Numerical Analysis and Applications with Music

Lectures on YouTube:
https://www.youtube.com/@mathtalent

Seongjai Kim

Department of Mathematics and Statistics

Mississippi State University

Mississippi State, MS 39762 USA

Email: skim@math.msstate.edu

October 19, 2026

Seongjai Kim, Professor of Mathematics, Department of Mathematics and Statistics, Mississippi State University, Mississippi State, MS 39762 USA. Email: skim@math.msstate.edu.

Prologue

Currently the lecture note is not fully grown up; other useful techniques and interesting examples would be soon incorporated.

Some questions show answers which are thankfully computed by *Mr. Sungwook Yang*, a graduate student in Aerospace Engineering, Mississippi State University.

Note: You **can** use any of my lecture notes as textbooks or references.

- Please **just** let me know what you are using.
- Any questions, suggestions, comments will be deeply appreciated. Thank you.

Seongjai Kim
October 19, 2026

Contents

Title iii

Prologue v

Table of Contents xiii

1 Mathematical Preliminaries 1

1.1. Review of Calculus . . . 2

1.1.1. Continuity and differentiation . . . 2

1.1.2. Integration . . . 7

1.1.3. Taylor's Theorem . . . 9

1.2. Review of Linear Algebra . . . 13

1.2.1. Vector Equations . . . 13

1.2.2. Invertible (nonsingular) matrices . . . 17

1.2.3. Determinants . . . 19

1.2.4. Eigenvectors and eigenvalues . . . 21

1.2.5. Vector and matrix norms . . . 23

1.3. Computer Arithmetic and Convergence . . . 26

1.3.1. Computational algorithms . . . 27

1.3.2. Big $\mathcal{O}$ and little o notation . . . 29

1.4. Programming with Matlab/Octave . . . 33

Exercises for Chapter 1 . . . 38

2 Solutions of Nonlinear Equations 41

2.1. The Bisection Method . . . 42

2.1.1. Implementation of the bisection method . . . 42

2.1.2. Error analysis for the bisection method . . . 45

2.2. Fixed-Point Iteration . . . 49

2.2.1. Existence and uniqueness of fixed points . . . 50

2.2.2. Fixed-point iteration . . . 51

2.3. Newton's Method and Its Variants . . . 56

2.3.1. The Newton's method . . . 56
2.3.2. Systems of nonlinear equations . . . 60
2.3.3. The secant method . . . 63
2.3.4. The method of false position . . . 65
2.4. Zeros of Polynomials . . . 67
2.4.1. Horner's method . . . 68
2.4.2. Complex zeros: Finding quadratic factors . . . 72
2.4.3. Bairstow's method . . . 73
Exercises for Chapter 2 . . . 78

3 Interpolation and Polynomial Approximation 81
3.1. Polynomial Interpolation . . . 82
3.1.1. Newton Form of the Interpolating Polynomials . . . 84
3.1.2. Lagrange Form of Interpolating Polynomials . . . 90
3.1.3. Polynomial interpolation error . . . 93
3.1.4. Chebyshev polynomials . . . 97
3.2. Divided Differences . . . 101
3.3. Data Approximation and Neville's Method . . . 106
3.4. Hermite Interpolation . . . 110
3.5. Spline Interpolation . . . 114
3.5.1. Runge's phenomenon . . . 114
3.5.2. Linear splines . . . 115
3.5.3. Quadratic (Second Degree) Splines . . . 117
3.5.4. Cubic splines . . . 120
3.6. Parametric Curves . . . 126
Exercises for Chapter 3 . . . 131

The 15% Rule: Turning Curiosity into Financial Freedom 134

4 Numerical Differentiation and Integration 139
4.1. Numerical Differentiation . . . 140
4.2. Richardson Extrapolation . . . 145
4.3. Numerical Integration . . . 151
4.3.1. The Trapezoid Rule . . . 152
4.3.2. Simpson's Rule . . . 155
4.3.3. Simpson's Three-Eights Rule . . . 159
4.4. Romberg Integration . . . 161
4.4.1. Recursive Trapezoid Rule . . . 161

4.4.2. The Romberg Algorithm . . . 163
4.5. Gaussian Quadrature . . . 166
4.5.1. The Method of Undetermined Coefficients . . . 166
4.5.2. Legendre Polynomials: Gauss and Gauss-Lobatto Integration . . . 169
Exercises for Chapter 4 . . . 176

5 Numerical Solution of Ordinary Differential Equations 179
5.1. Elementary Theory of Initial-Value Problems . . . 180
5.2. Taylor-Series Methods . . . 183
5.2.1. The Euler Method . . . 183
5.2.2. Higher-order Taylor Methods . . . 189
5.3. Runge-Kutta Methods . . . 192
5.3.1. Second-order Runge-Kutta method . . . 193
5.3.2. Fourth-order Runge-Kutta method . . . 195
5.3.3. Adaptive methods . . . 197
5.4. One-Step Methods: Accuracy Comparison . . . 198
5.5. Multi-step Methods . . . 201
5.6. High-Order Equations & Systems of Differential Equations . . . 205
5.7. Implicit Methods and Implicit Differential Equations . . . 212
5.7.1. Revisit of the Euler Method . . . 212
5.7.2. Implicit Differential Equations . . . 216
Exercises for Chapter 5 . . . 218

6 Approximation Theory 221
6.1. Orthogonality and the Gram-Schmidt Process . . . 222
6.1.1. Orthogonal projections . . . 222
6.1.2. The Gram-Schmidt process . . . 224
6.2. Least-Squares Approximation . . . 226
6.2.1. The Least-Squares (LS) Problem . . . 227
6.2.2. Normal Equations . . . 229
6.3. Regression Analysis . . . 231
6.3.1. Regression Line . . . 231
6.3.2. Least-Squares Fitting of Other Curves . . . 234
6.3.3. Nonlinear Regression: Linearization . . . 235
6.4. Scene Analysis with Noisy Data: RANSAC . . . 237
6.4.1. Weighted Least-Squares . . . 237
6.4.2. RANdom SAmple Consensus (RANSAC) . . . 239
6.5. Padé Rational Function Approximation . . . 243

6.6. Rank-One Matrices and Structure Tensors . . . 247
Exercises for Chapter 6 . . . 253

7 Gauss Elimination and Its Variants 257
7.1. Systems of Linear Equations . . . 258
7.1.1. Nonsingular matrices . . . 259
7.1.2. Numerical Solutions of Differential Equations . . . 260
7.2. Solutions of Triangular Systems . . . 264
7.3. Gauss Elimination . . . 267
7.4. Gauss Elimination with Pivoting . . . 274
Exercises for Chapter 7 . . . 280

8 Eigenvalues and QR Factorization 281
8.1. Preliminaries: Norms of Vectors and Matrices . . . 282
8.1.1. Vectors . . . 282
8.1.2. Vector and matrix norms . . . 283
8.1.3. Matrix norms . . . 285
8.2. The Power Method . . . 287
8.2.1. Power iteration . . . 287
8.2.2. Symmetric Power Method & Inverse Power Method . . . 291
8.3. QR Factorization by the Gram-Schmidt Process . . . 297
8.4. Householder Reflectors . . . 301
8.5. QR Factorization by the Householder Reflectors . . . 305
8.6. QR Algorithm for Eigenvalues and Eigenvectors . . . 312
8.7. Eigenvalues and Eigenvectors of Stochastic Matrices . . . 315
Exercises for Chapter 8 . . . 321

9 Principal Component Analysis 323
9.1. Singular Value Decomposition . . . 324
9.1.1. Interpretation of the SVD . . . 326
9.1.2. Properties and Computation of the SVD . . . 330
9.2. Application of the SVD for LS Problems . . . 335
9.3. Principal Component Analysis . . . 338
9.3.1. Computation of Principal Components . . . 339
9.3.2. Dimensionality Reduction and Data Compression . . . 342
Exercises for Chapter 9 . . . 344

10 Iterative Algebraic Solvers 347

10.1. Eigenvalues and Convergent Matrices . . . 348
10.1.1. Spectral radius . . . 350
10.1.2. Convergent matrices . . . 351
10.2. Basics of Iterative Algebraic Solvers . . . 352
10.2.1. Basic concepts for iterative solvers . . . 352
10.2.2. Richardson method: the simplest iteration . . . 355
10.3. Relaxation Methods . . . 357
10.4. Graph Theory: Estimation of the Spectral Radius . . . 365
10.4.1. Irreducible matrices . . . 365
10.4.2. Eigenvalue locus theorem . . . 367
10.4.3. Regular splitting and M-matrices . . . 370
10.5. Krylov Subspace Methods . . . 371
10.5.1. Gradient descent (GD) method . . . 372
10.5.2. Conjugate gradient (CG) method . . . 373
10.5.3. Preconditioned CG method . . . 377
10.6. Generalized Minimum Residuals – GMRES . . . 378
10.6.1. Arnoldi Iteration . . . 378
10.6.2. Derivation of the GMRES Method . . . 381
Exercises for Chapter 10 . . . 386

11 Mathematical Optimization **387**
11.1. Method of Lagrange Multipliers . . . 388
11.1.1. Equality Constraints . . . 388
11.1.2. Inequality Constraints . . . 389
11.2. Optimization: Primal and Dual Problems . . . 391
11.2.1. The Lagrangian . . . 391
11.2.2. Lagrange Dual Problem . . . 393
11.3. Weak Duality, Strong Duality, and Complementary Slackness . . . 396
11.3.1. Weak Duality . . . 397
11.3.2. Strong Duality . . . 398
11.3.3. Complementary Slackness . . . 399
11.4. Geometric Interpretation of Duality . . . 400
11.5. Quadratic Programming . . . 407
11.5.1. Equality-constrained quadratic programming . . . 408
11.5.2. Interior-point methods . . . 412
Exercises for Chapter 11 . . . 414

The 15% Rule: Turning Curiosity into Financial Freedom **416**

12 Introduction to Machine Learning **421**
12.1. What is Machine Learning? . . . 422
12.2. Binary Classifiers . . . 427
12.2.1. The Perceptron Algorithm . . . 429
12.2.2. Adaline: ADAptive LInear NEuron . . . 433
12.3. Logistic Regression . . . 438
12.4. Support Vector Machine . . . 442
12.4.1. Linear SVM . . . 442
12.4.2. Soft-margin classification for inseparable data . . . 448
12.4.3. Nonlinear SVM and kernel trick . . . 453
12.5. k-Nearest Neighbors . . . 457
12.6. Neural Networks . . . 459
12.7. A Machine Learning Modelcode: Scikit-Learn Comparisons and Ensembling . . . 469
Exercises for Chapter 12 . . . 474

13 Matrix Analysis in Data Mining **477**
13.1. Introduction to Data Mining . . . 478
13.2. Vectors and Matrices in Data Mining . . . 482
13.3. Text Mining . . . 487
13.3.1. The Vector Space Model: Preprocessing and Query Matching . . . 488
13.3.2. Latent Semantic Indexing . . . 494
13.4. Eigenvalue Methods in Data Mining . . . 497
13.4.1. Pagerank: Perron-Frobenius Theory . . . 498
13.4.2. The Google Matrix . . . 502
Exercises for Chapter 13 . . . 507

14 Finite Difference Methods for Boundary-Value Problems **509**
14.1. The Shooting Method . . . 510
14.2. Finite Difference Methods (FDMs) . . . 517
14.2.1. The FD discretization procedure . . . 518
14.2.2. Existence and Uniqueness of the FDM Solution . . . 522
14.3. FDMs for Non-constant Diffusion . . . 527
Exercises for Chapter 14 . . . 532

15 Finite Element Methods for Boundary-Value Problems **533**
15.1. Variational Principle: From Strong Form to Weak . . . 534
15.2. Finite Element Methods (FEMs) . . . 537
15.3. Other Variational Principles for FEMs . . . 549

15.3.1. Rayleigh-Ritz Method 549
15.3.2. Least-Squares Method 551
15.3.3. Collocation Method 552
15.4. FEMs for More General Boundary-Value Problems 556
Exercises for Chapter 15 559

16 Properties of Data-First Numerical Methods **561**
16.1. Parabolic PDEs in One Spatial Variable: Consistency 562
16.2. Convergence and Accuracy 567
16.3. Stability 572
16.4. The θ-Method: Accuracy and Efficiency 577
16.5. Data-First Numerical Methods 584
Exercises for Chapter 16 587

P Projects **589**
P.1. Applications of Richardson Extrapolation for Elliptic PDEs 590
P.2. Richardson Extrapolation for Evolutionary PDEs 598
P.3. Project Regression Analysis: Linear, Piecewise Linear, and Nonlinear Models 602
P.4. Quasi-Newton Method Using Partial Information of the Hessian 611

Bibliography **617**

Index **621**

CHAPTER 1

Mathematical Preliminaries

In this chapter, after briefly reviewing calculus and linear algebra, we study about computer arithmetic and convergence. The last section of the chapter presents a brief introduction on programming with Matlab/Octave.

Contents of Chapter 1

1.1. Review of Calculus 2
1.2. Review of Linear Algebra 13
1.3. Computer Arithmetic and Convergence 26
1.4. Programming with Matlab/Octave 33
Exercises for Chapter 1 38

1.1. Review of Calculus

1.1.1. Continuity and differentiation

Continuity

Definition 1.1. A function f is **continuous** at x_0 if

$$\lim_{x \to x_0} f(x) = f(x_0). \tag{1.1}$$

In other words, if for every $\varepsilon > 0$, there exists a $\delta > 0$ such that

$$|f(x) - f(x_0)| < \varepsilon \text{ for all } x \text{ such that } |x - x_0| < \delta. \tag{1.2}$$

Example 1.2. Examples and Discontinuities

Solution.

Ans: Jump discontinuity, infinite discontinuity, and removable discontinuity

Definition 1.3. Let $\{x_n\}_{n=1}^{\infty}$ be an infinite sequence of real numbers. This sequence has the **limit** x (converges to x), if for every $\varepsilon > 0$ there exists a positive integer N_ε such that $|x_n - x| < \varepsilon$ whenever $n > N_\varepsilon$. The notation

$$\lim_{n \to \infty} x_n = x \quad \text{or} \quad x_n \to x \text{ as } n \to \infty$$

means that the sequence $\{x_n\}_{n=1}^{\infty}$ converges to x.

Theorem 1.4. *If f is a function defined on a set X of real numbers and $x \in X$, then the following are equivalent:*

- *f is continuous at x*
- *If $\{x_n\}_{n=1}^{\infty}$ is any sequence in X converging to x, then*

$$\lim_{n \to \infty} f(x_n) = f(x).$$

Differentiation

Definition 1.5. Let f be a function defined on an open interval containing x_0. The function is **differentiable** at x_0, if

$$f'(x_0) := \lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} \tag{1.3}$$

exists. The number $f'(x_0)$ is called the **derivative** of f at x_0.

Important theorems for continuous/differentiable functions

Theorem 1.6. *If the function f is differentiable at x_0, then f is continuous at x_0.*

Note: The converse is not true.

Example 1.7. Consider $f(x) = |x|$.

Solution.

Theorem 1.8. ***(Intermediate Value Theorem; IVT)**: Suppose $f \in C[a, b]$ and K is a number between $f(a)$ and $f(b)$. Then, there exists a number $c \in (a, b)$ for which $f(c) = K$.*

Example 1.9. Show that $x^5 - 2x^3 + 3x^2 = 1$ has a solution in the interval $[0, 1]$.

Solution. Define $f(x) = x^5 - 2x^3 + 3x^2 - 1$. Then check values $f(0)$ and $f(1)$ for the IVT.

Theorem 1.10. ***(Rolle's Theorem)**: Suppose $f \in C[a, b]$ and f is differentiable on (a, b). If $f(a) = f(b)$, then there exists a number $c \in (a, b)$ such that $f'(c) = 0$.*

Mean Value Theorem (MVT)

Theorem 1.11. *Suppose $f \in C[a,b]$ and f is differentiable on (a,b). Then there exists a number $c \in (a,b)$ such that*

$$f'(c) = \frac{f(b) - f(a)}{b - a}, \tag{1.4}$$

which can be equivalently written as

$$f(b) = f(a) + f'(c)(b - a). \tag{1.5}$$

Example 1.12. Let $f(x) = x + \sin x$ be defined on [0,2]. Find c which assigns the average slope.

Solution, using Maple.

```
a := 0:
b := 2:
f := x→x + sin(x):
AS := (f(b) - f(a))/(b - a) = 1 + 1/2 sin(2)
evalf(%) = 1.454648713
EQN := f'(x) = (f(b) - f(a))/(b - a)
```

$$1 + \cos(x) = 1 + \frac{1}{2}\sin(2) \tag{7.1}$$

```
c := solve(EQN, x)
```

$$\arccos\left(\frac{1}{2}\sin(2)\right) \tag{7.2}$$

```
evalf(%) = 1.098818559
with(plots):
xx := Vector(3): xx(1) := a: xx(2) := c: xx(3) := b:
yy := Vector(3): yy(1) := f(a): yy(2) := f(c): yy(3) := f(b):
pp := plot(xx, yy, style = point, symbol = solidbox, symbolsize = 15, color = red):
L := x→f'(c)·(x - c) + f(c):
M := x→AS·(x - a) + f(a):
pf := plot([f(x), L(x), M(x)], x = a..b, thickness = [2, 1, 1], linestyle = [solid, solid,
   longdash], color = black, legend = ["f(x)", "L(x)", "Average slope"], legendstyle = [font
   = ["HELVETICA", 10], location = right]):
display({pf, pp})
```

Figure 1.1: A **maple** implementation.

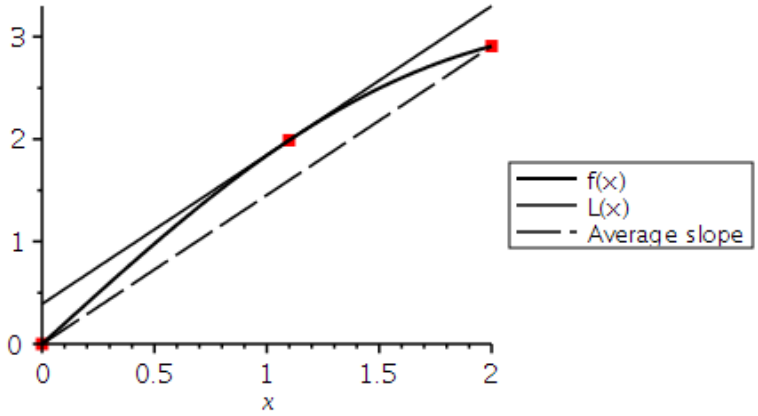


Figure 1.2: The resulting figure, from the implementation in Figure 1.1.

Theorem 1.13. ***(Extreme Value Theorem)**: If $f \in C[a,b]$, then there exist $c_1, c_2 \in [a,b]$ for $f(c_1) \leq f(x) \leq f(c_2)$ for all $x \in [a,b]$. In addition, if f is differentiable on (a,b), then the numbers c_1 and c_2 occur either at the endpoints of $[a,b]$ or where f' is zero.*

Example 1.14. Find the absolute minimum and absolute maximum values of $f(x) = 5 * \cos(2 * x) - 2 * x * \sin(2 * x)$ on the interval $[1, 2]$.

Maple-code

```
a := 1: b := 2:
f := x -> 5*cos(2*x) - 2*x*sin(2*x):
fa := f(a);
    = 5 cos(2) - 2 sin(2)
fb := f(b);
    = 5 cos(4) - 4 sin(4)

#Now, find the derivative of "f"
fp := x -> diff(f(x), x):
fp(x);
    = -12 sin(2 x) - 4 x cos(2 x)

fsolve(fp(x), x, a..b);
    1.358229874
fc := f(%);
    -5.675301338
Maximum := evalf(max(fa, fb, fc));
    = -0.241008123
Minimum := evalf(min(fa, fb, fc));
    = -5.675301338
```

```
21
22  with(plots);
23  plot([f(x), fp(x)], x = a..b, thickness = [2, 2],
24      linestyle = [solid, dash], color = black,
25      legend = ["f(x)", "f'(x)"],
26      legendstyle = [font = ["HELVETICA", 10], location = right]);
```

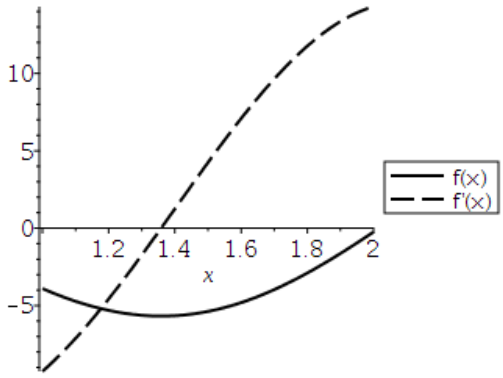


Figure 1.3: The figure from the Maple-code.

The following theorem can be derived by applying **Rolle's Theorem** successively to $f, f', \cdots$ and finally to $f^{(n-1)}$.

Theorem 1.15. ***(Generalized Rolle's Theorem)**: Suppose $f \in C[a,b]$ is n times differentiable on (a,b). If $f(x) = 0$ at the $(n+1)$ distinct points $a \leq x_0 < x_1 < \cdots < x_n \leq b$, then there exists a number $c \in (x_0, x_n)$ such that $f^{(n)}(c) = 0$.*

1.1.2. Integration

Definition 1.16. The **Riemann integral** of a function f on the interval $[a, b]$ is the following limit, provided it exists:

$$\int_a^b f(x)dx = \lim_{\max \Delta x_i \to 0} \sum_{i=1}^{n} f(x_i^*)\Delta x_i, \tag{1.6}$$

where $a = x_0 < x_1 < \cdots < x_n = b$, with $\Delta x_i = x_i - x_{i-1}$ and x_i^* arbitrarily chosen in the subinterval $[x_{i-1}, x_i]$.

Note: Continuous functions are Riemann integrable, which allows us to choose, for computational convenience, the points x_i to be equally spaced in $[a, b]$ and choose $x_i^* = x_i$, where $x_i = a + i\Delta x$, $\Delta x = \dfrac{b-a}{n}$. In this case,

$$\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n} f(x_i)\Delta x. \tag{1.7}$$

Theorem 1.17. *(**Fundamental Theorem of Calculus**; **FTC**): Let f be continuous on $[a, b]$. Then,*

Part I: $\dfrac{d}{dx}\displaystyle\int_a^x f(t)dt = f(x)$.

Part II: $\displaystyle\int_a^b f(x)dx = F(b) - F(a)$, *where F is an **antiderivative** of f, i.e. $F' = f$.*

Example 1.18. Use **The Fundamental Theorem of Calculus** to find

$$\lim_{x\to 0} \frac{1}{x} \int_0^{\sin 2x} \sqrt{t^3 + 4}\, dt$$

Solution.

Ans: 4.

Weighted Mean Value Theorem on Integral (WMVT)

Theorem 1.19. *Suppose $f \in C[a,b]$, the Riemann integral of g exists on $[a,b]$, and $g(x)$ does not change sign on $[a,b]$. Then, there exists a number $c \in (a,b)$ such that*

$$\int_a^b f(x)g(x)dx = f(c)\int_a^b g(x)dx. \tag{1.8}$$

Example 1.20. Prove or disprove.

1. There exists a number $\xi \in (-\pi, \pi)$ such that
$$\int_{-\pi}^{\pi} \ln(6x+30)\sin x\,dx = \ln(6\xi+30)\int_{-\pi}^{\pi} \sin x\,dx.$$
2. There exists a number $\xi \in (x_0, x_1)$, $x_0 < x_1$, such that
$$\int_{x_0}^{x_1} \sin^2 x(x-x_0)(x-x_1)\,dx = \sin^2\xi \int_{x_0}^{x_1} (x-x_0)(x-x_1)\,dx.$$

Solution.

Remark 1.21. When $g(x) \equiv 1$, the WMVT becomes the usual **Mean Value Theorem on Integral**, which gives the average value of $f \in C[a,b]$ over the interval $[a,b]$:

$$f(c) = \frac{1}{b-a}\int_a^b f(x)dx. \tag{1.9}$$

1.1.3. Taylor's Theorem

Theorem 1.22. ***(Taylor's Theorem with Lagrange Remainder):*** *Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b), and $x_0 \in [a,b]$. Then, for every $x \in [a,b]$,*

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \mathcal{R}_n(x), \tag{1.10}$$

where, for some ξ between x and x_0,

$$\mathcal{R}_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}.$$

Example 1.23. Let $f(x) = cos(x)$ and $x_0 = 0$. Determine the second and third Taylor polynomials for f about x_0.

Maple-code

```
f := x -> cos(x):
fp := x -> -sin(x):
fpp := x -> -cos(x):
fp3 := x -> sin(x):
fp4 := x -> cos(x):

p2 := x -> f(0) + fp(0)*x/1! + fpp(0)*x^2/2!:
p2(x);
    = 1 - 1/2 x^2
R2 := fp3(xi)*x^3/3!;
    = 1/6 sin(xi) x^3
p3 := x -> f(0) + fp(0)*x/1! + fpp(0)*x^2/2! + fp3(0)*x^3/3!:
p3(x);
    =  1 - 1/2 x^2
R3 := fp4(xi)*x^4/4!;
    = 1/24 cos(xi) x^4

# On the other hand, you can find the Taylor polynomials easily
# using built-in functions in Maple:
s3 := taylor(f(x), x = 0, 4);
    = 1 - 1/2 x^2 + O(x^4)
convert(s3, polynom);
    = 1 - 1/2 x^2
```

```
plot([f(x), p3(x)], x = -2 .. 2, thickness = [2, 2],
    linestyle = [solid, dash], color = black,
    legend = ["f(x)", "p3(x)"],
    legendstyle = [font = ["HELVETICA", 10], location = right])
```

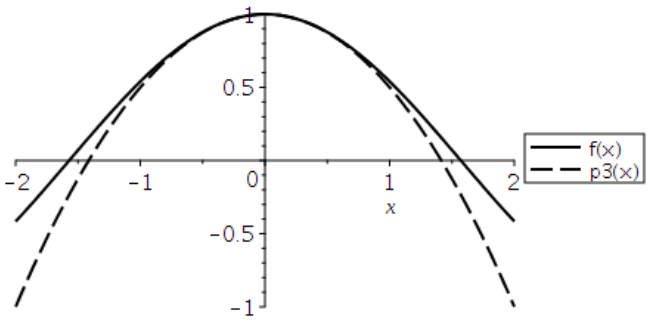


Figure 1.4: $f(x) = \cos x$ and its third Taylor polynomial $P_3(x)$.

Note: When $n = 0, x = b$, and $x_0 = a$, the Taylor's Theorem reads

$$f(b) = f(a) + \mathcal{R}_0(b) = f(a) + f'(\xi) \cdot (b - a), \tag{1.11}$$

which is the **Mean Value Theorem**.

Theorem 1.24. ***(Taylor's Theorem with Integral Remainder)**: Suppose $f \in C^n[a, b]$ and $x_0 \in [a, b]$. Then, for every $x \in [a, b]$,*

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + \mathcal{E}_n(x), \tag{1.12}$$

where

$$\mathcal{E}_n(x) = \frac{1}{n!} \int_{x_0}^{x} f^{(n+1)}(t) \cdot (x - t)^n dt.$$

Alternative Form of Taylor's Theorem

Remark 1.25. Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b). Then, for every $x,\ x+h \in [a,b]$,

$$f(x+h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!} h^k + \mathcal{R}_n(h), \tag{1.13}$$

where, for some ξ between x and $x+h$,

$$\mathcal{R}_n(h) = \frac{f^{(n+1)}(\xi)}{(n+1)!} h^{n+1}.$$

In detail,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \cdots + \frac{f^{(n)}(x)}{n!}h^n + \mathcal{R}_n(h). \tag{1.14}$$

Theorem 1.26. ***(Taylor's Theorem for Two Variables)****: Let $f \in C^{(n+1)}([a,b]\times[c,d])$. If (x,y) and $(x+h, y+k)$ are points in $[a,b]\times[c,d] \subset \mathbb{R}^2$, then*

$$f(x+h, y+k) = \sum_{i=0}^{n} \frac{1}{i!}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right)^i f(x,y) + \mathcal{R}_n(h,k), \tag{1.15}$$

where

$$\mathcal{R}_n(h,k) = \frac{1}{(n+1)!}\left(h\frac{\partial}{\partial x} + k\frac{\partial}{\partial y}\right)^{n+1} f(x+\theta h, y+\theta k),$$

in which $\theta \in [0,1]$.

Note: For $n=1$, the Taylor's theorem for two variables reads

$$f(x+h, y+k) = f(x,y) + h\, f_x(x,y) + k\, f_y(x,y) + \mathcal{R}_1(h,k), \tag{1.16}$$

where

$$\mathcal{R}_1(h,k) = \mathcal{O}(h^2 + k^2).$$

Equation (1.16), as a **linear approximation** or **tangent plane approximation**, will be used for various applications.

Example 1.27. Find the tangent plane approximation of

$$f(x, y) = \frac{2x + 3}{4y + 1} \text{ at } (0, 0).$$

Maple-code

```
f := (2*x + 3)/(4*y + 1):
f0 := eval(%, {x = 0, y = 0});
    = 3
fx := diff(f, x);
    = 2/(4*y + 1)
fx0 := eval(%, {x = 0, y = 0});
    = 2
fy := diff(f, y);
    = 4*(2*x + 3)/(4*y + 1)^2
fy0 := eval(%, {x = 0, y = 0});
    = -12

# Thus the tangent plane approximation $L(x, y)$ at $(0, 0)$ is
    L(x, y) = 3 + 2*x - 12*y
```

1.2. Review of Linear Algebra

1.2.1. Vector Equations

Vectors

Definition 1.28. Let $\mathbf{u} = [u_1, u_2, \cdots, u_n]^T$ and $\mathbf{v} = [v_1, v_2, \cdots, v_n]^T$ are vectors in $\mathbb{R}^n$. Then, the **inner product** (or **dot product**) of $\mathbf{u}$ and $\mathbf{v}$ is given by

$$\begin{aligned} \mathbf{u}\bullet\mathbf{v} &= \mathbf{u}^T\mathbf{v} = [u_1\ u_2\ \cdots\ u_n]\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \\ &= u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{k=1}^{n} u_k v_k. \end{aligned} \tag{1.17}$$

Definition 1.29. The **length** (**Euclidean norm**) of $\mathbf{v}$ is nonnegative scalar $\|\mathbf{v}\|$ defined by

$$\|\mathbf{v}\| = \sqrt{\mathbf{v}\bullet\mathbf{v}} = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \quad \text{and} \quad \|\mathbf{v}\|^2 = \mathbf{v}\bullet\mathbf{v}. \tag{1.18}$$

Definition 1.30. For $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, the **distance** between $\mathbf{u}$ and $\mathbf{v}$ is

$$dist(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|, \tag{1.19}$$

the length of the vector $\mathbf{u} - \mathbf{v}$.

Example 1.31. Let $\mathbf{u} = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix}$. Find $\mathbf{u}\bullet\mathbf{v}$, $\|\mathbf{u}\|$, and $dist(\mathbf{u}, \mathbf{v})$.

Solution.

Definition **1.32.** Two vectors $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{R}^n$ are **orthogonal** if $\mathbf{u} \bullet \mathbf{v} = 0$.

Theorem **1.33.** ***Pythagorean Theorem****: Two vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *are orthogonal if and only if*

$$\|\mathbf{u} + \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2. \tag{1.20}$$

Note: The **inner product** can be defined as

$$\mathbf{u} \bullet \mathbf{v} = \|\mathbf{u}\| \, \|\mathbf{v}\| \cos\theta, \tag{1.21}$$

where θ is the angle between $\mathbf{u}$ and $\mathbf{v}$.

Example **1.34.** Let $\mathbf{u} = \begin{bmatrix} 1 \\ \sqrt{3} \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} -1/2 \\ \sqrt{3}/2 \end{bmatrix}$. Use (1.21) to find the angle between $\mathbf{u}$ and $\mathbf{v}$.

Solution.

System of Linear Equations

Linear systems of m equations of n unknowns can be expressed as the algebraic system:

$$A\mathbf{x} = \mathbf{b}, \tag{1.22}$$

where $\mathbf{b} \in \mathbb{R}^m$ is the source (input), $\mathbf{x} \in \mathbb{R}^n$ is the solution (output), and

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \in \mathbb{R}^{m\times n}.$$

The above algebraic system can be solved by the *elementary row operations* applied to the **augmented matrix**, **augmented system**:

$$[A \ \mathbf{b}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & \cdots & a_{2n} & b_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} & b_m \end{bmatrix}, \tag{1.23}$$

and by transforming it to the **reduced echelon form**.

Tools **1.35. Three Elementary Row Operations (ERO)**:

- ***Replacement***: Replace one row by the sum of itself and a multiple of another row
  $$R_i \leftarrow R_i + k \cdot R_j, \quad j \neq i$$
- ***Interchange***: Interchange two rows
  $$R_i \leftrightarrow R_j, \quad j \neq i$$
- ***Scaling***: Multiply all entries in a row by a nonzero constant
  $$R_i \leftarrow k \cdot R_i, \quad k \neq 0$$

Every elementary row operation can be expressed as a matrix to be left-multiplied. Such a matrix is called an **elementary matrix**.

Example 1.36. Solve the following system of linear equations, using the 3 EROs. Then, determine if the system is consistent.

$$\begin{aligned} 4x_2 + 2x_3 &= 6 \\ x_1 - 4x_2 + 2x_3 &= -1 \\ 4x_1 - 8x_2 + 12x_3 &= 8 \end{aligned}$$

Solution.

Example 1.37. Find the parabola $y = a_0 + a_1x + a_2x^2$ that passes through $(1, 1)$, $(2, 2)$, and $(3, 5)$.

Solution.

Ans: $y = 2 - 2x + x^2$

1.2.2. Invertible (nonsingular) matrices

Definition **1.38.** An $n \times n$ matrix A is said to be **invertible (nonsingular)** if there is an $n \times n$ matrix B such that $AB = I_n = BA$, where I_n is the identity matrix.

Note: In this case, B is the *unique inverse* of A denoted by A^{-1}.
(Thus $AA^{-1} = I_n = A^{-1}A$.)

Theorem **1.39.** **(Inverse of an $n \times n$ matrix, $n \geq 2$)** *An $n \times n$ matrix A is invertible if and only if A is row equivalent to I_n and in this case any sequence of elementary row operations that reduces A into I_n will also reduce I_n to A^{-1}.*

Algorithm **1.40.** *Algorithm to find A^{-1}:*

1) Row reduce the augmented matrix $[A : I_n]$

2) If A is row equivalent to I_n, then $[A : I_n]$ is row equivalent to $[I_n : A^{-1}]$. Otherwise A does not have any inverse.

Example **1.41.** Find the inverse of $A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 3 \\ 4 & -3 & 8 \end{bmatrix}$, if it exists.

Solution.

Theorem 1.42.

a. **(Inverse of a 2×2 matrix)** *Let* $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$. *If* $ad - bc \neq 0$, *then* A *is invertible and*

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix} \tag{1.24}$$

b. *If* A *is an invertible matrix, then* A^{-1} *is also invertible;* $(A^{-1})^{-1} = A$.

c. *If* A *and* B *are* $n \times n$ *invertible matrices then* AB *is also invertible and* $(AB)^{-1} = B^{-1}A^{-1}$.

d. *If* A *is invertible, then* A^T *is also invertible and* $(A^T)^{-1} = (A^{-1})^T$.

e. *If* A *is an* $n \times n$ *invertible matrix, then for each* $\mathbf{b} \in \mathbb{R}^n$, *the equation* $A\mathbf{x} = \mathbf{b}$ *has a unique solution* $\mathbf{x} = A^{-1}\mathbf{b}$.

Theorem 1.43. (Invertible Matrix Theorem) *Let* A *be an* $n \times n$ *matrix. Then the following are equivalent.*

a. A *is an invertible matrix.*

b. A *is row equivalent to the* $n \times n$ *identity matrix.*

c. A *has* n *pivot positions.*

d. *The columns of* A *are linearly independent.*

e. *The equation* $A\mathbf{x} = \mathbf{0}$ *has only the trivial solution* $\mathbf{x} = 0$.

f. *The equation* $A\mathbf{x} = \mathbf{b}$ *has unique solution for each* $\mathbf{b} \in \mathbb{R}^n$.

g. *The linear transformation* $\mathbf{x} \mapsto A\mathbf{x}$ *is one-to-one.*

h. *The linear transformation* $\mathbf{x} \mapsto A\mathbf{x}$ *maps* $\mathbb{R}^n$ *onto* $\mathbb{R}^n$.

i. *There is a matrix* $C \in \mathbb{R}^{n \times n}$ *such that* $CA = I$

j. *There is a matrix* $D \in \mathbb{R}^{n \times n}$ *such that* $AD = I$

k. A^T *is invertible and* $(A^T)^{-1} = (A^{-1})^T$.

l. *The number* 0 *is not an eigenvalue of* A.

m. *det* $A \neq 0$.

1.2.3. Determinants

Definition **1.44.** Let A be an $n \times n$ square matrix. Then **determinant** is a scalar value denoted by $\textit{det}\, A$ or $|A|$.

1) Let $A = [a] \in \mathbb{R}^{1\times 1}$. Then $\textit{det}\, A = a$.

2) Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbb{R}^{2\times 2}$. Then $\textit{det}\, A = ad - bc$.

Example **1.45.** Let $A = \begin{bmatrix} 2 & 1 \\ 0 & 3 \end{bmatrix}$. Consider a linear transformation $T : \mathbb{R}^2 \to \mathbb{R}^2$ defined by $T(\mathbf{x}) = A\mathbf{x}$.

a. Find the determinant of A.
b. Determine the image of a rectangle $R = [0, 2] \times [0, 1]$ under T.
c. Find the area of the image.
d. Figure out how $\textit{det}\, A$, the area of the rectangle ($= 2$), and the area of the image are related.

Solution.

Ans: c. 12

Note: The determinant can be viewed as the **volume scaling factor**.

Definition 1.46. Let A_{ij} be the *submatrix* of A obtained by deleting row i and column j of A. Then the (i,j)**-cofactor** of $A = [a_{ij}]$ is the scalar C_{ij}, given by

$$C_{ij} = (-1)^{i+j} \det A_{ij}. \tag{1.25}$$

Definition 1.47. For $n \geq 2$, the **determinant** of an $n \times n$ matrix $A = [a_{ij}]$ is given by the following formulas:

1. The *cofactor expansion* across the first row:

$$\det A = a_{11}C_{11} + a_{12}C_{12} + \cdots + a_{1n}C_{1n} \tag{1.26}$$

2. The *cofactor expansion* across the row i:

$$\det A = a_{i1}C_{i1} + a_{i2}C_{i2} + \cdots + a_{in}C_{in} \tag{1.27}$$

3. The *cofactor expansion* down the column j:

$$\det A = a_{1j}C_{1j} + a_{2j}C_{2j} + \cdots + a_{nj}C_{nj} \tag{1.28}$$

Example 1.48. Find the determinant of $A = \begin{bmatrix} 1 & 5 & 0 \\ 2 & 4 & -1 \\ 0 & -2 & 0 \end{bmatrix}$, by expanding across the first row and down column 3.

Solution.

Ans: -2

1.2.4. Eigenvectors and eigenvalues

Definition 1.49. Let A be an $n \times n$ matrix. An **eigenvector** of A is a *nonzero* vector $\mathbf{x}$ such that $A\mathbf{x} = \lambda\mathbf{x}$ for some scalar λ. In this case, a scalar λ is an **eigenvalue** and $\mathbf{x}$ is the *corresponding* **eigenvector**.

Definition 1.50. The scalar equation $det\,(A - \lambda I) = 0$ is called the **characteristic equation** of A; the polynomial $p(\lambda) = det\,(A - \lambda I)$ is called the **characteristic polynomial** of A. The solutions of $det\,(A - \lambda I) = 0$ are the **eigenvalues** of A.

Example 1.51. Find the characteristic polynomial and all eigenvalues of

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 6 & 0 & 5 \\ 0 & 0 & 2 \end{bmatrix}$$

Solution.

Remark 1.52. Let A be an $n \times n$ matrix. Then the characteristic equation of A is of the form

$$\begin{aligned} p(\lambda) = det\,(A - \lambda I) &= (-1)^n \left(\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0\right) \\ &= (-1)^n \prod_{i=1}^{n} (\lambda - \lambda_i), \end{aligned} \tag{1.29}$$

where some of eigenvalues λ_i can be complex-valued numbers. Thus

$$det\,A = p(0) = (-1)^n \prod_{i=1}^{n} (0 - \lambda_i) = \prod_{i=1}^{n} \lambda_i. \tag{1.30}$$

That is, $det\,A$ is the product of all eigenvalues of A.

Theorem 1.53. *If $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_r$ are eigenvectors that correspond to distinct eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_r$ of $n \times n$ matrix A, then the set $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_r\}$ is linearly independent.*

Proof.

- Assume that $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_r\}$ is *linearly dependent*.
- One of the vectors in the set is a linear combination of the preceding vectors.
- $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ is linearly independent; $\mathbf{v}_{p+1}$ is a linear combination of the preceding vectors.
- Then, there exist scalars $c_1, c_2, \cdots, c_p$ such that

$$c_1 \mathbf{v}_1 + c_2 \mathbf{v}_2 + \cdots + c_p \mathbf{v}_p = \mathbf{v}_{p+1} \tag{1.31}$$

- Multiplying both sides of (1.31) by A, we obtain

$$c_1 A\mathbf{v}_1 + c_2 A\mathbf{v}_2 + \cdots + c_p A\mathbf{v}_p = A\mathbf{v}_{p+1}$$

and therefore, using the fact $A\mathbf{v_k} = \lambda_k \mathbf{v_k}$:

$$c_1\lambda_1\mathbf{v}_1 + c_2\lambda_2\mathbf{v}_2 + \cdots + c_p\lambda_p\mathbf{v}_p = \lambda_{p+1}\mathbf{v}_{p+1} \tag{1.32}$$

- Multiplying both sides of (1.31) by λ_{p+1} and subtracting the result from (1.32), we have

$$c_1(\lambda_1 - \lambda_{p+1})\mathbf{v}_1 + c_2(\lambda_2 - \lambda_{p+1})\mathbf{v}_2 + \cdots + c_p(\lambda_p - \lambda_{p+1})\mathbf{v}_p = \mathbf{0}. \tag{1.33}$$

- Since $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ is linearly independent,

$$c_1(\lambda_1 - \lambda_{p+1}) = 0, \;\; c_2(\lambda_2 - \lambda_{p+1}) = 0, \;\; \cdots, \;\; c_p(\lambda_p - \lambda_{p+1}) = 0.$$

- Since $\lambda_1, \lambda_2, \cdots, \lambda_r$ are distinct,

$$c_1 = c_2 = \cdots = c_p = 0 \;\; \Rightarrow \;\; \mathbf{v}_{p+1} = \mathbf{0},$$

which is a contradiction.

□

1.2.5. Vector and matrix norms

Definition 1.54. A **norm** (or, **vector norm**) on $\mathbb{R}^n$ is a function that assigns to each $\mathbf{x} \in \mathbb{R}^n$ a nonnegative real number $\|\mathbf{x}\|$ such that the following three properties are satisfied: for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$,

$$\begin{aligned} &\|\mathbf{x}\| > 0 \text{ if } \mathbf{x} \neq 0 && \text{(positive definiteness)} \\ &\|\lambda \mathbf{x}\| = |\lambda| \, \|\mathbf{x}\| && \text{(homogeneity)} \\ &\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| && \text{(triangle inequality)} \end{aligned} \tag{1.34}$$

Example 1.55. The most common norms are

$$\|\mathbf{x}\|_p = \Big(\sum_i |x_i|^p \Big)^{1/p}, \quad 1 \leq p < \infty, \tag{1.35}$$

which we call the p**-norms**, and

$$\|\mathbf{x}\|_\infty = \max_i |x_i|, \tag{1.36}$$

which is called the **infinity-norm** or **maximum-norm**.

Note: Two of frequently used p-norms are

$$\|\mathbf{x}\|_1 = \sum_i |x_i|, \quad \|\mathbf{x}\|_2 = \Big(\sum_i |x_i|^2 \Big)^{1/2} \tag{1.37}$$

The 2-norm is also called the Euclidean norm, often denoted by $\|\cdot\|$.

Example 1.56. One may consider the infinity-norm as the limit of p-norms, as $p \to \infty$.

Solution.

Definition 1.57. A **matrix norm** on $m \times n$ matrices is a vector norm on the mn-dimensional space, satisfying

$$\begin{aligned} &\|A\| \geq 0, \text{ and } \|A\| = 0 \;\Leftrightarrow\; A = 0 && \text{(positive definiteness)} \\ &\|\lambda\, A\| = |\lambda|\, \|A\| && \text{(homogeneity)} \\ &\|A + B\| \leq \|A\| + \|B\| && \text{(triangle inequality)} \end{aligned} \tag{1.38}$$

Example 1.58. $\|A\|_F \equiv \Big(\sum_{i,j} |a_{ij}|^2 \Big)^{1/2}$ is called the **Frobenius norm**.

Definition 1.59. *Once a vector norm $||\cdot||$ has been specified, the **induced matrix norm** is defined by*

$$\|A\| = \max_{\mathbf{x} \neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{1.39}$$

It is also called an **operator norm** *or* **subordinate norm**.

Theorem 1.60.

a. For all operator norms and the Frobenius norm,

$$\|Ax\| \leq \|A\|\, \|x\|, \quad \|A\, B\| \leq \|A\|\, \|B\|. \tag{1.40}$$

b. $\|A\|_1 \equiv \max_{\mathbf{x} \neq 0} \dfrac{\|A\mathbf{x}\|_1}{\|\mathbf{x}\|_1} = \max_j \sum_i |a_{ij}|$

c. $\|A\|_\infty \equiv \max_{\mathbf{x} \neq 0} \dfrac{\|A\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} = \max_i \sum_j |a_{ij}|$

d. $\|A\|_2 \equiv \max_{\mathbf{x} \neq 0} \dfrac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\lambda_{\max}(A^T A)}$,
where $\lambda_{\max}$ denotes the largest eigenvalue.

e. $\|A\|_2 = \|A^T\|_2$.

f. $\|A\|_2 = \max_i |\lambda_i(A)|$, *when $A^T A = A A^T$ (**normal matrix**).*

Definition **1.61.** *Let $A \in \mathbb{R}^{n \times n}$. Then*

$$\kappa(A) \equiv \|A\| \, \|A^{-1}\|$$

*is called the **condition number** of A, associated to the matrix norm.*

Example **1.62.** Let $A = \begin{bmatrix} 1 & 2 & -2 \\ 0 & 4 & 1 \\ 1 & -2 & 2 \end{bmatrix}$. Then, we have

$$A^{-1} = \frac{1}{20} \begin{bmatrix} 10 & 0 & 10 \\ 1 & 4 & -1 \\ -4 & 4 & 4 \end{bmatrix} \quad \text{and} \quad A^T A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 24 & -4 \\ 0 & -4 & 9 \end{bmatrix}.$$

a. Find $\|A\|_1$, $\|A\|_\infty$, and $\|A\|_2$.

b. Compute the ℓ^1-condition number $\kappa_1(A)$.

Solution.

1.3. Computer Arithmetic and Convergence

Errors in Machine Numbers and Computational Results

- Numbers are saved with an approximation by either rounding or chopping.
  - integer: in 4 bites (32 bits)
  - float: in 4 bites
  - double: in 8 bites (64 bits)
- Computations can be carried out only for finite sizes of data points.

Maple-code

```
Pi;
    = Pi
evalf(Pi);
    = 3.141592654
evalf[8](Pi);
    = 3.1415927
evalf[16](Pi);
    = 3.141592653589793
evalf(Pi)*(evalf[8](Pi) - evalf[16](Pi));
    = 1.445132621*E-07
#On the other hand,
Pi*(Pi - Pi);
    = 0
```

Definition 1.63. Suppose that p^* is an approximation to p. Then

- The **absolute error** is $|p - p^*|$, and
- the **relative error** is $\dfrac{|p - p^*|}{|p|}$, provided that $p \neq 0$.

Definition 1.64. The number p^* is said to approximate p to t-**significant digits** (or figures) if t is the largest nonnegative integer for which

$$\frac{|p - p^*|}{|p|} \leq 5 \times 10^{-t}.$$

1.3.1. Computational algorithms

Definition 1.65. An **algorithm** is a procedure that describes, in an unambiguous manner, a finite sequence of steps to be carried out in a specific order.

Algorithms consist of various steps for inputs, outputs, and functional operations, which can be described effectively by a so-called **pseudocode**.

Definition 1.66. An algorithm is called **stable**, if small changes in the initial data produce correspondingly small changes in the final results. Otherwise, it is called **unstable**. Some algorithms are stable only for certain choices of data/parameters, and are called **conditionally stable**.

Notation 1.67. (Growth rates of the error): Suppose that $E_0 > 0$ denotes an error introduced at *some* stage in the computation and E_n represents the magnitude of the error after n subsequent operations.

- If $E_n = C \times n\, E_0$, where C is a constant independent of n, then the growth of error is said to be **linear**, for which the algorithm is stable.
- If $E_n = C^n\, E_0$, for some $C > 1$, then the growth of error is **exponential**, which turns out unstable.

Rates (Orders) of Convergence

Definition 1.68. Let $\{x_n\}$ be a sequence of real numbers tending to a limit x^*.

- The rate of convergence is at least **linear** if there are a constant $c_1 < 1$ and an integer N such that
$$|x_{n+1} - x^*| \leq c_1 |x_n - x^*|, \ \forall \ n \geq N. \tag{1.41}$$
- We say that the rate of convergence is at least **superlinear** if there exist a sequence ε_n tending to 0 and an integer N such that
$$|x_{n+1} - x^*| \leq \varepsilon_n |x_n - x^*|, \ \forall \ n \geq N. \tag{1.42}$$
- The rate of convergence is at least **quadratic** if there exist a constant C (not necessarily less than 1) and an integer N such that
$$|x_{n+1} - x^*| \leq C |x_n - x^*|^2, \ \forall \ n \geq N. \tag{1.43}$$
- In general, we say that the rate of convergence is **of α at least** if there exist a constant C (not necessarily less than 1 for $\alpha > 1$) and an integer N such that
$$|x_{n+1} - x^*| \leq C |x_n - x^*|^\alpha, \ \forall \ n \geq N. \tag{1.44}$$

Example 1.69. Consider a sequence defined recursively as
$$x_1 = 2, \quad x_{n+1} = \frac{x_n}{2} + \frac{1}{x_n}. \tag{1.45}$$

(a) Find the limit of the sequence; (b) show that the convergence is quadratic.

Hint: You may first prove that $x_n > \sqrt{2}$ for all $n \geq 1$ ($\because x_{n+1}^2 - 2 > 0$). Then you can see that $x_{n+1} < x_n$ ($\because x_n - x_{n+1} = x_n(\frac{1}{2} - \frac{1}{x_n^2}) > 0$).

Solution.

1.3.2. Big $\mathcal{O}$ and little o notation

Definition 1.70.

- A sequence $\{\alpha_n\}_{n=1}^{\infty}$ is said to be **in $\mathcal{O}$ (big Oh)** of $\{\beta_n\}_{n=1}^{\infty}$ if a positive number K exists for which

$$|\alpha_n| \le K|\beta_n|, \text{ for large } n \left(\text{or equivalently, } \frac{|\alpha_n|}{|\beta_n|} \le K\right). \tag{1.46}$$

  In this case, we say "α_n is in $\mathcal{O}(\beta_n)$" and denote $\alpha_n \in \mathcal{O}(\beta_n)$ or $\alpha_n = \mathcal{O}(\beta_n)$.
- A sequence $\{\alpha_n\}$ is said to be **in o (little oh)** of $\{\beta_n\}$ if there exists a sequence ε_n tending to 0 such that

$$|\alpha_n| \le \varepsilon_n|\beta_n|, \text{ for large } n \left(\text{or equivalently, } \lim_{n\to\infty} \frac{|\alpha_n|}{|\beta_n|} = 0\right). \tag{1.47}$$

  In this case, we say "α_n is in $o(\beta_n)$" and denote $\alpha_n \in o(\beta_n)$ or $\alpha_n = o(\beta_n)$.

Example 1.71. Show that $\alpha_n = \dfrac{n+1}{n^2} = \mathcal{O}\left(\dfrac{1}{n}\right)$ and

$$f(n) = \frac{n+3}{n^3+20n^2} \in \mathcal{O}(n^{-2}) \cap o(n^{-1}).$$

Solution.

Big-O and little-o: Body and Sweat YouTube (female voice)

Big-O and little-o: Body and Sweat YouTube (male voice)

Definition 1.72. Suppose $\lim_{h\to 0} G(h) = 0$. A quantity $F(h)$ is said to be **in $\mathcal{O}$ (big Oh)** of $G(h)$ if a positive number K exists for which

$$\frac{|F(h)|}{|G(h)|} \leq K, \quad \text{for } h \text{ sufficiently small.} \tag{1.48}$$

In this case, we say $F(h)$ is in $\mathcal{O}(G(h))$, and denote $F(h) \in \mathcal{O}(G(h))$. **Little oh of $G(h)$** can be defined the same way as for sequences.

Example 1.73. Taylor's series expansion for $\cos(x)$ is given as

$$\begin{aligned} \cos(x) &= 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \cdots \\ &= 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 + \cdots . \end{aligned}$$

If you use a computer algebra software (e.g. Maple), you will obtain

$$\texttt{taylor}(\texttt{cos}(\texttt{x}), \texttt{x} = 0, 4) \;\; = 1 - \frac{1}{2!}x^2 + \mathcal{O}(x^4)$$

which implies that

$$\underbrace{\frac{1}{24}x^4 - \frac{1}{720}x^6 + \cdots}_{=: F(x)} = \mathcal{O}(\mathbf{x^4}). \tag{1.49}$$

Indeed,

$$\frac{|F(x)|}{|x^4|} = \left|\frac{1}{24} - \frac{1}{720}x^2 + \cdots\right| \leq \frac{1}{24}, \quad \text{for sufficiently small } x. \tag{1.50}$$

Thus $F(x) \in \mathcal{O}(x^4)$. □

Example 1.74. Choose the correct assertions (in each, $n \to \infty$)

a. $(n^2+1)/n^3 \in o(1/n)$

b. $(n+1)/\sqrt{n} \in o(1)$

c. $1/\ln n \in \mathcal{O}(1/n)$

d. $1/(n \ln n) \in o(1/n)$

e. $e^n/n^5 \in \mathcal{O}(1/n)$

Example **1.75.** Determine the best integer value of k in the following equation

$$\arctan(x) = x + \mathcal{O}(x^k), \quad \text{as } x \to 0.$$

Solution.

Ans: $k = 3$.

Self-study **1.76.** Let $f(h) = \dfrac{1}{h}(1 + h - e^h)$. What are the limit and the rate of convergence of $f(h)$ as $h \to 0$?

Solution.

Self-study **1.77.** Show that these assertions are not true.

a. $e^x - 1 = \mathcal{O}(x^2)$, as $x \to 0$

b. $x = \mathcal{O}(\tan^{-1} x)$, as $x \to 0$

c. $\sin x \cos x = o(x)$, as $x \to 0$

Solution.

Example **1.78.** Let $\{a_n\} \to 0$ and $\lambda > 1$. Show that

$$\sum_{k=0}^{n} a_k \lambda^k = o(\lambda^n), \quad \text{as } n \to \infty.$$

Hint: $\dfrac{|\sum_{k=0}^{n} a_k \lambda^k|}{|\lambda^n|} = |a_n + a_{n-1}\lambda^{-1} + \cdots + a_0 \lambda^{-n}| =: \varepsilon_n$. Then, we have to show $\varepsilon_n \to 0$ as $n \to \infty$. For this, you can first observe $\varepsilon_{n+1} = a_{n+1} + \dfrac{1}{\lambda}\varepsilon_n$, which implies that ε_n is bounded and converges to ε. Now, can you see $\varepsilon = 0$?

Solution.

1.4. Programming with Matlab/Octave

Note: In computer programming, important things are

- How to deal with **objects** (variables, arrays, functions)
- How to deal with **repetition** effectively
- How to make the program **reusable**

Vectors and matrices

The most basic thing you will need to do is to enter vectors and matrices. You would enter commands to **Matlab** or **Octave** at a prompt that looks like >>.

- Rows are separated by semicolons (;) or Enter .
- Entries in a row are separated by commas (,) or space Space .

For example,

Vectors and Matrices

```
>> u = [1; 2; 3]       % column vector
u =
   1
   2
   3
>> v = [4; 5; 6];
>> u + 2*v
ans =
    9
   12
   15
>> w = [5, 6, 7, 8]    % row vector
w =
   5   6   7   8
>> A = [2 1; 1 2];     % matrix
>> B = [-2, 5
        1, 2]
B =
  -2   5
   1   2
>> C = A*B  % matrix multiplication
C =
   -3   12
    0    9
```

You can save the commands in a file to run and get the same results.

tutorial1_vectors.m

```
u = [1; 2; 3]
v = [4; 5; 6];
u + 2*v
w = [5, 6, 7, 8]
A = [2 1; 1 2];
B = [-2, 5
     1, 2]
C = A*B
```

Solving equations

Let $A = \begin{bmatrix} 1 & -4 & 2 \\ 0 & 3 & 5 \\ 2 & 8 & -4 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 3 \\ -7 \\ -3 \end{bmatrix}$. Then $A\mathbf{x} = \mathbf{b}$ can be *numerically* solved by implementing a code as follows.

tutorial2_solve.m

```
A = [1 -4  2; 0  3  5; 2  8 -4];
b = [3; -7; -3];
x = A\b
```

Result

```
x =
   0.75000
  -0.97115
  -0.81731
```

Graphics with Matlab

In Matlab, the most popular graphic command is `plot`, which creates a 2D line plot of the data in `Y` versus the corresponding values in `X`. A general syntax for the command is

```
plot(X1,Y1,LineSpec1,...,Xn,Yn,LineSpecn)
```

tutorial3_plot.m

```
1  close all
2
3  %% a curve
4  X1 = linspace(0,2*pi,10); % n=10
5  Y1 = cos(X1);
6
7  %% another curve
8  X2=linspace(0,2*pi,20); Y2=sin(X2);
9
10 %% plot together
11 plot(X1,Y1,'-or',X2,Y2,'--b','linewidth',3);
12 legend({'y=cos(x)','y=sin(x)'},'location','best',...
13       'FontSize',16,'textcolor','blue')
14 print -dpng 'fig_cos_sin.png'
```

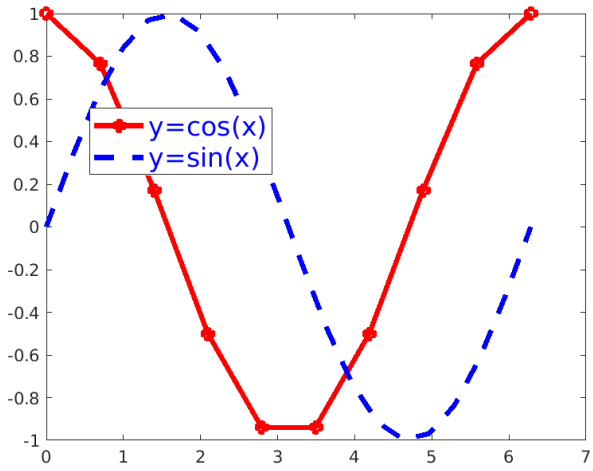


Figure 1.5: fig_cos_sin.png: `plot` of $y = \cos x$ and $y = \sin x$.

Above `tutorial3_plot.m` is a typical M-file for figuring with `plot`.

- Line 1: It closes all figures currently open.
- Lines 3, 4, 7, and 10 (comments): When the percent sign (%) appears, the rest of the line will be ignored by Matlab.
- Lines 4 and 8: The command `linspace(x1,x2,n)` returns a row vector of `n` evenly spaced points between `x1` and `x2`.
- Line 11: Its result is a figure shown in Figure 1.5.
- Line 14: it saves the figure into a png format, named `fig_cos_sin.png`.

Repetition: iteration loops

Note: In scientific computation, one of most frequently occurring events is **repetition**. Each repetition of the process is also called an **iteration**. It is the act of repeating a process, to generate a (possibly unbounded) sequence of outcomes, with the aim of approaching a desired goal, target or result. Thus,

- *iteration must start with an initialization (starting point) and*
- *perform a step-by-step marching in which the results of one iteration are used as the starting point for the next iteration.*

In the context of mathematics or computer science, iteration (along with the related technique of recursion) is a very basic building block in programming. Matlab provides various types of loops: while loops, for loops, and nested loops.

while loop

The syntax of a while loop in Matlab is as follows.

```
while <expression>
    <statements>
end
```

An expression is true when the result is nonempty and contains all nonzero elements, logical or real numeric; otherwise the expression is false. Here is an example for the while loop.

```
n1=11; n2=20;
sum=n1;
while n1<n2
    n1 = n1+1; sum = sum+n1;
end
fprintf('while loop: sum=%d\n',sum);
```

When the code above is executed, the result will be:

```
while loop: sum=155
```

for loop

A **for loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times. The syntax of a for loop in Matlab is as following:

```
for index = values
    <program statements>
end
```

Here is an example for the for loop.

```
n1=11; n2=20;
sum=0;
for i=n1:n2
    sum = sum+i;
end
fprintf('for loop: sum=%d\n',sum);
```

When the code above is executed, the result will be:

```
for loop: sum=155
```

Functions: Enhancing reusability

Program scripts can be saved to **reuse later conveniently**. For example, the script for the summation of integers from `n1` to `n2` can be saved as a form of **function**.

mysum.m

```
1 function s = mysum(n1,n2)
2 % sum of integers from n1 to n2
3
4 s=0;
5 for i=n1:n2
6     s = s+i;
7 end
```

Now, you can call it with e.g. `mysum(11,20)`.
Then the result reads `ans = 155`.

Exercises for Chapter 1

1.1. Prove that the following equations have *at least one* solution in the given intervals.

(a) $x - (\ln x)^3 = 0, \quad [5, 7]$

(b) $(x-1)^2 - 2\sin(\pi x/2) = 0, \quad [0, 2]$

(c) $x - 3 - x^2 e^{-x} = 0, \quad [2, 4]$

(d) $5x\cos(\pi x) - 2x^2 + 3 = 0, \quad [0, 2]$

Ans: (d) $f(0) = 3$, $f(2) = 5$, and $f(1) = -4$.

1.2. C[1] Let $f(x) = 5x\cos(3x) - (x-1)^2$ and $x_0 = 0$.

(a) Find the third Taylor polynomial of f about $x = x_0$, $p_3(x)$, and use it to approximate $f(0.2)$.

(b) Use the Taylor's Theorem (Theorem 1.22) to find an upper bound for the error $|f(x) - p_3(x)|$ at $x = 0.2$. Compare it with the actual error.

(c) Find the fifth Taylor polynomial of f about $x = x_0$, $p_5(x)$, and use it to approximate $f(0.2)$.

(d) Use the Taylor's Theorem to find an upper bound for the error $|f(x) - p_5(x)|$ at $x = 0.2$. Compare it with the actual error.

Ans: (b) $|f(0.2) - p_3(0.2)| = |0.1853356149096783 - 0.18| = 0.005335614909678$. $\mathcal{R}_3(0.2) = \dfrac{f^{(4)}(\xi)}{4!}(0.2)^4 = \dfrac{9}{250}\sin(3\xi) + \dfrac{27}{1000}\xi\cos(3\xi)$. Now, try to estimate the upper bound of $\max|\mathcal{R}_3|$.

1.3. For the fair (x_n, α_n), is it true that $x_n = \mathcal{O}(\alpha_n)$ as $n \to \infty$?

(a) $x_n = 3n^2 - n^4 + 1; \quad \alpha_n = 3n^2$

(b) $x_n = n - \dfrac{1}{\sqrt{n}} + 1; \quad \alpha_n = \sqrt{n}$

(c) $x_n = \sqrt{n - 10}; \quad \alpha_n = 1$

(d) $x_n = -n^2 + 1; \quad \alpha_n = n^3$

Ans: (d)

1.4. Let a sequence x_n be defined recursively by $x_{n+1} = g(x_n)$, where g is continuously differentiable. Suppose that $x_n \to x^*$ as $n \to \infty$ and $g'(x^*) = 0$. Show that

$$x_{n+2} - x_{n+1} = o(x_{n+1} - x_n). \tag{1.51}$$

Hint: Begin with

$$\left|\frac{x_{n+2} - x_{n+1}}{x_{n+1} - x_n}\right| = \left|\frac{g(x_{n+1}) - g(x_n)}{x_{n+1} - x_n}\right|,$$

[1]The mark C indicates that you *should* solve the problem via computer programming. Attach hard copies of your code and results. For other problems, if you like and doable, you may try to solve them with computer programming.

and use the Mean Value Theorem (on page 4) and the fact that is continuously differentiable, to show that the quotient converges to zero as $n \to \infty$.

1.5. A square matrix $A \in \mathbb{R}^{n\times n}$ is said to be **skew-symmetric** if $A^T = -A$. Prove that if A is skew-symmetric, then $\mathbf{x}^T A\mathbf{x} = 0$ for all $\mathbf{x} \in \mathbb{R}^n$.

Hint: The quantity $\mathbf{x}^T A\mathbf{x}$ is scalar so that $(\mathbf{x}^T A\mathbf{x})^T = \mathbf{x}^T A\mathbf{x}$.

1.6. Suppose that A, B, and C are square matrices and that ABC is invertible. Show that each of A, B, and C is invertible.

1.7. Find the determinant and eigenvalues of the following matrices, if it exists. Compare the determinant with the product of eigenvalues, i.e. check if (1.30) is true.

(a) $P = \begin{bmatrix} 2 & 3 \\ 7 & 6 \end{bmatrix}$

(b) $Q = \begin{bmatrix} 1 & 2 \\ -3 & 1 \\ 0 & 1 \end{bmatrix}$

(c) $R = \begin{bmatrix} 6 & 0 & 5 \\ 0 & 4 & 0 \\ 1 & -5 & 2 \end{bmatrix}$

(d) $S = \begin{bmatrix} 1 & 2 & 4 & 8 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 5 & 7 \end{bmatrix}$

1.8. Show that the ℓ^2**-norm** $\|\mathbf{x}\|_2$, defined as

$$\|\mathbf{x}\|_2 = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2},$$

satisfies the three conditions in (1.34), page 23.

Hint: For the last condition, you may begin with

$$\|\mathbf{x} + \mathbf{y}\|_2^2 = (\mathbf{x} + \mathbf{y})\bullet(\mathbf{x} + \mathbf{y}) = \|\mathbf{x}\|_2^2 + 2\,\mathbf{x}\bullet\mathbf{y} + \|\mathbf{y}\|_2^2.$$

Now, compare this with $(\|\mathbf{x}\|_2 + \|\mathbf{y}\|_2)^2$.

1.9. Show that $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1$ for all $\mathbf{x} \in \mathbb{R}^n$.

CHAPTER 2

Solutions of Nonlinear Equations

Through the chapter, the objective is to find solutions for equations of the form

$$f(x) = 0. \tag{2.1}$$

Various numerical methods will be considered for the solutions of (2.1). Although the methods will be derived for a simple form of equations, they will be applicable for various general problems.

Contents of Chapter 2

2.1. The Bisection Method 42
2.2. Fixed-Point Iteration 49
2.3. Newton's Method and Its Variants 56
2.4. Zeros of Polynomials 67
Exercises for Chapter 2 78

2.1. The Bisection Method

It is also called the **binary-search method** or **interval-halving method**.

Note: The objective is to find solutions for

$$f(x) = 0. \tag{2.2}$$

2.1.1. Implementation of the bisection method

Assumption. For the **bisection method**, we assume that

1) f is continuous in $[a, b]$.
2) $f(a) \cdot f(b) < 0$ (so that there must be a solution by the IVT).
3) There is a single solution in $[a, b]$.

Pseudocode **2.1. The Bisection Method**:

- `Given` $[a_1, b_1] = [a, b], \quad p_1 = (a_1 + b_1)/2;$
- `For` $n = 1, 2, \cdots,$ `itmax`

```
if ( f(p_n) = 0 ) then
    stop;
elseif ( f(a_n) · f(p_n) < 0 ) then
    a_{n+1} = a_n;  b_{n+1} = p_n;
else
    a_{n+1} = p_n;  b_{n+1} = b_n;
endif
p_{n+1} = (a_{n+1} + b_{n+1})/2
```

Example 2.2. Find the solution of the equation $x^3 + 4x^2 - 10 = 0$ in $[1, 2]$.

Bisection

```
with(Student[NumericalAnalysis]):
f := x -> x^3 + 4*x^2 - 10:

Bisection(f(x), x = [1,2], tolerance = 0.1,
 stoppingcriterion = absolute, output = sequence);
    [1., 2.],
    [1., 1.500000000],
    [1.250000000, 1.500000000],
    [1.250000000, 1.375000000],
    1.312500000

Bisection(f(x), x = [1, 2], tolerance = 0.1,
 stoppingcriterion = absolute, output = plot)
```

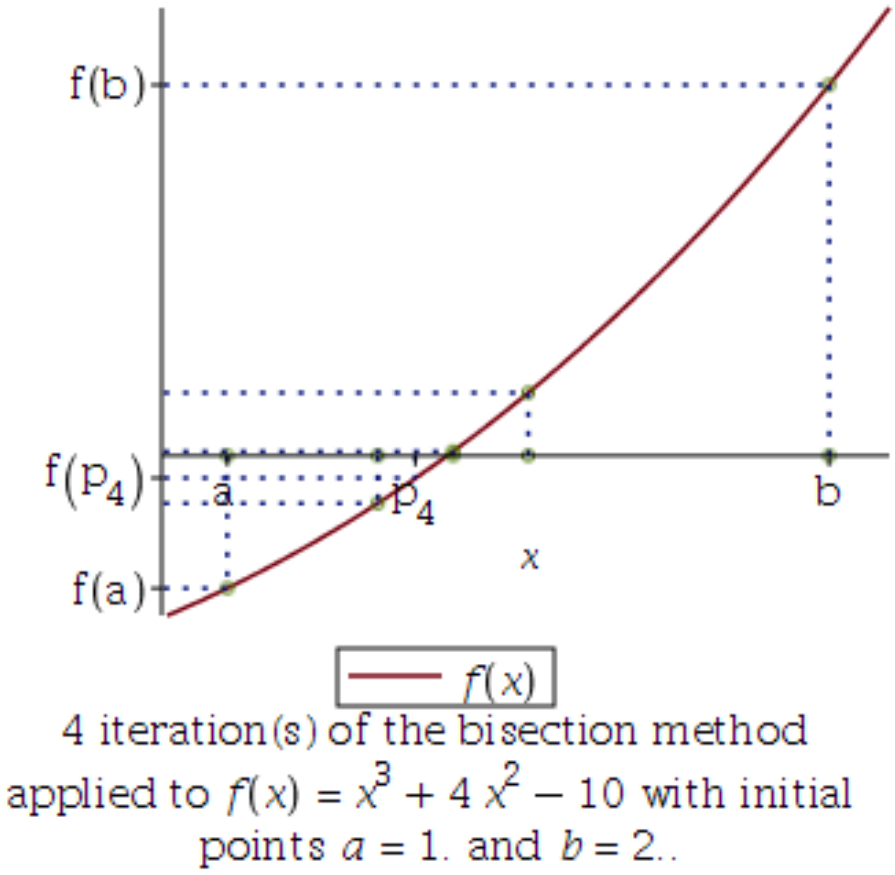


Figure 2.1: The bisection method.

```
> bisection2 := proc(a0, b0, TOL, itmax)
      local k, a, b, p, pp, fp;
      a := a0;
      b := b0;
      p := (a + b) / 2;  pp := p + 10·TOL;
      fp := f(p);
      k := 1;
      printf(" k=%2d: a=%9f  b=%9f  p=%9f  f(p)=%9f\n", k, a, b, p, fp);
      while (k ≤ itmax) do
          if (abs(p − pp) < TOL) then       # stoppingcriterion = absolute
          # if (abs((p − pp) / p)< TOL) then  # stoppingcriterion = relative
          # if (abs(fp) < TOL) then           # stoppingcriterion = function_value
              break;
          end if;
          pp := p;
          k := k + 1;
          if (0 < f(b) * fp) then
              b := p;
          else
              a := p;
          end if;
          p := (a + b) / 2;
          fp := f(p);
          printf(" k=%2d: a=%9f  b=%9f  p=%9f  f(p)=%9f\n", k, a, b, p, fp);
      end do;
      print("f(x) =",f(x));
      printf("  p_%d  = %13.9f \n", k, p);
      printf("  dp   = +-%10f  = (b0-a0)/2^k=%10f\n", 1 / 2 * abs(b − a), (b0 − a0) / 2^k);
      printf("  f(p) = %13.9f \n", fp);
      RETURN(p)
  end proc:
```

Figure 2.2: A Maple code for the bisection method

Result

```
> bisection2(1, 2, 0.01, 20);
 k= 1: a= 1.000000  b= 2.000000  p= 1.500000  f(p)= 2.375000
 k= 2: a= 1.000000  b= 1.500000  p= 1.250000  f(p)=-1.796875
 k= 3: a= 1.250000  b= 1.500000  p= 1.375000  f(p)= 0.162109
 k= 4: a= 1.250000  b= 1.375000  p= 1.312500  f(p)=-0.848389
 k= 5: a= 1.312500  b= 1.375000  p= 1.343750  f(p)=-0.350983
 k= 6: a= 1.343750  b= 1.375000  p= 1.359375  f(p)=-0.096409
 k= 7: a= 1.359375  b= 1.375000  p= 1.367188  f(p)= 0.032356
                                    3      2
                     "f(x)=", x  + 4 x  - 10
  p_7  =   1.367187500
  dp   = +-  0.007812  = (b0-a0)/2^k=  0.007812
  f(p) =   0.032355785
                                  175
                                  ---
                                  128
```

2.1.2. Error analysis for the bisection method

Theorem 2.3. *Suppose that $f \in C[a,b]$ and $f(a) \cdot f(b) < 0$. Then, the Bisection method generates a sequence p_n approximating a zero p of f with*

$$|p - p_n| \leq \frac{b-a}{2^n}, \quad n \geq 1. \tag{2.3}$$

Proof. For $n \geq 1$,

$$b_n - a_n = \frac{1}{2^{n-1}}(b-a) \text{ and } p \in (a_n, b_n). \tag{2.4}$$

It follows from $p_n = (a_n + b_n)/2$ that

$$|p - p_n| \leq \frac{1}{2}(b_n - a_n) = \frac{1}{2^n}(b-a), \tag{2.5}$$

which completes the proof. □

Note: The right-side of (2.3) is the upper bound of the error in the n-th iteration.

Example 2.4. Determine the number of iterations necessary to solve $x^3 + 4x^2 - 10 = 0$ with accuracy 10^{-3} using $[a_1, b_1] = [1, 2]$.

Solution. We have to find the iteration count n such that the error upper-bound is not larger than 10^{-3}. That is, incorporating (2.3),

$$|p - p_n| \leq \frac{b-a}{2^n} \leq 10^{-3}. \tag{2.6}$$

Since $b - a = 1$, it follows from the last inequality that $2^n \geq 10^3$, which implies that

$$n \geq \frac{3\ln(10)}{\ln(2)} \approx 9.965784285.$$

Ans: $n = 10$

Remark 2.5. The zero p is unknown so that the quantity $|p - p_n|$ is a theoretical value; it is not useful in computation.

Note that p_n is the midpoint of $[a_n, b_n]$ and p_{n+1} is the midpoint of either $[a_n, p_n]$ or $[p_n, b_n]$. So,

$$|p_{n+1} - p_n| = \frac{1}{4}(b_n - a_n) = \frac{1}{2^{n+1}}(b - a). \tag{2.7}$$

In other words,

$$|p_n - p_{n-1}| = \frac{1}{2^n}(b - a), \tag{2.8}$$

which implies that

$$|p - p_n| \leq \frac{1}{2^n}(b - a) = |p_n - p_{n-1}|. \tag{2.9}$$

The approximate solution, carried out with the absolute difference $|p_n - p_{n-1}|$ being used for the **stopping criterion**, guarantees the actual error not greater than the given tolerance.

Example 2.6. Suppose that the bisection method begins with the interval $[45, 60]$. How many steps should be taken to compute a root with a relative error not larger than 10^{-8}?

Solution.

Ans: $n \geq \ln(10^8/3)/\ln(2)$. Thus $n = 25$

bisect: a Matlab code

bisect.m

```
function [c,err,fc]=bisect(f,a,b,TOL)
    %Input - f is the function input as a string 'f'
    %       - a and b are the left and right endpoints
    %       - TOL is the tolerance
    %Output - c is the zero
    %         - err is the error estimate for c
    %         - fc= f(c)

fa=feval(f,a);
fb=feval(f,b);
if fa*fb > 0,return,end
max1=1+round((log(b-a)-log(TOL))/log(2));

for k=1:max1
    c=(a+b)/2;
    fc=feval(f,c);
    if fc==0
        a=c; b=c;
    elseif fa*fc<0
        b=c; fb=fc;
    else
        a=c; fa=fc;
    end
    if b-a < TOL, break,end
end

c=(a+b)/2; err=(b-a)/2; fc=feval(f,c);
```

Example 2.7. You can call the above algorithm with varying function, by

```
>> f = @(x) x.^3+4*x.^2-10;
>> [c,err,fc]=bisect(f,1,2,0.005)
c =
    1.3652
err =
    0.0020
fc =
    7.2025e-005
```

Example **2.8.** In the bisection method, does $\lim_{n\to\infty} \dfrac{|p - p_{n+1}|}{|p - p_n|}$ exist?

Solution.

Ans: no

2.2. Fixed-Point Iteration

Definition 2.9. A number p is a **fixed point** for g, if $g(p) = p$.

Note: A point p is a fixed point of g, when the point remains unaltered under the action of g.

Example 2.10. Find fixed points of $g(x) = x^2 - 2$.

Solution.

Maple-code

```
g := x -> x^2 - 2:
g(p) = p;
                              2
                             p  - 2 = p
solve(g(p) = p, p);
                            2, -1
```

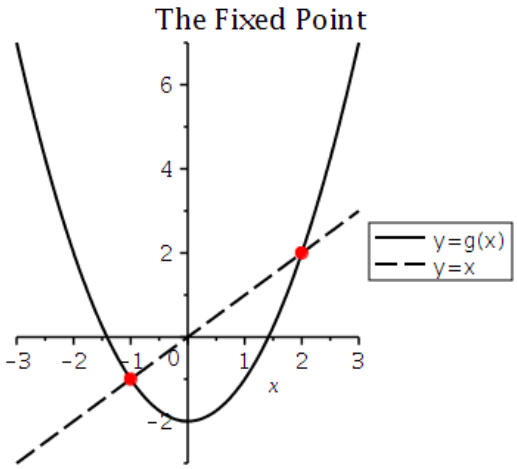


Figure 2.3: Fixed points of $g(x) = x^2 - 2$.

Remark 2.11. Given a **root-finding problem** $f(p) = 0$, let

$$g(x) = x - h(x) \cdot f(x), \tag{2.10}$$

for some $h(x)$. Then, since $g(p) = p - h(p) \cdot f(p) = p - 0 = p$, Equation (2.10) defines a **fixed-point problem**.

2.2.1. Existence and uniqueness of fixed points

Theorem 2.12. (Existence and Uniqueness).

- *If* $g \in C[a,b]$ *and* $g(x) \in [a,b]$ *for all* $x \in [a,b]$, *then* g **has at least one fixed point** *in* $[a,b]$.
- *If, in addition,* g *is differentiable in* (a,b) *and there exists a positive constant* $K < 1$ *such that*

$$|g'(x)| \leq K < 1 \quad \text{for all } x \in (a,b), \tag{2.11}$$

then **there is a unique fixed point** *in* $[a,b]$.

Proof.

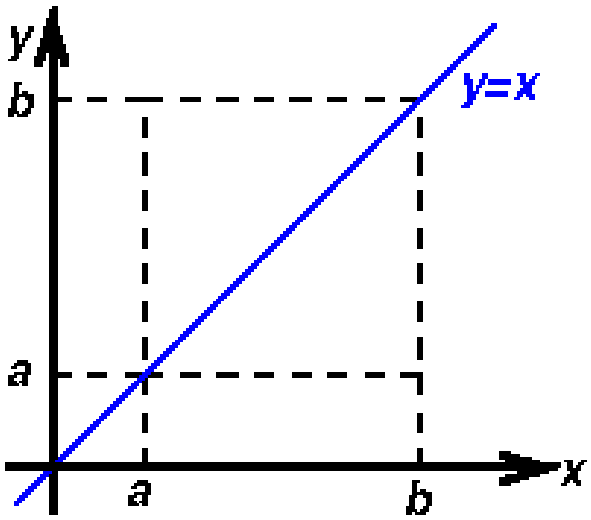


Figure 2.4: Illustration of the existence-and-uniqueness theorem.

Example 2.13. Show that $g(x) = (x^2 - 2)/3$ has a unique fixed point on $[-1,1]$.

Solution.

2.2.2. Fixed-point iteration

Definition 2.14. A **fixed-point iteration** is an iterative procedure of the form: For a given p_0,

$$p_n = g(p_{n-1}) \quad \text{for } n \geq 1. \tag{2.12}$$

If the sequence p_n converges to p, since g is continuous, we have

$$p = \lim_{n\to\infty} p_n = \lim_{n\to\infty} g(p_{n-1}) = g(\lim_{n\to\infty} p_{n-1}) = g(p).$$

This implies that **the limit p is a fixed point of g**, i.e., the iteration converges to a fixed point.

Example 2.15. The equation $x^3 + 4x^2 - 10 = 0$ has a unique root in $[1, 2]$. There are many ways to transform the equation to the fixed-point form $x = g(x)$:

(1) $x = g_1(x) = x - (x^3 + 4x^2 - 10)$

(2) $x = g_2(x) = \frac{1}{4}\left(\frac{10}{x} - x^2\right)$ $\quad \Leftarrow x^2 + 4x - \frac{10}{x} = 0$

(3) $x = g_3(x) = \left(\frac{10}{x} - 4x\right)^{1/2}$

(4) $x = g_4(x) = \frac{1}{2}(-x^3 + 10)^{1/2}$ $\quad \Leftarrow 4x^2 = -x^3 + 10$

(5) $x = g_5(x) = \left(\frac{10}{x+4}\right)^{1/2}$ $\quad \Leftarrow x^2(x+4) - 10 = 0$

(6) $x = g_6(x) = x - \frac{x^3 + 4x^2 - 10}{3x^2 + 8x}$

The associated fixed-point iteration may not converge for some choices of g. Let's check it.

Evaluation of $\max_{x\in[1,2]} |g_k'(x)|$ for the fixed-point iteration

The real root of $x^3 + 4x^2 - 10 = 0$ is $\boldsymbol{p = 1.3652300134142}$.

Maple-code

```
with(Student[NumericalAnalysis]);

g1 := x -> x - x^3 - 4*x^2 + 10:
maximize(abs(diff(g1(x), x)), x = 1..2);
                                27
FixedPointIteration(x-g1(x), x=1.5, tolerance=10^-3, output=sequence);
  1.5, -0.875, 6.732421875, -469.7200120, 1.027545552 10 ,
                   24                72                  216
    -1.084933871 10   , 1.277055593 10   , -2.082712916 10    ,
                  648                1946                5840
    9.034169425 10    , -7.373347340 10    , 4.008612522 10

g2 := x -> 5/2*1/x - 1/4*x^2:
maximize(abs(diff(g2(x), x)), x = 1..2);
                                3
FixedPointIteration(x-g2(x), x=1.5, tolerance=10^-3, output=sequence);
 1.5, 1.104166667, 1.959354936, 0.3161621898, 7.882344262,
                                                          5
   -15.21567323, -58.04348221, -842.3045280, -1.773692325 10 ,
                  9                 19
   -7.864961160 10 , -1.546440351 10

g3 := x -> (10/x - 4*x)^(1/2):
maximize(abs(diff(g3(x), x)), x = 1..2);
                             infinity
FixedPointIteration(x-g3(x), x=1.5, tolerance=10^-3, output=sequence);
                        1.5, 0.8164965811

g4 := x -> 1/2*(10 - x^3)^(1/2):
evalf(maximize(abs(diff(g4(x), x)), x = 1..2));
                           2.121320343
FixedPointIteration(x-g4(x), x=1.5, tolerance=10^-3, output=sequence);
1.5, 1.286953768, 1.402540804, 1.345458374, 1.375170253,
  1.360094192, 1.367846968, 1.363887004, 1.365916734, 1.364878217

g5 := x -> 10^(1/2)*(1/(x + 4))^(1/2):
evalf(maximize(abs(diff(g5(x), x)), x = 1..2));
                           0.1414213562
FixedPointIteration(x-g5(x), x=1.5, tolerance=10^-3, output=sequence);
    1.5, 1.348399725, 1.367376372, 1.364957015, 1.365264748
```

```
41
42  g6 := x -> x - (x^3 + 4*x^2 - 10)/(3*x^2 + 8*x):
43  maximize(diff(g6(x), x), x = 1..2);
44                                  5
45                                  --
46                                  14
47  maximize(-diff(g6(x), x), x = 1..2);
48                                 70
49                                 ---
50                                 121
51  FixedPointIteration(x-g6(x), x=1.5, tolerance=10^-3, output=sequence);
52            1.5, 1.373333333, 1.365262015, 1.365230014
```

Theorem 2.16. ***(Fixed-Point Theorem)**:*
Let $g \in C[a,b]$ *and* $g(x) \in [a,b]$ *for all* $x \in [a,b]$. *Suppose that* g *is differentiable in* (a,b) *and there is a positive constant* $K < 1$ *such that*

$$|g'(x)| \leq K < 1 \text{ for all } x \in (a,b). \tag{2.13}$$

Then, for any number $p_0 \in [a,b]$, *the sequence defined by*

$$p_n = g(p_{n-1}), \quad n \geq 1, \tag{2.14}$$

converges to the unique fixed point $p \in [a,b]$.

Proof.

- It follows from Theorem 2.12 that there exists a unique fixed point $p \in [a,b]$, i.e., $p = g(p) \in [a,b]$.
- Since $g(x) \in [a,b]$ for all $x \in [a,b]$, we have $p_n \in [a,b]$ for all $n \geq 1$. It follows from the *MVT* that

  $$|p - p_n| = |g(p) - g(p_{n-1})| = |g'(\xi_n)(p - p_{n-1})| \leq K|p - p_{n-1}|,$$

  for some $\xi_n \in (a,b)$. Therefore

  $$|p - p_n| \leq K|p - p_{n-1}| \leq K^2|p - p_{n-2}| \leq \cdots \leq K^n|p - p_0|, \tag{2.15}$$

  which converges to 0 as $n \to \infty$.

□

Remark 2.17. The Fixed-Point Theorem deserves some remarks.

- From (2.15), we can see

$$|p - p_n| \le K^n \max\{p_0 - a,\ b - p_0\}. \tag{2.16}$$

- For $m > n \ge 1$,

$$\begin{aligned} |p_m - p_n| &= |p_m - p_{m-1} + p_{m-1} - \cdots - p_{n+1} + p_{n+1} - p_n| \\ &\le |p_m - p_{m-1}| + |p_{m-1} - p_{m-2}| + \cdots + |p_{n+1} - p_n| \\ &\le K^{m-1}|p_1 - p_0| + K^{m-2}|p_1 - p_0| + \cdots + K^n|p_1 - p_0| \\ &= K^n|p_1 - p_0|(1 + K + K^2 + \cdots + K^{m-1-n}). \end{aligned}$$

(Here we have used the MVT, for the last inequality.) Thus,

$$|p - p_n| = \lim_{m\to\infty} |p_m - p_n| \le K^n|p_1 - p_0| \sum_{i=0}^{\infty} K^i = \frac{K^n}{1-K}|p_1 - p_0|.$$

That is,

$$|p - p_n| \le \frac{K^n}{1-K}|p_1 - p_0|. \tag{2.17}$$

- The Fixed-Point Theorem holds for any contractive mapping g defined on any closed subset $C \subset \mathbb{R}$. By a **contractive mapping**, we mean a function g that satisfies for some $0 < K < 1$,

$$|g(x) - g(y)| \le K|x - y| \text{ for all } x, y \in C. \tag{2.18}$$

Note: If a contractive mapping g is differentiable, then (2.18) implies that

$$|g'(x)| \le K \text{ for all } x \in C.$$

Practice 2.18. In practice, p is unknown. Consider the following:

$$\begin{aligned} |p_{n+1} - p_n| &\ge |p_n - p| - |p_{n+1} - p| \\ &\ge |p_n - p| - K|p_n - p| = (1 - K)|p_n - p| \end{aligned}$$

and therefore

$$|p - p_n| \le \frac{1}{1-K}|p_{n+1} - p_n| \le \frac{K}{1-K}|p_n - p_{n-1}|, \tag{2.19}$$

which is useful for stopping of the iteration.

Example 2.19. For each of the following equations, (1) determine an interval $[a, b]$ on which the fixed-point iteration will converge. (2) Estimate the number of iterations necessary to obtain approximations accurate to within 10^{-5}.

(a) $x = \dfrac{2 - e^x + x^2}{3}$ (b) $x = \dfrac{5}{x^2} + 2$

Solution. You may first try to visualize the functions.

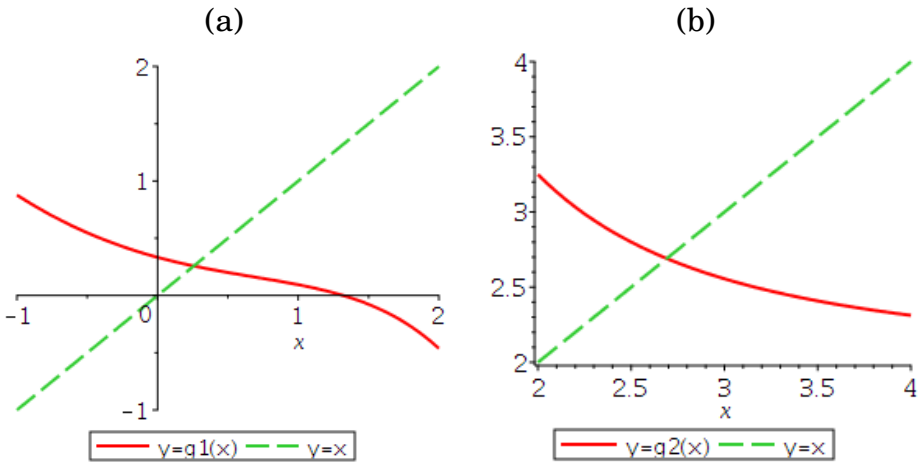


Figure 2.5: Visualization of the functions.

Ans: (a): (1) $[0, 1]$, (2) $K = 1/3 \Rightarrow n \geq 5\ln(10)/\ln(3) \approx 10.48$.

Example 2.20. Prove that the sequence x_n defined recursively as follows is convergent.

$$\begin{cases} x_0 &= -15 \\ x_{n+1} &= 3 - \dfrac{1}{2}|x_n| \quad (n \geq 0) \end{cases}$$

Solution. Begin with setting $g(x) = 3 - \frac{1}{2}|x|$; then show g is a contractive mapping on $C = \mathbb{R}$.

2.3. Newton's Method and Its Variants

2.3.1. The Newton's method

The **Newton's method** is also called the **Newton-Raphson method**.

Recall: The objective is to find a zero p of f:

$$f(p) = 0. \tag{2.20}$$

Strategy 2.21. Let p_0 be an approximation of p. We will try to find a **correction term** h such that $(p_0 + h)$ is a better approximation of p than p_0; ideally $(p_0 + h) = p$.

- If f'' exists and is continues, then by Taylor's Theorem

$$0 = f(p) = f(p_0 + h) = f(p_0) + (p - p_0)f'(p_0) + \frac{(p - p_0)^2}{2} f''(\xi), \tag{2.21}$$

  where ξ lies between p and p_0.
- If $|p - p_0|$ is small, it is reasonable to ignore the last term of (2.21) and solve for $h = p - p_0$:

$$h = p - p_0 \approx -\frac{f(p_0)}{f'(p_0)}. \tag{2.22}$$

- Define

$$p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}; \tag{2.23}$$

  then p_1 may be a better approximation of p than p_0.
- The above can be repeated.

Algorithm 2.22. ***(Newton's method for solving*** $\mathrm{f(x)} = 0$***).*** *For p_0 chosen close to a root p, compute $\{p_n\}$ repeatedly satisfying*

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad n \geq 1. \tag{2.24}$$

Graphical interpretation

- Let p_0 be the initial approximation close to p. Then, the **tangent line** at $(p_0, f(p_0))$ reads

$$L(x) = f'(p_0)(x - p_0) + f(p_0). \tag{2.25}$$

- To find the **x-intercept** of $y = L(x)$, let

$$0 = f'(p_0)(x - p_0) + f(p_0).$$

Solving the above equation for x becomes

$$x = p_0 - \frac{f(p_0)}{f'(p_0)}, \tag{2.26}$$

of which the right-side is the same as in (2.23).

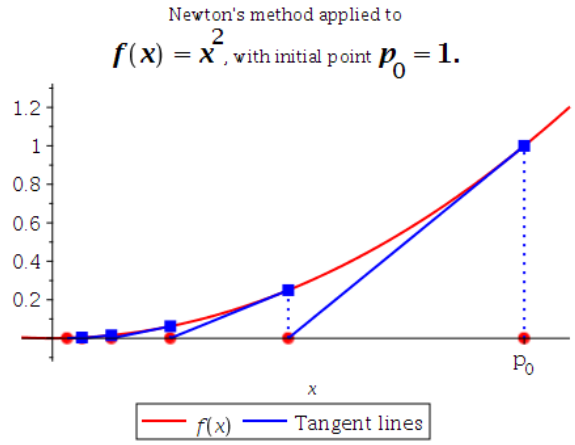
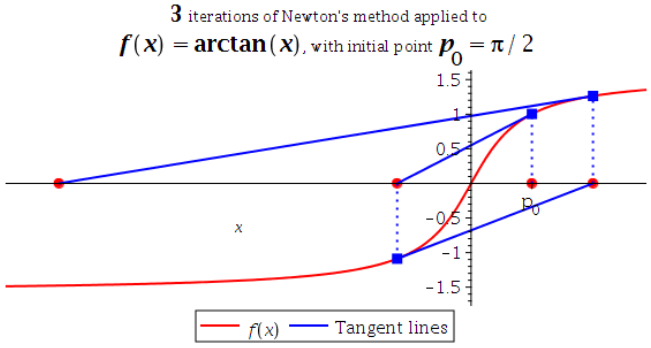


Figure 2.6: Graphical interpretation of the Newton's method.

An Example of Divergence

```
1 f := arctan(x);
2 Newton(f, x = Pi/2, output = plot, maxiterations = 3);
```



Remark 2.23.

- The Newton's method may diverge, unless the initialization is accurate.
- The Newton's method can be interpreted as a **fixed-point iteration**:

$$p_n = g(p_{n-1}) := p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}. \tag{2.27}$$

- It cannot be continued if $f'(p_{n-1}) = 0$ for some n. As a matter of fact, the Newton's method is most effective when $f'(x)$ is bounded away from zero near p.

Convergence analysis for the Newton's method: Define the error in the n-th iteration: $e_n = p_n - p$. Then

$$e_n = p_n - p = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} - p = \frac{e_{n-1}f'(p_{n-1}) - f(p_{n-1})}{f'(p_{n-1})}. \tag{2.28}$$

On the other hand, it follows from the Taylor's Theorem that

$$0 = f(p) = f(p_{n-1} - e_{n-1}) = f(p_{n-1}) - e_{n-1}f'(p_{n-1}) + \frac{1}{2}e_{n-1}^2 f''(\xi_{n-1}), \tag{2.29}$$

for some ξ_{n-1}. Thus, from (2.28) and (2.29), we have

$$e_n = \frac{1}{2}\frac{f''(\xi_{n-1})}{f'(p_{n-1})}e_{n-1}^2. \tag{2.30}$$

Theorem 2.24. (Convergence of Newton's method): *Let $f \in C^2[a,b]$ and $p \in (a,b)$ is such that $f(p) = 0$ and $f'(p) \neq 0$. Then, there is a neighborhood of p such that if the Newton's method is started p_0 in that neighborhood, it generates a convergent sequence p_n satisfying*

$$|p_n - p| \leq C|p_{n-1} - p|^2, \tag{2.31}$$

for a positive constant C.

Example 2.25. Apply the Newton's method to solve $f(x) = \arctan(x) = 0$, with $p_0 = \pi/5$.

```
Newton(arctan(x), x = Pi/5, output = sequence, maxiterations = 5)
    0.6283185308, -0.1541304479, 0.0024295539, -9.562*10^(-9), 0., 0.
```

Since $p = 0$, $e_n = p_n$ and

$$|e_n| \leq 0.67|e_{n-1}|^3, \tag{2.32}$$

which is an occasional **super-convergence**. □

Theorem 2.26. (Newton's Method for a Convex Function): *Let $f \in C^2(\mathbb{R})$ be increasing, convex, and of a zero. Then, the zero is unique and the Newton iteration will converge to it from any starting point.*

Example 2.27. Use the Newton's method to find the **square root** of a positive number Q.

Solution. Let $x = \sqrt{Q}$. Then x is a root of $x^2 - Q = 0$. Define $f(x) = x^2 - Q$; set $f'(x) = 2x$. The Newton's method reads

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})} = p_{n-1} - \frac{p_{n-1}^2 - Q}{2p_{n-1}} = \frac{1}{2}\left(p_{n-1} + \frac{Q}{p_{n-1}}\right). \tag{2.33}$$

(Compare the above with (1.45), p. 28.)

NR.mw

```
NR := proc(Q, p0, itmax)
    local p, n;
    p := p0;
    for n to itmax do
        p := (p+Q/p)/2;
        print(n, evalf[14](p));
    end do;
end proc:
```

```
Q := 16: p0 := 1: itmax := 8:
NR(Q,p0,itmax);
        1, 8.5000000000000
        2, 5.1911764705882
        3, 4.1366647225462
        4, 4.0022575247985
        5, 4.0000006366929
        6, 4.0000000000000
        7, 4.0000000000000
        8, 4.0000000000000
```

```
Q := 16: p0 := -1: itmax := 8:
NR(Q,p0,itmax);
        1, -8.5000000000000
        2, -5.1911764705882
        3, -4.1366647225462
        4, -4.0022575247985
        5, -4.0000006366929
        6, -4.0000000000000
        7, -4.0000000000000
        8, -4.0000000000000
```

2.3.2. Systems of nonlinear equations

The Newton's method for **systems of nonlinear equations** follows the same strategy that was used for single equation.That is,

(a) we first linearize,

(b) solve for the correction vector, and

(c) update the solution,

repeating the steps as often as necessary.

An illustration:

- We begin with a pair of equations involving two variables:

$$\begin{cases} f_1(x_1, x_2) &= 0 \\ f_2(x_1, x_2) &= 0 \end{cases} \tag{2.34}$$

- Suppose that (x_1, x_2) is an approximate solution of the system. Let us compute the correction vector (h_1, h_2) so that $(x_1 + h_1, x_2 + h_2)$ is a better approximate solution.

$$\begin{cases} 0 &= f_1(x_1 + h_1, x_2 + h_2) &\approx f_1(x_1, x_2) + h_1 \dfrac{\partial f_1}{\partial x_1} + h_2 \dfrac{\partial f_1}{\partial x_2}, \\ 0 &= f_2(x_1 + h_1, x_2 + h_2) &\approx f_2(x_1, x_2) + h_1 \dfrac{\partial f_2}{\partial x_1} + h_2 \dfrac{\partial f_2}{\partial x_2}. \end{cases} \tag{2.35}$$

- Define the **Jacobian** of (f_1, f_2) at (x_1, x_2):

$$J(x_1, x_2) := \begin{bmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 \end{bmatrix} (x_1, x_2). \tag{2.36}$$

Then, the Newton's method for two nonlinear equations in two variables reads

$$\begin{bmatrix} x_1^n \\ x_2^n \end{bmatrix} = \begin{bmatrix} x_1^{n-1} \\ x_2^{n-1} \end{bmatrix} + \begin{bmatrix} h_1^{n-1} \\ h_2^{n-1} \end{bmatrix}, \tag{2.37}$$

where the correction vector satisfies

$$J(x_1^{n-1}, x_2^{n-1}) \begin{bmatrix} h_1^{n-1} \\ h_2^{n-1} \end{bmatrix} = - \begin{bmatrix} f_1(x_1^{n-1}, x_2^{n-1}) \\ f_2(x_1^{n-1}, x_2^{n-1}) \end{bmatrix}. \tag{2.38}$$

Summary **2.28.** *In general, the system of m nonlinear equations,*

$$f_i(x_1, x_2, \cdots, x_m) = 0, \;\; 1 \le i \le m,$$

can be expressed as

$$F(X) = 0, \tag{2.39}$$

where $X = (x_1, x_2, \cdots, x_m)^T$ and $F = (f_1, f_2, \cdots, f_m)^T$. Then

$$0 = F(X + H) \approx F(X) + J(X)H, \tag{2.40}$$

where $H = (h_1, h_2, \cdots, h_m)^T$, the correction vector, and $J(X) = \left[\dfrac{\partial f_i}{\partial x_j}\right](X)$, the Jacobian of F at X. Hence, Newton's method for m nonlinear equations in m variables is given by

$$X^n = X^{n-1} + H^{n-1}, \tag{2.41}$$

where H^{n-1} is the solution of the linear system:

$$J(X^{n-1})H^{n-1} = -F(X^{n-1}). \tag{2.42}$$

Example 2.29. Starting with $(1, 1, 1)^T$, carry out 6 iterations of the Newton's method to find a root of the nonlinear system

$$\begin{aligned} xy &= z^2 + 1 \\ xyz + y^2 &= x^2 + 2 \\ e^x + z &= e^y + 3 \end{aligned}$$

Solution.

Procedure NewtonRaphsonSYS.mw

```
NewtonRaphsonSYS := proc(X, F, X0, TOL, itmax)
   local Xn, H, FX, J, i, m, n, Err;
   m := LinearAlgebra[Dimension](Vector(X));
   Xn := Vector(m);
   H := Vector(m);
   FX := Vector(m);
   J := Matrix(m, m);
   Xn := X0;
   for n to itmax do
      FX := eval(F, [seq(X[i] = Xn[i], i = 1..m)]);
      J := evalf[15](VectorCalculus[Jacobian](F, X=convert(Xn,list)));
      H := -MatrixInverse(J).Vector(FX);
      Xn := Xn + H;
      printf(" %3d    %.8f ", n, Xn[1]);
      for i from 2 to m do; printf("  %.8f ", Xn[i]); end do;
      for i to m do; printf("  %.3g ", H[i]); end do;
      printf("\n");
      if (LinearAlgebra[VectorNorm](H, 2) < TOL) then break endif:
   end do;
end proc:
```

Result

```
F := [x*y-z^2-1, x*y*z-x^2+y^2-2, exp(x)+z-exp(y)-3]:
X := [x, y, z]:
X0 := <1, 1, 1>:
TOL := 10^-8: itmax := 10:
NewtonRaphsonSYS(X, F, X0, TOL, itmax):
  1  2.18932610  1.59847516  1.39390063  1.19  0.598  0.394
  2  1.85058965  1.44425142  1.27822400  -0.339  -0.154  -0.116
  3  1.78016120  1.42443598  1.23929244  -0.0704  -0.0198  -0.0389
  4  1.77767471  1.42396093  1.23747382  -0.00249  -0.000475  -0.00182
  5  1.77767192  1.42396060  1.23747112  -2.79e-006  -3.28e-007  -2.7e-006
  6  1.77767192  1.42396060  1.23747112  -3.14e-012  -4.22e-014  -4.41e-012
```

2.3.3. The secant method

Recall: The Newton's method, defined as

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}, \quad n \geq 1, \tag{2.43}$$

is a powerful technique. However it has a major drawback: *the need to know the value of derivative of f at each iteration*. Frequently, $f'(x)$ is far more difficult to calculate than $f(x)$.

Algorithm 2.30. (The Secant method). To overcome the disadvantage of the Newton's method, a number of methods have been proposed. One of most popular variants is the **secant method**, which replaces $f'(p_{n-1})$ by a difference quotient:

$$f'(p_{n-1}) \approx \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}. \tag{2.44}$$

Thus, the resulting algorithm reads

$$p_n = p_{n-1} - f(p_{n-1})\left[\frac{p_{n-1} - p_{n-2}}{f(p_{n-1}) - f(p_{n-2})}\right], \quad n \geq 2. \tag{2.45}$$

Note:

- Two initial values (p_0, p_1) must be given, which however is not a drawback.
- In each iteration, it requires only one new evaluation of f.
- Convergence:

$$|e_n| \approx \left|\frac{f''(p)}{2f'(p)} e_{n-1}e_{n-2}\right| \approx \left|\frac{f''(p)}{2f'(p)}\right|^{0.62} |e_{n-1}|^{(1+\sqrt{5})/2}. \tag{2.46}$$

Here `evalf((1+sqrt(5))/2) = 1.618033988`.

The graphical interpretation of the secant method is similar to that of Newton's method. ***The secant method utilizes the secant line, while Newton's method updates the iterate through the tangent line.***

Graphical interpretation

```
with(Student[NumericalAnalysis]):
f := x^3 - 1:
Secant(f, x = [1.5, 0.5], maxiterations = 3, output = sequence);
    1.5, 0.5, 0.7692307692, 1.213510253, 0.9509757891
Secant(f, x = [1.5, 0.5], maxiterations = 3, output = plot);
```

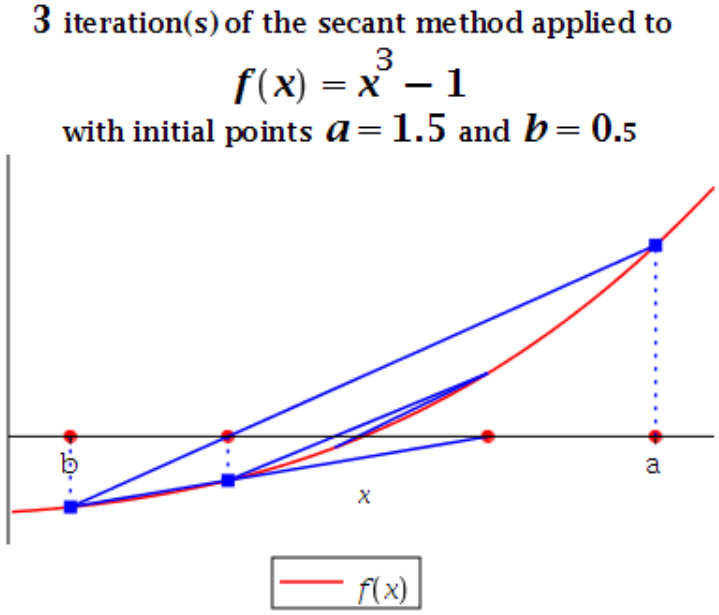


Figure 2.7: Graphical interpretation of the secant method.

Example 2.31. Apply one iteration of the secant method to find p_2 if

$$p_0 = 1, \;\; p_1 = 2, \;\; f(p_0) = 2, \;\; f(p_1) = 1.5.$$

Solution.

Ans: $p_2 = 5.0$

2.3.4. The method of false position

It generates approximations in a similar manner as the secant method; however, it includes a test to ensure that the root is always bracketed between successive iterations.

Algorithm 2.32. (Method of false position).

- Select p_0 and p_1 such that $f(p_0) \cdot f(p_1) < 0$.
- Compute

  $p_2 =$ the x-intercept of the line joining $(p_0, f(p_0))$ and $(p_1, f(p_1))$.

- If $(f(p_1) \cdot f(p_2) < 0)$, set ($p_1$ and p_2 bracket the root)

  $p_3 =$ the x-intercept of the line joining $(p_2, f(p_2))$ and $(p_1, f(p_1))$.

  else, set

  $p_3 =$ the x-intercept of the line joining $(p_2, f(p_2))$ and $(p_0, f(p_0))$.

  endif

Graphical interpretation

```
with(Student[NumericalAnalysis]):
f := x^3 - 1:
FalsePosition(f,x=[1.5,0.5], maxiterations=3, output=plot);
```

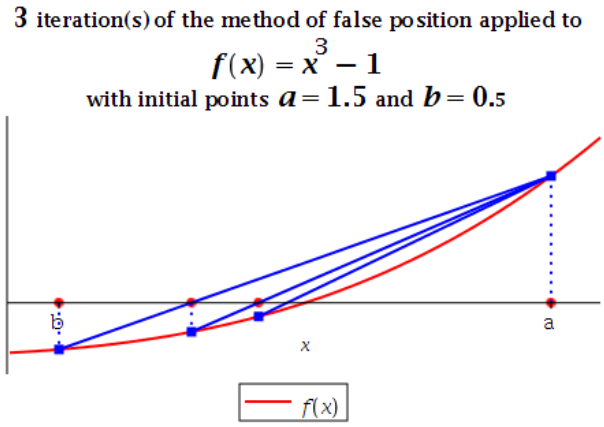


Figure 2.8: Graphical interpretation of the false position method.

Convergence Speed:

Find a root for $x = \cos x$, starting with $\pi/4$ or $[0.5, \pi/4]$.

Comparison

```
with(Student[NumericalAnalysis]):
f := cos(x) - x:

N := Newton(f, x=Pi/4, tolerance=10^-8, maxiterations=10,
      output=sequence);
   0.7853981635, 0.7395361335, 0.7390851781, 0.7390851332, 0.7390851332

S := Secant(f,x=[0.5,Pi/4],tolerance=10^-8,maxiterations=10,
      output=sequence);
   0.5, 0.7853981635, 0.7363841388, 0.7390581392, 0.7390851493,
   0.7390851332, 0.7390851332

F := FalsePosition(f,x=[0.5,Pi/4],tolerance=10^-8,maxiterations=10,
     output=sequence);
   [0.5, 0.7853981635], [0.7363841388, 0.7853981635],
   [0.7390581392, 0.7853981635], [0.7390848638, 0.7853981635],
   [0.7390851305, 0.7853981635], [0.7390851332, 0.7853981635],
   [0.7390851332, 0.7853981635], [0.7390851332, 0.7853981635],
   [0.7390851332, 0.7853981635], [0.7390851332, 0.7853981635],
   [0.7390851332, 0.7853981635]
```

```
# print out
  n      Newton         Secant     False Position
  0   0.7853981635  0.5000000000  0.5000000000
  1   0.7395361335  0.7853981635  0.7363841388
  2   0.7390851781  0.7363841388  0.7390581392
  3   0.7390851332  0.7390581392  0.7390848638
  4   0.7390851332  0.7390851493  0.7390851305
  5   0.7390851332  0.7390851332  0.7390851332
  6   0.7390851332  0.7390851332  0.7390851332
  7   0.7390851332  0.7390851332  0.7390851332
  8   0.7390851332  0.7390851332  0.7390851332
```

2.4. Zeros of Polynomials

Definition 2.33. A **polynomial of degree** n has a form

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0, \tag{2.47}$$

where $a_n \neq 0$ and a_i's are called the **coefficients** of P.

Theorem 2.34. (Theorem on Polynomials).

- ***Fundamental Theorem of Algebra****: Every nonconstant polynomial has at least one root (possibly, in the complex field).*
- **Complex Roots of Polynomials***: A polynomial of degree n has exactly n roots in the complex plane, being agreed that each root shall be counted a number of times equal to its multiplicity. That is, there are unique (complex) constants $x_1, x_2, \cdots, x_k$ and unique integers $m_1, m_2, \cdots, m_k$ such that*

$$P(x) = a_n(x - x_1)^{m_1}(x - x_2)^{m_2}\cdots(x - x_k)^{m_k}, \quad \sum_{i=1}^{k} m_i = n. \tag{2.48}$$

- **Localization of Roots***: All roots of the polynomial P lie in the open disk centered at the origin and of radius of*

$$\rho = 1 + \frac{1}{|a_n|}\max_{0\leq i<n}|a_i|. \tag{2.49}$$

- **Uniqueness of Polynomials***: Let $P(x)$ and $Q(x)$ be polynomials of degree n. If $x_1, x_2, \cdots, x_r$, with $r > n$, are distinct numbers with $P(x_i) = Q(x_i)$, for $i = 1, 2, \cdots, r$, then $P(x) = Q(x)$ for all x. For example, two polynomials of degree n are the same if they agree at $(n + 1)$ points.*

2.4.1. Horner's method

Note: Known as **nested multiplication** and also as **synthetic division**, **Horner's method** can evaluate polynomials very efficiently. It requires n multiplications and n additions to evaluate an arbitrary n-th degree polynomial.

Algorithm 2.35. Let us try to evaluate $P(x)$ at $x = x_0$.

- Utilizing the **Remainder Theorem**, we can rewrite the polynomial as

$$P(x) = (x - x_0)Q(x) + r = (x - x_0)Q(x) + P(x_0), \tag{2.50}$$

where $Q(x)$ is a polynomial of degree $n - 1$, say

$$Q(x) = b_n x^{n-1} + \cdots + b_2 x + b_1. \tag{2.51}$$

- Substituting the above into (2.50), utilizing (2.47), and setting equal the coefficients of like powers of x on the two sides of the resulting equation, we have

$$\begin{array}{rcl} b_n & = & a_n \\ b_{n-1} & = & a_{n-1} + x_0 b_n \\ & \vdots & \\ b_1 & = & a_1 + x_0 b_2 \\ P(x_0) & = & a_0 + x_0 b_1 \end{array} \tag{2.52}$$

- Introducing $b_0 = P(x_0)$, the above can be rewritten as

$$b_{n+1} = 0; \quad b_k = a_k + x_0 b_{k+1}, \ \ n \geq k \geq 0. \tag{2.53}$$

- If the calculation of Horner's algorithm is to be carried out with pencil and paper, the following arrangement is often used (known as **synthetic division**):

	a_n	a_{n-1}	a_{n-2}	$\cdots$	a_0
x_0		$x_0 b_n$	$x_0 b_{n-1}$	$\cdots$	$x_0 b_1$
	b_n	b_{n-1}	b_{n-2}	$\cdots$	$P(x_0) = b_0$

Example 2.36. Use Horner's algorithm to evaluate $P(3)$, where

$$P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2. \tag{2.54}$$

Solution. For $x_0 = 3$, we arrange the calculation as mentioned above:

	1	−4	7	−5	−2
3		3	−3	12	21
	1	−1	4	7	19 = P(3)

Note that the 4-th degree polynomial in (2.54) is written as

$$P(x) = (x-3)(x^3 - x^2 + 4x + 7) + 19. \quad \square$$

Note: When the Newton's method is applied for finding an approximate zero of $P(x)$, the iteration reads

$$x_n = x_{n-1} - \frac{P(x_{n-1})}{P'(x_{n-1})}. \tag{2.55}$$

Thus both $P(x)$ and $P'(x)$ must be evaluated in each iteration.

How to evaluate $P'(x)$

The derivative $P'(x)$ can be evaluated by using the Horner's method with the same efficiency.

- Recall the Remainder Theorem (2.50):

$$P(x) = (x - x_0)Q(x) + r = (x - x_0)Q(x) + P(x_0).$$

- Differentiating (2.50) reads

$$P'(x) = Q(x) + (x - x_0)Q'(x). \tag{2.56}$$

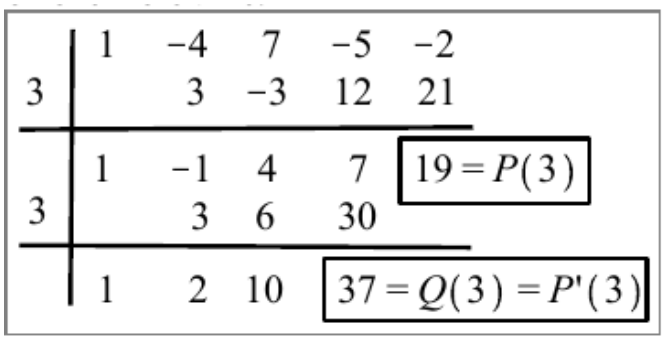
- Thus

$$P'(x_0) = Q(x_0); \tag{2.57}$$

i.e., the evaluation of Q at x_0 becomes the desired quantity $P'(x_0)$. $\square$

Example 2.37. Evaluate $P'(3)$ for $P(x)$ considered in Example 2.36, the previous example.

Solution. As in the previous example, we arrange the calculation and carry out the synthetic division one more time:



Example 2.38. Implement the Horner's algorithm to evaluate $P(3)$ and $P'(3)$, for the polynomial in (2.54): $P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2$.

Solution.

horner.m

```
function [p,d] = horner(A,x0)
% function [px0,dpx0] = horner(A,x0)
%   input:  A = [a_0,a_1,...,a_n]
%   output: p=P(x0), d=P'(x0)

n = size(A(:),1);
p = A(n); d=0;

for i = n-1:-1:1
   d = p + x0*d;
   p = A(i) +x0*p;
end
```

Call_horner.m

```
a = [-2 -5 7 -4 1];
x0=3;
[p,d] = horner(a,x0);
fprintf("  P(%g)=%g; P'(%g)=%g\n",x0,p,x0,d)
% Output:  P(3)=19; P'(3)=37
```

Example 2.39. Let $P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2$, as in (2.54). Use the Newton's method and the Horner's method to implement a code and find an approximate zero of P near 3.

Solution.

newton_horner.m

```
function [x,it] = newton_horner(A,x0,tol,itmax)
% function x = newton_horner(A,x0)
%   input:  A = [a_0,a_1,...,a_n]; x0: initial for P(x)=0
%   outpue: x: P(x)=0

x = x0;
for it=1:itmax
   [p,d] = horner(A,x);
   h = -p/d;
   x = x + h;
   if(abs(h)<tol), break; end
end
```

Call_newton_horner.m

```
a = [-2 -5 7 -4 1];
x0=3;
tol = 10^-12; itmax=1000;
[x,it] = newton_horner(a,x0,tol,itmax);
fprintf("  newton_horner: x0=%g; x=%g, in %d iterations\n",x0,x,it)
% Output:  newton_horner: x0=3; x=2, in 7 iterations
```

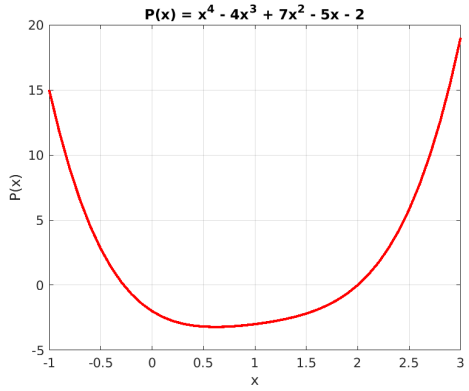


Figure 2.9: Polynomial $P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2$. Its two zeros are -0.275682 and 2.

2.4.2. Complex zeros: Finding quadratic factors

Note: **(Quadratic Factors of Real-coefficient Polynomials)**.
As mentioned in (2.47), a **polynomial of degree** n has a form

$$P(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0. \tag{2.58}$$

- **Theorem on Real Quadratic Factor**: If P is a polynomial whose coefficients are all real, and if z is a nonreal root of P, then $\overline{z}$ is also a root and
$$(x - z)(x - \overline{z})$$
is a real quadratic factor of P.
- **Polynomial Factorization**: If P is a nonconstant polynomial of real coefficients, then it can be factorized as a multiple of linear and quadratic polynomials of which coefficients are all real.
- **Theorem on Quotient and Remainder**: If the polynomial is divided by the quadratic polynomial $(x^2 - ux - v)$, then we can formally write the **quotient** and **remainder** as

$$\begin{aligned} Q(x) &= b_n x^{n-2} + b_{n-1}x^{n-3} + \cdots + b_3 x + b_2 \\ r(x) &= b_1(x - u) + b_0, \end{aligned} \tag{2.59}$$

with which $P(x) = (x^2 - ux - v)Q(x) + r(x)$. As in Algorithm 2.35, the coefficients b_k can be computed recursively as follows.

$$\begin{aligned} b_{n+1} &= b_{n+2} = 0 \\ b_k &= a_k + ub_{k+1} + vb_{k+2}, \quad n \geq k \geq 0. \end{aligned} \tag{2.60}$$

2.4.3. Bairstow's method

Bairstow's method seeks a real quadratic factor of P of the form $(x^2 - ux - v)$. For simplicity, all the coefficients a_i's are real so that both u and v will be real.

Observation 2.40. In order for the quadratic polynomial to be a factor of P, the remainder $r(x)$ must be zero. That is, the process seeks a quadratic factor $(x^2 - ux - v)$ of P such that

$$b_0(u, v) = 0, \quad b_1(u, v) = 0. \tag{2.61}$$

The quantities b_0 and b_1 must be functions of (u, v), which is clear from (2.59) and (2.60).

Key Idea 2.41. An outline of the process is as follows:

- Starting values are assigned to (u, v). We seek corrections $(\delta u, \delta v)$ so that
$$b_0(u + \delta u, v + \delta v) = b_1(u + \delta u, v + \delta v) = 0 \tag{2.62}$$
- Linearization of these equations reads
$$\begin{aligned} 0 &\approx b_0(u, v) + \frac{\partial b_0}{\partial u}\delta u + \frac{\partial b_0}{\partial v}\delta v \\ 0 &\approx b_1(u, v) + \frac{\partial b_1}{\partial u}\delta u + \frac{\partial b_1}{\partial v}\delta v \end{aligned} \tag{2.63}$$
- Thus, the corrections can be found by solving the linear system
$$J \begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = - \begin{bmatrix} b_0(u, v) \\ b_1(u, v) \end{bmatrix}, \quad \text{where } J = \frac{\partial(b_0, b_1)}{\partial(u, v)}. \tag{2.64}$$
Here J is the **Jacobian matrix**.

Question: How to compute the Jacobian matrix

Bairstows method

Algorithm 2.42.

- As first appeared in the appendix of the 1920 book “Applied Aerodynamics" by *Leonard Bairstow*, we consider the partial derivatives

$$c_k = \frac{\partial b_k}{\partial u}, \quad d_k = \frac{\partial b_{k-1}}{\partial v} \quad (0 \le k \le n). \tag{2.65}$$

- Differentiating the **recurrence relation**, (2.60), results in the following pair of additional recurrences:

$$\begin{aligned} c_k &= b_{k+1} + uc_{k+1} + vc_{k+2} \quad (c_{n+1} = c_{n+2} = 0) \\ d_k &= b_{k+1} + ud_{k+1} + vd_{k+2} \quad (d_{n+1} = d_{n+2} = 0) \end{aligned} \tag{2.66}$$

  Note that these recurrence relations obviously generate the same two sequences ($c_k = d_k$); we need only the first.

- The Jacobian explicitly reads

$$J = \frac{\partial(b_0, b_1)}{\partial(u, v)} = \begin{bmatrix} c_0 & c_1 \\ c_1 & c_2 \end{bmatrix}, \tag{2.67}$$

  and therefore

$$\begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = -J^{-1} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \frac{1}{c_0 c_2 - c_1^2} \begin{bmatrix} b_1 c_1 - b_0 c_2 \\ b_0 c_1 - b_1 c_0 \end{bmatrix}. \tag{2.68}$$

We summarize the above procedure as in the following code:

```
Bairstow := proc(n, a, u0, v0, itmax, TOL)
   local u, v, b, c, j, k, DetJ, du, dv, s1, s2;
   u := u0;
   v := v0;
   b := Array(0..n);
   c := Array(0..n);
   b[n] := a[n];
   c[n] := 0;
   c[n-1] := a[n];
   for j to itmax do
      b[n-1] := a[n-1] + u*b[n];
      for k from (n-2) by -1 to 0 do
         b[k] := a[k] + u*b[k+1] + v*b[k+2];
         c[k] := b[k+1] + u*c[k+1] + v*c[k+2];
      end do;
      DetJ := c[0]*c[2] - c[1]*c[1];
      du := (c[1]*b[1] - c[2]*b[0]) / DetJ;
      dv := (c[1]*b[0] - c[0]*b[1]) / DetJ;
      u := u + du;
      v := v + dv;
      printf("%3d %12.7f %12.7f %12.4g %12.4g\n", j, u, v, du, dv);
      if (max(abs(du), abs(dv)) < TOL) then break end if;
   end do;
   # Post-processing
   printf("  Q(x) = (%g)x^%d", b[n], n-2);
   for k from n-3 by -1 to 1 do
      printf(" + (%g)x^%d", b[k+2], k);
   end do;
   printf(" + (%g)\n", b[2]);
   printf("  Remainder: %g (x - (%g)) + (%g)\n", b[1], u, b[0]);
   printf("  Quadratic Factor: x^2 - (%g)x - (%g)\n", u, v);
   s1 := evalf(u + sqrt(u*u + 4*v)) / 2;
   if ((u^2 + 4*v) < 0) then
      printf("  Zeros: %.13g +- (%.13g) i\n", Re(s1), abs(Im(s1)));
   else
      s2 := evalf(u - sqrt(u*u + 4*v)) / 2;
      printf("  Zeros: %.13g, %.13\n", s1, s2);
   end if;
end proc:
```

Figure 2.10: Bairstow's method.

Run Bairstow

```
P := x -> x^4 - 4*x^3 + 7*x^2 - 5*x - 2:
n := degree(P(x)):
a := Array(0..n):
for i from 0 to n do
    a[i] := coeff(P(x), x, i);
end do:
itmax := 10: TOL := 10^-10:

u := 3:
v := -4:
Bairstow(n, a, u, v, itmax, TOL);
   1     2.2000000    -2.7000000          -0.8            1.3
   2     2.2727075    -3.9509822       0.07271         -1.251
   3     2.2720737    -3.6475280    -0.0006338         0.3035
   4     2.2756100    -3.6274260      0.003536         0.0201
   5     2.2756822    -3.6273651     7.215e-05      6.090e-05
   6     2.2756822    -3.6273651     6.316e-09     -9.138e-09
   7     2.2756822    -3.6273651    -1.083e-17     -5.260e-17
  Q(x) = (1)x^2 + (-1.72432)x^1 + (-0.551364)
  Remainder: -2.66446e-18 (x - (2.27568)) + (-2.47514e-16)
  Quadratic Factor: x^2 - (2.27568)x - (-3.62737)
  Zeros:  1.137841102 +- (1.527312251) i
```

Deflation

- Given a polynomial of degree n, $P(x)$, if the Newton's method finds a zero (say, $\widehat{x}_1$), it will be written as

$$P(x) \approx (x - \widehat{x}_1)Q_1(x). \tag{2.69}$$

- Then, we can find a second approximate zero $\widehat{x}_2$ (or, a quadratic factor) of P by applying Newton's method to the reduced polynomial $Q_1(x)$:

$$Q_1(x) \approx (x - \widehat{x}_2)Q_2(x). \tag{2.70}$$

- The computation continues up to the point that P is factorized by linear and quadratic factors. The procedure is called **deflation**.

Remark 2.43.

- The deflation process introduces an accuracy issue, due to the fact that when we obtain the approximate zeros of $P(x)$, the Newton's method is applied to the reduced polynomials $Q_k(x)$.
- An approximate zero $\widehat{x}_{k+1}$ of $Q_k(x)$ will generally not approximate a root of $P(x) = 0$; inaccuracy increases as k increases.
- One way to overcome the difficulty is to improve the approximate zeros; *starting with these zeros, apply the Newton's method with the original polynomial* $P(x)$.

Exercises for Chapter 2

2.1. Let the bisection method be applied to a continuous function, resulting in intervals $[a_1, b_1]$, $[a_2, b_2]$, $\cdots$. Let $p_n = (a_n + b_n)/2$ and $p = \lim_{n\to\infty} p_n$. Which of these statements can be false?

(a) $a_1 \le a_2 \le \cdots$

(b) $|p - p_n| \le \dfrac{b_1 - a_1}{2^n}, \quad n \ge 1$

(c) $|p - p_{n+1}| \le |p - p_n|, \quad n \ge 1$

(d) $[a_{n+1}, b_{n+1}] \subset [a_n, b_n]$

(e) $|p - p_n| = \mathcal{O}\left(\dfrac{1}{2^n}\right)$ as $n \to \infty$

Ans: (c)

2.2. C Modify the Matlab code used in Example 2.7 for the bisection method to incorporate

$$\begin{cases} \text{Inputs :} & \text{f, a, b, TOL, itmax} \\ \text{Stopping criterion :} & \text{Relative error} \le \text{TOL or } k \le \text{itmax} \end{cases}$$

Consider the following equations defined on the given intervals:

I. $3x - e^x = 0, \quad [0, 1]$

II. $2x\cos(2x) - (x+1)^2 = 0, \quad [-1, 0]$

For each of the above equations,

(a) Use Maple or Matlab (or something else) to find a very accurate solution in the interval.

(b) Find the approximate root by using your Matlab with `TOL=`10^{-6} and `itmax=10`.

(c) Report p_n, $|p - p_n|$, and $|p - p_{n-1}|$, for $n \ge 1$, in a table format.

Ans:

Table 2.1: Results of Bisection Method (Problem II)

Iteration	p_n	$\lvert p - p_n \rvert$	$\lvert p - p_{n-1} \rvert$
1	-0.7500	0.0482	0.2982
2	-0.8750	0.0768	0.0482
3	-0.8125	0.0143	0.0768
4	-0.7812	0.0169	0.0143
5	-0.7969	0.0013	0.0169
6	-0.8047	0.0065	0.0013
7	-0.8008	0.0026	0.0065
8	-0.7988	0.0007	0.0026
9	-0.7979	0.0003	0.0007
10	-0.7983	0.0002	0.0003

2.3. C Let us try to find $5^{1/3}$ by the fixed-point method. Use the fact that the result must be the positive solution of $f(x) = x^3 - 5 = 0$ to solve the following:

(a) Introduce two different fixed-point forms which are convergent for $x \in [1, 2]$.
(b) Perform five iterations for each of the iterations with $p_0 = 1.5$, and measure $|p - p_5|$.
(c) Rank the associated iterations based on their apparent speed of convergence with $p_0 = 1.5$. Discuss why one is better than the other.

2.4. **Kepler's equation** in astronomy reads

$$y = x - \varepsilon \sin(x), \quad \text{with } 0 < \varepsilon < 1. \tag{2.71}$$

(a) Show that for each $y \in [0, \pi]$, there exists an x satisfying the equation.
(b) Interpret this as a fixed-point problem.
(c) C Find x's for $y = 1, \pi/2, 2$, using the fixed-point iteration. Set $\varepsilon = 1/2$.

Hint: For (a), you may have to use the IVT for $x - \varepsilon * sin(x)$ defined on $[0, \pi]$, while for (b) you should rearrange the equation in the form of $x = g(x)$. For (c), you may use any source of program which utilizes the fixed-point iteration.

2.5. Consider a variation of Newton's method in which only one derivative is needed; that is,

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_0)}, \quad n \geq 1. \tag{2.72}$$

Find C and s such that

$$e_n \approx C e_{n-1}^s. \tag{2.73}$$

Hint: You may have to use $f(p_{n-1}) = e_{n-1} f'(p_{n-1}) - \frac{1}{2} e_{n-1}^2 f''(\xi_{n-1})$.

2.6. (**Note**: Do not use programming for this problem.) Starting with $\mathbf{x}_0 = (0,1)^T$, carry out two iterations of the Newton's method on the system:

$$\begin{cases} 4x^2 - y^2 &= 0 \\ 4xy^2 - x &= 1 \end{cases}$$

Hint: Define $f_1(x,y) = 4x^2 - y^2$, $f_2(x,y) = 4xy^2 - x - 1$. Then try to use (2.37)-(2.38), p.61. Note that $J(x,y) = \begin{bmatrix} 8x & -2y \\ 4y^2 - 1 & 8xy \end{bmatrix}$. Thus, for example, $J(x_0,y_0) = \begin{bmatrix} 0 & -2 \\ 3 & 0 \end{bmatrix}$ and $\begin{bmatrix} f_1(x_0,y_0) \\ f_2(x_0,y_0) \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$. Now you can find the correction vector to update iterate and get $\mathbf{x}_1$. Do it once more for $\mathbf{x}_2$.

2.7. C Consider the polynomial

$$P(x) = 3x^5 - 7x^4 - 5x^3 + x^2 - 8x + 2.$$

(a) Use the Horner's algorithm to find $P(4)$.

(b) Use the Newton's method to find a real-valued root, starting with $x_0 = 4.$ and applying the Horner's algorithm for the evaluation of $P(x_k)$ and $P'(x_k)$.

(c) Apply the Bairstow's method, with the initial point $(u,v) = (0,-1)$, to find a pair of complex-valued zeros.

(d) Find a disk centered at the origin that contains all the roots.

Ans: (c) Quadratic Factor $= x^2 - 0.3275x + 0.7703$. Thus, the complex-valued solutions are $x_1 = 0.1637 + 0.8623i$ and $x_2 = 0.1637 - 0.8623i$.

CHAPTER 3

Interpolation and Polynomial Approximation

This chapter introduces the following.

Topics	Applications/Properties
Polynomial interpolation	The first step toward approximation theory
Newton form	
Lagrange form	Basis functions for various applications including visualization and FEMs
Chebyshev polynomial	Optimized interpolation
Divided differences	
Neville's method	Evaluation of interpolating polynomials
Hermite interpolation	It incorporates $f(x_i)$ and $f'(x_i)$
Spline interpolation	Less oscillatory interpolation
B-splines	
Parametric curves	Curves in the plane or the space

Contents of Chapter 3

3.1. Polynomial Interpolation 82
3.2. Divided Differences 101
3.3. Data Approximation and Neville's Method 106
3.4. Hermite Interpolation 110
3.5. Spline Interpolation 114
3.6. Parametric Curves 126
Exercises for Chapter 3 131

3.1. Polynomial Interpolation

Each continuous function can be approximated (arbitrarily close) by a polynomial, and polynomials of degree n interpolating values at $(n+1)$ distinct points are all the same polynomial, as shown in the following theorems.

Theorem 3.1. ***(Weierstrass approximation theorem)**: Suppose $f \in C[a,b]$. Then, for each $\varepsilon > 0$, there exists a polynomial $P(x)$ such that*

$$|f(x) - P(x)| < \varepsilon, \quad \textit{for all } x \in [a,b]. \tag{3.1}$$

Example 3.2. Let $f(x) = e^x$. Then

$$f := x \mapsto e^x$$

$$taylor(f(x), x = 0, 7) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + O(x^7)$$

$$p0 := x \to 1 :$$

$$p2 := x \to 1 + x + \frac{1}{2}x^2 :$$

$$p4 := x \to 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 :$$

$$p6 := x \to 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 :$$

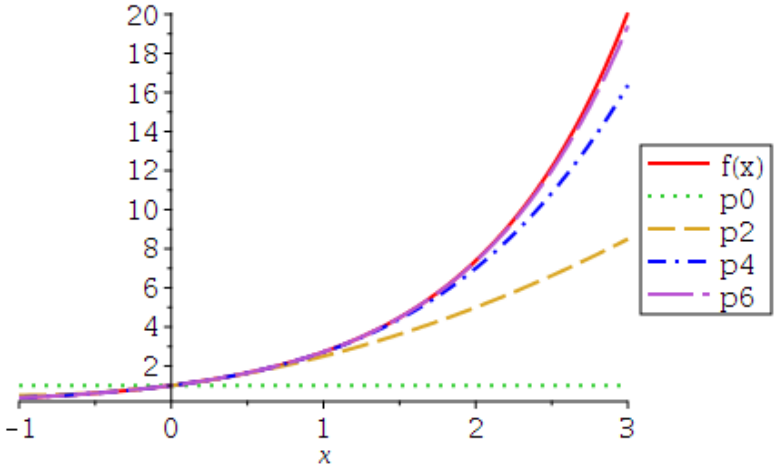


Figure 3.1: Polynomial approximations for $f(x) = e^x$.

Theorem 3.3. ***(Polynomial Interpolation Theorem)**:*
If $x_0, x_1, x_2, \cdots, x_n$ are $(n+1)$ distinct real numbers, then for arbitrary values $y_0, y_1, y_2, \cdots, y_n$, there is a unique polynomial p_n of degree at most n such that

$$p_n(x_i) = y_i \quad (0 \le i \le n). \tag{3.2}$$

Proof. *(Uniqueness)*.
Suppose there were two such polynomials, p_n and q_n. Then $p_n - q_n$ would have the property

$$(p_n - q_n)(x_i) = 0, \quad \text{for } 0 \le i \le n. \tag{3.3}$$

Since the degree of $p_n - q_n$ is at most n, the polynomial can have at most n zeros unless it is a zero polynomial. Since x_i are distinct, $p_n - q_n$ has $n+1$ zeros and therefore it must be 0. Hence,

$$p_n \equiv q_n.$$

(Existence).
For the existence part, we proceed ***inductively through construction***.

- For $n = 0$, the existence is obvious since we may choose the constant function

$$p_0(x) = y_0. \tag{3.4}$$

- Now suppose that we have obtained a polynomial p_{k-1} of degree $\le k-1$ with

$$p_{k-1}(x_i) = y_i, \quad \text{for } 0 \le i \le k-1. \tag{3.5}$$

- We try to construct p_k in the form

$$p_k(x) = p_{k-1}(x) + c_k(x - x_0)(x - x_1)\cdots(x - x_{k-1}) \tag{3.6}$$

  for some c_k.

  (a) Note that (3.6) is unquestionably a polynomial of degree $\le k$.
  (b) Furthermore, p_k interpolates the data that p_{k-1} interpolates:

$$p_k(x_i) = p_{k-1}(x_i) = y_i, \quad 0 \le i \le k-1. \tag{3.7}$$

- Now we determine the constant c_k to satisfy the condition

$$p_k(x_k) = y_k, \tag{3.8}$$

which leads to

$$p_k(x_k) = p_{k-1}(x_k) + c_k(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1}) = y_k. \tag{3.9}$$

This equation can certainly be solved for c_k:

$$c_k = \frac{y_k - p_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1)\cdots(x_k - x_{k-1})}, \tag{3.10}$$

because the denominator is not zero. □

3.1.1. Newton Form of the Interpolating Polynomials

As in the proof of the previous theorem, each p_k $(k \geq 1)$ is obtained by adding a single term to p_{k-1}. Thus, at the end of the process, p_n will be a sum of terms and $p_0,\ p_1,\ \cdots,\ p_{n-1}$ will be easily visible in the expression of p_n. Each p_k has the form

$$p_k(x) = c_0 + c_1(x - x_0) + \cdots + c_k(x - x_0)(x - x_1)\cdots(x - x_{k-1}). \tag{3.11}$$

The compact form of this reads

$$p_k(x) = \sum_{i=0}^{k} c_i \prod_{j=0}^{i-1}(x - x_j). \tag{3.12}$$

(Here the convention has been adopted that $\prod_{j=0}^{m}(x - x_j) = 1$ when $m < 0$.)

The first few cases of (3.12) are

$$\begin{aligned} p_0(x) &= c_0, \\ p_1(x) &= c_0 + c_1(x - x_0), \\ p_2(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1). \end{aligned} \tag{3.13}$$

These polynomials are called the **interpolating polynomials in Newton form**, or **Newton form of interpolating polynomials**.

Illustration of Newton's interpolating polynomials

Example 3.4. Let $f(x) = \sin(x \cdot (x-1)) + 1$. Let $[0, 0.5, 1.0, 2.0, 1.5]$ be a collection of distinct points. Find Newton's interpolating polynomials which pass $\{(x_i, f(x_i)) \mid i \leq k\}$ for each k.

A Maple implementation

```
restart:
with(Student[NumericalAnalysis]):
f := x -> sin(x*(x - 1)) + 1:

xk := 0:
xy := [[xk, f(xk)]]:
P0 := PolynomialInterpolation(xy, independentvar = x,
                              method = newton, function = f);
p0 := x -> Interpolant(P0):
p0(x)
                                 1

xk := 0.5:
P1 := AddPoint(P0, [xk, f(xk)]):
p1 := x -> Interpolant(P1):
p1(x)
                       1. - 0.4948079186 x

xk := 1.0:
P2 := AddPoint(P1, [xk, f(xk)]):
p2 := x -> Interpolant(P2):
p2(x)
          1. - 0.4948079186 x + 0.9896158372 x (x - 0.5)

xk := 2.0:
P3 := AddPoint(P2, [xk, f(xk)]):
p3 := x -> Interpolant(P3):
p3(x)
          1. - 0.4948079186 x + 0.9896158372 x (x - 0.5)
             - 0.3566447492 x (x - 0.5) (x - 1.0)

```

```
xk := 1.5:
P4 := AddPoint(P3, [xk, f(xk)]):
p4 := x -> Interpolant(P4):
p4(x)
       1. - 0.4948079186 x + 0.9896158372 x (x - 0.5)
          - 0.3566447492 x (x - 0.5) (x - 1.0)
          - 0.5517611839 x (x - 0.5) (x - 1.0) (x - 2.0)
```

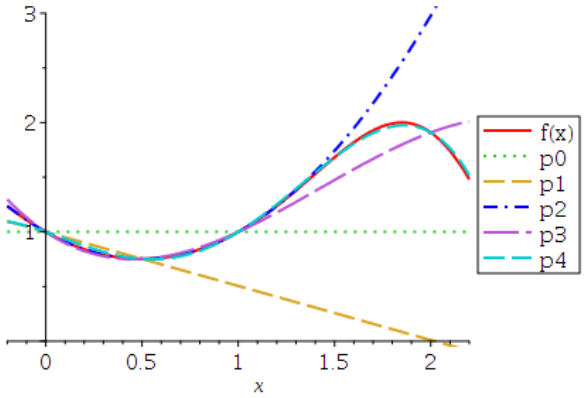


Figure 3.2: Illustration of Newtons interpolating polynomials, with $f(x) = \sin(x \cdot (x-1)) + 1$, at $[0, 0.5, 1.0, 2.0, 1.5]$.

Evaluation of $p_k(x)$, assuming that c_0, c_1, $\cdots$, c_k are known:

We may use an efficient method called **nested multiplication** or **Horner's method**. This can be explained most easily for an arbitrary expression of the form

$$u = \sum_{i=0}^{k} c_i \prod_{j=0}^{i-1} d_j. \tag{3.14}$$

The idea begins with rewriting it in the form

$$\begin{aligned} u &= c_0 + c_1 d_0 + c_2 d_0 d_1 + \cdots + c_{k-1} d_0 d_1 \cdots d_{k-2} + c_k d_0 d_1 \cdots d_{k-1} \\ &= c_k d_0 d_1 \cdots d_{k-1} + c_{k-1} d_0 d_1 \cdots d_{k-2} + \cdots + c_2 d_0 d_1 + c_1 d_0 + c_0 \\ &= (c_k d_1 \cdots d_{k-1} + c_{k-1} d_1 \cdots d_{k-2} + \cdots + c_2 d_1 + c_1) d_0 + c_0 \\ &= ((c_k d_2 \cdots d_{k-1} + c_{k-1} d_2 \cdots d_{k-2} + \cdots + c_2) d_1 + c_1) d_0 + c_0 \\ &\quad \ddots \\ &= (\cdots (((c_k) d_{k-1} + c_{k-1}) d_{k-2} + c_{k-2}) d_{k-3} + \cdots + c_1) d_0 + c_0 \end{aligned} \tag{3.15}$$

Algorithm 3.5. (Nested Multiplication). Thus the algorithm for the evaluation of u in (3.14) can be written as

```
u := c[k];
for i from k-1 by -1 to 0 do
    u := u*d[i] + c[i];
end do
```

The computation of c_k, using Horner's algorithm

Algorithm 3.6. The Horner's algorithm for the computation of coefficients c_k in Equation (3.12) gives

```
c[0] := y[0];
for k to n do
    d := x[k] - x[k-1];
    u := c[k-1];
    for i from k-2 by -1 to 0 do
        u := u*(x[k] - x[i]) + c[i];
        d := d*(x[k] - x[i]);
    end do;
    c[k] := (y[k] - u)/d;
end do
```

A more efficient procedure exists that achieves the same result. The alternative method uses **divided differences** to compute the coefficients c_k. The method will be presented later.

Example 3.7. Let

$$f(x) = 4x^3 + 35x^2 - 84x - 954.$$

Four values of this function are given as

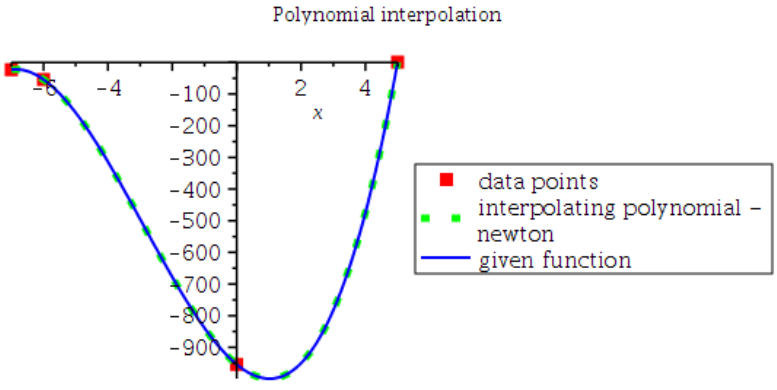
x_i	5	-7	-6	0
y_i	1	-23	-54	-954

Construct the Newton form of the polynomial from the data.

Solution.

Maple-code

```
with(Student[NumericalAnalysis]):
f := 4*x^3 + 35*x^2 - 84*x - 954:
xy := [[5, 1], [-7, -23], [-6, -54], [0, -954]]:
N := PolynomialInterpolation(xy, independentvar = x,
     method = newton, function = f):
Interpolant(N)
    -9 + 2 x + 3 (x - 5) (x + 7) + 4 (x - 5) (x + 7) (x + 6)
# Since "-9 + 2*x = 1 + 2*(x - 5)", the coefficients are
#     "c[0] = 1, c[1] = 2, c[2] = 3, c[3] = 4"
expand(Interpolant(N));
                        3       2
                   4 x  + 35 x  - 84 x - 954
# which is the same as f
RemainderTerm(N);
            0 &where {-7 <= xi_var and xi_var <= 5}
Draw(N);
```



```
DividedDifferenceTable(N);
```

$$\begin{bmatrix} \underline{1} & 0 & 0 & 0 \\ -23 & \underline{2} & 0 & 0 \\ -54 & -31 & \underline{3} & 0 \\ -954 & -150 & -17 & \underline{4} \end{bmatrix}$$

Example 3.8. Find the Newton form of the interpolating polynomial of the data.

x_i	2	-1	1
y_i	1	4	-2

Solution.

Ans: $p_2(x) = 1 - (x-2) + 2(x-2)(x+1)$

3.1.2. Lagrange Form of Interpolating Polynomials

Let data points (x_k, y_k), $0 \le k \le n$ be given, where $n+1$ *abscissas* x_i are distinct. The interpolating polynomial will be sought in the form

$$p_n(x) = y_0 L_{n,0}(x) + y_1 L_{n,1}(x) + \cdots + y_n L_{n,n}(x) = \sum_{k=0}^{n} y_k L_{n,k}(x), \tag{3.16}$$

where $L_{n,k}(x)$ are polynomials that depend on the nodes $x_0, x_1, \cdots, x_n$, but not on the *ordinates* $y_0, y_1, \cdots, y_n$.

How to determine the basis $\{L_{n,k}(x)\}$

Observation 3.9. Let all the ordinates be 0 except for a 1 occupying i-th position, that is, $y_i = 1$ and other ordinates are all zero.

- Then,
$$p_n(x_j) = \sum_{k=0}^{n} y_k L_{n,k}(x_j) = L_{n,i}(x_j). \tag{3.17}$$
- On the other hand, the polynomial p_n interpolating the data must satisfy $p_n(x_j) = \delta_{ij}$, where δ_{ij} is the ***Kronecker delta***
$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$
- Thus all the basis polynomials must satisfy
$$L_{n,i}(x_j) = \delta_{ij}, \quad \text{for all } 0 \le i, j \le n. \tag{3.18}$$
Polynomials satisfying such a property are known as the **cardinal functions**.

Example **3.10. Construction of** $L_{n,0}(x)$: It is to be an nth-degree polynomial that takes the value 0 at $x_1, x_2, \cdots, x_n$ and the value 1 at x_0. Clearly, it must be of the form

$$L_{n,0}(x) = c(x - x_1)(x - x_2)\cdots(x - x_n) = c\prod_{j=1}^{n}(x - x_j), \tag{3.19}$$

where c is determined for which $L_{n,0}(x_0) = 1$. That is,

$$1 = L_{n,0}(x_0) = c(x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_n) \tag{3.20}$$

and therefore

$$c = \frac{1}{(x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_n)}. \tag{3.21}$$

Hence, we have

$$L_{n,0}(x) = \frac{(x - x_1)(x - x_2)\cdots(x - x_n)}{(x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_n)} = \prod_{j=1}^{n}\frac{(x - x_j)}{(x_0 - x_j)}. \tag{3.22}$$

Summary **3.11.** *Each cardinal function is obtained by similar reasoning; the general formula is then*

$$L_{n,i}(x) = \prod_{j=0,\, j\neq i}^{n}\frac{(x - x_j)}{(x_i - x_j)}, \quad i = 0, 1, \cdots, n. \tag{3.23}$$

Example **3.12.** Find the Lagrange form of interpolating polynomial for the two-point table

x	x_0	x_1
y	y_0	y_1

Solution.

Example 3.13. Determine the Lagrange interpolating polynomial that passes through $(2, 4)$ and $(5, 1)$.

Solution.

Example 3.14. Let $x_0 = 2$, $x_1 = 4$, $x_2 = 5$

(a) Use the points to find the second Lagrange interpolating polynomial p_2 for $f(x) = 1/x$.

(b) Use p_2 to approximate $f(3) = 1/3$.

Solution.

Maple-code

```
with(Student[NumericalAnalysis]);
f := x -> 1/x:
unassign('xy'):
xy := [[2, 1/2], [4, 1/4], [5, 1/5]]:

L2 := PolynomialInterpolation(xy, independentvar = x,
       method = lagrange, function = f(x)):
Interpolant(L2);
  1                    1                    1
  -- (x - 4) (x - 5) - - (x - 2) (x - 5) + -- (x - 2) (x - 4)
  12                   8                    15
RemainderTerm(L2);
/  (x - 2) (x - 4) (x - 5)\
|- ----------------------| &where {2 <= xi_var and xi_var <= 5}
|                4       |
\          xi_var        /
p2 := x -> expand(Interpolant(L2));
                        1   2   11     19
                        -- x  - -- x + --
                        40      40     20
evalf(p2(3));
                       0.3500000000
```

3.1.3. Polynomial interpolation error

Theorem 3.15. ***(Polynomial Interpolation Error Theorem).*** *Let $f \in C^{n+1}[a,b]$, and let P_n be the polynomial of degree $\leq n$ that interpolates f at $n+1$ distinct points $x_0, x_1, \cdots, x_n$ in the interval $[a,b]$. Then, for each $x \in (a,b)$, there exists a number ξ_x between $x_0, x_1, \cdots, x_n$, hence in the interval $[a,b]$, such that*

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!} \prod_{i=0}^{n} (x - x_i) =: R_n(x). \tag{3.24}$$

Recall: **Theorem 1.22.** **(Taylor's Theorem with Lagrange Remainder)**, page 9. Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b), and $x_0 \in [a,b]$. Then, for every $x \in [a,b]$,

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \mathcal{R}_n(x), \tag{3.25}$$

where, for some ξ between x and x_0,

$$\mathcal{R}_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}.$$

Example 3.16. For Example 3.14, determine the error bound in $[2,5]$.

Solution.

Maple-code

```
p2 := x -> interp([2, 4, 5], [1/2, 1/4, 1/5], x):
p2(x):
f := x -> 1/x:
fd := x -> diff(f(x), x, x, x):
fd(xi)
                                  6
                                - ---
                                    4
                                  xi
fdmax := maximize(abs(fd(x)), x = 2..5)
                                  3
                                  -
                                  8
r := x -> (x - 2)*(x - 4)*(x - 5):
rmax := maximize(abs(r(x)), x = 2..5);
          /5   1  (1/2)\ /4   1  (1/2)\ /1   1  (1/2)\
          |- - - 7     | |- + - 7     | |- + - 7     |
          \3   3       / \3   3       / \3   3       /
#Thus, "|f(x)-p2(x)|<=(max)|R[2](x)|="
evalf(fdmax*rmax/3!)
                            0.1320382370
```

Example 3.17. If the function $f(x) = \sin(x)$ is approximated by a polynomial of degree 5 that interpolates f at six equally distributed points in $[-1, 1]$ including end points, how large is the error on this interval?

Solution. The nodes x_i are $-1, -0.6, -0.2, 0.2, 0.6$, and 1. It is easy to see that

$$|f^{(6)}(\xi)| = |-\sin(\xi)| \leq \sin(1).$$

```
g := x -> (x+1)*(x+0.6)*(x+0.2)*(x-0.2)*(x-0.6)*(x-1):
gmax := maximize(abs(g(x)), x = -1..1)
                              0.06922606316
```

Thus,

$$|\sin(x) - P_5(x)| = \left|\frac{f^{(6)}(\xi)}{6!}\prod_{i=0}^{5}(x - x_i)\right| \leq \frac{\sin(1)}{6!}\texttt{gmax} \tag{3.26}$$
$$= 0.00008090517158$$

Theorem 3.18. (Polynomial Interpolation Error Theorem for Equally Spaced Nodes): *Let $f \in C^{n+1}[a, b]$, and let P_n be the polynomial of degree $\leq n$ that interpolates f at*

$$x_i = a + ih, \quad h = \frac{b-a}{n}, \quad i = 0, 1, \cdots, n.$$

Then, for each $x \in (a, b)$,

$$|f(x) - P_n(x)| \leq \frac{h^{n+1}}{4(n+1)}M, \tag{3.27}$$

where

$$M = \max_{\xi\in[a,b]} |f^{(n+1)}(\xi)|.$$

Proof. Recall the interpolation error $R_n(x)$ given in (3.24). We consider bounding

$$\max_{x\in[a,b]} \prod_{j=0}^{n} |x - x_i|.$$

Start by picking an x. We can assume that x is not one of the nodes, because

otherwise the product in question is zero. Let $x \in (x_j, x_{j+1})$, for some j. Then we have

$$|x - x_j| \cdot |x - x_{j+1}| \leq \frac{h^2}{4}. \tag{3.28}$$

Now note that

$$|x - x_i| \leq \begin{cases} (j+1-i)h & \text{for } i < j \\ (i-j)h & \text{for } j+1 < i. \end{cases} \tag{3.29}$$

Thus

$$\prod_{j=0}^{n} |x - x_i| \leq \frac{h^2}{4}[(j+1)!\, h^j]\,[(n-j)!\, h^{n-j-1}]. \tag{3.30}$$

Since $(j+1)!(n-j)! \leq n!$, we can reach the following bound

$$\prod_{j=0}^{n} |x - x_i| \leq \frac{1}{4} h^{n+1} n!. \tag{3.31}$$

The result of the theorem follows from the above bound. □

Example 3.19. How many equally spaced nodes are required to interpolate $f(x) = \cos x + \sin x$ to within 10^{-8} on the interval $[-1, 1]$?

Solution. Recall the formula: $|f(x) - P_n(x)| \leq \dfrac{h^{n+1}}{4(n+1)} M$. Then, for n, solve

$$\frac{(2/n)^{n+1}}{4(n+1)} \sqrt{2} \leq 10^{-8}.$$

Ans: $n = 10$

3.1.4. Chebyshev polynomials

Recall the **Polynomial Interpolation Error** in (3.24), p. 93:

$$f(x) - P_n(x) \;=\; \frac{f^{(n+1)}(\xi_x)}{(n+1)!}\prod_{\mathbf{i=0}}^{\mathbf{n}}(\mathbf{x}-\mathbf{x_i}) \;=:\; R_n(x).$$

There is a term that can be minimized by choosing the nodes in a special way. **An analysis of this problem** was first given by a great mathematician **Chebyshev** (1821-1894). The optimization process leads naturally to a system of polynomials called **Chebyshev polynomials**.

Definition 3.20. The **Chebyshev polynomials** (of the first kind) are defined recursively as follows:

$$\begin{cases} T_0(x) = 1, \quad T_1(x) = x \\ T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x), \;\; n \geq 1. \end{cases} \tag{3.32}$$

The explicit forms of the next few T_n are readily calculated:

Chebyshev-polynomials

```
T2 := x -> simplify(ChebyshevT(2, x)): T2(x)
                                2
                             2 x  - 1

T3 := x -> simplify(ChebyshevT(3, x)): T3(x)
                              3
                           4 x  - 3 x

T4 := x -> simplify(ChebyshevT(4, x)): T4(x)
                            4      2
                         8 x  - 8 x  + 1

T5 := x -> simplify(ChebyshevT(5, x)): T5(x)
                            5       3
                        16 x  - 20 x  + 5 x
```

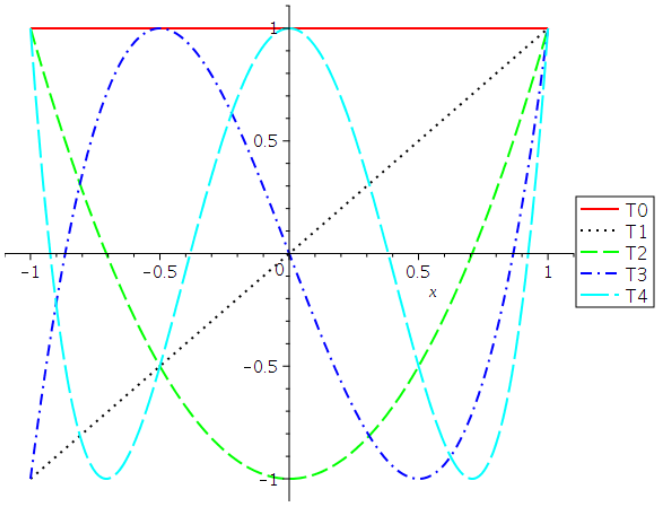


Figure 3.3: Chebyshev polynomials.

Theorem 3.21. (Properties of Chebyshev polynomials):

(a) $T_n(x) \in [-1, 1]$, *for all* $x \in [-1, 1]$, *and the leading coefficient of* $T_n(x)$ *is* 2^{n-1}.

(b) The Chebyshev polynomials have this closed-form expression:

$$T_n(x) = \cos\left(n \cos^{-1}(x)\right), \quad n \geq 0. \tag{3.33}$$

(c) It has been verified that if the nodes $x_0, x_1, \cdots, x_n \in [-1, 1]$, *then*

$$\max_{|x|\leq 1} \left| \prod_{i=0}^{n} (x - x_i) \right| \geq 2^{-n}, \quad n \geq 0, \tag{3.34}$$

and its minimum value will be attained if

$$\prod_{i=0}^{n} (x - x_i) = 2^{-n} T_{n+1}(x). \tag{3.35}$$

(d) The nodes then must be the roots of T_{n+1}, *which are*

$$x_i = \cos\left(\frac{(2i+1)\pi}{2n+2}\right), \quad i = 0, 1, \cdots, n. \tag{3.36}$$

Theorem 3.22. ***(Interpolation Error Theorem, Chebyshev nodes)****: If the nodes are the roots of the Chebyshev polynomial T_{n+1}, as in (3.36), then the error bound for the nth-degree interpolating polynomial P_n reads*

$$|f(x) - P_n(x)| \leq \frac{1}{2^n(n+1)!} \max_{|t|\leq 1} \left|f^{(n+1)}(t)\right|. \tag{3.37}$$

Example 3.23. (A variant of Example 3.17): If the function $f(x) = \sin(x)$ is approximated by a polynomial of degree 5 that interpolates f at at roots of the Chebyshev polynomial T_6 in $[-1, 1]$, how large is the error on this interval?

Solution. From Example 3.17, we know that

$$|f^{(6)}(\xi)| = |-\sin(\xi)| \leq \sin(1).$$

Thus

$$|f(x) - P_5(x)| \leq \frac{\sin(1)}{2^n(n+1)!} = 0.00003652217816. \tag{3.38}$$

It is an optimal upper bound of the error and smaller than the one in Equation (3.26), 0.00008090517158.

Accuracy comparison between uniform nodes and Chebyshev nodes:

Maple-code

```
with(Student[NumericalAnalysis]):
n := 5:
f := x -> sin(2*x*Pi):
xd := Array(0..n):

for i from 0 to n do
    xd[i] := evalf[15](-1 + (2*i)/n);
end do:
xyU := [[xd[0],f(xd[0])], [xd[1],f(xd[1])], [xd[2],f(xd[2])],
        [xd[3],f(xd[3])], [xd[4],f(xd[4])], [xd[5],f(xd[5])]]:
U := PolynomialInterpolation(xyU, independentvar = x,
     method = lagrange, function = f(x)):
pU := x -> Interpolant(U):
```

```
for i from 0 to n do
    xd[i] := evalf[15](cos((2*i + 1)*Pi/(2*n + 2)));
end do:
xyC := [[xd[0],f(xd[0])], [xd[1],f(xd[1])], [xd[2],f(xd[2])],
        [xd[3],f(xd[3])], [xd[4],f(xd[4])], [xd[5],f(xd[5])]]:
C := PolynomialInterpolation(xyC, independentvar = x,
       method = lagrange, function = f(x)):
pC := x -> Interpolant(C):

plot([pU(x), pC(x)], x = -1..1, thickness = [2,2],
    linestyle = [solid, dash], color = [red, blue],
    legend = ["Uniform nodes", "Chebyshev nodes"],
    legendstyle = [font = ["HELVETICA", 13], location = bottom])
```

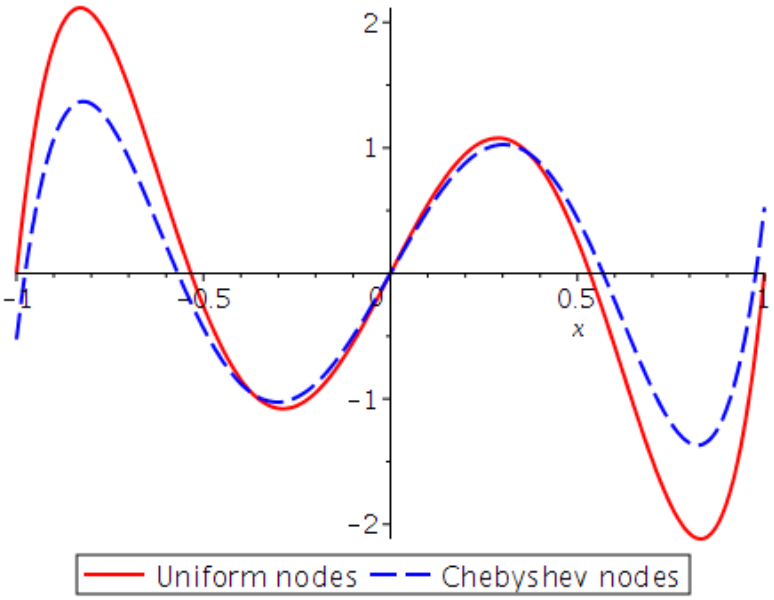


Figure 3.4: Accuracy comparison between uniform nodes and Chebyshev nodes.

3.2. Divided Differences

It turns out that the coefficients c_k for the interpolating polynomials in Newton's form can be calculated relatively easily by using **divided differences**.

Remark 3.24. For $\{(x_k, y_k)\}$, $0 \le k \le n$, the kth-degree Newton interpolating polynomials are of the form

$$p_k(x) = c_0 + c_1(x - x_0) + \cdots + c_k(x - x_0)(x - x_1)\cdots(x - x_{k-1}), \tag{3.39}$$

for which $p_k(x_k) = y_k$. The first few cases are

$$\begin{aligned} p_0(x) &= c_0 = y_0, \\ p_1(x) &= c_0 + c_1(x - x_0), \\ p_2(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1). \end{aligned} \tag{3.40}$$

(a) The coefficient c_1 is determined to satisfy

$$y_1 = p_1(x_1) = c_0 + c_1(x_1 - x_0). \tag{3.41}$$

Note $c_0 = y_0$. Thus, we have

$$y_1 - y_0 = c_1(x_1 - x_0) \tag{3.42}$$

and therefore

$$c_1 = \frac{y_1 - y_0}{x_1 - x_0}. \tag{3.43}$$

(b) Now, since

$$y_2 = p_2(x_2) = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1),$$

it follows from the above, (3.42), and (3.43) that

$$\begin{aligned} c_2 &= \frac{y_2 - y_0 - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(y_2 - y_1) + (y_1 - y_0) - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= \frac{(y_2 - y_1) + c_1(x_1 - x_0) - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(y_2 - y_1)/(x_2 - x_1) - c_1}{x_2 - x_0}. \end{aligned} \tag{3.44}$$

Definition 3.25. (Divided differences):

- The **zeroth divided difference** of the function f with respect to x_i, denoted $f[x_i]$, is the value of at x_i:

$$f[x_i] = f(x_i) \tag{3.45}$$

- The remaining divided differences are defined recursively. The **first divided difference** of f with respect to x_i, x_{i+1} is defined as

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}. \tag{3.46}$$

- The **second divided difference** relative to x_i, x_{i+1}, x_{i+2} is defined as

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}. \tag{3.47}$$

- In general, the **kth divided difference** relative to $x_i, x_{i+1}, \cdots, x_{i+k}$ is defined as

$$f[x_i, x_{i+1}, \cdots, x_{i+k}] = \frac{f[x_{i+1}, \cdots, x_{i+k}] - f[x_i, \cdots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{3.48}$$

Note: It follows from Remark 3.24 that the coefficients of the Newton interpolating polynomials read

$$c_0 = f[x_0], \quad c_1 = f[x_0, x_1], \quad c_2 = f[x_0, x_1, x_2]. \tag{3.49}$$

In general,

$$c_k = f[x_0, x_1, \cdots, x_k]. \tag{3.50}$$

Newton's Divided Difference Table

x	f[x]	DD1 (f[,])	DD2 (f[, ,])	DD3 (f[, , ,])
x_0	$f[x_0]$			
x_1	$f[x_1]$	$f[x_0, x_1]$ $= \frac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_2	$f[x_2]$	$f[x_1, x_2]$ $= \frac{f[x_2]-f[x_1]}{x_2-x_1}$	$f[x_0, x_1, x_2]$ $= \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
x_3	$f[x_3]$	$f[x_2, x_3]$ $= \frac{f[x_3]-f[x_2]}{x_3-x_2}$	$f[x_1, x_2, x_3]$ $= \frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	$f[x_0, x_1, x_2, x_3]$ $= \frac{f[x_1,x_2,x_3]-f[x_0,x_1,x_2]}{x_3-x_0}$

(3.51)

Pseudocode **3.26. (Newton's Divided Difference Formula)**:

Input: (x_i, y_i), $i = 0, 1, \cdots, n$, saved as $F_{i,0} = y_i$

Output: $F_{i,i}$, $i = 0, 1, \cdots, n$

Step 1: For $i = 1, 2, \cdots, n$

For $j = 1, 2, \cdots, i$

$$F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$$

Step 2: Return $(F_{0,0}, F_{1,1}, \cdots, F_{n,n})$

Example **3.27.** Determine the Newton interpolating polynomial for the data:

x	0	1	2	5
y	1	-1	3	-189

Solution.

Ans: $f(x) = 1 - 2x + 3x(x-1) - 4x(x-1)(x-2)$

Theorem 3.28. (Properties of Divided Differences):

- If f is a polynomial of degree k, then
$$f[x_0, x_1, \cdots, x_n] = 0, \quad \text{for all } n > k. \tag{3.52}$$
- **Permutations in Divided Differences**: The divided difference is a symmetric function of its arguments. That is, if $z_0, z_1, \cdots, z_n$ is a permutation of $x_0, x_1, \cdots, x_n$, then
$$f[z_0, z_1, \cdots, z_n] = f[x_0, x_1, \cdots, x_n]. \tag{3.53}$$
- **Error in Newton Interpolation**: Let P be the polynomial of degree $\leq n$ that interpolates f at $n+1$ distinct nodes, $x_0, x_1, \cdots, x_n$. If t is a point different from the nodes, then
$$f(t) - P(t) = f[x_0, x_1, \cdots, x_n, t] \prod_{i=0}^{n} (t - x_i). \tag{3.54}$$

  Proof: Let Q be the polynomial of degree at most $(n+1)$ that interpolates f at nodes, $x_0, x_1, \cdots, x_n, t$. Then, we know that Q is obtained from P by adding one more term. Indeed,
$$Q(x) = P(x) + f[x_0, x_1, \cdots, x_n, t] \prod_{i=0}^{n} (x - x_i). \tag{3.55}$$
  Since $f(t) = Q(t)$, the result follows. □
- **Derivatives and Divided Differences**: If $f \in C^n[a, b]$ and if $x_0, x_1, \cdots, x_n$ are distinct points in $[a, b]$, then there exists a point $\xi \in (a, b)$ such that
$$f[x_0, x_1, \cdots, x_n] = \frac{1}{n!} f^{(n)}(\xi). \tag{3.56}$$

  Proof: Let p_{n-1} be the polynomial of degree at most $n-1$ that interpolates f at $x_0, x_1, \cdots, x_{n-1}$. By the **Polynomial Interpolation Error Theorem**, there exists a point $\xi \in (a, b)$ such that
$$f(x_n) - p_{n-1}(x_n) = \frac{1}{n!} f^{(n)}(\xi) \prod_{i=1}^{n-1} (x_n - x_i). \tag{3.57}$$
  On the other hand, by the previous theorem, we have
$$f(x_n) - p_{n-1}(x_n) = f[x_0, x_1, \cdots, x_n] \prod_{i=1}^{n-1} (x_n - x_i). \tag{3.58}$$
  The theorem follows from the comparison of above two equations. □

Self-study **3.29.** Prove that for $h > 0$,

$$f(x) - 2f(x+h) + f(x+2h) = h^2 f''(\xi), \tag{3.59}$$

for some $\xi \in (x, x+2h)$.

Hint: Use the last theorem; employ the divided difference formula to find $f[x, x+h, x+2h]$.

Solution.

3.3. Data Approximation and Neville's Method

Remark 3.30.

- We have studied how to construct interpolating polynomials. A frequent use of these polynomials involves the interpolation of tabulated data.
- However, in many applications, **an explicit representation of the polynomial is not needed**, but only the values of the polynomial at specified points.
- In this situation, the function underlying the data might be unknown so the explicit form of the error cannot be used to assure the accuracy of the interpolation.
- **Neville's Method** provides an ***adaptive mechanism*** for the evaluation of accurate interpolating values.

Definition 3.31. (Interpolating polynomial at $x_{m_1}, x_{m_2}, \cdots, x_{m_k}$): Let f be defined at $x_0, x_1, \cdots, x_n$, and suppose that $m_1, m_2, \cdots, m_k$ are k distinct integers with $0 \leq m_i \leq n$ for each i. The polynomial that agrees with at the points $x_{m_1}, x_{m_2}, \cdots, x_{m_k}$ is denoted by $\boldsymbol{P_{m_1, m_2, \cdots, m_k}}$.

Example 3.32. Suppose that $x_0 = 1$, $x_1 = 2$, $x_2 = 3$, $x_3 = 4$, $x_4 = 6$ and $f(x) = e^x$. Determine the interpolating polynomial $P_{1,2,4}(x)$ and use this polynomial to approximate $f(5)$.

Solution. It can be the Lagrange polynomial that agrees with $f(x)$ at $x_1 = 2$, $x_2 = 3$, $x_4 = 6$:

$$P_{1,2,4}(x) = \frac{(x-3)(x-6)}{(2-3)(2-6)}e^2 + \frac{(x-2)(x-6)}{(3-2)(3-6)}e^3 + \frac{(x-2)(x-3)}{(6-2)(6-3)}e^6.$$

Thus

$$P_{1,2,4}(5) = -\frac{1}{2}e^2 + e^3 + \frac{1}{2}e^6 \approx 218.1054057.$$

On the other hand, $f(5) = e^5 \approx 148.4131591$.

Theorem 3.33. *Let f be defined at $(n+1)$ distinct points, $x_0, x_1, \cdots, x_n$. Then for each $0 \le i < j \le n$,*

$$P_{i,i+1,\cdots,j}(x) = \frac{(x - x_i)P_{i+1,i+2,\cdots,j}(x) - (x - x_j)P_{i,i+1,\cdots,j-1}(x)}{x_j - x_i}, \tag{3.60}$$

which is the polynomial interpolating f at $x_i, x_{i+1}, \cdots, x_j$.

Note: The above theorem implies that the interpolating polynomial can be generated recursively. For example,

$$\begin{aligned} P_{0,1}(x) &= \frac{(x - x_0)P_1(x) - (x - x_1)P_0(x)}{x_1 - x_0} \\ P_{1,2}(x) &= \frac{(x - x_1)P_2(x) - (x - x_2)P_1(x)}{x_2 - x_1} \\ P_{0,1,2}(x) &= \frac{(x - x_0)P_{1,2}(x) - (x - x_2)P_{0,1}(x)}{x_2 - x_0} \end{aligned} \tag{3.61}$$

and so on. They are generated in the manner shown in the following table, where each row is completed before the succeeding rows are begun.

x_0	$y_0 = P_0$			
x_1	$y_1 = P_1$	$P_{0,1}$		
x_2	$y_2 = P_2$	$P_{1,2}$	$P_{0,1,2}$	
x_3	$y_3 = P_3$	$P_{2,3}$	$P_{1,2,3}$	$P_{0,1,2,3}$

(3.62)

For simplicity in computation, we may try to avoid multiple subscripts by defining the new variable

$$Q_{i,j} = P_{i-j,i-j+1,\cdots,i}$$

Then the above table can be expressed as

x_0	$P_0 = Q_{0,0}$			
x_1	$P_1 = Q_{1,0}$	$P_{0,1} = Q_{1,1}$		
x_2	$P_2 = Q_{2,0}$	$P_{1,2} = Q_{2,1}$	$P_{0,1,2} = Q_{2,2}$	
x_3	$P_3 = Q_{3,0}$	$P_{2,3} = Q_{3,1}$	$P_{1,2,3} = Q_{3,2}$	$P_{0,1,2,3} = Q_{3,3}$

(3.63)

Example 3.34. Let $x_0 = 2.0,\ x_1 = 2.2,\ x_2 = 2.3,\ x_3 = 1.9,\ x_4 = 2.15$. Use Neville's method to approximate $f(2.1) = \ln(2.1)$ in a four-digit accuracy.

Solution.

Maple-code

```
with(Student[NumericalAnalysis]):
x0 := 2.0:
x1 := 2.2:
x2 := 2.3:
x3 := 1.9:
x4 := 2.15:
xy := [[x0,ln(x0)], [x1,ln(x1)], [x2,ln(x2)], [x3,ln(x3)], [x4,ln(x4)]]:
P := PolynomialInterpolation(xy, method = neville):

Q := NevilleTable(P, 2.1)
  [[0.6931471806, 0,           0,           0,           0          ],
   [0.7884573604, 0.7408022680, 0,           0,           0          ],
   [0.8329091229, 0.7440056025, 0.7418700461, 0,           0          ],
   [0.6418538862, 0.7373815030, 0.7417975693, 0.7419425227, 0          ],
   [0.7654678421, 0.7407450500, 0.7418662324, 0.7419348958, 0.7419374382]]
```

Note that

$$\begin{aligned} |Q_{3,3} - Q_{2,2}| &= |0.7419425227 - 0.7418700461| = 0.0000724766 \\ |Q_{4,4} - Q_{3,3}| &= |0.7419374382 - 0.7419425227| = 0.0000050845 \end{aligned}$$

Thus $Q_{3,3} = 0.7419425227$ is already in a four-digit accuracy.
Check: The real value is $\ln(2.1) = 0.7419373447$. The absolute error: $|\ln(2.1) - Q_{3,3}| = 0.0000051780$. □

Pseudocode 3.35.

Input: the nodes $x_0, x_1, \cdots, x_n$; the evaluation point x; the tolerance ε; and values $y_0, y_1, \cdots, y_n$ in the 1st column of $Q \in \mathbb{R}^{(n+1)\times(n+1)}$

Output: Q

Step 1: For $i = 1, 2, \cdots, n$

For $j = 1, 2, \cdots, i$

$$Q_{i,j} = \frac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}}$$

if $(|Q_{i,i} - Q_{i-1,i-1}| < \varepsilon)$ $\{i_0 = i;$ break; $\}$

Step 2: Return $(Q,\ i_0)$

Example **3.36.** Neville's method is used to approximate $f(0.3)$, giving the following table.

$x_0 = 0$	$Q_{0,0} = 1$			
$x_1 = 0.25$	$Q_{1,0} = 2$	$Q_{1,1} = 2.2$		
$x_2 = 0.5$	$\mathbf{Q_{2,0}}$	$Q_{2,1}$	$Q_{2,2}$	
$x_3 = 0.75$	$Q_{3,0} = 5$	$Q_{3,1}$	$Q_{3,2} = 2.12$	$Q_{3,3} = 2.168$

(3.64)

Determine $Q_{2,0} = f(x_2)$.

Solution.

Ans: $Q_{2,2} = 2.2;\ \ Q_{2,1} = 2.2;\ \ Q_{2,0} = 3$

3.4. Hermite Interpolation

The **Hermite interpolation** refers to the interpolation of a function and *some of its derivatives* at a set of nodes. When a distinction is being made between this type of interpolation and its simpler type (in which no derivatives are interpolated), the latter is often called **Lagrange interpolation**.

Key Idea **3.37. (Basic Concepts of Hermite Interpolation)**:

- For example, we require a polynomial of least degree that interpolates a function f and its derivative f' at two distinct points, say x_0 and x_1.
- Then the polynomial p sought will satisfy these four conditions:

$$p(x_i) = f(x_i), \;\; p'(x_i) = f'(x_i); \quad i = 0, 1. \tag{3.65}$$

- Since there are four conditions, it seems reasonable to look for a solution in $\mathbb{P}_3$, the space of all polynomials of degree at most 3. Rather than writing $p(x)$ in terms of $1, x, x^2, x^3$, let us write it as

$$p(x) = a + b(x - x_0) + c(x - x_0)^2 + d(x - x_0)^2(x - x_1), \tag{3.66}$$

because this will simplify the work. This leads to

$$p'(x) = b + 2c(x - x_0) + 2d(x - x_0)(x - x_1) + d(x - x_0)^2. \tag{3.67}$$

- The four conditions on p, in (3.65), can now be written in the form

$$\begin{aligned} f(x_0) &= a \\ f'(x_0) &= b \\ f(x_1) &= a + bh + ch^2 \qquad (h = x_1 - x_0) \\ f'(x_1) &= b + 2ch + dh^2 \end{aligned} \tag{3.68}$$

Thus, the coefficients a, b, c, d can be obtained easily.

Theorem 3.38. ***(Hermite Interpolation Theorem)****: If $f \in C^1[a,b]$ and $x_0, x_1, \cdots, x_n \in [a,b]$ are distinct, then the unique polynomial of least degree agreeing with f and f' at the $(n+1)$ points is the* ***Hermite polynomial*** *of degree at most $(2n+1)$ given by*

$$H_{2n+1}(x) = \sum_{i=0}^{n} f(x_i) H_{n,i}(x) + \sum_{i=0}^{n} f(x_i) \widehat{H}_{n,i}(x), \tag{3.69}$$

where

$$\begin{aligned} H_{n,i}(x) &= [1 - 2(x - x_i) L'_{n,i}(x_i)] L^2_{n,i}(x), \\ \widehat{H}_{n,i}(x) &= (x - x_i) L^2_{n,i}(x). \end{aligned}$$

Here $L_{n,i}(x)$ is the ith Lagrange polynomial of degree n. Moreover, if $f \in C^{2n+2}[a,b]$, then

$$f(x) - H_{2n+1}(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \prod_{i=0}^{n} (x - x_i)^2. \tag{3.70}$$

Construction of Hermite Polynomials using Divided Differences

Recall: The polynomial P_n that interpolates f at $x_0, x_1, \cdots, x_n$ is given

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \cdots, x_k](x - x_0) \cdots (x - x_{k-1}). \tag{3.71}$$

Strategy 3.39. (Construction of Hermite Polynomials):

- Define a new sequence by $z_0, z_1, \cdots, z_{2n+1}$ by

$$z_{2i} = z_{2i+1} = x_i, \quad i = 0, 1, \cdots, n. \tag{3.72}$$

- Then the ***Newton form of the Hermite polynomial*** is given by

$$H_{2n+1}(x) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, z_1, \cdots, z_k](x - z_0) \cdots (x - z_{k-1}), \tag{3.73}$$

with

$$f[z_{2i}, z_{2i+1}] = f[x_i, x_i] = \frac{f[x_i] - f[x_i]}{x_i - x_i} \quad \text{replaced by } f'(x_i). \tag{3.74}$$

Note: For each $i = 0, 1, \cdots, n$,

$$\lim_{x \to x_i} \frac{f(x) - f(x_i)}{x - x_i} = f'(x_i). \tag{3.75}$$

The extended Newton divided difference table: Consider the Hermite polynomial that interpolates f and f' at three points, x_0, x_1, x_2.

$\mathbf{z}$	$\mathbf{f(z)}$	DD1	Higher DDs
$z_0 = x_0$	$f[z_0] = f(x_0)$		
$z_1 = x_0$	$f[z_1] = f(x_0)$	$f[z_0, z_1] = \mathbf{f'(x_0)}$	
$z_2 = x_1$	$f[z_2] = f(x_1)$	$f[z_1, z_2] = \dfrac{f[z_2] - f[z_1]}{z_2 - z_1}$	
$z_3 = x_1$	$f[z_3] = f(x_1)$	$f[z_2, z_3] = \mathbf{f'(x_1)}$	as usual
$z_4 = x_2$	$f[z_4] = f(x_2)$	$f[z_3, z_4] = \dfrac{f[z_4] - f[z_3]}{z_4 - z_3}$	
$z_5 = x_2$	$f[z_5] = f(x_2)$	$f[z_4, z_5] = \mathbf{f'(x_2)}$	

(3.76)

Each **zero-over-zero** is replaced by its corresponding derivative value, while other divided differences are obtained as usual.

Example 3.40. Use the extended Newton divided difference method to obtain a cubic polynomial that takes these values:

x	$f(x)$	$f'(x)$
0	2	-9
1	-4	4

Ans: $H_3(x) = 2 - 9x + 3x^2 + 7x^2(x-1)$.

Example 3.41. (Continuation): Find a quartic polynomial p_4 that takes values given in the preceding example and, in addition, satisfies $p_4(2) = 44$.

Ans: $p_4(x) = H_3(x) + 5x^2(x-1)^2$.

3.5. Spline Interpolation

3.5.1. Runge's phenomenon

Recall: (**Weierstrass approximation theorem**): Suppose $f \in C[a,b]$. Then, for each $\varepsilon > 0$, there exists a polynomial $P(x)$ such that

$$|f(x) - P(x)| < \varepsilon, \text{ for all } x \in [a,b]. \tag{3.77}$$

Interpolation at equidistant points is a natural and common approach to construct approximating polynomials. **Runge's phenomenon** demonstrates, however, that interpolation can easily result in divergent approximations.

Example 3.42. (**Runge's phenomenon**): Consider the function

$$f(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1]. \tag{3.78}$$

Runge found that if this function was interpolated at equidistant points

$$x_i = -1 + i\,\frac{2}{n}, \quad i = 0, 1, \cdots, n,$$

the resulting interpolation p_n oscillated toward the end of the interval, i.e. close to -1 and 1. It can even be proven that the interpolation error tends toward infinity when the degree of the polynomial increases:

$$\lim_{n\to\infty} \left(\max_{-1\le x\le 1} |f(x) - p_n(x)| \right) = \infty. \tag{3.79}$$

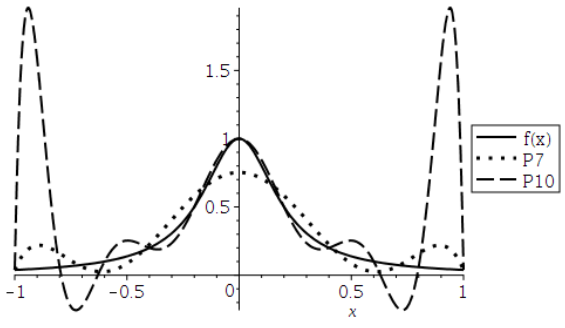


Figure 3.5: Runge's phenomenon.

Mitigation to the problem

- **Change of interpolation points**: e.g., Chebyshev nodes
- **Constrained minimization**: e.g., Hermite-like higher-order polynomial interpolation, whose first (or second) derivative has minimal norm.
- **Use of piecewise polynomials**: e.g., Spline interpolation

Definition 3.43. A **partition** of the interval $[a, b]$ is an ordered sequence $\{x_i\}_{i=0}^n$ such that

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b.$$

The numbers x_i are known as **knots** or **nodes**.

Definition 3.44. A function S is a **spline of degree** k on $[a, b]$ if

1) The domain of S is $[a, b]$.
2) There exits a partition $\{x_i\}_{i=0}^n$ of $[a, b]$ such that on each subinterval $[x_{i-1}, x_i]$, $S \in \mathbb{P}_k$.
3) $S, S', \cdots, S^{(k-1)}$ are continuous on (a, b).

3.5.2. Linear splines

A **linear spline** is a continuous function which is linear on each subinterval. Thus it is defined entirely by its values at the nodes. That is, given

x	x_0	x_1	$\cdots$	x_n
y	y_0	y_1	$\cdots$	y_n

the linear polynomial on each subinterval is defined as

$$L_i(x) = y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}), \quad x \in [x_{i-1}, x_i]. \tag{3.80}$$

Example 3.45. Find the linear spline for

x	0.0	0.2	0.5	0.8	1.0
y	1.3	3.0	2.0	2.1	2.5

Solution.

The linear spline can be easily computed as

$$L(x) = \begin{cases} 1.3 + 8.5x, & x < 0.2 \\ \frac{11}{3} - \frac{10x}{3}, & x < 0.5 \\ \frac{13}{6} + \frac{x}{3}, & x < 0.8 \\ 0.5 + 2.0x, & \text{otherwise} \end{cases} \tag{3.81}$$

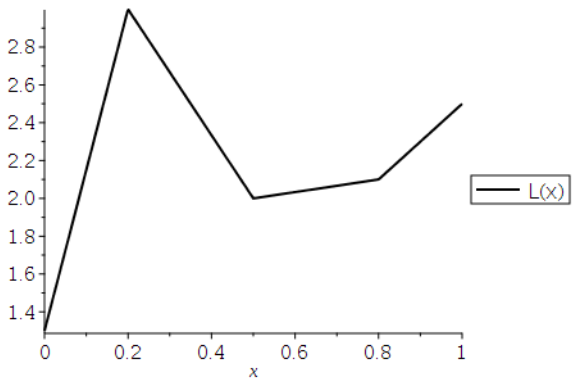


Figure 3.6: Linear spline.

First-Degree Spline Accuracy

Theorem 3.46. *To find the error bound, we will consider the error on a single subinterval of the partition, and apply a little calculus. Let* $p(x)$ *be the linear polynomial interpolating* $f(x)$ *at the endpoints of* $[x_{i-1}, x_i]$. *Then,*

$$f(x) - p(x) = \frac{f''(\xi)}{2!}(x - x_{i-1})(x - x_i), \tag{3.82}$$

for some $\xi \in (x_{i-1}, x_i)$. *Thus*

$$|f(x) - p(x)| \leq \frac{M_2}{8} \max_{1 \leq i \leq n} (x_i - x_{i-1})^2, \quad x \in [a, b], \tag{3.83}$$

where

$$M_2 = \max_{x \in (a,b)} |f''(x)|.$$

3.5.3. Quadratic (Second Degree) Splines

Remark 3.47. A **quadratic spline** is a piecewise quadratic function, of which the derivative is continuous on (a, b).

- Typically, a quadratic spline Q is defined by its piecewise polynomials: Let $Q_i = Q\big|_{[x_{i-1},x_i]}$. Then

$$Q_i(x) = a_i x^2 + b_i x + c_i, \quad x \in [x_{i-1}, x_i], \quad i = 1, 2, \cdots, n. \tag{3.84}$$

  Thus there are **$3n$ parameters** to define $Q(x)$.

- For each of the n **subintervals**, the data (x_i, y_i), $i = 1, 2, \cdots, n$, gives two equations regarding $Q_i(x)$:

$$Q_i(x_{i-1}) = y_{i-1} \quad \text{and} \quad Q_i(x_i) = y_i, \quad i = 1, 2, \cdots, n. \tag{3.85}$$

  This is $2n$ equations. The continuity condition on Q' gives a single equation for each of the $(n-1)$ **internal nodes**:

$$Q_i'(x_i) = Q_{i+1}'(x_i), \quad i = 1, 2, \cdots, n-1. \tag{3.86}$$

  This **totals $(3n-1)$ equations**, but $3n$ unknowns.

- Thus **an additional user-chosen condition** is required, e.g.,

$$Q'(a) = f'(a), \quad Q'(a) = 0, \quad \text{or} \quad Q''(a) = 0. \tag{3.87}$$

  Alternatively, the additional condition can be given at $x = b$.

Algorithm 3.48. (Construction of quadratic splines):

(0) Define

$$z_i = Q'(x_i), \quad i = 0, 1, \cdots, n; \tag{3.88}$$

suppose that the additional condition is given by specifying z_0.

(1) Because $Q_i' = Q'\big|_{[x_{i-1}, x_i]}$ is a *linear function* satisfying

$$Q_i'(x_{i-1}) = z_{i-1} \quad \text{and} \quad Q_i'(x_i) = z_i, \quad (\textit{continuity of } Q') \tag{3.89}$$

we have

$$Q_i'(x) = z_{i-1} + \frac{z_i - z_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}), \quad x \in [x_{i-1}, x_i]. \tag{3.90}$$

(2) By integrating it and using $Q_i(x_{i-1}) = y_{i-1}$ (*left edge value*)

$$Q_i(x) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x - x_{i-1})^2 + z_{i-1}(x - x_{i-1}) + y_{i-1}. \tag{3.91}$$

(3) In order to determine z_i, $1 \le i \le n$, we use the above at x_i (*right edge value*):

$$y_i = Q_i(x_i) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x_i - x_{i-1})^2 + z_{i-1}(x_i - x_{i-1}) + y_{i-1}, \tag{3.92}$$

which implies

$$\begin{aligned} y_i - y_{i-1} &= \frac{1}{2}(z_i - z_{i-1})(x_i - x_{i-1}) + z_{i-1}(x_i - x_{i-1}) \\ &= (x_i - x_{i-1})\frac{(z_i + z_{i-1})}{2}. \end{aligned}$$

Thus we have

$$z_i = 2\frac{y_i - y_{i-1}}{x_i - x_{i-1}} - z_{i-1}, \quad i = 1, 2, \cdots, n. \tag{3.93}$$

Note:

a. You should first decide z_i using (3.93) and then finalize Q_i from (3.91).

b. When z_n is specified, Equation (3.93) can be replaced by

$$z_{i-1} = 2\frac{y_i - y_{i-1}}{x_i - x_{i-1}} - z_i, \quad i = n, n-1, \cdots, 1. \tag{3.94}$$

Example 3.49. Find the quadratic spline for the same dataset used in Example 3.45, p. 116:

x	0.0	0.2	0.5	0.8	1.0
y	1.3	3.0	2.0	2.1	2.5

Solution. $z_i = Q_i'(x_i)$ are computed as

```
z[0]=8.5
z[1]=8.5
z[2]=-15.1667
z[3]=15.8333
z[4]=-11.8333
```

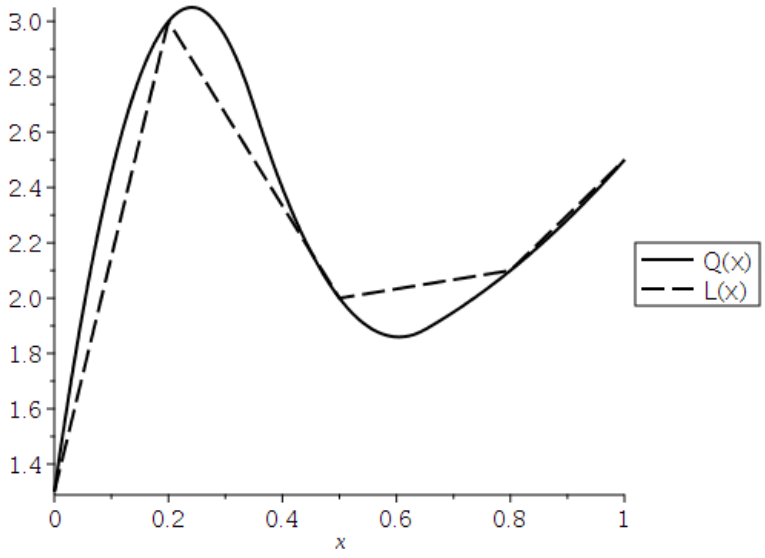


Figure 3.7: The graph of $Q(x)$ is superposed over the graph of the linear spline $L(x)$.

3.5.4. Cubic splines

Recall: **(Definition 3.44)**: A function S is a **cubic spline** on $[a, b]$ if

1) The domain of S is $[a, b]$.
2) $S \in \mathbb{P}_3$ on each subinterval $[x_{i-1}, x_i]$.
3) S, S', S'' are continuous on (a, b).

Remark 3.50. By definition, a **cubic spline** is a continuous piecewise cubic polynomial whose first and second derivatives are continuous.

- On each subinterval $[x_{i-1}, x_i]$, $1 \leq i \leq n$, we have to determine coefficients of a cubic polynomial of the form

$$S_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i, \quad i = 1, 2, \cdots, n. \tag{3.95}$$

  Thus there are $4n$ **unknowns** to define $S(x)$.

- On the other hand, equations we can get are

left an right values of S_i :	$2n$
continuity of S' :	$n-1$
continuity of S'' :	$n-1$

(3.96)

  Thus there are $(4n-2)$ **equations**.

- **Two degrees of freedom remain**, and there have been various ways of choosing them to advantage.

Algorithm 3.51. (Construction of cubic splines):

(0) Similarly as for quadratic splines, we define

$$z_i = S''(x_i), \quad i = 0, 1, \cdots, n. \tag{3.97}$$

(1) Because $S_i'' = S''\big|_{[x_{i-1},x_i]}$ is a *linear function* satisfying

$$S_i''(x_{i-1}) = z_{i-1} \quad \text{and} \quad S_i''(x_i) = z_i, \quad (\textit{continuity of } S'') \tag{3.98}$$

and therefore is given by the straight line between z_{i-1} and z_i:

$$S_i''(x) = \frac{z_{i-1}(x_i - x)}{h_i} + \frac{z_i(x - x_{i-1})}{h_i}, \quad h_i = x_i - x_{i-1}, \quad x \in [x_{i-1}, x_i]. \tag{3.99}$$

(2) If (3.99) is integrated twice, the result reads

$$S_i(x) = \frac{z_{i-1}(x_i - x)^3}{6h_i} + \frac{z_i(x - x_{i-1})^3}{6h_i} + C(x - x_{i-1}) + D(x_i - x). \tag{3.100}$$

In order to determine C and D, we use $S_i(x_{i-1}) = y_{i-1}$ and $S_i(x_i) = y_i$ (*left and right edge values*):

$$S_i(x_{i-1}) = \frac{z_{i-1}}{6}h_i^2 + Dh_i = y_{i-1}, \quad S_i(x_i) = \frac{z_i}{6}h_i^2 + Ch_i = y_i. \tag{3.101}$$

Thus (3.100) becomes

$$\begin{aligned} S_i(x) \;=\; & \frac{z_{i-1}(x_i - x)^3}{6h_i} + \frac{z_i(x - x_{i-1})^3}{6h_i} \\ & + \Big(\frac{y_i}{h_i} - \frac{1}{6}z_i h_i\Big)(x - x_{i-1}) + \Big(\frac{y_{i-1}}{h_i} - \frac{1}{6}z_{i-1}h_i\Big)(x_i - x). \end{aligned} \tag{3.102}$$

(3) The values $z_1, z_2, \cdots, z_{n-1}$ can be determined from the *continuity of* S':

$$\begin{aligned} S_i'(x) \;=\; & -\frac{z_{i-1}(x_i - x)^2}{2h_i} + \frac{z_i(x - x_{i-1})^2}{2h_i} \\ & + \Big(\frac{y_i}{h_i} - \frac{1}{6}z_i h_i\Big) - \Big(\frac{y_{i-1}}{h_i} - \frac{1}{6}z_{i-1}h_i\Big). \end{aligned} \tag{3.103}$$

Construction of cubic splines (continue):

Then substitution of $x = x_i$ and simplification lead to

$$S_i'(x_i) = \frac{h_i}{6} z_{i-1} + \frac{h_i}{3} z_i + \frac{y_i - y_{i-1}}{h_i}. \tag{3.104}$$

Analogously, after obtaining S_{i+1}', we have

$$S_{i+1}'(x_i) = -\frac{h_{i+1}}{3} z_{i-1} - \frac{h_{i+1}}{6} z_{i+1} + \frac{y_{i+1} - y_i}{h_{i+1}}. \tag{3.105}$$

When the right sides of (3.104) and (3.105) are set equal to each other, the result reads

$$h_i\, z_{i-1} + 2(h_i + h_{i+1})\, z_i + h_{i+1}\, z_{i+1} = \frac{6(y_{i+1} - y_i)}{h_{i+1}} - \frac{6(y_i - y_{i-1})}{h_i}, \tag{3.106}$$

for $i = 1, 2, \cdots, n-1$.

(4) **Two additional user-chosen conditions** are required to determine $(n+1)$ unknowns, $z_0, z_1, \cdots, z_n$. There are two popular approaches for the choice of the two additional conditions.

Natural Cubic Spline :	$z_0 = 0, \;\; z_n = 0$
Clamped Cubic Spline :	$S'(a) = f'(a), \;\; S'(b) = f'(b)$

Natural Cubic Splines: Let $z_0 = z_n = 0$. Then the system of linear equations in (3.106) can be written as

$$A \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} b_2 - b_1 \\ b_3 - b_2 \\ \vdots \\ b_n - b_{n-1} \end{bmatrix}, \tag{3.107}$$

where

$$A = \begin{bmatrix} 2(h_1+h_2) & h_2 & & & \\ h_2 & 2(h_2+h_3) & h_3 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & h_{n-1} & 2(h_{n-1}+h_n) \end{bmatrix} \quad \text{and} \quad b_i = \frac{6}{h_i}(y_i - y_{i-1}).$$

Clamped Cubic Splines: Let $f'(a)$ and $f'(b)$ be prescribed. Then the two extra conditions read

$$S'(a) = f'(a), \quad S'(b) = f'(b). \tag{3.108}$$

Since $a = x_0$ and $b = x_n$, utilizing Equation (3.103), the conditions read

$$\begin{aligned} 2h_1 z_0 + h_1 z_1 &= \frac{6}{h_1}(y_1 - y_0) - 6f'(x_0) \\ h_n z_{n-1} + 2h_n z_n &= 6f'(x_n) - \frac{6}{h_n}(y_n - y_{n-1}) \end{aligned} \tag{3.109}$$

Equation (3.106) and the above two equations clearly make $(n+1)$ conditions for $(n+1)$ unknowns, $z_0, z_1, \cdots, z_n$. It is a good exercise to compose an algebraic system for the computation of clamped cubic splines.

Example 3.52. Find the natural cubic spline for the same dataset

x	0.0	0.2	0.5	0.8	1.0
y	1.3	3.0	2.0	2.1	2.5

Solution.

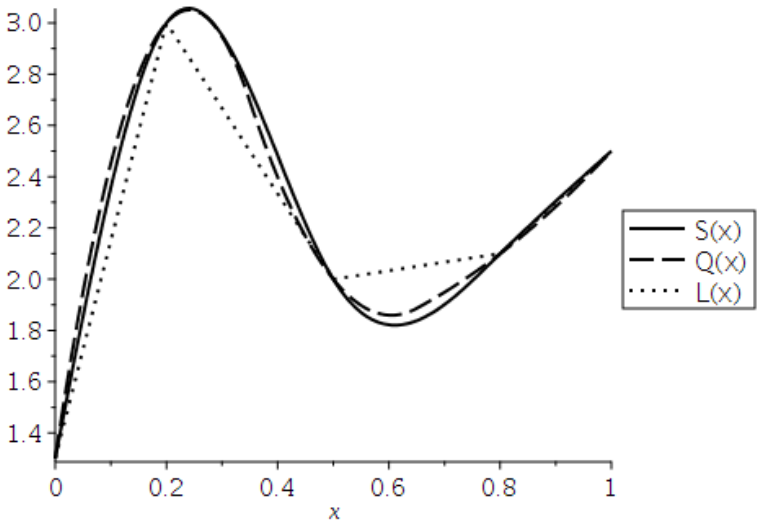


Figure 3.8: The graph of $S(x)$ is superposed over the graphs of the quadratic spline $Q(x)$ and the linear spline $L(x)$.

Example 3.53. Find the natural cubic spline that interpolates the data

x	0	1	3
y	4	2	7

Solution.

Maple-code

```
with(CurveFitting):
xy := [[0, 4], [1, 2], [3, 7]]:
n := 2:
L := x -> Spline(xy, x, degree = 1, endpoints = 'natural'):
Q := x -> Spline(xy, x, degree = 2, endpoints = 'notaknot'):
S := x -> Spline(xy, x, degree = 3, endpoints = 'natural'):
S(x)
          /          11     3  3            41   49     27  2   3  3\
  piecewise|x < 1, 4 - -- x + - x , otherwise, -- - -- x + -- x  - - x |
          \          4      4               8    8      8       8   /
```

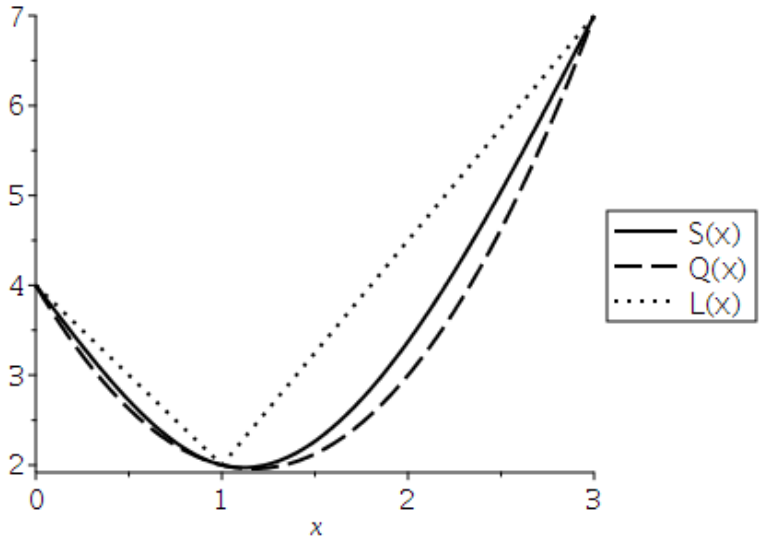


Figure 3.9: Splines on two subintervals.

Optimality Theorem for Natural Cubic Splines:

We now present a theorem to the effect that the natural cubic spline produces the smoothest interpolating function. The word *smooth* is given a technical meaning in the theorem.

Theorem **3.54.** *Let f'' be continuous in $[a, b]$ and $a = x_0 < x_1 < \cdots < x_n = b$. If S is the* ***natural cubic spline*** *interpolating f at the nodes x_i for $0 \leq i \leq n$, then*

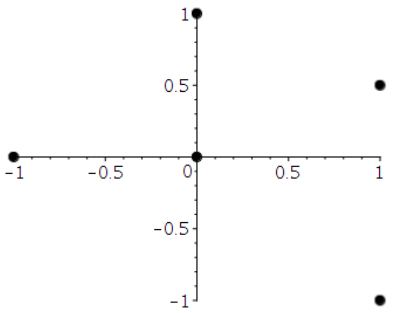
$$\int_a^b [S''(x)]^2 \, dx \leq \int_a^b [f''(x)]^2 \, dx. \tag{3.110}$$

3.6. Parametric Curves

Consider the data of the form:

```
xy := [[-1, 0], [0, 1], [1, 0.5], [0, 0], [1, -1]]
```

of which the point plot is given



- None of the interpolation methods we have learnt so far can be used to generate an interpolating curve for this data, because the curve cannot be expressed as a function of one coordinate variable to the other.
- In this section we will see how to represent general curves by using a parameter to express both the x- and y-coordinate variables.

Example 3.55. Construct a pair of interpolating polynomials, as a function of t, for the data:

i	0	1	2	3	4
t	0	0.25	0.5	0.75	1
x	-1	0	1	0	1
y	0	1	0.5	0	-1

Solution.

Maple-code

```
with(CurveFitting):
unassign('t'):
tx := [[0, -1], [0.25, 0], [0.5, 1], [0.75, 0], [1, 1]]:
ty := [[0, 0], [0.25, 1], [0.5, 0.5], [0.75, 0], [1, -1]]:
x := t -> PolynomialInterpolation(tx, t, form = Lagrange):
y := t -> PolynomialInterpolation(ty, t, form = Lagrange):
plot([x(t), y(t), t = 0..1], color = blue, thickness = 2)
```

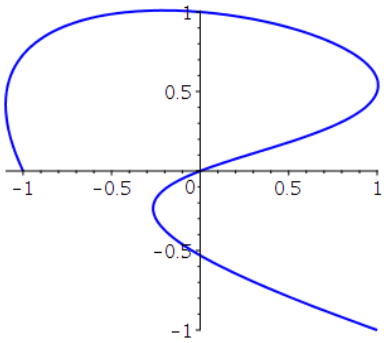


Figure 3.10

Remark 3.56. (Applications in Computer Graphics):

- **Required**: Rapid generation of smooth curves that can be quickly and easily modified.
- **Preferred**: Change of one portion of a curve should have little or no effect on other portions of the curve.

$\Rightarrow$ **The choice of curve** is a form of the **piecewise cubic Hermite polynomial**.

Example 3.57. For data $\{(x_i, f(x_i), f'(x_i)\}$, $i = 0, 1, \cdots, n$, **the piecewise cubic Hermite polynomial** can be generated independently in each portion $[x_{i-1}, x_i]$. Why?

Solution.

Piecewise cubic Hermite polynomial for General Curve Fitting

Algorithm 3.58. Let us focus on the first portion of the **piecewise cubic Hermite polynomial** interpolating between

$$(x_0, y_0) \quad \text{and} \quad (x_1, y_1).$$

- For the first portion, the given data are

$$\begin{aligned} &x(0) = x_0, \;\; y(0) = y_0, \;\; \frac{dy}{dx}(t=0); \\ &x(1) = x_1, \;\; y(1) = y_1, \;\; \frac{dy}{dx}(t=1); \end{aligned} \tag{3.111}$$

  - Only six conditions are specified, while the cubic polynomials $x(t)$ and $y(t)$ each have four parameters, for a total of eight.
  - The natural form for determining $x(t)$ and $y(t)$ requires

$$\begin{aligned} &x(0), \;\; x(1), \;\; x'(0), \;\; x'(1); \\ &y(0), \;\; y(1), \;\; y'(0), \;\; y'(1); \end{aligned} \tag{3.112}$$

  - $\boldsymbol{dy/dx = y'(t)/x'(t)}$ provides flexibility for the construction of parametric curves.

- The slopes at the endpoints can be expressed using the so-called **guidepoints** which are to be chosen from the desired tangent line:

$$\begin{array}{|l|} \hline (x_0 + \alpha_0, y_0 + \beta_0): \;\; \text{guidepoint for } (x_0, y_0) \\ (x_1 - \alpha_1, y_1 - \beta_1): \;\; \text{guidepoint for } (x_1, y_1) \\ \hline \end{array} \tag{3.113}$$

  Thus

$$\begin{aligned} \frac{dy}{dx}(t=0) &= \frac{y'(0)}{x'(0)} = \frac{\beta_0}{\alpha_0} = \frac{(y_0 + \beta_0) - y_0}{(x_0 + \alpha_0) - x_0} \\ \frac{dy}{dx}(t=1) &= \frac{y'(1)}{x'(1)} = \frac{\beta_1}{\alpha_1} = \frac{y_1 - (y_1 - \beta_1)}{x_1 - (x_1 - \alpha_1)} \end{aligned} \tag{3.114}$$

- Therefore, we may specify

$$x'(0) = \alpha_0, \;\; y'(0) = \beta_0; \qquad x'(1) = \alpha_1, \;\; y'(1) = \beta_1. \tag{3.115}$$

Formula 3.59. (The cubic Hermite polynomial $(x(t), y(t))$ on $[0,1]$):

- *The unique cubic Hermite polynomial $x(t)$ satisfying*

$$x(0) = x_0, \;\; x'(0) = \alpha_0; \qquad x(1) = x_1, \;\; x'(1) = \alpha_1$$

can be constructed as

$$\begin{aligned} x(t) \;\; &= \;\; [2(x_0 - x_1) + (\alpha_0 + \alpha_1)]\, t^3 + [3(x_1 - x_0) - (2\alpha_0 + \alpha_1)]\, t^2 \\ &\quad +\alpha_0\, t + x_0. \end{aligned} \tag{3.116}$$

- *Similarly, the unique cubic Hermite polynomial $y(t)$ satisfying*

$$y(0) = y_0, \;\; y'(0) = \beta_0; \qquad y(1) = y_1, \;\; y'(1) = \beta_1$$

can be constructed as

$$\begin{aligned} y(t) \;\; &= \;\; [2(y_0 - y_1) + (\beta_0 + \beta_1)]\, t^3 + [3(y_1 - y_0) - (2\beta_0 + \beta_1)]\, t^2 \\ &\quad +\beta_0\, t + y_0. \end{aligned} \tag{3.117}$$

Example 3.60. Determine the parametric curve when

$$(x_0, y_0) = (0,0), \;\; \frac{dy}{dx}(t=0) = 1; \qquad (x_1, y_1) = (1,0), \;\; \frac{dy}{dx}(t=1) = -1.$$

Solution.

- Let $\alpha_0 = 1, \;\; \beta_0 = 1$ and $\alpha_1 = 1, \;\; \beta_1 = -1$.
  - The cubic Hermite polynomial $x(t)$ satisfying

```
x0 := 0:  a0 := 1:  x1 := 1:  a1 := 1:
```

    is

```
x:=t->(2*(x0-x1)+a0+a1)*t^3+(3*(x1-x0)-a1-2*a0)*t^2+a0*t+x0
```

    $\Rightarrow$ `x(t) = t`
  - The cubic Hermite polynomial $y(t)$ satisfying

```
y0 := 0:  b0 := 1:  y1 := 0:  b1 := -1:
```

    is

```
y:=t->(2*(y0-y1)+b0+b1)*t^3+(3*(y1-y0)-b1-2*b0)*t^2+b0*t+y0
```

    $\Rightarrow$ `y(t) = -t^2 + t`

- `H1 := plot([x(t),y(t),t=0..1], coordinateview=[0..1, 0..1], thickness=2, linestye=solid)`

- **Let** $\alpha_0 = 0.5,\ \beta_0 = 0.5$ **and** $\alpha_1 = 0.5,\ \beta_1 = -0.5$.

  - ```
    a0 := 0.5:  b0 := 0.5:  a1 := 0.5:  b1 := -0.5:
    x:=t->(2*(x0-x1)+a0+a1)*t^3+(3*(x1-x0)-a1-2*a0)*t^2+a0*t+x0
    ⇒ x(t) = -1.0*t^3 + 1.5*t^2 + 0.5*t
    y:=t->(2*(y0-y1)+b0+b1)*t^3+(3*(y1-y0)-b1-2*b0)*t^2+b0*t+y0
    ⇒ y(t) = -0.5*t^2 + 0.5*t
    ```
  - `H2 := plot([x(t),y(t),t=0..1], coordinateview=[0..1, 0..1], thickness=2, linestye=dash)`

- ```
  Tan :=plot([t,-t+1],t =0..1, thickness=[2,2],
        linestyle=dot, color = blue)
  display(H1, H2, Tan)
  ```

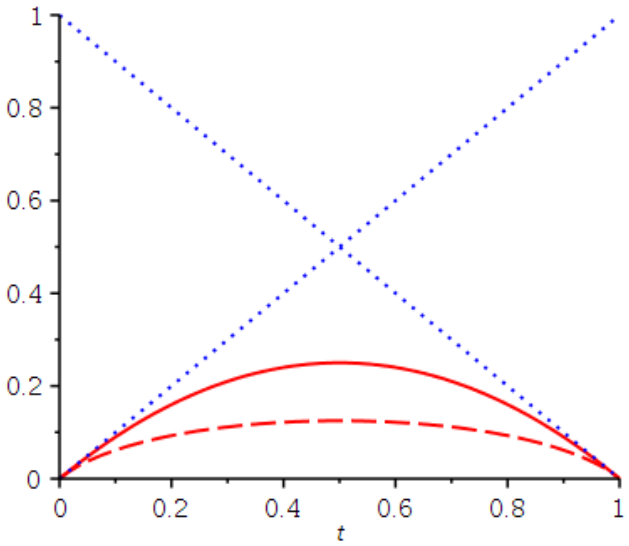


Figure 3.11: The parametric curves: $H_1(t)$ and $H_2(t)$.

Exercises for Chapter 3

3.1. C For the given functions $f(x)$, let $x_0 = 0$, $x_1 = 0.5$, $x_2 = 1$. Construct interpolation polynomials of degree at most one and at most two to approximate $f(0.4)$, and find the absolute error.

(a) $f(x) = \cos x$
(b) $f(x) = \ln(1 + x)$

3.2. Use the **Polynomial Interpolation Error Theorem** to find an error bound for the approximations in Problem 1 above.

3.3. The polynomial $p(x) = 1 - x + x(x + 1) - 2x(x + 1)(x - 1)$ interpolates the first four points in the table:

x	-1	0	1	2	3
y	2	1	2	-7	10

By adding one additional term to p, find a polynomial that interpolates the whole table. (Do not try to find the polynomial from the scratch.)

Ans: By adding another term, $p_4(x) == 1-x+x\,(x+1)-2x\,(x+1)\,(x-1)+cx\,(x+1)\,(x-1)\,(x-2)$. $p_4(3) = 10$ gives $c = 2$.

3.4. Determine the Newton interpolating polynomial for the data:

x	4	2	0	3
y	63	11	7	28

3.5. Neville's method is used to approximate $f(0.4)$, giving the following table.

$x_0 = 0$	$Q_{0,0}$		
$x_1 = 0.5$	$Q_{1,0} = 1.5$	$Q_{1,1} = 1.4$	
$x_2 = 0.8$	$Q_{2,0}$	$Q_{2,1}$	$Q_{2,2} = 1.2$

Fill out the whole table.

Ans: Q[2,:]=[3., 1., 1.2]

3.6. Use the extended Newton divided difference method to obtain a quintic polynomial that takes these values:

x	$f(x)$	$f'(x)$
0	2	-9
1	-4	4
2	44	
3	2	

3.7. Find a **natural cubic spline** for the data.

x	-1	0	1
$f(x)$	5	7	9

(Do not use computer programming for this problem.)

Ans: $S(x) = 2x + 7$.

3.8. C Consider the data

x	0	1	3
$f(x)$	4	2	7

with $f'(0) = -1/4$ and $f'(3) = 5/2$. (The points are used in Example 3.53, p.124.)

(a) Find the **quadratic spline** that interpolates the data (with $z_0 = f'(0)$).
(b) Find the **clamped cubic spline** that interpolates the data.
(c) Plot the splines and display them superposed.

3.9. Construct the **piecewise cubic Hermite interpolating polynomial** for

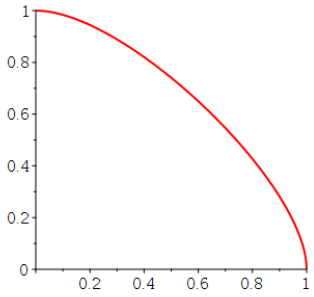
x	$f(x)$	$f'(x)$
0	2	-9
1	-4	4
2	4	12

Ans: $H_1(x) = 2 - 9x + 3x^2 + 7x^2(x-1)$ and $H_2(x) = -4 + 4(x-1) + 4(x-1)^2$.

3.10. C Let C be the unit circle of radius 1: $x^2 + y^2 = 1$. Find a piecewise cubic parametric curve that interpolates the circle at $(1,0), (0,1), (-1,0), (1,0)$. Try to make the parametric curve as circular as possible.

Hint: For the first portion, you may set

```
x0 := 1: x1 := 0: a0 := 0: a1 := -1:
x := t->(2*x0-2*x1+a0+a1)*t^3+(3*x1-3*x0-a1-2*a0)*t^2+a0*t+x0:
x(t)
                              3      2
                             t  - 2 t  + 1
y0 := 0: y1 := 1: b0 := 1: b1 := 0:
y := t->(2*y0-2*y1+b0+b1)*t^3+(3*y1-3*y0-b1-2*b0)*t^2+b0*t+y0:
y(t)
                               3    2
                              -t  + t  + t
plot([x(t),y(t),t=0..1], coordinateview = [0..1, 0..1], thickness = 2)
```



Now, (1) you can make it better, (2) you should find parametric curves for the other two portions, and (3) combine them for a piece.

The 15% Rule: Turning Curiosity into Financial Freedom

The 15% Rule

- Use 85% of your effort to become excellent at what you do.
- Use 15% to discover what else you could become.

Invest 15% of your discretionary/optional time in something outside your field: One day a week–or one or two hours a day:

- Put your phone away.
- Stop scrolling.
- Pick a field you know little about.
- Study it as if you were going to become the CEO of a company in that field.

Not for a grade. Not for your résumé.
Study it because someday it may become an opportunity.

Why Think Like a CEO?

Don't just ask:

How does renewable energy work?

Ask:

Where is the money made?

What problems remain unsolved?

Who are the customers?

What technologies are changing the industry?

Where could mathematics or computing create value?

If I were the CEO, what would I do?

That changes learning from

$$\begin{array}{c} \text{Information} \longrightarrow \text{Knowledge} \\ \downarrow \\ \text{Information} \longrightarrow \text{Knowledge} \longrightarrow \text{Opportunity} \end{array}$$

Three hundred years ago, **land** was wealth.
Later, the **industrialists** came to possess wealth.
Today, wealth lies in **information**.

Your Learning Speed Is Trainable

Let T_n be the time required to understand the n-th field. Then

$$T_1 \approx \textbf{1–2 years}$$

- However, as you keep learning new fields, you start to see **common structures**:

  Technology → Market → Cost → Risk → Competition → Opportunity

- So, conceptually: $T_{k+1} < T_k$
- With practice, you will eventually reach a point where you can grasp the essentials of a new subject **in just one or two months**:

$$T_n \approx \textbf{1–2 months, for a large } n$$

I call it the **learning compound interest**. It's not just money that compounds; the ability to learn compounds as well.

Here are some **topic examples** for the 15% Rule:

- Renewable Energy, AI for Healthcare
- Robotics, Cybersecurity
- Agriculture Technology, Stock Market

Every business or industrial R&D project focuses on **optimizing five key variables: scale, shape, direction, interface, and flow.**

Beyond Equations: Building True

Core Lessons from Robert Kiyosaki's ***Rich Dad Poor Dad***

- Your career earns money. Your financial system buys freedom.
- Don't measure wealth by salary. Measure it by freedom.

High Income $\neq$ Financial Freedom

The Poor Dad Mindset: The Lifestyle Trap

Imagine your salary grows:

$\$60K \rightarrow \$80K \rightarrow \$100K \rightarrow \$130K$

But your spending grows with it:

$\$50K \rightarrow \$70K \rightarrow \ \$90K \rightarrow \$120K.$

Your career improved enormously.
Your freedom did not.

A raise should increase your assets before it increases your lifestyle.

The Rich Dad Idea: Assets First

Robert Kiyosaki's useful simplification:

Assets put money into your pocket.

Liabilities take money out of your pocket.

Financial Thinking for Scientists

- Your Greatest Asset is Time
- Avoid the Three Delays:

  "I'll start when I make more money."
  "I'll invest after I buy everything I need."
  "I'll learn about finance later."

  $$\boxed{\text{Earn} \rightarrow \text{Save/Invest} \rightarrow \text{Spend}}$$

- Build a Machine, Not a Willpower Test

  Good scientists automate repetitive processes.
  Do the same with money.
  When income arrives: automatically
  save $\rightarrow$ invest $\rightarrow$ live on the remainder
  Don't make the same financial decision twelve times a year.
  Design the system once.
  This is essentially an algorithm for personal finance.

The poor and the middle class work for money
The rich have money work for them

- If you work for money, you give the power to your employer.
- If money works for you, you keep the power and control it.

One Equation to Remember

I might finish with a deliberately informal equation:

$$\text{Freedom} \approx \frac{\text{Assets + Skills + Time}}{\text{Lifestyle Demand}}$$

Invest time to get skills. (*The 15% Rule*)
Earn with your skills.
Build assets with your income.
Buy back your time.

CHAPTER 4

Numerical Differentiation and Integration

In this chapter:

Topics	**Applications/Properties**
Numerical Differentiation	$f(x) \approx P_n(x)$ locally $\Rightarrow f'(x) \approx P_n'(x)$
Three/five-point rules Richardson extrapolation	Combination of low-order differences, to get higher-order accuracy
Numerical Integration	$f(x) \approx P_n(x)$ piecewisely $\Rightarrow \int_a^b f(x) \approx \int_a^b P_n(x)$
Trapezoid rule Simpson's rule Simpson's Three-Eights rule Romberg integration	Newton-Cotes formulas
Gaussian Quadrature	Method of undetermined coefficients & orthogonal polynomials
Legendre polynomials	

Contents of Chapter 4

4.1. Numerical Differentiation 140
4.2. Richardson Extrapolation 145
4.3. Numerical Integration 151
4.4. Romberg Integration 161
4.5. Gaussian Quadrature 166
Exercises for Chapter 4 176

4.1. Numerical Differentiation

Note: The derivative of f at x_0 is defined as

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}. \tag{4.1}$$

This formula gives an obvious way to generate an approximation of $f'(x_0)$:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}. \tag{4.2}$$

Formula 4.1. (Two-Point Difference Formulas): *Let $x_1 = x_0 + h$ and $P_{0,1}$ be the first **Lagrange polynomial** interpolating f on $[x_0, x_1]$. Then*

$$\begin{aligned} f(x) &= P_{0,1}(x) + \frac{(x - x_0)(x - x_1)}{2!} f''(\xi) \\ &= \frac{x - x_1}{-h} f(x_0) + \frac{x - x_0}{h} f(x_1) + \frac{(x - x_0)(x - x_1)}{2!} f''(\xi). \end{aligned} \tag{4.3}$$

Differentiating it, we obtain

$$f'(x) = \frac{f(x_1) - f(x_0)}{h} + \frac{2x - x_0 - x_1}{2} f''(\xi) + \frac{(x - x_0)(x - x_1)}{2!} \frac{d}{dx} f''(\xi). \tag{4.4}$$

Thus

$$\begin{aligned} f'(x_0) &= \frac{f(x_1) - f(x_0)}{h} - \frac{h}{2} f''(\xi(x_0)) \\ f'(x_1) &= \frac{f(x_1) - f(x_0)}{h} + \frac{h}{2} f''(\xi(x_1)) \end{aligned} \tag{4.5}$$

Definition 4.2. For $h > 0$,

$$\begin{aligned} f'(x_i) &\approx D_x^+ f(x_i) = \frac{f(x_i + h) - f(x_i)}{h}, \quad \textbf{(forward-difference)} \\ f'(x_i) &\approx D_x^- f(x_i) = \frac{f(x_i) - f(x_i - h)}{h}. \quad \textbf{(backward-difference)} \end{aligned} \tag{4.6}$$

Example 4.3. Use the forward-difference formula to approximate $f(x) = x^3$ at $x_0 = 1$ using $h = 0.1, 0.05, 0.025$.

Solution. Note that $f'(1) = 3$.

Maple-code

```
f := x -> x^3: x0 := 1:

h := 0.1:
(f(x0 + h) - f(x0))/h
                              3.310000000
h := 0.05:
(f(x0 + h) - f(x0))/h
                              3.152500000
h := 0.025:
(f(x0 + h) - f(x0))/h
                              3.075625000
```

The error becomes half, as h halves?

Formula 4.4. (In general): *Let $\{x_0, x_1, \cdots, x_n\}$ be $(n+1)$ distinct points in some interval I and $f \in C^{n+1}(I)$. Then the Interpolation Error Theorem reads*

$$f(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0}^{n} (x - x_k). \tag{4.7}$$

Its derivative gives

$$\begin{aligned} f'(x) \;=\; & \sum_{k=0}^{n} f(x_k) L'_{n,k}(x) + \frac{d}{dx}\Big(\frac{f^{(n+1)}(\xi)}{(n+1)!}\Big) \prod_{k=0}^{n} (x - x_k) \\ & + \frac{f^{(n+1)}(\xi)}{(n+1)!} \frac{d}{dx}\Big(\prod_{k=0}^{n} (x - x_k)\Big). \end{aligned} \tag{4.8}$$

Hence,

$$f'(x_i) \;=\; \sum_{k=0}^{n} f(x_k) L'_{n,k}(x_i) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{k=0, k\neq i}^{n} (x_i - x_k). \tag{4.9}$$

Definition 4.5. An $(n+1)$**-point difference formula** to approximate $f'(x_i)$ is

$$f'(x_i) \approx \sum_{k=0}^{n} f(x_k) L'_{n,k}(x_i) \tag{4.10}$$

Formula 4.6. (Three-Point Difference Formulas $(n = 2)$**)**: *For convenience, let*

$$x0, \;\; x_1 = x_0 + h, \;\; x_2 = x_0 + 2h, \quad h > 0.$$

Recall

$$L_{2,0}(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}, \quad L_{2,1}(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)},$$
$$L_{2,2}(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}.$$

Thus, the ***three-point endpoint formulas*** *and the* ***three-point midpoint formula*** *read*

$$\begin{aligned}
f'(x_0) &= f(x_0)L'_{2,0}(x_0) + f(x_1)L'_{2,1}(x_0) + f(x_2)L'_{2,2}(x_0) + \frac{f^{(3)}(\xi)}{3!} \prod_{k=0,k\neq 0}^{2} (x_0 - x_k) \\
&= \frac{\mathbf{-3f(x_0) + 4f(x_1) - f(x_2)}}{\mathbf{2h}} + \frac{h^2}{3} f^{(3)}(\xi_0), \\
f'(x_1) &= \frac{\mathbf{f(x_2) - f(x_0)}}{\mathbf{2h}} - \frac{h^2}{6} f^{(3)}(\xi_1), \\
f'(x_2) &= \frac{\mathbf{f(x_0) - 4f(x_1) + 3f(x_2)}}{\mathbf{2h}} + \frac{h^2}{3} f^{(3)}(\xi_2).
\end{aligned} \tag{4.11}$$

Formula 4.7. (Five-Point Difference Formulas): *Let* $f_i = f(x_0 + i\,h)$, $h > 0$, $-\infty < i < \infty$.

$$\begin{aligned}
f'(x_0) &= \frac{f_{-2} - 8f_{-1} + 8f_1 - f_2}{12h} + \frac{h^4}{30} f^{(5)}(\xi), \\
f'(x_0) &= \frac{-25f_0 + 48f_1 - 36f_2 + 16f_3 - 3f_4}{12h} + \frac{h^4}{5} f^{(5)}(\xi).
\end{aligned} \tag{4.12}$$

Summary **4.8. (Numerical Differentiation, the $(n+1)$-point difference formulas)**:

1. $f(x) = P_n(x) + R_n(x), \quad P_n(x) \in \mathbb{P}_n$
2. $f'(x) = P_n'(x) + \mathcal{O}(h^n)$,
   $f''(x) = P_n''(x) + \mathcal{O}(h^{n-1})$, *and so on.*

Second-Derivative Midpoint Formula

Example **4.9.** We can see from the above summary that

when the **three-point** $(n = 2)$ or **five-point** $(n = 4)$ difference formula is applied for the approximation of f'', the accuracy reads $\mathcal{O}(h)$ or $\mathcal{O}(h^3)$.

Use the *Taylor series* to derive the (central-point) formulas

$$\begin{aligned} f''(x_0) &= \frac{f_{-1} - 2f_0 + f_1}{h^2} \\ &\quad -\frac{h^2}{12}f^{(4)}(x_0) - \frac{h^4}{360}f^{(6)}(x_0) - \frac{h^6}{20160}f^{(8)}(x_0) - \cdots \\ f''(x_0) &= \frac{-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2}{12h^2} \\ &\quad +\frac{h^4}{90}f^{(6)}(x_0) + \frac{h^6}{1008}f^{(8)}(x_0) + \cdots \end{aligned} \tag{4.13}$$

Solution. See Example 4.11, p. 145, for the derivation of $f''(x_0)$.

Note: In general, the higher-order accuracy in (4.13) can be achieved at the **central point** only.

Example 4.10. Use the second-derivative midpoint formula to approximate $f''(1)$ for $f(x) = x^5 - 3x^2$, using $h = 0.2, 0.1, 0.05$.

Solution.

Maple-code

```
f := x -> x^5 - 3*x^2:
x0 := 1:

eval(diff(f(x), x, x), x = x0)
                                  14
h := 0.2:
(f(x0 - h) - 2*f(x0) + f(x0 + h))/h^2
                             14.40000000
h := 0.1:
(f(x0 - h) - 2*f(x0) + f(x0 + h))/h^2
                             14.10000000
h := 0.05:
(f(x0 - h) - 2*f(x0) + f(x0 + h))/h^2
                             14.02500000
```

4.2. Richardson Extrapolation

Richardson extrapolation is used to generate **high-accuracy results**, using low-order formulas (on two or more different grids).

Example 4.11. Derive the **three-point midpoint formulas**:

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x-h)}{2h} - \left[\frac{h^2}{3!}f^{(3)}(x) + \frac{h^4}{5!}f^{(5)}(x) + \frac{h^6}{7!}f^{(7)}(x) + \cdots\right], \\ f''(x) &= \frac{f(x-h)-2f(x)+f(x+h)}{h^2} - \left[2\frac{h^2}{4!}f^{(4)}(x) + 2\frac{h^4}{6!}f^{(6)}(x) + \cdots\right]. \end{aligned} \tag{4.14}$$

Solution. It follows from the **Taylor's series formula** (1.14) that

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \frac{f^{(4)}(x)}{4!}h^4 + \cdots \\ f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2!}h^2 - \frac{f'''(x)}{3!}h^3 + \frac{f^{(4)}(x)}{4!}h^4 - \cdots \end{aligned} \tag{4.15}$$

Adding and subtracting these two equations, we have

$$\begin{aligned} f(x+h)+f(x-h) &= 2f(x) + 2\frac{\mathbf{f''(x)}}{2!}h^2 + 2\frac{f^{(4)}(x)}{4!}h^4 + \cdots \\ f(x+h)-f(x-h) &= 2\,\mathbf{f'(x)}h + 2\frac{f'''(x)}{3!}h^3 + 2\frac{f^{(5)}(x)}{5!}h^5 + \cdots \end{aligned} \tag{4.16}$$

Now, solve these equations for $f''(x)$ and $f'(x)$. □

Observation 4.12. The results in Example 4.11 can be written as

$$M = N(h) + K_2h^2 + K_4h^4 + K_6h^6 + \cdots, \tag{4.17}$$

where M is the desired (unknown) quantity, $N(h)$ is an approximation of M using the parameter h, and ***K_i* are independent of *h*.**

How can we take advantage of the observation?

Strategy 4.13. (Richardson extrapolation): Rewrite (4.17):

$$M = N(h) + \mathbf{K_2 h^2} + K_4 h^4 + K_6 h^6 + \cdots . \tag{4.18}$$

1. Let's write out (4.18) with h replaced by $h/2$:

$$M = N(h/2) + \underline{\mathbf{K_2 h^2/4}} + K_4 h^4/16 + K_6 h^6/64 + \cdots . \tag{4.19}$$

Then the leading term in the error series, $K_2 h^2$, can be eliminated as follows:

$$\begin{array}{rcrl} M &=& N(h) & +K_2 h^2 + K_4 h^4 + K_6 h^6 + \cdots \\ 4M &=& 4N(h/2) & +K_2 h^2 + K_4 h^4/4 + K_6 h^6/16 + \cdots \\ \hline 3M &=& 4N(h/2) - N(h) & -\frac{3}{4} K_4 h^4 - \frac{15}{16} K_6 h^6 - \cdots \end{array} \tag{4.20}$$

Thus we have

$$M = \frac{1}{3}[4N(h/2) - N(h)] - \frac{1}{4} K_4 h^4 - \frac{5}{16} K_6 h^6 - \cdots . \tag{4.21}$$

The above equation embodies the first step in *Richardson extrapolation*. It show that a simple combination of two second-order approximations, $N(h)$ and $N(h/2)$, furnishes an estimate of M with accuracy $\mathcal{O}(h^4)$.

2. For simplicity, we rewrite (4.21) as

$$M = N_2(h) - \frac{1}{4} K_4 h^4 - \frac{5}{16} K_6 h^6 - \cdots . \tag{4.22}$$

Then, similarly,

$$M = N_2(h/2) - \frac{1}{64} K_4 h^4 - \frac{5}{2^{10}} K_6 h^6 - \cdots . \tag{4.23}$$

Subtract (4.21) from 16 times (4.23) to produce a new $\mathcal{O}(h^6)$ formula:

$$M = \frac{1}{15}[16 N_2(h/2) - N_2(h)] + \frac{1}{64} K_6 h^6 + \cdots . \tag{4.24}$$

Algorithm 4.14. (Richardson extrapolation): The above idea can be applied recursively. The complete algorithm of Richardson extrapolation algorithm is formulated as:

1. Select a convenient h and compute

$$D(i,0) = N(h/2^i), \quad i = 0, 1, \cdots, n. \tag{4.25}$$

2. Compute additional quantities using the formula

for $i = 1, 2, \cdots, n$ do
  for $j = 1, 2, \cdots, i$ do

$$D(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot D(i, j-1) - D(i-1, j-1)\right] \tag{4.26}$$

  end do
end do

Note:

(a) One can prove that

$$D(i,j) = M + \mathcal{O}(h^{2(j+1)}). \tag{4.27}$$

(b) The second step in the algorithm can be rewritten for a **column-wise computation**:

for $j = 1, 2, \cdots, i$ do
  for $i = j, j+1, \cdots, n$ do

$$D(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot D(i, j-1) - D(i-1, j-1)\right] \tag{4.28}$$

  end do
end do

Example 4.15. Let $f(x) = \ln x$. Use the Richardson extrapolation to estimate $f'(1) = 1$ using $h = 0.2, 0.1, 0.05$.

Solution.

Maple-code

```
f := x -> ln(x):
h := 0.2:
D00 := (f(1 + h) - f(1 - h))/(2*h);
                              1.013662770
h := 0.1:
D10 := (f(1 + h) - f(1 - h))/(2*h);
                              1.003353478
h := 0.05:
D20 := (f(1 + h) - f(1 - h))/(2*h);
                              1.000834586

D11 := (4*D10 - D00)/3;
                              0.9999170470
D21 := (4*D20 - D10)/3;
                              0.9999949557
D22 := (16*D21 - D11)/15;
                              1.000000150
#Error Convergence:
abs(1 - D11);
                              0.0000829530
abs(1 - D21);
                              0.0000050443
#The Ratio:
abs(1 - D11)/abs(1 - D21);
                              16.44489820
```

h	$j = 0 : \mathcal{O}(h^2)$	$j = 1 : \mathcal{O}(h^4)$	$j = 2 : \mathcal{O}(h^6)$
0.2	$D_{0,0} = 1.013662770$		
0.1	$D_{1,0} = 1.003353478$	$D_{1,1} = 0.9999170470$	
0.05	$D_{2,0} = 1.000834586$	$D_{2,1} = 0.9999949557$	$D_{2,2} = 1.000000150$

Example 4.16. Let $f(x) = \ln x$, as in the previous example, Example 4.15. Produce a Richardson extrapolation table for the approximation of $f''(1) = -1$, using $h = 0.2, 0.1, 0.05$.

Solution.

Maple-code

```
f := x -> ln(x):
h := 0.2:
D00 := (f(1 - h) - 2*f(1) + f(1 + h))/h^2;
                              -1.020549862
h := 0.1:
D10 := (f(1 - h) - 2*f(1) + f(1 + h))/h^2;
                              -1.005033590
h := 0.05:
D20 := (f(1 - h) - 2*f(1) + f(1 + h))/h^2;
                              -1.001252088

D11 := (4*D10 - D00)/3;
                             -0.9998614997
D21 := (4*D20 - D10)/3;
                             -0.9999915873
D22 := (16*D21 - D11)/15;
                              -1.000000260
#Error Convergence:
abs(-1 - D11);
                              0.0001385003
abs(-1 - D21);
                              0.0000084127
#The Ratio:
abs(-1 - D11)/abs(-1 - D21);
                              16.46324010
```

h	$j = 0:\ \mathcal{O}(h^2)$	$j = 1:\ \mathcal{O}(h^4)$	$j = 2:\ \mathcal{O}(h^6)$
0.2	$D_{0,0} = -1.020549862$		
0.1	$D_{1,0} = -1.005033590$	$D_{1,1} = -0.9998614997$	
0.05	$D_{2,0} = -1.001252088$	$D_{2,1} = -0.9999915873$	$D_{2,2} = -1.000000260$

Example 4.17. In Example 4.9, we used the *Taylor series* to derive the formulas

$$\begin{aligned} f''(x_0) &= \frac{f_{-1} - 2f_0 + f_1}{h^2} \\ &\quad -\frac{h^2}{12}f^{(4)}(x_0) - \frac{h^4}{360}f^{(6)}(x_0) - \frac{h^6}{20160}f^{(8)}(x_0) - \cdots \\ f''(x_0) &= \frac{-f_{-2} + 16f_{-1} - 30f_0 + 16f_1 - f_2}{12h^2} \\ &\quad +\frac{h^4}{90}f^{(6)}(x_0) + \frac{h^6}{1008}f^{(8)}(x_0) + \cdots \end{aligned} \tag{4.29}$$

Let $N(h)$ be such that

$$f''(x_0) = N(h) - \frac{h^2}{12}f^{(4)}(x_0) - \frac{h^4}{360}f^{(6)}(x_0) - \frac{h^6}{20160}f^{(8)}(x_0) - \cdots \tag{4.30}$$

Then

$$N(2h) = \frac{f_{-2} - 2f_0 + f_2}{(2h)^2}, \tag{4.31}$$

and

$$f''(x_0) = N(2h) - \frac{(2h)^2}{12}f^{(4)}(x_0) - \frac{(2h)^4}{360}f^{(6)}(x_0) - \frac{(2h)^6}{20160}f^{(8)}(x_0) - \cdots \tag{4.32}$$

Claim 4.18. Four times of (4.30) minus (4.32), divided by 3, gives exactly the same formula as the second equation of (4.29). This implies that the Richardson extrapolation results in the numerical solution of a higher-order accuracy on the **fine grid** level. See Exercise 4.3.

4.3. Numerical Integration

Note: **Numerical integration** can be performed by

(1) approximating the function f by an nth-degree polynomial P_n, and

(2) integrating the polynomial over the prescribed interval.

What a simple task it is!

Let $\{x_0, x_1, \cdots, x_n\}$ be distinct points (nodes) in $[a, b]$. Then the *Lagrange interpolating polynomial* reads

$$P_n(x) = \sum_{i=0}^{n} f(x_i) L_{n,i}(x), \tag{4.33}$$

which interpolates the function f. Then, as just mentioned, we simply approximate

$$\int_a^b f(x)\,dx \approx \int_a^b P_n(x)\,dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_{n,i}(x)\,dx. \tag{4.34}$$

Definition 4.19. In this way, we obtain a formula which is a *weighted sum* of the function values:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} A_i\, f(x_i), \tag{4.35}$$

where

$$A_i = \int_a^b L_{n,i}(x)\,dx. \tag{4.36}$$

The formula of the form in (4.35) is called a **Newton-Cotes formula** when the nodes are equally spaced.

4.3.1. The Trapezoid Rule

The simplest case results if $n = 1$ and the nodes are $x_0 = a$ and $x_1 = b$. In this case,

$$f(x) = \sum_{i=0}^{1} f(x_i)L_{1,i}(x) + \frac{f''(\xi_x)}{2!}(x - x_0)(x - x_1), \tag{4.37}$$

and its integration reads

$$\int_{x_0}^{x_1} f(x)\,dx = \sum_{i=0}^{1} f(x_i) \int_{x_0}^{x_1} L_{1,i}(x)\,dx + \int_{x_0}^{x_1} \frac{f''(\xi_x)}{2!}(x - x_0)(x - x_1)\,dx. \tag{4.38}$$

Derivation **4.20.** *Terms in the right-side of (4.38) must be verified to get a formula and its error bound. Note that*

$$\begin{aligned} A_0 &= \int_{x_0}^{x_1} L_{1,0}(x)\,dx = \int_{x_0}^{x_1} \frac{x - x_1}{x_0 - x_1}dx = \frac{1}{2}(x_1 - x_0), \\ A_1 &= \int_{x_0}^{x_1} L_{1,1}(x)\,dx = \int_{x_0}^{x_1} \frac{x - x_0}{x_1 - x_0}dx = \frac{1}{2}(x_1 - x_0), \end{aligned} \tag{4.39}$$

and

$$\begin{aligned} \int_{x_0}^{x_1} \frac{f''(\xi_x)}{2!}(x - x_0)(x - x_1)\,dx &= \frac{f''(\xi)}{2!}\int_{x_0}^{x_1} (x - x_0)(x - x_1)\,dx \\ &= -\frac{f''(\xi)}{12}(x_1 - x_0)^3. \end{aligned} \tag{4.40}$$

(Here we could use the Weighted Mean Value Theorem on Integral because $(x - x_0)(x - x_1) \leq 0$ does not change the sign over $[x_0, x_1]$.)

Definition **4.21.** The corresponding quadrature formula is

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{h}{2}[f(x_0) + f(x_1)] - \frac{h^3}{12}f''(\xi), \quad \textbf{(Trapezoid)} \tag{4.41}$$

which is known as the **trapezoid rule**.

Graphical interpretation:

```
with(Student[Calculus1]):
f := x^3 + 2 + sin(2*Pi*x):
ApproximateInt(f, 0..1, output = animation, partition = 1,
    method = trapezoid, refinement = halve,
    boxoptions = [filled = [color=pink,transparency=0.5]]);
```

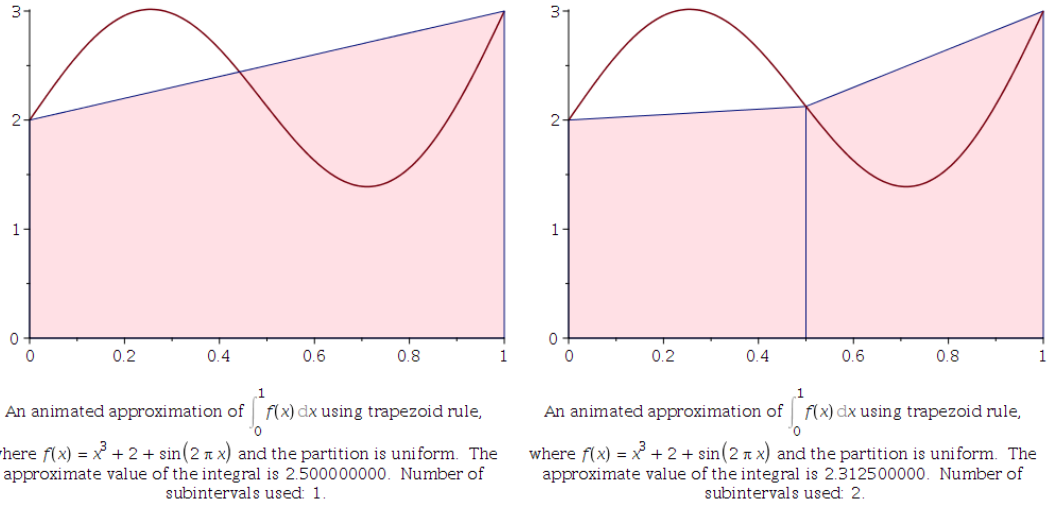


Figure 4.1: Trapazoid rule.

Composite Trapezoid Rule

Let the interval $[a, b]$ be partitioned as

$$a = x_0 < x_1 < \cdots < x_n = b.$$

Then the trapezoid rule can be applied to each subinterval. Here the nodes are not necessarily uniformly spaced. Thus, we obtain the **composite trapezoid rule** reads

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)\,dx \approx \sum_{i=1}^{n} \frac{h_i}{2}\left(f(x_{i-1}) + f(x_i)\right), \quad h_i = x_i - x_{i-1}. \tag{4.42}$$

With a uniform spacing:

$$x_i = a + i\,h, \quad h = \frac{b-a}{n},$$

the composite trapezoid rule takes the form

$$\int_a^b f(x)\,dx \approx h \cdot \left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right], \tag{4.43}$$

for which the **composite error** becomes

$$\sum_{i=1}^{n}\left(-\frac{h^3}{12}f''(\xi_i)\right) = -f''(\xi)\sum_{i=1}^{n}\frac{h^3}{12} = -f''(\xi)\frac{h^3}{12}\cdot n = -f''(\xi)\frac{(b-a)h^2}{12}, \tag{4.44}$$

where we have used $\left(h = \dfrac{b-a}{n} \Rightarrow n = \dfrac{b-a}{h}\right)$.

Example 4.22.

```
with(Student[Calculus1]):
f := x^3 + 2 + sin(2*Pi*x):
ApproximateInt(f, 0..1, output = animation, partition = 8,
    method = trapezoid, refinement = halve,
    boxoptions = [filled = [color=pink,transparency=0.5]]);
```

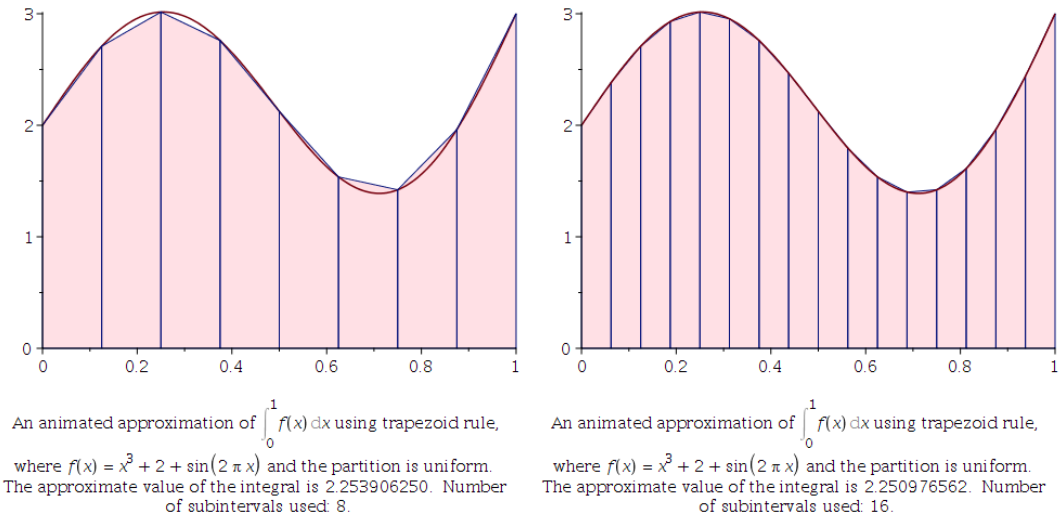


Figure 4.2: Composite trapazoid rule.

4.3.2. Simpson's Rule

Remark 4.23. Let $a = x_0 < x_1 < x_2 < \cdots < x_n = b$ be a partition of $[a, b]$.

- **The Trapezoid rule** uses **locally linear** interpolating polynomials, each of which is formulated on **one subinterval**.
- We may utilize **locally quadratic** interpolating polynomials, formulated on **each two subintervals** (as a group).
- Furthermore, **three and more subintervals** can be combined, for a higher-order interpolating polynomial.

Simpson's rule results from integrating the **second Lagrange polynomial** with three equally spaced nodes, combining **two subintervals**:

$$x_0, \quad x_1 = x_0 + h, \quad x_2 = x_0 + 2h,$$

for som $h > 0$.

Definition 4.24. Approximating f by the quadratic interpolating polynomial on $[x_0, x_2]$, the **elementary Simpson's rule** reads

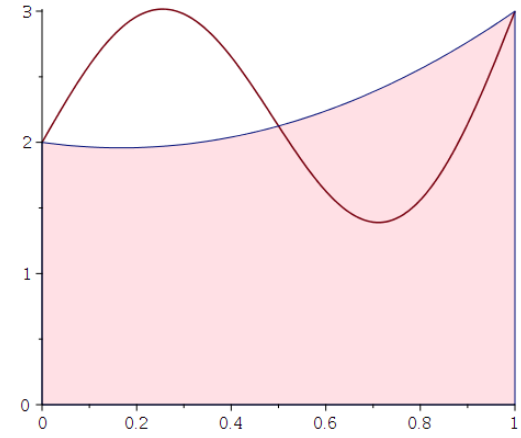
$$\int_{x_0}^{x_2} f(x)\,dx \approx \int_{x_0}^{x_2} \sum_{i=0}^{2} f(x_i) L_{2,i}(x) = \sum_{i=0}^{2} f(x_i) \int_{x_0}^{x_2} L_{2,i}(x), \tag{4.45}$$

which is reduced to

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{2h}{6}\big[f(x_0) + 4f(x_1) + f(x_2)\big]. \tag{4.46}$$

Graphical interpretation:

```
with(Student[Calculus1]):
f := x^3 + 2 + sin(2*Pi*x):
ApproximateInt(f, 0..1, output = animation, partition = 1,
    method = simpson, refinement = halve,
    boxoptions = [filled = [color=pink,transparency=0.5]]);
```



An animated approximation of $\int_0^1 f(x)\,dx$ using Simpson's rule, where $f(x) = x^3 + 2 + \sin(2\pi x)$ and the partition is uniform. The approximate value of the integral is 2.250000000. Number of subintervals used: 1.

Figure 4.3: The elementary Simpson's rule, which is **exact** for the given problem.

Remark 4.25. Error for the Simpsons Rule

- The error for the elementary Simpson's rule can be analyzed from

$$\int_{x_0}^{x_2} \frac{f'''(\xi)}{3!}(x-x_0)(x-x_1)(x-x_2)\,dx, \tag{4.47}$$

  which must be in $\mathcal{O}(h^4)$.
- However, by approximating the problem in another way, one can show **the error is in** $\mathcal{O}(h^5)$.

Error Analysis for the Elementary Simpson's Rule

- It follows from the *Taylor's Theorem* that for each $x \in [x_0, x_2]$, there is a number $\xi \in (x_0, x_2)$ such that

$$f(x) = f(x_1)+f'(x_1)(x-x_1)+\frac{f''(x_1)}{2!}(x-x_1)^2+\frac{f'''(x_1)}{3!}(x-x_1)^3+\frac{f^{(4)}(\xi)}{4!}(x-x_1)^4. \tag{4.48}$$

- By integrating the terms over $[x_0, x_2]$, we have

$$\begin{aligned}\int_{x_0}^{x_2} f(x)\,dx &= \Big|f(x_1)(x-x_1)+\frac{f'(x_1)}{2}(x-x_1)^2+\frac{f''(x_1)}{3!}(x-x_1)^3 \\ &\quad +\frac{f'''(x_1)}{4!}(x-x_1)^4\Big|_{x_0}^{x_2}+\int_{x_0}^{x_2}\frac{f^{(4)}(\xi)}{4!}(x-x_1)^4\,dx.\end{aligned} \tag{4.49}$$

  The last term can be easily computed by using the *Weighted Mean Value Theorem on Integral*:

$$\int_{x_0}^{x_2}\frac{f^{(4)}(\xi)}{4!}(x-x_1)^4\,dx = \frac{f^{(4)}(\xi_1)}{4!}\int_{x_0}^{x_2}(x-x_1)^4\,dx = \frac{f^{(4)}(\xi_1)}{60}h^5. \tag{4.50}$$

- Thus, (4.49) reads

$$\int_{x_0}^{x_2} f(x)\,dx = 2h\,f(x_1)+\frac{h^3}{3}\mathbf{f''(x_1)}+\frac{f^{(4)}(\xi_1)}{60}h^5. \tag{4.51}$$

- See (4.14), p.145, to recall that

$$f''(x_1) = \frac{f(x_0)-2f(x_1)+f(x_2)}{h^2}-\frac{h^2}{12}f^{(4)}(\xi_2). \tag{4.52}$$

- Plugging this to (4.51) reads

$$\begin{aligned}\int_{x_0}^{x_2} f(x)\,dx &= 2h\,f(x_1)+\frac{h^3}{3}\Big(\frac{f(x_0)-2f(x_1)+f(x_2)}{h^2}\Big) \\ &\quad -\frac{h^3}{3}\Big(\frac{h^2}{12}f^{(4)}(\xi_2)\Big)+\frac{f^{(4)}(\xi_1)}{60}h^5,\end{aligned} \tag{4.53}$$

  and therefore

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{2h}{6}\big[f(x_0)+4f(x_1)+f(x_2)\big]-\frac{h^5}{90}f^{(4)}(\xi_3). \tag{4.54}$$

Composite Simpson's Rule

A **composite Simpson's rule**, using an even number of subintervals, is often adopted. Let n be even, and set

$$x_i = a + i\,h, \quad h = \frac{b-a}{n}. \qquad (0 \le i \le n)$$

Then

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n/2} \int_{x_{2i-2}}^{x_{2i}} f(x)\,dx \approx \frac{2h}{6} \sum_{i=1}^{n/2} \big[f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\big]. \tag{4.55}$$

Example 4.26. Show that the error term for the composite Simpson's rule becomes

$$-\frac{(b-a)h^4}{180} f^{(4)}(\xi). \tag{4.56}$$

Solution.

4.3.3. Simpson's Three-Eights Rule

We have developed quadrature rules when the function f is approximated by piecewise Lagrange polynomials of degrees 1 and 2. Such integration formulas are called the **closed Newton-Cotes formulas**, and the idea can be extended for any degrees. The word *closed* is used, because the formulas include endpoints of the interval $[a, b]$ as nodes.

Theorem 4.27. *When* **three equal subintervals** *are combined, the resulting integration formula is called the* ***Simpson's three-eights rule****:*

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}\big[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\big] - \frac{3h^5}{80} f^{(4)}(\xi_0). \tag{4.57}$$

Example 4.28. Let n be a multiple of 3. For the nodes,

$$x_i = a + i\,h, \quad h = \frac{b-a}{n}, \qquad (0 \le i \le n)$$

derive the error term for the **composite Simpson's three-eights rule**.

Solution.

Ans: $-\dfrac{(b-a)h^4}{80} f^{(4)}(\xi)$

Note: When n is not even, you may approximate the integration by a combination of the Simpson's rule and the Simpson's three-eights rule. For example, let $n = 13$. Then you may apply the Simpson's rule for $[x_0, x_{10}]$ and the Simpson's three-eights rule for the last three subintervals $[x_{10}, x_{13}]$.

Self-study **4.29.** Consider

$$\int_0^2 \frac{x}{x^2+1} dx,$$

of which the true value is $\frac{\ln 5}{2} \approx 0.80471895621705018730$. Use 6 equally spaced points ($n = 5$) to approximate the integral using

(a) the **trapezoid rule**, and

(b) a **combination of the Simpson's rule and the Simpson's three-eights rule**.

Solution.

(a) 0.7895495781.

(so, the error = **0.0151693779**)

(b) $\int_0^{0.8} f(x)dx + \int_{0.8}^2 f(x)dx \approx 0.2474264468 + 0.5567293981 = 0.8041558449$.

(so, the error = **0.0005631111**)

4.4. Romberg Integration

Note: **Romberg integration** is to generate high-accuracy results, using multiple Trapezoid quadrature (obtained on refined partitions).

- In the previous section, we have found that the **Composite Trapezoid rule** has a truncation error of order $\mathcal{O}(h^2)$.
- Over n equal subintervals of $[a, b]$

$$x_k = a + k\,h, \quad h = (b-a)/n, \qquad (0 \le k \le n),$$

let $T(n)$ be the **Trapezoid quadrature**:

$$T(n) = h\Big[\frac{1}{2}f(a) + \sum_{k=1}^{n-1} f(x_k) + \frac{1}{2}f(b)\Big]. \tag{4.58}$$

Let us begin with an effective computation technique for $T(2n)$ as a refinement of $T(n)$.

4.4.1. Recursive Trapezoid Rule

Example 4.30. What is the explicit formula for $T(1), T(2), T(4)$, and $T(8)$ in the case in which the interval is $[0, 1]$?

Solution. Using Equation (4.58), we have

$$\begin{aligned}
T(1) &= 1 \cdot \Big[\frac{1}{2}f(0) + \frac{1}{2}f(1)\Big] \\
T(2) &= \frac{1}{2} \cdot \Big[\frac{1}{2}f(0) + f\Big(\frac{1}{2}\Big) + \frac{1}{2}f(1)\Big] \\
T(4) &= \frac{1}{4} \cdot \Big[\frac{1}{2}f(0) + f\Big(\frac{1}{4}\Big) + f\Big(\frac{1}{2}\Big) + f\Big(\frac{3}{4}\Big) + \frac{1}{2}f(1)\Big] \\
T(8) &= \frac{1}{8} \cdot \Big[\frac{1}{2}f(0) + f\Big(\frac{1}{8}\Big) + f\Big(\frac{1}{4}\Big) + f\Big(\frac{3}{8}\Big) + f\Big(\frac{1}{2}\Big) \\
&\qquad + f\Big(\frac{5}{8}\Big) + f\Big(\frac{3}{4}\Big) + f\Big(\frac{7}{8}\Big) + \frac{1}{2}f(1)\Big]
\end{aligned} \tag{4.59}$$

Remark 4.31. It is clear that if $T(2n)$ is to be computed, then we can take advantage of the work already done in the computation of $T(n)$. For example, from the preceding example, we see that

$$\begin{aligned} T(2) &= \frac{1}{2}T(1) + \frac{1}{2}\cdot\left[f\left(\frac{1}{2}\right)\right] \\ T(4) &= \frac{1}{2}T(2) + \frac{1}{4}\cdot\left[f\left(\frac{1}{4}\right) + f\left(\frac{3}{4}\right)\right] \\ T(8) &= \frac{1}{2}T(4) + \frac{1}{8}\cdot\left[f\left(\frac{1}{8}\right) + f\left(\frac{3}{8}\right) + f\left(\frac{5}{8}\right) + f\left(\frac{7}{8}\right)\right] \end{aligned} \tag{4.60}$$

With $h = (b-a)/(2n)$, the general formula pertaining to any interval $[a, b]$ is as follows:

$$T(2n) = \frac{1}{2}T(n) + h\left[f(a+h) + f(a+3h) + \cdots + f(a+(2n-1)h)\right], \tag{4.61}$$

or

$$T(2n) = \frac{1}{2}T(n) + h\Big(\sum_{k=1}^{n} f(x_{2k-1})\Big). \tag{4.62}$$

Now, if there are 2^i uniform subintervals, Equation (4.61) provides a **recursive Trapezoid rule**:

$$T(2^i) = \frac{1}{2}T(2^{i-1}) + h_i\Big(\sum_{k=1}^{2^{i-1}} f(a+(2k-1)h_i)\Big), \tag{4.63}$$

where

$$h_0 = b-a, \quad h_i = \frac{1}{2}h_{i-1}, \quad i \geq 1.$$

4.4.2. The Romberg Algorithm

By the Taylor series method, it can be shown that if $f \in C^{\infty}[a,b]$, **the Composite Trapezoid rule (4.58)** can also be written with an error term in the form

$$\int_a^b f(x)\,dx \;=\; T(n) + K_2h^2 + K_4h^4 + K_6h^6 + \cdots, \tag{4.64}$$

where K_i are constants independent of h. Since

$$\int_a^b f(x)\,dx \;=\; T(2n) + K_2h^2/4 + K_4h^4/16 + K_6h^6/64 + \cdots, \tag{4.65}$$

as for **Richardson extrapolation**, we have

$$\int_a^b f(x)\,dx \;=\; \frac{1}{3}\big[4T(2n) - T(n)\big] - \frac{3}{4}K_4h^4 - \frac{15}{16}K_6h^6 - \cdots. \tag{4.66}$$

Algorithm 4.32. **(Romberg algorithm)**: The above idea can be applied recursively. The complete algorithm of **Romberg integration** is formulated as:

1. The computation of $R(i,0)$ which is the trapezoid estimate with 2^i subintervals obtained using the formula (4.63):

$$\begin{aligned} R(0,0) &= \frac{b-a}{2}[f(a)+f(b)], \\ R(i,0) &= \frac{1}{2}R(i-1,0) + h_i\Big(\sum_{k=1}^{2^{i-1}} f(a+(2k-1)h_i)\Big). \end{aligned} \tag{4.67}$$

2. Then, evaluate higher-order approximations recursively using

for $i = 1, 2, \cdots, n$ do
  for $j = 1, 2, \cdots, i$ do

$$R(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot R(i,j-1) - R(i-1,j-1)\right] \tag{4.68}$$

  end do
end do

Example 4.33. Use the Composite Trapezoid rule to find approximations to $\int_0^{\pi} \sin x \, dx$, with $n = 1, 2, 4$, and 8. Then perform Romberg extrapolation on the results.

Solution.

Romberg-extrapolation

```
a := 0: b := Pi:
f := x -> sin(x):
n := 3:
R := Array(0..n, 0..n):

# Trapezoid estimates
#---------------------------------
R[0, 0] := (b - a)/2*(f(a) + f(b));
                                  0
for i to n do
    hi := (b-a)/2^i;
    R[i,0] := R[i-1,0]/2 +hi*add(f(a+(2*k-1)*hi), k=1..2^(i-1));
end do:

# Now, perform Romberg Extrapolation:
# ----------------------------------------------
for i to n do
    for j to i do
       R[i, j] := (4^j*R[i,j-1] - R[i-1, j-1])/(4^j-1);
    end do
end do
```

$j = 0 : \mathcal{O}(h^2)$	$j = 1 : \mathcal{O}(h^4)$	$j = 2 : \mathcal{O}(h^6)$	$j = 3 : \mathcal{O}(h^8)$
$R_{0,0} = 0$			
$R_{1,0} = 1.570796327$	$R_{1,1} = 2.094395103$		
$R_{2,0} = 1.896118898$	$R_{2,1} = 2.004559755$	$R_{2,2} = 1.998570731$	
$R_{3,0} = 1.974231602$	$R_{3,1} = 2.000269171$	$R_{3,2} = 1.999983131$	$R_{3,3} = 2.000005551$

(4.69)

Self-study **4.34.** The true value for the integral: $\int_0^{\pi} \sin x \, dx = 2$. Use the table in (4.69) to verify that the error is in $\mathcal{O}(h^4)$ for $j = 1$ and $\mathcal{O}(h^6)$ for $j = 2$.

Hint: For example, for $j = 1$, you should measure $|R_{1,1} - 2|/|R_{2,1} - 2|$ and $|R_{2,1} - 2|/|R_{3,1} - 2|$ and interpret them.

4.5. Gaussian Quadrature

Recall: **Newton-Cotes Formulas**

- In Section 4.3, we saw how to create quadrature formulas of the form

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} w_i\, f(x_i), \tag{4.70}$$

that are exact for polynomials of degree $\leq n$**,** which is the case if and only if

$$w_i = \int_a^b L_{n,i}(x)\,dx = \int_a^b \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\,dx. \tag{4.71}$$

- In those formulas, the choice of nodes $x_0, x_1, x_2, \cdots, x_n$ were made *a priori*. Once the nodes were fixed, the weights were determined uniquely from the requirement that Formula (4.70) must be an equality for $f \in \mathbb{P}_n$.

4.5.1. The Method of Undetermined Coefficients

Example 4.35. Find w_0, w_1, w_2 with which the following formula is exact for all polynomials of degree ≤ 2:

$$\int_0^1 f(x)\,dx \;\approx\; w_0\, f(0) + w_1\, f(1/2) + w_2\, f(1). \tag{4.72}$$

Solution. Formula (4.72) must be exact for some low-order polynomials. Consider trial functions $f(x) = 1,\ x,\ x^2$. Then, for each of them,

$$\begin{aligned}
1 &= \int_0^1 1\,dx &&= w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 1 &&= w_0 + w_1 + w_2 \\
\frac{1}{2} &= \int_0^1 x\,dx &&= w_0 \cdot 0 + w_1 \cdot \frac{1}{2} + w_2 \cdot 1 &&= \frac{1}{2}w_1 + w_2 \\
\frac{1}{3} &= \int_0^1 x^2\,dx &&= w_0 \cdot 0 + w_1 \cdot \left(\frac{1}{2}\right)^2 + w_2 \cdot 1 &&= \frac{1}{4}w_1 + w_2
\end{aligned} \tag{4.73}$$

The solution of this system of three simultaneous equations is

$$(w_0, w_1, w_2) = \left(\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\right). \tag{4.74}$$

Thus the formula can be written as

$$\int_0^1 f(x)\,dx \approx \frac{1}{6}\big[f(0) + 4f(1/2) + f(1)\big], \tag{4.75}$$

which will produce **exact** values of integrals for any quadratic polynomial, $f(x) = a_0 + a_1 x + a_2 x^2$. □

Note: It must be noticed that Formula (4.75) is the *elementary Simpson's rule* with $h = 1/2$.

Key Idea **4.36.** **Gaussian quadrature chooses the nodes in an optimal way**, rather than equally-spaced points.

- The nodes $x_1, x_2, \cdots, x_n$ in the interval $[a, b]$ and the weights $w_1, w_2, \cdots, w_n$ are chosen **to minimize the expected error** obtained in the approximation

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i\, f(x_i). \tag{4.76}$$

- To measure this accuracy, we assume that the best choice of these values produces the exact result for the largest class of polynomials, that is, the choice that gives the greatest degree of precision.
- The above formula gives $2n$ parameters to choose:

$$x_1, x_2, \cdots, x_n \text{ and } w_1, w_2, \cdots, w_n$$

Since the class of polynomials of degree at most $(2n-1)$ is $2n$-dimensional (containing $2n$ parameters), one may try to decide the parameters with which the quadrature formula is exact for all polynomials in $\mathbb{P}_{2n-1}$.

Example 4.37. Determine x_1, x_2 and w_1, w_2 so that the integration formula

$$\int_{-1}^{1} f(x)\,dx \approx w_1 f(x_1) + w_2 f(x_2) \tag{4.77}$$

gives the exact result whenever $f \in \mathbb{P}_3$.

Solution. As in the previous example, we may apply the *method of undetermined coefficients*. This time, use $f(x) = 1, x, x^2, x^3$ as trial functions.

Note: The **method of undetermined coefficients** is a classical method to determine the nodes and weights for formulas, but an alternative method can obtain them more easily. The alternative is related to **Legendre orthogonal polynomials**.

- The Chebyshev polynomials, defined in Definition 3.20, are also orthogonal polynomials; see page 97.
- Frequently-cited classical orthogonal polynomials are: *Jacobi polynomials*, *Laguerre polynomials*, *Chebyshev polynomials*, and *Legendre polynomials*.

4.5.2. Legendre Polynomials: Gauss and Gauss-Lobatto Integration

Definition 4.38. Let $\{P_0(x), P_1(x), \cdots, P_k(x), \cdots\}$ be a collection of polynomials with $P_k \in \mathbb{P}_k$. It is called **orthogonal polynomials** when it satisfies

$$\int_{-1}^{1} Q(x)P_k(x)\,dx = 0, \quad \forall\, Q(x) \in \mathbb{P}_{k-1}. \tag{4.78}$$

Such orthogonal polynomials can be formulated by a certain **three-term recurrence relation**.

Definition 4.39. The **Legendre polynomials** obey the three-term recurrence relation, known as *Bonnet's recursion formula*:

$$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x), \tag{4.79}$$

beginning with $P_0(x) = 1$, $P_1(x) = x$. A few first Legendre polynomials are

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= \frac{3}{2}(x^2 - \frac{1}{3}) \\ P_3(x) &= \frac{5}{2}(x^3 - \frac{3}{5}x) \\ P_4(x) &= \frac{35}{8}(x^4 - \frac{6}{7}x^2 + \frac{3}{35}) \\ P_5(x) &= \frac{63}{8}(x^5 - \frac{10}{9}x^3 + \frac{5}{21}x) \end{aligned} \tag{4.80}$$

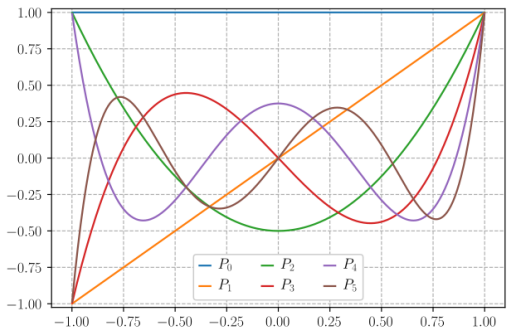


Figure 4.4: Legendre polynomials - Wikipedia

Theorem 4.40. *The Legendre polynomials satisfy*

$$\begin{aligned} &|P_k(x)| \le 1, \quad \forall\, x \in [-1,1], \\ &P_k(\pm 1) = (\pm 1)^k, \\ &\int_{-1}^{1} P_j(x) P_k(x)\, dx = 0, \quad j \ne k; \qquad \int_{-1}^{1} P_k(x)^2\, dx = \frac{1}{k+1/2}. \end{aligned} \tag{4.81}$$

Gauss Integration

Theorem 4.41. ***(Gauss integration)**: Suppose that $\{x_1, x_2, \cdots, x_n\}$ are the roots of the **nth Legendre polynomial** P_n and $\{w_1, w_2, \cdots, w_n\}$ are obtained by*

$$w_i = \int_{-1}^{1} \prod_{j=1, j\ne i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \left(= \int_{-1}^{1} L_{n-1,i}(x)\, dx \right) \tag{4.82}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i) \ \textbf{\textit{is exact}}, \ \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.83}$$

Note: Once the nodes are determined, the weights $\{w_1, w_2, \cdots, w_n\}$ can also be found by using the *method of undetermined coefficients*. That is, the weights are the solution of the linear system

$$\sum_{j=1}^{n} (x_j)^i w_j = \int_{-1}^{1} x^i\, dx, \quad i = 0, 1, \cdots, n-1. \tag{4.84}$$

Gauss-integration.mw

```
with(LinearAlgebra):
with(Statistics):
WeightSystem := proc(n, A, b, X)
    local i, j;
    for j to n do
        A[1, j] := 1;
        for i from 2 to n do
            A[i, j] := A[i - 1, j]*X[j];
        end do:
    end do:
    for i to n do
        b[i] := int(x^(i - 1), x = -1..1);
    end do:
end proc

nmax := 5:
for k to nmax do
    Legendre[k] := sort(orthopoly[P](k, x));
end do;
```

$$\text{Legendre}[1] := x$$

$$\text{Legendre}[2] := \frac{3}{2}x^2 - \frac{1}{2}$$

$$\text{Legendre}[3] := \frac{5}{2}x^3 - \frac{3}{2}x$$

$$\text{Legendre}[4] := \frac{35}{8}x^4 - \frac{15}{4}x^2 + \frac{3}{8}$$

$$\text{Legendre}[5] := \frac{63}{8}x^5 - \frac{35}{4}x^3 + \frac{15}{8}x$$

```
Node := Array(0..nmax):  Weight := Array(0..nmax):

for k to nmax do
    solve(Legendre[k] = 0, x):
    Node[k] := Sort(Vector([%])):
    n := Dimensions(Node[k]):
    A := Matrix(n, n):  b := Vector(n):
    WeightSystem(n, A, b, Node[k]):
    Weight[k] := A^(-1).b:
end do:
```

```
44
45  for k to nmax do
46      printf("          k=%d\n", k);
47      print(Nodek = evalf(Node[k]));
48      print(Weightk = evalf(Weight[k]));
49  end do;
50            k=1
51                                Nodek = [0.]
52                               Weightk = [2.]
53            k=2
54                              [   -0.5773502693     ]
55                     Nodek = [                      ]
56                              [    0.5773502693     ]
57
58                                         [1.]
59                               Weightk = [  ]
60                                         [1.]
61            k=3
62                              [   -0.7745966692     ]
63                              [                     ]
64                     Nodek = [         0.          ]
65                              [                     ]
66                              [    0.7745966692     ]
67
68                                       [0.5555555556]
69                                       [            ]
70                           Weightk = [0.8888888889]
71                                       [            ]
72                                       [0.5555555556]
73            k=4
74                              [   -0.8611363114     ]
75                              [                     ]
76                              [   -0.3399810437     ]
77                     Nodek = [                      ]
78                              [    0.3399810437     ]
79                              [                     ]
80                              [    0.8611363114     ]
81
82                                       [0.3478548456]
83                                       [            ]
84                                       [0.6521451560]
85                           Weightk = [            ]
86                                       [0.6521451563]
87                                       [            ]
88                                       [0.3478548450]
```

```
k=5
                  [   -0.9061798457     ]
                  [                     ]
                  [   -0.5384693100     ]
                  [                     ]
          Nodek = [         0.          ]
                  [                     ]
                  [    0.5384693100     ]
                  [                     ]
                  [    0.9061798457     ]

                        [0.2427962711]
                        [            ]
                        [0.472491159 ]
                        [            ]
             Weightk = [0.568220204 ]
                        [            ]
                        [0.478558682 ]
                        [            ]
                        [0.2369268937]
```

Remark 4.42. (Gaussian Quadrature on Arbitrary Intervals): An integral $\int_a^b f(x)\,dx$ over an interval $[a,b]$ can be transformed into an integral over $[-1,1]$ by using the *change of variables*:

$$T : [-1,1] \to [a,b], \quad x = T(t) = \frac{b-a}{2}t + \frac{a+b}{2}. \tag{4.85}$$

Using it, we have

$$\int_a^b f(x)\,dx = \int_{-1}^{1} f\Big(\frac{b-a}{2}t + \frac{a+b}{2}\Big)\frac{b-a}{2}\,dt. \tag{4.86}$$

Example 4.43. Find the Gaussian Quadrature for $\int_0^{\pi} \sin(x)\,dx$, with $n = 2, 3, 4$.

Solution. The transformation $T : [-1, 1] \to [0, \pi]$ reads

$$T(t) = \frac{\pi}{2}(t+1) = \frac{\pi}{2}t + \frac{\pi}{2} \quad (= x). \tag{4.87}$$

Using the **change of variables**, the integral is transformed as

$$\int_0^{\pi} \sin(x)\,dx = \int_{-1}^{1} \sin(T(t))\,\frac{\pi}{2}\,dt \tag{4.88}$$

Gaussian-Quadrature.mw

```
a := 0:  b := Pi:
f := x -> sin(x):
nmax := 4:

T := t -> (b - a)/2 *t + (a+b)/2:
T(t)
                          1        1
                          - Pi t + - Pi
                          2        2
trueI := int(f(x), x = a..b)
                                2

g := t -> f(T(t))*(b - a)/2:
GI := Vector(nmax):
for k from 2 to nmax do
    GI[k] := add(evalf(Weight[k][i]*g(Node[k][i])), i = 1..k);
end do:
for k from 2 to nmax do
  printf("  n=%d  GI=%g   error=%g\n",k,GI[k],abs(trueI-GI[k]));
end do

   n=2  GI=1.93582   error=0.0641804
   n=3  GI=2.00139   error=0.00138891
   n=4  GI=1.99998   error=1.5768e-05
```

Gauss-Lobatto Integration

Theorem 4.44. ***(Gauss-Lobatto integration)**: Let* $x_0 = -1$, $x_n = 1$, *and* $\{x_1, x_2, \cdots, x_{n-1}\}$ *are* **the roots of the first-derivative of the nth Legendre polynomial**, $P_n'(x)$. *Let* $\{w_0, w_1, w_2, \cdots, w_n\}$ *be obtained by*

$$w_i = \int_{-1}^{1} \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \left(= \int_{-1}^{1} L_{n,i}(x)\, dx \right) \tag{4.89}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=0}^{n} w_i\, f(x_i) \ \textit{is exact}, \ \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.90}$$

Recall: **Theorem 4.41 (Gauss integration)**: Suppose that $\{x_1, x_2, \cdots, x_n\}$ are **the roots of the nth Legendre polynomial** P_n and $\{w_1, w_2, \cdots, w_n\}$ are obtained by

$$w_i = \int_{-1}^{1} \prod_{j=1, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \left(= \int_{-1}^{1} L_{n-1,i}(x)\, dx \right) \tag{4.91}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i) \ \text{is exact}, \ \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.92}$$

Remark 4.45. Once the nodes are determined, $\{w_0, w_1, w_2, \cdots, w_n\}$ can also be found by using the *method of undetermined coefficients*, as for Gauss integration; the weights are the solution of the linear system

$$\sum_{j=0}^{n} (x_j)^i w_j = \int_{-1}^{1} x^i\, dx, \quad i = 0, 1, \cdots, n. \tag{4.93}$$

The Gauss-Lobatto integration is a **closed formula** for numerical integrations, which is **more popular in real-world applications** than **open formulas** such as the Gauss integration.

Self-study 4.46. Find the Gauss-Lobatto Quadrature for $\int_0^{\pi} \sin(x)\, dx$, with $n = 2, 3, 4$.

Exercises for Chapter 4

4.1. Use the most accurate three-point formulas to determine the missing entries.

x	$f(x)$	$f'(x)$	$f''(x)$
1.0	2.0000		6.00
1.2	1.7536		
1.4	1.9616		
1.6	2.8736		
1.8	4.7776		
2.0	8.0000		
2.2	12.9056		52.08

Hint: The central scheme is more accurate than one-sided schemes.

4.2. Use your results in the above table to approximate $f'(1.6)$ and $f''(1.6)$ with $\mathcal{O}(h^4)$-accuracy. Make a conclusion by comparing all results (obtained here and from Problem 1) with the exact values:

$$f'(1.6) = 6.784, \quad f''(1.6) = 24.72.$$

Hint: In order to get a 4th-order Richardson approximation, you should have a coarse grid approximation, using $f(1.2),\ f(1.6),\ f(2.0)$

Ans: $f''(1.6) \approx 24.72$

4.3. Verify Claim 4.18, p. 150: The Richardson extrapolation results in the numerical solution of a higher-order accuracy on the **fine grid** level.

4.4. Let a numerical process be described by

$$M = N(h) + K_1 h + K_2 h^2 + K_3 h^3 + \cdots \tag{4.94}$$

Explain how Richardson extrapolation will work in this case. Introduce a formula described as in (4.26), page 147.
Hint: You should ***first*** try to remove the first-order error term, $K_1 h$.

4.5. In order to approximate $\int_0^2 x \ln(x^2+1)\,dx$ with $h = 0.4$, use

(a) the Trapezoid rule, and
(b) Simpson's rules.

Hint: For (b), you may use the Simpson's rule for the first two subintervals and the Simpson's three-eighth rule for the remained three subintervals.

Ans: (a) 2.06735. (b) 2.02253.

4.6. A car laps a race track in 65 seconds. The speed of the car at each 5 second interval is determined by using a radar gun and is given from the beginning of the lap, in feet/second, by the entries in the following table:

Time	0	5	10	15	20	25	30	35	40	25	50	55	60	65
Speed	0	90	124	142	156	147	133	121	109	99	95	78	89	98

Estimate the length of the track, using the Simpson's rules.

Ans: 7190.7 ft

4.7. C Consider integrals

I. $\int_1^3 \frac{1}{x}\,dx$ II. $\int_0^{\pi} \cos^2 x\,dx$

(a) For each of the integrals, use the Romberg extrapolation to find $R[3,3]$.

(b) Determine the number of subintervals required when the Composite Trapezoid rule is used to find approximations within the same accuracy as $R[3,3]$.

Ans: (a) $R[3,3] = 1.0986$

4.8. C Find the Gaussian Quadrature for $\int_0^{\pi/2} \cos^2 x\,dx$, with $n = 2, 3, 4$.

Ans:

Table 4.1: Integration Resulta using Gaussian Quadrature

n	Quadrature	Absolute Error
2	0.785398	0.0
3	0.785398	0.2220×10^{-15}
4	0.785398	0.0

CHAPTER 5

Numerical Solution of Ordinary Differential Equations

In this chapter:

Topics	Applications/Properties
Elementary Theory of IVPs	Existence and uniqueness of the solution
Taylor-series Methods	
Euler's method Higher-Order Taylor methods	
Runge-Kutta (RK) Methods	
Second-order RK (Heun's method) Fourth-order RK Runge-Kutta-Fehlberg method	Modified Euler's method Variable step-size (adaptive method)
Multi-step Methods	
Adams-Bashforth-Moulton method	
Higher-Order Equations & Systems of Differential Equations	
Implicit Methods	Implicit Differential Equations

Contents of Chapter 5

5.1. Elementary Theory of Initial-Value Problems 180
5.2. Taylor-Series Methods 183
5.3. Runge-Kutta Methods 192
5.4. One-Step Methods: Accuracy Comparison 198
5.5. Multi-step Methods 201
5.6. High-Order Equations & Systems of Differential Equations 205
5.7. Implicit Methods and Implicit Differential Equations 212
Exercises for Chapter 5 218

5.1. Elementary Theory of Initial-Value Problems

Definition 5.1. The first-order **initial value problem (IVP)** is formulated as follows: find $\{y_i(x) : i = 1, 2, \cdots, M\}$ satisfying

$$\begin{aligned} \frac{dy_i}{dx} &= f_i(x, y_1, y_2, \cdots, y_M), \quad i = 1, 2, \cdots, M, \\ y_i(x_0) &= y_{i0}, \end{aligned} \tag{5.1}$$

for a prescribed initial values $\{y_{i0} : i = 1, 2, \cdots, M\}$.

- We assume that (5.1) admits a unique solution in a neighborhood of x_0.
- For simplicity, we consider the case $M = 1$:

$$\begin{aligned} \frac{dy}{dx} &= f(x, y), \\ y(x_0) &= y_0. \end{aligned} \tag{5.2}$$

Theorem 5.2. (Existence and Uniqueness of the Solution): *Suppose that $R = \{(x, y) \mid a \le x \le b, -\infty < y < \infty\}$, f is continuous on R, and $x_0 \in [a, b]$. If f satisfies a Lipschitz condition on R in the variable y, i.e., there is a constant $L > 0$ such that*

$$|f(x, y_1) - f(x, y_2)| \le L|y_1 - y_2|, \quad \forall\, y_1, y_2, \tag{5.3}$$

then the IVP (5.2) has a unique solution $y(x)$ in an interval I, where $x_0 \in I \subset (a, b)$.

Theorem 5.3. *Suppose that $f(x, y)$ is defined on $R \subset \mathbb{R}^2$. If a constant $L > 0$ exists with*

$$\left|\frac{\partial f(x, y)}{\partial y}\right| \le L, \quad \forall\, (x, y) \in R, \tag{5.4}$$

then f satisfies a Lipschitz condition on R in the variable y with the same Lipschitz constant L.

Example **5.4.** Prove that the initial-value problem

$$y' = (x + \sin y)^2, \quad y(0) = 3,$$

has a unique solution on the interval $[-1, 2]$.

Solution.

Example **5.5.** Show that each of the initial-value problems has a unique solution and find the solution.

(a) $y' = e^{x-y}, \quad 0 \leq x \leq 1, \quad y(0) = 1$

(b) $y' = (1 + x^2)y, \quad 3 \leq x \leq 5, \quad y(3) = 1$

Solution. *(Existence and uniqueness)*:

(Find the solution): Here, we will find the solution for (b), using Maple.

Maple-code

```
DE := diff(y(x), x) = y(x)*(x^2 + 1);
                        d                2    \
                       --- y(x) = y(x) \x  + 1/
                        dx
IC := y(3) = 1;
                                 y(3) = 1
dsolve({DE, IC}, y(x));
                                   /1   / 2    \\
                                exp|- x \x  + 3/|
                                   \3           /
                       y(x) = -----------------
                                     exp(12)
```

Strategy **5.6. (Numerical Solution)**: In the following, we describe **step-by-step methods** for (5.2); that is, we start from $y_0 = y(x_0)$ and proceed stepwise.

- In the first step, we compute y_1 which approximate the solution y of (5.2) at $x = x_1 = x_0 + h$, where h is the step size.
- The second step computes an approximate value y_2 of the solution at $x = x_2 = x_0 + 2h$, etc..

Note: We first introduce the Taylor-series methods for (5.2), followed by Runge-Kutta methods and multi-step methods. All of these methods are applicable straightforwardly to (5.1).

5.2. Taylor-Series Methods

Preliminary **5.7.** Here we rewrite the initial value problem (IVP):

$$\begin{cases} y' &= f(x,y), \\ y(x_0) &= y_0. \end{cases} \quad \text{(IVP)} \tag{5.5}$$

For the problem, a continuous approximation to the solution $y(x)$ will not be obtained; instead, approximations to y will be generated at various points, called **mesh points** in the interval $[x_0, T]$, for some $T > x_0$. Let

- $h = (T - x_0)/N_t$, for an integer $N_t \geq 1$
- $x_n = x_0 + nh$, $n = 0, 1, 2, \cdots, N_t$
- y_n be the approximate solution of y at x_n, i.e., $y_n \approx y(x_n)$.

5.2.1. The Euler Method

Step 1

- It is to find an approximation of $y(x_1)$, marching through the first subinterval $[x_0, x_1]$ and using a Taylor-series involving only up to the first-derivative of y.
- Consider the *Taylor series*

$$y(x_1) = y(x_0 + h) = y(x_0) + hy'(x_0) + \frac{h^2}{2}y''(x_0) + \cdots. \tag{5.6}$$

- Utilizing $y(x_0) = y_0$ and $y'(x_0) = f(x_0, y_0)$, the value $y(x_1)$ can be approximated by

$$y_1 = y_0 + h\, f(x_0, y_0), \tag{5.7}$$

where the second- and higher-order terms of h are ignored.

Such an idea can be applied recursively for the computation of solution on later subintervals. Indeed, since

$$y(x_2) = y(x_1) + hy'(x_1) + \frac{h^2}{2}y''(x_1) + \cdots,$$

by replacing $y(x_1)$ and $y'(x_1)$ with y_1 and $f(x_1, y_1)$, respectively, we obtain

$$y_2 = y_1 + h\, f(x_1, y_1), \tag{5.8}$$

which approximates the solution at $x_2 = x_0 + 2h$.

Algorithm 5.8. Summarizing the above, the **Euler method** solving the first-order IVP (5.5) is formulated as

$$y_{n+1} = y_n + h\, f(x_n, y_n), \quad n \geq 0. \tag{5.9}$$

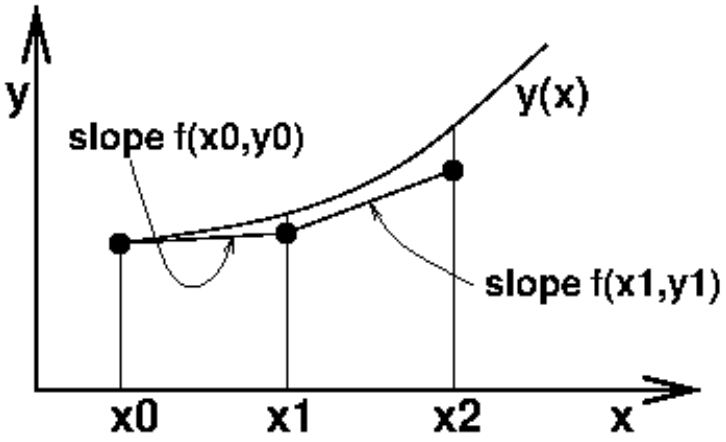


Figure 5.1: The Euler method.

Geometrically it is an approximation of the curve $\{x, y(x)\}$ by a polygon of which **the first segment is tangent to the curve at x_0**, as shown in the above figure.

- For example, y_1 is determined by moving the point (x_0, y_0) by the length of h with the slope $f(x_0, y_0)$, **the evaluation of f ad the left edge point**.

Algegraic Interpretation of the Euler Method

Remark 5.9. Applying the **Fundamental Theorem of Calculus** (Part 1), the solution for the IVP (5.5) can be *implicitly* formulated as

$$y(x) = y_0 + \int_{x_0}^{x} f(t, y(t))\, dt. \tag{5.10}$$

- **Check**:

$$y'(x) = f(x, y(x)) \text{ and } y(x_0) = y_0 + \int_{x_0}^{x_0} f(t, y(t))\, dt = y_0.$$

- When $x = x_1$, Equation (5.10) reads

$$y(x_1) = y_0 + \int_{x_0}^{x_1} f(t, y(t))\, dt. \tag{5.11}$$

- Let us **approximate the integral using the left edge value**:

$$\int_{x_0}^{x_1} f(t, y(t))\, dt \approx \int_{x_0}^{x_1} f(x_0, y(x_0))\, dt = (x_1 - x_0) f(x_0, y_0). \tag{5.12}$$

- Now, it follows from (5.10) and (5.12) that

$$y(x_1) \approx y_0 + h f(x_0, y_0) \quad (=: \mathbf{y_1}), \tag{5.13}$$

which represents the Euler method. □

Convergence of the Euler Method

Theorem 5.10. *Let f satisfy the Lipschitz condition in its second variable, i.e., there is $\lambda > 0$ such that*

$$|f(x, y_1) - f(x, y_2)| \le \lambda |y_1 - y_2|, \quad \forall\, y_1, y_2. \tag{5.14}$$

Then, the Euler method is convergent; more precisely,

$$|y_n - y(x_n)| \le \frac{C}{\lambda} h[(1 + \lambda h)^n - 1], \quad n = 0, 1, 2, \cdots, N_t. \tag{5.15}$$

Note: The term $(1 + \lambda h)^n$ is bounded by a constant:

$$(1 + \lambda h)^n \le (1 + \lambda h)^{N_t} \le e^{\lambda(T - x_0)}, \quad n = 0, 1, 2, \cdots, N_t. \tag{5.16}$$

See Exercise 5.5.

Proof. The true solution y satisfies

$$y(x_{n+1}) = y(x_n) + h f(x_n, y(x_n)) + \mathcal{O}(h^2). \tag{5.17}$$

Thus it follows from (5.9) and (5.17) that

$$\begin{aligned} e_{n+1} &= e_n + h[f(x_n, y_n) - f(x_n, y(x_n))] + \mathcal{O}(h^2) \\ &= e_n + h[f(x_n, y(x_n) + e_n) - f(x_n, y(x_n))] + \mathcal{O}(h^2), \end{aligned}$$

where $e_n = y_n - y(x_n)$. Utilizing (5.14), we have

$$|e_{n+1}| \le (1 + \lambda h)|e_n| + Ch^2. \tag{5.18}$$

Here we will prove (5.15) by using (5.18) and induction. It holds trivially when $n = 0$. Suppose it holds for n. Then,

$$\begin{aligned} |e_{n+1}| &\le (1 + \lambda h)|e_n| + Ch^2 \\ &\le (1 + \lambda h) \cdot \frac{C}{\lambda} h[(1 + \lambda h)^n - 1] + Ch^2 \\ &= \frac{C}{\lambda} h[(1 + \lambda h)^{n+1} - (1 + \lambda h)] + Ch^2 \\ &= \frac{C}{\lambda} h[(1 + \lambda h)^{n+1} - 1], \end{aligned}$$

which completes the proof. □

Example 5.11. Consider

$$\begin{aligned} y' &= y - x^3 + x + 1, \quad 0 \le x \le 3, \\ y(0) &= 0.5. \end{aligned} \tag{5.19}$$

As the step lengths become smaller, $h = 1 \to \frac{1}{4} \to \frac{1}{16}$, the numerical solutions represent the exact solution better, as shown in the following figures:

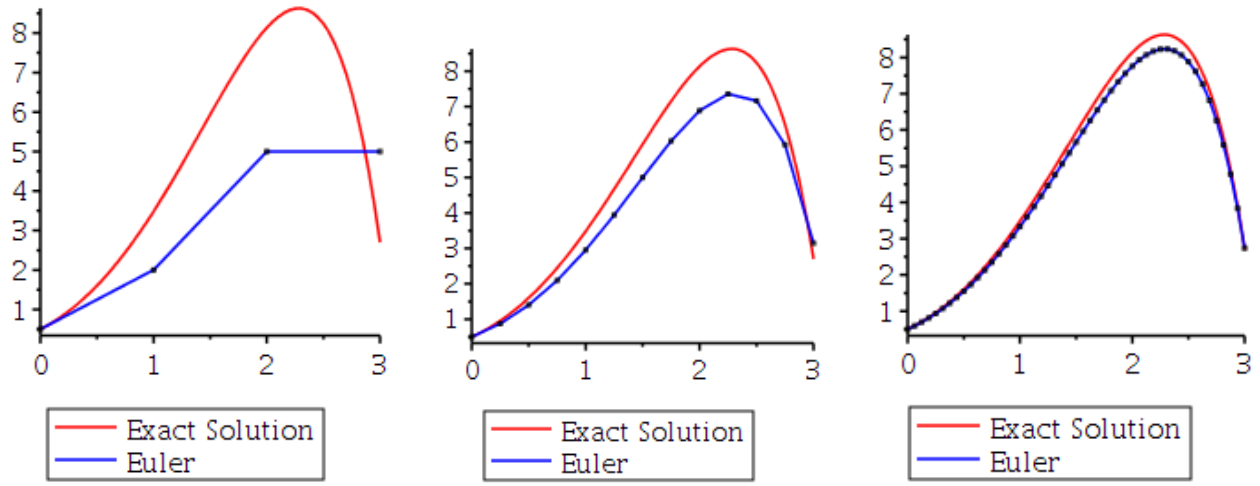


Figure 5.2: The Euler method, with $h = 1 \to \frac{1}{4} \to \frac{1}{16}$.

Example 5.12. Implement a code for the Euler method to solve

$$y' = y - x^3 + x + 1, \quad 0 \le x \le 3, \quad y(0) = 0.5, \quad \text{with } h = \frac{1}{16}.$$

Solution.

Euler.mw

```
Euler := proc(f, x0, b, nx, y0, Y)
    local h, t, w, n:
    h := (b - x0)/nx:
    t := x0; w := y0:
    Y[0] := w:
    for n to nx do
        w := w + h*eval(f, [x = t, y = w]);
        Y[n] := w;
        t := t + h;
    end do:
end proc:

# Now, solve it using "Euler"
f := -x^3 + x + y + 1:
x0 := 0: b := 3: y0 := 0.5:

nx := 48:
YEuler := Array(0..nx):
Euler(f, x0, b, nx, y0, YEuler):

# Check the maximum error
DE := diff(y(x), x) = y(x) - x^3 + x + 1:
dsolve([DE, y(x0) = y0], y(x))
                                    2    3   7
             y(x) = 4 + 5 x + 3 x  + x  - - exp(x)
                                         2
exacty := x -> 4 + 5*x + 3*x^2 + x^3 - 7/2*exp(x):
maxerr := 0:
h := (b - x0)/nx:
for n from 0 to nx do
    maxerr := max(maxerr,abs(exacty(n*h)-YEuler[n]));
end do:
evalf(maxerr)
                              0.39169859
```

5.2.2. Higher-order Taylor Methods

Preliminary **5.13. Higher-order Taylor methods** are based on Taylor series expansion.

- If we expand the solution $y(x)$, in terms of its mth-order Taylor polynomial about x_n and evaluated at x_{n+1}, we obtain

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \cdots + \frac{h^m}{m!}y^{(m)}(x_n) \\ &\quad + \frac{h^{m+1}}{(m+1)!}y^{(m+1)}(\xi_n). \end{aligned} \tag{5.20}$$

- Successive differentiation of the solution, $y(x)$, gives

$$y'(x) = f(x, y(x)), \;\; y''(x) = f'(x, y(x)), \;\; \cdots,$$

and generally,

$$y^{(k)}(x) = f^{(k-1)}(x, y(x)). \tag{5.21}$$

- Thus, we have

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + hf(x_n, y(x_n)) + \frac{h^2}{2!}f'(x_n, y(x_n)) + \cdots + \frac{h^m}{m!}f^{(m-1)}(x_n, y(x_n)) \\ &\quad + \frac{h^{m+1}}{(m+1)!}f^{(m)}(\xi_n, y(\xi_n)) \end{aligned} \tag{5.22}$$

Algorithm **5.14.** The **Taylor method of order** m corresponding to (5.22) is obtained by deleting the remainder term involving ξ_n:

$$y_{n+1} = y_n + h\,T_m(x_n, y_n), \tag{5.23}$$

where

$$T_m(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!}f'(x_n, y_n) + \cdots + \frac{h^{m-1}}{m!}f^{(m-1)}(x_n, y_n). \tag{5.24}$$

Remark 5.15.

- $m = 1 \Rightarrow y_{n+1} = y_n + hf(x_n, y_n)$
  which is the Euler method.
- $m = 2 \Rightarrow y_{n+1} = y_n + h\left[f(x_n, y_n) + \frac{h}{2}f'(x_n, y_n)\right]$
- As m increases, the method achieves higher-order accuracy; however, **it requires to compute derivatives of $f(x, y(x))$.**

Example 5.16. Consider the initial-value problem

$$y' = y - x^3 + x + 1, \quad y(0) = 0.5. \tag{5.25}$$

(a) Find $T_3(x, y)$.

(b) Perform two iterations to find y_2, with $h = 1/2$.

Solution. Part (a): Since $y' = f(x, y) = y - x^3 + x + 1$,

$$\begin{aligned} f'(x, y) &= y' - 3x^2 + 1 \\ &= (y - x^3 + x + 1) - 3x^2 + 1 \\ &= y - x^3 - 3x^2 + x + 2 \end{aligned}$$

and

$$\begin{aligned} f''(x, y) &= y' - 3x^2 - 6x + 1 \\ &= (y - x^3 + x + 1) - 3x^2 - 6x + 1 \\ &= y - x^3 - 3x^2 - 5x + 2 \end{aligned}$$

Thus

$$\begin{aligned} T_3(x, y) &= f(x, y) + \frac{h}{2}f'(x, y) + \frac{h^2}{6}f''(x, y) \\ &= y - x^3 + x + 1 + \frac{h}{2}(y - x^3 - 3x^2 + x + 2) \\ &\quad + \frac{h^2}{6}(y - x^3 - 3x^2 - 5x + 2) \end{aligned} \tag{5.26}$$

For m large, the computation of T_m is time-consuming and cumbersome.

Part (b):

Euler_T3.mw

```
T3 := y - x^3 + x + 1 + 1/2*h*(-x^3 - 3*x^2 + x + y + 2)
      + 1/6*h^2*(-x^3 - 3*x^2 - 5*x + y + 2):
h := 1/2:
x0 := 0: y0 := 1/2:
y1 := y0 + h*eval(T3, [x = x0, y = y0])
                                155
                                ---
                                96
y2 := y1 + h*eval(T3, [x = x0 + h, y = y1])
                               16217
                               -----
                               4608
evalf(%)
                            3.519314236

exacty := x -> 4 + 5*x + 3*x^2 + x^3 - 7/2*exp(x):
exacty(1)
                                7
                           13 - - exp(1)
                                2
evalf(%)
                            3.486013602
#The absolute error:
evalf(abs(exacty(1) - y2))
                            0.033300634
```

5.3. Runge-Kutta Methods

Note: What we are going to do is to solve the *initial value problem* (IVP):

$$\begin{cases} y' &= f(x,y), \\ y(x_0) &= y_0. \end{cases} \quad \text{(IVP)} \tag{5.27}$$

- The **Taylor-series methods** of the preceding section have the drawback of requiring the **computation of derivatives of $f(x,y)$**.
  - It is a tedious and time-consuming procedure for most cases,
  - which makes the Taylor methods seldom used in practice.

Definition 5.17. Runge-Kutta methods

- have **high-order local truncation error of the Taylor methods**,
- but **eliminate the need to compute the derivatives of $f(x,y)$**.

That is, the Runge-Kutta methods are formulated, incorporating a **weighted average of the slope**, as follows:

$$y_{n+1} = y_n + h\left(w_1K_1 + w_2K_2 + \cdots + w_mK_m\right), \tag{5.28}$$

where

(a) $w_j \geq 0$ and $w_1 + w_2 + \cdots + w_m = 1$

(b) K_j are recursive evaluations of the slope $f(x,y)$

(c) Need to determine w_j and other parameters to satisfy

$$w_1K_1 + w_2K_2 + \cdots + w_mK_m \approx T_m(x_n, y_n) + \mathcal{O}(h^m) \tag{5.29}$$

That is, Runge-Kutta methods evaluate an *average slope* of $f(x,y)$ on the interval $[x_n, x_{n+1}]$ in the same order of accuracy as the mth-order Taylor method.

5.3.1. Second-order Runge-Kutta method

Formulation:

$$y_{n+1} = y_n + h\,(w_1 K_1 + w_2 K_2) \tag{5.30}$$

where

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f(x_n + \alpha h, y_n + \beta h K_1) \end{aligned}$$

Requirement: Determine w_1, w_2, α, β such that

$$w_1 K_1 + w_2 K_2 \;=\; T_2(x_n, y_n) + \mathcal{O}(h^2) \;=\; f(x_n, y_n) + \frac{h}{2} f'(x_n, y_n) + \mathcal{O}(h^2). \tag{5.31}$$

Derivation: For the left-hand side of (5.30), the Taylor series reads

$$y(x+h) = y(x) + h\,y'(x) + \frac{h^2}{2} y''(x) + \mathcal{O}(h^3).$$

Since $y' = f$ and $y'' = f_x + f_y y' = f_x + f_y f$,

$$y(x+h) = y(x) + h\,f + \frac{h^2}{2}(f_x + f_y f) + \mathcal{O}(h^3). \tag{5.32}$$

On the other hand, the right-side of (5.30) can be reformulated as

$$\begin{aligned} & y + h(w_1 K_1 + w_2 K_2) \\ &= y + w_1 h\, f(x, y) + w_2 h\, f(x + \alpha h, y + \beta h\, K_1) \\ &= y + w_1 h\, f + w_2 h\,(f + \alpha h\, f_x + \beta h\, f_y f) + \mathcal{O}(h^3), \end{aligned}$$

which reads

$$y + h(w_1 K_1 + w_2 K_2) \;=\; y + (w_1 + w_2) h\, f + h^2 (w_2 \alpha f_x + w_2 \beta f_y f) + \mathcal{O}(h^3). \tag{5.33}$$

The comparison of (5.32) and (5.33) drives the following result, for the second-order Runge-Kutta methods.

Results:

$$w_1 + w_2 = 1, \quad w_2\,\alpha = \frac{1}{2}, \quad w_2\,\beta = \frac{1}{2}. \tag{5.34}$$

Common Choices:

Algorithm 5.18.

I. $w_1 = w_2 = \dfrac{1}{2}$, $\alpha = \beta = 1$:
Then, the algorithm (5.30) becomes

$$y_{n+1} = y_n + \frac{h}{2}(K_1 + K_2), \tag{5.35}$$

where

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f(x_n + h, y_n + h\,K_1) \end{aligned}$$

This algorithm is the **Second-order Runge-Kutta (RK2) method**, which is also known as the **Heun's method**.

II. $w_1 = 0$, $w_2 = 1$, $\alpha = \beta = \dfrac{1}{2}$:
For the choices, the algorithm (5.30) reads

$$y_{n+1} = y_n + h\, f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right) \tag{5.36}$$

which is also known as the **Modified Euler method**.

It follows from (5.32) and (5.33) that the **local truncation error** for the above Runge-Kutta methods are $\mathcal{O}(h^3)$. Thus the **global error** becomes

$$\mathcal{O}(h^2). \tag{5.37}$$

5.3.2. Fourth-order Runge-Kutta method

Formulation:

$$y_{n+1} = y_n + h\left(w_1 K_1 + w_2 K_2 + w_3 K_3 + w_4 K_4\right) \tag{5.38}$$

where

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f(x_n + \alpha_1 h, y_n + \beta_1 h\, K_1) \\ K_3 &= f(x_n + \alpha_2 h, y_n + \beta_2 h\, K_1 + \beta_3 h\, K_2) \\ K_4 &= f(x_n + \alpha_3 h, y_n + \beta_4 h\, K_1 + \beta_5 h\, K_2 + \beta_6 h\, K_3) \end{aligned}$$

Requirement: Determine w_j, α_j, β_j such that

$$w_1 K_1 + w_2 K_2 + w_3 K_3 + w_4 K_4 \;=\; T_4(x_n, y_n) + \mathcal{O}(h^4)$$

The most common choice:

Algorithm 5.19. Fourth-order Runge-Kutta method (RK4): The most commonly used set of parameter values yields

$$y_{n+1} = y_n + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) \tag{5.39}$$

where

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_1) \\ K_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_2) \\ K_4 &= f(x_n + h, y_n + hK_3) \end{aligned}$$

The **local truncation error** for the above RK4 can be derived as

$$\frac{h^5}{5!} y^{(5)}(\xi_n), \tag{5.40}$$

for some $\xi_n \in [x_n, x_{n+1}]$. Thus the **global error** reads, for some $\xi \in [x_0, T]$,

$$\frac{(T - x_0)h^4}{5!} y^{(5)}(\xi). \tag{5.41}$$

Example 5.20. Use RK4 to solve the initial-value problem

$$y' = y - x^3 + x + 1, \quad y(0) = 0.5. \tag{5.42}$$

RK4.mw

```
f := proc(x,w)
        w-x^3+x+1
   end proc:
RK4 := proc(x0,xt,nt,y0,y)
        local h,x,w,n,K1,K2,K3,K4;
        h:=(xt-x0)/nt:
        x:=x0: w:=y0:
        y[0]:=w;
        for n from 1 by 1 to nt do
            K1:=f(x,w);
            K2:=f(x+h/2,w+(h/2)*K1);
            K3:=f(x+h/2,w+(h/2)*K2);
            x:=x+h;
            K4:=f(x,w+h*K3);
            w:=w+(h/6.)*(K1+2*K2+2*K3+K4);
            y[n]:=w;
        end do
end proc:

x0 := 0: xt := 3: nt := 48: y0 := 0.5:
yRK4 := Array(0..nt);
RK4(x0,xt,nt,y0,yRK4):

exacty := x -> 4 + 5*x + 3*x^2 + x^3 - 7/2*exp(x):
h := (xt - x0)/nt:
maxerr := 0:
for n from 0 by 1 to nt do
    maxerr:=max(maxerr,abs(exacty(n*h)-yRK4[n]));
end do:
evalf[16](maxerr)
                          0.00000184873274
```

Convergence Test for RK4, with (5.42):

h	Max-error	Error-ratio
$1/4$	$4.61 \cdot 10^{-4}$	
$1/8$	$2.93 \cdot 10^{-5}$	$\frac{4.61 \cdot 10^{-4}}{2.93 \cdot 10^{-5}} = 15.73378840$
$1/16$	$1.85 \cdot 10^{-6}$	$\frac{2.93 \cdot 10^{-5}}{1.85 \cdot 10^{-6}} = 15.83783784$
$1/32$	$1.01 \cdot 10^{-7}$	$\frac{1.85 \cdot 10^{-6}}{1.01 \cdot 10^{-7}} = 18.31683168$

Thus, the global truncation error of RK4 is in $\mathcal{O}(h^4)$.

5.3.3. Adaptive methods

Remark 5.21.

- **Accuracy** of numerical methods can be improved **by decreasing the step size**.
- Decreasing the step size $\approx$ Increasing the computational cost
- There may be subintervals where a relatively large step size suffices and other subintervals where a small step is necessary to keep the truncation error within a desired limit.
- An **adaptive method** is a numerical method which uses a variable step size.
- Example: **Runge-Kutta-Fehlberg method** (**RKF45**), which uses RK5 to estimate local truncation error of RK4.

5.4. One-Step Methods: Accuracy Comparison

For an accuracy comparison among the one-step methods presented in the previous sections, consider the motion of the **spring-mass system**:

$$\begin{aligned} &y''(t) + \frac{\kappa}{m} y = \frac{F_0}{m}\cos(\mu t), \\ &y(0) = c_0, \quad y'(0) = 0, \end{aligned} \tag{5.43}$$

where m is the mass attached at the end of a spring of the spring constant κ, the term $F_0\cos(\mu t)$ is a periodic driving force of frequency μ, and c_0 is the initial displacement from the equilibrium position.

- It is not difficult to find the analytic solution of (5.43):

$$y(t) = A\cos(\omega t) + \frac{F_0}{m(\omega^2 - \mu^2)}\cos(\mu t), \tag{5.44}$$

  where $\omega = \sqrt{\kappa/m}$ is the angular frequency and the coefficient A is determined corresponding to c_0.

- Let $y_1 = y$ and $y_2 = -y_1'/\omega$. Then, we can reformulate (5.43) as

$$\begin{aligned} y_1' &= -\omega y_2, & y_0(0) &= c_0, \\ y_2' &= \omega y_1 - \frac{F_0}{m\omega}\cos(\mu t), & y_2(0) &= 0. \end{aligned} \tag{5.45}$$

  We will deal with details of *High-Order Equations & Systems of Differential Equations* in § 5.6 on page 205.

- The motion is periodic only if μ/ω is a rational number. We choose

$$m = 1, \;\; F_0 = 40, \;\; A = 1, \;\; \omega = 4\pi, \;\; \mu = 2\pi. \quad (\Rightarrow c_0 \approx 1.33774) \tag{5.46}$$

  Thus the **fundamental period of the motion**

$$T = \frac{2\pi q}{\omega} = \frac{2\pi p}{\mu} = 1.$$

  See Figure 5.3 for the trajectory of the mass satisfying (5.45)-(5.46).

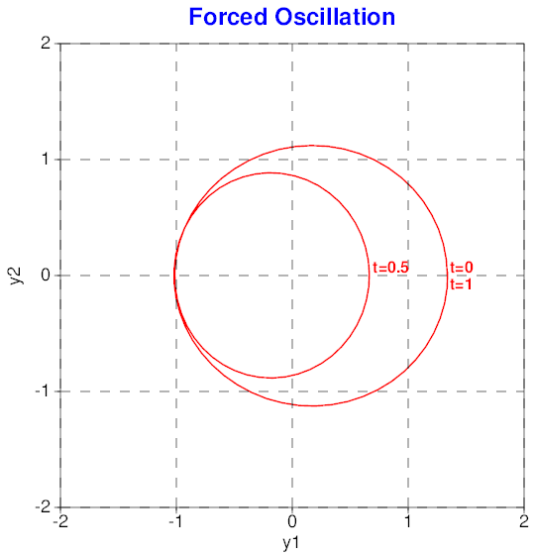


Figure 5.3: The trajectory of the mass satisfying (5.45)-(5.46).

Accuracy comparison

Table 5.1: The ℓ^2-error at $t = 1$ for various time step sizes.

$1/h$	Euler	RK2	RK4
100	1.19	3.31E-2	2.61E-5
200	4.83E-1 (1.3)	8.27E-3 (2.0)	1.63E-6 (4.0)
400	2.18E-1 (1.1)	2.07E-3 (2.0)	1.02E-7 (4.0)
800	1.04E-1 (1.1)	5.17E-4 (2.0)	6.38E-9 (4.0)

- Table 5.1 presents the ℓ^2-error at $t = 1$ for various time step sizes h, defined as

$$|\mathbf{y}_{N_t}^h - \mathbf{y}(1)| = \left(\left[y_{1,N_t}^h - y_1(1) \right]^2 + \left[y_{2,N_t}^h - y_2(1) \right]^2 \right)^{1/2}, \tag{5.47}$$

  where $\mathbf{y}_{N_t}^h$ denotes the computed solution at the N_t-th time step with $h = 1/N_t$.

- The numbers in parenthesis indicate the **order of convergence** α, defined as

$$\alpha := \frac{\ln(E(2h)/E(h))}{\ln 2}, \tag{5.48}$$

  where $E(h)$ and $E(2h)$ denote the errors obtained with the grid spacing to be h and $2h$, respectively.

- As one can see from the table, the one-step methods exhibit the expected accuracy.
- RK4 shows a much better accuracy than the lower-order methods, which explains its popularity.

Definition 5.22. (Order of Convergence): Let's assume that the algorithm under consideration produces error in $\mathcal{O}(h^\alpha)$. Then, we may write

$$E(h) = C\,h^\alpha, \tag{5.49}$$

where h is the grid size. When the grid size is ph, the error will become

$$E(ph) = C\,(ph)^\alpha. \tag{5.50}$$

It follows from (5.49) and (5.50) that

$$\frac{E(ph)}{E(h)} = \frac{C\,(ph)^\alpha}{C\,h^\alpha} = p^\alpha. \tag{5.51}$$

By taking a logarithm, one can solve the above equation for the **order of convergence** α:

$$\alpha = \frac{\ln(E(ph)/E(h))}{\ln p}. \tag{5.52}$$

When $p = 2$, the above becomes (5.48).

5.5. Multi-step Methods

The problem: The first-order initial value problem (IVP)

$$\begin{cases} y' &= f(x,y), \\ y(x_0) &= y_0. \end{cases} \quad \text{(IVP)} \tag{5.53}$$

Numerical Methods:

- **Single-step/Starting methods**: Euler's method, Modified Euler's, Runge-Kutta methods
- **Multi-step/Continuing methods**: Adams-Bashforth-Moulton

Definition 5.23. An ***m*-step method**, $m \geq 2$, for solving the IVP, is a difference equation for finding the approximation y_{n+1} at $x = x_{n+1}$, given by

$$\begin{aligned} y_{n+1} &= a_1 y_n + a_2 y_{n-1} + \cdots + a_m y_{n+1-m} \\ &\quad + h[b_0 f(x_{n+1}, y_{n+1}) + b_1 f(x_n, y_n) + \cdots + b_m f(x_{n+1-m}, y_{n+1-m})]. \end{aligned} \tag{5.54}$$

The m-step method is said to be

$$\begin{cases} \text{explicit or open}, & \text{if } b_0 = 0 \\ \text{implicit or closed}, & \text{if } b_0 \neq 0 \end{cases}$$

Algorithm 5.24. (Fourth-order multi-step methods):

Let $y'_i = f(x_i, y_i)$.

- **Adams-Bashforth method** (explicit)

$$y_{n+1} = y_n + \frac{h}{24}(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3}) \tag{5.55}$$

- **Adams-Moulton method** (implicit)

$$y_{n+1} = y_n + \frac{h}{24}(9y'_{n+1} + 19y'_n - 5y'_{n-1} + y'_{n-2}) \tag{5.56}$$

- **Adams-Bashforth-Moulton method** (predictor-corrector)

$$\begin{aligned} y^*_{n+1} &= y_n + \frac{h}{24}(55y'_n - 59y'_{n-1} + 37y'_{n-2} - 9y'_{n-3}) \\ y_{n+1} &= y_n + \frac{h}{24}(9y'^*_{n+1} + 19y'_n - 5y'_{n-1} + y'_{n-2}) \end{aligned} \tag{5.57}$$

where $y'^*_{n+1} = f(x_{n+1}, y^*_{n+1})$

Remark 5.25.

- y_1, y_2, y_3 can be computed by RK4.
- Multi-step methods may save evaluations of $f(x, y)$ such that in each step, they require only **one or two** new evaluations of $f(x, y)$ to fulfill the step.
- RK methods are accurate enough and easy to implement, so that multi-step methods are rarely applied in practice.
- ABM shows a **strong stability** for special cases, occasionally but not often [6].

ABM.mw

```
## Maple code: Adams-Bashforth-Moulton (ABM) Method
## Model Problem: y'= y- x^3 + x + 1, y(0) = 0.5, 0 <= x <= 3

f := proc(x,w)
        w-x^3+x+1
    end proc:
RK4 := proc(x0,xt,nt,y0,y)
        local h,x,w,n,K1,K2,K3,K4;
        h:=(xt-x0)/nt:
        x:=x0: w:=y0:
        y[0]:=w;
        for n from 1 by 1 to nt do
            K1:=f(x,w);
            K2:=f(x+h/2,w+(h/2)*K1);
            K3:=f(x+h/2,w+(h/2)*K2);
            x:=x+h;
            K4:=f(x,w+h*K3);
            w:=w+(h/6.)*(K1+2*K2+2*K3+K4);
            y[n]:=w;
        end do
end proc:

ABM:= proc(x0,xt,nt,y0,y)
        local h,x,w,n,ystar;
        h:=(xt-x0)/nt:
         ### Initialization with RK4
        RK4(x0,x0+3*h,3,y0,y);
        w:=y[3];
        ### Now, ABM steps
        for n from 4 by 1 to nt do
           x:=x0+n*h;
           ystar:=w +(h/24)*(55*f(x-h,y[n-1])-59*f(x-2*h,y[n-2])
                        +37*f(x-3*h,y[n-3])-9*f(x-4*h,y[n-4]));
           w:=w +(h/24)*(9*f(x,ystar)+19*f(x-h,y[n-1])
                        -5*f(x-2*h,y[n-2])+f(x-3*h,y[n-3]));
```

```
            y[n]:=w;
        end do;
end proc:

x0 := 0: xt := 3: nt := 48: y0 := 0.5:
yABM := Array(0..nt);
ABM(x0,xt,nt,y0,yABM):

exacty := x -> 4 + 5*x + 3*x^2 + x^3 - 7/2*exp(x):
h := (xt - x0)/nt:
maxerr := 0:
for n from 0 by 1 to nt do
    maxerr:=max(maxerr,abs(exacty(n*h)-yABM[n]));
end do:
evalf[16](maxerr)
                          0.00005294884316
```

Note: The maximum error for RK4 $= 1.85 \cdot 10^{-6}$.

5.6. High-Order Equations & Systems of Differential Equations

The problem: 2nd-order initial value problem (IVP)

$$\begin{cases} y'' = f(x, y, y'), & x \in [x_0, T] \\ y(x_0) = y_0, \quad y'(x_0) = u_0, \end{cases} \tag{5.58}$$

An equivalent problem: Let $u = y'$. Then,

$$u' = y'' = f(x, y, y') = f(x, y, u)$$

Thus, the above 2nd-order IVP can be equivalently written as the following system of first-order DEs:

$$\begin{cases} y' = u, & y(x_0) = y_0, \\ u' = f(x, y, u), & u(x_0) = u_0, \end{cases} \quad x \in [x_0, T]. \tag{5.59}$$

The right-side of the DEs involves no derivatives. □

Remark 5.26. The system (5.59) can be solved by one of the numerical methods (we have studied), after modifying it for vector functions.

- Let
$$Y = \begin{bmatrix} y \\ u \end{bmatrix}, \quad F(x, Y) = \begin{bmatrix} u \\ f(x, y, u) \end{bmatrix}, \quad Y_0 := \begin{bmatrix} y_0 \\ u_0 \end{bmatrix}.$$
- Then (5.59) reads
$$Y' = F(x, Y), \quad Y(x_0) = Y_0. \tag{5.60}$$

Example 5.27. Write the following DEs as a system of first-order differential equations.

(a) $\begin{cases} y'' + xy' + y = 0, \\ y(0) = 1, \quad y'(0) = 2. \end{cases}$

(b) $\begin{cases} x'' - x' + 2y'' = e^t, \\ -2x + y'' + 2y = 3t^2. \end{cases}$

Hint: For (b), you should first rewrite ***the first equation*** as $x'' = F(t, x, x')$ and introduce $x' = u$ and $y' = v$.

Solution.

RK4SYS.mw

```
## Ex) IVP of 2 equations:
##  x' = 2x+4y,  x(0)=-1
##  y' = -x+6y,  y(0)= 6, 0 <= t <= 1

ef := proc(t,w,f)
        f(1):=2*w(1)+4*w(2);
        f(2):=-w(1)+6*w(2);
end proc:

RK4SYS := proc(t0,tt,nt,m,x0,x)
        local h,t,w,n,j,K1,K2,K3,K4;
        #### initial setting
        w:=Vector(m):
        K1:=Vector(m):
        K2:=Vector(m):
        K3:=Vector(m):
        K4:=Vector(m):
        h:=(tt-t0)/nt:
        t:=t0;
        w:=x0;
        for j from 1 by 1 to m do
               x[0,j]:=x0(j);
        end do;
        #### RK4 marching
        for n from 1 by 1 to nt do
             ef(t,w,K1);
             ef(t+h/2,w+(h/2)*K1,K2);
             ef(t+h/2,w+(h/2)*K2,K3);
             ef(t+h,w+h*K3,K4);
             w:=w+(h/6.)*(K1+2*K2+2*K3+K4);
             for j from 1 by 1 to m do
                    x[n,j]:=w(j);
             end do
        end do
end proc:
```

```
36
37  m := 2:
38  x0 := Vector(m):
39
40  t0 := 0: tt := 1.: nt := 40:
41  x0(1) := -1:
42  x0(2) := 6:
43
44  xRK4 := Array(0..nt, 1..m):
45  RK4SYS(t0,tt,nt,m,x0,xRK4):
46
47  # Compute the analytic solution
48  #-------------------------------
49  ODE := diff(x(t),t)=2*x(t)+4*y(t), diff(y(t),t)=-x(t)+6*y(t):
50  ivs := x(0) = -1, y(0) = 6;
51  dsolve([ODE, ivs]);
52    /                                      1                        \
53   { x(t) = exp(4 t) (-1 + 26 t), y(t) = - exp(4 t) (24 + 52 t) }
54    \                                      4                        /
55  ex := t -> exp(4*t)*(-1 + 26*t):
56  ey := t -> 1/4*exp(4*t)*(24 + 52*t):
57
58  # Check error
59  #-------------------------------
60  h := (tt - t0)/nt:
61  printf("     n     x(n)      y(n)     error(x)  error(y)\n");
62  printf(" ------------------------------------------------\n");
63  for n from 0 by 2 to nt do
64      printf(" \t %5d %10.3f  %10.3f     %-10.3g %-10.3g\n",
65            n, xRK4[n,1], xRK4[n,2], abs(xRK4[n,1]-ex(n*h)),
66             abs(xRK4[n,2]-ey(n*h)) );
67  end do;
```

Result

n	x(n)	y(n)	error(x)	error(y)
0	-1.000	6.000	0	0
2	0.366	8.122	6.04e-06	4.24e-06
4	2.387	10.890	1.54e-05	1.07e-05
6	5.284	14.486	2.92e-05	2.01e-05
8	9.347	19.140	4.94e-05	3.35e-05
10	14.950	25.144	7.81e-05	5.26e-05
12	22.577	32.869	0.000118	7.91e-05
14	32.847	42.782	0.000174	0.000115
16	46.558	55.474	0.000251	0.000165
18	64.731	71.688	0.000356	0.000232
20	88.668	92.363	0.000498	0.000323
22	120.032	118.678	0.000689	0.000443
24	160.937	152.119	0.000944	0.000604
26	214.072	194.550	0.00128	0.000817
28	282.846	248.313	0.00174	0.0011
30	371.580	316.346	0.00233	0.00147
32	485.741	402.332	0.00312	0.00195
34	632.238	510.885	0.00414	0.00258
36	819.795	647.785	0.00549	0.0034
38	1059.411	820.262	0.00725	0.00447
40	1364.944	1037.359	0.00954	0.00586

RK4SYSTEM.mw

```
## Ex) y''-2*y'+2*y = exp(2*x)*sin(x), 0 <= x <= 1,
##     y(0) = -0.4, y'(0) = -0.6
Digits := 20:
RK4SYSTEM := proc(a,b,nt,X,F,x0,xn)
   local h,hh,t,m,n,j,w,K1,K2,K3,K4;
   #### initial setting
   with(LinearAlgebra):
   m := Dimension(Vector(F));
   w  :=Vector(m);
   K1:=Vector(m);
   K2:=Vector(m);
   K3:=Vector(m);
   K4:=Vector(m);
   h:=(b-a)/nt;  hh:=h/2;
   t :=a;
   w:=x0;
   for j from 1 by 1 to m do
          xn[0,j]:=x0[j];
   end do;
   #### RK4 marching
   for n from 1 by 1 to nt do
      K1:=Vector(eval(F,[x=t,seq(X[i+1]=xn[n-1,i], i = 1..m)]));
      K2:=Vector(eval(F,[x=t+hh,seq(X[i+1]=xn[n-1,i]+hh*K1[i], i = 1..m)]));
      K3:=Vector(eval(F,[x=t+hh,seq(X[i+1]=xn[n-1,i]+hh*K2[i], i = 1..m)]));
      t:=t+h;
      K4:=Vector(eval(F,[x=t,seq(X[i+1]=xn[n-1,i]+h*K3[i], i = 1..m)]));
      w:=w+(h/6)*(K1+2*K2+2*K3+K4);
      for j from 1 by 1 to m do
          xn[n,j]:=evalf(w[j]);
      end do
   end do
end proc:

# Call RK4SYSTEM.mw
#-------------------------------
with(LinearAlgebra):
m := 2:
F := [yp, exp(2*x)*sin(x) - 2*y + 2*yp]:
X := [x, y, yp]:
X0 := <-0.4, -0.6>:
a := 0: b := 1: nt := 10:
Xn := Array(0..nt, 1..m):
RK4SYSTEM(a, b, nt, X, F, X0, Xn):
```

```
44
45  # Compute the analytic solution
46  #-------------------------------
47  DE := diff(y(x), x, x) - 2*diff(y(x), x) + 2*y(x) = exp(2*x)*sin(x):
48  IC := y(0) = -0.4, D(y)(0) = -0.6:
49  dsolve({DE, IC}, y(x))
50                        1
51                y(x) = - exp(2 x) (sin(x) - 2 cos(x))
52                        5
53  ey := x -> 1/5*exp(2*x)*(sin(x) - 2*cos(x))
54  diff(ey(x), x)
55       2                                    1
56       - exp(2 x) (sin(x) - 2 cos(x)) + - exp(2 x) (2 sin(x) + cos(x))
57       5                                    5
58  eyp:=x->2/5*exp(2*x)*(sin(x)-2*cos(x))+1/5*exp(2*x)*(2*sin(x) + cos(x)):
59
60  # Check error
61  #-------------------------------
62  printf(" n   y_n    y(x_n)    y'_n    y'(x_n)   err(y)   err(y')\n");
63  printf("-------------------------------------------------------\n");
64  for n from 0 to nt do
65      xp := h*n + a;
66      printf("  %2d   %12.8f   %12.8f   %12.8f   %12.8f    %.3g    %.3g\n",
67          n, Xn[n, 1], ey(xp), Xn[n, 2], eyp(xp),
68          abs(Xn[n, 1] - ey(xp)), abs(Xn[n, 2] - eyp(xp)));
69  end do:
```

Result

```
1    n       y_n           y(x_n)        y'_n          y'(x_n)      err(y)      err(y')
2   ---------------------------------------------------------------------------------
3     0    -0.40000000   -0.40000000   -0.60000000   -0.60000000   0           0
4     1    -0.46173334   -0.46173297   -0.63163124   -0.63163105   3.72e-07    1.91e-07
5     2    -0.52555988   -0.52555905   -0.64014895   -0.64014866   8.36e-07    2.84e-07
6     3    -0.58860144   -0.58860005   -0.61366381   -0.61366361   1.39e-06    1.99e-07
7     4    -0.64661231   -0.64661028   -0.53658203   -0.53658220   2.02e-06    1.68e-07
8     5    -0.69356666   -0.69356395   -0.38873810   -0.38873905   2.71e-06    9.58e-07
9     6    -0.72115190   -0.72114849   -0.14438087   -0.14438322   3.41e-06    2.35e-06
10    7    -0.71815295   -0.71814890    0.22899702    0.22899243   4.06e-06    4.59e-06
11    8    -0.66971133   -0.66970677    0.77199180    0.77198383   4.55e-06    7.97e-06
12    9    -0.55644290   -0.55643814    1.53478148    1.53476862   4.77e-06    1.29e-05
13   10    -0.35339886   -0.35339436    2.57876634    2.57874662   4.50e-06    1.97e-05
```

5.7. Implicit Methods and Implicit Differential Equations

5.7.1. Revisit of the Euler Method

The problem: The first-order initial value problem (IVP)

$$\begin{cases} y' &= f(x,y), \\ y(x_0) &= y_0. \end{cases} \quad \text{(IVP)} \tag{5.61}$$

The Euler method $y_{n+1} = y_n + hf(x_n, y_n)$ can be rewritten as

$$\frac{y_{n+1}-y_n}{h} = f(x_n, y_n), \quad \text{(Forward Euler)} \tag{5.62}$$

in which the left side is a **forward approximation of** $y'(x_n)$. The formula (5.62) is called the **forward Euler method** for the IVP (5.61).

Definition 5.28. The **backward Euler method** for the IVP (5.61) is defined as

$$\frac{y_{n+1}-y_n}{h} = f(x_{n+1}, y_{n+1}), \quad \text{(Backward Euler)} \tag{5.63}$$

in which the left side is a **backward approximation of** $y'(x_{n+1})$.

Remark 5.29. **The Euler Methods**

- The forward Euler method (5.62) is an explicit method.
- The backward Euler method (5.63) is an implicit method, which requires to solve the following equation for y_{n+1}:

$$y_{n+1} = y_n + h\, f(x_{n+1}, y_{n+1}). \tag{5.64}$$

Claim 5.30. Let f satisfy a Lipschitz condition in y with a Lipschitz constant L. Then the fixed-point iteration applied for (5.64) converges when $h < 1/L$.

Example 5.31. Apply the Euler methods to solve the following IVP

$$y' + 2y = 2 - e^{-4x}, \quad y(0) = 1, \quad 0 \le x \le 1, \tag{5.65}$$

whose theoretical solution is $y(x) = 1 + \frac{1}{2}(e^{-4x} - e^{-2x})$.

Euler.py

```
#!/usr/bin/python3

import numpy as np
from numpy.linalg import norm

def f(x,y): return -2*y -np.exp(-4*x)+2

def exact_y(x):
    return 1+0.5*(np.exp(-4*x)-np.exp(-2*x))

#-----------------------------------------
def forward_Euler(x0,T,Nt,y0):
    x=x0; h =(T-x0)/Nt;
    y = np.zeros([Nt+1,]); y[0]=y0;
    for n in range(Nt):
         y[n+1] = y[n]+h*f(x,y[n])
         x += h;
    return y

#-----------------------------------------
def fixed_point(x,p,yn,h,tol):
    pp=p; err=10.; it0=0
    while(err>tol):
        p = yn+h*f(x,p)
        err = abs(p-pp);
        pp=p; it0+=1
    return p,it0

#-----------------------------------------
def backward_Euler(x0,T,Nt,y0,tol):
    x=x0; h =(T-x0)/Nt;
    y = np.zeros([Nt+1,]); y[0]=y0; it=0;
    for n in range(Nt):
         x += h;
         p,it0 = fixed_point(x,y[n],y[n],h,tol)
         y[n+1] = p; it += it0
    return y,it

```

```
#-----------------------------------------
def Crank_Nicolson(x0,T,Nt,y0,tol):
    x=x0; h =(T-x0)/Nt;
    y = np.zeros([Nt+1,]); y[0]=y0; it=0;
    for n in range(Nt):
         fn = f(x,y[n])
         x += h;
         #fixed-point iteration: included
         p = y[n]; pp=p; err=10.
         while(err>tol):
            p = y[n]+(h/2)*(f(x,p) + fn);
            err = abs(p-pp);
            pp=p; it += 1
         y[n+1] = p
    return y,it

#=========================================
if __name__ == "__main__":
    x0=0.0; T=1; y0=1.0; Nt = 40
    h = (T-x0)/Nt;
    tol1 = h/5
    tol2 = h**2/5

    X = np.linspace(x0,T,Nt+1)
    Y = exact_y(X.T)

    yf = forward_Euler(x0,T,Nt,y0)
    print('forward_Euler:  max-error = %g' %(norm(Y-yf,np.inf)))

    yb,it = backward_Euler(x0,T,Nt,y0,tol1)
    print('backward_Euler: max-error = %g' %(norm(Y-yb,np.inf)))
    print('   average fixed-point iter = %g' %(it/Nt))

    yc,it = Crank_Nicolson(x0,T,Nt,y0,tol2)
    print('Crank_Nicolson: max-error = %g' %(norm(Y-yc,np.inf)))
    print('   average fixed-point iter = %g' %(it/Nt))
```

Output

```
[Fri Jun.23] Euler.py
forward_Euler:  max-error = 0.0085829
backward_Euler: max-error = 0.00787605
   average fixed-point iter = 1.225
Crank_Nicolson: max-error = 0.000168574
   average fixed-point iter = 2.225
```

Algorithm 5.32. **The Crank-Nicolson method** For the IVP (5.61), the **Crank-Nicolson (CN) method** is formulated as

$$\frac{y_{n+1}-y_n}{h}=\frac{f(x_n,y_n)+f(x_{n+1},y_{n+1})}{2}, \quad \text{(CN)} \tag{5.66}$$

of which the global truncation error is $\mathcal{O}(h^2)$:

$$\frac{f(x_n,y_n)+f(x_{n+1},y_{n+1})}{2}=f(x_{n+1/2},y_{n+1/2})+\mathcal{O}(h^2).$$

Note: Equation (5.66) can be rewritten as

$$y_{n+1} = y_n+\frac{h}{2}\big(f_n+f(x_{n+1},y_{n+1})\big), \quad f_n=f(x_n,y_n). \tag{5.67}$$

- Equation (5.67) is implemented in `Euler.py`.
- The CN method is also called the **Trapezoid method**.
- When $f(x_{n+1},y_{n+1})$ is evaluated by using a predicted value:

$$f(x_{n+1},y_{n+1})\to f(x_{n+1},y^*_{n+1}),\ \ y^*_{n+1}=y_n+hf(x_n,y_n),$$

  the resulting algorithm becomes the Heun's method (RK2).

Remark 5.33. **Implicit Methods**
The disadvantage to using implicit methods is the cost to solve (5.64) or (5.67). However, **advantages** include that

- they are usually **more numerically stable**, and
- **a larger step size h** can be used, in particular for the numerical solution of stiff equations.

5.7.2. Implicit Differential Equations

Definition 5.34. An equation of the type

$$F(x, y, y') = 0, \quad y(x_0) = y_0, \tag{5.68}$$

where F is continuous, is called the **first-order implicit ordinary differential equation (IODE-1)**.

- Under a suitable assumption on F, (5.68) can be reduced to

$$y' = f(x, y, y'), \quad y(x_0) = y_0. \tag{5.69}$$

Algorithm 5.35. The **third-order Runge-Kutta method (RK3)** for solving (5.69) is defined as

$$\begin{aligned} y_{n+1} &= y_n + \frac{h}{6}(K_1 + 4K_2 + K_3), \quad \text{where} \\ & K_1 = f(x_n, y_n, K_1) \\ & K_2 = f(x_n + h/2, y_n + (h/2)K_1, K_2) \\ & K_3 = f(x_{n+1}, y_n - hK_1 + 2hK_2, K_3) \end{aligned} \tag{5.70}$$

Note: The terms K_i, $i = 1, 2, 3$, are defined implicitly.

- They can be estimated by applying an iterative method, e.g., the **fixed-point iteration**.
- Each estimation of K_i is of the form $\boldsymbol{d = f(x, y, d)}$.
  - It can be initialized by the last K value.
  - When $n = 0$, $K_1 = y'(x_0)$ must be initialized appropriately.
- The iteration may stop with a *reasonable* stopping tolerance:

$$\texttt{tol} = \mathcal{O}(h^\alpha), \quad \alpha = 3. \tag{5.71}$$

Example 5.36. Consider the problem

$$y' = \frac{1}{10}\big(\cos(x^2y') - \cos(e^y)\big) + \frac{1}{x}, \quad 1 \le x \le 3, \qquad y(1) = 0. \tag{5.72}$$

The exact solution is $y(x) = \ln x$. Apply the RK3 and the fixed-point iteration to solve the problem, with $N_t = 20,\ 40,\ 80$ ($h = 0.1,\ 0.05,\ 0.025$).

Hint: You may use some portions/ideas in `Euler.py`, implemented in Example 5.31, p. 213. For example, you may start with

IODE1.py

```
#!/usr/bin/python3

import numpy as np
from numpy.linalg import norm

def f(x,y,d): return (np.cos(x**2*d)-np.cos(np.exp(y)))/10+1/x
def exact_y(x): return np.log(x)

#-----------------------------------------
def fixed_pointF3(x,y,d,tol):
    err=1.; pd=d;
    while(err>tol):
        d = f(x,y,d)
        err = abs(d-pd); pd=d
    return d

#-----------------------------------------
# Here, implement "a function for the RK3"
#       that uses fixed_pointF3
```

Ans: Set `tol` = $h^3/5$ and for the fixed-point iteration, $y'(x_0)$ is initialized by 1.
Max.Error= [1.55e-05, 3.38e-06, 9.60e-08], for $h = [0.1,\ 0.05,\ 0.025]$.

Exercises for Chapter 5

5.1. Show that the initial-value problem

$$x'(t) = \tan(x), \quad x(0) = 0$$

has a unique solution in the interval $|t| \le \pi/4$. Can you find the solution, by guessing?

5.2. C Use **Taylor method of order** m to approximate the solution of the following initial-value problems.

(a) $y' = e^{x-y}$, $0 \le x \le 1$; $y(0) = 1$, with $h = 0.25$ and $m = 2$.
(b) $y' = e^{x-y}$, $0 \le x \le 1$; $y(0) = 1$, with $h = 0.5$ and $m = 3$.
(c) $y' = \dfrac{\sin x - 2xy}{x^2}$ $1 \le x \le 2$; $y(1) = 2$, with $h = 0.5$ and $m = 4$.

5.3. (Do not use computer programming for this problem.) Consider the initial-value problem:

$$\begin{cases} y' &= 1 + (x-y)^2, \quad 2 \le x \le 3, \\ y(2) &= 1, \end{cases} \tag{5.73}$$

of which the actual solution is $y(x) = x + 1/(1-x)$. Use $h = 1/2$ and a calculator to get the approximate solution at $x = 3$ by applying

(a) Euler's method
(b) RK2
(c) Modified Euler method
(d) RK4

Then, compare their results with the actual value $y(3) = 2.5$.
Ans:

Table 5.2: Comparisons

Method	y_2	Error
Euler	2.625	0.125
RK2	2.48155	0.018448
Modified Euler	2.45506	0.044936
RK4	2.49996	0.00004

5.4. C Now, solve the problem in the preceding exercise, (5.73), by implementing

(a) RK4
(b) Adams-Bashforth-Moulton method

Use $h = 0.05$ and compare the accuracy.
Ans: Error: (RK4, ABM) = $(1.0616 \times 10^{-7}, 6.9225 \times 10^{-6})$

5.5. Verify (5.16).

Hint: Use ① $\lim_{x\to 0^+}(1+x)^{1/x} = e$ and ② the limit is obtained from an increasing sequence, i.e., $(1+x)^{1/x} \nearrow e$ as $x \searrow 0$.

5.6. Prove Claim 5.30, page 212.

5.7. C Consider the following system of first-order differential equations:

$$\begin{cases} u_1' &= u_2 - u_3 + t, & u_1(0) = 1, \\ u_2' &= 3t^2, & u_2(0) = 1, \\ u_3' &= u_2 + e^{-t}, & u_3(0) = -1, \end{cases} \qquad (0 \le t \le 1) \tag{5.74}$$

The actual solution is

$$\begin{aligned} u_1(t) &= -t^5/20 + t^4/4 + t + 2 - e^{-t} \\ u_2(t) &= t^3 + 1 \\ u_3(t) &= t^4/4 + t - e^{-t} \end{aligned}$$

Use *RK4SYSTEM* to approximate the solution with $h = 0.2, 0.1, 0.05$, and compare the errors to see if you can conclude that the algorithm is a fourth-order method for systems of differential equations.

Ans:

Table 5.3: Comparison Results

Step-size	E-Time (sec)	Error $(\times 10^{-4})$
0.2	0.0002	0.3219
0.1	0.0005	0.0266
0.05	0.0013	0.0023

5.8. C Complete an implementation for the problem in Example 5.36, p. 217.
Note: This problem is optional for **undergraduate students**; you will get an extra credit when you solve it. If you decide not to solve the problem, simply write ***"undergraduate"***.

CHAPTER 6

Approximation Theory

In this chapter:

Topics	Applications/Properties
Orthogonal Projection	Gram-Schmidt Process
Least Squares (LS) Approximation Linear models Method of normal equations	Over-determined systems
Regression analysis Linear regression Nonlinear models	 Linearization
Scene analysis RANSAC	Noisy data
Rational Function Approximation	Padé rational approximation
Rank-1 Matrices	Structure Tensor

Contents of Chapter 6

6.1. Orthogonality and the Gram-Schmidt Process 222
6.2. Least-Squares Approximation 226
6.3. Regression Analysis 231
6.4. Scene Analysis with Noisy Data: RANSAC 237
6.5. Padé Rational Function Approximation 243
6.6. Rank-One Matrices and Structure Tensors 247
Exercises for Chapter 6 253

6.1. Orthogonality and the Gram-Schmidt Process

6.1.1. Orthogonal projections

Definition **6.1.** Given a nonzero vector $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$, let

$$\mathbf{y} = \widehat{\mathbf{y}} + \mathbf{z}, \quad \widehat{\mathbf{y}} \,//\, \mathbf{u} \text{ and } \mathbf{z} \perp \mathbf{u}. \tag{6.1}$$

Then

$$\widehat{\mathbf{y}} = \alpha\mathbf{u} = \frac{\mathbf{y}\bullet\mathbf{u}}{\mathbf{u}\bullet\mathbf{u}}\,\mathbf{u}, \quad \mathbf{z} = \mathbf{y} - \widehat{\mathbf{y}}. \tag{6.2}$$

The vector $\widehat{\mathbf{y}}$ is called the **orthogonal projection** of $\mathbf{y}$ onto $\mathbf{u}$, and $\mathbf{z}$ is called the component of $\mathbf{y}$ orthogonal to $\mathbf{u}$. Let $L = \mathit{Span}\{\mathbf{u}\}$. Then we denote

$$\widehat{\mathbf{y}} = \frac{\mathbf{y}\bullet\mathbf{u}}{\mathbf{u}\bullet\mathbf{u}}\,\mathbf{u} = \text{proj}_L\mathbf{y}, \tag{6.3}$$

which is called the orthogonal projection of $\mathbf{y}$ onto L.

We will expand this orthogonal projection for a subspace.

Theorem **6.2. (The Orthogonal Decomposition Theorem)** *Let W be a subspace of $\mathbb{R}^n$. Then each $\mathbf{y} \in \mathbb{R}^n$ can be written uniquely in the form*

$$\mathbf{y} = \widehat{\mathbf{y}} + \mathbf{z}, \tag{6.4}$$

where $\widehat{\mathbf{y}} \in W$ and $\mathbf{z} \in W^\perp$. In fact, if $\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_p\}$ is an orthogonal basis for W, then

$$\begin{aligned} \widehat{\mathbf{y}} &= \text{proj}_W\mathbf{y} = \frac{\mathbf{y}\bullet\mathbf{u}_1}{\mathbf{u}_1\bullet\mathbf{u}_1}\mathbf{u}_1 + \frac{\mathbf{y}\bullet\mathbf{u}_2}{\mathbf{u}_2\bullet\mathbf{u}_2}\mathbf{u}_2 + \cdots + \frac{\mathbf{y}\bullet\mathbf{u}_p}{\mathbf{u}_p\bullet\mathbf{u}_p}\mathbf{u}_p, \\ \mathbf{z} &= \mathbf{y} - \widehat{\mathbf{y}}. \end{aligned} \tag{6.5}$$

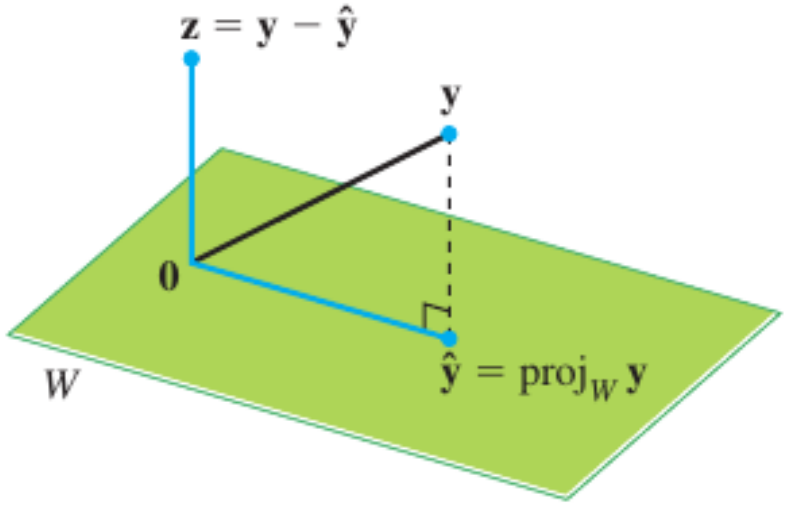


Figure 6.1: Orthogonal projection of $\mathbf{y}$ onto W.

Remark 6.3. (Properties of Orthogonal Decomposition)
Let $\mathbf{y} = \widehat{\mathbf{y}} + \mathbf{z}$, where $\widehat{\mathbf{y}} \in W$ and $\mathbf{z} \in W^{\perp}$. Then

1. $\widehat{\mathbf{y}}$ is called the **orthogonal projection** of $\mathbf{y}$ onto W $(= \text{proj}_W \mathbf{y})$
2. $\widehat{\mathbf{y}}$ is the **closest point** to $\mathbf{y}$ in W.
   (in the sense $\|\mathbf{y} - \widehat{\mathbf{y}}\| \leq \|\mathbf{y} - \mathbf{v}\|$, for all $\mathbf{v} \in W$)
3. $\widehat{\mathbf{y}}$ is called the **best approximation** to $\mathbf{y}$ by elements of W.
4. If $\mathbf{y} \in W$, then $\text{proj}_W \mathbf{y} = \mathbf{y}$.

Proof. 2. For an arbitrary $\mathbf{v} \in W$, $\mathbf{y} - \mathbf{v} = (\mathbf{y} - \widehat{\mathbf{y}}) + (\widehat{\mathbf{y}} - \mathbf{v})$, where $(\widehat{\mathbf{y}} - \mathbf{v}) \in W$. Thus, by the *Pythagorean theorem*,

$$\|\mathbf{y} - \mathbf{v}\|^2 = \|\mathbf{y} - \widehat{\mathbf{y}}\|^2 + \|\widehat{\mathbf{y}} - \mathbf{v}\|^2,$$

which implies that $\|\mathbf{y} - \mathbf{v}\| \geq \|\mathbf{y} - \widehat{\mathbf{y}}\|$. □

Note: The orthogonal projection can be viewed as a **matrix transformation**; see Exercise 6.1, p.253.

6.1.2. The Gram-Schmidt process

Note: The **Gram-Schmidt process** is an algorithm to produce an **orthogonal or orthonormal basis** for any nonzero subspace of $\mathbb{R}^n$.

Example 6.4. Let $W = Span\{\mathbf{x}_1, \mathbf{x}_2\}$, where $\mathbf{x}_1 = \begin{bmatrix} 3 \\ 6 \\ 0 \end{bmatrix}$ and $\mathbf{x}_2 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}$. Find an orthogonal basis for W.

Key idea: Orthogonal projection

$$\left\{ \begin{array}{l} \mathbf{x}_1 \\ \mathbf{x}_2 \end{array} \right\} \Rightarrow \left\{ \begin{array}{l} \mathbf{x}_1 \\ \mathbf{x}_2 = \alpha \mathbf{x}_1 + \mathbf{v}_2 \end{array} \right. \Rightarrow \left\{ \begin{array}{l} \mathbf{v}_1 = \mathbf{x}_1 \\ \mathbf{v}_2 = \mathbf{x}_2 - \text{proj}_{\{\mathbf{x}_1\}} \mathbf{x}_2 \end{array} \right.$$

where $\mathbf{x}_1 \bullet \mathbf{v}_2 = 0$. Then $W = Span\{\mathbf{x}_1, \mathbf{x}_2\} = Span\{\mathbf{v}_1, \mathbf{v}_2\}$.

Solution. $\alpha = \dfrac{\mathbf{x}_2 \bullet \mathbf{x}_1}{\mathbf{x}_1 \bullet \mathbf{x}_1} = \dfrac{15}{45} = \dfrac{1}{3}$.

$$\Rightarrow \mathbf{v}_2 = \mathbf{x}_2 - \frac{1}{3}\mathbf{x}_1 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \frac{1}{3}\begin{bmatrix} 3 \\ 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

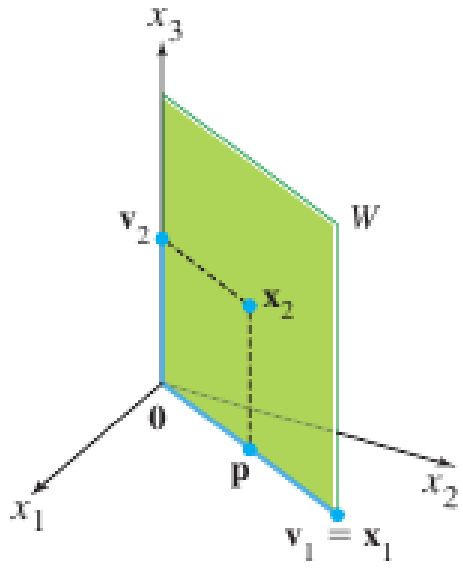


Figure 6.2: Construction of an orthogonal basis $\{\mathbf{v}_1, \mathbf{v}_2\}$.

Theorem 6.5. (The Gram-Schmidt Process) *Given a basis* $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ *for a nonzero subspace* W *of* $\mathbb{R}^n$*, define*

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{x}_1 \\
\mathbf{v}_2 &= \mathbf{x}_2 - \frac{\mathbf{x_2}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 \\
\mathbf{v}_3 &= \mathbf{x}_3 - \frac{\mathbf{x_3}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 - \frac{\mathbf{x_3}\bullet\mathbf{v_2}}{\mathbf{v_2}\bullet\mathbf{v_2}}\mathbf{v}_2 \\
&\vdots \\
\mathbf{v}_p &= \mathbf{x}_p - \frac{\mathbf{x_p}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 - \frac{\mathbf{x_p}\bullet\mathbf{v_2}}{\mathbf{v_2}\bullet\mathbf{v_2}}\mathbf{v}_2 - \cdots - \frac{\mathbf{x_p}\bullet\mathbf{v_{p-1}}}{\mathbf{v}_{p-1}\bullet\mathbf{v}_{p-1}}\mathbf{v}_{p-1}
\end{aligned} \tag{6.6}$$

Then $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ *is an* ***orthogonal basis*** *for* W*. In addition,*

$$\mathit{Span}\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\} = \mathit{Span}\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\}, \quad \textit{for } 1 \le k \le p. \tag{6.7}$$

Remark 6.6. **The Gram-Schmidt process**

- Define
$$\mathbf{u}_k = \frac{\mathbf{v}_k}{\|\mathbf{v}_k\|}, \quad \text{for } 1 \le k \le p. \tag{6.8}$$
Then $\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_p\}$ is an **orthonormal basis** for W. It is often implemented for the **normalized Gram-Schmidt process**.
- The Gram-Schmidt process can be easily modified for QR factorization; see Section 8.3.

Self-study 6.7. Find an *orthonormal basis* for $W = \mathit{Span}\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, where

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} -2 \\ 2 \\ 1 \\ 0 \end{bmatrix}, \text{ and } \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \end{bmatrix}.$$

6.2. Least-Squares Approximation

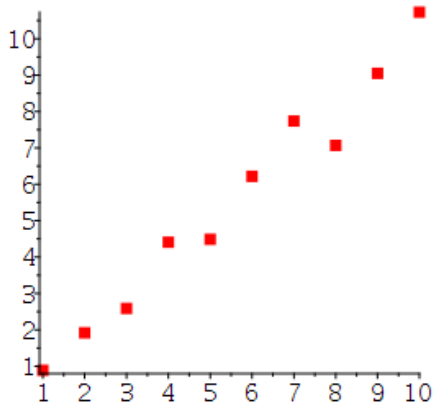
Definition 6.8. For a given dataset $\{(x_i, y_i)\}$, let a continuous function $p(x)$ be constructed.

(a) p is an **interpolation** if it passes (interpolates) all the data points.

(b) p is an **approximation** if it approximates (represents) the data points.

Dataset

```
with(LinearAlgebra): with(CurveFitting):
n := 100: roll := rand(-n..n):
m := 10: xy := Matrix(m, 2):
for i to m do
    xy[i, 1] := i;
    xy[i, 2] := i + roll()/n;
end do:
plot(xy,color=red, style=point, symbol=solidbox, symbolsize=20);
```

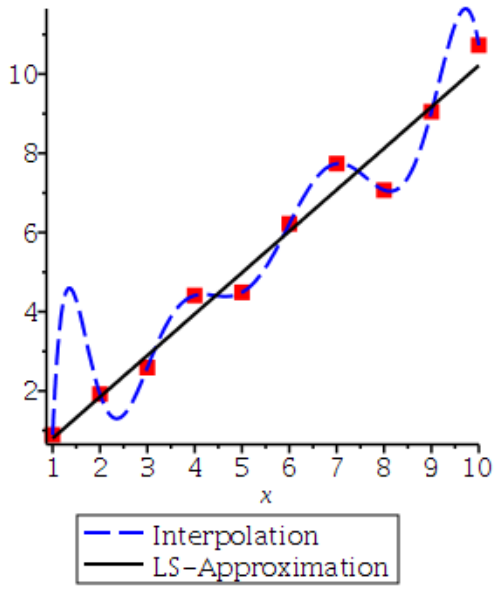


$$p := PolynomialInterpolation(xy, x);$$

$$\frac{1507}{12096000}x^9 - \frac{27977}{4032000}x^8 + \frac{36751}{224000}x^7 - \frac{68851}{32000}x^6 + \frac{1975291}{115200}x^5 - \frac{5471791}{64000}x^4 + \frac{199836041}{756000}x^3 - \frac{486853651}{1008000}x^2 + \frac{39227227}{84000}x - \frac{1771}{10}$$

$$L := CurveFitting[LeastSquares](xy, x);$$

$$-\frac{121}{500} + \frac{523}{500}x$$



Note: Interpolation may be too oscillatory to be useful; furthermore, it may not be defined.

6.2.1. The Least-Squares (LS) Problem

Note: Let A is an $m \times n$ matrix. Then $A\mathbf{x} = \mathbf{b}$ may have no solution, particularly when $m > n$. In real-world,

- $m \gg n$, where m represents the number of data points and n denotes the dimension of the points
- Need to find a best solution for $A\mathbf{x} \approx \mathbf{b}$

Definition 6.9. Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, and $\mathbf{b} \in \mathbb{R}^m$. The **least-squares problem** is to find $\widehat{\mathbf{x}} \in \mathbb{R}^n$ which minimizes $\|A\mathbf{x} - \mathbf{b}\|_2$:

$$\begin{aligned} \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2, \\ &\quad \text{or, equivalently,} \\ \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2, \end{aligned} \tag{6.9}$$

where $\widehat{\mathbf{x}}$ is called a **least-squares solution** of $A\mathbf{x} = \mathbf{b}$.

Remark 6.10.

- For all $\mathbf{x} \in \mathbb{R}^n$, $A\mathbf{x}$ will necessarily be in *Col* A, a subspace of $\mathbb{R}^m$. So we seek an $\mathbf{x}$ that makes $A\mathbf{x}$ the closest point in *Col* A to $\mathbf{b}$.
- Let $\widehat{\mathbf{b}} = \text{proj}_{Col\,A}\,\mathbf{b}$. Then $A\mathbf{x} = \widehat{\mathbf{b}}$ has a solution and there is an $\widehat{\mathbf{x}} \in \mathbb{R}^n$ such that
$$A\widehat{\mathbf{x}} = \widehat{\mathbf{b}}. \tag{6.10}$$
- $\widehat{\mathbf{x}}$ is an LS solution of $A\mathbf{x} = \mathbf{b}$.
- The quantity $\|\mathbf{b} - \widehat{\mathbf{b}}\| = \|\mathbf{b} - A\widehat{\mathbf{x}}\|$ is called the **least-squares error**. That is,
$$\|\mathbf{b} - A\widehat{\mathbf{x}}\| \leq \|\mathbf{b} - A\mathbf{x}\|, \quad \text{for all } \mathbf{x} \in \mathbb{R}^n, \tag{6.11}$$
where $\|\cdot\| = \|\cdot\|_2$.

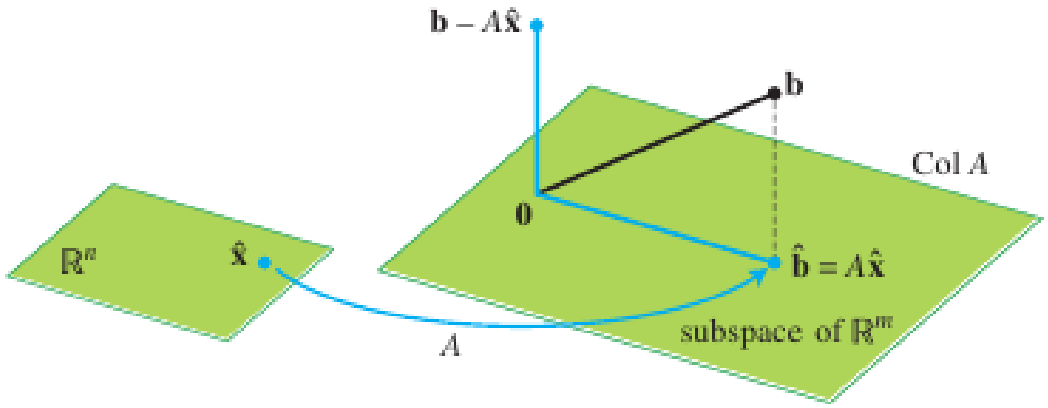


Figure 6.3: The LS solution $\widehat{\mathbf{x}}$ is in $\mathbb{R}^n$.

Remark 6.11. If $A \in \mathbb{R}^{n \times n}$ is invertible, then $A\mathbf{x} = \mathbf{b}$ has a unique solution $\widehat{\mathbf{x}}$ and therefore
$$\|\mathbf{b} - A\widehat{\mathbf{x}}\| = 0. \tag{6.12}$$

6.2.2. Normal Equations

Theorem 6.12. *The set of LS solutions of* $A\mathbf{x} = \mathbf{b}$ *coincides with the nonempty set of solutions of the* ***normal equations***

$$A^T A\mathbf{x} = A^T\mathbf{b}. \tag{6.13}$$

Proof. Suppose $\hat{\mathbf{x}}$ satisfies $A\hat{\mathbf{x}} = \hat{\mathbf{b}}$

$\Rightarrow \mathbf{b} - \hat{\mathbf{b}} = \mathbf{b} - A\hat{\mathbf{x}} \perp \mathit{Col}\, A$

$\Rightarrow \mathbf{a}_j \bullet (\mathbf{b} - A\hat{\mathbf{x}}) = 0$ for all columns $\mathbf{a}_j$

$\Rightarrow \mathbf{a}_j{}^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$ for all columns $\mathbf{a}_j$ (Note that $\mathbf{a}_j{}^T$ is a row of A^T)

$\Rightarrow A^T(\mathbf{b} - A\hat{\mathbf{x}}) = 0$

$\Rightarrow A^T A\hat{\mathbf{x}} = A^T\mathbf{b}$ □

Remark 6.13. Theorem 6.12 implies that LS solutions of $A\mathbf{x} = \mathbf{b}$ are solutions of the normal equations $A^T A\hat{\mathbf{x}} = A^T\mathbf{b}$.

- When $A^T A$ is not invertible, the normal equations have either no solution or infinitely many solutions.
- So, **data acquisition** is important, to make it invertible.

Theorem 6.14. (Method of normal equations) *Let* $A \in \mathbb{R}^{m\times n}$, $m \geq n$. *The following statements are logically equivalent:*

a. *The equation* $A\mathbf{x} = \mathbf{b}$ *has* ***a unique LS solution*** *for each* $\mathbf{b} \in \mathbb{R}^m$.

b. ***The columns of A are linearly independent.***

c. *The matrix* $A^T A$ *is invertible.*

When these statements are true, the **unique** *LS solution* $\hat{\mathbf{x}}$ *is given by*

$$\hat{\mathbf{x}} = (A^T A)^{-1} A^T\mathbf{b}. \tag{6.14}$$

Definition 6.15. *The matrix* $(A^T A)^{-1}A^T$ *is called the* ***pseudoinverse*** *of* A*:*

$$A^+ \overset{\text{def}}{=\!=} (A^T A)^{-1}A^T. \tag{6.15}$$

Example 6.16. Describe all LS solutions of the equation $A\mathbf{x} = \mathbf{b}$, given

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 1 \\ 3 \\ 8 \\ 2 \end{bmatrix}.$$

Solution. There will be infinitely many LS solutions.

$$\text{rref}([A^TA, A^T\mathbf{b}]) = \text{rref}\left(\begin{bmatrix} 4 & 2 & 2 & 14 \\ 2 & 2 & 0 & 4 \\ 2 & 0 & 2 & 10 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 1 & 5 \\ 0 & 1 & -1 & -3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\textit{Ans: } \widehat{\mathbf{x}} = \begin{bmatrix} 5 \\ -3 \\ 0 \end{bmatrix} + x_3 \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}, x_3 \in \mathbb{R}$$

Method of Calculus

Let $\mathcal{J}(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|^2 = (A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b})$ and $\widehat{\mathbf{x}}$ a minimizer of $\mathcal{J}(\mathbf{x})$.

- Then we must have

$$\nabla_{\mathbf{x}}\mathcal{J}(\widehat{\mathbf{x}}) = \frac{\partial \mathcal{J}(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\widehat{\mathbf{x}}} = \mathbf{0}. \tag{6.16}$$

- Let's compute the gradient of $\mathcal{J}$.

$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial\big((A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b})\big)}{\partial \mathbf{x}} \\ &= \frac{\partial(\mathbf{x}^TA^TA\mathbf{x} - 2\mathbf{x}^TA^T\mathbf{b} + \mathbf{b}^T\mathbf{b})}{\partial \mathbf{x}} \\ &= 2A^TA\mathbf{x} - 2A^T\mathbf{b}. \end{aligned} \tag{6.17}$$

- By setting the last term to zero, we obtain the normal equations in (6.16).

6.3. Regression Analysis

Definition 6.17. Regression analysis is a set of statistical methods used for the estimation of relationships between a dependent variable and one or more independent variables.

6.3.1. Regression Line

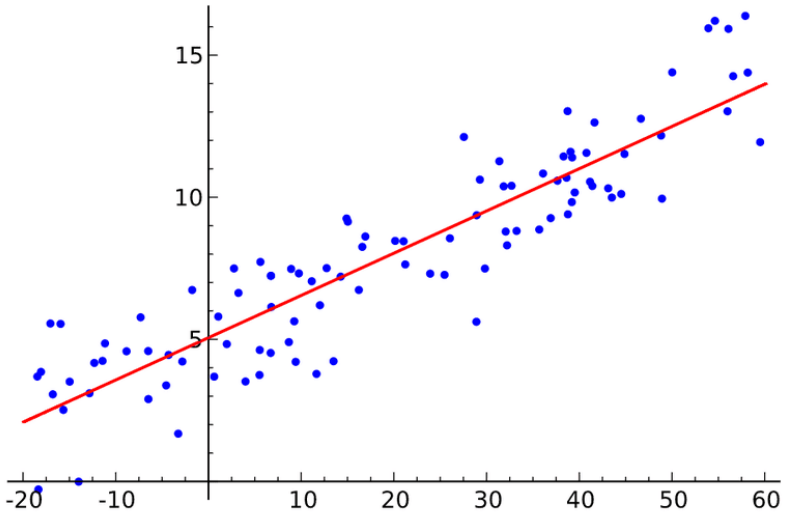


Figure 6.4: A regression line.

Definition 6.18. Suppose a set of experimental data points are given as

$$(x_1, y_1),\ (x_2, y_2),\ \cdots,\ (x_m, y_m)$$

such that the graph is close to a line. We determine a line

$$y = \beta_0 + \beta_1 x \tag{6.18}$$

that is as close as possible to the given points, measured in the L^2-norm. This line is called the **least-squares line** or **regression line** of y on x. The parameters β_0, β_1 are called **regression coefficients**.

Calculation of Least-Squares Lines

Consider a least-squares (LS) model of the form $y = \beta_0 + \beta_1 x$, for a given data set $\{(x_i, y_i) \mid i = 1, 2, \cdots, m\}$.

- Then

$$\begin{array}{|c|} \hline \text{Predicted } y\text{-value} \\ \hline \end{array} \qquad \begin{array}{|c|} \hline \text{Observed } y\text{-value} \\ \hline \end{array}$$

$$\begin{array}{ccc} \beta_0 + \beta_1 x_1 & = & y_1 \\ \beta_0 + \beta_1 x_2 & = & y_2 \\ \vdots & & \vdots \\ \beta_0 + \beta_1 x_m & = & y_m \end{array} \tag{6.19}$$

- It can be equivalently written as

$$X\boldsymbol{\beta} = \mathbf{y}, \tag{6.20}$$

where

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_m \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Here we call X the **design matrix**, $\boldsymbol{\beta}$ the **parameter vector**, and $\mathbf{y}$ the **observation vector**.

- Thus the LS solution can be determined by solving the normal equations:

$$X^T X\boldsymbol{\beta} = X^T \mathbf{y}, \tag{6.21}$$

provided that $X^T X$ is invertible.

- The normal equations for the regression line read

$$\begin{bmatrix} m & \Sigma x_i \\ \Sigma x_i & \Sigma x_i^2 \end{bmatrix} \boldsymbol{\beta} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \end{bmatrix}. \tag{6.22}$$

Remark 6.19. (Pointwise construction of the normal equations) The normal equations for the regression line in (6.22) can be rewritten as

$$\sum_{i=1}^{m} \begin{bmatrix} 1 & x_i \\ x_i & x_i^2 \end{bmatrix} \boldsymbol{\beta} = \sum_{i=1}^{m} \begin{bmatrix} y_i \\ x_i y_i \end{bmatrix}. \tag{6.23}$$

- The pointwise construction of the normal equation is convenient when either points are first to be searched or weights are applied depending on the point location.
- The idea is applicable for other regression models as well.

Self-study 6.20. Find the equation $y = \beta_0 + \beta_1 x$ of least-squares line that best fits the given points:

$(-1, 0),\ (0, 1),\ (1, 2),\ (2, 4)$

Solution.

6.3.2. Least-Squares Fitting of Other Curves

Remark 6.21. Consider a regression model of the form

$$y = \beta_0 + \beta_1 x + \beta_2 x^2,$$

for a given data set $\{(x_i, y_i) \mid i = 1, 2, \cdots, m\}$. Then

Predicted y-value		Observed y-value
$\beta_0 + \beta_1 x_1 + \beta_2 x_1^2$	$=$	y_1
$\beta_0 + \beta_1 x_2 + \beta_2 x_2^2$	$=$	y_2
$\vdots$		$\vdots$
$\beta_0 + \beta_1 x_m + \beta_2 x_m^2$	$=$	y_m

(6.24)

It can be equivalently written as

$$X\boldsymbol{\beta} = \mathbf{y}, \tag{6.25}$$

where

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{bmatrix}, \quad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Now, it can be solved through *normal equations*:

$$X^T X\boldsymbol{\beta} = \begin{bmatrix} \Sigma 1 & \Sigma x_i & \Sigma x_i^2 \\ \Sigma x_i & \Sigma x_i^2 & \Sigma x_i^3 \\ \Sigma x_i^2 & \Sigma x_i^3 & \Sigma x_i^4 \end{bmatrix} \boldsymbol{\beta} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \\ \Sigma x_i^2 y_i \end{bmatrix} = X^T \mathbf{y} \tag{6.26}$$

Self-study 6.22. Find an LS curve of the form $y = \beta_0 + \beta_1 x + \beta_2 x^2$ that best fits the given points:

$(0, 1), (1, 1), (1, 2), (2, 3)$.

Solution.

Ans: $y = 1 + 0.5x^2$

6.3.3. Nonlinear Regression: Linearization

Strategy 6.23. For nonlinear models, a **change of variables** can be applied to get a linear model.

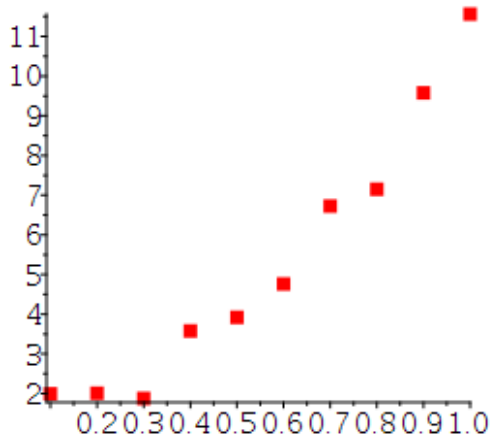
Model	Change of Variables		Linearization
$y = A + \dfrac{B}{x}$	$\widetilde{x} = \dfrac{1}{x},\ \widetilde{y} = y$	$\Rightarrow$	$\widetilde{y} = A + B\widetilde{x}$
$y = \dfrac{1}{A + Bx}$	$\widetilde{x} = x,\ \widetilde{y} = \dfrac{1}{y}$	$\Rightarrow$	$\widetilde{y} = A + B\widetilde{x}$
$y = Ce^{Dx}$	$\widetilde{x} = x,\ \widetilde{y} = \ln y$	$\Rightarrow$	$\widetilde{y} = \ln C + D\widetilde{x}$
$y = \dfrac{1}{A + B\ln x}$	$\widetilde{x} = \ln x,\ \widetilde{y} = \dfrac{1}{y}$	$\Rightarrow$	$\widetilde{y} = A + B\widetilde{x}$

(6.27)

The above table contains just a few examples of linearization; for other nonlinear models, use your imagination and creativity.

Example 6.24. Find the best fitting curve of the form $y = ce^{dx}$ for the data

$$\begin{bmatrix} 0.1 & 1.9940 \\ 0.2 & 2.0087 \\ 0.3 & 1.8770 \\ 0.4 & 3.5783 \\ 0.5 & 3.9203 \\ 0.6 & 4.7617 \\ 0.7 & 6.7246 \\ 0.8 & 7.1491 \\ 0.9 & 9.5777 \\ 1.0 & 11.5625 \end{bmatrix}$$



Solution. Applying the natural log function (ln) to $y = ce^{dx}$ gives

$$\ln y = \ln c + dx. \tag{6.28}$$

Using the change of variables

$$Y = \ln y, \quad a_0 = \ln c, \quad a_1 = d, \quad X = x,$$

the equation (6.28) reads

$$Y = a_0 + a_1 X, \tag{6.29}$$

for which one can apply the **linear LS procedure**.

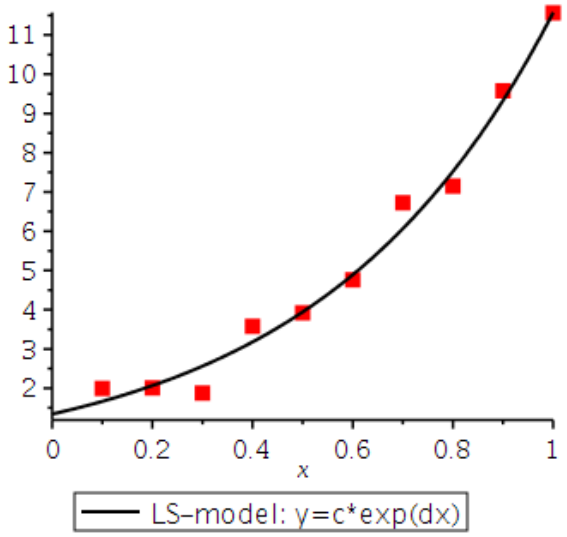
Linearized regression

```
# The transformed data
xlny := Matrix(m, 2):
for i to m do
    xlny[i, 1] := xy[i, 1];
    xlny[i, 2] := ln(xy[i, 2]);
end do:

# The linear LS
L := CurveFitting[LeastSquares](xlny, x, curve = b*x + a);
    0.295704647799999 + 2.1530740654363654 x

# Back to the original parameters
c := exp(0.295704647799999) = 1.344073123
d := 2.15307406543637:

# The desired nonlinear model
c*exp(d*x);
                1.344073123 exp(2.15307406543637 x)
```



6.4. Scene Analysis with Noisy Data: RANSAC

Note: **Scene analysis** is concerned with **the interpretation of acquired data (data interpretation)** in terms of a set of predefined models. It consists of 2 subproblems:

1. finding the best model (**classification problem**)
2. computing the best parameter values (**parameter estimation problem**)

- **Traditional parameter estimation techniques**, such as *least-squares* (LS), optimize the model to all of the presented data.
  - Those techniques are simple averaging methods, based on the **smoothing assumption**: *There will always be good data points enough to smooth out any gross deviation.*
- However, in many interesting parameter estimation problems, **the smoothing assumption does not hold**; that is, the data set may involve gross errors such as noise.
  - Thus, in order to obtain more reliable model parameters, there must be **internal mechanisms** to determine which points are matching to the model (**inliers**) and which points are false matches (**outliers**).

6.4.1. Weighted Least-Squares

Definition 6.25. When certain data points are more important or more reliable than the others, one may try to compute the coefficient vector with larger weights on more reliable data points. The **weighted least-squares method** is an LS method which involves a weight. The weight is often given as a diagonal matrix

$$W = \text{diag}(w_1, w_2, \cdots, w_m). \tag{6.30}$$

The **weight matrix** W can be decided either *manually* or *automatically*.

Algorithm 6.26. (Weighted Least-Squares)

- Given data $\{(x_i, y_i)\}$, $1 \le i \le m$, the best-fitting curve can be found by solving an over-determined algebraic system (6.20):

$$X\boldsymbol{\beta} = \mathbf{y}. \tag{6.31}$$

- When a weight matrix is applied, the above system can be written as

$$WX\boldsymbol{\beta} = W\mathbf{y}. \tag{6.32}$$

- Thus its **weighted normal equations** read

$$X^T W X\boldsymbol{\beta} = X^T W\mathbf{y}. \tag{6.33}$$

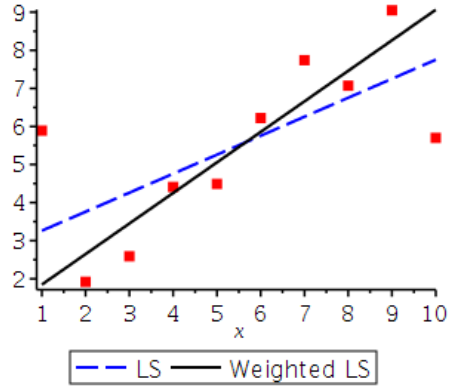
Example 6.27. Given data, find the LS line with and without a weight. When a weight is applied, weigh the first and the last data point by $1/4$.

$$\text{xy} := \begin{bmatrix} 1. & 2. & 3. & 4. & 5. & 6. & 7. & 8. & 9. & 10. \\ 5.89 & 1.92 & 2.59 & 4.41 & 4.49 & 6.22 & 7.74 & 7.07 & 9.05 & 5.7 \end{bmatrix}^T$$

Solution.

Weighted-LS

```
LS := CurveFitting[LeastSquares](xy, x);
   2.7639999999999967 + 0.49890909090909125 x
WLS := CurveFitting[LeastSquares](xy, x,
          weight = [1/4,1,1,1,1,1,1,1,1,1/4]);
   1.0466694879390623 + 0.8019424460431653 x
```



6.4.2. RANdom SAmple Consensus (RANSAC)

The **random sample consensus** (**RANSAC**) is one of the most powerful tools for the reconstruction of ground structures from point cloud observations in many applications. The algorithm utilizes **iterative search techniques** for a set of inliers to find a proper model for given data.

Algorithm 6.28. (RANSAC) (Fischler-Bolles, 1981) [11]

Input: Measurement set $\boldsymbol{X} = \{\mathbf{x}_i\}$, the error tolerance τ_e, the stopping threshold η, and the maximum number of iterations N.

1. Select randomly a minimum point set S, required to determine a hypothesis.
2. Generate a hypothesis $\mathbf{p} = g(S)$.
3. Compute the hypothesis consensus set, fitting within the error tolerance τ_e:
   $\mathcal{C} = \text{inlier}(\boldsymbol{X}, \mathbf{p}, \tau_\text{e})$
4. If $|\mathcal{C}| \geq \eta|\boldsymbol{X}|$, then re-estimate a hypothesis $\mathbf{p} = g(\mathcal{C})$ and stop.
5. Otherwise, repeat steps 1–4 (maximum of N times).

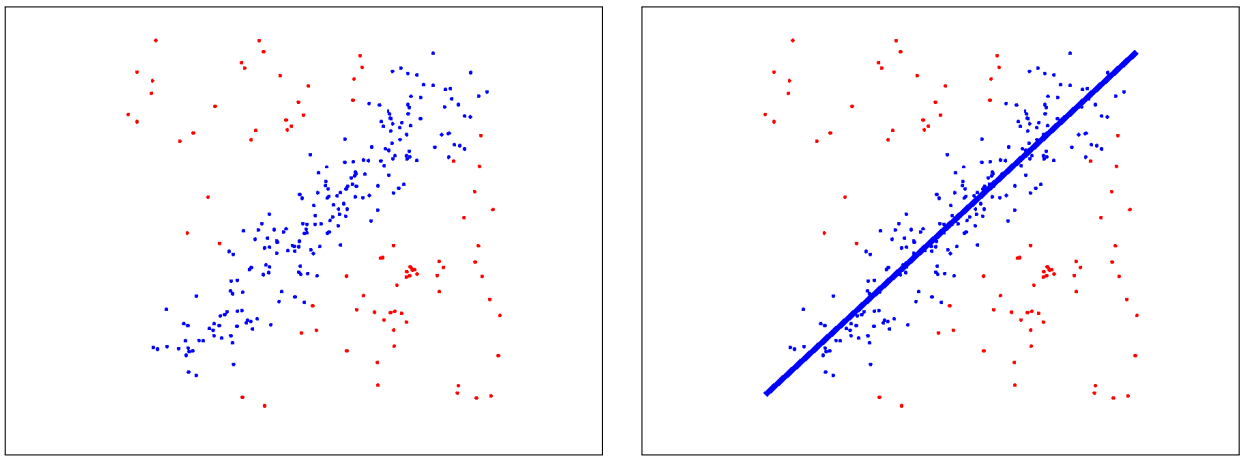


Figure 6.5: A synthetic data

Example 6.29. Let's set a hypothesis for a **regression line**.

1. **Minimum point set** S: a set of two points, (x_1, y_1) and (x_2, y_2).
2. **Hypothesis** p: $y = a + bx \quad (\Rightarrow a + bx - y = 0)$

$$y = b(x - x_1) + y_1 = a + bx \quad \Leftarrow \quad b = \frac{y_2 - y_1}{x_2 - x_1}, \;\; a = y_1 - bx_1.$$

3. **Consensus set** $\mathcal{C}$:

$$\mathcal{C} = \left\{ (x_i, y_i) \in \boldsymbol{X} \mid d = \frac{|a + bx_i - y_i|}{\sqrt{b^2 + 1}} \leq \tau_e \right\} \tag{6.34}$$

Note: In practice:

- **Step 2**: A hypothesis p is the set of model parameters, rather than the model itself.
- **Step 3**: The consensus set can be represented more conveniently using an index array. That is,

$$\text{ID}(i) = \begin{cases} 1 & \text{if } \mathbf{x}_i \in \mathcal{C} \\ 0 & \text{if } \mathbf{x}_i \notin \mathcal{C} \end{cases} \tag{6.35}$$

Remark 6.30. The "`inlier`" function in Step 3 collects points whose distance from the model, $f(\mathbf{p})$, is not larger than τ_e. Thus, the function can be interpreted as an **automatic weighting mechanism**. Indeed, for each point $\mathbf{x}_i$,

$$\textit{dist}(f(\mathbf{p}), \mathbf{x}_i) \begin{cases} \leq \tau_e, & \text{then} \quad w_i = 1 \\ > \tau_e, & \text{then} \quad w_i = 0 \end{cases} \tag{6.36}$$

Then the re-estimation in Step 4, $\mathbf{p} = g(\mathcal{C})$, can be seen as an parameter estimation $\mathbf{p} = g(\boldsymbol{X})$ with the corresponding weight matrix $W = \{w_1, w_2, \cdots, w_m\}$.

Remark 6.31.

- The above basic RANSAC algorithm is an *iterative search* method for a set of inliers which may produce presumably accurate model parameters.
- It is simple to implement and efficient. However, it is problematic and often erroneous.
- The main disadvantage of RANSAC is that RANSAC is **unrepeatable**; it may yield different results in each run so that none of the results can be optimal.

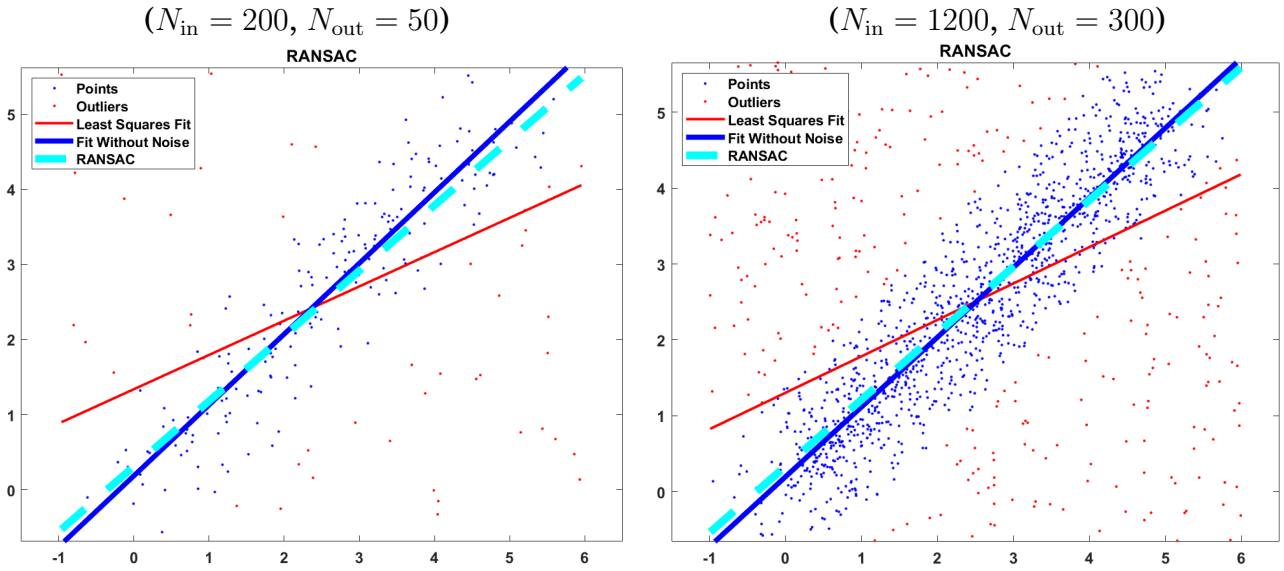


Figure 6.6: The RANSAC for linear-type synthetic datasets.

Table 6.1: The RANSAC: model fitting $y = a_0 + a_1 x$. The algorithm runs 1000 times for each dataset to find the standard deviation of the error: $\sigma(a_0 - \widehat{a}_0)$ and $\sigma(a_1 - \widehat{a}_1)$.

Data	$\sigma(a_0 - \widehat{a}_0)$	$\sigma(a_1 - \widehat{a}_1)$	E-time (sec)
1	0.1156	0.0421	0.0156
2	0.1101	0.0391	0.0147

The RANSAC is neither repeatable nor optimal.
In order to overcome the drawbacks, various variants have been studied in the literature. Nonetheless, it remains a prevailing algorithm for finding inliers. For variants, see e.g.,

- Maximum Likelihood Estimation Sample Consensus (MLESAC) [36]
- Progressive Sample Consensus (PROSAC) [4]
- Recursive RANSAC (R-RANSAC) [26]

Note: Recently an optimal variant of RANSAC is suggested by (Kim *et al.*, 2024) [24], called the *Iterative Update Sample Consensus* (**IUSAC**). The IUSAC is a few times more effective in both accuracy and efficiency than RANSAC and its variants.

6.5. Padé Rational Function Approximation

The Class of Algebraic Polynomials

Advantages:

- There are a **sufficient number of polynomials** to approximate any continuous function on a closed interval to within an arbitrary tolerance.
- Polynomials are **easy to evaluate**.
- Their **derivatives and integrals** exist and are easily determined.

Disadvantage:

- **Tendency for oscillation**, which often causes the error bound to significantly exceed the average approximation error.

Definition **6.32.** A **rational function** r of degree N has the form

$$r(x) = \frac{p(x)}{q(x)}, \tag{6.37}$$

where $p(x)$ and $q(x)$ are polynomials whose degrees sum to N.

Padé Approximation:

Definition **6.33.** **Padé rational function** of degree $N = n + m$, approximating $f(x)$, has the form

$$f(x) \approx R_{n,m}(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + \cdots + p_n x^n}{1 + q_1 x + \cdots + q_m x^m} \tag{6.38}$$

Determination of the constants $\{p_0, p_1, \cdots, p_n, q_1, \cdots, q_m\}$

We will find them to satisfy

$$\frac{d^k}{dx^k}(f - R_{n,m})(0) = 0, \quad \text{for } k = 0, 1, \cdots, N. \tag{6.39}$$

- Let f have the **Maclaurin series expansion**

$$f(x) = \sum_{i=0}^{\infty} a_i x^i. \tag{6.40}$$

- Then, for $q_0 = 1$,

$$f(x) - R_{n,m}(x) = \frac{\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{m} q_i x^i\right) - \left(\sum_{i=0}^{n} p_i x^i\right)}{q(x)}. \tag{6.41}$$

- Our objective is to determine the constants $\{p_0, p_1, \cdots, p_n, q_1, \cdots, q_m\}$ such that (6.39) is satisfied, which is equivalent to $(f - R_{n,m})$ having a zero of multiplicity $N + 1$ at $x = 0$.
- As a consequence, we choose the constants so that the numerator on the right side of (6.41),

$$\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{m} q_i x^i\right) - \left(\sum_{i=0}^{n} p_i x^i\right), \tag{6.42}$$

  has no terms of degree $\leq N$.
- In order to simplify the notation, we define

$$q_{m+1} = q_{m+2} = \cdots = q_N = 0,$$
$$p_{n+1} = p_{n+2} = \cdots = p_N = 0.$$

- Then the coefficient of x^k in (6.42) can be expresses as

$$\sum_{i=0}^{k} a_i q_{k-i} - p_k, \quad k = 0, 1, \cdots, N. \tag{6.43}$$

Summary **6.34.** **(Padé approximation)** Let f have the Maclaurin series expansion $f(x) = \sum_{i=0}^{\infty} a_i x^i$. Then the rational function for Padé approximation, $R_{n,m}$, results from the solution of $N+1$ linear equations

$$\sum_{i=0}^{k} a_i q_{k-i} = p_k, \quad k = 0, 1, \cdots, N. \tag{6.44}$$

Example **6.35.** Find the Padé approximation to e^{-x} of degree 5 with $n = 3$ and $m = 2$.

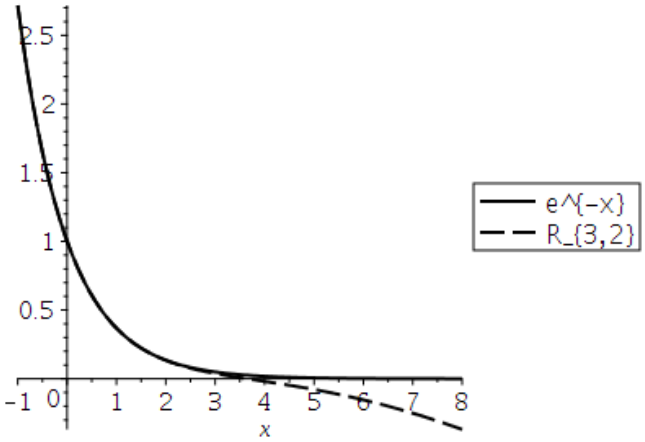
Solution.

```
PadeRatApprox := proc(f, n, m)
    local i, j, N, p, q, eq, PQ, T, a, R;
    N := m + n;
    T := convert(taylor(f, x = 0, N + 1), polynom);
    for i from 0 to N do a_i := coeff(T, x, i); end do;
    q_0 := 1;
    for i from m + 1 to N do q_i := 0; end do;
    for i from n + 1 to N do p_i := 0; end do;
    for i from 0 to N do
        eq_i := add(a_j · q_{i-j}, j = 0..i) = p_i;
    end do;
    PQ := solve({seq(eq_i, i = 0..N)}, {seq(p_i, i = 0..n), seq(q_i, i = 1..m)});
    R := x → add(rhs(PQ[1 + i]) · x^i, i = 0..n) / (q_0 + add(rhs(PQ[n + 1 + i]) · x^i, i = 1..m));
    return R;
end proc:
```

$$R32 := PadeRatApprox\left(e^{-x}, 3, 2\right):$$

> $R32(x)$

$$\frac{1 - \frac{3}{5}x + \frac{3}{20}x^2 - \frac{1}{60}x^3}{1 + \frac{2}{5}x + \frac{1}{20}x^2}$$



Use of built-in command

```
T := taylor(exp(-x), x = 0, 6):
convert(T, ratpoly, 3, 2)
                          3     3   2   1   3
                      1 - - x + -- x  - -- x
                          5     20      60
                      ----------------------
                              2     1   2
                          1 + - x + -- x
                              5     20
```

6.6. Rank-One Matrices and Structure Tensors

Rank-one Matrices

Definition 6.36. A **rank-one matrix** is a matrix with rank equal to one.

Theorem 6.37. *Every **rank-1 matrix** $A \in \mathbb{R}^{m\times n}$ can be written as an **outer product***

$$A = \mathbf{u}\mathbf{v}^T, \quad \mathbf{u} \in \mathbb{R}^m, \ \mathbf{v} \in \mathbb{R}^n. \tag{6.45}$$

Theorem 6.38. *Let $A = \mathbf{u}\mathbf{v}^T$ be a rank-1 matrix. Then*

$$||A||_2 = ||A||_F = ||\mathbf{u}||_2||\mathbf{v}||_2. \tag{6.46}$$

Proof. Using the definitions of the norms, we have

$$\begin{aligned} \|A\|_2 &\equiv \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{||\mathbf{x}||_2=1} \|A\mathbf{x}\|_2 = \max_{||\mathbf{x}||_2=1} \|\mathbf{u}\mathbf{v}^T\mathbf{x}\|_2 \\ &= \max_{||\mathbf{x}||_2=1} \|\mathbf{u}\|_2 \cdot |\mathbf{v}^T\mathbf{x}| = ||\mathbf{u}||_2||\mathbf{v}||_2, \\ \|A\|_F &= \sqrt{\mathbf{tr}(AA^T)} = \sqrt{\mathbf{tr}(\mathbf{u}\mathbf{v}^T\mathbf{v}\mathbf{u}^T)} \\ &= \sqrt{||\mathbf{u}||_2^2||\mathbf{v}||_2^2} = ||\mathbf{u}||_2||\mathbf{v}||_2, \end{aligned} \tag{6.47}$$

which completes the proof. □

Example 6.39. Let $\mathbf{u} = [6, 3, -6]^T$ and $\mathbf{v} = [1, -2, 2]^T$.

(a) Form $A = \mathbf{u}\mathbf{v}^T$.

(b) Find $||A||_2$.

The Induced Matrix 2-Norm of A

Definition 6.40. *The **induced matrix** 2**-norm** can be defined by*

$$\|A\|_2 = \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \max_{||\mathbf{x}||_2=1} \|A\mathbf{x}\|_2 = \max_{||\mathbf{x}||_2=||\mathbf{y}||_2=1} |\mathbf{y}^T A\mathbf{x}|. \tag{6.48}$$

*It is also called an **operator** 2**-norm** or **subordinate** 2**-norm**.*

Theorem 6.41. Properties of the Matrix 2-Norm

(a) $||A^T||_2 = ||A||_2$

(b) $||A^T A||_2 = ||A||_2^2$

(c) $||AA^T||_2 = ||A||_2^2$

Proof.

(a) The claim follows from the fact that $\mathbf{y}^T A\mathbf{x}$ is a scalar and therefore $(\mathbf{y}^T A\mathbf{x})^T = \mathbf{x}^T A^T \mathbf{y}$ and $|\mathbf{y}^T A\mathbf{x}| = |\mathbf{x}^T A^T \mathbf{y}|$.

(b) Using the **Cauchy-Schwarz inequality**,

$$\begin{aligned} ||A^T A||_2 &= \max_{||\mathbf{x}||_2=||\mathbf{y}||_2=1} |\mathbf{y}^T A^T A\mathbf{x}| \\ &\leq \max_{||\mathbf{x}||_2=||\mathbf{y}||_2=1} ||A\mathbf{y}||_2 ||A\mathbf{x}||_2 = ||A||_2^2. \end{aligned} \tag{6.49}$$

Now, we choose a unit vector $\mathbf{z}$, for which $||A\mathbf{z}||_2 = ||A||_2$. Then

$$\begin{aligned} ||A||_2^2 &= ||A\mathbf{z}||_2^2 = (A\mathbf{z})^T(A\mathbf{z}) = \mathbf{z}^T A^T A\,\mathbf{z} \\ &\leq \max_{||\mathbf{x}||_2=||\mathbf{y}||_2=1} |\mathbf{y}^T A^T A\mathbf{x}| \equiv ||A^T A||_2. \end{aligned} \tag{6.50}$$

(c) Using the results in (a) and (b),

$$||A||_2^2 = ||A^T||_2^2 = ||(A^T)^T A^T||_2 = ||AA^T||_2, \tag{6.51}$$

which completes the proof. □

Structure Tensor

Definition 6.42. The **structure tensor** is a matrix derived from the gradient of a function $f(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^n$: it is defined as

$$S(\mathbf{x}) = (\nabla f)(\nabla f)^T(\mathbf{x}), \tag{6.52}$$

which describes the distribution of the gradient of f in a neighborhood of the point $\mathbf{x}$. The structure tensor is often used in **image processing** and **computer vision**.

Claim 6.43. Structure Tensor in Two Variables $\mathbf{x} = (x, y) \in \mathbb{R}^2$
When $\mathbf{x} = (x, y) \in \mathbb{R}^2$, the structure tensor in (6.52) reads

$$S(\mathbf{x}) = (\nabla f)(\nabla f)^T(\mathbf{x}) = \begin{bmatrix} f_x^2 & f_x f_y \\ f_x f_y & f_y^2 \end{bmatrix}(\mathbf{x}). \tag{6.53}$$

(a) The matrix S is symmetric and **positive semidefinite**.

(b) $||S||_2 = ||\nabla f||_2^2$, which is the maximum eigenvalue of S.

(c) $det\, S = 0$, which implies that the number 0 is an eigenvalue of S.

The two eigenvalue-eigenvector pairs $(\lambda_i, \mathbf{v}_i)$, $i = 1, 2$, are

$$\begin{aligned} \lambda_1 &= ||\nabla f||_2^2, & \mathbf{v}_1 &= \nabla f \\ \lambda_2 &= 0, & \mathbf{v}_2 &= [-f_y,\ f_x]^T \end{aligned} \tag{6.54}$$

Proof.

(a) The matrix S is clearly symmetric. Let $\mathbf{v} \in \mathbb{R}^2$. Then

$$\mathbf{v}^T S \mathbf{v} = \mathbf{v}^T (\nabla f)(\nabla f)^T \mathbf{v} = |(\nabla f)^T \mathbf{v}|^2 \geq 0, \tag{6.55}$$

which proves that S is positive semidefinite.

(b) It follows from (6.46). Since S is symmetric, $||S||_2$ must be the maximum eigenvalue of S. (See Theorem 1.60 (f), p. 24)

(c) $det\, S = f_x^2 f_y^2 - (f_x f_y)^2 = 0 \Rightarrow S$ is not invertible $\Rightarrow$ An eigenvalue of S must be 0.

(d) It is not difficult to check that $S\mathbf{v}_i = \lambda_i \mathbf{v}_i$, $i = 1, 2$. □

Structure Tensor, Applied to Color Image Processing

- Let $I(\mathbf{x}) = (r, g, b)(\mathbf{x})$ be a **color image** defined on a rectangular region Ω, $\mathbf{x} = (x, y) \in \Omega \subset \mathbb{R}^2$.
- Then

$$\nabla I(\mathbf{x}) = \begin{bmatrix} (r,g,b)_x \\ (r,g,b)_y \end{bmatrix} (\mathbf{x}), \tag{6.56}$$

and therefore the structure tensor of I reads

$$\begin{aligned} S_I(\mathbf{x}) &= (\nabla I)(\nabla I)^T(\mathbf{x}) = \begin{bmatrix} (r,g,b)_x \\ (r,g,b)_y \end{bmatrix} \left[(r,g,b)_x^T \;\; (r,g,b)_y^T \right] (\mathbf{x}) \\ &= \begin{bmatrix} r_x^2 + g_x^2 + b_x^2 & r_x r_y + g_x g_y + b_x b_y \\ r_x r_y + g_x g_y + b_x b_y & r_y^2 + g_y^2 + b_y^2 \end{bmatrix} (\mathbf{x}), \end{aligned} \tag{6.57}$$

which is **no longer a rank-1 matrix**, but symmetric.

Claim 6.44. Properties of the Structure Tensor S_I

(a) It follows from Theorem 6.41 and Theorem 1.60 (f) that

$$||S_I||_2 = ||\nabla I||_2^2 = \lambda_{\max}(S_I). \tag{6.58}$$

Thus the **gradient magnitude** reads

$$||\nabla \mathbf{I}||_2 = \sqrt{\lambda_{\max}(\mathbf{S_I})}. \tag{6.59}$$

(b) Rewrite S_I as

$$S_I = \begin{bmatrix} J_{xx} & J_{xy} \\ J_{xy} & J_{yy} \end{bmatrix}. \tag{6.60}$$

Then, the **quadratic formula** solving $det\,(S_I - \lambda I) = 0$ for λ gives

$$\lambda_{\max}(S_I) = \lambda_1 = \frac{J_{xx} + J_{yy} + \sqrt{(J_{xx} - J_{yy})^2 + 4J_{xy}^2}}{2}, \tag{6.61}$$

and its corresponding eigenvector reads

$$\mathbf{v}_1 = \begin{bmatrix} J_{yy} - \lambda_1 \\ -J_{xy} \end{bmatrix}, \tag{6.62}$$

which is the **edge normal direction**.

Matlab-code 6.45. Structure Tensor

structure_tensor.m

```
function [grad,theta] = structure_tensor(u)
% For a color image u, apply Structure Tensor (ST) to get
%   the gradient magnitude, grad, &
%   the edge normal direction, theta.

%%--- Sobel Derivatives ----------------------------------
[ux,uy] = sobel_derivatives(u);

%%--- Jacobian: J = (grad u)*(grad u)' -------------------
Jxx = dot(ux,ux,3); Jxy = dot(ux,uy,3); Jyy = dot(uy,uy,3);

%%--- The first eigenvalue-eigenvecor (e1,v1) ------------
D  = (Jxx-Jyy).^2 + 4*Jxy.^2;
e1 = (Jxx+Jyy+sqrt(D))/2;
grad  = sqrt(e1);                % sqrt(e1) = |grad u|

%%--- Choose the max from two rows, for theta ------------
% [ Jxx-e1   Jxy   ];   v = [  Jxy      ]  or  [ Jyy-e1 ]
% [  Jxy    Jyy-e1 ];       [ -(Jxx-e1) ]      [  -Jxy  ]

S11 = Jxx-e1;
idx = abs(S11)>=abs(Jxy);
V11 = (idx.*Jxy + (1-idx).*(Jyy-e1));
V12 =-(idx.*S11 + (1-idx).*Jxy);
theta = atan2(V12,V11);
```

Remark 6.46. Eigenvalues and Eigenvectors of S_I

- $(\lambda_1, \mathbf{v}_1)$ is an eigenvalue-eigenvector pair for S_I:

$$(S_I - \lambda_1 I)\mathbf{v}_1 = \begin{bmatrix} J_{xx} - \lambda_1 & J_{xy} \\ J_{xy} & J_{yy} - \lambda_1 \end{bmatrix} \mathbf{v}_1 = 0. \tag{6.63}$$

  Thus a row of $(S_I - \lambda_1 I)$ must be a multiple of the other row.
- A normal vector to a row of $(S_I - \lambda_1 I)$ can decide $\mathbf{v}_1$.
- If ux or uy is zero (at a point), then a row of $(S_I - \lambda_1 I)$ must be zero at the point.
- The eigenvector cannot be zero by definition; you can select the row having a larger magnitude for $\mathbf{v}_1$.

sobel_derivatives.m

```
function [ux,uy] = sobel_derivatives(u)
% This function returns Sobel derivatives, using
% conv2(u,C,'valid') which requires boundary correction.
% input u: either in gray or color

%%--- initialization -------------------------
if isa(u,'uint8'), u = im2double(u); end
[m,n,d] = size(u);
ux = zeros(m,n,d); uy = zeros(m,n,d);
C = [1 2 1; 0 0 0; -1 -2 -1];

%%--- conv2, with 'valid' --------------------
for k = 1:d
    ux(2:end-1,2:end-1,k) = conv2(u(:,:,k),C, 'valid');
    uy(2:end-1,2:end-1,k) = conv2(u(:,:,k),C','valid');
end

%%--- expand, up to the boundary -------------
ux(1,:,:) = ux(2,:,:); ux(end,:,:) = ux(end-1,:,:);
ux(:,1,:) = ux(:,2,:); ux(:,end,:) = ux(:,end-1,:);
uy(1,:,:) = uy(2,:,:); uy(end,:,:) = uy(end-1,:,:);
uy(:,1,:) = uy(:,2,:); uy(:,end,:) = uy(:,end-1,:);
```

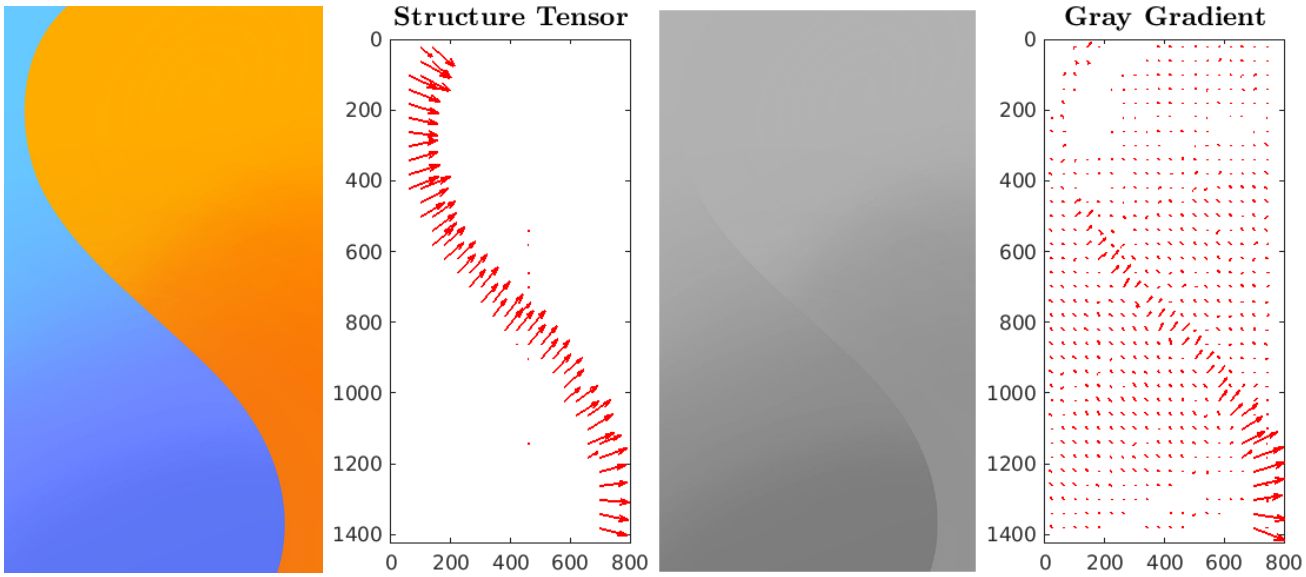


Figure 6.7: Detection of edges and angles.

You will use the method of structure tensor for the detection of edges for color images; see Exercise 6.6.

Exercises for Chapter 6

6.1. Let $\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_p\}$ be an **orthonormal basis** for a subspace W of $\mathbb{R}^n$, that is, $\mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}$. Let $U = [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_p] \in \mathbb{R}^{n\times p}$. Prove that

$$\text{proj}_w\mathbf{y} = UU^T\mathbf{y}, \quad \text{for all } \mathbf{y} \in \mathbb{R}^n. \tag{6.64}$$

(Thus the orthogonal projection can be viewed as a *matrix transformation*.)

6.2. C Let $A = \begin{bmatrix} -10 & 13 & 7 & -11 \\ 2 & 1 & -5 & 3 \\ -6 & 3 & 13 & -3 \\ 16 & -16 & -2 & 5 \\ 2 & 1 & -5 & -7 \end{bmatrix}$. Use the Gram-Schmidt process to produce an orthogonal basis for the column space of A.

Ans: $\mathbf{v}_4 = (0, 5, 0, 0, -5)$

6.3. C Given data

x_i	0.2	0.4	0.6	0.8	1.	1.2	1.4	1.6	1.8	2.
y_i	1.88	2.13	1.76	2.78	3.23	3.82	6.13	7.22	6.66	9.07

(a) Plot the data (scattered point plot)
(b) Decide what curve fits the data best.
(c) Implement an LS code to find the curve.
(d) Plot the curve superposed over the point plot.

6.4. Determine Padé approximation of degree 6 for $f(x) = \sin x$, and compare the results at $x_i = 0.2i$, $i = 0, 1, \cdots, 5$, with $f(x)$ and with its sixth Maclaurin polynomial

$$\texttt{convert}(\texttt{taylor}(\texttt{sin}(\texttt{x}), \texttt{x} = 0, 7), \texttt{polynom}) = x - \frac{1}{6}x^3 + \frac{1}{120}x^5$$

(a) with $n = 2,\ m = 4$
(b) with $n = 3,\ m = 3$
(c) with $n = 4,\ m = 2$

(You should not use any built-in functions which produce the results immediately.)

6.5. This problem uses the data in Example 6.27, p.238.

(a) Implement the method of normal equations for the least-squares regression to find the best-fitting line.
(b) The RANSAC, Algorithm 6.28 is implemented for you below. Use the code to analyze the performance of the RANSAC.
- Set $\tau_e = 1$, $\gamma = \eta|\boldsymbol{X}| = 8$, and $N = 100$.
- Run `ransac2` 100 times to get the minimum, maximum, and average number of iterations for the RANSAC to find an acceptable hypothesis consensus set.

(c) Plot the best-fitting lines found from (a) and (b), superposed over the data.

get_hypothesis_WLS.m

```
function p = get_hypothesis_WLS(X,C)
% Get hypothesis p, with C being used as weights
% Output: p = [a,b], where y= a+b*x

m = size(X,1);

A = [ones(m,1) X(:,1)];
A = A.*C;   %A = bsxfun(@times,A,C);
r = X(:,2).*C;

p = ((A'*A)\(A'*r))';
```

inlier.m

```
function C = inlier(X,p,tau_e)
% Input: p=[a,b] s.t. a+b*x-y=0

m = size(X,1);
C = zeros(m,1);

a = p(1); b=p(2);
factor = 1./sqrt(b^2+1);
for i=1:m
    xi = X(i,1); yi = X(i,2);
    dist = abs(a+b*xi-yi)*factor;  %distance from point to line
    if dist<=tau_e, C(i)=1; end
end
```

ransac2.m

```
function [p,C,iter] = ransac2(X,tau_e,gamma,N)
% Input:   X = {(x_i,y_i)}
%          tau_e: the error tolerance
%          gamma = eta*|X|
%          N: the maximum number of iterations
% Output:  p = [a,b], where y= a+b*x

%%-----------
[m,n] = size(X);
if n>m, X=X'; [m,n] = size(X); end

for iter = 1:N
    % step 1
    s1 = randi([1 m]); s2 = randi([1 m]);
    while s1==s2, s2 = randi([1 m]); end
```

```
    S = [X(s1,:);X(s2,:)];
    % step 2
    p = get_hypothesis_WLS(S,[1;1]);
    % step 3
    C = inlier(X,p,tau_e);
    % step 4
    if sum(C)>=gamma
        p = get_hypothesis_WLS(X,C);
        break;
    end
end
```

6.6. Download or generate a simple color image of your own choice. Apply the method of structure tensor to identify rapid transitions such as edges. Plot the resulting **gradient magnitude** which is 'grad' returned from `structure_tensor.m`; you do not have to spend time to produce angle plots as in Figure 6.7.

CHAPTER 7

Gauss Elimination and Its Variants

One of the most frequently occurring problems, in all areas of scientific endeavor, is that of **solving a linear system** of n linear equations in n unknowns.

- The main subject of this chapter is to study **the use of Gauss elimination** to solve such systems.
- You will also study **finite difference methods**, for **boundary-value problems**, of which approximate solutions can be obtained by solving linear systems.

Contents of Chapter 7

7.1. Systems of Linear Equations 258
7.2. Solutions of Triangular Systems 264
7.3. Gauss Elimination 267
7.4. Gauss Elimination with Pivoting 274
Exercises for Chapter 7 280

7.1. Systems of Linear Equations

Note: Consider a system of n linear equations in n unknowns

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases} \tag{7.1}$$

Given the coefficients a_{ij} and the source b_i, we wish to find $[x_1,\, x_2,\, \cdots,\, x_n]$ which satisfy the equations.

- Since it is tedious to write (7.1) again and again, we generally prefer to write it as a single **matrix equation**

$$A\mathbf{x} = \mathbf{b}, \tag{7.2}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

Solvers for Linear Systems:

- **Direct algebraic solvers**
  - LU, LL^T, LDL^T, QR, SVD, SuperLU, $\cdots$ (factorization)
  - Harder to optimize and parallelize
  - Numerically robust, but higher algorithmic complexity
- **Iterative algebraic solvers** (Chapter 10)
  - Stationary and nonstationary methods
    (Jacobi, Gauss-Seidel, SOR, SSOR; CG, MINRES, GMRES, BiCG, QMR, $\cdots$)
  - Easier to optimize and parallelize
  - Low algorithmic complexity (**Convergence** is an issue)

7.1.1. Nonsingular matrices

Definition 7.1. (Definition 1.38) An $n \times n$ matrix A is said to be **invertible (nonsingular)** if there is an $n \times n$ matrix B such that

$$AB = I_n = BA, \tag{7.3}$$

where I_n is the identity matrix.

Note: In this case, B is the *unique inverse* of A denoted by A^{-1}.
(Thus $AA^{-1} = I_n = A^{-1}A$.)

Theorem 7.2. (Invertible Matrix Theorem; Theorem 1.43) *Let A be an $n \times n$ matrix. Then the following are equivalent.*

a. ***A is an invertible matrix.***
b. *A is row equivalent to the $n \times n$ identity matrix.*
c. *A has n pivot positions.*
d. ***The columns of A are linearly independent.***
e. ***The equation*** $A\mathrm{x} = 0$ ***has only the trivial solution*** $\mathrm{x} = 0$.
f. ***The equation*** $A\mathrm{x} = \mathrm{b}$ ***has unique solution for each*** $\mathrm{b} \in \mathbb{R}^n$.
g. *The linear transformation $\mathrm{x} \mapsto A\mathrm{x}$ is one-to-one.*
h. *The linear transformation $\mathrm{x} \mapsto A\mathrm{x}$ maps $\mathbb{R}^n$ onto $\mathbb{R}^n$.*
i. *There is a matrix $C \in \mathbb{R}^{n\times n}$ such that $CA = I$*
j. *There is a matrix $D \in \mathbb{R}^{n\times n}$ such that $AD = I$*
k. *A^T is invertible and $(A^T)^{-1} = (A^{-1})^T$.*
l. ***The number*** 0 ***is not an eigenvalue of A.***
m. ***det*** $A \neq 0$.

Example 7.3. Let $A \in \mathbb{R}^{n\times n}$ and eigenvalues of A be λ_i, $i = 1, 2, \cdots, n$. Show that

$$det\,(A) = \prod_{i=1}^{n} \lambda_i. \tag{7.4}$$

Thus we conclude that A is singular if and only if 0 is an eigenvalue of A.

Hint: Consider the characteristic polynomial of A, $\phi(\lambda) = det\,(A - \lambda I)$, and $\phi(0)$. See Remark 1.52.

7.1.2. Numerical Solutions of Differential Equations

Model Problem: Consider the following differential equation:

$$\begin{aligned} &\text{(a)} \quad -u_{xx} + cu &&= f, \quad x \in (a_x, b_x), \\ &\text{(b)} \quad -u_x + \beta u &&= g_a, \quad x = a_x, \\ &\text{(c)} \quad u_x + \beta u &&= g_b, \quad x = b_x. \end{aligned} \tag{7.5}$$

where

$$c \geq 0 \text{ and } \beta \geq 0 \quad (c + \beta > 0).$$

Numerical Discretization

- **(Partition)** Select n_x equally spaced grid points on the interval $[a_x, b_x]$:

$$x_i = a_x + ih_x, \quad i = 0, 1, \cdots, n_x, \quad h_x = \frac{b_x - a_x}{n_x}.$$

  Let $u_i = u(x_i)$, for $i = 0, 1, \cdots, n_x$.

- **(Finite Difference Method)** It follows from the **Taylor series** that

$$-u_{xx}(x_i) = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h_x^2} + \frac{u_{xxxx}(x_i)}{12} h_x^2 + \cdots,$$

  which we studied in Chapter 4. Thus the central second-order **finite difference** (FD) scheme for u_{xx} at x_i reads

$$-u_{xx}(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h_x^2}. \tag{7.6}$$

  Basis functions are the second Lagrange polynomials; see Section 4.1.

- **(Assembly – Algebraic System)** Apply the FD scheme for (7.5.a) to have

$$-u_{i-1} + (2 + h_x^2 c)u_i - u_{i+1} = h_x^2 f_i, \quad i = 0, 1, \cdots, n_x. \tag{7.7}$$

- However, we will meet ghost grid values at the end points. For example, at the point $a_x = x_0$, the formula becomes

$$-u_{-1} + (2 + h_x^2 c)u_0 - u_1 = h_x^2 f_0. \tag{7.8}$$

  Here the value u_{-1} is not defined and we call it a **ghost grid value**.

- Now, let's replace the ghost grid value u_{-1} by using the boundary condition (7.5.b). The central FD scheme for u_x at x_0 can be formulated as

$$u_x(x_0) \approx \frac{u_1 - u_{-1}}{2h_x}, \quad \textbf{Trunc.Err} = -\frac{u_{xxx}(x_0)}{6}h_x^2 + \cdots . \tag{7.9}$$

  Thus the equation (7.5.b), $\boldsymbol{-u_x + \beta u = g_a}$, can be approximated (at x_0)

$$u_{-1} + 2h_x\beta u_0 - u_1 = 2h_x g_a. \tag{7.10}$$

- Hence it follows from (7.8) and (7.10) that

$$(2 + h_x^2 c + 2h_x\beta)u_0 - 2u_1 = h_x^2 f_0 + 2h_x g_a. \tag{7.11}$$

  The same can be considered for the algebraic equation at the point x_n.

Scheme 7.4. The problem (7.5) is reduced to finding the solution $\mathbf{u}$ satisfying

$$A\mathbf{u} = \mathbf{b}, \tag{7.12}$$

where $A \in \mathbb{R}^{(n_x+1)\times(n_x+1)}$,

$$A = \begin{bmatrix} 2 + h_x^2 c + 2h_x\beta & -2 & & & & \\ -1 & 2 + h_x^2 c & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 2 + h_x^2 c & -1 & \\ & & & -2 & 2 + h_x^2 c + 2h_x\beta \end{bmatrix},$$

and

$$\mathbf{b} = \begin{bmatrix} h_x^2 f_0 \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ h_x^2 f_{n_x} \end{bmatrix} + \begin{bmatrix} 2h_x g_a \\ 0 \\ \vdots \\ 0 \\ 2h_x g_b \end{bmatrix}.$$

Definition 7.5. Such a technique of removing ghost grid values is called **outer bordering**.

Dirichlet Boundary Condition:

Formulation 7.6. When the boundary values of the DE are known (**Dirichlet boundary condition**), the algebraic system does not have to include rows corresponding to the nodal points.

- However, it is more reusable if the algebraic system incorporates rows for all nodal points.
- For example, consider

$$\begin{array}{lrcll} \text{(a)} & -u_{xx} + cu & = & f, & x \in (a_x, b_x), \\ \text{(b)} & -u_x + \beta u & = & g_a, & x = a_x, \\ \text{(c)} & u & = & u_b, & x = b_x. \end{array} \tag{7.13}$$

- Then, the corresponding algebraic system can be formulated as

$$A' \mathbf{u} = \mathbf{b}', \tag{7.14}$$

where $A' \in \mathbb{R}^{(n_x+1)\times(n_x+1)}$,

$$A' = \begin{bmatrix} 2 + h_x^2 c + 2h_x\beta & -2 & & & \\ -1 & 2 + h_x^2 c & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 + h_x^2 c & -1 \\ & & & 0 & 1 \end{bmatrix},$$

and

$$\mathbf{b}' = \begin{bmatrix} h_x^2 f_0 \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ 0 \end{bmatrix} + \begin{bmatrix} 2h_x g_a \\ 0 \\ \vdots \\ 0 \\ u_b \end{bmatrix} = \begin{bmatrix} h_x^2 f_0 + 2h_x g_a \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ u_b \end{bmatrix}.$$

Note: In the above formulation, each and every point corresponds to a row in the algebraic system. It involves only central FD schemes.

Alternative-Formulation **7.7.** *Again, consider (7.13):*

$$\begin{array}{lrcll} \text{(a)} & -u_{xx}+cu & = & f, & x\in(a_x,b_x), \\ \text{(b)} & -u_x+\beta u & = & g_a, & x=a_x, \\ \text{(c)} & u & = & u_b, & x=b_x. \end{array} \tag{7.15}$$

- *For $i=0,1,\cdots,n_x-1$, the same schemes in* **Formulation 7.6** *would apply.*
- *At x_{n_x-1} ($i=n_x-1$), (7.7) reads*

$$-u_{n_x-2}+(2+h_x^2c)u_{n_x-1}-u_{n_x}=h_x^2f_{n_x-1}. \tag{7.16}$$

Since $u_{n_x}=u(b_x)=u_b$, we have

$$-u_{n_x-2}+(2+h_x^2c)u_{n_x-1}=h_x^2f_{n_x-1}+u_b. \tag{7.17}$$

- *Thus the corresponding algebraic system can be formulated as*

$$A''\,\mathbf{u}=\mathbf{b}'', \tag{7.18}$$

where $A''\in\mathbb{R}^{n_x\times n_x}$,

$$A''=\begin{bmatrix} 2+h_x^2c+2h_x\beta & -2 & & & & \\ -1 & 2+h_x^2c & -1 & & & \\ & \ddots & \ddots & \ddots & & \\ & & -1 & 2+h_x^2c & -1 & \\ & & & -1 & 2+h_x^2c & \end{bmatrix},$$

and

$$\mathbf{b}''=\begin{bmatrix} h_x^2f_0 \\ h_x^2f_1 \\ \vdots \\ h_x^2f_{n_x-2} \\ h_x^2f_{n_x-1} \end{bmatrix}+\begin{bmatrix} 2h_xg_a \\ 0 \\ \vdots \\ 0 \\ u_b \end{bmatrix}=\begin{bmatrix} h_x^2f_0+2h_xg_a \\ h_x^2f_1 \\ \vdots \\ h_x^2f_{n_x-2} \\ h_x^2f_{n_x-1}+u_b \end{bmatrix}.$$

Note: In the above alternative formulation, the Dirichlet point does not introduce an *explicit* row in the algebraic system.

7.2. Solutions of Triangular Systems

Definition 7.8.

(a) A matrix $L = (\ell_{ij}) \in \mathbb{R}^{n\times n}$ is **lower triangular** if

$$\ell_{ij} = 0 \text{ whenever } i < j.$$

(b) A matrix $U = (u_{ij}) \in \mathbb{R}^{n\times n}$ is **upper triangular** if

$$u_{ij} = 0 \text{ whenever } i > j.$$

Theorem 7.9. *Let G be a triangular matrix. Then G is nonsingular if and only if $g_{ii} \neq 0$ for $i = 1, \cdots, n$.*

Lower-triangular Systems

Consider the $n \times n$ system

$$L\,\mathbf{y} = \mathbf{b}, \tag{7.19}$$

where L is a nonsingular, **lower-triangular matrix** ($\ell_{ii} \neq 0$). It is easy to see how to solve this system if we write it in detail:

$$\begin{aligned}
\ell_{11}\,y_1 &= b_1 \\
\ell_{21}\,y_1 + \ell_{22}\,y_2 &= b_2 \\
\ell_{31}\,y_1 + \ell_{32}\,y_2 + \ell_{33}\,y_3 &= b_3 \\
&\vdots \\
\ell_{n1}\,y_1 + \ell_{n2}\,y_2 + \ell_{n3}\,y_3 + \cdots + \ell_{nn}\,y_n &= b_n
\end{aligned} \tag{7.20}$$

The first equation involves only the unknown y_1, which can be found as

$$y_1 = b_1/\ell_{11}. \tag{7.21}$$

With y_1 just obtained, we can determine y_2 from the second equation:

$$y_2 = (b_2 - \ell_{21}\,y_1)/\ell_{22}. \tag{7.22}$$

Now with y_2 known, we can solve the third equation for y_3, and so on.

Algorithm 7.10. In general, once we have $y_1, y_2, \cdots, y_{i-1}$, we can solve for y_i using the ith equation:

$$\begin{aligned} y_i &= (b_i - \ell_{i1}\, y_1 - \ell_{i2}\, y_2 - \cdots - \ell_{i,i-1}\, y_{i-1})/\ell_{ii} \\ &= \frac{1}{\ell_{ii}}\Big(b_i - \sum_{j=1}^{i-1} \ell_{ij}\, y_j\Big) \end{aligned} \tag{7.23}$$

Matlab-code 7.11. (Forward Substitution/Elimination):

```
for i=1:n
    for j=1:i-1
        b(i) = b(i)-L(i,j)*b(j)
    end
    if L(i,i)==0, error('L: singular!'); end
    b(i) = b(i)/L(i,i)
end
```
(7.24)

The result is y.

Computational complexity: For each i, the forward substitution requires $2(i-1)+1$ flops. Thus the total number of flops becomes

$$\sum_{i=1}^{n}\{2(i-1)+1\} = \sum_{i=1}^{n}\{2i-1\} = n(n+1) - n = n^2. \tag{7.25}$$

† A **flop** is a **floating point operation** by $+$, $-$, $\times$ or $\div$.

Upper triangular Systems

Consider the system

$$U\,\mathbf{x} = \mathbf{y}, \tag{7.26}$$

where $U = (u_{ij}) \in \mathbb{R}^{n\times n}$ is a nonsingular, **upper triangular matrix**. Writing it out in detail, we get

$$\begin{aligned} u_{11}\,x_1 + u_{12}\,x_2 + \cdots + u_{1,n-1}\,x_{n-1} + u_{1,n}\,x_n &= y_1 \\ u_{22}\,x_2 + \cdots + u_{2,n-1}\,x_{n-1} + u_{2,n}\,x_n &= y_2 \\ \vdots \quad = \quad \vdots & \\ u_{n-1,n-1}\,x_{n-1} + u_{n-1,n}\,x_n &= y_{n-1} \\ u_{n,n}\,x_n &= y_n \end{aligned} \tag{7.27}$$

It is clear that we should solve the system from bottom to top.

Matlab-code 7.12. (Back Substitution):

```
for i=n:-1:1
    if(U(i,i)==0), error('U: singular!'); end
    x(i)=b(i)/U(i,i);
    b(1:i-1)=b(1:i-1)-U(1:i-1,i)*x(i);
end
```
(7.28)

Computational complexity: $n^2 + \mathcal{O}(n)$ flops.

7.3. Gauss Elimination

— a very basic algorithm for solving $A\mathrm{x} = \mathrm{b}$

The algorithms developed here ***produce*** (in the absence of rounding errors) ***the unique solution*** of $A\mathrm{x} = \mathrm{b}$, whenever $A \in \mathbb{R}^{n\times n}$ is nonsingular.

Strategy **7.13. (Gauss Elimination)**:

- First, transform the system $A\mathrm{x} = \mathrm{b}$ to an equivalent system $U\mathrm{x} = \mathrm{y}$, where U is upper triangular (**Forward Elimination**).
- Then transform the system $U\mathrm{x} = \mathrm{y}$ to get x (**Back Substitution**).
- It is convenient to represent $A\mathrm{x} = \mathrm{b}$ by an **augmented matrix** $[A|\mathrm{b}]$.
- **Transformation of the system**: By means of three **elementary row operations**, applied on the augmented matrix.

Definition **7.14. Elementary row operations** (EROs).

$$\begin{array}{ll} \text{Replacement:} & R_i \leftarrow R_i + \alpha R_j \ \ (i \neq j) \\ \text{Interchange:} & R_i \leftrightarrow R_j \\ \text{Scaling:} & R_i \leftarrow \beta R_i \ \ (\beta \neq 0) \end{array} \tag{7.29}$$

Proposition **7.15.**

(a) If $[\hat{A}\,|\,\hat{\mathrm{b}}]$ is obtained from $[A\,|\,\mathrm{b}]$ by elementary row operations (EROs), then systems $[A\,|\,\mathrm{b}]$ and $[\hat{A}\,|\,\hat{\mathrm{b}}]$ represent the same solution.

(b) Suppose $\hat{A}$ is obtained from A by EROs. Then $\hat{A}$ is nonsingular if and only if A is.

(c) Each ERO corresponds to left-multiple of an **elementary matrix**.

(d) Each elementary matrix is nonsingular.

(e) The elementary matrices corresponding to “Replacement" and “Scaling" operations are lower triangular.

The LU Factorization/Decomposition

The LU factorization is motivated by the **fairly common industrial and business problem** of solving **a sequence of equations**, all with the same coefficient matrix:

$$A\mathbf{x} = \mathbf{b}_1,\ A\mathbf{x} = \mathbf{b}_2,\ \cdots,\ A\mathbf{x} = \mathbf{b}_p. \tag{7.30}$$

Definition 7.16. Let $A \in \mathbb{R}^{m\times n}$. The **LU factorization** of A is $A = LU$, where $L \in \mathbb{R}^{m\times m}$ is a *unit lower triangular matrix* and $U \in \mathbb{R}^{m\times n}$ is an echelon form of A (upper triangular matrix):

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U}$$

Remark 7.17. Let $A\mathbf{x} = \mathbf{b}$ be to be solved. Then $A\mathbf{x} = LU\,\mathbf{x} = \mathbf{b}$ and it can be solved as

$$\begin{cases} L\mathbf{y} &= \mathbf{b}, \\ U\mathbf{x} &= \mathbf{y}, \end{cases} \tag{7.31}$$

each algebraic equation can be solved effectively, via substitutions.

Algorithm 7.18. (An LU Factorization Algorithm) *The derivation introduces an LU factorization: Let $A \in \mathbb{R}^{m\times n}$. Then*

$$\begin{aligned} A &= I_m A \\ &= I_m E_1^{-1} E_1 A \\ &= I_m E_1^{-1} E_2^{-1} E_2 E_1 A = (E_2 E_1)^{-1} E_2 E_1 A \\ &= \quad \vdots \\ &= I_m E_1^{-1} E_2^{-1} \cdots E_p^{-1} \underbrace{E_p \cdots E_2 E_1 A}_{\text{an echelon form}} = \underbrace{(E_p \cdots E_1)^{-1}}_{L} \underbrace{E_p \cdots E_1 A}_{U}, \end{aligned} \tag{7.32}$$

where E_i are elementary matrices for "replacement".

Example **7.19.** Find the LU factorization of

$$A = \begin{bmatrix} 4 & 3 & -5 \\ -4 & -5 & 7 \\ 8 & 8 & -7 \end{bmatrix}.$$

Solution.

Ans: $A = LU = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & -1 & 1 \end{bmatrix} \begin{bmatrix} 4 & 3 & -5 \\ 0 & -2 & 2 \\ 0 & 0 & 5 \end{bmatrix}$.

Theorem **7.20.** **(LU Decomposition Theorem)**
Let $A \in \mathbb{R}^{n\times n}$. The following are equivalent.

1. *All leading principal submatrices of A are nonsingular. (The jth leading principal submatrix is $A(1:j, 1:j)$.)*
2. *There exists a unique unit lower triangular L and nonsingular upper triangular U such that $A = LU$.*

Proof. **(2)** $\Rightarrow$ **(1)**: $A = LU$ may also be written

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & \mathbf{0} \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ \mathbf{0} & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix} \quad (7.33)$$

where A_{11} is a $j \times j$ leading principal submatrix. Thus

$$\det(A_{11}) = \det(L_{11}U_{11}) = 1 \cdot \det(U_{11}) = \prod_{k=1}^{j} (U_{11})_{kk} \neq 0.$$

Here we have used the assumption that U is nonsingular and so is U_{11}.

(1) $\Rightarrow$ **(2)**: It can be proved by induction on n. □

Example 7.21. Find the LU factorization of $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$.

Solution. (Practical Implementation):

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + 2R_1]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} \mathbf{3} & -1 & 1 \\ \mathbf{3} & 4 & -1 \\ \mathbf{-2} & 3 & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \frac{3}{4}R_2} \begin{bmatrix} \mathbf{3} & -1 & 1 \\ \mathbf{3} & 4 & -1 \\ \mathbf{-2} & \frac{3}{4} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ -2 & \frac{3}{4} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

* It is easy to verify that $A = LU$.

Example 7.22. *Find the LU factorization of* $A = \begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix}$.

Solution.

$$\begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix} \xrightarrow[R_4 \leftarrow R_4 - 6R_1]{R_2 \leftarrow R_2 - 3R_1, \; R_3 \leftarrow R_3 + 5R_1} \begin{bmatrix} \mathbf{2} & -1 \\ \mathbf{3} & 8 \\ -5 & -2 \\ \mathbf{6} & 4 \end{bmatrix} \xrightarrow[R_4 \leftarrow R_4 - \frac{1}{2}R_2]{R_3 \leftarrow R_3 + \frac{1}{4}R_2} \begin{bmatrix} \mathbf{2} & -1 \\ \mathbf{3} & 8 \\ -5 & -\frac{1}{4} \\ \mathbf{6} & \frac{1}{2} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{3} & 1 & 0 & 0 \\ \mathbf{-5} & -\frac{1}{4} & 1 & 0 \\ \mathbf{6} & \frac{1}{2} & 0 & 1 \end{bmatrix}_{4\times 4}, \quad U = \begin{bmatrix} 2 & -1 \\ 0 & 8 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{4\times 2}.$$

Note: U has the same size as A, i.e,, $A \in \mathbb{R}^{4\times 2}$, while L is a square matrix and is a unit lower triangular matrix.

Remark **7.23.** **For an $n \times n$ dense matrix A** (with most entries nonzero) with n moderately large.

- Computing an LU factorization of A takes about $2n^3/3$ flops† ($\sim$ row reducing $[A\ \mathbf{b}]$), while finding A^{-1} requires about $2n^3$ flops.
- Solving $L\mathbf{y} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{y}$ requires about $2n^2$ flops, because any $n \times n$ triangular system can be solved in about n^2 flops.
- Multiplying $\mathbf{b}$ by A^{-1} also requires about $2n^2$ flops, but the result may not as accurate as that obtained from L and U (due to round-off errors in computing A^{-1} & $A^{-1}\mathbf{b}$).

If A is sparse (with mostly zero entries), then L and U may be sparse, too. On the other hand, A^{-1} is likely to be **dense**.

- In this case, a solution of $A\mathbf{x} = \mathbf{b}$ with LU factorization is much faster than using A^{-1}.

Summary **7.24.** **Solving Linear Systems by LU Factorization**

- The LU factorization can be applied for general $m \times n$ matrices:

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U} \tag{7.34}$$

- Let $A \in \mathbb{R}^{n\times n}$ be nonsingular. If $A = LU$, where L is a **unit lower-triangular matrix** and U is an **upper triangular matrix**, then

$$A\mathbf{x} = \mathbf{b} \iff (LU)\mathbf{x} = L(U\mathbf{x}) = \mathbf{b} \iff \begin{cases} L\mathbf{y} = \mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{cases} \tag{7.35}$$

In the following couple of examples, LU is given.

Example 7.25. Let $A = \begin{bmatrix} 1 & 4 & -2 \\ 2 & 5 & -3 \\ -3 & -18 & 16 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -12 \\ -14 \\ 64 \end{bmatrix}$;

$$A = LU \triangleq \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & -2 \\ 0 & -3 & 1 \\ 0 & 0 & 8 \end{bmatrix}.$$

Use the LU factorization of A to solve $A\mathbf{x} = \mathbf{b}$.

Solution. From (7.35), we know there are two steps to perform:

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$;
(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ by row reduction

$$[L \vdots \mathbf{b}] = \left[\begin{array}{ccc:c} 1 & 0 & 0 & -12 \\ 2 & 1 & 0 & -14 \\ -3 & 2 & 1 & 64 \end{array}\right] \to \cdots \to \left[\begin{array}{ccc:c} 1 & 0 & 0 & -12 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 8 \end{array}\right] = [I \vdots \mathbf{y}]. \quad (7.36)$$

(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by row reduction

$$[U \vdots \mathbf{y}] = \left[\begin{array}{ccc:c} 1 & 4 & -2 & -12 \\ 0 & -3 & 1 & 10 \\ 0 & 0 & 8 & 8 \end{array}\right] \to \cdots \to \left[\begin{array}{ccc:c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 1 \end{array}\right] = [I \vdots \mathbf{x}]. \quad (7.37)$$

Thus, $\mathbf{x} = [2, -3, 1]^T$. □

Example 7.26. Let $A = \begin{bmatrix} 5 & 4 & -2 & -3 \\ 15 & 13 & 2 & -10 \\ -5 & -1 & 28 & 3 \\ 10 & 10 & 8 & -8 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -10 \\ -29 \\ 30 \\ -22 \end{bmatrix}$;

$$A = LU \triangleq \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 \\ 2 & 2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 5 & 4 & -2 & -3 \\ 0 & 1 & 8 & -1 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 6 \end{bmatrix}.$$

Use the LU factorization of A to solve $A\mathbf{x} = \mathbf{b}$.

Solution.

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$:

$$[L \vdots \mathbf{b}] = \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & -10 \\ 3 & 1 & 0 & 0 & -29 \\ -1 & 3 & 1 & 0 & 30 \\ 2 & 2 & -2 & 1 & -22 \end{array}\right] \to \cdots \to \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & -10 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 17 \\ 0 & 0 & 0 & 1 & 30 \end{array}\right] = [I \vdots \mathbf{y}].$$

(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$:

$$[U \vdots \mathbf{y}] = \left[\begin{array}{cccc:c} 5 & 4 & -2 & -3 & -10 \\ 0 & 1 & 8 & -1 & 1 \\ 0 & 0 & 2 & 3 & 17 \\ 0 & 0 & 0 & 6 & 30 \end{array}\right] \to \cdots \to \left[\begin{array}{cccc:c} 1 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & -2 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 5 \end{array}\right] = [I \vdots \mathbf{x}].$$

Thus, $\mathbf{x} = [3, -2, 1, 5]^T$. □

7.4. Gauss Elimination with Pivoting

Definition 7.27. A **permutation matrix** is a matrix that has exactly one 1 in each row and in each column, all other entries being zero.

Thus a permutation matrix can be obtained by ***scrambling rows of the identity matrix I***.

Self-study 7.28. Prove that if P is a permutation matrix, then

$$P^T P = P P^T = I.$$

Thus P is nonsingular and $P^{-1} = P^T$; it is an **orthogonal matrix**.

Solution. Let $P = [\mathbf{p}_1\ \mathbf{p}_2\ \cdots\ \mathbf{p}_n]$. Then you may check that $\mathbf{p}_i \cdot \mathbf{p}_j = \delta_{ij}$.

Claim 7.29.
Let P and Q be $n \times n$ **permutation matrices** and $A \in \mathbb{R}^{n\times n}$. Then

(a) PA is A with its rows permuted
AP is A with its columns permuted.

(b) $det(P) = \pm 1$.

(c) PQ is also a permutation matrix.

Example 7.30. Let $A \in \mathbb{R}^{n\times n}$, and let $\widehat{A}$ be a matrix obtained from scrambling the rows of A. Show that there is a unique permutation matrix $P \in \mathbb{R}^{n\times n}$ such that $\widehat{A} = PA$.

Hint: Consider the row indices in the scrambled matrix $\widehat{A}$, say $\{k_1, k_2, \cdots, k_n\}$. (This means that for example, the first row of $\widehat{A}$ is the same as the k_1-th row of A.) Use the index set to define a permutation matrix P.

Proof. (Self-study)

Theorem 7.31. ***Gauss elimination with partial pivoting****, applied to $A \in \mathbb{R}^{n\times n}$, produces a unit lower-triangular matrix L with $|\ell_{ij}| \le 1$, an upper triangular matrix U, and a permutation matrix P such that*

$$\widehat{A} = PA = LU \tag{7.38}$$

or, equivalently,

$$A = P^T LU \tag{7.39}$$

Note: If A is singular, then so is U.

- Matlab: `[L,U,P] = lu(A)`
- Python: `P,L,U = scipy.linalg.lu(A)`

Algorithm 7.32.
Solving $A\mathrm{x} = \mathrm{b}$ using *Gauss elimination with partial pivoting*:

1. Factorize A into $A = P^T LU$, where
   P = permutation matrix,
   L = unit lower triangular matrix
   (i.e., with ones on the diagonal),
   U = nonsingular upper triangular matrix.
2. Solve $A\mathrm{x} = P^T LU\mathrm{x} = \mathrm{b}$; in practice,
   (a) $LU\mathrm{x} = P\mathrm{b}$ (permuting b)
   (b) $U\mathrm{x} = L^{-1}(P\mathrm{b})$ (forward substitution)
   (c) $\mathrm{x} = U^{-1}(L^{-1}P\mathrm{b})$ (back substitution)

Theorem 7.33. *If A is nonsingular, then there exist permutations P_1 and P_2, a unit lower triangular matrix L, and a nonsingular upper triangular matrix U such that*

$$P_1 A P_2 = LU,$$

where only one of P_1 and P_2 is necessary.

Note: $P_1 A$ reorders the *rows* of A, AP_2 reorders the *columns*, and $P_1 A P_2$ reorders both.

Remark 7.34. Partial Pivoting
Consider

$$\begin{aligned} P_1' A P_2' &= \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} \\ L_{21} & I \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ \mathbf{0} & \tilde{A}_{22} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & U_{12} \\ L_{21} u_{11} & L_{21} U_{12} + \tilde{A}_{22} \end{bmatrix} \end{aligned} \tag{7.40}$$

- We can choose $P_2' = I$ and P_1' so that **a_{11} is the largest entry in absolute value in its column**, which implies $L_{21} = \frac{A_{21}}{a_{11}}$ has entries bounded by 1 in modulus.
- More generally, at step k of Gaussian elimination, where we are computing the kth column of L, we reorder the rows so that ***the largest entry in the column is on the pivot.*** This is called **Gaussian elimination with partial pivoting**, or **GEPP** for short. GEPP guarantees that all entries of L are bounded by one in modulus.

Remark 7.35. Complete Pivoting
We can choose P_1 and P_2 so that **a_{11} in (7.40) is the largest entry in modulus in the whole matrix.** More generally, at step k of Gaussian elimination, we reorder the rows and columns so that the largest entry in the matrix is on the pivot. This is called **Gaussian elimination with complete pivoting**, or **GECP** for short.

Example 7.36. Find the LU factorization of $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$, which is considered in Example 7.21.

Solution. **(Without pivoting)**

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + 2R_1]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} 3 & -1 & 1 \\ \mathbf{3} & 4 & -1 \\ \mathbf{-2} & 3 & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \frac{3}{4}R_2} \begin{bmatrix} 3 & -1 & 1 \\ \mathbf{3} & 4 & -1 \\ \mathbf{-2} & \frac{3}{4} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ -2 & \frac{3}{4} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

(With partial pivoting)

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow{\mathbf{R_1 \leftrightarrow R_2}} \begin{bmatrix} \boxed{9} & 1 & 2 \\ 3 & -1 & 1 \\ -6 & 5 & -5 \end{bmatrix}$$

$$\xrightarrow[R_3 \leftarrow R_3 + \frac{2}{3}R_1]{R_2 \leftarrow R_2 - \frac{1}{3}R_1} \begin{bmatrix} \boxed{9} & 1 & 2 \\ \frac{1}{3} & -\frac{4}{3} & \frac{1}{3} \\ -\frac{2}{3} & \frac{17}{3} & -\frac{11}{3} \end{bmatrix} \xrightarrow{\mathbf{R_2 \leftrightarrow R_3}} \begin{bmatrix} \boxed{9} & 1 & 2 \\ -\frac{2}{3} & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ \frac{1}{3} & -\frac{4}{3} & \frac{1}{3} \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 + \frac{4}{17}R_2} \begin{bmatrix} \boxed{9} & 1 & 2 \\ -\frac{2}{3} & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ \frac{1}{3} & -\frac{4}{17} & -\frac{9}{17} \end{bmatrix}, \quad I \xrightarrow{\mathbf{R_1 \leftrightarrow R_2}} E \xrightarrow{\mathbf{R_2 \leftrightarrow R_3}} P$$

$$PA = LU$$

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{2}{3} & 1 & 0 \\ \frac{1}{3} & -\frac{4}{17} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \boxed{9} & 1 & 2 \\ 0 & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ 0 & 0 & \boxed{-\frac{9}{17}} \end{bmatrix}$$

Implementation of LU

LU_pivot.m

```
function [L, U, P] = LU_pivot(A)
% function [L, U, P] = LU_pivot(A)
    [m, n] = size(A);
    L=eye(n); P=eye(n); U=A;
    for k=1:m-1
        [~,r]=max(abs(U(k:m,k)));
        r = r+(k-1);
        L([k,r],1:k-1)=L([r,k],1:k-1)
        U([k,r],k:m)  =U([r,k],k:m)
        P([k,r],:)    =P([r,k],:)
        for j=k+1:m
            L(j,k)=U(j,k)/U(k,k)
            U(j,k:m)=U(j,k:m)-L(j,k)*U(k,k:m)
        end
    end
```

call_LU_pivot.m

```
A = [ 1 2 3; 2 5 6; 4 8 9];
[L, U, P] = LU_pivot(A)
fprintf("P'*L*U =\n")
disp(P'*L*U)
```

Output

```
L =
    1.0000         0         0
    0.5000    1.0000         0
    0.2500         0    1.0000
U =
    4.0000    8.0000    9.0000
         0    1.0000    1.5000
         0         0    0.7500
P =
     0     0     1
     0     1     0
     1     0     0
P'*L*U =
     1     2     3
     2     5     6
     4     8     9
```

Computation of A^{-1}

Algorithm 7.37. (Computation of A^{-1}):

- The program to solve $Ax = b$ can be used to calculate the inverse of a matrix. Letting $X = A^{-1}$, we have

$$AX = I. \tag{7.41}$$

- This equation can be written in partitioned form:

$$A[\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n] = [\mathbf{e}_1\ \mathbf{e}_2\ \cdots\ \mathbf{e}_n], \tag{7.42}$$

  where $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n$ and $\mathbf{e}_1, \mathbf{e}_2, \cdots, \mathbf{e}_n$ are columns of X and I, respectively.
- Thus $AX = I$ is equivalent to the n equations

$$A\mathbf{x}_i = \mathbf{e}_i, \quad i = 1, 2, \cdots, n. \tag{7.43}$$

  Solving these n systems by Gauss elimination with partial pivoting, we obtain A^{-1}.

Computational complexity

- **A naive flop count**:

LU-factorization of A:	$\frac{2}{3}n^3 + \mathcal{O}(n^2)$
Solve for n equations in (7.43):	$n \cdot 2n^2 = 2n^3$
Total cost:	$\frac{8}{3}n^3 + \mathcal{O}(n^2)$

- **A modification**: The forward-substitution phase requires the solution of

$$L\mathbf{y}_i = \mathbf{e}_i, \quad i = 1, 2, \cdots, n. \tag{7.44}$$

  Some operations can be saved by exploiting the leading zeros in e_i. (For each i, the portion of L to be accessed is triangular.) With these savings, one can conclude that A^{-1} can be computed in $2n^3 + \mathcal{O}(n^2)$ flops.

Exercises for Chapter 7

7.1. Solve the equation $A\mathrm{x} = \mathrm{b}$ by using the LU factorization.

$$A = \begin{bmatrix} 4 & 3 & -5 \\ -4 & -5 & 7 \\ 8 & 6 & -8 \end{bmatrix} \text{ and } \mathrm{b} = \begin{bmatrix} 2 \\ -4 \\ 6 \end{bmatrix}.$$

(Do not use computer programming for this problem.)

Ans: $\mathrm{x} = \begin{bmatrix} 1/4 \\ 2 \\ 1 \end{bmatrix}$.

7.2. Let $L = [\ell_{ij}]$ and $M = [m_{ij}]$ be lower-triangular matrices.

(a) Prove that LM is lower triangular.
(b) Prove that the entries of the main diagonal of LM are

$$\ell_{11}m_{11},\ \ell_{22}m_{22},\ \cdots,\ \ell_{nn}m_{nn}$$

Thus the product of two unit lower-triangular matrices is unit lower triangular.

7.3. C Consider the system $A\mathrm{x} = \mathrm{b}$, where

$$A = \begin{bmatrix} 1 & -2 & -1 & 3 \\ 1 & -2 & 0 & 1 \\ -3 & -2 & 1 & 7 \\ 0 & -2 & 8 & 5 \end{bmatrix} \text{ and } \mathrm{b} = \begin{bmatrix} -12 \\ -5 \\ -14 \\ -7 \end{bmatrix}.$$

(a) Perform LU decomposition with partial pivoting for A to show P, L, and U.
(b) Solve the system.

(You may use any built-in functions for this problem.)

7.4. C Consider the finite difference method on uniform meshes to solve

$$\begin{aligned} &\text{(a)} \quad -u_{xx} + u = (\pi^2 + 1)\cos(\pi x), \quad x \in (0, 1), \\ &\text{(b)} \quad u(0) = 1 \text{ and } u_x(1) = 0. \end{aligned} \tag{7.45}$$

(a) Implement a function to construct algebraic systems in the full matrix form, for general $n_x \geq 1$.
(b) Use a direct method (e.g., $A\backslash b$) to find approximate solutions for $n_x = 25,\ 50,\ 100$.
(c) The actual solution for (7.45) is $u(x) = \cos(\pi x)$. Measure the maximum errors for the approximate solutions.

Note: This problem is optional for **undergraduate students**; you will get an extra credit when you solve it. If you decide not to solve the problem, simply write ***"undergraduate"***.

CHAPTER 8

Eigenvalues and QR Factorization

In This Chapter:

Topics	Applications/Properties
Power Method	
Symmetric power method	Symmetric matrices
Inverse power method	
QR Factorization	
Gram-Schmidt process	
Householder reflectors	QR algorithm

Contents of Chapter 8

8.1. Preliminaries: Norms of Vectors and Matrices 282
8.2. The Power Method 287
8.3. QR Factorization by the Gram-Schmidt Process 297
8.4. Householder Reflectors 301
8.5. QR Factorization by the Householder Reflectors 305
8.6. QR Algorithm for Eigenvalues and Eigenvectors 312
8.7. Eigenvalues and Eigenvectors of Stochastic Matrices 315
Exercises for Chapter 8 321

8.1. Preliminaries: Norms of Vectors and Matrices

8.1.1. Vectors

Definition 8.1. Let $\mathbf{u} = [u_1, u_2, \cdots, u_n]^T$ and $\mathbf{v} = [v_1, v_2, \cdots, v_n]^T$ are vectors in $\mathbb{R}^n$. Then, the **inner product** (or **dot product**) of $\mathbf{u}$ and $\mathbf{v}$ is given by

$$\begin{aligned} \mathbf{u}\bullet\mathbf{v} &= \mathbf{u}^T\mathbf{v} = [u_1\ u_2\ \cdots\ u_n]\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \\ &= u_1\,v_1 + u_2\,v_2 + \cdots + u_n\,v_n = \sum_{k=1}^{n} u_k v_k. \end{aligned} \tag{8.1}$$

Definition 8.2. The **length** (**Euclidean norm**) of $\mathbf{v}$ is nonnegative scalar $\|\mathbf{v}\|$ defined by

$$\|\mathbf{v}\| = \sqrt{\mathbf{v}\bullet\mathbf{v}} = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \quad \text{and} \quad \|\mathbf{v}\|^2 = \mathbf{v}\bullet\mathbf{v}. \tag{8.2}$$

Definition 8.3. For $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, the **distance** between $\mathbf{u}$ and $\mathbf{v}$ is

$$dist(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|, \tag{8.3}$$

the length of the vector $\mathbf{u} - \mathbf{v}$.

Definition 8.4. Two vectors $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{R}^n$ are **orthogonal** if $\mathbf{u}\bullet\mathbf{v} = 0$.

Theorem 8.5. ***Pythagorean Theorem****: Two vectors* $\mathbf{u}$ *and* $\mathbf{v}$ *are orthogonal if and only if*

$$\|\mathbf{u} + \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2. \tag{8.4}$$

Note: The **inner product** can be defined as

$$\mathbf{u}\bullet\mathbf{v} = \|\mathbf{u}\|\,\|\mathbf{v}\|\cos\theta, \tag{8.5}$$

where θ is the angle between $\mathbf{u}$ and $\mathbf{v}$.

Self-study **8.6.** Let $\mathbf{u} = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix}$. Use (8.5) to find the angle between $\mathbf{u}$ and $\mathbf{v}$.

Solution.

8.1.2. Vector and matrix norms

Definition **8.7.** A **norm** (or, **vector norm**) on $\mathbb{R}^n$ is a function that assigns to each $x \in \mathbb{R}^n$ a nonnegative real number $\|x\|$ such that the following three properties are satisfied: for all $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$,

$$\begin{aligned} &\|\mathbf{x}\| > 0 \text{ if } \mathbf{x} \neq 0 && \text{(positive definiteness)} \\ &\|\lambda\mathbf{x}\| = |\lambda|\,\|\mathbf{x}\| && \text{(homogeneity)} \\ &\|\mathbf{x}+\mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| && \text{(triangle inequality)} \end{aligned} \tag{8.6}$$

Example **8.8.** The most vector common norms are the p**-norms**

$$\|\mathbf{x}\|_p = \Big(\sum_i |x_i|^p\Big)^{1/p}, \quad 1 \leq p < \infty, \tag{8.7}$$

and the **infinity-norm** or **maximum-norm**

$$\|\mathbf{x}\|_\infty = \max_i |x_i|. \tag{8.8}$$

Note: Two of frequently used p-norms are

$$\|\mathbf{x}\|_1 = \sum_i |x_i|, \quad \|\mathbf{x}\|_2 = \Big(\sum_i |x_i|^2\Big)^{1/2} \tag{8.9}$$

The ℓ^2**-norm** is also called the **Euclidean norm**, often denoted by $\|\cdot\|$.

Example 8.9. One may consider the infinity-norm as the limit of p-norms, as $p \to \infty$.

Solution.

Theorem 8.10. ***(Cauchy-Schwarz Inequality)***.

$$|\mathbf{x}\cdot\mathbf{y}| = \left|\sum_{i=1}^n x_i y_1\right| \le \Big(\sum_{i=1}^n x_i^2\Big)^{1/2}\Big(\sum_{i=1}^n y_i^2\Big)^{1/2} = \|\mathbf{x}\|\,\|\mathbf{y}\|. \tag{8.10}$$

Theorem 8.11. *The sequence of vectors* $\{\mathbf{x}^{(k)}\}$ *converges to* $\mathbf{x}$ *in* $\mathbb{R}^n$ *with respect to the* ℓ^∞**-norm** *if and only if*

$$\lim_{k\to\infty} x_i^{(k)} = x_i, \quad \textit{for all } \ i = 1, 2, \cdots, n.$$

Example 8.12. $\mathbf{x}^{(k)} = \Big(1,\ 2+\dfrac{1}{k},\ \dfrac{3}{k^2},\ e^{-k}\sin k\Big)^T \to (1,\ 2,\ 0,\ 0)^T$ with respect to the ℓ^∞-norm.

Solution.

Theorem 8.13. (Equivalence between the ℓ^2-norm and the ℓ^∞-norm). *For each* $\mathbf{x} \in \mathbb{R}^n$,

$$\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \sqrt{n}\|\mathbf{x}\|_\infty. \tag{8.11}$$

8.1.3. Matrix norms

Definition 8.14. A **matrix norm** on $m \times n$ matrices is a vector norm on the mn-dimensional space, satisfying

$$\begin{aligned} &\|A\| \geq 0, \text{ and } \|A\| = 0 \;\Leftrightarrow\; A = 0 && \text{(positive definiteness)} \\ &\|\lambda\, A\| = |\lambda|\, \|A\| && \text{(homogeneity)} \\ &\|A + B\| \leq \|A\| + \|B\| && \text{(triangle inequality)} \end{aligned} \tag{8.12}$$

Example 8.15. $\|A\|_F \equiv \Big(\sum_{i,j} |a_{ij}|^2\Big)^{1/2}$ is called the **Frobenius norm**.

Definition 8.16. *Once a vector norm $||\cdot||$ has been specified, the **induced matrix norm** is defined by*

$$\|A\| = \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{8.13}$$

It is also called an **operator norm** *or* **subordinate norm**.

Theorem 8.17.

a. For all operator norms and the Frobenius norm,

$$\|Ax\| \leq \|A\|\, \|x\|, \quad \|A\, B\| \leq \|A\|\, \|B\|. \tag{8.14}$$

b. $\|A\|_1 \equiv \max_{\mathbf{x}\neq 0} \dfrac{\|A\mathbf{x}\|_1}{\|\mathbf{x}\|_1} = \max_j \sum_i |a_{ij}|$ *(max of vertical sums)*

c. $\|A\|_\infty \equiv \max_{\mathbf{x}\neq 0} \dfrac{\|A\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} = \max_i \sum_j |a_{ij}|$ *(max of horizontal sums)*

d. $\|A\|_2 \equiv \max_{\mathbf{x}\neq 0} \dfrac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\lambda_{\max}(A^T A)}$,
where $\lambda_{\max}$ denotes the largest eigenvalue.

e. $\|A\|_2 = \|A^T\|_2$.

f. $\|A\|_2 = \max_i |\lambda_i(A)|$, *when* $A^T A = A A^T$ ***(normal matrix)***.

Definition **8.18.** *Let $A \in \mathbb{R}^{n\times n}$. Then*

$$\kappa(A) \equiv \|A\|\,\|A^{-1}\|$$

is called the ***condition number*** *of A, associated to the matrix norm.*

Example **8.19.** Let $A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & -1 \\ 1 & -2 & 1 \end{bmatrix}$. Then, we have

$$A^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 1 \\ -1 & 2 & 1 \\ -3 & 4 & 3 \end{bmatrix} \quad \text{and} \quad A^TA = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 17 & -7 \\ 0 & -7 & 3 \end{bmatrix}.$$

a. Find $\|A\|_1$, $\|A\|_\infty$, and $\|A\|_2$.

b. Compute the ℓ^1-condition number $\kappa_1(A)$.

Solution.

Ans: $\text{eig}(A^TA) \approx \{0.10051,\ 2,\ 19.89949\}$. $\|A\|_2 \approx \sqrt{19.89949} \approx 4.46088$.

Theorem **8.20. (Neumann Series Theorem)**. *If $A \in \mathbb{R}^{n\times n}$ and $\|A\| < 1$ for any subordinate matrix norm, then $I - A$ is invertible and*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k. \tag{8.15}$$

8.2. The Power Method

The **power method** is an eigenvalue algorithm:
Given a matrix $A \in \mathbb{R}^{n\times n}$, the algorithm will produce a number λ, which is the greatest (in absolute value) eigenvalue of A, and a nonzero vector $\mathbf{v}$, which is a corresponding eigenvector of λ ($A\mathbf{v} = \lambda\mathbf{v}$).

Assumption. To apply the power method, we assume that $A \in \mathbb{R}^{n\times n}$ has

- n eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$,
- n associated eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$, which are **linearly independent**, and
- **precisely one eigenvalue** that is largest in magnitude, λ_1:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|. \tag{8.16}$$

The power method approximates the largest eigenvalue λ_1 and its associated eigenvector $\mathbf{v}_1$.

8.2.1. Power iteration

- Since eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$ are linearly independent, any vector $\mathbf{x} \in \mathbb{R}^n$ can be expressed as

$$\mathbf{x} = \sum_{j=1}^{n} \beta_j \mathbf{v}_j, \tag{8.17}$$

  for some constants $\{\beta_1, \beta_2, \cdots, \beta_n\}$.

- Multiplying both sides of (8.17) by A and A^2 gives

$$\begin{aligned} A\mathbf{x} &= A\Big(\sum_{j=1}^{n} \beta_j \mathbf{v}_j\Big) = \sum_{j=1}^{n} \beta_j A\mathbf{v}_j = \sum_{j=1}^{n} \beta_j \lambda_j \mathbf{v}_j, \\ A^2\mathbf{x} &= A\Big(\sum_{j=1}^{n} \beta_j \lambda_j \mathbf{v}_j\Big) = \sum_{j=1}^{n} \beta_j \lambda_j^2 \mathbf{v}_j. \end{aligned} \tag{8.18}$$

- In general,

$$A^k\mathbf{x} = \sum_{j=1}^{n} \beta_j \lambda_j^k \mathbf{v}_j, \quad k = 1, 2, \cdots, \tag{8.19}$$

which gives, for $k = 1, 2, \cdots$,

$$A^k\mathbf{x} = \lambda_1^k \sum_{j=1}^{n} \beta_j \Big(\frac{\lambda_j}{\lambda_1}\Big)^k \mathbf{v}_j = \lambda_1^k \left(\beta_1 \Big(\frac{\lambda_1}{\lambda_1}\Big)^k \mathbf{v}_1 + \beta_2 \Big(\frac{\lambda_2}{\lambda_1}\Big)^k \mathbf{v}_2 + \cdots + \beta_n \Big(\frac{\lambda_n}{\lambda_1}\Big)^k \mathbf{v}_n\right). \tag{8.20}$$

- Since $|\lambda_1| > |\lambda_j|$, $2 \le j \le n$, we have $\lim_{k\to\infty} |\lambda_j/\lambda_1|^k = 0$, and

$$\lim_{k\to\infty} A^k\mathbf{x} = \lim_{k\to\infty} \lambda_1^k \beta_1 \mathbf{v}_1. \tag{8.21}$$

Remark 8.21. The sequence in (8.21) converges to 0 if $|\lambda_1| < 1$ and diverges if $|\lambda_1| > 1$, provided that $\beta_1 \neq 0$.

- The entries of $A^k\mathbf{x}$ will grow with k if $|\lambda_1| > 1$ and will go to 0 if $|\lambda_1| < 1$.
- In either case, it is hard to decide the largest eigenvalue λ_1 and its associated eigenvector $\mathbf{v}_1$.
- **Remedy**: We **scale $A^k\mathbf{x}$** in an appropriate manner, to ensure that the limit in (8.21) is finite and nonzero.

Algorithm 8.22. (The Power Iteration) Given $\mathbf{x} \neq 0$:

$$\begin{aligned}
&\textbf{initialization}: \quad \mathbf{x}^0 = \mathbf{x}/||\mathbf{x}||_\infty \\
&\textbf{for } k = 1, 2, \cdots \\
&\qquad \mathbf{y}^k = A\mathbf{x}^{k-1}; \quad \mu_k = ||\mathbf{y}^k||_\infty \\
&\qquad \mathbf{x}^k = \mathbf{y}^k/\mu_k \\
&\textbf{end for}
\end{aligned} \tag{8.22}$$

Properties

- First of all,
$$||\mathbf{x}^k||_\infty = 1, \text{ for all } k \geq 0.$$
- It follows from (8.21) that for k sufficiently large,
$$\begin{aligned} \mathbf{x}^k &= \Big(\prod_{j=1}^{k} \mu_j\Big)^{-1} A^k \mathbf{x}^0 \approx \Big(\prod_{j=1}^{k} \mu_j\Big)^{-1} \lambda_1^k \beta_1 \mathbf{v}_1, \\ \mathbf{x}^{k+1} &= \Big(\prod_{j=1}^{k+1} \mu_j\Big)^{-1} A^{k+1} \mathbf{x}^0 \approx \Big(\prod_{j=1}^{k+1} \mu_j\Big)^{-1} \lambda_1^{k+1} \beta_1 \mathbf{v}_1. \end{aligned} \tag{8.23}$$
- Note that $||\mathbf{x}^k||_\infty = ||\mathbf{x}^{k+1}||_\infty = 1$. Comparison of the above two equations reads
$$\Big|\frac{\lambda_1}{\mu_{k+1}}\Big| \approx 1. \tag{8.24}$$

Claim 8.23. Let $\{\mathbf{x}^k, \mu_k\}$ be sequences produced by the power method. Then,
$$\mathbf{x}^k \to \mathbf{v}_1, \ \mu_k \to |\lambda_1|, \quad \text{as } k \to \infty. \tag{8.25}$$
More precisely, the power method converges as
$$\mu_k = |\lambda_1| + \mathcal{O}(|\lambda_2/\lambda_1|^k). \tag{8.26}$$

Note: We have assumed that A has n linearly independent eigenvectors, which equivalently implies that A is **diagonalizable**. Thus, the **Diagonalization Theorem** says that A can be written as
$$A = PDP^{-1} \tag{8.27}$$
where, for $A\mathbf{v}_k = \lambda_k \mathbf{v}_k$, $k = 1, 2, \cdots, n$,
$$\begin{aligned} P &= [\mathbf{v}_1 \ \mathbf{v}_2 \ \cdots \ \mathbf{v}_n], \\ D &= \text{diag}(\lambda_1, \lambda_2, \cdots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}. \end{aligned}$$

Example 8.24. The matrix $A = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$ has eigenvalues and eigenvectors as follows

$$LinearAlgebra[Eigenvectors](A) = \begin{bmatrix} 6 \\ 3 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 & -2 & 0 \\ -1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Verify that the sequences produced by the power method converge to the largest eigenvalue and its associated eigenvector.

Solution.

```
x := ⟨1, 0, 0⟩ :
for k from 1 to 10 do
    y := A x;
    muk := LinearAlgebra[VectorNorm](y, infinity);
    x := 1/muk y;
    print( `k= `, k, `muk= `, evalf[8](muk), evalf[4](x^%T), evalf[4](abs(6 - muk)) );
end do:
```

Results

```
       k=, 1, muk=, 4.,  [1., -0.2500, 0.2500], 2.
  k=, 2, muk=, 4.500000, [1., -0.5000, 0.5000], 1.500
       k=, 3, muk=, 5.,  [1., -0.7000, 0.7000], 1.
 k=, 4, muk=, 5.4000000, [1., -0.8333, 0.8333], 0.6000
 k=, 5, muk=, 5.6666667, [1., -0.9118, 0.9118], 0.3333
 k=, 6, muk=, 5.8235294, [1., -0.9545, 0.9545], 0.1765
 k=, 7, muk=, 5.9090909, [1., -0.9769, 0.9769], 0.09091
 k=, 8, muk=, 5.9538462, [1., -0.9884, 0.9884], 0.04615
 k=, 9, muk=, 5.9767442, [1., -0.9942, 0.9942], 0.02326
k=, 10, muk=, 5.9883268, [1., -0.9971, 0.9971], 0.01167
```

Notice that $|6 - \mu_{k+1}| \approx \frac{1}{2}|6 - \mu_k|$, for which $|\lambda_2/\lambda_1| = \frac{1}{2}$.

power_method.m

```
fmt = ['k=%2d: x = [',repmat('%.5f, ',1,numel(x)-1),'%.5f], ',...
        'mu=%.5f (error = %.5f)\n'];
x = [1 0 0]';
for k=1:10
    y = A*x;
    [~,ind] = max(abs(y)); mu = y(ind);
    x = y/mu;
    fprintf(fmt,k,x,mu,abs(6-mu))
end
```

8.2.2. Symmetric Power Method & Inverse Power Method

Symmetric Power Method

Theorem 8.25. (The Spectral Theorem) *Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric. Then*

(a) A has n eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n$ such that $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$ is an orthonormal basis for $\mathbb{R}^n$.

(b) Every eigenvalue λ of A is a real number.

(c) Eigenvectors corresponding to distinct eigenvalues are orthogonal.

*(d) There exists a diagonal matrix $D \in \mathbb{R}^{n\times n}$ and an **orthogonal matrix** $U \in \mathbb{R}^{n\times n}$ such that*

$$A = UDU^T. \tag{8.28}$$

The diagonal entries of D are the eigenvalues of A and the columns of U are the corresponding eigenvectors.

Note that (a) implies (d).

When $A \in \mathbb{R}^{n\times n}$ is symmetric, the power method can exploit it to converge faster.

Algorithm 8.26. (Symmetric Power Method) Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric. Given $\mathbf{x} \neq 0$:

$$\begin{aligned}
&\textbf{initialization}: \quad \mathbf{x}^0 = \mathbf{x}/||\mathbf{x}||_2 \\
&\textbf{for } \; k = 1, 2, \cdots \\
&\qquad \mathbf{y}^k = A\mathbf{x}^{k-1}; \;\; \mu_k = \mathbf{y}^k \bullet \mathbf{x}^{k-1} \\
&\qquad \mathbf{x}^k = \mathbf{y}^k/||\mathbf{y}^k||_2 \\
&\textbf{end for}
\end{aligned} \tag{8.29}$$

Claim 8.27. The symmetric power method converges as

$$\mu_k = \lambda_1 + \mathcal{O}(|\lambda_2/\lambda_1|^{2k}), \tag{8.30}$$

which converges twice faster than the **basic power method**.

Example 8.28. The matrix A is as in Example 8.24: $A = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$.
Find the largest eigenvalue using the symmetric power method.

Solution.

```
x := ⟨1, 0, 0⟩ :
for k from 1 to 10 do
    y := A x;
    muk := x^%T . y;
    ynorm := LinearAlgebra[VectorNorm](y, 2);
    x := 1/ynorm y;
    print( `k= `, k, `muk= `, evalf[8](muk), evalf[4](x^%T), evalf[4](abs(6 − muk)) );
end do:
```

Results

```
    k=, 1, muk=, 4., [0.9427, -0.2357, 0.2357], 2.
    k=, 2, muk=, 5., [0.8163, -0.4082, 0.4082], 1.
  k=, 3, muk=, 5.6666667, [0.7105, -0.4974, 0.4974], 0.3333
  k=, 4, muk=, 5.9090909, [0.6471, -0.5393, 0.5393], 0.09091
  k=, 5, muk=, 5.9767442, [0.6127, -0.5587, 0.5587], 0.02326
  k=, 6, muk=, 5.9941520, [0.5950, -0.5679, 0.5679], 0.005848
  k=, 7, muk=, 5.9985359, [0.5863, -0.5728, 0.5728], 0.001464
  k=, 8, muk=, 5.9996338, [0.5819, -0.5751, 0.5751], 0.0003662
  k=, 9, muk=, 5.9999084, [0.5792, -0.5760, 0.5760], 0.00009155
  k=, 10, muk=, 5.9999771, [0.5781, -0.5764, 0.5764], 0.00002289
```

Notice that $|6 - \mu_{k+1}| \approx \frac{1}{4}|6 - \mu_k|$, for which $|\lambda_2/\lambda_1| = \frac{1}{2}$.

Accuracy Analysis

Let μ be a real number that approximates an eigenvalue of a symmetric matrix A and $\mathbf{x}$ an associated eigenvector. Then $A\mathbf{x} - \mu\mathbf{x}$ is approximately the zero vector. The following theorem shows a relation between the norm of the vector and the accuracy of μ to the eigenvalue.

Theorem 8.29. *Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric, having eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$. Then, for $\|\mathbf{x}\|_2 = 1$ and $\mu \in \mathbb{R}$,*

$$\min_{1\le i\le n} |\lambda_i - \mu| \le ||A\mathbf{x} - \mu\mathbf{x}||_2. \tag{8.31}$$

Proof. Let $\{\lambda_1, \lambda_2, \cdots, \lambda_n\}$ form an orthonormal set of eigenvectors of A associated with the eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$. Then there is a unique set of constants $\{c_1, c_2, \cdots, c_n\}$ such that

$$\mathbf{x} = \sum_{i=1}^{n} c_i \mathbf{v}_i,$$

where $\sum_{i=1}^{n} c_i^2 = \|\mathbf{x}\|_2^2 = 1$. Thus,

$$\begin{aligned} ||A\mathbf{x} - \mu\mathbf{x}||_2^2 &= \left\| \sum_{i=1}^{n} c_i(A\mathbf{v}_i - \mu\mathbf{v}_i) \right\|_2^2 = \left\| \sum_{i=1}^{n} c_i(\lambda_i - \mu)\mathbf{v}_i \right\|_2^2 \\ &= \sum_{i=1}^{n} c_i^2(\lambda_i - \mu)^2 \\ &\ge \min_{1\le i\le n} |\lambda_i - \mu|^2 \sum_{i=1}^{n} c_i^2 = \min_{1\le i\le n} |\lambda_i - \mu|^2, \end{aligned} \tag{8.32}$$

which completes the proof. □

Note: The above theorem implies that the approximated eigenvalue represents one of the eigenvalues of A, with accuracy that is in the same order as the **stopping tolerance**.

Inverse Power Method

Some applications require to find **an eigenvalue near a prescribed value q.** The **inverse power method** is a variant of the power method to solve such a problem.

- We begin with the eigenvalues and eigenvectors of $(A - qI)^{-1}$. Let

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, 2, \cdots, n. \tag{8.33}$$

- Then it is easy to see that

$$(A - qI)\mathbf{v}_i = (\lambda_i - q)\mathbf{v}_i. \tag{8.34}$$

Thus, we obtain

$$(A - qI)^{-1}\mathbf{v}_i = \frac{1}{\lambda_i - q}\mathbf{v}_i. \tag{8.35}$$

- That is, when $q \notin \{\lambda_1, \lambda_2, \cdots, \lambda_n\}$, the eigenvalues of $(A - qI)^{-1}$ are

$$\frac{1}{\lambda_1 - q}, \frac{1}{\lambda_2 - q}, \cdots, \frac{1}{\lambda_n - q}, \tag{8.36}$$

with the same eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$ of A.

Note: $\dfrac{1}{\lambda - q} = \mu_k \quad \Rightarrow \quad \lambda = 1/\mu_k + q.$

Algorithm 8.30. (Inverse Power Method) Applying the power method to $(A - qI)^{-1}$ gives the **inverse power method**. Given $\mathrm{x} \neq 0$:

$$\begin{aligned}
&\textbf{set}: \quad \mathbf{x}^0 = \mathbf{x}/||\mathbf{x}||_\infty \\
&\textbf{for } k = 1, 2, \cdots \\
&\qquad \mathbf{y}^k = (A - qI)^{-1}\mathbf{x}^{k-1}; \;\; \mu_k = ||\mathbf{y}^k||_\infty \\
&\qquad \mathbf{x}^k = \mathbf{y}^k/\mu_k \\
&\qquad \lambda = 1/\mu_k + q \\
&\textbf{end for}
\end{aligned} \tag{8.37}$$

Example 8.31. The matrix A is as in Example 8.24: $A = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$. Find the the eigenvalue of A nearest to $q = 5/2$, using the inverse power method.

Solution.

```
x := ⟨1, 0, 0⟩ :
q := 5/2 :
Id := Matrix(3, 3, shape = identity) :

for k from 1 to 10 do
    y := (A − q Id)^(−1) . x;
    ynorm := LinearAlgebra[VectorNorm](y, infinity);
    x := 1/ynorm y;
    muk := 1/ynorm + q;
    print( `k= `, k, `muk= `, evalf[8](muk), evalf[4](x^%T), evalf[8](abs(3 − muk)) );
end do:
```

Inverse Power Method

```
1    k=, 1, muk=, 3.2000000, [1., 0.4000, -0.4000], 0.20000000
2    k=, 2, muk=, 3.0303030, [1., 0.4848, -0.4848], 0.030303030
3    k=, 3, muk=, 3.0043668, [1., 0.4978, -0.4978], 0.0043668122
4    k=, 4, muk=, 3.0006246, [1., 0.4997, -0.4997], 0.00062460962
5    k=, 5, muk=, 3.0000892, [1., 0.5000, -0.5000], 0.000089245872
6    k=, 6, muk=, 3.0000127, [1., 0.5000, -0.5000], 0.000012749735
7    k=, 7, muk=, 3.0000018, [1., 0.5000, -0.5000], 0.0000018213974
8    k=, 8, muk=, 3.0000003, [1., 0.5000, -0.5000], 2.6019977 10^-7
9    k=, 9, muk=, 3.0000000, [1., 0.5000, -0.5000], 3.7171398 10^-8
10   k=, 10, muk=, 3.0000000, [1., 0.5000, -0.5000], 5.3101998 10^-9
```

Symmetric Inverse Power method

```
for k from 1 to 10 do
    y := (A - q Id)^(-1) . x;
    lam := x^%T . y;
    ynorm := LinearAlgebra[VectorNorm](y, 2);
    x := 1/ynorm y;
    muk := 1/lam + q;
    print( `k= `, k, `muk= `, evalf[8](muk), evalf[4](x^%T), evalf[10](abs(3 - muk)) );
end do:
```

```
                                Inverse Power Method
  k=, 1, muk=, 3.2000000, [0.8704, 0.3482, -0.3482], 0.2000000000
  k=, 2, muk=, 3.0043668, [0.8245, 0.3997, -0.3997], 0.004366812227
  k=, 3, muk=, 3.0000892, [0.8178, 0.4071, -0.4071], 0.00008924587238
  k=, 4, muk=, 3.0000018, [0.8164, 0.4080, -0.4080], 0.000001821397413
  k=, 5, muk=, 3.0000000, [0.8167, 0.4083, -0.4083], 3.717139787 10^-8
  k=, 6, muk=, 3.0000000, [0.8159, 0.4080, -0.4080], 7.585999658 10^-10
  k=, 7, muk=, 3.0000000, [0.8166, 0.4084, -0.4084], 1.548163196 10^-10
  k=, 8, muk=, 3.0000000, [0.8167, 0.4082, -0.4082], 3.159516726 10^-13
  k=, 9, muk=, 3.0000000, [0.8166, 0.4083, -0.4083], 6.447993319 10^-15
  k=, 10, muk=, 3.0000000, [0.8171, 0.4084, -0.4084], 1.315917004 10^-16
```

The **symmetric inverse power method** converges twice faster than the **basic inverse power method**.

8.3. QR Factorization by the Gram-Schmidt Process

Note: Power methods are not in general suitable for calculating all the eigenvalues of a matrix.
In this section, we will consider the so-called **QR factorization**. The factorization can be utilized **to find all the eigenvalues simultaneously**. However, it is useful for a wide range of applications. Thus here we will derive the QR algorithm for general purpose.

Theorem 8.32. (QR Factorization Theorem) *For $A \in \mathbb{R}^{m\times n}$, we can factor it as*

$$A = QR,$$

*where Q is an $m \times n$ matrix whose columns are orthonormal and R is an $n \times n$ upper triangular matrix. If A is of **full rank**, the factorization is **unique** with R having positive main diagonal entries.*

Corollary 8.33. *Let $A \in \mathbb{R}^{n\times n}$. Then there exist an **orthogonal matrix** Q and an **upper triangular matrix** R such that $A = QR$.*

Corollary 8.34. *Let $A \in \mathbb{R}^{n\times n}$ be **nonsingular**. Then there exist **unique** $Q, R \in \mathbb{R}^{n\times n}$ such that Q is orthogonal, R is upper triangular with positive main diagonal entries, and $A = QR$.*

Computation of the QR factorization

There have been several methods for the computation of the QR factorization, such as by means of

$$\begin{cases} \textbf{Gram-Schmidt process} \\ \textbf{Householder reflectors} \end{cases}$$

The Gram-Schmidt process is considered earlier in Section 6.1.2, p. 224.

Theorem 8.35. (The Gram-Schmidt process – Revisit of Theorem 6.5) *Given a basis* $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ *for a nonzero subspace* W *of* $\mathbb{R}^n$*, define*

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{x}_1 \\
\mathbf{v}_2 &= \mathbf{x}_2 - \frac{\mathbf{x_2}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 \\
\mathbf{v}_3 &= \mathbf{x}_3 - \frac{\mathbf{x_3}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 - \frac{\mathbf{x_3}\bullet\mathbf{v_2}}{\mathbf{v_2}\bullet\mathbf{v_2}}\mathbf{v}_2 \\
&\vdots \\
\mathbf{v}_p &= \mathbf{x}_p - \frac{\mathbf{x_p}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 - \frac{\mathbf{x_p}\bullet\mathbf{v_2}}{\mathbf{v_2}\bullet\mathbf{v_2}}\mathbf{v}_2 - \cdots - \frac{\mathbf{x_p}\bullet\mathbf{v}_{p-1}}{\mathbf{v}_{p-1}\bullet\mathbf{v}_{p-1}}\mathbf{v}_{p-1}
\end{aligned} \tag{8.38}$$

Then $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ *is an* ***orthogonal basis*** *for* W*. In addition,*

$$\textit{Span}\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\} = \textit{Span}\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\}, \quad \textit{for } 1 \le k \le p. \tag{8.39}$$

Algorithm 8.36. (QR Factorization by Gram-Schmidt Process) *Let* $A = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n]$ *have full rank. From the Gram-Schmidt process, obtain an* ***orthonormal basis*** $\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_n\}$*. Then*

$$\begin{aligned}
\mathbf{x}_1 &= (\mathbf{u}_1\bullet\mathbf{x}_1)\mathbf{u}_1 \\
\mathbf{x}_2 &= (\mathbf{u}_1\bullet\mathbf{x}_2)\mathbf{u}_1 + (\mathbf{u}_2\bullet\mathbf{x}_2)\mathbf{u}_2 \\
\mathbf{x}_3 &= (\mathbf{u}_1\bullet\mathbf{x}_3)\mathbf{u}_1 + (\mathbf{u}_2\bullet\mathbf{x}_3)\mathbf{u}_2 + (\mathbf{u}_3\bullet\mathbf{x}_3)\mathbf{u}_3 \\
&\vdots \\
\mathbf{x}_n &= \textstyle\sum_{j=1}^{n}(\mathbf{u}_j\bullet\mathbf{x}_n)\mathbf{u}_j,
\end{aligned} \tag{8.40}$$

which are vector expressions over the orthonormal basis. Thus

$$A = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n] = QR \tag{8.41}$$

implies that

$$\begin{aligned}
Q &= [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_n], \\
R &= \begin{bmatrix}
\mathbf{u}_1\bullet\mathbf{x}_1 & \mathbf{u}_1\bullet\mathbf{x}_2 & \mathbf{u}_1\bullet\mathbf{x}_3 & \cdots & \mathbf{u}_1\bullet\mathbf{x}_n \\
0 & \mathbf{u}_2\bullet\mathbf{x}_2 & \mathbf{u}_2\bullet\mathbf{x}_3 & \cdots & \mathbf{u}_2\bullet\mathbf{x}_n \\
0 & 0 & \mathbf{u}_3\bullet\mathbf{x}_3 & \cdots & \mathbf{u}_3\bullet\mathbf{x}_n \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & \mathbf{u}_n\bullet\mathbf{x}_n
\end{bmatrix} = Q^T A.
\end{aligned} \tag{8.42}$$

Example **8.37.** Find the QR factorization for $A = \begin{bmatrix} 4 & -1 \\ 3 & 2 \end{bmatrix}$.

Solution. Using the Gram-Schmidt process,

$$\begin{cases} \mathbf{v}_1 = \mathbf{x}_1 = \begin{bmatrix} 4 \\ 3 \end{bmatrix} \\ \mathbf{v}_2 = \mathbf{x}_2 - \dfrac{\mathbf{x_2} \bullet \mathbf{v_1}}{\mathbf{v_1} \bullet \mathbf{v_1}} \mathbf{v}_1 = \begin{bmatrix} -1 \\ 2 \end{bmatrix} - \dfrac{2}{25} \begin{bmatrix} 4 \\ 3 \end{bmatrix} = \dfrac{1}{25} \begin{bmatrix} -33 \\ 44 \end{bmatrix} \end{cases}$$

and

$$\begin{cases} \mathbf{u}_1 = \mathbf{v}_1 / ||\mathbf{v}_1|| = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} \\ \mathbf{u}_2 = \mathbf{v}_2 / ||\mathbf{v}_2|| = \begin{bmatrix} -0.6 \\ 0.8 \end{bmatrix} \end{cases} \tag{8.43}$$

Thus,

$$Q = [\mathbf{u}_1, \mathbf{u}_2] = \begin{bmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{bmatrix} \tag{8.44}$$

Now, you can get $R = Q^T A$.

Ans: $Q = \begin{bmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{bmatrix}$ $R = \begin{bmatrix} 5 & 0.4 \\ 0 & 2.2 \end{bmatrix}$

Note: The **determinant** of the orthogonal matrix has a value of ± 1. Thus $det\,(A) = \pm det\,(R)$.

Example 8.38. Find the QR factorization for $A = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \\ 3 & 3 & 7 \end{bmatrix}$.

QR factorization

```
with(LinearAlgebra):
A := Matrix([[3, 1, 3], [1, 6, 4], [6, 7, 8], [3, 3, 7]]):
a[1] := Column(A, 1): v[1] := a[1]: u[1] := v[1]/norm(v[1], 2):

n:=3:
for k from 2 to n do
    a[k] := Column(A, k);
    v[k] := a[k] - add((a[k].u[j]) u[j], j = 1..k-1);
    u[k] := v[k]/norm(v[k],2);
end do:

R := Matrix(n):
for i to n do
    for j from i to n do R[i,j] := u[i].a[j]; end do;
end do:
```

$$Q := \langle u_1 | u_2 | u_3 \rangle = \begin{bmatrix} \frac{3\sqrt{55}}{55} & -\frac{5\sqrt{143}}{143} & -\frac{23\sqrt{4030}}{12090} \\ \frac{\sqrt{55}}{55} & \frac{54\sqrt{143}}{715} & \frac{\sqrt{4030}}{2015} \\ \frac{6\sqrt{55}}{55} & \frac{\sqrt{143}}{143} & -\frac{38\sqrt{4030}}{6045} \\ \frac{3\sqrt{55}}{55} & -\frac{3\sqrt{143}}{715} & \frac{173\sqrt{4030}}{12090} \end{bmatrix}$$

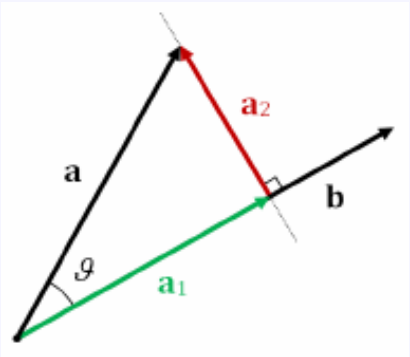
$$R = \begin{bmatrix} \sqrt{55} & \frac{12\sqrt{55}}{11} & \frac{82\sqrt{55}}{55} \\ 0 & \frac{5\sqrt{143}}{11} & \frac{32\sqrt{143}}{143} \\ 0 & 0 & \frac{3\sqrt{4030}}{65} \end{bmatrix}$$

8.4. Householder Reflectors

Let's revisit the orthogonal projection, Definition 6.1:

Definition 8.39. For a nonzero vector $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{a} \in \mathbb{R}^n$, the **orthogonal projection of $\mathbf{a}$ onto $\mathbf{b}$** is defined as

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a}\bullet\mathbf{b}}{\mathbf{b}\bullet\mathbf{b}}\mathbf{b}. \tag{8.45}$$



In the figure,

- $\text{proj}_{\mathbf{b}}\mathbf{a} = \mathbf{a}_1$
- $||\mathbf{a}_1|| = ||\mathbf{a}|| \cos\theta$
- $\mathbf{a} = \mathbf{a}_1 + \mathbf{a}_2$ is called an **orthogonal decomposition** of $\mathbf{a}$.

A **Householder reflector** (or, **Householder transformation**, **elementary reflector**) is a transformation that takes a vector and reflects it about a hyperplane.

- Let $\mathbf{u}$ be a **unit vector** which is orthogonal to the hyperplane. Then, the reflection of a point $\mathbf{x}$ about this hyperplane is given as

$$\mathbf{x} - 2\,\text{proj}_{\mathbf{u}}\mathbf{x} = \mathbf{x} - 2(\mathbf{x}\bullet\mathbf{u})\mathbf{u}. \tag{8.46}$$

  (In the above figure, you may think *Span*$\{\mathbf{b}\}$ is a hyperplane; $\mathbf{x} = \mathbf{a}$ and $\mathbf{u} = \mathbf{a}_2/||\mathbf{a}_2||$.)

- Note that

$$\mathbf{x} - 2(\mathbf{x}\bullet\mathbf{u})\mathbf{u} = \mathbf{x} - 2\mathbf{u}(\mathbf{u}^T\mathbf{x}) = (I - 2\mathbf{u}\mathbf{u}^T)\mathbf{x}. \tag{8.47}$$

Definition 8.40. The matrix

$$Q := I - 2\mathbf{u}\mathbf{u}^T \tag{8.48}$$

is called a **Householder matrix** or a **Householder reflector**.

Theorem 8.41. Properties of Householder Reflectors

I. Let $\mathbf{u} \in \mathbb{R}^n$, with $||\mathbf{u}|| = 1$, and define $P \in \mathbb{R}^{n\times n}$ by $\boldsymbol{P = \mathbf{u}\mathbf{u}^T}$. Then

(a) $P\mathbf{u} = \mathbf{u}$
(b) $P\mathbf{v} = 0$ *if* $\mathbf{u}\bullet\mathbf{v} = 0$
(c) $P^2 = P$
(d) $P^T = P$

II. Let $\mathbf{u} \in \mathbb{R}^n$, with $||\mathbf{u}|| = 1$, and define $Q \in \mathbb{R}^{n\times n}$ by $\boldsymbol{Q = I - 2\mathbf{u}\mathbf{u}^T}$. Then

(a) $Q\mathbf{u} = -\mathbf{u}$
(b) $Q\mathbf{v} = \mathbf{v}$ *if* $\mathbf{u}\bullet\mathbf{v} = 0$
(c) $Q = Q^T$ *(Q is symmetric)*
(d) $Q^T = Q^{-1}$ *(Q is orthogonal)*
(e) $Q^{-1} = Q$ *(Q is an **involution**)*

III. Let $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ with $\mathbf{x} \neq \mathbf{y}$ and $||\mathbf{x}|| = ||\mathbf{y}||$. Then there is a unique reflector Q such that $Q\mathbf{x} = \mathbf{y}$.
(We will prove it by constructing such a reflector Q, in Algorithm 8.44 below.)

Corollary 8.42. *Let $\mathbf{u} \in \mathbb{R}^n$ be a nonzero vector. Define $Q \in \mathbb{R}^{n\times n}$ by*

$$Q = I - \gamma\,\mathbf{u}\mathbf{u}^T, \quad \gamma = 2/||\mathbf{u}||^2. \tag{8.49}$$

Then

(a) $Q\mathbf{u} = -\mathbf{u}$
(b) $Q\mathbf{v} = \mathbf{v}$ *if* $\mathbf{u}\bullet\mathbf{v} = 0$

Clearly, Q satisfies all other properties in Theorem 8.41.II.

Corollary 8.43. *Let $\mathbf{x} \in \mathbb{R}^n$ be a nonzero vector. Then there is a reflector Q such that*

$$Q\mathbf{x} = Q\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{8.50}$$

Proof. Let $\mathbf{y} = [-\tau, 0, \cdots, 0]^T$ with $\tau = \pm||\mathbf{x}||$. By choosing the sign appropriately, we can guarantee that $\mathbf{x} \neq \mathbf{y}$. Clearly $||\mathbf{x}|| = ||\mathbf{y}||$. Thus, the corollary follows from Theorem 8.41.III. □

Algorithm 8.44. (Construction of the Reflector)

$$Q : \mathbf{x} \mapsto \mathbf{y} = [-\tau, 0, \cdots, 0]^T, \quad \text{where } \tau = \pm||\mathbf{x}||. \tag{8.51}$$

- Let
$$\mathbf{u} = \mathbf{x} - \mathbf{y} = [x_1 + \tau, x_2, \cdots, x_n]^T, \tag{8.52}$$
where the sign of τ is such that $x_1 + \tau \neq 0$.
- Define
$$Q = I - \gamma\, \mathbf{u}\mathbf{u}^T, \quad \gamma = \frac{2}{||\mathbf{u}||^2}. \tag{8.53}$$

Check if $Q\mathbf{x} = \mathbf{y}$

- Rewrite $\mathbf{x}$ as
$$\mathbf{x} = \frac{1}{2}(\mathbf{x} - \mathbf{y}) + \frac{1}{2}(\mathbf{x} + \mathbf{y}). \tag{8.54}$$
- By Corollary 8.42 (a),
$$Q(\mathbf{x} - \mathbf{y}) = Q\mathbf{u} = -\mathbf{u} = -\mathbf{x} + \mathbf{y}. \tag{8.55}$$
- Since $(\mathbf{x} - \mathbf{y})\bullet(\mathbf{x} + \mathbf{y}) = ||\mathbf{x}||^2 - ||\mathbf{y}||^2 = 0$, it follows from Corollary 8.42 (b) that
$$Q(\mathbf{x} + \mathbf{y}) = \mathbf{x} + \mathbf{y}. \tag{8.56}$$
- Thus, $Q\mathbf{x} = \mathbf{y}$ holds. □

Example 8.45. Let $\mathbf{x} = [2,\ 2,\ -1]^T$ and $\mathbf{y} = [*,\ 0,\ 0]^T$.
Find $Q = I - \gamma\,\mathbf{u}\mathbf{u}^T : \mathbf{x} \mapsto \mathbf{y}$, where $\mathbf{u} = \mathbf{x} - \mathbf{y}$ and $\gamma = 2/||\mathbf{u}||^2$.

Solution.

- Let $\tau = ||\mathbf{x}||$. Then $\tau = 3$.
- $\mathbf{y} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$ and $\mathbf{u} = \mathbf{x} - \mathbf{y} = \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix} - \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ -1 \end{bmatrix}$.
- $Q = I - \dfrac{2}{||\mathbf{u}||^2}\mathbf{u}\mathbf{u}^T = I - \dfrac{2}{30}\begin{bmatrix} 25 & 10 & -5 \\ 10 & 4 & -2 \\ -5 & -2 & 1 \end{bmatrix} = \dfrac{1}{15}\begin{bmatrix} -10 & -10 & 5 \\ -10 & 11 & 2 \\ 5 & 2 & 14 \end{bmatrix}$
- $Q\mathbf{x} = \begin{bmatrix} -3 \\ 0 \\ 0 \end{bmatrix}$

What if we set $\tau = -3$?

- Then, $\mathbf{y} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$ and $\mathbf{u} = \mathbf{x} - \mathbf{y} = \begin{bmatrix} 2 \\ 2 \\ -1 \end{bmatrix} - \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix}$.
- $Q = I - \dfrac{2}{||\mathbf{u}||^2}\mathbf{u}\mathbf{u}^T = I - \dfrac{2}{6}\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} = \dfrac{1}{3}\begin{bmatrix} 2 & 2 & -1 \\ 2 & -1 & 2 \\ -1 & 2 & 2 \end{bmatrix}$
- $Q\mathbf{x} = \begin{bmatrix} 3 \\ 0 \\ 0 \end{bmatrix}$

Note: For each $\mathbf{y}$, Q is symmetric, orthogonal, and an involution matrix. In addition, it is **unique** for each choice of $\mathbf{y}$.

8.5. QR Factorization by the Householder Reflectors

We first revisit Corollary 8.33, p. 297:

Corollary 8.46. (Revisit of Corollary 8.33)
*Let $A \in \mathbb{R}^{n\times n}$. Then there exist an **orthogonal matrix** Q and an **upper triangular matrix** R such that $A = QR$.*

Proof. The proof is by **mathematical induction** on n.

- When $n = 1$: $A = [a_{11}]$.
  Then let $Q = [1]$ and $R = [a_{11}]$ to get $A = QR$.
- Let $A \in \mathbb{R}^{n\times n}$ for $n \geq 2$.
  (a) Assume that QR factorization exists for $(n-1)\times(n-1)$ matrices.
  (b) Let Q_1 be a reflector that creates zeros in the first column of A, $\mathbf{a}_1$:

$$Q_1\mathbf{a}_1 = Q_1\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1} \end{bmatrix} = \begin{bmatrix} -\tau_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{where } \tau_1 = \pm||\mathbf{a}_1||. \tag{8.57}$$

  (c) Thus,

$$Q_1^T A = Q_1 A = \begin{bmatrix} -\tau & \widehat{a_{12}} & \cdots & \widehat{a_{1n}} \\ 0 & & & \\ \vdots & & \widehat{A_2} & \\ 0 & & & \end{bmatrix} \tag{8.58}$$

  where, by the induction hypothesis, $\widehat{A_2} \in \mathbb{R}^{(n-1)\times(n-1)}$ has a QR decomposition

$$\widehat{A_2} = \widehat{Q_2}\widehat{R_2}.$$

  (d) Define $Q_2 \in \mathbb{R}^{n\times n}$ by

$$Q_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{Q_2} & \\ 0 & & & \end{bmatrix}. \tag{8.59}$$

(e) Then Q_2 is obviously orthogonal and, from (8.58) and (8.59), we have

$$Q_2^T Q_1^T A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{Q_2} & \\ 0 & & & \end{bmatrix} \begin{bmatrix} -\tau & \widehat{a_{12}} & \cdots & \widehat{a_{1n}} \\ 0 & & & \\ \vdots & & \widehat{A_2} & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} -\tau & \widehat{a_{12}} & \cdots & \widehat{a_{1n}} \\ 0 & & & \\ \vdots & & \widehat{R_2} & \\ 0 & & & \end{bmatrix}. \tag{8.60}$$

(f) The last matrix is upper triangular; let us call it R. Let $Q = Q_1 Q_2$. Then $A = QR$, clearly.

This completes the proof. □

Remark 8.47.

1. For $Q = I - \gamma \mathbf{u}\mathbf{u}^T : \mathbf{x} \mapsto \mathbf{y} = [-\tau, 0, \cdots, 0]^T$, since any multiple of $\mathbf{u} = \mathbf{x} - \mathbf{y}$ will generate the same reflector, you may scale $\mathbf{u}$ so that its first entry is 1. That is,

$$\mathbf{u} = \frac{\mathbf{x} - \mathbf{y}}{x_1 + \tau} = \begin{bmatrix} 1 \\ x_2/(x_1+\tau) \\ \vdots \\ x_n/(x_1+\tau) \end{bmatrix}. \tag{8.61}$$

In this case, the first entry of $\mathbf{u}$ does not need to be saved. Furthermore

$$||\mathbf{u}||^2 = \frac{(x_1+\tau)^2 + x_2^2 + \cdots + x_n^2}{(x_1+\tau)^2} = \frac{\tau^2 + 2\tau x_1 + ||\mathbf{x}||^2}{(x_1+\tau)^2}.$$

Since $\tau^2 = ||\mathbf{x}||^2$,

$$||\mathbf{u}||^2 = \frac{2\tau^2 + 2\tau x_1}{(x_1+\tau)^2} = \frac{2\tau(\tau + x_1)}{(x_1+\tau)^2} = \frac{2\tau}{x_1+\tau}, \tag{8.62}$$

and therefore

$$\gamma = \frac{2}{||\mathbf{u}||^2} = \frac{x_1+\tau}{\tau}. \tag{8.63}$$

2. The the proof of Corollary 8.46 suggests an algorithm for the construction of Q and R.

$$Q_1 = I - \gamma_1 \mathbf{u}_1 \mathbf{u}_1^T, \quad \text{where } \mathbf{u}_1 = \begin{bmatrix} 1 \\ a_{21}/(a_{11}+\tau_1) \\ \vdots \\ a_{n1}/(a_{11}+\tau_1) \end{bmatrix}, \tag{8.64}$$

and

$$Q_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & I - \gamma_2 \mathbf{u}_2 \mathbf{u}_2^T & \\ 0 & & & \end{bmatrix}, \tag{8.65}$$

where $\mathbf{u}_2$ is determined from $\widehat{A_2}$. In general,

$$Q_k = \begin{bmatrix} I_{k-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & & & \\ \vdots & & I - \gamma_k \mathbf{u}_k \mathbf{u}_k^T & \\ \mathbf{0} & & & \end{bmatrix}. \tag{8.66}$$

Then, if $Q = Q_1 Q_2 \cdots Q_{n-1}$, then $Q^T A = R$ is an upper triangular matrix.

3. **We do not need form $Q_1, Q_2, \cdots, Q_{n-1}$ explicitly.** For example, for

$$Q_1 = I - \gamma_1 \mathbf{u}_1 \mathbf{u}_1^T,$$

we store only $-\tau_1, \gamma_1, \mathbf{u}_1$. (The construction of Q_1 costs $\mathcal{O}(n^2)$ flops.)

4. Then, for each $\mathbf{a}_k$ (or columns of A), $k = 2, 3, \cdots, n$,

$$Q_1^T \mathbf{a}_k = Q_1 \mathbf{a}_k = (I - \gamma_1 \mathbf{u}_1 \mathbf{u}_1^T)\mathbf{a}_k = \mathbf{a}_k - \gamma_1 (\mathbf{u}_1^T \mathbf{a}_k)\mathbf{u}_1, \tag{8.67}$$

where the last term requires **about $4n$ flops.** The flop count for $Q_1^T A$:

$$\text{cost}(Q_1^T A) \approx 4n^2. \tag{8.68}$$

Algorithm 8.48. (The QR Factorization by Reflectors)
Let $A \in \mathbb{R}^{n\times n}$.

for $k = 1 : (n-1)$

(a) Determine $Q_k = I - \gamma_k \mathbf{u}_k \mathbf{u}_k^T$ such that

$$Q_k[\widehat{a_{kk}}, \widehat{a_{k+1,k}}, \cdots, \widehat{a_{nk}}]^T = [-\tau_k, 0, \cdots, 0]^T$$

(b) Store $\mathbf{u}_k$ over $A[k:n, k]$

(c) Save $A[k,k] = -\tau_k$ (8.69)

(d) Transform $A[k:n, k+1:n] \leftarrow Q_k A[k:n, k+1:n]$

(e) Save $G[k] = \gamma_k$

end for

$G[n] = A[n,n]$

Note: **(The QR Factorization by Reflectors)**

- The output (Q and R) is saved over A.
- Recall that the flop count for $k = 1$ is about $4n^2$. For $k = 2$, it is about $4(n-1)^2$; for $k = 2$, it is about $4(n-2)^2$; and so on. Thus the total flop count for **QR Decomposition by Reflectors** reads

$$4n^2 + 4(n-1)^2 + \cdots \approx \mathcal{O}\Big(\frac{4}{3}n^3\Big), \tag{8.70}$$

  which is ***twice*** that of an **LU-factorization**.
- Although each of Q_k is symmetric and an involution,

$$Q = Q_1 Q_2 \cdots Q_{n-1}$$

  may not be symmetric nor an involution matrix. However, it is still **orthogonal**.
- The matrix A is singular if at least one entry of G (particularly, $G[n]$) is zero.

Example 8.49. Use reflectors to get QR decomposition of $A = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix}$.

Solution. Consider a program showing details.

Q_1

```
m := 3: n := 3:
a[1] := Column(A, 1): tau[1] := norm(a[1], 2):
if a[1][1] < 0 then tau[1] := -tau[1]; end if:

u[1] := a[1]: scale := a[1][1] + tau[1]:
u[1][1] := 1:
for i from 2 to m do
    u[1][i] := u[1][i]/scale;
end do:
g[1] := scale/tau[1]:
```

Result:

$$Q1 := Matrix(m, shape = identity) - g_1 \cdot u_1 . u_1^{\%T};$$

$$\begin{bmatrix} 1 - \frac{1}{46}(3+\sqrt{46})\sqrt{46} & -\frac{1}{46}\sqrt{46} & -\frac{3}{23}\sqrt{46} \\ -\frac{1}{46}\sqrt{46} & 1 - \frac{1}{46}\frac{\sqrt{46}}{3+\sqrt{46}} & -\frac{3}{23}\frac{\sqrt{46}}{3+\sqrt{46}} \\ -\frac{3}{23}\sqrt{46} & -\frac{3}{23}\frac{\sqrt{46}}{3+\sqrt{46}} & 1 - \frac{18}{23}\frac{\sqrt{46}}{3+\sqrt{46}} \end{bmatrix}$$

$$Q1A := simplify\left(Q1^{\%T} . A\right): \quad evalf(Q1A)$$

$$\begin{bmatrix} -6.782329983 & -7.519539763 & -8.993959329 \\ 0. & 5.129088899 & 2.773915892 \\ 0. & 1.774533402 & 0.6434953557 \end{bmatrix}$$

Q_2

```
A2 := Q1A[2..m, 2..n]:
a[2] := Column(A2, 1): tau[2] := norm(a[2], 2):
if evalf(a[2][1]) < 0 then tau[2] := -tau[2]; end if:

u[2] := a[2]: scale := a[2][1] + tau[2]:
u[2][1] := 1:
for i from 2 to m - 1 do
    u[2][i] := u[2][i]/scale;
end do:
g[2] := scale/tau[2]:

Q2sub := Matrix(m-1, shape=identity) - g[2]* u[2].u[2]~%T:

Q2 := Matrix(m): Q2[1, 1] := 1:
for i from 2 to m do
    for j from 2 to m do Q2[i,j] := Q2sub[i-1, j-1]; end do:
end do:
```

Result:

$$evalf(Q2)$$

$$\begin{bmatrix} 1. & 0. & 0. \\ 0. & -0.9450384856 & -0.3269591131 \\ 0. & -0.3269591131 & 0.9450384849 \end{bmatrix}$$

Check:

$$Q := simplify(Q1.Q2) : \ evalf(Q)$$

$$\begin{bmatrix} -0.4423258684 & 0.4285832703 & -0.7878224459 \\ -0.1474419561 & -0.9012265027 & -0.4074943688 \\ -0.8846517369 & -0.06408721798 & 0.4618269510 \end{bmatrix}$$

$$R := simplify\left(Q2^{\%T}.Q1A\right) : \ evalf(R)$$

$$\begin{bmatrix} -6.782329983 & -7.519539763 & -8.993959329 \\ 0. & -5.427386271 & -2.831853944 \\ 0. & 0. & -0.2988292036 \end{bmatrix}$$

$$simplify(Q.R)$$

$$\begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix}$$

Use the build-in command:

$$q, r := QRDecomposition(A)$$

$$\begin{bmatrix} \frac{3}{46}\sqrt{46} & -\frac{107}{62330}\sqrt{62330} & \frac{29}{1355}\sqrt{1355} \\ \frac{1}{46}\sqrt{46} & \frac{45}{12466}\sqrt{62330} & \frac{3}{271}\sqrt{1355} \\ \frac{3}{23}\sqrt{46} & \frac{8}{31165}\sqrt{62330} & -\frac{17}{1355}\sqrt{1355} \end{bmatrix}, \begin{bmatrix} \sqrt{46} & \frac{51}{46}\sqrt{46} & \frac{61}{46}\sqrt{46} \\ 0 & \frac{1}{46}\sqrt{62330} & \frac{707}{62330}\sqrt{62330} \\ 0 & 0 & \frac{11}{1355}\sqrt{1355} \end{bmatrix}$$

$$evalf(q)$$

$$\begin{bmatrix} 0.4423258684 & -0.4285832703 & 0.7878224461 \\ 0.1474419561 & 0.9012265027 & 0.4074943687 \\ 0.8846517369 & 0.06408721798 & -0.4618269512 \end{bmatrix}$$

$$evalf(r)$$

$$\begin{bmatrix} 6.782329983 & 7.519539763 & 8.993959329 \\ 0. & 5.427386271 & 2.831853944 \\ 0. & 0. & 0.2988292037 \end{bmatrix}$$

See Exercise 8.4 for transforming the QR decomposition for all diagonal entries of R are nonnetative.

8.6. QR Algorithm for Eigenvalues and Eigenvectors

The **QR algorithm**, or the **QR iteration**, is very simple and quite useful.

Algorithm 8.50. (QR Algorithm) Let $A \in \mathbb{R}^{n\times n}$.

set $A_0 = A$ and $U_0 = I$
for $k = 1, 2, \cdots$ **do**
(a) $A_{k-1} = Q_k R_k$; % QR factorization
(b) $A_k = R_k Q_k$; (8.71)
(c) $U_k = U_{k-1} Q_k$; % Update transformation matrix
end for
set $T := A_\infty$ and $U := U_\infty$

Claim 8.51. Algorithm 8.50 produces an upper triangular matrix T, with its diagonals being **eigenvalues** of A, and an orthogonal matrix U such that

$$A = UTU^T, \tag{8.72}$$

which is called the **Schur decomposition** of A.

- When A is ***symmetric***, T becomes a diagonal matrix of **eigenvalues** of A and U is the collection of corresponding **eigenvectors**.

Remark 8.52. It follows from (b) and (a) of Algorithm 8.50 that

$$A_k = R_k Q_k = Q_k^T A_{k-1} Q_k, \tag{8.73}$$

and therefore

$$\begin{aligned} A_k &= R_k Q_k = Q_k^T A_{k-1} Q_k = Q_k^T Q_{k-1}^T A_{k-2} Q_{k-1} Q_k = \cdots \\ &= Q_k^T Q_{k-1}^T \cdots Q_1^T A_0 \underbrace{Q_1 Q_2 \cdots Q_k}_{U_k} \end{aligned} \tag{8.74}$$

The above converges to

$$T = U^T A U \tag{8.75}$$

Example 8.53. Let $A = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix}$ and $B = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$. Apply the QR algorithm, Algorithm 8.50, to find their Schur decompositions.

Solution.

qr_algorithm.m

```
function [T,U,iter] = qr_algorithm(A)
% It produces the Schur decomposition: A = U*T*U^T
%   T: upper triangular, with diagonals being eigenvalues of A
%   U: orthogonal
% Once A is symmetric,
%   T becomes diagonal && U contains eigenvectors of A

T = A; U = eye(size(A));

% for stopping
D0 = diag(T); change = 1;
tol = 10^-15; iter=0;

%%-----------------
while change>tol
   [Q,R] = qr(T);
   T = R*Q;
   U = U*Q;

   % for stopping
   iter= iter+1;
   D=diag(T); change=norm(D-D0); D0=D;
end
```

We may call it as

call_qr_algorithm.m

```
A =[3 1 3; 1 6 4; 6 7 8];
[T1,U1,iter1] = qr_algorithm(A)
U1*T1*U1'
[V1,D1] = eig(A)  % built-in

B =[4 -1 1; -1 3 -2; 1 -2 3];
[T2,U2,iter2] = qr_algorithm(B)
U2*T2*U2'
[V2,D2] = eig(B)  % built-in
```

A

```
T1 =
   13.8343    1.0429   -4.0732
    0.0000    3.3996    0.5668
    0.0000   -0.0000   -0.2339
U1 =
    0.2759   -0.5783   -0.7677
    0.4648    0.7794   -0.4201
    0.8414   -0.2409    0.4838
iter1 =
    26
ans =
    3.0000    1.0000    3.0000
    1.0000    6.0000    4.0000
    6.0000    7.0000    8.0000

%---- [V1,D1] = eig(A)
V1 =
   -0.2759   -0.5630    0.6029
   -0.4648   -0.3805   -0.7293
   -0.8414    0.7337    0.3234
D1 =
   13.8343         0         0
         0   -0.2339         0
         0         0    3.3996
```

B

```
T2 =
    6.0000   -0.0000    0.0000
   -0.0000    3.0000   -0.0000
    0.0000   -0.0000    1.0000
U2 =
    0.5774    0.8165   -0.0000
   -0.5774    0.4082    0.7071
    0.5774   -0.4082    0.7071
iter2 =
    28
ans =
    4.0000   -1.0000    1.0000
   -1.0000    3.0000   -2.0000
    1.0000   -2.0000    3.0000

%---- [V2,D2] = eig(B)
V2 =
   -0.0000    0.8165    0.5774
    0.7071    0.4082   -0.5774
    0.7071   -0.4082    0.5774
D2 =
    1.0000         0         0
         0    3.0000         0
         0         0    6.0000
```

Note: QR decomposition can also be used for solving LS problems, when the system matrix has full rank; see Exercise 8.10.

8.7. Eigenvalues and Eigenvectors of Stochastic Matrices

Definition 8.54. Probability Vector and Stochastic Matrix

- A vector $\mathrm{p} = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}$ with ***nonnegative entries that add up to 1*** is called a **probability vector**.
- A **(left) stochastic matrix** is a square matrix whose columns are probability vectors.

A **stochastic matrix** is also called a **probability matrix**, **transition matrix**, **substitution matrix**, or **Markov matrix**.

Lemma 8.55. If p is a probability vector and T is a stochastic matrix, then $T\mathrm{p}$ is a probability vector.

Proof. Let $\mathrm{v}_1, \mathrm{v}_2, \cdots, \mathrm{v}_n$ be the columns of T. Then

$$\boldsymbol{q} := T\mathrm{p} = p_1\mathrm{v}_1 + p_2\mathrm{v}_2 + \cdots + p_n\mathrm{v}_n \in \mathbb{R}^n.$$

Clearly $\boldsymbol{q}$ has nonnegative entries; their sum reads

$$\mathrm{sum}(\boldsymbol{q}) = \mathrm{sum}(p_1\mathrm{v}_1 + p_2\mathrm{v}_2 + \cdots + p_n\mathrm{v}_n) = p_1 + p_2 + \cdots + p_n = 1. \square$$

Definition 8.56. Markov Chain

In general, a finite **Markov chain** is a sequence of **probability vectors** $\mathrm{x}_0, \mathrm{x}_1, \mathrm{x}_2, \cdots$, together with a **stochastic matrix** T, such that

$$\mathrm{x}_1 = T\mathrm{x}_0, \quad \mathrm{x}_2 = T\mathrm{x}_1, \quad \mathrm{x}_3 = T\mathrm{x}_2, \quad \cdots \tag{8.76}$$

We can rewrite the above conditions as a **recurrence relation**

$$\mathrm{x}_{k+1} = T\mathrm{x}_k, \quad k = 0, 1, 2, \cdots \tag{8.77}$$

The vector x_k is often called a **state vector**.

The Maximum of Eigenvalues

Definition 8.57. For a vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T \in \mathbb{R}^n$, the **p-norm** is defined as

$$||\mathbf{x}||_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{1/p}, \quad p > 0. \tag{8.78}$$

For example,

$$\begin{aligned} ||\mathbf{x}||_1 &= |x_1| + |x_2| + \cdots + |x_n| && \text{(1-norm)} \\ ||\mathbf{x}||_2 &= \sqrt{|x_1|^2 + |x_2|^2 + \cdots + |x_n|^2} && \text{(2-norm)} \end{aligned} \tag{8.79}$$

Theorem 8.58. *Let $T \in \mathbb{R}^{n\times n}$ be a stochastic matrix. Then*

$$||T\mathbf{x}||_1 \le ||\mathbf{x}||_1, \quad \forall\, \mathbf{x} \in \mathbb{R}^n. \tag{8.80}$$

Proof. Consider the case, $n = 2$: $T = \begin{bmatrix} t_{11} & t_{12} \\ t_{21} & t_{22} \end{bmatrix}$. Then, for $\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$,

$$\begin{aligned} ||T\mathbf{x}||_1 &= |t_{11}x_1 + t_{12}x_2| + |t_{21}x_1 + t_{22}x_2| \\ &\le t_{11}|x_1| + t_{12}|x_2| \; + \; t_{21}|x_1| + t_{22}|x_2| \\ &= (t_{11} + t_{21})|x_1| \; + \; (t_{12} + t_{22})|x_2| \\ &= |x_1| + |x_2| = ||\mathbf{x}||_1. \end{aligned} \tag{8.81}$$

For general $n \ge 2$, use the same argument to complete the proof. □

Corollary 8.59. *Let $T \in \mathbb{R}^{n\times n}$ be a stochastic matrix. Then every eigenvalue of T is bounded by 1 in modulus. That is,*

$$T\mathbf{v} = \lambda\mathbf{v} \quad \Rightarrow \quad |\lambda| \le 1. \tag{8.82}$$

Proof. Let $T\mathbf{v} = \lambda\mathbf{v}$. Then it follows from Theorem 8.58 that

$$||T\mathbf{v}||_1 = ||\lambda\mathbf{v}||_1 = |\lambda|\,||\mathbf{v}||_1 \le ||\mathbf{v}||_1, \tag{8.83}$$

which completes the proof. □

The Eigenvalue 1 and Its Corresponding Eigenvector

Theorem 8.60. *Let $T \in \mathbb{R}^{n\times n}$ be a stochastic matrix. Then the number 1 is an eigenvalue of T.*

Proof. We prove the theorem in two different ways.

(a) Note that $det\,(T^T - \lambda I) = det\,(T - \lambda I)$, which implies that ***$T$* and *$T^T$* have exactly the same eigenvalues.** Consider the **all-ones vector** $\mathbf{1} = [1, 1, \cdots, 1]^T \in \mathbb{R}^n$. Then

$$T^T\mathbf{1} = \mathbf{1},$$

which implies that the number 1 is an eigenvalue of T^T and therefore it is an eigenvalue of T.

(b) Construct $T - \lambda I$ for $\lambda = 1$:

$$T - I = \begin{bmatrix} t_{11} - 1 & t_{12} & \cdots & T_{1n} \\ t_{21} & t_{22} - 1 & \cdots & T_{2n} \\ \vdots & & \ddots & \vdots \\ t_{n1} & t_{12} & \cdots & T_{1n} - 1 \end{bmatrix}. \tag{8.84}$$

We apply **replacement** operations so that all rows are added to the bottom row. Then **the resulting bottom row must become a zero row**, which implies

$$det\,(T - I) = 0, \tag{8.85}$$

and therefore the number 1 is an eigenvalue of T. □

Definition 8.61. The **eigenvector** v corresponding to the eigenvalue 1 is called a **steady-state vector** of T. It is also called a **Perron-Frobenius eigenvector** or a **stable equilibrium distribution**.

The steady-state vector represents a **long term behavior** of a Markov chain.

The Steady-State Vector

Strategy 8.62. How to Find a Steady-State Vector

(a) First, solve for $\mathbf{x} = [x_1, x_2, \cdots, x_n]^T$:

$$T\mathbf{x} = \mathbf{x} \;\Leftrightarrow\; T\mathbf{x} - \mathbf{x} = \mathbf{0} \;\Leftrightarrow\; (T - I)\mathbf{x} = \mathbf{0}. \tag{8.86}$$

(b) Then, set

$$\boldsymbol{q} = \frac{1}{x_1 + x_2 + \cdots + x_n}\mathbf{x}. \tag{8.87}$$

Definition 8.63. A stochastic matrix T is **regular** if some matrix power T^k contains only strictly positive entries.

Theorem 8.64. *If T is an $n \times n$ **regular stochastic matrix**, then T has a **unique steady-state vector** q.*

(a) The entries of q are strictly positive.

*(b) The steady-state vector can be computed by the **power method***

$$\boldsymbol{q} = \lim_{k\to\infty} T^k \mathbf{x}_0, \tag{8.88}$$

*where x_0 is **a probability vector**.*

Remark 8.65. Let $T \in \mathbb{R}^{n\times n}$ be a regular stochastic matrix. Then

- If $T\mathrm{v} = \lambda\mathrm{v}$, then $|\lambda| \le 1$.
  (The above is true for every stochastic matrix.)
- Every column of T^k converges to $\boldsymbol{q}$ as $k \to \infty$, i.e.,

$$T^k \to [\boldsymbol{q}\ \ \boldsymbol{q}\ \ \cdots\ \ \boldsymbol{q}] \in \mathbb{R}^{n\times n}, \quad \text{as } k \to \infty. \tag{8.89}$$

See Exercise 8 on p. 321.

Example 8.66. Find eigenvalues and corresponding eigenvectors of the transition matrix.

$$T = \begin{bmatrix} 1/2 & 1/4 & 1/6 \\ 1/3 & 1/2 & 1/3 \\ 1/6 & 1/4 & 1/2 \end{bmatrix}. \tag{8.90}$$

Solution.

stochastic_eigen.py

```
import numpy as np
np.set_printoptions(precision=4,suppress=True)

T = [[1/2,1/4,1/6],
     [1/3,1/2,1/3],
     [1/6,1/4,1/2]]
T = np.array(T)

D,V = np.linalg.eig(T)

print('Eigenvalues:');  print(D)
print('Eigenvectors:'); print(V)

print('----- steady-state vector ----')
v1 = V[:,0]; v1 /= sum(v1)
print('v1 = ',v1)

print('----- power method -----------')
x = np.array([1,0,0]);
print('k = %2d; '%(0),x)
for k in range(10):
    x = T.dot(x);
    print('k = %2d; '%(k+1),x)
```

Output

```
Eigenvalues:
[1.     0.3333 0.1667]
Eigenvectors:
[[-0.5145 -0.7071  0.4082]
 [-0.686  -0.     -0.8165]
 [-0.5145  0.7071  0.4082]]
----- steady-state vector ----
v1 =  [0.3 0.4 0.3]
----- power method -----------
k =  0;  [1 0 0]
k =  1;  [0.5    0.3333 0.1667]
k =  2;  [0.3611 0.3889 0.25  ]
```

```
k =  3;  [0.3194 0.3981 0.2824]
k =  4;  [0.3063 0.3997 0.294 ]
k =  5;  [0.3021 0.3999 0.298 ]
k =  6;  [0.3007 0.4    0.2993]
k =  7;  [0.3002 0.4    0.2998]
k =  8;  [0.3001 0.4    0.2999]
k =  9;  [0.3 0.4 0.3]
k = 10;  [0.3 0.4 0.3]
```

Question: What is the convergence rate?

Remark **8.67.** *Some eigenvalues of a stochastic matrix can be negative.*
For example, apply a **row interchange operation** to T in (8.90):

```
T[[1,2]] = T[[2,1]]
```

Then

- The resulting matrix is still a stochastic matrix.
- Its eigenvalues become

```
[ 1.   0.281 -0.1977].
```

True-or-False **8.68.** Let T be a stochastic matrix.

a. The steady-state vector is an eigenvector of T.
b. Every eigenvector of T is a steady-state vector.
c. The all-ones vector is an eigenvector of T^T.
d. The number 2 can be an eigenvalue of T or T^T.
e. All stochastic matrices are regular.

Ans: T, F, T, F, F

Exercises for Chapter 8

8.1. Prove that $||\mathbf{x}||_2 \leq ||\mathbf{x}||_1$ for all $\mathbf{x} \in \mathbb{R}^n$.

8.2. Let $A = \begin{bmatrix} 2 & 1 & -2 \\ 0 & 3 & -1 \\ 4 & 5 & 1 \end{bmatrix}$. Find $||A||_1$, $||A||_2$, and $||A||_\infty$.

8.3. Prove (c) in Theorem 8.25. ***Hint***: Let $A\mathbf{v}_i = \lambda_i \mathbf{v}_i$, $i = 1, 2$, and $\lambda_1 \neq \lambda_2$. Consider $\mathbf{v}_1^T A \mathbf{v}_2$ and $\mathbf{v}_2^T A \mathbf{v}_1$.

8.4. Let $A = QR$ be a QR factorization of A. Suppose a single diagonal entry of R, r_{jj}, is negative. Suggest a strategy to transform the QR factorization so that all diagonal entries of R are nonnegative. ***Hint***: Let N_j be a perturbation of I where the (j, j)-th entry is -1. Then $N_j^2 = I$.

8.5. Find a reflector that maps the vector $\mathbf{x} = [-4, 1, 2, -2]^T$ to a vector of the form $\mathbf{y} = [-\tau, 0, 0, 0]^T$. Write Q in two ways: (a) in the form of $I - \gamma \mathbf{u}\mathbf{u}^T$ and (b) as a completely assembled matrix.

8.6. Let $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$. Find QR decompositions for by using

(a) the Gram-Schmidt process

(b) Householder reflectors.

(You have to show your solutions step-by-step in detail.)

8.7. Recall that the Frobenius matrix norm is defined by

$$||A||_F = \Big(\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2 \Big)^{1/2}, \quad A \in \mathbb{R}^{m \times n}.$$

Show that $||A||_F = (\sigma_1^2 + \cdots + \sigma_k^2)^{1/2}$, where σ_j are nonzero singular values of A. **Hint**: You may use the norm-preserving property of orthogonal matrices. That is, if U is orthogonal, then $||UB||_2 = ||B||_2$ and $||UB||_F = ||B||_F$.

8.8. Let $T \in \mathbb{R}^{n \times n}$ be a regular stochastic matrix. Prove (8.89). ***Hint***: You may use (8.88).

8.9. C[1] Let $A = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & 0 & -1 \\ 0 & 0 & 4 & -2 \\ 0 & -1 & -2 & 4 \end{bmatrix}$. Use indicated methods to approximate eigenvalues and their associated eigenvectors of A within to 10^{-12} accuracy.

[1]The mark C indicates that you *should* solve the problem via computer programming. Attach hard copies of your code and results. For other problems, if you like and doable, you may try to solve them with computer programming.

(a) The power method, the largest eigenvalue.
(b) The inverse power method, the smallest eigenvalue.
(c) The inverse power method, an eigenvalue near $q = 3$.
(d) Repeat the above with their symmetric versions.

8.10. C This problem revisits Exercise 6.3, p. 253. Consider the same data

x_i	0.2	0.4	0.6	0.8	1.	1.2	1.4	1.6	1.8	2.
y_i	1.88	2.13	1.76	2.78	3.23	3.82	6.13	7.22	6.66	9.07

(a) Plot the data (scattered point plot) and decide which curve fits the data best.
(b) Construct an algebraic system of the form $X\boldsymbol{\beta} = \mathbf{y}$, where $X \in \mathbb{R}^{m\times n}$, $n < m = 10$.
(c) Use **QR decomposition** to find $\widehat{\boldsymbol{\beta}}$ and the curve.
(You do not have to implement a code for QR; you can use a built-in function)
Clue: Let $X = QR$. Then $X\boldsymbol{\beta} = \mathbf{y}$ can be written as $R\boldsymbol{\beta} = Q^T\mathbf{y}$.
(d) Plot the curve superposed over the point plot.

CHAPTER 9

Principal Component Analysis

The *Singular Value Decomposition* (SVD) is now the standard way to carry out the *Principal Component Analysis* (PCA).

In This Chapter:

Topics	Applications/Properties
Singular Value Decomposition (SVD)	LS problems, pseudoinverse
Principal Component Analysis (PCA)	Feature extraction, data compression

Contents of Chapter 9

9.1. Singular Value Decomposition 324
9.2. Application of the SVD for LS Problems 335
9.3. Principal Component Analysis 338
Exercises for Chapter 9 344

9.1. Singular Value Decomposition

Here we first deal with the **singular value decomposition** (**SVD**).

Theorem 9.1. (SVD Theorem). *Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. Then we can write*

$$A = U\,\Sigma\,V^T, \tag{9.1}$$

where $U \in \mathbb{R}^{m\times n}$ and satisfies $U^TU = I$, $V \in \mathbb{R}^{n\times n}$ and satisfies $V^TV = I$, and $\Sigma = diag(\sigma_1, \sigma_2, \cdots, \sigma_n)$, where

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0.$$

Remark 9.2. The matrices are illustrated pictorially as

$$\begin{bmatrix} \\ A \\ \\ \end{bmatrix} = \begin{bmatrix} \\ U \\ \\ \end{bmatrix} \begin{bmatrix} \\ \Sigma \\ \\ \end{bmatrix} \begin{bmatrix} \\ V^T \\ \\ \end{bmatrix}, \tag{9.2}$$

where

$$\begin{aligned} U &: m \times n \text{ orthogonal (the } \textbf{left singular vectors} \text{ of } A.) \\ \Sigma &: n \times n \text{ diagonal (the } \textbf{singular values} \text{ of } A.) \\ V &: n \times n \text{ orthogonal (the } \textbf{right singular vectors} \text{ of } A.) \end{aligned}$$

- For some $r \leq n$, the singular values may satisfy

$$\underbrace{\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r}_{\text{nonzero singular values}} > \sigma_{r+1} = \cdots = \sigma_n = 0. \tag{9.3}$$

  In this case, $rank\,(A) = r$.
- If $m < n$, the **SVD** is defined by considering A^T.

Proof. (of Theorem 9.1) Use induction on m and n: we assume that the SVD exists for $(m-1)\times(n-1)$ matrices, and prove it for $m\times n$. We assume $A\neq 0$; otherwise we can take $\Sigma = 0$ and let U and V be arbitrary orthogonal matrices.

- The basic step occurs when $n = 1$ ($m \geq n$). We let $A = U\Sigma V^T$ with $U = A/||A||_2$, $\Sigma = ||A||_2$, $V = 1$.
- For the induction step, choose $\mathbf{v}$ so that

$$||\mathbf{v}||_2 = 1 \text{ and } ||A||_2 = ||A\mathbf{v}||_2 > 0.$$

- Let $\mathbf{u} = \dfrac{A\mathbf{v}}{||A\mathbf{v}||_2}$, which is a unit vector. Choose $\widetilde{U}$, $\widetilde{V}$ such that

$$U = [\mathbf{u}\ \ \widetilde{U}] \in \mathbb{R}^{m\times n} \text{ and } V = [\mathbf{v}\ \ \widetilde{V}] \in \mathbb{R}^{n\times n}$$

  are orthogonal.
- Now, we write

$$U^TAV = \begin{bmatrix} \mathbf{u}^T \\ \widetilde{U}^T \end{bmatrix} \cdot A \cdot [\mathbf{v}\ \ \widetilde{V}] = \begin{bmatrix} \mathbf{u}^TA\mathbf{v} & \mathbf{u}^TA\widetilde{V} \\ \widetilde{U}^TA\mathbf{v} & \widetilde{U}^TA\widetilde{V} \end{bmatrix}$$

  Since

$$\begin{aligned} \mathbf{u}^TA\mathbf{v} &= \frac{(A\mathbf{v})^T(A\mathbf{v})}{||A\mathbf{v}||_2} = \frac{||A\mathbf{v}||_2^2}{||A\mathbf{v}||_2} = ||A\mathbf{v}||_2 = ||A||_2 \equiv \sigma, \\ \widetilde{U}^TA\mathbf{v} &= \widetilde{U}^T\mathbf{u}||A\mathbf{v}||_2 = 0, \end{aligned}$$

  we have

$$U^TAV = \begin{bmatrix} \sigma & 0 \\ 0 & U_1\Sigma_1V_1^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & U_1 \end{bmatrix} \begin{bmatrix} \sigma & 0 \\ 0 & \Sigma_1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix}^T,$$

  or equivalently

$$A = \left(U \begin{bmatrix} 1 & 0 \\ 0 & U_1 \end{bmatrix}\right) \begin{bmatrix} \sigma & 0 \\ 0 & \Sigma_1 \end{bmatrix} \left(V \begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix}\right)^T. \tag{9.4}$$

Equation (9.4) is our desired decomposition. □

9.1.1. Interpretation of the SVD

Algebraic interpretation of the SVD

Let $\textit{rank}\,(A) = r$. Let the SVD of A be $A = U\,\Sigma\,V^T$, with

$$\begin{aligned} U &= [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_n], \\ \Sigma &= \mathbf{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n), \\ V &= [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n], \end{aligned}$$

and σ_r be the **smallest** positive singular value. Since

$$A = U\,\Sigma\,V^T \iff AV = U\Sigma V^T V = U\Sigma,$$

we have

$$\begin{aligned} AV &= A[\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n] = [A\mathbf{v}_1 \quad A\mathbf{v}_2 \quad \cdots \quad A\mathbf{v}_n] \\ &= [\mathbf{u}_1 \quad \cdots \quad \mathbf{u}_r \quad \cdots \quad \mathbf{u}_n] \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix} \\ &= [\sigma_1\mathbf{u}_1 \quad \cdots \quad \sigma_r\mathbf{u}_r \quad \mathbf{0} \quad \cdots \quad \mathbf{0}]. \end{aligned} \tag{9.5}$$

Therefore,

$$A = U\,\Sigma\,V^T \Leftrightarrow \begin{cases} A\mathbf{v}_j = \sigma_j\mathbf{u}_j, & j = 1, 2, \cdots, r \\ A\mathbf{v}_j = \mathbf{0}, & j = r+1, \cdots, n \end{cases} \tag{9.6}$$

Similarly, starting from $A^T = V\,\Sigma\,U^T$,

$$A^T = V\,\Sigma\,U^T \Leftrightarrow \begin{cases} A^T\mathbf{u}_j = \sigma_j\mathbf{v}_j, & j = 1, 2, \cdots, r \\ A^T\mathbf{u}_j = \mathbf{0}, & j = r+1, \cdots, n \end{cases} \tag{9.7}$$

Summary **9.3.** It follows from (9.6) and (9.7) that

- $(\mathbf{v}_j, \sigma_j^2)$, $j = 1, 2, \cdots, r$, are eigenvector-eigenvalue pairs of $A^T A$.

$$A^T A\, \mathbf{v}_j = A^T(\sigma_j \mathbf{u}_j) = \sigma_j^2 \mathbf{v}_j, \;\; j = 1, 2, \cdots, r. \tag{9.8}$$

  So, the singular values play the role of eigenvalues.

- Similarly, we have

$$AA^T\, \mathbf{u}_j = A(\sigma_j \mathbf{v}_j) = \sigma_j^2 \mathbf{u}_j, \;\; j = 1, 2, \cdots, r. \tag{9.9}$$

- Equation (9.8) gives how to find the **singular values** $\{\sigma_j\}$ and the **right singular vectors** V, while (9.6) shows a way to compute the **left singular vectors** U.

- **(Dyadic decomposition)** The matrix $A \in \mathbb{R}^{m\times n}$ can be expressed as

$$A = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{9.10}$$

  When $\mathit{rank}\,(A) = r \leq n$,

$$A = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{9.11}$$

  This property has been utilized for various approximations and applications, e.g., by dropping singular vectors corresponding to *small* singular values.

Geometric interpretation of the SVD

- The matrix A maps an **orthonormal basis**

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$$

of $\mathbb{R}^n$ onto a new "scaled" **orthogonal basis**

$$\mathcal{B}_2 = \{\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2, \cdots, \sigma_r\mathbf{u}_r\}$$

for a subspace of $\mathbb{R}^m$:

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\} \xrightarrow{A} \mathcal{B}_2 = \{\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2, \cdots, \sigma_r\mathbf{u}_r\} \tag{9.12}$$

- Consider a unit sphere $\mathcal{S}^{n-1}$ in $\mathbb{R}^n$:

$$\mathcal{S}^{n-1} = \left\{\mathbf{x} \middle| \sum_{j=1}^{n} x_j^2 = 1\right\}.$$

Then, $\forall \mathbf{x} \in \mathcal{S}^{n-1}$,

$$\begin{aligned} \mathbf{x} &= x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + \cdots + x_n\mathbf{v}_n \\ A\mathbf{x} &= \sigma_1 x_1\mathbf{u}_1 + \sigma_2 x_2\mathbf{u}_2 + \cdots + \sigma_r x_r\mathbf{u}_r \\ &= y_1\mathbf{u}_1 + y_2\mathbf{u}_2 + \cdots + y_r\mathbf{u}_r, \quad (y_j = \sigma_j x_j) \end{aligned} \tag{9.13}$$

So, we have

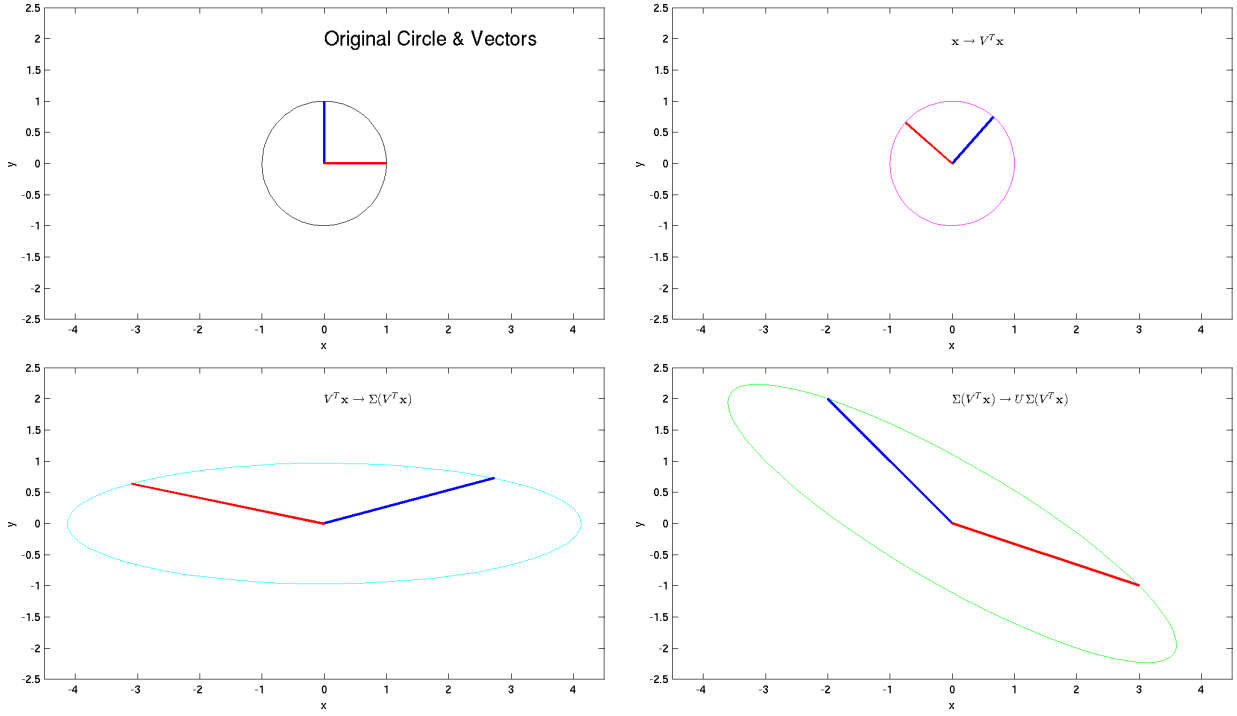
$$\begin{aligned} y_j = \sigma_j x_j \iff x_j = \frac{y_j}{\sigma_j} \\ \sum_{j=1}^{n} x_j^2 = 1 \text{ (sphere)} \iff \sum_{j=1}^{r} \frac{y_j^2}{\sigma_j^2} = \alpha \leq 1 \text{ (ellipsoid)} \end{aligned} \tag{9.14}$$

Example **9.4.** *We build the set $A(\mathcal{S}^{n-1})$ by multiplying one factor of $A = U\Sigma V^T$ at a time. Assume for simplicity that $A \in \mathbb{R}^{2\times 2}$ and nonsingular. Let*

$$\begin{aligned} A &= \begin{bmatrix} 3 & -2 \\ -1 & 2 \end{bmatrix} = U\Sigma V^T \\ &= \begin{bmatrix} -0.8649 & 0.5019 \\ 0.5019 & 0.8649 \end{bmatrix} \begin{bmatrix} 4.1306 & 0 \\ 0 & 0.9684 \end{bmatrix} \begin{bmatrix} -0.7497 & 0.6618 \\ 0.6618 & 0.7497 \end{bmatrix} \end{aligned}$$

Then, for $\mathbf{x} \in \mathcal{S}^1$,

$$A\mathbf{x} = U\Sigma V^T\mathbf{x} = U\left(\Sigma(V^T\mathbf{x})\right)$$



In general,

- $V^T : \mathcal{S}^{n-1} \to \mathcal{S}^{n-1}$ (rotation in $\mathbb{R}^n$)
- $\Sigma : \mathbf{e}_j \mapsto \sigma_j \mathbf{e}_j$ (scaling from $\mathcal{S}^{n-1}$ to $\mathbb{R}^n$)
- $U : \mathbb{R}^n \to \mathbb{R}^m$ (rotation)

9.1.2. Properties and Computation of the SVD

Theorem 9.5. *Let $A \in \mathbb{R}^{m\times n}$ with $m \geq n$. Let $A = U\Sigma V^T$ be the SVD of A, with*

$$\sigma_1 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0.$$

Then,

$$\begin{cases} \textit{rank}\,(A) &= r \\ \textit{Null}(A) &= \textit{Span}\{\mathbf{v}_{r+1}, \cdots, \mathbf{v}_n\} \\ \textit{Range}(A) &= \textit{Span}\{\mathbf{u}_1, \cdots, \mathbf{u}_r\} \\ A &= \displaystyle\sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \end{cases} \tag{9.15}$$

and

$$\begin{cases} ||A||_2 &= \sigma_1 \quad \text{(See Exercise 9.1.)} \\ ||A||_F^2 &= \sigma_1^2 + \cdots + \sigma_r^2 \quad \text{(See Exercise 9.7.)} \\ \displaystyle\min_{\mathbf{x}\neq\mathbf{0}} \frac{||A\mathbf{x}||_2}{||\mathbf{x}||_2} &= \sigma_n \qquad (m \geq n) \end{cases} \tag{9.16}$$

Theorem 9.6. *Let $A \in \mathbb{R}^{m\times n}$ with $\textit{rank}\,(A) = r > 0$. Let $A = U\Sigma V^T$ be the SVD of A, with singular values*

$$\sigma_1 \geq \cdots \geq \sigma_r > 0.$$

Define, for $k = 1, \cdots, r-1$,

$$A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \quad \text{(sum of rank-1 matrices).} \tag{9.17}$$

Then, $\textit{rank}\,(A_k) = k$ and

$$\begin{aligned} ||A - A_k||_2 &= \min\{||A - B||_2 \,\big|\, \textit{rank}\,(B) \leq k\} = \sigma_{k+1}, \\ ||A - A_k||_F^2 &= \min\{||A - B||_F^2 \,\big|\, \textit{rank}\,(B) \leq k\} = \sigma_{k+1}^2 + \cdots + \sigma_r^2. \end{aligned} \tag{9.18}$$

That is, of all matrices of rank $\leq k$, $\boldsymbol{A_k}$ is closest to $\boldsymbol{A}$.

Note: The matrix A_k in (9.17) can be written as

$$A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T = U\,\Sigma_k V^T, \tag{9.19}$$

where $\Sigma_k = \text{diag}(\sigma_1, \cdots, \sigma_k, 0, \cdots, 0)$.

Computation of the SVD

For $A \in \mathbb{R}^{m\times n}$, the procedure is as follows.

1. Form A^TA, the **covariance matrix** of A.
2. Find the eigen-decomposition of A^TA by orthogonalization process, i.e., $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_n)$,

$$A^TA = V\Lambda V^T, \tag{9.20}$$

   where $V = [\mathbf{v}_1 \quad \cdots \quad \mathbf{v}_n]$ is orthogonal, i.e., $V^TV = I$.
3. Sort the eigenvalues according to their magnitude and let

$$\sigma_j = \sqrt{\lambda_j}, \;\; j = 1, 2, \cdots, n. \tag{9.21}$$

4. Form the U matrix as follows ($\textit{rank}\,(A) = k$),

$$\mathbf{u}_j = \frac{1}{\sigma_j} A\mathbf{v}_j, \quad j = 1, 2, \cdots, k. \tag{9.22}$$

Note:

- Note that $\mathbf{u}_j$ are eigenvectors of AA^T, as shown in (9.9). The alternative computation of $\{\mathbf{u}_j\}$ in (9.22) is more efficient.
- If necessary, pick up the remaining columns of U so it becomes orthogonal. These additional columns must be in Null(AA^T).

Example 9.7. *Find the* SVD *for* $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$.

Solution.

1. $A^TA = \begin{bmatrix} 14 & 6 \\ 6 & 9 \end{bmatrix}$.

2. Solving $det\,(A^TA - \lambda I) = 0$ gives the eigenvalues of A^TA

$$\lambda_1 = 18 \text{ and } \lambda_2 = 5,$$

of which corresponding eigenvectors are

$$\widetilde{\mathbf{v}}_1 = \begin{bmatrix} 3 \\ 2 \end{bmatrix}, \widetilde{\mathbf{v}}_2 = \begin{bmatrix} -2 \\ 3 \end{bmatrix}. \Longrightarrow V = \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{2}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

3. $\sigma_1 = \sqrt{\lambda_1} = \sqrt{18} = 3\sqrt{2}$, $\sigma_2 = \sqrt{\lambda_2} = \sqrt{5}$. So

$$\Sigma = \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix}$$

4. $\mathbf{u}_1 = \frac{1}{\sigma_1}A\mathbf{v}_1 = \frac{1}{\sqrt{18}}A \begin{bmatrix} \frac{3}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} \end{bmatrix} = \frac{1}{\sqrt{18}}\frac{1}{\sqrt{13}} \begin{bmatrix} 7 \\ -4 \\ 13 \end{bmatrix} = \begin{bmatrix} \frac{7}{\sqrt{234}} \\ -\frac{4}{\sqrt{234}} \\ \frac{13}{\sqrt{234}} \end{bmatrix}$

$\mathbf{u}_2 = \frac{1}{\sigma_2}A\mathbf{v}_2 = \frac{1}{\sqrt{5}}A \begin{bmatrix} \frac{-2}{\sqrt{13}} \\ \frac{3}{\sqrt{13}} \end{bmatrix} = \frac{1}{\sqrt{5}}\frac{1}{\sqrt{13}} \begin{bmatrix} 4 \\ 7 \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{4}{\sqrt{65}} \\ \frac{7}{\sqrt{65}} \\ 0 \end{bmatrix}$.

5. $A = U\Sigma V^T = \begin{bmatrix} \frac{7}{\sqrt{234}} & \frac{4}{\sqrt{65}} \\ -\frac{4}{\sqrt{234}} & \frac{7}{\sqrt{65}} \\ \frac{13}{\sqrt{234}} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \\ -\frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$

Computer implementation [16, 17]

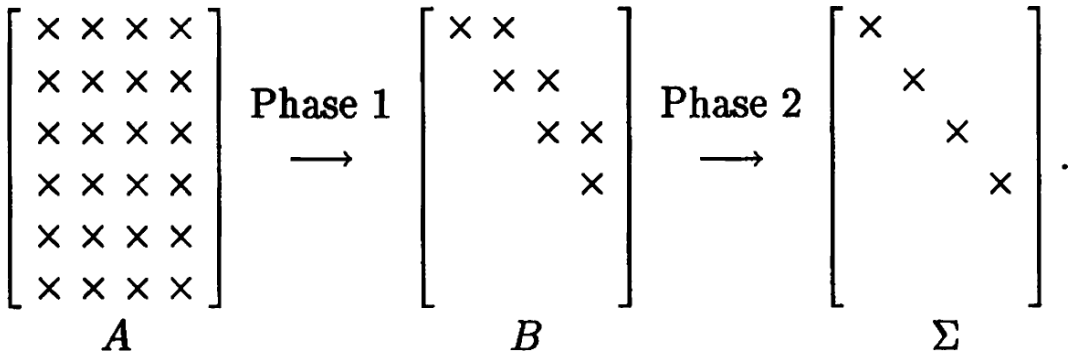


Figure 9.1: A two-phase procedure for the SVD: $A = U\Sigma V^T$.

Algorithm 9.8. (Golub and Reinsch, 1970) [17]. Let $A \in \mathbb{R}^{m\times n}$.

- **Phase 1**: It **constructs two finite sequences** of **Householder transformations** to find an **upper bidiagonal matrix**:

$$P_n \cdots P_1 \, A \, Q_1 \cdots Q_{n-2} \;=\; B \tag{9.23}$$

- **Phase 2**: It is to **iteratively diagonalize** B using the **QR algorithm**.

Golub-Reinsch SVD algorithm

- It is extremely stable.
- Computational complexity:
  - Computation of U, V, and Σ: $4m^2n + 8mn^2 + 9n^3$.
  - Computation of V and Σ: $4mn^2 + 8n^3$.
- Phases 1 & 2 take $\mathcal{O}(mn^2)$ and $\mathcal{O}(n^2)$ flops, respectively. (when Phase 2 is done with $\mathcal{O}(n)$ iterations)
- Python: `U,S,V = numpy.linalg.svd(A)`
- Matlab/Maple: `[U,S,V] = svd(A)`
- Mathematica: `{U,S,V} = SingularValueDecomposition[A]`

Numerical rank

In the absence of round-off errors and uncertainties in the data, the SVD reveals the rank of the matrix. Unfortunately the presence of errors makes rank determination problematic. For example, consider

$$A = \begin{bmatrix} 1/3 & 1/3 & 2/3 \\ 2/3 & 2/3 & 4/3 \\ 1/3 & 2/3 & 3/3 \\ 2/5 & 2/5 & 4/5 \\ 3/5 & 1/5 & 4/5 \end{bmatrix} \tag{9.24}$$

- Obviously A is of rank 2, as its third column is the sum of the first two.
- Matlab “svd" (with IEEE double precision) produces
$$\sigma_1 = 2.5987, \quad \sigma_2 = 0.3682, \quad \text{and } \sigma_3 = 8.6614 \times 10^{-17}.$$
- What is the rank of A, 2 or 3? What if σ_3 is in $\mathcal{O}(10^{-13})$?
- For this reason we must introduce a **threshold** T. Then we say that A has **numerical rank** r if A has r singular values larger than T, that is,
$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > T \geq \sigma_{r+1} \geq \cdots \tag{9.25}$$

In Matlab

- Matlab has a “rank" command, which computes the numerical rank of the matrix with a default threshold
$$T = 2 \max\{m, n\} \, \epsilon \, ||A||_2 \tag{9.26}$$
where ϵ is the unit round-off error.
- In Matlab, the unit round-off error can be found from the parameter “eps"
$$\text{eps} = 2^{-52} = 2.2204 \times 10^{-16}.$$
- For the matrix A in (9.24),
$$T = 2 \cdot 5 \cdot \text{eps} \cdot 2.5987 = 5.7702 \times 10^{-15}$$
and therefore `rank(A)=2`.

See Exercise 9.3.

9.2. Application of the SVD for LS Problems

Recall: **(Definition 6.9, p. 227)**: Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, and $\mathbf{b} \in \mathbb{R}^m$. The **least-squares problem** is to find $\widehat{\mathbf{x}} \in \mathbb{R}^n$ which minimizes $\|A\mathbf{x} - \mathbf{b}\|_2$:

$$\begin{aligned} \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2, \\ &\quad \text{or, equivalently,} \\ \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2, \end{aligned} \tag{9.27}$$

where $\widehat{\mathbf{x}}$ is called a **least-squares solution** of $A\mathbf{x} = \mathbf{b}$.

Note: **When A^TA is invertible**, the equation $A\mathbf{x} = \mathbf{b}$ has a unique LS solution for each $\mathbf{b} \in \mathbb{R}^m$ (Theorem 6.14). It can be solved by the **method of normal equations**; the **unique** LS solution $\widehat{\mathbf{x}}$ is given by

$$\widehat{\mathbf{x}} = (A^TA)^{-1}A^T\mathbf{b}. \tag{9.28}$$

Definition 9.9. *$(A^TA)^{-1}A^T$ is called the* **pseudoinverse** *of A. Let $A = U\Sigma V^T$ be the SVD of A. Then*

$$(A^TA)^{-1}A^T = V\Sigma^{-1}U^T \overset{\text{def}}{=\!=} A^+. \tag{9.29}$$

Example 9.10. *Find the* ***pseudoinverse*** *of* $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$.

Solution. From Example 9.7, we have

$$A = U\Sigma V^T = \begin{bmatrix} \frac{7}{\sqrt{234}} & \frac{4}{\sqrt{65}} \\ -\frac{4}{\sqrt{234}} & \frac{7}{\sqrt{65}} \\ \frac{13}{\sqrt{234}} & 0 \end{bmatrix} \begin{bmatrix} \sqrt{18} & 0 \\ 0 & \sqrt{5} \end{bmatrix} \begin{bmatrix} \frac{3}{\sqrt{13}} & \frac{2}{\sqrt{13}} \\ -\frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix}$$

Thus,

$$\begin{aligned} A^+ = V\Sigma^{-1}U^T &= \begin{bmatrix} \frac{3}{\sqrt{13}} & -\frac{2}{\sqrt{13}} \\ \frac{2}{\sqrt{13}} & \frac{3}{\sqrt{13}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{18}} & 0 \\ 0 & \frac{1}{\sqrt{5}} \end{bmatrix} \begin{bmatrix} \frac{7}{\sqrt{234}} & -\frac{4}{\sqrt{234}} & \frac{13}{\sqrt{234}} \\ \frac{4}{\sqrt{65}} & \frac{7}{\sqrt{65}} & 0 \end{bmatrix} \\ &= \begin{bmatrix} -\frac{1}{30} & -\frac{4}{15} & \frac{1}{6} \\ \frac{11}{45} & \frac{13}{45} & \frac{1}{9} \end{bmatrix} \end{aligned}$$

Solving LS Problems by the SVD

Let $A \in \mathbb{R}^{m\times n}$, $m \geq n$, with $\boldsymbol{rank\,(A) = k \leq n}$.

- Suppose that the SVD of A is given, that is,

$$A = U\Sigma V^T.$$

- Since $U^TU = I$, ***U* is ℓ^2-norm preserving**.

$$||A\mathbf{x} - \mathbf{b}|| = ||U\Sigma V^T\mathbf{x} - \mathbf{b}|| = ||\Sigma V^T\mathbf{x} - U^T\mathbf{b}||. \tag{9.30}$$

- Define $\mathbf{z} = V^T\mathbf{x}$ and $\mathbf{c} = U^T\mathbf{b}$. Then

$$||A\mathbf{x} - \mathbf{b}|| = \Big(\sum_{i=1}^{k}(\sigma_i z_i - c_i)^2 + \sum_{i=k+1}^{n} c_i^2\Big)^{1/2}. \tag{9.31}$$

- Thus the norm is minimized when z is chosen with

$$z_i = \begin{cases} c_i/\sigma_i, & \text{when } i \leq k, \\ \text{arbitrary}, & \text{otherwise.} \end{cases} \tag{9.32}$$

- After determining $\mathbf{z}$, one can find the solution as

$$\widehat{\mathbf{x}} = V\mathbf{z}. \tag{9.33}$$

Then the least-squares error reads

$$\min_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}|| = \Big(\sum_{i=k+1}^{n} c_i^2\Big)^{1/2} \tag{9.34}$$

Strategy **9.11.** When z is obtained as in (9.32), it is better to **choose zero** for the "arbitrary" part:

$$\mathbf{z} = [c_1/\sigma_1,\, c_2/\sigma_2,\, \cdots,\, c_k/\sigma_k,\, 0,\, \cdots,\, 0]^T. \tag{9.35}$$

In this case, z can be written as

$$\mathbf{z} \;=\; \Sigma_k^+ \mathbf{c} \;=\; \Sigma_k^+ U^T \mathbf{b}, \tag{9.36}$$

where

$$\Sigma_k^+ \;=\; [1/\sigma_1,\, 1/\sigma_2,\, \cdots,\, 1/\sigma_k,\, 0,\, \cdots,\, 0]^T. \tag{9.37}$$

Thus the corresponding LS solution reads

$$\widehat{\mathbf{x}} \;=\; V\mathbf{z} \;=\; V\Sigma_k^+ U^T \mathbf{b}. \tag{9.38}$$

Note that $\widehat{\mathbf{x}}$ involves **no components of the null space** of A;
$\widehat{\mathbf{x}}$ is unique in this sense.

Remark **9.12.**

- **When *rank* $(A) = k = n$**: It is easy to see that

$$V\Sigma_k^+ U^T = V\Sigma^{-1} U^T, \tag{9.39}$$

  which is the **pseudoinverse** of A.

- **When *rank* $(A) = k < n$**: $A^T A$ is not invertible. However,

$$A_k^+ \;:=\; V\Sigma_k^+ U^T \tag{9.40}$$

  plays the role of the pseudoinverse of A. It is often called the ***k*-th pseudoinverse** of A.

Note: For some LS applications, although *rank* $(A) = n$, the k-th pseudoinverse A_k^+, with a small $k < n$, may give more reliable solutions. Finding such a k is one of interesting research topics in **data analysis**.

9.3. Principal Component Analysis

- **Principal component analysis** (PCA) (a.k.a. **orthogonal linear transformation**) was invented in 1901 by K. Pearson [28], as an analogue of the principal axis theorem in mechanics; it was later independently developed and named by H. Hotelling in the 1930s [19, 20].
- **The PCA is a statistical procedure** that uses an **orthogonal transformation** to convert a set of observations of *possibly correlated variables* into a set of *linearly uncorrelated variables* called the **principal components**.
- The **orthogonal axes** of the new subspace can be interpreted as the **directions of maximum variance** given the constraint that the new feature axes are orthogonal to each other:

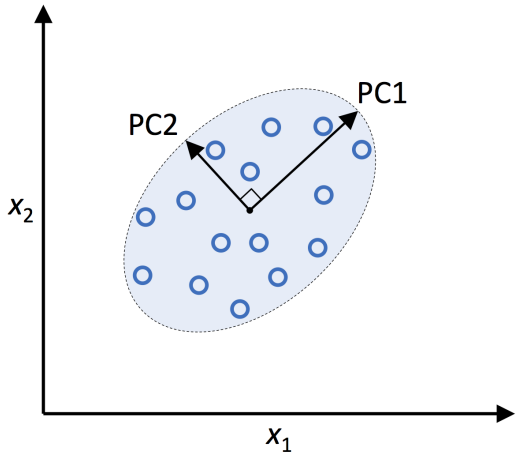


Figure 9.2: Principal components.

- The PCA directions are highly **sensitive to data scaling**. Thus we may need to **standardize** the entries of data matrix (features) prior to the PCA, particularly when
  - **the features were measured on different scales** and/or
  - we want to assign **equal importance** to all features.

9.3.1. Computation of Principal Components

- Consider a **data matrix** $X \in \mathbb{R}^{N\times d}$:
  - each of the N rows represents a different data point,
  - each of the d columns gives a particular kind of feature, and
  - each column has **zero empirical mean.**
- Our goal is to find an **orthogonal weight matrix** $W \in \mathbb{R}^{d\times d}$ such that
$$Z = XW \tag{9.41}$$
maximizes the variance.
- Here $Z \in \mathbb{R}^{N\times d}$ is call the **score matrix**, of which columns represent **principal components** of X.

First weight vector $\mathbf{w}_1$: the first column of W:
In order to maximize variance of $\mathbf{z}_1$, the first weight vector $\mathbf{w}_1$ should satisfy

$$\begin{aligned}\mathbf{w}_1 &= \arg\max_{\|\mathbf{w}\|=1} \|\mathbf{z}_1\|^2 = \arg\max_{\|\mathbf{w}\|=1} \|X\mathbf{w}\|^2 \\ &= \arg\max_{\|\mathbf{w}\|=1} \mathbf{w}^T X^T X\mathbf{w} = \arg\max_{\mathbf{w}\neq 0} \frac{\mathbf{w}^T X^T X\mathbf{w}}{\mathbf{w}^T\mathbf{w}},\end{aligned} \tag{9.42}$$

where the quantity to be maximized can be recognized as a **Rayleigh quotient**.

Theorem 9.13. *For a **positive semidefinite matrix** (such as X^TX), the maximum of the Rayleigh quotient is the same as the largest eigenvalue of the matrix, which occurs when $\mathbf{w}$ is the corresponding eigenvector, i.e.,*

$$\mathbf{w}_1 = \arg\max_{\mathbf{w}\neq 0} \frac{\mathbf{w}^T X^T X\mathbf{w}}{\mathbf{w}^T\mathbf{w}} = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}, \quad (X^TX)\mathbf{v}_1 = \lambda_1\mathbf{v}_1, \tag{9.43}$$

where λ_1 is the largest eigenvalue of $X^TX \in \mathbb{R}^{d\times d}$.

Further weight vectors $\mathbf{w}_k$:

The k-th weight vector can be found by ① subtracting the first $(k-1)$ principal components from X:

$$\widehat{X}_k := X - \sum_{i=1}^{k-1} X\mathbf{w}_i\mathbf{w}_i^T, \tag{9.44}$$

and then ② finding the weight vector which extracts the maximum variance from this new data matrix

$$\mathbf{w}_k = \arg\max_{\|\mathbf{w}\|=1} \|\widehat{X}_k\mathbf{w}\|^2. \tag{9.45}$$

Thus **$\mathbf{w}_k$ is an eigenvector of $\widehat{X}_k^T\widehat{X}_k \Rightarrow$ an eigenvector of X^TX.**

Summary 9.14. The transformation matrix W is the stack of eigenvectors of X^TX, i.e.,

$$W = [\mathbf{w}_1|\mathbf{w}_2|\cdots|\mathbf{w}_d], \quad (X^TX)\,\mathbf{w}_j = \lambda_j\,\mathbf{w}_j, \;\; \mathbf{w}_i^T\mathbf{w}_j = \delta_{ij}, \tag{9.46}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \geq 0$.

With W found, a **data vector** $\mathbf{x}$ is transformed to a d-dimensional row vector of principal components

$$\mathbf{z} = \mathbf{x}W, \tag{9.47}$$

of which components z_j, $j = 1, 2, \cdots, d$, are decorrelated.

Remark 9.15. The principal components transformation is closely related with the **SVD** of X:

$$X = U\,\Sigma V^T, \tag{9.48}$$

where

U : $n \times d$ orthogonal (the **left singular vectors** of X.)
Σ : $d \times d$ diagonal (the **singular values** of X.)
V : $d \times d$ orthogonal (the **right singular vectors** of X.)

- The matrix Σ explicitly reads

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \cdots, \sigma_d), \tag{9.49}$$

  where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d \geq 0$.
- In terms of this factorization, the matrix $X^T X$ reads

$$X^T X = (U\,\Sigma V^T)^T U\,\Sigma V^T = V\,\Sigma U^T U\,\Sigma V^T = V\Sigma^2 V^T. \tag{9.50}$$

- Comparing with the **eigenvector factorization** of $X^T X$, we have
  - the right singular vectors $V \cong$ the eigenvectors of $X^T X \Rightarrow V \cong W$
  - the square of singular values of X are equal to the eigenvalues of $X^T X$ $\Rightarrow \sigma_j^2 = \lambda_j,\ j = 1, 2, \cdots, d$.

Note: Let $W = V$. Then

$$Z = X\,W = U\,\Sigma V^T\,V = U\,\Sigma. \tag{9.51}$$

Claim 9.16. While the **weight matrix** $W \in \mathbb{R}^{d\times d}$ is the collection of eigenvectors of $\boldsymbol{X^T X}$, the **score matrix** $Z \in \mathbb{R}^{N\times d}$ is the stack of eigenvectors of $\boldsymbol{X X^T}$, scaled by the square-root of eigenvalues:

$$Z = [\sqrt{\lambda_1}\,\mathbf{u}_1 | \sqrt{\lambda_2}\,\mathbf{u}_2 | \cdots | \sqrt{\lambda_d}\,\mathbf{u}_d], \quad (XX^T)\,\mathbf{u}_j = \lambda_j\,\mathbf{u}_j, \ \ \mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}. \tag{9.52}$$

9.3.2. Dimensionality Reduction and Data Compression

The transformation $Z = XW$ maps a data vector $\mathbf{x}^{(i)} \in \mathbb{R}^d$ to a new space of d variables which are now uncorrelated. However, **not all the principal components need to be kept.**
Keeping only the first k principal components, produced by using only the first k eigenvectors of $X^T X$ ($k \ll d$), gives the truncated transformation:

$$Z_k = X\, W_k : \mathbf{x}^{(i)} \in \mathbb{R}^d \mapsto \mathbf{z}^{(i)} \in \mathbb{R}^k, \tag{9.53}$$

where $Z_k \in \mathbb{R}^{N\times k}$ and $W_k \in \mathbb{R}^{d\times k}$. Let

$$X_k := Z_k W_k^T. \tag{9.54}$$

Quesitons. How can we choose k? &
Is the difference $\|X - X_k\|$ (that we truncated) small?

Analysis using the SVD

- Let $X = U\,\Sigma V^T$. Then the **score matrix** Z reads

$$Z = XW = U\,\Sigma V^T W = U\,\Sigma, \tag{9.55}$$

  and therefore each column of Z is given by one of the left singular vectors of X multiplied by the corresponding singular value. This form is also the **polar decomposition** of Z. See (9.52).

- As with the eigen-decomposition, the SVD, the **truncated score matrix** $Z_k \in \mathbb{R}^{N\times k}$ can be obtained by considering only the first k largest singular values and their singular vectors:

$$Z_k = XW_k = U\,\Sigma V^T W_k = U\,\Sigma_k, \tag{9.56}$$

  where

$$\Sigma_k := \text{diag}(\sigma_1, \cdots, \sigma_k, 0, \cdots, 0). \tag{9.57}$$

- Now, using (9.56), the truncated data matrix reads

$$X_k = Z_k W_k^T = U\,\Sigma_k W_k^T = U\,\Sigma_k W^T = U\,\Sigma_k V^T. \tag{9.58}$$

Claim **9.17.** It follows from (9.48) and (9.58) that

$$\begin{aligned} \|X - X_k\| &= \|U\,\Sigma V^T - U\,\Sigma_k V^T\| \\ &= \|U(\Sigma - \Sigma_k)V^T\| \\ &= \|\Sigma - \Sigma_k\| \;=\; \sigma_{k+1}, \end{aligned} \tag{9.59}$$

where $\|\cdot\|$ is the induced matrix L^2-norm.

Remark **9.18.** Efficient algorithms exist to calculate the SVD of X without having to form the matrix $X^T X$; see Algorithm 9.8. **Computing the SVD is now the standard way to carry out the PCA** [18, 39].

Exercises for Chapter 9

9.1. Let $A \in \mathbb{R}^{m\times n}$. Prove that $||A||_2 = \sigma_1$, the largest singular value of A.
Hint: Use the following

$$\frac{||A\mathbf{v}_1||_2}{||\mathbf{v}_1||_2} = \frac{\sigma_1||\mathbf{u}_1||_2}{||\mathbf{v}_1||_2} = \sigma_1 \Longrightarrow ||A||_2 \geq \sigma_1.$$

For the opposite directional inequality, you may use arguments around Equations (9.13) and (9.14). In particular, $||A\mathbf{x}||^2 = \sum_{j=1}^{r}(\sigma_j x_j)^2 \leq \sigma_1^2$, for all $||\mathbf{x}|| = 1$. Why?

9.2. Left singular vectors $\mathbf{u}_j$ of A are eigenvectors of AA^T, as shown in (9.9). However, it can be *alternatively and more efficiently* computed as in (9.22):

$$\mathbf{u}_j = \frac{1}{\sigma_j}A\mathbf{v}_j, \quad j = 1, 2, \cdots, k.$$

Prove $\mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}$.

9.3. C[1] Use Matlab to generate a random matrix $A \in \mathbb{R}^{8\times 6}$ with rank 4. For example,

```
A = randn(8,4);
A(:,5:6) = A(:,1:2)+A(:,3:4);
[Q,R] = qr(randn(6));
A = A*Q;
```

(a) Print out A on your computer screen. Can you tell by looking if it has (numerical) rank 4?
(b) Use Matlab's "svd" command to obtain the singular values of A. How many are "large?" How many are "tiny?" (You may use the command "format short e" to get a more accurate view of the singular values.)
(c) Use Matlab's "rank" command to confirm that the numerical rank is 4.
(d) Use the "rank" command with a small enough threshold that it returns the value 6. (Type "help rank" for information about how to do this.)

9.4. C (Continuation of Exercise 8.10) This problem revisits Exercise 6.3, p. 253. Consider the same data

x_i	0.2	0.4	0.6	0.8	1.	1.2	1.4	1.6	1.8	2.
y_i	1.88	2.13	1.76	2.78	3.23	3.82	6.13	7.22	6.66	9.07

(a) Plot the data (scattered point plot) and decide which curve fits the data best.

[1] The mark C indicates that you *should* solve the problem via computer programming. Attach hard copies of your code and results. For other problems, if you like and doable, you may try to solve them with computer programming.

(b) Construct an algebraic system of the form $X\boldsymbol{\beta} = \mathbf{y}$, where $X \in \mathbb{R}^{m\times n}$, $n < m = 10$.

(c) Use **QR decomposition** to find $\widehat{\boldsymbol{\beta}}$ and the curve.
Clue: Let $X = QR$. Then $X\boldsymbol{\beta} = \mathbf{y}$ can be written as $R\boldsymbol{\beta} = Q^T\mathbf{y}$.

(d) Use the **SVD** to find $\widehat{\boldsymbol{\beta}}$ and the curve. (You do not have to implement a code for QR or the SVD; you can use built-in functions.)

(e) Plot the curves superposed over the point plot, and compare them. Are they the same?

CHAPTER 10

Iterative Algebraic Solvers

This chapter is concerned with **iterative algebraic solvers**.

For $A \in \mathbb{R}^{n\times n}$, $\mathbf{b} \in \mathbb{R}^n$, we will learn iterative methods for an approximate solution of

$$A\mathbf{x} = \mathbf{b}. \tag{10.1}$$

Iterative methods we will consider can be formulated as

$$\mathbf{x}^k = \mathbf{x}^{k-1} + G(\mathbf{b} - A\mathbf{x}^{k-1}), \tag{10.2}$$

where $\mathbf{b} - A\mathbf{x}^{k-1} = \mathbf{r}^{k-1}$ is the $(k-1)$st residual and G is an operator which can be either a scalar or a matrix.

Contents of Chapter 10

10.1. Eigenvalues and Convergent Matrices 348
10.2. Basics of Iterative Algebraic Solvers 352
10.3. Relaxation Methods 357
10.4. Graph Theory: Estimation of the Spectral Radius 365
10.5. Krylov Subspace Methods 371
10.6. Generalized Minimum Residuals – GMRES 378
Exercises for Chapter 10 386

10.1. Eigenvalues and Convergent Matrices

Definition **10.1.** Let A be an $n \times n$ matrix. An **eigenvector** of A is a *nonzero* vector x such that

$$A\mathbf{x} = \lambda\mathbf{x} \tag{10.3}$$

for some scalar λ. In this case, a scalar λ is an **eigenvalue** and x is the *corresponding* **eigenvector**.

Definition **10.2.** The scalar equation

$$det\,(A - \lambda I) = 0 \tag{10.4}$$

is called the **characteristic equation** of A; the polynomial $p(\lambda) = det\,(A - \lambda I)$ is called the **characteristic polynomial** of A. The solutions of $det\,(A - \lambda I) = 0$ are the **eigenvalues** of A.

Claim **10.3.** λ is an eigenvalue of A if and only if $det\,(A - \lambda I) = 0$.

Example **10.4.** Find eigenvalues and eigenvectors of $A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$.

Solution.

Example 10.5. Find the eigenvalues and eigenvectors for the matrix

$$A := \begin{bmatrix} 2 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & -1 & 4 \end{bmatrix}$$

Solution.

Maple

with(LinearAlgebra): Eigenvectors = $\begin{bmatrix} 2 \\ 2 \\ 3 \end{bmatrix}, \begin{bmatrix} -2 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$

p := CharacteristicPolynomial(A, lambda); factor(p);

$$-12 + \lambda^3 - 7\lambda^2 + 16\lambda \tag{10.5}$$

Now, let us find them by using pencils.

Remark 10.6. Let A be an $n \times n$ matrix. Then the characteristic equation of A is of the form

$$\begin{aligned} p(\lambda) = \textit{det}\,(A - \lambda I) &= (-1)^n \left(\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0\right) \\ &= (-1)^n \prod_{i=1}^{n} (\lambda - \lambda_i), \end{aligned} \tag{10.6}$$

where some of eigenvalues λ_i can be complex-valued numbers. Thus

$$\textit{det}\, A = p(0) = (-1)^n \prod_{i=1}^{n} (0 - \lambda_i) = \prod_{i=1}^{n} \lambda_i. \tag{10.7}$$

That is, *det* A is the product of all eigenvalues of A.

10.1.1. Spectral radius

Definition 10.7. The **spectral radius** $\rho(A)$ of a matrix A is defined by

$$\rho(A) = \max |\lambda|, \tag{10.8}$$

where λ is an eigenvalue of A.

Theorem 10.8. *If $A \in \mathbb{R}^{n\times n}$, then*

(a) $\|A\|_2 = \sqrt{\rho(A^T A)}$

(b) $\rho(A) \leq \|A\|$*, or any natural matrix norm* $||\cdot||$.

Proof. The proof of part (a) requires more advanced matrix algebra. For part (b), let λ be an eigenvalue of A with its corresponding eigenvector x with $||\mathrm{x}|| = 1$. Then

$$|\lambda| = |\lambda|\,||\mathbf{x}|| = ||\lambda\mathbf{x}|| = ||A\mathbf{x}|| \leq ||A||\,||\mathbf{x}|| = ||A||. \tag{10.9}$$

Thus, the assertion follows. □

10.1.2. Convergent matrices

Definition **10.9.** A matrix $A \in \mathbb{R}^{n\times n}$ is **convergent** if

$$\lim_{k\to\infty}(A^k)_{ij} = 0, \quad \text{for each } 1 \le i, j \le n. \tag{10.10}$$

Example **10.10.** Is $A := \begin{bmatrix} 1/2 & 0 \\ 1/4 & 1/2 \end{bmatrix}$ convergent?

Solution.

$$A^2 = \begin{bmatrix} 1/4 & 0 \\ 1/4 & 1/4 \end{bmatrix} \quad A^3 = \begin{bmatrix} 1/8 & 0 \\ 3/16 & 1/8 \end{bmatrix} \quad A^4 = \begin{bmatrix} 1/16 & 0 \\ 1/8 & 1/16 \end{bmatrix} \quad \cdots$$

In general,

$$A^k = \begin{bmatrix} \dfrac{1}{2^k} & 0 \\ \dfrac{k}{2^{k+1}} & \dfrac{1}{2^k} \end{bmatrix} \to \mathbf{0}.$$

Thus the matrix A is convergent. □

Theorem **10.11.** The following statements are equivalent.

(a) A is convergent.

(b) $\lim_{n\to\infty} ||A^n|| = 0$ for some natural matrix norm.

(c) $\lim_{n\to\infty} ||A^n|| = 0$ for all natural matrix norms.

(d) $\rho(A) < 1$.

(e) $\lim_{n\to\infty} A^n\mathbf{x} = \mathbf{0}$ for all $\mathbf{x} \in \mathbb{R}^n$.

10.2. Basics of Iterative Algebraic Solvers

10.2.1. Basic concepts for iterative solvers

We consider iterative methods in a more general mathematical setting. **A general type of iterative process** for solving the algebraic system

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n\times n} \tag{10.11}$$

can be described as follows.

- **Matrix Splitting**: Split the matrix A as

$$A = M - N, \tag{10.12}$$

where M is an invertible matrix. Then the system (10.11) can be expressed equivalently as

$$A\mathbf{x} = (M-N)\mathbf{x} = \mathbf{b} \;\Rightarrow\; M\mathbf{x} = N\mathbf{x} + \mathbf{b}. \tag{10.13}$$

- **Iterative Algorithm**: Associated with the splitting is an iterative method

$$M\mathbf{x}^k = N\mathbf{x}^{k-1} + \mathbf{b} \;\Rightarrow\; \mathbf{x}^k = M^{-1}N\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \tag{10.14}$$

for a given initialization $\mathbf{x}^0$, $k \geq 1$.

- **Reformulation**: Since $N = M - A$, (10.14) can be rewritten as

$$\begin{aligned} \mathbf{x}^k &= (I - M^{-1}A)\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \\ &\quad \text{or} \\ \mathbf{x}^k &= \mathbf{x}^{k-1} + M^{-1}(\mathbf{b} - A\mathbf{x}^{k-1}) \end{aligned} \tag{10.15}$$

Here, the matrix $I - M^{-1}A$ $(= M^{-1}N)$ is called the **iteration matrix**.

We shall say that the iterative method in (10.15) is **convergent** if it converges for any initial vector $\mathbf{x}^0$. A sequence of vectors $\{\mathbf{x}^1, \mathbf{x}^2, \cdots\}$ will be computed from (10.15), and our objective is to choose M so that these two conditions are met:

1. The sequence $\{\mathbf{x}^k\}$ is easily computed. (Of course, M must be invertible.)
2. The sequence $\{\mathbf{x}^k\}$ converges rapidly to the solution.

Both conditions can be satisfied if ① M **is easy to invert** and ② M^{-1} **approximates** A^{-1} **well**.

Convergence: Rewrite (10.15):

$$\mathbf{x}^k = (I - M^{-1}A)\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \quad (k \geq 1).$$

If the sequence $\{\mathbf{x}^k\}$ converges, say to a vector $\mathbf{x}$, then it follows from (10.15) that

$$\mathbf{x} = (I - M^{-1}A)\mathbf{x} + M^{-1}\mathbf{b} \tag{10.16}$$

Thus, by letting $\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k$, we have

$$\mathbf{e}^k = (I - M^{-1}A)\mathbf{e}^{k-1} \quad (k \geq 1), \tag{10.17}$$

which implies

$$\begin{aligned} ||\mathbf{e}^k|| &\leq ||I - M^{-1}A||\,||\mathbf{e}^{k-1}|| \\ &\leq ||I - M^{-1}A||^2\,||\mathbf{e}^{k-2}|| \\ &\leq \cdots \leq ||I - M^{-1}A||^k\,||\mathbf{e}^0|| \end{aligned} \tag{10.18}$$

Thus, it can be concluded as in the following theorem.

Theorem 10.12. (Sufficient condition for convergence). *If* $||I - M^{-1}A|| = ||M^{-1}N|| < 1$ *for some induced matrix norm, then the sequence produced by (10.15) converges to the solution of* $A\mathbf{x} = \mathbf{b}$ *for any initial vector* $\mathbf{x}^0$.

Note: Let $\delta = ||I - M^{-1}A||$.

- **(Choice of M)** When

$$M^{-1}A \approx I, \text{ or equivalently } M^{-1} \approx A^{-1},$$

the quantity δ will become smaller and therefore the iteration converges faster.

- **(Stopping criterion)** If $\delta = ||I - M^{-1}A|| < 1$, then **it is safe to halt the iterative process when $||\mathbf{x}^k - \mathbf{x}^{k-1}||$ is small.** Indeed, since

$$\mathbf{e}^k = (I - M^{-1}A)\mathbf{e}^{k-1} = (I - M^{-1}A)(\mathbf{e}^k + \mathbf{x}^k - \mathbf{x}^{k-1}),$$

we can obtain

$$||\mathbf{e}^k|| \leq \delta(||\mathbf{e}^k|| + ||\mathbf{x}^k - \mathbf{x}^{k-1}||)$$

which implies

$$||\mathbf{e}^k|| \leq \frac{\delta}{1-\delta}||\mathbf{x}^k - \mathbf{x}^{k-1}||. \tag{10.19}$$

Theorem 10.13. *The iteration (10.15) converges if and only if*

$$\rho(I - M^{-1}A) = \rho(M^{-1}N) < 1. \tag{10.20}$$

Note: An iterative algorithm converges if and only if its iteration matrix is convergent.

- See Theorem (10.11), p.351, for details of matrix convergence.

10.2.2. Richardson method: the simplest iteration

The iterative method is called the **Richardson method** if M is simply chosen to be the identity matrix:

$$M = I \text{ and } N = I - A.$$

In this case, the second equation in (10.15) reads

$$\mathbf{x}^k = \mathbf{x}^{k-1} + (\mathbf{b} - A\mathbf{x}^{k-1}). \tag{10.21}$$

```
Richardson := proc(n, A, b, x, itmax)
    local k, i, r;
    r := Vector(n);
    for k from 1 to itmax do
        for i from 1 to n do
            r[i] := b[i] - add(A[i, j]·x[j], j = 1 ..n);
        end do;
        for i from 1 to n do
            x[i] := x[i] + r[i];
        end do;
        print( `k= ` , k, evalf(x^%T) );
    end do;
end proc:
```

Figure 10.1: A maple implementation for the Richardson method.

Results of Richardson

```
A := 1/6*Matrix([[6, 3, 2], [2, 6, 3], [3, 2, 6]]):
b := 1/6*Vector([11, 11, 11]):
x := Vector([0, 0, 0]):
Richardson(3, A, b, x, 10)
       k=, 1, [1.833333333, 1.833333333, 1.833333333]
       k=, 2, [0.3055555556, 0.3055555556, 0.3055555556]
       k=, 3, [1.578703704, 1.578703704, 1.578703704]
       k=, 4, [0.5177469136, 0.5177469136, 0.5177469136]
       k=, 5, [1.401877572, 1.401877572, 1.401877572]
       k=, 6, [0.6651020233, 0.6651020233, 0.6651020233]
       k=, 7, [1.279081647, 1.279081647, 1.279081647]
       k=, 8, [0.7674319606, 0.7674319606, 0.7674319606]
       k=, 9, [1.193806699, 1.193806699, 1.193806699]
      k=, 10, [0.8384944171, 0.8384944171, 0.8384944171]
```

Eigenvalues of the iteration matrix

```
with(LinearAlgebra):  Id := Matrix(3, shape = identity):
evalf(Eigenvalues(Id - A));
            [         -0.8333333333       ]
            [0.4166666667 - 0.1443375673 I]
            [0.4166666667 + 0.1443375673 I]
```

Note: Eigenvalues of A must be $\begin{bmatrix} 1.833333333 \\ 0.5833333333 + i\,0.1443375673 \\ 0.5833333333 - i\,0.1443375673 \end{bmatrix}$.
Thus all eigenvalues λ of A are in the open disk $\{z \in \mathbb{C} : |z-1| < 1\}$, which is a sufficient and necessary condition for the convergence of the Richardson method.

Generalization of the Richardson method

Consider

$$A\mathbf{x} = \mathbf{b}, \tag{10.22}$$

where some eigenvalues of A are **not** in $\{z \in \mathbb{C} : |z-1| < 1\}$.

- First, scale (10.22) with a constant η:

$$\eta A\mathbf{x} = \eta\mathbf{b}, \tag{10.23}$$

  where all the eigenvalues of ηA are in the open disk.

- Then, apply the Richardson method to (10.23):
  with $M = I$ and $N = I - \eta A$, the iteration reads

$$\begin{aligned} \mathbf{x}^k &= \mathbf{x}^{k-1} + (\eta\mathbf{b} - \eta A\mathbf{x}^{k-1}) \\ &= \mathbf{x}^{k-1} + \eta(\mathbf{b} - A\mathbf{x}^{k-1}). \end{aligned} \tag{10.24}$$

  Thus the Richardson method converges by choosing an appropriate **scaling factor** η.

Self-study **10.14.** Let $A \in \mathbb{R}^{n\times n}$ be a **definite matrix**. Prove that the **generalized Richardson method** is convergent for the solution of $A\mathbf{x} = \mathbf{b}$.

Solution.

10.3. Relaxation Methods

Definition 10.15. A **matrix splitting** is an expression which represents a given matrix as a sum or difference of matrices: $A = M - N$.

Relaxation Methods

- We first express $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ as the matrix sum:

$$A = D - E - F, \tag{10.25}$$

where

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{bmatrix} \qquad -E = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{bmatrix}$$

$$-F = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ 0 & \cdots & 0 & 0 \end{bmatrix}$$

- Then, **relaxation methods** can be formulated by selecting M and N for the matrix splitting. Examples are:

Table 10.1: Common relaxation methods.

	M	N
Jacobi method	D	$E + F$
Gauss-Seidel method	$D - E$	F
SOR method	$\frac{1}{\omega}D - E$, $\omega \in (0, 2)$	$\frac{1-\omega}{\omega}D + F$

Here, **SOR** stands for **successive over relaxation**.

Jacobi Method

The **Jacobi method** is formulated with $M = D$ **and** $N = E + F$:

$$D\mathbf{x}^k = (E + F)\mathbf{x}^{k-1} + \mathbf{b}. \tag{10.26}$$

The i-th component of (10.26) reads

$$a_{ii}x_i^k = b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^{k-1}) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1}), \tag{10.27}$$

or, equivalently,

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k-1} - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\right) \tag{10.28}$$

Example 10.16. Let $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ and $\mathbf{b} := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}$. Use the Jacobi method to find $\mathbf{x}^k$, $k = 1, 2, 3$, beginning from $\mathbf{x}^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

Solution.

Manual checking in Maple

```
x0 := Vector([1,1,1]):
x1 := Vector(3): x2 := Vector(3): x3 := Vector(3):

x1[1] := (-A[1, 2]*x0[2] - A[1, 3]*x0[3] + b[1])/A[1, 1];
x1[2] := (-A[2, 1]*x0[1] - A[2, 3]*x0[3] + b[2])/A[2, 2];
x1[3] := (-A[3, 1]*x0[1] - A[3, 2]*x0[2] + b[3])/A[3, 3];
x1^%T
                              [1, 1, 3]

x2[1] := (-A[1, 2]*x1[2] - A[1, 3]*x1[3] + b[1])/A[1, 1];
x2[2] := (-A[2, 1]*x1[1] - A[2, 3]*x1[3] + b[2])/A[2, 2];
x2[3] := (-A[3, 1]*x1[1] - A[3, 2]*x1[2] + b[3])/A[3, 3];
x2^%T
                              [1, 2, 3]

x3[1] := (-A[1, 2]*x2[2] - A[1, 3]*x2[3] + b[1])/A[1, 1];
```

```
x3[2] := (-A[2, 1]*x2[1] - A[2, 3]*x2[3] + b[2])/A[2, 2];
x3[3] := (-A[3, 1]*x2[1] - A[3, 2]*x2[2] + b[3])/A[3, 3];
x3^%T
                              [3     7]
                              [-, 2, -]
                              [2     2]
```

Note: The true solution is $[2, 3, 4]^T$.

Jacobi: Maple implementation

```
Jacobi := proc(n, A, b, x, tol, itmax)
    local i, j, k, id1, id2, xx;
    xx := Matrix(n, 2);
    for i from 1 to n do
        xx[i, 1] := x[i];
    end do;
    for k from 1 to itmax do
        id1 := modp(k + 1, 2) + 1;
        id2 := modp(k, 2) + 1;
        for i from 1 to n do
            xx[i, id2] := (b[i] - add(A[i, j]·xx[j, id1], j = 1 ..i - 1)
                               - add(A[i, j]·xx[j, id1], j = i + 1 ..n)) / A[i, i];
        end do;
        print( `k= `, k, evalf(xx[1 ..n, id2]^%T) );
    end do;
end proc:
```

Figure 10.2: Maple implementation for the Jacobi method.

Results of Jacobi

```
Jacobi(3, A, b, x0, tol, 10)
                        k=, 1, [1., 1., 3.]
                        k=, 2, [1., 2., 3.]
                k=, 3, [1.500000000, 2., 3.500000000]
            k=, 4, [1.500000000, 2.500000000, 3.500000000]
            k=, 5, [1.750000000, 2.500000000, 3.750000000]
            k=, 6, [1.750000000, 2.750000000, 3.750000000]
            k=, 7, [1.875000000, 2.750000000, 3.875000000]
            k=, 8, [1.875000000, 2.875000000, 3.875000000]
            k=, 9, [1.937500000, 2.875000000, 3.937500000]
           k=, 10, [1.937500000, 2.937500000, 3.937500000]
```

Jacobi: the ℓ^∞-error = 0.0625

Gauss-Seidel Method

The **Gauss-Seidel method** is formulated with $M = D - E$ & $N = F$:

$$(D - E)\mathbf{x}^k = F\mathbf{x}^{k-1} + \mathbf{b}. \tag{10.29}$$

Note that (10.29) can be equivalently written as

$$D\mathbf{x}^k = \mathbf{b} + E\mathbf{x}^k + F\mathbf{x}^{k-1}. \tag{10.30}$$

The i-th component of (10.30) reads

$$a_{ii}x_i^k = b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^k) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1}), \tag{10.31}$$

or, equivalently,

$$x_i^k = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\right) \tag{10.32}$$

The difference is that the GS method utilizes ***updated values***, which makes not only its convergence faster but also its implementation simpler.

Gauss-Seidel (GS): Maple implementation

```
GaussSeidel := proc(n, A, b, x, tol, itmax)
    local i, j, k;
    for k from 1 to itmax do
        for i from 1 to n do
            x[i] := (b[i] - add(A[i,j]·x[j], j = 1..i-1)
                          - add(A[i,j]·x[j], j = i+1..n)) / A[i,i];
        end do;
        print(`k= `, k, evalf(x^%T));
    end do;
end proc:
```

Figure 10.3: Maple implementation for the Gauss-Seidel method.

Results of GaussSeidel

```
GaussSeidel(3, A, b, x0, tol, 10)
                        k=, 1, [1., 1., 3.]
                    k=, 2, [1., 2., 3.500000000]
            k=, 3, [1.500000000, 2.500000000, 3.750000000]
            k=, 4, [1.750000000, 2.750000000, 3.875000000]
            k=, 5, [1.875000000, 2.875000000, 3.937500000]
            k=, 6, [1.937500000, 2.937500000, 3.968750000]
            k=, 7, [1.968750000, 2.968750000, 3.984375000]
            k=, 8, [1.984375000, 2.984375000, 3.992187500]
            k=, 9, [1.992187500, 2.992187500, 3.996093750]
           k=, 10, [1.996093750, 2.996093750, 3.998046875]
```

Gauss-Seidel: the ℓ^∞-error ≈ 0.0039 = 3.9E-3.

Recall: **Jacobi: the ℓ^∞-error = 0.0625**

Note: By comparison with the result of the Jacobi method, we can see Gauss-Seidel (GS) method is faster.

- It is theoretically known that the GS method is **asymptotically twice faster** than the Jacobi method.

Successive Over Relaxation (SOR) Method

The **successive over relaxation (SOR)** is formulated with $M = \frac{1}{\omega}D - E$ **and** $N = \frac{1-\omega}{\omega}D + F$**:**

$$(D - \omega E)\mathbf{x}^k = \big((1-\omega)D + \omega F\big)\mathbf{x}^{k-1} + \omega\mathbf{b}. \tag{10.33}$$

Note that (10.33) can be equivalently written as

$$D\mathbf{x}^k = (1-\omega)D\mathbf{x}^{k-1} + \omega(\mathbf{b} + E\mathbf{x}^k + F\mathbf{x}^{k-1}). \tag{10.34}$$

The i-th component of (10.34) reads

$$a_{ii}x_i^k = (1-\omega)a_{ii}x_i^{k-1} + \omega\Big(b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^k) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1})\Big), \tag{10.35}$$

or, equivalently,

$$\begin{aligned} x_{GS,i}^k &= \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big) \\ x_i^k &= (1-\omega)x_i^{k-1} + \omega x_{GS,i}^k \end{aligned} \tag{10.36}$$

When $\omega = 1$, the SOR becomes the Gauss-Seidel method.

Note: The SOR is an **extrapolation scheme** based on the GS method.

SOR: Maple implementation

```
SOR := proc(n, A, b, x, w, tol, itmax)
    local i, j, k, xGS;
    for k from 1 to itmax do
        for i from 1 to n do
            xGS := (b[i] - add(A[i,j]·x[j], j = 1..i-1)
                         - add(A[i,j]·x[j], j = i+1..n)) / A[i,i];
            x[i] := (1 - w)·x[i] + w·xGS;
        end do;
        print(`k=`, k, evalf(x^%T));
    end do;
end proc:
```

Figure 10.4: Maple implementation for the SOR method.

Results of SOR

```
SOR(3, A, b, x0, 1.2, tol, 10)
              k=, 1, [1.0, 1.000000000, 3.400000000]
          k=, 2, [1.000000000, 2.440000000, 3.784000000]
          k=, 3, [1.864000000, 2.900800000, 3.983680000]
          k=, 4, [1.967680000, 2.990656000, 3.997657600]
          k=, 5, [2.000857600, 3.000977920, 4.001055232]
          k=, 6, [2.000415232, 3.000686694, 4.000200970]
          k=, 7, [2.000328970, 3.000180625, 4.000068180]
          k=, 8, [2.000042580, 3.000030331, 4.000004563]
          k=, 9, [2.000009683, 3.000002482, 4.000000576]
         k=, 10, [1.999999552, 2.999999581, 3.999999633]
```

SOR: the ℓ^∞-error ≈ 0.00000045 = 4.5E-7.

Recall: **Gauss-Seidel: the ℓ^∞-error ≈ 0.0039 = 3.9E-3.**

Recall: **Jacobi: the ℓ^∞-error = 0.0625**

Note: The SOR with $\omega = 1.2$:

- It is much faster than the Jacobi and GS methods.
- **Question**: How can we find the optimal parameter $\widehat{\omega}$?
  We will see it soon.

Convergence Theory

Theorem 10.17. *For and $\mathbf{x}^0 \in \mathbb{R}^n$, the sequence defined by*

$$\mathbf{x}^k = T\,\mathbf{x}^{k-1} + \mathbf{c} \tag{10.37}$$

converges to the unique solution of $\mathbf{x} = T\,\mathbf{x} + \mathbf{c}$ if and only if $\rho(T) < 1$. In this case, the iterates satisfy

$$||\mathbf{x} - \mathbf{x}^k|| \leq ||T||^k\, ||\mathbf{x} - \mathbf{x}^0||. \tag{10.38}$$

For example:

Relaxation method	T (Iteration matrix)
Jacobi method	$T_J = D^{-1}(E+F)$
Gauss-Seidel method	$T_{GS} = (D-E)^{-1}F$
SOR method	$T_{SOR} = (D-\omega E)^{-1}\big[(1-\omega)D + \omega F\big]$

Theorem **10.18. (Stein and Rosenberg, 1948)** *[34]*
One and only one of the following mutually exclusive relations is valid:

1. $\rho(T_J) = \rho(T_{GS}) = 0$
2. $0 < \rho(T_{GS}) < \rho(T_J) < 1$
3. $\rho(T_J) = \rho(T_{GS}) = 1$
4. $1 < \rho(T_J) < \rho(T_{GS})$

Theorem **10.19.** *Let A be symmetric. Then,*

$$\rho(T_{SOR}) < 1 \iff A \text{ is is positive definite and } 0 < \omega < 2. \tag{10.39}$$

Parameter **10.20. (Optimal ω for the SOR).** For algebraic systems of good properties, it is theoretically known that the convergence of the SOR can be optimized when

$$\widehat{\omega} = \frac{2}{1 + \sqrt{1 - \rho(T_J)^2}}. \tag{10.40}$$

However, in many cases, you can find a *practically* better ω for a given algebraic system.

Remark **10.21.**

- When $\omega > 1$, the blending of the SOR, the second equation in (10.36), is an extrapolation. It is how the algorithm is named.
- On the other hand, when $\omega < 1$, the algorithm is also called the **successive under relaxation (SUR)**.

10.4. Graph Theory: Estimation of the Spectral Radius

Definition 10.22. A **permutation matrix** is a square matrix in which each row and each column has one entry of unity, all others zero.

10.4.1. Irreducible matrices

Definition 10.23. For $n \geq 2$, an $n \times n$ complex-valued matrix A is **reducible** if there is a permutation matrix P such that

$$PAP^T = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix}, \tag{10.41}$$

where A_{11} and A_{22} are respectively $r \times r$ and $(n-r) \times (n-r)$ submatrices, $0 < r < n$. If no such permutation matrix exists, then A is **irreducible**.

The operation PAP^T is called a **similarity transformation**. A geometrical interpretation of the irreducibility by means of graph theory is useful.

Geometrical Interpretation of Irreducibility

Directed graph

- Given $A = [a_{ij}] \in \mathbb{C}^{n \times n}$, consider n distinct points
$$P_1, P_2, \cdots, P_n$$
in the plane, which we will call **nodes** or **nodal points**.
- For any nonzero entry a_{ij} of A, we connect P_i to P_j by a path $\overrightarrow{P_iP_j}$, directed from the node P_i to the node P_j; a nonzero a_{ii} is joined to itself by a directed loop, as shown in Figure 10.5.
- In this way, every $n \times n$ matrix A can be associated with a **directed graph** $G(A)$.

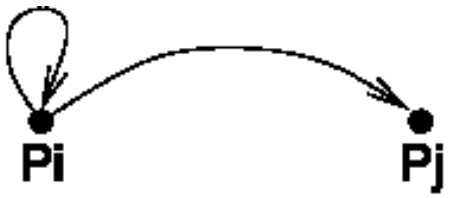


Figure 10.5: The directed paths for nonzero a_{ii} and a_{ij}.

Example **10.24.** For example, the matrix

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \tag{10.42}$$

has a directed graph shown in Figure 10.6.

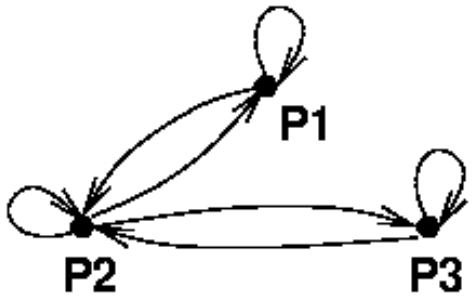


Figure 10.6: The directed graph $G(A)$ for A in (10.42).

Definition **10.25.** A directed graph is **strongly connected** if, for any ordered pair of nodes (P_i, P_j), there is a directed path of a finite length

$$\overrightarrow{P_i P_{k_1}},\ \overrightarrow{P_{k_1} P_{k_2}},\ \cdots,\ \overrightarrow{P_{k_{r-1}} P_{k_r=j}},$$

connecting from P_i to P_j.

The theorems to be presented in this subsection can be found in (Varga, 2000) [37] along with their proofs.

Theorem **10.26.** *An $n \times n$ complex-valued matrix A is irreducible if and only if its directed graph $G(A)$ is strongly connected.*

Note: The operation PAP^T in (10.41) is related to a **point reordering**.

- Thus, if the graph of a matrix is strongly connected, then the re-formed matrix must have a strongly connected graph, no matter how the points are reordered.

10.4.2. Eigenvalue locus theorem

For $A = [a_{ij}] \in \mathbb{C}^{n\times n}$, let

$$\Lambda_i := \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \tag{10.43}$$

Theorem 10.27. (Eigenvalue locus theorem) *Let $A = [a_{ij}]$ be an irreducible $n \times n$ complex matrix. Then,*

1. **(Gerschgorin, 1931) [14]**: *All eigenvalues of A lie in the union of the disks in the complex plane*

$$|z - a_{ii}| \leq \Lambda_i, \quad 1 \leq i \leq n. \tag{10.44}$$

2. **(Taussky, 1948) [35]**: *In addition, assume that λ, an eigenvalue of A, is a boundary point of the union of the disks $|z - a_{ii}| \leq \Lambda_i$. Then, all the n circles $|z - a_{ii}| = \Lambda_i$ must pass through the point λ, i.e., $|\lambda - a_{ii}| = \Lambda_i$ for all $1 \leq i \leq n$.*

Example 10.28. Consider an algebraic system $A\mathrm{x} = \mathrm{b}$, with

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}. \tag{10.45}$$

Estimate spectral radii of iteration matrices, for the Jacobi, the Gauss-Seidel, and the SOR.

Solution. **Jacobi**:

Jacobi

```
M := Matrix([[2, 0, 0], [0, 2, 0], [0, 0, 2]]):
N := M - A:
TJ := M^-1 N
                                  [ 0   1/2   0 ]
                            TJ := [1/2   0   1/2]
                                  [ 0   1/2   0 ]
```

$\Lambda_1 = 1/2$, $\Lambda_2 = 1$, and $\Lambda_3 = 1/2$. Thus $|\lambda_i - 0| = |\lambda_i| < 1$, for $i = 1, 2, 3$.

```
with(LinearAlgebra):
Eigenvalues(TJ)^%T
                    [0    1/sqrt(2)   -1/sqrt(2)]
```

Thus, $\rho(T_J) = 1/\sqrt{2} \approx 0.7071067810$.

Gauss-Seidel:

Gauss-Seidel

```
M := Matrix([[2, 0, 0], [-1, 2, 0], [0, -1, 2]]):
N := M - A:
TGS := M^-1 N
                                  [0   1/2   0 ]
                        TGS := [0   1/4   1/2]
                                  [0   1/8   1/2]
```

$\Lambda_1 = 1/2$, $\Lambda_2 = 1/2$, and $\Lambda_3 = 1/8$:

$$\lambda_i \in \left\{|z-0| < \frac{1}{2}\right\} \cup \left\{|z-\frac{1}{4}| < \frac{1}{2}\right\} \cup \left\{|z-\frac{1}{2}| < \frac{1}{8}\right\}$$

Thus $-\frac{1}{2} < \Re(\lambda_i) < \frac{3}{4}$ and therefore $\rho(T_{GS}) < \frac{3}{4}$.

```
Eigenvalues(TGS)^%T
                        [0   0   1/2]
```

Thus, $\rho(T_{GS}) = 0.5$, **which is exactly the square of** $\rho(T_J)$**.**

SOR

$$M := \begin{bmatrix} 2/w & 0 & 0 \\ -1 & 2/w & 0 \\ 0 & -1 & 2/w \end{bmatrix}: \quad N := M - A: \quad w := \frac{6}{5}:$$

$$TSOR := M^{-1}.N$$

$$TSOR := \begin{bmatrix} -\frac{1}{5} & \frac{3}{5} & 0 \\ -\frac{3}{25} & \frac{4}{25} & \frac{3}{5} \\ -\frac{9}{125} & \frac{12}{125} & \frac{4}{25} \end{bmatrix}$$

$\Lambda_1 = 3/5$, $\Lambda_2 = 18/25$, and $\Lambda_3 = 21/125$.
Thus $-\frac{4}{5} < \Re(\lambda_i) < \frac{22}{25}$ and therefore $\rho(T_{SOR}) < \frac{22}{25}$.

$$Eigenvalues(TSOR)^{\%T} = \begin{bmatrix} -\frac{1}{5} & \frac{4}{25} - \frac{3\,I}{25} & \frac{4}{25} + \frac{3\,I}{25} \end{bmatrix}$$

$$\rho(TSOR) = \max\left(\operatorname{abs}\left(\frac{4}{25} + \frac{3}{25}\,I\right), \frac{1}{5}\right) = \frac{1}{5}$$

Positiveness

Definition 10.29. An $n \times n$ complex-valued matrix $A = [a_{ij}]$ is **diagonally dominant** if

$$|a_{ii}| \geq \Lambda_i := \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \tag{10.46}$$

for all $1 \leq i \leq n$. An $n \times n$ matrix A is *irreducibly diagonally dominant* if A is irreducible and diagonally dominant, with **strict inequality** holding in (10.46) for at least one i.

Theorem 10.30. *Let A be an $n \times n$ strictly or irreducibly diagonally dominant complex-valued matrix. Then, A is nonsingular. If all the diagonal entries of A are in addition positive real, then the real parts of all eigenvalues of A are positive.*

Corollary 10.31. *A Hermitian matrix satisfying the conditions in Theorem 10.30 is positive definite.*

10.4.3. Regular splitting and M-matrices

M-matrices appear in various engineering applications, for which relaxation methods can be employed *effectively*.

Definition 10.32. For $n \times n$ real matrices, A, M, and N, $A = M - N$ is a **regular splitting** of A if M is nonsingular with $M^{-1} \geq 0$, and $N \geq 0$.

Theorem 10.33. *If $A = M - N$ is a regular splitting of A and $A^{-1} \geq 0$, then*

$$\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1 + \rho(A^{-1}N)} < 1. \tag{10.47}$$

Thus, the matrix $M^{-1}N$ is convergent and the iterative method (10.14) converges for any initial value $\mathbf{x}^0$.

Definition 10.34. An $n \times n$ real matrix $A = [a_{ij}]$ with $a_{ij} \leq 0$ for all $i \neq j$ is an **M-matrix** if A is nonsingular and $A^{-1} \geq 0$.

Theorem 10.35. *Let $A = (a_{ij})$ be an $n \times n$ M-matrix. If M is any $n \times n$ matrix obtained by setting certain off-diagonal entries of A to zero, then $A = M - N$ is a regular splitting of A and $\rho(M^{-1}N) < 1$.*

Theorem 10.36. *Let A be an $n \times n$ real matrix with $A^{-1} > 0$, and $A = M_1 - N_1 = M_2 - N_2$ be two regular splittings of A. If $N_2 \geq N_1 \geq 0$, where neither $N_2 - N_1$ nor N_1 is null, then*

$$1 > \rho(M_2^{-1}N_2) > \rho(M_1^{-1}N_1) > 0. \tag{10.48}$$

10.5. Krylov Subspace Methods

Definition 10.37. A matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is said to be **positive definite** if

$$\mathbf{x}^T A\mathbf{x} = \sum_{i,j=1}^{n} x_i a_{ij} x_j > 0, \quad \forall\, \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x} \neq 0. \tag{10.49}$$

For solving a linear system

$$A\mathbf{x} = \mathbf{b}, \tag{10.50}$$

where A is **symmetric positive definite**, Krylov subspace methods update the iterates as follows.

Given an initial guess $\mathbf{x}_0 \in \mathbb{R}^n$, find successive approximations $\mathbf{x}_k \in \mathbb{R}^n$ of the form

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad k = 0, 1, \cdots, \tag{10.51}$$

where $\mathbf{p}_k$ is the *search direction* and $\alpha_k > 0$ is the *step length*.

- Different methods differ in the choice of the search direction and the step length.
- In this section, we consider the **gradient descent method**, the **conjugate gradient (CG) method**, and **preconditioned CG method**.

- In Section 10.6, we will consider the *generalized Minimum Residuals* (GMRES) for the solution of linear systems $A\mathbf{x} = \mathbf{b}$, with **arbitrary nonsingular square matrices** A.
- For other Krylov subspace methods, see e.g. [1, 23].

Remark 10.38. The algebraic system (10.50) admits a unique solution $\mathbf{x} \in \mathbb{R}^n$, which is equivalently characterized by

$$\mathbf{x} = \arg\min_{\eta\in\mathbb{R}^n} f(\eta), \quad f(\eta) = \frac{1}{2}\eta \cdot A\eta - \mathbf{b}\cdot\eta. \tag{10.52}$$

10.5.1. Gradient descent (GD) method

The **gradient descent method** is also known as the **steepest descent method** or the **Richardson's method**.

Derivation of the GD method

- We denote the **gradient** and **Hessian** of f by f' and f'', respectively:

$$f'(\eta) = A\eta - \mathbf{b}, \quad f''(\eta) = A. \tag{10.53}$$

- Given $\mathbf{x}_{k+1}$ as in (10.51), we have by Taylor's formula

$$\begin{aligned} f(\mathbf{x}_{k+1}) &= f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \\ &= f(\mathbf{x}_k) + \alpha_k f'(\mathbf{x}_k) \cdot \mathbf{p}_k + \frac{\alpha_k^2}{2} \mathbf{p}_k \cdot f''(\boldsymbol{\xi}) \mathbf{p}_k, \end{aligned} \tag{10.54}$$

  for some $\boldsymbol{\xi}$.

- Since $f''(\eta)\ (= A)$ is bounded,

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k) + \alpha_k f'(\mathbf{x}_k) \cdot \mathbf{p}_k + \mathcal{O}(\alpha_k^2), \quad \text{as } \alpha_k \to 0.$$

- **The goal** is to find $\mathbf{p}_k$ and α_k such that

$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k), \tag{10.55}$$

  which can be achieved if

$$f'(\mathbf{x}_k) \cdot \mathbf{p}_k < 0 \tag{10.56}$$

  and either γ_k is sufficiently small or $f''(\boldsymbol{\xi})$ is nonnegative.

- **Choice**: When $f'(\mathbf{x}_k) \neq 0$, (10.56) holds, if we choose:

$$\mathbf{p}_k = -f'(\mathbf{x}_k) = \mathbf{b} - A\mathbf{x}_k =: \mathbf{r}_k \tag{10.57}$$

  That is, the search direction is the **negative gradient**, the steepest descent direction.

Optimal step length: We may determine α_k such that

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) = \min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k), \tag{10.58}$$

in which case α_k is said to be **optimal**.

If α_k is optimal, then

$$\begin{aligned} 0 &= \frac{\mathrm{d}}{\mathrm{d}\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k)\Big|_{\alpha=\alpha_k} = f'(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \cdot \mathbf{p}_k \\ &= (A(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \mathbf{b}) \cdot \mathbf{p}_k \\ &= (A\mathbf{x}_k - \mathbf{b}) \cdot \mathbf{p}_k + \alpha_k \mathbf{p}_k \cdot A\mathbf{p}_k. \end{aligned} \tag{10.59}$$

So,

$$\alpha_k = \frac{\mathbf{r}_k \cdot \mathbf{p}_k}{\mathbf{p}_k \cdot A\mathbf{p}_k}. \tag{10.60}$$

Theorem 10.39. (Convergence of the GD method): *The GD method converges, satisfying*

$$\| \mathbf{x} - \mathbf{x}_k \|_2 \leq \left(1 - \frac{1}{\kappa(A)}\right)^k \| \mathbf{x} - \mathbf{x}_0 \|_2. \tag{10.61}$$

Thus, the number of iterations required to reduce the error by a factor of ε is in the order of the condition number of A:

$$k \geq \kappa(A) \log \frac{1}{\varepsilon}. \tag{10.62}$$

10.5.2. Conjugate gradient (CG) method

In this method the search directions $\mathbf{p}_k$ are **conjugate**, i.e.,

$$\mathbf{p}_i \cdot A\mathbf{p}_j = 0, \quad i \neq j, \tag{10.63}$$

and the step length α_k is chosen to be optimal.

Algorithm 10.40. (CG Algorithm, V.1)

$$
\begin{aligned}
&\text{Select } \mathbf{x}_0,\ \varepsilon;\\
&\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0,\ \ \mathbf{p}_0 = \mathbf{r}_0;\\
&\text{Do } k = 0, 1, \cdots\\
&\qquad \alpha_k = \mathbf{r}_k \cdot \mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k; \qquad \text{(CG1)}\\
&\qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k; \qquad \text{(CG2)}\\
&\qquad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k; \qquad \text{(CG3)}\\
&\qquad \text{if } \| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2, \ \text{stop};\\
&\qquad \beta_k = -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k; \qquad \text{(CG4)}\\
&\qquad \mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k; \qquad \text{(CG5)}\\
&\text{End Do}
\end{aligned}
\tag{10.64}
$$

Remark 10.41. (CG Algorithm, V.1)

- In practice, $\boldsymbol{q}_k = A\mathbf{p}_k$ is computed only once, and saved.
- $\mathbf{r}_k = \mathbf{b} - A\mathbf{x}_k$, by definition. So,

$$
\begin{aligned}
\mathbf{r}_{k+1} &= \mathbf{b} - A\mathbf{x}_{k+1} = \mathbf{b} - A(\mathbf{x}_k + \alpha_k \mathbf{p}_k)\\
&= \mathbf{b} - A\mathbf{x}_k - \alpha_k A\mathbf{p}_k = \mathbf{r}_k - \alpha_k A\mathbf{p}_k. \qquad \text{(CG3)}
\end{aligned}
$$

- α_k in (CG1) is optimal as shown in (10.60). Also it satisfies $\mathbf{r}_{k+1} \cdot \mathbf{p}_k = 0$. You may easily verify it using $\mathbf{r}_{k+1}$ in (CG3).
- β_k in (CG4) is determined such that $\mathbf{p}_{k+1} \cdot A\mathbf{p}_k = 0$. You may verify it using $\mathbf{p}_{k+1}$ in (CG5).

Theorem 10.42. *For* $m = 0, 1, \cdots,$

$$
\begin{aligned}
\text{span}\{\mathbf{p}_0, \cdots, \mathbf{p}_m\} &= \text{span}\{\mathbf{r}_0, \cdots, \mathbf{r}_m\}\\
&= \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \cdots, A^m \mathbf{r}_0\}.
\end{aligned}
\tag{10.65}
$$

Theorem 10.43. *The search directions and the residuals satisfy the orthogonality,*

$$
\mathbf{p}_i \cdot A\mathbf{p}_j = 0; \quad \mathbf{r}_i \cdot \mathbf{r}_j = 0, \quad i \neq j. \tag{10.66}
$$

Theorem **10.44.** *For some $m \leq n$, we have $A\mathbf{x}_m = \mathbf{b}$ and*

$$\| \mathbf{x} - \mathbf{x}_k \|_A \leq 2\left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \| \mathbf{x} - \mathbf{x}_0 \|_A. \tag{10.67}$$

So the required iteration number to reduce the error by a factor of ε is

$$k \geq \frac{1}{2}\sqrt{\kappa(A)}\, \log \frac{2}{\varepsilon}. \tag{10.68}$$

Proofs of the above theorems can be found in e.g. [21].

Simplification of the CG method: Using the properties and identities for the method, one can derive a ***more popular form*** of the CG method.

Algorithm **10.45. (CG Algorithm, V.2)**

$$\begin{aligned}
&\text{Select } \mathbf{x}_0, \ \varepsilon; \\
&\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0, \ \ \mathbf{p}_0 = \mathbf{r}_0; \\
&\text{Compute } \rho_0 = \mathbf{r}_0 \cdot \mathbf{r}_0; \\
&\text{Do } k = 0, 1, \cdots \\
&\qquad \alpha_k = \rho_k / \mathbf{p}_k \cdot A\mathbf{p}_k; \\
&\qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k; \\
&\qquad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k; \\
&\qquad \text{if } \| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2, \ \text{stop}; \\
&\qquad \rho_{k+1} = \mathbf{r}_{k+1} \cdot \mathbf{r}_{k+1}; \\
&\qquad \beta_k = \rho_{k+1} / \rho_k; \\
&\qquad \mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k; \\
&\text{End Do}
\end{aligned} \tag{10.69}$$

Note:

$$\begin{aligned}
\mathbf{r}_k \cdot \mathbf{p}_k &= \mathbf{r}_k \cdot (\mathbf{r}_k + \beta_{k-1}\mathbf{p}_{k-1}) = \mathbf{r}_k \cdot \mathbf{r}_k, \\
\beta_k &= -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k = -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k \frac{\alpha_k}{\rho_k} \\
&= \mathbf{r}_{k+1} \cdot (\mathbf{r}_{k+1} - \mathbf{r}_k)/\rho_k = \rho_{k+1}/\rho_k.
\end{aligned} \tag{10.70}$$

Example 10.46. (Revisit to Example 10.16)

Let $A := \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$ and $\mathrm{b} := \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}$. Use the CG method to find x^k, $k = 1, 2, 3$, beginning from $\mathrm{x}^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

Solution.

```
n := 3:
x := x0: r := b - A.x: p := r:
rho0 := r^%T.r:
for k from 1 to n do
    q := A.p;
    al := rho0/(p^%T.q);
    x := x + al*p;
    r := r - al*q;
    rho1 := r.r;
    be := rho1/rho0;
    p := r + be*p;
    rho0 := rho1;
    print(`k=`, k, x, r);
    if (sqrt(rho1) < 10^(-8)) then break; end if;
end do:
```

$$k=,\ 1,\ \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix},\ \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix}$$

$$k=,\ 2,\ \begin{bmatrix} 1 \\ \frac{7}{3} \\ \frac{11}{3} \end{bmatrix},\ \begin{bmatrix} \frac{4}{3} \\ 0 \\ 0 \end{bmatrix}$$

$$k=,\ 3,\ \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix},\ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Remark 10.47.

- For the example, the CG **converged completely** in three iterations.
- The CG method was originally developed **as a direct solver**.

10.5.3. Preconditioned CG method

- The condition number of A, $\kappa(A)$, is the critical factor for the convergence of the CG method.
- If we can find a matrix M such that
$$M \approx A$$
and it is easy to invert, we may try to apply the CG algorithm to the following system
$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{10.71}$$
- Since
$$\kappa(M^{-1}A) \ll \kappa(A) \tag{10.72}$$
(hopefully, $\kappa(M^{-1}A) \approx 1$), the CG algorithm will converge much faster.

In practice, we do not have to multiply M^{-1} to the original algebraic system and the algorithm can be implemented as

Algorithm 10.48. (Preconditioned CG)

$$\begin{aligned}
&\text{Select } \mathbf{x}_0,\ \varepsilon;\\
&\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0,\ \ M\mathbf{z}_0 = \mathbf{r}_0;\\
&\mathbf{p}_0 = \mathbf{z}_0,\ \text{compute } \rho_0 = \mathbf{z}_0^*\mathbf{r}_0;\\
&\text{Do } k = 0, 1, \cdots\\
&\qquad \alpha_k = \rho_k / \mathbf{p}_k^* A\mathbf{p}_k;\\
&\qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k;\\
&\qquad \mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k;\\
&\qquad \text{if } \| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2,\ \text{stop};\\
&\qquad M\mathbf{z}_{k+1} = \mathbf{r}_{k+1};\\
&\qquad \rho_{k+1} = \mathbf{z}_{k+1}^* \mathbf{r}_{k+1};\\
&\qquad \beta_k = \rho_{k+1} / \rho_k;\\
&\qquad \mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k \mathbf{p}_k;\\
&\text{End Do}
\end{aligned} \tag{10.73}$$

Here the superscript * indicates the transpose complex-conjugate; it is the transpose for real-valued systems.

10.6. Generalized Minimum Residuals – GMRES

Note: **GMRES** was developed in 1986 by Saad and Schultz [31].

- The GMRES can be used for linear systems $Ax = b$, with **arbitrary nonsingular square matrices** $A \in \mathbb{R}^{n\times n}$,
  - while application of the classical iterative solvers was limited to either diagonally dominant or positive definite matrices.
- The essential ingredient in *this general solver* is **Arnoldi iteration**.

10.6.1. Arnoldi Iteration

The Arnoldi method will be applicable for both linear systems and eigenvalue problems.

- Consider a matrix decomposition in the form of ***similarity transformations***
$$A = QHQ^T, \tag{10.74}$$
where Q is an orthogonal matrix, $Q^TQ = I$, and H is an **upper Hessenberg matrix**.
- For the similarity transformation, we will use a **modified Gram-Schmidt process**.
  - It is less stable than the Householder reflectors.
  - However, it has the advantage that we can get columns of Q, ***one at a time.***

Derivation of Arnoldi Iteration

- We start with the similarity transformation: for $Q, H \in \mathbb{R}^{n\times n}$,
$$A = QHQ^T.$$
- Clearly, this is equivalent to
$$AQ = QH,$$

which in detail is

$$A[\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \cdots, \boldsymbol{q}_n] = [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_k, \boldsymbol{q}_{k+1}, \cdots, \boldsymbol{q}_n] \times \times \begin{bmatrix} h_{11} & h_{12} & & \cdots & \cdots & h_{1n} \\ h_{21} & h_{22} & & \cdots & \cdots & h_{2n} \\ 0 & h_{32} & h_{33} & \cdots & \cdots & h_{3n} \\ \vdots & 0 & h_{43} & h_{44} & \cdots & h_{4n} \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & h_{n-1,n} & h_{nn} \end{bmatrix}. \tag{10.75}$$

- Next, we consider only part of this system. For $k < n$, let

$$\begin{aligned} Q_k &= [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_k] \\ Q_{k+1} &= [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_k, \boldsymbol{q}_{k+1}] \\ \widetilde{H}_k &= \begin{bmatrix} h_{11} & h_{12} & & \cdots & \cdots & h_{1k} \\ h_{21} & h_{22} & & \cdots & \cdots & h_{2k} \\ 0 & h_{32} & h_{33} & \cdots & \cdots & h_{3k} \\ & 0 & h_{43} & h_{44} & \cdots & h_{4k} \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ & & & 0 & h_{k-1,k} & h_{kk} \\ 0 & \cdots & \cdots & & 0 & h_{k+1,k} \end{bmatrix} \in \mathbb{R}^{k+1,k}. \end{aligned} \tag{10.76}$$

and then take

$$AQ_k = Q_{k+1}\widetilde{H}_k \quad (\in \mathbb{R}^{n\times k}). \tag{10.77}$$

In the following, we will see how to get the last columns of $\widetilde{H}_k$ and Q_{k+1}. The recursive computation of the columns of Q, given as in (10.79), is known as the **Arnoldi iteration**.

- If we compare the kth columns on both sides, then we get

$$A\boldsymbol{q}_k = h_{1k}\boldsymbol{q}_1 + h_{2k}\boldsymbol{q}_2 + \cdots + h_{kk}\boldsymbol{q}_k + h_{k+1,k}\boldsymbol{q}_{k+1}, \tag{10.78}$$

which can be rewritten for $\boldsymbol{q}_{k+1}$:

$$\boldsymbol{q}_{k+1} = \frac{A\boldsymbol{q}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{q}_j}{h_{k+1,k}}, \tag{10.79}$$

which is a **modified Gram-Schmidt process**.

- **Determination of the coefficients, h_{jk}**: It follows from (10.78) that

$$h_{jk} = \boldsymbol{q}_j^T A \boldsymbol{q}_k. \tag{10.80}$$

Since $||\boldsymbol{q}_{k+1}|| = 1$, we must have

$$h_{k+1,k} = \Big\| A\boldsymbol{q}_k - \sum_{j=1}^{k} h_{jk}\boldsymbol{q}_j \Big\|. \tag{10.81}$$

Example 10.49. The first step of the Arnoldi iteration proceeds as follows.

- Start with an arbitrary normalized vector $\boldsymbol{q}_1$. Then, according to (10.79),

$$\boldsymbol{q}_2 = \frac{A\boldsymbol{q}_1 - h_{11}\boldsymbol{q}_1}{h_{21}}. \tag{10.82}$$

- Since it requires $\boldsymbol{q}_2^T \boldsymbol{q}_1 = 0$, orthogonality of the columns of Q, we get

$$0 \;=\; \boldsymbol{q}_1^T A\boldsymbol{q}_1 - h_{11}\boldsymbol{q}_1^T\boldsymbol{q}_1 \;=\; \boldsymbol{q}_1^T A\boldsymbol{q}_1 - h_{11},$$

and therefore h_{11} is a **Rayleigh quotient**:

$$h_{11} \;=\; \boldsymbol{q}_1^T A\boldsymbol{q}_1. \tag{10.83}$$

- Now, we set

$$h_{21} = ||A\boldsymbol{q}_1 - h_{11}\boldsymbol{q}_1||, \tag{10.84}$$

which finalizes $\widetilde{H}_1$ and $\boldsymbol{q}_2$. □

Algorithm 10.50. Arnoldi iteration

0. Let b be an arbitrary initial nonzero vector
1. $\boldsymbol{q}_1 = \mathrm{b}/||\mathrm{b}||$
2. For $k = 1, 2, 3, \cdots$
   (a) $\mathbf{v} = A\boldsymbol{q}_k$
   (b) For $j = 1 : k$
   $$h_{jk} = \boldsymbol{q}_j^T \mathbf{v}$$
   $$\mathbf{v} = \mathbf{v} - h_{jk}\boldsymbol{q}_j$$
   (c) $h_{k+1,k} = ||\mathbf{v}||$
   (d) $\boldsymbol{q}_{k+1} = \mathbf{v}/h_{k+1,k}$

10.6.2. Derivation of the GMRES Method

Key Idea **10.51.** The main idea of the GMRES is to solve a **least-squares problem** at each step of the iteration. At step k:

We find the kth iterate $\mathbf{x}_k \in \mathcal{K}_k$ which minimizes the residual

$$\mathbf{x}_k = \arg\min_{\mathbf{x}\in\mathcal{K}_k} ||A\mathbf{x} - \mathbf{b}||, \tag{10.85}$$

where $\mathcal{K}_k$ is the ***k*th-order Krylov subspace**:

$$\mathcal{K}_k = \textit{Span}\{\mathbf{b}, A\mathbf{b}, \cdots, A^{k-1}\mathbf{b}\} \tag{10.86}$$

Note: In order to solve the least-squares problem (10.85):

- We may start with the **Krylov matrix**

$$K_k \overset{\text{def}}{=\!=} [\mathbf{b},\ A\mathbf{b},\ \cdots,\ A^{k-1}\mathbf{b}] \in \mathbb{R}^{n\times k}. \tag{10.87}$$

- Then the desired solution $\mathbf{x}_k \in \mathcal{K}_k$ can be expressed as

$$\mathbf{x}_k = K_k\mathbf{c}, \ \text{ for some } \mathbf{c} \in \mathbb{R}^k, \tag{10.88}$$

  and therefore the least-squares problem reads

$$\min_{\mathbf{c}\in\mathbb{R}^k} ||AK_k\mathbf{c} - \mathbf{b}||. \tag{10.89}$$

- The problem (10.89) can be solved using the QR-factorization of the matrix $A\mathcal{K}_k \in \mathbb{R}^{n\times k}$;

However this is **both unstable and too expensive**.

- We consider an **orthonormal basis for the Krylov subspace $\mathcal{K}_k$:**

$$\{\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_k\} \tag{10.90}$$

the columns of the matrix Q_k found in the **Arnoldi iteration**.

- With the new basis, the kth iterate $\mathbf{x}_k \in \mathcal{K}_k$ can be written as

$$\mathbf{x}_k = Q_k \mathbf{y}, \quad \text{for some } \mathbf{y} \in \mathbb{R}^k. \tag{10.91}$$

Then the residual minimization problem (10.85) becomes

$$\min_{\mathbf{y} \in \mathbb{R}^k} ||AQ_k\mathbf{y} - \mathbf{b}||. \tag{10.92}$$

- Now, we utilize the **partial similarity transform** of the Arnoldi iteration (10.77):

$$AQ_k = Q_{k+1}\widetilde{H}_k, \quad \widetilde{H}_k \in \mathbb{R}^{(k+1)\times k}. \tag{10.93}$$

- Then, due to the **norm-preserving property** of Q_{k+1}, the least-squares problem (10.92) reads

$$\min_{\mathbf{y} \in \mathbb{R}^k} \left\| \widetilde{H}_k\mathbf{y} - Q_{k+1}^T\mathbf{b} \right\|. \tag{10.94}$$

- The above is equivalent to

$$\min_{\mathbf{y} \in \mathbb{R}^k} \left\| \widetilde{H}_k\mathbf{y} - ||\mathbf{b}||\mathbf{e}_1 \right\|. \tag{10.95}$$

Note: **This is much simpler**; it will permit a more efficient solution.

Example 10.52. Prove that $Q_{k+1}^T \mathbf{b} = ||\mathbf{b}||\mathbf{e}_1$.

Proof. The vector $Q_{k+1}^T \mathbf{b}$ is given by

$$Q_{k+1}^T \mathbf{b} = \begin{bmatrix} \boldsymbol{q}_1^T \mathbf{b} \\ \boldsymbol{q}_2^T \mathbf{b} \\ \vdots \\ \boldsymbol{q}_{k+1}^T \mathbf{b} \end{bmatrix}, \tag{10.96}$$

and, in detail, the Krylov subspaces are

$$\begin{aligned} \mathcal{K}_1 &= \textit{Span}\{\mathbf{b}\}, \\ \mathcal{K}_2 &= \textit{Span}\{\mathbf{b},\, A\mathbf{b}\}, \\ &\vdots \end{aligned} \tag{10.97}$$

Since the columns $\{\boldsymbol{q}_j\}$ of Q_k form an orthonormal basis for $\mathcal{K}_k$,

$$\boldsymbol{q}_1 = \frac{\mathbf{b}}{||\mathbf{b}||}, \quad \text{and } \boldsymbol{q}_j^T \mathbf{b} = 0 \text{ for } j > 1. \tag{10.98}$$

It follows from (10.96) and (10.98) that

$$Q_{k+1}^T \mathbf{b} \;=\; ||\mathbf{b}||\mathbf{e}_1, \tag{10.99}$$

which completes the proof. □

The GMRES method is summarized as follows.

Algorithm 10.53. GMRES

1. Let $\boldsymbol{q}_1 = \mathbf{b}/||\mathbf{b}||$
2. For $k = 1, 2, 3, \cdots$
   (a) Perform Step k of the Arnoldi iteration
   (i.e., compute new entries for $\widetilde{H}_k$ and Q_k)
   (b) Find $\mathbf{y}$ that minimizes $\left\|\widetilde{H}_k \mathbf{y} - ||\mathbf{b}||\mathbf{e}_1\right\|$
   (e.g., with QR-factorization)
   (c) Set $\mathbf{x}_k = Q_k \mathbf{y}$

The GMRES can be implemented to start with the initial residual.

Algorithm 10.54. GMRES Implementation

0. $\mathbf{x}_0$ = **arbitrary initial guess**
1. $\mathbf{r} = \mathbf{b} - A\mathbf{x}_0$; $\boldsymbol{q}_1 = \mathbf{r}/||\mathbf{r}||$
2. For $k = 1, 2, 3, \cdots$
   (a) $\mathbf{v} = A\boldsymbol{q}_k$
   (b) For $j = 1 : k$
   $h_{jk} = \boldsymbol{q}_j^T \mathbf{v}$
   $\mathbf{v} = \mathbf{v} - h_{jk}\boldsymbol{q}_j$
   (c) $h_{k+1,k} = ||\mathbf{v}||$ (If $h_{k+1,k} = 0$, skip next line and stop at bottom.)
   (d) $\boldsymbol{q}_{k+1} = \mathbf{v}/h_{k+1,k}$
   (e) Find y that minimizes $\left\|\widetilde{H}_k\mathbf{y} - ||\mathbf{b}||\mathbf{e}_1\right\|$
   (f) $\mathbf{x}_k = Q_k\mathbf{y} + \mathbf{x}_0$

Here is an example code for the GMRES.

GMRES.py

```
import numpy as np

def GMRES(A, b, x0, nmax_iter, restart=None):
    r = b - np.asarray(np.dot(A, x0)).reshape(-1)
    x = []; nmax_iter += 1
    q = [0] * (nmax_iter)
    q[0] = r / np.linalg.norm(r)
    h = np.zeros((nmax_iter + 1, nmax_iter))

    for k in range(nmax_iter):
        y = np.asarray(np.dot(A, q[k])).reshape(-1)
        for j in range(k):
            h[j, k] = np.dot(q[j], y)
            y = y - h[j, k] * q[j]
        h[k + 1, k] = np.linalg.norm(y)
        if (h[k + 1, k] != 0 and k != nmax_iter - 1):
            q[k + 1] = y / h[k + 1, k]
        p = np.zeros(nmax_iter + 1); p[0] = np.linalg.norm(r)
        result = np.linalg.lstsq(h, p, rcond=None)[0]
        xk = np.dot(np.asarray(q).transpose(), result) + x0
        x.append(list(xk))
    return np.array(x)
```

```
23
24 if __name__ == '__main__':
25     A = np.matrix('1,1; 3,-4')
26     b = np.array([3, 2])
27     x0 = np.array([0, 0])
28     nmax_iter = 10
29     x = GMRES(A, b, x0, nmax_iter)
30
31     print(x)
```

Output

```
1  [[0.         0.        ]
2   [1.27388535 0.25477707]
3   [1.34525175 0.21259709]
4   [1.38792563 0.55277043]
5   [1.79232901 0.85794889]
6   [1.88977212 0.95300971]
7   [1.89522528 0.96784706]
8   [2.00186354 1.01185198]
9   [2.0222287  1.01548282]
10  [2.00838746 1.00162003]
11  [2.00797838 0.99981297]]
```

GMRES is implemented in various software packages. For example:

Matlab:	`gmres`
Python:	`scipy.sparse.linalg.gmres`

Exercises for Chapter 10

10.1. Let $A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}$.

(a) Use Gerschgorin-Taussky theorem to find a range of eigenvalues of A.

(b) Is A nonsingular?

10.2. C When the **boundary-value problem**

$$\begin{cases} -u_{xx} = -2, & 0 < x < 4 \\ u_x(0) = 0, & u(4) = 16 \end{cases} \tag{10.100}$$

is discretized by the second-order finite difference method with $h = 1$, the algebraic system reads $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 2 & -2 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ -2 \\ -2 \\ 14 \end{bmatrix} \tag{10.101}$$

and the exact solution is $\mathbf{x} = [0,\ 1,\ 4,\ 9]^T$.

(a) Is A irreducibly diagonally dominant?

(b) Perform 10 iterations of the Jacobi and Gauss-Seidel methods, starting from $\mathbf{x}_0 = [0,\ 0,\ 0,\ 0]^T$.

(c) Try to find the best ω with which the SOR method converges fastest during the first 10 iterations.

(d) Find the spectral radii of the iteration matrices of the Jacobi, the Gauss-Seidel, and the SOR.

10.3. C Symmetrize the algebraic system (10.101) and

(a) Apply the CG method to solve it.

(b) Download a public-domain code for GMRES to solve the algebraic system; compare its performance with that of the CG method.

CHAPTER 11

Mathematical Optimization

In This Chapter:
You will see **key ideas** in **mathematical optimization**.

The Balance - A Lagrangian Flow

YouTube (Acoustic Folk) (Melodic Rap)

Contents of Chapter 11

11.1. Method of Lagrange Multipliers 388
11.2. Optimization: Primal and Dual Problems 391
11.3. Weak Duality, Strong Duality, and Complementary Slackness 396
11.4. Geometric Interpretation of Duality 400
11.5. Quadratic Programming 407
Exercises for Chapter 11 414

11.1. Method of Lagrange Multipliers

11.1.1. Equality Constraints

In this subsection, we briefly consider Lagrange's method to solve the problem of the form

$$\min / \max_{\mathbf{x}} \ f(\mathbf{x}) \quad \text{subj.to} \quad g(\mathbf{x}) = c. \tag{11.1}$$

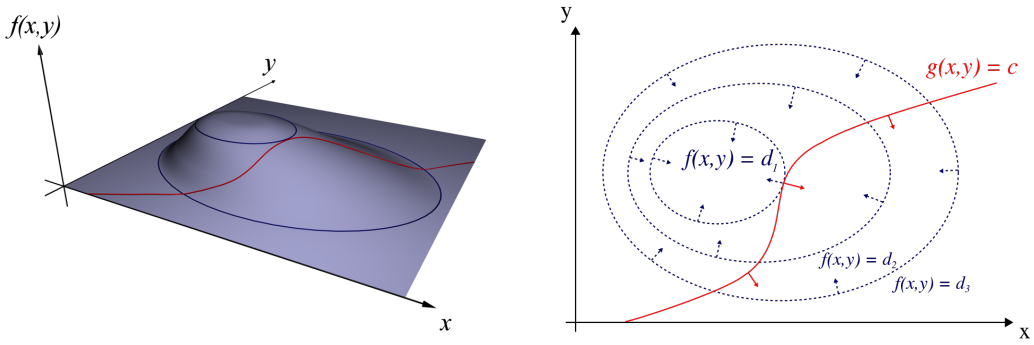


Figure 11.1: The method of Lagrange multipliers in $\mathbb{R}^2$: **∇f // ∇g, at optimum**.

Strategy 11.1. (Method of Lagrange multipliers). For the maximum and minimum values of $f(x)$ **subject to** $g(x) = c$,

(a) Find x and λ such that

$$\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x}) \ \text{ and } \ g(\mathbf{x}) = c. \tag{11.2}$$

(b) Evaluate f at all these points, to find the maximum and minimum.

Note: The first equation in (11.2) can be written as $\nabla \big(f + \lambda(c - g)\big) = 0$.

Self-study 11.2. Use the method of Lagrange multipliers to find the extreme values of $f(x, y) = x^2 + 2y^2$ on the circle $x^2 + y^2 = 1$.

Hint: $\nabla f = \lambda \nabla g \Longrightarrow \begin{bmatrix} 2x \\ 4y \end{bmatrix} = \lambda \begin{bmatrix} 2x \\ 2y \end{bmatrix}$. Therefore, $\begin{cases} 2x = 2x\,\lambda & ① \\ 4y = 2y\,\lambda & ② \\ x^2 + y^2 = 1 & ③ \end{cases}$

From ①, $x = 0$ or $\lambda = 1$.

Ans: min: $f(\pm 1, 0) = 1$; max: $f(0, \pm 1) = 2$

11.1.2. Inequality Constraints

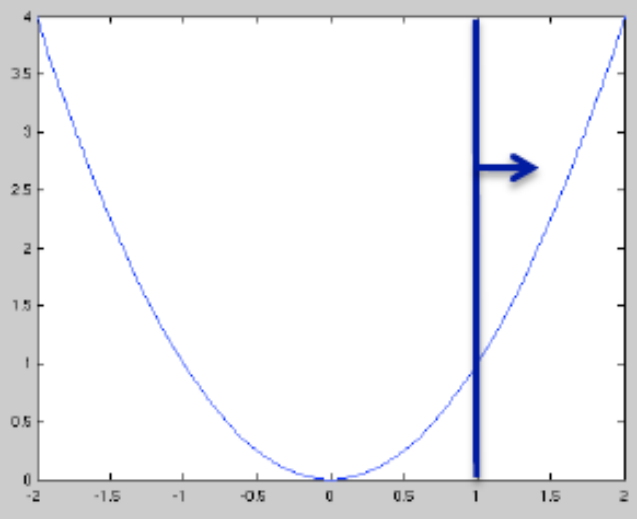


Figure 11.2: $\min_x x^2$ subj.to $x \geq 1$.

For simplicity, consider

$$\min_x x^2 \quad \text{subj.to } x \geq 1. \tag{11.3}$$

Rewriting the constraint

$$x - 1 \geq 0,$$

introduce **Lagrangian (objective)**:

$$\mathcal{L}(x, \alpha) = x^2 - \alpha\,(x - 1). \tag{11.4}$$

Now, consider

$$\min_x \max_\alpha \; \mathcal{L}(x, \alpha) \;\; \text{subj.to } \alpha \geq 0. \tag{11.5}$$

Claim 11.3. The minimization problem (11.3) is equivalent to the max-min problem (11.5).

Proof. ① Let $x > 1$. $\Rightarrow \max_{\alpha \geq 0}\{-\alpha(x-1)\} = 0$ and $\alpha^* = 0$. Thus,

$$\mathcal{L}(x, \alpha) = x^2. \text{ (original objective)}$$

② Let $x = 1$. $\Rightarrow \max_{\alpha \geq 0}\{-\alpha(x-1)\} = 0$ and α is arbitrary. Thus, again,

$$\mathcal{L}(x, \alpha) = x^2. \text{ (original objective)}$$

③ Let $x < 1$. $\Rightarrow \max_{\alpha \geq 0}\{-\alpha(x-1)\} = \infty$. However, $\min_x$ won't make this happen! ($\min_x$ is fighting $\max_\alpha$) That is, when $x < 1$, the objective $\mathcal{L}(x, \alpha)$ becomes huge as α grows; then, $\min_x$ will push $x \nearrow 1$ or increase it to become $x \geq 1$. In other words, **$\min_x$ forces $\max_\alpha$ to behave, so constraints will be satisfied**. □

Now, the goal is **to solve (11.5)**. In the following, we will define the **dual problem** of (11.5), which is equivalent to the **primal problem**.

Recall: The min-max problem in (11.5), which is equivalent to the (original) primal problem:

$$\min_x \max_\alpha \mathcal{L}(x, \alpha) \quad \text{subj.to} \quad \alpha \geq 0, \qquad \text{(Primal)} \tag{11.6}$$

where

$$\mathcal{L}(x, \alpha) = x^2 - \alpha\,(x - 1).$$

Definition 11.4. *The **dual problem** of (11.6) is formulated by swapping* $\min_x$ *and* $\max_\alpha$ *as follows:*

$$\max_\alpha \min_x \mathcal{L}(x, \alpha) \quad \textit{subj.to} \quad \alpha \geq 0, \qquad \textit{(Dual)} \tag{11.7}$$

The term $\min_x \mathcal{L}(x, \alpha)$ *is called the **Lagrange dual function** and the Lagrange multiplier* α *is also called the **dual variable**.*

How to solve it. For the Lagrange dual function $\min_x \mathcal{L}(x, \alpha)$, the minimum occurs where the gradient is equal to zero.

$$\frac{d}{dx}\mathcal{L}(x, \alpha) = 2x - \alpha = 0 \;\Rightarrow\; x = \frac{\alpha}{2}. \tag{11.8}$$

Plugging this to $\mathcal{L}(x, \alpha)$, we have

$$\mathcal{L}(x, \alpha) = \left(\frac{\alpha}{2}\right)^2 - \alpha\left(\frac{\alpha}{2} - 1\right) = \alpha - \frac{\alpha^2}{4}.$$

We can rewrite the dual problem (11.7) as

$$\max_{\alpha \geq 0} \left[\alpha - \frac{\alpha^2}{4}\right]. \qquad \text{(Dual)} \tag{11.9}$$

$\Rightarrow$ **the maximum is 1 when $\alpha^* = 2$ (for the dual problem)**.
Plugging $\alpha = \alpha^*$ into (11.8) to get $x^* = 1$. Or, using the Lagrangian objective, we have

$$\mathcal{L}(x, \alpha) = x^2 - 2(x - 1) = (x - 1)^2 + 1. \tag{11.10}$$

$\Rightarrow$ **the minimum is 1 when $x^* = 1$ (for the primal problem)**. □

11.2. Optimization: Primal and Dual Problems

11.2.1. The Lagrangian

Problem 11.5. Consider a **general optimization problem** of the form

$$\begin{array}{ll} \min\limits_{\mathbf{x}} & f(\mathbf{x}) \\ \text{subj.to} & h_i(\mathbf{x}) \le 0, \quad i = 1, \cdots, m \\ & q_j(\mathbf{x}) = 0, \quad j = 1, \cdots, p \end{array} \qquad \text{(Primal)} \tag{11.11}$$

We define its **Lagrangian** $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ as

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}) + \sum_{j=1}^{p} \beta_j q_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}), \end{aligned} \tag{11.12}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_m) \ge 0$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_p)$ are **Lagrange multipliers**.

Definition 11.6. The set of points that satisfy the constraints,

$$\mathcal{C} \overset{\text{def}}{=\!=} \{\mathbf{x} \in \mathbb{R}^n \mid \boldsymbol{h}(\mathbf{x}) \le 0 \text{ and } \boldsymbol{q}(\mathbf{x}) = 0\}, \tag{11.13}$$

is called the **feasible set**.

Lemma 11.7. For each $\mathbf{x}$ in the feasible set $\mathcal{C}$,

$$f(\mathbf{x}) = \max_{\boldsymbol{\alpha} \ge 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad \mathbf{x} \in \mathcal{C}. \tag{11.14}$$

The maximum is taken iff $\boldsymbol{\alpha}$ satisfies the **complementary slackness**

$$\alpha_i h_i(\mathbf{x}) = 0, \quad i = 1, \cdots, m. \tag{11.15}$$

Proof. When $\mathbf{x} \in \mathcal{C}$, we have $\boldsymbol{h}(\mathbf{x}) \le 0$ and $\boldsymbol{q}(\mathbf{x}) = 0$ and therefore

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}) \le f(\mathbf{x})$$

Clearly, the last inequality becomes equality iff (11.15) holds. □

We will see the details of complementary slackness in ***Section 11.3.3, p.399.***

Remark **11.8.** Recall $\mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathrm{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathrm{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathrm{x})$ and $\mathcal{C} = \{\mathrm{x} \mid \boldsymbol{h}(\mathrm{x}) \leq 0 \text{ and } \boldsymbol{q}(\mathrm{x}) = 0\}$. It is not difficult to see

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \infty, \quad \mathrm{x} \notin \mathcal{C}. \tag{11.16}$$

Theorem **11.9.** *Let f^* be the optimal value of the primal problem (11.11):*

$$f^* = \min_{\mathrm{x} \in \mathcal{C}} f(\mathrm{x}).$$

Then f^* satisfies

$$f^* = \min_{\mathrm{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{11.17}$$

Note: The minimum in (11.17) **does not require** x **in** $\mathcal{C}$.

Proof. For $\mathrm{x} \in \mathcal{C}$, it follows from (11.14) that

$$f^* = \min_{\mathrm{x} \in \mathcal{C}} f(\mathrm{x}) = \min_{\mathrm{x} \in \mathcal{C}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{11.18}$$

When $\mathrm{x} \notin C$, since (11.16) holds, we have

$$\min_{\mathrm{x} \notin \mathcal{C}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \infty. \tag{11.19}$$

The assertion (11.17) follows from (11.18) and (11.19). □

Summary **11.10.** **Primal Problem**

The primal problem (11.11) is equivalent to the **minimax problem**

$$\min_{\mathrm{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \qquad \text{(Primal)} \tag{11.20}$$

where

$$\mathcal{L}(\mathrm{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathrm{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathrm{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathrm{x}).$$

Here the minimum does **not** require x in the feasible set $\mathcal{C}$.

11.2.2. Lagrange Dual Problem

Given a Lagrangian $\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$, we define its **Lagrange dual function** as

$$\begin{aligned} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) &\overset{\text{def}}{=\!=} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ &= \min_{\mathbf{x}} \big\{ f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}) \big\}. \end{aligned} \tag{11.21}$$

Claim 11.11. **Lower Bound Property**

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \le f^*, \quad \text{for } \boldsymbol{\alpha} \ge 0. \tag{11.22}$$

Proof. Let $\boldsymbol{\alpha} \ge 0$. Then for $\mathbf{x} \in \mathcal{C}$,

$$f(\mathbf{x}) \ge \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \ge \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = g(\boldsymbol{\alpha}, \boldsymbol{\beta}).$$

Minimizing over all feasible points $\mathbf{x}$ gives $f^* \ge g(\boldsymbol{\alpha}, \boldsymbol{\beta})$. □

Definition 11.12. Given primal problem (11.11), we define its **Lagrange dual problem** as

$$\begin{aligned} \max_{\boldsymbol{\alpha}, \boldsymbol{\beta}} \min_{\mathbf{x}} \quad & \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ \text{subj.to} \quad & \boldsymbol{\alpha} \ge 0 \end{aligned} \qquad \text{(Dual)} \tag{11.23}$$

Thus the dual problem is a **maximin problem**.

Remark 11.13. It is clear to see from the definition, the optimal value of the dual problem, named as g^*, satisfies

$$g^* = \max_{\boldsymbol{\alpha} \ge 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \le f^*. \tag{11.24}$$

Although the primal problem is not convex, **the dual problem is always convex** (actually, concave).

Theorem 11.14. *The dual problem (11.23) is a* ***convex optimization problem****. Thus* **it is easy to optimize**.

Proof. From the definition,

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} \big\{ f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}) \big\},$$

which can be viewed as pointwise infimum of **affine functions** of α and β. Thus it is concave. Hence the dual problem is a **concave maximization problem**, which is a convex optimization problem. □

Summary 11.15. Given the optimization problem (11.11):

- It is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \qquad \text{(Primal)} \tag{11.25}$$

  where the **Lagrangian** is defined as

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}). \tag{11.26}$$

- Its dual problem is a **maximin problem**

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \qquad \text{(Dual)} \tag{11.27}$$

  and the **dual function** is defined as

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \tag{11.28}$$

- **The Lagrangian and Duality**
  - The **Lagrangian** is a lower bound of the objective function.

$$f(\mathbf{x}) \geq \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \quad \text{for } \mathbf{x} \in \mathcal{C}, \ \boldsymbol{\alpha} \geq 0. \tag{11.29}$$

  - The **dual function** is a lower bound of the primal optimum.

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \leq f^*. \tag{11.30}$$

  - The dual problem is a **convex optimization problem**.

How to Solve the Dual Problem

Recall: **(Definition 11.12)**
Given primal problem (11.11), its **Lagrangian** reads

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x})$$

and the **Lagrange dual problem** is a **maximin problem** defined as

$$\begin{aligned} \max_{\boldsymbol{\alpha},\boldsymbol{\beta}} \min_{\mathbf{x}} \quad & \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \\ \text{subj.to} \quad & \boldsymbol{\alpha} \geq 0 \end{aligned} \qquad \text{(Dual)} \tag{11.31}$$

First, we will try to solve $\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})$. Once optimal point and optimal multipliers $(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ are found, we will have

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta})\big|_{(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta})=(\mathbf{x}^*,\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)} = 0, \tag{11.32}$$

which introduces the **Karush-Kuhn-Tucker (KKT) conditions**, or the **first-derivative tests**.

The KKT Conditions:

$$\begin{aligned} & \nabla f + \nabla \boldsymbol{h}^T \boldsymbol{\alpha} + \nabla \boldsymbol{q}^T \boldsymbol{\beta} = \mathbf{0}, \\ & \boldsymbol{h} \leq \mathbf{0}, \quad \boldsymbol{q} = \mathbf{0}, && \text{(primal feasibility)} \\ & \boldsymbol{\alpha} \cdot \boldsymbol{h} = 0, && \text{(complementary slackness)} \\ & \boldsymbol{\alpha} \geq \mathbf{0}. && \text{(dual feasibility)} \end{aligned} \tag{11.33}$$

There have been many algorithms for the solution of (11.33), which can get $(\mathbf{x}^*, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ either simultaneously or $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ first and then $\mathbf{x}^*$.

We will consider special cases in Section 11.5.

Note: The inequality $\boldsymbol{h} \leq \mathbf{0}$ is a condition for the vector function $\boldsymbol{h} = [h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_m(\mathbf{x})]^T$. We may introduce slack variables $\mathbf{z} = (z_1, z_2, \cdots, z_m) \geq 0$, in order to *equivalently reformulate* the conditions in (11.33):

$$\begin{aligned} & \boldsymbol{h} + \mathbf{z} = \mathbf{0}, \quad \boldsymbol{q} = \mathbf{0}, \\ & \boldsymbol{\alpha} \cdot \mathbf{z} = 0, \\ & \boldsymbol{\alpha} \geq \mathbf{0}, \quad \mathbf{z} \geq \mathbf{0}. \end{aligned} \tag{11.34}$$

11.3. Weak Duality, Strong Duality, and Complementary Slackness

Let's assume we solved the dual problem. Here, a question is

- ***Is the dual optimum the same as the primal optimum?***
- ***If not always, when are they the same?***

Recall: For an **optimization problem** of the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{subj.to} \quad & h_i(\mathbf{x}) \leq 0, \quad i = 1, \cdots, m \\ & q_j(\mathbf{x}) = 0, \quad j = 1, \cdots, p \end{aligned} \qquad \text{(Primal)} \tag{11.35}$$

the **Lagrangian** $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ is defined as

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}) + \sum_{j=1}^{p} \beta_j q_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}), \end{aligned} \tag{11.36}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_m) \geq 0$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_p)$ are **Lagrange multipliers**.

- The problem (11.35) is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \qquad \text{(Primal)} \tag{11.37}$$

- Its dual problem is a **maximin problem**

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \qquad \text{(Dual)} \tag{11.38}$$

  and the **dual function** is defined as

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \overset{\text{def}}{=\!=} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}); \quad g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \leq f^*, \text{ for } \boldsymbol{\alpha} \geq 0. \tag{11.39}$$

11.3.1. Weak Duality

Theorem 11.16. *The dual problem yields a lower bound for the primal problem. That is, the minimax f^* is greater or equal to the maximin g^*:*

$$f^* = \min_{\mathbf{x}} \max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) \geq \max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) = g^* \tag{11.40}$$

Proof. Let $\mathbf{x}^*$ be the minimizer, the primal optimum. Then

$$\mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) \geq \min_{\mathbf{x}} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathcal{L}(\mathbf{x}^*,\boldsymbol{\alpha},\boldsymbol{\beta}), \quad \forall\, \mathbf{x},\ \boldsymbol{\alpha}\geq 0,\ \boldsymbol{\beta}.$$

Let $(\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*)$ be the maximizer, the dual optimum. Then

$$\mathcal{L}(\mathbf{x},\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*) = \max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) \geq \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}), \quad \forall\, \mathbf{x},\ \boldsymbol{\alpha}\geq 0,\ \boldsymbol{\beta}.$$

It follows from the two inequalities that for all $\mathbf{x},\ \boldsymbol{\alpha}\geq 0,\ \boldsymbol{\beta}$,

$$\mathcal{L}(\mathbf{x},\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*) = \max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) \geq \min_{\mathbf{x}} \mathcal{L}(\mathbf{x},\boldsymbol{\alpha},\boldsymbol{\beta}) = \mathcal{L}(\mathbf{x}^*,\boldsymbol{\alpha},\boldsymbol{\beta}). \tag{11.41}$$

Notice that the left side depends on $\mathbf{x}$, while the right side is a function of $(\boldsymbol{\alpha},\ \boldsymbol{\beta})$. The inequality holds true for all $\mathbf{x},\ \boldsymbol{\alpha}\geq 0,\ \boldsymbol{\beta}$.
$\Rightarrow$ We may take $\min_{\mathbf{x}}$ and $\max_{\boldsymbol{\alpha}\geq 0,\boldsymbol{\beta}}$ respectively to the left side and the right side, to conclude (11.40). □

Definition 11.17. **Weak and Strong Duality**

(a) It always holds true that $f^* \geq g^*$, called as **weak duality**.
(b) In some problems, we actually have $f^* = g^*$, which is called **strong duality**.

11.3.2. Strong Duality

Theorem 11.18. Slater's Theorem

If the primal is a **convex** *problem, and there exists at least one strictly feasible* $\widetilde{\mathbf{x}}$*, satisfying the* ***Slater's condition****:*

$$\boldsymbol{h}(\widetilde{\mathbf{x}}) < 0 \quad \textit{and} \quad \boldsymbol{q}(\widetilde{\mathbf{x}}) = 0, \tag{11.42}$$

then strong duality holds.

A conception having a close relationship with strong duality is the duality gap.

Definition 11.19. Given primal feasible $\mathbf{x}$ and dual feasible $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, the quantity

$$f(\mathbf{x}) - g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) - \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{11.43}$$

is called the **duality gap**.

From the weak duality, we have

$$f(\mathbf{x}) - g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \geq f^* - g^* \geq 0$$

Furthermore, we declare a sufficient and necessary condition for duality gap equal to 0.

Proposition 11.20. With $\mathbf{x}$, $(\boldsymbol{\alpha}, \boldsymbol{\beta})$, **the duality gap equals to 0** iff

(a) $\mathbf{x}$ is the primal optimal solution,
(b) $(\boldsymbol{\alpha}, \boldsymbol{\beta})$ is the dual optimal solution, and
(c) the **strong duality holds.**

Proof. From definitions and the weak duality, we have

$$f(\mathbf{x}) \geq f^* \geq g^* \geq g(\boldsymbol{\alpha}, \boldsymbol{\beta}).$$

The duality gap equals to 0, iff the three inequalities become equalities. □

11.3.3. Complementary Slackness

Assume that **strong duality holds**, $\mathbf{x}^*$ is the primal optimum, and $(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$ is the dual optimum. Then

$$\begin{aligned} f(\mathbf{x}^*) = g(\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) &\overset{\text{def}}{=} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*) \\ &= \min_{\mathbf{x}} \left\{ f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i^* h_i(\mathbf{x}) + \sum_{j=1}^{p} \beta_j^* q_j(\mathbf{x}) \right\} \\ &\leq f(\mathbf{x}^*) + \sum_{i=1}^{m} \boldsymbol{\alpha_i^* h_i(\mathbf{x}^*)} + \sum_{j=1}^{p} \boldsymbol{\beta_j^* q_j(\mathbf{x}^*)} \\ &\leq f(\mathbf{x}^*), \end{aligned} \tag{11.44}$$

hence two inequalities hold with equality.

- The primal optimum $\mathbf{x}^*$ minimizes $\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}^*, \boldsymbol{\beta}^*)$.
- The **complementary slackness** holds:

$$\alpha_i^* h_i(\mathbf{x}^*) = 0, \quad \text{for all } i = 1, \cdots, m, \tag{11.45}$$

which implies that

$$\alpha_i^* > 0 \;\Rightarrow\; h_i(\mathbf{x}^*) = 0, \qquad h_i(\mathbf{x}^*) < 0 \;\Rightarrow\; \alpha_i^* = 0. \tag{11.46}$$

Note: **Complementary slackness** says that

- If a dual variable is greater than zero (slack/loose), then the corresponding primal constraint must be an equality (tight.)
- If the primal constraint is slack, then the corresponding dual variable is tight (or zero).

Remark 11.21. Complementary slackness is **key to designing primal-dual algorithms**. The basic idea is

1. Start with a feasible dual solution $\boldsymbol{\alpha}$.
2. Attempt to find primal feasible $\mathbf{x}$ such that $(\mathbf{x}, \boldsymbol{\alpha})$ satisfy complementary slackness.
3. If Step 2 succeeded, we are done; otherwise the misfit on $\mathbf{x}$ gives a way to modify α. Repeat.

11.4. Geometric Interpretation of Duality

Recall: For an **optimization problem** of the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & f(\mathbf{x}) \\ \text{subj.to} \quad & h_i(\mathbf{x}) \le 0, \quad i = 1, \cdots, m \\ & q_j(\mathbf{x}) = 0, \quad j = 1, \cdots, p \end{aligned} \qquad \text{(Primal)} \tag{11.47}$$

the **Lagrangian** $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^p \to \mathbb{R}$ is defined as

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= f(\mathbf{x}) + \sum_{i=1}^{m} \alpha_i h_i(\mathbf{x}) + \sum_{j=1}^{p} \beta_j q_j(\mathbf{x}) \\ &= f(\mathbf{x}) + \boldsymbol{\alpha} \cdot \boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta} \cdot \boldsymbol{q}(\mathbf{x}), \end{aligned} \tag{11.48}$$

where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_m) \ge 0$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_p)$ are **Lagrange multipliers**.

- The problem (11.47) is equivalent to the **minimax problem**

$$\min_{\mathbf{x}} \max_{\boldsymbol{\alpha} \ge 0, \boldsymbol{\beta}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}). \qquad \text{(Primal)} \tag{11.49}$$

- Its dual problem is a **maximin problem**

$$\max_{\boldsymbol{\alpha} \ge 0, \boldsymbol{\beta}} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}), \qquad \text{(Dual)} \tag{11.50}$$

  and the **dual function** is defined as

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \overset{\text{def}}{=\!=} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}); \quad \max_{\boldsymbol{\alpha} \ge 0, \boldsymbol{\beta}} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = g^* \le f^*. \tag{11.51}$$

Definition 11.22. Given a primal problem (11.47), we define its **epigraph (supergraph)** as

$$\mathcal{A} = \{(\mathbf{r}, \mathbf{s}, t) \mid \boldsymbol{h}(\mathbf{x}) \le \mathbf{r}, \ \boldsymbol{q}(\mathbf{x}) = \mathbf{s}, \ f(\mathbf{x}) \le t, \ \text{for some } \mathbf{x} \in \mathbb{R}^n\}. \tag{11.52}$$

Note: f^* can be depicted on the t-axis of $\mathcal{A}$, as a minimum.

Geometric-interpretation **11.23.** **(f^*, $g(\alpha,\beta)$, and g^*)**

(a) **f^* is the lowest projection of the feasible zone of $\mathcal{A}$ to the the t-axis.**

$$\begin{aligned} f^* &= \min\{t \mid \boldsymbol{h}(\mathbf{x}) \le \mathbf{0},\ \boldsymbol{q}(\mathbf{x}) = \mathbf{0},\ f(\mathbf{x}) \le t\} \\ &= \min\{t \mid (\mathbf{0},\mathbf{0},t) \in \mathcal{A}\}. \end{aligned} \tag{11.53}$$

Note: The minimum is taken over the feasible zone of $\mathcal{A}$.

(b) The Lagrange dual function $g(\boldsymbol{\alpha},\boldsymbol{\beta})$ is the intersection of the t-axis and a hyperplane of normal vector $(\boldsymbol{\alpha},\boldsymbol{\beta},1)$:

$$\begin{aligned} g(\boldsymbol{\alpha},\boldsymbol{\beta}) &\overset{\text{def}}{=\!=} \min_{\mathbf{x}} \left\{ f(\mathbf{x}) + \boldsymbol{\alpha}\cdot\boldsymbol{h}(\mathbf{x}) + \boldsymbol{\beta}\cdot\boldsymbol{q}(\mathbf{x}) \right\} \\ &= \min\left\{ (\boldsymbol{\alpha},\boldsymbol{\beta},1)^T(\mathbf{r},\mathbf{s},t) \mid (\mathbf{r},\mathbf{s},t) \in \mathcal{A} \right\}. \end{aligned} \tag{11.54}$$

This is referred to as a **nonvertical supporting hyperplane**, because the last component of the normal vector is nonzero (it is 1).

(c) **g^* is the highest intersection of the t-axis and all nonvertical supporting hyperplanes of $\mathcal{A}$.**

From the geometric interpretation of f^* and g^*, we actually have an equivalent geometric statement of strong duality:

Theorem **11.24.** *The strong duality holds, iff there exists a nonvertical supporting hyperplane of $\mathcal{A}$ passing through $(0,0,f^*)$.*

Proof. From weak duality $f^* \ge g^*$, the intersection of the t-axis and a nonvertical supporting hyperplane cannot exceed $(0,0,f^*)$. The strong duality holds, i.e, $f^* = g^*$, iff $(0,0,f^*)$ is just the highest intersection, meaning that there exists a nonvertical supporting hyperplane of $\mathcal{A}$ passing through $(0,0,f^*)$. □

Example 11.25. Solve a simple inequality-constrained **convex problem**

$$\begin{aligned} \min_{x} \quad & x^2 + 1 \\ \text{subj.to} \quad & x \geq 1. \end{aligned} \tag{11.55}$$

Solution. A code is implemented **to draw a figure**, shown at the end of the solution.

- **Lagrangian**: The inequality constraint can be written as $-x + 1 \leq 0$. Thus the **Lagrangian** reads

$$\begin{aligned} \mathcal{L}(x, \alpha) &= x^2 + 1 + \alpha(-x + 1) = \mathbf{x^2 - \alpha x + \alpha + 1} \\ &= \left(x - \frac{\alpha}{2}\right)^2 - \frac{\alpha^2}{4} + \alpha + 1, \end{aligned} \tag{11.56}$$

  and therefore the **dual function** reads (when $x = \alpha/2$)

$$g(\alpha) = \min_{x} \mathcal{L}(x, \alpha) = -\frac{\alpha^2}{4} + \alpha + 1. \tag{11.57}$$

Remark 11.26. **The Solution of $\min_{x} \mathcal{L}(x, \alpha)$**

- We may obtain it by applying a **calculus technique**:

$$\frac{\partial}{\partial x}\mathcal{L}(x, \alpha) = 2x - \alpha = 0, \tag{11.58}$$

  and therefore $x = \alpha/2$ and (11.57) follows.

  Equation (11.58) is one of the **Karush-Kuhn-Tucker (KKT) conditions**, the **first-order necessary conditions**, which **defines the relationship between the primal variable (x) and the dual variable (α)**.
- Using the KKT condition, (11.57) defines the dual function $g(\alpha)$ as a function of the dual variable (α).
- The dual function $g(\alpha)$ is **concave**, while the Lagrangian is an affine function of α.

- **Epigraph**: For the convex problem (11.55), its epigraph is defined as

$$\mathcal{A} = \{(r, t) \mid -x + 1 \leq r, \;\; x^2 + 1 \leq t, \;\; \text{for } x \in \mathbb{R}\}. \tag{11.59}$$

To find **the edge of the epigraph**, we replace inequalities with equalities:

$$-x + 1 = r, \;\; x^2 + 1 = t \tag{11.60}$$

and define t as a function of r:

$$t = x^2 + 1 = (-r + 1)^2 + 1. \tag{11.61}$$

See Figure 11.3, where the shaded region is the **epigraph** of the problem.

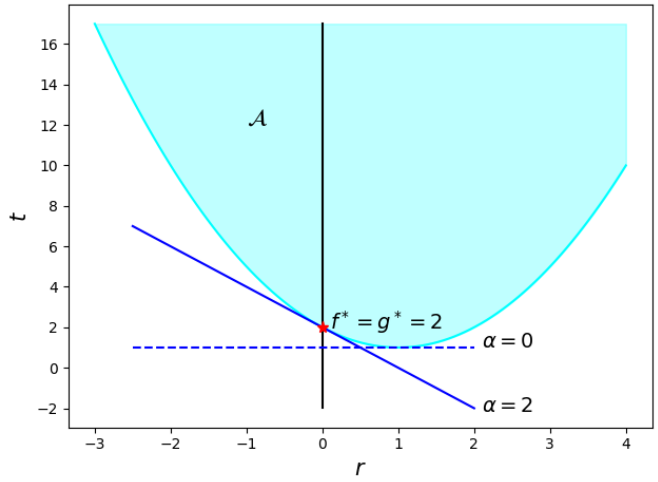


Figure 11.3: The epigraph of the convex problem (11.55), the shaded region, and strong duality.

The Primal Optimum: For a feasible point point x,

$$-x + 1 \leq 0 \;\; \Rightarrow \;\; r = -x + 1 \leq 0.$$

Thus **the left side of the t-axis in $\mathcal{A}$** corresponds to the **feasible set**; it follows from (11.61) that

$$f^* = \min\{t \mid (0, t) \in \mathcal{A}\} = 2, \tag{11.62}$$

which is **the lowest projection of the feasible zone of $\mathcal{A}$** to the t-axis.

- **Nonvertical Supporting Hyperplanes**: For the convex problem, it follows from a Geometric-interpretation (11.54) that

$$g(\alpha) = \min_{(r,t)\in\mathcal{A}} \{\alpha r + t\}. \tag{11.63}$$

For some (r, t), we have

$$g(\alpha) = \alpha r + t.$$

From (11.57), we know $g(\alpha) = -\frac{\alpha^2}{4} + \alpha + 1$. Thus

$$\alpha r + t = -\frac{\alpha^2}{4} + \alpha + 1.$$

Now, we define a family of **nonvertical supporting hyperplanes** as

$$t = -\alpha r - \frac{\alpha^2}{4} + \alpha + 1, \tag{11.64}$$

which is a line in the (r, t)-coordinates for a fixed α. Figure 11.3 depicts two of the lines: $\alpha = 0$ and $\alpha = 2$.

- **Strong Duality**: Note that on the t-axis ($r = 0$), (11.64) reads

$$t = -\frac{\alpha^2}{4} + \alpha + 1 = -\frac{1}{4}(\alpha - 2)^2 + 2, \tag{11.65}$$

of which **the maximum $g^* = 2$ when $\alpha = 2$.**

Thus we can conclude

$$f^* = g^* = 2; \tag{11.66}$$

strong duality holds for the convex problem. □

duality_convex.py

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

# Convex: min f(x), s.t. x >= 1 (i.e., -x+1 <= 0)
def f(x): return x**2+1
def g(r,alpha): return -alpha*r+(-alpha**2/4+alpha+1)

#--- Epigraph: t= f(r)
#-----------------------------------------------
r = np.linspace(-3,4,100); x = -r+1
t = f(x); mint = t.min(); maxt = t.max()

plt.fill_between(r,t,maxt,color='cyan',alpha=0.25)
plt.plot(r,t,color='cyan')
plt.xlabel(r'$r$',fontsize=15); plt.ylabel(r'$t$',fontsize=15)
plt.text(-1,12,r'$\cal A$',fontsize=16)
plt.plot([0,0],[mint-3,maxt],color='black',ls='-')  # t-axis
plt.yticks(np.arange(-2,maxt,2)); plt.tight_layout()

#--- Two Supporting hyperplanes
#-----------------------------------------------
r = np.linspace(-2.5,2,2)
plt.plot(r,g(r,2),color='blue',ls='-')
plt.plot(r,g(r,0),color='blue',ls='--')

#--- Add Texts
#-----------------------------------------------
p=2.1
plt.text(p,g(p,0),r'$\alpha=0$',fontsize=14)
plt.text(p,g(p,2),r'$\alpha=2$',fontsize=14) # the optimum
plt.plot(0,2,'r*',markersize=8)
plt.text(0.1,1.9,r'$f^*=g^*=2$',fontsize=14)

plt.savefig('png-duality-example.png',bbox_inches='tight')
plt.show()
```

Example 11.27. Solve the following **nonconvex problem**

$$\begin{aligned} \min_{x} \quad & x^4 - 50x^2 + 25x \\ \text{subj.to} \quad & x \geq -2. \end{aligned} \tag{11.67}$$

Solution. For this problem, a code must be similar to `duality_convex.py` on p.405.

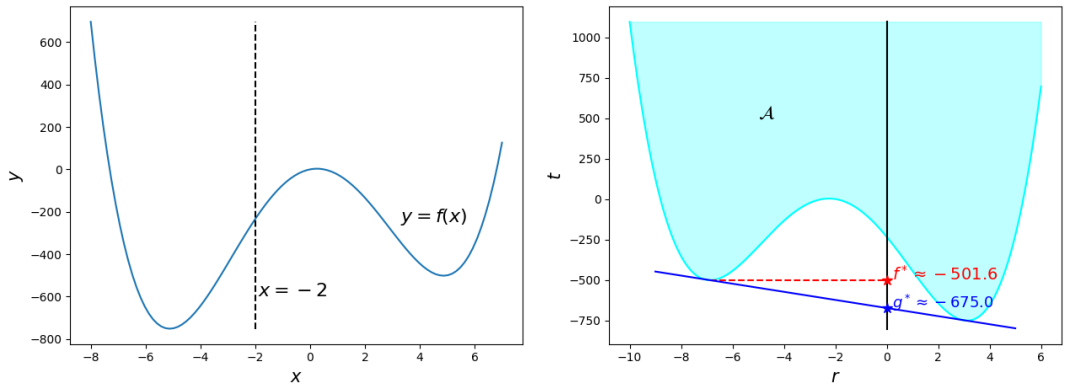


Figure 11.4: The nonconvex problem: (left) The graph of $y = f(x)$ and (right) the epigraph and weak duality.

- The **Lagrangian** of the problem (11.67) reads

$$\mathcal{L}(x, \alpha) = x^4 - 50x^2 + 25x + \alpha(-x - 2); \tag{11.68}$$

  its **epigraph** is defined as

$$\mathcal{A} = \big\{(r, t) \mid -x - 2 \leq r, \;\; x^4 - 50x^2 + 25x \leq t, \;\; \text{for some } x\big\}, \tag{11.69}$$

  which is shown as the cyan-colored region in Figure 11.4.
- The **primal optimum f^*** is obtained by projecting **the negative side of the epigraph** ($r \leq 0$) to the t-axis and taking the minimum, $f^* \approx -501.6$.
- The **dual optimum g^*** is computed as the highest intersection of the t-axis and all nonvertical supporting hyperplanes of $\mathcal{A}$, $g^* \approx -673.4$.
- For the nonconvex problem, there does not exist a supporting hyperplane of $\mathcal{A}$ passing through $(0, f^*)$, thus **strong duality does not hold**.

Note: You will implement a code to verify Example 11.27; see Exercise 11.5.

11.5. Quadratic Programming

Quadratic programming (QP) is the process of solving a *constrained quadratic optimization problem.*

- **The objective f is quadratic** and **the constraints are linear** in several variables $\mathbf{x} \in \mathbb{R}^n$.
- Quadratic programming is a particular type of nonlinear programming. Its **general form** is given as

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}, \quad \text{subj.to} \\ C\mathbf{x} = \mathbf{c}, \\ D\mathbf{x} \leq \mathbf{d}, \end{aligned} \tag{11.70}$$

where $A \in \mathbb{R}^{n\times n}$ is symmetric, $C \in \mathbb{R}^{m\times n}$, $D \in \mathbb{R}^{p\times n}$, $\mathbf{b} \in \mathbb{R}^n$, $\mathbf{c} \in \mathbb{R}^m$, and $\mathbf{d} \in \mathbb{R}^p$.

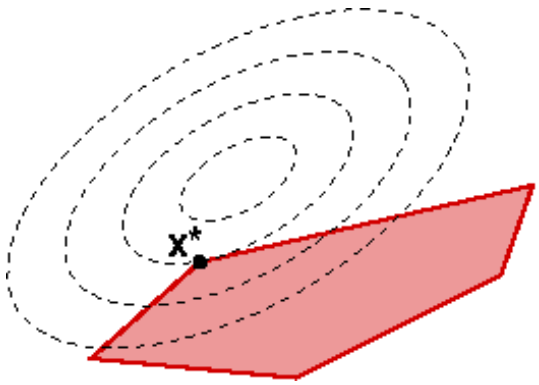


Figure 11.5: A QP objective & constraints (the feasible set).

11.5.1. Equality-constrained quadratic programming

If only **equality constraints** are imposed, the QP (11.70) reduces to

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}, \quad \text{subj.to} \quad C\mathbf{x} = \mathbf{c}, \tag{11.71}$$

where $A \in \mathbb{R}^{n\times n}$, $C \in \mathbb{R}^{m\times n}$, $m \le n$. For the time being we assume that C has full row rank m and $m < n$.

To solve the problem, let's begin with its **Lagrangian**:

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b} + \boldsymbol{\lambda}\cdot(C\mathbf{x} - \mathbf{c}), \tag{11.72}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the the associated Lagrange multiplier.

Then its primal (minimax) and dual (maximin) problems read

$$\begin{aligned} &\min_{\mathbf{x}}\max_{\boldsymbol{\lambda}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}), \quad \text{(Primal)} \\ &\max_{\boldsymbol{\lambda}}\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}). \quad \text{(Dual)} \end{aligned} \tag{11.73}$$

Then, by trying to solve $\min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$ in the dual problm, we obtain

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = A\mathbf{x} - \mathbf{b} + C^T\boldsymbol{\lambda} = 0 \quad \Rightarrow \quad A\mathbf{x} + C^T\boldsymbol{\lambda} = \mathbf{b}. \tag{11.74}$$

The **KKT conditions** (**first-derivative tests**) for the solution $\mathbf{x} \in \mathbb{R}^n$ of (11.71) give rise to the following linear system

$$\underbrace{\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}}_{:=K} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}, \tag{11.75}$$

where the second row is the **primal feasibility**: $C\mathbf{x} = \mathbf{c}$.

Let $Z \in \mathbb{R}^{n\times(n-m)}$ be a matrix whose columns span the null space of C, $\mathcal{N}(C)$, i.e.,

$$CZ = 0 \ \text{ or } \ \mathrm{Span}(Z) = \mathcal{N}(C). \tag{11.76}$$

Definition 11.28. *The matrix K in (11.75) is called the **KKT matrix** and the matrix $Z^T AZ$ is referred to as the **reduced Hessian**.*

Is the KKT Matrix K Nonsingular?

Theorem 11.29. ***(Existence and uniqueness)*** *Assume that $C \in \mathbb{R}^{m\times n}$ has full row rank $m \leq n$, rank $(C) = m$, and that the reduced Hessian $Z^T AZ$ is positive definite. Then, the KKT matrix K is **nonsingular** and therefore the KKT system (11.75) admits a **unique solution** $(\mathrm{x}^*, \boldsymbol{\lambda}^*)$.*

Recall: The KKT system for the solution $\mathbf{x} \in \mathbb{R}^n$ of (11.71) is given by

$$\underbrace{\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix}}_{:=K} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix}, \tag{11.77}$$

where the second row is the primal feasibility.

It is often the case that a direct solution of the KKT system (11.77) is **computationally too costly**; an alternative is to use an iterative method. An iterative solver can be applied

- either to the **entire KKT system**
- or to the **special structure of the KKT matrix**, as in the **range-space approach** and **null-space approach**, and based on **regular splitting**[a] of specifically transformed matrices of K.

The **transforming null-space iteration** does not require regularity of A and therefore has a wider range of applicability than the **transforming range-space iteration**.

[a] For $n \times n$ real matrices, A, M, and N, $A = M - N$ is a **regular splitting** of A if M is nonsingular with $M^{-1} \geq 0$, and $N \geq 0$.

Here we will not deal with these iterative/semi-iterative methods. Instead, we will consider a direct solver which incorporates a symmetric factorization.

Direct Solution for the KKT System

- A direct method to solve the KKT system (11.77) is to provide a **symmetric factorization** of the KKT matrix:

$$PKP^T = LDL^T, \tag{11.78}$$

  where P is a permutation matrix (appropriately chosen), L is lower triangular with $\text{diag}(L) = I$, and D is block diagonal.

- Based on (11.78), we rewrite the KKT system (11.77) as

$$P\begin{bmatrix}\mathbf{b}\\ \mathbf{c}\end{bmatrix} = PK\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\lambda}\end{bmatrix} = PKP^T\Big(P\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\lambda}\end{bmatrix}\Big) = LDL^T\Big(P\begin{bmatrix}\mathbf{x}\\ \boldsymbol{\lambda}\end{bmatrix}\Big). \tag{11.79}$$

- Thus it can be solved as follows.

$$\begin{aligned}&\text{solve} \quad L\mathbf{y}_1 = P\begin{bmatrix}\mathbf{b}\\ \mathbf{c}\end{bmatrix}\\ &\text{solve} \quad D\mathbf{y}_2 = \mathbf{y}_1\\ &\text{solve} \quad L^T\mathbf{y}_3 = \mathbf{y}_2\\ &\text{set} \quad \begin{bmatrix}\mathbf{x}^*\\ \boldsymbol{\lambda}^*\end{bmatrix} = P^T\mathbf{y}_3\end{aligned} \tag{11.80}$$

Note: In Python, `scipy.linalg.ldl` is available.

$$\texttt{L0, D, P} = \texttt{ldl(A)}$$

- The output `L0` of Python function `ldl` is permuted as $\texttt{L0} = P^T L$. Thus you can obtain L as

$$L = P\texttt{L0}. \tag{11.81}$$

- **Reference**: [3] J.R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, **Math. Comput.** 31, 1977, pp.163-179.

Example 11.30. Consider the following QP

$$\min_{x_1,x_2} f(x_1,x_2) := 4x_1 - 2x_2 + 4x_1^2 - 4x_1x_2 + 3x_2^2, \quad \text{subj.to} \quad 2x_1 - x_2 = 1. \tag{11.82}$$

(a) Express the problem in the marix-vector form as in (11.70).

(b) Find the KKT system; solve it by simply inverting K.

Solution. (a) Note that the form of the objective is: $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}$.

$$f(x_1,x_2) = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \underbrace{\begin{bmatrix} 4 & -2 \\ -2 & 3 \end{bmatrix}}_{=A/2} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 & x_2 \end{bmatrix} \underbrace{\begin{bmatrix} 4 \\ -2 \end{bmatrix}}_{=-\mathbf{b}},$$

$$\underbrace{\begin{bmatrix} 2 & -1 \end{bmatrix}}_{=C} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 \end{bmatrix}}_{=\mathbf{c}}.$$

(b) $K = \begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} = \begin{bmatrix} 8 & -4 & 2 \\ -4 & 6 & -1 \\ 2 & -1 & 0 \end{bmatrix}, \quad \begin{bmatrix} \mathbf{b} \\ \mathbf{c} \end{bmatrix} = \begin{bmatrix} -4 \\ 2 \\ 1 \end{bmatrix}.$

Note: The second-order terms in (11.82) gives $\frac{1}{2}\mathbf{x}^T A\mathbf{x}$, of which the Hessian becomes A. This implies that you can get A by computing the Hessian of f, that is, $A = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]$.

exm_QP.m

```
A = [ 8 -4; -4  6]; b = [-4; 2];
C = [2 -1]; c=[1];

n= 3; K=zeros(n,n); r=zeros(n,1);
K(1:2,1:2)=A;
K(3,1:2)=C; K(1:2,3)=C';
r(1:2) = b; r(3)=c;

sol = K\r
x = sol(1:2)
f = (1/2)*x'*A*x - x'*b
```

Output

```
sol =
    0.5000 0 -4.0000

x =
    0.5000 0

f =
     3
```

11.5.2. Interior-point methods

Interior-point methods are iterative schemes where the iterates approach the optimal solution *from the interior of the feasible set*. For simplicity, we consider **inequality-constrained quadratic programming** problems.

$$\min_{\mathbf{x}\in\mathbb{R}^n} Q(\mathbf{x}) := \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{x}^T\mathbf{b}, \quad \text{subj.to} \quad D\mathbf{x} \le \mathbf{d}, \tag{11.83}$$

where $A \in \mathbb{R}^{n\times n}$ is SPD, $D \in \mathbb{R}^{p\times n}$, and $\mathbf{b} \in \mathbb{R}^n$ and $\mathbf{d} \in \mathbb{R}^p$.

Note: Its Lagrangian reads

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) \overset{\text{def}}{=\!=} Q(\mathbf{x}) + \sum_{i=1}^{p} \mu_i(\boldsymbol{D}_i^T\mathbf{x} - d_i), \tag{11.84}$$

where $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_p)^T$ and $D = [\boldsymbol{D}_1, \cdots, \boldsymbol{D}_p]^T$, and therefore

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}, \boldsymbol{\mu}) = \nabla_{\mathbf{x}} Q(\mathbf{x}) + D^T\boldsymbol{\mu} = A\mathbf{x} - \mathbf{b} + D^T\boldsymbol{\mu}. \tag{11.85}$$

Thus the **KKT conditions** for (11.83) are stated as

$$\begin{aligned} A\mathbf{x} + D^T\boldsymbol{\mu} - \mathbf{b} &= 0, \\ D\mathbf{x} - \mathbf{d} &\le 0, \\ \mu_i(D\mathbf{x} - \mathbf{d})_i &= 0, \quad i = 1, 2, \cdots, p, \\ \mu_i &\ge 0, \quad i = 1, 2, \cdots, p. \end{aligned} \tag{11.86}$$

$\{\mu_i(D\mathbf{x} - \mathbf{d})_i = 0\}$ is the **complementary slackness**.

By introducing a **slack variable** $\mathbf{z} = \mathbf{d} - D\mathbf{x}$, the above conditions can be equivalently formulated as follows:

$$\begin{aligned} A\mathbf{x} + D^T\boldsymbol{\mu} - \mathbf{b} &= 0, \\ D\mathbf{x} - \mathbf{d} + \mathbf{z} &= 0, \\ \mu_i z_i &= 0, \quad i = 1, 2, \cdots, p, \\ \mu_i,\ z_i &\ge 0, \quad i = 1, 2, \cdots, p. \end{aligned} \tag{11.87}$$

The **interior-point method** begins with replacing $\mu_i z_i = 0$ by $\mu_i z_i = \theta > 0$ for all $i = 1, 2, \cdots, p$, and enforces $\theta \searrow 0$.

Equation (11.87) can be rewritten as a constrained system of nonlinear equations. We define the **nonlinear map**

$$F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) \overset{\text{def}}{=\!=} \begin{bmatrix} A\mathbf{x} + D^T\boldsymbol{\mu} - \mathbf{b} \\ D\mathbf{x} - \mathbf{d} + \mathbf{z} \\ \mathcal{Z}\mathcal{M}\mathbf{e} \end{bmatrix}, \tag{11.88}$$

where

$$\mathcal{Z} = \text{diag}(z_1, \cdots, z_p), \quad \mathcal{M} = \text{diag}(\mu_1, \cdots, \mu_p), \quad \mathbf{e} = (1, \cdots, 1)^T.$$

Definition 11.31. ***(Central path).*** *The set of points* $(\mathbf{x}_\tau, \boldsymbol{\mu}_\tau, \mathbf{z}_\tau)$, $\tau > 0$, *satisfying*

$$F(\mathbf{x}_\tau, \boldsymbol{\mu}_\tau, \mathbf{z}_\tau) = [0,\ 0,\ \tau\mathbf{e}]^T, \quad \mathbf{z},\ \boldsymbol{\mu} \geq 0, \tag{11.89}$$

is called the ***central path****.*

Not that **on the central path:** $\mathbf{z} > \mathbf{0} \ \Rightarrow D\mathbf{x} - \mathbf{d} < \mathbf{0}$**.**

The Idea: Apply **Newton's method** to (11.88) to compute the iterates $(\mathbf{x}_k, \boldsymbol{\mu}_k, \mathbf{z}_k)$ **on the central path**, as **interior points**.

- Given a feasible iterate $(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) = (\mathbf{x}_k, \boldsymbol{\mu}_k, \mathbf{z}_k)$, we introduce a duality measure θ:

$$\theta := \frac{1}{p}\sum_{i=1}^{p} z_i\mu_i = \frac{\mathbf{z}^T\boldsymbol{\mu}}{p}. \tag{11.90}$$

- The Newton increment $(\Delta\mathbf{x}, \Delta\boldsymbol{\mu}, \Delta\mathbf{z})$ solves the linear system

$$\nabla F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\mu} \\ \Delta\mathbf{z} \end{bmatrix} = -F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) + \begin{bmatrix} 0 \\ 0 \\ \sigma\theta\mathbf{e} \end{bmatrix}, \tag{11.91}$$

  where $\sigma \in [0, 1]$ is an algorithm parameter and

$$\nabla F(\mathbf{x}, \boldsymbol{\mu}, \mathbf{z}) \overset{\text{def}}{=\!=} \begin{bmatrix} A & D^T & 0 \\ D & 0 & I \\ 0 & \mathcal{Z} & \mathcal{M} \end{bmatrix}.$$

- Then, the new iterate $(\mathbf{x}_{k+1}, \boldsymbol{\mu}_{k+1}, \mathbf{z}_{k+1})$ is determined by means of

$$(\mathbf{x}_{k+1}, \boldsymbol{\mu}_{k+1}, \mathbf{z}_{k+1}) = (\mathbf{x}_k, \boldsymbol{\mu}_k, \mathbf{z}_k) + \alpha(\Delta\mathbf{x}, \Delta\boldsymbol{\mu}, \Delta\mathbf{z}), \tag{11.92}$$

  with α chosen such that the new iterate stays **feasible**.

Exercises for Chapter 11

11.1. Use the **method of Lagrange multipliers** to solve the problem.

$$\min_{x_1,x_2} x_1^2 + x_2^2, \quad \text{subj.to} \begin{cases} x_1 \geq 1 \\ x_2 \geq 2 \end{cases} \tag{11.93}$$

Hint: You may start with the **Lagrangian**

$$\mathcal{L}(x_1, x_2, \alpha_1, \alpha_2) = x_1^2 + x_2^2 - \alpha_1(x_1 - 1) - \alpha_2(x_2 - 2), \quad \alpha_1, \alpha_2 \geq 0, \tag{11.94}$$

and consider the dual problem $\max_{\boldsymbol{\alpha}\geq 0} \min_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha})$, where $\boldsymbol{\alpha} = (\alpha_1, \alpha_2)$ and $\mathbf{x} = (x_1, x_2)$. Then

$$\nabla_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = \begin{bmatrix} 2x_1 - \alpha_1 \\ 2x_2 - \alpha_2 \end{bmatrix} = 0 \;\Rightarrow\; \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \alpha_1/2 \\ \alpha_2/2 \end{bmatrix}. \tag{11.95}$$

Ans: 5

11.2. Use pencil-and-paper to solve the following QP problem:

$$\min_{x_1,x_2} x_1^2 + x_2^2, \quad \text{subj.to } x_1 + x_2 = 5. \tag{11.96}$$

(a) Express the problem in the matrix-vector form as in (11.70).
(b) Formulate its Lagrangian.
(c) Find the KKT system.
(d) Then, find $\min f$ by solving the KKT system.
(e) Check if your solution satisfies the equality constraints.

Hint: You may follow Example 11.30.

Ans: $x_1^* = x_2^* = 2.5, \lambda^* = -5$.

11.3. C The previous problem (Exercise 11.2) can be interpreted geometrically.

(a) Draw contour plots $x_1^2 + x_2^2 = c$ for various c, including $c = 12.5$.
(b) Superpose the constraint line: $x_1 + x_2 = 5$.
(c) Add the solution point (x_1^*, x_2^*).
(d) Make a conclusion.

To be continued $\Longrightarrow$

11.4. C Solve the following QP problem

$$\min_{x_1,x_2,x_3} f(x_1,x_2,x_3) := 4x_1^2 + 5x_2^2 + 2x_3^2 - 3x_1x_2 + 3x_2x_3 + 2x_1, \quad \text{subj.to} \\ 2x_1 - x_2 - 3x_3 = 1, \quad x_1 - 3x_2 + 3x_3 = -1. \tag{11.97}$$

(a) Express the problem in the matrix-vector form as in (11.70).
(b) Formulate its Lagrangian.
(c) Find the KKT system.
(d) Then, find $\min f$ by solving the KKT system.
(e) Check if your solution satisfies the equality constraints.

Hint: First determine `A, b, C, c` as in Example 11.30. Then form the KKT matrix `K` and the right-side `r`:

```
n= 5; K=zeros(n,n); r=zeros(n,1);
K(1:3,1:3)=A; K(4:5,1:3)=C; K(1:3,4:5)=C';
r(1:3) = b; r(4:5)=c;
```

Ans: $\min f = 4.2300$, when $\mathbf{x} = (0.6243, 0.6879, 0.1464)^T$

11.5. C **(Optional for undergraduate students)** Implement a code to verify Example 11.27, by computing the primal optimum f^* and the dual optimum g^* in particular.

Hint: You may modify `duality_convex.py` on 405. But you may need to employ a polynomial solver.

The 15% Rule: Turning Curiosity into Financial Freedom

The 15% Rule

- Use 85% of your effort to become excellent at what you do.
- Use 15% to discover what else you could become.

Invest 15% of your discretionary/optional time in something outside your field: One day a week–or one or two hours a day:

- Put your phone away.
- Stop scrolling.
- Pick a field you know little about.
- Study it as if you were going to become the CEO of a company in that field.

Not for a grade. Not for your résumé.
Study it because someday it may become an opportunity.

Why Think Like a CEO?

Don't just ask:

How does renewable energy work?

Ask:

Where is the money made?

What problems remain unsolved?

Who are the customers?

What technologies are changing the industry?

Where could mathematics or computing create value?

If I were the CEO, what would I do?

That changes learning from

$$\begin{array}{c} \text{Information} \longrightarrow \text{Knowledge} \\ \downarrow \\ \text{Information} \longrightarrow \text{Knowledge} \longrightarrow \text{Opportunity} \end{array}$$

Three hundred years ago, **land** was wealth.
Later, the **industrialists** came to possess wealth.
Today, wealth lies in **information**.

Your Learning Speed Is Trainable

Let T_n be the time required to understand the n-th field. Then

$$T_1 \approx \textbf{1–2 years}$$

- However, as you keep learning new fields, you start to see **common structures**:

  Technology → Market → Cost → Risk → Competition → Opportunity

- So, conceptually: $T_{k+1} < T_k$
- With practice, you will eventually reach a point where you can grasp the essentials of a new subject **in just one or two months**:

$$T_n \approx \textbf{1–2 months, for a large } n$$

I call it the **learning compound interest**. It's not just money that compounds; the ability to learn compounds as well.

Here are some **topic examples** for the 15% Rule:

- Renewable Energy, AI for Healthcare
- Robotics, Cybersecurity
- Agriculture Technology, Stock Market

Every business or industrial R&D project focuses on **optimizing five key variables: scale, shape, direction, interface, and flow.**

Beyond Equations: Building True

Core Lessons from Robert Kiyosaki's ***Rich Dad Poor Dad***

- Your career earns money. Your financial system buys freedom.
- Don't measure wealth by salary. Measure it by freedom.

High Income $\neq$ Financial Freedom

The Poor Dad Mindset: The Lifestyle Trap

Imagine your salary grows:

$\$60K \rightarrow \$80K \rightarrow \$100K \rightarrow \$130K$

But your spending grows with it:

$\$50K \rightarrow \$70K \rightarrow \ \$90K \rightarrow \$120K.$

Your career improved enormously.
Your freedom did not.

A raise should increase your assets before it increases your lifestyle.

The Rich Dad Idea: Assets First

Robert Kiyosaki's useful simplification:

Assets put money into your pocket.

Liabilities take money out of your pocket.

Financial Thinking for Scientists

- Your Greatest Asset is Time
- Avoid the Three Delays:

  "I'll start when I make more money."
  "I'll invest after I buy everything I need."
  "I'll learn about finance later."

  $$\boxed{\text{Earn} \rightarrow \text{Save/Invest} \rightarrow \text{Spend}}$$

- Build a Machine, Not a Willpower Test

  Good scientists automate repetitive processes.
  Do the same with money.
  When income arrives: automatically
  save $\rightarrow$ invest $\rightarrow$ live on the remainder
  Don't make the same financial decision twelve times a year.
  Design the system once.
  This is essentially an algorithm for personal finance.

The poor and the middle class work for money
The rich have money work for them

- If you work for money, you give the power to your employer.
- If money works for you, you keep the power and control it.

One Equation to Remember

I might finish with a deliberately informal equation:

$$\text{Freedom} \approx \frac{\text{Assets + Skills + Time}}{\text{Lifestyle Demand}}$$

Invest time to get skills. (*The 15% Rule*)
Earn with your skills.
Build assets with your income.
Buy back your time.

CHAPTER 12

Introduction to Machine Learning

In this chapter, you will learn:

- What machine learning (ML) is
- Popular ML classifiers
- Scikit-Learn: A Python ML library
- A machine learning modelcode

The chapter is a brief introduction to **machine learning** (ML), showing **key ideas** of it, along with

- mathematical issues &
- how to solve them.

CNN - Convolutional Neural Networks - Miner and the Jeweler
YouTube (Musical) (Melodic Rap)

Contents of Chapter 12

12.1. What is Machine Learning? 422
12.2. Binary Classifiers 427
12.3. Logistic Regression 438
12.4. Support Vector Machine 442
12.5. k-Nearest Neighbors 457
12.6. Neural Networks 459
12.7. A Machine Learning Modelcode: Scikit-Learn Comparisons and Ensembling 469
Exercises for Chapter 12 474

12.1. What is Machine Learning?

The Three Tasks (T3)

Most real-world problems are expressed as

$$f(\mathbf{x}) = y, \tag{12.1}$$

where f is an **operation**, $\mathbf{x}$ denotes the **input**, and y is the **output**.

1. Known: $(f, \mathbf{x}) \Rightarrow y$: simple to get
2. Known: $(f, y) \Rightarrow \mathbf{x}$: solve the equation (12.1)
3. Known: $(\mathbf{x}, y) \Rightarrow f$: approximated; Machine Learning

(12.2)

Definition 12.1. Machine learning (ML)

- **ML algorithms** are algorithms that can learn from **data** (input) and produce **functions/models** (output).
- **Machine learning** is the science of getting machines to act, without functions/models being explicitly programmed to do so.

Example 12.2. There are different types of ML:

- **Supervised learning**: e.g., classification, regression $\Leftarrow$ Labeled data
- **Unsupervised learning**: e.g., clustering $\Leftarrow$ No labels
- **Reinforcement learning**: e.g., chess engine $\Leftarrow$ Reward system

The most popular type is **supervised learning**.

A Belief in Machine Learning

The average is correct (at least, acceptable)

- **"Guess The Weight of the Ox"** Competition
  - Questioner: **Francis Galton** – a cousin of Charles Darwin
  - A county fair, Plymouth, MA, 1907
  - The annual West of England Fat Stock and Poultry Exhibition
- 800 people gathered: **mean=1197 lbs**; **real=1198 lbs**

Supervised Learning

Assumption. Given a data set $\{(\mathbf{x}_i, y_i)\}$, where y_i are labels, there exists a relation $f : X \to Y$.

Supervised learning:

$$\begin{cases} \text{Given}: & \text{A } \textbf{training data } \{(\mathbf{x}_i, y_i) \mid i = 1, \cdots, N\} \\ \text{Find}: & \widehat{f} : X \to Y, \text{ a good approximation to } f \end{cases} \tag{12.3}$$

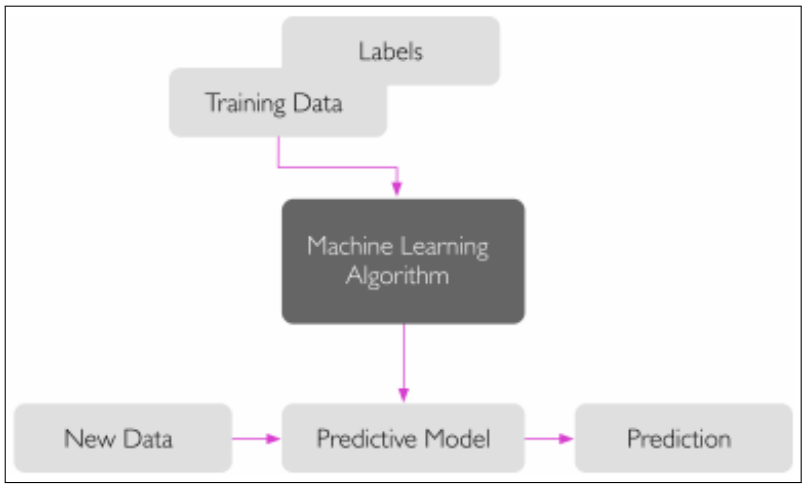


Figure 12.1: Supervised learning and prediction.

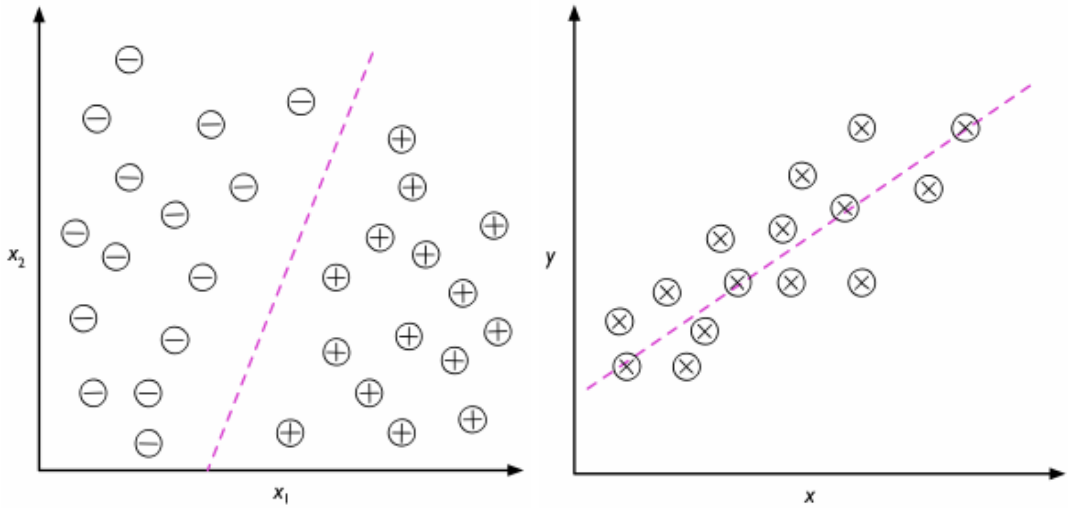


Figure 12.2: Classification and regression.

Unsupervised Learning

Note:

- **In supervised learning**, we know the right answer beforehand when we train our model, and **in reinforcement learning**, we define a measure of reward for particular actions by the agent.
- **In unsupervised learning**, however, we are dealing with unlabeled data or data of unknown structure. Using unsupervised learning techniques, we are able **to explore the structure of our data** to extract meaningful information, without the guidance of a known outcome variable or reward function.
- **Clustering** is an exploratory data analysis technique that allows us to organize a pile of information into **meaningful subgroups** (clusters) without having any prior knowledge of their group memberships.

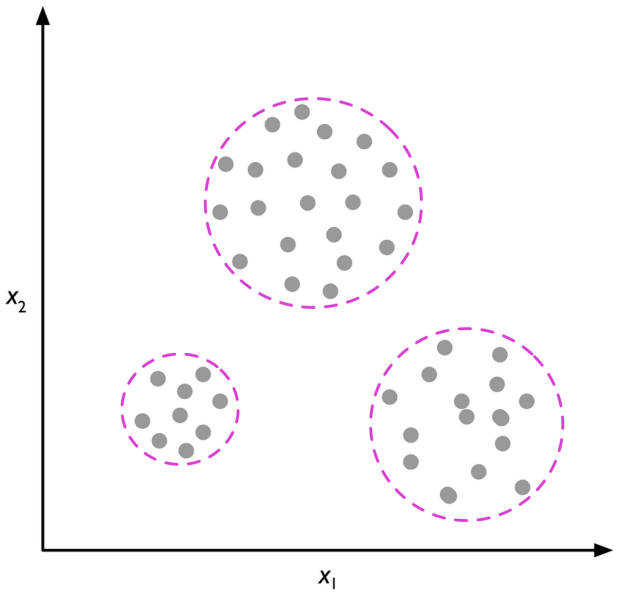
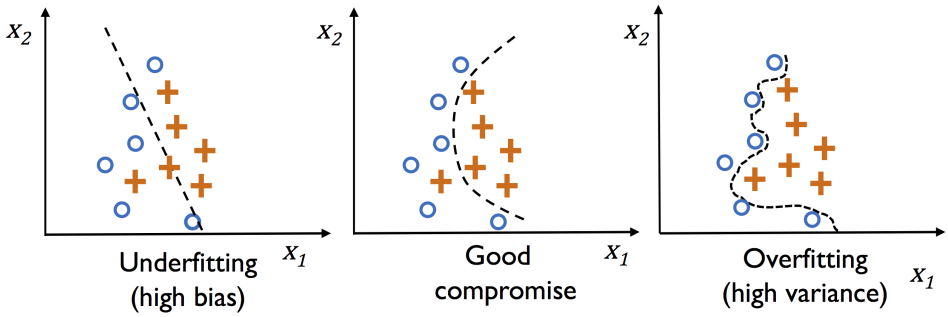


Figure 12.3: Clustering.

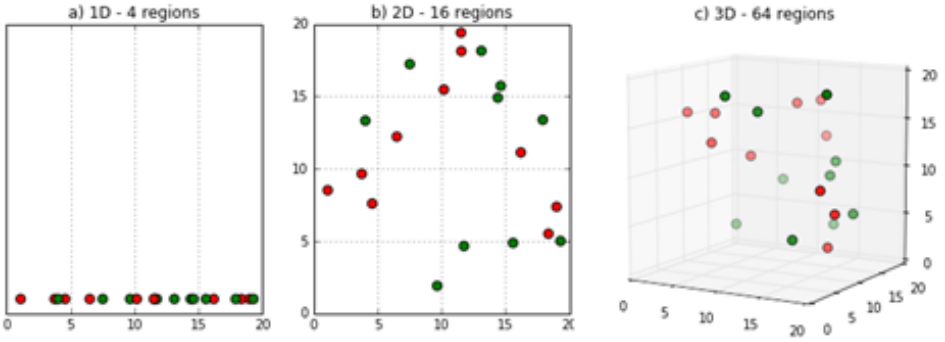
Why is ML not Always Simple?

Major Issues in ML

1. **Overfitting**: Fitting **training data** *too tightly*
   - **Difficulties**: Accuracy drops significantly for **test data**
   - **Remedies**:
     - More training data (often, impossible)
     - Early stopping; feature selection
     - Regularization; ensembling (multiple classifiers)



2. **Curse of Dimensionality**: **The feature space becomes increasingly sparse** for **an increasing number of dimensions** (of a fixed-size training dataset)
   - **Difficulties**: Larger error, more computation time;
     Data points **appear equidistant** from all the others
   - **Remedies**
     - More training data (often, impossible)
     - **Dimensionality reduction** (e.g., Feature selection, PCA)



3. **Multiple Local Minima Problem**
   Training often involves minimizing an objective function.
   - **Difficulties**: Larger error, unrepeatable
   - **Remedies**
     - Gaussian sailing; regularization
     - **Careful access to the data** (e.g., mini-batch)

4. **Interpretability**:
   Although ML has come very far, researchers still **don't know exactly how some algorithms (neural nets) work**.
   - If we don't know how training nets actually work, how do we make any real progress?

5. **One-Shot Learning**:
   We still haven't been able to achieve one-shot learning. ***Traditional gradient-based networks*** **need a huge amount of data**, and are often in the form of **extensive iterative training**.
   - Instead, we should find a way to enable neural networks to learn, **using just a few examples**.

12.2. Binary Classifiers

A **binary classifier** is a function which can decide whether or not an input vector belongs to some specific class (e.g., spam/ham).

- Binary classification often refers to those classification tasks that have **two class labels.** (two-class classification)
- It is **a type of linear classifier**, i.e. a classification algorithm that makes **its predictions based on a linear predictor function**.
- Linear classifiers are **artificial neurons**.

Examples: Perceptron [30], Adaline, Logistic Regression, Support Vector Machine [5]

Remark 12.3. Neurons are interconnected nerve cells, involved in the processing and transmitting of chemical and electrical signals. Such a nerve cell can be described as a simple logic gate with binary outputs;

- multiple signals arrive at the dendrites,
- they are integrated into the cell body,
- and if the accumulated signal exceeds a certain threshold, an output signal is generated that will be passed on by the axon.

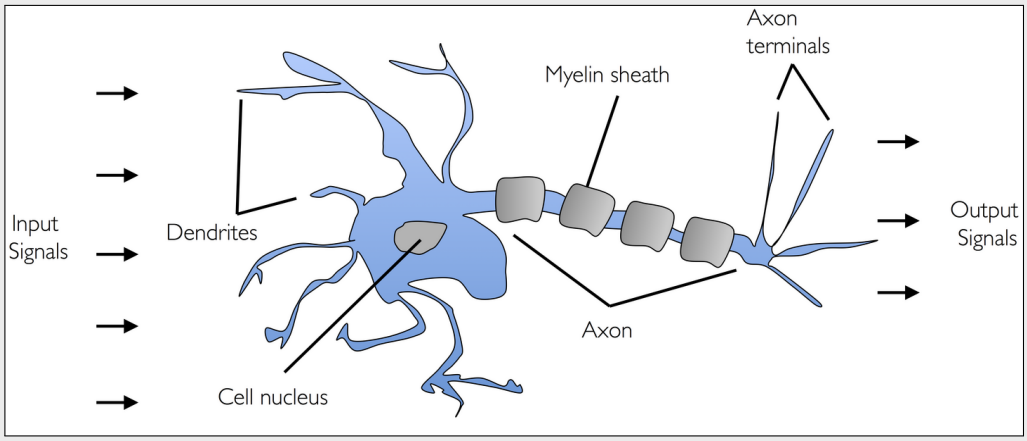


Figure 12.4: A schematic description of a neuron.

Definition 12.4. Let $\{(\mathbf{x}^{(i)}, y^{(i)})\}$ be labeled data, with $\mathbf{x}^{(i)} \in \mathbb{R}^d$ and $y^{(i)} \in \{0, 1\}$. A **binary classifier** finds a **hyperplane** in $\mathbb{R}^d$ that separates data points $X = \{\mathbf{x}^{(i)}\}$ to two classes:

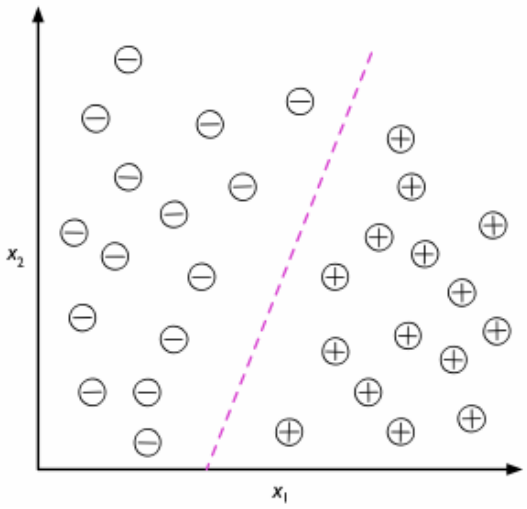


Figure 12.5: A binary classifier, finding a **hyperplane** in $\mathbb{R}^d$.

Observation 12.5. Binary Classifiers

- The **labels** $\{0, 1\}$ are chosen for simplicity.
- A **hyperplane** can be formulated by a normal vector $\mathbf{w} \in \mathbb{R}^d$ and a shift (bias) w_0:

$$z = \mathbf{w}^T \mathbf{x} + w_0. \tag{12.4}$$

- To **learn $\mathbf{w}$ and w_0**, you may formulate a **cost function** to minimize, as the **Sum of Squared Errors** (SSE):

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2} \sum_i \left(y^{(i)} - \phi(\mathbf{w}^T \mathbf{x}^{(i)} + w_0) \right)^2, \tag{12.5}$$

where $\phi(z)$ is an **activation function**.

An important task in ML is on the choice of effective

- **activation functions** and
- **cost functions**.

12.2.1. The Perceptron Algorithm

The **perceptron** is a binary classifier of supervised learning.

- 1957: Perceptron algorithm is invented by **Frank Rosenblatt**, Cornell Aeronautical Laboratory
  - Built on work of Hebbs (1949)
  - Improved by Widrow-Hoff (1960): Adaline
- 1960: Perceptron Mark 1 Computer – hardware implementation
- 1970's: Learning methods for two-layer neural networks

Definition 12.6. We can pose the **perceptron** as a **binary classifier**, in which we refer to our two classes as 1 (positive class) and -1 (negative class) for simplicity.

- **Input values**: $\mathbf{x} = (x_1, x_2, \cdots, x_d)^T$
- **Weight vector**: $\mathbf{w} = (w_1, w_2, \cdots, w_d)^T$
- **Net input**: $z = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- **Activation function**: $\phi(z)$, defined by

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq \theta, \\ -1 & \text{otherwise}, \end{cases} \tag{12.6}$$

where θ is a threshold.

For simplicity, we can bring the threshold θ in (12.6) to the left side of the equation; define a weight-zero as $w_0 = -\theta$ and reformulate as

$$\phi(z) = \begin{cases} 1 & \text{if } z \geq 0, \\ -1 & \text{otherwise}, \end{cases} \qquad z = \mathbf{w}^T \mathbf{x} = w_0 + w_1 x_1 + \cdots + w_d x_d. \tag{12.7}$$

In the ML literature, the variable w_0 is called the **bias**.
The equation $w_0 + w_1 x_1 + \cdots + w_d x_d = 0$ represents a hyperplane in $\mathbb{R}^d$, while w_0 decides the intercept.

The Perceptron Learning Rule

The whole idea behind the **Rosenblatt's thresholded perceptron model** is to use a reductionist approach to mimic how a single neuron in the brain works: it either fires or it doesn't.

Algorithm 12.7. Rosenblatt's Initial Perceptron Rule

1. Initialize the weights to 0 or small random numbers.
2. For each training sample $\mathbf{x}^{(i)}$,
   (a) Compute the output value $\widehat{y}^{(i)}$ $(:= \phi(\mathbf{w}^T\mathbf{x}^{(i)}))$.
   (b) Update the weights.

The update of the weight vector $\mathbf{w}$ can be more formally written as:

$$\begin{aligned} \mathbf{w} &= \mathbf{w} + \Delta\mathbf{w}, \quad \Delta\mathbf{w} = \eta\,(y^{(i)} - \widehat{y}^{(i)})\,\mathbf{x}^{(i)}, \\ w_0 &= w_0 + \Delta w_0, \quad \Delta w_0 = \eta\,(y^{(i)} - \widehat{y}^{(i)}), \end{aligned} \tag{12.8}$$

where η is the **learning rate**, $0 < \eta < 1$, $y^{(i)}$ is the true class label of the i-th training sample, and $\widehat{y}^{(i)}$ denotes the predicted class label.

Remark 12.8. A simple thought experiment for the perceptron learning rule:

- Let the perceptron predict the class label correctly. Then $y^{(i)} - \widehat{y}^{(i)} = 0$ so that the weights remain unchanged.
- Let the perceptron make a wrong prediction. Then

$$\Delta w_j = \eta\,(y^{(i)} - \widehat{y}^{(i)})\,x_j^{(i)} = \pm 2\,\eta\,x_j^{(i)}$$

  so that the weight w_j is pushed towards the direction of the positive or negative target class, respectively.

Perceptron for Iris Dataset

perceptron.py

```
import numpy as np

class Perceptron():
    def __init__(self, xdim, epoch=10, learning_rate=0.01):
        self.epoch = epoch
        self.learning_rate = learning_rate
        self.weights = np.zeros(xdim + 1)

    def activate(self, x):
        net_input = np.dot(x,self.weights[1:])+self.weights[0]
        return 1 if (net_input > 0) else 0

    def fit(self, Xtrain, ytrain):
        for k in range(self.epoch):
            for x, y in zip(Xtrain, ytrain):
                yhat = self.activate(x)
                self.weights[1:] += self.learning_rate*(y-yhat)*x
                self.weights[0]  += self.learning_rate*(y-yhat)

    def predict(self, Xtest):
        yhat=[]
        #for x in Xtest: yhat.append(self.activate(x))
        [yhat.append(self.activate(x)) for x in Xtest]
        return yhat

    def score(self, Xtest, ytest):
        count=0;
        for x, y in zip(Xtest, ytest):
            if self.activate(x)==y: count+=1
        return count/len(ytest)

    #-----------------------------------------------------
    def fit_and_fig(self, Xtrain, ytrain):
        wgts_all = []
        for k in range(self.epoch):
            for x, y in zip(Xtrain, ytrain):
                yhat = self.activate(x)
                self.weights[1:] += self.learning_rate*(y-yhat)*x
                self.weights[0]  += self.learning_rate*(y-yhat)
                if k==0: wgts_all.append(list(self.weights))
        return np.array(wgts_all)
```

Iris_perceptron.py

```
import numpy as np; import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets; #print(dir(datasets))
np.set_printoptions(suppress=True)
from perceptron import Perceptron

#-------------------------------------------------------------
data_read = datasets.load_iris();  #print(data_read.keys())
X = data_read.data;
y = data_read.target
targets = data_read.target_names; features = data_read.feature_names

N,d = X.shape; nclass=len(set(y));
print('N,d,nclass=',N,d,nclass)

#---- Take 2 classes in 2D ---------------------------------
X2 = X[y<=1]; y2 = y[y<=1];
X2 = X2[:,[0,2]]

#---- Train and Test ---------------------------------------
Xtrain, Xtest, ytrain, ytest = train_test_split(X2, y2,
        random_state=None, train_size=0.7e0)
clf = Perceptron(X2.shape[1],epoch=2)
#clf.fit(Xtrain, ytrain);
wgts_all = clf.fit_and_fig(Xtrain, ytrain);
accuracy = clf.score(Xtest, ytest); print('accuracy =', accuracy)
#yhat = clf.predict(Xtest);
```

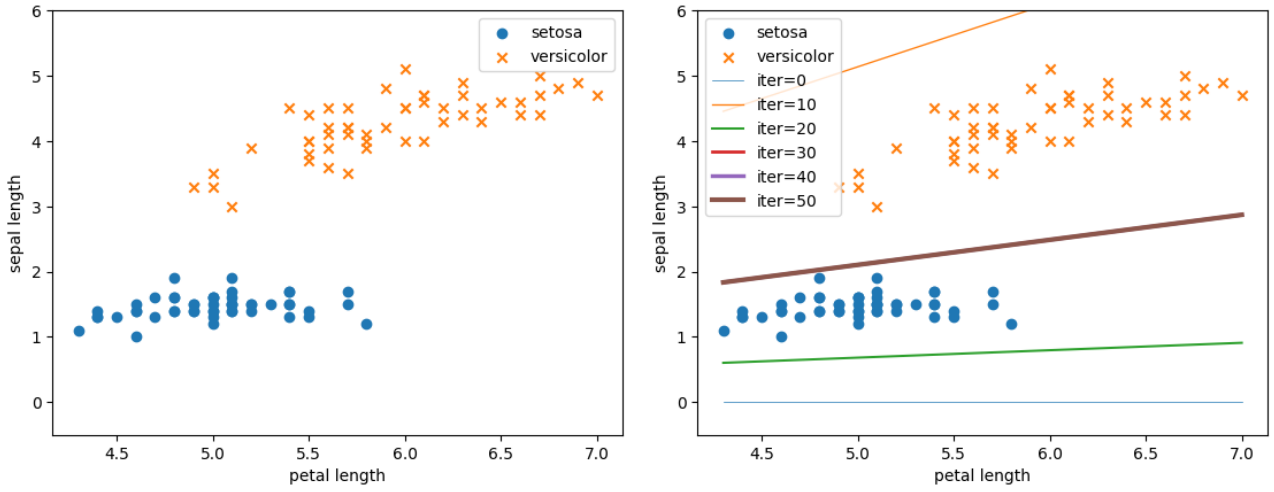


Figure 12.6: A part of Iris data (left) and the convergence of Perceptron iteration (right).

12.2.2. Adaline: ADAptive LInear NEuron

- (Widrow & Hoff, 1960)
- Weights are updated based on linear activation: e.g.,

$$\phi(\mathbf{w}^T\mathbf{x}) = \mathbf{w}^T\mathbf{x}.$$

  That is, ϕ is the **identity function**.
- Adaline algorithm is particularly interesting because it illustrates the key concept of defining and minimizing **continuous cost functions**, which will **lay the groundwork for understanding more advanced machine learning algorithms** for classification, such as logistic regression and support vector machines, as well as regression models.
- Continuous cost functions allow the ML optimization to incorporate **advanced mathematical techniques** such as **calculus**.

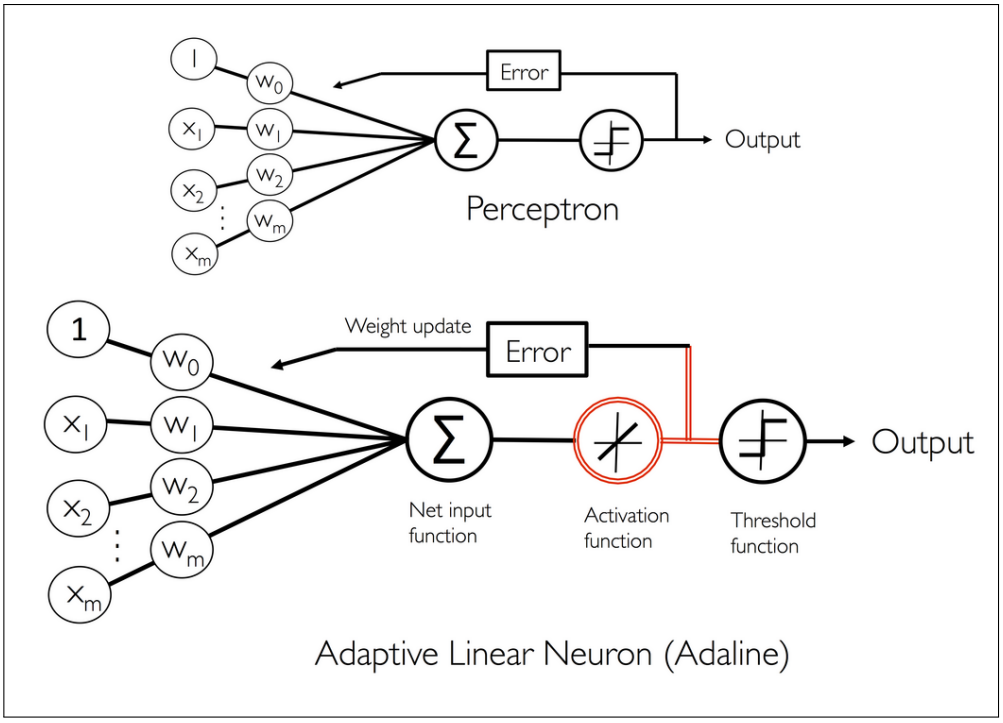


Figure 12.7: Perceptron vs. Adaline

Algorithm 12.9. Adaline Learning:
Given a dataset $\{(\mathbf{x}^{(i)}, y^{(i)}) \mid i = 1, 2, \cdots, N\}$, learn the weights $\mathbf{w}$ and bias $b = w_0$:

- **Activation function**: $\phi(z) = z$ (i.e., identity activation)
- **Cost function**: the SSE

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2} \sum_{i=1}^{N} \left(y^{(i)} - \phi(z^{(i)}) \right)^2, \tag{12.9}$$

where $\boldsymbol{z^{(i)}} = \mathbf{w}^T \mathbf{x}^{(i)} + \boldsymbol{b}$ and $\phi = I$, the identity.

The dominant algorithm for the minimization of the cost function is the the Gradient Descent Method.

Algorithm 12.10. The Gradient Descent Method uses $-\nabla \mathcal{J}$ for the **search direction** (update direction):

$$\begin{array}{rclcl} \mathbf{w} & = & \mathbf{w} + \Delta \mathbf{w} & = & \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}, b), \\ b & = & b + \Delta b & = & b - \eta \nabla_b \mathcal{J}(\mathbf{w}, b), \end{array} \tag{12.10}$$

where $\eta > 0$ is the **step length** (learning rate).

Computation of $\nabla \mathcal{J}$ for Adaline:
The partial derivatives of the cost function $\mathcal{J}$ w.r.to w_j and b read

$$\begin{array}{rcl} \dfrac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial w_j} & = & -\displaystyle\sum_i \left(y^{(i)} - \phi(z^{(i)}) \right) x_j^{(i)}, \\ \dfrac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial b} & = & -\displaystyle\sum_i \left(y^{(i)} - \phi(z^{(i)}) \right). \end{array} \tag{12.11}$$

Thus, with $\phi = I$,

$$\begin{array}{rclcl} \Delta \mathbf{w} & = & -\eta \nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}, b) & = & \eta \displaystyle\sum_i \left(y^{(i)} - \phi(z^{(i)}) \right) \mathbf{x}^{(i)}, \\ \Delta b & = & -\eta \nabla_b \mathcal{J}(\mathbf{w}, b) & = & \eta \displaystyle\sum_i \left(y^{(i)} - \phi(z^{(i)}) \right). \end{array} \tag{12.12}$$

You will modify `perceptron.py` **for Adaline**; an implementation issue is considered in Exercise 12.3, p.474.

Implementation of Adaline Learning

- The Adaline classifier can be implemented quite simply using matrix and vector operations. For example, let

$$\mathbf{a}^{(i)} = [1, \mathbf{x}^{(i)}] \text{ and } \widetilde{\mathbf{w}} = [b, \mathbf{w}]^T. \tag{12.13}$$

  Then

$$z^{(i)} := b + \mathbf{w}^T\mathbf{x}^{(i)} = \mathbf{a}^{(i)}\widetilde{\mathbf{w}}. \tag{12.14}$$

- **Let A be the vertical stack of $\mathbf{a}^{(i)}$**, which reads

$$A = \texttt{np.hstack}(\,(\texttt{np.ones}((N,1)), X)\,).$$

  Then, the Adaline objective can be rewritten as

$$\mathcal{J}(\widetilde{\mathbf{w}}) = \frac{1}{2}||\mathbf{y} - A\widetilde{\mathbf{w}}||^2, \tag{12.15}$$

  which is a least-squares problem.
- Its gradient becomes

$$\nabla_{\widetilde{\mathbf{w}}}\mathcal{J}(\widetilde{\mathbf{w}}) = -A^T(\mathbf{y} - A\widetilde{\mathbf{w}}). \tag{12.16}$$

Example 12.11. Use the dataset and labels to train the Adaline classifier.

$$X = \begin{bmatrix} 1 & 1 \\ 0 & -2 \\ -2 & 0 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

Carry out **two iterations of the Gradient descent method** for the Adaline model, with the learning rate $\eta = 0.1$ and the zero initial weights.

Solution. Given data, $A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & -2 \\ 1 & -2 & 0 \end{bmatrix}$, and given initialization $\widetilde{\mathbf{w}}^0 = 0$, we obtain

$$\nabla_{\widetilde{\mathbf{w}}}\mathcal{J}(\widetilde{\mathbf{w}}^0) = -A^T(\mathbf{y} - 0) = \begin{bmatrix} 1 \\ -3 \\ -3 \end{bmatrix} \Rightarrow \widetilde{\mathbf{w}}^1 = \widetilde{\mathbf{w}}^0 - 0.1\begin{bmatrix} 1 \\ -3 \\ -3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ 0.3 \\ 0.3 \end{bmatrix}.$$

Now, for the second iteration:

$$\nabla_{\widetilde{\mathbf{w}}}\mathcal{J}(\widetilde{\mathbf{w}}^1) = -A^T(\mathbf{y} - A\widetilde{\mathbf{w}}^1) = -A^T\begin{bmatrix} 0.5 \\ -0.3 \\ -0.3 \end{bmatrix} = \begin{bmatrix} 0.1 \\ -1.1 \\ -1.1 \end{bmatrix}$$

$$\Rightarrow \quad \widetilde{\mathbf{w}}^2 = \widetilde{\mathbf{w}}^1 - 0.1\begin{bmatrix} 0.1 \\ -1.1 \\ -1.1 \end{bmatrix} = \begin{bmatrix} -0.11 \\ 0.41 \\ 0.41 \end{bmatrix}$$

which represents the line $\mathbf{-0.11 + 0.41x_1 + 0.41x_2 = 0}$. □

Hyperparameters

Definition 12.12. In ML, a **hyperparameter** is a parameter whose value is set before the learning process begins. Thus it is an **algorithmic parameter**. Examples are

- The learning rate (η)
- The number of maximum epochs/iterations (`n_iter`)

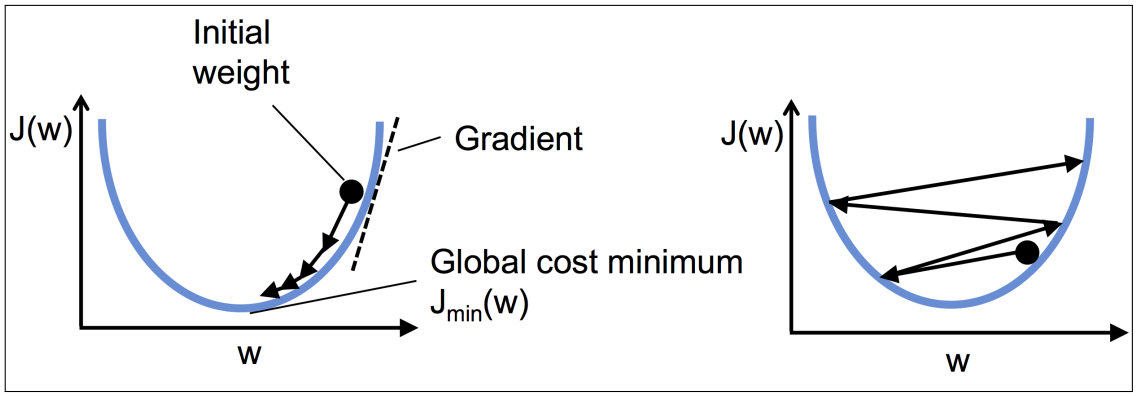


Figure 12.8: Well-chosen learning rate vs. a large learning rate

Note: There are effective searching schemes to set the learning rate η automatically.

Multi-class Classification

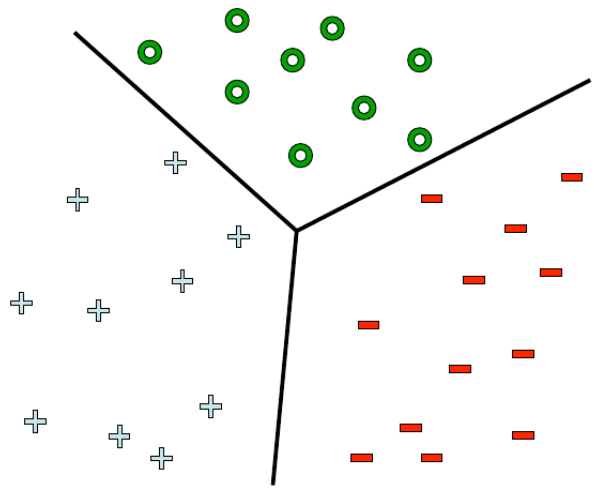


Figure 12.9: Classification for three classes.

One-versus-all (one-versus-rest) classification

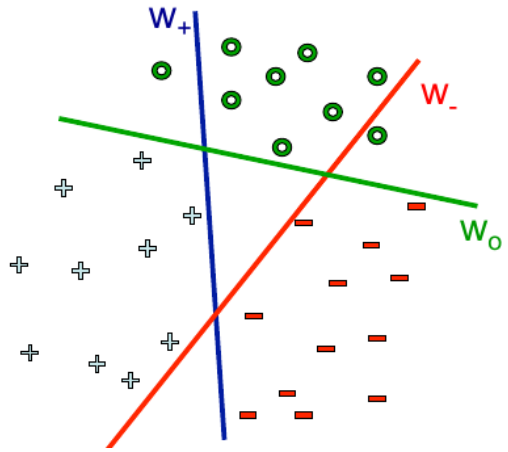


Figure 12.10: Three weights: $\mathbf{w}_-$, $\mathbf{w}_+$, and $\mathbf{w}_\circ$.

Learning: learn 3 classifiers

- $-$ vs $\{\circ, +\} \Rightarrow$ weights $\mathbf{w}_-$
- $+$ vs $\{\circ, -\} \Rightarrow$ weights $\mathbf{w}_+$
- $\circ$ vs $\{+, -\} \Rightarrow$ weights $\mathbf{w}_\circ$

Prediction: for a new data sample $\mathbf{x}$,

$$\widehat{y} = \arg \max_{i \in \{-,+,\circ\}} \phi(\mathbf{w}_i^T \mathbf{x}).$$

OvA (OvR) is readily applicable for classification of general n classes, $n \geq 2$.

12.3. Logistic Regression

The **logistic regression** classifier employs a specific activation function, called the **standard logistic sigmoid function** or the **sigmoid function**.

Remark 12.13. The Standard Logistic Sigmoid Function

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{12.17}$$

- The standard logistic function is the solution of the simple first-order non-linear ordinary differential equation

$$\frac{d}{dx}y = y(1-y), \quad y(0) = \frac{1}{2}. \tag{12.18}$$

- It can be verified easily as

$$\sigma'(x) = \frac{e^x(1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} = \sigma(x)(1-\sigma(x)). \tag{12.19}$$

- σ' is even: $\sigma'(-x) = \sigma'(x)$.
- **Rotational symmetry** about $(0, 1/2)$:

$$\sigma(x) + \sigma(-x) = \frac{1}{1+e^{-x}} + \frac{1}{1+e^x} = \frac{2+e^x+e^{-x}}{2+e^x+e^{-x}} \equiv 1. \tag{12.20}$$

- $\int \sigma(x)\,dx = \int \frac{e^x}{1+e^x}\,dx = \ln(1+e^x)$, which is known as the **softplus function** in artificial neural networks. It is a smooth approximation of the the **rectifier** (an activation function) defined as

$$f(x) = x^+ = \max(x, 0). \tag{12.21}$$

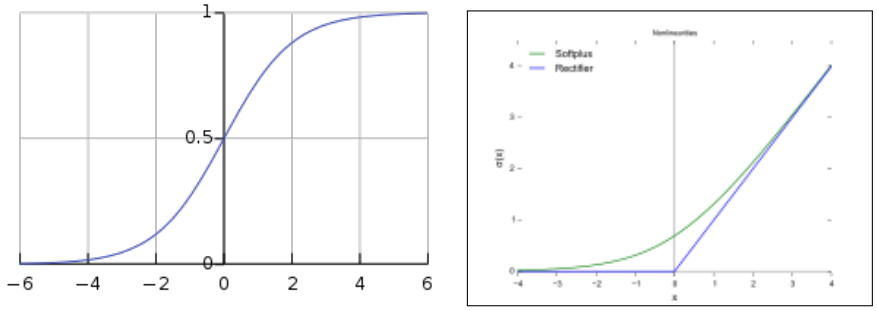


Figure 12.11: **Popular activation functions**: (left) The standard logistic sigmoid function and (right) the rectifier and softplus function.

Logistic regression is a probabilistic model.

- **Logistic regression maximizes the likelihood of the parameter** w; in realization, it is similar to **Adaline**.
- Only the difference is the **activation function** (the sigmoid function), as illustrated in the figure:

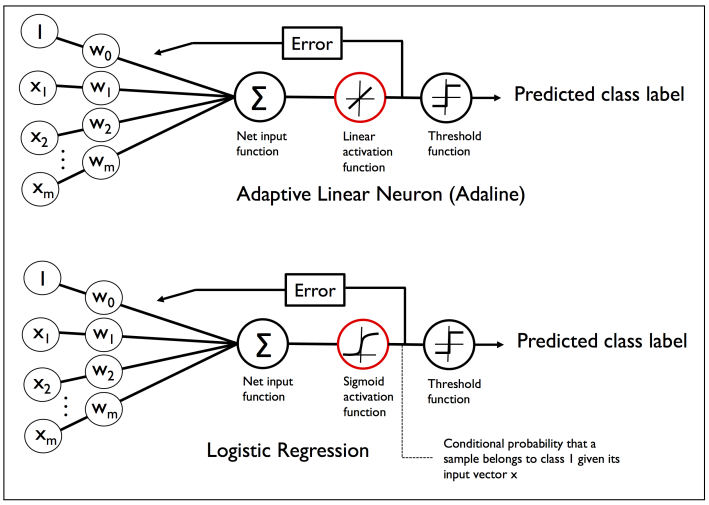


Figure 12.12: Adaline vs. Logistic regression.

- **The prediction** (the output of the sigmoid function) is interpreted as the **probability** of a particular sample belonging to class 1,

$$\phi(z) = p(y = 1|\mathbf{x}; \mathbf{w}), \tag{12.22}$$

given its features x parameterized by the weights w, $z = \mathbf{w}^T\mathbf{x} + b$.

Remark 12.14. Activation Functions

- Perceptron: Thresholding
- Adaline: Identity
- Logistic Regression: The Sigmoid Function

The nonlinear **activation function** is incorporated in order

(a) **to keep the net input restricted to a certain limit** as per our requirement and, more importantly,

(b) **to add nonlinearity** to the classifier.

Derivation of the Logistic Cost Function

- Assume that the individual samples in our dataset are **independent** of one another. Then we can define the **likelihood** L as

$$\begin{aligned} L(\mathbf{w}) &= P(\mathbf{y}|\mathbf{x};\mathbf{w}) = \prod_{i=1}^{N} P(y^{(i)}|\mathbf{x}^{(i)};\mathbf{w}) \\ &= \prod_{i=1}^{N} \left(\phi(z^{(i)})\right)^{y^{(i)}} \left(1-\phi(z^{(i)})\right)^{1-y^{(i)}}, \end{aligned} \tag{12.23}$$

  where $z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)} + b$.

- In practice, it is easier to maximize the (natural) log of this equation, which is called the **log-likelihood function**:

$$\ell(\mathbf{w}) = \ln(L(\mathbf{w})) = \sum_{i=1}^{N} \left[y^{(i)} \ln\left(\phi(z^{(i)})\right) + (1-y^{(i)}) \ln\left(1-\phi(z^{(i)})\right)\right]. \tag{12.24}$$

Algorithm 12.15. Logistic Regression Learning:
From data $\{(\mathbf{x}^{(i)}, y^{(i)})\}$, learn the weights $\mathbf{w}$ and bias b, with

- **Activation function**: $\phi(z) = \sigma(z)$, the logistic sigmoid function
- **Cost function**: The likelihood is maximized.
  Based on the log-likelihood, we define the **logistic cost function** to be minimized:

$$\mathcal{J}(\mathbf{w}, b) = \sum_{i} \left[-y^{(i)} \ln\left(\phi(z^{(i)})\right) - (1-y^{(i)}) \ln\left(1-\phi(z^{(i)})\right)\right], \tag{12.25}$$

  where $z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)} + b$.

Computation of $\nabla\mathcal{J}$ for Logistic Regression:

Let's start by calculating the partial derivative of the logistic cost function with respect to the j–th weight, w_j:

$$\frac{\partial\mathcal{J}(\mathbf{w},b)}{\partial w_j} = \sum_i \left[-y^{(i)}\frac{1}{\phi(z^{(i)})} + (1-y^{(i)})\frac{1}{1-\phi(z^{(i)})}\right]\frac{\partial\phi(z^{(i)})}{\partial w_j}, \tag{12.26}$$

where, using $z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)}$ and (12.19),

$$\frac{\partial\phi(z^{(i)})}{\partial w_j} = \phi'(z^{(i)})\frac{\partial z^{(i)}}{\partial w_j} = \phi(z^{(i)})\left(1-\phi(z^{(i)})\right)x_j^{(i)}.$$

Thus, if follows from the above and (12.26) that

$$\begin{aligned}\frac{\partial\mathcal{J}(\mathbf{w},b)}{\partial w_j} &= \sum_i \left[-y^{(i)}\left(1-\phi(z^{(i)})\right) + (1-y^{(i)})\phi(z^{(i)})\right]x_j^{(i)} \\ &= -\sum_i \left[y^{(i)} - \phi(z^{(i)})\right]x_j^{(i)}\end{aligned}$$

and therefore

$$\nabla_{\mathbf{w}}\mathcal{J}(\mathbf{w}) = -\sum_i \left[y^{(i)} - \phi(z^{(i)})\right]\mathbf{x}^{(i)}. \tag{12.27}$$

Similarly, one can get

$$\nabla_b\mathcal{J}(\mathbf{w}) = -\sum_i \left[y^{(i)} - \phi(z^{(i)})\right]. \tag{12.28}$$

Algorithm 12.16. *Gradient descent learning for Logistic Regression is formulated as*

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \quad b := b + \Delta b, \tag{12.29}$$

*where $\eta > 0$ is the **step length** (learning rate) and*

$$\begin{aligned}\Delta\mathbf{w} &= -\eta\nabla_{\mathbf{w}}\mathcal{J}(\mathbf{w},b) = \eta\sum_i \left[y^{(i)} - \phi(z^{(i)})\right]\mathbf{x}^{(i)}, \\ \Delta b &= -\eta\nabla_b\mathcal{J}(\mathbf{w},b) = \eta\sum_i \left[y^{(i)} - \phi(z^{(i)})\right].\end{aligned} \tag{12.30}$$

Note: ***The above gradient descent rule for Logistic Regression is of the same form as that of Adaline;*** see (12.12) on p. 434. Only the difference is the activation function ϕ.

12.4. Support Vector Machine

Observation 12.17. ML algorithms such as Perceptron, Adaline, and Logistic Regression would **stop or slow down the convergence**, when the linear separator becomes one of hyperplanes (lines) shown below.

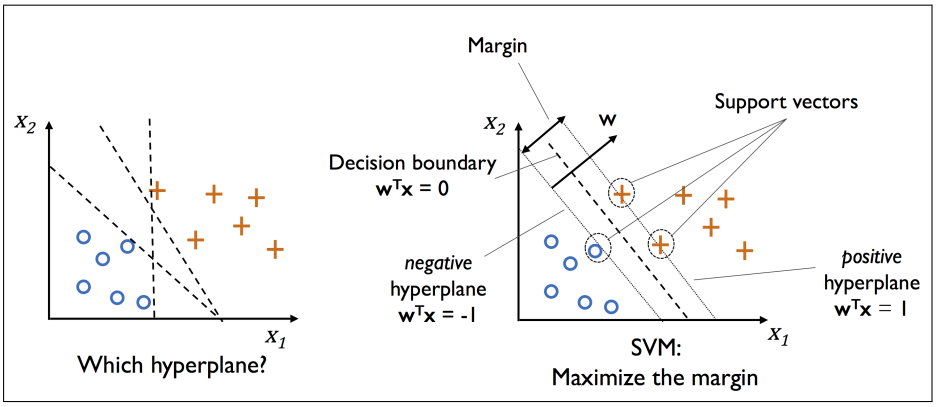


Figure 12.13: (left) Converged linear separators and (right) the one that maximizes the margin.

12.4.1. Linear SVM

- **Support Vector Machine** (**SVM**), developed in 1995 by Cortes-Vapnik [5], can be considered as an extension of the Perceptron/Adaline, **which maximizes the margin**.
- The **rationale** behind having decision boundaries with large margins is that they tend to have a **lower generalization error**, whereas **models with small margins are more prone to overfitting**.

Computation of the Optimal Hyperplane:

- To find an optimal hyperplane that maximizes the margin, let's begin with considering the **positive** and **negative** hyperplanes that are parallel to the decision boundary:

$$\begin{aligned} w_0 + \mathbf{w}^T\mathbf{x}_+ &= 1, \\ w_0 + \mathbf{w}^T\mathbf{x}_- &= -1. \end{aligned} \tag{12.31}$$

  where $\mathbf{w} = [w_1, w_2, \cdots, w_d]^T$.

- If we subtract those two linear equations from each other, then we have

$$\mathbf{w} \cdot (\mathbf{x}_+ - \mathbf{x}_-) = 2$$

and therefore

$$\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot (\mathbf{x}_+ - \mathbf{x}_-) = \frac{2}{\|\mathbf{w}\|}. \tag{12.32}$$

- Note $\mathrm{w} = [w_1, w_2, \cdots, w_d]^T$ is a **normal vector** to the decision boundaries (hyperplanes); the left side of (12.32) is the distance between the positive and negative hyperplanes.
- Maximizing the distance (margin) is equivalent to minimizing its reciprocal $\frac{1}{2}\|\mathrm{w}\|$, or minimizing $\frac{1}{2}\|\mathrm{w}\|^2$.

Problem 12.18. *The **linear SVM** is formulated as*

$$\min_{\mathbf{w}, w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \textit{subject to}$$
$$\begin{cases} w_0 + \mathbf{w}^T\mathbf{x}^{(i)} \geq 1 & \textit{if } y^{(i)} = 1, \\ w_0 + \mathbf{w}^T\mathbf{x}^{(i)} \leq -1 & \textit{if } y^{(i)} = -1. \end{cases} \tag{12.33}$$

Remark 12.19. The constraints in Problem 12.18 can be written as

$$y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1 \geq 0, \qquad \forall\, i. \tag{12.34}$$

- The beauty of the linear SVM is that if the data is linearly separable, there is a unique global minimum value.
- An ideal SVM analysis should produce a hyperplane that completely separates the vectors (cases) into two non-overlapping classes.
- However, perfect separation may not be possible, in practice.

Note: Constrained optimization problems such as (12.33) are typically solved using the **method of Lagrange multipliers**; see Chapter 11.

Solution of the Linear SVM

Recall: The **linear SVM** formulated in Problem 12.18:

$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subj.to} \quad y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1 \geq 0, \quad \forall\, i. \qquad \text{(Primal)} \tag{12.35}$$

To solve the problem, let's begin with its **Lagrangian**:

$$\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1], \tag{12.36}$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_N]^T$, the dual variable (Lagrange multipliers). As in Section 11.2, we formulate the dual problem.

The primal problem (12.35) is equivalent to the **minimax problem**

$$\min_{\mathbf{w},w_0} \max_{\boldsymbol{\alpha}\geq 0} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}), \qquad \text{(Primal)} \tag{12.37}$$

and its **Lagrange dual problem** is defined as the **maximin problem**:

$$\max_{\boldsymbol{\alpha}\geq 0} \min_{\mathbf{w},w_0} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}). \qquad \text{(Dual)} \tag{12.38}$$

Note: **The Linear SVM**

- The linear SVM is a quadratic programming problem; its dual problem is equivalent to the primal problem.
- Assume optimal values $([\mathbf{w}^*, w_0^*], \boldsymbol{\alpha}^*)$ of (12.38) are found. Then

$$\nabla_{\mathbf{w}} \mathcal{L}([\mathbf{w}^*, w_0^*], \boldsymbol{\alpha}^*) = \mathbf{0}, \quad \frac{\partial}{\partial w_0} \mathcal{L}([\mathbf{w}^*, w_0^*], \boldsymbol{\alpha}^*) = 0, \tag{12.39}$$

  which are *necessary* conditions.

Definition 12.20. **Karush-Kuhn-Tucker (KKT) Conditions**
In optimization, KKT conditions [22, 25] are **first derivative tests** (a.k.a. **first-order necessary conditions**) for a solution in nonlinear programming to be optimized, provided that some regularity conditions are satisfied.

Writing the **KKT conditions**, starting with Lagrangian stationarity, where we need to find the first-order derivatives w.r.t. $\mathbf{w}$ and w_0:

$$
\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= \mathbf{w} - \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)} = \mathbf{0} \;\Rightarrow\; \mathbf{w} = \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)}, \\
\frac{\partial}{\partial w_0}\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= -\sum_{i=1}^{N} \alpha_i y^{(i)} = 0 \;\Rightarrow\; \sum_{i=1}^{N} \alpha_i y^{(i)} = 0, \\
&\alpha_i \geq 0, && \text{(dual feasibility)} \\
&\alpha_i \left[y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \right] = 0, && \text{(complementary slackness)} \\
&y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \geq 0. && \text{(primal feasibility)}
\end{aligned}
\tag{12.40}
$$

Using the KKT conditions (12.40), we can simplify the Lagrangian:

$$
\begin{aligned}
\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i y^{(i)} w_0 - \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} + \sum_{i=1}^{N} \alpha_i \\
&= \frac{1}{2}\|\mathbf{w}\|^2 - 0 - \mathbf{w}^T \mathbf{w} + \sum_{i=1}^{N} \alpha_i \\
&= -\frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} \alpha_i.
\end{aligned}
\tag{12.41}
$$

Again using the first KKT condition, we can rewrite the first term.

$$
\begin{aligned}
-\frac{1}{2}\|\mathbf{w}\|^2 &= -\frac{1}{2}\Big(\sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)}\Big) \cdot \Big(\sum_{j=1}^{N} \alpha_j y^{(j)} \mathbf{x}^{(j)}\Big) \\
&= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)}\, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}.
\end{aligned}
\tag{12.42}
$$

Plugging (12.42) into the (simplified) Lagrangian (12.41), we see that the Lagrangian now depends on $\boldsymbol{\alpha}$ only.

Problem 12.21. *The **dual problem** of (12.35) is formulated as*

$$\max_{\boldsymbol{\alpha}} \Big[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \Big], \quad \textit{subj.to} \quad \begin{bmatrix} \alpha_i \geq 0, & \forall \, i, \\ \sum_{i=1}^{N} \alpha_i y^{(i)} = 0. \end{bmatrix} \tag{12.43}$$

Remark 12.22. (Solving the dual problem).

- We can solve the dual problem (12.43), by using either a generic **quadratic programming** solver or the **Sequential Minimal Optimization (SMO)**.
- For now, **assume** that we solved it to have $\boldsymbol{\alpha}^* = [\alpha_1^*, \cdots, \alpha_n^*]^T$.
- Then we can plug it into the first KKT condition to get

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y^{(i)} \mathbf{x}^{(i)}. \tag{12.44}$$

- We still need to get w_0^*.

Support vectors

Assume momentarily that we have w_0^*. Consider the **complementary slackness KKT condition** along with the primal and dual feasibility conditions:

$$\begin{aligned}
&\alpha_i^* \left[y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) - 1\right] = 0 \\
&\Rightarrow \begin{cases}
\alpha_i^* > 0 \Rightarrow y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) = 1 \\
\alpha_i^* < 0 \quad \text{(can't happen)} \\
y^{(i)}(w_0^* + \mathbf{w}^{T}\mathbf{x}^{(i)}) - 1 > 0 \Rightarrow \alpha_i^* = 0 \\
y^{(i)}(w_0^* + \mathbf{w}^{T}\mathbf{x}^{(i)}) - 1 < 0 \quad \text{(can't happen)}.
\end{cases}
\end{aligned} \tag{12.45}$$

We define the **optimal (scaled) scoring function**:

$$f^*(\mathbf{x}^{(i)}) = w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}. \tag{12.46}$$

Then

$$\begin{cases}
\alpha_i^* > 0 & \Rightarrow \quad y^{(i)} f^*(\mathbf{x}^{(i)}) = \text{scaled margin} = 1, \\
y^{(i)} f^*(\mathbf{x}^{(i)}) > 1 & \Rightarrow \quad \alpha_i^* = 0.
\end{cases} \tag{12.47}$$

Definition 12.23. The examples in the first category, for which the scaled margin is 1 and the constraints are active, are called **support vectors**. They are the closest to the decision boundary.

Finding the optimal value of w_0

To get w_0^*, use the primal feasibility condition:

$$y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) \geq 1 \quad \text{and} \quad \min_i y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) = 1.$$

If you take a positive support vector ($y^{(i)} = 1$), then

$$w_0^* = 1 - \min_{i:y^{(i)}=1} \mathbf{w}^{*T}\mathbf{x}^{(i)}. \tag{12.48}$$

Here, you'd better refer to **Summary of SVM** in Algorithm 12.26, p. 452.

12.4.2. Soft-margin classification for inseparable data

When the dataset is inseparable, there would be no separating hyperplane; there is no feasible solution to the linear SVM.

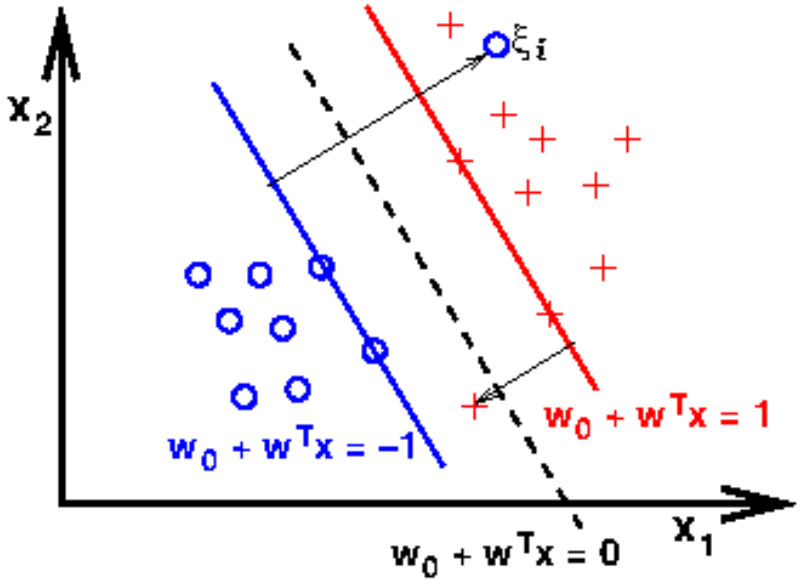


Figure 12.14: Slack variable: ξ_i.

Let's fix our SVM so it can accommodate the inseparable case.

- The new formulation involves the **slack variable**; it allows some instances to fall off the margin, but penalize them.
- So we are allowed to make mistakes now, but we pay a price.

Note: The motivation for introducing the slack variable ξ was that the linear constraints need to be relaxed for nonlinearly separable data to allow the convergence of the optimization in the presence of misclassifications, under appropriate cost penalization. Such strategy of SVM is called the **soft-margin classification**.

Recall: The **linear SVM** formulated in Problem 12.18:

$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subj.to} \quad y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1 \geq 0, \quad \forall\, i. \qquad \text{(Primal)} \qquad (12.49)$$

Let's change it to this new primal problem:

Problem 12.24. ***(Soft-margin classification).*** *The SVM with the slack variable is formulated as*

$$\min_{\mathbf{w},w_0,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \quad \textit{subj.to} \quad \begin{cases} y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{cases} \qquad \textit{(Primal)} \qquad (12.50)$$

Via the variable C, we can then control the penalty for misclassification. **Large values of** C correspond to large error penalties, whereas we are less strict about misclassification errors if we choose smaller values for C. We can then use the C parameter to control the width of the margin and therefore tune the **bias-variance trade-off**, as illustrated in the following figure:

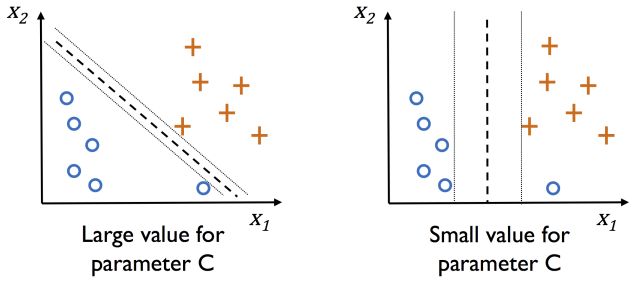


Figure 12.15: Bias-variance trade-off, via C.

The constraints allow some slack of size ξ_i, but we pay a price for it in the objective. That is,

if $y^{(i)}f(\mathbf{x}^{(i)}) \geq 1$, then $\xi_i = 0$ and penalty is 0. Otherwise, $y^{(i)}f(\mathbf{x}^{(i)}) = 1 - \xi_i$ and we pay price $\xi_i > 0$

.

Going on a bit for soft-margin classification

We rewrite the penalty another way:

$$\xi_i = \left\{ \begin{array}{ll} 0, & \text{if } y^{(i)} f(\mathbf{x}^{(i)}) \geq 1 \\ 1 - y^{(i)} f(\mathbf{x}^{(i)}), & \text{otherwise} \end{array} \right\} = [1 - y^{(i)} f(\mathbf{x}^{(i)})]_+$$

Thus the objective function for soft-margin classification becomes

$$\min_{\mathbf{w}, w_0, \boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} [1 - y^{(i)} f(\mathbf{x}^{(i)})]_+. \tag{12.51}$$

The Dual for soft-margin classification

Form the Lagrangian of (12.50):

$$\begin{aligned} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) \;=\; & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} r_i \xi_i \\ & - \sum_{i=1}^{N} \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 + \xi_i], \end{aligned} \tag{12.52}$$

where α_i's and r_i's are Lagrange multipliers (constrained to be ≥ 0). After some work, the dual turns out to be

Problem 12.25. The **dual problem** of (12.49) is formulated as

$$\max_{\boldsymbol{\alpha}} \left[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \right], \quad \text{subj.to} \quad \left[\begin{array}{l} 0 \leq \alpha_i \leq C, \quad \forall\, i, \\ \sum_{i=1}^{N} \alpha_i y^{(i)} = 0. \end{array} \right. \tag{12.53}$$

So the only difference from the original problem's Lagrangian (12.43) is that $0 \leq \alpha_i$ was changed to $0 \leq \alpha_i \leq C$. Neat!

Algebraic expression for the dual problem:

Let

$$Z = \begin{bmatrix} y^{(1)}\mathbf{x}^{(1)} \\ y^{(2)}\mathbf{x}^{(2)} \\ \vdots \\ y^{(N)}\mathbf{x}^{(N)} \end{bmatrix} \in \mathbb{R}^{N\times m}, \quad \mathbf{1} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^{N}.$$

Then **dual problem** (12.53) can be written as

$$\max_{0\le\boldsymbol{\alpha}\le C} \left[\boldsymbol{\alpha}\cdot\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T ZZ^T\boldsymbol{\alpha}\right] \quad \text{subj.to} \quad \boldsymbol{\alpha}\cdot\mathbf{y} = 0. \tag{12.54}$$

Note:

- $G = ZZ^T \in \mathbb{R}^{N\times N}$ is called the **Gram matrix**. That is,

$$G_{ij} = y^{(i)}y^{(j)}\,\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}. \tag{12.55}$$

- The optimization problem in (12.53) or (12.54) is a typical form of **quadratic programming** (QP) problems.
- The dual problem (12.54) admits a unique solution.

Algorithm 12.26. (Summary of SVM)

- **Training**
  - Compute Gram matrix: $G_{ij} = y^{(i)}y^{(j)}\,\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$
  - Solve for $\boldsymbol{\alpha}^*$ (QP or the SMO)
  - Compute the weights: $\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y^{(i)} \mathbf{x}^{(i)}$ (12.44)
  - Compute the intercept: $w_0^* = 1 - \min_{i:y^{(i)}=1} \mathbf{w}^{*T}\mathbf{x}^{(i)}$ (12.48)
- **Classification** (for a new sample $\mathbf{x}$)
  - Compute $k_i = \mathbf{x} \cdot \mathbf{x}^{(i)}$ for support vectors $\mathbf{x}^{(i)}$
  - Compute $f(\mathbf{x}) = w_0^* + \sum_i \alpha_i y^{(i)} k_i \;\; (:= w_0^* + \mathbf{w}^{*T}\mathbf{x})$ (12.33)
  - Test $\operatorname{sign}(f(\mathbf{x}))$.

Why are SVMs popular in ML?

- **Margins**
  - to **reduce overfitting**
  - to **enhance classification accuracy**
- **Feature expansion**
  - mapping to a higher-dimension
  - to classify **inseparable datasets**
- **Kernel trick**
  - to avoid writing out high-dimensional feature vectors

12.4.3. Nonlinear SVM and kernel trick

Note: A reason why SVMs enjoy high popularity among machine learning practitioners is that it can be **easily kernelized** to solve nonlinear classification problems incorporating **linearly inseparable data**. The basic idea behind **kernel methods** is to create nonlinear combinations of the original features to project them onto a higher-dimensional space via a mapping ϕ where it becomes linearly separable.

For example, for the inseparable data set in Figure 12.16, we define

$$\phi(x_1, x_2) = (z_1, z_2, z_2) = (x_1, x_2, x_1^2 + x_2^2).$$

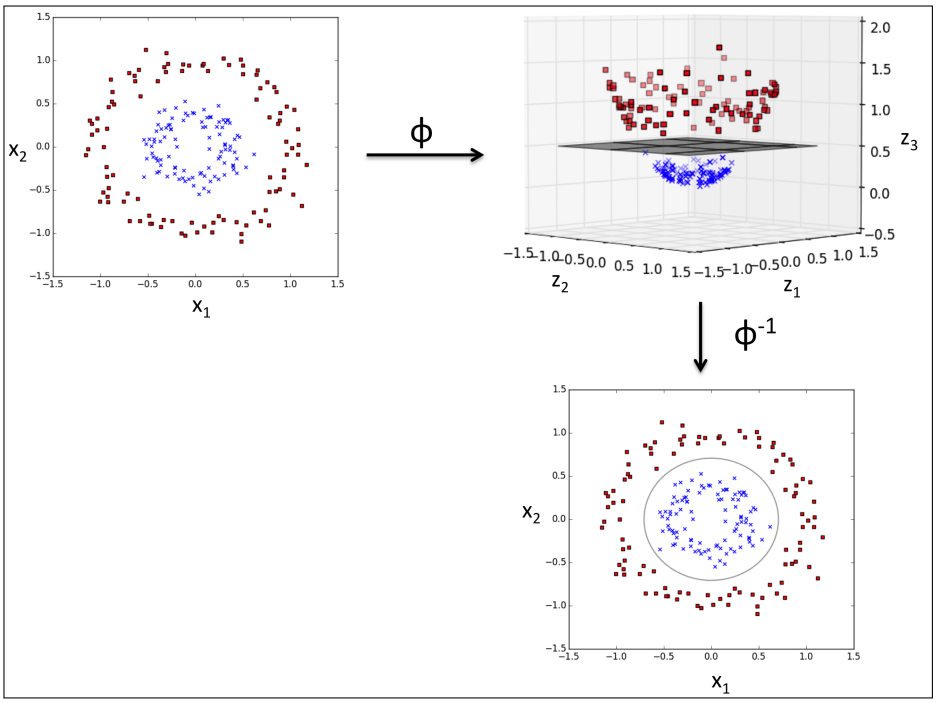


Figure 12.16: Inseparable dataset, feature expansion, and kernel SVM.

To solve a nonlinear problem using an SVM, we would ⓐ *transform the training data onto a higher-dimensional feature space* via a mapping ϕ and ⓑ *train a linear SVM model* to classify the data in this new feature space. Then, we can ⓒ *use the same mapping function ϕ to transform new, unseen data to classify it using the linear SVM model*.

Kernel trick

Recall: the **dual problem** to the soft-margin SVM given in (12.53):

$$\max_{\boldsymbol{\alpha}} \Big[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \Big], \quad \text{subj.to} \quad \begin{cases} 0 \leq \alpha_i \leq C, & \forall \, i, \\ \sum_{i=1}^{N} \alpha_i y^{(i)} = 0. \end{cases} \tag{12.56}$$

Observation 12.27. The objective is a linear combination of dot products $\{\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}\}$. Thus,

- If the kernel SVM transforms the data samples through ϕ, the dot product $\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$ in the objective must be replaced by $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$.
- The dot product $\phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)})$ is performed in a higher-dimension, which may be costly.

Definition 12.28. In order to **save the expensive step of explicit computation** of this dot product (in a higher-dimension), we define a so-called **kernel function**:

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \approx \phi(\mathbf{x}^{(i)}) \cdot \phi(\mathbf{x}^{(j)}). \tag{12.57}$$

One of the most widely used kernels is the **Radial Basis Function (RBF)** kernel or simply called the **Gaussian kernel**:

$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\Big(- \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}{2\sigma^2} \Big) = \exp\Big(-\gamma \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2 \Big), \tag{12.58}$$

where $\gamma = 1/(2\sigma^2)$. Occasionally, the parameter γ plays an important role in controlling overfitting.

Note: Roughly speaking, the term **kernel** can be interpreted as a **similarity function** between a pair of samples. The minus sign inverts the distance measure into a similarity score, and, due to the exponential term, the resulting similarity score will fall into a range between 1 (for exactly similar samples) and 0 (for very dissimilar samples).

This is the big picture behind the **kernel trick**.

(Kernel SVM algorithm): It can be summarized as in Algorithm 12.26, p. 452; only the difference is that dot products $\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}$ and $\mathbf{x} \cdot \mathbf{x}^{(i)}$ are replaced by $\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ and $\mathcal{K}(\mathbf{x}, \mathbf{x}^{(i)})$, respectively.

Common kernels

- Polynomial of degree exactly k (e.g. $k = 2$):
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^k \tag{12.59}$$
- Polynomial of degree up to k: for some $c > 0$,
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (c + \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^k \tag{12.60}$$
- Sigmoid:
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(a\, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} + b) \tag{12.61}$$
- Gaussian RBF:
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\Big(- \frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}{2\sigma^2} \Big) \tag{12.62}$$
- And many others: Fisher kernel, graph kernel, string kernel, ... **very active area of research!**

Example 12.29. (Quadratic kernels). $\mathcal{K}(\mathbf{x}, \mathbf{z}) = (c + \mathbf{x} \cdot \mathbf{z})^2$

$$\begin{aligned}(c + \mathbf{x} \cdot \mathbf{z})^2 &= \Big(c + \sum_{j=1}^m x_j z_j\Big)\Big(c + \sum_{\ell=1}^m x_\ell z_\ell\Big) \\ &= c^2 + 2c \sum_{j=1}^m x_j z_j + \sum_{j=1}^m \sum_{\ell=1}^m x_j z_j x_\ell z_\ell \\ &= c^2 + \sum_{j=1}^m (\sqrt{2c} x_j)(\sqrt{2c} z_j) + \sum_{j,\ell=1}^m (x_j x_\ell)(z_j z_\ell).\end{aligned} \tag{12.63}$$

So, the corresponding **feature expansion** is given by

$$\phi([x_1, \cdots, x_m]) = [x_1^2, x_1 x_2, \cdots, x_m x_{m-1}, x_m^2,\ \sqrt{2c} x_1, \cdots, \sqrt{2c} x_m,\ c], \tag{12.64}$$

which is in $\mathbb{R}^{m^2+m+1}$. ***Q**: How is this feature expansion meaningful?*

Although **not** expressible by $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ [ever, or in finite dimensions], the kernel $\mathcal{K}(\mathbf{x}, \mathbf{z})$ must be formulated meaningfully!

Summary: Linear classifiers & their variants

- **Perceptrons** are a simple, popular way to learn a classifier
- They suffer from inefficient use of data, overfitting, and lack of expressiveness
- **SVMs** can fix these problems using ① **margins** and ② **feature expansion** (mapping to a higher-dimension)
- In order to make feature expansion computationally feasible, we need the ③ **kernel trick**, which avoids writing out high-dimensional feature vectors
- *SVMs are popular classifiers because they usually achieve good error rates and can handle unusual types of data*

Focus on solutions instead of problems;
ask what is possible instead of what is wrong.

12.5. k-Nearest Neighbors

The **k-nearest neighbor** (k-NN) classifier is a typical example of a **lazy learner**.

- It is called *lazy* not because of its apparent simplicity, but because it **doesn't learn a discriminative function** from the training data, but memorizes the training dataset instead.
- Analysis of the training data is **delayed until a query is made** to the system.

Algorithm 12.30. (k-NN algorithm). The algorithm itself is fairly straightforward and can be summarized by the following steps:

1. Choose the number k and a distance metric.
2. For the new sample, find the k-nearest neighbors.
3. Assign the class label by majority vote.

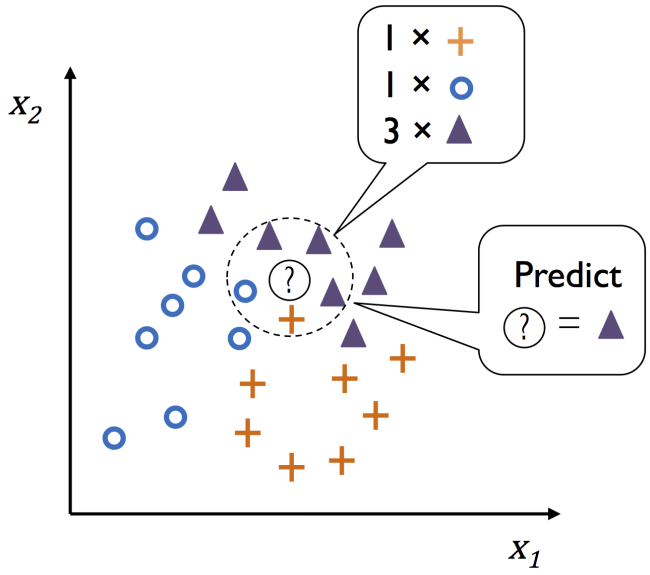


Figure 12.17: Illustration for how a new data point (?) is assigned the triangle class label, based on majority voting, when $k = 5$.

k-NN: pros and cons

- Since it is memory-based, the classifier **immediately adapts** as we collect new training data.
- **(Prediction Cost)** The **computational complexity** for classifying new samples grows linearly with the number of samples in the training dataset in the worst-case scenario.[a]
- Furthermore, we can't discard training samples since *no training step* is involved. Thus, **storage space** can become a challenge if we are working with large datasets.

[a] J. H. Friedman, J. L. Bentley, and R.A. Finkel (1977). *An Algorithm for Finding Best Matches in Logarithmic Expected Time*, ACM transactions on Mathematical Software (TOMS), **3**, no. 3, pp. 209–226. The algorithm in the article is called the **KD-tree**.

k-NN: what to choose k and a distance metric?

- The **right choice of k is crucial** to find a good balance between overfitting and underfitting.
  (For `sklearn.neighbors.KNeighborsClassifier`, default `n_neighbors = 5`.)
- We also choose a distance metric that is appropriate for the features in the dataset. (e.g., the simple Euclidean distance, along with data standardization)
- Alternatively, we can choose the **Minkowski distance**:

$$d(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|_p \overset{\text{def}}{=\!=} \left(\sum_{i=1}^{m} |x_i - z_i|^p \right)^{1/p}. \tag{12.65}$$

  (For `sklearn.neighbors.KNeighborsClassifier`, default `p = 2`.)

12.6. Neural Networks

Recall: The **Perceptron** (or, Adaline, Logistic Regression) is **the simplest artificial neuron** that makes decisions by **weighting up** evidence.

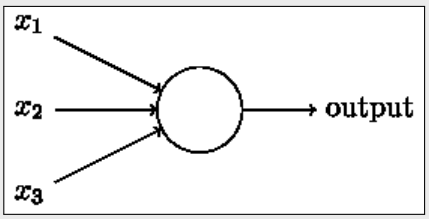


Figure 12.18: A simplest artificial neuron.

Complex Neural Networks

- Obviously, a simple artificial neuron is not a complete model of human decision-making!
- However, they can be use as **building blocks** for more complex neural networks.

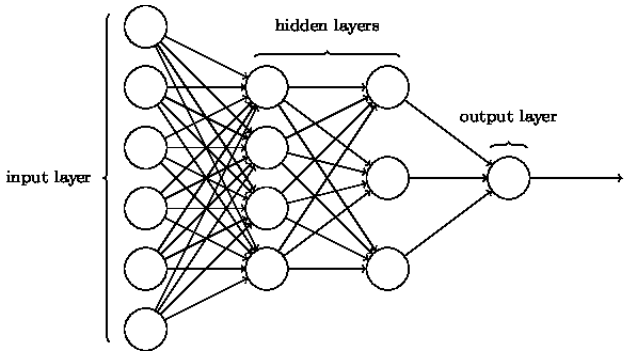


Figure 12.19: A complex neural network.

Remark **12.31.** In machine learning, a **black box model** refers to a model whose internal workings and decision-making process are difficult to understand, even for the developers.

- Neural networks are mostly black box models.
- Their accuracy may not be better than other algorithms, for tabular datasets.
- Their accuracy may not increase with more hidden layers.

A Simple Network to Classify Handwritten Digits

- The problem of recognizing handwritten digits has two components: segmentation and classification.

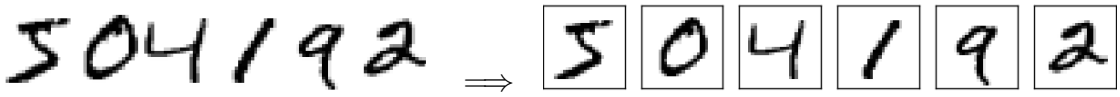


Figure 12.20: Segmentation.

- We'll focus on algorithmic components for the classification of individual digits.

MNIST Dataset:

A modified subset of two data sets collected by NIST (US National Institute of Standards and Technology):

- Its first part contains 60,000 images (for training)
- The second part is 10,000 images (for test), each of which is in 28×28 grayscale pixels

A Simple Neural Network

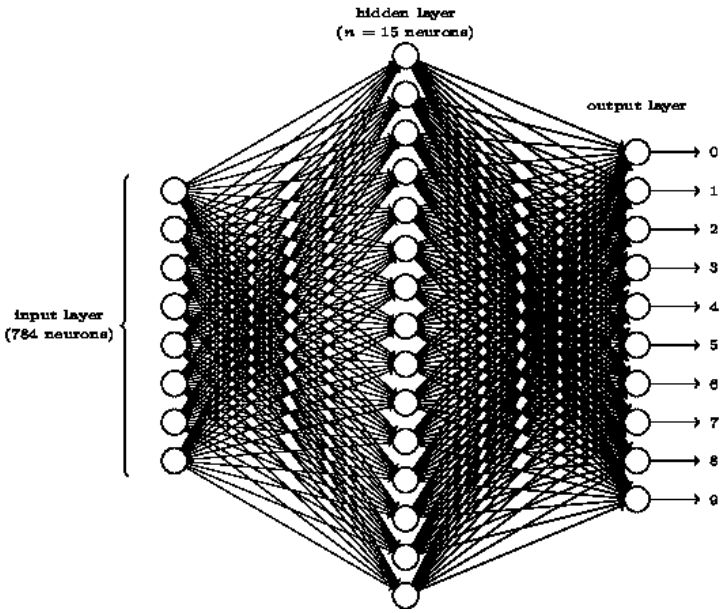


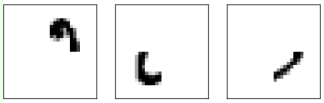
Figure 12.21: A sigmoid network **having a single hidden layer**.

What the Neural Network Will Do

- Let's concentrate on **the first output neuron**, the one that is trying to decide whether or not the input digit is a **0**.
- It does this by weighing up evidence from the hidden layer of neurons.

- **What are those hidden neurons doing?**
- Let's suppose **for the sake of argument** that **the first neuron** in the hidden layer may detect whether or not an image like the following is present

  It can do this by heavily weighting input pixels which overlap with the image, and only lightly weighting the other inputs.
- Similarly, let's suppose that **the second, third, and fourth neurons** in the hidden layer detect whether or not the following images are present



- As you may have guessed, these four images together make up the 0 image that we saw in the line of digits shown in Figure 12.20:

- So if all four of these hidden neurons are firing, then we can conclude that the digit is a 0.

Learning with Gradient Descent

- **Data set** $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}$, $i = 1, 2, \cdots, N$
  (e.g., if an image $\mathbf{x}^{(k)}$ depicts a 2, then $\mathbf{y}^{(k)} = (0, 0, 1, 0, 0, 0, 0, 0, 0, 0)^T$.)
- **Cost function**

$$C(\boldsymbol{W}, B) = \frac{1}{2N} \sum_i ||\mathbf{y}^{(i)} - \mathbf{a}(\mathbf{x}^{(i)})||^2, \tag{12.66}$$

  where $\boldsymbol{W}$ denotes the collection of all weights in the network, B all the biases, and $\mathbf{a}(\mathbf{x}^{(i)})$ is the vector of outputs from the network when $\mathbf{x}^{(i)}$ is input.
- **Gradient descent method**

$$\begin{bmatrix} \boldsymbol{W} \\ B \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{W} \\ B \end{bmatrix} + \begin{bmatrix} \Delta\boldsymbol{W} \\ \Delta B \end{bmatrix}, \tag{12.67}$$

  where

$$\begin{bmatrix} \Delta\boldsymbol{W} \\ \Delta B \end{bmatrix} = -\eta \begin{bmatrix} \nabla_{\boldsymbol{W}} C \\ \nabla_B C \end{bmatrix}.$$

Note: To compute the gradient ∇C, we need to compute the gradients $\nabla C_{\mathbf{x}^{(i)}}$ separately for each training input, $\mathbf{x}^{(i)}$, and then average them:

$$\nabla C = \frac{1}{N} \sum_i \nabla C_{\mathbf{x}^{(i)}}. \tag{12.68}$$

Unfortunately, when the number of training inputs is very large, it can take a long time, and learning thus occurs slowly. An idea called **stochastic gradient descent** can be used to speed up learning.

Stochastic Gradient Descent

The idea is to estimate the gradient ∇C by computing $\nabla C_{\mathbf{x}^{(i)}}$ for a **small sample of randomly chosen training inputs**. By averaging over this small sample, it turns out that we can quickly get a good estimate of the true gradient ∇C; this helps speed up gradient descent, and thus learning.

- Pick out a small number of randomly chosen training inputs ($m \ll N$):

$$\widetilde{\mathbf{x}}^{(1)}, \widetilde{\mathbf{x}}^{(2)}, \cdots, \widetilde{\mathbf{x}}^{(m)},$$

  which we refer to as a **mini-batch**.
- Average $\nabla C_{\widetilde{\mathbf{x}}^{(k)}}$ to approximate the gradient ∇C. That is,

$$\frac{1}{m}\sum_{k=1}^{m} \nabla C_{\widetilde{\mathbf{x}}^{(k)}} \approx \nabla C \overset{\text{def}}{=\!=} \frac{1}{N}\sum_{i} \nabla C_{\mathbf{x}^{(i)}}. \tag{12.69}$$

- For classification of handwritten digits for the MNIST data set, you may choose: `batch_size = 10`.

Note: In practice, you can implement the stochastic gradient descent as follows. **For an epoch**,

- Shuffle the data set
- For each m samples (selected from the beginning), update $(\boldsymbol{W}, B)$ using the approximate gradient (12.69).

Implementing a Network to Classify Digits [27]

network.py

```
"""
network.py       (by Michael Nielsen)
~~~~~~~~~~
A module to implement the stochastic gradient descent learning
algorithm for a feedforward neural network.  Gradients are calculated
using backpropagation. """
#### Libraries
# Standard library
import random
# Third-party libraries
import numpy as np

class Network(object):
    def __init__(self, sizes):
        """The list ``sizes`` contains the number of neurons in the
        respective layers of the network.  For example, if the list
        was [2, 3, 1] then it would be a three-layer network, with the
        first layer containing 2 neurons, the second layer 3 neurons,
        and the third layer 1 neuron. """

        self.num_layers = len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        self.weights = [np.random.randn(y, x)
                        for x, y in zip(sizes[:-1], sizes[1:])]

    def feedforward(self, a):
        """Return the output of the network if ``a`` is input."""
        for b, w in zip(self.biases, self.weights):
            a = sigmoid(np.dot(w, a)+b)
        return a

    def SGD(self, training_data, epochs, mini_batch_size, eta,
            test_data=None):
        """Train the neural network using mini-batch stochastic
        gradient descent.  The ``training_data`` is a list of tuples
        ``(x, y)`` representing the training inputs and the desired
        outputs.  """

        if test_data: n_test = len(test_data)
        n = len(training_data)
        for j in xrange(epochs):
            random.shuffle(training_data)
            mini_batches = [
                training_data[k:k+mini_batch_size]
```

```
                for k in xrange(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta)
            if test_data:
                print "Epoch {0}: {1} / {2}".format(
                    j, self.evaluate(test_data), n_test)
            else:
                print "Epoch {0} complete".format(j)

    def update_mini_batch(self, mini_batch, eta):
        """Update the network's weights and biases by applying
        gradient descent using backpropagation to a single mini batch.
        The ``mini_batch`` is a list of tuples ``(x, y)``, and ``eta``
        is the learning rate."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        for x, y in mini_batch:
            delta_nabla_b, delta_nabla_w = self.backprop(x, y)
            nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
            nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
        self.weights = [w-(eta/len(mini_batch))*nw
                        for w, nw in zip(self.weights, nabla_w)]
        self.biases = [b-(eta/len(mini_batch))*nb
                       for b, nb in zip(self.biases, nabla_b)]

    def backprop(self, x, y):
        """Return a tuple ``(nabla_b, nabla_w)`` representing the
        gradient for the cost function C_x.  ``nabla_b`` and
        ``nabla_w`` are layer-by-layer lists of numpy arrays, similar
        to ``self.biases`` and ``self.weights``."""
        nabla_b = [np.zeros(b.shape) for b in self.biases]
        nabla_w = [np.zeros(w.shape) for w in self.weights]
        # feedforward
        activation = x
        activations = [x] #list to store all the activations, layer by layer
        zs = [] # list to store all the z vectors, layer by layer
        for b, w in zip(self.biases, self.weights):
            z = np.dot(w, activation)+b
            zs.append(z)
            activation = sigmoid(z)
            activations.append(activation)
        # backward pass
        delta = self.cost_derivative(activations[-1], y) * \
            sigmoid_prime(zs[-1])
        nabla_b[-1] = delta
        nabla_w[-1] = np.dot(delta, activations[-2].transpose())

```

```
        for l in xrange(2, self.num_layers):
            z = zs[-l]
            sp = sigmoid_prime(z)
            delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
            nabla_b[-l] = delta
            nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
        return (nabla_b, nabla_w)

    def evaluate(self, test_data):
        test_results = [(np.argmax(self.feedforward(x)), y)
                        for (x, y) in test_data]
        return sum(int(x == y) for (x, y) in test_results)

    def cost_derivative(self, output_activations, y):
        """Return the vector of partial derivatives \partial C_x /
        \partial a for the output activations."""
        return (output_activations-y)

#### Miscellaneous functions
def sigmoid(z):
    return 1.0/(1.0+np.exp(-z))

def sigmoid_prime(z):
    return sigmoid(z)*(1-sigmoid(z))
```

The code is executed using

Run_network.py

```
import mnist_loader
training_data, validation_data, test_data = mnist_loader.load_data_wrapper()

import network
n_neurons = 20
net = network.Network([784 , n_neurons, 10])

n_epochs, batch_size, eta = 30, 10, 3.0
net.SGD(training_data , n_epochs, batch_size, eta, test_data = test_data)
```

`len(training_data)=50000, len(validation_data)=10000, len(test_data)=10000`

Validation Accuracy

Validation Accuracy

```
Epoch 0: 9006 / 10000
Epoch 1: 9128 / 10000
Epoch 2: 9202 / 10000
Epoch 3: 9188 / 10000
Epoch 4: 9249 / 10000
 ...
Epoch 25: 9356 / 10000
Epoch 26: 9388 / 10000
Epoch 27: 9407 / 10000
Epoch 28: 9410 / 10000
Epoch 29: 9428 / 10000
```

Accuracy Comparisons

- scikit-learn's SVM classifier using the default settings: 9435/10000
- A well-tuned SVM: ≈98.5%
- Well-designed **Convolutional NN** (**CNN**):
  9979/10000 **(only 21 missed!)**

Note: For **well-designed deep neural networks**, the performance is close to **human-equivalent**, and is **arguably better**, since quite a few of the MNIST images are difficult even for humans to recognize with confidence, e.g.,

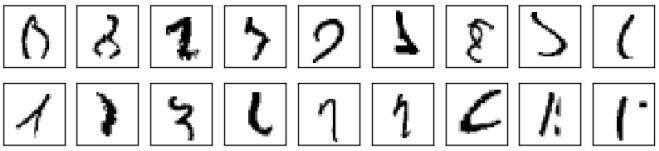


Figure 12.22: MNIST images difficult even for humans to recognize.

Moral of Deep Neural Networks

- Let all the complexity be learned, automatically, from data
- Simple algorithms can perform well for some problems:

(sophisticated algorithm) $\leq$ **(simple learning algorithm + good training data)**

CNN - Convolutional Neural Networks - Miner and the Jeweler

YouTube (Musical) (Melodic Rap)

Deep Learning Packages

- **Keras**
  - A **high-level** ANN *application programming interface* (**API**)
  - It can run on top of TensorFlow, Theano, and Microsoft Cognitive Toolkit (CNTK).
  - Keras focuses on being modular, **user-friendly**, and extensible.
  - Written in **Python**
- **Tensorflow**
  - **Low and High API levels**: An end-to-end open-source deep learning framework developed by **Google** in 2015
  - It is a symbolic math library used for neural networks $\Rightarrow$ **fast!**
  - Tensorflow is **difficult** to use and debug.
  - Written in **Python**, **C++**, **CUDA**
- **Pytorch**
  - **Low and High API levels**: It is a deep learning framework based on Torch, developed by **Facebook** in 2017, and taken over by the PyTorch Foundation (part of Linux Foundation) in late 2022.
  - It has **outstanding community support and development**.
  - Pytorch is **much easier** to use and debug than Tensorflow.
  - Written in **Lua**

You will experience one of them; see Exercise 12.2.

Remark 12.32. You do not need to learn C++ or Lua.

- Currently, **Keras** is most popular due to its simplicity and long history.
- **Pytorch** is most rapidly growing.
  - It can be viewed as a **trade-off** between Keras and Tensorflow.
  - It is particularly good for **natural language processing applications**.
  - **Mathematicians and experienced researchers** will find Pytorch better than Keras, for many applications.

12.7. A Machine Learning Modelcode: Scikit-Learn Comparisons and Ensembling

In machine learning, you can write a code **easily and effectively** using the following **modelcode**. It is also useful for **algorithm comparisons and ensembling**. You may download

https://skim.math.msstate.edu/LectureNotes/data/Machine-Learning-Modelcode.PY.tar.

Machine_Learning_Model.py

```
import numpy as np; import pandas as pd; import time
import seaborn as sbn; import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import datasets
np.set_printoptions(suppress=True)

#=====================================================================
# Upload a Dataset: print(dir(datasets))
# load_iris, load_wine, load_breast_cancer, ...
#=====================================================================
data_read = datasets.load_iris();  #print(data_read.keys())

X = data_read.data
y = data_read.target
dataname = data_read.filename
targets  = data_read.target_names
features = data_read.feature_names

#---------------------------------------------------------------------
# SETTING
#---------------------------------------------------------------------
N,d = X.shape; nclass=len(set(y));
print('DATA: N, d, nclass =',N,d,nclass)
rtrain = 0.7e0; run = 50; CompEnsm = 2;

def multi_run(clf,X,y,rtrain,run):
    t0 = time.time(); acc = np.zeros([run,1])
    for it in range(run):
        Xtrain, Xtest, ytrain, ytest = train_test_split(
            X, y, train_size=rtrain, random_state=it, stratify = y)
        clf.fit(Xtrain, ytrain);
        acc[it] = clf.score(Xtest, ytest)
    etime = time.time()-t0
    return np.mean(acc)*100, np.std(acc)*100, etime # accmean,acc_std,etime
```

```
35
36 #=======================================================================
37 # My Classifier
38 #=======================================================================
39 from myclf import *    # My Classifier = MyCLF()
40 if 'MyCLF' in locals():
41     accmean, acc_std, etime = multi_run(MyCLF(mode=1),X,y,rtrain,run)
42
43     print('%s: MyCLF()      : Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
44           %(dataname,accmean,acc_std,etime/run))
45
46 #=======================================================================
47 # Scikit-learn Classifiers, for Comparisions && Ensembling
48 #=======================================================================
49 if CompEnsm >= 1:
50     exec(open("sklearn_classifiers.py").read())
```

myclf.py

```
1  import numpy as np
2  from sklearn.base import BaseEstimator, ClassifierMixin
3  from sklearn.tree import DecisionTreeClassifier
4
5  class MyCLF(BaseEstimator, ClassifierMixin):        #a child class
6      def __init__(self, mode=0, learning_rate=0.01):
7          self.mode = mode
8          self.learning_rate = learning_rate
9          self.clf = DecisionTreeClassifier(max_depth=5)
10         if self.mode==1: print('MyCLF() = %s' %(self.clf))
11
12     def fit(self, X, y):
13         self.clf.fit(X, y)
14
15     def predict(self, X):
16         return self.clf.predict(X)
17
18     def score(self, X, y):
19         return self.clf.score(X, y)
```

Note: Replace `DecisionTreeClassifier()` with your own classier.

- The classifier must be implemented as **a child class** if if it is used in ensembling.

sklearn_classifiers.py

```
#=========================================================================
# Required: X, y, multi_run [dataname, rtrain, run, CompEnsm]
#=========================================================================
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from sklearn.ensemble import VotingClassifier

#-----------------------------------------------
classifiers = [
    LogisticRegression(max_iter = 1000),
    KNeighborsClassifier(5),
    SVC(kernel="linear", C=0.5),
    SVC(gamma=2, C=1),
    RandomForestClassifier(max_depth=5, n_estimators=50, max_features=1),
    MLPClassifier(hidden_layer_sizes=[100], activation='logistic',
                  alpha=0.5, max_iter=1000),
    AdaBoostClassifier(),
    GaussianNB(),
    QuadraticDiscriminantAnalysis(),
    GaussianProcessClassifier(),
]
names = [
    "Logistic-Regr",
    "KNeighbors-5 ",
    "SVC-Linear   ",
    "SVC-RBF      ",
    "Random-Forest",
    "MLPClassifier",
    "AdaBoost     ",
    "Naive-Bayes  ",
    "QDA          ",
    "Gaussian-Proc",
]
```

```
#-------------------------------------------
if dataname is None: dataname = 'No-dataname';
if run      is None: run      = 50;
if rtrain   is None: rtrain   = 0.7e0;
if CompEnsm is None: CompEnsm = 2;

#=====================================================================
print('====== Comparision: Scikit-learn Classifiers ================')
#=====================================================================
import os;
acc_max=0; Acc_CLF = np.zeros([len(classifiers),1]);

for k, (name, clf) in enumerate(zip(names, classifiers)):
    accmean, acc_std, etime = multi_run(clf,X,y,rtrain,run)

    Acc_CLF[k] = accmean
    if accmean>acc_max: acc_max,algname = accmean,name
    print('%s: %s: Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
        %(os.path.basename(dataname),name,accmean,acc_std,etime/run))
print('---------------------------------------------------------------')
print('sklearn classifiers Acc: (mean,max) = (%.2f,%.2f)%%; Best = %s'
      %(np.mean(Acc_CLF),acc_max,algname))

if CompEnsm <2: quit()
#=====================================================================
print('====== Ensembling: SKlearn Classifiers ======================')
#=====================================================================
names = [x.rstrip() for x in names]
popped_clf = []
popped_clf.append(names.pop(9)); classifiers.pop(9);  #Gaussian Proc
popped_clf.append(names.pop(7)); classifiers.pop(7);  #Naive Bayes
popped_clf.append(names.pop(6)); classifiers.pop(6);  #AdaBoost
popped_clf.append(names.pop(4)); classifiers.pop(4);  #Random Forest
popped_clf.append(names.pop(0)); classifiers.pop(0);  #Logistic Regr
#print('popped_clf=',popped_clf[::-1])

CLFs = [(name, clf) for name, clf in zip(names, classifiers)]
#if 'MyCLF' in locals(): CLFs += [('MyCLF',MyCLF())]
EnCLF = VotingClassifier(estimators=CLFs, voting='hard')
accmean, acc_std, etime = multi_run(EnCLF,X,y,rtrain,run)

print('EnCLF =',[lis[0] for lis in CLFs])
print('%s: Ensemble CLFs: Acc.(mean,std) = (%.2f,%.2f)%%; E-time= %.5f'
       %(os.path.basename(dataname),accmean,acc_std,etime/run))
```

Output

```
DATA: N, d, nclass = 150 4 3
MyCLF() = DecisionTreeClassifier(max_depth=5)
iris.csv: MyCLF()       : Acc.(mean,std) = (94.53,3.12)%; E-time= 0.00074
====== Comparision: Scikit-learn Classifiers =================
iris.csv: Logistic-Regr: Acc.(mean,std) = (96.13,2.62)%; E-time= 0.01035
iris.csv: KNeighbors-5 : Acc.(mean,std) = (96.49,1.99)%; E-time= 0.00176
iris.csv: SVC-Linear   : Acc.(mean,std) = (97.60,2.26)%; E-time= 0.00085
iris.csv: SVC-RBF      : Acc.(mean,std) = (96.62,2.10)%; E-time= 0.00101
iris.csv: Random-Forest: Acc.(mean,std) = (94.84,3.16)%; E-time= 0.03647
iris.csv: MLPClassifier: Acc.(mean,std) = (98.58,1.32)%; E-time= 0.20549
iris.csv: AdaBoost     : Acc.(mean,std) = (94.40,2.64)%; E-time= 0.04119
iris.csv: Naive-Bayes  : Acc.(mean,std) = (95.11,3.20)%; E-time= 0.00090
iris.csv: QDA          : Acc.(mean,std) = (97.64,2.06)%; E-time= 0.00085
iris.csv: Gaussian-Proc: Acc.(mean,std) = (95.64,2.63)%; E-time= 0.16151
------------------------------------------------------------
sklearn classifiers Acc: (mean,max) = (96.31,98.58)%; Best = MLPClassifier
====== Ensembling: SKlearn Classifiers ======================
EnCLF = ['KNeighbors-5', 'SVC-Linear', 'SVC-RBF', 'MLPClassifier', 'QDA']
iris.csv: Ensemble CLFs: Acc.(mean,std) = (97.60,1.98)%; E-time= 0.22272
```

Ensembling:
You may stack the best and its siblings of other options.

Exercises for Chapter 12

12.1. The modelcode in Section 12.7 will run without requiring any other implementation of yours.

(a) Download the code or save it by copy-and-paste.

(b) Install all imported packages to run the code.

(c) Modification:

- Select another dataset in `Machine_Learning_Model.py`, line 11. (You may use `print(dir(datasets))` to find datasets available.)
- Set different options for some of the classifiers in `sklearn_classifiers.py`, lines 20–30.

(d) Report the output.

Installation: If you are on Ubuntu, for example, you may begin with

Install-Python-packages

```
sudo apt update
sudo apt install python3 -y
sudo apt install python3-pip -y
rehash
sudo pip3 install numpy scipy matplotlib sympy -y
sudo pip3 install sklearn seaborn pandas -y
```

Note: Some parts in the code may complain, due to updates in the sklearn package. In the case, you may replace or comment them out.

12.2. **(Exploring and designing a deep network).** Popular deep learning packages are summarized on page 468. Choose one of them to design a CNN for the MNIST dataset. For each one, if you choose a simple CNN, its test accuracy will become approximately 99%. You can make it better through appropriate additions and modifications. For example, you may try:

(a) Set multiple convolution-pooling layers.

(b) Choose various number of hidden layers and units on each layer.

(c) Select various activation functions.

Explore the package of your choice yourself, to design a CNN showing an accuracy better than 99.5%. **You can do it yourself!**

12.3. Modify `perceptron.py`, p. 431, to get a code for Adaline.

- For a given training dataset, Adaline converges to a unique *weights*, while Perceptron does not.

- Note that the correction terms are accumulated from all data points in each iteration. As a consequence, the learning rate η may be chosen smaller as the number of points increases.

  Implementation: In order to overcome the problem, you may scale the correction terms by the number of data points.

  - Redefine the **cost function** (12.9):

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2N} \sum_{i=1}^{N} \left(y^{(i)} - \phi(z^{(i)}) \right)^2 . \tag{12.70}$$

    where $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$ and $\phi = I$, the identity.
  - Then the correction terms in (12.12) become correspondingly

$$\begin{aligned} \Delta \mathbf{w} &= \eta \frac{1}{N} \sum_{i} \left(y^{(i)} - \phi(z^{(i)}) \right) \mathbf{x}^{(i)}, \\ \Delta b &= \eta \frac{1}{N} \sum_{i} \left(y^{(i)} - \phi(z^{(i)}) \right) . \end{aligned} \tag{12.71}$$

CHAPTER 13

Matrix Analysis in Data Mining

The **Google search engine** works with two main subtasks: finding relevant web pages and sorting them. We will see how these subtasks are mathematically formulated and solved.

This chapter shows only an example in data mining; however it will give you insights for solving other data mining tasks.

Contents of Chapter 13

13.1. Introduction to Data Mining 478
13.2. Vectors and Matrices in Data Mining 482
13.3. Text Mining 487
13.4. Eigenvalue Methods in Data Mining 497
Exercises for Chapter 13 507

13.1. Introduction to Data Mining

Why Mine Data?

Commercial Viewpoint

- Lots of data is being collected and warehoused.
  - Web data, e-commerce
  - Purchases at department/grocery stores
  - Bank/Credit Card transactions
- Computers have become cheaper and more powerful.
- Competitive pressure is strong.
  - **Provide better, customized services**
    (e.g. in Customer Relationship Management)

Scientific Viewpoint

- Data collected and stored at enormous speeds (GB/hour)
  - Remote sensors on a satellite
  - Telescopes scanning the skies
  - Microarrays generating gene expression data
  - Scientific simulations generating terabytes of data
- Traditional techniques infeasible for raw data
- Data mining may help scientists
  - **in classifying and segmenting data**
  - **in Hypothesis Formation**

Mining Large Data Sets - Motivation

- There is often information "hidden" in the data that is not readily evident.
- Human analysts may take weeks to discover useful information.
- Much of the data is never analyzed at all.
  - **Data gap** becomes larger and larger.

What is Data Mining?

- **Data mining** is a process to turn raw data into useful information/patterns.
  - Non-trivial extraction of implicit, previously unknown and potentially useful information from data.
  - Exploration & analysis, by automatic or semi-automatic means, of large quantities of data in order to discover meaningful patterns.
  - **Patterns must be: valid, novel, understandable, and potentially useful**.

Note: Data mining is also called **Knowledge Discovery in Data** (KDD).

Origins of Data Mining

- Draws ideas from machine learning/AI, pattern recognition, statistics, and database systems.
- Traditional Techniques may be unsuitable, due to
  - Enormity of data
  - High dimensionality of data
  - Variety: Heterogeneous, distributed nature of data

Data Mining Methods

- **Prediction Methods**
  - Use some variables to predict unknown or future values of other variables.
- **Description Methods**
  - Find human-interpretable patterns that describe the data.

Data Mining Tasks

- **Classification** [Predictive]
  - Given a collection of records (training set), find a **model** for class attribute as a function of the values of other attributes.
- **Regression** [Predictive]
  - Predict a value of a given continuous valued variable based on the values of other variables, assuming a linear or nonlinear model of dependency.
- **Clustering** [Descriptive]
  - Given a set of data points and a similarity measure among them, find clusters such that data points in one cluster are more similar to one another than points in other clusters.
- **Association Rule Discovery** [Descriptive]
  - Given a set of records each of which contain some number of items from a given collection, produce dependency rules which will predict occurrence of an item based on occurrences of other items.
- **Sequential Pattern Discovery** [Descriptive]
  - Given is a set of objects, with each object associated with its own timeline of events, find rules that predict strong sequential dependencies among different events.
- **Deviation/Anomaly Detection** [Predictive]
  - Detect significant deviations from normal behavior.

Challenges in Data Mining

- Scalability
- Dimensionality
- Complex and Heterogeneous Data
  - Spatial and temporal data
  - Point and interval data
  - Categorical data
  - Graph data
  - semi/un-structured Data
- Data Quality
- Data Ownership and Distribution
- Privacy Preservation

Related Fields

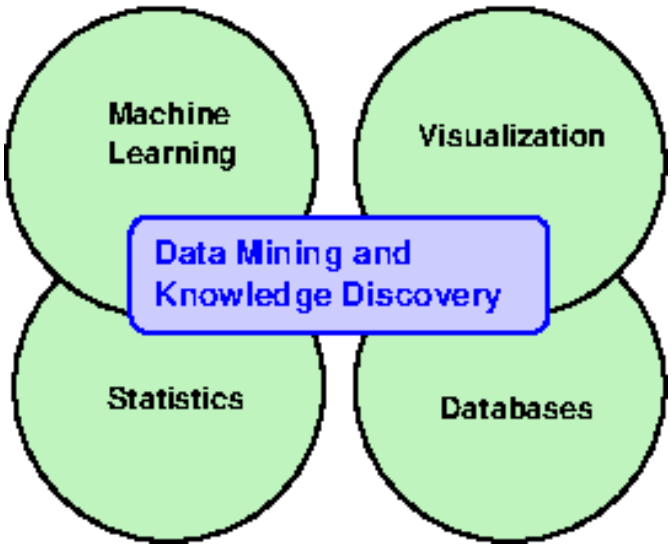


Figure 13.1: Related fields.

13.2. Vectors and Matrices in Data Mining

Note: Often the data are numerical, and the data points can be seen as **vectors** in a high-dimensional vector space. Ensembles of data points can then be organized as **matrices.** In such cases it is natural to use concepts and techniques from linear algebra. Here, we present a modification of *Numerical Linear Algebra in Data Mining* (Eldén, 2006) [10].

In this section, we will consider **two algebraic examples** in data mining: **text mining** and **Google pagerank**. Details will be discussed in following sections.

Example 13.1. **(Text Mining) Term-document matrices** are used in information retrieval. Consider the following set of five documents. Key words, referred to as **terms**, are marked in boldface.

Document 1:	The **Google matrix** P is a model of the **Internet**.
Document 2:	P_{ij} is nonzero if there is a **link** from **web page** j to i.
Document 3:	The **Google matrix** is used to **rank** all **web pages**.
Document 4:	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5:	**England** dropped out of the top 10 in the **FIFA ranking**.

- Counting the frequency of terms in each document we get the following result.

Term	Doc 1	Doc 2	Doc 3	Doc 4	Doc 5
eigenvalue	0	0	0	1	0
England	0	0	0	0	1
FIFA	0	0	0	0	1
Google	1	0	1	0	0
Internet	1	0	0	0	0
link	0	1	0	0	0
matrix	1	0	1	1	0
page	0	1	1	0	0
rank	0	0	1	1	1
web	0	1	1	0	0

The set of terms is called the **dictionary**.

- Each document is represented by a vector in $\mathbb{R}^{10}$, and we can organize the data as a **term-document matrix**,

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{10\times 5}. \tag{13.1}$$

- Assume that we want to find all documents that are relevant with respect to the query "ranking of web pages". This is represented by a query vector, constructed in an analogous way as the term-document matrix, using the same dictionary.

$$\boldsymbol{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{10}. \tag{13.2}$$

  Thus the query itself is considered as a document.

- Now, the information retrieval task can be formulated as a mathematical problem: *find the columns of A that are close to the vector $\boldsymbol{q}$.*
  - To solve this problem we use some distance measure in $\mathbb{R}^{10}$.

Remark 13.2. Information Retrieval, with term-document matrix $A \in \mathbb{R}^{m \times n}$.

- It is common that m **is large**, of the order 10^6, say.
- **The matrix** A **is sparse**, because most of the documents only contain a small fraction of the terms in the dictionary.
- In some methods for information retrieval, linear algebra techniques, such as **singular value decomposition** (**SVD**), are used for **data compression** and **retrieval enhancement**.

Note: **The very idea of data mining is to extract useful information from large and often unstructured datasets**. Therefore

- **the methods must be efficient** and often specially designed for large problems.

Example 13.3. Google Pagerank algorithm. Its main task is to sort the Internet web pages found by **search engines**.

- The core of the **Google search engine** is a matrix computation, probably the largest that is performed routinely.
- The **Google matrix** P is assumed to be of dimension of **the order billions (2005)**, and it is used as a model of (all) the web pages on the Internet.
- In the **Google Pagerank algorithm**, the problem of assigning ranks to all the web pages is formulated as a **matrix eigenvalue problem**.

The Link Matrix, for Google Pagerank

- Let all web pages be ordered from 1 to n, and let i be a particular web page.
- Then O_i will denote the set of pages that i is linked to, the **outlinks**. The number of outlinks is denoted $N_i = |O_i|$.
- The set of inlinks, denoted I_i, are the pages that have an outlink to i.
- Now define Q to be a square matrix of dimension n, and let

$$Q_{ij} = \begin{cases} 1/N_j, & \text{if there is a link from } j \text{ to } i, \\ 0, & \text{otherwise.} \end{cases} \tag{13.3}$$

  - This definition means that row i has nonzero elements in those positions that correspond to inlinks of i.
  - Similarly, column j has nonzero elements equal to $1/N_j$ in those positions that correspond to the outlinks of j.
  - Thus, the sum of each column is either 0 or 1.
- The following **link graph** illustrates a set of web pages with outlinks and inlinks.

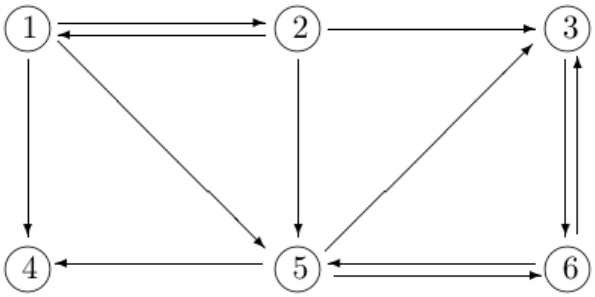


Figure 13.2: A link graph, for six web pages.

The corresponding **link matrix** becomes

$$Q = \begin{bmatrix} 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\ 1/3 & 0 & 0 & 0 & 1/3 & 0 \\ 1/3 & 1/3 & 0 & 0 & 0 & 1/2 \\ 0 & 0 & 1 & 0 & 1/3 & 0 \end{bmatrix} \tag{13.4}$$

- Define a **pagerank vector** r, which holds the ranks of all pages. Then, the vector r can be found as the eigenvector corresponding to the eigenvalue $\lambda = 1$ of Q:

$$Q\mathbf{r} = \lambda\mathbf{r}. \tag{13.5}$$

  We discuss a **realistic formulation of the Pagerank equation** and **numerical aspects of the Pagerank computation** in Section 13.4.

13.3. Text Mining

Definition 13.4. Text mining is methods that extract useful information from large and often unstructured collections of texts.

- A related term is **information retrieval**.
- A typical application is **search in data bases of abstract of scientific papers.**
  - For instance, in **medical applications** one may want to find all the abstracts in the data base that deal with a particular syndrome.
  - So one puts together a search phrase, a query, with key words that are relevant for the syndrome.
  - Then the retrieval system is used to match the query to the documents in the data base, and present to the user all the documents that are relevant, preferably ranked according to relevance.

Example 13.5. The following is a typical query (Eldén, 2006) [10].

9. the use of induced hypothermia in heart surgery, neurosurgery, head injuries and infectious diseases. (13.6)

We will refer to this query as **Q9** in the sequel.

Note: Another well-known area of text mining is **web search engines**.

- There the search phrase is usually very short.
- Often there are so many relevant documents that it is out of the question to present them all to the user.

- In that application the **ranking of the search result** is critical for the efficiency of the search engine. We will come back to this problem in Section 13.4.

Public Domain Text Mining Software

A number of public domain software are available.

- **R**
  - **textmineR**
- **Python**
  - **nltk** (natural language toolkit)
  - **spaCy** (written in Cython)

In this section

We will review one of the most common methods for **text mining**, namely the **vector space model** (Salton *et al.*, 1975) [32].

- In **Example 13.1**, we demonstrated the basic ideas of the construction of a **term-document matrix** in the vector space model.
- Below we first give a very brief overview of the **preprocessing** that is usually done before the actual term-document matrix is set up.
- Then we describe a variant of the vector space model: **Latent Semantic Indexing** (LSI) (Deerwester *et al.*, 1990) [9], which is based on the SVD of the term-document matrix.

13.3.1. The Vector Space Model: Preprocessing and Query Matching

Note: In information retrieval, **keywords** that carry information about the contents of a document are called **terms**.

- A basic task is to create a list of all the terms in alphabetic order, a so called **index**.
- But before the index is made, two preprocessing steps should be done:

  (a) removal of stop words

  (b) stemming

Removal of Stop Words

- **Stop words** are words that one can find in virtually any document.
- The occurrence of such a word in a document does not distinguish this document from other documents.
- The following is the beginning of one stop list

  a, able, about, above, according, accordingly, across, actually, after, afterwards, again, against, ain't, all, allow, allows, almost, alone, along, already, also, although, always, am, among, amongst, an, and, ...

- Various sets of stop words are available on the Internet, e.g. https://countwordsfree.com/stopwords.

Stemming

- **Stemming** is the process of reducing each word that is conjugated or has a suffix to its stem.
- Clearly, from the point of view of information retrieval, no information is lost in the following reduction.

$$\left.\begin{array}{l}\textbf{comput}\text{able}\\ \textbf{comput}\text{ation}\\ \textbf{comput}\text{ing}\\ \textbf{comput}\text{ed}\\ \textbf{comput}\text{ational}\end{array}\right\} \Longrightarrow \textbf{comput}$$

- Public domain stemming algorithms are available on the Internet, e.g. the **Porter Stemming Algorithm** https://tartarus.org/martin/PorterStemmer/.

The Term-Document Matrix

- The **term-document matrix** $A \in \mathbb{R}^{m \times n}$, where

  m = the number of terms in the dictionary
  n = the number of documents

- It is common not only **to count the occurrence of terms in documents** but also **to apply a term weighting scheme**.
- Similarly, **document weighting** is usually done.

Example 13.6. For example, one can define the elements in A by

$$a_{ij} = f_{ij} \log(n/n_i), \tag{13.7}$$

where

- f_{ij} is **term frequency**,
  the number of times term i appears in document j,
- n_i is the number of documents that contain term i
  (*inverse document frequency*).

If a term occurs frequently in only a few documents, then both factors are large. In this case the term discriminates well between different groups of documents, and it gets a large weight in the documents where it appears.

Normally, the term-document matrix is *sparse*: most of the matrix elements are equal to zero.

Example 13.7. For the stemmed Medline collection in Example 13.5, p.487, the matrix is 4163×1063, with 48263 non-zero elements, i.e. approximately 1%. (It includes 30 query columns.) The first 500 rows and columns of the matrix are illustrated in Figure 13.3.

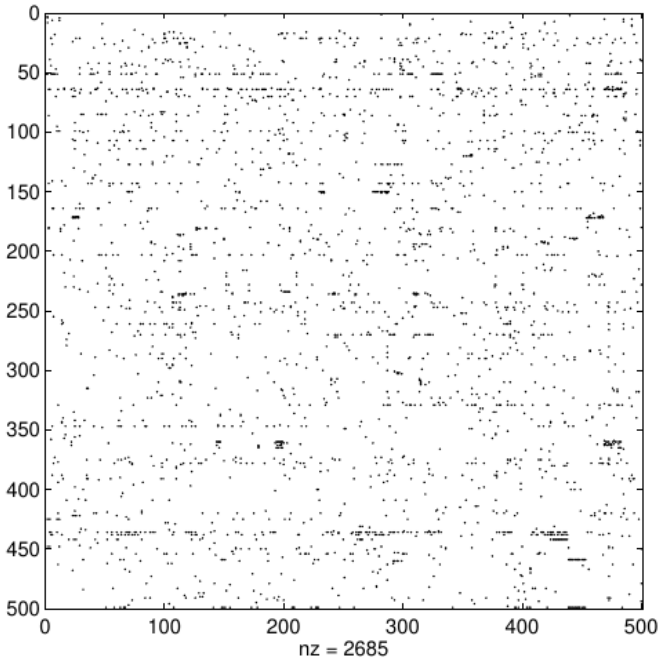


Figure 13.3: The first 500 rows and columns of the Medline matrix. Each dot represents a non-zero element.

Query Matching

- The query is parsed using the same dictionary as the documents, giving a vector $\boldsymbol{q} \in \mathbb{R}^m$.
- **Query matching** is the process of finding all documents that are considered relevant to a particular query $\boldsymbol{q}$.
- This is often done using the **cosine distance measure**: All documents $\{\mathbf{a}_j\}$ are returned for which

$$\frac{\boldsymbol{q} \cdot \mathbf{a}_j}{||\boldsymbol{q}||\,||\mathbf{a}_j||} \geq \texttt{tol}, \tag{13.8}$$

  where `tol` is user-defined tolerance.

Example 13.8. Query matching is performed for query **Q9** in the stemmed Medline collection. With $\texttt{tol} = 0.19$ only a single document is considered relevant. When the tolerance was lowered to 0.17, then three documents are retrieved.

- Irrelevant documents may be returned.
  - For a high value of the tolerance, the retrieved documents are likely to be relevant.
  - When the cosine tolerance is lowered, irrelevant documents may be returned *relatively more*.

Definition 13.9. In performance modelling for information retrieval, we define the following measures:

$$P = \frac{T_r}{T_t} \ (\textbf{precision}) \qquad R = \frac{T_r}{B_r} \ (\textbf{recall}), \tag{13.9}$$

where

T_r = the number of relevant documents retrieved

T_t = the total number of documents retrieved

B_r = the total number of relevant documents in the database

Note: With the cosine measure:

- We see that with a large value of `tol`, we have high precision, but low recall.
- For a small value of `tol`, we have high recall, but low precision.

Example 13.10. Query matching is performed for query **Q9** in the Medline collection using the cosine measure, in order to obtain recall and precision as illustrated in Figure 13.4.

- In the comparison of different methods, it is more illustrative to draw the **recall versus precision diagram**.
- Ideally a method has high recall at the same time as the precision is high. Thus, the closer the curve is to the upper right corner, the better the method is.

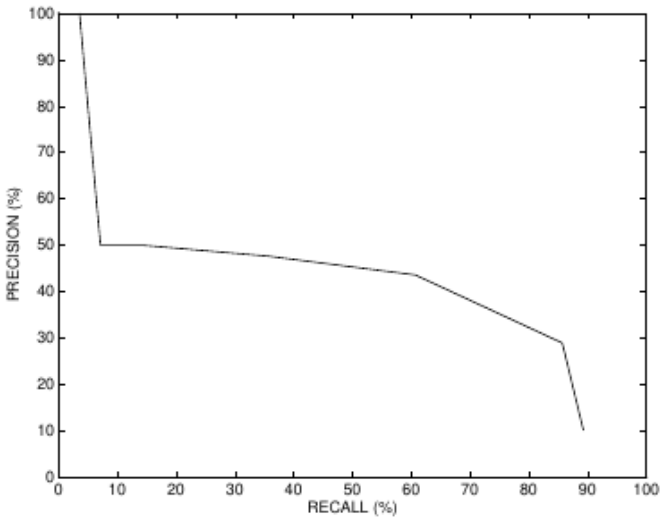


Figure 13.4: Recall versus precision diagram for query matching for Q9, using the vector space method.

13.3.2. Latent Semantic Indexing

Latent Semantic Indexing (LSI) is based on the assumption

- that there is **some underlying latent semantic structure** in the data that is corrupted by the wide variety of words used
- and that this semantic structure can be enhanced by projecting the data onto a lower-dimensional space using the **singular value decomposition**.

Algorithm 13.11. Latent Semantic Indexing (LSI)

- Let $A = U\Sigma V^T$ be the SVD of the term-document matrix.
- Let A_k be its approximation of rank k:

$$A_k = U_k \Sigma_k V_k^T = U_k (\Sigma_k V_k^T) =: U_k D_k, \tag{13.10}$$

  where $V_k \in \mathbb{R}^{n\times k}$ so that $D_k \in \mathbb{R}^{k\times n}$.

  - The columns of U_k live in the document space and are an orthogonal basis that we use to approximate all the documents.
  - Column j of D_k holds the coordinates of document j in terms of the orthogonal basis.

- Note that

$$\boldsymbol{q}^T A_k = \boldsymbol{q}^T U_k D_k = (U_k^T \boldsymbol{q})^T D_k \in \mathbb{R}^{1\times n}. \tag{13.11}$$

- Thus, in query matching, we compute the coordinates of the query in terms of the new document basis and compute the cosines from

$$\cos\theta_j = \frac{\widehat{\boldsymbol{q}}_k \cdot (D_k \mathbf{e}_j)}{||\widehat{\boldsymbol{q}}_k||\,||D_k \mathbf{e}_j||}, \quad \widehat{\boldsymbol{q}}_k = U_k^T \boldsymbol{q}. \tag{13.12}$$

- This means that the query-matching is performed in a k-dimensional space.

Example **13.12.** Query matching is carried out for **Q9** in the Medline collection, approximating the matrix using the truncated SVD with of rank 100 ($k = 100$). The recall-precision curve is given in Figure 13.5. It is seen that for this query, the LSI improves the retrieval performance.

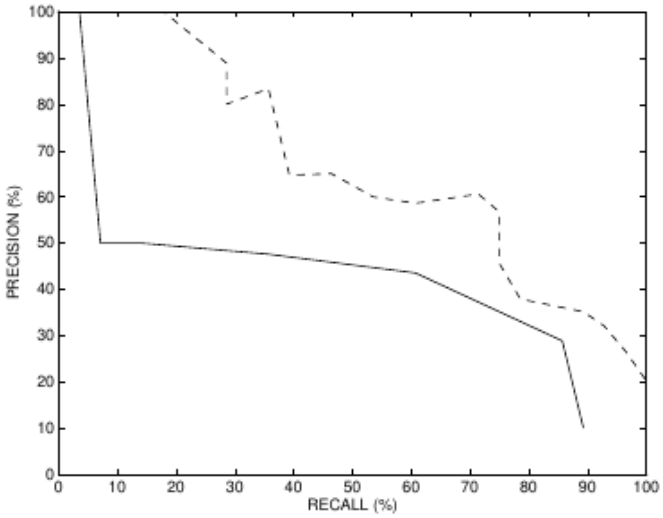


Figure 13.5: Recall versus precision diagram for query matching for Q9, using the full vector space method (solid curve) and the rank 100 approximation (dashed).

Example 13.13. Recall Example 13.1. Consider the term-document matrix $A \in \mathbb{R}^{10\times 5}$ and the query vector $\boldsymbol{q} \in \mathbb{R}^{10}$, of which the query is "**ranking** of **web pages**". See pages 482–483 for details.

Document 1:	The **Google matrix** P is a model of the **Internet**.
Document 2:	P_{ij} is nonzero if there is a **link** from **web page** j to i.
Document 3:	The **Google matrix** is used to **rank** all **web pages**.
Document 4:	The **ranking** is done by solving a **matrix eigenvalue** problem.
Document 5:	**England** dropped out of the top 10 in the **FIFA ranking**.

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{10\times 5}, \qquad \boldsymbol{q} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \end{bmatrix} \in \mathbb{R}^{10}.$$

- (Eldén, 2006) [10]
  "**Obviously, Documents 1-4 are relevant with respect to the query, while Document 5 is totally irrelevant.** However, we obtain the following cosines for query and the original data

  $$(0 \;\; 0.6667 \;\; 0.7746 \;\; 0.3333 \;\; 0.3333)$$

  We then compute the SVD of the term-document matrix, and use a rank 2 approximation. After projection to the two-dimensional subspace the cosines, computed according to (13.12), are

  $$(0.7857 \quad 0.8332 \quad 0.9670 \quad 0.4873 \quad 0.1819)$$

  It turns out that Document 1, which was deemed totally irrelevant for the query in the original representation, is now highly relevant. In addition, the scores for the relevant Documents 2-4 have been reinforced. At the same time, the score for Document 5 has been significantly reduced."
- **However, I view it as a warning.**

13.4. Eigenvalue Methods in Data Mining

Note: An **Internet search** performs two major operations, using a search engine.

(a) **Traditional text processing**. The aim is to find all the web pages containing the words of the query.

(b) **Sorting out**.

- Due to **the massive size of the Web**, the number of hits is likely to be much too large to be handled by the user.
- Therefore, **some measure of quality is needed to sort out the pages** that are likely to be most relevant to the particular query.

When one uses a web search engine, then typically the search phrase is under-specified.

Example 13.14. A **Google search** conducted on October 21, 2022, using the search phrase "**university**":

- The result: links to universities, including *Mississippi State University, University of Arizona, University of Washington - Seattle, University of WisconsinMadison, The University of Texas at Austin*, and *University of Southern California - Los Angeles*.
- The total number of web pages relevant to the search phrase was more than 7 billions.

Remark 13.15. **Google** uses an algorithm (**Pagerank**) for ranking all the web pages that agrees rather well with a common-sense quality measure.

- Google assigns a high rank to a web page, if it has **inlinks** from other pages that have a high rank.
- We will see that this "**self-referencing**" statement can be formulated mathematically as an eigenvalue problem.

13.4.1. Pagerank: Perron-Frobenius Theory

Note: Google uses the concept of **Pagerank** as a quality measure of web pages. It is based on the assumption that

the number of links to and from a page gives information about the importance of the page.

- Let all web pages be ordered from 1 to n, and let i be a particular web page.
- Then O_i will denote the set of pages that i is linked to, the **outlinks**. The number of outlinks is denoted $N_i = |O_i|$.
- The set of **inlinks**, denoted I_i, are the pages that have an outlink to i.

Note: In general, a page i can be considered as follows.
The more important, the more inlinks it has.

- However, a ranking system based ***only*** *on the number of inlinks* is easy to manipulate.
  - When you design a web page i that you would like to be seen by as many as possible, you could simply create a large number of (information-less and unimportant) pages that have outlinks to i.
- In order to discourage this, one may define **the rank of i** in such a way that if **a highly ranked page j**, has an outlink to i, this should add to the importance of i.
- Here the manner is:

  the rank of page i is a weighted sum of the ranks of the pages that have outlinks to i.

Definition 13.16. The preliminary definition of **Pagerank** is

$$r_i = \sum_{j \in I_i} \frac{r_j}{N_j}, \quad i = 1, 2, \cdots, n. \tag{13.13}$$

That is, the weighting is such that the rank of a page j is **divided evenly** among its outlinks.

Remark 13.17. Pagerank may not be solvable.

- As in (13.3)-(13.4), p. 485, let Q be a **link matrix**, a square matrix of dimension n, and let

$$Q_{ij} = \begin{cases} 1/N_j, & \text{if there is a link from } j \text{ to } i, \\ 0, & \text{otherwise,} \end{cases} \tag{13.14}$$

  where Q is sometimes called the **normalized web matrix**.
- Then, (13.13) can be written as

$$\lambda \mathbf{r} = Q\mathbf{r}, \quad \lambda = 1, \tag{13.15}$$

  i.e., r is an eigenvector of Q with eigenvalue $\lambda = 1$.
- However, it is not clear that Pagerank is well-defined, because we do not know if there exists an eigenvalue of Q equal to 1.

Reformulation of (13.13)

Modify the matrix Q to have an eigenvalue $\lambda = 1$.

- Assume that a surfer visiting a web page, always chooses the next page among the outlinks with equal probability.
- Assume that the **random surfers never get stuck.**
  - In other words, there should be no web pages without outlinks (such a page corresponds to a zero column in Q).
- Therefore the model is modified so that **zero columns are replaced by a constant value in each position.**

- Define the vectors

$$\mathbf{e} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^n, \qquad \mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix}, \; d_j = \begin{cases} 1, & \text{if } N_j = 0, \\ 0, & \text{otherwise.} \end{cases} \tag{13.16}$$

- Then the modified matrix is defined

$$P = Q + \frac{1}{n}\mathbf{e}\mathbf{d}^T. \tag{13.17}$$

- Then P is a **column-stochastic matrix**, of which columns are probability vectors. That is, it has non-negative elements ($P \geq 0$) and the sum of each column is 1. See Section 8.7 for details of stochastic matrices.
- Furthermore,

$$\mathbf{e}^T P = \mathbf{e}^T Q + \frac{1}{n}\mathbf{e}^T\mathbf{e}\mathbf{d}^T = \mathbf{e}^T Q + d^T = \mathbf{e}^T, \tag{13.18}$$

  which implies that $\lambda = 1$ is a **left eigenvalue** and therefore a **right eigenvalue**. Note that

$$\begin{aligned} &\mathbf{e}^T P = \mathbf{e}^T \Longleftrightarrow P^T\mathbf{e} = \mathbf{e}, \\ &\textit{det}\,(A - \lambda I) = \textit{det}\,(A^T - \lambda I). \end{aligned} \tag{13.19}$$

- Now, we define the **Pagerank vector** $\mathbf{r}$ as *a unique eigenvector of* P *with eigenvalue* $\lambda = 1$*,*

$$P\mathbf{r} = \mathbf{r}. \tag{13.20}$$

- However, uniqueness is still not guaranteed.
  - To ensure this, the directed graph corresponding to the matrix must be **strongly connected**
  - Equivalently, in matrix terms, P must be **irreducible**.
  - Equivalently, there must not exist any subgraph, which has no outlinks.

The uniqueness of the eigenvalue is guaranteed by the **Perron-Frobenius theory**.

Theorem 13.18. (Perron-Frobenius) *If $A \in \mathbb{R}^{n\times n}$ is nonnegative, then*

- *$\rho(A)$ is an eigenvalue of A.*
- *There is a nonnegative eigenvector $\mathbf{x}$ such that $A\mathbf{x} = \rho(A)\mathbf{x}$.*

Theorem 13.19. (Perron-Frobenius) *If $A \in \mathbb{R}^{n\times n}$ is nonnegative and irreducible, then*

- *$\rho(A)$ is an eigenvalue of A.*
- *$\rho(A) > 0$.*
- *There is a positive eigenvector $\mathbf{x}$ such that $A\mathbf{x} = \rho(A)\mathbf{x}$.*
- *$\rho(A)$ is a simple eigenvalue.*

Theorem 13.20. (Perron) *If $A \in \mathbb{R}^{n\times n}$ is positive, then*

- *Theorem 13.19 holds, and in addition,*
- *$|\lambda| < \rho(A)$ for any eigenvalue λ with $\lambda \neq \rho(A)$.*

Summary 13.21. Let A be an irreducible column-stochastic matrix. Then

- The largest eigenvalue in magnitude is equal to 1.
- There is a unique corresponding eigenvector $\mathbf{r}$ satisfying $\mathbf{r} > 0$ and $||\mathbf{r}||_1 = 1$; this is the only eigenvector that is non-negative.
- If $A > 0$, then $|\lambda_i| < 1$, $i = 2, 3, \cdots, n$.

Remark 13.22. Given the size of the Internet and reasonable assumptions about its structure,

- it is highly probable that the **link graph** is not strongly connected,
- which means that **the Pagerank eigenvector of P may not be well-dened.**

13.4.2. The Google Matrix

To ensure connectedness, i.e., to make it impossible for the random walker to get trapped in a subgraph, **one can add, articially, a link from every web page to all the other**. In matrix terms, this can be made by taking a convex combination of P and a rank one matrix.

One billion dollar idea, by **Sergey Brin** and **Lawrence Page** in 1996

- The **Google matrix** is the matrix

$$G = \alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T, \tag{13.21}$$

  for some α satisfying $0 < \alpha < 1$, called the **damping factor**.
- Obviously G is irreducible (since $G > 0$) and column-stochastic.[a]
- Furthermore,

$$\mathbf{e}^T G = \alpha \mathbf{e}^T P + (1-\alpha)\mathbf{e}^T \frac{1}{n}\mathbf{e}\mathbf{e}^T = \alpha \mathbf{e}^T + (1-\alpha)\mathbf{e}^T = \mathbf{e}^T. \tag{13.22}$$

- The **pagerank equation** reads

$$G\mathbf{r} = \left[\alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\right]\mathbf{r} = \mathbf{r}. \tag{13.23}$$

[a] A $n \times n$ matrix is called a **Markov matrix** if all entries are nonnegative and the sum of each column vector is equal to 1. A Markov matrix are also called a **stochastic matrix**.

Note: The random walk interpretation of the additional rank one term is that each time step a page is visited, the surfer will jump to any page in the whole web with probability $(1-\alpha)/n$ (sometimes referred to as **teleportation**).

- Recall (13.17): $P = Q + \frac{1}{n}\mathbf{e}\mathbf{d}^T$, which can be interpreted as follows.

  When a random surfer visits a web page of no outlinks, the surfer will jump to any page with an equal probability $1/n$.

- The convex combination in (13.21): $G = \alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T$.

  Although there are outlinks, the surfer will jump to any page with an equal probability $(1-\alpha)/n$.

Proposition 13.23. Let the eigenvalues of the column-stochastic matrix P be $\{1, \lambda_2, \lambda_3, \cdots, \lambda_n\}$.
Then, the eigenvalues of G are $\{1, \alpha\lambda_2, \alpha\lambda_3, \cdots, \alpha\lambda_n\}$.

- This means that even if P has a multiple eigenvalue equal to 1, the second largest eigenvalue of G is equal to α.

Remark 13.24. The vector $\frac{1}{n}\mathbf{e}$ in (13.21) can be replaced by a non-negative vector $\mathbf{v}$ with $||\mathbf{v}||_1 = 1$, which can be chosen in order to make the search biased towards certain kinds of web pages. Therefore, it is referred to as a **personalization vector**.

Solution of the Pagerank Equation

Now, we should solve the Pagerank equation, an eigenvalue problem

$$G\mathbf{r} = \mathbf{r}, \tag{13.24}$$

where $\mathbf{r} \geq 0$ with $||\mathbf{r}||_1 = 1$.

Observation 13.25. The Google matrix $G \in \mathbb{R}^{n \times n}$

- G is a full matrix, although it is not necessary to construct it explicitly.
- n represents the number of all web pages, which is order of billions.
- It is **impossible** to use **sparse eigenvalue algorithms** that require the storage of more than very few vectors.

The only viable method so far for Pagerank computations on the whole web seems to be the **power method**. See Algorithm 8.22, p.288.

- The rate of convergence of the power method depends on the ratio of the second largest and the largest eigenvalue in magnitude.
- Here, we have
$$|1 - \lambda^{(k)}| = \mathcal{O}(\alpha^k), \tag{13.25}$$
due to Proposition 13.23.
- In view of the huge dimension of the Google matrix, it is non-trivial to compute the matrix-vector product. We will consider some details.

The power method: matrix-vector product

Recall: It follows from (13.14), (13.17), and (13.21) that the **Google matrix** is formulated as

$$G = \alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T, \tag{13.26}$$

where

$$P = Q + \frac{1}{n}\mathbf{e}\mathbf{d}^T.$$

Here Q is the **link matrix** and $\mathbf{e}$ and $\mathbf{d}$ are defined as in (13.16).

Derivation 13.26. Let $\mathbf{z} = G\mathbf{y}$.

- **Normalization-free**: Since G is column-stochastic ($\mathbf{e}^T G = \mathbf{e}^T$),

$$||\mathbf{z}||_1 = \mathbf{e}^T\mathbf{z} = \mathbf{e}^T G\mathbf{y} = \mathbf{e}^T\mathbf{y} = ||\mathbf{y}||_1. \tag{13.27}$$

  Thus, when the power method begins with $\mathbf{y}^{(0)}$ with $||\mathbf{y}^{(0)}||_1 = 1$, the normalization step in the power method is unnecessary.

- Let us look at the multiplication in some detail:

$$\mathbf{z} = \left[\alpha P + (1-\alpha)\frac{1}{n}\mathbf{e}\mathbf{e}^T\right]\mathbf{y} = \alpha Q\mathbf{y} + \beta\frac{\mathbf{e}}{n}, \tag{13.28}$$

  where

$$\beta = \alpha\mathbf{d}^T\mathbf{y} + (1-\alpha)\mathbf{e}^T\mathbf{y}. \tag{13.29}$$

- Apparently we need to know which pages lack outlinks ($\mathbf{d}$), in order to find β. However, in reality, we do not need to define $\mathbf{d}$. It follows from (13.27) and (13.28), multiplying (13.28) by $\mathbf{e}^T$, that

$$\beta = 1 - \alpha\mathbf{e}^T Q\mathbf{y} = 1 - ||\alpha Q\mathbf{y}||_1. \tag{13.30}$$

Algorithm 13.27. The following Matlab code implements the matrix vector multiplication: $\mathrm{z} = G\mathrm{y}$.

```
zhat = alpha*Q*y;
beta = 1-norm(zhat,1);
z = zhat + beta*v;
residual = norm(y-z,1);
```

Here v is $(1/n)$e or a **personalization vector**; see Remark 13.24.

Note:

- From Proposition 13.23, we know that the second eigenvalue of the Google matrix is $\alpha\lambda_2$.
- A typical value of $\alpha = 0.85$.
- Approximately $k = 57$ iterations are needed to reach $0.85^k < 10^{-4}$.
- This is reported to be close the number of iterations used by Google.

The 15 Percent Rule - Google PageRank

YouTube (Rap) (Acoustic Pop)

Exercises for Chapter 13

13.1. Consider Example 13.13, p.496. Compute vectors of cosines, for each subspace approximations, i.e., with A_k where $k = 1, 2, \cdots, 5$.

13.2. Verify equations in Derivation 13.26, p.505, particularly (13.28), (13.29), and (13.30).

13.3. Consider the link matrix Q in (13.4) and its corresponding link graph in Figure 13.2. Find the pagerank vector $\mathbf{r}$ by solving the Google pagerank equation.

- You may initialize the power method with any vector $\mathbf{r}^{(0)}$ satisfying $||\mathbf{r}^{(0)}||_1 = 1$.
- Set $\alpha = 0.85$.
- Let the iteration stop, when $\texttt{residual} < 10^{-4}$.

13.4. Now, consider a **modified** link matrix $\widetilde{Q}$, by adding an outlink from page ④ to ⑤ in Figure 13.2. Find the pagerank vector $\widetilde{\mathbf{r}}$, by setting parameters and initialization the same way as for the previous problem.

- Compare $\mathbf{r}$ with $\widetilde{\mathbf{r}}$.
- Compare the number of iterations for convergence.

CHAPTER 14

Finite Difference Methods for Boundary-Value Problems

In This Chapter:

Contents of Chapter 14

14.1. The Shooting Method 510
14.2. Finite Difference Methods (FDMs) 517
14.3. FDMs for Non-constant Diffusion 527
Exercises for Chapter 14 532

14.1. The Shooting Method

Problem 14.1. The two-point **Boundary-Value Problems** (BVPs) involve a second-order differential equation of the form

$$\begin{aligned} y'' &= f(x, y, y'), \quad x \in [a, b] \\ y(a) &= \alpha, \quad y(b) = \beta \end{aligned} \tag{14.1}$$

Theorem 14.2. (Existence and Uniqueness)
Let

$$D = \{(x, y, y') \mid a \le x \le b, \ \textit{with} \ -\infty < y < \infty, \ -\infty < y' < \infty\}.$$

For the BVP of the form in (14.1), if the functions f, f_y, $f_{y'}$ are continuous on D and satisfy

(i) $f_y(x, y, y') > 0$, *for all* $(x, y, y') \in D$

(ii) there is a constant M such that

$$|f_{y'}(x, y, y')| \le M, \quad \textit{for all} \ (x, y, y') \in D$$

then the BVP has a unique solution.

Example 14.3. Show that the following BVP has a unique solution.

$$\begin{aligned} y'' + e^{-xy} + sin(y') &= 0, \quad 1 \le x \le 2 \\ y(1) = 0, \ \ y(2) &= 0 \end{aligned}$$

Solution.

Linear Boundary-Value Problems

Definition 14.4. The differential equation $y'' = f(x, y, y')$ is **linear** if f can be expressed as a linear combination of y and y', that is,

$$y'' = f(x, y, y') = p(x)\, y' + q(x)\, y + r(x), \tag{14.2}$$

for some p, q, r.

Corollary 14.5. *Suppose the linear BVP*

$$\begin{gathered} y'' = f(x, y, y') = p(x)\, y' + q(x)\, y + r(x), \quad a \le a \le b \\ y(a) = \alpha, \quad y(b) = \beta \end{gathered} \tag{14.3}$$

satisfies

(i) $p(x)$, $q(x)$, $r(x)$ *are continuous on* $[a, b]$

(ii) $q(x) > 0$ *on* $[a, b]$

then the linear BVP has a unique solution.

The Linear Shooting Method

1. To approximate the unique solution to the linear BVP (14.3), we first solve the problem twice, with two different *initial conditions*, obtaining solutions y_1 and y_2, say,

$$\begin{cases} y_1(a) = \alpha, \;\; y_1'(a) = z_1 \\ y_2(a) = \alpha, \;\; y_2'(a) = z_2 \end{cases} \tag{14.4}$$

   (We will choose z_1 and z_2 appropriately later.)

2. Then we form a linear combination of y_1 and y_2:

$$y(x) \;=\; \lambda y_1(x) + (1 - \lambda) y_2(x), \tag{14.5}$$

   where λ is a parameter. We can easily verify that $y(a) = \alpha$.

3. Thus the remained requirement is to satisfy $y(b) = \beta$. We select λ such that

$$y(b) \;=\; \lambda y_1(b) + (1 - \lambda) y_2(b) \;=\; \beta, \tag{14.6}$$

which reads

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)}, \tag{14.7}$$

provided that $y_1(b) \neq y_2(b)$.

4. Note that z_1 and z_2 can be chosen arbitrarily, $z_1 \neq z_2$; however, the most common choice is $z_1 = 0$ and $z_2 = 1$.

Claim 14.6. If the linear BVP (14.3) has a solution , then either

$$\begin{cases} y \equiv y_1, & \text{or} \\ y_1(b) \neq y_2(b). \end{cases} \tag{14.8}$$

For the later case, the solution is a linear combination (14.5) with (14.7).

Summary 14.7. Summary of the Linear Shooting Method

1. Solve **two IVPs** for y_1 and y_2:

$$\begin{cases} y_1'' = f(x, y_1, y_1') \\ y_1(a) = \alpha, \;\; y_1'(a) = 0 \end{cases} \qquad \begin{cases} y_2'' = f(x, y_2, y_2') \\ y_2(a) = \alpha, \;\; y_2'(a) = 1 \end{cases} \tag{14.9}$$

   where $f(x, y, y') = p(x)\, y' + q(x)\, y + r(x)$.

2. Compute

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)}, \tag{14.10}$$

3. Get the solution of the linear BVP (14.3) as a linear combination of y_1 and y_2:

$$y(x) = \lambda y_1(x) + (1 - \lambda) y_2(x). \tag{14.11}$$

Remark 14.8. Note that

$$y'' = f(x, y, y') = p(x)\, y' + q(x)\, y + r(x).$$

Each of the problems in the first step can be solved by applying an ODE solver for systems. For example, the second problem equivalently reads as a system of differential equations:

$$\begin{cases} y' = u, & y(a) = \alpha \\ u' = p\, u + q\, y + r, & u(a) = 1 \end{cases} \tag{14.12}$$

which can be solved efficiently by using an ODE solver. See **Section 5.2, p. 183, and Section 5.3 p. 192.**

Here we state one of the most popular ODE solvers, the **Fourth-order Runge-Kutta method (RK4)**.

Algorithm 14.9. (RK4: Algorithm 5.19, p. 195)

$$y_{n+1} = y_n + \frac{h}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) \tag{14.13}$$

where

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_1) \\ K_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hK_2) \\ K_4 &= f(x_n + h, y_n + hK_3) \end{aligned}$$

Note: The **local truncation error** for the above RK4 can be derived as

$$\frac{h^5}{5!} y^{(5)}(\xi_n), \tag{14.14}$$

for some $\xi_n \in [x_n, x_{n+1}]$. Thus the **global error** reads, for some $\xi \in [x_0, T]$,

$$\frac{(T - x_0)h^4}{5!} y^{(5)}(\xi). \tag{14.15}$$

RK4SYSTEM.mw

```
RK4SYSTEM := proc(a,b,nt,X,F,x0,xn)
    local h,hh,t,m,n,j,w,K1,K2,K3,K4;
    #### initial setting
    with(LinearAlgebra):
    m := Dimension(Vector(F));
    w  :=Vector(m);
    K1:=Vector(m); K2:=Vector(m);
    K3:=Vector(m); K4:=Vector(m);
    h:=(b-a)/nt;  hh:=h/2;
    t :=a; w:=x0;
    for j from 1 by 1 to m do
        xn[0,j]:=x0[j];
    end do;
    #### RK4 marching
    for n from 1 by 1 to nt do
        K1:=Vector(eval(F,[x=t,seq(X[i+1]=xn[n-1,i],i=1..m)]));
        K2:=Vector(eval(F,[x=t+hh,seq(X[i+1]=xn[n-1,i]+hh*K1[i],i=1..m)]));
        K3:=Vector(eval(F,[x=t+hh,seq(X[i+1]=xn[n-1,i]+hh*K2[i],i=1..m)]));
        t:=t+h;
        K4:=Vector(eval(F,[x=t,seq(X[i+1]=xn[n-1,i]+h*K3[i], i=1..m)]));
        w:=w+(h/6.)*(K1+2*K2+2*K3+K4);
        for j from 1 by 1 to m do
            xn[n,j]:=evalf(w[j]);
        end do
    end do
end proc:
```

Example 14.10. Use the RK4SYSTEM to solve

$$\begin{cases} y'' - 2y' + 2y \;=\; e^{2x}\sin(x), \quad 0 \le x \le 1 \\ y(0) = -0.4, \quad y'(0) = -0.6 \end{cases} \tag{14.16}$$

Solution. The IVP (14.16) reads

$$\begin{cases} y' = u, & y(0) = -0.4 \\ u' = 2u - 2y + e^{2x}\sin(x), & u(0) = -0.6 \end{cases} \tag{14.17}$$

Run

```
m := 2:
F := [yp, exp(2*x)*sin(x) -2*y +2*yp]:
X := [x, y, yp]:
X0 := <-0.4, -0.6>:
a := 0:  b := 1:
nt := 10:
Xn := Array(0 .. nt, 1 .. m):
RK4SYSTEM(a, b, nt, X, F, X0, Xn):

DE := diff(y(x),x,x) - 2*diff(y(x),x) + 2*y(x) = exp(2*x)*sin(x);
IC := y(0) = -0.4, D(y)(0) = -0.6;

dsolve({DE, IC}, y(x));
                      1
             y(x) = - (sin(x) - 2 cos(x)) exp(2 x)
                      5
```

Result

n	y_n	y(x_n)	y'_n	y'(x_n)	err(y)	err(y')
0	-0.40000	-0.40000	-0.60000	-0.60000	0	0
1	-0.46173	-0.46173	-0.63163	-0.63163	3.72e-07	1.91e-07
2	-0.52556	-0.52556	-0.64015	-0.64015	8.36e-07	2.84e-07
3	-0.58860	-0.58860	-0.61366	-0.61366	1.39e-06	1.99e-07
4	-0.64661	-0.64661	-0.53658	-0.53658	2.02e-06	1.68e-07
5	-0.69357	-0.69356	-0.38874	-0.38874	2.71e-06	9.58e-07
6	-0.72115	-0.72115	-0.14438	-0.14438	3.41e-06	2.35e-06
7	-0.71815	-0.71815	0.22900	0.22899	4.06e-06	4.59e-06
8	-0.66971	-0.66971	0.77199	0.77198	4.55e-06	7.97e-06
9	-0.55644	-0.55644	1.53478	1.53477	4.77e-06	1.29e-05
10	-0.35340	-0.35339	2.57877	2.57875	4.50e-06	1.97e-05

Example 14.11. Use the shooting method to approximate the solution of

$$\begin{cases} y'' = \dfrac{2}{x}y' + \dfrac{2}{x^2}y - 3x^2, \quad 1 \le x \le 2, \\ y(1) = 0, \quad y(2) = 2. \end{cases}$$

Solution.

Maple Code

```
m := 2:
F := [yp, -2*yp/x + 2*y/x^2 - 3*x^2]:
X := [x, y, yp]:
a := 1:  b := 2:
al := 0:  be := 2:
N := 5:
Y := Array(0..2*N, 1..2):
for k to 2 do
    nt := 2^(k-1)*N;
    Y1 := Array(0..nt, 1..m);
    Y2 := Array(0..nt, 1..m);
    X0 := <al, 0>; RK4SYSTEM(a, b, nt, X, F, X0, Y1);
    X0 := <al, 1>; RK4SYSTEM(a, b, nt, X, F, X0, Y2);
    lam := (be - Y2[nt, 1])/(Y1[nt, 1] - Y2[nt, 1]);
    for i from 0 to nt do
        Y[i, k] := lam*Y1[i, 1] + (1 - lam)*Y2[i, 1];
    end do:
end do:

## The exact solution & Error analysis
DE := diff(y(x), x, x) = -2*diff(y(x), x)/x + 2*y(x)/x^2 - 3*x^2;
BC := y(1) = 0, y(2) = 2;
dsolve({BC, DE}, y(x));
                              52     1  4   37
                   y(x) = - ----- - - x  + -- x
                               2     6      14
                             21 x
```

Result

	h=1/5	error	h=1/10	error
x=1.00	0.0000000000	0	0.0000000000	0
x=1.20	1.1056335076	0.000618	1.1062189010	3.3e-005
x=1.40	1.7958326514	0.000538	1.7963419934	2.89e-005
x=1.60	2.1686944799	0.000348	2.1690241106	1.87e-005
x=1.80	2.2431240407	0.000162	2.2432777718	8.77e-006
x=2.00	2.0000000000	0	2.0000000000	0

14.2. Finite Difference Methods (FDMs)

Recall: A linear BVP (14.3) has the form

$$\begin{aligned} y'' &= p(x)\,y' + q(x)\,y + r(x), \quad a \le x \le b, \\ y(a) &= \alpha, \quad y(b) = \beta. \end{aligned} \tag{14.18}$$

The last section introduced the shooting method for its numerical solution.

In order to develop **finite difference methods (FDM)** for the solution of the problem in **more general environments**, in this section, we consider differential equations of the following form

$$\begin{cases} -y'' + p(x)\,y' + q(x)\,y = f(x), \quad a \le x \le b, \\ y(a) = \alpha, \quad y'(b) = \beta. \end{cases} \tag{14.19}$$

Recall: **(Theorem 1.22: Taylor's Theorem with Lagrange Remainder)**: Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b), and $x_0 \in [a,b]$. Then, for every $x \in [a,b]$,

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \mathcal{R}_n(x), \quad \mathcal{R}_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}, \tag{14.20}$$

for some ξ between x and x_0.

Recall: **(Remark 1.25: Alternative Form of Taylor's Theorem)**: Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b). Then, for every x, $x+h \in [a,b]$,

$$f(x+h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!}h^k + \mathcal{R}_n(h), \quad \mathcal{R}_n(h) = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1}, \tag{14.21}$$

for some ξ between x and $x+h$. In detail,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \cdots + \frac{f^{(n)}(x)}{n!}h^n + \mathcal{R}_n(h). \tag{14.22}$$

Scheme **14.12.** *The* **FD discretization procedure** *consists of four steps:*

1. **Partitioning (Split the zone!)**
2. **FD approximations (Unknowns get shown!)**
3. **System of difference equations (Equations arise!)**
4. **Assembly (An algebraic system flies!)**

14.2.1. The FD discretization procedure

Step 1: Partitioning

Let, for some $n > 0$,

$$x_i = a + ih, \quad h = \frac{b-a}{n}, \quad i = 0, 1, \cdots, n. \tag{14.23}$$

Define $g_i = g(x_i)$, for $g = y, p, q, f$; for example,

$$y_i = y(x_i). \tag{14.24}$$

The objective with finite difference methods is to approximate $\{y_i\}_{i=0}^{n}$, the solution at **discrete nodal points**.

(Split the zone!)

Step 2: FD approximations

- Expanding y in the **Taylor series** about $x = x_i$ evaluated at $x_{i+1} = x_i + h$ and $x_{i-1} = x_i - h$, we have

$$\begin{aligned}
\text{(a)}\quad y_{i+1} &= y(x_i) + y'(x_i)h + \frac{y''(x_i)}{2!}h^2 + \frac{y'''(x_i)}{3!}h^3 + \frac{y^{(4)}(x_i)}{4!}h^4 \\
&\quad + \frac{y^{(5)}(x_i)}{5!}h^5 + \frac{y^{(6)}(x_i)}{6!}h^6 + \cdots \\
\text{(b)}\quad y_{i-1} &= y(x_i) - y'(x_i)h + \frac{y''(x_i)}{2!}h^2 - \frac{y'''(x_i)}{3!}h^3 + \frac{y^{(4)}(x_i)}{4!}h^4 \\
&\quad - \frac{y^{(5)}(x_i)}{5!}h^5 + \frac{y^{(6)}(x_i)}{6!}h^6 - \cdots
\end{aligned} \tag{14.25}$$

- **Approximation of $y''(x_i)$**: Adding (14.25)(a) and (14.25)(b) gives

$$y_{i+1} + y_{i-1} = 2y_i + 2\frac{y''(x_i)}{2!}h^2 + 2\frac{y^{(4)}(x_i)}{4!}h^4 + 2\frac{y^{(6)}(x_i)}{6!}h^6 + \cdots \tag{14.26}$$

- Solving for $y''(x_i)$ reads

$$y''(x_i) = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} - 2\frac{y^{(4)}(x_i)}{4!}h^2 - 2\frac{y^{(6)}(x_i)}{6!}h^4 - \cdots \tag{14.27}$$

- **Approximation of $y'(x_i)$**: Subtracting (14.25)(b) from (14.25)(a) gives

$$y_{i+1} - y_{i-1} = 2y'(x_i)h + 2\frac{y'''(x_i)}{3!}h^3 + 2\frac{y^{(5)}(x_i)}{5!}h^5 + \cdots \tag{14.28}$$

- Solving for $y'(x_i)$ reads

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} - \frac{y'''(x_i)}{3!}h^2 - \frac{y^{(5)}(x_i)}{5!}h^4 - \cdots \tag{14.29}$$

(Unknowns get shown!)

Step 3: System of Difference Equations

- **Differential Equation**: Using these central FD schemes for the DE

$$-y'' + py' + qy = f,$$

at x_i results in

$$-\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + p_i \cdot \frac{y_{i+1} - y_{i-1}}{2h} + q_i \cdot y_i + K_2 h^2 + \mathcal{O}(h^4) = f_i, \quad (14.30)$$

where

$$K_2 = \frac{1}{12}\big(y^{(4)}(x_i) - 2p_i y'''(x_i)\big).$$

- Note that $y_0 = y(a) = \alpha$; (14.30) is defined only for $i = 1, 2, \cdots, n$.
- When $i = n$, (14.30) reads

$$\frac{-y_{n-1} + 2y_n - y_{n+1}}{h^2} + p_n \cdot \frac{y_{n+1} - y_{n-1}}{2h} + q_n \cdot y_n + K_2 h^2 + \mathcal{O}(h^4) = f_n, \quad (14.31)$$

> **Note**: The unknowns are $\{y_1, y_2, \cdots, y_n\}$; the **ghost grid value** y_{n+1} must be eliminated appropriately and accurately.

- **Boundary Condition**: The boundary condition $y'(b) = \beta$ can be written as follows. Since $n = x_n$,

$$\frac{y_{n+1} - y_{n-1}}{2h} - \frac{y'''(b)}{3!}h^2 + \mathcal{O}(h^4) = \beta. \quad (14.32)$$

> The **second-order FDM** for the model (14.19) follows from (14.30), (14.31), and (14.32), truncating the terms involving h^2 and higher-orders of h.

- Let v_i denote the second-order approximation of y_i. Then, the **second-order FDM** for (14.19) reads

$$\frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2} + p_i \cdot \frac{v_{i+1} - v_{i-1}}{2h} + q_i \cdot v_i = f_i, \quad i = 1, 2, \cdots, n, \quad (14.33)$$

where $v_0 = \alpha$ and $v_{n+1} = v_{n-1} + 2h\,\beta$.

(Equations arise!)

Step 4: Assembly

- **Assembly** is to build an algebraic system from (14.33).
- Multiplying (14.33) with h^2 and simplifying the result reads

$$-c_i v_{i-1} + (2 + h^2 q_i) v_i - d_i v_{i+1} = h^2 f_i, \quad i = 1, 2, \cdots, n, \tag{14.34}$$

where

$$c_i = 1 + \frac{hp_i}{2} \quad \text{and} \quad d_i = 1 - \frac{hp_i}{2}. \tag{14.35}$$

- Using $v_0 = \alpha$ and $v_{n+1} = v_{n-1} + 2h\,\beta$, obtained from the boundary conditions, we present the details of algebraic equations as follows.

$$\begin{aligned} (2 + h^2 q_1) v_1 - d_1 v_2 &= h^2 f_1 + c_1 \cdot \alpha, & i &= 1 \\ -c_i v_{i-1} + (2 + h^2 q_i) v_i - d_i v_{i+1} &= h^2 f_i, & i &= 2, \cdots, n-1 \\ -2v_{n-1} + (2 + h^2 q_n) v_n &= h^2 f_n + d_n \cdot 2h\beta, & i &= n \end{aligned} \tag{14.36}$$

- The corresponding **algebraic system** reads

$$A\mathbf{v} = \mathbf{b}, \tag{14.37}$$

where

$$A = \begin{bmatrix} 2 + h^2 q_1 & -d_1 & & & \\ -c_2 & 2 + h^2 q_2 & -d_2 & & \\ & \ddots & \ddots & \ddots & \\ & & -c_{n-1} & 2 + h^2 q_{n-1} & -d_{n-1} \\ & & & -2 & 2 + h^2 q_n \end{bmatrix},$$

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n \end{bmatrix} + \begin{bmatrix} c_1 \cdot \alpha \\ 0 \\ \vdots \\ 0 \\ d_n \cdot 2h\beta \end{bmatrix}. \tag{14.38}$$

(An algebraic system flies!)

14.2.2. Existence and Uniqueness of the FDM Solution

Theorem 14.13. *Suppose that p, q and f are continuous on $[a, b]$. If*

$$0 \le q(x), \quad \forall\, x \in [a, b], \tag{14.39}$$

then the linear system (14.37) has unique solution provided that

$$c_i, \;\; d_i > 0, \quad i = 1, 2, \cdots, n. \tag{14.40}$$

Proof. When the two conditions are satisfied, the matrix A becomes **diagonally dominant**, i.e.,

$$a_{ii} \ge \sum_{j \ne i} |a_{ij}| =: \Lambda_i, \quad \forall\, i = 1, 2, \cdots, n. \tag{14.41}$$

Indeed,

$$\begin{aligned} a_{11} &= 2 + h^2 q_1 \ge 2 > d_1 = \Lambda_1, \\ a_{ii} &= 2 + h^2 q_i \ge 2 = \Lambda_i, \;\; i = 2, \cdots, n. \end{aligned} \tag{14.42}$$

Since A is clearly irreducible, it is **irreducibly diagonally dominant**. Thus, A is nonsingular and therefore the linear system has a unique solution. Because all the diagonal entries of are in addition positive real, real parts of eigenvalues of are all positive. □

Remark 14.14. Let $L = \max\limits_{x \in [a,b]} |p(x)|$. Then the conditions in (14.40) are equivalent to

$$\frac{1}{2} hL < 1 \iff h < \frac{2}{L}. \tag{14.43}$$

That is, h must be chosen **small enough** to satisfy the inequality.

Example 14.15. Use the second-order FDM to approximate the solution of

$$\begin{aligned} y'' &= \frac{1}{x}y' + \frac{3}{x^2}y - 4x^2, \quad x \in [1,2] \\ y(1) &= 0, \;\; y'(2) = -2, \end{aligned} \tag{14.44}$$

with $h = 1/5$ and $h = 1/10$.

Note: The differential equation can be written as

$$-y'' + \frac{1}{x}y' + \frac{3}{x^2}y = 4x^2, \quad x \in [1,2]. \tag{14.45}$$

Thus, $q(x) = 3/x^2 > 0$ and $L = \max|p(x)| = \max|1/x| = 1$ and therefore $hL/2 = h/2 < 1$. Its exact solution becomes

$$y(x) = -8/(7 * \mathtt{x}) + (68/35) * \mathtt{x}^3 - (4/5) * \mathtt{x}^4. \tag{14.46}$$

Solution.

get_Ab_pqf.m

```
function [A,b] = get_Ab_pqf(X,p,q,f,BC,IBC)

m = size(X,1);

A=zeros(m,m); b=zeros(m,1);
h=X(2)-X(1); h2 = h^2; hh=h/2;

P = p(X); Q = q(X); F = f(X);
C = 1+hh*P; D = 1-hh*P;

%%--------------------
for i=1:m
    A(i,i) = 2+h2*Q(i);
    b(i)   = h2*F(i);
end

for i=2:m-1,
    A(i,i-1) = -C(i);
    A(i,i+1) = -D(i);
end
```

```

%--- left-end ---------
i=1;
if IBC(1) == 1                  % Dirichlet
    A(i,i) = 1; b(i) = BC(1);
else                            % Neumann
    A(i,i+1) = -2;
    b(i) = b(i) +D(i)*2*h*BC(1);
end

%--- right-end --------
i=m;
if IBC(2) == 1
    A(i,i) = 1; b(i) = BC(2);
else
    A(i,i-1) = -2;
    b(i) = b(i) +D(i)*2*h*BC(2);
end
```

FDM_1D.m

```
p = @(x) 1./x;
q = @(x) 3./x.^2;
f = @(x) 4*x.^2;
y = @(x) -8./(7*x)+(68/35)*x.^3 -(4/5)*x.^4;
al = 0; be = -2;

Int=[1,2]; n0 = 5;

ngrid =5;
Y = cell(ngrid,1); V  = cell(ngrid,1);
W = cell(ngrid-1,1);

%BC =[y(1),y(2)]; IBC=[1,1];    % Dirichlet-Dirichlet
BC =[y(1),be];   IBC=[1,2];    % Dirichlet-Neumann

%%-- Y & V
for i=1:ngrid
```

```
    n = n0*2^(i-1); X =linspace(Int(1),Int(2),n+1)';
    P = p(X); Q = q(X); F = f(X);
    Y{i} = y(X);
    [A,b] = get_Ab_pqf(X,p,q,f,BC,IBC);
    V{i}= A\b;
    fprintf(' n = %2d; L8-Error = %.8f',n,norm(Y{i}-V{i},inf))
    if i==1,
        fprintf('\n');
    else
        W{i-1} = (4*V{i}(1:2:end)-V{i-1})/3;
        fprintf('; Richardson = %.4g\n',norm(Y{i-1}-W{i-1},inf))
    end
end
```

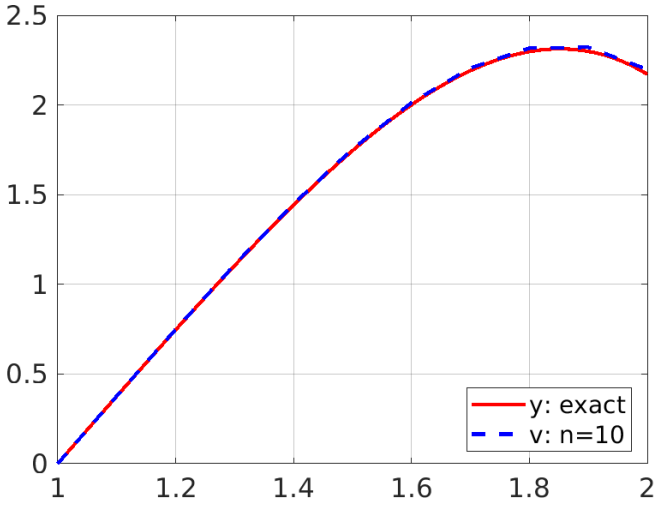
Result

```
 n =  5; L8-Error = 0.10740412
 n = 10; L8-Error = 0.02688154; Richardson = 4.068e-05
 n = 20; L8-Error = 0.00672220; Richardson = 2.413e-06
 n = 40; L8-Error = 0.00168066; Richardson = 1.486e-07
 n = 80; L8-Error = 0.00042017; Richardson = 9.248e-09
```

Remark 14.16. The **Richardson extrapolation** is a numerical procedure that produces a numerical solution of a **higher-order accuracy**, when two numerical solutions are available from a mesh and its refined one. See § 4.2, p. 145, for details.

FDM: Details

```
      exact_y(x) uh(h=1/5) error   uh(h=1/10) error  Richardson error
x=1.20 0.74600 | 0.75688 0.01088 | 0.74871 0.00271 | 0.74598 1.423e-05
x=1.40 1.44159 | 1.46883 0.02723 | 1.44840 0.00680 | 1.44159 6.646e-06
x=1.60 2.00078 | 2.04943 0.04865 | 2.01295 0.01217 | 2.00078 7.089e-06
x=1.80 2.29774 | 2.37302 0.07527 | 2.31658 0.01884 | 2.29777 2.315e-05
x=2.00 2.17143 | 2.27883 0.10740 | 2.19831 0.02688 | 2.17147 4.068e-05
```


2.5
2
1.5
1
0.5
0
1
1.2
1.4
1.6
1.8
2
y: exact
v: n=10

14.3. FDMs for Non-constant Diffusion

In this section, we consider differential equations of the following form

$$\begin{aligned} -(p(x)\,y_x)_x + q(x)y &= f(x), \quad x \in [a,b], \\ y(a) = \alpha, &\quad y'(b) = \beta, \end{aligned} \tag{14.47}$$

where $q(x) \geq 0$.

Step 1: Partitioning

As in §14.2.1, p.518, we start with partitioning of the interval: For some $n > 0$,

$$h = \frac{b-a}{n}, \quad x_i = a + i \cdot h, \;\; i = 0, 1, \cdots, n.$$

Define $g_i = g(x_i)$, for $g = y, p, q, f$.

(Split the zone!)

Step 2: FD approximations: The Central FD Scheme for $(py_x)_x$

- The term (py_x) can be viewed as a function and approximated as

$$(py_x)_x(x_i) \approx \frac{(py_x)_{i+1/2} - (py_x)_{i-1/2}}{h} + \mathcal{O}(h^2), \tag{14.48}$$

  where $(py_x)_{i+1/2}$ is the value of (py_x) evaluated at $x_{i+1/2} := (x_i + x_{i+1})/2$.
- The terms $(py_x)_{i+1/2}$ and $(py_x)_{i-1/2}$ can be again approximated as

$$\begin{aligned} (py_x)_{i+1/2} &\approx p_{i+1/2}\frac{y_{i+1}-y_i}{h} - p_{i+1/2}\frac{y_{xxx}(x_{i+1/2})}{3!}\left(\frac{h}{2}\right)^2 + \cdots, \\ (py_x)_{i-1/2} &\approx p_{i-1/2}\frac{y_i-y_{i-1}}{h} - p_{i-1/2}\frac{y_{xxx}(x_{i-1/2})}{3!}\left(\frac{h}{2}\right)^2 + \cdots. \end{aligned} \tag{14.49}$$

- Combine the above two equations to have

$$(py_x)_x(x_i) \approx \frac{p_{i-1/2}y_{i-1} - (p_{i-1/2}+p_{i+1/2})y_i + p_{i+1/2}y_{i+1}}{h^2} + \mathcal{O}(h^2). \tag{14.50}$$

  Overall truncation error becomes $\mathcal{O}(h^2)$. See Exercise 1, p. 532.

Note: The reader should also notice that the quantities $p_{i+1/2}$ evaluated at mid-points are not available in general.

- We may replace it by the arithmetic/harmonic average of p_i and p_{i+1}:

$$p_{i+1/2} \approx \frac{p_i + p_{i+1}}{2} \quad \text{or} \quad \left[\frac{1}{2}\left(\frac{1}{p_i} + \frac{1}{p_{i+1}}\right)\right]^{-1}. \tag{14.51}$$

- The harmonic average is preferred; the resulting system holds a **flux conservation** property.

(Unknowns get shown!)

Step 3: System of Difference Equations

- Use of the central FD scheme for (14.47) at $x = x_i$ results in

$$\frac{-p_{i-1/2}y_{i-1} + (p_{i-1/2} + p_{i+1/2})y_i - p_{i+1/2}y_{i+1}}{h^2} + q_i y_i = f_i + \mathcal{O}(h^2), \tag{14.52}$$

  where $i = 1, 2, \cdots, n$.

- From the boundary condition in (14.47), we have

$$y_0 = \alpha \tag{14.53}$$

  and

$$\frac{y_{n+1} - y_{n-1}}{2h} = \beta + \frac{h^2}{6}y'''(x_n) + \mathcal{O}(h^4). \tag{14.54}$$

  The above can be rewritten as

$$y_{n+1} = y_{n-1} + 2h\beta + \mathcal{O}(h^3). \tag{14.55}$$

(Equations arise!)

Step 4: Assembly

- It follows from (14.52), (14.53), and (14.55) that

$$
\begin{aligned}
&\boldsymbol{i=1}: \\
&\quad \big(p_{1-1/2}+p_{1+1/2}+h^2q_1\big)y_1 - p_{1+1/2}y_2 = h^2f_1 + p_{1-1/2}\alpha \\
&\boldsymbol{2\le i\le n-1}: \\
&\quad -p_{i-1/2}y_{i-1} + \big(p_{i-1/2}+p_{i+1/2}+h^2q_i\big)y_i - p_{i+1/2}y_{i+1} = h^2f_i \\
&\boldsymbol{i=n}: \\
&\quad -\big(p_{n-1/2}+p_{n+1/2}\big)y_{n-1} + \big(p_{n-1/2}+p_{n+1/2}+h^2q_n\big)y_n = h^2f_n + 2h\beta p_{n+1/2}
\end{aligned}
\tag{14.56}
$$

- The above can be expressed in an $n\times n$ tridiagonal algebraic system.

$$
A\mathbf{u}=\mathbf{b}, \tag{14.57}
$$

where, defining $d_i = p_{i-1/2}+p_{i+1/2}$,

$$
A=\begin{bmatrix}
d_1+h^2q_1 & -p_{1+1/2} & 0 & \cdots & 0 \\
-p_{2-1/2} & d_2+h^2q_2 & -p_{2+1/2} & & \vdots \\
0 & \ddots & \ddots & \ddots & 0 \\
\vdots & & -p_{n-1-1/2} & d_{n-1}+h^2q_{n-1} & -p_{n-1+1/2} \\
0 & \cdots & 0 & -d_n & d_n+h^2q_n
\end{bmatrix} \tag{14.58}
$$

and

$$
\mathbf{u}=\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}, \quad
\mathbf{b}=\begin{bmatrix} h^2f_1 \\ h^2f_2 \\ \vdots \\ h^2f_{n-1} \\ h^2f_n \end{bmatrix} + \begin{bmatrix} p_{1-1/2}\alpha \\ 0 \\ \vdots \\ 0 \\ 2h\beta p_{n+1/2} \end{bmatrix}. \tag{14.59}
$$

(An algebraic system flies!)

Example 14.17. (Revisit of Example 14.15 and Example 15.20, p.558) Consider the boundary-value problem

$$\begin{aligned} -y'' + \frac{1}{x}y' + \frac{3}{x^2}y &= 4x^2, \quad x \in [1,2] \\ y(1) = 0, \;\; y'(2) &= -2, \end{aligned} \tag{14.60}$$

Use the second-order FDM to approximate the solution of the problem, with $h = 1/5$ and $h = 1/10$. The exact solution is

$$y(x) = -8/(7 * \mathtt{x}) + (68/35) * \mathtt{x}^3 - (4/5) * \mathtt{x}^4. \tag{14.61}$$

Note: By dividing both sides of the differential equation by x, we have

$$-\frac{1}{x}y'' + \frac{1}{x^2}y' + \frac{3}{x^3}y = 4x.$$

Thus the problem in (14.60) equivalently reads

$$\begin{aligned} -\left(\frac{1}{x}y'\right)' + \frac{3}{x^3}y &= 4x, \quad x \in [1,2] \\ y(1) = 0, \;\; y'(2) &= -2. \end{aligned} \tag{14.62}$$

You will implement a code for solving (14.62), using the FD schemes considered in this section. See Exercise 3.

Solution.

FDM, with (14.60): Section 14.2

```
      exact_y(x) uh(h=1/5) error   uh(h=1/10) error  Richardson error
x=1.20 0.74600 | 0.75688 0.01088 | 0.74871 0.00271 | 0.74598 1.423e-05
x=1.40 1.44159 | 1.46883 0.02723 | 1.44840 0.00680 | 1.44159 6.646e-06
x=1.60 2.00078 | 2.04943 0.04865 | 2.01295 0.01217 | 2.00078 7.089e-06
x=1.80 2.29774 | 2.37302 0.07527 | 2.31658 0.01884 | 2.29777 2.315e-05
x=2.00 2.17143 | 2.27883 0.10740 | 2.19831 0.02688 | 2.17147 4.068e-05
```

FDM, with (14.62)

```
      exact_y(x) u(h=1/5) error    u(h=1/10) error   Richardson error
x=1.20 0.74600 | 0.75386 0.00787 | 0.74795 0.00195 | 0.74597 2.317e-05
x=1.40 1.44159 | 1.46325 0.02166 | 1.44700 0.00540 | 1.44158 1.456e-05
x=1.60 2.00078 | 2.04176 0.04098 | 2.01103 0.01025 | 2.00078 4.150e-06
x=1.80 2.29774 | 2.36390 0.06615 | 2.31430 0.01656 | 2.29777 2.774e-05
x=2.00 2.17143 | 2.26914 0.09772 | 2.19590 0.02447 | 2.17148 5.516e-05
```

Remark 14.18. The differential equation

$$-(py')' + qy = f \tag{14.63}$$

reads equivalently

$$-py'' - p'y' + qy = f. \tag{14.64}$$

However, it is ***occasionally*** the case that the numerical solution of (14.63) is much more accurate than that of (14.64), ***particularly when the diffusion function*** $p(x)$ ***is oscillatory or non-smooth.***

- When an FEM is applied, you should not use DE of the form (14.64) when (14.63) is available.

Exercises for Chapter 14

14.1. Verify that the overall truncation error for the FD scheme (14.50) is second-order in h. *Hint*: Define

$$K(x) = p(x)\frac{y_{xxx}(x)}{3!}\left(\frac{h}{2}\right)^2 + \cdots,$$

for the truncation errors appeared in (14.49). Then the truncation error for the approximation of $(py_x)_{i+1/2} - (py_x)_{i-1/2}$ becomes $K(x_{i+1/2}) - K(x_{i-1/2}) = hK'(x_i) + \cdots$.

14.2. C Consider a BVP

$$y'' = y' + 2y + \cos x, \quad 0 \le x \le \pi/2,$$
$$y(0) = -0.3, \quad y(\pi/2) = -0.1.$$

Its exact solution is $y = -(\sin x + 3\cos x)/10$.

(a) Show that the BVP has a unique solution.
(b) Use the shooting method (with RK4SYSTEM) to approximate the solution, $h = \pi/10$ and $h = \pi/20$.
(c) Measure the errors for the approximate solutions computed in (b) and see if the numerical solutions show the desired accuracy.

14.3. C For numerical solution of the differential equation in (14.62),

(a) Implement a code, using the FD schemes studied in Section 14.3.
(b) Perform an error analysis comparing with results in Example 14.15.

CHAPTER 15

Finite Element Methods for Boundary-Value Problems

You may watch these FEM songs in the beginning of the chapter and at the end as well. Currently they include only melodies with lyrics, which show the core concepts of FEMs. Later on, they will be added by movies.

Finite Element Method - Break It Down with FEM
YouTube (Melodic Rap) (Rock)

Here the goal is:

Music Movie Maps
Capture Core Concepts

Contents of Chapter 15

15.1. Variational Principle: From Strong Form to Weak 534
15.2. Finite Element Methods (FEMs) 537
15.3. Other Variational Principles for FEMs 549
15.4. FEMs for More General Boundary-Value Problems 556
Exercises for Chapter 15 559

15.1. Variational Principle: From Strong Form to Weak

- Consider approximating the solution to a two-point **boundary-value problem** of the form

$$\text{(D)} \quad \begin{cases} -\dfrac{d}{dx}\Big(p(x)\dfrac{du}{dx}\Big) = f(x), \quad x \in [0,1], \\ u(0) = 0, \quad u(1) = 0, \end{cases} \tag{15.1}$$

  for which we assume $p \in C^1[0,1]$ and there is a constant $\delta > 0$ such that

$$p(x) \geq \delta > 0, \quad x \in [0,1]. \tag{15.2}$$

- The problem (D) in (15.1) is called the **differential problem**.
- The **finite element method** (**FEM**) is formulated with the so-called **variational principle**.

Variational Formulation

Define a linear space over the interval $[0,1]$:

$$V = \big\{ v \mid v \in C^0[0,1], v' \text{ is piecewise continuous on } [0,1] \text{ and } v(0) = v(1) = 0 \big\} \tag{15.3}$$

- Multiply both sides of (15.1) by a **test function** $v \in V$:

$$-\big(pu'\big)'v = fv, \quad v \in V. \tag{15.4}$$

- From the **integration by parts**, we have

$$\int_0^1 -(pu')'v\,dx = \big[-(pu')v\big]_0^1 + \int_0^1 pu'v'\,dx = \int_0^1 pu'v'\,dx, \quad v \in V. \tag{15.5}$$

- The differential problem (D) can be written as the **weak form**

$$\int_0^1 pu'v'\,dx = \int_0^1 fv\,dx, \quad v \in V. \tag{15.6}$$

- Define a bilinear functional and a linear product as

$$B(u,v) = \int_0^1 pu'v'\,dx, \quad (f,v) = \int_0^1 fv\,dx. \tag{15.7}$$

Now, we define the **variational problem** corresponding to the differential problem (D):

$$\text{(V)} \quad \text{Find } u \in V \text{ such that } B(u,v) = (f,v), \text{ for all } v \in V \tag{15.8}$$

Theorem 15.1. *The differential problem (D) in its **strong form** is equivalent to the variational problem (V), when the solutions are sufficiently smooth.*

Proof. ((D) $\Rightarrow$ (V)): Clear.
((D) $\Leftarrow$ (V)): Let u be a solution of (V). Then,

$$B(u,v) = (f,v), \quad \text{for all } v \in V. \tag{15.9}$$

Now, **assume that u'' exists.** Then, since

$$B(u,v) = \int_0^1 pu'v'\,dx = pu'v\Big|_0^1 - \int_0^1 (pu')'v\,dx = \int_0^1 -(pu')'v\,dx$$

and

$$(f,v) = \int_0^1 fv\,dx,$$

it follows from (15.9) that

$$\int_0^1 \big[(pu')' + f\big]v\,dx = 0, \quad \text{for all } v \in V. \tag{15.10}$$

So u should satisfy (15.1). □

Minimization Problem

Let $F : V \to \mathbb{R}$ be a functional defined as

$$F(v) = \frac{1}{2}B(v, v) - (f, v). \tag{15.11}$$

We define the **minimization problem**

$$\text{(M)} \quad \text{Find } u \in V \text{ such that } F(u) \leq F(v), \text{ for all } v \in V \tag{15.12}$$

Theorem 15.2. *The minimization problem (M) is equivalent to the variational problem (V).*

Theorem 15.3. *Problem (V) admits a unique solution, provided the condition (15.2) is satisfied.*

Summary 15.4.

- (D) $\iff$ (V) $\iff$ (M) [(D) $\Leftarrow$ (V), when u'' exists]
- They admit a unique solution.

Remark 15.5. **Strong Form vs. Weak Form & FEM**

- The **strong form** represents the original differential equations and boundary conditions describing the physical problem.
- The **weak form** is an integral formulation derived from the strong form, ***relaxed to allow for more manageable numerical approximation.***
- The **finite element method** (FEM) finds numerical solutions from the weak form.
  - Makes the problem more amenable to numerical solution.
  - Reduces smoothness requirements.
  - Facilitates the handling of complex geometries and boundary conditions.

15.2. Finite Element Methods (FEMs)

Recall: As shown in **Scheme** 14.12, the ***FD discretization procedure*** consists of four steps:

1. Partitioning **(Split the zone!)**
2. FD approximations **(Unknowns get shown!)**
3. System of difference equations **(Equations arise!)**
4. Assembly **(An algebraic system flies!)**

FEMs have also four steps.

Formulation 15.6. **Four Steps in FEMs**

1. **Partitioning**: The domain should be partitioned into a collection of elements of the mesh size h.
2. **Subspace $V_h \subset V$ and basis functions $\{\varphi_j(x)\}$**: A subspace is set to represent the numerical solution that is a linear combination of basis functions. That is,

$$u_h(x) = \sum_{j=1}^{M} \xi_j \varphi_j(x). \tag{15.13}$$

   For example, $\varphi_j(x)$ are piecewise polynomials (splines).
3. **Application of variational principles**: Different variational principles produce various FEMs.
   - the **minimization principle** (Rayleigh-Ritz)
   - **weighted residual approaches**
     with the weights being either the basis functions (Galerkin) or different functions (Petrov-Galerkin)
   - **least-square approaches**
   - **collocation method**
4. **Assembly for a linear system**: The linear system can be assembled for $(\xi_1, \xi_2, \cdots, \xi_M)^T$ with the integrals approximated by numerical quadrature.

Step 1. Partitioning

Let

$$0 = x_0 < x_1 < \cdots < x_M < x_{M+1} = 1$$

be a partition of the unit interval. Define

$$h_j = x_j - x_{j-1}, \quad I_j = [x_{j-1}, x_j], \quad j = 1, 2, \cdots, M+1$$

and

$$h = \max_{1 \le j \le M+1} h_j.$$

(Split the zone!)

Step 2. Subspace and Basis Functions

Subspace: Define a finite-dimensional subspace of V as

$$V_h = \{v \in V : v \text{ is a polynomial of degree} \le k \text{ on each } I_j\}. \tag{15.14}$$

Basis Functions:

- Corresponding basis functions are determined depending on the choice of polynomial degree $k \ge 1$ and therefore on the nodal points.
- Each of basis functions is related to a nodal point.
- Basis functions $\varphi_j \in V_h$ are often defined to satisfy

$$\varphi_j(x_i) = \delta_{ij} := \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{else.} \end{cases} \tag{15.15}$$

Example 15.7. **When $k = 1$ (the linear FEM)**:
The basis function φ_j is defined as follows and depicted in Figure 15.1:

$$\varphi_j(x) = \begin{cases} \dfrac{1}{h_j}(x - x_{j-1}), & x \in [x_{j-1}, x_j], \\ \dfrac{-1}{h_{j+1}}(x - x_{j+1}), & x \in [x_j, x_{j+1}], \\ 0, & \text{elsewhere.} \end{cases} \tag{15.16}$$

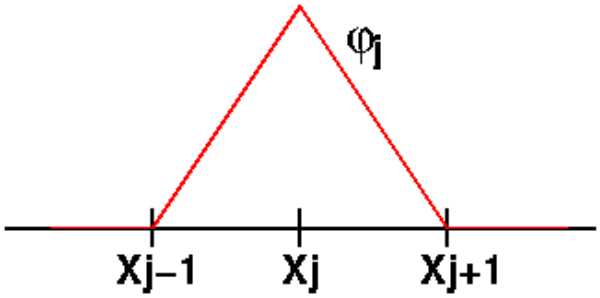


Figure 15.1: The basis function φ_j.

Note: **Functions $v \in V_h$**

- The functions $v \in V_h$ can be expressed as a linear combination of the basis functions as

$$v(x) = \sum_{j=1}^{M} \eta_j \varphi_j(x), \quad x \in [0,1]. \tag{15.17}$$

- The above expression is unique for given $v \in V_h$; in fact,

$$\eta_j = v(x_j), \quad j = 1, 2, \cdots, M.$$

(Unknowns get shown!)

Example 15.8. **When $k > 1$ (higher-order FEMs)**

- **Degree of Freedom**: For each interval $I_j = [x_{j-1}, x_j]$, the **degree of freedom** of k-th order polynomials is $k+1$.
  $\Rightarrow$ It requires to choose $k+1$ nodal points in each interval.
- **Choice of Nodal Points**: As for the linear FEM, the two endpoints can naturally become nodal points.
  $\Rightarrow$ We should select $k-1$ extra nodal points inside the interval I_j.
- **Accuracy Concerns**: In the literature, a common practice is to select those nodal points in such a way that the numerical quadrature of the integrals is as accurate as possible when the nodal points are used as quadrature points.
- **Orthogonal Polynomials**: Such selection is related to the family of **orthogonal polynomials** such as Legendre polynomials and Chebyshev polynomials.

Step 3. Application of Variational Principles

The most popular FEM is the **Galerkin method**, which is a *weighted residual approach* with the weights being basis functions.

Weighted Residual Approaches

- Let $P(u) = -(pu')'$.
- For the differential problem (D) in (15.1), define the **residual** R as

$$R(v) = P(v) - f. \tag{15.18}$$

- Then, for $u \in V$, we have

$$R(u) = P(u) - f = 0.$$

However, for its approximation $u_h(x) = \sum_{j=1}^{M} \xi_j \varphi_j(x)$,

$$R(u_h) = P(u_h) - f \neq 0, \quad \text{in general.} \tag{15.19}$$

- Weighed residual approaches are seeking an approximate solution

$$u_h(x) = \sum_{j=1}^{M} \xi_j \varphi_j(x),$$

which satisfies

$$\int_0^1 R(u_h)\, w(x)\, dx = 0, \tag{15.20}$$

for a sequence of **weight functions** $w(x) \in \{w_i(x)\}$, which is also called **trial functions**.

- Note that (15.20) reads in detail

$$0 = \int_0^1 R(u_h)\, w(x)\, dx = \int_0^1 \big[-(pu_h')' - f \big] w(x)\, dx. \tag{15.21}$$

- When the integration by parts is utilized, (15.21) becomes

$$B(u_h, w) = (f, w). \tag{15.22}$$

Algorithm 15.9. **The Linear Galerkin Method**

For the subspace V_h of **linear** basis functions $\{\varphi_j(x)\}$, let

$$w_i(x) = \varphi_i(x). \tag{15.23}$$

Then, the **linear Galerkin FEM** for the differential problem (D) in (15.1) is formulated as

$$\text{Find } u_h \in V_h \text{ s.t. } B(u_h, \varphi_i) = (f, \varphi_i), \text{ for all } \varphi_i \in V_h. \tag{15.24}$$

As in §15.1, one can show that (15.24) admits a unique solution.

(Equations arise!)

Step 4. Assembly for a Linear System

- Given basis functions $\{\varphi_j(x)\} \subset V_h$, the numerical solution u_h is uniquely expressed as

$$u_h(x) = \sum_{j=1}^{M} \xi_j \varphi_j(x). \tag{15.25}$$

- The numerical solution must be the solution of a variational formulation. For example, the solution of the **linear Galerkin FEM** satisfies

$$B(u_h, \varphi_i) = (f, \varphi_i), \text{ for all } \varphi_i \in V_h. \tag{15.26}$$

- The objective is to assemble a linear system for the unknown vector $\boldsymbol{\xi} := (\xi_1, \xi_2, \cdots, \xi_M)^T$. From (15.25) and (15.26),

$$B(u_h, \varphi_i) = \sum_{j=1}^{M} \xi_j B(\varphi_j, \varphi_i) = (f, \varphi_i), \text{ for all } \varphi_i \in V_h. \tag{15.27}$$

- Define

$$a_{ij} = B(\varphi_j, \varphi_i) = \int_0^1 \mathbf{p}\, \varphi_\mathbf{j}' \varphi_\mathbf{i}' \, \mathbf{dx}, \quad b_i = (f, \varphi_i) = \int_0^1 \mathbf{f} \varphi_\mathbf{i} \, \mathbf{dx}. \tag{15.28}$$

Then (15.27) can be written as

$$A\boldsymbol{\xi} = \mathbf{b}, \tag{15.29}$$

where $A = [a_{ij}] \in \mathbb{R}^{M \times M}$ and $\mathbf{b} = (b_1, b_1, \cdots, b_M)^T$ is the **source vector**.

Remark 15.10. The Algebraic System $A\xi = \mathrm{b}$

- The remaining task is **to compute a_{ij} and b_i.**
- We will consider details for the computation of a_{ij} and b_i. Note that

$$a_{ij} = B(\varphi_j, \varphi_i) = \int_0^1 \mathrm{p}\,\varphi_\mathrm{j}'\varphi_\mathrm{i}'\,\mathrm{dx} = 0, \quad \text{if } |i-j| \geq 2, \tag{15.30}$$

because the support of φ_j is $[x_{j-1}, x_{j+1}]$. Thus, in the i-th row of A, there are only three cases for nonzero entries:

$$j = i-1,\ i,\ i+1.$$

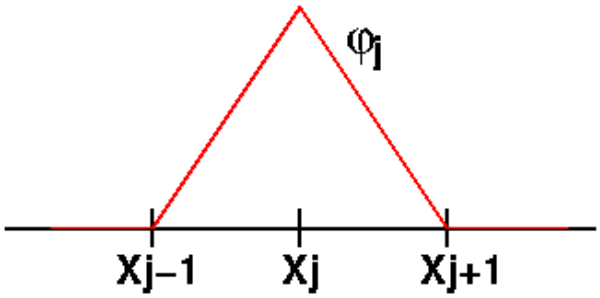


Figure 15.2: The linear basis function φ_j, as shown in Figure 15.1.

Remark 15.11. Each part of the basis function determines the shape of the approximate solution u_h, so is called a **shape function**.

Recall: The basis function φ_j in (15.16):

$$\varphi_j(x) = \begin{cases} \dfrac{1}{h_j}(x - x_{j-1}), & x \in [x_{j-1}, x_j], \\ \dfrac{-1}{h_{j+1}}(x - x_{j+1}), & x \in [x_j, x_{j+1}], \\ 0, & \text{elsewhere.} \end{cases} \tag{15.31}$$

Its derivative reads

$$\varphi_j'(x) = \begin{cases} 1/h_j, & x \in [x_{j-1}, x_j], \\ -1/h_{j+1}, & x \in [x_j, x_{j+1}], \\ 0, & \text{elsewhere.} \end{cases} \tag{15.32}$$

Computation of a_{ij} and b_i

- $a_{i,i-1}$ (when $j = i-1$): It follows from (15.32) that

$$\begin{aligned} a_{i,i-1} &= \int_{x_{i-1}}^{x_i} p\,\varphi_{i-1}'(x)\varphi_i'(x)dx \\ &= \int_{x_{i-1}}^{x_i} p\,\frac{-1}{h_i}\cdot\frac{1}{h_i}dx \approx \frac{-1}{h_i}\cdot\frac{p_{i-1}+p_i}{2}, \end{aligned} \tag{15.33}$$

  where the **trapezoid rule** is used for the evaluation of the integral. See Section 4.3.1 for the trapezoid rule.

- $a_{i,i}$ (when $j = i$): Again utilizing (15.32), we have

$$\begin{aligned} a_{i,i} &= \int_{x_{i-1}}^{x_{i+1}} p\,\varphi_i'(x)\varphi_i'(x)dx = \int_{x_{i-1}}^{x_i} + \int_{x_i}^{x_{i+1}} p\,\varphi_i'(x)\varphi_i'(x)dx \\ &= \int_{x_{i-1}}^{x_i} p\frac{1}{h_i^2}\,dx + \int_{x_i}^{x_{i+1}} p\frac{i}{h_{i+1}^2}\,dx \\ &\approx \frac{1}{h_i}\cdot\frac{p_{i-1}+p_i}{2} + \frac{1}{h_{i+1}}\cdot\frac{p_i+p_{i+1}}{2}. \end{aligned} \tag{15.34}$$

- $a_{i,i+1}$ (when $j = i+1$):

$$\begin{aligned} a_{i,i+1} &= \int_{x_i}^{x_{i+1}} p\,\varphi_{i+1}'(x)\varphi_i'(x)dx \\ &= \int_{x_i}^{x_{i+1}} p\,\frac{1}{h_{i+1}}\cdot\frac{-1}{h_{i+1}}dx \approx \frac{-1}{h_{i+1}}\cdot\frac{p_i+p_{i+1}}{2}. \end{aligned} \tag{15.35}$$

- b_i: Finally, it can be done as

$$b_i = (f,\varphi_i) = \int_{x_{i-1}}^{x_{i+1}} f(x)\varphi_i(x)dx \approx \frac{h_i+h_{i+1}}{2}\cdot f_i, \tag{15.36}$$

  where $f_i = f(x_i)$.

The above results in the algebraic system

$$A\boldsymbol{\xi} = \mathrm{b}, \tag{15.37}$$

where A is called the **stiffness matrix**.

(An algebraic system flies!)

Properties of the Algebraic System

Recall: (**Definition 10.37**, p.371)
A matrix $S = (s_{ij}) \in \mathbb{R}^{M\times M}$ is said to be **positive definite** if

$$\eta \cdot S\eta = \sum_{i,j=1}^{M} \eta_i s_{ij} \eta_j > 0, \quad \text{for all } \eta \in \mathbb{R}^M, \ \eta \neq 0. \tag{15.38}$$

It has been known that *a matrix S is symmetric positive definite if and only if all eigenvalues of S are strictly positive.*

Lemma 15.12. The matrix A in (15.37) is symmetric positive definite.

Proof. Symmetry is easy to see, because

$$a_{ij} := \int_0^1 p\,\varphi_j'\varphi_i'\,dx = \int_0^1 p\,\varphi_i'\varphi_j'\,dx =: a_{ji}.$$

Given $\eta \in \mathbb{R}^M$, we define $v(x) = \sum_{j=1}^{M} \eta_j \varphi_j(x)$. Then

$$\begin{aligned} \eta \cdot A\eta &= \sum_{i,j=1}^{M} \eta_i a_{ij} \eta_j = \sum_{i,j=1}^{M} \eta_i \Big[\int_0^1 p\,\varphi_j'\varphi_i'\,dx \Big] \eta_j \\ &= \int_0^1 p \Big[\sum_{i,j=1}^{M} \eta_i \varphi_j' \varphi_i \eta_j \Big]\, dx \\ &= \int_0^1 p \Big(\sum_{i=1}^{M} \eta_i \varphi_i' \Big) \Big(\sum_{j=1}^{M} \eta_j \varphi_j' \Big) dx \ = \ \int_0^1 p (v')^2 dx \geq 0, \end{aligned} \tag{15.39}$$

with equality satisfied only if $v' = 0$, and therefore only if $v = 0$ because $v(0) = 0$; which implies that equality holds only if $\eta = 0$. This completes the proof. □

Higher-order FEMs

- Higher-order FEMs introduce **higher-order basis functions.**
- As an example, Figure 15.3 presents the element $I_i = [x_{i-1}, x_i]$ and the basis functions each of which is **cubic in I_i**.
  - Since the **degree of freedom** for cubic polynomials is four, we need to provide four independent information to determine the polynomial **uniquely**.
  - One can choose four distinct points (including two edge points), as shown in Figure 15.3. The points are called the **nodal points**.

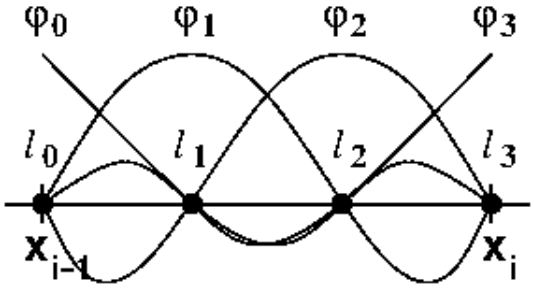


Figure 15.3: The element $I_i = [x_{i-1}, x_i]$ and the basis functions for the **cubic FEM**.

Construction of General-order Basis Functions

For kth-order FEMs, $k \geq 1$, here we consider **how to construct kth-order basic functions**, piecewise kth-order polynomials.

- In each element I_i, select extra $(k-1)$ nodal points, which makes $(k+1)$ nodal points including the two edge points.
  - It is often to choose **Gauss-Lobatto points** for the interior nodal points; see Theorem 4.44, p. 175.
- Denote them by ℓ_m, $m = 0, \cdots, k$.
- Define the local basis functions as

$$\varphi_j(x) = \prod_{m=0,\, m\neq j}^{k} \left(\frac{x-\ell_m}{\ell_j - \ell_m}\right), \quad j = 0, \cdots, k.$$

These are **Lagrange cardinal functions**; see Summary 3.11, p. 91.

Error Analysis for FEMs

Theorem 15.13. *Let u and u_h be the solutions of Problem* (V) *and the linear FEM, respectively. Then*

$$\|u - u_h\| \leq Ch^2 ||u''||, \tag{15.40}$$

where $C > 0$ is independent of h.

For kth-order FEMs, $k \geq 1$, the error in general is in $\mathcal{O}(h^{k+1})$.

Example 15.14. Use the linear Galerkin FEM to approximate the solution to the boundary-value problem

$$\begin{aligned} -\frac{d}{dx}\Big((x+1)\frac{d}{dx}u\Big) + 6u &= -12x^4 + 44x^3 - 2x + 1, \quad 0 \leq x \leq 1, \\ u(0) &= u(1) = 0, \end{aligned} \tag{15.41}$$

with the nodal points $\{0, 0.25, 0.5, 0.75, 1\}$. Compare your result with the exact solution $u(x) = -2x^4 + 2x^3 - x^2 + x$. (Note that this problem involves a reaction term.)

Solution. The **stiffness matrix** A is *often* formulated from the **elementary stiffness matrices**.

elementary_Info.m

```
function [Ae,be] = elementary_Info(element,p,q,f)
% This routine computes the elementary stiffness matrix and the source.

Ae = zeros(2,2); be = zeros(2,1);
x0 = element(1); x1 = element(2); h = x1-x0;

Ae(1,1) = (1/h) * (p(x0)+p(x1))/2 + h*(q(x0)/2);
Ae(2,2) = (1/h) * (p(x0)+p(x1))/2 + h*(q(x1)/2);
Ae(1,2) =-(1/h) * (p(x0)+p(x1))/2;
Ae(2,1) = Ae(1,2);

be(1) =  (h/2) * f(x0);
be(2) =  (h/2) * f(x1);
```

FEM1_Galerkin.m

```
close all

nodes = linspace(0,1,5);
p = @(x) (x+1);
q = @(x) (6);
f = @(x) (-12*x.^4 + 44*x.^3 - 2*x  + 1);

%%--- Assembly
n = length(nodes); A = zeros(n,n); b = zeros(n,1);
for i=2:n
    element = [nodes(i-1),nodes(i)];
    [Ae,be] = elementary_Info(element,p,q,f);
    A(i-1:i,i-1:i) = A(i-1:i,i-1:i) + Ae;
    b(i-1:i) = b(i-1:i) + be;
end
A,bT=b'

%%-- Solve for the interior
fprintf('FEM Solution:\n')
A0 = A(2:n-1,2:n-1); b0 = b(2:n-1);
uh = zeros(n,1);
uh(2:n-1) = A0\b0;
uhT = uh'

%%-- Checking Error
u = @(x) (-2 *x.^4 + 2 *x.^3 - x.^2 + x);
U = u(nodes);
fprintf('m = %2d; max-error = %.6f\n',n-1,norm(abs(U-uhT),Inf));

%%-- Plotting
if n==5,
    x = linspace(0,1,101);
    figure, plot(x,u(x),'b-','linewidth',1.5); hold on
    plot(nodes,uh,'r-*','linewidth',2);
    legend('Exact Solution','FEM Solution','location','best')
    title('Linear Galerkin FEM')
    exportgraphics(gca,'Linear-Galerkin-FEM.png','Resolution',100)
    hold off
end
```

Output

```
>> FEM1_Galerkin
A =
    5.2500   -4.5000         0         0         0
   -4.5000   11.5000   -5.5000         0         0
         0   -5.5000   13.5000   -6.5000         0
         0         0   -6.5000   15.5000   -7.5000
         0         0         0   -7.5000    8.2500

bT =
    0.1250    0.2852    1.1875    3.5664    3.8750

FEM Solution:
uhT =
         0    0.1904    0.3462    0.3753         0

m =  4; max-error = 0.028788
m =  8; max-error = 0.007266
m = 16; max-error = 0.001821
m = 32; max-error = 0.000457
```

The error reduces in $\mathcal{O}(h^2)$.

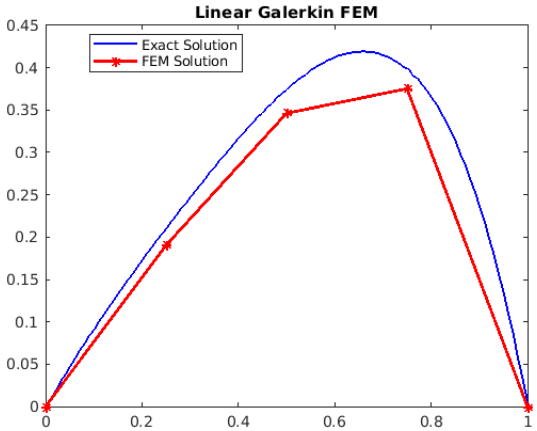


Figure 15.4: The linear Galerkin FEM solution, with the exact solution.

15.3. Other Variational Principles for FEMs

The FEM we have considerd so far is the **Galerkin method**, which is a weighted residual approach with ***the weights being basis functions.***

There have been other variational principles such as

- the minimization principle (Rayleigh-Ritz methods),
- least-square approaches,
- collocation methods, and
- weighted residual approaches with the weights being different from the basis functions (Petrov-Galerkin methods).

15.3.1. Rayleigh-Ritz Method

Once basis functions are determined, the approximate solution u_h can be formulated as in (15.13), that is,

$$u_h(x) = \sum_{j=1}^{M} \xi_j \varphi_j(x). \tag{15.42}$$

- The **Rayleigh-Ritz Method** seeks an approximate solution of the form (15.42) which satisfies

$$u_h = \arg\min_{v \in V_h} F(v), \tag{15.43}$$

where

$$F(v) = \frac{1}{2} B(v,v) - (f,v) = \frac{1}{2}\int_0^1 pv'v'\,dx - \int_0^1 fv\,dx. \tag{15.44}$$

- It follows from (15.42) and (15.44) that

$$F(u_h) = \frac{1}{2}\int_0^1 p\Big(\sum_{j=1}^{M} \xi_j \varphi_j'\Big)^2 dx - \int_0^1 f\Big(\sum_{j=1}^{M} \xi_j \varphi_j\Big)\,dx. \tag{15.45}$$

- In order for a minimum to occur, it is necessary, when considering as a function of $(\xi_1, \xi_2, \cdots, \xi_M)$ to have the **normal equations**

$$\frac{\partial}{\partial \xi_i} F(u_h) = 0, \quad i = 1, 2, \cdots, M. \tag{15.46}$$

- From (15.45) and (15.46), we have

$$\frac{\partial}{\partial \xi_i} F(u_h) = \int_0^1 p\Big(\sum_{j=1}^{M} \xi_j \varphi_j'\Big)\varphi_i' \, dx - \int_0^1 f\varphi_i \, dx = 0, \tag{15.47}$$

  which equivalently reads

$$\sum_{j=1}^{M} \Big(\int_0^1 p\varphi_j'\varphi_i' \, dx\Big)\xi_j = \int_0^1 f\varphi_i \, dx \tag{15.48}$$

  and

$$B(u_h, \varphi_i) = (f, \varphi_i), \quad i = 1, 2, \cdots, M. \tag{15.49}$$

- Letting

$$a_{ij} := \int_0^1 p\varphi_j'\varphi_i' \, dx, \quad b_i := \int_0^1 f\varphi_i \, dx, \tag{15.50}$$

  the normal equations in (15.48) produce an $M \times M$ linear system

$$A\boldsymbol{\xi} = \mathbf{b}. \tag{15.51}$$

Note: The resulting linear system of the Rayleigh-Ritz method is the same as that of the Galerkin method; see (15.29), p. 541. In general, they are the same for linear PDEs.

15.3.2. Least-Squares Method

Recall: Earlier on p. 540:

- Let $P(u) = -(pu')'$.
- For the differential problem (D) in (15.1), define the **residual** R as

$$R(v) = P(v) - f. \tag{15.52}$$

- Then, for $u \in V$, we have

$$R(u) = P(u) - f = 0.$$

However, for its approximation $u_h(x) = \sum_{j=1}^{M} \xi_j \varphi_j(x)$,

$$R(u_h) = P(u_h) - f \neq 0, \quad \text{in general.} \tag{15.53}$$

Definition 15.15. **The Least-Squares FEM** is formulated as

$$u_h = \arg\min_{v \in V_h} \int_0^1 R(v)^2 \, dx, \tag{15.54}$$

which is equivalent to

$$\frac{\partial}{\partial \xi_i} \int_0^1 R(u_h)^2 \, dx = 0, \quad i = 1, 2, \cdots, M. \tag{15.55}$$

Differentiating the integral reads

$$\int_0^1 R(u_h) w(x) \, dx = 0, \tag{15.56}$$

where

$$w(x) = \frac{\partial}{\partial \xi_i} R(u_h) = -(p\varphi_i')' = P(\varphi_i).$$

Thus the **least-squares FEM** can be formulated as

$$\sum_{j=1}^{M} \Big(\int_0^1 P(\varphi_j) P(\varphi_i) \, dx \Big) \xi_i = \int_0^1 f P(\varphi_i) \, dx. \tag{15.57}$$

Remark **15.16.** **Least-Squares FEM**

- Although rarely happening, utilizing the integration by parts gives

$$B(u_h, P(\varphi_i)) = (f, P(\varphi_i)). \tag{15.58}$$

- Due to the derivatives applied on the basis functions, the basis functions must be high-order polynomials locally.

15.3.3. Collocation Method

Definition **15.17.** **The Collocation FEMs** are formulated as

$$\int_0^1 R(u_h)w(x)\,dx = 0, \tag{15.59}$$

with a set of **collocation points** $\{x_1, x_2, \cdots, x_M\}$:

$$w(x) = \delta(x - x_i). \tag{15.60}$$

Since

$$\int_0^1 R(u_h(x))\delta(x - x_i)\,dx = R(u_h(x_i)),$$

the collocation method produces an $n \times n$ algebraic system:

$$\sum_{j=1}^{M} -\xi_j(p(x)\varphi_j(x)')'\big|_{x=x_i} = f(x_i), \quad i = 1, 2, \cdots, M. \tag{15.61}$$

Note: **Collocation FEM**

- Much attention in the literature has been given to the choice of the basis functions $\{\varphi_j\}$ and the collocation points $\{x_i\}$.
- One popular choice is that $\{\varphi_j\}$ are cubic B-Splines and $\{x_i\}$ are Gaussian points.

Example 15.18. (Revisit of Example 15.14, p.546)
Consider the boundary-value problem

$$-\frac{d}{dx}\left((x+1)\frac{d}{dx}u\right)+6u=-12x^4+44x^3-2x+1, \quad 0\le x\le 1, \\ u(0)=u(1)=0, \tag{15.62}$$

of which the exact solution $u(x)=-2x^4+2x^3-x^2+x$. Find the numerical solutions, using the collocation method.

Solution. **Collocation FEM**: Let's select three basis functions

$$\varphi_1(x)=x(x-1), \quad \varphi_2(x)=x^2(x-1), \quad \varphi_3(x)=x^2(x-1)^3, \tag{15.63}$$

and collocation points: $\{1/4, 1/2, 3/4\}$.

collocation_FEM.py

```
# import library
import numpy as np; np.set_printoptions(suppress=True)
import sympy as sym
from sympy import plot

#--- Define symbolic variable
x = sym.symbols('x')
c1, c2, c3 = sym.symbols('c1 c2 c3')

#--- Define functions
p = x+1; q = 6
f = -12*x**4 + 44*x**3 - 2*x + 1
u = -2*x**4 + 2*x**3 - x**2 + x

#--- Basis functions, uh, and P(uh)
phi1 = x*(x-1)
phi2 = x**2*(x-1)
phi3 = x**2*(x-1)**3
uh = c1*phi1 + c2*phi2 + c3*phi3

Puh = -sym.diff( p*sym.diff(uh,x), x) + q*uh
print("P(uh) =",Puh)
Puh = sym.simplify(Puh); Puh = sym.collect(Puh,x)
print("P(uh) =",Puh)

#--- Collocation method
coll = [1/4,1/2,3/4]
```

```
A =np.ndarray([len(coll),len(coll)])
b =np.ndarray([len(coll),1])
for i,xi in enumerate(coll):
    Pxi = Puh.evalf(subs={x:xi})
    A[i] = [Pxi.coeff(c1),Pxi.coeff(c2),Pxi.coeff(c3)]
    b[i] = f.evalf(subs={x:xi})
print("[A,b] = \n",np.hstack((A,b)))
sol = np.linalg.solve(A,b); sol = sol[:,0]
print("sol =", sol.T)

uh = uh.subs(c1,sol[0]).subs(c2,sol[1]).subs(c3,sol[2])
print("uh =",uh)

#--- Plotting
pu = plot(u, (x, 0, 1), linewidth=3,line_color='blue',legend=True,show=False)
pu[0].label = 'Exact Solution'
puh= plot(uh, (x, 0, 1), linewidth=2,line_color='red',legend=True,show=False)
puh[0].label = 'Collocation Solution'
pu.extend(puh)
pu.xlabel='x'; pu.ylabel='u'
pu.save('collocation-FEM.png')
pu.show()
```

Result

```
P(uh) = 6*c1*x*(x - 1) - c1*x - c1*(x - 1) + 6*c2*x**2*(x - 1) - c2*x**2
  - 2*c2*x*(x - 1) + 6*c3*x**2*(x - 1)**3 - 3*c3*x**2*(x - 1)**2
  - 2*c3*x*(x - 1)**3 - (x + 1)*(2*c1 + 4*c2*x + 2*c2*(x - 1)
  + 3*c3*x**2*(2*x - 2) + 12*c3*x*(x - 1)**2 + 2*c3*(x - 1)**3)
P(uh) = -c1 + 2*c2 + 6*c3*x**5 - 43*c3*x**4 + 2*c3 + x**3*(6*c2 + 46*c3)
  + x**2*(6*c1 - 15*c2 + 3*c3) + x*(-10*c1 - 2*c2 - 14*c3)
[A,b] =
 [[-3.125       0.65625    -0.75585938  1.140625  ]
 [-4.5        -2.         -1.          4.75      ]
 [-5.125      -5.40625     0.41210938 14.265625  ]]
sol = [-1.4846609  -0.94069413  3.81236231]
uh = 3.81236231224397*x**2*(x - 1)**3 - 0.940694128356851*x**2*(x - 1)
  - 1.48466090122895*x*(x - 1)
```

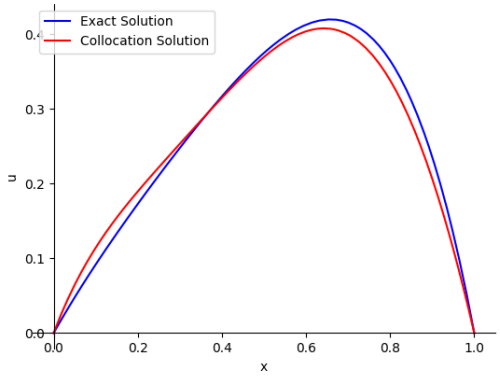


Figure 15.5: The exact solution, with the numerical solution from the collocation FEM.

15.4. FEMs for More General Boundary-Value Problems

For a deeper understanding of FEMs, we consider approximating the solution to a more general boundary-value problem of the form

$$\text{(D)} \quad \begin{cases} -\dfrac{d}{dx}\Big(p(x)\dfrac{du}{dx}\Big) + q(x)\dfrac{du}{dx} + r(x)u = f(x), \quad x \in [0,1], \\ u(0) = \alpha, \;\; u'(1) = \beta. \end{cases} \tag{15.64}$$

The **Dirichlet BC** is called an **essential BC**, while the **Neumann BC** is called a **natural BC**.

Variational Formulation

- Define linear spaces

$$\begin{aligned} V &= \big\{v \mid v \in C^0[0,1], v' \text{ is piecewise continuous on } [0,1]\big\} \\ V_0 &= \{v \in V \mid v(0) = 0\}. \end{aligned} \tag{15.65}$$

- Since, by the integration by parts,

$$\begin{aligned} \int_0^1 -(pu')'v\,dx &= [-(pu')v]_{x=0}^{x=1} + \int_0^1 pu'v'\,dx \\ &= -p(1)\beta v(1) + \int_0^1 pu'v'\,dx, \quad \text{for all } v \in V_0, \end{aligned} \tag{15.66}$$

  the differential problem (D) can be written as the **weak form**

$$\int_0^1 pu'v'\,dx + \int_0^1 qu'v\,dx + \int_0^1 ruv\,dx = \int_0^1 fv\,dx + \beta p(1)v(1). \tag{15.67}$$

- Define a **bilinear functional** $B(u,v)$ and a **linear term** $\ell(v)$ as to be the left side and right side of the above equation, respectively:

$$B(u,v) = \ell(v), \quad \text{for all } v \in V_0. \tag{15.68}$$

- Now, we define the **variational problem** corresponding to the differential problem (D):

$$\text{(V)} \quad \text{Find } u \in V \text{ such that } B(u,v) = \ell(v), \;\; \text{for all } v \in V_0 \tag{15.69}$$

Theorem 15.19. *The differential problem (D) is equivalent to the variational problem (V), when u'' exists.*

The Linear Galerkin FEM

- **Partitioning**: Let
$$0 = x_0 < x_i < \cdots < x_M = 1,$$
with
$$I_j = [x_{j-1}, x_j], \;\; h_j = x_j - x_{j-1}, \;\; (1 \le j \le M); \quad h = \max_{1 \le j \le M} h_j.$$

- **Subspaces**:
$$V_h = \{v \in V \mid v\text{: linear on each } I_j\}, \;\; V_{0,h} = \{v \in V_h \mid v(0) = 0\}. \tag{15.70}$$
When the basis functions are chosen as
$$\{\varphi_j(x) \mid j = 0, 1, \cdots, M\},$$
the finite element solution is
$$u_h(x) = \sum_{j=0}^{M} \xi_j \varphi_j(x), \quad \xi_0 = \alpha. \tag{15.71}$$

- **Application of Variational Principles**: The linear Galerkin FEM is formulated as
$$B(u_h, \varphi_i) = \ell(\varphi_i), \quad i = 1, 2, \cdots, M, \tag{15.72}$$
which can be rewritten as
$$\sum_{j=0}^{M} B(\varphi_j, \varphi_i)\xi_j = \ell(\varphi_i), \quad i = 1, 2, \cdots, M, \tag{15.73}$$
where
$$\begin{aligned} B(\varphi_j, \varphi_i) &= \int_0^1 p\varphi_j'\varphi_i' \, dx + \int_0^1 q\varphi_j'\varphi_i \, dx + \int_0^1 r\varphi_j\varphi_i \, dx \;\; (= a_{ij}), \\ \ell(\varphi_i) &= \int_0^1 f\varphi_i \, dx + \beta p(1)\varphi_i(1) \;\; (= b_i). \end{aligned} \tag{15.74}$$

- **Assembly**: An algebraic system for (15.73), by evaluating (15.74).

Example 15.20. (Revisit of Example 14.15, p.523, FDM)
Consider the boundary-value problem

$$\begin{aligned} -y'' + \frac{1}{x}y' + \frac{3}{x^2}y &= 4x^2, \quad x \in [1,2] \\ y(1) = 0, \;\; y'(2) &= -2. \end{aligned} \tag{15.75}$$

Use the **linear Galerkin FEM** to approximate the solution of the problem, with $h = 1/5$ and $h = 1/10$. The exact solution is

$$y(x) = -8/(7 * \mathtt{x}) + (68/35) * \mathtt{x}^3 - (4/5) * \mathtt{x}^4. \tag{15.76}$$

Solution.

Linear Galerkin FEM

```
 x     exact_y(x) uh(h=1/5) error   uh(h=1/10) error  Richardson error
x=1.20 0.74600 | 0.74927 0.00328 | 0.74681 0.00081 | 0.74598 1.155e-005
x=1.40 1.44159 | 1.44769 0.00610 | 1.44311 0.00151 | 1.44158 1.498e-005
x=1.60 2.00078 | 2.00842 0.00764 | 2.00268 0.00190 | 2.00076 1.256e-005
x=1.80 2.29774 | 2.30483 0.00709 | 2.29951 0.00177 | 2.29774 4.459e-006
x=2.00 2.17143 | 2.17505 0.00362 | 2.17234 0.00091 | 2.17144 9.758e-006
```

FDM, Section 14.2

```
       exact_y(x) uh(h=1/5) error   uh(h=1/10) error  Richardson error
x=1.20 0.74600 | 0.75688 0.01088 | 0.74871 0.00271 | 0.74598 1.423e-05
x=1.40 1.44159 | 1.46883 0.02723 | 1.44840 0.00680 | 1.44159 6.646e-06
x=1.60 2.00078 | 2.04943 0.04865 | 2.01295 0.01217 | 2.00078 7.089e-06
x=1.80 2.29774 | 2.37302 0.07527 | 2.31658 0.01884 | 2.29777 2.315e-05
x=2.00 2.17143 | 2.27883 0.10740 | 2.19831 0.02688 | 2.17147 4.068e-05
```

Is the Galerkin FEM better than the FDM?

Finite Element Method - Break It Down with FEM
YouTube (Melodic Rap) (Rock)

Exercises for Chapter 15

Consider the boundary-value problem

$$-\frac{d}{dx}\left(e^x\frac{d}{dx}u\right)+e^x u=(2-x)e^x+x,\quad 0\le x\le 1,$$
$$u(0)=u(1)=0. \tag{15.77}$$

The actual solution is $u(x)=(x-1)(e^{-x}-1)$. Find numerical solutions of the problem.

15.1. C First use the **linear Rayleigh-Ritz-Galerkin FEM**.

(a) Carry out the integrations using the Trapezoidal rule.
(b) Select $h=10$ and $h=20$.
(c) Compare both of your results with the actual solution
(d) Measure the errors for the approximate solutions computed in (c) and see if the numerical solutions show the desired accuracy.

15.2. C Now, use the **method of collocation** to approximate the solution.

(a) Choose a set of three basis functions in your own. (The basis functions should be linearly independent each other and satisfy the boundary condition.)
(b) Select the three nodal points inside $[0,1]$ conveniently, again in your own.
(c) Compare your result with the actual solution.
(d) Add one more basis function and a nodal point and find the numerical solution to see if accuracy is improved.

CHAPTER 16

Properties of Data-First Numerical Methods

In This Chapter: We will study properties of numerical methods, both traditional and modern. See the table below for a comparison.

Table 16.1: Traditional vs. modern properties of numerical methods, with key challenges.

Property	Traditional Focus	Modern Expansion	Key Challenges
Stability	Error amplification control	**Adaptability**	Self-organization in dynamic systems
Convergence and Accuracy	Theoretical error reduction	**Interpretability**	Integration with explainable AI (XAI)
Consistency	Parameter $(\Delta x, \Delta t)$ dependence	**Robustness to Noise**	Regularization of ill-posed problems
Efficiency	Computational complexity	**Scalability**	Exascale parallelization bottlenecks
		Energy Efficiency	Green computing framework design

We will first consider **traditional properties** of numerical methods with an example, then the lecture will address their **modern expansions** with key challenges.

The Gardener's Song for Data-First Numerical Methods
YouTube (Musical) (Melodic Rap)

Contents of Chapter 16

16.1. Parabolic PDEs in One Spatial Variable: Consistency 562
16.2. Convergence and Accuracy 567
16.3. Stability 572
16.4. The θ-Method: Accuracy and Efficiency 577
16.5. Data-First Numerical Methods 584
Exercises for Chapter 16 587

16.1. Parabolic PDEs in One Spatial Variable: Consistency

Let $\Omega = (0,1)$ and $J = (0,T]$, for some $T > 0$. Consider the following simplest **model problem** for parabolic equations in one-dimensional (1D) space:

$$\begin{aligned} u_t - u_{xx} &= f, \quad (x,t) \in \Omega \times J, \\ u &= 0, \qquad (x,t) \in \Gamma \times J, \\ u &= u^0, \qquad x \in \Omega, \ \ t = 0, \end{aligned} \tag{16.1}$$

where f is a heat source, Γ denotes the boundary of Ω, i.e., $\Gamma = \{0,1\}$, and u^0 is the **initial condition**, the prescribed initial value of the solution at $t = 0$.

Finite Difference Methods

- We begin with our discussion of **finite difference (FD) methods** for (16.1) by partitioning the domain. Let

$$\begin{aligned} &\Delta t = T/n_t, \quad t^n = n\Delta t, \ \ n = 0, 1, \cdots, n_t; \\ &\Delta x = 1/n_x, \quad x_j = j\Delta x, \ \ j = 0, 1, \cdots, n_x; \end{aligned}$$

  for some positive integers n_t and n_x. Define $u_j^n = u(x_j, t^n)$.

- Let

$$\mathcal{S}^n := \Omega \times (t^{n-1}, t^n] \tag{16.2}$$

  be the nth **space-time slice**.

- Suppose that the computation has been performed for

$$u^k = \{u_j^k\}, \quad 0 \le k \le n-1.$$

  Then, the task is to compute u^n by integrating the equation on the space-time slice $\mathcal{S}^n$, utilizing FD schemes.

- The basic idea of FD schemes is to replace derivatives by FD approximations. It can be done in various ways; here we consider most common ways that are based on the **Taylor's formula**.

Recall: **Approximation of $y''(x_i)$**, considered in (14.27), p. 519:

$$y''(x_i) = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} - 2\frac{y^{(4)}(x_i)}{4!}h^2 - 2\frac{y^{(6)}(x_i)}{6!}h^4 - \cdots, \tag{16.3}$$

where h is the mesh size.

Semi-Discrete Problem

- Apply (16.3) to have

$$u_{xx}(x_j, t^n) = \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} - 2\frac{u_{xxxx}(x_j, t^n)}{4!}\Delta x^2 + \mathcal{O}(\Delta x^4). \tag{16.4}$$

- Let

$$\mathcal{A}_1 u_j^n \overset{\text{def}}{=\!=} -\frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2}. \tag{16.5}$$

- **When $t = t^n$**, the differential equation $u_t - u_{xx} = f$ can then be written as

$$\mathbf{u}_t^n + \mathcal{A}_1 \mathbf{u}^n = \mathbf{f}^n + \mathcal{O}(\Delta x^2), \tag{16.6}$$

  where $\mathbf{u}^n = \{u_j^n\}$ and $\mathbf{f}^n = \{f_j^n\}$.
- The above equation is called the **semi-discrete equation** for (16.1), in which the spatial variable is discretized, while the time variable remains continuous.
- For the temporal direction, one can also apply a difference formula for the approximation of the time-derivative u_t.
  - Depending on the way of combining the spatial and temporal differences, the resulting scheme can behave quite differently.

The semi-discrete equation:

$$\mathbf{u}_t^n + \mathcal{A}_1 \mathbf{u}^n = \mathbf{f}^n + \mathcal{O}(\Delta x^2). \tag{16.7}$$

Explicit Scheme

- The following presents the simplest scheme:

$$\frac{\mathbf{u}^n - \mathbf{u}^{n-1}}{\Delta t} + \mathcal{A}_1 \mathbf{u}^{n-1} = \mathbf{f}^{n-1} + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2), \tag{16.8}$$

  of which the approximation at $x = x_j$ reads

$$\frac{v_j^n - v_j^{n-1}}{\Delta t} - \frac{v_{j-1}^{n-1} - 2v_j^{n-1} + v_{j+1}^{n-1}}{\Delta x^2} = f_j^{n-1}, \tag{16.9}$$

  which is an explicit scheme for (16.1), called the **forward Euler method**. Here v_j^n is an approximation of u_j^n.

- The above scheme in (16.9) can be rewritten as

$$v_j^n = \mu\, v_{j-1}^{n-1} + (1 - 2\mu)\, v_j^{n-1} + \mu\, v_{j+1}^{n-1} + \Delta t f_j^{n-1}, \tag{16.10}$$

  where

$$\mu = \frac{\Delta t}{\Delta x^2}. \tag{16.11}$$

  Note: The unknown value v_j^n is expressed **explicitly** as a linear combination of known values in the previous time level.

- Define the error

$$e_j^n = u_j^n - v_j^n. \tag{16.12}$$

  Then it follows from (16.8) and (16.9) that

$$\frac{e_j^n - e_j^{n-1}}{\Delta t} - \frac{e_{j-1}^{n-1} - 2e_j^{n-1} + e_{j+1}^{n-1}}{\Delta x^2} = \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2). \tag{16.13}$$

Now, we begin with the concept of consistency, for the numerical procedure (16.9).

Consistency

A simple requirement for an accurate numerical method is that ***the discretization becomes exact as the grid spacing tends to zero***, which is the basis of **consistency**.

Definition 16.1. Given a PDE $Pu = f$ and a FD scheme $P_{\Delta x,\Delta t} v = f$, the FD scheme is said to be consistent with the PDE if for every smooth function $\phi(x,t)$

$$P\phi - P_{\Delta x,\Delta t}\phi \to 0 \quad \text{as} \quad (\Delta x, \Delta t) \to 0, \tag{16.14}$$

with the convergence being pointwise at each grid point.

Not all numerical methods based on Taylor series expansions are consistent; sometimes, we may have to restrict the manner in which Δx and Δt approach zero in order for the numerical method to be consistent.

Example 16.2. The forward Euler scheme (16.9) is consistent.

Proof. For the heat equation in 1D,

$$P\phi \equiv \Big(\frac{\partial}{\partial t} - \frac{\partial^2}{\partial x^2}\Big)\phi = \phi_t - \phi_{xx}.$$

The forward Euler scheme (16.9) reads

$$P_{\Delta x,\Delta t}\phi = \frac{\phi_j^n - \phi_j^{n-1}}{\Delta t} - \frac{\phi_{j-1}^{n-1} - 2\phi_j^{n-1} + \phi_{j+1}^{n-1}}{\Delta x^2}.$$

Consider the Taylor's formulas

$$\begin{aligned}
\phi_{\mathbf{t}}(\mathbf{x_j}, \mathbf{t^{n-1}}) &= \frac{\phi_{\mathbf{j}}^{\mathbf{i}} - \phi_{\mathbf{j}}^{\mathbf{n-1}}}{\mathbf{\Delta t}} - \frac{\phi_{tt}(x_j, t^{n-1})}{2!}\Delta t + \mathcal{O}(\Delta t^2), \\
\phi_{\mathbf{xx}}(\mathbf{x_j}, \mathbf{t^{n-1}}) &= \frac{\phi_{\mathbf{j-1}}^{\mathbf{n-1}} - \mathbf{2}\phi_{\mathbf{j}}^{\mathbf{n-1}} + \phi_{\mathbf{j+1}}^{\mathbf{n-1}}}{\mathbf{\Delta x^2}} - 2\frac{\phi_{xxxx}(x_j, t^{n-1})}{4!}\Delta x^2 + \mathcal{O}(\Delta x^4),
\end{aligned} \tag{16.15}$$

from which the truncation error of the forward Euler scheme evaluated at (x_j, t^{n-1}) becomes

$$\begin{aligned}
&(\mathbf{P}\phi - \mathbf{P_{\Delta x,\Delta t}}\phi)\,(x_j, t^{n-1}) \\
&= -\frac{\phi_{tt}(x_j, t^{n-1})}{2!}\Delta t + 2\frac{\phi_{xxxx}(x_j, t^{n-1})}{4!}\Delta x^2 + \mathcal{O}(\Delta t^2 + \Delta x^4),
\end{aligned} \tag{16.16}$$

which clearly approaches zero, as $(\Delta x, \Delta t) \to 0$. □

Definition 16.3. Let u be smooth and

$$P\,u(x_j, t^n) \;=\; P_{\Delta x,\Delta t}\,u_j^n + \mathcal{T}u_j^n, \tag{16.17}$$

Then, $\mathcal{T}u_j^n$ is called the **truncation error** of the FD scheme $P_{\Delta x,\Delta t}v = f$ evaluated at (x_j, t^n).

Remark 16.4. The **consistency** says that the truncation error approaches zero, as $(\Delta x, \Delta t) \to 0$. That is,

$$(Pu - P_{\Delta x,\Delta t}v) \to 0 \quad \text{as} \quad (\Delta x, \Delta t) \to 0. \tag{16.18}$$

However this does not always imply that $(u - v) \to 0$.

We can say that the forward Euler scheme (16.9) is consistent, because its truncation error approaches zero as the discretization steps $(\Delta x, \Delta t)$ decrease. That is,

$$\mathcal{T}u_j^n = \mathcal{O}(\Delta t + \Delta x^2) \to 0, \quad \text{as} \quad (\Delta x, \Delta t) \to 0, \tag{16.19}$$

for all grid points (x_j, t^n).

Pizza Delivery: While **a pizza box** is delivered to the correct address (**consistency**), it doesn't guarantee that **the pizza inside** is the one ordered. This introduces the concept of **accuracy**.

- In **business**, **consistency** refers to the sustained and reliable adherence to a set of standards, strategies, or practices.
- In a **relationship**, **consistency** refers to predictable and dependable behavior that builds trust and security.

16.2. Convergence and Accuracy

A numerical method is said to be *convergent* if the solution of the FD scheme tends to the exact solution of the PDE as the grid spacing tends to zero. We define convergence in a formal way as follows:

Definition 16.5. A FD scheme approximating a PDE is said to be convergent if

$$u(x,t) - v_j^n \to 0, \text{ as } (x_j, t^n) \to (x,t) \text{ and } (\Delta x, \Delta t) \to 0,$$

where $u(x,t)$ is the exact solution of PDE and v_j^n denotes the the solution of the FD scheme.

Remark 16.6. **Consistency vs. Convergence**

- Consistency implies that the truncation error

$$(Pu - P_{\Delta x, \Delta t} u) \to 0, \text{ as } (\Delta x, \Delta t) \to 0.$$

- Generally, convergence implies consistency, but the reverse is not always true.

Pizza Delivery: **Increasing the number of people** involved in verifying the pizza order before delivery significantly **increases the chances of the correct pizza being delivered** (convergence & accuracy).

Theorem 16.7. *The forward Euler scheme (16.9) is convergent, when*

$$\mu = \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}. \tag{16.20}$$

Proof. (Error equation) We first rewrite (16.13):

$$\frac{e_j^n - e_j^{n-1}}{\Delta t} - \frac{e_{j-1}^{n-1} - 2e_j^{n-1} + e_{j+1}^{n-1}}{\Delta x^2} = -\mathcal{T} u_j^{n-1}, \tag{16.21}$$

where $\mathcal{T} u_j^{n-1}$ is the truncation error,

$$\mathcal{T} u_j^{n-1} = \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2).$$

Multiply both sides of (16.21) by Δt and rearrange to get

$$e_j^n \;=\; \mu\, e_{j-1}^{n-1} + (1-2\mu)\, e_j^{n-1} + \mu\, e_{j+1}^{n-1} - \Delta t\, \mathcal{T} u_j^{n-1}. \tag{16.22}$$

(Error analysis with ℓ_∞-norm) Define

$$\mathcal{E}^n = \max_j |e_j^n|, \quad \mathcal{T}^n = \max_j |\mathcal{T} u_j^n|, \quad \widehat{\mathcal{T}} = \max_n \mathcal{T}^n.$$

Note that $v_j^0 = u_j^0$ for all j and therefore $\mathcal{E}^0 = 0$.

It follows from (16.22) and the assumption (16.20) that

$$\begin{aligned} |e_j^n| \;&\le\; \mu\, |e_{j-1}^{n-1}| + (1-2\mu)\, |e_j^{n-1}| + \mu\, |e_{j+1}^{n-1}| + \Delta t\, |\mathcal{T} u_j^{n-1}| \\ &\le\; \mu\, \mathcal{E}^{n-1} + (1-2\mu)\, \mathcal{E}^{n-1} + \mu\, \mathcal{E}^{n-1} + \Delta t\, \mathcal{T}^{n-1} \\ &=\; \mathcal{E}^{n-1} + \Delta t\, \mathcal{T}^{n-1}. \end{aligned} \tag{16.23}$$

Since the above inequality holds for all j, we have

$$\mathcal{E}^n \le \mathcal{E}^{n-1} + \Delta t\, \mathcal{T}^{n-1}, \tag{16.24}$$

and therefore

$$\begin{aligned} \mathcal{E}^n \;&\le\; \mathcal{E}^{n-1} + \Delta t\, \mathcal{T}^{n-1} \le \mathcal{E}^{n-2} + \Delta t\, \mathcal{T}^{n-1} + \Delta t\, \mathcal{T}^{n-2} \\ &\le\; \cdots \le \mathcal{E}^0 + \textstyle\sum_{k=1}^{n-1} \Delta t\, \mathcal{T}^k. \end{aligned} \tag{16.25}$$

Since $\mathcal{E}^0 = 0$,

$$\mathcal{E}^n \le (n-1)\Delta t\, \widehat{\mathcal{T}} \le T\, \widehat{\mathcal{T}}, \tag{16.26}$$

where T is the upper bound of the time available. Since $\widehat{\mathcal{T}} = \mathcal{O}(\Delta t + \Delta x^2)$, the maximum norm of the error approaches zero as $(\Delta x, \Delta t) \to 0$. □

Remark 16.8.

- The assumption $\mu \leq 1/2$ makes coefficients in the forward Euler scheme (16.10) nonnegative, which in turn makes v_j^n a weighted average of $\{v_{j-1}^{n-1}, v_j^{n-1}, v_{j+1}^{n-1}\}$.
- The **convergence analysis** can often conclude

$$\mathcal{E}^n = \mathcal{O}(\widehat{\mathcal{T}}), \ \forall\, n,$$

  which is the **numerical error**. The convergence analysis is also called the **accuracy analysis**.

- Convergence/accuracy is what a numerical scheme must satisfy.
  - A numerical scheme is considered **convergent** if its solution approaches the exact solution as the discretization mesh is refined.
  - **Accuracy**, on the other hand, measures how close the numerical solution is to the true solution.
- **Showing convergence is not easy** in general, if attempted in a direct manner as in the previous theorem.
  - There is a related concept, **stability**, that is easier to check.

In computational terms, **convergence** is the property of the numerical solution to approach the exact or true solution as the mesh is made finer.

- **Archers can make more accurate shots through practice** because they develop muscle memory, refine their technique, and become more comfortable with their equipment.

Example 16.9. Solve the problem using the explicit time-stepping procedure:

$$\begin{aligned} u_t - u_{xx} = 0, &\quad (x,t) \in [0,1] \times [0,1], &\quad \text{(PDE)} \\ u = 0, &\quad (x,t) \in \{0,1\} \times [0,1], &\quad \text{(BC)} \\ u = \sin(\pi x), &\quad x \in [0,1], \ \ t = 0. &\quad \text{(IC)} \end{aligned} \tag{16.27}$$

The exact solution is $u(x,t) = e^{-\pi^2 t}\sin(\pi x)$.

Solution.

Heat_1D_Euler.m

```
close all; clear all

%% Problem setting
a=0; b=1; T=1;
u  = @(x,t) exp(-pi^2*t)*sin(pi*x);
u0 = @(x) sin(pi*x);

%% Initialization
nx = 10;
Dx = (b-a)/nx;
grid = linspace(a,b,nx+1);
v0 = u0(grid);

%% Exact solution @ t=1
u1 = u(grid,1);

%% Solve
for nt = [200,170]
    Dt = T/nt; mu = Dt/Dx^2;
    v1 = euler_march(v0,mu,nt);
    l8_rel_err = norm(u1-v1,inf)/max(u1);
    fprintf("nt=%3d; mu=%.2f; l8_rel_err = %g\n",nt,mu,l8_rel_err)
    figure,plot(grid,u1,'k-',grid,v1,'r--','linewidth',2)
    STR1 = strcat("Exact Solution: nx=",num2str(nx)," @ t=",num2str(T));
    STR2 = strcat("Euler Solution: nt=",num2str(nt));
    legend(STR1,STR2)
    filename = strcat("fig-Euler-Solution-nt=",num2str(nt),".png");
    exportgraphics(gca,filename,'Resolution',100)
end
```

euler_march.m

```
function v = euler_march(v0,mu,nt)

m = length(v0);
v = zeros(1,m);

for n =1:nt
    for j=2:m-1
        v(j) = mu*v0(j-1) + (1-2*mu)*v0(j) + mu*v0(j+1);
    end
    v0 = v;
end
```

OUTPUT

```
nt=200; mu=0.50; l8_rel_err = 0.153592
nt=170; mu=0.59; l8_rel_err = 1.78609e+06
```

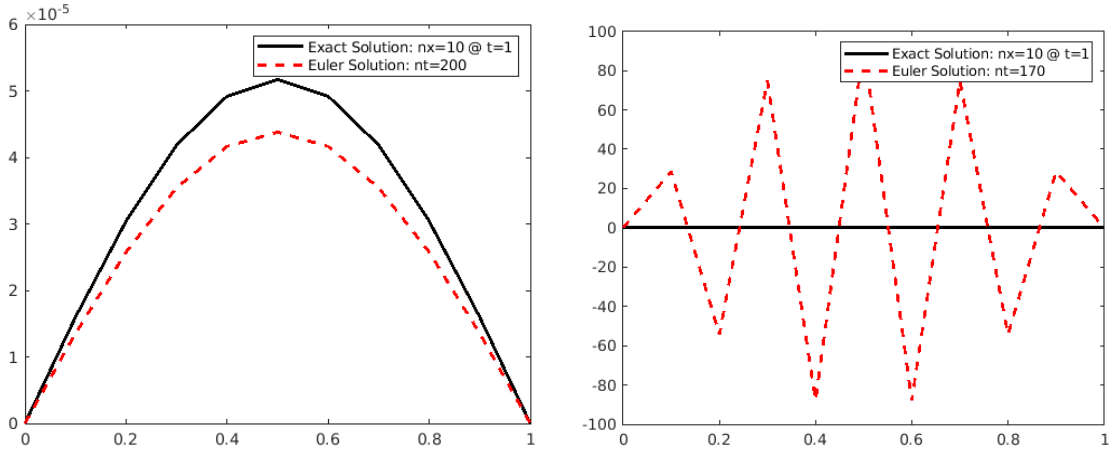


Figure 16.1: Euler solutions, with $n_x = 10$: $\mu = 0.5$ and $\mu = 0.59$.

Consistency of a numerical method is not enough to guarantee convergence of its solution to the exact solution.

Remark 16.10. The **stability condition** in (16.20),

$$\mu = \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}, \tag{16.28}$$

is not only a **sufficient condition** but also a **necessary condition**.

16.3. Stability

Example 16.9 shows that consistency of a numerical method is not enough to guarantee convergence of its solution to the exact solution.

- In order for a **consistent** numerical scheme to be **convergent**, a required property is **stability**.
- If a scheme is convergent, it produces a bounded solution whenever the exact solution is bounded.

This is the basis of **stability**.

We first define the L^2-norm of **grid function** v:

$$\|v\|_{\Delta x} = \left(\Delta x \sum_j |v_j|^2\right)^{1/2}. \tag{16.29}$$

Definition 16.11. A FD scheme $P_{\Delta x,\Delta t} v = 0$ for a homogeneous PDE $Pu = 0$ is **stable** if for any positive T, there is a constant C_T such that

$$\|v^n\|_{\Delta x} \le C_T \|u^0\|_{\Delta x}, \qquad \forall\ 0 \le t^n \le T, \tag{16.30}$$

for Δx and Δt sufficiently small. When the initial data is given over multiple time levels, the stability requires to satisfy

$$\|v^n\|_{\Delta x} \le C_T \sum_{m=0}^{M} \|u^m\|_{\Delta x}, \qquad \forall\ 0 \le t^n \le T. \tag{16.31}$$

Note: The concept of stability is considered for **homogeneous PDEs**.

Pizza Delivery generally follows a sequential process, from order placement to deliver completion, with **smooth transition and no abrupt change,** between each step (stability).

Pizza Logic in Numerical Methods YouTube (Melodic Rap)

Theorem 16.12. Lax-Richtmyer Equivalence Theorem
Given a well-posed linear initial value problem and its FD approximation that satisfies the *consistency* condition, ***stability is a necessary and sufficient condition for convergence.***

The above theorem is very useful and important.

- Proving convergence is difficult for most problems.
- However, the determination of consistency of a scheme is quite easy as shown in Example 16.2, and determining stability is also easier than showing convergence.

Theorem 16.13. *A FD scheme $P_{\Delta x,\Delta t}v = 0$ for a homogeneous PDE $Pu = 0$ is stable if*

$$\|v^n\|_{\Delta x} \le (1 + C\Delta t)\|v^{n-1}\|_{\Delta x}, \tag{16.32}$$

for some $C \ge 0$ independent of Δt

Proof. Recall $\Delta t = T/n_t$, for some positive integer n_t. A recursive application of (16.32) reads

$$\begin{aligned} \|v^n\|_{\Delta x} &\le (1 + C\Delta t)\|v^{n-1}\|_{\Delta x} \le (1 + C\Delta t)^2\|v^{n-2}\|_{\Delta x} \\ &\le \cdots \le (1 + C\Delta t)^n\|v^0 (= u^0)\|_{\Delta x}. \end{aligned} \tag{16.33}$$

Here the task is to show $(1 + C\Delta t)^n$ is bounded by some positive number C_T for $n = 1, \cdots, n_t$, independently on Δt. Since $\Delta t = T/n_t$, we have

$$\begin{aligned} (1 + C\Delta t)^n &= (1 + CT/n_t)^n \\ &\le (1 + CT/n_t)^{n_t} = \left[(1 + CT/n_t)^{n_t/CT}\right]^{CT} \\ &\le e^{CT}, \end{aligned}$$

which proves (16.30) with by $C_T := e^{CT}$. □

Approaches for Proving Stability

There are two fundamental approaches for proving stability:

- **The Fourier analysis (von Neumann analysis)**
  It applies only to linear constant coefficient problems.
- **The energy method**
  It can be used for more general problems with variable coefficients and nonlinear terms. But it is quite complicated and the proof is problem dependent.

The von Neumann Analysis

- Let ϕ be a grid function defined on grid points of spacing Δx and $\phi_j = \phi(j\Delta x)$. Then, its Fourier transform is given by, for $\xi \in [-\pi/\Delta x, \pi/\Delta x]$,

$$\widehat{\phi}(\xi) = \frac{1}{\sqrt{2\pi}} \sum_{j=-\infty}^{\infty} e^{-ij\Delta x\xi}\, \phi_j, \tag{16.34}$$

  and the inverse formula is

$$\phi_j = \frac{1}{\sqrt{2\pi}} \int_{-\pi/\Delta x}^{\pi/\Delta x} e^{ij\Delta x\xi}\, \widehat{\phi}(\xi) d\xi. \tag{16.35}$$

- **Parseval's identity**

$$\|\phi^n\|_{\Delta x} = \|\widehat{\phi^n}\|_{\Delta x}, \tag{16.36}$$

  where

$$\|\phi^n\|_{\Delta x} = \Big(\sum_{j=-\infty}^{\infty} |\phi_j|^2 \Delta x \Big)^{1/2}, \quad \|\widehat{\phi^n}\|_{\Delta x} = \Big(\int_{-\pi/\Delta x}^{\pi/\Delta x} |\widehat{\phi}(\xi)|^2 d\xi \Big)^{1/2}$$

- The stability inequality (16.30) can be replaced by

$$\|\widehat{v^n}\|_{\Delta x} \le C_T \sum_{m=0}^{M} \|\widehat{v^m}\|_{\Delta x}. \tag{16.37}$$

- Thus stability can be determined by providing (16.37) in the frequency domain.

Example 16.14. To show how one can use the above analysis, we exemplify the forward Euler scheme (16.10), with $f = 0$:

$$v_j^n = \mu\, v_{j-1}^{n-1} + (1-2\mu)\, v_j^{n-1} + \mu\, v_{j+1}^{n-1}. \tag{16.38}$$

- The inversion formula implies

$$v_j^n = \frac{1}{\sqrt{2\pi}} \int_{-\pi/\Delta x}^{\pi/\Delta x} e^{ij\Delta x\xi}\, \widehat{v}^n(\xi)\, d\xi. \tag{16.39}$$

Thus it follows from (16.38) and (16.39) that

$$v_j^n = \frac{1}{\sqrt{2\pi}} \int_{-\pi/\Delta x}^{\pi/\Delta x} \mathcal{F}_{\Delta x,j}(\xi)\, d\xi, \tag{16.40}$$

where

$$\begin{aligned} \mathcal{F}_{\Delta x,j}(\xi) &= \mu e^{i(j-1)\Delta x\xi}\, \widehat{v}^{n-1}(\xi) + (1-2\mu) e^{ij\Delta x\xi}\, \widehat{v}^{n-1}(\xi) + \mu e^{i(j+1)\Delta x\xi}\, \widehat{v}^{n-1}(\xi) \\ &= e^{ij\Delta x\xi} \left[\mu\, e^{-i\Delta x\xi} + (1-2\mu) + \mu\, e^{i\Delta x\xi}\right] \widehat{v}^{n-1}(\xi). \end{aligned}$$

- Comparing (16.39) with (16.40), we obtain

$$\widehat{v}^n(\xi) = \left[\mu\, e^{-i\Delta x\xi} + (1-2\mu) + \mu\, e^{i\Delta x\xi}\right] \widehat{v}^{n-1}(\xi). \tag{16.41}$$

- Letting $\vartheta = \Delta x\xi$, we define the **amplification factor** for the scheme (16.10) by

$$\begin{aligned} g(\vartheta) &= \mu\, e^{-i\Delta x\xi} + (1-2\mu) + \mu\, e^{i\Delta x\xi} \\ &= \mu\, e^{-i\vartheta} + (1-2\mu) + \mu\, e^{i\vartheta} \;=\; (1-2\mu) + 2\mu\cos(\vartheta) \\ &= 1 - 2\mu(1-\cos(\vartheta)) = 1 - 4\mu \sin^2(\vartheta/2). \end{aligned} \tag{16.42}$$

- Equation (16.41) can be rewritten as

$$\widehat{v}^n(\xi) = g(\vartheta)\, \widehat{v}^{n-1}(\xi) = g(\vartheta)^2\, \widehat{v}^{n-2}(\xi) = \cdots = g(\vartheta)^n\, \widehat{v}^0(\xi). \tag{16.43}$$

Therefore, when $g(\vartheta)^n$ is suitably bounded, the scheme is stable. In fact, **$g(\vartheta)^n$ would be uniformly bounded only if $|g(\vartheta)| \le 1 + C\Delta t$.**

- It is not difficult to see

$$|g(\vartheta)| = |1 - 2\mu(1-\cos(\vartheta))| \le 1$$

only if

$$0 \le \mu \le 1/2. \tag{16.44}$$

which is the **stability condition** of the scheme (16.10).

The von Neumann analysis: Is it Complicated?

A simpler and equivalent procedure of the von Neumann analysis can be summarized as follows:

- Replace v_j^n by $g^n e^{ij\vartheta}$ for each value of j and n.
- Find conditions on coefficients and grid spacings which would satisfy $|g| \leq 1 + C\Delta t$, for some $C \geq 0$.

Example 16.15. Use the von Neumann analysis to analyze the stability of the forward Euler scheme (16.10).

Solution. Rewrite the scheme:

$$v_j^n \;=\; \mu\, v_{j-1}^{n-1} + (1-2\mu)\, v_j^{n-1} + \mu\, v_{j+1}^{n-1}.$$

Replacing v_j^n with $g^n e^{ij\vartheta}$ gives

$$g^n e^{ij\vartheta} = \mu\, g^{n-1} e^{i(j-1)\vartheta} + (1-2\mu)\, g^{n-1} e^{ij\vartheta} + \mu\, g^{n-1} e^{i(j+1)\vartheta}.$$

Dividing both sides of the above by $g^{n-1}e^{ij\vartheta}$, we obtain

$$g = \mu\, e^{-i\vartheta} + (1-2\mu) + \mu\, e^{i\vartheta} = \cdots = 1 - 4\mu \sin^2(\vartheta/2),$$

which is exactly the same as in (16.42). □

In computation and systems, **stability** refers to a system's ability to maintain its desired behavior despite small errors or perturbations.

- Essentially, a stable system will not amplify or magnify these errors, but rather keep them within acceptable bounds.

Stability in life refers to a state of balance and predictability, characterized by consistent behavior, reasonable mood swings, and well-thought-out decisions.

16.4. The θ-Method: Accuracy and Efficiency

Recall: **The Semi-Discrete Problem**

- The model problem is defined in (16.1):

$$\begin{aligned} u_t - u_{xx} &= f, && (x,t) \in \Omega \times J, \\ u &= 0, && (x,t) \in \Gamma \times J, \\ u &= u^0, && x \in \Omega, \ t = 0. \end{aligned} \tag{16.45}$$

- The spatial discretization, (16.4):

$$u_{xx}(x_j,t^n) = \frac{u_{j-1}^n - 2u_j^n + u_{j+1}^n}{\Delta x^2} - 2\frac{u_{xxxx}(x_j,t^n)}{4!}\Delta x^2 + \mathcal{O}(\Delta x^4). \tag{16.46}$$

- **Semi-Discrete Equation**: **When $t = t^n$**, the differential equation can then be written as in (16.6):

$$\mathbf{u}_t^n + \mathcal{A}_1\mathbf{u}^n = \mathbf{f}^n + \mathcal{O}(\Delta x^2), \tag{16.47}$$

  where $\mathbf{u}^n = \{u_j^n\}$ and $\mathbf{f}^n = \{f_j^n\}$.

- The **forward Euler method** is an **explicit scheme**:

$$\frac{\mathbf{u}^n - \mathbf{u}^{n-1}}{\Delta t} + \mathcal{A}_1\mathbf{u}^{n-1} = \mathbf{f}^{n-1} + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2), \tag{16.48}$$

Observation 16.16. The temporal discretization can be carried out in various ways. For example,

$$\begin{aligned} \frac{\mathbf{u}^n - \mathbf{u}^{n-1}}{\Delta t} + \mathcal{A}_1\mathbf{u}^n &= \mathbf{f}^n && + \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2), \\ \frac{\mathbf{u}^n - \mathbf{u}^{n-1}}{\Delta t} + \mathcal{A}_1\mathbf{u}^{n-1/2} &= \mathbf{f}^{n-1/2} && + \mathcal{O}(\mathbf{\Delta t^2}) + \mathcal{O}(\Delta x^2). \end{aligned} \tag{16.49}$$

The first scheme involves a backward discretization of u_t, while the second one is a central temporal discretization.

- In practice, we approximate the term $\mathbf{u}^{n-1/2}$ by

$$\mathbf{u}^{n-1/2} = \frac{1}{2}(\mathbf{u}^n + \mathbf{u}^{n-1}) + \mathcal{O}(\Delta t^2). \tag{16.50}$$

Definition 16.17. The θ-Method

Given the observation, we define the **θ-method** as

$$\frac{\mathbf{v}^n - \mathbf{v}^{n-1}}{\Delta t} + \mathcal{A}_1\left[\theta \mathbf{v}^n + (1-\theta)\mathbf{v}^{n-1}\right] = \mathbf{f}^{n-1+\theta}, \quad \theta \in [0,1], \tag{16.51}$$

or equivalently

$$(I + \theta \Delta t \mathcal{A}_1)\mathbf{v}^n = \left[I - (1-\theta)\Delta t \mathcal{A}_1\right]\mathbf{v}^{n-1} + \Delta t\, \mathbf{f}^{n-1+\theta}. \tag{16.52}$$

The following three choices of θ are popular.

- **The forward Euler method ($\theta = 0$)**: The algorithm (16.52) is reduced to

$$\mathbf{v}^n = (I - \Delta t \mathcal{A}_1)\mathbf{v}^{n-1} + \Delta t\, \mathbf{f}^{n-1}, \tag{16.53}$$

  which is the explicit scheme in (16.10), requiring the stability condition

$$\mu = \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2}.$$

- **The backward Euler method ($\theta = 1$)**: This is an implicit method written as

$$(I + \Delta t \mathcal{A}_1)\mathbf{v}^n = \mathbf{v}^{n-1} + \Delta t\, \mathbf{f}^n. \tag{16.54}$$

  - The method must invert a tridiagonal matrix to get the solution in each time level.
  - But it is **unconditionally stable**, stable independently of the choice of Δt.

- **The Crank-Nicolson method ($\theta = 1/2$)**:

$$\left(I + \frac{\Delta t}{2}\mathcal{A}_1\right)\mathbf{v}^n = \left(I - \frac{\Delta t}{2}\mathcal{A}_1\right)\mathbf{v}^{n-1} + \Delta t\, \mathbf{f}^{n-1/2}. \tag{16.55}$$

  - It requires to solve a tridiagonal system in each time level, as in the backward Euler method.
  - However, the Crank-Nicolson method is most popular, because it is second-order in both space and time and unconditionally stable.

– The Crank-Nicolson method can be viewed as an explicit method in the first half of the space-time slice $\mathcal{S}^n (:= \Omega \times (t^{n-1}, t^n])$ and an implicit method in the second half of $\mathcal{S}^n$. Hence it is often called a **semi-implicit method**.

Stability Analysis for the θ-Method

- Setting $f \equiv 0$, the algebraic system (16.52) reads pointwisely

$$\begin{aligned} &-\theta\mu\, v_{j-1}^n + (1+2\theta\mu)v_j^n - \theta\mu\, v_{j+1}^n \\ &\quad = (1-\theta)\mu\, v_{j-1}^{n-1} + [1-2(1-\theta)\mu]v_j^{n-1} + (1-\theta)\mu\, v_{j+1}^{n-1}, \end{aligned} \tag{16.56}$$

  where $\mu = \Delta t/\Delta x^2$.

- For an stability analysis, utilizing the von Neumann analysis, we substitute $g^n e^{ij\vartheta}$ for v_j^n in (16.56) to have

$$\begin{aligned} &g\left[-\theta\mu\, e^{-i\vartheta} + (1+2\theta\mu) - \theta\mu\, e^{i\vartheta}\right] \\ &\quad = (1-\theta)\mu\, e^{-i\vartheta} + [1-2(1-\theta)\mu] + (1-\theta)\mu\, e^{i\vartheta}. \end{aligned}$$

  That is,

$$\begin{aligned} g &= \frac{1-2(1-\theta)\mu\,(1-\cos\vartheta)}{1+2\theta\mu\,(1-\cos\vartheta)} \\ &= \frac{1-4(1-\theta)\mu\,\sin^2(\vartheta/2)}{1+4\theta\mu\,\sin^2(\vartheta/2)} \ (\le 1). \end{aligned} \tag{16.57}$$

  Because $\mu > 0$ and $\theta \in [0,1]$, the amplification factor g cannot be larger than one.

- The condition $g \ge -1$ is equivalent to

$$1 - 4(1-\theta)\mu\,\sin^2(\vartheta/2) \ \ge\ -\left[1 + 4\theta\mu\,\sin^2(\vartheta/2)\right],$$

  or

$$(1-2\theta)\mu\,\sin^2\frac{\vartheta}{2} \le \frac{1}{2}.$$

  Thus the θ-method (16.52) is stable if

$$(1-2\theta)\mu \le \frac{1}{2}. \tag{16.58}$$

Summary **16.18.** In conclusion:

- The θ-method is unconditionally stable for $\theta \geq 1/2$.
- When $\theta < 1/2$, the method is stable only if

$$\mu = \frac{\Delta t}{\Delta x^2} \leq \frac{1}{2(1-2\theta)}, \quad \theta \in [0, 1/2). \tag{16.59}$$

  Recall: The forward Euler method ($\theta = 0$) is stable only if $\mu \leq 1/2$.
- The Crank-Nicolson method ($\theta = 1/2$) is most popular, because it is **second-order** in both spatial and temporal directions, and **unconditionally stable**.

Example **16.19. (A Variant of Example 16.9)** Consider the problem

$$\begin{aligned} u_t - u_{xx} &= (1+\pi^2)e^t \sin(\pi x), && (x,t) \in [0,1] \times [0,1], && \text{(PDE)} \\ u &= 0, && (x,t) \in \{0,1\} \times [0,1], && \text{(BC)} \\ u &= \sin(\pi x), && x \in [0,1], \ \ t = 0. && \text{(IC)} \end{aligned} \tag{16.60}$$

The exact solution is $u(x,t) = e^t \sin(\pi x)$.

(a) Solve the problem using the θ-method, with $\theta = 0,\ 1,\ 1/2$.

(b) Compare their **accuracy and efficiency.**

Solution.

Heat_1D_Solve.m

```
close all; clear all

%% Problem setting
a=0; b=1; T=1;
u  = @(x,t) exp(t)*sin(pi*x);
f  = @(x,t) (1+pi^2)*exp(t)*sin(pi*x);
u0 = @(x) sin(pi*x);

%% Initialization
nx = 10;
Dx = (b-a)/nx;
grid = linspace(a,b,nx+1);
v0 = u0(grid);
nt_forward = round(2/Dx^2);

%% Exact solution @ t=1
```

```
u1 = u(grid,1);

%% Solve
for theta = [0, 1, 1/2]
    if theta==0, nt=nt_forward; else, nt=2*nx; end
    Dt = T/nt; mu = Dt/Dx^2;
    v1 = theta_march(v0,theta,mu,T,nt,f,grid);
    l8_rel_err = norm(u1-v1,inf)/max(u1);
    fprintf("nx=%-3d; nt=%-4d; theta=%.2f; mu=%.2f; l8_rel_err = %.6f\n",...
             nx,nt,theta,mu,l8_rel_err)
    figure,plot(grid,u1,'k-',grid,v1,'r--','linewidth',2)
    STR1= strcat("Exact-Solution: nx=",num2str(nx)," @ t=",num2str(T));
    STR2= strcat("Theta-Method:  nt=",num2str(nt),"; theta=",num2str(theta));
    legend(STR1,STR2,'location','best')
    if nx==10
        filename = strcat("fig-Theta-Method-nx=",num2str(nx),...
                          "-theta=",num2str(theta),".png");
        exportgraphics(gca,filename,'Resolution',100)
    end
end
```

theta_march.m

```
function v = theta_march(v0,theta,mu,T,nt,f,grid)

m = length(v0);

B = zeros(m,m);
b1=theta*mu; b0=1+2*b1;
for i=2:m-1, B(i,i-1:i+1) = [-b1,b0,-b1]; end
B(1,1)=1; B(m,m)=1;
f1=(1-theta)*mu; f0=1-2*f1;

Dt=T/nt;
v = zeros(1,m);
for n =1:nt
    % forward step
    for j=2:m-1,
        v(j) = f1*v0(j-1) + f0*v0(j) + f1*v0(j+1);
    end
    v = v + Dt*f(grid,(n-1+theta)*Dt); %the source "f"
    % backward step
    v0 = B\v';
end
v = v0';
```

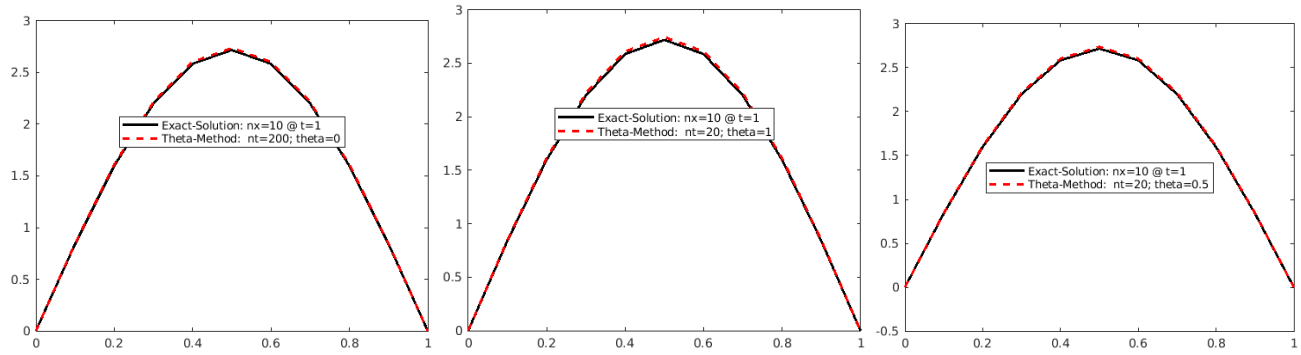


Figure 16.2: Solutions at $t = 1$, $n_x = 10$: $\theta = 0,\ 1,\ 1/2$. For the forward Euler, n_t is chosen to satisfy $\nu = 1/2$; for others, $n_t = 2 * n_x$.

OUTPUT

```
>> Heat_1D_Solve
nx=10 ; nt=200 ; theta=0.00; mu=0.50;  l8_rel_err = 0.007265
nx=10 ; nt=20  ; theta=1.00; mu=5.00;  l8_rel_err = 0.009799
nx=10 ; nt=20  ; theta=0.50; mu=5.00;  l8_rel_err = 0.007204
>> Heat_1D_Solve
nx=20 ; nt=800 ; theta=0.00; mu=0.50;  l8_rel_err = 0.001811
nx=20 ; nt=40  ; theta=1.00; mu=10.00; l8_rel_err = 0.003015
nx=20 ; nt=40  ; theta=0.50; mu=10.00; l8_rel_err = 0.001795
>> Heat_1D_Solve
nx=40 ; nt=3200; theta=0.00; mu=0.50;  l8_rel_err = 0.000452
nx=40 ; nt=80  ; theta=1.00; mu=20.00; l8_rel_err = 0.001040
nx=40 ; nt=80  ; theta=0.50; mu=20.00; l8_rel_err = 0.000449
```

OUTPUT

```
>> Heat_1D_Solve2
Forward-Euler:
nx=80 ; nt=12800; theta=0.00; mu=0.50; l8_rel_err = 0.000113
Backward-Euler:
nx=80 ; nt=10  ; theta=1.00; mu=640.00; l8_rel_err = 0.004586
nx=80 ; nt=20  ; theta=1.00; mu=320.00; l8_rel_err = 0.002384
nx=80 ; nt=40  ; theta=1.00; mu=160.00; l8_rel_err = 0.001259
Crank-Nicolson:
nx=80 ; nt=10  ; theta=0.50; mu=640.00; l8_rel_err = 0.001056
nx=80 ; nt=20  ; theta=0.50; mu=320.00; l8_rel_err = 0.000177
nx=80 ; nt=40  ; theta=0.50; mu=160.00; l8_rel_err = 0.000043
```

Remark 16.20. **The θ-Method**: For this example,

- For $\theta = 0.0$, error $= \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2)$.
- For $\theta = 1.0$, error $\approx \mathcal{O}(\Delta t) + \mathcal{O}(\Delta x^2)$
- For $\theta = 0.5$, error $= \mathcal{O}(\Delta t^2) + \mathcal{O}(\Delta x^2)$.

The **Crank-Nicolson method** is **most accurate and efficient**.

Pizza Delivery: **The core of effective pizza delivery** lies **in balancing efficiency** (delivering multiple pizzas to similar addresses) **with accuracy** (verifying those addresses).

- This clearly means optimizing routes to minimize travel time while ensuring deliveries reach the intended recipients.

16.5. Data-First Numerical Methods

Traditional numerical methods exhibit several important properties: consistency, stability, convergence, accuracy, and efficiency.

- **Consistency** means the numerical method approximates the correct solution as the step size approaches zero.
- **Stability** ensures errors don't accumulate uncontrollably.
- **Convergence** means the solution approaches the true solution as the step size decreases.
- **Accuracy** refers to how close the numerical solution is to the true solution.
- **Efficiency** measures how quickly the solution is obtained.

Those concepts have been explained, exemplifying **Pizza Delivery.** Let's consider two more **Real-life Examples**.

Numer.Alg	**Self-Improvement**	**Business**
Stability	Habit systems	Financial resilience
Convergence	Continuous learning	Customer feedback cycles
Accuracy	Strength-opportunity fit	Market-need alignment
Efficiency	Prioritization & focus (80/20)	Resources optimization

Life Lessons from Numerical Algorithms:

- Iterative refinement
- Recognizing limitations
- Adapting to changing conditions
- Thinking step-by-step
- The value of approximation

Modern Paradigm Shifts

- **Reversal of Scientific Methodology** (**data-first discovery**)
  - **Traditional: deductive approach**, which starts with a hypothesis/theory and tests it against data.
  - **Modern: inductive reasoning**, which prioritizes observing the world and then constructing theories to explain it.

    (a) Data Collection
    (b) Pattern Recognition
    (c) Theory Development
    (d) Further Testing

  Ex. Protein structure prediction of **AlphaFold** (Google DeepMind)
  $\Rightarrow$ surpass previous methods & enable significant advancements
- **Redefined Proof** (Rigorosity $\rightarrow$ Feasibility)
  - Traditional numerical analysis: **Theory-first**
  - **Data-first discovery**: discover by analyzing vast amounts of data

  In modern science, **the idea of definitive "proof"** is largely replaced by **the concept of the strength of evidence**. Rather than seeking absolute certainty, scientists focus on how well a theory is supported by empirical evidence.
- **Revolution in Tools** (Calculus $\rightarrow$ Algorithms)
  In modern science, algorithms are increasingly valued for their practical applications and ability to solve real-world problems, often overshadowing the beauty of theoretical concepts.
- **Traditional numerical methods** rely on known governing equations for solving problems, while **modern approaches**, particularly in **physics-informed machine learning** (**PIML**), leverage observed data and partial knowledge of the underlying physics to construct consistent solvers.

The world encounters an era of **great transition**.

Data-First Numerical Analysis

New View on Numerical Methods

Traditional numerical methods have been much like **lighthouses** guiding ships through a stormy sea.

- These methods are increasingly used to analyze and interpret large datasets, offering structured approaches to problem-solving that would otherwise be difficult or impossible to manage.

Paradigm Shift: **Observation** → **Inference** → **Theory & Testing**

(a) Data-driven legal reasoning, Inductive reasoning (AI)

(b) Theory development (Mathematics)

(c) **Hybrid Verification** (Uncertainty Quantification)

In the modern era, numerical methods become **tasks of constructing lighthouses on stormy seas.**

Table 16.2: Traditional vs. modern properties of numerical methods, with key challenges.

Property	Traditional Focus	Modern Expansion	Key Challenges
Stability	Error amplification control	**Adaptability**	Self-organization in dynamic systems
Convergence and Accuracy	Theoretical error reduction	**Interpretability**	Integration with explainable AI (XAI)
Consistency	Parameter $(\Delta x, \Delta t)$ dependence	**Robustness to Noise**	Regularization of ill-posed problems
Efficiency	Computational complexity	**Scalability**	Exascale parallelization bottlenecks
		Energy Efficiency	Green computing framework design

Life Lessons from Data-First Numerical Analysis

- Stability is not stagnation, but **adaptability.**
- Error is not failure, but **feedback.**
- Numerical algorithms are not merely for numbers but for **insights.**
- **A new version of mindset makes a new version of you.**

The Gardener's Song for Data-First Numerical Methods YouTube (Musical) (Melodic Rap)

Exercises for Chapter 16

16.1. The energy method can be utilized to prove stability of the forward Euler scheme for $u_t - u_{xx} = 0$:

$$v_j^n = \mu\, v_{j-1}^{n-1} + (1 - 2\mu)\, v_j^{n-1} + \mu\, v_{j+1}^{n-1} \tag{16.61}$$

The analysis requires you to prove

$$\|v^n\|_{\Delta x}^2 \le (1 + C\Delta t)^2 \|v^{n-1}\|_{\Delta x}^2, \tag{16.62}$$

for some $C \ge 0$. Prove it, assuming $1 - 2\mu \ge 0$ and using the following hint.

- Start with squaring (16.61).
- Apply the inequality $|ab| \le \dfrac{a^2 + b^2}{2}$.
- Use the observation

$$\sum_j |v_{j-1}^{n-1}|^2 = \sum_j |v_j^{n-1}|^2 = \sum_j |v_{j+1}^{n-1}|^2.$$

16.2. This problem shows a way of boundedness of the numerical solution of FD methods. Prove that the solution of the forward Euler method (16.9) satisfies

$$\min_j v_j^{n-1} \le v_j^n \le \max_j v_j^{n-1} \tag{16.63}$$

when $f \equiv 0$ and $\mu \le 1/2$. In this case, the numerical algorithm is said to hold the **maximum principle**.

16.3. C Consider Example 16.19.

(a) Copy-and-paste the code in the example, and modify as required.
(b) Set `nx = 100`.
(c) Set `theta = 1`. Then, for each of `nt = 10, 20, 40`, compute the numerical error and find the convergence order.
(d) Set `theta = 1/2`. Then, for each of `nt = 10, 20, 40`, compute the numerical error and find the convergence order.
(e) Finally, most importantly, explain why you are performing such experiments. What is the purpose of the analysis?

APPENDIX **P**

Projects

Finally we add projects.

Contents of Projects

P.1. Applications of Richardson Extrapolation for Elliptic PDEs 590
P.2. Richardson Extrapolation for Evolutionary PDEs 598
P.3. Project Regression Analysis: Linear, Piecewise Linear, and Nonlinear Models 602
P.4. Quasi-Newton Method Using Partial Information of the Hessian 611

P.1. Applications of Richardson Extrapolation for Elliptic PDEs

Preliminary **P.1.** The **Richardson extrapolation** studied in Section 4.2 is a numerical procedure that produces **a numerical solution of a higher-order accuracy**, when numerical solutions are available on two meshes: a mesh and its refined one.
By applying extrapolation recursively, the Richardson extrapolation becomes a **sequence acceleration method** that improves the rate of convergence of a sequence.

Applications of Richardson Extrapolation for PDEs

- **Issue**: The Richardson extrapolation can compute the solution of a higher-order accuracy **on the course mesh only**.
- **Remedy**: Interpolation
  - *Completed Richardson extrapolation* (CR) (Roache-Knupp, 1993) [29]
  - *Operator-based iterative interpolation* (OBI) (Wang-Zhang, 2009) [38]
  - *Multiple coarse grid extrapolation* (MCG) (Dai-Zhang-Wang, 2014) [8]

  CR is the most simple, effective, and popular (Dai-Lin,2018) [7].
  - It is based on a simple assumption that the correction term from the low-order to the high-order is smooth over the whole domain. More precisely, **the correction term is assumed to be continuous and locally linear.** In the method, the correction term is interpolated *locally linearly* over the fine mesh and then added to the fine-mesh solution to get a higher-order solution.
- The extrapolated solutions often show a desired order of accuracy, when they compared with each other. **However, their error can be larger than that of the numerical solution obtained using the *original* high-order scheme.**

Quesiton. Can we get higher-order solutions on finer meshes, better than the CR.

Note: For simplicity, we will focus on **1D boundary-value problems**.

Model-Problem P.2. Consider

$$\begin{aligned} -u''(x) &= \pi^2 \sin(\pi x), \quad x \in (0,1) \\ u(0) &= 0, \quad u(1) = 2, \end{aligned} \tag{P.1.1}$$

for which the exact solution is $u(x) = \sin(\pi x) + 2x$.

(a) Partition $[0,1]$ into n equal subintervals. That is,

$$h = \frac{1}{n}; \quad x_i = i \cdot h, \;\; i = 0, 1, \cdots, n.$$

Let $\mathbf{u}_h$ be such that $\mathbf{u}_h[i] = u(x_i)$.

(b) Approximate $-u''(x_i)$ using the formula: For each $i = 1, 2, \cdots, n-1$,

$$-u''(x_i) = \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} + K_2 h^2 + K_4 h^4 + \cdots = f_i, \tag{P.1.2}$$

where

$$K_j = 2 \cdot \frac{u^{(j+2)}(x_i)}{(j+2)!}, \quad j = 2, 4, \cdots. \tag{P.1.3}$$

(c) Assemble the above for an algebraic system to solve:

$$A_h \mathbf{v}_h = \mathbf{b}_h \tag{P.1.4}$$

Remark P.3. Model-Problem P.2: Assume we got $\mathbf{v}_h$, $\mathbf{v}_{h/2}$, and $\mathbf{v}_{h/4}$.

- **Richardson extrapolation**:
  - Solutions of a *fourth-order accuracy* can be obtained on the meshes of grid sizes h and $h/2$.
  - Solutions of a *sixth-order accuracy* can be computed **only on the coarsest mesh** of grid size h.
- By applying interpolation methods, we can get higher-order solutions on finer meshes.

Algorithm P.4. Completed Richardson extrapolation (CR) [29]

Let Ω_h be the set of (uniform) grid points.

1. *Solve for second-order numerical solutions*, for grid sizes h and $h/2$:

$$A_h \mathbf{v}_h = \mathbf{b}_h \quad \text{and} \quad A_{h/2}\mathbf{v}_{h/2} = \mathbf{b}_{h/2}. \tag{P.1.5}$$

2. *The fourth-order solution of Richardson extrapolation*:

$$\mathbf{v}^{(4)}_{h/2}\big|_{\Omega_h} = \frac{1}{3}\big(4\mathbf{v}_{h/2}\big|_{\Omega_h} - \mathbf{v}_h\big), \tag{P.1.6}$$

   where values are available only on the coarse mesh.

3. The *correction term* is defined as $\mathbf{c} \equiv \mathbf{v}^{(4)}_{h/2} - \mathbf{v}_{h/2}$; its available values are

$$\mathbf{c}\big|_{\Omega_h} = \big(\mathbf{v}^{(4)}_{h/2} - \mathbf{v}_{h/2}\big)\big|_{\Omega_h} = \frac{1}{3}\big(\mathbf{v}_{h/2}\big|_{\Omega_h} - \mathbf{v}_h\big). \tag{P.1.7}$$

4. *Interpolate the correction* over the whole fine mesh $\Omega_{h/2}$:

$$c_i = \frac{c_{i-1} + c_{i+1}}{2}, \tag{P.1.8}$$

   when c_{i-1} and c_{i+1} are given.
   Note: (P.1.8) is the simple *linear interpolation*.

5. *Obtain a fourth-order numerical solution*:

$$\mathbf{v}^{(4)}_{h/2} = \mathbf{v}_{h/2} + \mathbf{c}. \tag{P.1.9}$$

Remark P.5. The CR Algorithm

- It can be easily applied for boundary-value problems on 2D or 3D.
- It involves the simple **linear interpolation**. Its recursive application may not work, e.g. to get sixth-order solutions. You may apply higher-order interpolation methods to get more accurate solutions.
- A **modelcode** is given on page 595, for your convenience.

Challenge **P.6.** Solutions of a *sixth-order accuracy* on finer meshes.

Let us consider **a specific strategy** for a high-order interpolation.

Strategy **P.7.** **Method of Modified Equation**

- Equation (P.1.2) can be rewritten as

$$u_i = \frac{u_{i-1} + u_{i+1}}{2} + \frac{h^2}{2}(f_i - K_2 h^2) - \frac{1}{2} K_4 h^6 - \cdots, \tag{P.1.10}$$

  where

$$K_2 = \frac{u^{(4)}(x_i)}{12} \quad \text{and} \quad K_4 = \frac{u^{(6)}(x_i)}{360}. \tag{P.1.11}$$

- **Since $u^{(4)} = -f''$**, for example, K_2 is approximated as

$$K_2 = \frac{u^{(4)}(x_i)}{12} = -\frac{1}{12} f''(x_i) = -\frac{1}{12} \frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} + \mathcal{O}(h^2), \tag{P.1.12}$$

  and therefore

$$f_i - K_2 h^2 = \frac{1}{12}(f_{i-1} + 10 f_i + f_{i+1}) + \mathcal{O}(h^4). \tag{P.1.13}$$

- It follows from (P.1.10) and (P.1.13) that

$$u_i = \frac{u_{i-1} + u_{i+1}}{2} + \frac{h^2}{24}(f_{i-1} + 10 f_i + f_{i+1}) + \mathcal{O}(h^6). \tag{P.1.14}$$

Note: The scheme in (P.1.14) can replace the linear interpolation in (P.1.8). However, the Method of Modified Equation is *hardly* applicable for more general problems. Thus **you will not use (P.1.14)**.

Remark P.8. The CR algorithm for Sixth-order Solutions
You may apply the CR algorithm to get *possibly* sixth-order solutions from fourth-order ones. Let us see some details to be applied for Steps 2 and 3; the other steps are the same as the original CR.

2. *The sixth-order solution of Richardson extrapolation*:

$$\mathbf{v}_{h/2}^{(6)}\big|_{\Omega_h} = \frac{1}{15}\big(16\mathbf{v}_{h/2}^{(4)}\big|_{\Omega_h} - \mathbf{v}_h^{(4)}\big). \tag{P.1.15}$$

3. The *corresponding correction term* is defined as $\mathbf{c} \equiv \mathbf{v}_{h/2}^{(6)} - \mathbf{v}_{h/2}^{(4)}$; its available values are

$$\mathbf{c}\big|_{\Omega_h} = \big(\mathbf{v}_{h/2}^{(6)} - \mathbf{v}_{h/2}^{(4)}\big)\big|_{\Omega_h} = \frac{1}{15}\big(\mathbf{v}_{h/2}^{(4)}\big|_{\Omega_h} - \mathbf{v}_h^{(4)}\big). \tag{P.1.16}$$

What To Do

1. Find $\mathbf{v}_h$, $\mathbf{v}_{h/2}$, $\mathbf{v}_{h/4}$, and $\mathbf{v}_{h/8}$, **with $h = 1/10$ ($n = 10$)**.
   (You should solve the linear system (P.1.4) for each grid size.)
2. Apply the Richardson extrapolation and check the accuracy order as mentioned in **Remark P.3**.
   (For order of convergence, see **Definition 5.22**, p. 200.)
3. Use **Algorithm P.4** (CR): *Richardson extrapolation & Linear interpolation* (RELI).
   - Get fourth-order solutions on three fine meshes.
   - Apply the CR once more to get *possibly* sixth-order solutions on the last two fine meshes, from the above fourth-order solutions. Are they in a sixth-order accuracy?

   Plot error for each of RELI solutions.

Note: Below is optional for **undergraduate students**; you will get an extra credit when you do.

4. Replace the linear interpolation with a **cubic spline interpolation** for Step 4 of **Algorithm P.4**. Then carry out Step 3.

Report

On the first page, write 'Undergraduate' or 'Graduate' for your identity. Then add summary, analysis, code you implemented, and results including figures.

- *Undergraduate students*
  - Each step is worth 30 points.
  - A summary is worth 10 points.
- *Graduate students*
  - Each step is worth 20 points.
  - A summary is worth 20 points.

If you use Matlab, you may start with

get_alg_system.m

```
function [A,b] = get_alg_system(interval,n,f,BV)

A=zeros(n+1,n+1);
b=zeros(n+1,1);
h=(interval(2)-interval(1))/n;

%%---------
for i=2:n
   A(i,i-1) = -1;
   A(i,i) = 2;
   A(i,i+1) = -1;
   b(i) = h^2*f( interval(1)+(i-1)*h );
end

A(1,1)=1;   A(n+1,n+1)=1;
b(1)=BV(1); b(n+1)=BV(2);
```

linear_interpol.m

```
function Q = linear_interpol(P)
% Input:  P, of length n0
% Output: Q, of length 2*n0-1

n0= length(P); n = 2*n0-1;

Q = zeros(n,1);
Q(1:2:end) = P;
for i=2:2:n
    Q(i) = (Q(i-1)+Q(i+1))/2;
end
```

Richardson_extrapolation.m

```
clear all; close all

u = @(x) sin(pi*x)+2*x;
f = @(x) pi^2*sin(pi*x);
BV= [0,2];
interval=[0,1]; n0 = 10; ngrid=4;

%%----------------------------------------
%%-- U & V
%%----------------------------------------
U = cell(ngrid,1);  V  = cell(ngrid,1);
for i=1:ngrid
    n = n0*2^(i-1); X =linspace(0,1,n+1)';
    U{i} = u(X);
    [A,b] = get_alg_system(interval,n,f,BV);
    V{i}= A\b;
end

%%----------------------------------------
%%-- Richardson: 4th-order & 6th-order
%%----------------------------------------
R = cell(ngrid,2);
for i=2:ngrid
    R{i,1} = (1/3)*(4*V{i}(1:2:end)-V{i-1});
end
for i=3:ngrid
    R{i,2} = (1/15)*(16*R{i,1}(1:2:end)-R{i-1,1});
end

%%----------------------------------------
%%-- RI: Interpolation/Expansion of R
%%----------------------------------------
RI = cell(ngrid,2);
for i=2:ngrid
    C = (V{i}(1:2:end)-V{i-1})/3;
    RI{i,1} = V{i}+linear_interpol(C);
end

% RI{i,2}??

%%----------------------------------------
%%-- Check Error & Print
%%----------------------------------------
```

```
E = zeros(ngrid,1); ER = zeros(ngrid,1); ERI = zeros(ngrid,1);
for i=2:ngrid
   E(i)  = norm(U{i}-V{i},inf);
   ER(i) = norm(U{i-1}-R{i,1},inf);
   ERI(i)= norm(U{i}-RI{i,1},inf);
end

fmt=['L8-Error: %s =' repmat(' %-10.4g',1,numel(E)) '\n'];
fprintf(fmt, '|U-V| ',E);
fprintf(fmt, '|U-R| ',ER);
fprintf(fmt, '|U-RI|',ERI);
```

Output

```
>> Richardson_extrapolation
L8-Error: |U-V|  = 0          0.002059   0.0005142  0.0001285
L8-Error: |U-R|  = 0          1.02e-05   6.35e-07   3.965e-08
L8-Error: |U-RI| = 0          1.509e-05  9.492e-07  5.942e-08
```

P.2. Richardson Extrapolation for Evolutionary PDEs

It is recommended to first examine the project regarding **Richardson extrapolation** for elliptic PDEs, located on page 590, P.1. Let's go over the main concept of Richardson extrapolation; see Section 4.2 for details.

Richardson extrapolation is used to generate **high-accuracy results**, using low-order formulas (on two or more different grids).

Example P.9. Derive the **three-point midpoint formulas**:

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x-h)}{2h} - \left[\frac{h^2}{3!}f^{(3)}(x) + \frac{h^4}{5!}f^{(5)}(x) + \frac{h^6}{7!}f^{(7)}(x) + \cdots\right], \\ f''(x) &= \frac{f(x-h)-2f(x)+f(x+h)}{h^2} - \left[2\frac{h^2}{4!}f^{(4)}(x) + 2\frac{h^4}{6!}f^{(6)}(x) + \cdots\right]. \end{aligned} \tag{P.2.1}$$

Solution. It follows from the **Taylor's series formula** (1.14) that

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \frac{f^{(4)}(x)}{4!}h^4 + \cdots \\ f(x-h) &= f(x) - f'(x)h + \frac{f''(x)}{2!}h^2 - \frac{f'''(x)}{3!}h^3 + \frac{f^{(4)}(x)}{4!}h^4 - \cdots \end{aligned} \tag{P.2.2}$$

Adding and subtracting these two equations, we have

$$\begin{aligned} f(x+h)+f(x-h) &= 2f(x) + 2\frac{\mathbf{f''(x)}}{2!}h^2 + 2\frac{f^{(4)}(x)}{4!}h^4 + \cdots \\ f(x+h)-f(x-h) &= 2\,\mathbf{f'(x)}h + 2\frac{f'''(x)}{3!}h^3 + 2\frac{f^{(5)}(x)}{5!}h^5 + \cdots \end{aligned} \tag{P.2.3}$$

Now, solve these equations for $f''(x)$ and $f'(x)$. □

Observation P.10. The results in Example P.9 can be written as

$$M = N(h) + K_2h^2 + K_4h^4 + K_6h^6 + \cdots, \tag{P.2.4}$$

where M is the desired (unknown) quantity, $N(h)$ is an approximation of M using the parameter h, and ***K_i* are independent of *h*.**

How can we take advantage of the observation?

Strategy **P.11. (Richardson extrapolation)**: Rewrite (P.2.4):

$$M \;=\; N(h) \;+\; \mathbf{K_2 h^2} + K_4 h^4 + K_6 h^6 + \cdots . \tag{P.2.5}$$

1. Let's write out (P.2.5) with h replaced by $h/2$:

$$M \;=\; N(h/2) \;+\; \underline{\mathbf{K_2 h^2}/4} + K_4 h^4/16 + K_6 h^6/64 + \cdots . \tag{P.2.6}$$

Then the leading term in the error series, $K_2 h^2$, can be eliminated as follows:

$$\begin{array}{rcrl} M &=& N(h) & +K_2 h^2 + K_4 h^4 + K_6 h^6 + \cdots \\ 4M &=& 4N(h/2) & +K_2 h^2 + K_4 h^4/4 + K_6 h^6/16 + \cdots \\ \hline 3M &=& 4N(h/2) - N(h) & -\frac{3}{4} K_4 h^4 - \frac{15}{16} K_6 h^6 - \cdots \end{array} \tag{P.2.7}$$

Thus we have

$$M \;=\; \frac{1}{3}[4N(h/2) - N(h)] - \frac{1}{4} K_4 h^4 - \frac{5}{16} K_6 h^6 - \cdots . \tag{P.2.8}$$

The above equation embodies the first step in *Richardson extrapolation*. It show that a simple combination of two second-order approximations, $N(h)$ and $N(h/2)$, furnishes an estimate of M with accuracy $\mathcal{O}(h^4)$.

2. For simplicity, we rewrite (P.2.8) as

$$M \;=\; N_2(h) - \frac{1}{4} K_4 h^4 - \frac{5}{16} K_6 h^6 - \cdots . \tag{P.2.9}$$

Then, similarly,

$$M \;=\; N_2(h/2) - \frac{1}{64} K_4 h^4 - \frac{5}{2^{10}} K_6 h^6 - \cdots . \tag{P.2.10}$$

Subtract (P.2.8) from 16 times (P.2.10) to produce a new $\mathcal{O}(h^6)$ formula:

$$M \;=\; \frac{1}{15}[16 N_2(h/2) - N_2(h)] + \frac{1}{64} K_6 h^6 + \cdots . \tag{P.2.11}$$

Recall: **Example 16.19, p. 580.** Consider the problem

$$\begin{aligned} u_t - u_{xx} &= (1+\pi^2)e^t \sin(\pi x), && (x,t) \in [0,1]\times[0,1], && \text{(PDE)} \\ u &= 0, && (x,t) \in \{0,1\}\times[0,1], && \text{(BC)} \\ u &= \sin(\pi x), && x \in [0,1],\ t = 0. && \text{(IC)} \end{aligned} \tag{P.2.12}$$

The exact solution is $u(x,t) = e^t \sin(\pi x)$.

What To Do

1. Consider four different meshing/partitioning:

Ω_0 :	the mesh with $\Delta x = 1/10$ & $\Delta t = 1/20$
Ω_1 :	the mesh with $\Delta x = 1/20$ & $\Delta t = 1/40$
Ω_2 :	the mesh with $\Delta x = 1/40$ & $\Delta t = 1/80$
Ω_3 :	the mesh with $\Delta x = 1/80$ & $\Delta t = 1/160$

   The major goal of the project is **to obtain high-order numerical solutions for (P.2.12)**, ideally, 6th-order, employing the **second-order Crank-Nicolson method**.

2. Use appropriate portions in the code, `Heat_1D_Solve.m` and `theta_march.m` on pp.580–581, to implement the **Richardson extrapolation**. For example, with Ω_0 and Ω_1:
   - For (each one step of Ω_0) and (each two steps of Ω_1), upgrade both the solutions to become 4th-order; **both of them must be upgraded to be identical on the coarse grid Ω_0.**
     - Following the Richardson extrapolation, you should apply an interpolation method.
     - First, try the simple **linear interpolation**.
     - Then, apply the **cubic spline**, which is built-in in Matlab and Python. For example, in Matlab: `interpn(...,'spline')`
     - Measure the error.
   - March till the end, which will produce a numerical solution of **4th-order accuracy.**

3. **(Graduate students)** Now, edit your code to get a 6th-order numerical solution.

Report

On the first page, write 'Undergraduate' or 'Graduate' for your identity. Then add summary, analysis, code you implemented, and results including figures. Note that **for 4th-order solutions, you have to run your code 3 times with different pairs of meshes.**

- *Undergraduate students*
  - The 4th-order solutions are worth 90%.
  - A summary is worth 10%.
  - Any reasonable attempt at the 6th-order numerical solution will be recognized.
- *Graduate students*
  - The 4th-order solutions are worth 60%.
  - The 6th-order solutions are worth 30%.
  - A summary is worth 10%.

P.3. Project Regression Analysis: Linear, Piecewise Linear, and Nonlinear Models

Regression analysis is a set of statistical processes for estimating the relationships between independent variables and dependent variables.

- Regression analysis is **a way to find trends in data**.
- There are variations: **linear, multiple linear, and nonlinear.**
  - The most common models are simple linear and multiple linear.
  - Nonlinear regression analysis is commonly used when the dataset shows a nonlinear relationship.
- Choosing an appropriate regression model is often a difficult task. **In this project**: we'll try **to find best regression models**, for given datasets.

Strategy P.12. Determination of the Best Model

Suppose we are given a dataset as in the figure.

1. We may plot the dataset for a **visual inspection**.
2. One or more good models can be selected.
3. Then, the best model is determined through ***analysis***.

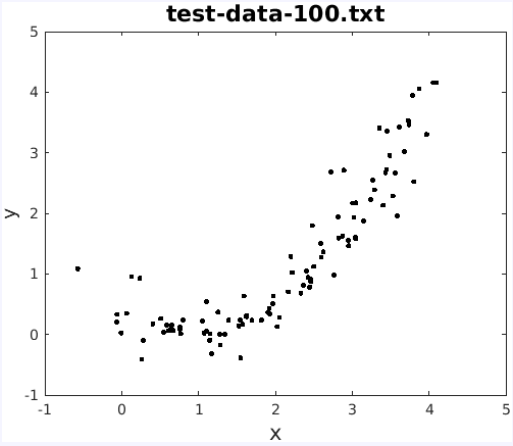


Figure P.1: A test dataset of 100 points $\{(x_i, y_i) \mid i = 1, 2, \cdots, 100\}$.

It looks like a quadratic polynomial!

Polynomial Fitting: A Review

For the dataset $\{(x_i, y_i) \mid i = 1, 2, \cdots, m\}$ in Figure P.1, consider a regression model of the form

$$y = a_0 + a_1 x + a_2 x^2.$$

- Then

Predicted y-value		Observed y-value
$a_0 + a_1 x_1 + a_2 x_1^2$	$=$	y_1
$a_0 + a_1 x_2 + a_2 x_2^2$	$=$	y_2
$\vdots$		$\vdots$
$a_0 + a_1 x_m + a_2 x_m^2$	$=$	y_m

(P.3.1)

- It is equivalently written as

$$X\mathbf{p} = \mathbf{y}, \tag{P.3.2}$$

where

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_m & x_m^2 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

- The system can be solved using the **method of normal equations**:

$$X^T X\mathbf{p} = \begin{bmatrix} \Sigma 1 & \Sigma x_i & \Sigma x_i^2 \\ \Sigma x_i & \Sigma x_i^2 & \Sigma x_i^3 \\ \Sigma x_i^2 & \Sigma x_i^3 & \Sigma x_i^4 \end{bmatrix} \mathbf{p} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \\ \Sigma x_i^2 y_i \end{bmatrix} = X^T \mathbf{y} \tag{P.3.3}$$

Note: The above **polynomial fitting** is well implemented in most programming languages.

Matlab: `p = polyfit(x,y,deg);`

Python: `p = np.polyfit(x,y,deg)`

where `x` and `y` are arrays of x- and y-coordinates, respectively, and `deg` denotes the degree of the regression polynomial. **We will use it!**

test_data_100.m

```
close all; clear all

DATA = readmatrix('test-data-100.txt');
x = DATA(:,1); y = DATA(:,2);
p = polyfit(x,y,2);
yhat = polyval(p,x);                    % predicted y-values
LS_error = norm(y-yhat)^2/length(y);  % variance

%-----------------------------------------------------
fprintf('LS_error= %.3f; p=',LS_error); disp(p)
% Output: LS_error= 0.130; p= 0.3944 -0.6824 0.3577

%-----------------------------------------------------
x1 = linspace(min(x),max(x),100);
y1 = polyval(p,x1);                     % regression curve

figure, plot(x,y,'k.','MarkerSize',8); hold on
xlim([-1,5]); ylim([-1,5]);
xlabel('x','fontsize',15); ylabel('y','fontsize',14);
title('test-data-100: Regression','fontsize',14);
plot(x1,y1,'r-','linewidth',2)
exportgraphics(gcf,'test-data-100-regression.png','Resolution',100);
```

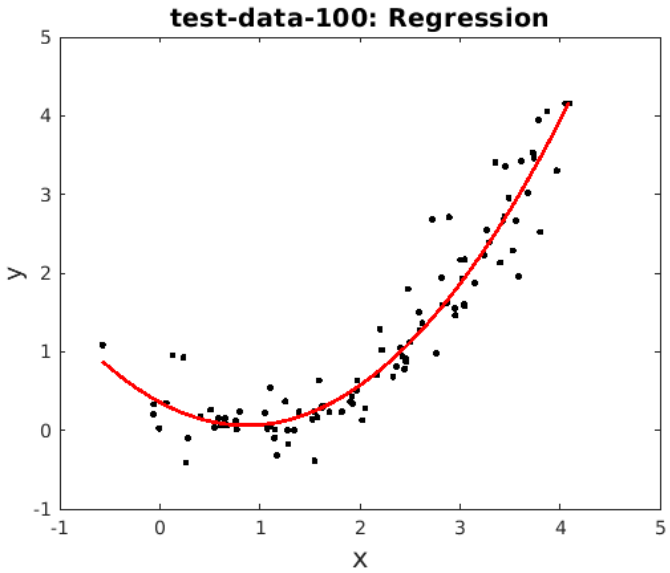


Figure P.2: test-data-100-regression.png: $y = 0.3944 * x^2 - 0.6824 * x + 0.3577$.

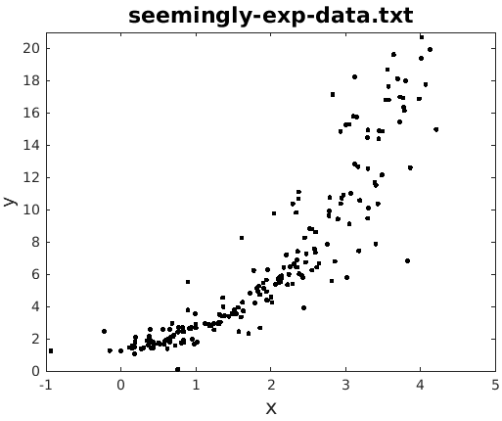
Nonlinear Regression

Example P.13. See the dataset $\{(x_i, y_i)\}$ shown in the figure. When we try to find the best fitting model of the form

$$y = c\, e^{dx}, \tag{P.3.4}$$

the corresponding **nonlinear least-squares problem** reads

$$\min_{c,d} \sum_{i=1}^{m} \left(y_i - c\, e^{dx_i}\right)^2. \tag{P.3.5}$$



The problem can be solved by applying a nonlinear iterative solver such as the **Newton's method** with a good initialization (c_0, d_0).

We can solve it **much more easily** through **linearization**.

Linearization by Change of Variables

$$y = c\, e^{dx}. \tag{P.3.6}$$

The Goal: find the best-fitting (c, d) for the dataset.

- **(Transform)** Apply the logarithmic function to have

$$\ln y = \ln(c\, e^{dx}) = \ln c + \ln e^{dx} = dx + \ln c. \tag{P.3.7}$$

- **(Change of Variables)** Define

$$X = x; \quad Y = \ln y; \quad a = \ln c. \tag{P.3.8}$$

- **(Linear Model)** Then the model in (P.3.7) reads

$$Y = d\, X + a, \tag{P.3.9}$$

  which is a linear model; we can get best (d, a), by `polyfit(X,Y,1)`.
- Finally we recover c: $c = e^a$.

nonlinear_regression.m

```
lose all; clear all

FILE = 'seemingly-exp-data.txt';
DATA = readmatrix(FILE);
% fitting: y =c*exp(d*x) ---> ln(y) = d*x + ln(c)
%-------------------------------------------------

x = DATA(:,1); y = DATA(:,2);
lny = log(y);                    % data transform
p = polyfit(x,lny,1);            % [p1,p2] = [d, ln(c)]
d = p(1); c = exp(p(2));

LS_error = norm(y-c*exp(d*x))^2/length(y);  % variance

%-------------------------------------------------
fprintf('c=%.3f; d=%.3f; LS_error=%.3f\n',c,d,LS_error)
% Outout: c=1.346; d=0.669; LS_error=4.212

% figure
```

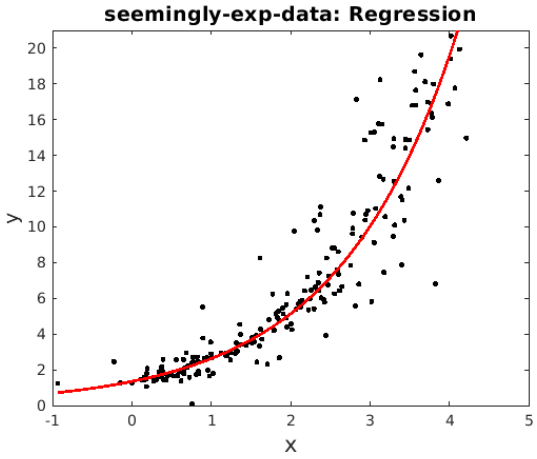


Figure P.3: The data and a nonlinear regression via linearization: `y=1.346*exp(0.669*x)`.

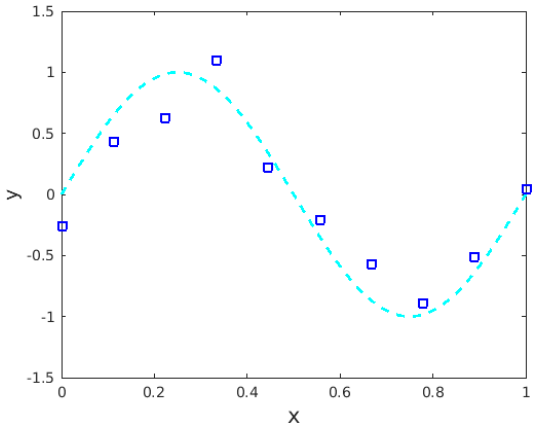
Note: If you want to test `nonlinear_regression.m`, you may download
https://skim.math.msstate.edu/LectureNotes/data/nonlinear_regression.m
https://skim.math.msstate.edu/LectureNotes/data/seemingly-exp-data.txt
The dataset consists of 200 points, generated by `y=1.3*exp(0.7*x)` with a random positioning and random noise.

Finding the Best Regression Model

Example P.14. Consider a simple dataset: 10 points generated from ***a sine function***, with noise.

Want: Find the ***best regression polynomial*** for the dataset:

- Let's select a model from $\mathbb{P}_n$, polynomials of degree $\leq n$



Sine_Noisy_Data_Regression.m

```
close all; clear all

a=0; b=1; m=10;
f = @(t) sin(2*pi*t);
DATAFILE = 'sine-noisy-data.txt';
renew_data = 0;

%%-----------------------------------------------
if isfile(DATAFILE) && renew_data == 0
   DATA = readmatrix(DATAFILE);        % np.loadtxt()
else
   X = linspace(a,b,m); Y0 = f(X);
   noise = rand([1,m]); noise = noise-mean(noise(:));
   Y = Y0 + noise; DATA = [X',Y'];
   writematrix(DATA,DATAFILE);         % np.savetxt()
end

%%-----------------------------------------------
x = linspace(a,b,101); y = f(x);
x1 = DATA(:,1); y1 = DATA(:,2);
E = zeros(1,m);
for n = 0:m-1
    p = polyfit(x1,y1,n);              % np.polyfit()
    yhat = polyval(p,x1);              % np.polyval()
    E(n+1) = norm(y1-yhat,2)^2;
    %savefigure(x,y,x1,y1,polyval(p,x),n)
end

% figure
```

Polynomial-fitting

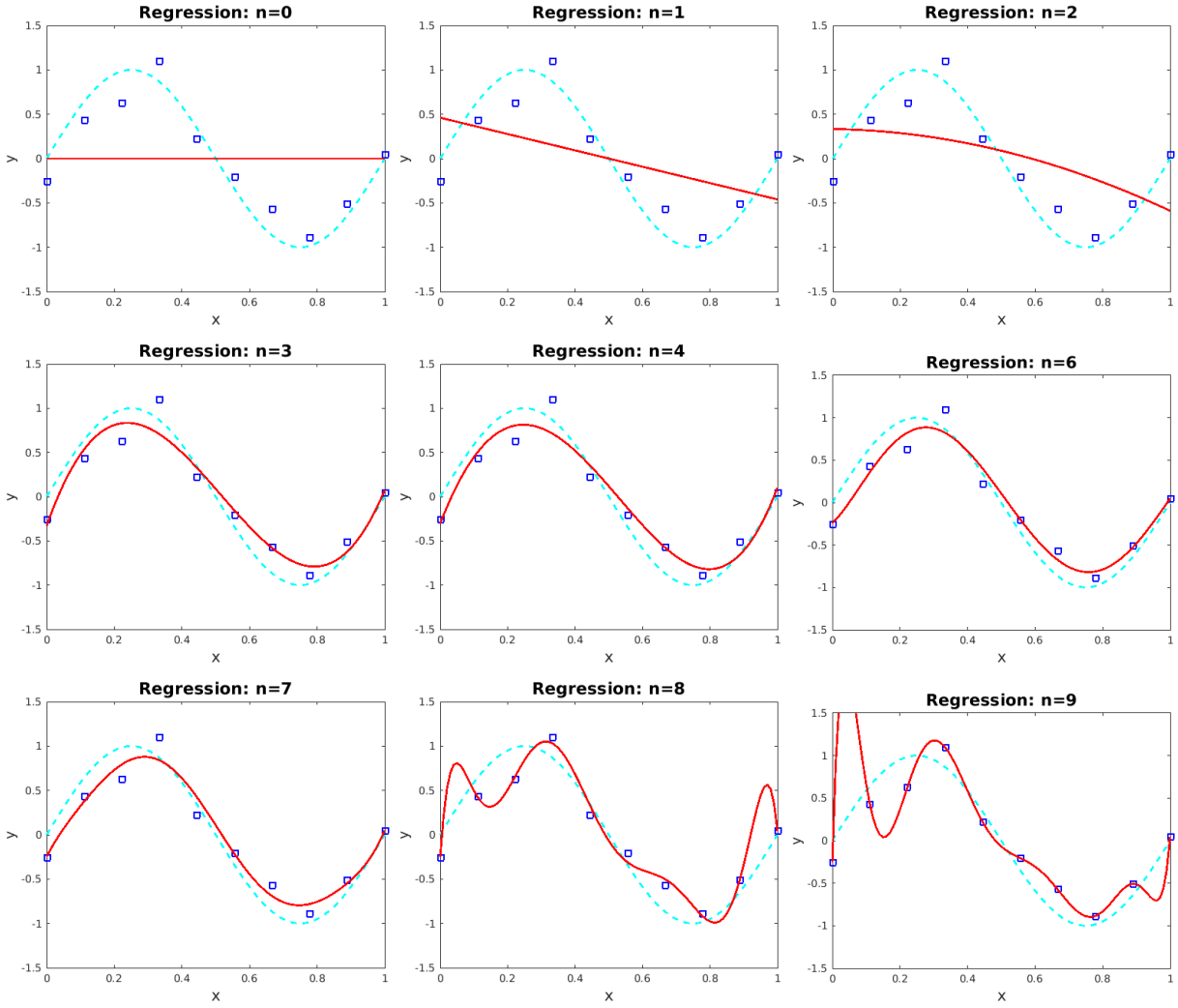


Figure P.4: Regression models P_n, $n = 0, 1, \cdots, 9$.

Which model is the best?

Analysis: The LS Error

Given the dataset $\{(x_i, y_i) \mid i = 1, 2, \cdots, m\}$ and the model P_n, define the LS-error

$$E_n = \sum_{i=1}^{m} \big(y_i - P_n(x_i)\big)^2, \qquad (m = 10), \tag{P.3.10}$$

which is also called the **mean square error**.

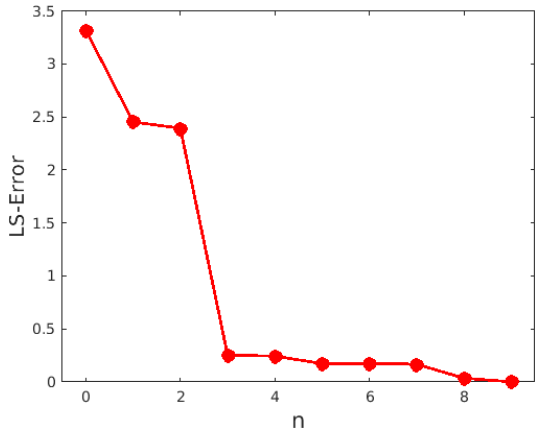
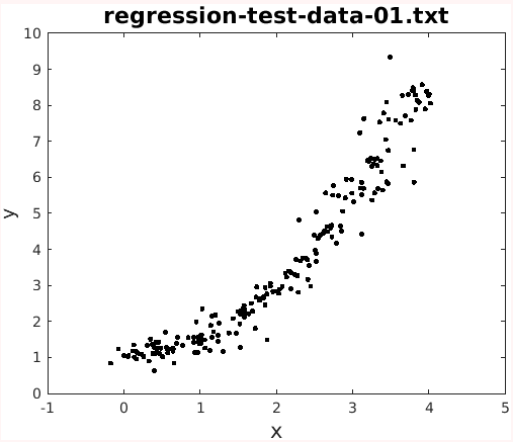


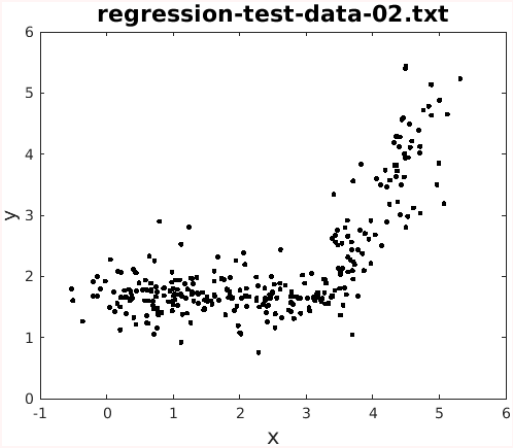
Figure P.5: **The best choice is P_3, the third-order polynomial.**

Strategy **P.15.** Given several models with similar explanatory ability, **the simplest is most likely to be the best choice.**

- Start simple, and only make the model more complex as needed.

Project Objective: To find the best model for each of the datasets:





What to Do

First download two datasets:

https://skim.math.msstate.edu/LectureNotes/data/regression-test-data-01.txt
https://skim.math.msstate.edu/LectureNotes/data/regression-test-data-02.txt

1. **Finding Best Models**
   (a) For **regression-test-data-01.txt**, which model is better,
   `y=a0+a1*x+a2*x.^2` or `y = c*exp(d*x)`?
   (b) For **regression-test-data-02.txt** (Data-02), which order of polynomial is fitting the best? Your claim must be supported pictorially as in Figure P.4.
2. **Verification**: For all models, measure the LS-errors. Show them ***in a tabular form***.
3. **Figuring**: For Data-02, display the LS-error as in Figure P.5.
4. **Extra Credit**: Find a **piecewise regression** model for Data-02. Is it better than polynomial models? (You must verify your answer.)

Note: You may use parts of the codes shown in this project section. Report your code, numerical outputs, and figures, with a summary.

- A code itself will not give you any credit; include outputs or figures.
- The summary will be worth 20% the full credit.
- Include all into a single file in pdf or doc/docx format.

P.4. Quasi-Newton Method Using Partial Information of the Hessian

Consider an **unconstrained optimization problem** of the form

$$\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x}). \tag{P.4.1}$$

Given an initialization $\mathbf{x}_0$:

- The **gradient descent (GD) method** is formulated as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma\, \nabla f(\mathbf{x}_k), \tag{P.4.2}$$

  for some $\gamma > 0$.

- The **Newton's method** updates the iterates as

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma\, \left[\nabla^2 f(\mathbf{x}_k)\right]^{-1} \nabla f(\mathbf{x}_k), \tag{P.4.3}$$

  where $\nabla^2 f$ is the **Hessian** (or **Hessian matrix**) defined as

$$\nabla^2 f = \begin{bmatrix} \dfrac{\partial^2 f}{\partial {x_1}^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_d} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial {x_2}^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_d \partial x_1} & \dfrac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial {x_d}^2} \end{bmatrix} \in \mathbb{R}^{d\times d}. \tag{P.4.4}$$

- For large problems, the computation of the Hessian and its inverse is computationally expensive, occasionally becomes impractical.
- Wherever applicable, the **Newton's method converges much faster** towards a local extremum than the gradient descent method. Every local minimum has a neighborhood such that,
  - if we start within this neighborhood,
  - the Newton's method with step size $\gamma = 1$ converges quadratically,

  assuming the Hessian is invertible and Lipschitz continuous.

Example P.16. The Newton's method can be implemented **conveniently**, using the symbolic library in Python, for example.

opt_sympy_Newton.py

```
import numpy as np
from sympy import symbols, diff, Matrix
np.set_printoptions(precision=10)

def gradient(f,Vars):
    """Symbolic Computation of Grad."""
    return Matrix([diff(f,var) for var in Vars])

def grad_Hessian(f,Vars):
    """Symbolic Computation of Grad and Hessian."""
    G = Matrix([diff(f,var) for var in Vars])
    H = Matrix([[diff(G[i],var) for var in Vars] for i in range(len(Vars))])
    return G, H

def evaluate(Math_Form,Vars,P):
    """Evaluation of Mathematical Expression at P."""
    return np.array(Math_Form.subs( zip(Vars,P) )).astype(float)
    # list(zip(Vars,P)) = [(x0,P[0]), (x1,P[1]), (x2,P[2])]

if __name__ == '__main__':
    x0, x1, x2 = Vars = symbols('x:3')
    f = 2*x0**2 + 2*x1**2 + x2**2 + x0*x1 -2*x1*x2 -x0*x2 - 2*x0 + x2

    G,H = grad_Hessian(f,Vars)
    #print("Gradient:", G);
    print("Hessian:", H)
    #point = np.arange(1,len(Vars)+1)
    #print("G(x=point):\n", evaluate(G,Vars,point))
    #print("H(x=point):\n", evaluate(H,Vars,point))

    #-----------------------------------------------
    print("Find the minimum, by the Newton's method")
    #-----------------------------------------------
    x = np.zeros((len(Vars),)); print("x_0 =",x)
    gamma = 1.
    itmax = 5
    for k in range(1,itmax+1):
       grad = evaluate(G,Vars,x); hess = evaluate(H,Vars,x)
       x = x - gamma * np.squeeze(np.linalg.inv(hess) @ grad)
       print("x_%d = " %(k),end=""); print(x)
```

OUT

```
Hessian: Matrix([[4, 1, -1], [1, 4, -2], [-1, -2, 2]])
Find the minimum, by the Newton's method
x_0 = [0. 0. 0.]
x_1 = [ 0.4285714286 -0.5          -0.7857142857]
x_2 = [ 0.4285714286 -0.5          -0.7857142857]
x_3 = [ 0.4285714286 -0.5          -0.7857142857]
x_4 = [ 0.4285714286 -0.5          -0.7857142857]
x_5 = [ 0.4285714286 -0.5          -0.7857142857]
```

For **quadratic optimization problems**, the Newton's method converges in a single iteration *always*, no matter what the initialization x_0 is.

Issues. The central issue with the Newton's method is that we need to be able to compute the **inverse Hessian matrix**, *effectively*.

- For many applications, the dimensionality of the problem can be of the **order of thousands or millions**; computing the Hessian or its inverse is often impractical.
- Because of these reasons, the Newton's method is **rarely used in practice** to optimize functions corresponding to **large problems**.
- Luckily, the algorithm can still work, even if the Hessian is replaced by a **good approximation**.
- Various "**quasi-Newton**" methods have been developed so as to approximate and update the Hessian matrix (without evaluating the second derivatives).

One of the most popular **quasi-Newton methods** is the **BFGS algorithm**, which is named after Charles George **Broyden** [2], Roger **Fletcher** [12], Donald **Goldfarb** [15], and David **Shanno** [33].

Note: As a quasi-Newton method, what we will try to do in this project is **to use a banded submatrix** of the Hessian.

- Given a matrix $A \in \mathbb{R}^{n\times n}$, consider a **similarity transformation** of the form

$$S = PAP^T, \tag{P.4.5}$$

where P is a **permutation matrix**, as defined in Definition 10.22, p. 365.

- Assume that there is a permutation matrix such that the Hessian H is transformed to a banded matrix B. That is,

$$B = PHP^T. \tag{P.4.6}$$

This task is not easy to achieve; however, in many applications, the given matrix can be transformed to a **near-banded matrix**.

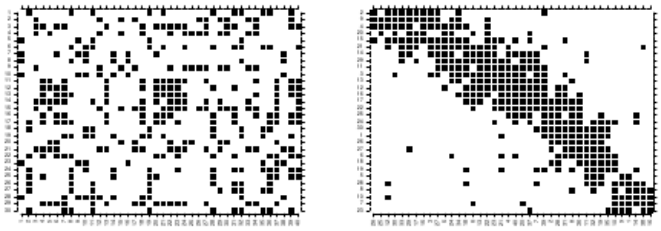


Figure P.6: A matrix in its original form (left) and its transformed matrix (right), as shown by Garriga *et al.*[13].

Application of Banded Structure for the Newton's Method

- Recall the Newton's method:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \gamma\, H_k^{-1}\, \mathbf{g}_k; \quad H_k = \nabla^2 f(\mathbf{x}_k), \;\; \mathbf{g}_k = \nabla f(\mathbf{x}_k). \tag{P.4.7}$$

- Let

$$\mathbf{y} = H_k^{-1}\, \mathbf{g}_k \quad \Rightarrow \quad H_k\mathbf{y} = \mathbf{g}_k \tag{P.4.8}$$

- Let $B_k = PH_kP^T$, a banded matrix or its banded approximation (submatrix). Then $P^TB_kP = H_k$ and

$$P\mathbf{g}_k = PH_k\mathbf{y} = P(P^TB_kP)\mathbf{y} = B_kP\mathbf{y}. \tag{P.4.9}$$

- Thus, the update y can be obtained **much more efficiently**:

$$B_k\mathbf{v} = P\mathbf{g}_k \quad \text{and then} \quad \mathbf{y} = P^T\mathbf{v}\,. \tag{P.4.10}$$

Objectives

- **Algorithm Development**:
  - Define f_4 as a function of four variables, having a local minimum.
  - Define f_6 as a function of six variables, having a local minimum.
  - For $\min_{\mathbf{x}} f_d(\mathbf{x})$, $d = 4, 6$: Use parts of `opt_sympy_Newton.py` for
    - the **Gradient Descent (GD) method**, and
    - the **Newton's method**.
  - Add another function for a **quasi-Newton method**, in which the Hessian $H = [h_{ij}]$ is replaced by its tri-diagonal portion, $H^{(3)} = [t_{ij}]$. (That is, $t_{ij} = h_{ij}$ if $|i - j| \leq 1$ else 0.)
    Let's call this method the **Banded Hessian (BH) method**, which is a quasi-Newton method.

  Now, you have two problems and three algorithms.

- **Comparisons**: For each problem, compare the performances of the three algorithms: GD, Newton's, and BH.
  - Try to set γ as good as possible, for each problem and each method, while trying to understand the algorithms' convergence behaviors.

  For the Newton's method, you may set $\gamma = 1$. If the iterates are oscillatory or diverging, particularly for the GD method, then it indicates that γ is too large.

  - Compare the number of iterations and the total elapsed time, to conclude **the best algorithm** for each problem.
  - When the problem size is much larger, e.g. a thousand or a million, is it possible for you to conclude differently?

(To be continued $\Rightarrow$)

Note: **Objective Functions: f_4 and f_6**

- **To All**: Try to include the term `x0*x3` for f_4 and `x0*x5` for f_6, which will make each of the Hessians have a nonzero value at top right and bottom left corners.
- **Undergraduate Students**: For simplicity, try to select f_4 and f_6 to be quadratic functions so that their Hessians are constant matrices.
- **Graduate Students**: You should add at least one fourth-order term, of the form `a*`x_i`**4`, with $a > 0$. Do not add any third-order terms.
- **To All**: Select the coefficients of square and quartic terms larger than those of mixed terms. This will make the Hessian diagonally dominant and symmetric positive definite.

  $\Rightarrow$ The banded Hessian submatrix can be a good approximation, although a similarity transformation is not applied.

Report: Start with a one-page summary, including your conclusion.

- Attach your whole code.
- Include your comparison analysis, with supporting results.

Bibliography

[1] R. BARRETT, M. BERRY, T. CHAN, J. DEMMEL, J. DONATO, J. DONGARRA, V. EIJKHOUT, R. POZO, C. ROMINE, AND H. VAN DER VORST, *Templates for the solution of linear systems: Building blocks for iterative methods*, SIAM, Philadelphia, 1994. The postscript file is free to download from `http://www.netlib.org/templates/` along with source codes.

[2] C. G. BROYDEN, *The convergence of a class of double-rank minimization algorithms 1. General considerations*, IMA Journal of Applied Mathematics, 6 (1970), pp. 76–90.

[3] J. BUNCH AND L. KAUFMAN, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comput., 31 (1977), pp. 163–179.

[4] O. CHUM AND J. MATAS, *Matching with prosac-progressive sample consensus*, in 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05), vol. 1, IEEE, 2005, pp. 220–226.

[5] C. CORTES AND V. N. VAPNIK, *Support-vector networks*, Machine Learning, 20 (1995), pp. 273–297.

[6] G. DAHLQUIST, *A special stability problem for linear multistep methods*, BIT, 3 (1963), pp. 27–43.

[7] R. DAI AND P. LIN, *Analysis on sixth-order compact approximations with Richardson extrapolation for 2D Poisson equation*, Journal of Applied Mathematics and Physics, 6 (2018), pp. 1139–1159.

[8] R. DAI, J. ZHANG, AND Y. WANG, *Multiple coarse grid acceleration for multiscale multigrid computation*, Journal of Computational and Applied Mathematics, 269 (2014), pp. 75–85.

[9] S. DEERWESTER, S. DUMAIS, G. FURNAS, T. LANDAUER, AND R. HARSHMAN, *Indexing by latent semantic analysis.*, Journal of the American Society for Information Science, (1990), pp. 391–407.

[10] L. ELDÉN, *Numerical linear algebra in data mining*, Acta Numerica, 15 (2006), pp. 327 – 384.

[11] M. FISCHLER AND R. BOLLES, *Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography*, Communications of the ACM, 24 (1981), pp. 381–395.

[12] R. FLETCHER, *A new approach to variable metric algorithms*, The Computer Journal, 13 (1970), pp. 317–322.

[13] G. C. GARRIGA, E. JUNTTILA, AND H. MANNILA, *Banded structure in binary matrices*, Knowledge and Information Systems, 28 (2011), pp. 197–226.

[14] S. GERSCHGORIN, *Über die abgrenzung der eigenwerte einer matrix*, Izv. Akad. Nauk SSSR Ser. Mat., 7 (1931), pp. 746–754.

[15] D. GOLDFARB, *A family of variable metric methods derived by variational means*, Mathematics of Computation, 24 (1970), pp. 23–26.

[16] F. GOLUB AND C. V. LOAN, *Matrix Computations, 3rd Ed.*, The Johns Hopkins University Press, Baltimore, 1996.

[17] G. H. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403–420.

[18] B. GROSSER AND B. LANG, *An $\wr(n^2)$ algorithm for the bidiagonal svd*, Lin. Alg. Appl., 358 (2003), pp. 45–70.

[19] H. HOTELLING, *Analysis of a complex of statistical variables into principal components*, Journal of Educational Psychology, 24 (1933), pp. 417–441 and 498–520.

[20] ——, *Relations between two sets of variates*, Biometrika, 28 (1936), pp. 321–377.

[21] C. JOHNSON, *Numerical Solutions of Partial Differential Equations by the Finite Element Method*, Cambridge University Press, New York, New Rochelle, Melbourne, Sydney, 1987.

[22] W. KARUSH, *Minima of functions of several variables with inequalities as side constraints*, M.Sc. Dissertation, Department of Mathematics, Univ. of Chicago, Chicago, Illinois, 1939.

[23] C. KELLY, *Iterative methods for linear and nonlinear equations*, SIAM, Philadelphia, 1995.

[24] J. KIM, B. LEE, M. R. ZANETTI, K. A. MILLER, AND S. KIM, *Iterative update sample consensus (iusac): A repeatable algorithm for optimal consensus set*, Journal of Computational and Applied Mathematics, 436 (2024), p. 115423.

[25] H. W. KUHN AND A. W. TUCKER, *Nonlinear programming*, in Proceedings of 2nd Berkeley Symposium, Berkeley, CA, USA, 1951, University of California Press., pp. 481–492.

[26] P. C. NIEDFELDT AND R. W. BEARD, *Recursive ransac: multiple signal estimation with outliers*, IFAC Proceedings Volumes, 46 (2013), pp. 430–435.

[27] M. NIELSEN, *Neural networks and deep learning*. (The online book can be found at http://neuralnetworksanddeeplearning.com), 2013.

[28] K. PEARSON, *On lines and planes of closest fit to systems of points in space*, Philosophical Magazine, 2 (1901), pp. 559–572.

[29] P. ROACHE AND P. KNUPP, *Completed Richardson extrapolation*, Communications in Numerical Methods in Engineering, 9 (1993), pp. 365–374.

[30] F. ROSENBLATT, *The Perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, (1958), pp. 65–386.

[31] Y. SAAD AND M. H. SCHULTZ, *GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems*, SIAM Journal on Scientific and Statistical Computing, 7 (1986), pp. 856–869.

[32] G. SALTON, A. WONG, AND C.-S. YANG, *A vector space model for automatic indexing*, Communications of the ACM, 18 (1975), pp. 613–620.

[33] D. F. SHANNO, *Conditioning of quasi-Newton methods for function minimization*, Mathematics of Computation, 24 (1970), pp. 647–656.

[34] P. STEIN AND R. L. ROSENBERG, *On the solution of linear simultaneous equations by iteration*, Journal of the London Mathematical Society, s1-23 (1948), pp. 111–118.

[35] O. TAUSSKY, *Bounds for characteristic roots of matrices*, Duke Math. J., 15 (1948), pp. 1043–1044.

[36] P. H. TORR AND A. ZISSERMAN, *Mlesac: A new robust estimator with application to estimating image geometry*, Computer vision and image understanding, 78 (2000), pp. 138–156.

[37] R. VARGA, *Matrix Iterative Analysis, 2nd Ed.*, Springer-Verlag, Berlin, Heidelberg, 2000.

[38] Y. WANG AND J. ZHANG, *Sixth order compact scheme combined with multigrid method and extrapolation technique for 2D Poisson equation*, Journal of Computational Physics, 228 (2009), pp. 137–146.

[39] P. R. WILLEMS, B. LANG, AND C. VÖMEL, *Computing the bidiagonal SVD using multiple relatively robust representations*, SIAM Journal on Matrix Analysis and Applications, 28 (2006), pp. 907–926.

Index

YouTube Big-O and little-o: Body and Sweat, 29
YouTube CNN - Miner and the Jeweler, 421, 467
YouTube Data-First Numerical Methods - Gardener's Song, 561, 586
YouTube FEM - Break It Down with FEM, 533, 558
YouTube Pizza Logic in Numerical Methods, 572
YouTube The 15 Percent Rule - Google PageRank, 506
YouTube The Balance - A Lagrangian Flow, 387
1-norm, 316
15% Rule, 134, 416
2-norm, 316

ABM.mw, 203
absolute error, 26
accuracy, 566, 569
accuracy analysis, 569
activation function, 428, 439
activation functions, popular, 438
Adaline, 434, 439
Adams-Bashforth method, 202
Adams-Bashforth-Moulton method, 202
Adams-Moulton method, 202
adaptive mechanism, 106
adaptive method, 197
affine function, 394
algebraic system, 521
algorithm, 27
algorithmic parameter, 436
all-ones vector, 317
AlphaFold, 585
amplification factor, 575
angle, between two vectors, 14, 283
antiderivative, 7
API, 468
application programming interface, 468
approximation, 226
archers, 569
Arnoldi iteration, 378–380
artificial neurons, 427
assembly, 521
association rule discovery, 480
augmented matrix, 15, 267
augmented system, 15
average slope, 192

back substitution, 266, 267
backward Euler method, 212, 578
backward-difference, 140
Bairstow's method, 73
Bairstow, Leonard, 74
banded Hessian method, 615
best approximation, 223
Beyond Equations, 136, 418
BFGS algorithm, 613
BH method, 615
bias, 429
big Oh, 29, 30
bilinear functional, 556
binary classifier, 427, 428
binary-search method, 42
bisect.m, 47
bisection method, 42
black box model, 459
Bonnet's recursion formula, 169
boundary-value problem, 257, 386, 510, 534, 590
business, 566, 584
BVP, 510

C++, 468

call_LU_pivot.m, 278
cardinal functions, 90
Cauchy-Schwarz Inequality, 284
Cauchy-Schwarz inequality, 248
central path, 413
central point, 143
CG method, 373
change of variables, 173, 174, 235, 605
characteristic equation, 21, 348
characteristic polynomial, 21, 348
Chebyshev, 97
Chebyshev polynomials, 97, 168
clamped cubic spline, 123, 132
classification, 480
classification problem, 237
closed formula, 175
closed Newton-Cotes formulas, 159
closest point, 223
clustering, 424, 480
CNN, 467
coefficients, 67
cofactor, 20
cofactor expansion, 20
collocation FEM, 552
collocation method, 537
collocation points, 552
collocation_FEM.py, 553
color image, 250
color image processing, 250
column-stochastic matrix, 500
common structures, 135, 417
complementary slackness, 391, 399, 412, 445
complete pivoting, 276
composite error, 154
composite Simpson's rule, 158
composite Simpson's three-eights rule, 159
Composite Trapezoid rule, 161
composite trapezoid rule, 153
computation of a^{-1}, 279
computer vision, 249
concave, 402
concave maximization problem, 394
condition number, 25, 286
conditionally stable, 27
conjugate, 373
conjugate gradient method, 373
consistency, 565, 566
continuity, 2
continuous, 2
contractive mapping, 54
convergence, 569
convergence analysis, 569
convergence of Newton's method, 58
convergence of oeder α, 28
convergent, 353
convergent matrix, 351
convex optimization problem, 394
convex problem, 402
correction term, 56, 592
cosine distance measure, 492
cost function, 428
covariance matrix, 331
Crank-Nicolson method, 215, 578, 583, 600
cubic FEM, 545
cubic spline, 120, 600
cubic spline interpolation, 594
CUDA, 468
curiosity, 134, 416
curse of dimensionality, 425

damping factor, 502
data acquisition, 229
data analysis, 337
data compression, 342
data gap, 479
data interpretation, 237
data matrix, 339
data mining, 479
data-first discovery, 585
data-first numerical analysis, 586
Data-First Numerical Methods, 584
deductive approach, 585
deep neural networks, 467
definite matrix, 356
deflation, 77
degree of freedom, 539, 545
dense matrix, 271

derivative, 3
description methods, 480
design matrix, 232
determinant, 19, 20, 299
deviation/anomaly detection, 480
diagonal dominance, 369
diagonalizable, 289
Diagonalization Theorem, 289
diagonally dominant, 522
dictionary, 482
difference formula, $(n+1)$-point, 142
difference formula, five-point, 142
difference formula, three-point, 142
difference formula, two-point, 140
differentiable, 3
differential problem, 534
differentiation, 3
dimensionality reduction, 342
direct algebraic solvers, 258
directed graph, 365
Dirichlet BC, 556
Dirichlet boundary condition, 262
discretization, FD, 518
distance, 13, 282
divided difference, the kth, 102
divided difference, the first, 102
divided difference, the second, 102
divided difference, the zeroth, 102
divided differences, 87, 101
document weighting, 490
dot product, 13, 282
dual feasibility, 445
dual function, 394, 396, 400, 402
dual problem, 390, 393, 395, 444, 446, 450, 454
dual variable, 390
duality gap, 398
duality_convex.py, 405
dyadic decomposition, 327

edge normal direction, 250
eigenvalue, 21, 348
eigenvalue locus theorem, 367
eigenvalues of stochastic matrices, 315
eigenvector, 21, 348
elementary matrix, 15, 267
elementary reflector, 301
elementary row operations, 15, 267
elementary Simpson's rule, 155, 167
elementary stiffness matrix, 546
elementary_Info.m, 546
energy method, 574, 587
ensembling, 471
epigraph, 400, 403
equality constraints, 388
equality-constrained quadratic programming, 408
essential BC, 556
Euclidean norm, 13, 23, 282, 284
Euler method, 183, 184
Euler.mw, 188
Euler.py, 213
euler_march.m, 571
Euler_T3.mw, 191
existence and uniqueness of fixed points, 50
explicit scheme, 564, 577
exponential growth of error, 27
extended Newton divided difference, 112
extrapolation scheme, 362
Extreme Value Theorem, 5

false position method, 65
FD approximation, 519
FD scheme, 260
FDM, 517
FDM_1D.m, 524
feasible set, 391
feature expansion, 456
FEM, 534, 536
FEM, collocation method, 552
FEM, least-squares method, 551
FEM, Rayleigh-Ritz method, 549
FEM1_Galerkin.m, 547
financial freedom, 134, 416
fine grid, 150, 176
finite difference, 260
finite difference method, 257, 517, 562
finite element method, 534, 536
first divided difference, 102

first-degree spline accuracy, 116
first-derivative tests, 395, 408
first-order necessary conditions, 402, 444
five-point difference formula, 142
fixed point, 49
fixed-point iteration, 51, 58, 216
Fixed-Point Theorem, 53
floating point operation, 265
flop, 265
flux conservation, 528
for loop, 37
forward elimination, 265, 267
forward Euler method, 212, 564, 577, 578
forward substitution, 265
forward-difference, 140
Fourth-order Runge-Kutta method, 195, 513
Frobenius norm, 24, 285
FTC, 7
full rank, 297
function, 37
fundamental period of the motion, 198
Fundamental Theorem of Algebra, 67
Fundamental Theorem of Calculus, 7, 185

Galerkin method, 537, 540, 549
Gauss elimination, 257, 267
Gauss elimination with partial pivoting, 275
Gauss integration), 170, 175
Gauss-integration.mw, 171
Gauss-Lobatto integration, 175
Gauss-Lobatto points, 545
Gauss-Seidel method, 360
Gaussian elimination with complete pivoting, 276
Gaussian elimination with partial pivoting, 276
Gaussian kernel, 454
Gaussian quadrature, 167
Gaussian-Quadrature.mw, 174
GD method, 372
GECP, 276
generalized minimum residuals, 378
generalized Richardson method, 356
Generalized Rolle's Theorem, 6
GEPP, 276
get_Ab_pqf.m, 523
get_alg_system.m, 595
get_hypothesis_WLS.m, 254
ghost grid value, 260, 520
global error, 194, 195, 513
GMRES, 378, 383
GMRES.py, 384
Golub-Reinsch SVD algorithm, 333
Google, 497
Google matrix, 484, 502, 505
Google pagerank, 482
Google Pagerank algorithm, 484
Google search, 497
Google search engine, 477, 484
gradient, 372
gradient descent method, 372, 434, 462, 611
gradient magnitude, 250, 255
gradient, using sympy, 612
Gram matrix, 451
Gram-Schmidt process, 224, 297, 298
Gram-Schmidt process, modified, 378
Gram-Schmidt process, normalized, 225
graph theory, 365
great transition, 585
grid function, 572
Guess The Weight of the Ox Competition, 422
guidepoints, 128

Heat_1D_Euler.m, 570
Heat_1D_Solve.m, 580
Hermite interpolation, 110
Hermite Interpolation Theorem, 111
Hermite polynomial, 111
Hessenberg matrix, 378
Hessian, 372, 611
Hessian matrix, 611, 613
Hessian, using sympy, 612
Heun's method, 194
higher-order FEMs, 539
higher-order Taylor methods, 189
homogeneous PDEs, 572

Horner's method, 68, 86
horner.m, 70
Householder matrix, 301
Householder reflector, 301
Householder reflectors, 297
Householder transformation, 301, 333
hybrid verification, 586
hyperparameter, 436
hyperplane, 428, 429
hypothesis formation, 478

identity function, 433
image processing, 249
implicit ordinary differential equation, first-order, 216
index, 488
induced matrix 2-norm, 248
induced matrix norm, 24, 285
induction, 305
inductive reasoning, 585
inequality constraints, 389
infinity-norm, 23, 283
information retrieval, 484, 487
initial condition, 562
initial conditions, 511
initial value problem, 180, 183, 192
inlier.m, 254
inliers, 237
inlinks, 498
inner product, 13, 14, 282, 283
install Python packages, 474
integration by parts, 534
intercept, 429
interchange operation, 320
interior-point method, 412
Intermediate Value Theorem, 3
Internet search, 497
interpolating polynomials in Newton form, 84
interpolation, 226
Interpolation Error Theorem, 93, 141
Interpolation Error Theorem, Chebyshev nodes, 99
interpretability, 426
interval-halving method, 42
inverse document frequency, 490
inverse power method, 294
inverse-power.mw, 295
invertible matrix, 17, 259
invertible matrix theorem, 18, 259
involution, 302
IODE-1, 216
IODE1.py, 217
Iris_perceptron.py, 431
irreducible, 500
irreducible matrix, 365
irreducibly diagonally dominant, 522
iteration, 36
iteration matrix, 352
iterative algebraic solver, 347, 352
iterative algebraic solvers, 258
Iterative Update Sample Consensus, 242
IUSAC, 242
IVP, 180
IVT, 3

Jacobi method, 358
Jacobi polynomials, 168
Jacobian, 61
Jacobian matrix, 73

k-nearest neighbor, 457
k-NN, 457
Karush-Kuhn-Tucker conditions, 395, 402, 444
KD-tree, 458
Kepler's equation, 79
Keras, 468
kernel function, 454
kernel method, 453
kernel SVM, 453
kernel trick, 454, 456
KKT conditions, 395, 402, 408, 412, 444
KKT matrix, 408
knots, 115
Knowledge Discovery in Data, 479
Kronecker delta, 90
Krylov matrix, 381
Krylov subspace, 381
Krylov subspace method, 371

l^2-norm, 39, 284
l^∞-norm, 284
l^∞-norm, 284
Lagrange cardinal functions, 545
Lagrange dual function, 390, 393
Lagrange dual problem, 393, 395, 444
Lagrange form of interpolating polynomial, 90
Lagrange interpolating polynomial, 90, 151
Lagrange interpolation, 110
Lagrange multipliers, 388, 391, 396, 400
Lagrange polynomial, 111, 140
Lagrangian, 391, 394–396, 400, 402, 408, 444
Lagrangian (objective), 389
Laguerre polynomials, 168
latent semantic indexing, 488, 494
Lawrence Page, 502
Lax-Richtmyer Equivalence Theorem, 573
lazy learner, 457
leading principal submatrix, 269
learning compound interest, 135, 417
learning rate, 430, 434
least-square approach, 537
least-squares (LS) problem, 226
least-squares error, 228
least-squares FEM, 551
least-squares line, 231
least-squares problem, 227, 335
least-squares solution, 227, 335
left eigenvalue, 500
left singular vectors, 324, 341
Legendre orthogonal polynomials, 168
Legendre polynomials, 168, 169
length, 13, 282
Leonard Bairstow, 74
life lessons from numerical algorithms, 584
lighthouses, 586
likelihood, 440
limit, 2
linear approximation, 11
linear boundary-value problem, 511
linear convergence, 28
linear FEM, 538
linear function, 118, 121
linear Galerkin FEM, 541, 558
linear Galerkin method, 541
linear growth of error, 27
linear interpolation, 592, 600
linear shooting method, 511
linear spline, 115
linear spline accuracy, 116
linear SVM, 443, 444, 449
linear term, 556
linear_interpol.m, 595
linearization, 235, 605
link graph, 485, 501
link matrix, 485, 499, 505
linspace, in Matlab, 35
Lipschitz condition, 180, 186
little oh, 29, 30
local truncation error, 194, 195, 513
localization of roots, 67
log-likelihood function, 440
logistic cost function, 440
Logistic Regression, 440
logistic regression, 438, 439
lower bound property, 393, 394
lower-triangular matrix, 264
LS problem, 227, 335
LSI, 494
LU decomposition theorem, 269
LU factorization, 268
LU factorization algorithm, 268
lu, usage in Matlab and Python, 275
LU-factorization, 308
LU_pivot.m, 278
Lua, 468

M-matrix, 370
m-step method, 201
machine learning, 421, 422
machine learning algorithm, 422
machine learning modelcode, 469
Machine_Learning_Model.py, 469
Maclaurin series expansion, 244
majority vote, 471
maple, 4

Markov chain, 315
Markov matrix, 315, 502
mathematical induction, 305
mathematical optimization, 387
Matlab, 33
matlab: fileparts, 606
matlab: polyfit, 606
matlab: readmatrix, 607
matlab: writematrix, 607
matrix 2-norm, 248
matrix eigenvalue problem, 484
matrix equation, 258
matrix norm, 24, 285
matrix splitting, 352, 357
matrix transformation, 253
maximin problem, 393–396, 400, 444
maximum principle, 587
maximum-norm, 23, 283
mean square error, 609
Mean Value Theorem, 4, 10
Mean Value Theorem on Integral, 8
mesh points, 183
method of calculus, 230
method of collocation, 559
method of false position, 65
method of Lagrange multipliers, 388, 443
method of modified equation, 593
method of normal equations, 229, 335, 603
method of undetermined coefficients, 166, 168, 170
mini-batch, 463
minimax problem, 392, 394, 396, 400, 444
minimization principle, 537, 549
minimization problem, 536
Minkowski distance, 458
MLESAC, 242
MNIST data, 460
model, 480
modelcode, 469, 592
modern paradigm shifts, 585
Modified Euler method, 194
modified Gram-Schmidt process, 378, 379
multi-class classification, 437
multi-step methods, 201
multiple local minima problem, 426
MVT, 4, 53
myclf.py, 470
mysum.m, 37

natural BC, 556
natural cubic spline, 122, 125, 132
natural cubic splines, optimality theorem, 125
near-banded matrix, 614
nested multiplication, 68, 86, 87
network.py, 464
Neumann BC, 556
Neumann Series Theorem, 286
neuron, 427
Neville's Method, 106
Newton form of interpolating polynomials, 84
Newton form of the Hermite polynomial, 111
Newton's Divided Difference Formula, 103
Newton's method, 56, 413, 605, 611
Newton-Cotes formula, 151
Newton-Raphson method, 56
newton_horner.m, 71
NewtonRaphsonSYS.mw, 62
nltk, 488
nodal point, 365, 545
nodes, 115
nonconvex problem, 406
nonlinear least-squares problem, 605
nonlinear map, 413
nonlinear regression, 235
nonlinear SVM, 453
nonlinear_regression.m, 606
nonsingular matrix, 17, 259
nonvertical supporting hyperplane, 401, 404
norm, 13, 23, 282, 283
norm-preserving property, 382
normal equations, 229, 234, 549
normal matrix, 24, 285
normalized web matrix, 499
np.linalg.eig, 319
np.loadtxt, 607

np.polyfit, 607
np.polyval, 607
np.savetxt, 607
np.set_printoptions, 319
NR.mw, 59
null-space approach, 409
numerical differentiation, 140
numerical discretization, 260
numerical error, 569
numerical integration, 151
numerical rank, 334

objects, 33
observation vector, 232
Octave, 33
one-shot learning, 426
one-versus-all, 437
one-versus-rest, 437
open formula, 175
operator 2-norm, 248
operator norm, 24, 285
opt_sympy_Newton.py, 612
optimal ω for SOR, 364
optimal step length, 373
optimization problem, 391, 396, 400
order of convergence, 199, 200
orthogonal, 14, 282
orthogonal basis, 224, 225, 298
orthogonal decomposition, 301
orthogonal decomposition theorem, 222
orthogonal linear transformation, 338
orthogonal matrix, 274, 291, 297, 305
orthogonal polynomials, 169, 539
orthogonal projection, 222, 223, 301
orthogonal transformation, 338
orthonormal basis, 224, 225, 253, 298, 382
outer bordering, 261
outer product, 247
outliers, 237
outlinks, 485, 498
overfitting, 425

p-norm, 316
p-norms, 23, 283
Padé approximation, 243, 245
Padé rational function, 243
Pagerank, 497–499
pagerank equation, 502
Pagerank vector, 500
pagerank vector, 486
parameter estimation problem, 237
parameter vector, 232
parametric curves, 126
Parseval's identity, 574
partial pivoting, 276
partition, 115
partitioning, 518
PCA, 338
PCG, 377
perceptron, 429
perceptron.py, 431
permutation matrices, 274
permutation matrix, 274, 365, 614
Perron-Frobenius eigenvector, 317
Perron-Frobenius theory, 501
personalization vector, 503, 506
Petrov-Galerkin method, 537
physics-informed machine learning, 585
piecewise cubic Hermite interpolating polynomial, 132
piecewise cubic Hermite polynomial, 127, 128
piecewise regression, 610
PIML, 585
pizza, 566
pizza box, 566
pizza delivery, 566, 567, 572, 583, 584
plot, in Matlab, 34
point reordering, 366
polar decomposition, 342
polynomial fitting, 603
Polynomial Interpolation Error Theorem, 93, 104, 131
Polynomial Interpolation Theorem, 83
polynomial of degree n, 67, 72
polynomial-fitting, 608
Porter Stemming Algorithm, 489
positive definite, 371, 544
positive semidefinite, 249
positive semidefinite matrix, 339

power iteration, 287
power method, 287, 318, 504
power-method.mw, 290
power_method.m, 290
precision, 492
preconditioned CG method, 377
prediction methods, 480
preprocessing, 488
primal feasibility, 408, 445
principal component analysis, 338
principal components, 339
probabilistic model, 439
probability matrix, 315
probability vector, 315
programming, 33
PROSAC, 242
pseudocode, 27
pseudoinverse, 229, 335, 337
pseudoinverse, the k-th, 337
Pythagorean Theorem, 14, 282
Pythagorean theorem, 223

Q9, 487
QR algorithm, 312, 333
QR decomposition, 322, 345
QR factorization, 297
QR factorization by Gram-Schmidt process, 298
QR factorization by reflectors, 308
QR Factorization Theorem, 297
QR iteration, 312
qr_algorithm.m, 313
quadratic convergence, 28
quadratic formula, 250
quadratic programming, 407, 446, 451
quadratic programming, equality-constrained, 408
quadratic programming, general form, 407
quadratic programming, inequality-constrained, 412
quadratic spline, 117, 132
quasi-Newton method, 611
quasi-Newton methods, 613
query matching, 492
quotient, 72

R-RANSAC, 242
radial basis function, 454
random sample consensus, 239
range-space approach, 409
rank-1 matrix, 247
rank-one matrix, 247
RANSAC, 239
ransac2.m, 254
rational function, 243
Rayleigh quotient, 339, 380
Rayleigh-Ritz method, 537
Rayleigh-Ritz method, FEM, 549
real-life examples, 584
recall, 492
recall versus precision diagram, 493
rectifier, 438
recurrence relation, 74, 315
recursive Trapezoid rule, 161, 162
reduced echelon form, 15
reduced Hessian, 408
reducible matrix, 365
regression, 480
regression analysis, 231, 602
regression coefficients, 231
regression line, 231
regular, 318
regular splitting, 370, 409
regular stochastic matrix, 318
relationship, 566
relative error, 26
relaxation methods, 357
remainder, 72
Remainder Theorem, 68
repetition, 33, 36
replacement, 317
residual, 540, 551
reusability, 37
reusable, 33
Rich Dad Poor Dad, 136, 418
Richardson extrapolation, 145, 146, 163, 525, 590, 598–600
Richardson method, 355
Richardson's method, 372

Richardson_extrapolation.m, 596
Riemann integral, 7
right eigenvalue, 500
right singular vectors, 324, 341
RK2, 193, 194
RK3, 216
RK4, 195, 513
RK4.mw, 196
RK4SYS.mw, 207
RK4SYSTEM.mw, 210, 514
RKF45, 197
Robert Kiyosaki, 136, 418
Rolle's Theorem, 3, 6
Romberg algorithm, 163
Romberg integration, 161, 163
rotational symmetry, 438
Run_network.py, 466
Runge's phenomenon, 114
Runge-Kutta method, third-order, 216
Runge-Kutta methods, 192
Runge-Kutta-Fehlberg method, 197

scaling factor, 356
scene analysis, 237
Schur decomposition, 312
score matrix, 339, 341
search direction, 371, 434
search engine, 484
secant method, 63
second divided difference, 102
second-derivative midpoint formula, 143
second-order FDM, 520
Second-order Runge-Kutta method, 193, 194
self-improvement, 584
self-referencing, 497
Semi-Discrete Equation, 577
semi-discrete equation, 563
semi-discrete problem, 563
semi-implicit method, 579
sequence acceleration method, 590
Sequential Minimal Optimization, 446
sequential pattern discovery, 480
Sergey Brin, 502
shape function, 542
sigmoid function, 438
significant digits, 26
similarity function, 454
similarity transformation, 365, 614
Simpson's rule, 155
Simpson's three-eights rule, 159
Sine_Noisy_Data_Regression.m, 607
singular value decomposition, 324, 484, 494
singular values, 324, 341
skew-symmetric, 39
sklearn.ensemble.VotingClassifier, 471
sklearn_classifiers.py, 471
slack variable, 412, 448
Slater's condition, 398
Slater's Theorem, 398
SMO, 446
smoothing assumption, 237
sobel_derivatives.m, 252
soft-margin classification, 448
softplus function, 438
SOR, 357, 362
SOR, optimal ω, 364
source vector, 541
space-time slice, 562
spaCy, 488
sparse, 491
sparse eigenvalue algorithms, 504
sparse matrix, 271
spectral radius , 350
Spectral Theorem, 291
spline, 537
spline of degree k, 115
spring-mass system, 198
square root, 59
SSE, 428
stability, 569, 572, 576
stability analysis, 579
stability condition, 571, 575
stability in life, 576
stable, 27, 572
stable equilibrium distribution, 317
standard logistic sigmoid function, 438
state vector, 315
steady-state vector, 317

steepest descent method, 372
stemming, 489
step length, 371, 434, 441
step-by-step methods, 182
stiffness matrix, 543, 546
stochastic gradient descent, 463
stochastic matrix, 315, 502
stochastic_eigen.py, 319
stop words, 489
stopping criterion, 46, 354
stopping tolerance, 293
strong duality, 397, 404
strong form, 535, 536
strongly connected, 366, 500
structure tensor, 247, 249
structure_tensor.m, 251
submatrix, 20
subordinate 2-norm, 248
subordinate norm, 24, 285
substitution matrix, 315
successive over relaxation, 357, 362
successive under relaxation, 364
Sum of Squared Errors, 428
summary of SVM, 452
super-convergence, 59
supergraph, 400
superlinear convergence, 28
supervised learning, 423
Support Vector Machine, 442
support vectors, 447
SUR, 364
SVD, 324, 341, 345, 484
SVD theorem, 324
SVD, algebraic interpretation, 326
SVD, geometric interpretation, 328
SVM, 442
SVM summary, 452
SVM, nonlinear, 453
symmetric factorization, 410
symmetric inverse power method, 296
symmetric positive definite, 371
symmetric power method, 291
symmetric-inverse-power.mw, 296
symmetric-power-method.mw, 292
sympy.diff, 612
sympy.Matrix, 612
sympy.symbols, 612
synthetic division, 68
systems of nonlinear equations, 60

tangent line, 57
tangent plane approximation, 11
Taylor method of order m, 189, 218
Taylor series, 143, 150, 183, 260, 519
Taylor's formula, 562
Taylor's series formula, 145, 598
Taylor's Theorem, 9, 157
Taylor's Theorem for Two Variables, 11
Taylor's Theorem with Integral Remainder, 10
Taylor's Theorem with Lagrange Remainder, 9, 94, 517
Taylor's Theorem, Alternative Form of, 11, 517
Taylor-series methods, 183
teleportation, 503
term, 482
term frequency, 490
term weighting scheme, 490
term-document matrix, 482, 483, 488, 490
terms, 488
test function, 534
test_data_100.m, 604
text mining, 482, 487
textmineR, 488
theory-first, 585
theta(θ)-method, 578
theta_march.m, 581
three tasks, 422
three-point difference formula, 142
three-point endpoint formulas, 142
three-point midpoint formula, 142, 145, 598
three-term recurrence relation, 169
topic examples, 135, 417
training data, 423
transforming null-space iteration, 409
transforming range-space iteration, 409
transition matrix, 315
trapezoid method, 215

Trapezoid quadrature, 161
trapezoid rule, 152, 543
trial function, 540
triangular systems, 264
truncated score matrix, 342
truncation error, 566
two-point difference formula, 140

unconditionally stable, 578, 580
unique inverse, 17, 259
unit lower triangular matrix, 268
unit lower-triangular matrix, 271
unit round-off error, 334
unit vector, 301
unstable, 27
unsupervised learning, 424
update direction, 434
upper bidiagonal matrix, 333
upper Hessenberg matrix, 378
upper triangular matrix, 264, 266, 271, 297, 305

variational principle, 534
variational problem, 535, 556
vector norm, 23, 283
vector space model, 488
visual inspection, 602
volume scaling factor, 19
von Neumann analysis, 574
VotingClassifier, 471

weak duality, 397
weak form, 534, 536, 556
web search engines, 487
Weierstrass approximation theorem, 82, 114
weight function, 540
weight matrix, 237, 341
weighted least-squares method, 237
Weighted Mean Value Theorem on Integral, 8, 152, 157
weighted normal equations, 238
weighted residual approach, 537, 540
weighted sum, 151
WMVT, 8

x-intercept, 57

zero-over-zero, 112
zeroth divided difference, 102