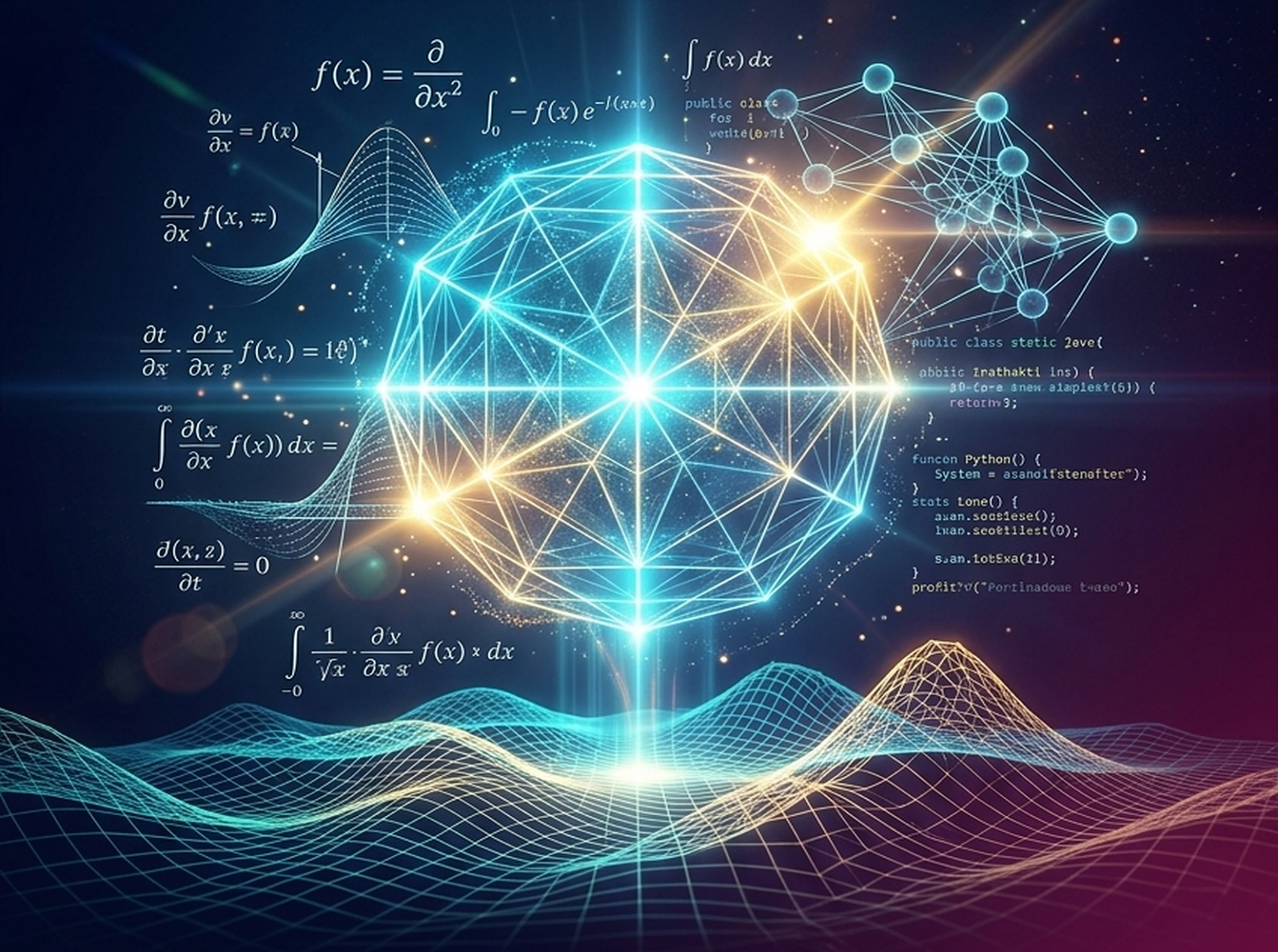
Metode Numerik

Dari Analisis Galat hingga Machine Learning, dengan Java dan Python



Irfan Subakti

Profitina
profitina.com

Metode Numerik

Dari Analisis Galat hingga Machine Learning, dengan Java dan Python

Irfan Subakti

Hak cipta © 2026 Irfan Subakti. Hak cipta dilindungi undang-undang.

Diterbitkan oleh Profitina · profitina.com

Edisi pertama, September 2026.

Setiap program dalam buku ini sudah dikompilasi dan dijalankan sebelum keluarannya dicetak. Kode sumber lengkapnya ada di folder Code yang menyertai buku ini. Program-program tersebut disediakan untuk keperluan belajar-mengajar; Irfan Subakti dan penerbit tidak bertanggung jawab atas penggunaannya di luar keperluan itu.

Daftar Isi

Kata Pengantar ..... **xxiv**
Untuk siapa buku ini ditulis xxiv
Satu gagasan, dua bahasa xxiv
Isi buku ini xxv
Cara membacanya xxvi
Tentang penulis xxvi
Ucapan terima kasih xxvii

BAB 1 Angka di Dalam Mesin: Galat, Floating Point, dan Meja Kerja Anda **1**
1.1 Nota yang Tidak Cocok 1
1.2 Apa Itu Metode Numerik 2
1.3 Meja Kerja Anda: Apa yang Dipasang dan Mengapa 3
1.4 Memasang di Windows 10 dan 11 4
1.4.1 JDK 4
1.4.2 Python 5
1.4.3 VS Code 5
1.5 Memasang di macOS 5
1.5.1 JDK 5
1.5.2 Python 6
1.5.3 VS Code 6
1.6 Memasang di Linux 6
1.7 VS Code, Lingkungan Virtual, NumPy, dan SciPy 7
1.8 Program Pertama Anda 8
1.8.1 Halo, dalam Java 8
1.8.2 Halo, dalam Python 9
1.8.3 Memeriksa seluruh meja kerja 9
1.9 Mengukur Galat 12
1.9.1 Galat sejati dan galat relatif 12
1.9.2 Galat hampiran dan aturan berhenti 13
1.9.3 Dua jenis galat 13
1.10 Bagaimana Mesin Menyimpan Bilangan 13
1.10.1 Pecahan biner 13
1.10.2 *Double* IEEE 754 14
1.10.3 Epsilon mesin dan batas pembulatan 14
1.11 Lima Percobaan dengan FloatLab 15
1.12 Galat Pembulatan dan Galat Pemotongan Bersama-sama 19
1.12.1 Deret Taylor dan pemotongan 19
1.12.2 Dilema ukuran langkah 22
1.13 Buku Kas *Laundry*: Menjumlahkan Uang dengan Aman 24
1.14 Membandingkan dengan Pustaka 27
1.15 Metode Numerik dalam Praktik: Profitina 29
1.16 Memakainya Sungguhan 29
1.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 30
1.18 Ringkasan Bab 30
1.19 Soal Latihan 31
1.20 Penyelesaian Soal Latihan 31
Daftar Pustaka 34

BAB 2 Mencari Akar I: Biseksi dan Regula Falsi **36**
2.1 Mas Dimas dan Ponsel "Bunga 1,67% Sebulan" 36

2.2 Gagasannya dengan Kata-Kata Sederhana 37
2.2.1 Uang hari ini dan uang nanti 37
2.2.2 Mengubah pertanyaan menjadi akar 37
2.2.3 Tabel tebakan 37
2.3 Biseksi 38
2.3.1 Mengapa pergantian tanda menjamin adanya akar 38
2.3.2 Metodenya 38
2.3.3 Berapa langkah: batas galat 39
2.3.4 Mencari kurung sejak awal 40
2.3.5 Rincian kecil yang penting 41
2.4 Regula Falsi 41
2.4.1 Talibusur 41
2.4.2 Mengapa regula falsi bisa merangkak 42
2.4.3 Modifikasi Illinois 43
2.5 Membangunnya dalam Java dan Python 43
2.6 Menjalankannya 48
2.6.1 Arti 2,92% sebulan dalam rupiah 50
2.7 Tempat Regula Falsi Merangkak 51
2.8 Mengujinya 52
2.9 Membandingkan dengan SciPy 54
2.10 Metode Numerik dalam Praktik: Profitina 55
2.11 Memakainya Sungguhan 56
2.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 57
2.13 Ringkasan Bab 57
2.14 Soal Latihan 57
2.15 Penyelesaian Soal Latihan 58
Daftar Pustaka 60

BAB 3 Mencari Akar II: Newton-Raphson, Secant, dan Iterasi Titik Tetap 61
3.1 Truk Air Pak Harun 61
3.2 Gagasannya dengan Kata-Kata Sederhana 62
3.2.1 Volume air dalam tangki yang berbaring 62
3.2.2 Ikuti garis singgungnya 63
3.3 Newton-Raphson 63
3.3.1 Penurunan 63
3.3.2 Mengapa begitu cepat: konvergensi kuadratik 65
3.3.3 Ketika Newton gagal 65
3.4 Metode Secant 66
3.5 Iterasi Titik Tetap 67
3.5.1 Menulis ulang $f(x) = 0$ sebagai $x = g(x)$ 67
3.5.2 Kapan dan seberapa cepat ia konvergen 67
3.6 Membangunnya dalam Java dan Python 68
3.7 Menjalankannya 74
3.8 Mengukur Orde Konvergensi 74
3.9 Ketika Newton Tersesat 78
3.10 Mengujinya 81
3.11 Membandingkan dengan SciPy 84
3.12 Memilih Metode 84
3.13 Metode Numerik dalam Praktik: Profitina 85
3.14 Memakainya Sungguhan 86
3.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 87
3.16 Ringkasan Bab 87
3.17 Soal Latihan 87
3.18 Penyelesaian Soal Latihan 88

Daftar Pustaka 90

BAB 4 Sistem Linear I: Eliminasi Gauss dan Pivoting 91

4.1 Pohon-Pohon Durian Bu Rahma 91
4.2 Gagasannya dengan Kata-Kata Sederhana 92
4.2.1 Tiga nota, tiga harga 92
4.2.2 Eliminasi dengan tangan 93
4.3 Eliminasi Gauss 93
4.3.1 Eliminasi maju 93
4.3.2 Substitusi mundur 94
4.3.3 Menghitung pekerjaannya 95
4.3.4 Ketika tidak ada penyelesaian tunggal 96
4.3.5 Banyak ruas kanan 97
4.4 Pivoting 97
4.4.1 Pivot nol 97
4.4.2 Pivot yang sangat kecil 97
4.4.3 *Partial pivoting* 98
4.5 Membangunnya dalam Java dan Python 98
4.6 Menjalankannya: Nota-Nota Belanja 103
4.7 Pupuk untuk Kebun Durian 105
4.8 Pengondisian: Residu Kecil Bukan Galat Kecil 109
4.8.1 Dua garis yang hampir sejajar 109
4.8.2 Bilangan kondisi 109
4.8.3 Residu dan galat 110
4.9 Mengapa *Pivoting* Penting, dalam Kode 113
4.10 Mengujinya 114
4.11 Membandingkan dengan NumPy 116
4.12 Metode Numerik dalam Praktik: Profitina 117
4.13 Memakainya Sungguhan 118
4.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 118
4.15 Ringkasan Bab 119
4.16 Soal Latihan 119
4.17 Penyelesaian Soal Latihan 120
Daftar Pustaka 121

BAB 5 Sistem Linear II: Jacobi, Gauss-Seidel, dan SOR 122

5.1 Wajan Pak Joni 122
5.2 Gagasannya dalam Kata-Kata Sederhana 123
5.2.1 Setiap titik adalah rata-rata tetangganya 123
5.2.2 Empat titik dengan tangan 123
5.2.3 Jacobi: semua memperbarui dari gambaran lama 124
5.2.4 Gauss-Seidel: langsung pakai angka terbaru 125
5.2.5 SOR: melangkahlah sedikit lebih jauh 125
5.3 Metodenya secara Tepat 126
5.3.1 Sistem linear umum 126
5.3.2 Pemisahan matriks 127
5.4 Kapan dan Seberapa Cepat Metode Ini Konvergen 128
5.4.1 Galat mengikuti iterasi yang sama 128
5.4.2 Uji yang bisa dilakukan dengan mata: dominansi diagonal 128
5.4.3 Masalah model: jari-jari spektral eksak 129
5.4.4 Berapa sapuan? 130
5.4.5 Kapan berhenti, dan apa arti sesungguhnya pergeseran kecil 130
5.4.6 Untuk apa repot: biaya satu sapuan 130
5.5 Membangunnya dalam Java dan Python 131
5.5.1 Sistem kecil 131

5.5.2 Wajan tanpa matriks 138
5.6 Menjalankannya 145
5.7 Memeriksa dengan Pustaka 150
5.8 Membuatnya Lebih Baik 151
5.9 Memakainya dengan Sungguh-Sungguh 152
5.10 Metode Numerik dalam Praktik: Profitina 152
5.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 153
5.12 Ringkasan Bab 153
5.13 Soal Latihan 154
5.14 Penyelesaian Soal Latihan 154
Daftar Pustaka 156

BAB 6 Dekomposisi Matriks: LU, Cholesky, dan QR 157
6.1 Pertanyaan Pupuk Pak Harun 157
6.2 Gagasannya dalam Kata-Kata Sederhana 158
6.2.1 Persamaannya 158
6.2.2 Eliminasi yang mengingat: mencari L dan U dengan tangan 158
6.2.3 Dua segitiga per blok 159
6.3 Dekomposisi LU 160
6.3.1 Dari mana faktor-faktor itu berasal 160
6.3.2 Biayanya 161
6.3.3 Determinan dan invers tanpa biaya tambahan 162
6.3.4 Seberapa teliti? 162
6.3.5 Apa yang salah tanpa *pivoting* 163
6.4 Cholesky: Separuh Kerja untuk Matriks Simetris Definit Positif 163
6.4.1 Hara keempat 163
6.4.2 Persamaan normal 163
6.4.3 Matriks simetris definit positif 164
6.4.4 Mengapa Cholesky tidak memerlukan *pivoting* 165
6.5 QR: Kuadrat Terkecil tanpa Mengkuadratkan 165
6.5.1 Harga ketelitian persamaan normal 165
6.5.2 Matriks ortogonal tidak meregangkan 166
6.5.3 Refleksi Householder 166
6.5.4 Pemeriksaan yang menghubungkan kedua metode 167
6.5.5 Gram-Schmidt, jalan lain menuju QR 167
6.6 Membangunnya dalam Java dan Python 167
6.6.1 LU dalam Python dan Java 167
6.6.2 Cholesky dan QR dalam Python dan Java 173
6.7 Menjalankannya 181
6.8 Memeriksa dengan Pustaka 183
6.9 Membuatnya Lebih Baik 185
6.10 Memakainya dengan Sungguh-Sungguh 185
6.11 Metode Numerik dalam Praktik: Profitina 186
6.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 186
6.13 Ringkasan Bab 187
6.14 Soal Latihan 188
6.15 Penyelesaian Soal Latihan 188
Daftar Pustaka 189

BAB 7 Interpolasi: Lagrange, Newton, dan Spline 191
7.1 Tongkat Ukur Pak Darto 191
7.2 Gagasannya dalam Kata-Kata Sederhana 192
7.2.1 Garis lurus di antara dua tanda 192
7.2.2 Kurva yang melalui lebih banyak tanda 193
7.3 Interpolasi Polinomial 193

7.3.1 Satu polinomial, dan hanya satu 193
7.3.2 Bentuk Lagrange 193
7.3.3 Bentuk Newton dan selisih terbagi 194
7.3.4 Evaluasi murah: perkalian bersarang 195
7.3.5 Seberapa salah polinomial interpolasi? 195
7.3.6 Fenomena Runge dan titik Chebyshev 196
7.3.7 Bila kemiringannya juga diketahui: interpolasi Hermite 197
7.4 Spline Kubik 197
7.4.1 Sepotong-sepotong, tetapi mulus 197
7.4.2 Menurunkan sistem tridiagonal 198
7.4.3 Dua syarat ujung 198
7.4.4 Algoritma Thomas 198
7.4.5 Seberapa teliti spline? 199
7.4.6 Satu bacaan spline dengan tangan 199
7.4.7 Membaca tabel secara terbalik 199
7.5 Membangunnya dalam Java dan Python 200
7.6 Menjalankannya 213
7.7 Memeriksa dengan Pustaka 216
7.8 Membuatnya Lebih Baik 217
7.9 Memakainya dengan Sungguh-Sungguh 218
7.10 Metode Numerik dalam Praktik: Profitina 218
7.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 219
7.12 Ringkasan Bab 219
7.13 Soal Latihan 220
7.14 Penyelesaian Soal Latihan 221
Daftar Pustaka 222

BAB 8 Pencocokan Kurva: Regresi Kuadrat Terkecil 223
8.1 Buku Catatan Mas Dimas 223
8.2 Gagasannya dalam Kata-Kata Sederhana 224
8.2.1 Meleset, dan mengapa kita mengkuadratkannya 224
8.2.2 Lima hari dengan tangan 225
8.3 Garis Lurus secara Tepat 225
8.3.1 Persamaan normal untuk garis 225
8.3.2 Dua fakta tentang residu 226
8.3.3 Seberapa baik pencocokannya? $R2$ dan galat baku 226
8.3.4 Seberapa yakin kemiringannya? 227
8.3.5 Meramalkan satu hari baru 227
8.3.6 Memperhatikan residu 227
8.4 Lebih Banyak Prediktor: Matriks Desain 228
8.4.1 Bensin dari jarak dan banyak perjalanan 228
8.4.2 Mengapa QR, bahkan di sini 228
8.4.3 Arti koefisien-koefisiennya 228
8.5 Kurva yang Linear dalam Koefisiennya 229
8.5.1 Polinomial 229
8.5.2 Kecepatan terbaik 229
8.5.3 Eksponensial melalui logaritma 229
8.6 *Overfitting*, dan Cara Memilih Model 230
8.7 Membangunnya dalam Java dan Python 231
8.8 Menjalankannya 242
8.9 Memeriksa dengan Pustaka 244
8.10 Membuatnya Lebih Baik 246
8.11 Memakainya dengan Sungguh-Sungguh 246
8.12 Metode Numerik dalam Praktik: Profitina 247
8.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 247

8.14 Ringkasan Bab 248
8.15 Soal Latihan 248
8.16 Penyelesaian Soal Latihan 249
Daftar Pustaka 250

BAB 9 Diferensiasi Numerik 252
9.1 Laporan Pengereman Mas Dimas 252
9.2 Gagasannya dalam Kata-Kata Sederhana 253
9.2.1 Kecepatan dari dua bacaan 253
9.2.2 Percepatan dari tiga bacaan 254
9.3 Menurunkan Rumus dari Teorema Taylor 254
9.3.1 Teorema Taylor 254
9.3.2 Beda maju dan beda mundur 255
9.3.3 Beda pusat 255
9.3.4 Turunan kedua 256
9.3.5 Rumus satu sisi untuk ujung-ujung 256
9.3.6 Rumus-rumus berdampingan 257
9.4 Dua Jenis Galat: Pemotongan dan Pembulatan 258
9.5 Ekstrapolasi Richardson 262
9.6 Membangun Penganalisis Perjalanan 265
9.7 Menyempurnakannya: Haluskan Dulu, Baru Turunkan 270
9.7.1 Cocokkan parabola, lalu turunkan parabolanya 270
9.7.2 Penganalisis yang dihaluskan 270
9.8 Memeriksa terhadap Pustaka, dan Pengujian 274
9.9 Metode Numerik dalam Praktik: Profitina 275
9.10 Memakainya Sungguhan 276
9.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 277
9.12 Ringkasan Bab 277
9.13 Soal Latihan 278
9.14 Penyelesaian Soal Latihan 278
Daftar Pustaka 281

BAB 10 Integrasi Numerik: Trapesium, Simpson, Romberg, dan Gauss 282
10.1 Seberapa Luas Kebun Itu? 282
10.2 Gagasannya dalam Kata-Kata Sederhana 283
10.3 Aturan Trapesium 284
10.3.1 Penurunan 284
10.3.2 Aturan gabungan 285
10.4 Aturan-Aturan Simpson 286
10.4.1 Aturan Simpson 1/3 286
10.4.2 Aturan Simpson 3/8 287
10.5 Membangun Kalkulator Kebun 287
10.6 Memeriksa Ordenya 291
10.7 Integrasi Romberg 294
10.8 Kuadratur Gauss: Memilih Tempat Mengukur 297
10.8.1 Gagasannya 297
10.8.2 Polinomial Legendre dan aturan umumnya 297
10.8.3 Programnya 298
10.9 Menyempurnakannya: Simpson Adaptif 302
10.10 Memeriksa terhadap Pustaka 306
10.11 Metode Numerik dalam Praktik: Profitina 307
10.12 Memakainya Sungguhan 307
10.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 308
10.14 Ringkasan Bab 308

10.15 Soal Latihan 309
10.16 Penyelesaian Soal Latihan 309
Daftar Pustaka 310

BAB 11 Persamaan Diferensial Biasa: dari Euler ke Runge-Kutta 312
11.1 Cukupkah Tandon Itu sampai Pukul Sepuluh? 312
11.2 Gagasannya dalam Kata-Kata Sederhana 313
11.3 Modelnya dan Solusi Eksaknya 313
11.3.1 Neraca massa 313
11.3.2 Solusi eksak sebagai pembanding 314
11.4 Metode Euler 315
11.4.1 Penurunan 315
11.4.2 Galat lokal dan galat global 315
11.4.3 Program pertama 315
11.5 Metode Runge–Kutta 318
11.5.1 Orde dua: Heun dan titik tengah 318
11.5.2 Metode orde empat klasik 319
11.5.3 Mengukur ordenya 320
11.6 Membangun Simulator Pagi Hari 323
11.7 Menyempurnakannya: Merencanakan Pompa 329
11.8 Sistem dan Kestabilan: Jok Sepeda Motor 332
11.8.1 Dari satu persamaan ke sebuah sistem 332
11.8.2 Mengapa Euler meledak: kestabilan 336
11.9 Memeriksa terhadap Pustaka 337
11.10 Metode Numerik dalam Praktik: Profitina 338
11.11 Memakainya Sungguhan 338
11.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 339
11.13 Ringkasan Bab 340
11.14 Soal Latihan 340
11.15 Penyelesaian Soal Latihan 341
Daftar Pustaka 343

BAB 12 Nilai Eigen: Metode Pangkat dan Kegunaannya 344
12.1 Tiga Binatu dalam Satu Kampung 344
12.2 Gagasannya dalam Kata-Kata Sederhana 345
12.2.1 Arah yang hanya diregangkan matriks 345
12.2.2 Mengapa perkalian berulang menemukan yang terbesar 345
12.3 Pasar Binatu sebagai Rantai Markov 346
12.3.1 Matriks perpindahan 346
12.3.2 Apa yang dikatakan nilai eigennya 346
12.3.3 Programnya 347
12.3.4 Keadaan keempat: mencuci di rumah 350
12.4 Metode Pangkat secara Umum 350
12.4.1 Algoritmanya dan hasil bagi Rayleigh 350
12.4.2 Iterasi invers dan iterasi invers bergeser 351
12.4.3 Iterasi invers dengan tangan 351
12.4.4 Ketika metode pangkat gagal 352
12.5 Bagaimana Sebuah Ruko Bergoyang 352
12.5.1 Modelnya 352
12.5.2 Tiga metode, tiga ragam 353
12.5.3 Atap yang lebih berat 358
12.5.4 Mengapa frekuensi itu penting: resonansi 359
12.6 PageRank: Vektor Eigen dari Web 359
12.6.1 Pembaca acak 359
12.6.2 Jaringan desa wisata 360

12.7 Memeriksa terhadap Pustaka 365
12.8 Metode Numerik dalam Praktik: Profitina 366
12.9 Memakainya Sungguhan 367
12.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 368
12.11 Ringkasan Bab 368
12.12 Soal Latihan 369
12.13 Penyelesaian Soal Latihan 369
Daftar Pustaka 371

BAB 13 Optimasi: Golden Section, Newton, dan Gradient Descent 372
13.1 Harga Mbak Rina 372
13.2 Gagasannya dengan Kata-Kata Sederhana 373
13.2.1 Dari mencari nol ke mencari puncak 373
13.2.2 Menjepit puncak di antara dua titik uji 373
13.2.3 Mengikuti kemiringan: Newton dan gradient descent 374
13.3 Metode-Metodenya, Diturunkan 374
13.3.1 Golden-section search 374
13.3.2 Metode Newton untuk maksimum 376
13.3.3 Banyak variabel: gradien dan Hessian 377
13.3.4 Gradient descent dan besar langkahnya 378
13.3.5 Backtracking: membiarkan metode memilih langkah 379
13.3.6 Metode Newton untuk banyak variabel 379
13.4 Program-Programnya 380
13.4.1 Golden section untuk satu harga 380
13.4.2 Metode Newton dengan dan tanpa pengaman 383
13.4.3 Dua harga: penurunan dan Newton 385
13.5 Menjalankan Program 391
13.6 Membandingkan dengan SciPy 394
13.7 Membuatnya Lebih Baik 395
13.8 Memakainya Sungguhan 396
13.9 Metode Numerik dalam Praktik: Profitina 397
13.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 397
13.11 Ringkasan Bab 398
13.12 Soal Latihan 398
13.13 Penyelesaian Soal Latihan 399
Daftar Pustaka 400

BAB 14 Metode Numerik untuk Machine Learning 401
14.1 Dua Pertanyaan Mas Dimas 401
14.2 Gagasannya dengan Kata-Kata Sederhana 402
14.3 Metode-Metodenya, Diturunkan 403
14.3.1 Kuadrat terkecil: persamaan normal atau QR 403
14.3.2 Gradient descent dan skala fitur 404
14.3.3 Mini-batch, momentum, dan Adam 405
14.3.4 Regresi logistik dan metode Newton 405
14.3.5 Rumus yang tidak overflow 407
14.3.6 Memeriksa gradien, dan lebar sebuah float 407
14.4 Program-Programnya 407
14.4.1 Blok-blok bangunan 408
14.4.2 Mencocokkan waktu perjalanan 415
14.4.3 Melatih dengan metode gradien 417
14.4.4 Meramal pembatalan 422
14.5 Menjalankan Program 428
14.6 Membandingkan dengan NumPy dan SciPy 431
14.7 Membuatnya Lebih Baik 433

14.8 Memakainya Sungguhan 433
14.9 Metode Numerik dalam Praktik: Profitina 434
14.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 434
14.11 Ringkasan Bab 435
14.12 Soal Latihan 435
14.13 Penyelesaian Soal Latihan 436
Daftar Pustaka 437

BAB 15 Komputasi Numerik Cepat dan Paralel 438
15.1 Satu Jam untuk Satu Gambar 438
15.2 Gagasannya dengan Kata-Kata Sederhana 439
15.2.1 Ke mana waktu habis 439
15.2.2 Mengerjakan beberapa hal sekaligus 439
15.2.3 Jumlah yang sama, jawaban yang berbeda 439
15.3 Metode-Metodenya, Diturunkan 440
15.3.1 Model biaya: operasi dan memori 440
15.3.2 Vektorisasi 440
15.3.3 Percepatan paralel dan batas-batasnya 441
15.3.4 Mengerjakan lebih sedikit: banyaknya sapuan 442
15.3.5 Jumlah yang teliti dan dapat diulang 443
15.4 Program-Programnya 444
15.4.1 Penjumlahan 444
15.4.2 Wajan dengan gelung 447
15.4.3 Tervektorisasi, dan dibagi ke beberapa thread 448
15.4.4 Sapuan yang lebih sedikit: SOR merah-hitam 451
15.5 Menjalankan Program 454
15.6 Membandingkan dengan NumPy dan SciPy 456
15.7 Membuatnya Lebih Baik 457
15.8 Memakainya Sungguhan 458
15.9 Metode Numerik dalam Praktik: Profitina 459
15.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 459
15.11 Ringkasan Bab 460
15.12 Soal Latihan 460
15.13 Penyelesaian Soal Latihan 460
Daftar Pustaka 462

BAB 16 Proyek Akhir: Menyimulasikan Sistem Nyata dari Awal sampai Akhir 463
16.1 Musim Kemarau Pak Hadi 463
16.2 Gagasannya dengan Kata-Kata Sederhana 464
16.3 Modelnya, Bagian demi Bagian 465
16.3.1 Tandon: sebuah tabel dan dua spline 465
16.3.2 Pompa: kurva hasil pencocokan dan titik kerja 466
16.3.3 Cuaca: rantai Markov dan tabel kuantil 466
16.3.4 Pohon dan sumur sepanjang musim 467
16.3.5 Jalur tetes: persamaan diferensial kecil 467
16.3.6 Satu hari, dan satu musim 468
16.3.7 Banyak musim, secara dapat diulang 468
16.3.8 Keputusan, dan rumus di baliknya 469
16.4 Program-Programnya 469
16.4.1 Kotak perkakas 469
16.4.2 Model kebun 481
16.4.3 Memeriksa bagian-bagian 489
16.4.4 Memeriksa model tetes terhadap kebun 491
16.4.5 Banyak musim secara paralel 493
16.4.6 Memilih tandon 496

16.5 Menjalankan Simulasi 498
16.5.1 Bagian-bagiannya 499
16.5.2 Model tetes terhadap sensor 499
16.5.3 Empat ratus musim 500
16.5.4 Tandon terbaik 501
16.6 Membandingkan dengan SciPy 502
16.7 Membuatnya Lebih Baik 503
16.8 Memakainya Sungguhan 504
16.9 Metode Numerik dalam Praktik: Profitina 504
16.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya 505
16.11 Ringkasan Bab 505
16.12 Soal Latihan 506
16.13 Penyelesaian Soal Latihan 506
Daftar Pustaka 507

Daftar Gambar

Gambar 1.1 Bagaimana program Java dan program Python dijalankan. Java dikompilasi sekali oleh `javac` menjadi *bytecode* yang dijalankan Java Virtual Machine; Python dibaca dan dijalankan interpreter setiap kali. 4

Gambar 1.2 Bagian-bagian Visual Studio Code yang dipakai buku ini: (1) Activity Bar, (2) Explorer, (3) editor, (4) tombol Run, (5) terminal terpadu, (6) Status Bar. 7

Gambar 1.3 Ke-64 bit *double* terdekat dengan 0,1: tanda 0, eksponen 01111111011 (1019, berarti $2-4$), dan 52 bit pecahan tempat pola 1001 berulang sampai dipotong dan dibulatkan. 14

Gambar 1.4 Galat selisih maju x di $x = 2$ terhadap langkah h: galat pemotongan turun bersama h, galat pembulatan tumbuh sebanding $1/h$, dan totalnya paling kecil di dekat $h = 10^{-8}$. 24

Gambar 2.1 Fungsi pinjaman $f(r)$ dari Persamaan (2.3) dan kurung setelah setiap langkah biseksi, mulai dari $[1\%, 5\%]$: setiap langkah menyimpan separuh yang memuat pergantian tanda (titik jingga). 39

Gambar 2.2 Galat $ck - r*$ ketiga metode pengurung pada fungsi pinjaman, dilanjutkan melewati toleransi sampai galatnya mencapai batas presisi ganda. 50

Gambar 3.1 Tangki truk air dilihat dari ujungnya (kiri): air mengisi tembereng lingkaran dengan kedalaman h dan lebar permukaan w. Volume Persamaan (3.1) sebagai fungsi kedalaman (kanan) berbentuk S, paling curam pada setengah kedalaman. 62

Gambar 3.2 Metode Newton untuk tanda 1.000 liter: dari tebakan kotak $x0$ garis singgung sedikit melampaui ke $x1$, dan garis singgung kedua mendarat hampir tepat di akar $x2$. 64

Gambar 3.3 Galat empat metode yang menghitung 2: biseksi dan iterasi titik tetap turun sepanjang garis lurus (konvergensi linear), kurva secant dan Newton melengkung ke bawah makin curam (orde 1,618 dan 2). Sebuah garis berhenti ketika iterasinya sama dengan *double* terdekat dengan 2. 78

Gambar 4.1 Eliminasi Gauss pada empat persamaan: di setiap tahap pivot (jingga) membersihkan kolomnya di bawahnya dan hanya entri berarsir yang berubah; substitusi mundur kemudian mengubah kolom kanan menjadi penyelesaiannya. 94

Gambar 4.2 Sistem berkondisi baik (kiri) dan sistem berkondisi buruk (kanan): bila garis-garisnya hampir sejajar, perubahan 0,01 pada data menggeser penyelesaian dari $(1,1)$ ke $(0,2)$. 109

Gambar 5.1 Wajan paling kasar: empat suhu titik dalam yang belum diketahui, masing-masing rata-rata keempat tetangganya; dua tetangga selalu berupa tepi yang suhunya diketahui. 124

Gambar 5.2 Galat terbesar wajan 20×20 setelah setiap sapuan, terhadap solusi acuan. Garis lurus pada skala ini berarti faktor tetap per sapuan: $\rho J = 0.989$, $\rho GS = 0.978$, dan sekitar 0.74 untuk SOR dengan $\omega = 1.741$. 148

Gambar 5.3 Banyak sapuan yang diperlukan SOR pada wajan 20×20 untuk faktor relaksasi dari 1 sampai 1.95. Titik terendahnya berada pada nilai Young $\omega\text{opt} = 1.741$. 149

Gambar 5.4 Suhu tunak wajan pada kisi 40×40 (kompor di kiri, tepi depan yang hangat di bawah), dengan garis 140 °C dan 170 °C yang membatasi zona martabak. 149

Gambar 6.1 Faktorkan sekali, selesaikan berkali-kali: matriks kadar difaktorkan sebagai $PA = LU$ sekali, dan rekomendasi setiap blok hanya memerlukan satu substitusi maju dan satu substitusi mundur. 160

Gambar 6.2 Banyak perkalian untuk K ruas kanan dengan matriks 100×100: eliminasi setiap kali tumbuh seperti $Kn3/3$, faktorisasi sekali seperti $n3/3 + Kn2$. 162

Gambar 6.3 Campuran kuadrat terkecil: Ax adalah titik ruang kolom yang paling dekat ke target b, dan melesetnya $r = b - Ax$ tegak lurus terhadap setiap kolom. 164

Gambar 6.4 Galat terbesar solusi kuadrat terkecil untuk kolom polinomial derajat d pada 20 titik: persamaan normal kehilangan sekitar dua kali lebih banyak digit daripada QR Householder, dan mulai derajat 12 Cholesky gagal. 183

Gambar 7.1 Tangki rebah di kios bahan bakar: tongkat ukur mengukur kedalaman h, dan volumenya adalah luas tembereng lingkaran dikalikan panjang tangki. 192

Gambar 7.2 Fungsi Runge dan interpolan derajat 10-nya: melalui 11 titik berjarak sama polinomialnya berayun liar di dekat ujung; melalui 11 titik Chebyshev ia mengikuti fungsinya dengan rapat. 196

Gambar 7.3 Galat interpolasi linear sepotong-sepotong dan polinomial derajat 6 dari tanda-tanda 20 cm, diukur terhadap rumus eksak tangki. 214
Gambar 7.4 Galat spline natural, spline terjepit (kemiringan ujung 0), dan polinomial derajat 6 melalui tanda-tanda 20 cm. Di dekat ujung, tempat kelengkungan tangki tak hingga, spline paling menderita. 215
Gambar 8.1 Empat puluh hari dalam buku catatan Mas Dimas: bensin terpakai terhadap jarak tempuh, garis kuadrat terkecil, dan residunya, yaitu meleset tegak yang dibuat sekecil mungkin oleh garis itu dalam jumlah kuadratnya. 224
Gambar 8.2 RMSE pencocokan polinomial derajat 1 sampai 8 pada 24 hari pencocokan dan pada 12 hari yang disisihkan. Galat pencocokan hanya turun; galat hari yang disisihkan paling kecil pada derajat 2 dan tumbuh tajam mulai derajat 5. 231
Gambar 8.3 Efisiensi bensin terhadap kecepatan rata-rata pada 36 hari, dengan pencocokan derajat 2 (puncaknya pada 45.8 km/jam ditandai) dan pencocokan derajat 8, yang mengikuti hari-hari tertentu alih-alih kecenderungannya. 244
Gambar 8.4 Kesehatan baterai ponsel Mas Dimas setiap dua bulan, pencocokan eksponensial melalui logaritma, dan tingkat 80% yang dicapainya setelah sekitar 23.4 bulan. 244
Gambar 9.1 Galat beda maju dan beda pusat dari x di $x = 4$ terhadap langkah h (kedua sumbu logaritmik, h mengecil ke kanan). Galat pemotongan turun seperti h atau $h2$; galat pembulatan naik seperti ε/h. 261
Gambar 9.2 Kecepatan perjalanan Mas Dimas (atas) dan percepatannya (bawah) dari rumus tiga titik biasa dan dari rumus kuadrat terkecil lima titik; hanya kurva yang dihaluskan yang memisahkan rem mendadak sungguhan dari alarm palsu. 274
Gambar 10.1 Kebun di antara jalan desa yang lurus dan tepi sungai, dengan tujuh belas jarak tegak yang diukur setiap 10 m; pias yang disorot adalah salah satu dari enam belas trapesium dalam aturan trapesium. 283
Gambar 10.2 Galat terhadap banyaknya nilai fungsi untuk $01dx/(1 + x) = \ln 2$: trapesium (orde 2), Simpson (orde 4), Romberg, dan Gauss–Legendre. 301
Gambar 11.1 Tandon yang mengosong: solusi eksak, metode Euler dengan langkah 10 menit (setiap langkah mengikuti garis singgung di awalnya dan mengeluarkan air sedikit terlalu banyak), dan metode Runge–Kutta klasik dengan langkah yang sama. 318
Gambar 11.2 Ketinggian air pada dua pagi: dengan pompa dibiarkan menyala sampai 07.30 tandon meluap selama tiga perempat jam; dengan pompa dimatikan pukul 06.42, seperti direncanakan dengan biseksi, ketinggiannya memuncak di 98 cm dan tidak pernah meluap. 331
Gambar 11.3 Jok dengan peredam kejut yang aus setelah polisi tidur: RK4 memperlihatkan ayunan yang meredam, sedangkan metode Euler dengan langkah yang sama membuatnya membesar, sebuah ketidakstabilan metode, bukan ketidakstabilan sepeda motornya. 336
Gambar 12.1 Pangsa ketiga binatu bulan demi bulan dari 40%, 35%, dan 25%; garis titik-titik adalah pangsa stasioner, vektor eigen matriks perpindahan untuk nilai eigen 1. 349
Gambar 12.2 Ketiga bentuk ragam model ruko: simpangan ke samping setiap lantai (tanah di 0), diskalakan agar atap bergerak satu satuan, beserta frekuensi alami setiap ragam. 358
Gambar 12.3 Jaringan desa wisata: panah adalah tautan, luas lingkaran sebanding dengan PageRank, dan pemandu air terjun (kuning tua) adalah halaman menggantung tanpa tautan keluar. 364
Gambar 13.1 Laba harian Mbak Rina dan selang *golden section* pada langkah 0 sampai 6: setiap batang panjangnya 0,618 kali batang di atasnya, dan semuanya memuat harga terbaik. 392
Gambar 13.2 Kontur laba harian (ribu rupiah) terhadap kedua harga, beserta lintasan *gradient descent* (langkah 0,008), langkah tetap di atas batas kestabilan (0,012), dan metode Newton, semuanya mulai dari (3, 1). 394
Gambar 14.1 Rugi di atas optimum kuadrat terkecil selama pelatihan: *gradient descent* batch penuh pada masukan mentah dan terstandardisasi, serta SGD *mini-batch*, *momentum*, dan Adam pada masukan terstandardisasi (kedua sumbu logaritmik). 430
Gambar 14.2 Peluang sebuah pesanan dibatalkan, sebagaimana dipelajari regresi logistik, terhadap jarak jemput untuk empat kombinasi hujan dan jam sibuk; kotak menunjukkan proporsi teramati per setengah kilometer atas semua kondisi. 431
Gambar 15.1 Waktu untuk menyelesaikan wajan 400×400 pada mesin uji dua inti: Jacobi (diperkirakan dari waktu terukur per sapuan dan banyaknya sapuan yang diperlukan), SOR merah-hitam, dan pemecah langsung *sparse* dari SciPy (terukur). Skalanya logaritmik. 456

Gambar 15.2 Suhu tunak wajan 40 cm pada grid 1 mm, dengan tepi kiri 200, kanan 60, atas 80, dan bawah 150 derajat Celsius; garis putih adalah isoterm. 456
Gambar 16.1 Sistem air kebun sebagai simulasi: setiap bagian dihitung dengan metode dari bab-bab sebelumnya, bagian-bagian itu bertemu di tandon, dan banyak musim simulasi menjadi masukan bagi pilihan ukuran tandon. 464
Gambar 16.2 Satu musim kemarau simulasi (cuaca yang sama) dengan tandon 13.000 liter dan 30.000 liter: air di tandon setelah penyiraman pagi, hari-hari ketika pohon mendapat kurang dari jatahnya (tanda silang), dan hujan harian (batang, sumbu kanan). 501
Gambar 16.3 Biaya tahunan terhadap kapasitas tandon dari 200 musim simulasi per ukuran: biaya tandon, nilai air yang kurang, dan totalnya, dengan kapasitas-kapasitas yang dicoba *golden-section search* dan ukuran yang dipilih. 502

Daftar Tabel

Tabel 1.1 Alat-alat meja kerja dan versi yang dipakai di seluruh buku. 3
Tabel 1.2 Dua format IEEE 754 yang dipakai dalam buku ini. 15
Tabel 2.1 Fungsi pinjaman Persamaan (2.3) pada lima suku bunga bulanan: tandanya berganti antara 2% dan 3%. 38
Tabel 2.2 Utang Dimas bulan demi bulan pada bunga efektif: bunganya turun ketika utangnya turun, dan angsuran kedua belas melunasi utang dengan tepat. 51
Tabel 3.1 Pencari akar Bab 2 dan 3 dibandingkan: apa yang diperlukan masing-masing, seberapa cepat konvergen di dekat akar sederhana, dan apakah dijamin berhasil. 85
Tabel 5.1 Banyak sapuan yang diperlukan agar pergeseran terbesar pada wajan $n \times n$ turun di bawah $10 - 6$. Gauss-Seidel memerlukan sekitar separuh sapuan Jacobi, sesuai ramalan $\rho GS = \rho J2$; SOR optimal hanya tumbuh linear terhadap n. 148
Tabel 6.1 Gram setiap hara dalam 100 g setiap pupuk, dan rekomendasi untuk satu pohon muda. 158
Tabel 6.2 Ketiga dekomposisi bab ini, apa yang diperlukannya, berapa biayanya, dan untuk apa. 167
Tabel 7.1 Tabel selisih terbagi untuk empat tanda pertama; entri teratas setiap kolom adalah koefisien bentuk Newton. 194
Tabel 8.1 Garis kuadrat terkecil melalui lima hari pertama, dengan tangan: simpangan dari rata-rata, kuadratnya, dan hasil kalinya. 225
Tabel 9.1 Lima bacaan jarak dari perjalanan Mas Dimas, satu per detik, di sekitar pengereman keras. 253
Tabel 9.2 Rumus beda hingga untuk turunan pertama dan kedua, beserta banyaknya nilai fungsi yang diperlukan dan galat pemotongannya (ξ tidak ditulis). 258
Tabel 13.1 Pengoptimal dalam bab ini: apa yang diperlukan, berapa biaya satu langkah, dan seberapa cepat konvergensinya. 380
Tabel 16.1 Tempat metode setiap bab bekerja dalam simulasi kebun. 465

Daftar Kode

Kode 1.1 `Hello.java`: program Java pertama, yang juga melaporkan versi Java dan sistem operasi. 8
Kode 1.2 `hello.py`: program Python pertama. 9
Kode 1.3 `check_setup.py`: memeriksa Python, lingkungan virtual, NumPy, SciPy, dan Java. 9
Kode 1.4 `CheckSetup.java`: memeriksa versi Java dan mencari Python. 11
Kode 1.5 `float_lab.py`: lima percobaan dengan bilangan presisi ganda. 15
Kode 1.6 `FloatLab.java`: percobaan yang sama dalam Java. 16
Kode 1.7 `exp_series.py`: *ex* dari deret Taylor-nya dengan aturan berhenti Persamaan (1.4). 19

Kode 1.8 `ExpSeries.java`: deret yang sama dalam Java. 20
Kode 1.9 `diff_step.py`: selisih maju x dengan langkah yang makin kecil. 22
Kode 1.10 `DiffStep.java`: percobaan yang sama dalam Java. 23
Kode 1.11 `laundry_ledger.py`: nota *laundry* setahun dijumlahkan sebagai bilangan bulat, sebagai *double*, sebagai *float*, dan sebagai *float* dengan kompensasi Kahan. 25
Kode 1.12 `LaundryLedger.java`: buku kas yang sama dalam Java. 25
Kode 1.13 `compare_library.py`: epsilon mesin, *double* berikutnya, float32, dan tiga cara menjumlahkan, dengan NumPy 2.x. 27
Kode 1.14 `CompareLibrary.java`: fakta yang sama dari pustaka standar Java. 28
Kode 1.15 `harmonic32.py`: jumlah harmonik dalam presisi tunggal, maju dan mundur. 32
Kode 1.16 `Harmonic32.java`: hal yang sama dalam Java. 33
Kode 2.1 `roots.py`: biseksi, regula falsi, dan varian Illinois, dengan tabel iterasi opsional. 43
Kode 2.2 `Roots.java`: ketiga metode yang sama dalam Java, mengembalikan *record* `Result` berisi akar dan banyaknya iterasi. 45
Kode 2.3 `loan_rate.py`: suku bunga bulanan tersembunyi sebuah rencana cicilan dengan tiga metode. 46
Kode 2.4 `LoanRate.java`: program yang sama dalam Java. 47
Kode 2.5 `stagnation.py`: ketiga metode pada $x10 - 1$ 51
Kode 2.6 `Stagnation.java`: perbandingan yang sama dalam Java. 51
Kode 2.7 `test_roots.py`: uji unit metode-metode pengurung. 52
Kode 2.8 `RootsTest.java`: pemeriksaan yang sama dalam Java. 53
Kode 2.9 `compare_scipy.py`: suku bunga pinjaman dengan `bisect`, `brentq`, dan TOMS 748 dari SciPy. 54
Kode 2.10 `sphere_tank.py`: kedalaman air di tangki bola untuk volume-volume tertentu. 59
Kode 2.11 `SphereTank.java`: hal yang sama dalam Java. 59
Kode 3.1 `roots2.py`: Newton-Raphson, secant, dan iterasi titik tetap. 68
Kode 3.2 `Roots2.java`: ketiga metode yang sama dalam Java. 69
Kode 3.3 `dipstick.py`: tanda-tanda kedalaman pada tangki silinder yang berbaring dengan Newton-Raphson dan secant. 71
Kode 3.4 `Dipstick.java`: program yang sama dalam Java. 72
Kode 3.5 `convergence.py`: 2 dengan tiga metode, beserta rasio dan orde konvergensi yang terukur. 75
Kode 3.6 `Convergence.java`: pengukuran yang sama dalam Java. 75
Kode 3.7 `failures.py`: divergensi, siklus, garis singgung datar, dan titik awal yang baik. 78
Kode 3.8 `Failures.java`: keempat kasus yang sama dalam Java. 79
Kode 3.9 `test_roots2.py`: uji unit metode-metode terbuka dan tongkat ukur. 81
Kode 3.10 `Roots2Test.java`: pemeriksaan yang sama dalam Java. 82
Kode 3.11 `compare_scipy.py`: tanda 3.000 liter dengan `newton` (dengan dan tanpa turunan) dan `brentq` dari SciPy. 84
Kode 3.12 `loan_secant.py`: suku bunga pinjaman Dimas dengan metode secant. 88
Kode 3.13 `LoanSecant.java`: hal yang sama dalam Java. 89
Kode 4.1 `op_count.py`: flop eliminasi Gauss yang dihitung terhadap rumus Persamaan (4.6). 95
Kode 4.2 `OpCount.java`: penghitungan yang sama dalam Java. 95
Kode 4.3 `gauss.py`: eliminasi Gauss dengan *partial pivoting* opsional, substitusi mundur, residu, dan pembaca data. 98
Kode 4.4 `Gauss.java`: modul yang sama dalam Java. 100
Kode 4.5 `receipts.py`: tiga nota, tiga harga, dengan setiap langkah eliminasi dicetak. 104
Kode 4.6 `Receipts.java`: program yang sama dalam Java. 104
Kode 4.7 `fertilizer.py`: gram setiap pupuk per pohon, dan kilogram untuk seluruh kebun. 105
Kode 4.8 `Fertilizer.java`: program yang sama dalam Java. 106
Kode 4.9 `conditioning.py`: matriks Hilbert dengan ukuran yang makin besar: bilangan kondisi, residu, dan galat. 110
Kode 4.10 `Conditioning.java`: percobaan yang sama dalam Java. 111

Kode 4.11 `pivot_demo.py`: pivot nol dan pivot yang sangat kecil, dengan dan tanpa *partial pivoting*.....113
Kode 4.12 `PivotDemo.java`: peragaan yang sama dalam Java. 113
Kode 4.13 `test_gauss.py`: uji unit eliminasi Gauss. 114
Kode 4.14 `GaussTest.java`: pemeriksaan yang sama dalam Java. 115
Kode 4.15 `compare_numpy.py`: sistem pupuk dan matriks Hilbert dengan NumPy dan SciPy. 117
Kode 5.1 `iterative_solvers.py`: sapuan Jacobi, Gauss-Seidel, dan SOR untuk sistem padat kecil, dengan uji dominansi diagonal dan pemeriksaan residu. 131
Kode 5.2 `IterativeSolvers.java`: ketiga metode yang sama dalam Java. 134
Kode 5.3 `griddle.py`: wajan $n \times n$ diselesaikan dengan sapuan Jacobi, Gauss-Seidel, dan SOR pada kisi berbingkai, lengkap dengan banyak sapuan, galat sebenarnya, pemindaian ω, dan peta suhu. 138
Kode 5.4 `Griddle.java`: pemecah wajan dalam Java.....141
Kode 5.5 `compare_scipy.py`: wajan diperiksa dengan pemecah langsung jarang SciPy dan metode gradien konjugatnya. 150
Kode 6.1 `lu_blend.py`: dekomposisi LU dengan *pivoting* parsial, determinan, dan campuran untuk setiap blok dari satu faktorisasi.....168
Kode 6.2 `LuBlend.java`: program faktorkan-sekali, selesaikan-berkali-kali yang sama dalam Java. 169
Kode 6.3 `least_blend.py`: kuadrat terkecil dengan persamaan normal plus Cholesky dan dengan QR Householder, untuk kebun dan untuk kolom polinomial. 173
Kode 6.4 `LeastBlend.java`: kedua metode kuadrat terkecil yang sama dalam Java.....176
Kode 6.5 `compare_scipy.py`: faktor dan solusi kita dibandingkan dengan `lu`, `lu_factor`/`lu_solve`, `cho_factor`/`cho_solve` dari SciPy serta `lstsq` dan `qr` dari NumPy. 184
Kode 7.1 `interp.py`: interpolasi linear, Lagrange, dan Newton atas tabel tongkat ukur, galatnya terhadap rumus eksak tangki, dan percobaan Runge dengan titik berjarak sama dan titik Chebyshev.....200
Kode 7.2 `Interp.java`: program interpolasi yang sama dalam Java.....203
Kode 7.3 `spline.py`: spline kubik natural dan terjepit melalui sistem tridiagonal kelengkungan dan algoritma Thomas. 208
Kode 7.4 `Spline.java`: spline kubik dalam Java. 209
Kode 7.5 `compare_scipy.py`: interpolan kita dibandingkan dengan `numpy.interp`, `BarycentricInterpolator`, dan `CubicSpline`.....216
Kode 8.1 `lsq.py`: membaca CSV, kuadrat terkecil dengan QR Householder, ramalan, dan $R2$, dipakai bersama oleh program-program bab ini. 232
Kode 8.2 `Lsq.java`: pembantu bersama dalam Java. 233
Kode 8.3 `line_fit.py`: garis kuadrat terkecil dengan $R2$, galat baku, dan ketidakpastian kemiringan; dua prediktor dengan QR; kurva baterai eksponensial. 236
Kode 8.4 `LineFit.java`: pencocokan yang sama dalam Java. 237
Kode 8.5 `poly_fit.py`: pencocokan polinomial derajat 1 sampai 8 yang dinilai pada hari yang disisihkan, dan kecepatan jelajah terbaik dari parabola. 239
Kode 8.6 `PolyFit.java`: pencocokan polinomial dalam Java. 240
Kode 8.7 `compare_numpy.py`: pencocokan diperiksa dengan `linregress`, `lstsq`, `Polynomial.fit`, dan `curve_fit` taklinear langsung untuk kurva baterai. 245
Kode 9.1 `diff_steps.py`: beda maju dan beda pusat dari x di $x = 4$ untuk langkah yang makin kecil. 258
Kode 9.2 `DiffSteps.java`: percobaan yang sama dalam Java. 259
Kode 9.3 `richardson.py`: tabel Richardson untuk beda pusat. 263
Kode 9.4 `Richardson.java`: tabel yang sama dalam Java. 263
Kode 9.5 `speed.py`: kecepatan dan percepatan dari catatan jarak, dengan pengereman mendadak ditandai. 265
Kode 9.6 `Speed.java`: penganalisis yang sama dalam Java. 267
Kode 9.7 `smooth.py`: kecepatan dan percepatan dari parabola kuadrat terkecil lima titik, di samping rumus biasa.....271
Kode 9.8 `Smooth.java`: perbandingan yang sama dalam Java. 271

Kode 9.9 `compare_numpy.py`: kecepatan perjalanan dari `np.gradient` dan turunan x dari `scipy.differentiate.derivative`. 275
Kode 9.10 `uneven.py`: kecepatan dari bacaan berjarak tak seragam. 280
Kode 9.11 `Uneven.java`: perhitungan yang sama dalam Java. 280
Kode 10.1 `orchard.py`: luas kebun dengan aturan trapesium dan Simpson, beserta taksiran galat dari membagi dua data. 288
Kode 10.2 `Orchard.java`: kalkulator yang sama dalam Java. 288
Kode 10.3 `quadrature.py`: aturan trapesium dan Simpson gabungan untuk integral yang diketahui, beserta rasio galat berturutan. 291
Kode 10.4 `Quadrature.java`: pemeriksaan yang sama dalam Java. 292
Kode 10.5 `romberg.py`: tabel Romberg, yang hanya menambahkan titik tengah baru pada setiap baris. 294
Kode 10.6 `Romberg.java`: tabel yang sama dalam Java. 295
Kode 10.7 `gauss.py`: kuadratur Gauss–Legendre dengan satu sampai lima titik. 298
Kode 10.8 `Gauss.java`: aturan yang sama dalam Java, dengan titik simpul dalam larik tiga dimensi. 299
Kode 10.9 `adaptive.py`: integrasi Simpson adaptif, dibandingkan dengan Simpson seragam. 302
Kode 10.10 `Adaptive.java`: perbandingan yang sama dalam Java. 304
Kode 10.11 `compare_scipy.py`: kebun dan integral-integral uji dengan SciPy. 306
Kode 11.1 `tank_euler.py`: metode Euler untuk tandon yang mengosong, dibandingkan dengan solusi eksak. 315
Kode 11.2 `TankEuler.java`: program yang sama dalam Java. 316
Kode 11.3 `ode_orders.py`: Euler, Heun, titik tengah, dan RK4 sebagai fungsi langkah yang dapat saling menggantikan, beserta rasio galat untuk langkah yang dibagi dua. 320
Kode 11.4 `OdeOrders.java`: perbandingan yang sama dalam Java. 321
Kode 11.5 `morning.py`: satu pagi tandon irigasi tetes dengan RK4, termasuk sakelar pompa, luapan, dan saat kering. 324
Kode 11.6 `Morning.java`: simulator yang sama dalam Java. 325
Kode 11.7 `pump_plan.py`: biseksi pada jam mati pompa, dengan simulasi RK4 sebagai fungsi yang akarnya dicari. 329
Kode 11.8 `PumpPlan.java`: perencana yang sama dalam Java. 330
Kode 11.9 `bump.py`: jok setelah polisi tidur sebagai sistem dua persamaan, dengan Euler dan RK4. 332
Kode 11.10 `Bump.java`: sistem yang sama dalam Java, dengan operasi vektor ditulis dalam fungsi bantu `axpy`. 333
Kode 11.11 `compare_scipy.py`: tandon dan jok dengan `solve_ivp`. 337
Kode 11.12 `semi_implicit.py`: Euler semi-implisit untuk peredam aus. 341
Kode 11.13 `SemiImplicit.java`: metode yang sama dalam Java. 342
Kode 12.1 `market.py`: pangsa binatu bulan demi bulan, metode pangkat pada matriks perpindahan. 347
Kode 12.2 `Market.java`: program yang sama dalam Java. 347
Kode 12.3 `power.py`: iterasi pangkat, invers, dan invers bergeser untuk ketiga ragam ruko. 353
Kode 12.4 `Power.java`: metode yang sama dalam Java; hasil sebuah iterasi berupa `record` kecil. 354
Kode 12.5 `pagerank.py`: PageRank dengan metode pangkat, dengan halaman menggantung ditangani secara eksplisit. 360
Kode 12.6 `PageRank.java`: program yang sama dalam Java; peringkatnya diurutkan dengan sebuah pembanding. 361
Kode 12.7 `compare_numpy.py`: pasar binatu, ruko, dan PageRank dengan `numpy.linalg`. 365
Kode 12.8 `deflation.py`: deflasi Hotelling untuk menemukan ragam tertinggi kedua. 370
Kode 12.9 `Deflation.java`: program yang sama dalam Java. 370
Kode 13.1 `golden.py`: *golden-section search* untuk harga teh terbaik. 380
Kode 13.2 `Golden.java`: pencarian yang sama dalam Java. 382
Kode 13.3 `newton1d.py`: metode Newton untuk harga terbaik, biasa dan berpengaman. 383
Kode 13.4 `Newton1D.java`: hal yang sama dalam Java. 384

Kode 13.5 `twoprices.py`: *gradient descent* dengan langkah tetap dan *backtracking*, serta metode Newton, untuk dua harga. 385
Kode 13.6 `TwoPrices.java`: pengoptimal yang sama dalam Java. 388
Kode 13.7 `check_scipy.py`: masalah yang sama diselesaikan dengan pengoptimal SciPy. 394
Kode 14.1 `mlcore.py`: data, kuadrat terkecil dengan persamaan normal dan dengan QR, pembangkit bilangan acak yang portabel, dan fungsi logistik yang stabil. 408
Kode 14.2 `MlCore.java`: blok-blok bangunan yang sama dalam Java. 411
Kode 14.3 `linfit.py`: model waktu perjalanan dengan persamaan normal dan dengan QR, serta percobaan polinomial. 415
Kode 14.4 `LinFit.java`: pencocokan yang sama dalam Java. 416
Kode 14.5 `train_gd.py`: pelatihan model waktu perjalanan dengan batch penuh, *mini-batch*, *momentum*, dan Adam. 418
Kode 14.6 `TrainGd.java`: percobaan yang sama dalam Java. 420
Kode 14.7 `logistic.py`: regresi logistik dengan metode Newton (IRLS) dan dengan *gradient descent*, dengan pemeriksaan gradien dan rumus yang stabil. 423
Kode 14.8 `Logistic.java`: hal yang sama dalam Java. 425
Kode 14.9 `check_numpy.py`: hasil bab ini diperiksa dengan NumPy dan SciPy, bilangan kondisi, dan percobaan float32. 431
Kode 15.1 `summation.py`: penjumlahan naif, berpotongan, berpasangan, dan Kahan. 444
Kode 15.2 `Summation.java`: penjumlahan yang sama dalam Java. 445
Kode 15.3 `griddle_loops.py`: sapuan Jacobi pada wajan halus dengan gelung biasa. 447
Kode 15.4 `griddle_numpy.py`: sapuan Jacobi tervektorisasi, bila perlu dibagi ke beberapa *thread* 448
Kode 15.5 `Griddle.java`: sapuan Jacobi dalam Java pada satu atau beberapa *thread*. 449
Kode 15.6 `griddle_rb.py`: SOR merah-hitam dengan NumPy. 451
Kode 15.7 `GriddleRb.java`: SOR merah-hitam dalam Java pada satu atau beberapa *thread* 452
Kode 15.8 `check_numpy.py`: jumlah eksak dan berpasangan, serta wajan dengan pemecah langsung *sparse*. 456
Kode 16.1 `toolbox.py`: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun. 469
Kode 16.2 `Toolbox.java`: metode-metode yang sama dalam Java. 475
Kode 16.3 `orchard.py`: sistem air kebun sebagai satu model. 482
Kode 16.4 `Orchard.java`: model kebun dalam Java. 484
Kode 16.5 `components.py`: setiap bagian model, diperiksa dengan metode kedua. 489
Kode 16.6 `Components.java`: pemeriksaan yang sama dalam Java. 490
Kode 16.7 `sensor.py`: koefisien tetes yang ditaksir dari sensor tinggi air. 491
Kode 16.8 `Sensor.java`: taksiran yang sama dalam Java. 492
Kode 16.9 `montecarlo.py`: banyak musim kemarau secara paralel, dapat diulang. 493
Kode 16.10 `MonteCarlo.java`: jalannya Monte Carlo yang sama dengan *thread* Java. 495
Kode 16.11 `design.py`: ukuran tandon dengan *golden-section search* pada biaya tahunan. 496
Kode 16.12 `Design.java`: pencarian yang sama dalam Java. 497
Kode 16.13 `check_scipy.py`: bagian-bagian model dihitung ulang dengan NumPy dan SciPy. 502

Daftar Pseudokode

Pseudokode 2.1 Metode biseksi dengan toleransi pada perubahan titik tengah. 38
Pseudokode 2.2 Regula falsi, dengan modifikasi Illinois yang dinyalakan lewat sebuah penanda (baris 10 dan 14). 42
Pseudokode 3.1 Metode Newton-Raphson dengan tiga pengamannya. 64

Pseudokode 4.1 Eliminasi Gauss dengan *partial pivoting* opsional (baris 2 sampai 4) dan substitusi mundur. 94
Pseudokode 5.1 *Successive over-relaxation*; dengan omega = 1 ia adalah metode Gauss-Seidel. 127
Pseudokode 6.1 Dekomposisi LU dengan *pivoting* parsial, disimpan di tempat, dan dua penyelesaian segitiga. 160
Pseudokode 13.1 *Golden-section search* untuk maksimum fungsi unimodal pada $[a, b]$. 375
Pseudokode 15.1 Penjumlahan terkompensasi Kahan: c memulihkan galat pembulatan setiap penjumlahan dan menyuapkannya ke penjumlahan berikutnya. 443

Daftar Persamaan

(1.1) 12
(1.2) 12
(1.3) 13
(1.4) 13
(1.5) 14
(1.6) 14
(1.7) 15
(1.8) 15
(1.9) 18
(1.10) 19
(1.11) 19
(1.12) 19
(1.13) 22
(1.14) 24
(1.15) 24
(1.16) 27
(2.1) 37
(2.2) 37
(2.3) 37
(2.4) 40
(2.5) 40
(2.6) 41
(2.7) 42
(2.8) 43
(3.1) 62
(3.2) 63
(3.3) 63
(3.4) 65
(3.5) 65
(3.6) 66
(3.7) 66
(3.8) 67
(3.9) 67
(3.10) 74
(4.1) 92
(4.2) 92
(4.3) 93
(4.4) 93

(4.5) 94
(4.6) 95
(4.7) 97
(4.8) 97
(4.9) 97
(4.10) 98
(4.11) 105
(4.12) 109
(4.13) 110
(5.1) 123
(5.2) 124
(5.3) 126
(5.4) 126
(5.5) 126
(5.6) 127
(5.7) 127
(5.8) 127
(5.9) 128
(5.10) 128
(5.11) 128
(5.12) 129
(5.13) 129
(5.14) 130
(5.15) 130
(6.1) 158
(6.2) 159
(6.3) 159
(6.4) 160
(6.5) 161
(6.6) 162
(6.7) 163
(6.8) 163
(6.9) 164
(6.10) 165
(6.11) 165
(6.12) 166
(6.13) 166
(7.1) 192
(7.2) 192
(7.3) 193
(7.4) 194
(7.5) 194
(7.6) 195
(7.7) 195
(7.8) 196
(7.9) 196
(7.10) 197
(7.11) 198
(7.12) 198
(7.13) 199
(8.1) 224

(8.2) 225
(8.3) 226
(8.4) 226
(8.5) 226
(8.6) 227
(8.7) 227
(8.8) 228
(8.9) 228
(8.10) 229
(8.11) 229
(8.12) 230
(8.13) 230
(9.1) 254
(9.2) 255
(9.3) 256
(9.4) 256
(9.5) 257
(9.6) 261
(9.7) 263
(9.8) 270
(10.1) 285
(10.2) 285
(10.3) 286
(10.4) 286
(10.5) 287
(10.6) 287
(10.7) 287
(10.8) 294
(10.9) 297
(10.10) 298
(11.1) 314
(11.2) 314
(11.3) 314
(11.4) 315
(11.5) 315
(11.6) 319
(11.7) 319
(11.8) 332
(11.9) 336
(11.10) 338
(12.1) 345
(12.2) 346
(12.3) 346
(12.4) 350
(12.5) 351
(12.6) 352
(12.7) 359
(13.1) 372
(13.2) 373
(13.3) 375
(13.4) 375

(13.5) 375
(13.6) 375
(13.7) 376
(13.8) 376
(13.9) 376
(13.10) 377
(13.11) 377
(13.12) 377
(13.13) 378
(13.14) 378
(13.15) 378
(13.16) 379
(13.17) 379
(14.1) 402
(14.2) 402
(14.3) 403
(14.4) 403
(14.5) 403
(14.6) 403
(14.7) 404
(14.8) 405
(14.9) 405
(14.10) 405
(14.11) 406
(14.12) 406
(14.13) 406
(14.14) 406
(14.15) 406
(14.16) 407
(14.17) 407
(15.1) 440
(15.2) 441
(15.3) 442
(15.4) 442
(15.5) 443
(15.6) 443
(16.1) 466
(16.2) 466
(16.3) 466
(16.4) 466
(16.5) 467
(16.6) 467
(16.7) 468
(16.8) 468
(16.9) 468
(16.10) 469
(16.11) 469

Kata Pengantar

Spreadsheet sebuah binatu menyatakan bahwa 4,1 kg dengan harga Rp7.500 per kilogram tidak persis Rp30.750. Seorang penjual air menandai tongkat celup untuk tangki yang berbaring miring. Seorang penjual martabak ingin tahu bagian mana dari wajannya yang cukup panas. Seorang pengemudi ojek bertanya-tanya berapa sebenarnya biaya "1,67% sebulan" baginya. Tidak satu pun pertanyaan itu memiliki rumus rapi untuk jawabannya, dan setiap pertanyaan itu dapat dijawab, dengan digit benar sebanyak yang diperlukan siapa pun, oleh program pendek yang mengulang satu langkah sederhana sampai jawabannya berhenti berubah. Itulah metode numerik: keterampilan memperoleh jawaban yang salahnya sebesar jumlah yang kita ketahui dan dapat kita terima.

Buku ini tumbuh dari mata kuliah Metode Numerik di ITS, tempat materi ini telah diajarkan kepada mahasiswa informatika selama bertahun-tahun, dan telah ditulis ulang dari dasar untuk tahun 2026. Metodenya adalah metode-metode klasik — biseksi dan metode Newton, eliminasi Gauss beserta kerabat iteratifnya, interpolasi, kuadrat terkecil, diferensiasi dan integrasi, persamaan diferensial, nilai eigen, dan optimasi — karena metode-metode itulah fondasi setiap pustaka ilmiah dan setiap *framework machine learning* yang dipakai saat ini. Aplikasinya baru, dan setiap aplikasi menjawab pertanyaan yang benar-benar diajukan seseorang di binatu, di kebun, di lapak pasar, atau di atas sadel sepeda motor.

Untuk siapa buku ini ditulis

Buku ini lengkap dengan sendirinya, dan Anda dianggap seorang pemula. Jika Anda belum pernah menulis sebaris kode, belum pernah memasang bahasa pemrograman, dan tidak banyak ingat kalkulus, Andalah pembaca yang dibayangkan saat buku ini ditulis. Setiap istilah diperkenalkan dengan kata-kata sederhana sebelum dipakai, setiap metode lebih dulu dikerjakan dengan tangan pada contoh kecil dengan angka sungguhan, matematika yang diperlukan dijelaskan di tempat ia diperlukan, dan setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak di sini. Bab 1 dibuka dengan sebuah nota yang jumlahnya tidak cocok dan ditutup dengan meja kerja yang siap untuk semua bab sesudahnya.

Pembaca yang sudah bisa memprogram, dan mahasiswa yang memerlukan ketelitian sebuah mata kuliah universitas, akan menemukan separuh lain buku ini: setiap metode diturunkan dengan benar, galat dan laju konvergensinya dianalisis, diukur dalam kode dan dibandingkan dengan ramalan buku teks, serta program yang utuh dan teruji, siap disesuaikan untuk masalah mereka sendiri. Prinsip itu membuat buku ini tebal. Buku yang lebih tipis akan menganggap Anda sudah menguasai bagian-bagian yang sulit, dan buku semacam itu tidak berguna bagi orang yang belum menguasainya.

Satu gagasan, dua bahasa

Setiap program dalam buku ini ditulis dua kali, dalam **Java** dan dalam **Python**, dan kedua versi itu mengerjakan hal yang sama, membaca data yang sama, dan mencetak keluaran yang identik, digit demi digit. Java adalah bahasa sistem besar yang berumur panjang, bahasa ponsel Android, dan bahasa banyak layanan bank serta pemerintahan; kompilernya menangkap banyak kesalahan sebelum program dijalankan. Python adalah bahasa percobaan cepat, pengolahan data, dan

sebagian besar komputasi ilmiah; ringkas dan mudah dibaca. Melihat satu metode ditulis dengan dua cara adalah jalan tercepat untuk memahami metode itu sendiri, bukan sekadar kebiasaan satu bahasa. Jika Anda belum mengenal keduanya, mulailah dengan Python, lalu baca versi Java-nya sesudah itu; perbandingan itu akan mengajari Anda Java dengan cepat.

Program-program kita sendiri hanya memakai pustaka standar kedua bahasa itu, karena tujuannya adalah melihat setiap langkah sebuah metode. Namun jawaban numerik layak mendapat pendapat kedua, sehingga setiap bab memeriksa hasilnya terhadap **NumPy** dan **SciPy**, pustaka Python yang membungkus kode numerik yang telah diuji dengan cermat selama puluhan tahun, dan menjelaskan setiap perbedaan pada digit-digit terakhir. Semuanya ditulis dan dijalankan di **Visual Studio Code**, editor gratis yang bekerja dengan cara yang sama di Windows, macOS, dan Linux; Bab 1 memasangnya, bersama JDK 25, Python 3.14, NumPy, dan SciPy, langkah demi langkah pada ketiga sistem itu.

Isi buku ini

Bab 1 menjelaskan mengapa jawaban komputer hampir selalu sedikit salah: bagaimana bilangan disimpan dalam *floating point*, apa itu galat pembulatan dan galat pemotongan, dan bagaimana mengukurnya. Bab ini menyiapkan meja kerja dan ditutup dengan laboratorium kecil berisi percobaan pada aritmetika mesin.

Bab 2 dan 3 mencari akar persamaan. Bab 2 membantu Mas Dimas, seorang pengemudi ojek, menemukan suku bunga sebenarnya yang tersembunyi dalam cicilan sebuah ponsel dengan biseksi dan *regula falsi*, metode yang tidak akan pernah kehilangan akar setelah akar itu diapit. Bab 3 menandai tongkat celup truk air Pak Harun dengan Newton-Raphson, metode *secant*, dan iterasi titik tetap, mengukur seberapa cepat masing-masing konvergen, dan menunjukkan kapan metode Newton gagal.

Bab 4 sampai 6 menyelesaikan sistem persamaan linear. Bab 4 menghitung berapa banyak setiap pupuk yang diperlukan pohon-pohon durian Bu Rahma dengan eliminasi Gauss, menunjukkan mengapa *pivoting* penting, dan menjelaskan mengapa residu yang kecil tidak menjamin galat yang kecil. Bab 5 menemukan suhu tunak di permukaan wajan martabak Pak Joni dengan metode iteratif Jacobi, Gauss-Seidel, dan SOR. Bab 6 memfaktorkan sebuah matriks sekali lalu memakainya berulang kali, dengan dekomposisi LU, Cholesky, dan QR, untuk menjawab pertanyaan pupuk bagi setiap blok sebuah kebun.

Bab 7 dan 8 menarik kurva melalui data. Bab 7 mengubah tabel dari pembuat tangki kios BBM Pak Darto menjadi liter yang akurat pada setiap kedalaman dengan interpolasi Lagrange dan Newton serta *spline* kubik. Bab 8 mencocokkan model pada buku catatan bensin dan perjalanan Mas Dimas dengan kuadrat terkecil, dan menunjukkan cara memilih model tanpa *overfitting*.

Bab 9 sampai 11 membahas perubahan. Bab 9 menjelaskan bagaimana aplikasi ojek daring dapat membedakan pengereman keras dari berhenti perlahan, dengan beda hingga yang diturunkan dari teorema Taylor dan ekstrapolasi Richardson. Bab 10 mengukur luas kebun durian yang batasnya berupa sungai berkelok dengan aturan trapesium, aturan Simpson, integrasi Romberg, dan kuadratur Gauss. Bab 11 menentukan kapan pompa kebun harus dimatikan dengan metode Euler dan Runge–Kutta untuk persamaan diferensial biasa, lalu membahas sistem dan kestabilan.

Bab 12 sampai 14 membawa metode-metode itu menuju komputasi modern. Bab 12 mengikuti tiga binatu yang bersaing di satu kampung dengan metode pangkat untuk nilai eigen, dan menunjukkan bagaimana gagasan yang sama memeringkat halaman-halaman web dengan

PageRank. Bab 13 menemukan harga paling menguntungkan untuk segelas es teh dengan *golden-section search*, metode Newton, dan *gradient descent*. Bab 14 menunjukkan bahwa *machine learning* pada intinya adalah metode numerik: kuadrat terkecil, *gradient descent* dengan *mini-batch*, *momentum*, dan Adam, serta regresi logistik yang dilatih dengan metode Newton, untuk meramalkan lama sebuah perjalanan ojek dan apakah seorang pelanggan akan membatalkan pesanan.

Bab 15 membuat program-program itu cepat: ke mana waktu habis, vektorisasi, *thread* paralel beserta batas-batasnya, penjumlahan yang akurat dan dapat diulang hasilnya, serta metode yang lebih cerdas yang memang mengerjakan lebih sedikit. Bab 16, proyek akhir, menyimulasikan musim kemarau sebuah kebun durian dari awal sampai akhir — tangki, pompa, cuaca, pohon, sumur, dan saluran tetes — dengan metode-metode dari seluruh buku, menjalankan ratusan musim, dan membantu Pak Hadi memilih tangki airnya.

Setiap bab menempuh jalan yang sama: masalahnya dan siapa yang dibantu, gagasannya dalam kata-kata sederhana dengan angka yang dihitung dengan tangan, metodenya diturunkan dengan galat dan konvergensinya dianalisis, program-programnya dibangun langkah demi langkah dalam Java dan Python, menjalankannya, memeriksa jawabannya terhadap NumPy dan SciPy, menyempurnakannya, memakainya sungguh-sungguh, metode numerik dalam praktik di Profitina, kesalahan yang sering terjadi, ringkasan, soal latihan dengan penyelesaian lengkap, dan daftar pustaka.

Setiap program dalam buku ini sudah dijalankan sebelum keluarannya dicetak. Program-program itu dikumpulkan, bab demi bab, dalam folder `Code` yang menyertai buku ini, dari `Chapter01` sampai `Chapter16`, masing-masing dengan folder `java` dan `python`, data yang diperlukannya, dan README yang menjelaskan cara menjalankannya di terminal dan di VS Code, pada Windows, macOS, dan Linux.

Cara membacanya

Dengan VS Code terbuka dan program-programnya berjalan. Bab 1 didahulukan bagi semua pembaca, karena bab itulah yang membangun meja kerja dan menjelaskan galat yang diukur oleh setiap bab sesudahnya; setelah itu bab-bab mendaki dari yang mudah menuju yang sulit, dan setiap bab menyebutkan bab-bab sebelumnya yang menjadi pijakannya. Kerjakan contoh-contoh kecilnya dengan tangan sebelum membaca kodenya, dan ketikkan programnya, jangan hanya menyalinnya; saat mengetik itulah kesalahan terjadi, dan dari kesalahan itulah pelajaran didapat. Setelah sebuah program berjalan, ubahlah — satu toleransi, satu ukuran langkah, satu angka dalam datanya — lalu perhatikan bagaimana galat dan jumlah iterasinya berubah: tidak ada cara yang lebih cepat untuk merasakan bagaimana sebuah metode numerik berperilaku. Semua soal latihan di akhir tiap bab dilengkapi penyelesaian lengkap — gunakan sesudah usaha Anda sendiri, bukan sebagai penggantinya.

Gambar, tabel, kode, pseudokode, dan persamaan yang bernomor semuanya adalah tautan hidup: pada edisi elektronik, klik “Kode 5.3” atau “[4]” di bagian mana pun dan Anda langsung dibawa ke sana. Daftar Isi serta daftar-daftar di bagian depan buku bekerja dengan cara yang sama.

Tentang penulis

Irfan Subakti menggambarkan dirinya dalam satu baris — *Father | Educator | Entrepreneur | Bridging Tech & Nature: CS Lecturer × Crypto/Digital Business × AI/ML Specialist × Fruit Farmer |*

Storyteller — ayah, pendidik, wirausahawan, penjembatan teknologi dan alam: dosen ilmu komputer, pelaku bisnis kripto dan digital, spesialis AI/ML, petani buah, dan pencerita. Setiap bagian dari baris itu ikut masuk ke dalam buku ini. Ia mengajar di Departemen Teknik Informatika ITS sejak 1998. Ia membimbing dan menguji tugas akhir mahasiswa sarjana, mengajar MLOps dalam program IBM SkillsBuild, dan membangun produk: ia mendirikan Profitina, platform visibilitas AI, yang pemakaian metode numeriknya sehari-hari menutup setiap bab dalam buku ini. Jauh dari papan ketik, ia merawat kebun buah dan durian keluarga — di sanalah ia belajar bahwa perangkat lunak yang baik, seperti kebun yang baik, ditumbuhkan dengan sabar, satu bagian yang terawat demi satu bagian. Jiwa penceritanya adalah alasan setiap bab dalam buku ini dibuka dengan seseorang dan masalahnya, bukan dengan sebuah definisi.

Ucapan terima kasih

Kepada para mahasiswa yang mempelajari versi-versi awal materi ini dalam mata kuliah Metode Numerik dan mengajukan pertanyaan yang menyingkapkan di mana sebuah penjelasan masih tipis, serta kepada rekan-rekan di ITS yang bersama-sama mengajarkan dan menyempurnakan materi metode numerik dari tahun ke tahun. Kepada para pemilik binatu, petani, pengemudi, penjual air, dan pedagang kaki lima yang masalah sehari-harinya memberi setiap bab kisahnya. Standar buku ini juga berutang pada dokumentasi resmi Java, Python, NumPy, SciPy, dan VS Code, serta pada buku teks klasik dan literatur riset analisis numerik, yang dikutip di setiap tempat penggunaannya.

Surabaya, September 2026

Irfan Subakti · yifana@gmail.com

Profitina

profitina.com

Bab 1

Angka di Dalam Mesin: Galat, Floating Point, dan Meja Kerja Anda

Komputer bekerja cepat dan tidak kenal lelah, tetapi ia tidak menghitung dengan bilangan real matematika. Ia menghitung dengan himpunan terbatas bilangan biner yang hanya menghampiri bilangan real, sehingga hampir setiap jawaban yang diberikannya sedikit keliru. Metode numerik adalah keterampilan untuk memperoleh jawaban yang kelirunya dapat kita ketahui besarnya dan dapat kita terima. Bab ini meletakkan fondasi bagi seluruh buku. Bab ini menyiapkan meja kerja Anda langkah demi langkah di Windows, macOS, dan Linux: Java Development Kit (JDK) 25, Python 3.14, Visual Studio Code, serta pustaka NumPy dan SciPy yang dipakai bab-bab berikutnya untuk memeriksa program kita sendiri. Setelah itu, bab ini membuka mesin dan melihat bagaimana bilangan disimpan. Di akhir bab Anda akan mampu memasang dan memeriksa perangkat Java dan Python yang profesional; mengukur galat sebagai galat mutlak, galat relatif, dan galat hampiran, lalu memutuskan kapan sebuah iterasi boleh berhenti; menjelaskan bagaimana bilangan presisi ganda disimpan dalam 64 bit dan mengapa 0.1 + 0.2 tidak sama dengan 0.3; membedakan galat pembulatan dari galat pemotongan dan menemukan ukuran langkah tempat keduanya seimbang; mengenali dan memperbaiki pembatalan katastrofik; serta menjumlahkan nota toko selama setahun tanpa kehilangan satu rupiah pun.

1.1 Nota yang Tidak Cocok

Bu Wati mengelola usaha *laundry* kecil di dekat sebuah kampus. Mahasiswa membawa pakaian dalam kantong plastik, Bu Wati menimbang setiap kantong dengan timbangan digital yang menampilkan kilogram dengan satu angka desimal, dan ia menarik Rp7.500 per kilogram. Kantong seberat 4,1 kg dihargai Rp30.750. Setiap sore ia menyalin pesanan dari buku catatannya ke *spreadsheet* yang disiapkan keponakannya di sebuah laptop tua, dan setiap akhir bulan ia mencocokkan total di *spreadsheet* dengan uang di laci kas.

Suatu sore keponakannya menunjukkan sesuatu yang aneh. Di sebuah sel ia mengetik berat 4.1 dan harga 7500, meminta hasil kalinya, lalu bertanya apakah hasil kali itu sama dengan 30750. *Spreadsheet* menjawab FALSE. Ketika kolomnya dilebarkan agar menampilkan lebih banyak desimal, hasil kali itu tampil sebagai 30749.999999999996. Tidak ada yang salah ketik. Laptop itu memang tidak mampu menyimpan bilangan 4,1 secara tepat, dan selisih kecil itu terbawa sampai ke hasil perkalian.

Untuk satu nota, empat per seribu miliar rupiah tidak berarti apa-apa. Namun *laundry* Bu Wati menerima sekitar enam puluh pesanan sehari, setiap hari sepanjang tahun: lebih dari dua puluh ribu nota. Keponakannya juga menulis program kasir kecil untuk sebuah tablet murah yang menyimpan nominal dalam format bilangan yang lebih pendek demi menghemat memori, dan di sana total setahun ternyata kurang Rp826. Jumlah itu memang tidak besar, tetapi total itu *salah*, dan total yang salah tanpa sebab yang kelihatan merusak kepercayaan. Kalau program bisa kehilangan Rp826, apa lagi yang bisa hilang?

Pertanyaan Bu Wati, "mengapa komputer tidak menjumlahkan nota saya dengan benar?", adalah pertanyaan tempat metode numerik bermula. Galat-galat kecil yang sama hidup di dalam setiap perhitungan komputer: dalam model cuaca BMKG, dalam tabel angsuran bank, dalam aplikasi yang memberi tahu pengemudi ojek berapa jauh ia sudah berkendara, dan dalam setiap program buku ini. Biasanya galat itu tidak berbahaya. Kadang-kadang berbahaya. Pada Perang Teluk 1991, sebuah baterai rudal Patriot di Dhahran gagal mencegat rudal yang datang karena jamnya menghitung waktu dalam persepuluhan detik dengan format bilangan yang tidak dapat menyimpan 0,1 secara tepat; setelah sekitar 100 jam beroperasi, galat yang menumpuk mencapai sekitar 0,34 detik, cukup untuk mencari sasaran di tempat yang salah, dan 28 prajurit tewas [1]. Galat per detak jam hanya sekitar sepersepuluh juta detik. Yang mematikan adalah penumpukannya.

Bab ini menjelaskan dari mana galat semacam itu berasal, bagaimana mengukurnya, dan bagaimana menjaganya tetap kecil. Subbab 1.13 kembali ke buku kas Bu Wati dan menjumlahkan setahun penuh secara tepat. Namun sebelum itu kita memerlukan meja kerja.

1.2 Apa Itu Metode Numerik

Matematika sekolah menyelesaikan masalah dengan rumus. Persamaan $2x + 3 = 11$ mempunyai penyelesaian $x = 4$, dan persamaan kuadrat $ax^2 + bx + c = 0$ mempunyai rumus akar yang terkenal. Masalah nyata jarang menyediakan rumus seperti itu. Bab 2 bertanya berapa suku bunga bulanan yang tersembunyi dalam sebuah rencana cicilan; persamaannya mudah ditulis, tetapi tidak ada rumus aljabar yang menyelesaikannya untuk suku bunga. Bab 3 membuat tanda pada tongkat ukur truk air yang tangkinya berbaring; volume air pada kedalaman tertentu ada rumusnya, tetapi kedalaman untuk volume tertentu tidak. Bab 4 mencari berapa banyak setiap pupuk yang dibutuhkan sebatang pohon durian dari empat persamaan dengan empat peubah, yang pada prinsipnya ada rumusnya, tetapi bukan rumus yang ingin dipakai siapa pun dengan tangan untuk sepuluh atau seribu peubah.

Metode numerik mengganti masalah seperti itu dengan barisan penjumlahan, pengurangan, perkalian, dan pembagian yang terbatas, yang dapat dijalankan komputer, dan yang menghasilkan jawaban sedekat yang kita perlukan dengan jawaban sebenarnya. Biseksi, metode pertama Bab 2, adalah contoh yang baik: untuk mencari tempat sebuah fungsi berganti tanda, ia membagi dua sebuah selang berulang-ulang; setelah 20 kali pembagian, jawabannya diketahui sampai satu per sejuta. Jawabannya tidak pernah tepat, tetapi kita tahu *seberapa* tidak tepat, dan kita dapat membeli ketelitian lebih dengan kerja lebih.

Karena itu setiap metode dalam buku ini dinilai dengan tiga pertanyaan yang sama, yang sudah menjadi pertanyaan buku-buku teks klasik [2], [3]:

1. **Apakah ia konvergen?** Ketika kita menambah kerja (lebih banyak iterasi, langkah lebih kecil), apakah galatnya menuju nol?
2. **Seberapa cepat?** Apakah setiap iterasi menambah sejumlah tetap digit benar, atau melipatgandakannya? Inilah *orde konvergensi*, yang diukur di Bab 3.
3. **Apakah ia stabil?** Apakah galat kecil mesin tetap kecil, atau metode itu memperbesarnya sampai jawabannya tak berguna? Subbab 1.11 menunjukkan sebuah rumus yang benar secara aljabar tetapi tak berguna di komputer.

Buku ini bergerak dari alat-alat dasar ke pemakaian modernnya. Bab 2 dan 3 mencari akar persamaan; Bab 4 sampai 6 menyelesaikan sistem persamaan linear; Bab 7 dan 8 mencocokkan kurva pada data; Bab 9 sampai 11 menurunkan, mengintegralkan, dan menyelesaikan persamaan

diferensial; Bab 12 dan 13 mencari nilai eigen dan titik optimum; Bab 14 menunjukkan bagaimana semua ini berada di dalam *machine learning*; Bab 15 membuatnya cepat; dan Bab 16 menyimulasikan sebuah sistem nyata dari awal sampai akhir. Setiap bab berangkat dari masalah yang dialami orang biasa, mengerjakan gagasannya dengan tangan memakai angka nyata, menurunkan metodenya beserta analisis galatnya, membangunnya dalam Java dan Python, menjalankannya, lalu membandingkan jawaban kita dengan jawaban pustaka profesional.

1.3 Meja Kerja Anda: Apa yang Dipasang dan Mengapa

Pekerjaan pertama tukang kayu di bengkel baru adalah menata meja kerjanya. Meja kerja kita memuat alat-alat yang diringkas dalam **Tabel 1.1**. Versinya adalah versi terbaru per September 2026; rilis tambalan yang lebih baru (JDK 25.0.5 atau Python 3.14.8) bekerja persis sama.

Tabel 1.1 Alat-alat meja kerja dan versi yang dipakai di seluruh buku.

Alat	Apa itu	Versi yang dipakai buku ini
JDK (Java Development Kit)	Kompilator Java `javac`, *runtime* Java `java`, dan pustaka standar	JDK 25 LTS, pembaruan 25.0.4 (Juli 2026) [4], [5]
Python	Interpreter Python `python` beserta pustaka standar, `pip`, dan `venv`	3.14.7 (5 Agustus 2026) [6]
NumPy dan SciPy	Pustaka Python untuk larik, aljabar linear, pencarian akar, dan banyak lagi	NumPy 2.5.3, SciPy 1.18.1 [7], [8]
Visual Studio Code	Editor kode gratis untuk Windows, macOS, dan Linux	1.139 (23 September 2026) [9]
Extension Pack for Java	Menambahkan penyuntingan, eksekusi, dan *debugging* Java ke VS Code	dari tampilan Extensions [10]
Ekstensi Python	Menambahkan penyuntingan (Pylance), eksekusi, dan *debugging* Python ke VS Code	dari tampilan Extensions [11]

Mengapa JDK 25? Java mempunyai versi baru setiap enam bulan, dan setiap dua tahun salah satunya menjadi rilis *long-term support* (LTS) yang menerima pembaruan keamanan selama bertahun-tahun. JDK 25 dirilis pada 16 September 2025 sebagai LTS terkini [5]; pembaruan Juli 2026 adalah 25.0.4 [4]. JDK 27 muncul pada 15 September 2026 [12], tetapi itu rilis berjangka pendek, dan untuk buku yang programnya harus tetap berjalan bertahun-tahun, LTS adalah pilihan yang tepat. Setiap program buku ini hanya memakai fitur yang sudah ada sejak Java 17, sehingga JDK 17 dan 21 juga dapat menjalankannya. Ada satu perbedaan kecil yang patut diketahui: sejak JDK 19, Java mencetak sebuah `double` dengan untaian desimal terpendek yang mengenalinya, persis seperti Python, sedangkan versi lama kadang mencetak satu digit terlalu banyak [13]. Dengan JDK 19 atau yang lebih baru, program Java dan Python buku ini mencetak keluaran yang sama persis sampai ke tiap bita.

Ada beberapa *build* dari JDK yang sama, semuanya dibuat dari kode sumber terbuka OpenJDK: milik Oracle sendiri, Eclipse Temurin dari proyek Adoptium, milik Microsoft, Amazon Corretto, dan paket distribusi Linux. Untuk keperluan kita semuanya berperilaku sama. Kita memakai **Eclipse Temurin** karena gratis untuk keperluan apa pun, mempunyai penginstal sederhana untuk setiap sistem, dan tersedia di pengelola paket ketiga sistem; rilis keamanan September 2026 adalah Temurin 25.0.4.1 [14].

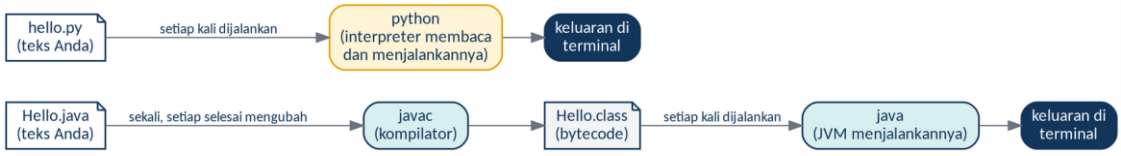
Mengapa Python 3.14? Python 3.14 pertama kali dirilis pada 7 Oktober 2025 dan membawa, antara lain, *template string*, evaluasi anotasi yang ditunda, *build* tanpa *global interpreter lock* yang didukung resmi, dan *shell* interaktif yang lebih ramah [15]. Program buku ini tidak memerlukan fitur-fitur itu; program-programnya memakai Python sederhana yang sudah lama mapan dan

juga berjalan di Python 3.11 atau yang lebih baru. Kita memasang 3.14 karena itulah versi terkini dan karena pustaka yang kita pakai sebagai pembanding dibangun dan diuji untuknya.

Mengapa NumPy dan SciPy? Program kita sendiri hanya memakai pustaka standar Java dan Python, karena tujuannya adalah melihat setiap langkah sebuah metode. Namun hasil numerik layak mendapat pendapat kedua. NumPy dan SciPy membungkus kode numerik yang telah diuji dengan cermat selama puluhan tahun (LAPACK untuk aljabar linear, metode Brent untuk akar, dan banyak lagi), dan di akhir setiap bab kita mengajukan pertanyaan yang sama kepada keduanya lalu membandingkan. Java tidak mempunyai padanan tunggal, sehingga sisi Java tetap memakai kode biasa dan alat bawaan pustaka standar seperti `BigDecimal` dan `Math.ulp`.

Mengapa VS Code? Karena gratis, berjalan di ketiga sistem, sama baiknya untuk Java dan Python, dan mempunyai terminal bawaan. Editor lain juga dapat dipakai; IntelliJ IDEA dan PyCharm sangat baik.

Sebelum memasang apa pun, ada gunanya mengetahui apa yang terjadi ketika Anda "menjalankan" program di masing-masing bahasa, karena perbedaannya menjelaskan sebagian besar pesan galat di hari pertama. **Gambar 1.1** memperlihatkan kedua jalurnya.



Gambar 1.1 Bagaimana program Java dan program Python dijalankan. Java dikompilasi sekali oleh `javac` menjadi *bytecode* yang dijalankan Java Virtual Machine; Python dibaca dan dijalankan interpreter setiap kali.

Program Java adalah teks dalam berkas seperti `Hello.java`. Kompilator `javac` memeriksanya dan menerjemahkannya menjadi *bytecode* yang disimpan di `Hello.class`, dan perintah `java` menyalakan Java Virtual Machine (JVM) yang menjalankan *bytecode* itu. Program Python adalah teks dalam berkas seperti `hello.py`, dan perintah `python` membaca serta menjalankannya langsung setiap kali; kesalahan yang ditemukan Java saat kompilasi baru ditemukan Python ketika mencapai baris yang salah. Sejak Java 11 Anda juga dapat menjalankan satu berkas Java tanpa mengompilasinya dulu, dengan `java Hello.java` [16]. Untuk program yang terdiri atas beberapa berkas, seperti kebanyakan bab buku ini, kita mengompilasi dengan `javac`.

1.4 Memasang di Windows 10 dan 11

1.4.1 JDK

Cara paling sederhana adalah **winget**, pengelola paket bawaan Windows 10 dan 11. Buka **PowerShell** (tekan tombol Windows, ketik `powershell`, tekan Enter) lalu ketik:

```
PS C:\Users\you> winget install EclipseAdoptium.Temurin.25.JDK
```

Jika Anda lebih suka mengeklik, buka `https://adoptium.net`, pilih **Temurin 25 (LTS)**, **Windows**, dan **x64** (atau **aarch64** pada laptop ARM), unduh penginstal `.msi`, lalu jalankan. Pada halaman **Custom Setup**, atur **Add to PATH** dan **Set JAVA_HOME variable** menjadi "Will be installed on local hard drive"; pilihan kedua mati secara bawaan, padahal beberapa alat membutuhkannya.

Tutup PowerShell, buka yang baru (jendela yang dibuka sebelum pemasangan tidak mengenal program baru), lalu periksa kedua perintah:

```
PS C:\Users\you> java -version
openjdk version "25.0.4" 2026-07-21 LTS
PS C:\Users\you> javac -version
javac 25.0.4
```

Baris pertama `java -version` itulah yang penting; dua baris berikutnya menyebut *build*-nya. Jika Windows menjawab bahwa `java` "is not recognized as the name of a cmdlet", penginstal tidak menambahkan Java ke `PATH`, yaitu daftar folder tempat Windows mencari perintah. Jalankan penginstal lagi dan pilih **Add to PATH**, atau tambahkan sendiri `C:\Program Files\Eclipse Adoptium\jdk-25...\bin` melalui **Settings › System › About › Advanced system settings › Environment Variables**.

1.4.2 Python

Di Windows, Python kini dipasang dengan **Python install manager**, program kecil yang mengunduh, memperbarui, dan berpindah antarversi Python [17]. Dapatkan dari Microsoft Store (cari "Python Install Manager"), dari `https://www.python.org/downloads/windows/`, atau dengan winget:

```
PS C:\Users\you> winget install 9NQ7512CXL7T -e --accept-package-agreements
```

Kemudian buka jendela PowerShell baru, pasang Python 3.14, dan periksa:

```
PS C:\Users\you> py install 3.14
PS C:\Users\you> python --version
Python 3.14.7
```

Setelah ini, `python`, `py`, dan `python3` semuanya menjalankan Python. Jika `python --version` malah membuka Microsoft Store, tutup semua terminal, buka yang baru, lalu coba lagi; pengelola itu juga dapat memperbaiki pengaturannya dengan `py install --configure` [17].

1.4.3 VS Code

Pasang VS Code dengan `winget install Microsoft.VisualStudioCode`, atau unduh **User Installer** dari `https://code.visualstudio.com`. Di penginstal, centang **Add to PATH** dan **Add "Open with Code" action to Windows Explorer file context menu**: yang pertama memungkinkan Anda mengetik `code .` di terminal untuk membuka folder saat ini, dan yang kedua memungkinkan Anda mengeklik kanan sebuah folder dan membukanya di VS Code.

1.5 Memasang di macOS

1.5.1 JDK

Buka `https://adoptium.net`, pilih **Temurin 25 (LTS)**, **macOS**, dan **aarch64** untuk Mac dengan cip Apple (M1 dan sesudahnya) atau **x64** untuk Mac Intel yang lebih tua, unduh berkas `.pkg`, lalu klik ganda. Dengan pengelola paket Homebrew, satu perintah melakukan hal yang sama:

```
you@mac ~ % brew install --cask temurin@25
```

Buka **Terminal** (tekan Cmd+Space, ketik "Terminal", tekan Enter) lalu periksa:

```
you@mac ~ % java -version
openjdk version "25.0.4" 2026-07-21 LTS
you@mac ~ % javac -version
javac 25.0.4
```

Jika terpasang beberapa JDK, macOS memilih yang terbaru; `/usr/libexec/java_home -V` menampilkan semuanya.

1.5.2 Python

macOS mungkin sudah berisi perintah `python3`, tetapi itu versi lama untuk alat pengembang Apple; jangan membangun di atasnya. Unduh penginstal macOS untuk Python 3.14.7 dari `https://www.python.org/downloads/macos/` lalu klik ganda; paket ini universal dan berjalan langsung di cip Apple maupun Intel. Di akhir pemasangan, penginstal membuka folder **Applications › Python 3.14**: klik ganda **Install Certificates.command** di sana sekali saja, agar Python (dan `pip`) dapat membuat sambungan internet yang aman. Dengan Homebrew, `brew install python@3.14` juga bisa. Lalu periksa:

```
you@mac ~ % python3 --version
Python 3.14.7
```

Di macOS dan Linux perintahnya `python3`, bukan `python`. Buku ini menulis `python3` untuk perintah di macOS dan Linux dan `python` untuk Windows.

1.5.3 VS Code

Unduh VS Code untuk macOS dari `https://code.visualstudio.com`, buka berkas yang terunduh, lalu seret **Visual Studio Code** ke **Applications**. Agar dapat mengetik `code .` di terminal, jalankan VS Code, tekan Cmd+Shift+P untuk membuka Command Palette, lalu jalankan **Shell Command: Install 'code' command in PATH**.

1.6 Memasang di Linux

Sebagian besar distribusi Linux menyediakan versi Java dan Python yang baru di repositorinya sendiri, dan itulah pilihan terbaik karena sistem akan terus memperbaruinya. Ubuntu 26.04 LTS memakai OpenJDK 25 sebagai Java bawaan [18] dan Python 3.14 sebagai `python3` bawaan, tetapi tidak memasang `javac` maupun modul `venv` Python kecuali Anda memintanya:

```
$ sudo apt update
$ sudo apt install openjdk-25-jdk python3 python3-venv python3-pip
$ java -version
openjdk version "25.0.4" 2026-07-21
$ javac -version
javac 25.0.4
$ python3 --version
Python 3.14.7
```

Di Fedora padanannya adalah `sudo dnf install java-25-openjdk-devel python3 python3-pip`. Pada distribusi lama yang repositorinya belum memuat JDK 25, pasang Temurin dari repositori Adoptium sendiri dengan mengikuti `https://adoptium.net/installation`, atau pertahankan JDK 21 bawaan distribusi: setiap program buku ini berjalan di JDK 17 atau yang lebih baru. Hal yang sama berlaku untuk Python: Ubuntu 24.04 membawa Python 3.12, yang menjalankan semua program buku ini. Jangan pernah menghapus atau mengganti Python sistem di Linux; sebagian sistem operasi bergantung padanya.

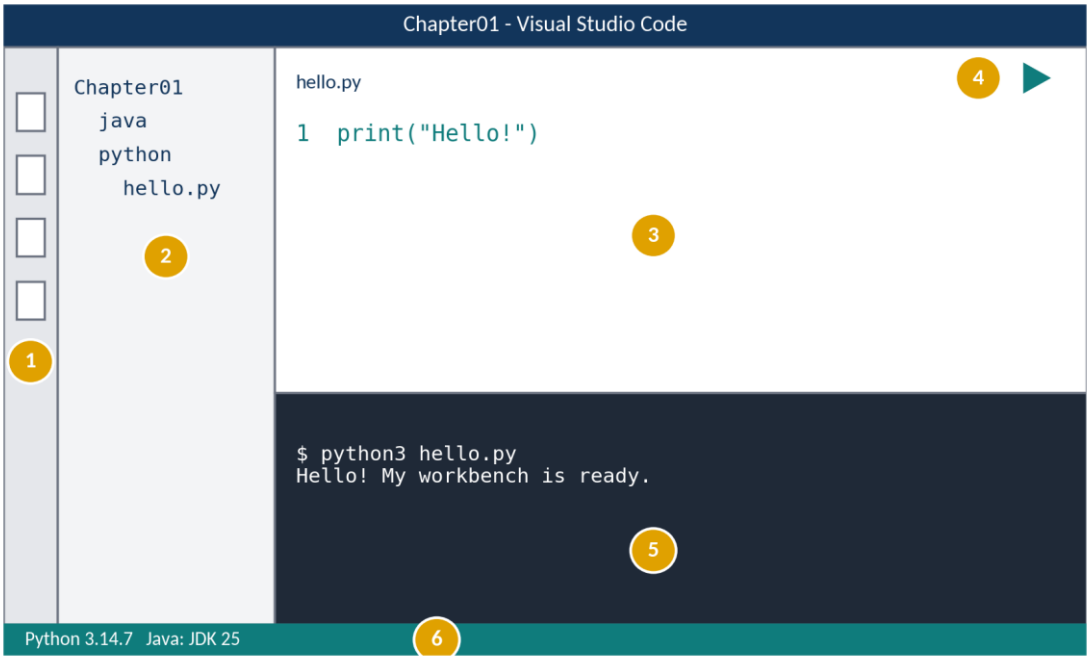
Untuk VS Code, unduh paket `.deb` (Ubuntu, Debian, Mint) atau `.rpm` (Fedora, openSUSE) dari `https://code.visualstudio.com` lalu pasang dengan `sudo apt install ./code_*.deb` atau `sudo dnf install ./code-*.rpm`. Paket itu juga menambahkan repositori Microsoft, sehingga VS Code kemudian diperbarui bersama bagian lain sistem.

1.7 VS Code, Lingkungan Virtual, NumPy, dan SciPy

Jalankan VS Code. Klik ikon **Extensions** di Activity Bar sebelah kiri (atau tekan Ctrl+Shift+X, di macOS Cmd+Shift+X) lalu pasang dua ekstensi, keduanya diterbitkan Microsoft:

- **Extension Pack for Java** (pengenal `vscjava.vscode-java-pack`). Paket ini menggabungkan dukungan bahasa Java, *debugger*, penjalan uji, dan pengelola proyek [10]. Ia menemukan sendiri JDK yang Anda pasang; jika ada beberapa, perintah **Java: Configure Java Runtime** di Command Palette memungkinkan Anda memilih.
- **Python** (pengenal `ms-python.python`). Ekstensi ini membawa **Pylance**, yang melengkapi nama dan menandai kesalahan sambil Anda mengetik, serta **Python Debugger** [11].

Gambar 1.2 memperlihatkan bagian-bagian jendela yang dirujuk buku ini. Activity Bar (1) berpindah antara Explorer, Search, Source Control, Run and Debug, dan Extensions. Explorer (2) menampilkan berkas-berkas di folder yang Anda buka. Editor (3) adalah tempat Anda mengetik. Tombol Run (4), segitiga di kanan atas editor, menjalankan berkas yang sedang dibuka. Terminal terpadu (5), dibuka dengan **Terminal › New Terminal** atau Ctrl+`, adalah jendela PowerShell, zsh, atau bash biasa yang sudah berada di folder proyek Anda. Status Bar (6) menampilkan interpreter Python dan *runtime* Java yang sedang dipakai VS Code; mengeklik versi Python memungkinkan Anda memilih yang lain.



Gambar 1.2 Bagian-bagian Visual Studio Code yang dipakai buku ini: (1) Activity Bar, (2) Explorer, (3) editor, (4) tombol Run, (5) terminal terpadu, (6) Status Bar.

Selalu buka sebuah *folder* di VS Code (**File › Open Folder**), bukan satu berkas saja. Folder itu menjadi ruang kerja Anda: Explorer menampilkannya, terminal dimulai di dalamnya, dan ekstensi Java memperlakukan berkas `.java` di dalamnya sebagai satu proyek. Untuk buku ini, bukalah folder bab yang sedang Anda baca, misalnya `Chapter01`. Setiap folder bab mempunyai folder `java`, folder `python`, folder `data` bila programnya membaca data, berkas `README.md` yang mengulang setiap perintah untuk Windows, macOS, dan Linux, serta berkas `requirements.txt`.

Berbeda dengan bab-bab awal banyak buku pemrograman, buku ini sudah memerlukan dua pustaka Python sejak bab pertama, karena setiap bab membandingkan hasil kita dengan NumPy dan SciPy. Pustaka dipasang dengan `pip`, dan sebaiknya dipasang ke dalam sebuah **lingkungan**

virtual (*virtual environment*): folder pribadi, lazimnya bernama `.venv`, dengan salinan `python` sendiri dan tempat sendiri untuk paket, sehingga pustaka satu proyek tidak pernah bertabrakan dengan pustaka proyek lain [19]. Buat satu lingkungan untuk seluruh buku, di folder yang memuat folder-folder bab, lalu pasang kedua pustaka ke dalamnya. Di Windows:

```
PS C:\Users\you\MetodeNumerik> python -m venv .venv
PS C:\Users\you\MetodeNumerik> .venv\Scripts\Activate.ps1
(.venv) PS C:\Users\you\MetodeNumerik> python -m pip install numpy scipy
```

Di macOS dan Linux:

```
$ python3 -m venv .venv
$ source .venv/bin/activate
(.venv) $ python -m pip install numpy scipy
```

Kini *prompt* diawali `(.venv)`, tanda bahwa lingkungan itu aktif; di dalamnya, `python` berarti Python milik lingkungan itu di sistem mana pun. Jika PowerShell menolak menjalankan `Activate.ps1` dengan pesan tentang *execution policy*, jalankan `Set-ExecutionPolicy -Scope CurrentUser RemoteSigned` sekali lalu coba lagi. Di VS Code, tekan Ctrl+Shift+P, jalankan **Python: Select Interpreter**, lalu pilih interpreter di dalam `.venv`; sejak itu tombol Run dan terminal terpadu memakainya. Berkas `requirements.txt` setiap bab mencantumkan apa yang dibutuhkan bab itu, sehingga `python -m pip install -r requirements.txt` di folder bab selalu cukup.

1.8 Program Pertama Anda

1.8.1 Halo, dalam Java

Buka folder `Chapter01` di VS Code, atau buat folder itu beserta subfolder `java` dan `python`. Pilih **File › New File**, ketik program di **Kode 1.1**, lalu simpan sebagai `java/Hello.java`. Di Java, nama berkas harus sama dengan nama kelas diikuti `.java`, dengan huruf besar-kecil yang sama.

Kode 1.1 `Hello.java`: program Java pertama, yang juga melaporkan versi Java dan sistem operasi.

```java
/** The first program: say hello and show which Java runs it. */
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello! My workbench for numerical methods is ready.");
        System.out.println("Java version: " + System.getProperty("java.version"));
        System.out.println("Running on:   " + System.getProperty("os.name"));
    }
}
```

`public class Hello` mendeklarasikan sebuah *kelas*, satuan tempat Java menyimpan kode. Komentar di atasnya, di antara `/**` dan `*/`, ditujukan untuk manusia dan diabaikan kompilator. `public static void main(String[] args)` adalah *metode main*, tempat setiap aplikasi Java dimulai; `String[] args` menerima kata-kata yang diketik sesudah nama program di baris perintah. `System.out.println(...)` mencetak satu baris, dan `System.getProperty("java.version")` menanyakan versi JVM yang sedang berjalan, hal pertama yang perlu diperiksa ketika sesuatu berperilaku aneh. Di terminal VS Code (Ctrl+`), masuk ke folder `java`, kompilasi, lalu jalankan:

```
$ cd java
$ javac -d out Hello.java
$ java -cp out Hello
Hello! My workbench for numerical methods is ready.
Java version: 25.0.4
Running on:   Windows 11
```

Opsi `-d out` menaruh `Hello.class` hasil kompilasi di folder bernama `out` agar folder sumber tetap rapi, dan `-cp out` (*class path*) memberi tahu `java` tempat menemukannya. Di macOS baris terakhir berbunyi `Mac OS X`, di Linux `Linux`. Tombol Run (4 pada **Gambar 1.2**) melakukan hal yang sama dengan satu klik.

JDK 25 juga mengizinkan program pertama yang jauh lebih pendek. Sejak JEP 512 menjadi final di JDK 25, sebuah berkas boleh hanya berisi metode `void main()` tanpa kelas yang membungkusnya, dan kelas bantu `IO` mencetak satu baris [20]:

```
void main() {
    IO.println("Hello!");
}
```

Meski begitu, program-program buku ini memakai bentuk klasik dengan kelas dan `public static void main`, karena kebanyakan terdiri atas beberapa kelas, dan karena bentuk klasik juga berjalan di JDK 17 dan 21.

1.8.2 Halo, dalam Python

Kode 1.2 adalah program yang sama dalam Python. Simpan sebagai `python/hello.py`.

Kode 1.2 `hello.py`: program Python pertama.

```
"""The first program: say hello and show which Python runs it."""
import platform

print("Hello! My workbench for numerical methods is ready.")
print("Python version: " + platform.python_version())
print("Running on:     " + platform.system())
```

Teks di antara tanda kutip tiga di bagian atas adalah *docstring*, keterangan tentang berkas. `import platform` memuat modul standar yang mengetahui seluk-beluk komputer dan interpreter. Tidak ada kelas dan tidak ada metode `main`: Python menjalankan berkas dari atas ke bawah.

```
$ cd ../python
$ python3 hello.py
Hello! My workbench for numerical methods is ready.
Python version: 3.14.7
Running on:     Linux
```

Di Windows ketik `python hello.py`, dan baris terakhir berbunyi `Windows`; di macOS berbunyi `Darwin`, nama inti macOS.

1.8.3 Memeriksa seluruh meja kerja

Daripada sekadar percaya bahwa semuanya berjalan lancar, kita menulis program yang memeriksa. **Kode 1.3** melaporkan versi Python, apakah lingkungan virtual aktif, apakah NumPy dan SciPy terpasang, dan apakah *runtime* Java dapat ditemukan; **Kode 1.4** melakukan hal yang sama dari sisi Java.

Kode 1.3 `check_setup.py`: memeriksa Python, lingkungan virtual, NumPy, SciPy, dan Java.

```
"""Check the workbench: Python, the virtual environment, NumPy, SciPy and Java."""
import importlib
import platform
import shutil
import subprocess
import sys

RECOMMENDED = (3, 14)      # the version this book is written for
MINIMUM = (3, 11)          # the oldest version the book's code runs on


def java_version():
    """Ask the 'java' command for its version; None when it is not found."""
    if shutil.which("java") is None:
        return None
    done = subprocess.run(["java", "-version"], capture_output=True, text=True)
    for line in (done.stderr + done.stdout).splitlines():  # java uses stderr
        if " version " in line and '"' in line:
            return line.split('"')[1]         # openjdk version "25.0.4" ...
    return "unknown"


def library_version(name):
    """The version of an installed library, or None when it is missing."""
    try:
        return importlib.import_module(name).__version__
    except ImportError:
        return None


def main():
    version = sys.version_info[:2]
    in_venv = sys.prefix != sys.base_prefix
    print(f"Python version : {platform.python_version()}")
    print(f"System         : {platform.system()} ({platform.machine()})")
    print(f"Virtual env.   : {'yes' if in_venv else 'no'}")
    for name in ("numpy", "scipy"):
        found = library_version(name)
        print(f"{name:<15}: {found or 'not installed (pip install ' + name + ')'}")
    java = java_version()
    print(f"Java version   : {java or 'NOT FOUND - install JDK 25'}")
    if version < MINIMUM:
        print("Result         : Python is too old, please install Python 3.14")
    elif java is None:
        print("Result         : Python is fine; now install the JDK")
    elif version < RECOMMENDED:
```

Kode 1.3 (Lanjutan) *check_setup.py: memeriksa Python, lingkungan virtual, NumPy, SciPy, dan Java.*

```
        print("Result         : works; Python 3.14 is recommended")
    else:
        print("Result         : your workbench is ready")


if __name__ == "__main__":
    main()
```

`sys.version_info[:2]` adalah versi sebagai pasangan bilangan seperti `(3, 14)`, dan Python membandingkan pasangan unsur demi unsur, sehingga `(3, 12) < (3, 14)` bernilai benar. `sys.prefix` dan `sys.base_prefix` berbeda tepat ketika lingkungan virtual aktif. `importlib.import_module(name)` memuat pustaka menurut namanya dan memunculkan `ImportError` bila pustaka itu tidak ada. `shutil.which("java")` mencari perintah `java` di `PATH`, dan `subprocess.run` menjalankan `java -version` sebagai program terpisah lalu menangkap apa

yang dicetaknya; Java mencetak versinya ke aliran *error*, sebuah kebiasaan historis, itulah sebabnya fungsi itu membaca `done.stderr`.

Kode 1.4 `CheckSetup.java`: memeriksa versi Java dan mencari Python.

```
/** Check the workbench: the Java version, and whether Python can be found. */
public class CheckSetup {
    static final int RECOMMENDED = 25;       // the JDK this book is written for
    static final int MINIMUM = 17;           // the oldest JDK the code runs on

    /** Ask a command for its version; null when the command is not found. */
    static String versionOf(String... command) {
        try {
            Process p = new ProcessBuilder(command).redirectErrorStream(true)
                                                   .start();
            String out = new String(p.getInputStream().readAllBytes()).trim();
            p.waitFor();
            return out.isEmpty() ? null : out.lines().findFirst().get();
        } catch (Exception e) {
            return null;                     // not installed, or not on PATH
        }
    }

    public static void main(String[] args) {
        int feature = Runtime.version().feature();
        System.out.println("Java version   : " + Runtime.version());
        System.out.println("Java vendor    : " + System.getProperty("java.vendor"));
        System.out.println("System         : " + System.getProperty("os.name")
                           + " (" + System.getProperty("os.arch") + ")");
        String python = versionOf("python3", "--version");
        if (python == null) {
            python = versionOf("python", "--version");  // the name on Windows
        }
        System.out.println("Python         : "
                           + (python != null ? python : "NOT FOUND"));
        if (feature < MINIMUM) {
            System.out.println("Result         : Java is too old, install JDK 25");
        } else if (python == null) {
            System.out.println("Result         : Java is fine; now install Python");
        } else if (feature < RECOMMENDED) {
            System.out.println("Result         : works; JDK 25 is recommended");
        } else {
            System.out.println("Result         : your workbench is ready");
        }
    }
}
```

`Runtime.version().feature()` adalah nomor versi utama Java yang sedang berjalan, 25 untuk JDK 25. `ProcessBuilder` menjalankan program lain; `redirectErrorStream(true)` menggabungkan aliran *error*-nya ke keluaran normal, sehingga versi ditemukan dari aliran mana pun yang dipakai Python. Karena Python di Windows biasanya bernama `python` dan di macOS serta Linux `python3`, program mencoba keduanya. Pada meja kerja yang sudah lengkap, dengan lingkungan virtual aktif, kedua program mencetak:

```
(.venv) $ python check_setup.py
Python version : 3.14.7
System         : Linux (x86_64)
Virtual env.   : yes
numpy          : 2.5.3
scipy          : 1.18.1
Java version   : 25.0.4
Result         : your workbench is ready
(.venv) $ java -cp out CheckSetup
Java version   : 25.0.4+8-LTS
Java vendor    : Eclipse Adoptium
System         : Linux (amd64)
Python         : Python 3.14.7
Result         : your workbench is ready
```

Nomor *build* persisnya berbeda dari komputer ke komputer. Baris terakhirlah yang perlu dilihat: jika bunyinya bukan "your workbench is ready", baris itu juga menyebutkan apa yang harus dipasang, dan program-program tetap berjalan selama bunyinya "works".

1.9 Mengukur Galat

Dengan meja kerja yang siap, kita beralih ke pokok bahasan buku ini. Sebelum dapat memperkecil galat, kita harus mampu mengukurnya, dan ada beberapa cara untuk itu.

1.9.1 Galat sejati dan galat relatif

Misalkan nilai sejati sebuah besaran adalah p dan kita mempunyai hampiran $\tilde{p}$ (dibaca "p tilde"). **Galat sejati** adalah selisih keduanya,

$$E_t = p - \tilde{p} \tag{1.1}$$

dan **galat mutlak** adalah besarnya, $|E_t|$. Galat mutlak saja bisa menyesatkan. Galat Rp1.000 pada nota Rp30.750 itu serius; galat Rp1.000 pada harga rumah tidak berarti apa-apa. Yang penting adalah galat dibandingkan dengan besar hal yang diukur, yaitu **galat relatif**, biasanya dinyatakan dalam persen:

$$\varepsilon_t = \frac{p - \tilde{p}}{p} \times 100\% \tag{1.2}$$

Berikut contoh yang dikerjakan. Bilangan $\pi = 3.14159265\ldots$ sering dihampiri dengan $22/7 = 3.14285714\ldots$ dan dengan $355/113 = 3.14159292\ldots$ Untuk $22/7$, $E_t = 3.14159265 - 3.14285714 = -0.00126449$, sehingga galat mutlaknya 0.00126 dan galat relatifnya $0.00126449/3.14159265 \times 100\% = 0.0402\%$. Untuk $355/113$, $E_t = -0.00000027$ dan $\varepsilon_t = -0.0000085\%$, hampir lima ribu kali lebih kecil. Para astronom Tiongkok sudah mengenal $355/113$ lima belas abad yang lalu. (Dalam rumus dan keluaran program, buku ini memakai titik sebagai pemisah desimal, sebagaimana komputer; dalam kalimat biasa dipakai koma.)

Gagasan yang berkaitan adalah banyaknya **angka bena** (*significant digits*): digit-digit $\tilde{p}$ yang dapat dipercaya. $\tilde{p}$ menghampiri p sampai n angka bena apabila galat relatifnya di bawah 5×10^{-n}, yaitu di bawah $0.5 \times 10^{2-n}$ persen [3]. $22/7$ mempunyai galat relatif 4.0×10^{-4}, di bawah 5×10^{-4}, jadi memberikan π sampai 4 angka bena (3,142 terhadap 3,14159); $355/113$ memberikan 7.

1.9.2 Galat hampiran dan aturan berhenti

Masalah pada Persamaan (1.2) adalah bahwa ia memerlukan nilai sejati p, dan kalau kita tahu p, kita tidak perlu menghitungnya. Namun kebanyakan metode numerik bersifat *iteratif*: metode itu menghasilkan barisan hampiran $\tilde{p}_1, \tilde{p}_2, \ldots$ yang makin lama makin baik. Maka perubahan dari satu hampiran ke hampiran berikutnya adalah ukuran yang baik tentang berapa banyak yang masih kita pelajari. **Galat relatif hampiran** membandingkan hampiran sekarang dengan hampiran sebelumnya [2]:

$$\varepsilon_a = \frac{\tilde{p}_{\text{new}} - \tilde{p}_{\text{old}}}{\tilde{p}_{\text{new}}} \times 100\% \tag{1.3}$$

Kita berhenti ketika $|\varepsilon_a|$ turun di bawah toleransi ε_s yang kita pilih sebelumnya. Jika kita menginginkan n angka bena yang benar, pilihan yang praktis, dikenal sebagai kriteria Scarborough, adalah

$$\varepsilon_s = 0.5 \times 10^{2-n}\ \% \tag{1.4}$$

sehingga $n = 3$ digit memberi $\varepsilon_s = 0.05\%$ dan $n = 6$ memberi 0.00005%. Kriteria ini bukan jaminan, karena barisan yang konvergen lambat bisa berubah sedikit saja dari langkah ke langkah padahal masih jauh dari limitnya (Bab 2 menjumpai persis hal ini pada regula falsi). Namun untuk metode yang konvergen dengan baik, aturan ini andal, dan bila galat hampiran sudah di bawah ε_s, hasilnya biasanya benar sampai sedikitnya n angka bena [2].

1.9.3 Dua jenis galat

Dari mana datangnya galat? Di luar kekeliruan (rumus yang salah, timbangan yang salah dibaca) dan data yang tidak pasti (timbangan Bu Wati hanya menampilkan satu desimal), ada dua jenis galat yang melekat pada komputasi numerik itu sendiri, dan buku ini menjumpai keduanya di setiap bab.

- **Galat pembulatan** (*round-off error*) berasal dari mesin. Komputer menyimpan setiap bilangan dengan banyak digit biner yang tetap, sehingga kebanyakan bilangan, seperti 2,3 dan 0,1, tersimpan sedikit keliru, dan setiap operasi aritmetika membulatkan hasilnya lagi. Subbab 1.10 menunjukkan caranya secara persis.
- **Galat pemotongan** (*truncation error*) berasal dari metode. Ketika kita mengganti proses tak berhingga dengan proses berhingga, misalnya menjumlahkan hanya suku-suku pertama sebuah deret tak hingga, atau mengganti turunan dengan selisih pada langkah kecil, kita memotong sesuatu. Itu akan terjadi bahkan pada mesin sempurna dengan digit tak terbatas.

Subbab 1.12 menyandingkan keduanya dan menunjukkan bahwa melawan yang satu sering kali memberi makan yang lain.

1.10 Bagaimana Mesin Menyimpan Bilangan

1.10.1 Pecahan biner

Komputer menyimpan bilangan dalam sistem biner, dengan digit 0 dan 1. Bilangan bulat mudah: $13 = 8 + 4 + 1 = 1101_2$. Pecahan bekerja dengan cara yang sama memakai pangkat negatif dari dua: $0.625 = \frac{1}{2} + \frac{1}{8} = 0.101_2$. Masalahnya, kebanyakan pecahan desimal tidak mempunyai bentuk

biner yang berhingga, persis seperti $1/3 = 0.333\ldots$ tidak mempunyai bentuk desimal berhingga. Untuk mencari digit biner 0,1, gandakan terus-menerus dan catat bagian bulatnya setiap kali:

$0.1 \times 2 = 0.2$ (digit 0), $0.2 \times 2 = 0.4$ (0), $0.4 \times 2 = 0.8$ (0), $0.8 \times 2 = 1.6$ (1, sisakan 0,6), $0.6 \times 2 = 1.2$ (1, sisakan 0,2), dan kini 0,2 muncul kembali, sehingga digit-digitnya berulang selamanya:

$$0.1 = 0.0\,0011\,0011\,0011\ldots_2 \tag{1.5}$$

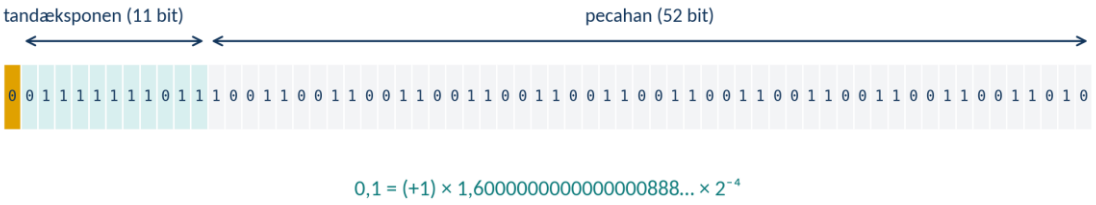
Mesin harus berhenti di suatu tempat, dan di mana pun ia berhenti, bilangan yang tersimpan bukanlah 0,1. Hal yang sama berlaku untuk 0,2, 0,3, 4,1, dan hampir setiap bilangan berkoma yang diketik orang.

1.10.2 *Double* IEEE 754

Sejak 1985, praktis setiap komputer menyimpan bilangan real menurut standar IEEE 754, yang revisi terakhirnya terbit pada 2019 [21]. `double` di Java dan `float` di Python keduanya adalah format **binary64** standar itu, yang disebut *presisi ganda*. Bilangan binary64 memakai 64 bit yang dibagi menjadi tiga medan, seperti diperlihatkan **Gambar 1.3** untuk bilangan 0,1: satu bit tanda s, sebelas bit eksponen yang memuat bilangan bulat e antara 0 dan 2047, dan 52 bit pecahan $f_1 f_2 \ldots f_{52}$. Untuk bilangan biasa (*normal*), nilainya adalah

$$x = (-1)^s \times (1.f_1 f_2 \ldots f_{52})_2 \times 2^{e-1023} \tag{1.6}$$

Angka 1 di depan titik biner tidak disimpan, karena setiap bilangan normal memilikinya; *bit tersembunyi* ini memberikan 53 digit biner bena dengan harga 52. Eksponen disimpan dengan *bias* 1023, sehingga $e = 1023$ berarti 2^0, dan bilangan kecil mendapat eksponen di bawah 1023. Nilai ekstrem $e = 0$ dan $e = 2047$ dicadangkan untuk nol, bilangan *subnormal* yang sangat kecil, tak hingga, dan NaN ("*not a number*", hasil $0/0$).



Gambar 1.3 Ke-64 bit *double* terdekat dengan 0,1: tanda 0, eksponen 01111111011 (1019, berarti 2^{-4}), dan 52 bit pecahan tempat pola 1001 berulang sampai dipotong dan dibulatkan.

Mari kita urai **Gambar 1.3** dengan tangan. Bit tandanya 0, jadi bilangannya positif. Bit eksponen 01111111011_2 bernilai 1019, dan $1019 - 1023 = -4$. Pecahannya dimulai dengan 1001 1001 1001 ..., sehingga signifikannya $1.1001\,1001\ldots_2 = 1.6000\ldots$, dan $1.6 \times 2^{-4} = 0.1$. Bit-bit terakhirnya ...1010, bukan ...1001 seperti lanjutan polanya, karena ekor tak hingga 1001 1001 ... dibulatkan *ke atas*. Itulah sebabnya nilai yang tersimpan sedikit *lebih besar* daripada 0,1, seperti akan dicetak lengkap oleh program pertama Subbab 1.11.

1.10.3 Epsilon mesin dan batas pembulatan

Seberapa jauh jarak dua *double* yang bertetangga? Di antara 1 dan 2 eksponennya tetap, dan 52 bit pecahan menghitung dengan langkah 2^{-52}. Jarak antara 1 dan *double* berikutnya yang lebih besar ini adalah **epsilon mesin**,

$$\varepsilon_{\text{mach}} = 2^{-52} \approx 2.22 \times 10^{-16} \tag{1.7}$$

Di antara 2 dan 4 jaraknya $2\varepsilon_{\text{mach}}$, di antara 1024 dan 2048 jaraknya $1024\,\varepsilon_{\text{mach}}$: *double* tersebar seperti penggaris yang tanda-tandanya makin renggang ketika bilangannya makin besar, tetapi selalu dengan jarak *relatif* yang sama. Ketika bilangan real x disimpan, ia dibulatkan ke *double* terdekat $\text{fl}(x)$, yang jaraknya paling jauh setengah celah. Dalam ukuran relatif,

$$\text{fl}(x) = x\,(1+\delta), \qquad |\delta| \le u = \frac{1}{2}\varepsilon_{\text{mach}} = 2^{-53} \approx 1.11 \times 10^{-16} \tag{1.8}$$

Bilangan u disebut **satuan pembulatan** (*unit round-off*). IEEE 754 juga mensyaratkan bahwa setiap operasi $+$, $-$, $\times$, $\div$, dan akar kuadrat memberikan hasil yang dibulatkan secara tepat: komputer berperilaku seolah-olah menghitung jawaban eksak lalu membulatkannya sekali [21]. Jadi setiap operasi tunggal mempunyai galat relatif paling besar u, sekitar 16 digit desimal bena. Persamaan (1.8) adalah fakta terpenting dalam bab ini, dan analisis galat di seluruh buku berangkat darinya [22].

Standar itu juga mendefinisikan format lain. **Tabel 1.2** membandingkan dua format yang langsung tersedia di Java dan Python; `float` di Java adalah binary32, dan di Python tersedia melalui `float32` dari NumPy.

Tabel 1.2 Dua format IEEE 754 yang dipakai dalam buku ini.

	Binary32 (`float`)	Binary64 (`double`)
Bit: tanda, eksponen, pecahan	1, 8, 23	1, 11, 52
Digit biner bena	24	53
Epsilon mesin	$2^{-23} \approx 1.19 \times 10^{-7}$	$2^{-52} \approx 2.22 \times 10^{-16}$
Digit desimal, kira-kira	7	16
Bilangan terbesar, kira-kira	3.4×10^{38}	1.8×10^{308}
Bilangan bulat tersimpan tepat sampai	$2^{24} = 16{,}777{,}216$	$2^{53} \approx 9.0 \times 10^{15}$

Baris terakhir menjelaskan Rp826 yang hilang di Subbab 1.1. Sebuah `float` menyimpan bilangan bulat secara tepat hanya sampai sekitar 16,8 juta. Nota Bu Wati selama setahun berjumlah lebih dari 700 juta rupiah, tempat jarak antara dua `float` bertetangga adalah 64 rupiah: program tablet itu bahkan tidak *mampu menuliskan* total yang benar.

1.11 Lima Percobaan dengan FloatLab

Teori menjadi meyakinkan ketika mesin membenarkannya. **Kode 1.5** menjalankan lima percobaan kecil; **Kode 1.6** adalah program yang sama dalam Java.

Kode 1.5 `float_lab.py`: lima percobaan dengan bilangan presisi ganda.

```python
"""Five small experiments that show how a computer stores real numbers."""
from decimal import Decimal
import math


def main():
    print("1. Adding 0.1 and 0.2")
    s = 0.1 + 0.2
    print(f"   0.1 + 0.2        = {s!r}")
    print(f"   0.1 + 0.2 == 0.3 : {s == 0.3}")

    print("2. What is really stored for 0.1")
    print(f"   {Decimal(0.1)}")        # the exact value of the double

    print("3. Machine epsilon, found by halving")
    eps = 1.0
    while 1.0 + eps / 2 > 1.0:     # stop when half of eps is no longer seen
        eps /= 2
    print(f"   eps              = {eps:.6e}")
    print(f"   1 + eps/2 == 1   : {1.0 + eps / 2 == 1.0}")

    print("4. Adding 0.1 again and again")
    for n in (10, 1000, 1000000):
        total = 0.0
        for _ in range(n):
            total += 0.1
        exact = n / 10
        print(f"   {n:>7} times: {total!r:<22} error {abs(total - exact):.3e}")

    print("5. Subtracting nearly equal numbers: x^2 - 1e8 x + 1 = 0")
    a, b, c = 1.0, -1.0e8, 1.0
    root = math.sqrt(b * b - 4 * a * c)
    naive = (-b - root) / (2 * a)          # small root, textbook formula
    big = (-b + root) / (2 * a)            # large root: no cancellation
    stable = c / (a * big)                 # small root from x1 * x2 = c / a
    print(f"   textbook formula : {naive:.10e}")
    print(f"   stable formula   : {stable:.10e}")
    print(f"   relative error of the textbook formula: "
          f"{abs(naive - stable) / stable:.3e}")


if __name__ == "__main__":
    main()
```

Kode 1.6 `FloatLab.java`: percobaan yang sama dalam Java.

```java
import java.math.BigDecimal;
import java.util.Locale;

/** Five small experiments that show how a computer stores real numbers. */
public class FloatLab {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);         // a dot as decimal point
        System.out.println("1. Adding 0.1 and 0.2");
        double s = 0.1 + 0.2;
        System.out.println("   0.1 + 0.2        = " + s);
        System.out.println("   0.1 + 0.2 == 0.3 : "
                           + (s == 0.3 ? "True" : "False"));

        System.out.println("2. What is really stored for 0.1");
        System.out.println("   " + new BigDecimal(0.1));  // exact value

        System.out.println("3. Machine epsilon, found by halving");
        double eps = 1.0;
        while (1.0 + eps / 2 > 1.0) {   // stop when half of eps is not seen
            eps /= 2;
        }
        System.out.printf("   eps              = %.6e%n", eps);
        System.out.println("   1 + eps/2 == 1   : "
                           + (1.0 + eps / 2 == 1.0 ? "True" : "False"));

        System.out.println("4. Adding 0.1 again and again");
        for (int n : new int[] {10, 1000, 1000000}) {
            double total = 0.0;
            for (int i = 0; i < n; i++) {
                total += 0.1;
            }
            double exact = n / 10.0;
            System.out.printf("   %7d times: %-22s error %.3e%n", n,
                              Double.toString(total), Math.abs(total - exact));
        }
```

Kode 1.6 (Lanjutan) *FloatLab.java: percobaan yang sama dalam Java.*

```java
        System.out.println("5. Subtracting nearly equal numbers: "
                           + "x^2 - 1e8 x + 1 = 0");
        double a = 1.0, b = -1.0e8, c = 1.0;
        double root = Math.sqrt(b * b - 4 * a * c);
        double naive = (-b - root) / (2 * a);    // small root, textbook formula
        double big = (-b + root) / (2 * a);      // large root: no cancellation
        double stable = c / (a * big);           // small root from x1 x2 = c/a
        System.out.printf("   textbook formula : %.10e%n", naive);
        System.out.printf("   stable formula   : %.10e%n", stable);
        System.out.printf("   relative error of the textbook formula: %.3e%n",
                          Math.abs(naive - stable) / stable);
    }
}
```

Dua hal pada versi Java perlu diperhatikan. `Locale.setDefault(Locale.ROOT)` membuat `printf` menulis titik desimal, bukan koma desimal yang akan dipakai komputer berbahasa Indonesia; setiap program Java buku ini diawali baris ini agar keluarannya sama di mana pun. Dan `new BigDecimal(0.1)` mengubah *double* itu *secara eksak* menjadi bilangan desimal dengan digit sebanyak yang diperlukan, sama seperti `Decimal(0.1)` di Python. Jalankan salah satu versi:

```
$ python3 float_lab.py          (or: java -cp out FloatLab)
```

```
1. Adding 0.1 and 0.2
   0.1 + 0.2        = 0.30000000000000004
   0.1 + 0.2 == 0.3 : False
2. What is really stored for 0.1
   0.1000000000000000055511151231257827021181583404541015625
3. Machine epsilon, found by halving
   eps              = 2.220446e-16
   1 + eps/2 == 1   : True
4. Adding 0.1 again and again
        10 times: 0.9999999999999999     error 1.110e-16
      1000 times: 99.9999999999986       error 1.407e-12
   1000000 times: 100000.00000133288     error 1.333e-06
5. Subtracting nearly equal numbers: x^2 - 1e8 x + 1 = 0
   textbook formula : 7.4505805969e-09
   stable formula   : 1.0000000000e-08
   relative error of the textbook formula: 2.549e-01
```

Kedua program mencetak baris-baris yang persis sama. Mari kita baca satu per satu.

Percobaan 1. $0.1 + 0.2$ menghasilkan `0.30000000000000004`, dan tidak sama dengan `0.3`. Baik 0,1 maupun 0,2 tersimpan sedikit terlalu besar, jumlahnya dibulatkan sekali lagi, dan hasilnya jatuh pada *double* tepat di atas *double* terdekat dengan 0,3. Menurut IEEE 754, tidak ada jawaban yang "salah"; keduanya hanyalah *double* yang berbeda. Pelajarannya: `==` di antara dua *double* hasil hitungan hampir selalu merupakan kekeliruan. Bandingkan dengan toleransi: $|a - b| \le \tau \max(|a|, |b|)$ untuk toleransi relatif τ seperti 10^{-12}.

Percobaan 2. *Double* terdekat dengan 0,1 tepatnya `0.1000000000000000055511151231257827021181583404541015625`. Galatnya 5.55×10^{-18}, galat relatifnya 5.55×10^{-17}, di bawah batas $u = 1.11 \times 10^{-16}$ dari Persamaan (1.8), sebagaimana mestinya.

Percobaan 3. Membagi dua bilangan `eps` sampai $1 + \text{eps}/2$ tidak lagi lebih besar daripada 1 menemukan 2.220446×10^{-16}, epsilon mesin dari Persamaan (1.7). Program menemukan sifat perangkat keras lewat percobaan.

Percobaan 4. Menjumlahkan 0,1 sepuluh kali memberikan `0.9999999999999999`, satu celah di bawah 1. Setelah seribu penjumlahan galatnya 1.4×10^{-12}, dan setelah sejuta penjumlahan 1.3×10^{-6}. Setiap penjumlahan membulatkan, dan galat pembulatan itu tidak saling meniadakan: galat tumbuh bersama banyaknya operasi. Inilah jam Patriot dari Subbab 1.1 dalam ukuran kecil. Batas yang cermat: n penjumlahan bilangan berukuran sampai S dapat menghasilkan galat sampai sekitar $n\,u\,S$; untuk $n = 10^6$ dan jumlah berjalan sampai 10^5, itu $10^6 \times 1.1 \times 10^{-16} \times 10^5 \approx 10^{-5}$, dan galat terukur berada jauh di dalamnya [22].

Percobaan 5 adalah yang terpenting. Persamaan $x^2 - 10^8 x + 1 = 0$ mempunyai dua akar, satu di dekat 10^8 dan satu di dekat 10^{-8}. Rumus sekolah untuk akar kecil adalah

$$x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a} \tag{1.9}$$

dengan $b = -10^8$, sehingga rumus itu menghitung $10^8 - \sqrt{10^{16} - 4}$: selisih dua bilangan yang sama pada 16 digit pertamanya. Akar kuadratnya benar sampai sekitar 16 digit, galatnya sekitar $10^8 \times 1.1 \times 10^{-16} \approx 10^{-8}$, dan galat itu sebesar jawabannya sendiri. Pengurangannya sendiri eksak; ia hanya menyingkap sampah yang ditaruh pembulatan di digit-digit terakhir. Gejala ini disebut **pembatalan katastrofik** (*catastrophic cancellation*), dan hasil `7.45e-09` mempunyai galat relatif 25%. Obatnya adalah menghindari pengurangan itu. Akar besar x_1 dihitung dengan

menjumlahkan bilangan-bilangan bertanda sama, yang aman, dan hasil kali kedua akar $ax^2 + bx + c$ adalah c/a (rumus Vieta), sehingga

$$x_2 = \frac{c}{a\,x_1} \tag{1.10}$$

hanya memerlukan satu perkalian dan satu pembagian, masing-masing teliti sampai u. Rumus yang stabil memberikan `1.0000000000e-08`, benar pada semua digit yang dicetak. Persamaan (1.9) dan Persamaan (1.10) sama dalam aljabar tetapi sangat berbeda di komputer. Itulah arti *kestabilan numerik*.

1.12 Galat Pembulatan dan Galat Pemotongan Bersama-sama

1.12.1 Deret Taylor dan pemotongan

Banyak fungsi dihitung dari **deret Taylor**-nya. Jika fungsi f mempunyai turunan semua orde di sekitar titik a, maka untuk x di dekat a,

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n + R_n \tag{1.11}$$

dengan sisa $R_n = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-a)^{n+1}$ untuk suatu ξ di antara a dan x [3]. Mempertahankan suku-suku sampai $(x-a)^n$ dan membuang R_n adalah *pemotongan*, dan R_n adalah galat pemotongannya. Untuk $f(x) = e^x$ di $a = 0$, setiap turunannya $e^0 = 1$, sehingga

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \tag{1.12}$$

Mari kita hitung $e^{0.5}$ sampai tiga angka bena dengan tangan, berhenti menurut Persamaan (1.4): $\varepsilon_s = 0.5 \times 10^{2-3} = 0.05\%$. Nilai sejatinya $e^{0.5} = 1.648721\ldots$ Satu suku memberi 1. Dua suku memberi $1 + 0.5 = 1.5$, dengan $\varepsilon_a = (1.5-1)/1.5 = 33.3\%$. Tiga suku menambah $0.5^2/2 = 0.125$: 1.625, $\varepsilon_a = 0.125/1.625 = 7.69\%$. Empat suku menambah $0.5^3/6 = 0.0208333$: 1.6458333, $\varepsilon_a = 1.27\%$. Lima suku menambah $0.5^4/24 = 0.0026042$: 1.6484375, $\varepsilon_a = 0.158\%$. Enam suku menambah $0.5^5/120 = 0.00026042$: 1.6486979, $\varepsilon_a = 0.0158\%$, di bawah 0,05%, jadi kita berhenti. Galat relatif sejatinya $(1.648721 - 1.648698)/1.648721 = 0.0014\%$: lebih baik daripada yang dijanjikan. **Kode 1.7** dan **Kode 1.8** melakukan hal yang sama dan juga mencoba sesuatu yang lebih sulit.

Kode 1.7 `exp_series.py`: e^x dari deret Taylor-nya dengan aturan berhenti Persamaan (1.4).

```python
"""e^x from its Taylor series, stopped by the tolerance for n significant digits."""
import math


def exp_series(x, digits, show=False):
    """Sum 1 + x + x^2/2! + ... until the approximate relative error is small."""
    tol = 0.5 * 10 ** (2 - digits)          # percent (Scarborough criterion)
    total, term, k = 1.0, 1.0, 0
    if show:
        print(f"{'terms':>5} {'sum':>14} {'true err %':>12} {'approx err %':>13}")
        true = math.exp(x)
        print(f"{1:>5} {total:>14.9f} {abs(true - total) / true * 100:>12.6f}")
    while True:
        k += 1
        term *= x / k                        # x^k / k! from the previous term
        old, total = total, total + term
        approx = abs((total - old) / total) * 100
        if show:
            err = abs(true - total) / true * 100
            print(f"{k + 1:>5} {total:>14.9f} {err:>12.6f} {approx:>13.6f}")
        if approx < tol:
            return total, k + 1


def main():
    print("e^0.5 to 3 significant digits (tolerance 0.05 %):")
    exp_series(0.5, 3, show=True)
    print()
    print("e^-20 in three ways:")
    direct, n = exp_series(-20.0, 15)
    inverse, _ = exp_series(20.0, 15)
    print(f"  series at x = -20 ({n} terms): {direct:.10e}")
    print(f"  1 / series at x = 20         : {1 / inverse:.10e}")
    print(f"  library exp(-20)             : {math.exp(-20.0):.10e}")


if __name__ == "__main__":
    main()
```

Kode 1.8 `ExpSeries.java`: deret yang sama dalam Java.

```java
import java.util.Locale;

/** e^x from its Taylor series, stopped by the tolerance for n digits. */
public class ExpSeries {
    static int terms;                       // terms used by the last call

    /** Sum 1 + x + x^2/2! + ... until the approximate error is small. */
    static double expSeries(double x, int digits, boolean show) {
        double tol = 0.5 * Math.pow(10, 2 - digits);  // percent (Scarborough)
        double total = 1.0, term = 1.0, t = Math.exp(x);
        int k = 0;
        if (show) {
            System.out.printf("%5s %14s %12s %13s%n", "terms", "sum",
                              "true err %", "approx err %");
            System.out.printf("%5d %14.9f %12.6f%n", 1, total,
                              Math.abs(t - total) / t * 100);
        }
        while (true) {
            k++;
            term *= x / k;                  // x^k / k! from the previous term
            double old = total;
            total = total + term;
            double approx = Math.abs((total - old) / total) * 100;
            if (show) {
                double err = Math.abs(t - total) / t * 100;
                System.out.printf("%5d %14.9f %12.6f %13.6f%n", k + 1, total,
                                  err, approx);
            }
            if (approx < tol) {
                terms = k + 1;
                return total;
            }
        }
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.println("e^0.5 to 3 significant digits (tolerance 0.05 %):");
        expSeries(0.5, 3, true);
        System.out.println();
        System.out.println("e^-20 in three ways:");
        double direct = expSeries(-20.0, 15, false);
        int n = terms;
        double inverse = expSeries(20.0, 15, false);
        System.out.printf("  series at x = -20 (%d terms): %.10e%n", n, direct);
        System.out.printf("  1 / series at x = 20         : %.10e%n", 1 / inverse);
```

Kode 1.8 (Lanjutan) *ExpSeries.java: deret yang sama dalam Java.*

```java
        System.out.printf("  library exp(-20)             : %.10e%n",
                          Math.exp(-20.0));
    }
}
```

Perulangan menghitung setiap suku dari suku sebelumnya, $x^k/k! = \frac{x^{k-1}}{(k-1)!} \cdot \frac{x}{k}$, alih-alih menghitung pangkat dan faktorial secara terpisah, yang lambat dan, untuk k besar, bisa meluap. Keluarannya:

```
e^0.5 to 3 significant digits (tolerance 0.05 %):
terms            sum   true err %  approx err %
    1    1.000000000    39.346934
    2    1.500000000     9.020401     33.333333
    3    1.625000000     1.438768      7.692308
    4    1.645833333     0.175162      1.265823
    5    1.648437500     0.017212      0.157978
    6    1.648697917     0.001416      0.015795

e^-20 in three ways:
  series at x = -20 (95 terms): 6.1475618289e-09
  1 / series at x = 20        : 2.0611536224e-09
  library exp(-20)            : 2.0611536224e-09
```

Tabel itu cocok baris demi baris dengan hitungan tangan kita, dan inilah contoh klasik Chapra dan Canale [2]. Bagian keduanya adalah peringatan. Di $x = -20$ deret itu mempunyai suku-suku sebesar $20^{20}/20! \approx 4.3 \times 10^{7}$ dengan tanda berselang-seling, yang harus saling meniadakan sampai tinggal $e^{-20} \approx 2.06 \times 10^{-9}$. Setiap suku besar membawa galat pembulatan sekitar $4.3 \times 10^{7} \times 1.1 \times 10^{-16} \approx 5 \times 10^{-9}$, lebih besar daripada jawabannya, dan jumlah `6.1e-09` keliru tiga kali lipat walaupun tidak ada suku yang terpotong terlalu dini. Cara yang stabil adalah menjumlahkan deret di $+20$, tempat semua suku positif dan tidak ada yang saling meniadakan, lalu mengambil kebalikannya; hasilnya cocok dengan pustaka pada kesepuluh digitnya. Masalahnya di sini bukan pemotongan, melainkan pembulatan yang diperbesar oleh pembatalan.

1.12.2 Dilema ukuran langkah

Turunan $f'(x)$ adalah kemiringan f di x. Rumus Taylor Persamaan (1.11) dengan $a = x$ dan $x + h$ sebagai pengganti x memberikan $f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2$, dan bila diselesaikan untuk $f'(x)$,

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(\xi)}{2}\,h \tag{1.13}$$

Pecahan itu adalah **selisih maju** (*forward difference*), dan suku terakhir adalah galat pemotongannya, sebanding dengan h: langkah dibagi dua, galat pemotongan pun terbagi dua. Jadi h yang sangat kecil semestinya memberi turunan yang sempurna. Program **Kode 1.9** mencobanya untuk $f(x) = \sqrt{x}$ di $x = 2$, tempat turunan eksaknya $1/\left(2\sqrt{2}\right) = 0.35355339\ldots$, dengan $h = 10^{-1}, 10^{-2}, \ldots, 10^{-16}$.

Kode 1.9 `diff_step.py`: selisih maju $\sqrt{x}$ dengan langkah yang makin kecil.

```python
"""Truncation against round-off: the forward difference of sqrt(x) at x = 2."""
import math


def main():
    x = 2.0
    true = 0.5 / math.sqrt(x)             # the exact derivative 1 / (2 sqrt x)
    print(f"{'h':>8} {'estimate':>20} {'error':>11}")
    for p in range(1, 17):
        h = 1.0 / 10 ** p                 # 10^p is exact for p <= 22
        estimate = (math.sqrt(x + h) - math.sqrt(x)) / h
        print(f"{h:>8.0e} {estimate:>20.15f} {abs(estimate - true):>11.3e}")


if __name__ == "__main__":
    main()
```

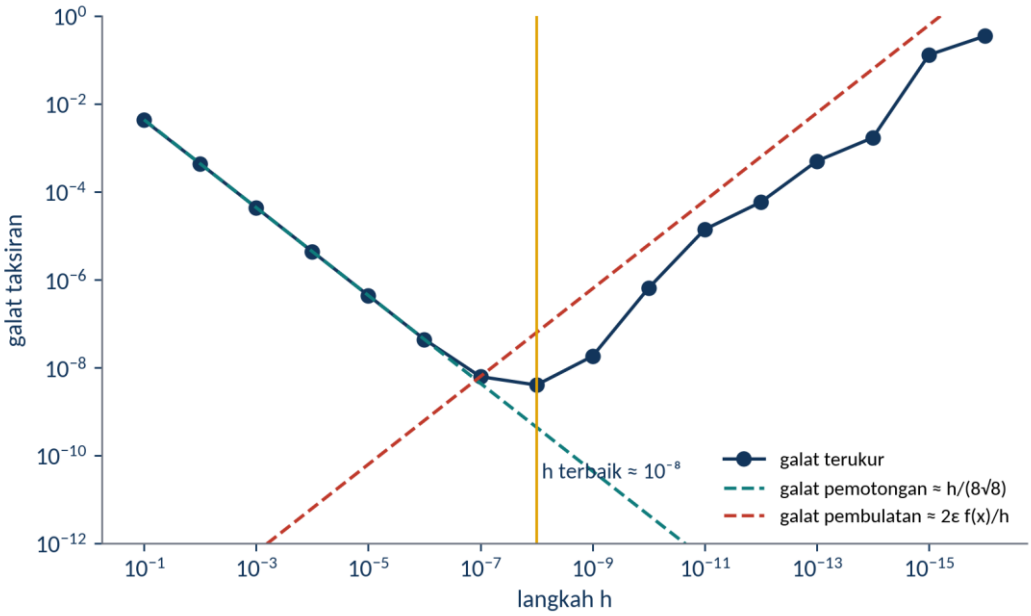
Kode 1.10 `DiffStep.java`: percobaan yang sama dalam Java.

```java
import java.util.Locale;

/** Truncation against round-off: the forward difference of sqrt(x) at x = 2. */
public class DiffStep {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        double x = 2.0;
        double exact = 0.5 / Math.sqrt(x);   // the derivative 1 / (2 sqrt x)
        System.out.printf("%8s %20s %11s%n", "h", "estimate", "error");
        for (int p = 1; p <= 16; p++) {
            double h = 1.0 / Math.pow(10, p);    // 10^p is exact for p <= 22
            double estimate = (Math.sqrt(x + h) - Math.sqrt(x)) / h;
            System.out.printf("%8.0e %20.15f %11.3e%n", h, estimate,
                              Math.abs(estimate - exact));
        }
    }
}
```

Pada **Kode 1.9** dan **Kode 1.10**, langkahnya dihitung sebagai `1.0 / 10 ** p` di Python dan `1.0 / Math.pow(10, p)` di Java, bukan langsung sebagai 10^{-p}: 10^{p} adalah bilangan bulat yang dapat disimpan tepat oleh *double* (sampai 10^{22}), sehingga pembagiannya dibulatkan sekali di kedua bahasa dan kedua program memakai langkah yang persis sama.

```
       h             estimate       error
   1e-01    0.349241122458488   4.312e-03
   1e-02    0.353112550268730   4.408e-04
   1e-03    0.353509207464464   4.418e-05
   1e-04    0.353548971285811   4.419e-06
   1e-05    0.353552948650915   4.419e-07
   1e-06    0.353553346377211   4.422e-08
   1e-07    0.353553384346839   6.246e-09
   1e-08    0.353553386567285   4.026e-09
   1e-09    0.353553408771745   1.818e-08
   1e-10    0.353552742637930   6.480e-07
   1e-11    0.353539419961635   1.397e-05
   1e-12    0.353495011040650   5.838e-05
   1e-13    0.353050921830800   5.025e-04
   1e-14    0.355271367880050   1.718e-03
   1e-15    0.222044604925031   1.315e-01
   1e-16    0.000000000000000   3.536e-01
```

Sampai $h = 10^{-7}$ galatnya turun sepuluh kali pada setiap langkah, sesuai janji Persamaan (1.13). Lalu galat berbalik, membesar, dan pada $h = 10^{-16}$ taksirannya tepat 0: $2 + 10^{-16}$ dibulatkan menjadi 2, karena 10^{-16} kurang dari setengah celah antar-*double* di dekat 2. **Gambar 1.4** memperlihatkan kurva berbentuk V pada sumbu logaritmik.



Gambar 1.4 Galat selisih maju $\sqrt{x}$ di $x = 2$ terhadap langkah h: galat pemotongan turun bersama h, galat pembulatan tumbuh sebanding $1/h$, dan totalnya paling kecil di dekat $h = 10^{-8}$.

Penjelasannya menggabungkan kedua jenis galat. Setiap nilai f yang dihitung membawa galat pembulatan sekitar $\varepsilon_{\text{mach}}|f(x)|$ (termasuk pembulatan $x + h$), sehingga selisih keduanya bisa keliru sampai $2\varepsilon_{\text{mach}}|f(x)|$, dan pembagian dengan h mengalikannya dengan $1/h$. Dengan $M = |f''(x)|$, galat totalnya kira-kira

$$E(h) \approx \frac{M}{2}\,h + \frac{2\,\varepsilon_{\text{mach}}\,|f(x)|}{h} \tag{1.14}$$

Suku pertama turun dan suku kedua naik ketika h mengecil. Menyamakan turunan $E'(h) = M/2 - 2\varepsilon_{\text{mach}}|f|/h^2$ dengan nol memberikan langkah terbaik,

$$h_{\text{opt}} = 2\sqrt{\frac{\varepsilon_{\text{mach}}\,|f(x)|}{M}} \tag{1.15}$$

Untuk $\sqrt{x}$ di 2, $f = \sqrt{2} = 1.414$ dan $M = \frac{1}{4}x^{-3/2} = 0.0884$, sehingga $h_{\text{opt}} = 2\sqrt{2.22 \times 10^{-16} \times 1.414/0.0884} \approx 1.2 \times 10^{-7}$, dan galat terkecilnya sekitar $Mh_{\text{opt}} \approx 10^{-8}$. Pengukuran sesuai dalam orde besarnya: langkah terbaik adalah 10^{-7} dan 10^{-8}, dengan galat 6 dan 4 per semiliar. Persamaan (1.14) adalah batas, dan galat pembulatan yang sebenarnya sebagian saling meniadakan, itulah sebabnya kurva terukur berada di bawah garis merah. Aturan praktisnya patut diingat: untuk selisih maju dalam presisi ganda, ambil h sekitar $\sqrt{\varepsilon_{\text{mach}}} \approx 10^{-8}$ kali ukuran x, jangan pernah "sekecil mungkin". Bab 9 membangun rumus selisih yang lebih baik, yang galat pemotongannya turun seperti h^2 atau h^4 sehingga langkah terbaiknya lebih besar.

1.13 Buku Kas *Laundry*: Menjumlahkan Uang dengan Aman

Kembali ke Bu Wati. Kini kita dapat menyimulasikan setahun usahanya dan melihat persis ke mana Rp826 di tablet itu pergi. **Kode 1.11** dan kembarannya dalam Java, **Kode 1.12**,

membangkitkan 365 hari dengan masing-masing 60 pesanan, dengan berat dari 1,0 kg sampai 8,0 kg dalam langkah 100 g, lalu menjumlahkan harganya dengan empat cara.

Kode 1.11 `laundry_ledger.py`: nota *laundry* setahun dijumlahkan sebagai bilangan bulat, sebagai *double*, sebagai *float*, dan sebagai *float* dengan kompensasi Kahan.

```
"""A year of laundry receipts added up in four ways.

The receipts are generated by a small random-number generator that gives the
same numbers in Java and Python, so both programs print the same report."""

PRICE_PER_KG = 7500           # rupiah per kilogram
DAYS = 365
ORDERS_PER_DAY = 60


def lcg(seed):
    """A linear congruential generator: the same sequence in every language."""
    while True:
        seed = (1103515245 * seed + 12345) % 2147483648
        yield seed


def to_float32(x):
    """Round a Python float (a double) to the nearest single-precision value."""
    import struct
    return struct.unpack("f", struct.pack("f", x))[0]


def main():
    rnd = lcg(2026)
    exact = 0                  # whole rupiah, as an integer: no error at all
    naive64 = 0.0              # double precision, simple running sum
    naive32 = 0.0              # single precision, simple running sum
    kahan32, carry = 0.0, 0.0  # single precision with Kahan's compensation
    for _ in range(DAYS * ORDERS_PER_DAY):
        grams = 1000 + (next(rnd) % 71) * 100     # 1.0 kg to 8.0 kg
        rupiah = grams * PRICE_PER_KG // 1000     # exact: grams is 100s
        kg = grams / 1000.0
        exact += rupiah
        naive64 += kg * PRICE_PER_KG
        naive32 = to_float32(naive32 + to_float32(kg * PRICE_PER_KG))
        y = to_float32(to_float32(kg * PRICE_PER_KG) - carry)
        t = to_float32(kahan32 + y)
        carry = to_float32(to_float32(t - kahan32) - y)   # what was lost
        kahan32 = t
    print(f"Orders in the year     : {DAYS * ORDERS_PER_DAY:,}")
    print(f"Exact total (integer)  : Rp {exact:,}")
    for name, value in (("double, simple sum", naive64),
                        ("float, simple sum", naive32),
                        ("float, Kahan sum", kahan32)):
        print(f"{name:<23}: Rp {value:,.2f}  (off by {value - exact:+,.2f})")
```

Kode 1.11 (Lanjutan) *laundry_ledger.py: nota laundry setahun dijumlahkan sebagai bilangan bulat, sebagai double, sebagai float, dan sebagai float dengan kompensasi Kahan.*

```


if __name__ == "__main__":
    main()
```

Kode 1.12 `LaundryLedger.java`: buku kas yang sama dalam Java.

```java
import java.util.Locale;

/**
 * A year of laundry receipts added up in four ways. The receipts come from a
 * small random-number generator that gives the same numbers in Java and
 * Python, so both programs print the same report.
 */
public class LaundryLedger {
    static final long PRICE_PER_KG = 7500;    // rupiah per kilogram
    static final int DAYS = 365;
    static final int ORDERS_PER_DAY = 60;
    static long seed = 2026;

    /** A linear congruential generator: the same sequence in every language. */
    static long next() {
        seed = (1103515245L * seed + 12345L) % 2147483648L;
        return seed;
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        long exact = 0;             // whole rupiah, as an integer: no error
        double naive64 = 0.0;       // double precision, simple running sum
        float naive32 = 0.0f;       // single precision, simple running sum
        float kahan32 = 0.0f;       // single precision with Kahan's method
        float carry = 0.0f;
        for (int i = 0; i < DAYS * ORDERS_PER_DAY; i++) {
            long grams = 1000 + (next() % 71) * 100;     // 1.0 kg to 8.0 kg
            long rupiah = grams * PRICE_PER_KG / 1000;   // exact: 100s of g
            double kg = grams / 1000.0;
            exact += rupiah;
            naive64 += kg * PRICE_PER_KG;
            naive32 = naive32 + (float) (kg * PRICE_PER_KG);
            float y = (float) (kg * PRICE_PER_KG) - carry;
            float t = kahan32 + y;
            carry = (t - kahan32) - y;                   // what was lost
            kahan32 = t;
        }
        System.out.printf("Orders in the year     : %,d%n", DAYS * ORDERS_PER_DAY);
        System.out.printf("Exact total (integer)  : Rp %,d%n", exact);
        String[] names = {"double, simple sum", "float, simple sum",
                          "float, Kahan sum"};
        double[] values = {naive64, naive32, kahan32};
        for (int k = 0; k < names.length; k++) {
            System.out.printf("%-23s: Rp %,.2f  (off by %+,.2f)%n", names[k],
                              values[k], values[k] - exact);
```

Kode 1.12 (Lanjutan) *LaundryLedger.java: buku kas yang sama dalam Java.*

```java
        }
    }
}
```

Nota-nota itu berasal dari **pembangkit kongruensial linear** (*linear congruential generator*), yaitu aturan $s_{k+1} = (1103515245\, s_k + 12345) \bmod 2^{31}$, bukan dari bilangan acak bawaan bahasa, karena `Random` di Java dan `random` di Python menghasilkan barisan yang berbeda sedangkan kita ingin kedua program melihat tahun yang sama. Python tidak mempunyai tipe presisi tunggal di pustaka standarnya, sehingga `to_float32` membulatkan sebuah *double* ke *float* terdekat dengan mengemasnya menjadi empat bita memakai `struct` lalu membongkarnya kembali. Di Java, `(float)` melakukan hal yang sama. Karena IEEE 754 membulatkan setiap operasi secara tepat, jumlah *float* yang dihitung dalam *double* lalu dibulatkan ke *float* sama dengan jumlah *float* yang dihitung langsung, dan kedua program sepakat:

```
Orders in the year     : 21,900
Exact total (integer)  : Rp 739,304,250
double, simple sum     : Rp 739,304,250.00  (off by +0.00)
float, simple sum      : Rp 739,303,424.00  (off by -826.00)
float, Kahan sum       : Rp 739,304,256.00  (off by +6.00)
```

Total bilangan bulat eksak menurut konstruksinya. Berat dalam gram adalah bilangan bulat, grams $\times$ 7500/1000 adalah bilangan bulat rupiah karena gramnya kelipatan 100, dan bilangan bulat dijumlahkan tanpa galat sama sekali sampai melampaui 2^{63}, sekitar 9.2×10^{18} rupiah.

Total *double* benar sampai sen yang dicetak, walaupun beberapa harga tunggal, seperti 4.1×7500, sedikit meleset. Dengan 16 digit bena, *double* mempunyai ruang lebih dari cukup untuk total 7.4×10^{8} rupiah; galat yang menumpuk jauh di bawah satu sen.

Total *float* kurang Rp826. Di sekitar 7×10^{8}, dua *float* bertetangga berjarak 64 (**Tabel 1.2**), sehingga setiap penambahan nota sekitar Rp30.000 dibulatkan ke kelipatan 64 dan kehilangan sampai 32 rupiah setiap kali. Selama 21.900 penjumlahan, kehilangan ini sebagian saling meniadakan dan sebagian menumpuk.

Total Kahan juga dihitung dalam *float*, tetapi hanya meleset Rp6, dan Rp6 adalah yang terbaik yang dapat dicapai *float*: 739.304.256 adalah *float* terdekat dengan total eksak 739.304.250. Trik ini, diterbitkan William Kahan pada 1965 [23], adalah mengingat apa yang hilang pada setiap penjumlahan lalu mengembalikannya pada penjumlahan berikutnya. Ketika jumlah berjalan t besar dan nilai baru y kecil, $t + y$ yang dihitung kehilangan digit-digit rendah y. Namun $(t + y) - t$ dapat dihitung *secara eksak* dalam *floating point* (dengan syarat ringan), dan mengurangkan y darinya memberikan bagian yang hilang beserta tandanya:

$$c = ((t + y) - t) - y \tag{1.16}$$

Peubah `carry` menyimpan c, dan nilai berikutnya yang akan ditambahkan dikoreksi menjadi $y - c$. Metode Kahan membatasi galat total sekitar $2u$ kali jumlah nilai mutlaknya, tidak bergantung pada banyaknya suku, alih-alih tumbuh bersamanya [22].

Pelajaran untuk urusan uang sederhana, dan itulah aturan yang diikuti bank dan sistem pembayaran: **hitung uang dalam bilangan bulat satuan terkecilnya** (rupiah bulat, atau sen bila diperlukan), dan pakai *floating point* hanya untuk besaran hasil ukur, seperti berat dan tingkat bunga. Bila *floating point* harus dijumlahkan dalam jumlah besar, pakai presisi ganda dan penjumlahan terkompensasi atau berpasangan.

1.14 Membandingkan dengan Pustaka

Program kita menemukan epsilon mesin lewat percobaan dan menjumlahkan dengan kode kita sendiri. Pustaka menyediakan pengetahuan yang sama dalam bentuk siap pakai, dan ada gunanya bertanya kepadanya. **Kode 1.13** memakai modul `math` Python dan NumPy; **Kode 1.14** memakai `Math` dan `BigDecimal` Java.

Kode 1.13 `compare_library.py`: epsilon mesin, *double* berikutnya, float32, dan tiga cara menjumlahkan, dengan NumPy 2.x.

```python
"""What the libraries already know about floating point (needs NumPy)."""
from decimal import Decimal
import math

import numpy as np


def main():
    print(f"float64 eps (np.finfo)  : {np.finfo(np.float64).eps:.6e}")
    print(f"float32 eps (np.finfo)  : {np.finfo(np.float32).eps:.6e}")
    print(f"math.ulp(1.0)           : {math.ulp(1.0):.6e}")
    print(f"next double after 1.0   : {float(np.nextafter(1.0, 2.0))!r}")
    print(f"0.1 stored as float32   : {Decimal(float(np.float32(0.1)))}")
    tenths = [0.1] * 1000000
    loop = 0.0
    for v in tenths:
        loop += v
    print(f"loop sum of 10^6 x 0.1  : {loop!r}")
    print(f"np.sum (pairwise)       : {float(np.sum(np.array(tenths)))!r}")
    print(f"math.fsum (exact round) : {math.fsum(tenths)!r}")


if __name__ == "__main__":
    main()
```

Kode 1.14 `CompareLibrary.java`: fakta yang sama dari pustaka standar Java.

```java
import java.math.BigDecimal;
import java.util.Locale;

/** What the Java standard library already knows about floating point. */
public class CompareLibrary {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.printf("double eps (Math.ulp)   : %.6e%n", Math.ulp(1.0));
        System.out.printf("float eps (Math.ulp)    : %.6e%n", Math.ulp(1.0f));
        System.out.println("next double after 1.0   : " + Math.nextUp(1.0));
        System.out.println("0.1 stored as float     : " + new BigDecimal(0.1f));
        double loop = 0.0;
        BigDecimal exact = BigDecimal.ZERO;
        BigDecimal tenth = new BigDecimal("0.1");   // decimal, not binary
        for (int i = 0; i < 1000000; i++) {
            loop += 0.1;
            exact = exact.add(tenth);
        }
        System.out.println("loop sum of 10^6 x 0.1  : " + loop);
        System.out.println("BigDecimal sum          : " + exact);
    }
}
```

```
float64 eps (np.finfo)  : 2.220446e-16
float32 eps (np.finfo)  : 1.192093e-07
math.ulp(1.0)           : 2.220446e-16
next double after 1.0   : 1.0000000000000002
0.1 stored as float32   : 0.100000001490116119384765625
loop sum of 10^6 x 0.1  : 100000.00000133288
np.sum (pairwise)       : 100000.00000000003
math.fsum (exact round) : 100000.0
double eps (Math.ulp)   : 2.220446e-16
float eps (Math.ulp)    : 1.192093e-07
next double after 1.0   : 1.0000000000000002
0.1 stored as float     : 0.100000001490116119384765625
loop sum of 10^6 x 0.1  : 100000.00000133288
BigDecimal sum          : 100000.0
```

Pustaka membenarkan percobaan kita: `np.finfo` dan `math.ulp` melaporkan epsilon yang sama dengan yang kita temukan lewat pembagian dua, dan presisi tunggal menyimpan 0,1 sebagai `0.100000001490116119384765625`, benar hanya sampai sekitar 8 digit. Tiga jumlah sejuta sepersepuluh itu patut dicermati. Perulangan sederhana meleset 1.3×10^{-6}. `np.sum` di NumPy memakai **penjumlahan berpasangan** (*pairwise summation*), yang menjumlahkan tetangga-tetangga dalam bentuk pohon sehingga galatnya tumbuh seperti $\log_2 n$, bukan n [24]; ia hanya meleset 3×10^{-11}. `math.fsum` melacak bagian-bagian yang hilang secara eksak dan mengembalikan jumlah nilai-nilai tersimpan yang dibulatkan dengan benar, `100000.0`. Di Java, `BigDecimal` dengan untaian desimal `"0.1"` menghindari pecahan biner sama sekali dan eksak, dengan harga jauh lebih lambat; itulah alat yang tepat untuk uang bila satuan bulat tidak cukup. Dokumentasi Python mempunyai bab tutorial yang baik tentang hal-hal ini [25].

1.15 Metode Numerik dalam Praktik: Profitina

Profitina adalah platform visibilitas AI, berjalan di app.profitina.com, yang mengukur seberapa siap situs web sebuah merek untuk dikutip mesin jawaban AI dan membantu memperbaiki apa yang ditemukannya. Beberapa fiturnya bertumpu tepat pada gagasan-gagasan bab ini.

Uang dalam satuan bulat. Langganan Pro berharga Rp50.000 untuk tiga puluh hari, dibayar melalui QRIS atau dengan PROFIT, koin utilitas platform itu di *blockchain* Solana. Koin itu mempunyai 6 desimal, yang di *blockchain* berarti setiap saldo disimpan sebagai bilangan bulat satuan terkecilnya, sepersejuta koin. Itulah aturan Subbab 1.13 yang diterapkan dalam skala besar: nominal yang harus cocok secara tepat adalah bilangan bulat, dan saldo tidak pernah bergeser oleh pecahan yang tak dapat dijelaskan siapa pun. Konversi antara PROFIT dan rupiah melewati sebuah *oracle* harga, dan konversi adalah satu-satunya tempat sebuah kurs, bilangan real, bertemu dengan sebuah nominal; hasilnya harus dibulatkan ke satuan bulat sekali saja, pada titik yang jelas, dengan aturan yang dinyatakan.

Skor yang dijumlahkan. AI Visibility Engine memberi skor pada sebuah situs dari banyak sinyal dan menjelaskan masing-masing. Menggabungkan banyak nilai ukur menjadi satu bilangan adalah jumlah hasil kali, dan Subbab 1.13 menunjukkan bahwa urutan dan cara penjumlahan penting bila sukunya banyak. Menormalkan skor ke skala 0 sampai 100 berarti membagi dengan sebuah rentang; jika rentangnya sangat kecil, pembagian itu memperbesar galat apa pun yang dibawa masukannya, efek yang sama dengan $1/h$ pada Persamaan (1.14).

Tren dari waktu ke waktu. Dasbor historis menampilkan tren skor mingguan per merek dan mengekspornya sebagai CSV. Selisih dua skor yang hampir sama, misalnya “naik 0,3 poin minggu ini”, adalah pengurangan dua bilangan yang hampir sama, dan baru bermakna bila skor-skor itu sendiri diketahui sampai lebih banyak digit daripada yang ditampilkan selisihnya: pembatalan katastrofik dalam bahasa bisnis. Pelajaran umumnya sama dengan di *laundry* Bu Wati: bilangan yang dilaporkan harus disimpan dan dihitung dengan presisi yang cukup agar digit terakhir yang ditampilkan dapat dipercaya.

1.16 Memakainya Sungguhan

Semua yang dibahas bab ini berlaku langsung pada *spreadsheet* dan program kecil yang dipakai toko, kebun, dan klinik setiap hari.

- **Kolom uang.** Di *spreadsheet*, simpan harga dan total dalam rupiah bulat dan bulatkan setiap nota sekali, saat nota dibuat, dengan fungsi `ROUND`; jangan pernah membandingkan dua total

hasil hitungan dengan "sama dengan", melainkan periksa bahwa selisihnya di bawah setengah rupiah. Di program, simpan uang dalam `long` (Java) atau `int` (Python), atau dalam `BigDecimal` dan `Decimal` bila sen diperlukan.

- **Nilai hasil ukur.** Berat, panjang, suhu, dan tingkat bunga memang diukur dengan sedikit digit; *double* jauh lebih dari cukup untuknya, dan *float* biasanya juga cukup, asalkan Anda tidak menjumlahkan ribuan nilai menjadi total yang besar.
- **Jumlah yang panjang.** Catatan harian bacaan sensor, transaksi setahun, simulasi panjang: pakai presisi ganda, dan untuk jumlah yang sangat panjang pakai `math.fsum` atau `sum` dari NumPy di Python, atau penjumlahan Kahan di Java.
- **Selisih bilangan yang hampir sama.** Sebelum mengurangkan dua bilangan besar yang hampir sama, tanyakan apakah rumusnya dapat disusun ulang, seperti rumus kuadrat tadi. Sering kali bisa.

1.17 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Membandingkan *double* dengan `==`.** `0.1 + 0.2 == 0.3` bernilai salah. Bandingkan dengan toleransi, relatif untuk bilangan besar, mutlak di dekat nol.
- **Menyimpan uang dalam *floating point*.** Rp826 lenyap dari total *float* setahun. Hitung uang dalam bilangan bulat satuan terkecilnya.
- **Memercayai semua digit yang tercetak.** Sebuah *double* dicetak dengan 17 digit bena, tetapi setelah pembatalan mungkin hanya sedikit yang benar. Banyaknya digit yang dapat dipercaya ditentukan oleh analisis galat, bukan oleh cetakan.
- **Membuat langkah "sekecil mungkin".** Selisih maju dengan $h = 10^{-16}$ memberi 0, bukan 0,35. Langkah terbaik menyeimbangkan pemotongan dan pembulatan (Persamaan (1.15)).
- **Menjumlahkan deret berselang-seling dengan suku raksasa.** e^{-20} dari deretnya keliru tiga kali lipat. Susun ulang (di sini, $1/e^{20}$) agar suku-suku besar yang berlawanan tanda tidak saling meniadakan.
- **Berhenti berdasarkan galat sejati.** Program tidak mungkin mengetahui galat sejati. Pakai galat hampiran Persamaan (1.3) dengan toleransi dari Persamaan (1.4), dan ingat bahwa metode yang konvergen lambat bisa berhenti terlalu dini.
- **Mencetak dengan *locale* yang salah di Java.** Pada komputer berbahasa Indonesia, `printf("%.2f", 1.5)` mencetak `1,50`. Pasang `Locale.setDefault(Locale.ROOT)` bila keluarannya berupa data, bukan teks untuk dibaca orang.
- **Lupa mengaktifkan lingkungan virtual.** `ModuleNotFoundError: No module named 'numpy'` biasanya berarti terminal menjalankan Python sistem, bukan `.venv`. Aktifkan lingkungannya, atau pilih interpreter `.venv` di VS Code.

1.18 Ringkasan Bab

- Metode numerik mengganti sebuah masalah dengan barisan operasi aritmetika yang berhingga, yang jawabannya dekat dengan jawaban sejati sejauh yang dapat kita taksir. Metode dinilai dari konvergensi, kecepatan konvergensi, dan kestabilan.
- Meja kerjanya adalah JDK 25 LTS, Python 3.14 dengan lingkungan virtual berisi NumPy dan SciPy, serta VS Code dengan ekstensi Java dan Python; `check_setup.py` dan `CheckSetup.java` memastikannya.

- Galat diukur sebagai galat sejati Persamaan (1.1), galat relatif Persamaan (1.2), dan, bila nilai sejati tidak diketahui, galat relatif hampiran Persamaan (1.3); iterasi berhenti ketika galat itu turun di bawah $\varepsilon_s = 0.5 \times 10^{2-n}\%$ untuk n digit (Persamaan (1.4)).
- Sebuah *double* disimpan sebagai tanda, eksponen 11 bit, dan pecahan 52 bit (Persamaan (1.6)); kebanyakan pecahan desimal, seperti 0,1 (Persamaan (1.5)), tersimpan sedikit keliru. Setiap bilangan yang disimpan dan setiap operasi mempunyai galat relatif paling besar $u = 2^{-53}$ (Persamaan (1.8)); epsilon mesin adalah 2^{-52} (Persamaan (1.7)).
- Galat pembulatan menumpuk sepanjang banyak operasi, dan pengurangan bilangan yang hampir sama menyingkapnya (pembatalan katastrofik); menyusun ulang rumus, seperti pada Persamaan (1.10), menyembuhkannya.
- Galat pemotongan berasal dari memotong proses tak hingga, seperti pada deret Taylor Persamaan (1.11) dan selisih maju Persamaan (1.13). Memperkecil langkah menurunkan galat pemotongan tetapi menaikkan galat pembulatan; langkah terbaik menyeimbangkan keduanya (Persamaan (1.14), Persamaan (1.15)).
- Uang dihitung dalam bilangan bulat; jumlah panjang dari nilai hasil ukur memakai presisi ganda dan penjumlahan terkompensasi (Persamaan (1.16)) atau berpasangan. Program kita sepakat dengan jawaban pustaka NumPy dan Java.

1.19 Soal Latihan

1. Hitung dengan tangan galat sejati, galat relatif, dan banyaknya angka bena yang benar dari hampiran 3.14 dan 3.1416 untuk π.
2. Tuliskan 0,2 dan 0,375 dalam biner. Manakah yang tersimpan tepat sebagai *double*, dan mengapa?
3. Uraikan *double* yang ke-64 bitnya, ditulis dalam heksadesimal, adalah `3FF8000000000000`. (Digit heksadesimal 3 adalah 0011, F adalah 1111, 8 adalah 1000.)
4. Berapa suku deret Persamaan (1.12) yang diperlukan aturan berhenti untuk memberikan e^1 sampai 6 angka bena? Kerjakan suku-suku pertamanya dengan tangan, lalu ubah `exp_series.py` atau `ExpSeries.java` untuk memeriksanya.
5. Rumus $\sqrt{x+1} - \sqrt{x}$ kehilangan ketelitian untuk x besar. Jelaskan sebabnya, tulis ulang tanpa pengurangan, lalu bandingkan kedua bentuk di $x = 10^{12}$ dengan beberapa baris Python atau Java.
6. Jumlahkan $1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{10^6}$ dalam presisi tunggal sekali maju (dari 1 ke atas) dan sekali mundur (dari $\frac{1}{10^6}$ ke bawah), lalu bandingkan keduanya dengan jumlah presisi ganda. Urutan mana yang lebih baik, dan mengapa?
7. Dengan Persamaan (1.15), ramalkan langkah terbaik untuk selisih maju $f(x) = e^x$ di $x = 1$ dan galat terkecil yang dapat diharapkan.
8. Bu Wati menaikkan harga menjadi Rp8.000 per kilogram, dan timbangannya kini menampilkan berat sampai 10 g. Apakah total bilangan bulat `laundry_ledger.py` masih eksak? Apa yang berubah jika harganya menjadi Rp7.750?

1.20 Penyelesaian Soal Latihan

1. Dengan $\pi = 3.14159265$: untuk 3.14, $E_t = 0.00159265$ dan $\varepsilon_t = 0.00159265/3.14159265 = 5.07 \times 10^{-4} = 0.0507\%$. Nilai itu di bawah 5×10^{-3} tetapi tidak di bawah 5×10^{-4}, jadi 3.14

mempunyai 3 angka bena yang benar. Untuk 3.1416, $E_t = -0.00000735$ dan $\varepsilon_t = -2.34 \times 10^{-6}$, di bawah 5×10^{-6}: 5 angka bena.

2. Menggandakan 0,2: 0,4 (0), 0,8 (0), 1,6 (1, sisakan 0,6), 1,2 (1, sisakan 0,2), dan 0,2 kembali, sehingga $0.2 = 0.0011\,0011\,\ldots_2$, panjangnya tak berhingga dan tidak tersimpan tepat. Menggandakan 0,375: 0,75 (0), 1,5 (1, sisakan 0,5), 1,0 (1, sisakan 0), sehingga $0.375 = 0.011_2$, eksak. Sebuah pecahan eksak dalam biner tepat ketika penyebutnya dalam bentuk paling sederhana adalah pangkat dua: $0.375 = 3/8$, sedangkan $0.2 = 1/5$.

3. Bitnya 0011 1111 1111 1000 0000 … Tandanya 0. Eksponennya 11 bit berikutnya, $01111111111_2 = 1023$, sehingga pangkatnya 2^0. Pecahannya dimulai dengan 1000 …, sehingga signifikannya $1.1_2 = 1.5$. Bilangannya adalah 1.5.

4. Dengan $n = 6$, $\varepsilon_s = 0.5 \times 10^{-4}\% = 0.00005\%$. Jumlah parsial e^1 adalah 1; 2; 2,5; 2,666667; 2,708333; 2,716667; 2,718056; 2,718254; 2,718279; 2,718282. Suku tambahan dibagi jumlah baru adalah $(1/9!)/2.718282 = 1.01 \times 10^{-6}$, yaitu 0.000101%, setelah 10 suku, masih di atas toleransi, dan $(1/10!)/2.718282 = 1.01 \times 10^{-7}$, yaitu 0.0000101%, setelah 11 suku, yang pertama di bawah 0.00005%. Jadi diperlukan 11 suku. Memanggil `exp_series(1.0, 6, show=True)` mencetak tabelnya dan berhenti pada 11 suku, dengan galat relatif sejati sekitar 0.000001%.

5. Untuk x besar kedua akar kuadrat hampir sama, sehingga pengurangan meniadakan digit-digit depannya dan menyisakan galat pembulatan. Kalikan dan bagi dengan jumlahnya: $\sqrt{x+1} - \sqrt{x} = \frac{(x+1)-x}{\sqrt{x+1}+\sqrt{x}} = \frac{1}{\sqrt{x+1}+\sqrt{x}}$. Di $x = 10^{12}$ bentuk langsung memberikan `5.000038072466850e-07` dan bentuk tulis ulang `4.999999999998749e-07`; nilai sejatinya $4.99999999999875 \times 10^{-7}$, jadi bentuk langsung hanya benar 5 digit dan bentuk tulis ulang 15. Dalam Python: `x = 1e12; print(math.sqrt(x + 1) - math.sqrt(x), 1 / (math.sqrt(x + 1) + math.sqrt(x)))`.

6. **Kode 1.15** dan **Kode 1.16** mengerjakannya.

Kode 1.15 `harmonic32.py`: jumlah harmonik dalam presisi tunggal, maju dan mundur.

```python
"""Exercise 1.6: the harmonic sum in single precision, forward and backward."""
import struct

N = 1000000


def f32(x):
    """Round to the nearest single-precision number."""
    return struct.unpack("f", struct.pack("f", x))[0]


def main():
    forward = 0.0
    for k in range(1, N + 1):
        forward = f32(forward + f32(1.0 / k))
    backward = 0.0
    for k in range(N, 0, -1):
        backward = f32(backward + f32(1.0 / k))
    double = 0.0
    for k in range(N, 0, -1):
        double += 1.0 / k
    print(f"float, 1 up to N   : {forward:.7f}")
    print(f"float, N down to 1 : {backward:.7f}")
    print(f"double, N down to 1: {double:.7f}")


if __name__ == "__main__":
    main()
```

Kode 1.16 `Harmonic32.java`: hal yang sama dalam Java.

```java
import java.util.Locale;

/** Exercise 1.6: the harmonic sum in single precision, forward and backward. */
public class Harmonic32 {
    static final int N = 1000000;

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        float forward = 0.0f;
        for (int k = 1; k <= N; k++) {
            forward = forward + (float) (1.0 / k);
        }
        float backward = 0.0f;
        for (int k = N; k >= 1; k--) {
            backward = backward + (float) (1.0 / k);
        }
        double dbl = 0.0;
        for (int k = N; k >= 1; k--) {
            dbl += 1.0 / k;
        }
        System.out.printf("float, 1 up to N   : %.7f%n", forward);
        System.out.printf("float, N down to 1 : %.7f%n", backward);
        System.out.printf("double, N down to 1: %.7f%n", dbl);
    }
}
```

```
float, 1 up to N   : 14.3573580
float, N down to 1 : 14.3926516
double, N down to 1: 14.3927267
```

Jumlah mundur benar sampai 5 digit (14,39265 terhadap 14,39273); jumlah maju sudah keliru pada digit ketiga. Ketika maju, jumlah berjalan segera mencapai sekitar 10, tempat jarak antar-*float* sekitar 10^{-6}, dan suku-suku kecil $1/k$ untuk k besar hampir seluruhnya terbulatkan hilang. Ketika mundur, suku-suku kecil dijumlahkan satu sama lain lebih dulu, selagi jumlah berjalan

masih kecil, dan suku-suku besar datang terakhir. Menjumlahkan dari yang terkecil ke yang terbesar menjaga jumlah berjalan sekecil mungkin selama mungkin.

7. Untuk e^x di 1, $f = f'' = e = 2.718$, sehingga $h_{\text{opt}} = 2\sqrt{\varepsilon_{\text{mach}}\, e/e} = 2\sqrt{2.22 \times 10^{-16}} = 3.0 \times 10^{-8}$, dan galat terkecilnya sekitar $Mh_{\text{opt}} = 2.718 \times 3.0 \times 10^{-8} \approx 8 \times 10^{-8}$, galat relatif sekitar 3×10^{-8}: hanya separuh dari 16 digit mesin yang selamat melewati selisih maju.

8. Pada Rp8.000 per kilogram, grams $\times$ 8000/1000 $=$ 8 $\times$ grams, bilangan bulat untuk setiap bilangan bulat gram: tetap eksak. Pada Rp7.750 dan berat dengan langkah 10 g, grams $\times$ 7750/1000 $=$ 7.75 $\times$ grams, yang untuk 1.010 g memberikan Rp7.827,5. Harganya tidak lagi berupa rupiah bulat, sehingga toko harus memutuskan cara membulatkan setiap nota (misalnya ke atas sampai Rp100 berikutnya, seperti banyak toko), dan program harus membulatkan setiap nota sekali, dengan aritmetika bilangan bulat seperti `(grams * 7750 + 999) // 1000` untuk pembulatan ke atas ke rupiah bulat, sebelum menjumlahkan. Pembagian bulat `//` di Python dan `/` pada nilai `long` di Java sama-sama membuang pecahan, itulah sebabnya suku pembulatan ditambahkan lebih dulu.

Daftar Pustaka

[1] U.S. General Accounting Office, "Patriot Missile Defense: Software Problem Led to System Failure at Dhahran, Saudi Arabia," GAO/IMTEC-92-26, Feb. 1992. [Online]. Available: https://www.gao.gov/products/imtec-92-26

[2] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[3] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[4] Oracle, "Java Downloads: JDK 25 (LTS) and JDK 27," accessed Sep. 2026. [Online]. Available: https://www.oracle.com/java/technologies/downloads/

[5] OpenJDK, "JDK 25," General Availability Sep. 16, 2025. [Online]. Available: https://openjdk.org/projects/jdk/25/

[6] Python Software Foundation, "Python 3.14.7," Aug. 5, 2026. [Online]. Available: https://www.python.org/downloads/release/python-3147/

[7] NumPy Developers, "numpy 2.5.3," Python Package Index, Sep. 6, 2026. [Online]. Available: https://pypi.org/project/numpy/

[8] SciPy Developers, "scipy 1.18.1," Python Package Index, Aug. 21, 2026. [Online]. Available: https://pypi.org/project/scipy/

[9] Microsoft, "Visual Studio Code 1.139," release notes, Sep. 23, 2026. [Online]. Available: https://code.visualstudio.com/updates

[10] Microsoft, "Java in Visual Studio Code," *VS Code Documentation*. [Online]. Available: https://code.visualstudio.com/docs/languages/java

[11] Microsoft, "Python in Visual Studio Code," *VS Code Documentation*. [Online]. Available: https://code.visualstudio.com/docs/languages/python

[12] Inside.java, "The Arrival of Java 27!," Sep. 15, 2026. [Online]. Available: https://inside.java/2026/09/15/jdk-27-available/

[13] Oracle, "JDK 19 Release Notes: Double.toString(double) and Float.toString(float) May Return Slightly Different Results," 2022. [Online]. Available: https://www.oracle.com/java/technologies/javase/19-relnote-issues.html

[14] Adoptium, "Eclipse Temurin 8u504, 11.0.32.1, 17.0.20.1, 21.0.12.1, 25.0.4.1 and 26.0.2.1 Available," Sep. 2, 2026. [Online]. Available: https://adoptium.net/news/2026/09/eclipse-temurin-8u504-110321-170201-210121-25041-26021-available

[15] Python Software Foundation, "What's New in Python 3.14," *Python 3.14 Documentation*. [Online]. Available: https://docs.python.org/3.14/whatsnew/3.14.html

[16] OpenJDK, "JEP 330: Launch Single-File Source-Code Programs," JDK 11. [Online]. Available: https://openjdk.org/jeps/330

[17] Python Software Foundation, "Using Python on Windows," *Python 3.14 Documentation*. [Online]. Available: https://docs.python.org/3.14/using/windows.html

[18] Phoronix, "Ubuntu 26.04 LTS Moving To OpenJDK 25 By Default," 2026. [Online]. Available: https://www.phoronix.com/news/Ubuntu-26.04-OpenJDK-25

[19] Python Software Foundation, "venv — Creation of virtual environments," *The Python Standard Library*. [Online]. Available: https://docs.python.org/3/library/venv.html

[20] OpenJDK, "JEP 512: Compact Source Files and Instance Main Methods," JDK 25. [Online]. Available: https://openjdk.org/jeps/512

[21] *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019, Jul. 2019.

[22] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[23] W. Kahan, "Further remarks on reducing truncation errors," *Communications of the ACM*, vol. 8, no. 1, p. 40, 1965.

[24] NumPy Developers, "numpy.sum," *NumPy Reference*. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.sum.html

[25] Python Software Foundation, "Floating-Point Arithmetic: Issues and Limitations," *The Python Tutorial*. [Online]. Available: https://docs.python.org/3/tutorial/floatingpoint.html

Bab 2

Mencari Akar I: Biseksi dan Regula Falsi

Banyak pertanyaan sehari-hari berujung pada persamaan yang tidak dapat diselesaikan dengan rumus apa pun: berapa suku bunga yang tersembunyi dalam rencana cicilan, seberapa dalam air di tangki bundar yang berisi 3.000 liter, kapan harga yang terus turun mencapai tingkat tertentu. Pertanyaan semacam itu menanyakan sebuah *akar*, yaitu bilangan yang membuat suatu fungsi bernilai nol. Bab ini membangun dua cara tertua dan teraman untuk menemukannya. Keduanya berangkat dari sebuah selang tempat fungsi berganti tanda, lalu menyempitkannya sampai akar terkunci. Di akhir bab Anda akan mampu mengubah pertanyaan praktis menjadi masalah pencarian akar; menjelaskan dan membuktikan mengapa pergantian tanda menjamin adanya akar; menjalankan biseksi dan regula falsi dengan tangan; meramalkan dengan tepat berapa langkah biseksi yang diperlukan untuk ketelitian tertentu; mengenali saat regula falsi merangkak dan memperbaikinya dengan modifikasi Illinois; mengimplementasikan ketiga metode dalam Java dan Python dengan tabel iterasi yang sama sampai digit terakhir; serta memeriksa hasilnya terhadap pencari akar profesional SciPy.

2.1 Mas Dimas dan Ponsel “Bunga 1,67% Sebulan”

Mas Dimas adalah pengemudi ojek daring di Yogyakarta. Ponselnya adalah kantornya: pesanan masuk lewat ponsel, peta berjalan di ponsel, dan pembayaran lewat ponsel. Ketika layar ponsel lamanya retak untuk ketiga kalinya, ia pergi ke gerai ponsel di sebuah pusat perbelanjaan untuk membeli ponsel baru secara kredit. Harganya Rp6.000.000. Pramuniaga menawarkan dua belas angsuran bulanan sebesar Rp600.000, “dengan bunga hanya 1,67% sebulan”.

Dimas berhitung di balik brosur. Dua belas kali Rp600.000 adalah Rp7.200.000, lebih Rp1.200.000 daripada harga. Rp1.200.000 dibagi 12 bulan adalah Rp100.000 per bulan, dan Rp100.000 adalah 1,67% dari Rp6.000.000. Jadi pramuniaga itu tidak berbohong. Tetapi ada yang mengganjal. Pada bulan-bulan terakhir, utangnya kepada toko tinggal sedikit, namun ia tetap membayar “bunga” Rp100.000 yang sama setiap bulan. Bukankah bunga atas utang yang kecil semestinya kecil?

Kecurigaannya beralasan. Angka “1,67%” adalah **bunga flat**: dihitung dari harga awal sepanjang masa cicilan, padahal utangnya menyusut setiap kali ia mengangsur. Suku bunga yang akan dihitung bank, yaitu laju pertumbuhan utang yang menyusut itu setiap bulan, disebut **bunga efektif**, dan nilainya jauh lebih tinggi. Aturan perlindungan konsumen di Indonesia mewajibkan penyedia jasa keuangan menjelaskan biaya produknya dengan jelas kepada konsumen [1], dan bunga efektiflah yang memungkinkan dua penawaran dibandingkan secara adil. Namun brosur itu tidak mencantumkannya, dan Dimas tidak mempunyai rumus untuk menghitungnya, karena rumus itu memang tidak ada. Suku bunga itu adalah akar sebuah persamaan.

Bab ini menemukan akar tersebut. Ini aritmetika, bukan nasihat keuangan: program memberi tahu Dimas berapa suku bunga yang tersirat dalam sebuah rencana cicilan, agar ia dapat membandingkannya dengan pinjaman bank, koperasi, atau menabung tiga bulan lebih dulu;

keputusannya tetap miliknya. Program yang sama berlaku untuk sepeda motor, lemari es, atau laptop, dan metode yang sama menemukan akar di setiap bab berikutnya.

2.2 Gagasannya dengan Kata-Kata Sederhana

2.2.1 Uang hari ini dan uang nanti

Satu rupiah yang dibayar sebulan lagi bernilai lebih kecil bagi toko daripada satu rupiah yang dibayar hari ini, karena rupiah hari ini dapat dipinjamkan dan menghasilkan bunga selama sebulan. Jika suku bunga bulanannya r (misalnya $r = 0.02$ untuk 2%), maka Rp1 hari ini tumbuh menjadi Rp$(1 + r)$ setelah sebulan dan menjadi Rp$(1 + r)^k$ setelah k bulan. Dibalik, angsuran A yang dibayar setelah k bulan bernilai $A/(1 + r)^k$ hari ini; inilah **nilai kini**-nya (*present value*).

Rencana cicilan yang adil adalah rencana yang nilai kini semua angsurannya berjumlah sama dengan harga P:

$$P = \frac{A}{1+r} + \frac{A}{(1+r)^2} + \cdots + \frac{A}{(1+r)^n} \tag{2.1}$$

Ruas kanan Persamaan (2.1) adalah deret geometri dengan suku pertama $a = A/(1 + r)$ dan rasio $q = 1/(1 + r)$. Rumus biasa untuk jumlah seperti itu adalah $a(1 - q^n)/(1 - q)$; di sini $1 - q = r/(1 + r)$, sehingga $a/(1 - q) = \frac{A}{1+r} \cdot \frac{1+r}{r} = A/r$, dan $q^n = (1 + r)^{-n}$. Karena itu jumlahnya menyederhana menjadi

$$P = A\,\frac{1 - (1+r)^{-n}}{r} \tag{2.2}$$

Inilah **rumus anuitas**, dan bank memakainya ke dua arah. Bila suku bunga r diketahui, rumus itu memberikan angsuran A; misalnya, pada $r = 2\%$ harga Rp6.000.000 selama 12 bulan memerlukan $A = 6{,}000{,}000/10.5753 =$ Rp567,358. Tetapi Dimas mengetahui P, A, dan n, dan ingin mengetahui r, sedangkan r muncul sekaligus di dalam pangkat dan di penyebut. Tidak ada rangkaian langkah aljabar yang dapat memisahkannya.

2.2.2 Mengubah pertanyaan menjadi akar

Setiap pertanyaan seperti itu dapat dibawa ke bentuk yang sama: pindahkan semuanya ke satu ruas, lalu tanyakan di mana hasilnya nol. Definisikan

$$f(r) = A\,\frac{1 - (1+r)^{-n}}{r} - P \tag{2.3}$$

Untuk Dimas, $P = 6{,}000{,}000$, $A = 600{,}000$, dan $n = 12$. Suku bunga yang dicari adalah bilangan r^* dengan $f(r^*) = 0$, sebuah **akar** (atau *titik nol*) dari f. Dengan kata-kata, $f(r)$ adalah "nilai kini angsuran-angsuran pada suku bunga r, dikurangi harga": positif bila r terlalu rendah (angsuran bernilai lebih daripada ponselnya), negatif bila r terlalu tinggi.

2.2.3 Tabel tebakan

Sebelum memakai metode apa pun, mari kita coba beberapa suku bunga, seperti yang mungkin dilakukan Dimas dengan kalkulator. Pecahan dalam Persamaan (2.2) disebut faktor anuitas; bila dikalikan A lalu dikurangi P, hasilnya f.

Tabel 2.1 Fungsi pinjaman Persamaan (2.3) pada lima suku bunga bulanan: tandanya berganti antara 2% dan 3%.

Suku bunga r	Faktor anuitas	$f(r)$ (rupiah)
1%	11.2551	+753,046
2%	10.5753	+345,205
3%	9.9540	−27,598
4%	9.3851	−368,956
5%	8.8633	−682,049

Tabel 2.1 sudah memberi tahu banyak hal. Pada 2%, angsuran bernilai Rp345.205 lebih daripada ponselnya; pada 3%, bernilai Rp27.598 kurang; jadi suku bunga sebenarnya terletak antara 2% dan 3%, dan karena $-27{,}598$ jauh lebih dekat ke nol daripada $+345{,}205$, kemungkinan besar ia dekat dengan 3%. Pergantian tanda adalah kuncinya. Dua metode bab ini tidak lain adalah cara yang tertib untuk meneruskan tabel ini: biseksi selalu mencoba titik tengah selang tempat tanda berganti, sedangkan regula falsi memakai *besar* kedua nilai, seperti yang baru saja kita lakukan dengan mata, untuk menebak letak titik nolnya.

2.3 Biseksi

2.3.1 Mengapa pergantian tanda menjamin adanya akar

Seluruh metode bertumpu pada satu teorema tentang fungsi kontinu, yaitu fungsi yang grafiknya dapat digambar tanpa mengangkat pensil.

> **DEFINISI.** *Teorema Nilai Antara.* Jika f kontinu pada selang tertutup $[a, b]$ dan K adalah bilangan apa pun di antara $f(a)$ dan $f(b)$, maka ada sedikitnya satu c di $[a, b]$ dengan $f(c) = K$. Khususnya, jika $f(a)$ dan $f(b)$ berlawanan tanda, yaitu $f(a)\,f(b) < 0$, maka f mempunyai sedikitnya satu akar di (a, b) [2].

Gambaran di baliknya sederhana: lintasan kontinu yang berawal di bawah sumbu dan berakhir di atasnya pasti memotong sumbu di suatu tempat. Fungsi pinjaman kita Persamaan (2.3) kontinu untuk setiap $r > 0$ (fungsi itu dibangun dari penjumlahan, pangkat, dan pembagian dengan r, yang di sana tidak nol), dan $f(0.01) > 0 > f(0.05)$, sehingga teorema menjanjikan sebuah akar antara 1% dan 5%. Selang $[a, b]$ dengan $f(a)f(b) < 0$ disebut **kurung** (*bracket*), dan metode yang mempertahankan kurung di setiap langkah disebut *metode pengurung* (*bracketing methods*).

Dua peringatan perlu disampaikan di sini. Teorema itu menjanjikan *sedikitnya* satu akar; jika selang memuat tiga akar, biseksi menemukan salah satunya, dan yang mana bergantung pada selangnya. Selain itu kekontinuan penting: $f(x) = 1/x$ berganti tanda pada $[-1,1]$ tetapi tidak mempunyai akar di sana, dan biseksi akan dengan senang hati konvergen ke kutub di 0. Pada fungsi pinjaman, tidak ada masalah seperti itu, karena f menurun: suku bunga yang lebih tinggi selalu membuat angsuran bernilai lebih kecil hari ini, sehingga akarnya tepat satu.

2.3.2 Metodenya

Biseksi membagi dua kurungnya berulang-ulang. Ambil titik tengah c dari $[a, b]$. Jika $f(c)$ bertanda sama dengan $f(a)$, pergantian tanda, dan karena itu akarnya, berada di $[c, b]$; jika tidak, di $[a, c]$. Bagaimanapun, kita memperoleh kurung baru selebar setengahnya. **Pseudokode 2.1** menuliskan prosedurnya.

Pseudokode 2.1 Metode biseksi dengan toleransi pada perubahan titik tengah.

```
BISECTION(f, a, b, tol, maxIter)
 1  fa = f(a);  fb = f(b)
 2  if fa * fb > 0: error "no sign change: [a, b] is not a bracket"
 3  c_old = a
 4  for k = 1 to maxIter
 5      c = a + (b - a) / 2              // the midpoint
 6      fc = f(c)
 7      if fc == 0 or |c - c_old| < tol: return c
 8      if fa * fc < 0: b = c; fb = fc   // root in [a, c]
 9      else:           a = c; fa = fc   // root in [c, b]
10      c_old = c
11  return c                             // tolerance not reached
```

Mari kita jalankan dengan tangan pada fungsi Dimas dengan kurung [0.01,0.05], memakai nilai-nilai **Tabel 2.1**.

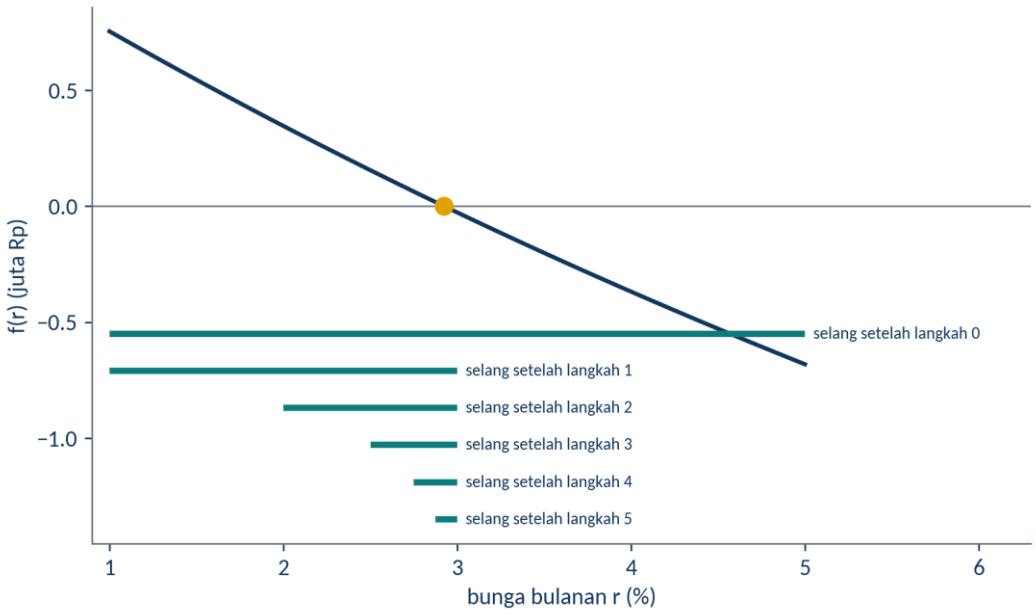
Langkah 1. $c = 0.01 + 0.04/2 = 0.03$ dan $f(0.03) = -27{,}598$. Karena $f(0.01) > 0$ dan $f(0.03) < 0$, tandanya berbeda: akarnya di [0.01,0.03], jadi b menjadi 0,03.

Langkah 2. $c = 0.01 + 0.02/2 = 0.02$ dan $f(0.02) = +345{,}205$, bertanda sama dengan $f(a)$: akarnya di [0.02,0.03], jadi a menjadi 0,02.

Langkah 3. $c = 0.025$, dan faktor anuitas pada 2,5% adalah 10.25776, sehingga $f(0.025) = 600{,}000 \times 10.25776 - 6{,}000{,}000 = +154{,}659$: positif, $a = 0.025$, kurung [0.025,0.03].

Langkah 4. $c = 0.0275$, $f = +62{,}522$: positif, kurung [0.0275,0.03].

Setelah empat langkah, suku bunga itu diketahui terletak antara 2,75% dan 3%, dan sudah jelas bahwa "1,67%" dari penjual bukanlah bunga efektif. **Gambar 2.1** menggambar fungsinya beserta kurung-kurung pertama.



Gambar 2.1 Fungsi pinjaman $f(r)$ dari Persamaan (2.3) dan kurung setelah setiap langkah biseksi, mulai dari $[1\%, 5\%]$: setiap langkah menyimpan separuh yang memuat pergantian tanda (titik jingga).

2.3.3 Berapa langkah: batas galat

Biseksi mempunyai sifat yang tidak dimiliki metode lain mana pun di buku ini: kita tahu *sebelumnya* berapa langkah yang diperlukannya. Tulis $[a_0, b_0]$ untuk kurung pertama dan c_k

untuk titik tengah ke-k. Kurung yang dibagi dua oleh c_k lebarnya $(b_0 - a_0)/2^{k-1}$, akarnya terletak di suatu tempat di dalamnya, dan c_k adalah titik tengahnya, sehingga

$$|c_k - r^*| \le \frac{b_0 - a_0}{2^k} \tag{2.4}$$

Setiap langkah membagi dua batas itu: biseksi memperoleh tepat satu digit biner jawaban per langkah, bagaimanapun bentuk fungsinya [2]. Agar galatnya pasti di bawah toleransi τ, cukup bila $(b_0 - a_0)/2^k \le \tau$, yang setelah mengambil logaritma memberikan

$$k \ge \log_2 \frac{b_0 - a_0}{\tau} \tag{2.5}$$

Untuk Dimas, $b_0 - a_0 = 0.04$ dan $\tau = 10^{-7}$ (suku bunga benar sampai 0.00001%, jauh melebihi keperluan), sehingga $k \ge \log_2 400{,}000 = 18.6$: 19 langkah. Karena $\log_2 10 \approx 3.32$, setiap tambahan satu digit *desimal* yang benar memerlukan sedikit lebih dari tiga langkah, dan 16 digit presisi ganda habis setelah sekitar 53 kali membagi dua selang sepanjang 1. Uji berhenti program, $|c_k - c_{k-1}| < \tau$, cocok persis dengan analisis ini, karena dua titik tengah berturutan selalu berjarak setengah kurung: $|c_k - c_{k-1}| = (b_0 - a_0)/2^k$, ruas kanan Persamaan (2.4).

Metode yang galatnya menyusut dengan faktor tetap $\lambda < 1$ di setiap langkah, $|e_{k+1}| \approx \lambda |e_k|$, dikatakan **konvergen linear** dengan laju λ. Batas galat biseksi menyusut dengan $\lambda = 1/2$. (Galat sebenarnya $|c_k - r^*|$ naik turun, karena sebuah titik tengah bisa kebetulan jatuh sangat dekat dengan akar dan titik berikutnya lebih jauh; hanya batasnya yang turun dengan mulus.) Bab 3 menjumpai metode yang jauh lebih cepat konvergen, tetapi tak satu pun disertai jaminan seperti Persamaan (2.5).

2.3.4 Mencari kurung sejak awal

Semua uraian tadi mengandaikan kurungnya sudah ada. Untuk pinjaman, akal sehat memberikannya: tidak ada rencana cicilan di toko yang menarik kurang dari 1% atau lebih dari 5% sebulan, dan **Tabel 2.1** memastikan tandanya. Bila akal sehat tidak berkata apa-apa, kurung dapat *dicari*. Pencarian paling sederhana berjalan dari titik awal x_0 dengan langkah Δ, menghitung $f(x_0), f(x_0 + \Delta), f(x_0 + 2\Delta), \ldots$, sampai dua nilai bertetangga berbeda tanda; kedua titik itu adalah kurung. Langkahnya harus cukup kecil agar tidak melompati dua akar sekaligus (dua pergantian tanda saling meniadakan dan pencarian tidak melihat satu pun), dan cukup besar agar tidak membuang ribuan evaluasi. Strategi kedua menggandakan langkah setiap kali, $x_0 + \Delta, x_0 + 2\Delta, x_0 + 4\Delta, \ldots$, yang menemukan pergantian tanda yang jauh dalam sedikit langkah; itulah pilihan wajar untuk besaran seperti suku bunga atau kedalaman yang diketahui positif tetapi tidak diketahui besarnya.

Grafik bahkan lebih baik. Sekali pandang pada grafik f di rentang yang lapang memperlihatkan di mana ia memotong sumbu, berapa kali, dan apakah ada kutub atau bagian yang datar. **Gambar 2.1** adalah grafik seperti itu untuk pinjaman. Menggambar dulu, baru mengurung, adalah kebiasaan yang patut dibentuk: kebiasaan itu menangkap sebagian besar kesalahan di Subbab 2.12 sebelum satu baris pun pencari akar dijalankan.

> **WAWASAN KUNCI:** Metode pengurung mengubah pertanyaan yang tidak punya rumus menjadi rangkaian pertanyaan ya-atau-tidak (“akarnya di kiri atau di kanan titik ini?”), yang masing-masing dijawab oleh tanda satu nilai fungsi. Itulah sebabnya metode ini begitu kokoh: tanda $f(c)$ hampir selalu terhitung benar, bahkan ketika nilainya sendiri kurang teliti.

2.3.5 Rincian kecil yang penting

Tiga rincian dalam **Pseudokode 2.1** tampak remeh, padahal tidak.

Titik tengah dihitung sebagai $a + (b - a)/2$, bukan $(a + b)/2$. Dalam aritmetika eksak keduanya sama. Dalam *floating point*, $(a + b)/2$ dapat jatuh *di luar* $[a, b]$ bila a dan b sangat berdekatan dan pembulatan $a + b$ salah arah, dan untuk a dan b yang sangat besar jumlahnya bisa meluap. Bentuk $a + (b - a)/2$ menambahkan koreksi kecil pada a dan tetap berada di dalam kurung [3].

Uji tanda memakai $f(a)$ **yang tersimpan**, diperbarui ketika a berpindah, sehingga setiap langkah hanya sekali mengevaluasi f. Untuk pinjaman, satu evaluasi itu murah; untuk fungsi yang menjalankan sebuah simulasi, banyaknya evaluasi adalah seluruh biaya metode. (Implementasi yang cermat membandingkan tanda alih-alih mengalikan, karena hasil kali dua nilai yang sangat kecil dapat *underflow* menjadi nol dan dua nilai yang sangat besar dapat meluap; untuk fungsi berukuran biasa, seperti di sini, hasil kali tidak bermasalah.)

Uji berhenti dilakukan pada perubahan c, dengan batas banyaknya iterasi sebagai jaring pengaman. Uji lain juga mungkin: $|f(c)|$ yang kecil, atau perubahan relatif $|c_k - c_{k-1}|/|c_k|$ yang kecil seperti galat relatif hampiran di Subbab 1.9.2. Uji pada $|f(c)|$ saja berbahaya, karena "kecil" bergantung pada skala f: fungsi pinjaman kita diukur dalam rupiah, dan $|f| < 1$ berarti satu rupiah, sedangkan uji yang sama pada fungsi yang diukur dalam jutaan rupiah berarti sesuatu yang sama sekali lain.

2.4 Regula Falsi

2.4.1 Talibusur

Biseksi mengabaikan informasi yang berguna. Pada **Tabel 2.1**, $f(0.02) = +345{,}205$ dan $f(0.03) = -27{,}598$; orang akan menebak bahwa akarnya jauh lebih dekat ke 3% daripada ke 2%, tetapi biseksi mencoba 2,5%. **Regula falsi**, "aturan posisi palsu", memakai besar kedua nilai itu. Ia menarik garis lurus, yaitu *talibusur* (*chord*), melalui $(a, f(a))$ dan $(b, f(b))$, lalu mengambil titik c tempat talibusur memotong sumbu sebagai tebakan berikutnya. Kemiringan talibusur adalah $(f(b) - f(a))/(b - a)$; garis melalui $(b, f(b))$ dengan kemiringan itu bernilai nol di

$$c = b - f(b)\,\frac{a - b}{f(a) - f(b)} \tag{2.6}$$

Metode ini sangat tua: juru tulis Mesir, Babilonia, dan Tiongkok memakai "posisi palsu" untuk menyelesaikan soal-soal linear, dan buku aritmetika Arab serta Eropa abad pertengahan mengajarkannya sebagai aturan baku [4]. Kurungnya lalu diperbarui persis seperti pada biseksi, dengan menyimpan bagian tempat tanda berganti.

Langkah pertamanya untuk Dimas, dengan $a = 0.01$, $f(a) = 753{,}046.5$, $b = 0.05$, $f(b) = -682{,}049.0$, adalah

$$c = 0.05 - (-682{,}049.0)\,\frac{0.01 - 0.05}{753{,}046.5 + 682{,}049.0} = 0.05 - \frac{27{,}282.0}{1{,}435{,}095.5} = 0.030989$$

sudah berjarak kurang dari 0,2 poin persentase dari akarnya, sedangkan titik tengah pertama biseksi, 0,03, berjarak 0,08 poin karena kebetulan, tetapi titik keduanya, 0,02, jatuh jauh ke belakang. **Pseudokode 2.2** memperlihatkan seluruh metode.

Pseudokode 2.2 Regula falsi, dengan modifikasi Illinois yang dinyalakan lewat sebuah penanda (baris 10 dan 14).

```
REGULA-FALSI(f, a, b, tol, maxIter, illinois)
 1  fa = f(a);  fb = f(b)
 2  if fa * fb > 0: error "no sign change"
 3  c_old = a;  side = 0
 4  for k = 1 to maxIter
 5      c = b - fb * (a - b) / (fa - fb)     // where the chord meets zero
 6      fc = f(c)
 7      if fc == 0 or |c - c_old| < tol: return c
 8      if fa * fc < 0:                      // root in [a, c]: b moves
 9          b = c; fb = fc
10          if illinois and side == -1: fa = fa / 2
11          side = -1
12      else:                                // root in [c, b]: a moves
13          a = c; fa = fc
14          if illinois and side == +1: fb = fb / 2
15          side = +1
16      c_old = c
17  return c
```

2.4.2 Mengapa regula falsi bisa merangkak

Regula falsi mempertahankan kurung, sehingga tidak pernah kehilangan akar, dan bila fungsinya hampir lurus ia konvergen dengan cepat. Namun ia mempunyai kelemahan yang tidak dimiliki biseksi. Jika f melengkung ke arah yang sama di seluruh kurung (cembung atau cekung), talibusur selalu berada di sisi yang sama terhadap grafik, titik baru c selalu jatuh di sisi yang sama terhadap akar, dan salah satu ujung kurung *tidak pernah bergerak*. Fungsi pinjaman kita cembung (grafiknya melengkung ke atas), f negatif di kanan akar, dan setiap talibusur memotong sumbu di kanan akar; jadi $a = 0.01$ tetap diam sepanjang perhitungan, dan hanya b yang merayap mendekati akar.

Dengan satu ujung terpaku di a, metode ini tetap konvergen, tetapi hanya linear, dan lajunya dapat kita hitung. Tulis ujung yang bergerak sebagai $b = r^* + e$ dengan galat kecil e, sehingga $f(b) \approx f'(r^*)\,e$ menurut rumus Taylor. Memasukkan ini ke Persamaan (2.6) dan hanya menyimpan suku-suku orde pertama dalam e memberikan $c - r^* \approx \lambda\, e$ dengan

$$\lambda = 1 - \frac{f'(r^*)\,(a - r^*)}{f(a)} \tag{2.7}$$

Pecahan itu membandingkan dua kemiringan: kemiringan $f'(r^*)$ garis singgung di akar dengan kemiringan $f(a)/(a - r^*)$ talibusur dari ujung yang terpaku ke akar. Bila keduanya hampir sama, fungsinya hampir lurus antara a dan akar, λ mendekati 0, dan metodenya cepat. Bila talibusur jauh lebih curam atau jauh lebih landai daripada garis singgung, λ mendekati 1 dan metodenya merangkak. Untuk fungsi Dimas, kemiringan garis singgung di akar sekitar -3.59×10^7 rupiah per satuan suku bunga dan kemiringan talibusur dari $a = 0.01$ adalah -3.92×10^7, sehingga $\lambda = 1 - 3.59/3.92 = 0.083$: setiap langkah semestinya memangkas galat sekitar 12 kali, dan Subbab 2.6 akan mengukur tepat itu.

Contoh klasiknya adalah $f(x) = x^{10} - 1$ pada [0,1.3] [5]: fungsinya hampir datar di dekat 0 dan sangat curam di dekat 1,3, sehingga setiap talibusur dari ujung kanan yang terpaku $b = 1.3$ hampir merupakan garis curam yang sama, dan setiap langkah hanya menggeser a sedikit mendekati akar 1. Dengan $f'(1) = 10$ dan $f(1.3) = 12.79$, Persamaan (2.7) (dengan b sebagai ujung yang terpaku) memberikan $\lambda = 1 - 10 \times 0.3/12.79 = 0.77$: setiap langkah menghapus kurang dari seperempat galat. Subbab 2.7 mengukurnya. Masalah kedua menyusul dari yang pertama: dengan satu ujung terpaku, lebar kurung tidak menyusut ke nol, sehingga uji pada lebar tidak pernah berhasil, dan uji pada perubahan $|c_k - c_{k-1}|$ bisa berhasil padahal galatnya masih lebih besar daripada toleransi, karena barisan yang merangkak lambat berubah sedikit dari langkah ke langkah.

2.4.3 Modifikasi Illinois

Obatnya sederhana dan elegan. Bila ujung yang sama dari kurung sudah diam dua kali berturut-turut, bagi dua nilai fungsi yang disimpan untuknya:

$$f(a) \leftarrow \frac{1}{2} f(a) \tag{2.8}$$

(atau hal yang sama untuk b). Nilai yang dibagi dua itu menarik talibusur berikutnya ke arah ujung yang macet, sehingga titik c berikutnya melompat ke sisi *lain* akar, dan ujung yang macet akhirnya bergerak. Inilah **metode Illinois**, yang sudah lama dikenal dengan nama laboratorium komputasi University of Illinois dan dianalisis Dowell dan Jarratt pada 1971 [6]. Mereka membuktikan bahwa orde konvergensinya sekitar 1,442: kasarnya, banyak digit yang benar dikalikan 1,44 di setiap langkah, jauh lebih baik daripada pertambahan tetap. Baris 10 dan 14 dari **Pseudokode 2.2** mengimplementasikannya dengan peubah `side` yang mengingat ujung mana yang terakhir bergerak.

2.5 Membangunnya dalam Java dan Python

Ketiga metode masuk ke satu modul, **Kode 2.1** dalam Python dan **Kode 2.2** dalam Java, agar program pinjaman dan setiap program selanjutnya dapat memakainya. Setiap fungsi menerima fungsi f sebagai parameter. Di Python, fungsi adalah nilai biasa yang dapat dioper ke mana-mana; di Java, parameternya bertipe `DoubleUnaryOperator`, antarmuka standar untuk "fungsi dari `double` ke `double`", dan pemanggil dapat mengoper ekspresi lambda seperti `x -> x * x - 2`.

Kode 2.1 `roots.py`: biseksi, regula falsi, dan varian Illinois, dengan tabel iterasi opsional.

```
"""Bracketing root finders: bisection, regula falsi and the Illinois variant."""

HEADER = f"{'k':>3} {'a':>11} {'b':>11} {'c':>11} {'f(c)':>15} {'|c - c_old|':>12}"


def _check(fa, fb):
    if fa * fb > 0:
        raise ValueError("f(a) and f(b) have the same sign: no bracket")


def bisection(f, a, b, tol=1e-7, max_iter=100, trace=False):
    """Halve [a, b] until two midpoints differ by less than tol.
    Returns (root, number of iterations)."""
    fa, fb = f(a), f(b)
    _check(fa, fb)
    c_old = a
    for k in range(1, max_iter + 1):
        c = a + (b - a) / 2                 # safer than (a + b) / 2
        fc = f(c)
        if trace:
            print(f"{k:>3} {a:>11.8f} {b:>11.8f} {c:>11.8f} {fc:>15.4f} "
                  f"{abs(c - c_old):>12.3e}")
        if fc == 0 or abs(c - c_old) < tol:
            return c, k
        if fa * fc < 0:                     # the root is in [a, c]
            b, fb = c, fc
        else:                               # the root is in [c, b]
            a, fa = c, fc
        c_old = c
    return c, max_iter


def regula_falsi(f, a, b, tol=1e-7, max_iter=100, trace=False,
                 illinois=False):
    """False position: cut [a, b] where the chord through (a, f(a)) and
    (b, f(b)) crosses zero. With illinois=True, an end point that stays
    twice in a row has its function value halved."""
    fa, fb = f(a), f(b)
    _check(fa, fb)
    c_old = a
    side = 0                                # which end stayed last time
    for k in range(1, max_iter + 1):
        c = b - fb * (a - b) / (fa - fb)    # where the chord meets zero
        fc = f(c)
        if trace:
            print(f"{k:>3} {a:>11.8f} {b:>11.8f} {c:>11.8f} {fc:>15.4f} "
```

Kode 2.1 (Lanjutan) *roots.py: biseksi, regula falsi, dan varian Illinois, dengan tabel iterasi opsional.*

```
                  f"{abs(c - c_old):>12.3e}")
        if fc == 0 or abs(c - c_old) < tol:
            return c, k
        if fa * fc < 0:                     # root in [a, c]: b moves
            b, fb = c, fc
            if illinois and side == -1:     # a stayed twice: halve f(a)
                fa /= 2
            side = -1
        else:                               # root in [c, b]: a moves
            a, fa = c, fc
            if illinois and side == 1:      # b stayed twice: halve f(b)
                fb /= 2
            side = 1
        c_old = c
    return c, max_iter
```

Kode 2.2 `Roots.java`: ketiga metode yang sama dalam Java, mengembalikan *record* `Result` berisi akar dan banyaknya iterasi.

```
import java.util.function.DoubleUnaryOperator;

/** Bracketing root finders: bisection, regula falsi and the Illinois variant. */
public class Roots {
    /** A root and the number of iterations that found it. */
    public record Result(double root, int iterations) { }

    public static final String HEADER = String.format(
            "%3s %11s %11s %11s %15s %12s", "k", "a", "b", "c", "f(c)",
            "|c - c_old|");

    static void check(double fa, double fb) {
        if (fa * fb > 0) {
            throw new IllegalArgumentException(
                    "f(a) and f(b) have the same sign: no bracket");
        }
    }

    static void row(int k, double a, double b, double c, double fc, double d) {
        System.out.printf("%3d %11.8f %11.8f %11.8f %15.4f %12.3e%n",
                          k, a, b, c, fc, d);
    }

    /** Halve [a, b] until two midpoints differ by less than tol. */
    public static Result bisection(DoubleUnaryOperator f, double a, double b,
                                   double tol, int maxIter, boolean trace) {
        double fa = f.applyAsDouble(a), fb = f.applyAsDouble(b);
        check(fa, fb);
        double c = a, cOld = a;
        for (int k = 1; k <= maxIter; k++) {
            c = a + (b - a) / 2;                 // safer than (a + b) / 2
            double fc = f.applyAsDouble(c);
            if (trace) {
                row(k, a, b, c, fc, Math.abs(c - cOld));
            }
            if (fc == 0 || Math.abs(c - cOld) < tol) {
                return new Result(c, k);
            }
            if (fa * fc < 0) {                   // the root is in [a, c]
                b = c;
                fb = fc;
            } else {                             // the root is in [c, b]
                a = c;
                fa = fc;
            }
            cOld = c;
```

Kode 2.2 (Lanjutan) *Roots.java: ketiga metode yang sama dalam Java, mengembalikan record Result berisi akar dan banyaknya iterasi.*

```
        }
        return new Result(c, maxIter);
    }

    /**
     * False position: cut [a, b] where the chord through (a, f(a)) and
     * (b, f(b)) crosses zero. With illinois = true, an end point that stays
     * twice in a row has its function value halved.
     */
    public static Result regulaFalsi(DoubleUnaryOperator f, double a, double b,
                                     double tol, int maxIter, boolean trace,
                                     boolean illinois) {
        double fa = f.applyAsDouble(a), fb = f.applyAsDouble(b);
        check(fa, fb);
        double c = a, cOld = a;
        int side = 0;                               // which end stayed last time
        for (int k = 1; k <= maxIter; k++) {
            c = b - fb * (a - b) / (fa - fb);       // where the chord meets zero
            double fc = f.applyAsDouble(c);
            if (trace) {
                row(k, a, b, c, fc, Math.abs(c - cOld));
            }
            if (fc == 0 || Math.abs(c - cOld) < tol) {
                return new Result(c, k);
            }
            if (fa * fc < 0) {                      // root in [a, c]: b moves
                b = c;
                fb = fc;
                if (illinois && side == -1) {       // a stayed twice: halve f(a)
                    fa /= 2;
                }
                side = -1;
            } else {                                // root in [c, b]: a moves
                a = c;
                fa = fc;
                if (illinois && side == 1) {        // b stayed twice: halve f(b)
                    fb /= 2;
                }
                side = 1;
            }
            cOld = c;
        }
        return new Result(c, maxIter);
    }
}
```

Kodenya mengikuti pseudokode baris demi baris. Bila `trace` menyala, setiap iterasi mencetak satu baris: nomor iterasi k, kurung $[a, b]$ sebelum langkah, titik baru c, $f(c)$, dan $|c - c_{\text{old}}|$. Pada baris pertama c_{old} adalah titik awal a, jadi kolom terakhir di sana menunjukkan seberapa jauh titik pertama dari a. Python mengembalikan pasangan `(root, k)`; Java mengembalikan `record Result(double root, int iterations)`, kelas ringkas yang tersedia di Java sejak versi 16. Kurung yang tidak sah ditolak dengan eksepsi di keduanya: `ValueError` di Python dan `IllegalArgumentException` di Java.

Program pinjaman, **Kode 2.3** dan **Kode 2.4**, membaca harga, angsuran, dan banyaknya bulan dari baris perintah (dengan rencana Dimas sebagai nilai bawaan), mendefinisikan f dari Persamaan (2.3), lalu menjalankan ketiga metode pada $[0.01, 0.05]$ dengan toleransi 10^{-7}.

Kode 2.3 `loan_rate.py`: suku bunga bulanan tersembunyi sebuah rencana cicilan dengan tiga metode.

```python
"""The hidden monthly interest rate of an instalment plan.
Usage: python loan_rate.py [price] [instalment] [months]"""
import sys
from roots import HEADER, bisection, regula_falsi


def power(x, n):
    """x to the whole power n by repeated multiplication."""
    result = 1.0
    for _ in range(n):
        result *= x
    return result


def main():
    price = float(sys.argv[1]) if len(sys.argv) > 1 else 6000000.0
    pay = float(sys.argv[2]) if len(sys.argv) > 2 else 600000.0
    months = int(sys.argv[3]) if len(sys.argv) > 3 else 12

    def f(r):
        """Present value of all instalments at monthly rate r, minus price."""
        return pay * (1 - 1 / power(1 + r, months)) / r - price

    print(f"Price Rp {price:,.0f}, {months} instalments of Rp {pay:,.0f}")
    flat = (pay * months - price) / price / months
    print(f"Flat rate the seller quotes: {flat * 100:.4f} % per month")
    print("\nBisection on [0.01, 0.05]")
    print(HEADER)
    r, _ = bisection(f, 0.01, 0.05, trace=True)
    print("\nRegula falsi on [0.01, 0.05]")
    print(HEADER)
    regula_falsi(f, 0.01, 0.05, trace=True)
    print("\nIllinois on [0.01, 0.05]")
    print(HEADER)
    regula_falsi(f, 0.01, 0.05, trace=True, illinois=True)
    print(f"\nTrue monthly rate : {r * 100:.4f} %")
    print(f"Per year, compound: {(power(1 + r, 12) - 1) * 100:.2f} %")


if __name__ == "__main__":
    main()
```

Kode 2.4 `LoanRate.java`: program yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/**
 * The hidden monthly interest rate of an instalment plan.
 * Usage: java LoanRate [price] [instalment] [months]
 */
public class LoanRate {
    /** x to the whole power n by repeated multiplication. */
    static double power(double x, int n) {
        double result = 1.0;
        for (int i = 0; i < n; i++) {
            result *= x;
        }
        return result;
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        double price = args.length > 0 ? Double.parseDouble(args[0]) : 6000000.0;
        double pay = args.length > 1 ? Double.parseDouble(args[1]) : 600000.0;
        int months = args.length > 2 ? Integer.parseInt(args[2]) : 12;

        // present value of all instalments at monthly rate r, minus the price
        DoubleUnaryOperator f =
                r -> pay * (1 - 1 / power(1 + r, months)) / r - price;

        System.out.printf("Price Rp %,.0f, %d instalments of Rp %,.0f%n",
                          price, months, pay);
        double flat = (pay * months - price) / price / months;
        System.out.printf("Flat rate the seller quotes: %.4f %% per month%n",
                          flat * 100);
        System.out.println("\nBisection on [0.01, 0.05]");
        System.out.println(Roots.HEADER);
        double r = Roots.bisection(f, 0.01, 0.05, 1e-7, 100, true).root();
        System.out.println("\nRegula falsi on [0.01, 0.05]");
        System.out.println(Roots.HEADER);
        Roots.regulaFalsi(f, 0.01, 0.05, 1e-7, 100, true, false);
        System.out.println("\nIllinois on [0.01, 0.05]");
        System.out.println(Roots.HEADER);
        Roots.regulaFalsi(f, 0.01, 0.05, 1e-7, 100, true, true);
        System.out.printf("%nTrue monthly rate : %.4f %%%n", r * 100);
        System.out.printf("Per year, compound: %.2f %%%n",
                          (power(1 + r, 12) - 1) * 100);
    }
}
```

Dua keputusan kecil menjaga keluaran Java dan Python tetap identik. Pangkat $(1+r)^n$ dihitung dengan perkalian berulang dalam fungsi bantu `power`, bukan dengan `Math.pow` atau `**`: IEEE 754 menjamin bahwa satu perkalian memberi hasil yang sama di setiap mesin, sedangkan fungsi pangkat pustaka kedua bahasa bisa berbeda pada bit terakhir. Dan di Java, fungsi `f` adalah lambda yang memakai peubah lokal `pay`, `price`, dan `months`; Java mengizinkannya karena ketiganya tidak pernah diubah setelah diisi (*effectively final*).

2.6 Menjalankannya

```
$ python3 loan_rate.py          (or: java -cp out LoanRate)
```

```
Price Rp 6,000,000, 12 instalments of Rp 600,000
Flat rate the seller quotes: 1.6667 % per month

Bisection on [0.01, 0.05]
  k           a           b           c            f(c)  |c - c_old|
  1  0.01000000  0.05000000  0.03000000     -27597.6039    2.000e-02
  2  0.01000000  0.03000000  0.02000000     345204.7326    1.000e-02
  3  0.02000000  0.03000000  0.02500000     154658.7589    5.000e-03
  4  0.02500000  0.03000000  0.02750000      62522.1949    2.500e-03
  5  0.02750000  0.03000000  0.02875000      17213.5840    1.250e-03
  6  0.02875000  0.03000000  0.02937500      -5253.7705    6.250e-04
  7  0.02875000  0.02937500  0.02906250       5964.4147    3.125e-04
  8  0.02906250  0.02937500  0.02921875        351.4556    1.563e-04
  9  0.02921875  0.02937500  0.02929688      -2452.1233    7.813e-05
 10  0.02921875  0.02929688  0.02925781      -1050.5754    3.906e-05
 11  0.02921875  0.02925781  0.02923828       -349.6203    1.953e-05
 12  0.02921875  0.02923828  0.02922852          0.9026    9.766e-06
 13  0.02922852  0.02923828  0.02923340       -174.3626    4.883e-06
 14  0.02922852  0.02923340  0.02923096        -86.7310    2.441e-06
 15  0.02922852  0.02923096  0.02922974        -42.9145    1.221e-06
 16  0.02922852  0.02922974  0.02922913        -21.0060    6.104e-07
 17  0.02922852  0.02922913  0.02922882        -10.0517    3.052e-07
 18  0.02922852  0.02922882  0.02922867         -4.5746    1.526e-07
 19  0.02922852  0.02922867  0.02922859         -1.8360    7.629e-08

Regula falsi on [0.01, 0.05]
  k           a           b           c            f(c)  |c - c_old|
  1  0.01000000  0.05000000  0.03098945     -62719.8812    2.099e-02
  2  0.01000000  0.03098945  0.02937568      -5278.1637    1.614e-03
  3  0.01000000  0.02937568  0.02924082       -440.7396    1.349e-04
  4  0.01000000  0.02924082  0.02922957        -36.7789    1.125e-05
  5  0.01000000  0.02922957  0.02922863         -3.0690    9.391e-07
  6  0.01000000  0.02922863  0.02922855         -0.2561    7.836e-08

Illinois on [0.01, 0.05]
  k           a           b           c            f(c)  |c - c_old|
  1  0.01000000  0.05000000  0.03098945     -62719.8812    2.099e-02
  2  0.01000000  0.03098945  0.02937568      -5278.1637    1.614e-03
  3  0.01000000  0.02937568  0.02910782       4335.4506    2.679e-04
  4  0.02910782  0.02937568  0.02922862         -2.8128    1.208e-04
  5  0.02910782  0.02922862  0.02922854         -0.0015    7.832e-08

True monthly rate : 2.9229 %
Per year, compound: 41.30 %
```

Ketiga tabel itu menceritakan kisah bab ini.

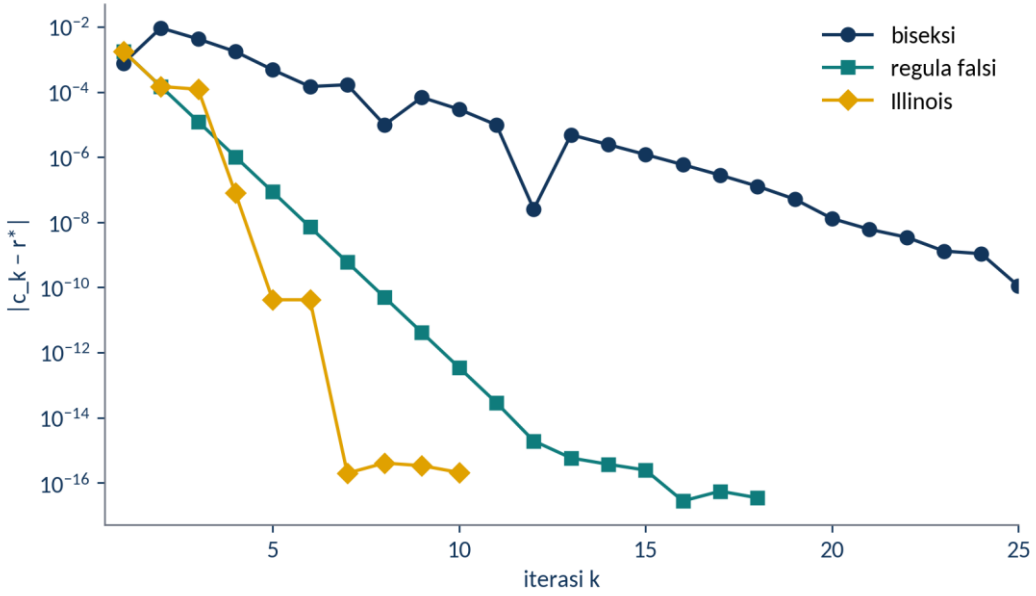
Biseksi mengulang hitungan tangan kita pada empat baris pertamanya, lalu memerlukan tepat 19 langkah seperti diramalkan Persamaan (2.5). Kolom terakhir terbagi dua di setiap baris, sebagaimana diharuskan Persamaan (2.4). Kolom $f(c)$ tidak turun dengan teratur: pada langkah 12 titik tengah 0,02922852 kebetulan berada dalam dua per sejuta poin persentase dari akar, dan f di sana kurang dari satu rupiah, tetapi metodenya tidak menyadarinya dan terus berjalan sampai *batasnya* di bawah toleransi.

Regula falsi hanya memerlukan 6 langkah. Seperti diramalkan di Subbab 2.4.2, ujung kiri $a = 0.01$ tidak pernah bergerak, karena fungsinya cembung; tetapi fungsinya begitu hampir lurus antara 1% dan akar sehingga rangkakannya ringan: setiap langkah memperkecil galat sekitar 12 kali, persis laju $\lambda = 0.083$ yang diramalkan Persamaan (2.7).

Illinois memerlukan 5 langkah. Langkah 1 dan 2 sama dengan regula falsi; setelah langkah 2, a sudah diam dua kali, nilainya dibagi dua, dan langkah 3 melompat ke kiri akar ($f > 0$). Kini a

akhirnya bergerak, dan langkah 4 mendarat dalam jarak 10^{-7} dari akar, langkah 5 dalam jarak 10^{-10}.

Jawaban atas pertanyaan Dimas ada di bagian bawah: suku bunga bulanan yang sebenarnya **2,9229%**, bukan 1,67%, dan bila dimajemukkan selama setahun menjadi **41,30%**. Angka penjual tidak palsu, tetapi mengukur hal lain. **Gambar 2.2** membandingkan ketiga metode menurut jarak c_k dari akar (di sini dihitung dengan presisi penuh), pada skala logaritmik tempat konvergensi linear tampak sebagai garis lurus.



Gambar 2.2 Galat $|c_k - r^*|$ ketiga metode pengurung pada fungsi pinjaman, dilanjutkan melewati toleransi sampai galatnya mencapai batas presisi ganda.

Galat biseksi berzig-zag di sekitar garis lurus dengan kemiringan $\log_{10} 2$ per langkah. Regula falsi turun sepanjang garis lurus yang lebih curam: ia juga linear, dengan laju yang lebih baik pada fungsi yang ramah ini. Illinois melengkung ke bawah, tanda orde konvergensi di atas satu, dan mencapai batas presisi ganda dalam 7 langkah.

2.6.1 Arti 2,92% sebulan dalam rupiah

Suku bunga itu abstrak; jadwal angsuran tidak. Setelah suku bunganya ditemukan, kita dapat mengikuti utang Dimas bulan demi bulan, persis seperti yang dilakukan bank. Pada awal bulan 1 ia berutang Rp6.000.000. Bunga bulan itu $6{,}000{,}000 \times 0.029229 = \text{Rp}175{,}371$; dari angsurannya yang Rp600.000, sebesar itu membayar bunga dan sisanya, Rp424.629, melunasi pokok utang, yang turun menjadi Rp5.575.371. Pada bulan 2 bunganya dihitung dari utang yang lebih kecil, $5{,}575{,}371 \times 0.029229 = \text{Rp}162{,}960$, sehingga bagian angsuran yang mengurangi utang lebih besar, Rp437.040. **Tabel 2.2** memperlihatkan beberapa bulan pilihan.

Tabel 2.2 Utang Dimas bulan demi bulan pada bunga efektif: bunganya turun ketika utangnya turun, dan angsuran kedua belas melunasi utang dengan tepat.

Bulan	Utang di awal	Bunga (2,9229%)	Pokok yang dibayar	Utang di akhir
1	6,000,000	175,371	424,629	5,575,371
2	5,575,371	162,960	437,040	5,138,331
3	5,138,331	150,186	449,814	4,688,517
6	3,749,062	109,580	490,420	3,258,642
9	2,234,380	65,308	534,692	1,699,687
12	582,961	17,039	582,961	0

Tiga hal menonjol. Pertama, utangnya tepat menjadi nol pada angsuran kedua belas: itulah *arti* 2,9229% sebagai akar Persamaan (2.3), dan inilah pemeriksaan kedua yang tidak bergantung pada yang pertama. Kedua, bunga yang dibayar selama setahun berjumlah Rp1.200.000, sama dengan hitungan flat, karena angsurannya sama; hanya *cara menggambarkannya* yang berbeda. Ketiga, bunga pada bulan 1 adalah Rp175.371, bukan Rp100.000: pada bulan-bulan pertama, ketika utangnya besar, Dimas membayar jauh lebih banyak daripada kesan yang ditimbulkan gambaran flat, dan pada bulan-bulan terakhir jauh lebih sedikit. Bunga flat meratakan bunga di atas utang yang tidak rata, dan itulah sebabnya ia menggambarkan biaya pinjaman yang sebenarnya hampir separuh terlalu rendah.

2.7 Tempat Regula Falsi Merangkak

Untuk melihat regula falsi pada keadaan terburuknya, **Kode 2.5** dan **Kode 2.6** menerapkan ketiga metode pada $f(x) = x^{10} - 1$ di [0,1.3], yang akarnya 1.

Kode 2.5 `stagnation.py`: ketiga metode pada $x^{10} - 1$.

```
"""Where regula falsi crawls: f(x) = x^10 - 1 on [0, 1.3]."""
from roots import bisection, regula_falsi


def f(x):
    y = 1.0
    for _ in range(10):
        y *= x
    return y - 1


def main():
    for name, method in (("bisection", bisection),
                         ("regula falsi", regula_falsi),
                         ("Illinois", None)):
        if method is None:
            root, k = regula_falsi(f, 0.0, 1.3, illinois=True)
        else:
            root, k = method(f, 0.0, 1.3)
        print(f"{name:<13}: root {root:.9f} after {k:>3} iterations")


if __name__ == "__main__":
    main()
```

Kode 2.6 `Stagnation.java`: perbandingan yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Where regula falsi crawls: f(x) = x^10 - 1 on [0, 1.3]. */
public class Stagnation {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        DoubleUnaryOperator f = x -> {
            double y = 1.0;
            for (int i = 0; i < 10; i++) {
                y *= x;
            }
            return y - 1;
        };
        Roots.Result[] results = {
            Roots.bisection(f, 0.0, 1.3, 1e-7, 100, false),
            Roots.regulaFalsi(f, 0.0, 1.3, 1e-7, 100, false, false),
            Roots.regulaFalsi(f, 0.0, 1.3, 1e-7, 100, false, true)};
        String[] names = {"bisection", "regula falsi", "Illinois"};
        for (int i = 0; i < 3; i++) {
            System.out.printf("%-13s: root %.9f after %3d iterations%n",
                              names[i], results[i].root(),
                              results[i].iterations());
        }
    }
}
```

```
bisection    : root 1.000000018 after  24 iterations
regula falsi : root 0.999999702 after  65 iterations
Illinois     : root 1.000000000 after  14 iterations
```

Biseksi memerlukan 24 langkah, seperti diramalkan Persamaan (2.5) untuk kurung selebar 1,3 dan $\tau = 10^{-7}$: $\log_2(1.3 \times 10^7) = 23.6$. Regula falsi memerlukan 65, hampir tiga kali lipat, dan *tetap* berhenti dengan galat 3×10^{-7}, tiga kali toleransinya: langkah-langkahnya sudah begitu kecil sehingga uji perubahan terpenuhi padahal titiknya masih merangkak. Inilah kegagalan yang diramalkan di Subbab 2.4.2, dan itulah sebabnya regula falsi sebaiknya tidak dipakai tanpa modifikasi Illinois. Illinois memerlukan 14 langkah dan benar pada kesembilan digit yang dicetak.

2.8 Mengujinya

Pengujian memastikan apa yang harus dilakukan kode, agar perubahan di kemudian hari tidak dapat merusaknya tanpa ketahuan. **Kode 2.7** memeriksa ketiga metode pada $x^2 - 2$, yang akarnya $\sqrt{2}$ kita ketahui, memeriksa bahwa biseksi memerlukan tepat banyak langkah yang diramalkan Persamaan (2.5) (untuk kurung [0,2] dan $\tau = 10^{-6}$, $\log_2(2 \times 10^6) = 20.9$, jadi 21), memeriksa bahwa Illinois mengalahkan regula falsi biasa, dan memeriksa bahwa kurung yang tidak sah ditolak. **Kode 2.8** melakukan hal yang sama dalam Java, sebagai program biasa tanpa kerangka kerja pengujian.

Kode 2.7 `test_roots.py`: uji unit metode-metode pengurung.

```python
"""Tests of the bracketing root finders: python -m unittest test_roots.py"""
import unittest
from roots import bisection, regula_falsi


def square_minus_two(x):
    return x * x - 2


class RootsTest(unittest.TestCase):
    def test_bisection_finds_sqrt2(self):
        root, _ = bisection(square_minus_two, 0.0, 2.0, tol=1e-10)
        self.assertAlmostEqual(root, 2 ** 0.5, places=9)

    def test_bisection_iteration_count(self):
        _, k = bisection(square_minus_two, 0.0, 2.0, tol=1e-6)
        self.assertEqual(k, 21)        # 2 / 2^k < 1e-6 first at k = 21

    def test_regula_falsi_and_illinois(self):
        r1, k1 = regula_falsi(square_minus_two, 0.0, 2.0, tol=1e-10)
        r2, k2 = regula_falsi(square_minus_two, 0.0, 2.0, tol=1e-10,
                              illinois=True)
        self.assertAlmostEqual(r1, 2 ** 0.5, places=9)
        self.assertAlmostEqual(r2, 2 ** 0.5, places=9)
        self.assertLess(k2, k1)

    def test_no_bracket(self):
        with self.assertRaises(ValueError):
            bisection(square_minus_two, 2.0, 3.0)


if __name__ == "__main__":
    unittest.main()
```

Kode 2.8 `RootsTest.java`: pemeriksaan yang sama dalam Java.

```java
import java.util.function.DoubleUnaryOperator;

/** Tests of the bracketing root finders: java -cp out RootsTest */
public class RootsTest {
    static int failures = 0;

    static void check(boolean ok, String name) {
        System.out.println((ok ? "ok    " : "FAILED") + "  " + name);
        if (!ok) {
            failures++;
        }
    }

    public static void main(String[] args) {
        DoubleUnaryOperator f = x -> x * x - 2;
        double s2 = Math.sqrt(2);
        Roots.Result b = Roots.bisection(f, 0.0, 2.0, 1e-10, 100, false);
        check(Math.abs(b.root() - s2) < 1e-9, "bisection finds sqrt 2");
        check(Roots.bisection(f, 0.0, 2.0, 1e-6, 100, false).iterations() == 21,
              "bisection needs 21 iterations for tol 1e-6");
        Roots.Result r1 = Roots.regulaFalsi(f, 0.0, 2.0, 1e-10, 100, false, false);
        Roots.Result r2 = Roots.regulaFalsi(f, 0.0, 2.0, 1e-10, 100, false, true);
        check(Math.abs(r1.root() - s2) < 1e-9 && Math.abs(r2.root() - s2) < 1e-9,
              "regula falsi and Illinois find sqrt 2");
        check(r2.iterations() < r1.iterations(), "Illinois is faster");
        try {
            Roots.bisection(f, 2.0, 3.0, 1e-7, 100, false);
            check(false, "no bracket is refused");
        } catch (IllegalArgumentException e) {
            check(true, "no bracket is refused");
        }
        System.out.println(failures == 0 ? "all tests passed"
                                         : failures + " test(s) failed");
    }
}
```

```
$ python3 -m unittest test_roots.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
$ java -cp out RootsTest
```

```
ok      bisection finds sqrt 2
ok      bisection needs 21 iterations for tol 1e-6
ok      regula falsi and Illinois find sqrt 2
ok      Illinois is faster
ok      no bracket is refused
all tests passed
```

2.9 Membandingkan dengan SciPy

Modul `scipy.optimize` dari SciPy menyediakan beberapa pencari akar pengurung. **Kode 2.9** menanyakan suku bunga Dimas kepada tiga di antaranya.

Kode 2.9 `compare_scipy.py`: suku bunga pinjaman dengan `bisect`, `brentq`, dan TOMS 748 dari SciPy.

```
"""The loan rate with SciPy's root finders (needs SciPy)."""
from scipy.optimize import bisect, brentq, root_scalar

PRICE, PAY, MONTHS = 6000000.0, 600000.0, 12


def f(r):
    return PAY * (1 - (1 + r) ** -MONTHS) / r - PRICE


def main():
    r1 = bisect(f, 0.01, 0.05, xtol=1e-7)
    r2, info = brentq(f, 0.01, 0.05, xtol=1e-12, full_output=True)
    sol = root_scalar(f, bracket=(0.01, 0.05), method="toms748")
    print(f"scipy bisect : {r1 * 100:.8f} %")
    print(f"scipy brentq : {r2 * 100:.8f} %  ({info.iterations} iterations)")
    print(f"toms748      : {sol.root * 100:.8f} %  ({sol.iterations} iterations)")


if __name__ == "__main__":
    main()
```

```
scipy bisect : 2.92285919 %
scipy brentq : 2.92285408 %  (7 iterations)
toms748      : 2.92285408 %  (3 iterations)
```

`bisect` mengembalikan 2,92285919%, persis titik tengah ke-19 biseksi kita, karena metodenya sama dengan toleransi yang sama. `brentq` mengimplementasikan **metode Brent** [7], pencari akar pengurung baku di pustaka numerik sejak tahun 1970-an: ia mempertahankan kurung seperti biseksi, tetapi di dalam kurung mencoba langkah-langkah cepat (langkah secant dari Bab 3 dan interpolasi kuadratik invers), dan kembali ke langkah biseksi setiap kali langkah cepat akan keluar dari kurung atau terlalu lambat menyempitkannya. Ia mencapai toleransi 10^{-12} dalam 7 iterasi dan cocok dengan hasil Illinois kita pada semua digit yang dicetak. `toms748` adalah algoritma Alefeld, Potra, dan Shi [8], metode pengurung yang lebih halus lagi, yang di sini hanya memerlukan 3 iterasi. Dokumentasi SciPy menganjurkan `brentq` sebagai pilihan bawaan untuk akar skalar yang sudah terkurung [9]. Pelajarannya bagi kita menenteramkan: metode Illinois kita yang sederhana mencapai jawaban yang sama dalam 5 langkah, dan program-program kita sepakat dengan pustaka.

2.10 Metode Numerik dalam Praktik: Profitina

Koin utilitas PROFIT milik Profitina dirancang deflasioner: koin baru diterbitkan menurut jadwal yang *meluruh*, dan koin yang dipakai di platform dibakar. Kurva emisi yang meluruh adalah fungsi $E(t)$ yang memberikan banyaknya koin yang telah diterbitkan sampai waktu t; ia naik cepat pada awalnya, lalu makin lama makin lambat menuju sebuah plafon. Pertanyaan tentang kurva seperti itu secara alami adalah pertanyaan pencarian akar. *Kapan emisi kumulatif mencapai separuh plafon? Pada bulan ke berapa emisi bulanan turun di bawah pembakaran bulanan?* Masing-masing menanyakan t tempat suatu fungsi $E(t)$ − tingkat atau $e(t) - b(t)$ bernilai nol.

Fungsi-fungsi seperti itu memiliki dua sifat yang membuat metode pengurung menjadi alat yang tepat. Fungsinya monoton, sehingga pergantian tanda mengurung tepat satu akar, dan kurung alaminya sudah diketahui sebelumnya (dari peluncuran sampai beberapa dasawarsa kemudian). Fungsi monoton dengan kurung yang diketahui adalah persis keadaan ketika jaminan biseksi Persamaan (2.5) paling berharga: perhitungannya tidak mungkin gagal, dan biayanya diketahui sebelum dimulai. Bentuk yang sama muncul di sisi harga platform, tempat langganan Pro yang dibayar dengan PROFIT pada kurs yang ditetapkan *oracle* PROFIT↔IDR memunculkan

pertanyaan seperti "pada harga koin berapa sebuah saldo cukup untuk dua belas kali perpanjangan?": sekali lagi fungsi monoton dan sebuah kurung.

2.11 Memakainya Sungguhan

Program ini siap dipakai untuk rencana cicilan Anda sendiri; hanya tiga bilangan yang berubah. Sepeda motor seharga Rp18.000.000 dengan 24 angsuran Rp950.000:

```
$ python3 loan_rate.py 18000000 950000 24
```

mencetak (di antara tabel-tabelnya) suku bunga bulanan sebenarnya 1,9844% dan suku bunga tahunan majemuk 26,59%, sedangkan bunga flatnya hanya 1,1111% sebulan. Beberapa saran praktis agar program ini dipakai dengan baik:

- **Masukkan setiap pembayaran.** Uang muka mengurangi harga yang dibiayai: masukkan harga dikurangi uang muka. Biaya administrasi yang dibayar di awal bekerja ke arah sebaliknya: itu uang yang Anda bayar tanpa menerima apa-apa, jadi masukkan harga *dikurangi* biaya itu sebagai P, yang menaikkan suku bunganya.
- **Periksa kurungnya.** Program mencari antara 1% dan 5% sebulan. Jika sebuah rencana mempunyai suku bunga lebih rendah atau lebih tinggi, kedua ujung bertanda sama dan program berhenti dengan "no bracket"; lebarkan kurungnya di kode, misalnya menjadi [0.001,0.2].
- **Bandingkan yang sepadan.** Pakai program yang sama untuk setiap penawaran, termasuk penawaran bank; suku bunga yang diiklankan bank biasanya sudah efektif. Angsuran lebih rendah dengan bulan lebih banyak tetap bisa menjadi rencana yang lebih mahal.

Pemeriksaan cepat tanpa komputer juga patut diketahui. Rata-rata sepanjang masa cicilan, utangnya hanya sekitar separuh harga, sehingga suku bunga sebenarnya kira-kira *dua kali* bunga flat; aturan praktis yang sedikit lebih baik adalah $r \approx \frac{2n}{n+1}$ kali bunga flat. Untuk Dimas, $\frac{24}{13} \times 1.667\% = 3.08\%$, sedikit di atas 2,92% yang sebenarnya; untuk sepeda motor tadi, $\frac{48}{25} \times 1.111\% = 2.13\%$ terhadap 1,98% yang sebenarnya. Aturan ini sedikit melebih-lebihkan, karena mengabaikan kenyataan bahwa angsuran awal lebih berbobot daripada angsuran akhir, tetapi aturan ini cara yang sangat baik untuk memilih kurung (dari bunga flat sampai dua kali bunga flat) dan untuk mengenali hasil yang mustahil benar. Jika program suatu saat melaporkan bunga efektif yang *lebih rendah* daripada bunga flat, pasti ada yang salah pada masukannya.

Ada pula godaan untuk hanya membandingkan besar angsuran bulanan. Angsuran yang lebih kecil terasa lebih ringan, dan memang lebih ringan bagi arus kas harian seorang pengemudi ojek; tetapi angsuran yang lebih kecil dengan masa yang lebih panjang bisa berarti total bunga yang jauh lebih besar. Program ini memberikan dua angka yang saling melengkapi: bunga efektif, untuk membandingkan harga uang di antara penawaran, dan jadwal seperti **Tabel 2.2**, untuk melihat berapa rupiah bunga yang dibayar setiap bulan. Keputusan akhirnya menimbang keduanya bersama keadaan keuangan pribadi, dan itu tetap urusan pemakainya.

Metode ini tidak terbatas pada uang. Setiap pertanyaan berbentuk "masukan apa yang memberikan keluaran ini?", ketika keluarannya naik atau turun secara teratur bersama masukannya, adalah sebuah akar: takaran pupuk yang membawa bacaan tanah ke sasaran, suhu ketika baterai mencapai tegangan tertentu, kedalaman air di tangki dengan bentuk apa pun (Soal Latihan 2.6). Tulis $f(x) = \text{output}(x) - \text{target}$, cari dua masukan yang keluarannya berada di sisi berlawanan dari sasaran, lalu lakukan biseksi.

2.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memulai tanpa kurung.** Jika $f(a)$ dan $f(b)$ bertanda sama, biseksi tidak mempunyai bahan untuk bekerja. Tabulasikan f lebih dulu, seperti pada **Tabel 2.1**, dan periksa tandanya; program menolak kurung yang tidak sah alih-alih mengembalikan jawaban asal-asalan.
- **Mengurung kutub, bukan akar.** Fungsi yang melompat dari $+\infty$ ke $-\infty$ berganti tanda tanpa mempunyai akar. Periksa bahwa $|f(c)|$ menjadi kecil di akhir, bukan hanya bahwa kurungnya menyempit.
- **Mengevaluasi f di tempat ia tidak terdefinisi.** Fungsi pinjaman membagi dengan r, sehingga kurungnya tidak boleh memuat $r = 0$ (Soal Latihan 2.4 memperlihatkan apa yang terjadi bila memuatnya).
- **Berhenti hanya karena $|f(c)|$ kecil.** Apakah $|f| < 1$ itu kecil bergantung pada satuan f. Berhentilah berdasarkan perubahan c atau lebar kurung, relatif terhadap besar akarnya.
- **Memercayai uji berhenti regula falsi biasa.** Barisan yang merangkak berubah sedikit per langkah padahal masih jauh dari akar (Subbab 2.7). Pakai modifikasi Illinois, atau biseksi bila jaminan diperlukan.
- **Menghitung titik tengah sebagai $(a + b)/2$.** Pakai $a + (b - a)/2$, yang tidak mungkin keluar dari kurung.
- **Membaca bunga flat sebagai bunga efektif.** Untuk rencana cicilan, keduanya berbeda hampir dua kali lipat; bandingkan penawaran menurut bunga efektif.
- **Lupa memeriksa jawaban.** Pencari akar tetap mengembalikan sebuah bilangan meskipun kurung, fungsi, atau satuannya keliru. Masukkan akarnya kembali ke dalam masalah, seperti yang dilakukan jadwal pada **Tabel 2.2**, dan lihat apakah ceritanya menjadi benar.

2.13 Ringkasan Bab

- Pertanyaan praktis “masukan apa yang memberikan keluaran ini?” menjadi masalah pencarian akar $f(x) = 0$; untuk rencana cicilan, f adalah nilai kini angsuran dikurangi harga (Persamaan (2.2), Persamaan (2.3)).
- Jika f kontinu dan $f(a)f(b) < 0$, Teorema Nilai Antara menjamin adanya akar di $[a, b]$: sebuah kurung.
- Biseksi membagi dua kurung di setiap langkah (**Pseudokode 2.1**). Galatnya dibatasi $(b_0 - a_0)/2^k$ (Persamaan (2.4)), sehingga banyaknya langkah untuk suatu toleransi diketahui sebelumnya (Persamaan (2.5)): satu digit biner, atau sekitar 0,3 digit desimal, per langkah.
- Regula falsi memotong kurung di tempat talibusur memotong nol (Persamaan (2.6), **Pseudokode 2.2**); sering lebih cepat, tetapi bila satu ujungnya terpaku ia konvergen linear dengan laju Persamaan (2.7) yang bisa mendekati 1, dan uji berhentinya bisa menyesatkan.
- Modifikasi Illinois membagi dua nilai di ujung yang macet (Persamaan (2.8)) dan menaikkan orde konvergensi menjadi sekitar 1,44.
- Untuk ponsel Dimas, ketiga metode memerlukan 19, 6, dan 5 langkah dan sepakat bahwa bunga efektifnya 2,9229% sebulan (41,30% setahun, majemuk), terhadap bunga flat 1,67%. `bisect`, `brentq`, dan TOMS 748 dari SciPy memberikan jawaban yang sama.

2.14 Soal Latihan

1. Dengan tangan, jalankan empat langkah pertama biseksi untuk $f(x) = x^2 - 2$ pada $[1,2]$. Berikan setiap kurung dan titik tengahnya, serta batas galat Persamaan (2.4) setelah langkah 4.

2. Buktikan Persamaan (2.4) dengan induksi pada k, lalu pakai untuk menunjukkan bahwa biseksi selalu konvergen untuk f yang kontinu dan sebuah kurung.
3. Jalankan dua langkah regula falsi untuk $f(x) = x^2 - 2$ pada $[1,2]$ dengan tangan. Ujung mana yang terpaku, dan mengapa?
4. Apa yang terjadi bila kurung pinjaman adalah $[0,0.05]$, bukan $[0.01,0.05]$? Cobalah di Python dan di Java, lalu jelaskan dua perilaku yang berbeda itu. Bagaimana f dapat diperbaiki di $r = 0$?
5. Sebuah toko menawarkan lemari es seharga Rp6.000.000 dalam 24 angsuran Rp350.000. Sebelum menjalankan program, taksirlah dari bunga flat dan dari rencana Dimas apakah bunga efektifnya lebih tinggi atau lebih rendah daripada 2,92%. Berapa langkah biseksi yang diperlukan kurung $[0.001,0.1]$ untuk $\tau = 10^{-7}$? Lalu jalankan programnya.
6. Tangki air berbentuk bola berjari-jari $R = 1$ m berisi, pada kedalaman h, volume $V(h) = \pi h^2(3R - h)/3$. Tulis program yang mencari kedalaman untuk 500, 1000, 2000, 3000, dan 4000 liter dengan biseksi, untuk ditandai pada sebuah pengukur.
7. Mengapa regula falsi membiarkan ujung kiri terpaku pada fungsi Dimas, sedangkan Illinois menggerakkannya? Jawablah dengan melihat tanda $f(c)$ pada ketiga tabel di Subbab 2.6.

2.15 Penyelesaian Soal Latihan

1. $f(1) = -1$, $f(2) = 2$. Langkah 1: $c = 1.5$, $f = 0.25 > 0$, kurung $[1,1.5]$. Langkah 2: $c = 1.25$, $f = -0.4375$, kurung $[1.25,1.5]$. Langkah 3: $c = 1.375$, $f = -0.109375$, kurung $[1.375,1.5]$. Langkah 4: $c = 1.4375$, $f = 0.06640625$, kurung $[1.375,1.4375]$. Batas setelah langkah 4 adalah $(2-1)/2^4 = 0.0625$; memang $|1.4375 - 1.41421| = 0.023$. Jejak `bisection(lambda x: x * x - 2, 1, 2, tol=1e-3, trace=True)` memperlihatkan baris-baris yang sama dan berhenti setelah 10 langkah di 1,41504, karena $\log_2(1/10^{-3}) = 9.97$.

2. Misalkan $[a_k, b_k]$ adalah kurung setelah k langkah, dengan $[a_0, b_0]$ sebagai awalnya. Klaim: $b_k - a_k = (b_0 - a_0)/2^k$. Untuk $k = 0$ klaim ini benar. Jika benar untuk k, kurung berikutnya adalah separuh dari $[a_k, b_k]$, sehingga lebarnya $(b_0 - a_0)/2^{k+1}$. Titik tengah $c_{k+1} = a_k + (b_k - a_k)/2$ dan akar r^* keduanya berada di $[a_k, b_k]$, dan titik tengah berjarak paling jauh setengah lebar dari titik mana pun di dalamnya: $|c_{k+1} - r^*| \le (b_k - a_k)/2 = (b_0 - a_0)/2^{k+1}$. Itulah Persamaan (2.4) untuk $k + 1$. Setiap kurung memuat akar menurut Teorema Nilai Antara, karena metode menyimpan separuh yang memuat pergantian tanda. Karena batasnya menuju nol, $c_k \to r^*$.

3. Langkah 1: $c = 2 - 2 \times (1-2)/(-1-2) = 2 - 2/3 = 1.33333$, $f(c) = -0.22222 < 0$, jadi a berpindah: kurung $[1.33333,2]$. Langkah 2: $c = 2 - 2 \times (1.33333 - 2)/(-0.22222 - 2) = 2 - 1.33333/2.22222 = 1.4$, $f = -0.04$, kurung $[1.4,2]$. Ujung kanan $b = 2$ terpaku: $x^2 - 2$ cembung, dan pada fungsi cembung yang naik setiap talibusur memotong sumbu di kiri akar, sehingga $f(c) < 0$ setiap kali.

4. Di $r = 0$ rumusnya membagi 0 dengan 0. Di Python, `/ r` dengan `r = 0.0` langsung memunculkan `ZeroDivisionError` ketika `bisection` mengevaluasi $f(a)$. Di Java, 0.0/0.0 adalah `NaN` tanpa galat apa pun; hasil kali `fa * fb` lalu `NaN`, dan `NaN > 0` bernilai salah, sehingga pemeriksaan kurung *lolos*. Perulangan kemudian membandingkan tanda dengan `NaN`: `fa * fc < 0` selalu salah, sehingga a berpindah ke c, dan karena `fa` menjadi bilangan biasa setelah langkah pertama, metodenya pulih karena kebetulan dan mengembalikan akar yang benar. Mengandalkan kebetulan seperti itu adalah kesalahan. Perbaikannya adalah memakai limit ketika $r \to 0$: $(1 - (1+r)^{-n})/r \to n$, jadi definisikan $f(0) = nA - P$ di kode (`if r == 0: return months * pay - price`).

5. Bunga flatnya $(24 \times 350{,}000 - 6{,}000{,}000)/6{,}000{,}000/24 = 1.667\%$, sama dengan milik Dimas. Tetapi dengan masa yang lebih panjang, utangnya tetap tinggi lebih lama, sehingga bunga flat yang sama sepadan dengan bunga efektif yang *lebih rendah*. Kurung itu memerlukan $\log_2(0.099/10^{-7}) = 19.9$, jadi 20 langkah. `python3 loan_rate.py 6000000 350000 24` memberikan 2,8874% sebulan dan 40,72% setahun: lebih rendah daripada 2,92%, sesuai taksiran.

6. **Kode 2.10** dan **Kode 2.11** memakai kurung $[0, 2R]$ (kosong sampai penuh), tempat $V(h) - V_{\text{target}}$ berganti tanda untuk setiap sasaran di bawah volume penuh $\frac{4}{3}\pi R^3 = 4{,}189$ liter, dan toleransi 10^{-6} m.

Kode 2.10 `sphere_tank.py`: kedalaman air di tangki bola untuk volume-volume tertentu.

```python
"""Exercise 2.6: how deep is the water in a spherical tank holding V litres?"""
import math
from roots import bisection

RADIUS = 1.0                 # metres


def volume(h):
    """Water volume in cubic metres when the depth is h metres."""
    return math.pi * h * h * (3 * RADIUS - h) / 3


def main():
    for litres in (500, 1000, 2000, 3000, 4000):
        target = litres / 1000.0
        h, k = bisection(lambda d: volume(d) - target, 0.0, 2 * RADIUS,
                         tol=1e-6)
        print(f"{litres:>5} L: depth {h * 100:6.2f} cm  ({k} iterations)")


if __name__ == "__main__":
    main()
```

Kode 2.11 `SphereTank.java`: hal yang sama dalam Java.

```java
import java.util.Locale;

/** Exercise 2.6: how deep is the water in a spherical tank holding V litres? */
public class SphereTank {
    static final double RADIUS = 1.0;            // metres

    /** Water volume in cubic metres when the depth is h metres. */
    static double volume(double h) {
        return Math.PI * h * h * (3 * RADIUS - h) / 3;
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        for (int litres : new int[] {500, 1000, 2000, 3000, 4000}) {
            double target = litres / 1000.0;
            Roots.Result r = Roots.bisection(d -> volume(d) - target, 0.0,
                                             2 * RADIUS, 1e-6, 100, false);
            System.out.printf("%5d L: depth %6.2f cm  (%d iterations)%n",
                              litres, r.root() * 100, r.iterations());
        }
    }
}
```

```
  500 L: depth  43.11 cm  (21 iterations)
 1000 L: depth  63.55 cm  (21 iterations)
 2000 L: depth  96.99 cm  (21 iterations)
 3000 L: depth 129.70 cm  (21 iterations)
 4000 L: depth 174.37 cm  (21 iterations)
```

Setiap sasaran memerlukan 21 langkah, karena $\log_2(2/10^{-6}) = 20.9$ berapa pun sasarannya: biaya biseksi tidak bergantung pada fungsinya. Tanda-tandanya jauh dari berjarak sama; 500 liter pertama mengisi 43 cm, 1.000 liter terakhir (dari 3.000 ke 4.000) mengisi 45 cm, karena bola sempit di bawah dan di atas serta lebar di tengah. Bab 3 menandai pengukur untuk silinder yang berbaring dengan metode Newton.

7. Pada tabel regula falsi setiap $f(c)$ negatif, sehingga akar selalu berada di $[a, c]$ dan hanya b yang berpindah; fungsi yang cembung menempatkan titik nol setiap talibusur di kanan akar. Pada tabel Illinois, $f(c)$ negatif di langkah 1 dan 2; setelah yang kedua, $f(a)$ dibagi dua, talibusur berikutnya lebih landai di ujung kiri, titik nolnya jatuh di *kiri* akar, $f(c) = +4{,}335 > 0$ di langkah 3, dan a berpindah.

Daftar Pustaka

[1] Otoritas Jasa Keuangan, *Peraturan OJK Nomor 22 Tahun 2023 tentang Pelindungan Konsumen dan Masyarakat di Sektor Jasa Keuangan*, Dec. 2023. [Online]. Available: https://www.ojk.go.id/id/regulasi/Pages/POJK-22-Tahun-2023-Pelindungan-Konsumen-dan-Masyarakat-di-Sektor-Jasa-Keuangan.aspx

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[4] J.-L. Chabert, Ed., *A History of Algorithms: From the Pebble to the Microchip*. Berlin, Germany: Springer, 1999.

[5] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[6] M. Dowell and P. Jarratt, "A modified regula falsi method for computing the root of an equation," *BIT Numerical Mathematics*, vol. 11, no. 2, pp. 168–174, 1971.

[7] R. P. Brent, *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973.

[8] G. E. Alefeld, F. A. Potra, and Y. Shi, "Algorithm 748: Enclosing zeros of continuous functions," *ACM Transactions on Mathematical Software*, vol. 21, no. 3, pp. 327–344, 1995.

[9] SciPy Developers, "Optimization and root finding (scipy.optimize)," *SciPy 1.18.1 Reference Guide*, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html

Bab 3

Mencari Akar II: Newton-Raphson, Secant, dan Iterasi Titik Tetap

Biseksi aman tetapi lambat: satu digit biner per langkah, bagaimanapun bentuk fungsinya. Bila kita mengetahui lebih banyak tentang fungsinya, terutama kemiringannya, kita dapat bekerja jauh lebih baik. Bab ini membangun tiga metode *terbuka* klasik, yang tidak mempertahankan kurung tetapi mengikuti bentuk fungsi itu sendiri menuju akarnya: Newton-Raphson, yang mengikuti garis singgung; metode secant, yang mengikuti garis melalui dua titik terakhir; dan iterasi titik tetap, yang menulis ulang persamaan sebagai $x = g(x)$ lalu cukup mengulanginya. Di akhir bab Anda akan mampu menurunkan metode Newton dari rumus Taylor dan membuktikan bahwa di dekat akar sederhana metode itu melipatduakan banyaknya digit benar di setiap langkah; menurunkan metode secant dan menjelaskan mengapa ordenya adalah rasio emas 1,618; menyatakan kapan iterasi titik tetap konvergen dan seberapa cepat; mengukur orde konvergensi metode apa pun dari galat-galatnya; mengenali cara-cara Newton gagal (garis singgung datar, siklus, divergensi) dan menjaganya; serta memakai ketiganya dalam Java dan Python untuk menandai tongkat ukur sebuah truk air, sambil memeriksa hasilnya terhadap SciPy.

3.1 Truk Air Pak Harun

Setiap musim kemarau, dari Juli sampai Oktober, sumur-sumur di banyak desa di lereng Gunungkidul menyusut, dan rumah tangga membeli air per tangki. Pak Harun memiliki truk kecil dengan tangki baja yang berbaring di baknya: silinder sepanjang 3,20 m dengan jari-jari 0,90 m, yang bila penuh memuat sedikit lebih dari 8.000 liter. Pelanggannya tidak membeli satu tangki penuh. Sebuah keluarga mengisi bak penampungnya dengan 1.000 atau 2.000 liter, sebuah warung makan membeli 500, sebuah masjid menjelang salat Jumat membeli 3.000. Di setiap perhentian Pak Harun harus tahu berapa banyak air yang sudah ia keluarkan.

Truk itu tidak mempunyai meteran. Yang ada hanyalah tongkat ukur: sebatang kayu panjang yang ia turunkan tegak lurus melalui lubang di atas tangki sampai menyentuh dasar, lalu ia tarik keluar untuk melihat seberapa dalam bagian yang basah. Seandainya tangkinya berdiri tegak, kedalaman sebanding dengan volume, dan menandai tongkat itu mudah: 8.000 liter pada 1,80 m, jadi 1.000 liter setiap 22,5 cm. Tetapi tangkinya berbaring, dan penampangnya lingkaran. Di dekat dasar dan puncak, tempat lingkaran itu sempit, seribu liter membentuk lapisan yang tebal; di tengah, tempat lingkaran itu lebar, seribu liter yang sama membentuk lapisan yang tipis. Tanda-tanda pada tongkat harus berjarak tidak sama, dan tidak ada yang menjual truk itu beserta tongkat yang sudah bertanda.

Keponakan Pak Harun, yang bersekolah di sebuah SMK di Wonosari, mencari rumus volume air dalam silinder yang berbaring pada kedalaman tertentu (Subbab 3.2 menurunkannya). Itu menyelesaikan separuh masalah yang keliru. Rumus itu memberikan volume untuk suatu kedalaman, padahal Pak Harun memerlukan kedalaman untuk suatu volume: di mana harus menoreh takik untuk 1.000 liter, untuk 2.000, dan seterusnya. Rumus itu memuat kedalaman di dalam arkus kosinus *dan* di bawah tanda akar, dan tidak ada cara untuk menyelesaikannya bagi kedalaman. Tanda untuk 3.000 liter adalah akar sebuah persamaan.

Bab 2 dapat menemukannya dengan biseksi dalam sekitar 30 langkah per tanda. Bab ini menemukannya dalam 4. Perbedaan itu hampir tidak berarti untuk delapan tanda pada sebatang tongkat, tetapi sangat berarti ketika sebuah akar harus ditemukan jutaan kali: di setiap langkah simulasi, di setiap piksel gambar yang dirender, di setiap prediksi model *machine learning*. Metode Newton, jantung bab ini, adalah salah satu algoritma yang paling banyak dipakai di seluruh dunia komputasi.

3.2 Gagasannya dengan Kata-Kata Sederhana

3.2.1 Volume air dalam tangki yang berbaring

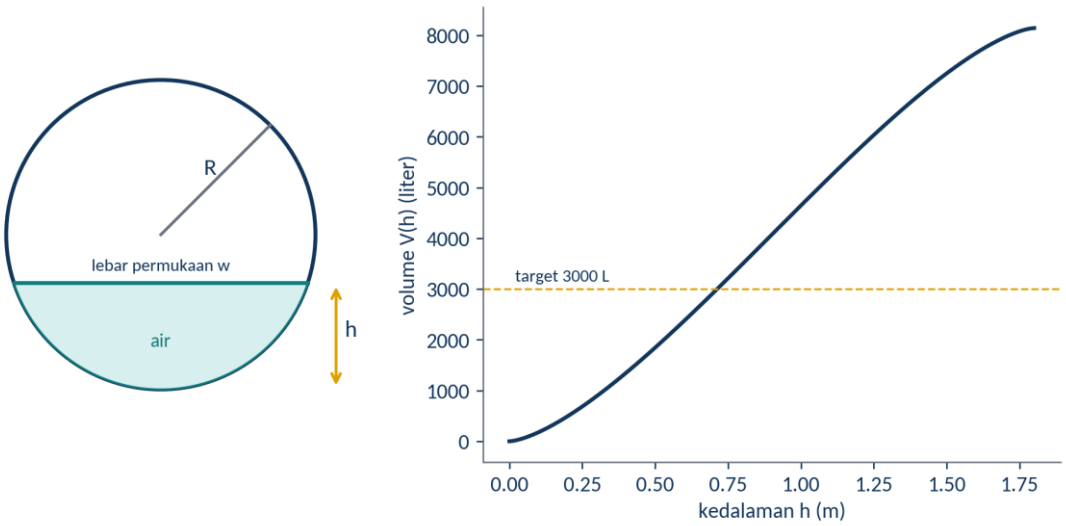
Lihatlah tangki itu dari ujungnya. Air mengisi sebuah *tembereng lingkaran*: bagian lingkaran berjari-jari R di bawah garis mendatar setinggi h dari dasar. **Gambar 3.1** memperlihatkannya. Garis itu berjarak $R - h$ di bawah pusat. Tarik dua jari-jari dari pusat ke titik-titik tempat permukaan air menyentuh lingkaran; masing-masing membentuk sudut θ dengan garis tegak ke bawah, dengan $\cos\theta = (R - h)/R$. Luas tembereng adalah luas "potongan kue" di antara kedua jari-jari dikurangi segitiga di atas air:

- potongan kue itu merentang sudut 2θ, sehingga luasnya $\frac{2\theta}{2\pi}\pi R^2 = R^2\theta$;
- segitiga itu bertinggi $R - h$ dan beralas sama dengan lebar permukaan $w = 2\sqrt{R^2 - (R-h)^2} = 2\sqrt{2Rh - h^2}$, sehingga luasnya $(R - h)\sqrt{2Rh - h^2}$.

Mengalikan luas dengan panjang tangki L memberikan volumenya:

$$V(h) = L\left[R^2 \arccos\left(\frac{R-h}{R}\right) - (R-h)\sqrt{2Rh - h^2}\right] \tag{3.1}$$

Periksa keadaan ekstremnya. Pada $h = 0$, $\arccos 1 = 0$ dan akar kuadratnya 0, sehingga $V = 0$. Pada $h = R$ (setengah penuh), $\arccos 0 = \pi/2$ dan $R - h = 0$, sehingga $V = L\pi R^2/2$, separuh volume penuh. Pada $h = 2R$, $\arccos(-1) = \pi$ dan akar kuadratnya kembali 0, sehingga $V = L\pi R^2$. Untuk Pak Harun, $L\pi R^2 = 3.20 \times \pi \times 0.81 = 8.143\ \text{m}^3$, yaitu 8.143 liter.



Gambar 3.1 Tangki truk air dilihat dari ujungnya (kiri): air mengisi tembereng lingkaran dengan kedalaman h dan lebar permukaan w. Volume Persamaan (3.1) sebagai fungsi kedalaman (kanan) berbentuk S, paling curam pada setengah kedalaman.

Kita juga memerlukan kemiringan kurva ini, yaitu turunan $V'(h)$. Menurunkan Persamaan (3.1) suku demi suku memerlukan beberapa baris kalkulus, tetapi ada jalan pintas yang menjelaskan jawabannya. Jika kedalaman naik sebesar Δh yang sangat kecil, air tambahannya adalah lempengan tipis dengan luas permukaan air, $w \times L$, dan tebal Δh. Jadi $\Delta V \approx wL\,\Delta h$, dan

$$V'(h) = L\,w = 2L\sqrt{2Rh - h^2} \tag{3.2}$$

Kemiringannya terbesar pada setengah kedalaman, tempat permukaan paling lebar ($w = 2R$), dan nol di dasar dan di puncak, tempat permukaan menyusut menjadi garis. Itulah bentuk S pada **Gambar 3.1**.

3.2.2 Ikuti garis singgungnya

Untuk menandai 3.000 liter, kita memerlukan h dengan $V(h) = 3$ m^3, yaitu akar dari $f(h) = V(h) - 3$. Tebakan pertama Pak Harun adalah letak tanda seandainya tangkinya berbentuk kotak: 3.000 dari 8.143 liter adalah 36,8%, dan 36,8% dari tinggi 1,80 m adalah $h_0 = 0.6631$ m.

Pada tebakan ini, Persamaan (3.1) memberikan $V(0.6631) = 2.7231$ m^3: kurang 277 liter. Seberapa dalam lagi airnya harus naik? Kemiringan Persamaan (3.2) mengatakan bahwa di dekat h_0, setiap meter kedalaman menambah $V'(0.6631) = 5.5570$ m^3. Untuk menambah 0.2769 m^3 yang kurang, kita memerlukan sekitar $0.2769/5.5570 = 0.0498$ m lagi. Tebakan baru adalah $h_1 = 0.6631 + 0.0498 = 0.7130$ m.

Itulah seluruh gagasan metode Newton: pada tebakan saat ini, anggaplah fungsinya adalah garis lurus dengan nilai dan kemiringan yang sama, yaitu *garis singgung*, lalu lompat ke tempat garis itu mencapai sasaran. Subbab berikutnya menunjukkan bahwa gagasan sederhana ini luar biasa ampuh: tebakan h_1 sudah berjarak kurang dari 0,4 mm dari tanda yang sebenarnya, dan langkah berikutnya berjarak kurang dari seperseratus ribu milimeter.

3.3 Newton-Raphson

3.3.1 Penurunan

Misalkan x_k adalah tebakan saat ini untuk akar f. Rumus Taylor (Subbab 1.12) menguraikan f di sekitar x_k:

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{f''(\xi)}{2}(x - x_k)^2$$

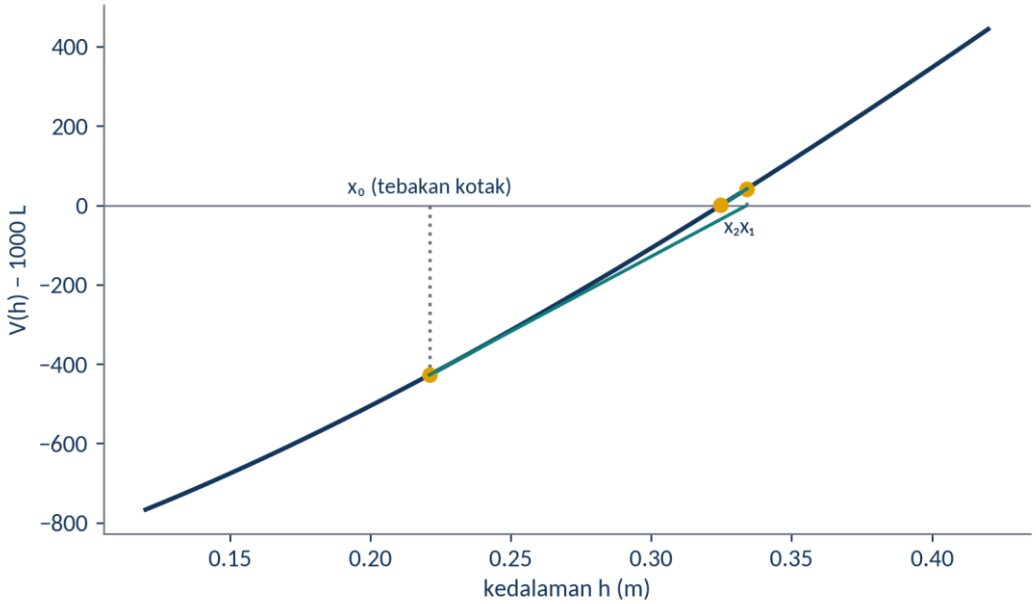
Buang suku terakhir dan samakan sisanya, garis singgung itu, dengan nol. Menyelesaikannya untuk x memberikan tebakan berikutnya,

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \tag{3.3}$$

yaitu iterasi **Newton-Raphson**. Isaac Newton menguraikan gagasan ini pada 1669 untuk polinomial, dalam bentuk yang lebih rumit; Joseph Raphson menerbitkan bentuk yang lebih sederhana pada 1690, dan Thomas Simpson memberikan versi umumnya untuk fungsi apa pun yang mempunyai turunan pada 1740 [1]. Metode ini sudah dikenal jauh sebelumnya dalam kasus-kasus khusus: metode Babilonia untuk akar kuadrat, yang dikaitkan dengan Heron dari

Aleksandria, adalah metode Newton untuk $f(x) = x^2 - a$, karena $x - (x^2 - a)/(2x) = \frac{1}{2}(x + a/x)$.

Gambar 3.2 menggambar dua langkah Newton untuk tanda 1.000 liter, tempat kurvanya melengkung lebih kuat daripada di 3.000 liter. Setiap langkah mengikuti garis singgung dari titik saat ini pada kurva turun ke sumbu.



Gambar 3.2 Metode Newton untuk tanda 1.000 liter: dari tebakan kotak x_0 garis singgung sedikit melampaui ke x_1, dan garis singgung kedua mendarat hampir tepat di akar x_2.

Pseudokode 3.1 menuliskan prosedurnya beserta pengaman yang diperlukan implementasi sungguhan: uji turunan nol, toleransi pada perubahan x, dan batas banyaknya iterasi.

Pseudokode 3.1 Metode Newton-Raphson dengan tiga pengamannya.

```
NEWTON(f, df, x0, tol, maxIter)
 1  x = x0
 2  for k = 1 to maxIter
 3      d = df(x)
 4      if d == 0: error "f'(x) = 0: the tangent is flat"
 5      x_new = x - f(x) / d
 6      if |x_new - x| < tol: return x_new
 7      x = x_new
 8  error "no convergence in maxIter iterations"
```

Uji berhentinya layak dilihat sekali lagi. Untuk metode yang konvergen kuadratik, perubahan di antara dua iterasi adalah taksiran yang sangat baik bagi galat iterasi yang *lebih lama*: $x_{k+1} - x_k = e_{k+1} - e_k \approx -e_k$, karena e_{k+1} jauh lebih kecil daripada e_k. Jadi ketika uji $|x_{k+1} - x_k| < \tau$ berhasil, galat x_k sekitar τ, dan galat x_{k+1} yang dikembalikan sekitar $C\tau^2$, jauh lebih kecil. Dengan $\tau = 10^{-9}$, seperti pada program tongkat ukur, kedalaman yang dikembalikan benar sampai digit terakhir presisi ganda. Metode yang konvergen linear dengan laju λ tidak mendapat bonus seperti itu: perubahannya $|x_{k+1} - x_k| \approx (1 - \lambda)|e_k|$ bisa jauh *lebih kecil* daripada galatnya bila λ mendekati 1, dan begitulah regula falsi mengelabui uji berhentinya di Subbab 2.7.

Berbeda dengan biseksi, metode Newton mengembalikan galat, bukan tebakan terakhirnya, ketika kehabisan iterasi. Tebakan terakhir biseksi selalu berada di dalam kurung dan karena itu selalu berguna; tebakan terakhir Newton bisa berada di mana saja.

3.3.2 Mengapa begitu cepat: konvergensi kuadratik

Misalkan r adalah akarnya dan $e_k = x_k - r$ galat tebakan ke-k. Uraikan lagi f di sekitar x_k, kali ini dievaluasi *di akar*, tempat $f(r) = 0$:

$$0 = f(r) = f(x_k) + f'(x_k)(r - x_k) + \frac{f''(\xi_k)}{2}(r - x_k)^2$$

Bagi dengan $f'(x_k)$ lalu susun ulang: $x_k - \frac{f(x_k)}{f'(x_k)} - r = \frac{f''(\xi_k)}{2f'(x_k)}(x_k - r)^2$. Ruas kiri tepat sama dengan $x_{k+1} - r$, sehingga

$$e_{k+1} = \frac{f''(\xi_k)}{2\,f'(x_k)}\,e_k^2 \approx C\,e_k^2, \qquad C = \frac{f''(r)}{2\,f'(r)} \tag{3.4}$$

Galat baru sebanding dengan *kuadrat* galat lama [2]. Jika galatnya 10^{-3}, galat berikutnya sekitar $C \times 10^{-6}$, lalu $C^3 \times 10^{-12}$: banyaknya digit benar kira-kira *berlipat dua* di setiap langkah. Inilah **konvergensi kuadratik**. Secara formal, sebuah barisan konvergen dengan **orde** p dan konstanta asimtotik C jika

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|\,e_k\,|^p} = |C| \tag{3.5}$$

Batas galat biseksi dan regula falsi mempunyai $p = 1$ (linear), Illinois mempunyai $p \approx 1.44$, Newton mempunyai $p = 2$ pada akar *sederhana*, yaitu akar dengan $f'(r) \neq 0$.

Periksa pada tangki. Di akar 3.000 liter $r = 0.712612$, turunan kedua V adalah $V''(r) = 2L(R - r)/\sqrt{2Rr - r^2} = 1.362$, dan $V'(r) = 5.634$, sehingga $C = 1.362/(2 \times 5.634) = 0.121$. Galat pertamanya $e_0 = 0.6631 - 0.7126 = -0.0495$; Persamaan (3.4) meramalkan $e_1 \approx 0.121 \times 0.0495^2 = 2.96 \times 10^{-4}$, dan program akan menemukan 3.56×10^{-4} (ramalan itu hanya kira-kira selama galatnya masih besar). Lalu rumus itu meramalkan $e_2 \approx 0.121 \times (3.56 \times 10^{-4})^2 = 1.53 \times 10^{-8}$, dan program menemukan tepat 1.53×10^{-8}. Rumus itu juga mengatakan bahwa galat setelah langkah pertama *positif* (kurva di sini melengkung ke atas, $C > 0$): Newton mendekati akar ini dari atas.

Harga kecepatan ini adalah turunan. Setiap langkah memerlukan $f'(x_k)$, yang harus diturunkan dengan tangan (seperti yang kita lakukan pada Persamaan (3.2)), dihampiri, atau dihitung dengan *diferensiasi otomatis*, teknik yang dijumpai Bab 14 di dalam pustaka *machine learning*. Dan jaminannya hanya *lokal*: Persamaan (3.4) berlaku bila x_k sudah dekat dengan akar sederhana. Dari titik awal yang buruk, apa saja bisa terjadi.

3.3.3 Ketika Newton gagal

Empat hal bisa salah, dan setiap pemakai metode Newton patut mengetahuinya [2], [3].

1. **Garis singgung datar.** Jika $f'(x_k) = 0$, garis singgungnya mendatar dan tidak pernah bertemu sumbu; Persamaan (3.3) membagi dengan nol. Jika $f'(x_k)$ hanya kecil, langkahnya sangat besar dan melempar tebakan jauh sekali. Untuk tangki, turunannya nol di dasar dan di puncak, sehingga tebakan $h = 0$ atau $h = 2R$ berakibat fatal. Tebakan kotak menghindari keduanya.
2. **Divergensi.** Untuk $f(x) = x^{1/3}$, yang akarnya hanya 0, langkah Newton adalah $x - x^{1/3}/\left(\frac{1}{3}x^{-2/3}\right) = x - 3x = -2x$: setiap langkah *melipatduakan* jarak dari akar, bergantian di

dua sisi. Fungsinya terlalu curam di akarnya ($f'(0)$ tak hingga) sehingga garis singgung tidak menjadi penunjuk jalan yang berguna.

3. **Siklus.** Untuk $f(x) = x^3 - 2x + 2$ mulai dari $x_0 = 0$: $f(0) = 2$, $f'(0) = -2$, sehingga $x_1 = 1$; $f(1) = 1$, $f'(1) = 1$, sehingga $x_2 = 0$, dan metodenya memantul antara 0 dan 1 selamanya. Akarnya, di dekat -1.77, tidak pernah didekati.
4. **Akar ganda.** Pada akar tempat $f'(r) = 0$ juga, seperti $x = 1$ untuk $f(x) = (x-1)^2$, Persamaan (3.4) tidak berlaku. Newton tetap konvergen, tetapi hanya linear: untuk akar ganda, setiap langkah membagi dua galatnya, tidak lebih baik daripada biseksi.

Subbab 3.9 menjalankan tiga kegagalan pertama. Pertahanan praktisnya adalah tebakan awal yang baik (dari tabel, grafik, fisika masalahnya, atau beberapa langkah biseksi), batas banyaknya iterasi, dan, dalam pustaka yang kokoh, sebuah *pengaman* yang kembali ke biseksi bila langkah Newton keluar dari kurung yang diketahui, yaitu gagasan di balik metode Brent di Bab 2.

3.4 Metode Secant

Bila turunannya tidak tersedia atau mahal, turunan itu dapat diganti dengan kemiringan garis melalui dua tebakan terakhir, sebuah *secant* (tali busur) kurva:

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}$$

Memasukkannya ke Persamaan (3.3) memberikan **metode secant**:

$$x_{k+1} = x_k - f(x_k)\,\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \tag{3.6}$$

Rumusnya sama dengan rumus regula falsi di Subbab 2.4.1, dengan satu perbedaan penting: metode secant selalu memakai *dua titik terbaru*, entah keduanya mengurung akar atau tidak, sedangkan regula falsi mempertahankan kurung. Melepas kurung berarti melepas jaminan, tetapi juga menyingkirkan ujung yang macet, dan hasilnya jauh lebih cepat. Analisis yang cermat, serupa dengan analisis untuk Newton tetapi dengan dua galat sebelumnya, menunjukkan bahwa $e_{k+1} \approx C\, e_k\, e_{k-1}$ dengan $C = f''(r)/(2f'(r))$ [2]. Jika ordenya p, maka $e_{k+1} \sim e_k^{\,p}$ dan $e_k \sim e_{k-1}^{\,p}$; dengan substitusi, $e_k^{\,p} \sim e_k \cdot e_k^{\,1/p}$, sehingga $p = 1 + 1/p$, yaitu $p^2 - p - 1 = 0$, yang penyelesaian positifnya adalah rasio emas:

$$p = \frac{1 + \sqrt{5}}{2} \approx 1.618 \tag{3.7}$$

Karena itu metode secant bersifat *superlinear*: per langkah lebih lambat daripada Newton, tetapi setiap langkah hanya memerlukan satu nilai fungsi baru, sedangkan langkah Newton memerlukan satu nilai dan satu turunan. Bila turunan sama mahalnya dengan fungsi, dua langkah secant (orde $1.618^2 = 2.618$) berbiaya sama dengan satu langkah Newton (orde 2), dan metode secant menang. Metode ini memerlukan dua titik awal; pilihan yang lazim adalah tebakan x_0 dan sebuah titik sedikit menjauh darinya, misalnya $1.01\,x_0$.

3.5 Iterasi Titik Tetap

3.5.1 Menulis ulang $f(x) = 0$ sebagai $x = g(x)$

Cara ketiga memandang pencarian akar adalah menulis ulang persamaan $f(x) = 0$ sebagai $x = g(x)$ untuk suatu fungsi g, lalu cukup mengulang:

$$x_{k+1} = g(x_k) \tag{3.8}$$

Bilangan r dengan $g(r) = r$ adalah **titik tetap** g: sekali iterasi sampai di sana, ia tinggal di sana. Metode Newton sendiri adalah iterasi titik tetap dengan $g(x) = x - f(x)/f'(x)$, dan banyak model fisika dan ekonomi secara alami ditulis dalam bentuk ini ("nilai tahun depan adalah fungsi dari nilai tahun ini"). Namun persamaan apa pun dapat ditulis ulang dengan banyak cara, dan cara-cara itu berperilaku sangat berbeda. Ambil $x^2 = 2$. Membagi dengan x memberikan $x = 2/x$; dari $x_0 = 1$ ini menghasilkan 2,1,2,1, … selamanya. Menambahkan x ke kedua ruas $x^2 - 2 = 0$ memberikan $x = x^2 + x - 2$, yang lari ke tak hingga. Tetapi menulis $x^2 - 1 = 1$ sebagai $(x - 1)(x + 1) = 1$ memberikan $x = 1 + \frac{1}{1+x}$, dan dari $x_0 = 1$ ini menghasilkan 1.5,1.4,1.41667,1.41379,1.41429, …, makin mendekati $\sqrt{2} = 1.41421$. Mengapa?

3.5.2 Kapan dan seberapa cepat ia konvergen

Kurangkan $r = g(r)$ dari $x_{k+1} = g(x_k)$ dan pakai teorema nilai rata-rata, yang mengatakan bahwa $g(x_k) - g(r) = g'(\xi_k)(x_k - r)$ untuk suatu ξ_k di antara x_k dan r:

$$e_{k+1} = g'(\xi_k)\, e_k$$

Jadi setiap langkah mengalikan galat dengan kira-kira $g'(r)$. Jika $|g'(x)| \le K < 1$ di dekat akar, galat menyusut sedikitnya secepat K^k, dan iterasinya konvergen linear dengan laju $|g'(r)|$; jika $|g'(r)| > 1$, galatnya tumbuh dan iterasinya divergen. Inilah inti **teorema titik tetap** [3]:

> **DEFINISI.** *Teorema titik tetap.* Misalkan g kontinu pada $[a, b]$ dengan $g(x) \in [a, b]$ untuk setiap x di $[a, b]$, dan misalkan g' ada pada (a, b) dengan $|g'(x)| \le K < 1$ di sana. Maka g mempunyai tepat satu titik tetap r di $[a, b]$, dan untuk setiap x_0 di $[a, b]$ iterasi Persamaan (3.8) konvergen ke r dengan
>
> $$|x_k - r| \le \frac{K^k}{1 - K}\, |x_1 - x_0|$$
>
> (3.9)

Untuk $g(x) = 1 + 1/(1 + x)$, $g'(x) = -1/(1 + x)^2$, dan di akar $g'(\sqrt{2}) = -1/(1 + \sqrt{2})^2 = -0.1716$. Setiap langkah mengalikan galat dengan -0.17: tandanya berselang-seling (iterasinya melompat dari satu sisi akar ke sisi lain) dan galatnya menyusut hampir 6 kali per langkah, sekitar 0,77 digit desimal. Untuk $g(x) = 2/x$, $g'(\sqrt{2}) = -2/2 = -1$, tepat di perbatasan: galatnya tidak menyusut dan tidak tumbuh, itulah 2,1,2,1 yang tak berujung. Untuk $g(x) = x^2 + x - 2$, $g'(\sqrt{2}) = 2\sqrt{2} + 1 = 3.83 > 1$: divergen.

Teorema ini berguna bahkan sebelum perhitungan apa pun dilakukan, karena ia meramalkan banyaknya langkah. Untuk $g(x) = 1 + 1/(1 + x)$ pada $[1,2]$: g menurun, $g(1) = 1.5$ dan $g(2) = 1.3333$, jadi g memetakan $[1,2]$ ke $[1.3333,1.5]$, jauh di dalam $[1,2]$. Pada $[1,2]$, $|g'(x)| = 1/(1 + x)^2 \le 1/4$, jadi $K = 0.25$. Dari $x_0 = 1$, $x_1 = 1.5$ dan $|x_1 - x_0| = 0.5$. Menurut

Persamaan (3.9), galat setelah k langkah paling besar $0.25^k \times 0.5/0.75$, yang berada di bawah 10^{-6} begitu $0.25^k < 1.5 \times 10^{-6}$, yaitu $k \geq \log(1.5 \times 10^{-6})/\log 0.25 = 9.7$: sepuluh langkah dijamin cukup. Batas ini pesimistis, karena memakai kemiringan terburuk di seluruh selang (0.25) alih-alih kemiringan di akar (0.17); iterasi yang sebenarnya mencapai 10^{-6} setelah 8 langkah, seperti akan diperlihatkan Subbab 3.8.

Analisis yang sama menjelaskan kecepatan Newton. Untuk $g(x) = x - f(x)/f'(x)$, aturan hasil bagi memberikan $g'(x) = f(x)f''(x)/f'(x)^2$, yang bernilai *nol* pada akar sederhana, tempat $f(r) = 0$. Iterasi titik tetap yang g'-nya lenyap di titik tetap konvergen lebih cepat daripada linear: suku galat $g'(\xi_k)e_k$ menjadi $\frac{1}{2}g''(\xi_k)e_k^2$, yang tidak lain adalah Persamaan (3.4).

3.6 Membangunnya dalam Java dan Python

Ketiga metode masuk ke satu modul, **Kode 3.1** dan **Kode 3.2**, bersebelahan dengan metode-metode pengurung Bab 2. Bila `trace` menyala, setiap iterasi mencetak iterasi baru, nilai fungsi di sana, dan perubahannya dari iterasi sebelumnya.

Kode 3.1 `roots2.py`: Newton-Raphson, secant, dan iterasi titik tetap.

```
"""Open root finders: Newton-Raphson, secant and fixed-point iteration."""

HEADER = f"{'k':>3} {'x':>18} {'f(x)':>14} {'|x - x_old|':>12}"


def _row(k, x, fx, dx):
    print(f"{k:>3} {x:>18.12f} {fx:>14.6e} {dx:>12.3e}")


def newton(f, df, x0, tol=1e-10, max_iter=50, trace=False):
    """x_new = x - f(x) / f'(x) until two iterates differ by less than tol.
    Returns (root, iterations); raises ArithmeticError if f'(x) is 0."""
    x = x0
    for k in range(1, max_iter + 1):
        d = df(x)
        if d == 0:
            raise ArithmeticError(f"f'(x) = 0 at x = {x}")
        x_new = x - f(x) / d
        if trace:
            _row(k, x_new, f(x_new), abs(x_new - x))
        if abs(x_new - x) < tol:
            return x_new, k
        x = x_new
    raise ArithmeticError(f"no convergence in {max_iter} iterations")


def secant(f, x0, x1, tol=1e-10, max_iter=50, trace=False):
    """Newton with the derivative replaced by the slope through the last
    two iterates."""
    f0, f1 = f(x0), f(x1)
    for k in range(1, max_iter + 1):
        if f1 == f0:
            raise ArithmeticError("flat secant: f(x0) = f(x1)")
        x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
        if trace:
            _row(k, x2, f(x2), abs(x2 - x1))
        if abs(x2 - x1) < tol:
            return x2, k
        x0, f0 = x1, f1
        x1, f1 = x2, f(x2)
    raise ArithmeticError(f"no convergence in {max_iter} iterations")
```

Kode 3.1 (Lanjutan) *roots2.py: Newton-Raphson, secant, dan iterasi titik tetap.*

```
def fixed_point(g, x0, tol=1e-10, max_iter=200, trace=False):
    """x_new = g(x) until two iterates differ by less than tol."""
    x = x0
    for k in range(1, max_iter + 1):
        x_new = g(x)
        if trace:
            _row(k, x_new, x_new - g(x_new), abs(x_new - x))
        if abs(x_new - x) < tol:
            return x_new, k
        x = x_new
    raise ArithmeticError(f"no convergence in {max_iter} iterations")
```

Kode 3.2 `Roots2.java`: ketiga metode yang sama dalam Java.

```java
import java.util.function.DoubleUnaryOperator;

/** Open root finders: Newton-Raphson, secant and fixed-point iteration. */
public class Roots2 {
    /** A root and the number of iterations that found it. */
    public record Result(double root, int iterations) { }

    public static final String HEADER = String.format("%3s %18s %14s %12s",
            "k", "x", "f(x)", "|x - x_old|");

    static void row(int k, double x, double fx, double dx) {
        System.out.printf("%3d %18.12f %14.6e %12.3e%n", k, x, fx, dx);
    }

    /** x_new = x - f(x) / f'(x) until two iterates differ by less than tol. */
    public static Result newton(DoubleUnaryOperator f, DoubleUnaryOperator df,
                                double x0, double tol, int maxIter,
                                boolean trace) {
        double x = x0;
        for (int k = 1; k <= maxIter; k++) {
            double d = df.applyAsDouble(x);
            if (d == 0) {
                throw new ArithmeticException("f'(x) = 0 at x = " + x);
            }
            double xNew = x - f.applyAsDouble(x) / d;
            if (trace) {
                row(k, xNew, f.applyAsDouble(xNew), Math.abs(xNew - x));
            }
            if (Math.abs(xNew - x) < tol) {
                return new Result(xNew, k);
            }
            x = xNew;
        }
        throw new ArithmeticException("no convergence in " + maxIter
                                      + " iterations");
    }
```

Kode 3.2 (Lanjutan) *Roots2.java: ketiga metode yang sama dalam Java.*

```
    /** Newton with the derivative replaced by the slope of the secant. */
    public static Result secant(DoubleUnaryOperator f, double x0, double x1,
                                double tol, int maxIter, boolean trace) {
        double f0 = f.applyAsDouble(x0), f1 = f.applyAsDouble(x1);
        for (int k = 1; k <= maxIter; k++) {
            if (f1 == f0) {
                throw new ArithmeticException("flat secant: f(x0) = f(x1)");
            }
            double x2 = x1 - f1 * (x1 - x0) / (f1 - f0);
            if (trace) {
                row(k, x2, f.applyAsDouble(x2), Math.abs(x2 - x1));
            }
            if (Math.abs(x2 - x1) < tol) {
                return new Result(x2, k);
            }
            x0 = x1;
            f0 = f1;
            x1 = x2;
            f1 = f.applyAsDouble(x2);
        }
        throw new ArithmeticException("no convergence in " + maxIter
                                      + " iterations");
    }

    /** x_new = g(x) until two iterates differ by less than tol. */
    public static Result fixedPoint(DoubleUnaryOperator g, double x0,
                                    double tol, int maxIter, boolean trace) {
        double x = x0;
        for (int k = 1; k <= maxIter; k++) {
            double xNew = g.applyAsDouble(x);
            if (trace) {
                row(k, xNew, xNew - g.applyAsDouble(xNew), Math.abs(xNew - x));
            }
            if (Math.abs(xNew - x) < tol) {
                return new Result(xNew, k);
            }
            x = xNew;
        }
        throw new ArithmeticException("no convergence in " + maxIter
                                      + " iterations");
    }
}
```

Kegagalan-kegagalan di Subbab 3.3.3 ditangani dengan eksepsi alih-alih mengembalikan bilangan yang meragukan: `ArithmeticError` di Python (induk `ZeroDivisionError`) dan `ArithmeticException` di Java, dengan pesan yang menyebutkan apakah garis singgungnya datar, secant-nya datar, atau iterasinya kehabisan langkah. Jejak titik tetap mencetak $x_{\text{new}} - g(x_{\text{new}})$ di kolom $f(x)$, yaitu sisa alami persamaan $x = g(x)$.

Program tongkat ukur, **Kode 3.3** dan **Kode 3.4**, mendefinisikan volume Persamaan (3.1) dan turunannya Persamaan (3.2), mencetak iterasi Newton untuk 3.000 liter dari tebakan kotak, lalu menghitung kedelapan tanda dari 1.000 sampai 8.000 liter dengan Newton dan secant, sambil menghitung iterasi masing-masing.

Kode 3.3 `dipstick.py`: tanda-tanda kedalaman pada tangki silinder yang berbaring dengan Newton-Raphson dan secant.

```python
"""Marks for the dipstick of a water truck: a lying cylindrical tank.
For every volume V we solve V(h) = V for the depth h with Newton-Raphson."""
import math
from roots2 import HEADER, newton, secant

R = 0.90                    # radius of the tank, metres
L = 3.20                    # length of the tank, metres
FULL = math.pi * R * R * L  # cubic metres


def volume(h):
    """Water volume (m^3) at depth h: the circular segment times the length."""
    return L * (R * R * math.acos((R - h) / R)
                - (R - h) * math.sqrt(2 * R * h - h * h))


def dvolume(h):
    """dV/dh: the width of the water surface times the length."""
    return 2 * L * math.sqrt(2 * R * h - h * h)


def depth(litres, trace=False):
    target = litres / 1000.0
    guess = 2 * R * target / FULL          # as if the tank were a box
    return newton(lambda h: volume(h) - target, dvolume, guess, tol=1e-9,
                  trace=trace)


def main():
    print(f"Tank: radius {R} m, length {L} m, full {FULL * 1000:.1f} L")
    print("\nNewton-Raphson for 3000 L, starting from the box guess")
    print(HEADER)
    depth(3000, trace=True)
    print(f"\n{'volume':>7} {'depth':>9} {'Newton':>7} {'secant':>7}")
    for litres in range(1000, 8001, 1000):
        h, k1 = depth(litres)
        target = litres / 1000.0
        g = 2 * R * target / FULL
        _, k2 = secant(lambda d: volume(d) - target, g, g * 1.01, tol=1e-9)
        print(f"{litres:>5} L {h * 100:>6.2f} cm {k1:>7} {k2:>7}")


if __name__ == "__main__":
    main()
```

Kode 3.4 `Dipstick.java`: program yang sama dalam Java.

```java
import java.util.Locale;

/**
 * Marks for the dipstick of a water truck: a lying cylindrical tank. For
 * every volume V we solve V(h) = V for the depth h with Newton-Raphson.
 */
public class Dipstick {
    static final double R = 0.90;                // radius of the tank, metres
    static final double L = 3.20;                // length of the tank, metres
    static final double FULL = Math.PI * R * R * L;   // cubic metres

    /** Water volume (m^3) at depth h: the circular segment times the length. */
    static double volume(double h) {
        return L * (R * R * Math.acos((R - h) / R)
                    - (R - h) * Math.sqrt(2 * R * h - h * h));
    }

    /** dV/dh: the width of the water surface times the length. */
    static double dvolume(double h) {
        return 2 * L * Math.sqrt(2 * R * h - h * h);
    }

    static Roots2.Result depth(double litres, boolean trace) {
        double target = litres / 1000.0;
        double guess = 2 * R * target / FULL;     // as if the tank were a box
        return Roots2.newton(h -> volume(h) - target, Dipstick::dvolume, guess,
                             1e-9, 50, trace);
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.printf("Tank: radius %s m, length %s m, full %.1f L%n",
                          R, L, FULL * 1000);
        System.out.println("\nNewton-Raphson for 3000 L, starting from the box "
                           + "guess");
        System.out.println(Roots2.HEADER);
        depth(3000, true);
        System.out.printf("%n%7s %9s %7s %7s%n", "volume", "depth", "Newton",
                          "secant");
        for (int litres = 1000; litres <= 8000; litres += 1000) {
            Roots2.Result r = depth(litres, false);
            double target = litres / 1000.0;
            double g = 2 * R * target / FULL;
            Roots2.Result s = Roots2.secant(d -> volume(d) - target, g,
                                            g * 1.01, 1e-9, 50, false);
            System.out.printf("%5d L %6.2f cm %7d %7d%n", litres,
```

Kode 3.4 (Lanjutan) *Dipstick.java: program yang sama dalam Java.*

```java
                              r.root() * 100, r.iterations(), s.iterations());
        }
    }
}
```

Fungsi `depth` membangun $f(h) = V(h) - V_{\text{target}}$ sebagai fungsi anonim kecil (sebuah `lambda` di Python, ekspresi lambda di Java) dan menyerahkannya, bersama turunannya, kepada `newton`. Turunan f adalah turunan V, karena mengurangkan konstanta tidak mengubah kemiringan. Metode secant dimulai dari tebakan kotak dan sebuah titik 1% di atasnya. Toleransi 10^{-9} m adalah sepersejuta milimeter, jauh melampaui keperluan sebatang tongkat kayu, tetapi hampir tidak ada biayanya, seperti akan kita lihat.

3.7 Menjalankannya

```
$ python3 dipstick.py          (or: java -cp out Dipstick)
```

```
Tank: radius 0.9 m, length 3.2 m, full 8143.0 L

Newton-Raphson for 3000 L, starting from the box guess
  k                  x           f(x)  |x - x_old|
  1     0.712968084725   2.005405e-03    4.982e-02
  2     0.712612153675   8.618449e-08    3.559e-04
  3     0.712612138378   0.000000e+00    1.530e-08
  4     0.712612138378   0.000000e+00    0.000e+00

 volume     depth  Newton  secant
 1000 L  32.47 cm       5       6
 2000 L  52.96 cm       4       5
 3000 L  71.26 cm       4       4
 4000 L  88.76 cm       3       3
 5000 L 106.21 cm       4       4
 6000 L 124.33 cm       4       5
 7000 L 144.36 cm       4       6
 8000 L 171.37 cm       5       7
```

Jejak untuk 3.000 liter memperlihatkan konvergensi kuadratik yang sedang bekerja. Langkah pertama bergeser 4,98 cm dari tebakan kotak ke 71,2968 cm, persis seperti hitungan tangan kita di Subbab 3.2.2. Langkah kedua mengoreksi 0,36 mm, langkah ketiga 0,0000153 mm, dan langkah keempat tidak mengoreksi apa-apa: iterasinya tidak lagi berubah dalam presisi ganda, dan f tepat nol di sana. Perubahan 4.98×10^{-2}, 3.56×10^{-4}, 1.53×10^{-8} memperlihatkan digit yang berlipat dua: 1, 3, 8 digit benar, lalu seluruh 16.

Tabelnya adalah tongkat ukur Pak Harun. Tanda-tandanya jelas tidak berjarak sama: 1.000 liter pertama mengisi 32,47 cm, 1.000 berikutnya hanya 20,49 cm (sampai 52,96 cm), 1.000 liter di tengah (dari 4.000 ke 5.000) hanya 17,45 cm, dan 1.000 liter terakhir sebelum 8.000 liter kembali 27,01 cm. Newton memerlukan 3 sampai 5 iterasi untuk setiap tanda, metode secant 3 sampai 7. Keduanya memerlukan langkah terbanyak di dekat dasar dan di dekat puncak, tempat tebakan kotak paling buruk dan kurvanya paling melengkung, dan paling sedikit di dekat 4.000 liter, tempat tangki hampir setengah penuh dan tebakan kotak hampir tepat. Biseksi dengan kurung $[0,1.8]$ dan toleransi yang sama akan memerlukan $\log_2(1.8/10^{-9}) = 30.7$, jadi 31 iterasi untuk setiap tanda.

Dalam praktik Pak Harun memerlukan tanda-tanda itu sampai milimeter: 32,5; 53,0; 71,3; 88,8; 106,2; 124,3; 144,4; dan 171,4 cm, diukur dari dasar tangki. Keponakannya mencetak tabel itu, menempelkannya pada tongkat baru, dan menoreh takik di setiap tanda. Pertama kali mereka memeriksanya, dengan mengisi bak 1.000 liter yang volumenya sudah diketahui, permukaan air pada tongkat turun tepat satu tanda.

3.8 Mengukur Orde Konvergensi

Definisi Persamaan (3.5) dapat diubah menjadi pengukuran. Jika $|e_{k+1}| \approx |C| \cdot |e_k|^p$ dan $|e_k| \approx |C| \cdot |e_{k-1}|^p$, membagi keduanya dan mengambil logaritma melenyapkan C:

$$p \approx \frac{\ln(|e_{k+1}|/|e_k|)}{\ln(|e_k|/|e_{k-1}|)} \tag{3.10}$$

Kode 3.5 dan **Kode 3.6** menghitung $\sqrt{2}$ dengan tiga metode (Newton pada $x^2 - 2$, secant pada $x^2 - 2$, dan iterasi titik tetap $x = 1 + 1/(1 + x)$), mencetak galat setiap iterasi terhadap $\sqrt{2}$ dari pustaka, rasio $|e_k|/|e_{k-1}|$, dan taksiran Persamaan (3.10).

Kode 3.5 `convergence.py`: $\sqrt{2}$ dengan tiga metode, beserta rasio dan orde konvergensi yang terukur.

```python
"""How fast the error falls: sqrt(2) by three methods, with the order p
estimated from three consecutive errors."""
import math

ROOT = math.sqrt(2.0)


def report(name, xs):
    print(name)
    print(f"{'k':>3} {'x_k':>18} {'error':>11} {'ratio':>8} {'order p':>8}")
    errs = [abs(x - ROOT) for x in xs]
    for k, (x, e) in enumerate(zip(xs, errs), 1):
        ratio, p = "", ""
        if k >= 2 and e > 0 and errs[k - 2] > 0:
            ratio = f"{e / errs[k - 2]:.4f}"          # e_k / e_(k-1)
        if k >= 3 and e > 0 and errs[k - 2] > 0 and errs[k - 3] > 0:
            top = math.log(e / errs[k - 2])
            p = f"{top / math.log(errs[k - 2] / errs[k - 3]):.3f}"
        print(f"{k:>3} {x:>18.14f} {e:>11.3e} {ratio:>8} {p:>8}")
    print()


def main():
    xs = []
    x = 1.0
    for _ in range(5):                    # Newton for x^2 - 2
        x = x - (x * x - 2) / (2 * x)
        xs.append(x)
    report("Newton-Raphson, x0 = 1", xs)
    xs, a, b = [], 1.0, 2.0
    for _ in range(7):                    # secant for x^2 - 2
        c = b - (b * b - 2) * (b - a) / ((b * b - 2) - (a * a - 2))
        xs.append(c)
        a, b = b, c
        if a == b:
            break
    report("Secant, x0 = 1, x1 = 2", xs)
    xs, x = [], 1.0
    for _ in range(8):                    # fixed point x = 1 + 1 / (1 + x)
        x = 1 + 1 / (1 + x)
        xs.append(x)
    report("Fixed point x = 1 + 1/(1 + x), x0 = 1", xs)


if __name__ == "__main__":
    main()
```

Kode 3.6 `Convergence.java`: pengukuran yang sama dalam Java.

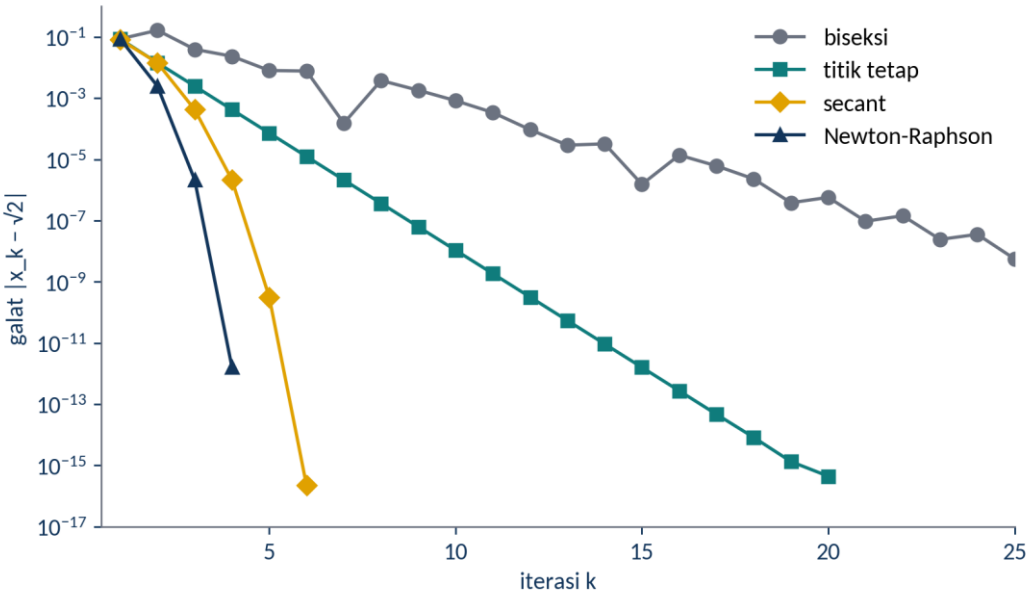
```java
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

/**
 * How fast the error falls: sqrt(2) by three methods, with the order p
 * estimated from three consecutive errors.
 */
public class Convergence {
    static final double ROOT = Math.sqrt(2.0);

    static void report(String name, List<Double> xs) {
        System.out.println(name);
        System.out.printf("%3s %18s %11s %8s %8s%n", "k", "x_k", "error",
                          "ratio", "order p");
        double[] e = new double[xs.size()];
        for (int i = 0; i < e.length; i++) {
            e[i] = Math.abs(xs.get(i) - ROOT);
        }
        for (int i = 0; i < e.length; i++) {
            String ratio = "", p = "";
            if (i >= 1 && e[i] > 0 && e[i - 1] > 0) {
                ratio = String.format("%.4f", e[i] / e[i - 1]);  // e_k / e_(k-1)
            }
            if (i >= 2 && e[i] > 0 && e[i - 1] > 0 && e[i - 2] > 0) {
                double top = Math.log(e[i] / e[i - 1]);
                p = String.format("%.3f", top / Math.log(e[i - 1] / e[i - 2]));
            }
            System.out.printf("%3d %18.14f %11.3e %8s %8s%n", i + 1, xs.get(i),
                              e[i], ratio, p);
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        List<Double> xs = new ArrayList<>();
        double x = 1.0;
        for (int i = 0; i < 5; i++) {            // Newton for x^2 - 2
            x = x - (x * x - 2) / (2 * x);
            xs.add(x);
        }
        report("Newton-Raphson, x0 = 1", xs);
        xs = new ArrayList<>();
        double a = 1.0, b = 2.0;
        for (int i = 0; i < 7; i++) {            // secant for x^2 - 2
```

Kode 3.6 (Lanjutan) *Convergence.java: pengukuran yang sama dalam Java.*

```
            double c = b - (b * b - 2) * (b - a) / ((b * b - 2) - (a * a - 2));
            xs.add(c);
            a = b;
            b = c;
            if (a == b) {
                break;
            }
        }
        report("Secant, x0 = 1, x1 = 2", xs);
        xs = new ArrayList<>();
        x = 1.0;
        for (int i = 0; i < 8; i++) {            // fixed point x = 1 + 1/(1 + x)
            x = 1 + 1 / (1 + x);
            xs.add(x);
        }
        report("Fixed point x = 1 + 1/(1 + x), x0 = 1", xs);
    }
}
```

```
Newton-Raphson, x0 = 1
  k                x_k        error     ratio  order p
  1   1.50000000000000   8.579e-02
  2   1.41666666666667   2.453e-03   0.0286
  3   1.41421568627451   2.124e-06   0.0009    1.984
  4   1.41421356237469   1.595e-12   0.0000    2.000
  5   1.41421356237310   0.000e+00

Secant, x0 = 1, x1 = 2
  k                x_k        error     ratio  order p
  1   1.33333333333333   8.088e-02
  2   1.40000000000000   1.421e-02   0.1757
  3   1.41463414634146   4.206e-04   0.0296    2.025
  4   1.41421143847487   2.124e-06   0.0050    1.502
  5   1.41421356205732   3.158e-10   0.0001    1.667
  6   1.41421356237310   2.220e-16   0.0000    1.607
  7   1.41421356237310   0.000e+00

Fixed point x = 1 + 1/(1 + x), x0 = 1
  k                x_k        error     ratio  order p
  1   1.50000000000000   8.579e-02
  2   1.40000000000000   1.421e-02   0.1657
  3   1.41666666666667   2.453e-03   0.1726    0.977
  4   1.41379310344828   4.205e-04   0.1714    1.004
  5   1.41428571428571   7.215e-05   0.1716    0.999
  6   1.41420118343195   1.238e-05   0.1716    1.000
  7   1.41421568627451   2.124e-06   0.1716    1.000
  8   1.41421319796954   3.644e-07   0.1716    1.000
```

Orde-orde yang terukur cocok dengan teori. Newton memberikan 1,984 dan 2,000: kuadratik, Persamaan (3.4). Kolom rasionya turun menuju nol, ciri setiap metode yang lebih cepat daripada linear. Metode secant memberikan 2,025; 1,502; 1,667; dan 1,607: berisik di awal, lalu mengendap menuju rasio emas 1,618 dari Persamaan (3.7). Iterasi titik tetap memberikan orde tepat 1,000 dan rasio yang mengendap di 0,1716, persis $|g'(\sqrt{2})|$ dari Subbab 3.5.2. Galatnya turun dengan faktor yang sama di setiap langkah, selamanya, sedangkan galat Newton runtuh: Newton mencapai presisi penuh dalam 5 langkah, metode secant dalam 6, dan iterasi titik tetap akan memerlukan sekitar 20. **Gambar 3.3** memperlihatkan ketiganya bersama biseksi pada skala logaritmik.



Gambar 3.3 Galat empat metode yang menghitung $\sqrt{2}$: biseksi dan iterasi titik tetap turun sepanjang garis lurus (konvergensi linear), kurva secant dan Newton melengkung ke bawah makin curam (orde 1,618 dan 2). Sebuah garis berhenti ketika iterasinya sama dengan *double* terdekat dengan $\sqrt{2}$.

Langkah-langkah Newton pertama juga contoh yang bagus dari metode Heron: $\frac{1}{2}(1 + 2/1) = 1.5$, lalu $\frac{1}{2}(1.5 + 2/1.5) = 1.41667$, lalu 1.414215686, nilai-nilai di kolom pertama. Sebuah lempeng tanah liat dari Babilonia kuno, YBC 7289, memperlihatkan $\sqrt{2}$ sampai sekitar enam angka bena, kira-kira seteliti langkah ketiga.

3.9 Ketika Newton Tersesat

Kode 3.7 dan **Kode 3.8** menjalankan kegagalan-kegagalan di Subbab 3.3.3 dan, sebagai pembanding, titik awal yang baik untuk kubik yang sama.

Kode 3.7 `failures.py`: divergensi, siklus, garis singgung datar, dan titik awal yang baik.

```python
"""Three ways Newton-Raphson goes wrong, and the secant method on the same."""
import math
from roots2 import newton, secant


def cube_root(x):
    return math.copysign(abs(x) ** (1.0 / 3.0), x)


def main():
    print("1. f(x) = cube root of x, x0 = 0.1: each step doubles the distance")
    x = 0.1
    for k in range(1, 6):
        x = x - cube_root(x) / (cube_root(x) / (3 * x))   # f / f'
        print(f"   x{k} = {x:.6f}")

    print("2. f(x) = x^3 - 2x + 2, x0 = 0: a cycle 0, 1, 0, 1, ...")
    x = 0.0
    for k in range(1, 7):
        x = x - (x * x * x - 2 * x + 2) / (3 * x * x - 2)
        print(f"   x{k} = {x:.6f}")

    print("3. f(x) = x^2 - 4x + 5 has no real root; f'(2) = 0")
    try:
        newton(lambda t: t * t - 4 * t + 5, lambda t: 2 * t - 4, 2.0)
    except ArithmeticError as err:
        print(f"   stopped: {err}")

    print("4. Newton from a good start, x0 = -2 for x^3 - 2x + 2")
    root, k = newton(lambda t: t * t * t - 2 * t + 2,
                     lambda t: 3 * t * t - 2, -2.0)
    print(f"   root {root:.12f} after {k} iterations")
    root, k = secant(lambda t: t * t * t - 2 * t + 2, -2.0, -1.9)
    print(f"   secant: root {root:.12f} after {k} iterations")


if __name__ == "__main__":
    main()
```

Kode 3.8 `Failures.java`: keempat kasus yang sama dalam Java.

```
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Three ways Newton-Raphson goes wrong, and the secant method on the same. */
public class Failures {
    static double cubeRoot(double x) {
        return Math.copySign(Math.pow(Math.abs(x), 1.0 / 3.0), x);
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.println("1. f(x) = cube root of x, x0 = 0.1: each step "
                           + "doubles the distance");
        double x = 0.1;
        for (int k = 1; k <= 5; k++) {
            x = x - cubeRoot(x) / (cubeRoot(x) / (3 * x));   // f / f'
            System.out.printf("   x%d = %.6f%n", k, x);
        }

        System.out.println("2. f(x) = x^3 - 2x + 2, x0 = 0: a cycle 0, 1, 0, "
                           + "1, ...");
        x = 0.0;
        for (int k = 1; k <= 6; k++) {
            x = x - (x * x * x - 2 * x + 2) / (3 * x * x - 2);
            System.out.printf("   x%d = %.6f%n", k, x);
        }

        System.out.println("3. f(x) = x^2 - 4x + 5 has no real root; "
                           + "f'(2) = 0");
        try {
            Roots2.newton(t -> t * t - 4 * t + 5, t -> 2 * t - 4, 2.0, 1e-10,
                          50, false);
        } catch (ArithmeticException err) {
            System.out.println("   stopped: " + err.getMessage());
        }
```

Kode 3.8 (Lanjutan) *Failures.java: keempat kasus yang sama dalam Java.*

```
        System.out.println("4. Newton from a good start, x0 = -2 for "
                           + "x^3 - 2x + 2");
        DoubleUnaryOperator f = t -> t * t * t - 2 * t + 2;
        Roots2.Result r = Roots2.newton(f, t -> 3 * t * t - 2, -2.0, 1e-10, 50,
                                        false);
        System.out.printf("   root %.12f after %d iterations%n", r.root(),
                          r.iterations());
        r = Roots2.secant(f, -2.0, -1.9, 1e-10, 50, false);
        System.out.printf("   secant: root %.12f after %d iterations%n",
                          r.root(), r.iterations());
    }
}
```

```
1. f(x) = cube root of x, x0 = 0.1: each step doubles the distance
   x1 = -0.200000
   x2 = 0.400000
   x3 = -0.800000
   x4 = 1.600000
   x5 = -3.200000
2. f(x) = x^3 - 2x + 2, x0 = 0: a cycle 0, 1, 0, 1, ...
   x1 = 1.000000
   x2 = 0.000000
   x3 = 1.000000
   x4 = 0.000000
   x5 = 1.000000
   x6 = 0.000000
3. f(x) = x^2 - 4x + 5 has no real root; f'(2) = 0
   stopped: f'(x) = 0 at x = 2.0
4. Newton from a good start, x0 = -2 for x^3 - 2x + 2
   root -1.769292354239 after 5 iterations
   secant: root -1.769292354239 after 6 iterations
```

Akar pangkat tiga melipatduakan jaraknya di setiap langkah, $0.1, -0.2, 0.4, -0.8, 1.6, -3.2$, persis seperti ramalan aljabarnya. Kubik dari $x_0 = 0$ memantul antara 0 dan 1 tanpa akhir; program tanpa batas banyaknya iterasi tidak akan pernah berhenti. Fungsi $x^2 - 4x + 5 = (x-2)^2 + 1$ tidak pernah nol, dan di titik terendahnya $x = 2$ garis singgungnya datar, sehingga `newton` langsung menolak. Tetapi dari $x_0 = -2$, kubik yang sama yang berputar-putar dari 0 konvergen dalam 5 langkah, dan metode secant dalam 6: perbedaannya sepenuhnya terletak pada titik awal.

3.10 Mengujinya

Kode 3.9 dan **Kode 3.10** memeriksa ketiga metode pada $\sqrt{2}$, memeriksa bahwa garis singgung datar ditolak, dan memeriksa satu sifat tongkat ukur yang dapat kita pastikan tanpa komputasi apa pun: ketika tangki berisi tepat separuh volumenya, kedalamannya harus tepat sama dengan jari-jari, 0,90 m.

Kode 3.9 `test_roots2.py`: uji unit metode-metode terbuka dan tongkat ukur.

```python
"""Tests of the open root finders: python -m unittest test_roots2.py"""
import math
import unittest
from roots2 import fixed_point, newton, secant


class Roots2Test(unittest.TestCase):
    def test_newton_sqrt2(self):
        root, k = newton(lambda x: x * x - 2, lambda x: 2 * x, 1.0)
        self.assertAlmostEqual(root, math.sqrt(2), places=12)
        self.assertLessEqual(k, 6)

    def test_secant_sqrt2(self):
        root, _ = secant(lambda x: x * x - 2, 1.0, 2.0)
        self.assertAlmostEqual(root, math.sqrt(2), places=12)

    def test_fixed_point_sqrt2(self):
        root, _ = fixed_point(lambda x: 1 + 1 / (1 + x), 1.0)
        self.assertAlmostEqual(root, math.sqrt(2), places=9)

    def test_zero_derivative(self):
        with self.assertRaises(ArithmeticError):
            newton(lambda x: x * x + 1, lambda x: 2 * x, 0.0)

    def test_dipstick_full_and_half(self):
        from dipstick import FULL, depth
        h, _ = depth(FULL * 1000 / 2)      # half full: depth = radius
        self.assertAlmostEqual(h, 0.90, places=9)


if __name__ == "__main__":
    unittest.main()
```

Kode 3.10 `Roots2Test.java`: pemeriksaan yang sama dalam Java.

```java
/** Tests of the open root finders: java -cp out Roots2Test */
public class Roots2Test {
    static int failures = 0;

    static void check(boolean ok, String name) {
        System.out.println((ok ? "ok    " : "FAILED") + "  " + name);
        if (!ok) {
            failures++;
        }
    }

    public static void main(String[] args) {
        double s2 = Math.sqrt(2);
        Roots2.Result n = Roots2.newton(x -> x * x - 2, x -> 2 * x, 1.0, 1e-10,
                                        50, false);
        check(Math.abs(n.root() - s2) < 1e-12 && n.iterations() <= 6,
              "Newton finds sqrt 2 in at most 6 steps");
        Roots2.Result s = Roots2.secant(x -> x * x - 2, 1.0, 2.0, 1e-10, 50,
                                        false);
        check(Math.abs(s.root() - s2) < 1e-12, "secant finds sqrt 2");
        Roots2.Result p = Roots2.fixedPoint(x -> 1 + 1 / (1 + x), 1.0, 1e-10,
                                            200, false);
        check(Math.abs(p.root() - s2) < 1e-9, "fixed point finds sqrt 2");
        try {
            Roots2.newton(x -> x * x + 1, x -> 2 * x, 0.0, 1e-10, 50, false);
            check(false, "zero derivative is refused");
        } catch (ArithmeticException e) {
            check(true, "zero derivative is refused");
        }
        double h = Dipstick.depth(Dipstick.FULL * 1000 / 2, false).root();
        check(Math.abs(h - 0.90) < 1e-9, "half-full tank: depth = radius");
        System.out.println(failures == 0 ? "all tests passed"
                                         : failures + " test(s) failed");
    }
}
```

```
$ python3 -m unittest test_roots2.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.000s

OK
$ java -cp out Roots2Test
```

```
ok      Newton finds sqrt 2 in at most 6 steps
ok      secant finds sqrt 2
ok      fixed point finds sqrt 2
ok      zero derivative is refused
ok      half-full tank: depth = radius
all tests passed
```

Uji setengah penuh adalah contoh baik uji yang tidak sekadar mengulang aritmetika program itu sendiri. Uji itu memeriksa program terhadap fakta yang independen, yaitu kesimetrisan lingkaran. Uji yang membandingkan keluaran dengan bilangan yang disalin dari jalannya program yang sama sebelumnya akan tetap lolos meskipun rumus volumenya salah. Fakta independen lain juga menjadi uji yang sama baiknya: tangki penuh harus memberikan $2R$, dan tanda untuk V serta tanda untuk volume penuh dikurangi V harus berjumlah $2R$, karena tangki itu simetris terhadap bagian tengahnya. Uji-uji semacam ini murah ditulis dan mahal bila dilewatkan.

3.11 Membandingkan dengan SciPy

Fungsi `newton` dari SciPy memakai metode Newton bila diberi turunannya (`fprime`) dan metode secant bila tidak; `brentq` adalah metode pengurung dari Bab 2. **Kode 3.11** menanyakan tanda 3.000 liter kepada ketiganya, dengan memakai kembali fungsi-fungsi volume dari `dipstick.py`.

Kode 3.11 `compare_scipy.py`: tanda 3.000 liter dengan `newton` (dengan dan tanpa turunan) dan `brentq` dari SciPy.

```
"""The dipstick depth for 3000 L with SciPy (needs SciPy)."""
from scipy.optimize import brentq, newton

from dipstick import FULL, R, dvolume, volume

TARGET = 3.0                              # cubic metres


def f(h):
    return volume(h) - TARGET


def main():
    guess = 2 * R * TARGET / FULL
    h1 = newton(f, guess, fprime=dvolume)            # Newton-Raphson
    h2 = newton(f, guess)                            # no fprime: secant
    h3 = brentq(f, 0.0, 2 * R)                       # bracketing, Brent
    print(f"newton with fprime : {h1:.12f} m")
    print(f"newton, secant     : {h2:.12f} m")
    print(f"brentq on [0, 2R]  : {h3:.12f} m")


if __name__ == "__main__":
    main()
```

```
newton with fprime : 0.712612138378 m
newton, secant     : 0.712612138378 m
brentq on [0, 2R]  : 0.712612138378 m
```

Ketiganya sepakat dengan 0,712612138378 m milik kita sampai dua belas desimal, sepersejuta mikrometer. Dokumentasi SciPy mencatat pertimbangan yang sama dengan yang diturunkan bab ini: Newton konvergen paling cepat di dekat akar sederhana tetapi memerlukan titik awal yang baik dan turunan, metode secant tidak memerlukan turunan, dan metode pengurung seperti `brentq` adalah pilihan aman bila kurungnya diketahui [4].

3.12 Memilih Metode

Bab 2 dan 3 telah membangun enam pencari akar. **Tabel 3.1** menyandingkan semuanya.

Tabel 3.1 Pencari akar Bab 2 dan 3 dibandingkan: apa yang diperlukan masing-masing, seberapa cepat konvergen di dekat akar sederhana, dan apakah dijamin berhasil.

Metode	Memerlukan	Orde	Evaluasi per langkah	Terjamin?
Biseksi	sebuah kurung	1 (laju 1/2)	1 nilai	ya, dengan banyak langkah yang diketahui
Regula falsi	sebuah kurung	1 (laju berubah-ubah)	1 nilai	ya, tetapi bisa merangkak
Illinois	sebuah kurung	sekitar 1,44	1 nilai	ya
Secant	dua titik awal	sekitar 1,618	1 nilai	tidak
Newton-Raphson	titik awal dan f'	2	1 nilai dan 1 turunan	tidak
Titik tetap	bentuk $x = g(x)$	1 (laju $\lvert g'(r)\rvert$)	1 nilai g	hanya bila $\lvert g'\rvert < 1$

Tabel itu menyarankan cara berpikir yang sederhana. Bila kegagalan tidak dapat diterima dan Anda mempunyai kurung, pakailah metode pengurung: Illinois atau, di Python, `brentq` dari SciPy, yang memadukan jaminan biseksi dengan kecepatan metode secant. Bila Anda memerlukan akar yang sejenis ribuan kali, mempunyai tebakan awal yang baik (seperti tebakan kotak untuk tangki), dan turunannya mudah, pakailah Newton: biasanya selesai dalam tiga atau empat langkah. Bila turunannya sulit, pakailah metode secant. Pakailah iterasi titik tetap bila masalahnya sudah datang dalam bentuk $x = g(x)$ dan Anda sudah memeriksa bahwa $\lvert g'\rvert$ kecil.

Dalam perangkat lunak profesional pilihannya sering kali bukan salah satu. Metode **Newton berpengaman** (*safeguarded Newton*) mempertahankan kurung $[a, b]$ seperti biseksi, mengambil langkah Newton dari titik saat ini, dan menerimanya hanya bila langkah itu mendarat di dalam kurung dan cukup memperkecil $\lvert f\rvert$; jika tidak, ia mengambil langkah biseksi. Setelah setiap langkah, kurung diperbarui menurut tanda f di titik yang baru. Metode seperti itu mewarisi jaminan biseksi (tidak mungkin kehilangan akar) dan kecepatan Newton di dekat akar (begitu langkah-langkah Newton diterima, langkah itu tidak pernah diganti). Metode Brent, yang dijumpai di Subbab 2.9, adalah gagasan yang sama dengan langkah secant dan interpolasi kuadratik invers sebagai pengganti langkah Newton [2].

Satu pertimbangan lagi menentukan banyak pilihan nyata: *biaya mengevaluasi* f. Untuk tangki Pak Harun, satu evaluasi adalah beberapa operasi aritmetika, dan keenam metode selesai dalam hitungan mikrodetik. Bila satu evaluasi f berarti menjalankan simulasi selama sejam, atau mengukur sesuatu di laboratorium, banyaknya evaluasi adalah satu-satunya biaya yang penting, dan metode berorde 1,618 dengan satu evaluasi per langkah dapat mengalahkan metode berorde 2 dengan dua evaluasi.

3.13 Metode Numerik dalam Praktik: Profitina

Banyak besaran dalam sebuah produk dinyatakan oleh kurva yang mulus dan naik, dan pertanyaan yang diajukan orang menjalankan kurva itu *mundur*. Koin PROFIT milik Profitina mempunyai *oracle* harga yang mengonversi antara PROFIT dan rupiah; langganan Pro berharga Rp50.000 untuk tiga puluh hari dan dapat dibayar dengan PROFIT. Mengonversi nominal rupiah menjadi koin pada kurs tetap adalah sebuah pembagian, tetapi begitu nominalnya bergantung pada harga secara tidak linear (biaya yang mengecil bersama nominal, tingkatan diskon, emisi yang meluruh yang menentukan banyaknya koin yang ada), "berapa koin yang saya perlukan untuk sekian rupiah?" menjadi kebalikan sebuah kurva, persis masalah Pak Harun: rumusnya berjalan dari kedalaman ke volume, pertanyaannya dari volume ke kedalaman.

Hal yang sama terjadi pada skor. AI Visibility Engine melaporkan skor pada skala 0 sampai 100, dan dasbor historis menampilkan tren mingguannya. Bila sebuah skor dihasilkan dari suatu pengukuran mentah melalui kurva normalisasi yang mulus, pertanyaan "nilai mentah berapa yang akan menaikkan skor menjadi 70?" membalik kurva itu. Untuk kurva yang mulus dan monoton dengan turunan yang diketahui, metode Newton menjawab pertanyaan seperti itu dalam segelintir langkah, dan gagasan tebakan kotak dari Subbab 3.2.2 (mulai dari jawaban yang diberikan hampiran garis lurus) persis cara sistem produksi memilih titik awalnya. Bila kurvanya hanya diketahui dari data, metode secant, yang tidak memerlukan turunan, adalah pilihan yang wajar.

3.14 Memakainya Sungguhan

Program tongkat ukur berlaku untuk silinder berbaring mana pun; hanya `R` dan `L` yang berubah. Beberapa hal praktis membedakan tanda yang benar dari tanda yang hanya tampak benar.

- **Ukur bagian dalam, bukan bagian luar.** Volume bergantung pada jari-jari dan panjang bagian dalam. Tangki baja berdinding 4 mm mempunyai jari-jari dalam 4 mm lebih kecil daripada jari-jari luarnya, yang pada $R = 0.9$ m mengubah volume penuh hampir 1%, sekitar 70 liter.
- **Perhitungkan kedua ujungnya.** Banyak tangki truk mempunyai ujung yang sedikit cembung, yang menampung air tambahan yang tidak termasuk dalam Persamaan (3.1). Bandingkan volume penuh hasil hitungan dengan volume yang dinyatakan pembuat tangki, atau lebih baik lagi, dengan pengisian yang diukur.
- **Kalibrasi sekali dengan volume yang diketahui.** Isi tangki dari meteran atau dengan sejumlah bak 1.000 liter yang diketahui, baca tongkatnya setelah setiap pengisian, lalu bandingkan dengan tabel. Jika bacaannya konsisten meleset dengan pecahan yang sama, sesuaikan L; jika melesetnya terutama di dekat dasar, periksa jari-jari dalamnya.
- **Bentuk lain.** Silinder tegak sama sekali tidak memerlukan pencarian akar ($V = \pi R^2 h$). Tangki bola memakai volume dari Soal Latihan 2.6 dengan turunan $V'(h) = \pi(2Rh - h^2)$, dan Newton lalu memerlukan sekitar 4 langkah, sedangkan biseksi memerlukan 21. Tangki elips, yang lazim pada truk bahan bakar, bekerja seperti silinder berbaring dengan lingkaran yang direntangkan: setiap volume dikalikan rasio lebar elips terhadap tingginya, dan kedalaman tanda-tandanya direntangkan menurut rasio tinggi.
- **Ketelitian yang sebenarnya diperlukan.** Turunan Persamaan (3.2) juga memberi tahu seberapa penting sebuah galat pembacaan. Membaca tongkat keliru 2 mm pada setengah kedalaman, tempat permukaan paling lebar, berarti $V' \times 0.002 = 2 \times 3.2 \times 0.9 \times 0.002 = 0.0115$ m^3, sekitar 12 liter; di dekat tanda 1.000 liter, tempat $V' = 4.43$ m^3 per meter, 2 mm yang sama berarti sekitar 9 liter. Air juga memuai kira-kira 0,2% antara pagi yang sejuk dan siang yang terik, sekitar 16 liter pada tangki penuh. Karena itu tanda sampai milimeter sudah lebih dari cukup teliti; mata rantai yang lemah bukan perhitungannya, melainkan pembacaannya. Hal ini patut diingat setiap kali sebuah program mencetak banyak digit: digit-digit itu hanya berguna sejauh dunia nyata dapat membacanya.

Bentuk program yang sama (rumus dari masukan ke keluaran, turunannya, tebakan pertama yang baik, dan Newton) menyelesaikan banyak pertanyaan kebalikan lain: diameter pipa yang mengalirkan debit tertentu, waktu ketika minuman yang mendingin mencapai suhu tertentu, suku bunga yang tersembunyi dalam rencana cicilan (Soal Latihan 3.5 mengerjakannya dengan metode secant, tanpa turunan).

3.15 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memulai di tempat turunannya nol.** Tebakan $h = 0$ untuk tangki membuat $V'(0) = 0$ dan langkah pertama membagi dengan nol. Mulailah dari taksiran yang masuk akal, seperti tebakan kotak, dan periksa `d == 0` di kode.
- **Tanpa batas banyaknya iterasi.** Siklus seperti 0,1,0,1 berjalan selamanya. Selalu batasi iterasinya dan munculkan galat, alih-alih mengembalikan nilai terakhir, bila batas itu tercapai.
- **Turunan yang salah.** Newton dengan f' yang salah biasanya tetap konvergen, tetapi hanya linear dan lambat, sehingga kesalahannya tersembunyi. Periksa turunannya terhadap selisih hingga $(f(x+h) - f(x))/h$ dengan $h \approx 10^{-8}$ (Subbab 1.12.2) di satu atau dua titik.
- **Memercayai satu kali jalan dari titik awal yang buruk.** Metode terbuka bisa konvergen ke akar lain daripada yang diinginkan, atau tidak ke akar mana pun. Periksa bahwa hasilnya berada di rentang yang diharapkan dan bahwa $|f|$ kecil di sana.
- **Memilih bentuk titik tetap tanpa memeriksa $|g'|$.** $x = 2/x$ dan $x = x^2 + x - 2$ keduanya rumusan ulang yang benar dari $x^2 = 2$ dan keduanya gagal. Hitung g' di akar yang diharapkan lebih dulu: iterasinya konvergen hanya bila $|g'(r)| < 1$, dan cepat hanya bila $|g'(r)|$ kecil.
- **Mengharapkan konvergensi kuadratik pada akar ganda.** Pada akar ganda Newton hanya membagi dua galatnya per langkah. Bila Newton ternyata lambat, periksa apakah $f'(r)$ juga mendekati nol.

3.16 Ringkasan Bab

- Metode terbuka mengikuti bentuk fungsi alih-alih mempertahankan kurung. Metode ini cepat di dekat akar tetapi memerlukan titik awal yang baik.
- Newton-Raphson mengikuti garis singgung, $x_{k+1} = x_k - f(x_k)/f'(x_k)$ (Persamaan (3.3), **Pseudokode 3.1**). Di dekat akar sederhana galatnya dikuadratkan di setiap langkah (Persamaan (3.4)): konvergensi kuadratik, orde 2 dalam arti Persamaan (3.5).
- Newton gagal bila garis singgungnya datar, bila fungsinya terlalu curam di akar, bila ia berputar dalam siklus, dan melambat menjadi konvergensi linear pada akar ganda.
- Metode secant mengganti turunan dengan kemiringan melalui dua titik terakhir (Persamaan (3.6)); ordenya adalah rasio emas 1,618 (Persamaan (3.7)), dan setiap langkah hanya memerlukan satu nilai fungsi.
- Iterasi titik tetap $x_{k+1} = g(x_k)$ (Persamaan (3.8)) konvergen bila $|g'| < 1$ di dekat akar, linear dengan laju $|g'(r)|$ (Persamaan (3.9)); Newton adalah iterasi titik tetap yang g'-nya lenyap di akar.
- Orde konvergensi dapat diukur dari tiga galat berturutan (Persamaan (3.10)); untuk $\sqrt{2}$ pengukurannya memberikan 2, sekitar 1,6, dan 1 dengan rasio 0,1716.
- Volume truk air Persamaan (3.1) dan turunannya Persamaan (3.2) memberikan kedelapan tanda tongkat ukur dalam 3 sampai 5 langkah Newton masing-masing, terhadap 31 untuk biseksi, sesuai dengan `newton` dan `brentq` dari SciPy.

3.17 Soal Latihan

1. Jalankan dua langkah Newton dengan tangan untuk $f(x) = x^2 - 5$ dari $x_0 = 2$. Berapa digit benar pada setiap iterasi? Bandingkan dengan $\sqrt{5} = 2.2360680$.

2. Turunkan iterasi Newton untuk menghitung $1/a$ hanya dengan perkalian dan pengurangan, dari $f(x) = 1/x - a$. (Komputer awal yang tidak mempunyai instruksi pembagian bekerja dengan cara ini.) Jalankan tiga langkah untuk $a = 3$ dari $x_0 = 0.3$.
3. Untuk tangki, hitung konstanta C dari Persamaan (3.4) di akar 1.000 liter $h = 0.3247$ m, lalu ramalkan besar galat kedua bila galat pertama 10^{-2} m.
4. Tunjukkan bahwa metode Newton untuk $f(x) = (x-1)^2$ memberikan $e_{k+1} = e_k/2$. Lalu tunjukkan bahwa langkah yang dimodifikasi $x_{k+1} = x_k - 2f(x_k)/f'(x_k)$ menemukan akarnya dalam satu langkah. Apa aturan umumnya untuk akar berkelipatan m?
5. Cari suku bunga pinjaman Dimas dari Bab 2 dengan metode secant, mulai dari 1% dan 5%, dengan toleransi 10^{-12}. Berapa langkah yang diperlukan, dan mengapa Newton kurang praktis di sini?
6. Tiga bentuk titik tetap dari $x^3 - x - 1 = 0$, yang akarnya $r \approx 1.3247$, adalah (a) $x = x^3 - 1$, (b) $x = (x+1)^{1/3}$, dan (c) $x = 1/x + 1/x^2$. Periksa bahwa masing-masing adalah rumusan ulang yang benar, lalu tentukan mana yang konvergen dan seberapa cepat.
7. Untuk tongkat ukur, mengapa metode secant memerlukan 7 iterasi untuk 8.000 liter tetapi hanya 3 untuk 4.000 liter?

3.18 Penyelesaian Soal Latihan

1. $f'(x) = 2x$, sehingga $x_{k+1} = x_k - (x_k^2 - 5)/(2x_k) = \frac{1}{2}(x_k + 5/x_k)$. Dari $x_0 = 2$: $x_1 = \frac{1}{2}(2 + 2.5) = 2.25$, dan $x_2 = \frac{1}{2}(2.25 + 5/2.25) = \frac{1}{2}(2.25 + 2.222222) = 2.2361111$. Terhadap $\sqrt{5} = 2.2360680$, x_0 mempunyai 1 digit benar, x_1 mempunyai 2 (galat 0,014), x_2 mempunyai 5 (galat 0,000043); langkah berikutnya memberikan 2.2360680, benar sampai 10 digit.

2. $f(x) = 1/x - a$ dan $f'(x) = -1/x^2$, sehingga $x_{k+1} = x_k - \frac{1/x_k - a}{-1/x_k^2} = x_k + x_k - a x_k^2 = x_k(2 - a x_k)$: hanya perkalian dan satu pengurangan. Untuk $a = 3$: $x_1 = 0.3(2 - 0.9) = 0.33$, $x_2 = 0.33(2 - 0.99) = 0.3333$, $x_3 = 0.3333(2 - 0.9999) = 0.33333333$. Banyaknya angka 3 berlipat dua di setiap langkah. Galatnya memenuhi $e_{k+1} = -a e_k^2$ secara tepat, sehingga metode ini konvergen untuk titik awal mana pun dengan $0 < x_0 < 2/a$.

3. Pada $h = 0.3247$: $2Rh - h^2 = 0.58446 - 0.10543 = 0.47903$, akarnya 0.69212, sehingga $V' = 2 \times 3.2 \times 0.69212 = 4.4296$ dan $V'' = 2L(R-h)/\sqrt{2Rh - h^2} = 6.4 \times 0.5753/0.69212 = 5.3198$. Maka $C = 5.3198/(2 \times 4.4296) = 0.600$, sekitar lima kali nilainya di 3.000 liter: kurvanya lebih melengkung di dekat dasar. Galat pertama 10^{-2} memberikan galat kedua sekitar $0.6 \times 10^{-4} = 6 \times 10^{-5}$ m, dan galat ketiga sekitar $0.6 \times (6 \times 10^{-5})^2 = 2 \times 10^{-9}$ m. Itulah sebabnya tanda 1.000 liter memerlukan satu langkah lebih banyak daripada tanda 3.000 liter.

4. $f'(x) = 2(x-1)$, sehingga $x_{k+1} = x_k - (x_k - 1)^2/(2(x_k - 1)) = x_k - (x_k - 1)/2$, dan $e_{k+1} = x_{k+1} - 1 = e_k - e_k/2 = e_k/2$: linear dengan laju 1/2. Dengan faktor 2, $x_{k+1} = x_k - (x_k - 1) = 1$: akarnya dalam satu langkah. Secara umum, untuk akar berkelipatan m (dengan $f = (x-r)^m h(x)$ dan $h(r) \neq 0$), Newton biasa konvergen linear dengan laju $1 - 1/m$, dan langkah $x_{k+1} = x_k - m f(x_k)/f'(x_k)$ memulihkan konvergensi kuadratik, asalkan m diketahui.

5. **Kode 3.12** dan **Kode 3.13** memakai kembali fungsi pinjaman Bab 2 dengan metode secant.

Kode 3.12 `loan_secant.py`: suku bunga pinjaman Dimas dengan metode secant.

```python
"""Exercise 3.5: the loan rate of Chapter 2 with the secant method."""
from roots2 import HEADER, secant

PRICE, PAY, MONTHS = 6000000.0, 600000.0, 12


def f(r):
    q = 1.0
    for _ in range(MONTHS):
        q *= 1 + r
    return PAY * (1 - 1 / q) / r - PRICE


def main():
    print(HEADER)
    r, k = secant(f, 0.01, 0.05, tol=1e-12, trace=True)
    print(f"monthly rate {r * 100:.10f} % after {k} iterations")


if __name__ == "__main__":
    main()
```

Kode 3.13 `LoanSecant.java`: hal yang sama dalam Java.

```java
import java.util.Locale;

/** Exercise 3.5: the loan rate of Chapter 2 with the secant method. */
public class LoanSecant {
    static final double PRICE = 6000000.0, PAY = 600000.0;
    static final int MONTHS = 12;

    static double f(double r) {
        double q = 1.0;
        for (int i = 0; i < MONTHS; i++) {
            q *= 1 + r;
        }
        return PAY * (1 - 1 / q) / r - PRICE;
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.println(Roots2.HEADER);
        Roots2.Result r = Roots2.secant(LoanSecant::f, 0.01, 0.05, 1e-12, 50,
                                        true);
        System.out.printf("monthly rate %.10f %% after %d iterations%n",
                          r.root() * 100, r.iterations());
    }
}
```

```
  k                  x           f(x)  |x - x_old|
  1     0.030989445871  -6.271988e+04    1.901e-02
  2     0.029064234179   5.902075e+03    1.925e-03
  3     0.029229818857  -4.587681e+01    1.656e-04
  4     0.029228541695  -3.324991e-02    1.277e-06
  5     0.029228540769   2.067536e-07    9.263e-10
  6     0.029228540769   4.656613e-09    5.759e-15
monthly rate 2.9228540769 % after 6 iterations
```

Enam langkah mencapai toleransi 10^{-12}, sedangkan biseksi memerlukan 19 langkah untuk toleransi 10^{-7} yang jauh lebih longgar. Langkah pertamanya sama dengan langkah pertama regula falsi di Bab 2, tetapi kemudian metode secant meninggalkan kurung: langkah 2 memakai dua titik terbaru, 5% dan 3,10%, mendarat di sisi lain akar ($f > 0$), dan sejak itu galatnya turun superlinear: 1.8×10^{-3}, 1.6×10^{-4}, 1.3×10^{-6}, 9×10^{-10}. Newton akan memerlukan turunan fungsi pinjaman di Subbab 2.2.2 terhadap r, yang memang ada tetapi berupa ekspresi panjang

dengan pangkat dan hasil bagi, dan satu kesalahan di dalamnya akan diam-diam memperlambat metodenya. Metode secant mendapat kecepatan yang hampir sama secara cuma-cuma.

6. Ketiganya benar: (a) menambahkan x ke kedua ruas; (b) mengambil akar pangkat tiga dari $x^3 = x + 1$; (c) membagi $x^3 = x + 1$ dengan x^2. (a) $g(x) = x^3 - 1$ mempunyai $g'(r) = 3r^2 = 5.26 > 1$: divergen. (b) $g(x) = (x+1)^{1/3}$ mempunyai $g'(x) = \frac{1}{3}(x+1)^{-2/3}$, dan di akar, tempat $(r+1)^{1/3} = r$, $g'(r) = 1/(3r^2) = 0.19$: konvergen linear, setiap langkah memperoleh sekitar $-\log_{10}0.19 = 0.72$ digit. (c) $g(x) = 1/x + 1/x^2$ mempunyai $g'(r) = -1/r^2 - 2/r^3 = -0.570 - 0.860 = -1.43$: divergen sambil berosilasi. Bentuk (b)-lah yang dipakai; Newton, yang g'-nya lenyap di akar, lebih baik lagi. Selalu periksa rumusan ulang dengan mengalikannya kembali: mudah sekali menghasilkan bentuk seperti $x = 1 + 1/x$, yang tampak serupa tetapi milik persamaan lain, $x^2 = x + 1$.

7. Keduanya dimulai dari tebakan kotak, dan banyaknya langkah bergantung pada seberapa jauh tebakan itu dari akar dan seberapa kuat kurvanya melengkung di sana. Pada 4.000 liter tangki hampir tepat setengah penuh, tempat kurvanya hampir lurus (lengkungannya berganti tanda pada setengah kedalaman, $V''(R) = 0$), dan tebakan kotak 0.8842 m berjarak 0,35 cm dari akar 0.8876 m: secant melalui dua titik yang berdekatan hampir sama dengan garis singgung, dan 3 langkah sudah cukup. Pada 8.000 liter tebakan kotak 1.769 m berjarak 5,5 cm dari akar 1.7137 m, kurvanya mendatar ke arah puncak, dan metode secant, yang ordenya hanya 1,618, memerlukan 7 langkah sedangkan Newton memerlukan 5.

Daftar Pustaka

[1] T. J. Ypma, "Historical development of the Newton–Raphson method," *SIAM Review*, vol. 37, no. 4, pp. 531–551, 1995.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[4] SciPy Developers, "scipy.optimize.newton," *SciPy 1.18.1 Reference Guide*, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html

Bab 4

Sistem Linear I: Eliminasi Gauss dan Pivoting

Bab 2 dan 3 mencari satu peubah dari satu persamaan. Kebanyakan masalah nyata mempunyai banyak peubah yang terikat oleh banyak persamaan sekaligus: beberapa harga yang tersembunyi dalam beberapa nota, beberapa pupuk yang dicampur untuk memenuhi beberapa sasaran hara, ribuan suhu pada pelat yang dipanaskan, jutaan bobot dalam sebuah model. Bila persamaan-persamaannya linear, satu metode yang sangat tua menyelesaikan semuanya: lenyapkan peubah satu demi satu sampai tersisa satu persamaan dengan satu peubah, lalu bekerja mundur. Bab ini membangun metode itu, eliminasi Gauss, dengan kecermatan yang diperlukan komputer. Di akhir bab Anda akan mampu menulis sistem linear dalam bentuk matriks dan menyelesaikannya dengan tangan melalui eliminasi dan substitusi mundur; menghitung banyaknya operasi aritmetika yang diperlukannya dan meramalkan bagaimana biayanya tumbuh bersama ukuran sistem; menjelaskan mengapa pivot nol atau pivot yang sangat kecil merusak perhitungan dan bagaimana *partial pivoting* memperbaikinya; mengukur kepekaan sistem terhadap perubahan kecil dengan bilangan kondisi, serta membedakan residu kecil dari galat kecil; mengimplementasikan seluruh metode dalam Java dan Python dengan jejak langkah demi langkah; dan memakainya untuk mencampur pupuk bagi kebun durian, sambil memeriksa setiap jawaban terhadap NumPy.

4.1 Pohon-Pohon Durian Bu Rahma

Bu Rahma dan suaminya merawat kebun buah dan durian di lereng bukit dekat Jember: rambutan dan manggis di sepanjang jalan setapak, dan empat puluh pohon durian yang sudah enam tahun berbuah. Pohon durian yang berbuah lebat tahun demi tahun mengambil banyak sekali hara dari tanah, dan penyuluh pertanian lapangan (PPL) di kecamatannya, setelah uji tanah, telah menuliskan apa yang semestinya diterima setiap pohon produktif selama setahun: 480 g nitrogen (N), 490 g fosfat (dihitung sebagai P_2O_5), 660 g kalium (K_2O), dan 320 g belerang (S).

Toko tani di desa tidak menjual nitrogen atau fosfat. Toko itu menjual karung-karung pupuk, dan setiap pupuk mengandung beberapa hara sekaligus. Label pupuk majemuk “NPK 15-15-15” berarti 15% N, 15% P_2O_5, dan 15% K_2O menurut massa, dan yang ada di toko itu juga membawa 10% belerang. DAP (diamonium fosfat, “18-46-0”) mengandung 18% N dan 46% P_2O_5. KCl (“0-0-60”) mengandung 60% K_2O. ZA (amonium sulfat) mengandung 21% N dan 24% S. Berapa gram setiap pupuk yang harus diterima setiap pohon agar keempat hara itu tepat?

Bu Rahma mencoba dengan kalkulator. Lebih banyak DAP memberi lebih banyak fosfat tetapi juga lebih banyak nitrogen, yang berarti lebih sedikit ZA, yang berarti lebih sedikit belerang, yang berarti lebih banyak NPK, yang membawa lebih banyak fosfat lagi… Setiap perubahan menarik semua yang lain. Itulah ciri sebuah *sistem* persamaan: empat peubah, empat syarat, semuanya saling terkait. Masalah seperti ini juga persis jenis masalah yang diselesaikan orang di Tiongkok dengan batang hitung dua ribu tahun yang lalu, dan yang diselesaikan setiap *spreadsheet*, program rekayasa, dan pustaka *machine learning* hari ini dengan metode bab ini [1].

Kita mulai dari versi yang lebih kecil yang dapat diperiksa siapa pun di meja dapur, lalu kembali ke kebun di Subbab 4.7. Semua yang dikatakan tentang pupuk di sini adalah aritmetika: rekomendasinya sendiri harus berasal dari uji tanah dan penyuluh pertanian Anda, karena tanah, pohon, dan musim berbeda-beda.

4.2 Gagasannya dengan Kata-Kata Sederhana

4.2.1 Tiga nota, tiga harga

Putri Bu Rahma menyimpan tiga nota belanja dari warung yang sama, dari tiga minggu ketika harga tidak berubah, tetapi nota-nota itu hanya menunjukkan total:

- minggu 1: 2 kg beras, 1 liter minyak goreng, dan 2 kg gula, total Rp82.000;
- minggu 2: 5 kg beras, 2 liter minyak, dan 1 kg gula, total Rp128.000;
- minggu 3: 3 kg beras, 2 liter minyak, dan 3 kg gula, total Rp132.000.

Berapa harga masing-masing? Sebut harganya (dalam ribuan rupiah) x_1 untuk beras, x_2 untuk minyak, dan x_3 untuk gula. Setiap nota adalah satu persamaan:

$$\begin{aligned} 2x_1 + x_2 + 2x_3 &= 82 \\ 5x_1 + 2x_2 + x_3 &= 128 \\ 3x_1 + 2x_2 + 3x_3 &= 132 \end{aligned} \tag{4.1}$$

Setiap persamaan bersifat **linear**: setiap peubah hanya muncul dikalikan sebuah bilangan lalu dijumlahkan, tidak pernah dikuadratkan, dikalikan peubah lain, atau berada di dalam sebuah fungsi. Akan lebih mudah bila bilangan dipisahkan dari peubahnya. Koefisien-koefisien membentuk **matriks** A, tabel bilangan persegi panjang dengan entri a_{ij} di baris i dan kolom j; peubah-peubah membentuk vektor kolom x; total-total membentuk vektor kolom b; dan seluruh sistem ditulis

$$Ax = b, \qquad A = \begin{pmatrix} 2 & 1 & 2 \\ 5 & 2 & 1 \\ 3 & 2 & 3 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad b = \begin{pmatrix} 82 \\ 128 \\ 132 \end{pmatrix} \tag{4.2}$$

Baris ke-i dari $Ax = b$ berbunyi $a_{i1}x_1 + a_{i2}x_2 + a_{i3}x_3 = b_i$, yaitu nota ke-$i$. Untuk pekerjaan tangan, kita menulis A dan b berdampingan sebagai **matriks diperbesar** (*augmented matrix*) $[A \mid b]$, dan setiap operasi pada persamaan menjadi operasi pada barisnya.

Mengapa tiga nota? Dengan satu nota ada tak hingga banyaknya kombinasi harga yang cocok dengan totalnya; dengan dua nota, masih tak hingga banyaknya, karena tiga peubah memerlukan tiga keping informasi yang saling bebas. Setiap nota yang bukan gabungan nota-nota lain menghapus satu derajat kebebasan, dan dengan nota bebas sebanyak barangnya, tersisa tepat satu himpunan harga. Nota keempat akan menjadi pemeriksaan yang berharga: jika harga-harganya memang tidak berubah, nota itu harus cocok tepat dengan ketiga harga tadi, dan jika tidak cocok, berarti ada harga yang berubah atau ada nota yang salah disalin. Bab 8 menunjukkan cara memakai *semua* nota ketika banyaknya melebihi banyak barang dan nota-nota itu tidak cocok dengan sempurna. Pertanyaan-pertanyaan seperti ini, berapa banyak informasi yang diperlukan dan apakah informasi itu saling bebas, akan muncul kembali di setiap bab tentang sistem linear.

4.2.2 Eliminasi dengan tangan

Aljabar sekolah menyelesaikan sistem seperti itu dengan *eliminasi*: memakai satu persamaan untuk melenyapkan sebuah peubah dari persamaan lainnya. Tiga jenis operasi baris mengubah bentuk sistem tetapi tidak mengubah penyelesaiannya: menambahkan kelipatan satu persamaan ke persamaan lain, menukar dua persamaan, dan mengalikan sebuah persamaan dengan bilangan bukan nol. Eliminasi Gauss terutama memakai yang pertama.

Langkah 1: lenyapkan x_1 dari baris 2 dan 3. Baris pertama memuat $2x_1$. Untuk meniadakan $5x_1$ di baris 2, kurangkan $5/2 = 2.5$ kali baris 1 dari baris 2; untuk meniadakan $3x_1$ di baris 3, kurangkan $3/2 = 1.5$ kali baris 1. Bilangan 2,5 dan 1,5 disebut **pengali** (*multiplier*), dan entri 2 yang menjadi pembagi disebut **pivot**:

- baris 2: $(5,2,1 \mid 128) - 2.5 \times (2,1,2 \mid 82) = (0, -0.5, -4 \mid -77)$;
- baris 3: $(3,2,3 \mid 132) - 1.5 \times (2,1,2 \mid 82) = (0,0.5,0 \mid 9)$.

Langkah 2: lenyapkan x_2 dari baris 3. Pivot yang baru adalah -0.5, di baris 2. Pengali untuk baris 3 adalah $0.5/(-0.5) = -1$, sehingga baris 3 menjadi $(0,0.5,0 \mid 9) - (-1) \times (0, -0.5, -4 \mid -77) = (0,0, -4 \mid -68)$.

Sistemnya kini **segitiga atas**: setiap entri di bawah diagonal bernilai nol.

$$\left(\begin{array}{ccc|c} 2 & 1 & 2 & 82 \\ 0 & -0.5 & -4 & -77 \\ 0 & 0 & -4 & -68 \end{array}\right)$$

Langkah 3: substitusi mundur. Baris terakhir berbunyi $-4x_3 = -68$, jadi $x_3 = 17$. Baris tengah berbunyi $-0.5x_2 - 4 \times 17 = -77$, jadi $-0.5x_2 = -9$ dan $x_2 = 18$. Baris pertama berbunyi $2x_1 + 18 + 2 \times 17 = 82$, jadi $2x_1 = 30$ dan $x_1 = 15$. Beras Rp15.000 per kg, minyak Rp18.000 per liter, gula Rp17.000 per kg, dan memasukkan harga-harga ini kembali ke ketiga nota memberikan tepat 82, 128, dan 132 ribu. Program di Subbab 4.6 mencetak setiap matriks antara ini.

4.3 Eliminasi Gauss

4.3.1 Eliminasi maju

Untuk sistem umum berisi n persamaan dengan n peubah, prosedur di atas diulang untuk setiap kolom $k = 1,2, \ldots, n-1$. Entri a_{kk} pada diagonal adalah pivotnya. Untuk setiap baris i di bawahnya, pengalinya adalah

$$m_{ik} = \frac{a_{ik}}{a_{kk}} \tag{4.3}$$

dan baris i diperbarui dengan mengurangkan m_{ik} kali baris k:

$$a_{ij} \leftarrow a_{ij} - m_{ik}\, a_{kj} \quad (j = k+1, \ldots, n), \qquad b_i \leftarrow b_i - m_{ik}\, b_k \tag{4.4}$$

setelah itu a_{ik} bernilai nol menurut konstruksinya. **Gambar 4.1** memperlihatkan polanya untuk empat persamaan: setiap tahap membersihkan satu kolom di bawah pivot, dan hanya blok berarsir di kanan bawah yang berubah. Nol-nolnya membentuk tangga, dan pekerjaannya menyusut ketika tangganya tumbuh.

Gambar 4.1 Eliminasi Gauss pada empat persamaan: di setiap tahap pivot (jingga) membersihkan kolomnya di bawahnya dan hanya entri berarsir yang berubah; substitusi mundur kemudian mengubah kolom kanan menjadi penyelesaiannya.

Metode ini dinamai menurut Carl Friedrich Gauss, yang memakainya sekitar 1809 untuk menyelesaikan sistem-sistem dalam metode kuadrat terkecilnya, tetapi metode ini jauh lebih tua. Kitab klasik Tiongkok *Sembilan Bab tentang Seni Matematika* (*Jiuzhang Suanshu*), yang dihimpun sekitar abad pertama, menguraikan eliminasi yang sama pada papan hitung sebagai metode *fangcheng*, diterapkan pada soal-soal hasil panen gandum yang sangat mirip dengan nota-nota kita [1].

4.3.2 Substitusi mundur

Setelah eliminasi selesai, matriksnya segitiga atas, dan peubah-peubahnya ditemukan dari yang terakhir ke yang pertama. Baris i berbunyi $a_{ii}x_i + a_{i,i+1}x_{i+1} + \cdots + a_{in}x_n = b_i$, dengan $x_{i+1}, \ldots, x_n$ sudah diketahui, sehingga

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}\, x_j \right), \qquad i = n, n-1, \ldots, 1 \tag{4.5}$$

Pseudokode 4.1 menggabungkan kedua tahap itu.

Pseudokode 4.1 Eliminasi Gauss dengan *partial pivoting* opsional (baris 2 sampai 4) dan substitusi mundur.

```
GAUSS(A, b, pivot)                  // A is n x n, b has n entries
 1  for k = 1 to n - 1                          // forward elimination
 2      if pivot:
 3          p = the row i >= k with the largest |a_ik|
 4          swap rows k and p of A and b
 5      if a_kk == 0: error "zero pivot"
 6      for i = k + 1 to n
 7          m = a_ik / a_kk                     // the multiplier
 8          a_ik = 0
 9          for j = k + 1 to n: a_ij = a_ij - m * a_kj
10          b_i = b_i - m * b_k
11  if a_nn == 0: error "zero pivot"
12  for i = n downto 1                          // back substitution
13      s = b_i
14      for j = i + 1 to n: s = s - a_ij * x_j
15      x_i = s / a_ii
16  return x
```

4.3.3 Menghitung pekerjaannya

Berapa lama eliminasi berlangsung untuk 10, 100, atau 10.000 peubah? Kita menghitung **flop**, operasi *floating point* (masing-masing satu penjumlahan, pengurangan, perkalian, atau pembagian). Pada tahap k, ada $m = n - k$ baris di bawah pivot. Untuk masing-masing kita menghitung satu pengali (1 pembagian), memperbarui m entri baris itu (satu perkalian dan satu pengurangan masing-masing, $2m$ flop), dan memperbarui b_i (2 flop): $2m + 3$ flop per baris, $m(2m + 3)$ per tahap. Menjumlahkan untuk $m = 1, \dots, n - 1$ dengan rumus $\sum m = \frac{(n-1)n}{2}$ dan $\sum m^2 = \frac{(n-1)n(2n-1)}{6}$ memberikan biaya eliminasi $\frac{(n-1)n(2n-1)}{3} + \frac{3(n-1)n}{2}$. Substitusi mundur memerlukan, untuk baris i, $2(n - i)$ flop untuk jumlahnya dan 1 untuk pembagiannya, total n^2. Seluruhnya

$$\text{flops}(n) = \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{7}{6}n \approx \frac{2}{3}n^3 \tag{4.6}$$

Untuk n besar suku pangkat tiga mendominasi: menggandakan banyaknya peubah melipatdelapankan pekerjaannya. Eliminasi berbiaya $\frac{2}{3}n^3$ sedangkan substitusi mundur hanya n^2, hal yang menjadi penting di Bab 6, tempat satu eliminasi dipakai ulang untuk banyak ruas kanan. **Kode 4.1** dan **Kode 4.2** menghitung flop implementasi kita dan membandingkannya dengan Persamaan (4.6).

Kode 4.1 `op_count.py`: flop eliminasi Gauss yang dihitung terhadap rumus Persamaan (4.6).

```
"""Counting the work: flops of Gaussian elimination against 2n^3/3."""
import gauss


def main():
    print(f"{'n':>4} {'flops counted':>14} {'formula':>14} {'2n^3/3':>14}")
    seed = 7
    for n in (3, 10, 50, 100, 200):
        a, b = [], []
        for i in range(n):                          # a random, safe matrix
            row = []
            for j in range(n):
                seed = (1103515245 * seed + 12345) % 2147483648
                row.append(seed / 2147483648.0 + (n if i == j else 0))
            a.append(row)
            b.append(1.0)
        gauss.solve(a, b)
        exact = (4 * n ** 3 + 9 * n ** 2 - 7 * n) // 6
        print(f"{n:>4} {gauss.flops:>14,} {exact:>14,} {2 * n ** 3 / 3:>14,.0f}")


if __name__ == "__main__":
    main()
```

Kode 4.2 `OpCount.java`: penghitungan yang sama dalam Java.

```
import java.util.Locale;

/** Counting the work: flops of Gaussian elimination against 2n^3/3. */
public class OpCount {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.printf("%4s %14s %14s %14s%n", "n", "flops counted",
                          "formula", "2n^3/3");
        long seed = 7;
        for (int n : new int[] {3, 10, 50, 100, 200}) {
            double[][] a = new double[n][n];
            double[] b = new double[n];
            for (int i = 0; i < n; i++) {        // a random, safe matrix
                for (int j = 0; j < n; j++) {
                    seed = (1103515245L * seed + 12345L) % 2147483648L;
                    a[i][j] = seed / 2147483648.0 + (i == j ? n : 0);
                }
                b[i] = 1.0;
            }
            Gauss.solve(a, b);
            long nn = n;
            long exact = (4 * nn * nn * nn + 9 * nn * nn - 7 * nn) / 6;
            System.out.printf("%4d %,14d %,14d %,14.0f%n", n, Gauss.flops, exact,
                              2.0 * n * n * n / 3);
        }
    }
}
```

```
   n  flops counted        formula         2n^3/3
   3              28             28             18
  10             805            805            667
  50          87,025         87,025         83,333
 100         681,550        681,550        666,667
 200       5,393,100      5,393,100      5,333,333
```

Hitungannya cocok persis dengan rumus untuk setiap ukuran, dan untuk $n = 200$ suku utama $\frac{2}{3}n^3$ sudah berselisih hanya sekitar 1% dari totalnya. Matriks-matriksnya dibuat *dominan diagonal* (setiap entri diagonal lebih besar daripada jumlah entri lain di barisnya) dengan menambahkan n ke diagonal, yang menjamin tidak ada pivot yang nol; Bab 5 menjelaskan sebabnya. Dengan beberapa miliar flop per detik, sebuah laptop menyelesaikan 1.000 persamaan dalam jauh kurang dari satu detik (6.7×10^8 flop), tetapi 100.000 persamaan akan memerlukan 6.7×10^{14} flop, beberapa hari. Itulah sebabnya sistem-sistem besar dalam praktik bersifat *jarang* (*sparse*, sebagian besar entrinya nol, yang dilewati oleh metode yang baik) atau diselesaikan dengan metode iteratif Bab 5.

4.3.4 Ketika tidak ada penyelesaian tunggal

Eliminasi dapat menjumpai pivot yang tepat nol, dengan hanya nol di bawahnya dalam kolom itu. Maka pertukaran baris tidak menolong, dan sistem itu tidak mempunyai tepat satu penyelesaian: tidak ada sama sekali (persamaan-persamaannya saling bertentangan) atau tak hingga banyaknya (satu persamaan mengulang informasi persamaan lain). Dua nota "1 kg beras dan 2 liter minyak" seharga Rp51.000 dan "2 kg beras dan 4 liter minyak" seharga Rp102.000 mengatakan hal yang sama dua kali dan tidak dapat menetapkan dua harga. Matriks seperti itu disebut **singular**. Untuk matriks yang tak singular, eliminasi dengan pertukaran baris selalu berhasil, dan ia memberikan **determinan** sebagai hasil sampingan: operasi baris jenis Persamaan (4.4) tidak mengubahnya, setiap pertukaran membalik tandanya, dan determinan matriks segitiga adalah hasil kali diagonalnya, sehingga

$$\det A = (-1)^s \prod_{k=1}^{n} a_{kk}^{(\text{final})} \tag{4.7}$$

dengan s banyaknya pertukaran baris. Untuk nota-nota tadi, tanpa pertukaran dan dengan pivot $2, -0.5, -4$, $\det A = 4$. Determinan yang tepat nol berarti singular, tetapi besar determinan yang tidak nol hampir tidak mengatakan apa-apa tentang seberapa *layak dipercaya* sebuah penyelesaian; Subbab 4.8 memperkenalkan ukuran yang tepat.

4.3.5 Banyak ruas kanan

Kebun Bu Rahma berisi pohon-pohon dengan umur berbeda. Pohon muda yang belum berbuah menerima rekomendasi yang berbeda dari pohon produktif, dan pohon tua rekomendasi ketiga. *Matriks* Persamaan (4.11) hanya bergantung pada pupuknya, jadi sama untuk setiap umur; hanya ruas kanan b yang berubah. Mengulang seluruh eliminasi untuk setiap b akan memboroskan kerja, karena pengali m_{ik} hanya bergantung pada A. Cara yang efisien adalah mengeliminasi sekali, *menyimpan pengalinya*, dan untuk setiap b baru menerapkan pengali yang sama pada b (pembaruan $b_i \leftarrow b_i - m_{ik} b_k$ dari Persamaan (4.4), sekitar n^2 flop) lalu melakukan substitusi mundur (n^2 lagi). Untuk tiga umur dan empat pupuk penghematannya remeh, tetapi untuk model yang menyelesaikan sistem yang sama dengan seribu ruas kanan berbeda, biayanya turun dari $1000 \times \frac{2}{3}n^3$ menjadi $\frac{2}{3}n^3 + 1000 \times 2n^2$. Menyimpan pengali ternyata sama dengan menulis A sebagai hasil kali matriks segitiga bawah L (yang memuat pengali-pengali) dan matriks segitiga atas U (hasil eliminasi): dekomposisi LU, yang dikembangkan Bab 6 bersama kerabatnya, Cholesky dan QR.

4.4 Pivoting

4.4.1 Pivot nol

Seandainya putri Bu Rahma menulis nota-nota itu dalam urutan lain, dan nota pertama dalam daftar baru tidak memuat beras: 1 liter minyak dan 2 kg gula seharga Rp52.000. Sistem

$$\begin{pmatrix} 0 & 1 & 2 \\ 5 & 2 & 1 \\ 3 & 2 & 3 \end{pmatrix} x = \begin{pmatrix} 52 \\ 128 \\ 132 \end{pmatrix} \tag{4.8}$$

mempunyai harga-harga yang sama sebagai penyelesaiannya (periksa: $18 + 34 = 52$), tetapi eliminasi biasa langsung berhenti, karena pivot pertamanya 0 dan pengali $5/0$ serta $3/0$ tidak ada. Obatnya jelas bagi manusia: pakai persamaan lain lebih dulu. Menukar baris 1 dan 2 memberikan pivot 5, dan eliminasi berjalan seperti biasa. Komputer harus diberi tahu untuk melakukannya.

4.4.2 Pivot yang sangat kecil

Pivot nol setidaknya mengumumkan dirinya. Pivot yang *sangat kecil* lebih buruk, karena diam-diam menghancurkan jawabannya. Perhatikan

$$10^{-20}\, x + y = 1, \qquad x + y = 2 \tag{4.9}$$

yang penyelesaiannya sangat dekat dengan $x = 1$, $y = 1$. Eliminasi biasa memakai pivot 10^{-20} dan pengali $m = 1/10^{-20} = 10^{20}$. Baris 2 menjadi $(1 - 10^{20})\, y = 2 - 10^{20}$. Dalam presisi ganda,

dengan sekitar 16 angka bena, $1 - 10^{20}$ dibulatkan menjadi -10^{20} dan $2 - 10^{20}$ dibulatkan menjadi -10^{20}: angka 1 dan 2 itu lenyap, persis seperti beberapa rupiah lenyap ketika ditambahkan ke satu miliar dalam presisi tunggal (Subbab 1.13). Jadi $y = 1$, yang kebetulan benar. Tetapi substitusi mundur di baris 1 lalu memberikan $10^{-20}x = 1 - y = 0$, sehingga $x = 0$, sama sekali salah. Informasi persamaan kedua tenggelam oleh pengali yang sangat besar.

Dengan baris yang ditukar, pivotnya 1, pengalinya 10^{-20}, dan baris 2 menjadi $(1 - 10^{-20})y = 1 - 2 \times 10^{-20}$, yang dibulatkan menjadi $y = 1$; baris 1 lalu memberikan $x = 2 - y = 1$. Benar. Perbedaannya sepenuhnya terletak pada besar pengalinya: pengali yang lebih besar dari 1 memperbesar galat pembulatan baris pivot ketika baris itu dikurangkan dari baris-baris lain.

4.4.3 *Partial pivoting*

Partial pivoting membuat pilihan itu sistematis. Pada tahap k, sebelum mengeliminasi, telusuri kolom k ke bawah dari diagonal dan bawa baris dengan entri terbesar dalam nilai mutlak ke posisi k:

$$p = \operatorname{argmax}_{i \geq k} |a_{ik}|, \qquad \text{tukar baris } k \text{ dan } p \tag{4.10}$$

Maka setiap pengali memenuhi $|m_{ik}| \leq 1$, sehingga tidak ada baris yang pernah dibesarkan sebelum dikurangkan. Baris 2 sampai 4 dari **Pseudokode 4.1** mengimplementasikannya. Pertukaran tidak memakan flop, hanya satu perbandingan per baris, dan menyingkirkan baik pivot nol (matriks tak singular selalu mempunyai entri bukan nol di suatu tempat di kolom itu) maupun pivot yang sangat kecil.

Apakah *partial pivoting* selalu cukup? Secara teori tidak. Wilkinson menunjukkan bahwa entri-entri masih bisa tumbuh selama eliminasi, dalam kasus terburuk dengan faktor 2^{n-1}, dan membangun matriks-matriks tempat hal itu terjadi. Tetapi dalam praktik pertumbuhan seperti itu begitu jarang sehingga eliminasi Gauss dengan *partial pivoting* dipandang stabil untuk segala keperluan praktis, dan itulah metode yang dipakai LAPACK, dan karena itu NumPy, MATLAB, dan R, untuk sistem umum [2], [3]. Ada dua varian yang lebih kuat: *scaled partial pivoting* membandingkan $|a_{ik}|$ relatif terhadap entri terbesar baris i, yang melindungi dari baris-baris yang ditulis dalam satuan yang sangat berbeda, dan *complete pivoting* mencari entri terbesar di seluruh blok yang tersisa dan juga menukar kolom, dengan biaya $O(n^3)$ perbandingan tambahan. Keduanya jarang diperlukan [4].

4.5 Membangunnya dalam Java dan Python

Kode 4.3 dan **Kode 4.4** mengimplementasikan **Pseudokode 4.1**, sebuah fungsi untuk residu (Subbab 4.8), dan pembaca kecil untuk berkas data berpemisah koma. Matriks dan ruas kanannya disalin lebih dulu, sehingga data milik pemanggil tidak pernah berubah.

Kode 4.3 `gauss.py`: eliminasi Gauss dengan *partial pivoting* opsional, substitusi mundur, residu, dan pembaca data.

```
"""Gaussian elimination with (optional) partial pivoting and back
substitution, written out in full so that every step can be followed."""

flops = 0                    # floating-point operations of the last solve


def show(a, b, title):
    """Print the augmented matrix [A | b]."""
    print(title)
    for row, rhs in zip(a, b):
        print(" " + "".join(f" {v:10.4f}" for v in row) + f" | {rhs:10.4f}")


def solve(a, b, pivot=True, trace=False):
    """Solve A x = b. A and b are copied, never changed.
    Raises ZeroDivisionError when a pivot is exactly zero."""
    global flops
    flops = 0
    n = len(b)
    a = [list(map(float, row)) for row in a]
    b = [float(v) for v in b]
    if trace:
        show(a, b, "start")
    for k in range(n - 1):
        if pivot:                                  # partial pivoting:
            p = max(range(k, n), key=lambda i: abs(a[i][k]))
            if p != k:                             # bring the largest |a_ik|
                a[k], a[p] = a[p], a[k]            # of column k to row k
                b[k], b[p] = b[p], b[k]
                if trace:
                    print(f"swap rows {k + 1} and {p + 1}")
        if a[k][k] == 0:
            raise ZeroDivisionError(f"zero pivot in column {k + 1}")
        for i in range(k + 1, n):
            m = a[i][k] / a[k][k]                  # the multiplier
            flops += 1
            a[i][k] = 0.0
            for j in range(k + 1, n):
                a[i][j] -= m * a[k][j]
                flops += 2
            b[i] -= m * b[k]
            flops += 2
        if trace:
            show(a, b, f"after eliminating column {k + 1}")
    if a[n - 1][n - 1] == 0:
        raise ZeroDivisionError(f"zero pivot in column {n}")
```

Kode 4.3 (Lanjutan) *gauss.py: eliminasi Gauss dengan partial pivoting opsional, substitusi mundur, residu, dan pembaca data.*

```
    x = [0.0] * n
    for i in range(n - 1, -1, -1):                # back substitution
        s = b[i]
        for j in range(i + 1, n):
            s -= a[i][j] * x[j]
            flops += 2
        x[i] = s / a[i][i]
        flops += 1
    return x


def residual(a, x, b):
    """The largest |b_i - (A x)_i|."""
    worst = 0.0
    for row, rhs in zip(a, b):
        s = rhs
        for aij, xj in zip(row, x):
            s -= aij * xj
        worst = max(worst, abs(s))
    return worst


def read_table(path):
    """Read a comma-separated file of numbers; the first field of a line may
    be a name. Lines starting with # are comments."""
    names, rows = [], []
    with open(path, encoding="utf-8") as fh:
        for line in fh:
            line = line.strip()
            if not line or line.startswith("#"):
                continue
            fields = [f.strip() for f in line.split(",")]
            try:
                float(fields[0])
                names.append("")
                rows.append([float(f) for f in fields])
            except ValueError:
                names.append(fields[0])
                rows.append([float(f) for f in fields[1:]])
    return names, rows
```

Kode 4.4 `Gauss.java`: modul yang sama dalam Java.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

/**
 * Gaussian elimination with (optional) partial pivoting and back
 * substitution, written out in full so that every step can be followed.
 */
public class Gauss {
    static long flops = 0;          // floating-point operations of last solve

    /** Print the augmented matrix [A | b]. */
    static void show(double[][] a, double[] b, String title) {
        System.out.println(title);
        for (int i = 0; i < b.length; i++) {
            StringBuilder line = new StringBuilder(" ");
            for (double v : a[i]) {
                line.append(String.format(" %10.4f", v));
            }
            System.out.println(line + String.format(" | %10.4f", b[i]));
        }
    }

    /** Solve A x = b; A and b are copied, never changed. */
    static double[] solve(double[][] a0, double[] b0, boolean pivot,
                          boolean trace) {
        flops = 0;
        int n = b0.length;
        double[][] a = new double[n][];
        for (int i = 0; i < n; i++) {
            a[i] = a0[i].clone();
        }
        double[] b = b0.clone();
        if (trace) {
            show(a, b, "start");
        }
        for (int k = 0; k < n - 1; k++) {
            if (pivot) {                          // partial pivoting: bring the
                int p = k;                        // largest |a_ik| of column k
                for (int i = k + 1; i < n; i++) {    // to row k
                    if (Math.abs(a[i][k]) > Math.abs(a[p][k])) {
                        p = i;
                    }
                }
```

Kode 4.4 (Lanjutan) *Gauss.java: modul yang sama dalam Java.*

```
                if (p != k) {
                    double[] row = a[k];
                    a[k] = a[p];
                    a[p] = row;
                    double t = b[k];
                    b[k] = b[p];
                    b[p] = t;
                    if (trace) {
                        System.out.printf("swap rows %d and %d%n", k + 1, p + 1);
                    }
                }
            }
            if (a[k][k] == 0) {
                throw new ArithmeticException("zero pivot in column " + (k + 1));
            }
            for (int i = k + 1; i < n; i++) {
                double m = a[i][k] / a[k][k];    // the multiplier
                flops += 1;
                a[i][k] = 0.0;
                for (int j = k + 1; j < n; j++) {
                    a[i][j] -= m * a[k][j];
                    flops += 2;
                }
                b[i] -= m * b[k];
                flops += 2;
            }
            if (trace) {
                show(a, b, "after eliminating column " + (k + 1));
            }
        }
        if (a[n - 1][n - 1] == 0) {
            throw new ArithmeticException("zero pivot in column " + n);
        }
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {       // back substitution
            double s = b[i];
            for (int j = i + 1; j < n; j++) {
                s -= a[i][j] * x[j];
                flops += 2;
            }
            x[i] = s / a[i][i];
            flops += 1;
        }
        return x;
    }
```

Kode 4.4 (Lanjutan) *Gauss.java: modul yang sama dalam Java.*

```
    static double[] solve(double[][] a, double[] b) {
        return solve(a, b, true, false);
    }

    /** The largest |b_i - (A x)_i|. */
    static double residual(double[][] a, double[] x, double[] b) {
        double worst = 0.0;
        for (int i = 0; i < b.length; i++) {
            double s = b[i];
            for (int j = 0; j < x.length; j++) {
                s -= a[i][j] * x[j];
            }
            worst = Math.max(worst, Math.abs(s));
        }
        return worst;
    }

    /** One line of a data file: an optional name and its numbers. */
    record Line(String name, double[] values) { }

    /** Read a comma-separated file; lines starting with # are comments. */
    static List<Line> readTable(String path) throws IOException {
        List<Line> lines = new ArrayList<>();
        for (String raw : Files.readAllLines(Path.of(path))) {
            String line = raw.strip();
            if (line.isEmpty() || line.startsWith("#")) {
                continue;
            }
            String[] f = line.split(",");
            String name = "";
            int start = 0;
            try {
                Double.parseDouble(f[0].strip());
            } catch (NumberFormatException e) {
                name = f[0].strip();
                start = 1;
            }
            double[] v = new double[f.length - start];
            for (int i = start; i < f.length; i++) {
                v[i - start] = Double.parseDouble(f[i].strip());
            }
            lines.add(new Line(name, v));
        }
        return lines;
    }
}
```

Matriks adalah daftar baris di Python dan `double[][]` di Java, larik berisi larik-larik baris. Karena itu menukar dua baris berarti menukar dua referensi tanpa menyalin bilangan apa pun, itulah sebabnya *pivoting* murah. Pencacah global `flops` (medan statis di Java) menjumlahkan operasi pemanggilan terakhir, persis seperti yang dihitung di Subbab 4.3.3: satu untuk setiap pengali, dua untuk setiap entri yang diperbarui dan setiap pembaruan b, dua untuk setiap suku jumlah substitusi mundur, dan satu untuk setiap pembagian. Pembaca data menerima baris yang diawali nama (“NPK 15-15-15, 0.15, …”) atau bilangan, dan melewati komentar yang diawali `#`, sehingga berkas data dapat menjelaskan dirinya sendiri. Pivot nol memunculkan `ZeroDivisionError` di Python dan `ArithmeticException` di Java, beserta kolom tempat hal itu terjadi.

4.6 Menjalankannya: Nota-Nota Belanja

Nota-nota itu disimpan di `data/receipts.csv`:

```
# Three receipts: kg of rice, litres of oil, kg of sugar, total (thousand Rp)
2, 1, 2, 82
5, 2, 1, 128
3, 2, 3, 132
```

Kode 4.5 dan **Kode 4.6** membaca berkas itu, menyelesaikan Persamaan (4.1) *tanpa pivoting* agar setiap langkah cocok dengan hitungan tangan kita, lalu mencetak harga-harga dan residunya.

Kode 4.5 `receipts.py`: tiga nota, tiga harga, dengan setiap langkah eliminasi dicetak.

```python
"""Three shopping receipts, three unknown prices: the worked example."""
from pathlib import Path
from gauss import read_table, residual, solve

DATA = Path(__file__).resolve().parent.parent / "data" / "receipts.csv"


def main():
    _, rows = read_table(DATA)
    a = [row[:3] for row in rows]
    b = [row[3] for row in rows]
    x = solve(a, b, pivot=False, trace=True)
    for name, price in zip(("rice", "oil", "sugar"), x):
        print(f"{name:<6}: Rp {price * 1000:,.0f}")
    print(f"largest residual: {residual(a, x, b):.3e}")


if __name__ == "__main__":
    main()
```

Kode 4.6 `Receipts.java`: program yang sama dalam Java.

```java
import java.io.IOException;
import java.util.List;
import java.util.Locale;

/** Three shopping receipts, three unknown prices: the worked example. */
public class Receipts {
    public static void main(String[] args) throws IOException {
        Locale.setDefault(Locale.ROOT);
        List<Gauss.Line> rows = Gauss.readTable("../data/receipts.csv");
        double[][] a = new double[3][3];
        double[] b = new double[3];
        for (int i = 0; i < 3; i++) {
            System.arraycopy(rows.get(i).values(), 0, a[i], 0, 3);
            b[i] = rows.get(i).values()[3];
        }
        double[] x = Gauss.solve(a, b, false, true);
        String[] names = {"rice", "oil", "sugar"};
        for (int i = 0; i < 3; i++) {
            System.out.printf("%-6s: Rp %,.0f%n", names[i], x[i] * 1000);
        }
        System.out.printf("largest residual: %.3e%n", Gauss.residual(a, x, b));
    }
}
```

```
$ python3 receipts.py          (or, in the java folder: java -cp out Receipts)
```

```
start
      2.0000     1.0000     2.0000 |    82.0000
      5.0000     2.0000     1.0000 |   128.0000
      3.0000     2.0000     3.0000 |   132.0000
after eliminating column 1
      2.0000     1.0000     2.0000 |    82.0000
      0.0000    -0.5000    -4.0000 |   -77.0000
      0.0000     0.5000     0.0000 |     9.0000
after eliminating column 2
      2.0000     1.0000     2.0000 |    82.0000
      0.0000    -0.5000    -4.0000 |   -77.0000
      0.0000     0.0000    -4.0000 |   -68.0000
rice  : Rp 15,000
oil   : Rp 18,000
sugar : Rp 17,000
largest residual: 0.000e+00
```

Ketiga matriks itu persis sama dengan yang di Subbab 4.2.2, dan harganya Rp15.000, Rp18.000, dan Rp17.000. Setiap bilangan di sini adalah bilangan bulat kecil atau setengah, semuanya dapat disajikan tepat dalam biner, sehingga aritmetikanya eksak dan residunya nol. Kebun durian tidak akan semurah hati itu.

4.7 Pupuk untuk Kebun Durian

Data pupuk masuk ke `data/fertilizer.csv`: satu baris per pupuk dengan pecahan massa keempat haranya, dan baris terakhir berisi sasaran dalam gram per pohon per tahun.

```
# One line per fertilizer: name, then the fraction of its mass that is
# N, P2O5, K2O and S. The last line is the target in grams per tree per year.
NPK 15-15-15, 0.15, 0.15, 0.15, 0.10
DAP 18-46-0, 0.18, 0.46, 0.00, 0.00
KCl 0-0-60, 0.00, 0.00, 0.60, 0.00
ZA 21-0-0-24S, 0.21, 0.00, 0.00, 0.24
target, 480, 490, 660, 320
```

Misalkan x_1, x_2, x_3, x_4 adalah gram per pohon untuk NPK, DAP, KCl, dan ZA. Nitrogen yang diberikan adalah $0.15x_1 + 0.18x_2 + 0x_3 + 0.21x_4$, dan harus sama dengan 480; hara-hara lainnya memberikan tiga persamaan lagi:

$$\begin{pmatrix} 0.15 & 0.18 & 0 & 0.21 \\ 0.15 & 0.46 & 0 & 0 \\ 0.15 & 0 & 0.60 & 0 \\ 0.10 & 0 & 0 & 0.24 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 480 \\ 490 \\ 660 \\ 320 \end{pmatrix} \tag{4.11}$$

Perhatikan bahwa berkasnya mencantumkan satu *pupuk* per baris, tetapi dalam matriks setiap *hara* adalah satu baris: matriksnya adalah transpos tabel di berkas. **Kode 4.7** dan **Kode 4.8** membangunnya, menyelesaikannya dengan *partial pivoting*, mencetak jumlah per pohon dan untuk seluruh kebun, lalu memeriksa setiap hara.

Kode 4.7 `fertilizer.py`: gram setiap pupuk per pohon, dan kilogram untuk seluruh kebun.

```python
"""How much of each fertilizer gives a durian tree its N, P, K and S?
Usage: python fertilizer.py [number of trees]"""
import sys
from pathlib import Path
from gauss import read_table, residual, solve

DATA = Path(__file__).resolve().parent.parent / "data" / "fertilizer.csv"
NUTRIENTS = ("N", "P2O5", "K2O", "S")


def main():
    trees = int(sys.argv[1]) if len(sys.argv) > 1 else 40
    names, rows = read_table(DATA)
    ferts, target = names[:-1], rows[-1]
    # row i of A is nutrient i, column j is fertilizer j
    a = [[rows[j][i] for j in range(len(ferts))] for i in range(len(NUTRIENTS))]
    x = solve(a, target, pivot=True, trace=True)
    print(f"\n{'fertilizer':<14} {'g per tree':>11} {'kg for ' + str(trees):>11}")
    for name, grams in zip(ferts, x):
        print(f"{name:<14} {grams:>11.1f} {grams * trees / 1000:>11.2f}")
    if min(x) < 0:
        print("A negative amount means: this target cannot be met with "
              "these fertilizers.")
    print(f"largest residual: {residual(a, x, target):.3e} g")
    for i, nut in enumerate(NUTRIENTS):
        got = sum(a[i][j] * x[j] for j in range(len(x)))
        print(f"  {nut:<5} wanted {target[i]:6.1f} g, gets {got:8.3f} g")


if __name__ == "__main__":
    main()
```

Kode 4.8 `Fertilizer.java`: program yang sama dalam Java.

```
import java.io.IOException;
import java.util.List;
import java.util.Locale;

/**
 * How much of each fertilizer gives a durian tree its N, P, K and S?
 * Usage: java Fertilizer [number of trees]   (run from the java folder)
 */
public class Fertilizer {
    static final String[] NUTRIENTS = {"N", "P2O5", "K2O", "S"};

    public static void main(String[] args) throws IOException {
        Locale.setDefault(Locale.ROOT);
        int trees = args.length > 0 ? Integer.parseInt(args[0]) : 40;
        List<Gauss.Line> rows = Gauss.readTable("../data/fertilizer.csv");
        int m = rows.size() - 1, n = NUTRIENTS.length;
        double[] target = rows.get(m).values();
        double[][] a = new double[n][m];        // row: nutrient, column: fert.
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                a[i][j] = rows.get(j).values()[i];
            }
        }
        double[] x = Gauss.solve(a, target, true, true);
        System.out.printf("%n%-14s %11s %11s%n", "fertilizer", "g per tree",
                          "kg for " + trees);
        double least = x[0];
        for (int j = 0; j < m; j++) {
            System.out.printf("%-14s %11.1f %11.2f%n", rows.get(j).name(), x[j],
                              x[j] * trees / 1000);
            least = Math.min(least, x[j]);
        }
        if (least < 0) {
            System.out.println("A negative amount means: this target cannot be "
                               + "met with these fertilizers.");
        }
        System.out.printf("largest residual: %.3e g%n",
                          Gauss.residual(a, x, target));
        for (int i = 0; i < n; i++) {
            double got = 0;
            for (int j = 0; j < m; j++) {
                got += a[i][j] * x[j];
            }
            System.out.printf("  %-5s wanted %6.1f g, gets %8.3f g%n",
                              NUTRIENTS[i], target[i], got);
        }
```

Kode 4.8 (Lanjutan) *Fertilizer.java: program yang sama dalam Java.*

```
    }
}
```

```
start
      0.1500     0.1800     0.0000     0.2100 |   480.0000
      0.1500     0.4600     0.0000     0.0000 |   490.0000
      0.1500     0.0000     0.6000     0.0000 |   660.0000
      0.1000     0.0000     0.0000     0.2400 |   320.0000
after eliminating column 1
      0.1500     0.1800     0.0000     0.2100 |   480.0000
      0.0000     0.2800     0.0000    -0.2100 |    10.0000
      0.0000    -0.1800     0.6000    -0.2100 |   180.0000
      0.0000    -0.1200     0.0000     0.1000 |    -0.0000
after eliminating column 2
      0.1500     0.1800     0.0000     0.2100 |   480.0000
      0.0000     0.2800     0.0000    -0.2100 |    10.0000
      0.0000     0.0000     0.6000    -0.3450 |   186.4286
      0.0000     0.0000     0.0000     0.0100 |     4.2857
after eliminating column 3
      0.1500     0.1800     0.0000     0.2100 |   480.0000
      0.0000     0.2800     0.0000    -0.2100 |    10.0000
      0.0000     0.0000     0.6000    -0.3450 |   186.4286
      0.0000     0.0000     0.0000     0.0100 |     4.2857

fertilizer      g per tree   kg for 40
NPK 15-15-15        2171.4       86.86
DAP 18-46-0          357.1       14.29
KCl 0-0-60           557.1       22.29
ZA 21-0-0-24S        428.6       17.14
largest residual: 1.421e-14 g
  N     wanted  480.0 g, gets  480.000 g
  P2O5  wanted  490.0 g, gets  490.000 g
  K2O   wanted  660.0 g, gets  660.000 g
  S     wanted  320.0 g, gets  320.000 g
```

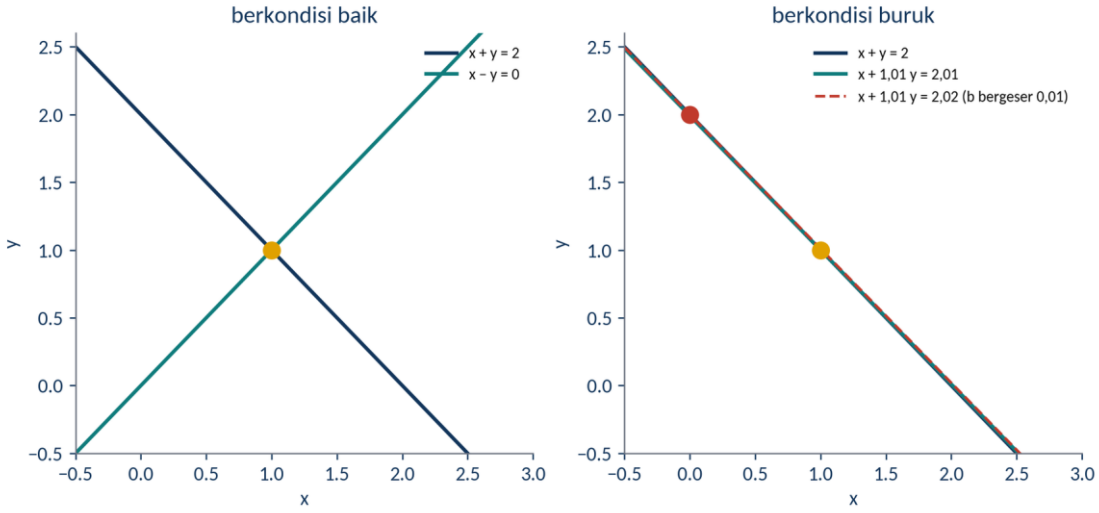
Tidak diperlukan pertukaran baris: kolom pertama memuat 0,15; 0,15; 0,15; dan 0,10, dan baris pertama sudah memuat entri terbesar (bila sama besar, baris yang lebih atas dipertahankan). Setiap pohon memerlukan sekitar 2,17 kg NPK, 357 g DAP, 557 g KCl, dan 429 g ZA setahun; untuk empat puluh pohon Bu Rahma membeli sekitar 87 kg NPK, 14 kg DAP, 22 kg KCl, dan 17 kg ZA, yang akan ia bagi ke dalam dua atau tiga kali pemupukan sesuai anjuran penyuluhnya. Pemeriksaan di bagian bawah memberikan setiap hara sampai seperseribu gram, dan residu terbesar, 1.4×10^{-14} g, adalah beberapa satuan pembulatan pada bilangan-bilangan berukuran 500. (Nilai `-0.0000` di baris keempat setelah tahap pertama adalah ruas kanan $320 - \frac{0.10}{0.15} \times 480$, yang nol dalam aritmetika eksak tetapi menjadi bilangan negatif yang sangat kecil setelah pembulatan; ia dicetak beserta tandanya.)

Ada satu kejutan, dan kejutan itu membawa kita ke gagasan terpenting bab ini. Rekomendasi penyuluh itu dibulatkan: mungkin sasaran fosfat sebenarnya 500 g, bukan 490. Mengubah satu bilangan itu saja lalu menyelesaikan lagi (Soal Latihan 4.5) memberikan 1.143 g NPK alih-alih 2.171, dan 714 g DAP alih-alih 357. Perubahan 2% pada satu sasaran mengubah dua jawaban dengan faktor dua. Dan sasaran fosfat 480 g, hanya 2% di bawah 490, memberikan jawaban "3.200 g NPK, 0 g DAP, 300 g KCl, 0 g ZA", sedangkan sasaran fosfat yang lebih rendah lagi memberikan jumlah *negatif*, yang tidak mungkin dibeli. Aritmetikanya tidak salah; sistemnya *peka*. Subbab 4.8 mengukur kepekaan itu.

4.8 Pengondisian: Residu Kecil Bukan Galat Kecil

4.8.1 Dua garis yang hampir sejajar

Dengan dua peubah, setiap persamaan linear adalah garis lurus pada bidang, dan penyelesaiannya adalah titik potong kedua garis. **Gambar 4.2** memperlihatkan dua sistem. Di kiri, $x + y = 2$ dan $x - y = 0$ berpotongan tegak lurus, dan menggeser salah satu garis sedikit menggeser titik potongnya sedikit. Di kanan, $x + y = 2$ dan $x + 1.01y = 2.01$ hampir sejajar. Keduanya tetap berpotongan di (1,1), tetapi mengubah ruas kanan kedua hanya sebesar 0,01, menjadi 2,02, menggeser titik potong itu jauh sekali ke (0,2).



Gambar 4.2 Sistem berkondisi baik (kiri) dan sistem berkondisi buruk (kanan): bila garis-garisnya hampir sejajar, perubahan 0,01 pada data menggeser penyelesaian dari (1,1) ke (0,2).

Sistem yang penyelesaiannya banyak berubah ketika datanya sedikit berubah disebut **berkondisi buruk** (*ill-conditioned*). Sifat ini milik *masalahnya*, bukan metodenya: tidak ada algoritma yang dapat menemukan titik potong dua garis yang hampir sejajar dengan teliti bila garis-garis itu sendiri hanya diketahui sampai dua desimal. Bahkan dengan aritmetika eksak, ketidakpastian data menjadi ketidakpastian jawaban yang besar. Di komputer, data selalu sedikit tidak pasti, karena dibulatkan ketika disimpan (Subbab 1.10), sehingga sistem berkondisi buruk juga memperbesar galat pembulatan.

4.8.2 Bilangan kondisi

Untuk mengukur efeknya kita memerlukan ukuran besar vektor dan matriks. **Norma tak hingga** sebuah vektor adalah entri terbesarnya dalam nilai mutlak, $\| x \|_\infty = \max_i |x_i|$, dan norma matriks yang diinduksinya adalah jumlah baris terbesar dari nilai-nilai mutlaknya, $\| A \|_\infty = \max_i \sum_j |a_{ij}|$. **Bilangan kondisi** (*condition number*) sebuah matriks tak singular adalah

$$\kappa(A) = \| A \| \, \| A^{-1} \| \tag{4.12}$$

Nilainya sedikitnya 1, dan ia membatasi seberapa besar perubahan relatif pada b dapat diperbesar dalam penyelesaiannya. Jika $Ax = b$ dan $A(x + \delta x) = b + \delta b$, maka $A\,\delta x = \delta b$, sehingga $\| \delta x \| \le \| A^{-1} \| \, \| \delta b \|$, dan $\| b \| \le \| A \| \, \| x \|$; membagi ketaksamaan pertama dengan yang kedua memberikan [2]

$$\frac{\| \delta x \|}{\| x \|} \leq \kappa(A) \frac{\| \delta b \|}{\| b \|} \tag{4.13}$$

Untuk garis-garis yang hampir sejajar, $A = \begin{pmatrix} 1 & 1 \\ 1 & 1.01 \end{pmatrix}$ mempunyai invers $\frac{1}{0.01}\begin{pmatrix} 1.01 & -1 \\ -1 & 1 \end{pmatrix}$, sehingga $\| A \|_\infty = 2.01$, $\| A^{-1} \|_\infty = 201$, dan $\kappa = 404$. Perubahan 0,5% pada b (0,01 dari 2,01) dapat mengubah x sampai $404 \times 0.5\% = 200\%$. Perubahan yang sebenarnya, dari (1,1) ke (0,2), adalah 1 dalam norma tak hingga, 100% dari $\| x \|_\infty = 1$: masih di dalam batas, dan tetap sangat besar.

Aturan praktis yang berguna menyusul dari Persamaan (4.13) dan satuan pembulatan Bab 1: karena data membawa galat relatif sekitar 10^{-16}, **penyelesaian yang dihitung dalam presisi ganda dapat kehilangan sekitar** $\log_{10}\kappa(A)$ **dari 16 angka benanya** [3]. Dengan $\kappa = 10^4$, sekitar 12 digit tersisa; dengan $\kappa = 10^{16}$, tidak ada.

Matriks pupuk Persamaan (4.11) mempunyai $\kappa_\infty = 447$ (Subbab 4.11 menghitungnya). Mengubah sasaran fosfat 10 g adalah perubahan $10/660 = 1.5\%$ pada $\| b \|_\infty$; Persamaan (4.13) mengizinkan perubahan sampai $447 \times 1.5\% = 677\%$ pada x, dan perubahan sebenarnya, 1.029 g dari 2.171 g, adalah 47%. Batas itu tidak tercapai, tetapi menjelaskan mengapa rekomendasi yang dibulatkan memberikan jawaban yang begitu berbeda. Kimianya juga menjelaskannya: NPK dan DAP sama-sama memberikan nitrogen dan fosfat, dengan proporsi yang mirip untuk nitrogen, sehingga kolom NPK dan DAP *hampir sejajar* pada bidang nitrogen-fosfat, seperti garis-garis pada **Gambar 4.2**.

4.8.3 Residu dan galat

Residu penyelesaian terhitung $\tilde{x}$ adalah $r = b - A\tilde{x}$: seberapa baik $\tilde{x}$ memenuhi persamaan-persamaannya. Residu mudah dihitung, dan program kita mencetaknya. **Galat** adalah $\tilde{x} - x$: seberapa jauh $\tilde{x}$ dari penyelesaian sejati, yang tidak kita ketahui. Hubungan keduanya lagi-lagi bilangan kondisi: $\| \tilde{x} - x \| / \| x \| \leq \kappa(A) \, \| r \| / \| b \|$. Eliminasi Gauss dengan *partial pivoting* selalu menghasilkan *residu kecil*, seorde galat pembulatan; apakah galatnya kecil bergantung pada κ.

Peragaan klasiknya memakai **matriks Hilbert**, $H_{ij} = 1/(i + j - 1)$, yang dijumpai Hilbert pada 1894 dalam masalah penghampiran fungsi dan yang terkenal berkondisi buruk. **Kode 4.9** dan **Kode 4.10** memilih ruas kanan sehingga penyelesaian eksaknya semua satu, menyelesaikan, lalu mencetak bilangan kondisi (dihitung dari invers, satu kolom demi satu kolom), residu, dan galatnya.

Kode 4.9 `conditioning.py`: matriks Hilbert dengan ukuran yang makin besar: bilangan kondisi, residu, dan galat.

```python
"""Small residual, large error: Hilbert matrices H_ij = 1 / (i + j - 1).
The exact answer is chosen to be all ones, so the error can be measured."""
from gauss import residual, solve


def inverse_norm(a):
    """||A^-1|| in the infinity norm, one column of A^-1 at a time."""
    n = len(a)
    cols = []
    for j in range(n):
        e = [1.0 if i == j else 0.0 for i in range(n)]
        cols.append(solve(a, e))
    return max(sum(abs(cols[j][i]) for j in range(n)) for i in range(n))


def main():
    print(f"{'n':>3} {'cond(H)':>10} {'residual':>10} {'max error':>10}")
    for n in range(2, 14, 2):
        h = [[1.0 / (i + j + 1) for j in range(n)] for i in range(n)]
        b = [sum(row) for row in h]                  # so that x = all ones
        x = solve(h, b)
        err = max(abs(v - 1.0) for v in x)
        norm = max(sum(abs(v) for v in row) for row in h)
        cond = norm * inverse_norm(h)
        print(f"{n:>3} {cond:>10.2e} {residual(h, x, b):>10.2e} {err:>10.2e}")


if __name__ == "__main__":
    main()
```

Kode 4.10 `Conditioning.java`: percobaan yang sama dalam Java.

```java
import java.util.Locale;

/**
 * Small residual, large error: Hilbert matrices H_ij = 1 / (i + j - 1).
 * The exact answer is chosen to be all ones, so the error can be measured.
 */
public class Conditioning {
    /** ||A^-1|| in the infinity norm, one column of A^-1 at a time. */
    static double inverseNorm(double[][] a) {
        int n = a.length;
        double[][] cols = new double[n][];
        for (int j = 0; j < n; j++) {
            double[] e = new double[n];
            e[j] = 1.0;
            cols[j] = Gauss.solve(a, e);
        }
        double best = 0;
        for (int i = 0; i < n; i++) {
            double s = 0;
            for (int j = 0; j < n; j++) {
                s += Math.abs(cols[j][i]);
            }
            best = Math.max(best, s);
        }
        return best;
    }

    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.printf("%3s %10s %10s %10s%n", "n", "cond(H)", "residual",
                          "max error");
        for (int n = 2; n < 14; n += 2) {
            double[][] h = new double[n][n];
            double[] b = new double[n];
            double norm = 0;
            for (int i = 0; i < n; i++) {
                double rowSum = 0, absSum = 0;
                for (int j = 0; j < n; j++) {
                    h[i][j] = 1.0 / (i + j + 1);
                    rowSum += h[i][j];
                    absSum += Math.abs(h[i][j]);
                }
                b[i] = rowSum;                    // so that x = all ones
                norm = Math.max(norm, absSum);
            }
            double[] x = Gauss.solve(h, b);
```

Kode 4.10 (Lanjutan) *Conditioning.java: percobaan yang sama dalam Java.*

```java
            double err = 0;
            for (double v : x) {
                err = Math.max(err, Math.abs(v - 1.0));
            }
            double cond = norm * inverseNorm(h);
            System.out.printf("%3d %10.2e %10.2e %10.2e%n", n, cond,
                              Gauss.residual(h, x, b), err);
        }
    }
}
```

```
  n    cond(H)   residual  max error
  2   2.70e+01   1.11e-16   6.66e-16
  4   2.84e+04   1.67e-16   6.11e-13
  6   2.91e+07   8.33e-17   5.26e-10
  8   3.39e+10   1.67e-16   4.19e-07
 10   3.54e+13   1.80e-16   3.18e-04
 12   3.83e+16   4.30e-16   3.56e-01
```

Residunya tetap di tingkat pembulatan, sekitar 10^{-16}, untuk setiap ukuran: setiap penyelesaian terhitung memenuhi persamaannya sampai digit terakhir. Tetapi galatnya tumbuh bersama bilangan kondisi persis seperti ramalan aturan praktis tadi: $\kappa = 3 \times 10^{7}$ untuk $n = 6$ dan galat 5×10^{-10} (sekitar 7 dari 16 digit hilang); $\kappa = 3.5 \times 10^{13}$ untuk $n = 10$ dan galat 3×10^{-4}; $\kappa = 3.8 \times 10^{16}$ untuk $n = 12$ dan galat 0,36, ketika "penyelesaian" itu tidak mempunyai satu digit benar pun. Program yang hanya memeriksa residu akan menyatakan keenam jawaban itu sempurna. (Bilangan kondisi untuk $n = 12$ itu sendiri hanya kira-kira benar, karena dihitung dari invers yang sama tidak telitinya.)

4.9 Mengapa *Pivoting* Penting, dalam Kode

Kode 4.11 dan **Kode 4.12** menjalankan kedua contoh Subbab 4.4 dengan dan tanpa *pivoting*.

Kode 4.11 `pivot_demo.py`: pivot nol dan pivot yang sangat kecil, dengan dan tanpa *partial pivoting*.

```python
"""Why pivoting matters: a zero pivot and a tiny pivot."""
from gauss import solve


def main():
    print("1. The receipts with rice missing from the first one")
    a = [[0, 1, 2], [5, 2, 1], [3, 2, 3]]
    b = [52, 128, 132]
    try:
        solve(a, b, pivot=False)
    except ZeroDivisionError as err:
        print(f"   without pivoting: {err}")
    x = solve(a, b, pivot=True)
    print("   with pivoting   : " + ", ".join(f"{v:.6f}" for v in x))

    print("2. A tiny pivot: 1e-20 x + y = 1, x + y = 2 (answer near 1, 1)")
    a = [[1e-20, 1], [1, 1]]
    b = [1, 2]
    x = solve(a, b, pivot=False)
    print(f"   without pivoting: x = {x[0]:.6f}, y = {x[1]:.6f}")
    x = solve(a, b, pivot=True)
    print(f"   with pivoting   : x = {x[0]:.6f}, y = {x[1]:.6f}")


if __name__ == "__main__":
    main()
```

Kode 4.12 `PivotDemo.java`: peragaan yang sama dalam Java.

```java
import java.util.Locale;

/** Why pivoting matters: a zero pivot and a tiny pivot. */
public class PivotDemo {
    public static void main(String[] args) {
        Locale.setDefault(Locale.ROOT);
        System.out.println("1. The receipts with rice missing from the first "
                           + "one");
        double[][] a = {{0, 1, 2}, {5, 2, 1}, {3, 2, 3}};
        double[] b = {52, 128, 132};
        try {
            Gauss.solve(a, b, false, false);
        } catch (ArithmeticException err) {
            System.out.println("   without pivoting: " + err.getMessage());
        }
        double[] x = Gauss.solve(a, b, true, false);
        System.out.printf("   with pivoting   : %.6f, %.6f, %.6f%n",
                          x[0], x[1], x[2]);

        System.out.println("2. A tiny pivot: 1e-20 x + y = 1, x + y = 2 "
                           + "(answer near 1, 1)");
        double[][] t = {{1e-20, 1}, {1, 1}};
        double[] c = {1, 2};
        x = Gauss.solve(t, c, false, false);
        System.out.printf("   without pivoting: x = %.6f, y = %.6f%n", x[0], x[1]);
        x = Gauss.solve(t, c, true, false);
        System.out.printf("   with pivoting   : x = %.6f, y = %.6f%n", x[0], x[1]);
    }
}
```

```
1. The receipts with rice missing from the first one
   without pivoting: zero pivot in column 1
   with pivoting   : 15.000000, 18.000000, 17.000000
2. A tiny pivot: 1e-20 x + y = 1, x + y = 2 (answer near 1, 1)
   without pivoting: x = 0.000000, y = 1.000000
   with pivoting   : x = 1.000000, y = 1.000000
```

Tanpa *pivoting*, nota-nota yang diurutkan ulang Persamaan (4.8) berhenti dengan "zero pivot in column 1", dan pivot yang sangat kecil Persamaan (4.9) memberikan $x = 0$ alih-alih 1 tanpa peringatan sama sekali, persis seperti ramalan hitungan tangan di Subbab 4.4.2. Dengan *pivoting*, keduanya terselesaikan dengan benar. Kasus kedua itulah yang berbahaya: metode yang gagal dengan keras dapat diperbaiki, sedangkan metode yang mengembalikan bilangan salah dengan penuh keyakinan bahkan tidak dapat disadari. Karena itu `solve` kita melakukan *pivoting* secara bawaan, dan hanya jejak pengajaran untuk nota-nota yang mematikannya.

4.10 Mengujinya

Kode 4.13 dan **Kode 4.14** memeriksa nota-nota, memeriksa bahwa matriks pemanggil tidak berubah, memeriksa bahwa *pivoting* menangani pivot nol dan bahwa matriks singular ditolak, serta memeriksa banyaknya flop untuk $n = 3$ terhadap Persamaan (4.6): $\frac{2}{3} \cdot 27 + \frac{3}{2} \cdot 9 - \frac{7}{6} \cdot 3 = 18 + 13.5 - 3.5 = 28$.

Kode 4.13 `test_gauss.py`: uji unit eliminasi Gauss.

```python
"""Tests of Gaussian elimination: python -m unittest test_gauss.py"""
import unittest
import gauss


class GaussTest(unittest.TestCase):
    def test_receipts(self):
        a = [[2, 1, 2], [5, 2, 1], [3, 2, 3]]
        b = [82, 128, 132]
        x = gauss.solve(a, b, pivot=False)
        for got, want in zip(x, (15, 18, 17)):
            self.assertAlmostEqual(got, want, places=12)
        self.assertEqual(a[0][0], 2)          # the input is not changed

    def test_zero_pivot_needs_pivoting(self):
        x = gauss.solve([[0, 1], [1, 1]], [1, 2])
        self.assertAlmostEqual(x[0], 1.0)
        self.assertAlmostEqual(x[1], 1.0)
        with self.assertRaises(ZeroDivisionError):
            gauss.solve([[0, 1], [1, 1]], [1, 2], pivot=False)

    def test_singular(self):
        with self.assertRaises(ZeroDivisionError):
            gauss.solve([[1, 2], [2, 4]], [3, 6])

    def test_flop_count(self):
        gauss.solve([[4, 1, 0], [1, 4, 1], [0, 1, 4]], [1, 1, 1])
        self.assertEqual(gauss.flops, 28)


if __name__ == "__main__":
    unittest.main()
```

Kode 4.14 `GaussTest.java`: pemeriksaan yang sama dalam Java.

```java
/** Tests of Gaussian elimination: java -cp out GaussTest */
public class GaussTest {
    static int failures = 0;

    static void check(boolean ok, String name) {
        System.out.println((ok ? "ok    " : "FAILED") + "  " + name);
        if (!ok) {
            failures++;
        }
    }

    public static void main(String[] args) {
        double[][] a = {{2, 1, 2}, {5, 2, 1}, {3, 2, 3}};
        double[] b = {82, 128, 132};
        double[] x = Gauss.solve(a, b, false, false);
        check(Math.abs(x[0] - 15) < 1e-12 && Math.abs(x[1] - 18) < 1e-12
              && Math.abs(x[2] - 17) < 1e-12, "receipts give 15, 18, 17");
        check(a[0][0] == 2 && b[0] == 82, "the input is not changed");
        double[][] z = {{0, 1}, {1, 1}};
        double[] y = Gauss.solve(z, new double[] {1, 2});
        check(Math.abs(y[0] - 1) < 1e-15 && Math.abs(y[1] - 1) < 1e-15,
              "pivoting handles a zero pivot");
        try {
            Gauss.solve(new double[][] {{1, 2}, {2, 4}}, new double[] {3, 6});
            check(false, "a singular matrix is refused");
        } catch (ArithmeticException e) {
            check(true, "a singular matrix is refused");
        }
        Gauss.solve(new double[][] {{4, 1, 0}, {1, 4, 1}, {0, 1, 4}},
                    new double[] {1, 1, 1});
        check(Gauss.flops == 28, "28 flops for n = 3");
        System.out.println(failures == 0 ? "all tests passed"
                                         : failures + " test(s) failed");
    }
}
```

```
$ python3 -m unittest test_gauss.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.000s

OK
$ java -cp out GaussTest
```

```
ok      receipts give 15, 18, 17
ok      the input is not changed
ok      pivoting handles a zero pivot
ok      a singular matrix is refused
ok      28 flops for n = 3
all tests passed
```

Uji matriks singular patut diberi catatan. Untuk $\begin{pmatrix}1 & 2\\ 2 & 4\end{pmatrix}$, *pivoting* menukar barisnya, pengalinya 0,5, dan pivot keduanya $2 - 0.5 \times 4 = 0$ *secara eksak*, sehingga program mendeteksinya. Untuk matriks singular yang entri-entrinya tidak dapat disajikan secara tepat, pembulatan biasanya meninggalkan pivot sekitar 10^{-16} alih-alih nol, program dengan senang hati membagi dengannya, dan "penyelesaian"-nya sangat besar. Uji nol eksak hanya menangkap kasus-kasus yang bersih; bilangan kondisi menangkap sisanya.

4.11 Membandingkan dengan NumPy

`np.linalg.solve` dari NumPy memanggil rutin LAPACK `gesv`, yang melakukan eliminasi Gauss dengan *partial pivoting* dalam bentuk faktorisasi LU (Bab 6) memakai operasi blok yang sangat

teroptimasi [5]. **Kode 4.15** menyelesaikan sistem pupuk dengannya, menghitung bilangan kondisi dalam dua norma, dan mengulang percobaan Hilbert dengan fungsi `hilbert` dari SciPy.

Kode 4.15 `compare_numpy.py`: sistem pupuk dan matriks Hilbert dengan NumPy dan SciPy.

```
"""The fertilizer system and the Hilbert matrices with NumPy (needs NumPy)."""
import numpy as np
import scipy.linalg

A = np.array([[0.15, 0.18, 0.00, 0.21],
              [0.15, 0.46, 0.00, 0.00],
              [0.15, 0.00, 0.60, 0.00],
              [0.10, 0.00, 0.00, 0.24]])
T = np.array([480.0, 490.0, 660.0, 320.0])


def main():
    x = np.linalg.solve(A, T)                    # LAPACK gesv: LU + pivoting
    print("np.linalg.solve :", np.array2string(x, precision=1))
    print(f"cond(A), 2-norm : {np.linalg.cond(A):.1f}")
    print(f"cond(A), inf    : {np.linalg.cond(A, np.inf):.1f}")
    for n in (6, 12):
        h = scipy.linalg.hilbert(n)
        xh = np.linalg.solve(h, h.sum(axis=1))
        print(f"Hilbert n = {n:>2}: cond {np.linalg.cond(h, np.inf):.2e}, "
              f"max error {np.max(np.abs(xh - 1)):.2e}")


if __name__ == "__main__":
    main()
```

```
np.linalg.solve : [2171.4  357.1  557.1  428.6]
cond(A), 2-norm : 265.7
cond(A), inf    : 446.8
Hilbert n =  6: cond 2.91e+07, max error 2.60e-10
Hilbert n = 12: cond 3.99e+16, max error 5.16e-01
```

Jumlah-jumlah dari NumPy cocok dengan milik kita sampai gram yang dicetak. Bilangan kondisinya dalam norma tak hingga, 446,8, adalah nilai yang dipakai di Subbab 4.8.2; dalam norma-2, bawaan `np.linalg.cond`, nilainya 265,7. Kedua norma mengukur besar dengan cara berbeda dan memberikan bilangan yang berbeda, tetapi selalu seorde. Untuk matriks Hilbert, galat NumPy (2.6×10^{-10} untuk $n = 6$, 0.52 untuk $n = 12$) seukuran dengan galat kita (5.3×10^{-10} dan 0.36), tidak lebih kecil: perangkat lunak profesional juga tidak dapat mengalahkan bilangan kondisi. Yang ditawarkannya adalah kecepatan, dan pengaman kestabilan yang penting untuk matriks besar.

4.12 Metode Numerik dalam Praktik: Profitina

AI Visibility Engine milik Profitina memberi skor pada sebuah situs web dari banyak sinyal, seperti seberapa mudah kontennya diekstraksi, seberapa lengkap data terstrukturnya, dan seberapa kokoh sisi teknisnya, lalu menjelaskan setiap sinyal. Skor yang dibangun dari banyak sinyal pada dasarnya adalah jumlah berbobot: setiap sinyal yang dinormalkan dikalikan bobotnya, lalu dijumlahkan. Platform itu tidak menerbitkan bobotnya, dan rincian cara menetapkannya adalah urusannya sendiri; tetapi bentuk matematis pertanyaan “bobot mana yang akan menghasilkan kembali penilaian-penilaian acuan ini?” adalah sebuah sistem linear, dengan satu persamaan per situs acuan dan satu peubah per bobot. Bila banyaknya acuan yang andal tepat sama dengan banyaknya bobot, sistem itu persegi seperti sistem-sistem di bab ini; bila lebih banyak, ia menjadi masalah kuadrat terkecil Bab 8.

Peringatan bab ini berlaku langsung. Dua sinyal yang mengukur hal yang hampir sama (misalnya, dua pemeriksaan berbeda yang keduanya terutama mendeteksi apakah sebuah halaman mempunyai blok data terstruktur) seperti kolom NPK dan DAP pada Persamaan (4.11): hampir sejajar. Sistem untuk bobot-bobotnya berkondisi buruk, dan perubahan sangat kecil pada data acuan dapat mengayunkan satu bobot ke atas dan yang lain ke bawah sementara skornya sendiri hampir tidak berubah. Gejalanya sama dengan di kebun: residu kecil, koefisien yang besar dan tidak stabil, dan kadang bobot negatif yang tidak masuk akal. Obatnya juga sama: ukur bilangan kondisinya, gabungkan atau buang sinyal yang saling menduplikasi, dan utamakan metode yang tetap stabil; semuanya praktik umum dalam sistem skor apa pun yang menggabungkan pengukuran.

4.13 Memakainya Sungguhan

Kedua program dapat disesuaikan untuk banyak masalah sehari-hari hanya dengan mengubah berkas datanya.

- **Nota-nota Anda sendiri.** Setiap himpunan n nota dengan n barang yang harganya tidak berubah dapat diselesaikan untuk harga-harganya. Pastikan nota-notanya sungguh berbeda: dua nota dengan proporsi yang sama (yang satu tepat dua kali yang lain) membuat matriksnya singular.
- **Pupuk-pupuk Anda sendiri.** Ganti baris-baris `fertilizer.csv` dengan pupuk yang dijual toko Anda, menurut persentase di karungnya, dan baris terakhir dengan sasaran dari uji tanah dan penyuluh pertanian Anda. Program ini bekerja untuk berapa pun banyaknya pupuk asalkan sama dengan banyaknya hara. Jika jawabannya memuat jumlah negatif, sasarannya tidak dapat dipenuhi secara tepat dengan pupuk-pupuk itu: tambahkan pupuk lain (misalnya urea, “46-0-0”, untuk nitrogen tambahan), atau terima sasaran yang dipenuhi secara hampiran, yaitu pendekatan kuadrat terkecil Bab 8.
- **Periksa kepekaannya.** Ubah setiap sasaran sebesar ketidakpastiannya, mungkin 5%, lalu selesaikan lagi. Jika jumlah-jumlahnya banyak berubah, pupuk-pupuknya terlalu tumpang tindih (bilangan kondisi yang besar); memilih pupuk yang profil haranya lebih berbeda, seperti pupuk tunggal, membuat sistemnya berkondisi baik dan rencananya kokoh.
- **Ransum dan campuran.** Program yang sama mencampur pakan ayam atau lele dari bahan-bahan dengan kandungan protein, energi, dan serat yang diketahui, mencampur dua mutu kopi atau beras ke harga dan mutu sasaran, atau mencari konsentrasi larutan dari beberapa pengukuran. Semuanya adalah $Ax = b$.
- **Ukuran.** Program kita memadai sampai beberapa ratus peubah. Untuk ribuan, pakailah NumPy (`np.linalg.solve`) atau pustaka aljabar linear Java, yang memakai metode yang sama dengan perulangan dalam yang jauh lebih cepat; untuk sistem jarang dengan jutaan peubah, lihat Bab 5.

4.14 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Eliminasi tanpa *pivoting*.** Gagal pada pivot nol dan, lebih buruk, mengembalikan jawaban salah tanpa peringatan pada pivot yang sangat kecil. Selalu pakai *partial pivoting* kecuali Anda tahu matriksnya tidak memerlukannya (misalnya matriks dominan diagonal).
- **Tertukar baris dan kolom.** Pada masalah pupuk, berkasnya berisi satu pupuk per baris, tetapi matriksnya memerlukan satu hara per baris. Tuliskan arti baris i dan kolom j sebelum membangun matriksnya, dan periksa satu persamaan dengan tangan.

- **Menilai penyelesaian dari residunya.** Eliminasi Gauss dengan *pivoting* selalu memberikan residu kecil. Apakah penyelesaiannya teliti bergantung pada bilangan kondisi; taksirlah, dan bersiaplah kehilangan sekitar $\log_{10}\kappa$ digit.
- **Menghitung invers untuk menyelesaikan sistem.** $x = A^{-1}b$ benar dalam aljabar, tetapi menghitung A^{-1} lebih dulu berbiaya tiga kali eliminasi dan kurang teliti. Selesaikan $Ax = b$ secara langsung; hitung invers hanya bila entri-entrinya sendiri diperlukan.
- **Memercayai determinan yang sangat kecil atau mengabaikan yang besar.** Determinan 10^{-3} kali matriks identitas adalah 10^{-3n}, sangat kecil untuk n besar, padahal matriks itu berkondisi sempurna. Pakai bilangan kondisi, bukan determinan, untuk menilai kepekaan.
- **Mengubah data milik pemanggil.** Eliminasi menimpa matriksnya. Salin dulu, seperti yang dilakukan `solve` kita, atau nyatakan dengan jelas bahwa matriks itu dirusak.
- **Terlalu percaya pada data.** Jawaban tidak mungkin lebih baik daripada sasaran dan persentase yang dimasukkan. Bila sistemnya berkondisi buruk, pembulatan kecil pada data menjadi perubahan besar pada jawaban.

4.15 Ringkasan Bab

- Sistem n persamaan linear dengan n peubah ditulis $Ax = b$ (Persamaan (4.2)); setiap persamaan adalah satu baris matriks diperbesar $[A \mid b]$.
- Eliminasi Gauss membuat matriksnya segitiga atas dengan pengali $m_{ik} = a_{ik}/a_{kk}$ (Persamaan (4.3)) dan pembaruan baris (Persamaan (4.4)), lalu mencari peubah-peubahnya dari yang terakhir ke yang pertama dengan substitusi mundur (Persamaan (4.5)); **Pseudokode 4.1** menggabungkan keduanya.
- Biayanya sekitar $\frac{2}{3}n^3$ flop (Persamaan (4.6)): menggandakan n melipatdelapankan pekerjaannya. Substitusi mundur hanya berbiaya n^2.
- Pivot nol menghentikan eliminasi biasa dan pivot yang sangat kecil diam-diam merusaknya (Persamaan (4.8), Persamaan (4.9)); *partial pivoting* (Persamaan (4.10)) menjaga setiap pengali paling besar 1 dan stabil dalam praktik. Determinan adalah hasil kali pivot-pivot beserta tandanya (Persamaan (4.7)).
- Bilangan kondisi $\kappa(A) = \| A \| \, \| A^{-1} \|$ (Persamaan (4.12)) membatasi seberapa besar perubahan relatif pada data diperbesar dalam penyelesaian (Persamaan (4.13)); sekitar $\log_{10}\kappa$ digit hilang. Residu kecil tidak berarti galat kecil.
- Nota-nota memberikan Rp15.000, Rp18.000, dan Rp17.000 secara tepat; keempat pupuk kebun menghasilkan 2.171, 357, 557, dan 429 g per pohon, sesuai dengan NumPy, tetapi sistemnya berkondisi agak buruk ($\kappa_\infty = 447$), sehingga sasaran yang dibulatkan cukup banyak mengubah rencananya.

4.16 Soal Latihan

1. Selesaikan dengan tangan, memakai eliminasi dan substitusi mundur, nota-nota berikut: 1 kg beras, 2 kg telur, dan 1 kg bawang merah seharga Rp111.000; 3 kg beras, 1 kg telur, dan 2 kg bawang merah seharga Rp153.000; 2 kg beras, 3 kg telur, dan 4 kg bawang merah seharga Rp274.000.
2. Selesaikan nota-nota yang diurutkan ulang Persamaan (4.8) dengan tangan memakai *partial pivoting*. Tuliskan setiap pertukaran dan setiap pengali, lalu periksa bahwa setiap pengali paling besar 1.

3. Hitung determinan matriks nota Soal Latihan 1 dari pivot-pivotnya dengan Persamaan (4.7), lalu periksa dengan rumus 3×3 yang biasa.
4. Dengan Persamaan (4.6), berapa flop yang diperlukan eliminasi Gauss untuk $n = 1{,}000$ dan untuk $n = 10{,}000$? Jika sebuah komputer melakukan 10^{10} flop per detik untuk pekerjaan semacam ini, berapa lama masing-masing?
5. Ubah sasaran fosfat di `fertilizer.csv` dari 490 menjadi 500 lalu selesaikan lagi. Kemudian kembalikan, dan ubah sasaran kalium dari 660 menjadi 700. Perubahan mana yang lebih banyak menggeser rencananya, dan mengapa?
6. Untuk matriks 2×2 $\begin{pmatrix} 1 & 1 \\ 1 & 1+\varepsilon \end{pmatrix}$, hitung κ_∞ sebagai fungsi ε. Seberapa kecil ε boleh menjadi sebelum penyelesaian dalam presisi ganda tidak mempunyai satu digit benar pun?
7. Jelaskan, dengan memakai hasil Hilbert di Subbab 4.8.3, mengapa sebuah program tidak boleh melaporkan "penyelesaian terverifikasi" hanya karena residunya kecil. Apa yang sebaiknya dilaporkannya?

4.17 Penyelesaian Soal Latihan

1. Matriks diperbesarnya (1,2,1 | 111), (3,1,2 | 153), (2,3,4 | 274), dalam ribuan rupiah. Dengan pivot 1 dan pengali 3 dan 2: baris 2 menjadi $(0, -5, -1 \mid 153 - 333 = -180)$ dan baris 3 menjadi $(0, -1, 2 \mid 274 - 222 = 52)$. Dengan pivot -5 dan pengali $(-1)/(-5) = 0.2$: baris 3 menjadi $(0{,}0{,}2 + 0.2 \mid 52 - 0.2 \times (-180)) = (0{,}0{,}2.2 \mid 88)$. Substitusi mundur: $x_3 = 88/2.2 = 40$; $-5x_2 - 40 = -180$, jadi $x_2 = 28$; $x_1 + 56 + 40 = 111$, jadi $x_1 = 15$. Beras Rp15.000, telur Rp28.000, bawang merah Rp40.000 per kg. (Tanpa *pivoting* program mengikuti persis langkah-langkah ini; dengan *pivoting* ia lebih dulu menukar baris 2 ke atas.)

2. Kolom 1 memuat 0,5,3: yang terbesar 5, di baris 2, jadi tukar baris 1 dan 2: (5,2,1 | 128), (0,1,2 | 52), (3,2,3 | 132). Pengali $0/5 = 0$ dan $3/5 = 0.6$: baris 3 menjadi $(0{,}2 - 1.2{,}3 - 0.6 \mid 132 - 76.8) = (0{,}0.8{,}2.4 \mid 55.2)$. Kolom 2 di bawah diagonal memuat 1 dan 0.8: yang terbesar sudah di tempatnya, tidak ada pertukaran. Pengali $0.8/1 = 0.8$: baris 3 menjadi $(0{,}0{,}2.4 - 1.6 \mid 55.2 - 41.6) = (0{,}0{,}0.8 \mid 13.6)$. Substitusi mundur: $x_3 = 17$, $x_2 = 52 - 34 = 18$, $5x_1 = 128 - 36 - 17 = 75$, $x_1 = 15$. Pengali-pengalinya 0; 0,6; dan 0,8, semuanya paling besar 1.

3. Tanpa pertukaran, pivot 1, -5, dan 2.2: $\det = 1 \times (-5) \times 2.2 = -11$. Rumus 3×3: $1(1 \cdot 4 - 2 \cdot 3) - 2(3 \cdot 4 - 2 \cdot 2) + 1(3 \cdot 3 - 1 \cdot 2) = -2 - 16 + 7 = -11$.

4. Untuk $n = 1{,}000$: $\frac{2}{3} \times 10^9 + 1.5 \times 10^6 - 1{,}167 \approx 6.68 \times 10^8$ flop, sekitar 0,07 detik pada 10^{10} flop per detik. Untuk $n = 10{,}000$: $\approx 6.67 \times 10^{11}$ flop, sekitar 67 detik, seribu kali lebih lama untuk ukuran sepuluh kali. Matriksnya sendiri lalu memerlukan 10^8 *double*, 800 MB memori, yang sering menjadi batas yang sebenarnya.

5. Fosfat 500 (sisanya tidak berubah) memberikan NPK 1.142,9 g, DAP 714,3 g, KCl 814,3 g, dan ZA 857,1 g: setiap jumlah berubah, NPK sebesar -47% dan DAP sebesar $+100\%$. Kalium 700 memberikan NPK 2.171,4 g, DAP 357,1 g, KCl 623,8 g, dan ZA 428,6 g: hanya KCl yang berubah, sebesar $40/0.60 = 66.7$ g. Kalium hampir hanya berasal dari KCl (NPK juga mengandung sedikit, tetapi jumlah NPK ditetapkan oleh tiga persamaan lainnya), sehingga persamaannya terpisah dengan baik dari yang lain. Fosfat berasal dari NPK dan DAP, yang juga berbagi nitrogen, dan kedua kolom itu hampir sejajar: bagian sistem yang berkondisi buruk.

6. $\det = \varepsilon$, $A^{-1} = \frac{1}{\varepsilon}\begin{pmatrix} 1+\varepsilon & -1 \\ -1 & 1 \end{pmatrix}$, sehingga $\| A \|_\infty = 2 + \varepsilon$ dan $\| A^{-1} \|_\infty = (2+\varepsilon)/\varepsilon$, memberikan $\kappa_\infty = (2+\varepsilon)^2/\varepsilon \approx 4/\varepsilon$. Dengan sekitar 16 digit yang tersedia, semuanya hilang ketika $4/\varepsilon \approx$

10^{16}, yaitu $\varepsilon \approx 4 \times 10^{-16}$; di bawah sekitar 2.2×10^{-16}, entri matriks $1 + \varepsilon$ bahkan tidak dapat disajikan dan dibulatkan menjadi 1, sehingga matriks yang tersimpan menjadi singular.

7. Untuk $n = 12$ residunya 4.3×10^{-16}, sekecil untuk $n = 2$, tetapi galatnya 0,36: penyelesaian yang "terverifikasi" itu tidak mempunyai satu digit benar pun. Residu kecil hanya menunjukkan bahwa $\tilde{x}$ menyelesaikan sebuah sistem yang *sangat dekat* dengan sistem yang diberikan; jika sistemnya berkondisi buruk, sistem-sistem yang sangat dekat mempunyai penyelesaian yang sangat berbeda. Program yang bertanggung jawab melaporkan residu bersama taksiran bilangan kondisinya, dan dari keduanya taksiran banyaknya digit benar, $16 - \log_{10}\kappa$, sambil memberi peringatan bila nilainya kecil. NumPy menyediakan `np.linalg.cond` untuk ini; LAPACK mempunyai rutin yang menaksir κ dengan murah selama penyelesaian.

Daftar Pustaka

[1] J. F. Grcar, "How ordinary elimination became Gaussian elimination," *Historia Mathematica*, vol. 38, no. 2, pp. 163–218, 2011.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins University Press, 2013.

[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[4] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[5] NumPy Developers, "numpy.linalg.solve," *NumPy Reference*, 2026. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html

Bab 5

Sistem Linear II: Jacobi, Gauss-Seidel, dan SOR

Bab 4 menyelesaikan sistem linear seperti seorang juru buku yang teliti mengerjakan buku besarnya: eliminasi, eliminasi, substitusi balik, selesai, dengan banyak langkah yang sudah pasti sejak awal. Itulah alat yang tepat untuk beberapa ratus variabel. Namun banyak sistem nyata memiliki puluhan ribu hingga jutaan variabel, dan pada hampir semuanya setiap persamaan hanya menyebut segelintir variabel. Untuk sistem seperti itu, gagasan yang sama sekali lain justru unggul: tebak sebuah jawaban, lalu perbaiki setiap variabel secara bergiliran dari tetangga-tetangganya, dan ulangi sampai tidak ada yang bergerak lagi. Bab ini membangun gagasan tersebut dari sebuah wajan datar besi di gerobak martabak. Di akhir bab ini Anda akan mampu mengubah aturan fisika "setiap titik adalah rata-rata tetangganya" menjadi sistem linear; mengerjakan sapuan (*sweep*) Jacobi, Gauss-Seidel, dan *successive over-relaxation* (SOR) dengan tangan; menuliskannya sebagai pemisahan matriks (*matrix splitting*) dan membuktikan kapan metode itu konvergen; meramalkan berapa sapuan yang diperlukan dari jari-jari spektral; memilih faktor relaksasi ω; berhenti pada saat yang tepat sambil mengetahui seberapa jauh Anda masih dari jawaban; membangun ketiga metode dalam Java dan Python; serta memeriksa hasilnya dengan pemecah matriks jarang SciPy.

5.1 Wajan Pak Joni

Setiap sore pukul lima, Pak Joni menyalakan kompor di bawah wajan datar besi gerobak martabak dan roti bakarnya. Wajan itu berupa pelat besi tebal berbentuk persegi, empat puluh sentimeter setiap sisinya. Sebuah kompor panjang terpasang di bawah tepi kirinya; kompor kedua yang lebih kecil menjaga tepi depan tetap hangat agar martabak yang sudah matang bisa menunggu di sana; tepi belakang, di dekat kaleng-kaleng margarin, dan tepi kanan, tempat ia memotong roti, sama sekali tidak terkena api. Setelah seperempat jam, pelat itu berhenti bertambah panas. Pelat telah mencapai *keadaan tunak* (*steady state*): panas mengalir masuk dari kompor dan keluar lewat tepi-tepi yang dingin dengan laju yang sama, dan suhu di setiap titik tidak berubah lagi.

Pak Joni mengenal wajannya seperti sopir mengenal jalan. Martabak telur memerlukan tempat yang panas, tetapi bukan yang paling panas, karena telurnya akan gosong sebelum kulitnya renyah; roti bakar memerlukan tempat yang lebih lembut; pojok kanan jauh hanya cocok untuk menjaga makanan tetap hangat. Ia menemukan tempat-tempat itu selama bertahun-tahun, satu martabak gosong demi satu martabak gosong. Keponakannya, yang baru mulai membantu, terus-menerus menaruh adonan di tempat yang salah, dan wajan baru dengan kompor yang berbeda berarti harus belajar semuanya dari awal.

Yang ia inginkan adalah sebuah peta: suhu di setiap titik pelat, dengan zona antara 140 dan 170 derajat Celsius, tempat martabak telur paling baik dimasak, ditandai di atasnya. Termometer inframerah dapat mengukur beberapa titik, tetapi peta memerlukan ratusan titik, dan mengukur wajan baru sebelum membelinya sama sekali tidak mungkin. Bab ini menghitung peta itu dari empat angka: suhu di sepanjang keempat tepi. Pak Joni mengukurnya dengan termometer

inframerah pinjaman: sekitar 200 °C di atas kompor utama di kiri, 150 °C di sepanjang tepi depan, 80 °C di belakang, dan 60 °C di kanan.

Fisika di balik peta ini sudah tua dan sederhana, dan ia menghasilkan sistem linear yang persis sejenis dengan yang diajarkan Bab 4, hanya saja besar. Peta yang berguna untuk pelat 40 cm memerlukan satu titik setiap satu atau dua sentimeter, artinya 400 hingga 1600 variabel; model teknik yang teliti untuk pelat yang sama akan memakai satu juta. Namun setiap persamaan hanya melibatkan lima variabel. Kombinasi inilah, banyak variabel dan sangat sedikit per persamaan, yang menjadi sasaran metode-metode dalam bab ini.

5.2 Gagasannya dalam Kata-Kata Sederhana

5.2.1 Setiap titik adalah rata-rata tetangganya

Bayangkan pelat itu dibagi oleh garis-garis kisi menjadi persegi-persegi kecil, lalu perhatikan satu titik tempat garis-garis itu bersilangan. Dalam keadaan tunak, panas tidak menumpuk di titik itu: yang mengalir masuk dari tetangga yang lebih panas mengalir keluar ke tetangga yang lebih dingin. Untuk pelat tipis yang seragam, keseimbangan ini membawa akibat yang indah: *suhu di setiap titik dalam sama dengan rata-rata suhu keempat tetangganya*, kiri, kanan, atas, dan bawah. Jika tetangganya bersuhu 120, 100, 90, dan 110 °C, titik itu sendiri bersuhu 105 °C.

Mengapa rata-rata? Panas mengalir dari yang panas ke yang dingin dengan laju sebanding dengan selisih suhunya (hukum Fourier). Panas yang mengalir dari tetangga kiri ke titik kita sebanding dengan $T_{\text{kiri}} - T$, dan demikian pula untuk tiga tetangga lainnya. Dalam keadaan tunak keempat aliran itu berjumlah nol:

$$(T_{\text{kiri}} - T) + (T_{\text{kanan}} - T) + (T_{\text{atas}} - T) + (T_{\text{bawah}} - T) = 0,$$

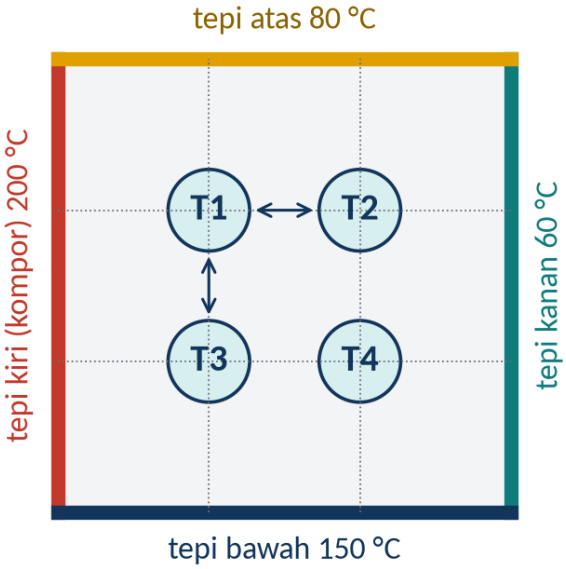
dan bila diselesaikan untuk T diperoleh

$$T = \frac{T_{\text{kiri}} + T_{\text{kanan}} + T_{\text{atas}} + T_{\text{bawah}}}{4}. \tag{5.1}$$

Persamaan yang sama muncul dari persamaan panas dalam fisika, persamaan Laplace $\partial^2 T/\,\partial x^2 + \partial^2 T/\,\partial y^2 = 0$, bila setiap turunan kedua diganti dengan selisih pusatnya, yang diturunkan di Bab 9: $\partial^2 T/\,\partial x^2 \approx (T_{\text{kiri}} - 2T + T_{\text{kanan}})/h^2$ untuk jarak kisi h [1]. Galat penggantian ini mengecil sebanding dengan h^2, sehingga kisi yang lebih halus memberi peta yang lebih setia. Persamaan (5.1) adalah seluruh fisika yang kita perlukan. Persamaan itu mengabaikan panas yang keluar lewat permukaan atas pelat ke udara, jadi ia model pertama, bukan kata terakhir; Subbab 5.9 menjelaskan cara memeriksanya dengan termometer.

5.2.2 Empat titik dengan tangan

Mulailah dengan kisi paling kasar yang masih memiliki isi: dua garis melintang dan dua garis membujur, yang menyisakan empat titik dalam. **Gambar 5.1** menamainya T_1 (kiri atas), T_2 (kanan atas), T_3 (kiri bawah), dan T_4 (kanan bawah). Setiap titik dalam menyentuh dua tepi dan dua titik dalam lainnya.



Gambar 5.1 Wajan paling kasar: empat suhu titik dalam yang belum diketahui, masing-masing rata-rata keempat tetangganya; dua tetangga selalu berupa tepi yang suhunya diketahui.

Tuliskan Persamaan (5.1) untuk T_1. Tetangganya adalah tepi kiri (200), tepi atas (80), T_2, dan T_3, sehingga $4T_1 = 200 + 80 + T_2 + T_3$. Memindahkan variabel ke ruas kiri dan melakukan hal yang sama untuk tiga titik lainnya memberi empat persamaan:

$$\begin{aligned} 4T_1 - T_2 - T_3 &= 280 \\ -T_1 + 4T_2 - T_4 &= 140 \\ -T_1 + 4T_3 - T_4 &= 350 \\ -T_2 - T_3 + 4T_4 &= 210 \end{aligned} \tag{5.2}$$

Ruas kanannya adalah jumlah suhu tepi: 280 adalah kiri ditambah atas, 140 kanan ditambah atas, 350 kiri ditambah bawah, dan 210 kanan ditambah bawah. Eliminasi Gauss dari Bab 4 menyelesaikan Persamaan (5.2) secara eksak: $T_1 = 131.25$, $T_2 = 96.25$, $T_3 = 148.75$, $T_4 = 113.75$. Sekarang kita akan mencapai jawaban yang sama dengan cara yang sama sekali berbeda, dan melihat mengapa cara itu dapat diperbesar hingga sejuta titik sementara eliminasi tidak.

5.2.3 Jacobi: semua memperbarui dari gambaran lama

Tebaklah bahwa seluruh bagian dalam pelat bersuhu 0 °C. Tebakan itu buruk sekali, tetapi tidak apa-apa. Sekarang terapkan Persamaan (5.1) pada setiap titik dalam, dengan memakai tebakan *lama* untuk setiap tetangga:

- $T_1 = (280 + T_2 + T_3)/4 = (280 + 0 + 0)/4 = 70$
- $T_2 = (140 + T_1 + T_4)/4 = 140/4 = 35$
- $T_3 = (350 + T_1 + T_4)/4 = 350/4 = 87.5$
- $T_4 = (210 + T_2 + T_3)/4 = 210/4 = 52.5$

Itulah satu *sapuan*. Gambaran baru (70, 35, 87.5, 52.5) masih salah, karena tetangga yang kita pakai salah, tetapi kesalahannya berkurang: tepi-tepi mulai "merembeskan" panasnya ke dalam. Lakukan sekali lagi, kini dengan angka-angka sapuan pertama sebagai gambaran lama:

- $T_1 = (280 + 35 + 87.5)/4 = 402.5/4 = 100.625$
- $T_2 = (140 + 70 + 52.5)/4 = 262.5/4 = 65.625$
- $T_3 = (350 + 70 + 52.5)/4 = 472.5/4 = 118.125$
- $T_4 = (210 + 35 + 87.5)/4 = 332.5/4 = 83.125$

Inilah **metode Jacobi**, dinamai menurut Carl Gustav Jacobi yang memakainya pada 1845. Aturannya mudah diucapkan: pada setiap sapuan, setiap variabel dihitung ulang dari persamaannya sendiri, dan setiap variabel lain dalam persamaan itu mengambil nilainya dari sapuan sebelumnya. Perhatikan seberapa jauh setiap nilai bergeser: pada sapuan pertama pergeseran terbesar 87.5, pada sapuan kedua 30.625. Program di Subbab 5.5 melanjutkan tabelnya; pergeserannya 87.5, 30.6, 15.3, 7.7, 3.8, 1.9, masing-masing separuh dari yang sebelumnya setelah dua yang pertama. Setelah 27 sapuan tidak ada nilai yang bergeser sebesar sepersejuta derajat pun, dan keempat suhu itu cocok dengan nilai eksaknya sampai enam desimal.

5.2.4 Gauss-Seidel: langsung pakai angka terbaru

Ada yang boros dalam sapuan Jacobi. Ketika menghitung T_2 kita sudah mengetahui T_1 yang baru (70), tetapi kita memakai yang lama (0). Carl Friedrich Gauss tidak menyia-nyiakan informasi seperti itu dalam perhitungan tangannya sendiri, dan Philipp Ludwig von Seidel menuliskan metodenya pada 1874. **Metode Gauss-Seidel** adalah Jacobi dengan satu perubahan: setiap nilai baru langsung dipakai begitu selesai dihitung. Dari awal yang sama, semuanya nol:

- $T_1 = (280 + 0 + 0)/4 = 70$
- $T_2 = (140 + \mathbf{70} + 0)/4 = 210/4 = 52.5$
- $T_3 = (350 + \mathbf{70} + 0)/4 = 420/4 = 105$
- $T_4 = (210 + \mathbf{52.5} + \mathbf{105})/4 = 367.5/4 = 91.875$

Angka yang dicetak tebal adalah angka yang baru. Sapuan kedua menghasilkan 109.375, 85.3125, 137.8125, dan 108.28125, sudah lebih dekat ke jawaban daripada sapuan kedua Jacobi. Pergeserannya kini mengecil empat kali lipat per sapuan, bukan dua kali, dan 15 sapuan sudah cukup sementara Jacobi memerlukan 27. Ada bonus praktis: Gauss-Seidel menimpa setiap nilai di tempatnya dan hanya memerlukan satu salinan variabel, sedangkan Jacobi memerlukan dua, gambaran lama dan gambaran baru.

5.2.5 SOR: melangkahlah sedikit lebih jauh

Perhatikan nilai-nilai T_1 pada Gauss-Seidel: 70, 109.4, 125.8, 129.9, 130.9, 131.2. Setiap langkah menuju arah yang sama, naik menuju 131.25, dan setiap langkah berhenti sebelum sampai. Bila koreksi selalu kurang jauh ke arah yang sama, perbaikan yang wajar adalah mengambil langkah yang lebih besar. **Successive over-relaxation** (SOR) menghitung nilai Gauss-Seidel lalu melampauinya: bila nilai lama x_{lama} dan Gauss-Seidel mengusulkan x_{GS}, SOR mengambil

$$x_{\text{baru}} = x_{\text{lama}} + \omega\,(x_{\text{GS}} - x_{\text{lama}}),$$

dengan ω (omega) sebagai *faktor relaksasi*. Dengan $\omega = 1$ ini adalah Gauss-Seidel; dengan $\omega = 1.1$ setiap langkah 10% lebih panjang. Dengan tangan, pada sapuan SOR pertama dengan $\omega = 1.1$, T_1 bergerak dari 0 ke $0 + 1.1 \times (70 - 0) = 77$, dan T_2 memakai 77 ini: usulan Gauss-Seidel adalah $(140 + 77 + 0)/4 = 54.25$, dan SOR mengambil $1.1 \times 54.25 = 59.675$. Dengan $\omega = 1.1$ wajan empat titik konvergen dalam 10 sapuan. Dengan $\omega = 1.5$ diperlukan 29, lebih banyak daripada Gauss-Seidel: melangkah terlalu jauh berarti melewati sasaran, dan nilai-nilainya lalu berayun bolak-balik di sekitar jawaban. Ada ω terbaik, dan Subbab 5.4 menunjukkan cara menemukannya. Untuk wajan empat titik nilainya 1.072; untuk wajan 400 titik nilainya 1.741 dan memangkas kerja dari 683 sapuan menjadi 80.

> **WAWASAN KUNCI:** Metode iteratif tidak pernah menyelesaikan sistem sekaligus. Ia menyimpan tebakan lengkap atas jawaban dan memperbaikinya sapuan demi sapuan, dan setiap sapuan kira-kira semahal memeriksa persamaan sekali. Apakah itu sepadan bergantung pada dua pertanyaan: apakah tebakan menjadi lebih baik setiap kali, dan seberapa cepat? Sisa bab ini menjawab keduanya.

5.3 Metodenya secara Tepat

5.3.1 Sistem linear umum

Misalkan A matriks $n \times n$ dengan entri a_{ij}, b vektor berisi n bilangan, dan x vektor n variabel dalam $Ax = b$. Persamaan ke-i berbunyi $\sum_{j=1}^{n} a_{ij}\, x_j = b_i$. Bila entri diagonal a_{ii} tidak nol, persamaan ke-i dapat diselesaikan untuk variabel "miliknya sendiri" x_i:

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}\, x_j \right). \tag{5.3}$$

Untuk wajan, $a_{ii} = 4$ dan entri di luar diagonal bernilai -1 untuk tetangga, sehingga Persamaan (5.3) persis aturan rata-rata Persamaan (5.1). Ketiga metode menerapkan Persamaan (5.3) untuk $i = 1,2, \ldots, n$ secara bergiliran; bedanya hanya pada nilai variabel lain yang dimasukkan. Tulislah $x^{(k)}$ untuk tebakan setelah k sapuan (pangkat dalam kurung adalah penghitung sapuan, bukan perpangkatan), dengan $x^{(0)}$ sebagai tebakan awal.

Jacobi hanya memakai sapuan sebelumnya:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j \neq i} a_{ij}\, x_j^{(k)} \right), \qquad i = 1, \ldots, n. \tag{5.4}$$

Gauss-Seidel memakai nilai baru untuk variabel yang sudah diperbarui pada sapuan ini ($j < i$) dan nilai lama untuk sisanya ($j > i$):

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)} \right). \tag{5.5}$$

SOR mencampur nilai lama dengan nilai Gauss-Seidel melalui faktor ω:

$$x_i^{(k+1)} = (1-\omega)\,x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}\,x_j^{(k+1)} - \sum_{j>i} a_{ij}\,x_j^{(k)}\right). \tag{5.6}$$

Pseudokode 5.1 menuliskan seluruh pemecah SOR dalam pseudokode; dengan $\omega = 1$ ia menjadi Gauss-Seidel. Perhatikan baris 5: nilai baru menimpa nilai lama dalam larik yang sama, dan itulah yang membuat jumlah pada baris 4 otomatis memakai nilai baru untuk $j < i$.

Pseudokode 5.1 *Successive over-relaxation*; dengan omega = 1 ia adalah metode Gauss-Seidel.

```
SOR(A, b, x, omega, tol, maxSweeps)
 1  for k = 1 to maxSweeps
 2      change = 0
 3      for i = 1 to n
 4          s = b[i] - (jumlah untuk j != i dari A[i][j] * x[j])
 5          new = (1 - omega) * x[i] + omega * s / A[i][i]
 6          change = max(change, |new - x[i]|)
 7          x[i] = new
 8      if change < tol
 9          return x, k
10  return x, maxSweeps        // belum konvergen: laporkan
```

5.3.2 Pemisahan matriks

Ketiga metode itu tampak seperti tiga resep, tetapi sebenarnya satu gagasan. Pisahkan A menjadi bagian diagonalnya D, bagian bawahnya yang sejati, dan bagian atasnya yang sejati. Lazimnya bagian bawah dan atas ditulis dengan tanda minus, $A = D - L - U$, dengan $-L$ memuat entri di bawah diagonal dan $-U$ entri di atasnya. Untuk wajan, $D = 4I$ dan L serta U berisi angka satu di posisi tetangga. Maka $Ax = b$ menjadi $Dx = (L+U)x + b$, dan Persamaan (5.4) adalah persamaan ini dengan x baru di kiri dan yang lama di kanan. Gauss-Seidel juga memindahkan bagian bawah ke kiri, karena variabel-variabel itu sudah baru, dan SOR melakukan hal yang sama setelah penskalaan dengan ω:

$$\begin{aligned} \text{Jacobi:} \quad & D\,x^{(k+1)} = (L+U)\,x^{(k)} + b \\ \text{Gauss-Seidel:} \quad & (D-L)\,x^{(k+1)} = U\,x^{(k)} + b \\ \text{SOR:} \quad & (D-\omega L)\,x^{(k+1)} = ((1-\omega)D + \omega U)\,x^{(k)} + \omega b \end{aligned} \tag{5.7}$$

Setiap baris berbentuk $Mx^{(k+1)} = Nx^{(k)} + c$ dengan $A = M - N$ (untuk SOR, sampai faktor ω). Pemisahan yang baik membuat M mudah "dibalik": D diagonal, $D - L$ dan $D - \omega L$ segitiga bawah, sehingga menyelesaikan dengannya hanyalah substitusi maju dari Bab 4, dan itulah yang dikerjakan sapuan baris demi baris. Mengalikan dengan M^{-1} memberi bentuk baku *iterasi stasioner*,

$$x^{(k+1)} = T\,x^{(k)} + c, \qquad T = M^{-1}N, \tag{5.8}$$

dengan *matriks iterasi* $T_J = D^{-1}(L+U)$ untuk Jacobi, $T_{GS} = (D-L)^{-1}U$ untuk Gauss-Seidel, dan $T_\omega = (D-\omega L)^{-1}((1-\omega)D + \omega U)$ untuk SOR [2]. Tidak ada yang menghitung matriks-matriks ini di dalam program. Mereka ada untuk menjawab pertanyaan-pertanyaan subbab berikut.

5.4 Kapan dan Seberapa Cepat Metode Ini Konvergen

5.4.1 Galat mengikuti iterasi yang sama

Misalkan x solusi eksak. Solusi itu juga memenuhi iterasi, $x = Tx + c$, karena itu hanyalah $Ax = b$ yang disusun ulang. Kurangkan dari Persamaan (5.8). Konstanta c saling meniadakan, dan galat $e^{(k)} = x^{(k)} - x$ memenuhi

$$e^{(k+1)} = T\,e^{(k)}, \qquad \text{sehingga} \qquad e^{(k)} = T^k e^{(0)}. \tag{5.9}$$

Segalanya ada di baris ini. Tebakan awal hanya masuk lewat $e^{(0)}$; sesudah itu galat dikalikan dengan matriks T yang sama pada setiap sapuan. Iterasi konvergen dari setiap tebakan awal tepat bila pangkat-pangkat T^k menyusut menuju matriks nol.

Untuk melihat kapan hal itu terjadi, andaikan sejenak bahwa T memiliki n vektor eigen bebas $v_1, \dots, v_n$ dengan nilai eigen $\lambda_1, \dots, \lambda_n$ (jadi $Tv_i = \lambda_i v_i$; Bab 12 membahas cara menghitungnya). Tuliskan galat pertama sebagai kombinasi $e^{(0)} = c_1 v_1 + \cdots + c_n v_n$. Maka $T^k e^{(0)} = c_1 \lambda_1^k v_1 + \cdots + c_n \lambda_n^k v_n$. Setiap suku menyusut bila $|\lambda_i| < 1$ dan bertahan atau membesar bila tidak. Yang menentukan adalah $|\lambda_i|$ terbesar. Besaran ini disebut **jari-jari spektral** (*spectral radius*),

$$\rho(T) = \max_i |\lambda_i|. \tag{5.10}$$

> **DEFINISI.** Iterasi $x^{(k+1)} = Tx^{(k)} + c$ *konvergen* bila $x^{(k)}$ mendekati solusi untuk setiap tebakan awal $x^{(0)}$. **Teorema** [2]: iterasi itu konvergen jika dan hanya jika $\rho(T) < 1$. Bila konvergen, galat menyusut kira-kira dengan faktor $\rho(T)$ per sapuan, dan $-\log_{10}\rho(T)$, yaitu banyaknya digit desimal benar yang diperoleh per sapuan, disebut *laju konvergensi*.

Teorema ini juga berlaku untuk matriks yang tidak memiliki himpunan vektor eigen lengkap; pembuktiannya memakai bentuk Jordan [2], [3]. Kasus itu membawa satu peringatan: galat dapat membesar sebentar sebelum faktor ρ mengambil alih, dan kita akan melihatnya pada SOR di Subbab 5.6.

Periksalah pada wajan empat titik. Matriks Jacobinya $T_J = 1/4\,(L + U)$, dengan nilai eigen $1/2\,, 0, 0, -1/2$ (Soal 2 menemukannya). Jadi $\rho(T_J) = 0.5$, dan galat seharusnya menjadi separuh pada setiap sapuan. Tabel Jacobi di Subbab 5.6 menunjukkan persis hal itu: pergeseran 7.66, 3.83, 1.91 menjadi separuh, demikian pula jaraknya ke 131.25 (juga 7.66, 3.83, 1.91). Untuk Gauss-Seidel, $\rho(T_{GS}) = 0.25$, dan pergeserannya menyusut empat kali lipat.

5.4.2 Uji yang bisa dilakukan dengan mata: dominansi diagonal

Nilai eigen mahal dihitung, jadi kita menginginkan uji yang bisa dibaca langsung dari matriks. Sebuah matriks **dominan diagonal sejati menurut baris** (*strictly diagonally dominant*) bila di setiap baris entri diagonalnya melebihi gabungan semua entri lain:

$$|a_{ii}| > \sum_{j \neq i} |\,a_{ij}\,| \qquad \text{untuk setiap } i. \tag{5.11}$$

Teorema [2]: bila A dominan diagonal sejati, Jacobi maupun Gauss-Seidel konvergen dari setiap tebakan awal. Bukti untuk Jacobi cukup tiga baris. Baris ke-i dari T_J memiliki entri $-a_{ij}/a_{ii}$ untuk $j \neq i$ dan nol di diagonal, sehingga jumlah nilai mutlak baris terbesar dari T_J adalah $q = \max_i \sum_{j \neq i} |\,a_{ij}|/|a_{ii}\,|$, dan Persamaan (5.11) menyatakan $q < 1$. Untuk sembarang vektor e,

setiap komponen $T_J e$ paling besar q kali komponen terbesar e dalam nilai mutlak. Jadi komponen galat terbesar menyusut setidaknya dengan faktor q pada setiap sapuan, dan galat menuju nol.

Setiap baris sistem empat titik memenuhi $|4| > |-1| + |-1| = 2$, jadi sistem itu dominan diagonal sejati dan teorema menjamin konvergensi ($q = 0.5$, di sini sama dengan ρ). Titik dalam pada wajan yang lebih besar memiliki empat tetangga dan $|4| = 4$, yang tidak sejati. Teorema tetap berlaku untuk matriks seperti itu bila matriksnya dominan diagonal secara *takteruraikan* (*irreducibly*), artinya setiap titik terhubung ke tepi dan setidaknya satu baris sejati; setiap baris di sebelah tepi memang sejati. Matriks wajan juga simetris definit positif, dan untuk matriks seperti itu teorema lain (Ostrowski dan Reich) menjamin bahwa Gauss-Seidel dan SOR dengan $0 < \omega < 2$ konvergen [2], [4]. Di luar $0 < \omega < 2$ SOR tidak pernah konvergen, karena teorema Kahan memberi $\rho(T_\omega) \geq |\omega - 1|$ untuk setiap matriks [5].

> **PERINGATAN:** Dominansi diagonal bergantung pada *urutan* persamaan ditulis. Keempat persamaan wajan yang sama, ditulis dalam urutan 2, 1, 4, 3, menaruh -1 di diagonal baris pertama dan 4 di sebelahnya. Solusinya sama sekali tidak berubah, tetapi Jacobi pada sistem yang diurutkan ulang itu divergen: setelah 16 sapuan nilainya bergeser lebih dari 10^{12}, seperti ditunjukkan keluaran di Subbab 5.6. Sebelum beriterasi, tempatkan di posisi diagonal setiap baris variabel yang koefisiennya terbesar.

5.4.3 Masalah model: jari-jari spektral eksak

Untuk wajan dengan $n \times n$ titik dalam, nilai eigen T_J diketahui dalam bentuk tertutup [2], [5]. Vektor eigennya berupa gelombang sinus melintasi pelat, dan nilai eigennya

$$\mu_{p,q} = 1/2\left(\cos\frac{p\pi}{n+1} + \cos\frac{q\pi}{n+1}\right), \qquad p, q = 1, \ldots, n,$$

sehingga yang terbesar dalam nilai mutlak adalah gelombang paling halus, $p = q = 1$:

$$\rho_J = \cos\frac{\pi}{n+1}. \tag{5.12}$$

Untuk $n = 2$ nilainya $\cos(\pi/3) = 0.5$, yaitu wajan empat titik. Untuk $n = 20$ nilainya $\cos(\pi/21) = 0.98883$ dan untuk $n = 40$ nilainya 0.99707. Makin halus kisi, makin dekat ρ_J merayap ke 1 dan makin lambat Jacobi. Biang keladinya adalah gelombang paling halus: aturan rata-rata hanya memindahkan panas satu langkah kisi per sapuan, sehingga gelombang yang membentang di seluruh pelat memerlukan banyak sapuan untuk dikoreksi.

Untuk matriks dengan struktur seperti wajan (“terurut secara konsisten”, dalam istilah disertasi David Young tahun 1950 [4]), Young membuktikan tiga hubungan yang membuat SOR dapat diramalkan:

$$\rho_{GS} = \rho_J^2, \qquad \omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho_J^2}}, \qquad \rho(T_{\omega_{\text{opt}}}) = \omega_{\text{opt}} - 1. \tag{5.13}$$

Jadi Gauss-Seidel tepat dua kali lebih cepat daripada Jacobi pada wajan: mengkuadratkan ρ menggandakan digit yang diperoleh per sapuan. Dengan Persamaan (5.12), $\sqrt{1 - \rho_J^2} = \sin(\pi/(n+1))$ dan faktor terbaiknya $\omega_{\text{opt}} = 2/(1 + \sin(\pi/(n+1)))$. Untuk $n = 2$ nilainya $2/(1 + 0.8660) = 1.0718$; untuk $n = 20$ nilainya 1.741 dan untuk $n = 40$ nilainya 1.858.

5.4.4 Berapa sapuan?

Bila galat menyusut dengan faktor ρ per sapuan, memperkecilnya dari ukuran awal E_0 sampai toleransi ε memerlukan k sapuan dengan $\rho^k E_0 \approx \varepsilon$, yaitu

$$k \approx \frac{\ln(\varepsilon/E_0)}{\ln\rho}. \tag{5.14}$$

Untuk ρ yang dekat dengan 1, $\ln\rho \approx -(1-\rho)$, sehingga banyaknya sapuan tumbuh seperti $1/(1-\rho)$. Dengan $\rho_J = \cos(\pi/(n+1)) \approx 1 - \pi^2/(2(n+1)^2)$, Jacobi memerlukan banyak sapuan yang sebanding dengan n^2: menggandakan kisi di setiap arah melipatempatkan sapuannya. Gauss-Seidel memangkas jumlah itu menjadi separuh tetapi pertumbuhannya tetap n^2. SOR optimal memiliki $1-\rho = 2-\omega_{\text{opt}} \approx 2\pi/(n+1)$, sehingga sapuannya hanya tumbuh seperti n. Pada wajan 40×40, itulah perbedaan antara ribuan dan seratus lima puluh, seperti akan dikonfirmasi **Tabel 5.1**.

5.4.5 Kapan berhenti, dan apa arti sesungguhnya pergeseran kecil

Program tidak dapat membandingkan tebakannya dengan jawaban eksak, karena ia tidak mengetahuinya. Aturan berhenti yang wajar adalah "berhenti bila tidak ada variabel yang bergeser sebesar toleransi dalam satu sapuan". Bahayanya, pergeseran kecil bukanlah galat kecil. Bila galat menyusut dengan faktor ρ per sapuan, maka setelah k sapuan galat yang tersisa adalah jumlah semua pergeseran yang akan datang, sebuah deret geometri:

$$\| x^{(k)} - x \| \le \frac{\rho}{1-\rho} \, \| x^{(k)} - x^{(k-1)} \|. \tag{5.15}$$

Untuk Jacobi pada wajan 20×20, $\rho/(1-\rho) = 0.98883/0.01117 \approx 88.5$: ketika pergeseran terakhir 10^{-6}, galatnya masih bisa sekitar 8.9×10^{-5}, hampir sembilan puluh kali lebih besar. Subbab 5.6 mengukur galat sebenarnya dan menemukan 8.76×10^{-5}. Batas Persamaan (5.15) bukan pesimisme; itulah yang benar-benar terjadi. Metode yang konvergen lambat tampak sudah konvergen jauh sebelum waktunya. Jadi bagilah toleransi Anda dengan $\rho/(1-\rho)$ bila ρ diketahui, atau berhentilah berdasarkan *residu* $b - Ax^{(k)}$, yang mengukur seberapa parah persamaan dilanggar dan jujur untuk metode apa pun (Bab 4 menunjukkan hubungannya dengan galat melalui bilangan kondisi).

5.4.6 Untuk apa repot: biaya satu sapuan

Satu sapuan menyentuh setiap entri tak-nol A sekali. Wajan memiliki paling banyak lima entri tak-nol per baris, sehingga satu sapuan atas $N = n^2$ variabel memerlukan sekitar $5N$ perkalian, dan tidak ada yang perlu disimpan selain variabelnya: matriksnya tidak pernah dibangun, karena Persamaan (5.1) *adalah* matriksnya. Eliminasi Gauss pada matriks penuh memerlukan sekitar $\frac{2}{3}N^3$ operasi dan N^2 bilangan untuk disimpan. Untuk $n = 40$, $N = 1600$: eliminasi memerlukan sekitar 2.7×10^9 operasi dan 20 MB untuk matriksnya; SOR optimal memerlukan 155 sapuan masing-masing 8000 operasi, sekitar 1.2×10^6, dua ribu kali lebih sedikit, dan 13 KB untuk variabelnya. Untuk sejuta variabel, matriks penuh memerlukan 8 terabita; vektor SOR memerlukan 8 megabita. Eliminasi dapat dibuat lebih cerdik dengan memanfaatkan struktur pita, tetapi ia selalu mengisi nol di dalam pita dan biayanya tumbuh lebih cepat daripada biaya iteratif [5], [3]. Karena itulah setiap simulasi besar tentang panas, air tanah, medan listrik, atau struktur bangunan memakai metode iteratif atau keturunannya.

5.5 Membangunnya dalam Java dan Python

Bab ini memiliki dua program. Yang pertama, `iterative_solvers`, menjalankan Jacobi, Gauss-Seidel, dan SOR pada keempat persamaan Persamaan (5.2), mencetak sapuan-sapuan pertama agar Anda dapat mencocokkannya dengan Subbab 5.2, menguji dominansi diagonal, dan memperlihatkan sistem yang diurutkan ulang menjadi divergen. Yang kedua, `griddle`, menyelesaikan wajan $n \times n$ tanpa pernah membangun matriks, mengukur banyaknya sapuan dan galat sebenarnya dari setiap metode, memindai ω, dan mencetak peta suhu kasar.

Kedua program ditulis dua kali, dan versi Java serta Python mencetak karakter yang persis sama. Untuk itu diperlukan dua langkah pencegahan kecil, yang terlihat pada listing dan telah dijelaskan di Bab 1. Pertama, Python dan Java memformat bilangan dengan aturan pembulatan yang sedikit berbeda pada kasus-kasus langka, sehingga program Java memformat lewat `BigDecimal` dengan aturan setengah-genap yang sama dengan Python. Kedua, sejak Python 3.12 fungsi bawaan `sum()` menjumlahkan bilangan titik mengambang dengan algoritma terkompensasi (Neumaier) yang lebih teliti daripada gelung biasa [6]; agar kedua bahasa tetap seiring, program Python menjumlahkan dengan gelung biasa bernama `total`.

5.5.1 Sistem kecil

Kode 5.1 memuat ketiga sapuan sebagai fungsi. `jacobi_step` membangun list baru dari list lama, persis Persamaan (5.4); `sor_step` menyalin list sekali lalu menimpanya elemen demi elemen, sehingga baris-baris berikutnya melihat nilai baru, yaitu Persamaan (5.6) (dan Persamaan (5.5) bila `omega` bernilai 1). Fungsi penggerak `solve` adalah **Pseudokode 5.1** ditambah pencetak tabel dan penjaga yang menghentikan iterasi yang divergen.

Kode 5.1 `iterative_solvers.py`: sapuan Jacobi, Gauss-Seidel, dan SOR untuk sistem padat kecil, dengan uji dominansi diagonal dan pemeriksaan residu.

```
"""Jacobi, Gauss-Seidel, dan SOR pada wajan empat titik, langkah demi langkah."""

# 4 T1 - T2 - T3 = 280 dan seterusnya: satu persamaan per titik dalam
A = [[4.0, -1.0, -1.0, 0.0],
     [-1.0, 4.0, 0.0, -1.0],
     [-1.0, 0.0, 4.0, -1.0],
     [0.0, -1.0, -1.0, 4.0]]
B = [280.0, 140.0, 350.0, 210.0]



def total(values):
    """Jumlah biasa dari kiri ke kanan. sum() Python 3.12+ mengompensasi
    pembulatan, sehingga digit kita akan berbeda dari versi Java."""
    s = 0.0
    for v in values:
        s += v
    return s

def is_diagonally_dominant(a):
    """True bila setiap |a_ii| melebihi jumlah |a_ij| lain di barisnya."""
    for i, row in enumerate(a):
        others = total(abs(v) for j, v in enumerate(row) if j != i)
        if abs(row[i]) <= others:
            return False
    return True


def jacobi_step(a, b, x):
    """Satu sapuan Jacobi: setiap nilai baru hanya memakai vektor lama x."""
    n = len(b)
    new = [0.0] * n
    for i in range(n):
        s = b[i]
        for j in range(n):
            if j != i:
                s -= a[i][j] * x[j]
        new[i] = s / a[i][i]
    return new
```

Kode 5.1 (Lanjutan) *iterative_solvers.py: sapuan Jacobi, Gauss-Seidel, dan SOR untuk sistem padat kecil, dengan uji dominansi diagonal dan pemeriksaan residu.*

```
def sor_step(a, b, x, omega):
    """Satu sapuan SOR; omega = 1 ialah Gauss-Seidel. Nilai baru segera dipakai."""
    x = list(x)
    n = len(b)
    for i in range(n):
        s = b[i]
        for j in range(n):
            if j != i:
                s -= a[i][j] * x[j]
        x[i] = (1.0 - omega) * x[i] + omega * s / a[i][i]
    return x


def residual(a, b, x):
    """|b_i - (Ax)_i| terbesar: seberapa jauh x gagal memenuhi persamaan."""
    n = len(b)
    return max(abs(b[i] - total(a[i][j] * x[j] for j in range(n)))
               for i in range(n))


def solve(name, step, a, b, tol=1e-6, max_iter=200, show=0):
    x = [0.0] * len(b)
    print(name)
    if show:
        print("   k       x1        x2        x3        x4     change")
    for k in range(1, max_iter + 1):
        new = step(a, b, x)
        change = max(abs(p - q) for p, q in zip(new, x))
        x = new
        if k <= show:
            cells = "".join(f"{v:10.4f}" for v in x)
            print(f"{k:4d}{cells}{change:11.4f}")
        if change < tol:
            print(f"  converged after {k} iterations,"
                  f" residual {residual(a, b, x):.3e}")
            return x
        if change > 1e12:
            print(f"  diverged: change {change:.3e} at iteration {k}")
            return x
    print(f"  stopped after {max_iter} iterations")
    return x
```

Kode 5.1 (Lanjutan) *iterative_solvers.py: sapuan Jacobi, Gauss-Seidel, dan SOR untuk sistem padat kecil, dengan uji dominansi diagonal dan pemeriksaan residu.*

```
def main():
    print("diagonally dominant:", is_diagonally_dominant(A))
    x = solve("Jacobi", jacobi_step, A, B, show=6)
    print("  T =", " ".join(f"{v:.6f}" for v in x))
    x = solve("Gauss-Seidel", lambda a, b, v: sor_step(a, b, v, 1.0),
              A, B, show=6)
    x = solve("SOR, omega = 1.1", lambda a, b, v: sor_step(a, b, v, 1.1),
              A, B, show=3)
    x = solve("SOR, omega = 1.5", lambda a, b, v: sor_step(a, b, v, 1.5),
              A, B)
    # persamaan yang sama, ditulis dalam urutan lain
    order = [1, 0, 3, 2]
    a2 = [A[i] for i in order]
    b2 = [B[i] for i in order]
    print("reordered, diagonally dominant:", is_diagonally_dominant(a2))
    solve("Jacobi on the reordered system", jacobi_step, a2, b2)


if __name__ == "__main__":
    main()
```

Kode 5.2 adalah program yang sama dalam Java. Metode-metodenya dioper sebagai antarmuka kecil `Step`, sehingga satu metode `solve` menjalankan ketiganya. Fungsi pembantu di bagian akhir memformat bilangan seperti Python.

Kode 5.2 `IterativeSolvers.java`: ketiga metode yang sama dalam Java.

```java
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;

/** Jacobi, Gauss-Seidel, dan SOR pada wajan empat titik, langkah demi langkah. */
public class IterativeSolvers {
    // 4 T1 - T2 - T3 = 280 dan seterusnya: satu persamaan per titik dalam
    static final double[][] A = {{4, -1, -1, 0}, {-1, 4, 0, -1},
                                 {-1, 0, 4, -1}, {0, -1, -1, 4}};
    static final double[] B = {280, 140, 350, 210};

    interface Step { double[] apply(double[][] a, double[] b, double[] x); }

    /** True bila setiap |a_ii| melebihi jumlah |a_ij| lain di barisnya. */
    static boolean isDiagonallyDominant(double[][] a) {
        for (int i = 0; i < a.length; i++) {
            double others = 0;
            for (int j = 0; j < a.length; j++) {
                if (j != i) others += Math.abs(a[i][j]);
            }
            if (Math.abs(a[i][i]) <= others) return false;
        }
        return true;
    }

    /** Satu sapuan Jacobi: setiap nilai baru hanya memakai vektor lama x. */
    static double[] jacobiStep(double[][] a, double[] b, double[] x) {
        int n = b.length;
        double[] next = new double[n];
        for (int i = 0; i < n; i++) {
            double s = b[i];
            for (int j = 0; j < n; j++) {
                if (j != i) s -= a[i][j] * x[j];
            }
            next[i] = s / a[i][i];
        }
        return next;
    }
```

Kode 5.2 (Lanjutan) *IterativeSolvers.java: ketiga metode yang sama dalam Java.*

```
    /** Satu sapuan SOR; omega = 1 ialah Gauss-Seidel. Nilai baru segera dipakai. */
    static double[] sorStep(double[][] a, double[] b, double[] x0, double omega) {
        double[] x = x0.clone();
        int n = b.length;
        for (int i = 0; i < n; i++) {
            double s = b[i];
            for (int j = 0; j < n; j++) {
                if (j != i) s -= a[i][j] * x[j];
            }
            x[i] = (1.0 - omega) * x[i] + omega * s / a[i][i];
        }
        return x;
    }

    /** |b_i - (Ax)_i| terbesar: seberapa jauh x gagal memenuhi persamaan. */
    static double residual(double[][] a, double[] b, double[] x) {
        double worst = 0;
        for (int i = 0; i < b.length; i++) {
            double s = 0;
            for (int j = 0; j < b.length; j++) s += a[i][j] * x[j];
            worst = Math.max(worst, Math.abs(b[i] - s));
        }
        return worst;
    }

    static double[] solve(String name, Step step, double[][] a, double[] b,
                          double tol, int maxIter, int show) {
        double[] x = new double[b.length];
        System.out.println(name);
        if (show > 0) {
            System.out.println("   k       x1        x2        x3        x4"
                               + "      change");
        }
        for (int k = 1; k <= maxIter; k++) {
            double[] next = step.apply(a, b, x);
            double change = 0;
            for (int i = 0; i < x.length; i++) {
                change = Math.max(change, Math.abs(next[i] - x[i]));
            }
            x = next;
            if (k <= show) {
                StringBuilder row = new StringBuilder(pad("" + k, 4));
                for (double v : x) row.append(f(v, 4, 10));
                System.out.println(row + f(change, 4, 11));
            }
            if (change < tol) {
```

Kode 5.2 (Lanjutan) *IterativeSolvers.java: ketiga metode yang sama dalam Java.*

```
                System.out.println("  converged after " + k + " iterations,"
                        + " residual " + e(residual(a, b, x), 3));
                return x;
            }
            if (change > 1e12) {
                System.out.println("  diverged: change " + e(change, 3)
                        + " at iteration " + k);
                return x;
            }
        }
        System.out.println("  stopped after " + maxIter + " iterations");
        return x;
    }

    public static void main(String[] args) {
        System.out.println("diagonally dominant: "
                + (isDiagonallyDominant(A) ? "True" : "False"));
        double[] x = solve("Jacobi", IterativeSolvers::jacobiStep, A, B,
                           1e-6, 200, 6);
        StringBuilder t = new StringBuilder("  T =");
        for (double v : x) t.append(" ").append(f(v, 6));
        System.out.println(t);
        solve("Gauss-Seidel", (a, b, v) -> sorStep(a, b, v, 1.0), A, B,
              1e-6, 200, 6);
        solve("SOR, omega = 1.1", (a, b, v) -> sorStep(a, b, v, 1.1), A, B,
              1e-6, 200, 3);
        solve("SOR, omega = 1.5", (a, b, v) -> sorStep(a, b, v, 1.5), A, B,
              1e-6, 200, 0);
        // persamaan yang sama, ditulis dalam urutan lain
        int[] order = {1, 0, 3, 2};
        double[][] a2 = new double[4][];
        double[] b2 = new double[4];
        for (int i = 0; i < 4; i++) {
            a2[i] = A[order[i]];
            b2[i] = B[order[i]];
        }
        System.out.println("reordered, diagonally dominant: "
                + (isDiagonallyDominant(a2) ? "True" : "False"));
        solve("Jacobi on the reordered system", IterativeSolvers::jacobiStep,
              a2, b2, 1e-6, 200, 0);
    }
```

Kode 5.2 (Lanjutan) *IterativeSolvers.java: ketiga metode yang sama dalam Java.*

```
    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }

    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                  + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }

    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }
}
```

5.5.2 Wajan tanpa matriks

Untuk wajan besar kita sama sekali tidak menyimpan A. Kisinya berupa larik $(n+2)\times(n+2)$ yang bingkainya memuat suhu tepi dan bagian dalamnya memuat variabel; tetangga titik di sebelah tepi hanyalah sel bingkai. Sapuan Jacobi membaca dari salinan kisi dan menulis rata-rata ke kisi yang asli; sapuan SOR bekerja di tempat dan melakukan relaksasi lebih. **Kode 5.3** mengerjakannya dalam Python. Fungsi `run` menghitung sapuan sampai tidak ada titik yang bergeser sebesar `tol` atau lebih, dan `optimal_omega` adalah Persamaan (5.13) dengan Persamaan (5.12). Program utama menyelesaikan wajan 20×20 sekali dengan sangat teliti (toleransi 10^{-12}) sebagai acuan, lalu mengukur setiap metode terhadapnya.

Kode 5.3 `griddle.py`: wajan $n\times n$ diselesaikan dengan sapuan Jacobi, Gauss-Seidel, dan SOR pada kisi berbingkai, lengkap dengan banyak sapuan, galat sebenarnya, pemindaian ω, dan peta suhu.

```python
"""Suhu tunak wajan datar martabak, dicari dengan Jacobi, Gauss-Seidel
dan SOR pada kisi n x n titik dalam; keempat tepi dijaga tetap."""
import math

LEFT, RIGHT, TOP, BOTTOM = 200.0, 60.0, 80.0, 150.0   # suhu tepi, C


def new_grid(n):
    """Kisi (n+2) x (n+2), titik dalam 0, bingkainya memuat suhu tepi."""
    u = [[0.0] * (n + 2) for _ in range(n + 2)]
    for i in range(1, n + 1):
        u[i][0] = LEFT
        u[i][n + 1] = RIGHT
        u[0][i] = TOP
        u[n + 1][i] = BOTTOM
    return u


def jacobi_sweep(u, n):
    """Setiap titik menjadi rata-rata keempat tetangga LAMA-nya."""
    old = [row[:] for row in u]
    change = 0.0
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            v = 0.25 * (old[i - 1][j] + old[i + 1][j]
                        + old[i][j - 1] + old[i][j + 1])
            change = max(change, abs(v - old[i][j]))
            u[i][j] = v
    return change


def sor_sweep(u, n, omega):
    """Nilai Gauss-Seidel dari tetangga terbaru, lalu direlaksasi lebih."""
    change = 0.0
    for i in range(1, n + 1):
        for j in range(1, n + 1):
            gs = 0.25 * (u[i - 1][j] + u[i + 1][j]
                         + u[i][j - 1] + u[i][j + 1])
            v = u[i][j] + omega * (gs - u[i][j])
            change = max(change, abs(v - u[i][j]))
            u[i][j] = v
    return change
```

Kode 5.3 (Lanjutan) *griddle.py: wajan n \times n diselesaikan dengan sapuan Jacobi, Gauss-Seidel, dan SOR pada kisi berbingkai, lengkap dengan banyak sapuan, galat sebenarnya, pemindaian \omega, dan peta suhu.*

```
def run(n, method, omega=1.0, tol=1e-6, max_iter=100000):
    """Sapu sampai tak ada titik bergeser >= tol; kembalikan kisi dan sapuan."""
    u = new_grid(n)
    for k in range(1, max_iter + 1):
        if method == "jacobi":
            change = jacobi_sweep(u, n)
        else:
            change = sor_sweep(u, n, omega)
        if change < tol:
            return u, k
    return u, max_iter


def optimal_omega(n):
    return 2.0 / (1.0 + math.sin(math.pi / (n + 1)))


def max_diff(u, v, n):
    return max(abs(u[i][j] - v[i][j])
               for i in range(1, n + 1) for j in range(1, n + 1))


def main():
    n = 20
    w = optimal_omega(n)
    ref, _ = run(n, "sor", w, tol=1e-12)
    print(f"Griddle with {n} x {n} inner points ({n * n} unknowns)")
    print("stop when no point moves by 1e-06 or more in a sweep")
    print("method           omega  sweeps  max error")
    for name, method, om in [("Jacobi", "jacobi", 1.0),
                             ("Gauss-Seidel", "sor", 1.0),
                             ("SOR", "sor", 1.5),
                             ("SOR (optimal)", "sor", w)]:
        u, k = run(n, method, om)
        print(f"{name:15s}{om:7.3f}{k:8d}{max_diff(u, ref, n):11.2e}")

    print()
    print("   n  unknowns  rho_Jacobi  Jacobi  Gauss-Seidel  SOR  omega_opt")
    for m in (10, 20, 40):
        kj = run(m, "jacobi")[1]
        kg = run(m, "sor", 1.0)[1]
        ks = run(m, "sor", optimal_omega(m))[1]
        rho = math.cos(math.pi / (m + 1))
        print(f"{m:4d}{m * m:10d}{rho:12.5f}{kj:8d}{kg:14d}{ks:5d}"
              f"{optimal_omega(m):11.3f}")
```

Kode 5.3 (Lanjutan) *griddle.py: wajan n \times n diselesaikan dengan sapuan Jacobi, Gauss-Seidel, dan SOR pada kisi berbingkai, lengkap dengan banyak sapuan, galat sebenarnya, pemindaian \omega, dan peta suhu.*

```
    print()
    print("SOR sweeps for n = 20 and different omega")
    for step in range(0, 20):
        om = 1.0 + 0.05 * step
        print(f"  omega {om:.2f}: {run(n, 'sor', om)[1]:5d}")

    print()
    print("Temperature (C) at sample points, n = 20"
          " (row 1 lies next to the top edge)")
    cols = (1, 5, 10, 15, 20)
    print("row " + "".join(f"{'col ' + str(c):>8s}" for c in cols))
    for r in (1, 5, 10, 15, 20):
        print(f"{r:3d} " + "".join(f"{ref[r][c]:8.1f}" for c in cols))
    good = sum(1 for i in range(1, n + 1) for j in range(1, n + 1)
               if 140.0 <= ref[i][j] <= 170.0)
    print(f"points between 140 and 170 C: {good} of {n * n}")


if __name__ == "__main__":
    main()
```

Kode 5.4 adalah kembarannya dalam Java. Kisinya berupa `double[n + 2][n + 2]`, dan `run` mengisi kisi yang diberikan kepadanya, sehingga solusi acuan dan solusi yang diukur dihitung dengan kode yang sama.

Kode 5.4 `Griddle.java`: pemecah wajan dalam Java.

```java
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;

/** Suhu tunak wajan datar martabak, dicari dengan Jacobi, Gauss-Seidel
 *  dan SOR pada kisi n x n titik dalam; keempat tepi dijaga tetap. */
public class Griddle {
    static final double LEFT = 200, RIGHT = 60, TOP = 80, BOTTOM = 150;

    /** Kisi (n+2) x (n+2), titik dalam 0, bingkainya memuat suhu tepi. */
    static double[][] newGrid(int n) {
        double[][] u = new double[n + 2][n + 2];
        for (int i = 1; i <= n; i++) {
            u[i][0] = LEFT;
            u[i][n + 1] = RIGHT;
            u[0][i] = TOP;
            u[n + 1][i] = BOTTOM;
        }
        return u;
    }

    /** Setiap titik menjadi rata-rata keempat tetangga LAMA-nya. */
    static double jacobiSweep(double[][] u, int n) {
        double[][] old = new double[n + 2][];
        for (int i = 0; i < n + 2; i++) old[i] = u[i].clone();
        double change = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                double v = 0.25 * (old[i - 1][j] + old[i + 1][j]
                                 + old[i][j - 1] + old[i][j + 1]);
                change = Math.max(change, Math.abs(v - old[i][j]));
                u[i][j] = v;
            }
        }
        return change;
    }
```

Kode 5.4 (Lanjutan) *Griddle.java: pemecah wajan dalam Java.*

```
    /** Nilai Gauss-Seidel dari tetangga terbaru, lalu direlaksasi lebih. */
    static double sorSweep(double[][] u, int n, double omega) {
        double change = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                double gs = 0.25 * (u[i - 1][j] + u[i + 1][j]
                                  + u[i][j - 1] + u[i][j + 1]);
                double v = u[i][j] + omega * (gs - u[i][j]);
                change = Math.max(change, Math.abs(v - u[i][j]));
                u[i][j] = v;
            }
        }
        return change;
    }

    /** Sapu sampai tak ada titik bergeser >= tol; kisi terisi di u. */
    static int run(double[][] u, int n, boolean jacobi, double omega,
                   double tol) {
        for (int k = 1; k <= 100000; k++) {
            double change = jacobi ? jacobiSweep(u, n) : sorSweep(u, n, omega);
            if (change < tol) return k;
        }
        return 100000;
    }

    static int sweeps(int n, boolean jacobi, double omega) {
        return run(newGrid(n), n, jacobi, omega, 1e-6);
    }

    static double optimalOmega(int n) {
        return 2.0 / (1.0 + Math.sin(Math.PI / (n + 1)));
    }

    static double maxDiff(double[][] u, double[][] v, int n) {
        double worst = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                worst = Math.max(worst, Math.abs(u[i][j] - v[i][j]));
            }
        }
        return worst;
    }
```

Kode 5.4 (Lanjutan) *Griddle.java: pemecah wajan dalam Java.*

```
    public static void main(String[] args) {
        int n = 20;
        double w = optimalOmega(n);
        double[][] ref = newGrid(n);
        run(ref, n, false, w, 1e-12);
        System.out.println("Griddle with " + n + " x " + n + " inner points ("
                           + n * n + " unknowns)");
        System.out.println("stop when no point moves by 1e-06 or more in a sweep");
        System.out.println("method           omega  sweeps  max error");
        String[] names = {"Jacobi", "Gauss-Seidel", "SOR", "SOR (optimal)"};
        double[] omegas = {1.0, 1.0, 1.5, w};
        for (int m = 0; m < 4; m++) {
            double[][] u = newGrid(n);
            int k = run(u, n, m == 0, omegas[m], 1e-6);
            String name = (names[m] + " ".repeat(15)).substring(0, 15);
            System.out.println(name + f(omegas[m], 3, 7) + pad("" + k, 8)
                               + pad(e(maxDiff(u, ref, n), 2), 11));
        }

        System.out.println();
        System.out.println("   n  unknowns  rho_Jacobi  Jacobi  Gauss-Seidel"
                           + "  SOR  omega_opt");
        for (int m : new int[] {10, 20, 40}) {
            double rho = Math.cos(Math.PI / (m + 1));
            System.out.println(pad("" + m, 4) + pad("" + m * m, 10)
                    + f(rho, 5, 12) + pad("" + sweeps(m, true, 1.0), 8)
                    + pad("" + sweeps(m, false, 1.0), 14)
                    + pad("" + sweeps(m, false, optimalOmega(m)), 5)
                    + f(optimalOmega(m), 3, 11));
        }

        System.out.println();
        System.out.println("SOR sweeps for n = 20 and different omega");
        for (int step = 0; step < 20; step++) {
            double om = 1.0 + 0.05 * step;
            System.out.println("  omega " + f(om, 2) + ": "
                               + pad("" + sweeps(n, false, om), 5));
        }
```

Kode 5.4 (Lanjutan) *Griddle.java: pemecah wajan dalam Java.*

```
        System.out.println();
        System.out.println("Temperature (C) at sample points, n = 20"
                           + " (row 1 lies next to the top edge)");
        int[] cols = {1, 5, 10, 15, 20};
        StringBuilder head = new StringBuilder("row ");
        for (int c : cols) head.append(pad("col " + c, 8));
        System.out.println(head);
        for (int r : cols) {
            StringBuilder line = new StringBuilder(pad("" + r, 3) + " ");
            for (int c : cols) line.append(f(ref[r][c], 1, 8));
            System.out.println(line);
        }
        int good = 0;
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= n; j++) {
                if (ref[i][j] >= 140.0 && ref[i][j] <= 170.0) good++;
            }
        }
        System.out.println("points between 140 and 170 C: " + good + " of "
                           + n * n);
    }

    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }

    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                    + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }
```

Kode 5.4 (Lanjutan) *Griddle.java: pemecah wajan dalam Java.*

```
    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }
}
```

5.6 Menjalankannya

Kompilasikan program Java sekali di folder `java` dengan `javac -d out *.java` lalu jalankan dengan `java -cp out IterativeSolvers` dan `java -cp out Griddle`; jalankan versi Python dengan `python3 iterative_solvers.py` dan `python3 griddle.py` di folder `python`. Kedua bahasa mencetak teks yang persis sama. Sistem kecil lebih dahulu:

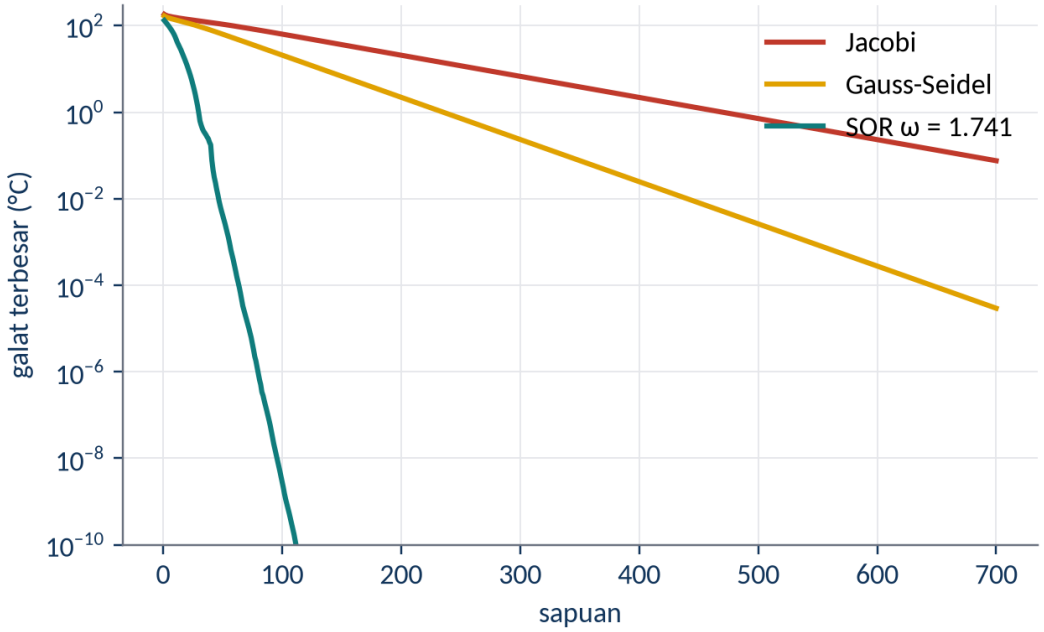
```
diagonally dominant: True
Jacobi
   k       x1        x2        x3        x4     change
   1   70.0000   35.0000   87.5000   52.5000    87.5000
   2  100.6250   65.6250  118.1250   83.1250    30.6250
   3  115.9375   80.9375  133.4375   98.4375    15.3125
   4  123.5938   88.5938  141.0938  106.0938     7.6562
   5  127.4219   92.4219  144.9219  109.9219     3.8281
   6  129.3359   94.3359  146.8359  111.8359     1.9141
  converged after 27 iterations, residual 1.825e-06
  T = 131.249999 96.249999 148.749999 113.749999
Gauss-Seidel
   k       x1        x2        x3        x4     change
   1   70.0000   52.5000  105.0000   91.8750   105.0000
   2  109.3750   85.3125  137.8125  108.2812    39.3750
   3  125.7812   93.5156  146.0156  112.3828    16.4062
   4  129.8828   95.5664  148.0664  113.4082     4.1016
   5  130.9082   96.0791  148.5791  113.6646     1.0254
   6  131.1646   96.2073  148.7073  113.7286     0.2563
  converged after 15 iterations, residual 9.779e-07
SOR, omega = 1.1
   k       x1        x2        x3        x4     change
   1   77.0000   59.6750  117.4250  106.4525   117.4250
   2  118.0025   94.2576  146.2326  113.2396    41.0025
   3  131.3346   96.3321  148.8846  113.8606    13.3321
  converged after 10 iterations, residual 1.122e-07
SOR, omega = 1.5
  converged after 29 iterations, residual 8.545e-07
reordered, diagonally dominant: False
Jacobi on the reordered system
  diverged: change 1.073e+12 at iteration 16
```

Bacalah tabel Jacobi bersama Subbab 5.2.3: dua baris pertama adalah yang kita hitung dengan tangan, dan mulai baris ketiga kolom “change” menjadi separuh setiap kali, sebagaimana diramalkan $\rho(T_J) = 0.5$. Nilai terakhir yang dilaporkan, 131.249999 dan seterusnya, berada dalam 10^{-6} dari jawaban eksak. Gauss-Seidel mencapai ketelitian yang sama dalam 15 sapuan dengan pergeseran yang menyusut empat kali lipat, SOR dengan $\omega = 1.1$ (dekat dengan optimum 1.072) dalam 10, dan SOR dengan $\omega = 1.5$ melewati sasaran dan memerlukan 29. Residu yang dicetak di akhir sangat kecil untuk semuanya. Sistem yang diurutkan ulang tidak dominan diagonal, dan Jacobi lepas kendali, persis seperti peringatan di Subbab 5.4.2.

Sekarang wajan yang sebenarnya:

```
Griddle with 20 x 20 inner points (400 unknowns)
stop when no point moves by 1e-06 or more in a sweep
method           omega  sweeps  max error
Jacobi           1.000    1302   8.76e-05
Gauss-Seidel     1.000     683   4.31e-05
SOR              1.500     235   1.36e-05
SOR (optimal)    1.741      80   8.85e-07

   n  unknowns  rho_Jacobi  Jacobi  Gauss-Seidel  SOR  omega_opt
  10       100     0.95949     385           202   42      1.560
  20       400     0.98883    1302           683   80      1.741
  40      1600     0.99707    4518          2377  155      1.858

SOR sweeps for n = 20 and different omega
  omega 1.00:   683
  omega 1.05:   621
  omega 1.10:   565
  omega 1.15:   513
  omega 1.20:   465
  omega 1.25:   420
  omega 1.30:   378
  omega 1.35:   340
  omega 1.40:   303
  omega 1.45:   268
  omega 1.50:   235
  omega 1.55:   204
  omega 1.60:   173
  omega 1.65:   143
  omega 1.70:   111
  omega 1.75:    84
  omega 1.80:    92
  omega 1.85:   128
  omega 1.90:   189
  omega 1.95:   380

Temperature (C) at sample points, n = 20 (row 1 lies next to the top edge)
row    col 1   col 5  col 10  col 15  col 20
  1    139.8    94.4    85.6    81.1    70.5
  5    183.9   135.7   106.0    86.8    65.1
 10    190.7   156.4   123.9    96.6    66.4
 15    191.5   162.3   136.4   111.2    70.5
 20    174.5   153.7   147.7   141.6   105.2
points between 140 and 170 C: 94 of 400
```

Tabel pertama layak dibaca pelan-pelan. Keempat iterasi berhenti dengan aturan yang sama, pergeseran di bawah 10^{-6}, tetapi galat sebenarnya berbeda sampai seratus kali lipat. Galat Jacobi 8.76×10^{-5} cocok dengan batas Persamaan (5.15), 88.5×10^{-6}, hampir sampai digit terakhir; galat Gauss-Seidel 4.31×10^{-5} cocok dengan $\rho_{GS}/(1-\rho_{GS}) \times 10^{-6} = 44.0 \times 10^{-6}$. Hanya SOR optimal, yang ρ-nya jauh dari 1, berhenti dengan galat di bawah toleransi. **Gambar 5.2** memperlihatkan seluruh riwayatnya: pada skala logaritmik galat Jacobi dan Gauss-Seidel turun mengikuti garis lurus, dengan kemiringan $\log_{10}\rho_J$ dan $\log_{10}\rho_{GS}$, yang satu dua kali lebih curam daripada yang lain; SOR turun jauh lebih cepat setelah keraguan singkat di awal, yaitu pertumbuhan sementara yang telah diperingatkan di Subbab 5.4.1.



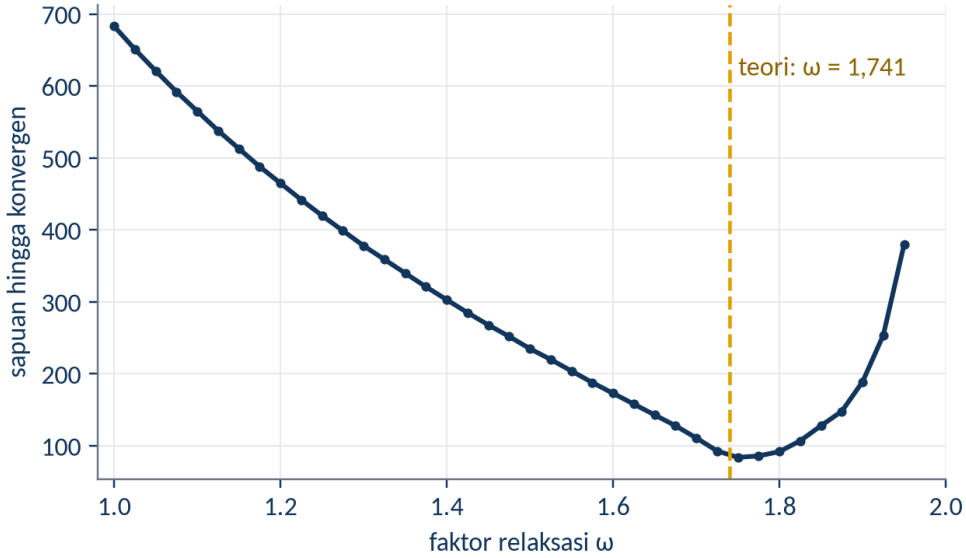
Gambar 5.2 Galat terbesar wajan 20×20 setelah setiap sapuan, terhadap solusi acuan. Garis lurus pada skala ini berarti faktor tetap per sapuan: $\rho_J = 0.989$, $\rho_{GS} = 0.978$, dan sekitar 0.74 untuk SOR dengan $\omega = 1.741$.

Tabel kedua keluaran itu dirangkum, bersama ramalan Subbab 5.4, dalam **Tabel 5.1**. Ramalan Persamaan (5.14) untuk Jacobi dengan $n = 20$, yang dimulai dari galat sekitar 200 °C dan berhenti ketika pergeserannya 10^{-6} (jadi ketika $(1-\rho)\,E_0\rho^k \approx 10^{-6}$), memberi 1301.6 sapuan; program memerlukan 1302. Menggandakan n dari 20 menjadi 40 mengalikan sapuan Jacobi dan Gauss-Seidel dengan sekitar 3.5 (dekat dengan faktor 4 dari hukum n^2) dan sapuan SOR dengan sekitar 1.9 (dekat dengan faktor 2 dari hukum n).

Tabel 5.1 Banyak sapuan yang diperlukan agar pergeseran terbesar pada wajan $n \times n$ turun di bawah 10^{-6}. Gauss-Seidel memerlukan sekitar separuh sapuan Jacobi, sesuai ramalan $\rho_{GS} = \rho_J^2$; SOR optimal hanya tumbuh linear terhadap n.

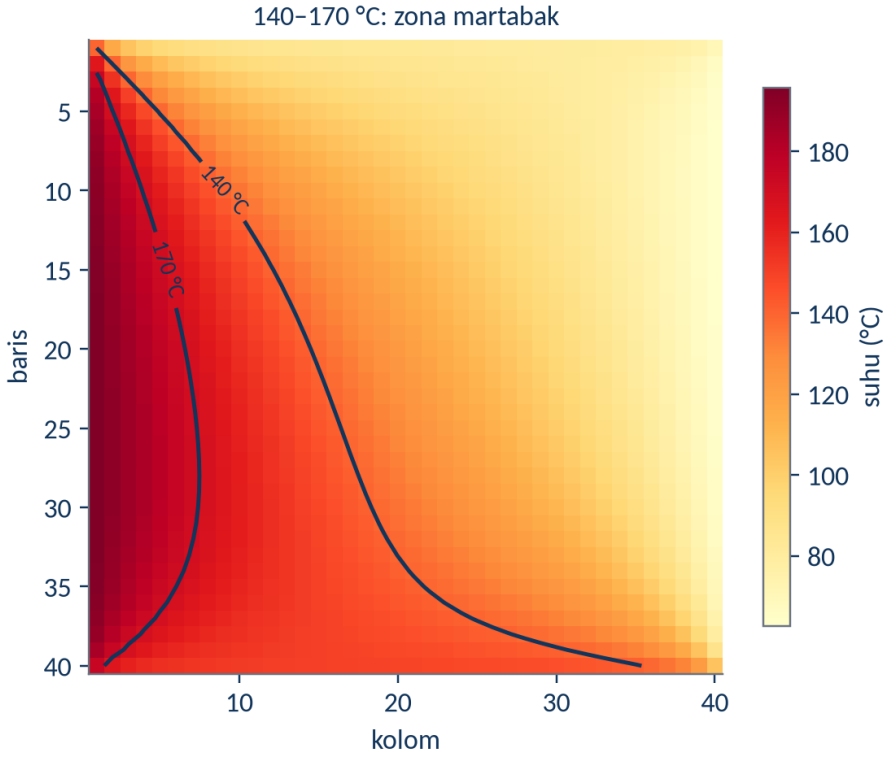
n	Variabel	ρ_J	Jacobi	Gauss-Seidel	SOR, ω_{opt}	ω_{opt}
10	100	0.95949	385	202	42	1.560
20	400	0.98883	1302	683	80	1.741
40	1600	0.99707	4518	2377	155	1.858

Pemindaian ω pada keluaran digambar dalam **Gambar 5.3**. Kurvanya turun perlahan dari 683 sapuan pada $\omega = 1$, mencapai dasarnya 84 sapuan pada $\omega = 1.75$ (80 pada optimum eksak 1.741), lalu naik tajam: 380 sapuan pada $\omega = 1.95$, dan divergen pada $\omega = 2$. Ketidaksimetrisan ini pelajaran praktis. Bila Anda tidak mengetahui ω_{opt} secara tepat, lebih baik keliru sedikit ke atas, tempat kurvanya curam tetapi masih rendah di dekat dasar, daripada jauh ke bawah; lebih baik lagi, taksirlah ρ_J dan pakailah Persamaan (5.13).



Gambar 5.3 Banyak sapuan yang diperlukan SOR pada wajan 20 × 20 untuk faktor relaksasi dari 1 sampai 1.95. Titik terendahnya berada pada nilai Young $\omega_{\text{opt}} = 1.741$.

Akhirnya, petanya. Tabel sampel pada keluaran menunjukkan 191.5 °C di dekat kompor dan 65 sampai 70 °C di sepanjang tepi kanan. **Gambar 5.4** menggambar seluruh pelat pada kisi 40 × 40 yang lebih halus, satu titik per sentimeter, dengan garis 140 °C dan 170 °C. Zona martabak berupa pita selebar delapan sampai sepuluh sentimeter yang dimulai beberapa sentimeter dari kompor dan melebar ke arah tepi depan yang hangat, lalu membelok ke kanan di sepanjang tepi itu. Pada kisi 20 × 20, 94 dari 400 titik, hampir seperempat pelat, berada di zona itu. Keponakan Pak Joni kini memiliki gambaran: martabak telur di pita itu, roti bakar tepat di sebelah kanannya, dan pojok kanan belakang untuk menjaga makanan tetap hangat.



Gambar 5.4 Suhu tunak wajan pada kisi 40 × 40 (kompor di kiri, tepi depan yang hangat di bawah), dengan garis 140 °C dan 170 °C yang membatasi zona martabak.

5.7 Memeriksa dengan Pustaka

Program kita cukup kecil untuk dipercaya dengan membacanya, tetapi pendapat kedua dari pustaka yang teruji baik itu murah. Dalam Python, alat baku untuk sistem jarang ada di `scipy.sparse`; pada saat buku ini ditulis (September 2026) rilis terbarunya adalah NumPy 2.5.3 dan SciPy 1.18.1 [7], [8]. **Kode 5.5** membangun matriks wajan secara eksplisit dengan `diags` dan hasil kali Kronecker (cara ringkas untuk mengatakan "pola satu dimensi yang sama di kedua arah"), menyimpannya dalam format *compressed sparse row* yang hanya menyimpan entri tak-nol, lalu menyelesaikannya dua kali: dengan pemecah langsung jarang `spsolve`, yaitu eliminasi Gauss yang bekerja keras menghindari pengisian (*fill-in*), dan dengan metode gradien konjugat `cg` [9], metode iteratif untuk matriks simetris definit positif yang merupakan sepupu lebih cerdas dari metode-metode di bab ini.

Kode 5.5 `compare_scipy.py`: wajan diperiksa dengan pemecah langsung jarang SciPy dan metode gradien konjugatnya.

```
"""Periksa wajan dengan pemecah langsung matriks jarang SciPy dan CG-nya."""
import numpy as np
import scipy
from scipy.sparse import diags, identity, kron
from scipy.sparse.linalg import cg, spsolve

import griddle
from iterative_solvers import A, B


def griddle_system(n):
    """Matriks dan ruas kanan n*n persamaan wajan, baris demi baris."""
    t = diags([-1.0, 4.0, -1.0], [-1, 0, 1], shape=(n, n))
    s = diags([-1.0, -1.0], [-1, 1], shape=(n, n))
    a = (kron(identity(n), t) + kron(s, identity(n))).tocsr()
    b = np.zeros(n * n)
    for i in range(n):
        for j in range(n):
            k = i * n + j
            if i == 0:
                b[k] += griddle.TOP
            if i == n - 1:
                b[k] += griddle.BOTTOM
            if j == 0:
                b[k] += griddle.LEFT
            if j == n - 1:
                b[k] += griddle.RIGHT
    return a, b


def main():
    print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
    print("4 x 4 system, numpy.linalg.solve:",
          np.round(np.linalg.solve(np.array(A), np.array(B)), 6))
    n = 20
    a, b = griddle_system(n)
    print(f"griddle matrix: {a.shape[0]} x {a.shape[1]},"
          f" {a.nnz} nonzeros of {a.shape[0] * a.shape[1]}")
    x = spsolve(a, b)
    u, k = griddle.run(n, "sor", griddle.optimal_omega(n), tol=1e-12)
    ours = np.array([u[i][j] for i in range(1, n + 1)
                     for j in range(1, n + 1)])
    print(f"SOR ({k} sweeps) vs spsolve: max difference"
          f" {np.max(np.abs(ours - x)):.1e}")
    count = []
    xc, info = cg(a, b, rtol=1e-10, callback=lambda xk: count.append(1))
```

Kode 5.5 (Lanjutan) *compare_scipy.py: wajan diperiksa dengan pemecah langsung jarang SciPy dan metode gradien konjugatnya.*

```
    print(f"conjugate gradient: {len(count)} iterations,"
          f" max difference {np.max(np.abs(xc - x)):.1e}")


if __name__ == "__main__":
    main()
```

```
NumPy 2.5.3, SciPy 1.18.1
4 x 4 system, numpy.linalg.solve: [131.25  96.25 148.75 113.75]
griddle matrix: 400 x 400, 1920 nonzeros of 160000
SOR (126 sweeps) vs spsolve: max difference 1.4e-12
conjugate gradient: 64 iterations, max difference 6.9e-09
```

Sistem kecil cocok dengan `numpy.linalg.solve` sampai digit terakhir yang dicetak. Matriks wajan memiliki 1920 entri tak-nol dari 160 000 entri, 1.2%; untuk kisi 1000×1000 pecahannya menjadi 0.0005%, dan itulah sebabnya menyimpan entri tak-nol saja itu penting. Solusi SOR kita, yang diiterasi sampai 10^{-12}, cocok dengan pemecah langsung sampai 1.4×10^{-12}. Gradien konjugat memerlukan 64 iterasi untuk residu relatif 10^{-10}; setiap iterasinya kira-kira semahal satu sapuan SOR, dan ia sama sekali tidak memerlukan ω. Dalam praktik, inilah yang Anda panggil untuk masalah simetris yang besar, biasanya dengan *prapengondisi* (*preconditioner*) yang sering berupa satu sapuan SOR atau faktorisasi tak lengkap [5], [10]. Jadi metode-metode bab ini bukan barang museum: mereka adalah batu bata di dalam pemecah-pemecah modern.

5.8 Membuatnya Lebih Baik

Empat perbaikan layak diketahui, meskipun program di atas tidak memerlukannya.

- **Mulailah dari tebakan yang baik.** Banyak sapuan bergantung pada ukuran galat pertama E_0 melalui Persamaan (5.14). Ketika Pak Joni membesarkan kompor dari 200 menjadi 210 °C, peta lama adalah tebakan awal yang sangat baik untuk peta baru; Soal 5 menunjukkan bahwa hal itu menghemat sekitar 30% sapuan bahkan pada kisi kecil ini. Sistem yang diselesaikan berulang-ulang dengan data yang berubah perlahan (model cuaca, perhitungan ulang mingguan) sebaiknya selalu dimulai dari jawaban sebelumnya.
- **Pengurutan merah-hitam.** Warnailah kisi seperti papan catur. Setiap titik merah hanya bertetangga dengan titik hitam dan sebaliknya, sehingga semua titik merah dapat diperbarui serentak dari titik hitam, lalu semua titik hitam dari titik merah. Ini Gauss-Seidel dengan urutan lain, konvergen dengan laju yang sama pada wajan, dan setiap setengah sapuan sepenuhnya paralel. Bab 15 kembali membahasnya.
- **Multigrid.** Bagian galat yang lambat adalah gelombang halus pada Persamaan (5.12). Pada kisi yang lebih kasar gelombang itu tidak lagi halus, dan beberapa sapuan saja melenyapkannya. Metode multigrid menyapu pada hierarki kisi dan menyelesaikan wajan dengan banyak operasi yang sebanding dengan banyak variabel, yang terbaik yang dapat diharapkan [5].
- **Metode Krylov.** Gradien konjugat, yang dilihat di Subbab 5.7, dan kerabatnya untuk sistem taksimetris (GMRES, BiCGSTAB) memilih setiap langkah dengan memakai semua langkah sebelumnya. Dengan prapengondisi yang baik, mereka menjadi pilihan bawaan untuk sistem jarang besar di pustaka seperti SciPy [10], [9].

5.9 Memakainya dengan Sungguh-Sungguh

Untuk membuat peta wajan, dinding oven, lantai penjemuran, atau benda datar apa pun yang dipanaskan atau didinginkan dari tepinya, empat hal perlu diubah dalam `griddle.py`, dan tidak ada yang lain.

1. **Ukur tepi-tepinya.** Termometer inframerah tidak mahal. Ukurlah setiap tepi di tiga atau empat titik setelah pelat berhenti bertambah panas. Bila suatu tepi tidak seragam, isilah sel-sel bingkai dengan nilai terukur, yang diinterpolasi di antara titik-titik ukur (Bab 7 menunjukkan caranya), alih-alih satu konstanta per tepi.
2. **Pilih kisinya.** Satu titik per sentimeter sudah lebih dari cukup untuk memasak; kisi 40×40 pada **Gambar 5.4** hanya memerlukan sepersekian detik dengan SOR.
3. **Tangani tepi yang terisolasi.** Bila suatu tepi tidak bersentuhan dengan apa pun dan tidak kehilangan panas, tepi itu tidak memiliki suhu tetap. Yang berlaku adalah tidak ada panas yang melintasinya, artinya suhu tepat di dalam sama dengan suhu tepat di luar. Terapkan dengan menyalin baris dalam pertama ke baris bingkai sebelum setiap sapuan; aturan rata-rata akan mengerjakan sisanya.
4. **Periksa modelnya.** Ukurlah tiga atau empat titik dalam dengan termometer dan bandingkan. Bila bagian tengah pelat yang terukur jelas lebih dingin daripada peta, panas yang lolos dari permukaan atas berperan, dan aturan rata-rata memerlukan suku kehilangan: $T = (T_{\text{kiri}} + T_{\text{kanan}} + T_{\text{atas}} + T_{\text{bawah}} + \beta T_{\text{udara}})/(4 + \beta)$ untuk konstanta kecil β yang dicocokkan dengan pengukuran Anda (Bab 8 menunjukkan cara mencocokkannya). Matriksnya tetap dominan diagonal, bahkan sejati, sehingga segala isi bab ini tetap berlaku.

Satu kata hati-hati untuk alat apa pun: peta hasil hitungan menunjukkan di mana pelat lebih panas atau lebih dingin, bukan apakah alatnya aman. Kompor gas yang memanaskan secara tidak merata, berbau gas, atau berapi kuning memerlukan teknisi, bukan model.

5.10 Metode Numerik dalam Praktik: Profitina

AI Visibility Engine Profitina melihat sebuah situs web sebagai satu kesatuan, halaman demi halaman, dan menilai seberapa siap situs itu untuk dikutip oleh mesin jawaban AI. Halaman-halaman sebuah situs saling bertautan, dan beberapa besaran yang mungkin ingin diketahui tentang situs itu didefinisikan dengan cara yang sama seperti suhu wajan: nilai suatu halaman bergantung pada nilai halaman-halaman di sekitarnya. Bila dituliskan, definisi seperti itu adalah sistem linear jarang yang besar, satu variabel per halaman dan hanya sebanyak entri tak-nol per baris sebanyak tautan halaman itu. Itulah persis bentuk yang dibahas bab ini, dan golongan metode yang dipakai untuknya adalah sapuan: mulai dari tebakan, perbarui setiap halaman dari tetangganya, ulangi. Bab 12 bertemu dengan contoh yang paling terkenal, PageRank, yaitu iterasi jenis Jacobi pada graf tautan web.

Dua pelajaran bab ini terbawa langsung ke produk yang mengukur sesuatu berulang-ulang. Yang pertama adalah *warm start*. Penjadwal Profitina memindai ulang setiap merek secara otomatis dengan irama yang ditentukan oleh tingkat langganannya dan menyimpan tren skor mingguan di dasbor analitik historis. Dari satu pemindaian ke pemindaian berikutnya hanya beberapa halaman yang berubah, sehingga nilai minggu lalu adalah tebakan awal yang sangat baik, persis seperti peta wajan lama untuk kompor yang sedikit lebih panas. Pelajaran kedua adalah kejujuran tentang konvergensi. Persamaan (5.15) menunjukkan bahwa "angkanya sudah berhenti bergerak" dapat menyembunyikan galat sembilan puluh kali lebih besar daripada pergeseran terakhir. Dasbor yang menampilkan skor kepada pemilik merek, yang bukan ahli analisis

numerik, harus menghentikan perhitungannya dengan kriteria yang membatasi galat sesungguhnya, atau melaporkan ketidakpastiannya. Aplikasi memberi saran dan pemilik yang memutuskan, dan saran hanya sebaik angka-angka di bawahnya.

5.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Beriterasi pada sistem yang urutannya buruk.** Jacobi dan Gauss-Seidel memerlukan entri diagonal yang besar. Sebelum beriterasi, periksa Persamaan (5.11) atau setidaknya tempatkan koefisien terbesar setiap persamaan di diagonal; wajan yang diurutkan ulang di Subbab 5.6 divergen meskipun solusinya sama.
- **Menganggap pergeseran kecil sebagai galat kecil.** Bila ρ dekat dengan 1, galat setelah berhenti kira-kira $\rho/(1-\rho)$ kali pergeseran terakhir, 88 kali untuk Jacobi pada wajan 20×20. Perketat toleransinya, atau berhentilah berdasarkan residu.
- **Tak sengaja mencampur Jacobi dan Gauss-Seidel.** Menulis pembaruan Jacobi ke larik yang sama dengan yang dibaca mengubahnya menjadi sapuan Gauss-Seidel dengan urutan yang tidak Anda pilih (dalam Python, `new = x` alih-alih `new = x[:]` melakukan persis hal itu). Jacobi memerlukan salinan terpisah dari nilai lama.
- **Memilih ω di luar (0,2).** SOR divergen untuk setiap matriks bila $\omega \le 0$ atau $\omega \ge 2$. Nilai sedikit di atas optimum murah akibatnya; nilai jauh di bawahnya membuang sebagian besar manfaatnya.
- **Memakai ω optimal suatu kisi untuk kisi lain.** ω_{opt} bergantung pada n: 1.560 untuk $n = 10$, 1.858 untuk $n = 40$. Hitung ulang dengan Persamaan (5.13) bila kisinya berubah.
- **Lupa batas banyaknya sapuan.** Gelung `while change >= tol` berjalan selamanya pada masalah yang divergen atau konvergen sangat lambat. Selalu batasi banyak sapuan dan laporkan bila batas itu tercapai, seperti baris 10 pada **Pseudokode 5.1**.
- **Membangun matriks yang tidak diperlukan.** Untuk kisi, aturan rata-rata itulah matriksnya. Menyimpan N^2 bilangan untuk $N = 10^6$ mustahil; menyimpan N variabel itu sepele.

5.12 Ringkasan Bab

- Banyak sistem linear besar hanya memiliki sedikit variabel per persamaan. Suhu tunak pelat adalah salah satunya: setiap titik dalam adalah rata-rata keempat tetangganya (Persamaan (5.1)), yang menghasilkan sistem empat titik Persamaan (5.2) untuk kisi paling kasar dan sistem dengan n^2 variabel untuk kisi $n \times n$.
- Metode iteratif memperbaiki tebakan lengkap sapuan demi sapuan. Jacobi (Persamaan (5.4)) memperbarui setiap variabel dari sapuan sebelumnya, Gauss-Seidel (Persamaan (5.5)) langsung memakai nilai baru, dan SOR (Persamaan (5.6)) melakukan relaksasi lebih atas langkah Gauss-Seidel dengan faktor ω (**Pseudokode 5.1**).
- Ketiganya adalah pemisahan $A = M - N$ (Persamaan (5.7)) dan iterasi stasioner $x^{(k+1)} = Tx^{(k)} + c$ (Persamaan (5.8)). Galat dikalikan dengan T pada setiap sapuan (Persamaan (5.9)), sehingga metode konvergen dari setiap awal tepat bila jari-jari spektral (Persamaan (5.10)) kurang dari 1.
- Dominansi diagonal sejati (Persamaan (5.11)) menjamin konvergensi Jacobi dan Gauss-Seidel dan dapat diperiksa dengan mata; matriks simetris definit positif menjamin Gauss-Seidel dan SOR dengan $0 < \omega < 2$.
- Pada wajan, $\rho_J = \cos(\pi/(n+1))$ (Persamaan (5.12)), $\rho_{GS} = \rho_J^2$, dan $\omega_{\text{opt}} = 2/(1+\sin(\pi/(n+1)))$ (Persamaan (5.13)). Banyak sapuan (Persamaan (5.14)) tumbuh seperti n^2 untuk Jacobi

dan Gauss-Seidel dan seperti n untuk SOR optimal: 4518, 2377, dan 155 sapuan untuk $n = 40$ (**Tabel 5.1**).

- Pergeseran kecil bukan galat kecil: galat yang tersisa bisa sampai $\rho/(1-\rho)$ kali pergeseran terakhir (Persamaan (5.15)), dan pengukuran membenarkan batas itu hampir sampai digit terakhir.
- Program Java dan Python cocok bita demi bita, dan `spsolve` serta gradien konjugat SciPy membenarkan solusi wajan sampai 10^{-12} dan 10^{-9}.

5.13 Soal Latihan

1. Lanjutkan tabel Jacobi wajan empat titik dengan tangan untuk sapuan 7 dan 8, dimulai dari nilai setelah sapuan 6 yang dicetak di Subbab 5.6. Dengan faktor berapa pergeserannya menyusut, dan seberapa jauh T_1 dari 131.25 setelah sapuan 8?
2. Untuk wajan empat titik, tuliskan $T_J = 1/4\,(L+U)$ sebagai matriks 4×4 dan tentukan nilai eigennya. (Petunjuk: cobalah vektor $(1,1,1,1)$, $(1,-1,-1,1)$, $(1,0,0,-1)$, dan $(0,1,-1,0)$.) Pakailah Persamaan (5.13) untuk menentukan ρ_{GS} dan ω_{opt}, lalu bandingkan dengan Subbab 5.4.3.
3. Apakah sistem $5x_1 + x_2 + 2x_3 = 13$, $x_1 + 4x_2 + x_3 = 12$, $2x_1 + x_2 + 6x_3 = 22$ dominan diagonal sejati? Kerjakan tiga sapuan Gauss-Seidel dengan tangan dari $x^{(0)} = 0$ dan bandingkan dengan tiga sapuan Jacobi. Solusi eksaknya $(1,2,3)$.
4. Pak Joni membesarkan kompor utama sehingga tepi kiri bersuhu 220 °C. Tanpa beriterasi, manfaatkan kenyataan bahwa solusi Persamaan (5.2) bergantung linear pada ruas kanan untuk menentukan keempat suhu yang baru. (Petunjuk: perubahan jawaban menyelesaikan sistem yang sama dengan ruas kanan $(20,0,20,0)$.)
5. Wajan 20×20 sudah diselesaikan untuk tepi kiri 200 °C. Kompor dibesarkan menjadi 210 °C. Ubahlah `griddle.py` sehingga SOR dengan ω_{opt} dimulai sekali dari nol dan sekali dari solusi lama (dengan tepi kiri yang baru ditulis ke bingkai), lalu hitung banyak sapuan sampai pergeseran di bawah 10^{-6} untuk kedua kasus. Jelaskan hasilnya dengan Persamaan (5.14).
6. Tepi kanan wajan diganti dengan gagang kayu tebal yang tidak meneruskan panas sama sekali. Jelaskan cara mengubah `new_grid` dan `sor_sweep` agar tepi kanan terisolasi, seperti di Subbab 5.9, dan ramalkan, tanpa menghitung, apakah separuh kanan pelat menjadi lebih panas atau lebih dingin.
7. Dengan Persamaan (5.14) dan ketergantungan ketiga jari-jari spektral pada n, taksirlah berapa sapuan yang diperlukan Jacobi, Gauss-Seidel, dan SOR optimal untuk wajan 160×160, bertolak dari banyak sapuan untuk $n = 40$ pada **Tabel 5.1**.

5.14 Penyelesaian Soal Latihan

1. Setelah sapuan 6 nilainya 129.3359, 94.3359, 146.8359, 111.8359. Sapuan 7: $T_1 = (280 + 94.3359 + 146.8359)/4 = 521.1719/4 = 130.2930$; $T_2 = (140 + 129.3359 + 111.8359)/4 = 381.1719/4 = 95.2930$; $T_3 = (350 + 129.3359 + 111.8359)/4 = 591.1719/4 = 147.7930$; $T_4 = (210 + 94.3359 + 146.8359)/4 = 451.1719/4 = 112.7930$. Pergeserannya $130.2930 - 129.3359 = 0.9570$ (sampai pembulatan digit terakhir), separuh dari 1.9141 sebelumnya. Sapuan 8 memberi 130.7715, 95.7715, 148.2715, 113.2715 dengan pergeseran 0.4785, sekali lagi separuh. Setelah sapuan 8, T_1 berada $131.25 - 130.7715 = 0.4785$ di bawah jawaban: di sini galat sama dengan pergeseran terakhir, karena galat menjadi separuh setiap sapuan dan $\rho/(1-\rho) = 0.5/0.5 = 1$.

2. Baris demi baris, T_J bernilai $1/4$ di tempat dua titik bertetangga:

$$T_J = \frac{1}{4}\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

Dengan mengalikan: $T_J(1,1,1,1) = 1/4\,(2,2,2,2) = 1/2\,(1,1,1,1)$, nilai eigen $1/2$; $T_J(1,-1,-1,1) = 1/4\,(-2,2,2,-2) = -1/2\,(1,-1,-1,1)$, nilai eigen $-1/2$; $T_J(1,0,0,-1) = 1/4\,(0,0,0,0)$ dan $T_J(0,1,-1,0) = 0$, nilai eigen 0 dua kali. Jadi $\rho_J = 1/2$, yaitu $\cos(\pi/3)$ dari Persamaan (5.12) dengan $n = 2$. Maka $\rho_{GS} = 1/4$ dan $\omega_{\text{opt}} = 2/\left(1+\sqrt{3/4}\right) = 2/1.8660 = 1.0718$, dengan $\rho(T_{\omega_{\text{opt}}}) = 0.0718$. Perhitungan nilai eigen T_{GS} dan $T_{1.0718}$ secara numerik memberi 0.25 dan 0.0718 yang sama.

3. Baris 1: $5 > 1 + 2 = 3$; baris 2: $4 > 1 + 1 = 2$; baris 3: $6 > 2 + 1 = 3$. Matriksnya dominan diagonal sejati, sehingga kedua metode konvergen. Gauss-Seidel: sapuan 1, $x_1 = 13/5 = 2.6$, $x_2 = (12 - 2.6)/4 = 2.35$, $x_3 = (22 - 2 \times 2.6 - 2.35)/6 = 14.45/6 = 2.4083$. Sapuan 2: $x_1 = (13 - 2.35 - 2 \times 2.4083)/5 = 1.1667$, $x_2 = (12 - 1.1667 - 2.4083)/4 = 2.1063$, $x_3 = (22 - 2.3333 - 2.1063)/6 = 2.9267$. Sapuan 3: (1.0081,2.0163,2.9946). Setelah tiga sapuan Jacobi baru mencapai (1.3933,2.2917,3.25) (sapuan 1: (2.6,3.0,3.6667); sapuan 2: (0.5333,1.4333,2.3)): ia berayun bolak-balik melewati sasaran, sedangkan Gauss-Seidel mendekat dengan mantap dan sekitar sepuluh kali lebih dekat.

4. Ruas kanannya menjadi (300,140,370,210), yaitu yang lama ditambah (20,0,20,0). Karena A tidak berubah, solusi baru adalah solusi lama ditambah solusi d dari $Ad = (20,0,20,0)$. Karena simetri $d_1 = d_3$ dan $d_2 = d_4$; persamaan pertama memberi $4d_1 - d_2 - d_1 = 20$, yaitu $3d_1 - d_2 = 20$, dan persamaan kedua $-d_1 + 4d_2 - d_2 = 0$, yaitu $d_1 = 3d_2$. Jadi $8d_2 = 20$, $d_2 = 2.5$, $d_1 = 7.5$. Suhu barunya $T_1 = 138.75$, $T_2 = 98.75$, $T_3 = 156.25$, $T_4 = 116.25$. Menaikkan kompor 20 °C menaikkan titik-titik dekat sebesar 7.5 °C dan titik-titik jauh sebesar 2.5 °C.

5. Setel `griddle.LEFT = 210.0` sebelum memanggil `new_grid` untuk awal dingin; untuk awal hangat, salinlah kisi acuan yang dihitung dengan 200 °C dan tuliskan 210.0 ke `u[i][0]` untuk $i = 1, \ldots, n$. Dengan $\omega_{\text{opt}} = 1.741$, awal dingin memerlukan 80 sapuan dan awal hangat 57. Menurut Persamaan (5.14), banyak sapuan bergantung pada $\ln(\varepsilon/E_0)$: dari nol galat pertamanya sekitar 200 °C, dari peta lama paling besar 10 °C (perubahan tepi kiri, yang merambat ke dalam dan lebih kecil di tempat lain). Perbandingan 20 itu bernilai $\ln 20/\ln(1/0.741) \approx 10$ sapuan menurut laju asimtotiknya, dan lebih dari itu dalam praktik karena awal hangat juga menghindari keadaan sementara SOR di awal. Pada kisi yang lebih besar dan perubahan yang lebih kecil, penghematannya bertambah.

6. Dalam `new_grid`, kolom bingkai kanan tidak lagi memuat suhu tetap. Dalam `sor_sweep`, sebelum gelung ganda, setel `u[i][n + 1] = u[i][n]` untuk setiap baris i (nilai tepat di luar sama dengan nilai tepat di dalam, sehingga tidak ada panas yang melintasi tepi); baris matriks untuk titik di sebelah tepi kanan kemudian memiliki 3, bukan 4, pada diagonal efektifnya dan tidak lagi dominan sejati, tetapi seluruh sistem tetap dominan secara takteruraikan karena tiga tepi lainnya tetap, dan SOR tetap konvergen. Separuh kanan menjadi lebih panas: sebelumnya panas mengalir keluar lewat tepi kanan yang bersuhu 60 °C; kini panas tidak dapat keluar di sana, sehingga harus keluar lewat tepi atas dan bawah, dan pelat di dekat gagang menghangat menuju suhu tepi-tepi lainnya.

7. Sapuan Jacobi dan Gauss-Seidel tumbuh seperti n^2 (lebih tepatnya seperti $(n+1)^2$, dengan faktor logaritmik dari E_0 yang kita abaikan), dan SOR optimal seperti $n+1$. Dari $n = 40$ ke $n = 160$, $(161/41)^2 \approx 15.4$ dan $161/41 \approx 3.93$. Taksirannya: Jacobi $4518 \times 15.4 \approx 70{,}000$ sapuan, Gauss-Seidel $2377 \times 15.4 \approx 37{,}000$, SOR optimal $155 \times 3.93 \approx 610$. Dengan 25 600 variabel dan sekitar lima operasi per variabel, Jacobi memerlukan sekitar 9×10^9 operasi, SOR sekitar 8×10^7, lebih dari seratus kali lebih sedikit.

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins University Press, 2013.

[4] D. M. Young, *Iterative Solution of Large Linear Systems*. New York, NY, USA: Academic Press, 1971 (reprinted by Dover, 2003).

[5] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM, 2003.

[6] Python Software Foundation, "What's New In Python 3.12: sum()," accessed Sep. 2026. [Online]. Available: https://docs.python.org/3/whatsnew/3.12.html

[7] SciPy Developers, "SciPy 1.18.0 Release Notes," Jun. 19, 2026, and "scipy 1.18.1" on the Python Package Index, Aug. 21, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/release/1.18.0-notes.html and https://pypi.org/project/scipy/

[8] NumPy Developers, "NumPy 2.5.0 Release Notes," Jun. 21, 2026, and "numpy 2.5.3" on the Python Package Index, Sep. 6, 2026. [Online]. Available: https://numpy.org/doc/2.5/release/2.5.0-notes.html and https://pypi.org/project/numpy/

[9] SciPy Developers, "scipy.sparse.linalg.cg," SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.cg.html

[10] R. Barrett et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd ed. Philadelphia, PA, USA: SIAM, 1994.

Bab 6

Dekomposisi Matriks: LU, Cholesky, dan QR

Eliminasi Gauss, sebagaimana dikerjakan di Bab 4, membuang hasil kerjanya: begitu jawaban keluar, pengali-pengali yang memungkinkannya dilupakan, dan ruas kanan yang baru berarti mulai lagi dari atas. Bab ini menyimpan hasil kerja itu. Sebuah matriks ditulis sebagai hasil kali matriks-matriks yang lebih sederhana, sebuah *dekomposisi*, lalu potongan-potongannya dipakai ulang sesering yang kita mau. Contoh yang menyertai kita sepanjang bab adalah kebun buah dan durian yang harus mengubah rekomendasi hara dari penyuluh pertanian menjadi gram pupuk, blok demi blok dan musim demi musim. Di akhir bab ini Anda akan mampu memfaktorkan matriks sebagai $PA = LU$ dengan *pivoting* parsial dan menyelesaikan banyak ruas kanan dengan masing-masing dua substitusi segitiga; menghitung determinan dan invers dari faktor-faktornya; mengenali matriks simetris definit positif dan memfaktorkannya dengan Cholesky dengan separuh biaya; menyusun masalah kuadrat terkecil ketika persyaratan lebih banyak daripada variabel dan menyelesaikannya baik dengan persamaan normal maupun dengan QR Householder; menjelaskan, dengan angka, mengapa QR yang lebih aman di antara keduanya; membangun semuanya dalam Java dan Python; serta memeriksanya dengan SciPy.

6.1 Pertanyaan Pupuk Pak Harun

Pak Harun dan keluarganya mengurus kebun buah dan durian di sebuah lereng bukit. Pohon-pohonnya dikelompokkan dalam empat blok: pohon muda yang ditanam tiga tahun lalu, pohon dewasa yang baru selesai panen, pohon dewasa menjelang berbunga, dan pohon yang buahnya sedang membesar di dahan. Dua kali setahun seorang penyuluh pertanian memeriksa pohon-pohon itu dan hasil uji tanah, lalu menuliskan rekomendasi untuk setiap blok, dalam gram tiga hara utama per pohon: nitrogen (N), fosfat (ditulis P_2O_5), dan kalium (K_2O). Untuk pohon muda musim ini tertulis 150 g N, 60 g fosfat, dan 70 g kalium per pohon.

Toko tani di desa tidak menjual hara; yang dijual adalah karung pupuk, masing-masing dengan *kadar* yang tercetak. Karung berlabel NPK 12-12-17 mengandung 12% N, 12% P_2O_5, dan 17% K_2O menurut berat; NPK 20-10-10 lebih kaya nitrogen; urea, 46-0-0, hampir separuhnya nitrogen dan tidak berisi hara lain. Pak Harun menyimpan ketiganya. Pertanyaannya, untuk setiap blok, berapa gram setiap pupuk yang harus ditabur di bawah setiap pohon agar pohon itu memperoleh tepat apa yang dianjurkan.

Bila dituliskan, pertanyaan itu adalah sistem linear, tiga persamaan (satu per hara) dan tiga variabel (satu per pupuk). Bab 4 dapat menyelesaikannya. Tetapi Pak Harun tidak hanya punya satu sistem, ia punya empat musim ini, empat lagi musim depan, dan rekomendasinya berubah setiap kali penyuluh berubah pikiran. Matriks kadarnya tetap sama bertahun-tahun; hanya ruas kanannya yang berubah. Bab ini menunjukkan cara mengerjakan bagian mahal dari eliminasi sekali saja dan memakainya ulang, lalu apa yang harus dilakukan ketika penyuluh menambahkan hara keempat, magnesium, sehingga tiga pupuk tidak lagi mampu memenuhi empat target secara tepat.

Satu catatan praktis sebelum matematikanya. Program menghitung campuran yang memenuhi suatu rekomendasi; program tidak menentukan rekomendasinya. Dosis untuk sebatang pohon bergantung pada umurnya, tanahnya, dan musimnya, dan pupuk yang berlebihan membakar akar serta mencemari sungai. Angka-angka dalam bab ini adalah contoh; untuk pohon Anda sendiri, bertanyalah kepada penyuluh pertanian atau laboratorium tanah, lalu pakailah program untuk mengubah saran mereka menjadi gram.

6.2 Gagasannya dalam Kata-Kata Sederhana

6.2.1 Persamaannya

Ukurlah pupuk dalam satuan 100 gram, sehingga satu satuan NPK 12-12-17 mengandung 12 g N, 12 g fosfat, dan 17 g kalium. Bila x_1, x_2, dan x_3 adalah banyaknya (dalam ratusan gram) NPK 12-12-17, NPK 20-10-10, dan urea, nitrogen yang diberikan adalah $12x_1 + 20x_2 + 46x_3$, dan seterusnya. **Tabel 6.1** mengumpulkan kadarnya; setiap kolom satu karung, setiap baris satu hara.

Tabel 6.1 Gram setiap hara dalam 100 g setiap pupuk, dan rekomendasi untuk satu pohon muda.

Hara	NPK 12-12-17	NPK 20-10-10	Urea 46-0-0	Kebutuhan pohon muda
N	12	20	46	150 g
P_2O_5	12	10	0	60 g
K_2O	17	10	0	70 g

Maka sistem untuk pohon muda adalah

$$\begin{pmatrix} 12 & 20 & 46 \\ 12 & 10 & 0 \\ 17 & 10 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 150 \\ 60 \\ 70 \end{pmatrix}, \tag{6.1}$$

yaitu $Ax = b$ dengan matriks kadar A dan rekomendasi b. Tiga blok lainnya memiliki A yang sama dan b yang berbeda: (180, 90, 110) setelah panen, (120, 90, 120) menjelang berbunga, dan (100, 80, 110) saat pengisian buah.

6.2.2 Eliminasi yang mengingat: mencari L dan U dengan tangan

Jalankan eliminasi Gauss dengan *pivoting* parsial pada A (Bab 4), tetapi kali ini catat setiap pengalinya.

Langkah 1. Entri terbesar kolom 1 dalam nilai mutlak adalah 17, di baris kalium, jadi tukarkan ke atas. Urutan baris sekarang K, P, N:

$$\begin{pmatrix} 17 & 10 & 0 \\ 12 & 10 & 0 \\ 12 & 20 & 46 \end{pmatrix}.$$

Pengali untuk baris P adalah $12/17 = 0.7059$: mengurangkan 0.7059 kali baris teratas mengubah (12,10,0) menjadi $(0,\ 10 - 7.0588,\ 0) = (0,\ 2.9412,\ 0)$. Pengali untuk baris N juga 12/17: (12,20,46) menjadi $(0,\ 12.9412,\ 46)$.

Langkah 2. Di kolom 2, di bawah baris teratas, calonnya 2.9412 (baris P) dan 12.9412 (baris N). Yang lebih besar di baris N, jadi tukarkan dua baris terakhir; urutannya kini K, N, P, dan pengali-pengalinya ikut berpindah bersama barisnya. Pengali untuk baris P adalah $2.9412/12.9412 = 0.2273$ (tepatnya $5/22$), dan $(0,2.9412,0)$ menjadi $(0,\ 0,\ 0 - 0.2273 \times 46) = (0,0,-10.4545)$.

Yang tersisa adalah matriks segitiga atas U, dan pengali-pengalinya, ditempatkan di posisi tempat mereka dipakai, membentuk matriks segitiga bawah L dengan angka satu di diagonalnya:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0.7059 & 1 & 0 \\ 0.7059 & 0.2273 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 17 & 10 & 0 \\ 0 & 12.9412 & 46 \\ 0 & 0 & -10.4545 \end{pmatrix}.$$

Penukaran baris dicatat dalam *matriks permutasi* P, yaitu matriks identitas dengan baris-baris dalam urutan akhir (K, N, P): mengalikan dengan P menyusun baris-baris A dalam urutan itu. Klaim bab ini adalah bahwa

$$PA = LU. \tag{6.2}$$

Anda dapat memeriksanya dengan mengalikan: baris kedua LU adalah $0.7059 \times (17,10,0) + (0,12.9412,46) = (12,20,46)$, yaitu baris N dari A, baris kedua PA. Tidak ada yang baru dihitung. L dan U hanyalah hasil kerja eliminasi, dicatat alih-alih dibuang.

6.2.3 Dua segitiga per blok

Dengan faktor-faktor di tangan, menyelesaikan $Ax = b$ tinggal dua langkah mudah. Kalikan $Ax = b$ dengan P: $PAx = Pb$, yaitu $L(Ux) = Pb$. Namakan $y = Ux$. Maka

$$Ly = Pb \quad \text{(substitusi maju)}, \qquad Ux = y \quad \text{(substitusi mundur)}. \tag{6.3}$$

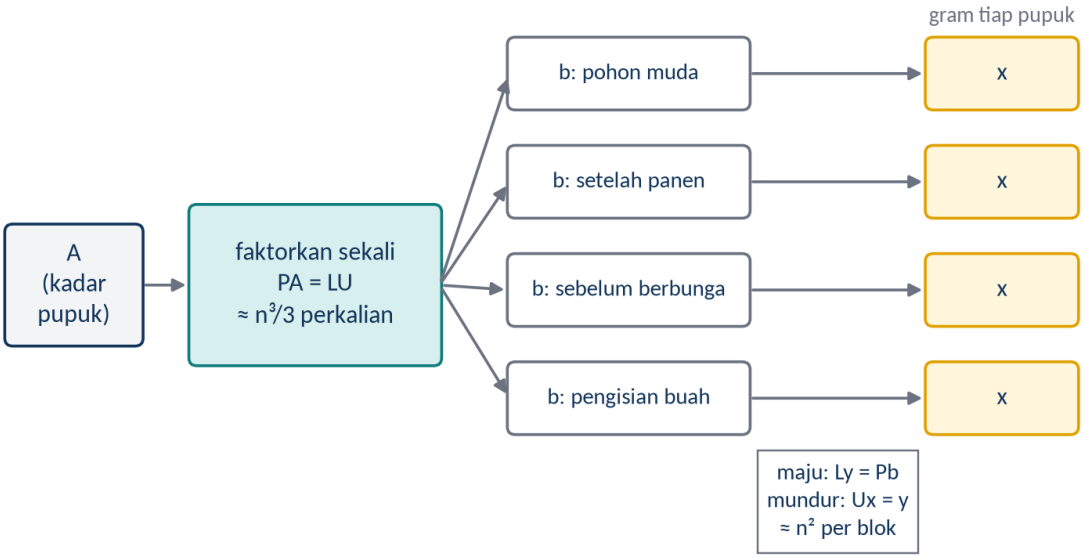
Untuk pohon muda, Pb menyusun rekomendasi dalam urutan K, N, P: $(70,150,60)$. Maju, dari atas ke bawah, karena L segitiga bawah:

- $y_1 = 70$
- $y_2 = 150 - 0.7059 \times 70 = 150 - 49.4118 = 100.5882$
- $y_3 = 60 - 0.7059 \times 70 - 0.2273 \times 100.5882 = 60 - 49.4118 - 22.8610 = -12.2727$

Mundur, dari bawah ke atas, karena U segitiga atas:

- $x_3 = -12.2727/-10.4545 = 1.1739$
- $x_2 = (100.5882 - 46 \times 1.1739)/12.9412 = (100.5882 - 54.0000)/12.9412 = 3.6$
- $x_1 = (70 - 10 \times 3.6 - 0 \times 1.1739)/17 = 34/17 = 2$

Jadi setiap pohon muda mendapat 200 g NPK 12-12-17, 360 g NPK 20-10-10, dan 117.4 g urea. Periksa nitrogennya: $12 \times 2 + 20 \times 3.6 + 46 \times 1.1739 = 24 + 72 + 54.0 = 150$ g. Untuk blok berikutnya hanya kedua substitusi itu yang diulang, segelintir perkalian, sedangkan eliminasinya sendiri tidak pernah dikerjakan lagi. **Gambar 6.1** menggambarkan polanya: faktorkan sekali, lalu dua segitiga per ruas kanan.



Gambar 6.1 Faktorkan sekali, selesaikan berkali-kali: matriks kadar difaktorkan sebagai $PA = LU$ sekali, dan rekomendasi setiap blok hanya memerlukan satu substitusi maju dan satu substitusi mundur.

6.3 Dekomposisi LU

6.3.1 Dari mana faktor-faktor itu berasal

Mengapa hasil kali pengali dan baris-baris akhir sama dengan A? Perhatikan apa yang dilakukan eliminasi terhadap baris ke-i: ia mengurangkan kelipatan l_{ik} dari baris-baris pivot $u_{k\cdot}$ (baris-baris U) untuk $k = 1,2,\dots,i-1$, dan yang tersisa adalah baris ke-i dari U. Dibaca terbalik, baris aslinya adalah jumlah dari yang dikurangkan ditambah yang tersisa:

$$\text{baris } i \text{ dari } A = \sum_{k<i} l_{ik}\ (\text{baris } k \text{ dari } U) + 1 \cdot (\text{baris } i \text{ dari } U),$$

dan itulah persis baris ke-i dari LU bila L memuat pengali di bawah diagonal dan angka satu di diagonalnya. Dengan penukaran baris, argumen yang sama berlaku untuk baris-baris PA. Membandingkan entri-entri $A = LU$ memberi faktor-faktornya secara langsung, tanpa membicarakan eliminasi. Inilah bentuk *Doolittle* dari algoritmanya [1], [2]: untuk $k = 1,\dots,n$,

$$u_{kj} = a_{kj} - \sum_{m<k} l_{km}\,u_{mj} \quad (j \ge k), \qquad l_{ik} = \frac{1}{u_{kk}}\left(a_{ik} - \sum_{m<k} l_{im}\,u_{mk}\right) \quad (i > k). \tag{6.4}$$

Rumus pertama menghitung baris ke-k dari U dari yang sudah diketahui, rumus kedua kolom ke-k dari L. Keduanya memerlukan $u_{kk} \neq 0$, dan di sinilah *pivoting* masuk: sebelum menghitung kolom ke-k dari L, pilih di antara baris $i \ge k$ yang $|a_{ik} - \sum l_{im}u_{mk}|$-nya terbesar dan tukarkan ke posisi k. **Pseudokode 6.1** mengerjakan persis hal itu di tempat: matriks ditimpa oleh faktor-faktornya, U pada dan di atas diagonal serta pengali L di bawahnya (angka satu di diagonal L tidak disimpan), dan vektor `perm` mencatat penukarannya.

Pseudokode 6.1 Dekomposisi LU dengan *pivoting* parsial, disimpan di tempat, dan dua penyelesaian segitiga.

```
LU-DECOMPOSE(A)                    // A berukuran n x n, ditimpa
 1  perm = (1, 2, ..., n); sign = 1
 2  for k = 1 to n - 1
 3      p = baris i >= k dengan |A[i][k]| terbesar
 4      if A[p][k] == 0: error "singular"
 5      if p != k: tukar baris k dan p dari A dan dari perm; sign = -sign
 6      for i = k + 1 to n
 7          A[i][k] = A[i][k] / A[k][k]            // pengali l_ik
 8          for j = k + 1 to n
 9              A[i][j] = A[i][j] - A[i][k] * A[k][j]
10  return A, perm, sign

LU-SOLVE(A, perm, b)
11  for i = 1 to n: y[i] = b[perm[i]] - sum_{j<i} A[i][j] * y[j]
12  for i = n downto 1: x[i] = (y[i] - sum_{j>i} A[i][j] * x[j]) / A[i][i]
13  return x
```

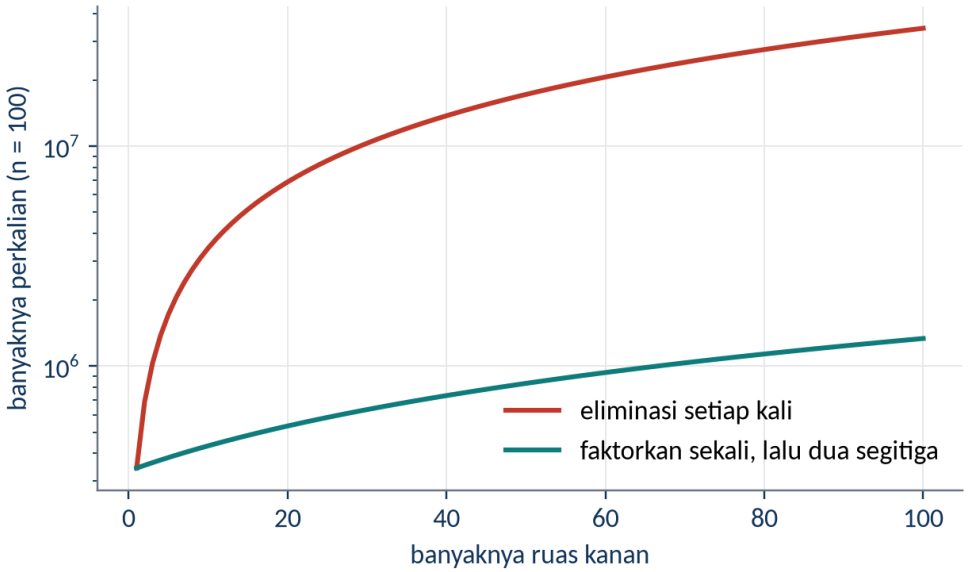
DEFINISI. *Dekomposisi LU* dari matriks persegi A adalah faktorisasi $PA = LU$ dengan matriks permutasi P, matriks segitiga bawah satuan L (angka satu di diagonal), dan matriks segitiga atas U. Dengan *pivoting* parsial, dekomposisi ini ada untuk setiap matriks persegi, dan tunggal bila A tak singular dan aturan *pivoting*-nya tetap [3].

6.3.2 Biayanya

Hitunglah perkalian, ukuran tradisional (banyaknya penjumlahan setara). Langkah ke-k faktorisasi memperbarui blok $(n-k)\times(n-k)$, sekitar $(n-k)^2$ perkalian, sehingga seluruh faktorisasi memerlukan

$$\sum_{k=1}^{n-1}(n-k)^2 \approx \frac{n^3}{3}, \qquad \text{sedangkan setiap penyelesaian} \quad \approx n^2. \tag{6.5}$$

Substitusi maju memerlukan sekitar $n^2/2$ perkalian dan substitusi mundur $n^2/2$ lagi. Menyelesaikan K sistem dengan matriks yang sama melalui eliminasi dari awal memerlukan $K(n^3/3 + n^2)$; dengan LU diperlukan $n^3/3 + Kn^2$. **Gambar 6.2** menggambarkan keduanya untuk $n = 100$ dan sampai 100 ruas kanan: penghematannya 26 kali lipat pada $K = 100$, dan tumbuh bersama n, menjadi 77 kali untuk $n = 1000$ (program di Subbab 6.6 mencetak hitungan ini).



Gambar 6.2 Banyak perkalian untuk K ruas kanan dengan matriks 100×100: eliminasi setiap kali tumbuh seperti $Kn^3/3$, faktorisasi sekali seperti $n^3/3 + Kn^2$.

6.3.3 Determinan dan invers tanpa biaya tambahan

Determinan suatu hasil kali adalah hasil kali determinan-determinannya. Matriks segitiga memiliki determinan berupa hasil kali diagonalnya, $\det L = 1$, dan permutasi berdeterminan $+1$ atau -1 bergantung pada genap atau ganjilnya banyak penukaran. Dari Persamaan (6.2),

$$\det A = (\pm 1)\, u_{11} u_{22} \cdots u_{nn}. \tag{6.6}$$

Untuk matriks kadar: dua penukaran, sehingga tandanya $+1$, dan $\det A = 17 \times 12.9412 \times (-10.4545) = -2300$ (tepat, karena $12.9412 = 220/17$ dan $-10.4545 = -115/11$). Determinan tak nol memastikan bahwa setiap rekomendasi memiliki tepat satu campuran. Menghitung determinan dengan ekspansi kofaktor, seperti yang diajarkan di beberapa mata pelajaran, memerlukan sekitar $n!$ operasi; dengan Persamaan (6.6) hanya $n^3/3$. Untuk $n = 20$ itu berarti 2.4×10^{18} berbanding 2700.

Invers A^{-1} adalah matriks yang kolom ke-j-nya menyelesaikan $Ax = e_j$, dengan e_j kolom ke-j matriks identitas. Dengan faktor-faktornya, itu berarti n pasang penyelesaian segitiga, sekitar n^3 perkalian seluruhnya. Anda jarang memerlukannya: untuk menyelesaikan $Ax = b$, kedua segitiga lebih murah dan lebih teliti daripada membentuk A^{-1} lalu mengalikannya. Namun kolom-kolomnya bisa memiliki makna sendiri, seperti ditunjukkan Soal 3 untuk kebun.

6.3.4 Seberapa teliti?

Bab 4 menunjukkan bahwa *pivoting* parsial menjaga pengali paling besar 1 dalam nilai mutlak, sehingga galat tidak diperbesar oleh pengali yang besar, dan bahwa ketelitian jawaban dikendalikan oleh *bilangan kondisi* $\kappa(A)$: kira-kira, solusi terhitung kehilangan $\log_{10}\kappa(A)$ dari enam belas digit yang dimuat sebuah `double`. Analisis galat mundur Wilkinson menyatakannya secara tepat: LU dengan *pivoting* parsial menghitung faktor eksak dari matriks $A + \delta A$ dengan $\| \delta A \|$ kelipatan sedang dari $\varepsilon_{\text{mach}} \| A \|$ pada semua kasus kecuali yang dibuat-buat [4]. Untuk matriks kadar $\kappa(A) \approx 27$, dan program menemukan $\| PA - LU \| = 0$ sampai bit terakhir: entrinya bilangan bulat kecil dan setiap pembulatan kebetulan saling meniadakan.

6.3.5 Apa yang salah tanpa *pivoting*

Ada gunanya melihat sekali mengapa penukaran di Subbab 6.2.2 bukan pilihan bebas. Ambil sistem 2×2 $\varepsilon x_1 + x_2 = 1$, $x_1 + x_2 = 2$ dengan $\varepsilon = 10^{-17}$, yang solusinya sangat dekat dengan $x_1 = x_2 = 1$. Tanpa *pivoting*, pivot pertama adalah ε dan pengalinya $l_{21} = 1/\varepsilon = 10^{17}$. Entri diagonal yang baru adalah $u_{22} = 1 - 10^{17}$, yang dalam presisi ganda dibulatkan menjadi -10^{17}: angka 1-nya hilang, karena sebuah `double` hanya menyimpan sekitar 16 digit. Substitusi maju memberi $y_2 = 2 - 10^{17}$, yang juga dibulatkan menjadi -10^{17}, sehingga $x_2 = 1$, benar. Tetapi kemudian $x_1 = (1 - x_2)/\varepsilon = 0/10^{-17} = 0$, yang sama sekali salah. Informasi tentang x_1 hancur pada saat pengali raksasa itu menenggelamkan baris kedua. Dengan *pivoting*, baris-barisnya ditukar lebih dahulu, pivotnya 1, pengalinya ε, tidak ada yang tenggelam, dan kedua variabel keluar sebagai 1. Matriks kadar di bab ini juga tidak berbahaya tanpa *pivoting*, tetapi program yang memfaktorkan matriks apa pun yang diberikan kepadanya harus melakukan *pivoting*, dan **Pseudokode 6.1** melakukannya.

6.4 Cholesky: Separuh Kerja untuk Matriks Simetris Definit Positif

6.4.1 Hara keempat

Setelah analisis daun, penyuluh menambahkan satu baris pada rekomendasi untuk pohon muda: 6 g magnesium oksida (MgO) per pohon, karena daun-daun tua menguning di antara tulang daunnya. NPK 12-12-17 milik Pak Harun dijual sebagai 12-12-17-2, dengan 2% MgO; dua pupuk lainnya tidak mengandung MgO. Sekarang ada empat persyaratan dan tetap tiga pupuk:

$$\begin{pmatrix} 12 & 20 & 46 \\ 12 & 10 & 0 \\ 17 & 10 & 0 \\ 2 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \approx \begin{pmatrix} 150 \\ 60 \\ 70 \\ 6 \end{pmatrix}. \tag{6.7}$$

Tiga baris pertama saja sudah memaksa $x_1 = 2$ (Subbab 6.2.3), dan baris terakhir lalu memberi 4 g MgO, bukan 6. Tidak ada campuran ketiga karung ini yang memenuhi keempat target. Sistemnya *terlalu ditentukan* (*overdetermined*), dan pertanyaan yang jujur bukan lagi "campuran mana yang memenuhi target" melainkan "campuran mana yang paling dekat". Mengukur kedekatan dengan jumlah kuadrat meleset memberi **masalah kuadrat terkecil** (*least squares*)

$$\min_x \; \| b - Ax \|_2^2 = \min_x \sum_{i=1}^{m} \left(b_i - \sum_{j=1}^{n} a_{ij}\, x_j \right)^2, \tag{6.8}$$

di sini dengan $m = 4$ persyaratan dan $n = 3$ variabel. Kuadrat dipilih karena membuat masalahnya mulus dan linear untuk diselesaikan, menghukum meleset besar lebih berat daripada yang kecil, dan menghasilkan geometri pada **Gambar 6.3**; Bab 8 seluruhnya membahasnya.

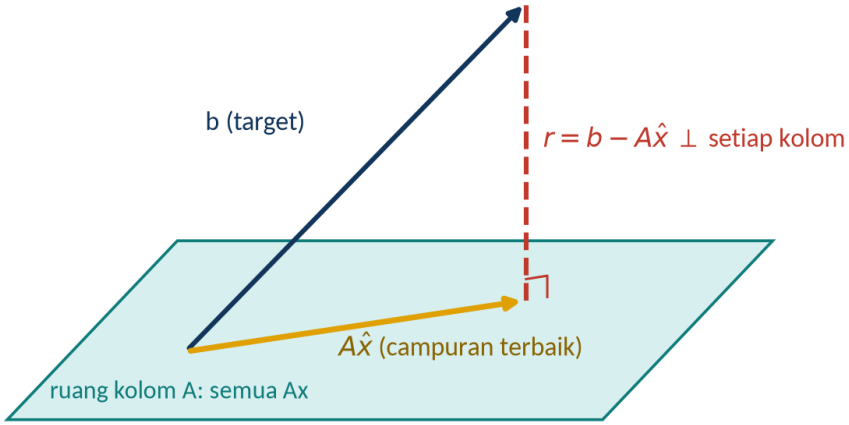
6.4.2 Persamaan normal

Setiap campuran x menghasilkan vektor Ax, dan semua vektor ini bersama-sama membentuk *ruang kolom A*, subruang datar tiga dimensi di dalam ruang empat dimensi vektor hara. Target b mencuat keluar darinya. Titik terdekat ke b dalam ruang datar adalah kaki garis tegak lurus: campuran terbaik $\hat{x}$ membuat residu $r = b - A\hat{x}$ tegak lurus terhadap setiap kolom A, seperti

ditunjukkan **Gambar 6.3**. Tegak lurus terhadap kolom ke-j berarti hasil kali titiknya nol, $\sum_i a_{ij}\, r_i = 0$, dan untuk semua kolom sekaligus $A^{\mathsf{T}} r = 0$. Substitusikan r:

$$A^{\mathsf{T}} A\,\hat{x} = A^{\mathsf{T}} b. \tag{6.9}$$

Inilah **persamaan normal** ("normal" dalam arti lamanya, tegak lurus). Persamaan yang sama keluar dari kalkulus: menyamakan turunan Persamaan (6.8) terhadap setiap x_j dengan nol memberi $-2\sum_i a_{ij}\,(b_i - (Ax)_i) = 0$, yaitu baris ke-$j$ dari Persamaan (6.9).



Gambar 6.3 Campuran kuadrat terkecil: $A\hat{x}$ adalah titik ruang kolom yang paling dekat ke target b, dan melesetnya $r = b - A\hat{x}$ tegak lurus terhadap setiap kolom.

Untuk kebun, $A^{\mathsf{T}}A$ berupa matriks 3×3. Entri (1,1)-nya adalah kolom NPK 12-12-17-2 dikalikan titik dengan dirinya sendiri, $12^2 + 12^2 + 17^2 + 2^2 = 581$; entri (1,2) adalah $12 \cdot 20 + 12 \cdot 10 + 17 \cdot 10 + 2 \cdot 0 = 530$; dan seterusnya:

$$A^{\mathsf{T}}A = \begin{pmatrix} 581 & 530 & 552 \\ 530 & 600 & 920 \\ 552 & 920 & 2116 \end{pmatrix}, \qquad A^{\mathsf{T}}b = \begin{pmatrix} 3722 \\ 4300 \\ 6900 \end{pmatrix}.$$

6.4.3 Matriks simetris definit positif

$A^{\mathsf{T}}A$ istimewa dalam dua hal. Matriks itu *simetris*, karena entri (i, j) dan entri (j, i) adalah hasil kali titik yang sama. Dan matriks itu *definit positif*: untuk setiap vektor tak nol z, $z^{\mathsf{T}}(A^{\mathsf{T}}A)z = (Az)^{\mathsf{T}}(Az) = \| Az \|^2 > 0$, asalkan kolom-kolom A bebas linear (tidak ada pupuk yang merupakan campuran pupuk lain), sehingga $Az \neq 0$.

> **DEFINISI.** Matriks persegi S *simetris definit positif* (SPD) bila $S^{\mathsf{T}} = S$ dan $z^{\mathsf{T}}Sz > 0$ untuk setiap vektor tak nol z. Matriks seperti ini muncul dari jumlah kuadrat: energi pegas, variansi, persamaan normal, dan wajan di Bab 5.

Untuk matriks SPD, eliminasi tidak pernah memerlukan *pivoting*, dan U ternyata sama dengan L^{T} sampai faktor diagonalnya. Membagi diagonal sama rata di antara kedua faktor memberi **dekomposisi Cholesky** $S = LL^{\mathsf{T}}$, dengan L segitiga bawah dan diagonal positif. Dekomposisi ini dikembangkan oleh André-Louis Cholesky, seorang perwira artileri dan juru ukur Prancis, untuk persamaan normal dalam pemetaan, dan diterbitkan oleh seorang rekannya setelah ia gugur dalam Perang Dunia Pertama [5]. Rumusnya diperoleh dengan membandingkan entri $S = LL^{\mathsf{T}}$ kolom demi kolom: untuk $j = 1, \dots, n$,

$$l_{jj} = \sqrt{s_{jj} - \sum_{k<j} l_{jk}^2}, \qquad l_{ij} = \frac{1}{l_{jj}}\left(s_{ij} - \sum_{k<j} l_{ik}\, l_{jk} \right) \quad (i > j). \tag{6.10}$$

Hanya separuh matriks yang disentuh, sehingga biayanya sekitar $n^3/6$ perkalian dan n akar kuadrat, separuh dari LU. Bilangan di bawah tanda akar selalu positif bila S SPD, dan inilah uji gratis: bila ternyata nol atau negatif, S tidak definit positif (atau pembulatan membuatnya tampak demikian), dan algoritmanya berhenti. Menyelesaikan $Sx = c$ lalu tinggal dua segitiga $Ly = c$ dan $L^{\mathsf{T}}x = y$.

Dengan tangan untuk kebun: $l_{11} = \sqrt{581} = 24.1039$; $l_{21} = 530/24.1039 = 21.9881$; $l_{31} = 552/24.1039 = 22.9008$; $l_{22} = \sqrt{600 - 21.9881^2} = \sqrt{600 - 483.477} = \sqrt{116.523} = 10.7946$; $l_{32} = (920 - 21.9881 \times 22.9008)/10.7946 = (920 - 503.545)/10.7946 = 38.5799$; $l_{33} = \sqrt{2116 - 22.9008^2 - 38.5799^2} = \sqrt{2116 - 524.448 - 1488.409} = \sqrt{103.143} = 10.1559$. Dua penyelesaian segitiga lalu memberi campuran terbaik, yang dicetak di Subbab 6.7: 224.2 g NPK 12-12-17-2, 324.8 g NPK 20-10-10, dan 126.4 g urea.

6.4.4 Mengapa Cholesky tidak memerlukan *pivoting*

LU memerlukan *pivoting* karena pivot yang kecil menghasilkan pengali yang besar, dan pengali yang besar memperbesar galat pembulatan. Cholesky tidak pernah menghadapi bahaya itu pada matriks SPD, karena dua alasan [4]. Pertama, setiap entri diagonal matriks SPD bernilai positif ($s_{jj} = e_j^{\mathsf{T}} S e_j > 0$), dan setelah satu langkah faktorisasi, bagian matriks yang masih harus difaktorkan kembali berupa matriks SPD, sehingga setiap pivot di bawah tanda akar positif. Kedua, entri-entri L tidak dapat membesar: baris ke-j dari Persamaan (6.10) menyatakan $l_{j1}^2 + \cdots + l_{jj}^2 = s_{jj}$, sehingga tidak ada entri pada baris ke-j dari L yang melebihi $\sqrt{s_{jj}}$ dalam nilai mutlak. Untuk kebun, baris 3 dari L adalah (22.9008,38.5799,10.1559) dan $22.9008^2 + 38.5799^2 + 10.1559^2 = 2116.0$, tepat s_{33}. Entri yang terbatas berarti galat pembulatan yang terbatas, dan itulah sebabnya Cholesky sekaligus faktorisasi termurah dan salah satu yang paling stabil yang dikenal. Satu hal yang tidak dapat diperbaikinya adalah kondisi matriks itu sendiri, yang menjadi pokok bahasan subbab berikutnya.

6.5 QR: Kuadrat Terkecil tanpa Mengkuadratkan

6.5.1 Harga ketelitian persamaan normal

Persamaan normal itu cepat, tetapi ada harga tersembunyinya. Bilangan kondisi $A^{\mathsf{T}}A$ adalah kuadrat bilangan kondisi A:

$$\kappa(A^{\mathsf{T}}A) = \kappa(A)^2. \tag{6.11}$$

(Untuk norma-2 hal ini mengikuti dari nilai singular: nilai singular $A^{\mathsf{T}}A$ adalah kuadrat nilai singular A [6].) Bila $\kappa(A) = 10^6$, masalah kuadrat terkecilnya sendiri layak mendapat jawaban yang benar sekitar $16 - 6 = 10$ digit, tetapi persamaan normal memiliki $\kappa = 10^{12}$ dan hanya memberi sekitar 4. Untuk kebun hal ini tidak penting ($\kappa(A) \approx 24$, $\kappa(A^{\mathsf{T}}A) \approx 560$). Untuk mencocokkan polinomial derajat 9 pada data, yang diperlukan Bab 8, hal ini sangat penting: Subbab 6.7 menunjukkan persamaan normal kehilangan tujuh digit di sana, dan gagal sama sekali pada derajat 12.

6.5.2 Matriks ortogonal tidak meregangkan

Matriks persegi Q disebut *ortogonal* bila kolom-kolomnya vektor satuan yang saling tegak lurus, yang sama artinya dengan $Q^{\mathsf{T}}Q = I$. Mengalikan dengan Q adalah rotasi atau pencerminan: panjang vektor apa pun tidak berubah, $\| Qz \| = \| z \|$, karena $\| Qz \|^2 = z^{\mathsf{T}}Q^{\mathsf{T}}Qz = z^{\mathsf{T}}z$. Andaikan kita dapat menulis $A = QR$ dengan Q ortogonal berukuran $m \times m$ dan matriks R berukuran $m \times n$ yang blok atasnya R_1 berukuran $n \times n$ segitiga atas dan $m - n$ baris sisanya nol. Kalikan residu dengan Q^{T}, yang tidak mengubah panjangnya, dan pecah $c = Q^{\mathsf{T}}b$ menjadi n entri teratasnya c_1 dan sisanya c_2:

$$\| b - Ax \|^2 = \| Q^{\mathsf{T}}b - Rx \|^2 = \| c_1 - R_1 x \|^2 + \| c_2 \|^2. \tag{6.12}$$

Suku kedua sama sekali tidak bergantung pada x. Suku pertama menjadi nol dengan menyelesaikan sistem segitiga $R_1 x = c_1$ lewat substitusi mundur. Jadi solusi kuadrat terkecil adalah satu substitusi mundur, dan besarnya meleset yang tak terhindarkan adalah $\| c_2 \|$, gratis. Hasil kali $A^{\mathsf{T}}A$ tidak pernah dibentuk, tidak ada yang dikuadratkan, dan ketelitiannya dikendalikan oleh $\kappa(A)$ alih-alih $\kappa(A)^2$ [6].

6.5.3 Refleksi Householder

Bagaimana menemukan Q? Gagasan Alston Householder (1958) membangunnya dari pencerminan [7]. Diberikan vektor a (kolom pertama A), kita menginginkan matriks ortogonal yang memetakannya ke kelipatan $e_1 = (1, 0, \ldots, 0)$, menolkan semua entri di bawah entri pertama, persis seperti langkah pertama eliminasi, tetapi tanpa meregangkan apa pun. Karena pemetaannya mempertahankan panjang, sasarannya harus αe_1 dengan $|\alpha| = \| a \|$. Cermin yang menukar a dan αe_1 tegak lurus terhadap $v = a - \alpha e_1$, dan pencerminan pada cermin itu adalah

$$H = I - \frac{2\, vv^{\mathsf{T}}}{v^{\mathsf{T}}v}. \tag{6.13}$$

H simetris dan ortogonal ($H^2 = I$: mencerminkan dua kali mengembalikan yang asli), dan $Ha = \alpha e_1$. Ada dua hal praktis. Pertama, tanda mana untuk α? Memilih $\alpha = -\text{sign}(a_1) \| a \|$ membuat entri pertama $v = a - \alpha e_1$ sama dengan $a_1 + \text{sign}(a_1) \| a \|$, jumlah dua bilangan bertanda sama, sehingga tidak terjadi pembatalan (Bab 1) bahkan ketika a hampir sejajar dengan e_1. Kedua, H tidak pernah dibentuk sebagai matriks: menerapkannya pada kolom z hanya memerlukan satu hasil kali titik, $Hz = z - (2v^{\mathsf{T}}z / v^{\mathsf{T}}v)\, v$.

Terapkan refleksi pertama pada semua kolom A dan pada b; kolom 1 menjadi $(\alpha, 0, \ldots, 0)$. Lalu ulangi pada baris dan kolom di bawah dan di kanannya, dengan refleksi yang membiarkan baris pertama, dan seterusnya untuk semua n kolom. Yang tersisa dari A adalah R, yang tersisa dari b adalah $c = Q^{\mathsf{T}}b$, dan Q sendiri, hasil kali refleksi-refleksi itu, tidak pernah diperlukan. Kerjanya sekitar $mn^2 - 1/3\, n^3$ perkalian, dua kali lipat dari membentuk dan menyelesaikan persamaan

normal bila m jauh lebih besar daripada n: harga untuk tidak mengkuadratkan bilangan kondisi [3], [6].

6.5.4 Pemeriksaan yang menghubungkan kedua metode

Karena $A^{\mathsf{T}}A = R^{\mathsf{T}}Q^{\mathsf{T}}QR = R_1^{\mathsf{T}}R_1$, faktor segitiga dari QR dan faktor Cholesky dari persamaan normal adalah matriks yang sama sampai tanda baris-barisnya: $R_1 = \pm L^{\mathsf{T}}$. Subbab 6.8 membenarkannya: R dari NumPy berdiagonal -24.1039, 10.7946, -10.1559, dan faktor Cholesky kita berdiagonal 24.1039, 10.7946, 10.1559. Kedua metode menghitung hal yang sama melalui jalan yang berbeda; jalan lewat $A^{\mathsf{T}}A$ lebih pendek dan lebih bergelombang.

6.5.5 Gram-Schmidt, jalan lain menuju QR

Banyak buku teks membangun Q dengan cara lain, dengan proses *Gram-Schmidt*: ambil kolom-kolom A satu per satu, kurangkan dari masing-masing komponennya sepanjang kolom-kolom yang sudah diterima, lalu normalkan sisanya. Hasilnya faktorisasi "tipis", $A = \hat{Q}R_1$ dengan matriks $\hat{Q}$ berukuran $m \times n$ yang kolom-kolomnya ortonormal, dan begitulah kebanyakan orang pertama kali memahami QR. Namun dalam aritmetika titik mengambang, versi klasiknya kehilangan keortogonalan ketika kolom-kolomnya hampir bergantung linear: kolom-kolom $\hat{Q}$ yang terhitung jauh dari saling tegak lurus, dan jawaban kuadrat terkecilnya menderita. Proses Gram-Schmidt *termodifikasi*, yang mengurangkan komponen-komponennya satu per satu dari sisa yang sedang berjalan, jauh lebih baik, tetapi refleksi Householder ortogonal sejak dari konstruksinya, bagaimanapun rupa kolom-kolomnya, dan itulah sebabnya LAPACK, dan karenanya NumPy serta SciPy, memakainya [3], [6].

Tabel 6.2 menjajarkan ketiga dekomposisi.

Tabel 6.2 Ketiga dekomposisi bab ini, apa yang diperlukannya, berapa biayanya, dan untuk apa.

Dekomposisi	Untuk matriks apa	Biaya (perkalian)	Dipakai untuk
$PA = LU$	sembarang persegi	$n^3/3$	sistem persegi, banyak ruas kanan, determinan, invers
$S = LL^{\mathsf{T}}$ (Cholesky)	simetris definit positif	$n^3/6$	sistem SPD, persamaan normal, uji SPD
$A = QR$ (Householder)	sembarang $m \times n$, $m \geq n$	$mn^2 - n^3/3$	kuadrat terkecil tanpa mengkuadratkan κ

6.6 Membangunnya dalam Java dan Python

Sekali lagi dua program. `lu_blend` memfaktorkan matriks kadar sekali, mencetak P, L, dan U, memeriksa $PA = LU$, menghitung determinan, menyelesaikan keempat blok, dan mencetak hitungan operasi Persamaan (6.5). `least_blend` menyelesaikan masalah empat hara Persamaan (6.7) dengan persamaan normal plus Cholesky dan dengan QR Householder, memperlihatkan hara mana yang meleset, lalu mengulangi perbandingannya pada kolom-kolom polinomial yang makin buruk kondisinya seiring naiknya derajat.

6.6.1 LU dalam Python dan Java

Kode 6.1 mengikuti **Pseudokode 6.1** baris demi baris. `lu_decompose` mengembalikan faktor yang dikemas, permutasi, dan tanda; `lu_solve` adalah pasangan substitusi Persamaan (6.3); `main` membongkar L dan U hanya untuk dicetak. Hitungan kerjanya memakai suku-suku utama $n^3/3$ dan n^2.

Kode 6.1 `lu_blend.py`: dekomposisi LU dengan *pivoting* parsial, determinan, dan campuran untuk setiap blok dari satu faktorisasi.

```
"""Campuran pupuk untuk empat blok kebun: faktorkan A = LU sekali (dengan
pivoting parsial), lalu selesaikan tiap blok dengan dua segitiga murah."""

NAMES = ["NPK 12-12-17", "NPK 20-10-10", "Urea 46-0-0"]
# baris: gram N, P2O5, K2O dalam 100 g tiap pupuk (satu pupuk per kolom)
A = [[12.0, 20.0, 46.0],
     [12.0, 10.0, 0.0],
     [17.0, 10.0, 0.0]]
BLOCKS = [("young trees", [150.0, 60.0, 70.0]),
          ("after harvest", [180.0, 90.0, 110.0]),
          ("before flowering", [120.0, 90.0, 120.0]),
          ("fruit filling", [100.0, 80.0, 110.0])]



def total(values):
    """Jumlah biasa dari kiri ke kanan. sum() Python 3.12+ mengompensasi
    pembulatan, sehingga digit kita akan berbeda dari versi Java."""
    s = 0.0
    for v in values:
        s += v
    return s

def lu_decompose(a):
    """Kembalikan (lu, perm, sign): L di bawah diagonal (diagonal satuan
    tidak disimpan), U pada dan di atasnya, perm[i] = baris asal di baris i."""
    n = len(a)
    lu = [row[:] for row in a]
    perm = list(range(n))
    sign = 1.0
    for k in range(n - 1):
        p = max(range(k, n), key=lambda i: abs(lu[i][k]))   # pivot parsial
        if lu[p][k] == 0.0:
            raise ValueError("matrix is singular")
        if p != k:
            lu[k], lu[p] = lu[p], lu[k]
            perm[k], perm[p] = perm[p], perm[k]
            sign = -sign
        for i in range(k + 1, n):
            m = lu[i][k] / lu[k][k]
            lu[i][k] = m                    # simpan pengalinya: itulah L
            for j in range(k + 1, n):
                lu[i][j] -= m * lu[k][j]
    return lu, perm, sign
```

Kode 6.1 (Lanjutan) *lu_blend.py: dekomposisi LU dengan pivoting parsial, determinan, dan campuran untuk setiap blok dari satu faktorisasi.*

```
def lu_solve(lu, perm, b):
    """Selesaikan Ly = Pb (maju), lalu Ux = y (mundur)."""
    n = len(lu)
    y = [0.0] * n
    for i in range(n):
        y[i] = b[perm[i]] - total(lu[i][j] * y[j] for j in range(i))
    x = [0.0] * n
    for i in range(n - 1, -1, -1):
        s = y[i] - total(lu[i][j] * x[j] for j in range(i + 1, n))
        x[i] = s / lu[i][i]
    return x


def show(title, m):
    print(title)
    for row in m:
        print("  " + "".join(f"{v:10.4f}" for v in row))


def main():
    n = len(A)
    lu, perm, sign = lu_decompose(A)
    p = [[1.0 if j == perm[i] else 0.0 for j in range(n)] for i in range(n)]
    low = [[lu[i][j] if j < i else (1.0 if i == j else 0.0)
            for j in range(n)] for i in range(n)]
    up = [[lu[i][j] if j >= i else 0.0 for j in range(n)] for i in range(n)]
    show("P", p)
    show("L", low)
    show("U", up)
    err = max(abs(total(low[i][k] * up[k][j] for k in range(n))
                  - A[perm[i]][j]) for i in range(n) for j in range(n))
    print(f"max |PA - LU| = {err:.1e}")
    det = sign
    for i in range(n):
        det *= lu[i][i]
    print(f"det A = {det:.4f}")
    print()
    print("grams per tree         " + "".join(f"{s:>14s}" for s in NAMES))
    for name, b in BLOCKS:
        x = lu_solve(lu, perm, b)
        print(f"{name:23s}" + "".join(f"{100 * v:14.1f}" for v in x))
    print()
    print("work for 100 right-hand sides (multiplications, leading terms)")
    for size in (3, 100, 1000):
        again = 100 * (size ** 3 / 3 + size ** 2)
        once = size ** 3 / 3 + 100 * size ** 2
```

Kode 6.1 (Lanjutan) *lu_blend.py: dekomposisi LU dengan pivoting parsial, determinan, dan campuran untuk setiap blok dari satu faktorisasi.*

```
        print(f"  n = {size:4d}: eliminate each time {again:14.0f},"
              f" factor once {once:14.0f}")


if __name__ == "__main__":
    main()
```

Kode 6.2 adalah versi Java-nya; faktor-faktornya disimpan dalam field statis agar `luSolve` dapat dipanggil sekali per blok.

Kode 6.2 `LuBlend.java`: program faktorkan-sekali, selesaikan-berkali-kali yang sama dalam Java.

```java
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;

/** Campuran pupuk untuk empat blok kebun: faktorkan A = LU sekali (dengan
 *  pivoting parsial), lalu selesaikan tiap blok dengan dua segitiga murah. */
public class LuBlend {
    static final String[] NAMES = {"NPK 12-12-17", "NPK 20-10-10", "Urea 46-0-0"};
    // baris: gram N, P2O5, K2O dalam 100 g tiap pupuk (satu pupuk per kolom)
    static final double[][] A = {{12, 20, 46}, {12, 10, 0}, {17, 10, 0}};
    static final String[] BLOCK_NAMES = {"young trees", "after harvest",
                                         "before flowering", "fruit filling"};
    static final double[][] BLOCKS = {{150, 60, 70}, {180, 90, 110},
                                      {120, 90, 120}, {100, 80, 110}};

    /** LU dikemas dalam satu matriks: L di bawah diagonal (diagonal satuan
     *  tidak disimpan), U pada dan di atasnya; perm[i] = baris asal di baris i. */
    static double[][] lu;
    static int[] perm;
    static double sign;

    static void luDecompose(double[][] a) {
        int n = a.length;
        lu = new double[n][];
        for (int i = 0; i < n; i++) lu[i] = a[i].clone();
        perm = new int[n];
        for (int i = 0; i < n; i++) perm[i] = i;
        sign = 1.0;
        for (int k = 0; k < n - 1; k++) {
            int p = k;                                      // pivot parsial
            for (int i = k + 1; i < n; i++) {
                if (Math.abs(lu[i][k]) > Math.abs(lu[p][k])) p = i;
            }
            if (lu[p][k] == 0.0) throw new ArithmeticException("singular");
            if (p != k) {
                double[] t = lu[k]; lu[k] = lu[p]; lu[p] = t;
                int q = perm[k]; perm[k] = perm[p]; perm[p] = q;
                sign = -sign;
            }
            for (int i = k + 1; i < n; i++) {
                double m = lu[i][k] / lu[k][k];
                lu[i][k] = m;                       // simpan pengalinya: itulah L
                for (int j = k + 1; j < n; j++) lu[i][j] -= m * lu[k][j];
            }
        }
    }
```

Kode 6.2 (Lanjutan) *LuBlend.java: program faktorkan-sekali, selesaikan-berkali-kali yang sama dalam Java.*

```
    /** Selesaikan Ly = Pb (maju), lalu Ux = y (mundur). */
    static double[] luSolve(double[] b) {
        int n = lu.length;
        double[] y = new double[n];
        for (int i = 0; i < n; i++) {
            double s = 0;
            for (int j = 0; j < i; j++) s += lu[i][j] * y[j];
            y[i] = b[perm[i]] - s;
        }
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double s = 0;
            for (int j = i + 1; j < n; j++) s += lu[i][j] * x[j];
            x[i] = (y[i] - s) / lu[i][i];
        }
        return x;
    }

    static void show(String title, double[][] m) {
        System.out.println(title);
        for (double[] row : m) {
            StringBuilder line = new StringBuilder("  ");
            for (double v : row) line.append(f(v, 4, 10));
            System.out.println(line);
        }
    }

    public static void main(String[] args) {
        int n = A.length;
        luDecompose(A);
        double[][] p = new double[n][n], low = new double[n][n],
                   up = new double[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                p[i][j] = j == perm[i] ? 1.0 : 0.0;
                low[i][j] = j < i ? lu[i][j] : (i == j ? 1.0 : 0.0);
                up[i][j] = j >= i ? lu[i][j] : 0.0;
            }
        }
        show("P", p);
        show("L", low);
        show("U", up);
        double err = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
```

Kode 6.2 (Lanjutan) *LuBlend.java: program faktorkan-sekali, selesaikan-berkali-kali yang sama dalam Java.*

```
                double s = 0;
                for (int k = 0; k < n; k++) s += low[i][k] * up[k][j];
                err = Math.max(err, Math.abs(s - A[perm[i]][j]));
            }
        }
        System.out.println("max |PA - LU| = " + e(err, 1));
        double det = sign;
        for (int i = 0; i < n; i++) det *= lu[i][i];
        System.out.println("det A = " + f(det, 4));
        System.out.println();
        StringBuilder head = new StringBuilder("grams per tree         ");
        for (String s : NAMES) head.append(pad(s, 14));
        System.out.println(head);
        for (int b = 0; b < BLOCKS.length; b++) {
            double[] x = luSolve(BLOCKS[b]);
            StringBuilder line = new StringBuilder(padRight(BLOCK_NAMES[b], 23));
            for (double v : x) line.append(f(100 * v, 1, 14));
            System.out.println(line);
        }
        System.out.println();
        System.out.println("work for 100 right-hand sides"
                           + " (multiplications, leading terms)");
        for (long size : new long[] {3, 100, 1000}) {
            double again = 100 * ((double) (size * size * size) / 3
                                  + size * size);
            double once = (double) (size * size * size) / 3 + 100 * size * size;
            System.out.println("  n = " + pad("" + size, 4)
                    + ": eliminate each time " + f(again, 0, 14)
                    + ", factor once " + f(once, 0, 14));
        }
    }

    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }
```

Kode 6.2 (Lanjutan) *LuBlend.java: program faktorkan-sekali, selesaikan-berkali-kali yang sama dalam Java.*

```
    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                  + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }

    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }

    static String padRight(String s, int w) {
        return s + " ".repeat(Math.max(0, w - s.length()));
    }
}
```

6.6.2 Cholesky dan QR dalam Python dan Java

Kode 6.3 memiliki satu fungsi per rumus: `normal_equations` membangun $A^{\mathsf{T}}A$ dan $A^{\mathsf{T}}b$, `cholesky` adalah Persamaan (6.10) dan memunculkan galat bila suatu pivot tidak positif, `cholesky_solve` mengerjakan dua segitiga, dan `householder_lstsq` menerapkan refleksi Persamaan (6.13) kolom demi kolom pada salinan A dan pada b, lalu melakukan substitusi mundur dan mengembalikan $\| c_2 \|$ dari Persamaan (6.12). Fungsi `vandermonde` membangun kolom $1, t, t^2, \ldots, t^d$ pada 20 titik berjarak sama, yaitu kolom-kolom pencocokan polinomial, dengan pangkat yang dibentuk lewat perkalian berulang agar kedua bahasa menghitung bilangan yang identik.

Kode 6.3 `least_blend.py`: kuadrat terkecil dengan persamaan normal plus Cholesky dan dengan QR Householder, untuk kebun dan untuk kolom polinomial.

```
"""Empat target hara, tiga pupuk: campuran terbaik dalam arti kuadrat
terkecil, dengan persamaan normal + Cholesky dan dengan QR Householder."""
import math

NUTRIENTS = ["N", "P2O5", "K2O", "MgO"]
# gram tiap hara dalam 100 g NPK 12-12-17-2, NPK 20-10-10, urea
A = [[12.0, 20.0, 46.0],
     [12.0, 10.0, 0.0],
     [17.0, 10.0, 0.0],
     [2.0, 0.0, 0.0]]
B = [150.0, 60.0, 70.0, 6.0]      # gram per pohon muda



def total(values):
    """Jumlah biasa dari kiri ke kanan. sum() Python 3.12+ mengompensasi
    pembulatan, sehingga digit kita akan berbeda dari versi Java."""
    s = 0.0
    for v in values:
        s += v
    return s

def normal_equations(a, b):
    """Kembalikan A^T A dan A^T b."""
    m, n = len(a), len(a[0])
    ata = [[total(a[k][i] * a[k][j] for k in range(m)) for j in range(n)]
           for i in range(n)]
    atb = [total(a[k][i] * b[k] for k in range(m)) for i in range(n)]
    return ata, atb


def cholesky(s):
    """Segitiga bawah L dengan L L^T = s; s harus simetris definit
    positif; jika tidak, ada pivot tak positif dan kita berhenti."""
    n = len(s)
    low = [[0.0] * n for _ in range(n)]
    for j in range(n):
        d = s[j][j] - total(low[j][k] * low[j][k] for k in range(j))
        if d <= 0.0:
            raise ValueError(f"pivot {j + 1} is {d:.3e}: not positive")
        low[j][j] = math.sqrt(d)
        for i in range(j + 1, n):
            t = s[i][j] - total(low[i][k] * low[j][k] for k in range(j))
            low[i][j] = t / low[j][j]
    return low
```

Kode 6.3 (Lanjutan) *least_blend.py: kuadrat terkecil dengan persamaan normal plus Cholesky dan dengan QR Householder, untuk kebun dan untuk kolom polinomial.*

```
def cholesky_solve(low, b):
    """Selesaikan L y = b, lalu L^T x = y."""
    n = len(low)
    y = [0.0] * n
    for i in range(n):
        y[i] = (b[i] - total(low[i][k] * y[k] for k in range(i))) / low[i][i]
    x = [0.0] * n
    for i in range(n - 1, -1, -1):
        s = total(low[k][i] * x[k] for k in range(i + 1, n))
        x[i] = (y[i] - s) / low[i][i]
    return x


def householder_lstsq(a, b):
    """Kuadrat terkecil dengan QR Householder: kembalikan x dan norma residu."""
    m, n = len(a), len(a[0])
    r = [row[:] for row in a]
    c = b[:]                                    # menjadi Q^T b
    for k in range(n):
        norm = math.sqrt(total(r[i][k] * r[i][k] for i in range(k, m)))
        alpha = -norm if r[k][k] >= 0.0 else norm   # hindari pembatalan
        v = [0.0] * m
        for i in range(k, m):
            v[i] = r[i][k]
        v[k] -= alpha
        vv = total(v[i] * v[i] for i in range(k, m))
        if vv == 0.0:
            continue
        for j in range(k, n):
            f = 2.0 * total(v[i] * r[i][j] for i in range(k, m)) / vv
            for i in range(k, m):
                r[i][j] -= f * v[i]
        f = 2.0 * total(v[i] * c[i] for i in range(k, m)) / vv
        for i in range(k, m):
            c[i] -= f * v[i]
    x = [0.0] * n
    for i in range(n - 1, -1, -1):
        s = total(r[i][j] * x[j] for j in range(i + 1, n))
        x[i] = (c[i] - s) / r[i][i]
    rest = math.sqrt(total(c[i] * c[i] for i in range(n, m)))
    return x, rest
```

Kode 6.3 (Lanjutan) *least_blend.py: kuadrat terkecil dengan persamaan normal plus Cholesky dan dengan QR Householder, untuk kebun dan untuk kolom polinomial.*

```
def vandermonde(m, degree):
    """m titik t berjarak sama di [0, 1], kolom 1, t, ..., t^degree."""
    rows = []
    for i in range(m):
        t = i / (m - 1)
        row = [1.0]
        for _ in range(degree):
            row.append(row[-1] * t)          # pangkat lewat kali berulang
        rows.append(row)
    return rows


def main():
    ata, atb = normal_equations(A, B)
    print("A^T A and A^T b")
    for row, v in zip(ata, atb):
        print("  " + "".join(f"{x:9.1f}" for x in row) + f"   | {v:9.1f}")
    low = cholesky(ata)
    print("Cholesky factor L")
    for row in low:
        print("  " + "".join(f"{x:9.4f}" for x in row))
    x1 = cholesky_solve(low, atb)
    x2, rest = householder_lstsq(A, B)
    print("grams per tree       normal+Cholesky  Householder QR")
    for name, u, v in zip(["NPK 12-12-17-2", "NPK 20-10-10", "Urea"], x1, x2):
        print(f"  {name:18s}{100 * u:15.3f}{100 * v:16.3f}")
    print("nutrient  target  delivered  miss")
    for i, name in enumerate(NUTRIENTS):
        got = total(A[i][j] * x2[j] for j in range(3))
        print(f"  {name:6s}{B[i]:8.1f}{got:11.2f}{got - B[i]:+7.2f}")
    print(f"residual norm {rest:.4f}")
    print()
    print("Polynomial columns t^0..t^d at 20 points, true x = all ones")
    print(" d   normal+Cholesky error   Householder QR error")
    for d in (3, 6, 9, 12):
        a = vandermonde(20, d)
        b = [total(row) for row in a]
        xq, _ = householder_lstsq(a, b)
        eq = max(abs(v - 1.0) for v in xq)
        try:
            s, sb = normal_equations(a, b)
            xn = cholesky_solve(cholesky(s), sb)
            en = f"{max(abs(v - 1.0) for v in xn):22.2e}"
        except ValueError:
            en = f"{'fails (pivot <= 0)':>22s}"
        print(f"{d:2d}{en}{eq:23.2e}")
```

Kode 6.3 (Lanjutan) *least_blend.py: kuadrat terkecil dengan persamaan normal plus Cholesky dan dengan QR Householder, untuk kebun dan untuk kolom polinomial.*

```


if __name__ == "__main__":
    main()
```

Kode 6.4 adalah kembarannya dalam Java. Metode Java mengembalikan satu nilai, sehingga `householderLstsq` mengembalikan x dan menyimpan norma residu $\| c_2 \|$ di field statis `lastRest`, dan `normalEquations` menulis $A^{\mathsf{T}}b$ ke larik yang diberikan pemanggil. Kegagalan Cholesky diisyaratkan dengan `ArithmeticException`, yang ditangkap `main` untuk mencetak pesan yang sama dengan versi Python.

Kode 6.4 `LeastBlend.java`: kedua metode kuadrat terkecil yang sama dalam Java.

```
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;

/** Empat target hara, tiga pupuk: campuran terbaik dalam arti kuadrat
 *  terkecil, dengan persamaan normal + Cholesky dan dengan QR Householder. */
public class LeastBlend {
    static final String[] NUTRIENTS = {"N", "P2O5", "K2O", "MgO"};
    // gram tiap hara dalam 100 g NPK 12-12-17-2, NPK 20-10-10, urea
    static final double[][] A = {{12, 20, 46}, {12, 10, 0}, {17, 10, 0},
                                 {2, 0, 0}};
    static final double[] B = {150, 60, 70, 6};     // gram per pohon muda

    /** A^T A (dikembalikan) dan A^T b (ditulis ke atb). */
    static double[][] normalEquations(double[][] a, double[] b, double[] atb) {
        int m = a.length, n = a[0].length;
        double[][] ata = new double[n][n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                double s = 0;
                for (int k = 0; k < m; k++) s += a[k][i] * a[k][j];
                ata[i][j] = s;
            }
            double s = 0;
            for (int k = 0; k < m; k++) s += a[k][i] * b[k];
            atb[i] = s;
        }
        return ata;
    }

    /** Segitiga bawah L dengan L L^T = s; s harus simetris definit
     *  positif; jika tidak, ada pivot tak positif dan kita berhenti. */
    static double[][] cholesky(double[][] s) {
        int n = s.length;
        double[][] low = new double[n][n];
        for (int j = 0; j < n; j++) {
            double sum = 0;
            for (int k = 0; k < j; k++) sum += low[j][k] * low[j][k];
            double d = s[j][j] - sum;
            if (d <= 0.0) {
                throw new ArithmeticException("pivot " + (j + 1)
                                              + " is not positive");
            }
            low[j][j] = Math.sqrt(d);
            for (int i = j + 1; i < n; i++) {
                double t = 0;
```

Kode 6.4 (Lanjutan) *LeastBlend.java: kedua metode kuadrat terkecil yang sama dalam Java.*

```
                for (int k = 0; k < j; k++) t += low[i][k] * low[j][k];
                low[i][j] = (s[i][j] - t) / low[j][j];
            }
        }
        return low;
    }

    /** Selesaikan L y = b, lalu L^T x = y. */
    static double[] choleskySolve(double[][] low, double[] b) {
        int n = low.length;
        double[] y = new double[n];
        for (int i = 0; i < n; i++) {
            double s = 0;
            for (int k = 0; k < i; k++) s += low[i][k] * y[k];
            y[i] = (b[i] - s) / low[i][i];
        }
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double s = 0;
            for (int k = i + 1; k < n; k++) s += low[k][i] * x[k];
            x[i] = (y[i] - s) / low[i][i];
        }
        return x;
    }

    /** Kuadrat terkecil dengan QR Householder: mengembalikan x; norma residu
     *  disimpan di lastRest. */
    static double lastRest;

    static double[] householderLstsq(double[][] a, double[] b) {
        int m = a.length, n = a[0].length;
        double[][] r = new double[m][];
        for (int i = 0; i < m; i++) r[i] = a[i].clone();
        double[] c = b.clone();                    // menjadi Q^T b
        for (int k = 0; k < n; k++) {
            double ss = 0;
            for (int i = k; i < m; i++) ss += r[i][k] * r[i][k];
            double norm = Math.sqrt(ss);
            double alpha = r[k][k] >= 0.0 ? -norm : norm; // hindari pembatalan
            double[] v = new double[m];
            for (int i = k; i < m; i++) v[i] = r[i][k];
            v[k] -= alpha;
            double vv = 0;
            for (int i = k; i < m; i++) vv += v[i] * v[i];
            if (vv == 0.0) continue;
            for (int j = k; j < n; j++) {
```

Kode 6.4 (Lanjutan) *LeastBlend.java: kedua metode kuadrat terkecil yang sama dalam Java.*

```
                double s = 0;
                for (int i = k; i < m; i++) s += v[i] * r[i][j];
                double f = 2.0 * s / vv;
                for (int i = k; i < m; i++) r[i][j] -= f * v[i];
            }
            double s = 0;
            for (int i = k; i < m; i++) s += v[i] * c[i];
            double f = 2.0 * s / vv;
            for (int i = k; i < m; i++) c[i] -= f * v[i];
        }
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double s = 0;
            for (int j = i + 1; j < n; j++) s += r[i][j] * x[j];
            x[i] = (c[i] - s) / r[i][i];
        }
        double rest = 0;
        for (int i = n; i < m; i++) rest += c[i] * c[i];
        lastRest = Math.sqrt(rest);
        return x;
    }

    /** m titik t berjarak sama di [0, 1], kolom 1, t, ..., t^degree. */
    static double[][] vandermonde(int m, int degree) {
        double[][] rows = new double[m][degree + 1];
        for (int i = 0; i < m; i++) {
            double t = (double) i / (m - 1);
            rows[i][0] = 1.0;
            for (int j = 1; j <= degree; j++) {
                rows[i][j] = rows[i][j - 1] * t;  // pangkat lewat kali berulang
            }
        }
        return rows;
    }

    public static void main(String[] args) {
        double[] atb = new double[3];
        double[][] ata = normalEquations(A, B, atb);
        System.out.println("A^T A and A^T b");
        for (int i = 0; i < 3; i++) {
            StringBuilder line = new StringBuilder("  ");
            for (double v : ata[i]) line.append(f(v, 1, 9));
            System.out.println(line + "   | " + f(atb[i], 1, 9));
        }
        double[][] low = cholesky(ata);
        System.out.println("Cholesky factor L");
```

Kode 6.4 (Lanjutan) *LeastBlend.java: kedua metode kuadrat terkecil yang sama dalam Java.*

```
        for (double[] row : low) {
            StringBuilder line = new StringBuilder("  ");
            for (double v : row) line.append(f(v, 4, 9));
            System.out.println(line);
        }
        double[] x1 = choleskySolve(low, atb);
        double[] x2 = householderLstsq(A, B);
        double rest = lastRest;
        System.out.println("grams per tree       normal+Cholesky  Householder QR");
        String[] names = {"NPK 12-12-17-2", "NPK 20-10-10", "Urea"};
        for (int j = 0; j < 3; j++) {
            System.out.println("  " + padRight(names[j], 18) + f(100 * x1[j], 3, 15)
                               + f(100 * x2[j], 3, 16));
        }
        System.out.println("nutrient  target  delivered  miss");
        for (int i = 0; i < 4; i++) {
            double got = 0;
            for (int j = 0; j < 3; j++) got += A[i][j] * x2[j];
            String miss = f(got - B[i], 2);
            if (!miss.startsWith("-")) miss = "+" + miss;
            System.out.println("  " + padRight(NUTRIENTS[i], 6) + f(B[i], 1, 8)
                               + f(got, 2, 11) + pad(miss, 7));
        }
        System.out.println("residual norm " + f(rest, 4));
        System.out.println();
        System.out.println("Polynomial columns t^0..t^d at 20 points,"
                           + " true x = all ones");
        System.out.println(" d   normal+Cholesky error   Householder QR error");
        for (int d : new int[] {3, 6, 9, 12}) {
            double[][] a = vandermonde(20, d);
            double[] b = new double[20];
            for (int i = 0; i < 20; i++) {
                double s = 0;
                for (double v : a[i]) s += v;
                b[i] = s;
            }
            double eq = maxError(householderLstsq(a, b));
            String en;
            try {
                double[] sb = new double[d + 1];
                double[][] s = normalEquations(a, b, sb);
                en = pad(e(maxError(choleskySolve(cholesky(s), sb)), 2), 22);
            } catch (ArithmeticException err) {
                en = pad("fails (pivot <= 0)", 22);
            }
            System.out.println(pad("" + d, 2) + en + pad(e(eq, 2), 23));
```

Kode 6.4 (Lanjutan) *LeastBlend.java: kedua metode kuadrat terkecil yang sama dalam Java.*

```
        }
    }

    static double maxError(double[] x) {
        double worst = 0;
        for (double v : x) worst = Math.max(worst, Math.abs(v - 1.0));
        return worst;
    }

    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }

    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                    + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }

    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }

    static String padRight(String s, int w) {
        return s + " ".repeat(Math.max(0, w - s.length()));
    }
}
```

6.7 Menjalankannya

Kompilasikan berkas Java dengan `javac -d out *.java` di folder `java`, lalu jalankan `java -cp out LuBlend` dan `java -cp out LeastBlend`; di folder `python` jalankan `python3 lu_blend.py` dan `python3 least_blend.py`. Kedua bahasa mencetak teks yang persis sama. Program LU lebih dahulu:

```
P
      0.0000    0.0000    1.0000
      1.0000    0.0000    0.0000
      0.0000    1.0000    0.0000
L
      1.0000    0.0000    0.0000
      0.7059    1.0000    0.0000
      0.7059    0.2273    1.0000
U
     17.0000   10.0000    0.0000
      0.0000   12.9412   46.0000
      0.0000    0.0000  -10.4545
max |PA - LU| = 0.0e+00
det A = -2300.0000

grams per tree           NPK 12-12-17  NPK 20-10-10   Urea 46-0-0
young trees                     200.0         360.0         117.4
after harvest                   400.0         420.0         104.3
before flowering                600.0         180.0          26.1
fruit filling                   600.0          80.0          26.1

work for 100 right-hand sides (multiplications, leading terms)
  n =    3: eliminate each time           1800, factor once            909
  n =  100: eliminate each time       34333333, factor once        1333333
  n = 1000: eliminate each time    33433333333, factor once      433333333
```

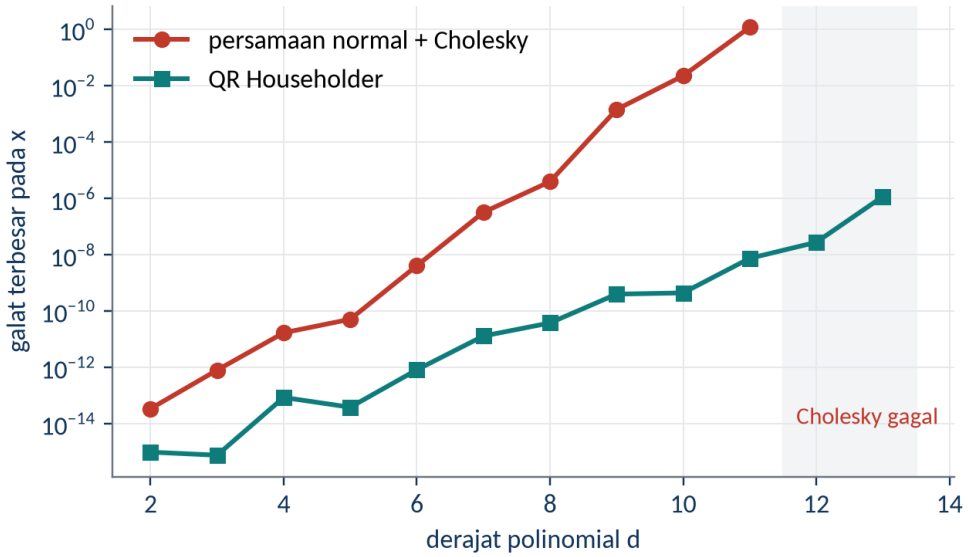
P, L, dan U adalah yang kita temukan dengan tangan di Subbab 6.2.2, dan $PA - LU$ tepat nol. Determinannya -2300. Tabel campuran memuat 200, 360, dan 117.4 g untuk pohon muda dari Subbab 6.2.3 serta tiga blok lagi seharga enam substitusi. Perhatikan baris “before flowering” (menjelang berbunga) dan “fruit filling” (pengisian buah): keduanya memerlukan urea yang sama banyak (26.1 g) dan NPK 12-12-17 yang sama (600 g), dan hanya berbeda pada NPK 20-10-10. Rekomendasinya berselisih (20,10,10), yang tepat sama dengan satu satuan NPK 20-10-10, sehingga selisih campurannya 100 g karung itu. Sistem linear menghargai cara membaca seperti ini. Tabel kerja memperlihatkan gagasan yang sama dalam skala besar: untuk 100 ruas kanan dan $n = 1000$, 33 miliar perkalian berbanding 0.43 miliar.

Program kuadrat terkecil:

```
A^T A and A^T b
      581.0    530.0    552.0   |    3722.0
      530.0    600.0    920.0   |    4300.0
      552.0    920.0   2116.0   |    6900.0
Cholesky factor L
    24.1039   0.0000   0.0000
    21.9881  10.7946   0.0000
    22.9008  38.5799  10.1559
grams per tree       normal+Cholesky  Householder QR
  NPK 12-12-17-2             224.242         224.242
  NPK 20-10-10               324.848         324.848
  Urea                       126.350         126.350
nutrient  target  delivered  miss
  N        150.0     150.00  +0.00
  P2O5      60.0      59.39  -0.61
  K2O       70.0      70.61  +0.61
  MgO        6.0       4.48  -1.52
residual norm 1.7408

Polynomial columns t^0..t^d at 20 points, true x = all ones
 d   normal+Cholesky error   Householder QR error
 3              8.14e-13               7.77e-16
 6              4.14e-09               8.35e-13
 9              1.45e-03               4.05e-10
12    fails (pivot <= 0)               2.81e-08
```

$A^{\mathsf{T}}A$ dan L yang dicetak adalah yang dihitung dengan tangan di Subbab 6.4. Kedua metode memberi campuran yang sama sampai tiga desimal yang ditampilkan: 224.242 g NPK 12-12-17-2, 324.848 g NPK 20-10-10, dan 126.350 g urea. Tabel melesetnya layak diperhatikan. Nitrogen dipenuhi tepat, karena urea tidak berisi apa pun selain nitrogen: apa pun yang dilakukan dua karung lainnya, sejumlah urea dapat membetulkan baris N tanpa mengganggu yang lain, sehingga kompromi terbaik tidak pernah meleset di sana. Fosfat dan kalium meleset masing-masing 0.61 g, ke arah yang berlawanan, dan magnesium 1.52 g: campurannya memberi 4.48 g alih-alih 6. Kuadrat terkecil telah menyebarkan meleset yang tak terhindarkan ke tempat yang biayanya paling kecil dalam kuadrat gram. Apakah 1.5 g magnesium lebih penting daripada 0.6 g fosfat adalah pertanyaan agronomi, bukan numerik; Soal 4 menunjukkan cara memberitahukannya kepada program dengan memberi bobot pada baris-barisnya.

Tabel kedua adalah pelajaran ketelitian. Solusi eksak setiap masalah polinomial diketahui (semuanya satu), jadi galat yang tercetak adalah galat sungguhan. Pada derajat 3 kedua metode sangat baik. Pada derajat 6 persamaan normal kehilangan tiga digit lebih banyak daripada QR; pada derajat 9 persamaan normal salah di desimal ketiga (1.45×10^{-3}) sementara QR masih memiliki sepuluh digit benar (4.05×10^{-10}); pada derajat 12 pembulatan telah membuat $A^{\mathsf{T}}A$ secara numerik tak definit, sebuah pivot Cholesky bernilai negatif, dan persamaan normal sama sekali tidak memberi jawaban, sementara QR masih memberi delapan digit. **Gambar 6.4** memperlihatkan seluruh rentang derajat. Skrip pembanding di subbab berikutnya mengukur $\kappa(A) = 3.79 \times 10^{6}$ pada derajat 9, sehingga Persamaan (6.11) meramalkan galat sekitar $\kappa(A)^2 \varepsilon_{\text{mach}} = 1.44 \times 10^{13} \times 1.1 \times 10^{-16} \approx 1.6 \times 10^{-3}$ untuk persamaan normal dan $\kappa(A)\,\varepsilon_{\text{mach}} \approx 4 \times 10^{-10}$ untuk QR. Kedua ramalan itu tepat mengenai galat yang terukur.



Gambar 6.4 Galat terbesar solusi kuadrat terkecil untuk kolom polinomial derajat d pada 20 titik: persamaan normal kehilangan sekitar dua kali lebih banyak digit daripada QR Householder, dan mulai derajat 12 Cholesky gagal.

6.8 Memeriksa dengan Pustaka

SciPy 1.18, rilis terbaru pada September 2026, menulis ulang `scipy.linalg.lu` dan `det` dalam C++ dengan dukungan batch matriks serta mempercepat `cholesky` [8], [9]; antarmuka yang kita pakai tidak berubah. **Kode 6.5** meminta faktorisasi yang sama dari SciPy. Perhatikan satu perbedaan konvensi: `scipy.linalg.lu` mengembalikan $A = PLU$, dengan permutasi di sisi yang lain, sehingga P-nya adalah transpos dari milik kita [10].

Kode 6.5 `compare_scipy.py`: faktor dan solusi kita dibandingkan dengan `lu`, `lu_factor`/`lu_solve`, `cho_factor`/`cho_solve` dari SciPy serta `lstsq` dan `qr` dari NumPy.

```
"""Periksa campuran dengan LU, Cholesky SciPy dan kuadrat terkecil NumPy."""
import numpy as np
import scipy
from scipy.linalg import cho_factor, cho_solve, lu, lu_factor, lu_solve

import least_blend
import lu_blend
from lu_blend import A, BLOCKS


def main():
    print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
    a = np.array(A)
    p, low, up = lu(a)                    # SciPy mengembalikan A = P L U
    print("scipy.linalg.lu: U diagonal", np.round(np.diag(up), 4))
    factor = lu_factor(a)
    ours, perm, _ = lu_blend.lu_decompose(A)
    worst = 0.0
    for _, b in BLOCKS:
        x = lu_solve(factor, np.array(b))
        mine = lu_blend.lu_solve(ours, perm, b)
        worst = max(worst, np.max(np.abs(x - np.array(mine))))
    print(f"lu_solve vs ours, all four blocks: max difference {worst:.1e}")

    m = np.array(least_blend.A)
    b = np.array(least_blend.B)
    x, res, rank, sv = np.linalg.lstsq(m, b)
    print("numpy.linalg.lstsq grams:", np.round(100 * x, 3),
          f" residual norm {np.sqrt(res[0]):.4f}")
    c = cho_factor(m.T @ m, lower=True)
    print("cho_solve grams:        ", np.round(100 * cho_solve(c, m.T @ b), 3))
    q, r = np.linalg.qr(m)
    print("numpy.linalg.qr R diagonal:", np.round(np.diag(r), 4))

    v = np.vander(np.linspace(0.0, 1.0, 20), 10, increasing=True)
    print(f"degree 9 columns: cond(A) = {np.linalg.cond(v):.2e},"
          f" cond(A^T A) = {np.linalg.cond(v.T @ v):.2e}")


if __name__ == "__main__":
    main()
```

```
NumPy 2.5.3, SciPy 1.18.1
scipy.linalg.lu: U diagonal [ 17.      12.9412 -10.4545]
lu_solve vs ours, all four blocks: max difference 6.2e-15
numpy.linalg.lstsq grams: [224.242 324.848 126.35 ]  residual norm 1.7408
cho_solve grams:         [224.242 324.848 126.35 ]
numpy.linalg.qr R diagonal: [-24.1039  10.7946 -10.1559]
degree 9 columns: cond(A) = 3.79e+06, cond(A^T A) = 1.44e+13
```

Diagonal U sama, keempat blok cocok dengan `lu_solve` sampai 6×10^{-15}, dan `numpy.linalg.lstsq` (yang memakai dekomposisi nilai singular, kerabat dekat QR) memberi campuran yang sama dan norma residu yang sama, 1.7408. R dari NumPy memiliki diagonal yang sama dengan faktor Cholesky kita sampai tandanya, seperti diramalkan Subbab 6.5.4, dan bilangan kondisi pada derajat 9, 3.79×10^{6} dan 1.44×10^{13}, membenarkan Persamaan (6.11). Dalam praktik, panggillah rutin-rutin pustaka ini: mereka memakai algoritma berblok yang membuat *cache* prosesor tetap sibuk dan berjalan ratusan kali lebih cepat daripada gelung kita pada matriks besar [3]. Program kita ada agar Anda tahu apa yang dikerjakan pustaka dan dapat menilai kapan jawabannya layak dipercaya.

6.9 Membuatnya Lebih Baik

- **Gram negatif.** Baik LU maupun QR tidak tahu bahwa pupuk tidak mungkin negatif. Empat blok musim ini kebetulan memberi campuran positif, tetapi rekomendasi (150, 100, 120) memberi -4.35 g urea. Alat yang tepat adalah *kuadrat terkecil nonnegatif*, yang mencari campuran terbaik di antara yang $x \geq 0$; SciPy menyediakannya sebagai `scipy.optimize.nnls`, berdasarkan algoritma Lawson dan Hanson [11]. Bab 13 membahas optimasi berkendala secara lebih umum.
- **Matriks tridiagonal dan berpita.** Bila hanya beberapa diagonal yang tak nol, LU tidak mengisi apa pun di luar pita, dan biayanya turun dari $n^3/3$ menjadi sekitar nb^2 untuk lebar pita b. Kasus tridiagonal, dengan $b = 1$, adalah *algoritma Thomas*, dan Bab 7 memakainya untuk spline dengan biaya $O(n)$.
- **Penghalusan iteratif.** Setelah menyelesaikan $Ax = b$, hitung residu $r = b - Ax$ (dengan presisi lebih tinggi bila mungkin), selesaikan $A\,d = r$ dengan faktor yang sama, yang murah, dan koreksi $x \leftarrow x + d$. Satu atau dua putaran memulihkan sebagian besar digit yang direnggut pembulatan, karena faktor-faktornya dipakai ulang [4].
- **LU jarang.** Untuk matriks jarang yang besar, seperti wajan di Bab 5, urutan eliminasi variabel menentukan berapa banyak nol yang terisi. Pemecah langsung jarang, seperti SuperLU di balik `spsolve` SciPy, lebih dahulu menyusun ulang variabel agar pengisiannya tetap sedikit, baru kemudian memfaktorkan [3].
- **Memperbarui alih-alih memfaktorkan ulang.** Bila satu pupuk diganti dengan pupuk lain, hanya satu kolom A yang berubah. Ada rumus yang memperbarui LU atau QR untuk perubahan seperti itu dalam $O(n^2)$ alih-alih memfaktorkan ulang dalam $O(n^3)$ [3].

6.10 Memakainya dengan Sungguh-Sungguh

Untuk memakai `lu_blend.py` bagi kebun, sawah, atau kebun sayur Anda sendiri, ubahlah tiga hal.

1. **Kadarnya.** Salin kadar dari setiap karung ke satu kolom `A`: persentase N, P_2O_5, dan K_2O, yang juga merupakan gram per 100 g. Bila Anda menambahkan hara keempat (MgO, belerang, kalsium), tambahkan satu baris, dan beralihlah ke `least_blend.py` kecuali Anda juga menambahkan pupuk keempat.
2. **Rekomendasinya.** Masukkan rekomendasi setiap blok, dalam gram per pohon (atau per meter persegi), ke `BLOCKS`. Dapatkan dari penyuluh pertanian atau laboratorium tanah; program tidak dapat melihat daun yang menguning.
3. **Periksa jawabannya sebelum menabur.** Jumlah negatif berarti rekomendasi itu tidak dapat dipenuhi dengan karung-karung ini; determinan yang dekat dengan nol, atau peringatan dari `lu_decompose`, berarti dua karung Anda hampir berupa campuran yang sama dan campurannya rapuh. Bulatkan ke takaran yang dapat Anda timbang (sendok takar 10 gram), lalu hitung ulang hara yang diberikan oleh jumlah yang sudah dibulatkan, seperti kolom “delivered” pada `least_blend.py`.

Beginilah langkah 3 untuk pohon muda. Campuran eksaknya 200 g, 360 g, dan 117.4 g. Dengan sendok takar 10 gram, pembantu Pak Harun akan menakar 200 g, 360 g, dan 120 g. Hara yang diberikan oleh jumlah yang sudah dibulatkan adalah N: $12 \times 2 + 20 \times 3.6 + 46 \times 1.2 = 24 + 72 + 55.2 = 151.2$ g, fosfat $12 \times 2 + 10 \times 3.6 = 60$ g, dan kalium $17 \times 2 + 10 \times 3.6 = 70$ g. Pembulatan urea menambahkan 1.2 g nitrogen, kurang dari 1%, yang tidak akan terasa oleh pohon mana pun. Untuk satu blok berisi 120 pohon muda, daftar belanjanya adalah 120×200 g $= 24$ kg NPK 12-12-17, 43.2 kg NPK 20-10-10, dan 14.4 kg urea: dalam karung 50 kg, masing-masing satu karung, dengan sebagian besar karung urea tersisa untuk musim berikutnya. Mengerjakan hitungan

dalam urutan ini, campuran eksak, lalu pembulatan, lalu pemeriksaan apa yang diberikan campuran yang dibulatkan, menangkap kasus-kasus ketika pembulatan berpengaruh, misalnya campuran yang hanya memuat beberapa gram salah satu pupuk.

Program yang sama menjawab pertanyaan di usaha apa pun yang mencampur: pabrik pakan yang memadukan jagung, dedak padi, dan bungkil kedelai untuk memenuhi target protein, energi, dan serat; toko roti kecil yang mencampur tepung untuk mencapai kadar protein tertentu; usaha laundry yang mengencerkan dua konsentrat deterjen sampai kekuatan tertentu. Setiap kali buku resepnya (matriks) tetap dan pesanannya (ruas kanan) berubah, faktorkan sekali.

6.11 Metode Numerik dalam Praktik: Profitina

AI Visibility Engine Profitina mengubah sinyal-sinyal yang diukurnya pada sebuah situs web, seperti seberapa mudah kontennya diekstrak, data terstruktur apa yang dimuat halaman-halamannya, dan seberapa kokoh situs itu secara teknis, menjadi skor kesiapan, dan menjelaskan setiap sinyal secara tersendiri. Mengubah banyak pengukuran menjadi satu angka adalah agregasi berbobot, dan bobotnya harus dipilih sedemikian rupa sehingga skornya sesuai dengan apa yang teramati: situs yang sering dikutip mesin jawaban seharusnya mendapat skor tinggi. Bila situs yang teramati lebih banyak daripada bobotnya, yang selalu terjadi, memilih bobot adalah masalah kuadrat terkecil yang terlalu ditentukan persis berbentuk Persamaan (6.8), satu baris per situs yang diamati dan satu kolom per sinyal. Pelajaran bab ini berlaku tanpa perubahan. Sinyal-sinyal yang mengukur hal yang hampir sama membuat kolom-kolomnya hampir bergantung linear, $\kappa(A)$ besar dan $\kappa(A^{\mathsf{T}}A)$ sangat besar; metode yang membentuk persamaan normal lalu dapat menghasilkan bobot yang tidak lain derau pembulatan, dan pemecah berbasis QR, yang dipakai pustaka secara bawaan, adalah jalan yang aman. Dan bila model sinyal yang sama dievaluasi untuk banyak merek pada setiap pemindaian ulang terjadwal, faktorisasi yang mahal diletakkan di luar gelung dan hanya substitusi yang murah di dalamnya, seperti pada **Gambar 6.1**.

Profitina menjelaskan apa yang diukur skornya dan mengapa, bukan rincian bobotnya yang merupakan hak milik, dan menyajikan skor itu sebagai saran: pemilik merek yang memutuskan apa yang akan diubah. Kecermatan numerik yang diuraikan di sini membuat saran seperti itu stabil dari minggu ke minggu, alih-alih melompat-lompat karena pembulatan.

6.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menghitung invers untuk menyelesaikan sistem.** `x = inv(A) @ b` tiga kali lebih mahal daripada `solve` dan kurang teliti. Faktorkan lalu substitusikan; bentuk A^{-1} hanya bila entri-entrinya sendiri yang diinginkan.
- **Melupakan permutasi.** Dengan *pivoting*, faktor-faktornya memenuhi $PA = LU$, bukan $A = LU$. Menyelesaikan $Ly = b$ alih-alih $Ly = Pb$ memberi jawaban salah tanpa peringatan. Pustaka yang berbeda menaruh P di sisi yang berbeda (SciPy mengembalikan $A = PLU$); bacalah dokumentasinya.
- **Memfaktorkan ulang untuk setiap ruas kanan.** Bila matriksnya tidak berubah, faktor-faktornya juga tidak. Simpanlah.
- **Memakai Cholesky pada matriks yang bukan SPD.** Matriks simetris bisa tak definit; Cholesky akan bertemu bilangan negatif di bawah tanda akar. Kegagalan itu adalah diagnosis, bukan cacat program; jangan "memperbaikinya" dengan mengambil nilai mutlak.

- **Menyelesaikan kuadrat terkecil dengan persamaan normal karena kebiasaan.** Persamaan normal mengkuadratkan bilangan kondisi (Persamaan (6.11)). Untuk masalah yang kondisinya baik, persamaan normal baik dan cepat; untuk pencocokan polinomial, kolom yang hampir bergantung linear, atau data dengan skala yang sangat berbeda, pakailah QR (`numpy.linalg.lstsq` atau `scipy.linalg.lstsq`).
- **Membaca jawaban kuadrat terkecil sebagai jawaban eksak.** Campuran di Subbab 6.7 meleset 25% pada magnesium. Selalu cetak apa yang sebenarnya diberikan jawaban di samping apa yang diminta, lalu putuskan apakah melesetnya dapat diterima.
- **Membaca determinan sebagai ukuran masalah.** Determinan yang sangat kecil tidak berarti matriksnya hampir singular: mengalikan setiap entri matriks identitas 10×10 dengan 0.1 memberi determinan 10^{-10} dan matriks yang kondisinya sempurna. Mengukur pupuk dalam kilogram alih-alih ratusan gram akan memperkecil $\det A$ dengan faktor 10^3 tanpa mengubah apa pun yang nyata. Pakailah bilangan kondisi, yang tidak bergantung pada satuan, untuk menilai apakah suatu sistem rapuh.
- **Mencampur satuan di baris-baris.** Bila satu baris dalam gram dan baris lain dalam kilogram, kuadrat terkecil memperlakukan meleset 1 satuan pada keduanya sama buruknya. Samakan satuan semua baris, atau beri bobot dengan sengaja (Soal 4).

6.13 Ringkasan Bab

- Eliminasi Gauss dengan *pivoting* parsial, dengan pengali-pengalinya dicatat, adalah dekomposisi $PA = LU$ (Persamaan (6.2), **Pseudokode 6.1**); entri-entrinya diperoleh langsung dari rumus Doolittle (Persamaan (6.4)).
- Memfaktorkan memerlukan $n^3/3$ perkalian; setiap ruas kanan lalu memerlukan dua substitusi segitiga (Persamaan (6.3)), sekitar n^2 (Persamaan (6.5), **Gambar 6.2**). Empat blok kebun (Persamaan (6.1)) berbagi satu faktorisasi (**Gambar 6.1**).
- Determinan adalah hasil kali bertanda dari diagonal U (Persamaan (6.6)); invers berarti n penyelesaian, dan jarang diperlukan.
- *Pivoting* bukan pilihan bebas: tanpanya, pivot yang sangat kecil menghasilkan pengali raksasa yang menghapus informasi di baris-baris lain (Subbab 6.3.5). Cholesky tidak memerlukan *pivoting* pada matriks SPD, karena entri-entrinya dibatasi oleh akar kuadrat diagonalnya (Subbab 6.4.4). Dengan demikian, pilihan faktorisasi yang tepat bergantung pada sifat matriksnya, bukan pada kebiasaan.
- Persyaratan yang lebih banyak daripada variabel (Persamaan (6.7)) membawa ke kuadrat terkecil (Persamaan (6.8)). x terbaik membuat residu tegak lurus terhadap kolom-kolom (**Gambar 6.3**), yang menghasilkan persamaan normal (Persamaan (6.9)).
- $A^{\mathsf{T}}A$ simetris definit positif dan dapat difaktorkan dengan Cholesky (Persamaan (6.10)) dengan separuh biaya LU; pivot yang tidak positif membuktikan bahwa matriksnya bukan SPD.
- Persamaan normal mengkuadratkan bilangan kondisi (Persamaan (6.11)). QR Householder (Persamaan (6.13)) mengubah kuadrat terkecil menjadi satu penyelesaian segitiga (Persamaan (6.12)) tanpa pengkuadratan; pada derajat 9 ia mempertahankan sepuluh digit ketika persamaan normal hanya tiga, dan pada derajat 12 ia bekerja ketika Cholesky gagal (**Gambar 6.4**). **Tabel 6.2** merangkum ketiga dekomposisi.
- Program Java dan Python cocok bita demi bita, dan SciPy serta NumPy menghasilkan kembali setiap angka.

6.14 Soal Latihan

1. Dengan faktor P, L, dan U dari Subbab 6.2.2, selesaikan dengan tangan untuk blok "menjelang berbunga", $b = (120,90,120)$. Periksa jawaban Anda dengan program.
2. Sebuah toko tani baru hanya menjual NPK 12-12-17 dalam dua ukuran karung dan urea, sehingga matriks kadar Pak Harun menjadi $\begin{pmatrix} 12 & 12 & 46 \\ 12 & 12 & 0 \\ 17 & 17 & 0 \end{pmatrix}$. Pada langkah mana **Pseudokode 6.1** berhenti, dan mengapa? Apa yang dikatakan determinannya tentang campuran-campurannya?
3. Hitung A^{-1} untuk matriks kadar pada Persamaan (6.1) dengan faktor LU (tiga penyelesaian). Tafsirkan kolom keduanya: bagaimana Anda memberi sebatang pohon satu gram fosfat tambahan dan tidak ada yang lain, dan mengapa hal itu mustahil dalam praktik?
4. Dalam masalah empat hara, meleset 1.5 g MgO adalah 25% dari target, sedangkan meleset 0.6 g fosfat hanya 1%. Bagilah setiap baris A dan b dengan target baris itu, sehingga melesetnya menjadi relatif, lalu selesaikan masalah berbobot itu. Bagaimana campuran dan meleset relatifnya berubah?
5. Tentukan faktor Cholesky dari $S = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 5 & 3 \\ 2 & 3 & 6 \end{pmatrix}$ dengan tangan. Lalu cobalah rumus yang sama pada $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$ dan jelaskan apa yang terjadi.
6. Susunlah refleksi Householder H yang memetakan $a = (3,4)$ ke kelipatan e_1, dengan aturan tanda dari Subbab 6.5.3. Tuliskan H sebagai matriks 2×2, periksa bahwa $H^{\mathsf{T}}H = I$, dan hitung Ha.
7. Sebuah laptop mengerjakan sekitar 10^9 perkalian per detik dalam gelung sederhana. Taksirlah waktu untuk menyelesaikan 100 sistem dengan matriks 1000×1000 yang sama melalui eliminasi setiap kali dan melalui LU sekali, dengan hitungan yang dicetak `lu_blend`.

6.15 Penyelesaian Soal Latihan

1. Pb menyusun ulang b sebagai K, N, P: $(120,120,90)$. Maju: $y_1 = 120$; $y_2 = 120 - 0.7059 \times 120 = 120 - 84.7059 = 35.2941$; $y_3 = 90 - 0.7059 \times 120 - 0.2273 \times 35.2941 = 90 - 84.7059 - 8.0214 = -2.7273$. Mundur: $x_3 = -2.7273/-10.4545 = 0.2609$; $x_2 = (35.2941 - 46 \times 0.2609)/12.9412 = (35.2941 - 12.0000)/12.9412 = 1.8$; $x_1 = (120 - 10 \times 1.8)/17 = 102/17 = 6$. Jadi 600 g NPK 12-12-17, 180 g NPK 20-10-10, dan 26.1 g urea, sesuai yang dicetak program.

2. Langkah 1 memilih pivot 17 (baris K) dan mengeliminasi: kedua baris sisanya mendapat $12 - (12/17) \times 17 = 0$ di kolom 2, karena kolom 1 dan 2 identik. Langkah 2 lalu hanya menemukan nol di kolom 2 pada dan di bawah diagonal, dan baris 4 berhenti dengan "singular" (program Python memunculkan `ValueError: matrix is singular`). Determinannya 0, karena dua kolomnya sama. Artinya rekomendasi tidak dapat menentukan campurannya: bila ada campuran yang cocok, memindahkan gram dari satu ukuran karung ke ukuran lain memberi campuran lain yang sama cocoknya, dan bila rekomendasi tidak berada di ruang kolom, tidak ada campuran yang cocok sama sekali. Dua karung dengan kadar yang sama adalah satu pupuk, bukan dua.

3. Menyelesaikan $Ax = e_1$, e_2, dan e_3 dengan faktor-faktornya memberi (dalam ratusan gram)

$$A^{-1} = \begin{pmatrix} 0 & -0.2 & 0.2 \\ 0 & 0.34 & -0.24 \\ 0.02174 & -0.09565 & 0.05217 \end{pmatrix}.$$

Kolom 1 menyatakan bahwa satu gram nitrogen tambahan saja memerlukan 2.17 g urea (0.02174 ratus gram), wajar, karena urea adalah nitrogen murni sebesar 46%. Kolom 2 menyatakan bahwa satu gram fosfat tambahan saja memerlukan -20 g NPK 12-12-17, $+34$ g NPK 20-10-10, dan -9.57 g urea. Jumlah negatif berarti mengambil pupuk dari campuran yang sudah ada: hanya mungkin bila campurannya memuat setidaknya sebanyak itu. Dengan ketiga karung ini fosfat tidak pernah datang sendirian, sehingga menaikkannya tanpa menyentuh N dan K memerlukan pertukaran antarkarung. Inilah gunanya A^{-1}: bukan untuk menyelesaikan sistem, melainkan untuk membaca kepekaan.

4. Membagi baris ke-i dengan b_i membuat setiap residu menjadi meleset relatif. Solusi berbobotnya 297.42 g NPK 12-12-17-2, 222.46 g NPK 20-10-10, dan 151.78 g urea. Hara yang diberikan adalah 150.00 g N, 57.94 g fosfat, 72.81 g kalium, dan 5.95 g MgO, meleset relatif 0%, -3.4%, $+4.0\%$, dan -0.9%, dibandingkan dengan 0%, -1.0%, $+0.9\%$, dan -25.3% tanpa bobot. Lebih banyak NPK 12-12-17-2 membawa magnesium hampir ke target, dan harganya dibayar oleh fosfat dan kalium. Pembobotan mana yang benar adalah wewenang penyuluh; program hanya mengerjakan apa yang diperintahkan bobotnya.

5. $l_{11} = \sqrt{4} = 2$, $l_{21} = 2/2 = 1$, $l_{31} = 2/2 = 1$; $l_{22} = \sqrt{5-1} = 2$, $l_{32} = (3 - 1 \times 1)/2 = 1$; $l_{33} = \sqrt{6-1-1} = 2$. Jadi $L = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 2 \end{pmatrix}$, dan mengalikan LL^{T} mengembalikan S. Untuk $\begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix}$: $l_{11} = 1$, $l_{21} = 2$, dan $l_{22} = \sqrt{1-4} = \sqrt{-3}$, bukan bilangan real. Matriks itu simetris tetapi tidak definit positif: $z = (1, -1)$ memberi $z^{\mathrm{T}}Sz = 1 - 2 - 2 + 1 = -2 < 0$. Kegagalan Cholesky itulah buktinya.

6. $\| a \| = 5$ dan $a_1 = 3 > 0$, sehingga $\alpha = -5$ dan $v = a - \alpha e_1 = (3 + 5, 4) = (8, 4)$, $v^{\mathrm{T}}v = 80$. Maka $H = I - \frac{2}{80}\begin{pmatrix} 64 & 32 \\ 32 & 16 \end{pmatrix} = \begin{pmatrix} 1 - 1.6 & -0.8 \\ -0.8 & 1 - 0.4 \end{pmatrix} = \begin{pmatrix} -0.6 & -0.8 \\ -0.8 & 0.6 \end{pmatrix}$. $H^{\mathrm{T}}H = \begin{pmatrix} 0.36 + 0.64 & 0.48 - 0.48 \\ 0.48 - 0.48 & 0.64 + 0.36 \end{pmatrix} = I$, dan $Ha = (-1.8 - 3.2, -2.4 + 2.4) = (-5, 0)$, sesuai maksud. Seandainya kita memilih $\alpha = +5$, v akan menjadi $(-2, 4)$, yang di sini tidak masalah, tetapi untuk $a = (1, 10^{-9})$ pilihan itu akan menghitung $v_1 = 1 - \| a \|$ sebagai selisih dua bilangan yang hampir sama dan kehilangan semua digitnya.

7. `lu_blend` mencetak 3.34×10^{10} perkalian untuk eliminasi 100 kali dan 4.33×10^{8} untuk memfaktorkan sekali. Dengan 10^9 per detik, itu sekitar 33 detik berbanding 0.43 detik. (LU dari pustaka berjalan jauh lebih cepat daripada gelung sederhana, tetapi perbandingan antara kedua pendekatan tetap sama.)

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins University Press, 2013.

[4] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[5] Commandant Benoît, “Note sur une méthode de résolution des équations normales provenant de l’application de la méthode des moindres carrés à un système d’équations linéaires en nombre inférieur à celui des inconnues (Procédé du Commandant Cholesky),” *Bulletin Géodésique*, vol. 2, pp. 67–77, 1924.

[6] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, 25th anniversary ed. Philadelphia, PA, USA: SIAM, 2022.

[7] A. S. Householder, “Unitary triangularization of a nonsymmetric matrix,” *Journal of the ACM*, vol. 5, no. 4, pp. 339–342, 1958.

[8] SciPy Developers, “SciPy 1.18.0 Release Notes,” Jun. 19, 2026, and “scipy 1.18.1” on the Python Package Index, Aug. 21, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/release/1.18.0-notes.html and https://pypi.org/project/scipy/

[9] NumPy Developers, “NumPy 2.5.0 Release Notes,” Jun. 21, 2026, and “numpy 2.5.3” on the Python Package Index, Sep. 6, 2026. [Online]. Available: https://numpy.org/doc/2.5/release/2.5.0-notes.html and https://pypi.org/project/numpy/

[10] SciPy Developers, “scipy.linalg.lu” and “scipy.linalg.lu_factor,” SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.linalg.lu.html

[11] C. L. Lawson and R. J. Hanson, *Solving Least Squares Problems*. Philadelphia, PA, USA: SIAM, 1995 (reprint of the 1974 edition).

Bab 7

Interpolasi: Lagrange, Newton, dan Spline

Tabel adalah cara dunia menyerahkan fungsi kepada kita. Pembuat tangki mencetak liter terhadap kedalaman tongkat ukur, dinas cuaca memberi suhu setiap tiga jam, pemberi pinjaman mendaftar angsuran untuk beberapa besar pinjaman, pembuat baterai mendaftar tegangan terhadap muatan. Nilai yang kita perlukan hampir tidak pernah tepat berada di satu baris tabel. *Interpolasi* adalah seni membaca di antara baris-baris: membangun fungsi yang melewati titik-titik tabel secara tepat dan memakainya di antara titik-titik itu. Bab ini membaca tabel tongkat ukur sebuah tangki silinder yang rebah, tempat jawaban eksaknya diketahui dan setiap galat dapat diukur. Di akhir bab ini Anda akan mampu melakukan interpolasi linear dan menjelaskan galatnya; membangun polinomial interpolasi yang tunggal dalam bentuk Lagrange dan dalam bentuk selisih terbagi Newton serta mengevaluasinya dengan murah; menyatakan dan memakai rumus galat interpolasi polinomial; mengenali fenomena Runge dan menghindarinya dengan titik Chebyshev; menurunkan spline kubik, menyusun sistem tridiagonalnya dengan ujung natural atau terjepit, dan menyelesaikannya dalam waktu linear; memprogram semuanya dalam Java dan Python; membandingkannya dengan NumPy dan SciPy; serta memilih, untuk suatu tabel, metode yang benar-benar terbaik, yang tidak selalu yang paling canggih.

7.1 Tongkat Ukur Pak Darto

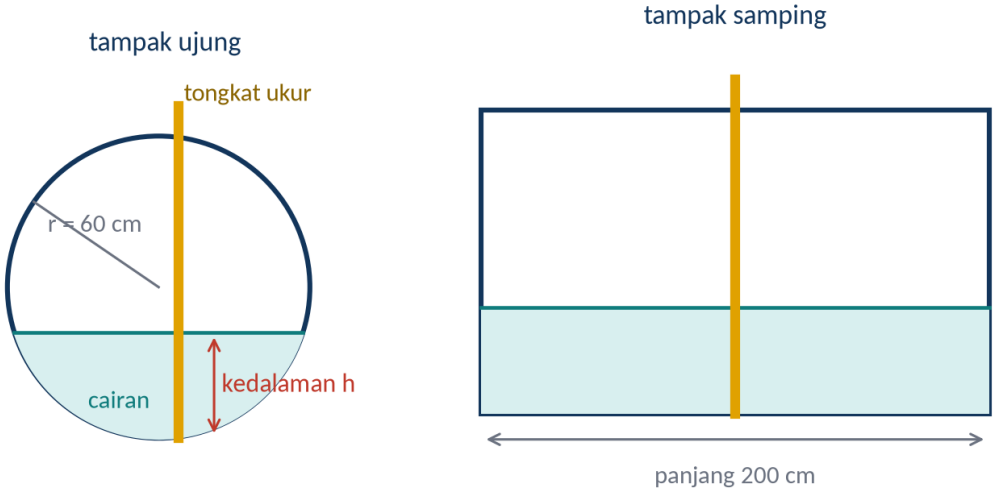
Pak Darto mengelola kios bahan bakar kecil di tepi desanya. Bahan bakarnya menunggu di dalam tangki baja yang rebah, panjang 200 cm dan bergaris tengah 120 cm, dan satu-satunya cara mengetahui isinya adalah tongkat ukur dari kayu: masukkan sampai dasar, tarik keluar, lalu baca sampai mana tanda basahnya. Setiap sore ia menulis kedalaman itu di buku catatan, mengubahnya menjadi liter, dan membandingkannya dengan angka penjualan pada pompa. Bila keduanya berselisih lebih dari sedikit, ada yang tidak beres: kebocoran, pompa yang rusak, atau pembantu yang kelelahan dan lupa mencatat penjualan.

Masalahnya ada pada pengubahan itu. Tangki yang rebah bukan ember: 10 cm pertama hanya memuat sedikit, karena dasarnya sempit, sedangkan 10 cm di tengah tangki memuat banyak. Pembuat tangki memberikan tabel cetak berisi volume pada setiap 10 cm, dan bagan yang dicat di dinding kios mengulang setiap baris kedua tabel itu, setiap 20 cm. Namun tongkat ukur menunjukkan 47 cm, atau 13 cm, atau 108 cm. Bertahun-tahun Pak Darto menarik garis lurus di kepalanya di antara dua tanda terdekat. Keponakannya, yang bersekolah di sekolah kejuruan di kota, curiga bahwa dari situlah sebagian liter yang “hilang” berasal, dan ia benar: di dekat dasar dan puncak tangki, bacaan garis lurus dari bagan dinding meleset lebih dari 30 liter.

Tangki ini guru yang sempurna, karena berbeda dengan kebanyakan tabel, fungsi eksaknya diketahui. Silinder berjari-jari r dan panjang L yang rebah, diisi sampai kedalaman h, memuat luas tembereng lingkaran dikalikan panjangnya:

$$V(h) = L\left[r^2 \arccos\left(\frac{r-h}{r}\right) - (r-h)\sqrt{2rh-h^2}\right]. \tag{7.1}$$

Dengan $r = 60$ cm, $L = 200$ cm, dan hasilnya dibagi 1000 untuk mengubah sentimeter kubik menjadi liter, rumus ini menghasilkan kembali tabel pembuat tangki: 247.80 L pada 20 cm, 1130.97 L pada 60 cm (setengah penuh), 2261.95 L penuh. Kita akan berpura-pura tidak mengetahui Persamaan (7.1) selama melakukan interpolasi, lalu memakainya untuk mengukur dengan tepat seberapa salah setiap metode. **Gambar 7.1** memperlihatkan tangkinya.



Gambar 7.1 Tangki rebah di kios bahan bakar: tongkat ukur mengukur kedalaman h, dan volumenya adalah luas tembereng lingkaran dikalikan panjang tangki.

7.2 Gagasannya dalam Kata-Kata Sederhana

7.2.1 Garis lurus di antara dua tanda

Bagan dinding memberi kedalaman $x_0, x_1, \ldots, x_n$ (0, 20, ..., 120 cm) dan volume $y_0, y_1, \ldots, y_n$. Untuk kedalaman x di antara dua tanda x_i dan x_{i+1}, garis lurus yang melalui kedua titik memberi

$$p(x) = y_i + \frac{x - x_i}{x_{i+1} - x_i}\,(y_{i+1} - y_i). \tag{7.2}$$

Pecahan itu menyatakan seberapa jauh x berada di sepanjang selang, dari 0 di x_i sampai 1 di x_{i+1}, dan volumenya bergerak dengan pecahan yang sama dari y_i ke y_{i+1}. Pada 50 cm, tepat di tengah antara 40 cm (660.02 L) dan 60 cm (1130.97 L), hasilnya $660.02 + 0.5 \times 470.95 = 895.50$ L. Tabel pembuat tangki menyebut 892.09 L pada 50 cm, jadi garis lurus kelebihan 3.4 L. Pada 10 cm hasilnya $0 + 0.5 \times 247.80 = 123.90$ L terhadap 90.03 L yang sebenarnya: kelebihan 33.9 L, lebih dari sepertiga.

Mengapa begitu berbeda? Karena kurva volume melengkung, dan paling melengkung di dekat dasar dan puncak. Di antara dua tanda, garis lurus memotong lengkungan itu. Di dekat dasar kurvanya cembung (melengkung ke atas, seperti senyum), talibusurnya berada di atas kurva, dan garis lurus membaca terlalu tinggi; di dekat puncak kurvanya cekung dan garis membaca terlalu rendah. Di tengah tangki kurvanya hampir lurus, begitu pula garisnya.

7.2.2 Kurva yang melalui lebih banyak tanda

Bila satu potong lurus tidak dapat mengikuti lengkungan, pakailah kurva yang melalui lebih banyak tanda. Parabola yang melalui tiga titik melengkung sekali. Melalui tanda 0, 20, dan 40 cm, parabola memberi 103.35 L pada 10 cm (kita menghitungnya dengan tangan di Subbab 7.3.2): masih kelebihan 13 L, tetapi jauh lebih baik daripada 123.90. Kubik yang melalui empat tanda 0, 20, 40, 60 cm memberi 96.74 L, dan polinomial derajat 6 yang melalui ketujuh tanda bagan dinding memberi 93.62 L, hanya meleset 3.6 L. Sejauh ini, lebih banyak titik berarti jawaban lebih baik.

Harapan itu tidak selalu terpenuhi. Subbab 7.3.6 memperlihatkan fungsi yang tampak tidak berbahaya, yang polinomialnya melalui makin banyak titik berjarak sama justru makin *buruk*, secara liar, di dekat ujung-ujung tabel. Dan jawaban modern yang paling populer, spline kubik, ternyata lebih buruk daripada polinomial derajat 6 untuk tangki yang satu ini, karena suatu alasan yang mengajarkan sesuatu tentang tangki itu. Sebelum menilai, kita memerlukan alat-alatnya.

7.3 Interpolasi Polinomial

7.3.1 Satu polinomial, dan hanya satu

> **DEFINISI.** Diberikan $n+1$ titik $(x_0, y_0), \dots, (x_n, y_n)$ dengan x_i yang berbeda-beda (*simpul*), polinomial p *menginterpolasi* titik-titik itu bila $p(x_i) = y_i$ untuk setiap i. **Teorema** [1], [2]: terdapat tepat satu polinomial interpolasi berderajat paling tinggi n.

Mengapa paling banyak satu. Bila p dan q sama-sama menginterpolasi, selisihnya $p - q$ berderajat paling tinggi n dan bernilai nol di $n+1$ simpul. Polinomial tak nol berderajat paling tinggi n memiliki paling banyak n akar, sehingga $p - q$ adalah polinomial nol.

Mengapa paling sedikit satu. Menulis $p(x) = c_0 + c_1 x + \cdots + c_n x^n$ dan menuntut $p(x_i) = y_i$ memberi $n+1$ persamaan linear untuk $n+1$ koefisien, dengan *matriks Vandermonde* (baris $1, x_i, x_i^2, \dots, x_i^n$). Karena ketunggalan yang baru ditunjukkan, sistem homogennya (semua $y_i = 0$) hanya memiliki solusi nol, sehingga matriksnya tak singular dan sistemnya memiliki solusi. Namun menyelesaikan sistem itu adalah cara yang buruk untuk mencari p: Bab 6 telah melihat betapa buruk kondisi matriks Vandermonde. Dua bentuk berikut menghasilkan polinomial yang sama tanpa menyelesaikan apa pun.

7.3.2 Bentuk Lagrange

Gagasan Joseph-Louis Lagrange (1795) adalah membangun, untuk setiap simpul x_i, polinomial ℓ_i yang bernilai 1 di x_i dan 0 di setiap simpul lain:

$$\ell_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \qquad p(x) = \sum_{i=0}^{n} y_i \, \ell_i(x). \tag{7.3}$$

Setiap ℓ_i berderajat n (hasil kali n faktor linear); pembilangnya lenyap di setiap simpul lain, dan penyebutnya membuat nilainya 1 di x_i. Jadi $p(x_k) = \sum_i y_i \, \ell_i(x_k) = y_k$, sesuai tuntutan. Polinomial ℓ_i disebut *polinomial basis Lagrange*.

Dengan tangan, dengan tanda 0, 20, dan 40 cm pada $x = 10$:

- $\ell_0(10) = \frac{(10-20)(10-40)}{(0-20)(0-40)} = \frac{(-10)(-30)}{800} = 0.375$

- $\ell_1(10) = \frac{(10-0)(10-40)}{(20-0)(20-40)} = \frac{(10)(-30)}{-400} = 0.75$
- $\ell_2(10) = \frac{(10-0)(10-20)}{(40-0)(40-20)} = \frac{(10)(-10)}{800} = -0.125$

dan $p(10) = 0.375 \times 0 + 0.75 \times 247.80 - 0.125 \times 660.02 = 185.85 - 82.50 = 103.35$ L. Perhatikan bahwa ketiga bobot berjumlah 1 (selalu demikian, karena menginterpolasi fungsi konstan 1 harus menghasilkan 1), dan salah satunya negatif: parabola "meminjam" dari tanda yang jauh untuk melengkung.

Bentuk Lagrange mudah ditulis dan dinalar, tetapi setiap evaluasi memerlukan $O(n^2)$ operasi, dan menambah simpul berarti menghitung ulang setiap polinomial basis. Bentuk Newton memperbaiki keduanya.

7.3.3 Bentuk Newton dan selisih terbagi

Isaac Newton menulis polinomial interpolasi sebagai jumlah suku-suku yang masing-masing lenyap di satu simpul lebih banyak:

$$\begin{aligned} p(x) \\ = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \cdots \\ + f[x_0, \ldots, x_n] \prod_{j=0}^{n-1} (x - x_j). \end{aligned} \tag{7.4}$$

Koefisien $f[\ldots]$ adalah **selisih terbagi** (*divided differences*), didefinisikan secara rekursif: $f[x_i] = y_i$, dan

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{7.5}$$

Selisih terbagi pertama $f[x_i, x_{i+1}]$ adalah kemiringan talibusur antara dua tetangga; yang kedua adalah selisih kemiringan dibagi lebar, ukuran kelengkungan (untuk fungsi mulus, $f[x_0, x_1, x_2] \approx f''/2$); secara umum $f[x_0, \ldots, x_k] = f^{(k)}(\xi)/k!$ untuk suatu ξ di antara simpul-simpul [2]. Mengapa Persamaan (7.4) menginterpolasi? $k+1$ suku pertama membentuk polinomial yang menginterpolasi $x_0, \ldots, x_k$ (dibuktikan dengan induksi pada k memakai Persamaan (7.5)), dan setiap suku sesudahnya memuat faktor $(x - x_0) \cdots (x - x_k)$ sehingga lenyap di simpul-simpul itu.

Selisih terbagi dihitung dalam tabel segitiga, kolom demi kolom. Untuk empat tanda pertama bagan dinding:

Tabel 7.1 Tabel selisih terbagi untuk empat tanda pertama; entri teratas setiap kolom adalah koefisien bentuk Newton.

x_i	$f[x_i]$	Pertama	Kedua	Ketiga
0	0.00	$247.80/20 = 12.39$	$(20.611 - 12.39)/40 = 0.205525$	$(0.0734125 - 0.205525)/60 = -0.00220188$
20	247.80	$412.22/20 = 20.611$	$(23.5475 - 20.611)/40 = 0.0734125$	
40	660.02	$470.95/20 = 23.5475$		
60	1130.97			

Koefisien Newton adalah baris teratasnya: 0, 12.39, 0.205525, −0.00220188. Kubik yang melalui keempat tanda ini adalah

$$p_3(x) = 0 + 12.39\,x + 0.205525\,x(x-20) - 0.00220188\,x(x-20)(x-40),$$

dan pada $x = 10$: $123.90 + 0.205525 \times 10 \times (-10) - 0.00220188 \times 10 \times (-10) \times (-30) = 123.90 - 20.55 - 6.61 = 96.74$ L. Bacalah ketiga suku itu sebagai cerita: talibusur menyebut 123.90, koreksi kelengkungan menarik 20.55 L, dan perubahan kelengkungan 6.61 L lagi. Menambah tanda kelima menambah satu suku dan membiarkan suku-suku ini, dan itulah keunggulan kedua bentuk Newton.

7.3.4 Evaluasi murah: perkalian bersarang

Ditulis seperti pada Persamaan (7.4), polinomial derajat n memerlukan $O(n^2)$ perkalian per evaluasi. Faktorkan seperti aturan Horner untuk polinomial biasa:

$$p(x) = c_0 + (x - x_0)(c_1 + (x - x_1)(c_2 + \cdots + (x - x_{n-1})\,c_n)), \tag{7.6}$$

dengan $c_k = f[x_0, \ldots, x_k]$. Evaluasi dari dalam ke luar: mulai dengan c_n, lalu n kali kalikan dengan $(x - x_k)$ dan tambahkan c_k. Itulah n perkalian per evaluasi, setelah tabel $O(n^2)$ yang dihitung sekali.

7.3.5 Seberapa salah polinomial interpolasi?

Mengetahui bahwa jawabannya tepat di simpul tidak mengatakan apa pun tentang yang di antaranya. Rumus galat dasarlah yang mengatakannya [1], [2]: bila f memiliki $n+1$ turunan kontinu dan p menginterpolasinya di $x_0, \ldots, x_n$, maka untuk setiap x terdapat ξ di selang terkecil yang memuat x dan simpul-simpul sehingga

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i). \tag{7.7}$$

Mengapa. Tetapkan x (bukan simpul) dan definisikan $g(t) = f(t) - p(t) - K\prod_i (t - x_i)$, dengan konstanta K dipilih sehingga $g(x) = 0$. Maka g lenyap di $n+2$ titik: $n+1$ simpul dan x. Menurut teorema Rolle, g' lenyap di $n+1$ titik di antaranya, g'' di n titik, dan seterusnya, sampai $g^{(n+1)}$ lenyap di suatu ξ. Tetapi $p^{(n+1)} = 0$ (berderajat n) dan turunan ke-$(n+1)$ hasil kali itu adalah $(n+1)!$, sehingga $0 = f^{(n+1)}(\xi) - K(n+1)!$, yang memberi K dan rumusnya.

Untuk interpolasi linear ($n = 1$) pada selang selebar h, hasil kali $(x - x_i)(x - x_{i+1})$ terbesar di tengah, tempat nilai mutlaknya $h^2/4$, sehingga galatnya paling besar $\frac{h^2}{8}\max|f''|$. Periksalah pada tangki di 50 cm. Menurunkan Persamaan (7.1) memberi $V'(h) = 2L\sqrt{2rh - h^2}/1000$ liter per sentimeter (lebar permukaan cairan dikalikan panjang, yaitu tambahan isi untuk satu sentimeter kedalaman) dan $V''(h) = 2L(r - h)/\left(1000\sqrt{2rh - h^2}\right)$. Di titik tengah selang, galatnya tepat $\frac{h^2}{8}V''(\xi)$ untuk suatu ξ di selang itu. Di antara 40 dan 60 cm, V'' sekitar 0.068 di dekat 50 cm, yang memberi $\frac{400}{8} \times 0.068 = 3.4$ L, dan galat yang terukur di 50 cm memang 3.4 L; batas kasus terburuknya, dengan V'' terbesar pada selang itu (0.141, di 40 cm), adalah 7.1 L. Di dekat dasar V''

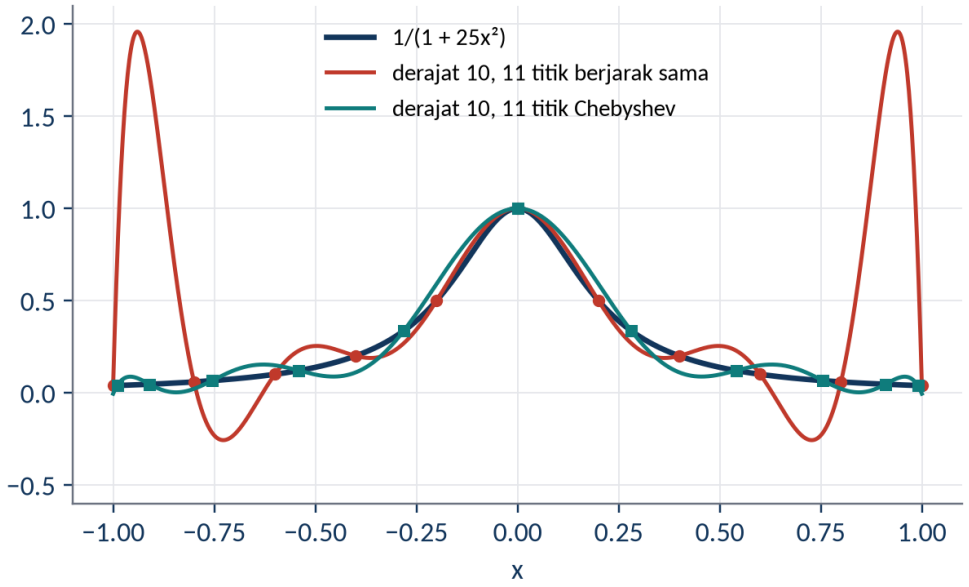
membesar tanpa batas ($\sqrt{2rh - h^2} \to 0$ ketika $h \to 0$), begitu pula batasnya, dan itulah sebabnya garis lurus gagal di sana.

7.3.6 Fenomena Runge dan titik Chebyshev

Persamaan (7.7) memiliki dua bagian: faktor turunan, yang bergantung pada f, dan polinomial simpul $\prod(x - x_i)$, yang hanya bergantung pada letak simpul. Untuk simpul berjarak sama pada $[-1,1]$, polinomial simpul kecil di tengah dan sangat besar di dekat ujung, dan untuk sebagian fungsi, turunan-turunan tingginya tumbuh lebih cepat daripada $(n+1)!$. Carl Runge menemukan contoh klasiknya pada 1901 [3]:

$$f(x) = \frac{1}{1 + 25x^2}, \qquad -1 \le x \le 1. \tag{7.8}$$

Ini lonceng yang mulus, tidak ada yang aneh. Namun polinomial interpolasinya melalui $n+1$ titik berjarak sama *divergen* di dekat ± 1 ketika n membesar: program di Subbab 7.6 menemukan galat terbesar 0.43 untuk $n = 5$, 1.92 untuk $n = 10$, dan 59.8 untuk $n = 20$. **Gambar 7.2** memperlihatkan polinomial derajat 10 berayun sampai hampir 2 di dekat ujung, tempat fungsinya di bawah 0.04. Penyebabnya berada di luar garis real: f memiliki kutub di $x = \pm i/5$, dekat dengan selang, dan hal ini membatasi seberapa baik polinomial dapat mengikutinya dengan simpul yang letaknya buruk [4].



Gambar 7.2 Fungsi Runge dan interpolan derajat 10-nya: melalui 11 titik berjarak sama polinomialnya berayun liar di dekat ujung; melalui 11 titik Chebyshev ia mengikuti fungsinya dengan rapat.

Obatnya adalah memindahkan simpul. **Titik Chebyshev**

$$x_i = \cos\frac{(2i+1)\pi}{2n+2}, \qquad i = 0,1,\ldots,n, \tag{7.9}$$

adalah proyeksi ke sumbu dari titik-titik berjarak sama pada setengah lingkaran, sehingga titik-titik itu merapat ke arah ujung. Titik-titik inilah yang membuat nilai terbesar $|\prod(x - x_i)|$ pada $[-1,1]$ sekecil mungkin, yaitu 2^{-n} [2], [4]. Seberapa besar bedanya? Untuk $n = 10$ pada $[-1,1]$, nilai terbesar $|\prod(x - x_i)|$ sekitar 8.5×10^{-3} untuk simpul berjarak sama, dicapai di selang-selang terluar, dan tepat $2^{-10} \approx 9.8 \times 10^{-4}$ untuk simpul Chebyshev, tersebar merata di seluruh selang:

hampir sembilan kali lebih kecil, dan perbandingannya tumbuh cepat bersama n. Dengan titik Chebyshev galat Runge turun menjadi 0.109 untuk $n = 10$ dan 0.015 untuk $n = 20$, dan terus turun ketika n membesar. Untuk sembarang selang $[a, b]$, petakan titik-titiknya dengan $x \mapsto \frac{a+b}{2} + \frac{b-a}{2}x$. Pelajarannya umum: bila Anda dapat memilih tempat mengukur, jangan memilih jarak yang sama untuk polinomial berderajat tinggi; rapatkan pengukuran ke arah ujung.

> **WAWASAN KUNCI:** Lebih banyak titik tidak otomatis berarti polinomial yang lebih baik. Galat Persamaan (7.7) adalah hasil kali dari apa yang dilakukan fungsinya (turunan-turunan tingginya) dan letak titik-titiknya (polinomial simpul). Jarak yang sama membuat polinomial simpul meledak di dekat ujung; jarak Chebyshev menjaganya tetap kecil secara merata.

7.3.7 Bila kemiringannya juga diketahui: interpolasi Hermite

Kadang-kadang tabel memberi lebih dari sekadar nilai. Untuk tangki kita juga mengetahui kemiringan $V'(h)$, yaitu lebar permukaan cairan dikalikan panjang, dan di dasar kemiringan itu nol. Polinomial yang cocok dengan nilai sekaligus kemiringan di simpul-simpul disebut *interpolan Hermite*, menurut Charles Hermite. Pada satu selang $[x_0, x_1]$ selebar h dengan nilai y_0, y_1 dan kemiringan s_0, s_1, interpolan itu adalah kubik tunggal dengan keempat sifat tersebut, dan di titik tengahnya bentuknya sederhana

$$p\left(\frac{x_0 + x_1}{2}\right) = \frac{y_0 + y_1}{2} + \frac{h}{8}(s_0 - s_1). \tag{7.10}$$

Suku pertama adalah garis lurus; suku kedua melengkungkannya dengan memakai selisih kemiringan di ujung-ujung. Untuk selang pertama tangki, $y_0 = 0$, $y_1 = 247.80$, $s_0 = 0$, dan $s_1 = V'(20) = 0.4\sqrt{2400 - 400} = 17.89$ L/cm, sehingga $p(10) = 123.90 + \frac{20}{8}(0 - 17.89) = 123.90 - 44.72 = 79.18$ L. Volume sebenarnya 90.03 L: garis lurus kelebihan 33.9 L, kubik Hermite kekurangan 10.9 L. Mengetahui kemiringan telah memangkas galat dua pertiga pada selang terburuk, hanya dengan memakai dua tanda tetangga. Kubik Hermite sepotong-sepotong, dengan kemiringan yang diketahui atau ditaksir dari data, adalah dasar interpolan PCHIP yang mempertahankan bentuk di Subbab 7.8.

7.4 Spline Kubik

7.4.1 Sepotong-sepotong, tetapi mulus

Bila titik-titiknya sudah tertentu, seperti pada tabel, dan Anda tidak dapat memindahkannya, ada jalan keluar lain dari derajat tinggi: pakailah *polinomial berderajat rendah yang berbeda pada setiap selang*, lalu rekatkan potongan-potongannya dengan mulus. Potongan lurus menghasilkan Persamaan (7.2), dengan sudut tajam di simpul. Potongan kubik cukup leluasa untuk menyembunyikan sambungannya: **spline kubik** adalah fungsi S yang berupa polinomial kubik pada setiap selang $[x_i, x_{i+1}]$, menginterpolasi data, dan turunan pertama serta keduanya kontinu di mana-mana. Namanya berasal dari bilah kayu tipis yang lentur (*spline*) yang dilengkungkan juru gambar dan pembuat kapal melalui pasak-pasak tetap untuk menggambar kurva mulus; bilah yang dilengkungkan mengambil bentuk yang meminimumkan energi lenturnya, dan spline kubik natural meminimumkan $\int S''(x)^2\,dx$ di antara semua interpolan mulus, versi matematis dari sifat yang sama [5].

7.4.2 Menurunkan sistem tridiagonal

Hitunglah syarat-syaratnya. Dengan n selang terdapat $4n$ koefisien kubik. Interpolasi di kedua ujung setiap potongan memberi $2n$ syarat; kekontinuan S' dan S'' di $n-1$ simpul dalam memberi $2(n-1)$ lagi. Jumlahnya $4n-2$: dua syarat kurang, dan keduanya dipasok di kedua ujung tabel.

Cara yang rapi untuk menyusunnya memakai turunan kedua yang belum diketahui $M_i = S''(x_i)$ di simpul-simpul. Pada $[x_i, x_{i+1}]$, selebar $h_i = x_{i+1} - x_i$, S berupa kubik, jadi S'' linear, bergerak dari M_i ke M_{i+1}. Mengintegralkan dua kali dan memilih kedua konstanta integrasinya agar $S(x_i) = y_i$ dan $S(x_{i+1}) = y_{i+1}$ memberi, dengan $a = x_{i+1} - x$ dan $b = x - x_i$,

$$S(x) = \frac{M_i a^3 + M_{i+1} b^3}{6h_i} + \left(\frac{y_i}{h_i} - \frac{M_i h_i}{6}\right) a + \left(\frac{y_{i+1}}{h_i} - \frac{M_{i+1} h_i}{6}\right) b. \tag{7.11}$$

Potongan ini menginterpolasi dan memiliki turunan kedua M_i dan M_{i+1} di ujung-ujungnya, sehingga S dan S'' kontinu dengan sendirinya. Yang tersisa adalah kekontinuan S' di setiap simpul dalam x_i. Tuliskan $\Delta_i = (y_{i+1} - y_i)/h_i$ untuk kemiringan talibusur selang i. Menurunkan Persamaan (7.11) (ingat bahwa a mengecil ketika x membesar) dan memasang $x = x_i$, tempat $a = h_i$ dan $b = 0$, memberi kemiringan di ujung kiri selang i: $S'(x_i^+) = \Delta_i - h_i(2M_i + M_{i+1})/6$. Melakukan hal yang sama pada selang $i-1$ di ujung kanannya, tempat $a = 0$ dan $b = h_{i-1}$, memberi $S'(x_i^-) = \Delta_{i-1} + h_{i-1}(M_{i-1} + 2M_i)/6$. Kedua kemiringan harus sama. Mengalikan persamaannya dengan 6 dan mengumpulkan kelengkungan di ruas kiri:

$$h_{i-1} M_{i-1} + 2(h_{i-1} + h_i)\, M_i + h_i M_{i+1} = 6\,(\Delta_i - \Delta_{i-1}), \qquad i = 1, \ldots, n-1. \tag{7.12}$$

Setiap persamaan hanya melibatkan tiga variabel yang bertetangga: sistemnya **tridiagonal**. Diagonalnya $2(h_{i-1} + h_i)$ lebih besar daripada jumlah entri di luar diagonal $h_{i-1} + h_i$, sehingga sistem itu dominan diagonal sejati (Bab 5): solusinya tunggal, dan eliminasinya tidak memerlukan *pivoting*.

7.4.3 Dua syarat ujung

- **Spline natural:** $M_0 = M_n = 0$, tidak melengkung di ujung, seperti bilah juru gambar yang bebas. Ini bawaan di banyak buku teks, tetapi merupakan asumsi tentang data, dan sering keliru.
- **Spline terjepit** (*clamped*): kemiringan ujung $S'(x_0) = s_0$ dan $S'(x_n) = s_n$ diberikan, yang menambah persamaan $2h_0 M_0 + h_0 M_1 = 6(\Delta_0 - s_0)$ dan $h_{n-1} M_{n-1} + 2h_{n-1} M_n = 6(s_n - \Delta_{n-1})$. Pakailah bila kemiringannya diketahui. Untuk tangki, kemiringan itu diketahui: tongkat ukur di dasar atau di puncak tangki menyentuh garis singgung mendatar lingkaran, dan $V'(0) = V'(120) = 0$.
- **Not-a-knot:** turunan ketiga kontinu di x_1 dan x_{n-1}, sehingga dua potongan pertama dan dua potongan terakhir masing-masing merupakan satu kubik. Inilah bawaan SciPy [6].

7.4.4 Algoritma Thomas

Sistem tridiagonal diselesaikan dengan eliminasi Gauss tanpa *pivoting* yang hanya menyentuh ketiga diagonal. Dengan diagonal bawah l_i, diagonal d_i, diagonal atas u_i, dan ruas kanan r_i:

$$d_i \leftarrow d_i - \frac{l_i}{d_{i-1}}\,u_{i-1}, \quad r_i \leftarrow r_i - \frac{l_i}{d_{i-1}}\,r_{i-1} \quad (i = 1, \ldots, n); \qquad M_n = \frac{r_n}{d_n}, \quad M_i = \frac{r_i - u_i M_{i+1}}{d_i}. \tag{7.13}$$

Inilah dekomposisi LU dari Bab 6 untuk matriks berlebar pita 1, dan biayanya sekitar $8n$ operasi alih-alih $n^3/3$: spline melalui sejuta titik hanya memerlukan beberapa milidetik. Algoritma ini menyandang nama fisikawan Llewellyn Thomas, yang menguraikannya pada 1949 [7], dan aman tanpa *pivoting* setiap kali matriksnya dominan diagonal, seperti matriks kita [1].

7.4.5 Seberapa teliti spline?

Untuk fungsi dengan empat turunan kontinu, spline kubik terjepit dengan selang terbesar h memenuhi $\max|f - S| \le \frac{5}{384}h^4 \max\left|f^{(4)}\right|$ [2], [5]. Membagi dua jarak simpul membagi galatnya dengan 16, dan konstanta $5/384$ kecil. Itulah sebabnya spline menjadi pilihan bawaan untuk data yang mulus. Namun hipotesisnya penting: batas itu mengandaikan $f^{(4)}$ terbatas. Untuk tangki tidak demikian: V'' sudah tak hingga di kedua ujung. Subbab 7.6 memperlihatkan akibatnya.

7.4.6 Satu bacaan spline dengan tangan

Setelah kelengkungannya diketahui, membaca spline tinggal aritmetika. Program di Subbab 7.6 menemukan, untuk spline natural melalui ketujuh tanda bagan dinding, $M_0 = 0$ dan $M_1 = 0.5990$. Pada $x = 10$ cm kita berada di selang pertama, $x_0 = 0$, $x_1 = 20$, $h_0 = 20$, serta $a = x_1 - x = 10$ dan $b = x - x_0 = 10$. Persamaan (7.11) memberi

$$S(10) = \frac{0 \cdot 10^3 + 0.5990 \cdot 10^3}{6 \cdot 20} + \left(\frac{0}{20} - 0\right) \cdot 10 + \left(\frac{247.80}{20} - \frac{0.5990 \cdot 20}{6}\right) \cdot 10 = 4.99 + 0 + (12.39 - 2.00) \times 10 = 108.93 \text{ L}.$$

Kurung terakhir adalah kemiringan talibusur 12.39 yang dikoreksi oleh kelengkungan. Dibandingkan dengan garis lurus (123.90 L), spline telah melengkungkan bacaannya turun 15 L menuju 90.03 L yang sebenarnya, tetapi syarat ujung natural $M_0 = 0$ menahannya agar tidak cukup melengkung: di dekat dasar, kelengkungan sebenarnya bukan nol melainkan tak hingga.

7.4.7 Membaca tabel secara terbalik

Kadang-kadang pertanyaannya berbalik arah. Truk tangki mengantar 1000 L; sampai kedalaman berapa bahan bakar akan naik? Inilah *interpolasi invers*: tukarkan peran kedua kolom dan interpolasikan kedalaman sebagai fungsi volume. Cara ini berhasil setiap kali tabelnya monoton, seperti tabel volume pasti monoton, karena setiap volume hanya dimiliki satu kedalaman. Andaikan tongkat ukur menunjukkan 40 cm (660.02 L) sebelum pengantaran 1000 L. Sesudahnya tangki berisi 1660.02 L, yang terletak di antara tanda 80 cm (1601.93 L) dan 100 cm (2014.15 L), dan garis lurus yang dibaca terbalik memberi $80 + 20 \times (1660.02 - 1601.93)/(2014.15 - 1601.93) = 80 + 20 \times 0.1409 = 82.8$ cm. Kedalaman eksak menurut Persamaan (7.1) adalah 82.6 cm, jadi bacaan tongkat ukur yang jauh dari 82.6 cm setelah pengantaran berarti kirimannya kurang. Cara lain adalah menyelesaikan $p(x) = 1660.02$ untuk x dengan pencari akar dari Bab 2 dan 3, memakai interpolan p dari tabel maju. Kedua jawaban sedikit berbeda, karena

menginterpolasi fungsi invers tidak sama dengan membalik interpolan, tetapi keduanya baik di dekat tengah tangki, tempat kurvanya hampir lurus.

7.5 Membangunnya dalam Java dan Python

Dua program: `interp` membaca tabel pembuat tangki dari `data/dipstick.csv`, mengambil setiap baris kedua sebagai bagan dinding (7 tanda, berjarak 20 cm), lalu menguji interpolasi linear, Lagrange, dan Newton pada baris-baris 10 cm yang tidak dipakainya, kemudian terhadap Persamaan (7.1) pada kisi 1 mm, dan akhirnya menjalankan percobaan Runge. `spline` membangun spline natural dan terjepit dari tanda-tanda yang sama dan dari ketiga belas baris tabel. Rumus tangki Persamaan (7.1) hanya dipakai untuk mengukur galat, tidak pernah untuk interpolasi. Menyimpan tabel sebagai berkas, dibulatkan ke dua desimal seperti tabel cetak, membuat kedua bahasa bekerja dari bilangan yang persis sama.

Kode 7.1 adalah versi Python program pertama. `lagrange` adalah Persamaan (7.3) yang ditulis sebagai dua gelung; `divided_differences` membangun tabel Persamaan (7.5) kolom demi kolom; `newton_eval` adalah bentuk bersarang Persamaan (7.6).

Kode 7.1 `interp.py`: interpolasi linear, Lagrange, dan Newton atas tabel tongkat ukur, galatnya terhadap rumus eksak tangki, dan percobaan Runge dengan titik berjarak sama dan titik Chebyshev.

```
"""Membaca di antara tanda tabel tongkat ukur: interpolasi linear, Lagrange
dan Newton untuk tangki rebah, serta peringatan Runge."""
import math

R, LENGTH = 60.0, 200.0          # jari-jari dan panjang tangki dalam cm


def tank_liters(h):
    """Volume eksak cairan (liter) pada kedalaman h cm di silinder rebah."""
    c = R - h
    area = R * R * math.acos(c / R) - c * math.sqrt(2.0 * R * h - h * h)
    return area * LENGTH / 1000.0


def lagrange(xs, ys, x):
    """Jumlah y_i kali polinomial basis yang bernilai 1 di x_i, 0 di lainnya."""
    total = 0.0
    for i in range(len(xs)):
        basis = 1.0
        for j in range(len(xs)):
            if j != i:
                basis *= (x - xs[j]) / (xs[i] - xs[j])
        total += ys[i] * basis
    return total


def divided_differences(xs, ys):
    """Tabel lengkap: table[k][i] = f[x_i, ..., x_(i+k)]."""
    table = [list(ys)]
    for k in range(1, len(xs)):
        prev = table[-1]
        table.append([(prev[i + 1] - prev[i]) / (xs[i + k] - xs[i])
                      for i in range(len(xs) - k)])
    return table


def newton_eval(xs, coef, x):
    """Evaluasi bersarang (ala Horner) bentuk Newton."""
    result = coef[-1]
    for k in range(len(coef) - 2, -1, -1):
        result = result * (x - xs[k]) + coef[k]
    return result
```

Kode 7.1 (Lanjutan) *interp.py: interpolasi linear, Lagrange, dan Newton atas tabel tongkat ukur, galatnya terhadap rumus eksak tangki, dan percobaan Runge dengan titik berjarak sama dan titik Chebyshev.*

```
def linear(xs, ys, x):
    for i in range(len(xs) - 1):
        if xs[i] <= x <= xs[i + 1]:
            t = (x - xs[i]) / (xs[i + 1] - xs[i])
            return ys[i] + t * (ys[i + 1] - ys[i])
    raise ValueError("x outside the table")


def runge(x):
    return 1.0 / (1.0 + 25.0 * x * x)


def read_table(path):
    """Baris (kedalaman cm, liter) dari tabel CSV pembuat tangki."""
    rows = []
    with open(path, encoding="utf-8") as f:
        next(f)                                     # lewati baris judul
        for line in f:
            h, v = line.strip().split(",")
            rows.append((float(h), float(v)))
    return rows


def main():
    table10 = read_table("../data/dipstick.csv")
    xs = [h for h, _ in table10 if h % 20 == 0]    # tabel dinding: 20 cm
    ys = [v for h, v in table10 if h % 20 == 0]
    dd = divided_differences(xs, ys)
    print("Newton coefficients f[x0..xk] from the 20 cm marks")
    for k, col in enumerate(dd):
        print(f"  k = {k}: {col[0]: .6e}")
    coef = [col[0] for col in dd]
    print()
    print("depth   table   linear   error  Lagrange   Newton   error")
    for h, v in table10:
        if h % 20 == 0:
            continue                                # sebuah simpul: tak diuji
        li = linear(xs, ys, h)
        la = lagrange(xs, ys, h)
        ne = newton_eval(xs, coef, h)
        print(f"{h:5.0f}{v:9.2f}{li:9.2f}{li - v:8.2f}"
              f"{la:10.2f}{ne:9.2f}{ne - v:8.2f}")
    worst_lin = worst_pol = 0.0
    for k in range(1201):
        h = 0.1 * k
        ex = tank_liters(h)
```

Kode 7.1 (Lanjutan) *interp.py: interpolasi linear, Lagrange, dan Newton atas tabel tongkat ukur, galatnya terhadap rumus eksak tangki, dan percobaan Runge dengan titik berjarak sama dan titik Chebyshev.*

```
        worst_lin = max(worst_lin, abs(linear(xs, ys, h) - ex))
        worst_pol = max(worst_pol, abs(newton_eval(xs, coef, h) - ex))
    print(f"largest error on 0..120 cm: linear {worst_lin:.2f} L,"
          f" degree 6 {worst_pol:.2f} L")
    print()
    print("Runge's function 1/(1+25x^2) on [-1, 1]: largest error")
    print("   n   equally spaced   Chebyshev points")
    for n in (5, 10, 15, 20):
        eq = [-1.0 + 2.0 * i / n for i in range(n + 1)]
        ch = [math.cos((2 * i + 1) * math.pi / (2 * n + 2))
              for i in range(n + 1)]
        errs = []
        for nodes in (eq, ch):
            vals = [runge(t) for t in nodes]
            worst = 0.0
            for k in range(2001):
                t = -1.0 + k / 1000.0
                worst = max(worst, abs(lagrange(nodes, vals, t) - runge(t)))
            errs.append(worst)
        print(f"{n:4d}{errs[0]:17.4f}{errs[1]:19.4f}")


if __name__ == "__main__":
    main()
```

Kode 7.2 adalah versi Java-nya. `readTable` dipakai bersama oleh program spline, sehingga keduanya dikompilasi bersama.

Kode 7.2 `Interp.java`: program interpolasi yang sama dalam Java.

```
import java.io.IOException;
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;

/** Membaca di antara tanda tabel tongkat ukur: interpolasi linear, Lagrange
 *  dan Newton untuk tangki rebah, serta peringatan Runge. */
public class Interp {
    static final double R = 60.0, LENGTH = 200.0;   // jari-jari, panjang tangki, cm

    /** Volume eksak cairan (liter) pada kedalaman h cm di silinder rebah. */
    static double tankLiters(double h) {
        double c = R - h;
        double area = R * R * Math.acos(c / R) - c * Math.sqrt(2.0 * R * h - h * h);
        return area * LENGTH / 1000.0;
    }

    /** Jumlah y_i kali polinomial basis yang bernilai 1 di x_i, 0 di lainnya. */
    static double lagrange(double[] xs, double[] ys, double x) {
        double total = 0.0;
        for (int i = 0; i < xs.length; i++) {
            double basis = 1.0;
            for (int j = 0; j < xs.length; j++) {
                if (j != i) basis *= (x - xs[j]) / (xs[i] - xs[j]);
            }
            total += ys[i] * basis;
        }
        return total;
    }

    /** Tabel lengkap: table[k][i] = f[x_i, ..., x_(i+k)]. */
    static double[][] dividedDifferences(double[] xs, double[] ys) {
        int n = xs.length;
        double[][] table = new double[n][];
        table[0] = ys.clone();
        for (int k = 1; k < n; k++) {
            table[k] = new double[n - k];
            for (int i = 0; i < n - k; i++) {
                table[k][i] = (table[k - 1][i + 1] - table[k - 1][i])
                              / (xs[i + k] - xs[i]);
            }
        }
```

Kode 7.2 (Lanjutan) *Interp.java: program interpolasi yang sama dalam Java.*

```
        return table;
    }

    /** Evaluasi bersarang (ala Horner) bentuk Newton. */
    static double newtonEval(double[] xs, double[] coef, double x) {
        double result = coef[coef.length - 1];
        for (int k = coef.length - 2; k >= 0; k--) {
            result = result * (x - xs[k]) + coef[k];
        }
        return result;
    }

    static double linear(double[] xs, double[] ys, double x) {
        for (int i = 0; i < xs.length - 1; i++) {
            if (xs[i] <= x && x <= xs[i + 1]) {
                double t = (x - xs[i]) / (xs[i + 1] - xs[i]);
                return ys[i] + t * (ys[i + 1] - ys[i]);
            }
        }
        throw new IllegalArgumentException("x outside the table");
    }

    static double runge(double x) { return 1.0 / (1.0 + 25.0 * x * x); }

    /** Baris (kedalaman cm, liter) dari tabel CSV pembuat tangki. */
    static double[][] readTable(String path) throws IOException {
        List<String> lines = Files.readAllLines(Path.of(path));
        List<double[]> rows = new ArrayList<>();
        for (String line : lines.subList(1, lines.size())) {  // lewati judul
            String[] p = line.strip().split(",");
            rows.add(new double[] {Double.parseDouble(p[0]),
                                   Double.parseDouble(p[1])});
        }
        return rows.toArray(new double[0][]);
    }
```

Kode 7.2 (Lanjutan) *Interp.java: program interpolasi yang sama dalam Java.*

```
    public static void main(String[] args) throws IOException {
        double[][] table10 = readTable("../data/dipstick.csv");
        List<double[]> marks = new ArrayList<>();
        for (double[] row : table10) {
            if (row[0] % 20 == 0) marks.add(row);    // tabel dinding: 20 cm
        }
        double[] xs = new double[marks.size()], ys = new double[marks.size()];
        for (int i = 0; i < xs.length; i++) {
            xs[i] = marks.get(i)[0];
            ys[i] = marks.get(i)[1];
        }
        double[][] dd = dividedDifferences(xs, ys);
        System.out.println("Newton coefficients f[x0..xk] from the 20 cm marks");
        double[] coef = new double[dd.length];
        for (int k = 0; k < dd.length; k++) {
            coef[k] = dd[k][0];
            String s = e(coef[k], 6);
            System.out.println("  k = " + k + ": "
                               + (s.startsWith("-") ? s : " " + s));
        }
        System.out.println();
        System.out.println("depth   table   linear   error  Lagrange"
                           + "   Newton   error");
        for (double[] row : table10) {
            double h = row[0], v = row[1];
            if (h % 20 == 0) continue;               // sebuah simpul: tak diuji
            double li = linear(xs, ys, h);
            double la = lagrange(xs, ys, h);
            double ne = newtonEval(xs, coef, h);
            System.out.println(f(h, 0, 5) + f(v, 2, 9) + f(li, 2, 9)
                    + f(li - v, 2, 8) + f(la, 2, 10) + f(ne, 2, 9)
                    + f(ne - v, 2, 8));
        }
        double worstLin = 0, worstPol = 0;
        for (int k = 0; k <= 1200; k++) {
            double h = 0.1 * k;
            double ex = tankLiters(h);
            worstLin = Math.max(worstLin, Math.abs(linear(xs, ys, h) - ex));
            worstPol = Math.max(worstPol, Math.abs(newtonEval(xs, coef, h) - ex));
        }
        System.out.println("largest error on 0..120 cm: linear " + f(worstLin, 2)
                           + " L, degree 6 " + f(worstPol, 2) + " L");
        System.out.println();
        System.out.println("Runge's function 1/(1+25x^2) on [-1, 1]:"
                           + " largest error");
        System.out.println("   n   equally spaced   Chebyshev points");
```

Kode 7.2 (Lanjutan) *Interp.java: program interpolasi yang sama dalam Java.*

```
        for (int n : new int[] {5, 10, 15, 20}) {
            double[] eq = new double[n + 1], ch = new double[n + 1];
            for (int i = 0; i <= n; i++) {
                eq[i] = -1.0 + 2.0 * i / n;
                ch[i] = Math.cos((2 * i + 1) * Math.PI / (2 * n + 2));
            }
            StringBuilder line = new StringBuilder(pad("" + n, 4));
            int width = 17;
            for (double[] nodes : new double[][] {eq, ch}) {
                double[] vals = new double[nodes.length];
                for (int i = 0; i < nodes.length; i++) vals[i] = runge(nodes[i]);
                double worst = 0;
                for (int k = 0; k <= 2000; k++) {
                    double t = -1.0 + k / 1000.0;
                    worst = Math.max(worst,
                            Math.abs(lagrange(nodes, vals, t) - runge(t)));
                }
                line.append(f(worst, 4, width));
                width = 19;
            }
            System.out.println(line);
        }
    }

    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }

    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                    + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }
```

Kode 7.2 (Lanjutan) *Interp.java: program interpolasi yang sama dalam Java.*

```
    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }

    static String padRight(String s, int w) {
        return s + " ".repeat(Math.max(0, w - s.length()));
    }
}
```

Kode 7.3 membangun spline. `spline_curvatures` mengisi ketiga diagonal Persamaan (7.12), menambahkan baris ujung natural atau terjepit, lalu menjalankan algoritma Thomas Persamaan (7.13); `spline_eval` mencari selangnya dan menerapkan Persamaan (7.11). Spline

natural ditulis sebagai dua baris sepele "$1 \cdot M_0 = 0$" dan "$1 \cdot M_n = 0$", sehingga kedua jenis ujung memakai pemecah yang sama.

Kode 7.3 `spline.py`: spline kubik natural dan terjepit melalui sistem tridiagonal kelengkungan dan algoritma Thomas.

```
"""Spline kubik natural dan terjepit melalui tabel tongkat ukur: satu sistem
tridiagonal kecil untuk kelengkungan, lalu kubik di antara dua tanda."""
from interp import read_table, tank_liters


def spline_curvatures(xs, ys, slopes=None):
    """Turunan kedua M_0..M_n spline kubik. slopes=None memberi
    spline natural (M_0 = M_n = 0); slopes=(s0, sn) menjepit kemiringan
    ujung. Satu sistem tridiagonal, diselesaikan dengan algoritma Thomas."""
    n = len(xs) - 1
    h = [xs[i + 1] - xs[i] for i in range(n)]
    low, diag, up, rhs = [0.0] * (n + 1), [0.0] * (n + 1), \
        [0.0] * (n + 1), [0.0] * (n + 1)
    for i in range(1, n):
        low[i], diag[i], up[i] = h[i - 1], 2.0 * (h[i - 1] + h[i]), h[i]
        rhs[i] = 6.0 * ((ys[i + 1] - ys[i]) / h[i]
                        - (ys[i] - ys[i - 1]) / h[i - 1])
    if slopes is None:
        diag[0] = diag[n] = 1.0                  # baris berarti M_0 = M_n = 0
    else:
        diag[0], up[0] = 2.0 * h[0], h[0]
        rhs[0] = 6.0 * ((ys[1] - ys[0]) / h[0] - slopes[0])
        low[n], diag[n] = h[n - 1], 2.0 * h[n - 1]
        rhs[n] = 6.0 * (slopes[1] - (ys[n] - ys[n - 1]) / h[n - 1])
    for i in range(1, n + 1):                    # eliminasi di bawah diagonal
        f = low[i] / diag[i - 1]
        diag[i] -= f * up[i - 1]
        rhs[i] -= f * rhs[i - 1]
    m = [0.0] * (n + 1)
    m[n] = rhs[n] / diag[n]
    for i in range(n - 1, -1, -1):               # substitusi mundur
        m[i] = (rhs[i] - up[i] * m[i + 1]) / diag[i]
    return m


def spline_eval(xs, ys, m, x):
    """Nilai potongan kubik pada selang yang memuat x."""
    i = 0
    while i < len(xs) - 2 and x > xs[i + 1]:
        i += 1
    h = xs[i + 1] - xs[i]
    a = xs[i + 1] - x
    b = x - xs[i]
    return (m[i] * a * a * a / (6.0 * h) + m[i + 1] * b * b * b / (6.0 * h)
            + (ys[i] / h - m[i] * h / 6.0) * a
            + (ys[i + 1] / h - m[i + 1] * h / 6.0) * b)
```

Kode 7.3 (Lanjutan) *spline.py: spline kubik natural dan terjepit melalui sistem tridiagonal kelengkungan dan algoritma Thomas.*

```
def worst_error(xs, ys, m, lo, hi):
    worst = 0.0
    for k in range(int(round(10 * lo)), int(round(10 * hi)) + 1):
        h = 0.1 * k
        worst = max(worst, abs(spline_eval(xs, ys, m, h) - tank_liters(h)))
    return worst


def main():
    table10 = read_table("../data/dipstick.csv")
    xs = [h for h, _ in table10 if h % 20 == 0]
    ys = [v for h, v in table10 if h % 20 == 0]
    nat = spline_curvatures(xs, ys)
    cla = spline_curvatures(xs, ys, (0.0, 0.0))  # tangki datar di kedua ujung
    print("curvatures M_i = S''(x_i) at the 20 cm marks")
    print("  natural " + " ".join(f"{v:7.4f}" for v in nat))
    print("  clamped " + " ".join(f"{v:7.4f}" for v in cla))
    print()
    print("depth   table  natural   error  clamped   error")
    for h, v in table10:
        if h % 20 == 0:
            continue
        s1 = spline_eval(xs, ys, nat, h)
        s2 = spline_eval(xs, ys, cla, h)
        print(f"{h:5.0f}{v:9.2f}{s1:9.2f}{s1 - v:8.2f}{s2:9.2f}{s2 - v:8.2f}")
    print()
    x10 = [h for h, _ in table10]
    y10 = [v for _, v in table10]
    print("largest error against the exact tank formula")
    print("  marks  ends       0..120 cm   20..100 cm")
    for name, a, b in (("20 cm", xs, ys), ("10 cm", x10, y10)):
        for kind, sl in (("natural", None), ("clamped", (0.0, 0.0))):
            m = spline_curvatures(a, b, sl)
            print(f"  {name}  {kind}{worst_error(a, b, m, 0, 120):10.2f} L"
                  f"{worst_error(a, b, m, 20, 100):11.2f} L")
    m10 = spline_curvatures(x10, y10, (0.0, 0.0))
    print(f"a reading of 47 cm: {spline_eval(x10, y10, m10, 47.0):.1f} L"
          f" (exact {tank_liters(47.0):.1f} L)")


if __name__ == "__main__":
    main()
```

Kode 7.4 `Spline.java`: spline kubik dalam Java.

```java
import java.io.IOException;
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;

/** Spline kubik natural dan terjepit melalui tabel tongkat ukur: satu sistem
 *  tridiagonal kecil untuk kelengkungan, lalu kubik di antara dua tanda. */
public class Spline {
    /** Turunan kedua M_0..M_n spline kubik. slopes == null memberi
     *  spline natural (M_0 = M_n = 0); slopes = {s0, sn} menjepit kemiringan
     *  ujung. Satu sistem tridiagonal, diselesaikan dengan algoritma Thomas. */
    static double[] curvatures(double[] xs, double[] ys, double[] slopes) {
        int n = xs.length - 1;
        double[] h = new double[n];
        for (int i = 0; i < n; i++) h[i] = xs[i + 1] - xs[i];
        double[] low = new double[n + 1], diag = new double[n + 1],
                 up = new double[n + 1], rhs = new double[n + 1];
        for (int i = 1; i < n; i++) {
            low[i] = h[i - 1];
            diag[i] = 2.0 * (h[i - 1] + h[i]);
            up[i] = h[i];
            rhs[i] = 6.0 * ((ys[i + 1] - ys[i]) / h[i]
                            - (ys[i] - ys[i - 1]) / h[i - 1]);
        }
        if (slopes == null) {
            diag[0] = diag[n] = 1.0;                // baris berarti M_0 = M_n = 0
        } else {
            diag[0] = 2.0 * h[0];
            up[0] = h[0];
            rhs[0] = 6.0 * ((ys[1] - ys[0]) / h[0] - slopes[0]);
            low[n] = h[n - 1];
            diag[n] = 2.0 * h[n - 1];
            rhs[n] = 6.0 * (slopes[1] - (ys[n] - ys[n - 1]) / h[n - 1]);
        }
        for (int i = 1; i <= n; i++) {              // eliminasi di bawah diagonal
            double f = low[i] / diag[i - 1];
            diag[i] -= f * up[i - 1];
            rhs[i] -= f * rhs[i - 1];
        }
        double[] m = new double[n + 1];
        m[n] = rhs[n] / diag[n];
        for (int i = n - 1; i >= 0; i--) {          // substitusi mundur
            m[i] = (rhs[i] - up[i] * m[i + 1]) / diag[i];
        }
        return m;
    }
```

Kode 7.4 (Lanjutan) *Spline.java: spline kubik dalam Java.*

```
    /** Nilai potongan kubik pada selang yang memuat x. */
    static double eval(double[] xs, double[] ys, double[] m, double x) {
        int i = 0;
        while (i < xs.length - 2 && x > xs[i + 1]) i++;
        double h = xs[i + 1] - xs[i];
        double a = xs[i + 1] - x;
        double b = x - xs[i];
        return m[i] * a * a * a / (6.0 * h) + m[i + 1] * b * b * b / (6.0 * h)
               + (ys[i] / h - m[i] * h / 6.0) * a
               + (ys[i + 1] / h - m[i + 1] * h / 6.0) * b;
    }

    static double worstError(double[] xs, double[] ys, double[] m,
                             double lo, double hi) {
        double worst = 0;
        for (long k = Math.round(10 * lo); k <= Math.round(10 * hi); k++) {
            double h = 0.1 * k;
            worst = Math.max(worst,
                             Math.abs(eval(xs, ys, m, h) - Interp.tankLiters(h)));
        }
        return worst;
    }

    public static void main(String[] args) throws IOException {
        double[][] table10 = Interp.readTable("../data/dipstick.csv");
        int n20 = 0;
        for (double[] row : table10) if (row[0] % 20 == 0) n20++;
        double[] xs = new double[n20], ys = new double[n20];
        double[] x10 = new double[table10.length], y10 = new double[table10.length];
        for (int i = 0, k = 0; i < table10.length; i++) {
            x10[i] = table10[i][0];
            y10[i] = table10[i][1];
            if (table10[i][0] % 20 == 0) {
                xs[k] = table10[i][0];
                ys[k++] = table10[i][1];
            }
        }
        double[] flat = {0.0, 0.0};                 // tangki datar di kedua ujung
        double[] nat = curvatures(xs, ys, null);
        double[] cla = curvatures(xs, ys, flat);
        System.out.println("curvatures M_i = S''(x_i) at the 20 cm marks");
        StringBuilder a = new StringBuilder("  natural");
        StringBuilder b = new StringBuilder("  clamped");
        for (int i = 0; i < nat.length; i++) {
            a.append(" ").append(f(nat[i], 4, 7));
```

Kode 7.4 (Lanjutan) *Spline.java: spline kubik dalam Java.*

```
            b.append(" ").append(f(cla[i], 4, 7));
        }
        System.out.println(a);
        System.out.println(b);
        System.out.println();
        System.out.println("depth   table  natural   error  clamped   error");
        for (double[] row : table10) {
            double h = row[0], v = row[1];
            if (h % 20 == 0) continue;
            double s1 = eval(xs, ys, nat, h), s2 = eval(xs, ys, cla, h);
            System.out.println(f(h, 0, 5) + f(v, 2, 9) + f(s1, 2, 9)
                    + f(s1 - v, 2, 8) + f(s2, 2, 9) + f(s2 - v, 2, 8));
        }
        System.out.println();
        System.out.println("largest error against the exact tank formula");
        System.out.println("  marks  ends       0..120 cm   20..100 cm");
        String[] names = {"20 cm", "10 cm"};
        double[][][] data = {{xs, ys}, {x10, y10}};
        for (int d = 0; d < 2; d++) {
            for (int kind = 0; kind < 2; kind++) {
                double[] px = data[d][0], py = data[d][1];
                double[] m = curvatures(px, py, kind == 0 ? null : flat);
                System.out.println("  " + names[d] + "  "
                        + (kind == 0 ? "natural" : "clamped")
                        + f(worstError(px, py, m, 0, 120), 2, 10) + " L"
                        + f(worstError(px, py, m, 20, 100), 2, 11) + " L");
            }
        }
        double[] m10 = curvatures(x10, y10, flat);
        System.out.println("a reading of 47 cm: " + f(eval(x10, y10, m10, 47.0), 1)
                           + " L (exact " + f(Interp.tankLiters(47.0), 1) + " L)");
    }

    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }
```

Kode 7.4 (Lanjutan) *Spline.java: spline kubik dalam Java.*

```
    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                  + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }

    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }

    static String padRight(String s, int w) {
        return s + " ".repeat(Math.max(0, w - s.length()));
    }
}
```

Kode 7.4 mencerminkan versi Python fungsi demi fungsi; kemiringan ujung dioper sebagai larik, atau `null` untuk spline natural.

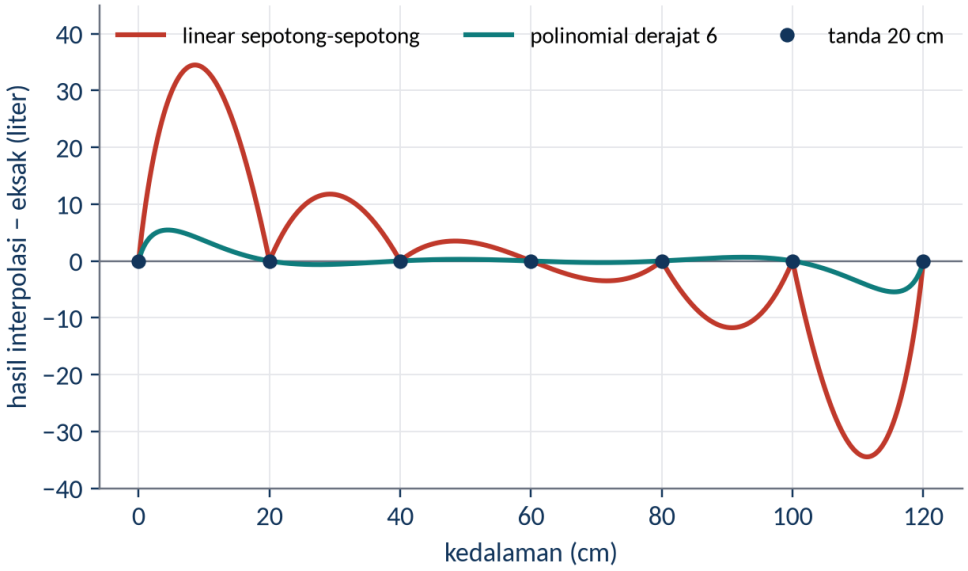
7.6 Menjalankannya

Di folder `java`, `javac -d out *.java` mengompilasi kedua program; `java -cp out Interp` dan `java -cp out Spline` menjalankannya. Di folder `python`, `python3 interp.py` dan `python3 spline.py`. Keluarannya identik dalam kedua bahasa.

```
Newton coefficients f[x0..xk] from the 20 cm marks
  k = 0:  0.000000e+00
  k = 1:  1.239000e+01
  k = 2:  2.055250e-01
  k = 3: -2.201875e-03
  k = 4:  1.223177e-05
  k = 5: -1.223958e-07
  k = 6:  2.170139e-12

depth   table   linear   error  Lagrange   Newton   error
   10    90.03   123.90   33.87     93.62    93.62    3.59
   30   442.21   453.91   11.70    441.62   441.62   -0.59
   50   892.09   895.50    3.40    892.37   892.37    0.28
   70  1369.86  1366.45   -3.41   1369.57  1369.57   -0.29
   90  1819.73  1808.04  -11.69   1820.34  1820.34    0.61
  110  2171.92  2138.05  -33.87   2168.32  2168.32   -3.60
largest error on 0..120 cm: linear 34.49 L, degree 6 5.45 L

Runge's function 1/(1+25x^2) on [-1, 1]: largest error
   n   equally spaced   Chebyshev points
   5           0.4327             0.5559
  10           1.9156             0.1092
  15           2.1076             0.0831
  20          59.8223             0.0153
```

Koefisien Newton $k = 0$ sampai 3 adalah yang ada di **Tabel 7.1**. Yang terakhir, 2.17×10^{-12}, perlu dicatat: volume eksaknya simetris terhadap tanda setengah penuh, dan untuk data eksak koefisien ini seharusnya nol; yang kita lihat adalah pembulatan tabel cetak ke dua desimal, yang diperbesar oleh selisih terbagi. Tabel galat memperlihatkan kedua metode berdampingan pada keenam baris 10 cm yang tidak dimiliki bagan dinding. Garis lurus meleset 33.9 L pada 10 cm dan

110 cm, 11.7 L pada 30 cm dan 90 cm, dan 3.4 L di tengah; polinomial derajat 6 meleset 3.6, 0.6, dan 0.3 L di tempat yang sama. Lagrange dan Newton mencetak angka yang sama, sebagaimana mestinya: keduanya dua cara menulis satu-satunya polinomial interpolasi. Pada kisi halus, galat terburuk garis lurus 34.5 L dan galat terburuk polinomial 5.45 L. **Gambar 7.3** memperlihatkan kedua kurva galat di seluruh kedalaman: setiap potongan lurus memiliki lengkungnya sendiri, paling tinggi di dekat ujung; polinomialnya hampir sempurna di tengah dan bergoyang di ujung.



Gambar 7.3 Galat interpolasi linear sepotong-sepotong dan polinomial derajat 6 dari tanda-tanda 20 cm, diukur terhadap rumus eksak tangki.

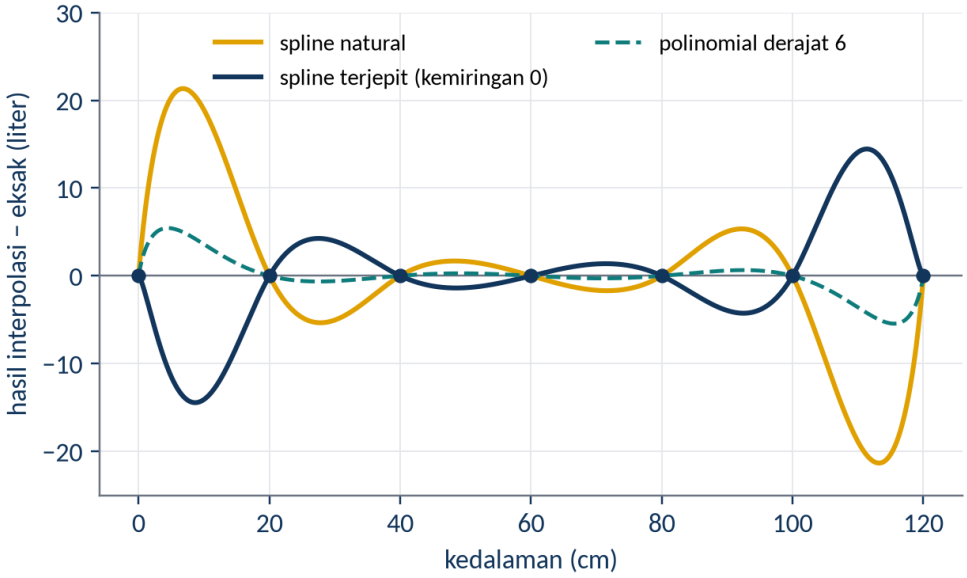
Terjemahkan tabel itu ke dalam pemeriksaan sore Pak Darto. Pada hari ketika tongkat ukur turun dari 88 cm di pagi hari menjadi 70 cm di sore hari, volume eksaknya 1777.7 L dan 1369.9 L, jadi 407.9 L keluar dari tangki; garis lurus dari bagan dinding menyebut 1766.8 L dan 1366.5 L, selisih 400.4 L, hanya kurang 7.5 L. Di dekat dasar ceritanya lain. Dari 28 cm ke 12 cm selisih eksaknya 283.4 L, sedangkan bagan dinding dengan garis lurus menyebut 412.7 L dikurangi 148.7 L, yaitu 264.0 L: celah 19.4 L antara yang dikatakan pompa telah terjual dan yang tampaknya keluar dari tangki, padahal baik pompa maupun tangki tidak bersalah. Galatnya tidak saling meniadakan antara bacaan pagi dan sore, karena keduanya berada di bagian tangki tempat garis lurus membaca terlalu tinggi, dan makin dekat ke dasar makin terlalu tinggi. Itulah persis jenis selisih tak terjelaskan yang dicurigai sang keponakan, dan asalnya dari cara membaca tabel, bukan dari bahan bakarnya.

Tabel Runge membenarkan Subbab 7.3.6: galat titik berjarak sama tumbuh dari 0.43 menjadi 59.8 ketika n bergerak dari 5 ke 20, galat Chebyshev menyusut dari 0.56 menjadi 0.015. (Untuk $n = 5$ simpul berjarak sama kebetulan sedikit lebih baik; keunggulan titik Chebyshev ada pada kecenderungannya, dan itulah yang penting ketika Anda menambah titik.)

Sekarang spline-nya:

```
curvatures M_i = S''(x_i) at the 20 cm marks
  natural  0.0000  0.5990  0.0705  0.0001 -0.0706 -0.5989  0.0000
  clamped  1.7989  0.1193  0.1904  0.0001 -0.1905 -0.1192 -1.7989

depth   table  natural   error  clamped   error
   10    90.03   108.93   18.90    75.95  -14.08
   30   442.21   437.17   -5.04   446.17    3.96
   50   892.09   893.73    1.64   890.73   -1.36
   70  1369.86  1368.21   -1.65  1371.21    1.35
   90  1819.73  1824.78    5.05  1815.78   -3.95
  110  2171.92  2153.02  -18.90  2186.00   14.08

largest error against the exact tank formula
  marks  ends       0..120 cm   20..100 cm
  20 cm  natural     21.34 L       5.35 L
  20 cm  clamped     14.45 L       4.26 L
  10 cm  natural      7.67 L       0.50 L
  10 cm  clamped      5.08 L       0.41 L
a reading of 47 cm: 821.4 L (exact 821.4 L)
```

Kelengkungan spline natural bernilai 0 di kedua ujung karena ditetapkan begitu, sedangkan spline terjepit menemukan kelengkungan besar di sana (1.80), dan itulah yang sebenarnya dimiliki tangki: V'' sejati tak hingga di kedua ujung. Kedua spline memiliki $M_3 \approx 0$ di 60 cm, tempat kurvanya berbelok (titik infleksi). Pada 50 cm spline natural meleset 1.64 L dan yang terjepit 1.36 L, keduanya lebih baik daripada garis lurus (3.4 L), tetapi keduanya lebih buruk daripada polinomial derajat 6 (0.28 L). Di dekat ujung, spline meleset 14 sampai 19 L, lebih baik daripada garis lurus (34 L) tetapi sekali lagi lebih buruk daripada polinomial (3.6 L). **Gambar 7.4** membandingkan kurva-kurva galatnya.



Gambar 7.4 Galat spline natural, spline terjepit (kemiringan ujung 0), dan polinomial derajat 6 melalui tanda-tanda 20 cm. Di dekat ujung, tempat kelengkungan tangki tak hingga, spline paling menderita.

Mengapa spline yang tersohor itu kalah dari polinomial biasa di sini? Dua alasan, keduanya mendidik. Pertama, fungsi tangki analitik di dalam selang dan simetris, dan polinomial derajat 6 yang dicocokkan pada tujuh nilai eksak menangkap bentuk keseluruhannya dengan sangat baik; batas galat Persamaan (7.7) kecil di tengah. Kedua, jaminan ketelitian spline di Subbab 7.4.5 mengandaikan turunan keempat yang terbatas, dan di dekat ujung tangki bahkan turunan keduanya tidak terbatas: volume tumbuh seperti $h^{3/2}$ di dekat dasar, dan tidak ada kubik yang dapat mengikutinya. Spline natural memperparah keadaan dengan memaksa $S'' = 0$ tepat di

tempat kelengkungan sebenarnya tak hingga; menjepit dengan kemiringan yang benar membantu (21.3 L turun menjadi 14.5 L), dan membagi dua jarak tanda menjadi 10 cm jauh lebih membantu (5.1 L dengan ujung terjepit, dan hanya 0.41 L di antara 20 dan 100 cm). Dengan ketiga belas baris tabel dan ujung terjepit, bacaan 47 cm adalah 821.4 L, cocok dengan rumus eksak sampai sepersepuluh liter yang ditampilkan.

> **PERINGATAN:** "Pakai spline" adalah pilihan bawaan yang baik untuk data yang mulus, tetapi kata *mulus* itulah yang memikul beban. Bila data berasal dari fungsi yang bersudut, berpatahan, atau berkelakuan seperti akar kuadrat, ukurlah galatnya sebelum memercayai interpolan apa pun, dan tambahkan pengukuran di tempat fungsinya berulah.

Apa arti setiap pilihan bagi pemeriksaan sore Pak Darto? Dengan bagan dinding dan garis lurus, bacaan di dekat dasar atau puncak bisa meleset sampai 34 L, jauh lebih besar daripada selisih yang sedang ia coba jelaskan. Dengan tabel 10 cm yang lengkap dan garis lurus, galat terburuknya turun menjadi 13.0 L di dekat ujung dan 3.6 L di antara 20 dan 100 cm. Dengan tabel 10 cm dan spline terjepit, galatnya turun menjadi 5.1 L di dekat ujung dan 0.4 L di tengah, dan dengan rumus eksak Persamaan (7.1) galat itu lenyap. Tangki yang dijaga di antara seperempat dan tiga perempat penuh, seperti kebanyakan tangki yang dipakai sehari-hari, hidup di rentang tengah, tempat setiap metode kecuali garis lurus yang kasar sudah cukup baik; ujung-ujunglah tempat pemilik yang cermat memakai rumus atau membaca dengan ekstra hati-hati.

7.7 Memeriksa dengan Pustaka

NumPy dan SciPy menyediakan setiap interpolan di bab ini [6], [8]. **Kode 7.5** mengevaluasi polinomial dan spline kita pada kisi halus lalu membandingkannya dengan `BarycentricInterpolator` (polinomial Lagrange dalam bentuk yang stabil secara numerik [4]), `numpy.interp` (linear sepotong-sepotong), dan `CubicSpline` dengan syarat ujung natural, terjepit, dan bawaan.

Kode 7.5 `compare_scipy.py`: interpolan kita dibandingkan dengan `numpy.interp`, `BarycentricInterpolator`, dan `CubicSpline`.

```python
"""Periksa interpolan kita dengan numpy.interp dan interpolator SciPy."""
import numpy as np
import scipy
from scipy.interpolate import BarycentricInterpolator, CubicSpline

import spline
from interp import newton_eval, divided_differences, read_table, tank_liters


def main():
    print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
    rows = read_table("../data/dipstick.csv")
    xs = [h for h, _ in rows if h % 20 == 0]
    ys = [v for h, v in rows if h % 20 == 0]
    t = np.linspace(0.0, 120.0, 1201)
    exact = np.array([tank_liters(h) for h in t])

    coef = [col[0] for col in divided_differences(xs, ys)]
    ours = np.array([newton_eval(xs, coef, h) for h in t])
    bary = BarycentricInterpolator(xs, ys)(t)
    print(f"degree 6: Newton vs BarycentricInterpolator"
          f" {np.max(np.abs(ours - bary)):.1e}")
    lin = np.interp(t, xs, ys)
    print(f"numpy.interp largest error {np.max(np.abs(lin - exact)):.2f} L")

    for kind, sl in (("natural", None), ("clamped", (0.0, 0.0))):
        m = spline.spline_curvatures(xs, ys, sl)
        mine = np.array([spline.spline_eval(xs, ys, m, h) for h in t])
        bc = "natural" if sl is None else ((1, 0.0), (1, 0.0))
        cs = CubicSpline(xs, ys, bc_type=bc)(t)
        print(f"{kind}: ours vs CubicSpline {np.max(np.abs(mine - cs)):.1e},"
              f" largest error {np.max(np.abs(cs - exact)):.2f} L")
    cs = CubicSpline(xs, ys)(t)                       # bawaan: not-a-knot
    print(f"not-a-knot (SciPy default): largest error"
          f" {np.max(np.abs(cs - exact)):.2f} L")


if __name__ == "__main__":
    main()
```

```
NumPy 2.5.3, SciPy 1.18.1
degree 6: Newton vs BarycentricInterpolator 9.1e-13
numpy.interp largest error 34.49 L
natural: ours vs CubicSpline 4.5e-13, largest error 21.34 L
clamped: ours vs CubicSpline 1.2e-12, largest error 14.45 L
not-a-knot (SciPy default): largest error 7.87 L
```

Polinomial derajat 6 cocok dengan bentuk barisentrik SciPy sampai 9×10^{-13} L, garis lurus dengan `numpy.interp` secara tepat (galat terburuk yang sama, 34.49 L), dan kedua spline dengan `CubicSpline` sampai sekitar 10^{-12}. Syarat ujung bawaan SciPy, *not-a-knot*, yang tidak berpura-pura mengetahui kelengkungan di ujung, memberi galat terburuk 7.87 L: lebih baik daripada natural (21.34) maupun terjepit (14.45), dan tetap belum sebaik polinomial (5.45). Di SciPy 1.18, rutin FITPACK lama berbahasa Fortran di balik sebagian fungsi spline telah dipindahkan ke C, sebuah modernisasi yang tidak mengubah apa pun bagi pemakai [6].

7.8 Membuatnya Lebih Baik

- **Pakailah apa yang Anda ketahui.** Tangki Pak Darto berbentuk silinder, dan Persamaan (7.1) memberi volume pada kedalaman berapa pun tanpa interpolasi. Setiap kali sebuah tabel berasal dari rumus yang diketahui, pakailah rumusnya. Interpolasi diperuntukkan bagi tabel yang asal-usulnya tidak diketahui atau terlalu rumit untuk dimodelkan.

- **Tempatkan tanda di tempat fungsinya melengkung.** Jarak seperti Chebyshev, rapat di dekat ujung, cocok dengan perilaku akar kuadrat tangki. Tujuh tanda pada kedalaman Chebyshev (1.5, 13.1, 34.0, 60.0, 86.0, 106.9, dan 118.5 cm) memberi polinomial derajat 6 dengan galat terburuk 3.95 L, dibandingkan 5.45 L untuk jarak yang sama (Soal 4).
- **Interpolasi monoton.** Volume hanya bertambah seiring kedalaman, tetapi spline atau polinomial berderajat tinggi dapat menukik di antara dua titik dan membuat tambahan satu sentimeter bahan bakar "kehilangan" liter. Kubik sepotong-sepotong yang mempertahankan bentuk (PCHIP, `scipy.interpolate.PchipInterpolator`) memilih kemiringan sehingga interpolannya monoton di mana pun datanya monoton [6].
- **Evaluasi barisentrik.** Bentuk Lagrange yang ditulis ulang sebagai $p(x) = \sum_i \frac{w_i}{x-x_i} y_i / \sum_i \frac{w_i}{x-x_i}$, dengan bobot yang dihitung sekali, dievaluasi dalam $O(n)$ dan stabil secara numerik; pada titik Chebyshev inilah metode pilihan untuk interpolasi polinomial berderajat berapa pun [4].

7.9 Memakainya dengan Sungguh-Sungguh

Untuk tangki, drum, atau jeriken Anda sendiri yang bentuknya tidak biasa, program ini bekerja dengan tabel apa pun. Bila Anda tidak memiliki tabel, buatlah: tuangkan volume yang diketahui sedikit demi sedikit (jeriken 20 liter memudahkan), masukkan tongkat ukur setelah setiap tuangan, dan catat kedalamannya. Masukkan baris-barisnya ke `data/dipstick.csv`. Ukurlah lebih sering di dekat dasar dan puncak, tempat bentuknya berubah paling cepat. Lalu bandingkan bacaan program dengan beberapa tuangan tambahan yang tidak Anda masukkan ke tabel; bila galat di sana dapat diterima, tabelnya sudah cukup baik. Simpan tabel yang sudah jadi bersama dokumen tangki, dan ulangi uji tuangan setiap kali tangki dipindahkan atau diletakkan ulang, karena tangki yang miring beberapa sentimeter dari ujung ke ujung mengubah setiap bacaan, terutama di dekat dasar. Dengan tabel buatan sendiri yang teruji seperti itu, pemeriksaan sore menjadi pekerjaan lima menit yang dapat dipercaya. Ingatlah bahwa tangki bahan bakar adalah tempat untuk berhati-hati, bukan bereksperimen: ukurlah dengan pompa mati, jangan pernah ada api atau telepon genggam di dekat lubangnya, dan serahkan hal yang tidak biasa (bau, noda di tanah, bacaan yang terus turun di malam hari) kepada teknisi pemasok.

Beberapa baris yang sama dapat menginterpolasi tabel apa pun. Prakiraan BMKG memberi suhu setiap tiga jam; spline yang melaluinya memberi kurva mulus untuk sepanjang hari (Soal 5 menunjukkan bahwa kurva itu mungkin sedikit melampaui nilai prakiraan tertinggi). Tabel pemberi pinjaman memberi angsuran bulanan untuk beberapa besar pinjaman; interpolasi linear di antara besar pinjaman yang bertetangga biasanya cukup teliti di sana, karena angsuran tumbuh hampir linear terhadap besar pinjaman. Tabel muatan-tegangan telepon genggam memungkinkan pemantau baterai kecil menampilkan persentase. Dalam setiap kasus: perhatikan bentuk datanya, pilih linear, polinomial berderajat rendah, atau spline, dan ukurlah galatnya pada titik-titik yang tidak Anda pakai.

7.10 Metode Numerik dalam Praktik: Profitina

Dasbor analitik historis Profitina memperlihatkan, untuk setiap merek, bagaimana skor visibilitas AI-nya bergerak minggu demi minggu, dan menawarkan riwayat itu sebagai ekspor CSV. Skor hanya ada pada tanggal pemindaian: di antara dua pemindaian mingguan tidak ada pengukuran. Ketika grafik menarik garis di antara titik-titik mingguan, grafik itu melakukan interpolasi, dan bab ini menjelaskan garis jenis apa yang jujur. Potongan lurus tidak mengklaim apa pun selain "di suatu tempat di antara kedua nilai ini". Spline mulus yang melalui titik-titik itu tampak lebih

profesional, tetapi dapat melampaui sasaran, memperlihatkan puncak di atas skor tertinggi yang sebenarnya atau lembah di bawah skor terendah, nilai yang tidak pernah diukur. Pada dasbor yang dibaca pemilik merek yang bukan ahli analisis numerik, puncak rekaan adalah kebohongan kecil. Pilihan yang aman adalah potongan lurus atau interpolan monoton yang mempertahankan bentuk seperti PCHIP, yang tidak pernah menciptakan maksimum atau minimum yang tidak dimiliki datanya.

Pertanyaan yang sama muncul di mana pun suatu besaran diketahui menurut jadwal dan diperlukan di antaranya. Emisi koin utilitas PROFIT mengikuti jadwal yang meluruh, dan nilainya dalam rupiah berasal dari oracle PROFIT ke IDR yang menerbitkan harga pada saat-saat tertentu. Membaca jadwal atau harga seperti itu di antara titik-titik terbitnya adalah interpolasi, dan pilihan antara garis lurus, polinomial, dan spline adalah pilihan tentang apa yang akan dikatakan kepada pembaca tentang yang terjadi di antaranya.

7.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menaikkan derajat demi ketelitian.** Dengan titik berjarak sama, polinomial berderajat lebih tinggi bisa jauh lebih buruk (Runge, Persamaan (7.8)). Pakailah titik Chebyshev bila Anda dapat memilihnya, dan spline atau derajat rendah sepotong-sepotong bila tidak.
- **Ekstrapolasi.** Rumus interpolasi dirancang untuk titik di antara simpul. Di luar tabel, polinomial simpul dalam Persamaan (7.7) tumbuh cepat, begitu pula galatnya; bacaan 125 cm dari tabel tangki kita tidak bermakna, dan harus ditolak, seperti yang dilakukan `linear`.
- **Menyelesaikan sistem Vandermonde.** Hasilnya polinomial yang sama dengan Lagrange dan Newton, tetapi dengan matriks yang kondisinya buruk. Pakailah bentuk Newton atau barisentrik.
- **Memercayai syarat ujung natural.** $S'' = 0$ di ujung adalah asumsi. Bila Anda mengetahui kemiringan ujung, jepitlah; bila Anda tidak mengetahui apa-apa, *not-a-knot* biasanya lebih aman.
- **Lupa bahwa spline dapat melampaui sasaran.** Spline yang melalui data monoton belum tentu monoton. Untuk besaran yang hanya dapat bertambah (volume kumulatif, total) atau harus tetap dalam suatu rentang (persentase), pakailah PCHIP atau potongan lurus.
- **Selang tak sama dalam rumus spline.** Persamaan (7.12) ditulis untuk h_i umum. Bentuk sederhana $M_{i-1} + 4M_i + M_{i+1}$ hanya berlaku untuk jarak yang sama; memakainya pada tabel yang tidak rata menghasilkan spline yang salah tanpa peringatan.
- **Menginterpolasi pengukuran yang berderau.** Interpolan melewati setiap titik, termasuk galat-galatnya. Bila tabel berasal dari pengukuran kasar, seperti bacaan tongkat ukur dengan mata, kurva yang melalui setiap titik akan mengikuti deraunya; cocokkan kurva mulus dengan kuadrat terkecil sebagai gantinya (Bab 8).
- **Membulatkan tabel, lalu membaca terlalu banyak digit.** Koefisien Newton terakhir di Subbab 7.6 murni derau pembulatan dari tabel yang dicetak dengan dua desimal. Ketelitian interpolan apa pun dibatasi oleh ketelitian datanya.

7.12 Ringkasan Bab

- Interpolasi membangun fungsi melalui titik-titik tabel dan membacanya di antaranya. Garis lurus di antara tetangga (Persamaan (7.2)) sederhana dan galatnya paling besar $\frac{h^2}{8}\max|f''|$, 34 L di dekat ujung tangki (Persamaan (7.1), **Gambar 7.1**) dan 3.4 L di tengah.

- Melalui $n+1$ titik terdapat tepat satu polinomial berderajat paling tinggi n. Bentuk Lagrange (Persamaan (7.3)) menuliskannya dengan polinomial basis; bentuk Newton (Persamaan (7.4)) dengan selisih terbagi (Persamaan (7.5), **Tabel 7.1**), yang dapat diperluas ke titik baru dengan satu suku dan dievaluasi dalam n perkalian (Persamaan (7.6)).
- Galat interpolasi polinomial adalah faktor turunan dikalikan polinomial simpul (Persamaan (7.7)). Simpul berjarak sama membuatnya meledak di dekat ujung untuk fungsi Runge (Persamaan (7.8), **Gambar 7.2**); titik Chebyshev (Persamaan (7.9)) menjaganya tetap kecil secara merata.
- Bila kemiringan diketahui selain nilai, interpolasi Hermite memakainya (Persamaan (7.10)); di dasar tangki ia memangkas galat garis lurus dua pertiga.
- Spline kubik adalah rantai kubik yang mulus (Persamaan (7.11)). Kelengkungannya menyelesaikan sistem tridiagonal yang dominan diagonal sejati (Persamaan (7.12)), yang diselesaikan dengan algoritma Thomas (Persamaan (7.13)) dalam waktu linear. Ujung natural, terjepit, dan *not-a-knot* memasok dua syarat yang kurang.
- Untuk tangki, polinomial derajat 6 mengalahkan spline (**Gambar 7.3**, **Gambar 7.4**), karena kelengkungan tangki tak hingga di ujung-ujungnya; menjepit kemiringan ujung yang diketahui dan tanda yang lebih rapat paling membantu. Ukurlah sebelum memercayai.
- Program Java dan Python cocok bita demi bita, dan NumPy serta SciPy menghasilkan kembali setiap interpolan sampai sekitar 10^{-12}.

7.13 Soal Latihan

1. Dengan bagan dinding, hitunglah dengan tangan bacaan garis lurus pada 30 cm dan bacaan parabola yang melalui tanda 20, 40, dan 60 cm. Bandingkan keduanya dengan 442.21 L dari pembuat tangki.
2. Perluas **Tabel 7.1** dengan satu baris: tambahkan tanda 80 cm (1601.93 L), hitung selisih terbagi yang baru, dan periksa koefisien teratas yang baru terhadap nilai program untuk $k = 4$. Koefisien bentuk Newton mana yang berubah?
3. Pakailah batas galat interpolasi linear untuk menaksir galat terburuk bagan dinding di antara 20 dan 40 cm, dengan $\max|V''|$ pada selang itu diambil di 20 cm. Bandingkan dengan galat yang diukur program pada 30 cm.
4. Ganti tanda-tanda berjarak sama dengan tujuh kedalaman Chebyshev pada [0,120] dari Persamaan (7.9) (dipetakan ke selang itu), dengan volume dari Persamaan (7.1). Dengan fungsi-fungsi dalam `interp.py`, carilah galat terburuk polinomial derajat 6 pada 0 sampai 120 cm.
5. Prakiraan BMKG untuk sebuah kota memberi suhu berikut (sebagai contoh): 24 °C pukul 07.00, 29 °C pukul 10.00, 32 °C pukul 13.00, 30 °C pukul 16.00, 27 °C pukul 19.00, dan 25 °C pukul 22.00. Pakailah `spline_curvatures` dan `spline_eval` dengan ujung natural untuk menaksir suhu pukul 11.30 dan suhu tertinggi hari itu. Bandingkan dengan interpolasi linear. Apakah maksimum spline itu masuk akal?
6. Tunjukkan bahwa koefisien Newton $f[x_0, x_1, x_2]$ tepat sama dengan $1/2\, f''$ bila f polinomial kuadrat, dan jelaskan mengapa kolom kedua tabel untuk tangki (0.2055 dan 0.0734) menyatakan bahwa tangki lebih melengkung di dekat dasar daripada di tengah.
7. Jelaskan, dengan Persamaan (7.7), mengapa titik Chebyshev memperbaiki fungsi Runge secara dramatis tetapi hanya sedikit memperbaiki tangki (Soal 4).
8. Pakailah rumus titik tengah Hermite Persamaan (7.10) pada selang terakhir bagan dinding, dari 100 cm (2014.15 L) sampai 120 cm (2261.95 L), dengan kemiringan $V'(100)$ dari rumus di

Subbab 7.3.5 dan $V'(120) = 0$, untuk membaca volume pada 110 cm. Bandingkan dengan garis lurus dan dengan 2171.92 L dari pembuat tangki.

7.14 Penyelesaian Soal Latihan

1. Garis lurus di antara 20 cm (247.80) dan 40 cm (660.02): $247.80 + 0.5 \times 412.22 = 453.91$ L, galat $+11.70$ L. Parabola melalui 20, 40, dan 60 cm pada $x = 30$: $\ell_0 = \frac{(30-40)(30-60)}{(20-40)(20-60)} = \frac{300}{800} = 0.375$, $\ell_1 = \frac{(30-20)(30-60)}{(40-20)(40-60)} = \frac{-300}{-400} = 0.75$, $\ell_2 = \frac{(30-20)(30-40)}{(60-20)(60-40)} = \frac{-100}{800} = -0.125$. Jadi $p(30) = 0.375 \times 247.80 + 0.75 \times 660.02 - 0.125 \times 1130.97 = 92.925 + 495.015 - 141.371 = 446.57$ L, galat $+4.36$ L. Parabola memangkas galat hampir dua pertiga.

2. Entri baru: selisih pertama $f[60,80] = (1601.93 - 1130.97)/20 = 23.548$; kedua $f[40,60,80] = (23.548 - 23.5475)/40 = 0.0000125$; ketiga $f[20,\ldots,80] = (0.0000125 - 0.0734125)/60 = -0.00122333$; keempat $f[0,\ldots,80] = (-0.00122333 - (-0.00220188))/80 = 0.0000122319$, yang cocok dengan 1.223177×10^{-5} dari program sampai digit yang kita bawa. Hanya satu koefisien baru yang muncul; keempat yang pertama tidak berubah, dan itulah keunggulan bentuk Newton.

3. $V''(h) = 2L(r-h)/\left(1000\sqrt{2rh - h^2}\right) = 0.4 \times 40/\sqrt{2400 - 400} = 16/44.72 = 0.358$ pada $h = 20$. Batasnya $\frac{20^2}{8} \times 0.358 = 17.9$ L. Galat terukur pada 30 cm adalah 11.70 L, di bawah batas, sebagaimana mestinya; batas itu pesimistis karena V'' di 20 cm lebih besar daripada nilainya di bagian lain selang.

4. Kedalaman Chebyshev adalah $60 - 60\cos\frac{(2i+1)\pi}{14}$ untuk $i = 0,\ldots,6$: 1.5, 13.1, 34.0, 60.0, 86.0, 106.9, dan 118.5 cm. Dengan volume eksak pada kedalaman ini dan `lagrange` yang dievaluasi pada kisi 1 mm, galat terburuknya 3.95 L, dibandingkan 5.45 L dengan tanda berjarak sama. Ini membantu, karena simpul-simpul bergeser ke arah ujung tempat tangki berulah, tetapi tidak mengubah hasilnya secara mendasar.

5. Spline natural memberi 31.02 °C pukul 11.30, interpolasi linear 30.5 °C. Dengan mengevaluasi spline setiap 0.01 jam, nilai tertingginya 32.01 °C sekitar pukul 13.11. Maksimum spline berada tepat di atas nilai prakiraan tertinggi dan beberapa menit sesudahnya; itu masuk akal, karena maksimum suhu harian yang sebenarnya biasanya datang pada awal siang dan tidak harus jatuh tepat pada jam prakiraan. Tetapi prakiraan itu sendiri tidak pasti sampai satu derajat atau lebih, sehingga kelebihan 0.01 °C tidak membawa informasi apa pun. Kelengkungannya $M = (0, -0.134, -0.797, -0.010, 0.169, 0)$ per jam kuadrat: kurvanya paling kuat melengkung ke bawah sekitar pukul 13.00.

6. Bila $f(x) = a + bx + cx^2$, maka $f[x_0, x_1] = b + c(x_0 + x_1)$ dan $f[x_1, x_2] = b + c(x_1 + x_2)$, sehingga $f[x_0, x_1, x_2] = c(x_2 - x_0)/(x_2 - x_0) = c = f''/2$. Secara umum, $f[x_0, x_1, x_2] = f''(\xi)/2$ untuk suatu ξ di selang itu. Selisih terbagi kedua tangki, 0.2055 (dari 0 sampai 40 cm) dan 0.0734 (dari 20 sampai 60 cm), karenanya menaksir $V''/2$ sekitar 0.41 dan 0.15 liter per cm^2 pada bentangan itu: dasar tangki melengkung hampir tiga kali lebih kuat daripada tengahnya.

7. Galatnya adalah $f^{(n+1)}(\xi)/(n+1)!$ dikalikan polinomial simpul. Titik Chebyshev meminimumkan polinomial simpul: faktor kedua mengecil secara merata. Untuk fungsi Runge, masalahnya adalah pertumbuhan polinomial simpul di dekat ujung pada jarak yang sama, digabung dengan turunan yang tumbuh seperti $(n+1)!\,5^{n+1}$; titik Chebyshev menghilangkan masalah pertama dan mengubah divergensi menjadi konvergensi. Untuk tangki, masalahnya ada

pada faktor pertama: turunan-turunan V tak hingga di ujung, sehingga tidak ada pilihan simpul yang dapat membuat rumus galat kecil di sana. Jarak Chebyshev sedikit membantu dengan menaruh simpul lebih dekat ke ujung yang singular, tetapi obat untuk tangki adalah rumus eksak atau jauh lebih banyak pengukuran di dekat ujung.

8. $V'(100) = 0.4\sqrt{2 \times 60 \times 100 - 100^2} = 0.4\sqrt{2000} = 17.89$ L/cm dan $V'(120) = 0$. Garis lurus memberi $(2014.15 + 2261.95)/2 = 2138.05$ L, kurang 33.87 L. Hermite menambahkan $\frac{20}{8}(17.89 - 0) = 44.72$ L dan memberi 2182.77 L, kelebihan 10.85 L. Karena simetri tangki, ini bayangan cermin bacaan pada 10 cm di Subbab 7.3.7: besar galat yang sama, dengan tanda berlawanan.

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] C. Runge, "Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten," *Zeitschrift für Mathematik und Physik*, vol. 46, pp. 224–243, 1901.

[4] L. N. Trefethen, *Approximation Theory and Approximation Practice*, extended ed. Philadelphia, PA, USA: SIAM, 2019.

[5] C. de Boor, *A Practical Guide to Splines*, rev. ed. New York, NY, USA: Springer, 2001.

[6] SciPy Developers, "SciPy 1.18.0 Release Notes," Jun. 19, 2026, and "scipy.interpolate.CubicSpline," SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/release/1.18.0-notes.html and https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.CubicSpline.html

[7] L. H. Thomas, "Elliptic problems in linear difference equations over a network," Watson Scientific Computing Laboratory, Columbia University, New York, NY, USA, 1949.

[8] NumPy Developers, "NumPy 2.5.0 Release Notes," Jun. 21, 2026, and "numpy 2.5.3" on the Python Package Index, Sep. 6, 2026. [Online]. Available: https://numpy.org/doc/2.5/release/2.5.0-notes.html and https://pypi.org/project/numpy/

Bab 8

Pencocokan Kurva: Regresi Kuadrat Terkecil

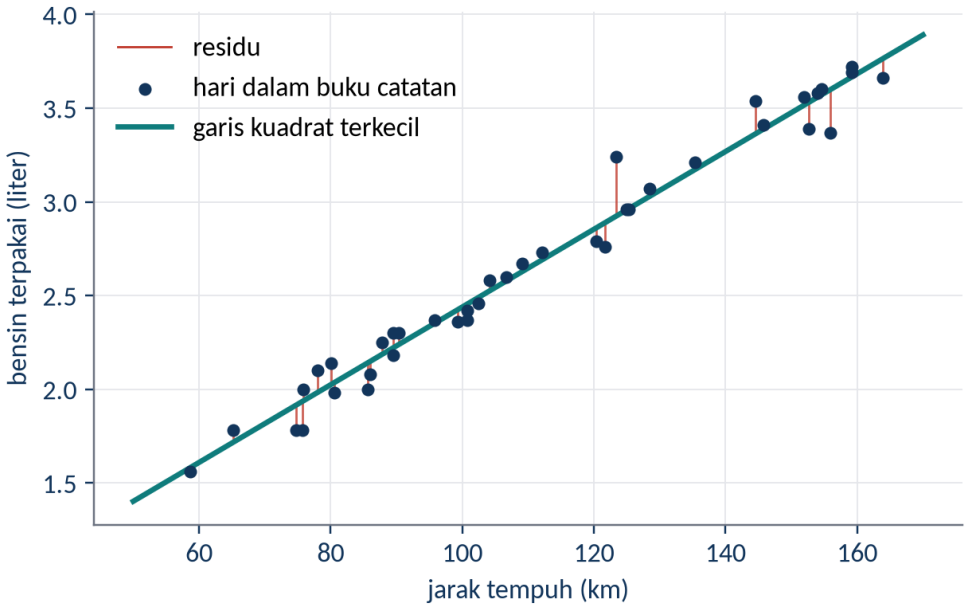
Bab 7 menggambar kurva yang melewati setiap titik tabel secara tepat. Itu benar bila tabelnya eksak, seperti tabel pembuat tangki. Kebanyakan data tidak demikian: data adalah pengukuran dari hari-hari nyata, dan setiap hari memiliki kebetulan-kebetulan kecilnya sendiri. Kurva yang dipaksa melewati setiap titik akan mengikuti kebetulan itu alih-alih polanya. Bab ini meminta lebih sedikit dan memperoleh lebih banyak: kurva sederhana yang melintas *di dekat* semua titik, dipilih sehingga jumlah kuadrat melesetnya sekecil mungkin. Contoh yang menyertai kita adalah buku catatan seorang pengemudi ojek yang ingin memahami pemakaian bensinnya. Di akhir bab ini Anda akan mampu mencocokkan garis lurus dengan kuadrat terkecil secara manual dan menurunkan rumus-rumusnya; mengukur mutu pencocokan dengan residu, R^2, dan galat baku, serta memberi ketidakpastian yang jujur pada sebuah kemiringan; mencocokkan beberapa prediktor sekaligus melalui matriks desain dan metode QR dari Bab 6; mencocokkan polinomial dan kurva eksponensial dengan membuatnya linear dalam koefisiennya; mengenali *overfitting* (pencocokan berlebih) dan memilih model dengan mengujinya pada data yang belum pernah dilihatnya; membangun semuanya dalam Java dan Python; serta memeriksanya dengan NumPy dan SciPy.

8.1 Buku Catatan Mas Dimas

Mas Dimas mengemudikan ojek, mengantar penumpang dan pesanan makanan melalui aplikasi transportasi daring di sebuah kota menengah. Sepeda motornya adalah kantornya, dan bensin adalah biaya hariannya yang terbesar. Selama empat puluh hari kerja ia menyimpan buku catatan kecil: setiap sore ia menuliskan jarak tempuh yang dilaporkan aplikasi, banyaknya perjalanan yang ia lakukan, dan berapa liter bensin yang ia beli untuk mengisi tangki kembali penuh. Pada sebagian hari ia juga mencatat kecepatan rata-rata yang dilaporkan aplikasi, dan setiap dua bulan ia melihat kesehatan baterai yang ditampilkan ponselnya, karena ponsel yang menjalankan aplikasi itu adalah alat kerjanya yang kedua.

Ia memiliki empat pertanyaan. Berapa sebenarnya bensin untuk satu kilometer, agar ia dapat menilai apakah perjalanan jauh dengan tarif tetap layak diambil? Berapa bensin yang harus ia siapkan untuk besok, ketika jadwal aplikasi mengisyaratkan sekitar 120 km dan 17 perjalanan? Pada kecepatan rata-rata berapa motornya menempuh jarak terjauh per liter? Dan kapan baterai ponselnya akan turun ke 80% dari kapasitas awalnya, tingkat yang oleh banyak bengkel ponsel dijadikan saran untuk mengganti baterai?

Tidak satu pun pertanyaan itu memiliki jawaban eksak dalam buku catatan. Pada dua hari dengan jarak yang sama ia membeli bensin dalam jumlah berbeda, karena kemacetan, angin, menunggu lebih lama dengan mesin menyala, atau tangki yang diisi sampai batas yang sedikit berbeda. Titik-titiknya berserakan. Yang ia perlukan adalah *kecenderungan* di balik serakan itu, beserta gambaran jujur tentang seberapa jauh kecenderungan itu dapat dipercaya. **Gambar 8.1** memperlihatkan keempat puluh hari itu: jarak di sumbu mendatar, liter di sumbu tegak. Titik-titiknya jelas berada di dekat sebuah garis lurus. Garis yang mana?



Gambar 8.1 Empat puluh hari dalam buku catatan Mas Dimas: bensin terpakai terhadap jarak tempuh, garis kuadrat terkecil, dan residunya, yaitu meleset tegak yang dibuat sekecil mungkin oleh garis itu dalam jumlah kuadratnya.

8.2 Gagasannya dalam Kata-Kata Sederhana

8.2.1 Meleset, dan mengapa kita mengkuadratkannya

Andaikan kita mencoba garis $y = a + bx$, dengan x jarak dan y bensin. Pada hari ke-i garis itu meramalkan $a + bx_i$, dan buku catatan menyebut y_i. Selisihnya

$$e_i = y_i - (a + bx_i)$$

adalah *residu* hari ke-i: positif bila garis melintas di bawah titik, negatif bila di atasnya. Garis yang baik membuat residunya kecil, semuanya sekaligus. Kita memerlukan satu bilangan yang mengukur "semuanya sekaligus". Jumlah residu tidak berguna, karena meleset positif dan negatif saling meniadakan: garis yang jauh di atas separuh titik dan jauh di bawah separuh lainnya bisa berjumlah nol. Jumlah nilai mutlak berfungsi tetapi canggung diminimumkan (ada sudut-sudutnya). Adrien-Marie Legendre pada 1805, dan Carl Friedrich Gauss, yang mengaku telah memakainya sejak 1795 dan yang pada 1801 memakainya untuk menemukan kembali asteroid Ceres yang hilang, memilih **jumlah kuadrat residu** [1], [2]:

$$\mathrm{SSE}(a, b) = \sum_{i=1}^{n} (y_i - a - bx_i)^2. \tag{8.1}$$

Kuadrat selalu positif, sehingga tidak ada yang saling meniadakan; meleset besar jauh lebih mahal daripada yang kecil, sehingga garis ditarik agar adil terhadap setiap titik; dan terutama, SSE adalah fungsi mulus berbentuk mangkuk dari a dan b yang titik terendahnya ditemukan dengan menyelesaikan persamaan linear. Gauss kemudian menunjukkan bahwa bila galat pengukuran saling bebas dengan sebaran yang sama, tidak ada resep garis lurus lain yang linear terhadap data dan tak bias yang variansinya lebih kecil (teorema Gauss-Markov) [3].

8.2.2 Lima hari dengan tangan

Ambil lima hari pertama dalam buku catatan. Resepnya, yang diturunkan di Subbab 8.3, adalah: hitung rata-rata jarak $\bar{x}$ dan rata-rata bensin $\bar{y}$, lalu dua jumlah, S_{xx} dari kuadrat simpangan x dan S_{xy} dari hasil kali simpangan, dan tetapkan $b = S_{xy}/S_{xx}$ dan $a = \bar{y} - b\bar{x}$. **Tabel 8.1** menguraikan hitungannya. Rata-ratanya $\bar{x} = 537.8/5 = 107.56$ km dan $\bar{y} = 12.96/5 = 2.592$ L.

Tabel 8.1 Garis kuadrat terkecil melalui lima hari pertama, dengan tangan: simpangan dari rata-rata, kuadratnya, dan hasil kalinya.

Hari	x_i (km)	y_i (L)	$x_i - x$	$y_i - y$	$(x_i - x)^2$	$(x_i - x)(y_i - y)$
1	74.7	1.78	−32.86	−0.812	1079.78	26.6823
2	95.8	2.37	−11.76	−0.222	138.30	2.6107
3	154.6	3.60	47.04	1.008	2212.76	47.4163
4	87.8	2.25	−19.76	−0.342	390.46	6.7579
5	124.9	2.96	17.34	0.368	300.68	6.3811
jumlah	537.8	12.96	0	0	$S_{xx} = 4121.97$	$S_{xy} = 89.848$

Jadi $b = 89.848/4121.97 = 0.021797$ L/km dan $a = 2.592 - 0.021797 \times 107.56 = 0.2475$ L. Garis itu menyatakan bahwa setiap kilometer memerlukan sekitar 0.022 L dan bahwa sehari di jalan memerlukan sekitar seperempat liter sebelum kilometer pertama, yaitu bensin yang terbakar saat mesin menyala diam, menunggu pesanan, dan menghidupkan motor. Kelima residunya -0.096, 0.034, -0.017, 0.089, dan -0.010 L; jumlahnya nol, sebagaimana akan dibuktikan Subbab 8.3. Dengan keempat puluh hari, program di Subbab 8.8 menemukan garis yang sedikit berbeda, $0.3652 + 0.020737x$: lima hari adalah sampel kecil, dan kecenderungan yang diisyaratkannya bergeser ketika hari-hari lain berdatangan. Seberapa jauh ia dapat bergeser adalah bagian dari jawaban, dan Subbab 8.3.4 mengukurnya.

8.3 Garis Lurus secara Tepat

8.3.1 Persamaan normal untuk garis

SSE pada Persamaan (8.1) adalah fungsi kuadratik dari dua variabel a dan b. Di titik minimumnya kedua turunan parsialnya nol:

$$\frac{\partial \,\mathrm{SSE}}{\partial a} = -2\sum_i (y_i - a - bx_i) = 0, \qquad \frac{\partial \,\mathrm{SSE}}{\partial b} = -2\sum_i x_i\,(y_i - a - bx_i) = 0.$$

Membagi dengan -2 dan menyusun ulang memberi dua persamaan linear, **persamaan normal** garis lurus:

$$\begin{aligned} n\,a + \left(\sum x_i\right) b &= \sum y_i \\ \left(\sum x_i\right) a + \left(\sum x_i^2\right) b &= \sum x_i y_i \end{aligned} \tag{8.2}$$

Inilah kasus 2×2 dari persamaan normal $A^{\mathsf{T}}A\,x = A^{\mathsf{T}}b$ di Bab 6, dengan satu kolom berisi angka satu dan satu kolom berisi jarak [4], [5]. Membagi persamaan pertama dengan n memberi $a = \bar{y} - b\bar{x}$: *garis itu melewati titik rata-rata* $(\bar{x}, \bar{y})$. Menyubstitusikannya ke persamaan kedua dan menyederhanakan (kurangkan $n\bar{x}$ kali persamaan pertama dari persamaan kedua) menyisakan

$$b = \frac{S_{xy}}{S_{xx}}, \quad a = \bar{y} - b\,\bar{x}, \qquad S_{xx} = \sum_i (x_i - \bar{x})^2\,, \quad S_{xy} = \sum_i (x_i - \bar{x})\,(y_i - \bar{y}). \tag{8.3}$$

Jumlah-jumlah yang dipusatkan ini juga cara hitung yang masuk akal secara numerik: jalan pintas buku teks $S_{xx} = \sum x_i^2 - n\bar{x}^2$ mengurangkan dua bilangan besar yang hampir sama dan dapat kehilangan sebagian besar digitnya bila x_i besar dan berdekatan (pembatalan dari Bab 1). Program kita menghitung simpangannya lebih dahulu.

8.3.2 Dua fakta tentang residu

Persamaan normal pertama menyatakan $\sum_i e_i = 0$: residu garis kuadrat terkecil yang memiliki intersep selalu berjumlah nol. Persamaan kedua menyatakan $\sum_i x_i\, e_i = 0$: residu tidak berkorelasi dengan x, tegak lurus terhadap kolom jarak dalam arti Bab 6. Residu program untuk keempat puluh hari berjumlah 7×10^{-15}, nol sampai pembulatan. Ini bukan sifat data; ini sifat metodenya, dan sekaligus alat pemeriksa: bila residu Anda tidak berjumlah nol, garis Anda bukan garis kuadrat terkecil.

8.3.3 Seberapa baik pencocokannya? R^2 dan galat baku

Dua bilangan merangkum sebuah pencocokan. Yang pertama membandingkan pencocokan dengan model yang paling kasar, "setiap hari memakai jumlah rata-rata". Melesetnya $y_i - \bar{y}$, dan jumlah kuadratnya adalah *jumlah kuadrat total* $\text{SST} = \sum (y_i - \bar{y})^2$. Meleset garis memiliki jumlah kuadrat SSE. **Koefisien determinasi** adalah

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}, \tag{8.4}$$

yaitu bagian keragaman bensin dari hari ke hari yang dijelaskan oleh garis. Untuk keempat puluh hari, SSE $=0.387$ dan SST $=15.91$, sehingga $R^2 = 0.9757$: jarak menjelaskan sekitar 97.6% keragaman itu. R^2 bernilai 1 untuk pencocokan sempurna dan 0 untuk garis yang tidak lebih baik daripada rata-rata.

Bilangan kedua berbicara dalam satuan data. **Galat baku taksiran** (*standard error of the estimate*)

$$s_e = \sqrt{\frac{\text{SSE}}{n-2}} \tag{8.5}$$

adalah ukuran khas sebuah residu. Pembaginya $n-2$, bukan n, karena dua bilangan, a dan b, dicocokkan dari data yang sama, dan pencocokan dengan n titik dan 2 parameter hanya menyisakan $n-2$ derajat kebebasan untuk memperlihatkan melesetnya; dengan dua titik garis itu melewati keduanya dan SSE nol apa pun kebenarannya. Untuk buku catatan ini $s_e = 0.101$ L: bensin sehari biasanya berbeda sekitar sepersepuluh liter dari garis.

8.3.4 Seberapa yakin kemiringannya?

Bila buku catatan diteruskan empat puluh hari lagi, kemiringan yang dicocokkan akan sedikit berbeda. Di bawah asumsi baku (galat saling bebas dengan sebaran yang sama di sekitar garis sebenarnya), galat baku kemiringan adalah

$$\mathrm{SE}(b) = \frac{s_e}{\sqrt{S_{xx}}}, \tag{8.6}$$

dan kemiringan sebenarnya berada dalam jarak sekitar dua galat baku dari yang dicocokkan dengan keyakinan 95% (untuk 38 derajat kebebasan, faktor tepatnya dari distribusi t Student adalah 2.02) [3]. Untuk Mas Dimas, $\mathrm{SE}(b) = 0.101/\sqrt{36092.6} = 0.000531$ L/km, dan kemiringannya 0.020737 ± 0.001062 L/km. Persamaan (8.6) juga mengatakan cara mendapatkan kemiringan yang lebih tajam: lebih banyak hari (s_e yang lebih kecil tidak di tangan Anda, S_{xx} yang lebih besar ada di tangan Anda), dan terutama hari-hari yang tersebar, sebagian pendek dan sebagian panjang, karena S_{xx} tumbuh bersama sebaran x. Empat puluh hari yang semuanya 110 km hampir tidak mengatakan apa pun tentang biaya per kilometer.

8.3.5 Meramalkan satu hari baru

Ketidakpastian kemiringan berbicara tentang *kecenderungan*. Pertanyaan tentang *satu* hari baru menambahkan serakan hari itu sendiri. Untuk jarak baru x_0, ramalan kuadrat terkecilnya $\hat{y}_0 = a + bx_0$, dan *selang prediksi* 95% untuk bensin hari tunggal itu adalah [3]

$$\hat{y}_0 \pm t\, s_e \sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{S_{xx}}}, \tag{8.7}$$

dengan $t = 2.02$ untuk 38 derajat kebebasan. Ketiga suku di bawah tanda akar adalah serakan satu hari (angka 1), ketidakpastian ketinggian garis ($1/n$), dan ketidakpastian kemiringannya, yang tumbuh bersama jarak x_0 dari rata-rata. Untuk $x_0 = 150$ km, $1 + 1/40 + (150 - 111.75)^2/36092.6 = 1.0655$, akarnya 1.032, dan selangnya $3.476 \pm 2.02 \times 0.1009 \times 1.032 = 3.476 \pm 0.211$ L. Hampir seluruhnya berasal dari serakan hari itu sendiri; garisnya sendiri sudah diketahui dengan baik. Jauh di luar data, suku terakhir akan mengambil alih, dan itulah cara rumus ini memperingatkan terhadap ekstrapolasi.

8.3.6 Memperhatikan residu

Bilangan-bilangan di Subbab 8.3.3 merangkum pencocokan; residunya sendirilah yang memberi tahu apakah pencocokan itu layak dipercaya. Gambarkan residu terhadap x (atau terhadap nilai hasil pencocokan). Bila modelnya benar, residu membentuk pita tanpa bentuk di sekitar nol dengan lebar yang kira-kira tetap. Tiga pola merupakan tanda bahaya [4], [3]. *Lengkungan* pada residu (positif di kedua ujung, negatif di tengah, atau sebaliknya) berarti datanya melengkung dan garis lurus adalah model yang salah; Subbab 8.5 memperbaikinya. *Corong*, residu yang makin melebar ketika x membesar, berarti serakannya tidak sama di mana-mana, dan ketidakpastian Persamaan (8.6) lalu terlalu optimistis untuk x besar; pembobotan (Subbab 8.10) atau transformasi log membantu. *Satu titik yang jauh*, seperti hari ke-17 di Subbab 8.8, meminta kita menengok buku catatan. Untuk keempat puluh hari Mas Dimas, residu pada **Gambar 8.1** tidak memperlihatkan satu pun pola itu: ruas-ruas merah tegaknya pendek dan panjang, di atas dan di bawah garis, pada setiap jarak, dan begitulah rupa model garis lurus yang baik.

WAWASAN KUNCI: Bilangan hasil pencocokan tanpa ketidakpastiannya hanyalah separuh jawaban. Kemiringan 0.0207 ± 0.0011 L/km menyatakan "sekitar 0.021 L per km" sekaligus "jangan mendasarkan keputusan pada digit keempat".

8.4 Lebih Banyak Prediktor: Matriks Desain

8.4.1 Bensin dari jarak dan banyak perjalanan

Jarak bukan satu-satunya yang membakar bensin. Setiap perjalanan berarti berhenti, menunggu penumpang atau makanan, dan berangkat lagi. Buku catatan memuat banyaknya perjalanan, sehingga kita dapat mencoba

$$y \approx \beta_0 + \beta_1 x_1 + \beta_2 x_2,$$

dengan x_1 jarak dan x_2 banyak perjalanan. Tuliskan satu baris per hari, $[1,\ x_{i1},\ x_{i2}]$, dan tumpuk baris-barisnya menjadi **matriks desain** X (40 baris, 3 kolom); tumpuk bensinnya menjadi y dan koefisiennya menjadi β. Model untuk semua hari sekaligus adalah

$$y \approx X\beta, \tag{8.8}$$

dan kuadrat terkecil memilih β yang meminimumkan $\| y - X\beta \|^2$. Inilah persis masalah di Subbab 6.4 dengan nama-nama baru, dan semua alatnya berlaku: persamaan normal

$$X^{\mathsf{T}} X\, \beta = X^{\mathsf{T}} y, \tag{8.9}$$

yang diselesaikan dengan Cholesky, atau lebih baik, dengan QR Householder, yang tidak mengkuadratkan bilangan kondisi. Persamaan (8.2) adalah kasus khusus dengan kolom 1 dan x. Kata "linear" dalam *kuadrat terkecil linear* merujuk pada koefisien β, bukan pada kolom-kolomnya: kolom apa pun yang dapat dihitung dari data diperbolehkan, dan subbab berikut memanfaatkannya.

8.4.2 Mengapa QR, bahkan di sini

Matriks desain bensin memiliki satu kolom angka satu, satu kolom jarak di sekitar 110, dan satu kolom perjalanan di sekitar 15. Bilangan kondisinya sekitar 480, dan bilangan kondisi $X^{\mathsf{T}}X$ sekitar 230 000, kuadratnya, sebagaimana ditunjukkan Bab 6 pasti terjadi. Di sini persamaan normal pun masih mempertahankan sepuluh atau sebelas digit, lebih dari yang layak diterima data. Tetapi kebiasaannya lebih penting daripada kasus ini: tambahkan kolom kuadrat jarak (nilai di sekitar 12 000) dan bilangan kondisi $X^{\mathsf{T}}X$ melonjak beberapa orde besaran. QR sedikit lebih mahal dan tidak pernah mengkuadratkan apa pun, sehingga program kita selalu memakainya. Memusatkan dan menskalakan kolom-kolomnya membantu lebih jauh: bila setiap prediktor diganti dengan simpangannya dari rata-rata dibagi sebarannya, bilangan kondisi matriks desain ini turun dari 480 menjadi 2.2.

8.4.3 Arti koefisien-koefisiennya

Program menemukan liter $= 0.2654 + 0.018568\,\text{km} + 0.02285\,\text{perjalanan}$. Koefisien perjalanan menyatakan bahwa setiap perjalanan tambahan, *pada jarak yang sama*, memerlukan sekitar 0.023 L; koefisien jarak turun dari 0.0207 menjadi 0.0186 L/km, karena sebagian dari yang oleh garis sederhana dianggap akibat kilometer sebenarnya disebabkan oleh perjalanan-perjalanan

yang menyertainya. Kedua prediktor itu berkorelasi (dalam buku catatan ini korelasi jarak dan perjalanan 0.66: hari yang lebih panjang memiliki lebih banyak perjalanan), dan garis sederhana tidak dapat memisahkan pengaruh keduanya; pencocokan ganda dapat, asalkan korelasinya tidak terlalu dekat dengan 1. Bila dua kolom X hampir sebanding, matriks $X^{\mathsf{T}}X$ hampir singular, koefisien-koefisiennya menjadi sangat tidak pasti dan dapat berayun liar, dan masalahnya disebut *kolinear* [3]. R^2 naik dari 0.9757 menjadi 0.9894 dan galat baku turun dari 0.101 menjadi 0.068 L (kini dihitung dengan $n-3$ derajat kebebasan, karena tiga koefisien dicocokkan). Untuk besok, 120 km dan 17 perjalanan, model merencanakan 2.88 L.

8.5 Kurva yang Linear dalam Koefisiennya

8.5.1 Polinomial

Pertanyaan ketiga Mas Dimas menyangkut kecepatan. Pada kecepatan orang berjalan di tengah kemacetan, sepeda motor membakar bensin tanpa bergerak ke mana-mana; pada kecepatan tinggi, udara mendorong balik makin kuat. Karena itu efisiensi bensin (km/L) naik bersama kecepatan rata-rata, mencapai puncak, lalu turun: sebuah kurva, bukan garis. **Gambar 8.3** memperlihatkan 36 hari yang kecepatan rata-ratanya tercatat. Kurva paling sederhana yang berpuncak satu adalah parabola, dan model polinomial berderajat d,

$$y \approx \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d, \tag{8.10}$$

adalah *linear dalam koefisiennya*: matriks desainnya memiliki kolom $1, x, x^2, \ldots, x^d$. Jadi pencocokan polinomial adalah kuadrat terkecil linear, dan program di Subbab 8.4 mencocokkannya dengan matriks desain yang lain.

Ada satu jebakan, yang telah diukur Bab 6. Dengan kecepatan antara 15 dan 70 km/jam, kolom x^8 memiliki entri sampai $70^8 \approx 5.8 \times 10^{14}$, dan kolom-kolom pangkat hampir sejajar: bilangan kondisinya astronomis. Obatnya murah: **pusatkan dan skalakan** variabelnya lebih dahulu. Program kita memakai $t = (\text{kecepatan} - 45)/25$, yang kira-kira terletak di antara -1 dan 1, dan mencocokkan polinomial dalam t. Kurva hasilnya adalah fungsi kecepatan yang sama; hanya koefisiennya yang ditulis lain, dan perhitungannya tetap mempertahankan digitnya.

8.5.2 Kecepatan terbaik

Untuk pencocokan derajat 2 $y = c_0 + c_1 t + c_2 t^2$ dengan $c_2 < 0$, puncaknya berada di tempat turunan $c_1 + 2c_2 t$ bernilai nol:

$$t^* = -\frac{c_1}{2c_2}, \qquad \text{kecepatan}^* = 45 + 25\,t^*. \tag{8.11}$$

Program menemukan $c_0 = 54.653$, $c_1 = 0.535$, dan $c_2 = -8.459$, sehingga $t^* = 0.535/16.918 = 0.0316$ dan kecepatan rata-rata terbaiknya $45 + 25 \times 0.0316 = 45.8$ km/jam, tempat motor memberi sekitar 54.7 km/L. Bab 13 kembali membahas cara mencari puncak kurva ketika tidak ada rumusnya.

8.5.3 Eksponensial melalui logaritma

Baterai ponsel kehilangan kapasitas kira-kira sebanding dengan yang masih tersisa: model $H(t) = Ce^{kt}$, dengan H kesehatan dalam persen, t bulan pemakaian, dan $k < 0$. Model ini tidak linear dalam C dan k. Tetapi logaritmanya linear:

$$\ln H = \ln C + k\,t, \tag{8.12}$$

garis lurus dalam t dengan intersep $\ln C$ dan kemiringan k. Cocokkan garis itu pada titik-titik $(t_i, \ln H_i)$ dengan Persamaan (8.3) lalu kembalikan: $C = e^{\text{intersep}}$. Untuk ketiga belas bacaan Mas Dimas (setiap dua bulan selama dua tahun, **Gambar 8.4**) hasilnya $H = 100.03\,e^{-0.00954\,t}$, dan kesehatannya mencapai 80% ketika $100.03\,e^{-0.00954\,t} = 80$, yaitu setelah $t = \ln(80/100.03)/(-0.00954) = 23.4$ bulan. Trik yang sama mencocokkan hukum pangkat, $y = Cx^k$ menjadi $\ln y = \ln C + k \ln x$, dan banyak kurva pertumbuhan dan peluruhan.

Trik ini ada harganya: kuadrat terkecil pada $\ln H$ meminimumkan kuadrat meleset *relatif* alih-alih kuadrat meleset dalam persen, sehingga memberi bacaan awal yang lebih besar sedikit lebih kecil bobotnya daripada pencocokan langsung. Subbab 8.9 mencocokkan kurva itu secara langsung dengan `curve_fit` taklinear dari SciPy dan memperoleh $100.04\,e^{-0.00956\,t}$, praktis sama di sini, karena bacaannya dekat dengan kurva. Bila datanya berderau, keduanya dapat berbeda, dan pencocokan langsunglah yang menjawab pertanyaan sebagaimana diajukan.

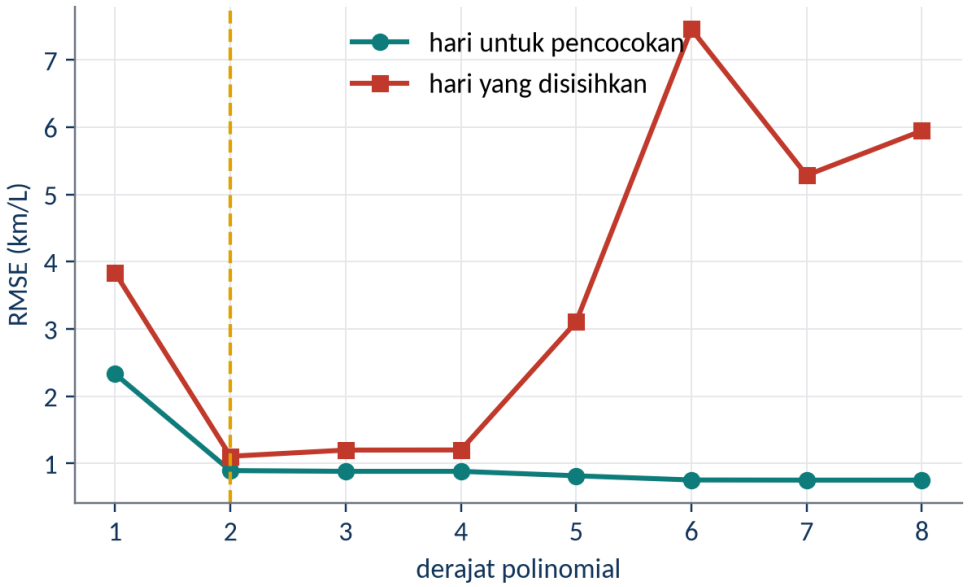
8.6 *Overfitting*, dan Cara Memilih Model

Polinomial berderajat lebih tinggi selalu mencocokkan data yang ada setidaknya sebaik yang lebih rendah: menambah kolom pada X hanya dapat menurunkan SSE, karena solusi lama masih tersedia (dengan koefisien baru nol). Karena itu R^2 naik pada setiap derajat, dan polinomial derajat 35 melalui 36 titik akan mencapai $R^2 = 1$. Itu tidak menjadikannya model yang lebih baik. Model itu telah mempelajari kebetulan-kebetulan dari 36 hari tersebut, dan pada hari ke-37 besar kemungkinan ia salah besar. Inilah **overfitting** (pencocokan berlebih).

Uji yang jujur adalah menilai model pada data yang belum pernah dilihatnya. Bagilah hari-hari menjadi dua kelompok: cocokkan pada yang satu (hari *pelatihan*) dan ukur galat pada yang lain (hari *yang disisihkan*). Ukurannya adalah **akar rata-rata kuadrat galat** (*root mean square error*)

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}, \tag{8.13}$$

meleset khas dalam satuan y, dengan $\hat{y}_i$ ramalan model untuk hari ke-i. Program polinomial menyisihkan setiap hari ketiga (hari 3, 6, 9, …, 36), mencocokkan derajat 1 sampai 8 pada 24 hari lainnya, dan mengukur RMSE pada kedua kelompok. **Gambar 8.2** memperlihatkan gambaran yang khas untuk setiap masalah pemilihan model: galat pada hari pencocokan terus turun bersama derajat; galat pada hari yang disisihkan turun, mencapai minimum, lalu naik. Model terbaik berada di dasar kurva kedua, bukan yang pertama.



Gambar 8.2 RMSE pencocokan polinomial derajat 1 sampai 8 pada 24 hari pencocokan dan pada 12 hari yang disisihkan. Galat pencocokan hanya turun; galat hari yang disisihkan paling kecil pada derajat 2 dan tumbuh tajam mulai derajat 5.

Mengapa galat hari yang disisihkan naik lagi? Model yang lentur memiliki dua sumber galat pada data baru. *Bias*-nya adalah bagian kebenaran yang tidak dapat diwakilinya: garis lurus tidak dapat melengkung, berapa pun hari yang dilihatnya, sehingga ia meleset dari puncak pada setiap hari baru. *Variansi*-nya adalah seberapa besar ramalannya akan berubah bila ia dicocokkan pada kumpulan hari yang lain: polinomial derajat 8 membentuk ulang dirinya di sekitar setiap kebetulan, sehingga hari baru bertemu kurva yang dilengkungkan untuk orang lain. Menaikkan derajat menurunkan bias dan menaikkan variansi, dan galat hari yang disisihkan paling kecil di tempat jumlah keduanya paling kecil [6]. Derajat 2 hampir tidak berbias untuk kurva berpuncak satu dan variansinya sangat kecil.

Satu pembagian bisa beruntung atau sial. *Validasi silang* (*cross-validation*) memutar kelompok yang disisihkan (sepertiga pertama, lalu kedua, lalu ketiga) dan merata-ratakan galat hari yang disisihkan; Soal 6 mengerjakannya untuk ketiga pertigaan buku catatan. Versi yang paling menyeluruh menyisihkan satu hari setiap kali: cocokkan pada 35 hari, ramalkan hari ke-36, dan ulangi untuk setiap hari (validasi silang *leave-one-out*). Untuk data kecepatan, cara ini memberi RMSE hari yang disisihkan 3.13 km/L untuk derajat 1, 1.017 untuk derajat 2, 1.021 untuk derajat 3, 1.049 untuk derajat 4, 1.33 untuk derajat 5, dan 3.07 untuk derajat 6: sekali lagi minimum pada derajat 2, dengan derajat 3 di urutan kedua yang dekat tetapi tidak sepadan dengan koefisien tambahannya. Dalam pembelajaran mesin, gagasan yang sama melindungi setiap model agar tidak memercayai data pelatihannya sendiri, seperti ditunjukkan Bab 14.

8.7 Membangunnya dalam Java dan Python

Tiga berkas per bahasa. `lsq` memuat alat bersama: membaca berkas CSV, kuadrat terkecil dengan QR Householder (fungsi dari Bab 6, kini bernama `lstsq`), ramalan, dan R^2. `line_fit` menjawab pertanyaan pertama, kedua, dan keempat Mas Dimas: garis lurus dengan Persamaan (8.3), R^2-nya, galat baku, dan ketidakpastian kemiringan, model dua prediktor dengan QR, dan kurva baterai melalui logaritma. `poly_fit` menjawab pertanyaan ketiga: polinomial derajat 1 sampai 8 yang dinilai pada hari yang disisihkan, dan kecepatan terbaik dari pencocokan derajat 2.

Kode 8.1 adalah modul pembantu Python. Fungsi `lstsq` adalah QR Householder dari Subbab 6.5, tidak berubah kecuali hanya mengembalikan koefisiennya.

Kode 8.1 `lsq.py`: membaca CSV, kuadrat terkecil dengan QR Householder, ramalan, dan R^2, dipakai bersama oleh program-program bab ini.

```
"""Pembantu bersama: jumlah biasa, baca CSV, dan kuadrat terkecil dengan
QR Householder (metode Bab 6)."""
import math


def total(values):
    """Jumlah biasa dari kiri ke kanan (sum() Python 3.12+ mengompensasi
    pembulatan, sehingga digit kita akan berbeda dari versi Java)."""
    s = 0.0
    for v in values:
        s += v
    return s


def read_csv(path):
    """Nama kolom judul dan baris-baris sebagai list float."""
    with open(path, encoding="utf-8") as f:
        names = f.readline().strip().split(",")
        rows = [[float(v) for v in line.strip().split(",")]
                for line in f if line.strip()]
    return names, rows


def lstsq(a, b):
    """Minimumkan ||b - A x|| dengan refleksi Householder; kembalikan x."""
    m, n = len(a), len(a[0])
    r = [row[:] for row in a]
    c = b[:]
    for k in range(n):
        norm = math.sqrt(total(r[i][k] * r[i][k] for i in range(k, m)))
        alpha = -norm if r[k][k] >= 0.0 else norm
        v = [0.0] * m
        for i in range(k, m):
            v[i] = r[i][k]
        v[k] -= alpha
        vv = total(v[i] * v[i] for i in range(k, m))
        if vv == 0.0:
            continue
        for j in range(k, n):
            f = 2.0 * total(v[i] * r[i][j] for i in range(k, m)) / vv
            for i in range(k, m):
                r[i][j] -= f * v[i]
        f = 2.0 * total(v[i] * c[i] for i in range(k, m)) / vv
        for i in range(k, m):
            c[i] -= f * v[i]
    x = [0.0] * n
```

Kode 8.1 (Lanjutan) *lsq.py: membaca CSV, kuadrat terkecil dengan QR Householder, ramalan, dan R^2, dipakai bersama oleh program-program bab ini.*

```
    for i in range(n - 1, -1, -1):
        s = total(r[i][j] * x[j] for j in range(i + 1, n))
        x[i] = (c[i] - s) / r[i][i]
    return x


def predict(a, x):
    return [total(row[j] * x[j] for j in range(len(x))) for row in a]


def r_squared(y, fitted):
    """1 - (jumlah kuadrat residu) / (jumlah kuadrat total)."""
    mean = total(y) / len(y)
    sse = total((u - v) * (u - v) for u, v in zip(y, fitted))
    sst = total((u - mean) * (u - mean) for u in y)
    return 1.0 - sse / sst
```

Kode 8.2 adalah kelas pembantu Java. Kelas ini juga memuat pemformatan bilangan yang membuat keluaran Java identik dengan keluaran Python, dan pembantu `signed` untuk bilangan yang dicetak dengan tanda plus yang eksplisit.

Kode 8.2 `Lsq.java`: pembantu bersama dalam Java.

```java
import java.io.IOException;
import java.math.BigDecimal;
import java.math.MathContext;
import java.math.RoundingMode;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;

/** Pembantu bersama: baca CSV, kuadrat terkecil dengan QR Householder
 *  (metode Bab 6), R^2, dan format angka yang sama dengan Python. */
public class Lsq {
    /** Baris-baris file CSV yang berbaris judul, sebagai angka. */
    static double[][] readCsv(String path) throws IOException {
        List<String> lines = Files.readAllLines(Path.of(path));
        double[][] rows = new double[lines.size() - 1][];
        for (int i = 1; i < lines.size(); i++) {
            String[] p = lines.get(i).strip().split(",");
            rows[i - 1] = new double[p.length];
            for (int j = 0; j < p.length; j++) {
                rows[i - 1][j] = Double.parseDouble(p[j]);
            }
        }
        return rows;
    }

    /** Minimumkan ||b - A x|| dengan refleksi Householder; kembalikan x. */
    static double[] lstsq(double[][] a, double[] b) {
        int m = a.length, n = a[0].length;
        double[][] r = new double[m][];
        for (int i = 0; i < m; i++) r[i] = a[i].clone();
        double[] c = b.clone();
        for (int k = 0; k < n; k++) {
            double ss = 0;
            for (int i = k; i < m; i++) ss += r[i][k] * r[i][k];
            double norm = Math.sqrt(ss);
            double alpha = r[k][k] >= 0.0 ? -norm : norm;
            double[] v = new double[m];
            for (int i = k; i < m; i++) v[i] = r[i][k];
            v[k] -= alpha;
            double vv = 0;
            for (int i = k; i < m; i++) vv += v[i] * v[i];
            if (vv == 0.0) continue;
            for (int j = k; j < n; j++) {
                double s = 0;
                for (int i = k; i < m; i++) s += v[i] * r[i][j];
                double f = 2.0 * s / vv;
```

Kode 8.2 (Lanjutan) *Lsq.java: pembantu bersama dalam Java.*

```
                for (int i = k; i < m; i++) r[i][j] -= f * v[i];
            }
            double s = 0;
            for (int i = k; i < m; i++) s += v[i] * c[i];
            double f = 2.0 * s / vv;
            for (int i = k; i < m; i++) c[i] -= f * v[i];
        }
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double s = 0;
            for (int j = i + 1; j < n; j++) s += r[i][j] * x[j];
            x[i] = (c[i] - s) / r[i][i];
        }
        return x;
    }

    static double[] predict(double[][] a, double[] x) {
        double[] out = new double[a.length];
        for (int i = 0; i < a.length; i++) {
            double s = 0;
            for (int j = 0; j < x.length; j++) s += a[i][j] * x[j];
            out[i] = s;
        }
        return out;
    }

    static double total(double[] v) {
        double s = 0;
        for (double x : v) s += x;
        return s;
    }

    /** 1 - (jumlah kuadrat residu) / (jumlah kuadrat total). */
    static double rSquared(double[] y, double[] fitted) {
        double mean = total(y) / y.length;
        double sse = 0, sst = 0;
        for (int i = 0; i < y.length; i++) {
            sse += (y[i] - fitted[i]) * (y[i] - fitted[i]);
        }
        for (double u : y) sst += (u - mean) * (u - mean);
        return 1.0 - sse / sst;
    }
```

Kode 8.2 (Lanjutan) *Lsq.java: pembantu bersama dalam Java.*

```
    static String signed(double x, int d) {
        String s = f(x, d);
        return s.startsWith("-") ? s : "+" + s;
    }

    // ---- format angka yang sama persis dengan f"{x:w.df}" dan ".de" Python
    static String f(double x, int d) {
        String s = new BigDecimal(x).setScale(d, RoundingMode.HALF_EVEN)
                                    .toPlainString();
        boolean neg = x < 0 || (x == 0 && 1 / x < 0);
        return neg && !s.startsWith("-") ? "-" + s : s;
    }

    static String f(double x, int d, int w) { return pad(f(x, d), w); }

    static String e(double x, int d) {
        if (x == 0) return "0." + "0".repeat(d) + "e+00";
        BigDecimal b = new BigDecimal(x).round(
                new MathContext(d + 1, RoundingMode.HALF_EVEN));
        int exp = b.precision() - b.scale() - 1;
        String digits = b.unscaledValue().abs().toString();
        digits = (digits + "0".repeat(d + 1)).substring(0, d + 1);
        String ex = (exp < 0 ? "-" : "+") + (Math.abs(exp) < 10 ? "0" : "")
                    + Math.abs(exp);
        return (x < 0 ? "-" : "") + digits.charAt(0) + "." + digits.substring(1)
               + "e" + ex;
    }

    static String pad(String s, int w) {
        return " ".repeat(Math.max(0, w - s.length())) + s;
    }

    static String padRight(String s, int w) {
        return s + " ".repeat(Math.max(0, w - s.length()));
    }
}
```

Kode 8.3 menghitung garis lurus dari jumlah-jumlah yang dipusatkan, persis seperti pada **Tabel 8.1**, lalu membangun matriks desain tiga kolom dan memanggil `lstsq`, lalu mencocokkan garis baterai pada logaritma.

Kode 8.3 `line_fit.py`: garis kuadrat terkecil dengan R^2, galat baku, dan ketidakpastian kemiringan; dua prediktor dengan QR; kurva baterai eksponensial.

```
"""Buku catatan bensin Mas Dimas: garis lurus dengan rumus kuadrat
terkecil, lalu dua prediktor dengan QR, lalu kurva baterai lewat logaritma."""
import math

from lsq import lstsq, predict, r_squared, read_csv, total


def main():
    _, rows = read_csv("../data/fuel_log.csv")
    km = [r[1] for r in rows]
    trips = [r[2] for r in rows]
    liters = [r[3] for r in rows]
    n = len(rows)
    mx, my = total(km) / n, total(liters) / n
    sxx = total((x - mx) * (x - mx) for x in km)
    sxy = total((x - mx) * (y - my) for x, y in zip(km, liters))
    b = sxy / sxx
    a = my - b * mx
    fitted = [a + b * x for x in km]
    res = [y - f for y, f in zip(liters, fitted)]
    sse = total(e * e for e in res)
    se = math.sqrt(sse / (n - 2))
    print(f"{n} days, mean {mx:.2f} km and {my:.4f} L per day")
    print(f"Sxx = {sxx:.2f}, Sxy = {sxy:.4f}")
    print(f"liters = {a:.4f} + {b:.6f} * km")
    print(f"R^2 = {r_squared(liters, fitted):.4f},"
          f" standard error {se:.4f} L")
    print(f"slope {b:.6f} +/- {2 * se / math.sqrt(sxx):.6f} L/km (about 95%)")
    worst = max(range(n), key=lambda i: abs(res[i]))
    print(f"largest residual: day {rows[worst][0]:.0f},"
          f" {res[worst]:+.3f} L")
    print()
    design = [[1.0, x, t] for x, t in zip(km, trips)]
    coef = lstsq(design, liters)
    fit2 = predict(design, coef)
    se2 = math.sqrt(total((y - f) * (y - f) for y, f in zip(liters, fit2))
                    / (n - 3))
    print(f"liters = {coef[0]:.4f} + {coef[1]:.6f} * km"
          f" + {coef[2]:.5f} * trips")
    print(f"R^2 = {r_squared(liters, fit2):.4f}, standard error {se2:.4f} L")
    plan = coef[0] + coef[1] * 120.0 + coef[2] * 17.0
    print(f"tomorrow, 120 km and 17 trips: {plan:.2f} L")
    print()
    _, rows = read_csv("../data/battery.csv")
    months = [r[0] for r in rows]
    logs = [math.log(r[1]) for r in rows]
```

Kode 8.3 (Lanjutan) *line_fit.py: garis kuadrat terkecil dengan R^2, galat baku, dan ketidakpastian kemiringan; dua prediktor dengan QR; kurva baterai eksponensial.*

```
    mt, ml = total(months) / len(rows), total(logs) / len(rows)
    k = (total((t - mt) * (g - ml) for t, g in zip(months, logs))
         / total((t - mt) * (t - mt) for t in months))
    c = math.exp(ml - k * mt)
    print(f"battery health = {c:.2f} * exp({k:.5f} * month)")
    print(f"80% is reached after about {math.log(80.0 / c) / k:.1f} months")


if __name__ == "__main__":
    main()
```

Kode 8.4 `LineFit.java`: pencocokan yang sama dalam Java.

```java
import java.io.IOException;

/** Buku catatan bensin Mas Dimas: garis lurus dengan rumus kuadrat
 *  terkecil, lalu dua prediktor dengan QR, lalu kurva baterai lewat logaritma. */
public class LineFit {
    public static void main(String[] args) throws IOException {
        double[][] rows = Lsq.readCsv("../data/fuel_log.csv");
        int n = rows.length;
        double[] km = new double[n], trips = new double[n], liters = new double[n];
        for (int i = 0; i < n; i++) {
            km[i] = rows[i][1];
            trips[i] = rows[i][2];
            liters[i] = rows[i][3];
        }
        double mx = Lsq.total(km) / n, my = Lsq.total(liters) / n;
        double sxx = 0, sxy = 0;
        for (int i = 0; i < n; i++) sxx += (km[i] - mx) * (km[i] - mx);
        for (int i = 0; i < n; i++) sxy += (km[i] - mx) * (liters[i] - my);
        double b = sxy / sxx;
        double a = my - b * mx;
        double[] fitted = new double[n], res = new double[n];
        double sse = 0;
        for (int i = 0; i < n; i++) {
            fitted[i] = a + b * km[i];
            res[i] = liters[i] - fitted[i];
        }
        for (double e : res) sse += e * e;
        double se = Math.sqrt(sse / (n - 2));
        System.out.println(n + " days, mean " + Lsq.f(mx, 2) + " km and "
                           + Lsq.f(my, 4) + " L per day");
        System.out.println("Sxx = " + Lsq.f(sxx, 2) + ", Sxy = " + Lsq.f(sxy, 4));
        System.out.println("liters = " + Lsq.f(a, 4) + " + " + Lsq.f(b, 6)
                           + " * km");
        System.out.println("R^2 = " + Lsq.f(Lsq.rSquared(liters, fitted), 4)
                           + ", standard error " + Lsq.f(se, 4) + " L");
        System.out.println("slope " + Lsq.f(b, 6) + " +/- "
                           + Lsq.f(2 * se / Math.sqrt(sxx), 6)
                           + " L/km (about 95%)");
        int worst = 0;
        for (int i = 1; i < n; i++) {
            if (Math.abs(res[i]) > Math.abs(res[worst])) worst = i;
        }
        System.out.println("largest residual: day " + Lsq.f(rows[worst][0], 0)
                           + ", " + Lsq.signed(res[worst], 3) + " L");
        System.out.println();
        double[][] design = new double[n][];
```

Kode 8.4 (Lanjutan) *LineFit.java: pencocokan yang sama dalam Java.*

```
        for (int i = 0; i < n; i++) design[i] = new double[] {1.0, km[i], trips[i]};
        double[] coef = Lsq.lstsq(design, liters);
        double[] fit2 = Lsq.predict(design, coef);
        double sse2 = 0;
        for (int i = 0; i < n; i++) {
            sse2 += (liters[i] - fit2[i]) * (liters[i] - fit2[i]);
        }
        double se2 = Math.sqrt(sse2 / (n - 3));
        System.out.println("liters = " + Lsq.f(coef[0], 4) + " + "
                           + Lsq.f(coef[1], 6) + " * km + " + Lsq.f(coef[2], 5)
                           + " * trips");
        System.out.println("R^2 = " + Lsq.f(Lsq.rSquared(liters, fit2), 4)
                           + ", standard error " + Lsq.f(se2, 4) + " L");
        double plan = coef[0] + coef[1] * 120.0 + coef[2] * 17.0;
        System.out.println("tomorrow, 120 km and 17 trips: " + Lsq.f(plan, 2)
                           + " L");
        System.out.println();
        rows = Lsq.readCsv("../data/battery.csv");
        int m = rows.length;
        double[] months = new double[m], logs = new double[m];
        for (int i = 0; i < m; i++) {
            months[i] = rows[i][0];
            logs[i] = Math.log(rows[i][1]);
        }
        double mt = Lsq.total(months) / m, ml = Lsq.total(logs) / m;
        double num = 0, den = 0;
        for (int i = 0; i < m; i++) num += (months[i] - mt) * (logs[i] - ml);
        for (int i = 0; i < m; i++) den += (months[i] - mt) * (months[i] - mt);
        double k = num / den;
        double c = Math.exp(ml - k * mt);
        System.out.println("battery health = " + Lsq.f(c, 2) + " * exp("
                           + Lsq.f(k, 5) + " * month)");
        System.out.println("80% is reached after about "
                           + Lsq.f(Math.log(80.0 / c) / k, 1) + " months");
    }
}
```

Kode 8.4 adalah versi Java pencocokan garis. **Kode 8.5** membangun pangkat-pangkat kecepatan yang telah dipusatkan dan diskalakan dengan `powers`, membagi hari menurut nomornya, lalu berputar pada derajat-derajatnya.

Kode 8.5 `poly_fit.py`: pencocokan polinomial derajat 1 sampai 8 yang dinilai pada hari yang disisihkan, dan kecepatan jelajah terbaik dari parabola.

```
"""Kecepatan dan efisiensi bensin: kuadrat terkecil polinomial derajat 1-8,
dinilai pada hari yang tak dilihat pencocokan, dan kecepatan terbaik."""
import math

from lsq import lstsq, predict, r_squared, read_csv, total

CENTER, SCALE = 45.0, 25.0      # x = (speed - 45) / 25 kira-kira di [-1, 1]


def powers(speed, degree):
    x = (speed - CENTER) / SCALE
    row = [1.0]
    for _ in range(degree):
        row.append(row[-1] * x)
    return row


def rmse(y, fitted):
    return math.sqrt(total((u - v) * (u - v) for u, v in zip(y, fitted))
                     / len(y))


def main():
    _, rows = read_csv("../data/speed_economy.csv")
    train = [r for r in rows if int(r[0]) % 3 != 0]
    test = [r for r in rows if int(r[0]) % 3 == 0]
    print(f"{len(rows)} days: {len(train)} to fit, {len(test)} held back")
    print("degree  R^2 (fit)  RMSE fit  RMSE held back")
    for d in range(1, 9):
        a = [powers(r[1], d) for r in train]
        y = [r[2] for r in train]
        c = lstsq(a, y)
        fit = predict(a, c)
        at = [powers(r[1], d) for r in test]
        yt = [r[2] for r in test]
        print(f"{d:6d}{r_squared(y, fit):11.4f}{rmse(y, fit):10.3f}"
              f"{rmse(yt, predict(at, c)):15.3f}")
    print()
    a = [powers(r[1], 2) for r in rows]
    y = [r[2] for r in rows]
    c = lstsq(a, y)
    print(f"all days, degree 2: km/L = {c[0]:.3f} {c[1]:+.3f} x"
          f" {c[2]:+.3f} x^2")
    print("  where x = (speed - 45) / 25")
    xbest = -c[1] / (2.0 * c[2])
    best = c[0] + c[1] * xbest + c[2] * xbest * xbest
```

Kode 8.5 (Lanjutan) *poly_fit.py: pencocokan polinomial derajat 1 sampai 8 yang dinilai pada hari yang disisihkan, dan kecepatan jelajah terbaik dari parabola.*

```
    print(f"best average speed {CENTER + SCALE * xbest:.1f} km/h,"
          f" about {best:.1f} km/L")
    print(f"R^2 = {r_squared(y, predict(a, c)):.4f}")


if __name__ == "__main__":
    main()
```

Kode 8.6 `PolyFit.java`: pencocokan polinomial dalam Java.

```java
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/** Kecepatan dan efisiensi bensin: kuadrat terkecil polinomial derajat 1-8,
 *  dinilai pada hari yang tak dilihat pencocokan, dan kecepatan terbaik. */
public class PolyFit {
    static final double CENTER = 45.0, SCALE = 25.0;  // x kira-kira di [-1, 1]

    static double[] powers(double speed, int degree) {
        double x = (speed - CENTER) / SCALE;
        double[] row = new double[degree + 1];
        row[0] = 1.0;
        for (int j = 1; j <= degree; j++) row[j] = row[j - 1] * x;
        return row;
    }

    static double rmse(double[] y, double[] fitted) {
        double s = 0;
        for (int i = 0; i < y.length; i++) {
            s += (y[i] - fitted[i]) * (y[i] - fitted[i]);
        }
        return Math.sqrt(s / y.length);
    }

    static double[][] design(List<double[]> rows, int d) {
        double[][] a = new double[rows.size()][];
        for (int i = 0; i < a.length; i++) a[i] = powers(rows.get(i)[1], d);
        return a;
    }

    static double[] target(List<double[]> rows) {
        double[] y = new double[rows.size()];
        for (int i = 0; i < y.length; i++) y[i] = rows.get(i)[2];
        return y;
    }
```

Kode 8.6 (Lanjutan) *PolyFit.java: pencocokan polinomial dalam Java.*

```
    public static void main(String[] args) throws IOException {
        List<double[]> rows = List.of(Lsq.readCsv("../data/speed_economy.csv"));
        List<double[]> train = new ArrayList<>(), test = new ArrayList<>();
        for (double[] r : rows) ((int) r[0] % 3 != 0 ? train : test).add(r);
        System.out.println(rows.size() + " days: " + train.size() + " to fit, "
                           + test.size() + " held back");
        System.out.println("degree  R^2 (fit)  RMSE fit  RMSE held back");
        for (int d = 1; d <= 8; d++) {
            double[][] a = design(train, d);
            double[] y = target(train);
            double[] c = Lsq.lstsq(a, y);
            double[] fit = Lsq.predict(a, c);
            double[] yt = target(test);
            System.out.println(Lsq.pad("" + d, 6)
                    + Lsq.f(Lsq.rSquared(y, fit), 4, 11)
                    + Lsq.f(rmse(y, fit), 3, 10)
                    + Lsq.f(rmse(yt, Lsq.predict(design(test, d), c)), 3, 15));
        }
        System.out.println();
        double[][] a = design(rows, 2);
        double[] y = target(rows);
        double[] c = Lsq.lstsq(a, y);
        System.out.println("all days, degree 2: km/L = " + Lsq.f(c[0], 3) + " "
                           + Lsq.signed(c[1], 3) + " x " + Lsq.signed(c[2], 3)
                           + " x^2");
        System.out.println("  where x = (speed - 45) / 25");
        double xbest = -c[1] / (2.0 * c[2]);
        double best = c[0] + c[1] * xbest + c[2] * xbest * xbest;
        System.out.println("best average speed " + Lsq.f(CENTER + SCALE * xbest, 1)
                           + " km/h, about " + Lsq.f(best, 1) + " km/L");
        System.out.println("R^2 = " + Lsq.f(Lsq.rSquared(y, Lsq.predict(a, c)), 4));
    }
}
```

Dalam **Kode 8.6**, pembagian menjadi hari pencocokan dan hari yang disisihkan hanya satu baris yang memilih di antara dua list; selebihnya mengikuti versi Python.

8.8 Menjalankannya

Kompilasikan sekali di folder `java` dengan `javac -d out *.java`, lalu jalankan `java -cp out LineFit` dan `java -cp out PolyFit`; di folder `python`, `python3 line_fit.py` dan `python3 poly_fit.py`. Java dan Python mencetak teks yang identik.

```
40 days, mean 111.75 km and 2.6825 L per day
Sxx = 36092.60, Sxy = 748.4575
liters = 0.3652 + 0.020737 * km
R^2 = 0.9757, standard error 0.1009 L
slope 0.020737 +/- 0.001062 L/km (about 95%)
largest residual: day 17, +0.316 L

liters = 0.2654 + 0.018568 * km + 0.02285 * trips
R^2 = 0.9894, standard error 0.0675 L
tomorrow, 120 km and 17 trips: 2.88 L

battery health = 100.03 * exp(-0.00954 * month)
80% is reached after about 23.4 months
```

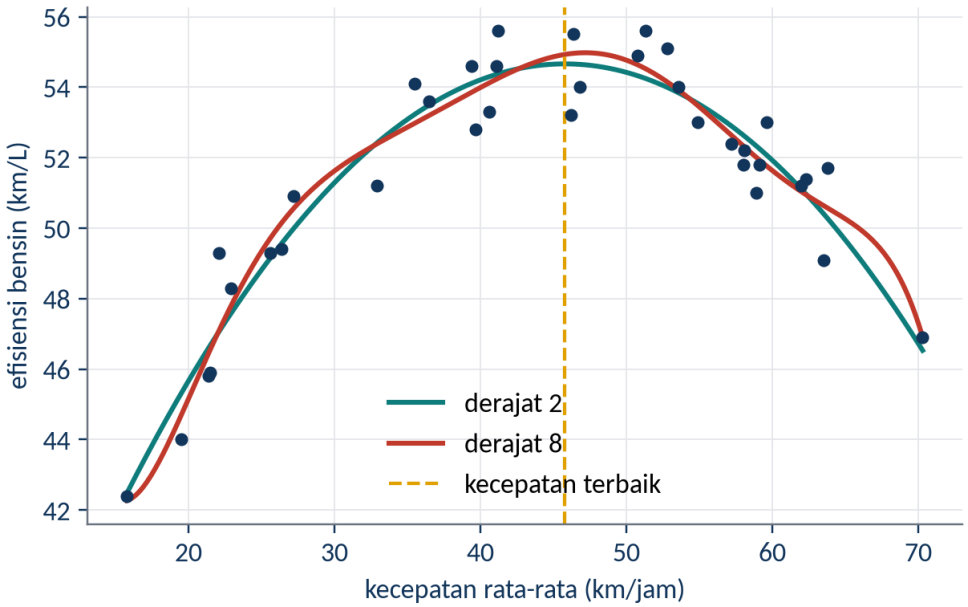
Blok pertama adalah garis lurus pada **Gambar 8.1**: 0.3652 L per hari ditambah 0.020737 L per kilometer, dengan $R^2 = 0.9757$, galat baku 0.1009 L, dan kemiringan yang diketahui sampai ± 0.0011 L/km. Residu terbesar ada pada hari ke-17, yang memakai 0.316 L lebih banyak daripada ramalan garis, tiga galat baku: layak ditengok di buku catatan (mungkin menunggu lama

dengan mesin menyala, atau mengisi tangki lebih penuh daripada biasanya). Satu hari yang tidak biasa di antara empat puluh itu wajar; banyak hari seperti itu akan mengisyaratkan ada sesuatu yang tidak diketahui model. Blok kedua menambahkan banyak perjalanan dan menjelaskan lebih banyak keragaman, seperti dibahas di Subbab 8.4.3, dan merencanakan 2.88 L untuk besok. Blok ketiga adalah baterai: 80% setelah sekitar 23.4 bulan, sedikit di bawah dua tahun.

Apa arti kemiringan itu bagi pertanyaan pertama Mas Dimas? Bensin per kilometer sekitar 0.019 sampai 0.021 L bergantung pada modelnya, sehingga perjalanan 10 km memerlukan sekitar 0.2 L bensin. Ia dapat mengalikannya dengan harga pompa yang ia bayar untuk dibandingkan dengan tarifnya. Perbandingan itu, dan apakah perjalanan itu diterima, adalah keputusannya; pencocokan hanya membuat biayanya terlihat.

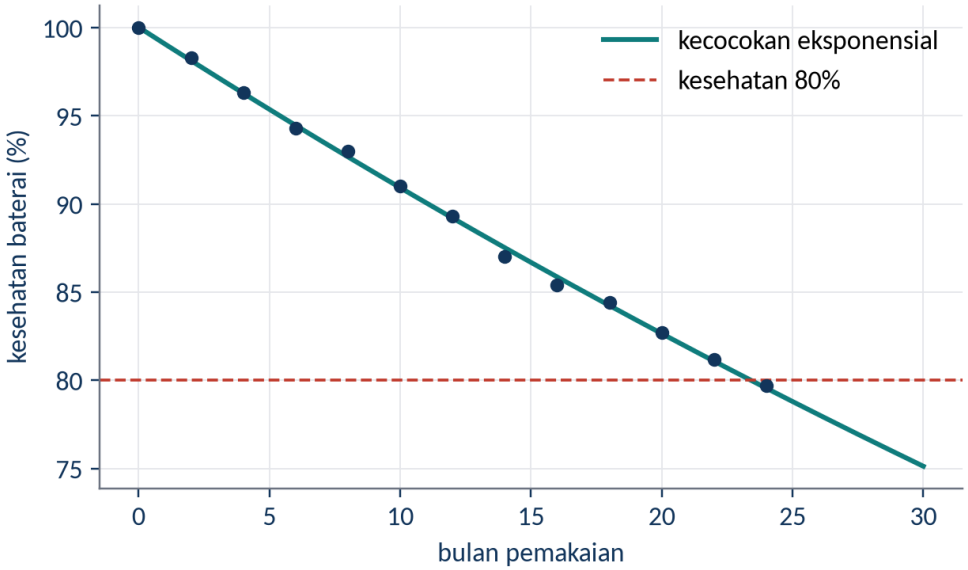
```
36 days: 24 to fit, 12 held back
degree  R^2 (fit)  RMSE fit  RMSE held back
     1     0.3998     2.330          3.834
     2     0.9120     0.892          1.106
     3     0.9141     0.881          1.198
     4     0.9141     0.881          1.199
     5     0.9265     0.815          3.103
     6     0.9373     0.753          7.460
     7     0.9377     0.751          5.286
     8     0.9377     0.751          5.948

all days, degree 2: km/L = 54.653 +0.535 x -8.459 x^2
  where x = (speed - 45) / 25
best average speed 45.8 km/h, about 54.7 km/L
R^2 = 0.9189
```

Tabel itu adalah **Gambar 8.2** dalam bentuk angka. Garis lurus (derajat 1) buruk pada kedua kelompok, RMSE 2.3 dan 3.8 km/L, karena datanya melengkung. Derajat 2 menurunkan galat hari yang disisihkan menjadi 1.106 km/L. Derajat 3 dan 4 mencocokkan hari pelatihan sedikit lebih baik (0.881 berbanding 0.892) dan hari yang disisihkan sedikit lebih buruk (1.198 dan 1.199); mulai derajat 5 galat hari yang disisihkan melonjak (3.1, lalu 7.5 pada derajat 6), sementara galat pencocokan terus membaik dan R^2 terus naik sampai 0.938. Laporan yang hanya melihat R^2 akan memilih derajat 8 dan menjadi yang terburuk untuk hari-hari baru. Pencocokan derajat 2 pada ke-36 hari menempatkan kecepatan rata-rata terbaik pada 45.8 km/jam dengan 54.7 km/L; **Gambar 8.3** memperlihatkannya di samping pencocokan derajat 8, yang melengkung mengikuti hari-hari tertentu, paling jelas di ujung kecepatan tinggi.



Gambar 8.3 Efisiensi bensin terhadap kecepatan rata-rata pada 36 hari, dengan pencocokan derajat 2 (puncaknya pada 45.8 km/jam ditandai) dan pencocokan derajat 8, yang mengikuti hari-hari tertentu alih-alih kecenderungannya.

Pencocokan baterai, yang digambar pada **Gambar 8.4**, hampir berupa garis lurus selama dua tahun: eksponensial dengan laju kecil tampak linear dalam waktu singkat. Garis lurus yang dicocokkan langsung pada bacaan kesehatan akan menyebut 80% setelah 23.1 bulan, eksponensial 23.4. Selama dua tahun perbedaannya kecil; selama lima tahun kedua model menyimpang, dan hanya bacaan tambahan yang dapat menentukan mana yang benar.



Gambar 8.4 Kesehatan baterai ponsel Mas Dimas setiap dua bulan, pencocokan eksponensial melalui logaritma, dan tingkat 80% yang dicapainya setelah sekitar 23.4 bulan.

8.9 Memeriksa dengan Pustaka

Ekosistem Python menyediakan kuadrat terkecil di setiap tingkat [7], [8]. `scipy.stats.linregress` mencocokkan garis dan melaporkan galat baku kemiringannya; `numpy.linalg.lstsq` menyelesaikan masalah kuadrat terkecil linear apa pun (dengan

dekomposisi nilai singular, anggota keluarga QR yang paling kokoh); `numpy.polynomial.Polynomial.fit` mencocokkan polinomial dan, seperti program kita, memetakan data ke $[-1,1]$ lebih dahulu; dan `scipy.optimize.curve_fit` mencocokkan model apa pun secara langsung dengan kuadrat terkecil taklinear [9]. **Kode 8.7** menjalankan keempatnya pada buku catatan.

Kode 8.7 `compare_numpy.py`: pencocokan diperiksa dengan `linregress`, `lstsq`, `Polynomial.fit`, dan `curve_fit` taklinear langsung untuk kurva baterai.

```
"""Periksa kecocokan kita dengan NumPy dan SciPy: linregress, lstsq,
Polynomial.fit, dan pencocokan taklinear kurva baterai dengan curve_fit."""
import numpy as np
import scipy
from numpy.polynomial import Polynomial
from scipy.optimize import curve_fit
from scipy.stats import linregress


def main():
    print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
    fuel = np.loadtxt("../data/fuel_log.csv", delimiter=",", skiprows=1)
    km, trips, liters = fuel[:, 1], fuel[:, 2], fuel[:, 3]
    r = linregress(km, liters)
    print(f"linregress: intercept {r.intercept:.4f}, slope {r.slope:.6f},"
          f" R^2 {r.rvalue ** 2:.4f}")
    print(f"  standard error of the slope {r.stderr:.6f}")
    design = np.column_stack([np.ones_like(km), km, trips])
    coef, *_ = np.linalg.lstsq(design, liters)
    print("lstsq with trips:", np.round(coef, 6))

    speed = np.loadtxt("../data/speed_economy.csv", delimiter=",", skiprows=1)
    p = Polynomial.fit(speed[:, 1], speed[:, 2], 2)
    c = p.convert().coef                        # kembali ke pangkat kecepatan
    best = -c[1] / (2 * c[2])
    print(f"Polynomial.fit degree 2: best speed {best:.1f} km/h,"
          f" {p(best):.1f} km/L")

    bat = np.loadtxt("../data/battery.csv", delimiter=",", skiprows=1)
    (c0, k), _ = curve_fit(lambda t, c, k: c * np.exp(k * t),
                           bat[:, 0], bat[:, 1], p0=(100.0, -0.01))
    print(f"curve_fit: health = {c0:.2f} * exp({k:.5f} * month),"
          f" 80% after {np.log(80.0 / c0) / k:.1f} months")


if __name__ == "__main__":
    main()
```

```
NumPy 2.5.3, SciPy 1.18.1
linregress: intercept 0.3652, slope 0.020737, R^2 0.9757
  standard error of the slope 0.000531
lstsq with trips: [0.265357 0.018568 0.022854]
Polynomial.fit degree 2: best speed 45.8 km/h, 54.7 km/L
curve_fit: health = 100.04 * exp(-0.00956 * month), 80% after 23.4 months
```

Garis dan R^2-nya cocok sampai semua digit yang dicetak, dan `linregress` melaporkan galat baku kemiringan 0.000531, separuh dari ± 0.001062 kita, yang merupakan dua galat baku. Ketiga koefisien dengan perjalanan cocok dengan QR kita sampai enam desimal. `Polynomial.fit` memberi kecepatan dan efisiensi terbaik yang sama. Pencocokan taklinear langsung untuk baterai memberi $100.04\,e^{-0.00956\,t}$ berbanding $100.03\,e^{-0.00954\,t}$ kita melalui logaritma, dengan 23.4 bulan yang sama: perbedaan pembobotan yang diuraikan Subbab 8.5.3 memang ada tetapi kecil untuk bacaan yang rapi seperti ini.

8.10 Membuatnya Lebih Baik

- **Kuadrat terkecil berbobot.** Bila sebagian titik lebih andal daripada yang lain (hari dengan pengisian penuh yang tepat dibandingkan hari dengan taksiran kasar), berikan setiap kuadrat residu bobot w_i, lebih besar untuk titik yang andal. Mengalikan baris ke-i dari X dan y_i dengan $\sqrt{w_i}$ mengubahnya menjadi masalah kuadrat terkecil biasa, seperti yang dilakukan Soal 4 Bab 6 untuk meleset relatif.
- **Pencocokan kokoh.** Kuadrat menghukum meleset besar dengan keras, sehingga satu entri yang salah dalam buku catatan (salah ketik 30.6 L untuk 3.06 L) dapat menyeret seluruh garis. Metode kokoh (*robust*) mengurangi bobot titik yang residunya besar, secara iteratif; pemeriksaan paling sederhana adalah mencocokkan lagi tanpa titik terburuk dan melihat apakah jawabannya banyak berubah.
- **Kuadrat terkecil taklinear.** Model yang tidak dapat dibuat linear, seperti $y = C(1 - e^{-kt})$ untuk kurva pengisian baterai, dicocokkan dengan mengiterasi kuadrat terkecil linear pada suatu linearisasi (metode Gauss-Newton dan Levenberg-Marquardt di balik `curve_fit`). Bab 13 membahas keluarga ini sebagai optimasi.
- **Galat pada kedua variabel.** Kuadrat terkecil mengukur meleset secara tegak, dan itu benar bila x eksak. Bila x diukur sekasar y, kemiringan hasil pencocokan berbias ke arah nol; *total least squares*, yang mengukur meleset tegak lurus terhadap garis dan dihitung dari dekomposisi nilai singular, memperlakukan kedua variabel secara setara [5].
- **Regularisasi.** Alih-alih memilih derajat rendah, pertahankan banyak kolom tetapi tambahkan penalti $\lambda \parallel \beta \parallel^2$ pada SSE (regresi *ridge*). Ini kembali masalah kuadrat terkecil, dengan $X^{\mathsf{T}}X + \lambda I$ menggantikan $X^{\mathsf{T}}X$, dan merupakan salah satu jembatan menuju pembelajaran mesin di Bab 14 [6].

8.11 Memakainya dengan Sungguh-Sungguh

Untuk mencocokkan buku catatan Anda sendiri, entah penjualan terhadap suhu harian untuk kedai es teh, pemakaian listrik terhadap jam menyala penyejuk udara, atau pemakaian air terhadap banyaknya tamu di sebuah penginapan kecil, masukkan kolom-kolomnya ke berkas CSV lalu ubah nama berkas dan kolom di `line_fit.py`. Kemudian ikuti rutinitas singkat berikut.

1. **Gambarkan lebih dahulu.** Diagram pencar langsung memperlihatkan apakah garis masuk akal, apakah ada lengkungan, dan apakah ada titik yang aneh. Tidak ada statistik yang menggantikan melihat. Simpan foto setiap halaman buku catatan bersama berkas datanya: bila sebuah titik tampak aneh, catatan aslinya dapat diperiksa dalam sekejap, dan kesalahan ketik dapat dibedakan dari hari yang memang tidak biasa.
2. **Cocokkan model paling sederhana yang mungkin berhasil**, lalu perhatikan residunya. Bila residu memperlihatkan pola (semuanya positif di tengah dan negatif di ujung), model itu melewatkan sebuah lengkungan.
3. **Sisihkan data.** Keluarkan seperempat atau sepertiga hari dari pencocokan dan periksa model pada hari-hari itu sebelum memercayainya.
4. **Laporkan ketidakpastiannya.** Kemiringan dengan galat bakunya, ramalan dengan galat baku taksirannya.
5. **Jangan berekstrapolasi jauh.** Garis Mas Dimas dicocokkan untuk 55 sampai 165 km sehari; untuk 300 km, di jalan tol, garis itu tidak mengatakan apa pun yang dapat diandalkan.

Dan satu peringatan tentang makna: pencocokan menunjukkan bahwa dua besaran bergerak bersama dalam data, bukan bahwa yang satu menyebabkan yang lain. Penjualan es teh naik

bersama suhu, dan penjualan payung pun naik di musim hujan; payung tidak menyebabkan es teh. Garis hasil pencocokan adalah ringkasan buku catatan, bukan hukum alam, dan tentu saja bukan nasihat keuangan: garis itu dapat memperlihatkan ke mana uang pergi, dan keputusan tetap berada pada orang yang mencarinya.

8.12 Metode Numerik dalam Praktik: Profitina

Dasbor analitik historis Profitina menyimpan skor visibilitas AI mingguan untuk setiap merek, dipindai ulang secara otomatis menurut jadwal, lengkap dengan peringatan dan ekspor CSV. Pemilik merek yang memandang dua belas titik mingguan yang naik turun ingin mengetahui satu hal: apakah situsnya membaik? Jawaban yang jujur adalah kemiringan garis kuadrat terkecil melalui titik-titik itu, *poin per minggu*, beserta galat bakunya dari Persamaan (8.6). Kemiringan $+0.8 \pm 0.3$ poin per minggu berarti "membaik, dan bukan karena kebetulan"; kemiringan $+0.8 \pm 1.5$ berarti "belum dapat dipastikan", seberapa pun menggembirakan dua minggu terakhir. Kecermatan yang sama yang diterapkan Subbab 8.6 pada derajat polinomial berlaku untuk garis kecenderungan di dasbor: kurva yang mengikuti setiap goyangan mingguan adalah *overfitting* dan tidak meramalkan apa pun.

Kuadrat terkecil juga berada di dalam lingkaran yang memperbaiki AI milik Profitina sendiri. Alur umpan baliknya yang transparan mengubah umpan balik pengguna menjadi *dataset* terkurasi tempat model Qwen2.5 yang berjalan secara lokal dilatih dan diukur sebelum dirilis, dan gerbang mutu Content Engine hanya menyajikan draf AI bila skornya lebih baik daripada templat bawaan (setelah revisi September 2026, draf AI menang pada 5 dari 6 kasus tolok ukur). Setiap pengukuran seperti itu adalah masalah statistik kecil sejenis yang diajarkan bab ini: rata-rata, sebaran, perbandingan yang harus besar dibandingkan deraunya sebelum ada yang memercayainya. Seperti biasa, angka-angka memberi saran dan manusia yang memutuskan.

Aritmetika kecenderungan seperti itu persis Persamaan (8.3) dengan x nomor minggu. Untuk dua belas minggu bernomor 1 sampai 12, $\bar{x} = 6.5$ dan $S_{xx} = 143$, sehingga menurut Persamaan (8.6) galat baku kemiringannya $s_e/\sqrt{143} \approx s_e/12$. Bila skor mingguan berserakan di sekitar kecenderungannya sekitar 3 poin, kemiringannya diketahui sampai sekitar ± 0.25 poin per minggu, dan sebuah kecenderungan harus lebih curam daripada setengah poin per minggu sebelum dua belas minggu dapat membedakannya dari derau. Dua puluh enam minggu akan memberi $S_{xx} = 1462.5$ dan galat baku lebih dari tiga kali lebih kecil: kesabaran membeli kepastian, dan hal itu layak dikatakan dengan terus terang kepada pemilik merek yang menginginkan vonis setelah tiga minggu.

8.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menilai model hanya dengan R^2.** R^2 tidak pernah turun ketika kolom ditambahkan. Bandingkan model pada data yang disisihkan (Persamaan (8.13), **Gambar 8.2**).
- **Mencocokkan polinomial berderajat tinggi dalam satuan mentah.** Pangkat 70 sampai kedelapan panjangnya 15 digit dan hampir sejajar. Pusatkan dan skalakan variabelnya, dan selesaikan dengan QR, bukan dengan persamaan normal.
- **Meminimumkan meleset tegak ketika galatnya ada pada x.** Kuadrat terkecil seperti yang dipakai di sini mengandaikan galat ada pada y; jarak dari aplikasi akurat, bensinnya tidak. Bila keduanya berderau, kemiringan hasil pencocokan berbias ke arah nol, dan metode seperti *total least squares* diperlukan.

- **Lupa apa yang dilakukan transformasi log.** Mencocokkan $\ln y$ meminimumkan galat relatif, bukan galat mutlak. Untuk jawaban yang tepat dalam satuan asli, periksalah dengan pencocokan taklinear langsung.
- **Mengekstrapolasi hasil pencocokan.** Setiap model hanya dapat dipercaya dalam rentang data yang membentuknya.
- **Membuang "pencilan" tanpa bertanya mengapa.** Hari ke-17 mungkin kesalahan dalam buku catatan, atau mungkin satu-satunya hari dengan penantian panjang di tengah hujan. Selidiki sebelum menghapus, dan jangan pernah menghapus titik hanya untuk memperbaiki R^2.
- **Membandingkan R^2 untuk sasaran yang berbeda.** R^2 yang dihitung pada $\ln H$ dan yang dihitung pada H mengukur hal yang berbeda dan tidak dapat dibandingkan. Bandingkan model pada besaran yang sama, dalam satuan yang sama, sebaiknya pada data yang disisihkan.
- **Membaca korelasi sebagai sebab.** Dua kolom yang bergerak bersama mungkin sama-sama mengikuti kolom ketiga.

8.14 Ringkasan Bab

- Kuadrat terkecil memilih kurva yang meminimumkan jumlah kuadrat residu (Persamaan (8.1)). Untuk garis lurus, persamaan normal (Persamaan (8.2)) memberi kemiringan S_{xy}/S_{xx} dan intersep yang membuat garis melewati titik rata-rata (Persamaan (8.3)); **Tabel 8.1** mengerjakannya dengan tangan untuk lima hari.
- Residu pencocokan yang memiliki intersep berjumlah nol dan tidak berkorelasi dengan prediktornya. R^2 (Persamaan (8.4)) adalah bagian keragaman yang dijelaskan, galat baku (Persamaan (8.5)) adalah meleset khas, dan galat baku kemiringan (Persamaan (8.6)) memberi ketidakpastian yang jujur: 0.0207 ± 0.0011 L/km untuk Mas Dimas (**Gambar 8.1**).
- Satu hari baru diramalkan dengan selang yang lebih lebar Persamaan (8.7), yang menambahkan serakan hari itu sendiri pada ketidakpastian garis: 3.48 ± 0.21 L untuk 150 km.
- Beberapa prediktor masuk melalui matriks desain (Persamaan (8.8)) dan persamaan normal $X^{\mathsf{T}}X\beta = X^{\mathsf{T}}y$ (Persamaan (8.9)), yang paling baik diselesaikan dengan QR Householder dari Bab 6. Koefisien perjalanan, 0.023 L per perjalanan, adalah pengaruhnya pada jarak yang tetap.
- Polinomial (Persamaan (8.10)) dan eksponensial melalui logaritma (Persamaan (8.12)) adalah masalah kuadrat terkecil linear. Pusatkan dan skalakan sebelum mencocokkan pangkat. Puncak parabola (Persamaan (8.11)) menempatkan kecepatan terbaik pada 45.8 km/jam (**Gambar 8.3**); baterai mencapai 80% setelah 23.4 bulan (**Gambar 8.4**).
- Model yang lebih lentur selalu mencocokkan data pelatihan lebih baik; hanya galat pada data yang disisihkan (Persamaan (8.13)) yang mengungkap *overfitting* (**Gambar 8.2**). Di sini derajat 2 yang menang.
- Program Java dan Python cocok bita demi bita, dan `linregress`, `lstsq`, `Polynomial.fit`, serta `curve_fit` membenarkan setiap hasil.

8.15 Soal Latihan

1. Dengan **Tabel 8.1**, periksalah bahwa residu garis lima hari berjumlah nol, lalu hitung R^2 lima hari itu. (Kelima nilai $y_i - \bar{y}$ memberi SST.)
2. Tunjukkan dari persamaan pertama Persamaan (8.2) bahwa garis kuadrat terkecil yang memiliki intersep melewati $(\bar{x}, \bar{y})$. Lalu jelaskan mengapa rumus $S_{xx} = \sum x_i^2 - n\bar{x}^2$ berisiko dalam titik mengambang dengan mencobanya, dalam kepala, pada jarak 100000.1, 100000.2, dan 100000.3 km yang disimpan dengan sekitar 16 digit bermakna.

3. Dengan garis empat puluh hari, ramalkan bensin untuk hari 150 km, dan berikan rentang kasar 95% untuk hari tunggal itu dengan $\pm 2s_e$. Lalu ramalkan dengan model dua prediktor untuk 150 km dan 20 perjalanan.
4. Pencocokan ganda menyatakan bahwa satu perjalanan tambahan pada jarak yang sama memerlukan 0.0229 L. Bila sebuah platform pengantaran menawarkan bonus untuk menyelesaikan 5 perjalanan pendek tambahan dalam jarak yang sama, berapa tambahan bensin yang diharapkan model? Mengapa Mas Dimas sebaiknya lebih sedikit memercayai angka ini daripada biaya per kilometer?
5. Cocokkan garis lurus (bukan eksponensial) pada bacaan baterai dan tentukan kapan garis itu mencapai 80%. Bandingkan dengan 23.4 bulan dan jelaskan mengapa kedua model sekarang sepakat tetapi akan berselisih lima tahun lagi.
6. Ulangi perbandingan derajat `poly_fit.py` dengan kelompok yang disisihkan dipilih sebagai hari yang nomornya bersisa 1 bila dibagi 3, lalu yang bersisa 2. Derajat mana yang RMSE hari yang disisihkannya terkecil dalam setiap kasus? Apa yang dianjurkan validasi silang, yang merata-ratakan ketiga pembagian?
7. Jelaskan mengapa, menurut Persamaan (8.6), buku catatan empat puluh hari dengan jarak yang tersebar antara 55 dan 165 km memberi biaya per kilometer yang lebih tepat daripada delapan puluh hari yang semuanya antara 100 dan 120 km. Taksirlah perbandingan kedua nilai S_{xx} dengan menganggap jaraknya tersebar merata.
8. Pakailah Persamaan (8.7) untuk memberi selang prediksi 95% bagi hari 60 km dan hari 250 km. Mengapa selang kedua hanya sedikit lebih lebar, dan mengapa Mas Dimas tetap harus jauh lebih sedikit memercayainya daripada yang pertama?

8.16 Penyelesaian Soal Latihan

1. Residunya $-0.096 + 0.034 - 0.017 + 0.089 - 0.010 = 0.000$ (tepat nol sampai pembulatan nilai yang dicetak). SST $= 0.812^2 + 0.222^2 + 1.008^2 + 0.342^2 + 0.368^2 = 0.6593 + 0.0493 + 1.0161 + 0.1170 + 0.1354 = 1.9771$. SSE $= 0.096^2 + 0.034^2 + 0.017^2 + 0.089^2 + 0.010^2 = 0.00922 + 0.00116 + 0.00029 + 0.00792 + 0.00010 = 0.01869$ (0.01862 dengan residu yang tidak dibulatkan). Jadi $R^2 = 1 - 0.01862/1.9771 = 0.9906$. Lima titik sangat cocok dengan garis; dengan empat puluh, pencocokannya tetap sangat baik, $R^2 = 0.9757$, tetapi lebih jujur.

2. Persamaan pertama Persamaan (8.2) adalah $na + b\sum x_i = \sum y_i$; membaginya dengan n memberi $a + b\bar{x} = \bar{y}$, yang menyatakan bahwa titik $(\bar{x}, \bar{y})$ terletak pada garis. Untuk jarak-jarak itu: $\bar{x} = 100000.2$, kuadratnya sekitar 10^{10}, dan $\sum x_i^2 - n\bar{x}^2$ mengurangkan dua bilangan sekitar 3×10^{10} yang sama pada 12 digit pertamanya, untuk memperoleh $S_{xx} = 0.02$ yang sebenarnya. Dengan 16 digit bermakna, hanya sekitar 4 yang tersisa untuk hasilnya, dan dengan bilangan yang lebih besar atau lebih banyak mungkin tidak ada yang tersisa sama sekali. Memusatkan lebih dahulu menghitung $(-0.1)^2 + 0^2 + 0.1^2 = 0.02$ dari bilangan kecil tanpa pembatalan.

3. $0.3652 + 0.020737 \times 150 = 3.476$ L. Satu hari berserakan di sekitar garis dengan ukuran khas $s_e = 0.101$ L, sehingga rentang kasar 95% adalah 3.48 ± 0.20 L, dari sekitar 3.28 sampai 3.68 L; selang prediksi lengkap Persamaan (8.7), yang juga memuat ketidakpastian kecil garis itu sendiri, memberi ± 0.21 L. Dengan perjalanan: $0.2654 + 0.018568 \times 150 + 0.02285 \times 20 = 0.2654 + 2.7852 + 0.4570 = 3.508$ L.

4. $5 \times 0.0229 = 0.114$ L, sekitar sepersepuluh liter. Angka ini kurang andal daripada biaya per kilometer karena dua alasan: banyak perjalanan berkorelasi kuat dengan jarak dalam buku catatan, sehingga pengaruh terpisahnya ditaksir dengan ketidakpastian lebih besar; dan

perjalanan pendek tambahan mungkin berbeda dari perjalanan rata-rata dalam buku catatan (lebih lama menunggu, atau lebih singkat). Koefisien itu menggambarkan hari-hari dalam buku catatan, bukan cara kerja yang baru.

5. Garis kuadrat terkecil melalui ketiga belas bacaan adalah $H = 99.67 - 0.8533\,t$, yang mencapai 80% pada $t = (99.67 - 80)/0.8533 = 23.1$ bulan, berbanding 23.4 untuk eksponensial. Selama dua tahun $e^{-0.00954t} \approx 1 - 0.00954t$ hampir linear, sehingga kedua model hampir berimpit pada data. Bila diekstrapolasi ke 60 bulan, garis menyebut $99.67 - 51.2 = 48.5\%$ dan eksponensial $100.03\,e^{-0.572} = 56.4\%$. Data tidak dapat membedakan keduanya; hanya bacaan pada tahun ketiga sampai kelima yang dapat, dan itulah aturan larangan ekstrapolasi dalam praktik.

6. Ubah kedua penyaring menjadi `int(r[0]) % 3 != 1` dan `== 1` (lalu 2). RMSE hari yang disisihkan untuk derajat 1 sampai 8: dengan sisa 1, 2.948, 0.795, 0.773, 0.849, 1.324, 2.576, 3.847, 5.340, terkecil pada derajat 3 dengan derajat 2 sangat dekat; dengan sisa 2, 2.106, 1.146, 1.177, 1.221, 1.216, 1.375, 1.335, 1.390, terkecil pada derajat 2. Merata-ratakan ketiga pembagian (termasuk pembagian di bab ini, 3.834, 1.106, 1.198, …), derajat 2 memiliki rata-rata galat hari yang disisihkan terkecil (sekitar 1.02 berbanding 1.05 untuk derajat 3). Validasi silang menganjurkan derajat 2, dan model yang lebih sederhana itu juga didukung cerita fisisnya (satu puncak).

7. S_{xx} tumbuh bersama banyaknya hari dan kuadrat sebarannya. Untuk n nilai yang tersebar merata pada rentang selebar w, $S_{xx} \approx nw^2/12$. Empat puluh hari pada 110 km: $40 \times 110^2/12 \approx 40{,}300$; delapan puluh hari pada 20 km: $80 \times 20^2/12 \approx 2{,}670$. Perbandingannya sekitar 15, sehingga menurut Persamaan (8.6) galat baku kemiringannya $\sqrt{15} \approx 3.9$ kali lebih kecil dengan buku catatan yang tersebar, meskipun jumlah harinya separuh. Untuk mengetahui bagaimana sesuatu berubah terhadap x, variasikan x.

8. Untuk 60 km: $\hat{y}_0 = 0.3652 + 0.020737 \times 60 = 1.609$ L, dan akarnya $\sqrt{1 + 0.025 + (60 - 111.75)^2/36092.6} = \sqrt{1.0992} = 1.048$, sehingga selangnya $1.609 \pm 2.02 \times 0.1009 \times 1.048 = 1.609 \pm 0.214$ L. Untuk 250 km: $\hat{y}_0 = 5.549$ L dan akarnya $\sqrt{1.5546} = 1.247$, sehingga 5.549 ± 0.255 L. Selang kedua hanya sekitar 20% lebih lebar, karena rumusnya hanya mengukur serakan acak *dengan anggapan bahwa garis lurus masih merupakan model yang benar* pada 250 km. Tidak ada apa pun dalam buku catatan yang mendukung anggapan itu: hari 250 km akan mencakup bentangan jalan raya panjang dengan kecepatan yang tidak pernah dilihat buku catatan, dan Subbab 8.5 menunjukkan bahwa efisiensi bensin berubah bersama kecepatan. Selang itu jujur tentang derau dan diam tentang model, dan itulah sebabnya ekstrapolasi memerlukan pertimbangan, bukan hanya rumus.

Daftar Pustaka

[1] A.-M. Legendre, *Nouvelles méthodes pour la détermination des orbites des comètes*. Paris, France: Firmin Didot, 1805.

[2] C. F. Gauss, *Theoria motus corporum coelestium in sectionibus conicis solem ambientium*. Hamburg, Germany: Perthes und Besser, 1809.

[3] D. C. Montgomery, E. A. Peck, and G. G. Vining, *Introduction to Linear Regression Analysis*, 6th ed. Hoboken, NJ, USA: Wiley, 2021.

[4] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[5] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[6] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. New York, NY, USA: Springer, 2009.

[7] NumPy Developers, "NumPy 2.5.0 Release Notes," Jun. 21, 2026, and "numpy 2.5.3" on the Python Package Index, Sep. 6, 2026. [Online]. Available: https://numpy.org/doc/2.5/release/2.5.0-notes.html and https://pypi.org/project/numpy/

[8] SciPy Developers, "SciPy 1.18.0 Release Notes," Jun. 19, 2026, and "scipy 1.18.1" on the Python Package Index, Aug. 21, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/release/1.18.0-notes.html and https://pypi.org/project/scipy/

[9] SciPy Developers, "scipy.optimize.curve_fit" and "scipy.stats.linregress," SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html

Bab 9
Diferensiasi Numerik

Ponsel dapat memberi tahu di mana Anda berada, tetapi tidak langsung memberi tahu seberapa cepat Anda melaju atau seberapa keras Anda baru saja mengerem. Keduanya adalah *laju perubahan*, turunan, dan komputer yang hanya memiliki bacaan setiap satu detik harus menaksirnya dari selisih antarbacaan. Bab ini menunjukkan cara melakukannya dengan baik. Di akhir bab Anda akan mampu menurunkan rumus beda maju, beda mundur, dan beda pusat beserta kerabatnya untuk turunan kedua dari teorema Taylor; menyatakan dan memeriksa orde galatnya; menjelaskan mengapa memperkecil langkah mula-mula membantu lalu justru merugikan, dan memilih langkah yang menyeimbangkan galat pemotongan dan galat pembulatan; memperbaiki taksiran dengan ekstrapolasi Richardson; membangun penganalisis perjalanan dalam Java dan Python yang mengubah catatan jarak seorang pengemudi ojek menjadi kecepatan dan percepatan serta menandai pengereman mendadak; memahami mengapa derau pengukuran adalah musuh sejati turunan numerik dan menghapus sebuah alarm palsu dengan rumus penghalusan kuadrat terkecil; serta memeriksa setiap jawaban terhadap NumPy dan SciPy.

9.1 Laporan Pengereman Mas Dimas

Mas Dimas mengemudikan ojek daring di Yogyakarta. Sebagian besar harinya berisi perjalanan pendek antara kampus, rumah kos, dan pasar di sekitar Jalan Kaliurang, dengan seorang mahasiswa atau sebuah paket di jok belakang. Suatu malam aplikasinya menampilkan pesan kecil berwarna jingga pada ringkasan perjalanan: *pengereman mendadak terdeteksi*. Itu bukan sanksi, tetapi pesan itu membuatnya cemas. Beberapa peringatan seperti itu, kata seorang kawan, dan skor keselamatan pengemudi turun, dan bersamanya peluang mendapat order yang lebih baik.

Ia ingat betul saat itu. Ia melaju sekitar empat puluh kilometer per jam di sebuah gang sepi ketika seekor kucing melompat keluar dari bawah mobil yang terparkir, dan ia mengerem keras. Itu tindakan yang benar, dan ia akan melakukannya lagi. Namun ia juga bertanya-tanya tentang perjalanan lain seminggu sebelumnya, ketika aplikasi mengeluh padahal ia yakin hanya meluncur pelan sampai berhenti di depan pagar rumah pelanggan. Apakah aplikasinya keliru? Bagaimana aplikasi itu tahu seberapa keras ia mengerem?

Ponsel hanya mengetahui satu hal dengan baik: *di mana* ia berada, dan karena itu seberapa jauh ia telah menempuh perjalanan sejak awal. Misalkan ponsel mencatat jarak itu setiap satu detik, seperti yang dilakukan aplikasi di ponsel Mas Dimas. Sedetik kemudian angkanya lebih besar; selisihnya adalah jarak yang ditempuh dalam detik itu, dan jarak per detik adalah kecepatan. Perubahan kecepatan dari satu detik ke detik berikutnya adalah percepatan, dan percepatan negatif yang besar adalah pengereman. Jadi kecepatan dan percepatan tersembunyi di dalam daftar jarak, tetapi tidak tertulis di sana. Keduanya harus *dihitung*, dengan mengurangkan bacaan-bacaan yang bertetangga lalu membaginya dengan selang waktu di antaranya.

Kedengarannya terlalu sederhana untuk sebuah bab. Ternyata tidak. Pengurangan yang sama yang menyingkap kecepatan juga memperbesar setiap galat kecil dalam bacaan, dan pengurangan kedua, untuk percepatan, memperbesarnya lagi. Galat beberapa puluh sentimeter pada jarak di ponsel, yang tidak akan disadari siapa pun di peta, menjadi galat lebih dari satu meter per detik

kuadrat pada percepatan, dan itulah beda antara “berhenti biasa” dan “pengereman mendadak”. Memahami dengan tepat seberapa besar galat itu, dari mana asalnya, dan apa yang dapat dilakukan terhadapnya adalah pokok bahasan diferensiasi numerik, dan kegunaannya jauh melampaui aplikasi ojek: debit air yang masuk ke tandon dari ketinggian airnya, laju habisnya baterai ponsel dari persentasenya, laju pertumbuhan usaha dari penjualan bulanannya, dan tren skor sebuah situs web dari pemindaian mingguannya semuanya adalah turunan yang ditaksir dari sampel.

Dalam bab ini kita mengambil catatan perjalanan Mas Dimas, empat puluh satu bacaan jarak, satu per detik, dan membangun penganalisis yang semestinya dimiliki aplikasi itu: penganalisis ini mencetak kecepatan dan percepatan pada setiap detik, menandai pengereman mendadak, dan, dalam bentuk yang disempurnakan, dapat membedakan rem mendadak yang sungguhan dari alarm palsu.

9.2 Gagasannya dalam Kata-Kata Sederhana

9.2.1 Kecepatan dari dua bacaan

Tabel 9.1 memperlihatkan lima bacaan dari perjalanan itu, di sekitar saat munculnya kucing.

Tabel 9.1 Lima bacaan jarak dari perjalanan Mas Dimas, satu per detik, di sekitar pengereman keras.

Waktu t (s)	16	17	18	19	20
jarak s (m)	148.2	159.5	168.7	174.4	176.5

Antara $t = 17$ dan $t = 18$ jarak bertambah $168.7 - 159.5 = 9.2$ m, jadi selama detik itu sepeda motor bergerak dengan kecepatan rata-rata 9.2 m/s. Antara $t = 18$ dan $t = 19$ motor bergerak $174.4 - 168.7 = 5.7$ m, rata-rata 5.7 m/s. Kecepatan *pada* $t = 18$ pasti berada di dekat keduanya, dan tebakan yang wajar adalah melihat satu detik ke belakang dan satu detik ke depan lalu mengambil kecepatan rata-rata selama dua detik itu:

$$v(18) \approx \frac{174.4 - 159.5}{19 - 17} = \frac{14.9}{2} = 7.45 \text{ m/s},$$

sekitar 27 km/jam. Pada grafik jarak terhadap waktu, ini adalah kemiringan garis lurus yang melalui titik pada $t = 17$ dan $t = 19$. Kecepatan sebenarnya pada $t = 18$ adalah kemiringan *garis singgung* kurva di titik itu; garis yang melalui dua tetangga, sebuah *garis potong* (sekan), memiliki kemiringan yang hampir sama bila kedua tetangga itu berdekatan. Itulah seluruh gagasan diferensiasi numerik: ganti garis singgung, yang memerlukan kurva yang persis, dengan garis potong, yang hanya memerlukan bacaan.

Ada tiga garis potong yang jelas untuk dipilih. Melihat ke depan saja, dari $t = 18$ ke $t = 19$, memberi 5,7 m/s; melihat ke belakang saja, dari $t = 17$ ke $t = 18$, memberi 9,2 m/s; melihat ke dua arah memberi 7,45 m/s. Ketiga jawaban itu berbeda jauh, karena kecepatannya sedang berubah cepat: motor sedang mengerem. Pandangan ke depan menggambarkan kecepatan setengah detik *sesudah* $t = 18$, pandangan ke belakang setengah detik *sebelumnya*, dan pandangan dua arah berpusat tepat pada $t = 18$. Subbab 9.3 membuktikan bahwa pilihan yang berpusat itu bukan hanya lebih adil, tetapi juga jauh lebih akurat.

9.2.2 Percepatan dari tiga bacaan

Percepatan adalah laju perubahan kecepatan. Kecepatan selama detik sebelum $t = 18$ adalah 9,2 m/s, dan selama detik sesudahnya 5,7 m/s; kedua rata-rata ini milik saat 17,5 dan 18,5, yang berjarak satu detik. Perubahannya $5.7 - 9.2 = -3.5$ m/s dalam satu detik, sehingga

$$a(18) \approx \frac{5.7 - 9.2}{1} = -3.5 \text{ m/s}^2.$$

Bila ditulis dengan bacaannya sendiri, ini adalah $(174.4 - 2 \times 168.7 + 159.5)/1^2 = -3.5$: *beda kedua* dari tiga bacaan bertetangga, dibagi kuadrat langkah waktunya. Perlambatan 3,5 m/s^2 berarti kehilangan kecepatan 12,6 km/jam setiap detik. Banyak aplikasi telematika menganggap pengereman yang lebih keras daripada sekitar 3 m/s^2 sebagai pengereman mendadak, dan penganalisis kita memakai ambang itu, -3.0 m/s^2. Pada $t = 18$ Mas Dimas jelas melewatinya, sesuai ingatannya.

Sekarang lihat lagi angka-angka itu dengan mata curiga. Jaraknya dicatat sampai sepersepuluh meter, dan posisi ponsel, bahkan setelah aplikasi menghaluskan GPS-nya, tidak pasti sebesar beberapa puluh sentimeter. Misalkan bacaan pada $t = 18$ kelebihan 0,3 m. Kecepatan pada $t = 18$ sama sekali tidak berubah (rumus berpusat tidak memakai bacaan tengah), tetapi percepatannya berubah sebesar $-2 \times 0.3 = -0.6$ m/s^2. Bila kedua tetangganya pada saat yang sama masing-masing kekurangan 0,3 m, galatnya mencapai -1.2 m/s^2. Galat sekecil itu pada jarak tidak terlihat; galat yang sama pada percepatan dapat mengubah berhenti pelan sebesar -1.5 m/s^2 menjadi "mendadak" -2.7 atau lebih buruk. Inilah misteri di depan pagar tadi, dan Subbab 9.6 akan menemukannya di dalam data.

> **WAWASAN KUNCI:** Diferensiasi dari data membagi selisih kecil antara bilangan-bilangan besar dengan langkah yang kecil. Setiap galat dalam bacaan dibagi dengan langkah itu, sekali untuk turunan pertama dan dua kali untuk turunan kedua. Integrasi, pokok bahasan Bab 10, menjumlahkan bacaan dan merata-ratakan galatnya sehingga saling meredam; diferensiasi mengurangkan bacaan dan memperbesar galatnya. Karena itulah diferensiasi lebih rawan di antara keduanya.

9.3 Menurunkan Rumus dari Teorema Taylor

Gagasan garis potong memberi kita rumus, tetapi tidak memberi tahu akurasinya. Untuk mengetahui seberapa keliru sebuah rumus, dan bagaimana galatnya berubah bila langkahnya berubah, kita memakai teorema Taylor, alat di balik hampir setiap taksiran galat dalam buku ini [1], [2].

9.3.1 Teorema Taylor

Misalkan f fungsi yang memiliki cukup banyak turunan kontinu di sekitar titik x, dan h langkah kecil, positif atau negatif. Teorema Taylor menyatakan bahwa nilai f di $x + h$ dapat disusun dari nilai dan turunan-turunan f di x:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \cdots + \frac{h^n}{n!}f^{(n)}(x) + R_n \tag{9.1}$$

dengan f', f'', dan f''' turunan pertama, kedua, dan ketiga, $f^{(n)}$ turunan ke-n, $n! = 1 \times 2 \times \cdots \times n$, dan sisa $R_n = \frac{h^{n+1}}{(n+1)!}f^{(n+1)}(\xi)$ untuk suatu titik ξ di antara x dan $x + h$. Dengan kata lain: di dekat

x, fungsi itu tampak seperti nilainya, ditambah kemiringan kali langkah, ditambah setengah kelengkungan kali kuadrat langkah, dan seterusnya, setiap suku lebih kecil daripada suku sebelumnya bila h kecil. Kita tidak tahu ξ, tetapi biasanya kita tahu, atau dapat membatasi, seberapa besar $f^{(n+1)}$, dan itu membatasi galatnya.

Untuk ojek, f adalah jarak $s(t)$, f' kecepatan, dan f'' percepatan, dan Persamaan (9.1) menyatakan sesuatu yang diketahui setiap pengemudi: jarak setelah waktu singkat h adalah jarak sekarang, ditambah kecepatan kali waktu, ditambah setengah percepatan kali kuadrat waktu, ditambah koreksi-koreksi kecil.

9.3.2 Beda maju dan beda mundur

Ambil Persamaan (9.1) dengan $n = 1$ lalu selesaikan untuk $f'(x)$:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(\xi) \quad \Rightarrow \quad f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2}f''(\xi).$$

Bagian pertama adalah **beda maju** (*forward difference*), kemiringan garis potong ke titik di depan:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \qquad \text{galat} = -\frac{h}{2}f''(\xi). \tag{9.2}$$

Bagian yang kita buang, $-\frac{h}{2}f''(\xi)$, adalah **galat pemotongan** (*truncation error*): galat yang timbul karena deret Taylor dipotong, dan yang tetap ada bahkan pada komputer dengan digit tak hingga. Galat ini sebanding dengan h. Bagi dua langkahnya dan galatnya menjadi separuh; bagi sepuluh langkahnya dan galatnya turun sepuluh kali. Kita katakan beda maju *berakurasi orde satu* dan menulis galatnya sebagai $O(h)$, dibaca "orde h", yang berarti "paling besar suatu konstanta kali h untuk h kecil".

Argumen yang sama dengan $-h$ menggantikan h menghasilkan **beda mundur** (*backward difference*), kemiringan garis potong ke titik di belakang:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}, \qquad \text{galat} = +\frac{h}{2}f''(\xi).$$

Rumus ini juga berorde satu. Untuk Mas Dimas pada $t = 18$, beda maju memberi 5,7 m/s dan beda mundur 9,2 m/s. Suku galatnya menjelaskan selisih itu: f'' adalah percepatan, sekitar -3.5 m/s^2, dan $h = 1$ s, sehingga setiap taksiran meleset sekitar $\frac{1}{2} \times 3.5 = 1.75$ m/s, ke arah yang berlawanan. Nilai sebenarnya semestinya dekat dengan tengahnya, 7.45.

9.3.3 Beda pusat

Pengamatan bahwa galat beda maju dan beda mundur berlawanan tanda mengundang kita untuk menggabungkannya. Tulis Persamaan (9.1) untuk $+h$ dan untuk $-h$, satu suku lebih jauh:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(\xi_1),$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(\xi_2).$$

Kurangkan baris kedua dari baris pertama. Suku $f(x)$ saling menghapus, demikian pula suku $f''(x)$, karena h^2 bertanda sama untuk $+h$ dan $-h$:

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{h^3}{6}(f'''(\xi_1) + f'''(\xi_2)).$$

Bagi dengan $2h$, lalu gunakan teorema nilai antara untuk mengganti rata-rata $f'''(\xi_1)$ dan $f'''(\xi_2)$ dengan satu $f'''(\xi)$:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}, \qquad \text{galat} = -\frac{h^2}{6}f'''(\xi). \tag{9.3}$$

Inilah **beda pusat** (*central difference*). Galatnya sebanding dengan h^2: bagi dua langkahnya dan galatnya turun menjadi seperempat; bagi sepuluh dan galatnya turun seratus kali. Rumus ini *berakurasi orde dua*, $O(h^2)$, dan biayanya persis sama, dua nilai fungsi, dengan beda maju. Jarang ada alasan memakai beda maju bila titik di depan dan titik di belakang sama-sama tersedia.

Satu kasus khusus yang berguna: bila f berupa parabola, $f''' = 0$ dan beda pusat menjadi *eksak*, berapa pun h. Sepeda motor yang mengerem dengan perlambatan tetap memiliki jarak yang berupa parabola terhadap waktu, sehingga pada setiap bentang pengereman tetap, beda pusat memberi kecepatan yang eksak, di luar galat pengukuran. Pada perjalanan Mas Dimas percepatannya berubah mendadak (ia mulai dan berhenti mengerem), sehingga f''' besar pada beberapa saat dan nol di antaranya.

9.3.4 Turunan kedua

Untuk percepatan kita memerlukan f''. Kali ini *jumlahkan* kedua ekspansi, sampai pangkat empat dari h:

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + \frac{h^4}{12}f^{(4)}(\xi).$$

Pangkat-pangkat ganjil saling menghapus, karena h dan h^3 berganti tanda bersama h. Selesaikan untuk $f''(x)$:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}, \qquad \text{galat} = -\frac{h^2}{12}f^{(4)}(\xi). \tag{9.4}$$

Inilah **beda kedua pusat**, rumus pada Subbab 9.2.2. Rumus ini juga berorde dua, dan eksak untuk polinomial kubik. Perhatikan h^2 pada penyebutnya: galat bacaan δ pada masing-masing dari tiga nilai menghasilkan galat hingga $4\delta/h^2$ pada hasilnya. Dengan $h = 1$ s dan $\delta = 0.3$ m itu berarti 1,2 m/s^2, persis bahaya yang kita lihat di Subbab 9.2.2.

9.3.5 Rumus satu sisi untuk ujung-ujung

Pada bacaan pertama sebuah perjalanan tidak ada titik di belakang, dan pada bacaan terakhir tidak ada titik di depan. Beda maju dan beda mundur dua titik dapat dipakai di sana, tetapi hanya

berorde satu. Pilihan yang lebih baik memakai *tiga* titik pada satu sisi. Kita mencari bobot c_0, c_1, c_2 sehingga

$$f'(x) \approx \frac{c_0 f(x) + c_1 f(x+h) + c_2 f(x+2h)}{h}$$

seakurat mungkin. Ekspansikan $f(x+h)$ dan $f(x+2h)$ dengan Persamaan (9.1) lalu kumpulkan suku-suku dalam f, f', dan f''. Agar rumus itu menghasilkan $f'(x)$, koefisien $f(x)$ harus nol, koefisien $hf'(x)$ harus 1, dan koefisien $h^2 f''(x)$ harus nol:

$$c_0 + c_1 + c_2 = 0, \qquad c_1 + 2c_2 = 1, \qquad \frac{1}{2}c_1 + 2c_2 = 0.$$

Dari persamaan ketiga $c_1 = -4c_2$; persamaan kedua memberi $-4c_2 + 2c_2 = 1$, jadi $c_2 = -\frac{1}{2}$ dan $c_1 = 2$; persamaan pertama memberi $c_0 = -\frac{3}{2}$. Dengan demikian

$$f'(x) \approx \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}, \qquad \text{galat} = \frac{h^2}{3} f'''(\xi), \tag{9.5}$$

rumus orde dua yang hanya melihat ke depan. Bayangan cerminnya, $(3f(x) - 4f(x-h) + f(x-2h))/(2h)$, hanya melihat ke belakang. Cara mencari bobot seperti ini, dengan menuntut agar rumus eksak untuk $1, h, h^2, \ldots$, disebut **metode koefisien tak tentu** (*method of undetermined coefficients*). Metode ini menghasilkan setiap rumus dalam bab ini, dan algoritma Fornberg melakukannya secara otomatis untuk himpunan titik apa pun, bahkan yang jaraknya tidak seragam [3]. Metode yang sama dengan empat titik menghasilkan turunan kedua satu sisi berorde dua, $(2f(x) - 5f(x+h) + 4f(x+2h) - f(x+3h))/h^2$, yang dipakai penganalisis di kedua ujung perjalanan.

Konstanta galat Persamaan (9.5), $\frac{1}{3}$, dua kali konstanta beda pusat, $\frac{1}{6}$. Rumus satu sisi selalu kurang akurat dibandingkan rumus berpusat yang ordenya sama, dan juga lebih peka terhadap derau, karena bobotnya lebih besar ($-3{,}4, -1$ dibandingkan $-1{,}1$). Subbab 9.6 memperlihatkan kedua efek itu di awal dan di akhir perjalanan.

9.3.6 Rumus-rumus berdampingan

Tabel 9.2 menghimpun rumus-rumus itu. Polanya patut diingat: untuk menambah satu orde akurasi diperlukan satu titik lagi, dan rumus berpusat mendapat satu orde gratis karena suku-suku galat ganjil saling menghapus berkat kesimetrisan.

Tabel 9.2 Rumus beda hingga untuk turunan pertama dan kedua, beserta banyaknya nilai fungsi yang diperlukan dan galat pemotongannya (ξ tidak ditulis).

Turunan	Rumus	Titik	Galat pemotongan
$f'(x)$ maju	$[f(x+h) - f(x)]/h$	2	$-\frac{h}{2}f''$
$f'(x)$ mundur	$[f(x) - f(x-h)]/h$	2	$+\frac{h}{2}f''$
$f'(x)$ pusat	$[f(x+h) - f(x-h)]/(2h)$	2	$-\frac{h^2}{6}f'''$
$f'(x)$ maju tiga titik	$[-3f(x) + 4f(x+h) - f(x+2h)]/(2h)$	3	$+\frac{h^2}{3}f'''$
$f'(x)$ pusat lima titik	$[f(x-2h) - 8f(x-h) + 8f(x+h) - f(x+2h)]/(12h)$	4	$+\frac{h^4}{30}f^{(5)}$
$f''(x)$ pusat	$[f(x+h) - 2f(x) + f(x-h)]/h^2$	3	$-\frac{h^2}{12}f^{(4)}$
$f''(x)$ maju empat titik	$[2f(x) - 5f(x+h) + 4f(x+2h) - f(x+3h)]/h^2$	4	$+\frac{11h^2}{12}f^{(4)}$

9.4 Dua Jenis Galat: Pemotongan dan Pembulatan

Teorema Taylor mengatakan galat mengecil bila h mengecil. Lalu mengapa tidak mengambil $h = 10^{-12}$ dan mendapatkan turunan yang nyaris sempurna? Untuk bacaan yang dicatat setiap detik jawabannya jelas: kita tidak dapat memilih h, besarnya satu detik. Namun bahkan ketika kita dapat mengevaluasi fungsi di mana saja, membuat h sangat kecil adalah kekeliruan, dan memahami sebabnya mengajarkan pelajaran terpenting bab ini.

Untuk mempelajari galat, kita memerlukan fungsi yang turunannya kita ketahui persis. Ambil $f(x) = \sqrt{x}$ di $x = 4$. Turunannya $f'(x) = \frac{1}{2\sqrt{x}}$, sehingga $f'(4) = 0.25$ tepat. **Kode 9.1** menghitung beda maju dan beda pusat untuk dua belas langkah dari 10^{-1} sampai 10^{-12} dan mencetak setiap taksiran beserta galatnya.

Kode 9.1 `diff_steps.py`: beda maju dan beda pusat dari $\sqrt{x}$ di $x = 4$ untuk langkah yang makin kecil.

```
"""Forward and central differences for f(x) = sqrt(x) at x = 4 with ever
smaller steps h. The exact derivative is 1/(2*sqrt(4)) = 0.25."""
import math


def forward(f, x, h):
    return (f(x + h) - f(x)) / h           # error O(h)


def central(f, x, h):
    return (f(x + h) - f(x - h)) / (2 * h)  # error O(h^2)


def main():
    x, exact = 4.0, 0.25
    steps = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6,
             1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12]
    print("     h     forward        error    central        error")
    for h in steps:
        d1 = forward(math.sqrt, x, h)
        d2 = central(math.sqrt, x, h)
        print(f"{h:6.0e}  {d1:.10f}  {abs(d1 - exact):.2e}"
              f"  {d2:.10f}  {abs(d2 - exact):.2e}")


if __name__ == "__main__":
    main()
```

Kode 9.2 adalah program yang sama dalam Java. Fungsinya dikirim sebagai `DoubleUnaryOperator`, sehingga `Math::sqrt` dapat diberikan kepada rumus-rumus itu sebagaimana `math.sqrt` di Python. Setiap baris keluaran diformat dengan `Locale.ROOT`, agar komputer yang diatur ke bahasa Indonesia tidak mencetak koma desimal; dengan begitu kedua program mencetak karakter yang persis sama.

Kode 9.2 `DiffSteps.java`: percobaan yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Forward and central differences for f(x) = sqrt(x) at x = 4 with ever
 *  smaller steps h. The exact derivative is 1/(2*sqrt(4)) = 0.25. */
public class DiffSteps {
    static double forward(DoubleUnaryOperator f, double x, double h) {
        return (f.applyAsDouble(x + h) - f.applyAsDouble(x)) / h;  // O(h)
    }

    static double central(DoubleUnaryOperator f, double x, double h) {
        // error O(h^2)
        return (f.applyAsDouble(x + h) - f.applyAsDouble(x - h)) / (2 * h);
    }

    public static void main(String[] args) {
        double x = 4.0, exact = 0.25;
        double[] steps = {1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6,
                          1e-7, 1e-8, 1e-9, 1e-10, 1e-11, 1e-12};
        System.out.println(
            "     h     forward        error    central        error");
        for (double h : steps) {
            double d1 = forward(Math::sqrt, x, h);
            double d2 = central(Math::sqrt, x, h);
            System.out.println(String.format(Locale.ROOT,
                "%6.0e  %.10f  %.2e  %.10f  %.2e",
                h, d1, Math.abs(d1 - exact), d2, Math.abs(d2 - exact)));
        }
    }
}
```

Keduanya mencetak:

```
     h     forward        error    central        error
 1e-01  0.2484567313  1.54e-03  0.2500195366  1.95e-05
 1e-02  0.2498439450  1.56e-04  0.2500001953  1.95e-07
 1e-03  0.2499843770  1.56e-05  0.2500000020  1.95e-09
 1e-04  0.2499984375  1.56e-06  0.2500000000  2.05e-11
 1e-05  0.2499998438  1.56e-07  0.2500000000  1.64e-12
 1e-06  0.2499999843  1.57e-08  0.2499999999  7.61e-11
 1e-07  0.2499999985  1.52e-09  0.2499999996  4.09e-10
 1e-08  0.2499999763  2.37e-08  0.2499999874  1.26e-08
 1e-09  0.2500000207  2.07e-08  0.2500000207  2.07e-08
 1e-10  0.2500000207  2.07e-08  0.2500000207  2.07e-08
 1e-11  0.2499778162  2.22e-05  0.2499889185  1.11e-05
 1e-12  0.2500222251  2.22e-05  0.2500222251  2.22e-05
```

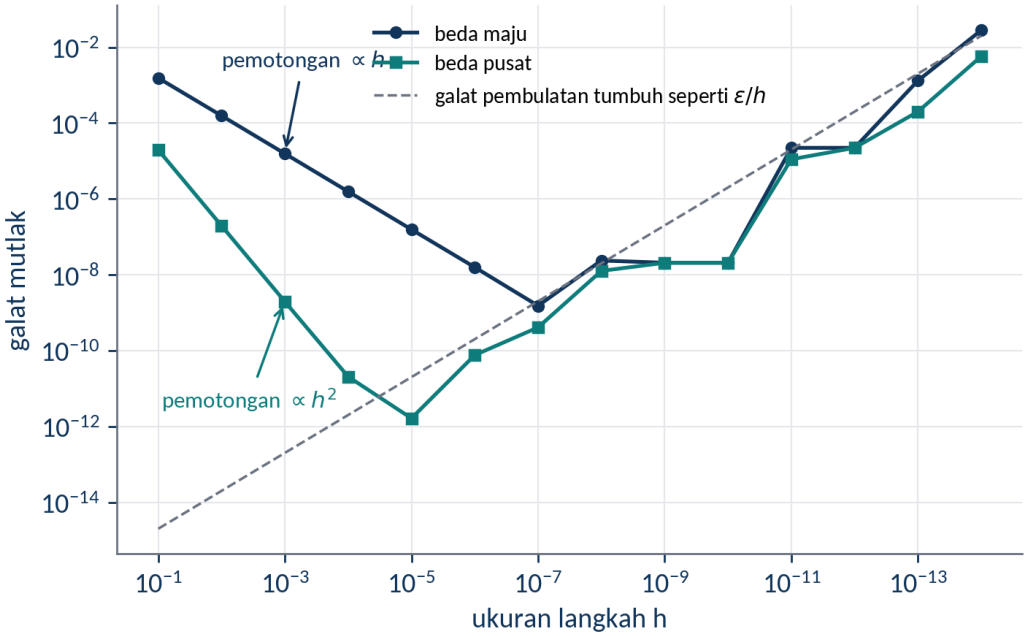
Bacalah kolom beda maju dari atas. Dari $h = 10^{-1}$ sampai $h = 10^{-7}$ galatnya turun sepuluh kali untuk setiap pengecilan h sepuluh kali, persis seperti janji $O(h)$; angkanya bahkan cocok dengan rumusnya: $\frac{h}{2}|f''(4)| = \frac{h}{2} \times \frac{1}{32} = 0.0156\,h$, yang memberi 1.56×10^{-3} untuk $h = 0.1$. Kolom beda pusat turun seratus kali per langkah, seperti janji $O(h^2)$, dengan $\frac{h^2}{6}|f'''(4)| = \frac{h^2}{6} \times \frac{3}{256} = 0.00195\,h^2$, yang memberi 1.95×10^{-5} untuk $h = 0.1$. Kemudian, pada kedua rumus, ada yang rusak. Galat beda maju paling kecil pada $h = 10^{-7}$ dan galat beda pusat pada $h = 10^{-5}$; di bawah langkah-langkah itu galatnya justru *membesar*, dan pada $h = 10^{-12}$ kedua taksiran keliru pada digit kelima, lebih buruk daripada dengan $h = 0.01$.

Biang keladinya adalah **galat pembulatan** (*round-off error*), pokok bahasan Bab 1. Bilangan presisi ganda membawa sekitar 16 digit bermakna; jarak relatif antarbilangan double adalah epsilon mesin $\varepsilon = 2^{-52} \approx 2.2 \times 10^{-16}$ [4]. Karena itu nilai $\sqrt{4 + h}$ tersimpan dengan galat hingga sekitar $\varepsilon \times 2$, demikian pula $\sqrt{4}$. Bila h sangat kecil, kedua nilai itu sama pada sebagian besar

digitnya, dan pengurangan menghapus digit-digit itu, menyisakan hanya digit yang tidak pasti: inilah pembatalan katastrofik (*catastrophic cancellation*) dari Bab 1. Selisih itu, dengan galat mutlak beberapa kali 10^{-16}, lalu dibagi h. Bagian galat akibat pembulatan kira-kira

$$E_{\text{bulat}} \approx \frac{2\varepsilon|f(x)|}{h}$$

untuk beda maju: galat ini membesar ketika h mengecil. Galat totalnya adalah jumlah bagian pemotongan yang mengecil bersama h dan bagian pembulatan yang membesar ketika h mengecil. **Gambar 9.1** menggambar kedua kolom keluaran itu pada sumbu logaritmik. Setiap kurva berbentuk huruf "V": turun lurus dengan kemiringan 1 (maju) atau 2 (pusat) selama pemotongan dominan, lalu naik dengan kasar berkemiringan -1 begitu pembulatan mengambil alih.



Gambar 9.1 Galat beda maju dan beda pusat dari $\sqrt{x}$ di $x = 4$ terhadap langkah h (kedua sumbu logaritmik, h mengecil ke kanan). Galat pemotongan turun seperti h atau h^2; galat pembulatan naik seperti ε/h.

Di mana dasar huruf V itu? Untuk beda maju galat totalnya kira-kira

$$E(h) = \frac{h}{2}M_2 + \frac{2\varepsilon|f|}{h},$$

dengan M_2 batas atas \$f'\$. Turunannya terhadap h adalah $\frac{M_2}{2} - \frac{2\varepsilon|f|}{h^2}$, yang bernilai nol di

$$h^* = 2\sqrt{\frac{\varepsilon|f|}{M_2}} \quad \text{(maju)}, \qquad h^* = \left(\frac{3\varepsilon|f|}{M_3}\right)^{1/3} \quad \text{(pusat)}, \tag{9.6}$$

dengan versi pusat diperoleh dengan cara yang sama dari $E(h) = \frac{h^2}{6}M_3 + \frac{\varepsilon|f|}{h}$ dan M_3 batas atas \$f'''\$. Untuk fungsi yang nilai dan turunan-turunannya semuanya berukuran sekitar 1, ini memberi aturan praktis $h^* \approx \sqrt{\varepsilon} \approx 10^{-8}$ untuk beda maju dan $h^* \approx \varepsilon^{1/3} \approx 10^{-5}$ untuk beda pusat, dengan galat terbaik sekitar 10^{-8} dan 10^{-11}. Untuk $\sqrt{x}$ di 4, dengan $|f| = 2$, $M_2 = 1/32$, dan $M_3 = 3/256$, Persamaan (9.6) memberi $h^* \approx 2.4 \times 10^{-7}$ dan $h^* \approx 4.8 \times 10^{-5}$, cocok dengan

dasar kurva pada **Gambar 9.1**. Dalam praktik h juga diskalakan dengan x, yaitu $h = \sqrt{\varepsilon}\max(1, |x|)$, karena jarak antarbilangan double membesar seiring besarnya x [2], [4].

Bagian datar kedua kurva antara $h = 10^{-9}$ dan 10^{-10}, tempat taksiran beda maju dan beda pusat sama persis, adalah galat pembulatan dalam bentuk paling murni: $4 + h$ dibulatkan ke bilangan double terdekat, sehingga langkah yang sesungguhnya dipakai bukan h yang kita minta, dan kedua rumus membagi beberapa bit yang tersisa dengan langkah keliru yang sama. Program yang cermat menghitung langkah yang benar-benar dipakai sebagai `(x + h) - x` lalu membagi dengannya; Soal Latihan 9.3 menjelajahi aturan praktis ini lebih jauh.

> **PERINGATAN:** Tidak ada langkah yang membuat turunan beda hingga seakurat nilai fungsinya. Dengan presisi ganda, beda maju kehilangan sekitar separuh digit dan beda pusat sekitar sepertiganya. Bila Anda memerlukan turunan dengan presisi penuh, turunkan rumusnya dengan tangan, atau gunakan diferensiasi otomatis (*automatic differentiation*), yang akan kita jumpai dalam machine learning di Bab 14.

Untuk data hasil pengukuran, pelajarannya lebih tajam lagi. Di sana, "pembulatan"-nya adalah galat pengukuran bacaan, yang bukan 10^{-16} melainkan, untuk ponsel Mas Dimas, sekitar 10^{-1} m. Argumen langkah optimal tetap berlaku, dengan f diganti galat bacaan δ: langkah yang lebih besar mengurangi pengaruh derau, langkah yang lebih kecil mengurangi pengaruh kelengkungan. Dengan bacaan setiap detik kita tidak dapat memperkecil langkah, tetapi kita dapat memperlebarnya, dan Subbab 9.7 melakukan sesuatu yang lebih baik lagi.

9.5 Ekstrapolasi Richardson

Rumus galat beda pusat menyatakan lebih dari sekadar "orde h^2". Teorema Taylor, bila diteruskan, memperlihatkan bahwa beda pusat $D(h)$ memiliki galat berupa deret dalam pangkat *genap* dari h:

$$D(h) = f'(x) + a_1 h^2 + a_2 h^4 + a_3 h^6 + \cdots$$

dengan a_i bergantung pada turunan-turunan tinggi f di x tetapi tidak pada h. Hitung D dua kali, dengan h dan $h/2$:

$$D(h) = f'(x) + a_1 h^2 + a_2 h^4 + \cdots, \qquad D(h/2) = f'(x) + \frac{a_1 h^2}{4} + \frac{a_2 h^4}{16} + \cdots$$

Empat kali yang kedua dikurangi yang pertama menghapus suku h^2:

$$\frac{4D(h/2) - D(h)}{3} = f'(x) - \frac{a_2 h^4}{4} + \cdots$$

sebuah taksiran orde empat dari dua taksiran orde dua. Inilah **ekstrapolasi Richardson**, yang diperkenalkan Lewis Fry Richardson pada 1911 untuk perhitungan cuacanya [5]. Tidak ada yang khusus tentang turunan di sini: setiap kali sebuah hampiran memiliki galat berupa deret pangkat h yang diketahui bentuknya, menggabungkan dua ukuran langkah menghapus suku terdepannya. Mengulang cara itu menghasilkan tabel segitiga. Bila $D_{i,0} = D(h/2^i)$, maka

$$D_{i,j} = \frac{4^j D_{i,j-1} - D_{i-1,j-1}}{4^j - 1}, \qquad j = 1,2,\ldots,i, \tag{9.7}$$

dan kolom j bergalat $O(h^{2j+2})$. Bab 10 membangun tabel yang persis sama untuk integral, dan di sana namanya integrasi Romberg.

Pemeriksaan kecil dengan tangan memperlihatkan langkah pertamanya bekerja. Beda pusat $\sqrt{x}$ di 4 dengan $h = 0.8$ adalah $(\sqrt{4.8} - \sqrt{3.2})/1.6 = (2.190890 - 1.788854)/1.6 = 0.251272$, dan dengan $h = 0.4$ adalah $(\sqrt{4.4} - \sqrt{3.6})/0.8 = (2.097618 - 1.897367)/0.8 = 0.250314$. Galatnya 1.27×10^{-3} dan 3.14×10^{-4}, rasio yang hampir tepat 4, seperti ramalan $O(h^2)$. Ekstrapolasinya memberi $(4 \times 0.250314 - 0.251272)/3 = 0.249994$, dengan galat sekitar 6×10^{-6}: lebih dari lima puluh kali lebih baik daripada taksiran yang lebih baik di antara keduanya, dari empat akar kuadrat yang sama.

Kode 9.3 membangun tabel Persamaan (9.7) dengan empat tingkat, mulai dari langkah yang sengaja dibuat kasar, $h = 0.8$.

Kode 9.3 `richardson.py`: tabel Richardson untuk beda pusat.

```python
"""Richardson extrapolation of the central difference for f(x) = sqrt(x)
at x = 4 (exact derivative 0.25)."""
import math


def central(f, x, h):
    return (f(x + h) - f(x - h)) / (2 * h)


def richardson(f, x, h, levels):
    """Row i starts with D(h/2^i); column j removes the h^(2j) error term."""
    table = []
    for i in range(levels):
        row = [central(f, x, h / 2 ** i)]
        for j in range(1, i + 1):
            factor = 4 ** j
            row.append((factor * row[j - 1] - table[i - 1][j - 1])
                       / (factor - 1))
        table.append(row)
    return table


def main():
    exact = 0.25
    table = richardson(math.sqrt, 4.0, 0.8, 4)
    for i, row in enumerate(table):
        cells = "  ".join(f"{d:.12f}" for d in row)
        print(f"h = {0.8 / 2 ** i:.3f}  {cells}")
    print("errors of the diagonal:")
    for i, row in enumerate(table):
        print(f"  level {i}: {abs(row[i] - exact):.2e}")


if __name__ == "__main__":
    main()
```

Kode 9.4 `Richardson.java`: tabel yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Richardson extrapolation of the central difference for f(x) = sqrt(x)
 *  at x = 4 (exact derivative 0.25). */
public class Richardson {
    static double central(DoubleUnaryOperator f, double x, double h) {
        return (f.applyAsDouble(x + h) - f.applyAsDouble(x - h)) / (2 * h);
    }

    /** Row i starts with D(h/2^i); column j removes the h^(2j) error term. */
    static double[][] richardson(DoubleUnaryOperator f, double x, double h,
                                 int levels) {
        double[][] table = new double[levels][];
        for (int i = 0; i < levels; i++) {
            table[i] = new double[i + 1];
            table[i][0] = central(f, x, h / Math.pow(2, i));
            for (int j = 1; j <= i; j++) {
                double factor = Math.pow(4, j);
                table[i][j] = (factor * table[i][j - 1] - table[i - 1][j - 1])
                              / (factor - 1);
            }
        }
        return table;
    }

    public static void main(String[] args) {
        double exact = 0.25;
        double[][] table = richardson(Math::sqrt, 4.0, 0.8, 4);
        for (int i = 0; i < table.length; i++) {
            StringBuilder cells = new StringBuilder();
            for (int j = 0; j < table[i].length; j++) {
                cells.append(j > 0 ? "  " : "")
                     .append(String.format(Locale.ROOT, "%.12f", table[i][j]));
            }
            System.out.println(String.format(Locale.ROOT, "h = %.3f  %s",
                0.8 / Math.pow(2, i), cells));
        }
        System.out.println("errors of the diagonal:");
        for (int i = 0; i < table.length; i++) {
            System.out.println(String.format(Locale.ROOT, "  level %d: %.2e",
                i, Math.abs(table[i][i] - exact)));
        }
    }
}
```

Kode 9.4 mencetak tabel yang sama:

```
h = 0.800  0.251272405013
h = 0.400  0.250313875299  0.249994365394
h = 0.200  0.250078210575  0.249999655667  0.250000008352
h = 0.100  0.250019536593  0.249999978598  0.250000000127  0.249999999996
errors of the diagonal:
  level 0: 1.27e-03
  level 1: 5.63e-06
  level 2: 8.35e-09
  level 3: 3.55e-12
```

Kolom pertama, beda pusat biasa, membaik empat kali per baris. Diagonalnya membaik jauh lebih banyak: dari 1.27×10^{-3} sampai 3.55×10^{-12} dengan total delapan akar kuadrat, tidak satu pun dengan langkah lebih kecil dari 0.1. Bandingkan dengan **Gambar 9.1**, tempat hasil terbaik satu beda pusat adalah sekitar 10^{-12} dengan $h = 10^{-5}$, berdiri di tepi jurang pembulatan. Ekstrapolasi Richardson mencapai akurasi tinggi dengan langkah yang *besar*, sehingga pembulatan tetap tidak berbahaya. Itulah nilai sejatinya.

Ada harganya: deret galat itu harus benar-benar ada, jadi f harus mulus, dan ekstrapolasi harus diterapkan pada nilai yang cukup akurat sehingga selisihnya bermakna. Pada data berderau, seperti catatan perjalanan, ekstrapolasi justru memperbesar derau; bobotnya, $4/3$ pada satu taksiran dan $-1/3$ pada yang lain, memperbesar galat acak. Richardson adalah alat untuk fungsi yang dapat Anda evaluasi, bukan untuk bacaan yang diberikan kepada Anda.

9.6 Membangun Penganalisis Perjalanan

Kembali ke Mas Dimas. Catatan perjalanannya, `data/trip.csv`, berisi satu baris judul lalu 41 baris berbentuk `t,distance`, dari `0,0.0` sampai `40,305.4`. Catatan dalam folder kode buku ini disiapkan untuk bab ini agar sesuai dengan perjalanan yang persis seperti yang ia ceritakan, sehingga kita tahu apa yang sebenarnya terjadi, dan justru itulah yang menjadikannya bahan uji yang baik: ia melaju sekitar 40 km/jam (11 m/s), mengerem keras karena kucing selama tiga detik mulai $t = 17$, melaju lagi sekitar 30 km/jam, dan mulai $t = 34$ meluncur pelan sampai berhenti di depan pagar pelanggan, dengan perlambatan sekitar -1.4 m/s^2. Jarak pada ponsel tidak pasti sekitar 0,3 m dan disimpan sampai sepersepuluh meter.

Penganalisis ini memerlukan tiga bagian: pembaca berkas, fungsi untuk kecepatan, dan fungsi untuk percepatan. Kecepatan memakai beda pusat Persamaan (9.3) pada setiap bacaan yang bertetangga di kedua sisinya, dan rumus satu sisi tiga titik Persamaan (9.5) pada bacaan pertama dan terakhir, sehingga setiap nilai berakurasi orde dua. Percepatan memakai beda kedua pusat Persamaan (9.4) di bagian dalam dan rumus satu sisi empat titik dari **Tabel 9.2** di kedua ujung. Bacaan yang lebih negatif daripada ambang `HARSH` ditandai. **Kode 9.5** adalah program lengkapnya.

Kode 9.5 `speed.py`: kecepatan dan percepatan dari catatan jarak, dengan pengereman mendadak ditandai.

```
"""Speed and acceleration of an ojek trip from the distance the phone logged
once a second. Usage: python speed.py [csv file]  (default ../data/trip.csv)"""
import sys

HARSH = -3.0   # m/s^2: braking harder than this is reported


def load(path):
    times, dist = [], []
    with open(path) as f:
        next(f)                                   # skip the header line
        for line in f:
            t, s = line.strip().split(",")
            times.append(float(t))
            dist.append(float(s))
    return times, dist


def speeds(s, h):
    """First derivative: central inside, three-point one-sided at the ends."""
    n = len(s)
    v = [0.0] * n
    v[0] = (-3 * s[0] + 4 * s[1] - s[2]) / (2 * h)
    for i in range(1, n - 1):
        v[i] = (s[i + 1] - s[i - 1]) / (2 * h)
    v[n - 1] = (3 * s[n - 1] - 4 * s[n - 2] + s[n - 3]) / (2 * h)
    return v


def accelerations(s, h):
    """Second derivative: central inside, four-point one-sided at the ends."""
    n = len(s)
    a = [0.0] * n
    a[0] = (2 * s[0] - 5 * s[1] + 4 * s[2] - s[3]) / (h * h)
    for i in range(1, n - 1):
        a[i] = (s[i + 1] - 2 * s[i] + s[i - 1]) / (h * h)
    a[n - 1] = (2 * s[n - 1] - 5 * s[n - 2] + 4 * s[n - 3] - s[n - 4]) / (h * h)
    return a
```

Kode 9.5 (Lanjutan) *speed.py: kecepatan dan percepatan dari catatan jarak, dengan pengereman mendadak ditandai.*

```
def main(args):
    path = args[0] if args else "../data/trip.csv"
    t, s = load(path)
    h = t[1] - t[0]
    v = speeds(s, h)
    a = accelerations(s, h)
    print("  t     s(m)    v(m/s)  v(km/h)  a(m/s^2)")
    for i in range(len(t)):
        flag = "  harsh braking" if a[i] < HARSH else ""
        print(f"{t[i]:3.0f}  {s[i]:7.1f}  {v[i]:7.2f}  {v[i] * 3.6:7.1f}"
              f"  {a[i]:8.2f}{flag}")
    top = max(range(len(v)), key=lambda i: v[i])
    print(f"top speed {v[top] * 3.6:.1f} km/h at t = {t[top]:.0f} s")


if __name__ == "__main__":
    main(sys.argv[1:])
```

Tiga hal kecil patut disebut. Langkah `h` diambil dari berkas (`t[1] - t[0]`), bukan ditulis mati di dalam program, sehingga catatan setiap dua detik pun dapat dipakai tanpa perubahan. Fungsi `speeds` dan `accelerations` menerima daftar biasa dan sebuah langkah, dan tidak tahu apa-apa

tentang ojek; keduanya akan dipakai ulang oleh program penghalusan dan oleh pengujian. Kecepatan tertinggi dicari dengan `max` atas indeks, agar waktu terjadinya juga dapat dicetak. **Kode 9.6** adalah versi Java-nya; `load` mengembalikan kedua kolom sebagai larik dua baris.

Kode 9.6 `Speed.java`: penganalisis yang sama dalam Java.

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Locale;

/** Speed and acceleration of an ojek trip from the distance the phone logged
 *  once a second. Usage: java Speed [csv file]  (default ../data/trip.csv) */
public class Speed {
    static final double HARSH = -3.0;   // m/s^2: harder braking is reported

    /** Returns {times, distances}. */
    static double[][] load(String path) throws IOException {
        List<String> lines = Files.readAllLines(Path.of(path));
        int n = lines.size() - 1;                    // the first line is a header
        double[] t = new double[n], s = new double[n];
        for (int i = 0; i < n; i++) {
            String[] parts = lines.get(i + 1).trim().split(",");
            t[i] = Double.parseDouble(parts[0]);
            s[i] = Double.parseDouble(parts[1]);
        }
        return new double[][] {t, s};
    }

    /** First derivative: central inside, three-point one-sided at the ends. */
    static double[] speeds(double[] s, double h) {
        int n = s.length;
        double[] v = new double[n];
        v[0] = (-3 * s[0] + 4 * s[1] - s[2]) / (2 * h);
        for (int i = 1; i < n - 1; i++) {
            v[i] = (s[i + 1] - s[i - 1]) / (2 * h);
        }
        v[n - 1] = (3 * s[n - 1] - 4 * s[n - 2] + s[n - 3]) / (2 * h);
        return v;
    }
```

Kode 9.6 (Lanjutan) *Speed.java: penganalisis yang sama dalam Java.*

```
    /** Second derivative: central inside, four-point one-sided at the ends. */
    static double[] accelerations(double[] s, double h) {
        int n = s.length;
        double[] a = new double[n];
        a[0] = (2 * s[0] - 5 * s[1] + 4 * s[2] - s[3]) / (h * h);
        for (int i = 1; i < n - 1; i++) {
            a[i] = (s[i + 1] - 2 * s[i] + s[i - 1]) / (h * h);
        }
        a[n - 1] = (2 * s[n - 1] - 5 * s[n - 2] + 4 * s[n - 3] - s[n - 4])
                   / (h * h);
        return a;
    }

    public static void main(String[] args) throws IOException {
        double[][] data = load(args.length > 0 ? args[0] : "../data/trip.csv");
        double[] t = data[0], s = data[1];
        double h = t[1] - t[0];
        double[] v = speeds(s, h), a = accelerations(s, h);
        System.out.println("  t     s(m)   v(m/s)  v(km/h)  a(m/s^2)");
        int top = 0;
        for (int i = 0; i < t.length; i++) {
            String flag = a[i] < HARSH ? "  harsh braking" : "";
            System.out.println(String.format(Locale.ROOT,
                "%3.0f  %7.1f  %7.2f  %7.1f  %8.2f%s",
                t[i], s[i], v[i], v[i] * 3.6, a[i], flag));
            if (v[i] > v[top]) {
                top = i;
            }
        }
        System.out.println(String.format(Locale.ROOT,
            "top speed %.1f km/h at t = %.0f s", v[top] * 3.6, t[top]));
    }
}
```

Dijalankan dari folder `python` atau `java`, keduanya mencetak 43 baris yang sama:

```
  t     s(m)   v(m/s)  v(km/h)  a(m/s^2)
  0      0.0    -0.90     -3.2      3.90
  1      0.6     2.10      7.6      3.00
  2      4.2     4.65     16.7      2.10
  3      9.9     6.40     23.0      1.40
  4     17.0     8.70     31.3      3.20
  5     27.3    11.15     40.1      1.70
  6     39.3    10.85     39.1     -2.30
  7     49.0    10.50     37.8      1.60
  8     60.3    11.25     40.5     -0.10
  9     71.5    10.75     38.7     -0.90
 10     81.8    11.00     39.6      1.40
 11     93.5    11.50     41.4     -0.40
 12    104.8    10.95     39.4     -0.70
 13    115.4    10.65     38.3      0.10
 14    126.1    11.05     39.8      0.70
 15    137.5    11.05     39.8     -0.70
 16    148.2    11.00     39.6      0.60
 17    159.5    10.25     36.9     -2.10
 18    168.7     7.45     26.8     -3.50  harsh braking
 19    174.4     3.90     14.0     -3.60  harsh braking
 20    176.5     2.15      7.7      0.10
 21    178.7     2.65      9.5      0.90
 22    181.8     4.40     15.8      2.60
 23    187.5     6.50     23.4      1.60
 24    194.8     7.85     28.3      1.10
 25    203.2     8.55     30.8      0.30
 26    211.9     8.55     30.8     -0.30
 27    220.3     8.50     30.6      0.20
 28    228.9     8.50     30.6     -0.20
 29    237.3     8.35     30.1     -0.10
 30    245.6     8.50     30.6      0.40
 31    254.3     8.65     31.1     -0.10
 32    262.9     8.65     31.1      0.10
 33    271.6     8.30     29.9     -0.80
 34    279.5     8.15     29.3      0.50
 35    287.9     6.85     24.7     -3.10  harsh braking
 36    293.2     5.35     19.3      0.10
 37    298.6     4.60     16.6     -1.60
 38    302.4     2.90     10.4     -1.80
 39    304.4     1.50      5.4     -1.00
 40    305.4     0.50      1.8     -0.20
top speed 41.4 km/h at t = 11 s
```

Bacalah tabel itu seperti Mas Dimas membacanya. Kecepatan naik sampai sekitar 40 km/jam dalam lima detik dan bertahan di sana, bergoyang antara 38 dan 41; turun sampai di bawah 8 km/jam pada $t = 20$; naik lagi sampai sekitar 31 km/jam; dan turun hampir nol di akhir. Percepatan pada $t = 18$ dan $t = 19$ adalah -3.50 dan -3.60 m/s^2, ditandai: itulah si kucing, dan aplikasinya benar. Tetapi ada tanda ketiga, pada $t = 35$, saat berhenti pelan di depan pagar, padahal perlambatan sebenarnya hanya -1.4 m/s^2. Aplikasi di ponselnya kemungkinan besar membuat kekeliruan yang sama. Mas Dimas pun benar.

Tabel itu juga memperlihatkan gejala derau lain yang telah diramalkan Subbab 9.3. Kecepatan pada $t = 0$ adalah -0.90 m/s, sesuatu yang mustahil: rumus satu sisi di awal berbobot $-3,4,-1$, dan derau 0,3 m pada bacaan $t = 1$ saja sudah menggesernya sebesar $4 \times 0.3/2 = 0.6$ m/s. Kecepatan tertinggi, 41,4 km/jam pada $t = 11$, adalah goyangan derau yang kebetulan, bukan saat benar-benar ngebut; kecepatan jelajah sebenarnya 39,6 km/jam. Kolom percepatan pun melompat-lompat satu sampai dua meter per detik kuadrat selama melaju tetap antara $t = 6$ dan $t = 16$, padahal percepatan sebenarnya nol; nilai -2.30 pada $t = 6$ nyaris menjadi alarm palsu kedua. Semua ini bukan kesalahan pemrograman. Ini aritmetika Persamaan (9.4): galat δ pada bacaan menjadi galat hingga $4\delta/h^2$ pada percepatan.

Kita dapat lebih teliti. Bila galat bacaan saling bebas dengan simpangan baku σ, galat $s_{i+1} - 2s_i + s_{i-1}$ memiliki variansi $(1+4+1)\sigma^2 = 6\sigma^2$, sehingga percepatan yang dihitung memiliki simpangan baku $\sqrt{6}\,\sigma/h^2 \approx 2.45 \times 0.3 = 0.73$ m/s². Selama berhenti pelan, nilai sebenarnya -1.42 hanya berjarak 1.58 m/s², sedikit lebih dari dua simpangan baku, dari ambang, sehingga masing-masing dari keenam bacaan itu berpeluang satu atau dua persen ditandai oleh derau semata. Dalam banyaknya pemberhentian sepanjang hari kerja, alarm palsu bukan nasib sial; alarm palsu itu pasti terjadi.

9.7 Menyempurnakannya: Haluskan Dulu, Baru Turunkan

9.7.1 Cocokkan parabola, lalu turunkan parabolanya

Beda pusat menurunkan parabola yang melalui tiga bacaan bertetangga. Bila bacaannya berderau, parabola itu setia mengikuti deraunya. Gagasan yang lebih baik, dari Savitzky dan Golay pada 1964 [6], adalah mencocokkan parabola melalui *lima* bacaan bertetangga dengan kuadrat terkecil, metode Bab 8, lalu menurunkan parabola hasil pencocokan itu. Parabola hasil pencocokan tidak dapat melalui kelima titik, sehingga sebagian derau diabaikan, dan turunan parabola mudah dihitung.

Ukur waktu dari bacaan tengah dalam satuan langkah: $\tau = -2, -1, 0, 1, 2$, dengan bacaan $s_{-2}, \ldots, s_2$, lalu cocokkan $p(\tau) = c_0 + c_1\tau + c_2\tau^2$. Persamaan normal dari Bab 8 memuat jumlah-jumlah $\sum \tau^k$ atas kelima titik: $\sum 1 = 5$, $\sum \tau = 0$, $\sum \tau^2 = 10$, $\sum \tau^3 = 0$, dan $\sum \tau^4 = 34$. Karena jumlah-jumlah ganjil bernilai nol, persamaan untuk c_1 terpisah dari yang lain:

$$c_1 = \frac{\sum \tau s_\tau}{\sum \tau^2} = \frac{-2s_{-2} - s_{-1} + s_1 + 2s_2}{10},$$

sedangkan c_0 dan c_2 memenuhi $5c_0 + 10c_2 = \sum s_\tau$ dan $10c_0 + 34c_2 = \sum \tau^2 s_\tau$, yang memberi $c_2 = (\sum \tau^2 s_\tau - 2\sum s_\tau)/14$. Turunan p di $\tau = 0$, dikonversi dari langkah ke detik, adalah c_1/h dan $2c_2/h^2$:

$$v \approx \frac{-2s_{-2} - s_{-1} + s_1 + 2s_2}{10h}, \qquad a \approx \frac{2s_{-2} - s_{-1} - 2s_0 - s_1 + 2s_2}{7h^2}. \tag{9.8}$$

Ini adalah jumlah berbobot tetap dari lima bacaan, sama murahnya dengan rumus biasa. Pengaruhnya pada derau mudah dihitung. Untuk kecepatan, faktor variansinya $(4+1+1+4)/100 = 0.1$, bukan 0.5 seperti pada beda pusat; untuk percepatan, $(4+1+4+1+4)/49 = 0.286$, bukan 6. Simpangan baku derau pada percepatan turun dari 0.73 menjadi 0.16 m/s², lebih dari empat kali lebih kecil.

Tidak ada yang gratis. Parabola melalui lima titik yang merentang empat detik tidak dapat mengikuti perubahan yang terjadi dalam satu detik, sehingga rem keras yang singkat menjadi tersebar dan tampak sedikit lebih lunak. Itulah tawar-menawar yang sama dengan langkah optimal di Subbab 9.4: derau berkurang, pemotongan bertambah. Untuk pengereman selama tiga detik, seperti di sini, tawar-menawar itu menguntungkan.

9.7.2 Penganalisis yang dihaluskan

Kode 9.7 menghitung Persamaan (9.8) pada setiap bacaan yang memiliki dua tetangga di setiap sisi, mencetaknya di samping nilai biasa, dan mendaftar saat-saat pengereman mendadak yang

ditemukan dengan masing-masing cara. Program ini mengimpor `load`, `speeds`, `accelerations`, dan `HARSH` dari `speed.py`, sehingga kedua program tidak mungkin berjalan sendiri-sendiri.

Kode 9.7 `smooth.py`: kecepatan dan percepatan dari parabola kuadrat terkecil lima titik, di samping rumus biasa.

```
"""Speed and acceleration from a local least-squares parabola through five
points (Savitzky-Golay), compared with the plain three-point formulas.
Usage: python smooth.py [csv file]  (default ../data/trip.csv)"""
import sys
from speed import HARSH, load, speeds, accelerations


def smooth_speed(s, i, h):
    return (-2 * s[i - 2] - s[i - 1] + s[i + 1] + 2 * s[i + 2]) / (10 * h)


def smooth_accel(s, i, h):
    return (2 * s[i - 2] - s[i - 1] - 2 * s[i] - s[i + 1]
            + 2 * s[i + 2]) / (7 * h * h)


def main(args):
    path = args[0] if args else "../data/trip.csv"
    t, s = load(path)
    h = t[1] - t[0]
    v, a = speeds(s, h), accelerations(s, h)
    plain, smooth = [], []
    print("  t   v plain  v smooth   a plain  a smooth")
    for i in range(2, len(t) - 2):
        vs, as_ = smooth_speed(s, i, h), smooth_accel(s, i, h)
        print(f"{t[i]:3.0f}  {v[i]:7.2f}  {vs:8.2f}  {a[i]:8.2f}  {as_:8.2f}")
        if a[i] < HARSH:
            plain.append(f"{t[i]:.0f}")
        if as_ < HARSH:
            smooth.append(f"{t[i]:.0f}")
    print("harsh braking, plain formulas:  t =", ", ".join(plain))
    print("harsh braking, smoothed:        t =", ", ".join(smooth))


if __name__ == "__main__":
    main(sys.argv[1:])
```

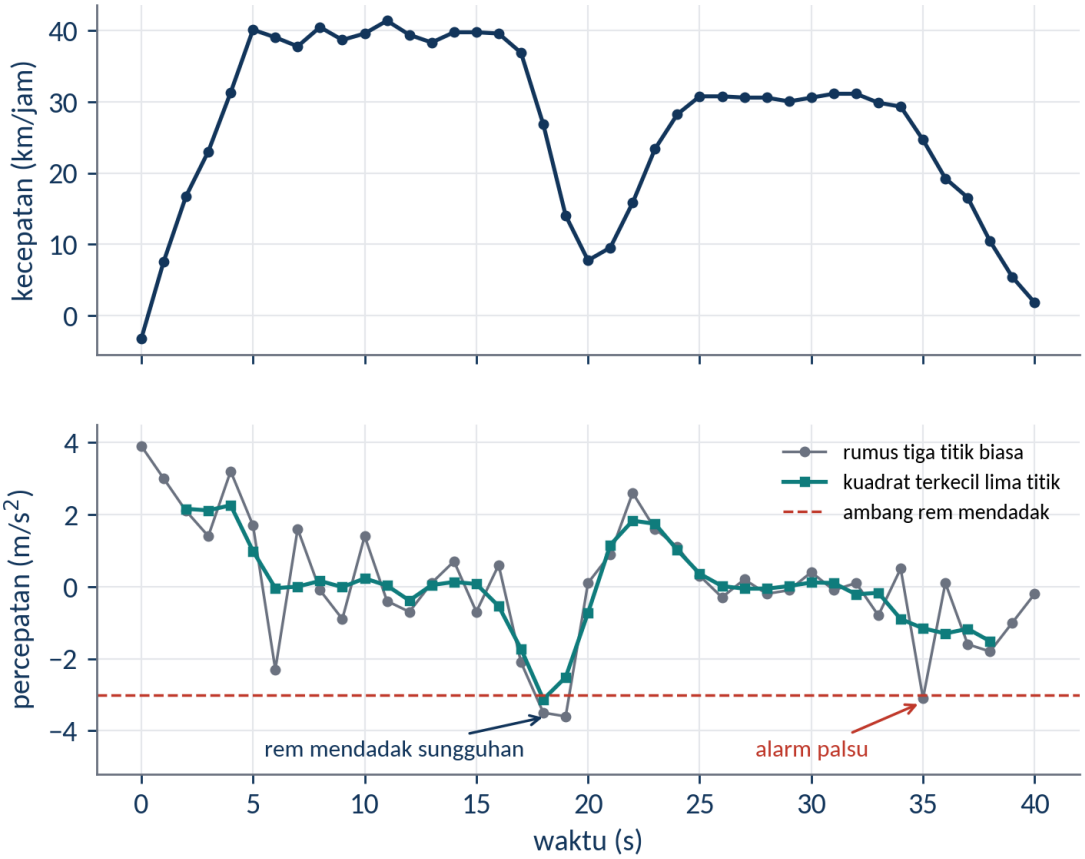
Kode 9.8 `Smooth.java`: perbandingan yang sama dalam Java.

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

/** Speed and acceleration from a local least-squares parabola through five
 *  points (Savitzky-Golay), compared with the plain three-point formulas.
 *  Usage: java Smooth [csv file]  (default ../data/trip.csv) */
public class Smooth {
    static double smoothSpeed(double[] s, int i, double h) {
        return (-2 * s[i - 2] - s[i - 1] + s[i + 1] + 2 * s[i + 2]) / (10 * h);
    }

    static double smoothAccel(double[] s, int i, double h) {
        return (2 * s[i - 2] - s[i - 1] - 2 * s[i] - s[i + 1]
                + 2 * s[i + 2]) / (7 * h * h);
    }

    public static void main(String[] args) throws IOException {
        String path = args.length > 0 ? args[0] : "../data/trip.csv";
        double[][] data = Speed.load(path);
        double[] t = data[0], s = data[1];
        double h = t[1] - t[0];
        double[] v = Speed.speeds(s, h), a = Speed.accelerations(s, h);
        List<String> plain = new ArrayList<>(), smooth = new ArrayList<>();
        System.out.println("  t   v plain  v smooth   a plain  a smooth");
        for (int i = 2; i < t.length - 2; i++) {
            double vs = smoothSpeed(s, i, h), as = smoothAccel(s, i, h);
            System.out.println(String.format(Locale.ROOT,
                "%3.0f  %7.2f  %8.2f  %8.2f  %8.2f", t[i], v[i], vs, a[i], as));
            if (a[i] < Speed.HARSH) {
                plain.add(String.format(Locale.ROOT, "%.0f", t[i]));
            }
            if (as < Speed.HARSH) {
                smooth.add(String.format(Locale.ROOT, "%.0f", t[i]));
            }
        }
        System.out.println("harsh braking, plain formulas:  t = "
                           + String.join(", ", plain));
        System.out.println("harsh braking, smoothed:        t = "
                           + String.join(", ", smooth));
    }
}
```

Kode 9.8 memakai ulang `Speed.load`, `Speed.speeds`, dan `Speed.accelerations` dengan cara yang sama. Keduanya mencetak:

```
  t   v plain  v smooth   a plain  a smooth
  2     4.65      4.33      2.10      2.16
  3     6.40      6.62      1.40      2.11
  4     8.70      8.76      3.20      2.26
  5    11.15     10.05      1.70      0.99
  6    10.85     10.83     -2.30     -0.04
  7    10.50     10.94      1.60      0.00
  8    11.25     10.75     -0.10      0.16
  9    10.75     11.05     -0.90     -0.01
 10    11.00     11.10      1.40      0.23
 11    11.50     11.08     -0.40      0.03
 12    10.95     11.05     -0.70     -0.39
 13    10.65     10.93      0.10      0.04
 14    11.05     10.89      0.70      0.13
 15    11.05     11.03     -0.70      0.07
 16    11.00     10.72      0.60     -0.54
 17    10.25      9.43     -2.10     -1.73
 18     7.45      7.15     -3.50     -3.13
 19     3.90      4.62     -3.60     -2.51
 20     2.15      3.05      0.10     -0.73
 21     2.65      3.15      0.90      1.16
 22     4.40      4.54      2.60      1.83
 23     6.50      6.20      1.60      1.74
 24     7.85      7.59      1.10      1.01
 25     8.55      8.27      0.30      0.36
 26     8.55      8.53     -0.30      0.01
 27     8.50      8.52      0.20     -0.06
 28     8.50      8.44     -0.20     -0.06
 29     8.35      8.47     -0.10      0.01
 30     8.50      8.50      0.40      0.11
 31     8.65      8.59     -0.10      0.10
 32     8.65      8.51      0.10     -0.21
 33     8.30      8.38     -0.80     -0.17
 34     8.15      7.69      0.50     -0.90
 35     6.85      6.77     -3.10     -1.16
 36     5.35      5.65      0.10     -1.30
 37     4.60      4.22     -1.60     -1.17
 38     2.90      3.02     -1.80     -1.51
harsh braking, plain formulas:  t = 18, 19, 35
harsh braking, smoothed:        t = 18
```

Percepatan yang dihaluskan selama melaju tetap berada dalam sekitar ± 0.4 m/s^2 dari nol, sedangkan yang biasa berayun antara -2.3 dan $+1.6$. Rem keras tetap ditemukan, pada $t = 18$ dengan -3.13 m/s^2, sedikit lebih lunak daripada -3.5 yang sebenarnya karena pencocokan menyebarkannya, tetapi jelas melewati ambang. Dan alarm palsu pada $t = 35$ menjadi -1.16 m/s^2, dekat dengan -1.4 yang sebenarnya: berhenti pelan, persis seperti kata Mas Dimas. **Gambar 9.2** memperlihatkan kecepatan dan kedua kurva percepatan; kurva biasa berkelana melintasi ambang merah, kurva yang dihaluskan melintasinya hanya di tempat rem itu terjadi.



Gambar 9.2 Kecepatan perjalanan Mas Dimas (atas) dan percepatannya (bawah) dari rumus tiga titik biasa dan dari rumus kuadrat terkecil lima titik; hanya kurva yang dihaluskan yang memisahkan rem mendadak sungguhan dari alarm palsu.

Mari periksa satu nilai dengan tangan, sebagaimana semestinya dilakukan pemakai program. Pada $t = 35$ kelima bacaannya 271.6,279.5,287.9,293.2,298.6. Menurut Persamaan (9.8), $a = (2 \times 271.6 - 279.5 - 2 \times 287.9 - 293.2 + 2 \times 298.6)/7 = (543.2 - 279.5 - 575.8 - 293.2 + 597.2)/7 = -8.1/7 = -1.157$ m/s^2, yaitu -1.16 pada tabel.

> **PERINGATAN:** Rumus penghalusan memakai bacaan dari masa depan: pada $t = 18$ ia memerlukan bacaan pada 19 dan 20. Aplikasi yang harus memperingatkan *selagi* pengemudi mengerem tidak dapat menunggu dua detik; ia hanya dapat memakai rumus satu sisi, yang lebih berderau, atau menerima sedikit penundaan. Setiap turunan waktu nyata menghadapi pilihan antara derau dan penundaan ini.

9.8 Memeriksa terhadap Pustaka, dan Pengujian

Dalam pekerjaan nyata Anda jarang menulis rumus beda sendiri; pustaka sudah menyediakannya. Rilis terbaru pada saat buku ini ditulis adalah NumPy 2.5.3, dirilis 6 September 2026 [7], dan SciPy 1.18.1, dirilis 21 Agustus 2026 [8]; keduanya memerlukan Python 3.12 atau yang lebih baru. `np.gradient` milik NumPy menerapkan beda pusat di bagian dalam dan rumus satu sisi di ujung-ujungnya; dengan `edge_order=2` ujung-ujungnya memakai persis rumus tiga titik kita Persamaan (9.5), dan jaraknya boleh berupa bilangan atau larik koordinat yang tidak seragam [9]. SciPy, sejak versi 1.15, memiliki `scipy.differentiate.derivative`, yang menurunkan sebuah *fungsi* (bukan data) dengan barisan beda pusat yang langkahnya makin kecil dan ekstrapolasi ala

Richardson, lalu berhenti bila taksiran galatnya sendiri sudah cukup kecil [10]. **Kode 9.9** memakai keduanya.

Kode 9.9 `compare_numpy.py`: kecepatan perjalanan dari `np.gradient` dan turunan $\sqrt{x}$ dari `scipy.differentiate.derivative`.

```
"""Check this chapter's formulas against NumPy and SciPy."""
import numpy as np
import scipy
from scipy.differentiate import derivative
from speed import load, speeds, accelerations

t, s = load("../data/trip.csv")
h = t[1] - t[0]
v_ours = np.array(speeds(s, h))
v_np = np.gradient(np.array(s), h, edge_order=2)   # same formulas as ours
print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
print(f"largest speed difference, ours vs np.gradient: "
      f"{np.max(np.abs(v_ours - v_np)):.1e} m/s")
a_np = np.gradient(v_np, h, edge_order=2)          # gradient of the gradient
a_ours = np.array(accelerations(s, h))
print(f"a at t = 19 s: ours {a_ours[19]:.2f}, gradient twice {a_np[19]:.2f}")
res = derivative(np.sqrt, 4.0)                     # adaptive, with error estimate
print(f"scipy derivative of sqrt at 4: {res.df:.15f}"
      f" (estimated error {res.error:.1e}, {res.nfev} evaluations)")
```

```
NumPy 2.4.4, SciPy 1.17.1
largest speed difference, ours vs np.gradient: 0.0e+00 m/s
a at t = 19 s: ours -3.60, gradient twice -2.65
scipy derivative of sqrt at 4: 0.250000000000000 (estimated error 8.9e-14, 11 evaluations)
```

Keluaran itu, dari versi yang terpasang pada komputer yang membangun buku ini, memperlihatkan tiga hal. Kecepatan dari `np.gradient(..., edge_order=2)` identik dengan kecepatan kita, sampai bit terakhir. Menerapkan `np.gradient` dua kali *tidak* memberi percepatan kita: pada $t = 19$ hasilnya -2.65, bukan -3.60, karena gradien dari gradien adalah beda kedua dengan langkah $2h$, $(s_{i+2} - 2s_i + s_{i-2})/(4h^2)$, rumus yang lebih lebar dan lebih halus, sehingga rem itu tersebar selama empat detik. Tidak ada jawaban yang salah; keduanya adalah tawar-menawar yang berbeda antara derau dan resolusi, dan Anda perlu tahu mana yang sedang Anda pakai. Terakhir, `derivative` milik SciPy menemukan turunan $\sqrt{x}$ di 4 dengan benar sampai sekitar 15 digit dengan 11 evaluasi, karena tidak pernah membiarkan langkahnya cukup kecil untuk dirusak pembulatan: gagasan Richardson dari Subbab 9.5 dalam bentuk industri.

Edisi Java tidak memiliki pustaka numerik standar, sehingga program Java diperiksa terhadap program Python: setiap pasangan mencetak keluaran yang identik, dan pembangunan buku ini memverifikasinya karakter demi karakter. Folder kode juga berisi pengujian, `test_diff.py` dan `DiffTest.java`, yang memeriksa empat fakta yang harus berlaku apa pun datanya: beda pusat eksak untuk parabola; galat beda maju menjadi separuh bila h dibagi dua; diagonal Richardson mengalahkan beda pusat biasa; dan untuk jarak $s = t^2$ (percepatan tetap 2 m/s^2) penganalisis mengembalikan tepat $v = 2t$ dan $a = 2$ di *setiap* titik, termasuk ujung-ujungnya. Pengujian terakhir memeriksa rumus satu sisi, yang paling mudah keliru, dan pengujian itu berhasil karena semua rumus kita eksak untuk parabola.

9.9 Metode Numerik dalam Praktik: Profitina

Profitina, platform visibilitas AI asal Indonesia, mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin penjawab AI dan membantu pemiliknya memperbaikinya. Salah satu

fiturnya yang sudah berjalan adalah dasbor analitik historis: platform memindai ulang situs setiap merek secara otomatis dengan irama yang bergantung pada jenis langganan, menyimpan skor per minggu, menggambar tren mingguannya, mengekspornya sebagai CSV, dan mengirimkan peringatan. Pertanyaan yang diajukan pemilik merek kepada tren itu adalah pertanyaan tentang turunan: *Apakah skor saya membaik? Seberapa cepat? Apakah baru saja anjlok?*

Perubahan skor dari minggu ke minggu adalah beda hingga dengan $h = 1$ minggu, dan seluruh isi bab ini berlaku untuknya. Minggu terbaru tidak memiliki "minggu depan", sehingga tren terkini mau tidak mau adalah taksiran mundur satu sisi, jenis yang paling kurang akurat, persis seperti pada detik terakhir perjalanan Mas Dimas. Satu pemindaian juga berderau karena hal-hal yang tidak berkaitan dengan kerja sang pemilik, misalnya halaman yang kebetulan lambat merespons, sehingga peringatan "skor turun tajam" yang dihitung dari dua minggu tunggal akan menghasilkan alarm palsu persis seperti yang membuat Mas Dimas kesal. Kelas penyelesaiannya adalah yang dari Subbab 9.7: taksir tren dari pencocokan kuadrat terkecil atas beberapa minggu terakhir, terima bahwa perubahan nyata dikenali sedikit lebih lambat, dan jelaskan tren itu kepada pemilik dengan kata-kata sederhana alih-alih meneriakkan setiap goyangan. Bagaimana Profitina menghitung skor dan peringatannya adalah urusannya sendiri; bentuk masalahnya persis bentuk masalah bab ini.

Bentuk yang sama muncul pada koin utilitas PROFIT milik Profitina, yang emisinya dirancang meluruh seiring waktu. Dasbor yang menampilkan *berapa koin diemisikan per hari* dari total berjalan koin yang telah diemisikan sedang mengambil beda mundur dari sebuah kurva kumulatif, dan kurva yang meluruh adalah kurva yang laju perubahannya sendiri berubah, tempat pilihan antara rumus satu sisi dan rumus berpusat mulai terasa. Seperti biasa pada Profitina, angka-angka memberi saran; pemakailah yang memutuskan.

9.10 Memakainya Sungguhan

- **Kenali derau Anda lebih dulu.** Sebelum menurunkan catatan apa pun, taksir galat satu bacaan: ulangi pengukuran yang semestinya tidak berubah (sepeda motor yang diparkir, tandon yang penuh, ponsel di pengisi daya) dan lihat bagaimana bacaannya bergoyang. Lalu gunakan $\sqrt{2}\,\sigma/(2h)$ dan $\sqrt{6}\,\sigma/h^2$ dari Subbab 9.6 untuk meramalkan derau pada turunan pertama dan kedua Anda *sebelum* memercayainya.
- **Pilih jarak antarbacaan sesuai pertanyaannya.** Baterai ponsel menampilkan persen bulat. Laju habisnya dari bacaan berjarak satu menit sebagian besar adalah derau kuantisasi (perubahan 0 atau 1 per menit); bacaan berjarak sepuluh menit memberi laju yang berguna. Sensor ketinggian air tandon yang membaca sampai sentimeter memberi debit yang berguna dalam hitungan menit, bukan detik. Bila pencatat merekam lebih cepat daripada yang Anda perlukan, pakailah langkah yang lebih lebar atau rumus penghalusan; jangan buang bacaan di antaranya, rata-ratakan.
- **Gunakan rumus berpusat di dalam dan rumus orde dua di ujung.** Jangan pernah mencampur ujung berorde satu dengan nilai dalam berorde dua tanpa menyadarinya; `np.gradient` melakukan persis hal itu kecuali Anda memberikan `edge_order=2`.
- **Tangani bacaan yang hilang dengan jujur.** Ponsel kadang melewatkan bacaan. Jangan berpura-pura jaraknya masih satu detik; gunakan rumus untuk jarak tak seragam dari Soal Latihan 9.6, atau `np.gradient` dengan cap waktu yang sebenarnya.
- **Sesuaikan penganalisisnya.** Untuk tandon air, kolom "jarak" menjadi ketinggian air dalam sentimeter dan turunan pertamanya adalah debit bersih yang masuk; kalikan dengan luas penampang tandon untuk mendapatkan liter per menit. Untuk toko kecil, kolom itu dapat berisi

penjualan kumulatif, dan turunannya adalah penjualan per hari, yang memberi tahu apakah sebuah promosi mengubah sesuatu. Programnya tidak berubah; hanya berkas dan ambangnya yang berubah.

- **Tetap libatkan manusia.** Peringatan yang dihitung dari turunan semestinya menyebutkan apa yang dilihatnya, misalnya "penurunan 12 km/jam dalam satu detik pada pukul 07.41", agar orang yang mengalaminya dapat setuju atau tidak, seperti yang dapat dilakukan Mas Dimas.

9.11 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Membuat h sekecil mungkin.** Di bawah titik optimal Persamaan (9.6), pembulatan mengambil alih, dan turunannya makin buruk, bukan makin baik. Dengan double, pakailah $h \approx 10^{-8}$ untuk beda maju dan $h \approx 10^{-5}$ untuk beda pusat (diskalakan dengan x), atau gunakan ekstrapolasi Richardson dengan langkah yang lebih besar.
- **Memakai beda maju padahal kedua tetangga ada.** Beda pusat sama biayanya dan satu orde lebih akurat. Simpan rumus satu sisi untuk ujung-ujungnya, dan jadikan rumus tiga titik.
- **Lupa kuadrat pada turunan kedua.** Beda kedua dibagi h^2, bukan h atau $2h$. Dengan $h = 1$ s kesalahan ini tidak terlihat; dengan catatan setiap 0,5 s percepatannya menjadi empat kali terlalu kecil.
- **Menurunkan data berderau dua kali tanpa berpikir.** Setiap diferensiasi membagi derau dengan langkah. Taksir deraunya lebih dulu (Subbab 9.10), lalu haluskan (Persamaan (9.8)) atau perlebar langkahnya sampai derau jauh di bawah apa yang ingin Anda deteksi.
- **Memercayai nilai yang mustahil.** Kecepatan negatif di awal perjalanan atau kecepatan tertinggi dari satu bacaan adalah derau. Periksa turunan terhadap apa yang dimungkinkan oleh fisikanya.
- **Menganggap jaraknya seragam.** Bila cap waktunya 14, 15, 16, 18, 19 karena satu bacaan hilang, rumus biasa memberi jawaban keliru pada 16 dan 18. Periksa jaraknya di dalam program, bukan di kepala.
- **Memanggil `np.gradient` dengan ujung bawaan.** Bawaan `edge_order=1` berorde satu di ujung-ujungnya. Ingat pula bahwa gradien dari gradien bukan beda kedua Persamaan (9.4), melainkan rumus yang lebih lebar.

9.12 Ringkasan Bab

- Turunan dari sampel mengganti garis singgung dengan garis potong. Teorema Taylor (Persamaan (9.1)) mengubah gagasan itu menjadi rumus dengan galat yang diketahui: beda maju dan beda mundur berorde satu (Persamaan (9.2)), beda pusat berorde dua (Persamaan (9.3)), demikian pula beda kedua pusat (Persamaan (9.4)).
- Metode koefisien tak tentu menghasilkan rumus untuk titik-titik apa pun, seperti rumus satu sisi orde dua untuk ujung-ujung (Persamaan (9.5)); **Tabel 9.2** menghimpun rumus-rumus yang umum.
- Galat pemotongan turun ketika h turun, galat pembulatan tumbuh seperti $1/h$; totalnya paling kecil pada langkah optimal (Persamaan (9.6)), sekitar 10^{-8} untuk beda maju dan 10^{-5} untuk beda pusat dalam presisi ganda.
- Ekstrapolasi Richardson (Persamaan (9.7)) menggabungkan taksiran dengan langkah berbeda untuk menghapus suku-suku galat, mencapai akurasi tinggi dengan langkah yang cukup besar sehingga pembulatan tidak berbahaya.

- Untuk data hasil pengukuran, "pembulatan"-nya adalah derau pengukuran, diperbesar dengan $1/h$ untuk kecepatan dan $1/h^2$ untuk percepatan. Pada perjalanan Mas Dimas derau itu menimbulkan kecepatan awal negatif, kecepatan tertinggi palsu, dan alarm pengereman mendadak palsu.
- Menurunkan parabola kuadrat terkecil melalui lima titik (Persamaan (9.8)) mengurangi derau pada percepatan lebih dari empat kali, mempertahankan rem keras yang sungguhan, dan menghapus alarm palsu.
- `np.gradient` dengan `edge_order=2` menghasilkan kembali kecepatan kita dengan persis; `scipy.differentiate.derivative` menurunkan fungsi secara adaptif.

9.13 Soal Latihan

1. Dengan tangan, dari bacaan pada $t = 34{,}35{,}36$ (279.5, 287.9, 293.2 m), hitung kecepatan dan percepatan biasa pada $t = 35$. Jelaskan dari sudut galat bacaan bagaimana berhenti pelan dengan perlambatan sekitar -1.42 m/s^2 dapat menghasilkan percepatan ini. Jarak sebenarnya pada saat-saat itu adalah 279.75, 287.54, dan 293.92 m; berapa galat bacaannya, dan bagaimana galat-galat itu terjumlah?
2. Tunjukkan bahwa satu langkah Richardson yang diterapkan pada beda pusat, $(4D(h/2) - D(h))/3$, tepat sama dengan rumus pusat lima titik pada **Tabel 9.2** dengan langkah $h/2$. Berapa orde akurasinya?
3. Untuk $f(x) = e^x$ di $x = 0$, gunakan Persamaan (9.6) dengan $\varepsilon = 2^{-52}$ untuk meramalkan langkah terbaik dan galat terkecil bagi beda maju dan beda pusat. Periksa ramalan Anda dengan sebuah program kecil.
4. Beda maju memiliki deret galat $D(h) = f'(x) + b_1 h + b_2 h^2 + \cdots$. Turunkan kombinasi Richardson yang menghapus suku $b_1 h$, terapkan pada $\sqrt{x}$ di 4 dengan $h = 0.1$ dan $h = 0.05$, lalu bandingkan galatnya dengan galat kedua beda maju itu.
5. Baterai ponsel Mas Dimas menunjukkan 100, 97, 93, 90, 86, 81, dan 77 persen pada menit ke-0, 10, 20, …, 60 berkendara dengan peta menyala. Taksir laju habisnya baterai dalam persen per menit pada setiap bacaan, dengan rumus orde dua di semua titik. Berapa lama sisa 77% itu akan bertahan? Mengapa taksiran satu sisi pada menit ke-60 adalah dasar yang buruk untuk jawaban itu, dan apa dasar yang lebih baik?
6. Pada perjalanan lain ponsel melewatkan bacaan pada $t = 17$, sehingga catatannya berisi waktu 14, 15, 16, 18, 19, 20 dengan jarak 126,1, 137,5, 148,2, 168,7, 174,4, 176,5 m. Turunkan turunan parabola yang melalui tiga titik berjarak tak seragam di titik tengahnya, lalu tulis program dalam Python dan Java yang mencetak kecepatan pada keempat waktu bagian dalam.

9.14 Penyelesaian Soal Latihan

1. Kecepatan: $v = (293.2 - 279.5)/2 = 6.85$ m/s. Percepatan: $a = 293.2 - 2 \times 287.9 + 279.5 = -3.1$ m/s^2, sedikit melewati ambang, seperti pada tabel di Subbab 9.6. Nilai sebenarnya sekitar -1.42, jadi galat bacaan menyumbang sekitar -1.7 m/s^2. Bila ketiga bacaan memiliki galat e_{34}, e_{35}, e_{36}, beda keduanya berubah sebesar $e_{34} - 2e_{35} + e_{36}$, karena rumusnya berbobot $1, -2, 1$. Di sini galatnya $e_{34} = 279.5 - 279.75 = -0.25$, $e_{35} = 287.9 - 287.54 = +0.36$, dan $e_{36} = 293.2 - 293.92 = -0.72$ m, sehingga perubahannya $-0.25 - 0.72 - 0.72 = -1.69$ m/s^2, dan $-1.42 - 1.69 \approx -3.1$. Pola terburuk yang mungkin, bacaan tengah yang kelebihan di antara dua bacaan yang kekurangan, persis itulah yang terjadi; dengan derau sekitar 0,3 m, pola seperti itu, bahkan galat 0,7 m pada satu bacaan, memang tidak biasa tetapi jauh dari langka dalam catatan yang panjang. Rumus lima titik memberi -1.16 (Subbab 9.7.2).

2. Dengan $D(h) = \frac{f(x+h)-f(x-h)}{2h}$ dan $D(h/2) = \frac{f(x+h/2)-f(x-h/2)}{h}$,

$$\frac{4D(h/2) - D(h)}{3} = \frac{8\left(f\left(x+\frac{h}{2}\right) - f\left(x-\frac{h}{2}\right)\right) - (f(x+h) - f(x-h))}{6h}.$$

Dengan menulis $k = h/2$, sehingga $6h = 12k$, ini menjadi $\frac{f(x-2k)-8f(x-k)+8f(x+k)-f(x+2k)}{12k}$, rumus lima titik dengan langkah k. Richardson menghapus suku h^2, dan deret galat beda pusat hanya memuat pangkat genap, sehingga hasilnya berorde empat, $O(k^4)$, sesuai dengan galat $\frac{k^4}{30}f^{(5)}$ pada **Tabel 9.2**.

3. Untuk e^x di 0 semua turunannya dan $f bernilai 1. Beda maju: $h^* = 2\sqrt{\varepsilon} = 2.98 \times 10^{-8}$, dan batas galat di sana $\frac{h^*}{2} + \frac{2\varepsilon}{h^*} = 1.49 \times 10^{-8} + 1.49 \times 10^{-8} \approx 3.0 \times 10^{-8}$. Beda pusat: $h^* = (3\varepsilon)^{1/3} = 8.7 \times 10^{-6}$, dan batasnya $\frac{(h^*)^2}{6} + \frac{\varepsilon}{h^*} \approx 1.3 \times 10^{-11} + 2.5 \times 10^{-11} \approx 3.8 \times 10^{-11}$. Mengubah `diff_steps.py` agar memakai `math.exp` di 0 dengan nilai eksak 1 dan kedua langkah ini memberi galat sebenarnya 1.6×10^{-8} (maju) dan 1.4×10^{-11} (pusat), di bawah batasnya dan berukuran sesuai ramalan. Taksiran kita adalah batas atas, sehingga galat sebenarnya biasanya sedikit lebih kecil.

4. $D(h/2) = f'(x) + b_1 h/2 + b_2 h^2/4 + \cdots$. Maka $2D(h/2) - D(h) = f'(x) - b_2 h^2/2 + \cdots$: kombinasi $2D(h/2) - D(h)$ berorde dua. Dengan nilai beda maju dari Subbab 9.4, $D(0.1) = 0.2484567313$ dan $D(0.05) = 0.2492235950$ (galat 1.54×10^{-3} dan 7.76×10^{-4}), kombinasinya memberi $2 \times 0.2492235950 - 0.2484567313 = 0.2499904587$, dengan galat 9.5×10^{-6}, delapan puluh kali lebih kecil daripada beda maju yang lebih baik. Kombinasi ini sebenarnya adalah rumus maju tiga titik Persamaan (9.5) dengan langkah 0.05.

5. Langkahnya $h = 10$ menit. Di bagian dalam, beda pusat memberi: pada menit ke-10 $(93 - 100)/20 = -0.35$; ke-20 $(90 - 97)/20 = -0.35$; ke-30 $(86 - 93)/20 = -0.35$; ke-40 $(81 - 90)/20 = -0.45$; ke-50 $(77 - 86)/20 = -0.45$ persen per menit. Pada menit ke-0 rumus maju tiga titik memberi $(-3 \times 100 + 4 \times 97 - 93)/20 = -0.25$, dan pada menit ke-60 rumus mundur tiga titik memberi $(3 \times 77 - 4 \times 81 + 86)/20 = -0.35$. Baterai terkuras lebih cepat pada setengah jam kedua, tetapi nilai satu sisi pada menit ke-60, -0.35, mengesankan sebaliknya; nilai itu terdistorsi oleh tampilan persen bulat, yang galatnya hingga setengah persen dikalikan bobot $3, -4, 1$. Dasar yang lebih baik adalah kemiringan kuadrat terkecil atas empat bacaan terakhir (menit ke-30 sampai ke-60: 90, 86, 81, 77). Dengan $\tau = -1.5, -0.5, 0.5, 1.5$ langkah, kemiringannya $\sum \tau s_\tau / \sum \tau^2 = (-135 - 43 + 40.5 + 115.5)/5 = -4.4$ persen per 10 menit, atau -0.44 per menit. Sisa 77% kemudian bertahan sekitar $77/0.44 \approx 175$ menit, hampir tiga jam, bila ia terus berkendara dengan cara yang sama.

6. Misalkan ketiga titik itu $t_0 < t_1 < t_2$ dengan $h_1 = t_1 - t_0$ dan $h_2 = t_2 - t_1$. Dengan koefisien tak tentu (atau dengan menurunkan bentuk Lagrange dari Bab 7), turunan di t_1 dari parabola yang melalui ketiga titik itu adalah

$$f'(t_1) \approx -\frac{h_2}{h_1(h_1+h_2)} f_0 + \frac{h_2 - h_1}{h_1 h_2} f_1 + \frac{h_1}{h_2(h_1+h_2)} f_2,$$

yang tereduksi menjadi beda pusat bila $h_1 = h_2$ dan secara umum berorde dua. **Kode 9.10** dan **Kode 9.11** mengimplementasikannya.

Kode 9.10 `uneven.py`: kecepatan dari bacaan berjarak tak seragam.

```
"""Exercise 9.6: speed from readings that are not evenly spaced (the phone
missed the reading at t = 17 s)."""


def speed_uneven(t, s, i):
    """Derivative of the parabola through points i-1, i, i+1 at t[i]."""
    h1, h2 = t[i] - t[i - 1], t[i + 1] - t[i]
    return (-h2 / (h1 * (h1 + h2)) * s[i - 1]
            + (h2 - h1) / (h1 * h2) * s[i]
            + h1 / (h2 * (h1 + h2)) * s[i + 1])


def main():
    t = [14.0, 15.0, 16.0, 18.0, 19.0, 20.0]
    s = [126.1, 137.5, 148.2, 168.7, 174.4, 176.5]
    for i in range(1, len(t) - 1):
        print(f"t = {t[i]:4.1f} s   v = {speed_uneven(t, s, i):6.2f} m/s")


if __name__ == "__main__":
    main()
```

Kode 9.11 `Uneven.java`: perhitungan yang sama dalam Java.

```
import java.util.Locale;

/** Exercise 9.6: speed from readings that are not evenly spaced (the phone
 *  missed the reading at t = 17 s). */
public class Uneven {
    /** Derivative of the parabola through points i-1, i, i+1 at t[i]. */
    static double speedUneven(double[] t, double[] s, int i) {
        double h1 = t[i] - t[i - 1], h2 = t[i + 1] - t[i];
        return -h2 / (h1 * (h1 + h2)) * s[i - 1]
               + (h2 - h1) / (h1 * h2) * s[i]
               + h1 / (h2 * (h1 + h2)) * s[i + 1];
    }

    public static void main(String[] args) {
        double[] t = {14.0, 15.0, 16.0, 18.0, 19.0, 20.0};
        double[] s = {126.1, 137.5, 148.2, 168.7, 174.4, 176.5};
        for (int i = 1; i < t.length - 1; i++) {
            System.out.println(String.format(Locale.ROOT,
                "t = %4.1f s   v = %6.2f m/s", t[i], speedUneven(t, s, i)));
        }
    }
}
```

Keduanya mencetak:

```
t = 15.0 s   v =  11.05 m/s
t = 16.0 s   v =  10.55 m/s
t = 18.0 s   v =   7.22 m/s
t = 19.0 s   v =   3.90 m/s
```

Pada $t = 16$ rumus itu memberi bobot lebih besar pada bacaan satu detik ke belakang daripada bacaan dua detik ke depan, dan menghasilkan 10,55 m/s; beda pusat biasa, bila dipakai seolah-olah jaraknya seragam, akan memberi $(168.7 - 137.5)/2 = 15.6$ m/s, kecepatan 56 km/jam yang tidak pernah terjadi.

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] B. Fornberg, "Generation of finite difference formulas on arbitrarily spaced grids," *Mathematics of Computation*, vol. 51, no. 184, pp. 699–706, 1988.

[4] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[5] L. F. Richardson, "The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam," *Philosophical Transactions of the Royal Society A*, vol. 210, pp. 307–357, 1911.

[6] A. Savitzky and M. J. E. Golay, "Smoothing and differentiation of data by simplified least squares procedures," *Analytical Chemistry*, vol. 36, no. 8, pp. 1627–1639, 1964.

[7] NumPy Developers, "numpy 2.5.3," Python Package Index, Sep. 6, 2026. [Online]. Available: https://pypi.org/project/numpy/

[8] SciPy Developers, "scipy 1.18.1," Python Package Index, Aug. 21, 2026. [Online]. Available: https://pypi.org/project/scipy/

[9] NumPy Developers, "numpy.gradient," NumPy v2.5 Manual, accessed Sep. 2026. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.gradient.html

[10] SciPy Developers, "scipy.differentiate.derivative," SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.differentiate.derivative.html

Bab 10

Integrasi Numerik: Trapesium, Simpson, Romberg, dan Gauss

Menjumlahkan banyak potongan kecil adalah salah satu hal tertua yang dilakukan manusia dengan bilangan: luas sebidang tanah dari pias-pias, hujan yang turun selama badai dari bacaan setiap sepuluh menit, listrik yang dipakai sebuah mesin dari dayanya setiap menit. Masing-masing adalah sebuah integral, dan bila yang kita miliki hanya hasil pengukuran, atau rumus yang tidak memiliki antiturunan, integral itu harus dihitung secara numerik. Di akhir bab ini Anda akan mampu menurunkan aturan trapesium dan aturan-aturan Simpson dari polinomial interpolasi serta membuktikan suku galatnya; menaksir galat sebuah jawaban dari jawaban itu sendiri dengan membagi dua langkahnya; menghitung luas sebidang tanah yang tidak beraturan dari jarak-jarak hasil pita ukur dalam Java dan Python; memverifikasi orde setiap aturan secara eksperimen; membangun tabel Romberg dari rumus Euler–Maclaurin; menurunkan kuadratur Gauss–Legendre dan menjelaskan mengapa ia hanya memerlukan separuh banyaknya titik; membiarkan rutin Simpson adaptif mencurahkan usahanya di tempat fungsi itu sulit; dan memeriksa setiap hasil terhadap SciPy.

10.1 Seberapa Luas Kebun Itu?

Keluarga Pak Harjono memiliki kebun buah dan durian di tepi sebuah desa di perbukitan. Kebun itu berupa sebidang tanah memanjang di antara jalan desa yang lurus dan sebuah sungai kecil, sepanjang 160 meter menyusuri jalan. Sisi jalan mudah: lurus, dan keluarga itu telah menanam sederet pohon rambutan muda di sepanjangnya. Sisi sungai tidak: tebingnya berkelok, menggembung ke arah perbukitan di sebuah tikungan lalu kembali mendekati jalan di ujung yang jauh.

Keluarga itu memerlukan luasnya karena dua alasan. Yang pertama praktis. Pohon durian ditanam dengan jarak sekitar sepuluh meter ke setiap arah, sehingga setiap pohon memerlukan sekitar seratus meter persegi, dan Anto, anak yang kini mengambil alih kebun, ingin tahu berapa bibit yang harus dipesan untuk bagian tanah yang masih kosong dan berapa banyak kompos yang harus dibawa untuk seluruh kebun. Yang kedua urusan surat-menyurat: surat tanah lama mencantumkan luas yang tidak begitu dipercaya oleh siapa pun di keluarga itu, dan sebelum membayar juru ukur untuk mengukurnya secara resmi, Anto ingin tahu kira-kira berapa jawaban yang semestinya.

Ia memiliki pita ukur 50 meter dan satu Sabtu sore. Bersama sepupunya ia berjalan menyusuri jalan dan, setiap sepuluh meter, merentangkan pita tegak lurus terhadap jalan sampai ke tepi sungai lalu mencatat jaraknya, yang disebut *ordinat* atau jarak tegak (*offset*). Di 0 m jaraknya 38,6 m, di 10 m 40,0 m, dan seterusnya sampai 31,9 m di sudut terjauh, 160 m dari awal: tujuh belas bilangan di buku catatan. Seberapa luas kebun itu?

Inilah masalah klasik juru ukur, dan persis masalah integrasi numerik: luas di bawah kurva $y(x)$ dari $x = 0$ sampai $x = 160$ adalah integral $\int_0^{160} y\,(x)\,dx$, dan kita hanya mengetahui kurva itu di tujuh belas titik. Matematika yang sama mengubah bacaan intensitas hujan menjadi milimeter

curah hujan, bacaan daya ponsel menjadi energi, dan bacaan kecepatan mobil menjadi jarak. Dalam bab ini kita menghitung luas kebun dengan beberapa cara, mempelajari seberapa akurat masing-masing cara dan mengapa, lalu meninggalkan kebun untuk pertanyaan yang harus dijawab setiap metode integrasi: bagaimana mendapatkan akurasi terbesar dari nilai fungsi yang paling sedikit.

10.2 Gagasannya dalam Kata-Kata Sederhana

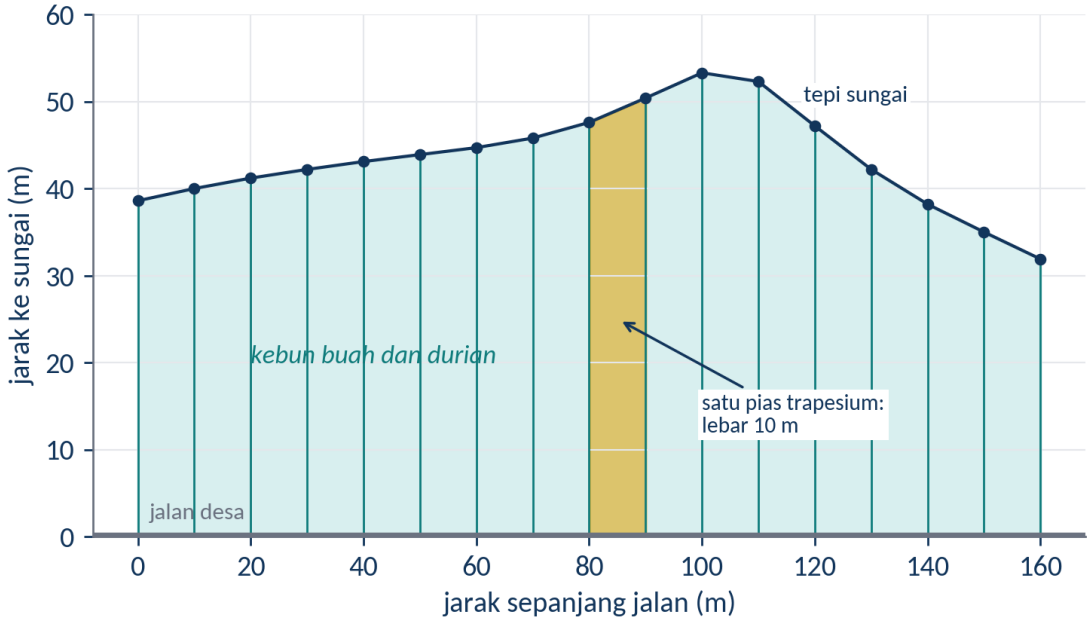
Bayangkan kebun itu dipotong menjadi enam belas pias, masing-masing selebar sepuluh meter, oleh garis-garis dari jalan ke sungai di setiap titik ukur. Setiap pias memiliki dua sisi lurus (jarak tegaknya), alas yang lurus (jalan), dan puncak berupa sepotong pendek tepi sungai. Bila kita berpura-pura puncaknya garis lurus, setiap pias adalah *trapesium*, bangun bersisi empat dengan dua sisi sejajar, dan luasnya adalah lebar kali rata-rata kedua sisi sejajar. Untuk pias pertama:

$$A_1 = 10 \times \frac{38.6 + 40.0}{2} = 10 \times 39.3 = 393.0\ \text{m}^2.$$

Pias kedua memberi $10 \times (40.0 + 41.2)/2 = 406.0\ \text{m}^2$, dan seterusnya. Jumlah keenam belas pias adalah luasnya. Ketika dijumlahkan, setiap jarak tegak di bagian dalam muncul di dua pias, sekali sebagai sisi kanan satu pias dan sekali sebagai sisi kiri pias berikutnya, masing-masing dengan separuh nilainya, sedangkan dua jarak tegak di ujung hanya muncul sekali, dengan separuh nilainya. Jadi seluruh jumlahnya adalah

$$A \approx 10 \times \left(\frac{38.6}{2} + 40.0 + 41.2 + \cdots + 35.0 + \frac{31.9}{2} \right).$$

Itulah **aturan trapesium** (*trapezoidal rule*), dan **Gambar 10.1** memperlihatkan pias-piasnya.



Gambar 10.1 Kebun di antara jalan desa yang lurus dan tepi sungai, dengan tujuh belas jarak tegak yang diukur setiap 10 m; pias yang disorot adalah salah satu dari enam belas trapesium dalam aturan trapesium.

Di tempat tebingnya lurus, trapesium-trapesium itu pas. Di tempat tebingnya berbelok, seperti di sekitar 100 m, puncak yang lurus memotong sebagian kurva atau menambahkan serpihan yang sebenarnya tidak ada, dan itulah galat aturan ini. Gagasan yang lebih baik adalah menggambar puncak yang *melengkung*: ambil pias berpasangan, gambar parabola melalui tiga jarak tegak setiap pasangan, dan gunakan luas di bawah parabola itu. Inilah **aturan Simpson**. Untuk pasangan pias pertama, dari 0 sampai 20 m, hasilnya

$$A_{1,2} \approx \frac{10}{3}(38.6 + 4 \times 40.0 + 41.2) = \frac{10}{3} \times 239.8 = 799.3 \text{ m}^2,$$

sedangkan trapesium memberi $393.0 + 406.0 = 799.0$ m^2. Selisihnya, 0,3 m^2, kecil karena tebing di dekat awal jalan hampir lurus. Di sekitar tikungan kedua aturan akan berselisih jauh lebih besar, dan perselisihan itu, seperti akan kita lihat, justru alat terbaik untuk menaksir galat.

Mengapa bobotnya 1,4,1 dan faktornya $\frac{1}{3}$? Subbab 10.4 menurunkannya. Untuk sekarang perhatikan bahwa jarak tegak di tengah mendapat bobot empat kali bobot ujung-ujungnya, dan bahwa jumlah bobotnya 6, sehingga pada tanah datar dengan semua jarak tegak sama dengan y, rumus itu memberi $\frac{10}{3} \times 6y = 20y$, luas persegi panjang sepanjang 20 m, sebagaimana mestinya.

> **WAWASAN KUNCI:** Setiap aturan dalam bab ini melakukan dua hal yang sama: mengganti kurva yang tidak diketahui dengan kurva sederhana yang melalui titik-titik yang diketahui (garis, parabola, polinomial), lalu mengintegralkan kurva sederhana itu secara eksak. Aturan-aturan itu hanya berbeda dalam kurva sederhana mana yang dipakai dan di mana letak titik-titik yang diketahui. Galatnya adalah luas di antara kurva sebenarnya dan kurva sederhana itu.

10.3 Aturan Trapesium

10.3.1 Penurunan

Misalkan f fungsi pada selang $[a, b]$, dan $h = b - a$. Garis lurus melalui $(a, f(a))$ dan $(b, f(b))$, interpolan linear dari Bab 7, adalah $p(x) = f(a) + \frac{f(b)-f(a)}{h}(x - a)$. Integralnya

$$\int_a^b p\,(x)\,dx = hf(a) + \frac{f(b) - f(a)}{h} \cdot \frac{h^2}{2} = \frac{h}{2}(f(a) + f(b)),$$

yaitu luas trapesium. Untuk mencari galatnya, ekspansikan f di sekitar titik tengah $m = (a + b)/2$ dengan teorema Taylor (Bab 9): $f(x) = f(m) + f'(m)(x - m) + \frac{1}{2}f''(m)(x - m)^2 + \cdots$. Bila diintegralkan pada $[a, b]$, pangkat ganjil $(x - m)$ lenyap karena simetri, dan $\int_a^b (x - m)^2\,dx = h^3/12$, sehingga

$$\int_a^b f\,(x)\,dx = hf(m) + \frac{h^3}{24}f''(m) + O(h^5).$$

Di ujung-ujungnya, $f(a) + f(b) = 2f(m) + \frac{h^2}{4}f''(m) + O(h^4)$, sehingga trapesiumnya $\frac{h}{2}(f(a) + f(b)) = hf(m) + \frac{h^3}{8}f''(m) + O(h^5)$. Dengan mengurangkan,

$$\int_a^b f\,(x)\,dx - \frac{h}{2}(f(a)+f(b)) = \left(\frac{1}{24}-\frac{1}{8}\right)h^3 f''(m) + \cdots = -\frac{h^3}{12}f''(\xi) \qquad (10.1)$$

untuk suatu ξ di $[a,b]$ [1], [2]. Galatnya nol bila $f''=0$: aturan ini eksak untuk garis lurus, sebagaimana mestinya. Bila f melengkung ke atas ($f''>0$), talibusurnya berada di atas kurva dan trapesiumnya terlalu besar; tanda minus menyatakan hal yang sama.

10.3.2 Aturan gabungan

Satu trapesium di atas selang yang panjang terlalu kasar. Bagi $[a,b]$ menjadi n pias yang sama lebar $h=(b-a)/n$, dengan titik $x_i = a+ih$ dan nilai $f_i = f(x_i)$, lalu jumlahkan trapesium-trapesiumnya:

$$T(h) = h\left(\frac{f_0}{2} + f_1 + f_2 + \cdots + f_{n-1} + \frac{f_n}{2}\right), \qquad I - T(h) = -\frac{(b-a)h^2}{12}f''(\xi). \qquad (10.2)$$

Galatnya diperoleh dengan menjumlahkan n galat Persamaan (10.1): n suku berukuran $\frac{h^3}{12}f''$, dan $nh = b-a$, sehingga totalnya $\frac{(b-a)h^2}{12}$ kali suatu nilai rata-rata f'', yaitu $f''(\xi)$ untuk suatu ξ menurut teorema nilai antara. Jadi aturan trapesium gabungan (*composite*) berorde **dua**: membagi dua h membagi galatnya dengan empat.

Rumus galat itu juga alat praktis, karena memberi batas yang *dijamin* bila kita dapat membatasi f''. Untuk $\int_0^1 \frac{dx}{1+x}$, yang nilai eksaknya ln2, turunan kedua $f''(x) = 2/(1+x)^3$ paling besar di $x=0$, yaitu 2. Dengan $n=4$ pias, $h=0.25$, Persamaan (10.2) menjanjikan galat paling besar $\frac{1\times 0.25^2}{12}\times 2 = 0.0104$. Subbab 10.6 akan memperlihatkan galat sebenarnya, 0.0039: di dalam batas, sebagaimana mestinya, dan lebih kecil, karena f'' bernilai 2 hanya di satu ujung dan jauh lebih kecil di tempat lain. Batas memberi tahu kapan Anda boleh berhenti khawatir; taksiran di Subbab 10.5 memberi tahu seberapa besar galatnya kemungkinan besar.

Rumus yang sama juga menjawab pertanyaan perencanaan *berapa pias yang saya perlukan?* sebelum perhitungan apa pun dilakukan. Untuk galat paling besar 10^{-6} pada ln2 kita perlu $\frac{h^2}{12}\times 2 \le 10^{-6}$, yaitu $h \le 0.00245$, atau $n \ge 409$ pias. Rumus galat Simpson Persamaan (10.4) pada subbab berikutnya, dengan $f^{(4)}(x) = 24/(1+x)^5$ yang paling besar 24, hanya memerlukan $\frac{h^4}{180}\times 24 \le 10^{-6}$, yaitu $h \le 0.0523$, atau $n \ge 20$ pias. Dua puluh nilai fungsi alih-alih empat ratus: perbandingan itu, yang dibuat sebelum satu nilai pun dihitung, adalah seluruh alasan untuk memakai aturan berorde lebih tinggi.

Untuk melihat aturan ini bekerja dengan tangan, ambil $n=4$ pias untuk ln2, sehingga $h=0.25$. Nilai $f(x)=1/(1+x)$ di $x = 0{,}0.25{,}0.5{,}0.75{,}1$ adalah 1, 0.8, 0.666667, 0.571429, dan 0.5. Menurut Persamaan (10.2), $T = 0.25\times(0.5+0.8+0.666667+0.571429+0.25) = 0.25\times 2.788095 = 0.697024$, dibandingkan nilai eksak 0.693147: galat 0.0039, di dalam batas 0.0104 yang dihitung di atas. Bobot Simpson pada kelima nilai yang sama memberi $\frac{0.25}{3}(1+4\times 0.8+2\times 0.666667+4\times 0.571429+0.5) = \frac{0.25}{3}\times 8.319048 = 0.693254$, galat 0.0001, empat puluh kali lebih kecil dari aritmetika yang sama.

Masih ada yang perlu dikatakan tentang galat itu, dan inilah kunci metode Romberg. **Rumus Euler–Maclaurin** memberikan seluruh galatnya sebagai deret dalam pangkat genap h:

$$T(h) - I = \frac{h^2}{12}(f'(b) - f'(a)) - \frac{h^4}{720}(f'''(b) - f'''(a)) + \cdots \tag{10.3}$$

Koefisien-koefisiennya hanya bergantung pada turunan di kedua ujung, tidak pada h [2], [3]. Dua akibatnya patut diingat. Pertama, karena galatnya berbentuk $c_1h^2 + c_2h^4 + \cdots$ dengan c_i tetap, ekstrapolasi Richardson (Subbab 9.5) langsung dapat diterapkan; itulah Subbab 10.7. Kedua, untuk fungsi *periodik* yang diintegralkan atas satu periode penuh, $f'(b) = f'(a)$, $f'''(b) = f'''(a)$, dan seterusnya, semua suku lenyap, dan aturan trapesium yang sederhana itu menjadi luar biasa akurat: inilah sebabnya ia menjadi aturan pilihan untuk sinyal periodik.

10.4 Aturan-Aturan Simpson

10.4.1 Aturan Simpson 1/3

Ambil dua pias, dari $x_0 = m - h$ sampai $x_2 = m + h$ dengan titik tengah $x_1 = m$, lalu ganti f dengan parabola yang melalui ketiga titik itu. Alih-alih menuliskan parabolanya, gunakan metode koefisien tak tentu dari Bab 9: cari bobot yang memenuhi

$$\int_{m-h}^{m+h} f\,(x)\,dx \approx h(w_0 f(m-h) + w_1 f(m) + w_2 f(m+h))$$

dan eksak untuk $f = 1$, $f = x - m$, dan $f = (x - m)^2$. Ketiga integralnya $2h$, 0, dan $\frac{2}{3}h^3$, sehingga

$$w_0 + w_1 + w_2 = 2, \qquad -w_0 + w_2 = 0, \qquad w_0 + w_2 = \frac{2}{3}.$$

Maka $w_0 = w_2 = \frac{1}{3}$ dan $w_1 = \frac{4}{3}$: bobot $\frac{h}{3}(1,4,1)$. Sekarang sebuah kejutan kecil: aturan ini juga eksak untuk $(x - m)^3$, karena integral maupun aturannya sama-sama nol untuk fungsi ganjil di sekitar m. Aturan yang dibangun untuk mengintegralkan parabola secara eksak ternyata juga mengintegralkan kubik secara eksak, gratis, persis seperti beda pusat di Bab 9 mendapat satu orde dari kesimetrisan.

Untuk galatnya, ulangi ekspansi di titik tengah dari Subbab 10.3.1 sampai pangkat empat. Integralnya $2hf(m) + \frac{h^3}{3}f''(m) + \frac{h^5}{60}f^{(4)}(m) + \cdots$ dan aturannya $\frac{h}{3}\left(6f(m) + h^2f''(m) + \frac{h^4}{12}f^{(4)}(m) + \cdots\right)$. Suku dalam f dan f'' sama; selisihnya $\left(\frac{1}{60} - \frac{1}{36}\right)h^5f^{(4)} = -\frac{h^5}{90}f^{(4)}$. Menjumlahkan $n/2$ panel seperti itu pada $[a, b]$ dengan n genap menghasilkan **aturan Simpson gabungan**:

$$S(h) = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n), \qquad I - S(h) = -\frac{(b-a)h^4}{180}f^{(4)}(\xi). \tag{10.4}$$

Pola bobot di bagian dalam adalah 4,2,4,2, … ,4: titik di tengah panel mendapat 4, titik yang dipakai bersama dua panel mendapat $1 + 1 = 2$. Aturan Simpson berorde **empat**: membagi dua h

membagi galatnya dengan enam belas, dengan jenis pekerjaan yang sama dengan aturan trapesium. Aturan ini memerlukan banyak pias yang genap. Tujuh belas jarak tegak Anto membentuk enam belas pias, jadi aturan ini dapat dipakai.

Ada hubungan yang rapi antara kedua aturan. Ambil aturan trapesium dengan langkah h dan dengan langkah $2h$ (setiap jarak tegak kedua). Sedikit aljabar memperlihatkan

$$S(h) = \frac{4T(h) - T(2h)}{3}, \tag{10.5}$$

yang merupakan satu langkah ekstrapolasi Richardson yang diterapkan pada Persamaan (10.3): suku h^2 saling menghapus, dan yang tersisa adalah aturan Simpson. Soal Latihan 10.3 meminta Anda memeriksanya.

10.4.2 Aturan Simpson 3/8

Bila banyaknya pias ganjil, misalnya pagar tetangga menghentikan pita ukur di 150 m sehingga tersisa lima belas pias, aturan Simpson 1/3 tidak dapat menutup semuanya. Kubik melalui empat titik yang merentang tiga pias menghasilkan **aturan Simpson 3/8**,

$$\int_{x_0}^{x_3} f(x)\,dx \approx \frac{3h}{8}(f_0 + 3f_1 + 3f_2 + f_3), \qquad \text{galat} = -\frac{3h^5}{80} f^{(4)}(\xi), \tag{10.6}$$

yang juga eksak untuk kubik, dengan konstanta galat sedikit lebih besar daripada dua panel aturan 1/3 [1]. Resep yang lazim untuk banyak pias ganjil adalah aturan 1/3 pada semua pias kecuali tiga yang terakhir dan aturan 3/8 pada ketiga pias itu; Soal Latihan 10.2 melakukan persis hal itu untuk lima belas pias.

10.5 Membangun Kalkulator Kebun

Buku catatan Anto menjadi berkas `data/orchard.csv`: satu baris judul `x_m,offset_m` dan tujuh belas baris seperti `0,38.6` dan `160,31.9`. Seperti catatan perjalanan di Bab 9, berkas dalam folder kode disiapkan untuk bab ini dari kurva sungai yang diketahui, sehingga kita tahu luas sebenarnya, 7026,8 m^2, dan dapat menilai setiap taksiran terhadapnya; Anto, tentu saja, tidak dapat, dan program harus memberi tahu seberapa jauh jawabannya dapat dipercaya tanpa mengetahui kebenarannya.

Program melakukannya dengan menghitung setiap aturan dua kali: sekali dengan ketujuh belas jarak tegak (langkah 10 m) dan sekali dengan setiap jarak tegak kedua (sembilan jarak tegak, langkah 20 m). Selisih kedua jawaban menaksir galatnya. Untuk aturan trapesium, bila $T(h) = I + ch^2$ dan $T(2h) = I + 4ch^2$, maka $T(h) - T(2h) = -3ch^2$, sehingga galat jawaban yang lebih halus adalah

$$I - T(h) \approx \frac{T(h) - T(2h)}{3}, \qquad I - S(h) \approx \frac{S(h) - S(2h)}{15}, \tag{10.7}$$

yang kedua dengan argumen yang sama memakai h^4 dan $16 - 1 = 15$. Program juga menambahkan taksiran galat pada jawaban Simpson, satu langkah Richardson lagi, lalu melaporkan hasilnya dalam meter persegi, dalam hektare, dan sebagai banyaknya pohon durian dengan jarak sepuluh kali sepuluh meter. **Kode 10.1** adalah program lengkapnya.

Kode 10.1 `orchard.py`: luas kebun dengan aturan trapesium dan Simpson, beserta taksiran galat dari membagi dua data.

```
"""Area of the orchard between the road and the river from offsets measured
every 10 m. Usage: python orchard.py [csv file]  (default ../data/orchard.csv)"""
import sys


def load(path):
    xs, ys = [], []
    with open(path) as f:
        next(f)                                  # skip the header line
        for line in f:
            x, y = line.strip().split(",")
            xs.append(float(x))
            ys.append(float(y))
    return xs, ys


def trapezoid(y, h):
    """Composite trapezoidal rule for equally spaced values y."""
    return h * (y[0] / 2 + sum(y[1:-1]) + y[-1] / 2)


def simpson(y, h):
    """Composite Simpson 1/3 rule; needs an even number of intervals."""
    n = len(y) - 1
    if n % 2:
        raise ValueError("Simpson's 1/3 rule needs an even number of intervals")
    odd = sum(y[1:n:2])                          # y1, y3, ..., y(n-1)
    even = sum(y[2:n:2])                         # y2, y4, ..., y(n-2)
    return h / 3 * (y[0] + 4 * odd + 2 * even + y[n])


def main(args):
    path = args[0] if args else "../data/orchard.csv"
    x, y = load(path)
    h = x[1] - x[0]
    print(f"{len(y)} offsets, every {h:.0f} m along {x[-1] - x[0]:.0f} m of road")
    t1, t2 = trapezoid(y, h), trapezoid(y[::2], 2 * h)
    s1, s2 = simpson(y, h), simpson(y[::2], 2 * h)
    print("rule        every 10 m  every 20 m  error estimate")
    print(f"trapezoid   {t1:10.1f}  {t2:10.1f}  {(t1 - t2) / 3:14.1f}")
    print(f"Simpson     {s1:10.1f}  {s2:10.1f}  {(s1 - s2) / 15:14.1f}")
    best = s1 + (s1 - s2) / 15                   # Richardson on Simpson
    print(f"area {best:.0f} m2 = {best / 10000:.2f} ha,"
          f" room for about {best / 100:.0f} durian trees at 10 m x 10 m")


if __name__ == "__main__":
    main(sys.argv[1:])
```

Kode 10.1 (Lanjutan) *orchard.py: luas kebun dengan aturan trapesium dan Simpson, beserta taksiran galat dari membagi dua data.*

Irisan `y[::2]` mengambil setiap jarak tegak kedua, sedangkan `y[1:n:2]` dan `y[2:n:2]` memilih titik dalam yang ganjil dan genap untuk bobot Simpson 4 dan 2. Fungsi itu memunculkan galat alih-alih diam-diam mengembalikan jawaban keliru bila banyaknya pias ganjil. **Kode 10.2** adalah versi Java-nya, dengan fungsi bantu kecil `everyOther` untuk irisan tersebut.

Kode 10.2 `Orchard.java`: kalkulator yang sama dalam Java.

```java
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Locale;

/** Area of the orchard between the road and the river from offsets measured
 *  every 10 m. Usage: java Orchard [csv file]  (default ../data/orchard.csv) */
public class Orchard {
    /** Returns {positions, offsets}. */
    static double[][] load(String path) throws IOException {
        List<String> lines = Files.readAllLines(Path.of(path));
        int n = lines.size() - 1;                       // the first line is a header
        double[] x = new double[n], y = new double[n];
        for (int i = 0; i < n; i++) {
            String[] parts = lines.get(i + 1).trim().split(",");
            x[i] = Double.parseDouble(parts[0]);
            y[i] = Double.parseDouble(parts[1]);
        }
        return new double[][] {x, y};
    }

    /** Composite trapezoidal rule for equally spaced values y. */
    static double trapezoid(double[] y, double h) {
        double inner = 0;
        for (int i = 1; i < y.length - 1; i++) {
            inner += y[i];
        }
        return h * (y[0] / 2 + inner + y[y.length - 1] / 2);
    }

    /** Composite Simpson 1/3 rule; needs an even number of intervals. */
    static double simpson(double[] y, double h) {
        int n = y.length - 1;
        if (n % 2 != 0) {
            throw new IllegalArgumentException(
                "Simpson's 1/3 rule needs an even number of intervals");
        }
        double odd = 0, even = 0;
        for (int i = 1; i < n; i += 2) {
            odd += y[i];                                // y1, y3, ..., y(n-1)
        }
        for (int i = 2; i < n; i += 2) {
            even += y[i];                               // y2, y4, ..., y(n-2)
        }
        return h / 3 * (y[0] + 4 * odd + 2 * even + y[n]);
```

Kode 10.2 (Lanjutan) *Orchard.java: kalkulator yang sama dalam Java.*

```
    }

    /** Every second value: y0, y2, y4, ... */
    static double[] everyOther(double[] y) {
        double[] r = new double[(y.length + 1) / 2];
        for (int i = 0; i < r.length; i++) {
            r[i] = y[2 * i];
        }
        return r;
    }

    public static void main(String[] args) throws IOException {
        double[][] data = load(args.length > 0 ? args[0] : "../data/orchard.csv");
        double[] x = data[0], y = data[1];
        double h = x[1] - x[0];
        System.out.println(String.format(Locale.ROOT,
            "%d offsets, every %.0f m along %.0f m of road",
            y.length, h, x[x.length - 1] - x[0]));
        double t1 = trapezoid(y, h), t2 = trapezoid(everyOther(y), 2 * h);
        double s1 = simpson(y, h), s2 = simpson(everyOther(y), 2 * h);
        System.out.println("rule        every 10 m  every 20 m  error estimate");
        System.out.println(String.format(Locale.ROOT,
            "trapezoid   %10.1f  %10.1f  %14.1f", t1, t2, (t1 - t2) / 3));
        System.out.println(String.format(Locale.ROOT,
            "Simpson     %10.1f  %10.1f  %14.1f", s1, s2, (s1 - s2) / 15));
        double best = s1 + (s1 - s2) / 15;            // Richardson on Simpson
        System.out.println(String.format(Locale.ROOT,
            "area %.0f m2 = %.2f ha, room for about %.0f durian trees"
            + " at 10 m x 10 m", best, best / 10000, best / 100));
    }
}
```

Keduanya mencetak:

```
17 offsets, every 10 m along 160 m of road
rule        every 10 m  every 20 m  error estimate
trapezoid       7023.5      7011.0             4.2
Simpson         7027.7      7039.3            -0.8
area 7027 m2 = 0.70 ha, room for about 70 durian trees at 10 m x 10 m
```

Bacalah dengan luas sebenarnya, 7026,8 m^2, dalam benak. Aturan trapesium dengan semua jarak tegak kurang 3,3 m^2, dan taksiran galatnya berkata “tambahkan sekitar 4,2”: peringatan yang baik, dengan tanda dan ukuran yang tepat. Aturan Simpson kelebihan 0,9 m^2, dan taksirannya berkata “kurangi sekitar 0,8”. Jawaban yang dikoreksi, 7026,9 m^2, berada dalam sepersepuluh meter persegi dari kebenaran, jauh lebih dekat daripada yang pantas diharapkan dari akurasi pita ukur itu sendiri. Perhatikan pula bahwa $T(h) + \frac{T(h)-T(2h)}{3} = 7023.5 + 4.2 = 7027.7$ tepat sama dengan jawaban Simpson, seperti dijanjikan Persamaan (10.5).

Dari mana sisa galatnya? Lihat pasangan pias di sekitar tikungan. Dari 100 sampai 120 m jarak tegaknya 53,3, 52,3, dan 47,2 m. Trapesium memberi $10 \times (26.65 + 52.3 + 23.6) = 1025.5$ m^2, Simpson memberi $\frac{10}{3}(53.3 + 4 \times 52.3 + 47.2) = 1032.3$ m^2, dan luas sebenarnya potongan itu 1031,3 m^2. Tebingnya menggembung ke luar di sana, sehingga puncak lurus trapesium memotong hampir enam meter persegi, sedangkan parabola mengikuti gembungan itu sampai selisih kurang dari satu. Di bentang pertama yang lurus, dari 0 sampai 20 m, kedua aturan hanya berselisih 0,3 m^2, seperti kita lihat di Subbab 10.2.

Tetapi ada sumber galat kedua yang tidak dapat dihapus oleh aturan apa pun: jarak tegak itu sendiri. Masing-masing dicatat sampai 0,1 m terdekat, sehingga masing-masing membawa galat

hingga 0,05 m. Galat 0,05 m pada satu jarak tegak mengubah luas hingga $0.05 \times 10 = 0.5$ m² (sedikit lebih dengan bobot Simpson 4); tersebar pada tujuh belas jarak tegak dengan tanda galat yang acak, totalnya biasanya sekitar satu meter persegi. Pada potongan pertama, misalnya, luas sebenarnya 798,8 m², sedangkan kedua aturan, yang diberi jarak tegak yang sudah dibulatkan, memberi sekitar 799. Galat pemotongan Simpson untuk seluruh kebun, sekitar satu meter persegi, dengan demikian sebesar galat pengukurannya. Beralih ke Romberg atau Gauss pada data ini hanya akan memoles aritmetika di bawah tingkat ketelitian pita ukur; perbaikan berikutnya harus datang dari pengukuran yang lebih baik, bukan dari rumus yang lebih baik. Keseimbangan antara galat metode dan galat data ini adalah kembaran integrasi dari langkah optimal di Bab 9.

Bagi Anto kesimpulannya sederhana: kebun itu seluas 0,70 hektare, lebih atau kurang beberapa meter persegi, dengan ruang untuk sekitar tujuh puluh pohon durian pada jarak tanam yang lazim. Bila surat tanah lama menyebut 0,6 hektare, ada baiknya meminta juru ukur memeriksa dengan cermat; bila menyebut 0,7, keluarga itu boleh tenang. Bagaimanapun, pengukuran resmi adalah wewenang juru ukur; perhitungan ini untuk perencanaan.

10.6 Memeriksa Ordenya

Orde yang diklaim dalam Persamaan (10.2) dan Persamaan (10.4) layak kita lihat dengan mata sendiri, pada integral yang nilai eksaknya kita ketahui. Ambil

$$\int_0^1 \frac{dx}{1+x} = \ln 2 = 0.693147\ldots$$

lalu hitung kedua aturan untuk $n = 2,4,8,\ldots,256$ pias. Bila sebuah aturan berorde p, menggandakan n membagi galatnya dengan 2^p: dengan 4 untuk aturan trapesium dan dengan 16 untuk aturan Simpson. **Kode 10.3** mencetak setiap galat dan rasionya terhadap galat sebelumnya. Nilai eksaknya ditulis sebagai konstanta presisi ganda `0.6931471805599453`, agar program Java dan Python membandingkan terhadap bilangan yang sama.

Kode 10.3 `quadrature.py`: aturan trapesium dan Simpson gabungan untuk integral yang diketahui, beserta rasio galat berturutan.

```python
"""Composite trapezoid and Simpson rules on the integral of 1/(1+x) from
0 to 1, which equals ln 2. Doubling n shows the order of each rule."""

LN2 = 0.6931471805599453                          # ln 2 to double precision


def f(x):
    return 1.0 / (1.0 + x)


def trapezoid(f, a, b, n):
    h = (b - a) / n
    inner = sum(f(a + i * h) for i in range(1, n))
    return h * (f(a) / 2 + inner + f(b) / 2)


def simpson(f, a, b, n):
    h = (b - a) / n                               # n must be even
    odd = sum(f(a + i * h) for i in range(1, n, 2))
    even = sum(f(a + i * h) for i in range(2, n, 2))
    return h / 3 * (f(a) + 4 * odd + 2 * even + f(b))


def main():
    print("   n   trapezoid error  ratio    Simpson error  ratio")
    old_t = old_s = None
    n = 2
    while n <= 256:
        et = abs(trapezoid(f, 0.0, 1.0, n) - LN2)
        es = abs(simpson(f, 0.0, 1.0, n) - LN2)
        rt = f"{old_t / et:6.1f}" if old_t else "      "
        rs = f"{old_s / es:6.1f}" if old_s else "      "
        print(f"{n:4d}  {et:15.3e}  {rt}  {es:15.3e}  {rs}")
        old_t, old_s = et, es
        n *= 2


if __name__ == "__main__":
    main()
```

Kode 10.4 `Quadrature.java`: pemeriksaan yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Composite trapezoid and Simpson rules on the integral of 1/(1+x) from
 *  0 to 1, which equals ln 2. Doubling n shows the order of each rule. */
public class Quadrature {
    static final double LN2 = 0.6931471805599453;    // ln 2 to double precision

    static double f(double x) {
        return 1.0 / (1.0 + x);
    }

    static double trapezoid(DoubleUnaryOperator f, double a, double b, int n) {
        double h = (b - a) / n, inner = 0;
        for (int i = 1; i < n; i++) {
            inner += f.applyAsDouble(a + i * h);
        }
        return h * (f.applyAsDouble(a) / 2 + inner + f.applyAsDouble(b) / 2);
    }

    static double simpson(DoubleUnaryOperator f, double a, double b, int n) {
        double h = (b - a) / n, odd = 0, even = 0;   // n must be even
        for (int i = 1; i < n; i += 2) {
            odd += f.applyAsDouble(a + i * h);
        }
        for (int i = 2; i < n; i += 2) {
            even += f.applyAsDouble(a + i * h);
        }
        return h / 3 * (f.applyAsDouble(a) + 4 * odd + 2 * even
                        + f.applyAsDouble(b));
    }

    public static void main(String[] args) {
        System.out.println("   n   trapezoid error  ratio    Simpson error  ratio");
        double oldT = 0, oldS = 0;
        for (int n = 2; n <= 256; n *= 2) {
            double et = Math.abs(trapezoid(Quadrature::f, 0.0, 1.0, n) - LN2);
            double es = Math.abs(simpson(Quadrature::f, 0.0, 1.0, n) - LN2);
            String rt = oldT > 0 ? String.format(Locale.ROOT, "%6.1f", oldT / et)
                                 : "      ";
            String rs = oldS > 0 ? String.format(Locale.ROOT, "%6.1f", oldS / es)
                                 : "      ";
            System.out.println(String.format(Locale.ROOT,
                "%4d  %15.3e  %s  %15.3e  %s", n, et, rt, es, rs));
            oldT = et;
            oldS = es;
```

Kode 10.4 (Lanjutan) *Quadrature.java: pemeriksaan yang sama dalam Java.*

```java
        }
    }
}
```

Kode 10.4 memberi tabel yang sama:

```
   n   trapezoid error  ratio    Simpson error  ratio
   2       1.519e-02               1.297e-03
   4       3.877e-03     3.9       1.068e-04    12.1
   8       9.747e-04     4.0       7.350e-06    14.5
  16       2.440e-04     4.0       4.723e-07    15.6
  32       6.103e-05     4.0       2.973e-08    15.9
  64       1.526e-05     4.0       1.862e-09    16.0
 128       3.815e-06     4.0       1.164e-10    16.0
 256       9.537e-07     4.0       7.275e-12    16.0
```

Rasio trapesium bernilai 4,0 sejak awal; rasio Simpson naik sampai 16,0 begitu h cukup kecil sehingga suku galat terdepan mendominasi. Angka-angkanya juga cocok dengan Persamaan (10.3): dengan $f'(x) = -1/(1+x)^2$, $f'(1) - f'(0) = 0.75$, dan untuk $n = 2$ ($h = 0.5$) suku pertamanya $\frac{0.25}{12} \times 0.75 = 0.0156$, dibandingkan galat sebenarnya 0.0152. Untuk pekerjaan yang sama, Simpson jauh lebih unggul: 257 nilai fungsi memberi aturan trapesium galat 10^{-6} dan aturan Simpson 7×10^{-12}.

Uji rasio ini juga alat pencari kesalahan terbaik untuk program kuadratur yang Anda tulis sendiri. Aturan trapesium yang rasionya mantap di 2, bukan 4, hampir pasti kehilangan separuh bobot di kedua ujungnya; aturan Simpson yang rasionya mantap di 2 tertukar bobot 4 dan 2-nya. Bobot yang keliru sering masih memberi jawaban yang tampak masuk akal sampai tiga digit, tetapi tidak pernah memberi rasio yang benar.

10.7 Integrasi Romberg

Deret galat Persamaan (10.3) hanya memuat pangkat genap h, dengan koefisien yang tidak bergantung pada h. Persis itulah keadaan di Subbab 9.5, dan tabel Richardson dapat diterapkan tanpa perubahan. Misalkan $R_{i,0} = T(h/2^i)$ adalah aturan trapesium dengan 2^i pias. Maka

$$R_{i,j} = \frac{4^j R_{i,j-1} - R_{i-1,j-1}}{4^j - 1}, \qquad j = 1, \dots, i, \tag{10.8}$$

bergalat $O(h^{2j+2})$. Kolom ekstrapolasi pertama adalah aturan Simpson, menurut Persamaan (10.5); kolom kedua ternyata aturan Boole, berorde enam; dan kolom-kolom berikutnya tidak bernama tetapi terus bertambah dua orde. Skema ini diusulkan Werner Romberg pada 1955 [4] dan disebut **integrasi Romberg**.

Satu cara lagi membuatnya murah. Ketika banyaknya pias digandakan, semua titik lama dipakai ulang, dan hanya titik-titik tengah yang baru yang harus dievaluasi:

$$T\left(\frac{h}{2}\right) = \frac{1}{2}T(h) + \frac{h}{2}\sum_{k \text{ ganjil}} f\left(a + k\frac{h}{2}\right).$$

Jadi seluruh tabel dengan lima baris, 1,2,4,8,16 pias, hanya memerlukan 17 nilai fungsi dari baris terhalus. **Kode 10.5** membangunnya untuk ln2.

Kode 10.5 `romberg.py`: tabel Romberg, yang hanya menambahkan titik tengah baru pada setiap baris.

```python
"""Romberg integration: trapezoid values with h, h/2, h/4, ... improved by
Richardson extrapolation. Test: the integral of 1/(1+x) from 0 to 1 is ln 2."""
from quadrature import f, LN2


def romberg(f, a, b, levels):
    r = [[(b - a) / 2 * (f(a) + f(b))]]           # R[0][0]: one trapezoid
    evals = 2
    for i in range(1, levels):
        n = 2 ** i                                # intervals on this row
        h = (b - a) / n
        mids = sum(f(a + k * h) for k in range(1, n, 2))   # only new points
        evals += n // 2
        row = [r[i - 1][0] / 2 + h * mids]        # trapezoid with n intervals
        for j in range(1, i + 1):
            factor = 4 ** j
            row.append((factor * row[j - 1] - r[i - 1][j - 1]) / (factor - 1))
        r.append(row)
    return r, evals


def main():
    table, evals = romberg(f, 0.0, 1.0, 5)
    for i, row in enumerate(table):
        print(f"n = {2 ** i:2d}  " + "  ".join(f"{v:.12f}" for v in row))
    best = table[-1][-1]
    print(f"R[4][4] = {best:.15f}, error {abs(best - LN2):.1e},"
          f" {evals} function values")


if __name__ == "__main__":
    main()
```

Kode 10.6 `Romberg.java`: tabel yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Romberg integration: trapezoid values with h, h/2, h/4, ... improved by
 *  Richardson extrapolation. Test: the integral of 1/(1+x) from 0 to 1 is ln 2. */
public class Romberg {
    static int evals;

    static double[][] romberg(DoubleUnaryOperator f, double a, double b,
                              int levels) {
        double[][] r = new double[levels][];
        r[0] = new double[] {(b - a) / 2
                             * (f.applyAsDouble(a) + f.applyAsDouble(b))};
        evals = 2;
        for (int i = 1; i < levels; i++) {
            int n = 1 << i;                          // intervals on this row
            double h = (b - a) / n, mids = 0;
            for (int k = 1; k < n; k += 2) {         // only the new points
                mids += f.applyAsDouble(a + k * h);
            }
            evals += n / 2;
            r[i] = new double[i + 1];
            r[i][0] = r[i - 1][0] / 2 + h * mids;    // trapezoid, n intervals
            for (int j = 1; j <= i; j++) {
                double factor = Math.pow(4, j);
                r[i][j] = (factor * r[i][j - 1] - r[i - 1][j - 1]) / (factor - 1);
            }
        }
        return r;
    }

    public static void main(String[] args) {
        double[][] table = romberg(Quadrature::f, 0.0, 1.0, 5);
        for (int i = 0; i < table.length; i++) {
            StringBuilder line = new StringBuilder(
                String.format(Locale.ROOT, "n = %2d", 1 << i));
            for (double v : table[i]) {
                line.append(String.format(Locale.ROOT, "  %.12f", v));
            }
            System.out.println(line);
        }
        double best = table[4][4];
        System.out.println(String.format(Locale.ROOT,
            "R[4][4] = %.15f, error %.1e, %d function values",
            best, Math.abs(best - Quadrature.LN2), evals));
    }
```

Kode 10.6 (Lanjutan) *Romberg.java: tabel yang sama dalam Java.*

```java
}
```

Kode 10.6 mencetak angka yang sama:

```
n =  1  0.750000000000
n =  2  0.708333333333  0.694444444444
n =  4  0.697023809524  0.693253968254  0.693174603175
n =  8  0.694121850372  0.693154530655  0.693147901481  0.693147477645
n = 16  0.693391202208  0.693147652819  0.693147194297  0.693147183072  0.693147181917
R[4][4] = 0.693147181916745, error 1.4e-09, 17 function values
```

Kolom pertama merayap menuju 0.693147: aturan trapesium dengan 16 pias masih keliru pada desimal keempat. Diagonalnya melesat ke sana: $R_{4,4}$ benar sampai delapan desimal dengan 17 nilai fungsi yang sama yang memberi aturan Simpson galat 4.7×10^{-7} pada tabel di Subbab 10.6. Metode Romberg adalah pilihan alami bila Anda dapat mengevaluasi f di mana saja, f mulus, dan Anda menginginkan banyak digit dengan murah.

Dapatkah Anto memakainya? Hampir. Tujuh belas jarak tegaknya tepat $2^4 + 1$ titik yang diperlukan Romberg untuk empat kali pembagian dua, dan fungsi `romb` milik SciPy (Subbab 10.10) menerima data seperti itu. Tetapi ekstrapolasi mengandaikan deret galat Persamaan (10.3), yang memerlukan kurva yang mulus, sedangkan jarak tegak hasil pita ukur tidak pasti beberapa sentimeter; kolom-kolom yang lebih tinggi memperbesar galat kecil itu, seperti yang selalu dilakukan Richardson. Untuk data hasil pengukuran, satu atau dua langkah ekstrapolasi, seperti di Subbab 10.5, adalah batas yang wajar.

10.8 Kuadratur Gauss: Memilih Tempat Mengukur

10.8.1 Gagasannya

Semua aturan sejauh ini memakai titik yang berjarak sama dan hanya memilih bobotnya. Misalkan Anto bebas memilih *di mana* ia mengukur jarak tegak, tidak setiap sepuluh meter tetapi di tempat-tempat yang dipilih dengan cerdik. Dengan n jarak tegak ia lalu memiliki $2n$ bilangan untuk dipilih: n posisi dan n bobot. Polinomial berderajat $2n-1$ memiliki $2n$ koefisien, sehingga ada harapan akan aturan yang eksak untuk setiap polinomial sampai derajat $2n-1$, hampir dua kali yang dimungkinkan titik berjarak sama. Carl Friedrich Gauss memperlihatkan pada 1814 bahwa harapan itu benar, dan aturan-aturan yang dihasilkan disebut **kuadratur Gauss** [2], [3].

Pada selang baku $[-1,1]$, cari dua titik $\pm t$ dan bobot yang sama w (kesimetrisan membuat keduanya sama) sehingga $\int_{-1}^{1} f(x)\,dx \approx wf(-t) + wf(t)$. Pangkat ganjil terintegralkan secara eksak karena simetri. Untuk $f = 1$ kita perlu $2w = 2$, jadi $w = 1$; untuk $f = x^2$ kita perlu $2t^2 = \frac{2}{3}$, jadi $t = 1/\sqrt{3} \approx 0.57735$. **Aturan Gauss–Legendre dua titik**

$$\int_{-1}^{1} f(x)\,dx \approx f\left(-\frac{1}{\sqrt{3}}\right) + f\left(\frac{1}{\sqrt{3}}\right) \tag{10.9}$$

eksak untuk $1, x, x^2$, dan x^3: dua nilai fungsi mengintegralkan setiap kubik secara eksak, sesuatu yang memerlukan tiga nilai pada aturan Simpson.

10.8.2 Polinomial Legendre dan aturan umumnya

Untuk n titik syarat-syaratnya menjadi sistem nonlinear yang tidak menyenangkan untuk diselesaikan langsung, tetapi jawabannya indah. Titik-titik terbaik adalah n akar **polinomial Legendre** $P_n(x)$, polinomial yang didefinisikan oleh $P_0 = 1$, $P_1 = x$, dan

$$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x),$$

sehingga, misalnya, $P_2(x) = \frac{1}{2}(3x^2 - 1)$ berakar $\pm 1/\sqrt{3}$, tepat titik-titik yang ditemukan di atas, dan $P_3(x) = \frac{1}{2}(5x^3 - 3x)$ berakar 0 dan $\pm\sqrt{3/5}$. Bobotnya $w_i = \frac{2}{(1-t_i^2)[P_n'(t_i)]^2}$. Alasannya, singkatnya, adalah bahwa polinomial Legendre saling *ortogonal*: $\int_{-1}^{1} P_n(x)q(x)\,dx = 0$ untuk setiap polinomial q berderajat kurang dari n. Bagilah sembarang polinomial f berderajat $2n-1$ dengan P_n: $f = qP_n + r$ dengan q dan r berderajat kurang dari n. Integral qP_n bernilai nol karena ortogonalitas, dan aturannya pun memberi nol untuknya, karena P_n lenyap di titik-titik simpulnya. Sisa r berderajat kurang dari n dan diintegralkan secara eksak oleh aturan interpolasi apa pun pada n titik. Maka f diintegralkan secara eksak [2]. Galat aturan n titik pada $[a, b]$ adalah

$$I - G_n = \frac{(b-a)^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} f^{(2n)}(\xi),$$

yang turun sangat cepat bersama n untuk fungsi yang mulus. Dalam praktik titik simpul dan bobotnya dihitung sebagai nilai eigen dan komponen vektor eigen sebuah matriks tridiagonal kecil, dengan algoritma Golub–Welsch [5]; nilai eigen adalah pokok bahasan Bab 12. Sampai lima titik, keduanya memiliki bentuk tertutup, dan itulah yang dipakai program kita.

Integral pada $[a, b]$ dipetakan ke $[-1,1]$ dengan $x = \frac{a+b}{2} + \frac{b-a}{2} t$, sehingga $dx = \frac{b-a}{2} dt$:

$$\begin{aligned} \int_a^b f(x)\,dx &= \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{a+b}{2} + \frac{b-a}{2} t\right) dt \\ &\approx \frac{b-a}{2} \sum_{i=1}^{n} w_i f\left(\frac{a+b}{2} + \frac{b-a}{2} t_i\right). \end{aligned} \tag{10.10}$$

Hitungan tangan untuk ln2 dengan dua titik: simpul ± 0.57735 dipetakan ke $x = 0.5 \pm 0.5 \times 0.57735$, yaitu 0.211325 dan 0.788675; di sana $f = 1/(1+x)$ bernilai 0.825542 dan 0.559073; jumlahnya kali $\frac{1}{2}$ adalah 0.692308, sudah benar tiga digit dari dua nilai fungsi. (Hasil eksak aturan dua titik di sini adalah $9/13$.)

10.8.3 Programnya

Kode 10.7 menyimpan titik simpul dan bobot untuk $n = 1, \ldots, 5$ dalam sebuah kamus, dibangun dari bentuk tertutupnya dengan akar kuadrat sehingga tidak ada yang perlu disalin dari tabel, menerapkan Persamaan (10.10), dan mencetak galatnya di samping galat aturan trapesium dan Simpson dengan banyak nilai fungsi yang sama.

Kode 10.7 `gauss.py`: kuadratur Gauss–Legendre dengan satu sampai lima titik.

```
"""Gauss-Legendre quadrature with 1 to 5 points on the integral of 1/(1+x)
from 0 to 1 (= ln 2), next to the trapezoid and Simpson rules that use the
same number of function values."""
import math
from quadrature import f, LN2, trapezoid, simpson

S = math.sqrt
NODES = {   # points and weights on [-1, 1]
    1: [(0.0, 2.0)],
    2: [(-1 / S(3), 1.0), (1 / S(3), 1.0)],
    3: [(-S(3 / 5), 5 / 9), (0.0, 8 / 9), (S(3 / 5), 5 / 9)],
    4: [(-S(3 / 7 + 2 / 7 * S(6 / 5)), (18 - S(30)) / 36),
        (-S(3 / 7 - 2 / 7 * S(6 / 5)), (18 + S(30)) / 36),
        (S(3 / 7 - 2 / 7 * S(6 / 5)), (18 + S(30)) / 36),
        (S(3 / 7 + 2 / 7 * S(6 / 5)), (18 - S(30)) / 36)],
    5: [(-S(5 + 2 * S(10 / 7)) / 3, (322 - 13 * S(70)) / 900),
        (-S(5 - 2 * S(10 / 7)) / 3, (322 + 13 * S(70)) / 900),
        (0.0, 128 / 225),
        (S(5 - 2 * S(10 / 7)) / 3, (322 + 13 * S(70)) / 900),
        (S(5 + 2 * S(10 / 7)) / 3, (322 - 13 * S(70)) / 900)],
}


def gauss(f, a, b, n):
    """Map [-1, 1] onto [a, b]: x = mid + half * t, dx = half * dt."""
    mid, half = (a + b) / 2, (b - a) / 2
    return half * sum(w * f(mid + half * t) for t, w in NODES[n])


def main():
    print("points  Gauss-Legendre   error    trapezoid error  Simpson error")
    for n in range(1, 6):
        g = gauss(f, 0.0, 1.0, n)
        line = f"{n:6d}  {g:.12f}  {abs(g - LN2):.1e}"
        if n > 1:                                  # n points = n - 1 intervals
            t = trapezoid(f, 0.0, 1.0, n - 1)
            line += f"  {abs(t - LN2):15.1e}"
        if n % 2 and n > 1:                        # Simpson needs n odd
            s = simpson(f, 0.0, 1.0, n - 1)
            line += f"  {abs(s - LN2):13.1e}"
        print(line)


if __name__ == "__main__":
    main()
```

Kode 10.8 `Gauss.java`: aturan yang sama dalam Java, dengan titik simpul dalam larik tiga dimensi.

```
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

/** Gauss-Legendre quadrature with 1 to 5 points on the integral of 1/(1+x)
 *  from 0 to 1 (= ln 2), next to the trapezoid and Simpson rules that use the
 *  same number of function values. */
public class Gauss {
    static double s(double x) {
        return Math.sqrt(x);
    }

    /** NODES[n] holds {point, weight} pairs on [-1, 1]. */
    static final double[][][] NODES = {
        {},
        {{0.0, 2.0}},
        {{-1 / s(3), 1.0}, {1 / s(3), 1.0}},
        {{-s(3.0 / 5), 5.0 / 9}, {0.0, 8.0 / 9}, {s(3.0 / 5), 5.0 / 9}},
        {{-s(3.0 / 7 + 2.0 / 7 * s(6.0 / 5)), (18 - s(30)) / 36},
         {-s(3.0 / 7 - 2.0 / 7 * s(6.0 / 5)), (18 + s(30)) / 36},
         {s(3.0 / 7 - 2.0 / 7 * s(6.0 / 5)), (18 + s(30)) / 36},
         {s(3.0 / 7 + 2.0 / 7 * s(6.0 / 5)), (18 - s(30)) / 36}},
        {{-s(5 + 2 * s(10.0 / 7)) / 3, (322 - 13 * s(70)) / 900},
         {-s(5 - 2 * s(10.0 / 7)) / 3, (322 + 13 * s(70)) / 900},
         {0.0, 128.0 / 225},
         {s(5 - 2 * s(10.0 / 7)) / 3, (322 + 13 * s(70)) / 900},
         {s(5 + 2 * s(10.0 / 7)) / 3, (322 - 13 * s(70)) / 900}},
    };

    /** Map [-1, 1] onto [a, b]: x = mid + half * t, dx = half * dt. */
    static double gauss(DoubleUnaryOperator f, double a, double b, int n) {
        double mid = (a + b) / 2, half = (b - a) / 2, sum = 0;
        for (double[] tw : NODES[n]) {
            sum += tw[1] * f.applyAsDouble(mid + half * tw[0]);
        }
        return half * sum;
    }
```

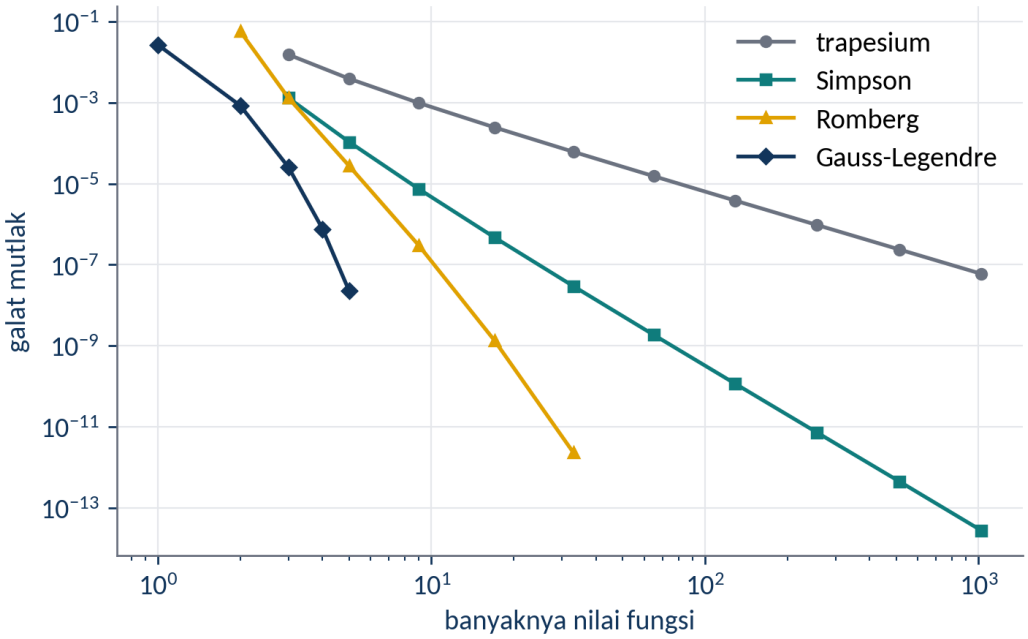
Kode 10.8 (Lanjutan) *Gauss.java: aturan yang sama dalam Java, dengan titik simpul dalam larik tiga dimensi.*

```
    public static void main(String[] args) {
        double ln2 = Quadrature.LN2;
        System.out.println(
            "points  Gauss-Legendre   error    trapezoid error  Simpson error");
        for (int n = 1; n <= 5; n++) {
            double g = gauss(Quadrature::f, 0.0, 1.0, n);
            StringBuilder line = new StringBuilder(String.format(Locale.ROOT,
                "%6d  %.12f  %.1e", n, g, Math.abs(g - ln2)));
            if (n > 1) {                              // n points = n - 1 intervals
                double t = Quadrature.trapezoid(Quadrature::f, 0.0, 1.0, n - 1);
                line.append(String.format(Locale.ROOT, "  %15.1e",
                                          Math.abs(t - ln2)));
            }
            if (n % 2 == 1 && n > 1) {                // Simpson needs n odd
                double sv = Quadrature.simpson(Quadrature::f, 0.0, 1.0, n - 1);
                line.append(String.format(Locale.ROOT, "  %13.1e",
                                          Math.abs(sv - ln2)));
            }
            System.out.println(line);
        }
    }
}
```

Kode 10.8 mencetak tabel yang sama:

```
points  Gauss-Legendre   error    trapezoid error  Simpson error
     1  0.666666666667  2.6e-02
     2  0.692307692308  8.4e-04          5.7e-02
     3  0.693121693122  2.5e-05          1.5e-02        1.3e-03
     4  0.693146417445  7.6e-07          6.9e-03
     5  0.693147157853  2.3e-08          3.9e-03        1.1e-04
```

Dengan lima nilai fungsi, Gauss benar sampai 2.3×10^{-8}; Simpson dengan lima nilai berada di 1.1×10^{-4} dan aturan trapesium di 3.9×10^{-3}. **Gambar 10.2** menempatkan keempat metode bab ini pada satu grafik, galat terhadap banyaknya nilai fungsi; makin curam sebuah garis, makin banyak akurasi yang dibeli setiap evaluasi tambahan.



Gambar 10.2 Galat terhadap banyaknya nilai fungsi untuk $\int_0^1 d\,x/(1+x) = \ln 2$: trapesium (orde 2), Simpson (orde 4), Romberg, dan Gauss–Legendre.

Aturan Gauss memiliki dua keterbatasan, keduanya relevan untuk kebun itu. Aturan ini memerlukan nilai di posisi irasional seperti $0.5 \pm 0.5/\sqrt{3}$ dari selang, sehingga tidak dapat dipakai dengan data yang diukur di tempat-tempat tetap; jarak tegak Anto setiap sepuluh meter tidak berguna baginya. Dan satu aturan berorde tinggi pada selang yang panjang berisiko untuk fungsi yang tidak mulus. Keduanya dijawab dengan memakai aturan Gauss per *panel*: bagi $[a, b]$ menjadi potongan-potongan dan gunakan aturan tiga atau lima titik pada masing-masing. Bila suatu hari Anto mengukur ulang, ia dapat merentangkan pitanya di tiga titik Gauss setiap panel dua puluh meter, yaitu 2,25 m, 10 m, dan 17,75 m dari awal panel, atau, dengan aturan dua titik, di 4,23 m dan 15,77 m, dan mendapatkan akurasi yang lebih baik dengan jarak tegak yang lebih sedikit.

Seberapa besar manfaatnya? Karena berkas itu dibuat dari kurva sungai yang diketahui, kita dapat mencobanya. Dengan aturan dua titik pada masing-masing dari delapan panel dua puluh meter, Anto akan mengambil enam belas jarak tegak, satu lebih sedikit daripada sekarang, di 4,23 m dan 15,77 m dari awal setiap panel. Bila ia dapat membaca pita ukur dengan sempurna, luasnya akan keluar 7026,81 m^2, meleset kurang dari seperseratus meter persegi, seratus kali lebih baik daripada aturan Simpson dengan tujuh belas jarak tegak. Bila dicatat sampai 0,1 m terdekat, seperti yang sebenarnya ia lakukan, keenam belas jarak tegak yang sama memberi 7028,0 m^2, meleset 1,2 m^2, tidak lebih baik daripada Simpson. Pelajaran Subbab 10.5 kembali dalam bentuk

yang lebih tajam: aturan yang lebih cerdik hanya menguntungkan bila datanya cukup akurat untuk layak mendapatkannya. Bagi juru ukur dengan alat ukur jarak laser yang membaca sampai milimeter, titik-titik Gauss adalah penghematan yang nyata; bagi pita ukur yang dibaca sampai desimeter, pilihan aturan hampir tidak berpengaruh asalkan setidaknya setara Simpson.

10.9 Menyempurnakannya: Simpson Adaptif

Langkah yang seragam memboroskan usaha. Di tempat fungsi mulus, langkah besar sudah cukup; di tempat fungsi berbelok tajam, diperlukan langkah kecil. Aturan *adaptif* memilih langkahnya sendiri. Versi klasiknya adalah Simpson adaptif [6], [7]: hitung aturan Simpson S_1 pada sebuah selang dan S_2 pada kedua paruhnya. Menurut Persamaan (10.7) galat S_2 sekitar $(S_2 - S_1)/15$. Bila itu di bawah toleransi, terima S_2 ditambah koreksinya; bila tidak, perlakukan setiap paruh dengan cara yang sama dengan separuh toleransi. Rekursinya berhenti sendiri di tempat fungsinya mudah dan menggali dalam hanya di tempat fungsinya sulit.

Fungsi ujinya $\sqrt{x}$ pada $[0,1]$, yang integralnya $2/3$. Kelihatannya tidak berbahaya, tetapi turunannya tak hingga di $x = 0$, sehingga suku galat Persamaan (10.4), yang memerlukan $f^{(4)}$ terbatas, tidak berlaku di sana, dan Simpson seragam konvergen lambat. **Kode 10.9** menghitung setiap evaluasi f.

Kode 10.9 `adaptive.py`: integrasi Simpson adaptif, dibandingkan dengan Simpson seragam.

```
"""Adaptive Simpson integration of sqrt(x) from 0 to 1 (exact 2/3): small
panels only where the function is hard, near x = 0."""
import math

calls = 0


def f(x):
    global calls
    calls += 1
    return math.sqrt(x)


def simpson_panel(a, fa, m, fm, b, fb):
    return (b - a) / 6 * (fa + 4 * fm + fb)


def adapt(a, fa, m, fm, b, fb, whole, tol, depth):
    """Split [a, b] in two; accept when the two halves agree with the whole."""
    lm, rm = (a + m) / 2, (m + b) / 2
    flm, frm = f(lm), f(rm)
    left = simpson_panel(a, fa, lm, flm, m, fm)
    right = simpson_panel(m, fm, rm, frm, b, fb)
    diff = left + right - whole
    if depth == 0 or abs(diff) <= 15 * tol:
        return left + right + diff / 15          # Richardson correction
    return (adapt(a, fa, lm, flm, m, fm, left, tol / 2, depth - 1)
            + adapt(m, fm, rm, frm, b, fb, right, tol / 2, depth - 1))


def adaptive_simpson(a, b, tol):
    fa, fm, fb = f(a), f((a + b) / 2), f(b)
    whole = simpson_panel(a, fa, (a + b) / 2, fm, b, fb)
    return adapt(a, fa, (a + b) / 2, fm, b, fb, whole, tol, 50)


def uniform_simpson(n):
    h = 1.0 / n
    return h / 3 * (f(0.0) + 4 * sum(f(i * h) for i in range(1, n, 2))
                    + 2 * sum(f(i * h) for i in range(2, n, 2)) + f(1.0))
```

Kode 10.9 (Lanjutan) *adaptive.py: integrasi Simpson adaptif, dibandingkan dengan Simpson seragam.*

```
def main():
    global calls
    exact = 2 / 3
    print("tolerance   adaptive result  error    function values")
    for tol in (1e-4, 1e-6, 1e-8):
        calls = 0
        r = adaptive_simpson(0.0, 1.0, tol)
        print(f"{tol:9.0e}   {r:.12f}   {abs(r - exact):.1e}  {calls:8d}")
    print("uniform Simpson needs, for an error below 1e-8:")
    n = 2
    while True:
        calls = 0
        err = abs(uniform_simpson(n) - exact)
        if err <= 1e-8:
            print(f"  n = {n} intervals, error {err:.1e}, {calls} function values")
            break
        n *= 2


if __name__ == "__main__":
    main()
```

Kode 10.10 `Adaptive.java`: perbandingan yang sama dalam Java.

```java
import java.util.Locale;

/** Adaptive Simpson integration of sqrt(x) from 0 to 1 (exact 2/3): small
 *  panels only where the function is hard, near x = 0. */
public class Adaptive {
    static int calls = 0;

    static double f(double x) {
        calls++;
        return Math.sqrt(x);
    }

    static double simpsonPanel(double a, double fa, double m, double fm,
                               double b, double fb) {
        return (b - a) / 6 * (fa + 4 * fm + fb);
    }

    /** Split [a, b] in two; accept when the two halves agree with the whole. */
    static double adapt(double a, double fa, double m, double fm, double b,
                        double fb, double whole, double tol, int depth) {
        double lm = (a + m) / 2, rm = (m + b) / 2;
        double flm = f(lm), frm = f(rm);
        double left = simpsonPanel(a, fa, lm, flm, m, fm);
        double right = simpsonPanel(m, fm, rm, frm, b, fb);
        double diff = left + right - whole;
        if (depth == 0 || Math.abs(diff) <= 15 * tol) {
            return left + right + diff / 15;          // Richardson correction
        }
        return adapt(a, fa, lm, flm, m, fm, left, tol / 2, depth - 1)
               + adapt(m, fm, rm, frm, b, fb, right, tol / 2, depth - 1);
    }

    static double adaptiveSimpson(double a, double b, double tol) {
        double fa = f(a), fm = f((a + b) / 2), fb = f(b);
        double whole = simpsonPanel(a, fa, (a + b) / 2, fm, b, fb);
        return adapt(a, fa, (a + b) / 2, fm, b, fb, whole, tol, 50);
    }
```

Kode 10.10 (Lanjutan) *Adaptive.java: perbandingan yang sama dalam Java.*

```
    static double uniformSimpson(int n) {
        double h = 1.0 / n, odd = 0, even = 0;
        double first = f(0.0);
        for (int i = 1; i < n; i += 2) {
            odd += f(i * h);
        }
        for (int i = 2; i < n; i += 2) {
            even += f(i * h);
        }
        return h / 3 * (first + 4 * odd + 2 * even + f(1.0));
    }

    public static void main(String[] args) {
        double exact = 2.0 / 3;
        System.out.println("tolerance   adaptive result  error    function values");
        for (double tol : new double[] {1e-4, 1e-6, 1e-8}) {
            calls = 0;
            double r = adaptiveSimpson(0.0, 1.0, tol);
            System.out.println(String.format(Locale.ROOT,
                "%9.0e   %.12f   %.1e  %8d", tol, r, Math.abs(r - exact), calls));
        }
        System.out.println("uniform Simpson needs, for an error below 1e-8:");
        for (int n = 2; ; n *= 2) {
            calls = 0;
            double err = Math.abs(uniformSimpson(n) - exact);
            if (err <= 1e-8) {
                System.out.println(String.format(Locale.ROOT,
                    "  n = %d intervals, error %.1e, %d function values",
                    n, err, calls));
                break;
            }
        }
    }
}
```

Kode 10.10 meneruskan nilai-nilai fungsi ke bawah sepanjang rekursi persis seperti versi Python, sehingga tidak ada titik yang dievaluasi dua kali. Keduanya mencetak:

```
tolerance   adaptive result  error    function values
    1e-04   0.666663972068   2.7e-06        37
    1e-06   0.666666647130   2.0e-08       105
    1e-08   0.666666666641   2.6e-11       317
uniform Simpson needs, for an error below 1e-8:
  n = 65536 intervals, error 4.8e-09, 65537 function values
```

Dengan 105 nilai fungsi, aturan adaptif mencapai galat 2×10^{-8}; Simpson seragam memerlukan 65.537 nilai untuk galat yang serupa, lebih dari enam ratus kali lipat. Aturan adaptif menghabiskan hampir semua evaluasinya di daerah kecil dekat $x = 0$, tempat $\sqrt{x}$ naik curam, dan mengambil beberapa langkah besar di tempat lain. Batas kedalaman 50 adalah jaring pengaman terhadap fungsi yang tidak pernah memenuhi uji itu.

> **PERINGATAN:** Aturan adaptif hanya sebaik taksiran galatnya, dan taksiran itu berasal dari membandingkan dua jawaban pada titik-titik yang sama. Fungsi dengan paku sempit yang jatuh di antara beberapa titik sampel pertama dapat mengecohnya sepenuhnya: kedua jawaban melewatkan paku itu, saling setuju, dan diterima. Bila Anda tahu di mana sebuah fungsi berulah, bagilah selangnya di sana sendiri.

10.10 Memeriksa terhadap Pustaka

Modul `integrate` milik SciPy memiliki semua aturan bab ini dan lebih. Rilis terbarunya, SciPy 1.18.1, keluar pada 21 Agustus 2026, dan NumPy 2.5.3 pada 6 September 2026 [8], [9]. Untuk *data*, `trapezoid`, `simpson`, dan `romb` menerima larik nilai; `simpson` menangani banyak pias ganjil dengan koreksi di ujungnya, dan `romb` memerlukan $2^k + 1$ nilai berjarak sama. Untuk *fungsi*, `quad` memanggil rutin Gauss–Kronrod adaptif dari pustaka QUADPACK [10], dan `fixed_quad` adalah Gauss–Legendre biasa dengan n titik [11]. Dua nama dari tutorial lama sudah tidak ada: `numpy.trapz` dihapus pada NumPy 2.4 dan digantikan `numpy.trapezoid` [12], sedangkan `scipy.integrate.romberg` dan `quadrature` dihapus pada SciPy 1.15 dan digantikan `quad` [13]. **Kode 10.11** memeriksa angka-angka kita.

Kode 10.11 `compare_scipy.py`: kebun dan integral-integral uji dengan SciPy.

```
"""Check this chapter's rules against SciPy."""
import math
import numpy as np
import scipy
from scipy import integrate
from orchard import load

x, y = load("../data/orchard.csv")
print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
print(f"orchard, trapezoid: {integrate.trapezoid(y, x):.1f} m2")
print(f"orchard, simpson:   {integrate.simpson(y, x=x):.1f} m2")
print(f"orchard, romb:      {integrate.romb(y, dx=x[1] - x[0]):.1f} m2")
val, err = integrate.quad(lambda t: 1 / (1 + t), 0, 1)
print(f"quad 1/(1+x):   {val:.15f} (estimated error {err:.1e})")
val, _ = integrate.fixed_quad(lambda t: 1 / (1 + t), 0, 1, n=5)
print(f"fixed_quad n=5: {val:.12f}")
val, err, info = integrate.quad(math.sqrt, 0, 1, full_output=True)[:3]
print(f"quad sqrt(x):   {val:.15f} ({info['neval']} function values)")
```

```
NumPy 2.4.4, SciPy 1.17.1
orchard, trapezoid: 7023.5 m2
orchard, simpson:   7027.7 m2
orchard, romb:      7026.6 m2
quad 1/(1+x):   0.693147180559945 (estimated error 7.7e-15)
fixed_quad n=5: 0.693147157853
quad sqrt(x):   0.666666666666667 (231 function values)
```

Luas trapesium dan Simpson sama dengan hasil kita sampai digit yang dicetak. `romb`, yang menerapkan tabel Romberg lengkap pada tujuh belas jarak tegak Anto, memberi 7026,6 m^2; seperti diperingatkan Subbab 10.7, kolom ekstrapolasi tinggi kurang dapat dipercaya pada data hasil pengukuran, walaupun di sini kebetulan mendarat dalam seperempat meter persegi dari kebenaran. `fixed_quad` dengan lima titik menghasilkan kembali nilai Gauss kita, `0.693147157853`, digit demi digit. Dan `quad` menemukan ln2 dengan presisi penuh, bahkan $\int_0^1 \sqrt{x}\,dx$ dengan presisi penuh memakai 231 nilai fungsi, lebih banyak daripada yang dipakai Simpson adaptif kita untuk 10^{-8} tetapi untuk enam belas digit yang benar: aturan Gauss–Kronrod QUADPACK memadukan dua gagasan besar bab ini, titik Gauss dan keadaptifan.

Folder kode juga memuat pengujian (`test_integ.py`, `IntegTest.java`) yang memeriksa fakta-fakta yang tidak mungkin dilanggar implementasi yang benar: aturan trapesium eksak untuk garis lurus, Simpson untuk kubik, Gauss tiga titik untuk x^5, diagonal Romberg mengalahkan trapesium tempat ia bermula, dan “kebun” berbentuk persegi panjang memiliki luas lebar kali panjang.

10.11 Metode Numerik dalam Praktik: Profitina

Dasbor analitik historis Profitina menyimpan skor visibilitas AI mingguan untuk setiap merek dan menggambar trennya. Pemilik merek sering menginginkan satu angka untuk suatu periode, misalnya "berapa rata-rata skor saya kuartal ini?", dan rata-rata terhadap waktu adalah integral dibagi panjang periodenya. Bila pemindaiannya berjarak sama, aturan trapesium atas skor-skor mingguan adalah rata-rata yang wajar, dengan bobot separuh pada minggu pertama dan terakhir, yang lebih adil daripada rata-rata biasa ketika suatu periode dimulai atau berakhir di tengah-tengah perubahan. Bila pemindaiannya tidak berjarak sama, seperti ketika pemilik meminta pemindaian tambahan sebelum sebuah kampanye, aturan trapesium pada tanggal pemindaian yang sebenarnya tetap bekerja, sedangkan rata-rata biasa akan memberi bobot terlalu besar pada minggu-minggu yang sibuk. Penalaran yang sama berlaku untuk "total perbaikan yang dicapai selama kampanye": luas di antara kurva skor dan tingkat awalnya.

Koin utilitas PROFIT milik Profitina dirancang dengan emisi yang meluruh: koin baru dilepas dengan laju yang turun seiring waktu, dan sebagian dibakar. Total yang diemisikan selama suatu periode adalah integral laju emisinya. Untuk laju yang diberikan oleh rumus sederhana hal ini dapat dikerjakan dengan tangan, tetapi jadwal yang berubah bertahap, atau yang digabungkan dengan pembakaran harian yang sebenarnya, adalah tabel nilai, dan alat-alat bab ini, trapesium untuk data serta Gauss atau aturan adaptif untuk rumus yang mulus, adalah cara total seperti itu dihitung dan diperiksa. Parameter emisi yang persis dan cara kerja internal orakelnya adalah milik Profitina; matematikanya milik bab ini, dan, seperti biasa, angka-angka memberi saran sedangkan pemakai yang memutuskan.

10.12 Memakainya Sungguhan

- **Ukur jarak tegak dengan sudut siku, berjarak sama, dengan banyak pias genap.** Dengan begitu aturan Simpson langsung berlaku dan taksiran galat Persamaan (10.7) diperoleh gratis. Bila tanahnya berakhir pada pias ganjil, gunakan aturan 3/8 pada tiga pias terakhir.
- **Ukur lebih rapat di tempat batasnya berbelok.** Di tempat sungai berbelok, tambahkan jarak tegak setiap 5 m dan perlakukan bentang itu sebagai potongan tersendiri dengan langkahnya sendiri; potongan-potongan itu dijumlahkan. Inilah keadaptifan yang dikerjakan dengan tangan.
- **Potong tanah yang tidak beraturan menjadi bagian-bagian dengan garis dasar lurus.** Bidang tanpa sisi lurus dapat dibagi menjadi poligon (yang luasnya diperoleh dari koordinat sudut-sudutnya) ditambah pias-pias di sepanjang setiap sisinya, masing-masing diintegralkan dari garis dasarnya sendiri.
- **Hujan menjadi liter.** Intensitas hujan dari penakar hujan atau stasiun cuaca, dalam milimeter per jam, yang diintegralkan terhadap waktu memberi milimeter curah hujan; satu milimeter di atas satu meter persegi adalah satu liter. Kalikan dengan luas atap dan dengan bagian yang sampai ke tandon (Soal Latihan 10.6).
- **Luas menjadi volume.** Aturan yang sama dapat dipakai untuk mengintegralkan dua kali. Untuk mengetahui berapa banyak air yang ditampung kolam ikan atau embung di kebun, ukurlah kedalamannya di sepanjang beberapa garis sejajar yang melintanginya; integralkan kedalaman pada setiap garis dengan aturan Simpson untuk mendapatkan luas penampang dalam meter persegi, lalu integralkan luas-luas penampang itu di sepanjang kolam dengan aturan Simpson sekali lagi untuk mendapatkan meter kubik. Satu meter kubik adalah seribu liter, yang memberi tahu berapa lama pompa dengan kapasitas tertentu memerlukan waktu untuk mengosongkan atau mengisinya, dan berapa banyak kapur atau pakan ikan yang diperlukan untuk dosis tertentu per meter kubik.

- **Daya menjadi energi.** Colokan pintar yang melaporkan daya mesin cuci dalam watt setiap menit dapat diintegralkan menjadi watt-jam dengan aturan trapesium; sebuah binatu seperti Profitina Laundry, studi kasus yang dipakai di sepanjang buku ini, dapat membandingkan mesin-mesinnya dengan cara ini.
- **Taksir galat dengan data yang Anda miliki.** Bagi dua datanya (setiap titik kedua), hitung ulang, dan gunakan Persamaan (10.7). Bila kedua jawaban berselisih lebih dari yang dapat Anda terima, ukurlah lebih rapat.
- **Serahkan angka resmi kepada pejabat yang berwenang.** Luas hasil hitungan untuk perencanaan. Sertifikat tanah di Indonesia diterbitkan berdasarkan pengukuran resmi oleh juru ukur badan pertanahan, dan jawaban pita ukur dan Simpson paling baik dipakai untuk mengetahui apa yang dapat diharapkan.

10.13 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Simpson dengan banyak pias ganjil.** Bobot 1,4,2,4, … ,4,1 hanya bermakna untuk banyak pias genap. Fungsi kita memunculkan galat; fungsi yang ceroboh diam-diam mengembalikan luas yang keliru. Gunakan aturan 3/8 untuk sisa yang ganjil.
- **Lupa separuh di ujung-ujung.** Aturan trapesium memberi nilai ujung bobot $\frac{1}{2}$. Jumlah biasa kali h adalah aturan persegi panjang, yang hanya berorde satu dan kelebihan atau kekurangan sekitar setengah pias di setiap ujung.
- **Jarak tak seragam dengan rumus jarak seragam.** Bila satu jarak tegak diambil di 25 m, bukan 20 m, rumus gabungan menjadi keliru. Integralkan setiap pias dengan lebarnya sendiri, atau berikan posisi yang sebenarnya (`x=`) kepada SciPy.
- **Memercayai ekstrapolasi orde tinggi pada data berderau.** Kolom atas Romberg mengandaikan fungsi yang mulus. Pada nilai hasil pengukuran, berhentilah setelah satu atau dua kolom.
- **Mengharapkan Gauss memakai data Anda.** Aturan Gauss memerlukan nilai di titik-titiknya sendiri. Untuk data tabel gunakan trapesium, Simpson, atau `romb`.
- **Memakai `np.trapz` atau `scipy.integrate.romberg`.** Keduanya sudah dihapus. Gunakan `numpy.trapezoid` atau `scipy.integrate.trapezoid`, dan `scipy.integrate.quad`.
- **Mengabaikan perilaku singular.** Rumus galat seperti Persamaan (10.4) memerlukan turunan yang terbatas. Pada akar kuadrat, lompatan, atau patahan, ordenya turun; bagilah selangnya di titik bermasalah itu atau gunakan aturan adaptif.

10.14 Ringkasan Bab

- Setiap aturan kuadratur mengintegralkan secara eksak sebuah kurva interpolasi sederhana. Aturan trapesium memakai garis dan bergalat $O(h^2)$ (Persamaan (10.1), Persamaan (10.2)); aturan Simpson 1/3 memakai parabola, eksak untuk kubik, dan bergalat $O(h^4)$ (Persamaan (10.4)); aturan 3/8 (Persamaan (10.6)) menangani banyak pias ganjil.
- Rumus Euler–Maclaurin (Persamaan (10.3)) memperlihatkan bahwa galat trapesium adalah deret dalam h^2. Ekstrapolasi Richardson atasnya menghasilkan aturan Simpson (Persamaan (10.5)) dan, bila diulang, tabel Romberg (Persamaan (10.8)).
- Membandingkan jawaban dengan langkah h dan dengan langkah $2h$ menaksir galatnya tanpa mengetahui kebenaran (Persamaan (10.7)); untuk kebun itu, taksiran ini meramalkan galat trapesium dan Simpson dengan baik dan memberi luas 7026,9 m^2 dibandingkan luas sebenarnya 7026,8 m^2.

- Kuadratur Gauss–Legendre memilih titik sekaligus bobotnya dan mengintegralkan polinomial berderajat $2n-1$ secara eksak dengan n titik (Persamaan (10.9), Persamaan (10.10)); ia memerlukan fungsi, bukan data tetap.
- Simpson adaptif hanya memakai langkah kecil di tempat fungsinya sulit; untuk $\sqrt{x}$ ia memerlukan 105 nilai sedangkan Simpson seragam memerlukan 65.537.
- SciPy menyediakan `trapezoid`, `simpson`, dan `romb` untuk data serta `quad` dan `fixed_quad` untuk fungsi; `np.trapz` dan `integrate.romberg` sudah tidak ada.

10.15 Soal Latihan

1. Dengan tangan, hitung luas 40 m pertama kebun itu (jarak tegak 38,6, 40,0, 41,2, 42,2, dan 43,1 m) dengan aturan trapesium dan dengan aturan Simpson, lalu taksir galat trapesium dari selisihnya.
2. Pagar tetangga menghentikan pita ukur di 150 m, sehingga hanya enam belas jarak tegak pertama (0 sampai 150 m, lima belas pias) yang dapat dipakai. Hitung luasnya dengan aturan Simpson 1/3 pada dua belas pias pertama dan aturan 3/8 pada tiga pias terakhir, lalu bandingkan dengan aturan trapesium. Luas sebenarnya bagian itu 6692,5 m².
3. Buktikan Persamaan (10.5): tuliskan $4T(h) - T(2h)$ untuk $n = 2$ pias selebar h dan tunjukkan bahwa membaginya dengan 3 menghasilkan aturan Simpson. Mengapa argumen ini berlaku untuk sembarang n genap?
4. Ulangi pemeriksaan orde di Subbab 10.6 dengan $\int_0^1 \frac{4}{1+x^2}\,dx = \pi$. Anda akan menemukan bahwa rasio Simpson mendekati 64, bukan 16. Jelaskan hal ini dengan Persamaan (10.3).
5. Turunkan aturan Gauss–Legendre tiga titik pada $[-1,1]$ dengan menuntut keeksakan untuk $1, x^2$, dan x^4 dengan titik $0, \pm t$ dan bobot w_0, w_1. Terapkan dengan tangan pada $\ln 2 = \int_0^1 d\,x/(1+x)$.
6. Penakar hujan di rumah Anto mencatat intensitas hujan setiap sepuluh menit selama badai sore hari: 0, 6, 18, 42, 30, 12, 6, dan 0 mm per jam. Taksir curah hujan totalnya dalam milimeter dengan aturan trapesium dan dengan aturan-aturan Simpson, serta liter air yang sampai ke tandon keluarga itu dari atap seluas 60 m² bila 80% hujan tertampung.

10.16 Penyelesaian Soal Latihan

1. Dengan $h = 10$: trapesium $T = 10 \times (38.6/2 + 40.0 + 41.2 + 42.2 + 43.1/2) = 10 \times 164.25 = 1642.5$ m². Simpson $S = \frac{10}{3}(38.6 + 4\times 40.0 + 2\times 41.2 + 4\times 42.2 + 43.1) = \frac{10}{3}\times 492.9 = 1643.0$ m². Dengan setiap jarak tegak kedua (langkah 20), $T(2h) = 20 \times (38.6/2 + 41.2 + 43.1/2) = 20 \times 82.05 = 1641.0$, sehingga menurut Persamaan (10.7) galat trapesium sekitar $(1642.5 - 1641.0)/3 = 0.5$ m², dan memang $1642.5 + 0.5 = 1643.0$ adalah nilai Simpson. Kedua aturan hanya berselisih setengah meter persegi pada bentang yang hampir lurus ini.

2. Aturan Simpson 1/3 pada 0 sampai 120 m (dua belas pias, jarak tegak 38,6 sampai 47,2) memberi 5480.0 m². Aturan 3/8 pada 120 sampai 150 m memberi $\frac{3\times 10}{8}(47.2 + 3\times 42.2 + 3\times 38.2 + 35.0) = 3.75 \times 323.4 = 1212.75$ m². Totalnya 6692.75 m², dalam 0.3 m² dari luas sebenarnya 6692.5. Aturan trapesium pada kelima belas pias memberi 6689.0 m², kurang 3,5 m².

3. Untuk $n = 2$, $T(h) = h\left(\frac{f_0}{2} + f_1 + \frac{f_2}{2}\right)$ dan $T(2h) = 2h \cdot \frac{f_0+f_2}{2} = h(f_0 + f_2)$. Maka $4T(h) - T(2h) = h(2f_0 + 4f_1 + 2f_2 - f_0 - f_2) = h(f_0 + 4f_1 + f_2)$, dan membaginya dengan 3 memberi

$\frac{h}{3}(f_0 + 4f_1 + f_2)$, aturan Simpson. Untuk sembarang n genap, $T(h)$ maupun $T(2h)$ adalah jumlah atas pasangan-pasangan pias yang berurutan, dan identitas itu berlaku pasangan demi pasangan; menjumlahkan pasangan-pasangan itu menghasilkan aturan Simpson gabungan Persamaan (10.4).

4. Mengganti `f` dan nilai eksak dalam `quadrature.py` memberi galat Simpson 2.365×10^{-9} untuk $n = 16$ dan 3.696×10^{-11} untuk $n = 32$, rasio 64,0, dan rasio yang sama untuk penggandaan berikutnya. Aturan Simpson adalah $\frac{4T(h) - T(2h)}{3}$, yang menghapus suku h^2 pada Persamaan (10.3); galat terdepannya lalu suku h^4, yang koefisiennya sebanding dengan $f'''(1) - f'''(0)$. Untuk $f(x) = 4/(1 + x^2)$, $f'''(x) = 96x(1 - x^2)/(1 + x^2)^4$, yang bernilai nol baik di $x = 0$ maupun di $x = 1$. Suku h^4 lenyap, galat terdepannya suku h^6, dan rasionya $2^6 = 64$. Aturannya tidak menjadi lebih baik secara umum; integran ini hanya beruntung di titik-titik ujungnya. Rasio trapesium tetap 4 karena $f'(1) - f'(0) = -2 - 0 \neq 0$.

5. Karena simetri, bobot luar sama dan pangkat ganjil eksak. Keeksakan untuk 1: $w_0 + 2w_1 = 2$. Untuk x^2: $2w_1 t^2 = \frac{2}{3}$. Untuk x^4: $2w_1 t^4 = \frac{2}{5}$. Membagi dua persamaan terakhir memberi $t^2 = \frac{3}{5}$, jadi $t = \sqrt{3/5} \approx 0.774597$; lalu $w_1 = \frac{1}{3t^2} = \frac{5}{9}$ dan $w_0 = 2 - \frac{10}{9} = \frac{8}{9}$. Untuk ln2 simpulnya dipetakan ke $x = 0.5 - 0.387298 = 0.112702$, 0.5, dan 0.887298, tempat $f = 0.898715$, 0.666667, dan 0.529858. Aturannya memberi $\frac{1}{2}\left(\frac{5}{9} \times 0.898715 + \frac{8}{9} \times 0.666667 + \frac{5}{9} \times 0.529858\right) = \frac{1}{2}(0.499286 + 0.592593 + 0.294366) = 0.693122$, sesuai dengan hasil program 0.693121693 dan galat 2.5×10^{-5}.

6. Langkahnya $h = 10$ menit, yaitu $\frac{1}{6}$ jam, dan ada tujuh pias. Trapesium: $\frac{1}{6}(0/2 + 6 + 18 + 42 + 30 + 12 + 6 + 0/2) = \frac{114}{6} = 19.0$ mm. Simpson: tujuh pias itu ganjil, jadi gunakan aturan 1/3 pada empat pias pertama dan aturan 3/8 pada tiga pias terakhir: $\frac{1}{18}(0 + 4 \times 6 + 2 \times 18 + 4 \times 42 + 30) = \frac{258}{18} = 14.33$ mm, ditambah $\frac{3}{8} \cdot \frac{1}{6}(30 + 3 \times 12 + 3 \times 6 + 0) = \frac{84}{16} = 5.25$ mm, total 19.58 mm. Satu milimeter di atas satu meter persegi adalah satu liter, jadi atap menerima $19.58 \times 60 = 1175$ liter dan tandon sekitar $0.8 \times 1175 = 940$ liter (jawaban trapesium memberi 912 liter). Dengan bacaan hanya setiap sepuluh menit, selisih kedua aturan, sekitar 3%, adalah ukuran ketidakpastian yang wajar; akurasi penakar hujan itu sendiri kemungkinan lebih buruk.

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] P. J. Davis and P. Rabinowitz, *Methods of Numerical Integration*, 2nd ed. Orlando, FL, USA: Academic Press, 1984; reprinted Mineola, NY, USA: Dover, 2007.

[4] W. Romberg, "Vereinfachte numerische Integration," *Det Kongelige Norske Videnskabers Selskab Forhandlinger*, vol. 28, no. 7, pp. 30–36, 1955.

[5] G. H. Golub and J. H. Welsch, "Calculation of Gauss quadrature rules," *Mathematics of Computation*, vol. 23, no. 106, pp. 221–230, 1969.

[6] W. M. McKeeman, "Algorithm 145: Adaptive numerical integration by Simpson's rule," *Communications of the ACM*, vol. 5, no. 12, p. 604, 1962.

[7] W. Gander and W. Gautschi, “Adaptive quadrature—revisited,” *BIT Numerical Mathematics*, vol. 40, no. 1, pp. 84–101, 2000.

[8] SciPy Developers, “scipy 1.18.1,” Python Package Index, Aug. 21, 2026. [Online]. Available: https://pypi.org/project/scipy/

[9] NumPy Developers, “numpy 2.5.3,” Python Package Index, Sep. 6, 2026. [Online]. Available: https://pypi.org/project/numpy/

[10] R. Piessens, E. de Doncker-Kapenga, C. W. Überhuber, and D. K. Kahaner, *QUADPACK: A Subroutine Package for Automatic Integration*. Berlin, Germany: Springer, 1983.

[11] SciPy Developers, “Integration and ODEs (scipy.integrate),” SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/integrate.html

[12] NumPy Developers, “NumPy 2.4.0 Release Notes,” accessed Sep. 2026. [Online]. Available: https://numpy.org/doc/stable/release/2.4.0-notes.html

[13] SciPy Developers, “SciPy 1.15.0 Release Notes,” accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/release/1.15.0-notes.html

Bab 11

Persamaan Diferensial Biasa: dari Euler ke Runge-Kutta

Banyak pertanyaan sehari-hari menyangkut bagaimana sesuatu berubah bila kita tahu apa yang mendorong perubahannya: seberapa cepat tandon air kosong bila kerannya terbuka, bagaimana jok sepeda motor kembali tenang setelah melewati polisi tidur, bagaimana saldo tabungan tumbuh karena bunga. Aturan untuk *laju* perubahannya sering mudah dituliskan; besaran itu sendiri, pada waktu kemudian, tidak. Persamaan diferensial biasa menyatakan aturannya, dan metode numerik melangkah maju dalam waktu, langkah demi langkah, untuk menemukan besarannya. Di akhir bab ini Anda akan mampu menuliskan situasi nyata sebagai masalah nilai awal dan, bila mungkin, menyelesaikannya secara eksak sebagai pembanding; menurunkan metode Euler dan membatasi galatnya; menurunkan metode Runge–Kutta orde dua dari teorema Taylor dan memakai metode orde empat klasik; mengukur orde setiap metode dari keluarannya sendiri; menyimulasikan satu pagi tandon irigasi tetes sebuah kebun dalam Java dan Python, lengkap dengan pompa, katup, luapan, dan saat tandon kering; merencanakan pompa dengan metode biseksi dari Bab 2 yang membungkus simulasi itu; mengubah persamaan orde dua menjadi sistem dan melihat mengapa metode Euler dapat membuat sistem yang stabil meledak; menaksir galat sebuah simulasi dari simulasi itu sendiri; serta memeriksa jawabannya dengan `solve_ivp` milik SciPy.

11.1 Cukupkah Tandon Itu sampai Pukul Sepuluh?

Musim kemarau telah tiba di kebun buah dan durian dari Bab 10. Anto, yang kini mengelolanya, telah menanam tujuh puluh pohon durian muda, dan pohon durian muda tidak boleh kehausan. Ia membangun sistem irigasi tetes sederhana: tandon plastik 1.000 liter di atas penyangga di puncak lereng, selang yang menyusuri barisan pohon, dan penetes kecil di kaki setiap pohon. Penyiramannya sendiri tidak memerlukan listrik; gravitasi yang mendorong airnya. Satu-satunya mesin adalah pompa listrik yang mengangkat air dari sumur ke tandon.

Setiap pagi rutinitasnya sama: pukul enam ia menyalakan pompa, pukul setengah tujuh ia membuka katup ke jalur tetes, dan kemudian, ketika ingat, ia mematikan pompa. Dua hal sering salah. Pada pagi-pagi tertentu ia lupa pada pompanya, tandon penuh, dan air sumur yang bersih tumpah dari pipa luapan ke tanah, bersama listrik yang telah dipakai untuk memompanya. Pada pagi yang lain tandon kering sebelum pukul sepuluh, dan pohon-pohon di ujung barisan hanya mendapat sedikit. Ia ingin tahu: *bila pompa kumatikan pada jam tertentu, seberapa tinggi tandon itu terisi, berapa yang meluap, dan kapan penetes berhenti?* Dan pertanyaan sebenarnya di balik itu: *kapan pompa harus kumatikan?*

Ia dapat mencari tahu dengan mencoba, satu pagi demi satu pagi. Tetapi tandon itu mematuhi fisika sederhana, dan fisika sederhana dapat dituliskan sebagai persamaan. Ketinggian air berubah karena air masuk (pompa) dan keluar (penetes). Aliran keluar melalui penetes bergantung pada setinggi apa air berdiri di atasnya: tandon yang penuh mendorong lebih kuat daripada yang hampir kosong. Jadi *laju* perubahan ketinggian bergantung pada ketinggian itu sendiri. Itulah **persamaan diferensial**, dan bab ini membahas cara menyelesaikan persamaan

seperti itu secara numerik, dengan melangkah maju dalam waktu: dari ketinggian pukul enam, ketinggian semenit kemudian; dari situ, ketinggian semenit berikutnya; dan seterusnya, sampai pukul sepuluh dan sesudahnya.

Langkah-langkah yang sama berada di balik prakiraan cuaca, orbit satelit, penyebaran penyakit dalam suatu populasi, kurva pengisian baterai ponsel, dan setiap mesin fisika dalam gim video. Kita mulai dengan metode paling sederhana, memahami dengan tepat seberapa kelirunya, lalu memperbaikinya sampai satu pagi tandon itu dapat disimulasikan dengan ketelitian lebih baik dari satu milimeter dalam sepersekian detik.

11.2 Gagasannya dalam Kata-Kata Sederhana

Misalkan tandon penuh, pompa mati, dan katup terbuka. Dari pengalaman Anto tahu bahwa tandon penuh meneteskan sekitar sepuluh liter per menit. Tandonnya berluas alas satu meter persegi, sehingga sepuluh liter (0,01 m^3) per menit menurunkan ketinggian satu sentimeter per menit. Bila alirannya tetap sepuluh liter per menit, ketinggian setelah sepuluh menit adalah $1.00 - 10 \times 0.01 = 0.90$ m.

Tetapi alirannya tidak tetap. Ketika ketinggian turun, tekanan di penetes turun, dan alirannya melambat. Jadi setelah sepuluh menit, pada 0,90 m, kita semestinya menghitung ulang alirannya sebelum mengambil langkah berikutnya. Untuk penetes, aliran sebanding dengan akar kuadrat ketinggian air (Subbab 11.3 menjelaskan sebabnya), sehingga pada 0,90 m alirannya $10 \times \sqrt{0.90} = 9.49$ liter per menit, dan dalam sepuluh menit berikutnya ketinggian turun sekitar 9.49 cm, menjadi $0.90 - 0.0949 = 0.805$ m. Melangkah, hitung ulang lajunya, melangkah lagi: itulah **metode Euler**, metode pertama dan paling sederhana untuk persamaan diferensial, yang diterbitkan Leonhard Euler pada 1768 [1].

Kelemahannya sudah terlihat. Selama sepuluh menit pertama kita memakai aliran di *awal* sepuluh menit itu, 10 L/menit, padahal alirannya terus turun sepanjang waktu, sampai 9,49 L/menit di akhirnya. Jadi kita mengeluarkan air sedikit terlalu banyak, dan ketinggian yang kita hitung, 0,900 m, sedikit terlalu rendah. Ketinggian sebenarnya setelah sepuluh menit, dari rumus eksak di Subbab 11.3, adalah 0,9025 m. Galatnya, 2,5 mm, kecil, tetapi berulang pada setiap langkah, dan setiap langkah bermula dari ketinggian yang sudah sedikit keliru. Subbab 11.4 mengukur bagaimana galat-galat itu terjumlah; Subbab 11.5 memperbaikinya dengan melihat laju bukan hanya di awal setiap langkah, tetapi juga di tengah dan di akhirnya.

> **WAWASAN KUNCI:** Persamaan diferensial tidak memberi tahu di mana Anda berada; ia memberi tahu ke mana harus melangkah dari mana pun Anda berada. Setiap metode numerik untuknya adalah strategi berjalan: lihat arahnya, melangkah, lihat lagi. Metode-metode itu berbeda dalam seberapa cermat mereka melihat sebelum setiap langkah.

11.3 Modelnya dan Solusi Eksaknya

11.3.1 Neraca massa

Misalkan $h(t)$ ketinggian air dalam meter pada waktu t dalam menit, dan A luas alas tandon dalam meter persegi. Volume airnya Ah, dan laju perubahannya adalah aliran masuk dikurangi aliran keluar:

$$A\frac{dh}{dt} = Q_{\text{masuk}}(t) - k\sqrt{h}. \tag{11.1}$$

Di sini Q_{masuk} adalah debit pompa dalam meter kubik per menit (0,02 untuk 20 L/menit, atau 0 bila pompa mati) dan $k\sqrt{h}$ adalah aliran keluar melalui jalur tetes. Akar kuadrat itu berasal dari hukum Torricelli: air yang keluar dari sebuah lubang di bawah tinggi tekan h memiliki kecepatan sekitar $\sqrt{2gh}$, dengan g percepatan gravitasi, sehingga debit melalui sekumpulan lubang yang tetap sebanding dengan $\sqrt{h}$ [2]. Konstanta k menggabungkan banyaknya penetes, ukurannya, dan $\sqrt{2g}$; Anto menemukannya dengan cara praktis, dari pengamatan bahwa tandon penuh ($h = 1$) memberi 10 L/menit, sehingga $k = 0.01$. Tandonnya memiliki $A = 1$ m^2 dan tinggi 1 m, sehingga menampung 1.000 liter dan ketinggian dalam meter sama dengan volume dalam meter kubik.

Persamaan (11.1) adalah **persamaan diferensial biasa** (PDB, *ordinary differential equation*): persamaan untuk fungsi $h(t)$ yang tidak diketahui yang melibatkan turunannya. *Biasa* berarti fungsinya bergantung pada satu peubah, waktu; persamaan dengan beberapa peubah, seperti suhu di seluruh permukaan pelat logam, adalah persamaan diferensial *parsial* dan tidak dibahas dalam buku ini. Bersama nilai awalnya, $h(0) = h_0$, persamaan itu membentuk **masalah nilai awal** (*initial value problem*). Secara umum kita menulis

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \tag{11.2}$$

dengan f fungsi yang diketahui yang memberi kemiringan untuk setiap waktu dan setiap nilai. Untuk tandon, $y = h$ dan $f(t, h) = \left(Q_{\text{masuk}}(t) - k\sqrt{h}\right)/A$. Di bawah syarat ringan pada f (kontinu dan memenuhi syarat Lipschitz, $f(t, y) - f(t, z) \le L\,|y - z|$) masalah ini memiliki tepat satu solusi [3]. Akar kuadrat melanggar syarat Lipschitz di $h = 0$, dan itulah sebabnya tandon kosong memerlukan sedikit kehati-hatian dalam program.

11.3.2 Solusi eksak sebagai pembanding

Dengan pompa mati, Persamaan (11.1) dapat diselesaikan dengan tangan, yang memberi kita tolok ukur bagi metode-metode numerik. Pisahkan peubahnya, dengan semua yang mengandung h di satu sisi:

$$\frac{dh}{\sqrt{h}} = -\frac{k}{A}\,dt.$$

Ruas kiri adalah turunan $2\sqrt{h}$; mengintegralkan dari waktu 0, ketika ketinggiannya h_0, sampai waktu t memberi $2\sqrt{h} - 2\sqrt{h_0} = -\frac{k}{A}t$, sehingga

$$h(t) = \left(\sqrt{h_0} - \frac{kt}{2A}\right)^2, \qquad 0 \le t \le \frac{2A\sqrt{h_0}}{k}. \tag{11.3}$$

Untuk tandon penuh Anto, $h_0 = 1$, $k = 0.01$, dan $A = 1$, sehingga $h(t) = (1 - 0.005\,t)^2$: 0,9025 m setelah 10 menit, 0,81 m setelah 20, 0,49 m setelah satu jam, dan kosong tepat setelah 200 menit. Rumus inilah yang akan dipakai untuk menguji setiap metode dalam bab ini. Sebagian besar masalah nyata tidak memiliki rumus seperti ini, dan justru itulah alasan metode numerik ada, tetapi metode yang gagal pada masalah dengan jawaban yang diketahui tidak akan lebih baik pada masalah tanpa jawaban.

11.4 Metode Euler

11.4.1 Penurunan

Bagi waktu menjadi langkah-langkah sepanjang h (huruf h lazim dipakai untuk langkah; dalam program ketinggian air bernama `h` dan langkahnya `step`, agar tidak tertukar), dengan $t_n = t_0 + nh$, dan misalkan y_n hampiran kita untuk $y(t_n)$. Teorema Taylor (Bab 9) memberi

$$\begin{aligned} y(t_{n+1}) &= y(t_n) + h y'(t_n) + \frac{h^2}{2} y''(\xi_n) \\ &= y(t_n) + h f(t_n, y(t_n)) + \frac{h^2}{2} y''(\xi_n). \end{aligned}$$

Membuang suku terakhir dan memakai hampiran y_n sebagai pengganti $y(t_n)$ yang tidak diketahui menghasilkan **metode Euler**:

$$y_{n+1} = y_n + h f(t_n, y_n). \tag{11.4}$$

Secara geometris, setiap langkah mengikuti garis singgung solusi yang melalui titik sekarang selama waktu h. Untuk tandon dengan langkah 10 menit, dua langkah pertamanya persis hitungan tangan di Subbab 11.2: $h_1 = 1 + 10 \times \left(-0{,}01\sqrt{1}\right) = 0{,}9$ dan $h_2 = 0{,}9 + 10 \times \left(-0{,}01\sqrt{0{,}9}\right) = 0{,}805132$.

11.4.2 Galat lokal dan galat global

Suku yang kita buang, $\frac{h^2}{2} y''$, adalah **galat pemotongan lokal**: galat yang dibuat dalam satu langkah bila langkah itu bermula dari nilai eksak. Besarnya $O(h^2)$. Tetapi kita mengambil $(t_{\text{akhir}} - t_0)/h$ langkah, dan galat-galat itu menumpuk. Argumen yang cermat [3], [4] memperlihatkan bahwa bila $|y''| \le M$ dan f berkonstanta Lipschitz L, **galat global** pada waktu t_n memenuhi

$$|y(t_n) - y_n| \le \frac{hM}{2L} \left(e^{L(t_n - t_0)} - 1\right). \tag{11.5}$$

Bagian terpentingnya adalah faktor h: galat global metode Euler adalah $O(h)$, satu orde lebih rendah daripada galat lokalnya, karena $O(1/h)$ langkah masing-masing menyumbang $O(h^2)$. Metode Euler adalah metode *orde satu*: bagi dua langkahnya, galatnya menjadi separuh. Faktor eksponensial itu memperingatkan bahwa galat dapat tumbuh dalam waktu yang lama bila persamaannya sendiri memperbesar perbedaan; untuk tandon yang mengosong yang terjadi justru sebaliknya (dua tandon dengan ketinggian berbeda saling mendekat ketika mengosong), dan batas itu terlalu pesimistis.

11.4.3 Program pertama

Kode 11.1 menerapkan Persamaan (11.4) pada tandon yang mengosong dengan langkah 10 menit, mencetak setiap langkah di samping solusi eksak Persamaan (11.3), lalu mengulang perhitungan dengan langkah 5, 2,5, dan 1,25 menit untuk melihat bagaimana galat pada satu jam berubah.

Kode 11.1 `tank_euler.py`: metode Euler untuk tandon yang mengosong, dibandingkan dengan solusi eksak.

```
"""Euler's method for a draining water tank: dh/dt = -k*sqrt(h)/A.
Full tank h = 1 m, area A = 1 m^2, drip-line constant k = 0.01; exact
solution h(t) = (1 - 0.005 t)^2 with t in minutes."""
import math

K, AREA = 0.01, 1.0


def slope(t, h):
    return -K * math.sqrt(max(h, 0.0)) / AREA


def exact(t):
    return (1.0 - K * t / (2 * AREA)) ** 2


def euler(f, t0, y0, step, t_end):
    t, y = t0, y0
    rows = [(t, y)]
    while t < t_end - 1e-9:
        y = y + step * f(t, y)          # follow the tangent for one step
        t = t + step
        rows.append((t, y))
    return rows


def main():
    print("  t(min)   Euler h(m)   exact h(m)     error")
    for t, y in euler(slope, 0.0, 1.0, 10.0, 60.0):
        print(f"{t:8.0f}   {y:10.6f}   {exact(t):10.6f}   {y - exact(t):9.6f}")
    print("error at t = 60 min for smaller steps:")
    old = None
    for step in (10.0, 5.0, 2.5, 1.25):
        err = euler(slope, 0.0, 1.0, step, 60.0)[-1][1] - exact(60.0)
        ratio = f"  ratio {old / err:.2f}" if old else ""
        print(f"  step {step:5.2f} min   error {err:.6f}{ratio}")
        old = err


if __name__ == "__main__":
    main()
```

Fungsi `euler` tidak tahu apa-apa tentang tandon: ia menerima sembarang fungsi kemiringan `f(t, y)`, titik awal, langkah, dan waktu akhir, yaitu bentuk umum Persamaan (11.2). Fungsi kemiringan memakai `max(h, 0.0)` agar ketinggian yang sedikit di bawah nol karena pembulatan tidak membuat akar kuadratnya gagal. Syarat gelung `t < t_end - 1e-9` menjaga agar banyaknya langkah tidak meleset satu karena pembulatan pada `t`. **Kode 11.2** adalah versi Java-nya, dengan kemiringan dikirim sebagai `DoubleBinaryOperator`.

Kode 11.2 `TankEuler.java`: program yang sama dalam Java.

```java
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.function.DoubleBinaryOperator;

/** Euler's method for a draining water tank: dh/dt = -k*sqrt(h)/A.
 *  Full tank h = 1 m, area A = 1 m^2, drip-line constant k = 0.01; exact
 *  solution h(t) = (1 - 0.005 t)^2 with t in minutes. */
public class TankEuler {
    static final double K = 0.01, AREA = 1.0;

    static double slope(double t, double h) {
        return -K * Math.sqrt(Math.max(h, 0.0)) / AREA;
    }

    static double exact(double t) {
        double r = 1.0 - K * t / (2 * AREA);
        return r * r;
    }

    /** Returns rows {t, y}. */
    static List<double[]> euler(DoubleBinaryOperator f, double t0, double y0,
                                double step, double tEnd) {
        double t = t0, y = y0;
        List<double[]> rows = new ArrayList<>();
        rows.add(new double[] {t, y});
        while (t < tEnd - 1e-9) {
            y = y + step * f.applyAsDouble(t, y);   // follow the tangent one step
            t = t + step;
            rows.add(new double[] {t, y});
        }
        return rows;
    }

    public static void main(String[] args) {
        System.out.println("  t(min)   Euler h(m)   exact h(m)     error");
        for (double[] r : euler(TankEuler::slope, 0.0, 1.0, 10.0, 60.0)) {
            System.out.println(String.format(Locale.ROOT,
                "%8.0f   %10.6f   %10.6f   %9.6f",
                r[0], r[1], exact(r[0]), r[1] - exact(r[0])));
        }
        System.out.println("error at t = 60 min for smaller steps:");
        double old = 0;
        for (double step : new double[] {10.0, 5.0, 2.5, 1.25}) {
            List<double[]> rows = euler(TankEuler::slope, 0.0, 1.0, step, 60.0);
            double err = rows.get(rows.size() - 1)[1] - exact(60.0);
```

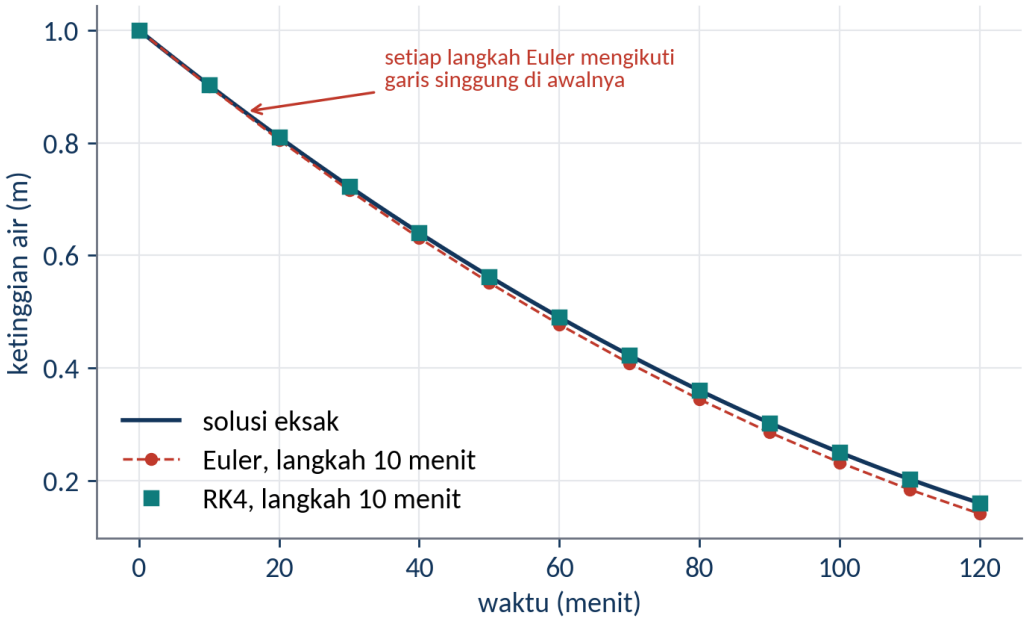
Kode 11.2 (Lanjutan) *TankEuler.java: program yang sama dalam Java.*

```java
            String ratio = old != 0
                ? String.format(Locale.ROOT, "  ratio %.2f", old / err) : "";
            System.out.println(String.format(Locale.ROOT,
                "  step %5.2f min   error %.6f%s", step, err, ratio));
            old = err;
        }
    }
}
```

Keduanya mencetak:

```
  t(min)   Euler h(m)   exact h(m)     error
       0     1.000000     1.000000    0.000000
      10     0.900000     0.902500   -0.002500
      20     0.805132     0.810000   -0.004868
      30     0.715403     0.722500   -0.007097
      40     0.630821     0.640000   -0.009179
      50     0.551397     0.562500   -0.011103
      60     0.477141     0.490000   -0.012859
error at t = 60 min for smaller steps:
  step 10.00 min   error -0.012859
  step  5.00 min   error -0.006335  ratio 2.03
  step  2.50 min   error -0.003144  ratio 2.01
  step  1.25 min   error -0.001566  ratio 2.01
```

Setiap nilai Euler berada di bawah nilai eksak, dan galatnya tumbuh terus, sampai 1,3 cm setelah satu jam. Alasannya sama dengan yang kita lihat dengan tangan: aliran keluar selalu diambil di awal langkah, ketika ketinggian, dan karena itu alirannya, paling besar, sehingga setiap langkah mengeluarkan terlalu banyak. Tabel kedua memastikan ordenya: setiap pembagian dua langkah membagi dua galatnya (rasio 2,0). **Gambar 11.1** memperlihatkan titik-titik Euler perlahan jatuh di bawah kurva eksak, di samping titik-titik metode RK4 dari subbab berikutnya, yang tidak dapat dibedakan dari kurvanya.



Gambar 11.1 Tandon yang mengosong: solusi eksak, metode Euler dengan langkah 10 menit (setiap langkah mengikuti garis singgung di awalnya dan mengeluarkan air sedikit terlalu banyak), dan metode Runge–Kutta klasik dengan langkah yang sama.

Bagi Anto, 1,3 cm adalah 13 liter, bukan bencana. Tetapi orde satu itu mahal: untuk menekan galatnya sampai satu milimeter, Euler memerlukan langkah sekitar satu menit; untuk sepersepuluh milimeter, sekitar lima detik. Kita dapat berbuat jauh lebih baik dengan banyak evaluasi kemiringan yang sama dengan melihat lebih cermat di dalam setiap langkah.

11.5 Metode Runge–Kutta

11.5.1 Orde dua: Heun dan titik tengah

Masalah Euler adalah bahwa ia hanya memakai kemiringan di awal langkah. Kemiringan rata-rata selama langkah itu akan tepat benar, tetapi tidak diketahui. Kita dapat menaksirnya. **Metode**

Heun mula-mula mengambil langkah Euler percobaan sampai ujung selang, mengevaluasi kemiringan di sana, lalu melangkah dengan rata-rata kedua kemiringan itu:

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t_n + h,\ y_n + hk_1), \qquad y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2).$$

Metode titik tengah (*midpoint*) sebaliknya mengambil setengah langkah Euler, mengevaluasi kemiringan di tengah, lalu memakainya untuk seluruh langkah: $k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right)$ dan $y_{n+1} = y_n + hk_2$. Keduanya memakai dua evaluasi kemiringan per langkah, dua kali pekerjaan Euler. Keduanya berorde dua. Inilah alasannya, untuk metode dua tahap yang umum

$$k_1 = f(t_n, y_n), \quad k_2 = f(t_n + ch,\ y_n + ahk_1), \quad y_{n+1} = y_n + h(b_1k_1 + b_2k_2). \tag{11.6}$$

Menurut aturan rantai, solusi eksak memenuhi $y'' = f_t + ff_y$, dengan f_t dan f_y turunan parsial f, sehingga ekspansi Taylornya $y(t+h) = y + hf + \frac{h^2}{2}(f_t + ff_y) + O(h^3)$. Mengekspansikan k_2 dengan deret Taylor dua peubah memberi $k_2 = f + chf_t + ahk_1f_y + O(h^2)$, sehingga metode itu memberi $y + h(b_1 + b_2)f + h^2b_2(cf_t + aff_y) + O(h^3)$. Mencocokkan kedua ekspansi suku demi suku menuntut

$$b_1 + b_2 = 1, \qquad b_2c = \frac{1}{2}, \qquad b_2a = \frac{1}{2}.$$

Tiga persamaan untuk empat bilangan yang tidak diketahui: satu keluarga metode orde dua dengan satu parameter. Heun adalah $b_1 = b_2 = \frac{1}{2}$, $a = c = 1$; titik tengah adalah $b_1 = 0$, $b_2 = 1$, $a = c = \frac{1}{2}$. Galat lokalnya $O(h^3)$ dan galat globalnya $O(h^2)$. Untuk langkah pertama tandon selama 10 menit, Heun memberi $k_1 = -0.01$, $k_2 = -0.01\sqrt{0.9} = -0.0094868$, dan $h_1 = 1 + 5 \times (-0.0194868) = 0.902566$, hanya 0,07 mm di atas nilai eksak 0,9025, dibandingkan Euler yang 2,5 mm di bawahnya.

11.5.2 Metode orde empat klasik

Gagasan yang sama dengan empat evaluasi kemiringan menghasilkan metode yang dikembangkan Carl Runge dan Wilhelm Kutta sekitar tahun 1900 [5] dan yang masih menjadi pilihan bawaan tak terhitung banyaknya program, yaitu **metode Runge–Kutta klasik**, atau **RK4**:

$$\begin{aligned} k_1 &= f(t_n, y_n), & k_2 &= f\left(t_n + \frac{h}{2},\ y_n + \frac{h}{2}k_1\right), \\ k_3 &= f\left(t_n + \frac{h}{2},\ y_n + \frac{h}{2}k_2\right), & k_4 &= f(t_n + h,\ y_n + hk_3), \\ y_{n+1} &= y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right). \end{aligned} \tag{11.7}$$

Metode ini mencicipi kemiringan di awal (k_1), dua kali di tengah (k_2 dari setengah langkah dengan k_1, k_3 dari setengah langkah dengan k_2) dan di akhir (k_4), lalu merata-ratakannya dengan

bobot 1,2,2,1. Mencocokkan ekspansi Taylor sampai orde empat menuntut delapan syarat, dan bobot-bobot ini memenuhinya [4], [6]; galat lokalnya $O(h^5)$ dan galat globalnya $O(h^4)$.

Bobot-bobot itu semestinya terasa akrab. Bila f hanya bergantung pada t, persamaan diferensialnya tinggal $y' = f(t)$ dan solusinya adalah sebuah integral; maka $k_2 = k_3 = f(t_n + h/2)$ dan RK4 menjadi $y_{n+1} = y_n + \frac{h}{6}\left(f(t_n) + 4f\left(t_n + \frac{h}{2}\right) + f(t_n + h)\right)$, yaitu aturan Simpson dari Bab 10 pada selang $[t_n, t_n + h]$. RK4 adalah aturan Simpson yang diperluas untuk kemiringan yang bergantung pada solusinya sendiri.

Satu langkah RK4 untuk tandon dengan tangan, dengan langkah 10 menit dari $h = 1$: $k_1 = -0.01\sqrt{1} = -0.01$; $k_2 = -0.01\sqrt{1 + 5k_1} = -0.01\sqrt{0.95} = -0.0097468$; $k_3 = -0.01\sqrt{1 + 5k_2} = -0.01\sqrt{0.951266} = -0.0097533$; $k_4 = -0.01\sqrt{1 + 10k_3} = -0.01\sqrt{0.902467} = -0.0094998$. Maka $h_1 = 1 + \frac{10}{6}(-0.01 - 0.0194936 - 0.0195066 - 0.0094998) = 1 + \frac{10}{6} \times (-0.0585000) = 0.9025000$, sama dengan nilai eksak 0,9025 sampai tujuh desimal, dengan langkah sepuluh menit.

11.5.3 Mengukur ordenya

Kode 11.3 mengimplementasikan keempat metode sebagai fungsi langkah dengan tanda tangan yang sama, sehingga satu fungsi `solve` dapat menjalankan salah satunya, lalu mencetak galat pada satu jam untuk langkah yang dibagi dua.

Kode 11.3 `ode_orders.py`: Euler, Heun, titik tengah, dan RK4 sebagai fungsi langkah yang dapat saling menggantikan, beserta rasio galat untuk langkah yang dibagi dua.

```
"""Euler, Heun, midpoint and classical Runge-Kutta (RK4) on the draining tank
of tank_euler.py: error at t = 60 min for halved steps."""
from tank_euler import slope, exact


def euler_step(f, t, y, h):
    return y + h * f(t, y)


def heun_step(f, t, y, h):
    k1 = f(t, y)
    k2 = f(t + h, y + h * k1)           # slope at the Euler prediction
    return y + h * (k1 + k2) / 2


def midpoint_step(f, t, y, h):
    k1 = f(t, y)
    k2 = f(t + h / 2, y + h / 2 * k1)   # slope half-way along
    return y + h * k2


def rk4_step(f, t, y, h):
    k1 = f(t, y)
    k2 = f(t + h / 2, y + h / 2 * k1)
    k3 = f(t + h / 2, y + h / 2 * k2)
    k4 = f(t + h, y + h * k3)
    return y + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6


def solve(step_fn, f, t0, y0, h, t_end):
    t, y = t0, y0
    for _ in range(round((t_end - t0) / h)):
        y = step_fn(f, t, y, h)
        t += h
    return y
```

Kode 11.3 (Lanjutan) *ode_orders.py: Euler, Heun, titik tengah, dan RK4 sebagai fungsi langkah yang dapat saling menggantikan, beserta rasio galat untuk langkah yang dibagi dua.*

```
def main():
    methods = [("Euler", euler_step), ("Heun", heun_step),
               ("midpoint", midpoint_step), ("RK4", rk4_step)]
    for name, fn in methods:
        print(f"{name}:")
        old = None
        for h in (10.0, 5.0, 2.5, 1.25):
            err = abs(solve(fn, slope, 0.0, 1.0, h, 60.0) - exact(60.0))
            ratio = f"  ratio {old / err:6.2f}" if old else ""
            print(f"  step {h:5.2f}   error {err:.3e}{ratio}")
            old = err


if __name__ == "__main__":
    main()
```

Dalam Java, **Kode 11.4** mendeklarasikan antarmuka kecil `Stepper` agar keempat metode langkah dapat disimpan dalam larik dan dikirim ke `solve` sebagai rujukan metode.

Kode 11.4 `OdeOrders.java`: perbandingan yang sama dalam Java.

```java
import java.util.Locale;
import java.util.function.DoubleBinaryOperator;

/** Euler, Heun, midpoint and classical Runge-Kutta (RK4) on the draining tank
 *  of TankEuler: error at t = 60 min for halved steps. */
public class OdeOrders {
    interface Stepper {
        double step(DoubleBinaryOperator f, double t, double y, double h);
    }

    static double eulerStep(DoubleBinaryOperator f, double t, double y, double h) {
        return y + h * f.applyAsDouble(t, y);
    }

    static double heunStep(DoubleBinaryOperator f, double t, double y, double h) {
        double k1 = f.applyAsDouble(t, y);
        double k2 = f.applyAsDouble(t + h, y + h * k1);  // at the Euler prediction
        return y + h * (k1 + k2) / 2;
    }

    static double midpointStep(DoubleBinaryOperator f, double t, double y,
                               double h) {
        double k1 = f.applyAsDouble(t, y);
        double k2 = f.applyAsDouble(t + h / 2, y + h / 2 * k1);  // half-way
        return y + h * k2;
    }

    static double rk4Step(DoubleBinaryOperator f, double t, double y, double h) {
        double k1 = f.applyAsDouble(t, y);
        double k2 = f.applyAsDouble(t + h / 2, y + h / 2 * k1);
        double k3 = f.applyAsDouble(t + h / 2, y + h / 2 * k2);
        double k4 = f.applyAsDouble(t + h, y + h * k3);
        return y + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    }

    static double solve(Stepper s, DoubleBinaryOperator f, double t0, double y0,
                        double h, double tEnd) {
        double t = t0, y = y0;
        long n = Math.round((tEnd - t0) / h);
        for (long i = 0; i < n; i++) {
            y = s.step(f, t, y, h);
            t += h;
        }
        return y;
    }
```

Kode 11.4 (Lanjutan) *OdeOrders.java: perbandingan yang sama dalam Java.*

```
    public static void main(String[] args) {
        String[] names = {"Euler", "Heun", "midpoint", "RK4"};
        Stepper[] steppers = {OdeOrders::eulerStep, OdeOrders::heunStep,
                              OdeOrders::midpointStep, OdeOrders::rk4Step};
        for (int m = 0; m < names.length; m++) {
            System.out.println(names[m] + ":");
            double old = 0;
            for (double h : new double[] {10.0, 5.0, 2.5, 1.25}) {
                double err = Math.abs(solve(steppers[m], TankEuler::slope, 0.0,
                                            1.0, h, 60.0) - TankEuler.exact(60.0));
                String ratio = old != 0
                    ? String.format(Locale.ROOT, "  ratio %6.2f", old / err) : "";
                System.out.println(String.format(Locale.ROOT,
                    "  step %5.2f   error %.3e%s", h, err, ratio));
                old = err;
            }
        }
    }
}
```

Keduanya mencetak:

```
Euler:
  step 10.00   error 1.286e-02
  step  5.00   error 6.335e-03  ratio   2.03
  step  2.50   error 3.144e-03  ratio   2.01
  step  1.25   error 1.566e-03  ratio   2.01
Heun:
  step 10.00   error 3.988e-04
  step  5.00   error 9.666e-05  ratio   4.13
  step  2.50   error 2.380e-05  ratio   4.06
  step  1.25   error 5.904e-06  ratio   4.03
midpoint:
  step 10.00   error 1.932e-04
  step  5.00   error 4.759e-05  ratio   4.06
  step  2.50   error 1.181e-05  ratio   4.03
  step  1.25   error 2.941e-06  ratio   4.02
RK4:
  step 10.00   error 1.535e-07
  step  5.00   error 9.345e-09  ratio  16.42
  step  2.50   error 5.762e-10  ratio  16.22
  step  1.25   error 3.577e-11  ratio  16.11
```

Rasio-rasio itu langsung memberi tahu ordenya: 2 untuk Euler (orde satu), 4 untuk Heun dan titik tengah (orde dua), 16 untuk RK4 (orde empat). Angka mutlaknya menceritakan kisah praktisnya. Dengan langkah 10 menit, enam langkah dalam satu jam, galat RK4 adalah 1.5×10^{-7} m, seperenam mikrometer, dengan 24 evaluasi kemiringan. Euler dengan 48 langkah 1,25 menit, 48 evaluasi, masih meleset 1,6 mm, sepuluh ribu kali lebih buruk untuk dua kali pekerjaan. Itulah sebabnya RK4 menjadi kuda beban bab ini dan sebagian besar komputasi praktis.

11.6 Membangun Simulator Pagi Hari

Sekarang pagi yang sebenarnya. Pukul 06.00 tandon berisi air setinggi 25 cm sisa kemarin. Pompa menambahkan 20 L/menit sejak 06.00 sampai Anto mematikannya. Katup tetes dibuka pukul 06.30 dan tetap terbuka. Tandon setinggi 1 m; air yang lebih tinggi dari itu keluar melalui pipa luapan. Bila ketinggian turun di bawah 1 cm, penetes praktis berhenti. Modelnya adalah Persamaan (11.1) dengan suku pompa yang mati dan suku tetes yang menyala.

Sakelar memerlukan kehati-hatian. Akurasi RK4 berasal dari ekspansi Taylor, yang mengandaikan f mulus *di dalam* setiap langkah. Bila pompa mati di tengah-tengah sebuah langkah, sebagian dari keempat kemiringan melihat pompa menyala dan sebagian tidak, dan langkah itu hanya berakurasi orde satu. Obatnya sederhana: jangan pernah biarkan sebuah langkah melintasi sakelar. **Kode 11.5** membagi setiap langkah yang memuat waktu pensakelaran menjadi dua potong, satu sampai sakelar dan satu sesudahnya, dan memutuskan apakah pompa dan katup menyala dari *awal* setiap potong. Program ini juga membatasi ketinggian pada luapan dan menghitung liter yang tumpah, mencegah akar kuadrat melihat ketinggian negatif, mencatat ketinggian tertinggi dan saat penetes berhenti, dan akhirnya menjalankan seluruh pagi sekali lagi dengan separuh langkah, sebagai pemeriksaan.

Kode 11.5 `morning.py`: satu pagi tandon irigasi tetes dengan RK4, termasuk sakelar pompa, luapan, dan saat kering.

```
"""One morning of the orchard's drip-irrigation tank, simulated with RK4.
The pump fills the tank at 20 L/min from 06:00 until the pump is switched off;
the drip valve opens at 06:30. Usage: python morning.py [pump-off minute]"""
import math
import sys
from ode_orders import rk4_step

AREA, HEIGHT, K = 1.0, 1.0, 0.01     # m^2, m, drip constant (m^2.5/min)
PUMP, DRIP_OPEN = 0.02, 30.0         # m^3/min, minutes after 06:00
DRY = 0.01                           # m: below 1 cm the drippers stop


def make_slope(pump_on, drip_on):
    """Pump and valve stay as they are during one step."""
    def slope(t, h):
        inflow = PUMP if pump_on else 0.0
        outflow = K * math.sqrt(max(h, 0.0)) if drip_on else 0.0
        return (inflow - outflow) / AREA
    return slope


def simulate(pump_off, h0=0.25, step=1.0, t_end=300.0):
    """Returns rows (t, h), highest level, liters spilled, minute run dry."""
    t, h, top, spilled, dry_at = 0.0, h0, h0, 0.0, None
    rows = [(t, h)]
    for _ in range(round(t_end / step)):
        cuts = [c for c in (pump_off, DRIP_OPEN) if t < c < t + step]
        start = t
        for stop in sorted(cuts) + [t + step]:   # split the step at a switch
            f = make_slope(start < pump_off, start >= DRIP_OPEN)
            h = max(rk4_step(f, start, h, stop - start), 0.0)
            start = stop
        t += step
        if h > HEIGHT:                   # the overflow pipe takes the excess
            spilled += (h - HEIGHT) * AREA * 1000
            h = HEIGHT
        top = max(top, h)
        if dry_at is None and t > pump_off and h < DRY:
            dry_at = t
        rows.append((t, h))
    return rows, top, spilled, dry_at
```

Kode 11.5 (Lanjutan) *morning.py: satu pagi tandon irigasi tetes dengan RK4, termasuk sakelar pompa, luapan, dan saat kering.*

```
def clock(minutes):
    m = int(minutes)                # whole minutes, rounded down
    return f"{6 + m // 60:02d}:{m % 60:02d}"


def main(args):
    pump_off = float(args[0]) if args else 90.0
    rows, top, spilled, dry_at = simulate(pump_off)
    print(f"pump off at {clock(pump_off)}, drip valve open from {clock(DRIP_OPEN)}")
    print(" time   level(cm)  liters")
    for t, h in rows[::30]:
        print(f"{clock(t)}   {100 * h:8.1f}  {1000 * h * AREA:6.0f}")
    print(f"highest level {100 * top:.1f} cm, spilled {spilled:.0f} liters")
    print(f"drippers stop at {clock(dry_at)}" if dry_at else "never runs dry")
    half = simulate(pump_off, step=0.5)
    print(f"check with half the step: highest {100 * half[1]:.1f} cm,"
          f" spilled {half[2]:.0f} liters, dry at {clock(half[3])}")


if __name__ == "__main__":
    main(sys.argv[1:])
```

Kode 11.6 adalah versi Java-nya. Kelas kecil `Day` membawa hasil-hasil yang dikembalikan versi Python sebagai tupel, dan `dryAt` bertipe objek `Double` agar dapat bernilai `null` bila tandon tidak pernah kering.

Kode 11.6 `Morning.java`: simulator yang sama dalam Java.

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Locale;
import java.util.function.DoubleBinaryOperator;

/** One morning of the orchard's drip-irrigation tank, simulated with RK4.
 *  The pump fills the tank at 20 L/min from 06:00 until the pump is switched
 *  off; the drip valve opens at 06:30. Usage: java Morning [pump-off minute] */
public class Morning {
    static final double AREA = 1.0, HEIGHT = 1.0, K = 0.01;   // m^2, m, drip
    static final double PUMP = 0.02, DRIP_OPEN = 30.0;        // m^3/min, minute
    static final double DRY = 0.01;          // m: below 1 cm the drippers stop

    /** Pump and valve stay as they are during one step. */
    static DoubleBinaryOperator makeSlope(boolean pumpOn, boolean dripOn) {
        return (t, h) -> {
            double inflow = pumpOn ? PUMP : 0.0;
            double outflow = dripOn ? K * Math.sqrt(Math.max(h, 0.0)) : 0.0;
            return (inflow - outflow) / AREA;
        };
    }

    /** Result of one simulated morning. */
    static class Day {
        List<double[]> rows = new ArrayList<>();
        double top, spilled;
        Double dryAt;                        // null when it never runs dry
    }

    static Day simulate(double pumpOff, double h0, double step, double tEnd) {
        Day d = new Day();
        double t = 0.0, h = h0;
        d.top = h0;
        d.rows.add(new double[] {t, h});
        long n = Math.round(tEnd / step);
        for (long i = 0; i < n; i++) {
            List<Double> stops = new ArrayList<>();
            for (double c : new double[] {pumpOff, DRIP_OPEN}) {
                if (t < c && c < t + step) {
                    stops.add(c);
                }
            }
            stops.sort(null);
            stops.add(t + step);
            double start = t;
```

Kode 11.6 (Lanjutan) *Morning.java: simulator yang sama dalam Java.*

```
            for (double stop : stops) {             // split the step at a switch
                DoubleBinaryOperator f = makeSlope(start < pumpOff,
                                                   start >= DRIP_OPEN);
                h = Math.max(OdeOrders.rk4Step(f, start, h, stop - start), 0.0);
                start = stop;
            }
            t += step;
            if (h > HEIGHT) {                       // the overflow pipe takes it
                d.spilled += (h - HEIGHT) * AREA * 1000;
                h = HEIGHT;
            }
            d.top = Math.max(d.top, h);
            if (d.dryAt == null && t > pumpOff && h < DRY) {
                d.dryAt = t;
            }
            d.rows.add(new double[] {t, h});
        }
        return d;
    }

    static Day simulate(double pumpOff) {
        return simulate(pumpOff, 0.25, 1.0, 300.0);
    }

    static String clock(double minutes) {
        long m = (long) Math.floor(minutes);      // whole minutes, rounded down
        return String.format(Locale.ROOT, "%02d:%02d", 6 + m / 60, m % 60);
    }

    public static void main(String[] args) {
        double pumpOff = args.length > 0 ? Double.parseDouble(args[0]) : 90.0;
        Day d = simulate(pumpOff);
        System.out.println("pump off at " + clock(pumpOff)
                           + ", drip valve open from " + clock(DRIP_OPEN));
        System.out.println(" time   level(cm)  liters");
        for (int i = 0; i < d.rows.size(); i += 30) {
            double[] r = d.rows.get(i);
            System.out.println(String.format(Locale.ROOT, "%s   %8.1f  %6.0f",
                clock(r[0]), 100 * r[1], 1000 * r[1] * AREA));
        }
        System.out.println(String.format(Locale.ROOT,
            "highest level %.1f cm, spilled %.0f liters", 100 * d.top, d.spilled));
        System.out.println(d.dryAt != null ? "drippers stop at " + clock(d.dryAt)
                                           : "never runs dry");
        Day half = simulate(pumpOff, 0.25, 0.5, 300.0);
        System.out.println(String.format(Locale.ROOT,
```

Kode 11.6 (Lanjutan) *Morning.java: simulator yang sama dalam Java.*

```
            "check with half the step: highest %.1f cm, spilled %.0f liters,"
            + " dry at %s", 100 * half.top, half.spilled, clock(half.dryAt)));
    }
}
```

Dengan pompa dibiarkan menyala sampai 07.30, seperti pada pagi yang pelupa, keduanya mencetak:

```
pump off at 07:30, drip valve open from 06:30
 time   level(cm)  liters
06:00       25.0     250
06:30       85.0     850
07:00      100.0    1000
07:30      100.0    1000
08:00       72.3     723
08:30       49.0     490
09:00       30.3     303
09:30       16.0     160
10:00        6.3      63
10:30        1.0      10
11:00        0.0       0
highest level 100.0 cm, spilled 454 liters
drippers stop at 10:31
check with half the step: highest 100.0 cm, spilled 455 liters, dry at 10:30
```

Tandon penuh sekitar pukul 06.44 dan bertahan di bibirnya selama tiga perempat jam sementara 454 liter tumpah dari pipa luapan, hampir setengah tandon, bersama listrik yang dipakai untuk memompanya. Setelah pompa mati, ketinggian turun mengikuti kurva Persamaan (11.3), dan penetes berhenti sekitar pukul 10.30.

Angka terakhir itu dapat kita periksa secara eksak. Sejak 07.30 pompa mati dan tandon mulai dari penuh, sehingga Persamaan (11.3) berlaku dengan $h_0 = 1$ dan jam dimulai pada 07.30: $h = (1 - 0.005\,s)^2$ dengan s waktu sejak 07.30. Ketinggian mencapai 1 cm ketika $1 - 0.005\,s = 0.1$, yaitu $s = 180$ menit: tepat pukul 10.30. Simulasi dengan langkah satu menit melaporkan 10.31, karena ia hanya memeriksa ketinggian pada menit-menit bulat dan pada 10.30 ketinggiannya sedikit di atas 1 cm; dengan langkah setengah menit ia melaporkan 10.30. Volume yang tumpah, 454 atau 455 liter, sedikit berbeda karena alasan yang sama: luapan ditangani di akhir setiap langkah. Inilah kebiasaan berguna yang perlu dipelajari di sini: **jalankan simulasinya lagi dengan separuh langkah**. Bila jawaban yang Anda pedulikan tidak berubah, langkahnya sudah cukup kecil; bila berubah, selisihnya adalah taksiran galat, persis seperti di Bab 9 dan 10.

Waktu yang dilaporkan melompat antara 10.30 dan 10.31 juga memperlihatkan cara berbuat lebih baik bila *saat* terjadinya suatu kejadian penting. Alih-alih hanya bertanya "apakah ketinggiannya di bawah 1 cm pada menit bulat ini?", program dapat mengenali menit tempat ketinggian melintasi 1 cm lalu menentukan letak perlintasan itu di dalam menit tersebut, dengan interpolasi linear di antara kedua ketinggian atau, lebih teliti lagi, dengan biseksi pada langkah RK4 parsial, persis seperti di Bab 2. `solve_ivp` milik SciPy melakukannya untuk Anda melalui argumen `events`: Anda memberinya fungsi yang berganti tanda pada kejadian itu, misalnya ketinggian dikurangi 0,01, dan ia mengembalikan waktu perlintasannya dengan akurasi penuh.

Argumen baris perintah memungkinkan Anto mencoba rencana lain; `python3 morning.py 45` mematikan pompa pukul 06.45:

```
pump off at 06:45, drip valve open from 06:30
 time   level(cm)  liters
06:00       25.0     250
06:30       85.0     850
07:00       85.6     856
07:30       60.1     601
08:00       39.1     391
08:30       22.6     226
09:00       10.6     106
09:30        3.1      31
10:00        0.1       1
10:30        0.0       0
11:00        0.0       0
highest level 100.0 cm, spilled 5 liters
drippers stop at 09:46
check with half the step: highest 100.0 cm, spilled 5 liters, dry at 09:45
```

Sekarang hanya 5 liter yang meluap, tetapi penetes berhenti pukul 09.46, tiga perempat jam lebih awal. **Gambar 11.2** menggambar kedua pagi itu. Kurva terencana dari subbab berikutnya berada di antara keduanya.

11.7 Menyempurnakannya: Merencanakan Pompa

Pertanyaan Anto yang sebenarnya adalah pertanyaan kebalikan: *pada jam berapa pompa harus mati agar tandon terisi sampai 98 cm, dua sentimeter di bawah luapan, tetapi tidak lebih tinggi?* Simulator menjawab pertanyaan maju, “bila jam mati pompa diketahui, seberapa tinggi tandon terisi?”, sehingga kita memerlukan metode pencarian akar yang membungkusnya. Fungsi $g(T) = \text{puncak}(T) - 0.98$, dengan $\text{puncak}(T)$ ketinggian tertinggi bila pompa mati pada menit T, bernilai negatif untuk pompa yang mati terlalu awal dan positif untuk yang terlambat. Persis itulah latar metode biseksi dari Bab 2, yang tidak memerlukan apa pun selain tanda g. Setiap evaluasi g adalah satu simulasi RK4 seluruh pagi, yang memakan beberapa milidetik. **Kode 11.7** menjalankan dua belas langkah biseksi antara 06.30 dan 07.30.

Kode 11.7 `pump_plan.py`: biseksi pada jam mati pompa, dengan simulasi RK4 sebagai fungsi yang akarnya dicari.

```
"""When should the pump be switched off so that the tank fills to 98 cm but
never overflows? Bisection (Chapter 2) around the RK4 simulation."""
from morning import simulate, clock

TARGET = 0.98                          # m: leave 2 cm below the overflow


def top_level(pump_off):
    return simulate(pump_off)[1]


def main():
    lo, hi = 30.0, 90.0                # 06:30 is too early, 07:30 too late
    print(" step   pump off   highest level(cm)")
    for i in range(1, 13):
        mid = (lo + hi) / 2
        top = top_level(mid)
        print(f"{i:5d}   {mid:8.3f}   {100 * top:10.2f}")
        if top > TARGET:
            hi = mid
        else:
            lo = mid
    best = (lo + hi) / 2
    rows, top, spilled, dry_at = simulate(best)
    print(f"switch the pump off after {best:.1f} min, at about {clock(best)}:"
          f" top {100 * top:.1f} cm, spilled {spilled:.0f} liters,"
          f" drippers stop at {clock(dry_at)}")


if __name__ == "__main__":
    main()
```

Kode 11.8 `PumpPlan.java`: perencana yang sama dalam Java.

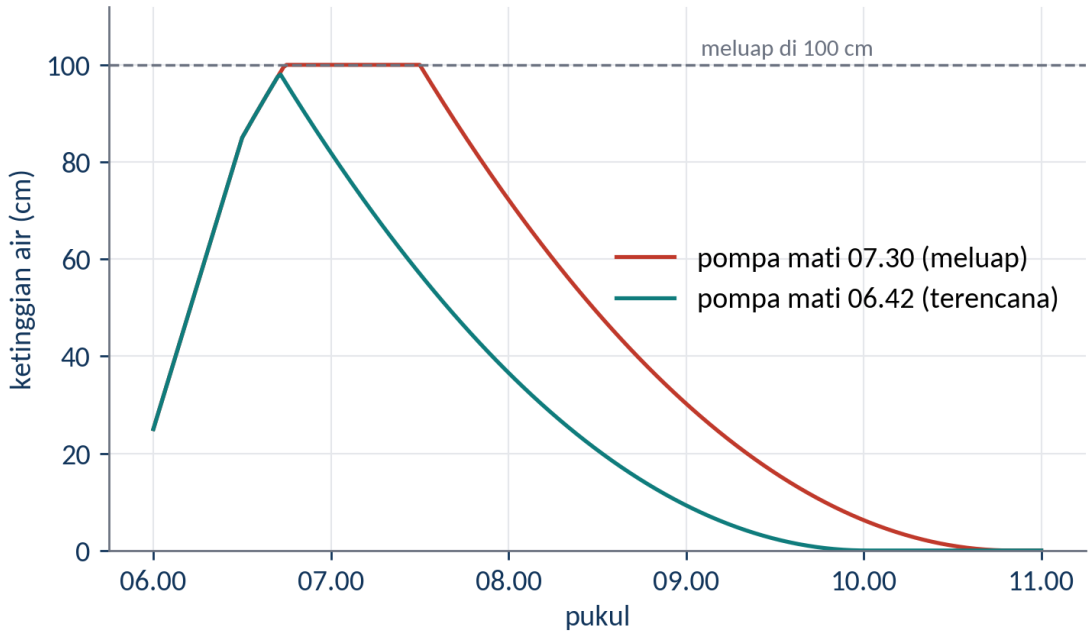
```
import java.util.Locale;

/** When should the pump be switched off so that the tank fills to 98 cm but
 *  never overflows? Bisection (Chapter 2) around the RK4 simulation. */
public class PumpPlan {
    static final double TARGET = 0.98;       // m: leave 2 cm below the overflow

    public static void main(String[] args) {
        double lo = 30.0, hi = 90.0;         // 06:30 is too early, 07:30 too late
        System.out.println(" step   pump off   highest level(cm)");
        for (int i = 1; i <= 12; i++) {
            double mid = (lo + hi) / 2;
            double top = Morning.simulate(mid).top;
            System.out.println(String.format(Locale.ROOT, "%5d   %8.3f   %10.2f",
                i, mid, 100 * top));
            if (top > TARGET) {
                hi = mid;
            } else {
                lo = mid;
            }
        }
        double best = (lo + hi) / 2;
        Morning.Day d = Morning.simulate(best);
        System.out.println(String.format(Locale.ROOT,
            "switch the pump off after %.1f min, at about %s: top %.1f cm,"
            + " spilled %.0f liters, drippers stop at %s", best,
            Morning.clock(best), 100 * d.top, d.spilled, Morning.clock(d.dryAt)));
    }
}
```

Kode 11.8 mencetak pencarian yang sama:

```
 step   pump off   highest level(cm)
    1     60.000        100.00
    2     45.000        100.00
    3     37.500         92.44
    4     41.250         96.52
    5     43.125         98.54
    6     42.188         97.53
    7     42.656         97.86
    8     42.891         98.33
    9     42.773         98.09
   10     42.715         97.97
   11     42.744         98.03
   12     42.729         98.00
switch the pump off after 42.7 min, at about 06:42: top 98.0 cm, spilled 0 liters, drippers
stop at 09:41
```

Setelah dua belas kali pembagian dua, lebar selangnya $60/2^{12} \approx 0.015$ menit, jauh lebih halus daripada ketelitian siapa pun mematikan pompa. Jawabannya: matikan pompa 42,7 menit setelah pukul enam, sekitar pukul 06.42. Tandon lalu memuncak di 98 cm, tidak ada yang meluap, dan penetes bekerja sampai 09.41. Membagi langkah tepat pada waktu pensakelaran, seperti yang dilakukan `simulate`, penting di sini: tanpa itu, setiap jam mati pompa di antara dua menit bulat akan berperilaku seperti menit bulat berikutnya, fungsi g akan bergerak meloncat-loncat, dan biseksi akan konvergen ke batas langkah, bukan ke jawaban sebenarnya.

Perencana ini juga menyingkap tawar-menawar yang selama ini hanya dirasakan Anto. Dengan pompa 20 L/menit, mematikannya pukul 06.42 mencegah pemborosan tetapi mengakhiri penyiraman pukul 09.41; menjalankan pompa lebih lama membuat tandon bertahan lebih lama tetapi menumpahkan air. Jalan keluarnya adalah pompa yang mengalirkan lebih sedikit air untuk waktu lebih lama, misalnya dengan menutup sebagian katup di keluaran pompa. Soal Latihan 11.4 mencari debit yang mengisi tandon sampai 98 cm dengan pompa menyala sampai 07.30, dan memperlihatkan bahwa penetes lalu bekerja sampai hampir 10.30 tanpa luapan sama sekali. **Gambar 11.2** memperlihatkan pagi yang terencana dibandingkan pagi yang pelupa.



Gambar 11.2 Ketinggian air pada dua pagi: dengan pompa dibiarkan menyala sampai 07.30 tandon meluap selama tiga perempat jam; dengan pompa dimatikan pukul 06.42, seperti direncanakan dengan biseksi, ketinggiannya memuncak di 98 cm dan tidak pernah meluap.

11.8 Sistem dan Kestabilan: Jok Sepeda Motor

11.8.1 Dari satu persamaan ke sebuah sistem

Banyak masalah melibatkan beberapa besaran yang berubah bersama, atau turunan kedua. Bayangkan sepeda motor Mas Dimas dari Bab 9 melewati polisi tidur. Suspensi belakangnya adalah pegas dengan peredam kejut (*shock absorber*). Bila $x(t)$ adalah seberapa jauh jok tertekan ke bawah dari posisi diamnya, hukum kedua Newton memberi

$$mx'' + cx' + kx = 0,$$

dengan m massa yang ditanggung suspensi (sekitar 100 kg motor dan pengendara), k kekakuan pegas (sekitar 20.000 N/m), dan c redaman peredam kejut: sekitar 1.000 N·s/m untuk yang baik, jauh lebih kecil untuk yang aus, misalnya 150 N·s/m. Setiap metode numerik dalam bab ini menyelesaikan persamaan *orde satu*. Caranya adalah memperkenalkan kecepatan $v = x'$ sebagai besaran tak diketahui yang kedua:

$$x' = v, \qquad v' = -\frac{cv + kx}{m}. \tag{11.8}$$

Itulah **sistem** dua persamaan orde satu untuk vektor $y = (x, v)$, persis berbentuk Persamaan (11.2), dengan f kini mengembalikan vektor. Euler dan RK4 berlaku tanpa perubahan, komponen demi komponen; setiap k_i pada Persamaan (11.7) menjadi vektor. Setiap persamaan berorde lebih tinggi dapat diubah menjadi sistem orde satu dengan cara yang sama.

Kode 11.9 menyimulasikan dua detik setelah polisi tidur, mulai dari jok yang tertekan 5 cm dan diam, untuk kedua peredam kejut, dengan Euler dan RK4 pada langkah 0,01 s, serta dengan RK4 pada 0,001 s sebagai rujukan.

Kode 11.9 `bump.py`: jok setelah polisi tidur sebagai sistem dua persamaan, dengan Euler dan RK4.

```
"""A motorbike seat after a speed bump: m x'' + c x' + k x = 0 as a system of
two first-order equations, solved with Euler and with RK4."""

MASS, SPRING = 100.0, 20000.0        # kg, N/m
SHOCKS = [("good shock", 1000.0), ("worn shock", 150.0)]   # c in N s/m


def make_rhs(damping):
    def rhs(t, y):
        x, v = y
        return [v, -(damping * v + SPRING * x) / MASS]
    return rhs


def euler_step(f, t, y, h):
    k1 = f(t, y)
    return [y[i] + h * k1[i] for i in range(len(y))]


def rk4_step(f, t, y, h):
    k1 = f(t, y)
    k2 = f(t + h / 2, [y[i] + h / 2 * k1[i] for i in range(len(y))])
    k3 = f(t + h / 2, [y[i] + h / 2 * k2[i] for i in range(len(y))])
    k4 = f(t + h, [y[i] + h * k3[i] for i in range(len(y))])
    return [y[i] + h * (k1[i] + 2 * k2[i] + 2 * k3[i] + k4[i]) / 6
            for i in range(len(y))]


def trace(step_fn, f, h, t_end=2.0, every=0.5):
    """Seat position (mm) at t = every, 2*every, ... up to t_end."""
    y, out = [0.05, 0.0], []         # 5 cm down, at rest, just after the bump
    per = round(every / h)
    for n in range(1, round(t_end / h) + 1):
        y = step_fn(f, (n - 1) * h, y, h)
        if n % per == 0:
            out.append(1000 * y[0])
    return out
```

Kode 11.9 (Lanjutan) *bump.py: jok setelah polisi tidur sebagai sistem dua persamaan, dengan Euler dan RK4.*

```
def main():
    print("seat position (mm) at t =        0.5 s     1.0 s     1.5 s     2.0 s")
    for name, c in SHOCKS:
        f = make_rhs(c)
        for label, fn, h in (("Euler h=0.01 ", euler_step, 0.01),
                             ("RK4   h=0.01 ", rk4_step, 0.01),
                             ("RK4   h=0.001", rk4_step, 0.001)):
            cells = "".join(f"{x:10.3f}" for x in trace(fn, f, h))
            print(f"{name}, {label}    {cells}")


if __name__ == "__main__":
    main()
```

Kode 11.10 `Bump.java`: sistem yang sama dalam Java, dengan operasi vektor ditulis dalam fungsi bantu `axpy`.

```java
import java.util.Locale;

/** A motorbike seat after a speed bump: m x'' + c x' + k x = 0 as a system of
 *  two first-order equations, solved with Euler and with RK4. */
public class Bump {
    static final double MASS = 100.0, SPRING = 20000.0;      // kg, N/m
    static final String[] NAMES = {"good shock", "worn shock"};
    static final double[] DAMPING = {1000.0, 150.0};         // c in N s/m

    interface Rhs {
        double[] f(double t, double[] y);
    }

    interface Stepper {
        double[] step(Rhs f, double t, double[] y, double h);
    }

    static Rhs makeRhs(double damping) {
        return (t, y) -> new double[] {y[1],
                                       -(damping * y[1] + SPRING * y[0]) / MASS};
    }

    /** y + a * k, element by element. */
    static double[] axpy(double[] y, double a, double[] k) {
        double[] r = new double[y.length];
        for (int i = 0; i < y.length; i++) {
            r[i] = y[i] + a * k[i];
        }
        return r;
    }

    static double[] eulerStep(Rhs f, double t, double[] y, double h) {
        return axpy(y, h, f.f(t, y));
    }

    static double[] rk4Step(Rhs f, double t, double[] y, double h) {
        double[] k1 = f.f(t, y);
        double[] k2 = f.f(t + h / 2, axpy(y, h / 2, k1));
        double[] k3 = f.f(t + h / 2, axpy(y, h / 2, k2));
        double[] k4 = f.f(t + h, axpy(y, h, k3));
        double[] r = new double[y.length];
        for (int i = 0; i < y.length; i++) {
            r[i] = y[i] + h * (k1[i] + 2 * k2[i] + 2 * k3[i] + k4[i]) / 6;
        }
        return r;
    }
```

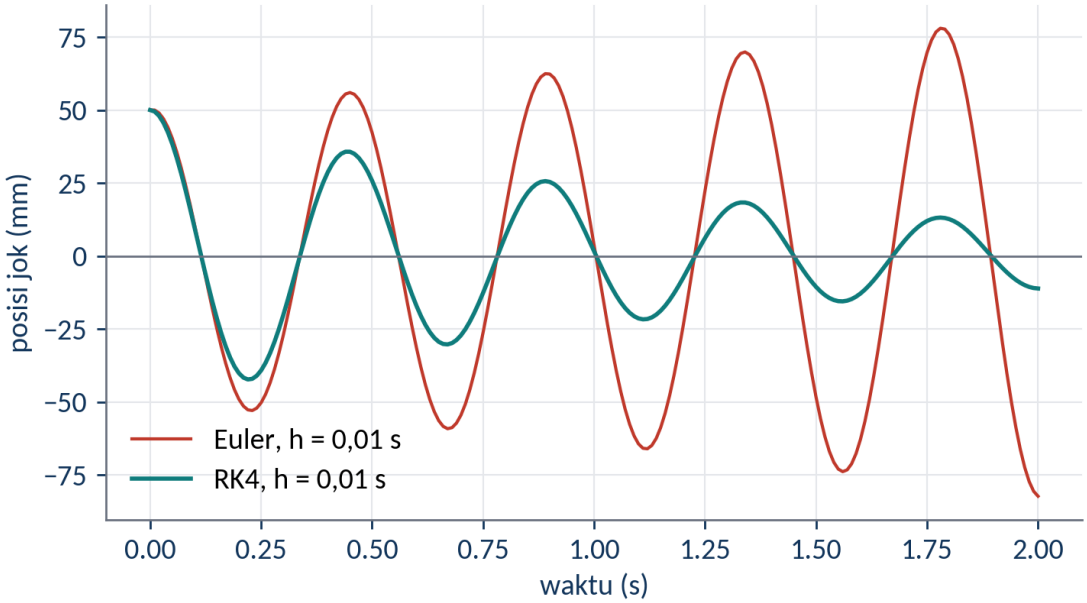
Kode 11.10 (Lanjutan) *Bump.java: sistem yang sama dalam Java, dengan operasi vektor ditulis dalam fungsi bantu axpy.*

```
    /** Seat position (mm) at t = 0.5, 1.0, 1.5 and 2.0 s. */
    static double[] trace(Stepper s, Rhs f, double h) {
        double[] y = {0.05, 0.0};          // 5 cm down, at rest, after the bump
        double[] out = new double[4];
        long per = Math.round(0.5 / h), total = Math.round(2.0 / h);
        for (long n = 1; n <= total; n++) {
            y = s.step(f, (n - 1) * h, y, h);
            if (n % per == 0) {
                out[(int) (n / per) - 1] = 1000 * y[0];
            }
        }
        return out;
    }

    public static void main(String[] args) {
        System.out.println(
            "seat position (mm) at t =        0.5 s     1.0 s     1.5 s     2.0 s");
        String[] labels = {"Euler h=0.01 ", "RK4   h=0.01 ", "RK4   h=0.001"};
        Stepper[] steppers = {Bump::eulerStep, Bump::rk4Step, Bump::rk4Step};
        double[] steps = {0.01, 0.01, 0.001};
        for (int s = 0; s < NAMES.length; s++) {
            Rhs f = makeRhs(DAMPING[s]);
            for (int m = 0; m < labels.length; m++) {
                StringBuilder cells = new StringBuilder();
                for (double x : trace(steppers[m], f, steps[m])) {
                    cells.append(String.format(Locale.ROOT, "%10.3f", x));
                }
                System.out.println(NAMES[s] + ", " + labels[m] + "    " + cells);
            }
        }
    }
}
```

Kode 11.10 dan program Python mencetak:

```
seat position (mm) at t =        0.5 s     1.0 s     1.5 s     2.0 s
good shock, Euler h=0.01          6.401     0.509     0.003    -0.007
good shock, RK4   h=0.01          4.386     0.344     0.024     0.001
good shock, RK4   h=0.001         4.386     0.344     0.024     0.001
worn shock, Euler h=0.01         42.236     3.606   -48.413   -82.305
worn shock, RK4   h=0.01         25.762     1.608   -10.594   -11.134
worn shock, RK4   h=0.001        25.761     1.607   -10.595   -11.134
```

Dengan peredam kejut yang baik semua metode kurang lebih sepakat, dan jok sudah berada dalam setengah milimeter dari posisi diam setelah satu detik. Dengan peredam yang aus, RK4 pada kedua langkah sepakat sampai seperseribu milimeter: jok masih berayun sekitar satu sentimeter setelah dua detik, yang dirasakan pengendara sebagai kendaraan yang limbung dan mengayun. Euler mengatakan hal yang sangat berbeda: ayunannya *membesar*, sampai lebih dari 8 cm setelah dua detik, lebih besar daripada 5 cm yang ditimbulkan polisi tidur itu. Pegas nyata dengan peredam nyata tidak mungkin berbuat demikian; energi hanya dapat dibuang. **Gambar 11.3** memperlihatkan ayunan Euler yang membesar sementara ayunan sebenarnya meredam.



Gambar 11.3 Jok dengan peredam kejut yang aus setelah polisi tidur: RK4 memperlihatkan ayunan yang meredam, sedangkan metode Euler dengan langkah yang sama membuatnya membesar, sebuah ketidakstabilan metode, bukan ketidakstabilan sepeda motornya.

11.8.2 Mengapa Euler meledak: kestabilan

Untuk memahaminya, terapkan metode Euler pada persamaan paling sederhana yang meluruh, yaitu **persamaan uji** $y' = \lambda y$ dengan konstanta λ yang bagian riilnya negatif. Solusi eksak $y_0 e^{\lambda t}$ meluruh ke nol. Euler memberi $y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda)y_n$, sehingga $y_n = (1 + h\lambda)^n y_0$, yang meluruh hanya bila

$$|1 + h\lambda| \le 1. \tag{11.9}$$

Untuk λ riil negatif ini berarti \$h /\$: langkah yang terlalu besar membuat solusi numerik berosilasi dan membesar, betapa pun kecil taksiran galat lokalnya. Himpunan nilai $h\lambda$ yang memenuhi Persamaan (11.9) adalah **daerah kestabilan mutlak** metode Euler: sebuah cakram berjari-jari 1 yang berpusat di -1 pada bidang kompleks [7].

Untuk suspensi, sistem Persamaan (11.8) berperilaku seperti persamaan uji dengan dua nilai kompleks $\lambda = -\zeta\omega \pm i\,\omega\sqrt{1-\zeta^2}$, dengan $\omega = \sqrt{k/m}$ frekuensi sudut alaminya dan $\zeta = c/\left(2\sqrt{km}\right)$ rasio redamannya. Di sini $\omega = \sqrt{200} = 14.14$ rad/s (jok memantul sekitar 2,25 kali per detik), $\zeta = 0.354$ untuk peredam yang baik dan $\zeta = 0.053$ untuk yang aus. Maka

$$|\,1 + h\lambda\,|^2 = (1 - h\zeta\omega)^2 + h^2\omega^2(1 - \zeta^2) = 1 - 2h\zeta\omega + h^2\omega^2,$$

yang paling besar 1 hanya bila $h \le 2\zeta/\omega$. Untuk peredam yang baik itu berarti $h \le 0.05$ s, dan langkah kita 0,01 s stabil, walaupun tidak akurat. Untuk yang aus itu berarti $h \le 0.0075$ s, dan 0,01 s berada *di luar* daerah kestabilan: setiap langkah mengalikan ayunan dengan $\sqrt{1 - 2 \times 0.01 \times 0.75 + 0.02} = \sqrt{1.005} \approx 1.0025$, dan 200 langkah mengalikannya dengan sekitar 1,65. Osilasi dengan redaman ringan adalah kasus terburuk bagi Euler: makin kecil redaman sistemnya, makin kecil langkah yang harus dipakai. Daerah kestabilan RK4 jauh lebih besar dan menjangkau sumbu imajiner sampai sekitar $|h\lambda| \le 2.8$; dengan $h\omega = 0.14$ kita nyaman berada di dalamnya.

> **PERINGATAN:** Metode yang tidak stabil tidak mengumumkan dirinya dengan pesan galat; ia menghasilkan angka yang mulus, tampak masuk akal, dan keliru. Suspensi simulasi yang memantul makin tinggi, ketinggian tandon simulasi yang berosilasi, populasi yang menjadi negatif: bila sebuah simulasi melakukan sesuatu yang tidak mungkin secara fisika, curigai ukuran langkahnya sebelum mencurigai fisikanya, dan bagi dua langkahnya untuk melihat apakah perilaku itu bertahan.

Sebagian masalah memuat proses yang sangat cepat sekaligus yang sangat lambat, seperti reaksi kimia dengan laju yang berbeda sejuta kali lipat, atau rangkaian listrik dengan kapasitor yang sangat kecil. Kestabilan lalu memaksa metode eksplisit seperti Euler atau RK4 mengambil langkah yang sangat kecil demi proses yang cepat, lama setelah proses itu padam. Masalah seperti itu disebut **kaku** (*stiff*), dan diselesaikan dengan metode *implisit*, seperti metode Euler mundur $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$, yang memerlukan penyelesaian sebuah persamaan (Bab 2–5) pada setiap langkah tetapi stabil untuk langkah berapa pun pada masalah yang meluruh [7]. `solve_ivp` milik SciPy menyediakannya sebagai `method="Radau"`, `"BDF"`, dan `"LSODA"`.

11.9 Memeriksa terhadap Pustaka

`scipy.integrate.solve_ivp` milik SciPy adalah alat Python baku untuk masalah nilai awal. Rilis terbarunya, SciPy 1.18.1, keluar pada 21 Agustus 2026 [8]. Metode bawaannya, `"RK45"`, adalah pasangan Dormand–Prince [9]: rumus Runge–Kutta orde lima dan orde empat yang berbagi evaluasi kemiringan. Selisih kedua hasilnya menaksir galat setiap langkah, dan langkahnya diperbesar atau diperkecil secara otomatis agar taksiran itu tetap di bawah `rtol` kali solusinya ditambah `atol`. Inilah gagasan pemeriksaan separuh langkah kita, diterapkan pada setiap langkah dan nyaris tanpa biaya tambahan. `"DOP853"` adalah metode orde delapan dari keluarga yang sama untuk akurasi tinggi [4], [10]. **Kode 11.11** menyelesaikan tandon yang mengosong dan jok dengan peredam aus memakainya.

Kode 11.11 `compare_scipy.py`: tandon dan jok dengan `solve_ivp`.

```
"""Check this chapter's answers with scipy.integrate.solve_ivp."""
import numpy as np
import scipy
from scipy.integrate import solve_ivp
from tank_euler import slope, exact
from bump import make_rhs

print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
for rtol in (1e-3, 1e-6, 1e-9):                       # RK45 is the default
    sol = solve_ivp(lambda t, y: [slope(t, y[0])], (0, 60), [1.0],
                    rtol=rtol, atol=1e-12)
    err = abs(sol.y[0, -1] - exact(60))
    print(f"tank, RK45 rtol={rtol:.0e}: h(60) = {sol.y[0, -1]:.9f},"
          f" error {err:.1e}, {sol.nfev} slope evaluations")
sol = solve_ivp(make_rhs(150.0), (0, 2), [0.05, 0.0], method="DOP853",
                rtol=1e-10, atol=1e-12, t_eval=[0.5, 1.0, 1.5, 2.0])
cells = "  ".join(f"{1000 * x:.3f}" for x in sol.y[0])
print(f"worn shock, DOP853, seat position (mm): {cells}")
```

```
NumPy 2.4.4, SciPy 1.17.1
tank, RK45 rtol=1e-03: h(60) = 0.490000035, error 3.5e-08, 26 slope evaluations
tank, RK45 rtol=1e-06: h(60) = 0.490000020, error 2.0e-08, 38 slope evaluations
tank, RK45 rtol=1e-09: h(60) = 0.490000000, error 5.7e-11, 68 slope evaluations
worn shock, DOP853, seat position (mm): 25.761  1.607  -10.595  -11.134
```

Pemecah adaptif itu mencapai ketinggian eksak Persamaan (11.3) pada satu jam sampai 3.5×10^{-8} m dengan 26 evaluasi kemiringan pada toleransi paling longgar, dan sampai 6×10^{-11}

m dengan 68 evaluasi pada toleransi paling ketat. Posisi jok dari `DOP853` sama dengan rujukan RK4 kita sampai digit terakhir yang dicetak (25,761, 1,607, −10.595, −11.134 mm). Perhatikan pembungkus kecil di sekitar `slope`: `solve_ivp` mengirim keadaan sebagai larik NumPy bahkan untuk satu persamaan, dan mengharapkan kemiringan berupa larik sebagai balasannya.

Prinsip di balik langkah adaptif layak dinyatakan sebagai rumus, karena berlaku untuk metode apa pun berorde p, termasuk metode kita sendiri. Bila y_h dan $y_{h/2}$ adalah hasil pada waktu yang sama dengan langkah h dan langkah $h/2$, maka

$$y(t) - y_{h/2} \approx \frac{y_{h/2} - y_h}{2^p - 1}, \tag{11.10}$$

sekali lagi taksiran Richardson dari Bab 9 dan 10. Untuk RK4 pada tandon di satu jam, $y_{10} = 0.4900001535$ dan $y_5 = 0.4900000093$, sehingga taksiran galat y_5 adalah $(0.4900000093 - 0.4900001535)/15 = -9.6 \times 10^{-9}$, dibandingkan galat sebenarnya -9.3×10^{-9}.

Folder kode juga memuat pengujian (`test_ode.py`, `OdeTest.java`) yang memeriksa fakta-fakta yang harus berlaku: RK4 eksak untuk $y' = 3t^2$ (ia tereduksi menjadi aturan Simpson, yang eksak untuk kubik); ketiga metode memperlihatkan ordenya pada tandon; jok yang diam tetap diam; dan dengan katup masih tertutup, ketinggian tandon naik tepat sebesar volume yang dipompa.

11.10 Metode Numerik dalam Praktik: Profitina

Persamaan diferensial muncul di dunia Profitina di mana pun suatu besaran berubah dengan laju yang bergantung pada besaran itu sendiri. Koin utilitas PROFIT dirancang dengan emisi yang meluruh: laju pelepasan koin baru turun seiring waktu, dan sebagian koin dibakar. Laju yang meluruh adalah solusi persamaan seperti $E' = -\lambda E$, dan total pasokannya lalu mematuhi $S' = E(t) - B(t)$, emisi dikurangi pembakaran. Bila jadwalnya berupa rumus yang bersih, pasokan dapat dihitung secara eksak; bila digabungkan dengan pembakaran nyata yang tidak teratur, pasokan dihitung dengan melangkah maju dalam waktu persis seperti tandon tadi, dan pelajaran tandon itu terbawa: kejadian sakelar tidak boleh dilintasi sebuah langkah, dan proyeksi perlu diperiksa dengan membagi dua langkahnya.

Contoh kedua, yang kurang kentara, adalah penghalusan. Banyak dasbor, termasuk tampilan tren seperti riwayat skor mingguan Profitina dan orakel harga seperti kurs PROFIT↔IDR, menampilkan nilai yang dihaluskan yang mengikuti pengukuran mentah dengan sedikit keterlambatan. "Rata-rata bergerak eksponensial" yang lazim, $y_{n+1} = y_n + \alpha(x_n - y_n)$, persis metode Euler dengan langkah h yang diterapkan pada $y' = (x(t) - y)/\tau$ dengan $\alpha = h/\tau$, sebuah keterlambatan orde satu. Dilihat dengan cara itu, perilakunya bukan misteri: dengan α di antara 0 dan 1 ia stabil dan halus, dan dengan α di atas 2 ia akan berosilasi dan meledak, yaitu Persamaan (11.9) dengan $\lambda = -1/\tau$. Bagaimana Profitina menghitung angka-angkanya sendiri adalah urusannya; bentuk matematikanya milik bab ini, dan angka-angka memberi saran sedangkan pemakai yang memutuskan.

11.11 Memakainya Sungguhan

- **Tuliskan neracanya lebih dulu.** Sebagian besar PDB praktis adalah neraca: laju perubahan suatu persediaan sama dengan yang masuk dikurangi yang keluar. Air dalam tandon, uang dalam rekening, barang dalam gudang, muatan dalam baterai, orang dalam antrean. Tuliskan neracanya dengan kata-kata, lalu dengan simbol, seperti pada Persamaan (11.1). Periksa satuan

setiap suku: bila ruas kiri bersatuan meter kubik per menit, setiap suku di ruas kanan juga harus demikian, dan suku yang tidak demikian adalah kesalahan yang sudah Anda temukan sebelum menjalankan apa pun.

- **Ukur konstantanya dengan cara praktis.** Anto tidak menghitung k dari geometri penetes; ia mengukur seberapa cepat tandon penuh mengosong. Satu atau dua pengukuran seperti itu menetapkan konstanta model sederhana lebih baik daripada buku pegangan.
- **Carilah kasus khusus yang eksak.** Sebelum memercayai simulasi, carilah situasi yang dapat Anda selesaikan dengan tangan (pompa mati, tanpa kebocoran, laju tetap) dan periksa programnya terhadap situasi itu, seperti Persamaan (11.3) memeriksa program kita.
- **Gunakan RK4 dengan langkah tetap untuk masalah yang sederhana dan mulus, dan `solve_ivp` untuk selebihnya.** Selalu periksa dengan membagi dua langkah atau memperketat toleransi.
- **Bagi langkah pada kejadian.** Pompa yang menyala atau mati, katup yang terbuka, harga yang berubah: buat langkahnya berakhir tepat di sana.
- **Sesuaikan tandon dengan keadaan Anda.** Tandon di atap rumah ketika air PDAM mati adalah model yang sama dengan pompa mati dan k yang berbeda untuk keran-keran rumah; sawah yang dikeringkan melalui pintu air adalah model yang sama dengan konstanta pintu; baterai ponsel yang terkuras dengan beban tetap adalah kasus paling sederhana, sebuah garis lurus. Ubah konstanta dalam `morning.py`, periksa terhadap satu pengamatan nyata, lalu buat rencana.
- **Utamakan keselamatan.** Simulasi kendaraan, bangunan, atau dosis obat adalah alat belajar, bukan sertifikat. Suspensi yang terasa limbung, atau model yang mengatakan demikian, adalah alasan untuk menemui montir.

11.12 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memercayai Euler karena langkahnya kecil.** Orde satu berarti akurasi itu mahal, dan untuk osilasi Euler dapat tidak stabil pada langkah yang tampak kecil (Subbab 11.8.2). Gunakan RK4 atau pemecah adaptif.
- **Melintasi sakelar.** Langkah yang memuat perubahan mendadak pada f kehilangan orde metodenya. Bagi langkah pada waktu pensakelaran, atau hentikan lalu mulai ulang pemecahnya di sana.
- **Lupa bahwa langkah dan solusi memakai huruf yang sama.** Di buku teks h adalah langkah; pada tandon kita h juga ketinggian. Dalam kode, beri nama yang berbeda.
- **Membiarkan akar kuadrat melihat bilangan negatif.** Pembulatan dapat mendorong ketinggian sedikit di bawah nol. Batasi, seperti yang dilakukan `max(h, 0.0)`, atau hentikan simulasi ketika tandon kosong.
- **Hanya memeriksa angka terakhir.** Sebuah simulasi dapat berakhir pada angka yang masuk akal setelah melakukan sesuatu yang mustahil di tengah jalan (tandon yang meluap tetapi tidak pernah menumpahkan air, jok yang memantul makin tinggi). Cetak atau gambar seluruh lintasannya.
- **Tidak memeriksa dengan separuh langkah.** Inilah uji termurah yang ada. Bila membagi dua langkah mengubah jawaban yang Anda pedulikan, langkahnya terlalu besar.
- **Memakai `odeint` dalam kode baru.** Fungsi itu masih bekerja, tetapi `solve_ivp` adalah antarmuka yang berlaku sekarang, dengan pilihan metode eksplisit dan implisit, deteksi kejadian, dan keluaran rapat.

11.13 Ringkasan Bab

- Persamaan diferensial biasa dengan nilai awal, masalah nilai awal (Persamaan (11.2)), memberikan laju perubahan suatu besaran sebagai fungsi waktu dan besaran itu sendiri. Tandon Anto mematuhi neraca massa Persamaan (11.1), dan dengan pompa mati ia memiliki solusi eksak Persamaan (11.3).
- Metode Euler (Persamaan (11.4)) mengikuti garis singgung selama satu langkah; galat lokalnya $O(h^2)$ dan galat globalnya $O(h)$ (Persamaan (11.5)).
- Metode Runge–Kutta mencicipi kemiringan beberapa kali di dalam setiap langkah. Metode dua tahap yang memenuhi syarat-syarat yang diturunkan dari Persamaan (11.6), seperti Heun dan titik tengah, berorde dua; RK4 klasik (Persamaan (11.7)) berorde empat dan tereduksi menjadi aturan Simpson bila f hanya bergantung pada t. Rasio galat yang terukur, 2, 4, dan 16, memastikan orde-orde itu.
- Simulator pagi hari membagi langkahnya pada sakelar pompa dan katup, menangani luapan dan saat kering, dan diperiksa terhadap waktu kering yang eksak (10.30) serta dengan membagi dua langkah. Biseksi yang membungkus simulasi menemukan jam mati pompa, 06.42, yang mengisi tandon sampai 98 cm tanpa luapan.
- Persamaan orde dua menjadi sistem orde satu (Persamaan (11.8)). Pada suspensi dengan redaman ringan, metode Euler dengan $h = 0.01$ s tidak stabil, karena $h\lambda$ berada di luar daerah kestabilannya (Persamaan (11.9)); RK4 akurat.
- Membagi dua langkah menaksir galat metode apa pun berorde p (Persamaan (11.10)); `solve_ivp` melakukannya secara otomatis dengan pasangan Runge–Kutta tertanam dan menyediakan metode implisit untuk masalah kaku.

11.14 Soal Latihan

1. Dengan tangan, ambil dua langkah metode Heun dengan langkah 10 menit untuk tandon yang mengosong dari $h = 1$, lalu bandingkan dengan ketinggian eksak pada menit ke-20 dan dengan dua langkah Euler dari Subbab 11.4.
2. Segelas teh panas bersuhu 90 °C mendingin dalam ruangan bersuhu 28 °C mengikuti hukum pendinginan Newton, $T' = -0.1\,(T - 28)$ dengan t dalam menit. Hitung tiga langkah Euler dengan ukuran langkah 5, 15, dan 25 menit. Ukuran langkah mana yang memberi jawaban masuk akal secara fisika, mana yang berosilasi, dan mana yang meledak? Jelaskan dengan Persamaan (11.9).
3. Tunjukkan bahwa bila f hanya bergantung pada t, satu langkah RK4 Persamaan (11.7) adalah aturan Simpson pada $[t_n, t_n + h]$. Mengapa ini berarti RK4 eksak untuk $y' = 3t^2$ dengan langkah berapa pun?
4. Alih-alih mematikan pompa lebih awal, Anto dapat menutup sebagian katup di keluaran pompa sehingga pompa mengalirkan lebih sedikit air dan menyala dari 06.00 sampai 07.30. Debit berapa, dalam liter per menit, yang mengisi tandon tepat sampai 98 cm? Kapan penetes berhenti? (Ubah `PUMP` dalam `morning.py` dan gunakan biseksi padanya.)
5. Gunakan Persamaan (11.10) dengan hasil RK4 untuk tandon yang mengosong pada satu jam dengan langkah 10 dan 5 menit (dari keluaran Subbab 11.5.3 atau dengan menjalankan programnya) untuk menaksir galat hasil 5 menit, lalu bandingkan dengan galat sebenarnya.
6. Metode Euler *semi-implisit* (atau simplektik) untuk jok memperbarui kecepatan lebih dulu, lalu bergerak dengan kecepatan yang *baru*: $v_{n+1} = v_n + h\,a(x_n, v_n)$, $x_{n+1} = x_n + h\,v_{n+1}$. Tuliskan dalam Python dan Java untuk peredam aus dengan $h = 0.01$ s dan bandingkan dengan rujukan RK4. Apakah metode ini stabil?

11.15 Penyelesaian Soal Latihan

1. Langkah pertama: $k_1 = -0.01$, $k_2 = -0.01\sqrt{1 - 0.1} = -0.0094868$, $h_1 = 1 + 5 \times (-0.0194868) = 0.9025658$. Langkah kedua: $k_1 = -0.01\sqrt{0.9025658} = -0.0095004$; nilai Euler percobaannya $0.9025658 - 0.0950040 = 0.8075618$; $k_2 = -0.01\sqrt{0.8075618} = -0.0089864$; $h_2 = 0.9025658 + 5 \times (-0.0184868) = 0.8101319$. Ketinggian eksaknya 0.81, jadi Heun kelebihan 0,13 mm, sedangkan dua langkah Euler memberi 0.805132, kurang 4,9 mm: dengan dua kali banyaknya evaluasi kemiringan, sekitar 37 kali lebih akurat.

2. Tulis $u = T - 28$, sehingga $u' = -0.1\,u$ dengan $u_0 = 62$, dan Euler memberi $u_{n+1} = (1 - 0.1\,h)\,u_n$. Dengan $h = 5$: faktornya 0.5, dan T berjalan 90,59,43.5,35.75, kurva pendinginan yang masuk akal (nilai eksaknya 65.6,50.8,41.8). Dengan $h = 15$: faktornya -0.5, dan T berjalan $90, -3, 43.5, 20.25$: angka-angkanya masih mendekati 28 tetapi melompat ke kedua sisinya, termasuk segelas teh bersuhu -3 °C, yang secara fisika mustahil. Dengan $h = 25$: faktornya -1.5, dan T berjalan $90, -65, 167.5, -181.25$: tidak stabil. Menurut Persamaan (11.9) dengan $\lambda = -0.1$, Euler stabil untuk $h \le 2/0.1 = 20$ menit dan bebas osilasi hanya untuk $h \le 10$ menit, ketika faktornya tetap tidak negatif.

3. Bila $f = f(t)$, maka $k_1 = f(t_n)$, $k_2 = k_3 = f\left(t_n + \frac{h}{2}\right)$ karena argumen keduanya diabaikan, dan $k_4 = f(t_n + h)$. Jadi $y_{n+1} - y_n = \frac{h}{6}\left(f(t_n) + 4f\left(t_n + \frac{h}{2}\right) + f(t_n + h)\right)$, yaitu aturan Simpson pada satu panel selebar h dengan titik tengah $t_n + \frac{h}{2}$. Perubahan eksaknya $\int_{t_n}^{t_n+h} f\,(t)\,dt$. Aturan Simpson eksak untuk polinomial sampai derajat tiga (Bab 10), dan $3t^2$ berderajat dua, sehingga setiap langkah RK4 eksak dan demikian pula seluruh solusinya, $y = t^3$, berapa pun langkahnya. Pengujian dalam `test_ode.py` memeriksa hal ini dengan langkah 0,5.

4. Dengan `PUMP` sebagai besaran yang dicari, biseksi antara 0 dan 20 L/menit (0 dan 0,02 m^3/menit) konvergen ke sekitar 14,2 L/menit. Tandon lalu naik lebih lambat, berada di 98 cm ketika pompa mati pukul 07.30, tidak pernah meluap, dan penetes berhenti sekitar pukul 10.28, tiga perempat jam lebih lambat daripada pemompaan debit penuh yang terencana di Subbab 11.7, dan tanpa air yang terbuang. Air yang sama, bila dialirkan lebih pelan, bertahan lebih lama.

5. $y_{10} = 0.4900001535$ dan $y_5 = 0.4900000093$. Menurut Persamaan (11.10) dengan $p = 4$, $y(60) - y_5 \approx (0.4900000093 - 0.4900001535)/15 = -9.6 \times 10^{-9}$, yang berarti y_5 kelebihan sekitar 9.6×10^{-9}. Galat sebenarnya $y_5 - 0.49 = 9.3 \times 10^{-9}$. Taksirannya sangat baik, karena pada langkah-langkah ini RK4 sudah berada di rezim asimtotiknya (rasio di Subbab 11.5.3 dekat dengan 16).

6. **Kode 11.12** dan **Kode 11.13** mengimplementasikan metode ini sebagai fungsi langkah yang dapat dipasang pada `trace` dari `bump.py`.

Kode 11.12 `semi_implicit.py`: Euler semi-implisit untuk peredam aus.

```python
"""Exercise 11.6: semi-implicit (symplectic) Euler for the worn shock: update
the speed first, then move with the new speed."""
from bump import MASS, SPRING, make_rhs, rk4_step, trace


def semi_implicit_step(damping):
    def step(f, t, y, h):
        x, v = y
        v = v + h * (-(damping * v + SPRING * x) / MASS)
        return [x + h * v, v]            # uses the speed just computed
    return step


def main():
    c = 150.0
    print("seat position (mm) at t =  "
          "         0.5 s     1.0 s     1.5 s     2.0 s")
    runs = (("semi-implicit Euler h=0.01", semi_implicit_step(c), 0.01),
            ("RK4 h=0.001 (reference)   ", rk4_step, 0.001))
    for label, fn, h in runs:
        cells = "".join(f"{x:10.3f}" for x in trace(fn, make_rhs(c), h))
        print(f"{label}{cells}")


if __name__ == "__main__":
    main()
```

Kode 11.13 `SemiImplicit.java`: metode yang sama dalam Java.

```java
import java.util.Locale;

/** Exercise 11.6: semi-implicit (symplectic) Euler for the worn shock: update
 *  the speed first, then move with the new speed. */
public class SemiImplicit {
    static Bump.Stepper semiImplicitStep(double damping) {
        return (f, t, y, h) -> {
            double x = y[0], v = y[1];
            v = v + h * (-(damping * v + Bump.SPRING * x) / Bump.MASS);
            return new double[] {x + h * v, v};   // uses the speed just computed
        };
    }

    public static void main(String[] args) {
        double c = 150.0;
        System.out.println("seat position (mm) at t =  "
                           + "         0.5 s     1.0 s     1.5 s     2.0 s");
        String[] labels = {"semi-implicit Euler h=0.01",
                           "RK4 h=0.001 (reference)   "};
        Bump.Stepper[] steppers = {semiImplicitStep(c), Bump::rk4Step};
        double[] steps = {0.01, 0.001};
        for (int m = 0; m < labels.length; m++) {
            StringBuilder cells = new StringBuilder();
            for (double x : Bump.trace(steppers[m], Bump.makeRhs(c), steps[m])) {
                cells.append(String.format(Locale.ROOT, "%10.3f", x));
            }
            System.out.println(labels[m] + cells);
        }
    }
}
```

Keduanya mencetak:

```
seat position (mm) at t =           0.5 s     1.0 s     1.5 s     2.0 s
semi-implicit Euler h=0.01    23.171    -1.598   -12.393   -10.955
RK4 h=0.001 (reference)       25.761     1.607   -10.595   -11.134
```

Metode ini stabil: ayunannya tetap berukuran benar dan meredam perlahan, seperti kata fisikanya, walaupun posisinya meleset beberapa milimeter karena fase osilasinya bergeser. Memakai kecepatan baru untuk bergerak mengaitkan kedua pembaruan sedemikian rupa sehingga metode ini tidak lagi menambahkan energi pada osilasi; batas kestabilannya untuk osilator tanpa redaman adalah $h\omega \leq 2$, jauh melampaui $h\omega = 0.14$ kita. Metode dengan sifat ini, yang disebut simplektik, adalah pilihan baku dalam mesin fisika dan simulasi molekul, tempat osilasi harus berjalan jutaan langkah tanpa membesar.

Daftar Pustaka

[1] L. Euler, *Institutionum Calculi Integralis*, vol. 1. St. Petersburg, Russia: Imperial Academy of Sciences, 1768.

[2] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[3] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[4] E. Hairer, S. P. Nørsett, and G. Wanner, *Solving Ordinary Differential Equations I: Nonstiff Problems*, 2nd rev. ed. Berlin, Germany: Springer, 1993.

[5] W. Kutta, "Beitrag zur näherungsweisen Integration totaler Differentialgleichungen," *Zeitschrift für Mathematik und Physik*, vol. 46, pp. 435–453, 1901.

[6] J. C. Butcher, *Numerical Methods for Ordinary Differential Equations*, 3rd ed. Chichester, UK: Wiley, 2016.

[7] E. Hairer and G. Wanner, *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*, 2nd rev. ed. Berlin, Germany: Springer, 1996.

[8] SciPy Developers, "scipy 1.18.1," Python Package Index, Aug. 21, 2026. [Online]. Available: https://pypi.org/project/scipy/

[9] J. R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *Journal of Computational and Applied Mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[10] SciPy Developers, "scipy.integrate.solve_ivp," SciPy v1.18 Manual, accessed Sep. 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

Bab 12

Nilai Eigen: Metode Pangkat dan Kegunaannya

Sebagian pertanyaan tentang sebuah sistem tidak menyangkut saat tertentu, melainkan apa yang dilakukan sistem itu *dalam jangka panjang* atau *menurut sifatnya*: berapa pangsa pelanggan yang akhirnya dimiliki setiap toko, bagaimanapun keadaan bulan pertama; pada frekuensi berapa sebuah gedung senang bergoyang; halaman web mana yang paling sering disinggahi pembaca yang berselancar secara acak. Masing-masing dijawab oleh sebuah nilai eigen dan vektor eigennya, dan ketiganya dapat dihitung dengan gagasan sederhana yang sama: kalikan dengan matriksnya lagi dan lagi. Di akhir bab ini Anda akan mampu menjelaskan apa itu nilai eigen dan vektor eigen serta menghitungnya dengan tangan untuk matriks kecil; memodelkan pelanggan yang berpindah-pindah antartoko sebagai rantai Markov dan menemukan pangsa jangka panjangnya; membuktikan mengapa metode pangkat konvergen dan seberapa cepat, serta mempercepatnya dengan hasil bagi Rayleigh; menemukan nilai eigen terkecil dan nilai eigen mana pun yang dipilih dengan iterasi invers dan iterasi invers bergeser, memakai ulang dekomposisi LU dari Bab 6; menghitung frekuensi alami dan bentuk ragam sebuah ruko tiga lantai; menghitung PageRank untuk jaringan kecil situs web, termasuk halaman tanpa tautan; menyingkirkan nilai eigen yang sudah ditemukan dengan deflasi; membatasi nilai eigen dengan teorema Gershgorin; dan memeriksa setiap hasil terhadap NumPy.

12.1 Tiga Binatu dalam Satu Kampung

Di sebuah kampung di Malang, tiga binatu bersaing memperebutkan beberapa ratus rumah tangga yang sama: Laundry Bersih di dekat masjid, Laundry Kilat di jalan utama, dan Laundry Wangi, yang terbaru, di dekat sekolah. Bu Endang, pemilik Bersih, memperhatikan bahwa pelanggan tidak setia selamanya. Sebagian mencoba Kilat karena searah dengan jalan ke tempat kerja; sebagian pindah ke Wangi karena tetangga memuji keharumannya; sebagian kembali. Ia bertanya-tanya apa akibat datang-perginya pelanggan ini bagi usahanya. Bila keadaan ini berlanjut, apakah Bersih akan mempertahankan pangsanya, tumbuh, atau perlahan kehilangan pelanggan kepada pendatang baru itu?

Seorang mahasiswa dari kampung itu yang membantu di Bersih pada akhir pekan membuat survei kecil untuknya: dua ratus rumah tangga ditanya binatu mana yang mereka pakai bulan lalu dan bulan ini. Jawabannya memberi, untuk setiap binatu, berapa bagian pelanggannya yang bertahan dan berapa bagian yang pindah ke setiap pesaing dalam sebulan. Untuk Bersih, 80% bertahan, 10% pindah ke Kilat dan 10% ke Wangi. Untuk Kilat, 75% bertahan, 15% datang ke Bersih dan 10% pindah ke Wangi. Untuk Wangi, yang terbaru, hanya 60% bertahan, 20% pindah ke Bersih dan 20% ke Kilat.

Dengan angka-angka ini kita dapat memutar bulan-bulan ke depan. Bulan ini Bersih memiliki 40% pelanggan, Kilat 35%, dan Wangi 25%. Bulan depan Bersih mempertahankan 80% dari 40% miliknya (32 poin), mendapat 15% dari 35% milik Kilat (5,25 poin), dan 20% dari 25% milik Wangi (5 poin): 42,25%. Kilat dan Wangi dihitung dengan cara yang sama. Ulangi untuk bulan berikutnya, dan bulan sesudahnya. Apakah pangsa-pangsa itu akhirnya tenang? Ke nilai berapa?

Dan seberapa cepat? Jawabannya adalah sebuah vektor eigen, sebuah nilai eigen, dan rasio dua nilai eigen, dan prosedur yang baru saja kita gambarkan, mengalikan dengan tabel yang sama lagi dan lagi, adalah **metode pangkat** (*power method*), alat utama bab ini.

Matematika yang sama menentukan, dalam latar yang sangat berbeda, bagaimana sebuah gedung bergoyang ketika gempa dan bagaimana mesin pencari web yang pertama memeringkat halaman. Kita akan menjumpai keduanya. Nilai eigen kadang diajarkan sebagai topik aljabar semata, sebuah determinan yang harus diekspansikan; di sini kita menjumpainya sebagai apa ia sesungguhnya dalam penerapan: bilangan-bilangan yang menggambarkan apa yang dilakukan sebuah sistem bila dibiarkan sendiri.

12.2 Gagasannya dalam Kata-Kata Sederhana

12.2.1 Arah yang hanya diregangkan matriks

Matriks persegi A mengubah vektor x menjadi vektor baru Ax. Biasanya vektor baru itu menunjuk ke arah yang berbeda. Tetapi bagi sebagian besar matriks ada beberapa arah istimewa yang sama sekali tidak dibelokkan matriks itu, hanya diregangkan atau dimampatkan. Vektor tak nol v yang memenuhi

$$Av = \lambda v \tag{12.1}$$

adalah **vektor eigen** A, dan bilangan λ adalah **nilai eigen**-nya: faktor peregangan arah itu. (Kata Jerman *eigen* berarti "milik sendiri": arah-arah dan nilai-nilai milik matriks itu sendiri.) Matriks $n \times n$ memiliki n nilai eigen, dihitung dengan multiplisitasnya, mungkin kompleks. Nilai-nilai itu adalah akar **polinomial karakteristik** $\det(A - \lambda I) = 0$, karena Persamaan (12.1) menyatakan bahwa $(A - \lambda I)v = 0$ memiliki solusi tak nol, yang terjadi tepat ketika $A - \lambda I$ singular [1], [2], [3].

Sebuah contoh kecil membuatnya konkret. Ambil

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

Maka $\det(A - \lambda I) = (2 - \lambda)^2 - 1 = 0$ memberi $\lambda = 3$ dan $\lambda = 1$. Untuk $\lambda = 3$, $(A - 3I)v = 0$ berarti $-v_1 + v_2 = 0$, jadi $v = (1,1)$; memang $A(1,1) = (3,3) = 3(1,1)$. Untuk $\lambda = 1$, $v = (1,-1)$, dan $A(1,-1) = (1,-1)$. Matriks itu meregangkan arah diagonal tiga kali dan membiarkan arah antidiagonal apa adanya.

12.2.2 Mengapa perkalian berulang menemukan yang terbesar

Sekarang kalikan sebuah vektor yang *bukan* vektor eigen dengan A beberapa kali, misalnya $x = (1,0)$. Kita mendapat $(2,1)$, lalu $(5,4)$, lalu $(14,13)$, lalu $(41,40)$. Vektor-vektor itu membesar, dan arahnya berayun menuju $(1,1)$: rasio kedua komponennya 2, 1,25, 1,077, 1,025. Alasannya sederhana. Tulis $(1,0) = \frac{1}{2}(1,1) + \frac{1}{2}(1,-1)$, separuh dari masing-masing vektor eigen. Setiap perkalian mengalikan separuh pertama dengan 3 dan separuh kedua dengan 1. Setelah k perkalian, bagian pertama 3^k kali lebih besar sedangkan bagian kedua tidak berubah, sehingga bagian pertama makin mendominasi. Rasio panjang berturutan mendekati 3, nilai eigen terbesar: $41/14 = 2.93$, sudah dekat.

Itulah metode pangkat dalam satu paragraf: *terus kalikan, dan arah dengan peregangan terbesar akan menang.* Seberapa cepat ia menang bergantung pada seberapa jauh peregangan terbesar melampaui peregangan terbesar kedua: di sini 3 dibanding 1, sehingga bagian yang tidak diinginkan mengecil tiga kali per langkah.

> **WAWASAN KUNCI:** Perkalian berulang dengan sebuah matriks adalah hal yang dilakukan banyak proses nyata dengan sendirinya: pelanggan yang berpindah bulan demi bulan, pembaca acak yang mengeklik tautan demi tautan, getaran yang merambat dari lantai ke lantai. Proses seperti itu melupakan titik awalnya dan menetap pada vektor eigen dominan matriksnya. Metode pangkat sekadar membiarkan proses itu berjalan di komputer.

12.3 Pasar Binatu sebagai Rantai Markov

12.3.1 Matriks perpindahan

Misalkan $x_k = (x_{k,1}, x_{k,2}, x_{k,3})$ adalah pangsa Bersih, Kilat, dan Wangi pada bulan k, sebagai pecahan yang jumlahnya 1. Survei itu memberi **matriks perpindahan** P, yang entri P_{ij}-nya adalah bagian pelanggan binatu j yang memakai binatu i bulan berikutnya:

$$P = \begin{pmatrix} 0.80 & 0.15 & 0.20 \\ 0.10 & 0.75 & 0.20 \\ 0.10 & 0.10 & 0.60 \end{pmatrix}, \qquad x_{k+1} = P x_k. \tag{12.2}$$

Kolom j menggambarkan ke mana pelanggan binatu j pergi, sehingga jumlah setiap kolom 1: setiap pelanggan pergi ke suatu tempat. Proses yang keadaan berikutnya hanya bergantung pada keadaan sekarang melalui matriks seperti itu adalah **rantai Markov**, dan P adalah matriks transisinya yang stokastik-kolom [1]. Menurut Persamaan (12.2), $x_k = P^k x_0$: pangsa setelah k bulan adalah pangsa awal yang dikalikan k kali dengan P, persis metode pangkat.

12.3.2 Apa yang dikatakan nilai eigennya

Karena setiap kolom P berjumlah 1, vektor baris (1,1,1) memenuhi $(1,1,1)P = (1,1,1)$. Jadi 1 adalah nilai eigen P^{T}, dan karena sebuah matriks dan transposnya memiliki nilai eigen yang sama, 1 adalah nilai eigen P. Vektor eigennya s dengan $Ps = s$, diskalakan agar jumlahnya 1, adalah sekumpulan pangsa yang *tidak berubah* dari bulan ke bulan: **distribusi stasioner**. Untuk matriks yang semua entrinya positif, seperti milik kita, teorema Perron–Frobenius menjamin bahwa vektor eigen ini tunggal dan semua komponennya positif, dan bahwa setiap nilai eigen lain memenuhi $|\lambda| < 1$ [2]. Dengan menulis pangsa awal sebagai $x_0 = s + c_2 v_2 + c_3 v_3$ dalam basis vektor eigen,

$$x_k = P^k x_0 = s + c_2 \lambda_2^k v_2 + c_3 \lambda_3^k v_3 \to s, \tag{12.3}$$

dan jarak ke pangsa stasioner mengecil seperti $|\,\lambda_2\,|^k$, dengan λ_2 nilai eigen terbesar kedua dalam nilai mutlak. (Di sini koefisien s tepat 1, karena x_0, s, dan setiap $P^k x_0$ berjumlah 1, sedangkan vektor-vektor eigen lain dari matriks stokastik berjumlah 0.) Argumen yang sama untuk matriks umum, dengan λ_1 menggantikan 1, adalah bukti kekonvergenan metode pangkat: komponen yang tidak diinginkan mengecil dengan faktor $_2 / _1$ per langkah.

12.3.3 Programnya

Kode 12.1 memutar lima belas bulan ke depan dari 40%, 35%, dan 25%, mencetak pangsa dan total perubahannya dari bulan sebelumnya, dan menaksir faktor kekonvergenan dari dua perubahan terakhir.

Kode 12.1 `market.py`: pangsa binatu bulan demi bulan, metode pangkat pada matriks perpindahan.

```
"""Where do the laundry customers of a kampung end up? Month after month the
shares are multiplied by the switching matrix: the power method at work."""

NAMES = ["Bersih", "Kilat", "Wangi"]
# P[i][j]: share of laundry j's customers who use laundry i next month.
P = [[0.80, 0.15, 0.20],
     [0.10, 0.75, 0.20],
     [0.10, 0.10, 0.60]]


def mat_vec(a, x):
    return [sum(a[i][j] * x[j] for j in range(len(x))) for i in range(len(a))]


def main():
    x = [0.40, 0.35, 0.25]                  # shares this month
    print("month   " + "   ".join(f"{n:>7}" for n in NAMES) + "     change")
    print(f"{0:5d}   " + "   ".join(f"{v:7.4f}" for v in x))
    changes = []
    for month in range(1, 16):
        new = mat_vec(P, x)
        change = sum(abs(new[i] - x[i]) for i in range(3))
        changes.append(change)
        x = new
        print(f"{month:5d}   " + "   ".join(f"{v:7.4f}" for v in x)
              + f"   {change:9.2e}")
    ratio = changes[-1] / changes[-2]
    print(f"change shrinks by a factor {ratio:.4f} per month")
    print("long-run shares: " + ", ".join(
        f"{n} {100 * v:.1f}%" for n, v in zip(NAMES, x)))


if __name__ == "__main__":
    main()
```

Kode 12.2 `Market.java`: program yang sama dalam Java.

```java
import java.util.Locale;

/** Where do the laundry customers of a kampung end up? Month after month the
 *  shares are multiplied by the switching matrix: the power method at work. */
public class Market {
    static final String[] NAMES = {"Bersih", "Kilat", "Wangi"};
    /** P[i][j]: share of laundry j's customers who use laundry i next month. */
    static final double[][] P = {{0.80, 0.15, 0.20},
                                 {0.10, 0.75, 0.20},
                                 {0.10, 0.10, 0.60}};

    static double[] matVec(double[][] a, double[] x) {
        double[] r = new double[a.length];
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < x.length; j++) {
                r[i] += a[i][j] * x[j];
            }
        }
        return r;
    }

    static String row(double[] x) {
        StringBuilder s = new StringBuilder();
        for (int i = 0; i < x.length; i++) {
            s.append(i > 0 ? "  " : "")
             .append(String.format(Locale.ROOT, "%7.4f", x[i]));
        }
        return s.toString();
    }

    public static void main(String[] args) {
        double[] x = {0.40, 0.35, 0.25};          // shares this month
        StringBuilder head = new StringBuilder("month  ");
        for (int i = 0; i < NAMES.length; i++) {
            head.append(i > 0 ? "  " : "").append(String.format("%7s", NAMES[i]));
        }
        System.out.println(head + "     change");
        System.out.println(String.format(Locale.ROOT, "%5d  ", 0) + row(x));
        double[] changes = new double[16];
        for (int month = 1; month <= 15; month++) {
            double[] next = matVec(P, x);
            double change = 0;
            for (int i = 0; i < 3; i++) {
                change += Math.abs(next[i] - x[i]);
            }
            changes[month] = change;
```

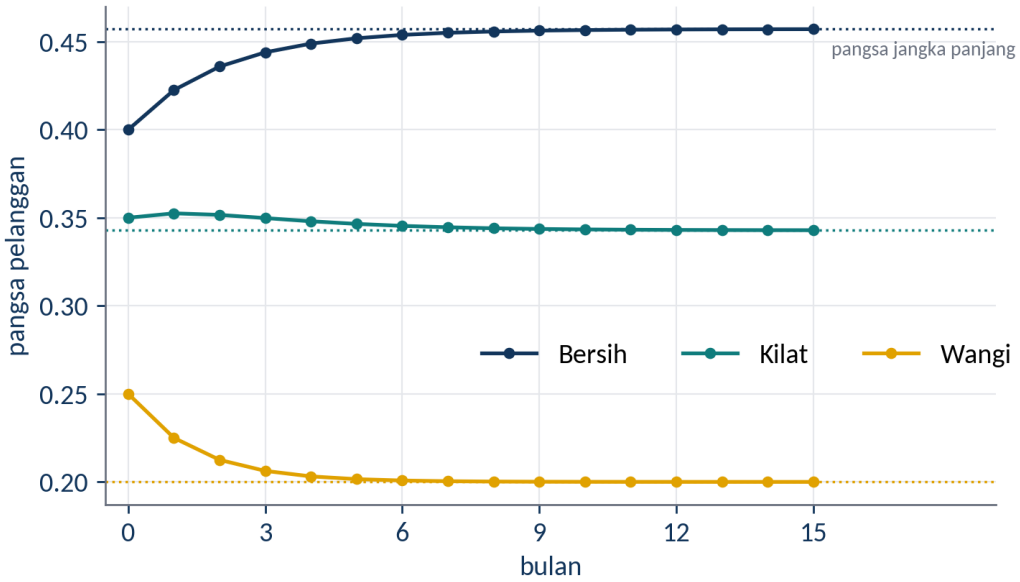
Kode 12.2 (Lanjutan) *Market.java: program yang sama dalam Java.*

```java
            x = next;
            System.out.println(String.format(Locale.ROOT, "%5d  ", month) + row(x)
                               + String.format(Locale.ROOT, "  %9.2e", change));
        }
        System.out.println(String.format(Locale.ROOT,
            "change shrinks by a factor %.4f per month",
            changes[15] / changes[14]));
        StringBuilder s = new StringBuilder("long-run shares: ");
        for (int i = 0; i < 3; i++) {
            s.append(i > 0 ? ", " : "").append(NAMES[i])
             .append(String.format(Locale.ROOT, " %.1f%%", 100 * x[i]));
        }
        System.out.println(s);
    }
}
```

Kode 12.2 mencetak tabel yang sama:

```
month   Bersih    Kilat    Wangi      change
    0   0.4000   0.3500   0.2500
    1   0.4225   0.3525   0.2250   5.00e-02
    2   0.4359   0.3516   0.2125   2.68e-02
    3   0.4439   0.3498   0.2063   1.61e-02
    4   0.4489   0.3480   0.2031   9.86e-03
    5   0.4519   0.3465   0.2016   6.10e-03
    6   0.4538   0.3454   0.2008   3.81e-03
    7   0.4550   0.3446   0.2004   2.40e-03
    8   0.4558   0.3440   0.2002   1.52e-03
    9   0.4563   0.3436   0.2001   9.68e-04
   10   0.4566   0.3434   0.2000   6.19e-04
   11   0.4568   0.3432   0.2000   3.98e-04
   12   0.4569   0.3431   0.2000   2.56e-04
   13   0.4570   0.3430   0.2000   1.65e-04
   14   0.4570   0.3430   0.2000   1.07e-04
   15   0.4571   0.3429   0.2000   6.91e-05
change shrinks by a factor 0.6471 per month
long-run shares: Bersih 45.7%, Kilat 34.3%, Wangi 20.0%
```

Baris pertama setelah bulan 0 adalah hitungan tangan di Subbab 12.1: Bersih 42,25%. Pangsa-pangsa itu menetap menuju Bersih 45,7%, Kilat 34,3%, dan Wangi 20,0%, apa pun titik awalnya, dan setiap bulan perubahan yang tersisa sekitar 0,65 kali perubahan sebelumnya. **Gambar 12.1** memperlihatkan ketiga kurva mendekati tingkat jangka panjangnya.



Gambar 12.1 Pangsa ketiga binatu bulan demi bulan dari 40%, 35%, dan 25%; garis titik-titik adalah pangsa stasioner, vektor eigen matriks perpindahan untuk nilai eigen 1.

Kedua angka itu dapat diperiksa secara eksak. Pangsa stasioner menyelesaikan $(P - I)s = 0$ dengan $s_1 + s_2 + s_3 = 1$. Baris ketiga memberi $0.1s_1 + 0.1s_2 - 0.4s_3 = 0$, dan baris pertama memberi $-0.2s_1 + 0.15s_2 + 0.2s_3 = 0$; bersama syarat jumlah, solusinya $s = (16/35, 12/35, 7/35) = (0.4571, 0.3429, 0.2000)$, tepat nilai yang dicapai program. Untuk nilai eigennya, jumlah nilai eigen sama dengan teras (jumlah diagonal), $0.80 + 0.75 + 0.60 = 2.15$, dan hasil kalinya sama dengan determinan, 0.325. Dengan $\lambda_1 = 1$, dua yang lain memenuhi $\lambda_2 + \lambda_3 = 1.15$ dan $\lambda_2\lambda_3 = 0.325$, sehingga keduanya akar $\lambda^2 - 1.15\lambda + 0.325 = 0$: $\lambda_2 = 0.65$ dan $\lambda_3 = 0.5$. Faktor terukur 0,6471 adalah λ_2 yang terlihat melalui sisa-sisa terakhir λ_3.

Bagi Bu Endang pesannya menggembirakan tetapi tidak dramatis: bila pelanggan terus berpindah seperti bulan ini, pangsa Bersih akan tumbuh dari 40% menjadi sekitar 46%, sebagian besar

dengan mengorbankan Wangi, dan dalam setengah tahun pasar akan berada dalam satu poin persentase dari angka itu. Angka yang perlu diawasi adalah kesetiaan Wangi yang rendah, 60%; bila Wangi memperbaikinya, vektor eigennya berubah, dan program menjawab pertanyaan baru itu dalam sedetik. Model ini memberi saran; angka-angkanya bergantung pada perilaku pelanggan tahun depan, yang tidak dapat dijamin survei mana pun.

12.3.4 Keadaan keempat: mencuci di rumah

Survei itu masih memuat satu jawaban yang diabaikan model tiga binatu: beberapa rumah tangga berhenti memakai binatu mana pun dan kembali mencuci di rumah, dan beberapa yang biasa mencuci di rumah mencoba binatu untuk pertama kalinya. Menambahkan "rumah" sebagai keadaan keempat menjaga model tetap jujur. Misalkan setiap binatu kehilangan 5% pelanggannya ke cucian rumah setiap bulan (diambil dari mereka yang semestinya bertahan), dan setiap bulan 6% dari yang mencuci di rumah mencoba Bersih, 4% Kilat, dan 5% Wangi, sedangkan 85% tetap mencuci di rumah. Jumlah setiap kolom masih 1, sehingga 1 masih merupakan nilai eigen, dan vektor stasionernya, yang dihitung persis seperti sebelumnya, adalah Bersih 34,6%, Kilat 25,6%, Wangi 14,8%, dan rumah 25,0%. Urutan di antara binatu tidak berubah, tetapi seperempat rumah tangga akhirnya berada di luar pasar sama sekali, dan itu informasi yang diperlukan Bu Endang: pesaingnya bukan hanya dua binatu lain, tetapi juga mesin cuci di rumah.

Nilai eigen matriks empat keadaan ini adalah 1, 0,8, 0,61, dan 0,46. Nilai eigen kedua naik dari 0,65 menjadi 0,8, karena keadaan rumah berubah lambat (85% bertahan), dan pasar kini memerlukan sekitar $\ln 0.5/\ln 0.8 \approx 3.1$ bulan untuk menutup separuh selisih yang tersisa, bukan 1,6. Sekelompok pelanggan yang berubah lambat memperlambat seluruh pasar, dan λ_2 mengukur persis seberapa besar.

12.4 Metode Pangkat secara Umum

12.4.1 Algoritmanya dan hasil bagi Rayleigh

Metode ini berasal dari Richard von Mises dan Hilda Pollaczek-Geiringer, yang menggambarkannya pada 1929 [4]. Untuk matriks umum, vektor $A^k x_0$ membesar atau mengecil seperti $|\lambda_1|^k$ dan sebentar saja akan meluap atau menyusut di bawah batas bilangan, sehingga metode pangkat menormalkan vektornya setelah setiap perkalian:

$$x_{k+1} = \frac{Ax_k}{\| Ax_k \|}.$$

Menurut Persamaan (12.3), x_k mendekati vektor eigen dominan (sampai tanda), dengan galat yang mengecil seperti $|\lambda_2/\lambda_1|^k$, asalkan $_1> _2$ dan vektor awalnya memiliki komponen sepanjang v_1 (awal yang acak hampir pasti memilikinya). Nilai eigennya ditaksir dari vektor sekarang dengan **hasil bagi Rayleigh** (*Rayleigh quotient*)

$$r(x) = \frac{x^{\mathsf{T}} A x}{x^{\mathsf{T}} x}, \tag{12.4}$$

yang sama dengan λ tepat ketika x vektor eigen. Untuk matriks **simetris**, yang vektor-vektor eigennya ortogonal, vektor dengan galat berukuran ϵ memberi hasil bagi Rayleigh dengan galat berukuran ϵ^2 [2]. Karena itu taksiran nilai eigennya konvergen seperti $|\lambda_2/\lambda_1|^{2k}$, dua kali

banyaknya digit benar per langkah dibandingkan vektornya. Program kita menormalkan vektor hingga panjangnya 1, sehingga penyebut Persamaan (12.4) bernilai 1 dan hasil baginya cukup $x^{\mathsf{T}}Ax$.

Metode pangkat hanya memerlukan perkalian matriks–vektor. Itulah kekuatan besarnya: untuk matriks dengan miliaran baris tetapi hanya sedikit entri tak nol per baris, seperti web di Subbab 12.6, satu perkalian memakan beberapa miliar operasi, sedangkan metode apa pun yang memfaktorkan matriksnya mustahil dijalankan. Kelemahannya pun sama jelasnya: ia hanya menemukan nilai eigen dominan, dan lambat bila $_2/_1$ dekat dengan 1. Subbab berikutnya mengatasi keduanya dengan satu gagasan.

12.4.2 Iterasi invers dan iterasi invers bergeser

Bila $Av = \lambda v$ dan σ sembarang bilangan yang bukan nilai eigen, maka $(A - \sigma I)v = (\lambda - \sigma)v$ dan karena itu

$$(A - \sigma I)^{-1}v = \frac{1}{\lambda - \sigma}\, v. \tag{12.5}$$

Matriks $(A - \sigma I)^{-1}$ memiliki vektor eigen yang sama dengan A, dan nilai eigennya $1/(\lambda_i - \sigma)$. Nilai eigen A yang *paling dekat* dengan σ menjadi nilai eigen *terbesar* dari $(A - \sigma I)^{-1}$, sehingga metode pangkat yang diterapkan pada $(A - \sigma I)^{-1}$ konvergen kepadanya, dengan laju $\{} - / {} - $. Dengan $\sigma = 0$ ini adalah **iterasi invers**, yang menemukan nilai eigen dengan nilai mutlak terkecil; dengan σ di dekat sebuah tebakan kasar ini adalah **iterasi invers bergeser**, yang menemukan nilai eigen terdekat dengan tebakan itu, dan konvergen sangat cepat bila tebakannya baik.

Inversnya tidak pernah dibentuk. Setiap langkah menyelesaikan sistem linear $(A - \sigma I)y = x_k$, dan karena matriksnya sama pada setiap langkah, matriks itu difaktorkan sekali dengan dekomposisi LU dari Bab 6 dan setiap langkah hanya memerlukan substitusi maju dan substitusi mundur. Bila geseran diperbarui pada setiap langkah dengan hasil bagi Rayleigh yang sekarang, metodenya disebut iterasi hasil bagi Rayleigh dan, untuk matriks simetris, konvergen secara kubik [2].

12.4.3 Iterasi invers dengan tangan

Matriks kecil dari Subbab 12.2 memperlihatkan iterasi invers bekerja. Nilai eigennya 3 dan 1, sehingga iterasi invers ($\sigma = 0$) semestinya menemukan yang lebih kecil, 1, dengan vektor eigen $(1, -1)$. Mulai lagi dari $x_0 = (1,0)$ dan selesaikan $Ay = x_0$: invers A adalah $\frac{1}{3}\begin{pmatrix} 2 & -1 \\ -1 & 2 \end{pmatrix}$, jadi $y = \frac{1}{3}(2, -1)$. Hanya arahnya yang penting, jadi ambil $x_1 = (2, -1)$ (normalisasi akan membaginya dengan $\sqrt{5}$). Hasil bagi Rayleigh-nya $x_1^{\mathsf{T}}Ax_1 / x_1^{\mathsf{T}}x_1 = (2 \cdot 4 + 2 \cdot 2 \cdot (-1) + 2 \cdot 1)/5 = 6/5 = 1.2$. Penyelesaian berikutnya memberi $y = \frac{1}{3}(5, -4)$, jadi $x_2 = (5, -4)$, dengan hasil bagi Rayleigh $(50 - 40 + 32)/41 = 42/41 = 1.024$. Lalu $x_3 = (14, -13)$ dan hasil bagi 1.0027. Komponen sepanjang $(1,1)$ dibagi 3 pada setiap langkah relatif terhadap komponen sepanjang $(1, -1)$, karena nilai eigen A^{-1} adalah $\frac{1}{3}$ dan 1; dan hasil bagi Rayleigh, seperti dijanjikan untuk matriks simetris, memperoleh kira-kira dua kali lebih banyak digit per langkah daripada vektornya. Bandingkan dengan Subbab 12.2.2: matriks yang sama, awal yang sama, kecepatan yang sama, tetapi kini konvergen ke vektor eigen yang *lain*.

12.4.4 Ketika metode pangkat gagal

Bukti kekonvergenan memerlukan $_1> _2$, dan bila syarat itu gagal, metodenya pun gagal. Contoh paling sederhana adalah matriks yang menukar dua komponen, $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$, dengan nilai eigen 1 dan -1 serta vektor eigen (1,1) dan $(1,-1)$. Mulai dari (1,0), metode pangkat menghasilkan (0,1), (1,0), (0,1), dan seterusnya tanpa akhir: kedua komponen sama besar, tidak ada yang menang, dan iterasinya berbalik-balik alih-alih menetap. Rantai Markov dapat berbuat sama: bila setiap pelanggan toko A pindah ke toko B setiap bulan dan setiap pelanggan B pindah ke A, pangsanya akan bertukar setiap bulan dan tidak pernah menetap. Rantai seperti itu disebut *periodik*, dan itulah keadaan yang dicegah oleh entri positif matriks binatu, atau oleh lompatan acak PageRank. Rotasi bidang adalah contoh lain: nilai eigennya sepasang bilangan kompleks dengan nilai mutlak yang sama, dan iterasi pangkat riil hanya berputar-putar. Bila metode biasa berkelana seperti ini, geseran membantu (nilai eigen $\lambda - \sigma$ tidak lagi sama besar), atau dipakai algoritma QR dari pustaka, yang menangani kasus seperti ini secara rutin.

12.5 Bagaimana Sebuah Ruko Bergoyang

12.5.1 Modelnya

Indonesia terletak di pertemuan beberapa lempeng tektonik, dan gempa bumi adalah bagian dari kehidupan. Ruko tiga lantai, rumah toko dengan toko di lantai dasar dan keluarga yang tinggal di atasnya, adalah jenis bangunan tempat banyak orang Indonesia tinggal dan bekerja. Ketika tanah berguncang, bangunan tidak bergoyang sembarangan; ia lebih menyukai pola-pola gerak tertentu, yaitu **ragam**-nya (*mode*), masing-masing dengan frekuensi alaminya sendiri. Bila tanah kebetulan berguncang di dekat salah satu frekuensi itu, gerakan bangunan membesar, dan itulah sebabnya para insinyur harus mengetahuinya. Standar perancangan gempa Indonesia, SNI 1726:2019, meminta perancang bangunan menentukan periode getar fundamental strukturnya, waktu satu goyangan pada ragam yang paling lambat [5].

Model pertama memperlakukan setiap pelat lantai sebagai massa m dan kolom-kolom setiap tingkat sebagai pegas berkekakuan k yang menghubungkannya dengan lantai di bawahnya. Bila u_1, u_2, u_3 adalah simpangan ke samping ketiga lantai, hukum kedua Newton untuk setiap lantai memberi $mu'' = -Ku$ dengan **matriks kekakuan**

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

Baris pertama, misalnya, menyatakan bahwa lantai 1 ditarik kembali oleh kolom di bawahnya $(-ku_1)$ dan oleh kolom di atasnya $(-k(u_1 - u_2))$. Atap hanya memiliki kolom di bawahnya, dan karena itu ada angka 1 di sudutnya. Solusi yang semua lantainya bergerak serempak dengan frekuensi yang sama, $u(t) = v\cos(\omega t)$, menuntut $-\omega^2 mv = -Kv$, yaitu

$$\frac{1}{m}Kv = \omega^2 v, \qquad f = \frac{\omega}{2\pi} = \frac{\sqrt{\lambda}}{2\pi}. \tag{12.6}$$

Bentuk ragamnya adalah vektor eigen $A = K/m$, dan nilai eigen $\lambda = \omega^2$ memberi frekuensi alami f dalam hertz (goyangan per detik) [6]. Dengan $k/m = 2000\ \text{s}^{-2}$, nilai yang realistis untuk rangka beton ringan, A adalah matriks dalam program. Matriks itu simetris, sehingga nilai eigennya riil

dan vektor eigennya ortogonal, dan hasil bagi Rayleigh menggandakan kecepatan metode pangkat.

12.5.2 Tiga metode, tiga ragam

Kode 12.3 menemukan ketiga ragam: yang tertinggi dengan metode pangkat biasa, yang terendah dengan iterasi invers, dan yang tengah dengan iterasi invers bergeser di sekitar $\sigma = 3000$. Faktorisasi LU kecil dengan pivot parsial disertakan agar program dapat berdiri sendiri; itulah metode Bab 6. Fungsi `iterate` dipakai bersama oleh ketiganya: ia menerima langkah yang diulang (kalikan dengan A, atau selesaikan dengan matriks yang sudah difaktorkan), menormalkan, menghitung hasil bagi Rayleigh dengan A yang *asli*, dan berhenti bila dua hasil bagi berturutan sama sampai sepuluh digit.

Kode 12.3 `power.py`: iterasi pangkat, invers, dan invers bergeser untuk ketiga ragam ruko.

```
"""Natural frequencies of a three-story shophouse (ruko) modeled as three
masses on springs: K v = omega^2 M v. Power method for the highest mode,
inverse and shifted inverse iteration for the lowest and the middle one."""
import math

KM = 2000.0                                  # story stiffness / story mass, 1/s^2
A = [[2 * KM, -KM, 0.0],
     [-KM, 2 * KM, -KM],
     [0.0, -KM, KM]]


def mat_vec(a, x):
    return [sum(a[i][j] * x[j] for j in range(len(x))) for i in range(len(a))]


def dot(x, y):
    return sum(xi * yi for xi, yi in zip(x, y))


def normalize(x):
    norm = math.sqrt(dot(x, x))
    return [xi / norm for xi in x]


def lu_factor(a):
    """LU with partial pivoting (Chapter 6); returns (LU in one matrix, pivots)."""
    n = len(a)
    lu = [row[:] for row in a]
    piv = list(range(n))
    for k in range(n):
        p = max(range(k, n), key=lambda i: abs(lu[i][k]))
        lu[k], lu[p] = lu[p], lu[k]
        piv[k], piv[p] = piv[p], piv[k]
        for i in range(k + 1, n):
            lu[i][k] /= lu[k][k]
            for j in range(k + 1, n):
                lu[i][j] -= lu[i][k] * lu[k][j]
    return lu, piv
```

Kode 12.3 (Lanjutan) *power.py: iterasi pangkat, invers, dan invers bergeser untuk ketiga ragam ruko.*

```
def lu_solve(lu, piv, b):
    n = len(lu)
    y = [b[piv[i]] for i in range(n)]
    for i in range(n):                          # forward: L y = P b
        y[i] -= sum(lu[i][j] * y[j] for j in range(i))
    for i in reversed(range(n)):                # backward: U x = y
        y[i] = (y[i] - sum(lu[i][j] * y[j] for j in range(i + 1, n))) / lu[i][i]
    return y


def iterate(step, a, x, tol=1e-10, max_iter=200):
    """Repeat x <- normalize(step(x)); the Rayleigh quotient estimates lambda."""
    lam_old = None
    for k in range(1, max_iter + 1):
        x = normalize(step(x))
        lam = dot(x, mat_vec(a, x))             # Rayleigh quotient, |x| = 1
        if lam_old is not None and abs(lam - lam_old) <= tol * abs(lam):
            return lam, x, k
        lam_old = lam
    return lam, x, max_iter


def report(name, lam, x, k):
    f = math.sqrt(lam) / (2 * math.pi)
    shape = "  ".join(f"{v / x[-1]:6.3f}" for v in x)   # top floor scaled to 1
    print(f"{name:7s} lambda {lam:10.3f}  f {f:5.2f} Hz  iterations {k:3d}"
          f"  shape {shape}")


def main():
    start = [1.0, 1.0, 1.0]
    report("highest", *iterate(lambda v: mat_vec(A, v), A, start))
    lu, piv = lu_factor(A)                      # factor once, solve many times
    report("lowest", *iterate(lambda v: lu_solve(lu, piv, v), A, start))
    shift = 3000.0
    shifted = [[A[i][j] - (shift if i == j else 0.0) for j in range(3)]
               for i in range(3)]
    lu_s, piv_s = lu_factor(shifted)
    report("middle", *iterate(lambda v: lu_solve(lu_s, piv_s, v), A, start))


if __name__ == "__main__":
    main()
```

Kode 12.4 `Power.java`: metode yang sama dalam Java; hasil sebuah iterasi berupa `record` kecil.

```java
import java.util.Locale;
import java.util.function.UnaryOperator;

/** Natural frequencies of a three-story shophouse (ruko) modeled as three
 *  masses on springs: K v = omega^2 M v. Power method for the highest mode,
 *  inverse and shifted inverse iteration for the lowest and the middle one. */
public class Power {
    static final double KM = 2000.0;       // story stiffness / story mass, 1/s^2
    static final double[][] A = {{2 * KM, -KM, 0.0},
                                 {-KM, 2 * KM, -KM},
                                 {0.0, -KM, KM}};

    static double[] matVec(double[][] a, double[] x) {
        double[] r = new double[a.length];
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < x.length; j++) {
                r[i] += a[i][j] * x[j];
            }
        }
        return r;
    }

    static double dot(double[] x, double[] y) {
        double s = 0;
        for (int i = 0; i < x.length; i++) {
            s += x[i] * y[i];
        }
        return s;
    }

    static double[] normalize(double[] x) {
        double norm = Math.sqrt(dot(x, x));
        double[] r = new double[x.length];
        for (int i = 0; i < x.length; i++) {
            r[i] = x[i] / norm;
        }
        return r;
    }
```

Kode 12.4 (Lanjutan) *Power.java: metode yang sama dalam Java; hasil sebuah iterasi berupa record kecil.*

```
    /** LU with partial pivoting (Chapter 6), in place; returns the pivots. */
    static int[] luFactor(double[][] lu) {
        int n = lu.length;
        int[] piv = new int[n];
        for (int i = 0; i < n; i++) {
            piv[i] = i;
        }
        for (int k = 0; k < n; k++) {
            int p = k;
            for (int i = k + 1; i < n; i++) {
                if (Math.abs(lu[i][k]) > Math.abs(lu[p][k])) {
                    p = i;
                }
            }
            double[] tmp = lu[k]; lu[k] = lu[p]; lu[p] = tmp;
            int t = piv[k]; piv[k] = piv[p]; piv[p] = t;
            for (int i = k + 1; i < n; i++) {
                lu[i][k] /= lu[k][k];
                for (int j = k + 1; j < n; j++) {
                    lu[i][j] -= lu[i][k] * lu[k][j];
                }
            }
        }
        return piv;
    }

    static double[] luSolve(double[][] lu, int[] piv, double[] b) {
        int n = lu.length;
        double[] y = new double[n];
        for (int i = 0; i < n; i++) {             // forward: L y = P b
            y[i] = b[piv[i]];
            double s = 0;
            for (int j = 0; j < i; j++) {
                s += lu[i][j] * y[j];
            }
            y[i] -= s;
        }
        for (int i = n - 1; i >= 0; i--) {        // backward: U x = y
            double s = 0;
            for (int j = i + 1; j < n; j++) {
                s += lu[i][j] * y[j];
            }
            y[i] = (y[i] - s) / lu[i][i];
        }
        return y;
    }
```

Kode 12.4 (Lanjutan) *Power.java: metode yang sama dalam Java; hasil sebuah iterasi berupa record kecil.*

```
    /** Result of an iteration: eigenvalue, eigenvector, iterations used. */
    record Eigen(double lam, double[] x, int k) { }

    /** Repeat x = normalize(step(x)); the Rayleigh quotient estimates lambda. */
    static Eigen iterate(UnaryOperator<double[]> step, double[][] a, double[] x) {
        double lam = 0, lamOld = Double.NaN;
        for (int k = 1; k <= 200; k++) {
            x = normalize(step.apply(x));
            lam = dot(x, matVec(a, x));           // Rayleigh quotient, |x| = 1
            if (!Double.isNaN(lamOld)
                    && Math.abs(lam - lamOld) <= 1e-10 * Math.abs(lam)) {
                return new Eigen(lam, x, k);
            }
            lamOld = lam;
        }
        return new Eigen(lam, x, 200);
    }

    static void report(String name, Eigen e) {
        double f = Math.sqrt(e.lam()) / (2 * Math.PI);
        StringBuilder shape = new StringBuilder();
        double[] x = e.x();
        for (int i = 0; i < x.length; i++) {       // top floor scaled to 1
            shape.append(i > 0 ? "  " : "")
                 .append(String.format(Locale.ROOT, "%6.3f",
                                       x[i] / x[x.length - 1]));
        }
        System.out.println(String.format(Locale.ROOT,
            "%-7s lambda %10.3f  f %5.2f Hz  iterations %3d  shape %s",
            name, e.lam(), f, e.k(), shape));
    }

    static double[][] copy(double[][] a) {
        double[][] c = new double[a.length][];
        for (int i = 0; i < a.length; i++) {
            c[i] = a[i].clone();
        }
        return c;
    }
```

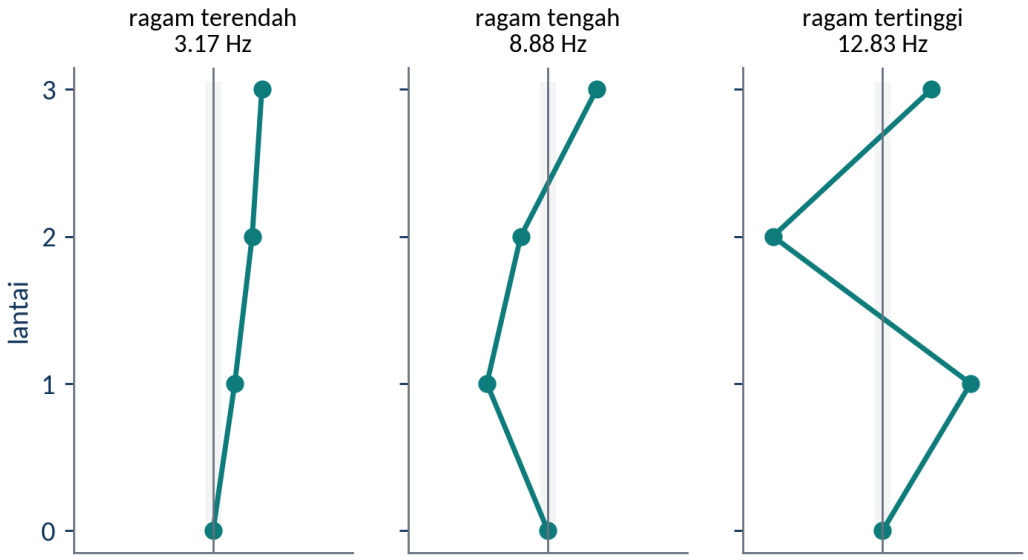
Kode 12.4 (Lanjutan) *Power.java: metode yang sama dalam Java; hasil sebuah iterasi berupa record kecil.*

```
    public static void main(String[] args) {
        double[] start = {1.0, 1.0, 1.0};
        report("highest", iterate(v -> matVec(A, v), A, start));
        double[][] lu = copy(A);
        int[] piv = luFactor(lu);                 // factor once, solve many times
        report("lowest", iterate(v -> luSolve(lu, piv, v), A, start));
        double shift = 3000.0;
        double[][] luS = copy(A);
        for (int i = 0; i < 3; i++) {
            luS[i][i] -= shift;
        }
        int[] pivS = luFactor(luS);
        report("middle", iterate(v -> luSolve(luS, pivS, v), A, start));
    }
}
```

Kode 12.4 mencetak tiga baris yang sama:

```
highest lambda   6493.959  f 12.83 Hz  iterations  18  shape  1.802  -2.247   1.000
lowest  lambda    396.125  f  3.17 Hz  iterations   7  shape  0.445   0.802   1.000
middle  lambda   3109.916  f  8.88 Hz  iterations   6  shape -1.247  -0.555   1.000
```

Ragam terendah, 3,17 Hz, adalah goyangan lambat tempat ketiga lantai bergerak searah, atap paling jauh (bentuk 0.445,0.802,1). Ragam tengah, 8,88 Hz, membengkokkan bangunan sehingga lantai-lantai bawah bergerak berlawanan dengan atap; ragam tertinggi, 12,83 Hz, membuat lantai-lantai yang bertetangga bergerak berlawanan arah. **Gambar 12.2** menggambar ketiga bentuk itu. Periode fundamentalnya $1/3.17 = 0.32$ s, sejalan dengan aturan praktis sekitar 0,1 s per tingkat untuk bangunan rendah.



Gambar 12.2 Ketiga bentuk ragam model ruko: simpangan ke samping setiap lantai (tanah di 0), diskalakan agar atap bergerak satu satuan, beserta frekuensi alami setiap ragam.

Banyaknya iterasi memastikan teorinya. Metode pangkat memerlukan 18 langkah, karena hasil bagi Rayleigh membaik dengan faktor $(\lambda_2/\lambda_1)^2 = (3110/6494)^2 = 0.23$ per langkah. Iterasi invers memerlukan 7: rasionya $(396/3110)^2 = 0.016$. Iterasi invers bergeser memerlukan 6: geseran 3000 berjarak 110 dari nilai eigen tengah dan 2604 dari nilai eigen lain yang terdekat, rasio $(110/2604)^2 = 0.0018$ per langkah. Geseran yang baik mengubah metode yang lambat menjadi cepat.

12.5.3 Atap yang lebih berat

Banyak pemilik ruko menaruh tandon air, tempat menjemur cucian, atau bahkan ruangan tambahan di atap. Apa pengaruh atap yang lebih berat pada frekuensinya? Dengan massa atap $1.5m$, bukan m, matriks massanya tidak lagi kelipatan matriks identitas, dan Persamaan (12.6) menjadi masalah nilai eigen *tergeneralisasi* $Kv = \omega^2 Mv$ dengan $M = m\text{diag}(1,1,1.5)$. Masalah itu dapat direduksi menjadi masalah biasa, $M^{-1}Kv = \omega^2 v$, tetapi $M^{-1}K$ tidak simetris, sehingga kecepatan ganda hasil bagi Rayleigh hilang. Reduksi yang lebih baik mempertahankan kesimetrisan: dengan $w = M^{1/2}v$ masalahnya menjadi $M^{-1/2}KM^{-1/2}w = \omega^2 w$, matriks yang simetris, dan M yang diagonal membuat $M^{-1/2}$ mudah dihitung. Rutin pustaka menerima pasangan itu secara langsung: `scipy.linalg.eigh(K, M)` menyelesaikan masalah tergeneralisasi simetris. Untuk atap yang lebih berat hasilnya frekuensi 2,80, 8,33, dan 12,62 Hz. Frekuensi fundamental turun dari 3,17 ke 2,80 Hz, dan periode fundamental naik dari 0,32 ke 0,36 s. Puncak yang lebih berat membuat bangunan bergoyang lebih lambat, dan itulah sebabnya insinyur harus

memperhitungkan segala sesuatu yang akan berdiri di atas atap, dan mengapa atap yang diubah untuk menanggung beban berat perlu diperiksa insinyur.

12.5.4 Mengapa frekuensi itu penting: resonansi

Mengapa para insinyur begitu memedulikan angka-angka ini? Pandanglah satu ragam sebagai satu massa pada pegas dengan frekuensi alami f_n dan sedikit redaman, dengan rasio redaman ζ seperti di Bab 11, yang diguncang pada dasarnya dengan frekuensi f. Ayunan tunaknya adalah gerakan tanah dikalikan faktor penguatan

$$\frac{1}{\sqrt{(1-r^2)^2+(2\zeta r)^2}}, \qquad r=\frac{f}{f_n}.$$

Untuk bangunan beton ζ biasanya sekitar 5% [6]. Guncangan tanah pada 1 Hz, jauh di bawah frekuensi fundamental ruko 3,17 Hz, memberi $r = 0.32$ dan faktor sekitar 1,1: bangunan itu sekadar ikut terbawa. Guncangan pada 3 Hz memberi $r = 0.95$ dan faktor sekitar 7; tepat pada 3,17 Hz faktornya $1/(2\zeta) = 10$. Gerakan tanah yang sama dapat menjadi ayunan lembut bagi satu bangunan dan ayunan berbahaya bagi bangunan lain, hanya bergantung pada sebuah nilai eigen. Itulah sebabnya standar perancangan pertama-tama meminta periode fundamental, dan mengapa atap yang lebih berat di Subbab 12.5.3, yang menggeser frekuensi fundamental dari 3,17 ke 2,80 Hz, dapat berarti: ia memindahkan resonansi bangunan ke bagian lain spektrum gempa.

Bagi insinyur, ragam terendah biasanya yang paling penting, karena ragam itu membawa sebagian besar massa bangunan ke arah yang sama dan karena gerakan tanah gempa yang umum mengandung banyak energi pada frekuensi seperti itu. Perancangan yang sebenarnya memakai model elemen hingga dengan ribuan besaran tak diketahui dan spektrum respons dari standar; model tiga massa ini hanya memperlihatkan gagasannya. Setiap pertanyaan tentang keselamatan bangunan nyata adalah wewenang insinyur struktur yang berlisensi.

12.6 PageRank: Vektor Eigen dari Web

12.6.1 Pembaca acak

Pada 1998 Sergey Brin dan Lawrence Page menggambarkan mesin pencari yang kemudian menjadi Google, dan di jantungnya terdapat masalah vektor eigen [7]. Halaman mana yang penting? Bukan sekadar halaman dengan banyak tautan masuk, karena tautan murah dibuat, melainkan halaman yang ditautkan *dari halaman-halaman penting*. Definisi itu melingkar, dan definisi melingkar semacam itu adalah masalah vektor eigen.

Bayangkan seorang pembaca yang berselancar secara acak. Di setiap halaman ia mengikuti salah satu tautannya, yang dipilih secara acak, dengan peluang d (*faktor redaman*, 0,85 dalam makalah aslinya), atau, dengan peluang $1-d$, merasa bosan dan melompat ke halaman acak di mana saja. Di halaman tanpa tautan, halaman *menggantung* (*dangling*), ia selalu melompat ke halaman acak. **PageRank** sebuah halaman adalah bagian waktu yang ia habiskan di sana dalam jangka panjang. Bila halaman j memiliki L_j tautan keluar, peringkat-peringkatnya memenuhi

$$r_i = \frac{1-d}{n} + d\sum_{j\to i}\frac{r_j}{L_j} + d\sum_{j\ \text{menggantung}}\frac{r_j}{n}, \tag{12.7}$$

dengan n banyaknya halaman dan $j \to i$ berarti "halaman j menautkan halaman i". Dalam bentuk matriks ini adalah $r = Gr$ dengan **matriks Google** $G = dS + \frac{1-d}{n}\mathbf{1}\mathbf{1}^{\mathsf{T}}$, dengan S matriks tautan stokastik-kolom yang kolom-kolom menggantungnya diganti $1/n$. G adalah matriks Markov yang semua entrinya positif, persis seperti matriks binatu, sehingga menurut Perron–Frobenius r adalah distribusi stasionernya yang tunggal, dan metode pangkat menemukannya. Lompatan acak itulah yang menjamin hal ini: tanpanya, sekelompok halaman yang hanya saling menautkan dapat menjebak pembaca, dan peringkatnya tidak akan tunggal. Lompatan itu juga membatasi kecepatannya: nilai eigen kedua G paling besar d dalam nilai mutlak [8], sehingga galatnya mengecil paling tidak dengan faktor 0,85 per langkah, bagaimanapun rupa web-nya.

12.6.2 Jaringan desa wisata

Sebuah desa di perbukitan mempromosikan diri sebagai *desa wisata* dengan delapan situs web kecil: portal desa, sebuah homestay, wisata kebun durian (kebun Anto dari Bab 10 dan 11 kini menerima pengunjung pada musimnya), sebuah warung, sanggar batik, pemandu air terjun, layanan ojek wisata yang mengantar pengunjung berkeliling, dan blog perjalanan yang ditulis seorang pengunjung. **Gambar 12.3** memperlihatkan siapa menautkan siapa. Pemandu air terjun adalah halaman menggantung: ia tidak memiliki tautan keluar.

Kode 12.5 menerapkan Persamaan (12.7) secara langsung, tanpa pernah membangun matriksnya: setiap halaman menyerahkan peringkatnya, dikali d, dalam bagian yang sama kepada halaman-halaman yang ditautkannya; peringkat halaman menggantung disebar merata; dan setiap halaman menerima $(1-d)/n$ dari lompatan acak. Iterasinya berhenti bila total perubahan di bawah 10^{-10}.

Kode 12.5 `pagerank.py`: PageRank dengan metode pangkat, dengan halaman menggantung ditangani secara eksplisit.

```
"""PageRank of the eight websites of a village-tourism network, computed with
the power method on the Google matrix (damping 0.85)."""

PAGES = ["Village portal", "Homestay Melati", "Durian orchard tour",
         "Warung Bu Tini", "Batik workshop", "Waterfall guide",
         "Ojek wisata", "Travel blog"]
LINKS = {0: [1, 2, 3, 4, 5], 1: [0, 2, 3, 6], 2: [0, 1, 3], 3: [0, 2],
         4: [0, 1], 5: [], 6: [1, 2, 5], 7: [2, 1, 5]}   # page 5 has no links
DAMPING = 0.85


def pagerank(links, n, d=DAMPING, tol=1e-10):
    r = [1.0 / n] * n
    for k in range(1, 1000):
        new = [(1 - d) / n] * n
        dangling = sum(r[j] for j in range(n) if not links[j])
        for j in range(n):
            for i in links[j]:                # j shares its rank among its links
                new[i] += d * r[j] / len(links[j])
        for i in range(n):                    # a page without links: to everyone
            new[i] += d * dangling / n
        change = sum(abs(new[i] - r[i]) for i in range(n))
        r = new
        if change < tol:
            return r, k
    return r, k


def main():
    r, k = pagerank(LINKS, len(PAGES))
    print(f"converged after {k} iterations, ranks sum to {sum(r):.12f}")
    order = sorted(range(len(PAGES)), key=lambda i: -r[i])
    for place, i in enumerate(order, 1):
        inbound = sum(i in LINKS[j] for j in LINKS)
        print(f"{place}. {PAGES[i]:20s} {r[i]:.4f}   (linked from {inbound})")


if __name__ == "__main__":
    main()
```

Kode 12.6 `PageRank.java`: program yang sama dalam Java; peringkatnya diurutkan dengan sebuah pembanding.

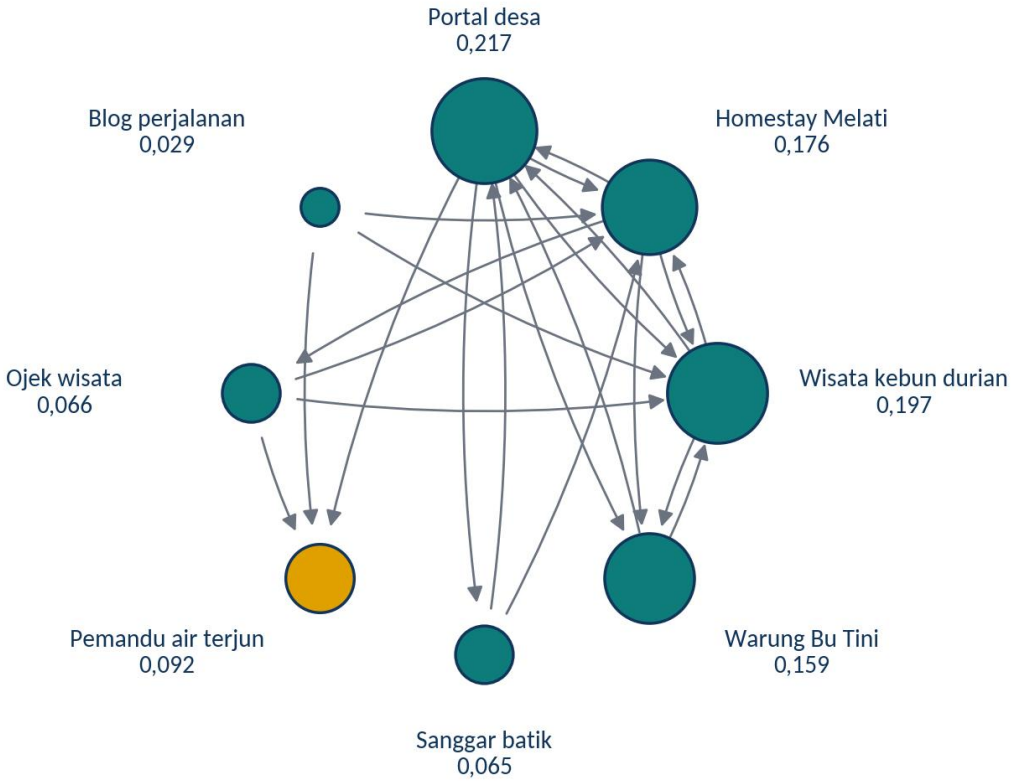
```
import java.util.Arrays;
import java.util.Comparator;
import java.util.Locale;

/** PageRank of the eight websites of a village-tourism network, computed with
 *  the power method on the Google matrix (damping 0.85). */
public class PageRank {
    static final String[] PAGES = {"Village portal", "Homestay Melati",
        "Durian orchard tour", "Warung Bu Tini", "Batik workshop",
        "Waterfall guide", "Ojek wisata", "Travel blog"};
    static final int[][] LINKS = {{1, 2, 3, 4, 5}, {0, 2, 3, 6}, {0, 1, 3}, {0, 2},
                                  {0, 1}, {}, {1, 2, 5}, {2, 1, 5}};  // 5: no links
    static final double DAMPING = 0.85;
    static int iterations;

    static double[] pagerank(int[][] links, double d, double tol) {
        int n = links.length;
        double[] r = new double[n];
        Arrays.fill(r, 1.0 / n);
        for (int k = 1; k < 1000; k++) {
            double[] next = new double[n];
            Arrays.fill(next, (1 - d) / n);
            double dangling = 0;
            for (int j = 0; j < n; j++) {
                if (links[j].length == 0) {
                    dangling += r[j];
                }
            }
            for (int j = 0; j < n; j++) {
                for (int i : links[j]) {           // j shares its rank among links
                    next[i] += d * r[j] / links[j].length;
                }
            }
            for (int i = 0; i < n; i++) {          // a page without links: to all
                next[i] += d * dangling / n;
            }
            double change = 0;
            for (int i = 0; i < n; i++) {
                change += Math.abs(next[i] - r[i]);
            }
            r = next;
            iterations = k;
            if (change < tol) {
                break;
            }
        }
```

Kode 12.6 (Lanjutan) *PageRank.java: program yang sama dalam Java; peringkatnya diurutkan dengan sebuah pembanding.*

```
        return r;
    }

    public static void main(String[] args) {
        double[] r = pagerank(LINKS, DAMPING, 1e-10);
        double sum = 0;
        for (double v : r) {
            sum += v;
        }
        System.out.println(String.format(Locale.ROOT,
            "converged after %d iterations, ranks sum to %.12f", iterations, sum));
        Integer[] order = new Integer[r.length];
        for (int i = 0; i < r.length; i++) {
            order[i] = i;
        }
        Arrays.sort(order, Comparator.comparingDouble(i -> -r[i]));
        for (int place = 0; place < order.length; place++) {
            int i = order[place];
            int inbound = 0;
            for (int[] out : LINKS) {
                for (int t : out) {
                    inbound += t == i ? 1 : 0;
                }
            }
            System.out.println(String.format(Locale.ROOT,
                "%d. %-20s %.4f   (linked from %d)",
                place + 1, PAGES[i], r[i], inbound));
        }
    }
}
```

Kode 12.6 dan program Python mencetak:

```
converged after 25 iterations, ranks sum to 1.000000000000
1. Village portal       0.2168   (linked from 4)
2. Durian orchard tour  0.1969   (linked from 5)
3. Homestay Melati      0.1757   (linked from 5)
4. Warung Bu Tini       0.1585   (linked from 3)
5. Waterfall guide      0.0922   (linked from 3)
6. Ojek wisata          0.0659   (linked from 1)
7. Batik workshop       0.0654   (linked from 1)
8. Travel blog          0.0285   (linked from 0)
```



Gambar 12.3 Jaringan desa wisata: panah adalah tautan, luas lingkaran sebanding dengan PageRank, dan pemandu air terjun (kuning tua) adalah halaman menggantung tanpa tautan keluar.

Langkah pertamanya layak dikerjakan dengan tangan untuk satu halaman, agar Persamaan (12.7) terlihat bekerja. Semua halaman mulai dengan $r_j = 1/8 = 0.125$. Portal desa ditautkan dari homestay (yang memiliki 4 tautan), wisata kebun durian (3 tautan), warung (2 tautan), dan sanggar batik (2 tautan), dan pemandu air terjun adalah halaman menggantung. Jadi setelah satu langkah portal memiliki $0.15/8 + 0.85 \times 0.125 \times \left(\frac{1}{4} + \frac{1}{3} + \frac{1}{2} + \frac{1}{2}\right) + 0.85 \times 0.125/8 = 0.01875 + 0.16823 + 0.01328 = 0.20026$. Setelah satu langkah saja nilainya sudah dekat dengan nilai akhirnya 0,2168, karena sebagian besar warisannya datang dari halaman-halaman yang peringkatnya sendiri dekat dengan rata-rata. Program mengulang pembukuan yang persis sama untuk kedelapan halaman, 25 kali.

Mengapa 0,85? Faktor redaman menyeimbangkan dua keinginan. d yang kecil memberi lompatan acak bobot besar, sehingga peringkat-peringkatnya nyaris sama dan hanya sedikit bercerita tentang tautan: dengan $d = 0.5$ kedelapan peringkat berada antara 0,07 dan 0,17. d yang dekat dengan 1 membiarkan tautan berbicara hampir sendirian, tetapi nilai eigen kedua G mendekati 1, sehingga metode pangkat melambat pada web yang besar, dan peringkatnya menjadi peka terhadap perubahan kecil pada tautan: dengan $d = 0.99$ blog perjalanan jatuh ke 0,011 dan portal naik ke 0,238, dan program yang sama memerlukan 30 iterasi, bukan 25 di sini, dan jauh lebih banyak pada web yang sebenarnya. Brin dan Page memilih 0,85 sebagai jalan tengah [7], dan nilai itu tetap menjadi nilai yang lazim [9].

Tiga pelajaran menonjol. Portal desa berada di peringkat pertama walaupun wisata kebun durian dan homestay memiliki lebih banyak tautan masuk (masing-masing lima, dibandingkan empat): portal ditautkan dari halaman-halaman penting, yaitu homestay, kebun, dan warung, yang

meneruskan bagian besar peringkatnya. Pemandu air terjun, yang ditautkan dari tiga halaman, berperingkat jauh di atas sanggar batik dan layanan ojek, yang masing-masing ditautkan dari satu halaman, walaupun ia tidak menautkan siapa pun; peringkatnya tidak lenyap, karena aturan halaman menggantung menyebarkannya kembali ke jaringan. Dan blog perjalanan, yang tidak ditautkan siapa pun, hanya mendapat jatah minimum yang diberikan lompatan acak kepada setiap halaman, $(1-d)/n = 0.15/8 = 0.019$, ditambah bagiannya dari peringkat halaman menggantung. Iterasinya konvergen dalam 25 langkah, jauh lebih cepat daripada kasus terburuk 0.85^k, karena jaringan kecil ini tercampur dengan baik (Subbab 12.7 memperlihatkan nilai eigen keduanya 0,38).

Varian yang berguna mengubah tempat pembaca yang bosan itu melompat. Pada PageRank *terpersonalisasi*, lompatan acak selalu mendarat di satu halaman pilihan, atau sekumpulan halaman pilihan, alih-alih di mana saja. Selalu melompat ke wisata kebun durian menjawab pertanyaan "bermula dari kebun, halaman mana yang akhirnya dibaca pengunjung?". Mengubah suku lompatan Persamaan (12.7) dengan cara itu (dan mengirim peringkat halaman menggantung ke kebun pula) memberi kebun 0,341, portal 0,217, warung 0,167, dan homestay 0,159, sedangkan blog perjalanan turun ke 0, karena tidak ada halaman yang menautkannya dan lompatan tidak pernah mendarat di sana. Pemilik kebun belajar bahwa pengunjungnya sebagian besar mengalir ke portal, warung, dan homestay, yang merupakan mitra alami untuk promosi bersama. Mesin pencari telah memakai varian terpersonalisasi semacam ini untuk pemeringkatan menurut topik [9].

Mesin pencari awal berhenti di sini. Mesin pencari modern menggabungkan banyak sinyal, dan struktur tautan hanya salah satunya, sedangkan mesin penjawab AI, yang makin sering langsung menjawab pertanyaan, memilih sumber yang dikutipnya dengan sinyal-sinyal lain lagi; tetapi gagasan bahwa kepentingan diwariskan melalui tautan, dan bahwa kepentingan itu dihitung sebagai vektor eigen, tetap menjadi salah satu gagasan pokok web [9].

12.7 Memeriksa terhadap Pustaka

`numpy.linalg` milik NumPy menghitung semua nilai eigen matriks padat sekaligus: `eig` untuk matriks umum dan `eigh` atau `eigvalsh` untuk matriks simetris, dengan rutin LAPACK yang berbasis algoritma QR, metode pilihan untuk matriks padat kecil dan menengah [2], [10]. Rilis terbarunya NumPy 2.5.3 tertanggal 6 September 2026 [11]. Untuk matriks jarang yang sangat besar, `scipy.sparse.linalg.eigs` dan `eigsh` menemukan beberapa nilai eigen dengan metode subruang Krylov, keturunan canggih metode pangkat yang tidak hanya memakai vektor terakhir tetapi semua vektor sebelumnya. **Kode 12.7** memeriksa ketiga masalah bab ini.

Kode 12.7 `compare_numpy.py`: pasar binatu, ruko, dan PageRank dengan `numpy.linalg`.

```
"""Check this chapter's eigenvalues with NumPy."""
import numpy as np
import scipy
from market import P
from power import A
from pagerank import LINKS, DAMPING

print(f"NumPy {np.__version__}, SciPy {scipy.__version__}")
w, v = np.linalg.eig(np.array(P))
i = int(np.argmax(w.real))
print("market eigenvalues:", np.round(np.sort(w.real)[::-1], 6))
print("stationary shares: ", np.round(v[:, i].real / v[:, i].real.sum(), 6))
w = np.linalg.eigvalsh(np.array(A))                  # symmetric: eigvalsh
print("building eigenvalues:", np.round(w, 3))
print("frequencies (Hz):    ", np.round(np.sqrt(w) / (2 * np.pi), 2))
n = len(LINKS)
S = np.zeros((n, n))
for j, out in LINKS.items():
    if out:
        S[out, j] = 1 / len(out)
    else:
        S[:, j] = 1 / n                              # dangling page
G = DAMPING * S + (1 - DAMPING) / n                  # the Google matrix
w, v = np.linalg.eig(G)
i = int(np.argmax(w.real))
r = v[:, i].real / v[:, i].real.sum()
print("PageRank:", np.round(r, 4))
print(f"second eigenvalue of G: {sorted(abs(w))[-2]:.4f}")
```

```
NumPy 2.4.4, SciPy 1.17.1
market eigenvalues: [1.   0.65 0.5 ]
stationary shares:  [0.457143 0.342857 0.2     ]
building eigenvalues: [ 396.125 3109.916 6493.959]
frequencies (Hz):     [ 3.17  8.88 12.83]
PageRank: [0.2168 0.1757 0.1969 0.1585 0.0654 0.0922 0.0659 0.0285]
second eigenvalue of G: 0.3847
```

Semuanya sepakat: nilai eigen pasar 1, 0,65, dan 0,5 beserta pangsa stasionernya; ketiga nilai eigen dan frekuensi bangunan; dan vektor PageRank, yang di sini dihitung dengan membangun matriks Google 8×8 secara lengkap dan meminta semua nilai eigennya. Nilai eigen kedua matriks itu, 0,3847, menjelaskan 25 iterasi tadi: $0.3847^{25} \approx 4 \times 10^{-11}$. Perhatikan dua rincian praktis dalam kode itu: `eig` mengembalikan vektor eigen yang panjangnya 1 dengan tanda sembarang, sehingga vektor stasioner harus diskalakan ulang agar berjumlah 1; dan untuk matriks simetris `eigvalsh` lebih cepat sekaligus lebih akurat daripada `eig`, serta menjamin hasil riil.

Untuk program Java tidak ada pustaka nilai eigen baku; program-program itu diperiksa terhadap program Python, yang mencetak karakter yang sama, dan oleh pengujian dalam `EigenTest.java` dan `test_eigen.py`: pangsa stasioner tidak berubah oleh satu perkalian, metode pangkat menemukan 3 untuk diag(3,1), pemecah LU menyelesaikan sistem yang diketahui, dan vektor PageRank berjumlah 1.

12.8 Metode Numerik dalam Praktik: Profitina

Profitina mengukur seberapa siap situs web sebuah merek untuk dikutip oleh mesin penjawab AI, dan salah satu kelompok sinyalnya menyangkut *kedudukan teknis*: seberapa baik sebuah situs dibangun dan bagaimana ia terhubung. Kedudukan di web sejak dulu diwariskan melalui tautan, yaitu gagasan Subbab 12.6; dan di dalam satu situs web, aritmetika yang sama menggambarkan halaman mana yang dijadikan pusat oleh tautan internal situs itu sendiri. Ketika pemindai

Profitina merayapi sebuah situs, ia membangun struktur ketetanggaan halaman-halamannya, dan pertanyaan seperti "halaman mana yang paling sering dicapai perayap yang mengikuti tautan internal?" atau "apakah halaman yang paling dipedulikan pemiliknya terkubur tiga klik dalamnya dengan sedikit tautan yang menunjuk kepadanya?" adalah pertanyaan tentang vektor eigen dominan graf tautan itu. Situs yang halaman produk pentingnya hanya menerima sedikit tautan internal sedang memberi tahu setiap perayap, manusia maupun mesin, bahwa halaman itu tidak penting. Rekomendasi Profitina tentang tautan internal mengikuti kelas analisis ini; sinyal-sinyal yang persis dan bobotnya adalah milik Profitina sendiri, tetapi gagasan vektor eigennya milik bab ini.

Rantai Markov juga muncul di sisi komunitas platform itu. Pembaca berpindah antarkanal komunitas waktu-nyata kurang lebih seperti pelanggan berpindah antarbinatu, dan pangsa perhatian jangka panjang yang diterima setiap kanal adalah distribusi stasioner dari perpindahan semacam itu. Mengetahui bahwa jawabannya adalah vektor eigen, dan bahwa kecepatan sebuah kanal baru mencapai pangsa jangka panjangnya ditentukan oleh nilai eigen kedua, mengubah kesan samar ("kanal baru itu mulai diminati") menjadi angka yang dapat diperiksa. Seperti biasa, angka-angka seperti itu memberi saran; orang-orang yang mengelola merek atau kanal itulah yang memutuskan.

12.9 Memakainya Sungguhan

- **Bangun tabel perpindahan dari hitungan nyata.** Untuk sebuah toko, tanyai sejumlah pelanggan di mana mereka berbelanja bulan lalu dan bulan ini; hitung, lalu bagi setiap kolom dengan totalnya. Dengan kurang dari beberapa lusin pelanggan per toko, pecahannya masih kasar; katakan hal itu ketika Anda melaporkan hasilnya.
- **Periksa bahwa jumlah setiap kolom 1.** Kolom yang berjumlah 0,97 berarti ada pelanggan yang hilang (mereka berhenti memakai binatu mana pun); tambahkan keadaan keempat, "tidak memakai binatu", alih-alih menyembunyikan kehilangan itu.
- **Lihat λ_2, bukan hanya vektor eigennya.** Pangsa stasioner memberi tahu ke mana pasar menuju; $_2$ memberi tahu seberapa cepat. Dengan $\lambda_2 = 0.65$ separuh dari selisih yang tersisa tertutup setiap $\ln 0.5 / \ln 0.65 \approx 1.6$ bulan; dengan $\lambda_2 = 0.95$ diperlukan tiga belas bulan.
- **Ukur seberapa jauh survei itu dapat dipercaya.** Dua ratus rumah tangga memberi pecahan yang bisa saja meleset beberapa poin persentase. Cara sederhana untuk melihat artinya bagi jawaban adalah pencuplikan ulang: ambil dua ratus rumah tangga secara acak, dengan pengembalian, dari jawaban survei, bangun ulang matriks perpindahannya, hitung ulang pangsa stasionernya, lalu ulangi seratus kali. Sebaran pangsa jangka panjang Bersih dari keseratus putaran itu adalah batas galat yang jujur bagi angka 45,7%, dan hanya memerlukan beberapa baris kode di sekitar `market.py`.
- **Perbarui matriksnya ketika bulan-bulan baru berdatangan.** Matriks perpindahan adalah potret sesaat sebuah perilaku. Ulangi survei setiap beberapa bulan, dan perhatikan apakah pangsa stasioner dan λ_2 bergeser; perubahan mendadak sering berarti ada pesaing baru atau harga baru, yang sudah terlihat pada angka-angka sebelum ada yang mengatakannya.
- **Cobalah pertanyaan andai-andai.** Ubah satu entri (kesetiaan Wangi dari 0,60 menjadi 0,75, dengan menyeimbangkan kembali kolomnya) lalu jalankan ulang; vektor eigen yang baru adalah pasar setelah perubahan itu. Inilah disiplin "ubah satu hal, putar ulang" yang sama seperti di bab-bab sebelumnya.
- **Untuk situs web Anda sendiri, hitung PageRank pada tautan internalnya.** Daftarkan setiap halaman dan halaman-halaman yang ditautkannya, seperti pada `LINKS`, jalankan `pagerank.py`,

lalu bandingkan peringkatnya dengan halaman-halaman yang benar-benar ingin Anda temukan oleh pengunjung dan perayap. Halaman yang penting bagi Anda tetapi berperingkat rendah memerlukan lebih banyak tautan internal dari halaman-halaman yang berperingkat tinggi.

- **Untuk bangunan, serahkan perancangan kepada insinyur.** Model ruko mengajarkan mengapa bangunan memiliki frekuensi kesukaan; model itu bukan penilaian keselamatan. Retakan setelah gempa, lantai yang melendut, atau getaran baru adalah alasan untuk memanggil insinyur struktur.

12.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Mencampur konvensi baris dan kolom.** Sebagian buku menulis matriks Markov dengan jumlah baris 1 dan mengalikan vektor baris dari kiri, $x_{k+1}^{\top} = x_k^{\top} P$. Matriks kita berjumlah kolom 1 dan dikalikan dari kanan. Keduanya benar; mencampurnya diam-diam mentranspos matriks dan memberi pangsa yang keliru. Periksa jumlahnya.
- **Lupa menormalkan.** Tanpa normalisasi $A^k x$ meluap untuk $|\lambda_1| > 1$ dan menyusut habis untuk $|\lambda_1| < 1$. (Untuk matriks stokastik jumlahnya terjaga, dan karena itulah `market.py` boleh melewatinya.)
- **Berhenti karena perubahan vektor kecil padahal nilai eigennya berdekatan.** Bila $|\lambda_2/\lambda_1| = 0.999$, vektornya hanya sedikit berubah per langkah jauh sebelum akurat. Taksir rasionya dari perubahan berturutan, seperti yang dilakukan `market.py`, dan gunakan geseran bila rasionya dekat dengan 1.
- **Mengharapkan metode pangkat menemukan nilai eigen kompleks atau yang kembar.** Bila λ_1 dan λ_2 bernilai mutlak sama (misalnya 1 dan -1, atau sepasang bilangan kompleks), metode pangkat biasa tidak konvergen. Gunakan `numpy.linalg.eig`.
- **Membalik matriks dalam iterasi invers.** Faktorkan $A - \sigma I$ sekali lalu selesaikan; jangan pernah membentuk inversnya.
- **Menggeser tepat ke sebuah nilai eigen.** Maka $A - \sigma I$ singular. Dalam praktik, geseran yang sangat dekat dengan nilai eigen justru baik, bahkan ideal: sistem yang hampir singular itu diselesaikan dengan hasil yang besar tetapi akurat ke arah yang benar, dan normalisasi mengurus ukurannya.
- **Mengabaikan halaman menggantung.** Bila halaman tanpa tautan dibiarkan di luar Persamaan (12.7), peringkat bocor pada setiap langkah dan vektornya tidak lagi berjumlah 1.

12.11 Ringkasan Bab

- Vektor eigen A adalah arah yang hanya diregangkan A; faktor peregangannya adalah nilai eigen (Persamaan (12.1)). Nilai eigen menggambarkan apa yang dilakukan sebuah sistem bila dibiarkan sendiri.
- Pelanggan yang berpindah antarbinatu membentuk rantai Markov (Persamaan (12.2)). Pangsa stasionernya adalah vektor eigen untuk nilai eigen 1, dan jarak kepadanya mengecil seperti $|\lambda_2|^k$ (Persamaan (12.3)). Untuk kampung itu: Bersih 45,7%, Kilat 34,3%, Wangi 20,0%, didekati dengan laju 0,65 per bulan.
- Metode pangkat mengulang $x \leftarrow Ax/\|Ax\|$ dan konvergen ke vektor eigen dominan dengan laju $_2/_1$; untuk matriks simetris hasil bagi Rayleigh (Persamaan (12.4)) menggandakan laju untuk nilai eigennya.
- Iterasi invers dan iterasi invers bergeser (Persamaan (12.5)) menemukan nilai eigen terdekat dengan geseran yang dipilih, memakai satu faktorisasi LU dan penyelesaian murah per langkah.

Keduanya memberi ketiga frekuensi alami ruko (Persamaan (12.6)), 3,17, 8,88, dan 12,83 Hz, dalam 7, 6, dan 18 langkah.

- Bila dua nilai eigen terbesar bernilai mutlak sama, seperti pada rantai periodik atau rotasi, metode pangkat biasa tidak konvergen; geseran atau algoritma QR dari pustaka dapat.
- Atap yang lebih berat mengubah masalah ruko menjadi masalah nilai eigen tergeneralisasi $Kv = \omega^2 Mv$, yang diselesaikan langsung oleh `scipy.linalg.eigh(K, M)`; frekuensi fundamentalnya turun dari 3,17 ke 2,80 Hz.
- PageRank (Persamaan (12.7)) adalah distribusi stasioner seorang pembaca acak dengan redaman; metode pangkat menemukannya, dan redaman menjamin jawaban yang tunggal serta kekonvergenan paling lambat dengan laju d.
- `numpy.linalg.eig` dan `eigvalsh` menghitung semua nilai eigen matriks padat kecil; `scipy.sparse.linalg.eigs` menemukan beberapa untuk matriks jarang yang besar.

12.12 Soal Latihan

1. Dengan tangan, hitung pangsa binatu setelah bulan pertama dan kedua dari 40%, 35%, dan 25%, lalu periksa terhadap program. Kemudian buktikan bahwa $s = (16/35, 12/35, 7/35)$ memenuhi $Ps = s$.
2. Dengan hanya memakai teras dan determinan P serta nilai eigen 1 yang sudah diketahui, temukan dua nilai eigen lainnya. Ramalkan pada bulan ke berapa total perubahan pertama kali turun di bawah 10^{-6}, bila pada bulan ke-15 besarnya 6.91×10^{-5}.
3. Deflasi Hotelling: setelah pasangan eigen dominan (λ_1, v_1) sebuah matriks simetris diketahui, dengan $\| v_1 \| = 1$, matriks $B = A - \lambda_1 v_1 v_1^{\mathsf{T}}$ memiliki vektor eigen yang sama, dengan λ_1 diganti 0. Tulis program dalam Python dan Java yang menemukan ragam tertinggi ruko dengan metode pangkat, mendeflasikannya, lalu menemukan ragam tertinggi kedua dengan metode pangkat sekali lagi.
4. Sanggar batik membujuk blog perjalanan untuk menambahkan tautan kepadanya, sehingga blog itu menautkan wisata kebun durian, homestay, pemandu air terjun, dan sanggar batik. Ramalkan secara kualitatif seberapa besar peringkat sanggar itu akan naik, lalu ubah `LINKS` dan jalankan programnya.
5. Teorema Gershgorin menyatakan bahwa setiap nilai eigen A terletak di dalam paling tidak satu cakram yang berpusat di a_{ii} dengan jari-jari \${j i} a{ij}\$. Terapkan pada matriks ruko $A = K/m$. Batas apa yang diberikannya untuk frekuensi-frekuensinya?
6. Dari nilai-nilai eigen matriks ruko, ramalkan berapa iterasi yang diperlukan metode pangkat dengan hasil bagi Rayleigh untuk mencapai akurasi relatif 10^{-10}, lalu bandingkan dengan 18 pada program. Lakukan hal yang sama untuk iterasi invers.

12.13 Penyelesaian Soal Latihan

1. Bulan 1: Bersih $0.80 \times 0.40 + 0.15 \times 0.35 + 0.20 \times 0.25 = 0.32 + 0.0525 + 0.05 = 0.4225$; Kilat $0.10 \times 0.40 + 0.75 \times 0.35 + 0.20 \times 0.25 = 0.04 + 0.2625 + 0.05 = 0.3525$; Wangi $0.10 \times 0.40 + 0.10 \times 0.35 + 0.60 \times 0.25 = 0.04 + 0.035 + 0.15 = 0.225$. Bulan 2: Bersih $0.80 \times 0.4225 + 0.15 \times 0.3525 + 0.20 \times 0.225 = 0.338 + 0.052875 + 0.045 = 0.435875$; Kilat $0.04225 + 0.264375 + 0.045 = 0.351625$; Wangi $0.04225 + 0.03525 + 0.135 = 0.2125$. Setelah dibulatkan, inilah 0,4359, 0,3516, dan 0,2125 dari program. Untuk s: komponen pertama Ps adalah $(0.80 \times 16 + 0.15 \times 12 + 0.20 \times 7)/35 = (12.8 + 1.8 + 1.4)/35 = 16/35$; yang kedua $(1.6 + 9 + 1.4)/35 = 12/35$; yang ketiga $(1.6 + 1.2 + 4.2)/35 = 7/35$. Jadi $Ps = s$.

2. Seperti di Subbab 12.3.3, terasnya 2.15 dan $\det P = 0.8(0.75 \times 0.6 - 0.2 \times 0.1) - 0.15(0.1 \times 0.6 - 0.2 \times 0.1) + 0.2(0.1 \times 0.1 - 0.75 \times 0.1) = 0.344 - 0.006 - 0.013 = 0.325$. Kedua nilai eigen lainnya menyelesaikan $\lambda^2 - 1.15\lambda + 0.325 = 0$, sehingga $\lambda = \left(1.15 \pm \sqrt{1.3225 - 1.3}\right)/2 = (1.15 \pm 0.15)/2$, yaitu 0,65 dan 0,5. Perubahannya mengecil sekitar 0,65 per bulan, jadi dari 6.91×10^{-5} diperlukan k bulan lagi dengan $6.91 \times 10^{-5} \times 0.65^k < 10^{-6}$, yaitu $k > \ln(0.01447)/\ln(0.65) = 9.8$: sepuluh bulan lagi, bulan ke-25.

3. **Kode 12.8** dan **Kode 12.9** memakai ulang `iterate` dan `report` dari program ruko.

Kode 12.8 `deflation.py`: deflasi Hotelling untuk menemukan ragam tertinggi kedua.

```
"""Exercise 12.3: after the highest mode, remove it (Hotelling deflation)
and run the power method again to find the second highest."""
from power import A, mat_vec, iterate, report


def main():
    start = [1.0, 1.0, 1.0]
    lam1, v1, k1 = iterate(lambda v: mat_vec(A, v), A, start)
    report("highest", lam1, v1, k1)
    b = [[A[i][j] - lam1 * v1[i] * v1[j] for j in range(3)] for i in range(3)]
    lam2, v2, k2 = iterate(lambda v: mat_vec(b, v), b, start)
    report("second", lam2, v2, k2)


if __name__ == "__main__":
    main()
```

Kode 12.9 `Deflation.java`: program yang sama dalam Java.

```
/** Exercise 12.3: after the highest mode, remove it (Hotelling deflation)
 *  and run the power method again to find the second highest. */
public class Deflation {
    public static void main(String[] args) {
        double[] start = {1.0, 1.0, 1.0};
        Power.Eigen e1 = Power.iterate(v -> Power.matVec(Power.A, v), Power.A,
                                       start);
        Power.report("highest", e1);
        double[][] b = new double[3][3];
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                b[i][j] = Power.A[i][j] - e1.lam() * e1.x()[i] * e1.x()[j];
            }
        }
        Power.report("second", Power.iterate(v -> Power.matVec(b, v), b, start));
    }
}
```

Keduanya mencetak:

```
highest lambda   6493.959  f 12.83 Hz  iterations  18  shape  1.802  -2.247   1.000
second  lambda   3109.916  f  8.88 Hz  iterations   8  shape -1.247  -0.555   1.000
```

Putaran kedua menemukan ragam tengah, 3109,916 dan 8,88 Hz, dalam 8 iterasi, dengan bentuk yang sama seperti yang ditemukan iterasi invers bergeser. B memiliki nilai eigen 0, 3110, dan 396, sehingga metode pangkat pada B konvergen dengan laju $(396/3110)^2$. Deflasi itu sederhana, tetapi galat pada λ_1 dan v_1 terbawa ke dalam B, sehingga setiap deflasi berikutnya kurang akurat; untuk lebih dari beberapa nilai eigen, gunakan iterasi invers bergeser atau pustaka.

4. Blog itu sendiri berperingkat terendah dalam jaringan, sekitar 0,029, dan ia akan membagi suaranya ke empat tautan, bukan tiga. Jadi sanggar itu paling banyak mendapat $0.85 \times 0.029/4 \approx 0.006$ secara langsung, sedikit lebih melalui halaman-halaman yang kemudian ditautkannya.

Dengan `LINKS[7] = [2, 1, 5, 4]` program memberi sanggar itu 0,0714, bukan 0,0654, kenaikan 0,006, dan menempatkannya sedikit di atas layanan ojek. Tautan dari portal desa, yang berperingkat 0,22, akan jauh lebih berharga daripada tautan dari blog: dalam PageRank, yang penting adalah siapa yang menautkan Anda, bukan hanya berapa banyak.

5. Baris-baris $A = 2000\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$ memberi cakram berpusat 4000 berjari-jari 2000, berpusat 4000 berjari-jari 4000, dan berpusat 2000 berjari-jari 2000. Gabungannya adalah selang [0,8000] (nilai eigen matriks simetris riil, sehingga hanya garis riil yang penting). Maka setiap $\lambda = \omega^2$ berada di antara 0 dan 8000, dan setiap frekuensi di bawah $\sqrt{8000}/(2\pi) = 14.2$ Hz. Frekuensi tertinggi yang sebenarnya 12,83 Hz. Batas Gershgorin tidak memerlukan biaya untuk dihitung dan sering sudah cukup untuk memilih geseran atau memeriksa kewajaran jawaban sebuah program.

6. Untuk metode pangkat, galat hasil bagi Rayleigh mengecil dengan faktor $(\lambda_2/\lambda_1)^2 = (3109.9/6494.0)^2 = 0.229$ per langkah. Dari galat relatif awal berorde 1, mencapai 10^{-10} memerlukan $k = \ln(10^{-10})/\ln(0.229) = 15.6$ langkah. Program memerlukan 18: langkah pertama tidak memiliki nilai sebelumnya untuk dibandingkan, dan uji hentinya membandingkan taksiran berturutan, bukan galat sebenarnya. Untuk iterasi invers lajunya $(396.1/3109.9)^2 = 0.0162$, sehingga $k = \ln(10^{-10})/\ln(0.0162) = 5.6$ langkah, dibandingkan 7 pada program, karena alasan yang sama.

Daftar Pustaka

[1] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins University Press, 2013.

[3] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[4] R. von Mises and H. Pollaczek-Geiringer, "Praktische Verfahren der Gleichungsauflösung," *Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 9, pp. 58–77 and 152–164, 1929.

[5] Badan Standardisasi Nasional, *SNI 1726:2019 Tata Cara Perencanaan Ketahanan Gempa untuk Struktur Bangunan Gedung dan Nongedung*. Jakarta, Indonesia: BSN, 2019.

[6] A. K. Chopra, *Dynamics of Structures: Theory and Applications to Earthquake Engineering*, 5th ed. Hoboken, NJ, USA: Pearson, 2017.

[7] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Computer Networks and ISDN Systems*, vol. 30, no. 1–7, pp. 107–117, 1998.

[8] T. H. Haveliwala and S. D. Kamvar, "The second eigenvalue of the Google matrix," Stanford University, Technical Report 2003-20, 2003.

[9] A. N. Langville and C. D. Meyer, *Google's PageRank and Beyond: The Science of Search Engine Rankings*. Princeton, NJ, USA: Princeton University Press, 2006.

[10] NumPy Developers, "Linear algebra (numpy.linalg)," NumPy v2.5 Manual, accessed Sep. 2026. [Online]. Available: https://numpy.org/doc/stable/reference/routines.linalg.html

[11] NumPy Developers, "numpy 2.5.3," Python Package Index, Sep. 6, 2026. [Online]. Available: https://pypi.org/project/numpy/

Bab 13

Optimasi: Golden Section, Newton, dan Gradient Descent

Setiap usaha kecil menanyakan hal yang sama dalam banyak bentuk: pengaturan mana dari tombol-tombol yang saya kendalikan yang memberi hasil terbaik? Warung teh memilih harga, *laundry* memilih berapa kilogram dicuci per putaran, kebun memilih kapan pompa dinyalakan. Bab 2 dan 3 mengajari komputer mencari di mana sebuah fungsi bernilai *nol*; bab ini mengajarinya mencari di mana sebuah fungsi *terbesar* atau *terkecil*. Di akhir bab ini Anda akan mampu menyatakan masalah optimasi dengan kata-kata sederhana dan dengan simbol; mencari nilai terbaik satu variabel dengan *golden-section search* (pencarian bagian emas), yang hanya memerlukan nilai fungsi, dan menjelaskan mengapa selang dipersempit dengan faktor yang sama, 0,618, di setiap langkah; memakai metode Newton untuk optimasi, menurunkan konvergensi kuadratiknya, dan melihat dengan tepat di mana dan mengapa metode itu bisa lari menjauh; beralih ke banyak variabel dengan *gradient descent* (penurunan gradien), menghitung langkah aman terbesar dari matriks Hessian, dan memperkirakan banyaknya langkah dari bilangan kondisinya; memilih langkah secara otomatis dengan *backtracking line search*; menulis semuanya dalam Java dan Python dengan keluaran yang identik; serta memeriksa jawabannya dengan pengoptimal SciPy.

13.1 Harga Mbak Rina

Mbak Rina berjualan *es teh jumbo*, segelas plastik besar teh manis dingin, dari lapak kecil di depan sebuah SMK di Sidoarjo. Teh, gula, es, gelas, dan sedotannya menghabiskan sekitar Rp1.500 per gelas. Selama dua tahun ia memasang harga Rp3.000, karena itulah harga lapak di sebelahnya ketika ia mulai berjualan. Bulan lalu tetangganya menaikkan harga menjadi Rp5.000 dan, di luar dugaan, dagangannya tetap laris. Haruskah ia ikut menaikkan harga?

Ia takut pada dua kesalahan yang berlawanan. Kalau harga terlalu rendah, ia menjual banyak gelas tetapi hampir tidak mendapat apa-apa dari tiap gelas; pada Rp1.500 ia bekerja tanpa upah. Kalau harga terlalu tinggi, tiap gelas memberi untung besar tetapi sedikit siswa yang membeli; pada Rp10.000 ia hanya akan menjual beberapa gelas sehari. Di antara keduanya ada harga yang membuat *laba harian*, yaitu margin per gelas dikali banyaknya gelas, sebesar-besarnya. Mencari harga itu adalah masalah optimasi, dan masalah yang sama dihadapi *laundry* dengan harga per kilogramnya, pengemudi ojek dengan banyaknya jam mengaspal, dan petani dengan takaran pupuknya: makin banyak makin baik sampai titik tertentu, lalu justru memburuk.

Mbak Rina melakukan apa yang dilakukan pedagang yang cermat. Selama empat minggu ia mencoba harga yang berbeda pada hari yang berbeda dan mencatat banyaknya gelas yang terjual. Bab 8 menunjukkan cara mencocokkan kurva pada catatan seperti itu; catatannya digambarkan dengan baik oleh kurva permintaan eksponensial

$$D(p) = A\,e^{-p/s}, \qquad A = 600,\ s = 2.5, \tag{13.1}$$

dengan p harga dalam ribuan rupiah, $D(p)$ banyaknya gelas terjual sehari, A banyaknya gelas yang akan habis kalau teh dibagikan gratis, dan *s skala harga*: setiap kali harga naik s ribu rupiah, penjualan turun menjadi $1/e \approx 37\%$ dari sebelumnya. Pada Rp3.000 Persamaan (13.1) memberi $600\,e^{-1.2} \approx 181$ gelas, cocok dengan penjualannya sekarang; pada Rp5.000 kurva memberi $600\,e^{-2} \approx 81$ gelas.

Dengan biaya $c = 1.5$ ribu rupiah per gelas, laba hariannya adalah

$$\pi(p) = (p - c)\,D(p) = (p - c)\,A\,e^{-p/s}. \tag{13.2}$$

Pada Rp3.000 laba itu $1.5 \times 181 = 271$ ribu rupiah sehari; pada Rp5.000 besarnya $3.5 \times 81 = 284$ ribu. Harga tetangganya memang lebih baik. Tetapi apakah itu yang terbaik? Itulah yang dihitung bab ini, dan sambil jalan bab ini membangun tiga alat yang menjadi bahan hampir semua pengoptimal di dunia sains dan industri.

Di bagian akhir bab, Mbak Rina menambah produk kedua, *gorengan* (tahu, tempe, dan pisang goreng), yang penjualannya bergantung pada kedua harga, karena siswa yang membeli teh sering juga membeli gorengan. Dua harga sekaligus tidak dapat ditemukan dengan mencoba beberapa nilai dengan tangan, dan di situlah *gradient descent* dan metode Newton untuk banyak variabel menunjukkan gunanya.

13.2 Gagasannya dengan Kata-Kata Sederhana

13.2.1 Dari mencari nol ke mencari puncak

Bayangkan kurva laba $\pi(p)$ sebagai sebuah bukit. Di lereng kiri, berjalan ke kanan (menaikkan harga) berarti mendaki; di lereng kanan berarti menurun; di puncak tanahnya datar. "Datar" berarti kemiringan, yaitu turunan $\pi'(p)$, bernilai nol. Jadi salah satu cara mencari puncak adalah mencari nol dari π', dan Bab 2 dan 3 sudah tahu cara mencari nol. Gagasan itu benar dan akan kita pakai, tetapi ia memerlukan turunan, yang tidak selalu tersedia, dan ia tidak bisa membedakan puncak bukit dari dasar lembah, yang tanahnya juga datar.

> **DEFINISI.** *Masalah optimasi* meminta sebuah titik x^* di antara titik-titik yang diizinkan, tempat *fungsi objektif* f bernilai terkecil (masalah *minimasi*) atau terbesar (masalah *maksimasi*). *Minimum lokal* adalah titik tempat f tidak lebih besar daripada di titik mana pun di sekitarnya; *minimum global* adalah titik tempat f tidak lebih besar daripada di mana pun dalam himpunan yang diizinkan. Memaksimumkan f sama dengan meminimumkan $-f$, sehingga metode biasanya hanya ditulis untuk minimasi.

Fungsi satu variabel disebut *unimodal* pada selang $[a, b]$ bila fungsi itu mempunyai tepat satu puncak di sana: naik, lalu turun. Laba Mbak Rina unimodal pada [2,10], dan untuk fungsi unimodal ada cara mencari puncak yang sama sekali tidak memerlukan turunan.

13.2.2 Menjepit puncak di antara dua titik uji

Misalkan kita tahu puncaknya terletak di antara $a = 2$ dan $b = 10$ (Rp2.000 dan Rp10.000). Cicipi kurva di dua titik di dalamnya, $x_1 < x_2$. Jika $\pi(x_1) > \pi(x_2)$, puncak tidak mungkin berada di kanan x_2: seandainya ada di sana, kurva harus naik dari x_1 ke x_2 dalam perjalanannya ke puncak, padahal kurva itu turun. Jadi potongan $(x_2, b]$ boleh dibuang. Jika sebaliknya $\pi(x_1) \le \pi(x_2)$, dengan alasan yang sama kita membuang $[a, x_1)$. Bagaimanapun, selang yang pasti memuat puncak menjadi lebih pendek, dan kita ulangi.

Di mana kedua titik uji harus diletakkan? Dua keinginan menentukannya. Pertama, kita tidak tahu sebelumnya potongan mana yang akan dibuang, jadi kedua kemungkinan selang sisa harus sama panjang; itu membuat titik uji simetris, berjarak $r(b-a)$ dari ujung yang berlawanan, untuk suatu pecahan r. Kedua, setiap langkah seharusnya hanya memerlukan *satu* evaluasi baru π: titik uji yang bertahan di dalam selang baru harus tepat berada di posisi salah satu titik uji selang baru. Subbab 13.3.1 menunjukkan bahwa kedua keinginan itu memaksa $r = \left(\sqrt{5}-1\right)/2 \approx 0.618$, kebalikan rasio emas, yang memberi metode ini namanya.

Mari kerjakan langkah pertama dengan tangan. Dengan $a = 2$, $b = 10$, dan $b - a = 8$:

- $x_1 = b - 0.618034 \times 8 = 10 - 4.944272 = 5.055728$ dan $x_2 = a + 0.618034 \times 8 = 6.944272$.
- $\pi(x_1) = 3.555728 \times 600\, e^{-2.022291} = 3.555728 \times 79.40 = 282.36$.
- $\pi(x_2) = 5.444272 \times 600\, e^{-2.777709} = 5.444272 \times 37.31 = 203.12$.
- $\pi(x_1) > \pi(x_2)$, jadi puncak tidak berada di (6.944272,10]. Selang baru adalah [2,6.944272], panjangnya $4.944272 = 0.618 \times 8$.
- Nilai lama $x_1 = 5.055728$ kini menjadi titik uji *kanan* selang baru, karena $2 + 0.618034 \times 4.944272 = 5.055728$. Hanya titik uji kiri yang baru, $x_1 = 6.944272 - 0.618034 \times 4.944272 = 3.888544$, yang perlu dievaluasi: $\pi(3.888544) = 302.53$.

Bandingkan dengan baris pertama cetakan program di Subbab 13.5: setelah langkah 1 selangnya [2,6.94427] dengan titik uji 3.88854 dan 5.05573 serta laba 302.5347 dan 282.3644. Setiap langkah membuang 38,2% selang, dan setelah n langkah panjangnya 8×0.618^n.

13.2.3 Mengikuti kemiringan: Newton dan gradient descent

Bila turunan tersedia, kita bisa jauh lebih baik daripada sekadar menjepit. Metode Newton dari Bab 3, diterapkan pada persamaan $\pi'(p) = 0$, mengganti kurva di dekat tebakan sekarang dengan parabola yang mempunyai nilai, kemiringan, dan kelengkungan yang sama, lalu melompat ke puncak parabola itu. Di dekat jawaban, banyaknya digit yang benar kira-kira berlipat dua di setiap langkah.

Dengan dua harga, bukit itu menjadi permukaan di atas peta (p_1, p_2), dan kemiringannya adalah vektor, yaitu *gradien*, yang menunjuk ke arah pendakian tercuram. *Gradient descent*, yang ditulis untuk minimasi, mengambil langkah kecil berlawanan arah dengan gradien fungsi rugi, berulang-ulang, seperti pejalan dalam kabut yang selalu melangkah lurus menuruni lereng. Metode ini hanya memerlukan gradien, bukan kelengkungan, sehingga tiap langkahnya murah; tetapi besar langkah sangat menentukan, dan Subbab 13.3.4 menunjukkan cara menghitung langkah terbesar yang masih aman.

> **WAWASAN KUNCI:** Setiap pengoptimal dalam bab ini adalah tawar-menawar antara informasi dan usaha per langkah. *Golden section* hanya memakai nilai dan memperoleh faktor tetap 0,618 per evaluasi. *Gradient descent* memakai kemiringan dan memperoleh faktor tetap per langkah yang bergantung pada bentuk bukit. Newton memakai kemiringan dan kelengkungan dan, di dekat jawaban, mengkuadratkan galat di setiap langkah.

13.3 Metode-Metodenya, Diturunkan

13.3.1 Golden-section search

Misalkan selang sekarang $[a, b]$ dengan panjang $L = b - a$, dan titik uji diletakkan di

$$x_1 = b - rL, \qquad x_2 = a + rL, \qquad \frac{1}{2} < r < 1. \tag{13.3}$$

Dengan titik uji pada Persamaan (13.3), misalkan $f(x_1) > f(x_2)$ (untuk maksimasi), sehingga selang barunya $[a, x_2]$ dengan panjang rL. Di dalamnya titik uji x_1 yang bertahan berjarak $x_1 - a = L - rL = (1-r)L$ dari ujung kiri. Titik uji kanan yang baru harus berada di $a + r(rL) = a + r^2 L$. Agar titik uji lama dapat berperan sebagai titik uji baru, kita perlu

$$r^2 = 1 - r \quad \Leftrightarrow \quad r^2 + r - 1 = 0 \quad \Leftrightarrow \quad r = \frac{\sqrt{5}-1}{2} \approx 0.6180340. \tag{13.4}$$

Akar yang lain negatif dan tidak berguna. Karena simetri, r yang sama berlaku untuk kasus cerminnya. Jadi setiap langkah memerlukan satu evaluasi dan mengalikan panjang selang dengan r. Setelah n langkah,

$$L_n = r^n L_0, \qquad n \geq \frac{\ln(L_0/\varepsilon)}{\ln(1/r)} \approx 2.078 \ln \frac{L_0}{\varepsilon} \tag{13.5}$$

langkah menjamin panjang akhir di bawah toleransi ε. Untuk $L_0 = 8$ dan $\varepsilon = 10^{-4}$, Persamaan (13.5) memberi $n \geq \ln(80000)/0.48121 = 11.290/0.48121 = 23.46$, jadi 24 langkah dan $24 + 2 = 26$ evaluasi, tepat seperti yang dilaporkan program. Ini konvergensi *linear* dengan laju 0,618, sedikit lebih lambat daripada 0,5 per evaluasi pada biseksi; tetapi biseksi memerlukan perubahan tanda, dan puncak tidak mempunyai perubahan tanda, hanya nilai.

Dua fakta menjadikan *golden section* pekerja keras yang andal [1], [2]. Metode ini tidak pernah gagal pada fungsi unimodal, karena setiap langkah menjaga puncak tetap di dalam selang; dan biayanya diketahui sebelum dimulai. Kiefer membuktikan pada 1953 bahwa pencarian Fibonacci, yang limitnya adalah *golden section*, adalah metode terbaik dari jenis ini untuk kasus terburuk [3]. Kelemahannya juga jelas: metode ini menemukan *sebuah* puncak, bukan puncak *tertinggi*, bila fungsinya mempunyai beberapa puncak; dan ketelitiannya terbatas di dekat puncak, tempat kurva datar. Di dekat maksimum x^*, teorema Taylor memberi $f(x) \approx f(x^*) + \frac{1}{2} f''(x^*)(x - x^*)^2$, sehingga perubahan δ pada x hanya mengubah f sekitar $1/2\,|f''|\delta^2$. Bila perubahan itu lebih kecil daripada galat pembulatan f, sekitar $\epsilon_{\text{mach}}|f|$ dengan epsilon mesin dari Bab 1, perbandingan $f(x_1) > f(x_2)$ menjadi seperti melempar koin. Dengan menyamakan keduanya,

$$\delta_{\min} \approx \sqrt{\epsilon_{\text{mach}}} \sqrt{\frac{2|f(x^*)|}{|f''(x^*)|}}, \tag{13.6}$$

yang besarnya sekitar $\sqrt{\epsilon_{\text{mach}}} \approx 1.5 \times 10^{-8}$ dikali suatu faktor skala. Toleransi di bawah itu hanya membuang kerja; inilah sebabnya kode yang baik, seperti yang ada di SciPy, berhenti sekitar 10^{-8} relatif [4].

Pseudokode 13.1 merangkum metode ini dalam bentuk yang diikuti program.

Pseudokode 13.1 *Golden-section search* untuk maksimum fungsi unimodal pada $[a, b]$.

```
GOLDEN-MAX(f, a, b, tol)
 1  r = (sqrt(5) - 1) / 2
 2  x1 = b - r(b - a);  x2 = a + r(b - a)
 3  f1 = f(x1);  f2 = f(x2)
 4  while b - a > tol
 5      if f1 > f2                      // puncak tidak di (x2, b]
 6          b = x2;  x2 = x1;  f2 = f1
 7          x1 = b - r(b - a);  f1 = f(x1)
 8      else                            // puncak tidak di [a, x1)
 9          a = x1;  x1 = x2;  f1 = f2
10          x2 = a + r(b - a);  f2 = f(x2)
11  return (a + b) / 2
```

13.3.2 Metode Newton untuk maksimum

Uraikan f di sekitar tebakan sekarang x_k sampai orde kedua:

$$f(x_k + h) \approx f(x_k) + f'(x_k)\,h + \frac{1}{2} f''(x_k)\,h^2.$$

Ruas kanan adalah parabola dalam h. Turunannya, $f'(x_k) + f''(x_k)h$, bernilai nol pada $h = -f'(x_k)/f''(x_k)$, yang memberi langkah Newton untuk optimasi,

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)}. \tag{13.7}$$

Persamaan (13.7) adalah iterasi Newton pencari akar dari Bab 3 yang diterapkan pada f', sehingga ia mewarisi konvergensi kuadratik di dekat titik dengan $f' = 0$ dan $f'' \neq 0$ [1], [2]. Ia juga mewarisi sebuah kebutaan: titik stasioner parabola adalah puncaknya bila $f''(x_k) < 0$ dan *dasarnya* bila $f''(x_k) > 0$. Metode Newton pergi ke mana pun parabolanya datar, entah ke atas entah ke bawah.

Kurva Mbak Rina memungkinkan kita melihat semua ini dengan tepat. Dari Persamaan (13.2), dengan $m = p - c$ sebagai margin,

$$\pi'(p) = Ae^{-p/s}\left(1 - \frac{m}{s}\right), \qquad \pi''(p) = Ae^{-p/s}\left(\frac{m}{s^2} - \frac{2}{s}\right). \tag{13.8}$$

Kemiringan bernilai nol hanya pada $m = s$, yaitu $p^* = c + s = 1.5 + 2.5 = 4.0$: **harga terbaik adalah Rp4.000**, ketika margin sama dengan skala harga. Kelengkungan negatif untuk $m < 2s$ dan positif untuk $m > 2s$; kurva mempunyai titik belok di $p = c + 2s = 6.5$.

Dalam langkah Newton faktor $Ae^{-p/s}$ saling menghapus, dan dengan $e = m - s$ sebagai galat margin (yang sama dengan galat harga), satu baris aljabar memberi

$$e_{k+1} = -\frac{e_k^2}{s - e_k}. \tag{13.9}$$

Tiga pelajaran terkemas dalam Persamaan (13.9). Di dekat jawaban, $e_{k+1} \approx -e_k^2/s$: galat dikuadratkan di setiap langkah, konvergensi kuadratik dengan konstanta $1/s = 0.4$. Mulai dari Rp3.000, $e_0 = -1$, sehingga $e_1 = -1/3.5 = -0.2857$, $e_2 = -0.0816/2.7857 = -0.0293$, $e_3 = -3.4 \times 10^{-4}$, $e_4 = -4.6 \times 10^{-8}$: digitnya berlipat dua seperti dijanjikan. Jauh dari jawaban, ketika e_k mendekati s (titik belok), penyebutnya lenyap dan langkahnya meledak; dan di luar itu, $e_k > s$, galat baru menjadi *positif dan lebih besar*: mulai dari Rp8.000, $e_0 = 4$ dan $e_1 = -16/(2.5 - 4) =$

10.67, melompat ke Rp14.667, dan iterasi terus berbaris menuju harga yang makin tinggi, tempat kurvanya datar dan tehnya tidak laku.

Obatnya sama dengan yang dipakai Bab 3 untuk pencarian akar: pertahankan kurung (*bracket*) dan kembali ke langkah aman bila perlu. Kita menyimpan selang $[a, b]$ tempat π' berganti tanda (positif di a, negatif di b), memperbaruinya dengan tanda π' di setiap titik baru, dan menerima titik Newton hanya bila $\pi'' < 0$ dan titik itu berada di dalam kurung; selain itu kita membagi dua. Hasilnya konvergen kapan pun biseksi konvergen dan secepat Newton begitu sudah dekat. Pustaka numerik menggabungkan dua bahan yang sama, metode yang aman dan metode yang cepat, dalam metode Brent [4], [5].

13.3.3 Banyak variabel: gradien dan Hessian

Sekarang fungsi rugi f bergantung pada vektor $\mathbf{x} = (x_1, \dots, x_n)$. *Gradien*-nya menghimpun turunan parsial pertama dan *Hessian*-nya turunan parsial kedua:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \partial f / \partial x_1 \\ \vdots \\ \partial f / \partial x_n \end{pmatrix}, \qquad H(\mathbf{x}) = \left[\frac{\partial^2 f}{\partial x_i \, \partial x_j} \right]_{i,j=1}^{n}. \tag{13.10}$$

Di dekat titik $\mathbf{x}$, teorema Taylor untuk banyak variabel, yang ditulis dengan besaran pada Persamaan (13.10), menyatakan

$$f(\mathbf{x} + \mathbf{h}) \approx f(\mathbf{x}) + \nabla f(\mathbf{x})^{\mathsf{T}} \mathbf{h} + \frac{1}{2} \, \mathbf{h}^{\mathsf{T}} H(\mathbf{x}) \, \mathbf{h}. \tag{13.11}$$

Di minimum lokal, gradien adalah vektor nol (kalau tidak, langkah kecil yang melawan gradien akan menurunkan f), dan Hessian bersifat *semidefinit positif*: $\mathbf{h}^{\mathsf{T}} H \mathbf{h} \geq 0$ untuk setiap arah $\mathbf{h}$. Jika H definit positif (semua nilai eigennya positif, Bab 12), titik dengan gradien nol adalah minimum lokal sejati. Untuk Hessian simetris 2×2 hal ini mudah diuji: entri diagonal dan determinannya harus positif.

Dua produk Mbak Rina adalah teh dengan harga p_1 dan gorengan dengan harga p_2, dengan biaya $c_1 = 1.5$ dan $c_2 = 0.6$ ribu rupiah. Catatannya memberi kurva permintaan dengan setiap harga juga sedikit menurunkan penjualan produk yang lain, karena keduanya dibeli bersama:

$$T = 600 \, e^{-p_1/2.5 \, - \, 0.10 \, p_2}, \qquad G = 900 \, e^{-p_2 \, - \, 0.08 \, p_1}, \tag{13.12}$$

dan fungsi rugi yang diminimumkan adalah minus laba, $f(p_1, p_2) = -[(p_1 - c_1)T + (p_2 - c_2)G]$. Dengan aturan perkalian, dan $m_1 = p_1 - c_1$ serta $m_2 = p_2 - c_2$, turunan parsial laba adalah

$$\frac{\partial \pi}{\partial p_1} = T - \frac{m_1 T}{2.5} - 0.08 \, m_2 G, \qquad \frac{\partial \pi}{\partial p_2} = G - 0.10 \, m_1 T - m_2 G,$$

dan gradien fungsi rugi adalah negatifnya. Di tebakan awal (3,1): $T = 600 \, e^{-1.3} = 163.52$ gelas, $G = 900 \, e^{-1.24} = 260.45$ gorengan, $m_1 = 1.5$, $m_2 = 0.4$, sehingga $\partial \pi / \partial p_1 = 163.52 - 98.11 - 8.33 = 57.07$ dan $\partial \pi / \partial p_2 = 260.45 - 24.53 - 104.18 = 131.74$. Keduanya positif: menaikkan harga mana pun akan menambah laba, dan harga gorengan lebih mendesak untuk dinaikkan.

Rumus gradien mudah keliru satu tanda atau satu faktor. Asuransi murahnya adalah selisih pusat dari Bab 9: $\partial f/\partial x_1 \approx [f(x_1+h,x_2)-f(x_1-h,x_2)]/(2h)$, dengan galat berorde h^2. Program mencetak keduanya di (3,1) dan keduanya sama sampai enam desimal yang dicetak. Setiap kode optimasi yang serius melakukan *pemeriksaan gradien* (*gradient check*) ini sebelum memercayai gradien yang ditulis dengan tangan.

13.3.4 Gradient descent dan besar langkahnya

Gradient descent mengulang

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\,\nabla f(\mathbf{x}_k), \tag{13.13}$$

dengan *besar langkah* $\alpha > 0$ (dalam *machine learning* disebut *learning rate*, laju belajar) yang kita pilih. Menurut Persamaan (13.11) dengan $\mathbf{h} = -\alpha\nabla f$, fungsi rugi berubah sebesar $-\alpha \parallel \nabla f \parallel^2 + O(\alpha^2)$, jadi langkah yang cukup kecil selalu menurun. Seberapa kecil yang cukup kecil?

Di dekat minimum $\mathbf{x}^*$ gradien kira-kira sama dengan $H(\mathbf{x}-\mathbf{x}^*)$, dengan H Hessian di minimum. Dengan menulis $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^*$, Persamaan (13.13) menjadi $\mathbf{e}_{k+1} = (I - \alpha H)\,\mathbf{e}_k$. Uraikan $\mathbf{e}_k$ menurut vektor-vektor eigen H (Bab 12): sepanjang vektor eigen dengan nilai eigen λ_i, komponennya dikalikan dengan $1-\alpha\lambda_i$ di setiap langkah. Semua komponen mengecil tepat ketika setiap faktor besarnya kurang dari satu:

$$|1-\alpha\lambda_i| < 1 \text{ untuk semua } i \quad\Leftrightarrow\quad 0 < \alpha < \frac{2}{\lambda_{\max}}. \tag{13.14}$$

Komponen yang paling lambat menentukan kecepatannya. Faktor terburuk, $\max_i|1-\alpha\lambda_i|$, paling kecil ketika faktor untuk $\lambda_{\min}$ dan $\lambda_{\max}$ sama besar dan berlawanan tanda, yang terjadi pada $\alpha = 2/(\lambda_{\min}+\lambda_{\max})$, dan saat itu

$$\rho = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}} = \frac{\kappa-1}{\kappa+1}, \qquad \kappa = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{13.15}$$

Bilangan κ adalah *bilangan kondisi* Hessian: perbandingan antara kelengkungan tercuram dan terlandai dari lembahnya. Mangkuk yang bulat mempunyai $\kappa = 1$ dan *gradient descent* mencapai dasarnya dalam satu langkah; lembah yang panjang dan sempit mempunyai κ besar, laju ρ mendekati 1, dan si pejalan zig-zag dari dinding ke dinding sambil merayap di sepanjang dasar lembah [6]. Untuk memperkecil galat dengan faktor 10^{-d} diperlukan sekitar $d\ln 10/\ln(1/\rho)$ langkah, yang untuk κ besar kira-kira $1.15\,d\,\kappa$ langkah.

Untuk dua harga Mbak Rina, program menghitung Hessian di titik optimum: nilai eigennya 49,34 dan 182,06, jadi $\kappa = 3.69$, batas kestabilannya $\alpha < 2/182.06 = 0.01099$, langkah tetap terbaiknya $2/231.39 = 0.00864$, dan Persamaan (13.15) memberi $\rho = 2.69/4.69 = 0.574$. Panjang gradien mula-mula $\sqrt{57.07^2+131.74^2} = 143.6$ dan harus turun di bawah 10^{-6}, faktor 1.4×10^8, jadi kita mengharapkan $\ln(1.4\times10^8)/\ln(1/0.574) = 18.8/0.556 \approx 34$ langkah. Program, dengan $\alpha = 0.008$, memerlukan 35. Dengan $\alpha = 0.002$ lajunya $1-0.002\times49.34 = 0.901$ dan diperlukan 166 langkah. Dengan $\alpha = 0.012$ faktor sepanjang arah curam adalah $1-0.012\times182.06 = -1.18$: galat sepanjang arah itu membesar dan berganti tanda di setiap langkah, dan iterasinya tidak pernah tenang.

> **CONTOH:** Satu langkah gradien dengan tangan. Di titik awal (3,1) labanya 349,46 ribu rupiah dan gradien fungsi ruginya $(-57.07, -131.74)$, negatif dari kemiringan laba yang dihitung di Subbab 13.3.3. Dengan $\alpha = 0.008$, Persamaan (13.13) memberi $p_1 = 3 - 0.008\times(-57.07) =$

3.4566 dan $p_2 = 1 - 0.008 \times (-131.74) = 2.0539$. Laba di sana 367,13, naik hampir 18 ribu rupiah sehari. Tetapi gradien fungsi rugi yang baru adalah $(-16.47, 63.72)$: komponen keduanya berganti tanda, artinya harga gorengan sudah melewati nilai terbaiknya (optimumnya 1,435) dan langkah berikutnya akan menariknya kembali. Harga teh, pada arah yang lebih landai, masih ingin naik. Inilah zig-zag pertama pada **Gambar 13.2**: sepanjang arah yang curam langkah α besar dibanding $1/\lambda_{\max}$, sehingga iterasinya melampaui sasaran; sepanjang arah yang landai langkah itu kecil dibanding $1/\lambda_{\min}$, sehingga iterasinya merayap. Dengan $\kappa = 3.7$ zig-zag itu padam dalam beberapa langkah; dengan $\kappa = 100$ ia akan menguasai seluruh perjalanan.

13.3.5 Backtracking: membiarkan metode memilih langkah

Menghitung nilai eigen Hessian hanya untuk memilih langkah bertentangan dengan tujuan metode yang murah. *Line search* memilih langkah di setiap iterasi dari fungsinya sendiri. Yang paling sederhana dan andal adalah *backtracking* (mundur bertahap): coba $\alpha = 1$; selama fungsi rugi tidak turun setidaknya sebagian kecil dari yang dijanjikan kemiringan, bagi dua α. Syarat yang diperiksa adalah *syarat Armijo* [7]:

$$f(\mathbf{x}_k - \alpha \nabla f_k) \le f(\mathbf{x}_k) - c\,\alpha \;\|\, \nabla f_k \,\|^2, \qquad c = 10^{-4}. \tag{13.16}$$

Karena f turun dengan laju $\| \nabla f \|^2$ untuk langkah yang sangat kecil, langkah yang memenuhi Persamaan (13.16) selalu ada, dan pembagian dua menemukannya dalam beberapa percobaan. *Backtracking* menghilangkan bahaya divergensi sama sekali, dengan harga evaluasi fungsi tambahan: pada masalah Mbak Rina ia memerlukan 35 langkah yang sama dengan langkah tetap terbaik tetapi 294 titik percobaan, karena setiap langkah dimulai dari $\alpha = 1$ dan dibagi dua sekitar delapan kali sampai $1/128 \approx 0.0078$. Perbaikan sederhana, yaitu memulai setiap pencarian dari dua kali langkah sebelumnya, adalah Soal Latihan 13.4. Nocedal dan Wright memberikan teori lengkapnya, termasuk *syarat Wolfe* yang juga membatasi langkah dari bawah [6].

13.3.6 Metode Newton untuk banyak variabel

Meminimumkan model kuadratik Persamaan (13.11) terhadap $\mathbf{h}$ memberi syarat $\nabla f + H\mathbf{h} = \mathbf{0}$, sebuah sistem linear. Langkah Newton adalah penyelesaiannya:

$$H(\mathbf{x}_k)\,\mathbf{s}_k = \nabla f(\mathbf{x}_k), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{s}_k. \tag{13.17}$$

Untuk dua variabel sistem ini diselesaikan dengan aturan Cramer, seperti dalam program; untuk banyak variabel dengan eliminasi Gauss dari Bab 4 atau, karena Hessian di minimum simetris definit positif, dengan faktorisasi Cholesky dari Bab 6. Metode Newton melihat bentuk mangkuk seutuhnya, sehingga ia tidak diperlambat oleh bilangan kondisi yang besar: pada fungsi kuadratik ia mendarat di minimum dalam satu langkah, berapa pun κ. Di dekat minimum dengan Hessian definit positif ia konvergen secara kuadratik [2], [6]. Harganya adalah turunan kedua, sebanyak $n(n+1)/2$, dan penyelesaian sistem linear berorde n^3 per langkah; itulah sebabnya masalah dengan jutaan variabel memakai metode gradien atau metode *quasi-Newton* seperti BFGS, yang membangun Hessian hampiran dari gradien-gradien berturutan [6].

Tabel 13.1 merangkum keempat metode.

Tabel 13.1 Pengoptimal dalam bab ini: apa yang diperlukan, berapa biaya satu langkah, dan seberapa cepat konvergensinya.

Metode	Memerlukan	Biaya per langkah	Konvergensi	Bahaya utama
Golden section	nilai, sebuah kurung	1 evaluasi	linear, laju 0,618	beberapa puncak
Newton (1 variabel)	f', f''	2 turunan	kuadratik dekat x^*	di sisi yang salah dari titik belok
Gradient descent	∇f	1 gradien	linear, $(\kappa - 1)/(\kappa + 1)$	langkah di atas $2/\lambda_{\max}$
Newton (n variabel)	∇f, H	gradien, Hessian, penyelesaian sistem	kuadratik dekat $\mathbf{x}^*$	H tidak definit positif

13.4 Program-Programnya

Kode bab ini berada di `Chapter13/python` dan `Chapter13/java`; setiap program mencetak teks yang persis sama dalam kedua bahasa, sehingga mana pun dapat dibaca berdampingan dengan keluarannya. Tidak ada yang memerlukan pustaka kecuali perbandingan dengan SciPy di Subbab 13.6.

13.4.1 Golden section untuk satu harga

Kode 13.1 mengimplementasikan **Pseudokode 13.1** untuk laba Mbak Rina. Fungsi `golden_max` menyimpan empat bilangan x_1, x_2, f_1, f_2 dan memperbaruinya persis seperti pseudokode; fungsi itu mencetak enam langkah pertama dan langkah terakhir. Setelah pencarian, `main` membandingkan hasilnya dengan optimum eksak $c + s$ dan dengan grid kasar setiap Rp50 antara Rp2.000 dan Rp10.000.

Kode 13.1 `golden.py`: *golden-section search* untuk harga teh terbaik.

```
"""Golden-section search for the price that maximizes Mbak Rina's daily profit.
Usage: python golden.py"""
import math

A, S, C = 600.0, 2.5, 1.5    # cups at price 0, price scale, cost (thousand Rp)
R = (math.sqrt(5.0) - 1.0) / 2.0   # 0.618..., the golden-section ratio


def cups(p):
    """Cups of tea sold in a day at price p (thousand rupiah)."""
    return A * math.exp(-p / S)


def profit(p):
    """Daily profit in thousand rupiah."""
    return (p - C) * cups(p)


def golden_max(f, a, b, tol, trace=False):
    """Maximize a unimodal f on [a, b]; return (x, iterations, evaluations)."""
    x1, x2 = b - R * (b - a), a + R * (b - a)
    f1, f2 = f(x1), f(x2)
    k, evals = 0, 2
    while b - a > tol:
        k += 1
        if f1 > f2:              # the peak cannot lie in (x2, b]
            b, x2, f2 = x2, x1, f1
            x1 = b - R * (b - a)
            f1 = f(x1)
        else:                    # the peak cannot lie in [a, x1)
            a, x1, f1 = x1, x2, f2
            x2 = a + R * (b - a)
            f2 = f(x2)
        evals += 1
        if trace and (k <= 6 or b - a <= tol):
            print(f"{k:3d} {a:9.5f} {b:9.5f} {b - a:9.5f} {x1:9.5f}"
                  f" {x2:9.5f} {f1:10.4f} {f2:10.4f}")
    return (a + b) / 2.0, k, evals
```

Kode 13.1 (Lanjutan) *golden.py: golden-section search untuk harga teh terbaik.*

```
def main():
    print("  k         a         b     b - a        x1        x2"
          "     pi(x1)     pi(x2)")
    p, k, evals = golden_max(profit, 2.0, 10.0, 1e-4, trace=True)
    print(f"golden section: p = {p:.5f} after {k} steps, {evals} evaluations")
    print(f"exact optimum : p = {C + S:.5f} (from pi'(p) = 0)")
    print(f"cups {cups(p):.1f}, profit {profit(p):.3f} thousand rupiah a day")
    grid = [2.0 + 0.05 * i for i in range(161)]     # every Rp 50
    best = max(grid, key=profit)
    print(f"grid of {len(grid)} prices: best p = {best:.2f},"
          f" profit {profit(best):.3f}")


if __name__ == "__main__":
    main()
```

Kode 13.2 adalah program yang sama dalam Java. Fungsi yang dimaksimumkan diberikan sebagai `DoubleUnaryOperator`, tipe Java untuk "fungsi dari `double` ke `double`", dan rujukan metode `Golden::profit` menyediakannya. Semua pemformatan melalui `String.format` dengan `Locale.ROOT`, sehingga pemisah desimalnya titik di setiap komputer, termasuk yang diatur ke bahasa Indonesia, yang secara bawaan akan mencetak koma.

Kode 13.2 `Golden.java`: pencarian yang sama dalam Java.

```java
// Golden-section search for the price that maximizes Mbak Rina's daily profit.
// Usage: java Golden
import java.util.Locale;
import java.util.function.DoubleUnaryOperator;

public class Golden {
    static final double A = 600.0, S = 2.5, C = 1.5;
    static final double R = (Math.sqrt(5.0) - 1.0) / 2.0;

    static double cups(double p) { return A * Math.exp(-p / S); }

    static double profit(double p) { return (p - C) * cups(p); }

    static String f(String fmt, Object... xs) {
        return String.format(Locale.ROOT, fmt, xs);
    }

    /** Maximize a unimodal f on [a, b]; returns {x, iterations, evaluations}. */
    static double[] goldenMax(DoubleUnaryOperator fn, double a, double b,
                              double tol, boolean trace) {
        double x1 = b - R * (b - a), x2 = a + R * (b - a);
        double f1 = fn.applyAsDouble(x1), f2 = fn.applyAsDouble(x2);
        int k = 0, evals = 2;
        while (b - a > tol) {
            k++;
            if (f1 > f2) {                  // the peak cannot lie in (x2, b]
                b = x2; x2 = x1; f2 = f1;
                x1 = b - R * (b - a);
                f1 = fn.applyAsDouble(x1);
            } else {                        // the peak cannot lie in [a, x1)
                a = x1; x1 = x2; f1 = f2;
                x2 = a + R * (b - a);
                f2 = fn.applyAsDouble(x2);
            }
            evals++;
            if (trace && (k <= 6 || b - a <= tol))
                System.out.println(f("%3d %9.5f %9.5f %9.5f %9.5f %9.5f"
                        + " %10.4f %10.4f", k, a, b, b - a, x1, x2, f1, f2));
        }
        return new double[] {(a + b) / 2.0, k, evals};
    }
```

Kode 13.2 (Lanjutan) *Golden.java: pencarian yang sama dalam Java.*

```java
    public static void main(String[] args) {
        System.out.println("  k         a         b     b - a        x1        x2"
                + "      pi(x1)     pi(x2)");
        double[] r = goldenMax(Golden::profit, 2.0, 10.0, 1e-4, true);
        double p = r[0];
        System.out.println(f("golden section: p = %.5f after %d steps,"
                + " %d evaluations", p, (int) r[1], (int) r[2]));
        System.out.println(f("exact optimum : p = %.5f (from pi'(p) = 0)", C + S));
        System.out.println(f("cups %.1f, profit %.3f thousand rupiah a day",
                cups(p), profit(p)));
        double best = 2.0;
        for (int i = 0; i <= 160; i++) {           // every Rp 50
            double q = 2.0 + 0.05 * i;
            if (profit(q) > profit(best)) best = q;
        }
        System.out.println(f("grid of 161 prices: best p = %.2f, profit %.3f",
                best, profit(best)));
    }
}
```

13.4.2 Metode Newton dengan dan tanpa pengaman

Kode 13.3 mengodekan turunan pada Persamaan (13.8) dan menjalankan Newton biasa dua kali, dari Rp3.000 dan dari Rp8.000, lalu versi berpengaman pada kurung [2,9]. Di dalam `safeguarded`, tanda π' memperbarui kurung lebih dahulu; titik Newton diterima hanya bila kelengkungannya negatif dan titik itu berada tegas di dalam kurung. Bila kelengkungannya tidak negatif, ungkapan `a - 1.0` adalah titik mustahil yang sengaja dibuat untuk memaksa cabang biseksi.

Kode 13.3 `newton1d.py`: metode Newton untuk harga terbaik, biasa dan berpengaman.

```
"""Newton's method for the best tea price, with and without a safeguard.
Usage: python newton1d.py"""
import math

A, S, C = 600.0, 2.5, 1.5


def d1(p):
    """pi'(p): the slope of the profit curve."""
    return A * math.exp(-p / S) * (1.0 - (p - C) / S)


def d2(p):
    """pi''(p): the curvature of the profit curve."""
    return A * math.exp(-p / S) * ((p - C) / (S * S) - 2.0 / S)


def newton(p, steps):
    print(f"plain Newton from p0 = {p:.2f}")
    for k in range(1, steps + 1):
        p = p - d1(p) / d2(p)
        print(f"  {k:2d}  p = {p:12.8f}  error = {abs(p - (C + S)):.3e}")


def safeguarded(a, b, tol):
    """Newton on pi' = 0 inside [a, b]; bisect when Newton would leave the
    bracket or when the curvature says 'minimum' (pi'' >= 0)."""
    p, k = (a + b) / 2.0, 0
    while b - a > tol and k < 60:
        k += 1
        g, h = d1(p), d2(p)
        if g == 0.0:              # landed exactly on the peak
            break
        if g > 0.0:               # still climbing: the peak is to the right
            a = p
        else:
            b = p
        q = p - g / h if h < 0.0 else a - 1.0
        how = "newton"
        if not (a < q < b):       # unsafe step: halve the bracket instead
            q, how = (a + b) / 2.0, "bisect"
        if abs(q - p) < tol:
            p = q
            break
        p = q
        print(f"  {k:2d}  {how}  p = {p:12.8f}  bracket [{a:.5f}, {b:.5f}]")
```

Kode 13.3 (Lanjutan) *newton1d.py: metode Newton untuk harga terbaik, biasa dan berpengaman.*

```
    return p, k


def main():
    newton(3.0, 5)
    newton(8.0, 5)
    print("safeguarded Newton on [2, 9]")
    p, k = safeguarded(2.0, 9.0, 1e-10)
    print(f"p = {p:.10f} after {k} steps")


if __name__ == "__main__":
    main()
```

Kode 13.4 menerjemahkannya baris demi baris ke Java.

Kode 13.4 `Newton1D.java`: hal yang sama dalam Java.

```
// Newton's method for the best tea price, with and without a safeguard.
// Usage: java Newton1D
import java.util.Locale;

public class Newton1D {
    static final double A = 600.0, S = 2.5, C = 1.5;

    /** pi'(p): the slope of the profit curve. */
    static double d1(double p) {
        return A * Math.exp(-p / S) * (1.0 - (p - C) / S);
    }

    /** pi''(p): the curvature of the profit curve. */
    static double d2(double p) {
        return A * Math.exp(-p / S) * ((p - C) / (S * S) - 2.0 / S);
    }

    static void out(String fmt, Object... xs) {
        System.out.println(String.format(Locale.ROOT, fmt, xs));
    }

    static void newton(double p, int steps) {
        out("plain Newton from p0 = %.2f", p);
        for (int k = 1; k <= steps; k++) {
            p = p - d1(p) / d2(p);
            out("  %2d  p = %12.8f  error = %.3e", k, p, Math.abs(p - (C + S)));
        }
    }

    /** Newton on pi' = 0 inside [a, b]; bisect when Newton would leave the
     *  bracket or when the curvature says 'minimum' (pi'' >= 0). */
    static double[] safeguarded(double a, double b, double tol) {
        double p = (a + b) / 2.0;
        int k = 0;
        while (b - a > tol && k < 60) {
            k++;
            double g = d1(p), h = d2(p);
            if (g == 0.0) break;              // landed exactly on the peak
            if (g > 0.0) a = p; else b = p;   // climbing: the peak is to the right
            double q = h < 0.0 ? p - g / h : a - 1.0;
            String how = "newton";
            if (!(a < q && q < b)) {          // unsafe step: halve the bracket
                q = (a + b) / 2.0;
                how = "bisect";
            }
            if (Math.abs(q - p) < tol) { p = q; break; }
```

Kode 13.4 (Lanjutan) *Newton1D.java: hal yang sama dalam Java.*

```
            p = q;
            out("  %2d  %s  p = %12.8f  bracket [%.5f, %.5f]", k, how, p, a, b);
        }
        return new double[] {p, k};
    }

    public static void main(String[] args) {
        newton(3.0, 5);
        newton(8.0, 5);
        System.out.println("safeguarded Newton on [2, 9]");
        double[] r = safeguarded(2.0, 9.0, 1e-10);
        out("p = %.10f after %d steps", r[0], (int) r[1]);
    }
}
```

13.4.3 Dua harga: penurunan dan Newton

Kode 13.5 adalah jantung bab ini. `demand`, `f`, `grad`, dan `hess` mengodekan Persamaan (13.12) beserta turunannya; `descent` adalah Persamaan (13.13) dengan langkah tetap dan penjaga yang berhenti bila bilangannya meledak; `backtracking` menambahkan uji Armijo Persamaan (13.16); `newton` menyelesaikan sistem 2×2 pada Persamaan (13.17) dengan aturan Cramer. Di akhir, program menghitung nilai eigen Hessian di titik optimum dari *trace* dan determinannya (untuk matriks simetris 2×2 nilainya $\frac{t}{2} \pm \sqrt{t^2/4 - d}$), bilangan kondisi, dan batas langkah pada Persamaan (13.14) dan Persamaan (13.15).

Kode 13.5 `twoprices.py`: *gradient descent* dengan langkah tetap dan *backtracking*, serta metode Newton, untuk dua harga.

```
"""Two prices at once: tea (p1) and fried snacks (p2).
Gradient descent with a fixed step, with backtracking, and Newton's method.
Usage: python twoprices.py"""
import math

C1, C2 = 1.5, 0.6          # cost of a cup of tea and of a snack (thousand Rp)


def demand(p1, p2):
    """Cups of tea and snacks sold in a day at prices p1 and p2."""
    tea = 600.0 * math.exp(-p1 / 2.5 - 0.10 * p2)
    snack = 900.0 * math.exp(-p2 / 1.0 - 0.08 * p1)
    return tea, snack


def f(p):
    """The loss to minimize: minus the daily profit."""
    tea, snack = demand(p[0], p[1])
    return -((p[0] - C1) * tea + (p[1] - C2) * snack)


def grad(p):
    tea, snack = demand(p[0], p[1])
    m1, m2 = p[0] - C1, p[1] - C2
    g1 = tea - m1 * tea / 2.5 - 0.08 * m2 * snack
    g2 = snack - 0.10 * m1 * tea - m2 * snack
    return [-g1, -g2]


def hess(p):
    tea, snack = demand(p[0], p[1])
    m1, m2 = p[0] - C1, p[1] - C2
    h11 = -2.0 * tea / 2.5 + m1 * tea / 6.25 + 0.0064 * m2 * snack
    h12 = -0.10 * tea + 0.04 * m1 * tea - 0.08 * snack + 0.08 * m2 * snack
    h22 = -2.0 * snack + 0.01 * m1 * tea + m2 * snack
    return [[-h11, -h12], [-h12, -h22]]


def norm(v):
    return math.sqrt(v[0] * v[0] + v[1] * v[1])
```

Kode 13.5 (Lanjutan) *twoprices.py: gradient descent dengan langkah tetap dan backtracking, serta metode Newton, untuk dua harga.*

```
def descent(p, alpha, tol=1e-6, kmax=2000):
    """Fixed-step gradient descent; returns (point, steps) or None."""
    for k in range(1, kmax + 1):
        g = grad(p)
        p = [p[0] - alpha * g[0], p[1] - alpha * g[1]]
        if not all(math.isfinite(x) for x in p) or norm(p) > 1e6:
            return None, k
        if norm(g) < tol:
            return p, k
    return p, kmax


def backtracking(p, tol=1e-6, kmax=2000):
    """Gradient descent where each step is halved until f drops enough."""
    tries = 0
    for k in range(1, kmax + 1):
        g = grad(p)
        gg = g[0] * g[0] + g[1] * g[1]
        if math.sqrt(gg) < tol:
            return p, k - 1, tries
        alpha, fp = 1.0, f(p)
        while True:                          # Armijo condition, c = 1e-4
            tries += 1
            q = [p[0] - alpha * g[0], p[1] - alpha * g[1]]
            if f(q) <= fp - 1e-4 * alpha * gg:
                break
            alpha /= 2.0
        p = q
    return p, kmax, tries


def newton(p, tol=1e-10, kmax=50, show=True):
    for k in range(1, kmax + 1):
        g, h = grad(p), hess(p)
        det = h[0][0] * h[1][1] - h[0][1] * h[1][0]
        s1 = (h[1][1] * g[0] - h[0][1] * g[1]) / det    # solve H s = g
        s2 = (h[0][0] * g[1] - h[1][0] * g[0]) / det
        p = [p[0] - s1, p[1] - s2]
        if show:
            print(f"  {k:2d}  p1 = {p[0]:.10f}  p2 = {p[1]:.10f}"
                  f"  |grad| = {norm(grad(p)):.3e}")
        if norm(grad(p)) < tol:
            return p, k
    return p, kmax
```

Kode 13.5 (Lanjutan) *twoprices.py: gradient descent dengan langkah tetap dan backtracking, serta metode Newton, untuk dua harga.*

```
def main():
    p0 = [3.0, 1.0]
    g, h = 1e-5, grad(p0)
    num = [(f([3.0 + g, 1.0]) - f([3.0 - g, 1.0])) / (2 * g),
           (f([3.0, 1.0 + g]) - f([3.0, 1.0 - g])) / (2 * g)]
    print(f"gradient at (3, 1), formula           ({h[0]:.6f}, {h[1]:.6f})")
    print(f"gradient at (3, 1), central difference"
          f" ({num[0]:.6f}, {num[1]:.6f})")
    for alpha in (0.002, 0.008, 0.012, 0.020):
        p, k = descent(p0, alpha)
        if p is None:
            print(f"fixed step {alpha:.3f}: diverged after {k} steps")
        elif k == 2000:
            print(f"fixed step {alpha:.3f}: no convergence in {k} steps,"
                  f" |grad| = {norm(grad(p)):.3f}")
        else:
            print(f"fixed step {alpha:.3f}: p = ({p[0]:.6f}, {p[1]:.6f})"
                  f" after {k} steps")
    p, k, tries = backtracking(p0)
    print(f"backtracking  : p = ({p[0]:.6f}, {p[1]:.6f}) after {k} steps,"
          f" {tries} trial points")
    print("Newton from (3, 1):")
    p, k = newton(p0)
    tea, snack = demand(p[0], p[1])
    print(f"best prices: tea Rp {1000 * p[0]:.0f}, snack Rp {1000 * p[1]:.0f}")
    print(f"sales: {tea:.1f} cups, {snack:.1f} snacks;"
          f" profit {-f(p):.3f} thousand rupiah a day")
    h = hess(p)
    tr, det = h[0][0] + h[1][1], h[0][0] * h[1][1] - h[0][1] * h[1][0]
    lo = tr / 2 - math.sqrt(tr * tr / 4 - det)
    hi = tr / 2 + math.sqrt(tr * tr / 4 - det)
    print(f"Hessian eigenvalues {lo:.4f} and {hi:.4f}:"
          f" condition number {hi / lo:.3f}")
    print(f"step limit 2/lambda_max = {2 / hi:.5f},"
          f" best fixed step 2/(lo + hi) = {2 / (lo + hi):.5f}")


if __name__ == "__main__":
    main()
```

Kode 13.6 adalah versi Java-nya. Java tidak mempunyai *tuple*, sehingga `descent` dan `backtracking` mengembalikan larik `double` kecil yang entri terakhirnya membawa banyaknya langkah dan sebuah status.

Kode 13.6 `TwoPrices.java`: pengoptimal yang sama dalam Java.

```java
// Two prices at once: tea (p1) and fried snacks (p2).
// Gradient descent with a fixed step, with backtracking, and Newton's method.
// Usage: java TwoPrices
import java.util.Locale;

public class TwoPrices {
    static final double C1 = 1.5, C2 = 0.6;   // cost of a tea and of a snack

    static void out(String fmt, Object... xs) {
        System.out.println(String.format(Locale.ROOT, fmt, xs));
    }

    /** Cups of tea and snacks sold in a day at prices p1 and p2. */
    static double[] demand(double p1, double p2) {
        return new double[] {600.0 * Math.exp(-p1 / 2.5 - 0.10 * p2),
                             900.0 * Math.exp(-p2 / 1.0 - 0.08 * p1)};
    }

    /** The loss to minimize: minus the daily profit. */
    static double f(double[] p) {
        double[] d = demand(p[0], p[1]);
        return -((p[0] - C1) * d[0] + (p[1] - C2) * d[1]);
    }

    static double[] grad(double[] p) {
        double[] d = demand(p[0], p[1]);
        double tea = d[0], snack = d[1], m1 = p[0] - C1, m2 = p[1] - C2;
        double g1 = tea - m1 * tea / 2.5 - 0.08 * m2 * snack;
        double g2 = snack - 0.10 * m1 * tea - m2 * snack;
        return new double[] {-g1, -g2};
    }

    static double[][] hess(double[] p) {
        double[] d = demand(p[0], p[1]);
        double tea = d[0], snack = d[1], m1 = p[0] - C1, m2 = p[1] - C2;
        double h11 = -2.0 * tea / 2.5 + m1 * tea / 6.25 + 0.0064 * m2 * snack;
        double h12 = -0.10 * tea + 0.04 * m1 * tea - 0.08 * snack
                + 0.08 * m2 * snack;
        double h22 = -2.0 * snack + 0.01 * m1 * tea + m2 * snack;
        return new double[][] {{-h11, -h12}, {-h12, -h22}};
    }

    static double norm(double[] v) { return Math.sqrt(v[0] * v[0] + v[1] * v[1]); }
```

Kode 13.6 (Lanjutan) *TwoPrices.java: pengoptimal yang sama dalam Java.*

```
    /** Fixed-step gradient descent; returns {p1, p2, steps, status}. */
    static double[] descent(double[] p, double alpha, double tol, int kmax) {
        for (int k = 1; k <= kmax; k++) {
            double[] g = grad(p);
            p = new double[] {p[0] - alpha * g[0], p[1] - alpha * g[1]};
            if (!Double.isFinite(p[0]) || !Double.isFinite(p[1])
                    || norm(p) > 1e6)
                return new double[] {p[0], p[1], k, -1};
            if (norm(g) < tol) return new double[] {p[0], p[1], k, 1};
        }
        return new double[] {p[0], p[1], kmax, 0};
    }

    /** Gradient descent where each step is halved until f drops enough. */
    static double[] backtracking(double[] p, double tol, int kmax) {
        int tries = 0;
        for (int k = 1; k <= kmax; k++) {
            double[] g = grad(p);
            double gg = g[0] * g[0] + g[1] * g[1];
            if (Math.sqrt(gg) < tol)
                return new double[] {p[0], p[1], k - 1, tries};
            double alpha = 1.0, fp = f(p);
            double[] q;
            while (true) {                          // Armijo condition, c = 1e-4
                tries++;
                q = new double[] {p[0] - alpha * g[0], p[1] - alpha * g[1]};
                if (f(q) <= fp - 1e-4 * alpha * gg) break;
                alpha /= 2.0;
            }
            p = q;
        }
        return new double[] {p[0], p[1], kmax, tries};
    }

    static double[] newton(double[] p, double tol, int kmax) {
        for (int k = 1; k <= kmax; k++) {
            double[] g = grad(p);
            double[][] h = hess(p);
            double det = h[0][0] * h[1][1] - h[0][1] * h[1][0];
            double s1 = (h[1][1] * g[0] - h[0][1] * g[1]) / det;   // solve H s = g
            double s2 = (h[0][0] * g[1] - h[1][0] * g[0]) / det;
            p = new double[] {p[0] - s1, p[1] - s2};
            out("  %2d  p1 = %.10f  p2 = %.10f  |grad| = %.3e",
                    k, p[0], p[1], norm(grad(p)));
            if (norm(grad(p)) < tol) return p;
        }
```

Kode 13.6 (Lanjutan) *TwoPrices.java: pengoptimal yang sama dalam Java.*

```
        return p;
    }

    public static void main(String[] args) {
        double[] p0 = {3.0, 1.0};
        double e = 1e-5;
        double[] h = grad(p0);
        double n1 = (f(new double[] {3.0 + e, 1.0})
                - f(new double[] {3.0 - e, 1.0})) / (2 * e);
        double n2 = (f(new double[] {3.0, 1.0 + e})
                - f(new double[] {3.0, 1.0 - e})) / (2 * e);
        out("gradient at (3, 1), formula           (%.6f, %.6f)", h[0], h[1]);
        out("gradient at (3, 1), central difference (%.6f, %.6f)", n1, n2);
        for (double alpha : new double[] {0.002, 0.008, 0.012, 0.020}) {
            double[] r = descent(p0, alpha, 1e-6, 2000);
            if (r[3] < 0)
                out("fixed step %.3f: diverged after %d steps", alpha, (int) r[2]);
            else if (r[2] == 2000)
                out("fixed step %.3f: no convergence in %d steps, |grad| = %.3f",
                        alpha, (int) r[2], norm(grad(new double[] {r[0], r[1]})));
            else
                out("fixed step %.3f: p = (%.6f, %.6f) after %d steps",
                        alpha, r[0], r[1], (int) r[2]);
        }
        double[] b = backtracking(p0, 1e-6, 2000);
        out("backtracking  : p = (%.6f, %.6f) after %d steps, %d trial points",
                b[0], b[1], (int) b[2], (int) b[3]);
        System.out.println("Newton from (3, 1):");
        double[] p = newton(p0, 1e-10, 50);
        double[] d = demand(p[0], p[1]);
        out("best prices: tea Rp %.0f, snack Rp %.0f", 1000 * p[0], 1000 * p[1]);
        out("sales: %.1f cups, %.1f snacks; profit %.3f thousand rupiah a day",
                d[0], d[1], -f(p));
        double[][] hs = hess(p);
        double tr = hs[0][0] + hs[1][1];
        double det = hs[0][0] * hs[1][1] - hs[0][1] * hs[1][0];
        double lo = tr / 2 - Math.sqrt(tr * tr / 4 - det);
        double hi = tr / 2 + Math.sqrt(tr * tr / 4 - det);
        out("Hessian eigenvalues %.4f and %.4f: condition number %.3f",
                lo, hi, hi / lo);
        out("step limit 2/lambda_max = %.5f, best fixed step 2/(lo + hi) = %.5f",
                2 / hi, 2 / (lo + hi));
    }
}
```

13.5 Menjalankan Program

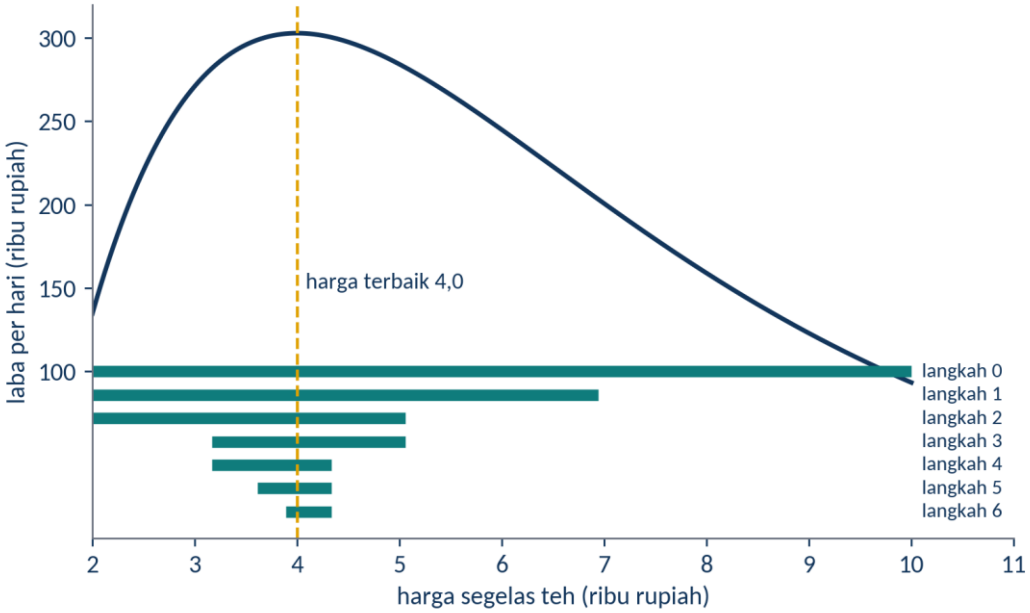
Kompilasi dan jalankan dari folder bab (di Windows ketik `python`, bukan `python3`):

```
cd Chapter13/python
python3 golden.py
python3 newton1d.py
python3 twoprices.py
cd ../java
javac -d out *.java
java -cp out Golden
java -cp out Newton1D
java -cp out TwoPrices
```

Golden-section search mencetak:

```
  k         a         b     b - a        x1        x2     pi(x1)     pi(x2)
  1   2.00000   6.94427   4.94427   3.88854   5.05573   302.5347   282.3644
  2   2.00000   5.05573   3.05573   3.16718   3.88854   281.7984   302.5347
  3   3.16718   5.05573   1.88854   3.88854   4.33437   302.5347   300.3659
  4   3.16718   4.33437   1.16718   3.61301   3.88854   298.8193   302.5347
  5   3.61301   4.33437   0.72136   3.88854   4.05883   302.5347   302.7622
  6   3.88854   4.33437   0.44582   4.05883   4.16408   302.7622   302.2204
 24   3.99998   4.00006   0.00008   4.00001   4.00003   302.8448   302.8448
golden section: p = 4.00002 after 24 steps, 26 evaluations
exact optimum : p = 4.00000 (from pi'(p) = 0)
cups 121.1, profit 302.845 thousand rupiah a day
grid of 161 prices: best p = 4.00, profit 302.845
```

Gambar 13.1 menggambar kurva laba dan tujuh selang pertama. Selang menyusut dengan faktor yang sama di setiap langkah, sehingga panjangnya turun dari 8 menjadi $8 \times 0.618^{24} = 7.7 \times 10^{-5}$ setelah 24 langkah, di bawah toleransi 10^{-4}. Jawabannya, Rp4.000, menghasilkan 302,8 ribu rupiah sehari dengan 121 gelas: sekitar Rp32.000 sehari lebih banyak daripada harganya sekarang dan Rp19.000 lebih banyak daripada harga tetangganya. Grid 161 harga menemukan jawaban yang sama sampai Rp50 terdekat tetapi memerlukan enam kali lebih banyak evaluasi, dan untuk mencapai ketelitian *golden section* grid itu memerlukan 80.000 evaluasi.



Gambar 13.1 Laba harian Mbak Rina dan selang *golden section* pada langkah 0 sampai 6: setiap batang panjangnya 0,618 kali batang di atasnya, dan semuanya memuat harga terbaik.

Perhatikan dalam keluaran bagaimana pada langkah 5 selang [3.61,4.33] masih memuat puncak walaupun nilai titik ujinya, 302.53 dan 302.76, berbeda kurang dari sepersepuluh persen. Di dekat puncak kurvanya datar, persis seperti peringatan Persamaan (13.6): di akhir, kedua titik uji menunjukkan 302,8448 sampai empat desimal. Metode ini memutuskan dari digit-digit yang tersisa, dan itulah sebabnya toleransi jauh di bawah 10^{-8} tidak bermakna.

Metode Newton mencetak:

```
plain Newton from p0 = 3.00
   1  p =   3.71428571  error = 2.857e-01
   2  p =   3.97069597  error = 2.930e-02
   3  p =   3.99966049  error = 3.395e-04
   4  p =   3.99999995  error = 4.610e-08
   5  p =   4.00000000  error = 8.882e-16
plain Newton from p0 = 8.00
   1  p =  14.66666667  error = 1.067e+01
   2  p =  17.93197279  error = 1.393e+01
   3  p =  20.97868508  error = 1.698e+01
   4  p =  23.91035411  error = 1.991e+01
   5  p =  26.76933590  error = 2.277e+01
safeguarded Newton on [2, 9]
   1  bisect  p =   3.75000000  bracket [2.00000, 5.50000]
   2  newton  p =   3.97727273  bracket [3.75000, 5.50000]
   3  newton  p =   3.99979525  bracket [3.97727, 5.50000]
   4  newton  p =   3.99999998  bracket [3.99980, 5.50000]
   5  newton  p =   4.00000000  bracket [4.00000, 5.50000]
p = 4.0000000000 after 6 steps
```

Jalannya yang pertama adalah Persamaan (13.9) dalam aksi: galat 2.9×10^{-1}, 2.9×10^{-2}, 3.4×10^{-4}, 4.6×10^{-8} masing-masing kira-kira kuadrat galat sebelumnya, dan langkah kelima mencapai tingkat pembulatan. Jalan kedua dimulai di luar titik belok 6,5 dan pergi menjauh selamanya, setiap langkah menambah sekitar tiga ribu rupiah pada harga yang tidak lagi membuat siapa pun membeli teh. Jalan berpengaman dimulai dari titik tengah 5,5, tempat kelengkungannya negatif tetapi titik Newton akan jatuh di luar kurung, sehingga ia membagi dua sekali ke 3,75; dari sana setiap langkah Newton diterima dan jawabannya ditemukan dalam enam langkah.

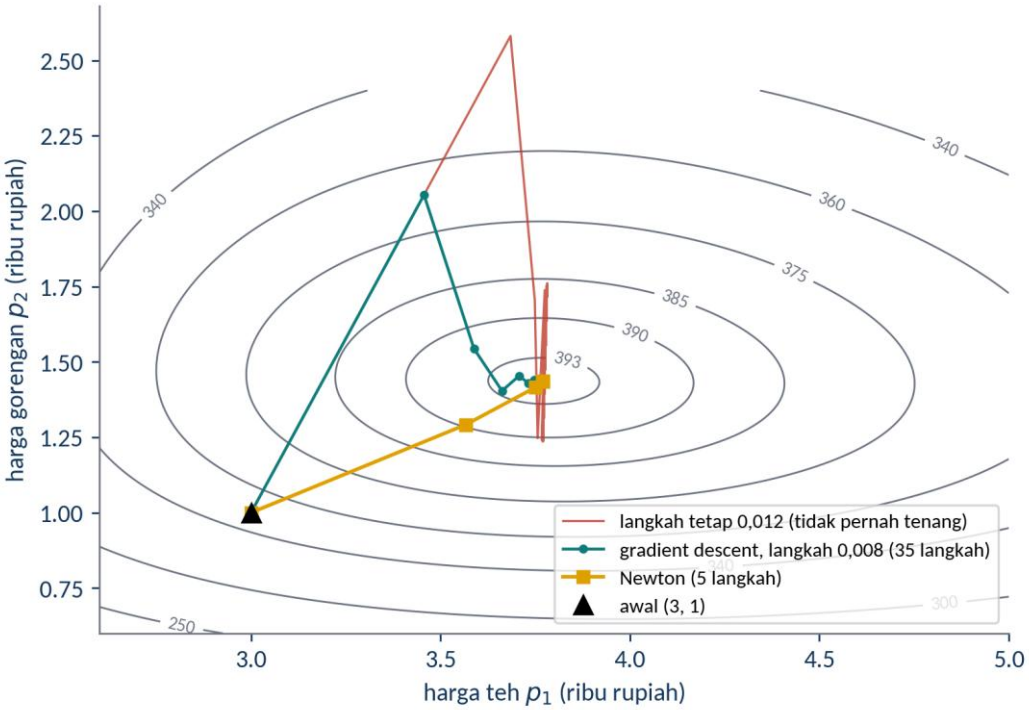
Program dua harga mencetak:

```
gradient at (3, 1), formula           (-57.073365, -131.739616)
gradient at (3, 1), central difference (-57.073365, -131.739616)
fixed step 0.002: p = (3.769928, 1.435197) after 166 steps
fixed step 0.008: p = (3.769928, 1.435197) after 35 steps
fixed step 0.012: no convergence in 2000 steps, |grad| = 43.533
fixed step 0.020: no convergence in 2000 steps, |grad| = 212.849
backtracking  : p = (3.769928, 1.435197) after 35 steps, 294 trial points
Newton from (3, 1):
   1  p1 = 3.5666834402  p2 = 1.2919630099  |grad| = 3.321e+01
   2  p1 = 3.7512263928  p2 = 1.4160375865  |grad| = 3.750e+00
   3  p1 = 3.7697253414  p2 = 1.4348269754  |grad| = 6.884e-02
   4  p1 = 3.7699275219  p2 = 1.4351964841  |grad| = 2.473e-05
   5  p1 = 3.7699275577  p2 = 1.4351966188  |grad| = 3.188e-12
best prices: tea Rp 3770, snack Rp 1435
sales: 115.1 cups, 158.5 snacks; profit 393.533 thousand rupiah a day
Hessian eigenvalues 49.3370 and 182.0565: condition number 3.690
step limit 2/lambda_max = 0.01099, best fixed step 2/(lo + hi) = 0.00864
```

Pemeriksaan gradiennya cocok; keempat langkah tetap berperilaku seperti yang diramalkan Persamaan (13.14) dan Persamaan (13.15): 166 langkah untuk langkah takut-takut 0,002, 35 untuk 0,008, dan tidak konvergen untuk 0,012 dan 0,020, keduanya di atas batas 0,01099. *Backtracking* konvergen dalam 35 langkah tanpa diberi tahu batas apa pun. Newton mencapai jawaban dalam lima langkah, dan panjang gradiennya, 33, 3,75, 0,069, 2.5×10^{-5}, 3.2×10^{-12}, menunjukkan konvergensi kuadratik dalam dua variabel.

Gambar 13.2 memperlihatkan lintasan-lintasan itu pada peta laba. Lintasan Newton memotong garis kontur hampir lurus ke puncak. *Gradient descent* dengan langkah 0,008 mula-mula melampaui sasaran ke arah yang curam, yaitu harga gorengan, lalu menyusuri lembah dengan beberapa zig-zag. Dengan langkah 0,012 iterasinya memantul naik-turun melintasi lembah ke

arah yang curam selamanya: analisis linear meramalkan osilasi yang membesar, dan ketaklinearan kurva mengubahnya menjadi siklus yang tidak membesar dan tidak pula padam.



Gambar 13.2 Kontur laba harian (ribu rupiah) terhadap kedua harga, beserta lintasan *gradient descent* (langkah 0,008), langkah tetap di atas batas kestabilan (0,012), dan metode Newton, semuanya mulai dari (3, 1).

Jawaban untuk Mbak Rina: teh Rp3.770 dan gorengan Rp1.435, dengan penjualan 115 gelas dan 158 gorengan, laba 393,5 ribu rupiah sehari. Dalam praktik ia akan membulatkannya menjadi Rp3.800 dan Rp1.500, dan datarnya puncak memberitahunya bahwa pembulatan seperti itu hampir tidak ada ruginya: Persamaan (13.11) dengan nilai eigen Hessian menunjukkan bahwa menggeser harga sebesar Rp50 mengubah laba jauh di bawah seribu rupiah sehari.

13.6 Membandingkan dengan SciPy

Modul `scipy.optimize` dari SciPy [4] menyediakan keluarga metode yang sama. **Kode 13.7** memanggilnya pada fungsi yang sama, yang diimpor dari berkas kita sendiri, sehingga hanya pengoptimalnya yang berbeda. Program ini dijalankan dengan Python 3.13, NumPy 2.5.3, dan SciPy 1.18.1, rilis terbaru pada September 2026 [8], [9]; `requirements.txt` di folder bab mencantumkannya.

Kode 13.7 `check_scipy.py`: masalah yang sama diselesaikan dengan pengoptimal SciPy.

```
"""Compare the chapter's optimizers with SciPy's.
Usage: python check_scipy.py   (needs numpy and scipy)"""
import numpy as np
from scipy.optimize import minimize, minimize_scalar
from golden import profit
from twoprices import f, grad, hess

r = minimize_scalar(lambda p: -profit(p), bracket=(2.0, 3.0, 10.0),
                    method="golden", tol=1e-8)
print(f"SciPy golden : p = {r.x:.8f}, {r.nfev} evaluations")
r = minimize_scalar(lambda p: -profit(p), bounds=(2.0, 10.0),
                    method="bounded", options={"xatol": 1e-8})
print(f"SciPy Brent  : p = {r.x:.8f}, {r.nfev} evaluations")
x0 = np.array([3.0, 1.0])
for m in ("BFGS", "Newton-CG", "Nelder-Mead"):
    kw = {"jac": grad} if m == "BFGS" else {}
    if m == "Newton-CG":
        kw = {"jac": grad, "hess": hess}
    r = minimize(f, x0, method=m, **kw)
    print(f"SciPy {m:11s}: p = ({r.x[0]:.6f}, {r.x[1]:.6f}),"
          f" {r.nit} iterations, {r.nfev} evaluations")
```

```
SciPy golden : p = 3.99999998, 45 evaluations
SciPy Brent  : p = 4.00000000, 11 evaluations
SciPy BFGS       : p = (3.769928, 1.435197), 9 iterations, 11 evaluations
SciPy Newton-CG  : p = (3.769928, 1.435197), 5 iterations, 6 evaluations
SciPy Nelder-Mead: p = (3.769919, 1.435177), 35 iterations, 67 evaluations
```

Golden-section search SciPy sesuai dengan milik kita; ia memakai lebih banyak evaluasi karena kita memintanya bertoleransi 10^{-8}, bukan 10^{-4}. Metode `bounded`, yaitu gabungan *golden section* dan interpolasi parabola dari Brent [5], hanya memerlukan 11 evaluasi: begitu puncak terkurung, parabola melalui tiga titik mendarat sangat dekat dengannya, mirip langkah Newton, tanpa memerlukan turunan. BFGS, yang mempelajari Hessian dari gradien, memerlukan 9 iterasi; Newton-CG, yang diberi Hessian kita, memerlukan 5, sama dengan Newton kita. Nelder–Mead, metode tanpa turunan yang menggerakkan segitiga tiga titik menuruni lereng [10], memerlukan 67 evaluasi dan hanya teliti sampai sekitar empat desimal dengan toleransi bawaannya. Pelajarannya berlaku juga untuk Java, yang tidak mempunyai pengoptimal di pustaka standarnya: *golden section* yang pendek dan teruji, Newton berpengaman, dan metode gradien dengan *backtracking* sudah mencakup sangat banyak masalah praktis.

13.7 Membuatnya Lebih Baik

Tiga perbaikan mengubah program-program ini menjadi alat yang berperilaku baik pada masalah yang belum pernah Anda lihat.

- **Kurung dulu, baru cari.** *Golden section* memerlukan selang yang memuat puncak. Bila belum diketahui, mulailah dari tebakan yang masuk akal, melangkah ke kanan, dan terus gandakan langkah selama fungsinya masih naik; tiga titik terakhir mengurung puncak. Fungsi `bracket` di SciPy melakukan persis hal ini.
- **Skalakan variabelnya.** Bilangan kondisi pada Persamaan (13.15) bergantung pada satuan. Seandainya Mbak Rina menulis harganya dalam rupiah alih-alih ribuan, dan harga gorengan dalam ratusan, entri-entri Hessian akan berbeda dengan faktor ratusan dan κ ikut membesar. Memilih satuan yang membuat semua variabel berukuran serupa itu gratis dan sering mempercepat *gradient descent* berlipat-lipat; Bab 14 melakukannya secara sistematis untuk *machine learning*.

- **Hormati batas masalah yang nyata.** Harga tidak boleh di bawah biaya dan tidak boleh melampaui apa yang sanggup ditanggung pasar. Cara paling sederhana untuk menjaga iterasi tetap di dalam kotak $l \le x \le u$ adalah *proyeksi*: setelah setiap langkah, jepit setiap variabel kembali ke dalam rentangnya. Untuk *golden section* kotaknya adalah kurung awal; untuk *gradient descent*, jepit setelah setiap langkah. Masalah dengan kendala yang lebih rumit, misalnya anggaran terbatas yang dibagi di antara produk, memerlukan metode optimasi berkendala, pengali Lagrange dan turunan-turunannya [6], [11].

13.8 Memakainya Sungguhan

Metode Mbak Rina dapat dipakai oleh penjual mana pun yang dapat mengubah harga dan menghitung apa yang terjual. Berikut cara memakainya untuk lapak, toko, atau jasa Anda sendiri.

- **Catat harga selama beberapa minggu.** Cobalah tiga sampai lima harga, masing-masing pada beberapa hari yang sebanding (hari yang sama dalam seminggu, bukan hari libur, cuaca serupa), dan catat banyaknya yang terjual. Mengubah harga hanya pada hari hujan, atau hanya di akhir bulan, mencampur pengaruh harga dengan pengaruh hari, dan kurva yang Anda cocokkan akan keliru. Bab 8 mencocokkan kurva eksponensial Persamaan (13.1) sebagai garis lurus: $\ln D = \ln A - p/s$, sehingga garis kuadrat terkecil melalui titik-titik $(p, \ln D)$ memberi s dari kemiringannya.
- **Hitung hanya biaya yang tumbuh bersama penjualan.** Dalam Persamaan (13.2), c adalah biaya satu unit tambahan: bahan, kemasan, komisi aplikasi pesan-antar. Sewa tempat, izin lapak, dan upah Anda sendiri untuk hari itu sama berapa pun yang terjual; mengurangkannya menurunkan seluruh kurva laba tetapi tidak menggeser puncaknya, sehingga tidak mengubah harga terbaik. Biaya-biaya itu menentukan apakah usahanya layak dijalankan sama sekali.
- **Lihat kurvanya sebelum mencari.** Gambarlah laba hasil pencocokan pada rentang harga yang mungkin Anda pertimbangkan, atau evaluasi pada grid kasar. Bila hanya ada satu puncak, kurunglah dan jalankan *golden section* dari **Pseudokode 13.1**; bila ada dua (bisa terjadi, bila permintaan anjlok pada harga "psikologis" seperti Rp5.000), carilah di setiap kurung secara terpisah lalu bandingkan.
- **Bulatkan, lalu uji.** Bulatkan optimumnya ke harga yang mudah dibayar tunai. Puncak yang datar berarti pembulatan hanya sedikit ruginya, dan selisih beberapa ratus rupiah sepadan dengan kemudahannya. Lalu pakailah harga baru itu selama dua minggu, terus mencatat, dan cocokkan ulang: orang bereaksi terhadap harga baru untuk sementara waktu sebelum tenang, dan hari-hari pertama bukan kata terakhir.

Contoh kedua memperlihatkan betapa sedikit yang berubah untuk usaha yang lain. Bu Wati menjalankan *laundry kiloan*. Deterjen, air, listrik, dan kantong plastik menghabiskan sekitar Rp3.500 per kilogram. Catatannya selama enam minggu, dengan harga dari Rp6.000 sampai Rp9.000, cocok dengan kurva eksponensial berskala harga $s = 4.0$ ribu rupiah: setiap tambahan Rp4.000 per kilogram akan kehilangan sekitar 63% pelanggannya. Analisis Subbab 13.3.2 langsung memberi harga terbaik tanpa pencarian apa pun: $p^* = c + s = 3.5 + 4.0 = 7.5$ ribu, Rp7.500 per kilogram. Bila ia juga menawarkan layanan kilat, yang pelanggannya kurang peka terhadap harga (s lebih besar) tetapi memakan lebih banyak listrik per kilogram (c lebih besar), kedua layanan itu membentuk masalah dua harga seperti teh dan gorengan Mbak Rina, dan program-program di Subbab 13.4.3 dapat dipakai dengan bilangan-bilangannya di dalam `demand`.

Model memberi saran; pemilik usaha yang memutuskan. Model tidak tahu apa-apa tentang pesaing baru di seberang jalan, pelanggan setia yang akan tersinggung oleh kenaikan harga, atau kesan baik dari angka yang bulat. Itu semua pengetahuan si pemilik, dan pemakaian komputasi

yang tepat adalah memperlihatkan kepadanya berapa kemungkinan hasil setiap harga, agar ia dapat menimbang sisanya sendiri.

13.9 Metode Numerik dalam Praktik: Profitina

Optimasi muncul di platform AI Visibility Profitina di mana pun sebuah pengaturan menukar satu hal baik dengan hal baik lainnya. Contoh yang paling jelas adalah pemindaian ulang otomatis. Platform ini memindai ulang situs web setiap merek dengan jeda yang bergantung pada tingkat langganannya dan menyimpan tren skor mingguan untuk dasbor historis [12]. Memindai lebih sering memberi skor yang lebih segar dan peringatan yang lebih cepat; tetapi juga memakan waktu server dan membebani situs web pelanggan. Sebagai fungsi jeda antarpemindaian, total "biaya" (keusangan ditambah beban) berbentuk seperti laba Mbak Rina yang dibalik: jeda yang terlalu pendek membuang kerja, jeda yang terlalu panjang menyajikan hasil yang usang, dan di antaranya terdapat minimum. Pencarian satu variabel sejenis *golden section* adalah alat yang wajar untuk memilih pengaturan seperti itu dari sebuah model biaya, dan datarnya minimum adalah kabar baik, karena bilangan bulat di dekatnya (harian, mingguan) hampir tidak merugikan apa pun.

Gerbang mutu Content Engine adalah optimasi dari jenis yang lain. Draf yang ditulis model lokal Qwen2.5 dinilai terhadap templat bawaan, dan yang lebih baik di antara keduanya yang disajikan; setelah revisi September 2026 draf AI menang dalam 5 dari 6 kasus uji tolok ukur [12]. Memilih yang terbaik dari sekumpulan kecil kandidat berdasarkan skor adalah optimasi atas himpunan diskret, tempat gradien tidak ada dan metodenya cukup mengevaluasi lalu membandingkan. Di balik koin utilitas PROFIT ada kurva emisi yang meluruh disertai pembakaran; menyetel parameter kurva seperti itu agar pasokan dan permintaan tetap dalam rentang yang wajar pada akhirnya adalah minimasi sebuah fungsi rugi atas beberapa parameter kontinu, yaitu masalah pada Subbab 13.3.4.

13.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Memaksimumkan padahal kodenya meminimumkan.** Kebanyakan pustaka, termasuk SciPy, meminimumkan. Berikan $-\pi$, bukan π, dan jangan lupa membalik tanda nilai yang dilaporkan.
- **Golden section pada fungsi dengan beberapa puncak.** Metode ini menjaga *sebuah* puncak di dalam kurung, bukan yang tertinggi. Gambarlah fungsinya atau evaluasi pada grid kasar lebih dahulu, lalu kurung puncak yang Anda inginkan.
- **Meminta ketelitian berlebihan.** Di dekat optimum fungsinya datar, dan Persamaan (13.6) membatasi ketelitian letak yang dapat dicapai sampai sekitar $\sqrt{\epsilon_{\text{mach}}}$ relatif. Toleransi 10^{-12} membuat metode membandingkan derau pembulatan.
- **Memercayai Newton jauh dari jawaban.** Langkah Newton menuju titik stasioner parabola lokal, yang bisa berupa minimum padahal Anda menginginkan maksimum, atau jauh di luar titik belok. Periksa tanda kelengkungannya dan pertahankan kurung.
- **Besar langkah yang disalin dari masalah lain.** Langkah yang aman bergantung pada kelengkungan terbesar, Persamaan (13.14); langkah yang berhasil untuk satu data dapat divergen pada data lain. Pakailah *backtracking*, atau perkirakan $\lambda_{\max}$ dengan metode pangkat dari Bab 12.
- **Gradien yang tidak diperiksa.** Satu tanda yang salah dalam satu turunan parsial membuat *gradient descent* mendaki dan Newton berputar-putar. Bandingkan dengan selisih pusat di satu atau dua titik sebelum menjalankan iterasi panjang.

- **Berhenti karena langkah kecil, bukan karena gradien kecil.** Dengan α yang sangat kecil, langkahnya kecil walaupun masih jauh dari jawaban. Berhentilah berdasarkan panjang gradien (relatif terhadap panjang awalnya) atau perubahan fungsi rugi, dan selalu batasi banyaknya langkah.

13.11 Ringkasan Bab

- Masalah optimasi meminta titik tempat fungsi objektif terkecil atau terbesar; memaksimumkan f sama dengan meminimumkan $-f$. Di optimum interior yang mulus, turunan atau gradien bernilai nol, dan turunan kedua atau Hessian membedakan minimum dari maksimum.
- *Golden-section search* mengurung puncak fungsi unimodal dan memperpendek kurung dengan faktor $r = (\sqrt{5}-1)/2$ per evaluasi (Persamaan (13.4)); banyaknya langkah diketahui sebelumnya (Persamaan (13.5)), dan ketelitiannya dibatasi oleh datarnya puncak (Persamaan (13.6)).
- Metode Newton untuk optimasi, $x_{k+1} = x_k - f'(x_k)/f''(x_k)$, konvergen kuadratik di dekat optimum; untuk laba Mbak Rina galatnya tepat memenuhi $e_{k+1} = -e_k^2/(s - e_k)$, yang juga memperlihatkan larinya iterasi di luar titik belok. Kurung dengan cadangan biseksi membuatnya aman.
- *Gradient descent* konvergen bila $\alpha < 2/\lambda_{\max}$ dan, dengan langkah tetap terbaik, berlaju $(\kappa - 1)/(\kappa + 1)$ yang ditentukan bilangan kondisi Hessian; *backtracking* dengan syarat Armijo memilih langkah yang aman secara otomatis.
- Metode Newton untuk banyak variabel menyelesaikan $H\mathbf{s} = \nabla f$ di setiap langkah dan konvergen kuadratik, berapa pun bilangan kondisinya, dengan harga turunan kedua dan penyelesaian sistem linear.
- Harga tunggal terbaik Mbak Rina adalah Rp4.000 (laba 302,8 ribu rupiah sehari); dengan gorengan, pasangan terbaiknya Rp3.770 dan Rp1.435 (393,5 ribu sehari). Java dan Python mencetak hasil yang identik, dan pengoptimal SciPy sependapat.

13.12 Soal Latihan

1. Kerjakan dua langkah *golden section* lagi dengan tangan setelah langkah di Subbab 13.2.2, lalu periksa selang Anda dengan baris 2 dan 3 dari program.
2. Pemasok Mbak Rina menaikkan biaya segelas menjadi Rp2.000. Tanpa menjalankan program apa pun, berapa harga terbaik yang baru, dan berapa laba hariannya di sana? Jelaskan mengapa harga terbaik naik tepat sebesar kenaikan biaya.
3. Dengan memulai Newton biasa dari $p_0 = 6$, hitung dua iterasi pertama dengan Persamaan (13.9). Lalu tentukan rentang harga awal yang membuat Newton biasa konvergen ke Rp4.000.
4. Ubah `backtracking` agar setiap *line search* dimulai dari dua kali langkah yang diterima pada iterasi sebelumnya (dan dari 1 pada kali pertama). Berapa titik percobaan yang diperlukan sekarang? Jelaskan mengapa cara ini berhasil.
5. Tunjukkan bahwa untuk fungsi rugi kuadratik $f(\mathbf{x}) = 1/2\,\mathbf{x}^{\mathsf{T}}Q\mathbf{x} - \mathbf{b}^{\mathsf{T}}\mathbf{x}$ dengan Q simetris definit positif, metode Newton mencapai minimum dalam satu langkah dari titik awal mana pun, dan bahwa minimum itu menyelesaikan sistem linear $Q\mathbf{x} = \mathbf{b}$.
6. Misalkan Mbak Rina mengukur harga gorengan dalam ratusan rupiah, bukan ribuan, sehingga $q_2 = 10\,p_2$. Bagaimana entri Hessian berubah, apa yang terjadi pada bilangan kondisi, dan berapa langkah yang diperlukan *gradient descent* dengan langkah tetap terbaik?

13.13 Penyelesaian Soal Latihan

1. Setelah langkah 1 selangnya [2,6.944272] dengan $x_1 = 3.888544$ ($\pi = 302.5347$) dan $x_2 = 5.055728$ ($\pi = 282.3644$). Langkah 2: $\pi(x_1) > \pi(x_2)$, jadi $b = 5.055728$, x_2 yang baru adalah nilai lama $x_1 = 3.888544$, dan titik uji kiri yang baru $x_1 = 5.055728 - 0.618034 \times 3.055728 = 3.167184$ dengan $\pi(3.167184) = 1.667184 \times 600\, e^{-1.266874} = 1.667184 \times 169.03 = 281.80$. Selangnya [2,5.055728], seperti baris 2. Langkah 3: sekarang $\pi(x_1) = 281.80 < \pi(x_2) = 302.53$, jadi $a = 3.167184$, x_1 yang baru adalah 3.888544, dan titik uji kanan yang baru $x_2 = 3.167184 + 0.618034 \times 1.888544 = 4.334369$ dengan $\pi = 2.834369 \times 600\, e^{-1.733748} = 2.834369 \times 105.97 = 300.37$. Selang [3.16718,5.05573] cocok dengan baris 3.

2. Menurut Persamaan (13.8) kemiringan bernilai nol ketika margin sama dengan skala harga, $p - c = s$, berapa pun c. Jadi $p^* = 2.0 + 2.5 = 4.5$, yaitu Rp4.500. Labanya di sana $s \cdot Ae^{-(c+s)/s} = 2.5 \times 600\, e^{-1.8} = 1500 \times 0.16530 = 247.9$ ribu rupiah sehari, lebih rendah daripada sebelumnya karena setiap gelas lebih mahal. Margin terbaik, s, hanya bergantung pada seberapa cepat pembeli pergi ketika harga naik, bukan pada biaya; biaya yang lebih tinggi menggeser seluruh kurva ke kanan dan meneruskan seluruh kenaikannya ke harga. (Dengan kurva permintaan lain hanya sebagian kenaikan biaya yang diteruskan; kurva eksponensial itu istimewa.)

3. Pada $p_0 = 6$: $m = 4.5$, $e_0 = 2$. Maka $e_1 = -4/(2.5 - 2) = -8$, sehingga $p_1 = 4 - 8 = -4$, harga negatif! Berikutnya $e_2 = -64/(2.5 + 8) = -6.095$, $p_2 = -2.10$; setelah itu galatnya mengecil ($e_3 = -37.15/8.595 = -4.32$, dan seterusnya) dan iterasinya merayap kembali ke 4, tetapi hanya setelah melewati harga-harga yang tidak masuk akal. Dari Persamaan (13.9), $|e_{k+1}| < |e_k|$ mensyaratkan $|e_k| < |s - e_k|$. Untuk $e_k < 0$ ini selalu benar, karena $s - e_k = s + |e_k|$; untuk $0 < e_k$ ini benar bila $e_k < s - e_k$, yaitu $e_k < s/2 = 1.25$. Jadi setiap awal dengan $p_0 < 5.25$ (margin di bawah $1.25 + 2.5 = 3.75$) konvergen secara monoton setelah langkah pertama; awal antara 5,25 dan 6,5 konvergen setelah memutar liar melalui harga negatif; awal pada atau di atas 6,5 divergen. Pengaman bukanlah pilihan tambahan.

4. Simpan variabel `last` (mula-mula 0,5) di luar gelung, mulai setiap pencarian dari `alpha = 2.0 * last`, dan setel `last = alpha` setelah langkah diterima. Dalam versi Java perubahan yang sama memerlukan dua baris. Banyaknya titik percobaan turun dari 294 menjadi 62, dan banyaknya langkah dari 35 menjadi 28: langkah yang diterima pada masalah ini biasanya sekitar 1/128, sehingga mulai dari dua kali langkah sebelumnya hanya perlu satu kali pembagian dua, dan sesekali langkah yang digandakan langsung diterima, yang memperpendek perjalanan. Gagasannya, langkah-langkah berturutan pada masalah yang sama mempunyai ukuran baik yang serupa, jadi langkah terakhir adalah tebakan awal terbaik; menggandakan lebih dahulu memungkinkan langkah tumbuh lagi ketika lembahnya melebar.

5. Gradiennya $\nabla f = Q\mathbf{x} - \mathbf{b}$ dan Hessiannya Q, konstan. Langkah Newton dari $\mathbf{x}_0$ mana pun menyelesaikan $Q\mathbf{s} = Q\mathbf{x}_0 - \mathbf{b}$, jadi $\mathbf{s} = \mathbf{x}_0 - Q^{-1}\mathbf{b}$ dan $\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{s} = Q^{-1}\mathbf{b}$, tidak bergantung pada $\mathbf{x}_0$. Di $\mathbf{x}_1$ gradiennya $QQ^{-1}\mathbf{b} - \mathbf{b} = \mathbf{0}$ dan Q definit positif, jadi titik itu minimum, dan ia menyelesaikan $Q\mathbf{x} = \mathbf{b}$. Inilah sebabnya meminimumkan fungsi kuadratik dan menyelesaikan sistem linear simetris definit positif adalah masalah yang sama, dan mengapa pemecah iteratif di Bab 5 dapat dibaca sebagai pengoptimal.

6. Dengan $p_2 = q_2/10$, aturan rantai memberi $\partial f/\partial q_2 = \frac{1}{10}\,\partial f/\partial p_2$ dan $\partial^2 f/\partial q_2^2 = \frac{1}{100}\,\partial^2 f/\partial p_2^2$, sedangkan entri campurannya dibagi 10. Hessian di titik optimum menjadi kira-kira diag(1,0.1) H diag(1,0.1). Kelengkungan arah gorengan turun dari sekitar 180 menjadi sekitar 1,8, sedangkan kelengkungan arah teh tetap sekitar 50, sehingga nilai eigennya menjadi kira-kira 1,8

dan 50, dan $\kappa \approx 27$, bukan 3,7. Laju langkah tetap terbaik Persamaan (13.15) menjadi $26/28 \approx 0.93$, dan banyaknya langkah untuk memperoleh faktor 10^8 naik dari sekitar 34 menjadi sekitar $18.8/0.074 \approx 250$. Masalah sama, jawaban sama, kerja tujuh kali lipat, hanya karena satuannya: itulah alasan penskalaan di Subbab 13.7.

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[3] J. Kiefer, "Sequential minimax search for a maximum," *Proceedings of the American Mathematical Society*, vol. 4, no. 3, pp. 502–506, 1953.

[4] The SciPy community, "Optimization and root finding (scipy.optimize)," SciPy 1.18.1 documentation, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/optimize.html

[5] R. P. Brent, *Algorithms for Minimization without Derivatives*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1973; reprinted Mineola, NY, USA: Dover, 2002.

[6] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York, NY, USA: Springer, 2006.

[7] L. Armijo, "Minimization of functions having Lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics*, vol. 16, no. 1, pp. 1–3, 1966.

[8] NumPy Developers, "numpy 2.5.3," Python Package Index, Sep. 6, 2026. [Online]. Available: https://pypi.org/project/numpy/

[9] SciPy Developers, "scipy 1.18.1," Python Package Index, Aug. 21, 2026. [Online]. Available: https://pypi.org/project/scipy/

[10] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[11] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, 2004.

[12] Profitina, "Profitina — AI Visibility (GEO) platform," 2026. [Online]. Available: https://profitina.com

Bab 14

Metode Numerik untuk Machine Learning

Machine learning terdengar seperti jenis komputasi yang baru, tetapi di balik layarnya ia adalah analisis numerik dari bab-bab sebelumnya yang dipekerjakan pada data: pencocokan kuadrat terkecil (Bab 8) yang diselesaikan dengan faktorisasi matriks (Bab 6), fungsi rugi yang diminimumkan dengan *gradient descent* atau metode Newton (Bab 13), nilai eigen yang menentukan seberapa cepat minimasi berjalan (Bab 12), dan aritmetika *floating point* (Bab 1) yang menentukan apakah jawabannya bisa dipercaya sama sekali. Bab ini mengikuti seorang pengemudi ojek daring yang menginginkan dua ramalan dari catatan perjalanannya sendiri, dan memakainya untuk mengajarkan sisi numerik dari belajar. Di akhir bab ini Anda akan mampu mencocokkan model linear dengan persamaan normal dan dengan QR, serta menjelaskan dengan bilangan kondisi mengapa yang kedua lebih aman; melatih model yang sama dengan *gradient descent*, menghitung laju belajar yang aman dengan metode pangkat, dan menunjukkan mengapa standardisasi masukan mempercepat pelatihan berlipat-lipat; menjelaskan apa yang diubah oleh *mini-batch*, *momentum*, dan Adam; menurunkan regresi logistik, gradien dan Hessiannya, lalu menyelesaikannya dengan metode Newton (IRLS); menulis *log-loss* agar tidak pernah *overflow*; memeriksa gradien dengan selisih hingga; mengenali kapan presisi tunggal tidak cukup; dan menulis semuanya dalam Java dan Python dengan keluaran yang identik, diperiksa terhadap NumPy dan SciPy.

14.1 Dua Pertanyaan Mas Dimas

Mas Dimas mengemudikan ojek di Yogyakarta, mengambil pesanan melalui aplikasi dari pukul enam pagi sampai petang. Ia menyimpan setiap perjalanan dua bulan terakhir dalam lembar kerja: panjang perjalanan dalam kilometer, apakah saat itu jam sibuk, apakah hujan, seberapa jauh ia harus berkendara untuk menjemput pelanggan, berapa menit perjalanannya, dan apakah pelanggan membatalkan sebelum ia tiba. Ada 400 baris.

Ia mempunyai dua pertanyaan, dan keduanya soal uang. Yang pertama: *berapa lama perjalanan ini?* Aplikasi menampilkan tarif dan perkiraan waktu, tetapi ia tahu perkiraan itu sering meleset ketika hujan dan pada jam sibuk, dan pengemudi yang bisa menakar waktu dapat merencanakan harinya, istirahatnya, dan bensinnya. Yang kedua: *akankah pelanggan ini membatalkan?* Pembatalan setelah ia berkendara tiga kilometer ke titik jemput membuang bensin dan seperempat jam dengan percuma. Seandainya ia tahu pesanan mana yang kemungkinan dibatalkan, ia bisa mendahulukan pesanan lain.

Pertanyaan pertama meminta sebuah bilangan dan disebut *regresi*; pertanyaan kedua meminta peluang ya atau tidak dan disebut *klasifikasi*. *Machine learning* menjawab keduanya dengan cara yang sama: pilih rumus dengan bilangan-bilangan yang dapat disetel, disebut *bobot*, lalu pilih bobot yang membuat rumus itu paling sesuai dengan catatan. Langkah kedua itulah tempat semua kesulitan numerik berada. Dalam datanya, panjang perjalanan berkisar 0,9 sampai 15 km, jam sibuk dan hujan bernilai 0 atau 1, dan jarak jemput berkisar 0,1 sampai 4 km; 110 dari 400 pesanan dibatalkan. Fakta-fakta sederhana itu, besar bilangan dan hubungan di antaranya, akan

menentukan apakah komputasinya memerlukan 12 langkah atau ribuan, dan apakah hasilnya memberi enam digit benar atau tidak satu pun.

14.2 Gagasannya dengan Kata-Kata Sederhana

Model adalah rumus yang mengubah fakta tentang sebuah perjalanan, yaitu *fitur*-nya, menjadi ramalan. Model paling sederhana untuk waktu perjalanan adalah jumlah berbobot:

$$\hat{y} = w_0 + w_1\,\text{km} + w_2\,\text{rush} + w_3\,\text{rain} = \mathbf{x}^{\mathsf{T}}\mathbf{w}, \tag{14.1}$$

dengan $\mathbf{x} = (1,\text{km},\text{rush},\text{rain})$ menghimpun fitur satu perjalanan (angka 1 di depan mengalikan konstanta w_0), dan $\mathbf{w} = (w_0, w_1, w_2, w_3)$ adalah bobotnya. Dengan kata-kata: waktu tetap untuk berangkat dan parkir, ditambah sekian menit per kilometer, ditambah menit tambahan untuk jam sibuk dan untuk hujan.

Mempelajari bobot berarti memilihnya sehingga ramalan $\hat{y}_i$ dekat dengan waktu tercatat y_i untuk semua $N = 400$ perjalanan. "Dekat" diukur dengan *fungsi rugi* (*loss*); untuk waktu, pilihan yang lazim adalah setengah galat kuadrat rata-rata,

$$L(\mathbf{w}) = \frac{1}{2N}\sum_{i=1}^{N}(\mathbf{x}_i^{\mathsf{T}}\mathbf{w} - y_i)^2 = \frac{1}{2N}\,\|\,X\mathbf{w} - \mathbf{y}\,\|^2, \tag{14.2}$$

dengan *matriks desain* X yang mempunyai satu baris $\mathbf{x}_i^{\mathsf{T}}$ per perjalanan. Ini tepat masalah kuadrat terkecil dari Bab 8 [1], dan belajar tepat optimasi dari Bab 13. Yang baru dalam *machine learning* adalah skalanya (jutaan baris, jutaan bobot) dan, karena skala itu, pilihan metodenya: rumus yang baik-baik saja untuk 400 baris menjadi rusak, dan metode yang tampak lambat di atas kertas justru menang dalam praktik.

Mari kita pelajari model kecil dengan tangan. Ambil tiga perjalanan: 2 km dalam 10 menit, 5 km dalam 17, 8 km dalam 26, dan model $\hat{y} = w_0 + w_1\,\text{km}$. Menyamakan turunan Persamaan (14.2) dengan nol memberi dua persamaan (Subbab 14.3.1 menurunkannya secara umum):

- $3w_0 + 15w_1 = 10 + 17 + 26 = 53$ (jumlah $\sum 1 = 3$ dan $\sum \text{km} = 15$),
- $15w_0 + 93w_1 = 2\cdot 10 + 5\cdot 17 + 8\cdot 26 = 313$ (dengan $\sum \text{km}^2 = 4 + 25 + 64 = 93$).

Determinannya $3\cdot 93 - 15\cdot 15 = 54$, sehingga $w_1 = (3\cdot 313 - 15\cdot 53)/54 = 144/54 = 2.667$ dan $w_0 = (93\cdot 53 - 15\cdot 313)/54 = 234/54 = 4.333$. Aturan yang dipelajari: sekitar 4,3 menit ditambah 2,7 menit per kilometer. Pada catatan lengkap Mas Dimas, dengan jam sibuk dan hujan sekalian, program bab ini menemukan 4,0 menit ditambah 2,6 per kilometer, ditambah 5,7 pada jam sibuk dan 5,5 ketika hujan.

Matriks 2-kali-2 $\begin{pmatrix} 3 & 15 \\ 15 & 93 \end{pmatrix}$ di ruas kiri sudah memperlihatkan bahaya numerik pertama. Nilai eigennya $48 \pm \sqrt{48^2 - 54} = 95.43$ dan 0.566, jadi bilangan kondisinya $95.43/0.566 = 169$. Ukurlah jarak dalam meter alih-alih kilometer, maka entri 93 menjadi 93.000.000 dan bilangan kondisinya sekitar 10^8. Datanya tidak berubah; hanya satuannya. Sebaliknya, pusatkan dan skalakan jaraknya, ganti 2, 5, 8 dengan $-1.22, 0, 1.22$, maka matriksnya menjadi $\begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$ dengan bilangan kondisi 1. Satuan dan penskalaan, yang tak terlihat dalam matematika, menguasai numeriknya.

WAWASAN KUNCI: Tiga pertanyaan numerik menentukan apakah belajar berhasil: *Seberapa buruk kondisi masalahnya?* (menentukan digit yang hilang dan langkah yang diperlukan); *bagaimana fitur diskalakan?* (menentukan bilangan kondisi fungsi rugi); dan *adakah besaran antara yang dapat overflow atau saling menghapus?* (menentukan apakah Anda mendapat bilangan sama sekali). Sisa bab ini menjawab ketiganya untuk dua pertanyaan Mas Dimas.

14.3 Metode-Metodenya, Diturunkan

14.3.1 Kuadrat terkecil: persamaan normal atau QR

Gradien Persamaan (14.2) adalah

$$\nabla L(\mathbf{w}) = \frac{1}{N} X^{\mathsf{T}}(X\mathbf{w} - \mathbf{y}), \qquad \nabla^2 L = \frac{1}{N} X^{\mathsf{T}} X, \tag{14.3}$$

dan menyamakannya dengan nol memberi *persamaan normal*

$$X^{\mathsf{T}} X\, \mathbf{w} = X^{\mathsf{T}} \mathbf{y}. \tag{14.4}$$

Matriks $X^{\mathsf{T}}X$ simetris dan, bila kolom-kolom X bebas linear, definit positif, sehingga faktorisasi Cholesky dari Bab 6 menyelesaikan Persamaan (14.4) dengan sekitar $n^3/3$ operasi untuk n bobot, setelah sekitar Nn^2 operasi untuk membentuk matriksnya. Ini cepat dan sederhana, dan dengan cara inilah contoh tangan tadi diselesaikan.

Masalahnya adalah ketelitian. Untuk matriks persegi panjang, bilangan kondisi $\kappa(X)$ adalah perbandingan *nilai singular* terbesar dan terkecilnya, yaitu akar-akar nilai eigen $X^{\mathsf{T}}X$. Maka

$$\kappa(X^{\mathsf{T}}X) = \kappa(X)^2. \tag{14.5}$$

Membentuk $X^{\mathsf{T}}X$ mengkuadratkan bilangan kondisi, dan teori perturbasi baku untuk kuadrat terkecil [2], [3] menyatakan bahwa metode yang stabil mundur kehilangan sekitar $\log_{10}\kappa$ digit desimal dari ≈ 16 yang tersedia dalam presisi ganda. Persamaan normal karena itu kehilangan sekitar $2\log_{10}\kappa(X)$ digit, sedangkan faktorisasi QR dari Bab 6, yang bekerja langsung pada X, kehilangan sekitar $\log_{10}\kappa(X)$ (ditambah suku yang sebanding dengan κ^2 kali ukuran relatif residu, yang kecil bila modelnya cocok):

$$\frac{\|\,\widehat{\mathbf{w}} - \mathbf{w}\,\|}{\|\,\mathbf{w}\,\|} \lesssim \kappa(X)^2\,\epsilon_{\text{mach}} \text{ (persamaan normal)}, \qquad \lesssim \kappa(X)\,\epsilon_{\text{mach}} \text{ (QR)}. \tag{14.6}$$

QR dengan refleksi Householder menulis $X = QR$ dengan Q ortogonal dan R segitiga atas. Matriks ortogonal tidak mengubah panjang, sehingga $\|\,X\mathbf{w} - \mathbf{y}\,\| = \|\,R\mathbf{w} - Q^{\mathsf{T}}\mathbf{y}\,\|$, yang diminimumkan dengan menyelesaikan sistem segitiga dari n baris teratas, $R_{1:n}\mathbf{w} = (Q^{\mathsf{T}}\mathbf{y})_{1:n}$. Biayanya sekitar dua kali persamaan normal dan tidak pernah mengkuadratkan κ [4].

Untuk model empat fitur Mas Dimas, $\kappa(X^{\mathsf{T}}X) = 546$, dan kedua metode sama pada setiap digit yang dicetak. Untuk melihat perbedaannya kita memberi model sesuatu yang lebih sulit ditelan: waktu perjalanan sebagai polinomial berderajat tinggi dalam jarak, dengan kolom $1, d, d^2, \ldots, d^{12}$. Pangkat bilangan antara 1 dan 15 tumbuh dengan laju yang sangat berbeda, kolom-kolomnya menjadi hampir sejajar, dan pada derajat 12 $\kappa(X) \approx 8 \times 10^{16}$. Menurut Persamaan (14.5), $\kappa(X^{\mathsf{T}}X)$ jauh melampaui $1/\epsilon_{\text{mach}} \approx 4.5 \times 10^{15}$: dalam *floating point*, $X^{\mathsf{T}}X$ tidak dapat dibedakan dari matriks singular, dan Cholesky menjumpai bilangan negatif di bawah akar. Subbab 14.5

memperlihatkan persis hal itu. (Polinomial derajat 12 memang model yang buruk untuk waktu perjalanan; kita memakainya hanya karena ia memancing kegagalan numerik yang, dalam pekerjaan nyata, dipancing diam-diam oleh fitur-fitur yang hampir kembar.)

14.3.2 Gradient descent dan skala fitur

Untuk jutaan baris dan fitur, bahkan QR terlalu mahal, dan belajar dilakukan dengan *gradient descent* seperti pada Subbab 13.3.4, dengan gradien Persamaan (14.3). Karena Hessian fungsi rugi kuadrat terkecil adalah matriks konstan $X^{\mathsf{T}}X/N$, analisis Subbab 13.3.4 berlaku persis: iterasi konvergen untuk laju belajar $\alpha < 2/\lambda_{\max}$, dan paling baik dengan faktor $(\kappa-1)/(\kappa+1)$ per langkah, dengan $\lambda_{\max}$ dan κ kini milik $X^{\mathsf{T}}X/N$.

Nilai eigen terbesar mudah diperoleh tanpa pustaka apa pun: metode pangkat dari Bab 12 yang diterapkan pada $X^{\mathsf{T}}X/N$. Untuk fitur mentah hasilnya $\lambda_{\max} = 77.4$, didominasi kolom kilometer (rata-rata km^2 sekitar 77); nilai eigen terkecil sekitar 0,14, sehingga $\kappa \approx 546$. Bahkan dengan laju belajar terbaik, galat hanya mengecil dengan faktor $545/547 = 0.9963$ per langkah, dan memperoleh enam digit memerlukan sekitar $6\ln 10/0.0037 \approx 3{,}800$ langkah. Dengan laju praktis $\alpha = 0.012$, sekitar $1/\lambda_{\max}$, keadaannya lebih buruk: arah yang paling landai hanya mengecil dengan $1 - 0.012 \times 0.14 = 0.9983$ per langkah.

Obatnya adalah *standardisasi* setiap fitur: kurangi dengan rata-ratanya μ_j dan bagi dengan simpangan bakunya σ_j,

$$u_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j}. \tag{14.7}$$

Setelah standardisasi, setiap fitur berata-rata 0 dan bervariansi 1, kolom konstan tegak lurus terhadap kolom lain, dan $X^{\mathsf{T}}X/N$ menjadi *matriks korelasi* fitur dengan angka 1 di pojoknya. Untuk Mas Dimas nilai eigennya terletak antara 0,90 dan 1,09: $\kappa = 1.21$, dan *gradient descent* dengan $\alpha = 0.5$ memperkecil galat dengan faktor sekitar 0,55 per langkah. Modelnya sama (bobot untuk masukan terstandardisasi dapat dikonversi balik dengan tepat, seperti diperlihatkan program); hanya geometri fungsi ruginya yang berubah, dari lembah panjang yang sempit menjadi mangkuk yang hampir bulat. Inilah trik numerik tunggal yang paling ampuh dalam *machine learning*, dan alasan setiap pustaka yang serius menstandardkan atau menormalkan masukannya [5], [6].

CONTOH: *Epoch* pertama dengan tangan. Mulailah dari $\mathbf{w} = \mathbf{0}$ dengan masukan terstandardisasi. Setiap ramalan bernilai 0, sehingga galat ramalannya $-y_i$ dan menurut Persamaan (14.3) gradiennya $-\frac{1}{N}X^{\mathsf{T}}\mathbf{y}$. Komponen pertamanya adalah minus rata-rata waktu perjalanan, -27.50 menit; komponen lainnya adalah minus rata-rata setiap fitur terstandardisasi dikali waktu perjalanan, yaitu minus kovariansi fitur itu dengan waktu dibagi simpangan bakunya: -10.38 untuk jarak, -2.69 untuk jam sibuk, dan -2.69 untuk hujan. Satu langkah dengan $\alpha = 0.5$ memberi $\mathbf{w} = (13.75, 5.19, 1.35, 1.34)$, dan fungsi rugi turun dari 442,3 menjadi 113,0. Konstantanya sudah bergerak tepat setengah jalan menuju rata-rata waktu 27,5, karena arahnya terlepas dari arah-arah lain (Soal Latihan 14.2) dan kelengkungannya 1; setiap *epoch* berikutnya kembali membagi dua jarak yang tersisa. Bobot-bobot fitur bergerak menuju nilai akhirnya, 10,42 menit per simpangan baku jarak, 2,73 untuk jam sibuk, dan 2,34 untuk hujan, dengan laju yang serupa, karena setelah standardisasi semua arah mempunyai kelengkungan sekitar 1. Dengan masukan mentah, langkah pertama yang sama akan dikuasai kolom kilometer yang kelengkungannya 77, dan langkahnya harus empat puluh kali lebih kecil.

14.3.3 Mini-batch, momentum, dan Adam

Gradien Persamaan (14.3) adalah rata-rata atas semua N baris. Bila N berjuta-juta, menghitungnya untuk setiap langkah adalah pemborosan: rata-rata atas segenggam baris acak menunjuk ke arah yang hampir sama dengan biaya yang sangat kecil. *Stochastic gradient descent* (SGD) dengan *mini-batch* berukuran B mengacak baris-baris di awal setiap putaran data (sebuah *epoch*), memotongnya menjadi *batch* $\mathcal{B}$, dan mengambil satu langkah per *batch*:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{1}{B} \sum_{i \in \mathcal{B}} (\mathbf{x}_i^{\mathsf{T}} \mathbf{w} - y_i)\, \mathbf{x}_i. \tag{14.8}$$

Karena *batch*-nya acak, gradiennya adalah taksiran *takbias* dari gradien penuh: benar secara rata-rata, keliru pada setiap pengambilan. Jauh dari minimum, kekeliruan tiap pengambilan hampir tidak berarti. Di dekat minimum, gradien sebenarnya menjadi kecil tetapi deraunya tidak, sehingga dengan α tetap bobotnya berhenti konvergen dan berkelana dalam awan kecil di sekitar minimum; ukuran awan itu tumbuh dengan α dan dengan variansi gradien per baris, dan mengecil dengan B [6]. Agar konvergen penuh, SGD harus menurunkan α seiring waktu, misalnya sebanding dengan $1/t$.

Momentum ("bola berat" dari Polyak [7]) menyimpan jumlah berjalan dari gradien-gradien sebelumnya dan melangkah sepanjang jumlah itu,

$$\mathbf{m} \leftarrow \beta\, \mathbf{m} + \mathbf{g}, \qquad \mathbf{w} \leftarrow \mathbf{w} - \alpha\, \mathbf{m}, \qquad \beta = 0.9. \tag{14.9}$$

Komponen gradien yang tandanya tetap terakumulasi menjadi sekitar $1/(1-\beta) = 10$ kali ukurannya, sedangkan komponen yang berganti tanda dari langkah ke langkah, yaitu zig-zag melintasi lembah sempit, saling menghapus. Pada fungsi kuadratik berkondisi buruk, *momentum* yang disetel dengan baik memperbaiki laju dari $(\kappa-1)/(\kappa+1)$ menjadi $\left(\sqrt{\kappa}-1\right)/\left(\sqrt{\kappa}+1\right)$ [5]. Karena *momentum* mengalikan langkah efektif sekitar 10 kali, α-nya harus sekitar sepuluh kali lebih kecil; itulah sebabnya program memakai 0,005.

Adam [8] menyimpan dua rata-rata berjalan per bobot, dari gradien dan dari kuadratnya, mengoreksinya karena dimulai dari nol, lalu membagi yang satu dengan akar yang lain:

$$\mathbf{m} \leftarrow \beta_1 \mathbf{m} + (1-\beta_1)\, \mathbf{g}, \quad \mathbf{v} \leftarrow \beta_2 \mathbf{v} + (1-\beta_2)\, \mathbf{g}^2, \quad \mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\mathbf{m}/(1-\beta_1^t)}{\sqrt{\mathbf{v}/(1-\beta_2^t)} + 10^{-8}}, \tag{14.10}$$

dengan kuadrat dan akar diambil elemen demi elemen, $\beta_1 = 0.9$, $\beta_2 = 0.999$, dan t nomor langkah. Pembagian itu memberi setiap bobot langkah berukuran sekitar α tanpa memedulikan skala gradiennya, sebuah penskalaan ulang per bobot yang sudah terpasang, yang membuat Adam tahan terhadap fitur berskala buruk dan menjadi pengoptimal bawaan untuk jaringan saraf. Harganya, langkahnya tidak pernah mengecil sendiri: seperti SGD, ia berakhir dalam awan di sekitar minimum kecuali α diturunkan.

14.3.4 Regresi logistik dan metode Newton

Untuk pertanyaan kedua, apakah sebuah pesanan akan dibatalkan, jumlah berbobot $z = \mathbf{x}^{\mathsf{T}}\mathbf{w}$ bisa bernilai berapa saja, padahal peluang harus terletak antara 0 dan 1. Fungsi *logistik* atau *sigmoid* memeras yang satu ke dalam yang lain:

$$p = \sigma(z) = \frac{1}{1+e^{-z}}, \qquad \sigma'(z) = \sigma(z)\,(1-\sigma(z)). \tag{14.11}$$

Fungsi ini bernilai 0,5 pada $z = 0$, mendekati 1 untuk z besar dan 0 untuk z sangat negatif. Di sini $\mathbf{x} = (1,\text{pickup km},\text{rain},\text{rush})$.

Bobot mana yang paling cocok? Prinsip *kemungkinan maksimum* (*maximum likelihood*) menjawab: bobot yang membuat hasil yang tercatat paling mungkin terjadi. Jika perjalanan i dibatalkan ($t_i = 1$), model memberinya peluang p_i, dan jika tidak ($t_i = 0$) peluang $1 - p_i$. Dengan menganggap perjalanan-perjalanan itu saling bebas, peluang seluruh catatan adalah hasil kali bilangan-bilangan ini, dan memaksimumkan hasil kali sama dengan meminimumkan minus logaritmanya, dirata-ratakan atas perjalanan: *log-loss* atau *cross-entropy*,

$$L(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N}[\, t_i \ln p_i + (1-t_i)\ln(1-p_i)\,], \qquad p_i = \sigma(\mathbf{x}_i^{\mathsf{T}}\mathbf{w}). \tag{14.12}$$

Dengan $\ln\sigma(z) = -\ln(1+e^{-z})$ dan $\ln(1-\sigma(z)) = -\ln(1+e^{z})$, serta menulis $\operatorname{softplus}(z) = \ln(1+e^{z})$, beberapa baris aljabar memberi bentuk yang setara

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}[\,\operatorname{softplus}(z_i) - t_i z_i\,], \qquad z_i = \mathbf{x}_i^{\mathsf{T}}\mathbf{w}. \tag{14.13}$$

Menurunkan dengan $\operatorname{softplus}'(z) = \sigma(z)$ dan $\sigma' = \sigma(1-\sigma)$ memberi gradien dan Hessian:

$$\nabla L = \frac{1}{N}X^{\mathsf{T}}(\mathbf{p}-\mathbf{t}), \qquad H = \frac{1}{N}X^{\mathsf{T}}WX, \qquad W = \operatorname{diag}(p_i(1-p_i)). \tag{14.14}$$

Gradiennya berbentuk sama dengan Persamaan (14.3): fitur dikali galat ramalan. Hessiannya adalah $X^{\mathsf{T}}X$ dengan setiap baris diberi bobot $p_i(1-p_i) > 0$, sehingga definit positif dan *log-loss* bersifat *konveks*: hanya ada satu minimum dan tidak ada jebakan. Metode Newton dari Subbab 13.3.6 dapat diterapkan langsung:

$$\frac{1}{N}X^{\mathsf{T}}WX\,\mathbf{s} = \frac{1}{N}X^{\mathsf{T}}(\mathbf{p}-\mathbf{t}), \qquad \mathbf{w} \leftarrow \mathbf{w} - \mathbf{s}. \tag{14.15}$$

Setiap langkah adalah masalah kuadrat terkecil berbobot, di sini diselesaikan dengan Cholesky, dan itulah sebabnya para statistikawan menyebut metode ini *iteratively reweighted least squares* (IRLS, kuadrat terkecil berbobot ulang secara iteratif) [9]. Dengan empat bobot, setiap langkah sepele, dan konvergensi kuadratik memberi minimum sampai presisi mesin dalam sekitar enam langkah. *Gradient descent* juga konvergen, karena $p(1-p) \le 1/4$ memberi $H \preccurlyeq X^{\mathsf{T}}X/(4N)$, sehingga setiap $\alpha < 8/\lambda_{\max}(X^{\mathsf{T}}X/N)$ aman; untuk fitur terstandardisasi batas itu $8/1.226 = 6.5$, dan program memakai nilai hati-hati 2,0.

Mari meramal dengan tangan memakai bobot yang ditemukan program, $\mathbf{w} = (-4.8074, 1.2994, 1.0072, 1.1131)$. Pesanan dengan jarak jemput 2 km ketika hujan di luar jam sibuk mempunyai $z = -4.8074 + 2\times 1.2994 + 1.0072 = -1.2014$, dan $p = 1/(1+e^{1.2014}) = 1/(1+3.325) = 0.231$: kira-kira satu dari empat kemungkinan perjalanan ke titik jemput itu sia-sia. Setiap bobot juga mempunyai makna sederhana: e^{w_j} adalah faktor perubahan *odds* (peluang banding) $p/(1-p)$ ketika fitur j bertambah satu. Dengan $e^{1.2994} = 3.67$, setiap kilometer jemput tambahan mengalikan *odds* pembatalan dengan 3,7.

14.3.5 Rumus yang tidak overflow

Log-loss versi buku teks Persamaan (14.12) menyembunyikan dua jebakan bagi aritmetika *floating point* (Bab 1). Jika $z = -800$ untuk perjalanan yang dibatalkan, $e^{-z} = e^{800}$ *overflow* (bilangan *double* terbesar sekitar $1.8 \times 10^{308} \approx e^{709.8}$), p menjadi 0, dan $\ln 0 = -\infty$. Dan jika $z = 40$ untuk perjalanan yang tidak dibatalkan, $e^{-40} \approx 4 \times 10^{-18}$ lebih kecil daripada setengah epsilon mesin, $1 + e^{-40}$ dibulatkan menjadi tepat 1, $p = 1$, dan $\ln(1 - p) = \ln 0$ lagi. Keduanya adalah *jawaban salah untuk pertanyaan yang bermakna*: rugi yang sebenarnya adalah 800 dan 40.

Bentuk Persamaan (14.13) menghindari keduanya, asalkan *softplus* sendiri dihitung dengan hati-hati:

$$\text{softplus}(z) = \max(z, 0) + \text{log1p}(e^{-|z|}), \tag{14.16}$$

dengan $\text{log1p}(u) = \ln(1 + u)$ yang dihitung teliti untuk u kecil, fungsi yang tersedia di Java maupun Python. Pangkat $-|z|$ tidak pernah positif, sehingga $e^{-|z|}$ tidak mungkin *overflow*. Sigmoid Persamaan (14.11) diperlakukan dengan cara yang sama: untuk $z \geq 0$ pakai $1/(1 + e^{-z})$, dan untuk $z < 0$ bentuk yang secara aljabar sama, $e^{z}/(1 + e^{z})$. Gagasan yang sama, kurangi dengan pangkat terbesar sebelum mengeksponensialkan, memberi trik *log-sum-exp* di balik setiap lapisan *softmax* jaringan saraf [10].

14.3.6 Memeriksa gradien, dan lebar sebuah float

Gradien yang ditulis dengan tangan, seperti Persamaan (14.14), perlu diperiksa terhadap selisih pusat (Bab 9) sebelum dipercaya: $\partial L/\partial w_j \approx [L(\mathbf{w} + h\mathbf{e}_j) - L(\mathbf{w} - h\mathbf{e}_j)]/(2h)$. Galat pemotongannya berorde h^2 dan galat pembulatannya berorde $\epsilon_{\text{mach}}|L|/h$; dengan $h = 10^{-6}$ keduanya sekitar 10^{-10}, yaitu besar selisih yang dilaporkan program. Tanda yang salah atau faktor yang hilang akan tampak sebagai selisih berorde 0,01 sampai 1.

Machine learning sering menghitung dengan *presisi tunggal* (float32), yang menyimpan 24 bit signifikan alih-alih 53 bit pada presisi ganda, atau bahkan dengan format 16 bit, karena bilangan yang lebih sempit bergerak lebih cepat melalui memori dan melalui prosesor grafis [11]. Satuan pembulatannya

$$\epsilon_{32} = 2^{-24} \approx 6.0 \times 10^{-8}, \qquad \epsilon_{64} = 2^{-53} \approx 1.1 \times 10^{-16}, \tag{14.17}$$

sekitar 7 berbanding 16 digit desimal. Bobot dan gradien menoleransinya dengan baik, karena belajar itu sendiri berderau. Jumlah atas banyak baris, dan rumus yang mengurangkan bilangan yang hampir sama, tidak: dalam float32 bilangan bulat berhenti tepat pada $2^{24} = 16{,}777{,}216$, dan rumus variansi $\overline{x^2} - \bar{x}^2$ yang diterapkan pada waktu di sekitar 4.3×10^4 menit mengurangkan dua bilangan di sekitar 1.9×10^9 yang jarak antartetangganya dalam float32 adalah 128. Subbab 14.6 memperlihatkan hasilnya. Karena itu kode pustaka mengakumulasi jumlah dengan presisi yang lebih tinggi, atau berpasangan (Bab 15), walaupun datanya disimpan dalam float32 [3].

14.4 Program-Programnya

Program-program berada di `Chapter14/python` dan `Chapter14/java`, dan membaca `Chapter14/data/trips.csv`, yang dibangkitkan secara dapat diulang oleh `data/make_trips.py`. Blok-blok bangunan numerik dihimpun dalam satu modul per bahasa, ketiga program memakainya, dan setiap program Java mencetak persis apa yang dicetak kembarannya dalam Python.

14.4.1 Blok-blok bangunan

Kode 14.1 berisi pembaca CSV, penyelesaian Cholesky (Bab 6), persamaan normal, kuadrat terkecil QR Householder, rata-rata dan simpangan baku, RMSE, pembangkit bilangan acak kecil, serta sigmoid dan *softplus* stabil dari Persamaan (14.16). Pembangkitnya adalah SplitMix64, pembangkit kecil yang teruji baik dan menghasilkan deret bilangan 64 bit yang sama dalam bahasa apa pun yang mempunyai aritmetika bilangan bulat 64 bit; Python memerlukan topeng `0xFFFF...` untuk meniru putaran 64 bit, Java mendapatkannya gratis. Pembangkit bawaan Python, `random`, dan milik Java, `java.util.Random`, menghasilkan deret yang berbeda, sehingga keduanya tidak bisa dipakai bila kedua program harus mengacak perjalanan secara identik. Dengan alasan yang sama modul ini menjumlahkan bilangan dengan fungsinya sendiri, `add`, sebuah gelung biasa dari kiri ke kanan: sejak Python 3.12 fungsi bawaan `sum` untuk *float* mengompensasi galat pembulatan (Bab 15 menjelaskan caranya), yang lebih teliti tetapi bukan lagi aritmetika yang sama dengan gelung Java.

Kode 14.1 `mlcore.py`: data, kuadrat terkecil dengan persamaan normal dan dengan QR, pembangkit bilangan acak yang portabel, dan fungsi logistik yang stabil.

```python
"""Numerical building blocks for Chapter 14: data, least squares, random numbers,
and numerically stable logistic functions. Plain Python, no libraries."""
import math
import os

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")


def add(values):
    """Left-to-right sum, the same additions as a Java loop. (Since Python
    3.12 the built-in sum() compensates rounding errors, so it can differ.)"""
    s = 0.0
    for v in values:
        s += v
    return s


def load_trips():
    """Rows of trips.csv as lists of floats (header skipped)."""
    with open(os.path.join(DATA, "trips.csv"), encoding="utf-8") as f:
        next(f)
        return [[float(x) for x in line.split(",")] for line in f if line.strip()]


def cholesky_solve(a, b):
    """Solve A x = b for symmetric positive definite A (Chapter 6)."""
    n = len(a)
    L = [[0.0] * n for _ in range(n)]
    for j in range(n):
        s = a[j][j] - add(L[j][k] * L[j][k] for k in range(j))
        if s <= 0.0:
            raise ArithmeticError(f"not positive definite at column {j}")
        L[j][j] = math.sqrt(s)
        for i in range(j + 1, n):
            L[i][j] = (a[i][j] - add(L[i][k] * L[j][k] for k in range(j))) / L[j][j]
    z = [0.0] * n
    for i in range(n):                                  # L z = b
        z[i] = (b[i] - add(L[i][k] * z[k] for k in range(i))) / L[i][i]
    x = [0.0] * n
    for i in range(n - 1, -1, -1):                      # L^T x = z
        x[i] = (z[i] - add(L[k][i] * x[k] for k in range(i + 1, n))) / L[i][i]
    return x
```

Kode 14.1 (Lanjutan) *mlcore.py: data, kuadrat terkecil dengan persamaan normal dan dengan QR, pembangkit bilangan acak yang portabel, dan fungsi logistik yang stabil.*

```
def normal_equations(X, y):
    """Least squares by forming X^T X w = X^T y and using Cholesky."""
    n = len(X[0])
    a = [[add(r[i] * r[j] for r in X) for j in range(n)] for i in range(n)]
    b = [add(r[i] * t for r, t in zip(X, y)) for i in range(n)]
    return cholesky_solve(a, b)


def qr_solve(X, y):
    """Least squares by Householder QR: never forms X^T X (Chapter 6)."""
    A = [row[:] for row in X]
    c = y[:]
    m, n = len(A), len(A[0])
    for k in range(n):
        norm = math.sqrt(add(A[i][k] * A[i][k] for i in range(k, m)))
        alpha = -norm if A[k][k] >= 0 else norm
        v = [0.0] * m
        v[k] = A[k][k] - alpha
        for i in range(k + 1, m):
            v[i] = A[i][k]
        vv = add(v[i] * v[i] for i in range(k, m))
        if vv == 0.0:
            continue
        for j in range(k, n):                              # A = (I - 2vv^T/v^Tv) A
            s = 2.0 * add(v[i] * A[i][j] for i in range(k, m)) / vv
            for i in range(k, m):
                A[i][j] -= s * v[i]
        s = 2.0 * add(v[i] * c[i] for i in range(k, m)) / vv
        for i in range(k, m):
            c[i] -= s * v[i]
    w = [0.0] * n
    for i in range(n - 1, -1, -1):                         # R w = (Q^T y)[0:n]
        w[i] = (c[i] - add(A[i][j] * w[j] for j in range(i + 1, n))) / A[i][i]
    return w


def mean_std(values):
    mu = add(values) / len(values)
    var = add((v - mu) * (v - mu) for v in values) / len(values)
    return mu, math.sqrt(var)


def rmse(X, y, w):
    e = [add(a * b for a, b in zip(r, w)) - t for r, t in zip(X, y)]
    return math.sqrt(add(v * v for v in e) / len(e))
```

Kode 14.1 (Lanjutan) *mlcore.py: data, kuadrat terkecil dengan persamaan normal dan dengan QR, pembangkit bilangan acak yang portabel, dan fungsi logistik yang stabil.*

```
class Rng:
    """SplitMix64: the same stream of numbers in Python and in Java."""

    def __init__(self, seed):
        self.s = seed & 0xFFFFFFFFFFFFFFFF

    def next64(self):
        self.s = (self.s + 0x9E3779B97F4A7C15) & 0xFFFFFFFFFFFFFFFF
        z = self.s
        z = ((z ^ (z >> 30)) * 0xBF58476D1CE4E5B9) & 0xFFFFFFFFFFFFFFFF
        z = ((z ^ (z >> 27)) * 0x94D049BB133111EB) & 0xFFFFFFFFFFFFFFFF
        return z ^ (z >> 31)

    def below(self, n):
        return self.next64() % n

    def shuffle(self, items):
        for i in range(len(items) - 1, 0, -1):          # Fisher-Yates
            j = self.below(i + 1)
            items[i], items[j] = items[j], items[i]


def sigmoid(z):
    """1 / (1 + e^-z) without overflow for large |z|."""
    if z >= 0:
        return 1.0 / (1.0 + math.exp(-z))
    e = math.exp(z)
    return e / (1.0 + e)


def softplus(z):
    """log(1 + e^z) without overflow: max(z, 0) + log1p(e^-|z|)."""
    return max(z, 0.0) + math.log1p(math.exp(-abs(z)))
```

Kode 14.2 adalah modul Java-nya. Setiap jumlah diakumulasi dalam urutan yang sama dengan Python, dan itulah yang membuat keluarannya identik sampai digit terakhir bahkan untuk pencocokan polinomial yang berkondisi buruk: penjumlahan *floating point* tidak asosiatif (Bab 1), sehingga dua program yang menjumlahkan bilangan yang sama dalam urutan berbeda dapat berbeda pada bit-bit terakhir, dan masalah berkondisi buruk membesarkan bit-bit itu.

Kode 14.2 `MlCore.java`: blok-blok bangunan yang sama dalam Java.

```java
// Numerical building blocks for Chapter 14: data, least squares, random numbers,
// and numerically stable logistic functions. Plain Java, no libraries.
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class MlCore {
    static final Path DATA = Path.of("..", "data");

    static String f(String fmt, Object... xs) {
        return String.format(Locale.ROOT, fmt, xs);
    }

    /** Rows of trips.csv as arrays of doubles (header skipped). */
    static double[][] loadTrips() throws IOException {
        List<double[]> rows = new ArrayList<>();
        List<String> lines = Files.readAllLines(DATA.resolve("trips.csv"));
        for (String line : lines.subList(1, lines.size())) {
            if (line.isBlank()) continue;
            String[] p = line.split(",");
            double[] r = new double[p.length];
            for (int i = 0; i < p.length; i++) r[i] = Double.parseDouble(p[i]);
            rows.add(r);
        }
        return rows.toArray(new double[0][]);
    }

    /** Solve A x = b for symmetric positive definite A (Chapter 6). */
    static double[] choleskySolve(double[][] a, double[] b) {
        int n = a.length;
        double[][] L = new double[n][n];
        for (int j = 0; j < n; j++) {
            double s = 0.0;
            for (int k = 0; k < j; k++) s += L[j][k] * L[j][k];
            s = a[j][j] - s;
            if (s <= 0.0)
                throw new ArithmeticException(
                        "not positive definite at column " + j);
            L[j][j] = Math.sqrt(s);
            for (int i = j + 1; i < n; i++) {
                double t = 0.0;
                for (int k = 0; k < j; k++) t += L[i][k] * L[j][k];
                L[i][j] = (a[i][j] - t) / L[j][j];
```

Kode 14.2 (Lanjutan) *MlCore.java: blok-blok bangunan yang sama dalam Java.*

```
            }
        }
        double[] z = new double[n], x = new double[n];
        for (int i = 0; i < n; i++) {                       // L z = b
            double t = 0.0;
            for (int k = 0; k < i; k++) t += L[i][k] * z[k];
            z[i] = (b[i] - t) / L[i][i];
        }
        for (int i = n - 1; i >= 0; i--) {                  // L^T x = z
            double t = 0.0;
            for (int k = i + 1; k < n; k++) t += L[k][i] * x[k];
            x[i] = (z[i] - t) / L[i][i];
        }
        return x;
    }

    /** Least squares by forming X^T X w = X^T y and using Cholesky. */
    static double[] normalEquations(double[][] X, double[] y) {
        int n = X[0].length;
        double[][] a = new double[n][n];
        double[] b = new double[n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                double s = 0.0;
                for (double[] r : X) s += r[i] * r[j];
                a[i][j] = s;
            }
            double s = 0.0;
            for (int k = 0; k < X.length; k++) s += X[k][i] * y[k];
            b[i] = s;
        }
        return choleskySolve(a, b);
    }

    /** Least squares by Householder QR: never forms X^T X (Chapter 6). */
    static double[] qrSolve(double[][] X, double[] y) {
        int m = X.length, n = X[0].length;
        double[][] A = new double[m][];
        for (int i = 0; i < m; i++) A[i] = X[i].clone();
        double[] c = y.clone();
        for (int k = 0; k < n; k++) {
            double norm = 0.0;
            for (int i = k; i < m; i++) norm += A[i][k] * A[i][k];
            norm = Math.sqrt(norm);
            double alpha = A[k][k] >= 0 ? -norm : norm;
            double[] v = new double[m];
```

Kode 14.2 (Lanjutan) *MlCore.java: blok-blok bangunan yang sama dalam Java.*

```
            v[k] = A[k][k] - alpha;
            for (int i = k + 1; i < m; i++) v[i] = A[i][k];
            double vv = 0.0;
            for (int i = k; i < m; i++) vv += v[i] * v[i];
            if (vv == 0.0) continue;
            for (int j = k; j < n; j++) {          // A = (I - 2vv^T/v^Tv) A
                double s = 0.0;
                for (int i = k; i < m; i++) s += v[i] * A[i][j];
                s = 2.0 * s / vv;
                for (int i = k; i < m; i++) A[i][j] -= s * v[i];
            }
            double s = 0.0;
            for (int i = k; i < m; i++) s += v[i] * c[i];
            s = 2.0 * s / vv;
            for (int i = k; i < m; i++) c[i] -= s * v[i];
        }
        double[] w = new double[n];
        for (int i = n - 1; i >= 0; i--) {                  // R w = (Q^T y)[0:n]
            double t = 0.0;
            for (int j = i + 1; j < n; j++) t += A[i][j] * w[j];
            w[i] = (c[i] - t) / A[i][i];
        }
        return w;
    }

    static double[] meanStd(double[] v) {
        double s = 0.0;
        for (double x : v) s += x;
        double mu = s / v.length, q = 0.0;
        for (double x : v) q += (x - mu) * (x - mu);
        return new double[] {mu, Math.sqrt(q / v.length)};
    }

    static double dot(double[] a, double[] b) {
        double s = 0.0;
        for (int i = 0; i < a.length; i++) s += a[i] * b[i];
        return s;
    }
```

Kode 14.2 (Lanjutan) *MlCore.java: blok-blok bangunan yang sama dalam Java.*

```
    static double rmse(double[][] X, double[] y, double[] w) {
        double s = 0.0;
        for (int i = 0; i < X.length; i++) {
            double e = dot(X[i], w) - y[i];
            s += e * e;
        }
        return Math.sqrt(s / X.length);
    }

    /** SplitMix64: the same stream of numbers in Python and in Java. */
    static class Rng {
        long s;

        Rng(long seed) { s = seed; }

        long next64() {
            s += 0x9E3779B97F4A7C15L;
            long z = s;
            z = (z ^ (z >>> 30)) * 0xBF58476D1CE4E5B9L;
            z = (z ^ (z >>> 27)) * 0x94D049BB133111EBL;
            return z ^ (z >>> 31);
        }

        int below(int n) { return (int) Long.remainderUnsigned(next64(), n); }

        void shuffle(int[] items) {
            for (int i = items.length - 1; i > 0; i--) {     // Fisher-Yates
                int j = below(i + 1);
                int t = items[i]; items[i] = items[j]; items[j] = t;
            }
        }
    }

    /** 1 / (1 + e^-z) without overflow for large |z|. */
    static double sigmoid(double z) {
        if (z >= 0) return 1.0 / (1.0 + Math.exp(-z));
        double e = Math.exp(z);
        return e / (1.0 + e);
    }

    /** log(1 + e^z) without overflow: max(z, 0) + log1p(e^-|z|). */
    static double softplus(double z) {
        return Math.max(z, 0.0) + Math.log1p(Math.exp(-Math.abs(z)));
    }
}
```

14.4.2 Mencocokkan waktu perjalanan

Kode 14.3 mencocokkan Persamaan (14.1) dengan kedua metode, melaporkan RMSE dan R^2 (bagian variansi waktu yang dijelaskan model), lalu mencocokkan polinomial berderajat 5, 9, 11, dan 12 dalam kilometer mentah, serta berderajat 12 dalam jarak terstandardisasi. Pangkat-pangkatnya dibangun dengan perkalian berulang, bukan dengan `**` atau `Math.pow`, karena fungsi pangkat dari pustaka matematika kedua bahasa tidak dijamin membulatkan secara identik.

Kode 14.3 `linfit.py`: model waktu perjalanan dengan persamaan normal dan dengan QR, serta percobaan polinomial.

```
"""Trip time from distance, rush hour and rain: normal equations versus QR.
Usage: python linfit.py"""
from mlcore import load_trips, normal_equations, qr_solve, rmse, mean_std

rows = load_trips()
y = [r[4] for r in rows]
X = [[1.0, r[0], r[1], r[2]] for r in rows]
w_ne, w_qr = normal_equations(X, y), qr_solve(X, y)
print(f"{len(rows)} trips; model minutes = w0 + w1 km + w2 rush + w3 rain")
print("normal equations:", " ".join(f"{v:9.5f}" for v in w_ne))
print("Householder QR  :", " ".join(f"{v:9.5f}" for v in w_qr))
mu, sd = mean_std(y)
e = rmse(X, y, w_qr)
print(f"RMSE {e:.4f} minutes, R^2 {1 - e * e / (sd * sd):.4f}")

print("\npolynomials in distance: RMSE (minutes) by method")
print("degree  normal equations   Householder QR")
d = [r[0] for r in rows]
m, s = mean_std(d)
for deg, scaled in ((5, False), (9, False), (11, False), (12, False), (12, True)):
    t = [(x - m) / s if scaled else x for x in d]
    P = []
    for v in t:                                  # 1, v, v^2, ... by products
        row = [1.0]
        for k in range(deg):
            row.append(row[-1] * v)
        P.append(row)
    try:
        ne = f"{rmse(P, y, normal_equations(P, y)):.6f}"
    except ArithmeticError:
        ne = "Cholesky fails"
    label = f"{deg} std" if scaled else f"{deg} km"
    print(f"{label:6s}  {ne:>16s}   {rmse(P, y, qr_solve(P, y)):14.6f}")
```

Kode 14.4 adalah kembaran Java-nya; program ini membangun kolom-kolom polinomial dalam larik biasa dan menangkap `ArithmeticException` milik Java di tempat Python menangkap `ArithmeticError`.

Kode 14.4 `LinFit.java`: pencocokan yang sama dalam Java.

```
// Trip time from distance, rush hour and rain: normal equations versus QR.
// Usage: java LinFit
public class LinFit {
    static String row(double[] w) {
        StringBuilder sb = new StringBuilder();
        for (double v : w) sb.append(MlCore.f(" %9.5f", v));
        return sb.toString();
    }

    public static void main(String[] args) throws Exception {
        double[][] rows = MlCore.loadTrips();
        int n = rows.length;
        double[] y = new double[n], d = new double[n];
        double[][] X = new double[n][];
        for (int i = 0; i < n; i++) {
            y[i] = rows[i][4];
            d[i] = rows[i][0];
            X[i] = new double[] {1.0, rows[i][0], rows[i][1], rows[i][2]};
        }
        double[] wNe = MlCore.normalEquations(X, y), wQr = MlCore.qrSolve(X, y);
        System.out.println(n + " trips; model minutes"
                + " = w0 + w1 km + w2 rush + w3 rain");
        System.out.println("normal equations:" + row(wNe));
        System.out.println("Householder QR  :" + row(wQr));
        double[] ms = MlCore.meanStd(y);
        double e = MlCore.rmse(X, y, wQr);
        System.out.println(MlCore.f("RMSE %.4f minutes, R^2 %.4f",
                e, 1 - e * e / (ms[1] * ms[1])));

        System.out.println("\npolynomials in distance: RMSE (minutes) by method");
        System.out.println("degree  normal equations   Householder QR");
        double[] md = MlCore.meanStd(d);
        int[] degs = {5, 9, 11, 12, 12};
        boolean[] scaled = {false, false, false, false, true};
        for (int c = 0; c < degs.length; c++) {
            int deg = degs[c];
            double[][] P = new double[n][deg + 1];
            for (int i = 0; i < n; i++) {        // 1, v, v^2, ... by products
                double v = scaled[c] ? (d[i] - md[0]) / md[1] : d[i];
                P[i][0] = 1.0;
                for (int k = 1; k <= deg; k++) P[i][k] = P[i][k - 1] * v;
            }
            String ne;
            try {
                double[] w = MlCore.normalEquations(P, y);
                ne = MlCore.f("%.6f", MlCore.rmse(P, y, w));
```

Kode 14.4 (Lanjutan) *LinFit.java: pencocokan yang sama dalam Java.*

```
            } catch (ArithmeticException ex) {
                ne = "Cholesky fails";
            }
            String label = deg + (scaled[c] ? " std" : " km");
            System.out.println(MlCore.f("%-6s  %16s   %14.6f", label, ne,
                    MlCore.rmse(P, y, MlCore.qrSolve(P, y))));
        }
    }
}
```

14.4.3 Melatih dengan metode gradien

Kode 14.5 melatih model yang sama dengan lima cara, masing-masing 200 *epoch*: *gradient descent* batch penuh pada masukan mentah dan terstandardisasi, serta SGD *mini-batch* (Persamaan (14.8)), SGD dengan *momentum*, dan Adam, semuanya dengan *batch* berisi 20 pada

masukan terstandardisasi. Program lebih dahulu mencetak $\lambda_{\max}$ untuk kedua jenis masukan, yang ditemukan dengan metode pangkat. Untuk setiap jalannya, program melaporkan *rugi berlebih* (*excess loss*), yaitu seberapa jauh rugi akhir di atas minimum eksak yang ditemukan QR, dan *epoch* pertama ketika rugi berlebih itu turun di bawah 10^{-4}. Koreksi bias Adam memerlukan β_1^t dan β_2^t; program menyimpannya sebagai hasil kali berjalan alih-alih memanggil fungsi pangkat, dengan alasan yang baru disebutkan.

Kode 14.5 `train_gd.py`: pelatihan model waktu perjalanan dengan batch penuh, *mini-batch*, *momentum*, dan Adam.

```
"""Training the trip-time model by gradient methods: why scaling matters,
and what mini-batches, momentum and Adam do.   Usage: python train_gd.py"""
import math
from mlcore import load_trips, qr_solve, mean_std, Rng, add

rows = load_trips()
y = [r[4] for r in rows]
RAW = [[1.0, r[0], r[1], r[2]] for r in rows]
stats = [mean_std([r[j] for r in rows]) for j in range(3)]
STD = [[1.0] + [(r[j] - stats[j][0]) / stats[j][1] for j in range(3)] for r in rows]
N = len(rows)


def loss(X, w, idx=None):
    idx = range(N) if idx is None else idx
    total = 0.0
    for i in idx:
        e = add(a * b for a, b in zip(X[i], w)) - y[i]
        total += e * e
    return total / (2 * len(idx))


def grad(X, w, idx):
    g = [0.0] * len(w)
    for i in idx:
        e = add(a * b for a, b in zip(X[i], w)) - y[i]
        for j in range(len(w)):
            g[j] += e * X[i][j]
    return [v / len(idx) for v in g]


def lambda_max(X, iters=100):
    """Largest eigenvalue of X^T X / N by the power method (Chapter 12)."""
    n = len(X[0])
    A = [[add(r[i] * r[j] for r in X) / N for j in range(n)] for i in range(n)]
    v, lam = [1.0] * n, 0.0
    for _ in range(iters):
        u = [add(A[i][j] * v[j] for j in range(n)) for i in range(n)]
        lam = math.sqrt(add(t * t for t in u))
        v = [t / lam for t in u]
    return lam
```

Kode 14.5 (Lanjutan) *train_gd.py: pelatihan model waktu perjalanan dengan batch penuh, mini-batch, momentum, dan Adam.*

```
def run(X, method, lr, epochs, batch=N, seed=7):
    best = loss(X, qr_solve(X, y))
    w, m, v = [0.0] * 4, [0.0] * 4, [0.0] * 4
    order, rng, reached = list(range(N)), Rng(seed), None
    b1t = b2t = 1.0                                  # 0.9^t and 0.999^t
    for ep in range(1, epochs + 1):
        if batch < N:
            rng.shuffle(order)
        for s in range(0, N, batch):
            b1t, b2t = b1t * 0.9, b2t * 0.999
            g = grad(X, w, order[s:s + batch])
            for j in range(4):
                if method == "momentum":             # heavy ball, beta = 0.9
                    m[j] = 0.9 * m[j] + g[j]
                    w[j] -= lr * m[j]
                elif method == "adam":               # beta1 0.9, beta2 0.999
                    m[j] = 0.9 * m[j] + 0.1 * g[j]
                    v[j] = 0.999 * v[j] + 0.001 * g[j] * g[j]
                    mh, vh = m[j] / (1 - b1t), v[j] / (1 - b2t)
                    w[j] -= lr * mh / (math.sqrt(vh) + 1e-8)
                else:
                    w[j] -= lr * g[j]
        if reached is None and loss(X, w) - best < 1e-4:
            reached = ep
    return w, max(loss(X, w) - best, 0.0), reached


def unscale(w):
    """Weights for standardized inputs -> weights for km, rush, rain."""
    out = [w[0]] + [w[j + 1] / stats[j][1] for j in range(3)]
    out[0] -= add(w[j + 1] * stats[j][0] / stats[j][1] for j in range(3))
    return out


def main():
    print(f"lambda_max of X^T X / N: raw {lambda_max(RAW):.3f},"
          f" standardized {lambda_max(STD):.3f}")
    print("method              inputs  lr      epochs  excess loss  reached")
    cases = [("full-batch GD", RAW, "gd", 0.012, N),
             ("full-batch GD", STD, "gd", 0.5, N),
             ("mini-batch SGD", STD, "gd", 0.05, 20),
             ("SGD + momentum", STD, "momentum", 0.005, 20),
             ("Adam", STD, "adam", 0.05, 20)]
    for name, X, method, lr, batch in cases:
        w, excess, reached = run(X, method, lr, 200, batch)
        tag = "raw" if X is RAW else "std"
```

Kode 14.5 (Lanjutan) *train_gd.py: pelatihan model waktu perjalanan dengan batch penuh, mini-batch, momentum, dan Adam.*

```
        print(f"{name:18s}  {tag:6s}  {str(lr):6s}  {200:6d}  {excess:11.2e}"
              f"  {reached if reached else '-':>7}")
        if X is STD and method == "gd" and batch == N:
            w_std = w
    print("GD on standardized inputs, back in km units:",
          " ".join(f"{v:.5f}" for v in unscale(w_std)))


if __name__ == "__main__":
    main()
```

Kode 14.6 menuliskan kelima kasus sebagai larik-larik sejajar, karena Java tidak mempunyai daftar *tuple*, dan menjaga urutan operasi yang sama di mana-mana.

Kode 14.6 `TrainGd.java`: percobaan yang sama dalam Java.

```java
// Training the trip-time model by gradient methods: why scaling matters,
// and what mini-batches, momentum and Adam do.   Usage: java TrainGd
public class TrainGd {
    static double[] y;
    static double[][] RAW, STD, stats = new double[3][];
    static int N;

    static double loss(double[][] X, double[] w) {
        double total = 0.0;
        for (int i = 0; i < N; i++) {
            double e = MlCore.dot(X[i], w) - y[i];
            total += e * e;
        }
        return total / (2 * N);
    }

    static double[] grad(double[][] X, double[] w, int[] idx, int from, int to) {
        double[] g = new double[w.length];
        for (int k = from; k < to; k++) {
            int i = idx[k];
            double e = MlCore.dot(X[i], w) - y[i];
            for (int j = 0; j < w.length; j++) g[j] += e * X[i][j];
        }
        for (int j = 0; j < w.length; j++) g[j] /= (to - from);
        return g;
    }

    /** Largest eigenvalue of X^T X / N by the power method (Chapter 12). */
    static double lambdaMax(double[][] X) {
        int n = X[0].length;
        double[][] A = new double[n][n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++) {
                double s = 0.0;
                for (double[] r : X) s += r[i] * r[j];
                A[i][j] = s / N;
            }
        double[] v = {1.0, 1.0, 1.0, 1.0};
        double lam = 0.0;
        for (int it = 0; it < 100; it++) {
            double[] u = new double[n];
            for (int i = 0; i < n; i++) u[i] = MlCore.dot(A[i], v);
            lam = Math.sqrt(MlCore.dot(u, u));
            for (int i = 0; i < n; i++) v[i] = u[i] / lam;
        }
        return lam;
```

Kode 14.6 (Lanjutan) *TrainGd.java: percobaan yang sama dalam Java.*

```
    }

    /** Returns {w0..w3, excess loss, epoch reached (0 = never)}. */
    static double[] run(double[][] X, String method, double lr, int epochs,
                        int batch) {
        double best = loss(X, MlCore.qrSolve(X, y));
        double[] w = new double[4], m = new double[4], v = new double[4];
        int[] order = new int[N];
        for (int i = 0; i < N; i++) order[i] = i;
        MlCore.Rng rng = new MlCore.Rng(7);
        int reached = 0;
        double b1t = 1.0, b2t = 1.0;                  // 0.9^t and 0.999^t
        for (int ep = 1; ep <= epochs; ep++) {
            if (batch < N) rng.shuffle(order);
            for (int s = 0; s < N; s += batch) {
                b1t *= 0.9;
                b2t *= 0.999;
                double[] g = grad(X, w, order, s, Math.min(s + batch, N));
                for (int j = 0; j < 4; j++) {
                    if (method.equals("momentum")) {      // heavy ball, beta = 0.9
                        m[j] = 0.9 * m[j] + g[j];
                        w[j] -= lr * m[j];
                    } else if (method.equals("adam")) {   // beta1 0.9, beta2 0.999
                        m[j] = 0.9 * m[j] + 0.1 * g[j];
                        v[j] = 0.999 * v[j] + 0.001 * g[j] * g[j];
                        double mh = m[j] / (1 - b1t), vh = v[j] / (1 - b2t);
                        w[j] -= lr * mh / (Math.sqrt(vh) + 1e-8);
                    } else {
                        w[j] -= lr * g[j];
                    }
                }
            }
            if (reached == 0 && loss(X, w) - best < 1e-4) reached = ep;
        }
        return new double[] {w[0], w[1], w[2], w[3],
                Math.max(loss(X, w) - best, 0.0), reached};
    }
```

Kode 14.6 (Lanjutan) *TrainGd.java: percobaan yang sama dalam Java.*

```
    public static void main(String[] args) throws Exception {
        double[][] rows = MlCore.loadTrips();
        N = rows.length;
        y = new double[N];
        RAW = new double[N][];
        STD = new double[N][];
        for (int j = 0; j < 3; j++) {
            double[] col = new double[N];
            for (int i = 0; i < N; i++) col[i] = rows[i][j];
            stats[j] = MlCore.meanStd(col);
        }
        for (int i = 0; i < N; i++) {
            double[] r = rows[i];
            y[i] = r[4];
            RAW[i] = new double[] {1.0, r[0], r[1], r[2]};
            STD[i] = new double[4];
            STD[i][0] = 1.0;
            for (int j = 0; j < 3; j++)
                STD[i][j + 1] = (r[j] - stats[j][0]) / stats[j][1];
        }
        System.out.println(MlCore.f(
                "lambda_max of X^T X / N: raw %.3f, standardized %.3f",
                lambdaMax(RAW), lambdaMax(STD)));
        System.out.println(
                "method              inputs  lr      epochs  excess loss  reached");
        String[] names = {"full-batch GD", "full-batch GD", "mini-batch SGD",
                          "SGD + momentum", "Adam"};
        double[][][] xs = {RAW, STD, STD, STD, STD};
        String[] methods = {"gd", "gd", "gd", "momentum", "adam"};
        double[] lrs = {0.012, 0.5, 0.05, 0.005, 0.05};
        int[] batches = {N, N, 20, 20, 20};
        double[] wStd = null;
        for (int c = 0; c < names.length; c++) {
            double[] r = run(xs[c], methods[c], lrs[c], 200, batches[c]);
            String tag = xs[c] == RAW ? "raw" : "std";
            String reached = r[5] > 0 ? String.valueOf((int) r[5]) : "-";
            System.out.println(MlCore.f("%-18s  %-6s  %-6s  %6d  %11.2e  %7s",
                    names[c], tag, Double.toString(lrs[c]), 200, r[4], reached));
            if (c == 1) wStd = r;
        }
        double[] out = new double[4];              // back to km, rush, rain units
        double shift = 0.0;
        for (int j = 0; j < 3; j++) {
            out[j + 1] = wStd[j + 1] / stats[j][1];
            shift += wStd[j + 1] * stats[j][0] / stats[j][1];
        }
```

Kode 14.6 (Lanjutan) *TrainGd.java: percobaan yang sama dalam Java.*

```
        out[0] = wStd[0] - shift;
        StringBuilder sb =
                new StringBuilder("GD on standardized inputs, back in km units:");
        for (double v : out) sb.append(MlCore.f(" %.5f", v));
        System.out.println(sb);
    }
}
```

14.4.4 Meramal pembatalan

Kode 14.7 mula-mula mencetak *log-loss* naif dan stabil untuk empat kasus ekstrem, lalu memeriksa gradien Persamaan (14.14) terhadap selisih pusat, menjalankan metode Newton Persamaan (14.15) dari $\mathbf{w} = \mathbf{0}$, menerjemahkan bobot menjadi faktor *odds*, melatih model yang sama dengan *gradient descent* dengan jarak jemput yang distandardkan, dan akhirnya

menghitung berapa pembatalan yang tertangkap oleh ambang 0,5. Rugi naif dihitung di dalam `try`, karena Python melempar pengecualian ketika *overflow* sedangkan Java diam-diam mengembalikan tak hingga; keduanya diubah menjadi teks `inf`.

Kode 14.7 `logistic.py`: regresi logistik dengan metode Newton (IRLS) dan dengan *gradient descent*, dengan pemeriksaan gradien dan rumus yang stabil.

```
"""Will the customer cancel? Logistic regression by Newton's method (IRLS) and by
gradient descent, with stable formulas and a gradient check.
Usage: python logistic.py"""
import math
from mlcore import (load_trips, cholesky_solve, mean_std, sigmoid, softplus,
                    add)

rows = load_trips()
y = [r[5] for r in rows]
X = [[1.0, r[3], r[2], r[1]] for r in rows]      # 1, pickup km, rain, rush
N, P = len(rows), 4


def z_of(w, x):
    return add(a * b for a, b in zip(w, x))


def loss(w):
    """Mean log-loss, written with softplus so that it never overflows."""
    return add(softplus(z_of(w, x)) - t * z_of(w, x) for x, t in zip(X, y)) / N


def grad(w):
    g = [0.0] * P
    for x, t in zip(X, y):
        e = sigmoid(z_of(w, x)) - t
        for j in range(P):
            g[j] += e * x[j] / N
    return g


def hessian(w):
    H = [[0.0] * P for _ in range(P)]
    for x in X:
        p = sigmoid(z_of(w, x))
        for i in range(P):
            for j in range(P):
                H[i][j] += p * (1 - p) * x[i] * x[j] / N
    return H
```

Kode 14.7 (Lanjutan) *logistic.py: regresi logistik dengan metode Newton (IRLS) dan dengan gradient descent, dengan pemeriksaan gradien dan rumus yang stabil.*

```
def naive_loss(z, t):
    """The textbook formula -[t log p + (1-t) log(1-p)], p = 1/(1+e^-z)."""
    try:
        p = 1.0 / (1.0 + math.exp(-z))
        return -math.log(p) if t == 1 else -math.log(1.0 - p)
    except (OverflowError, ValueError):
        return math.inf


def show(v):
    return "inf" if math.isinf(v) else f"{v:.6f}"


print("one trip, stable versus naive log-loss")
for z, t in ((-5.0, 1), (-40.0, 1), (-800.0, 1), (40.0, 0)):
    print(f"  z = {z:7.1f}, cancelled = {t}: naive {show(naive_loss(z, t)):>10s},"
          f" stable {softplus(z) - t * z:.6f}")

w0, h = [-1.0, 0.5, 0.5, 0.5], 1e-6
g = grad(w0)
num = []
for j in range(P):
    wp, wm = w0[:], w0[:]
    wp[j] += h
    wm[j] -= h
    num.append((loss(wp) - loss(wm)) / (2 * h))
print("gradient check, max difference:",
      f"{max(abs(a - b) for a, b in zip(g, num)):.2e}")

print("Newton / IRLS from w = 0")
w = [0.0] * P
for k in range(1, 20):
    step = cholesky_solve(hessian(w), grad(w))        # H s = g
    w = [a - b for a, b in zip(w, step)]
    gn = math.sqrt(add(v * v for v in grad(w)))
    print(f"  {k}  loss {loss(w):.10f}  |grad| {gn:.2e}")
    if gn < 1e-10:
        break
print("weights:", " ".join(f"{v:.5f}" for v in w))
print(f"odds of cancelling x{math.exp(w[1]):.3f} per km of pickup,"
      f" x{math.exp(w[2]):.3f} in rain, x{math.exp(w[3]):.3f} in rush hour")
best = loss(w)
```

Kode 14.7 (Lanjutan) *logistic.py: regresi logistik dengan metode Newton (IRLS) dan dengan gradient descent, dengan pemeriksaan gradien dan rumus yang stabil.*

```
pk = mean_std([x[1] for x in X])
Xs = [[1.0, (x[1] - pk[0]) / pk[1], x[2], x[3]] for x in X]
X, v, ep = Xs, [0.0] * P, 0
while loss(v) - best > 1e-6 and ep < 100000:
    ep += 1
    v = [a - 2.0 * b for a, b in zip(v, grad(v))]      # fixed step 2.0
print(f"gradient descent, standardized pickup, step 2.0: {ep} epochs")
print("  to come within 1e-6 of Newton's loss")

tp = fp = fn = tn = 0
for x, t in zip(X, y):
    pred = 1 if sigmoid(z_of(v, x)) >= 0.5 else 0
    tp += pred == 1 and t == 1
    fp += pred == 1 and t == 0
    fn += pred == 0 and t == 1
    tn += pred == 0 and t == 0
print(f"threshold 0.5: accuracy {(tp + tn) / N:.3f};"
      f" caught {tp} of {tp + fn} cancellations, {fp} false alarms")
```

Kode 14.8 tidak memerlukan `try`: dalam Java `Math.exp(800.0)` bernilai tak hingga dan `Math.log(0.0)` bernilai minus tak hingga, sehingga rugi naif begitu saja menjadi tak hingga, yang dicetak `show` sebagai `inf`.

Kode 14.8 `Logistic.java`: hal yang sama dalam Java.

```java
// Will the customer cancel? Logistic regression by Newton's method (IRLS) and by
// gradient descent, with stable formulas and a gradient check.
// Usage: java Logistic
public class Logistic {
    static double[][] X;
    static double[] y;
    static int N;
    static final int P = 4;

    /** Mean log-loss, written with softplus so that it never overflows. */
    static double loss(double[] w) {
        double s = 0.0;
        for (int i = 0; i < N; i++) {
            double z = MlCore.dot(w, X[i]);
            s += MlCore.softplus(z) - y[i] * z;
        }
        return s / N;
    }

    static double[] grad(double[] w) {
        double[] g = new double[P];
        for (int i = 0; i < N; i++) {
            double e = MlCore.sigmoid(MlCore.dot(w, X[i])) - y[i];
            for (int j = 0; j < P; j++) g[j] += e * X[i][j] / N;
        }
        return g;
    }

    static double[][] hessian(double[] w) {
        double[][] H = new double[P][P];
        for (double[] x : X) {
            double p = MlCore.sigmoid(MlCore.dot(w, x));
            for (int i = 0; i < P; i++)
                for (int j = 0; j < P; j++)
                    H[i][j] += p * (1 - p) * x[i] * x[j] / N;
        }
        return H;
    }

    /** The textbook formula -[t log p + (1-t) log(1-p)], p = 1/(1+e^-z). */
    static double naiveLoss(double z, int t) {
        double p = 1.0 / (1.0 + Math.exp(-z));
        return t == 1 ? -Math.log(p) : -Math.log(1.0 - p);
    }
```

Kode 14.8 (Lanjutan) *Logistic.java: hal yang sama dalam Java.*

```
    static String show(double v) {
        return Double.isInfinite(v) ? "inf" : MlCore.f("%.6f", v);
    }

    static double norm(double[] v) { return Math.sqrt(MlCore.dot(v, v)); }

    public static void main(String[] args) throws Exception {
        double[][] rows = MlCore.loadTrips();
        N = rows.length;
        X = new double[N][];
        y = new double[N];
        for (int i = 0; i < N; i++) {                  // 1, pickup km, rain, rush
            X[i] = new double[] {1.0, rows[i][3], rows[i][2], rows[i][1]};
            y[i] = rows[i][5];
        }
        System.out.println("one trip, stable versus naive log-loss");
        double[] zs = {-5.0, -40.0, -800.0, 40.0};
        int[] ts = {1, 1, 1, 0};
        for (int c = 0; c < 4; c++)
            System.out.println(MlCore.f(
                    "  z = %7.1f, cancelled = %d: naive %10s, stable %.6f",
                    zs[c], ts[c], show(naiveLoss(zs[c], ts[c])),
                    MlCore.softplus(zs[c]) - ts[c] * zs[c]));

        double[] w0 = {-1.0, 0.5, 0.5, 0.5};
        double h = 1e-6, worst = 0.0;
        double[] g = grad(w0);
        for (int j = 0; j < P; j++) {
            double[] wp = w0.clone(), wm = w0.clone();
            wp[j] += h;
            wm[j] -= h;
            double num = (loss(wp) - loss(wm)) / (2 * h);
            worst = Math.max(worst, Math.abs(g[j] - num));
        }
        System.out.println(MlCore.f("gradient check, max difference: %.2e", worst));
```

Kode 14.8 (Lanjutan) *Logistic.java: hal yang sama dalam Java.*

```
        System.out.println("Newton / IRLS from w = 0");
        double[] w = new double[P];
        for (int k = 1; k < 20; k++) {
            double[] step = MlCore.choleskySolve(hessian(w), grad(w));   // H s = g
            for (int j = 0; j < P; j++) w[j] -= step[j];
            double gn = norm(grad(w));
            System.out.println(MlCore.f("  %d  loss %.10f  |grad| %.2e",
                    k, loss(w), gn));
            if (gn < 1e-10) break;
        }
        System.out.println(MlCore.f("weights: %.5f %.5f %.5f %.5f",
                w[0], w[1], w[2], w[3]));
        System.out.println(MlCore.f(
                "odds of cancelling x%.3f per km of pickup, x%.3f in rain,"
                + " x%.3f in rush hour",
                Math.exp(w[1]), Math.exp(w[2]), Math.exp(w[3])));
        double best = loss(w);

        double[] pick = new double[N];
        for (int i = 0; i < N; i++) pick[i] = X[i][1];
        double[] pk = MlCore.meanStd(pick);
        for (int i = 0; i < N; i++)
            X[i] = new double[] {1.0, (X[i][1] - pk[0]) / pk[1], X[i][2], X[i][3]};
        double[] v = new double[P];
        int ep = 0;
        while (loss(v) - best > 1e-6 && ep < 100000) {
            ep++;
            double[] gv = grad(v);
            for (int j = 0; j < P; j++) v[j] -= 2.0 * gv[j];      // fixed step 2.0
        }
        System.out.println(MlCore.f(
                "gradient descent, standardized pickup, step 2.0: %d epochs", ep));
        System.out.println("  to come within 1e-6 of Newton's loss");

        int tp = 0, fp = 0, fn = 0, tn = 0;
        for (int i = 0; i < N; i++) {
            int pred = MlCore.sigmoid(MlCore.dot(v, X[i])) >= 0.5 ? 1 : 0;
            if (pred == 1 && y[i] == 1) tp++;
            if (pred == 1 && y[i] == 0) fp++;
            if (pred == 0 && y[i] == 1) fn++;
            if (pred == 0 && y[i] == 0) tn++;
        }
        System.out.println(MlCore.f(
                "threshold 0.5: accuracy %.3f; caught %d of %d cancellations,"
                + " %d false alarms", (double) (tp + tn) / N, tp, tp + fn, fp));
    }
```

Kode 14.8 (Lanjutan) *Logistic.java: hal yang sama dalam Java.*

```
}
```

14.5 Menjalankan Program

```
cd Chapter14/python
python3 linfit.py
python3 train_gd.py
python3 logistic.py
cd ../java
javac -d out *.java
java -cp out LinFit
java -cp out TrainGd
java -cp out Logistic
```

Program kuadrat terkecil mencetak:

```
400 trips; model minutes = w0 + w1 km + w2 rush + w3 rain
normal equations:   4.00351   2.59547   5.68935   5.52595
Householder QR  :   4.00351   2.59547   5.68935   5.52595
RMSE 2.5693 minutes, R^2 0.9486

polynomials in distance: RMSE (minutes) by method
degree  normal equations   Householder QR
5 km            4.524659         4.524659
9 km            4.503983         4.503983
11 km           4.502751         4.502704
12 km     Cholesky fails         4.494450
12 std          4.494450         4.494450
```

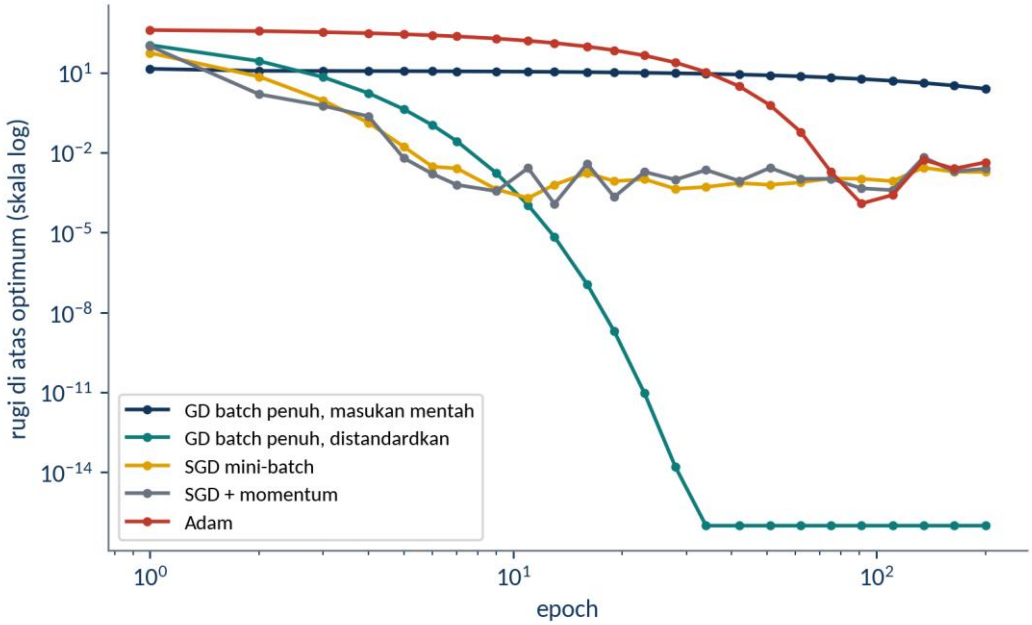
Untuk model yang sebenarnya, kedua metode memberi bobot yang sama: 4,0 menit waktu tetap, 2,6 menit per kilometer, 5,7 menit tambahan pada jam sibuk, dan 5,5 ketika hujan. RMSE-nya 2,57 menit dan $R^2 = 0.949$: keempat bilangan itu menjelaskan 95% variasi waktu perjalanan Mas Dimas, dan ramalan biasa meleset sekitar dua setengah menit, kira-kira sebesar sebaran yang diharapkan dari lampu lalu lintas.

Tabel polinomialnya adalah Persamaan (14.6) yang dibuat tampak. Pada derajat 5 ($\kappa(X) \approx 5.6 \times 10^6$, jadi $\kappa(X^\mathsf{T}X) \approx 3 \times 10^{13}$) persamaan normal masih sama dengan QR sampai enam desimal. Pada derajat 11 ($\kappa(X) \approx 2.5 \times 10^{15}$) persamaan normal mengembalikan bobot yang RMSE-nya, 4,502751, *lebih besar* daripada minimum sebenarnya 4,502704 yang ditemukan QR: persamaan normal sama sekali tidak menemukan penyelesaian kuadrat terkecil, hanya sesuatu di dekatnya. Pada derajat 12 Cholesky gagal total. Dengan jarak terstandardisasi, polinomial derajat 12 yang sama diselesaikan kedua metode tanpa kesulitan, karena pangkat-pangkat bilangan antara sekitar -1.6 dan 2.1 jauh kurang mirip satu sama lain daripada pangkat-pangkat bilangan antara 0,9 dan 15. Membangun pangkat dengan `**` alih-alih perkalian berulang, yang hanya mengubah bit terakhir sebagian entri, menggeser RMSE derajat 11 dari persamaan normal menjadi 4,502767: dalam masalah berkondisi buruk, perbedaan pembulatan pada bit terakhir masukan mengubah digit kelima hasilnya.

Program pelatihan mencetak:

```
lambda_max of X^T X / N: raw 77.409, standardized 1.092
method              inputs  lr      epochs  excess loss  reached
full-batch GD       raw     0.012      200     2.51e+00        -
full-batch GD       std     0.5        200     0.00e+00       12
mini-batch SGD      std     0.05       200     1.88e-03       92
SGD + momentum      std     0.005      200     2.51e-03       24
Adam                std     0.05       200     4.35e-03       88
GD on standardized inputs, back in km units: 4.00351 2.59547 5.68935 5.52595
```

Gambar 14.1 menggambar rugi berlebih terhadap *epoch* untuk kelima jalannya. *Descent* batch penuh pada masukan mentah, dibatasi oleh $\alpha < 2/77.4$ dan diperlambat oleh $\kappa = 546$, masih 2,5 di atas minimum setelah 200 *epoch*. Pada masukan terstandardisasi, algoritma yang sama mencapai 10^{-4} setelah 12 *epoch* dan presisi mesin tak lama kemudian, persis penurunan geometris yang diramalkan di Subbab 13.3.4. Ketiga metode *mini-batch* mula-mula turun cepat, karena setiap *epoch* berisi 20 langkah, bukan satu, lalu macet di lantai derau antara 10^{-3} dan 10^{-2}; kolom "reached" menunjukkan bahwa masing-masing sempat menyentuh 10^{-4} pada suatu *epoch* lalu menjauh lagi. Itulah perilaku yang diramalkan Subbab 14.3.3 untuk laju belajar tetap. Untuk 400 baris, *descent* batch penuh pada masukan terstandardisasi jelas pilihan terbaik; untuk 400 juta baris, ketika satu gradien penuh semahal sejuta langkah *mini-batch*, metode *mini-batch* yang menang, dan lantai deraunya diturunkan dengan laju belajar yang mengecil. Baris terakhir memastikan bahwa bobot terstandardisasi, setelah dikonversi balik, tepat sama dengan bobot QR.



Gambar 14.1 Rugi di atas optimum kuadrat terkecil selama pelatihan: *gradient descent* batch penuh pada masukan mentah dan terstandardisasi, serta SGD *mini-batch*, *momentum*, dan Adam pada masukan terstandardisasi (kedua sumbu logaritmik).

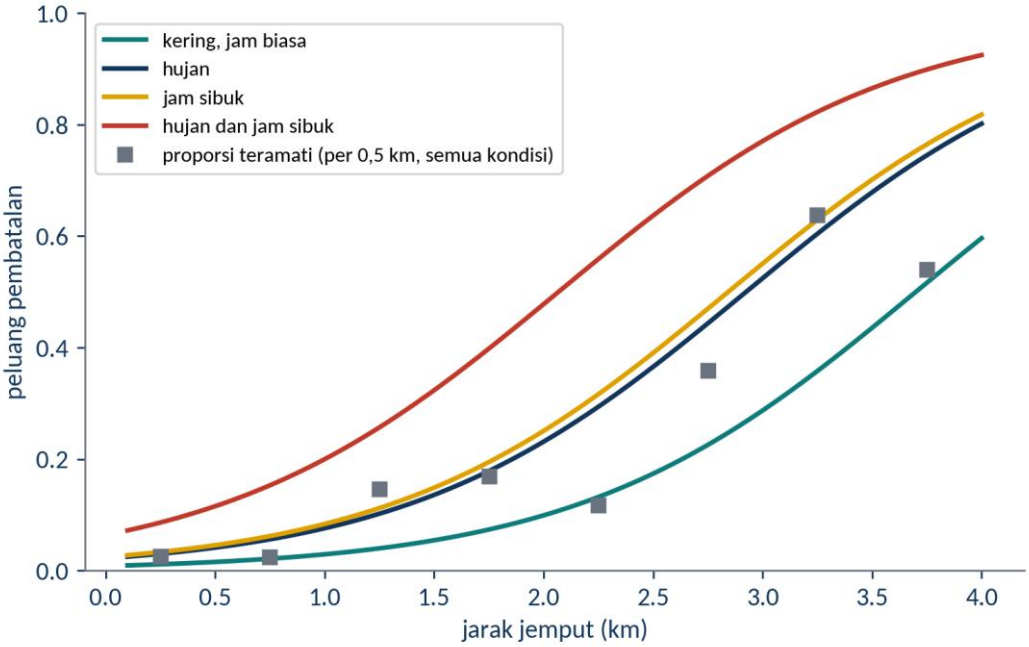
Program logistik mencetak:

```
one trip, stable versus naive log-loss
  z =    -5.0, cancelled = 1: naive   5.006715, stable 5.006715
  z =   -40.0, cancelled = 1: naive  40.000000, stable 40.000000
  z =  -800.0, cancelled = 1: naive         inf, stable 800.000000
  z =    40.0, cancelled = 0: naive         inf, stable 40.000000
gradient check, max difference: 2.98e-10
Newton / IRLS from w = 0
  1  loss 0.4596129941  |grad| 5.70e-02
  2  loss 0.4341017682  |grad| 1.40e-02
  3  loss 0.4317674110  |grad| 1.64e-03
  4  loss 0.4317353430  |grad| 2.64e-05
  5  loss 0.4317353354  |grad| 6.52e-09
  6  loss 0.4317353354  |grad| 4.68e-16
weights: -4.80740 1.29936 1.00720 1.11312
odds of cancelling x3.667 per km of pickup, x2.738 in rain, x3.044 in rush hour
gradient descent, standardized pickup, step 2.0: 128 epochs
  to come within 1e-6 of Newton's loss
threshold 0.5: accuracy 0.790; caught 57 of 110 cancellations, 31 false alarms
```

Blok pertama adalah Persamaan (14.16) yang sedang bekerja: untuk z sedang, rumus naif dan stabil sama, dan pada dua kasus ekstrem rumus naif mengembalikan tak hingga padahal rugi sebenarnya 800 atau 40. Pemeriksaan gradien cocok sampai 3×10^{-10}. Metode Newton memerlukan enam langkah, dan panjang gradiennya 5.7×10^{-2}, 1.4×10^{-2}, 1.6×10^{-3}, 2.6×10^{-5}, 6.5×10^{-9}, 4.7×10^{-16} memperlihatkan ciri khas berlipatnya digit yang benar. *Gradient descent* memerlukan 128 *epoch* untuk sampai dalam jarak 10^{-6} dari rugi yang sama.

Bobot-bobotnya dapat diterjemahkan menjadi saran yang bisa dipakai Mas Dimas: setiap kilometer jemput mengalikan *odds* pembatalan dengan 3,7, hujan dengan 2,7, jam sibuk dengan 3,0. **Gambar 14.2** menggambar peluang ramalan terhadap jarak jemput untuk empat kombinasi hujan dan jam sibuk, bersama proporsi pembatalan yang teramati dalam kelompok setengah kilometer. Pada siang yang kering, pesanan berjarak 1 km dibatalkan dengan peluang sekitar 0,03; ketika hujan pada jam sibuk, pesanan yang sama berpeluang 0,20, dan yang berjarak 3 km

berpeluang 0,77. Ambang 0,5 memberi akurasi 0,79, dibanding 0,725 untuk aturan malas "tidak ada yang membatalkan", dan menangkap 57 dari 110 pembatalan dengan 31 alarm palsu. Di mana ia memasang ambangnya adalah pilihannya: ambang yang lebih rendah menangkap lebih banyak pembatalan tetapi juga menolak lebih banyak pesanan yang baik. Model memberi saran; pengemudi yang memutuskan.



Gambar 14.2 Peluang sebuah pesanan dibatalkan, sebagaimana dipelajari regresi logistik, terhadap jarak jemput untuk empat kombinasi hujan dan jam sibuk; kotak menunjukkan proporsi teramati per setengah kilometer atas semua kondisi.

14.6 Membandingkan dengan NumPy dan SciPy

Kode 14.9 memeriksa jawaban dengan `lstsq` dari NumPy, yang menyelesaikan kuadrat terkecil dengan dekomposisi nilai singular [12], menghitung bilangan kondisi yang dikutip dalam bab ini, mencocokkan model logistik dengan BFGS dari SciPy memakai `log_expit` pustaka yang stabil [13], dan mengulang percobaan variansi dalam float64 dan float32. Program ini dijalankan dengan NumPy 2.5.3 dan SciPy 1.18.1 [14], [15].

Kode 14.9 `check_numpy.py`: hasil bab ini diperiksa dengan NumPy dan SciPy, bilangan kondisi, dan percobaan float32.

```
"""Chapter 14 answers checked with NumPy and SciPy, plus condition numbers and a
float32 experiment.   Usage: python check_numpy.py   (needs numpy and scipy)"""
import numpy as np
from scipy.optimize import minimize
from scipy.special import expit, log_expit
from mlcore import DATA

d = np.loadtxt(f"{DATA}/trips.csv", delimiter=",", skiprows=1)
km, rush, rain, pick, mins, canc = d.T
X = np.column_stack([np.ones(len(d)), km, rush, rain])
w, *_ = np.linalg.lstsq(X, mins, rcond=None)
print("lstsq weights:", " ".join(f"{v:.5f}" for v in w))
F = X[:, 1:]
Z = np.column_stack([np.ones(len(d)), (F - F.mean(0)) / F.std(0)])
print(f"cond(X^T X): raw {np.linalg.cond(X.T @ X):.1f},"
      f" standardized {np.linalg.cond(Z.T @ Z):.3f}")
for deg in (5, 9, 11, 12):
    V = np.vander(km, deg + 1, increasing=True)
    print(f"degree {deg:2d}: cond(X) {np.linalg.cond(V):.2e},"
          f" cond(X^T X) {np.linalg.cond(V.T @ V):.2e}")

A = np.column_stack([np.ones(len(d)), pick, rain, rush])


def f(w):
    z = A @ w
    return -np.mean(canc * log_expit(z) + (1 - canc) * log_expit(-z))


def g(w):
    return A.T @ (expit(A @ w) - canc) / len(d)


r = minimize(f, np.zeros(4), jac=g, method="BFGS", options={"gtol": 1e-10})
print("BFGS logistic weights:", " ".join(f"{v:.5f}" for v in r.x),
      f"loss {r.fun:.10f}")

t = mins + 43200.0          # a clock in minutes, 30 days into the month
for dt in (np.float64, np.float32):
    x = t.astype(dt)
    naive = np.mean(x * x) - np.mean(x) ** 2        # E[x^2] - E[x]^2
    two_pass = np.mean((x - np.mean(x)) ** 2)
    print(f"variance, {dt.__name__:7s}: naive {float(naive):12.4f},"
          f" two-pass {float(two_pass):10.4f}")
```

```
lstsq weights: 4.00351 2.59547 5.68935 5.52595
cond(X^T X): raw 546.1, standardized 1.209
degree  5: cond(X) 5.63e+06, cond(X^T X) 3.17e+13
degree  9: cond(X) 3.15e+12, cond(X^T X) 1.62e+23
degree 11: cond(X) 2.55e+15, cond(X^T X) 9.39e+26
degree 12: cond(X) 7.62e+16, cond(X^T X) 5.73e+28
BFGS logistic weights: -4.80740 1.29936 1.00720 1.11312 loss 0.4317353354
variance, float64: naive     128.4051, two-pass   128.4051
variance, float32: naive     128.0000, two-pass   128.4051
```

Bobot kedua model sama dengan milik kita pada setiap digit yang dicetak. Bilangan kondisinya menegaskan Persamaan (14.5): $3.17 \times 10^{13} \approx (5.63 \times 10^{6})^{2}$. Untuk derajat 9 ke atas, $\kappa(X^{\mathsf{T}}X)$ yang dihitung melampaui 10^{16}, dan bilangan sebesar itu sendiri tidak dapat diandalkan: ia hanya mengatakan "singular secara numerik". Standardisasi menurunkan $\kappa(X^{\mathsf{T}}X)$ model empat fitur dari 546 menjadi 1,21.

Dua baris terakhir memperlihatkan jebakan float32 dari Subbab 14.3.6. Rumus dua lintasan, yang lebih dahulu mengurangkan rata-rata lalu mengkuadratkan bilangan-bilangan kecil, memberi variansi yang benar, 128,4051, dalam kedua presisi. Rumus satu lintasan memberi tepat

128,0000 dalam float32: kedua rata-rata yang dikurangkannya berada di sekitar 1.87×10^9, tempat bilangan float32 yang berurutan berjarak 128, sehingga selisihnya hanya bisa kelipatan 128. Variansi 128,0 atau 0,0 atau 256,0 sama-sama mungkin keluar; bahwa hasilnya dekat dengan kebenaran hanyalah keberuntungan.

14.7 Membuatnya Lebih Baik

- **Regularisasi.** Menambahkan $\frac{\lambda}{2} \| \mathbf{w} \|^2$ pada fungsi rugi (*ridge regression*, untuk $\lambda > 0$) mengubah persamaan normal menjadi $(X^\top X + N\lambda I)\mathbf{w} = X^\top \mathbf{y}$. Setiap nilai eigen naik sebesar $N\lambda$, sehingga bilangan kondisi turun, Cholesky tidak bisa gagal lagi, dan bobot berhenti mengejar derau. Inilah obat numerik dan sekaligus obat statistik untuk fitur yang hampir kembar [5]. (Bobot konstan w_0 biasanya tidak ikut dikenai penalti.)
- **Sisihkan sebagian data.** Model hanya berguna bila ia dapat meramal perjalanan yang belum pernah dilihatnya. Cocokkan pada sekitar 80% catatan dan ukur RMSE atau akurasinya pada 20% sisanya. Polinomial derajat 12 dalam jarak selalu mencocokkan perjalanan yang dipakai untuk mencocokkannya setidaknya sebaik garis lurus dalam jarak, dan biasanya meramal perjalanan baru lebih buruk; hanya perjalanan yang disisihkan yang bisa mengungkapnya.
- **Jadwalkan laju belajar.** Untuk SGD dan Adam, turunkan α dengan suatu faktor setelah sejumlah *epoch*, atau sebanding dengan $1/\sqrt{t}$; lantai derau pada **Gambar 14.1** ikut turun.
- **Simpan akumulator float64.** Bila data atau bobot berupa float32, akumulasikan jumlah dan rugi dalam float64 (atau pakai penjumlahan berpasangan, Bab 15). Biayanya hampir nol dan menghilangkan masalah presisi yang paling parah.

14.8 Memakainya Sungguhan

Kedua model Mas Dimas dapat dibangun untuk usaha kecil mana pun yang menyimpan catatan. Langkah-langkahnya sama, entah catatannya berisi perjalanan, pesanan *laundry*, waktu tunggu pasien di antrean puskesmas, atau hasil panen setiap baris pohon di kebun.

- **Simpan catatan di lembar kerja dan ekspor sebagai CSV.** Satu baris per kejadian, satu kolom per fakta, hanya bilangan (0 dan 1 untuk tidak dan ya). Catatlah fakta *sebelum* hasilnya diketahui, karena hanya fakta itulah yang tersedia ketika ramalan diperlukan: jarak jemput diketahui ketika pesanan masuk, waktu perjalanan baru sesudahnya. Catatan yang berisi nama atau nomor telepon adalah data pribadi menurut UU PDP 27/2022; model-model ini tidak memerlukannya sama sekali, jadi jangan sertakan kolom seperti itu sejak awal.
- **Standardkan, cocokkan dengan QR, dan sisihkan seperlima.** Pakai Persamaan (14.7) dengan rata-rata dan simpangan baku baris-baris yang dipakai untuk mencocokkan, cocokkan pada empat perlima baris, dan ukur galatnya pada sisanya. Bila galat pada baris yang disisihkan jauh lebih besar daripada pada baris yang dicocokkan, modelnya terlalu lentur untuk datanya.
- **Pilih ambang dari uang, bukan dari 0,5.** Untuk keputusan ya atau tidak, bandingkan biaya setiap kesalahan. Bagi Mas Dimas, menerima pesanan yang kemudian dibatalkan menghabiskan sekitar Rp6.000 bensin dan waktu; menolak pesanan yang sebenarnya akan jadi kehilangan marginnya, sekitar Rp12.000 untuk perjalanan biasa. Menerima pesanan berpeluang batal p layak bila harapan keuntungan $(1-p) \times 12{,}000$ melebihi harapan kerugian $p \times 6{,}000$, yaitu bila $p < 12{,}000/18{,}000 = 0.67$. Ambang yang tepat baginya 0,67, bukan 0,5; pengemudi dengan motor yang lebih boros bensin akan memilih ambang yang lebih rendah.
- **Cocokkan ulang setiap bulan.** Lalu lintas, cuaca, dan pelanggan berubah. Model yang dicocokkan pada musim kemarau akan terlalu optimistis pada musim hujan; mencocokkan

ulang pada dua bulan terakhir menjaganya tetap jujur, dan metode Newton membuat pencocokan ulang berlangsung seketika.

Sebagai contoh kedua, ambil puskesmas kecil yang ingin memberi tahu pasien berapa lama mereka akan menunggu. Setiap kunjungan dalam catatan antreannya memberi waktu kedatangan, banyaknya orang yang sudah menunggu, apakah yang bertugas dokter atau bidan, dan waktu tunggunya. Model linear seperti Persamaan (14.1) dengan fitur-fitur itu, dicocokkan dengan QR, meramal waktu tunggu dalam selisih beberapa menit, dan kehati-hatian numerik yang sama berlaku: standardkan fiturnya, bersihkan catatan dari nama, dan periksa galat pada hari-hari yang disisihkan. Yang tidak boleh dilakukan model adalah memutuskan siapa yang dilayani lebih dulu: urutan penanganan adalah keputusan medis oleh petugas yang berjaga, dan pasien yang merasa makin sakit selama menunggu harus segera memberi tahu perawat, bukan menunggu waktu ramalan mana pun.

14.9 Metode Numerik dalam Praktik: Profitina

Fitur-fitur AI Profitina hanya berjalan pada model lokal: Qwen2.5 melalui Ollama, dengan model 7B untuk pekerjaan yang menuntut mutu dan model yang sangat kecil (qwen3:1.7b) sebagai jalur cepat opsional untuk tugas klasifikasi singkat [16]. Model kecil yang menjawab "ya atau tidak" untuk teks pendek sedang mengerjakan, dalam skala yang jauh lebih besar, apa yang dikerjakan regresi logistik Mas Dimas: mengubah jumlah berbobot menjadi peluang dengan sigmoid atau sepupunya untuk banyak kelas, *softmax*, yang evaluasi stabilnya adalah bentuk *log-sum-exp* dari Subbab 14.3.5.

Pipeline umpan balik platform itu membuat sisa bab ini menjadi konkret. Pengguna mengirim umpan balik atas jawaban AI; umpan balik dikurasi menjadi *dataset*; model dilatih dengannya dan *diukur* sebelum dirilis [16]. Pelatihan adalah minimasi fungsi rugi dengan metode gradien dari Subbab 14.3.3, dan pengukuran adalah evaluasi pada data yang disisihkan seperti di Subbab 14.7. Fitur Link Safety memperlihatkan mengapa ambang sebuah pengklasifikasi adalah keputusan, bukan rumus: lantai aturannya, yang menggabungkan beberapa sinyal, menangkap 27 dari 28 teks penipuan realistis tanpa satu pun positif palsu pada 30 teks normal dalam tolok ukur internal, dan pendapat AI lokal opsional berada di atasnya [16]. Menggeser ambang berarti menukar penipuan yang lolos dengan alarm palsu, persis seperti ambang Mas Dimas menukar pembatalan yang lolos dengan pesanan yang ditolak. Dan tren skor mingguan yang ditampilkan dasbor adalah jumlah atas tabel yang terus bertambah, tempat saran akumulasi di Subbab 14.7 berlaku.

14.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Menyelesaikan kuadrat terkecil dengan menginvers $X^\top X$.** Membentuk $X^\top X$ mengkuadratkan bilangan kondisi (Persamaan (14.5)), dan menginversnya menambah galat. Pakailah QR atau rutin kuadrat terkecil dari pustaka; pakai persamaan normal hanya bila $\kappa(X)$ diketahui kecil.
- **Fitur yang tidak diskalakan.** Fitur dalam rupiah di samping fitur dalam kilometer membuat κ sangat besar dan *gradient descent* merangkak. Standardkan dengan Persamaan (14.7), dan ingat untuk menerapkan rata-rata dan simpangan baku data *pelatihan* pada data baru.
- **Laju belajar yang disalin dari tutorial.** Laju yang aman bergantung pada $\lambda_{\max}$ data Anda. Perkirakan dengan beberapa langkah metode pangkat, atau mulai dari yang kecil lalu naikkan.

- **Mengodekan log-loss persis seperti di buku teks.** $\ln\sigma(z)$ dan $\ln(1-\sigma(z))$ *overflow* atau menghasilkan $-\infty$ untuk ramalan yang sangat yakin. Pakai bentuk *softplus* Persamaan (14.13) dengan Persamaan (14.16), atau `log_expit` dari pustaka.
- **Mengira lantai derau SGD sebagai konvergensi.** Rugi *mini-batch* yang berhenti turun mungkin sedang berkelana dalam awan di sekitar minimum. Bandingkan dengan evaluasi batch penuh, dan turunkan laju belajarnya.
- **Hanya memercayai akurasi.** Bila hanya 27,5% pesanan yang dibatalkan, "tidak ada yang membatalkan" sudah 72,5% akurat. Laporkan berapa banyak kasus positif yang tertangkap dan berapa alarm palsu yang muncul.
- **Menjumlahkan dalam float32.** Rumus satu lintasan dan jumlah berjalan yang panjang kehilangan segalanya dalam presisi tunggal; akumulasikan dalam float64.
- **Membandingkan dua implementasi bit demi bit tanpa menetapkan urutan operasi.** Urutan penjumlahan yang berbeda, fungsi pangkat yang berbeda, atau pembangkit acak yang berbeda memberi bit terakhir yang berbeda; kondisi yang buruk membesarkannya. Tetapkan urutan, fungsi, dan pembangkitnya, seperti yang dilakukan program-program di sini.

14.11 Ringkasan Bab

- Mempelajari model berarti mencocokkan bobot dengan meminimumkan fungsi rugi: kuadrat terkecil Persamaan (14.2) untuk bilangan, *log-loss* Persamaan (14.12) untuk peluang. Metode numeriknya adalah metode dari Bab 6, 8, 12, dan 13.
- Persamaan normal Persamaan (14.4) mengkuadratkan bilangan kondisi (Persamaan (14.5)) dan kehilangan dua kali lebih banyak digit daripada QR (Persamaan (14.6)); pada fitur polinomial berderajat tinggi, persamaan normal memberi minimum yang salah pada derajat 11 dan gagal pada derajat 12, sedangkan QR tidak.
- *Gradient descent* pada kuadrat terkecil konvergen dengan laju yang ditentukan $\kappa(X^\mathsf{T}X)$; metode pangkat memberi laju belajar yang aman, dan standardisasi Persamaan (14.7) memangkas κ dari 546 menjadi 1,21 dan *epoch* yang diperlukan untuk 10^{-4} dari ribuan menjadi 12.
- SGD *mini-batch*, *momentum* Persamaan (14.9), dan Adam Persamaan (14.10) menukar ketepatan per langkah dengan langkah yang murah; dengan laju belajar tetap, mereka berakhir di lantai derau.
- Regresi logistik mempunyai gradien $X^\mathsf{T}(\mathbf{p}-\mathbf{t})/N$ dan Hessian $X^\mathsf{T}WX/N$ (Persamaan (14.14)); fungsi ruginya konveks, dan metode Newton (IRLS, Persamaan (14.15)) konvergen dalam enam langkah.
- Rumus yang stabil (Persamaan (14.16)) menjaga *log-loss* tetap hingga; pemeriksaan gradien menangkap kesalahan turunan; float32 mempunyai sekitar 7 digit (Persamaan (14.17)) dan memerlukan kehati-hatian dalam penjumlahan dan pengurangan.
- Untuk Mas Dimas: waktu perjalanan sekitar 4,0 menit ditambah 2,6 per kilometer, ditambah 5,7 pada jam sibuk dan 5,5 ketika hujan; setiap kilometer jemput mengalikan *odds* pembatalan dengan 3,7.

14.12 Soal Latihan

1. Cocokkan $\hat{y} = w_0 + w_1$ km dengan tangan pada tiga perjalanan di Subbab 14.2 dengan jarak terstandardisasi ($u = -1.2247, 0, 1.2247$), dan tunjukkan bahwa konversi balik memberi garis yang sama, $4.333 + 2.667$ km.

2. Tunjukkan bahwa Hessian fungsi rugi kuadrat terkecil Persamaan (14.2) adalah $X^\top X/N$, dan bahwa setelah fitur distandardkan dan kolom konstan disertakan, baris dan kolom pertama $X^\top X/N$ adalah (1,0, … ,0).
3. Turunkan Persamaan (14.13) dari Persamaan (14.12), lalu gradien pada Persamaan (14.14).
4. Mas Dimas mencatat jarak jemputnya dalam meter, bukan kilometer. Bobot mana dari model logistik yang berubah, dan bagaimana? Apa yang terjadi pada bilangan kondisi $X^\top X/N$ dan pada banyaknya *epoch gradient descent*, dan apakah metode Newton terpengaruh?
5. Tambahkan regularisasi *ridge* pada `normal_equations` (parameter `lam`, menambahkan `N * lam` pada diagonal kecuali entri pertama) dan jalankan ulang polinomial derajat 12 dalam kilometer mentah dengan $\lambda = 10^{-8}$ dan dengan $\lambda = 10^{-6}$. Apakah Cholesky masih gagal? Jelaskan dengan nilai eigen.
6. Dengan bobot yang dipelajari, tentukan jarak jemput yang membuat pesanan pada jam sibuk ketika hujan berpeluang 50% dibatalkan.

14.13 Penyelesaian Soal Latihan

1. Dengan $u = (-1.2247, 0, 1.2247)$: $\sum 1 = 3$, $\sum u = 0$ dan $\sum u^2 = 2 \times 1.2247^2 = 3.0$. Persamaan normalnya $3w_0 = 53$ dan $3.0\, w_1 = \sum u_i y_i = -1.2247 \times 10 + 0 + 1.2247 \times 26 = 19.596$, sehingga $w_0 = 17.667$ dan $w_1 = 6.532$, ditemukan tanpa eliminasi apa pun karena matriksnya diagonal. Konversi balik dengan $\mu = 5$ dan $\sigma = \sqrt{6} = 2.4495$: kemiringan per kilometer $6.532/2.4495 = 2.667$ dan konstantanya $17.667 - 2.667 \times 5 = 4.333$, garis yang sama seperti sebelumnya.

2. $L = \frac{1}{2N}\sum_i (\mathbf{x}_i^\top \mathbf{w} - y_i)^2$, sehingga $\partial L/\partial w_j = \frac{1}{N}\sum_i (\mathbf{x}_i^\top \mathbf{w} - y_i)\, x_{ij}$ dan $\partial^2 L/\partial w_j\, \partial w_k = \frac{1}{N}\sum_i x_{ij}\, x_{ik}$, yaitu entri (j,k) dari $X^\top X/N$. Dengan kolom konstan $x_{i0} = 1$ dan fitur terstandardisasi, entri $(0,0)$ adalah $\frac{1}{N}\sum 1 = 1$ dan entri $(0,k)$ adalah $\frac{1}{N}\sum_i u_{ik}$, rata-rata fitur terstandardisasi, yang bernilai 0. Jadi konstanta terlepas dari bobot-bobot lain, salah satu alasan mengapa standardisasi membantu.

3. Dengan $p = \sigma(z)$: $\ln p = -\ln(1 + e^{-z})$ dan $\ln(1-p) = \ln\frac{e^{-z}}{1+e^{-z}} = -z - \ln(1+e^{-z})$. Maka $-[t \ln p + (1-t)\ln(1-p)] = t\ln(1+e^{-z}) + (1-t)(z + \ln(1+e^{-z})) = \ln(1+e^{-z}) + z - tz$. Karena $\ln(1+e^{-z}) + z = \ln(e^z(1+e^{-z})) = \ln(1+e^z) = \text{softplus}(z)$, rugi per perjalanan adalah $\text{softplus}(z) - tz$, yaitu Persamaan (14.13). Menurunkan terhadap w_j, dengan $\partial z/\partial w_j = x_j$ dan $\text{softplus}'(z) = e^z/(1+e^z) = \sigma(z)$, memberi $(\sigma(z) - t)\, x_j$; merata-ratakan atas perjalanan memberi $\frac{1}{N}X^\top(\mathbf{p} - \mathbf{t})$.

4. Fitur jemput dikalikan 1000, sehingga bobotnya dibagi 1000 (1,2994 menjadi 0,0012994), dan ramalannya tidak berubah. Kolom X yang bersangkutan 1000 kali lebih besar, entri-entri $X^\top X/N$ yang terkait naik 10^3 dan 10^6 kali, dan $\lambda_{\max}$ menjadi sekitar 5.6×10^6 (rata-rata kuadrat jarak jemput dalam meter). Laju belajar yang aman turun menjadi sekitar 10^{-6}, padahal arah bobot-bobot lain masih memerlukan langkah berorde 1, sehingga *gradient descent* akan memerlukan jutaan *epoch*. Metode Newton tidak terpengaruh: ia invarian terhadap penskalaan seperti itu, karena $H\mathbf{s} = \mathbf{g}$ berubah secara konsisten di kedua ruas, dan tetap konvergen dalam enam langkah. Standardisasi membuat *gradient descent* berperilaku serupa.

5. Dengan suku *ridge*, matriksnya menjadi $X^\top X + N\lambda D$, dengan D matriks identitas yang posisi pertamanya nol; menambahkan $N\lambda$ pada diagonal menaikkan nilai eigen arah-arah nonkonstan kira-kira sebesar itu. Dengan $\lambda = 10^{-8}$ kenaikannya $400 \times 10^{-8} = 4 \times 10^{-6}$, dan Cholesky *masih gagal* pada kolom 12: entri diagonal kolom itu adalah $\sum d^{24}$, sekitar 3×10^{29}, dan galat

pembulatan pada pivot terakhirnya jauh lebih besar daripada 4×10^{-6}. Dengan $\lambda = 10^{-6}$ kenaikannya 4×10^{-4}, cukup untuk menjaga pivot terakhir tetap positif, dan pencocokan berhasil dengan RMSE 4,500724, sedikit di atas 4,494450 milik QR tanpa regularisasi karena penalti menarik bobot ke arah nol. Pelajarannya, parameter *ridge* tidak mempunyai makna yang lepas dari satuan: pada pangkat mentah, λ yang tepat hanya ditemukan dengan coba-coba, sedangkan pada fitur terstandardisasi, ketika setiap entri diagonal $X^{\top}X/N$ bernilai 1, $\lambda = 10^{-3}$ bermakna sama dalam setiap masalah.

6. Peluang 0,5 berarti $z = 0$: $-4.8074 + 1.2994\,d + 1.0072 + 1.1131 = 0$, sehingga $1.2994\,d = 2.6871$ dan $d = 2.07$ km. Untuk jarak jemput lebih dari sekitar 2 km ketika hujan pada jam sibuk, pembatalan lebih mungkin terjadi daripada tidak, sesuai dengan kurva merah pada **Gambar 14.2**.

Daftar Pustaka

[1] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed. Baltimore, MD, USA: Johns Hopkins University Press, 2013.

[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[4] L. N. Trefethen and D. Bau, *Numerical Linear Algebra*, twenty-fifth anniversary ed. Philadelphia, PA, USA: SIAM, 2022.

[5] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[6] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Review*, vol. 60, no. 2, pp. 223–311, 2018.

[7] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[8] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015. [Online]. Available: https://arxiv.org/abs/1412.6980

[9] P. McCullagh and J. A. Nelder, *Generalized Linear Models*, 2nd ed. London, U.K.: Chapman and Hall, 1989.

[10] P. Blanchard, D. J. Higham, and N. J. Higham, "Accurately computing the log-sum-exp and softmax functions," *IMA Journal of Numerical Analysis*, vol. 41, no. 4, pp. 2311–2330, 2021.

[11] IEEE, *IEEE Standard for Floating-Point Arithmetic*, IEEE Std 754-2019, 2019.

[12] NumPy Developers, "numpy.linalg.lstsq," NumPy 2.5 documentation, 2026. [Online]. Available: https://numpy.org/doc/stable/reference/generated/numpy.linalg.lstsq.html

[13] The SciPy community, "scipy.special.log_expit," SciPy 1.18.1 documentation, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.special.log_expit.html

[14] NumPy Developers, "numpy 2.5.3," Python Package Index, Sep. 6, 2026. [Online]. Available: https://pypi.org/project/numpy/

[15] SciPy Developers, "scipy 1.18.1," Python Package Index, Aug. 21, 2026. [Online]. Available: https://pypi.org/project/scipy/

[16] Profitina, "Profitina — AI Visibility (GEO) platform," 2026. [Online]. Available: https://profitina.com

Bab 15

Komputasi Numerik Cepat dan Paralel

Metode numerik yang benar tetapi memakan waktu satu jam, bagi kebanyakan orang, adalah metode yang tidak akan mereka pakai. Bab ini mengambil satu komputasi dari Bab 5, yaitu suhu wajan martabak, menghaluskannya sampai menjadi lambat, lalu mempercepatnya dengan empat cara yang berbeda: dengan mengeluarkan *interpreter* dari gelung terdalam, dengan kompilasi, dengan membagi kerja ke beberapa inti prosesor, dan, yang paling ampuh, dengan memilih algoritma yang lebih baik. Di tengah jalan kita bertemu satu-satunya bahaya numerik yang ditambahkan komputasi paralel pada bahaya-bahaya lainnya: jumlah yang sama, bila ditambahkan dalam urutan berbeda, memberi jawaban yang berbeda. Di akhir bab ini Anda akan mampu memperkirakan biaya sebuah komputasi dalam banyaknya operasi dan lalu lintas memori sebelum menjalankannya; mengukurnya dengan jujur; memvektorkan gelung dengan NumPy dan menjelaskan mengapa itu membantu; membagi komputasi ke beberapa *thread* dalam Java dan Python tanpa mengubah hasilnya; meramalkan percepatan terbaik yang mungkin dengan hukum Amdahl dan menjelaskan mengapa percepatan nyata lebih rendah; mengenali metode mana yang aman diparalelkan (Jacobi) dan mana yang perlu ditata ulang (Gauss–Seidel, ditata ulang sebagai urutan merah-hitam); membuat jumlah paralel teliti dan dapat diulang dengan penjumlahan berpasangan dan penjumlahan Kahan; serta menilai kapan algoritma yang lebih baik mengalahkan perangkat keras yang lebih banyak.

15.1 Satu Jam untuk Satu Gambar

Di Bab 5, Pak Joni ingin tahu bagaimana panas menyebar di wajan besi persegi gerobak martabaknya, empat puluh sentimeter tiap sisi, dengan keempat tepinya bersuhu 200, 60, 80, dan 150 derajat Celsius. Metode Jacobi, Gauss–Seidel, dan SOR menghitung suhu tunak pada grid titik yang kasar, dan gambarnya menjawab pertanyaannya: di bagian mana pelat cukup panas untuk martabak manis dan di bagian mana hanya cukup hangat untuk menjaga roti bakar tidak dingin.

Kini seorang pandai besi di pasar yang sama, yang membuat wajan untuk selusin gerobak, meminta lebih. Ia ingin mencoba berbagai letak kompor dan ukuran pelat, dan ia menginginkan suhu setiap milimeter, karena titik dingin selebar satu sentimeter adalah tempat martabak tetap pucat. Setiap milimeter pada pelat 40 cm berarti grid $400 \times 400 = 160{,}000$ suhu yang tidak diketahui. Program Jacobi dari Bab 5, yang ditulis dengan gelung biasa, melakukan sekitar 15 juta pembaruan titik per detik dalam Python pada komputer yang dipakai untuk bab ini, yang kedengarannya cepat. Tetapi Subbab 15.3.4 menunjukkan bahwa Jacobi memerlukan sekitar 340.000 sapuan atas grid itu agar tenang pada resolusi ini, dan $160{,}000 \times 340{,}000 = 5.4 \times 10^{10}$ pembaruan dengan laju 15 juta per detik berarti **satu jam** untuk satu gambar. Si pandai besi menginginkan dua puluh gambar sebelum makan siang.

Situasi yang sama muncul di mana pun metode numerik meninggalkan buku teks. Model cuaca membagi Indonesia menjadi jutaan sel; rencana irigasi menyimulasikan seribu kemungkinan musim kemarau (Bab 16 melakukan persis hal itu); model *machine learning*, seperti diperlihatkan Bab 14, melewati jutaan baris berkali-kali. Mempercepat komputasi seperti itu sebagian adalah urusan rekayasa, tetapi keputusan yang paling menentukan bersifat numerik: seberapa banyak

kerja yang dilakukan metodenya, bagaimana kerja itu dapat dibagi, dan apakah pembagian itu mengubah jawabannya.

15.2 Gagasannya dengan Kata-Kata Sederhana

15.2.1 Ke mana waktu habis

Komputer menghabiskan waktu untuk tiga hal. Ia melakukan *aritmetika*: satu inti modern melakukan beberapa miliar penjumlahan atau perkalian per detik, lebih banyak lagi bila ia menerapkan satu instruksi pada beberapa bilangan sekaligus. Ia *memindahkan data*: bilangan harus bergerak dari memori utama melalui beberapa tingkat *cache* ke unit aritmetika, dan memori utama hanya mengirim beberapa miliar bilangan per detik, dibagi oleh semua inti. Dan ia melakukan *pembukuan*: bahasa yang diinterpretasikan seperti Python memeriksa tipe setiap bilangan, mencari setiap operasi, dan mengalokasikan setiap hasil, yang memakan puluhan nanodetik untuk setiap satu penjumlahan.

Untuk pembaruan Jacobi satu titik wajan, $u_{\text{new}} = 1/4\,(u_N + u_S + u_W + u_E)$, aritmetikanya 4 operasi (tiga penjumlahan dan satu perkalian). Dalam Python biasa, pembukuan di sekitar 4 operasi itu memakan sekitar 65 nanodetik (1/15,6 juta detik), kira-kira seratus kali lebih banyak daripada aritmetikanya. Pelajaran pertama karena itu sederhana: dalam Python, keluarkan gelung dari *interpreter*. NumPy melakukannya: satu baris aritmetika larik menjalankan gelung terkompilasi atas semua titik, dan program menjadi sekitar sepuluh kali lebih cepat. Java, yang programnya dikompilasi ke kode mesin selagi berjalan, sekitar sepuluh kali lebih cepat lagi.

15.2.2 Mengerjakan beberapa hal sekaligus

Komputer dengan beberapa inti dapat memperbarui beberapa bagian wajan sekaligus. Bagi 400 baris menjadi dua jalur berisi 200; satu inti memperbarui jalur atas, inti lain jalur bawah. Bila kerjanya terbagi sempurna, dua inti memerlukan separuh waktu.

Apakah sebuah metode dapat dibagi seperti ini bergantung pada apa yang dibaca setiap pembaruan. Dalam sapuan Jacobi, setiap nilai baru dihitung hanya dari grid *lama*, sehingga urutan pembaruan tidak berpengaruh dan pembagian jalur mana pun memberi bilangan yang persis sama, bit demi bit. Gauss–Seidel berbeda: setiap pembaruan membaca tetangga yang *baru saja* diperbarui, sehingga jawabannya bergantung pada urutan, dan dua inti yang mengerjakan baris bertetangga akan membaca nilai setengah jadi milik satu sama lain. Obatnya, seperti ditunjukkan Subbab 15.3.3, adalah mewarnai titik-titik seperti papan catur lalu memperbarui semua titik merah, kemudian semua titik hitam.

15.2.3 Jumlah yang sama, jawaban yang berbeda

Kerja paralel sering berakhir dengan sebuah jumlah: suhu rata-rata wajan, total air yang dipakai dalam satu musim, rugi atas semua baris sebuah *dataset*. Jumlah paralel membiarkan setiap inti menjumlahkan potongannya sendiri, lalu menjumlahkan jumlah-jumlah parsial itu. Dalam aritmetika eksak hasilnya sama. Dalam *floating point* tidak, karena penjumlahan *floating point* tidak asosiatif (Bab 1). Contoh kecil dalam presisi ganda: $(10^{16} + 1) - 10^{16} = 0$, karena $10^{16} + 1$ dibulatkan kembali ke 10^{16}, sedangkan $(10^{16} - 10^{16}) + 1 = 1$. Menjumlahkan sepuluh juta salinan 0,1 satu demi satu memberi 999999,99983897, tetapi membagi penjumlahan yang sama menjadi empat potongan memberi 1000000,00004026. Program yang jawabannya berubah menurut banyaknya inti yang kebetulan menjalankannya sulit diuji dan sulit dipercaya; Subbab

15.3.5 menunjukkan cara membuat jawabannya sekaligus teliti dan tidak bergantung pada banyaknya inti.

> **WAWASAN KUNCI:** Ada tiga cara mempercepat komputasi, dan ketiganya sebaiknya dicoba dalam urutan ini: *kerjakan lebih sedikit* (algoritma yang lebih baik, sering berfaktor ratusan), *kerjakan dengan beban tambahan yang lebih kecil* (kode tervektorisasi atau terkompilasi, faktor sepuluh sampai seratus), dan *kerjakan bersamaan* (inti paralel, paling baik berfaktor sebanyak intinya, biasanya kurang).

15.3 Metode-Metodenya, Diturunkan

15.3.1 Model biaya: operasi dan memori

Misalkan sebuah komputasi melakukan W operasi aritmetika dan memindahkan Q bait antara memori utama dan prosesor. Dengan laju aritmetika puncak F (operasi per detik) dan lebar pita memori B (bait per detik), waktunya paling sedikit

$$T \geq \max\left(\frac{W}{F}, \frac{Q}{B}\right). \tag{15.1}$$

Perbandingan $I = W/Q$, operasi per bait, adalah *intensitas aritmetika*. Bila $I < F/B$ komputasinya *terikat memori* (*memory-bound*): aritmetika yang lebih cepat atau inti yang lebih banyak tidak membantu begitu saluran memori penuh. Batas sederhana ini, yang digambar sebagai "atap" (*roofline*) pada grafik log-log, adalah cara para insinyur kinerja memutuskan apa yang perlu dioptimalkan [1].

Satu sapuan Jacobi pada grid $n \times n$ melakukan $W = 4n^2$ operasi. Setiap titik membaca empat tetangga, tetapi tiga di antaranya sudah dibaca untuk titik-titik di dekatnya dan masih berada di *cache*, sehingga dalam kasus terbaik setiap sapuan mengalirkan grid lama masuk dan grid baru keluar: $Q \approx 2 \times 8n^2 = 16n^2$ bait. Intensitasnya $I = 4/16 = 0.25$ operasi per bait. Satu inti biasa menghitung $F \approx 10^{10}$ operasi per detik sedangkan sistem memori biasa mengirim $B \approx 2 \times 10^{10}$ bait per detik yang dibagi oleh semua inti, jadi $F/B \approx 0.5$ untuk *satu* inti dan lebih kecil lagi untuk beberapa inti: Jacobi terikat memori, dan batas laju pembaruannya sekitar $B/16 \approx 1.25 \times 10^9$ pembaruan per detik. Program Java mengukur 1,36 miliar pembaruan per detik pada satu inti, tepat di batas itu (grid 400 baris, 1,3 MB per larik, sebagian muat di *cache* besar mesin uji), dan Subbab 15.5 memperlihatkan bahwa inti kedua tidak membantu. Itulah Persamaan (15.1) yang berbicara: bila memori menjadi leher botol, inti tambahan hanya ikut mengantre di pintu yang sama.

15.3.2 Vektorisasi

Vektorisasi berarti menyatakan komputasi sebagai operasi pada seluruh larik alih-alih gelung atas elemen-elemennya. Dalam NumPy, satu sapuan Jacobi utuh adalah satu pernyataan:

```
v[1:-1, 1:-1] = 0.25 * (u[:-2, 1:-1] + u[2:, 1:-1] + u[1:-1, :-2] + u[1:-1, 2:])
```

Setiap irisan seperti `u[:-2, 1:-1]` adalah grid yang digeser satu baris atau satu kolom, sebuah *view* atas memori yang sama tanpa menyalin. Keempat penjumlahan dan perkaliannya masing-masing berjalan sebagai gelung terkompilasi, sehingga pembukuan *interpreter* dibayar lima kali per sapuan, bukan lima kali per titik [2]. Harganya adalah larik sementara untuk jumlah antara, yang menambah lalu lintas memori; inilah sebabnya NumPy, dengan sekitar 130 juta pembaruan

per detik, sepuluh kali lebih cepat daripada gelung Python tetapi sepuluh kali lebih lambat daripada Java, yang gelung terkompilasinya menyimpan keempat nilai di register.

Urutan operasi di dalam ungkapan itu persis sama dengan urutan dalam gelung, $((u_N + u_S) + u_W) + u_E$ dikali 0,25, dan masing-masing dibulatkan dengan cara yang sama, sehingga program tervektorisasi mencetak digit yang sama dengan program gelung dan dengan Java. Vektorisasi mengubah kecepatan, bukan aritmetikanya. (Rutin pustaka yang menjumlahkan larik lain soalnya: `sum` milik NumPy menjumlahkan secara berpasangan, bukan dari kiri ke kanan, dan itulah sebabnya program-program bab ini mencetak jumlah dengan banyak digit yang wajar dan mengapa Subbab 15.3.5 penting.)

Prosesor modern juga dapat menerapkan satu instruksi pada 4 atau 8 *double* sekaligus (SIMD, *single instruction multiple data*). Gelung terkompilasi seperti milik Java memakai instruksi ini bila kompilatornya dapat membuktikan bahwa itu aman; Java juga menawarkan Vector API yang eksplisit, yang masih berstatus fitur inkubasi di JDK 25 (putaran inkubasi kesepuluh) dan di JDK 26 [3].

15.3.3 Percepatan paralel dan batas-batasnya

Misalkan bagian f dari waktu jalan sebuah program dapat dibagi sempurna di antara p inti dan sisanya, $1 - f$, tidak dapat (membaca masukan, memulai *thread*, menggabungkan hasil, dan bagian yang tidak diparalelkan siapa pun). Waktu pada p inti adalah $T_p = T_1\,[(1 - f) + f/p]$, dan *percepatannya* (*speedup*)

$$S(p) = \frac{T_1}{T_p} = \frac{1}{(1 - f) + f/p} \le \frac{1}{1 - f}. \tag{15.2}$$

Inilah *hukum Amdahl* [4]. Dengan $f = 0.95$, delapan inti memberi $S = 1/(0.05 + 0.11875) = 5.93$, dan tidak ada jumlah inti yang memberi lebih dari 20. Bagian serial, sekecil apa pun, akhirnya mendominasi. Gustafson mengamati bahwa dalam praktik orang memakai inti yang lebih banyak untuk menyelesaikan masalah yang *lebih besar* dalam waktu yang sama, dan untuk banyak metode numerik bagian paralelnya tumbuh bersama masalahnya sedangkan bagian serialnya tidak, sehingga percepatan yang dapat dicapai ikut tumbuh [5]. Untuk wajan, kedua pandangan itu berlaku: pada $n = 100$ koordinasi antar-*thread* per sapuan lebih mahal daripada kerja yang dibaginya, dan pada $n = 400$ atau lebih kerjanya yang mendominasi.

Percepatan nyata jatuh di bawah Persamaan (15.2) karena tiga alasan lagi. *Sinkronisasi*: setiap sapuan Jacobi harus selesai di semua jalur sebelum sapuan berikutnya dimulai, sehingga setiap sapuan diakhiri dengan *thread-thread* yang menunggu yang paling lambat. *Lebar pita memori*: seperti ditunjukkan Persamaan (15.1), kerja yang terikat memori berhenti berskala begitu saluran memori penuh. *Beban tambahan*: membuat tugas dan menyerahkannya ke *thread* memakan beberapa mikrodetik, yang berarti ketika satu sapuan hanya berlangsung seratus mikrodetik.

Komputasi mana yang boleh dibagi sama sekali? Pembagian aman bila tidak ada bagian yang menulis nilai yang dibaca bagian lain pada langkah yang sama. Jacobi hanya membaca grid lama `u` dan hanya menulis grid baru `v`, sehingga jalur-jalur tidak pernah saling mengganggu: itulah sebabnya program di Subbab 15.4 memberi hasil yang identik dengan satu *thread* maupun banyak. Gauss–Seidel memperbarui di tempat, sehingga nilai titik (i, j) bergantung pada apakah tetangga atasnya sudah diperbarui; dua *thread* pada jalur bertetangga akan menghasilkan nilai yang bergantung pada pewaktuan, sebuah *race condition* (kondisi balapan). *Urutan merah-hitam*

menghilangkan ketergantungan itu. Warnai titik (i, j) merah bila $i + j$ genap dan hitam bila ganjil, seperti papan catur. Setiap tetangga titik merah berwarna hitam dan sebaliknya. Jadi pembaruan semua titik merah hanya membaca titik hitam, yang tidak berubah selama setengah sapuan itu, dan boleh dikerjakan dalam urutan apa pun dan secara paralel; lalu semua titik hitam diperbarui dari titik merah yang baru. Dengan relaksasi berlebih, inilah *SOR merah-hitam*, yang pada masalah ini konvergen persis secepat SOR dengan urutan alami [6].

> **CONTOH:** Satu sapuan dengan tangan, dibagi di antara dua *thread*. Ambil wajan yang hanya berisi 3×3 titik dalam, semuanya mula-mula 0, dengan tepi dari Bab 5 (kiri 200, kanan 60, atas 80, bawah 150). Satu sapuan Jacobi memberi, baris demi baris: $1/4\,(80 + 200) = 70$, $1/4 \cdot 80 = 20$, $1/4\,(80 + 60) = 35$; lalu 50, 0, 15; lalu $1/4\,(150 + 200) = 87.5$, $1/4 \cdot 150 = 37.5$, $1/4\,(150 + 60) = 52.5$. Berikan baris 1 kepada *thread* A dan baris 2 dan 3 kepada *thread* B. Setiap *thread* hanya membaca grid lama, yang tidak diubah siapa pun selama sapuan, sehingga tidak ada bedanya apakah A selesai sebelum B mulai, sesudah B selesai, atau bersamaan: kesembilan bilangan itu selalu sama. Sekarang ukur apa yang dibeli pembagian itu. Untuk SOR merah-hitam pada grid 400×400, Java memerlukan 0,357 detik pada satu *thread* dan 0,266 pada dua, percepatan $S = 1.34$. Menyelesaikan Persamaan (15.2) dengan $p = 2$ untuk bagian paralelnya, $1/S = (1 - f) + f/2$ memberi $f = 2(1 - 1/1.34) = 0.51$: pada kenyataannya hanya separuh waktu yang berperilaku seolah-olah terbagi, sisanya lalu lintas memori dan sinkronisasi yang tidak dapat dibagi dengan inti kedua.

15.3.4 Mengerjakan lebih sedikit: banyaknya sapuan

Semua yang di atas membuat setiap sapuan lebih murah. Banyaknya sapuan bergantung pada metodenya. Bab 5 menunjukkan bahwa galat Jacobi pada masalah model $n \times n$ mengecil per sapuan dengan jari-jari spektral $\rho_J = \cos(\pi/(n + 1))$ [7]. Untuk n besar, $\cos x \approx 1 - x^2/2$ memberi $1 - \rho_J \approx \pi^2/(2(n + 1)^2)$, dan memperkecil galat dengan faktor ε memerlukan

$$k_J \approx \frac{\ln(1/\varepsilon)}{1 - \rho_J} \approx \frac{2(n + 1)^2}{\pi^2} \ln\frac{1}{\varepsilon} \tag{15.3}$$

sapuan. Untuk SOR dengan relaksasi optimal, $\rho = \omega_{\text{opt}} - 1 \approx 1 - 2\pi/(n + 1)$, sehingga

$$k_{\text{SOR}} \approx \frac{n + 1}{2\pi} \ln\frac{1}{\varepsilon}. \tag{15.4}$$

Banyaknya sapuan Jacobi tumbuh dengan n^2 dan SOR dengan n; perbandingannya sekitar $4(n + 1)/\pi$, yaitu 130 untuk $n = 100$ dan 510 untuk $n = 400$. Menggandakan resolusi melipatempatkan sapuan Jacobi *dan* melipatempatkan titik per sapuan: total kerjanya tumbuh seperti n^4, enam belas kali setiap penggandaan. Tidak ada jumlah inti yang realistis yang dapat mengimbanginya; metode yang lebih baik bisa. Untuk $n = 400$ dan perubahan di bawah 10^{-6} per sapuan, perubahan Jacobi terukur 3.168×10^{-2} setelah 2000 sapuan dan laju $\rho_J = \cos(\pi/401) = 1 - 3.07 \times 10^{-5}$ memberi $2000 + \ln(3.168 \times 10^4)/(3.07 \times 10^{-5}) \approx 340{,}000$ sapuan, bilangan yang dipakai di Subbab 15.1. SOR merah-hitam berhenti setelah 1171.

Lebih baik lagi adalah metode yang total kerjanya hanya tumbuh seperti banyaknya bilangan tak diketahui $N = n^2$: *multigrid* menghaluskan galat pada hierarki grid yang makin kasar dan menyelesaikan masalah model dalam banyak siklus yang tetap, tidak bergantung pada n [8]. Dan pemecah *langsung sparse* memfaktorkan matriks sistem linearnya sekali, seperti faktorisasi LU di Bab 6 tetapi dengan mengurutkan bilangan tak diketahui secara cerdik (*nested dissection*) agar faktor-faktornya tetap jarang; untuk grid 2-D biayanya tumbuh seperti $N^{3/2}$ [9]. `spsolve` dari

SciPy melakukannya dalam kurang dari satu detik untuk 160.000 bilangan tak diketahui, seperti diperlihatkan Subbab 15.6.

15.3.5 Jumlah yang teliti dan dapat diulang

Jumlahkan n bilangan $x_1, \ldots, x_n$ dari kiri ke kanan dalam *floating point* dengan satuan pembulatan u (Bab 1). Setiap penjumlahan membuat galat relatif paling besar u, dan galat yang dibuat pada langkah k terbawa sampai akhir, sehingga jumlah terhitung $\hat{s}$ memenuhi [10]

$$|\hat{s} - s| \;\le\; (n-1)\,u \sum_{i=1}^{n} |\,x_i\,| + O(u^2). \tag{15.5}$$

Untuk sepuluh juta salinan 0,1, batasnya $10^7 \times 1.1 \times 10^{-16} \times 10^6 \approx 1.1 \times 10^{-3}$; galat sebenarnya, 1.6×10^{-4}, lebih kecil karena galat-galat pembulatan sebagian saling menghapus, tetapi tetap tumbuh bersama n. Mekanismenya mudah dilihat: begitu jumlah berjalan mendekati 10^6, jarak antar-*double* di sana 1.16×10^{-10}, dan setiap 0,1 yang ditambahkan dibulatkan ke kelipatan jarak itu, kehilangan sekitar 10^{-11} setiap kali.

Penjumlahan berpasangan (*pairwise summation*) menjumlahkan kedua belahan secara terpisah lalu menjumlahkan hasilnya, secara rekursif. Setiap bilangan kemudian hanya ikut dalam sekitar $\log_2 n$ penjumlahan, dan batasnya menjadi

$$|\hat{s} - s| \;\le\; \lceil \log_2 n \rceil\, u \sum_{i=1}^{n} |\,x_i\,| + O(u^2), \tag{15.6}$$

lebih kecil dengan faktor $n/\log_2 n$: untuk $n = 10^7$, 24 alih-alih sepuluh juta. Penjumlahan berpasangan secara alami paralel, karena kedua belahan dapat dijumlahkan pada inti yang berbeda, dan itulah yang dipakai `sum` milik NumPy [2]. Fungsi bawaan `sum` milik Python melangkah lebih jauh pada versi 3.12: untuk *float* fungsi itu kini memakai algoritma terkompensasi sejenis yang dijelaskan berikut ini, sehingga `sum([0.1] * 10_000_000)` mengembalikan tepat 1000000.0 dalam Python 3.14, sedangkan gelung biasa tidak [11].

Penjumlahan terkompensasi Kahan [12] menyimpan variabel kedua c yang menampung bagian rendah yang hilang pada penjumlahan terakhir dan mengurangkannya dari suku berikutnya:

Pseudokode 15.1 Penjumlahan terkompensasi Kahan: c memulihkan galat pembulatan setiap penjumlahan dan menyuapkannya ke penjumlahan berikutnya.

```
KAHAN-SUM(x[1..n])
 1  s = 0;  c = 0
 2  for i = 1 to n
 3      y = x[i] - c          // suku berikutnya, dikoreksi dengan yang hilang
 4      t = s + y             // besar + kecil: bit rendah y hilang di sini
 5      c = (t - s) - y       // (t - s) yang benar-benar ditambahkan; dikurangi y
 6      s = t                 //   menyisakan tepat bagian yang hilang
 7  return s
```

Batas galatnya $2u\sum|x_i| + O(nu^2)\sum|x_i|$, pada dasarnya tidak bergantung pada n [10]. Biayanya empat operasi per suku alih-alih satu, dan algoritma ini tidak boleh "disederhanakan" oleh kompilator pengoptimal yang mengira $(t - s) - y$ bernilai nol; Java dan Python menghormati urutan yang ditulis, tetapi kompilator C dengan opsi *floating point* yang agresif tidak.

Jumlah paralel membagi suku-suku menjadi potongan. Bila potongan ditentukan oleh banyaknya *thread*, seperti pada `chunked(xs, parts)`, jawabannya bergantung pada mesin. Obatnya adalah menetapkan potongan berdasarkan *data*, bukan perangkat keras: misalnya selalu 64 potongan, dijumlahkan berpasangan atau dengan Kahan, dan ke-64 jumlah parsial digabungkan dalam urutan tetap, berapa pun *thread* yang menghitungnya. Maka satu inti dan enam puluh empat inti memberi bit yang sama.

15.4 Program-Programnya

Program-program berada di `Chapter15/python` dan `Chapter15/java`. Masing-masing mencetak teks yang sama dalam kedua bahasa; argumen opsional `--time` menambahkan satu baris berisi waktu terukur, yang tentu saja berbeda dari mesin ke mesin dan dari jalan ke jalan.

15.4.1 Penjumlahan

Kode 15.1 menjumlahkan sepuluh juta salinan 0,1 dengan tujuh cara: dari kiri ke kanan, dalam 2, 4, 8, dan 16 potongan (seperti yang dilakukan jumlah paralel dengan *thread* sebanyak itu), berpasangan, dan dengan metode Kahan (**Pseudokode 15.1**). Di sini potongan-potongan dijumlahkan satu demi satu, sehingga programnya deterministik, tetapi aritmetikanya persis seperti pada jalan paralel.

Kode 15.1 `summation.py`: penjumlahan naif, berpotongan, berpasangan, dan Kahan.

```
"""Ten million sums of 0.1: why the order of additions, and so the number of
threads, changes the answer, and how to make it not matter.
Usage: python summation.py [n]   (default n = 10000000)"""
import sys


def naive(xs, lo, hi):
    s = 0.0
    for i in range(lo, hi):
        s += xs[i]
    return s


def kahan(xs, lo, hi):
    """Compensated summation: c carries the low-order bits that s loses."""
    s = c = 0.0
    for i in range(lo, hi):
        y = xs[i] - c
        t = s + y
        c = (t - s) - y
        s = t
    return s


def pairwise(xs, lo, hi):
    """Sum each half separately and add the two halves (error ~ log n)."""
    if hi - lo <= 8:
        return naive(xs, lo, hi)
    mid = (lo + hi) // 2
    return pairwise(xs, lo, mid) + pairwise(xs, mid, hi)


def chunked(xs, parts):
    """What a parallel sum does: each thread sums one chunk, then add them."""
    n, total = len(xs), 0.0
    for p in range(parts):
        total += naive(xs, p * n // parts, (p + 1) * n // parts)
    return total
```

Kode 15.1 (Lanjutan) *summation.py: penjumlahan naif, berpotongan, berpasangan, dan Kahan.*

```
def main():
    n = int(sys.argv[1]) if len(sys.argv) > 1 else 10_000_000
    xs = [0.1] * n
    exact = n / 10
    print(f"sum of {n} copies of 0.1; exact value {exact:.1f}")
    rows = [("naive, one thread", naive(xs, 0, n))]
    for parts in (2, 4, 8, 16):
        rows.append((f"naive, {parts} chunks", chunked(xs, parts)))
    rows += [("pairwise", pairwise(xs, 0, n)), ("Kahan", kahan(xs, 0, n))]
    for name, s in rows:
        print(f"{name:18s} {s:.10f}  error {s - exact:+.3e}")


if __name__ == "__main__":
    main()
```

Kode 15.2 adalah versi Java-nya; `Arrays.fill` membuat sepuluh juta salinan itu, dan batas-batas potongan dihitung dengan aritmetika `long` agar `p * n` tidak *overflow*.

Kode 15.2 `Summation.java`: penjumlahan yang sama dalam Java.

```java
// Ten million sums of 0.1: why the order of additions, and so the number of
// threads, changes the answer, and how to make it not matter.
// Usage: java Summation [n]   (default n = 10000000)
import java.util.Arrays;
import java.util.Locale;

public class Summation {
    static double naive(double[] xs, int lo, int hi) {
        double s = 0.0;
        for (int i = lo; i < hi; i++) s += xs[i];
        return s;
    }

    /** Compensated summation: c carries the low-order bits that s loses. */
    static double kahan(double[] xs, int lo, int hi) {
        double s = 0.0, c = 0.0;
        for (int i = lo; i < hi; i++) {
            double y = xs[i] - c;
            double t = s + y;
            c = (t - s) - y;
            s = t;
        }
        return s;
    }

    /** Sum each half separately and add the two halves (error ~ log n). */
    static double pairwise(double[] xs, int lo, int hi) {
        if (hi - lo <= 8) return naive(xs, lo, hi);
        int mid = (lo + hi) / 2;
        return pairwise(xs, lo, mid) + pairwise(xs, mid, hi);
    }

    /** What a parallel sum does: each thread sums one chunk, then add them. */
    static double chunked(double[] xs, int parts) {
        long n = xs.length;
        double total = 0.0;
        for (int p = 0; p < parts; p++)
            total += naive(xs, (int) (p * n / parts), (int) ((p + 1) * n / parts));
        return total;
    }
```

Kode 15.2 (Lanjutan) *Summation.java: penjumlahan yang sama dalam Java.*

```java
    public static void main(String[] args) {
        int n = args.length > 0 ? Integer.parseInt(args[0]) : 10_000_000;
        double[] xs = new double[n];
        Arrays.fill(xs, 0.1);
        double exact = n / 10.0;
        System.out.println(String.format(Locale.ROOT,
                "sum of %d copies of 0.1; exact value %.1f", n, exact));
        String[] names = {"naive, one thread", "naive, 2 chunks", "naive, 4 chunks",
                "naive, 8 chunks", "naive, 16 chunks", "pairwise", "Kahan"};
        double[] sums = {naive(xs, 0, n), chunked(xs, 2), chunked(xs, 4),
                chunked(xs, 8), chunked(xs, 16), pairwise(xs, 0, n),
                kahan(xs, 0, n)};
        for (int k = 0; k < names.length; k++)
            System.out.println(String.format(Locale.ROOT,
                    "%-18s %.10f  error %+.3e",
                    names[k], sums[k], sums[k] - exact));
    }
}
```

15.4.2 Wajan dengan gelung

Kode 15.3 adalah metode Jacobi dari Bab 5 yang ditulis ulang demi kecepatan dalam Python biasa: dua grid disimpan dan ditukar setelah setiap sapuan alih-alih disalin, dan baris di atas, pada, dan di bawah baris sekarang diambil sekali per baris. Program menjalankan banyak sapuan yang tetap, agar setiap implementasi dapat diukur waktunya pada kerja yang sama, dan melaporkan suhu di tengah, suhu rata-rata, dan perubahan terbesar pada sapuan terakhir.

Kode 15.3 `griddle_loops.py`: sapuan Jacobi pada wajan halus dengan gelung biasa.

```
"""Pak Joni's griddle (Chapter 5) on a fine grid, Jacobi with plain Python loops.
Usage: python griddle_loops.py [n] [sweeps] [--time]   (default 100 1000)"""
import sys
import time

LEFT, RIGHT, TOP, BOTTOM = 200.0, 60.0, 80.0, 150.0   # edge temperatures, C


def new_grid(n):
    u = [[0.0] * (n + 2) for _ in range(n + 2)]
    for i in range(1, n + 1):
        u[i][0], u[i][n + 1] = LEFT, RIGHT
        u[0][i], u[n + 1][i] = TOP, BOTTOM
    return u


def jacobi(u, n, sweeps):
    """Each sweep: every inner point becomes the mean of its OLD neighbors."""
    v = [row[:] for row in u]
    for _ in range(sweeps):
        for i in range(1, n + 1):
            up, row, down, out = u[i - 1], u[i], u[i + 1], v[i]
            for j in range(1, n + 1):
                out[j] = 0.25 * (up[j] + down[j] + row[j - 1] + row[j + 1])
        u, v = v, u                              # swap old and new
    return u, v


def report(u, v, n):
    inner = [u[i][j] for i in range(1, n + 1) for j in range(1, n + 1)]
    change = max(abs(u[i][j] - v[i][j])
                 for i in range(1, n + 1) for j in range(1, n + 1))
    c = n // 2
    print(f"center {u[c][c]:.6f} C, mean {sum(inner) / len(inner):.6f} C,"
          f" last change {change:.3e}")
```

Kode 15.3 (Lanjutan) *griddle_loops.py: sapuan Jacobi pada wajan halus dengan gelung biasa.*

```
def main():
    args = [a for a in sys.argv[1:] if a != "--time"]
    n = int(args[0]) if args else 100
    sweeps = int(args[1]) if len(args) > 1 else 1000
    print(f"grid {n} x {n}, {sweeps} Jacobi sweeps, 1 thread(s)")
    t0 = time.perf_counter()
    u, v = jacobi(new_grid(n), n, sweeps)
    t = time.perf_counter() - t0
    report(u, v, n)
    if "--time" in sys.argv:
        print(f"time {t:.3f} s, {n * n * sweeps / t / 1e6:.1f} million updates/s")


if __name__ == "__main__":
    main()
```

15.4.3 Tervektorisasi, dan dibagi ke beberapa thread

Kode 15.4 mengerjakan setiap sapuan dengan satu pernyataan NumPy dari Subbab 15.3.2, diterapkan pada satu blok baris. Dengan lebih dari satu *thread*, `ThreadPoolExecutor` memberi setiap *thread* satu jalur baris; NumPy melepas *global interpreter lock* Python selama ia menghitung pada larik, sehingga jalur-jalur itu benar-benar berjalan bersamaan. Karena Jacobi hanya membaca grid lama, hasilnya sama untuk berapa pun banyaknya *thread*.

Kode 15.4 `griddle_numpy.py`: sapuan Jacobi tervektorisasi, bila perlu dibagi ke beberapa *thread*.

```
"""The same Jacobi sweeps vectorized with NumPy, and split over threads.
Usage: python griddle_numpy.py [n] [sweeps] [threads] [--time]"""
import sys
import time
from concurrent.futures import ThreadPoolExecutor
import numpy as np
from griddle_loops import LEFT, RIGHT, TOP, BOTTOM


def new_grid(n):
    u = np.zeros((n + 2, n + 2))
    u[1:-1, 0], u[1:-1, -1], u[0, 1:-1], u[-1, 1:-1] = LEFT, RIGHT, TOP, BOTTOM
    return u


def sweep_rows(u, v, lo, hi):
    """Jacobi update of inner rows lo..hi-1: one line does a whole block."""
    v[lo:hi, 1:-1] = 0.25 * (u[lo - 1:hi - 1, 1:-1] + u[lo + 1:hi + 1, 1:-1]
                             + u[lo:hi, :-2] + u[lo:hi, 2:])


def jacobi(n, sweeps, threads=1):
    u = new_grid(n)
    v = u.copy()
    cuts = [1 + k * n // threads for k in range(threads + 1)]
    with ThreadPoolExecutor(max_workers=threads) as pool:
        for _ in range(sweeps):
            if threads == 1:
                sweep_rows(u, v, 1, n + 1)
            else:                           # every thread takes a strip of rows
                list(pool.map(lambda k: sweep_rows(u, v, cuts[k], cuts[k + 1]),
                              range(threads)))
            u, v = v, u
    return u, v
```

Kode 15.4 (Lanjutan) *griddle_numpy.py: sapuan Jacobi tervektorisasi, bila perlu dibagi ke beberapa thread.*

```
def main():
    args = [a for a in sys.argv[1:] if a != "--time"]
    n = int(args[0]) if args else 100
    sweeps = int(args[1]) if len(args) > 1 else 1000
    threads = int(args[2]) if len(args) > 2 else 1
    print(f"grid {n} x {n}, {sweeps} Jacobi sweeps, {threads} thread(s)")
    t0 = time.perf_counter()
    u, v = jacobi(n, sweeps, threads)
    t = time.perf_counter() - t0
    change = np.max(np.abs(u[1:-1, 1:-1] - v[1:-1, 1:-1]))
    c = n // 2
    print(f"center {u[c, c]:.6f} C, mean {np.mean(u[1:-1, 1:-1]):.6f} C,"
          f" last change {change:.3e}")
    if "--time" in sys.argv:
        print(f"time {t:.3f} s, {n * n * sweeps / t / 1e6:.1f} million updates/s")


if __name__ == "__main__":
    main()
```

Kode 15.5 adalah versi Java-nya. Satu kumpulan *thread* yang tetap dibuat sekali; pada setiap sapuan setiap *thread* menerima satu jalur baris sebagai tugas, dan `invokeAll` menunggu sampai semua jalur selesai sebelum grid ditukar. Penantian itu adalah biaya sinkronisasi dari Subbab 15.3.3, dibayar sekali per sapuan.

Kode 15.5 `Griddle.java`: sapuan Jacobi dalam Java pada satu atau beberapa *thread*.

```java
// Pak Joni's griddle (Chapter 5) on a fine grid: Jacobi sweeps, one thread or
// many. Usage: java Griddle [n] [sweeps] [threads] [--time]   (default 100 1000 1)
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Griddle {
    static final double LEFT = 200.0, RIGHT = 60.0, TOP = 80.0, BOTTOM = 150.0;

    static double[][] newGrid(int n) {
        double[][] u = new double[n + 2][n + 2];
        for (int i = 1; i <= n; i++) {
            u[i][0] = LEFT;
            u[i][n + 1] = RIGHT;
            u[0][i] = TOP;
            u[n + 1][i] = BOTTOM;
        }
        return u;
    }

    /** Jacobi update of inner rows lo..hi-1 from the old grid u into v. */
    static void sweepRows(double[][] u, double[][] v, int n, int lo, int hi) {
        for (int i = lo; i < hi; i++) {
            double[] up = u[i - 1], row = u[i], down = u[i + 1], out = v[i];
            for (int j = 1; j <= n; j++)
                out[j] = 0.25 * (up[j] + down[j] + row[j - 1] + row[j + 1]);
        }
    }

    public static void main(String[] args) throws Exception {
        List<String> a = new ArrayList<>(List.of(args));
        boolean timing = a.remove("--time");
        int n = a.size() > 0 ? Integer.parseInt(a.get(0)) : 100;
        int sweeps = a.size() > 1 ? Integer.parseInt(a.get(1)) : 1000;
        int threads = a.size() > 2 ? Integer.parseInt(a.get(2)) : 1;
        System.out.println(String.format(Locale.ROOT,
                "grid %d x %d, %d Jacobi sweeps, %d thread(s)",
                n, n, sweeps, threads));
        long t0 = System.nanoTime();
        double[][] u = newGrid(n), v = newGrid(n);
        ExecutorService pool = Executors.newFixedThreadPool(threads);
        for (int s = 0; s < sweeps; s++) {
            if (threads == 1) {
```

Kode 15.5 (Lanjutan) *Griddle.java: sapuan Jacobi dalam Java pada satu atau beberapa thread.*

```
                sweepRows(u, v, n, 1, n + 1);
            } else {                          // every thread takes a strip of rows
                List<Callable<Object>> jobs = new ArrayList<>();
                for (int k = 0; k < threads; k++) {
                    final double[][] uu = u, vv = v;
                    final int lo = 1 + k * n / threads;
                    final int hi = 1 + (k + 1) * n / threads;
                    jobs.add(() -> { sweepRows(uu, vv, n, lo, hi); return null; });
                }
                pool.invokeAll(jobs);         // waits until all strips are done
            }
            double[][] t = u; u = v; v = t;   // swap old and new
        }
        pool.shutdown();
        double t = (System.nanoTime() - t0) / 1e9;
        double sum = 0.0, change = 0.0;
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++) {
                sum += u[i][j];
                change = Math.max(change, Math.abs(u[i][j] - v[i][j]));
            }
        int c = n / 2;
        System.out.println(String.format(Locale.ROOT,
                "center %.6f C, mean %.6f C, last change %.3e",
                u[c][c], sum / (n * n), change));
        if (timing)
            System.out.println(String.format(Locale.ROOT,
                    "time %.3f s, %.1f million updates/s",
                    t, (double) n * n * sweeps / t / 1e6));
    }
}
```

15.4.4 Sapuan yang lebih sedikit: SOR merah-hitam

Kode 15.6 mengganti sapuan yang lebih cepat dengan sapuan yang lebih sedikit. Program membangun dua topeng warna sekali dan, pada setiap setengah sapuan, menghitung nilai SOR untuk semua titik dalam dengan satu ungkapan tervektorisasi lalu menyimpannya hanya di tempat topengnya bernilai benar. Ini mengerjakan dua kali aritmetika yang diperlukan (menghitung kedua warna dan menyimpan satu), pertukaran aritmetika dengan kesederhanaan yang disengaja dan dibahas lagi di Soal Latihan 15.5. Faktor relaksasinya adalah $\omega = 2/(1 + \sin(\pi/(n+1)))$ yang optimal dari Bab 5, dan iterasinya berhenti bila tidak ada titik yang bergeser sebesar 10^{-6} atau lebih.

Kode 15.6 `griddle_rb.py`: SOR merah-hitam dengan NumPy.

```
"""Fewer sweeps instead of faster sweeps: red-black SOR with NumPy.
Usage: python griddle_rb.py [n] [--time]   (default n = 100)"""
import math
import sys
import time
import numpy as np
from griddle_numpy import new_grid


def red_black_sor(n, omega, tol=1e-6, max_sweeps=100000):
    """Points with i + j even ('red') depend only on 'black' points and vice
    versa, so each half-sweep is one vectorized (and parallel) step."""
    u = new_grid(n)
    i, j = np.meshgrid(np.arange(n + 2), np.arange(n + 2), indexing="ij")
    colors = [((i + j) % 2 == c)[1:-1, 1:-1] for c in (0, 1)]
    for k in range(1, max_sweeps + 1):
        change = 0.0
        for mask in colors:
            inner = u[1:-1, 1:-1]
            gs = 0.25 * (u[:-2, 1:-1] + u[2:, 1:-1] + u[1:-1, :-2] + u[1:-1, 2:])
            new = inner + omega * (gs - inner)
            change = max(change, float(np.max(np.abs(new - inner)[mask])))
            inner[mask] = new[mask]
        if change < tol:
            return u, k
    return u, max_sweeps


def main():
    args = [a for a in sys.argv[1:] if a != "--time"]
    n = int(args[0]) if args else 100
    omega = 2.0 / (1.0 + math.sin(math.pi / (n + 1)))    # Chapter 5
    t0 = time.perf_counter()
    u, k = red_black_sor(n, omega)
    t = time.perf_counter() - t0
    c = n // 2
    print(f"grid {n} x {n}, red-black SOR, omega {omega:.6f}: {k} sweeps")
    print(f"center {u[c, c]:.6f} C, mean {np.mean(u[1:-1, 1:-1]):.6f} C")
    if "--time" in sys.argv:
        print(f"time {t:.3f} s")


if __name__ == "__main__":
    main()
```

Kode 15.7 melakukan hal yang sama dalam Java, dengan hanya mengunjungi titik-titik berwarna yang sedang diproses (`j` melangkah 2 demi 2 dari titik pertama warna itu dalam baris). Setiap *thread* mengembalikan perubahan terbesar di jalurnya, dan *thread* utama mengambil maksimumnya. Maksimum sekumpulan bilangan, tidak seperti jumlahnya, tidak bergantung pada urutan, sehingga hasilnya lagi-lagi identik untuk berapa pun banyaknya *thread*.

Kode 15.7 `GriddleRb.java`: SOR merah-hitam dalam Java pada satu atau beberapa *thread*.

```java
// Fewer sweeps instead of faster sweeps: red-black SOR, one thread or many.
// Usage: java GriddleRb [n] [threads] [--time]   (default 100 1)
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class GriddleRb {
    /** Update the points of one color in rows lo..hi-1; return the largest move.
     *  A red point (i + j even) reads only black neighbors, and vice versa. */
    static double half(double[][] u, int n, double omega, int color,
                       int lo, int hi) {
        double change = 0.0;
        for (int i = lo; i < hi; i++) {
            int j0 = ((i + color) % 2 == 0) ? 2 : 1;   // first j: (i+j)%2 == color
            for (int j = j0; j <= n; j += 2) {
                double gs = 0.25 * (u[i - 1][j] + u[i + 1][j]
                        + u[i][j - 1] + u[i][j + 1]);
                double nu = u[i][j] + omega * (gs - u[i][j]);
                change = Math.max(change, Math.abs(nu - u[i][j]));
                u[i][j] = nu;
            }
        }
        return change;
    }

    public static void main(String[] args) throws Exception {
        List<String> a = new ArrayList<>(List.of(args));
        boolean timing = a.remove("--time");
        int n = a.size() > 0 ? Integer.parseInt(a.get(0)) : 100;
        int threads = a.size() > 1 ? Integer.parseInt(a.get(1)) : 1;
        double omega = 2.0 / (1.0 + Math.sin(Math.PI / (n + 1)));   // Chapter 5
        long t0 = System.nanoTime();
        double[][] u = Griddle.newGrid(n);
        ExecutorService pool = Executors.newFixedThreadPool(threads);
        int k = 0;
        while (k < 100000) {
            k++;
            double change = 0.0;
            for (int color = 0; color < 2; color++) {
                List<Callable<Double>> jobs = new ArrayList<>();
                for (int p = 0; p < threads; p++) {
                    final int lo = 1 + p * n / threads;
```

Kode 15.7 (Lanjutan) *GriddleRb.java: SOR merah-hitam dalam Java pada satu atau beberapa thread.*

```
                    final int hi = 1 + (p + 1) * n / threads;
                    final int col = color;
                    jobs.add(() -> half(u, n, omega, col, lo, hi));
                }
                for (Future<Double> f : pool.invokeAll(jobs))
                    change = Math.max(change, f.get());
            }
            if (change < 1e-6) break;
        }
        pool.shutdown();
        double t = (System.nanoTime() - t0) / 1e9, sum = 0.0;
        for (int i = 1; i <= n; i++)
            for (int j = 1; j <= n; j++) sum += u[i][j];
        int c = n / 2;
        System.out.println(String.format(Locale.ROOT,
                "grid %d x %d, red-black SOR, omega %.6f: %d sweeps",
                n, n, omega, k));
        System.out.println(String.format(Locale.ROOT, "center %.6f C, mean %.6f C",
                u[c][c], sum / (n * n)));
        if (timing)
            System.out.println(String.format(Locale.ROOT, "time %.3f s", t));
    }
}
```

15.5 Menjalankan Program

```
cd Chapter15/python
python3 summation.py
python3 griddle_loops.py
python3 griddle_numpy.py
python3 griddle_rb.py
cd ../java
javac -d out *.java
java -cp out Summation
java -cp out Griddle
java -cp out GriddleRb
```

Program penjumlahan mencetak:

```
sum of 10000000 copies of 0.1; exact value 1000000.0
naive, one thread  999999.9998389754  error -1.610e-04
naive, 2 chunks    999999.9999107814  error -8.922e-05
naive, 4 chunks    1000000.0000402561  error +4.026e-05
naive, 8 chunks    1000000.0000223045  error +2.230e-05
naive, 16 chunks   999999.9999899360  error -1.006e-05
pairwise           1000000.0000000000  error +0.000e+00
Kahan              1000000.0000000000  error +0.000e+00
```

Penjumlahan dari kiri ke kanan meleset 1.6×10^{-4}, dan setiap pemotongan memberi jawaban salah yang berbeda, kadang terlalu kecil dan kadang terlalu besar. Program yang menghitung jumlah ini dengan "*thread* sebanyak inti yang ada" akan mencetak hasil yang berbeda pada laptop dengan 4 inti dan server dengan 16. Penjumlahan berpasangan dan Kahan sama-sama benar pada semua digit yang dicetak, dan keduanya akan tetap benar bagaimanapun suku-suku dibagi, asalkan pembagiannya tetap.

Program-program wajan, dengan grid bawaan 100×100 dan 1000 sapuan, semuanya mencetak tiga baris yang sama, bahasa apa pun dan berapa pun *thread*-nya:

```
grid 100 x 100, 1000 Jacobi sweeps, 1 thread(s)
center 11.559613 C, mean 70.261992 C, last change 6.405e-02
```

Bagian tengah pelat masih bersuhu 11,6 derajat setelah 1000 sapuan, dan sapuan terakhir masih mengubah sebuah titik sebesar 0,064: Jacobi baru saja mulai. Menurut Persamaan (15.3), mencapai perubahan 10^{-6} memerlukan sekitar 24.000 sapuan pada ukuran ini. SOR merah-hitam sampai di sana dalam 316:

```
grid 100 x 100, red-black SOR, omega 1.939676: 316 sweeps
center 122.789173 C, mean 122.499995 C
```

Rata-ratanya, 122,5 derajat, adalah rata-rata keempat suhu tepi, sebagaimana dituntut oleh simetri masalahnya; suhu tengahnya, 122,8, dekat dengannya.

Dengan `--time` dan grid halus 400×400, program-program melaporkan kecepatannya. Pada mesin uji dua inti (Intel Xeon 2,1 GHz, Python 3.13 dengan NumPy 2.5.3, Java 21), program Python mencetak:

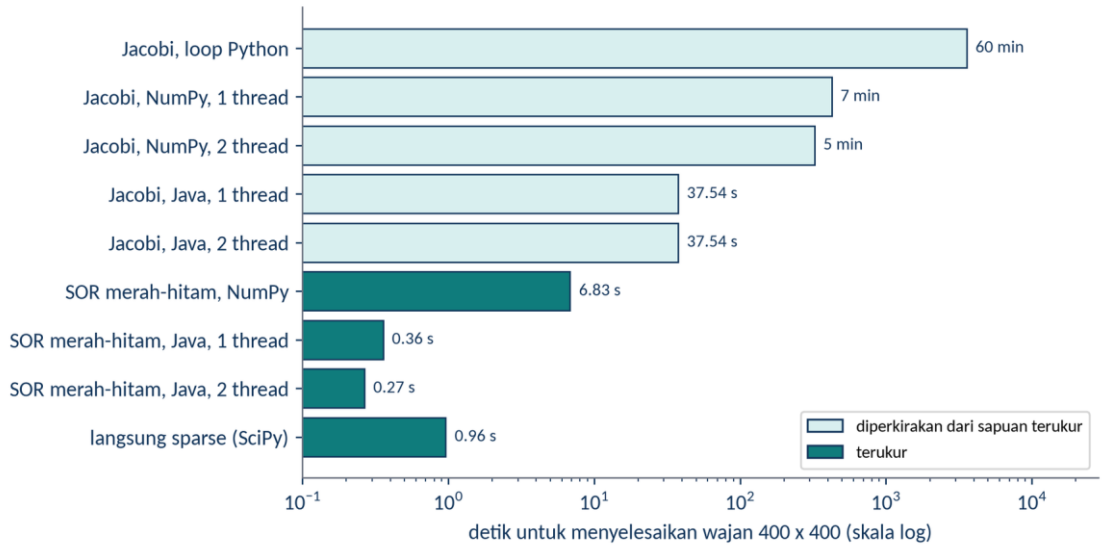
```
grid 400 x 400, 200 Jacobi sweeps, 1 thread(s)
center 0.000000 C, mean 9.008161 C, last change 3.172e-01
time 2.050 s, 15.6 million updates/s
grid 400 x 400, 2000 Jacobi sweeps, 1 thread(s)
center 0.000000 C, mean 28.428053 C, last change 3.168e-02
time 2.404 s, 133.1 million updates/s
grid 400 x 400, 2000 Jacobi sweeps, 2 thread(s)
center 0.000000 C, mean 28.428053 C, last change 3.168e-02
time 1.766 s, 181.2 million updates/s
```

dan program Java:

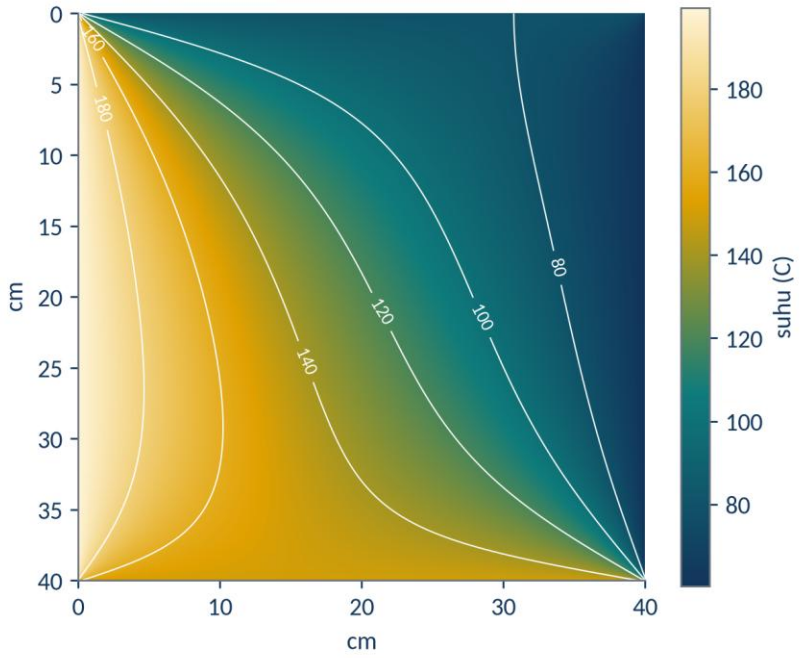
```
grid 400 x 400, 2000 Jacobi sweeps, 1 thread(s)
center 0.000000 C, mean 28.428053 C, last change 3.168e-02
time 0.236 s, 1356.4 million updates/s
grid 400 x 400, 2000 Jacobi sweeps, 2 thread(s)
center 0.000000 C, mean 28.428053 C, last change 3.168e-02
time 0.247 s, 1295.8 million updates/s
grid 400 x 400, red-black SOR, omega 1.984453: 1171 sweeps
center 122.572805 C, mean 122.499983 C
time 0.362 s
grid 400 x 400, red-black SOR, omega 1.984453: 1171 sweeps
center 122.572805 C, mean 122.499983 C
time 0.305 s
```

Bilangan-bilangan itu memperlihatkan setiap gagasan dari Subbab 15.3. Python biasa membuat 15,6 juta pembaruan per detik; NumPy, 133 juta pada satu *thread* dan 181 juta pada dua; Java, 1,36 miliar pada satu *thread* dan tidak lebih pada dua, karena pada laju itu gelungnya dibatasi oleh memori (Persamaan (15.1)), bukan oleh aritmetika. NumPy, yang berjalan jauh di bawah batas memori, masih punya ruang untuk memperoleh manfaat dari inti kedua; Java, yang sudah berada di batas itu, tidak. SOR merah-hitam dalam Java konvergen dalam sekitar sepertiga detik, dengan sedikit keuntungan dari *thread* kedua. Pada mesin dengan inti dan saluran memori yang lebih banyak, keuntungan paralelnya lebih besar; peringkat metodenya tidak berubah.

Gambar 15.1 menggabungkan pengukuran ini menjadi waktu untuk *menyelesaikan* wajan halus. Batang Jacobi mengalikan waktu terukur per sapuan dengan 340.000 sapuan dari Subbab 15.3.4 (Jacobi tidak dijalankan sampai selesai; satu jam Python biasa tidak membuktikan hal baru), sedangkan batang SOR dan pemecah langsung adalah jalan yang terukur. Dari yang paling lambat sampai yang paling cepat ada faktor lebih dari sepuluh ribu, dan faktor tunggal terbesar, beberapa ratus, datang dari algoritma, bukan dari bahasa atau dari inti. **Gambar 15.2** memperlihatkan jawaban yang diminta si pandai besi.



Gambar 15.1 Waktu untuk menyelesaikan wajan 400 × 400 pada mesin uji dua inti: Jacobi (diperkirakan dari waktu terukur per sapuan dan banyaknya sapuan yang diperlukan), SOR merah-hitam, dan pemecah langsung *sparse* dari SciPy (terukur). Skalanya logaritmik.



Gambar 15.2 Suhu tunak wajan 40 cm pada grid 1 mm, dengan tepi kiri 200, kanan 60, atas 80, dan bawah 150 derajat Celsius; garis putih adalah isoterm.

15.6 Membandingkan dengan NumPy dan SciPy

Kode 15.8 memeriksa jumlahnya dengan `math.fsum` dari Python, yang mengembalikan jumlah nilai-nilai eksak yang dibulatkan dengan benar [13], dan dengan `sum` berpasangan dari NumPy, lalu menyelesaikan wajan sebagai sistem linear *sparse* dari Bab 5, $A\mathbf{u} = \mathbf{b}$, dengan pemecah langsung *sparse* dari SciPy [14]. Matriksnya dibangun dengan `kronsum`, yang membentuk operator lima titik 2-D dari matriks selisih kedua 1-D tridiag$(-1,2,-1)$.

Kode 15.8 `check_numpy.py`: jumlah eksak dan berpasangan, serta wajan dengan pemecah langsung *sparse*.

```
"""Library answers for Chapter 15: exact and pairwise sums, and the griddle
solved directly by a sparse factorization.   Usage: python check_numpy.py [n]"""
import math
import sys
import time
import numpy as np
import scipy.sparse as sp
from scipy.sparse.linalg import spsolve
from griddle_loops import LEFT, RIGHT, TOP, BOTTOM

xs = [0.1] * 10_000_000
print(f"math.fsum (correctly rounded): {math.fsum(xs):.10f}")
print(f"numpy.sum (pairwise)         : {np.sum(np.array(xs)):.10f}")

n = int(sys.argv[1]) if len(sys.argv) > 1 else 400
t0 = time.perf_counter()
T = sp.diags([-1.0, 2.0, -1.0], [-1, 0, 1], shape=(n, n))
A = sp.kronsum(T, T, format="csc")        # 4 on the diagonal, -1 for neighbors
b = np.zeros((n, n))
b[:, 0] += LEFT
b[:, -1] += RIGHT
b[0, :] += TOP
b[-1, :] += BOTTOM
u = spsolve(A, b.ravel()).reshape(n, n)
t = time.perf_counter() - t0
c = n // 2 - 1                            # inner point (c+1, c+1) of the grid
print(f"sparse direct solve, {n * n} unknowns: center {u[c, c]:.6f} C,"
      f" mean {u.mean():.6f} C, {t:.2f} s")
```

```
math.fsum (correctly rounded): 1000000.0000000000
numpy.sum (pairwise)         : 1000000.0000000000
sparse direct solve, 160000 unknowns: center 122.572847 C, mean 122.500000 C, 0.87 s
```

Kedua jumlah pustaka sesuai dengan hasil berpasangan dan Kahan kita. Penyelesaian langsung, dalam 0,87 detik, memberi suhu tengah 122,572847; SOR merah-hitam pada grid yang sama berhenti di 122,572805:

```
grid 400 x 400, red-black SOR, omega 1.984453: 1171 sweeps
center 122.572805 C, mean 122.499983 C
```

Keduanya berbeda 4×10^{-5} walaupun SOR berhenti ketika tidak ada titik yang bergeser 10^{-6}. Itulah peringatan Bab 5: perubahan kecil per sapuan bukanlah galat yang kecil bila iterasinya konvergen lambat, karena galat yang tersisa kira-kira sebesar perubahan dibagi $1 - \rho$. Untuk wajan, 4×10^{-5} derajat tidak berarti apa-apa; untuk masalah yang memerlukan digit-digit itu, perketat toleransinya atau perkirakan galatnya dari ρ.

15.7 Membuatnya Lebih Baik

- **Ukur sebelum mengoptimalkan.** *Profiler* (`cProfile` dalam Python, Java Flight Recorder dalam JDK) memperlihatkan ke mana waktu sebenarnya pergi; jarang di tempat yang diduga. Ukur waktu sebuah jalan beberapa kali dan laporkan yang terbaik atau mediannya; jalan Java yang pertama termasuk kompilasi, dan program lain di mesin yang sama mengganggu setiap jalan.
- **Pakai proses atau Python *free-threaded* untuk kode Python yang terikat CPU.** *Thread* Python menjalankan kode Python murni bergantian karena *global interpreter lock*; *thread* hanya membantu bila kerjanya terjadi di dalam NumPy atau menunggu masukan dan keluaran. Untuk kerja Python murni, `concurrent.futures.ProcessPoolExecutor` menjalankan proses-proses terpisah, dan *build free-threaded* Python 3.14 yang kini didukung resmi menghapus kunci itu, dengan harga pada kecepatan satu *thread* sekitar 1% di macOS aarch64 sampai 8% di Linux x86-64, menurut dokumentasi Python [15].

- **Pindah ke algoritma yang lebih baik sebelum ke mesin yang lebih besar.** Untuk masalah eliptik seperti wajan, SOR, gradien konjugat dengan prapengondisi, *multigrid* [8], atau pemecah langsung *sparse* [9] masing-masing mengalahkan Jacobi berlipat-lipat. Paralelkan algoritma yang baik, bukan yang buruk.
- **Jaga agar hasil dapat diulang.** Tetapkan pemotongan setiap reduksi berdasarkan data, bukan banyaknya *thread*; beri benih setiap deret bilangan acak dari tugasnya, bukan dari *thread*-nya (Bab 16 melakukan ini untuk jalan-jalan Monte Carlo-nya); dan simpan satu jalan acuan dengan satu *thread* untuk pembanding.
- **Pertimbangkan prosesor grafis bila datanya besar dan teratur.** GPU mempunyai ribuan inti sederhana dan lebar pita memori yang sangat tinggi, dan sapuan *stencil* seperti Jacobi berjalan di atasnya dengan sedikit perubahan; pustaka larik dengan antarmuka mirip NumPy tersedia untuk GPU. Gagasan-gagasan bab ini, *roofline*, hukum Amdahl, dan reduksi yang dapat diulang, berlaku di sana tanpa perubahan.

15.8 Memakainya Sungguhan

Kebanyakan pembaca tidak akan menyelesaikan persamaan panas untuk pandai besi, tetapi setiap orang yang berhitung dengan komputer pada akhirnya menunggu sebuah program. Daftar periksa yang sama berlaku untuk makro lembar kerja yang menghitung ulang jadwal angsuran sebuah koperasi, skrip yang menyimulasikan seribu musim sebuah kebun, atau pelatihan sebuah model.

- **Perkirakan sebelum menjalankan.** Hitung banyaknya bilangan tak diketahui, operasi per bilangan tak diketahui, dan banyaknya iterasi, seperti yang dilakukan Subbab 15.3.4. Bila perkiraannya berjam-jam, penyetelan apa pun tidak akan membuatnya menjadi hitungan detik; carilah metode yang lebih baik lebih dahulu.
- **Ukur waktu sebenarnya**, dengan baris semacam `--time` atau dengan *profiler*, pada masalah yang cukup kecil untuk selesai dan cukup besar untuk realistis, lalu ekstrapolasikan dengan perkiraan tadi.
- **Vektorkan gelung terdalam** dalam Python dengan NumPy, atau pindahkan ke bahasa terkompilasi. Pertahankan urutan aritmetikanya bila Anda menginginkan hasil yang identik.
- **Paralelkan tugas-tugas yang saling bebas lebih dahulu.** Dua puluh rancangan wajan si pandai besi adalah dua puluh masalah terpisah: tidak berbagi apa pun, tidak perlu sinkronisasi, dan terbagi sempurna di antara inti, bentuk paralelisme yang paling mudah. Pada mesin uji dua inti, dua puluh penyelesaian SOR merah-hitam dalam Java memerlukan sekitar $20 \times 0.35 = 7$ detik pada satu inti dan separuhnya pada dua; dalam NumPy sekitar $20 \times 6.8 = 136$ detik pada satu inti, dan kira-kira separuhnya dengan dua proses `ProcessPoolExecutor`. Memparalelkan *di dalam* satu penyelesaian, seperti di Subbab 15.4.3, lebih sulit dan hasilnya lebih kecil.
- **Simpan jawaban acuan.** Jalankan kasus kecil dengan satu *thread* dan simpan keluarannya; setelah setiap optimasi, versi cepat harus mencetak digit yang sama (atau hanya berbeda pada tempat-tempat terakhir, dengan alasan yang diketahui seperti urutan penjumlahan yang berbeda).

Bagi koperasi simpan pinjam yang setiap bulan menghitung ulang jadwal angsuran lima ribu anggotanya, penalaran yang sama memberi rencana yang jelas. Jadwal setiap anggota saling bebas, sehingga anggota dapat dibagi ke beberapa inti tanpa interaksi. Setiap jadwal adalah gelung pendek berisi perkalian dan penjumlahan, yang murah dalam bahasa terkompilasi maupun dalam NumPy tervektorisasi. Dan total bulanan yang masuk ke pembukuan harus dijumlahkan secara dapat diulang, dalam urutan tetap atau dengan metode Kahan, agar laporan

yang dicetak di laptop bendahara sama dengan yang dicetak di komputer kantor sampai rupiah terakhir.

15.9 Metode Numerik dalam Praktik: Profitina

Platform AI Visibility Profitina adalah *pipeline* pengukuran atas situs-situs web, dan gagasan-gagasan bab ini menentukan seberapa cepat ia dapat menjawab [16]. Memindai sebuah situs sebagian besar adalah *menunggu*, yaitu menunggu halaman tiba melalui jaringan, sehingga banyak pemindaian dapat berlangsung bersamaan bahkan dengan sedikit inti; itu konkurensi yang dibatasi masukan dan keluaran, bukan aritmetika, dan analisis Persamaan (15.1) digantikan pertanyaan berapa banyak permintaan yang wajar dikirim ke sebuah situs sekaligus. Menilai halaman yang telah diunduh dan menjalankan model lokal Qwen2.5 adalah *komputasi*, yang dibatasi prosesor dan memori, tempat hukum Amdahl dan *roofline* berlaku: kecepatan model di server sebagian besar ditentukan oleh seberapa cepat bobotnya dapat dibaca dari memori, dan itulah sebabnya model yang sangat kecil (qwen3:1.7b) dipakai sebagai jalur cepat untuk tugas klasifikasi singkat.

Gerbang rilis platform ini adalah beban kerja paralel jenis lain: seluruh rangkaian uji otomatis, jauh lebih dari seribu uji Python dan JavaScript, harus lulus sebelum rilis [16]. Uji-uji itu saling bebas, seperti musim-musim Monte Carlo di Bab 16, sehingga dapat disebar ke banyak inti dengan percepatan yang hampir sempurna, asalkan tidak ada dua uji yang berbagi keadaan, padanan perangkat lunak dari *race condition* di Subbab 15.3.3. Dan tren skor mingguan pada dasbor historis adalah jumlah atas tabel yang terus bertambah, yang harus keluar sama apa pun urutan barisnya: penjumlahan yang dapat diulang dari Subbab 15.3.5.

15.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Mengoptimalkan hal yang salah.** Membuat sapuan Jacobi sepuluh kali lebih cepat tetap membuatnya ratusan kali lebih lambat daripada SOR. Hitung sapuan (atau iterasi) sebelum menghitung nanodetik.
- **Mengukur waktu sekali saja.** Satu jalan mencakup waktu mulai, kompilasi, dan apa pun yang sedang dikerjakan mesin. Ulangi, ambil yang terbaik atau mediannya, dan sebutkan mesinnya.
- **Mengharapkan p inti memberi kecepatan p kali.** Hukum Amdahl Persamaan (15.2), sinkronisasi, dan lebar pita memori (Persamaan (15.1)) semuanya mengambil bagian; pada kerja yang terikat memori, dua inti bisa tidak lebih cepat daripada satu.
- **Memparalelkan pembaruan di tempat.** Gauss–Seidel atau SOR yang dibagi begitu saja ke beberapa *thread* mengandung *race condition* dan memberi hasil yang bergantung pada pewaktuan. Pakai Jacobi, urutan merah-hitam, atau beri setiap *thread* keluarannya sendiri.
- **Membiarkan banyaknya thread menentukan sebuah jumlah.** Jumlah berpotongan berubah menurut banyaknya potongan. Tetapkan potongan berdasarkan data dan pakai penjumlahan berpasangan atau Kahan di dalamnya.
- **Memakai thread Python untuk aritmetika Python murni.** Dengan *global interpreter lock* mereka bergiliran; pakai NumPy, proses, atau *build free-threaded*.
- **"Menyederhanakan" algoritma Kahan.** $(t - s) - y$ tidak bernilai nol dalam *floating point*; ia adalah bagian yang hilang. Pertahankan rumusnya seperti tertulis dan jangan mengompilasi versi C dengan opsi yang mengizinkan penataan ulang asosiatif.

15.11 Ringkasan Bab

- Waktu dihabiskan untuk aritmetika, pemindahan data, dan pembukuan; Persamaan (15.1) membatasi waktu dengan banyaknya operasi dibagi laju aritmetika dan banyaknya bait dibagi lebar pita. Jacobi, dengan 0,25 operasi per bait, terikat memori.
- Vektorisasi dengan NumPy mengeluarkan *interpreter* dari gelung terdalam (15,6 menjadi 133 juta pembaruan per detik), dan Java yang terkompilasi mencapai 1,36 miliar, batas memori mesin uji; ketiganya memberi digit yang identik karena melakukan operasi yang sama dalam urutan yang sama.
- Hukum Amdahl Persamaan (15.2) membatasi percepatan paralel dengan bagian serialnya; sinkronisasi, beban tambahan, dan lebar pita memori menguranginya lagi. Jacobi aman dibagi menjadi jalur-jalur; Gauss–Seidel memerlukan urutan merah-hitam.
- Jacobi memerlukan sekitar $2(n+1)^2 \ln(1/\varepsilon)/\pi^2$ sapuan (Persamaan (15.3)) dan SOR sekitar $(n+1)\ln(1/\varepsilon)/(2\pi)$ (Persamaan (15.4)); untuk wajan 400×400 itu 340.000 berbanding 1171 sapuan, dan algoritma yang lebih baik jauh mengalahkan bahasa yang lebih cepat dan inti tambahan.
- Jumlah *floating point* bergantung pada urutan; penjumlahan naif meleset paling banyak $(n-1)u\sum|x_i|$ (Persamaan (15.5)), berpasangan $\lceil \log_2 n \rceil u\sum|x_i|$ (Persamaan (15.6)), dan metode Kahan (**Pseudokode 15.1**) sekitar $2u\sum|x_i|$. Potongan yang ditetapkan berdasarkan data membuat jumlah paralel dapat diulang.

15.12 Soal Latihan

1. Sebuah program menghabiskan 90% waktunya dalam gelung yang dapat diparalelkan sempurna. Berapa percepatan yang diramalkan hukum Amdahl pada 4, 16, dan tak hingga banyaknya inti? Berapa bagian yang harus paralel agar 16 inti memberi percepatan 12?
2. Untuk pembaruan SOR merah-hitam $u \leftarrow u + \omega(1/4\,(u_N + u_S + u_W + u_E) - u)$, hitung banyaknya operasi serta bait yang dibaca dan ditulis per titik dalam kasus terbaik, lalu hitung intensitas aritmetikanya. Apakah SOR merah-hitam terikat memori pada mesin di Subbab 15.3.1?
3. Jelaskan dengan grid titik dalam 3×3 mengapa membagi Gauss–Seidel menjadi dua jalur baris dapat memberi hasil berbeda bergantung pada *thread* mana yang berjalan lebih dulu, dan mengapa urutan merah-hitam tidak.
4. Dengan tangan, jumlahkan $x = (1, 10^{-16}, 10^{-16}, \ldots, 10^{-16})$ dengan sepuluh salinan 10^{-16}, mula-mula dari kiri ke kanan lalu dengan metode Kahan, dalam presisi ganda. Berapa jumlah eksaknya dan metode mana yang memperolehnya?
5. `griddle_rb.py` menghitung nilai SOR untuk semua titik pada setiap setengah sapuan dan menyimpan separuhnya. Perkirakan berapa waktu yang terbuang, dan jelaskan bagaimana irisan berlangkah seperti `u[1:-1:2, 1:-1:2]` dapat memperbarui satu warna saja.
6. Ubah `chunked` menjadi fungsi `reproducible_sum(xs, threads)` yang selalu memberi jawaban yang sama untuk berapa pun banyaknya *thread*. Jelaskan rancangan Anda.

15.13 Penyelesaian Soal Latihan

1. Dengan $f = 0.9$: $S(4) = 1/(0.1 + 0.225) = 3.08$; $S(16) = 1/(0.1 + 0.05625) = 6.40$; $S(\infty) = 1/0.1 = 10$. Agar $S(16) = 12$ kita perlu $(1-f) + f/16 = 1/12$, yaitu $1 - \frac{15}{16}f = 0.08333$,

sehingga $f = 0.9778$: lebih dari 97,8% waktunya harus paralel. Karena alasan inilah percepatan dua belas kali pada enam belas inti jarang terjadi.

2. Per titik: rata-rata empat tetangga memerlukan 3 penjumlahan dan 1 perkalian, lalu satu pengurangan, satu perkalian dengan ω, dan satu penjumlahan, jadi 7 operasi (ditambah perbandingan untuk perubahannya). Dalam kasus terbaik setiap titik grid dibaca sekali dan ditulis sekali per setengah sapuan yang memperbaruinya, dan setiap setengah sapuan membaca seluruh grid: sekitar 16 bait dibaca dan 8 bait ditulis per titik yang diperbarui, kira-kira 24 bait. Intensitasnya $7/24 \approx 0.3$ operasi per bait, di bawah $F/B \approx 0.5$ untuk satu inti: SOR merah-hitam juga terikat memori, dan itulah sebabnya *thread* keduanya hanya memberi sedikit keuntungan. Keunggulannya bukan sapuan yang lebih cepat, melainkan sapuan yang 290 kali lebih sedikit.

3. Nomori titik-titik dalam baris demi baris, $1, \ldots, 9$, dan berikan baris 1–2 kepada *thread* A dan baris 3 kepada *thread* B. Gauss–Seidel memperbarui titik 7 (baris 3) dengan memakai titik 4 (baris 2). Jika B berjalan sebelum A memperbarui titik 4, B memakai nilai lama; jika sesudahnya, nilai baru. Kedua urutan memberi nilai berbeda di titik 7, dan semua yang dihitung darinya ikut berbeda: hasilnya bergantung pada pewaktuan. Dalam urutan merah-hitam, setiap tetangga sebuah titik berwarna lain, dan selama setengah sapuan merah hanya titik merah yang berubah; setiap titik merah hanya membaca nilai hitam, yang tetap selama setengah sapuan itu. Urutan pembaruan titik merah, dan karena itu pembagiannya ke *thread-thread*, tidak mungkin berpengaruh.

4. Dari kiri ke kanan: $1 + 10^{-16}$ dibulatkan menjadi 1, karena 10^{-16} kurang dari setengah jarak 2.22×10^{-16} antar-*double* tepat di atas 1; hal yang sama terjadi sepuluh kali, dan hasilnya tepat 1. Kahan: pada suku kecil pertama $y = 10^{-16}$, $t = 1$, dan $c = (1 - 1) - 10^{-16} = -10^{-16}$, seluruh suku dicatat sebagai hilang. Pada suku kedua, $y = 10^{-16} - (-10^{-16}) = 2 \times 10^{-16}$, dan kini $t = 1 + 2 \times 10^{-16}$ dibulatkan naik ke *double* berikutnya, $1 + 2.22 \times 10^{-16}$, sehingga $c = 2.22 \times 10^{-16} - 2 \times 10^{-16} = 2.2 \times 10^{-17}$: kali ini yang ditambahkan sedikit terlalu banyak, dan suku berikutnya dikurangi sepadan. Terus demikian, jumlahnya naik satu jarak kira-kira setiap dua suku dan berakhir di $1 + 5 \times 2.22 \times 10^{-16} = 1 + 1.11 \times 10^{-15}$, yang dicetak sebagai 1.000000000000001, *double* terdekat dengan jumlah eksak $1 + 10^{-15}$. `math.fsum` mengembalikan nilai yang sama, sedangkan jumlah naif mengembalikan 1.

5. Setiap setengah sapuan menghitung ungkapan SOR untuk semua n^2 titik alih-alih $n^2/2$ titik satu warna, sehingga aritmetikanya dua kali lipat; karena komputasinya terikat memori, waktunya naik kurang dari dua kali tetapi terasa (penugasan bertopeng juga memerlukan satu lintasan). Titik merah dengan $i + j$ genap membentuk dua sub-kisi yang teratur: i ganjil dan j ganjil, serta i genap dan j genap. Dengan `r = u[1:-1, 1:-1]`, sub-kisi pertama adalah `r[0::2, 0::2]` dan tetangga utaranya `u[0:-2:2, 1:-1:2]`, tetangga selatannya `u[2::2, 1:-1:2]`, dan seterusnya; memperbarui kedua sub-kisi dengan *view* berlangkah seperti itu menyentuh setiap titik merah tepat sekali. Pengindeksan yang sama berlaku untuk hitam. Beginilah kode produksi memvektorkan metode merah-hitam.

6. Pilih batas-batas potongan hanya dari panjang data, misalnya $K = 64$ potongan tetap dengan batas $\lfloor kn/K \rfloor$. Setiap potongan dijumlahkan dengan metode Kahan (atau berpasangan), menghasilkan 64 jumlah parsial; *thread-thread* boleh menghitungnya dalam urutan apa pun dan dalam jumlah berapa pun, karena setiap jumlah parsial hanya bergantung pada potongannya sendiri. Akhirnya ke-64 jumlah parsial dijumlahkan dalam urutan potongan, secara berpasangan. Jawabannya kemudian hanya fungsi dari data, identik pada 1 *thread* maupun 64, dan teliti sampai beberapa satuan pada tempat desimal terakhir.

Daftar Pustaka

[1] S. Williams, A. Waterman, and D. Patterson, "Roofline: An insightful visual performance model for multicore architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[2] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, 2020.

[3] OpenJDK, "JEP 508: Vector API (Tenth Incubator)," JDK 25, 2025; and "JEP 529: Vector API (Eleventh Incubator)," JDK 26, 2026. [Online]. Available: https://openjdk.org/jeps/508

[4] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proc. AFIPS Spring Joint Computer Conference*, 1967, pp. 483–485.

[5] J. L. Gustafson, "Reevaluating Amdahl's law," *Communications of the ACM*, vol. 31, no. 5, pp. 532–533, 1988.

[6] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. Philadelphia, PA, USA: SIAM, 2003.

[7] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[8] W. L. Briggs, V. E. Henson, and S. F. McCormick, *A Multigrid Tutorial*, 2nd ed. Philadelphia, PA, USA: SIAM, 2000.

[9] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, 1973.

[10] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Philadelphia, PA, USA: SIAM, 2002.

[11] Python Software Foundation, "Built-in Functions: sum," Python 3.14 documentation, 2026. [Online]. Available: https://docs.python.org/3/library/functions.html#sum

[12] W. Kahan, "Further remarks on reducing truncation errors," *Communications of the ACM*, vol. 8, no. 1, p. 40, 1965.

[13] Python Software Foundation, "math — Mathematical functions: math.fsum," Python 3.14 documentation, 2026. [Online]. Available: https://docs.python.org/3/library/math.html#math.fsum

[14] The SciPy community, "scipy.sparse.linalg.spsolve," SciPy 1.18.1 documentation, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.linalg.spsolve.html

[15] Python Software Foundation, "Python support for free threading," Python 3.14 documentation, 2026. [Online]. Available: https://docs.python.org/3/howto/free-threading-python.html

[16] Profitina, "Profitina — AI Visibility (GEO) platform," 2026. [Online]. Available: https://profitina.com

Bab 16

Proyek Akhir: Menyimulasikan Sistem Nyata dari Awal sampai Akhir

Setiap bab dalam buku ini menyelesaikan satu jenis masalah: sebuah akar, sistem linear, interpolan, pencocokan, turunan, integral, persamaan diferensial, vektor eigen, optimum, model terlatih, jumlah yang cepat. Pertanyaan nyata jarang datang dalam satu potong. Bab ini mengambil satu pertanyaan nyata, yaitu berapa besar tampungan air yang diperlukan sebuah kebun keluarga agar pohon-pohonnya melewati musim kemarau, lalu menjawabnya dengan membangun simulasi yang memberi pekerjaan kepada hampir setiap metode dalam buku ini. Di akhir bab ini Anda akan mampu memecah sistem nyata menjadi bagian-bagian yang masing-masing cocok dengan satu metode numerik; membangun setiap bagian dan memeriksanya dengan metode kedua yang independen sebelum memercayainya; merakit bagian-bagian itu menjadi simulasi hari demi hari; menjalankan ratusan kemungkinan musim secara paralel dengan bilangan acak yang dapat diulang; mengubah hasilnya menjadi keputusan dengan sebuah pengoptimal, dan memahami keputusan itu melalui satu rumus sederhana; memeriksa keseluruhannya dengan SciPy; dan mengatakan dengan jujur apa yang dapat dan tidak dapat disampaikan simulasi kepada orang yang bertanya.

16.1 Musim Kemarau Pak Hadi

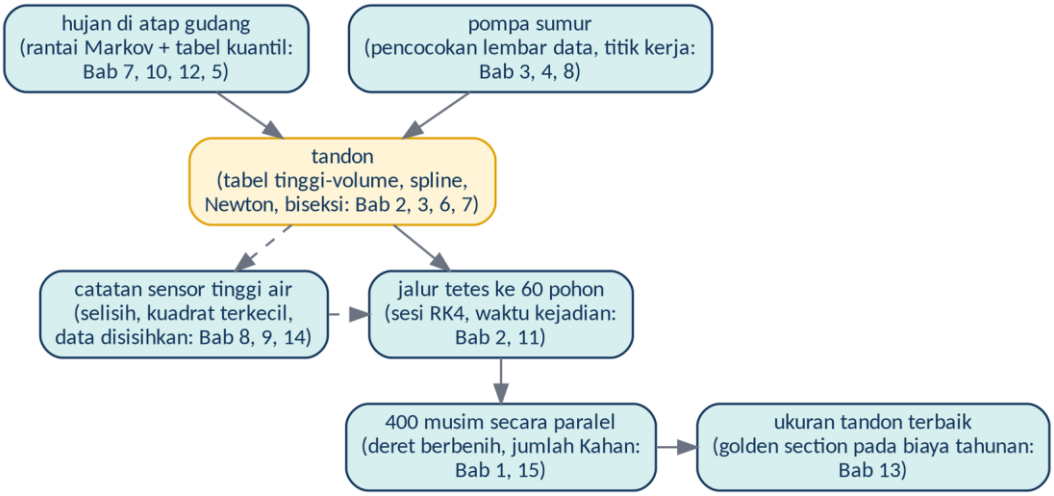
Keluarga Pak Hadi mengelola kebun buah dan durian di lereng bukit sebelah timur Jember, Jawa Timur. Bagian tertua kebun itu adalah satu blok berisi enam puluh pohon durian, ditanam oleh ayahnya dan kini sudah berbuah penuh, dengan rambutan, manggis, dan alpukat di teras-teras di bawahnya. Durian adalah sumber penghasilan kebun, dan buahnya bergantung pada pohon-pohon itu melewati musim kemarau dari Juni sampai Oktober tanpa cekaman kekeringan, terutama pada Agustus dan September, ketika bunga dan buah muda sedang terbentuk.

Air datang dari tiga tempat. Sebuah sumur di kaki lereng mempunyai pompa listrik yang mengangkat air ke tandon berkapasitas 13.000 liter di atas menara, lebih tinggi dari pohon-pohon. Atap gudang penyimpanan, 120 meter persegi seng gelombang, mengalirkan air ke tandon yang sama ketika hujan. Dari tandon, sebuah jalur irigasi tetes mengalir karena gravitasi ke setiap pohon; setiap pagi pukul enam Pak Hadi membuka keran dan menutupnya ketika jatah hari itu sudah keluar, atau paling lambat pukul setengah sembilan, ketika ia harus berangkat ke pasar.

Tahun lalu sistem itu gagal pada bulan September. Sumur, seperti setiap tahun, memberi air makin sedikit seiring berjalannya kemarau; pohon-pohon, seperti setiap tahun, memerlukan makin banyak. Selama tiga minggu tandon hampir kosong setiap pagi, tekanan di jalur tetes rendah, dan pohon-pohon mungkin hanya mendapat dua pertiga jatahnya. Panen bulan Januari lebih kecil. Seorang tetangga menyarankan tandon kedua. Seorang pedagang di kota menawarkan tandon plastik seharga sekitar Rp1.500 per liter kapasitas. Tetapi berapa liter? Lima ribu lagi? Dua puluh ribu? Apakah itu akan membantu sama sekali, bila masalah sebenarnya ada di sumur?

Tidak ada rumus yang menjawab ini. Jawabannya bergantung pada hujan, yang acak; pada sumur, yang melemah sepanjang musim; pada kebutuhan pohon, yang bertambah; pada bentuk tandon,

yang menentukan tekanan; dan pada pompa, yang debitnya bergantung pada seberapa tinggi ia harus mengangkat air. Masing-masing adalah masalah numerik kecil yang sudah dapat diselesaikan oleh salah satu bab sebelumnya. Bila digabungkan, hari demi hari, dan diulang atas ratusan kemungkinan musim kemarau, semuanya menjawab pertanyaan Pak Hadi. **Gambar 16.1** memperlihatkan rencananya.



Gambar 16.1 Sistem air kebun sebagai simulasi: setiap bagian dihitung dengan metode dari bab-bab sebelumnya, bagian-bagian itu bertemu di tandon, dan banyak musim simulasi menjadi masukan bagi pilihan ukuran tandon.

16.2 Gagasannya dengan Kata-Kata Sederhana

Simulasi meniru sebuah sistem langkah demi langkah dalam waktu, memakai model setiap bagian dan model interaksi antarbagian. Simulasi kita maju satu hari setiap langkah. Setiap hari ia menentukan cuaca, menambahkan air hujan dari atap ke tandon, menjalankan sesi irigasi tetes pagi hari dan mencatat apakah pohon-pohon mendapat jatah penuh, serta menjalankan pompa pada sore hari untuk mengisi ulang tandon sejauh yang diizinkan sumur. Setelah 153 hari, 1 Juni sampai 31 Oktober, simulasi melaporkan berapa banyak air yang tidak diterima pohon dan berapa listrik yang dipakai pompa.

Satu musim simulasi adalah satu kemungkinan masa depan, diambil dengan cuaca acak. Keputusan Pak Hadi harus baik untuk berbagai masa depan yang mungkin ia hadapi, sehingga simulasi diulang untuk ratusan musim, masing-masing dengan cuaca acaknya sendiri; inilah *metode Monte Carlo* [1]. Rata-rata atas musim-musim itu memperkirakan apa yang dapat diharapkan, dan sebarannya memperlihatkan seberapa besar peran keberuntungan.

Disiplin yang membuat simulasi dapat dipercaya adalah membangunnya dari *bagian-bagian yang sudah diperiksa*. Sebelum satu musim pun disimulasikan, setiap bagian dihitung dengan dua cara, dengan metode yang akan dipakai simulasi dan dengan metode yang independen, dan keduanya harus sesuai. Tinggi air dari spline harus sesuai dengan metode Newton pada spline yang sama; distribusi cuaca stasioner dari metode pangkat harus sesuai dengan Gauss–Seidel; sesi tetes yang diintegrasikan dengan langkah besar harus sesuai dengan sesi yang sama dengan langkah kecil. Barulah kemudian bagian-bagian itu dirakit. Inilah verifikasi yang dituntut buku-buku metode numerik untuk setiap metode tunggal [2], [3], kini diterapkan pada seluruh sistem. **Tabel 16.1** mendaftar di mana metode setiap bab bekerja.

Tabel 16.1 Tempat metode setiap bab bekerja dalam simulasi kebun.

Bagian sistem	Pertanyaan	Metode (bab)
Tandon	Berapa air pada suatu ketinggian, dan berapa ketinggian untuk suatu volume?	spline kubik, LU tridiagonal, Newton, biseksi (7, 6, 3, 2)
Pompa	Berapa debitnya ketika mengangkat air ke tandon?	kuadrat terkecil, eliminasi Gauss, Newton (8, 4, 3)
Cuaca	Seberapa sering hujan, dan seberapa banyak?	rantai Markov, metode pangkat, Gauss–Seidel, interpolasi, aturan trapesium (12, 5, 7, 10)
Catatan badai	Berapa hujan yang dibawa satu badai?	aturan trapesium dan Simpson (10)
Jalur tetes	Berapa lama jatah keluar pada tekanan tandon?	Runge–Kutta, *regula falsi* untuk waktu akhir (11, 2)
Sensor tinggi air	Apakah model tetes sesuai dengan kebun?	selisih pusat, kuadrat terkecil, data yang disisihkan (9, 8, 14)
Musim-musim	Apa yang terjadi dalam banyak musim kemarau?	deret acak berbenih, pekerja paralel, jumlah terkompensasi (1, 15)
Keputusan	Ukuran tandon mana yang biaya tahunannya paling kecil?	*golden-section search* (13)

Mari kerjakan satu hari dengan tangan untuk melihat neracanya. Pada 15 September, hari ke-106 musim ini, setiap pohon memerlukan 42 liter, sehingga jatahnya $60 \times 42 = 2520$ liter, sedangkan sumur hanya dapat memberi 1600 liter sore itu. Bila tidak hujan, tandon mengakhiri hari itu 920 liter lebih rendah daripada saat memulainya. Sepanjang musim, dengan menjumlahkan setiap hari ketika pohon memerlukan lebih dari yang dapat diberikan sumur, kekurangannya 45.665 liter; pada hari-hari lain sumur dapat memberi 50.595 liter lebih banyak daripada yang diperlukan, tetapi tandon yang sudah penuh tidak dapat menampung kelebihan itu. Jadi tandon memulai musim kemarau dalam keadaan penuh dan harus membawa kebun melewati kekurangan sekitar 45.000 liter, dikurangi apa pun yang ditambahkan hujan setelah tandon mempunyai ruang untuknya. Tandon 13.000 liter jelas tidak sanggup; simulasi akan memberi tahu seberapa besar tandon yang diperlukan, dan apakah liter-liter tambahan itu sepadan dengan harganya.

> **WAWASAN KUNCI:** Simulasi yang dapat dipercaya itu dibangun, bukan sekadar ditulis. Modelkan setiap bagian dengan metode yang cocok, periksa setiap bagian dengan metode yang independen, rakit, lalu periksa keseluruhannya dengan hitungan tangan yang sederhana (kekurangan air satu musim, hujan yang diharapkan) sebelum memercayai jawaban apa pun yang diberikannya.

16.3 Modelnya, Bagian demi Bagian

16.3.1 Tandon: sebuah tabel dan dua spline

Tandon berdiri di atas menara setinggi dua meter dan sedikit lebih lebar di atas daripada di bawah. Pak Hadi pernah menera tandon itu dengan ember 20 liter, mencatat volume setiap kenaikan 25 cm: 0, 1190, 2500, 3940, 5500, 7190, 9000, 10940, dan 13000 liter pada ketinggian 0, 25, …, 200 cm. Simulasi memerlukan tabel ini dua arah. Tekanan jalur tetes bergantung pada *ketinggian* h permukaan air, sedangkan neraca air disimpan dalam *volume* V. Spline kubik alami melalui titik-titik (h_i, V_i) (Bab 7) memberi $V(h)$; turunan keduanya berasal dari sistem linear tridiagonal yang diselesaikan dengan algoritma Thomas, faktorisasi LU dari Bab 6 untuk matriks tridiagonal.

Untuk $h(V)$ ada dua cara. Yang pertama adalah menyelesaikan $V(h) = V$ untuk h dengan metode Newton (Bab 3), memakai turunan spline, atau dengan biseksi (Bab 2). Yang kedua adalah *interpolasi invers*: mencocokkan spline kedua melalui titik-titik yang ditukar (V_i, h_i). Simulasi mengevaluasi $h(V)$ ribuan kali per musim, sehingga ia memakai spline invers, yang hanya memerlukan satu evaluasi alih-alih beberapa langkah Newton. Keduanya adalah interpolan berbeda dari data yang sama, jadi tidak harus sama persis; pemeriksaan komponen di Subbab 16.5 menunjukkan bahwa keduanya berselisih 0,03 mm pada 6500 liter, jauh di bawah ketelitian sebuah ember.

Kapasitas yang lebih besar dimodelkan sebagai beberapa tandon berbentuk sama yang berdiri berdampingan, diisi dan dikosongkan bersama, sehingga tandon berkapasitas C mempunyai ketinggian

$$h_C(V) = h\left(\frac{V}{C/13000}\right), \tag{16.1}$$

dengan tinggi yang sama, dua meter, dan luas yang lebih besar secara sebanding.

16.3.2 Pompa: kurva hasil pencocokan dan titik kerja

Lembar data pompa memberikan *head* yang dihasilkannya, yaitu tinggi dalam meter yang dapat dicapai air yang didorongnya, pada delapan debit dari 0 sampai 70 liter per menit. Fungsi kuadratik

$$H_p(Q) = a + bQ + cQ^2 \tag{16.2}$$

dicocokkan dengan kuadrat terkecil (Bab 8): ketiga persamaan normal diselesaikan dengan eliminasi Gauss dengan *pivoting* parsial (Bab 4). Pipa dari sumur ke tandon menuntut *head* yang bertambah bersama debit: angkatan statis 12 m dari muka air sumur ke puncak tandon, ditambah kehilangan gesekan yang tumbuh sebanding kuadrat debit, $H_s(Q) = 12 + 0.0011\,Q^2$. Pompa bekerja pada *titik kerja*, tempat pasokan sama dengan tuntutan,

$$g(Q) = H_p(Q) - H_s(Q) = 0, \tag{16.3}$$

akar yang ditemukan dengan metode Newton. Dengan tangan dari $Q_0 = 50$: $g(50) = 31.9958 - 0.023155 \times 50 - 0.0041131 \times 2500 - (12 + 2.75) = 5.8054$ dan $g'(50) = -0.023155 - 2 \times 0.0041131 \times 50 - 2 \times 0.0011 \times 50 = -0.5445$, sehingga $Q_1 = 50 + 5.8054/0.5445 = 60.66$ liter per menit; dua langkah lagi mengendap di 59,752. Pada debit itu, 1000 liter memerlukan $1000/59.752 = 16.74$ menit, dan pompa yang menarik daya 0,8 kW memakai

$$E = \frac{1000}{60\,Q^*}\,P = \frac{1000}{60 \times 59.752} \times 0.8 = 0.2231 \text{ kWh per 1000 liter}, \tag{16.4}$$

yang dengan tarif rumah tangga Rp1.445 per kWh untuk Juli sampai September 2026 [4] berharga Rp322 per 1000 liter.

16.3.3 Cuaca: rantai Markov dan tabel kuantil

Buku catatan penakar hujan kebun, yang diisi selama banyak musim kemarau, memperlihatkan bahwa hari-hari tidak saling bebas: hari hujan lebih mungkin diikuti hari hujan lagi. Simulasi menggolongkan hari ke dalam tiga keadaan mengikuti kategori hujan harian BMKG [5]: *kering* (di bawah 0,5 mm), *ringan* (0,5 sampai 20 mm), dan *lebat* (20 mm ke atas, yang menggabungkan

kelas sedang, lebat, dan sangat lebat BMKG). Keadaan esok hanya bergantung pada keadaan hari ini, melalui matriks transisi P yang ditaksir dari buku catatan,

$$P = \begin{pmatrix} 0.88 & 0.10 & 0.02 \\ 0.60 & 0.32 & 0.08 \\ 0.50 & 0.35 & 0.15 \end{pmatrix}, \tag{16.5}$$

dengan baris i memberikan peluang keadaan esok setelah hari berkeadaan i. Inilah *rantai Markov*, objek sejenis dengan rantai penjelajah web PageRank di Bab 12. Bagian hari jangka panjang dalam setiap keadaan adalah *distribusi stasioner* $\boldsymbol{\pi}$, vektor eigen kiri P untuk nilai eigen 1:

$$\boldsymbol{\pi} P = \boldsymbol{\pi}, \qquad \pi_1 + \pi_2 + \pi_3 = 1. \tag{16.6}$$

Metode pangkat menemukannya dengan mengulang $\boldsymbol{\pi} \leftarrow \boldsymbol{\pi} P$; galatnya mengecil setiap langkah dengan faktor nilai eigen terbesar kedua, di sini 0,305, sehingga 40 langkah memberi presisi ganda penuh. Sebagai pemeriksaan, Persamaan (16.6) juga merupakan sistem linear $(I - P^{\mathsf{T}})\boldsymbol{\pi} = \mathbf{0}$, yang diselesaikan Gauss–Seidel (Bab 5) sampai suatu skala, dan skalanya ditetapkan kemudian dengan jumlahnya. Keduanya memberi $\boldsymbol{\pi} = (0.8288, 0.1386, 0.0326)$: dalam jangka panjang 17% hari musim kemarau membawa hujan.

Berapa banyak hujannya? Pada hari ringan, kedalamannya diambil dari *tabel kuantil* buku catatan: hari ringan terkecil 0,5 mm, seperempatnya kurang dari 2 mm, separuhnya kurang dari 5, tiga perempatnya kurang dari 10, dan yang terbesar 20. Bilangan acak seragam u dalam $[0,1)$ diubah menjadi kedalaman dengan interpolasi linear dalam tabel ini (Bab 7), yang meniru distribusi buku catatan. Kedalaman rata-rata hari seperti itu adalah integral fungsi kuantil atas u dari 0 sampai 1, dan karena fungsinya linear sepotong-sepotong, aturan trapesium dari Bab 10 memberikannya secara eksak: $0.25 \times (0.5/2 + 2 + 5 + 10 + 20/2) = 6.8125$ mm. Untuk hari lebat nilainya 41,375 mm. Jadi hujan yang diharapkan $0.1386 \times 6.8125 + 0.0326 \times 41.375 = 2.29$ mm sehari, 351 mm semusim, dan dengan koefisien limpasan 0,85 atap seluas 120 m^2 memberi 102 liter per milimeter, sekitar 35.800 liter semusim. Perkiraan tangan ini akan dibandingkan dengan simulasi.

16.3.4 Pohon dan sumur sepanjang musim

Kebiasaan menyiram Pak Hadi sendiri memberi jatah per pohon di pertengahan setiap bulan: 25 liter sehari pada Juni, 32 pada Juli, 38 pada Agustus, 42 pada September, dan 35 pada Oktober. Air yang dapat diberikan sumur dalam sehari turun dari 2800 liter pada Juni menjadi 2400, 1900, 1600, lalu, setelah hujan pertama bulan Oktober, 1800. Di antara hari-hari pertengahan bulan simulasi menginterpolasi secara linear, pilihan yang sederhana, tidak pernah melampaui, dan seteliti bilangan-bilangan itu sendiri. Setelah hari dengan hujan 10 mm atau lebih, tanahnya basah dan pohon tidak disiram.

16.3.5 Jalur tetes: persamaan diferensial kecil

Air mengalir ke pohon karena gravitasi, sehingga debitnya bergantung pada tinggi permukaan air di atas jalur tetes: ketinggian h dalam tandon ditambah tinggi menara $Z = 1.5$ m. Untuk penetes dan keran, debit tumbuh sebanding akar kuadrat *head* tekanan, seperti tandon di Bab 11 yang mengosong melalui sebuah lubang, sehingga selama keran terbuka

$$\frac{dV}{dt} = -K\sqrt{\frac{h(V)}{100} + Z}, \qquad K = 12 \text{ L/min per } \sqrt{\text{m}}, \tag{16.7}$$

dengan h dalam sentimeter dan t dalam menit. Dengan tandon penuh debitnya $12\sqrt{3.5} = 22.4$ liter per menit; dengan sisa air 30 cm, $12\sqrt{1.8} = 16.1$. Jatah pagi sampai 2520 liter karena itu memerlukan antara sekitar 115 dan 150 menit, dan jendela dari pukul enam sampai setengah sembilan tepat 150 menit: bila tandon rendah, pohon mendapat kurang dari jatahnya *walaupun tandon masih berisi cukup air*, karena tekanannya terlalu rendah untuk mengalirkannya tepat waktu. Efek itulah yang membuat masalah ini memerlukan persamaan diferensial, bukan sekadar pengurangan.

Simulasi mengintegrasikan Persamaan (16.7) dengan metode Runge–Kutta klasik (Bab 11) dengan langkah 10 menit. Sesi berakhir ketika volume yang tersalurkan mencapai jatah, pada suatu saat di dalam sebuah langkah. Saat itu adalah akar dari $V(\tau) - V_{\text{target}} = 0$ terhadap panjang langkah τ, dengan $V(\tau)$ satu langkah Runge–Kutta sepanjang τ dari awal langkah; *regula falsi* (Bab 2) menemukannya dalam beberapa evaluasi karena fungsinya hampir linear dalam sepuluh menit. Sesi juga berakhir di ujung jendela, berapa pun yang sudah tersalurkan.

16.3.6 Satu hari, dan satu musim

Misalkan V_d air di tandon pada subuh hari ke-d, r_d hujan dalam milimeter, R_d jatahnya, dan W_d yang dapat diberikan sumur. Satu hari simulasi adalah

$$\begin{aligned} V' &= \min(C,\ V_d + 102\, r_d), \\ V'' &= V' - G_d, \qquad G_d = \text{sesi tetes dari } V' \text{ untuk } R_d \ (0 \text{ bila } r_d \geq 10), \\ V_{d+1} &= V'' + \min(C - V'',\ W_d), \end{aligned} \tag{16.8}$$

dengan C kapasitas dan G_d air yang benar-benar diterima pohon. Air yang kurang hari itu adalah $R_d - G_d$, luapannya $V_d + 102\, r_d - C$ bila positif, dan energi pompa mengikuti dari volume yang dipompa dan Persamaan (16.4). Tandon penuh pada 1 Juni, selepas musim hujan.

16.3.7 Banyak musim, secara dapat diulang

Misalkan X_s suatu hasil musim s, misalnya liter air yang kurang. Taksiran Monte Carlo dari N musim dan galat bakunya adalah

$$\bar{X} = \frac{1}{N}\sum_{s=1}^{N} X_s\,, \qquad \text{SE}(\bar{X}) = \frac{\sigma_X}{\sqrt{N}}, \tag{16.9}$$

dengan σ_X simpangan baku X atas musim-musim. Memperkecil galat menjadi separuhnya memerlukan empat kali lebih banyak musim, dan itulah sebabnya kecepatan (Bab 15) penting. Untuk sebuah peluang p, misalnya peluang sebuah musim mengalami kekurangan apa pun, $\sigma = \sqrt{p(1-p)}$, sehingga dengan $N = 400$ dan $p \approx 0.36$ galat bakunya $\sqrt{0.36 \times 0.64/400} = 0.024$.

Setiap musim mendapat deret acaknya sendiri, pembangkit SplitMix64 dari Bab 14 yang diberi benih $20260601 + s$. Karena keacakan sebuah musim hanya bergantung pada nomornya, musim-musim dapat dihitung dalam urutan apa pun oleh berapa pun banyaknya pekerja, dan hasilnya, dikumpulkan dalam urutan musim dan dijumlahkan dengan metode Kahan (Bab 15), sama bit demi bit pada satu inti maupun banyak, dan dalam Java maupun Python. Pemberian benih yang

sama juga memberi *bilangan acak bersama* (*common random numbers*): ketika dua ukuran tandon dibandingkan, musim s mempunyai cuaca yang persis sama untuk keduanya, sehingga perbedaan di antara keduanya disebabkan oleh tandon, bukan oleh keberuntungan.

16.3.8 Keputusan, dan rumus di baliknya

Biaya tahunan Pak Hadi untuk tandon berkapasitas C adalah harga tandon yang dibagi atas umur pakainya, listrik pompa, dan nilai buah yang hilang karena air yang kurang:

$$\text{cost}(C) = a\,C + e\,\mathbb{E}[\text{kWh}] + p\,\mathbb{E}[D(C)], \tag{16.10}$$

dengan $a = 150$ rupiah per liter kapasitas per tahun (Rp1.500 per liter selama sepuluh tahun), $e = 1445$ rupiah per kWh, $p = 500$ rupiah per liter air yang tidak diterima pohon (taksirannya sendiri atas buah yang hilang, yang kita terima apa adanya), dan $D(C)$ liter air yang kurang dalam satu musim. Suku pertama naik bersama C, suku terakhir turun, dan *golden-section search* dari Bab 13 menemukan minimumnya, memakai musim-musim simulasi yang sama untuk setiap C.

Ada cara untuk meramalkan jawabannya. Menambah satu liter kapasitas berbiaya a setahun. Tambahan itu menghemat satu liter air yang kurang tepat pada musim-musim ketika tandon kehabisan, dan tidak menghemat apa-apa pada musim lain, sehingga penghematannya p kali peluang terjadinya kekurangan. Di titik optimum keduanya harus seimbang:

$$a = p \cdot \Pr[D(C^*) > 0] \quad\Leftrightarrow\quad \Pr[\text{kekurangan dengan } C^*] = \frac{a}{p} = \frac{150}{500} = 0.30. \tag{16.11}$$

Inilah *aturan newsvendor* (penjual koran) dari riset operasi, yang pertama kali diturunkan untuk penjual koran yang memutuskan berapa eksemplar yang harus dibeli [6]: tambahlah persediaan sampai peluang kehabisan sama dengan perbandingan biaya menyimpan terhadap biaya kehabisan. Tandon terbaik adalah tandon yang kehabisan pada 30% musim kemarau. Kedengarannya banyak, tetapi kekurangan pada musim yang hanya sedikit lebih buruk daripada biasanya hanya kecil ruginya; membuat tandon cukup besar agar tidak pernah kehabisan akan berbiaya lebih besar setiap tahun daripada kerugian pada musim-musim buruk yang jarang itu.

16.4 Program-Programnya

Program-program bab ini berada di `Chapter16/python` dan `Chapter16/java`, dengan data di `Chapter16/data`: `tank_table.csv`, `pump_curve.csv`, `storm.csv` (sebuah badai yang dicatat setiap 10 menit), dan `sensor_log.csv` (sensor tinggi air selama satu kali penyiraman, dibangkitkan oleh `make_sensor.py`). Setiap program mencetak teks yang sama dalam kedua bahasa.

16.4.1 Kotak perkakas

Kode 16.1 menghimpun, dalam bentuk terpendek yang jujur, metode-metode dari Bab 2 sampai 15 yang dipakai simulasi. Masing-masing hanya beberapa baris, dan masing-masing diberi label babnya; pembaca yang telah mengikuti buku ini akan mengenali semuanya. Ada dua rincian baru. `Spline` menyediakan `slope` untuk metode Newton, dan `false_position` adalah *regula falsi* dari Bab 2, yang dipakai untuk akhir sesi tetes.

Kode 16.1 `toolbox.py`: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun.

```
"""The numerical tools of Chapters 2-15 that the orchard simulation reuses,
each in its shortest honest form. Plain Python, no libraries."""
import math

MASK = 0xFFFFFFFFFFFFFFFF


def add(values):
    """Left-to-right sum, the same additions as a Java loop. (Since Python
    3.12 the built-in sum() compensates rounding errors, so it can differ.)"""
    s = 0.0
    for v in values:
        s += v
    return s


class Rng:
    """SplitMix64 (Chapters 14-15): the same numbers in Python and Java."""

    def __init__(self, seed):
        self.s = seed & MASK

    def next64(self):
        self.s = (self.s + 0x9E3779B97F4A7C15) & MASK
        z = self.s
        z = ((z ^ (z >> 30)) * 0xBF58476D1CE4E5B9) & MASK
        z = ((z ^ (z >> 27)) * 0x94D049BB133111EB) & MASK
        return z ^ (z >> 31)

    def uniform(self):
        """A double in [0, 1) from the top 53 bits."""
        return (self.next64() >> 11) * (1.0 / 9007199254740992.0)


def bisection(f, a, b, tol):
    """Chapter 2: halve [a, b] keeping the sign change of f."""
    fa = f(a)
    while b - a > tol:
        m = (a + b) / 2.0
        fm = f(m)
        if (fm < 0.0) == (fa < 0.0):
            a, fa = m, fm
        else:
            b = m
    return (a + b) / 2.0
```

Kode 16.1 (Lanjutan) *toolbox.py: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun.*

```
def false_position(f, a, b, tol, kmax=100):
    """Chapter 2: regula falsi, the secant through the bracket's ends."""
    fa, fb = f(a), f(b)
    for _ in range(kmax):
        c = b - fb * (b - a) / (fb - fa)
        fc = f(c)
        if abs(fc) < tol:
            return c
        if (fc < 0.0) == (fa < 0.0):
            a, fa = c, fc
        else:
            b, fb = c, fc
    return c


def newton(f, df, x, tol, kmax=50):
    """Chapter 3: x <- x - f(x)/f'(x); returns (root, steps)."""
    for k in range(1, kmax + 1):
        dx = f(x) / df(x)
        x -= dx
        if abs(dx) < tol:
            return x, k
    return x, kmax


def gauss_solve(A, b):
    """Chapter 4: Gaussian elimination with partial pivoting."""
    n = len(b)
    M = [row[:] + [b[i]] for i, row in enumerate(A)]
    for k in range(n):
        p = max(range(k, n), key=lambda i: abs(M[i][k]))
        M[k], M[p] = M[p], M[k]
        for i in range(k + 1, n):
            f = M[i][k] / M[k][k]
            for j in range(k, n + 1):
                M[i][j] -= f * M[k][j]
    x = [0.0] * n
    for i in range(n - 1, -1, -1):
        s = M[i][n]
        for j in range(i + 1, n):
            s -= M[i][j] * x[j]
        x[i] = s / M[i][i]
    return x
```

Kode 16.1 (Lanjutan) *toolbox.py: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun.*

```
def gauss_seidel(A, b, x, tol, kmax=1000):
    """Chapter 5: sweep x_i <- (b_i - sum_{j != i} a_ij x_j) / a_ii."""
    n = len(b)
    for k in range(1, kmax + 1):
        change = 0.0
        for i in range(n):
            s = b[i]
            for j in range(n):
                if j != i:
                    s -= A[i][j] * x[j]
            new = s / A[i][i]
            change = max(change, abs(new - x[i]))
            x[i] = new
        if change < tol:
            return x, k
    return x, kmax


def thomas(sub, diag, sup, rhs):
    """Chapter 6: LU for a tridiagonal system (the Thomas algorithm)."""
    n = len(diag)
    c, d = [0.0] * n, [0.0] * n
    c[0], d[0] = sup[0] / diag[0], rhs[0] / diag[0]
    for i in range(1, n):
        m = diag[i] - sub[i] * c[i - 1]
        c[i] = sup[i] / m if i < n - 1 else 0.0
        d[i] = (rhs[i] - sub[i] * d[i - 1]) / m
    x = [0.0] * n
    x[n - 1] = d[n - 1]
    for i in range(n - 2, -1, -1):
        x[i] = d[i] - c[i] * x[i + 1]
    return x


class Spline:
    """Chapter 7: natural cubic spline through (xs, ys), xs increasing."""
```

Kode 16.1 (Lanjutan) *toolbox.py: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun.*

```
    def __init__(self, xs, ys):
        n = len(xs)
        h = [xs[i + 1] - xs[i] for i in range(n - 1)]
        sub, diag, sup, rhs = [0.0] * n, [1.0] * n, [0.0] * n, [0.0] * n
        for i in range(1, n - 1):
            sub[i], diag[i], sup[i] = h[i - 1], 2.0 * (h[i - 1] + h[i]), h[i]
            rhs[i] = 6.0 * ((ys[i + 1] - ys[i]) / h[i]
                            - (ys[i] - ys[i - 1]) / h[i - 1])
        self.xs, self.ys, self.h, self.m = xs, ys, h, thomas(sub, diag, sup, rhs)

    def _piece(self, x):
        i = 0
        while i < len(self.h) - 1 and x > self.xs[i + 1]:
            i += 1
        return i

    def __call__(self, x):
        i = self._piece(x)
        h, m, y = self.h[i], self.m, self.ys
        a, b = self.xs[i + 1] - x, x - self.xs[i]
        return (m[i] * a * a * a + m[i + 1] * b * b * b) / (6.0 * h) \
            + (y[i] / h - m[i] * h / 6.0) * a \
            + (y[i + 1] / h - m[i + 1] * h / 6.0) * b

    def slope(self, x):
        i = self._piece(x)
        h, m, y = self.h[i], self.m, self.ys
        a, b = self.xs[i + 1] - x, x - self.xs[i]
        return (-m[i] * a * a + m[i + 1] * b * b) / (2.0 * h) \
            + (y[i + 1] - y[i]) / h - (m[i + 1] - m[i]) * h / 6.0


def interp(xs, ys, x):
    """Chapter 7: piecewise-linear interpolation, constant outside the table."""
    if x <= xs[0]:
        return ys[0]
    for i in range(1, len(xs)):
        if x <= xs[i]:
            t = (x - xs[i - 1]) / (xs[i] - xs[i - 1])
            return ys[i - 1] + t * (ys[i] - ys[i - 1])
    return ys[-1]
```

Kode 16.1 (Lanjutan) *toolbox.py: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun.*

```
def lstsq(rows, ys):
    """Chapter 8: least squares by the normal equations, solved by Chapter 4."""
    n = len(rows[0])
    A = [[add(r[i] * r[j] for r in rows) for j in range(n)] for i in range(n)]
    b = [add(r[i] * y for r, y in zip(rows, ys)) for i in range(n)]
    return gauss_solve(A, b)


def central_diff(ts, vs):
    """Chapter 9: dv/dt at the inner points by central differences."""
    return [(vs[i + 1] - vs[i - 1]) / (ts[i + 1] - ts[i - 1])
            for i in range(1, len(ts) - 1)]


def trapezoid(ys, h):
    """Chapter 10: composite trapezoid rule on equally spaced samples."""
    return h * (add(ys) - (ys[0] + ys[-1]) / 2.0)


def simpson(ys, h):
    """Chapter 10: composite Simpson rule (an even number of intervals)."""
    s = ys[0] + ys[-1]
    for i in range(1, len(ys) - 1):
        s += (4.0 if i % 2 == 1 else 2.0) * ys[i]
    return h * s / 3.0


def rk4_step(f, t, y, dt):
    """Chapter 11: one classical Runge-Kutta step for y' = f(t, y)."""
    k1 = f(t, y)
    k2 = f(t + dt / 2.0, y + dt / 2.0 * k1)
    k3 = f(t + dt / 2.0, y + dt / 2.0 * k2)
    k4 = f(t + dt, y + dt * k3)
    return y + dt / 6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4)


def power_method(P, iters):
    """Chapter 12: left eigenvector of a transition matrix, pi <- pi P."""
    n = len(P)
    pi = [1.0 / n] * n
    for _ in range(iters):
        pi = [add(pi[i] * P[i][j] for i in range(n)) for j in range(n)]
    return pi
```

Kode 16.1 (Lanjutan) *toolbox.py: metode-metode numerik dari Bab 2 sampai 15 yang dipakai ulang oleh simulasi kebun.*

```
def golden_min(f, a, b, tol):
    """Chapter 13: golden-section search for the minimum of f on [a, b]."""
    r = (math.sqrt(5.0) - 1.0) / 2.0
    x1, x2 = b - r * (b - a), a + r * (b - a)
    f1, f2 = f(x1), f(x2)
    while b - a > tol:
        if f1 < f2:
            b, x2, f2 = x2, x1, f1
            x1 = b - r * (b - a)
            f1 = f(x1)
        else:
            a, x1, f1 = x1, x2, f2
            x2 = a + r * (b - a)
            f2 = f(x2)
    return (a + b) / 2.0


def kahan_sum(xs):
    """Chapter 15: compensated summation."""
    s = c = 0.0
    for x in xs:
        y = x - c
        t = s + y
        c = (t - s) - y
        s = t
    return s
```

Kode 16.2 adalah kotak perkakas Java. Fungsi diberikan sebagai `DoubleUnaryOperator`, dan antarmuka kecil `Ode` mewakili ruas kanan $f(t, y)$ sebuah persamaan diferensial. Setiap gelung menjumlahkan dalam urutan yang sama dengan Python, dan pencarian pivot mengambil kandidat pertama di antara yang sama besar, seperti `max` milik Python, sehingga kedua kotak perkakas memberi hasil yang identik.

Kode 16.2 `Toolbox.java`: metode-metode yang sama dalam Java.

```java
// The numerical tools of Chapters 2-15 that the orchard simulation reuses,
// each in its shortest honest form. Plain Java, no libraries.
import java.util.function.DoubleUnaryOperator;

public class Toolbox {
    /** SplitMix64 (Chapters 14-15): the same numbers in Python and Java. */
    static class Rng {
        long s;

        Rng(long seed) { s = seed; }

        long next64() {
            s += 0x9E3779B97F4A7C15L;
            long z = s;
            z = (z ^ (z >>> 30)) * 0xBF58476D1CE4E5B9L;
            z = (z ^ (z >>> 27)) * 0x94D049BB133111EBL;
            return z ^ (z >>> 31);
        }

        /** A double in [0, 1) from the top 53 bits. */
        double uniform() { return (next64() >>> 11) * (1.0 / 9007199254740992.0); }
    }

    /** Chapter 2: halve [a, b] keeping the sign change of f. */
    static double bisection(DoubleUnaryOperator f, double a, double b, double tol) {
        double fa = f.applyAsDouble(a);
        while (b - a > tol) {
            double m = (a + b) / 2.0, fm = f.applyAsDouble(m);
            if ((fm < 0.0) == (fa < 0.0)) { a = m; fa = fm; } else b = m;
        }
        return (a + b) / 2.0;
    }

    /** Chapter 2: regula falsi, the secant through the bracket's ends. */
    static double falsePosition(DoubleUnaryOperator f, double a, double b,
                                double tol) {
        double fa = f.applyAsDouble(a), fb = f.applyAsDouble(b), c = a;
        for (int k = 0; k < 100; k++) {
            c = b - fb * (b - a) / (fb - fa);
            double fc = f.applyAsDouble(c);
            if (Math.abs(fc) < tol) return c;
            if ((fc < 0.0) == (fa < 0.0)) { a = c; fa = fc; }
            else { b = c; fb = fc; }
        }
        return c;
    }
```

Kode 16.2 (Lanjutan) *Toolbox.java: metode-metode yang sama dalam Java.*

```
    /** Chapter 3: x <- x - f(x)/f'(x); returns {root, steps}. */
    static double[] newton(DoubleUnaryOperator f, DoubleUnaryOperator df,
                           double x, double tol) {
        for (int k = 1; k <= 50; k++) {
            double dx = f.applyAsDouble(x) / df.applyAsDouble(x);
            x -= dx;
            if (Math.abs(dx) < tol) return new double[] {x, k};
        }
        return new double[] {x, 50};
    }

    /** Chapter 4: Gaussian elimination with partial pivoting. */
    static double[] gaussSolve(double[][] A, double[] b) {
        int n = b.length;
        double[][] M = new double[n][n + 1];
        for (int i = 0; i < n; i++) {
            System.arraycopy(A[i], 0, M[i], 0, n);
            M[i][n] = b[i];
        }
        for (int k = 0; k < n; k++) {
            int p = k;
            for (int i = k + 1; i < n; i++)
                if (Math.abs(M[i][k]) > Math.abs(M[p][k])) p = i;
            double[] t = M[k]; M[k] = M[p]; M[p] = t;
            for (int i = k + 1; i < n; i++) {
                double f = M[i][k] / M[k][k];
                for (int j = k; j <= n; j++) M[i][j] -= f * M[k][j];
            }
        }
        double[] x = new double[n];
        for (int i = n - 1; i >= 0; i--) {
            double s = M[i][n];
            for (int j = i + 1; j < n; j++) s -= M[i][j] * x[j];
            x[i] = s / M[i][i];
        }
        return x;
    }
```

Kode 16.2 (Lanjutan) *Toolbox.java: metode-metode yang sama dalam Java.*

```
    /** Chapter 5: sweep x_i <- (b_i - sum_{j != i} a_ij x_j) / a_ii.
     *  Returns the number of sweeps; x is updated in place. */
    static int gaussSeidel(double[][] A, double[] b, double[] x, double tol) {
        int n = b.length;
        for (int k = 1; k <= 1000; k++) {
            double change = 0.0;
            for (int i = 0; i < n; i++) {
                double s = b[i];
                for (int j = 0; j < n; j++) if (j != i) s -= A[i][j] * x[j];
                double nu = s / A[i][i];
                change = Math.max(change, Math.abs(nu - x[i]));
                x[i] = nu;
            }
            if (change < tol) return k;
        }
        return 1000;
    }

    /** Chapter 6: LU for a tridiagonal system (the Thomas algorithm). */
    static double[] thomas(double[] sub, double[] diag, double[] sup,
                           double[] rhs) {
        int n = diag.length;
        double[] c = new double[n], d = new double[n], x = new double[n];
        c[0] = sup[0] / diag[0];
        d[0] = rhs[0] / diag[0];
        for (int i = 1; i < n; i++) {
            double m = diag[i] - sub[i] * c[i - 1];
            c[i] = i < n - 1 ? sup[i] / m : 0.0;
            d[i] = (rhs[i] - sub[i] * d[i - 1]) / m;
        }
        x[n - 1] = d[n - 1];
        for (int i = n - 2; i >= 0; i--) x[i] = d[i] - c[i] * x[i + 1];
        return x;
    }

    /** Chapter 7: natural cubic spline through (xs, ys), xs increasing. */
    static class Spline implements DoubleUnaryOperator {
        final double[] xs, ys, h, m;
```

Kode 16.2 (Lanjutan) *Toolbox.java: metode-metode yang sama dalam Java.*

```
        Spline(double[] xs, double[] ys) {
            int n = xs.length;
            this.xs = xs;
            this.ys = ys;
            h = new double[n - 1];
            for (int i = 0; i < n - 1; i++) h[i] = xs[i + 1] - xs[i];
            double[] sub = new double[n], diag = new double[n];
            double[] sup = new double[n], rhs = new double[n];
            java.util.Arrays.fill(diag, 1.0);
            for (int i = 1; i < n - 1; i++) {
                sub[i] = h[i - 1];
                diag[i] = 2.0 * (h[i - 1] + h[i]);
                sup[i] = h[i];
                rhs[i] = 6.0 * ((ys[i + 1] - ys[i]) / h[i]
                                - (ys[i] - ys[i - 1]) / h[i - 1]);
            }
            m = thomas(sub, diag, sup, rhs);
        }

        int piece(double x) {
            int i = 0;
            while (i < h.length - 1 && x > xs[i + 1]) i++;
            return i;
        }

        public double applyAsDouble(double x) {
            int i = piece(x);
            double hi = h[i], a = xs[i + 1] - x, b = x - xs[i];
            return (m[i] * a * a * a + m[i + 1] * b * b * b) / (6.0 * hi)
                    + (ys[i] / hi - m[i] * hi / 6.0) * a
                    + (ys[i + 1] / hi - m[i + 1] * hi / 6.0) * b;
        }

        double slope(double x) {
            int i = piece(x);
            double hi = h[i], a = xs[i + 1] - x, b = x - xs[i];
            return (-m[i] * a * a + m[i + 1] * b * b) / (2.0 * hi)
                    + (ys[i + 1] - ys[i]) / hi - (m[i + 1] - m[i]) * hi / 6.0;
        }
    }
```

Kode 16.2 (Lanjutan) *Toolbox.java: metode-metode yang sama dalam Java.*

```
    /** Chapter 7: piecewise-linear interpolation, constant outside the table. */
    static double interp(double[] xs, double[] ys, double x) {
        if (x <= xs[0]) return ys[0];
        for (int i = 1; i < xs.length; i++)
            if (x <= xs[i]) {
                double t = (x - xs[i - 1]) / (xs[i] - xs[i - 1]);
                return ys[i - 1] + t * (ys[i] - ys[i - 1]);
            }
        return ys[ys.length - 1];
    }

    /** Chapter 8: least squares by the normal equations, solved by Chapter 4. */
    static double[] lstsq(double[][] rows, double[] ys) {
        int n = rows[0].length;
        double[][] A = new double[n][n];
        double[] b = new double[n];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++)
                for (double[] r : rows) A[i][j] += r[i] * r[j];
            for (int k = 0; k < rows.length; k++) b[i] += rows[k][i] * ys[k];
        }
        return gaussSolve(A, b);
    }

    /** Chapter 10: composite trapezoid rule on equally spaced samples. */
    static double trapezoid(double[] ys, double h) {
        double s = 0.0;
        for (double y : ys) s += y;
        return h * (s - (ys[0] + ys[ys.length - 1]) / 2.0);
    }

    /** Chapter 10: composite Simpson rule (an even number of intervals). */
    static double simpson(double[] ys, double h) {
        double s = ys[0] + ys[ys.length - 1];
        for (int i = 1; i < ys.length - 1; i++)
            s += (i % 2 == 1 ? 4.0 : 2.0) * ys[i];
        return h * s / 3.0;
    }

    interface Ode { double f(double t, double y); }
```

Kode 16.2 (Lanjutan) *Toolbox.java: metode-metode yang sama dalam Java.*

```
    /** Chapter 11: one classical Runge-Kutta step for y' = f(t, y). */
    static double rk4Step(Ode f, double t, double y, double dt) {
        double k1 = f.f(t, y);
        double k2 = f.f(t + dt / 2.0, y + dt / 2.0 * k1);
        double k3 = f.f(t + dt / 2.0, y + dt / 2.0 * k2);
        double k4 = f.f(t + dt, y + dt * k3);
        return y + dt / 6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4);
    }

    /** Chapter 12: left eigenvector of a transition matrix, pi <- pi P. */
    static double[] powerMethod(double[][] P, int iters) {
        int n = P.length;
        double[] pi = new double[n];
        java.util.Arrays.fill(pi, 1.0 / n);
        for (int it = 0; it < iters; it++) {
            double[] nu = new double[n];
            for (int j = 0; j < n; j++)
                for (int i = 0; i < n; i++) nu[j] += pi[i] * P[i][j];
            pi = nu;
        }
        return pi;
    }

    /** Chapter 13: golden-section search for the minimum of f on [a, b]. */
    static double goldenMin(DoubleUnaryOperator f, double a, double b, double tol) {
        double r = (Math.sqrt(5.0) - 1.0) / 2.0;
        double x1 = b - r * (b - a), x2 = a + r * (b - a);
        double f1 = f.applyAsDouble(x1), f2 = f.applyAsDouble(x2);
        while (b - a > tol) {
            if (f1 < f2) {
                b = x2; x2 = x1; f2 = f1;
                x1 = b - r * (b - a);
                f1 = f.applyAsDouble(x1);
            } else {
                a = x1; x1 = x2; f1 = f2;
                x2 = a + r * (b - a);
                f2 = f.applyAsDouble(x2);
            }
        }
        return (a + b) / 2.0;
    }
```

Kode 16.2 (Lanjutan) *Toolbox.java: metode-metode yang sama dalam Java.*

```
    /** Chapter 15: compensated summation. */
    static double kahanSum(double[] xs) {
        double s = 0.0, c = 0.0;
        for (double x : xs) {
            double y = x - c, t = s + y;
            c = (t - s) - y;
            s = t;
        }
        return s;
    }
}
```

16.4.2 Model kebun

Kode 16.3 membangun model Subbab 16.3 dari kotak perkakas: kedua spline tandon dan `Tank` yang diskalakan menurut Persamaan (16.1); pencocokan pompa Persamaan (16.2) dan titik kerjanya Persamaan (16.3); rantai cuaca Persamaan (16.5) dan tabel hujan; tabel-tabel musim; sesi tetes Persamaan (16.7); dan `season`, neraca harian Persamaan (16.8) selama 153 hari. Semua

konstanta diberi nama di bagian atas bagiannya masing-masing, agar Subbab 16.8 dapat mengubahnya.

Kode 16.3 `orchard.py`: sistem air kebun sebagai satu model.

```
"""Pak Hadi's orchard as one model: rain on the shed roof, a tank, a well pump,
a drip line to 60 trees, and the dry season from June 1 to October 31."""
import math
import os
from toolbox import Rng, Spline, interp, lstsq, newton, false_position, rk4_step

DATA = os.path.join(os.path.dirname(os.path.abspath(__file__)), "..", "data")


def read_csv(name):
    with open(os.path.join(DATA, name), encoding="utf-8") as f:
        next(f)
        return [[float(x) for x in line.split(",")] for line in f if line.strip()]


# --- the tank: a bucket-calibrated table, a spline each way (Chapter 7) ---
TABLE = read_csv("tank_table.csv")
LEVELS, LITRES = [r[0] for r in TABLE], [r[1] for r in TABLE]
VOLUME = Spline(LEVELS, LITRES)          # liters at a level (cm)
LEVEL = Spline(LITRES, LEVELS)           # level (cm) at a volume: inverse table
BASE_CAPACITY = LITRES[-1]               # 13,000 L, 2 m high


class Tank:
    """A battery of tanks of the calibrated shape, capacity in liters."""

    def __init__(self, capacity):
        self.capacity, self.scale = capacity, capacity / BASE_CAPACITY

    def level(self, v):
        return LEVEL(min(max(v / self.scale, 0.0), BASE_CAPACITY))


# --- the pump: datasheet curve by least squares (Chapter 8), operating point
#     where pump head equals the pipe's need, by Newton (Chapter 3) ---
CURVE = read_csv("pump_curve.csv")
PA, PB, PC = lstsq([[1.0, q, q * q] for q, _ in CURVE], [h for _, h in CURVE])
STATIC, FRICTION = 12.0, 0.0011          # m lift, m per (L/min)^2 of pipe loss
PUMP_KW, TARIFF = 0.8, 1445.0            # kW drawn, rupiah per kWh (PLN R-1)


def pump_gap(q):
    return PA + PB * q + PC * q * q - (STATIC + FRICTION * q * q)
```

Kode 16.3 (Lanjutan) *orchard.py: sistem air kebun sebagai satu model.*

```
def pump_gap_slope(q):
    return PB + 2.0 * PC * q - 2.0 * FRICTION * q


Q_PUMP, _ = newton(pump_gap, pump_gap_slope, 50.0, 1e-10)     # L/min

# --- weather: a 3-state Markov chain (Chapter 12), rain depth drawn from
#     a quantile table by linear interpolation (Chapter 7) ---
STATES = ("dry", "light", "heavy")       # BMKG: < 0.5, 0.5-20, > 20 mm/day
P = [[0.88, 0.10, 0.02],
     [0.60, 0.32, 0.08],
     [0.50, 0.35, 0.15]]
QUANT = [0.0, 0.25, 0.5, 0.75, 1.0]
RAIN_MM = {1: [0.5, 2.0, 5.0, 10.0, 20.0], 2: [20.0, 26.0, 34.0, 48.0, 95.0]}
ROOF_L_PER_MM = 0.85 * 120.0             # runoff coefficient x roof area (m^2)


def next_state(rng, s):
    u, acc = rng.uniform(), 0.0
    for j in range(3):
        acc += P[s][j]
        if u < acc:
            return j
    return 2


def rain_mm(rng, s):
    return 0.0 if s == 0 else interp(QUANT, RAIN_MM[s], rng.uniform())


# --- trees and well through the season (day 0 = June 1) ---
MID = [14.0, 45.0, 76.0, 106.0, 137.0]  # mid-June ... mid-October
PER_TREE = [25.0, 32.0, 38.0, 42.0, 35.0]        # liters per tree per day
WELL = [2800.0, 2400.0, 1900.0, 1600.0, 1800.0]  # liters the well gives a day
TREES, DAYS, WET_SKIP = 60, 153, 10.0            # no watering after >= 10 mm

# --- the drip line: gravity flow q = K sqrt(level + Z) (Chapter 11) ---
K, Z, WINDOW, DT = 12.0, 1.5, 150.0, 10.0         # L/min/sqrt(m), m, min, min


def session(tank, v0, ration, dt=DT, method="rk4"):
    """Open the valve at 06:00: deliver `ration` liters or stop at the end of
    the window. Returns (liters delivered, minutes the valve was open)."""
    def f(t, v):
        return -K * math.sqrt(tank.level(v) / 100.0 + Z)
```

Kode 16.3 (Lanjutan) *orchard.py: sistem air kebun sebagai satu model.*

```
    def step(v, h):
        return rk4_step(f, 0.0, v, h) if method == "rk4" else v + h * f(0.0, v)

    target, t, v = max(v0 - ration, 0.0), 0.0, v0
    while t < WINDOW:
        h = min(dt, WINDOW - t)
        w = step(v, h)
        if w <= target:                 # the ration is out within this step:
            tau = false_position(lambda s: step(v, s) - target, 0.0, h, 1e-6)
            return v0 - target, t + tau  # when exactly? (Chapter 2)
        t, v = t + h, w
    return v0 - v, WINDOW


def season(seed, capacity):
    """One dry season, day by day. Returns the season's totals in liters,
    kilowatt-hours and days."""
    rng, tank = Rng(seed), Tank(capacity)
    v, s = capacity, 0                  # full tank at the end of the rains
    tot = {"rain": 0.0, "overflow": 0.0, "given": 0.0, "deficit": 0.0,
           "pumped": 0.0, "kwh": 0.0, "short_days": 0}
    for d in range(DAYS):
        s = next_state(rng, s)
        mm = rain_mm(rng, s)
        v += ROOF_L_PER_MM * mm
        tot["rain"] += ROOF_L_PER_MM * mm
        if v > capacity:
            tot["overflow"] += v - capacity
            v = capacity
        if mm < WET_SKIP:
            ration = TREES * interp(MID, PER_TREE, d)
            got, _ = session(tank, v, ration)
            v -= got
            tot["given"] += got
            tot["deficit"] += ration - got
            if ration - got > 1e-6:
                tot["short_days"] += 1
        p = min(capacity - v, interp(MID, WELL, d))   # refill in the evening
        v += p
        tot["pumped"] += p
        tot["kwh"] += p / (Q_PUMP * 60.0) * PUMP_KW
    return tot
```

Kode 16.4 adalah model yang sama dalam Java. Koefisien pompa dan titik kerjanya dihitung dalam blok `static` ketika kelas dimuat, seperti Python menghitungnya ketika modul diimpor. `season` mengembalikan ketujuh totalnya sebagai larik dengan urutan `rain, overflow, given, deficit, pumped, kwh, short_days`.

Kode 16.4 `Orchard.java`: model kebun dalam Java.

```java
// Pak Hadi's orchard as one model: rain on the shed roof, a tank, a well pump,
// a drip line to 60 trees, and the dry season from June 1 to October 31.
import java.io.IOException;
import java.io.UncheckedIOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.List;
import java.util.Locale;

public class Orchard {
    static final Path DATA = Path.of("..", "data");

    static double[][] readCsv(String name) {
        try {
            List<double[]> rows = new ArrayList<>();
            List<String> lines = Files.readAllLines(DATA.resolve(name));
            for (String line : lines.subList(1, lines.size())) {
                if (line.isBlank()) continue;
                String[] p = line.split(",");
                double[] r = new double[p.length];
                for (int i = 0; i < p.length; i++) r[i] = Double.parseDouble(p[i]);
                rows.add(r);
            }
            return rows.toArray(new double[0][]);
        } catch (IOException e) {
            throw new UncheckedIOException(e);
        }
    }

    static double[] column(double[][] rows, int j) {
        double[] c = new double[rows.length];
        for (int i = 0; i < rows.length; i++) c[i] = rows[i][j];
        return c;
    }

    static String f(String fmt, Object... xs) {
        return String.format(Locale.ROOT, fmt, xs);
    }
```

Kode 16.4 (Lanjutan) *Orchard.java: model kebun dalam Java.*

```
    // --- the tank: a bucket-calibrated table, a spline each way (Chapter 7) ---
    static final double[][] TABLE = readCsv("tank_table.csv");
    static final double[] LEVELS = column(TABLE, 0), LITRES = column(TABLE, 1);
    static final Toolbox.Spline VOLUME = new Toolbox.Spline(LEVELS, LITRES);
    static final Toolbox.Spline LEVEL = new Toolbox.Spline(LITRES, LEVELS);
    static final double BASE_CAPACITY = LITRES[LITRES.length - 1];

    /** A battery of tanks of the calibrated shape, capacity in liters. */
    static class Tank {
        final double capacity, scale;

        Tank(double capacity) {
            this.capacity = capacity;
            scale = capacity / BASE_CAPACITY;
        }

        double level(double v) {
            double x = Math.min(Math.max(v / scale, 0.0), BASE_CAPACITY);
            return LEVEL.applyAsDouble(x);
        }
    }

    // --- the pump: datasheet curve by least squares (Chapter 8), operating
    //     point where pump head equals the pipe's need, by Newton (Chapter 3) ---
    static final double[][] CURVE = readCsv("pump_curve.csv");
    static final double[] PCOEF;
    static final double STATIC = 12.0, FRICTION = 0.0011;
    static final double PUMP_KW = 0.8, TARIFF = 1445.0;
    static final double Q_PUMP;

    static {
        double[][] rows = new double[CURVE.length][];
        for (int i = 0; i < CURVE.length; i++) {
            double q = CURVE[i][0];
            rows[i] = new double[] {1.0, q, q * q};
        }
        PCOEF = Toolbox.lstsq(rows, column(CURVE, 1));
        Q_PUMP = Toolbox.newton(Orchard::pumpGap, Orchard::pumpGapSlope,
                                50.0, 1e-10)[0];
    }

    static double pumpGap(double q) {
        return PCOEF[0] + PCOEF[1] * q + PCOEF[2] * q * q
                - (STATIC + FRICTION * q * q);
    }
```

Kode 16.4 (Lanjutan) *Orchard.java: model kebun dalam Java.*

```
    static double pumpGapSlope(double q) {
        return PCOEF[1] + 2.0 * PCOEF[2] * q - 2.0 * FRICTION * q;
    }

    // --- weather: a 3-state Markov chain (Chapter 12), rain depth drawn from
    //     a quantile table by linear interpolation (Chapter 7) ---
    static final double[][] P = {{0.88, 0.10, 0.02},
                                 {0.60, 0.32, 0.08},
                                 {0.50, 0.35, 0.15}};
    static final double[] QUANT = {0.0, 0.25, 0.5, 0.75, 1.0};
    static final double[][] RAIN_MM = {null, {0.5, 2.0, 5.0, 10.0, 20.0},
                                       {20.0, 26.0, 34.0, 48.0, 95.0}};
    static final double ROOF_L_PER_MM = 0.85 * 120.0;

    static int nextState(Toolbox.Rng rng, int s) {
        double u = rng.uniform(), acc = 0.0;
        for (int j = 0; j < 3; j++) {
            acc += P[s][j];
            if (u < acc) return j;
        }
        return 2;
    }

    static double rainMm(Toolbox.Rng rng, int s) {
        return s == 0 ? 0.0 : Toolbox.interp(QUANT, RAIN_MM[s], rng.uniform());
    }

    // --- trees and well through the season (day 0 = June 1) ---
    static final double[] MID = {14.0, 45.0, 76.0, 106.0, 137.0};
    static final double[] PER_TREE = {25.0, 32.0, 38.0, 42.0, 35.0};
    static final double[] WELL = {2800.0, 2400.0, 1900.0, 1600.0, 1800.0};
    static final int TREES = 60, DAYS = 153;
    static final double WET_SKIP = 10.0;

    // --- the drip line: gravity flow q = K sqrt(level + Z) (Chapter 11) ---
    static final double K = 12.0, Z = 1.5, WINDOW = 150.0, DT = 10.0;
```

Kode 16.4 (Lanjutan) *Orchard.java: model kebun dalam Java.*

```
    /** Open the valve at 06:00: deliver `ration` liters or stop at the end of
     *  the window. Returns {liters delivered, minutes the valve was open}. */
    static double[] session(Tank tank, double v0, double ration, double dt,
                            String method) {
        Toolbox.Ode fn = (t, v) -> -K * Math.sqrt(tank.level(v) / 100.0 + Z);
        boolean rk4 = method.equals("rk4");
        double target = Math.max(v0 - ration, 0.0), t = 0.0, v = v0;
        while (t < WINDOW) {
            double h = Math.min(dt, WINDOW - t), v1 = v;
            double w = rk4 ? Toolbox.rk4Step(fn, 0.0, v, h) : v + h * fn.f(0.0, v);
            if (w <= target) {                  // the ration is out within this step:
                double tau = Toolbox.falsePosition(s -> (rk4
                        ? Toolbox.rk4Step(fn, 0.0, v1, s) : v1 + s * fn.f(0.0, v1))
                        - target, 0.0, h, 1e-6);
                return new double[] {v0 - target, t + tau};  // when? (Chapter 2)
            }
            t += h;
            v = w;
        }
        return new double[] {v0 - v, WINDOW};
    }

    /** One dry season, day by day: {rain, overflow, given, deficit, pumped,
     *  kwh, short_days}. */
    static double[] season(long seed, double capacity) {
        Toolbox.Rng rng = new Toolbox.Rng(seed);
        Tank tank = new Tank(capacity);
        double v = capacity;                    // full tank at the end of the rains
        int s = 0;
        double rain = 0, overflow = 0, given = 0, deficit = 0, pumped = 0, kwh = 0;
        int shortDays = 0;
        for (int d = 0; d < DAYS; d++) {
            s = nextState(rng, s);
            double mm = rainMm(rng, s);
            v += ROOF_L_PER_MM * mm;
            rain += ROOF_L_PER_MM * mm;
            if (v > capacity) {
                overflow += v - capacity;
                v = capacity;
            }
            if (mm < WET_SKIP) {
                double ration = TREES * Toolbox.interp(MID, PER_TREE, d);
                double got = session(tank, v, ration, DT, "rk4")[0];
                v -= got;
                given += got;
                deficit += ration - got;
```

Kode 16.4 (Lanjutan) *Orchard.java: model kebun dalam Java.*

```
                if (ration - got > 1e-6) shortDays++;
            }
            double p = Math.min(capacity - v, Toolbox.interp(MID, WELL, d));
            v += p;                             // refill in the evening
            pumped += p;
            kwh += p / (Q_PUMP * 60.0) * PUMP_KW;
        }
        return new double[] {rain, overflow, given, deficit, pumped, kwh,
                             shortDays};
    }
}
```

16.4.3 Memeriksa bagian-bagian

Kode 16.5 adalah program pertama yang dijalankan: ia membangun setiap bagian dan memeriksanya dengan metode kedua, seperti dituntut Subbab 16.2. Ketinggian tandon dihitung dengan spline invers, dengan Newton dan biseksi pada spline maju, dan dengan garis lurus melalui tabel; pencocokan pompa melaporkan simpangan terbesarnya pada lembar data; cuaca stasioner dihitung dengan metode pangkat dan dengan Gauss–Seidel; sebuah badai yang tercatat diintegrasikan dengan aturan trapesium dan aturan Simpson; dan sesi tetes dihitung dengan metode Euler dan dengan Runge–Kutta pada dua ukuran langkah, dari tandon penuh, dari 4000 liter, dan dari 3000 liter.

Kode 16.5 `components.py`: setiap bagian model, diperiksa dengan metode kedua.

```
"""Step 1 of the capstone: build every part of the orchard model and check it
against a second method before trusting the whole.   Usage: python components.py"""
import orchard as o
from toolbox import (newton, bisection, interp, gauss_seidel, power_method,
                     trapezoid, simpson, add)

print("1. Tank: level for 6,500 liters (Chapters 2, 3, 6, 7)")
v = 6500.0
print(f"   inverse spline h(V)          {o.LEVEL(v):9.4f} cm")
h, k = newton(lambda x: o.VOLUME(x) - v, o.VOLUME.slope, 100.0, 1e-10)
print(f"   Newton on V(h) = 6500        {h:9.4f} cm in {k} steps")
h2 = bisection(lambda x: o.VOLUME(x) - v, 0.0, 200.0, 1e-10)
print(f"   bisection on V(h) = 6500     {h2:9.4f} cm")
print(f"   straight lines through table {interp(o.LITRES, o.LEVELS, v):9.4f} cm")

print("2. Pump: datasheet fit and operating point (Chapters 3, 4, 8)")
print(f"   head = {o.PA:.4f} {o.PB:+.6f} Q {o.PC:+.8f} Q^2  (m, Q in L/min)")
worst = max(abs(o.PA + o.PB * q + o.PC * q * q - hd) for q, hd in o.CURVE)
print(f"   largest misfit on the datasheet {worst:.3f} m")
head = o.STATIC + o.FRICTION * o.Q_PUMP * o.Q_PUMP
print(f"   operating point {o.Q_PUMP:.3f} L/min at {head:.3f} m")
kwh = 1000.0 / (o.Q_PUMP * 60.0) * o.PUMP_KW
print(f"   1000 liters: {kwh:.4f} kWh, Rp {kwh * o.TARIFF:.0f}")

print("3. Weather: long-run share of dry, light and heavy days (5, 7, 10, 12)")
pi = power_method(o.P, 40)
print("   power method, 40 steps   " + "  ".join(f"{x:.6f}" for x in pi))
A = [[(1.0 if i == j else 0.0) - o.P[j][i] for j in range(3)] for i in range(3)]
x, k = gauss_seidel(A, [0.0, 0.0, 0.0], [1 / 3, 1 / 3, 1 / 3], 1e-13)
tot = add(x)
print(f"   Gauss-Seidel, {k} sweeps   " + "  ".join(f"{v / tot:.6f}" for v in x))
mean_mm = [0.0] + [trapezoid(o.RAIN_MM[s], 0.25) for s in (1, 2)]
per_day = add(p * m for p, m in zip(pi, mean_mm))
print(f"   mean rain: light day {mean_mm[1]:.4f} mm, heavy day {mean_mm[2]:.4f} mm")
print(f"   expected {per_day:.4f} mm a day, {per_day * o.DAYS:.1f} mm a season,"
      f" {per_day * o.DAYS * o.ROOF_L_PER_MM:.0f} L from the roof")

print("4. A storm on the rain gauge, every 10 minutes (Chapter 10)")
storm = o.read_csv("storm.csv")
rate = [r[1] / 60.0 for r in storm]               # mm per minute
t_mm, s_mm = trapezoid(rate, 10.0), simpson(rate, 10.0)
print(f"   trapezoid {t_mm:.4f} mm, Simpson {s_mm:.4f} mm,"
      f" roof catches {s_mm * o.ROOF_L_PER_MM:.0f} L")
```

Kode 16.5 (Lanjutan) *components.py: setiap bagian model, diperiksa dengan metode kedua.*

```
print("5. Drip session for 2,520 liters (Chapters 2, 11)")
tank = o.Tank(13000.0)
for start in (13000.0, 4000.0, 3000.0):
    for method, dt in (("euler", 10.0), ("rk4", 10.0), ("rk4", 0.5)):
        got, minutes = o.session(tank, start, 2520.0, dt, method)
        print(f"   start {start:7.0f} L, {method:5s} dt {dt:3.1f}: {got:8.2f} L"
              f" in {minutes:8.4f} min")
```

Kode 16.6 melakukan pemeriksaan yang sama dalam Java, dalam urutan yang sama dan dengan baris cetakan yang sama.

Kode 16.6 `Components.java`: pemeriksaan yang sama dalam Java.

```
// Step 1 of the capstone: build every part of the orchard model and check it
// against a second method before trusting the whole.   Usage: java Components
public class Components {
    static void out(String fmt, Object... xs) {
        System.out.println(Orchard.f(fmt, xs));
    }

    public static void main(String[] args) {
        System.out.println("1. Tank: level for 6,500 liters (Chapters 2, 3, 6, 7)");
        double v = 6500.0;
        out("   inverse spline h(V)          %9.4f cm",
                Orchard.LEVEL.applyAsDouble(v));
        double[] nw = Toolbox.newton(x -> Orchard.VOLUME.applyAsDouble(x) - v,
                Orchard.VOLUME::slope, 100.0, 1e-10);
        out("   Newton on V(h) = 6500        %9.4f cm in %d steps",
                nw[0], (int) nw[1]);
        double h2 = Toolbox.bisection(x -> Orchard.VOLUME.applyAsDouble(x) - v,
                0.0, 200.0, 1e-10);
        out("   bisection on V(h) = 6500     %9.4f cm", h2);
        out("   straight lines through table %9.4f cm",
                Toolbox.interp(Orchard.LITRES, Orchard.LEVELS, v));

        System.out.println(
                "2. Pump: datasheet fit and operating point (Chapters 3, 4, 8)");
        double[] c = Orchard.PCOEF;
        out("   head = %.4f %+.6f Q %+.8f Q^2  (m, Q in L/min)", c[0], c[1], c[2]);
        double worst = 0.0;
        for (double[] r : Orchard.CURVE)
            worst = Math.max(worst, Math.abs(c[0] + c[1] * r[0] + c[2] * r[0] * r[0]
                    - r[1]));
        out("   largest misfit on the datasheet %.3f m", worst);
        double q = Orchard.Q_PUMP;
        out("   operating point %.3f L/min at %.3f m", q,
                Orchard.STATIC + Orchard.FRICTION * q * q);
        double kwh = 1000.0 / (q * 60.0) * Orchard.PUMP_KW;
        out("   1000 liters: %.4f kWh, Rp %.0f", kwh, kwh * Orchard.TARIFF);
```

Kode 16.6 (Lanjutan) *Components.java: pemeriksaan yang sama dalam Java.*

```
        System.out.println("3. Weather: long-run share of dry, light and heavy days"
                + " (5, 7, 10, 12)");
        double[] pi = Toolbox.powerMethod(Orchard.P, 40);
        out("   power method, 40 steps   %.6f  %.6f  %.6f", pi[0], pi[1], pi[2]);
        double[][] A = new double[3][3];
        for (int i = 0; i < 3; i++)
            for (int j = 0; j < 3; j++)
                A[i][j] = (i == j ? 1.0 : 0.0) - Orchard.P[j][i];
        double[] x = {1 / 3.0, 1 / 3.0, 1 / 3.0};
        int k = Toolbox.gaussSeidel(A, new double[3], x, 1e-13);
        double tot = x[0] + x[1] + x[2];
        out("   Gauss-Seidel, %d sweeps   %.6f  %.6f  %.6f",
                k, x[0] / tot, x[1] / tot, x[2] / tot);
        double[] meanMm = {0.0, Toolbox.trapezoid(Orchard.RAIN_MM[1], 0.25),
                           Toolbox.trapezoid(Orchard.RAIN_MM[2], 0.25)};
        double perDay = 0.0;
        for (int s = 0; s < 3; s++) perDay += pi[s] * meanMm[s];
        out("   mean rain: light day %.4f mm, heavy day %.4f mm",
                meanMm[1], meanMm[2]);
        out("   expected %.4f mm a day, %.1f mm a season, %.0f L from the roof",
                perDay, perDay * Orchard.DAYS,
                perDay * Orchard.DAYS * Orchard.ROOF_L_PER_MM);

        System.out.println(
                "4. A storm on the rain gauge, every 10 minutes (Chapter 10)");
        double[][] storm = Orchard.readCsv("storm.csv");
        double[] rate = new double[storm.length];
        for (int i = 0; i < storm.length; i++) rate[i] = storm[i][1] / 60.0;
        double tMm = Toolbox.trapezoid(rate, 10.0);
        double sMm = Toolbox.simpson(rate, 10.0);
        out("   trapezoid %.4f mm, Simpson %.4f mm, roof catches %.0f L",
                tMm, sMm, sMm * Orchard.ROOF_L_PER_MM);

        System.out.println("5. Drip session for 2,520 liters (Chapters 2, 11)");
        Orchard.Tank tank = new Orchard.Tank(13000.0);
        String[] methods = {"euler", "rk4", "rk4"};
        double[] dts = {10.0, 10.0, 0.5};
        for (double start : new double[] {13000.0, 4000.0, 3000.0})
            for (int m = 0; m < 3; m++) {
                double[] r = Orchard.session(tank, start, 2520.0,
                        dts[m], methods[m]);
                out("   start %7.0f L, %-5s dt %3.1f: %8.2f L in %8.4f min",
                        start, methods[m], dts[m], r[0], r[1]);
            }
    }
}
```

16.4.4 Memeriksa model tetes terhadap kebun

Konstanta K dalam Persamaan (16.7) berasal dari spesifikasi jalur tetes. Apakah kebun sependapat? Tandon Pak Hadi mempunyai sensor ultrasonik yang mencatat ketinggian air setiap dua menit dengan resolusi 0,5 cm. **Kode 16.7** mengubah catatan satu pagi menjadi volume dengan spline tandon, menaksir debit keluar dengan selisih pusat (Bab 9), dan mencocokkan K dengan kuadrat terkecil (Bab 8) pada $q = Ks$ dengan $s = \sqrt{h/100 + Z}$, pencocokan satu parameter yang persamaan normalnya hanya $K = \sum s_i q_i / \sum s_i^2$. Program mencocokkan pada paruh pertama pagi dan menguji pada paruh kedua, pemeriksaan dengan data yang disisihkan dari Bab 14, dengan tetangga berjarak 2 menit dan 10 menit.

Kode 16.7 `sensor.py`: koefisien tetes yang ditaksir dari sensor tinggi air.

```
"""Check the drip coefficient K against the level sensor (Chapters 8, 9, 14):
outflow from central differences of the logged volume, then least squares for
q = K sqrt(level + Z), fitted on one half and tested on the other.
Usage: python sensor.py"""
import math
import orchard as o
from toolbox import add

log = o.read_csv("sensor_log.csv")
ts = [r[0] for r in log]
vol = [o.VOLUME(r[1]) for r in log]         # liters, from the calibrated table
print(f"{len(log)} readings, {ts[0]:.0f} to {ts[-1]:.0f} min,"
      f" level {log[0][1]:.1f} -> {log[-1][1]:.1f} cm")
for gap in (1, 5):                            # neighbours 2 or 10 minutes away
    pts = []
    for i in range(gap, len(ts) - gap):
        q = -(vol[i + gap] - vol[i - gap]) / (ts[i + gap] - ts[i - gap])
        pts.append((math.sqrt(log[i][1] / 100.0 + o.Z), q))
    half = len(pts) // 2
    fit, test = pts[:half], pts[half:]
    k_fit = add(s * q for s, q in fit) / add(s * s for s, _ in fit)   # 1x1 LS
    rms = math.sqrt(add((q - k_fit * s) * (q - k_fit * s) for s, q in test)
                    / len(test))
    k_all = add(s * q for s, q in pts) / add(s * s for s, _ in pts)
    print(f"step {2 * gap:2d} min: {len(pts)} rates, K (first half)"
          f" {k_fit:.3f}, test RMS {rms:.3f} L/min, K (all) {k_all:.3f}")
print(f"model value K = {o.K:.3f} L/min per sqrt(m)")
```

Kode 16.8 menuliskan jumlah-jumlah pencocokan satu parameter sebagai gelung eksplisit, dalam urutan yang sama dengan penjumlahan oleh `sum` milik Python.

Kode 16.8 `Sensor.java`: taksiran yang sama dalam Java.

```
// Check the drip coefficient K against the level sensor (Chapters 8, 9, 14):
// outflow from central differences of the logged volume, then least squares for
// q = K sqrt(level + Z), fitted on one half and tested on the other.
// Usage: java Sensor
public class Sensor {
    public static void main(String[] args) {
        double[][] log = Orchard.readCsv("sensor_log.csv");
        int n = log.length;
        double[] ts = new double[n], vol = new double[n];
        for (int i = 0; i < n; i++) {
            ts[i] = log[i][0];
            vol[i] = Orchard.VOLUME.applyAsDouble(log[i][1]);
        }
        System.out.println(Orchard.f(
                "%d readings, %.0f to %.0f min, level %.1f -> %.1f cm",
                n, ts[0], ts[n - 1], log[0][1], log[n - 1][1]));
        for (int gap : new int[] {1, 5}) {     // neighbours 2 or 10 minutes away
            int m = n - 2 * gap;
            double[] s = new double[m], q = new double[m];
            for (int i = gap; i < n - gap; i++) {
                q[i - gap] = -(vol[i + gap] - vol[i - gap])
                        / (ts[i + gap] - ts[i - gap]);
                s[i - gap] = Math.sqrt(log[i][1] / 100.0 + Orchard.Z);
            }
            int half = m / 2;
            double sq = 0.0, ss = 0.0;
            for (int i = 0; i < half; i++) sq += s[i] * q[i];
            for (int i = 0; i < half; i++) ss += s[i] * s[i];
            double kFit = sq / ss;                      // 1x1 least squares
            double e2 = 0.0;
            for (int i = half; i < m; i++) {
                double e = q[i] - kFit * s[i];
                e2 += e * e;
            }
            double rms = Math.sqrt(e2 / (m - half));
            double sqa = 0.0, ssa = 0.0;
            for (int i = 0; i < m; i++) sqa += s[i] * q[i];
            for (int i = 0; i < m; i++) ssa += s[i] * s[i];
            System.out.println(Orchard.f("step %2d min: %d rates, K (first half)"
                    + " %.3f, test RMS %.3f L/min, K (all) %.3f",
                    2 * gap, m, kFit, rms, sqa / ssa));
        }
        System.out.println(
                Orchard.f("model value K = %.3f L/min per sqrt(m)", Orchard.K));
    }
}
```

16.4.5 Banyak musim secara paralel

Kode 16.9 menjalankan *N* musim untuk sebuah kapasitas. Musim-musim dipotong menjadi satu potongan per pekerja; setiap potongan berjalan dalam prosesnya sendiri (`ProcessPoolExecutor`, karena simulasinya Python murni, dan *thread* Python akan bergiliran, Bab 15), dan hasilnya kembali dalam urutan musim. `report` menghitung rata-rata pada Persamaan (16.9) dengan penjumlahan Kahan, menghitung musim yang mengalami kekurangan apa pun, dan membaca persentil ke-90 dari air yang kurang pada daftar yang sudah diurutkan.

Kode 16.9 `montecarlo.py`: banyak musim kemarau secara paralel, dapat diulang.

```
"""Many dry seasons at once (Chapter 15): every season has its own random
stream, seeded by its number, so the results do not depend on the number of
worker processes.   Usage: python montecarlo.py [seasons] [workers] [liters]"""
import sys
from concurrent.futures import ProcessPoolExecutor
import orchard as o
from toolbox import kahan_sum

SEED0 = 20260601
KEYS = ("rain", "overflow", "given", "deficit", "pumped", "kwh", "short_days")


def chunk(job):
    capacity, first, count = job
    rows = []
    for i in range(first, first + count):
        tot = o.season(SEED0 + i, capacity)
        rows.append([tot[k] for k in KEYS])
    return rows


def run_seasons(capacity, n, workers):
    """Totals of seasons 0..n-1, in season order, computed by `workers`."""
    size = -(-n // workers)                          # ceiling division
    jobs = [(capacity, a, min(size, n - a)) for a in range(0, n, size)]
    if workers == 1:
        parts = [chunk(j) for j in jobs]
    else:
        with ProcessPoolExecutor(max_workers=workers) as pool:
            parts = list(pool.map(chunk, jobs))
    return [row for part in parts for row in part]


def report(capacity, rows):
    n = len(rows)
    mean = {k: kahan_sum([r[i] for r in rows]) / n for i, k in enumerate(KEYS)}
    deficits = sorted(r[3] for r in rows)
    short = sum(1 for d in deficits if d > 1.0)
    print(f"tank {capacity:.0f} L, {n} seasons:")
    print(f"  rain caught {mean['rain']:9.0f} L,"
          f" overflow {mean['overflow']:9.0f} L")
    print(f"  pumped      {mean['pumped']:9.0f} L,"
          f" electricity {mean['kwh']:.1f} kWh = Rp {mean['kwh'] * o.TARIFF:.0f}")
    print(f"  trees got   {mean['given']:9.0f} L, missed {mean['deficit']:.0f} L on"
          f" {mean['short_days']:.1f} days on average")
    print(f"  seasons with any shortage: {short} of {n};"
          f" 9 in 10 miss less than {deficits[int(0.9 * n)]:.0f} L")


def main():
    n = int(sys.argv[1]) if len(sys.argv) > 1 else 400
    workers = int(sys.argv[2]) if len(sys.argv) > 2 else 2
    caps = [float(sys.argv[3])] if len(sys.argv) > 3 else [13000.0, 30000.0]
    for cap in caps:
        report(cap, run_seasons(cap, n, workers))


if __name__ == "__main__":
    main()
```

Kode 16.9 (Lanjutan) *montecarlo.py: banyak musim kemarau secara paralel, dapat diulang.*

Kode 16.10 melakukan hal yang sama dengan kumpulan *thread* Java; setiap tugas menyimulasikan satu potongan musim dan mengembalikan totalnya, dan `invokeAll`

mengembalikan *future* dalam urutan tugas diberikan, sehingga hasilnya dirakit dalam urutan musim, *thread* mana pun yang selesai lebih dulu.

Kode 16.10 `MonteCarlo.java`: jalannya Monte Carlo yang sama dengan *thread* Java.

```
// Many dry seasons at once (Chapter 15): every season has its own random
// stream, seeded by its number, so the results do not depend on the number of
// threads.   Usage: java MonteCarlo [seasons] [threads] [liters]
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class MonteCarlo {
    static final long SEED0 = 20260601L;

    /** Totals of seasons 0..n-1, in season order, computed by `threads`. */
    static double[][] runSeasons(double capacity, int n, int threads)
            throws Exception {
        int size = (n + threads - 1) / threads;
        ExecutorService pool = Executors.newFixedThreadPool(threads);
        List<Callable<double[][]>> jobs = new ArrayList<>();
        for (int a = 0; a < n; a += size) {
            final int first = a, count = Math.min(size, n - a);
            jobs.add(() -> {
                double[][] rows = new double[count][];
                for (int i = 0; i < count; i++)
                    rows[i] = Orchard.season(SEED0 + first + i, capacity);
                return rows;
            });
        }
        double[][] all = new double[n][];
        int k = 0;
        for (Future<double[][]> f : pool.invokeAll(jobs))
            for (double[] row : f.get()) all[k++] = row;
        pool.shutdown();
        return all;
    }

    static double mean(double[][] rows, int j) {
        double[] col = new double[rows.length];
        for (int i = 0; i < rows.length; i++) col[i] = rows[i][j];
        return Toolbox.kahanSum(col) / rows.length;
    }
```

Kode 16.10 (Lanjutan) *MonteCarlo.java: jalannya Monte Carlo yang sama dengan thread Java.*

```
    static void report(double capacity, double[][] rows) {
        int n = rows.length;
        double[] deficits = new double[n];
        int shortN = 0;
        for (int i = 0; i < n; i++) {
            deficits[i] = rows[i][3];
            if (deficits[i] > 1.0) shortN++;
        }
        Arrays.sort(deficits);
        System.out.println(Orchard.f("tank %.0f L, %d seasons:", capacity, n));
        System.out.println(Orchard.f("  rain caught %9.0f L, overflow %9.0f L",
                mean(rows, 0), mean(rows, 1)));
        System.out.println(Orchard.f(
                "  pumped      %9.0f L, electricity %.1f kWh = Rp %.0f",
                mean(rows, 4), mean(rows, 5), mean(rows, 5) * Orchard.TARIFF));
        System.out.println(Orchard.f(
                "  trees got   %9.0f L, missed %.0f L on %.1f days on average",
                mean(rows, 2), mean(rows, 3), mean(rows, 6)));
        System.out.println(Orchard.f(
                "  seasons with any shortage: %d of %d;"
                + " 9 in 10 miss less than %.0f L",
                shortN, n, deficits[(int) (0.9 * n)]));
    }

    public static void main(String[] args) throws Exception {
        int n = args.length > 0 ? Integer.parseInt(args[0]) : 400;
        int threads = args.length > 1 ? Integer.parseInt(args[1]) : 2;
        double[] caps = args.length > 2 ? new double[] {Double.parseDouble(args[2])}
                : new double[] {13000.0, 30000.0};
        for (double cap : caps) report(cap, runSeasons(cap, n, threads));
    }
}
```

16.4.6 Memilih tandon

Kode 16.11 membungkus Persamaan (16.10) dalam sebuah fungsi kapasitas yang setiap kali menjalankan 200 musim yang sama, menyerahkannya kepada *golden-section search* dari kotak perkakas pada selang 10.000 sampai 50.000 liter dengan toleransi 500 liter, lalu mencetak setiap kapasitas yang dicoba beserta bagian-bagian biayanya, kemudian yang terbaik, kemudian ramalan *newsvendor* Persamaan (16.11).

Kode 16.11 `design.py`: ukuran tandon dengan *golden-section search* pada biaya tahunan.

```
"""How big should the tank be? Golden-section search (Chapter 13) on the
yearly cost, estimated from the same simulated seasons for every size.
Usage: python design.py [seasons] [workers]   (default 200 seasons, 2 workers)"""
import sys
import orchard as o
from montecarlo import run_seasons
from toolbox import golden_min, kahan_sum, add

TANK_RP_PER_L_YEAR = 150.0      # Rp 1,500 per liter of tank, over 10 years
MISSED_RP_PER_L = 500.0         # fruit lost per liter the trees did not get


def main():
    n = int(sys.argv[1]) if len(sys.argv) > 1 else 200
    workers = int(sys.argv[2]) if len(sys.argv) > 2 else 2
    seen = []

    def cost(capacity):
        rows = run_seasons(capacity, n, workers)    # common random numbers
        kwh = kahan_sum([r[5] for r in rows]) / n
        missed = kahan_sum([r[3] for r in rows]) / n
        short = sum(1 for r in rows if r[3] > 1.0) / n
        parts = (TANK_RP_PER_L_YEAR * capacity, kwh * o.TARIFF,
                 MISSED_RP_PER_L * missed)
        seen.append((capacity, short) + parts + (add(parts),))
        return add(parts)

    best = golden_min(cost, 10000.0, 50000.0, 500.0)
    print(f"{n} seasons per size; yearly cost in thousand rupiah")
    print("  tank (L)  P(short)   tank  power  missed   total")
    for cap, short, t, e, m, total in seen:
        print(f"  {cap:8.0f}  {short:8.3f}  {t / 1e3:5.0f}  {e / 1e3:5.1f}"
              f"  {m / 1e3:6.0f}  {total / 1e3:6.0f}")
    total = cost(best)
    print(f"best tank about {best:.0f} L, yearly cost {total / 1e3:.0f} thousand;"
          f" P(short) there {seen[-1][1]:.3f}")
    print(f"newsvendor check: P(short) should be near"
          f" {TANK_RP_PER_L_YEAR / MISSED_RP_PER_L:.3f}")


if __name__ == "__main__":
    main()
```

Kode 16.12 memberikan biaya kepada `goldenMin` dari kotak perkakas sebagai lambda yang mencatat setiap evaluasi, seperti yang dilakukan versi Python dengan daftar `seen`.

Kode 16.12 `Design.java`: pencarian yang sama dalam Java.

```java
// How big should the tank be? Golden-section search (Chapter 13) on the
// yearly cost, estimated from the same simulated seasons for every size.
// Usage: java Design [seasons] [threads]   (default 200 seasons, 2 threads)
import java.util.ArrayList;
import java.util.List;

public class Design {
    static final double TANK_RP_PER_L_YEAR = 150.0;  // Rp 1,500 a liter, 10 years
    static final double MISSED_RP_PER_L = 500.0;      // fruit lost per liter missed

    public static void main(String[] args) throws Exception {
        int n = args.length > 0 ? Integer.parseInt(args[0]) : 200;
        int threads = args.length > 1 ? Integer.parseInt(args[1]) : 2;
        List<double[]> seen = new ArrayList<>();
        java.util.function.DoubleUnaryOperator cost = capacity -> {
            double[][] rows;
            try {
                rows = MonteCarlo.runSeasons(capacity, n, threads);  // same seasons
            } catch (Exception e) {
                throw new RuntimeException(e);
            }
            double kwh = MonteCarlo.mean(rows, 5);
            double missed = MonteCarlo.mean(rows, 3);
            int shortN = 0;
            for (double[] r : rows) if (r[3] > 1.0) shortN++;
            double t = TANK_RP_PER_L_YEAR * capacity, e = kwh * Orchard.TARIFF;
            double m = MISSED_RP_PER_L * missed, total = t + e + m;
            seen.add(new double[] {capacity, (double) shortN / n, t, e, m, total});
            return total;
        };
        double best = Toolbox.goldenMin(cost, 10000.0, 50000.0, 500.0);
        System.out.println(n + " seasons per size; yearly cost in thousand rupiah");
        System.out.println("  tank (L)  P(short)   tank  power  missed   total");
        for (double[] r : seen)
            System.out.println(Orchard.f(
                    "  %8.0f  %8.3f  %5.0f  %5.1f  %6.0f  %6.0f",
                    r[0], r[1], r[2] / 1e3, r[3] / 1e3, r[4] / 1e3, r[5] / 1e3));
        double total = cost.applyAsDouble(best);
        System.out.println(Orchard.f(
                "best tank about %.0f L, yearly cost %.0f thousand;"
                + " P(short) there %.3f",
                best, total / 1e3, seen.get(seen.size() - 1)[1]));
        System.out.println(Orchard.f(
                "newsvendor check: P(short) should be near %.3f",
                TANK_RP_PER_L_YEAR / MISSED_RP_PER_L));
    }
```

Kode 16.12 (Lanjutan) *Design.java: pencarian yang sama dalam Java.*

```
}
```

16.5 Menjalankan Simulasi

```
cd Chapter16/python
python3 components.py
python3 sensor.py
python3 montecarlo.py
python3 design.py
cd ../java
javac -d out *.java
java -cp out Components
java -cp out Sensor
java -cp out MonteCarlo
java -cp out Design
```

16.5.1 Bagian-bagiannya

`components.py` mencetak:

```
1. Tank: level for 6,500 liters (Chapters 2, 3, 6, 7)
   inverse spline h(V)           115.0174 cm
   Newton on V(h) = 6500         115.0144 cm in 4 steps
   bisection on V(h) = 6500      115.0144 cm
   straight lines through table  114.7929 cm
2. Pump: datasheet fit and operating point (Chapters 3, 4, 8)
   head = 31.9958 -0.023155 Q -0.00411310 Q^2  (m, Q in L/min)
   largest misfit on the datasheet 0.087 m
   operating point 59.752 L/min at 15.927 m
   1000 liters: 0.2231 kWh, Rp 322
3. Weather: long-run share of dry, light and heavy days (5, 7, 10, 12)
   power method, 40 steps   0.828813  0.138638  0.032550
   Gauss-Seidel, 13 sweeps   0.828813  0.138638  0.032550
   mean rain: light day 6.8125 mm, heavy day 41.3750 mm
   expected 2.2912 mm a day, 350.6 mm a season, 35757 L from the roof
4. A storm on the rain gauge, every 10 minutes (Chapter 10)
   trapezoid 38.9167 mm, Simpson 39.1111 mm, roof catches 3989 L
5. Drip session for 2,520 liters (Chapters 2, 11)
   start   13000 L, euler dt 10.0:  2520.00 L in 114.5825 min
   start   13000 L, rk4   dt 10.0:  2520.00 L in 114.8071 min
   start   13000 L, rk4   dt 0.5:  2520.00 L in 114.8071 min
   start    4000 L, euler dt 10.0:  2520.00 L in 146.7040 min
   start    4000 L, rk4   dt 10.0:  2520.00 L in 147.2514 min
   start    4000 L, rk4   dt 0.5:  2520.00 L in 147.2514 min
   start    3000 L, euler dt 10.0:  2460.34 L in 150.0000 min
   start    3000 L, rk4   dt 10.0:  2450.54 L in 150.0000 min
   start    3000 L, rk4   dt 0.5:  2450.54 L in 150.0000 min
```

Setiap bagian lulus pemeriksaannya. Ketinggian tandon untuk 6500 liter adalah 115,017 cm menurut spline invers dan 115,014 cm menurut Newton (empat langkah) dan biseksi pada spline maju, selisih 0,03 mm antara dua interpolan yang sama-sama wajar dari tabel ember yang sama; garis lurus melalui tabel akan meleset 2 mm, masih tidak berbahaya. Fungsi kuadratik pompa meleset dari lembar data paling banyak 9 cm *head*, dan titik kerjanya, 59,75 liter per menit pada 15,9 m, sesuai dengan hitungan tangan di Subbab 16.3.2. Metode pangkat dan Gauss–Seidel sepakat tentang cuaca sampai enam digit, dan 35.757 liter air atap yang diharapkan per musim menjadi tolok ukur bagi simulasi. Badai itu terintegrasi menjadi 38,9 mm dengan aturan trapesium dan 39,1 mm dengan aturan Simpson; selisihnya, setengah persen, sebesar galat yang diharapkan dari sampel 10 menit sebuah badai yang memuncak, dan atap menampung sekitar 4000 liter darinya.

Sesi-sesi tetes memperlihatkan fisika sekaligus numeriknya. Dari tandon penuh jatah memerlukan 114,8 menit, dari 4000 liter 147,3 menit, tepat di dalam jendela, dan dari 3000 liter jendela tertutup setelah 2450,5 liter: kurang 70 liter padahal masih ada 550 liter di tandon. Runge–Kutta dengan langkah 10 menit sama dengan langkah 0,5 menit pada semua digit yang dicetak, sedangkan metode Euler dengan langkah 10 menit meleset seperempat menit dan 10 liter, galat berorde Δt seperti diramalkan Bab 11.

16.5.2 Model tetes terhadap sensor

`sensor.py` mencetak:

```
46 readings, 0 to 90 min, level 175.5 -> 151.0 cm
step  2 min: 44 rates, K (first half) 11.819, test RMS 6.839 L/min, K (all) 11.856
step 10 min: 36 rates, K (first half) 11.910, test RMS 1.253 L/min, K (all) 11.899
model value K = 12.000 L/min per sqrt(m)
```

Dengan tetangga berjarak 2 menit, ketinggian air hanya berubah sekitar setengah sentimeter antara dua pembacaan, sebesar resolusi sensor, sehingga setiap laju kebanyakan berisi derau pembulatan: galat ujinya 6,8 liter per menit untuk laju sekitar 22. Dengan tetangga berjarak 10 menit, perubahannya lima kali lebih besar dan deraunya lima kali lebih kecil secara relatif (tawar-menawar antara pemotongan dan derau di Bab 9), dan galat ujinya turun menjadi 1,25 liter per menit. Kedua taksiran K, 11,82 sampai 11,91, berada dalam 1,5% dari nilai model, 12, dan paruh yang tidak dipakai untuk mencocokkan teramalkan sebaik paruh yang dipakai. Model tetes konsisten dengan kebun.

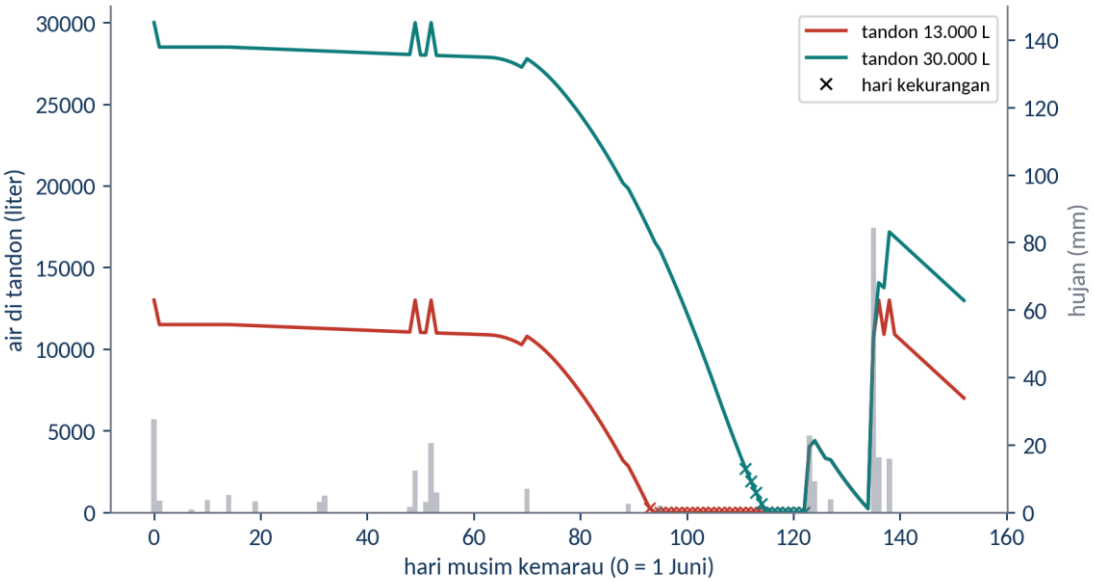
16.5.3 Empat ratus musim

`montecarlo.py` menyimulasikan 400 musim untuk tandon sekarang dan untuk tandon 30.000 liter:

```
tank 13000 L, 400 seasons:
  rain caught     35870 L, overflow     20496 L
  pumped         258005 L, electricity 57.6 kWh = Rp 83192
  trees got      280146 L, missed 14166 L on 26.2 days on average
  seasons with any shortage: 390 of 400; 9 in 10 miss less than 24446 L
tank 30000 L, 400 seasons:
  rain caught     35870 L, overflow     19543 L
  pumped         258745 L, electricity 57.7 kWh = Rp 83431
  trees got      292307 L, missed 2005 L on 5.8 days on average
  seasons with any shortage: 145 of 400; 9 in 10 miss less than 7446 L
```

Air atap, rata-rata 35.870 liter, sesuai dengan perkiraan tangan 35.757 dalam batas galat Monte Carlo: pemeriksaan pertama atas keseluruhan. Dengan tandon 13.000 liter yang sekarang, 390 dari 400 musim berakhir dengan kekurangan, dan rata-rata pohon kekurangan 14.166 liter pada 26 hari. Dengan 30.000 liter, 145 dari 400 musim mengalami kekurangan, rata-rata air yang kurang turun menjadi 2005 liter, dan pada sembilan dari sepuluh musim pohon kekurangan kurang dari 7446 liter. Listriknya hampir tidak berubah (Rp83.000 semusim): pompa mengangkat air yang kira-kira sama banyaknya berapa pun tandonnya, karena air yang diminum pohon harus datang dari suatu tempat. Lebih dari separuh air hujan meluap, karena pengisian sore hari menjaga tandon tetap penuh setiap kali sumur sanggup; Subbab 16.7 membahasnya.

Gambar 16.2 memperlihatkan salah satu dari 400 musim itu dengan kedua tandon. Sampai awal Agustus sumur masih mampu mengimbangi dan kedua tandon tetap hampir penuh. Kemudian sumur tertinggal, dan tandon-tandon mengosong dengan laju yang sama, sekitar 900 liter sehari pada September. Tandon kecil kosong pada hari ke-93 (2 September) dan pohon-pohon kekurangan setiap hari selama tiga minggu, persis pengalaman Pak Hadi tahun lalu; tandon besar bertahan sampai hari ke-111, ketika masa kering singkat sebelum hujan pertama bulan Oktober mengosongkannya juga. Hujan pertengahan Oktober mengisi keduanya kembali.



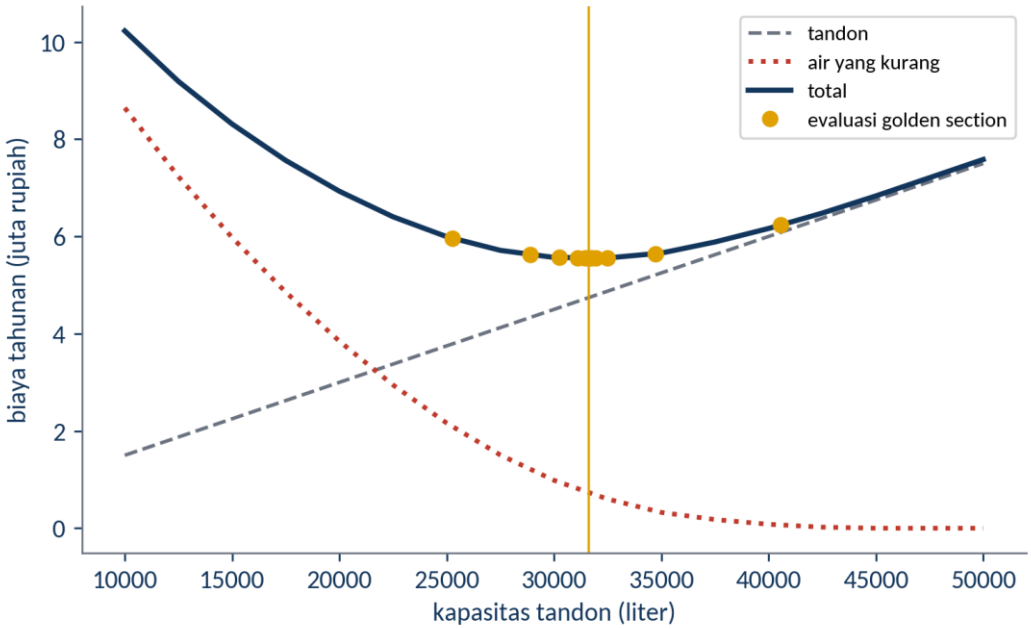
Gambar 16.2 Satu musim kemarau simulasi (cuaca yang sama) dengan tandon 13.000 liter dan 30.000 liter: air di tandon setelah penyiraman pagi, hari-hari ketika pohon mendapat kurang dari jatahnya (tanda silang), dan hujan harian (batang, sumbu kanan).

16.5.4 Tandon terbaik

`design.py` mencetak:

```
200 seasons per size; yearly cost in thousand rupiah
  tank (L)  P(short)   tank  power  missed   total
     25279     0.590   3792   83.4    2082    5958
     34721     0.175   5208   83.4     347    5639
     40557     0.060   6084   83.4      64    6231
     31115     0.315   4667   83.4     804    5555
     28885     0.415   4333   83.4    1199    5616
     32492     0.270   4874   83.4     601    5558
     30263     0.335   4539   83.4     939    5562
     31641     0.305   4746   83.4     723    5552
     31966     0.285   4795   83.4     675    5553
     31440     0.310   4716   83.4     753    5553
     31765     0.300   4765   83.4     704    5552
     31564     0.305   4735   83.4     734    5552
best tank about 31602 L, yearly cost 5552 thousand; P(short) there 0.305
newsvendor check: P(short) should be near 0.300
```

Golden-section search mencoba dua belas kapasitas dan mengendap di sekitar 31.600 liter, dengan biaya tahunan 5,55 juta rupiah: 4,7 juta untuk tandon, 0,08 juta untuk listrik, dan 0,7 juta buah yang hilang. Tandon sekarang akan berbiaya 1,95 juta setahun tetapi kehilangan buah yang diharapkan sebesar $500 \times 14{,}166 \approx 7.1$ juta, total sekitar 9 juta. **Gambar 16.3** menggambar ketiga bagian biaya pada seluruh rentang: garis tandon naik terus, kurva air yang kurang turun tajam lalu melandai, dan jumlahnya mempunyai dasar yang lebar dan datar antara sekitar 29.000 dan 34.000 liter, tempat biayanya berubah kurang dari 2%. Di antara kapasitas yang dicoba, peluang kekurangan di titik optimum adalah 0,305, seperti diramalkan aturan *newsvendor* Persamaan (16.11): simulasi dan teori satu baris saling menegaskan.



Gambar 16.3 Biaya tahunan terhadap kapasitas tandon dari 200 musim simulasi per ukuran: biaya tandon, nilai air yang kurang, dan totalnya, dengan kapasitas-kapasitas yang dicoba *golden-section search* dan ukuran yang dipilih.

Dasar yang datar itulah bagian jawaban yang paling berguna. Artinya Pak Hadi tidak perlu membeli tepat 31.602 liter, bilangan yang membawa lebih banyak digit daripada yang pantas diberikan buku catatan hujannya. Bila tersedia tandon 5.000 liter, tiga tandon lagi di samping yang sekarang memberi 28.000 liter dan empat memberi 33.000, keduanya dalam 2% dari biaya minimum. Pilihan di antara keduanya dapat didasarkan pada hal-hal yang tidak diketahui model: ruang di atas menara, uang yang tersedia tahun ini, dan seberapa jauh ia memercayai catatan bulan September.

16.6 Membandingkan dengan SciPy

Kode 16.13 menghitung ulang bagian-bagiannya dengan rutin pustaka: `CubicSpline` dengan syarat ujung alami untuk tandon, `polyfit` untuk kurva pompa, `brentq` (pencari akar berkurung dari Brent) untuk titik kerja, `eig` untuk cuaca stasioner, `simpson` untuk badai, dan `solve_ivp` dengan fungsi kejadian dan toleransi ketat untuk sesi tetes [7], [8].

Kode 16.13 `check_scipy.py`: bagian-bagian model dihitung ulang dengan NumPy dan SciPy.

```
"""The capstone's parts recomputed with NumPy and SciPy.
Usage: python check_scipy.py   (needs numpy and scipy)"""
import numpy as np
from scipy.integrate import simpson, solve_ivp
from scipy.interpolate import CubicSpline
from scipy.optimize import brentq
import orchard as o

inv = CubicSpline(o.LITRES, o.LEVELS, bc_type="natural")
print(f"tank: level for 6500 L {float(inv(6500.0)):.4f} cm"
      f" (ours {o.LEVEL(6500.0):.4f})")
q, h = np.array(o.CURVE).T
c2, c1, c0 = np.polyfit(q, h, 2)
print(f"pump: polyfit {c0:.4f} {c1:+.6f} Q {c2:+.8f} Q^2")
qop = brentq(o.pump_gap, 0.0, 100.0, xtol=1e-12)
print(f"      operating point {qop:.6f} L/min (ours {o.Q_PUMP:.6f})")
w, v = np.linalg.eig(np.array(o.P).T)
pi = np.real(v[:, np.argmin(np.abs(w - 1.0))])
print("weather: eigenvector for eigenvalue 1 "
      + "  ".join(f"{x:.6f}" for x in pi / pi.sum()))
storm = np.array(o.read_csv("storm.csv"))
print(f"storm: scipy simpson {simpson(storm[:, 1] / 60.0, x=storm[:, 0]):.4f} mm")
tank = o.Tank(13000.0)
sol = solve_ivp(lambda t, y: [-o.K * np.sqrt(tank.level(y[0]) / 100.0 + o.Z)],
                (0.0, 150.0), [13000.0], rtol=1e-10, atol=1e-8,
                events=lambda t, y: y[0] - (13000.0 - 2520.0))
print(f"drip: solve_ivp reaches 2520 L after {sol.t_events[0][0]:.4f} min"
      f" (ours {o.session(tank, 13000.0, 2520.0)[1]:.4f})")
```

```
tank: level for 6500 L 115.0174 cm (ours 115.0174)
pump: polyfit 31.9958 -0.023155 Q -0.00411310 Q^2
      operating point 59.751951 L/min (ours 59.751951)
weather: eigenvector for eigenvalue 1 0.828813  0.138638  0.032550
storm: scipy simpson 39.1111 mm
drip: solve_ivp reaches 2520 L after 114.8071 min (ours 114.8071)
```

Setiap bilangan sama dengan milik kita pada digit yang dicetak, termasuk akhir sesi tetes pada 114,8071 menit, yang ditemukan SciPy dengan metode Runge–Kutta adaptif berorde 5 dan pencari akar pada keluaran rapatnya. Kotak perkakas beberapa ratus baris ini karena itu seteliti pustaka untuk masalah ini; yang ditambahkan pustaka adalah keumuman, kemampuan adaptif, dan pengujian bertahun-tahun, dan itulah sebabnya untuk masalah baru pustaka adalah pilihan pertama dan metode yang ditulis sendiri menjadi pendapat kedua.

16.7 Membuatnya Lebih Baik

Simulasi tidak pernah selesai; ia cukup baik untuk sebuah keputusan. Empat perbaikan akan membuat simulasi ini lebih baik, dan masing-masing adalah perubahan kecil pada `orchard.py`.

- **Sisakan ruang untuk hujan.** Pengisian sore hari menjaga tandon tetap penuh, sehingga lebih dari separuh air atap meluap. Mengisi ulang hanya sampai, misalnya, 90% kapasitas pada Agustus dan September menyisakan ruang untuk badai; apakah itu membantu bergantung pada seberapa sering badai datang sebelum tandon diperlukan, dan itulah persis yang dapat dijawab simulasi. Soal Latihan 16.4 menunjukkan bahwa di sini tidak membantu.
- **Uji kepekaannya.** Bilangan-bilangan model tidak pasti: hasil sumur bulan September, peluang transisi, nilai buah yang hilang. Mengubah masing-masing sebesar 10% lalu menjalankan ulang `design.py` memperlihatkan mana yang menggeser jawaban. Untuk aturan *newsvendor* jawabannya dapat diramalkan: kapasitas terbaik ada di tempat Pr[kekurangan] $= a/p$, sehingga nilai buah yang lebih tinggi menggesernya ke tandon yang lebih besar, dan geserannya besar bila sebaran kekurangan air lebar.

- **Kurangi variansinya.** Bilangan acak bersama sudah membuat perbandingan antarkapasitas teliti. Menjalankan lebih banyak musim mengurangi galat sisanya hanya sebanding $1/\sqrt{N}$ (Persamaan (16.9)); untuk laporan akhir, 2000 musim dalam Java hanya memerlukan beberapa detik.
- **Modelkan sumur dengan lebih baik.** Hasil sumur adalah tabel lima bilangan. Bila muka air di sumur dicatat, pemulihannya setelah dipompa dapat dicocokkan sebagai persamaan diferensial tersendiri (Bab 8 dan 11), dan volume harian pompa kemudian akan bergantung pada seberapa banyak yang dipompa kemarin.

16.8 Memakainya Sungguhan

Model ini ditulis sedemikian rupa sehingga kebun lain, rumah tangga, sekolah, atau klinik kecil yang mempunyai tandon air hujan dapat memakainya dengan mengubah konstanta-konstanta bernama dan berkas-berkas datanya.

- **Tera tandon Anda.** Isilah dengan ember berukuran tertentu dan catat ketinggiannya setiap beberapa sentimeter; tabel delapan sampai sepuluh titik dan spline mengerjakan sisanya. Untuk tandon silinder biasa tabelnya berupa garis lurus dan spline mereproduksinya dengan tepat.
- **Baca lembar data pompa Anda** ke dalam `pump_curve.csv` dan ukur angkatannya dari muka air di sumur sampai puncak tandon. Periksa debit hasil hitungan sekali dengan ember dan pengukur waktu di lubang masuk tandon.
- **Simpan buku catatan hujan.** Hitung, untuk setiap bulan musim kemarau, seberapa sering hari kering diikuti hari kering, ringan, atau lebat, dan catat kedalamannya; matriks transisi dan tabel kuantil langsung berasal dari hitungan itu. Kategori BMKG [5] membuat buku catatan itu dapat dibandingkan dengan prakiraan resmi.
- **Catat air yang Anda berikan**, per pohon, per bedengan, atau per rumah tangga, bulan demi bulan, dan apa yang diberikan sumur atau air PDAM. Kedua tabel inilah, bukan hujannya, yang biasanya menentukan jawaban.
- **Tetapkan kedua harga dengan jujur**: berapa biaya satu liter tampungan per tahun, dan berapa kerugian Anda untuk satu liter air yang kurang. Aturan *newsvendor* Persamaan (16.11) kemudian memberi tahu Anda, sebelum simulasi apa pun, risiko kekurangan yang akan ditanggung tandon terbaik; simulasinya memberi tahu seberapa besar tandon itu.

Bagi rumah tangga, "air yang kurang" berarti membeli dari truk tangki. Misalkan sebuah keluarga membayar Rp150.000 untuk satu kiriman truk 5.000 liter, Rp30 per liter, dan tandon berbiaya Rp150 per liter per tahun. Maka $a/p = 150/30 = 5 > 1$: dari sisi ini tidak ada tandon di atas ukuran praktis terkecil yang sepadan dengan biayanya, dan aturan itu menyarankan membeli air ketika kurang. Aturan yang sama untuk air minum yang tidak dapat diganti berapa pun harganya (p sangat besar) menyarankan agar kekurangan dibuat sangat jarang. Model memberi saran; keluarga yang memutuskan, dan untuk air minum keselamatan didahulukan.

16.9 Metode Numerik dalam Praktik: Profitina

Platform AI Visibility Profitina, seperti kebun dalam bab ini, adalah sistem yang terdiri atas bagian-bagian, masing-masing sebuah metode numerik, yang dirakit menjadi satu *pipeline* [9]. Pemindaian sebuah situs web mengukur banyak sinyal, masing-masing dihitung dengan metodenya sendiri dan diperiksa tersendiri; sinyal-sinyal itu dinormalkan dan digabungkan dengan bobot menjadi sebuah skor, seperti tandon, pompa, dan cuaca digabungkan menjadi satu musim; dan skor itu diikuti sepanjang waktu dalam tren mingguan, dengan penjadwal yang

memindai ulang setiap merek dengan jeda menurut tingkat langganannya. Disiplin Subbab 16.2, periksa setiap bagian dengan metode yang independen sebelum memercayai keseluruhannya, adalah disiplin sebuah gerbang rilis: seluruh rangkaian uji otomatis, jauh lebih dari seribu uji, harus lulus sebelum rilis, dan rilis baru dilaporkan berhasil ketika versi yang dipasang menjawab pemeriksaan kesehatannya.

Koin utilitas PROFIT disimulasikan dengan semangat yang sama dengan kebun. Pasokannya mengikuti emisi yang meluruh disertai pembakaran, dan harganya dalam rupiah berasal dari sebuah *oracle* [9]. Bagaimana pasokan berkembang di bawah berbagai pola pemakaian adalah pertanyaan tentang model dinamis kecil yang dilangkahkan sepanjang waktu, dijalankan untuk banyak kemungkinan masa depan, dan diringkas dengan rata-rata dan rentang: Subbab 16.3.6 dan Subbab 16.3.7 dalam latar yang lain. Aturan bahwa jawaban Profitina adalah saran dan bukan jaminan sama dengan aturan paragraf terakhir bab ini: simulasi memperlihatkan apa yang mungkin terjadi di bawah asumsi yang dinyatakan, dan orang yang memiliki keputusan itulah yang memutuskan.

16.10 Kesalahan yang Sering Terjadi, dan Cara Menghindarinya

- **Merakit sebelum memeriksa.** Satu tanda yang salah dalam pencocokan pompa atau spline yang dibangun dengan kolom tertukar menghasilkan simulasi yang tetap berjalan dan mencetak bilangan yang masuk akal. Periksa setiap bagian dengan metode kedua lebih dahulu, seperti yang dilakukan `components.py`.
- **Terlalu sedikit musim, atau cuaca berbeda untuk setiap pilihan.** Membandingkan tandon 25.000 liter pada satu kumpulan 50 musim dengan tandon 30.000 liter pada kumpulan lain berarti membandingkan keberuntungan. Pakai benih yang sama untuk setiap pilihan dan musim yang cukup banyak agar galat baku Persamaan (16.9) kecil dibanding perbedaan yang Anda pedulikan.
- **Memberi benih dari jam atau dari thread.** Simulasi yang hasilnya berubah dari jalan ke jalan, atau menurut banyaknya inti, tidak dapat diuji maupun diulang. Beri benih setiap musim dari nomornya.
- **Memercayai digit yang tidak dimiliki data.** Tandon terbaik adalah "sekitar 30.000 liter", bukan 31.602. Laporkan rentang datar pada **Gambar 16.3**, bukan optimum yang eksak.
- **Hanya rata-rata.** Rata-rata air yang kurang menyembunyikan musim-musim buruk. Laporkan juga peluang terjadinya kekurangan dan sebuah persentil tinggi, seperti yang dilakukan `montecarlo.py`; keputusan bisa bergantung padanya.
- **Ukuran langkah yang disalin dari entah mana.** Sesi tetes tampak mulus, tetapi metode Euler dengan langkah 10 menit sudah meleset 10 liter. Periksa pengintegrasi dengan langkah yang lebih kecil, seperti di Subbab 16.5.1.
- **Lupa apa yang tidak dimuat model.** Hama, pompa rusak, tetangga yang mengambil dari sumur yang sama, atau musim kemarau yang tidak mirip satu pun dalam buku catatan tidak ada dalam model. Katakan hal itu ketika menyampaikan jawabannya.

16.11 Ringkasan Bab

- Sebuah pertanyaan nyata, seberapa besar tandon air yang diperlukan kebun durian untuk musim kemarau, dijawab dengan simulasi yang dibangun dari metode-metode seluruh buku (**Tabel 16.1**, **Gambar 16.1**).

- Setiap bagian dimodelkan dengan metode yang cocok dan diperiksa dengan metode kedua: spline dan Newton untuk tandon (Persamaan (16.1)), kuadrat terkecil dan Newton untuk pompa (Persamaan (16.2), Persamaan (16.3)), rantai Markov yang diselesaikan dengan metode pangkat dan Gauss–Seidel untuk cuaca (Persamaan (16.5), Persamaan (16.6)), Runge–Kutta dengan waktu kejadian untuk jalur tetes (Persamaan (16.7)), serta selisih dan kuadrat terkecil terhadap sensor tinggi air.
- Neraca air harian Persamaan (16.8) dijalankan selama 153 hari per musim dan 400 musim secara paralel, dapat diulang, dengan deret berbenih dan jumlah terkompensasi; galat Monte Carlo turun seperti $1/\sqrt{N}$ (Persamaan (16.9)).
- Tandon 13.000 liter yang sekarang kehabisan pada hampir setiap musim; *golden-section search* pada biaya tahunan Persamaan (16.10) memilih sekitar 31.600 liter, dengan optimum datar dari sekitar 29.000 sampai 34.000 liter.
- Di titik optimum peluang kekurangan adalah 0,305, seperti diramalkan aturan *newsvendor* Persamaan (16.11) dari perbandingan harga tampungan terhadap harga air yang kurang; teori dan simulasi saling menegaskan.
- Java dan Python mencetak hasil yang identik, dan SciPy sepakat dengan setiap bagian.

16.12 Soal Latihan

1. Dengan tangan, tentukan distribusi stasioner Persamaan (16.5) dari persamaan keseimbangan dua keadaan pertama dan syarat jumlahnya, lalu periksa dengan program.
2. Dengan Persamaan (16.7), perkirakan dengan tangan memakai satu langkah Euler 10 menit berapa air yang disalurkan tandon 13.000 liter yang penuh dalam 10 menit pertama. Mengapa jawabannya lebih besar daripada nilai Runge–Kutta, dan kira-kira seberapa besar?
3. Tunjukkan bahwa untuk tandon yang tabelnya berupa garis lurus melalui titik asal, $V = \alpha h$, spline kubik alami mereproduksi garis itu dengan tepat. Apa artinya bagi tandon silinder?
4. Ubah `season` agar pada Agustus dan September (hari ke-61 sampai 121) pengisian sore hari berhenti pada 90% kapasitas, lalu jalankan ulang `montecarlo.py` untuk 30.000 liter. Jelaskan perubahan pada luapan dan pada air yang kurang.
5. Nilai buah yang hilang tidak pasti. Tanpa menjalankan simulasi, pakailah aturan *newsvendor* untuk menyatakan peluang kekurangan yang seharusnya dimiliki tandon terbaik bila nilainya Rp300 atau Rp1.000 per liter, dan ke arah mana kapasitas terbaik bergeser. Lalu periksa dengan `design.py`.
6. Berapa musim yang diperlukan untuk menaksir peluang kekurangan pada 30.000 liter dengan galat baku 0,01? Berapa lama itu dengan program Java, dan dengan program Python pada dua pekerja?

16.13 Penyelesaian Soal Latihan

1. Persamaan stasioner $\pi_j = \sum_i \pi_i P_{ij}$ untuk $j = 2$ dan $j = 3$ adalah $\pi_2 = 0.10\pi_1 + 0.32\pi_2 + 0.35\pi_3$ dan $\pi_3 = 0.02\pi_1 + 0.08\pi_2 + 0.15\pi_3$. Yang kedua memberi $0.85\pi_3 = 0.02\pi_1 + 0.08\pi_2$. Yang pertama memberi $0.68\pi_2 = 0.10\pi_1 + 0.35\pi_3$. Dengan menyubstitusikan $\pi_3 = (0.02\pi_1 + 0.08\pi_2)/0.85$: $0.68\pi_2 = 0.10\pi_1 + 0.008235\pi_1 + 0.032941\pi_2$, sehingga $0.647059\pi_2 = 0.108235\pi_1$ dan $\pi_2 = 0.167273\pi_1$; lalu $\pi_3 = (0.02 + 0.08 \times 0.167273)\pi_1/0.85 = 0.039273\pi_1$. Syarat jumlah memberi $\pi_1(1 + 0.167273 + 0.039273) = 1$, sehingga $\pi_1 = 0.828813$, $\pi_2 = 0.138638$, $\pi_3 = 0.032550$, tepat nilai-nilai program.

2. Tandon penuh mempunyai $h = 200$ cm, sehingga debitnya $12\sqrt{2.0 + 1.5} = 22.45$ liter per menit, dan satu langkah Euler 10 menit menyalurkan 224,5 liter. Debit sebenarnya menurun selama 10 menit itu karena permukaan air turun (sekitar 2,6 cm dalam 10 menit ini, karena $V'(h) \approx 85$ liter per cm di dekat puncak), sehingga penyaluran yang eksak sedikit lebih kecil; Euler memakai debit di awal langkah untuk seluruh langkah dan karena itu menaksir terlalu tinggi, kira-kira sebesar setengah perubahan debit selama langkah dikali panjang langkah, beberapa persepuluh liter di sini. Sepanjang satu sesi galat-galat ini terkumpul menjadi seperempat menit, yaitu selisih waktu berakhirnya sesi Euler yang lebih awal di Subbab 16.5.1.

3. Untuk data pada garis $V = \alpha h$, semua selisih terbagi orde kedua bernilai nol, sehingga ruas kanan sistem tridiagonal spline bernilai nol, turunan-turunan kedua m_i semuanya nol, dan setiap potongan menjadi interpolan linear, yaitu garis itu sendiri. Tandon silinder mempunyai tabel yang linear, sehingga spline-nya eksak, dan kode yang sama menangani tandon silinder, meruncing, maupun tak beraturan; hanya tabel yang tak beraturan yang memerlukan kelengkungan spline.

4. Ganti baris pengisian dengan `top = 0.9 * capacity if 61 <= d <= 121 else capacity` dan `p = min(top - v, interp(MID, WELL, d))` dengan `max(p, 0.0)` untuk berjaga-jaga bila tandon sudah di atas `top`. Pada 400 musim yang sama rata-rata luapan turun dari 19.543 menjadi 16.266 liter, karena badai kini menemukan ruang sampai 3000 liter, tetapi air yang kurang justru *naik* dari 2005 menjadi 3142 liter, dan musim yang mengalami kekurangan dari 145 menjadi 200. Tandon memasuki kekurangan panjang bulan September dengan air 10% lebih sedikit, dan badai-badai yang jarang pada Agustus dan September tidak mengembalikan sebanyak itu. Dengan cuaca buku catatan ini, menyisakan ruang untuk hujan lebih banyak ruginya daripada untungnya; cara itu hanya menguntungkan di tempat badai sering datang selama masa kekurangan.

5. Aturan Persamaan (16.11) memberi Pr[kekurangan] $= 150/300 = 0.5$ untuk Rp300 dan $150/1000 = 0.15$ untuk Rp1.000. Nilai buah yang lebih rendah menoleransi kekurangan pada separuh musim, sehingga tandon terbaik lebih kecil; tabel yang dicetak `design.py` menunjukkan peluang kekurangan 0,59 pada 25.279 liter dan 0,415 pada 28.885, sehingga optimum yang baru terletak di sekitar 27.000 liter. Untuk Rp1.000 tabelnya menunjukkan 0,175 pada 34.721 dan 0,060 pada 40.557, sehingga optimum bergeser ke sekitar 35.000 liter. Menjalankan `design.py` dengan konstanta yang diubah menegaskan keduanya: 26.819 liter dengan peluang kekurangan 0,515, dan 35.085 liter dengan 0,155.

6. Dari Persamaan (16.9), SE $= \sqrt{p(1-p)/N} = 0.01$ dengan $p \approx 0.36$ memberi $N = 0.36 \times 0.64/0.0001 = 2304$ musim. Simulasi Python memerlukan sekitar 9 ms per musim pada mesin uji, sehingga 2304 musim memerlukan sekitar 21 detik waktu prosesor, sekitar 11 detik pada dua pekerja; program Java mengerjakan 400 musim untuk dua kapasitas dalam jauh kurang dari satu detik, sehingga 2304 musim memerlukan sekitar satu detik.

Daftar Pustaka

[1] S. M. Ross, *Simulation*, 6th ed. London, U.K.: Academic Press, 2023.

[2] S. C. Chapra and R. P. Canale, *Numerical Methods for Engineers*, 8th ed. New York, NY, USA: McGraw-Hill Education, 2021.

[3] R. L. Burden, J. D. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Boston, MA, USA: Cengage Learning, 2016.

[4] CNBC Indonesia, "Tarif Listrik Terbaru PLN per kWh, Berlaku Hingga September 2026," Aug. 10, 2026. [Online]. Available: https://www.cnbcindonesia.com/news/20260810101545-4-757861/tarif-listrik-terbaru-pln-per-kwh-berlaku-hingga-september-2026

[5] Badan Meteorologi, Klimatologi, dan Geofisika, "Data Prakiraan Cuaca Terbuka," accessed Sep. 2026. [Online]. Available: https://data.bmkg.go.id/prakiraan-cuaca/

[6] S. Nahmias and T. L. Olsen, *Production and Operations Analysis*, 7th ed. Long Grove, IL, USA: Waveland Press, 2015.

[7] The SciPy community, "scipy.integrate.solve_ivp," SciPy 1.18.1 documentation, 2026. [Online]. Available: https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

[8] P. Virtanen *et al.*, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[9] Profitina, "Profitina — AI Visibility (GEO) platform," 2026. [Online]. Available: https://profitina.com