ein Knoten in der Fläche **zwei** Verschiebungszahlen braucht (eine nach rechts, eine nach oben) und was das für die Größe des Gleichungssystems bedeutet ($2 \cdot n$ statt n),
2. … die **drei ebenen Dehnungen** anschaulich deuten — strecken in x, strecken in y, den rechten Winkel kippen (Schub) — am verformten Gitterbild,
3. … die **Querkontraktion** erklären (ein gedrückter Radiergummi wird dicker) und sagen, was $\nu = 0{,}30$ bedeutet,

4. … den Rechenweg der FEM-Kette nennen: Eckverschiebungen → Dehnungen (konstant pro Dreieck!) → Spannungen (Hooke mit ν) → Knotenkräfte,
5. … erklären, **warum** sich der einseitig erhitzte Träger krümmt — das Bimetall-Prinzip aus Kapitel 13, jetzt kontinuierlich über die Höhe,
6. … **Starrkörpermoden** diagnostizieren („der Träger schwimmt") — die 2D-Fassung des schwimmenden Systems aus Kapitel 8.

⚠ Naheliegende Vermutung

Vermutung: *„2D-Mechanik ist doppelt so schwer wie 1D — mehr Richtungen, mehr Formeln, eine ganz neue Maschine."*
Warum sie naheliegt: Die Fläche hat mehr Nachbarn, mehr Richtungen, mehr zu bedenken. Jeder Knoten trägt jetzt zwei Zahlen statt einer, die Elementmatrix wächst von 2×2 auf 6×6, und die Dehnung hat plötzlich drei Gesichter. Das *fühlt* sich nach einem großen Sprung an.
Was stattdessen stimmt: Es ist dasselbe Netz wie in 1, dieselbe Assemblierung, derselbe Löser. Die FEM-Maschinerie läuft unverändert weiter — sie merkt gar nicht, ob ein Knoten eine Temperatur oder zwei Verschiebungen trägt. Neu ist genau **eine** Zutat: die **Querkontraktion**, das seitliche Ausweichen des Materials. Und die kennt jeder, der je einen Radiergummi zusammengedrückt hat. Der reichere Steckbrief (6×6) und die drei Dehnungen sind bloß Buchhaltung — die Physik steckt in dem einen neuen Wort.

14.1. Schritt 1 — Zwei Zahlen pro Knoten

In Kapitel 13 hatte jeder Knoten des Trägers *eine* Unbekannte: seine Verschiebung u nach rechts. Der Träger konnte nur länger oder kürzer werden. In der Fläche kann ein Punkt in **zwei** Richtungen wandern: nach rechts (in x) und nach oben (in y). Also braucht jeder Knoten jetzt **zwei** Zahlen, sein Verschiebungspaar (u, v) — u misst die Wanderung nach rechts, v die nach oben. Zusammengenommen bilden die Pfeile aller Knoten das **Verschiebungsfeld**: An jedem Knoten ein kleiner Pfeil, der zeigt, wohin er sich bewegt hat.

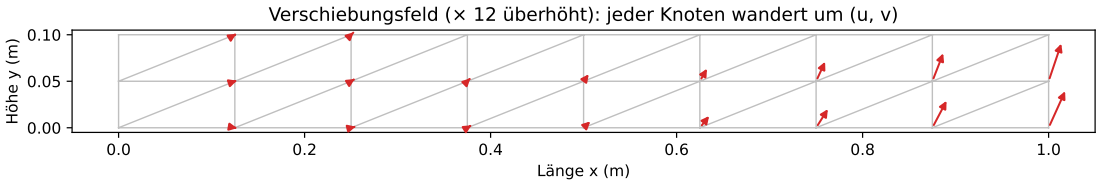

Abbildung 14.2.: Das Verschiebungsfeld auf dem Dreiecksnetz. Grau das unverformte Netz, an jedem Knoten ein roter Pfeil, der zeigt, wohin er wandert (stark überhöht). Jeder Pfeil ist ein Paar (u, v): u nach rechts, v nach oben. Der links eingespannte Träger mit unten-heißer Temperatur wölbt sich nach oben — die Pfeile werden nach rechts hin immer länger und drehen nach oben. Das ist die gesuchte Größe dieses Kapitels: nicht eine Zahl pro Knoten, sondern zwei.

Was man hier sieht: das unverformte Netz in Grau, an jedem Knoten einen roten Pfeil (stark überhöht) zu seiner neuen Lage. Am festen linken Rand sind die Pfeile null — dort ist der Träger eingespannt. Nach rechts hin werden sie länger und drehen nach oben: Der Träger wölbt sich vom Brand weg. Jeder einzelne Pfeil ist ein Zahlenpaar (u, v), und *diese Paare* sind die Unbekannten, die das Gleichungssystem gleich bestimmen muss.

Das hat eine schlichte, aber wichtige Folge für die Buchhaltung. Bei n Knoten gibt es jetzt $2n$ Unbekannte, also ein $2n \times 2n$-Gleichungssystem. Wir müssen uns nur einigen, in welcher Reihenfolge die beiden Zahlen eines Knotens im großen Vektor stehen. Die Abmachung dieses Buches ist die einfachste denkbare: Knoten i besetzt **zwei aufeinanderfolgende Zeilen** — Zeile $2i$ für seine Verschiebung u nach rechts, Zeile $2i+1$ für seine Verschiebung v nach oben.

Freiheitsgrad — jetzt zu zweit

In Kapitel 13 hat jeder Knoten *einen* **Freiheitsgrad** getragen (seine Verschiebung u); fünf Knoten gaben ein 5×5-System. In der Fläche hat jeder Knoten **zwei** Freiheitsgrade, u und v. Ein winziges Beispiel: Ein Netz mit vier Knoten hat acht Freiheitsgrade und ein 8×8-System. Knoten 0 sitzt in den Zeilen 0 und 1, Knoten 1 in den Zeilen 2 und 3, Knoten i allgemein in den Zeilen $2i$ (das u) und $2i+1$ (das v). Wer diese eine Regel — „mal zwei, plus null oder plus eins" — im Kopf hat, kann jede 2D-Elementmatrix richtig in die große Matrix einsortieren. Mehr Buchhaltung ist es nicht.

14.2. Schritt 2 — Die drei Dehnungen: ein Messprotokoll der Verformung

In Kapitel 13 gab es genau eine Dehnung: Der Träger wurde länger, $\varepsilon = \Delta L / L$. In der Fläche kann sich ein Materialstück auf **drei** Arten verformen, und die schauen wir uns am besten an einem kleinen Gitterquadrat an, das man verzieht. Stell dir ein aufgemaltes Quadrat auf einem Radiergummi vor — was kann damit passieren?

Erstens kann es in x-Richtung **gestreckt** werden: Es wird breiter, die senkrechten Linien wandern auseinander. Das misst die Dehnung ε_{xx} — genau die Dehnung aus Kapitel 13, nur jetzt ausdrücklich in x-Richtung. Zweitens kann es in y-Richtung gestreckt werden: Es wird höher, das misst ε_{yy}. Und drittens — das ist neu — kann der rechte Winkel des Quadrats **kippen**: Die senkrechten Linien neigen sich, aus dem Quadrat wird ein Parallelogramm, ohne dass es breiter oder höher würde. Diese Winkeländerung heißt **Schub** und wird mit γ_{xy} gemessen (der griechische Buchstabe Gamma). Sie zählt, um wie viel der ursprünglich rechte Winkel von 90° abweicht.

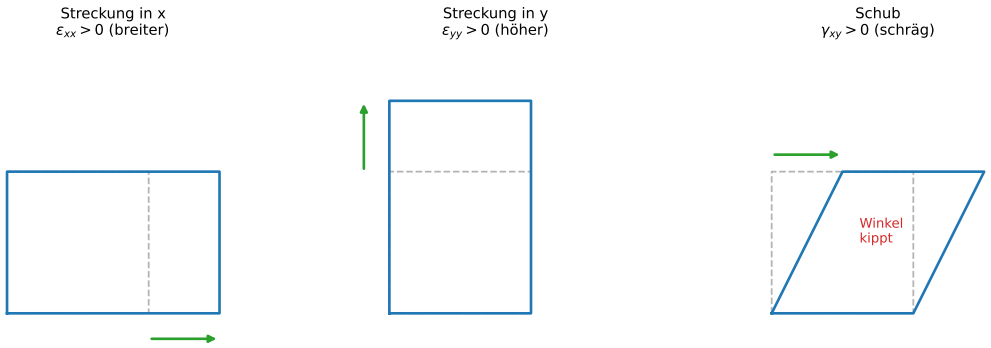

Abbildung 14.3.: Das Dehnungs-Trio: die drei Arten, ein Gitterquadrat zu verziehen. Grau gestrichelt das unverformte Einheitsquadrat, blau das verformte. Links Streckung in x (breiter, ε_xx > 0), Mitte Streckung in y (höher, ε_yy > 0), rechts Schub (der rechte Winkel kippt zum Parallelogramm, γ_xy > 0 — weder breiter noch höher, nur schräg). Jede echte Verformung ist eine Mischung dieser drei; zusammen sind sie das vollständige Messprotokoll dessen, was mit dem Material geschieht.

Was man hier sieht: die drei Grundverformungen einzeln. Streckung in x macht breiter, Streckung in y macht höher — beides misst, wie weit sich

Nachbarpunkte *voneinander entfernen*. Der Schub dagegen macht weder breiter noch höher; er *kippt* nur den rechten Winkel, aus dem Quadrat wird ein Parallelogramm. Jede wirkliche Verformung eines Materialstücks ist eine **Mischung** dieser drei — mehr Grundzutaten gibt es in der Fläche nicht. Damit haben wir ein vollständiges „Messprotokoll", das festhält, was mit einem Materialstückchen passiert ist: drei Zahlen, ε_{xx}, ε_{yy}, γ_{xy}.

Warum genau diese drei und keine vierte? Weil eine Bewegung, die *keine* dieser drei Zahlen verändert, gar keine Formänderung ist — sie verschiebt oder dreht das Stück bloß als Ganzes, und dagegen wehrt sich kein Material. Das ist derselbe Gedanke wie in Kapitel 8, wo der Träger „schwamm": Wer alles gleich verschiebt, erzeugt keine Kraft. In Schritt 6 kommen wir darauf zurück; hier merken wir uns nur, dass die drei Dehnungen genau das erfassen, was Spannung kostet.

Die Voigt-Spalte: nur ein Ordnungsschema

Damit das Programm mit den drei Dehnungen rechnen kann, schreiben wir sie ordentlich untereinander in eine Spalte — immer in derselben Reihenfolge:

$$\begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{pmatrix} \quad = \quad \begin{pmatrix} \text{Streckung in } x \\ \text{Streckung in } y \\ \text{Winkelkippung} \end{pmatrix}.$$

Genauso schreibt man die drei Spannungen als Spalte $(\sigma_{xx}, \sigma_{yy}, \tau_{xy})$: zweimal ein Ziehen/Drücken senkrecht zu einer Schnittfläche und einmal eine Schub-Spannung τ, die tangential zerrt. Diese Drei-Zeilen-Schreibweise heißt nach dem Physiker **Woldemar Voigt Voigt-Schreibweise**. Sie ist kein neues Konzept, sondern nur eine **Sortierregel** — welche Größe in welcher Zeile steht —, damit die Materialtabelle im nächsten Schritt eine handliche 3×3-Tabelle wird. Mehr steckt nicht dahinter.

14.3. Schritt 3 — Die Querkontraktion: das seitliche Ausweichen

Jetzt kommt die einzige wirklich neue Physik des Kapitels. Nimm einen Radiergummi und drücke ihn zwischen zwei Fingern zusammen. Er wird nicht nur kürzer — er wird auch **dicker**. Das Material, das du in einer Richtung zusammenschiebst, weicht in die Querrichtung aus. Umgekehrt: Ziehst du

an einem Gummiband, wird es länger *und dünner*. Dieses seitliche Ausweichen heißt **Querkontraktion**, und es ist der Grund, warum Mechanik in der Fläche „vernetzter" ist als die Wärmeleitung: Eine Dehnung in x zieht automatisch eine Verformung in y nach sich.

Wie stark ein Material seitlich ausweicht, sagt die **Querkontraktionszahl** ν (der griechische Buchstabe Ny). Sie ist das Verhältnis „seitliches Einschnüren geteilt durch Längsdehnung". Ziehe ich ein Material um 1 % in die Länge und es schnürt sich dabei um $0{,}30\,\%$ seitlich ein, dann ist $\nu = 0{,}30$ — der Wert unseres Baustahls. ν ist eine reine Zahl ohne Einheit und liegt für gewöhnliche Feststoffe zwischen 0 und $\frac{1}{2}$.

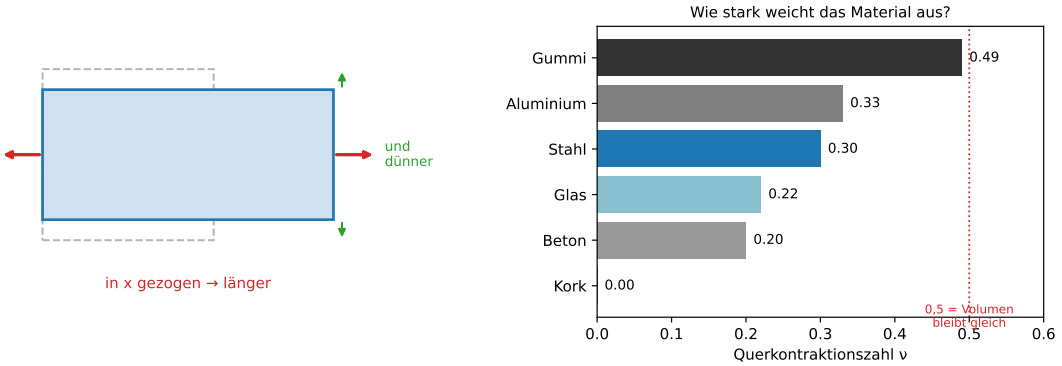

Abbildung 14.4.: Die Querkontraktion am gezogenen Block. Grau gestrichelt der Ausgangszustand, blau der in x gezogene Block: Er wird länger UND dünner — das seitliche Einschnüren ist die Querkontraktion. Rechts die Querkontraktionszahl ν einiger Materialien: Kork weicht kaum aus (ν ≈ 0, deshalb lässt sich ein Korken in den Flaschenhals drücken, ohne dick zu werden), Metalle liegen um 0,3 (unser Stahl 0,30, hervorgehoben), Gummi geht gegen 0,5 (es weicht vollständig aus, das Volumen bleibt gleich).

Was man hier sieht: links das seitliche Einschnüren beim Ziehen, rechts die ν-Werte einiger Stoffe. Kork weicht fast gar nicht aus ($\nu \approx 0$) — genau deshalb lässt sich ein Korken in den Flaschenhals drücken, ohne dass er seitlich dicker wird und klemmt. Metalle liegen um $0{,}3$, unser Stahl bei $0{,}30$, Aluminium mit $0{,}33$ knapp darüber. Gummi geht gegen $0{,}5$: Es weicht so vollständig zur Seite aus, dass sein Volumen praktisch gleich bleibt — man kann es kaum zusammendrücken, nur umformen. (Bei genau $\nu = \frac{1}{2}$ bekommt die Standard-FEM übrigens Schwierigkeiten; das ist der Grund, warum Gummi und wassergesättigter Boden Spezialverfahren brauchen — für unser $\nu = 0{,}30$ ist alles harmlos.)

Mit ν können wir das **Hookesche Gesetz für die Fläche** aufschreiben. In Kapitel 13 war es eine einzige Zeile, $\sigma = E\,\varepsilon$. Jetzt verknüpft es die drei Dehnungen mit den drei Spannungen, und weil die Querkontraktion x und y koppelt, wird daraus eine 3×3-**Materialtabelle**:

$$\begin{pmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{pmatrix} = \frac{E}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix} \begin{pmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{pmatrix}.$$

Lies die Tabelle Zeile für Zeile. Die erste Zeile sagt: Die Spannung in x-Richtung kommt aus der Dehnung in x (der Eintrag 1) **und** — über die Querkontraktion — aus der Dehnung in y (der Eintrag ν). Genau das ist die neue Kopplung: Ziehen in y erzeugt Spannung in x. Die zweite Zeile ist dieselbe Geschichte mit vertauschten Richtungen. Die dritte Zeile steht für sich allein: Der Schub γ_{xy} erzeugt nur Schubspannung τ_{xy}, über den Faktor $\frac{1-\nu}{2}$, und mischt sich nicht mit den beiden Streckungen. Setzt man $\nu = 0$, zerfällt die Tabelle in drei entkoppelte Federn — dann ist x von y unabhängig, und man hätte dreimal Kapitel 13 nebeneinander. Erst $\nu > 0$ macht aus dem Nebeneinander ein echtes Miteinander.

Warum der ebene Spannungszustand für unseren Träger passt

Die Tabelle oben gilt für den **ebenen Spannungszustand**: ein Bauteil, das in der dritten Richtung (der Tiefe, aus dem Papier heraus) **frei** ist und dort keine Spannung aufbaut. Das passt genau zu unserem Träger: Er ist mit $0{,}1$ m dünn, und nichts hindert ihn daran, in die Tiefe ein Härchen dicker oder dünner zu werden. Deshalb ist die Spannung quer zur Zeichenebene null — „ebener Spannungszustand".
Das **Gegenstück** heißt ebener *Verzerrungs*zustand: ein sehr langer Körper, der in die Tiefe *nicht* ausweichen kann (ein Tunnel, ein Staudamm, eine lange Schweißnaht). Dort ist die *Dehnung* quer null statt der Spannung, und die Materialtabelle sieht etwas anders aus. Für den dünnen, frei liegenden Träger dieses Buches ist der ebene Spannungszustand die richtige Wahl; den anderen Fall erwähnen wir nur, damit du weißt, dass es ihn gibt — die Verwechslung der beiden ist einer der häufigsten Fehler in der Praxis.

14.4. Schritt 4 — Vom Dreieck zum Steckbrief: die 6×6-Matrix

Jetzt fügen wir die Teile zum Element-Steckbrief zusammen. Der Rechenweg ist eine kurze Kette, und jedes Glied kennst du schon:

1. **Eckverschiebungen → Dehnungen.** Ein P1-Dreieck hat drei Ecken, jede mit ihrem Paar (u, v) — zusammen sechs Zahlen. Aus diesen sechs Zahlen die drei Dehnungen zu machen, ist genau die Arbeit der **Zeltflächen-Steigungen** aus

    1. Dort hatte jede Zeltfunktion über dem Dreieck eine konstante Steigung in x und eine in y; die Dehnung ε_{xx} ist die Summe der x-Steigungen mal den u-Werten, ε_{yy} die Summe der y-Steigungen mal den v-Werten, und der Schub γ_{xy} mischt beide. Weil die Steigungen über das Dreieck **konstant** sind, ist auch die Dehnung **konstant über das ganze Dreieck** — ein P1-Dreieck kennt genau einen Verformungszustand. (Das hat einen Preis, auf den wir im Finale ehrlich zu sprechen kommen.)

2. **Dehnungen → Spannungen.** Das erledigt die 3×3-Materialtabelle aus Schritt 3.
3. **Spannungen → Knotenkräfte.** Rückwärts über dieselben Steigungen, mal Tiefe mal Dreiecksfläche.

Setzt man alle drei Glieder hintereinander, entsteht der **6×6-Steckbrief** des Dreiecks: eine Tabelle, die sagt, welche Kraft an jedem der sechs Freiheitsgrade entsteht, wenn man einen anderen um eins verschiebt. Das ist das flächenhafte Gegenstück zur 2×2-Feder aus Kapitel 13 — nur eben reicher, weil ein Dreieck sechs Freiheitsgrade hat statt zwei.

Und hier sind wir ehrlich: Diese 6×6-Tabelle **von Hand** auszurechnen wäre eine Zumutung — sechsunddreißig Einträge, jeder eine kleine Rechnung. Kein Mensch tut das, und du sollst es auch nicht. Stattdessen lassen wir das **Programm** die Tabelle für *ein* Dreieck ausrechnen und **lesen sie Zeile für Zeile**, um zu prüfen, ob sie plausibel ist. Das ist der ehrliche Umgang mit einer Formel, die zu groß fürs Kopfrechnen ist: nicht herleiten, sondern nachrechnen lassen und *verstehen, was herauskommt*. Der Baustein steckt im gemeinsamen Modul `programme/gemeinsam/elastik2d.py`, das die Kapitel 14 und 15 tragen wird.

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from elastik2d import baue_element_6x6, starrkoerper_check
```

```
# Ein einzelnes Träger-Dreieck, wie es im 8-Dreiecke-Netz vorkommt.
ECKEN = [(0.0, 0.0), (0.5, 0.0), (0.5, 0.05)]
E_MODUL = 210e9       # Pa  (Baustahl S235)
QUERKONTRAKTION = 0.30
TIEFE = 0.1           # m

steckbrief = baue_element_6x6(ECKEN, E_MODUL, QUERKONTRAKTION, TIEFE)

namen = ["u0", "v0", "u1", "v1", "u2", "v2"]
print("6x6-Steckbrief (Werte in MN/m):")
print("      " + "".join("%8s" % n for n in namen))
for i in range(6):
    zeile = "  %2s " % namen[i]
    for j in range(6):
        zeile = zeile + "%8.1f" % (steckbrief[i][j] / 1e6)
    print(zeile)
```

```
6x6-Steckbrief (Werte in MN/m):
            u0      v0      u1      v1      u2      v2
  u0   1153.8     0.0 -1153.8  3461.5     0.0 -3461.5
  v0      0.0   403.8  4038.5  -403.8 -4038.5     0.0
  u1  -1153.8  4038.5 41538.5 -7500.0-40384.6  3461.5
  v1   3461.5  -403.8 -7500.0115788.5  4038.5-115384.6
  u2      0.0 -4038.5-40384.6  4038.5 40384.6     0.0
  v2  -3461.5     0.0  3461.5-115384.6     0.0115384.6
```

Interpretation der Ausgabe — Zeile für Zeile gelesen. Drei Dinge muss eine brauchbare Steifigkeitstabelle erfüllen, und alle drei kann man der Ausgabe ansehen:

- **Die Diagonale ist positiv.** Auf der Diagonale steht die Kraft, die an einem Freiheitsgrad entsteht, wenn man *ihn selbst* verschiebt und alles andere festhält. Sie muss positiv sein — das Material drückt zurück, nicht mit. Bei `u1`, `v1`, `u2`, `v2` (den Ecken auf der langen Seite) stehen große Werte: Diese Freiheitsgrade sitzen an kurzen, steifen Kanten und wehren sich heftig.
- **Die Tabelle ist symmetrisch.** Der Eintrag in Zeile `u0`, Spalte `v2` ist derselbe wie in Zeile `v2`, Spalte `u0`. Das ist kein Zufall, sondern ein tiefes Gesetz (es heißt Reziprozität): Die Kraft, die eine Verschiebung *hier dort* auslöst, ist dieselbe wie umgekehrt. Genau diese Symmetrie hatten schon die Federkette (Kapitel 3) und die Wärmematrix (1).
- **Die Zeilensummen der drei u-Spalten sind null** (ebenso die der drei v-Spalten). Verschiebt man alle drei Ecken um denselben Betrag

nach rechts — alle $u = 1$, alle $v = 0$ —, entsteht **keine Kraft**: Das Dreieck wandert bloß als Ganzes, ohne die Form zu ändern. Das ist die Starrkörper-Probe, die wir gleich formalisieren.

Diese letzte Eigenschaft prüft man am ehrlichsten, indem man dem Programm die drei starren Bewegungen — nach rechts schieben, nach oben schieben, drehen — vorlegt und die entstehende Kraft misst. Sie muss null sein:

```
moden = starrkoerper_check(ECKEN, E_MODUL, QUERKONTRAKTION, TIEFE)
print("Starrkörper-Probe (Länge der entstehenden Knotenkraft, sollte 0 sein):")
for name in ("schieben_x", "schieben_y", "drehen"):
    print("  %-11s: %.2e N" % (name, moden[name]))
```

```
Starrkörper-Probe (Länge der entstehenden Knotenkraft, sollte 0 sein):
  schieben_x : 7.63e-06 N
  schieben_y : 3.05e-05 N
  drehen     : 1.19e-07 N
```

Interpretation der Ausgabe: Alle drei Werte sind praktisch null (Größenordnung 10^{-6} N — nur Rundungsstaub gegen die 10^{9} N/m der Steifigkeit). Das Dreieck „merkt" eine starre Verschiebung oder Drehung nicht, weil sie keine Formänderung ist — genau wie es sein muss. Diese drei Nullbewegungen (zweimal schieben, einmal drehen) werden in Schritt 6 wichtig: Sie sind der Grund, warum ein *ungelagerter* Träger davonschwimmt.

14.5. Schritt 5 — Die thermische Kraft in der Fläche

In Kapitel 13 hat ein erwärmtes Stab-Element seine beiden Knoten mit dem Kraftpaar $E\,A\,\alpha\,\Delta T$ auseinandergedrückt. In der Fläche ist der Trick derselbe, nur wirkt die Wärme jetzt in **zwei** Richtungen zugleich: Ein erwärmtes Dreieck will sich in x **und** in y ausdehnen, in beide Richtungen gleich stark. Seine freie Wärmedehnung ist darum die Voigt-Spalte

$$\varepsilon_{\text{therm}} = \begin{pmatrix} \alpha\,\Delta T \\ \alpha\,\Delta T \\ 0 \end{pmatrix} \qquad \text{(strecken in } x\text{, strecken in } y\text{, kein Schub).}$$

Der Schub ist null, denn Wärme kippt keinen Winkel — sie bläht das Material gleichmäßig auf. Aus dieser Vordehnung werden wieder Knotenkräfte,

genau wie in Kapitel 13: Man hält das Dreieck gedanklich auf Form (dabei entsteht eine Druckspannung), rechnet die Haltekraft aus und nimmt sie dann wieder weg. Das Ergebnis ist ein **Sechser-Kraftpaket**, eines für jeden der sechs Freiheitsgrade des Dreiecks, das seine Ecken nach außen drückt.

Rezept: die thermische Kraft eines Dreiecks

Für jedes Dreieck:

1. Bestimme seine mittlere Übertemperatur ΔT (Mittel der drei Knotentemperaturen minus der spannungsfreien Referenz $20\,°\mathrm{C}$).
2. Bilde die Vordehnung $[\alpha\,\Delta T,\ \alpha\,\Delta T,\ 0]$.
3. Schicke sie rückwärts durch dieselbe Steigungs-Rechnung wie den Steckbrief (Materialtabelle, dann Steigungen, mal Tiefe mal Fläche). Heraus kommen sechs Knotenkräfte.
4. Addiere die sechs Kräfte an den Freiheitsgraden der drei Ecken auf die rechte Seite.

Das ist die einzige Stelle, an der die Wärme in die Mechanik übergeht — die **Einweg-Kopplung** aus Kapitel 13: Die Temperatur treibt die Verformung, aber die Verformung wirkt nicht auf die Temperatur zurück. Erst das Temperaturproblem lösen (Kapitel 6–10), dann sein Ergebnis als feste Last hier einsetzen.

Diese thermischen Kräfte assembliert man genau wie in Kapitel 13 die Stab-Kräfte: An einem inneren Knoten stoßen mehrere Dreiecke zusammen, ihre Kräfte zeigen gegeneinander, und nur ihre Differenz bleibt stehen. Wo alle Dreiecke gleich warm sind, hebt sich im Inneren alles auf. Steckt aber ein Temperaturgefälle im Netz — oben heißer als unten —, bleibt eine ungleichgewichtige Last übrig, und *die* ist es, die den Träger gleich krümmt.

14.6. Schritt 6 — Lagerung, und was ohne sie geschieht

Bevor wir lösen, müssen wir den Träger **festhalten**. Das Rezept ist das Streichen aus Kapitel 8, jetzt für Freiheitsgrade: Ein festgehaltener Knoten gibt seine beiden Zeilen und Spalten ab. Für unseren Kragträger halten wir die **ganze linke Kante fest** — dort ist er eingespannt, jeder linke Knoten bekommt $u = 0$ **und** $v = 0$. Das rechte Ende bleibt völlig **frei**.

Was passiert, wenn wir das Festhalten *vergessen*? Dann erleben wir die 2D-Fassung des schwimmenden Trägers aus Kapitel 8. Dort war der Träger „nur

bis auf eine Konstante bestimmt" — man konnte das ganze Temperaturprofil nach oben schieben, ohne die Bilanz zu verletzen. Beim mechanischen Träger gibt es **drei** solche freien Bewegungen statt einer: Man kann ihn als Ganzes nach rechts schieben, nach oben schieben oder drehen, ohne dass sich seine Form ändert — und ohne dass eine einzige Kraft entsteht. Es sind genau die drei Nullbewegungen aus der Starrkörper-Probe in Schritt 4, jetzt für den ganzen Träger statt für ein Dreieck.

⚠ Der schwimmende Träger, jetzt in der Fläche

Ein Wärmeproblem ohne festen Temperaturwert schwimmt in **einer** Richtung (die Höhe des Profils ist frei, Kapitel 8). Ein mechanischer Träger ohne Lager schwimmt in **drei**: zwei Verschiebungen (nach rechts, nach oben) und eine Drehung. Diese drei **Starrkörpermoden** sind Bewegungen ohne Formänderung, also ohne Kraft — die Systemmatrix hat für sie kein Rückstellvermögen, sie ist **nicht lösbar** (der Gauß-Löser stolpert über eine Null auf der Diagonale, das Gauß-Seidel-Verfahren driftet Runde um Runde weiter davon). Erst genügend Lager nehmen dem Träger alle drei Nullbewegungen und machen das System eindeutig. Für den Kragträger erledigt das die eingespannte Kante mühelos; wie man mit dem *kleinstmöglichen* Festhalten auskommt („gerade genug"), üben wir in Übung 14.3.

Die drei Starrkörpermoden sieht man am besten in Bewegung — das **Starrkörper-Ballett**: Derselbe erwärmte Träger, der ohne Lager nicht rechnen will, treibt in jeder dieser drei Richtungen davon.

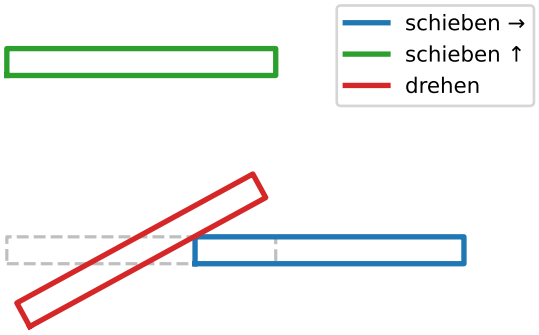

Abbildung 14.5.: Das Starrkörper-Ballett als Standbild (in der HTML-Fassung eine Animation): die drei kraftfreien Bewegungen eines ungelagerten Trägers — nach rechts schieben (blau), nach oben schieben (grün), drehen (rot), jeweils aus der grau gestrichelten Ausgangslage. Weil keine davon die Form ändert, kostet keine eine Kraft — und darum ist der ungelagerte Träger nicht berechenbar.

Was man sieht: drei Bewegungen des ganzen Trägers, keine ändert seine Form. Der Träger bleibt in jedem Fall ein starres Rechteck — er schiebt sich nur, oder er dreht sich. Genau deshalb erzeugt keine dieser Bewegungen eine Kraft, und genau deshalb kann ein Programm den ungelagerten Träger nicht ausrechnen: Es gibt unendlich viele gleichberechtigte Lagen, keine ist *die* Lösung. Erst das Festhalten sperrt die drei Nullbewegungen — dann steht der Träger, und das System wird lösbar.

14.7. Schritt 7 — Das Finale: der Träger krümmt sich

Jetzt haben wir alles. Fügen wir die Glieder zur vollständigen Rechenkette zusammen — es ist dieselbe Kette wie in Kapitel 13, nur in der Fläche.

Abbildung 14.6.: Die Rechenkette der 2D-Thermoelastik als Fließband. Das Temperaturfeld (aus Kapitel 6–10) wird je Dreieck gemittelt, daraus werden die thermischen Kräfte; das assemblierte Gleichungssystem K·u = F liefert die Verschiebungen (zwei je Knoten), aus denen man die drei Dehnungen und daraus die drei Spannungen zurückrechnet. Der Pfeil zeigt nur nach rechts: Einweg-Kopplung. Es ist Wort für Wort die Kette aus Kapitel 13, nur mit reicheren Zutaten.

Die Last dieses Kapitels ist ein **Temperaturgefälle über die Höhe**: unten $220\,°\mathrm{C}$ (die vom Brand beheizte Unterseite), oben $20\,°\mathrm{C}$ (die von der Betondecke kühl gehaltene Oberseite, zugleich die spannungsfreie Referenz). Zwischen unten und oben fällt die Temperatur geradlinig ab. Jede waagerechte Faser ist also gleich warm, aber die unteren Fasern sind heißer als die oberen — genau das kontinuierliche Bimetall aus dem Aufhänger. Die untere Faser will um $\alpha\,\Delta T = 12 \cdot 10^{-6} \cdot 200 = 2{,}4$ mm pro Meter länger sein als die obere, und weil sie es nicht darf, wölbt sich der Träger nach oben. Die $\Delta T = 200\,\mathrm{K}$ über die $0{,}1\,\mathrm{m}$ Höhe entsprechen einer Wärmestromdichte von rund $100\,\mathrm{kW/m^2}$ durch die Höhe — eine realistische Größenordnung für die Einwirkung eines Brandes von unten.

Bevor wir das Programm laufen lassen, rechnen wir die Krümmung von Hand — das Verifikations-Ritual aus Kapitel 12. Für ein lineares Temperaturgefälle über die Höhe krümmt sich der Träger zu einem **Kreisbogen** mit der Krümmung

$$\kappa = \frac{\alpha\,\Delta T_{\text{Höhe}}}{H} = \frac{12 \cdot 10^{-6} \cdot 200\ \mathrm{K}}{0{,}1\ \mathrm{m}} = 0{,}024\ \tfrac{1}{\mathrm{m}}.$$

Die Krümmung κ ist der Kehrwert des Krümmungsradius, also biegt sich der Träger auf einen Radius von

$$R = \frac{1}{\kappa} = \frac{H}{\alpha\,\Delta T_{\text{Höhe}}} = \frac{0{,}1}{12 \cdot 10^{-6} \cdot 200} = 41{,}7\ \mathrm{m}.$$

Ein Bogen mit $41{,}7$ m Radius ist sehr flach — auf einem Meter Länge weicht er nur wenig von der Geraden ab. Wie weit genau? Das freie Ende eines links eingespannten Trägers der Länge L hebt sich gegenüber der Geraden um näherungsweise

$$w \approx \frac{L^2}{2R} = \frac{(1\ \mathrm{m})^2}{2 \cdot 41{,}7\ \mathrm{m}} = 0{,}0120\ \mathrm{m} = 12{,}0\ \mathrm{mm}$$

nach oben. **Das ist unsere Vorhersage: rund 12,0 Millimeter Hebung am freien Ende, weg vom Brand.** Jetzt lassen wir die FEM rechnen und schauen, ob sie dieselbe Zahl trifft.

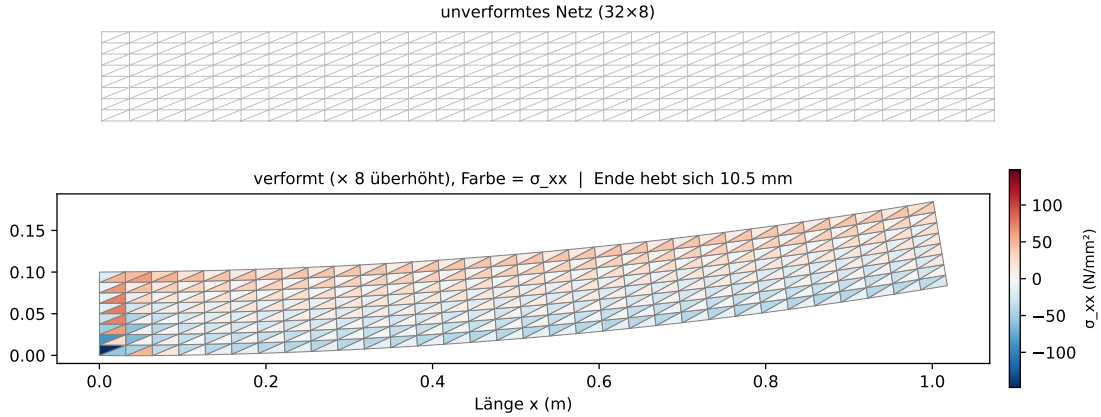

Abbildung 14.7.: Das Ergebnisbild des Buches. Oben das unverformte Netz (grau), darunter der gekrümmte Träger (× 8 überhöht), eingefärbt nach der Längsspannung σ_xx: rot Zug, blau Druck. Die heiße untere Faser steht (nahe der Einspannung) unter Druck, die kühle obere unter Zug — der Träger wölbt sich nach oben, heiße Unterseite außen. Netz 32×8; das freie Ende hebt sich um rund 10,5 mm und nähert sich der Handformel-Vorhersage von 12,0 mm von unten (grobe P1-Netze biegen zu steif, siehe Kasten).

Was man hier sieht: oben das ruhende Netz, unten der gekrümmte Träger (überhöht), eingefärbt nach der Längsspannung σ_{xx}. Der Träger wölbt sich nach oben — die heiße untere Seite legt sich nach außen, genau wie im Aufhänger. Die Farben zeigen: Nahe der Einspannung, wo die Krümmung erzwungen wird, steht die untere Faser unter Druck (blau) und die obere unter Zug (rot); weiter rechts, wo sich der Träger frei krümmen darf, werden die Spannungen klein. Das freie Ende hebt sich um rund $10{,}5$ mm — nah an unserer Handformel-Vorhersage von $12{,}0$ mm, aber noch nicht ganz dort.

Warum die Lücke? Das ist kein Fehler, sondern eine ehrliche Eigenschaft grober Dreiecksnetze, und sie verdient einen eigenen Kasten.

> **Grobe Netze biegen zu steif — ein Fall für Kapitel 12**
>
> Ein P1-Dreieck kennt (Schritt 4) nur **eine** Dehnung über seine ganze Fläche. Biegung aber verlangt, dass die Dehnung sich über die Höhe *ändert* — auf der Zugseite gestreckt, auf der Druckseite gestaucht. Ein einzelnes Dreieck kann das gar nicht darstellen; viele übereinandergestapelte Dreiecke können es nur als grobe Treppe nachbauen. Die Folge: Grobe P1-Netze sind bei Biegung **systematisch zu steif** — sie biegen sich zu wenig, die berechnete Durchbiegung ist zu klein. Erst mit feinerem Netz nähert sie sich der wahren Krümmung, und zwar **von unten**. Das ist genau die Netzkonvergenz aus Kapitel 12, jetzt an der Verformung: noch ein Grund für die Prüfrituale jenes Kapitels.

Wir rechnen dieselbe Aufgabe auf immer feineren Netzen und schauen, ob die Durchbiegung brav gegen die Handformel läuft:

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from elastik2d import (assembliere_elastik, assembliere_thermokraefte,
                       loese_elastik)
from dreiecke import strukturiertes_dreiecksnetz

E_MODUL, QUER, TIEFE, ALPHA = 210e9, 0.30, 0.1, 12e-6
L, H, REF = 1.0, 0.1, 20.0
W_HAND = 12.0   # mm, aus der Handformel L^2 / (2R)

print("Netz    | Hebung Ende (oben)| Anteil der Handformel")
print("--------+-------------------+----------------------")
for nx, ny in [(4, 1), (8, 2), (16, 4), (32, 8)]:
    knoten, dreiecke = strukturiertes_dreiecksnetz(nx, ny, L, H)
    delta_t = []
    for d in dreiecke:
        mittel = sum(220.0 - 200.0 * knoten[d[k]][1] / H for k in range(3)) / 3.0
        delta_t.append(mittel - REF)
    matrix = assembliere_elastik(knoten, dreiecke, E_MODUL, QUER, TIEFE)
    kraefte = assembliere_thermokraefte(knoten, dreiecke, E_MODUL, QUER,
                                        TIEFE, ALPHA, delta_t)
    lager = {}
    for i, (x, y) in enumerate(knoten):
        if abs(x) < 1e-12:
            lager[2 * i] = 0.0
```

```
            lager[2 * i + 1] = 0.0
    u = loese_elastik(matrix, kraefte, lager)
    for i, (x, y) in enumerate(knoten):
        if abs(x - L) < 1e-9 and abs(y) < 1e-9:
            w = u[2 * i + 1] * 1000.0
    print("%3d×%-2d  | %11.2f mm    | %5.0f %%"
          % (nx, ny, w, 100.0 * w / W_HAND))
```

```
Netz    | Hebung Ende (oben)| Anteil der Handformel
--------+-------------------+----------------------
  4×1   |         0.99 mm    |     8 %
  8×2   |         3.53 mm    |    29 %
 16×4   |         7.56 mm    |    63 %
 32×8   |        10.46 mm    |    87 %
```

Interpretation: Von 4×1 zu 32×8 wächst die Hebung von knapp 1 mm auf gut 10 mm und marschiert von unten auf die $12{,}0$ mm der Handformel zu. Sie ist bei 32×8 noch nicht ganz dort ($\approx 87\,\%$); ein feineres Netz (64×16) schafft rund $96\,\%$, aber das dauert in reinem Python schon spürbar. Die Lehre ist die von Kapitel 12: **Ein grobes buntes Bild sieht schon richtig aus, ist aber quantitativ noch zu steif.** Wer eine Zahl braucht, verfeinert, bis sie sich nicht mehr ändert — oder greift zu Elementen höherer Ordnung, die Biegung von Haus aus besser können (der Ausblick am Kapitelende).

Trotz der noch offenen $13\,\%$ ist die Hauptsache erreicht und **verifiziert**: Der Träger krümmt sich in die richtige Richtung, mit der richtigen Größenordnung, und die FEM-Durchbiegung läuft nachweislich auf die unabhängig hergeleitete Handformel zu. Das ist der Moment aus Kapitel 1, jetzt nicht mehr geglaubt, sondern gerechnet und geprüft.

14.8. Zwei Filme: die drei Dehnungen und das Krümmen

Der erste Film macht das Dehnungs-Trio aus Schritt 2 lebendig: Ein Einheitsquadrat durchläuft die drei Moden einzeln — strecken in x, strecken in y, kippen (Schub) — und zum Schluss alle drei gemischt.

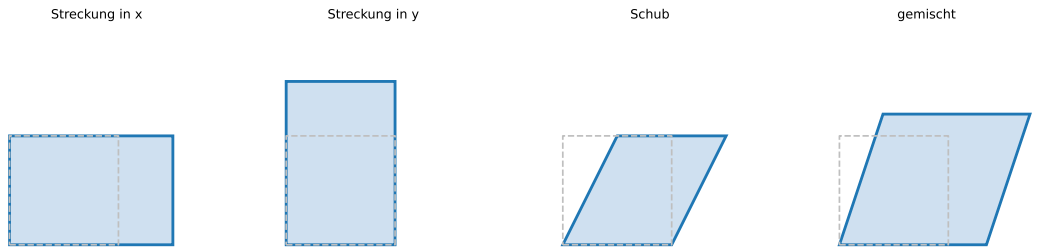

Abbildung 14.8.: Der Verzerrungs-Morph als Bildreihe (HTML: Animation): das Einheitsquadrat in den drei Grundmoden und einer Mischung. Grau gestrichelt der Ausgang, blau die Verformung.

Was man sieht: Streckung in x macht breiter, Streckung in y höher, Schub kippt den Winkel, und die Mischung tut alles zugleich. Diese drei Zahlen — mehr braucht es nicht, um jede ebene Verformung zu beschreiben.

Der zweite Film ist das Herzstück: Das Temperaturgefälle über die Höhe baut sich auf, und der Träger krümmt sich **synchron** mit. Es ist der Trailer aus Kapitel 1 — jetzt verstanden, Glied für Glied.

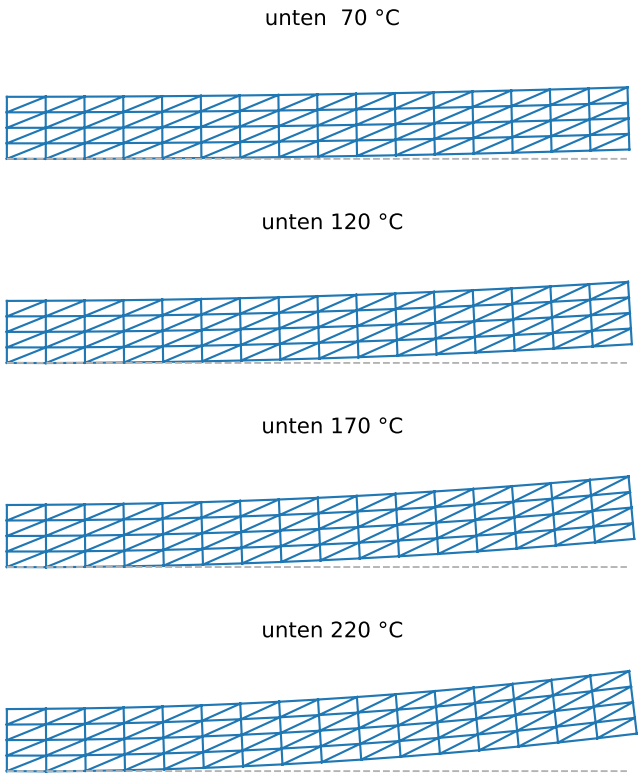

Abbildung 14.9.: Der Aufheiz-und-Krümm-Film als Bildreihe (HTML: Animation): mit wachsendem Temperaturgefälle über die Höhe (Unterseite 70, 120, 170, 220 °C, also 50, 100, 150, 200 K heißer als oben) wölbt sich der Träger immer stärker nach oben. Verformung × 8 überhöht; gestrichelt die Ausgangslage.

Was man sieht: Mit jedem Grad, das die untere Faser heißer wird, wölbt sich der Träger ein Stück weiter — die Krümmung wächst genau proportional zum Temperaturunterschied (das ist der Kern von Übung 14.2). Am Ende steht der gekrümmte Träger aus dem Ergebnisbild. Der Überhöhungsfaktor ist ehrlich eingeblendet; in Wahrheit sind es Millimeter.

14.9. Die interaktive Einheit: die Biegebank

Jetzt bist du dran. Die **Biegebank** spannt den Träger ein, legt ein Temperaturgefälle über die Höhe an und zeigt den gekrümmten Träger, die Spannung σ_{xx} als Farbe und den Vergleich mit der Handformel. Du

kannst das Netz verfeinern (`NX`, `NY`), das Gefälle ändern (`TEMPERATUR_OBEN`, `TEMPERATUR_UNTEN`), das Lager lösen (`LAGERUNG`) und — der spannendste Regler — die Querkontraktion `NU` von 0 bis $0{,}45$ verstellen. Bevor du ihn anfasst, eine Vorhersage:

> Vorhersage-Punkt
>
> **Bevor du das Programm startest:** Du stellst `NU` von $0{,}30$ auf 0 — du schaltest die Querkontraktion also *ab*. Krümmt sich der Träger dann

```
#| autorun: true
# Die Biegebank -- drehe an den Reglern:
#   LAGERUNG = "links_fest"  (Kragträger)  oder  "frei"  (Starrkoerper-Erlebnis)
#   TEMPERATUR_OBEN / _UNTEN: das Gefaelle ueber die Hoehe (Grad Celsius)
#   NU: Querkontraktion 0 ... 0,45
#   NX, NY: Netzfeinheit (im Browser klein halten!)
LAGERUNG = "links_fest"
TEMPERATUR_UNTEN = 220.0
TEMPERATUR_OBEN = 20.0
NU = 0.30
NX = 12
NY = 3
UEBERHOEHUNG = 8.0

E_MODUL = 210e9
TIEFE = 0.1
ALPHA = 12e-6
LAENGE = 1.0
HOEHE = 0.1
REFERENZ = 20.0


def baue_netz(nx, ny):
    """Strukturiertes Dreiecksnetz ueber dem Träger (wie Kapitel 10)."""
    knoten = []
    for zeile in range(ny + 1):
        for spalte in range(nx + 1):
            knoten.append((spalte * LAENGE / nx, zeile * HOEHE / ny))
    dreiecke = []
    for zeile in range(ny):
```

```
        for spalte in range(nx):
            unten_links = zeile * (nx + 1) + spalte
            unten_rechts = unten_links + 1
            oben_links = unten_links + (nx + 1)
            oben_rechts = oben_links + 1
            dreiecke.append((unten_links, unten_rechts, oben_rechts))
            dreiecke.append((unten_links, oben_rechts, oben_links))
    return knoten, dreiecke


def material_und_steigungen(ecken, nu):
    """Gibt (C-Tabelle, b, c, Flaeche) fuer ein Dreieck zurueck."""
    (xa, ya) = ecken[0]
    (xb, yb) = ecken[1]
    (xc, yc) = ecken[2]
    zwei = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    b = [(yb - yc) / zwei, (yc - ya) / zwei, (ya - yb) / zwei]
    c = [(xc - xb) / zwei, (xa - xc) / zwei, (xb - xa) / zwei]
    flaeche = zwei
    if flaeche < 0.0:
        flaeche = -flaeche
    flaeche = flaeche / 2.0
    faktor = E_MODUL / (1.0 - nu * nu)
    c_tab = [[faktor, faktor * nu, 0.0],
             [faktor * nu, faktor, 0.0],
             [0.0, 0.0, faktor * (1.0 - nu) / 2.0]]
    return c_tab, b, c, flaeche


def b_tabelle(b, c):
    """Die 3x6-Tabelle: aus sechs Eckverschiebungen die drei Dehnungen."""
    tab = [[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
           [0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
           [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]]
    for ecke in range(3):
        tab[0][2 * ecke] = b[ecke]
        tab[1][2 * ecke + 1] = c[ecke]
        tab[2][2 * ecke] = c[ecke]
        tab[2][2 * ecke + 1] = b[ecke]
    return tab


def assembliere(knoten, dreiecke, nu, delta_t):
```

```
    """Baut Systemmatrix und thermische rechte Seite."""
    n = 2 * len(knoten)
    matrix = []
    for zeile in range(n):
        neue = []
        for spalte in range(n):
            neue.append(0.0)
        matrix.append(neue)
    kraefte = []
    for i in range(n):
        kraefte.append(0.0)
    for nummer in range(len(dreiecke)):
        d = dreiecke[nummer]
        ecken = [knoten[d[0]], knoten[d[1]], knoten[d[2]]]
        c_tab, b, c, flaeche = material_und_steigungen(ecken, nu)
        bt = b_tabelle(b, c)
        # C*B
        cb = [[0.0] * 6, [0.0] * 6, [0.0] * 6]
        for zeile in range(3):
            for spalte in range(6):
                summe = 0.0
                for k in range(3):
                    summe = summe + c_tab[zeile][k] * bt[k][spalte]
                cb[zeile][spalte] = summe
        # Steckbrief 6x6 = B^T (C B) * Tiefe * Flaeche
        dofs = []
        for ecke in range(3):
            dofs.append(2 * d[ecke])
            dofs.append(2 * d[ecke] + 1)
        for a in range(6):
            for bb in range(6):
                wert = 0.0
                for k in range(3):
                    wert = wert + bt[k][a] * cb[k][bb]
                matrix[dofs[a]][dofs[bb]] += wert * TIEFE * flaeche
        # thermische Kraft: B^T C [aT, aT, 0] * Tiefe * Flaeche
        vor = ALPHA * delta_t[nummer]
        c_eps = [c_tab[0][0] * vor + c_tab[0][1] * vor,
                 c_tab[1][0] * vor + c_tab[1][1] * vor, 0.0]
        for a in range(6):
            wert = 0.0
            for k in range(3):
                wert = wert + bt[k][a] * c_eps[k]
```

```
            kraefte[dofs[a]] += wert * TIEFE * flaeche
    return matrix, kraefte


def loese_gauss(matrix, seite):
    """Gauss-Elimination (Kapitel 4), Inline-Kopie fuer den Browser."""
    n = len(matrix)
    werk = []
    for zeile in matrix:
        werk.append(list(zeile))
    s = list(seite)
    for spalte in range(n):
        piv = werk[spalte][spalte]
        for zeile in range(spalte + 1, n):
            f = werk[zeile][spalte] / piv
            for k in range(spalte, n):
                werk[zeile][k] = werk[zeile][k] - f * werk[spalte][k]
            s[zeile] = s[zeile] - f * s[spalte]
    x = []
    for i in range(n):
        x.append(0.0)
    for zeile in range(n - 1, -1, -1):
        summe = s[zeile]
        for k in range(zeile + 1, n):
            summe = summe - werk[zeile][k] * x[k]
        x[zeile] = summe / werk[zeile][zeile]
    return x


# --- Rechnen ---
knoten, dreiecke = baue_netz(NX, NY)
delta_t = []
for d in dreiecke:
    mittel = 0.0
    for k in range(3):
        y = knoten[d[k]][1]
        mittel = mittel + (TEMPERATUR_UNTEN
                           + (TEMPERATUR_OBEN - TEMPERATUR_UNTEN) * y / HOEHE)
    delta_t.append(mittel / 3.0 - REFERENZ)
matrix, kraefte = assembliere(knoten, dreiecke, NU, delta_t)

verschiebungen = []
for i in range(2 * len(knoten)):
```

```
    verschiebungen.append(0.0)

if LAGERUNG == "frei":
    print("LAGERUNG = 'frei': kein Lager -> der Träger SCHWIMMT.")
    print("Drei Starrkoerpermoden (schieben x, schieben y, drehen) sind")
    print("kraftfrei -- das System ist nicht loesbar (Gauss teilt durch 0).")
    print("Setze LAGERUNG = 'links_fest', um den Träger festzuhalten.")
else:
    lager = {}
    for i in range(len(knoten)):
        if abs(knoten[i][0]) < 1e-12:
            lager[2 * i] = 0.0
            lager[2 * i + 1] = 0.0
    frei = []
    for dof in range(2 * len(knoten)):
        if dof not in lager:
            frei.append(dof)
    red = []
    for i in frei:
        zeile = []
        for j in frei:
            zeile.append(matrix[i][j])
        red.append(zeile)
    rs = []
    for i in frei:
        rs.append(kraefte[i])
    loesung = loese_gauss(red, rs)
    for stelle in range(len(frei)):
        verschiebungen[frei[stelle]] = loesung[stelle]

    # freie Endverschiebung (unten rechts) und Handformel.
    w_fem = 0.0
    for i in range(len(knoten)):
        if abs(knoten[i][0] - LAENGE) < 1e-9 and abs(knoten[i][1]) < 1e-9:
            w_fem = verschiebungen[2 * i + 1] * 1000.0
    dt_hoehe = TEMPERATUR_UNTEN - TEMPERATUR_OBEN
    if abs(dt_hoehe) > 1e-9:
        radius = HOEHE / (ALPHA * dt_hoehe)
        w_hand = LAENGE * LAENGE / (2.0 * radius) * 1000.0
    else:
        w_hand = 0.0
    print("Netz %dx%d,  NU = %.2f,  Gefaelle unten %.0f / oben %.0f Grad"
          % (NX, NY, NU, TEMPERATUR_UNTEN, TEMPERATUR_OBEN))
```

```
    print("Hebung Ende (FEM):           %6.2f mm nach oben" % w_fem)
    print("Handformel  L^2 / (2R):      %6.2f mm nach oben" % w_hand)
    if abs(w_hand) > 1e-9:
        print("Anteil der Handformel:       %5.0f %% (grobe P1-Netze zu steif)"
              % (100.0 * w_fem / w_hand))

# --- Abbildung ---
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(8.0, 2.4))
bx = []
by = []
for (x, y) in knoten:
    bx.append(x)
    by.append(y)
if LAGERUNG == "frei":
    ax.triplot(bx, by, dreiecke, color="0.6", lw=0.9)
    ax.set_title("frei gelagert: der Träger schwimmt (nicht gerechnet)")
else:
    vx = []
    vy = []
    for i in range(len(knoten)):
        vx.append(bx[i] + UEBERHOEHUNG * verschiebungen[2 * i])
        vy.append(by[i] + UEBERHOEHUNG * verschiebungen[2 * i + 1])
    ax.triplot(bx, by, dreiecke, color="0.85", lw=0.6)
    ax.triplot(vx, vy, dreiecke, color="tab:blue", lw=0.9)
    ax.set_title("Biegebank (x %d ueberhoeht)" % int(UEBERHOEHUNG))
ax.set_aspect("equal")
ax.set_xlabel("Laenge x (m)")
plt.tight_layout()
plt.show()
```

```
import matplotlib.pyplot as plt
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from elastik2d import (assembliere_elastik, assembliere_thermokraefte,
                       loese_elastik)
from dreiecke import strukturiertes_dreiecksnetz

E, NU, TIEFE, ALPHA = 210e9, 0.30, 0.1, 12e-6
L, H, REF = 1.0, 0.1, 20.0
knoten, dreiecke = strukturiertes_dreiecksnetz(12, 3, L, H)
delta_t = []
for d in dreiecke:
```

```
    mittel = sum(220.0 - 200.0 * knoten[d[k]][1] / H for k in range(3)) / 3.0
    delta_t.append(mittel - REF)
K = assembliere_elastik(knoten, dreiecke, E, NU, TIEFE)
F = assembliere_thermokraefte(knoten, dreiecke, E, NU, TIEFE, ALPHA, delta_t)
lager = {}
for i, (x, y) in enumerate(knoten):
    if abs(x) < 1e-12:
        lager[2 * i] = 0.0
        lager[2 * i + 1] = 0.0
u = loese_elastik(K, F, lager)

UEBER = 8.0
bx = [p[0] for p in knoten]
by = [p[1] for p in knoten]
vx = [bx[i] + UEBER * u[2 * i] for i in range(len(knoten))]
vy = [by[i] + UEBER * u[2 * i + 1] for i in range(len(knoten))]
fig, ax = plt.subplots(figsize=(8.0, 2.4))
ax.triplot(bx, by, dreiecke, color="0.85", lw=0.6)
ax.triplot(vx, vy, dreiecke, color="tab:blue", lw=0.9)
ax.set_title("Biegebank (× 8 überhöht)")
ax.set_aspect("equal")
ax.set_xlabel("Länge x (m)")
plt.tight_layout()
plt.show()
```

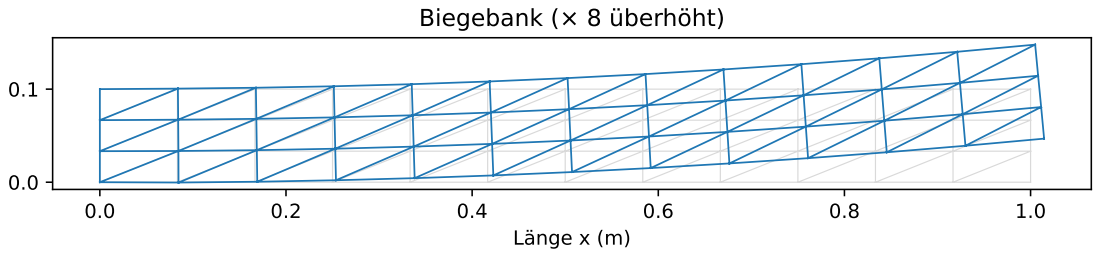

Abbildung 14.10.: Vorgerenderte Fassung der Biegebank (Kragträger, Gefälle unten 220 → oben 20 °C, $\nu = 0{,}30$, Netz 12×3, × 8 überhöht): grau das ruhende Netz, blau der nach oben gewölbte Träger. Im Browser wird dieses Bild durch dein Ergebnis ersetzt — verstelle die Regler und beobachte, wie sich Krümmung, Hebung und der Handformel-Vergleich ändern.

Auflösung des Vorhersage-Punkts. Überraschung: Für die **Krümmung**

selbst macht die Querkontraktion so gut wie **nichts** aus. Schau auf die Handformel — $\kappa = \alpha\,\Delta T/H$ enthält kein ν. Ein frei gebogener Träger krümmt sich bei $\nu = 0$ genauso stark wie bei $\nu = 0{,}30$; das kannst du prüfen, indem du in Übung 14.3 die „gerade genug"-Lagerung benutzt — dort ändert sich die Krümmung beim Verstellen von `NU` praktisch nicht.

In der **eingespannten** Biegebank oben siehst du trotzdem eine kleine Änderung: Bei `NU = 0` hebt sich das Ende etwas *weniger* (gut ein Zehntel weniger). Das liegt aber nicht an der Krümmung, sondern an der festen Einspannung: Die feste linke Kante verwandelt einen Teil der seitlichen Querkontraktions-Zerrung in zusätzlichen Biegeantrieb, und *dieser* Zusatz wächst mit ν. Nimm die Einspannung weg (freie Biegung), und ν verschwindet aus der Antwort. Die Lehre ist fein, aber wichtig: Die neue Zutat des Kapitels macht die 2D-Buchhaltung reicher, aber sie ist **nicht** der Motor der Krümmung — der Motor ist und bleibt der Längenunterschied $\alpha\,\Delta T$ zwischen heißer und kalter Faser.

Probiere zum Schluss noch `LAGERUNG = "frei"`: Das Programm weigert sich zu rechnen und erklärt, warum — der Träger schwimmt (das Starrkörper-Ballett aus Schritt 6). Und dreh `TEMPERATUR_UNTEN` höher: Die Hebung wächst genau proportional mit (Übung 14.2).

14.10. Das Kapitel-Programm

Das vollständige, eigenständige Programm zu diesem Kapitel ist `programme/kap14/kap14_b`. Es bündelt die Bausteine aus `programme/gemeinsam/elastik2d.py` (Steckbrief, thermische Kräfte, Assemblierung, Lösen mit Lagerung, Spannungen, Starrkörper-Probe), rechnet den gekrümmten Kragträger, führt die **Netzkonvergenz** gegen die Handformel vor, prüft den **2D-Nulltest** (frei, gleichmäßig warm → Spannung null) und speichert das Ergebnisbild. Die interaktive Biegebank oben ist seine im Browser lauffähige Kurzfassung.

Merkkasten

- In der Fläche trägt jeder Knoten **zwei** Freiheitsgrade (u, v); Knoten i besetzt die Zeilen $2i$ und $2i+1$. Das System wächst von n auf $2n$.
- Es gibt **drei Dehnungen**: strecken in x (ε_{xx}), strecken in y (ε_{yy}), den Winkel kippen (Schub γ_{xy}). Zusammen das vollständige Messprotokoll der Verformung.
- Die **Querkontraktion** ν (Radiergummi wird beim Drücken dicker) koppelt x und y; sie steckt in der 3×3-Materialtabelle des **ebenen**

Spannungszustands. Für unseren dünnen Träger ist „eben" die richtige Wahl.

- Die **6×6-Elementmatrix** entsteht aus den Zeltflächen-Steigungen (Kapitel 10): Eckverschiebungen → Dehnungen (konstant je Dreieck!) → Spannungen → Kräfte. Man gibt sie als Rezept und **liest** sie, statt sie von Hand herzuleiten.
- Ein Träger **ohne Lager** schwimmt in drei **Starrkörpermoden** (2× schieben, 1× drehen) — die 2D-Fassung des schwimmenden Trägers aus Kapitel 8.
- Ein **Temperaturgefälle über die Höhe** krümmt den Träger zum Kreisbogen, $\kappa = \alpha\,\Delta T_{\text{Höhe}}/H$ — **ohne** ν. Grobe P1-Netze biegen zu steif; die Durchbiegung konvergiert **von unten** gegen die Handformel.

Roter Faden

Wo kam das schon vor, wo kommt es wieder? — **Zurück:** Das Netz und die Zeltflächen-Steigungen sind die aus 1, die Assemblierung die aus Kapitel 3, der Löser der aus 2. Die thermische Kraft und die Einweg-Kopplung sind Kapitel 13 in der Fläche; die Analogie-Tafel jenes Kapitels wächst um die Querkontraktion. Der schwimmende Träger aus Kapitel 8 kehrt als Starrkörpermode wieder — jetzt dreifach. Das Verifikations-Ritual (Handformel, Netzkonvergenz, Nulltest) ist die Prüfkultur aus Kapitel 12, angewandt auf die Verformung. Und der Bimetall-Cliffhanger aus Übung 13.3 ist eingelöst: Nebeneinander liegende, verschieden heiße Fasern krümmen den Träger. — **Nach vorn:** 3 verschmilzt die transiente Wärme aus Kapitel 9 mit dieser Mechanik zum vollständigen, interaktiven Finale — dem Programm aus Kapitel 1, jetzt mit offenem Motorraum.

Übungen

Ü 14.1 (Verstehen). Unten sind drei verzogene Gitterquadrate skizziert (grau der Ausgang, blau die Verformung). Ordne jedem zu, welche der drei Dehnungen ungleich null sind — nur ε_{xx}, nur ε_{yy}, nur γ_{xy}, oder eine **Mischung**. (a) Ein Quadrat, das zum liegenden Rechteck wird. (b) Ein Quadrat, das zum Parallelogramm kippt, ohne breiter oder höher zu werden. (c) Ein Quadrat, das gleichzeitig höher wird **und** kippt.

Musterlösung zu Ü 14.1

(a) Nur $\varepsilon_{xx} \neq 0$ (breiter, sonst nichts). (b) Nur $\gamma_{xy} \neq 0$ (reiner Schub: Winkel kippt, Seitenlängen bleiben). (c) Eine **Mischung** aus ε_{yy} (höher) und γ_{xy} (kippt) — zwei der drei zugleich, $\varepsilon_{xx} = 0$ (nicht breiter). Die Kunst ist zu sehen, dass „höher **und** schräg" zwei getrennte Zutaten sind, keine einzige neue.

Ü 14.2 (Verändern). Verdopple in der Biegebank die Temperaturdifferenz über die Höhe (z. B. von $\Delta T = 200$ auf 400 K, indem du `TEMPERATUR_UNTEN` von 220 auf 420 °C stellst). Verdoppelt sich die Hebung des freien Endes? Und warum gilt das hier *exakt*, obwohl in Kapitel 8 die Mittentemperatur bei doppelter Brandeinwirkung gerade **nicht** einfach mitzog? Lösungsskript: `loesungen/kap14_ue2.py`.

Musterlösung zu Ü 14.2

Ja — die Hebung verdoppelt sich exakt. Die Krümmung ist $\kappa = \alpha\,\Delta T_{\text{Höhe}}/H$, also **linear** im Temperaturunterschied; doppeltes ΔT heißt doppeltes κ heißt doppelte Hebung. Das ganze Modell dieses Kapitels ist **linear**: doppelte Last, doppelte Antwort. In Kapitel 8 war das anders, weil dort die *stationäre Temperatur selbst* gesucht war und die Randbedingungen (das fest gehaltene $20\,°\text{C}$-Auflager) nicht mitskalierten — eine doppelt so heiße Brandeinwirkung ergibt eben kein doppelt so heißes Profil, wenn ein Ende festgenagelt bleibt. Hier dagegen ist das Temperaturfeld **fest vorgegeben** (die Einweg-Kopplung), und die Mechanik darauf reagiert streng proportional. Merke: Linearität gilt, wo Ursache und Wirkung durch eine Geradengleichung verbunden sind — und die Thermoelastik dieses Buches ist durchweg so gebaut.

Ü 14.3 (Übertragen). Wiederhole den Nulltest aus Kapitel 13 in 2D: Ein **frei** gelagerter, **gleichmäßig** (nicht über die Höhe gestaffelt!) um 100 K erwärmter Träger muss **Spannung null** liefern — er darf sich frei in alle Richtungen ausdehnen. Der Haken: „Frei" heißt hier nicht „gar kein Lager" (dann schwimmt er, Schritt 6), sondern **gerade genug** Lager, um die drei Starrkörpermoden zu sperren, ohne die Dehnung zu behindern: **ein** Knoten ganz fest ($u = v = 0$) und **ein zweiter** nur in y gehalten. Führe das Nulltest-Protokoll und lies die größte Spannung ab. Lösungsskript: `loesungen/kap14_ue3.py`.

Musterlösung zu Ü 14.3

Mit der „gerade genug"-Lagerung (Knoten unten links fest, Knoten unten rechts nur in y) ist die größte Spannung im ganzen Träger null — auf Maschinengenauigkeit (Größenordnung 10^{-12} N/mm^2). Der Grund ist derselbe wie in 1D: Die gleichmäßige Wärmedehnung ist eine **konstante** Dehnung $[\alpha\,\Delta T, \alpha\,\Delta T, 0]$, und eine konstante Dehnung liegt exakt im Ansatzraum der P1-Dreiecke — kein Diskretisierungsfehler, keine Spannung, auf *jedem* Netz. (Anders als bei der **Biegung** in Schritt 7: Dort ist die Dehnung über die Höhe *veränderlich*, das können die konstanten P1-Dreiecke nicht exakt, und darum ist die Biegung *kein* Nulltest, sondern konvergiert nur.) Zwei Lagerbedingungen zu wenig, und der Träger schwimmt; eine zu viel (etwa beide Knoten ganz fest), und man behindert die Dehnung und erzeugt künstliche Spannung. „Gerade genug festhalten" ist die Kunst — statisch bestimmt lagern.

Ü 14.4 (Übertragen). *Die Betonbrücke in der Sonne.* Dieselbe Krümmungsformel, ein ganz anderer Schauplatz: eine Betonbrücke, deren Fahrbahnplatte an einem Sommertag von oben von der Sonne beschienen wird, während die Unterseite im Schatten kühl bleibt. Über die $H = 0{,}3$ m dicke Platte stellt sich dann ein Temperaturgefälle ein — Brückennormen und Messungen setzen dafür größenordnungsmäßig $\Delta T \approx 12$ K (oben wärmer) an. Beton dehnt sich mit $\alpha_{\text{Beton}} \approx 10 \cdot 10^{-6}$ 1/K. (a) Berechne die Krümmung $\kappa = \alpha\,\Delta T / H$ und den Krümmungsradius. (b) Ein Brückenfeld der Spannweite $L = 30$ m liegt an beiden Enden auf Lagern. Bei konstanter Krümmung hebt sich seine Mitte gegenüber der Verbindungslinie der Lager um $w_{\text{Mitte}} = \kappa L^2/8$ (die Auflagerformel, wie sie ein durchhängendes Seil spiegelbildlich erfüllt) — schätze diese Hebung ab. Lösungsskript: `loesungen/kap14_ue4.py`.

Musterlösung zu Ü 14.4

(a) $\kappa = \alpha\,\Delta T/H = 10 \cdot 10^{-6} \cdot 12/0{,}3 = 4{,}0 \cdot 10^{-4}$ 1/m, also ein Krümmungsradius $R = 1/\kappa = 2500$ m — ein sehr flacher Bogen. (b) Für das beidseitig gelagerte Feld ist $w_{\text{Mitte}} = \kappa L^2/8 = 4{,}0 \cdot 10^{-4} \cdot 30^2/8 = 0{,}045$ m $= 45$ mm: Die Feldmitte hebt sich um gut vier Zentimeter. Genau darum bekommen Brücken Bewegungslager und Dehnfugen — das Bauwerk *muss* atmen dürfen.

Bemerkenswert ist, dass die Sonne das bei Beton überhaupt schafft, bei einem Stahlträger derselben Dicke aber nicht: Ein Temperaturgefälle über die Höhe entsteht nur, wenn die Wärme *nicht* sofort quer durch-

sickert. Beton ist ein schlechter Wärmeleiter ($\lambda_{\text{Beton}} \approx 1{,}8\ \mathrm{W/(m\,K)}$, rund dreißigmal weniger als die $50\ \mathrm{W/(m\,K)}$ von Stahl), die Platte ist mit $0{,}3$ m dick, und die Sonne kommt im Tagesrhythmus, lädt tagsüber auf und lässt nachts wieder abkühlen. Diese drei Umstände — schlechter Leiter, großer Querschnitt, langsamer Tageszyklus — halten den Gradienten aufrecht. Bei Stahl gliche sich derselbe Wärmeeintrag über die dünne Höhe fast augenblicklich aus; erst die geballte, intensive Einwirkung eines Brandes von unten erzeugt dort das steile Gefälle dieses Kapitels.

Das Kleingedruckte

Drei ehrliche Anmerkungen zum Schluss.

Die Farbbilder zeigen σ_{xx}, nicht die „Vergleichsspannung". Wir haben den Träger nach der Längsspannung σ_{xx} eingefärbt, weil sie anschaulich ist (Zug rot, Druck blau) und direkt die Biegung zeigt. In der Praxis fasst man die drei Spannungen $\sigma_{xx}, \sigma_{yy}, \tau_{xy}$ oft zu **einer** Zahl zusammen, der **von-Mises-Vergleichsspannung**, die angibt, wie nah das Material am Fließen ist. Das ist ein kleiner Rechenschritt mehr und ein eigenes Thema; unsere Farbbilder sagen ehrlich σ_{xx} dazu.

P1-Dreiecke sind schlechte Biege-Elemente. Wir haben es an der Netzkonvergenz gesehen und offen benannt: Weil ein P1-Dreieck nur *eine* Dehnung über seine Fläche kennt, bildet es Biegung nur grob und zu steif ab. Für ernsthafte Biegerechnungen nimmt man Elemente höherer Ordnung (quadratische Dreiecke mit Zwischenknoten) oder spezielle Biege-Elemente, die die veränderliche Dehnung über die Höhe von Haus aus darstellen. Unser P1-Netz ist ehrlich, aber nicht das schärfste Werkzeug — es zeigt die Physik richtig und die Zahl ungefähr.

Nur eine Richtung der Kopplung. Wie in Kapitel 13 rechnen wir die **Einweg-Kopplung**: Die Temperatur erzeugt die Verformung, aber die Verformung wirkt nicht auf die Temperatur zurück. Für unseren Träger ist dieser Rückkanal winzig; echte gekoppelte Probleme (die vollen thermomechanischen Prozesse) müssen beide Richtungen zugleich lösen. Das ist der Stoff jenseits dieses Buches — aber die Maschine, die du jetzt verstanden hast, ist genau die, auf der auch jene Programme aufbauen.

15. Das große Finale

Erinnerst du dich an Kapitel 1? Ein Stahlträger unter einer Betondecke, links im kühlen Auflager der Außenwand, und unten rechts ein Brandherd. Drei Regler — die Temperatur der Brandgase, die Netzfeinheit, die Zeit — , ein Knopfdruck, und ein Farbfilm zeigte, wie die Wärme vom Feuer her durch den Träger kroch und wie er sich dabei krümmte. Damals war das eine **Blackbox**: Du durftest daran drehen, aber der Motorraum war verschweißt. „Jede der rund vierhundert Zeilen dahinter", hieß es, „öffnen wir bis zum Schluss."

Jetzt ist der Schluss da. In diesem Kapitel steht dasselbe Programm noch einmal vor dir — mit **offener Motorhaube**. Wir bauen es aus Teilen zusammen, die du alle schon geschrieben hast: den Löser aus 2, die Dreiecke aus 1, die Rechtecke aus Kapitel 11, die Zeitschleife aus Kapitel 9, die Thermoelastik aus Kapitel 14. Kein einziger neuer Baustein kommt hinzu. Das Finale erklärt nichts Neues — es ist eine **geführte Werksbesichtigung** und danach eine Experiment-Werkstatt, in der du mit allem, was du gelernt hast, gezielte Versuche fährst.

Lernziele

Nach diesem Kapitel kannst du …

1. … die **Architektur** eines vollständigen FEM-Programms aus dem Gedächtnis zeichnen (Netz → Assemblierung → Randbedingungen → Zeitschleife → Löser → Kopplung → Ausgabe) und jedes Modul einem Buchkapitel zuordnen,
2. … ein gezieltes **Experiment entwerfen**: eine Frage stellen, die Regler wählen, das Ergebnis gegen eine Vorhersage prüfen,
3. … den Einfluss von **Netzdichte, Zeitschritt und Randbedingungen** auf ein gekoppeltes Ergebnis trennen — welcher Regler wirkt worauf,
4. … die **Grenzen** dieses Programms benennen (2D, linear, isotrop, Einweg-Kopplung, kleine Netze) und sagen, welches Werkzeug jenseits davon weiterhilft,
5. … jedes Ergebnis mit der **Simulations-Checkliste** aus Kapitel 12 absichern, bevor du ihm glaubst.

⚠ Naheliegende Vermutung

Vermutung: *„Ein ‚richtiges' FEM-Programm — OpenGeoSys, Abaqus, ANSYS — ist etwas grundlegend anderes als unsere Kapitel-Programme. Das hier war didaktisches Spielzeug; die Profis machen es völlig anders."*
Warum sie naheliegt: Die Profi-Programme haben Millionen Codezeilen, Handbücher wie Telefonbücher und lösen Netze mit Milliarden Elementen. Dagegen wirken unsere paar hundert Zeilen reines Python wie ein Bastelmodell neben einem Flugzeug.
Was stattdessen stimmt: Sie haben **exakt dieselben Module** — Geometrie und Netz, Elementmatrizen, Assemblierung, Randbedingungen, Zeitschritte, Löser, Kopplung. Was sie unterscheidet, ist nicht die *Architektur*, sondern die *Ausstattung*: mehr Elementtypen, schlauere Löser, krumme Ränder, Jahrzehnte Feinschliff. Wer dieses Finale versteht, liest ihre Handbücher als **Landkarte** statt als Fremdsprache — jeder Fachbegriff darin hat in diesem Buch seinen Platz. Genau das prüfen wir am Ende an OpenGeoSys nach.

15.1. Die Werksbesichtigung: das Programm als Fließband

Das vollständige Programm liegt als `programme/kap15/fem_balken.py` neben diesem Kapitel — rund vierhundert Zeilen reines Python, jede Funktion mit einem Kapitel-Rückverweis in ihrem Docstring. Bevor wir einzelne Module aufklappen, hier das ganze Werk aus der Vogelperspektive.

Abbildung 15.1 zeigt es als zwei ineinandergreifende Fließbänder. Oben läuft die **Zeitschleife** der Wärme (aus Kapitel 9): Sie assembliert einmal die Leitungsmatrix und die Wärmeeimer und lässt dann die Wärmefront Schritt für Schritt weiterkriechen. An den gewünschten Ausgabeschritten reicht sie das aktuelle Temperaturfeld nach unten an das **Kopplungs-Fließband** der Mechanik (aus Kapitel 14), das daraus Verformung und Spannung rechnet. Daneben läuft, ohne je einzugreifen, der **Energiebilanz-Wachhund** aus Kapitel 12 mit.

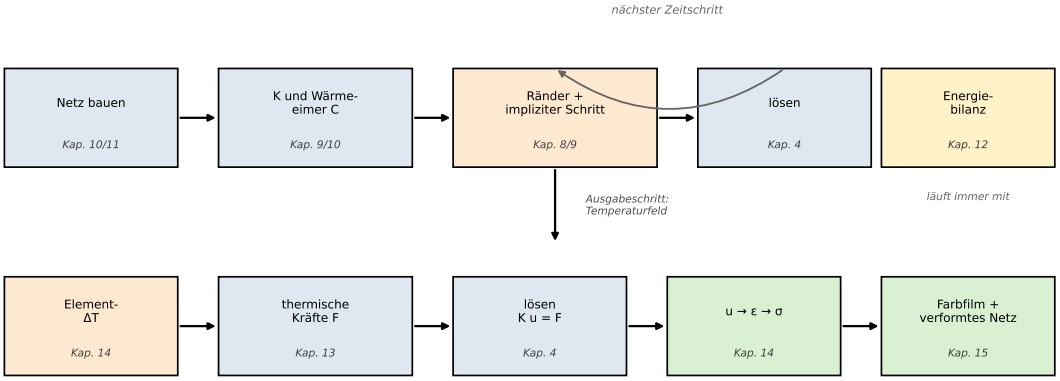

Abbildung 15.1.: Das ganze Finale-Programm als Fließband. Oben die Zeitschleife der Wärme (Kapitel 9): einmal das Netz bauen (Kapitel 10/11), Leitungsmatrix und Wärmeeimer assemblieren, dann in jedem Zeitschritt die Ränder einbauen (Kapitel 8) und implizit lösen (Kapitel 4). An den Ausgabeschritten fällt das Temperaturfeld nach unten in die Kopplungskette der Mechanik (Kapitel 14): Element-ΔT → thermische Kräfte → lösen → Spannung → Bild. Rechts wacht die Energiebilanz (Kapitel 12). Jeder Kasten trägt seine Kapitelnummer — die ganze Reise auf einen Blick.

Was man hier sieht: die ganze Reise dieses Buches auf einem Bild. Von links nach rechts, von oben nach unten fließt eine gekoppelte Simulation — und jeder Kasten trägt die Nummer des Kapitels, das ihn gebaut hat. Es gibt keinen Kasten ohne Kapitel. Genau das ist die Botschaft des Finales: Du kennst jedes Glied.

15.1.1. Fünf Module, fünf Ausschnitte

Jetzt klappen wir den Motorraum auf. Für fünf Kernmodule zeigen wir den entscheidenden Ausschnitt aus `fem_balken.py` und sagen in einem Absatz, *warum er genau so aussieht*. **Dies ist die einzige Stelle im ganzen Buch, an der Code im Mittelpunkt steht** — nicht, weil wir hier etwas erklären (das ist alles längst geschehen), sondern weil der Code selbst hier der Lerngegenstand ist: der Beweis, dass alles zusammenpasst.

Modul 1 — die Wärmeeimer (`gelumpte_kapazitaeten`, ab Zeile 191). Jedes Element verteilt seine Wärmekapazität $\rho\, c \cdot$ Fläche $\cdot$ Tiefe zu gleichen Teilen

auf seine Knoten — beim Dreieck auf drei, beim Rechteck auf vier. Das ist das **Lumping** aus Kapitel 9, nur eben in der Fläche:

```
speicher = material["dichte"] * material["kapazitaet"]
for element in elemente:
    gesamt = speicher * elementflaeche(knoten, element) * TIEFE_M
    anteil = gesamt / len(element)          # Dreieck: /3, Rechteck: /4
    for ecke in element:
        kapazitaeten[ecke] = kapazitaeten[ecke] + anteil
```

Warum genau so: `len(element)` ist drei oder vier — dieselbe eine Zeile bedient beide Elementfamilien, weil die verteilte Wärmeträgheit in beiden Fällen einfach zu gleichen Eimern zusammengeklumpt wird. Kein Sonderfall, kein `if`.

Modul 2 — der implizite Zeitschritt mit allen Rändern (`ein_waermeschritt`, ab Zeile 217). Das Herzstück. Für jeden Knoten steht die Gleichung des impliziten Euler aus Kapitel 9,

$$\frac{C_i}{\Delta t}\,T_i^{\text{neu}} \;+\; \sum_j K_{ij}\,T_j^{\text{neu}} \;=\; \frac{C_i}{\Delta t}\,T_i^{\text{alt}} \;+\; \text{Last}_i,$$

nur ist die linke Seite jetzt kein tridiagonales Band mehr, sondern der volle 9-Punkt-Stern der Fläche — also lösen wir sie mit **Gauß-Seidel** aus 2 statt mit Thomas. Die vier Randtypen aus Kapitel 8 greifen genau dort an, wo sie dort hergeleitet wurden:

```
if typ == "leistung":                       # Neumann: Watt auf die rechte Seite
    rechte_seite[knoten_nr] += bedingung[1] * rand_flaeche / gesamt_flaeche
elif typ == "robin":                        # Brandgas/Luft: Zwitter (Kapitel 8)
    matrix[knoten_nr][knoten_nr] += alpha * rand_flaeche          # -> Diagonale
    rechte_seite[knoten_nr] += alpha * rand_flaeche * t_umgebung  # -> rechte Seite
elif typ == "temperatur":                   # Dirichlet: festhalten und streichen
    festwerte[knoten_nr] = bedingung[1]
# "isoliert": nichts tun -- der Fluss ist null.
```

Warum genau so: Der **Robin**-Rand ist im Finale kein Nebendarsteller mehr, sondern die tragende Randbedingung — mit ihm treffen die Brandgase auf den Träger. Er ist der Zwitter, den Kapitel 8 zwar beschrieben, aber nicht ausgerechnet hat: Der Übergangsstrom $Q = \alpha_c\,A\,(T_{\text{Gas}} - T)$ hängt von der gesuchten Temperatur T selbst ab. Deshalb wandert sein einer Teil ($\alpha_c A$) auf die **Diagonale der Matrix**, sein anderer ($\alpha_c A\,T_{\text{Gas}}$) auf die **rechte Seite**. Genau dieselbe Aufteilung trägt auch der `brandherd`-Rand der Unterseite

— er ist ein Robin-Rand mit *ortsabhängiger* Umgebung: unter dem rechten Bereich das heiße Brandgas, links die kühle Luft. Damit ist das Versprechen aus Kapitel 8 eingelöst. Die feste Temperatur des Auflagers (Dirichlet) dagegen greift nicht in die Matrix, sondern wird per Streich-Rezept aus Kapitel 8 herausgelöst — dieselbe Funktion `reduziere_system`, die schon die Dreiecke aus 1 benutzten.

Modul 3 — die Einweg-Kopplung (`koppel_mechanik`, ab Zeile 367). An den Ausgabeschritten übergibt die Wärme ihr Temperaturfeld an die Mechanik — und nur in diese eine Richtung (Kapitel 14):

```
delta_t = element_uebertemperaturen(temperaturen, dreiecke)   # Kap. 14
matrix = assembliere_elastik(knoten, dreiecke, e_modul, nu, TIEFE_M)
kraefte = assembliere_thermokraefte(knoten, dreiecke, e_modul, nu, TIEFE_M,
                                    alpha, delta_t)
verschiebungen = loese_elastik(matrix, kraefte, lager)        # Kap. 4/14
spannungen = elementspannungen(knoten, dreiecke, verschiebungen, ...)
```

Warum genau so: Es ist Wort für Wort die Rechenkette aus Kapitel 14 — `assembliere_elastik`, `assembliere_thermokraefte`, `loese_elastik`, `elementspannungen` sind dieselben Funktionen aus `programme/gemeinsam/elastik2d.py`, die dort das Kapitel getragen haben. Das Finale importiert sie, es kopiert sie nicht. Die **Einbahnstraße** (Wärme treibt Mechanik, nicht umgekehrt) steckt schon darin, dass wir erst die ganze Wärmerechnung fertig machen und *dann* ihr Ergebnis als feste Last einsetzen.

Modul 4 — der Wachhund (`nodale_leistungen`, ab Zeile 400). In jedem Schritt sammelt er die Leistung ein, die über die Ränder herein- oder hinausfließt — als Rest-Term der impliziten Gleichung:

```
leitungsfluss = 0.0
for j in range(knotenzahl):
    leitungsfluss = leitungsfluss + systemmatrix[i][j] * temperaturen_neu[j]
traegheit = kapazitaeten[i] / dt * (temperaturen_neu[i] - temperaturen_alt[i])
leistungen.append(traegheit + leitungsfluss)      # >0 herein, <0 hinaus
```

Warum genau so: Was Trägheit (Aufheizen) und Leitung zusammen verlangen, muss von außen über den Rand geliefert worden sein. Im Inneren und an isolierten Rändern ist dieser Rest null — nur wo ein Rand Wärme ein- oder ausspeist, steht eine Zahl. Am Ende muss **hereingeflossen = gespeichert + hinausgeflossen** aufgehen (Kapitel 12). Schließt die Bilanz nicht, steckt ein Fehler in der Assemblierung — der Wachhund bellt, ganz ohne Vergleichslösung.

Modul 5 — die Zeitschleife selbst (`simuliere`, ab Zeile 449). Sie hält alles zusammen und ist kurz genug, um sie ganz zu zeigen — das ist das ganze Fließband aus Abbildung 15.1 in Python:

```
for schritt in range(1, zeitschritte + 1):
    alt = temperaturen
    temperaturen = ein_waermeschritt(alt, systemmatrix, kapazitaeten, dt, ...)
    leistungen = nodale_leistungen(temperaturen, alt, systemmatrix, kapazitaeten, dt)
    # ... Wachhund-Summen aktualisieren ...
    if schritt in kopplungsschritte:
        verschiebungen, spannungen = koppel_mechanik(knoten, dreiecke,
                                                     temperaturen, material, lager)
        momentaufnahmen.append(...)
```

Warum genau so: Ein Zeitschritt der Wärme, dann der Wachhund, dann — nur an den Ausgabeschritten — ein Sprung in die Mechanik. Mehr ist es nicht. Vierhundert Zeilen, aber ihr Kern passt in zehn.

15.2. Die Blackbox aus Kapitel 1 — jetzt mit offenem Motorraum

In Kapitel 1 lief die Blackbox mit ihren drei Reglern. Was damals nicht zu sehen war: Ihr Maschinenraum war schon **genau dieses Programm** — die volle 2D-Kette aus Dreiecksnetz, Brand-Rändern und Thermoelastik, nur in einer kompakten Fassung, die das Rechenpaket NumPy als Rechenknecht benutzt, damit der Browser flott bleibt. `fem_balken.py` rechnet dieselbe Kette in reinem Python nach — Zeile für Zeile aus den Kapiteln dieses Buches. Stellen wir beide für dieselben Regler nebeneinander — Brandgase $500\,°\mathrm{C}$, mittleres Netz, 30 Minuten.

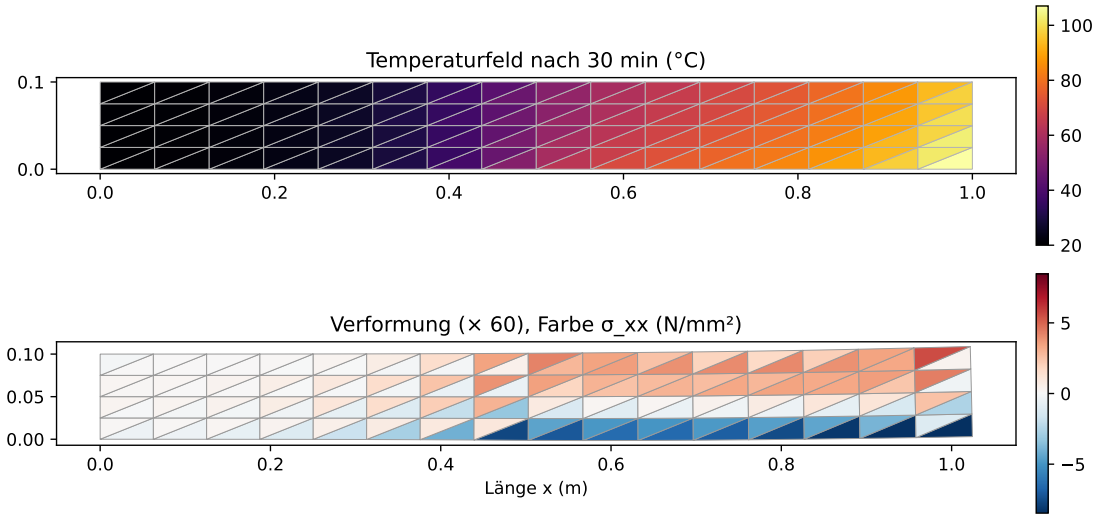

Abbildung 15.2.: Die Brand-Szene, echte Rechnung: oben das Temperaturfeld nach 30 min — die Brandgase (500 °C) wärmen die Unterseite und die rechte Stirn, das Auflager links und die Betondecke oben halten dagegen; die Mitte liegt erst bei rund 52 °C, das heißeste Eck bei etwa 112 °C (nichts glüht). Unten das verformte Netz (× 60 überhöht), eingefärbt nach σ_xx: Der heiße Untergurt dehnt sich stärker als der kühle Obergurt, also wölbt sich der Träger leicht nach oben (thermal bowing) — nach 30 min realistischer Konvektion erst um Bruchteile eines Millimeters.

Was man sieht: Die Brandgase heizen den Träger von unten und rechts, das Auflager und die Decke halten dagegen — nach 30 Minuten ist die Mitte erst rund 52 °C warm, das heißeste Eck knapp über 110 °C. Weil der Untergurt wärmer ist als der Obergurt, **wölbt sich der Träger leicht nach oben**. Aber die Wölbung ist winzig — bei realistischer Konvektion braucht es Bruchteile eines Millimeters, und das Bild muss sechzigfach überhöhen, um sie zu zeigen.

Blackbox vs. Finale-Programm — die Zahlen

Größe	Blackbox (Kapitel 1, NumPy)	`fem_balken.py` (reines Python)
Mittentemperatur	52 °C	52 °C
Heißester Punkt	112 °C	112 °C

Hebung des freien Endes	0,15 mm	0,15 mm

Identisch — und genau das ist die Pointe. Zwei unabhängig geschriebene Fassungen desselben Verfahrens, eine kompakt mit NumPy, eine ausführlich in reinem Python, liefern bis auf Rundung dieselben Zahlen. Das ist nebenbei ein waschechter **Verifikationstest** im Geist von Kapitel 12: Wer dasselbe Modell zweimal verschieden implementiert und zweimal dasselbe herausbekommt, hat eine der besten Proben bestanden, die es gibt. Der Unterschied zwischen Kapitel 1 und heute liegt also nicht im Ergebnis — er liegt darin, dass du jetzt **jede Zeile verstehst**.
Und die Physik dahinter bleibt bemerkenswert: Ein realistisches Konvektionsfeuer wölbt den Träger in einer halben Stunde erst um gut einen Zehntelmillimeter. Die **sichtbare, kräftige Krümmung** — bis zu den kanonischen 12 mm — entsteht erst, wenn der Brand voll entwickelt ist und die Unterseite selbst richtig heiß wird (Kapitel 14). Genau diesen Fall bauen wir gleich in Reihe A und in der Finale-Animation.

15.3. Die Experiment-Werkstatt

Von hier an ist das Kapitel eine Werkstatt. Ein FEM-Programm ist kein Orakel, dem man eine Frage stellt und eine Wahrheit erhält — es ist ein **Instrument**, mit dem man Experimente fährt. Und Experimente haben eine feste Form, damit am Ende ein Erkenntnisgewinn steht und nicht bloß ein weiteres buntes Bild. Abbildung 15.3 zeigt den Rahmen, den wir für jede Versuchsreihe benutzen.

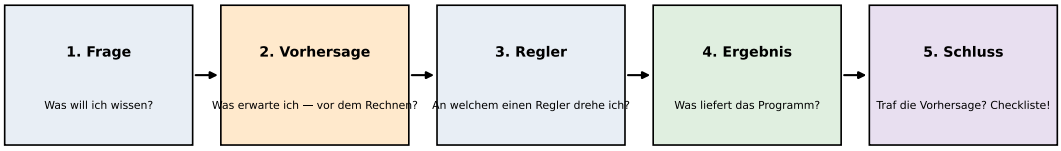

Abbildung 15.3.: Der Protokollrahmen jeder Versuchsreihe. Erst die Frage, dann eine ehrliche Vorhersage (bevor man rechnet!), dann der eine Regler, an dem man dreht, dann das Ergebnis — und zuletzt der Schluss, geprüft gegen die Simulations-Checkliste aus Kapitel 12. Ohne Vorhersage ist es kein Experiment, sondern nur Herumklicken.

Was man sieht: fünf Schritte, und der zweite ist der wichtigste. Die **Vorhersage vor dem Rechnen** ist es, die aus Klicken ein Experiment macht — nur wer sich festlegt, kann überrascht werden, und nur wer überrascht wird, lernt. Wir fahren jetzt fünf Reihen: Netz, Zeit, Ränder, Material, Elemente. (Damit lösen wir zugleich Übung 1.2 ein — dort solltest du Regler-Experimente durchführen und protokollieren; hier ist die Vorlage.)

15.3.1. Reihe A — Das Netz: wann darf ich der Zahl glauben?

Frage. Wenn ich das Netz feiner mache, ändert sich das Ergebnis. Ab wann ändert es sich nicht mehr — wann ist das Netz fein genug? Wir nehmen den voll entwickelten Brand: Die Flammen halten die Unterseite auf $220\,°\mathrm{C}$, die Betondecke die Oberseite auf $20\,°\mathrm{C}$ (Gefälle $\Delta T = 200$ K über die Höhe, links eingespannt), und messen die Auslenkung des freien Endes auf immer feineren Netzen.

Vorhersage. Aus Kapitel 14 kennen wir die Handformel für dieses Gefälle: Krümmung $\kappa = \alpha\,\Delta T/H = 12 \cdot 10^{-6} \cdot 200/0{,}1 = 0{,}024\ 1/\mathrm{m}$, Kragarm-Auslenkung $w = L^2\kappa/2 = 12{,}0$ mm nach oben. Und wir wissen: Grobe Dreiecksnetze biegen zu **steif** — die Auslenkung nähert sich der 12 mm *von unten*. Wir erwarten also eine Zahlenreihe, die wächst und sich langsam einpendelt.

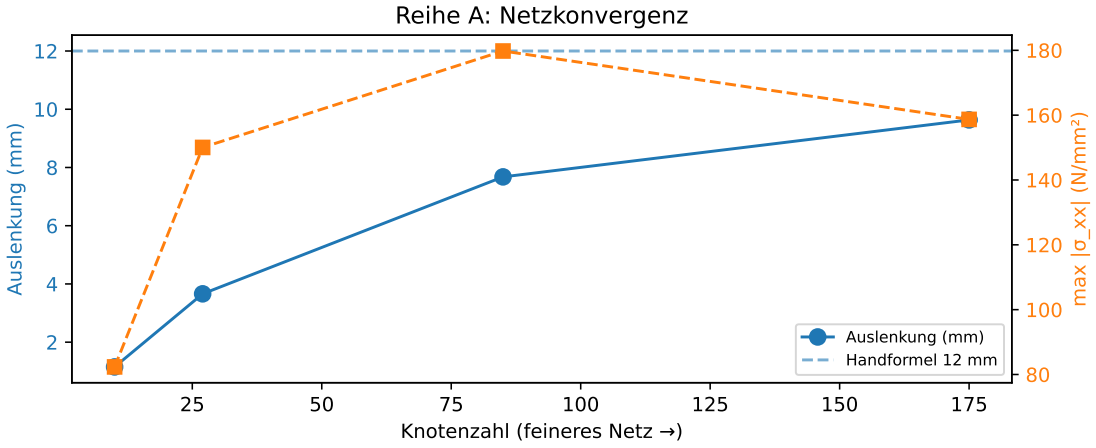

Abbildung 15.4.: Netzkonvergenz der Auslenkung (voll entwickelter Brand, Gefälle 20 → 220 °C über die Höhe). Blaue Punkte: die FEM-Auslenkung wächst mit feinerem Netz und nähert sich der gestrichelten Handformel-Vorhersage (12 mm) von unten — die grobe CST-Rechnung ist zu steif (Kapitel 14). Die Maximalspannung (orange) dagegen pendelt: Sie sitzt an der Einspann-Ecke, einer Spannungsspitze, die ein grobes Netz nie ganz einfängt (Kapitel 12).

Ergebnis. Die Auslenkung wächst brav — von rund 1 mm auf dem gröbsten Netz (4×1) über $3{,}7$ und $7{,}7$ bis auf $9{,}6$ mm auf dem feinsten (24×6), immer näher an die 12 mm. Die **Maximalspannung** dagegen pendelt ($82 \to 150 \to 180 \to 159$ N/mm^2) und will sich nicht einpendeln.

Schluss. Für die Auslenkung ist das Netz 16×4 brauchbar (rund $64\ \%$ der Handformel), 24×6 besser ($80\ \%$); ganz konvergiert ist sie noch nicht, für den qualitativen Eindruck aber gut. Die Spannung ist die härtere Nuss: Ihr Höchstwert sitzt an der **Einspann-Ecke**, wo die Lagerung einen scharfen Knick erzwingt — eine Spannungsspitze, die ein Dreiecksnetz mit konstanter Spannung je Element nie ganz einfängt (das ist dieselbe **Kerbwirkung**, die in Übung 15.2 das Loch erzeugt). Merke: **Verschiebungen konvergieren schneller als Spannungen, und Spannungsspitzen an Ecken konvergieren gar nicht** — genau davor warnt die Konvergenzprobe aus Kapitel 12.

15.3.2. Reihe B — Die Zeit: warum implizit, und wie groß darf Δt sein?

Frage. In Kapitel 9 explodierte der *explizite* Zeitschritt, sobald Δt zu groß wurde. Unser Finale rechnet **implizit**. Was passiert, wenn wir hier Δt vergrößern — explodiert es auch?

Vorhersage. Der implizite Euler war *immer* stabil (Kapitel 9). Wir erwarten also **keine Explosion**, egal wie groß Δt ist — aber grobe Schritte sollten die Wärmefront **verschmieren**, die Mittentemperatur also unterschätzen.

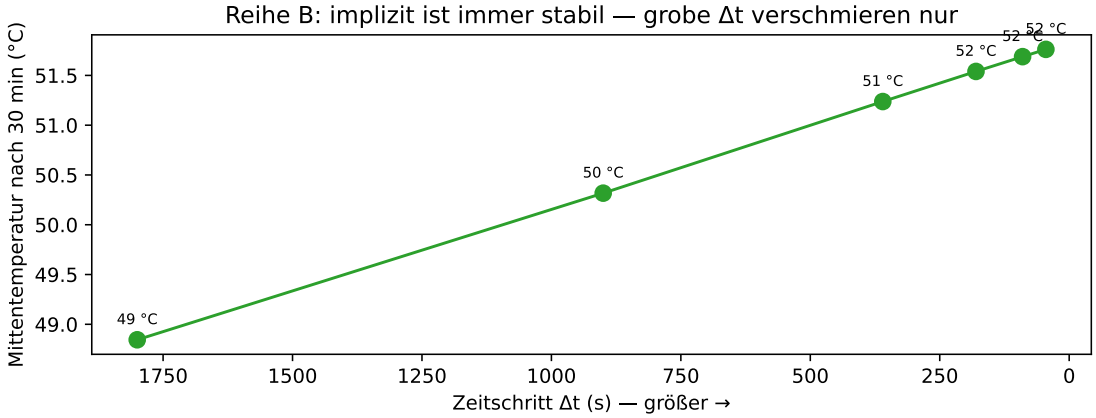

Abbildung 15.5.: Reihe B: dieselbe Simulation (Brand-Szene, 30 min) mit immer weniger, dafür größeren impliziten Zeitschritten. Nichts explodiert — auch der eine Riesenschritt (Δt = 1800 s) liefert eine brave Zahl. Aber die groben Schritte unterschätzen die Mittentemperatur, weil sie die wandernde Front verschmieren; mit feinerem Δt pendelt sich der Wert ein. Der Gegensatz zum explodierenden expliziten Euler aus Kapitel 9.

Ergebnis. Von rund $52\,°\mathrm{C}$ bei feinen Schritten fällt die Mittentemperatur auf $49\,°\mathrm{C}$ beim einen Riesenschritt ($\Delta t = 1800$ s) — aber **nichts explodiert**. Der grobe Schritt ist ungenau, nicht instabil.

Schluss. Genau das ist der Grund, warum das Finale (und jedes ernsthafte transiente Programm) implizit rechnet: Man darf große Zeitschritte nehmen, ohne dass die Rechnung entgleist — man zahlt nur mit Genauigkeit, nicht mit einer Katastrophe. Die Faustregel bleibt die Konvergenzprobe aus Kapitel 12: Δt halbieren; ändert sich das Ergebnis kaum noch, war es fein genug. (Beim impliziten Euler halbiert sich der Zeitfehler beim Halbieren von Δt — „Zeit ist gröber als Raum".)

15.3.3. Reihe C — Die Ränder: drei Flammen, drei Geschichten

Frage. Die Brandgase an der rechten Stirn lassen sich auf drei Arten modellieren (Kapitel 8): als **feste Temperatur** (Dirichlet), als **eingespeiste Leistung** (Neumann) oder als **Übergang** an ein Gas fester Temperatur (Robin). Wie unterscheiden sich die drei?

Vorhersage. Die feste Temperatur deckelt das Ende bei $500\,°\mathrm{C}$. Die feste Leistung kennt keine solche Grenze — das Ende wird so heiß, wie die eingespeisten Watt es treiben, und kann $500\,°\mathrm{C}$ **überschreiten**. Der Robin-Rand dagegen kann den Träger höchstens auf die Gastemperatur bringen, nie darüber: Sobald die Oberfläche das Gas eingeholt hat, versiegt der antreibende Temperaturunterschied.

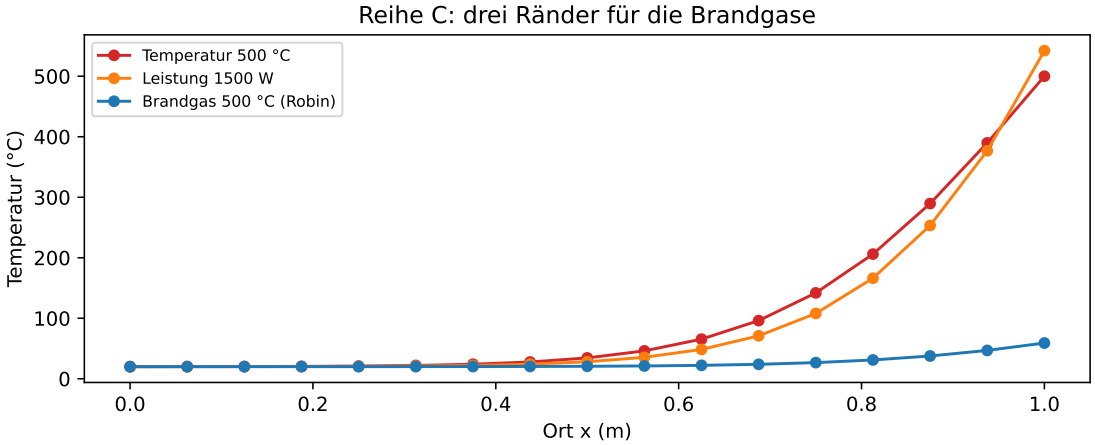

Abbildung 15.6.: Reihe C: drei Modellierungen derselben Brandgase an der rechten Stirn, jeweils das Temperaturprofil auf halber Höhe nach 30 min. Feste Temperatur (rot): das Ende sitzt fest auf 500 °C. Feste Leistung 1500 W (orange): das Ende regelt sich selbst ein — hier schon auf 542 °C, über die 500 hinaus. Brandgas per Robin-Rand (blau): der Übergang mit α_c = 25 W/(m²·K) erwärmt die kleine Stirnfläche in 30 min erst auf rund 59 °C — aber er kann die 500 °C der Gase grundsätzlich nie überschreiten. Drei Ränder, drei Geschichten (Kapitel 8).

Ergebnis. Genau wie vorhergesagt: Die Temperatur-Flamme klebt bei $500\,°\mathrm{C}$; die Leistungs-Flamme schießt darüber hinaus (bei 1500 W auf $542\,°\mathrm{C}$, bei 3000 W auf über $1000\,°\mathrm{C}$ — sie kennt keinen Deckel); der Robin-Rand erwärmt die kleine Stirnfläche in 30 Minuten erst auf $59\,°\mathrm{C}$.

i Der Robin-Rand, endlich ausgerechnet — und warum er sättigt

In Kapitel 8 blieb der **Übergangsrand** ein Versprechen: Der Strom $Q = \alpha_c\, A\,(T_{\text{Gas}} - T)$ hängt von der gesuchten Temperatur T selbst ab, deshalb — so hieß es — „rechnen wir das in 3 aus". Hier ist es. Der Wärmeübergangskoeffizient α_c beschreibt, wie gut die Brandgase ihre Wärme an den Stahl abgeben; die Norm für den Brandfall setzt $\alpha_c = 25\ \mathrm{W/(m^2K)}$ (CEN (Europäisches Komitee für Normung) 2005). Im Code wandert sein temperaturabhängiger Teil auf die **Diagonale** der Systemmatrix, sein bekannter Teil auf die **rechte Seite** (Modul 2 der Werksbesichtigung) — genau die „Zwitter"-Aufteilung aus Kapitel 8.

Der entscheidende Unterschied zur Leistung: **Die Brandgase können den Träger höchstens auf ihre eigene Temperatur bringen, nie darüber.** Lässt man dieselbe rechte Stirn stundenlang weiterheizen, kriecht das Ende beim Robin-Rand von 59 über 94 und 147 langsam gegen die $500\,°\mathrm{C}$ der Gase — und bleibt darunter, weil der antreibende Unterschied $T_{\text{Gas}} - T$ mit jedem Grad kleiner wird. Die feste **Leistung** dagegen kennt diese Bremse nicht: 3000 W treiben das Ende in denselben Stunden auf 1000, 2000, $4000\,°\mathrm{C}$ — längst geschmolzen. Ein echtes Feuer ist ein Gas fester Temperatur, kein Brenner fester Leistung; darum ist der Robin-Rand das ehrlichere Brandmodell. Für sehr großes α_c presst er das Ende sofort auf Gastemperatur (das ist Dirichlet), für $\alpha_c = 0$ fließt gar nichts (das ist der isolierte Rand) — alle wirklichen Übergänge liegen dazwischen.

Bonus: warum die Beplankung den Träger rettet

Jetzt der Grund, warum man Stahlträger im Bau **beplankt**. Wir vergleichen zwei Träger in der vollen Brand-Szene (Brandgase $500\,°\mathrm{C}$, eine Stunde): Beim ersten schlagen die Gase frei an die Unterseite (Robin), beim zweiten steckt die Unterseite unter einer dämmenden Brandschutzplatte (für die Wärme ein **isolierter** Rand). Das Ergebnis ist drastisch: Der freie Träger erwärmt sich in der Mitte auf rund $110\,°\mathrm{C}$ und wölbt sich (winzige, aber echte $0{,}42$ mm); der beplankte bleibt bei $23\,°\mathrm{C}$ und krümmt sich praktisch gar nicht ($0{,}03$ mm). Dieselbe Physik, ein Dämmbrett dazwischen — und aus einem sich verziehenden Bauteil wird ein ruhendes. Genau darauf beruhen die Feuerwiderstandsklassen R30/R60/R90.

Schluss. Welchen Rand man wählt, ist keine Rechenfrage, sondern eine **Modellierungsfrage** (Kapitel 12): Sind die Brandgase ein Thermostat (Temperatur), ein Brenner fester Leistung (Neumann) oder ein Gas, das über ei-

nen Übergang heizt (Robin)? Dieselbe Geometrie liefert je nach Randwahl verschiedene Wahrheiten — und nur eine passt zur Wirklichkeit.

15.3.4. Reihe D — Das Material: Stahl gegen Aluminium

Frage. Unser Träger ist aus Stahl. Wir stellen ihm einen **Aluminium**-Herausforderer aus derselben Geometrie gegenüber und fahren ein Renn-Duell mit drei Fragen: Wer wird **schneller heiß**? Wer **krümmt sich stärker**? Wer **trägt mehr Spannung**?

Vorhersage. Aluminium leitet Wärme viermal besser und speichert weniger — es sollte schneller heiß werden. Es dehnt sich fast doppelt so stark pro Grad ($\alpha_{\text{Alu}} = 23$ gegen $\alpha_{\text{Stahl}} = 12 \cdot 10^{-6}$) — es sollte sich stärker krümmen. Aber Stahl ist dreimal steifer ($E = 210$ gegen 70 GPa), und für die Einspann-Spannung zählt das Produkt $E\,\alpha$ (Kapitel 13) — da liegt Stahl vorn.

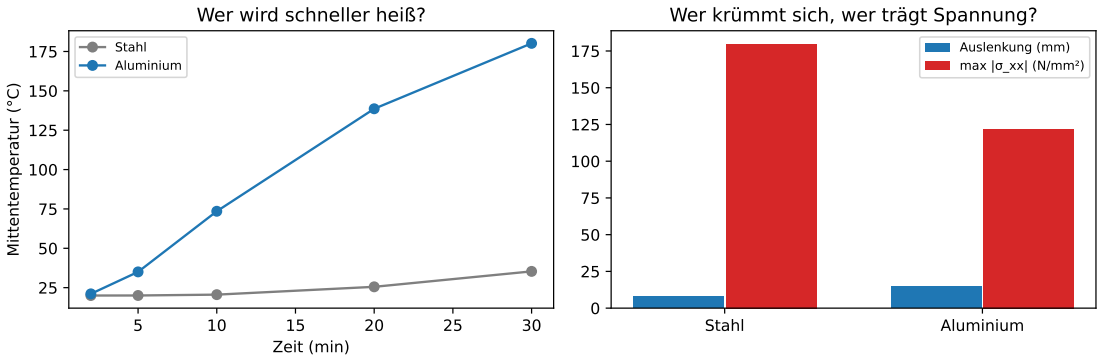

Abbildung 15.7.: Das Renn-Duell Stahl gegen Alu. Links: Aufheizen der Mitte, wenn die rechte Stirn auf 500 °C gehalten wird und die Wärme nach links kriecht — Aluminium (blau) zieht davon, weil es Wärme viel besser leitet und weniger speichert (nach 10 min schon 74 °C, Stahl kaum über 20). Rechts: der voll entwickelte Brand (Gefälle 200 K über die Höhe, eingeschwungen, Netz 16×4) — Aluminium krümmt sich fast doppelt so stark (14,8 gegen 7,7 mm), aber Stahl trägt mehr Spannung (180 gegen 122 N/mm²), weil für die Spannung das Produkt E·α zählt (Kapitel 13).

Ergebnis. Alle drei Vorhersagen treffen zu. Aluminium wird dramatisch schneller heiß (nach 10 Minuten 74 gegen kaum $21\,^{\circ}\text{C}$ in der Mitte). Aluminium krümmt sich stärker ($14{,}8$ gegen $7{,}7$ mm). Aber Stahl trägt mehr Spannung (180 gegen $122\ \text{N}/\text{mm}^2$).

Schluss. Es gibt keinen „besseren" Werkstoff — es kommt auf die Frage an. Wer schnelle, gleichmäßige Wärme will (Kühlkörper, Bratpfanne), nimmt Aluminium. Wer Kräfte tragen und Form halten will (Träger, Welle), nimmt Stahl. Und wer die Wärmespannung fürchtet, erinnert sich: Nicht die Dehnung allein spannt, sondern $E\,\alpha$ — der steife Stahl bezahlt seine Steifigkeit mit höherer Spannung (das ist Übung 13.2, jetzt am 2D-Träger bestätigt).

15.3.5. Reihe E — Die Elemente: Dreiecke gegen Rechtecke

Frage. Wir haben zwei Elementfamilien gebaut: P1-**Dreiecke** (1) und Q1-**Rechtecke** (Kapitel 11). Auf demselben Gitter, demselben Problem — liefern sie dasselbe?

Vorhersage. Beide sind lineare Elemente zweiter Konvergenzordnung (Kapitel 12), beide erfüllen die Energiebilanz exakt. Auf einem glatten Problem sollten sie **fast dasselbe** liefern; kleine Unterschiede sind möglich, weil das Rechteck auch die diagonalen Nachbarn koppelt (der 9-Punkt-Stern aus Kapitel 11).

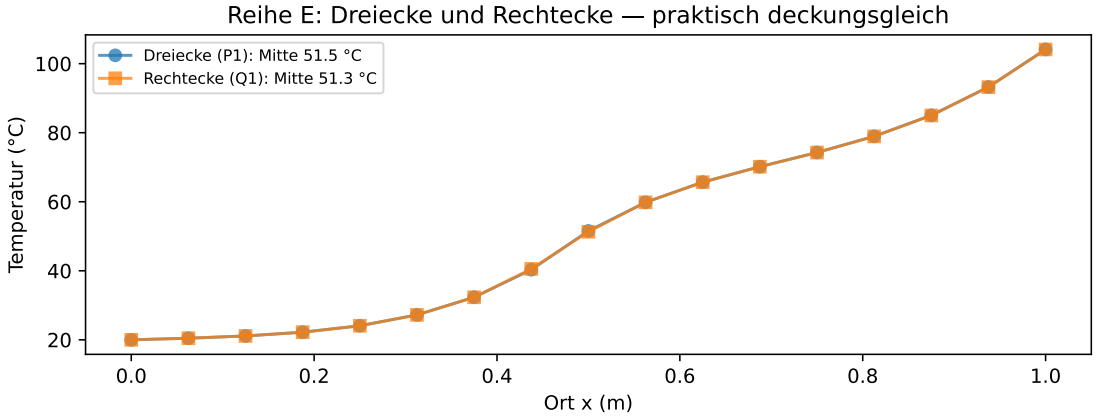

Abbildung 15.8.: Reihe E: Dreiecke gegen Rechtecke auf identischem Gitter (16×4), Brand-Szene. Die Temperaturprofile auf halber Höhe liegen praktisch übereinander — beide Elementfamilien treffen fast dieselbe Mittentemperatur (51,5 gegen 51,3 °C) und schließen die Energiebilanz auf Maschinengenauigkeit. Für dieses glatte Problem ist die Wahl des Elementtyps zweitrangig; erst bei krummen Rändern oder groben Netzen zeigen sich Unterschiede.

Ergebnis. Die beiden Kurven liegen übereinander — beide treffen rund $51{,}5\,°\mathrm{C}$ in der Mitte (Dreiecke $51{,}5$, Rechtecke $51{,}3$), beide schließen die

Energiebilanz perfekt.

Schluss. Für dieses glatte, achsparallele Problem ist der Elementtyp zweitrangig — ein beruhigendes Ergebnis, denn es heißt, dass die Physik und nicht die Elementwahl das Bild bestimmt. Unterschiede treten erst auf, wo es darauf ankommt: bei krummen Rändern (dort können Rechtecke gar nicht, und man braucht die **isoparametrischen** Elemente aus der Grenzen-Tafel) oder bei sehr groben Netzen. Die Wahl der Elementfamilie ist eine Frage der Geometrie, nicht der Genauigkeit.

15.4. Das Finale in Bewegung

Die fünf Reihen haben je einen Regler isoliert. Jetzt lassen wir alles zusammenlaufen — die Auflösung des Kapitel-1-Trailers, Glied für Glied verstanden: der voll entwickelte Brand hält die Unterseite heiß, die Wärme kriecht von unten herein, das Temperaturgefälle über die Höhe wächst, und der Träger wölbt sich synchron nach oben.

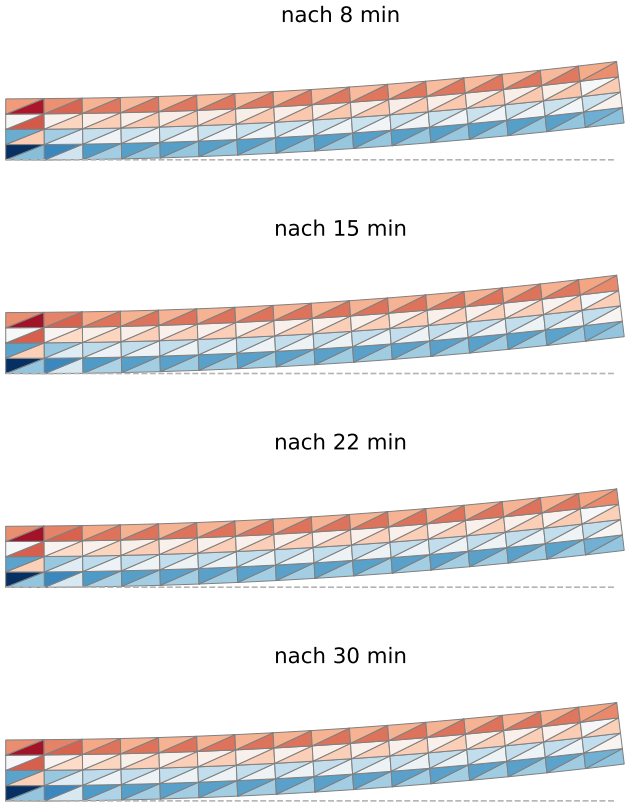

Abbildung 15.9.: Das große Finale als Bildreihe (HTML: Animation). Vier Zeitpunkte des voll entwickelten Brands: mit wachsender Zeit kriecht die Wärme von unten herein, das Höhengefälle wächst, und der Träger wölbt sich immer stärker nach oben (× 8 überhöht, Farbe σ_xx). Die Auflösung des Kapitel-1-Trailers.

Was man sieht: Genau der Trailer aus Kapitel 1 — nur ist jetzt jede Farbe, jede Krümmung, jeder Millimeter das Ergebnis einer Rechnung, die du Zeile für Zeile kennst. Das ist das Versprechen des ersten Kapitels, eingelöst.

15.5. Die interaktive Einheit: dein Finale

Jetzt gehört das Programm dir. Die folgende Einheit ist das ganze Finale im Browser — alle Regler offen. Der Code darin ist eine **kompakte Inline-Fassung**; das vollständige, ausführlich kommentierte Programm ist `programme/kap15/fem_balken.py`, an dem sich dieses Kapitel entlanghangelt.

Bevor du startest, eine **Meister-Vorhersage** über zwei Regler auf einmal:

Vorhersage-Punkt (der Meister-Regler)

Bevor du das Programm startest: Der Standardlauf ist **Stahl** im voll entwickelten Brand — die Flammen halten die Unterseite auf 220 °C, die Decke die Oberseite auf 20 °C (Gefälle 200 K über die Höhe). Jetzt drehst du **zwei** Regler zugleich: Du schaltest auf **Aluminium** *und* drosselst den Brand auf die Hälfte (Unterseite nur noch 120 °C, Gefälle 100 K). Wählst du den Träger dann **stärker, schwächer** oder **gleich** wie den

```
#| autorun: true
# ======================================================================
#  DAS FINALE -- kompakte Inline-Fassung (vollstaendig: programme/kap15/fem_balken.py)
#  Drehe an den Reglern und druecke Ausfuehren.
# ======================================================================
MATERIAL = "stahl"        # "stahl" (der Traeger) oder "alu" (der Herausforderer)
NX = 12                   # Netzspalten (im Browser klein halten: <= 12)
NY = 3                    # Netzreihen (<= 3)
ZEIT_MIN = 30.0           # simulierte Dauer (Minuten)
ZEITSCHRITTE = 8          # Zahl der impliziten Schritte (Δt = Zeit/Schritte)
UEBERHOEHUNG = 8.0        # wie stark die Verformung gezeichnet wird
# Raender je Kante: ("temperatur", Grad) | ("leistung", Watt)
#                 | ("isoliert",) | ("robin", alpha, T_Gas)
RAND_LINKS = ("isoliert",)
RAND_RECHTS = ("isoliert",)
RAND_OBEN = ("temperatur", 20.0)      # die kuehle Betondecke
RAND_UNTEN = ("temperatur", 220.0)    # die beflammte Unterseite (Hoehengefaelle)

# --- Material (Kapitel 6/9/13/14) ---------------------------------------
WERTE = {"stahl": {"lam": 50.0, "rho": 7850.0, "c": 460.0,
                   "E": 210e9, "nu": 0.30, "alpha": 12e-6},
         "alu":   {"lam": 200.0, "rho": 2700.0, "c": 900.0,
                   "E": 70e9, "nu": 0.33, "alpha": 23e-6}}
M = WERTE[MATERIAL]
L, H, TIEFE, REF = 1.0, 0.1, 0.1, 20.0

# --- Netz: Dreiecke auf dem Gitter (Kapitel 10) -----------------------
knoten = []
for zeile in range(NY + 1):
    for spalte in range(NX + 1):
```

```
        knoten.append((spalte * L / NX, zeile * H / NY))
dreiecke = []
for zeile in range(NY):
    for spalte in range(NX):
        ul = zeile * (NX + 1) + spalte
        dreiecke.append((ul, ul + 1, ul + NX + 2))
        dreiecke.append((ul, ul + NX + 2, ul + NX + 1))

def flaeche(a, b, c):
    kreuz = (b[0] - a[0]) * (c[1] - a[1]) - (c[0] - a[0]) * (b[1] - a[1])
    if kreuz < 0.0:
        kreuz = -kreuz
    return kreuz / 2.0

# --- Leitungsmatrix K und Waermeeimer C (Kapitel 9/10) ----------------
n = len(knoten)
K = []
for i in range(n):
    K.append([0.0] * n)
kap = [0.0] * n
for d in dreiecke:
    (xa, ya) = knoten[d[0]]; (xb, yb) = knoten[d[1]]; (xc, yc) = knoten[d[2]]
    zwei = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    bx = [(yb - yc) / zwei, (yc - ya) / zwei, (ya - yb) / zwei]
    cy = [(xc - xb) / zwei, (xa - xc) / zwei, (xb - xa) / zwei]
    fl = flaeche(knoten[d[0]], knoten[d[1]], knoten[d[2]])
    for a in range(3):
        kap[d[a]] = kap[d[a]] + M["rho"] * M["c"] * fl * TIEFE / 3.0
        for b in range(3):
            K[d[a]][d[b]] = K[d[a]][d[b]] + M["lam"] * TIEFE * fl \
                * (bx[a] * bx[b] + cy[a] * cy[b])

# --- Kantenknoten und ihre Randflaechen (Kapitel 8) -------------------
kanten = {"links": [], "rechts": [], "oben": [], "unten": []}
for i in range(n):
    x, y = knoten[i]
    if abs(x) < 1e-9:
        kanten["links"].append(i)
    if abs(x - L) < 1e-9:
        kanten["rechts"].append(i)
    if abs(y - H) < 1e-9:
        kanten["oben"].append(i)
    if abs(y) < 1e-9:
```

```
        kanten["unten"].append(i)

def randflaeche(kante, knoten_nr):
    if kante in ("oben", "unten"):
        stueck = (L / NX) * TIEFE
        rand = kanten[kante]
    else:
        stueck = (H / NY) * TIEFE
        rand = kanten[kante]
    if knoten_nr == rand[0] or knoten_nr == rand[-1]:
        return stueck / 2.0
    return stueck

RAENDER = {"links": RAND_LINKS, "rechts": RAND_RECHTS,
           "oben": RAND_OBEN, "unten": RAND_UNTEN}

def gauss_seidel(matrix, seite, start):
    x = list(start)
    for runde in range(4000):
        groesst = 0.0
        for i in range(len(matrix)):
            summe = seite[i]
            for j in range(len(matrix)):
                if j != i:
                    summe = summe - matrix[i][j] * x[j]
            neu = summe / matrix[i][i]
            aend = neu - x[i]
            if aend < 0.0:
                aend = -aend
            if aend > groesst:
                groesst = aend
            x[i] = neu
        if groesst < 1e-6:
            break
    return x

def waermeschritt(temp, dt):
    A = []
    for i in range(n):
        A.append(list(K[i]))
        A[i][i] = A[i][i] + kap[i] / dt
    seite = []
    for i in range(n):
```

```
        seite.append(kap[i] / dt * temp[i])
    fest = [None] * n
    for kante in ("unten", "oben", "links", "rechts"):
        typ = RAENDER[kante][0]
        for knoten_nr in kanten[kante]:
            rf = randflaeche(kante, knoten_nr)
            if typ == "leistung":
                gesamt = 0.0
                for kn2 in kanten[kante]:
                    gesamt = gesamt + randflaeche(kante, kn2)
                seite[knoten_nr] = seite[knoten_nr] + RAENDER[kante][1] * rf / gesamt
            elif typ == "robin":
                A[knoten_nr][knoten_nr] = A[knoten_nr][knoten_nr] + RAENDER[kante][1] * rf
                seite[knoten_nr] = seite[knoten_nr] + RAENDER[kante][1] * rf * RAENDER[kante][2]
            elif typ == "temperatur":
                fest[knoten_nr] = RAENDER[kante][1]
    frei = []
    for i in range(n):
        if fest[i] is None:
            frei.append(i)
    klein = []
    kseite = []
    for i in frei:
        zeile = []
        for j in frei:
            zeile.append(A[i][j])
        klein.append(zeile)
        beitrag = 0.0
        for j in range(n):
            if fest[j] is not None:
                beitrag = beitrag + A[i][j] * fest[j]
        kseite.append(seite[i] - beitrag)
    start = []
    for i in frei:
        start.append(temp[i])
    loesung = gauss_seidel(klein, kseite, start)
    neu = list(fest)
    for k in range(len(frei)):
        neu[frei[k]] = loesung[k]
    for i in range(n):
        if neu[i] is None:
            neu[i] = temp[i]
    return neu
```

```
# --- Zeitschleife (Kapitel 9) + Energiebilanz-Wachhund (Kapitel 12) ---
dt = ZEIT_MIN * 60.0 / ZEITSCHRITTE
temp = [REF] * n
start_temp = list(temp)
herein = 0.0
hinaus = 0.0
for schritt in range(ZEITSCHRITTE):
    alt = temp
    temp = waermeschritt(alt, dt)
    for i in range(n):
        fluss = 0.0
        for j in range(n):
            fluss = fluss + K[i][j] * temp[j]
        rest = kap[i] / dt * (temp[i] - alt[i]) + fluss
        if rest > 0.0:
            herein = herein + rest * dt
        else:
            hinaus = hinaus - rest * dt
gespeichert = 0.0
for i in range(n):
    gespeichert = gespeichert + kap[i] * (temp[i] - start_temp[i])
bilanzfehler = 0.0
if abs(herein) > 1e-9:
    bilanzfehler = abs(herein - (gespeichert + hinaus)) / abs(herein) * 100.0

# --- Kopplung an die Mechanik (Kapitel 14) ----------------------------
def loese_gauss(matrix, seite):
    m = len(matrix)
    werk = []
    for zeile in matrix:
        werk.append(list(zeile))
    s = list(seite)
    for spalte in range(m):
        piv = werk[spalte][spalte]
        for zeile in range(spalte + 1, m):
            fk = werk[zeile][spalte] / piv
            for k in range(spalte, m):
                werk[zeile][k] = werk[zeile][k] - fk * werk[spalte][k]
            s[zeile] = s[zeile] - fk * s[spalte]
    x = [0.0] * m
    for zeile in range(m - 1, -1, -1):
        summe = s[zeile]
        for k in range(zeile + 1, m):
```

```
            summe = summe - werk[zeile][k] * x[k]
        x[zeile] = summe / werk[zeile][zeile]
    return x

delta_t = []
for d in dreiecke:
    mittel = (temp[d[0]] + temp[d[1]] + temp[d[2]]) / 3.0
    delta_t.append(mittel - REF)
faktor = M["E"] / (1.0 - M["nu"] * M["nu"])
Cmat = [[faktor, faktor * M["nu"], 0.0],
        [faktor * M["nu"], faktor, 0.0],
        [0.0, 0.0, faktor * (1.0 - M["nu"]) / 2.0]]
Kel = []
for i in range(2 * n):
    Kel.append([0.0] * (2 * n))
Fel = [0.0] * (2 * n)
spannungen = []
b_tabs = []
for nummer in range(len(dreiecke)):
    d = dreiecke[nummer]
    (xa, ya) = knoten[d[0]]; (xb, yb) = knoten[d[1]]; (xc, yc) = knoten[d[2]]
    zwei = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    bx = [(yb - yc) / zwei, (yc - ya) / zwei, (ya - yb) / zwei]
    cy = [(xc - xb) / zwei, (xa - xc) / zwei, (xb - xa) / zwei]
    fl = flaeche(knoten[d[0]], knoten[d[1]], knoten[d[2]])
    B = [[0.0] * 6, [0.0] * 6, [0.0] * 6]
    for e in range(3):
        B[0][2 * e] = bx[e]; B[1][2 * e + 1] = cy[e]
        B[2][2 * e] = cy[e]; B[2][2 * e + 1] = bx[e]
    b_tabs.append(B)
    CB = [[0.0] * 6, [0.0] * 6, [0.0] * 6]
    for r in range(3):
        for cc in range(6):
            summe = 0.0
            for k in range(3):
                summe = summe + Cmat[r][k] * B[k][cc]
            CB[r][cc] = summe
    dofs = []
    for e in range(3):
        dofs.append(2 * d[e]); dofs.append(2 * d[e] + 1)
    vor = M["alpha"] * delta_t[nummer]
    c_eps = [Cmat[0][0] * vor + Cmat[0][1] * vor,
             Cmat[1][0] * vor + Cmat[1][1] * vor, 0.0]
```

```
    for a in range(6):
        kraft = 0.0
        for k in range(3):
            kraft = kraft + B[k][a] * c_eps[k]
        Fel[dofs[a]] = Fel[dofs[a]] + kraft * TIEFE * fl
        for b in range(6):
            wert = 0.0
            for k in range(3):
                wert = wert + B[k][a] * CB[k][b]
            Kel[dofs[a]][dofs[b]] = Kel[dofs[a]][dofs[b]] + wert * TIEFE * fl

lager = {}
for i in range(n):
    if abs(knoten[i][0]) < 1e-9:
        lager[2 * i] = 0.0
        lager[2 * i + 1] = 0.0
frei = []
for dof in range(2 * n):
    if dof not in lager:
        frei.append(dof)
red = []
rs = []
for i in frei:
    zeile = []
    for j in frei:
        zeile.append(Kel[i][j])
    red.append(zeile)
    rs.append(Fel[i])
loesung = loese_gauss(red, rs)
u = [0.0] * (2 * n)
for k in range(len(frei)):
    u[frei[k]] = loesung[k]

sig_xx = []
for nummer in range(len(dreiecke)):
    d = dreiecke[nummer]; B = b_tabs[nummer]
    eck = []
    for e in range(3):
        eck.append(u[2 * d[e]]); eck.append(u[2 * d[e] + 1])
    deh = [0.0, 0.0, 0.0]
    for r in range(3):
        summe = 0.0
        for k in range(6):
```

```
            summe = summe + B[r][k] * eck[k]
        deh[r] = summe
    vor = M["alpha"] * delta_t[nummer]
    verh = [deh[0] - vor, deh[1] - vor, deh[2]]
    sxx = Cmat[0][0] * verh[0] + Cmat[0][1] * verh[1]
    sig_xx.append(sxx / 1e6)

# --- Kennzahlen ausgeben ------------------------------------------
mitte = temp[n // 2]
w_ende = 0.0
for i in range(n):
    if abs(knoten[i][0] - L) < 1e-9:
        absenk = -u[2 * i + 1] * 1000.0
        if absenk > w_ende:
            w_ende = absenk
smax = 0.0
for s in sig_xx:
    if abs(s) > smax:
        smax = abs(s)
print("Material %s, Netz %dx%d, %.0f min in %d Schritten (dt=%.0f s)"
      % (MATERIAL, NX, NY, ZEIT_MIN, ZEITSCHRITTE, dt))
print("Mittentemperatur:        %6.1f Grad" % mitte)
print("Durchbiegung freies Ende:%6.2f mm" % w_ende)
print("groesste |sigma_xx|:     %6.1f N/mm2" % smax)
print("Energiebilanz-Wachhund:  Fehler %.3f %% (Toleranz 1 %%) -- %s"
      % (bilanzfehler, "OK" if bilanzfehler < 1.0 else "ALARM"))
print("Checkliste (Kap. 12): Nulltest? Konvergenz? Raender? Bilanz? Handprobe?")

# --- Abbildung ---
import matplotlib.pyplot as plt
from matplotlib.collections import PolyCollection
fig, (axl, axr) = plt.subplots(1, 2, figsize=(10.5, 3.0),
                               gridspec_kw={"width_ratios": [1.4, 1.6]})
tp = []
tw = []
for d in dreiecke:
    tp.append([(knoten[d[k]][0], knoten[d[k]][1]) for k in range(3)])
    tw.append((temp[d[0]] + temp[d[1]] + temp[d[2]]) / 3.0)
c1 = PolyCollection(tp, cmap="inferno", edgecolors="none")
c1.set_array(tw)
axl.add_collection(c1)
axl.autoscale_view()
axl.set_aspect("equal")
```

```
axl.set_title("Temperaturfeld")
axl.set_xlabel("x (m)")
verf = []
for d in dreiecke:
    verf.append([(knoten[d[k]][0] + UEBERHOEHUNG * u[2 * d[k]],
                  knoten[d[k]][1] + UEBERHOEHUNG * u[2 * d[k] + 1]) for k in range(3)])
grenze = 1.0
for s in sig_xx:
    if abs(s) > grenze:
        grenze = abs(s)
c2 = PolyCollection(verf, cmap="RdBu_r", edgecolors="0.6", linewidths=0.2)
c2.set_array(sig_xx)
c2.set_clim(-grenze, grenze)
axr.add_collection(c2)
axr.plot([0, L], [0, 0], "--", color="0.7", lw=0.7)
axr.autoscale_view()
axr.set_aspect("equal")
axr.set_title("Verformung (ueberhoeht), Farbe sigma_xx")
axr.set_xlabel("x (m)")
plt.tight_layout()
plt.show()
```

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap15"))
import matplotlib.pyplot as plt
from matplotlib.collections import PolyCollection
import fem_balken as fb

gefaelle = {"links": ("isoliert",), "rechts": ("isoliert",),
            "oben": ("temperatur", 20.0), "unten": ("temperatur", 220.0)}
erg = fb.simuliere(nx=12, ny=3, material_name="stahl", raender=gefaelle,
                   zeit_minuten=30.0, zeitschritte=8, kopplungsschritte=[8])
knoten = erg["knoten"]; dreiecke = erg["dreiecke"]
temp = erg["verlauf"][-1]; u = erg["momentaufnahmen"][-1]["verschiebungen"]
sig = erg["momentaufnahmen"][-1]["spannungen"]
fig, (axl, axr) = plt.subplots(1, 2, figsize=(10.5, 3.0),
                               gridspec_kw={"width_ratios": [1.4, 1.6]})
tp = [[(knoten[d[k]][0], knoten[d[k]][1]) for k in range(3)] for d in dreiecke]
tw = [(temp[d[0]] + temp[d[1]] + temp[d[2]]) / 3.0 for d in dreiecke]
c1 = PolyCollection(tp, array=tw, cmap="inferno", edgecolors="none")
axl.add_collection(c1); axl.autoscale_view(); axl.set_aspect("equal")
axl.set_title("Temperaturfeld"); axl.set_xlabel("x (m)")
ueber = 8.0
```

```
verf = [[(knoten[d[k]][0] + ueber * u[2 * d[k]],
          knoten[d[k]][1] + ueber * u[2 * d[k] + 1]) for k in range(3)] for d in dreiecke]
sp = [s[0] / 1e6 for s in sig]; g = max(1.0, max(abs(v) for v in sp))
c2 = PolyCollection(verf, array=sp, cmap="RdBu_r", edgecolors="0.6", linewidths=0.2)
c2.set_clim(-g, g); axr.add_collection(c2)
axr.plot([0, 1], [0, 0], "--", color="0.7", lw=0.7); axr.autoscale_view()
axr.set_aspect("equal"); axr.set_title("Verformung (× 8), Farbe σ_xx"); axr.set_xlabel("x (m)")
plt.tight_layout(); plt.show()
```

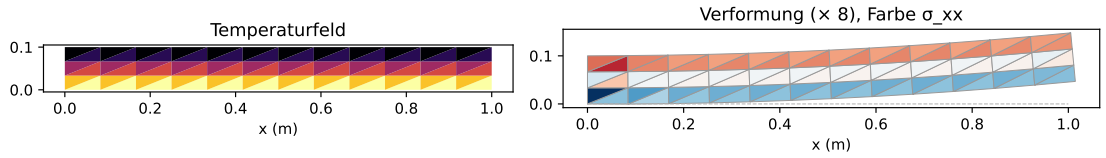

Abbildung 15.10.: Vorgerenderte Fassung der interaktiven Einheit (Stahl, Netz 12×3, voll entwickelter Brand: unten 220 °C, oben 20 °C, links eingespannt, 30 min). Links das Temperaturfeld, rechts der nach oben gewölbte Träger (× 8 überhöht), eingefärbt nach σ_xx. Im Browser wird dieses Bild durch dein Ergebnis ersetzt — verstelle Material, Netz, Zeit und die vier Ränder und prüfe jede Zahl gegen die Checkliste.

Auflösung der Meister-Vorhersage. Überraschung: Aluminium mit halber Flamme wölbt sich **fast genau gleich** wie Stahl mit voller — $5{,}8$ gegen $6{,}0$ mm (auf diesem groben 12×3-Netz). Der Grund steht in der Krümmungsformel $\kappa = \alpha\,\Delta T / H$: Aluminiums doppeltes α mal dem halben ΔT landet wieder bei Stahls Krümmung. Aber schau auf die Spannung — der Stahl trägt $184\ \mathrm{N/mm^2}$ gegen Aluminiums 62: **rund dreimal so viel**. Zwei Regler, die sich in der Krümmung aufheben, aber in der Spannung weit auseinanderlaufen. Genau solche Wechselwirkungen macht ein Experiment sichtbar, das eine bloße Vorhersage nie fände.

15.6. Die Landkarte, alle Häkchen gesetzt

Wir sind am Ziel. Abbildung 15.11 ist die Landkarte aus Kapitel 1, zum letzten Mal — jetzt mit *allen* Häkchen. Jeder der fünf Bausteine ist gefüllt, und die drei Erweiterungen (Verformung, Prüfen, Finale) stehen daneben, ebenfalls abgehakt.

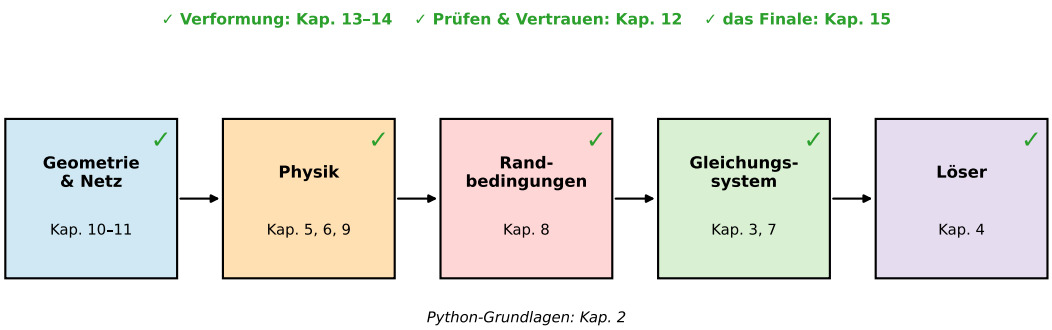

Abbildung 15.11.: Die Landkarte des Buches, vollendet. Die fünf Bausteine jeder FEM-Rechnung — Geometrie und Netz, Physik, Randbedingungen, Gleichungssystem, Löser — alle gefüllt und abgehakt, mit den Kapiteln, die sie gebaut haben. Darüber die drei Erweiterungen dieses Buches: Verformung (Kapitel 13/14), Prüfen und Vertrauen (Kapitel 12), das Finale (Kapitel 15). Das ist die ganze Reise auf einem Blatt — dein Ausweis, dass du jedes FEM-Programm von innen verstehst.

Was man sieht: dasselbe Bild wie am Anfang, aber jeder Kasten trägt jetzt sein grünes Häkchen. Das ist kein Schmuck — es ist die Buchhaltung deines Wissens. Du hast jeden dieser Kästen selbst gebaut.

15.7. Grenzen und Ausblick: die Landkarte des Weiterlernens

Ehrlichkeit gehört zum Handwerk (Kapitel 12). Dieses Programm hat **bewusste Grenzen**, und jede davon ist die Tür zu einem größeren Werkzeug. Eine dieser Grenzen wiegt im Brandfall schwerer als alle anderen und gehört deshalb an die Spitze der Tafel.

Die wichtigste Grenze: Stahl wird im Feuer weich

Unser ganzes Buch rechnet mit **temperaturunabhängigen Kennwerten** — mit demselben E, demselben λ, ob der Träger 20 oder $600\,°\mathrm{C}$ hat. Für die Brandphysik ist das die folgenreichste Vereinfachung, denn in Wirklichkeit **verliert Baustahl mit der Hitze seine Festigkeit**: Bei etwa $600\,°\mathrm{C}$ ist von der Streckgrenze nur noch rund die Hälfte übrig,

bei $700\,°\mathrm{C}$ etwa ein Viertel (CEN (Europäisches Komitee für Normung) 2005). Genau darum versagen Träger im Brand nicht durch Schmelzen (das käme erst bei $\sim 1500\,°\mathrm{C}$), sondern lange vorher durch **Erweichen** — sie tragen ihre Last nicht mehr und knicken weg. Der Eurocode EN 1993-1-2 gießt diese Abminderung in Tabellen und begründet damit die Feuerwiderstandsklassen R30/R60/R90 und die Brandschutzbeplankung aus Reihe C. Ein Profi-Programm rechnet mit genau diesen temperaturabhängigen Kurven; unseres nennt die Grenze ehrlich und überlässt ihr Überschreiten dem nächsten Werkzeug.

Abbildung 15.12 sammelt die übrigen Grenzen als Landkarte des Weiterlernens.

Abbildung 15.12.: Die Grenzen dieses Programms — und wohinter jede führt. Sechs Türen: von 2D zu 3D, von geraden zu krummen Rändern (isoparametrische Elemente), von linearem zu nichtlinearem Material, von der Einweg- zur echten Rückkopplung, von festen zu adaptiven Netzen, von Gauß-Seidel zu schnellen Lösern. Jede Tür ist in diesem Buch als Ausblick benannt worden; hinter allen zusammen liegen die professionellen FEM-Programme.

Was man sieht: sechs Türen, und hinter jeder wartet ein Stück Profi-FEM. Der Reihe nach:

- **Von 2D zu 3D.** Unser Träger lebt in der Ebene. Ein echtes Bauteil ist räumlich — dann treten an die Stelle der Dreiecke **Tetraeder**, jeder Knoten trägt *drei* Verschiebungen, und die Matrizen wachsen. Die Idee bleibt Wort für Wort dieselbe; es wird nur größer.

- **Von geraden zu krummen Rändern.** Unsere Rechtecke müssen achsparallel sein (Kapitel 11). Für runde Löcher und geschwungene Formen braucht man **isoparametrische** Elemente mit Jacobi-Matrix und Gauß-Quadratur — der Ausblick, den Kapitel 11 bewusst offenließ.
- **Von linearem zu nichtlinearem Material.** Wir rechnen mit festen Kennwerten (Kapitel 13). In Wahrheit ändern sich E und λ mit der Temperatur — die Erweichung aus dem Kasten oben —, und jenseits der Streckgrenze **fließt** das Metall (EN 1993-1-2, (CEN (Europäisches Komitee für Normung) 2005)). Dann wird aus einem Gleichungssystem eine Folge von Gleichungssystemen.
- **Von der Einweg- zur echten Rückkopplung.** Bei uns treibt die Wärme die Verformung, nicht umgekehrt (Kapitel 14). Echte Kopplung lässt auch die Verformung auf die Wärme zurückwirken — man rechnet dann **iterativ hin und her**, bis sich beide einig sind.
- **Von festen zu adaptiven Netzen.** Wir setzen das Netz von Hand (Kapitel 12). Profi-Programme **verfeinern selbst** dort, wo der Fehler groß ist — etwa an der Einspann-Ecke oder am Loch, wo unsere Spannung in Reihe A nicht konvergieren wollte.
- **Von Gauß-Seidel zu schnellen Lösern.** Unser Entspannungsverfahren

  (2) ist ehrlich, aber langsam. Für Millionen Unbekannte nimmt man **Mehrgitter-** und vorkonditionierte Verfahren, die das Einpendeln um Größenordnungen beschleunigen.

Wer weitermachen will: OpenGeoSys

Wer diese Türen durchschreiten möchte, findet in **OpenGeoSys** ein offenes, professionelles FEM-Programm für gekoppelte Prozesse in porösen Medien — Wärme, Strömung, Mechanik, Chemie, auch in Kombination (Kolditz u. a. 2012). Es hat exakt die Module dieses Buches, nur in groß: 3D-Netze, krumme Ränder, nichtlineare Materialien, echte Zwei-Wege-Kopplung, schnelle Löser. Sein Handbuch und seine **Benchmark**-Galerie (jeder Prüffall mit bekannter Lösung, ganz im Sinn von Kapitel 12) lesen sich für dich jetzt nicht mehr als Fremdsprache, sondern als vertraute Landkarte. Dort geht die Reise weiter.

Merkkasten

- Ein vollständiges FEM-Programm ist die **Zeitschleife der Wärme** (Kapitel 9), an deren Ausgabeschritten die **Kopplungskette der Mechanik** (Kapitel 14) hängt — mehr nicht.
- Profi-Programme haben **dieselben Module**, nur mehr Ausstat-

tung. Wer das Finale versteht, liest ihre Handbücher als Landkarte.

- Ein FEM-Programm ist ein **Instrument für Experimente**: Frage → Vorhersage → Regler → Ergebnis → Schluss. Ohne Vorhersage ist es Herumklicken.
- **Verschiebungen konvergieren schneller als Spannungen**; Spannungsspitzen an Ecken (Kerbwirkung) konvergieren gar nicht.
- Der **implizite** Zeitschritt explodiert nie — grobe Δt verschmieren nur.
- Jedes Ergebnis erst durch die **Checkliste** aus Kapitel 12, dann glauben. Der Energiebilanz-Wachhund läuft immer mit.

Schlusswort

Am Anfang stand eine Blackbox mit drei Reglern und ein Versprechen. Jetzt steht dasselbe Programm offen vor dir, und du weißt, was jede Zeile tut.

Du kannst ein reales Ding — einen Träger, eine Pfanne, eine Brücke — in ein **Netz** aus Elementen zerlegen. Du kannst für jedes Element die **Physik** in eine kleine Matrix gießen und die vielen kleinen Matrizen zu einem großen **Gleichungssystem assemblieren**. Du kannst die **Randbedingungen** richtig anbringen — fest, beflammt, isoliert, luftgekühlt. Du kannst das System **lösen**, in der Zeit **voranschreiten** und die Wärme an die **Verformung koppeln**. Und, am wichtigsten: Du kannst dem Ergebnis **misstrauen**, bis es die Checkliste bestanden hat. Das ist die ganze Finite-Elemente-Methode. Es gibt nichts Geheimes mehr.

Die schönste Art, dieses Wissen zu behalten, ist, es **umzubauen**. Nimm `fem_balken.py` und mach es zu *deinem* Programm: ein anderes Bauteil, ein anderes Material, eine andere Frage aus deinem Alltag. Die drei Abschlussprojekte unten sind Startpunkte — aber der beste Umbau ist der, den du selbst erfindest.

Roter Faden

Zurück: überall — das war der Zweck dieses Kapitels. Jedes Modul trug seine Kapitelnummer: das Netz aus 1 und Kapitel 11, die Zeitschleife aus Kapitel 9, die Ränder aus Kapitel 8, der Löser aus 2, die Kopplung aus Kapitel 13 und Kapitel 14, der Wachhund und die Checkliste aus Kapitel 12. Der

Bimetall-Cliffhanger aus Übung 13.3 wird in Übung 15.3 endgültig eingelöst, und der Robin-Ausblick aus Kapitel 8 ist in Reihe C ausgerechnet.

Vor: hinaus aus dem Buch. Die Grenzen-Tafel zeigt sechs Türen, und OpenGeoSys wartet dahinter. Von hier an ist die Landkarte deine.

Übungen

Ü 15.1 (Verstehen). Führe zwei der fünf Versuchsreihen selbst durch und schreibe für jede ein **Protokoll** nach dem Rahmen aus Abbildung 15.3 (Frage → Vorhersage → Regler → Ergebnis → Schluss). Wähle etwa Reihe A (Netz) und Reihe C (Ränder). Wichtig ist der zweite Schritt: Leg deine Vorhersage schriftlich fest, *bevor* du rechnest. (Dies ist die ausformulierte Fassung von Übung 1.2 — jetzt mit dem offenen Programm statt der Blackbox.)

> **Musterlösung zu Ü 15.1**
>
> Ein Beispiel-Protokoll für Reihe A: **Frage** — reicht `16×4` für die Auslenkung? **Vorhersage** — grobe Netze biegen zu steif (Kapitel 14), also wächst die Zahl mit feinerem Netz und pendelt sich unter der Handformel 12 mm ein. **Regler** — nur `NX`, `NY` (Elementtyp, Material, Zeit fest). **Ergebnis** — $1{,}2 \to 3{,}7 \to 7{,}7 \to 9{,}6$ mm für `4×1`, `8×2`, `16×4`, `24×6`. **Schluss** — Vorhersage bestätigt; `16×4` trifft rund 64 %, `24×6` rund 80 % der Handformel. Die Auslenkung ist noch nicht ganz konvergiert, für einen qualitativen Eindruck aber gut. Der Schluss hält der Checkliste stand (Konvergenz geprüft, Nulltest wäre der gleichmäßig warme freie Träger mit Spannung null).

Ü 15.2 (Verändern). Schneide ein **Loch** in den Träger. Das Programm bringt dafür die Funktion `netz_mit_loch` mit (ab Zeile 621 in `fem_balken.py`), die ein rechteckiges Loch aus dem Dreiecksnetz stanzt. Belaste den Träger auf **gleichmäßige Erwärmung** ($\Delta T = 100$ K) und halte ihn dabei längs fest (`beide_enden_laengs_fest`), sodass überall Druckspannung herrscht. Wo wird die Spannung am größten? Vergleiche mit dem vollen Träger ohne Loch.

> **Musterlösung zu Ü 15.2**
>
> Im vollen Träger steht überall dieselbe Längsspannung $E\,\alpha\,\Delta T = 210\,000 \cdot 12 \cdot 10^{-6} \cdot 100 = 252\ \mathrm{N/mm^2}$ Druck (Kapitel 13) — schon das liegt über der S235-Streckgrenze, der voll eingespannte Träger fließt also bereits. Mit Loch **drängen sich die Kraftlinien** um das Loch herum und rücken an seinen Ecken eng zusammen: An den Loch-Ecken

steigt die Spannung auf rund $468\ \mathrm{N/mm^2}$ — ein Konzentrationsfaktor von etwa $1{,}9$ auf diesem groben Netz (ein feineres Netz und ein *rundes* Loch zeigen den bekannten Faktor ≈ 3). Das ist die **Kerbwirkung**.

> **Kerbwirkung: warum Schiffe an eckigen Luken zerbrachen**
>
> Die Kerbwirkung ist keine Rechenspielerei. Im Zweiten Weltkrieg baute die USA rund 2700 **Liberty-Frachter** in Rekordzeit — durchgehend *geschweißt* statt genietet, mit *rechteckigen* Ladeluken im Deck. Etliche brachen im kalten Nordatlantik plötzlich auseinander, manche mitten entzwei. Die Rissanalyse zeigte: Fast alle Brüche begannen an den **scharfen Ecken der Luken**, wo sich die Spannung konzentrierte — verschlimmert durch Schweißnähte genau in der Ecke und durch den bei Kälte spröden Stahl (Sridhar 2016). Eine *runde* Ecke hätte die Spitze entschärft. Seither werden Luken, Fenster und Ausschnitte gerundet — dieselbe Physik, die dein Loch-Experiment zeigt, nur mit tödlichem Ausgang. Genau deshalb warnt Kapitel 12: Ein grobes Netz *unterschätzt* solche Spitzen — man muss gezielt verfeinern, um sie zu sehen.

Ü 15.3 (Übertragen — Abschlussprojekt, eines wählen). Bau das Programm zu *deinem* um. Jedes Projekt braucht ein **Verifikations-Protokoll**: Womit prüfst du, dass das Ergebnis stimmt (Nulltest, Handformel, Größenordnung)?

(a) Der echte Bimetall-Streifen. Schichte zwei Materialzonen übereinander — untere Hälfte Aluminium, obere Hälfte Stahl — und erwärme gleichmäßig. Jetzt krümmt sich der Streifen *wirklich*, denn die beiden Metalle liegen **nebeneinander** und halten einander fest (das löst den Cliffhanger aus Übung 13.3 ein). Prüfe die Durchbiegung gegen die Bimetall-Handformel von Timoshenko. Lösungsskript: `loesungen/kap15_ue3_bimetall.py`.

(b) Die Kühlrippe. Ersetze den Träger durch eine schmale, hohe Rippe, die unten an einem heißen Körper sitzt und oben in die Luft ragt: unten ein Temperatur-Rand, ringsum **Robin**-Luftkühlung. Wie weit oben ist die Rippe noch warm? (Verifikation: die Wärmestrombilanz — was unten hereinkommt, muss seitlich als Kühlung wieder hinaus.)

(c) Die zweite Flamme von unten. Setze eine zweite Flamme an die Unterkante des Trägers, versetzt zur ersten. Wo treffen sich die beiden Wärmefronten, und wie verbiegt sich der Träger, wenn oben und unten verschieden geheizt wird? (Verifikation: Schalte beide Flammen gleich heiß — dann muss das Höhengefälle und damit die Biegung verschwinden.)

> **Musterlösung zu Ü 15.3 (a)**
>
> Das Skript `loesungen/kap15_ue3_bimetall.py` weist jedem Dreieck sein eigenes Material zu (unten Alu, oben Stahl) und assembliert damit — sonst ist es die Rechenkette aus Kapitel 14. Ergebnis: Der Streifen krümmt sich, die Alu-Seite außen (sie will sich stärker dehnen). Die Timoshenko-Handformel liefert $\kappa = 24\,(\alpha_{\text{oben}} - \alpha_{\text{unten}})\,\Delta T / [H\,(14 + n + 1/n)]$ mit $n = E_{\text{unten}}/E_{\text{oben}}$, also eine Durchbiegung von $7{,}6$ mm. Die FEM nähert sich ihr von unten ($1{,}5 \to 4{,}2 \to 5{,}7 \to 6{,}5$ mm für `8×2` bis `32×8`) — wieder die CST-Biegesteifheit aus Kapitel 14. Der Bimetall-Cliffhanger aus Übung 13.3 ist damit endgültig eingelöst: Nebeneinander liegende, verschieden dehnende Fasern krümmen den Verbund und erzeugen Spannung.

Das Kleingedruckte

Auch dieses Finale bleibt bei den **bewussten Vereinfachungen** des Buches (Strategie §7), und es benennt sie ehrlich: Die Mechanik rechnet nur auf **Dreiecken** (P1) — löst die Wärme auf Rechtecken, triangulieren wir dasselbe Gitter für die Kopplung (die Knoten sind ja identisch). Die Kopplung ist **Einweg** (Wärme → Verformung, nicht zurück). Das Material ist **linear und isotrop**, seine Kennwerte temperaturunabhängig — die folgenreichste Grenze im Brandfall, wie der Kasten an der Spitze der Grenzen-Tafel zeigt. Die Netze sind **klein** genug fürs reine Python im Browser. Und die Brandgase koppeln über einen Robin-Übergang mit dem Normwert $\alpha_c = 25\ \mathrm{W/(m^2K)}$: realistisch, aber schwach — ein Konvektionsfeuer wölbt den Träger in einer halben Stunde nur um Bruchteile eines Millimeters; die kräftige, sichtbare Krümmung entsteht erst im voll entwickelten Brand, wenn die Unterseite selbst richtig heiß wird. Jede dieser Grenzen steht in der Tafel des Weiterlernens, und jede ist eine Einladung, weiterzugehen.

Teil VI.

Teil VI — Vertiefung: Die Methode wandert (Biologie)

16. Der Knochen als Träger: FEM in der Biologie

Am Ende von 3 stand eine Einladung: Nimm `fem_balken.py`, das Programm, das den brennenden Träger von der Wärme bis zur Verformung durchrechnet, und mach es zu *deinem* — ein anderes Bauteil, ein anderes Material, eine andere Frage. Dieses Kapitel nimmt die Einladung beim Wort. Wir wechseln das Reich: vom Stahlträger unter der Betondecke zu einem Oberschenkelknochen unter dem Gewicht eines Menschen. Und wir stellen fest, dass die Maschine unverändert bleibt.

Das ist der eigentliche Grund, dieses Kapitel zu schreiben. Fünfzehn Kapitel lang sah es aus, als ginge es um *einen* Träger im Feuer. In Wahrheit ging es die ganze Zeit um *eine Methode*. Der Beweis dafür ist, dass wir sie ohne eine einzige neue Löser-Zeile auf etwas Lebendiges übertragen können. Ein Knochen ist kein Stahl — aber für kleine, kurze Belastungen ist er ein Träger wie jeder andere: Er hat ein Netz, eine Last, ein Auflager und Spannungen, die wir lesen können.

Beim Gehen erreicht die Kraft, mit der der Hüftkopf in die Pfanne drückt, in Messungen mit instrumentierten Prothesen grob das Zweieinhalbfache des Körpergewichts (Bergmann u. a. 2001) — bei einem Menschen von 75 kg also rund 1,8 kN. Die feinen Knochenbälkchen im Inneren des Knochens sind nicht zufällig angeordnet; Belastung ist einer von mehreren Einflüssen auf ihre Architektur. Dein Programm aus 3 kann einen bewusst vereinfachten Ausschnitt davon untersuchen.

> **Ein neues Reich — kein neuer Erzählstrang**
>
> Die Brand-Szene ruht ab hier. Dieses Kapitel ist eine **Zugabe nach dem Finale**: Wir feiern, dass die Methode wandert, statt den Träger weiter im Feuer zu lassen. Der Wärmeteil aus 3 bleibt stehen, wo er ist — der Knochen wird **rein mechanisch** gerechnet.

Lernziele

Nach diesem Kapitel kannst du …

1. … erklären, warum ein tragender Knochen dasselbe FEM-Problem ist wie ein Träger — Netz, Last, Auflager, Spannungen — und warum die Hüftkontaktkraft am Gelenk wirkt, der Muskel aber nicht als zweite Last zusätzlich angesetzt wird,
2. … begründen, warum das Material eines Knochens **ortsabhängig** ist, und die Kette *kalibrierter CT-Wert → Scheindichte → E-Modul* erklären, ohne ein Röntgenbild, einen CT-Wert und eine Dichte gleichzusetzen,
3. … die Assemblierung mit **elementweise verschiedenem E** ausführen — den Baustein aus Kapitel 14, jetzt mit einer Materialtabelle je Dreieck,
4. … E-Modul-, Dehnungs- und Spannungsfelder unterscheiden und das Ergebnis mit der Prüf-Checkliste aus Kapitel 12 kontrollieren,
5. … das **Wolffsche Gesetz** und moderne Remodeling-Modelle als Ausblick auf eine Rückkopplung erklären, ohne daraus schon einen Algorithmus abzuleiten,
6. … die bewussten Grenzen benennen (2D statt 3D, isotrop statt faserig, statische Ersatzrechnung statt Wachstum) — Ehrlichkeit wie in Kapitel 12 und

    3.

⚠ Naheliegende Vermutung

Vermutung: *„Biologie ist zu weich und lebendig für die harte Mathematik der FEM.”*
Warum sie naheliegt: Knochen wächst, heilt, verändert sich — nichts davon kennt der starre Stahlträger. Und „weich” klingt nach etwas, das man nicht wie einen Balken durchrechnen kann.
Was stattdessen stimmt: Für kleine, kurzzeitige Verformungen lässt sich ein Knochen näherungsweise linear-elastisch rechnen — mit denselben Gleichungen wie der Träger. Die harte äußere Schale (Kompakta) liegt mit etwa 17 GPa zwar weit unter Stahl (210 GPa), ist aber keineswegs weich wie Gewebe; sie ist steifer als viele Kunststoffe. Das Modell ist bewusst linear und isotrop. Dass sich Knochen *langfristig* an Belastung anpassen, öffnet am Kapitelende eine Tür — es ersetzt aber nicht die statische Rechnung, sondern baut auf ihr auf.

16.1. Der Transfer, ausdrücklich

Legen wir die beiden Bauteile nebeneinander. Der Träger aus Kapitel 14 war ein Rechteck aus Stahl, unten vom Brand erwärmt, an einem Ende eingespannt. Der Femurschnitt ist eine krumme Kontur aus Knochen, oben von der Hüftkraft belastet, am Schaftfuß festgehalten. Verschieden sieht das aus — gleich ist die Frage: Wie verformt sich das Bauteil, und wo sitzen die Spannungen?

Der Träger (Kapitel 14)	Der Knochen (hier)
Rechteckscheibe, Stahl	Femur-Längsschnitt, Knochen
P1-Dreiecksnetz	dasselbe P1-Dreiecksnetz
eingespanntes Ende (Auflager)	abgeschnittener Schaftfuß (künstliches Auflager)
Wärme dehnt die Fasern	Hüftkontaktkraft drückt auf den Kopf
ein E-Modul für alles	ein E-Modul **je Ort** (neu)
$u \to \varepsilon \to \sigma$	dieselbe Kette $u \to \varepsilon \to \sigma$

Zwei Dinge übersetzen wir, eine Zutat ist wirklich neu.

Übersetzung 1 — die Last kommt vom Gelenk. Das Hüftgelenk überträgt die Last auf den Hüftkopf. Die gemessene Hüftkontaktkraft von 1,8 kN ist bereits die *Resultierende* aus Körpergewicht und Muskelzug — sie enthält die Muskelwirkung schon. Wir setzen deshalb **keine** zweite, erfundene Muskelkraft zusätzlich an; das würde die Wirkung doppelt zählen. Das Auflager ist der abgeschnittene Schaftfuß: eine Modell-Lagerung, kein echtes Gelenk. Ohne Wärmeteil entfällt die thermische rechte Seite von Kapitel 14 vollständig.

Übersetzung 2 — Misstrauen als Handwerk. Die Checkliste aus Kapitel 12 („Traue keinem bunten Bild") wird zur biologischen Frage: Ist eine Spannungsspitze am Schenkelhals echt, oder ist sie nur Kerbwirkung an einer zu grob modellierten Ecke? Wir werden genau das prüfen.

Die eine neue Zutat — das Material ist eine Landkarte. Der Knochen hat kein einheitliches Material. Sein E-Modul ist ortsabhängig: außen die steife kortikale Schale, innen der weiche spongiöse Kern. Das ist die konzeptionelle Fracht dieses Kapitels. Eine vollständige Umbau-Rechnung, in der die Belastung das Material *verändert*, wäre eine zweite neue Idee — sie bekommt ein eigenes, noch folgendes Kapitel; hier erscheint sie nur als Ausblick.

16.2. Die 2D-Idealisierung, ehrlich

Ein echter Femur ist dreidimensional, gekrümmt und verdreht. Wir nehmen einen **Längsschnitt** durch die Frontalebene — Hüftkopf, Schenkelhals, Schaft — und vernetzen ihn mit allgemeinen P1-Dreiecken wie in 1. Die runde Außenkontur nähern wir stückweise durch gerade Randkanten an; die Geometrie ist von Hand parametrisch gezeichnet, **nicht** aus einem CT-Bild importiert.

Was wir weglassen — und warum das Bild trotzdem stimmt

Wie beim „Rechteck statt I-Profil" aus Kapitel 1 nennen wir die Annahmen offen:

- **Ebener Spannungszustand.** Wir rechnen mit derselben Materialtabelle wie in Kapitel 14 — dem ebenen Spannungszustand einer in die Tiefe freien Scheibe. Der Knochen ist das nicht ganz; aber das Buch lehrt bewusst nur diesen Zustand, und für ein *qualitatives* Lastbild ist die Vereinfachung brauchbar.
- **Ersatzscheibe fester Dicke.** Der 2D-Schnitt steht für eine Scheibe der Dicke 0,02 m. Die volle Modellresultierende von 1,8 kN wird auf eine festgelegte Kopfkante und diese Ersatzdicke verteilt. **Absolute Spannungen gelten deshalb nur für dieses Modell** und sind keine klinische Aussage über einen echten Patienten.
- **Verrundeter Hals.** Wo der Schenkelhals in den Schaft übergeht, ist ein echter Knochen verrundet — und wir runden dort ebenfalls. Eine scharfe einspringende Ecke erzeugte sonst eine Spannungs*singularität*: eine Spitze, die mit jedem feineren Netz weiterwächst und nie konvergiert. Die Verrundung macht die Halsspannung zu einer sinnvollen, konvergierenden Zahl. Die scharfe Ecke lassen wir später **einmal bewusst** auftreten — als Kerbwirkungs-Demo im Prüfteil.

16.3. Das ortsabhängige Material — die neue Idee

Wie kommt man vom Knochen zum E-Modul? Nicht in einem Schritt. Die ehrliche Messkette lautet:

$$\text{CT-Zahl} \xrightarrow{\text{Kalibrierung}} \text{Aschedichte} \;\rightarrow\; \text{Scheindichte } \rho \xrightarrow{\text{Morgan}} E.$$

Eine rohe Graustufe aus dem Röntgen- oder CT-Bild ist noch **keine** Dichte; erst eine kalibrierte Auswertung führt über die Aschedichte zur Scheindichte (Schileo u. a. 2008). Wir importieren kein CT, sondern verwenden ein **synthetisches Scheindichtefeld**, das nur die letzte Stufe nachbildet: dichter entlang des Hauptlastpfads (vom Kopf zum medialen Schaftfuß), dünner in den Randzonen.

Die letzte Stufe, von der Scheindichte zum E-Modul, ist das Herzstück. Für den spongiösen Knochen benutzen wir das Potenzgesetz von Morgan (Morgan u. a. 2003) (Dichte ρ in g/cm³, E in GPa):

$$E \;=\; 6{,}95 \cdot \rho^{\,1{,}49}.$$

Dieses Gesetz gilt nur im belegten spongiösen Bereich $\rho \in [0{,}2;\ 1{,}0]$ g/cm³. Wir **extrapolieren es nicht** bis zur kortikalen Dichte (rund 1,9 g/cm³) — die kortikale Schale bekommt separat ihren gemessenen Wert $E = 17$ GPa. Morgan und Kollegen betonen außerdem, dass dieser Dichte-Steifigkeits-Zusammenhang vom **anatomischen Ort** abhängt: Die Formel ist ein didaktischer Arbeitskanon für den spongiösen Bereich, kein universelles Knochengesetz (Morgan u. a. 2003). Die folgende Zelle rechnet ein paar Belegwerte nach (Modul `programme/gemeinsam/dichtefeld.py`):

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from dichtefeld import morgan_e_modul

print("Scheindichte | Morgan-E (spongioes)")
print("-------------+---------------------")
for dichte in [0.2, 0.3, 0.5, 0.8, 1.0]:
    e_gpa = morgan_e_modul(dichte) / 1e9
    print(f"  {dichte:4.1f} g/cm3  |   {e_gpa:5.2f} GPa")
print()
print("kortikale Schale (fester Messwert): 17,00 GPa")
```

```
Scheindichte | Morgan-E (spongioes)
-------------+---------------------
   0.2 g/cm3  |    0.63 GPa
   0.3 g/cm3  |    1.16 GPa
   0.5 g/cm3  |    2.47 GPa
   0.8 g/cm3  |    4.98 GPa
   1.0 g/cm3  |    6.95 GPa

kortikale Schale (fester Messwert): 17,00 GPa
```

Man liest ab: Über den ganzen spongiösen Bereich reicht E von rund 0,6 bis knapp 7 GPa — und die kortikale Schale ist mit 17 GPa noch einmal gut doppelt so steif wie der dichteste Trabekelknochen. Der Knochen ist also nicht *ein* Material, sondern eine ganze Landkarte von Steifigkeiten. Dieselbe Kurve als Bild:

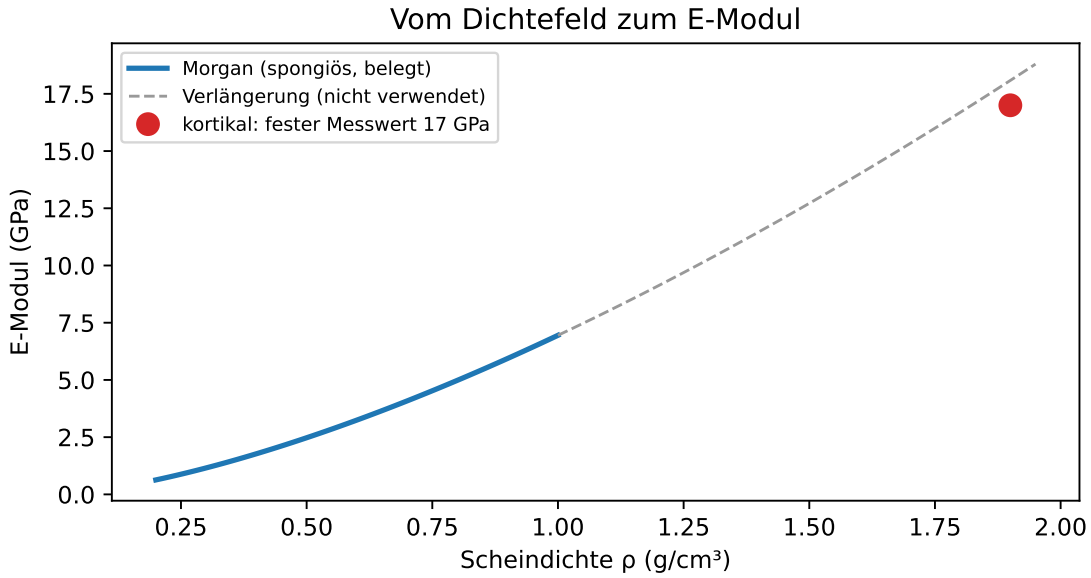

Abbildung 16.1.: Das Morgan-Gesetz E = 6,95·ρ^1,49 im belegten spongiösen Bereich (0,2–1,0 g/cm³, durchgezogen). Die kortikale Schale (roter Punkt, fester Messwert 17 GPa bei ρ ≈ 1,9 g/cm³) wird bewusst NICHT über die verlängerte Kurve (gestrichelt) bestimmt: Morgans Daten decken den kortikalen Dichtebereich gar nicht ab, deshalb der separate Messwert.

Ein historischer Vergleich: Carter & Hayes

Vor Morgan beschrieben Carter & Hayes (1977) die Steifigkeit aus Druckversuchen mit einer *kubischen* Dichteabhängigkeit, $E \approx 3{,}79 \cdot \rho^3$, zusätzlich schwach von der Dehnrate abhängig (Carter und Hayes 1977). Das Modul `dichtefeld.py` stellt beide Gesetze bereit — aber Carter & Hayes ist ein historischer Vergleich mit einer hier fest gewählten Dehnrate, **kein** universeller Materialschalter. Wir rechnen mit Morgan.

Die eigentliche Schlüsselgrafik dieses Kapitels ist diese Landkarte, über den vernetzten Femurschnitt gelegt:

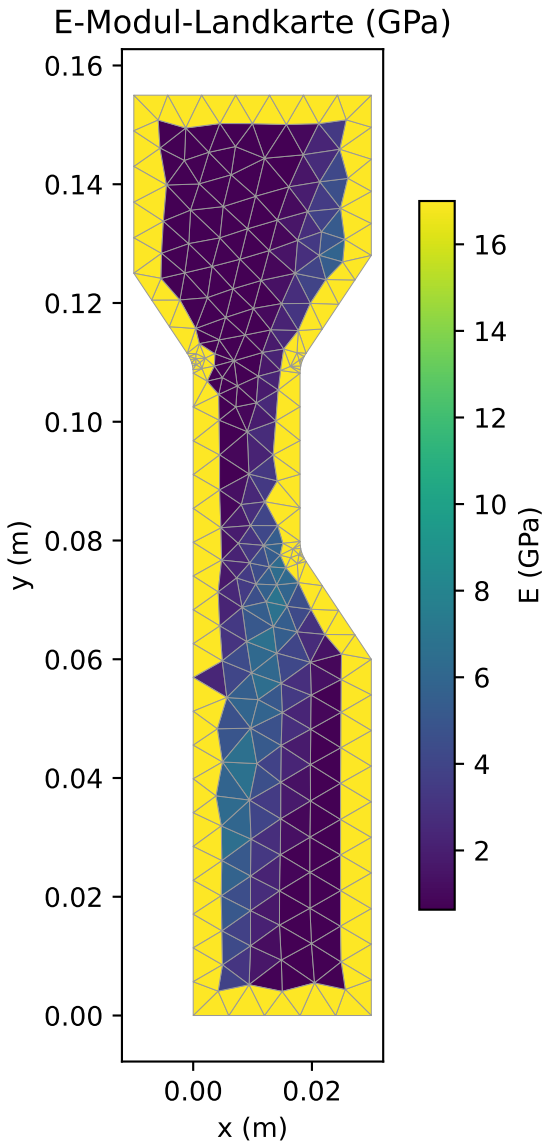

Abbildung 16.2.: Die Material-Landkarte des Femurschnitts: jedes Dreieck trägt seinen eigenen E-Modul. Die helle kortikale Randschale (17 GPa) umschließt den dunkleren spongiösen Kern, dessen Steifigkeit dem synthetischen Dichtefeld folgt — dichter (heller) entlang des Lastpfads vom Kopf zum medialen Schaftfuß.

16.4. Assemblieren mit variablem E

Was ändert sich am Programm? Erstaunlich wenig. Der Baustein aus Kapitel 14, `baue_element_6x6`, nahm den E-Modul schon immer *pro Dreieck* entgegen — wir haben ihn bisher nur für jedes Dreieck mit demselben Wert gefüttert. Jetzt geben wir jedem Dreieck seinen eigenen. Die einzige Änderung ist, dass die Materialtabelle je Dreieck aus dessen Dichte gebaut wird — eine Zeile.

Sehen wir es an einem Miniaturbeispiel: ein Streifen aus zwei Dreiecken, das linke steif, das rechte weich. Die Funktion `assembliere_elastik_inhomogen` nimmt eine **Liste** von E-Moduln, einen Wert je Dreieck:

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from elastik2d import assembliere_elastik_inhomogen

# Ein Rechteck-Streifen aus zwei Dreiecken.
knoten = [(0.0, 0.0), (0.1, 0.0), (0.0, 0.05), (0.1, 0.05)]
dreiecke = [(0, 1, 3), (0, 3, 2)]

# Zwei verschiedene Materialien: links steif, rechts weich.
e_module = [17.0e9, 2.0e9]      # ein E-Modul JE Dreieck
querkontraktion = 0.30
tiefe = 0.02

matrix = assembliere_elastik_inhomogen(knoten, dreiecke, e_module,
                                       querkontraktion, tiefe)

# Das steifste Diagonalelement heraussuchen (reines Python, keine Kurzform).
groesstes = 0.0
for i in range(len(matrix)):
    if matrix[i][i] > groesstes:
        groesstes = matrix[i][i]

print(f"Systemmatrix: {len(matrix)} x {len(matrix)} (4 Knoten x 2 Richtungen)")
print(f"steifstes Diagonalelement: {groesstes / 1e6:8.1f} MN/m")
```

```
Systemmatrix: 8 x 8 (4 Knoten x 2 Richtungen)
steifstes Diagonalelement:    406.3 MN/m
```

Der Assemblierungs-Loop ist Wort für Wort der aus 1 und Kapitel 14 — für jedes Dreieck den 6×6-Steckbrief bauen und an die Freiheitsgrade seiner

drei Knoten addieren. Neu ist allein, dass der Steckbrief mit `e_module[nummer]` statt mit einem globalen `e_modul` gebaut wird. Das Kapitelprogramm ruft danach genau diese geprüfte Funktion über alle 427 Dreiecke des Femurs auf, statt den Loop zu wiederholen.

Auch das Spannungslesen braucht das lokale E

Es gibt eine zweite Stelle, an der das ortsabhängige Material zählt: das Zurückrechnen der Spannungen. Eine steife kortikale Schale trägt bei *gleicher* Dehnung mehr Spannung als der weiche Kern. Würde man die Spannungen mit einem globalen E auslesen, stimmte das Verhältnis der Felder nicht. Deshalb gibt es neben `assembliere_elastik_inhomogen` auch `elementspannungen_inhomogen`, die jedem Dreieck seine *eigene* Materialtabelle gibt. Beide Funktionen sind so gebaut, dass sie bei überall gleichem E Zeile für Zeile dasselbe liefern wie die vertrauten Funktionen aus Kapitel 14.

16.5. Lasten, Lager — und die Rechnung

Die Hüftkontaktkraft von 1,8 kN wirkt nach unten auf eine kurze Kante der Kopfoberseite. Wie in Kapitel 13 verteilen wir eine Kantenlast in *äquivalente Knotenkräfte*: Jede Randkante gibt ihren Anteil hälftig an ihre beiden Endknoten. Der Schaftfuß bei $y = 0$ wird in beide Richtungen festgehalten.

Der schwimmende Knochen

Wie in Kapitel 8 und Kapitel 14 gilt: Ohne Auflager schwimmt auch der Knochen davon — die Systemmatrix hätte Starrkörpermoden und wäre nicht lösbar. Das künstliche Auflager am Schaftfuß sperrt diese Moden. Es steht für „hier geht der Knochen weiter, den wir nicht mitmodellieren", nicht für ein echtes Gelenk.

Jetzt läuft die vertraute Kette. Die folgende Zelle assembliert das ganze Femurnetz mit seiner Material-Landkarte, bringt die Hüftlast auf, hält den Schaftfuß fest, löst — und liest die Kennzahlen aus. Sie benutzt nur geprüfte Funktionen aus `elastik2d.py`:

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
sys.path.insert(0, os.path.join("..", "programme", "kap16"))
from elastik2d import (assembliere_elastik_inhomogen,
                       elementspannungen_inhomogen, loese_elastik)
```

```
from dichtefeld import e_module_landkarte
from femur_netz import KNOTEN, DREIECKE, KORTIKAL, DICHTEN

QUERKONTRAKTION = 0.30
TIEFE = 0.02
HUEFTLAST = 1800.0
LAST_Y = 0.155

# 1) Material-Landkarte: ein E-Modul je Dreieck.
e_module = e_module_landkarte(DICHTEN, KORTIKAL)

# 2) Systemmatrix mit variablem E.
matrix = assembliere_elastik_inhomogen(KNOTEN, DREIECKE, e_module,
                                       QUERKONTRAKTION, TIEFE)

# 3) Hueftlast als aequivalente Knotenkraefte auf die Kopfkante (nach unten).
#    Erst die Knoten auf der Kopfoberkante sammeln, dann nach x sortieren
#    (ueber Paare (x, Knotennummer) -- reines Python, ohne Lambda).
kanten_paare = []
for i in range(len(KNOTEN)):
    x = KNOTEN[i][0]
    y = KNOTEN[i][1]
    if abs(y - LAST_Y) < 1e-6 and -0.010 - 1e-9 <= x <= 0.030 + 1e-9:
        kanten_paare.append((x, i))
kanten_paare.sort()
auf_kante = []
for paar in kanten_paare:
    auf_kante.append(paar[1])
traktion = HUEFTLAST / ((0.030 - (-0.010)) * TIEFE)
kraefte = [0.0] * (2 * len(KNOTEN))
for stelle in range(len(auf_kante) - 1):
    links = auf_kante[stelle]
    rechts = auf_kante[stelle + 1]
    halbe = traktion * abs(KNOTEN[rechts][0] - KNOTEN[links][0]) * TIEFE / 2.0
    kraefte[2 * links + 1] = kraefte[2 * links + 1] - halbe
    kraefte[2 * rechts + 1] = kraefte[2 * rechts + 1] - halbe

# 4) Schaftfuss festhalten und loesen.
lager = {}
for i in range(len(KNOTEN)):
    if KNOTEN[i][1] < 1e-9:
        lager[2 * i] = 0.0
        lager[2 * i + 1] = 0.0
```

```
verschiebungen = loese_elastik(matrix, kraefte, lager)

# 5) Spannungen mit dem jeweiligen Element-E (rein mechanisch: keine Waerme).
null_dt = [0.0] * len(DREIECKE)
spannungen = elementspannungen_inhomogen(KNOTEN, DREIECKE, verschiebungen,
                                         e_module, QUERKONTRAKTION, 0.0, null_dt)

# Kennzahlen: Kopfsenkung und flaechengewichtete Halsspannung.
tiefste = 0.0
for i in range(len(KNOTEN)):
    if verschiebungen[2 * i + 1] < tiefste:
        tiefste = verschiebungen[2 * i + 1]
kopfsenkung_mm = -tiefste * 1000.0

def von_mises(spannung):
    sxx, syy, txy = spannung
    return (sxx * sxx - sxx * syy + syy * syy + 3.0 * txy * txy) ** 0.5

summe = 0.0
flaeche_hals = 0.0
for nummer in range(len(DREIECKE)):
    d = DREIECKE[nummer]
    sy = (KNOTEN[d[0]][1] + KNOTEN[d[1]][1] + KNOTEN[d[2]][1]) / 3.0
    if 0.082 < sy < 0.106:
        (x0, y0), (x1, y1), (x2, y2) = KNOTEN[d[0]], KNOTEN[d[1]], KNOTEN[d[2]]
        flaeche = 0.5 * abs((x1 - x0) * (y2 - y0) - (x2 - x0) * (y1 - y0))
        summe = summe + von_mises(spannungen[nummer]) * flaeche
        flaeche_hals = flaeche_hals + flaeche
halsspannung_mpa = summe / flaeche_hals / 1e6

print(f"Kopfsenkung (grobes Lehrnetz)     : {kopfsenkung_mm:.3f} mm")
print(f"Halsspannung, flaechengewichtet   : {halsspannung_mpa:.2f} MPa")
```

```
Kopfsenkung (grobes Lehrnetz)     : 0.115 mm
Halsspannung, flaechengewichtet   : 5.22 MPa
```

Zwei Zahlen zum Merken: Der Hüftkopf senkt sich um rund **ein Zehntel Millimeter**, und im Schenkelhals herrscht eine gemittelte Von-Mises-Vergleichsspannung von gut **5 MPa**. Das grobe Lehrnetz rechnet der reine-Python-Löser in Sekunden.

Grobe Netze sind zu steif — der Fall für Kapitel 12

Die hier gezeigte Kopfsenkung (rund 0,12 mm) ist noch nicht die ganze Wahrheit. Wie in Kapitel 12 und Kapitel 14 sind grobe P1-Netze **zu steif**: Sie unterschätzen die Verformung und nähern sich der richtigen Lösung *von unten*. Verfeinert man das Netz systematisch, konvergieren Kopfsenkung und gemittelte Halsspannung gegen die validierten Referenzwerte **0,15 mm** und **5,0 MPa** (nachgerechnet mit dem professionellen Löser scikit-fem auf immer feineren Netzen — Protokoll in `validierung/kap16_knochen/`). Die grobe Zahl ist nicht falsch, nur noch nicht konvergiert.

Das Ergebnisbild zeigt links die Material-Landkarte noch einmal, in der Mitte das überhöht verformte Netz und rechts das Von-Mises-Feld:

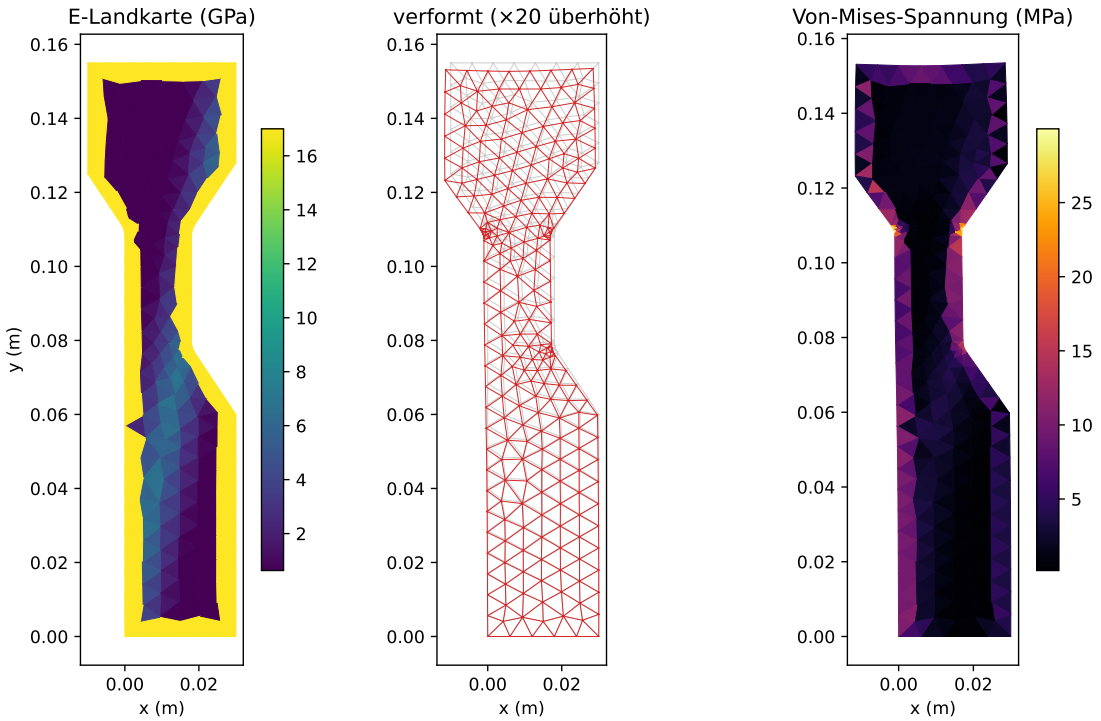

Abbildung 16.3.: Ergebnis des Femurmodells, drei Bilder. Links die E-Modul-Landkarte (kortikale Schale 17 GPa, spongiöser Kern nach Dichtefeld). Mitte: das überhöht (×20) verformte Netz über der Ausgangslage — der Kopf senkt sich, der Hals biegt sich leicht. Rechts die Von-Mises-Vergleichsspannung — hoch am Schenkelhals, wo der Querschnitt am schmalsten ist, und in der lasttragenden kortikalen Schale. Von Mises ist hier nur ein Farbwert, kein Versagenskriterium.

Lesen ohne Übertreiben. Das Bild zeigt hohe Spannung am Schenkelhals und in der kortikalen Schale. Verlockend wäre der Satz „die Schale trägt alles" — aber hohe Spannung allein beweist noch keinen Lastanteil. Von Mises fasst nur die drei Spannungskomponenten zu *einem* skalaren Farbwert zusammen; es ist hier ein Darstellungswert, **kein** Versagenskriterium und keine Aussage darüber, ab wann ein Knochen bricht.

> **Echte Spitze oder Kerbwirkung? — die Kap.-12-Frage**
>
> Am Übergang vom Hals zum Kopf leuchten im Von-Mises-Bild kleine Spitzen auf. Sind die echt? Die Prüfung ist die aus Kapitel 12: **Netz**

verfeinern und nachschauen. Für die *verrundete* Kontur konvergieren Kopfsenkung und gemittelte Halsspannung — sie sind belastbar. Für eine **scharfe**, unverrundete Ecke dagegen wächst die Spitzenspannung mit jedem feineren Netz weiter (in der Nachrechnung von rund 9 über 12 auf über 13 MPa und weiter): Das ist eine mathematische Singularität, kein physikalischer Messwert. Genau darum haben wir den Hals verrundet — damit die Halsspannung eine Zahl ist, der man trauen kann, und nicht ein Artefakt der Netzweite.

Der Lastaufbau, Bild für Bild. Weil das Modell linear ist, müssen wir nicht für jede Laststufe neu rechnen: Wächst die Hüftkraft von null auf 1,8 kN, so wachsen Verformung und Von-Mises-Feld *proportional* mit. Die folgende Darstellung skaliert die eine berechnete Lösung — der Kopf sinkt tiefer, das Spannungsfeld leuchtet stärker auf, aber das *Muster* bleibt sich gleich. Genau das bedeutet Linearität.

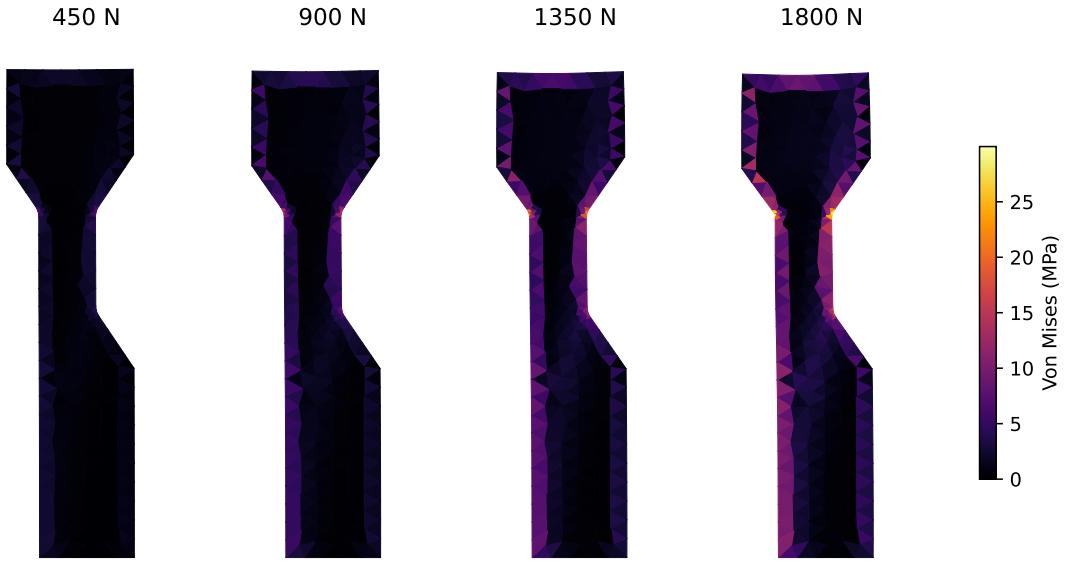

Abbildung 16.4.: Der Lastaufbau am Femur als vier Standbilder (in der HTML-Fassung eine Animation): Hüftkraft 450, 900, 1350 und 1800 N. Verformung (×20 überhöht) und Von-Mises-Feld wachsen proportional mit der Kraft — das Muster bleibt gleich, nur die Beträge steigen. Das ist die Linearität des Modells.

Was man sieht: Bei kleiner Kraft ist alles dunkel und fast unverformt; mit wachsender Last senkt sich der Kopf und der Schenkelhals leuchtet zuerst

auf. Nichts an der *Verteilung* ändert sich — sie ist von Anfang an da, nur schwächer. Ein nichtlineares Material (oder große Verformungen) würde hier ein *anderes* Muster bei starker Last zeigen; unser linear-elastischer Knochen tut das bewusst nicht.

16.6. Probier es aus: die Knochenwerkstatt

Zeit für dein eigenes Experiment. Die folgende interaktive Einheit rechnet ein grobes Femurnetz im Browser. Du kannst zwei Dinge einstellen: den Faktor der Hüftkraft (in Vielfachen des Körpergewichts) und die Steifigkeit des spongiösen Kerns über eine repräsentative Dichte.

> Vorhersage-Punkt
>
> **Bevor du rechnest:** Wenn die Spongiosa *dichter* und damit *steifer* wird — wird die Kopfsenkung dann **größer oder kleiner**? Leg dich fest, dann probier es aus.

```
#| autorun: true
# Die Knochenwerkstatt -- spiele mit Hueftkraft und Spongiosa-Dichte.
# Aendere die beiden Zahlen und druecke Ausfuehren.
KRAFT_FAKTOR = 2.5     # Vielfaches des Koerpergewichts (1 bis 4)
SPONGIOSA_DICHTE = 0.5  # g/cm^3, repraesentativ (0,2 bis 1,0)

KOERPERGEWICHT = 736.0  # N (75 kg)
QUERKONTRAKTION = 0.30
TIEFE = 0.02
E_KORTIKAL = 17.0e9

# --- Netz (grob, ~95 Knoten), einmal offline erzeugt und hier eingesetzt ---
KNOTEN = [(0.0000,0.0000), (0.0300,0.0000), (0.0300,0.0600), (0.0190,0.0765), (0.0187,0.0770), (0.01
DREIECKE = [(9,66,85), (33,2,55), (72,67,73), (16,37,61), (9,35,66), (27,44,86), (58,55,74), (53,10,
KORTIKAL = [0,1,1,1,1,1,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,1,0,1,0,0,1,1,0,0,0,1,0,1,0,0,

# --- Morgan-Gesetz (Quelle: dichtefeld.py) ---
def morgan_e(dichte):
    if dichte < 0.2:
        dichte = 0.2
    if dichte > 1.0:
        dichte = 1.0
    return 6.95 * dichte ** 1.49 * 1.0e9
```

```
# --- E-Modul je Dreieck: kortikal fest, sonst Morgan ---
e_module = []
for nummer in range(len(DREIECKE)):
    if KORTIKAL[nummer] == 1:
        e_module.append(E_KORTIKAL)
    else:
        e_module.append(morgan_e(SPONGIOSA_DICHTE))

# --- Bausteine je Dreieck (Quelle: elastik2d.py, reines Python) ---
def zeltsteigungen(ecken):
    (xa, ya) = ecken[0]
    (xb, yb) = ecken[1]
    (xc, yc) = ecken[2]
    zwei = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    b = [(yb - yc) / zwei, (yc - ya) / zwei, (ya - yb) / zwei]
    c = [(xc - xb) / zwei, (xa - xc) / zwei, (xb - xa) / zwei]
    return b, c, abs(zwei) / 2.0

def b_tabelle(b, c):
    tab = []
    for zeile in range(3):
        neue_zeile = []
        for spalte in range(6):
            neue_zeile.append(0.0)
        tab.append(neue_zeile)
    for ecke in range(3):
        tab[0][2 * ecke] = b[ecke]
        tab[1][2 * ecke + 1] = c[ecke]
        tab[2][2 * ecke] = c[ecke]
        tab[2][2 * ecke + 1] = b[ecke]
    return tab

def materialtabelle(e_modul):
    faktor = e_modul / (1.0 - QUERKONTRAKTION * QUERKONTRAKTION)
    schub = (1.0 - QUERKONTRAKTION) / 2.0
    return [[faktor, faktor * QUERKONTRAKTION, 0.0],
            [faktor * QUERKONTRAKTION, faktor, 0.0],
            [0.0, 0.0, faktor * schub]]

def element_matrix(ecken, e_modul):
    b, c, flaeche = zeltsteigungen(ecken)
    bt = b_tabelle(b, c)
    cc = materialtabelle(e_modul)
```

```
    # cb = C * B (3x6)
    cb = []
    for zeile in range(3):
        neue_zeile = []
        for spalte in range(6):
            summe = 0.0
            for k in range(3):
                summe = summe + cc[zeile][k] * bt[k][spalte]
            neue_zeile.append(summe)
        cb.append(neue_zeile)
    # steck = B-transponiert * cb, mal Tiefe mal Flaeche (6x6)
    steck = []
    for i in range(6):
        neue_zeile = []
        for j in range(6):
            summe = 0.0
            for k in range(3):
                summe = summe + bt[k][i] * cb[k][j]
            neue_zeile.append(summe * TIEFE * flaeche)
        steck.append(neue_zeile)
    return steck

# --- Assemblieren mit variablem E ---
n = len(KNOTEN)
matrix = []
for zeile in range(2 * n):
    neue_zeile = []
    for spalte in range(2 * n):
        neue_zeile.append(0.0)
    matrix.append(neue_zeile)
for nummer in range(len(DREIECKE)):
    d = DREIECKE[nummer]
    ecken = [KNOTEN[d[0]], KNOTEN[d[1]], KNOTEN[d[2]]]
    steck = element_matrix(ecken, e_module[nummer])
    dofs = []
    for ecke in range(3):
        dofs.append(2 * d[ecke])
        dofs.append(2 * d[ecke] + 1)
    for a in range(6):
        for bb in range(6):
            matrix[dofs[a]][dofs[bb]] = matrix[dofs[a]][dofs[bb]] + steck[a][bb]

# --- Hueftlast (nach unten) und Auflager ---
```

```
gesamtlast = KRAFT_FAKTOR * KOERPERGEWICHT
# Kopfkanten-Knoten sammeln und nach x sortieren (ueber Paare, ohne Lambda).
kanten_paare = []
for i in range(n):
    if abs(KNOTEN[i][1] - 0.155) < 1e-6 and -0.011 <= KNOTEN[i][0] <= 0.031:
        kanten_paare.append((KNOTEN[i][0], i))
kanten_paare.sort()
auf_kante = []
for paar in kanten_paare:
    auf_kante.append(paar[1])
traktion = gesamtlast / (0.040 * TIEFE)
kraefte = []
for i in range(2 * n):
    kraefte.append(0.0)
for s in range(len(auf_kante) - 1):
    li, re = auf_kante[s], auf_kante[s + 1]
    halbe = traktion * abs(KNOTEN[re][0] - KNOTEN[li][0]) * TIEFE / 2.0
    kraefte[2 * li + 1] = kraefte[2 * li + 1] - halbe
    kraefte[2 * re + 1] = kraefte[2 * re + 1] - halbe
lager = {}
for i in range(n):
    if KNOTEN[i][1] < 1e-9:
        lager[2 * i] = 0.0
        lager[2 * i + 1] = 0.0

# --- Gauss-Loeser (Quelle: loeser.py) auf den freien Freiheitsgraden ---
frei = []
for dof in range(2 * n):
    if dof not in lager:
        frei.append(dof)
a_frei = []
for i in frei:
    reduzierte_zeile = []
    for j in frei:
        reduzierte_zeile.append(matrix[i][j])
    a_frei.append(reduzierte_zeile)
b_frei = []
for i in frei:
    b_frei.append(kraefte[i])
m = len(frei)
for spalte in range(m):
    pivot = spalte
    for zeile in range(spalte + 1, m):
```

```
        if abs(a_frei[zeile][spalte]) > abs(a_frei[pivot][spalte]):
            pivot = zeile
    a_frei[spalte], a_frei[pivot] = a_frei[pivot], a_frei[spalte]
    b_frei[spalte], b_frei[pivot] = b_frei[pivot], b_frei[spalte]
    for zeile in range(spalte + 1, m):
        faktor = a_frei[zeile][spalte] / a_frei[spalte][spalte]
        for k in range(spalte, m):
            a_frei[zeile][k] = a_frei[zeile][k] - faktor * a_frei[spalte][k]
        b_frei[zeile] = b_frei[zeile] - faktor * b_frei[spalte]
loesung = [0.0] * m
for zeile in range(m - 1, -1, -1):
    rest = b_frei[zeile]
    for k in range(zeile + 1, m):
        rest = rest - a_frei[zeile][k] * loesung[k]
    loesung[zeile] = rest / a_frei[zeile][zeile]
verschiebungen = [0.0] * (2 * n)
for stelle in range(m):
    verschiebungen[frei[stelle]] = loesung[stelle]

# --- Von-Mises-Spannung je Dreieck (mit dem jeweiligen Element-E) ---
def von_mises_dreieck(ecken, ecku, e_modul):
    b, c, flaeche = zeltsteigungen(ecken)
    bt = b_tabelle(b, c)
    dehnung = [0.0, 0.0, 0.0]
    for zeile in range(3):
        summe = 0.0
        for k in range(6):
            summe = summe + bt[zeile][k] * ecku[k]
        dehnung[zeile] = summe
    cc = materialtabelle(e_modul)
    spannung = [0.0, 0.0, 0.0]
    for zeile in range(3):
        summe = 0.0
        for k in range(3):
            summe = summe + cc[zeile][k] * dehnung[k]
        spannung[zeile] = summe
    sxx = spannung[0]
    syy = spannung[1]
    txy = spannung[2]
    return (sxx * sxx - sxx * syy + syy * syy + 3.0 * txy * txy) ** 0.5

vm_feld = []
for nummer in range(len(DREIECKE)):
```

```
    d = DREIECKE[nummer]
    ecken = [KNOTEN[d[0]], KNOTEN[d[1]], KNOTEN[d[2]]]
    ecku = []
    for ecke in range(3):
        ecku.append(verschiebungen[2 * d[ecke]])
        ecku.append(verschiebungen[2 * d[ecke] + 1])
    vm_feld.append(von_mises_dreieck(ecken, ecku, e_module[nummer]) / 1e6)

tiefste = 0.0
for i in range(n):
    if verschiebungen[2 * i + 1] < tiefste:
        tiefste = verschiebungen[2 * i + 1]
print("Hueftkraft      :", round(gesamtlast), "N  (", KRAFT_FAKTOR, "x Koerpergewicht)")
print("Spongiosa-E     :", round(morgan_e(SPONGIOSA_DICHTE) / 1e9, 2), "GPa")
print("Kopfsenkung     :", round(-tiefste * 1000.0, 3), "mm")

# --- Abbildung ---
import matplotlib.pyplot as plt
from matplotlib.tri import Triangulation
tris = [list(d) for d in DREIECKE]
xs = [k[0] for k in KNOTEN]
ys = [k[1] for k in KNOTEN]
e_gpa = [e / 1e9 for e in e_module]
ueberhoehung = 20.0
vx = [xs[i] + ueberhoehung * verschiebungen[2 * i] for i in range(n)]
vy = [ys[i] + ueberhoehung * verschiebungen[2 * i + 1] for i in range(n)]
fig, achsen = plt.subplots(1, 3, figsize=(9.5, 5.2))
tp0 = achsen[0].tripcolor(Triangulation(xs, ys, tris), facecolors=e_gpa,
                          cmap="viridis", edgecolors="0.6", linewidth=0.2)
achsen[0].set_title("E-Landkarte (GPa)")
fig.colorbar(tp0, ax=achsen[0], shrink=0.7)
achsen[1].triplot(xs, ys, tris, color="0.85", linewidth=0.4)
achsen[1].triplot(vx, vy, tris, color="C3", linewidth=0.4)
achsen[1].set_title("verformt (x20)")
tp2 = achsen[2].tripcolor(Triangulation(vx, vy, tris), facecolors=vm_feld,
                          cmap="inferno", edgecolors="face")
achsen[2].set_title("Von Mises (MPa)")
fig.colorbar(tp2, ax=achsen[2], shrink=0.7)
for ax in achsen:
    ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
sys.path.insert(0, os.path.join("..", "programme", "kap16"))
import matplotlib.pyplot as plt
from matplotlib.tri import Triangulation
from elastik2d import (assembliere_elastik_inhomogen,
                       elementspannungen_inhomogen, loese_elastik)
from dichtefeld import morgan_e_modul, E_KORTIKAL
from femur_netz_grob import KNOTEN, DREIECKE, KORTIKAL

# Ausgangseinstellungen des interaktiven Stuecks reproduzieren.
QUERKONTRAKTION = 0.30
TIEFE = 0.02
HUEFTLAST = 2.5 * 736.0     # 2,5 x Koerpergewicht
DICHTE = 0.5                # uniforme Spongiosa-Dichte

e_module = [E_KORTIKAL if flag == 1 else morgan_e_modul(DICHTE) for flag in KORTIKAL]
matrix = assembliere_elastik_inhomogen(KNOTEN, DREIECKE, e_module, QUERKONTRAKTION, TIEFE)

paare = sorted((KNOTEN[i][0], i) for i in range(len(KNOTEN))
               if abs(KNOTEN[i][1] - 0.155) < 1e-6 and -0.011 <= KNOTEN[i][0] <= 0.031)
auf_kante = [i for _, i in paare]
traktion = HUEFTLAST / (0.040 * TIEFE)
kraefte = [0.0] * (2 * len(KNOTEN))
for s in range(len(auf_kante) - 1):
    li, re = auf_kante[s], auf_kante[s + 1]
    halbe = traktion * abs(KNOTEN[re][0] - KNOTEN[li][0]) * TIEFE / 2.0
    kraefte[2 * li + 1] -= halbe
    kraefte[2 * re + 1] -= halbe
lager = {}
for i in range(len(KNOTEN)):
    if KNOTEN[i][1] < 1e-9:
        lager[2 * i] = 0.0
        lager[2 * i + 1] = 0.0
u = loese_elastik(matrix, kraefte, lager)
sp = elementspannungen_inhomogen(KNOTEN, DREIECKE, u, e_module, QUERKONTRAKTION,
                                 0.0, [0.0] * len(DREIECKE))
vm = [(a * a - a * b + b * b + 3.0 * c * c) ** 0.5 / 1e6 for (a, b, c) in sp]

tris = [list(d) for d in DREIECKE]
xs = [k[0] for k in KNOTEN]
ys = [k[1] for k in KNOTEN]
e_gpa = [e / 1e9 for e in e_module]
```

```
vx = [xs[i] + 20.0 * u[2 * i] for i in range(len(KNOTEN))]
vy = [ys[i] + 20.0 * u[2 * i + 1] for i in range(len(KNOTEN))]
fig, achsen = plt.subplots(1, 3, figsize=(9.5, 5.2))
tp0 = achsen[0].tripcolor(Triangulation(xs, ys, tris), facecolors=e_gpa,
                          cmap="viridis", edgecolors="0.6", linewidth=0.2)
achsen[0].set_title("E-Landkarte (GPa)")
fig.colorbar(tp0, ax=achsen[0], shrink=0.7)
achsen[1].triplot(xs, ys, tris, color="0.85", linewidth=0.4)
achsen[1].triplot(vx, vy, tris, color="C3", linewidth=0.4)
achsen[1].set_title("verformt (x20)")
tp2 = achsen[2].tripcolor(Triangulation(vx, vy, tris), facecolors=vm,
                          cmap="inferno", edgecolors="face")
achsen[2].set_title("Von Mises (MPa)")
fig.colorbar(tp2, ax=achsen[2], shrink=0.7)
for ax in achsen:
    ax.set_aspect("equal")
plt.tight_layout()
plt.show()
```

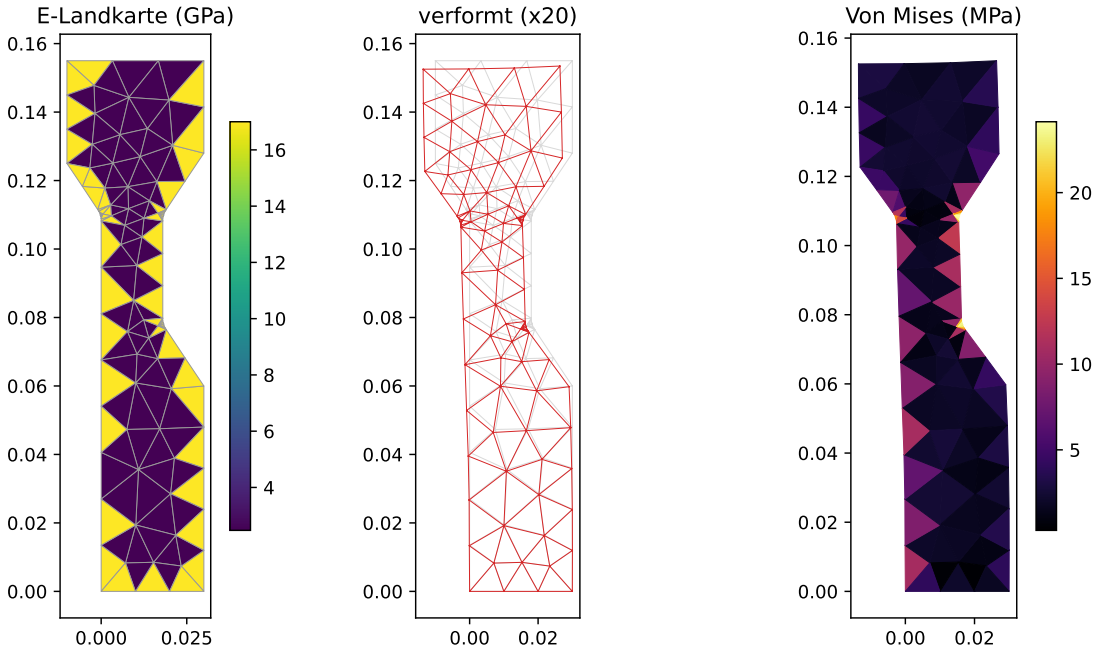

Abbildung 16.5.: Vorgerenderte Fassung der Knochenwerkstatt in den Ausgangseinstellungen (Hüftkraft 2,5·BW, Spongiosa-Dichte 0,5 g/cm³): E-Landkarte, überhöht verformtes Netz und Von-Mises-Feld auf demselben groben 95-Knoten-Netz wie das interaktive Stück. Im Browser wird dieses Bild durch dein eigenes Ergebnis ersetzt.

Die Antwort auf den Vorhersage-Punkt: Dichtere Spongiosa ist **steifer**, also senkt sich der Kopf **weniger**. Weil das Modell linear ist, verdoppelt eine doppelte Hüftkraft dagegen jede Verschiebung und jede Spannung exakt — das kannst du direkt am Kraftfaktor ablesen.

16.7. Ausblick: Warum Knochen sich umbauen

Wir haben aus einer *festen* Dichtekarte eine E-Modul-Karte gemacht und den Knochen einmal durchgerechnet. Aber Knochen ist lebendig: Über Wochen und Monate baut er dort Material auf, wo er stark belastet wird, und dort ab, wo er es kaum ist. Diese Beobachtung trägt einen Namen — das **Wolffsche Gesetz** — und einen modernen biologischen Rahmen, Frosts **Mechanostat** (Frost 2003).

Moderne FE-Remodelingmodelle gießen das in eine Rückkopplung: Sie vergleichen einen mechanischen Reiz (etwa die Formänderungsenergiedichte)

mit einem Zielwert — oberhalb wird Material aufgebaut, unterhalb abgebaut, in einer Totzone dazwischen bleibt alles, wie es ist (Huiskes u. a. 1987). Das ist genau die Form einer **Rückkopplung**, deren Tür 3 aufgestoßen hat: Diesmal reagiert nicht die Temperatur auf die Wärme, sondern das *Material* auf die Rechnung.

$$\text{Dichte} \;\to\; E\text{-Modul} \;\to\; \text{FEM-Rechnung} \;\to\; \text{Reiz} \;\to\; \text{neue Dichte} \;\to\; \dots$$

Anders als der Einweg-Pfeil des Trägers schließt sich dieser Kreis:

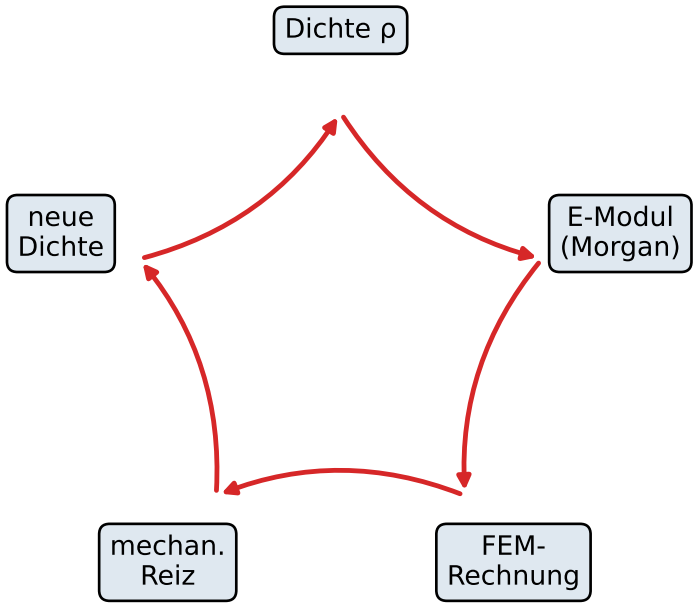

Abbildung 16.6.: Die Remodeling-Rückkopplung als geschlossener Kreis: Aus der Dichte wird über Morgan ein E-Modul, damit rechnet die FEM Verschiebungen und Spannungen, daraus ein mechanischer Reiz, und der verändert die Dichte für die nächste Runde. Genau dieser geschlossene Kreis unterscheidet das Remodeling von der Einweg-Kopplung des Trägers — gerechnet wird er erst im folgenden Kapitel.

In diesem Kapitel rechnen wir das **nicht**. Eine Rückkopplung will sorgfältig gezähmt werden: Ohne Begrenzung, Totzone und räumliche Mittelung erzeugt eine naive Update-Regel bunte Schachbrettmuster, die wie Trabekel *aussehen*, aber nur numerische Artefakte sind. Wie man die Schleife baut und ihr misstraut, ist die Aufgabe des folgenden Kapitels über das Remodeling.

Merkkasten

- Ein tragender Knochen ist für kleine, kurze Lasten **dasselbe FEM-Problem** wie ein Träger — dieselbe Maschine (`elastik2d.py`), rein mechanisch.
- Die Hüftkontaktkraft (~2,5·BW ≈ 1,8 kN) ist eine **Gelenkresultierende** und enthält die Muskelwirkung bereits — kein zweiter Muskel-Lastfall.
- Das Material ist eine **Landkarte**: kortikale Schale 17 GPa, spongiöser Kern aus dem Dichtefeld über Morgan ($E = 6{,}95\,\rho^{1{,}49}$, nur 0,2–1,0 g/cm³).
- **Zwei** Funktionen brauchen das lokale E: Assemblieren *und* Spannungslesen.
- Von Mises ist hier ein **Farbwert**, kein Versagenskriterium; scharfe Ecken erzeugen Kerbsingularitäten — verrunden, dann konvergiert die Halsspannung.

Roter Faden

Zurück: die 2D-Elastik-Kette $u \to \varepsilon \to \sigma$ (Kapitel 14), Auflager und Starrkörpermoden (Kapitel 8, Kapitel 14), die Knotenkraft (Kapitel 13), die Fachwerk-Analogie (Kapitel 3), die Prüfrituale (Kapitel 12) und die Rückkopplungs-Tür (3). — Vor: das folgende Kapitel schließt die Remodeling-Schleife; die Übertragen-Übung unten weist über den Knochen hinaus in Zahnmedizin und Botanik.

Übungen

Ü 16.1 (Verstehen). Im E-Landkarten-Bild (Abbildung 16.2) sind zwei Zonen zu sehen. Welche ist die kortikale Schale, welche der spongiöse Kern? Und warum müssen E-Feld, Dehnungsfeld und Spannungsfeld *nicht* dasselbe Bild zeigen?

Musterlösung zu Ü 16.1

Die helle, gleichmäßig steife Randzone (17 GPa) ist die Kompakta; der dunklere, fleckige Innenbereich ist die Spongiosa (0,6–7 GPa je nach Dichte). Die drei Felder unterscheiden sich, weil sie verschiedene Dinge zeigen: E ist eine reine Materialeigenschaft (wo ist der Knochen

steif?), die Dehnung misst, wie stark sich das Material *verformt*, und die Spannung ist deren Produkt mit E — eine steife Zone kann bei *kleiner* Dehnung *hohe* Spannung tragen. Deshalb sitzt die höchste Spannung nicht dort, wo E am größten ist, sondern wo Last und Steifigkeit zusammentreffen: am schmalen Schenkelhals und in der Schale.

Ü 16.2 (Verändern). Verdopple in der Knochenwerkstatt den Kraftfaktor von 2 auf 4. Verdoppelt sich die Kopfsenkung? Stelle danach die Spongiosa-Dichte von 0,3 auf 0,9 g/cm³ und vergleiche.

i Musterlösung zu Ü 16.2

Ja — die Kopfsenkung verdoppelt sich exakt, denn das Modell ist **linear**: Alle Verschiebungen und Spannungen wachsen proportional zur Last. Eine dichtere Spongiosa (0,9 statt 0,3 g/cm³) hebt deren E-Modul von rund 1,2 auf gut 5,9 GPa; der Kern wird steifer, die Kopfsenkung **kleiner**. Nichtlinear wäre die Dichte nur, wenn sie über Morgans Potenz $\rho^{1{,}49}$ ins E einginge — was sie tut, aber die *Rechnung* selbst bleibt bei fester Landkarte linear.

Ü 16.3 (Übertragen). Suche dir ein zweites tragendes Bauteil aus der Biologie — ein Zahnimplantat im Kieferknochen **oder** einen pflanzlichen Stängel als Rohr-Träger. Benenne (ohne zu rechnen): Was ist das Netz, was die Last, was das Auflager, und wo säße das ortsabhängige Material?

i Musterlösung zu Ü 16.3

Zahnimplantat: Das Netz ist ein Schnitt durch Implantat und umgebenden Kieferknochen; die Last ist die Kaukraft auf die Krone; das Auflager ist der Rand des mitmodellierten Knochenblocks. Ortsabhängiges Material: hartes Titan im Implantat, kortikale Schicht und weiche Spongiosa im Kiefer — dieselbe Landkarten-Idee. *Pflanzenstängel:* Netz ist ein Längs- oder Querschnitt; Last ist Wind oder Eigengewicht der Blüte; Auflager ist der Wurzelansatz. Das Material ist ortsabhängig zwischen fester Randfaser (Sklerenchym) und weichem Mark — ein Rohr-Träger wie das I-Profil aus Kapitel 1.

Das Kleingedruckte

Die Grenzen dieses Modells, offen benannt: Es ist **2D**, nicht 3D — ein echter Femur trägt räumlich. Es ist **isotrop**, obwohl Knochen faserig und rich-

tungsabhängig ist. Es ist eine **statische Ersatzrechnung** eines einzigen Lastmoments, kein Wachstum, keine Heilung, kein Umbau. Von Mises ist ein **Darstellungswert**, kein Bruchkriterium; wir treffen **keine klinische Aussage** über einen realen Knochen. Und die Geometrie ist von Hand gezeichnet, **nicht** aus einem CT importiert. Jede dieser Grenzen ist eine bewusste Vereinfachung — genau die Ehrlichkeit, die Kapitel 12 verlangt. Was das Modell dennoch zeigt, ist echt: dass dieselbe Methode, die den Träger im Feuer durchrechnet, ohne eine neue Idee auch einen Knochen trägt.

17. Warum Knochen sich umbauen: FEM mit Rückkopplung

Kapitel 16 gab jedem Dreieck eine feste Dichte und daraus einen festen E-Modul. Die FEM antwortete mit Verschiebungen und Spannungen, aber das Material hörte nicht zu. Jetzt schließen wir den Kreis:

$$\rho \longrightarrow E(\rho) \longrightarrow \text{FEM} \longrightarrow \text{mechanischer Reiz} \longrightarrow \rho_{\text{neu}}.$$

Das ist eine neue Art von Rechnung. Nicht der Löser ist neu, sondern die **Rückkopplung**: Das Ergebnis eines statischen FEM-Laufs verändert die Materialkarte für den nächsten Lauf. Wir untersuchen damit ein rechteckiges, spongiöses Ersatzfenster. Es entstehen dichtere und weniger dichte Bereiche, aber weder Löcher noch einzelne Knochenbälkchen.

Naheliegende Vermutung

Vermutung: *„Wo die Spannung groß ist, wächst Knochen. Man wiederholt das einfach, bis ein Trabekelmuster entsteht.“*
Was stattdessen stimmt: Eine rein lokale Regel kann benachbarte Elemente abwechselnd steif und weich machen. Das Schachbrett ist ein Netzartefakt, kein Trabekelmuster. Außerdem enthält unser Modell weder Zellen noch Stoffwechsel oder reale Umbauzeiten. Es ist ein prüfbares mechanisches Ersatzmodell für Dichteanpassung.

Lernziele

Nach diesem Kapitel kannst du …

1. … die Remodeling-Schleife als Folge statischer FEM-Rechnungen erklären,
2. … aus Spannung und Dehnung die Formänderungsenergiedichte und daraus einen Reiz je Knochenmasse berechnen,
3. … Abbau, Totzone und Aufbau in einer begrenzten Dichte-Regel unterscheiden,

4. … erklären, warum ein lokales Update Schachbrettmuster erzeugen kann und wie eine nichtlokale Mittelung dagegen wirkt,
5. … Stationarität, Homöostase und fehlende Konvergenz auseinanderhalten,
6. … die Grenzen eines isotropen Dichtekontinuums ehrlich benennen.

17.1. Viele statische Rechnungen — keine biologische Uhr

Das Ersatzfenster ist 60 mm lang, 30 mm hoch und 10 mm tief. Links halten wir alle Knoten fest. Rechts wirken zwei getrennte, kraftgesteuerte Lastfälle:

Lastfall	Resultierende	Gewicht
Biegung	600 N nach unten	0,65
Zug	600 N nach rechts	0,35

Die Gewichte addieren sich zu eins. Sie sind eine offengelegte Mischung zweier didaktischer Lastpfade, keine gemessenen Häufigkeiten von Gehen oder Treppensteigen. In jeder Recheniteration lösen wir beide Lastfälle mit denselben Kräften und Lagern. Danach erst ändern wir die Dichten synchron.

> **Iteration ist nicht Zeit**
>
> Iteration 40 bedeutet weder Tag 40 noch Monat 40. Die Schrittweite steuert eine numerische Fixpunktiteration. Ohne kalibriertes biologisches Zeitgesetz darf ihre Nummer nicht in Lebenszeit übersetzt werden.

```
# Der Modellvertrag in einer kleinen, sichtbaren Tabelle.
parameter = [
    ("Geometrie", "60 x 30 x 10 mm"),
    ("Lehrnetz", "45 Knoten, 64 P1-Dreiecke"),
    ("Dichtebereich", "0,2 bis 1,0 g/cm3"),
    ("Startdichte", "0,5 g/cm3"),
    ("Lastgewichte", "0,65 / 0,35"),
]
for name, wert in parameter:
    print("%-16s %s" % (name, wert))
```

```
Geometrie        60 x 30 x 10 mm
Lehrnetz         45 Knoten, 64 P1-Dreiecke
Dichtebereich    0,2 bis 1,0 g/cm3
Startdichte      0,5 g/cm3
Lastgewichte     0,65 / 0,35
```

17.2. Ein Dreieck speichert Verformungsenergie

Drückst du eine Feder langsam zusammen, steckt anschließend Energie in ihrer Verformung. Ein Dreieck speichert ebenso elastische Energie. Pro Volumen heißt sie **Formänderungsenergiedichte** U. Mit der Engineering-Voigt-Schreibweise

$$\varepsilon = [\varepsilon_{xx}, \varepsilon_{yy}, \gamma_{xy}]^T, \qquad \sigma = [\sigma_{xx}, \sigma_{yy}, \tau_{xy}]^T$$

gilt

$$U = \frac{1}{2}\, \sigma^T \varepsilon .$$

Der Schubterm ist damit $\tau_{xy}\gamma_{xy}$. Das ist wichtig: γ_{xy} ist die **Engineering-Schubdehnung**, nicht die tensorielle Komponente $\varepsilon_{xy} = \gamma_{xy}/2$.

Nehmen wir für ein Dreieck $\sigma = [2{,}0;\ 1{,}0;\ 0{,}5]$ Pa und $\varepsilon = [2{,}0;\ 1{,}0;\ 1{,}0]$. Dann ist

$$U = \tfrac{1}{2}(2 \cdot 2 + 1 \cdot 1 + 0{,}5 \cdot 1) = 2{,}75\ \mathrm{J/m^3}.$$

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "gemeinsam"))
from remodeling import formenergie_dreieck

spannung = [2.0, 1.0, 0.5]
dehnung = [2.0, 1.0, 1.0]
u = formenergie_dreieck(spannung, dehnung)
print("U =", u, "J/m3")
```

```
U = 2.75 J/m3
```

17.3. Energie je Masse wird zum Reiz

Dieselbe Energie pro Volumen bedeutet bei wenig vorhandener Knochenmasse mehr Energie je Masse. Für Dreieck i und Lastfall l bilden wir deshalb

$$S_i = \frac{\sum_l p_l U_{i,l}}{\rho_{\mathrm{m},i}}, \qquad \rho_{\mathrm{m},i} = 1000\,\rho_i.$$

Im Morgan-Gesetz steht ρ_i in g/cm³; nach Multiplikation mit 1000 steht $\rho_{\mathrm{m},i}$ in kg/m³. Weil U die Einheit J/m³ besitzt, hat S die Einheit J/kg. Energie je Knochenmasse wird in adaptiven Knochenmodellen als mechanische Rückkopplungsgröße verwendet (Huiskes u. a. 1987; Weinans u. a. 1992). Unsere lineare Lastfallmischung ist eine eigene didaktische Modellentscheidung.

Beispiel: $U = 1{,}0 \cdot 10^4$ J/m³ und $\rho = 0{,}5$ g/cm³ ergeben $\rho_{\mathrm{m}} = 500$ kg/m³ und $S = 20$ J/kg.

```
energie = 1.0e4       # J/m3
dichte = 0.5          # g/cm3
massendichte = 1000.0 * dichte
reiz = energie / massendichte
print("Massendichte =", massendichte, "kg/m3")
print("Reiz         =", reiz, "J/kg")
```

```
Massendichte = 500.0 kg/m3
Reiz         = 20.0 J/kg
```

Für den Zielreiz brauchen wir einen festen Maßstab. Wir rechnen einmal einen gleichmäßigen Referenzzustand mit $\rho_{\mathrm{ref}} = 0{,}5$ g/cm³ auf einem 16×8-Netz und mitteln sein Reizfeld flächengewichtet:

$$S_0 = \frac{\sum_i A_i S_i^{\mathrm{ref}}}{\sum_i A_i} = 19{,}4294626265\ \mathrm{J/kg}.$$

S_0 bleibt danach **fest** — auch bei anderer Startdichte, Schrittweite, Netzfeinheit oder Lastmischung. Er ist ein Referenzwert dieses Modells, kein universell gemessener Sollwert für Knochen.

17.4. Der schlechte erste Versuch: jedes Element hört nur sich selbst

Die verlockende Regel lautet: Ist $S_i > S_0$, erhöhe die Dichte; ist $S_i < S_0$, verringere sie. Unter Kraftlast wird ein etwas weicheres Element stärker verformt. Sein Reiz steigt, sein Update kann es wieder versteifen — und zugleich Last in ein Nachbarelement umlenken. Auf einem diskreten Netz kann diese Rückkopplung benachbarte Dreiecke gegeneinander treiben.

Das Ergebnis ist oft ein **Schachbrettmuster**: große Sprünge auf der Skala eines einzelnen Elements. Es kann stationär werden und trotzdem falsch sein. Wie in Kapitel 12 gilt: Ein buntes Bild und ein kleiner Abbruchfehler sind noch keine Netzunabhängigkeit.

17.5. Die Nachbarschaft hört mit

Wir ersetzen den lokalen Reiz durch einen Mittelwert über die Schwerpunkte der Nachbarelemente. Ein nahes Dreieck zählt stark, ein Dreieck am Filterrand gar nicht. Das Gewicht ist ein Zeltdach über dem Abstand:

$$\bar{S}_i = \begin{cases} S_i, & R = 0, \\ \dfrac{\sum_j w_{ij} A_j S_j}{\sum_j w_{ij} A_j}, & R > 0, \end{cases} \qquad w_{ij} = \max\left(0, 1 - \frac{d_{ij}}{R}\right).$$

Die Fläche A_j verhindert, dass viele kleine Dreiecke mehr zählen als wenige große. $R = 0$ ist ausdrücklich der ungefilterte Kontrollfall $\bar{S}_i = S_i$; ein negativer Radius ist ungültig. Für $R > 0$ trägt das Zielelement selbst immer mit Gewicht eins bei.

Der verwendete Schwerpunkt-/Flächenfilter ist eine bewusst einfache Adaption nichtlokaler Mittelung gegen Konvergenz- und Schachbrettprobleme (Calvo-Gallego u. a. 2021), nicht die identische Reproduktion des dortigen Algorithmus. Im Lehrmodell gilt $R = 30$ mm als feste physische Länge. Auf dem 8×4-Netz tragen je nach Randlage 28 bis 59 Dreiecke bei; der Median ist 43,5.

```
from remodeling import baue_filtergewichte, glaette_reize

schwerpunkte = [(0.0, 0.0), (1.0, 0.0), (2.0, 0.0)]
flaechen = [1.0, 2.0, 1.0]
```

```
lokaler_reiz = [10.0, 20.0, 40.0]

filter_aus = baue_filtergewichte(schwerpunkte, flaechen, 0.0)
filter_an = baue_filtergewichte(schwerpunkte, flaechen, 1.5)
print("R = 0  :", glaette_reize(lokaler_reiz, filter_aus))
print("R = 1,5:", glaette_reize(lokaler_reiz, filter_an))
```

```
R = 0  : [10.0, 20.0, 40.0]
R = 1,5: [14.0, 21.25, 32.0]
```

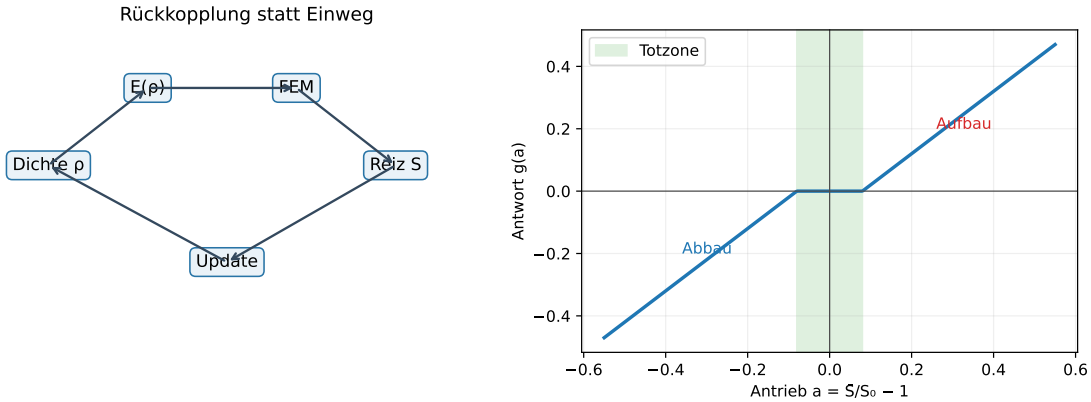

Abbildung 17.1.: Die geschlossene Remodeling-Schleife (links) und die Antwortfunktion (rechts). Innerhalb der Totzone bleibt die Dichte unverändert; außerhalb reagiert sie stetig mit Abbau oder Aufbau.

17.6. Aufbau, Abbau und die Totzone

Frosts Mechanostat liefert einen biologischen Rahmen dafür, dass Knochen auf unterschiedliche mechanische Beanspruchung unterschiedlich reagiert (Frost 2003). Wir übernehmen daraus keine scheinpräzisen biologischen Schwellen. Stattdessen definieren wir den dimensionslosen Antrieb

$$a_i^n = \frac{\bar{S}_i^n}{S_0} - 1$$

und eine transparente numerische Antwort mit Totzonenbreite $\delta = 0{,}08$:

$$g(a) = \begin{cases} a + \delta, & a < -\delta, \\ 0, & |a| \le \delta, \\ a - \delta, & a > \delta. \end{cases}$$

Mit $\eta = 0{,}08$ g/cm³ je Recheniteration und $\Delta\rho_{\max} = 0{,}04$ g/cm³ je Recheniteration folgt

$$\Delta\rho_i^{n} = \operatorname{clip}\left(\eta g(a_i^n), -\Delta\rho_{\max}, \Delta\rho_{\max}\right),$$

$$\rho_i^{n+1} = \operatorname{clip}\left(\rho_i^n + \Delta\rho_i^n, 0{,}2, 1{,}0\right).$$

`clip` bedeutet nur: Werte unter der Untergrenze werden auf die Untergrenze, Werte über der Obergrenze auf die Obergrenze gesetzt. Alle neuen Dichten werden aus derselben unveränderten Kopie von ρ^n berechnet. Ein Update mitten in der Elementschleife wäre ein anderes, von der Nummerierung abhängiges Verfahren.

```
from remodeling import aktualisiere_dichten

dichten = [0.50, 0.50, 0.50]
reize = [0.80 * 19.4294626265,
         1.04 * 19.4294626265,
         1.30 * 19.4294626265]
neu, antriebe, aenderungen = aktualisiere_dichten(
    dichten, reize, 19.4294626265, 0.08, 0.08, 0.04, 0.2, 1.0)
for i in range(3):
    print("a=%5.2f  delta_rho=%6.3f  rho_neu=%5.3f"
          % (antriebe[i], aenderungen[i], neu[i]))
```

```
a=-0.20  delta_rho=-0.010  rho_neu=0.490
a= 0.04  delta_rho= 0.000  rho_neu=0.500
a= 0.30  delta_rho= 0.018  rho_neu=0.518
```

17.7. Die vollständige Schleife

Eine Iteration besteht nun aus sechs klaren Schritten:

1. Aus jeder Dichte mit dem Morgan-Gesetz den E-Modul bilden.
2. Beide festen Lastfälle lösen.

3. Je Dreieck U und den gewichteten lokalen Reiz S berechnen.
4. S über die feste physische Nachbarschaft zu $\bar{S}$ mitteln.
5. Alle Dichten synchron und begrenzt aktualisieren.
6. Änderung, Zwei-Schritt-Abstand, Gleichgewicht und Diagnosen prüfen.

Das eigenständige Kapitelprogramm führt genau diese Schleife aus:

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap17"))
import knochen_remodeling as kr

ergebnis = kr.simuliere()
lokal = kr.simuliere(filter_radius=0.0)
diag = ergebnis["diagnosen"]
print("Status             :", ergebnis["status"])
print("Iterationen        :", ergebnis["iterationen"])
print("Dichtemittel       : %.5f g/cm3" % diag["dichtemittel"])
print("Masse              : %.6f kg" % diag["masse"])
print("Nachgiebigkeit     : %.6e J" % diag["nachgiebigkeit"])
print("Grenz-/Aussenanteil: %.1f %% / %.1f %%"
      % (100.0 * diag["grenzanteil"], 100.0 * diag["aussenanteil"]))
print("q_hf lokal/gefiltert: %.5f / %.5f" % (lokal["q_hf"], ergebnis["q_hf"]))
```

```
Status             : stationaer
Iterationen        : 88
Dichtemittel       : 0.42971 g/cm3
Masse              : 0.007735 kg
Nachgiebigkeit     : 3.164353e-01 J
Grenz-/Aussenanteil: 0.0 % / 3.1 %
q_hf lokal/gefiltert: 0.36135 / 0.02073
```

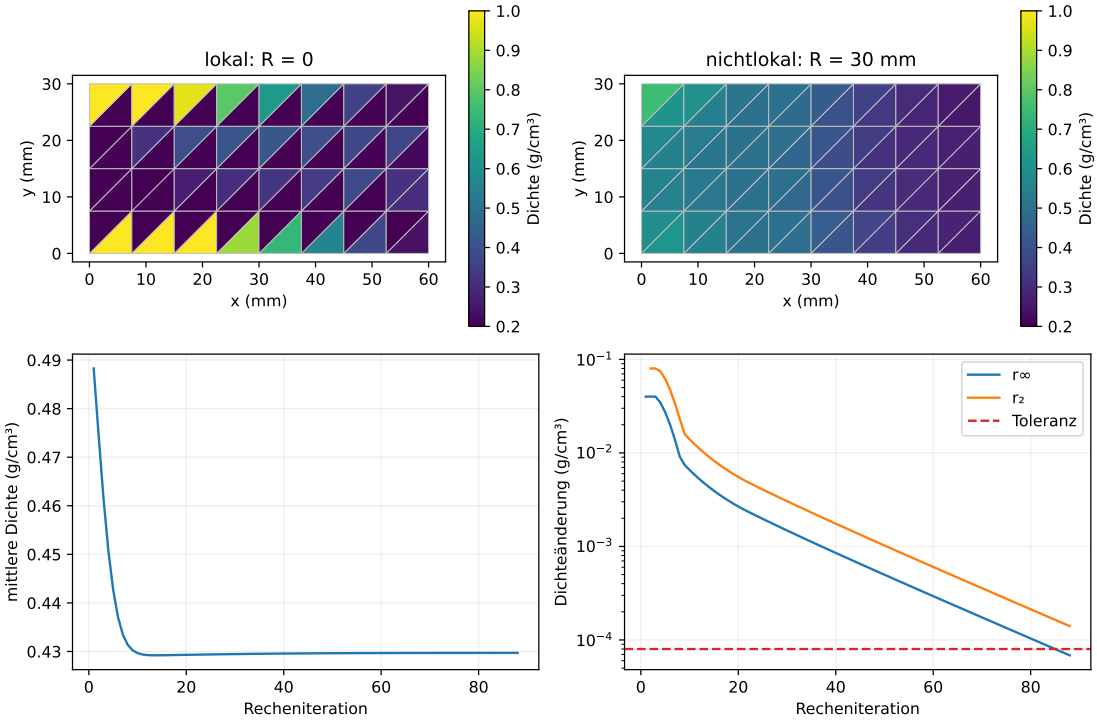

Abbildung 17.2.: Das lokale Modell (oben links) wird zwar stationär, zeigt aber ein elementgroßes Schachbrett. Der feste nichtlokale Filter (oben rechts) erhält den großskaligen Lastpfad und senkt den Hochfrequenzindikator um 94,3 %. Unten: Dichtemittel sowie Ein- und Zwei-Schritt-Änderung des gefilterten Laufs.

Das lokale Feld wirkt wie ein Dreiecksmosaik. Der Hochfrequenzindikator

$$q_{\mathrm{hf}} = \sqrt{\frac{\sum_{i} A_i \left(\rho_i - \langle \rho \rangle_{\mathcal{N}_i}\right)^2}{\sum_{i} A_i}}$$

misst die Abweichung vom Mittel der Kantennachbarn. Er fällt von 0,36135 auf 0,02073 g/cm³ — um 94,3 %. Ein kleinerer Kantensprung allein wäre kein Beweis, denn auch ein sinnvoller glatter Gradient besitzt Kantensprünge.

Im HTML lässt sich die Folge vor- und zurückspulen. Die feste Farbskala macht sichtbar, dass zuerst die Materialmenge abnimmt und sich danach der glatte räumliche Verlauf einpendelt. Für den Druck zeigt die nächste Abbildung sechs beschriftete Zustände derselben Rechnung.

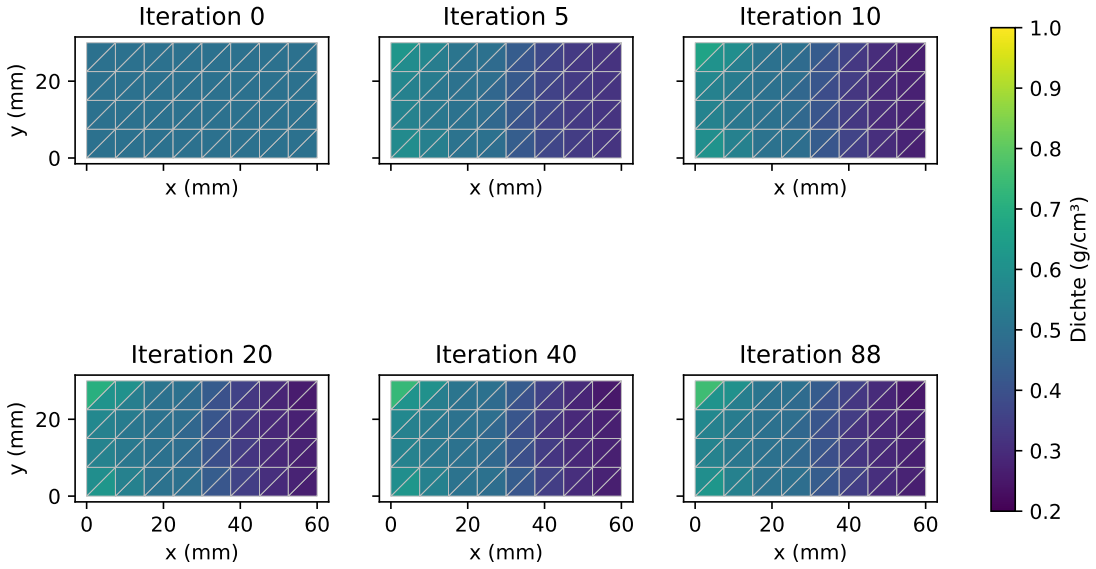

Abbildung 17.3.: Sechs Zustände derselben Recheniteration mit fester Farbskala. Die Startkarte ist gleichmäßig; danach bildet sich ein glatter großskaliger Lastpfad. Die Nummer ist eine Recheniteration, keine Zeitangabe.

17.8. Wann ist die Schleife fertig?

Wir messen nach jedem Update

$$r_\infty^n = \max_i |\rho_i^{n+1} - \rho_i^n|.$$

Der Lauf heißt **stationär**, wenn $r_\infty < 8 \cdot 10^{-5}$ g/cm³ in drei aufeinanderfolgenden Iterationen gilt. Nach spätestens 200 Iterationen endet er sonst als „nicht konvergiert“. Zusätzlich beobachten wir $r_2^n = \max_i |\rho_i^{n+1} - \rho_i^{n-1}|$: Wird nur dieser Wert klein, pendelt das Verfahren zwischen zwei Feldern und heißt „oszillierend“.

Stationär bedeutet aber nicht automatisch **homöostatisch**. Ein Element kann an einer Dichtegrenze festhängen, obwohl sein Reiz außerhalb der Totzone liegt. Deshalb geben wir getrennt den Flächenanteil an Dichtegrenzen und den Anteil außerhalb der Totzone aus. Nur stationär und ohne äußeren Anteil nennen wir in diesem didaktischen Modell homöostatisch. Der Referenzlauf ist stationär nach 88 Iterationen, aber mit 3,1 % außerhalb der Totzone nicht vollständig homöostatisch.

Weitere Kontrollgrößen sind die Masse $m = t\sum_i A_i \rho_{\mathrm{m},i}$, die gewichtete Nachgiebigkeit $C = \sum_l p_l \mathbf{f}_l^T \mathbf{u}_l$ sowie Kraft- und Momentengleichgewicht. Beim Referenzlauf beträgt die Masse 0,007735 kg und $C = 0{,}3164353$ J.

17.9. Gegen eine unabhängige Rechnung geprüft

Der reine Python-Code wurde nicht nur gegen sich selbst getestet. Eine unabhängige NumPy/scikit-fem-Implementierung rechnet dieselben Netze und Lasten. Die Freigabe umfasst harte Assertions; ein Fehler beendet das Validierungsprogramm mit einem Fehlercode.

Prüfung	Ergebnis
Verschiebung, Spannung, Dehnung, Reaktion	relative Fehler < $1{,}1 \cdot 10^{-13}$
Reizfeld	relativer Fehler $2{,}0 \cdot 10^{-13}$
vollständiger synchroner Update-Schritt	relativer Fehler $2{,}7 \cdot 10^{-14}$
Kraft-/Momentengleichgewicht	relative Fehler < $4{,}5 \cdot 10^{-14}$
12×6 gegen 16×8, Enddichtefeld	3,28 %
zweite Diagonalenorientierung	3,12 %
halbe Schrittweite, Enddichtefeld	0,51 %
Startdichten 0,4 / 0,55 g/cm³	3,43 % / 3,87 %
q_{hf}, lokal → nichtlokal	0,36135 → 0,02073

Alle festgelegten Grenzen werden eingehalten. Die Details und das maschinenlesbare Ergebnis liegen in `validierung/kap17_remodeling/`. Besonders wichtig: Drei Netze werden stationär, und zwischen den zwei feinsten Netzen bleibt nicht nur ein Mittelwert, sondern das auf ein gemeinsames Raster projizierte **Feld** innerhalb von 5 %.

17.10. Probier es aus: die Remodeling-Werkstatt

Die folgende Browserrechnung benutzt dasselbe 45-Knoten-Netz, dieselben zwei Lastfälle und dasselbe feste S_0. Verändere Filterradius, Totzone oder das Gewicht des Biegelastfalls. Es werden bewusst nur 15 Schritte gezeigt. Beim ersten Start muss der Browser Pyodide und Matplotlib laden; auf der

Referenzhardware dauerte dieser Kaltstart 55,7 s. Danach benötigten fünf Wiederholungen im Median 1,41 s (maximal 1,56 s).

> **Vorhersage-Punkt**
>
> Setze zuerst `FILTER_RADIUS = 0.0`. Erwartest du ein glatteres oder ein stärker wechselndes Dichtefeld? Begründe deine Vorhersage, bevor du ausführst.

```
#| autorun: true
# Die Remodeling-Werkstatt -- Werte aendern, dann Ausfuehren druecken.
FILTER_RADIUS = 0.030       # m; 0.0 schaltet den Filter aus
TOTZONE = 0.08              # dimensionslose Halbbreite
LASTFALL_1_ANTEIL = 0.65    # Biegung; Zug erhaelt den Rest

BREITE = 0.060
HOEHE = 0.030
TIEFE = 0.010
NU = 0.30
RHO_MIN = 0.20
RHO_MAX = 1.00
S0 = 19.4294626265          # J/kg, bleibt bei allen Varianten fest
ETA = 0.080
MAX_DELTA = 0.040
SCHRITTE = 15

if FILTER_RADIUS < 0.0:
    raise ValueError("FILTER_RADIUS darf nicht negativ sein")
if TOTZONE < 0.0:
    raise ValueError("TOTZONE darf nicht negativ sein")
if LASTFALL_1_ANTEIL < 0.0 or LASTFALL_1_ANTEIL > 1.0:
    raise ValueError("LASTFALL_1_ANTEIL muss zwischen 0 und 1 liegen")
GEWICHTE = [LASTFALL_1_ANTEIL, 1.0 - LASTFALL_1_ANTEIL]

# --- Netz und Material (Quellen: kap17/knochen_remodeling.py,
#     gemeinsam/elastik2d.py und gemeinsam/dichtefeld.py) ---
def rechtecknetz(nx, ny):
    knoten = []
    for j in range(ny + 1):
        for i in range(nx + 1):
            knoten.append((BREITE * i / nx, HOEHE * j / ny))
    dreiecke = []
    zeile = nx + 1
    for j in range(ny):
```

```
        for i in range(nx):
            lu = j * zeile + i
            ru = lu + 1
            lo = lu + zeile
            ro = lo + 1
            dreiecke.append((lu, ru, ro))
            dreiecke.append((lu, ro, lo))
    return knoten, dreiecke

def zeltsteigungen(ecken):
    xa, ya = ecken[0]
    xb, yb = ecken[1]
    xc, yc = ecken[2]
    zwei_a = xa * (yb - yc) + xb * (yc - ya) + xc * (ya - yb)
    b = [(yb - yc) / zwei_a, (yc - ya) / zwei_a,
         (ya - yb) / zwei_a]
    c = [(xc - xb) / zwei_a, (xa - xc) / zwei_a,
         (xb - xa) / zwei_a]
    return b, c, abs(zwei_a) / 2.0

def b_tabelle(ecken):
    b, c, flaeche = zeltsteigungen(ecken)
    tabelle = []
    for zeile in range(3):
        neu = []
        for spalte in range(6):
            neu.append(0.0)
        tabelle.append(neu)
    for ecke in range(3):
        tabelle[0][2 * ecke] = b[ecke]
        tabelle[1][2 * ecke + 1] = c[ecke]
        tabelle[2][2 * ecke] = c[ecke]
        tabelle[2][2 * ecke + 1] = b[ecke]
    return tabelle, flaeche

def materialtabelle(e_modul):
    faktor = e_modul / (1.0 - NU * NU)
    return [[faktor, faktor * NU, 0.0],
            [faktor * NU, faktor, 0.0],
            [0.0, 0.0, faktor * (1.0 - NU) / 2.0]]

def morgan_e(dichte):
    return 6.95e9 * dichte ** 1.49
```

```
def elementsteifigkeit(ecken, e_modul):
    b, flaeche = b_tabelle(ecken)
    d = materialtabelle(e_modul)
    db = []
    for i in range(3):
        zeile = []
        for j in range(6):
            summe = 0.0
            for k in range(3):
                summe = summe + d[i][k] * b[k][j]
            zeile.append(summe)
        db.append(zeile)
    steck = []
    for i in range(6):
        zeile = []
        for j in range(6):
            summe = 0.0
            for k in range(3):
                summe = summe + b[k][i] * db[k][j]
            zeile.append(summe * flaeche * TIEFE)
        steck.append(zeile)
    return steck

def assembliere(knoten, dreiecke, dichten):
    n = 2 * len(knoten)
    matrix = []
    for i in range(n):
        zeile = []
        for j in range(n):
            zeile.append(0.0)
        matrix.append(zeile)
    for nummer in range(len(dreiecke)):
        dreieck = dreiecke[nummer]
        ecken = []
        dofs = []
        for knotenindex in dreieck:
            ecken.append(knoten[knotenindex])
            dofs.append(2 * knotenindex)
            dofs.append(2 * knotenindex + 1)
        steck = elementsteifigkeit(ecken, morgan_e(dichten[nummer]))
        for i in range(6):
            for j in range(6):
                matrix[dofs[i]][dofs[j]] = matrix[dofs[i]][dofs[j]] + steck[i][j]
```

```
    return matrix

def gauss_loesen(matrix, rechte_seite):
    n = len(rechte_seite)
    a = []
    b = list(rechte_seite)
    for zeile in matrix:
        a.append(list(zeile))
    for spalte in range(n):
        pivot = spalte
        for zeile in range(spalte + 1, n):
            if abs(a[zeile][spalte]) > abs(a[pivot][spalte]):
                pivot = zeile
        if abs(a[pivot][spalte]) < 1.0e-14:
            raise ArithmeticError("singulaere Steifigkeitsmatrix")
        a[spalte], a[pivot] = a[pivot], a[spalte]
        b[spalte], b[pivot] = b[pivot], b[spalte]
        for zeile in range(spalte + 1, n):
            faktor = a[zeile][spalte] / a[spalte][spalte]
            for k in range(spalte, n):
                a[zeile][k] = a[zeile][k] - faktor * a[spalte][k]
            b[zeile] = b[zeile] - faktor * b[spalte]
    x = []
    for i in range(n):
        x.append(0.0)
    for zeile in range(n - 1, -1, -1):
        rest = b[zeile]
        for k in range(zeile + 1, n):
            rest = rest - a[zeile][k] * x[k]
        x[zeile] = rest / a[zeile][zeile]
    return x

def loese_mit_lager(matrix, kraefte, freie_dofs):
    reduziert = []
    rechts = []
    for i in freie_dofs:
        zeile = []
        for j in freie_dofs:
            zeile.append(matrix[i][j])
        reduziert.append(zeile)
        rechts.append(kraefte[i])
    frei_loesung = gauss_loesen(reduziert, rechts)
    u = []
```

```
    for i in range(len(kraefte)):
        u.append(0.0)
    for i in range(len(freie_dofs)):
        u[freie_dofs[i]] = frei_loesung[i]
    return u

def rechter_rand_last(knoten, gesamt_x, gesamt_y):
    paare = []
    for i in range(len(knoten)):
        if abs(knoten[i][0] - BREITE) < 1.0e-12:
            paare.append((knoten[i][1], i))
    paare.sort()
    kraefte = []
    for i in range(2 * len(knoten)):
        kraefte.append(0.0)
    for stelle in range(len(paare) - 1):
        unten = paare[stelle][1]
        oben = paare[stelle + 1][1]
        anteil = (knoten[oben][1] - knoten[unten][1]) / HOEHE
        kraefte[2 * unten] = kraefte[2 * unten] + 0.5 * anteil * gesamt_x
        kraefte[2 * oben] = kraefte[2 * oben] + 0.5 * anteil * gesamt_x
        kraefte[2 * unten + 1] = kraefte[2 * unten + 1] + 0.5 * anteil * gesamt_y
        kraefte[2 * oben + 1] = kraefte[2 * oben + 1] + 0.5 * anteil * gesamt_y
    return kraefte

def dehnung_spannung(ecken, eckwerte, e_modul):
    b, flaeche = b_tabelle(ecken)
    dehnung = []
    for i in range(3):
        summe = 0.0
        for j in range(6):
            summe = summe + b[i][j] * eckwerte[j]
        dehnung.append(summe)
    d = materialtabelle(e_modul)
    spannung = []
    for i in range(3):
        summe = 0.0
        for j in range(3):
            summe = summe + d[i][j] * dehnung[j]
        spannung.append(summe)
    return dehnung, spannung

def lokale_reize(knoten, dreiecke, dichten, lastfaelle, gewichte, freie_dofs):
```

```
    matrix = assembliere(knoten, dreiecke, dichten)
    energien = []
    for lastfall in range(len(lastfaelle)):
        u = loese_mit_lager(matrix, lastfaelle[lastfall], freie_dofs)
        feld = []
        for nummer in range(len(dreiecke)):
            dreieck = dreiecke[nummer]
            ecken = []
            eckwerte = []
            for index in dreieck:
                ecken.append(knoten[index])
                eckwerte.append(u[2 * index])
                eckwerte.append(u[2 * index + 1])
            dehnung, spannung = dehnung_spannung(
                ecken, eckwerte, morgan_e(dichten[nummer]))
            energie = 0.0
            for i in range(3):
                energie = energie + spannung[i] * dehnung[i]
            feld.append(0.5 * energie)
        energien.append(feld)
    reize = []
    for nummer in range(len(dreiecke)):
        energie = 0.0
        for lastfall in range(len(lastfaelle)):
            energie = energie + gewichte[lastfall] * energien[lastfall][nummer]
        reize.append(energie / (1000.0 * dichten[nummer]))
    return reize

def geometrie(knoten, dreiecke):
    flaechen = []
    schwerpunkte = []
    for dreieck in dreiecke:
        ecken = []
        for index in dreieck:
            ecken.append(knoten[index])
        b, c, flaeche = zeltsteigungen(ecken)
        flaechen.append(flaeche)
        schwerpunkte.append(((ecken[0][0] + ecken[1][0] + ecken[2][0]) / 3.0,
                             (ecken[0][1] + ecken[1][1] + ecken[2][1]) / 3.0))
    return flaechen, schwerpunkte

def filter_bauen(schwerpunkte, flaechen, radius):
    zeilen = []
```

```
    if radius == 0.0:
        for i in range(len(schwerpunkte)):
            zeilen.append([(i, 1.0)])
        return zeilen
    for i in range(len(schwerpunkte)):
        roh = []
        summe = 0.0
        for j in range(len(schwerpunkte)):
            dx = schwerpunkte[j][0] - schwerpunkte[i][0]
            dy = schwerpunkte[j][1] - schwerpunkte[i][1]
            abstand = (dx * dx + dy * dy) ** 0.5
            zelt = 1.0 - abstand / radius
            if zelt > 0.0:
                gewicht = zelt * flaechen[j]
                roh.append((j, gewicht))
                summe = summe + gewicht
        zeile = []
        for index, gewicht in roh:
            zeile.append((index, gewicht / summe))
        zeilen.append(zeile)
    return zeilen

def filtern(reize, filterzeilen):
    ergebnis = []
    for zeile in filterzeilen:
        summe = 0.0
        for index, gewicht in zeile:
            summe = summe + gewicht * reize[index]
        ergebnis.append(summe)
    return ergebnis

def antwort(antrieb):
    if antrieb < -TOTZONE:
        return antrieb + TOTZONE
    if antrieb > TOTZONE:
        return antrieb - TOTZONE
    return 0.0

def begrenzen(wert, unten, oben):
    if wert < unten:
        return unten
    if wert > oben:
        return oben
```

```
    return wert

# --- 15 synchrone Schritte ---
KNOTEN, DREIECKE = rechtecknetz(8, 4)
FLAECHEN, SCHWERPUNKTE = geometrie(KNOTEN, DREIECKE)
LASTFAELLE = [rechter_rand_last(KNOTEN, 0.0, -600.0),
              rechter_rand_last(KNOTEN, 600.0, 0.0)]
FREI = []
for i in range(len(KNOTEN)):
    if abs(KNOTEN[i][0]) >= 1.0e-12:
        FREI.append(2 * i)
        FREI.append(2 * i + 1)
FILTER = filter_bauen(SCHWERPUNKTE, FLAECHEN, FILTER_RADIUS)

dichten = []
for dreieck in DREIECKE:
    dichten.append(0.5)
mittelverlauf = []
aenderungsverlauf = []
for schritt in range(SCHRITTE):
    reize = lokale_reize(KNOTEN, DREIECKE, dichten,
                         LASTFAELLE, GEWICHTE, FREI)
    geglaettet = filtern(reize, FILTER)
    neue = []
    maximum = 0.0
    for i in range(len(dichten)):
        antrieb = geglaettet[i] / S0 - 1.0
        delta = begrenzen(ETA * antwort(antrieb), -MAX_DELTA, MAX_DELTA)
        neu = begrenzen(dichten[i] + delta, RHO_MIN, RHO_MAX)
        if abs(neu - dichten[i]) > maximum:
            maximum = abs(neu - dichten[i])
        neue.append(neu)
    dichten = neue
    summe = 0.0
    for dichte in dichten:
        summe = summe + dichte
    mittelverlauf.append(summe / len(dichten))
    aenderungsverlauf.append(maximum)

# Diagnose passend zum Feld nach dem 15. Schritt.
endreize = lokale_reize(KNOTEN, DREIECKE, dichten,
                        LASTFAELLE, GEWICHTE, FREI)
endmittel = filtern(endreize, FILTER)
```

```
grenzflaeche = 0.0
aussenflaeche = 0.0
gesamtflaeche = 0.0
for i in range(len(dichten)):
    gesamtflaeche = gesamtflaeche + FLAECHEN[i]
    if abs(dichten[i] - RHO_MIN) < 1.0e-12 or abs(dichten[i] - RHO_MAX) < 1.0e-12:
        grenzflaeche = grenzflaeche + FLAECHEN[i]
    if abs(endmittel[i] / S0 - 1.0) > TOTZONE:
        aussenflaeche = aussenflaeche + FLAECHEN[i]

print("Vorschau nach", SCHRITTE, "Recheniterationen (keine Zeitangabe)")
print("mittlere Dichte : %.5f g/cm3" % mittelverlauf[-1])
print("max. Aenderung  : %.5f g/cm3" % aenderungsverlauf[-1])
print("Grenzanteil     : %.1f %%" % (100.0 * grenzflaeche / gesamtflaeche))
print("ausser Totzone  : %.1f %%" % (100.0 * aussenflaeche / gesamtflaeche))

# --- Abbildung ---
import matplotlib.pyplot as plt
from matplotlib.tri import Triangulation
xs = [1000.0 * punkt[0] for punkt in KNOTEN]
ys = [1000.0 * punkt[1] for punkt in KNOTEN]
tris = [list(dreieck) for dreieck in DREIECKE]
iteration = list(range(1, SCHRITTE + 1))
fig, achsen = plt.subplots(1, 2, figsize=(9.2, 3.5))
p = achsen[0].tripcolor(Triangulation(xs, ys, tris), facecolors=dichten,
                        cmap="viridis", vmin=RHO_MIN, vmax=RHO_MAX,
                        edgecolors="0.7", linewidth=0.25)
achsen[0].set_title("Dichte nach 15 Schritten")
achsen[0].set_aspect("equal")
achsen[0].set_xlabel("x (mm)")
achsen[0].set_ylabel("y (mm)")
fig.colorbar(p, ax=achsen[0], label="Dichte (g/cm³)")
achsen[1].plot(iteration, mittelverlauf, label="Dichtemittel")
achsen[1].plot(iteration, aenderungsverlauf, label="max. Änderung")
achsen[1].set_xlabel("Recheniteration")
achsen[1].set_title("15-Schritt-Vorschau")
achsen[1].grid(alpha=0.2)
achsen[1].legend()
plt.tight_layout()
plt.show()
```

```
import os, sys
sys.path.insert(0, os.path.join("..", "programme", "kap17"))
```

```
import matplotlib.pyplot as plt
from matplotlib.tri import Triangulation
import knochen_remodeling as kr

vorschau = kr.simuliere(max_iterationen=15, filter_radius=0.030,
                        totzone=0.08, gewichte=[0.65, 0.35],
                        s0=kr.S0_REFERENZ)
xs = [1000.0 * punkt[0] for punkt in vorschau["knoten"]]
ys = [1000.0 * punkt[1] for punkt in vorschau["knoten"]]
tris = [list(dreieck) for dreieck in vorschau["dreiecke"]]
iteration = list(range(1, vorschau["iterationen"] + 1))
mittel = [eintrag["dichtemittel"] for eintrag in vorschau["diagnoseverlauf"]]

fig, achsen = plt.subplots(1, 2, figsize=(9.2, 3.5))
p = achsen[0].tripcolor(Triangulation(xs, ys, tris),
                        facecolors=vorschau["dichten"], cmap="viridis",
                        vmin=0.2, vmax=1.0, edgecolors="0.7", linewidth=0.25)
achsen[0].set_title("Dichte nach 15 Schritten")
achsen[0].set_aspect("equal")
achsen[0].set_xlabel("x (mm)")
achsen[0].set_ylabel("y (mm)")
fig.colorbar(p, ax=achsen[0], label="Dichte (g/cm³)")
achsen[1].plot(iteration, mittel, label="Dichtemittel")
achsen[1].plot(iteration, vorschau["aenderungsverlauf"], label="max. Änderung")
achsen[1].set_xlabel("Recheniteration")
achsen[1].set_title("15-Schritt-Vorschau")
achsen[1].grid(alpha=0.2)
achsen[1].legend()
plt.tight_layout()
plt.show()
```

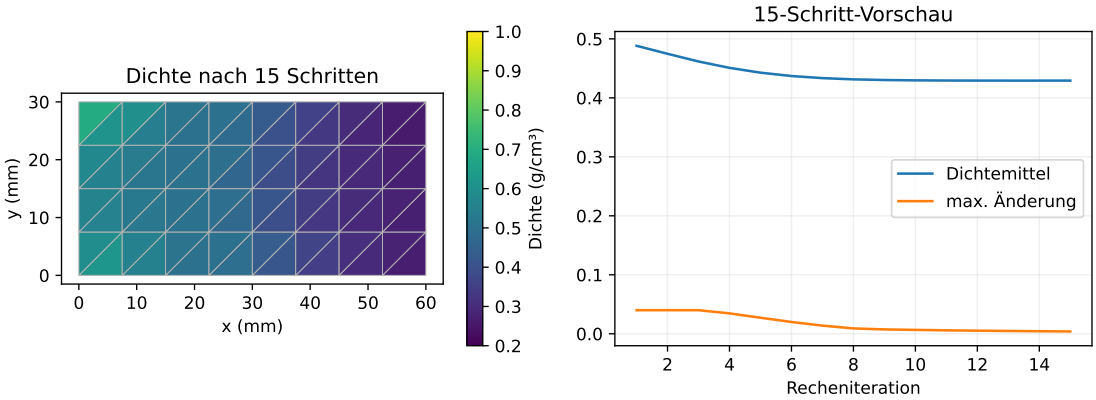

Abbildung 17.4.: Vorgerenderte Ausgangseinstellung der Remodeling-Werkstatt: 15 Recheniterationen mit R = 30 mm, Totzone 0,08 und Biegeanteil 0,65. Im HTML ersetzt die Browserrechnung dieses Bild durch dein Ergebnis.

17.11. Übungen

17.11.1. Übung 17.1 — Verstehen

Für $delta = 0{,}08$ entscheide ohne Programm, ob abgebaut, nichts geändert oder aufgebaut wird:

1. $\bar{S}/S_0 = 0{,}75$,
2. $\bar{S}/S_0 = 1{,}05$,
3. $\bar{S}/S_0 = 1{,}30$.

Berechne jeweils a und das Vorzeichen von $g(a)$.

> Lösung zu Übung 17.1
>
> 1. $a = -0{,}25 < -0{,}08$: Abbau, $g = -0{,}17$.
> 2. $a = 0{,}05$ liegt in der Totzone: keine Änderung, $g = 0$.
> 3. $a = 0{,}30 > 0{,}08$: Aufbau, $g = 0{,}22$.

17.11.2. Übung 17.2 — Verändern

Halbiere und verdopple eta. Vergleiche Status, Iterationszahl und Endfeld. Setze danach $R = 0$ und vergleiche den Hochfrequenzindikator. Warum ist

ein schneller stationärer Lauf nicht automatisch der bessere Lauf?

Lösung zu Übung 17.2

Das Skript `loesungen/kap17_ue2.py` rechnet alle vier Fälle. Mit $\eta = 0{,}04/0{,}08/0{,}16$ werden sie nach 149/88/69 Iterationen stationär. Der lokale Fall ist bereits nach 48 Iterationen stationär, hat aber $q_{\mathrm{hf}} = 0{,}36135$ statt 0,02073. Er erfüllt das Abbruchkriterium, nicht die Artefaktprüfung.

17.11.3. Übung 17.3 — Übertragen

Ergänze als dritten Lastfall 600 N diagonal nach rechts unten. Gib ihm Gewicht 0,20 und skaliere die bisherigen Gewichte auf 0,52 und 0,28. Sage vor dem Lauf voraus, wie sich der Lastpfad ändert. Interpretiere das Ergebnis nur für diese gewählte Lastmischung.

Lösung zu Übung 17.3

`loesungen/kap17_ue3.py` nutzt die öffentliche `lastfaelle`-Schnittstelle. Der erweiterte Lauf wird nach 132 Iterationen stationär; sein Dichtemittel beträgt 0,42273 g/cm³. Die diagonale Kraft koppelt Zug und Biegung im selben Lastfall. Das Resultat ist keine medizinische Vorhersage.

17.12. Was das Modell zeigt — und was nicht

Das Modell zeigt nachvollziehbar, wie Rückkopplung, Totzone, Begrenzung und räumliche Mittelung aus wiederholten FEM-Lösungen eine Dichtekarte formen. Es zeigt ebenso, dass Konvergenz mehrere Prüfungen braucht: Updatefehler, Zwei-Schritt-Abstand, Gleichgewicht, Grenzanteile, Netz- und Parameterstudien.

Es zeigt **keine** echten Trabekel. Jedes Dreieck bleibt ein isotropes, linearelastisches Dichtekontinuum; Poren werden nur über die Scheindichte vertreten. Das Modell enthält keine Osteoblasten, Osteoklasten, Hormone, Ermüdung, Schädigung, Fraktur oder Heilung. Der euklidische Filter ist nur für das konvexe Rechteck geprüft und dürfte bei Löchern oder Spalten nicht ungeprüft über getrennte Flächen hinweg mitteln. Die Lastfälle sind didaktisch, die Geometrie nicht patientenspezifisch und die Iteration keine biologische Zeit.

Die wichtigste neue Einsicht ist deshalb nicht ein bestimmtes Farbmuster, sondern die geschlossene Rechenkette: Ein Material kann auf die mechanische Antwort seines eigenen FEM-Modells reagieren — vorausgesetzt, wir formulieren und prüfen diese Rückkopplung ebenso streng wie die FEM selbst.

Glossar

Die wichtigsten Begriffe des Buches, knapp erklärt und mit Verweis auf das Kapitel, in dem sie eingeführt werden. Bei Begriffen mit Herkunftsgeschichte (Steifigkeitsmatrix, Element, Assemblierung …) steht zusätzlich, *woher* der Name stammt — das Glossar wächst mit jedem fertigen Kapitel.

A

Ableitung — die Steigung eines Profils an einer *einzelnen* Stelle: der Wert, dem die Sekantensteigung $\Delta T/\Delta x$ zuläuft, wenn man den Abstand h der beiden Punkte auf null schrumpfen lässt (Kapitel 5). Geschrieben $\mathrm{d}T/\mathrm{d}x$ — „Δ, zu Ende gedacht", mit demselben kleinen d für ein unendlich klein gewordenes Δ. Beim Träger misst sie das Temperaturgefälle, das den Wärmestrom antreibt (Kapitel 6). Für **Geraden** ist sie überall gleich und exakt gleich der gewöhnlichen Steigung — deshalb braucht die FEM dieses Buches keinen Grenzwert-Apparat.

Ansatzfunktion (auch *Formfunktion*) — legt fest, wie zwischen den Knotenwerten interpoliert wird. In diesem Buch das **Hütchen** auf der Linie (Kapitel 7) und später das „Zelt" über Dreiecken (1). Der „Ansatz" ist die Grundentscheidung der FEM, die gesuchte Lösung als gewichtete Summe $T(x) = \sum_j T_j N_j(x)$ anzusetzen — Knotenwerte T_j mal Ansatzfunktionen N_j. Beim Galerkin-Verfahren (Kapitel 7) dienen dieselben Funktionen auch als **Testfunktionen**. Ihre entscheidende Eigenschaft: Am eigenen Knoten hat N_j den Wert 1, an allen anderen 0, und sie bilden zusammen eine „Zerlegung der Eins" ($\sum_j N_j = 1$) — deshalb reproduziert die FEM konstante und lineare Verläufe exakt.

Assemblierung — das Zusammensetzen der großen Systemmatrix aus den kleinen Elementmatrizen: Jeder Element-Baustein wird an die Stelle seiner Knoten in die Systemmatrix **addiert**. Wo zwei Elemente sich einen Knoten teilen, überlappen ihre Bausteine auf dessen Diagonalfeld, und ihre Beiträge summieren sich (bei der Federkette $k + k = 2k$) — eingeführt an der Federkette (Kapitel 3). Das Wort stammt aus dem Maschinenbau (französisch *assembler*, zusammensetzen), wo man einzelne Bauteile zu einer Baugruppe fügt; die FEM übernahm es aus dem Flugzeugbau der 1950er.

B

Bandstruktur — die typische Gestalt einer FEM-Systemmatrix: Weil jeder Knoten nur mit seinen unmittelbaren Nachbarn zusammenhängt, drängen sich alle von null verschiedenen Zahlen dicht um die Diagonale, der weite Rest ist leer (Kapitel 3). Eine solche **dünn besetzte** Matrix lässt sich viel schneller lösen als eine voll besetzte — 2 nutzt das aus. Druckt man die Nullen als Punkte, wird das Band unmittelbar sichtbar.

Baryzentrische Koordinaten (Flächenanteile) — das Fachwort für die drei **Flächenanteile**, mit denen man einen Punkt in einem Dreieck beschreibt: Verbindet man den Punkt mit den drei Ecken, zerfällt das Dreieck in drei Teildreiecke, und der Anteil einer Ecke ist die Fläche des Teildreiecks *gegenüber* von ihr, geteilt durch die Gesamtfläche (1). Die drei Anteile summieren sich zu 1 und dienen als Gewichte der linearen Interpolation zwischen den drei Eckwerten — genau wie die Streckenanteile des **Hütchens** in 1D. „Baryzentrum" ist griechisch-lateinisch für **Schwerpunkt** (*barys* = schwer); die Anteile sind die Gewichte, die man auf die Ecken legen müsste, damit ihr Schwerpunkt im Punkt läge. Eingeführt von **August Ferdinand Möbius** 1827 (Möbius 1827). Als Ortsfunktion betrachtet, ist der Anteil einer Ecke ihre **Zeltfunktion** (Ansatzfunktion des P1-Dreiecks).

Benchmark (Prüffall, Referenzfall) — eine Rechenaufgabe mit *bekannter* Antwort, an der man ein Programm misst: eine analytische Lösung, eine Labormessung oder das Ergebnis eines etablierten Codes (Kapitel 12). Besteht die Software den Benchmark, darf man ihr auch bei Aufgaben *ohne* bekannte Antwort etwas mehr trauen. Die drei Benchmarks dieses Buches sind die **Gerade** (der **Nulltest**), die **Parabel** mit Wärmequelle und die transiente Abkühlkurve gegen die Reihenlösung. Profi-Programme pflegen ganze Benchmark-Sammlungen, die bei jeder Änderung automatisch nachgerechnet werden — etwa OpenGeoSys mit dem Thiem-Brunnentest (Kolditz u. a. 2012). Das Wort stammt aus der Landvermessung (englisch *bench mark*, die eingemeißelte Höhenmarke, an der man Messungen eicht).

Bilanz (Energiebilanz, Kontrollvolumen) — die Buchhaltung an einem herausgeschnittenen Stück (dem **Kontrollvolumen**): Was an Wärme hereinströmt, minus was hinausströmt, plus was im Inneren erzeugt wird, ergibt die Änderung des Gespeicherten (Kapitel 6). Im eingeschwungenen (**stationären**) Zustand ändert sich nichts mehr, also ist rein minus raus gleich null. Lässt man das Kontrollvolumen auf einen Punkt schrumpfen, wird aus der Bilanz die **Wärmeleitungsgleichung**. Das Wort „Kontrollvolumen" betont, dass man ein festes Raumstück ins Auge fasst und über seinen Rand Buch führt.

Bilineare Interpolation — die Interpolation auf einem achsparallelen Rechteck aus seinen **vier** Eckwerten: erst in x-Richtung auf Unter- und Oberkante interpolieren, dann in y-Richtung dazwischen (Kapitel 11). „Bi" (zwei) plus „linear": in jeder der beiden Richtungen einzeln geradlinig — auf jeder Kante also eine **Gerade** (das macht sie **kantenkompatibel**, sogar mit den Dreiecken aus 1). Im Inneren ist sie durch das gemischte Glied $x \cdot y$ jedoch **gekrümmt**: eine **Sattelfläche**. Das ist der eigentliche Unterschied zur linearen Interpolation des **P1-Dreiecks**, deren Ansatzfläche eben bleibt. Die vier Ansatzfunktionen sind Produkte zweier **Hütchen** (eins in x, eins in y); das Element heißt **Q1-Rechteck**.

D

Dehnung (ε) — die Längenänderung eines Bauteils im Verhältnis zu seiner Länge, $\varepsilon = \Delta L / L$, eine **dimensionslose** Zahl (Kapitel 13). Sie misst die Streckung, nicht die Strecke: derselbe Wert für einen kurzen und einen langen Träger, wenn beide relativ gleich viel nachgeben. Weil die Werte klein sind, oft in **Mikrostrain** (10^{-6}) angegeben. Es gibt zwei Quellen: **Ziehen** (elastische Dehnung) und **Erwärmen** (Wärmedehnung $\alpha\,\Delta T$) — und für die **Spannung** zählt nur die *Differenz* der beiden, die „verhinderte" Dehnung. In der Analogie-Tafel (Kapitel 13) ist die Dehnung das mechanische Gegenstück zum Temperaturgefälle $\mathrm{d}T/\mathrm{d}x$.

Delaunay-Triangulierung — das bekannteste Verfahren, aus einer Punktwolke automatisch ein **Dreiecksnetz** zu erzeugen: Es verbindet die Punkte so, dass die Dreiecke möglichst gedrungen werden (keine spitzen „Nadeln"), indem im Umkreis jedes Dreiecks kein weiterer Punkt liegt (1, nur als Namedrop). In diesem Buch schreiben wir unsere kleinen Netze von Hand; für echte, krumme Geometrien übernimmt ein Netzgenerator diese Aufgabe. Benannt nach **Boris Delaunay** (1890–1980), der das Kriterium 1934 formulierte (Delaunay 1934).

Diagonaldominant — Eigenschaft einer Matrix, bei der in jeder Zeile die Zahl auf der Diagonale betragsmäßig größer ist als der Rest der Zeile zusammen; bei der Federkette die 200 gegen zwei -100 (2). Sie kommt aus der Physik — ein Knoten reagiert stärker auf sich selbst als auf jeden einzelnen Nachbarn — und macht die FEM-Systeme gutartig: Man kann sie ohne Umsortieren (Pivotisierung) eliminieren, und Gauß-Seidel pendelt sich bei ihnen sicher ein.

Dichtekontinuum — ein Ersatzmodell, in dem jedes finite Element eine kontinuierliche **Scheindichte** und daraus einen E-Modul erhält. Poren und einzelne Trabekel werden nicht geometrisch aufgelöst; sie wirken nur über

die lokalen Materialwerte. Das Remodeling in Kapitel 17 erzeugt deshalb dichtere und weniger dichte Zonen, aber keine echten Knochenbälkchen.

Dirichlet-Randbedingung (wesentliche Randbedingung) — eine **Randbedingung**, die den **Wert** der gesuchten Größe am Rand vorgibt; beim Träger die festgehaltene Temperatur, etwa $T_1 = 20\,°\mathrm{C}$ am kühlen Auflager (Kapitel 8). Sie muss dem Gleichungssystem *aufgezwungen* werden — durch **Streichen** der Zeile und Spalte des bekannten Knotens (Kapitel 3, 7) oder durch Ersetzen seiner Zeile durch die triviale Gleichung $1 \cdot T = 20$; beide Wege liefern dieselbe Lösung, Profi-Programme benutzen oft einen **Strafterm**. Ohne mindestens eine Dirichlet-Bedingung ist ein reines Wärmeleitproblem nur bis auf eine Konstante bestimmt (siehe **isoliert**, schwimmender Träger). „Wesentlich" heißt sie, weil man sie von Hand einbauen muss — Gegenstück zur **natürlichen** (Neumann-)Randbedingung. Benannt nach **Peter Gustav Lejeune Dirichlet** (1805–1859), der die Wärmeleitung und die nach ihm benannte Randwertaufgabe untersuchte.

Differentialgleichung — eine Gleichung, in der nicht nur eine gesuchte Größe vorkommt, sondern auch ihre **Ableitungen** (Steigungen); gesucht ist darum keine einzelne Zahl, sondern eine ganze **Funktion**, die die Bedingung an *jedem* Punkt erfüllt (Kapitel 6). Die erste des Buches ist die stationäre **Wärmeleitungsgleichung** $\frac{\mathrm{d}}{\mathrm{d}x}\left(\lambda \frac{\mathrm{d}T}{\mathrm{d}x}\right) = 0$ — sie sucht das Temperaturprofil $T(x)$, dessen Steigungen die Wärme-**Bilanz** überall aufgehen lassen. „Differential" steht für die Ableitung (das zu Ende geschrumpfte Δ, Kapitel 5), „Gleichung" wie immer. Die FEM (Kapitel 7) ist die Maschine, die solche Gleichungen löst, wenn man sie nicht raten kann.

Diskretisierungsfehler — der Fehler, der allein daher rührt, dass die FEM mit *endlich* vielen Knoten und geraden Stücken rechnet statt mit dem unendlich fein aufgelösten echten Verlauf (Kapitel 12). Er ist der *einzige* Fehler, den ein feineres **Netz** heilt — mit dem **Viertelungs-Rhythmus** der **Konvergenz**. In 1D trifft die FEM die Knoten sogar exakt; der Diskretisierungsfehler lebt dann ganz *zwischen* den Knoten, wo der Streckenzug die Kurve abschneidet. Streng zu trennen von Fehlern, die Verfeinern *nicht* heilt: eine vergessene **Randbedingung**, ein Einheiten- oder Vorzeichenfehler, ein Programmierfehler — die bleiben auf jedem Netz falsch (Kapitel 12).

Dünn besetzt — Eigenschaft einer Matrix, deren allergrößter Teil aus Nullen besteht; nur wenige Felder tragen eine Zahl (2). Die FEM-Systemmatrix ist dünn besetzt, weil jeder Knoten nur mit seinen unmittelbaren Nachbarn zusammenhängt — die belegten Zahlen drängen sich als schmales Band um die Diagonale (siehe **Bandstruktur**). Ein Löser, der die Nullen gar nicht erst anfasst (siehe **Thomas-Algorithmus**), spart dadurch fast die gesamte Rechenarbeit.

E

Einwegkopplung (Kopplung, Einbahnstraße) — die im Buch verwendete Verbindung zwischen zwei Physiken, bei der der Einfluss nur in *eine* Richtung läuft: Die **Temperatur** erzeugt über die **Wärmeausdehnung** eine Verformung und **Spannung**, aber die Verformung wirkt *nicht* auf die Temperatur zurück (Kapitel 13, Kapitel 14). Deshalb kann man erst das Wärmeproblem vollständig lösen und sein Ergebnis anschließend als feste Last in die Mechanik einsetzen — zwei getrennte Rechnungen statt einer verzahnten. Eine echte **Kopplung** in beide Richtungen (die Verformung ändert die Wärmeleitung) ist möglich, aber teurer; das Finale (3) bleibt bewusst bei der Einbahnstraße (Strategie: bewusste Vereinfachungen).

Ebener Spannungszustand — die Modellannahme der 2D-Mechanik für ein **dünnes**, in die Tiefe **freies** Bauteil (ein Blech, unser $0{,}1$ m-Träger): Quer zur Zeichenebene baut sich keine Spannung auf ($\sigma_{zz} = 0$), das Material darf dort ungehindert dicker oder dünner werden (Kapitel 14). Aus ihm folgt die 3×3-**Materialtabelle** $\frac{E}{1-\nu^2}\begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}$, die die drei Dehnungen mit den drei Spannungen verknüpft. Gegenstück ist der **ebene Verzerrungszustand** (plane strain): ein sehr langer Körper, der in die Tiefe *nicht* ausweichen kann (Tunnel, Staudamm, Schweißnaht) — dort ist die Dehnung quer null statt der Spannung, und die Tabelle sieht etwas anders aus. Die Verwechslung der beiden ist ein häufiger Praxisfehler; für den dünnen Träger dieses Buches ist der ebene Spannungszustand die richtige Wahl.

Elastizitätsmodul (E-Modul, E) — die Materialzahl im **Hookeschen Gesetz** $\sigma = E\,\varepsilon$: wie viel **Spannung** ein Material für eine bestimmte **Dehnung** braucht, angegeben in GPa (Kapitel 13). Ein großes E heißt steif (Stahl 210 GPa), ein kleineres nachgiebig (Aluminium 70 GPa). Anders als die Federhärte k aus Kapitel 3 ist E eine reine **Material**eigenschaft, unabhängig von der Bauteilform. Im FEM-Gleichungssystem übernimmt E genau die Rolle, die bei der Wärme die **Wärmeleitfähigkeit** λ spielt — die Element-Steifigkeit ist $k = E\,A/h$ statt $\lambda A/h$ (Analogie-Tafel, Kapitel 13). Im Englischen *Young's modulus*, nach dem britischen Universalgelehrten **Thomas Young** (1773–1829), der die Steifigkeit um 1807 als bauteilunabhängige Kennzahl fasste (Young 1807).

Elementmatrix — der „Steckbrief" eines einzelnen finiten Elements: ein kleines, ordentlich aufgeschriebenes Gleichungssystem, das die Knoten des Elements verknüpft. Bei einer Feder ist es die 2×2-Tabelle $\begin{pmatrix} k & -k \\ -k & k \end{pmatrix}$ (Kapitel 3). Aus den vielen Elementmatrizen entsteht durch **Assemblierung**

die große Systemmatrix. Der Begriff „Element" stammt aus dem Flugzeugbau der 1950er, wo man den Tragflügel in kleine, berechenbare Teilstücke zerlegte.

Elementintegral — das über ein *einzelnes* Element gerechnete Integral, aus dem die Einträge der **Elementmatrix** entstehen; bei der 1D-Wärmeleitung $\int_{\text{Element}} \lambda A \, \frac{\mathrm{d}N_r}{\mathrm{d}x} \frac{\mathrm{d}N_c}{\mathrm{d}x} \, \mathrm{d}x$ (Kapitel 7). Weil die Hütchen-Steigungen über ein Element konstant sind ($\pm 1/h$), ist der Integrand konstant und das Integral eine bloße **Rechteckfläche** (Höhe mal Breite, Kapitel 5) — so wird jeder Matrixeintrag zu Steigung × Steigung × λA × Elementlänge. Die FEM rechnet grundsätzlich Element für Element und **assembliert** die Elementintegrale danach zur Systemmatrix.

Euler-Verfahren (expliziter und impliziter Euler) — die beiden einfachsten Wege, einen **Zeitschritt** zu machen (Kapitel 9). Der **explizite** Euler nimmt die Änderungsrate am *jetzigen* Zeitpunkt („neuer Wert = alter Wert + Zeitschritt × jetzige Rate") — billig, aber nur unter der **Stabilität**sgrenze brauchbar. Der **implizite** (auch *rückwärtige*) Euler setzt die Rate am *neuen* Zeitpunkt an und muss darum pro Schritt ein Gleichungssystem lösen — teurer, aber bei jedem Zeitschritt stabil. Benannt nach **Leonhard Euler** (1707–1783), dem Schweizer Mathematiker, der das Grundverfahren in seiner *Institutiones calculi integralis* (1768–1770) beschrieb (Euler 1768).

F

Freiheitsgrad — eine einzelne unbekannte Zahl, die das Gleichungssystem bestimmen muss; eine „Stellschraube" der Lösung (Kapitel 13, in Kapitel 3 bewusst noch vertagt). Beim 1D-Träger trägt jeder **Knoten** *einen* Freiheitsgrad: seine Verschiebung u (beim Wärmeproblem die Temperatur T); fünf Knoten ergeben ein 5×5-System. Ab Kapitel 14 bekommt jeder Knoten in der Fläche *zwei* Freiheitsgrade (Verschiebung nach rechts und nach oben), und die Systeme verdoppeln sich. Die Zahl der Freiheitsgrade ist das Maß für Größe und Rechenkosten einer FEM-Rechnung.

Funktion — ein benannter, wiederverwendbarer Rechenweg im Python-Programm: Vorne gehen die Argumente hinein, hinten kommt hinter `return` ein Ergebnis heraus; jede Funktion des Buches trägt einen Docstring (Kapitel 2). Funktionen sind die „Kapitel eines Programms"; ab Kapitel 7 werden die FEM-Bausteine als Funktionen gebaut (`baue_elementmatrix`, `loese_gleichungssystem`).

Finites Element — eines der kleinen, einfachen Stücke, in die man das zu berechnende Objekt zerlegt; auf jedem gilt eine einfache, lokale Physik

(Kapitel 1). „Finite" (endlich) betont den Gegensatz zum unendlich fein verschmierten echten Material — man rechnet mit endlich vielen Stücken. Den Namen prägte Ray Clough 1960 (Clough 1960); die Sache selbst geht auf den Flugzeugbau der 1950er zurück (Turner u. a. 1956) (Kapitel 3).

Flächenanteile — die drei Zahlen, die einen Punkt in einem Dreieck beschreiben: die Flächen der drei Teildreiecke (Punkt mit den Ecken verbunden), jede geteilt durch die Gesamtfläche (1). Sie summieren sich zu 1 und sind die Gewichte, mit denen zwischen den drei Eckwerten interpoliert wird — das flächenhafte Gegenstück zu den **Streckenanteilen** des **Hütchens** in 1D. Das Fachwort dafür ist **baryzentrische Koordinaten**; als Funktion des Ortes betrachtet, ist der Anteil einer Ecke ihre **Zeltfunktion**. Berechnet werden die Flächen mit der „halben Kreuzdifferenz" (Schulformel ohne Vektorprodukt).

Formänderungsenergiedichte (U) — die elastische Energie, die ein verformtes Bauteil je Volumen speichert, $U = \frac{1}{2}\,\sigma^T \varepsilon$ (Spannung mal Dehnung, halbiert wie die Federenergie $\frac{1}{2}kx^2$). Auf die Knochenmasse bezogen dient sie in Remodeling-Modellen als **mechanischer Reiz** (Kapitel 17).

Fouriersches Gesetz — das Naturgesetz der Wärmeleitung: Die Wärmestromdichte ist proportional zum Temperaturgefälle und fließt vom Warmen zum Kalten, $q = -\lambda \frac{\mathrm{d}T}{\mathrm{d}x}$ (Kapitel 1, ausführlich Kapitel 6). Drei Teile: das **Gefälle** $\mathrm{d}T/\mathrm{d}x$ (die Steigung aus Kapitel 5) treibt den Strom, die **Wärmeleitfähigkeit** λ sagt, wie gut das Material leitet, und das **Minuszeichen** dreht die Richtung um — Wärme läuft dem Gefälle entgegen, „bergab" vom Warmen zum Kalten. Benannt nach Joseph Fourier, der die Wärmeleitung 1822 in seiner *Théorie analytique de la chaleur* mathematisch fasste (Fourier 1822).

G

Galerkin-Verfahren — die Ausprägung der **Methode der gewichteten Reste**, bei der die **Testfunktionen** dieselben sind wie die **Ansatzfunktionen** (die Hütchen): „Ansatz und Test aus demselben Baukasten" (Kapitel 7). Diese Wahl macht die Systemmatrix **symmetrisch** und ist der Standardweg der FEM. Benannt nach dem russischen Ingenieur **Boris Galjorkin** (1871–1945), der das Verfahren um 1915 an schwingenden Platten und Stäben entwickelte. Andere Wahlen der Testfunktionen liefern andere Verfahren (Punktproben → die Finite-Differenzen-Idee, Zellmittel → Finite Volumen).

Gewichtete Reste (Methode der) — das Grundrezept, eine Differentialgleichung näherungsweise zu erfüllen: Statt den Fehlbetrag der Gleichung

(das **Residuum**) an *jedem* Punkt auf null zu zwingen — das ginge bei endlich vielen Knotenwerten nicht —, fordert man ihn nur *im gewichteten Mittel* null, und zwar einmal je Knoten mit dessen Hütchen als Gewicht (Kapitel 7). So viele Gewichte wie Knoten, so viele Gleichungen wie Unbekannte. Weil jedes Hütchen nur seine Nachbarschaft „befühlt", wird die Matrix **dünn besetzt**. Die Galerkin-Wahl (Gewicht = Ansatzfunktion) ist der in diesem Buch verwendete Spezialfall.

Gauß-Elimination — das grundlegende Verfahren, ein lineares Gleichungssystem zu lösen: Erst macht man mit geschickten Zeilen-Kombinationen alle Zahlen **unter** der Diagonale zu null (leerräumen, bis eine Dreiecksform steht), dann rechnet man von unten nach oben eine Unbekannte nach der anderen aus (**Rückwärtseinsetzen**) — 2. Es funktioniert für jedes lösbare System, kostet bei n Unbekannten aber rund $n^3/3$ Rechenschritte. Benannt nach Carl Friedrich Gauß, der es in eine feste Form brachte.

Gauß-Seidel-Verfahren — ein iteratives Lösungsverfahren (**Entspannungsverfahren**). Statt das System in einem Zug zu lösen, korrigiert man Runde um Runde jeden Knotenwert in den Mittelwert seiner Nachbarn, bis sich nichts mehr merklich ändert; die Lösung „pendelt sich ein" (2). Langsamer als Gauß bei kleinen Systemen, aber sparsam bei den großen 2D-Netzen (1). Benannt nach Carl Friedrich Gauß und Philipp Ludwig von Seidel.

Gleichungssystem — die vielen lokalen Element-Beziehungen, zu einem einzigen großen System aus Gleichungen zusammengesetzt; seine Lösung liefert die gesuchten Werte an allen Knoten auf einmal (Kapitel 1, gebaut ab Kapitel 3, gelöst in 2).

H

Homöostase — hier der Zustand, in dem ein stationäres Remodeling-Feld zugleich vollständig innerhalb der **Totzone** liegt. Ein bloß stationäres Feld kann dagegen nur wegen einer Dichtegrenze stehen bleiben, obwohl sein mechanischer Reiz weiter nach Aufbau oder Abbau verlangt (Kapitel 17).

Hookesches Gesetz — das Kraftgesetz einer Feder: Die Kraft ist proportional zur Dehnung, $F = k \cdot (u_2 - u_1)$, mit der Steifigkeit k als Proportionalitätszahl (Kapitel 3). Es ist die einfache, lokale Physik, aus der die Elementmatrix einer Feder folgt. Benannt nach Robert Hooke, der den Zusammenhang im 17. Jahrhundert formulierte.

Hütchenfunktion — die konkrete **Ansatzfunktion** des 1D-Trägers: eine Dachform, die über ihrem eigenen Knoten auf 1 steht und zu beiden Nachbarn geradlinig auf 0 fällt, weiter draußen null (Kapitel 7). Anschaulich das

Zuständigkeitsgebiet eines Knotens — nur dort, wo sein Hütchen ungleich null ist, hat er ein Wort mitzureden. Weil sich an jeder Stelle höchstens zwei Hütchen überlappen, wird jeder Punkt von höchstens zwei Knoten „regiert"; die Temperatur dazwischen ist ihre gewichtete Mischung (lineare Interpolation). In der Fläche wird aus dem Hütchen ein „Zelt" über Dreiecken (1).

I

Index — die Nummer eines Fachs in einer Liste, in eckigen Klammern geschrieben (`temperaturen[0]`). Python zählt **ab null**: Der erste Eintrag hat den Index 0, der letzte den Index `len(...) - 1` (Kapitel 2). Der Index eines Knotens zieht sich durch das ganze Buch — er verbindet die Zahl in der Liste mit ihrem Ort auf dem Träger.

Integral — die Fläche unter einer Kurve: der Wert, dem die Summe immer schmalerer Rechteckstreifen zuläuft (Kapitel 5). Geschrieben $\int T \,\mathrm{d}x$; das $\int$ ist ein langgezogenes **S** für „Summe", das $\mathrm{d}x$ die Breite eines unendlich schmalen Streifens, T seine Höhe. Beim Träger ist die Fläche unter dem Temperaturprofil ein Maß für die gespeicherte Wärme (mal $\rho\, c\, A$ ergibt sie Joule, Kapitel 7). Für **Geraden** ist die Fläche ein Trapez und darum exakt ohne Grenzwert.

Isoliert — der Sonderfall der **Neumann-Randbedingung** mit Wärmestrom **null**: Über einen isolierten Rand fließt keine Wärme (Kapitel 8). Anschaulich eine perfekte Dämmschicht — die Thermoskanne, die Vakuum-Doppelwand. In der FEM ist das die **eingebaute Standardannahme**: Wer an einem Rand *gar nichts* vorgibt, lässt den geparkten **Randterm** aus Kapitel 7 auf seinem Wert null stehen und isoliert dieses Ende, ob gewollt oder nicht. Genau darum ist „nichts vorschreiben" nicht „alles offen lassen", sondern eine harte physikalische Aussage. Mit links gehaltenen 20 °C, ohne Quelle und rechts isoliert läuft der ganze Träger auf 20 °C.

Isoparametrisch — der Standardtrick der Profi-FEM-Programme, um **schiefe, verzerrte Vierecke** (statt nur achsparalleler Rechtecke) zu berechnen: Man rechnet auf einem bequemen **Referenzquadrat** und benutzt *dieselben* vier bilinearen Ansatzfunktionen sowohl für die **Lösung** als auch für die **Form** des Elements — daher *iso* (griechisch „gleich") *parametrisch* (Kapitel 11, nur als Ausblick). Beim Umrechnen der Integrale aufs Referenzquadrat tritt ein Dehnungsfaktor auf (die **Jacobi-Determinante**), integriert wird mit **Gauß-Quadratur**. Für die achsparallelen Rechtecke dieses Buches ist das unnötig (die Integrale bleiben Produkte von 1D-Flächen), für krumme Ränder aber unverzichtbar.

I

Isotrop und anisotrop — *isotrop* heißt: Das Material verhält sich in **jede Richtung** gleich, ein einziger **Elastizitätsmodul** genügt — die Annahme dieses Buches für Stahl wie für Knochen. *Anisotrop* wäre richtungsabhängig, etwa faseriger Knochen, der längs steifer ist als quer. Der Femur in Kapitel 16 wird bewusst isotrop gerechnet (benannte Vereinfachung).

Iteration — ein Rechenschritt, der wiederholt ausgeführt wird und sein eigenes Ergebnis als Ausgangspunkt der nächsten Wiederholung nimmt; jede Wiederholung heißt eine **Runde**. Iterative Verfahren nähern sich der Lösung schrittweise an, statt sie in einem Zug auszurechnen — das **Gauß-Seidel-Verfahren** ist das Beispiel dieses Buches (2). Ein **Abbruchkriterium** legt fest, wann die Näherung gut genug ist (etwa: kein Wert ändert sich mehr um mehr als $0{,}001$).

K

Kerbwirkung (Spannungskonzentration) — die Erscheinung, dass die Spannung an einer plötzlichen Formänderung — einem Loch, einer Kante, einer scharfen Ecke — örtlich stark **überhöht** wird: Die Kraftlinien, die vorher gleichmäßig durch das Bauteil liefen, müssen sich um die Störung herumdrängen und rücken an ihren Rändern eng zusammen (3). Je schärfer die Ecke, desto höher der Spitzenwert — ein *runder* Übergang entschärft die Kerbe, eine *scharfe* Ecke verschlimmert sie. Berüchtigtes Beispiel sind die im Zweiten Weltkrieg an ihren **rechteckigen** Ladeluken zerbrochenen Liberty-Schiffe (Sridhar 2016). In der FEM zeigt sich die Kerbwirkung als lokale Spannungsspitze, die ein grobes **Netz** unterschätzt — sie ist ein Musterfall für die **Konvergenz**probe aus Kapitel 12.

Knoten — ein Punkt des Netzes (meist eine Ecke eines Elements), an dem der gesuchte Wert — beim Träger die Temperatur — tatsächlich gespeichert und ausgerechnet wird; zwischen den Knoten wird interpoliert (Kapitel 1).

Kompakta (kortikaler Knochen, Knochenrinde) — die dichte, harte Außenschicht eines Knochens; mit rund 17 GPa die steifste Zone und im Modell die kortikale Randschale (Kapitel 16). Gegenstück ist die **Spongiosa**.

Konvergenz (Konvergenzordnung, Viertelungs-Rhythmus) — die Eigenschaft einer FEM-Rechnung, dass ihr **Diskretisierungsfehler** gegen null geht, wenn das **Netz** feiner wird — und *wie schnell* (Kapitel 12). Für die linearen Elemente dieses Buches gilt die **zweite Ordnung**: Halbiert man

die Maschenweite h, so **viertelt** sich der Fehler ($2^2 = 4$). Der **Viertelungs-Rhythmus** ist der praktische Test dafür — man rechnet dieselbe Aufgabe auf h, $h/2$, $h/4$ und schaut, ob sich der Fehler brav viertelt; tut er das nicht, stimmt etwas nicht. Beim **Zeitschritt** des impliziten **Euler-Verfahrens** ist es nur *erste* Ordnung (Halbieren von Δt halbiert den Fehler) — „Zeit ist gröber als Raum". Konvergenz sagt nichts über *Richtigkeit*: Ein Programm kann sauber konvergieren und trotzdem gegen die falsche Antwort laufen (siehe **Verifikation**).

L

Liste — eine geordnete Reihe von Werten unter einem einzigen Namen (Bild: ein Eierkarton mit nummerierten Mulden), in eckigen Klammern geschrieben (Kapitel 2). Die Liste ist im Buch das Bild für eine **diskretisierte Größe**: eine Zahl pro Knoten — die Temperatur, später die Verschiebung oder Kraft. Eine **verschachtelte Liste** (Liste von Listen) ist eine Tabelle bzw. Matrix (Kapitel 3).

Löser — das Rechenverfahren, das das große Gleichungssystem knackt und die Knotenwerte bestimmt (Kapitel 1). Dieses Buch baut drei: die **Gauß-Elimination** (allgemein, aber teuer), den **Thomas-Algorithmus** (für dünn besetzte Bandmatrizen, sehr schnell) und das iterative **Gauß-Seidel-Verfahren** (2). Welchen man nimmt, entscheidet die Gestalt der Matrix und ihre Größe — bei einer Million Unbekannten liegen zwischen dem passenden und dem unpassenden Löser Jahre gegen Sekundenbruchteile.

Lumping (Mass Lumping) — das Zusammenklumpen der verteilten **Massenmatrix** zu einer reinen Diagonale: Die über ein Element verschmierte Wärmekapazität wird je zur Hälfte auf seine beiden Knoten geschoben, sodass jeder Knoten seinen eigenen „Eimer" bekommt und keine Kopplung zu den Nachbarn übrig bleibt (Kapitel 9). Vom englischen *lump*, der Klumpen: *to lump together* = in einen Topf werfen. Die **Gesamtmasse bleibt erhalten** (Zeilensumme unverändert). Der Gewinn: Der **explizite** Zeitschritt wird eine bloße Division je Knoten statt eines Gleichungssystems. Beim 1D-Träger bekommt jeder innere Knoten die volle Elementkapazität (9027,5 J/K), jeder Randknoten die halbe (4513,75 J/K).

M

Massenmatrix — die Matrix des **Speicherterms** einer transienten FEM-Rechnung: Sie wiegt ab, wie stark die zeitliche Temperaturänderung an einem Knoten in die Bilanz seiner Nachbarn hineinwirkt (Kapitel 9). Entsteht wie die **Steifigkeitsmatrix** aus der **schwachen Form**, nur mit dem Produkt zweier **Hütchen** statt ihrer Steigungen; ihr Vorfaktor ist die Wärmekapazität $\rho c A h$. Exakt integriert („konsistent") koppelt sie benachbarte Knoten; durch **Lumping** wird sie diagonal. Der Name stammt aus der Strukturmechanik, wo dieselbe Matrix die *Trägheit der Masse* beim Schwingen beschreibt — beim Wärmeproblem ist es die Trägheit der gespeicherten Wärme.

Mechanischer Reiz (S) — die gewichtete **Formänderungsenergiedichte** je Knochenmasse, im Remodeling-Modell $S = (\sum_l p_l U_l)/\rho_\mathrm{m}$ in J/kg. Er ist eine mechanische Rückkopplungsgröße des Ersatzmodells, kein universeller biologischer Messwert (Kapitel 17).

N

Mechanostat — der von Harold Frost geprägte biologische Rahmen für lastgesteuerten Knochenumbau: Wie ein Thermostat die Temperatur regelt, hält der Mechanostat die Knochenmasse nach dem mechanischen Reiz nach — Aufbau bei viel, Abbau bei wenig Belastung (Kapitel 17). Siehe **Remodeling**, **Wolffsches Gesetz**.

Natürliche Randbedingung — eine **Randbedingung**, die sich schon aus der **schwachen Form** ergibt, ohne dass man ins Gleichungssystem eingreifen muss; bei der Wärmeleitung die **Neumann**-Bedingung (Fluss vorgeben), weil sie genau der **Randterm** der partiellen Integration ist (Kapitel 7, Kapitel 8). Man baut sie durch einen bloßen Eintrag auf der **rechten Seite** ein. Der Fluss null („isoliert") ist der Standardfall, der ganz von selbst gilt, wenn man nichts tut. Gegenstück: die **wesentliche Randbedingung** (Dirichlet), die man dem System aufzwingen muss.

Neumann-Randbedingung (natürliche Randbedingung) — eine **Randbedingung**, die den **Fluss** (die Steigung) am Rand vorgibt; beim Träger den eintretenden Wärmestrom, etwa den Brandherd mit $200\,\mathrm{W}$ am rechten Ende (Kapitel 8). Sie ist genau der in Kapitel 7 geparkte **Randterm** und wandert darum als **einzige Zahl auf die rechte Seite** des Gleichungssystems — die Matrix bleibt unberührt. Der Sonderfall Fluss null ist der **isolierte** Rand.

„Natürlich", weil sie ohne Extra-Handgriff aus der schwachen Form folgt. Benannt nach **Carl Gottfried Neumann** (1832–1925), einem der Begründer der Potentialtheorie.

Netz — die Zerlegung der Geometrie in finite Elemente und Knoten. In 1D die Knotenkette auf der Linie, in der Fläche die **Triangulierung** in Dreiecke (1). Beschrieben wird es durch zwei Listen: eine **Knotenliste** (für jeden Knoten seine Koordinaten) und eine **Elementliste** (für jedes Element die Nummern seiner Eckknoten) — zusammen der „Bauplan", den ein Programm liest. Ein feineres Netz (mehr Knoten) bildet die Wirklichkeit genauer ab, kostet aber mehr Rechenzeit (Kapitel 1); *wie viel* genauer, misst Kapitel 12. Für echte, krumme Formen erzeugt ein Netzgenerator die Elementliste automatisch (siehe **Delaunay-Triangulierung**).

Nichtlokal — eine Rechengröße hängt nicht nur vom Zielelement selbst ab, sondern auch von einer räumlichen Nachbarschaft. In Kapitel 17 wird der mechanische Reiz mit abstands- und flächengewichteten Nachbarelementen gemittelt; `R = 0` schaltet diesen Filter als lokalen Kontrollfall aus.

Nulltest — der schärfste **Verifikations**-Prüffall: eine Aufgabe, deren exakte Lösung die FEM *punktgenau* treffen muss, weil sie im **Ansatzraum** liegt und darum *keinen* **Diskretisierungsfehler** zulässt (Kapitel 12). Beim Träger ist es die **Gerade** ohne Wärmequelle: Die linearen Elemente stellen eine Gerade exakt dar, also muss die FEM auf *jedem* Netz exakt 20 → 300 Grad liefern. Jede noch so kleine Abweichung ist dann *garantiert* ein Programm- oder **Randbedingungs**fehler — es gibt keine gutmütige Ausrede. Nulltests sind auch das erste Werkzeug in der Thermomechanik (Kapitel 13): freie Wärmedehnung ohne Einspannung muss **Spannung null** ergeben. Sein blinder Fleck: Fehler, die die exakte Lösung nicht ändern (etwa ein falsches λ ohne Quelle), bleiben unsichtbar — dann hilft nur die **Bilanz** bzw. der **Wärmestrom**.

P

P1-Dreieck (lineares Dreieckselement) — das einfachste finite Element in der Fläche: ein Dreieck mit einem Knoten je Ecke und **linearer** Interpolation dazwischen (1). Das „P1" steht für Polynome vom Grad 1 — über dem Dreieck ist das Feld immer eine **Ebene**, sein Gradient darum konstant. Es ist das flächenhafte Gegenstück zum linearen Stab-Element in 1D (Kapitel 7); seine Ansatzfunktionen sind die **Zeltfunktionen**, seine Elementmatrix ist 3×3 und entsteht aus „Steigung mal Steigung mal Fläche". Gegenstück in Kapitel 11 ist das **Q1-Rechteck** (bilinear).

Partielle Integration — die Rechenregel, die im Buch als **Umverteilung** eingeführt wird: Sie nimmt einem Faktor eine Ableitung ab und legt sie dem anderen auf („die Steigungslast auf beide Schultern"), wobei ein **Randterm** entsteht (Kapitel 7). In der schwachen Form der Wärmeleitung verwandelt sie $\int (\lambda A\, T')'\, N_i \,\mathrm{d}x$ in $-\int \lambda A\, T'\, N_i' \,\mathrm{d}x$ plus Randterm — aus *einer zweiten* Ableitung von T werden *zwei erste* (je eine auf T und auf das Gewicht N_i). Genau das rettet den Streckenzug, dessen zweite Ableitung an den Knicken nicht existiert.

Q

Q1-Rechteck (bilineares Viereckselement) — das einfachste finite Element auf einem **achsparallelen Rechteck**: vier Knoten (einer je Ecke) und **bilineare** Interpolation dazwischen (Kapitel 11). Das „Q" steht für *quadrilateral* (Viereck), die „1" für Grad 1 in jeder Richtung einzeln. Seine vier Ansatzfunktionen sind Produkte zweier **Hütchen** (eins in x, eins in y); auf jeder Kante sind sie linear (**kantenkompatibel**), im Inneren durch das Glied $x \cdot y$ zur **Sattelfläche** gekrümmt. Die Elementmatrix ist 4×4 und zerfällt in Produkte von 1D-Integralen (Steigung mal Steigung in der einen Richtung, Hütchen-Überlapp in der anderen). Im Gegensatz zum **P1-Dreieck** koppelt es auch diagonal benachbarte Knoten — auf dem Rechteckgitter ein **9-Punkt-Stern** (das Dreieck: 5 Punkte). Gegenstück ist das **P1-Dreieck** aus 1; für schiefe Vierecke wird es **isoparametrisch** verallgemeinert.

Querkontraktion (Querkontraktionszahl, Poisson-Zahl ν) — das seitliche Ausweichen eines Materials: Drückt man einen Radiergummi zusammen, wird er dicker; zieht man an einem Gummiband, wird es dünner (Kapitel 14). Die **Querkontraktionszahl** ν misst das Verhältnis „seitliches Einschnüren geteilt durch Längsdehnung", eine reine Zahl zwischen 0 und $\frac{1}{2}$: Kork ≈ 0 (deshalb lässt sich ein Korken eindrücken, ohne zu klemmen), Metalle um $0{,}3$ (unser Stahl $0{,}30$), Gummi $\approx 0{,}5$ (weicht vollständig aus, Volumen bleibt gleich — dort bekommt die Standard-FEM Schwierigkeiten). Sie ist die eine neue Physik-Zutat, die 2D-Mechanik von drei nebeneinandergelegten 1D-Stäben unterscheidet: Sie koppelt die Richtungen x und y in der 3×3-**Materialtabelle** des **ebenen Spannungszustands**. Benannt nach **Siméon Denis Poisson** (1781–1840), der den Effekt um 1829 beschrieb (Poisson 1829). Wichtig: Die thermische *Krümmung* $\alpha\, \Delta T / H$ enthält kein ν — die Querkontraktion macht die Buchhaltung reicher, ist aber nicht der Motor der Biegung.

R

Randbedingung — was am Rand des Gebiets vorgegeben ist und das im Inneren gültige Gleichungssystem erst lösbar macht (Kapitel 1, Thema von Kapitel 8). Drei Grundtypen: **Dirichlet** (Temperatur/Wert vorgeben, die *wesentliche*), **Neumann** (Wärmestrom/Fluss vorgeben, die *natürliche*) und **Robin** (Übergang, eine Mischung). Beim Träger sind es die fest gehaltene Temperatur links (20 °C) und der Brandherd oder Isolierung rechts. Faustregel: Ohne mindestens eine Dirichlet-Bedingung *schwimmt* das Wärmeproblem (nur bis auf eine Konstante bestimmt); ein Rand, über den man nichts sagt, ist nicht offen, sondern **isoliert**. Randbedingungen sind der häufigste Anwenderfehler — dafür die Checkliste „Randbedingungs-Review" in Kapitel 8.

Robin-Randbedingung (Übergangs-, Konvektionsrandbedingung) — eine **Randbedingung**, bei der der **Fluss linear von der Temperatur selbst abhängt**: $Q = \alpha\, A\,(T - T_{\text{Umgebung}})$, mit dem Wärmeübergangskoeffizienten α (Kapitel 8, angewendet ab 3). Sie ist eine **Mischung** aus **Dirichlet** und **Neumann**: für sehr großes α wird das Ende auf die Umgebungstemperatur gepresst (Dirichlet), für $\alpha = 0$ fließt nichts (isolierter Neumann). Weil ein Teil von der gesuchten Temperatur abhängt, wandert er teils in die **Matrix** (auf die Diagonale des Randknotens), teils auf die rechte Seite. Sie beschreibt die realistische Luftkühlung. Benannt nach **Victor Gustave Robin** (1855–1897), französischer Mathematiker mit Arbeiten zur Thermodynamik; die Zuschreibung gilt historisch als nicht ganz gesichert (Gustafson und Abe 1998), der Name hat sich aber eingebürgert.

Rückwärtseinsetzen — die zweite Phase der **Gauß-Elimination**, „die Ernte": Steht das System erst in Dreiecksform, rechnet man von der letzten Gleichung nach oben eine Unbekannte nach der anderen aus, jede aus den schon bekannten darüber (2). Beim **Thomas-Algorithmus** ist es die Rückwärtswelle, die von rechts nach links durch das Band läuft.

Remodeling (Knochenumbau) — der lebenslange Auf- und Abbau von Knochen als Antwort auf mechanische Belastung. Als FEM-**Rückkopplung** formuliert: Dichte → E-Modul → Rechnung → Reiz → neue Dichte (Kapitel 17). Siehe **Mechanostat**, **Wolffsches Gesetz**.

Residuum (Rest) — der Fehlbetrag einer Differentialgleichung, wenn man eine genäherte Lösung einsetzt: Wäre die Näherung exakt, wäre das Residuum überall null (Kapitel 7). Weil endlich viele Knotenwerte das nicht an jedem Punkt schaffen, verlangt die **Methode der gewichteten Reste** nur, dass das Residuum im Mittel gegen jedes Gewicht (jede Testfunktion) verschwindet — der verbleibende Fehler wird in die Richtungen abgedrängt, die der grobe Ansatzraum ohnehin nicht darstellen kann.

S

Sattelfläche (hyperbolisches Paraboloid) — die doppelt gekrümmte Form der bilinearen Ansatzfläche des **Q1-Rechtecks**: in der einen Diagonalrichtung durchhängend, in der anderen aufgewölbt — wie ein Reitsattel, ein Gebirgspass oder ein Kartoffelchip (Kapitel 11). Sie entsteht allein aus dem gemischten Glied $x \cdot y$, das die vier Eckwerte im Inneren verbindet; auf den Kanten selbst bleibt der Verlauf gerade. Genau diese Krümmung unterscheidet das Rechteck von den zwei ebenen Dreiecksplatten, in die man es zerschneiden könnte — und sie macht den bilinearen Wert unabhängig von einer willkürlichen Schnittdiagonale.

Schachbrettmuster — ein unphysikalischer Wechsel hoher und niedriger Elementwerte auf Netzskala. Bei einem rein lokalen Dichteupdate kann die Rückkopplung benachbarte Dreiecke gegeneinander treiben; ein stationäres buntes Muster ist dann noch kein belastbares Ergebnis. Kapitel 17 prüft und dämpft den Hochfrequenzanteil mit einem **nichtlokalen** Reizfilter.

Scheindichte (ρ, apparent density) — die Masse eines Knochenstücks bezogen auf sein **Gesamt**volumen (Material *und* Poren), in g/cm³. Sie steht am Ende der kalibrierten Kette CT-Wert → Aschedichte → Scheindichte und liefert über das Morgan-Gesetz den **Elastizitätsmodul** (Kapitel 16). Eine rohe Bildgraustufe ist noch keine Scheindichte.

Schleife — eine Anweisung „tu das Folgende mit jedem Eintrag der Reihe nach"; in Python die `for`-Schleife (Kapitel 2). Die häufigste Schleife des Buches bildet eine **laufende Summe** (`summe = summe + wert`) — genau das, was das mathematische Summenzeichen Σ auf Papier bedeutet.

Schubverzerrung (Schub, γ_{xy}) — die dritte der drei ebenen **Dehnungen**: nicht ein Strecken, sondern ein **Kippen** des rechten Winkels (Kapitel 14). Ein aufgemaltes Quadrat wird dabei zum Parallelogramm, ohne breiter oder höher zu werden — die senkrechten Linien neigen sich. γ_{xy} misst, um wie viel der ursprünglich rechte Winkel von 90° abweicht. Zusammen mit den beiden Streckungen ε_{xx} (in x) und ε_{yy} (in y) bildet sie das vollständige „Messprotokoll" der Verformung eines Materialstücks. Ihre zugehörige **Spannung** ist die Schubspannung τ_{xy}, die tangential an einer Schnittfläche zerrt. Bei der Wärmedehnung ist der Schub null — Wärme kippt keinen Winkel, sie bläht gleichmäßig auf.

Schwache Form — die Gestalt der Differentialgleichung *nach* der **partiellen Integration**, in der nur noch *einfache* Ableitungen der gesuchten Funktion vorkommen (Kapitel 7). „Schwach", weil sie weniger verlangt als die ursprüngliche („starke") Form: statt einer zweiten Ableitung an jedem Punkt nur ein gewichtetes Mittel erster Ableitungen. Gerade diese mildere Forderung macht sie mit dem eckigen **Streckenzug** verträglich, dessen

zweite Ableitung an den Knicken stirbt — die schwache Form ist der Trick, mit dem die FEM überhaupt erst rechnen kann.

Sekante — die Verbindungsgerade zweier Punkte einer Kurve; ihre Steigung ist der Differenzenquotient $\Delta T/\Delta x$ (Kapitel 5). Lässt man die beiden Punkte zusammenrücken, wird aus der Sekante die **Tangente**, aus ihrer Steigung die **Ableitung**. Der Name kommt von lateinisch *secare*, schneiden — die Sekante *schneidet* die Kurve in zwei Punkten.

Simulation — das Nachrechnen eines Stücks Wirklichkeit im Computer, bevor oder ohne dass es in echt geschieht; im Kern immer: zerlegen, lokal rechnen, zusammensetzen und lösen (Kapitel 1).

Spannung (σ) — die innere Kraft pro Querschnittsfläche in einem belasteten Bauteil, $\sigma = F/A$, in N/mm² (= MPa; Kapitel 13). Ein Minuszeichen heißt **Druck**, ein Plus **Zug**. Über das **Hookesche Gesetz** hängt sie an der **Dehnung** ($\sigma = E\,\varepsilon$), genauer an der Kernformel $\sigma = E\,(\varepsilon - \alpha\,\Delta T)$: Spannung entsteht nur aus dem **verhinderten** Teil der Dehnung, nicht aus der Wärme an sich. Nicht mit dem **E-Modul** (in GPa) verwechseln — $1\ \text{GPa} = 1000\ \text{N/mm}^2$. Sie ist das mechanische Gegenstück zum **Wärmestrom** (Analogie-Tafel, Kapitel 13); frei gelagert bleibt der erwärmte Träger spannungsfrei, beidseitig fest steht er unter $\sigma = -E\,\alpha\,\Delta T$.

Spongiosa (trabekulärer Knochen, Bälkchenknochen) — der schwammartige, poröse Innenknochen aus feinen **Trabekeln**; je nach **Scheindichte** rund 0,6 bis 7 GPa steif (Morgan-Gesetz). Gegenstück ist die **Kompakta** (Kapitel 16).

Stabilität — die Eigenschaft eines **Zeitschritt**verfahrens, dass sich kleine Fehler von Schritt zu Schritt *nicht* aufschaukeln (Kapitel 9). Sie ist eine Eigenschaft des *Verfahrens*, nicht der Physik: Der **explizite Euler** ist nur unterhalb einer scharfen Grenze $\Delta t_{\max} = h^2\rho c/(2\lambda)$ stabil — darüber **explodiert** die Lösung in wachsende Zickzack-Ausschläge. Der **implizite Euler** ist bei jedem Zeitschritt stabil. Wichtig: Stabilität ist nicht Genauigkeit — ein stabiler, aber grober impliziter Schritt bleibt beschränkt, verschmiert aber die Front.

Starrkörpermode — eine Bewegung, die ein Bauteil als Ganzes ausführt, ohne seine **Form** zu ändern — und die darum **keine Kraft** und keine **Spannung** erzeugt (Kapitel 14). In der Fläche gibt es drei: nach rechts schieben, nach oben schieben, drehen. Weil das Material sie nicht „spürt", hat die **Steifigkeitsmatrix** für sie kein Rückstellvermögen — ein Bauteil ohne ausreichende **Lager** ist deshalb nicht berechenbar, es **schwimmt** davon (der Gauß-Löser stolpert über eine Null, Gauß-Seidel driftet weiter). Es ist die 2D-Fassung des schwimmenden Trägers aus Kapitel 8 (dort schwamm das Wärmeproblem in *einer* Richtung, weil nur die Höhe des Profils frei war). Genug Lager sperren alle drei Moden und machen das System eindeutig; die

kleinstmögliche solche Lagerung heißt **statisch bestimmt** („gerade genug festhalten").

Stationär (eingeschwungen) — der Zustand, in dem sich zeitlich nichts mehr ändert: Die Temperatur an jeder Stelle bleibt stehen, weil an jedem Stück gleich viel Wärme herein- wie hinausströmt (die **Bilanz** geht auf, Kapitel 6). Gegenstück ist der **transiente** Zustand, in dem sich der Träger noch aufheizt (Kapitel 1) und ein Speicherterm hinzukommt (Kapitel 9). Vom lateinischen *stare*, stehen — im stationären Zustand „steht" das Profil.

Steifigkeit — die Zahl k, die sagt, wie stark sich ein Bauteil einer Verformung widersetzt: Eine steife Feder (großes k) gibt unter derselben Kraft weniger nach als eine weiche (Kapitel 3). Beim Träger wird aus der mechanischen Steifigkeit später die Wärmeleitfähigkeit — dieselbe Rolle im Gleichungssystem.

Steifigkeitsmatrix — die große Systemmatrix einer FEM-Rechnung, in der alle Element-Steifigkeiten zusammengesetzt (**assembliert**) sind; ihre Zeilen sind die Kraft-Gleichungen der Knoten (Kapitel 3). Der Name stammt aus dem Flugzeugbau der 1950er, wo das Verfahren *direct stiffness method* (Methode der direkten Steifigkeit) hieß (Turner u. a. 1956). Sie ist in aller Regel **dünn besetzt** (siehe Bandstruktur).

Steigung — wie stark sich eine Größe pro zurückgelegter Strecke ändert: beim Temperaturprofil der Temperaturunterschied geteilt durch den Ortsunterschied, $\Delta T / \Delta x$, in °C/m (Kapitel 5). Anschaulich das **Steigungsdreieck** (Höhe durch Breite). Für eine Gerade überall gleich und exakt; für eine krumme Kurve hängt sie vom Ort ab und wird dort zur **Ableitung**.

T

Tangente — die Gerade, die eine Kurve in einem Punkt nur *berührt*, statt sie zu schneiden; sie ist die Grenzlage der **Sekante**, wenn deren zwei Punkte zusammenrücken (Kapitel 5). Ihre Steigung ist die **Ableitung** an diesem Punkt. Der Name kommt von lateinisch *tangere*, berühren.

Testfunktion (auch *Gewichtsfunktion*, *Wichtungsfunktion*) — das Gewicht, mit dem die **Methode der gewichteten Reste** das **Residuum** mittelt: Jede Testfunktion „befühlt" den Fehlbetrag in ihrem Trägergebiet und erzwingt „Fehler im gewichteten Mittel = null" (Kapitel 7). Es gibt so viele Testfunktionen wie Knoten, darum so viele Gleichungen wie Unbekannte. Nimmt man als Testfunktionen dieselben **Hütchen** wie für den Ansatz, heißt das Verfahren **Galerkin** (Standardweg der FEM); andere Wahlen (Punktimpulse, Zellkästen) erzeugen die Finite-Differenzen- bzw. Finite-Volumen-Methode.

Thermische Kraft — die rechte Seite des Stab-Gleichungssystems in der Thermomechanik: das Kraftpaar $\pm E\,A\,\alpha\,\Delta T$, mit dem ein erwärmtes Element seine beiden Knoten auseinanderdrückt (Kapitel 13). Sie übersetzt die **Wärmedehnung** in eine Last — man hält das Element gedanklich auf Länge (dabei entsteht die Druckspannung $-E\,\alpha\,\Delta T$) und nimmt diese Haltekraft dann wieder weg. ΔT ist die mittlere Übertemperatur des Elements über der spannungsfreien Referenz. An inneren Knoten heben sich die thermischen Kräfte benachbarter Elemente teilweise auf; nur ihre **Differenz** bleibt — deshalb ist der gleichmäßig erwärmte *freie* Träger spannungsfrei. Das Gegenstück zur Wärmequelle auf der rechten Seite des Wärmeproblems.

Thomas-Algorithmus — ein besonders schneller Löser für **tridiagonale** Gleichungssysteme (nur Diagonale und ihre beiden Nachbarn tragen Zahlen). Er ist die **Gauß-Elimination**, die von vornherein weiß, dass außerhalb des schmalen Bandes nur Nullen stehen, und diese gar nicht erst anfasst: eine Vorwärtswelle räumt das Band leer, eine Rückwärtswelle erntet (2). Dadurch kostet er nur $\approx 8n$ statt $n^3/3$ Rechenschritte — der Bandlöser des 1D-Trägers ab Kapitel 7. Benannt nach dem britischen Physiker Llewellyn H. Thomas, der ihn 1949 beschrieb (Thomas 1949).

Totzone — das Intervall kleiner Reizabweichungen, in dem eine Remodeling-Regel die Dichte nicht verändert. In Kapitel 17 gilt $|\bar{S}/S_0 - 1| \le \delta$: Außerhalb folgt Abbau oder Aufbau, innerhalb ist die Antwort null. Die Totzone ist ein transparenter Modellparameter, keine direkt gemessene biologische Zeitschwelle.

Triangulierung — die Zerlegung einer Fläche in Dreiecke; das flächenhafte **Netz** der FEM (1). Sie ist der Bauplan aus **Knotenliste** und **Dreiecksliste**: welche Knoten wo liegen und welche drei ein Dreieck bilden. Weil sich benachbarte Dreiecke eine Kante samt ihrer beiden Eckknoten teilen, passen sie nahtlos zusammen (Kantenkompatibilität). Von Hand für kleine Netze, automatisch per **Delaunay-Triangulierung** für große.

Tridiagonal — die Gestalt einer Matrix, die nur auf der Hauptdiagonale und den beiden Nebendiagonalen direkt daneben Zahlen trägt, sonst überall null; das schmalste **Band** überhaupt (2). Die Systemmatrix einer Federkette und eines 1D-Trägers ist tridiagonal — genau die Form, die der **Thomas-Algorithmus** blitzschnell löst.

V

Validierung — die zweite der beiden Prüf-Fragen: „Löse ich die *richtigen* Gleichungen?" — bildet das Modell die *Wirklichkeit* ab? (Kapitel 12). Sie vergleicht die Rechnung mit **Messungen** an echten Bauteilen und deckt

Fehler auf, die keine noch so saubere Rechnung findet: ein weggelassener Effekt (Wärmeverlust an die Luft, temperaturabhängiges Material), eine falsche Annahme, ein zu einfaches Modell. Gegenstück zur **Verifikation**: Erst verifizieren (rechne ich richtig?), dann validieren (rechne ich das Richtige?). Dieses Buch verifiziert gründlich, validiert aber nicht — echte Messdaten sind sein Ausblick, nicht sein Stoff (Roache 1998).

Verifikation — die erste der beiden Prüf-Fragen: „Löse ich die Gleichungen *richtig*?" — rechnet das Programm die *gewählte* Mathematik korrekt, ohne Programmier-, Einheiten- oder **Randbedingungs**fehler? (Kapitel 12). Ihre Werkzeuge sind rein rechnerisch und brauchen kein Labor: analytische **Benchmarks** (der **Nulltest**, die Parabel, die Reihenlösung), die **Konvergenz**probe (Viertelungs-Rhythmus) und die Energie**bilanz**. Sie ist die wichtigste Berufsfähigkeit dieses Buchteils: **Bunt ist kein Beweis** — ein FEM-Programm liefert *immer* ein plausibles Farbbild, auch wenn es falsch rechnet. Gegenstück ist die **Validierung** (rechne ich das Richtige?). Die Trennung der beiden Begriffe geht auf Patrick Roache zurück (Roache 1998).

Voigt-Schreibweise — eine bloße **Sortierregel**, kein Konzept: Sie schreibt die drei ebenen **Dehnungen** als Spalte $(\varepsilon_{xx}, \varepsilon_{yy}, \gamma_{xy})$ und die drei **Spannungen** als Spalte $(\sigma_{xx}, \sigma_{yy}, \tau_{xy})$ — immer in dieser Reihenfolge (Kapitel 14). Der Nutzen ist rein praktisch: So wird aus dem **Hookeschen Gesetz** der Fläche eine handliche 3×3-**Materialtabelle**, mit der ein Programm rechnen kann. Sie ersetzt die umständlichere Tabellen-in-Tabellen-Schreibweise (den „Tensor"), die dieses Buch bewusst vermeidet. Benannt nach dem Physiker **Woldemar Voigt** (1850–1919), der die Notation in seiner Kristallphysik einführte.

W

Wärmeausdehnung (Wärmeausdehnungskoeffizient α) — die Materialzahl, um wie viel sich ein Stoff pro Grad Temperaturerhöhung dehnt: die freie Wärmedehnung ist $\varepsilon_{\text{therm}} = \alpha\,\Delta T$, mit α in $1/\text{K}$ (Kapitel 13). Für unseren Stahl $12 \cdot 10^{-6}$, zum Vergleich Aluminium $23 \cdot 10^{-6}$ und Fensterglas $9 \cdot 10^{-6}$ — Metalle dehnen sich deutlich stärker als Glas. Die Wärmedehnung entsteht **ohne Kraft**: Ein frei liegender Träger dehnt sich spannungsfrei. Dass Stahl und Beton fast dasselbe α haben, ist der Grund, warum Stahlbeton hält — und warum unser Träger und die Betondecke über ihm sich im Gleichtakt ausdehnen. Zusammen mit dem **E-Modul** bildet α das Produkt $E\,\alpha$, das die Einspann-Spannung $-E\,\alpha\,\Delta T$ bestimmt.

Wärmeleitfähigkeit (λ) — die Materialzahl im **Fourierschen Gesetz**, die sagt, wie gut ein Stoff Wärme leitet, in W/(m·K) (Kapitel 6). Die Spanne ist

riesig: Kupfer rund 400, Aluminium 200, unser Baustahl etwa 50, rostfreier Edelstahl nur 15, Holz 0,15, ruhende Luft nur 0,026 — Metalle leiten tausendfach besser als Dämmstoffe. Bei gleichem Temperaturgefälle treibt ein großes λ einen entsprechend großen **Wärmestrom**. Im Gleichungssystem der FEM übernimmt λ die Rolle, die bei der Federkette die **Steifigkeit** k spielte.

Wärmeleitungsgleichung — die **Differentialgleichung** der Wärmeleitung, in diesem Buch zuerst im stationären Fall $\frac{\mathrm{d}}{\mathrm{d}x}(\lambda \frac{\mathrm{d}T}{\mathrm{d}x}) = 0$ (Kapitel 6). Sie ist die Wärme-**Bilanz** „rein gleich raus", auf jeden Punkt heruntergeschrumpft, und sagt in Worten: Der **Wärmestrom** ist überall gleich. Ihre Lösung im homogenen Träger ist eine Gerade. Kapitel 9 fügt den Speicherterm (die Zeit) hinzu, Kapitel 7 löst sie mit der FEM.

Wärmestrom — die Wärme, die pro Sekunde durch eine Fläche strömt, in Watt (= Joule pro Sekunde); die **Wärmestromdichte** q ist derselbe Strom pro Quadratmeter Querschnitt, in W/m² (Kapitel 6). Er wird vom Temperaturgefälle getrieben (**Fouriersches Gesetz**): Durch unseren eingeschwungenen Träger fließen 140 W. Nicht zu verwechseln mit der *gespeicherten* Wärmemenge (Joule, Kapitel 5) — der Wärmestrom ist der Durchsatz, nicht der Vorrat.

Z

Wolffsches Gesetz — die klassische Beobachtung (Julius Wolff, 19. Jahrhundert), dass sich die innere Architektur des Knochens an die Belastung anpasst: Material entsteht, wo Kraft fließt, und schwindet, wo keine wirkt. Der moderne rechnerische Rahmen dafür ist der **Mechanostat**, die FEM-Umsetzung das **Remodeling** (Kapitel 17).

Zeitkonstante (τ) — die natürliche Zeitskala, auf der sich ein Vorgang abspielt; beim Träger $\tau = L^2 \rho c / \lambda$, für unseren Stahlträger rund 72 220 s ≈ 20 Stunden (Kapitel 9). Sie fällt aus der **Wärmeleitungsgleichung** selbst und sagt, wie lange das Aufheizen bis zum **stationären** Zustand ungefähr dauert. Das L^2 bedeutet: Ein doppelt so langer Träger braucht viermal so lange. Weil τ in Stunden liegt, zeigen alle Animationen des Buches **Zeitraffer**.

Zeitschritt (Δt) — der Sprung in der Zeit, mit dem ein transienter Vorgang Stück für Stück vorgerechnet wird (Kapitel 9). Aus dem Startzustand macht ein **Euler-Verfahren** Schritt um Schritt neue Zustände. Beim **expliziten** Euler ist Δt an die Maschenweite gefesselt (die **Stabilität**sgrenze $\propto h^2$); beim **impliziten** Euler ist er frei wählbar, nur die Genauigkeit setzt ihm eine Grenze. Kleiner Zeitschritt = genauer, aber mehr Schritte.

Zeltfunktion (Ansatz-/Formfunktion des P1-Dreiecks) — die flächenhafte Fortsetzung des **Hütchens** aus Kapitel 7: die Funktion, die über ihrem eigenen Knoten auf 1 steht und über jedes anliegende Dreieck geradlinig auf 0 an den Nachbarknoten abfällt (1). Setzt man die schrägen Dreiecksflächen aller Dreiecke zusammen, die an einem Knoten hängen, entsteht ein „Zelt". Als Ortsfunktion betrachtet, ist die Zeltfunktion eines Knotens gerade sein **Flächenanteil** (seine baryzentrische Koordinate). Weil jede Zeltfläche eben ist, hat sie über jedem Dreieck **konstante Steigungen** — daraus wird die 3×3-Elementmatrix des **P1-Dreiecks** gebaut.

Aluminium-Werkstoffdaten. 2024. *Streckgrenze von Aluminium und Aluminiumlegierungen*. Werkstoffkennwerte; reines Aluminium einige zehn N/mm², gaengige Konstruktionslegierungen (z. B. EN AW-6061 T6) rund 240–275 N/mm², hochfeste Luftfahrtlegierungen bis über 500 N/mm².

Argyris, John H. 1954--1955. „Energy Theorems and Structural Analysis". *Aircraft Engineering* 26–27 (1954--1955).

Bergmann, Georg, G. Deuretzbacher, M. Heller, u. a. 2001. „Hip contact forces and gait patterns from routine activities". *Journal of Biomechanics* 34 (7): 859–71. https://doi.org/10.1016/S0021-9290(01)00040-9.

Calvo-Gallego, José Luis, Peter Pivonka, José Manuel García-Aznar, und Javier Martínez-Reina. 2021. „A novel algorithm to resolve lack of convergence and checkerboard instability in bone adaptation simulations using non-local averaging". *International Journal for Numerical Methods in Biomedical Engineering* 37 (2): e3419. https://doi.org/10.1002/cnm.3419.

Carter, Dennis R., und Wilson C. Hayes. 1977. „The compressive behavior of bone as a two-phase porous structure". *The Journal of Bone and Joint Surgery* 59 (7): 954–62. https://doi.org/10.2106/00004623-197759070-00021.

CEN (Europäisches Komitee für Normung). 2005. *Eurocode 3: Bemessung und Konstruktion von Stahlbauten – Teil 1-2: Allgemeine Regeln – Tragwerksbemessung für den Brandfall*. EN 1993-1-2. CEN.

Clough, Ray W. 1960. „The Finite Element Method in Plane Stress Analysis". *Proceedings of the 2nd ASCE Conference on Electronic Computation* (Pittsburgh, Pennsylvania).

Delaunay, Boris. 1934. „Sur la sphère vide". *Bulletin de l'Académie des Sciences de l'URSS, Classe des sciences mathématiques et naturelles*, Nr. 6: 793–800.

Euler, Leonhard. 1768. *Institutiones calculi integralis*. Bd. 1. Impensis Academiae Imperialis Scientiarum.

Fourier, Joseph. 1822. *Théorie analytique de la chaleur*. Firmin Didot Père et Fils.

Frost, Harold M. 2003. „Bone's mechanostat: a 2003 update". *The Anatomical Record Part A* 275A (2): 1081–101. https://doi.org/10.1002/ar.a.10119.

Galerkin, Boris G. 1915. „Sterzhni i plastinki. Ryady v nekotorykh voprosakh uprugogo ravnovesiya sterzhney i plastinok". *Vestnik Inzhenerov i Tekhnikov* 19: 897–908.

Gustafson, Karl, und Takehisa Abe. 1998. „(Victor) Gustave Robin: 1855–1897". *The Mathematical Intelligencer* 20 (2): 47–53. https://doi.org/10.1007/BF03025298.

Huiskes, Rik, H. Weinans, H. J. Grootenboer, M. Dalstra, B. Fudala, und T. J. Slooff. 1987. „Adaptive bone-remodeling theory applied to prosthetic-design analysis". *Journal of Biomechanics* 20 (11–12): 1135–50. https://doi.org/10.1016/0021-9290(87)90030-3.

Kolditz, Olaf, Sebastian Bauer, Lars Bilke, Norbert Böttcher, Jens-Olaf Delfs, u. a. 2012. „OpenGeoSys: an open-source initiative for numerical simulation of thermo-hydro-mechanical/chemical (THM/C) processes in porous media". *Environmental Earth Sciences* 67 (2): 589–99. https://doi.org/10.1007/s12665-012-1546-x.

Möbius, August Ferdinand. 1827. *Der barycentrische Calcul*. Johann Ambrosius Barth.

Morgan, Elise F., Harun H. Bayraktar, und Tony M. Keaveny. 2003. „Trabecular bone modulus-density relationships depend on anatomic site". *Journal of Biomechanics* 36 (7): 897–904. https://doi.org/10.1016/S0021-9290(03)00071-X.

Poisson, Siméon Denis. 1829. „Mémoire sur l'équilibre et le mouvement des corps élastiques". *Mémoires de l'Académie Royale des Sciences de l'Institut de France* 8: 357–570.

Roache, Patrick J. 1998. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers.

Schileo, Enrico, Enrico Dall'Ara, Fulvia Taddei, u. a. 2008. „An accurate

estimation of bone density improves the accuracy of subject-specific finite element models“. *Journal of Biomechanics* 41 (11): 2483–91. https://doi.org/10.1016/j.jbiomech.2008.05.017.

Sridhar, Idapalapati. 2016. „Technical Problem Identification for the Failures of the Liberty Ships“. *Challenges* 7 (2): 20. https://doi.org/10.3390/challe7020020.

Thiem, Günther. 1906. *Hydrologische Methoden*. Gebhardt.

Thomas, Llewellyn H. 1949. *Elliptic Problems in Linear Difference Equations over a Network*. Watson Scientific Computing Laboratory, Columbia University.

Turner, M. J., R. W. Clough, H. C. Martin, und L. J. Topp. 1956. „Stiffness and Deflection Analysis of Complex Structures“. *Journal of the Aeronautical Sciences* 23 (9): 805–23. https://doi.org/10.2514/8.3664.

Usmani, A. S., J. M. Rotter, S. Lamont, A. M. Sanad, und M. Gillie. 2001. „Fundamental principles of structural behaviour under thermal effects“. *Fire Safety Journal* 36 (8): 721–44.

VDI / ingenieur.de. 2023. *Wenn Schienen 60 °C heiß werden: Wie hitzefest ist die Bahn?* Https://www.ingenieur.de/technik/fachbereiche/verkehr/wenn-schienen-60-c-heiss-werden-wie-hitzefest-ist-die-bahn/.

Weinans, H., R. Huiskes, und H. J. Grootenboer. 1992. „The behavior of adaptive bone-remodeling simulation models“. *Journal of Biomechanics* 25 (12): 1425–41. https://doi.org/10.1016/0021-9290(92)90056-7.

Young, Thomas. 1807. *A Course of Lectures on Natural Philosophy and the Mechanical Arts*. Joseph Johnson.