Numerical Analysis and Applications

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

Updated: July 4, 2023

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
July 4, 2023

Contents

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

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)

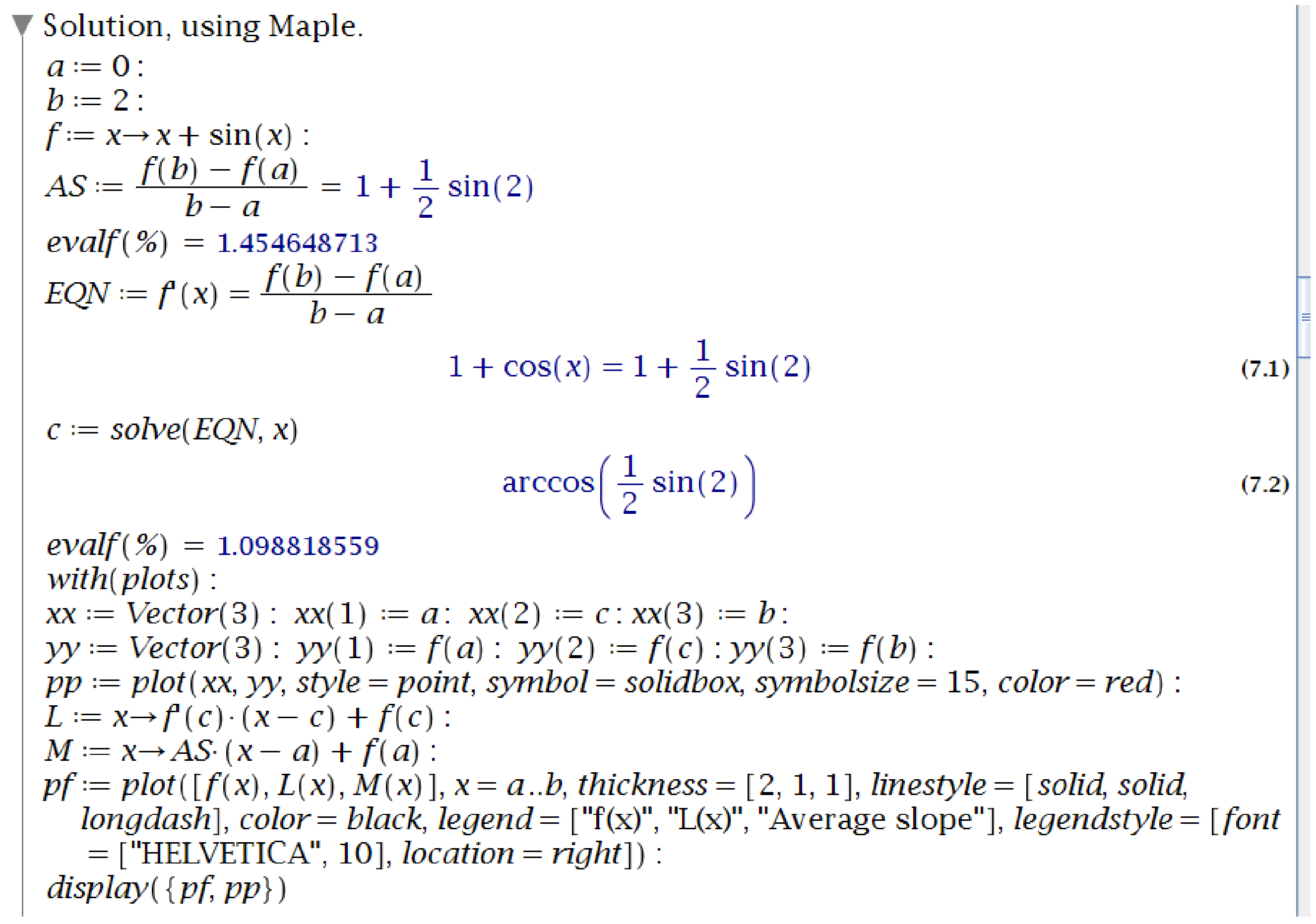
Theorem 1.11. *Suppose $f \in C[a,b]$ and f is differentiable on (a,b). Then there exists a number $c \in (a,b)$ such that*

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

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 + 1/2 sin(2)                    (7.1)
c := solve(EQN, x)
                    arccos(1/2 sin(2))                             (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.

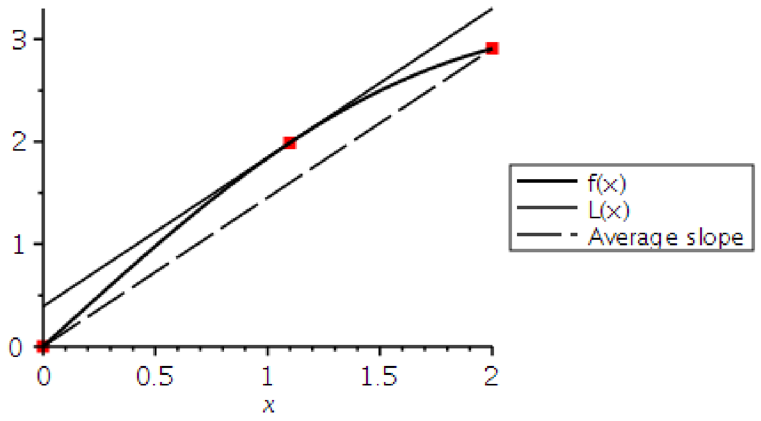


Figure 1.2: The resulting figure, from the implementation in Figure 1.1.

Theorem 1.13. ***(Extreme Value Theorem)**: If $f \in C[a,b]$, then there exist $c_1, c_2 \in [a,b]$ for $f(c_1) \le f(x) \le f(c_2)$ 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
```

```

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

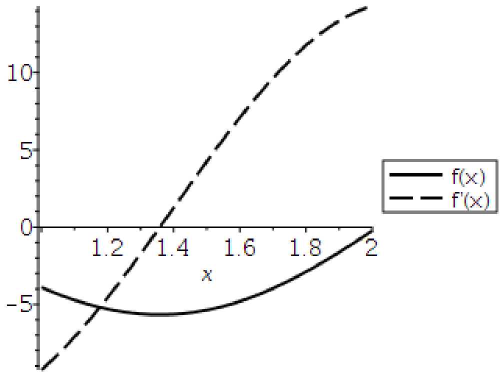


Figure 1.3: The figure from the Maple-code.

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

$\boxed{\textbf{Theorem}}$ **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* $a \le x_0 < x_1 < \cdots < x_n \le b$, *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])
```

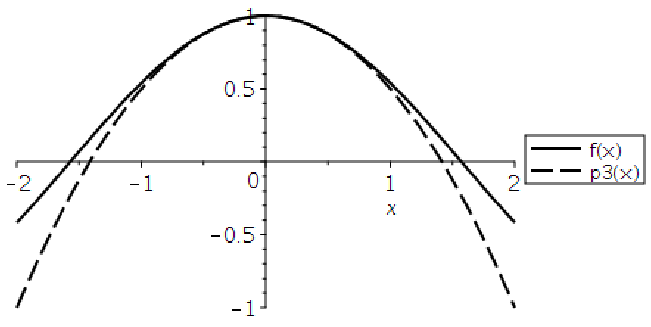


Figure 1.4: $f(x) = \cos x$ and its third Taylor polynomial $P_3(x)$.

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

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

which is the **Mean Value Theorem**.

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

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

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 u and 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 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 u and v is

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

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 $\mathit{dist}(\mathbf{u}, \mathbf{v})$.

Solution.

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

Theorem 1.33. ***Pythagorean Theorem**: Two vectors* u *and* 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 **u** and **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 u and 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 $y = a_0 + a_1 x + a_2 x^2$ 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 ***det*** A or $|A|$.

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

2) Let $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \in \mathbb{R}^{2\times 2}$. Then ***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 ***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

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

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:

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

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

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

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

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

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 x such that $A\mathrm{x} = \lambda\mathrm{x}$ for some scalar λ. In this case, a scalar λ is an **eigenvalue** and 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

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

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

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

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.

$\square$

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 $\lambda \in \mathbb{R}$,

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

Example 1.55. The most common norms are

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

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

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

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,*

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

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

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

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

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

f. $\|A\|_2 = \max_i |\lambda_i(A)|$, *when $A^TA = AA^T$ (**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
$$|x_{n+1} - x^*| \le c_1 |x_n - x^*|, \;\; \forall \;\; n \ge N. \tag{1.41}$$
- 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
$$|x_{n+1} - x^*| \le \varepsilon_n |x_n - x^*|, \;\; \forall \;\; n \ge N. \tag{1.42}$$
- 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
$$|x_{n+1} - x^*| \le C |x_n - x^*|^2, \;\; \forall \;\; n \ge N. \tag{1.43}$$
- 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
$$|x_{n+1} - x^*| \le C |x_n - x^*|^\alpha, \;\; \forall \;\; n \ge N. \tag{1.44}$$

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 $n \ge 1$ ($\because$ $x_{n+1}^2 - 2 > 0$). 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 "$\boldsymbol{\alpha_n}$ **is in** $\boldsymbol{\mathcal{O}(\beta_n)}$" and denote $\boldsymbol{\alpha_n \in \mathcal{O}(\beta_n)}$ or $\boldsymbol{\alpha_n = \mathcal{O}(\beta_n)}$.

- 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 "$\boldsymbol{\alpha_n}$ **is in** $\boldsymbol{o(\beta_n)}$" and denote $\boldsymbol{\alpha_n \in o(\beta_n)}$ or $\boldsymbol{\alpha_n = o(\beta_n)}$.

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.

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

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

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

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

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,

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

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 $f(h) = \frac{1}{h}(1 + h - e^h)$. 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'
```

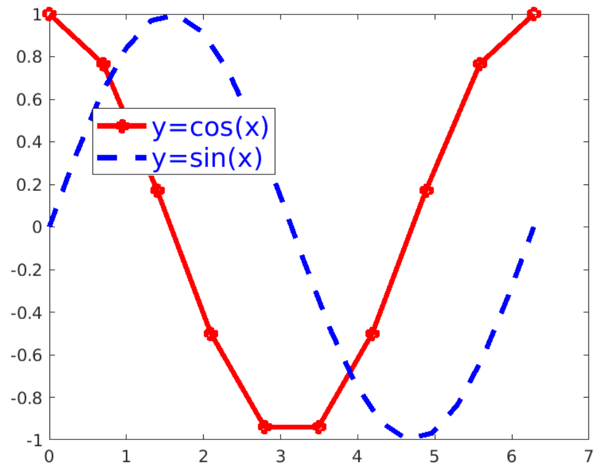


Figure 1.5: fig_cos_sin.png: `plot` of $y = \cos x$ and $y = \sin x$.

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

```
function s = mysum(n1,n2)
% sum of integers from n1 to n2

s=0;
for i=n1:n2
    s = s+i;
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. $\boxed{\text{C}}$[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 $\boxed{\text{C}}$ 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 $\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1$ 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

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 $[a, b]$.

$\boxed{\textbf{Pseudocode}}$ **2.1. The Bisection Method**:

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

      if ( $f(p_n) = 0$ ) then
          stop;
      elseif ( $f(a_n) \cdot f(p_n) < 0$ ) 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)
```

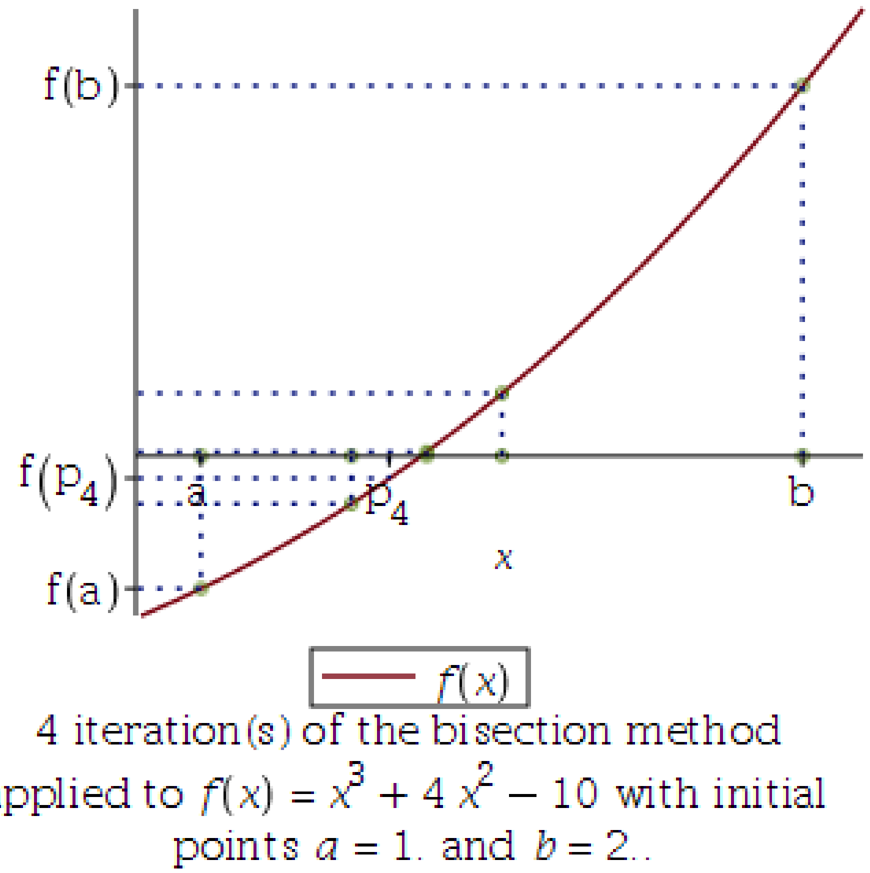
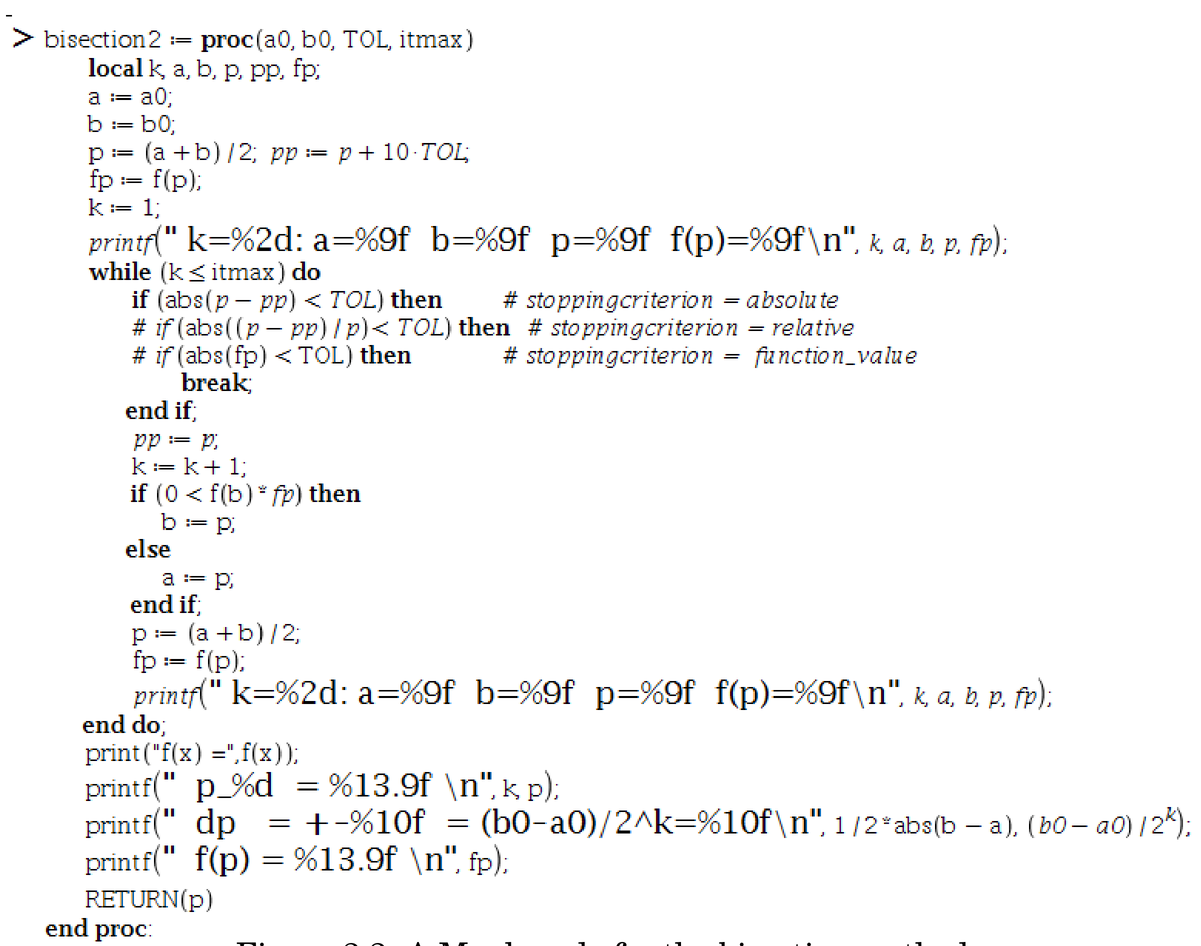


Figure 2.1: The bisection method.



```
> 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*

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

Proof. For $n \ge 1$,

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

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

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

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),

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

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

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

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

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

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: $n \ge \ln(10^8/3)/\ln(2)$. 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 $\lim_{n\to\infty} \frac{|p-p_{n+1}|}{|p-p_n|}$ 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
```

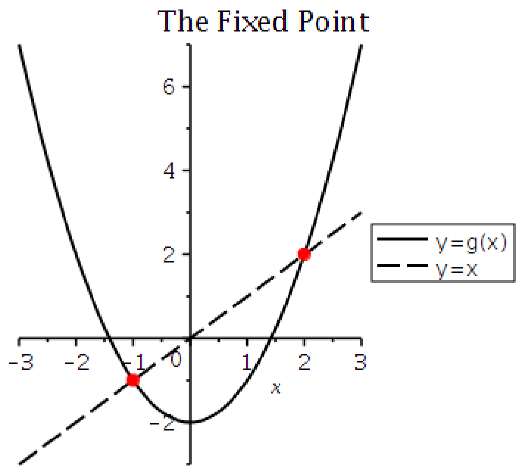


Figure 2.3: Fixed points of $g(x) = x^2 - 2$.

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*

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

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

Proof.

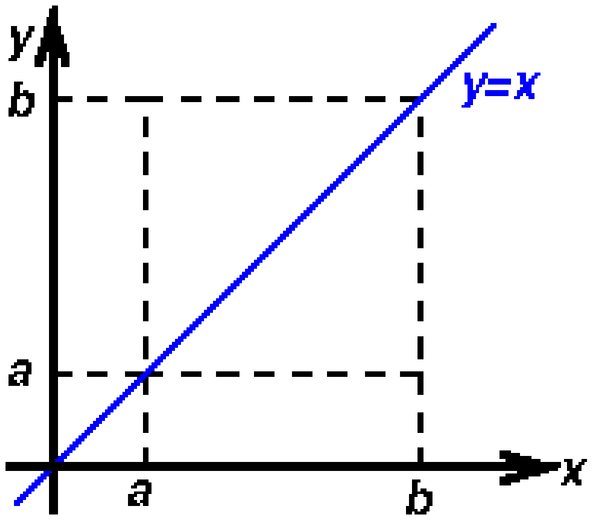


Figure 2.4: Illustration of the existence-and-uniqueness theorem.

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) $x = g_2(x) = \dfrac{1}{4}\Big(\dfrac{10}{x} - x^2\Big) \qquad \Leftarrow x^2 + 4x - \dfrac{10}{x} = 0$

(3) $x = g_3(x) = \Big(\dfrac{10}{x} - 4x\Big)^{1/2}$

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

(5) $x = g_5(x) = \Big(\dfrac{10}{x+4}\Big)^{1/2} \qquad \Leftarrow x^2(x + 4) - 10 = 0$

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

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
```

```

g6 := x -> x - (x^3 + 4*x^2 - 10)/(3*x^2 + 8*x):
maximize(diff(g6(x), x), x = 1..2);
                                5
                                --
                                14
maximize(-diff(g6(x), x), x = 1..2);
                               70
                               ---
                               121
FixedPointIteration(x-g6(x), x=1.5, tolerance=10^-3, output=sequence);
           1.5, 1.373333333, 1.365262015, 1.365230014
```

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

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

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

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

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

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 $n \ge 1$. It follows from the *MVT* that

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

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

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

  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.

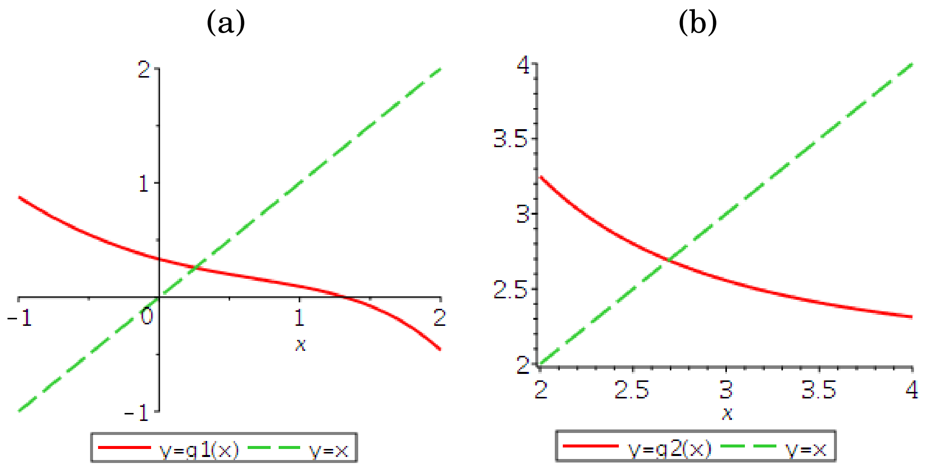


Figure 2.5: Visualization of the functions.

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).

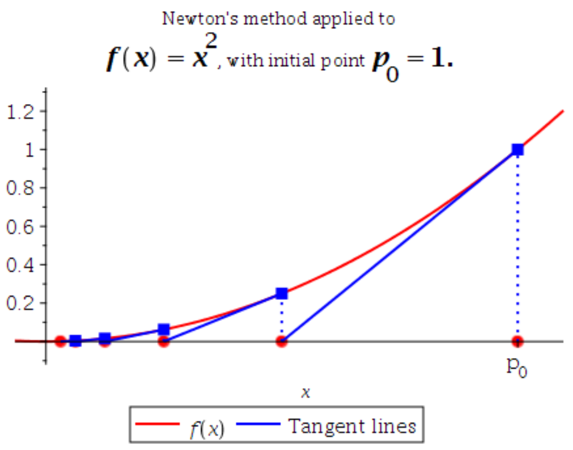
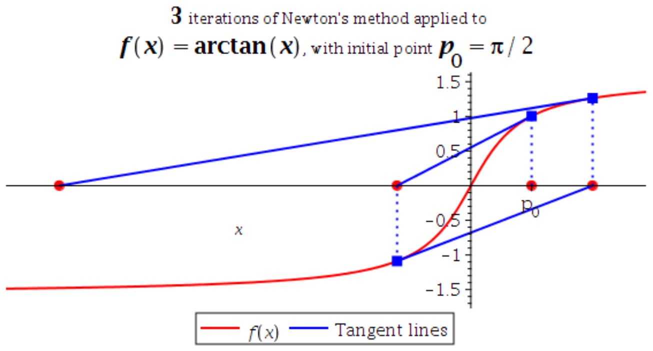


Figure 2.6: Graphical interpretation of the Newton's method.

An Example of Divergence
```
f := arctan(x);
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*

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

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

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

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

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

(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 *non-linear 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

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

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

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

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);
```

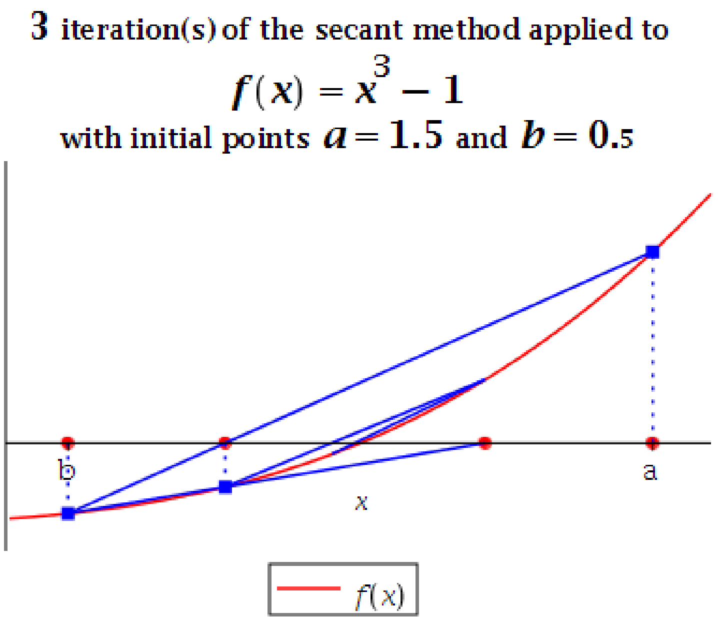


Figure 2.7: Graphical interpretation of the secant method.

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);
```

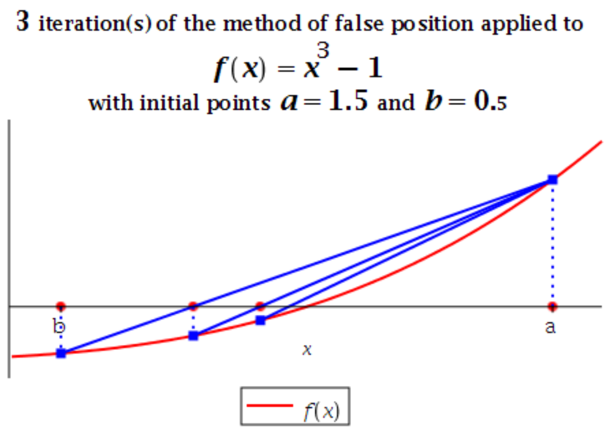


Figure 2.8: Graphical interpretation of the false position method.

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*

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

- **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.

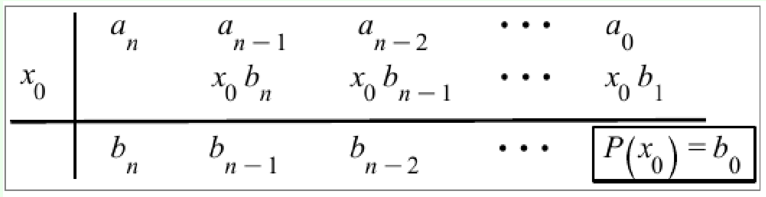
Algorithm **2.35. Let us try to evaluate $P(x)$ at $x = x_0$.**

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

- Introducing $b_0 = P(x_0)$, the above can be rewritten as
$$b_{n+1} = 0; \quad b_k = a_k + x_0 b_{k+1}, \quad n \geq k \geq 0. \tag{2.53}$$

- 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**):

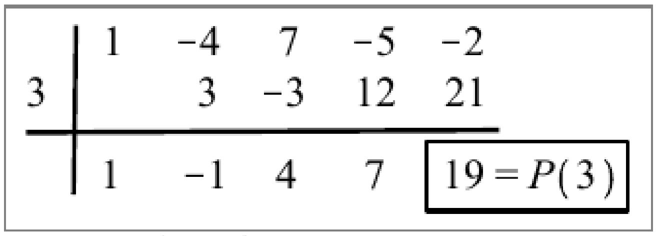


	a_n	a_{n-1}	a_{n-2}	$\cdots$	a_0
x_0		$x_0 b_n$	$x_0 b_{n-1}$	$\cdots$	$x_0 b_1$
	b_n	b_{n-1}	b_{n-2}	$\cdots$	$\boxed{P(x_0) = b_0}$

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	$\boxed{19 = P(3)}$

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

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

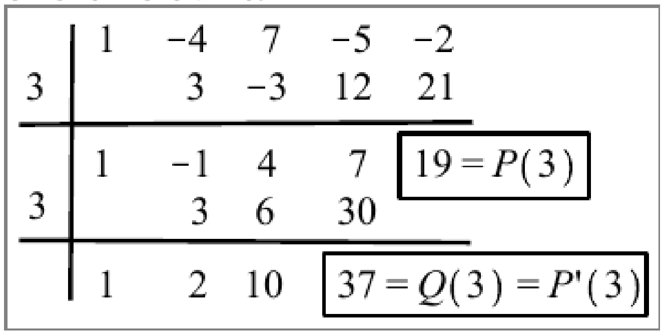
- Thus

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

  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:



$$\begin{array}{c|rrrrr}
 & 1 & -4 & 7 & -5 & -2 \\
3 & & 3 & -3 & 12 & 21 \\
\hline
 & 1 & -1 & 4 & 7 & \boxed{19 = P(3)} \\
3 & & 3 & 6 & 30 & \\
\hline
 & 1 & 2 & 10 & \boxed{37 = Q(3) = P'(3)} &
\end{array}$$

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
```

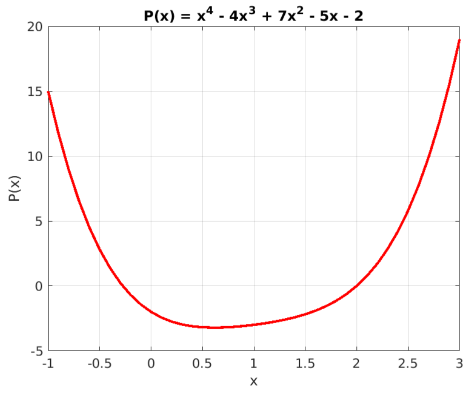


Figure 2.9: Polynomial $P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2$. Its two zeros are -0.275682 and 2.

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.
$$\begin{aligned} b_{n+1} &= b_{n+2} = 0 \\ b_k &= a_k + u b_{k+1} + v b_{k+2}, \quad n \ge k \ge 0. \end{aligned} \tag{2.60}$$

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

Bairstow's 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 $p = \lim\limits_{n\to\infty} p_n$. 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) $|p - p_n| = \mathcal{O}\Big(\dfrac{1}{2^n}\Big)$ as $n \to \infty$

Ans: (c)

2.2. $\boxed{\text{C}}$ 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 $f(p_{n-1}) = e_{n-1} f'(p_{n-1}) - \dfrac{1}{2} e_{n-1}^2 f''(\xi_{n-1})$.

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. $\boxed{\text{C}}$ 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

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

$$f := x \mapsto e^x$$
$$taylor(f(x), x = 0, 7) = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6 + O(x^7)$$
$$p0 := x \to 1:$$
$$p2 := x \to 1 + x + \frac{1}{2}x^2:$$
$$p4 := x \to 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4:$$
$$p6 := x \to 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \frac{1}{24}x^4 + \frac{1}{120}x^5 + \frac{1}{720}x^6:$$

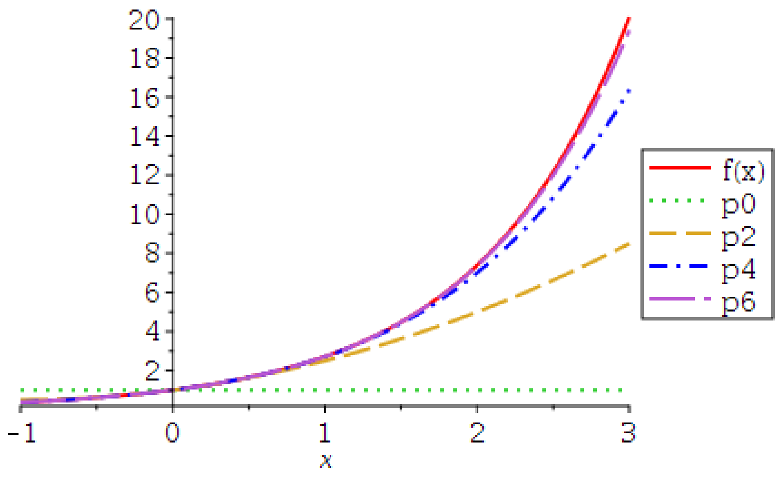


Figure 3.1: Polynomial approximations for $f(x) = e^x$.

$\boxed{\textbf{Theorem}}$ **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 $\displaystyle\prod_{j=0}^{m}(x - x_j) = 1$ 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 $\{(x_i, f(x_i)) \mid i \le k\}$ 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)
```

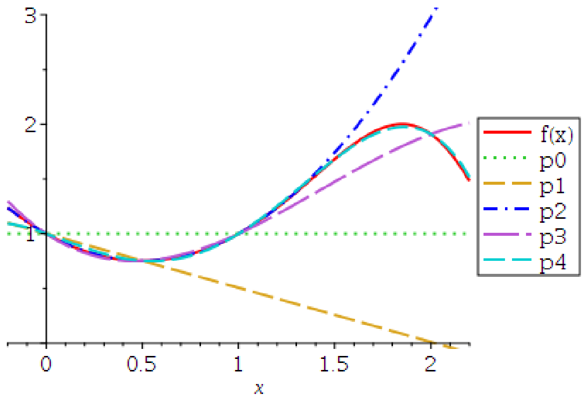


Figure 3.2: Illustration of Newton's interpolating polynomials, with $f(x) = \sin(x\cdot(x-1))+1$, at $[0,\ 0.5,\ 1.0,\ 2.0,\ 1.5]$.

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

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

$\boxed{\textbf{Algorithm}}$ **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

$\boxed{\textbf{Algorithm}}$ **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

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

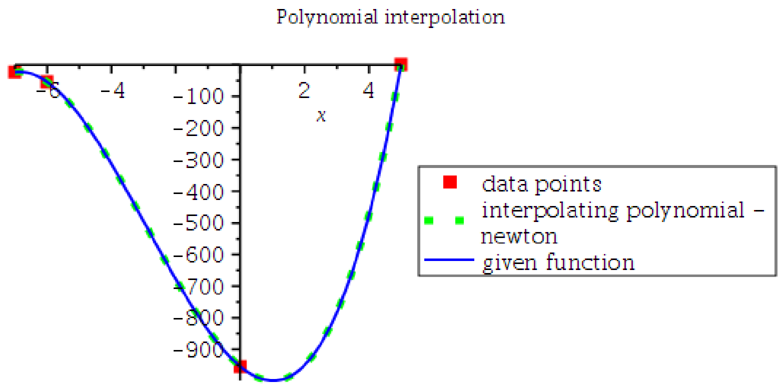
Four values of this function are given as

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***
$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \ne j. \end{cases}$$
- 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 $\le n$ 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

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

```
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,

$$\begin{aligned} |\sin(x) - P_5(x)| = \left|\frac{f^{(6)}(\xi)}{6!}\prod_{i=0}^{5}(x-x_i)\right| &\le \frac{\sin(1)}{6!}\text{gmax} \\ &= 0.00008090517158 \end{aligned} \tag{3.26}$$

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 $\le n$ that interpolates f at*

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

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

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

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

$$\max_{x\in[a,b]} \prod_{j=1}^{n} |x - x_i|.$$

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

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

Now note that

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

Thus

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

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

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

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

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

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
```

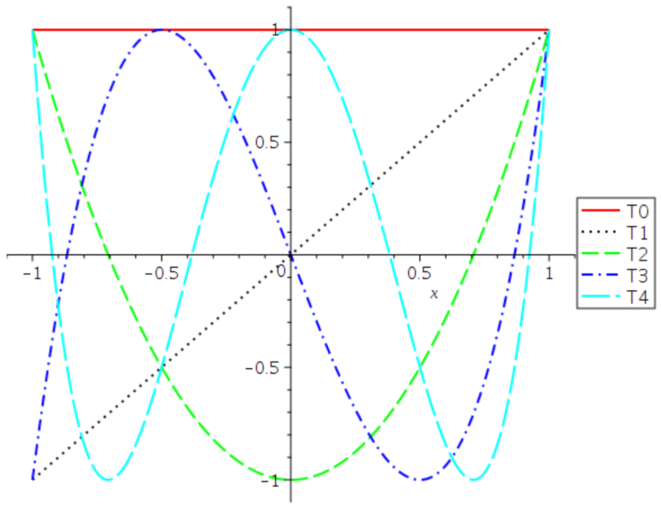


Figure 3.3: Chebyshev polynomials.

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:

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

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

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

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*

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

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

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

Thus

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

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])
```

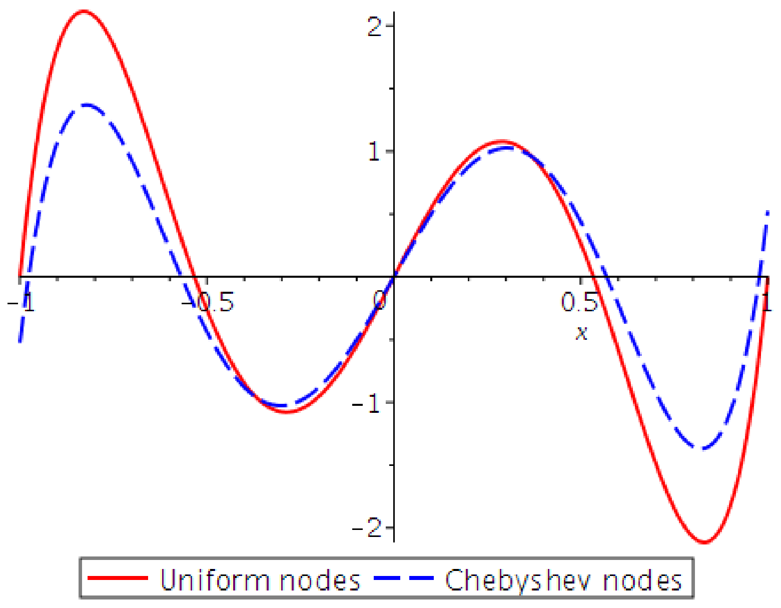


Figure 3.4: Accuracy comparison between uniform nodes and Chebyshev nodes.

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 $\boldsymbol{x_i}$, 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 $\boldsymbol{x_i, x_{i+1}}$ 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 $\boldsymbol{x_i, x_{i+1}, x_{i+2}}$ 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 ***k*th divided difference** relative to $\boldsymbol{x_i, x_{i+1}, \cdots, x_{i+k}}$ 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 $j = 1, 2, \cdots, i$
                  $F_{i,j} = \dfrac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$
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 $\le n$ 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 $0 \le m_i \le n$ 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 $0 \le i < j \le n$,*

$$P_{i,i+1,\cdots,j}(x) = \frac{(x-x_i)P_{i+1,i+2,\cdots,j}(x) - (x-x_j)P_{i,i+1,\cdots,j-1}(x)}{x_j - x_i}, \qquad (3.60)$$

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,

$$\begin{aligned} P_{0,1}(x) &= \frac{(x-x_0)P_1(x) - (x-x_1)P_0(x)}{x_1 - x_0} \\ P_{1,2}(x) &= \frac{(x-x_1)P_2(x) - (x-x_2)P_1(x)}{x_2 - x_1} \\ P_{0,1,2}(x) &= \frac{(x-x_0)P_{1,2}(x) - (x-x_2)P_{0,1}(x)}{x_2 - x_0} \end{aligned} \qquad (3.61)$$

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 $\varepsilon$;
       { 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 $j = 1, 2, \cdots, i$
                  $Q_{i,j} = \dfrac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}}$
              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

$$\begin{array}{rcl} f(x_0) &=& a \\ f'(x_0) &=& b \\ f(x_1) &=& a + bh + ch^2 \qquad (h = x_1 - x_0) \\ f'(x_1) &=& b + 2ch + dh^2 \end{array} \tag{3.68}$$

  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.

(3.76)

z	**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
$z_4 = x_2$	$f[z_4] = f(x_2)$	$f[z_3, z_4] = \dfrac{f[z_4] - f[z_3]}{z_4 - z_3}$	
$z_5 = x_2$	$f[z_5] = f(x_2)$	$f[z_4, z_5] = \mathbf{f'(x_2)}$	

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:

$$\lim_{n\to\infty} \Big(\max_{-1\le x\le 1} |f(x) - p_n(x)| \Big) = \infty. \tag{3.79}$$

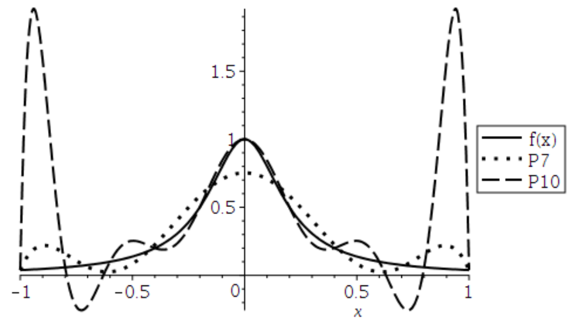


Figure 3.5: Runge's phenomenon.

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

$$L(x) = \begin{cases} 1.3 + 8.5x, & x < 0.2 \\ \dfrac{11}{3} - \dfrac{10x}{3}, & x < 0.5 \\ \dfrac{13}{6} + \dfrac{x}{3}, & x < 0.8 \\ 0.5 + 2.0x, & \text{otherwise} \end{cases} \tag{3.81}$$

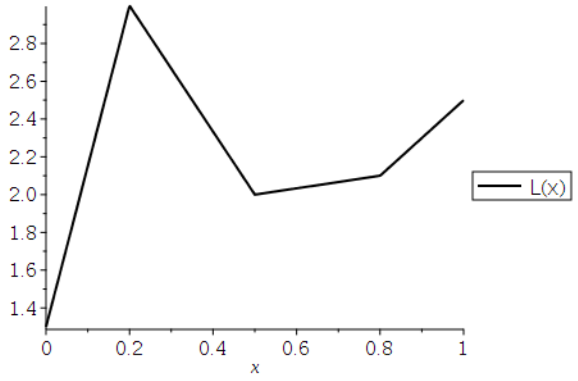


Figure 3.6: Linear spline.

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*

$$|f(x) - p(x)| \le \frac{M_2}{8} \max_{1 \le i \le n} (x_i - x_{i-1})^2, \quad x \in [a, b], \tag{3.83}$$

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 $x = b$.

$\boxed{\textbf{Algorithm}}$ **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*)

$$\boxed{Q_i(x) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x - x_{i-1})^2 + z_{i-1}(x - x_{i-1}) + y_{i-1}.} \tag{3.91}$$

(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

$$\boxed{z_i = 2\frac{y_i - y_{i-1}}{x_i - x_{i-1}} - z_{i-1}, \quad i = 1, 2, \cdots, n.} \tag{3.93}$$

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
```

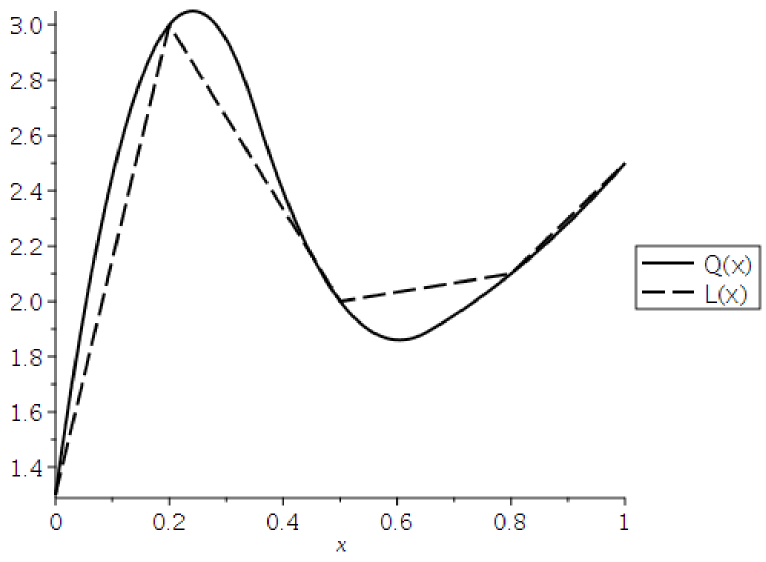


Figure 3.7: The graph of $Q(x)$ is superposed over the graph of the linear spline $L(x)$.

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 $[x_{i-1}, x_i]$, $1 \le i \le n$, 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' :	$n-1$	(3.96)
continuity of S'' :	$n-1$	

  Thus there are $(4n - 2)$ **equations**.
- **Two degrees of freedom remain**, and there have been various ways of choosing them to advantage.

$\boxed{\textbf{Algorithm}}$ **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

$$\begin{aligned} S_i(x) &= \frac{z_{i-1}(x_i - x)^3}{6h_i} + \frac{z_i(x - x_{i-1})^3}{6h_i} \\ &\quad + \Big(\frac{y_i}{h_i} - \frac{1}{6}z_i h_i\Big)(x - x_{i-1}) + \Big(\frac{y_{i-1}}{h_i} - \frac{1}{6}z_{i-1}h_i\Big)(x_i - x). \end{aligned} \tag{3.102}$$

(3) The values $z_1, z_2, \cdots, z_{n-1}$ can be determined from the *continuity of S'*:

$$\begin{aligned} S_i'(x) &= -\frac{z_{i-1}(x_i - x)^2}{2h_i} + \frac{z_i(x - x_{i-1})^2}{2h_i} \\ &\quad + \Big(\frac{y_i}{h_i} - \frac{1}{6}z_i h_i\Big) - \Big(\frac{y_{i-1}}{h_i} - \frac{1}{6}z_{i-1}h_i\Big). \end{aligned} \tag{3.103}$$

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.

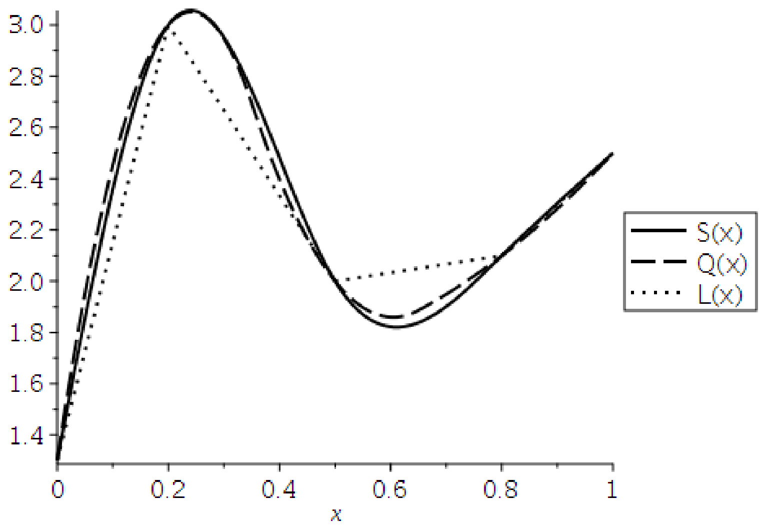


Figure 3.8: The graph of $S(x)$ is superposed over the graphs of the quadratic spline $Q(x)$ and the linear spline $L(x)$.

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   /
```

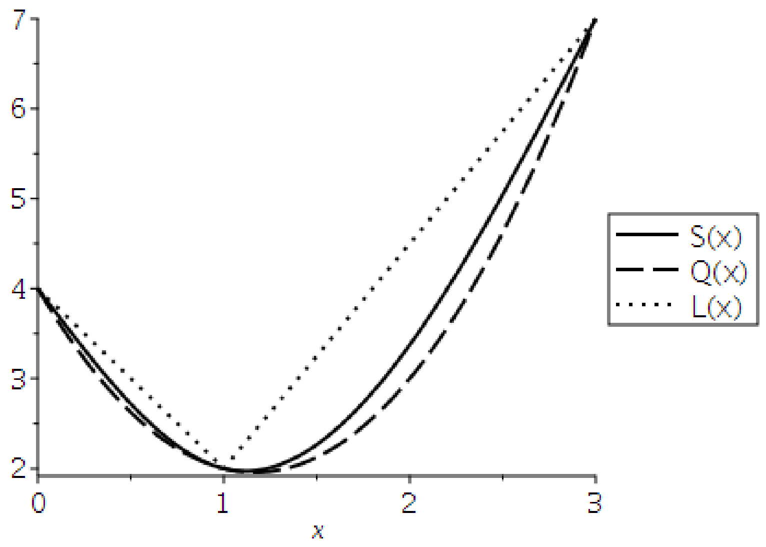


Figure 3.9: Splines on two subintervals.

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 $0 \le i \le n$, then*

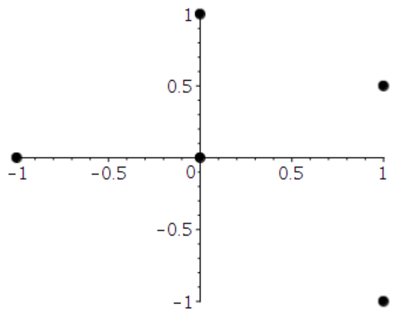
$$\int_a^b [S''(x)]^2\,dx \le \int_a^b [f''(x)]^2\,dx. \tag{3.110}$$

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)
```

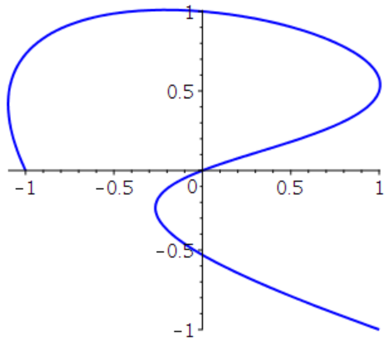


Figure 3.10

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:

$(x_0 + \alpha_0, y_0 + \beta_0)$:	guidepoint for (x_0, y_0)
$(x_1 - \alpha_1, y_1 - \beta_1)$:	guidepoint for (x_1, y_1)

(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
  ⇒ 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)
  ```

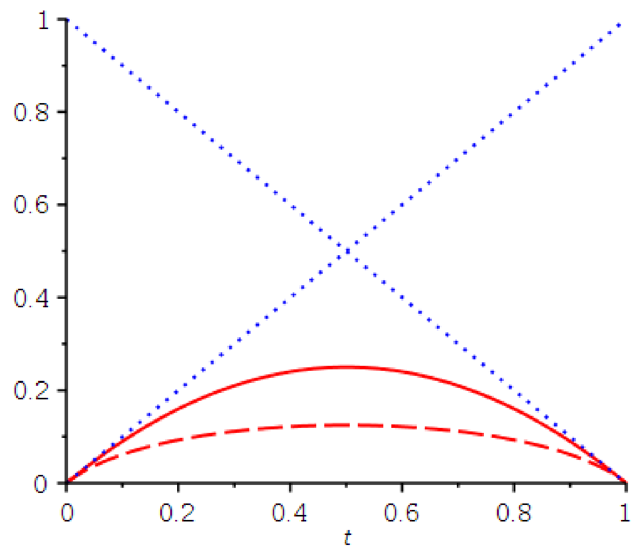


Figure 3.11: The parametric curves: $H_1(t)$ and $H_2(t)$.

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.

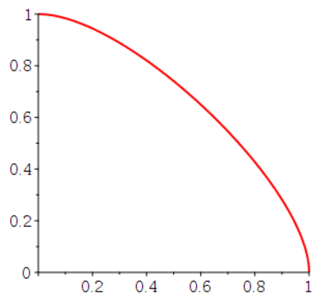
3.9. Construct the **piecewise cubic Hermite interpolating polynomial** for

x	$f(x)$	$f'(x)$
0	2	-9
1	-4	4
2	4	12

Ans: $H_1(x) = 2 - 9x - 3x^2 + 13x^2(x-1)$ 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.

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	$f(x) \approx P_n(x)$ piecewise $\Rightarrow \int_a^b f(x) \approx \int_a^b P_n(x)$
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

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

$$\begin{aligned} f'(x) \;&=\; \sum_{k=0}^{n} f(x_k) L'_{n,k}(x) + \frac{d}{dx}\Big(\frac{f^{(n+1)}(\xi)}{(n+1)!}\Big) \prod_{k=0}^{n} (x - x_k) \\ &\quad + \frac{f^{(n+1)}(\xi)}{(n+1)!} \frac{d}{dx}\Big(\prod_{k=0}^{n} (x - x_k)\Big). \end{aligned} \tag{4.8}$$

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

$$\begin{aligned} L_{2,0}(x) &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, \quad L_{2,1}(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}, \\ L_{2,2}(x) &= \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}. \end{aligned}$$

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. $f(x) = P_n(x) + R_n(x), \;\; P_n(x) \in \mathbb{P}_n$
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. 141, 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*:

$$\begin{aligned} f'(x) &= \frac{f(x+h)-f(x-h)}{2h} - \Big[\frac{h^2}{3!}f^{(3)}(x) + \frac{h^4}{5!}f^{(5)}(x) + \frac{h^6}{7!}f^{(7)}(x) + \cdots\Big], \\ f''(x) &= \frac{f(x-h)-2f(x)+f(x+h)}{h^2} - \Big[2\frac{h^2}{4!}f^{(4)}(x) + 2\frac{h^4}{6!}f^{(6)}(x) + \cdots\Big]. \end{aligned} \tag{4.14}$$

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):

$$M = N(h) + \mathbf{K_2 h^2} + K_4h^4 + K_6h^6 + \cdots . \tag{4.18}$$

1. Let's write out (4.18) with h replaced by $h/2$:

$$M = N(h/2) + \underline{\mathbf{K_2h^2/4}} + K_4h^4/16 + K_6h^6/64 + \cdots . \tag{4.19}$$

   Then the leading term in the error series, K_2h^2, can be eliminated as follows:

$$\begin{array}{rclll} M &=& N(h) & +K_2h^2 + K_4h^4 + K_6h^6 + \cdots \\ 4M &=& 4N(h/2) & +K_2h^2 + K_4h^4/4 + K_6h^6/16 + \cdots \\ \hline 3M &=& 4N(h/2) - N(h) & \quad -\frac{3}{4}K_4h^4 - \frac{15}{16}K_6h^6 - \cdots \end{array} \tag{4.20}$$

   Thus we have

$$M = \frac{1}{3}[4N(h/2) - N(h)] - \frac{1}{4}K_4h^4 - \frac{5}{16}K_6h^6 - \cdots . \tag{4.21}$$

   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

$$M = N_2(h) - \frac{1}{4}K_4h^4 - \frac{5}{16}K_6h^6 - \cdots . \tag{4.22}$$

   Then, similarly,

$$M = N_2(h/2) - \frac{1}{64}K_4h^4 - \frac{5}{2^{10}}K_6h^6 - \cdots . \tag{4.23}$$

   Subtract (4.21) from 16 times (4.23) to produce a new $\mathcal{O}(h^6)$ formula:

$$M = \frac{1}{15}[16N_2(h/2) - N_2(h)] + \frac{1}{64}K_6h^6 + \cdots . \tag{4.24}$$

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
        $D(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot D(i, j-1) - D(i-1, j-1)\right]$        (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
        $D(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot D(i, j-1) - D(i-1, j-1)\right]$        (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 $(x - x_0)(x - x_1) \le 0$ does not change the sign over $[x_0, x_1]$.)

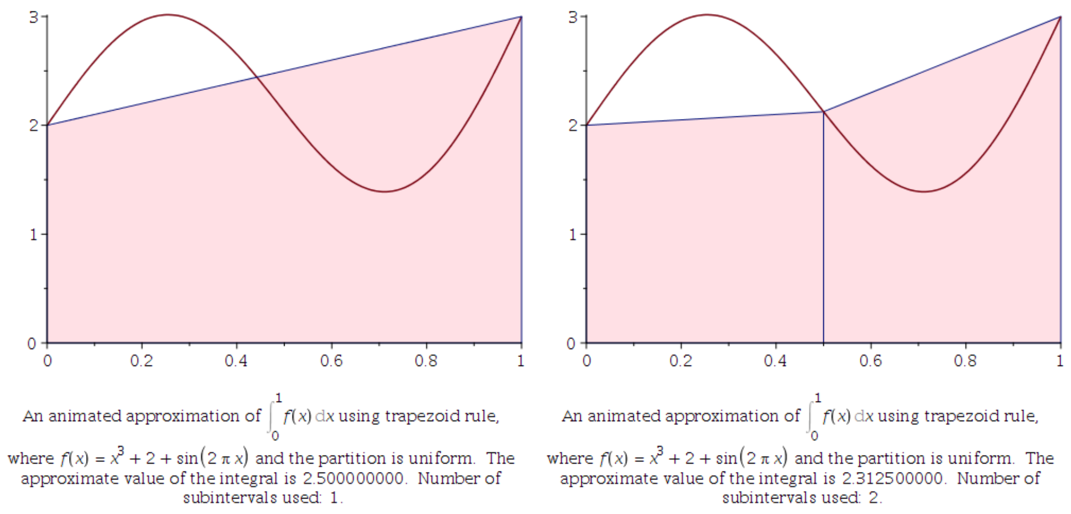
Definition 4.21. The corresponding quadrature formula is

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{h}{2}[f(x_0) + f(x_1)] - \frac{h^3}{12}f''(\xi), \quad \textbf{(Trapezoid)} \tag{4.41}$$

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]]);
```



An animated approximation of $\int_0^1 f(x)\,dx$ using trapezoid rule, where $f(x) = x^3 + 2 + \sin(2\,\pi\,x)$ and the partition is uniform. The approximate value of the integral is 2.500000000. Number of subintervals used: 1.

An animated approximation of $\int_0^1 f(x)\,dx$ using trapezoid rule, where $f(x) = x^3 + 2 + \sin(2\,\pi\,x)$ and the partition is uniform. The approximate value of the integral is 2.312500000. Number of subintervals used: 2.

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]]);
```

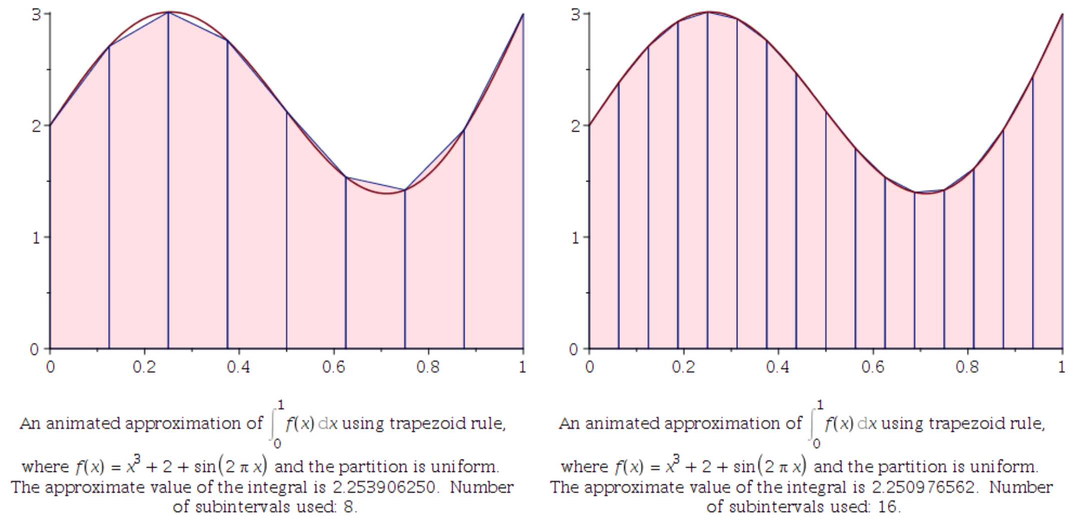


Figure 4.2: Composite trapazoid rule.

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

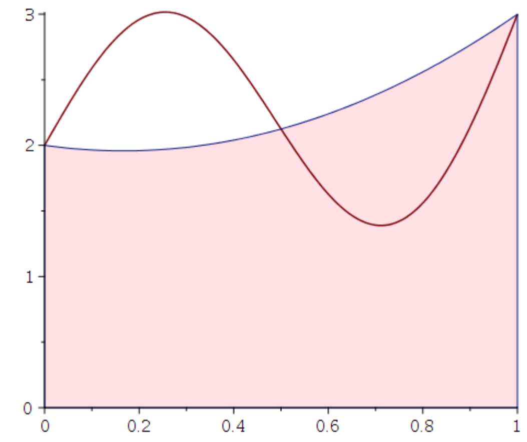
$$\int_{x_0}^{x_2} f(x)\,dx \approx \int_{x_0}^{x_2} \sum_{i=0}^{2} f(x_i) L_{2,i}(x) = \sum_{i=0}^{2} f(x_i) \int_{x_0}^{x_2} L_{2,i}(x), \tag{4.45}$$

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 Simpson's 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.141, 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 $\dfrac{\ln 5}{2} \approx 0.80471895621705018730$. 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-eighths 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

$$\begin{aligned} T(2) &= \frac{1}{2}T(1) + \frac{1}{2}\cdot\Big[f\Big(\frac{1}{2}\Big)\Big] \\ T(4) &= \frac{1}{2}T(2) + \frac{1}{4}\cdot\Big[f\Big(\frac{1}{4}\Big) + f\Big(\frac{3}{4}\Big)\Big] \\ T(8) &= \frac{1}{2}T(4) + \frac{1}{8}\cdot\Big[f\Big(\frac{1}{8}\Big) + f\Big(\frac{3}{8}\Big) + f\Big(\frac{5}{8}\Big) + f\Big(\frac{7}{8}\Big)\Big] \end{aligned} \tag{4.60}$$

With $h = (b-a)/(2n)$, the general formula pertaining to any interval $[a, b]$ is as follows:

$$T(2n) = \frac{1}{2}T(n) + h\,[f(a+h) + f(a+3h) + \cdots + f(a+(2n-1)h)], \tag{4.61}$$

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

$$h_0 = b-a, \quad h_i = \frac{1}{2}h_{i-1}, \quad i \ge 1.$$

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

$$\int_a^b f(x)\,dx = \frac{1}{3}\big[4T(2n) - T(n)\big] - \frac{3}{4}K_4h^4 - \frac{15}{16}K_6h^6 - \cdots. \tag{4.66}$$

$\boxed{\textbf{Algorithm}}$ **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
        $R(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot R(i,j-1) - R(i-1,j-1)\right]$        (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: $\displaystyle\int_0^\pi \sin x\, dx = 2$. 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 $\le n$**,** 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

$$(w_0, w_1, w_2) = \Big(\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\Big). \tag{4.74}$$

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

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= \frac{3}{2}(x^2 - \frac{1}{3}) \\ P_3(x) &= \frac{5}{2}(x^3 - \frac{3}{5}x) \\ P_4(x) &= \frac{35}{8}(x^4 - \frac{6}{7}x^2 + \frac{3}{35}) \\ P_5(x) &= \frac{63}{8}(x^5 - \frac{10}{9}x^3 + \frac{5}{21}x) \end{aligned} \tag{4.80}$$

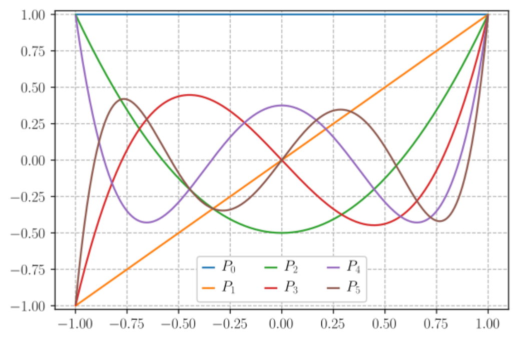


Figure 4.4: Legendre polynomials - Wikipedia

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*

$$w_i = \int_{-1}^{1} \prod_{j=1,\, j\ne i}^{n} \frac{x - x_j}{x_i - x_j}\,dx. \quad \Big(= \int_{-1}^{1} L_{n-1,i}(x)\,dx \Big) \tag{4.82}$$

Then,

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=1}^{n} w_i\, f(x_i) \ \text{ is exact}, \ \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.83}$$

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;
                        Legendre[1] := x
                                    3  2   1
                    Legendre[2] := - x  - -
                                    2      2
                                   5  3   3
                   Legendre[3] := - x  - - x
                                   2      2
                               35  4   15  2   3
                Legendre[4] := -- x  - -- x  + -
                               8       4       8
                             63  5   35  3   15
              Legendre[5] := -- x  - -- x  + -- x
                             8       4       8

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:
```

```

for k to nmax do
    printf("         k=%d\n", k);
    print(Nodek = evalf(Node[k]));
    print(Weightk = evalf(Weight[k]));
end do;
         k=1
                         Nodek = [0.]
                        Weightk = [2.]
         k=2
                        [   -0.5773502693     ]
                Nodek = [                     ]
                        [    0.5773502693     ]

                                  [1.]
                        Weightk = [  ]
                                  [1.]
         k=3
                        [   -0.7745966692     ]
                        [                     ]
                Nodek = [         0.          ]
                        [                     ]
                        [    0.7745966692     ]

                             [0.5555555556]
                             [            ]
                   Weightk = [0.8888888889]
                             [            ]
                             [0.5555555556]
         k=4
                        [   -0.8611363114     ]
                        [                     ]
                        [   -0.3399810437     ]
                Nodek = [                     ]
                        [    0.3399810437     ]
                        [                     ]
                        [    0.8611363114     ]

                             [0.3478548456]
                             [            ]
                             [0.6521451560]
                   Weightk = [            ]
                             [0.6521451563]
                             [            ]
                             [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*

$$w_i = \int_{-1}^{1} \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \Big(= \int_{-1}^{1} L_{n,i}(x)\, dx \Big) \tag{4.89}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=0}^{n} w_i\, f(x_i) \ \text{ is exact}, \quad \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.90}$$

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

$$w_i = \int_{-1}^{1} \prod_{j=1,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \Big(= \int_{-1}^{1} L_{n-1,i}(x)\, dx \Big) \tag{4.91}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i) \ \text{ is exact}, \quad \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.92}$$

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 $\displaystyle\int_0^{\pi} \sin(x)\, dx$, 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: $f''(1.6) \approx 24.8$

4.3. Verify Claim 4.18, p. 146: 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. (Try to introduce a formula described as in (4.26), page 143.)

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. $\displaystyle\int_1^3 \frac{1}{x}\,dx$ II. $\displaystyle\int_0^\pi \cos^2 x\,dx$

(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 $\displaystyle\int_0^{\pi/2} \cos^2 x\,dx$, 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

Definition 5.1. The first-order **initial value problem** (**IVP**) is formulated as follows: find $\{y_i(x) : i = 1, 2, \cdots, M\}$ satisfying

$$\begin{aligned} \frac{dy_i}{dx} &= f_i(x, y_1, y_2, \cdots, y_M), \qquad i = 1, 2, \cdots, M, \\ y_i(x_0) &= y_{i0}, \end{aligned} \tag{5.1}$$

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) $y' = e^{x-y}, \quad 0 \le x \le 1, \quad y(0) = 1$

(b) $y' = (1 + x^2)y, \quad 3 \le x \le 5, \quad y(3) = 1$

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

$$y_{n+1} = y_n + h\, f(x_n, y_n), \quad n \geq 0. \tag{5.9}$$

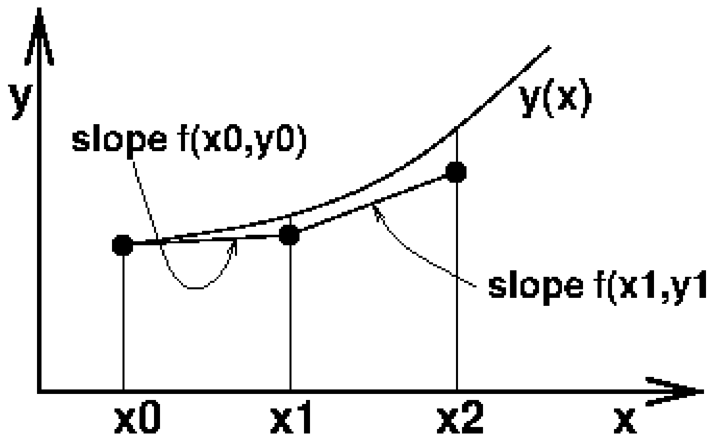


Figure 5.1: The Euler method.

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 5.6.

Proof. The true solution y satisfies

$$y(x_{n+1}) = y(x_n) + hf(x_n, y(x_n)) + \mathcal{O}(h^2). \tag{5.17}$$

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:

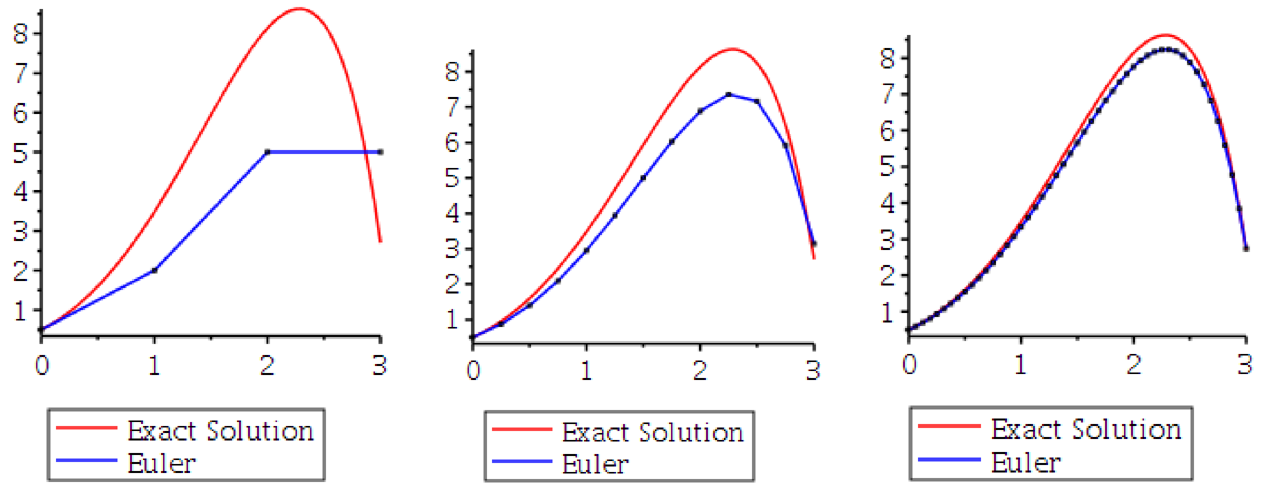


Figure 5.2: The Euler method, with $h = 1 \to \frac{1}{4} \to \frac{1}{16}$.

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.
- $m = 2 \Rightarrow y_{n+1} = y_n + h\Big[f(x_n, y_n) + \dfrac{h}{2} f'(x_n, y_n)\Big]$
- 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):

$$\begin{cases} y' &= f(x,y), \\ y(x_0) &= y_0. \end{cases} \qquad \text{(IVP)} \tag{5.27}$$

- 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) $w_j \ge 0$ 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

$$\begin{aligned} &y + h(w_1 K_1 + w_2 K_2) \\ &\quad = y + w_1 h\,f(x, y) + w_2 h\,f(x + \alpha h, y + \beta h\,K_1) \\ &\quad = y + w_1 h\,f + w_2 h\,(f + \alpha h\,f_x + \beta h\,f_y f) + \mathcal{O}(h^3), \end{aligned}$$

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:

$\boxed{\textbf{Algorithm}}$ **5.18.**

I. $w_1 = w_2 = \frac{1}{2}, \;\; \alpha = \beta = 1$:

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. $w_1 = 0, \;\; w_2 = 1, \;\; \alpha = \beta = \frac{1}{2}$:

For the choices, the algorithm (5.30) reads

$$y_{n+1} = y_n + h\,f\Big(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\Big) \tag{5.36}$$

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:

$$y_{n+1} = y_n + h\,(w_1 K_1 + w_2 K_2 + w_3 K_3 + w_4 K_4) \tag{5.38}$$

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:

$\boxed{\textbf{Algorithm}}$ **5.19. Fourth-order Runge-Kutta method (RK4)**: The most commonly used set of parameter values yields

$$y_{n+1} = y_n + \frac{h}{6}\,(K_1 + 2K_2 + 2K_3 + K_4) \tag{5.39}$$

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
$1/4$	$4.61 \cdot 10^{-4}$	
$1/8$	$2.93 \cdot 10^{-5}$	$\dfrac{4.61 \cdot 10^{-4}}{2.93 \cdot 10^{-5}} = 15.73378840$
$1/16$	$1.85 \cdot 10^{-6}$	$\dfrac{2.93 \cdot 10^{-5}}{1.85 \cdot 10^{-6}} = 15.83783784$
$1/32$	$1.01 \cdot 10^{-7}$	$\dfrac{1.85 \cdot 10^{-6}}{1.01 \cdot 10^{-7}} = 18.31683168$

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 201.

- 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).

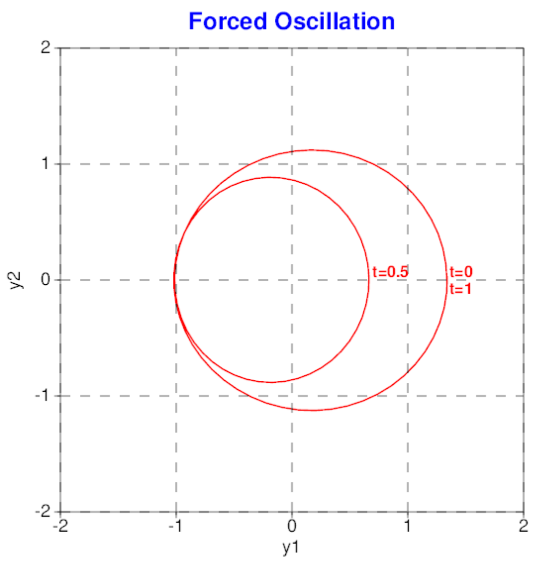


Figure 5.3: The trajectory of the mass satisfying (5.45)-(5.46).

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

$$|\mathbf{y}^h_{N_t} - \mathbf{y}(1)| = \Big(\big[y^h_{1,N_t} - y_1(1)\big]^2 + \big[y^h_{2,N_t} - y_2(1)\big]^2\Big)^{1/2}, \tag{5.47}$$

  where $\mathbf{y}^h_{N_t}$ 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 [4].

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:
```

```

m := 2:
x0 := Vector(m):

t0 := 0: tt := 1.: nt := 40:
x0(1) := -1:
x0(2) := 6:

xRK4 := Array(0..nt, 1..m):
RK4SYS(t0,tt,nt,m,x0,xRK4):

# Compute the analytic solution
#--------------------------------
ODE := diff(x(t),t)=2*x(t)+4*y(t), diff(y(t),t)=-x(t)+6*y(t):
ivs := x(0) = -1, y(0) = 6;
dsolve([ODE, ivs]);
  /                                    1                     \
 { x(t) = exp(4 t) (-1 + 26 t), y(t) = - exp(4 t) (24 + 52 t) }
  \                                    4                     /
ex := t -> exp(4*t)*(-1 + 26*t):
ey := t -> 1/4*exp(4*t)*(24 + 52*t):

# Check error
#--------------------------------
h := (tt - t0)/nt:
printf("     n     x(n)      y(n)     error(x)  error(y)\n");
printf(" -----------------------------------------------\n");
for n from 0 by 2 to nt do
    printf(" \t %5d %10.3f  %10.3f     %-10.3g %-10.3g\n",
          n, xRK4[n,1], xRK4[n,2], abs(xRK4[n,1]-ex(n*h)),
           abs(xRK4[n,2]-ey(n*h)) );
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):
```

```

# Compute the analytic solution
#--------------------------------
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) = - exp(2 x) (sin(x) - 2 cos(x))
                          5
ey := x -> 1/5*exp(2*x)*(sin(x) - 2*cos(x))
diff(ey(x), x)
     2                                1
     - exp(2 x) (sin(x) - 2 cos(x)) + - exp(2 x) (2 sin(x) + cos(x))
     5                                5
eyp:=x->2/5*exp(2*x)*(sin(x)-2*cos(x))+1/5*exp(2*x)*(2*sin(x) + cos(x)):

# Check error
#--------------------------------
printf(" n   y_n    y(x_n)    y'_n    y'(x_n)   err(y)   err(y')\n");
printf("--------------------------------------------------------\n");
for n from 0 to nt do
    xp := h*n + a;
    printf("  %2d   %12.8f   %12.8f   %12.8f   %12.8f    %.3g    %.3g\n",
        n, Xn[n, 1], ey(xp), Xn[n, 2], eyp(xp),
        abs(Xn[n, 1] - ey(xp)), abs(Xn[n, 2] - eyp(xp)));
end do:
```

Result

```
  n       y_n          y(x_n)        y'_n          y'(x_n)      err(y)     err(y')
  -----------------------------------------------------------------------------------
  0    -0.40000000   -0.40000000   -0.60000000   -0.60000000    0          0
  1    -0.46173334   -0.46173297   -0.63163124   -0.63163105    3.72e-07   1.91e-07
  2    -0.52555988   -0.52555905   -0.64014895   -0.64014866    8.36e-07   2.84e-07
  3    -0.58860144   -0.58860005   -0.61366381   -0.61366361    1.39e-06   1.99e-07
  4    -0.64661231   -0.64661028   -0.53658203   -0.53658220    2.02e-06   1.68e-07
  5    -0.69356666   -0.69356395   -0.38873810   -0.38873905    2.71e-06   9.58e-07
  6    -0.72115190   -0.72114849   -0.14438087   -0.14438322    3.41e-06   2.35e-06
  7    -0.71815295   -0.71814890    0.22899702    0.22899243    4.06e-06   4.59e-06
  8    -0.66971133   -0.66970677    0.77199180    0.77198383    4.55e-06   7.97e-06
  9    -0.55644290   -0.55643814    1.53478148    1.53476862    4.77e-06   1.29e-05
 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 $\boldsymbol{y_{n+1} = y_n + hf(x_n, y_n)}$ 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

```python
#!/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:

  $$f(x_{n+1},y_{n+1}) \to f(x_{n+1},y^*_{n+1}), \quad y^*_{n+1} = y_n + hf(x_n,y_n),$$

  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 $\boldsymbol{d} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{d})$.
  - 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:

$$\texttt{tol} = \mathcal{O}(h^{\alpha}), \quad \alpha = 3. \tag{5.71}$$

Self-study 5.36. Consider the problem

$$\begin{aligned} y' &= \frac{1}{10}\big(\cos(x^2y') - \cos(e^y)\big) + \frac{1}{x}, \quad 1 \le x \le 3, \\ & y(1) = 0. \end{aligned} \tag{5.72}$$

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. 209. 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. C Consider the following system of first-order differential equations:

$$\begin{cases} u_1' &= u_2 - u_3 + t, & u_1(0) = 1, \\ u_2' &= 3t^2, & u_2(0) = 1, & (0 \le t \le 1) \\ u_3' &= u_2 + e^{-t}, & u_3(0) = -1, \end{cases} \tag{5.74}$$

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

5.6. 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 $x \searrow 0$.

5.7. Prove Claim 5.30, page 208.

CHAPTER **6**

Gauss Elimination and Its Variants

One of the most frequently occurring problems in all areas of scientific endeavor is that of solving a 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. We will see that there are many ways to organize this fundamental algorithm.

Contents of Chapter 6

6.1. Systems of Linear Equations

Note: Consider a system of n linear equations in n unknowns

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots & \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases} \tag{6.1}$$

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 (6.1) again and again, we generally prefer to write it as a single matrix equation

$$A\mathbf{x} = \mathbf{b}, \tag{6.2}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \text{and} \;\; \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

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
  - Stationary and nonstationary methods
    (Jacobi, Gauss-Seidel, SOR, SSOR; CG, MINRES, GMRES, BiCG, QMR, $\cdots$)
  - Easier to optimize and parallelize
  - Low algorithmic complexity, but may not converge

6.1.1. Nonsingular matrices

Definition 6.1. **(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 $AB = I_n = BA$, 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 6.2. **(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 $A\mathrm{x} = \mathbf{0}$ 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 6.3. Let $A \in \mathbb{R}^{n\times n}$ and eigenvalues of A be λ_i, $i = 1, 2, \cdots, n$. Show that

$$\det(A) = \prod_{i=1}^{n} \lambda_i. \tag{6.3}$$

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.

6.1.2. Numerical Solutions of Differential Equations

Consider the following differential equation:

$$\begin{array}{lrcll} \text{(a)} & -u_{xx}+cu & = & f, & x\in(a_x,b_x), \\ \text{(b)} & -u_x+\beta u & = & g, & x=a_x, \\ \text{(c)} & u_x+\beta u & = & g, & x=b_x. \end{array} \tag{6.4}$$

where

$$c\ge 0 \text{ and } \beta\ge 0 \quad (c+\beta>0).$$

Numerical Discretization:

- 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 $i=0,1,\cdots,n_x$.
- It follows from the **Taylor series** that

$$-u_{xx}(x_i) = \frac{-u_{i-1}+2u_i-u_{i+1}}{h_x^2} + \frac{u_{xxxx}(x_i)}{12}h_x^2 + \cdots .$$

  Thus the central second-order **finite difference** (FD) scheme for u_{xx} at x_i reads

$$-u_{xx}(x_i) \approx \frac{-u_{i-1}+2u_i-u_{i+1}}{h_x^2}. \tag{6.5}$$

  See also (4.14).

- Apply the FD scheme for (6.4.a) to have

$$-u_{i-1} + (2+h_x^2 c)u_i - u_{i+1} = h_x^2 f_i, \ \ i=0,1,\cdots,n_x. \tag{6.6}$$

- However, we will meet ghost grid values at the end points. For example, at the point $a_x = x_0$, the formula becomes

$$-u_{-1} + (2+h_x^2 c)u_0 - u_1 = h_x^2 f_0. \tag{6.7}$$

  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 (6.4.b). The central FD scheme for u_x at x_0 can be formulated as

$$u_x(x_0) \approx \frac{u_1 - u_{-1}}{2h_x}, \;\; \textbf{Trunc.Err} = -\frac{u_{xxx}(x_0)}{6}h_x^2 + \cdots. \tag{6.8}$$

Thus the equation (6.4.b), $-u_x + \beta u = g$, can be approximated (at x_0)

$$u_{-1} + 2h_x\beta u_0 - u_1 = 2h_x g_0. \tag{6.9}$$

- Hence it follows from (6.7) and (6.9) that

$$(2 + h_x^2 c + 2h_x\beta)u_0 - 2u_1 = h_x^2 f_0 + 2h_x g_0. \tag{6.10}$$

The same can be considered for the algebraic equation at the point x_n.

$\boxed{\textbf{Scheme}}$ **6.4.** The problem (6.4) is reduced to finding the solution $\mathbf{u}$ satisfying

$$A\mathbf{u} = \mathbf{b}, \tag{6.11}$$

where $A \in \mathbb{R}^{(n_x+1)\times(n_x+1)}$,

$$A = \begin{bmatrix} 2+h_x^2c+2h_x\beta & -2 & & & \\ -1 & 2+h_x^2c & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2+h_x^2c & -1 \\ & & & -2 & 2+h_x^2c+2h_x\beta \end{bmatrix},$$

and

$$\mathbf{b} = \begin{bmatrix} h_x^2 f_0 \\ h_x^2 f_1 \\ \vdots \\ h_x^2 f_{n_x-1} \\ h_x^2 f_{n_x} \end{bmatrix} + \begin{bmatrix} 2h_x g_0 \\ 0 \\ \vdots \\ 0 \\ 2h_x g_{n_x} \end{bmatrix}.$$

Definition **6.5.** Such a technique of removing ghost grid values is called **outer bordering**.

Dirichlet Boundary Condition:

$\boxed{\textbf{Scheme}}$ **6.6.** 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

$$\begin{array}{lrcll} \textbf{(a)} & -u_{xx}+cu & = & f, & x\in(a_x,b_x),\\ \textbf{(b)} & -u_x+\beta u & = & g, & x=a_x,\\ \textbf{(c)} & u & = & u_d, & x=b_x. \end{array} \tag{6.12}$$

- Then, the corresponding algebraic system can be formulated as

$$A'\,\mathbf{u}=\mathbf{b}', \tag{6.13}$$

where $A'\in\mathbb{R}^{(n_x+1)\times(n_x+1)}$,

$$A'=\begin{bmatrix} 2+h_x^2c+2h_x\beta & -2 & & & \\ -1 & 2+h_x^2c & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2+h_x^2c & -1\\ & & & 0 & 1 \end{bmatrix},$$

and

$$\mathbf{b}'=\begin{bmatrix} h_x^2f_0\\ h_x^2f_1\\ \vdots\\ h_x^2f_{n_x-1}\\ u_d \end{bmatrix}+\begin{bmatrix} 2h_xg_0\\ 0\\ \vdots\\ 0\\ 0 \end{bmatrix}.$$

6.2. Triangular Systems

Definition 6.7.

(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 6.8. *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

$$L\,\mathbf{y} = \mathbf{b}, \tag{6.14}$$

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:

$$\begin{array}{lcl} \ell_{11}\,y_1 & = & b_1 \\ \ell_{21}\,y_1 + \ell_{22}\,y_2 & = & b_2 \\ \ell_{31}\,y_1 + \ell_{32}\,y_2 + \ell_{33}\,y_3 & = & b_3 \\ \quad\vdots & \vdots & \\ \ell_{n1}\,y_1 + \ell_{n2}\,y_2 + \ell_{n3}\,y_3 + \cdots + \ell_{nn}\,y_n & = & b_n \end{array} \tag{6.15}$$

The first equation involves only the unknown y_1, which can be found as

$$y_1 = b_1/\ell_{11}. \tag{6.16}$$

With y_1 just obtained, we can determine y_2 from the second equation:

$$y_2 = (b_2 - \ell_{21}\,y_1)/\ell_{22}. \tag{6.17}$$

Now with y_2 known, we can solve the third equation for y_3, and so on.

Algorithm 6.9. In general, once we have $y_1, y_2, \cdots, y_{i-1}$, we can solve for y_i using the ith equation:

$$\begin{aligned} y_i &= (b_i - \ell_{i1}\, y_1 - \ell_{i2}\, y_2 - \cdots - \ell_{i,i-1}\, y_{i-1})/\ell_{ii} \\ &= \frac{1}{\ell_{ii}}\Big(b_i - \sum_{j=1}^{i-1} \ell_{ij}\, y_j\Big) \end{aligned} \tag{6.18}$$

Matlab-code 6.10. (Forward Substitution/Elimination):

```
for i=1:n
   for j=1:i-1
      b(i) = b(i)-L(i,j)*b(j)
   end                                                    (6.19)
   if L(i,i)==0, error('L: singular!'); end
   b(i) = b(i)/L(i,i)
end
```

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

$$\sum_{i=1}^{n}\{2(i-1)+1\} = \sum_{i=1}^{n}\{2i-1\} = n(n+1) - n = n^2. \tag{6.20}$$

Upper-triangular systems

Consider the system

$$U\,\mathbf{x} = \mathbf{y}, \tag{6.21}$$

where $U = (u_{ij}) \in \mathbb{R}^{n\times n}$ is nonsingular, upper-triangular. Writing it out in detail, we get

$$\begin{array}{rcl} u_{11}\,x_1 + u_{12}\,x_2 + \cdots + u_{1,n-1}\,x_{n-1} + u_{1,n}\,x_n &=& y_1 \\ u_{22}\,x_2 + \cdots + u_{2,n-1}\,x_{n-1} + u_{2,n}\,x_n &=& y_2 \\ \vdots &=& \vdots \\ u_{n-1,n-1}\,x_{n-1} + u_{n-1,n}\,x_n &=& y_{n-1} \\ u_{n,n}\,x_n &=& y_n \end{array} \tag{6.22}$$

It is clear that we should solve the system from bottom to top.

Matlab-code 6.11. (Back Substitution):

```
for i=n:-1:1
   if(U(i,i)==0), error('U: singular!'); end
   x(i)=b(i)/U(i,i);                                    (6.23)
   b(1:i-1)=b(1:i-1)-U(1:i-1,i)*x(i);
end
```

Computational complexity: $n^2 + \mathcal{O}(n)$ flops.

6.3. 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 6.12. (**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;
- then Further transform the system $U\mathrm{x} = \mathrm{y}$ to get x.
  - It is convenient to represent $A\mathrm{x} = \mathrm{b}$ by an augmented matrix $[A|\mathrm{b}]$; each equation in $A\mathrm{x} = \mathrm{b}$ corresponds to a row of the augmented matrix.
  - **Transformation of the system**: By means of three **elementary row operations**, applied on the augmented matrix.

Definition 6.13. Elementary row operations (EROs).

$$\begin{array}{ll} \text{Replacement:} & R_i \leftarrow R_i + \alpha R_j \;\; (i \neq j) \\ \text{Interchange:} & R_i \leftrightarrow R_j \\ \text{Scaling:} & R_i \leftarrow \beta R_i \;\; (\beta \neq 0) \end{array} \tag{6.24}$$

Proposition 6.14.

(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.

6.3.1. 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:

$$A\mathbf{x} = \mathbf{b}_1,\ A\mathbf{x} = \mathbf{b}_2,\ \cdots,\ A\mathbf{x} = \mathbf{b}_p. \tag{6.25}$$

Definition 6.15. 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 6.16. 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

$$\begin{cases} L\mathbf{y} = \mathbf{b}, \\ U\mathbf{x} = \mathbf{y}, \end{cases} \tag{6.26}$$

each algebraic equation can be solved effectively, via substitutions.

Algorithm 6.17. (An LU Factorization Algorithm) *The derivation introduces an LU factorization: Let $A \in \mathbb{R}^{m\times n}$. Then*

$$\begin{aligned} A &= I_m\, A \\ &= I_m E_1^{-1}\, E_1 A \\ &= I_m E_1^{-1} E_2^{-1}\, E_2 E_1 A = (E_2 E_1)^{-1}\, E_2 E_1 A \\ &= \quad \vdots \\ &= I_m E_1^{-1} E_2^{-1} \cdots E_p^{-1}\, \underbrace{E_p \cdots E_2 E_1 A}_{\text{an echelon form}} = \underbrace{(E_p \cdots E_1)^{-1}}_{L}\, \underbrace{E_p \cdots E_1 A}_{U}, \end{aligned} \tag{6.27}$$

where E_i are elementary matrics for "replacement".

Example 6.18. 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 6.19. **(LU Decomposition Theorem)** *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

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & \mathbf{0} \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ \mathbf{0} & U_{22} \end{bmatrix} = \begin{bmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{bmatrix}, \tag{6.28}$$

where A_{11} is a $j \times j$ leading principal submatrix. Thus

$$\boldsymbol{det}\,(A_{11}) = \boldsymbol{det}\,(L_{11}U_{11}) = 1 \cdot \boldsymbol{det}\,(U_{11}) = \prod_{k=1}^{j} (U_{11})_{kk} \neq 0.$$

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 6.20. Find the LU factorization of $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$.

Solution. (Practical Implementation):

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + 2R_1]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{3} & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \frac{3}{4} R_2} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{\frac{3}{4}} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ -2 & \mathbf{\frac{3}{4}} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

* It is easy to verify that $A = LU$.

Example 6.21. *Find the LU factorization of* $A = \begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix}$.

Solution.

$$\begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix} \xrightarrow[\substack{R_3 \leftarrow R_3 + 5R_1 \\ R_4 \leftarrow R_4 - 6R_1}]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} \mathbf{2} & -1 \\ \mathbf{3} & 8 \\ -5 & -2 \\ \mathbf{6} & 4 \end{bmatrix} \xrightarrow[R_4 \leftarrow R_4 - \frac{1}{2} R_2]{R_3 \leftarrow R_3 + \frac{1}{4} R_2} \begin{bmatrix} \mathbf{2} & -1 \\ \mathbf{3} & \mathbf{8} \\ -5 & -\mathbf{\frac{1}{4}} \\ \mathbf{6} & \mathbf{\frac{1}{2}} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \mathbf{3} & 1 & 0 & 0 \\ \mathbf{-5} & -\mathbf{\frac{1}{4}} & 1 & 0 \\ \mathbf{6} & \mathbf{\frac{1}{2}} & 0 & 1 \end{bmatrix}_{4\times4}, \quad U = \begin{bmatrix} 2 & -1 \\ 0 & 8 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}_{4\times2}.$$

Note: U has the same size as A, i.e., $A \in \mathbb{R}^{4\times2}$, while L is a square matrix and is a unit lower triangular matrix.

Remark 6.22. **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 $[A\ \mathrm{b}]$), while finding A^{-1} requires about $2n^3$ flops.
- Solving $L\mathrm{y} = \mathrm{b}$ and $U\mathrm{x} = \mathrm{y}$ requires about $2n^2$ flops, because any $n \times n$ triangular system can be solved in about n^2 flops.
- Multiplying 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} & $A^{-1}\mathrm{b}$).

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 $A\mathrm{x} = \mathrm{b}$ with LU factorization is much faster than using A^{-1}.

[†] A **flop** is a **floating point operation** by $+$, $-$, $\times$ or $\div$.

Solving Linear Systems by *LU* Factorization

- The LU factorization can be applied for general $m \times n$ matrices:

$$A = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ * & 1 & 0 & 0 \\ * & * & 1 & 0 \\ * & * & * & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \blacksquare & * & * & * & * \\ 0 & \blacksquare & * & * & * \\ 0 & 0 & 0 & \blacksquare & * \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{U} \tag{6.29}$$

- 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

$$A\mathrm{x} = \mathrm{b} \iff (LU)\mathrm{x} = L(U\mathrm{x}) = \mathrm{b} \iff \begin{cases} L\mathrm{y} = \mathrm{b} \\ U\mathrm{x} = \mathrm{y} \end{cases} \tag{6.30}$$

In the following couple of examples, LU is given.

Example 6.23. 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 (6.30), we know there are two steps to perform:

$$\begin{array}{ll} (1) & \text{Solve } L\mathbf{y} = \mathbf{b} \text{ for } \mathbf{y}; \\ (2) & \text{Solve } U\mathbf{x} = \mathbf{y} \text{ for } \mathbf{x}. \end{array}$$

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$ by row reduction

$$[L \,\vdots\, \mathbf{b}] = \left[\begin{array}{ccc:c} 1 & 0 & 0 & -12 \\ 2 & 1 & 0 & -14 \\ -3 & 2 & 1 & 64 \end{array}\right] \to \cdots \to \left[\begin{array}{ccc:c} 1 & 0 & 0 & -12 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 8 \end{array}\right] = [I \,\vdots\, \mathbf{y}]. \quad \textbf{(6.31)}$$

(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$ by row reduction

$$[U \,\vdots\, \mathbf{y}] = \left[\begin{array}{ccc:c} 1 & 4 & -2 & -12 \\ 0 & -3 & 1 & 10 \\ 0 & 0 & 8 & 8 \end{array}\right] \to \cdots \to \left[\begin{array}{ccc:c} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -3 \\ 0 & 0 & 1 & 1 \end{array}\right] = [I \,\vdots\, \mathbf{x}]. \quad \textbf{(6.32)}$$

Thus, $\mathbf{x} = [2, -3, 1]^T$. □

Example 6.24. 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, $\mathbf{x} = [3, -2, 1, 5]^T$. □

6.3.2. Gauss Elimination with Pivoting

Definition 6.25. A **permutation matrix** is a matrix that has exactly one 1 in each row and in each column, all other entries being zero.

Self-study 6.26. Show that if P is permutation matrix, then $P^TP = PP^T = I$. Thus P is nonsingular and

$$P^{-1} = P^T.$$

Solution.

Lemma 6.27. 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 6.28. 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)

$\boxed{\textbf{Theorem}}$ **6.29.** ***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*

$$\widehat{A} = PA = LU \tag{6.33}$$

or, equivalently,

$$A = P^T LU \tag{6.34}$$

Note: If A is singular, then so is U.

$\boxed{\textbf{Algorithm}}$ **6.30.** Solving $A\mathrm{x} = \mathrm{b}$ using *Gauss elimination with partial pivoting*:

1. Factorize A into $A = P^T LU$, where

$$\begin{array}{rcl} P &=& \text{permutation matrix,} \\ L &=& \text{unit lower triangular matrix} \\ && \text{(i.e., with ones on the diagonal),} \\ U &=& \text{nonsingular upper-triangular matrix.} \end{array}$$

2. Solve $P^T LU\mathrm{x} = \mathrm{b}$
   (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)

In practice:

$$\left.\begin{array}{rl} A\mathbf{x} = \mathbf{b} \iff & P^T(LU)\mathbf{x} = \mathbf{b} \\ \iff & L(U\mathbf{x}) = P\mathbf{b} \end{array}\right\} \iff \left\{\begin{array}{l} L\mathbf{y} = P\mathbf{b} \\ U\mathbf{x} = \mathbf{y} \end{array}\right.$$

Theorem 6.31. *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*

$$P_1 A P_2 = LU.$$

Only one of P_1 and P_2 is necessary.

Remark 6.32. $P_1 A$ reorders the rows of A, AP_2 reorders the columns, and P_1AP_2 reorders both. Consider

$$\begin{aligned} P_1' A P_2' &= \begin{bmatrix} a_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{0} \\ L_{21} & I \end{bmatrix} \begin{bmatrix} u_{11} & U_{12} \\ \mathbf{0} & \tilde{A}_{22} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & U_{12} \\ L_{21}u_{11} & L_{21}U_{12} + \tilde{A}_{22} \end{bmatrix} \end{aligned} \qquad (6.35)$$

- 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 6.33. We can choose P_1 and P_2 so that a_{11} in (6.35) 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 6.34. Find the LU factorization of $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$, which is considered in Example 6.20.

Solution. (Without pivoting)

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow[R_3 \leftarrow R_3 + 2R_1]{R_2 \leftarrow R_2 - 3R_1} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \mathbf{3} & -3 \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 - \frac{3}{4} R_2} \begin{bmatrix} \mathbf{3} & -1 & \mathbf{1} \\ \mathbf{3} & \mathbf{4} & -1 \\ \mathbf{-2} & \frac{\mathbf{3}}{\mathbf{4}} & -\frac{9}{4} \end{bmatrix}$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ \mathbf{3} & 1 & 0 \\ -2 & \frac{\mathbf{3}}{\mathbf{4}} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} 3 & -1 & 1 \\ 0 & 4 & -1 \\ 0 & 0 & -\frac{9}{4} \end{bmatrix}.$$

(With partial pivoting)

$$A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix} \xrightarrow{\mathbf{R_1 \leftrightarrow R_2}} \begin{bmatrix} \boxed{9} & 1 & 2 \\ 3 & -1 & 1 \\ -6 & 5 & -5 \end{bmatrix}$$

$$\xrightarrow[R_3 \leftarrow R_3 + \frac{2}{3} R_1]{R_2 \leftarrow R_2 - \frac{1}{3} R_1} \begin{bmatrix} \boxed{9} & 1 & 2 \\ \frac{\mathbf{1}}{\mathbf{3}} & -\frac{4}{3} & \frac{1}{3} \\ -\frac{\mathbf{2}}{\mathbf{3}} & \frac{17}{3} & -\frac{11}{3} \end{bmatrix} \xrightarrow{\mathbf{R_2 \leftrightarrow R_3}} \begin{bmatrix} \boxed{9} & 1 & 2 \\ -\frac{\mathbf{2}}{\mathbf{3}} & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ \frac{\mathbf{1}}{\mathbf{3}} & -\frac{4}{3} & \frac{1}{3} \end{bmatrix}$$

$$\xrightarrow{R_3 \leftarrow R_3 + \frac{4}{17} R_2} \begin{bmatrix} \boxed{9} & 1 & 2 \\ -\frac{\mathbf{2}}{\mathbf{3}} & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ \frac{\mathbf{1}}{\mathbf{3}} & -\frac{\mathbf{4}}{\mathbf{17}} & -\frac{9}{17} \end{bmatrix}, \quad I \xrightarrow{\mathbf{R_1 \leftrightarrow R_2}} E \xrightarrow{\mathbf{R_2 \leftrightarrow R_3}} P$$

$$PA = LU$$

$$P = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}, \quad L = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{\mathbf{2}}{\mathbf{3}} & 1 & 0 \\ \frac{\mathbf{1}}{\mathbf{3}} & -\frac{\mathbf{4}}{\mathbf{17}} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} \boxed{9} & 1 & 2 \\ 0 & \boxed{\frac{17}{3}} & -\frac{11}{3} \\ 0 & 0 & \boxed{-\frac{9}{17}} \end{bmatrix}$$

6.3.3. The Computation of A^{-1}

Algorithm **6.35. (The 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
$$AX = I. \tag{6.36}$$
- This equation can be written in partitioned form:
$$A[\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n] = [\mathbf{e}_1\ \mathbf{e}_2\ \cdots\ \mathbf{e}_n], \tag{6.37}$$
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
$$A\mathbf{x}_i = \mathbf{e}_i, \quad i = 1, 2, \cdots, n. \tag{6.38}$$
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 (6.38):	$n \cdot 2n^2 = 2n^3$
Total cost:	$\frac{8}{3}n^3 + \mathcal{O}(n^2)$

- **A modification**: The forward-substitution phase requires the solution of
$$L\mathbf{y}_i = \mathbf{e}_i, \quad i = 1, 2, \cdots, n. \tag{6.39}$$
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 6

6.1. Solve the equation $A\mathbf{x} = \mathbf{b}$ by using the LU factorization.

$$A = \begin{bmatrix} 4 & 3 & -5 \\ -4 & -5 & 7 \\ 8 & 6 & -8 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} 2 \\ -4 \\ 6 \end{bmatrix}.$$

(Do not use computer programming for this problem.)

$$\textit{Ans}: \mathbf{x} = \begin{bmatrix} 1/4 \\ 2 \\ 1 \end{bmatrix}.$$

6.2. 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.

6.3. C Consider the system $A\mathbf{x} = \mathbf{b}$, where

$$A = \begin{bmatrix} 1 & -2 & -1 & 3 \\ 1 & -2 & 0 & 1 \\ -3 & -2 & 1 & 7 \\ 0 & -2 & 8 & 5 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} -12 \\ -5 \\ -14 \\ -7 \end{bmatrix}.$$

(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.)

6.4. C Consider the finite difference method on uniform meshes to solve

$$\begin{array}{ll} \text{(a)} & -u_{xx} + u = (\pi^2 + 1)\cos(\pi x), \quad x \in (0,1), \\ \text{(b)} & u(0) = 1 \text{ and } u_x(1) = 0. \end{array} \tag{6.40}$$

(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 (6.40) is $u(x) = \cos(\pi x)$. Measure the maximum errors for the approximate solutions.

(This problem is optional for **undergraduate students**; you will get an extra credit when you solve it.)

Chapter 7
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

$$A\mathbf{x} = \mathbf{b}. \tag{7.1}$$

Iterative methods we will consider can be formulated as

$$\mathbf{x}^k = \mathbf{x}^{k-1} + G(\mathbf{b} - A\mathbf{x}^{k-1}), \tag{7.2}$$

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.

Maple built-in command:

with(NumericalAnalysis);
IterativeApproximate(A, b, opts);

Contents of Chapter 7

7.1. Norms of Vectors and Matrices

7.1.1. Vectors

Definition 7.1. 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 u and v is given by

$$\begin{aligned} \mathbf{u}\bullet\mathbf{v} &= \mathbf{u}^T\mathbf{v} = [u_1\ u_2\ \cdots\ u_n]\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \\ &= u_1 v_1 + u_2 v_2 + \cdots + u_n v_n = \sum_{k=1}^{n} u_k v_k. \end{aligned} \tag{7.3}$$

Definition 7.2. The **length (Euclidean norm)** of v is nonnegative scalar $\|\mathbf{v}\|$ defined by

$$\|\mathbf{v}\| = \sqrt{\mathbf{v}\bullet\mathbf{v}} = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2} \quad \text{and} \quad \|\mathbf{v}\|^2 = \mathbf{v}\bullet\mathbf{v}. \tag{7.4}$$

Definition 7.3. For $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, the **distance** between u and v is

$$\mathit{dist}(\mathbf{u}, \mathbf{v}) = \|\mathbf{u} - \mathbf{v}\|, \tag{7.5}$$

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

Definition 7.4. Two vectors u and v in $\mathbb{R}^n$ are **orthogonal** if $\mathbf{u}\bullet\mathbf{v} = 0$.

Theorem 7.5. ***Pythagorean Theorem***: *Two vectors* u *and* v *are orthogonal if and only if*

$$\|\mathbf{u} + \mathbf{v}\|^2 = \|\mathbf{u}\|^2 + \|\mathbf{v}\|^2. \tag{7.6}$$

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

$$\mathbf{u}\bullet\mathbf{v} = \|\mathbf{u}\|\,\|\mathbf{v}\|\cos\theta, \tag{7.7}$$

where θ is the angle between u and v.

Example 7.6. Let $\mathbf{u} = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ -4 \end{bmatrix}$. Use (7.7) to find the angle between u and v.

Solution.

7.1.2. Vector and matrix norms

Definition 7.7. 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 $\lambda \in \mathbb{R}$,

$$\begin{array}{ll} \|\mathbf{x}\| > 0 \text{ if } \mathbf{x} \neq 0 & \text{(positive definiteness)} \\ \|\lambda\mathbf{x}\| = |\lambda|\,\|\mathbf{x}\| & \text{(homogeneity)} \\ \|\mathbf{x}+\mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\| & \text{(triangle inequality)} \end{array} \tag{7.8}$$

Example 7.8. The most vector common norms are the p-**norms**

$$\|\mathbf{x}\|_p = \Big(\sum_i |x_i|^p\Big)^{1/p}, \quad 1 \le p < \infty, \tag{7.9}$$

and the **infinity-norm** or **maximum-norm**

$$\|\mathbf{x}\|_\infty = \max_i |x_i|. \tag{7.10}$$

Note: Two of frequently used p-norms are

$$\|\mathbf{x}\|_1 = \sum_i |x_i|, \quad \|\mathbf{x}\|_2 = \Big(\sum_i |x_i|^2\Big)^{1/2} \tag{7.11}$$

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

Example 7.9. One may consider the infinity-norm as the limit of p-norms, as $p \to \infty$.

Solution.

Theorem 7.10. ***(Cauchy-Schwarz Inequality)***.

$$|\mathbf{x}\cdot\mathbf{y}| = \left|\sum_{i=1}^n x_i y_1\right| \le \Big(\sum_{i=1}^n x_i^2\Big)^{1/2}\Big(\sum_{i=1}^n y_i^2\Big)^{1/2} = \|\mathbf{x}\|\,\|\mathbf{y}\|. \tag{7.12}$$

Theorem 7.11. *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 7.12. $\mathbf{x}^{(k)} = \Big(1,\ 2+\frac{1}{k},\ \frac{3}{k^2},\ e^{-k}\sin k\Big)^T \to (1,\ 2,\ 0,\ 0)^T$ with respect to the ℓ^∞-norm.

Solution.

Theorem 7.13. **(Equivalence between the ℓ^2-norm and the ℓ^∞-norm)**. *For each $\mathbf{x} \in \mathbb{R}^n$,*

$$\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \sqrt{n}\|\mathbf{x}\|_\infty. \tag{7.13}$$

7.1.3. Matrix norms

Definition 7.14. A **matrix norm** on $m \times n$ matrices is a vector norm on the mn-dimensional space, satisfying

$$\begin{aligned} &\|A\| \ge 0, \text{ and } \|A\| = 0 \;\Leftrightarrow\; A = 0 && \text{(positive definiteness)} \\ &\|\lambda A\| = |\lambda|\,\|A\| && \text{(homogeneity)} \\ &\|A + B\| \le \|A\| + \|B\| && \text{(triangle inequality)} \end{aligned} \tag{7.14}$$

Example 7.15. $\|A\|_F \equiv \Big(\sum_{i,j} |a_{ij}|^2\Big)^{1/2}$ is called the **Frobenius norm**.

Definition 7.16. *Once a vector norm $||\cdot||$ has been specified, the* ***induced matrix norm*** *is defined by*

$$\|A\| = \max_{\mathbf{x}\neq 0} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{7.15}$$

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

Theorem 7.17.

a. *For all operator norms and the Frobenius norm,*

$$\|Ax\| \le \|A\|\,\|x\|, \quad \|A\,B\| \le \|A\|\,\|B\|. \tag{7.16}$$

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 $A^T A = AA^T$ (**normal matrix**).*

Definition 7.18. *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 7.19. Let $A = \begin{bmatrix} 1 & 2 & -1 \\ 0 & 3 & -1 \\ 1 & -2 & 1 \end{bmatrix}$. Then, we have

$$A^{-1} = \frac{1}{2}\begin{bmatrix} 1 & 0 & 1 \\ -1 & 2 & 1 \\ -3 & 4 & 3 \end{bmatrix} \quad \text{and} \quad A^T A = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 17 & -7 \\ 0 & -7 & 3 \end{bmatrix}.$$

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

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

Solution.

Theorem 7.20. (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*

$$(I-A)^{-1} = \sum_{k=0}^{\infty} A^k. \qquad \textbf{(7.17)}$$

7.2. Eigenvectors and Eigenvalues

Definition 7.21. Let A be an $n \times n$ matrix. An **eigenvector** of A is a *nonzero* vector x such that

$$A\mathrm{x} = \lambda\mathrm{x} \tag{7.18}$$

for some scalar λ. In this case, a scalar λ is an **eigenvalue** and x is the *corresponding* **eigenvector**.

Definition 7.22. The scalar equation

$$\mathit{det}\,(A - \lambda I) = 0 \tag{7.19}$$

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

Claim 7.23. λ is an eigenvalue of A if and only if *det* $(A - \lambda I) = 0$.

Example 7.24. Find eigenvalues and eigenvectors of $A = \begin{bmatrix} 1 & 6 \\ 5 & 2 \end{bmatrix}$.

Solution.

Example 7.25. 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);

$$-12 + \lambda^3 - 7\lambda^2 + 16\lambda \qquad (7.20)$$

Now, let us find them by using pencils.

Remark 7.26. Let A be an $n \times n$ matrix. Then the characteristic equation of A is of the form

$$\begin{aligned} p(\lambda) = \textit{\textbf{det}}\,(A - \lambda I) &= (-1)^n\,(\lambda^n + c_{n-1}\lambda^{n-1} + \cdots + c_1\lambda + c_0) \\ &= (-1)^n \prod_{i=1}^{n} (\lambda - \lambda_i), \end{aligned} \tag{7.21}$$

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

$$\textit{\textbf{det}}\,A = p(0) = (-1)^n \prod_{i=1}^{n} (0 - \lambda_i) = \prod_{i=1}^{n} \lambda_i. \tag{7.22}$$

That is, *det* A is the product of all eigenvalues of A.

7.2.1. Spectral radius

Definition 7.27. The **spectral radius** $\rho(A)$ of a matrix A is defined by

$$\rho(A) = \max |\lambda|, \tag{7.23}$$

where λ is an eigenvalue of A.

Theorem 7.28. *If $A \in \mathbb{R}^{n\times n}$, then*

(a) $\|A\|_2 = \sqrt{\rho(A^T A)}$

(b) $\rho(A) \le \|A\|$*, 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 $||\mathbf{x}|| = 1$. Then

$$|\lambda| = |\lambda|\,||\mathbf{x}|| = ||\lambda \mathbf{x}|| = ||A\mathbf{x}|| \le ||A||\,||\mathbf{x}|| = ||A||. \tag{7.24}$$

Thus, the assertion follows. □

7.2.2. Convergent matrices

Definition 7.29. A matrix $A \in \mathbb{R}^{n\times n}$ is **convergent** if

$$\lim_{k\to\infty} (A^k)_{ij} = 0, \quad \text{for each } 1 \le i,j \le n. \tag{7.25}$$

Example 7.30. 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 7.31. 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 $\mathbf{x} \in \mathbb{R}^n$.

7.2.3. Invertible matrices

Definition 7.32. 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. The matrix B is called **an inverse of** A.

Note: We may prove that such a matrix B is unique. Thus B is the *unique inverse* of A denoted by A^{-1}.

Definition 7.33. Let $A = (a_{ij}) \in \mathbb{R}^{m\times n}$. The **transpose** of A is $A^T = (a_{ji})$. The matrix is **symmetric** if $A = A^T$.

Note: $(AB)^T = B^T A^T$.

Theorem 7.34.

1. *A square matrix can possess at most one right inverse.*
2. *If A and B are square matrices such that $AB = I$, then $BA = I$.*
3. *Let A and B be* **invertible** *square matrices. Then,*

$$(AB)^{-1} = B^{-1}A^{-1}.$$

Proof. Let's prove Part 2 only. Let $C = BA - I + B$. Then,

$$AC = ABA - A + AB = A - A + I = I.$$

By the uniqueness of the right inverse (Part 1), we can conclude $B = C$, which implies $BA - I = 0$. □

Theorem 7.35. (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. *$\det A \neq 0$.*

7.3. Iterative Algebraic Solvers

7.3.1. 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

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{n\times n} \tag{7.26}$$

can be described as follows.

- Split the matrix A as
$$A = M - N, \tag{7.27}$$
where M is an invertible matrix.
- Then, the system (7.26) can be expressed equivalently as
$$M\mathbf{x} = N\mathbf{x} + \mathbf{b}. \tag{7.28}$$
- Associated with the splitting is an iterative method
$$M\mathbf{x}^k = N\mathbf{x}^{k-1} + \mathbf{b} \quad \Rightarrow \quad \mathbf{x}^k = M^{-1}N\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \tag{7.29}$$
for a given initialization $\mathbf{x}^0$, $k \ge 1$.
- Since $N = M - A$, (7.29) can be rewritten as
$$\begin{aligned} \mathbf{x}^k &= (I - M^{-1}A)\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \\ &\text{or} \\ \mathbf{x}^k &= \mathbf{x}^{k-1} + M^{-1}(\mathbf{b} - A\mathbf{x}^{k-1}) \end{aligned} \tag{7.30}$$
Here, the matrix $I - M^{-1}A$ $(= M^{-1}N)$ is called the **iteration matrix**.

We shall say that the iterative method in (7.30) is **convergent** if it converges for any initial vector x^0. A sequence of vectors $\{\mathrm{x}^1, \mathrm{x}^2, \cdots\}$ will be computed from (7.30), and our objective is to choose M so that these two conditions are met:

1. The sequence $\{\mathrm{x}^k\}$ is easily computed. (Of course, M must be invertible.)
2. The sequence $\{\mathrm{x}^k\}$ 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: Recall (7.30):

$$\mathbf{x}^k = (I - M^{-1}A)\mathbf{x}^{k-1} + M^{-1}\mathbf{b} \quad (k \geq 1).$$

If the sequence $\{\mathrm{x}^k\}$ converges, say to a vector x, then it follows from (7.30) that

$$\mathbf{x} = (I - M^{-1}A)\mathbf{x} + M^{-1}\mathbf{b} \tag{7.31}$$

Thus, by letting $\mathbf{e}^k = \mathbf{x} - \mathbf{x}^k$, we have

$$\mathbf{e}^k = (I - M^{-1}A)\mathbf{e}^{k-1} \quad (k \geq 1), \tag{7.32}$$

which implies

$$\begin{aligned} ||\mathbf{e}^k|| &\leq ||I - M^{-1}A||\,||\mathbf{e}^{k-1}|| \\ &\leq ||I - M^{-1}A||^2\,||\mathbf{e}^{k-2}|| \\ &\leq \cdots \leq ||I - M^{-1}A||^k\,||\mathbf{e}^0|| \end{aligned} \tag{7.33}$$

Thus, it can be concluded as in the following theorem.

$\boxed{\textbf{Theorem}}$ **7.36. (Sufficient condition for convergence)**.
If $||I - M^{-1}A|| = ||M^{-1}N|| < 1$ *for some induced matrix norm, then the sequence produced by (7.30) converges to the solution of* $A\mathrm{x} = \mathrm{b}$ *for any initial vector* 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
$$||\mathbf{e}^k|| \le \delta(||\mathbf{e}^k|| + ||\mathbf{x}^k - \mathbf{x}^{k-1}||)$$
which implies
$$||\mathbf{e}^k|| \le \frac{\delta}{1-\delta}||\mathbf{x}^k - \mathbf{x}^{k-1}||. \tag{7.34}$$

Theorem 7.37. *The iteration (7.30) converges if and only if*
$$\rho(I - M^{-1}A) = \rho(M^{-1}N) < 1. \tag{7.35}$$

Note: An iterative algorithm converges if and only if its iteration matrix is convergent. See Theorem (7.31), p.248, for details of matrix convergence.

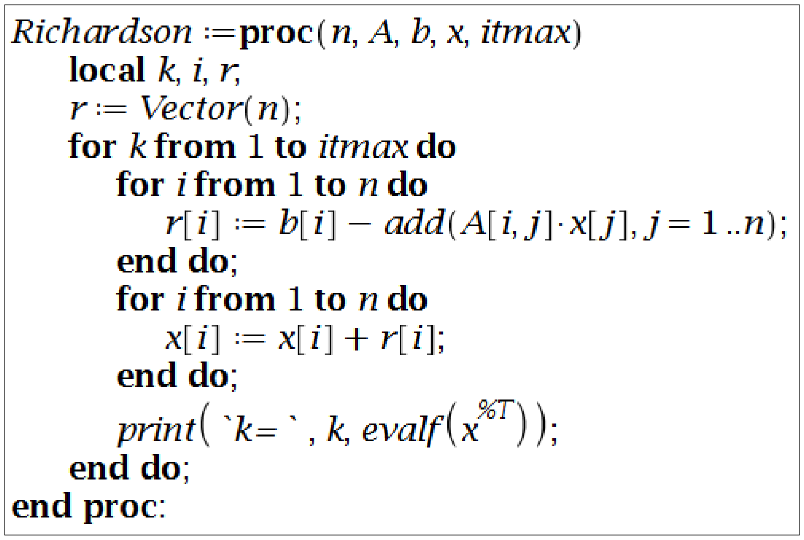
7.3.2. Richardson method: the simplest iteration

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 (7.30) reads

$$\mathbf{x}^k = \mathbf{x}^{k-1} + (\mathbf{b} - A\mathbf{x}^{k-1}). \tag{7.36}$$



```
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 7.1: 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

$$A\mathbf{x} = \mathbf{b}, \tag{7.37}$$

where some eigenvalues of A are **not** in $\{z \in \mathbb{C} : |z-1| < 1\}$.

- First, scale (7.37) with a constant η:

$$\eta A\mathbf{x} = \eta\mathbf{b}, \tag{7.38}$$

  where all the eigenvalues of ηA are in the open disk.
- Then, apply the Richardson method to (7.38):
  with $M = I$ and $N = I - \eta A$, the iteration reads

$$\begin{aligned} \mathbf{x}^k &= \mathbf{x}^{k-1} + (\eta\mathbf{b} - \eta A\mathbf{x}^{k-1}) \\ &= \mathbf{x}^{k-1} + \eta(\mathbf{b} - A\mathbf{x}^{k-1}). \end{aligned} \tag{7.39}$$

  Thus the Richardson method converges by choosing an appropriate **scaling factor** η.

Self-study 7.38. Let $A \in \mathbb{R}^{n\times n}$ be a **definite matrix**. Prove that the **generalized Richardson method** is convergent for the solution of $A\mathrm{x} = \mathrm{b}$.

Solution.

7.4. Relaxation Methods

Definition 7.39. 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:

$$A = D - E - F, \tag{7.40}$$

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 7.1: 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 $N = E + F$:

$$D\mathbf{x}^k = (E+F)\mathbf{x}^{k-1} + \mathbf{b}. \tag{7.41}$$

The i-th component of (7.41) reads

$$a_{ii}x_i^k = b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^{k-1}) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1}), \tag{7.42}$$

or, equivalently,

$$x_i^k = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k-1} - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big) \tag{7.43}$$

Example 7.40. 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.

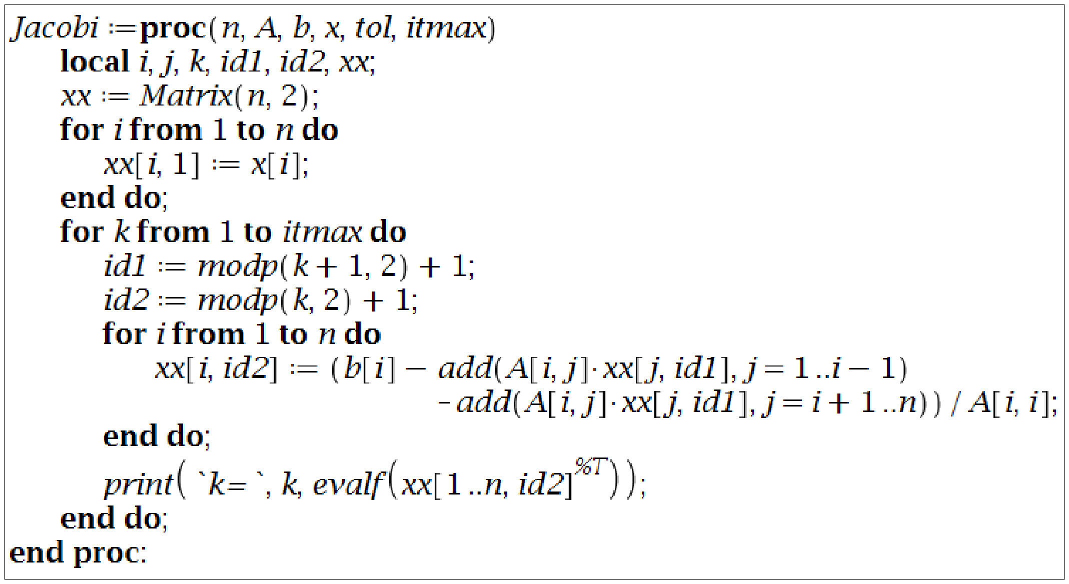
Manual checking in Maple

```
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 7.2: 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 $M = D - E$ and $N = F$:

$$(D - E)\mathbf{x}^k = F\mathbf{x}^{k-1} + \mathbf{b}. \tag{7.44}$$

Note that (7.44) can be equivalently written as

$$D\mathbf{x}^k = \mathbf{b} + E\mathbf{x}^k + F\mathbf{x}^{k-1}. \tag{7.45}$$

The i-th component of (7.45) reads

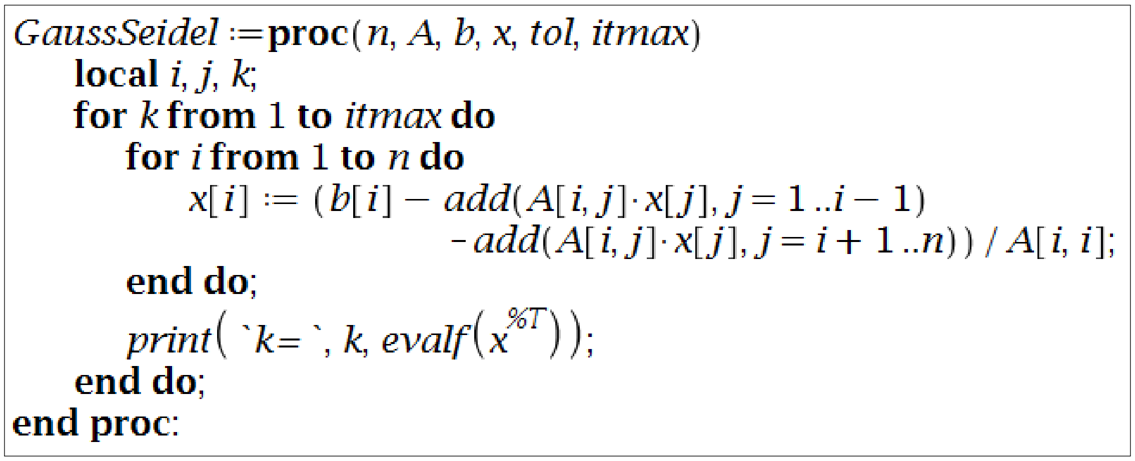
$$a_{ii}x_i^k = b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^k) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1}), \tag{7.46}$$

or, equivalently,

$$\boxed{x_i^k = \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big)} \tag{7.47}$$

The difference is that the SOR method utilizes updated values.

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 7.3: 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.

Note: By comparison with the result of the Jacobi method, we may conclude that Gauss-Seidel method is **twice faster** than the Jacobi method.

Successive Over Relaxation (SOR) Method

The **successive over relaxation** (**SOR**) is formulated with $M = \dfrac{1}{\omega}D - E$ and $N = \dfrac{1-\omega}{\omega}D + F$:

$$(D - \omega E)\mathbf{x}^k = \big((1-\omega)D + \omega F\big)\mathbf{x}^{k-1} + \omega\mathbf{b}. \tag{7.48}$$

Note that (7.48) can be equivalently written as

$$D\mathbf{x}^k = (1-\omega)D\mathbf{x}^{k-1} + \omega(\mathbf{b} + E\mathbf{x}^k + F\mathbf{x}^{k-1}). \tag{7.49}$$

The i-th component of (7.49) reads

$$a_{ii}x_i^k = (1-\omega)a_{ii}x_i^{k-1} + \omega\Big(b_i + \sum_{j=1}^{i-1}(-a_{ij}x_j^k) + \sum_{j=i+1}^{n}(-a_{ij}x_j^{k-1})\Big), \tag{7.50}$$

or, equivalently,

$$\begin{aligned} x_{GS,i}^k &= \frac{1}{a_{ii}}\Big(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\Big) \\ x_i^k &= (1-\omega)x_i^{k-1} + \omega x_{GS,i}^k \end{aligned} \tag{7.51}$$

Note that when $\omega = 1$, the SOR becomes the Gauss-Seidel 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 7.4: 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 $\approx$ 0.00000045 = 4.5E-7.

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

$\boxed{\textbf{Theorem}}$ **7.41.** *For and* $\mathbf{x}^0 \in \mathbb{R}^n$, *the sequence defined by*

$$\mathbf{x}^k = T\,\mathbf{x}^{k-1} + \mathbf{c} \tag{7.52}$$

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*

$$||\mathbf{x} - \mathbf{x}^k|| \le ||T||^k\, ||\mathbf{x} - \mathbf{x}^0||. \tag{7.53}$$

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 7.42. (Stein and Rosenberg, 1948) *[22]. 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 7.43. *Let A be symmetric. Then,*

$$\rho(T_{SOR}) < 1 \iff A \text{ is is positive definite and } 0 < \omega < 2. \tag{7.54}$$

Parameter 7.44. (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

$$\widehat{\omega} = \frac{2}{1 + \sqrt{1 - \rho(T_J)^2}}. \tag{7.55}$$

However, in many cases you can find a better ω for a given algebraic system.

Note: Let $0 < \rho(T_J) < 1$. Then the theoretically optimal SOR parameter

$$\begin{aligned} &1 < \widehat{\omega} < 2, \\ &\widehat{\omega} \approx 1 + \frac{1}{4}\rho(T_J)^2 + \frac{1}{8}\rho(T_J)^4. \end{aligned} \tag{7.56}$$

Remark 7.45.

- When $\omega > 1$, the blending of the SOR, the second equation in (7.51), 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)**.

7.5. Graph Theory: Estimation of the Spectral Radius

Definition 7.46. A **permutation matrix** is a square matrix in which each row and each column has one entry of unity, all others zero.

7.5.1. Irreducible matrices

Definition 7.47. For $n \geq 2$, an $n \times n$ complex-valued matrix A is **reducible** if there is a permutation matrix P such that

$$PAP^T = \begin{bmatrix} A_{11} & A_{12} \\ 0 & A_{22} \end{bmatrix},$$

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 geometrical interpretation of the concept 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 7.5.
- In this way, every $n \times n$ matrix A can be associated with a **directed graph** $G(A)$.

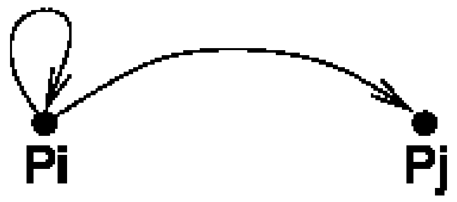


Figure 7.5: The directed paths for nonzero a_{ii} and a_{ij}.

Example 7.48. For example, the matrix

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix} \tag{7.57}$$

has a directed graph shown in Figure 7.6.

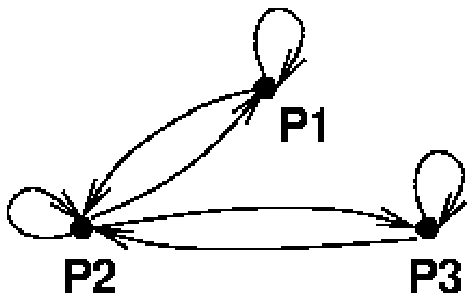


Figure 7.6: The directed graph $G(A)$ for A in (7.57).

Definition 7.49. 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

$$\overrightarrow{P_iP_{k_1}},\ \overrightarrow{P_{k_1}P_{k_2}},\ \cdots,\ \overrightarrow{P_{k_{r-1}}P_{k_r=j}},$$

connecting from P_i to P_j.

The theorems to be presented in this subsection can be found in (Varga, 2000) [25] along with their proofs.

Theorem 7.50. *An $n \times n$ complex-valued matrix A is irreducible if and only if its directed graph $G(A)$ is strongly connected.*

7.5.2. Eigenvalue locus theorem

For $A = [a_{ij}] \in \mathbb{C}^{n\times n}$, let

$$\Lambda_i := \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{ij}| \tag{7.58}$$

$\boxed{\textbf{Theorem}}$ **7.51. (Eigenvalue locus theorem)** *Let $A = [a_{ij}]$ be an irreducible $n\times n$ complex matrix. Then,*

1. **(Gerschgorin, 1931) [7]**: *All eigenvalues of A lie in the union of the disks in the complex plane*

$$|z - a_{ii}| \le \Lambda_i, \quad 1 \le i \le n. \tag{7.59}$$

2. **(Taussky, 1948) [23]**: *In addition, assume that λ, an eigenvalue of A, is a boundary point of the union of the disks $|z - a_{ii}| \le \Lambda_i$. 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 $1 \le i \le n$.*

Example 7.52. Consider an algebraic system $A\mathrm{x} = \mathrm{b}$, with

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{bmatrix}. \tag{7.60}$$

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 $-\dfrac{4}{5} < \Re(\lambda_i) < \dfrac{22}{25}$ and therefore $\rho(T_{SOR}) < \dfrac{22}{25}$.

$$Eigenvalues(TSOR)^{\%T} = \left[-\frac{1}{5} \quad \frac{4}{25} - \frac{3\,\mathrm{I}}{25} \quad \frac{4}{25} + \frac{3\,\mathrm{I}}{25} \right]$$

$$\rho(TSOR) = \max\left(\mathrm{abs}\left(\frac{4}{25} + \frac{3}{25}\,\mathrm{I}\right), \frac{1}{5}\right) = \frac{1}{5}$$

Positiveness

Definition 7.53. An $n \times n$ complex-valued matrix $A = [a_{ij}]$ is **diagonally dominant** if

$$|a_{ii}| \geq \Lambda_i := \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|, \tag{7.61}$$

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 (7.61) for at least one i.

Theorem 7.54. *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 7.55. *A Hermitian matrix satisfying the conditions in Theorem 7.54 is positive definite.*

7.5.3. Regular splitting and M-matrices

Definition 7.56. For $n \times n$ real matrices, A, M, and N, $A = M - N$ is a **regular splitting** of A if M is nonsingular with $M^{-1} \ge 0$, and $N \ge 0$.

Theorem 7.57. *If $A = M - N$ is a regular splitting of A and $A^{-1} \ge 0$, then*

$$\rho(M^{-1}N) = \frac{\rho(A^{-1}N)}{1 + \rho(A^{-1}N)} < 1. \tag{7.62}$$

Thus, the matrix $M^{-1}N$ is convergent and the iterative method (7.29) converges for any initial value $\mathbf{x}^0$.

Definition 7.58. An $n \times n$ real matrix $A = [a_{ij}]$ with $a_{ij} \le 0$ for all $i \neq j$ is an **M-matrix** if A is nonsingular and $A^{-1} \ge 0$.

Theorem 7.59. *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 7.60. *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 $N_2 \ge N_1 \ge 0$, where neither $N_2 - N_1$ nor N_1 is null, then*

$$1 > \rho(M_2^{-1}N_2) > \rho(M_1^{-1}N_1) > 0. \tag{7.63}$$

7.6. Krylov Subspace Methods

Definition 7.61. A matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$ is said to be **positive definite** if

$$\mathbf{x}^T A\mathbf{x} = \sum_{i,j=1}^{n} x_i a_{ij} x_j > 0, \quad \forall\, \mathbf{x} \in \mathbb{R}^n, \quad \mathbf{x} \neq 0. \tag{7.64}$$

For solving a linear system

$$A\mathbf{x} = \mathbf{b}, \tag{7.65}$$

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k, \quad k = 0, 1, \cdots, \tag{7.66}$$

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**.
- For other Krylov subspace methods, see e.g. [1, 15].

Remark 7.62. The algebraic system (7.65) admits a unique solution $\mathbf{x} \in \mathbb{R}^n$, which is equivalently characterized by

$$\mathbf{x} = \arg\min_{\eta\in\mathbb{R}^n} f(\eta), \quad f(\eta) = \frac{1}{2}\eta\cdot A\eta - \mathbf{b}\cdot\eta. \tag{7.67}$$

7.6.1. 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:
$$f'(\eta) = A\eta - \mathbf{b}, \quad f''(\eta) = A. \tag{7.68}$$
- Given $\mathbf{x}_{k+1}$ as in (7.66), we have by Taylor's formula
$$\begin{aligned} f(\mathbf{x}_{k+1}) &= f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \\ &= f(\mathbf{x}_k) + \alpha_k f'(\mathbf{x}_k)\cdot \mathbf{p}_k + \frac{\alpha_k^2}{2}\mathbf{p}_k \cdot f''(\boldsymbol{\xi})\mathbf{p}_k, \end{aligned} \tag{7.69}$$
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
$$f(\mathbf{x}_{k+1}) < f(\mathbf{x}_k), \tag{7.70}$$
which can be achieved if
$$f'(\mathbf{x}_k)\cdot \mathbf{p}_k < 0 \tag{7.71}$$
and either γ_k is sufficiently small or $f''(\boldsymbol{\xi})$ is nonnegative.
- **Choice**: When $f'(\mathbf{x}_k) \neq 0$, (7.71) holds, if we choose:
$$\mathbf{p}_k = -f'(\mathbf{x}_k) = \mathbf{b} - A\mathbf{x}_k =: \mathbf{r}_k \tag{7.72}$$
That is, the search direction is the **negative gradient**, the steepest descent direction.

Optimal step length: We may determine α_k such that

$$f(\mathbf{x}_k + \alpha_k \mathbf{p}_k) = \min_{\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k), \tag{7.73}$$

in which case α_k is said to be **optimal**.

If α_k is optimal, then

$$\begin{aligned} 0 &= \frac{\mathbf{d}}{\mathbf{d}\alpha} f(\mathbf{x}_k + \alpha \mathbf{p}_k)\Big|_{\alpha=\alpha_k} = f'(\mathbf{x}_k + \alpha_k \mathbf{p}_k) \cdot \mathbf{p}_k \\ &= (A(\mathbf{x}_k + \alpha_k \mathbf{p}_k) - \mathbf{b}) \cdot \mathbf{p}_k \\ &= (A\mathbf{x}_k - \mathbf{b}) \cdot \mathbf{p}_k + \alpha_k \mathbf{p}_k \cdot A\mathbf{p}_k. \end{aligned} \tag{7.74}$$

So,

$$\alpha_k = \frac{\mathbf{r}_k \cdot \mathbf{p}_k}{\mathbf{p}_k \cdot A\mathbf{p}_k}. \tag{7.75}$$

Theorem 7.63. (Convergence of the GD method): *The GD method converges, satisfying*

$$\|\,\mathbf{x} - \mathbf{x}_k\,\|_2 \le \left(1 - \frac{1}{\kappa(A)}\right)^k \|\,\mathbf{x} - \mathbf{x}_0\,\|_2. \tag{7.76}$$

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:

$$k \;\ge\; \kappa(A) \log \frac{1}{\varepsilon}. \tag{7.77}$$

7.6.2. Conjugate gradient (CG) method

In this method the search directions p_k are **conjugate**, i.e.,

$$\mathbf{p}_i \cdot A\mathbf{p}_j = 0, \quad i \ne j, \tag{7.78}$$

and the step length α_k is chosen to be optimal.

Algorithm 7.64. (CG Algorithm, V.1)

```
Select $\mathbf{x}_0$, $\varepsilon$;
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0$, $\mathbf{p}_0 = \mathbf{r}_0$;
Do $k = 0, 1, \cdots$
    $\alpha_k = \mathbf{r}_k \cdot \mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k$;            (CG1)
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;                        (CG2)
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$;                       (CG3)      (7.79)
    if $\| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2$, stop;
    $\beta_k = -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k$;      (CG4)
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$;                     (CG5)
End Do
```

Remark 7.65. (CG Algorithm, V.1)

- In practice, $\mathbf{q}_k = A\mathbf{p}_k$ 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 (7.75). 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 $\mathbf{p}_{k+1} \cdot A\mathbf{p}_k = 0$. Verify it using $\mathbf{p}_{k+1}$ in (CG5).

Theorem 7.66. *For* $m = 0, 1, \cdots,$

$$\begin{aligned} \text{span}\{\mathbf{p}_0, \cdots, \mathbf{p}_m\} &= \text{span}\{\mathbf{r}_0, \cdots, \mathbf{r}_m\} \\ &= \text{span}\{\mathbf{r}_0, A\mathbf{r}_0, \cdots, A^m \mathbf{r}_0\}. \end{aligned} \tag{7.80}$$

Theorem 7.67. *The search directions and the residuals satisfy the orthogonality,*

$$\mathbf{p}_i \cdot A\mathbf{p}_j = 0; \quad \mathbf{r}_i \cdot \mathbf{r}_j = 0, \quad i \neq j. \tag{7.81}$$

Theorem 7.68. *For some $m \le n$, we have $A\mathbf{x}_m = \mathbf{b}$ and*

$$\| \mathbf{x} - \mathbf{x}_k \|_A \le 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \| \mathbf{x} - \mathbf{x}_0 \|_A. \tag{7.82}$$

So the required iteration number to reduce the error by a factor of ε is

$$k \ge \frac{1}{2} \sqrt{\kappa(A)} \, \log \frac{2}{\varepsilon}. \tag{7.83}$$

Proofs of the above theorems can be found in e.g. [13].

Simplification of the CG method: Using the properties and identities involved in the method, one can derive a more popular form of the CG method.

Algorithm 7.69. (CG Algorithm, V.2)

```
Select $\mathbf{x}_0, \ \varepsilon$;
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0, \ \ \mathbf{p}_0 = \mathbf{r}_0$;
Compute $\rho_0 = \mathbf{r}_0 \cdot \mathbf{r}_0$;
Do $k = 0, 1, \cdots$
    $\alpha_k = \rho_k / \mathbf{p}_k \cdot A\mathbf{p}_k$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$;                          (7.84)
    if $\| \mathbf{r}_{k+1} \|_2 < \varepsilon \| \mathbf{r}_0 \|_2$, stop;
    $\rho_{k+1} = \mathbf{r}_{k+1} \cdot \mathbf{r}_{k+1}$;
    $\beta_k = \rho_{k+1} / \rho_k$;
    $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$;
End Do
```

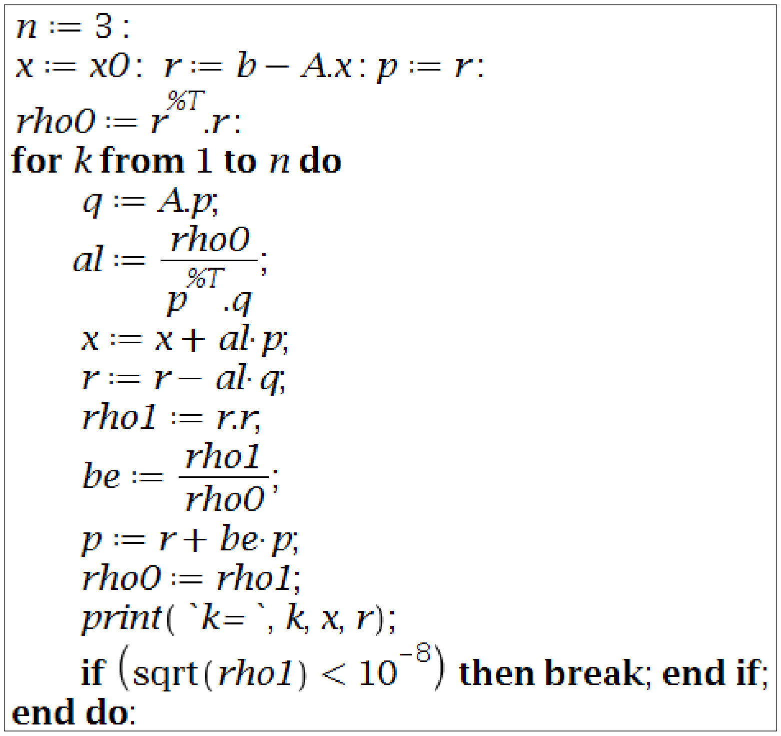
Note:

$$\begin{aligned} \mathbf{r}_k \cdot \mathbf{p}_k &= \mathbf{r}_k \cdot (\mathbf{r}_k + \beta_{k-1}\mathbf{p}_{k-1}) = \mathbf{r}_k \cdot \mathbf{r}_k, \\ \beta_k &= -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k / \mathbf{p}_k \cdot A\mathbf{p}_k = -\mathbf{r}_{k+1} \cdot A\mathbf{p}_k \frac{\alpha_k}{\rho_k} \\ &= \mathbf{r}_{k+1} \cdot (\mathbf{r}_{k+1} - \mathbf{r}_k)/\rho_k = \rho_{k+1}/\rho_k. \end{aligned} \tag{7.85}$$

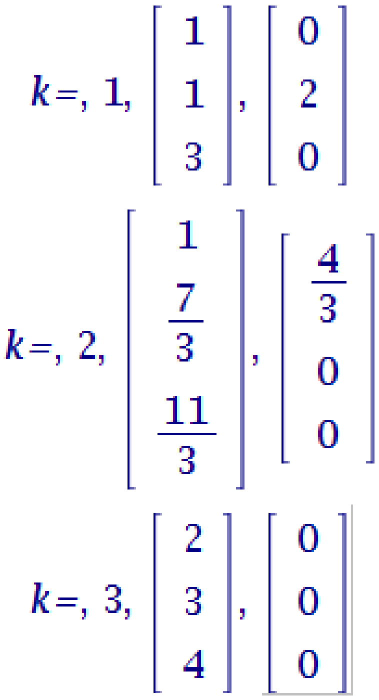
Example 7.70. (Revisit to Example 7.40)

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 CG method to find $\mathbf{x}^k$, $k = 1, 2, 3$, beginning from $\mathbf{x}^0 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$.

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 7.71.

- For the example, the CG **converged completely** in three iterations.
- The CG method was originally developed **as a direct solver**.

7.6.3. 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
$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}. \tag{7.86}$$
- Since
$$\kappa(M^{-1}A) \ll \kappa(A) \tag{7.87}$$
(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 7.72. (Preconditioned CG)

```
Select $\mathbf{x}_0,\ \varepsilon$;
$\mathbf{r}_0 = \mathbf{b} - A\mathbf{x}_0,\ \ M\mathbf{z}_0 = \mathbf{r}_0$;
$\mathbf{p}_0 = \mathbf{z}_0$, compute $\rho_0 = \mathbf{z}_0^*\mathbf{r}_0$;
Do $k = 0, 1, \cdots$
    $\alpha_k = \rho_k / \mathbf{p}_k^* A\mathbf{p}_k$;
    $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k\mathbf{p}_k$;
    $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k A\mathbf{p}_k$;                    (7.88)
    if $\|\mathbf{r}_{k+1}\|_2 < \varepsilon \|\mathbf{r}_0\|_2$, stop;
    $M\mathbf{z}_{k+1} = \mathbf{r}_{k+1}$;
    $\rho_{k+1} = \mathbf{z}_{k+1}^*\mathbf{r}_{k+1}$;
    $\beta_k = \rho_{k+1}/\rho_k$;
    $\mathbf{p}_{k+1} = \mathbf{z}_{k+1} + \beta_k\mathbf{p}_k$;
End Do
```

Here the superscript * indicates the transpose complex-conjugate; it is the transpose for real-valued systems.

7.7. Generalized Minimum Residuals – GMRES

Note: GMRES was developed in 1986 by Saad and Schultz [21].

- The GMRES can be used for linear systems $A\mathrm{x} = \mathrm{b}$, 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**.

7.7.1. Arnoldi Iteration

Note: The Arnoldi iteration method will be applicable to both linear systems and eigenvalue problems.

- Thus we are interested in re-examining similarity transformations of the form
$$A = QHQ^T, \tag{7.89}$$
where 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

$$A[\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_k,\mathbf{q}_{k+1},\cdots,\mathbf{q}_n] = [\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_k,\mathbf{q}_{k+1},\cdots,\mathbf{q}_n]\times \\ \times\begin{bmatrix} h_{11} & h_{12} & & \cdots & \cdots & h_{1n} \\ h_{21} & h_{22} & & \cdots & \cdots & h_{2n} \\ 0 & h_{32} & h_{33} & \cdots & \cdots & h_{3n} \\ \vdots & 0 & h_{43} & h_{44} & \cdots & h_{4n} \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & 0 & h_{n-1,n} & h_{nn} \end{bmatrix}. \tag{7.90}$$

- Next, we consider only part of this system. For $k < n$, let

$$\begin{aligned} Q_k &= [\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_k] \\ Q_{k+1} &= [\mathbf{q}_1,\mathbf{q}_2,\cdots,\mathbf{q}_k,\mathbf{q}_{k+1}] \\ \widetilde{H}_k &= \begin{bmatrix} h_{11} & h_{12} & & \cdots & \cdots & h_{1k} \\ h_{21} & h_{22} & & \cdots & \cdots & h_{2k} \\ 0 & h_{32} & h_{33} & \cdots & \cdots & h_{3k} \\ & 0 & h_{43} & h_{44} & \cdots & h_{4k} \\ \vdots & & \ddots & \ddots & \ddots & \vdots \\ & & & 0 & h_{k-1,k} & h_{kk} \\ 0 & \cdots & \cdots & & 0 & h_{k+1,k} \end{bmatrix} \end{aligned} \tag{7.91}$$

and then take

$$AQ_k = Q_{k+1}\widetilde{H}_k \quad (\in \mathbb{R}^{n\times k}). \tag{7.92}$$

- If we compare the kth columns on both sides, then we get

$$A\mathbf{q}_k = h_{1k}\mathbf{q}_1 + h_{2k}\mathbf{q}_2 + \cdots + h_{kk}\mathbf{q}_k + h_{k+1,k}\mathbf{q}_{k+1}, \tag{7.93}$$

which can be rewritten for $\mathbf{q}_{k+1}$:

$$\mathbf{q}_{k+1} = \frac{A\mathbf{q}_k - \sum_{j=1}^{k} h_{jk}\mathbf{q}_j}{h_{k+1,k}}, \tag{7.94}$$

which is a modified **Gram-Schmidt process**.

- It follows from (7.93) that

$$h_{jk} = \mathbf{q}_j^T A\mathbf{q}_k. \tag{7.95}$$

Since $||\mathbf{q}_{k+1}|| = 1$, we must have

$$h_{k+1,k} = \Big\|A\mathbf{q}_k - \sum_{j=1}^{k} h_{jk}\mathbf{q}_j\Big\|. \tag{7.96}$$

The recursive computation of the columns of Q, given as in (7.94), is known as the **Arnoldi iteration**.

Example 7.73. The first step of the Arnoldi iteration proceeds as follows.

- Start with an arbitrary normalized vector $\mathbf{q}_1$. Then, according to (7.94),

$$\mathbf{q}_2 = \frac{A\mathbf{q}_1 - h_{11}\mathbf{q}_1}{h_{21}}. \tag{7.97}$$

- Since it requires $\mathbf{q}_2^T\mathbf{q}_1 = 0$, orthogonality of the columns of Q, we get

$$0 \;=\; \mathbf{q}_1^T A\mathbf{q}_1 - h_{11}\mathbf{q}_1^T\mathbf{q}_1 \;=\; \mathbf{q}_1^T A\mathbf{q}_1 - h_{11},$$

  and therefore h_{11} is a **Rayleigh quotient**:

$$h_{11} \;=\; \mathbf{q}_1^T A\mathbf{q}_1. \tag{7.98}$$

- Now, we set

$$h_{21} = ||A\mathbf{q}_1 - h_{11}\mathbf{q}_1||, \tag{7.99}$$

  which finalizes $\mathbf{q}_2$. □

Algorithm 7.74. Arnoldi iteration

```
0. Let b be an arbitrary initial nonzero vector
1. $\mathbf{q}_1 = \mathbf{b}/||\mathbf{b}||$
2. For $k = 1, 2, 3, \cdots$
   (a) $\mathbf{v} = A\mathbf{q}_k$
   (b) For $j = 1 : k$
           $h_{jk} = \mathbf{q}_j^T\mathbf{v}$
           $\mathbf{v} = \mathbf{v} - h_{jk}\mathbf{q}_j$
   (c) $h_{k+1,k} = ||\mathbf{v}||$
   (d) $\mathbf{q}_{k+1} = \mathbf{v}/h_{k+1,k}$
```

7.7.2. Derivation of the GMRES Method

Key Idea 7.75. 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

$$\mathbf{x}_k = \arg\min_{\mathbf{x}\in\mathcal{K}_k} ||A\mathbf{x} - \mathbf{b}||, \tag{7.100}$$

where $\mathcal{K}_k$ is the ***k*****th-order Krylov subspace**:

$$\mathcal{K}_k = \mathit{Span}\{\mathbf{b}, A\mathbf{b}, \cdots, A^{k-1}\mathbf{b}\} \tag{7.101}$$

Note: In order to solve the least-squares problem (7.100):

- We may start with the **Krylov matrix**

$$K_k \overset{\text{def}}{=\!=} \{\mathbf{b},\ A\mathbf{b},\ \cdots,\ A^{k-1}\mathbf{b}\} \in \mathbb{R}^{n\times k}. \tag{7.102}$$

- Then the desired solution $\mathbf{x}_k \in \mathcal{K}_k$ can be expressed as

$$\mathbf{x}_k = K_k\mathbf{c}, \ \text{ for some } \mathbf{c} \in \mathbb{R}^k, \tag{7.103}$$

  and therefore the least-squares problem reads

$$\min_{\mathbf{c}\in\mathbb{R}^k} ||AK_k\mathbf{c} - \mathbf{b}||. \tag{7.104}$$

- The problem (7.104) 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 $\mathcal{K}_k$:

$$\{\mathbf{q}_1,\, \mathbf{q}_2, \cdots,\, \mathbf{q}_k\} \tag{7.105}$$

  the columns of the matrix Q_k used in the **Arnoldi iteration**.

- With the new basis, the kth iterate $\mathbf{x}_k \in \mathcal{K}_k$ can be written as

$$\mathbf{x}_k = Q_k\mathbf{y}, \text{ for some } \mathbf{y} \in \mathbb{R}^k. \tag{7.106}$$

  Then the residual minimization problem (7.100) becomes

$$\min_{\mathbf{y}\in\mathbb{R}^k} ||AQ_k\mathbf{y} - \mathbf{b}||. \tag{7.107}$$

- Now, we utilize the **partial similarity transform** of the Arnoldi iteration (7.92):

$$AQ_k = Q_{k+1}\widetilde{H}_k, \quad \widetilde{H}_k \in \mathbb{R}^{(k+1)\times k}. \tag{7.108}$$

- Then, the least-squares problem (7.107) reads

$$\min_{\mathbf{y}\in\mathbb{R}^k} \left|\left|\widetilde{H}_k\mathbf{y} - Q_{k+1}^T\mathbf{b}\right|\right|, \tag{7.109}$$

  which is equivalent to

$$\min_{\mathbf{y}\in\mathbb{R}^k} \left|\left|\widetilde{H}_k\mathbf{y} - ||\mathbf{b}||\mathbf{e}_1\right|\right|. \tag{7.110}$$

Note: **This is much simpler**; it will permit a more efficient solution.

Example 7.76. 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

$$Q_{k+1}^T \mathbf{b} = \begin{bmatrix} \mathbf{q}_1^T \mathbf{b} \\ \mathbf{q}_2^T \mathbf{b} \\ \vdots \\ \mathbf{q}_{k+1}^T \mathbf{b} \end{bmatrix}, \tag{7.111}$$

and, in detail, the Krylov subspaces are

$$\begin{aligned} \mathcal{K}_1 &= \mathit{Span}\{\mathbf{b}\}, \\ \mathcal{K}_1 &= \mathit{Span}\{\mathbf{b},\, A\mathbf{b}\}, \\ &\vdots \end{aligned} \tag{7.112}$$

Since the columns $\mathbf{q}_j$ of Q_k form an orthonormal basis for $\mathcal{K}_k$,

$$\mathbf{q}_1 = \frac{\mathbf{b}}{||\mathbf{b}||}, \quad \text{and } \mathbf{q}_j^T \mathbf{b} = 0 \text{ for } j > 1. \tag{7.113}$$

It follows from (7.111) and (7.113) that

$$Q_{k+1}^T \mathbf{b} = ||\mathbf{b}||\mathbf{e}_1, \tag{7.114}$$

which completes the proof. □

The GMRES method is summarized as follows.

Algorithm 7.77. GMRES

```
1. Let $\mathbf{q}_1 = \mathbf{b}/||\mathbf{b}||$
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 $\big|\big|\widetilde{H}_k\mathbf{y} - ||\mathbf{b}||\mathbf{e}_1\big|\big|$
          (e.g., with $QR$-factorization)
    (c) Set $\mathbf{x}_k = Q_k\mathbf{y}$
```

Exercises for Chapter 7

7.1. (a) Verify that the function $||\cdot||_1$, defined on $\mathbb{R}^n$ by

$$||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$$

is a norm on $\mathbb{R}^n$.

(b) Prove that $||\mathbf{x}||_2 \le ||\mathbf{x}||_1$ for all $\mathbf{x} \in \mathbb{R}^n$.

7.2. Let $A = \begin{bmatrix} 2 & 1 & -2 \\ 0 & 3 & -1 \\ 4 & 5 & 1 \end{bmatrix}$. Find $||A||_1$, $||A||_2$, and $||A||_\infty$.

7.3. 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?

7.4. C When the **boundary-value problem**

$$\begin{cases} -u_{xx} = -2, & 0 < x < 4 \\ u_x(0) = 0, & u(4) = 16 \end{cases} \tag{7.115}$$

is discretized by the second-order finite difference method with $h = 1$, the algebraic system reads $A\mathrm{x} = \mathrm{b}$, where

$$A = \begin{bmatrix} 2 & -2 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -2 \\ -2 \\ -2 \\ 14 \end{bmatrix} \tag{7.116}$$

and the exact solution is $\mathrm{x} = [0,\, 1,\, 4,\, 9]^T$.

(a) Is A irreducibly diagonally dominant?

(b) Perform 10 iterations of the Jacobi and Gauss-Seidel methods, starting from $\mathrm{x}_0 = [0,\, 0,\, 0,\, 0]^T$.

(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.

7.5. C Symmetrize the algebraic system (7.116) 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 8
Approximation Theory

In this chapter:

Topics	**Applications/Properties**
Orthogonal Projection	Gram-Schmidt Process
Least Squares (LS) Approximation	Over-determined systems
Linear models	
Method of normal equations	
Regression analysis	
Linear regression	
Nonlinear models	Linearization
Scene analysis	Noisy data
RANSAC	
Rational Function Approximation	Padé rational approximation

Contents of Chapter 8

8.1. Orthogonality and the Gram-Schmidt Process

8.1.1. Orthogonal projections

Definition 8.1. Given a nonzero vector $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$, let

$$\mathbf{y} = \widehat{\mathbf{y}} + \mathbf{z}, \quad \widehat{\mathbf{y}} \,//\, \mathbf{u} \text{ and } \mathbf{z} \perp \mathbf{u}. \tag{8.1}$$

Then

$$\widehat{\mathbf{y}} = \alpha\mathbf{u} = \frac{\mathbf{y}\bullet\mathbf{u}}{\mathbf{u}\bullet\mathbf{u}}\mathbf{u}, \quad \mathbf{z} = \mathbf{y} - \widehat{\mathbf{y}}. \tag{8.2}$$

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

$$\widehat{\mathbf{y}} = \frac{\mathbf{y}\bullet\mathbf{u}}{\mathbf{u}\bullet\mathbf{u}}\mathbf{u} = \text{proj}_L\mathbf{y}, \tag{8.3}$$

which is called the orthogonal projection of $\mathbf{y}$ onto L.

We will expand this orthogonal projection for a subspace.

Theorem 8.2. (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*

$$\mathbf{y} = \widehat{\mathbf{y}} + \mathbf{z}, \tag{8.4}$$

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

$$\begin{aligned} \widehat{\mathbf{y}} &= \text{proj}_W\mathbf{y} = \frac{\mathbf{y}\bullet\mathbf{u}_1}{\mathbf{u}_1\bullet\mathbf{u}_1}\mathbf{u}_1 + \frac{\mathbf{y}\bullet\mathbf{u}_2}{\mathbf{u}_2\bullet\mathbf{u}_2}\mathbf{u}_2 + \cdots + \frac{\mathbf{y}\bullet\mathbf{u}_p}{\mathbf{u}_p\bullet\mathbf{u}_p}\mathbf{u}_p, \\ \mathbf{z} &= \mathbf{y} - \widehat{\mathbf{y}}. \end{aligned} \tag{8.5}$$

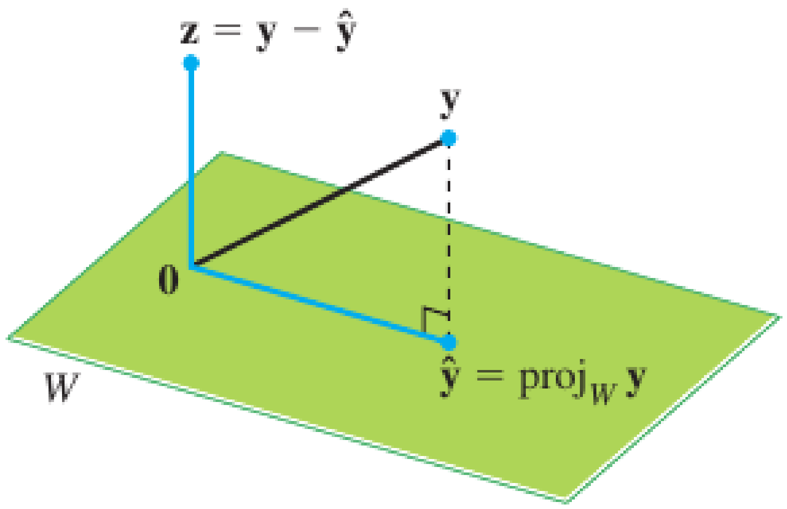

$\mathbf{z} = \mathbf{y} - \hat{\mathbf{y}}$
$\mathbf{y}$
$\mathbf{0}$
W
$\hat{\mathbf{y}} = \text{proj}_W \mathbf{y}$

Figure 8.1: Orthogonal projection of y onto W.

Remark 8.3. (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 $\| \mathbf{y} - \widehat{\mathbf{y}} \|_2 \le \| \mathbf{y} - \mathbf{v} \|_2$, 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*,

$$\| \mathbf{y} - \mathbf{v} \|_2^2 = \| \mathbf{y} - \widehat{\mathbf{y}} \|_2^2 + \| \widehat{\mathbf{y}} - \mathbf{v} \|_2^2,$$

which implies that $\| \mathbf{y} - \mathbf{v} \|_2 \ge \| \mathbf{y} - \widehat{\mathbf{y}} \|_2$. □

Note: The orthogonal projection can be viewed as a **matrix transformation**; see Exercise 8.1, p.310.

8.1.2. 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 8.4. 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.

Main 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. $\alpha = \dfrac{\mathbf{x}_2 \bullet \mathbf{x}_1}{\mathbf{x}_1 \bullet \mathbf{x}_1} = \dfrac{15}{45} = \dfrac{1}{3}$.

$$\Rightarrow \mathbf{v}_2 = \mathbf{x}_2 - \frac{1}{3}\mathbf{x}_1 = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix} - \frac{1}{3}\begin{bmatrix} 3 \\ 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix}$$

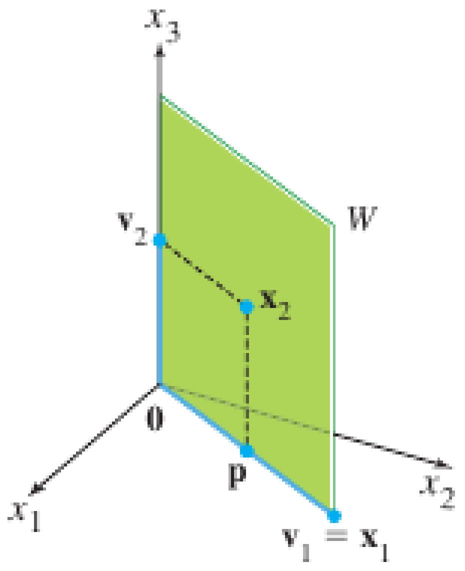


Figure 8.2: Construction of an orthogonal basis $\{\mathbf{v}_1, \mathbf{v}_2\}$.

Theorem 8.5. **(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*

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{x}_1 \\
\mathbf{v}_2 &= \mathbf{x}_2 - \frac{\mathbf{x_2} \bullet \mathbf{v_1}}{\mathbf{v_1} \bullet \mathbf{v_1}} \mathbf{v}_1 \\
\mathbf{v}_3 &= \mathbf{x}_3 - \frac{\mathbf{x_3} \bullet \mathbf{v_1}}{\mathbf{v_1} \bullet \mathbf{v_1}} \mathbf{v}_1 - \frac{\mathbf{x_3} \bullet \mathbf{v_2}}{\mathbf{v_2} \bullet \mathbf{v_2}} \mathbf{v}_2 \\
&\vdots \\
\mathbf{v}_p &= \mathbf{x}_p - \frac{\mathbf{x_p} \bullet \mathbf{v_1}}{\mathbf{v_1} \bullet \mathbf{v_1}} \mathbf{v}_1 - \frac{\mathbf{x_p} \bullet \mathbf{v_2}}{\mathbf{v_2} \bullet \mathbf{v_2}} \mathbf{v}_2 - \cdots - \frac{\mathbf{x_p} \bullet \mathbf{v}_{p-1}}{\mathbf{v}_{p-1} \bullet \mathbf{v}_{p-1}} \mathbf{v}_{p-1}
\end{aligned} \tag{8.6}$$

Then $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ *is an* ***orthogonal basis*** *for* W*. In addition,*

$$\textit{Span}\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\} = \textit{Span}\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\}, \quad \textit{for } 1 \le k \le p. \tag{8.7}$$

Remark 8.6. For the result of the Gram-Schmidt process, define

$$\mathbf{u}_k = \frac{\mathbf{v}_k}{\|\mathbf{v}_k\|}, \quad \text{for } 1 \le k \le p. \tag{8.8}$$

Then $\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_p\}$ is an **orthonormal basis** for W. In practice, it is often implemented with the **normalized Gram-Schmidt process**.

Self-study 8.7. Find an *orthonormal basis* for $W = \textit{Span}\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$, where

$$\mathbf{x}_1 = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 1 \end{bmatrix}, \mathbf{x}_2 = \begin{bmatrix} -2 \\ 2 \\ 1 \\ 0 \end{bmatrix}, \text{ and } \mathbf{x}_3 = \begin{bmatrix} 0 \\ 1 \\ -1 \\ 1 \end{bmatrix}.$$

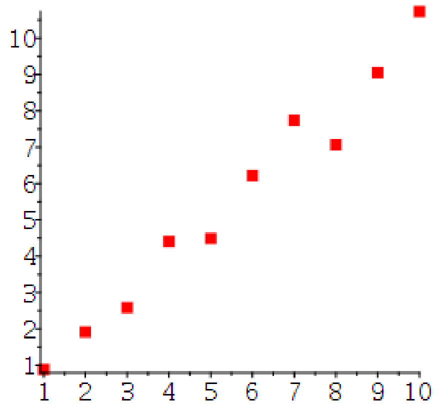
8.2. Least-Squares Approximation

Definition 8.8. 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);
```

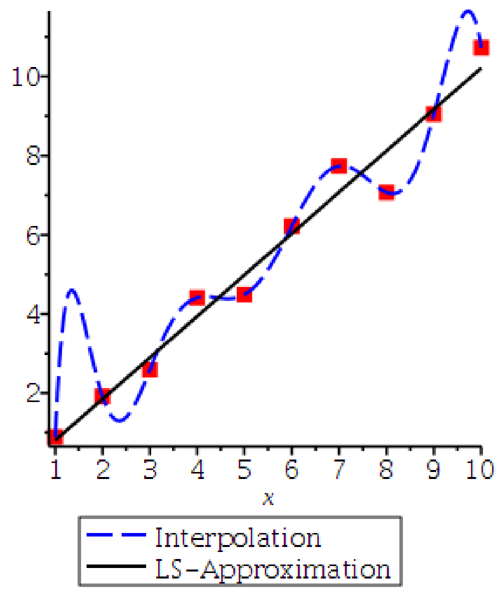


p := *PolynomialInterpolation*(*xy*, *x*);

$$\frac{1507}{12096000}x^9 - \frac{27977}{4032000}x^8 + \frac{36751}{224000}x^7 - \frac{68851}{32000}x^6 + \frac{1975291}{115200}x^5 - \frac{5471791}{64000}x^4 + \frac{199836041}{756000}x^3 - \frac{486853651}{1008000}x^2 + \frac{39227227}{84000}x - \frac{1771}{10}$$

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.

8.2.1. 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 8.9. Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$, and $\mathbf{b} \in \mathbb{R}^m$. The **least-squares problem** is to find $\widehat{\mathbf{x}} \in \mathbb{R}^n$ which minimizes $\|A\mathbf{x} - \mathbf{b}\|_2$:

$$\begin{aligned} \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2, \\ &\quad \text{or, equivalently,} \\ \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x} - \mathbf{b}\|_2^2, \end{aligned} \tag{8.9}$$

where $\widehat{\mathbf{x}}$ called a **least-squares solution** of $A\mathbf{x} = \mathbf{b}$.

Remark 8.10.

- For all $\mathrm{x} \in \mathbb{R}^n$, $A\mathrm{x}$ will necessarily be in *Col* A, a subspace of $\mathbb{R}^m$. So we seek an x that makes $A\mathrm{x}$ the closest point in *Col* A to b.
- Let $\widehat{\mathrm{b}} = \mathrm{proj}_{\mathit{Col}\,A}\mathrm{b}$. Then $A\mathrm{x} = \widehat{\mathrm{b}}$ has a solution and there is an $\widehat{\mathrm{x}} \in \mathbb{R}^n$ such that
$$A\widehat{\mathrm{x}} = \widehat{\mathrm{b}}. \tag{8.10}$$
- $\widehat{\mathrm{x}}$ is an LS solution of $A\mathrm{x} = \mathrm{b}$.
- The quantity $\|\mathrm{b} - \widehat{\mathrm{b}}\| = \|\mathrm{b} - A\widehat{\mathrm{x}}\|$ is called the **least-squares error**. That is,
$$\|\mathrm{b} - A\widehat{\mathrm{x}}\| \le \|\mathrm{b} - A\mathrm{x}\|, \quad \text{for all } \mathrm{x} \in \mathbb{R}^n, \tag{8.11}$$
where $\|\cdot\| = \|\cdot\|_2$.

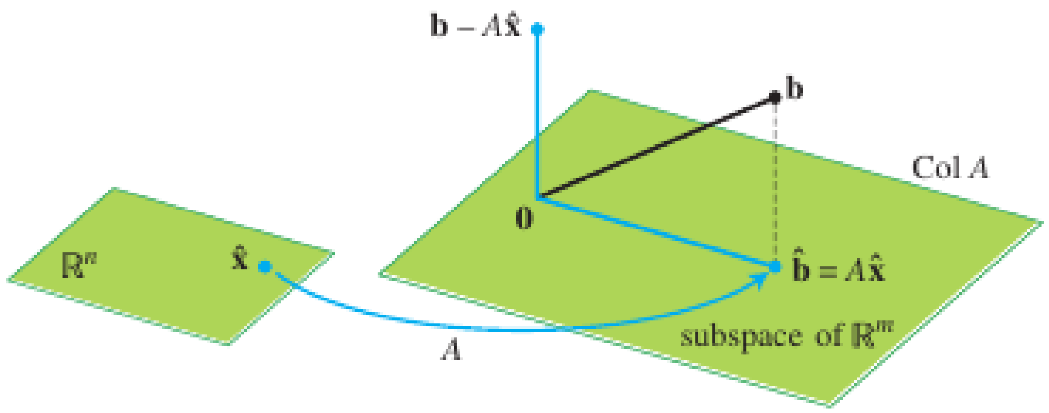


Figure 8.3: The LS solution $\widehat{\mathrm{x}}$ is in $\mathbb{R}^n$.

Remark 8.11. If $A \in \mathbb{R}^{n\times n}$ is invertible, then $A\mathrm{x} = \mathrm{b}$ has a unique solution $\widehat{\mathrm{x}}$ and therefore
$$\|\mathrm{b} - A\widehat{\mathrm{x}}\| = 0. \tag{8.12}$$

8.2.2. Normal equations

Theorem 8.12. *The set of LS solutions of $A\mathrm{x} = \mathrm{b}$ coincides with the nonempty set of solutions of the* ***normal equations***

$$A^T A\mathbf{x} = A^T\mathbf{b}. \tag{8.13}$$

Proof. Suppose $\widehat{\mathrm{x}}$ satisfies $A\widehat{\mathrm{x}} = \widehat{\mathrm{b}}$

$\Rightarrow \mathbf{b} - \widehat{\mathbf{b}} = \mathbf{b} - A\widehat{\mathbf{x}} \;\perp\; \mathit{Col}\, A$

$\Rightarrow \mathbf{a}_j \bullet (\mathbf{b} - A\widehat{\mathbf{x}}) = 0$ for all columns $\mathbf{a}_j$

$\Rightarrow {\mathbf{a}_j}^T(\mathbf{b} - A\widehat{\mathbf{x}}) = 0$ for all columns $\mathbf{a}_j$ (Note that ${\mathbf{a}_j}^T$ is a row of A^T)

$\Rightarrow A^T(\mathbf{b} - A\widehat{\mathbf{x}}) = 0$

$\Rightarrow A^T A\widehat{\mathbf{x}} = A^T\mathbf{b}$ □

Remark 8.13. Theorem 8.12 implies that LS solutions of $A\mathbf{x} = \mathbf{b}$ are solutions of the normal equations $A^T A\widehat{\mathbf{x}} = A^T\mathbf{b}$.

- 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 8.14. (Method of normal equations) *Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$. The following statements are logically equivalent:*

a. *The equation $A\mathrm{x} = \mathrm{b}$ has a unique LS solution for each $\mathrm{b} \in \mathbb{R}^m$.*

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 $\widehat{\mathrm{x}}$ is given by*

$$\widehat{\mathbf{x}} = (A^T A)^{-1} A^T\mathbf{b}. \tag{8.14}$$

Example 8.15. Describe all least squares 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.

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

$$\nabla_{\mathbf{x}}\mathcal{J}(\widehat{\mathbf{x}}) = \frac{\partial \mathcal{J}(\mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\widehat{\mathbf{x}}} = \mathbf{0}. \tag{8.15}$$

- Let's compute the gradient of $\mathcal{J}$.

$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{x})}{\partial \mathbf{x}} &= \frac{\partial\big((A\mathbf{x} - \mathbf{b})^T(A\mathbf{x} - \mathbf{b})\big)}{\partial \mathbf{x}} \\ &= \frac{\partial(\mathbf{x}^T A^T A\mathbf{x} - 2\mathbf{x}^T A^T\mathbf{b} + \mathbf{b}^T\mathbf{b})}{\partial \mathbf{x}} \\ &= 2A^T A\mathbf{x} - 2A^T\mathbf{b}. \end{aligned} \tag{8.16}$$

- By setting the last term to zero, we obtain normal equations.

8.3. Regression Analysis

Definition 8.16. Regression analysis is a set of statistical methods used for the estimation of relationships between a dependent variable and one or more independent variables.

8.3.1. Regression line

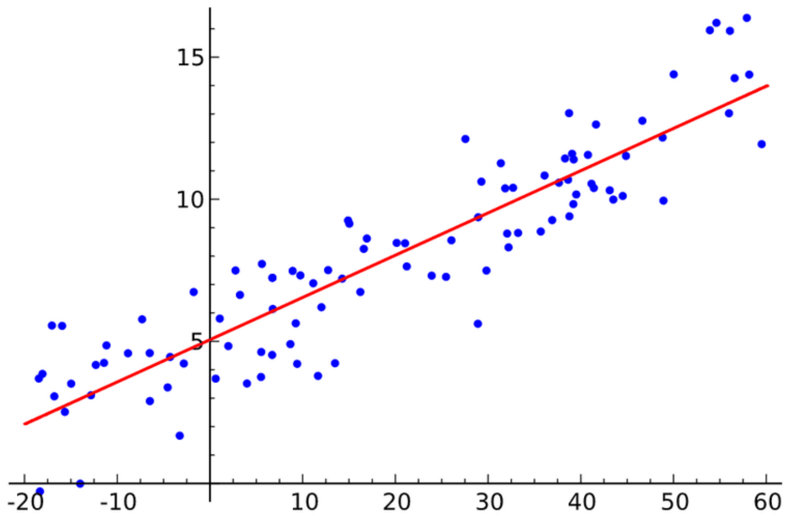


Figure 8.4: A regression line.

Definition 8.17. 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

$$y = \beta_0 + \beta_1 x \tag{8.17}$$

that is as close as possible to the given points. This line is called the **least-squares line**; it is also called the **regression line** of y on x and 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

$$\begin{array}{ccc} \boxed{\text{Predicted } y\text{-value}} & & \boxed{\text{Observed } y\text{-value}} \\ \hline \beta_0 + \beta_1 x_1 & = & y_1 \\ \beta_0 + \beta_1 x_2 & = & y_2 \\ \vdots & & \vdots \\ \beta_0 + \beta_1 x_m & = & y_m \\ \hline \end{array} \tag{8.18}$$

- It can be equivalently written as

$$X\boldsymbol{\beta} = \mathbf{y}, \tag{8.19}$$

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 y the **observation vector**.

- Thus the LS solution can be determined by solving the normal equations:

$$X^T X\boldsymbol{\beta} = X^T\mathbf{y}, \tag{8.20}$$

provided that $X^T X$ is invertible.

- The normal equations for the regression line read

$$\begin{bmatrix} m & \Sigma x_i \\ \Sigma x_i & \Sigma x_i^2 \end{bmatrix} \boldsymbol{\beta} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \end{bmatrix}. \tag{8.21}$$

Remark 8.18. (Pointwise construction of the normal equations) The normal equations for the regression line in (8.21) can be rewritten as

$$\sum_{i=1}^{m} \begin{bmatrix} 1 & x_i \\ x_i & x_i^2 \end{bmatrix} \boldsymbol{\beta} = \sum_{i=1}^{m} \begin{bmatrix} y_i \\ x_i y_i \end{bmatrix}. \tag{8.22}$$

- 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 8.19. 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.

8.3.2. Least-squares fitting of other curves

Remark 8.20. 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

$$\begin{array}{ccc} \boxed{\text{Predicted } y\text{-value}} & & \boxed{\text{Observed } y\text{-value}} \\ \hline \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 & = & y_1 \\ \beta_0 + \beta_1 x_2 + \beta_2 x_2^2 & = & y_2 \\ \vdots & & \vdots \\ \beta_0 + \beta_1 x_m + \beta_2 x_m^2 & = & y_m \\ \hline \end{array} \tag{8.23}$$

It can be equivalently written as

$$X\boldsymbol{\beta} = \mathbf{y}, \tag{8.24}$$

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*:

$$X^T X\boldsymbol{\beta} = \begin{bmatrix} \Sigma 1 & \Sigma x_i & \Sigma x_i^2 \\ \Sigma x_i & \Sigma x_i^2 & \Sigma x_i^3 \\ \Sigma x_i^2 & \Sigma x_i^3 & \Sigma x_i^4 \end{bmatrix} \boldsymbol{\beta} = \begin{bmatrix} \Sigma y_i \\ \Sigma x_i y_i \\ \Sigma x_i^2 y_i \end{bmatrix} = X^T\mathbf{y} \tag{8.25}$$

Self-study 8.21. 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: $y = 1 + 0.5x^2$

8.3.3. Nonlinear regression: Linearization

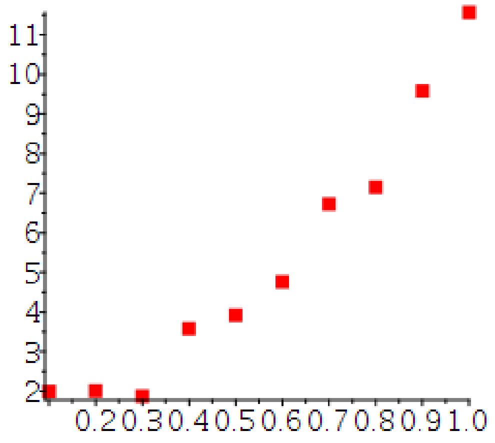
Strategy 8.22. For nonlinear models, a **change of variables** can be applied to get a linear model.

$$\begin{array}{l|lcl}
\text{Model} & \text{Change of Variables} & & \text{Linearization} \\
\hline
y = A + \dfrac{B}{x} & \widetilde{x} = \dfrac{1}{x},\ \ \widetilde{y} = y & \Rightarrow & \widetilde{y} = A + B\widetilde{x} \\
y = \dfrac{1}{A + Bx} & \widetilde{x} = x,\ \ \widetilde{y} = \dfrac{1}{y} & \Rightarrow & \widetilde{y} = A + B\widetilde{x} \\
y = Ce^{Dx} & \widetilde{x} = x,\ \ \widetilde{y} = \ln y & \Rightarrow & \widetilde{y} = \ln C + D\widetilde{x} \\
y = \dfrac{1}{A + B\ln x} & \widetilde{x} = \ln x,\ \ \widetilde{y} = \dfrac{1}{y} & \Rightarrow & \widetilde{y} = A + B\widetilde{x}
\end{array} \tag{8.26}$$

The above table contains just a few examples of linearization; for other nonlinear models, use your imagination and creativity.

Example 8.23. 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

$$\ln y = \ln c + dx. \tag{8.27}$$

Using the change of variables

$$Y = \ln y, \quad a_0 = \ln c, \quad a_1 = d, \quad X = x,$$

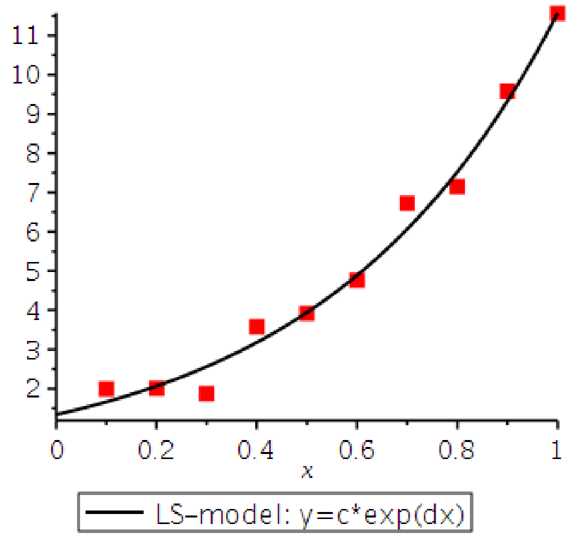
the equation (8.27) reads

$$Y = a_0 + a_1 X, \tag{8.28}$$

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)
```



8.4. Scene Analysis with Noisy Data: RANSAC

Note: **Scene analysis** is concerned with the interpretation of acquired data 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**).

8.4.1. Weighted least-squares

Definition 8.24. 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

$$W = \text{diag}(w_1, w_2, \cdots, w_m). \tag{8.29}$$

The **weight matrix** W can be decided either *manually* or *automatically*.

Algorithm 8.25. (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 (8.19):

$$X\boldsymbol{\beta} = \mathbf{y}. \tag{8.30}$$

- When a weight matrix is applied, the above system can be written as

$$WX\boldsymbol{\beta} = W\mathbf{y}. \tag{8.31}$$

- Thus its **weighted normal equations** read

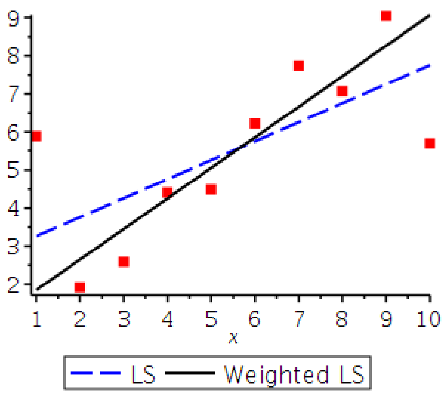
$$X^T W X\boldsymbol{\beta} = X^T W\mathbf{y}. \tag{8.32}$$

Example 8.26. 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
```



8.4.2. 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 8.27. (RANSAC) (Fischler-Bolles, 1981) [6]

Input: Measurement set $\boldsymbol{X} = \{\mathrm{x}_i\}$, 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 $\mathrm{p} = g(S)$.
3. Compute the hypothesis consensus set, fitting within the error tolerance τ_e:
   $\mathcal{C} = \mathrm{inlier}(\boldsymbol{X}, \mathbf{p}, \tau_\mathrm{e})$
4. If $|\mathcal{C}| \ge \eta|\boldsymbol{X}|$, then re-estimate a hypothesis $\mathrm{p} = g(\mathcal{C})$ and stop.
5. Otherwise, repeat steps 1–4 (maximum of N times).

Example 8.28. 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: $y = a + bx \quad (\Rightarrow a + bx - y = 0)$

$$y = b(x - x_1) + y_1 = a + bx \quad \Leftarrow \quad b = \frac{y_2 - y_1}{x_2 - x_1}, \quad a = y_1 - bx_1.$$

3. **Consensus set** $\mathcal{C}$:

$$\mathcal{C} = \left\{ (x_i, y_i) \in \boldsymbol{X} \mid d = \frac{|a + bx_i - y_i|}{\sqrt{b^2 + 1}} \le \tau_e \right\} \tag{8.33}$$

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,

$$\text{ID}(i) = \begin{cases} 1 & \text{if } \mathbf{x}_i \in \mathcal{C} \\ 0 & \text{if } \mathbf{x}_i \notin \mathcal{C} \end{cases} \tag{8.34}$$

Remark 8.29. 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 $\mathbf{x}_i$,

$$\textit{dist}(f(\mathbf{p}), \mathbf{x}_i) \begin{cases} \le \tau_e, & \text{then} \quad w_i = 1 \\ > \tau_e, & \text{then} \quad w_i = 0 \end{cases} \tag{8.35}$$

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 8.30.

- 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.

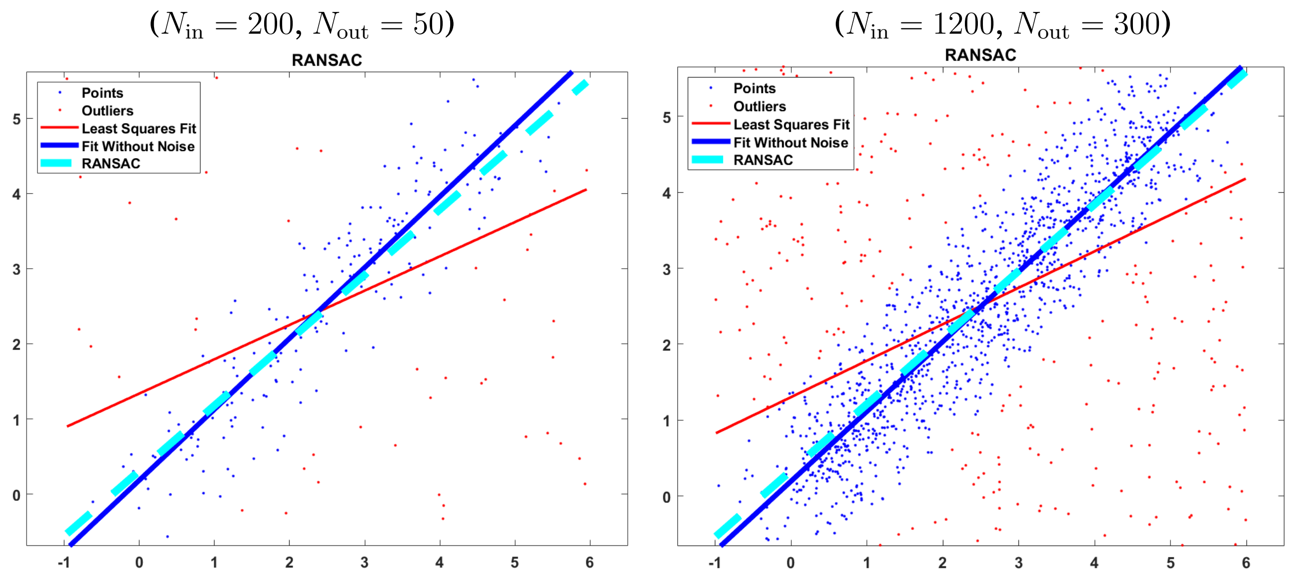


Figure 8.5: The RANSAC for linear-type synthetic datasets.

Table 8.1: 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) [24]
- Progressive Sample Consensus (PROSAC) [2]
- Recursive RANSAC (R-RANSAC) [17]

8.5. 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 8.31. A **rational function** r of degree N has the form

$$r(x) = \frac{p(x)}{q(x)}, \tag{8.36}$$

where $p(x)$ and $q(x)$ are polynomials whose degrees sum to N.

Padé Approximation:

Definition 8.32. **Padé rational function** of degree $N = n + m$, approximating $f(x)$, has the form

$$f(x) \approx R_{n,m}(x) = \frac{p(x)}{q(x)} = \frac{p_0 + p_1 x + \cdots + p_n x^n}{1 + q_1 x + \cdots + q_m x^m} \tag{8.37}$$

Determination of the constants $\{p_0, p_1, \cdots, p_n,\ q_1, \cdots, q_m\}$

We will find them to satisfy

$$\left(\frac{d^k f}{dx^k} - \frac{d^k R_{n,m}}{dx^k}\right)(0) = 0, \qquad \text{for } k = 0, 1, \cdots, N. \tag{8.38}$$

- Let f have the **Maclaurin series expansion**

$$f(x) = \sum_{i=0}^{\infty} a_i x^i. \tag{8.39}$$

- Then, for $q_0 = 1$,

$$f(x) - R_{n,m}(x) = \frac{\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{m} q_i x^i\right) - \left(\sum_{i=0}^{n} p_i x^i\right)}{q(x)}. \tag{8.40}$$

- Our objective is to determine the constants $\{p_0, p_1, \cdots, p_n,\ q_1, \cdots, q_m\}$ such that (8.38) 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 (8.40),

$$\left(\sum_{i=0}^{\infty} a_i x^i\right)\left(\sum_{i=0}^{m} q_i x^i\right) - \left(\sum_{i=0}^{n} p_i x^i\right), \tag{8.41}$$

  has no terms of degree $\le N$.
- In order to simplify the notation, we define

$$\begin{aligned} q_{m+1} &= q_{m+2} = \cdots = q_N = 0, \\ p_{n+1} &= p_{n+2} = \cdots = p_N = 0. \end{aligned}$$

- Then the coefficient of x^k in (8.41) can be expresses as

$$\sum_{i=0}^{k} a_i q_{k-i} - p_k, \quad k = 0, 1, \cdots, N. \tag{8.42}$$

Summary 8.33. (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

$$\sum_{i=0}^{k} a_i q_{k-i} = p_k, \quad k = 0, 1, \cdots, N. \tag{8.43}$$

Example 8.34. Find the Padé approximation to e^{-x} of degree 5 with $n = 3$ and $m = 2$.

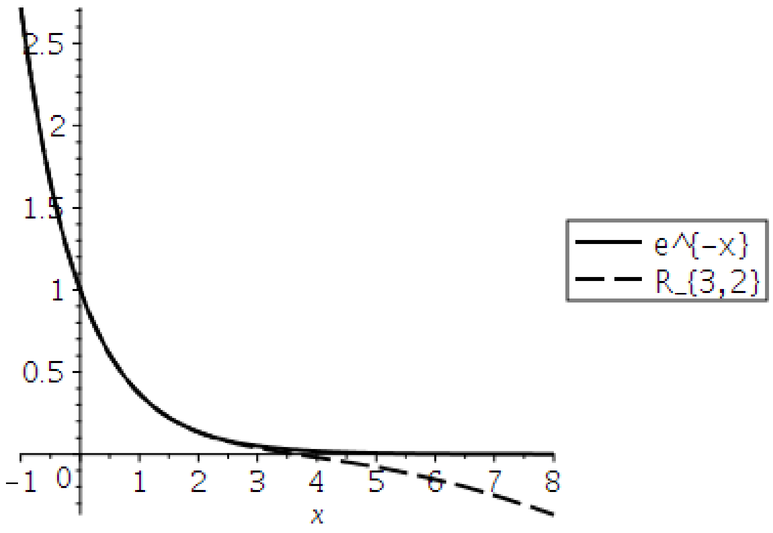
Solution.

```
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
```

Exercises for Chapter 8

8.1. 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

$$\text{proj}_w\mathbf{y} = UU^T\mathbf{y}, \quad \text{for all } \mathbf{y} \in \mathbb{R}^n. \tag{8.44}$$

(Thus the orthogonal projection can be viewed as a *matrix transformation*.)

8.2. $\boxed{\text{C}}$ 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: $\mathbf{v}_4 = (0, 5, 0, 0, -5)$

8.3. $\boxed{\text{C}}$ 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.

8.4. $\boxed{\text{C}}$ Implement a code for the RANSAC, Algorithm 8.27, and use the data in Example 8.26, p.302, to find a best-fitting regression line. Set $\tau_e = 1$ and $\eta|\boldsymbol{X}| = 8$.

8.5. $\boxed{\text{C}}$ Determine Pad'e 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

$$\texttt{convert(taylor(sin(x), x = 0, 7), polynom)} = x - \frac{1}{6}x^3 + \frac{1}{120}x^5$$

(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.)

CHAPTER 9

Eigenvalues and Matrix Decomposition

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
Singular Value Decomposition (SVD)	LS problems, pseudoinverse
Principal Component Analysis (PCA)	Feature extraction, data compression

Contents of Chapter 9

9.1. 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 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:

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|. \tag{9.1}$$

The power method approximates the largest eigenvalue λ_1 and its associated eigenvector $\mathbf{v}_1$.

9.1.1. 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

$$\mathbf{x} = \sum_{j=1}^{n} \beta_j \mathbf{v}_j, \tag{9.2}$$

  for some constants $\{\beta_1, \beta_2, \cdots, \beta_n\}$.

- Multiplying both sides of (9.2) by A and A^2 gives

$$\begin{aligned} A\mathbf{x} &= A\Big(\sum_{j=1}^{n} \beta_j \mathbf{v}_j\Big) = \sum_{j=1}^{n} \beta_j A\mathbf{v}_j = \sum_{j=1}^{n} \beta_j \lambda_j \mathbf{v}_j, \\ A^2\mathbf{x} &= A\Big(\sum_{j=1}^{n} \beta_j \lambda_j \mathbf{v}_j\Big) = \sum_{j=1}^{n} \beta_j \lambda_j^2 \mathbf{v}_j. \end{aligned} \tag{9.3}$$

- In general,

$$A^k\mathbf{x} = \sum_{j=1}^{n} \beta_j \lambda_j^k \mathbf{v}_j, \quad k = 1, 2, \cdots, \tag{9.4}$$

  which gives

$$A^k\mathbf{x} = \lambda_1^k \sum_{j=1}^{n} \beta_j \left(\frac{\lambda_j}{\lambda_1}\right)^k \mathbf{v}_j, \quad k = 1, 2, \cdots, \tag{9.5}$$

- Since $|\lambda_1| > |\lambda_j|$, $2 \le j \le n$, we have $\lim_{k\to\infty} |\lambda_j/\lambda_1|^k = 0$, and

$$\lim_{k\to\infty} A^k\mathbf{x} = \lim_{k\to\infty} \lambda_1^k \beta_1 \mathbf{v}_1. \tag{9.6}$$

Remark 9.1. The sequence in (9.6) converges to 0 if $|\lambda_1| < 1$ and diverges if $|\lambda_1| > 1$, provided that $\beta_1 \neq 0$.

- The entries of A^kx 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 v$_1$.
- To take care of that possibility, we scale A^kx in an appropriate manner to ensure that the limit in (9.6) is finite and nonzero.

Algorithm 9.2. (The Power Iteration) Given x $\neq 0$:

```
initialization :  $\mathbf{x}^0 = \mathbf{x}/||\mathbf{x}||_\infty$
for  $k = 1, 2, \cdots$
    $\mathbf{y}^k = A\mathbf{x}^{k-1}$;  $\mu_k = ||\mathbf{y}^k||_\infty$          (9.7)
    $\mathbf{x}^k = \mathbf{y}^k/\mu_k$
end for
```

Properties

- First of all,
$$||\mathbf{x}^k||_\infty = 1, \text{ for all } k \ge 0.$$
- It follows from (9.6) that for k sufficiently large,
$$\begin{aligned} \mathbf{x}^k &= \Big(\prod_{j=1}^{k}\mu_j\Big)^{-1} A^k\mathbf{x}^0 &\approx \Big(\prod_{j=1}^{k}\mu_j\Big)^{-1} \lambda_1^k\beta_1\mathbf{v}_1, \\ \mathbf{x}^{k+1} &= \Big(\prod_{j=1}^{k+1}\mu_j\Big)^{-1} A^{k+1}\mathbf{x}^0 &\approx \Big(\prod_{j=1}^{k+1}\mu_j\Big)^{-1} \lambda_1^{k+1}\beta_1\mathbf{v}_1. \end{aligned} \tag{9.8}$$
- Note that $||\mathbf{x}^k||_\infty = ||\mathbf{x}^{k+1}||_\infty = 1$. Comparison of the above two equations reads
$$\Big|\frac{\lambda_1}{\mu_{k+1}}\Big| \approx 1. \tag{9.9}$$

Claim 9.3. Let $\{\mathbf{x}^k, \mu_k\}$ be sequences produced by the power method. Then,
$$\mathbf{x}^k \to \mathbf{v}_1,\ \ \mu_k \to |\lambda_1|, \quad \text{as } k \to \infty. \tag{9.10}$$
More precisely, the power method converges as
$$\mu_k = |\lambda_1| + \mathcal{O}(|\lambda_2/\lambda_1|^k). \tag{9.11}$$

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
$$A = PDP^{-1} \tag{9.12}$$
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 9.4. 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 $|6 - \mu_{k+1}| \approx \dfrac{1}{2}|6 - \mu_k|$, for which $|\lambda_2/\lambda_1| = \dfrac{1}{2}$.

9.1.2. Symmetric Power Method

Theorem 9.5. (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 necessarily 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*

$$A = UDU^T. \tag{9.13}$$

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 9.6. (Symmetric Power Method) Suppose $A \in \mathbb{R}^{n\times n}$ is symmetric. Given $\mathbf{x} \neq 0$:

```
initialization :  $\mathbf{x}^0 = \mathbf{x}/||\mathbf{x}||_2$
for  $k = 1, 2, \cdots$
    $\mathbf{y}^k = A\mathbf{x}^{k-1}$;  $\mu_k = \mathbf{y}^k \bullet \mathbf{x}^{k-1}$        (9.14)
    $\mathbf{x}^k = \mathbf{y}^k/||\mathbf{y}^k||_2$
end for
```

Claim 9.7. The symmetric power method converges as

$$\mu_k = \lambda_1 + \mathcal{O}(|\lambda_2/\lambda_1|^{2k}), \tag{9.15}$$

which converges twice faster than the **basic power method**.

Example 9.8. The matrix A is as in Example 9.4: $A = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$.
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 := $\frac{1}{ynorm}$ 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 $|6-\mu_{k+1}| \approx \dfrac{1}{4}|6-\mu_k|$, for which $|\lambda_2/\lambda_1| = \dfrac{1}{2}$.

Accuracy Analysis

Let μ be a real number that approximates an eigenvalue of a symmetric matrix A and x an associated eigenvector. Then Ax $-$ μx 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 9.9. *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 $\mu \in \mathbb{R}$,*

$$\min_{1\le i\le n} |\lambda_i - \mu| \le ||A\mathbf{x} - \mu\mathbf{x}||_2. \tag{9.16}$$

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,

$$\begin{aligned} ||A\mathbf{x} - \mu\mathbf{x}||_2^2 &= \Big\| \sum_{i=1}^{n} c_i (A\mathbf{v}_i - \mu\mathbf{v}_i) \Big\|_2^2 = \Big\| \sum_{i=1}^{n} c_i (\lambda_i - \mu)\mathbf{v}_i \Big\|_2^2 \\ &= \sum_{i=1}^{n} c_i^2 (\lambda_i - \mu)^2 \\ &\ge \min_{1\le i\le n} |\lambda_i - \mu|^2 \sum_{i=1}^{n} c_i^2 = \min_{1\le i\le n} |\lambda_i - \mu|^2, \end{aligned} \tag{9.17}$$

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.

9.1.3. Inverse Power Method

Some applications require to find an eigenvalue of 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

$$A\mathbf{v}_i = \lambda_i \mathbf{v}_i, \quad i = 1, 2, \cdots, n. \tag{9.18}$$

- Then it is easy to see that

$$(A-qI)\mathbf{v}_i = (\lambda_i - q)\mathbf{v}_i. \tag{9.19}$$

  Thus, we obtain

$$(A-qI)^{-1}\mathbf{v}_i = \frac{1}{\lambda_i - q}\mathbf{v}_i. \tag{9.20}$$

- That is, when $q \notin \{\lambda_1, \lambda_2, \cdots, \lambda_n\}$, the eigenvalues of $(A-qI)^{-1}$ are

$$\frac{1}{\lambda_1 - q}, \frac{1}{\lambda_2 - q}, \cdots, \frac{1}{\lambda_n - q}, \tag{9.21}$$

  with the same eigenvectors $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\}$ of A.

Algorithm 9.10. **(Inverse Power Method)** Applying the power method to $(A-qI)^{-1}$ gives the **inverse power method**. Given x $\neq 0$:

```
set : $\mathbf{x}^0 = \mathbf{x}$
for $k = 1, 2, \cdots$
    $\mathbf{y}^k = (A-qI)^{-1}\mathbf{x}^{k-1}$;  $\mu_k = ||\mathbf{y}^k||_\infty$
    $\mathbf{x}^k = \mathbf{y}^k/\mu_k$                                   (9.22)
    $\lambda = 1/\mu_k + q$
end for
```

Example 9.11. The matrix A is as in Example 9.4: $A = \begin{bmatrix} 4 & -1 & 1 \\ -1 & 3 & -2 \\ 1 & -2 & 3 \end{bmatrix}$. Find the the eigenvalue of A nearest to $q = 5/2$, using the inverse power method.

Solution.

```
x := ⟨1, 0, 0⟩ :
q := $\frac{5}{2}$ :
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 := $\frac{1}{ynorm}$ y;
    muk := $\frac{1}{ynorm}$ + q;
    print( `k= `, k, `muk= `, $evalf_8$(muk), $evalf_4$($x^{\%T}$), $evalf_8$(abs(3 − muk)) );
end do:
```

Inverse Power Method

```
k=, 1, muk=, 3.2000000, [1., 0.4000, -0.4000], 0.20000000
k=, 2, muk=, 3.0303030, [1., 0.4848, -0.4848], 0.030303030
k=, 3, muk=, 3.0043668, [1., 0.4978, -0.4978], 0.0043668122
k=, 4, muk=, 3.0006246, [1., 0.4997, -0.4997], 0.00062460962
k=, 5, muk=, 3.0000892, [1., 0.5000, -0.5000], 0.000089245872
k=, 6, muk=, 3.0000127, [1., 0.5000, -0.5000], 0.000012749735
k=, 7, muk=, 3.0000018, [1., 0.5000, -0.5000], 0.0000018213974
k=, 8, muk=, 3.0000003, [1., 0.5000, -0.5000], 2.6019977 10^-7
k=, 9, muk=, 3.0000000, [1., 0.5000, -0.5000], 3.7171398 10^-8
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**.

9.2. QR Factorization

Note: Power methods are not in general suitable for calculating all the eigenvalues of a matrix because of the growth of round-off error. In this section, we will consider the so-called **QR factorization**. The method 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 9.12. (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 9.13. *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 9.14. *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

$$\left\{ \begin{array}{l} \textbf{Gram-Schmidt process} \\ \textbf{Householder reflectors} \end{array} \right.$$

The Gram-Schmidt process is considered earlier in Section 8.1.2, p. 288.

9.2.1. QR factorization by the Gram-Schmidt process

Theorem 9.15. (The Gram-Schmidt process – Revisit of Theorem 8.5) *Given a basis* $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_p\}$ *for a nonzero subspace* W *of* $\mathbb{R}^n$, *define*

$$\begin{aligned}
\mathbf{v}_1 &= \mathbf{x}_1\\
\mathbf{v}_2 &= \mathbf{x}_2 - \frac{\mathbf{x_2}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1\\
\mathbf{v}_3 &= \mathbf{x}_3 - \frac{\mathbf{x_3}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 - \frac{\mathbf{x_3}\bullet\mathbf{v_2}}{\mathbf{v_2}\bullet\mathbf{v_2}}\mathbf{v}_2\\
&\ \vdots\\
\mathbf{v}_p &= \mathbf{x}_p - \frac{\mathbf{x_p}\bullet\mathbf{v_1}}{\mathbf{v_1}\bullet\mathbf{v_1}}\mathbf{v}_1 - \frac{\mathbf{x_p}\bullet\mathbf{v_2}}{\mathbf{v_2}\bullet\mathbf{v_2}}\mathbf{v}_2 - \cdots - \frac{\mathbf{x_p}\bullet\mathbf{v}_{p-1}}{\mathbf{v}_{p-1}\bullet\mathbf{v}_{p-1}}\mathbf{v}_{p-1}
\end{aligned} \tag{9.23}$$

Then $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_p\}$ *is an* ***orthogonal basis*** *for* W. *In addition,*

$$\mathit{Span}\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_k\} = \mathit{Span}\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\}, \quad \text{for } 1 \le k \le p. \tag{9.24}$$

Algorithm 9.16. (QR Factorization by the 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*

$$\begin{aligned}
\mathbf{x}_1 &= (\mathbf{u}_1\bullet\mathbf{x}_1)\mathbf{u}_1\\
\mathbf{x}_2 &= (\mathbf{u}_1\bullet\mathbf{x}_2)\mathbf{u}_1 + (\mathbf{u}_2\bullet\mathbf{x}_2)\mathbf{u}_2\\
\mathbf{x}_3 &= (\mathbf{u}_1\bullet\mathbf{x}_3)\mathbf{u}_1 + (\mathbf{u}_2\bullet\mathbf{x}_3)\mathbf{u}_2 + (\mathbf{u}_3\bullet\mathbf{x}_3)\mathbf{u}_3\\
&\ \vdots\\
\mathbf{x}_n &= \textstyle\sum_{j=1}^{n}(\mathbf{u}_j\bullet\mathbf{x}_n)\mathbf{u}_j,
\end{aligned} \tag{9.25}$$

which is a rephrase (with normalization) of (9.23). Thus

$$A = [\mathbf{x}_1\ \mathbf{x}_2\ \cdots\ \mathbf{x}_n] = QR \tag{9.26}$$

implies that

$$\begin{aligned}
Q &= [\mathbf{u}_1\ \mathbf{u}_2\ \cdots\ \mathbf{u}_n],\\
R &= \begin{bmatrix} \mathbf{u}_1\bullet\mathbf{x}_1 & \mathbf{u}_1\bullet\mathbf{x}_2 & \mathbf{u}_1\bullet\mathbf{x}_3 & \cdots & \mathbf{u}_1\bullet\mathbf{x}_n\\ 0 & \mathbf{u}_2\bullet\mathbf{x}_2 & \mathbf{u}_2\bullet\mathbf{x}_3 & \cdots & \mathbf{u}_2\bullet\mathbf{x}_n\\ 0 & 0 & \mathbf{u}_3\bullet\mathbf{x}_3 & \cdots & \mathbf{u}_3\bullet\mathbf{x}_n\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & 0 & \cdots & \mathbf{u}_n\bullet\mathbf{x}_n \end{bmatrix} = Q^T A.
\end{aligned} \tag{9.27}$$

In practice, the coefficients $r_{ij} = \mathbf{u}_i\bullet\mathbf{x}_j$, $i < j$, *can be saved during the Gram-Schmidt process.*

Example 9.17. 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

$$\begin{cases} \mathbf{u}_1 &= \mathbf{v}_1/||\mathbf{v}_1|| = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} \\ \mathbf{u}_2 &= \mathbf{v}_2/||\mathbf{v}_2|| = \begin{bmatrix} -0.6 \\ 0.8 \end{bmatrix} \end{cases} \tag{9.28}$$

Thus,

$$Q = [\mathbf{u}_1,\, \mathbf{u}_2] = \begin{bmatrix} 0.8 & -0.6 \\ 0.6 & 0.8 \end{bmatrix} \tag{9.29}$$

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 9.18. 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:
```

$$Q := \langle u_1 | u_2 | u_3 \rangle = \begin{bmatrix} \frac{3\sqrt{55}}{55} & -\frac{5\sqrt{143}}{143} & -\frac{23\sqrt{4030}}{12090} \\ \frac{\sqrt{55}}{55} & \frac{54\sqrt{143}}{715} & \frac{\sqrt{4030}}{2015} \\ \frac{6\sqrt{55}}{55} & \frac{\sqrt{143}}{143} & -\frac{38\sqrt{4030}}{6045} \\ \frac{3\sqrt{55}}{55} & -\frac{3\sqrt{143}}{715} & \frac{173\sqrt{4030}}{12090} \end{bmatrix}$$

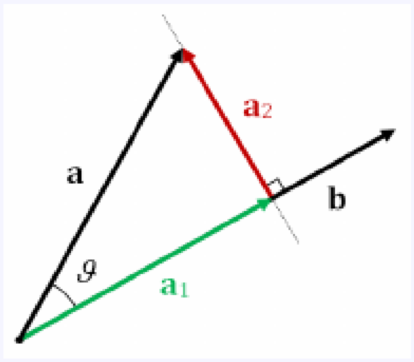
$$R = \begin{bmatrix} \sqrt{55} & \frac{12\sqrt{55}}{11} & \frac{82\sqrt{55}}{55} \\ 0 & \frac{5\sqrt{143}}{11} & \frac{32\sqrt{143}}{143} \\ 0 & 0 & \frac{3\sqrt{4030}}{65} \end{bmatrix}$$

9.2.2. Householder reflectors

Let's revisit the orthogonal projection, Definition 8.1:

Definition 9.19. 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

$$\text{proj}_{\mathbf{b}}\mathbf{a} = \frac{\mathbf{a}\bullet\mathbf{b}}{\mathbf{b}\bullet\mathbf{b}}\mathbf{b}. \tag{9.30}$$



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 $\mathbf{a}$.

Householder reflection (or, Householder transformation, Elementary reflector) is a transformation that takes a vector and reflects it about some plane or 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

$$\mathbf{x} - 2\,\text{proj}_{\mathbf{u}}\mathbf{x} = \mathbf{x} - 2(\mathbf{x}\bullet\mathbf{u})\mathbf{u}. \tag{9.31}$$

  (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

$$\mathbf{x} - 2(\mathbf{x}\bullet\mathbf{u})\mathbf{u} = \mathbf{x} - 2\mathbf{u}(\mathbf{u}^T\mathbf{x}) = (I - 2\mathbf{u}\mathbf{u}^T)\mathbf{x}. \tag{9.32}$$

Definition 9.20. The matrix

$$Q := I - 2\mathbf{u}\mathbf{u}^T \tag{9.33}$$

is called a **Householder matrix** or a **Householder reflector**.

Theorem 9.21. (Theorems on Reflectors)

I. Let $\mathbf{u} \in \mathbb{R}^n$, with $||\mathbf{u}|| = 1$, and define $P \in \mathbb{R}^{n\times n}$ by $P = \mathbf{u}\mathbf{u}^T$. 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 $Q = I - 2\mathbf{u}\mathbf{u}^T$. 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 9.24 below.)

Corollary 9.22. *Let $\mathbf{u} \in \mathbb{R}^n$ be a nonzero vector. Define $Q \in \mathbb{R}^{n\times n}$ by*

$$Q = I - \gamma\,\mathbf{u}\mathbf{u}^T, \quad \gamma = 2/||\mathbf{u}||^2. \tag{9.34}$$

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 9.21.II.

Corollary 9.23. *Let* $\mathbf{x} \in \mathbb{R}^n$ *be a nonzero vector. Then there is a reflector* Q *such that*

$$Q\mathbf{x} = Q\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{9.35}$$

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 9.21.III. □

Algorithm 9.24. (Construction of the Reflector)

$$Q : \mathbf{x} \mapsto \mathbf{y} = [-\tau, 0, \cdots, 0]^T, \quad \text{where } \tau = \pm||\mathbf{x}||. \tag{9.36}$$

- Let
$$\mathbf{u} = \mathbf{x} - \mathbf{y} = [x_1 + \tau, x_2, \cdots, x_n]^T, \tag{9.37}$$
where the sign of τ is such that $x_1 + \tau \neq 0$.
- Define
$$Q = I - \gamma\,\mathbf{u}\mathbf{u}^T, \quad \gamma = \frac{2}{||\mathbf{u}||^2}. \tag{9.38}$$

Check if $Q\mathbf{x} = \mathbf{y}$

- Rewrite $\mathbf{x}$ as
$$\mathbf{x} = \frac{1}{2}(\mathbf{x}-\mathbf{y}) + \frac{1}{2}(\mathbf{x}+\mathbf{y}). \tag{9.39}$$
- By Corollary 9.22 (a),
$$Q(\mathbf{x}-\mathbf{y}) = Q\mathbf{u} = -\mathbf{u} = -\mathbf{x}+\mathbf{y}. \tag{9.40}$$
- Since $(\mathbf{x}-\mathbf{y})\bullet(\mathbf{x}+\mathbf{y}) = ||\mathbf{x}||^2 - ||\mathbf{y}||^2 = 0$, it follows from Corollary 9.22 (b) that
$$Q(\mathbf{x}+\mathbf{y}) = \mathbf{x}+\mathbf{y}. \tag{9.41}$$
- Thus, $Q\mathbf{x} = \mathbf{y}$ holds. □

Example 9.25. 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 y, Q is symmetric, orthogonal, and an involution matrix. In addition, it is **unique** for each choice of y.

9.2.3. QR factorization by Householder reflectors

We will construct a QR factorization algorithm using Householder reflectors for $A \in \mathbb{R}^{n\times n}$, for simplicity. Consider Corollary 9.13, p. 322:

Corollary 9.26. (Revisit of Corollary 9.13) *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 induction on n.

- When $n = 1$. Then $A = [a_{11}]$.
  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, $\mathbf{a}_1$:

$$Q_1\mathbf{a}_1 = Q_1\begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{n1}\end{bmatrix} = \begin{bmatrix} -\tau_1 \\ 0 \\ \vdots \\ 0\end{bmatrix}, \quad \text{where } \tau_1 = \pm||\mathbf{a}_1||. \tag{9.42}$$

  (c) Thus,

$$Q_1^T A = Q_1 A = \begin{bmatrix} -\tau & \widehat{a_{12}} & \cdots & \widehat{a_{1n}} \\ 0 & & & \\ \vdots & & \widehat{A_2} & \\ 0 & & & \end{bmatrix} \tag{9.43}$$

  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

$$Q_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{Q_2} & \\ 0 & & & \end{bmatrix}. \tag{9.44}$$

(e) Then Q_2 is obviously orthogonal and, from (9.43) and (9.44), we have

$$Q_2^T Q_1^T A = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & \widehat{Q_2} & \\ 0 & & & \end{bmatrix} \begin{bmatrix} -\tau & \widehat{a_{12}} & \cdots & \widehat{a_{1n}} \\ 0 & & & \\ \vdots & & \widehat{A_2} & \\ 0 & & & \end{bmatrix} = \begin{bmatrix} -\tau & \widehat{a_{12}} & \cdots & \widehat{a_{1n}} \\ 0 & & & \\ \vdots & & \widehat{R_2} & \\ 0 & & & \end{bmatrix}. \tag{9.45}$$

(f) The last matrix is upper triangular; let us call it R. Let $Q = Q_1 Q_2$. Then $A = QR$, clearly.

It completes the proof. □

Remark 9.27.

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,

$$\mathbf{u} = \frac{\mathbf{x} - \mathbf{y}}{x_1 + \tau} = \begin{bmatrix} 1 \\ x_2/(x_1+\tau) \\ \vdots \\ x_n/(x_1+\tau) \end{bmatrix}. \tag{9.46}$$

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 $\tau^2 = ||\mathbf{x}||^2$,

$$||\mathbf{u}||^2 = \frac{2\tau^2 + 2\tau x_1}{(x_1+\tau)^2} = \frac{2\tau(\tau + x_1)}{(x_1+\tau)^2} = \frac{2\tau}{x_1+\tau}, \tag{9.47}$$

and therefore

$$\gamma = \frac{2}{||\mathbf{u}||^2} = \frac{x_1+\tau}{\tau}. \tag{9.48}$$

2. The the proof of Corollary 9.26 suggests an algorithm for the construction of Q and R.

$$Q_1 = I - \gamma_1 \mathbf{u}_1 \mathbf{u}_1^T, \quad \text{where } \mathbf{u}_1 = \begin{bmatrix} 1 \\ a_{21}/(a_{11}+\tau_1) \\ \vdots \\ a_{n1}/(a_{11}+\tau_1) \end{bmatrix}, \tag{9.49}$$

and

$$Q_2 = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & & & \\ \vdots & & I - \gamma_2 \mathbf{u}_2 \mathbf{u}_2^T & \\ 0 & & & \end{bmatrix}, \tag{9.50}$$

where $\mathbf{u}_2$ is determined from $\widehat{A_2}$. In general,

$$Q_k = \begin{bmatrix} I_{k-1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & & & \\ \vdots & & I - \gamma_k \mathbf{u}_k \mathbf{u}_k^T & \\ \mathbf{0} & & & \end{bmatrix}. \tag{9.51}$$

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 $Q_1, Q_2, \cdots, Q_n$ 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 $\mathcal{O}(n)$ flops.) Then, for each $\mathbf{a}_k$ (or columns of A), $k = 2, 3, \cdots, n$,

$$Q_1^T \mathbf{a}_k = Q_1 \mathbf{a}_k = (I - \gamma_1 \mathbf{u}_1 \mathbf{u}_1^T)\mathbf{a}_k = \mathbf{a}_k - \gamma_1 (\mathbf{u}_1^T \mathbf{a}_k)\mathbf{u}_1, \tag{9.52}$$

where the last term requires about $4n$ flops.

4. The flop count for $Q_1^T A$:

$$\text{cost}(Q_1^T A) \approx 4n^2. \tag{9.53}$$

Algorithm 9.28. (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 $A[k,k] = -\tau_k$                                   (9.54)
    $(d)$ 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
$$4n^2 + 4(n-1)^2 + \cdots \approx \mathcal{O}\Big(\frac{4}{3}n^3\Big), \tag{9.55}$$
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.

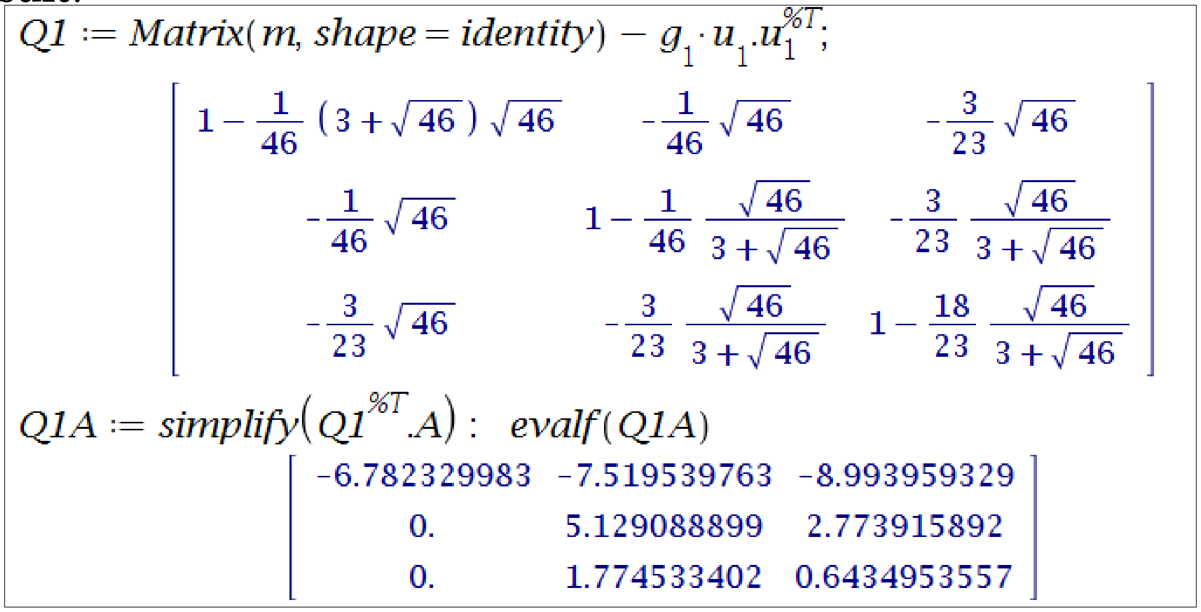
Example 9.29. Use reflectors to get QR decomposition of $A = \begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix}$.

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:



$Q1 := Matrix(m, shape = identity) - g_1 \cdot u_1 . u_1^{\%T};$

$$\begin{bmatrix} 1 - \frac{1}{46}\left(3+\sqrt{46}\right)\sqrt{46} & -\frac{1}{46}\sqrt{46} & -\frac{3}{23}\sqrt{46} \\ -\frac{1}{46}\sqrt{46} & 1 - \frac{1}{46}\,\frac{\sqrt{46}}{3+\sqrt{46}} & -\frac{3}{23}\,\frac{\sqrt{46}}{3+\sqrt{46}} \\ -\frac{3}{23}\sqrt{46} & -\frac{3}{23}\,\frac{\sqrt{46}}{3+\sqrt{46}} & 1 - \frac{18}{23}\,\frac{\sqrt{46}}{3+\sqrt{46}} \end{bmatrix}$$

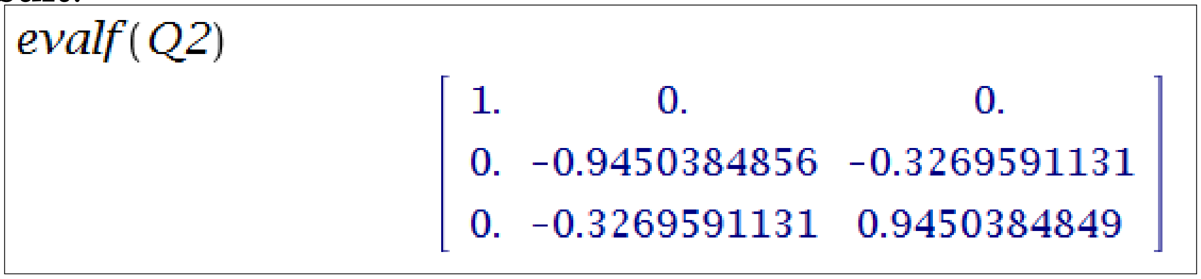
$Q1A := simplify\left(Q1^{\%T}.A\right):\ \ evalf(Q1A)$

$$\begin{bmatrix} -6.782329983 & -7.519539763 & -8.993959329 \\ 0. & 5.129088899 & 2.773915892 \\ 0. & 1.774533402 & 0.6434953557 \end{bmatrix}$$

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:
```

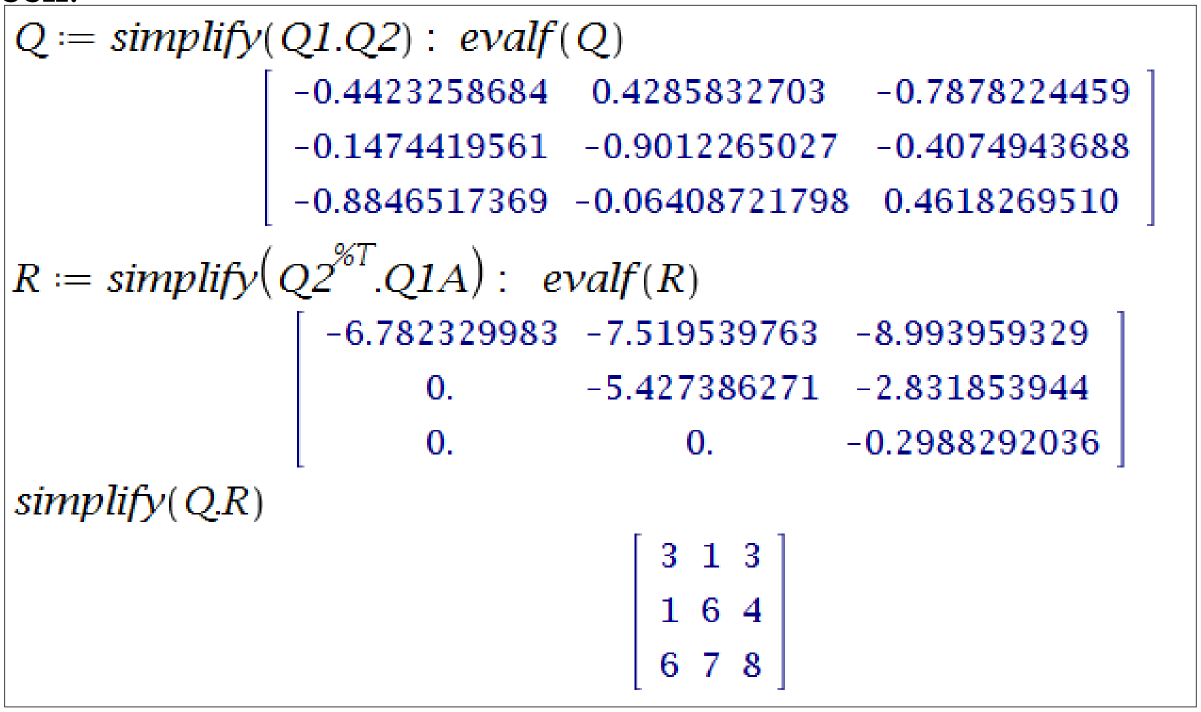
Result:



evalf(*Q2*)

$$\begin{bmatrix} 1. & 0. & 0. \\ 0. & -0.9450384856 & -0.3269591131 \\ 0. & -0.3269591131 & 0.9450384849 \end{bmatrix}$$

Check:



$$Q := simplify(Q1.Q2):\ \ evalf(Q)$$

$$\begin{bmatrix} -0.4423258684 & 0.4285832703 & -0.7878224459 \\ -0.1474419561 & -0.9012265027 & -0.4074943688 \\ -0.8846517369 & -0.06408721798 & 0.4618269510 \end{bmatrix}$$

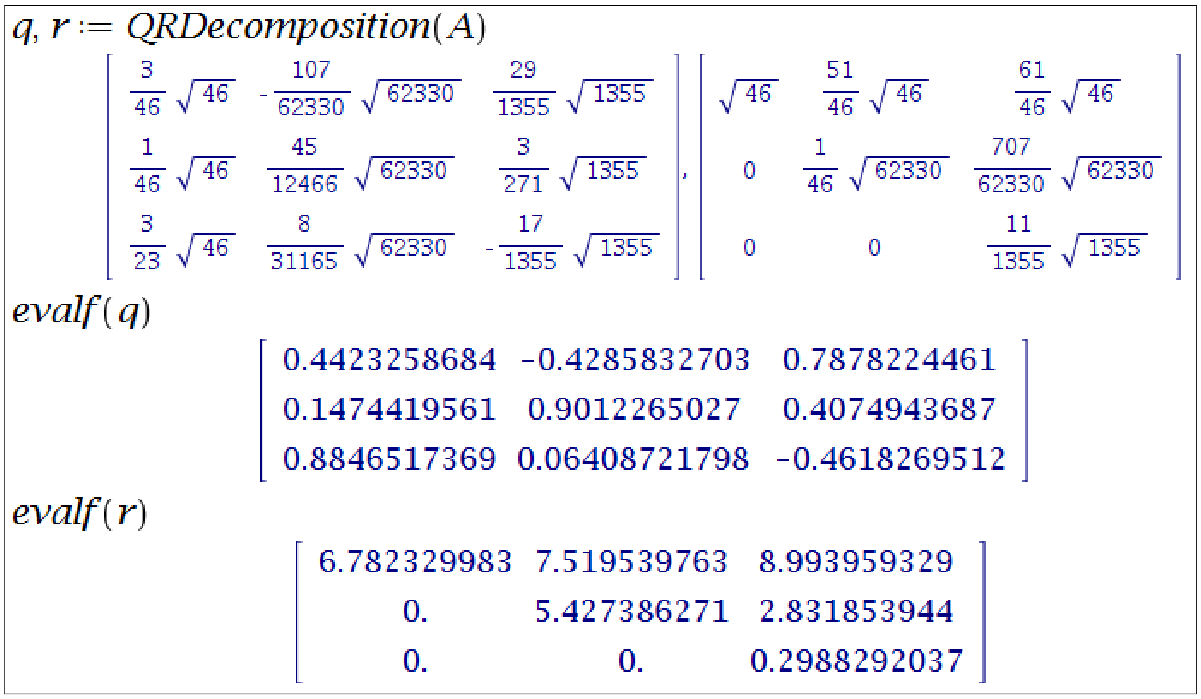
$$R := simplify\left(Q2^{\%T}.Q1A\right):\ \ evalf(R)$$

$$\begin{bmatrix} -6.782329983 & -7.519539763 & -8.993959329 \\ 0. & -5.427386271 & -2.831853944 \\ 0. & 0. & -0.2988292036 \end{bmatrix}$$

$$simplify(Q.R)$$

$$\begin{bmatrix} 3 & 1 & 3 \\ 1 & 6 & 4 \\ 6 & 7 & 8 \end{bmatrix}$$

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 9.2 for transforming the QR decomposition for all diagonal entries of R are nonnetative.

9.2.4. The QR method for finding eigenvalues

$\boxed{\textbf{Algorithm}}$ **9.30. (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
    $(b)$ $A_k = R_k Q_k$;                                   (9.56)
    $(c)$ $U_k = U_{k-1} Q_k$;  % Update transformation matrix
end for
set $T := A_\infty$ and $U := U_\infty$
```

Claim 9.31.

- Algorithm 9.30 produces an upper triangular matrix T, with its diagonals being **eigenvalues** of A, and an orthogonal matrix U such that

$$A = UTU^T, \tag{9.57}$$

which is called the **Schur decomposition** of A.

- If A is symmetric, then T becomes a diagonal matrix of **eigenvalues** of A and U is the collection of corresponding **eigenvectors**.

Remark 9.32. It follows from (a) and (b) of Algorithm 9.30 that

$$A_k = R_k Q_k = Q_k^T A_{k-1} Q_k, \tag{9.58}$$

and therefore

$$\begin{aligned} A_k &= R_k Q_k = Q_k^T A_{k-1} Q_k = Q_k^T Q_{k-1}^T A_{k-2} Q_{k-1} Q_k = \cdots \\ &= Q_k^T Q_{k-1}^T \cdots Q_1^T\, A_0\, \underbrace{Q_1 Q_2 \cdots Q_k}_{U_k} \end{aligned} \tag{9.59}$$

The above converges to

$$T = U^T A U \tag{9.60}$$

Example 9.33. 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 9.30, 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 9.9.

9.3. Singular Value Decomposition

Here we will deal with the SVD in detail.

$\boxed{\textbf{Theorem}}$ **9.34. (SVD Theorem).** *Let $A \in \mathbb{R}^{m\times n}$ with $m \ge n$. Then we can write*

$$A = U\,\Sigma\,V^T, \tag{9.61}$$

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

$$\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_n \ge 0.$$

Remark 9.35. The matrices are illustrated pictorially as

$$\begin{bmatrix} \\ \\ A \\ \\ \\ \end{bmatrix} = \begin{bmatrix} \\ \\ U \\ \\ \\ \end{bmatrix} \begin{bmatrix} \\ \Sigma \\ \\ \end{bmatrix} \begin{bmatrix} \\ V^T \\ \\ \end{bmatrix}, \tag{9.62}$$

where

U	:	$m \times n$ orthogonal (the **left singular vectors** of A.)
Σ	:	$n \times n$ diagonal (the **singular values** of A.)
V	:	$n \times n$ orthogonal (the **right singular vectors** of A.)

- For some $r \le n$, the singular values may satisfy

$$\underbrace{\sigma_1 \,\ge\, \sigma_2 \,\ge\, \cdots \,\ge\, \sigma_r}_{\text{nonzero singular values}} > \sigma_{r+1} = \cdots = \sigma_n = 0. \tag{9.63}$$

  In this case, *rank* $(A) = r$.
- If $m < n$, the **SVD** is defined by considering A^T.

Proof. (of Theorem 9.34) 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
$$A = \left(U\begin{bmatrix} 1 & 0 \\ 0 & U_1 \end{bmatrix}\right) \begin{bmatrix} \sigma & 0 \\ 0 & \Sigma_1 \end{bmatrix} \left(V\begin{bmatrix} 1 & 0 \\ 0 & V_1 \end{bmatrix}\right)^T. \tag{9.64}$$

Equation (9.64) is our desired decomposition. □

9.3.1. Interpretation of the SVD

Algebraic interpretation of the SVD

Let *rank* $(A) = r$. let the SVD of A be $A = U\,\Sigma\,V^T$, with

$$\begin{aligned} U &= [\mathbf{u}_1 \quad \mathbf{u}_2 \quad \cdots \quad \mathbf{u}_n], \\ \Sigma &= \mathrm{diag}(\sigma_1, \sigma_2, \cdots, \sigma_n), \\ V &= [\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n], \end{aligned}$$

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

$$\begin{aligned} AV &= A[\mathbf{v}_1 \quad \mathbf{v}_2 \quad \cdots \quad \mathbf{v}_n] = [A\mathbf{v}_1 \quad A\mathbf{v}_2 \quad \cdots \quad A\mathbf{v}_n] \\ &= [\mathbf{u}_1 \quad \cdots \quad \mathbf{u}_r \quad \cdots \quad \mathbf{u}_n] \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix} \\ &= [\sigma_1\mathbf{u}_1 \quad \cdots \quad \sigma_r\mathbf{u}_r \quad \mathbf{0} \quad \cdots \quad \mathbf{0}]. \end{aligned} \tag{9.65}$$

Therefore,

$$A = U\,\Sigma\,V^T \Leftrightarrow \begin{cases} A\mathbf{v}_j = \sigma_j\mathbf{u}_j, & j = 1, 2, \cdots, r \\ A\mathbf{v}_j = \mathbf{0}, & j = r+1, \cdots, n \end{cases} \tag{9.66}$$

Similarly, starting from $A^T = V\,\Sigma\,U^T$,

$$A^T = V\,\Sigma\,U^T \Leftrightarrow \begin{cases} A^T\mathbf{u}_j = \sigma_j\mathbf{v}_j, & j = 1, 2, \cdots, r \\ A^T\mathbf{u}_j = \mathbf{0}, & j = r+1, \cdots, n \end{cases} \tag{9.67}$$

Summary 9.36. It follows from (9.66) and (9.67) that

- $(\mathbf{v}_j, \sigma_j^2)$, $j = 1, 2, \cdots, r$, are eigenvector-eigenvalue pairs of $A^T A$.

$$A^T A\,\mathbf{v}_j = A^T(\sigma_j \mathbf{u}_j) = \sigma_j^2 \mathbf{v}_j, \ \ j = 1, 2, \cdots, r. \tag{9.68}$$

  So, the singular values play the role of eigenvalues.

- Similarly, we have

$$AA^T\,\mathbf{u}_j = A(\sigma_j \mathbf{v}_j) = \sigma_j^2 \mathbf{u}_j, \ \ j = 1, 2, \cdots, r. \tag{9.69}$$

- Equation (9.68) gives how to find the **singular values** $\{\sigma_j\}$ and the **right singular vectors** V, while (9.66) 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

$$A = \sum_{j=1}^{n} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{9.70}$$

  When *rank*$\,(A) = r \le n$,

$$A = \sum_{j=1}^{r} \sigma_j \mathbf{u}_j \mathbf{v}_j^T. \tag{9.71}$$

  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 $\mathbb{R}^m$:

$$\mathcal{B}_1 = \{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n\} \xrightarrow{A} \mathcal{B}_2 = \{\sigma_1\mathbf{u}_1, \sigma_2\mathbf{u}_2, \cdots, \sigma_r\mathbf{u}_r\} \tag{9.72}$$

Consider a unit sphere $\mathcal{S}^{n-1}$ in $\mathbb{R}^n$:

$$\mathcal{S}^{n-1} = \Big\{\mathbf{x}\Big|\sum_{j=1}^n x_j^2 = 1\Big\}.$$

Then, $\forall\mathbf{x} \in \mathcal{S}^{n-1}$,

$$\begin{aligned}\mathbf{x} &= x_1\mathbf{v}_1 + x_2\mathbf{v}_2 + \cdots + x_n\mathbf{v}_n\\ A\mathbf{x} &= \sigma_1x_1\mathbf{u}_1 + \sigma_2x_2\mathbf{u}_2 + \cdots + \sigma_rx_r\mathbf{u}_r\\ &= y_1\mathbf{u}_1 + y_2\mathbf{u}_2 + \cdots + y_r\mathbf{u}_r, \quad (y_j = \sigma_jx_j)\end{aligned} \tag{9.73}$$

So, we have

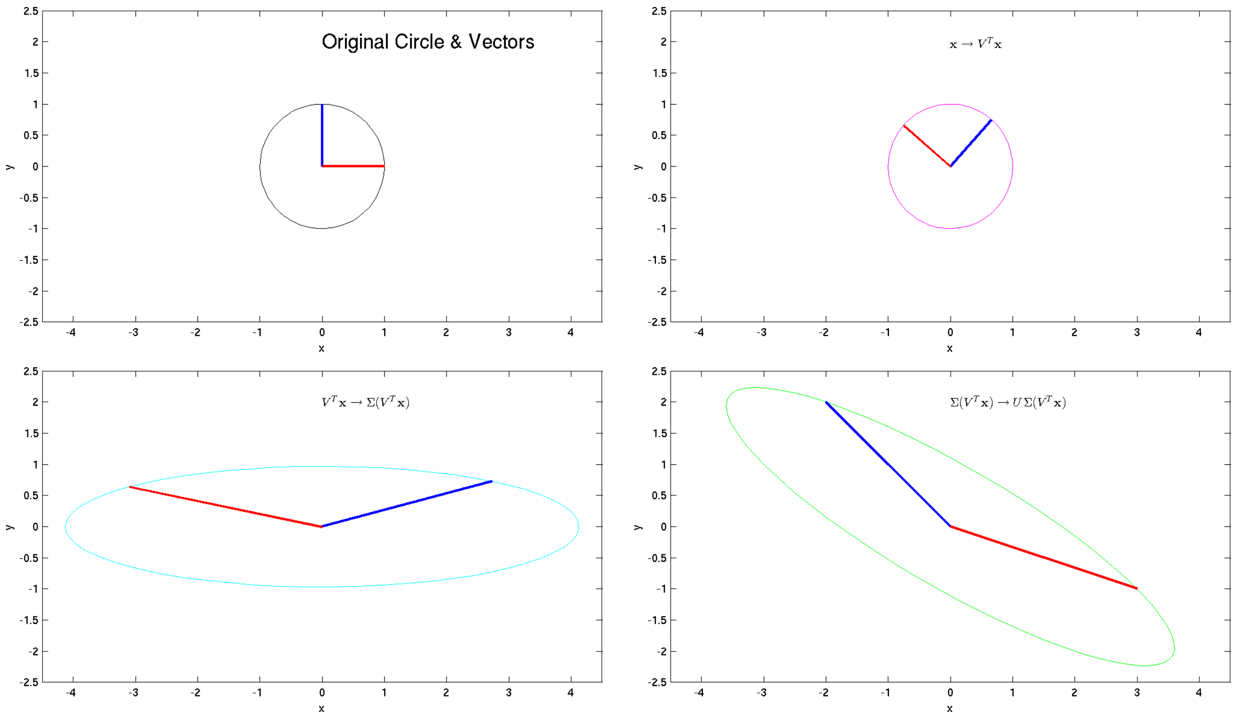
$$\begin{aligned} y_j = \sigma_jx_j \quad &\Longleftrightarrow \quad x_j = \frac{y_j}{\sigma_j}\\ \sum_{j=1}^n x_j^2 = 1 \ \text{(sphere)} \quad &\Longleftrightarrow \quad \sum_{j=1}^r \frac{y_j^2}{\sigma_j^2} = \alpha \le 1 \ \text{(ellipsoid)}\end{aligned} \tag{9.74}$$

Example 9.37. *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 $\mathrm{x} \in \mathcal{S}^1$,

$$A\mathbf{x} = U\Sigma V^T\mathbf{x} = U\left(\Sigma(V^T\mathbf{x})\right)$$



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)

9.3.2. Properties of the SVD

Theorem 9.38. *Let $A \in \mathbb{R}^{m\times n}$ with $m \le n$. Let $A = U\Sigma V^T$ be the SVD of A, with*

$$\sigma_1 \ge \cdots \ge \sigma_r > \sigma_{r+1} = \cdots = \sigma_n = 0.$$

Then,

$$\begin{cases} \textit{rank}\,(A) &= r \\ \textit{Null}(A) &= \textit{Span}\{\mathbf{v}_{r+1}, \cdots, \mathbf{v}_n\} \\ \textit{Range}(A) &= \textit{Span}\{\mathbf{u}_1, \cdots, \mathbf{u}_r\} \\ A &= \displaystyle\sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \end{cases} \tag{9.75}$$

and

$$\begin{cases} ||A||_2 &= \sigma_1 \quad \text{(See Exercise 9.3.)} \\ ||A||_F^2 &= \sigma_1^2 + \cdots + \sigma_r^2 \quad \text{(See Exercise 9.4.)} \\ \displaystyle\min_{\mathbf{x}\neq\mathbf{0}} \frac{||A\mathbf{x}||_2}{||\mathbf{x}||_2} &= \sigma_n \qquad (m \ge n) \end{cases} \tag{9.76}$$

Theorem 9.39. *Let $A \in \mathbb{R}^{m\times n}$ with rank $(A) = r > 0$. Let $A = U\Sigma V^T$ be the SVD of A, with singular values*

$$\sigma_1 \ge \cdots \ge \sigma_r > 0.$$

Define, for $k = 1, \cdots, r-1$,

$$A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T \quad \text{(sum of rank-1 matrices)}. \tag{9.77}$$

Then, rank $(A_k) = k$ and

$$\begin{aligned} ||A - A_k||_2 &= \min\{||A - B||_2 \,\big|\, \textit{rank}\,(B) \le k\} = \sigma_{k+1}, \\ ||A - A_k||_F^2 &= \min\{||A - B||_F^2 \,\big|\, \textit{rank}\,(B) \le k\} = \sigma_{k+1}^2 + \cdots + \sigma_r^2. \end{aligned} \tag{9.78}$$

That is, of all matrices of rank $\le k$, $\boldsymbol{A_k}$ **is closest to** $\boldsymbol{A}$.

Note: The matrix A_k in (9.77) can be written as

$$A_k = \sum_{j=1}^{k} \sigma_j \mathbf{u}_j \mathbf{v}_j^T = U\,\Sigma_k V^T, \tag{9.79}$$

where $\Sigma_k = \text{diag}(\sigma_1, \cdots, \sigma_k, 0, \cdots, 0)$.

9.3.3. Computation of the SVD

For $A \in \mathbb{R}^{m\times n}$, the procedure is as follows.

1. Form $A^T A$, the **covariance matrix** of A.
2. Find the eigen-decomposition of $A^T A$ by orthogonalization process, i.e., $\Lambda = \text{diag}(\lambda_1, \cdots, \lambda_n)$,
$$A^T A = V\Lambda V^T, \tag{9.80}$$
where $V = [\mathbf{v}_1 \quad \cdots \quad \mathbf{v}_n]$ is orthogonal, i.e., $V^T V = I$.
3. Sort the eigenvalues according to their magnitude and let
$$\sigma_j = \sqrt{\lambda_j}, \;\; j = 1, 2, \cdots, n. \tag{9.81}$$
4. Form the U matrix as follows (*rank*$(A) = k$),
$$\mathbf{u}_j = \frac{1}{\sigma_j} A\mathbf{v}_j, \quad j = 1, 2, \cdots, k. \tag{9.82}$$

Note:

- Note that $\mathbf{u}_j$ are eigenvectors of AA^T, as shown in (9.69). The alternative computation of $\{\mathbf{u}_j\}$ in (9.82) 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 9.40. *Find the SVD for* $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$.

Solution.

1. $A^T A = \begin{bmatrix} 14 & 6 \\ 6 & 9 \end{bmatrix}$.

2. Solving $det\,(A^T A - \lambda I) = 0$ gives the eigenvalues of $A^T A$

$$\lambda_1 = 18 \text{ and } \lambda_2 = 5,$$

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 [8, 9]

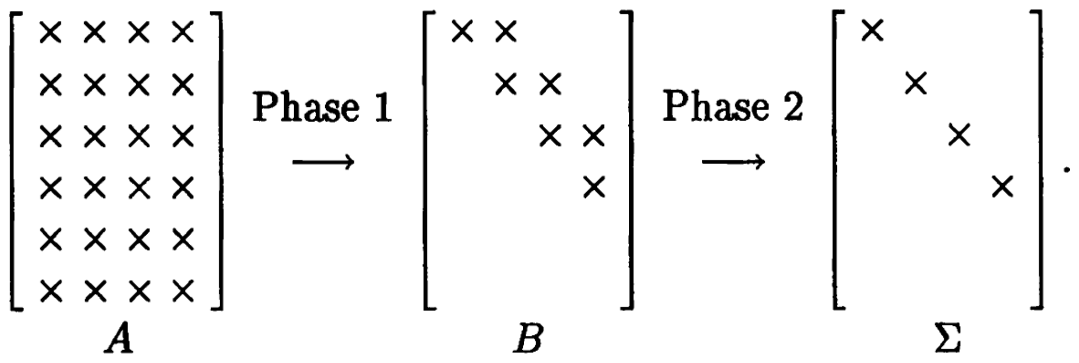


Figure 9.1: A two-phase procedure for the SVD: $A = U\Sigma V^T$.

Algorithm 9.41. (Golub and Reinsch, 1970) [9]. Let $A \in \mathbb{R}^{m\times n}$.

- **Phase 1**: It **constructs two finite sequences** of **Householder transformations** to find an **upper bidiagonal matrix**:

$$P_n \cdots P_1\, A\, Q_1 \cdots Q_{n-2} \;=\; B \tag{9.83}$$

- **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

$$A = \begin{bmatrix} 1/3 & 1/3 & 2/3 \\ 2/3 & 2/3 & 4/3 \\ 1/3 & 2/3 & 3/3 \\ 2/5 & 2/5 & 4/5 \\ 3/5 & 1/5 & 4/5 \end{bmatrix} \tag{9.84}$$

- 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,
  $$\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_r > T \ge \sigma_{r+1} \ge \cdots \tag{9.85}$$

In Matlab

- Matlab has a "rank" command, which computes the numerical rank of the matrix with a default threshold
  $$T = 2 \max\{m, n\}\, \epsilon\, ||A||_2 \tag{9.86}$$
  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 (9.84),
  $$T = 2 \cdot 5 \cdot \text{eps} \cdot 2.5987 = 5.7702 \times 10^{-15}$$
  and therefore `rank(A)=2`.

See Exercise 9.5.

9.3.4. Application of the SVD for LS problems

Recall: **(Definition 8.9, p. 291)**: Let $A \in \mathbb{R}^{m\times n}$, $m \ge n$, and $\mathbf{b} \in \mathbb{R}^m$. The **least-squares problem** is to find $\widehat{\mathbf{x}} \in \mathbb{R}^n$ which minimizes $\|A\mathbf{x} - \mathbf{b}\|_2$:

$$\begin{aligned} \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x}-\mathbf{b}\|_2, \\ &\quad \text{or, equivalently,} \\ \widehat{\mathbf{x}} &= \arg\min_{\mathbf{x}} \|A\mathbf{x}-\mathbf{b}\|_2^2, \end{aligned} \tag{9.87}$$

where $\widehat{\mathbf{x}}$ 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 8.14). It can be solved by the **method of normal equations**; the **unique** LS solution $\widehat{\mathbf{x}}$ is given by

$$\widehat{\mathbf{x}} = (A^TA)^{-1}A^T\mathbf{b}. \tag{9.88}$$

Definition 9.42. *$(A^TA)^{-1}A^T$ is called the* **pseudoinverse** *of A. Let $A = U\Sigma V^T$ be the SVD of A. Then*

$$(A^TA)^{-1}A^T = V\Sigma^{-1}U^T \overset{\text{def}}{=\!=} A^+. \tag{9.89}$$

Example 9.43. *Find the* **pseudoinverse** *of* $A = \begin{bmatrix} 1 & 2 \\ -2 & 1 \\ 3 & 2 \end{bmatrix}$.

Solution. From Example 9.40, 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 $A \in \mathbb{R}^{m\times n}$, $m > n$, with $\boldsymbol{rank}\,(\boldsymbol{A}) = \boldsymbol{k} \le \boldsymbol{n}$.

- Suppose that the SVD of A is given, that is,
$$A = U\Sigma V^T.$$
- Since U and V are ℓ^2-norm preserving, we have
$$||A\mathbf{x} - \mathbf{b}|| = ||U\Sigma V^T\mathbf{x} - \mathbf{b}|| = ||\Sigma V^T\mathbf{x} - U^T\mathbf{b}||. \tag{9.90}$$
- Define $\mathbf{z} = V^T\mathbf{x}$ and $\mathbf{c} = U^T\mathbf{b}$. Then
$$||A\mathbf{x} - \mathbf{b}|| = \Big(\sum_{i=1}^{k} (\sigma_i z_i - c_i)^2 + \sum_{i=k+1}^{n} c_i^2 \Big)^{1/2}. \tag{9.91}$$
- Thus the norm is minimized when z is chosen with
$$z_i = \begin{cases} c_i/\sigma_i, & \text{when } i \le k, \\ \text{arbitrary}, & \text{otherwise}. \end{cases} \tag{9.92}$$
- After determining $\mathbf{z}$, one can find the solution as
$$\widehat{\mathbf{x}} = V\mathbf{z}. \tag{9.93}$$

  Then the least-squares error reads
$$\min_{\mathbf{x}} ||A\mathbf{x} - \mathbf{b}|| = \Big(\sum_{i=k+1}^{n} c_i^2 \Big)^{1/2} \tag{9.94}$$

Strategy 9.44. When z is obtained as in (9.92), it is better to **choose zero** for the "arbitrary" part:

$$\mathbf{z} = [c_1/\sigma_1,\, c_2/\sigma_2,\, \cdots,\, c_k/\sigma_k,\, 0,\, \cdots,\, 0]^T. \tag{9.95}$$

In this case, z can be written as

$$\mathbf{z} \,=\, \Sigma_k^+\mathbf{c} \,=\, \Sigma_k^+ U^T\mathbf{b}, \tag{9.96}$$

where

$$\Sigma_k^+ \,=\, [1/\sigma_1,\, 1/\sigma_2,\, \cdots,\, 1/\sigma_k,\, 0,\, \cdots,\, 0]^T. \tag{9.97}$$

Thus the corresponding LS solution reads

$$\widehat{\mathbf{x}} \,=\, V\mathbf{z} \,=\, V\Sigma_k^+ U^T\mathbf{b}. \tag{9.98}$$

Note that $\widehat{\mathrm{x}}$ involves **no components of the null space** of A;
$\widehat{\mathrm{x}}$ is unique in this sense.

Remark 9.45.

- **When *rank* $(A) = k = n$**: It is easy to see that

$$V\Sigma_k^+ U^T = V\Sigma^{-1}U^T, \tag{9.99}$$

  which is the **pseudoinverse** of A.
- **When *rank* $(A) = k < n$**: A^TA not invertible. However,

$$A_k^+ \,:=\, V\Sigma_k^+ U^T \tag{9.100}$$

  plays the role of the pseudoinverse of A. Thus we will call it 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.

9.4. Principal Component Analysis

- **Principal component analysis** (PCA) (a.k.a. **orthogonal linear transformation**) was invented in 1901 by K. Pearson [19], as an analogue of the principal axis theorem in mechanics; it was later independently developed and named by H. Hotelling in the 1930s [11, 12].
- 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:

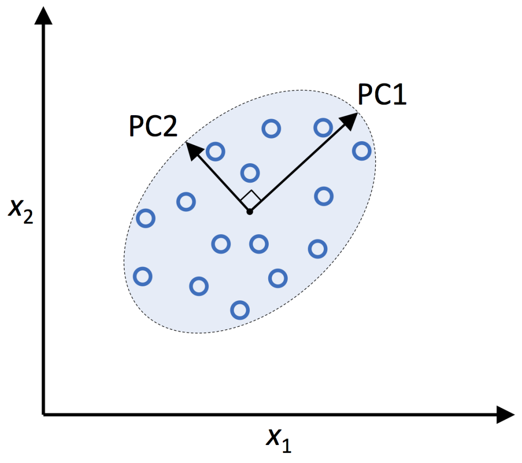


Figure 9.2: Principal components.

- 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 we want to assign equal importance to all features.

9.4.1. 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 (e.g., after standardization).
- Our goal is to find an **orthogonal** weight matrix $W \in \mathbb{R}^{d\times d}$ such that

$$Z = X\,W, \tag{9.101}$$

  where $Z \in \mathbb{R}^{N\times d}$ is call the **score matrix**. Columns of Z 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

$$\begin{aligned} \mathbf{w}_1 &= \arg\max_{\|\mathbf{w}\|=1} \|\mathbf{z}_1\|^2 = \arg\max_{\|\mathbf{w}\|=1} \|X\mathbf{w}\|^2 \\ &= \arg\max_{\|\mathbf{w}\|=1} \mathbf{w}^T X^T X\mathbf{w} = \arg\max_{\mathbf{w}\neq 0} \frac{\mathbf{w}^T X^T X\mathbf{w}}{\mathbf{w}^T\mathbf{w}}, \end{aligned} \tag{9.102}$$

where the quantity to be maximized can be recognized as a **Rayleigh quotient**.

Theorem 9.46. *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.,*

$$\mathbf{w}_1 = \arg\max_{\mathbf{w}\neq 0} \frac{\mathbf{w}^T X^T X\mathbf{w}}{\mathbf{w}^T\mathbf{w}} = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}, \quad (X^TX)\mathbf{v}_1 = \lambda_1\mathbf{v}_1, \tag{9.103}$$

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:

$$\widehat{X}_k := X - \sum_{i=1}^{k-1} X\mathbf{w}_i\mathbf{w}_i^T, \tag{9.104}$$

and then ② finding the weight vector which extracts the maximum variance from this new data matrix

$$\mathbf{w}_k = \arg\max_{\|\mathbf{w}\|=1} \|\widehat{X}_k\mathbf{w}\|^2. \tag{9.105}$$

Thus $\mathbf{w}_k$ is an eigenvector of $\widehat{X_k}^T \widehat{X}_k$ and therefore an eigenvector of X^TX.

Summary 9.47. The transformation matrix W is the stack of eigenvectors of X^TX, i.e.,

$$W = [\mathbf{w}_1|\mathbf{w}_2|\cdots|\mathbf{w}_d], \quad (X^TX)\,\mathbf{w}_j = \lambda_j\,\mathbf{w}_j, \;\; \mathbf{w}_i^T\mathbf{w}_j = \delta_{ij}, \tag{9.106}$$

where $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_d \ge 0$.

With W found, a **data vector** $\mathbf{x}$ is transformed to a d-dimensional row vector of principal components

$$\mathbf{z} = \mathbf{x}W, \tag{9.107}$$

of which components z_j, $j = 1, 2, \cdots, d$, are decorrelated.

Remark 9.48. The principal components transformation is closely related with the **SVD** of X:

$$X = U\,\Sigma V^T, \tag{9.108}$$

where

$U \;:\; n\times d$ orthogonal (the **left singular vectors** of X.)
$\Sigma \;:\; d\times d$ diagonal (the **singular values** of X.)
$V \;:\; d\times d$ orthogonal (the **right singular vectors** of X.)

- The matrix Σ explicitly reads

  $$\Sigma \;=\; \text{diag}(\sigma_1, \sigma_2, \cdots, \sigma_d), \tag{9.109}$$

  where $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_d \ge 0$.
- In terms of this factorization, the matrix X^TX reads

  $$X^TX = (U\,\Sigma V^T)^T U\,\Sigma V^T = V\,\Sigma U^T U\,\Sigma V^T = V\Sigma^2 V^T. \tag{9.110}$$

- Comparing with the **eigenvector factorization** of X^TX, we have
  - the right singular vectors $V \cong$ the eigenvectors of $X^TX \Rightarrow V \cong W$
  - the square of singular values of X are equal to the eigenvalues of X^TX $\Rightarrow \sigma_j^2 = \lambda_j,\ j = 1, 2, \cdots, d$.

Claim 9.49. While the weight matrix $W \in \mathbb{R}^{d\times d}$ is the collection of eigenvectors of X^TX, the score matrix $Z \in \mathbb{R}^{N\times d}$ is the stack of eigenvectors of XX^T, scaled by the square-root of eigenvalues:

$$Z = [\sqrt{\lambda_1}\,\mathbf{u}_1|\sqrt{\lambda_2}\,\mathbf{u}_2|\cdots|\sqrt{\lambda_d}\,\mathbf{u}_d], \quad (XX^T)\,\mathbf{u}_j = \lambda_j\,\mathbf{u}_j,\ \ \mathbf{u}_i^T\mathbf{u}_j = \delta_{ij}. \tag{9.111}$$

See (9.114).

9.4.2. Dimensionality reduction: 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^TX ($k \ll d$), gives the truncated transformation:

$$Z_k = X\,W_k : \mathbf{x}^{(i)} \in \mathbb{R}^d \mapsto \mathbf{z}^{(i)} \in \mathbb{R}^k, \tag{9.112}$$

where $Z_k \in \mathbb{R}^{N\times k}$ and $W_k \in \mathbb{R}^{d\times k}$. Let

$$X_k := Z_k W_k^T. \tag{9.113}$$

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

$$Z = XW = U\,\Sigma V^T W = U\,\Sigma, \tag{9.114}$$

  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 (9.111).

- 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:

$$Z_k = XW_k = U\,\Sigma V^T W_k = U\,\Sigma_k, \tag{9.115}$$

  where

$$\Sigma_k := \text{diag}(\sigma_1, \cdots, \sigma_k, 0, \cdots, 0). \tag{9.116}$$

- Now, using (9.115), the truncated data matrix reads

$$X_k = Z_k W_k^T = U\,\Sigma_k W_k^T = U\,\Sigma_k W^T = U\,\Sigma_k V^T. \tag{9.117}$$

Claim 9.50. It follows from (9.108) and (9.117) that

$$\begin{aligned}\|X - X_k\| &= \|U\,\Sigma V^T - U\,\Sigma_k V^T\| \\ &= \|U(\Sigma - \Sigma_k)V^T\| \\ &= \|\Sigma - \Sigma_k\| = \sigma_{k+1},\end{aligned} \tag{9.118}$$

where $\|\cdot\|$ is the induced matrix L^2-norm.

Remark 9.51. Efficient algorithms exist to calculate the SVD of X without having to form the matrix $X^T X$; see § 9.3.3. **Computing the SVD is now the standard way to carry out the PCA** [10, 26].

Exercises for Chapter 9

9.1. Prove (c) in Theorem 9.5. ***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 $\mathbf{v}_2^T A \mathbf{v}_1$.

9.2. 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 $N_j^2 = I$.

9.3. Let $A \in \mathbb{R}^{m\times n}$. Prove that $||A||_2 = \sigma_1$, the largest singular value of A. **Hint**: Use the following

$$\frac{||A\mathbf{v}_1||_2}{||\mathbf{v}_1||_2} = \frac{\sigma_1 ||\mathbf{u}_1||_2}{||\mathbf{v}_1||_2} = \sigma_1 \Longrightarrow ||A||_2 \geq \sigma_1$$

and arguments around Equations (9.73) and (9.74) for the opposite directional inequality.

9.4. 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 $||UB||_F = ||B||_F$.

9.5. C 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.)

(Continued on the next page)

9.6. C 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.

(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.

9.7. 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.

9.8. 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.)

9.9. C This problem revisits Exercise 8.3, p. 310. 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 the LS code (you have implemented for Exercise 8.3) to find the curve.
(d) Use **QR decomposition** to find $\widehat{\boldsymbol{\beta}}$ and the curve.
(You do not have to implement a code for QR or the SVD.)
Clue: Let $X = QR$. Then $X\boldsymbol{\beta} = \mathbf{y}$ can be written as $R\boldsymbol{\beta} = Q^T\mathbf{y}$.
(e) Use the **SVD** to find $\widehat{\boldsymbol{\beta}}$ and the curve.
(f) Plot the curves superposed over the point plot, and compre them. Are they the same?

9.10. Left signular vectors $\mathbf{u}_j$ of A are eigenvectors of AA^T, as shown in (9.69). However, it can be *alternatively and more effeciently* computed as in (9.82):

$$\mathbf{u}_j = \frac{1}{\sigma_j} A\mathbf{v}_j, \quad j = 1, 2, \cdots, k.$$

Prove $\mathbf{u}_i^T \mathbf{u}_j = \delta_{ij}$.

CHAPTER 10
Machine Learning

In This Chapter: You will see **key ideas** in **machine learning** (ML).

Topics	**Applications/Properties**
Machine Learning What is it?	
Binary Classifiers Perceptron, Adaline Logistic Regression	Simplest neural networks
Support Vector Machine Linear SVM Nonlinear SVM Sequential Minimal Optimization	Maximizes the margin Method of Lagrange multipliers Kernel trick
Neural Networks MNIST data Stochastic Gradient Descent	
Deep Learning Convolutional Neural Networks	

Contents of Chapter 10

10.1. What is Machine Learning?

Definition 10.1. 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 10.2. There are three different types of ML:

- **Supervised learning**: e.g., classification, regression
  - Labeled data
  - Direct feedback
  - Preduct outcome/future
- **Unsupervised learning**: e.g., clustering
  - No labels
  - No feedback
  - Find hidden structure in data
- **Reinforcement learning**: e.g., chess engine
  - Decision process
  - Reward system
  - Learn series of actions

Note: The most popular type is **supervised learning**.

10.1.1. 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:

$$\begin{cases} \text{Given : } & \text{A } \textbf{training data } \{(\mathbf{x}_i, y_i) \mid i = 1, \cdots, N\} \\ \text{Find : } & \widehat{f} : X \to Y, \text{ a good approximation to } f \end{cases} \tag{10.1}$$

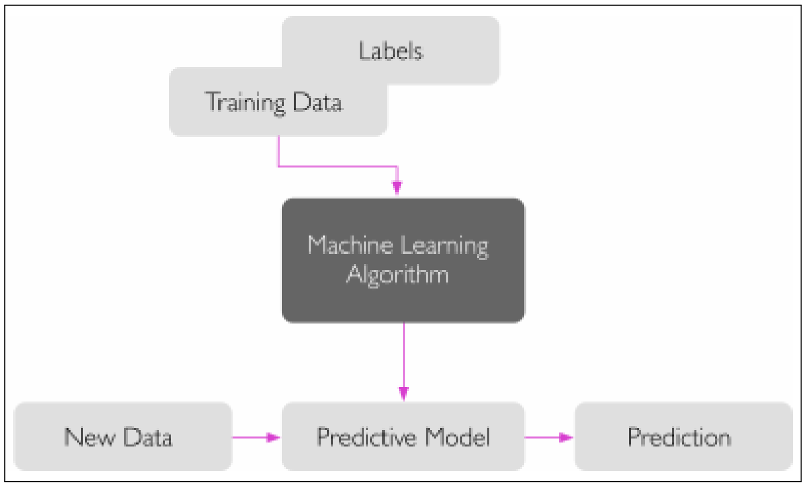


Figure 10.1: Supervised learning and prediction.

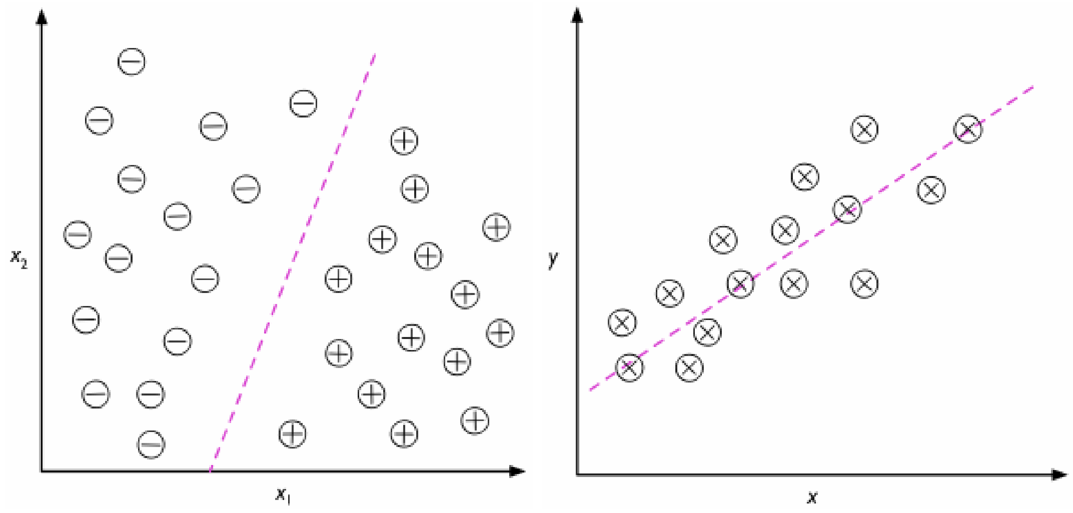
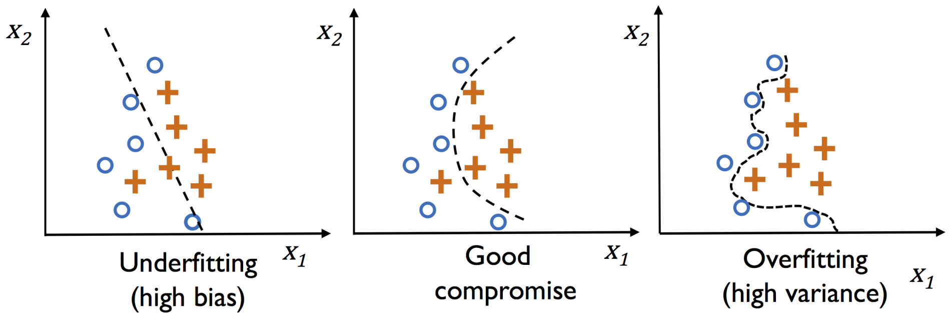


Figure 10.2: Classification and regression.

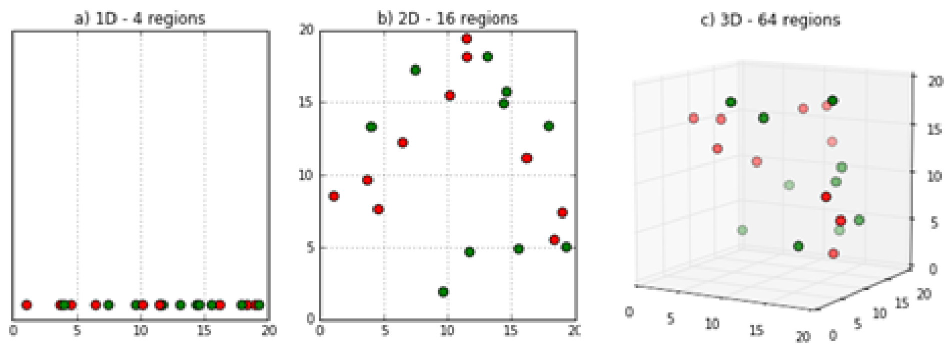
Why is ML not Always Simple?

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 invovles 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 (deep 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**.

10.1.2. 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.

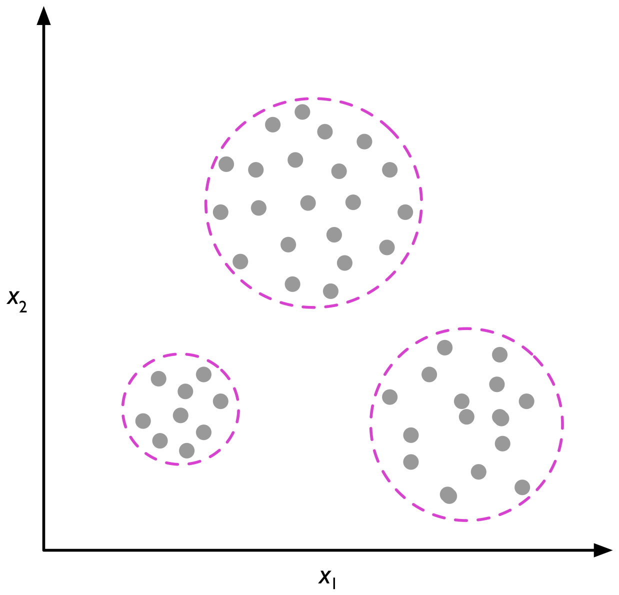


Figure 10.3: Clustering.

10.2. 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 [20], Adaline, Logistic Regression, Support Vector Machine [3]

Remark 10.3. 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.

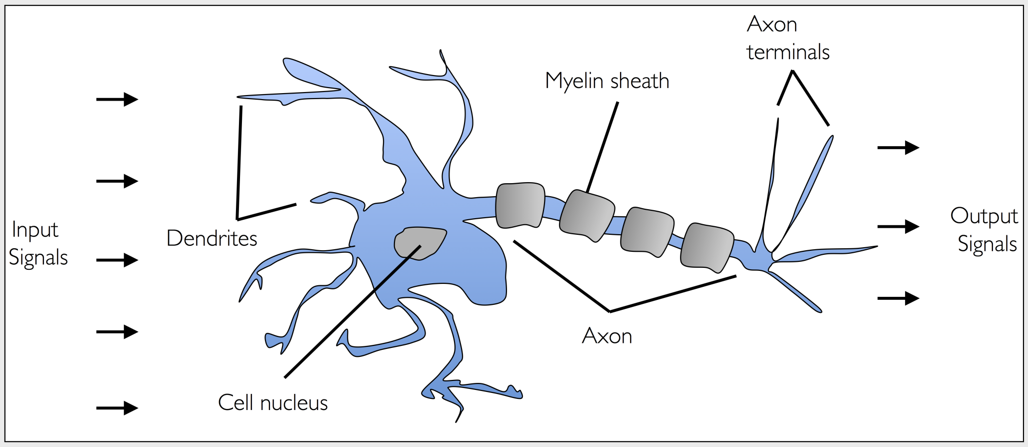


Figure 10.4: A schematic description of a neuron.

Definition 10.4. 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; see Figure 10.2, p. 365.

Observation 10.5. Let's consider the following to interpret binary classifiers in a unified manner.

- The **labels** (0 and 1) are chosen for simplicity.
- A **hyperplane** can be formulated by a normal vector $\mathbf{w} \in \mathbb{R}^d$ and a shift (bias) b:

$$z = \mathbf{w}^T\mathbf{x} + b. \tag{10.2}$$

  In ML, z is called the **net input**.

  - The net input can go very high in magnitude for some $\mathbf{x}$.
  - It is a weighted sum (linear combination) of input features $\mathbf{x}$.

- **Activation function**: 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 network; we apply an **activation function** $\phi(z)$.

- To **learn $\mathbf{w}$ and b**, you may formulate a **cost function** to minimize, as the **Sum of Squared Errors** (SSE):

$$\mathcal{J}(\mathbf{w}) = \frac{1}{2}\sum_i \Big(y^{(i)} - \phi(\mathbf{w}^T\mathbf{x}^{(i)})\Big)^2. \tag{10.3}$$

  However, in order to get them **more effectively**, we must formulate the cost function **more meaningfully**.

Thus, an important task in ML is on the choice of effective

- **activation functions** and
- **cost functions**.

Activation functions:

$$\begin{array}{lll} \text{Perceptron} & : & \phi(z) = \begin{cases} 1, & \text{if } z \ge \theta \\ 0, & \text{otherwise} \end{cases} \\ \text{Adaline} & : & \phi(z) = z \\ \text{Logistic Regression} & : & \phi(z) = \sigma(z) := \dfrac{1}{1+e^{-z}} \end{array} \tag{10.4}$$

where "Adaline" stands for ADAptive LInear NEuron. The activation function $\sigma(z)$ called the **standard logistic sigmoid function** or simply the **sigmoid function**.

Remark 10.6. (The standard logistic sigmoid function)

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{10.5}$$

- The standard logistic function is the solution of the simple first-order non-linear ordinary differential equation

$$\frac{d}{dx}y = y(1-y), \quad y(0) = \frac{1}{2}. \tag{10.6}$$

- It can be verified easily as

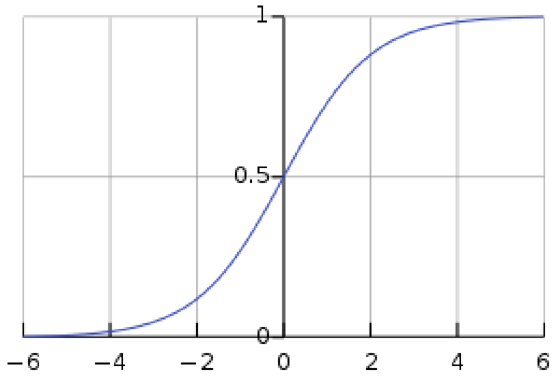
$$\sigma'(x) = \frac{e^x(1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} = \sigma(x)(1-\sigma(x)). \tag{10.7}$$

- σ' is even: $\sigma'(-x) = \sigma'(x)$.
- **Rotational symmetry** about $(0, 1/2)$:

$$\sigma(x) + \sigma(-x) = \frac{1}{1+e^{-x}} + \frac{1}{1+e^{x}} = \frac{2+e^x+e^{-x}}{2+e^x+e^{-x}} \equiv 1. \tag{10.8}$$

- $\int \sigma(x)\,dx = \int \frac{e^x}{1+e^x}\,dx = \ln(1+e^x)$, 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

$$f(x) = x^+ = \max(x, 0). \tag{10.9}$$



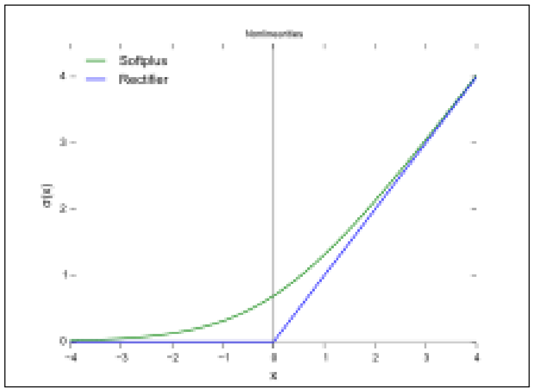


Figure 10.5: **Popular activation functions**: (left) The standard logistic sigmoid function and (right) the rectifier and softplus function.

10.2.1. Adaline

Algorithm 10.7. Adaline Learning:
From data $\{(\mathbf{x}^{(i)}, y^{(i)})\}$, learn the weights $\mathbf{w}$ and bias b, with

- **Activation function**: $\phi(z) = z$ (i.e., identity activation)
- **Cost function**: the SSE

$$\mathcal{J}(\mathbf{w}, b) = \frac{1}{2}\sum_i \left(y^{(i)} - \phi(z^{(i)})\right)^2. \tag{10.10}$$

where $z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)} + b$ and $\phi = I$, the identity.

The dominant algorithm for the minimization of the cost function is the the Gradient Descent Method.

Algorithm 10.8. The Gradient Descent Method uses $-\nabla\mathcal{J}$ for the **search direction** (update direction):

$$\begin{aligned} \mathbf{w} &= \mathbf{w} + \Delta\mathbf{w} &&= \mathbf{w} - \eta\nabla_{\mathbf{w}}\mathcal{J}(\mathbf{w}, b), \\ b &= b + \Delta b &&= b - \eta\nabla_b\mathcal{J}(\mathbf{w}, b), \end{aligned} \tag{10.11}$$

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

$$\begin{aligned} \frac{\partial\mathcal{J}(\mathbf{w},b)}{\partial w_j} &= -\sum_i \Big(y^{(i)} - \phi(z^{(i)})\Big)\, x_j^{(i)}, \\ \frac{\partial\mathcal{J}(\mathbf{w},b)}{\partial b} &= -\sum_i \Big(y^{(i)} - \phi(z^{(i)})\Big). \end{aligned} \tag{10.12}$$

Thus, with $\phi = I$,

$$\begin{aligned} \Delta\mathbf{w} &= -\eta\nabla_{\mathbf{w}}\mathcal{J}(\mathbf{w},b) &= \eta\sum_i \Big(y^{(i)} - \phi(z^{(i)})\Big)\,\mathbf{x}^{(i)}, \\ \Delta b &= -\eta\nabla_b\mathcal{J}(\mathbf{w},b) &= \eta\sum_i \Big(y^{(i)} - \phi(z^{(i)})\Big). \end{aligned} \tag{10.13}$$

Hyperparameters

Definition 10.9. 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`)

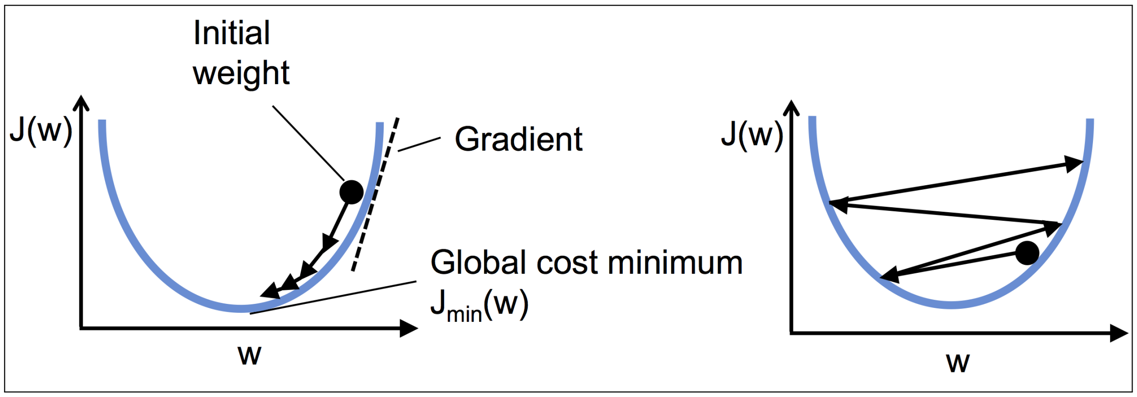


Figure 10.6: Well-chosen learning rate vs. a large learning rate

Note: There are effective searching schemes to set the learning rate η automatically.

10.2.2. Logistic Regression

Algorithm 10.10. 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:
$$\mathcal{J}(\mathbf{w}, b) = \sum_i \Big[-y^{(i)} \ln\Big(\phi(z^{(i)})\Big) - (1-y^{(i)}) \ln\Big(1-\phi(z^{(i)})\Big)\Big], \qquad (10.14)$$
  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:
$$\frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial w_j} = \sum_i \Big[-y^{(i)} \frac{1}{\phi(z^{(i)})} + (1-y^{(i)}) \frac{1}{1-\phi(z^{(i)})}\Big] \frac{\partial \phi(z^{(i)})}{\partial w_j}, \qquad (10.15)$$
where, using $z^{(i)} = \mathbf{w}^T\mathbf{x}^{(i)}$ and (10.7),
$$\frac{\partial \phi(z^{(i)})}{\partial w_j} = \phi'(z^{(i)}) \frac{\partial z^{(i)}}{\partial w_j} = \phi(z^{(i)})\Big(1-\phi(z^{(i)})\Big)\, x_j^{(i)}.$$
Thus, if follows from the above and (10.15) that
$$\begin{aligned} \frac{\partial \mathcal{J}(\mathbf{w}, b)}{\partial w_j} &= \sum_i \Big[-y^{(i)}\Big(1-\phi(z^{(i)})\Big) + (1-y^{(i)})\phi(z^{(i)})\Big]\, x_j^{(i)} \\ &= -\sum_i \Big[y^{(i)} - \phi(z^{(i)})\Big]\, x_j^{(i)} \end{aligned}$$
and therefore
$$\nabla_{\mathbf{w}} \mathcal{J}(\mathbf{w}) = -\sum_i \Big[y^{(i)} - \phi(z^{(i)})\Big]\, \mathbf{x}^{(i)}. \qquad (10.16)$$
Similarly, one can get
$$\nabla_b \mathcal{J}(\mathbf{w}) = -\sum_i \Big[y^{(i)} - \phi(z^{(i)})\Big]. \qquad (10.17)$$

Algorithm 10.11. *Gradient descent learning for Logistic Regression is formulated as*

$$\mathbf{w} := \mathbf{w} + \Delta\mathbf{w}, \quad b := b + \Delta b, \tag{10.18}$$

where $\eta > 0$ *is the* ***step length*** *(learning rate) and*

$$\begin{aligned} \Delta\mathbf{w} &= -\eta\nabla_{\mathbf{w}}\mathcal{J}(\mathbf{w}, b) &= \eta\sum_i \Big[y^{(i)} - \phi(z^{(i)})\Big]\,\mathbf{x}^{(i)}, \\ \Delta b &= -\eta\nabla_b\mathcal{J}(\mathbf{w}, b) &= \eta\sum_i \Big[y^{(i)} - \phi(z^{(i)})\Big]. \end{aligned} \tag{10.19}$$

Note: The above gradient descent rule for Logistic Regression is of the same form as that of Adaline; see (10.13) on p. 373. Only the difference is the activation function ϕ.

10.3. Support Vector Machine

Observation 10.12. ML algorithms such as Perceptron, Adaline, and Logistic Regression would **stop the iteration**, when the linear separator becomes one of hyperplanes (lines) shown in the left of Figure 10.7.

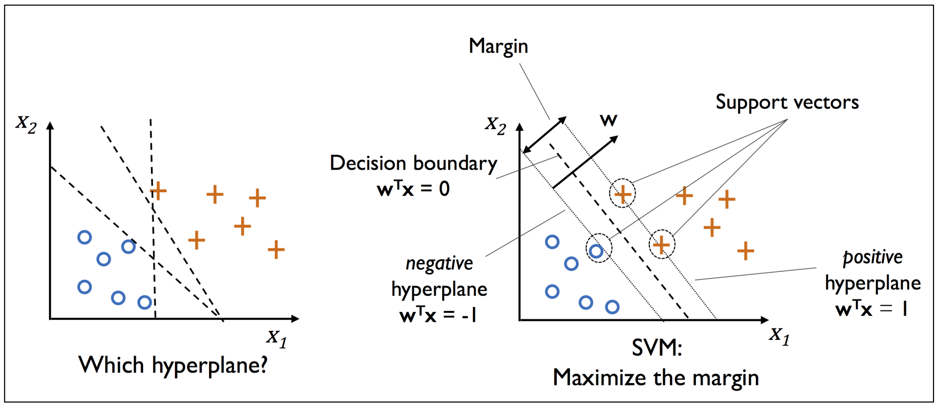


Figure 10.7: (left) Converged linear separators and (right) the one that maximizes the margin.

10.3.1. Linear SVM

- **Support Vector Machine (SVM)**, developed in 1995 by Cortes-Vapnik [3], 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:

$$\begin{aligned} w_0 + \mathbf{w}^T\mathbf{x}_+ &= 1, \\ w_0 + \mathbf{w}^T\mathbf{x}_- &= -1. \end{aligned} \tag{10.20}$$

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
$$\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot(\mathbf{x}_+ - \mathbf{x}_-) = \frac{2}{\|\mathbf{w}\|}. \tag{10.21}$$
- Note $\mathbf{w} = [w_1, w_2, \cdots, w_d]^T$ is a **normal vector** to the decision boundaries (hyperplanes); the left side of (10.21) is the distance between the positive and negative hyperplanes.
- Maximizing the distance (margin) is equivalent to minimizing its reciprocal $\frac{1}{2}\|\mathbf{w}\|$, or minimizing $\frac{1}{2}\|\mathbf{w}\|^2$.

Problem 10.13. *The **linear SVM** is formulated as*
$$\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \textit{subject to} \\ \left[\begin{array}{ll} w_0 + \mathbf{w}^T\mathbf{x}^{(i)} \ge 1 & \textit{if } y^{(i)} = 1, \\ w_0 + \mathbf{w}^T\mathbf{x}^{(i)} \le -1 & \textit{if } y^{(i)} = -1. \end{array}\right. \tag{10.22}$$

Remark 10.14. The constraints in Problem 10.13 can be written as
$$y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1 \ge 0, \quad \forall\, i. \tag{10.23}$$

- 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 (10.22) are typically solved using the **method of Lagrange multipliers**.

10.3.2. The method of Lagrange multipliers

In this subsection, we briefly consider Lagrange's method to solve the problem of the form

$$\min/\max_{\mathbf{x}} f(\mathbf{x}) \quad \textbf{subj.to} \quad g(\mathbf{x}) = c. \tag{10.24}$$

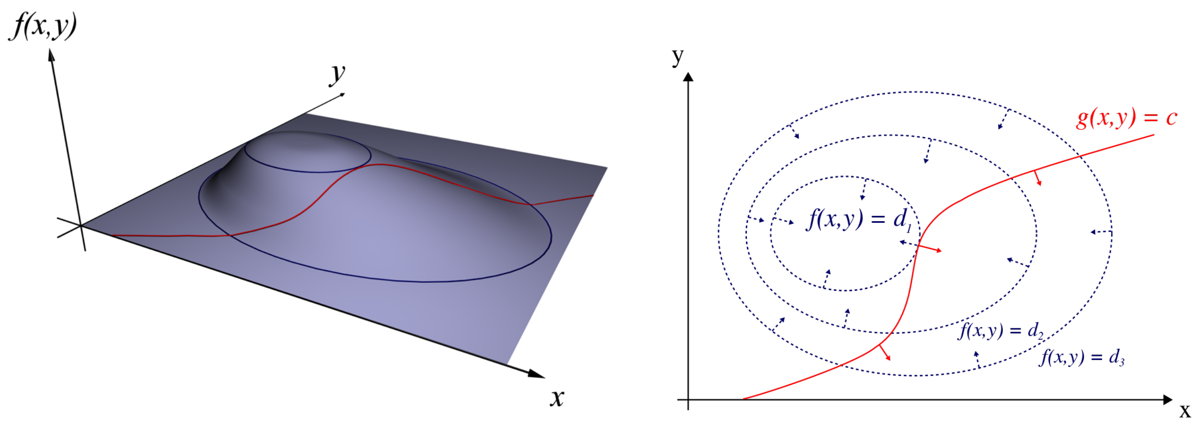


Figure 10.8: The method of Lagrange multipliers in $\mathbb{R}^2$: **∇f // ∇g, at optimum**.

Strategy 10.15. (Method of Lagrange multipliers). For the maximum and minimum values of $\mathrm{f}(\mathrm{x})$ **subject to** $\mathrm{g}(\mathrm{x}) = \mathrm{c}$,

(a) Find x and λ such that

$$\boxed{\nabla f(\mathbf{x}) = \lambda \nabla g(\mathbf{x}) \text{ and } g(\mathbf{x}) = c.}$$

(b) Evaluate f at all these points, to find the maximum and minimum.

Self-study 10.16. 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, $\begin{cases} 2x = 2x\,\lambda & \textcircled{1} \\ 4y = 2y\,\lambda & \textcircled{2} \\ x^2 + y^2 = 1 & \textcircled{3} \end{cases}$

From $\textcircled{1}$, $x = 0$ or $\lambda = 1$.

Ans: min: $f(\pm 1, 0) = 1$; max: $f(0, \pm 1) = 2$

Lagrange multipliers – Dual variables

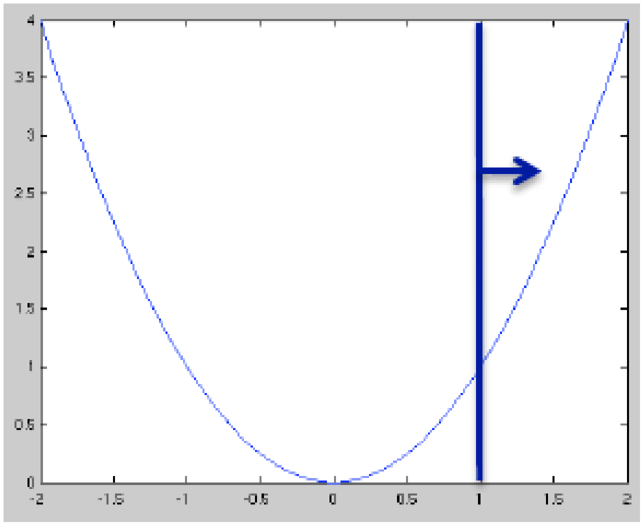


Figure 10.9: $\min_x x^2$ subj.to $x \geq 1$.

For simplicity, consider

$$\min_x x^2 \quad \text{subj.to } x \geq 1. \tag{10.25}$$

Rewriting the constraint

$$x - 1 \geq 0,$$

introduce **Lagrangian (objective)**:

$$\mathcal{L}(x, \alpha) = x^2 - \alpha\,(x - 1). \tag{10.26}$$

Now, consider

$$\min_x \max_\alpha \ \mathcal{L}(x, \alpha) \ \text{subj.to } \alpha \geq 0. \tag{10.27}$$

Claim 10.17. The minimization problem (10.25) is equivalent to the max-min problem (10.27).

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 (10.27)**. In the following, we will define the **dual problem** of (10.27), which is equivalent to the **primal problem**.

Recall: The min-max problem in (10.27), which is equivalent to the (original) primal problem:

$$\min_x \max_\alpha \mathcal{L}(x,\alpha) \quad \text{subj.to} \quad \alpha \ge 0, \qquad \text{(Primal)} \tag{10.28}$$

where

$$\mathcal{L}(x,\alpha) = x^2 - \alpha\,(x-1).$$

Definition 10.18. *The **dual problem** of (10.28) is formulated by swapping* $\min_x$ *and* $\max_\alpha$ *as follows:*

$$\max_\alpha \min_x \mathcal{L}(x,\alpha) \quad \textit{subj.to} \quad \alpha \ge 0, \qquad \textit{(Dual)} \tag{10.29}$$

The term $\underline{\min_x \mathcal{L}(x,\alpha)}$ *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 $\underline{\min_x \mathcal{L}(x,\alpha)}$, the minimum occurs where the gradient is equal to zero.

$$\frac{d}{dx}\mathcal{L}(x,\alpha) = 2x - \alpha = 0 \;\;\Rightarrow\;\; x = \frac{\alpha}{2}. \tag{10.30}$$

Plugging this to $\mathcal{L}(x,\alpha)$, we have

$$\mathcal{L}(x,\alpha) = \Big(\frac{\alpha}{2}\Big)^2 - \alpha\Big(\frac{\alpha}{2} - 1\Big) = \alpha - \frac{\alpha^2}{4}.$$

We can rewrite the dual problem (10.29) as

$$\max_{\alpha\ge 0}\Big[\alpha - \frac{\alpha^2}{4}\Big]. \qquad \text{(Dual)} \tag{10.31}$$

$\Rightarrow$ **the maximum is 1 when $\alpha^* = 2$** **(for the dual problem)**.
Plugging $\alpha = \alpha^*$ into (10.30) to get $x^* = 1$. Or, using the Lagrangian objective, we have

$$\mathcal{L}(x,\alpha) = x^2 - 2(x-1) = (x-1)^2 + 1. \tag{10.32}$$

$\Rightarrow$ **the minimum is 1 when $x^* = 1$** **(for the primal problem)**. □

10.3.3. Solution of the linear SVM

Recall: The **linear SVM** formulated in Problem 10.13:

$$\begin{aligned} &\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subj.to} \\ &y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1 \ge 0, \quad \forall\, i. \end{aligned} \qquad \text{(Primal)} \qquad (10.33)$$

To solve the problem, let's begin with its **Lagrangian**:

$$\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1], \qquad (10.34)$$

where $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \cdots, \alpha_N]^T$, the dual variable (Lagrange multipliers).

Definition 10.19. In optimization, the **Karush-Kuhn-Tucker (KKT) conditions** [14, 16] 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.

Note: Allowing inequality constraints, the KKT approach to nonlinear programming generalizes the method of Lagrange multipliers, which allows only equality constraints.

Writing the **KKT conditions**, starting with Lagrangian stationarity, where we need to find the first-order derivatives w.r.t. w and w_0:

$$\boxed{\begin{aligned}
\nabla_{\mathbf{w}}\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= \mathbf{w} - \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)}, \\
\frac{\partial}{\partial w_0}\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= -\sum_{i=1}^{N} \alpha_i y^{(i)} = 0 \qquad \Rightarrow \sum_{i=1}^{N} \alpha_i y^{(i)} = 0, \\
&\alpha_i \geq 0, \qquad \textbf{(dual feasibility)} \\
&\alpha_i \left[y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \right] = 0, \qquad \textbf{(complementary slackness)} \\
&y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 \geq 0. \qquad \textbf{(primal feasibility)}
\end{aligned} \qquad (10.35)}$$

Using the KKT conditions (10.35), we can simplify the Lagrangian:

$$\begin{aligned}
\mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\alpha}) &= \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{N} \alpha_i y^{(i)} w_0 - \sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} + \sum_{i=1}^{N} \alpha_i \\
&= \frac{1}{2}\|\mathbf{w}\|^2 - 0 - \mathbf{w}^T \mathbf{w} + \sum_{i=1}^{N} \alpha_i \\
&= -\frac{1}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{N} \alpha_i.
\end{aligned} \tag{10.36}$$

Again using the first KKT condition, we can rewrite the first term.

$$\begin{aligned}
-\frac{1}{2}\|\mathbf{w}\|^2 &= -\frac{1}{2}\Big(\sum_{i=1}^{N} \alpha_i y^{(i)} \mathbf{x}^{(i)}\Big) \cdot \Big(\sum_{j=1}^{N} \alpha_j y^{(j)} \mathbf{x}^{(j)}\Big) \\
&= -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)}\, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)}.
\end{aligned} \tag{10.37}$$

Plugging (10.37) into the (simplified) Lagrangian (10.36), we see that the Lagrangian now depends on $\boldsymbol{\alpha}$ only.

Problem 10.20. *The **dual problem** of (10.33) is formulated as*

$$\max_{\boldsymbol{\alpha}} \Big[\sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y^{(i)}y^{(j)}\,\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}\Big], \quad \textit{subj.to} \\ \left[\begin{array}{l} \alpha_i \ge 0, \quad \forall\, i, \\ \sum_{i=1}^{N} \alpha_i y^{(i)} = 0. \end{array}\right. \tag{10.38}$$

Remark 10.21. (Solving the dual problem).

- We can solve the dual problem (10.38), by using either a generic quadratic programming solver or the **Sequential Minimal Optimization (SMO)**, which we will discuss in § 10.3.6, p. 394.
- 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

$$\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y^{(i)}\mathbf{x}^{(i)}. \tag{10.39}$$

- We still need to get w_0^*.

Remark 10.22. The objective function $\mathcal{L}(\boldsymbol{\alpha})$ in (10.38) is a linear combination of the dot products of data samples $\{\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}\}$, which will be used when we generalize the SVM for nonlinear decision boundaries; see § 10.3.5.

Support vectors

Assume momentarily that we have w_0^*. Consider the **complementary slackness KKT condition** along with the primal and dual feasibility conditions:

$$\alpha_i^* \, [y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) - 1] = 0$$
$$\Rightarrow \begin{cases} \alpha_i^* > 0 \Rightarrow y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) = 1 \\ \alpha_i^* < 0 \quad \text{(can't happen)} \\ y^{(i)}(w_0^* + \mathbf{w}^{T}\mathbf{x}^{(i)}) - 1 > 0 \Rightarrow \alpha_i^* = 0 \\ y^{(i)}(w_0^* + \mathbf{w}^{T}\mathbf{x}^{(i)}) - 1 < 0 \quad \text{(can't happen).} \end{cases} \tag{10.40}$$

We define the **optimal (scaled) scoring function**:

$$f^*(\mathbf{x}^{(i)}) = w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}. \tag{10.41}$$

Then

$$\begin{cases} \alpha_i^* > 0 & \Rightarrow \; y^{(i)} f^*(\mathbf{x}^{(i)}) = \text{scaled margin} = 1, \\ y^{(i)} f^*(\mathbf{x}^{(i)}) > 1 & \Rightarrow \; \alpha_i^* = 0. \end{cases} \tag{10.42}$$

Definition 10.23. 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:

$$y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) \ge 1 \quad \text{and} \quad \min_i y^{(i)}(w_0^* + \mathbf{w}^{*T}\mathbf{x}^{(i)}) = 1.$$

If you take a positive support vector ($y^{(i)} = 1$), then

$$w_0^* = 1 - \min_{i:y^{(i)}=1} \mathbf{w}^{*T}\mathbf{x}^{(i)}. \tag{10.43}$$

Here, you'd better refer to **Summary of SVM** in Algorithm 10.26, p. 389.

10.3.4. The inseparable case: soft-margin classification

When the dataset is inseparable, there would be no separating hyperplane; there is no feasible solution to the linear SVM.

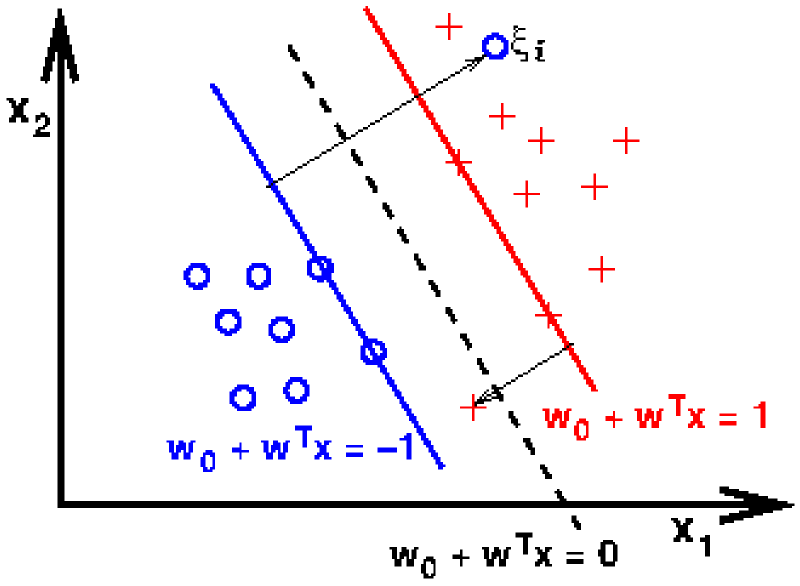


Figure 10.10: Slack variable: ξ_i.

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 10.13:

$$\begin{aligned}&\min_{\mathbf{w},w_0} \frac{1}{2}\|\mathbf{w}\|^2, \quad \text{subj.to} \\ &y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) - 1 \geq 0, \quad \forall\, i.\end{aligned} \qquad \text{(Primal)} \qquad (10.44)$$

Let's change it to this new primal problem:

Problem 10.24. ***(Soft-margin classification).*** *The SVM with the slack variable is formulated as*

$$\begin{aligned}&\min_{\mathbf{w},w_0,\boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i \quad \textit{subj.to} \\ &\left[\begin{array}{l} y^{(i)}(w_0 + \mathbf{w}^T\mathbf{x}^{(i)}) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{array}\right.\end{aligned} \qquad \textit{(Primal)} \qquad (10.45)$$

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:

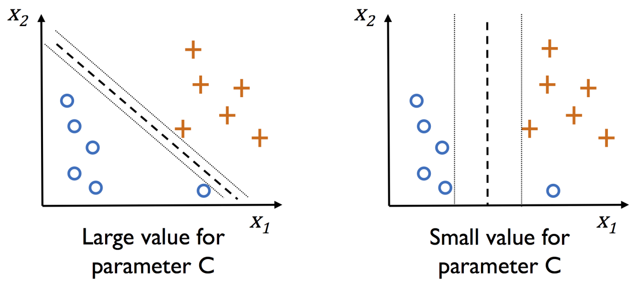


Figure 10.11: Bias-variance trade-off, via C.

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:

$$\xi_i = \left\{ \begin{array}{ll} 0, & \text{if } y^{(i)} f(\mathbf{x}^{(i)}) \ge 1 \\ 1 - y^{(i)} f(\mathbf{x}^{(i)}), & \text{otherwise} \end{array} \right\} = [1 - y^{(i)} f(\mathbf{x}^{(i)})]_+$$

Thus the objective function for soft-margin classification becomes

$$\min_{\mathbf{w}, w_0, \boldsymbol{\xi}} \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} [1 - y^{(i)} f(\mathbf{x}^{(i)})]_+. \tag{10.46}$$

The Dual for soft-margin classification

Form the Lagrangian of (10.45):

$$\begin{aligned} \mathcal{L}([\mathbf{w}, w_0], \boldsymbol{\xi}, \boldsymbol{\alpha}, \mathbf{r}) &= \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} r_i \xi_i \\ &\quad - \sum_{i=1}^{N} \alpha_i [y^{(i)}(w_0 + \mathbf{w}^T \mathbf{x}^{(i)}) - 1 + \xi_i], \end{aligned} \tag{10.47}$$

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 10.25. The **dual problem** of (10.44) is formulated as

$$\max_{\boldsymbol{\alpha}} \Big[\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)} \, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \Big], \quad \text{subj.to} \\ \left[\begin{array}{l} 0 \le \alpha_i \le C, \quad \forall\, i, \\ \sum_{i=1}^{N} \alpha_i y^{(i)} = 0. \end{array} \right. \tag{10.48}$$

So the only difference from the original problem's Lagrangian (10.38) is that $0 \le \alpha_i$ was changed to $0 \le \alpha_i \le C$. Neat!

See § 10.3.6, p. 394, for the solution of (10.48), using the SMO algorithm.

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** (10.48) can be written as

$$\max_{0\le\boldsymbol{\alpha}\le C} \Big[\boldsymbol{\alpha}\cdot\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}^T Z Z^T \boldsymbol{\alpha}\Big] \quad \textbf{subj.to} \quad \boldsymbol{\alpha}\cdot\mathbf{y} = 0. \tag{10.49}$$

Note:

- $G = ZZ^T \in \mathbb{R}^{N\times N}$ is called the **Gram matrix**. That is,

$$G_{ij} = y^{(i)}y^{(j)}\,\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}. \tag{10.50}$$

- The optimization problem in (10.48) or (10.49) is a typical form of **quadratic programming** (QP) problems.
- The dual problem (10.49) admits a unique solution.

Algorithm 10.26. (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 in § 10.3.6)
  - Compute the weights: $\mathbf{w}^* = \sum_{i=1}^{N} \alpha_i^* y^{(i)}\mathbf{x}^{(i)}$ (10.39)
  - Compute the intercept: $w_0^* = 1 - \min_{i:y^{(i)}=1} \mathbf{w}^{*T}\mathbf{x}^{(i)}$ (10.43)
- **Classification** (for a new sample x)
  - Compute $k_i = \mathbf{x} \cdot \mathbf{x}^{(i)}$ for support vectors $\mathbf{x}^{(i)}$
  - Compute $f(\mathbf{x}) = w_0^* + \sum_i \alpha_i y^{(i)} k_i \ \ (:= w_0^* + \mathbf{w}^{*T}\mathbf{x})$ (10.22)
  - Test $\mathrm{sign}(f(\mathbf{x}))$.

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

10.3.5. 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 10.12, we define

$$\phi(x_1, x_2) = (z_1, z_2, z_2) = (x_1, x_2, x_1^2 + x_2^2).$$

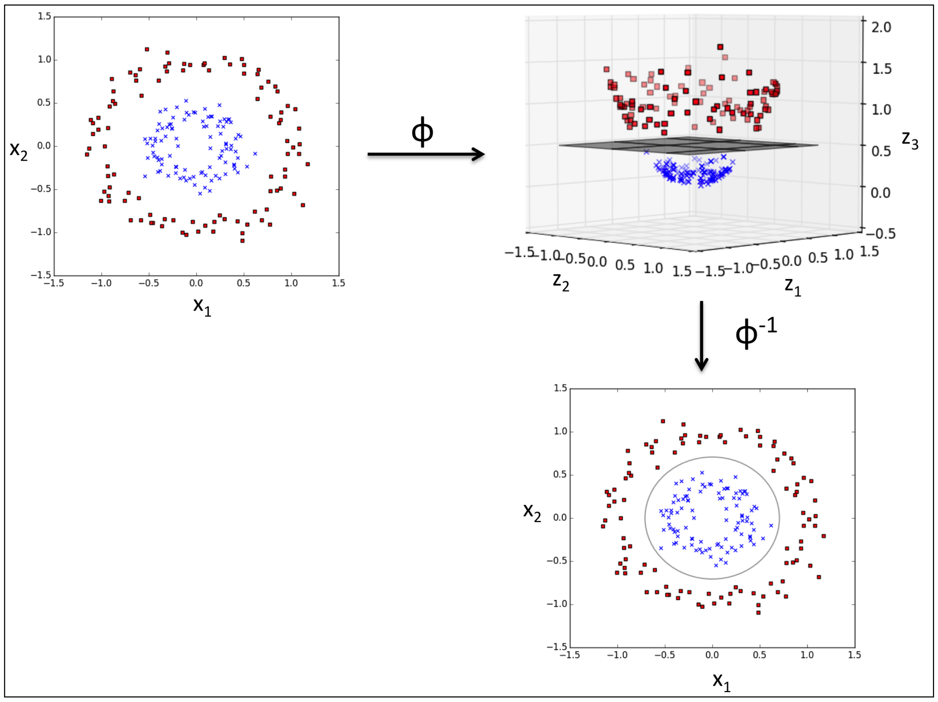


Figure 10.12: Inseparable dataset, feature expansion, and kernel SVM.

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 (10.48):

$$\max_{\boldsymbol{\alpha}} \Big[\sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y^{(i)}y^{(j)}\,\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}\Big], \quad \text{subj.to} \quad \left[\begin{array}{l} 0\le\alpha_i\le C, \qquad \forall\, i, \\ \sum_{i=1}^N \alpha_i y^{(i)} = 0. \end{array}\right. \tag{10.51}$$

Observation 10.27. 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 10.28. 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**:

$$\mathcal{K}(\mathbf{x}^{(i)},\mathbf{x}^{(j)}) \approx \phi(\mathbf{x}^{(i)})\cdot\phi(\mathbf{x}^{(j)}). \tag{10.52}$$

One of the most widely used kernels is the **Radial Basis Function (RBF)** kernel or simply called the **Gaussian kernel**:

$$\mathcal{K}(\mathbf{x}^{(i)},\mathbf{x}^{(j)}) = \exp\Big(-\frac{\|\mathbf{x}^{(i)}-\mathbf{x}^{(j)}\|^2}{2\sigma^2}\Big) = \exp\Big(-\gamma\|\mathbf{x}^{(i)}-\mathbf{x}^{(j)}\|^2\Big), \tag{10.53}$$

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 10.26, p. 389; 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. $k = 2$):
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^k \tag{10.54}$$
- Polynomial of degree up to k: for some $c > 0$,
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (c + \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)})^k \tag{10.55}$$
- Sigmoid:
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(a\, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} + b) \tag{10.56}$$
- Gaussian RBF:
$$\mathcal{K}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\Big(-\frac{\|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2}{2\sigma^2}\Big) \tag{10.57}$$
- And many others: Fisher kernel, graph kernel, string kernel, ...
very active area of research!

Example 10.29. (Quadratic kernels). $\mathcal{K}(\mathbf{x}, \mathbf{z}) = (c + \mathbf{x} \cdot \mathbf{z})^2$
$$\begin{aligned}(c + \mathbf{x} \cdot \mathbf{z})^2 &= \Big(c + \sum_{j=1}^m x_j z_j\Big)\Big(c + \sum_{\ell=1}^m x_\ell z_\ell\Big) \\ &= c^2 + 2c\sum_{j=1}^m x_j z_j + \sum_{j=1}^m \sum_{\ell=1}^m x_j z_j x_\ell z_\ell \\ &= c^2 + \sum_{j=1}^m (\sqrt{2c}x_j)(\sqrt{2c}z_j) + \sum_{j,\ell=1}^m (x_j x_\ell)(z_j z_\ell).\end{aligned} \tag{10.58}$$
So, the corresponding **feature expansion** is given by
$$\phi([x_1, \cdots, x_m]) = [x_1^2, x_1x_2, \cdots, x_m x_{m-1}, x_m^2,\ \sqrt{2c}x_1, \cdots, \sqrt{2c}x_m,\ c], \tag{10.59}$$
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

- $\boxed{\textbf{Perceptrons}}$ are a simple, popular way to learn a classifier
- They suffer from inefficient use of data, overfitting, and lack of expressiveness

- $\boxed{\textbf{SVMs}}$ 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*

10.3.6. Solving the dual problem with SMO

SMO (Sequential Minimal Optimization) is a type of coordinate ascent algorithm, but adapted to SVM so that the solution always stays within the feasible region.

Recall: The **dual problem** of the slack variable primal, formulated in (10.48):

$$\max_{\boldsymbol{\alpha}} \Big[\sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i=1}^N\sum_{j=1}^N \alpha_i\alpha_j y^{(i)}y^{(j)}\,\mathbf{x}^{(i)}\cdot\mathbf{x}^{(j)}\Big], \quad \text{subj.to} \quad \begin{bmatrix} 0 \le \alpha_i \le C, \quad \forall\, i, \\ \sum_{i=1}^N \alpha_i y^{(i)} = 0. \end{bmatrix} \tag{10.60}$$

Quesiton. Start with (10.60). Let's say you want to hold $\alpha_2, \cdots, \alpha_N$ fixed and take a coordinate step in the first direction. That is, change α_1 to maximize the objective in (10.60). Can we make any progress? Can we get a better feasible solution by doing this?

Turns out, no. Let's see why. Look at the constraint in (10.60), $\sum_{i=1}^N \alpha_i y^{(i)} = 0$. This means

$$\alpha_1 y^{(1)} = -\sum_{i=2}^N \alpha_i y^{(i)} \quad \Rightarrow \quad \alpha_1 = -y^{(1)}\sum_{i=2}^N \alpha_i y^{(i)}.$$

So, since $\alpha_2, \cdots, \alpha_N$ are fixed, α_1 is also fixed.

Thus, if we want to update any of the α_i's, **we need to update at least 2 of them simultaneously** to keep the solution feasible (i.e., to keep the constraints satisfied).

- Start with a feasible vector $\boldsymbol{\alpha}$.
- Let's update α_1 and α_2, holding $\alpha_3, \cdots, \alpha_N$ fixed. What values of α_1 and α_2 are we allowed to choose?
- The constraint is: $\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = -\sum_{i=3}^{N} \alpha_i y^{(i)} =: \xi$.

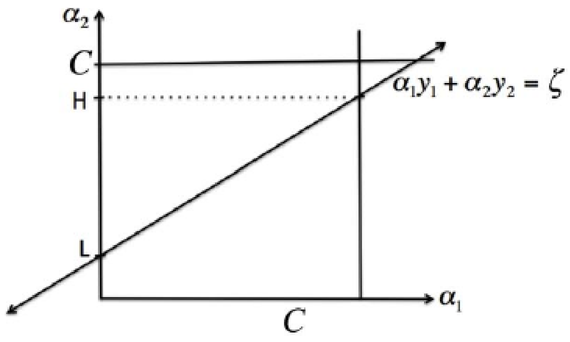


Figure 10.13

- We are only allowed to choose α_1 and α_2 on the line; when we pick α_2, we get α_1 automatically from

$$\alpha_1 = \frac{1}{y^{(1)}}(\xi - \alpha_2 y^{(2)}) = y^{(1)}(\xi - \alpha_2 y^{(2)}). \tag{10.61}$$

- **(Optimization for α_2)**: The other constraints in (10.60) says $0 \le \alpha_1, \alpha_2 \le C$. Thus, α_2 needs to be within $[L, H]$ on the figure ($\because \alpha_1 \in [0, C]$). To do the coordinate ascent step, we will optimize the objective over α_2, keeping it within $[L, H]$. Using (10.61), (10.60) becomes

$$\max_{\alpha_2 \in [L,H]} \Big[y^{(1)}(\xi - \alpha_2 y^{(2)}) + \alpha_2 + \sum_{i=3}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i \alpha_j y^{(i)} y^{(j)}\, \mathbf{x}^{(i)} \cdot \mathbf{x}^{(j)} \Big], \tag{10.62}$$

  of which the objective is quadratic in α_2. This means we can just set its derivative to 0 to optimize it, and to get α_2.
- Move to the next iteration of SMO.

Note: There are heuristics to choose the order of α_i's chosen to update.

10.4. Neural Networks

Recall: The **Perceptron** (or, Adaline, Logistic Regression) is **the simplest artificial neuron** that makes decisions by **weighting up** evidence.

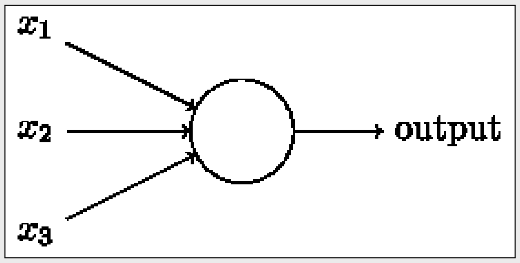


Figure 10.14: A simplest artificial neuron.

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.

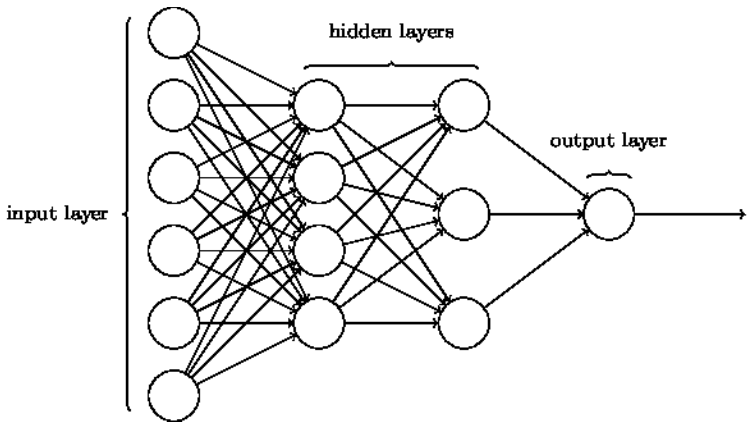


Figure 10.15: A complex neural network.

10.4.1. A simple network to classify handwritten digits

- The problem of recognizing handwritten digits has two components: segmentation and classification.

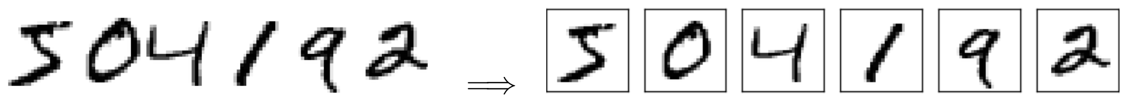


Figure 10.16: Segmentation.

- We'll focus on algorithmic components for the classification of individual digits.

MNIST data set:

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

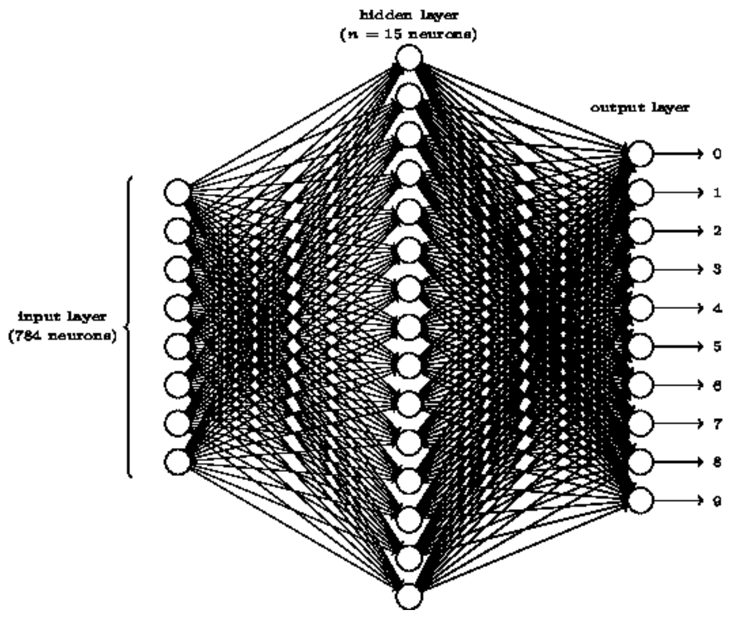
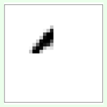


Figure 10.17: A sigmoid network **having a single hidden layer**.

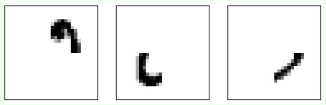
What the Neural Network Will Do

- 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.

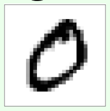
- **What are those hidden neurons doing?**
- Let's suppose **for the sake of argument** that **the first neuron** in the hidden layer may detect whether or not an image like the following is present



  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 10.16:



- 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**

$$C(\boldsymbol{W}, B) \;=\; \frac{1}{2N}\sum_i ||\mathbf{y}^{(i)} - \mathbf{a}(\mathbf{x}^{(i)})||^2, \tag{10.63}$$

  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**

$$\begin{bmatrix} \boldsymbol{W} \\ B \end{bmatrix} \leftarrow \begin{bmatrix} \boldsymbol{W} \\ B \end{bmatrix} + \begin{bmatrix} \Delta\boldsymbol{W} \\ \Delta B \end{bmatrix}, \tag{10.64}$$

  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:

$$\nabla C = \frac{1}{N}\sum_i \nabla C_{\mathbf{x}^{(i)}}. \tag{10.65}$$

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,
$$\frac{1}{m}\sum_{k=1}^{m} \nabla C_{\widetilde{\mathbf{x}}^{(k)}} \approx \nabla C \overset{\text{def}}{=\!=} \frac{1}{N}\sum_{i} \nabla C_{\mathbf{x}^{(i)}}. \tag{10.66}$$
- 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 (10.66).

10.4.2. Implementing a network to classify digits [18]

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 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.,

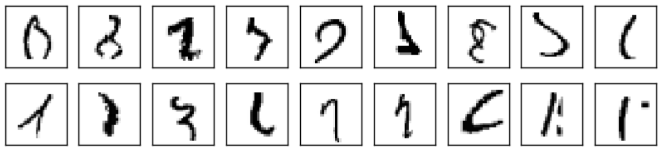


Figure 10.18: MNIST images difficult even for humans to recognize.

Moral of the Neural Networks

- Let all the complexity be learned, automatically, from data
- Simple algorithms can perform well for some problems:

(sophisticated algorithm) ≤ **(simple learning algorithm + good training data)**

10.5. Deep Learning: Convolutional Neural Networks

In this section, we will consider **deep neural networks**; the focus is on understanding core fundamental principles behind them, and applying those principles in the simple, easy-to-understand context of the MNIST problem.

Example 10.30. Consider neural networks for the classification of handwritten digits, as shown in the following images:

Figure 10.19: A few images in the MNIST data set.

A neural network can be built, with three hidden layers, as follows:

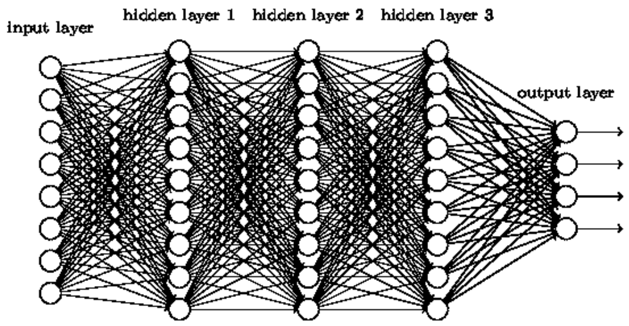


Figure 10.20

- Let each hidden layers have 30 neurons:
  - n_weights = $28^2 \cdot 30 + 30 \cdot 30 + 30 \cdot 30 + 30 \cdot 10 = 25,620$
  - n_biases = $30 + 30 + 30 + 10 = 100$
- Optimization is difficult
  - The number of parameters to teach is huge (**low efficiency**)
  - Multiple local minima problem (**low solvability**)
  - Adding hidden layers is **not necessarily improving** accuracy

In **fully-connected networks**, **deep** neural networks have been **hardly practical**, except for some special applications.

10.5.1. Introducing convolutional networks

Remarks 10.31. The neural network exemplified in Figure 10.20 can produce a **classification accuracy better than 98%**, for the MNIST handwritten digit data set. **But upon reflection, it's strange to use networks with fully-connected layers to classify images.**

- The network architecture does not take into account the **spatial structure of the images**.
- For instance, it treats **input pixels** which are far apart and close together, **on exactly the same footing**.

What if, instead of starting with a network architecture which is tabula rasa (blank mind), we used an architecture which tries to **take advantage of the spatial structure**? Could it be **better than 99%**?

Here, we will introduce **convolutional neural networks (CNN)**, which use a special architecture which is *particularly well-adapted to classify images*.

- The architecture makes convolutional networks **fast to train**.
- This, in turn, **helps train deep, many-layer networks**.
- Today, deep CNNs or some close variants are used in most neural networks for **image recognition**.

- CNNs use three basic ideas:

  (a) local receptive fields,

  (b) shared weights and biases, &

  (c) pooling.

(a) Local Receptive Fields

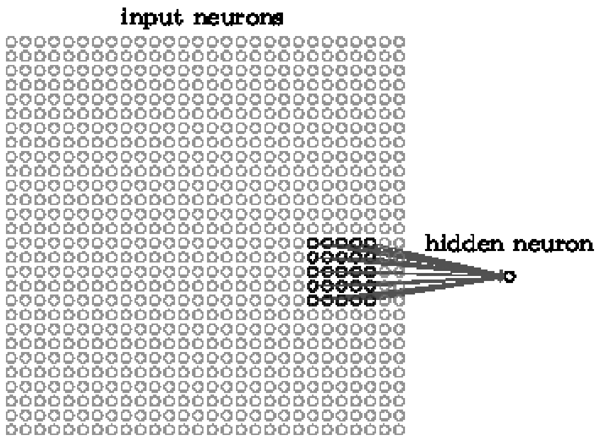


Figure 10.21: An illustration for local receptive fields.

- In CNNs, **the geometry of neurons (units) in the input layer** is exactly the same as that of images (e.g., 28×28).
  (rather than a vertical line of neurons as in fully-connected networks)
- As per usual, we'll **connect** the input neurons (pixels) to a layer of hidden neurons.
  - But we will **not connect fully** from every input pixel to every hidden neuron.
  - Instead, we only make connections in **small, localized regions** of the input image.
  - **For example**: Each neuron in the first hidden layer will be connected to a small region of the input neurons, say, a 5×5 region (Figure 10.21).
- That region in the input image is called the ***local receptive field*** for the hidden neuron.

- We slide the local receptive field across the entire input image.
  - For each local receptive field, there is a different hidden neuron in the first hidden layer.

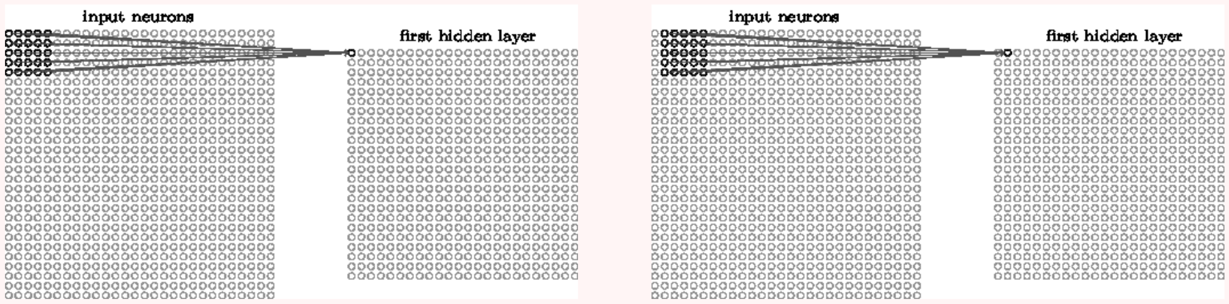


Figure 10.22: Two of local receptive fields, starting from the top-left corner. (Geometry of neurons in the first hidden layer is 24×24.)

Note: We have seen that the local receptive field is moved by one pixel at a time (`stride_length=1`).

- In fact, sometimes a different **stride length** is used.
  - For instance, we might move the local receptive field 2 pixels to the right (or down).
  - Most software gives a hyperparameter for the user to set the stride length.

(b) Shared Weights and Biases

Recall that each hidden neuron has **a bias** and 5×5 **weights** connected to its corresponding local receptive field.

- In CNNs, we use **the same weights and bias** for each of the 24×24 hidden neurons. In other words, for the (j, k)-th hidden neuron, the output is:

$$\sigma\Big(b + \sum_{p=0}^{4}\sum_{q=0}^{4} w_{p,q}\, a_{j+p,k+q}\Big), \tag{10.67}$$

  where σ is the neural activation function (e.g., the sigmoid function), b is the shared value for the bias, and $w_{p,q}$ is a 5×5 array of shared weights.
  - The weighting in (10.67) is just a form of **convolution**; we may rewrite it as
$$\mathbf{a}^1 = \sigma(b + \mathbf{w} * \mathbf{a}^0). \tag{10.68}$$
  - So the network is called a ***convolutional network***.
- We sometimes call the map, from the input layer to the hidden layer, a ***feature map***.
  - Suppose **the weights and bias** are such that the hidden neuron can **pick out a feature** (e.g., a vertical edge) in a particular local receptive field.
  - That ability is also likely to be useful at other places in the image.
  - And therefore it is useful to apply **the same feature detector** everywhere in the image.
- We call the weights and bias defining the feature map the ***shared weights*** and the ***shared bias***, respectively.
- A set of the shared weights and bias defines clearly **a kernel or filter**.

To put it in slightly more abstract terms, CNNs are well adapted to the **translation invariance** of images.[a]

[a]Move a picture of a cat a little ways, and it's still an image of a cat.

Remark 10.32. Multiple feature maps.

- The network structure we have considered so far can detect just **a single localized feature**.
- To do **more effective image recognition**, we'll need more than one **feature map**.
- Thus, a complete convolutional layer consists of **several different feature maps**:

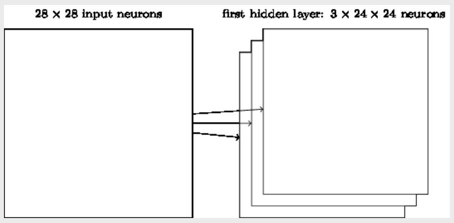


Figure 10.23: A convolutional network, consisting of 3 feature maps.

Modern CNNs are often built with 10 to 50 feature maps, each associated to a $r \times r$ local receptive field: $r = 3 \sim 9$.

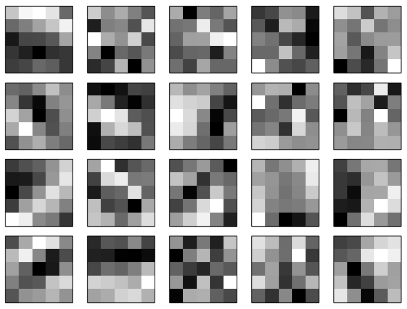


Figure 10.24: The 20 images corresponding to 20 different feature maps, which are actually learned when classifying the MNIST data set ($r = 5$).

The number of parameters to learn

A big advantage of sharing weights and biases is that it greatly reduces the number of parameters involved in a convolutional network.

- Convolutional networks: $(5 \times 5 + 1) * 20 = 520$
- Fully-connected networks: $(28 \times 28 + 1) * 20 = 15,700$

(c) Pooling

- CNNs also contain **pooling layers**, in addition to the convolutional layers just mentioned.
- Pooling layers are usually used **right after convolutional layers**.
- What they do is **to simplify the information** in the output from the convolutional layer.

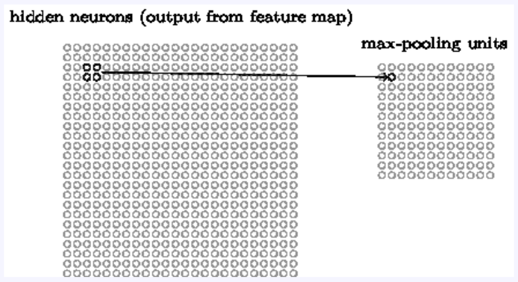


Figure 10.25: Pooling: summarizing a region of 2×2 neurons in the convolutional layer.

From Figure 10.23: Since we have 24×24 neurons output from the convolutional layer, after pooling we will have 12×12 neurons for each feature map:

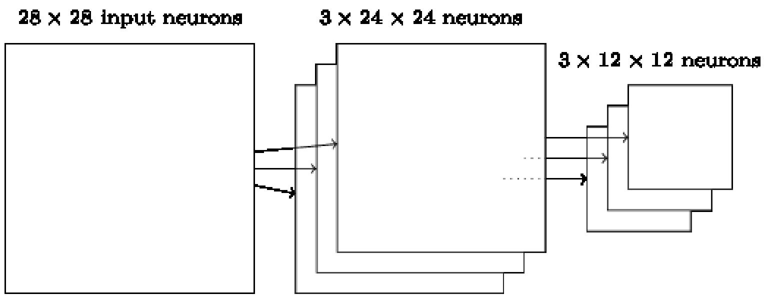


Figure 10.26: A convolutional network, consisting of 3 feature maps and pooling.

Types of pooling

1. **max-pooling**: simply outputs the maximum activation in the 2×2 input neurons.
2. L^2**-pooling**: outputs the L^2-average of the 2×2 input neurons.

   ...

10.5.2. CNNs, in practice

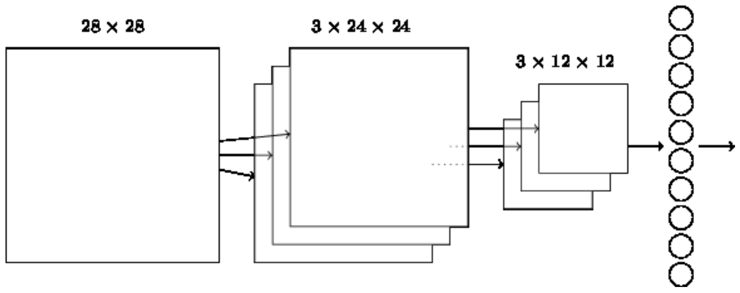


Figure 10.27: A *simple* CNN of three feature maps, to classify MNIST digits.

- To form a complete CNN by putting all these ideas, we need to ***add some extra layers***, below the convolution-pooling layers.
- Figure 10.27 shows a CNN that involves an extra layer of 10 output neurons, for the 10 possible values for MNIST digits ('0', '1', '2', etc.).
- The final layer of connections is a fully-connected layer. For example:
  - Let `filter_shape =(20, 1,5,5), poolsize =(2,2)`
    (20 feature maps; $1 \times 5 \times 5$ kernel; 2×2 pooling)
  - Then, the number of parameters to teach:
    $(5^2 + 1) \cdot 20 + (20 \cdot 12^2) \cdot 10 = 29,300.$
  - Classification accuracy for the MNIST data set $\lesssim$ 99%
- Add a **second convolution-pooling layer**:
  - Its input is the output of the first convolution-pooling layer.
  - Let `filter_shape =(40, 20,5,5), poolsize =(2,2)`
  - The output of the second convolution-pooling layer: $40 \times 5 \times 5$
  - Then, the number of parameters to teach:
    $(5^2 + 1) \cdot 20 + (5^2 + 1) \cdot 40 + (40 \cdot 5^2) \cdot 10 = 11,560$
  - Classification accuracy for the MNIST data set $\gtrsim$ 99%
- Add a **fully-connected layer** (up the output layer):
  - Let choose 40 neurons: $\Longrightarrow 41,960$ parameters
  - Classification accuracy $\approx$ 99.5%
- Use an ***ensemble of networks***
  - Using 5 CNNs, classification accuracy = 99.67% (**33 missed!**)

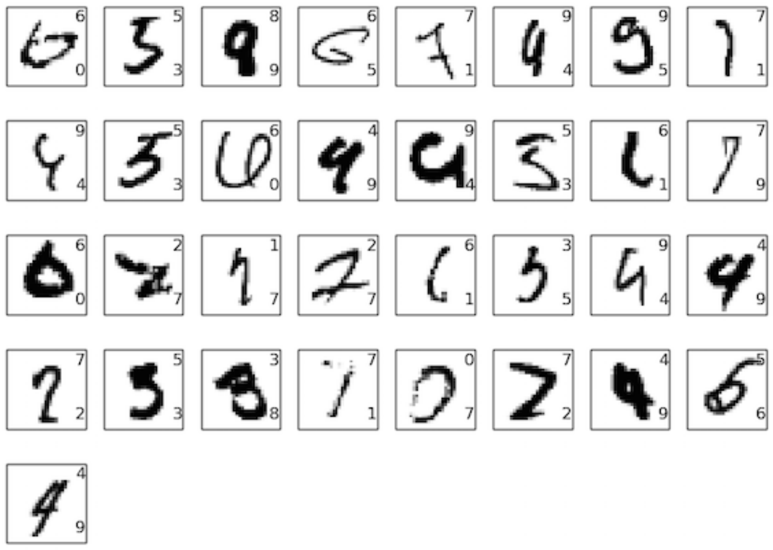


Figure 10.28: The images missed by an ensemble of 5 CNNs. The label in the top right is the correct classification, while in the bottom right is the label classified output.

Remarks 10.33. Intuitively speaking:

- **(Better representation).** The use of **translation invariance** by the convolutional layer will reduce the number of parameters it needs to get the same performance as the fully-connected model.
- **(Convolution kernels).** The filters try to detect **localized features**, producing **feature maps**.
- **(Efficiency). Pooling** simplifies the information in the output from the convolutional layer.
  - That, in turn, will result in **faster training** for the convolutional model, and, ultimately, will help us **build deep networks** using convolutional layers.

- **Fully-connected hidden layers** try to collect information for **more widely formed features**.

Exercises for Chapter 10

10.1. **(Designing a deep network).** First, download a CNN code (including the MNIST data set) by accessing to
https://github.com/mnielsen/neural-networks-and-deep-learning.git
or '`git clone`' it. In the 'src' directory, there are 8 python source files:

```
conv.py  mnist_average_darkness.py  mnist_svm.py  network2.py
expand_mnist.py  mnist_loader.py     network.py   network3.py
```

(a) Save `Run_network.py` in Section 10.4.2 in the 'src' directory to run. Edit to see how its performance changes.

CNN: On lines 16–22 in Figure 10.29 below, I put a design of a CNN model, which involved 2 hidden layers, one for a convolution-pooling layer and the other for a fully-connected layer. Its test accuracy becomes approximately 98.8% in 30 epochs.

(b) Set '`GPU = False`' in `network3.py`, if you are NOT using a GPU.
(Default: '`GPU = True`', set on line 50-some of network3.py)

(c) Modify `Run_network3.py` appropriately to design a CNN model as accurate as possible. Can your network achieve an accuracy better than 99.5%? ***Hint***: You may keep using the `SoftmaxLayer` for the final layer. **ReLU** (Rectified Linear Units) seems comparable with the sigmoid function (default) for activation. The default `p_dropout=0.0`. You should add some more layers, meaningful, and tune well all of hyperparameters: the number of feature maps for convolutional layers, the number of fully-connected neurons, η, `p_dropout`, etc..

Report your experiences.

Run_network3.py

```python
""" Run_network3.py:
   -------------------
    A CNN model, for the MNIST data set,
    which uses network3.py written by Michael Nielsen.
    The source code can be downloaded from
       https://github.com/mnielsen/neural-networks-and-deep-learning.git
    or 'git clone' it
"""
import network3
from network3 import Network, ReLU
from network3 import ConvPoolLayer, FullyConnectedLayer, SoftmaxLayer

training_data, validation_data, test_data = network3.load_data_shared()

mini_batch_size = 10
net = Network([
    ConvPoolLayer(image_shape=(mini_batch_size, 1, 28, 28),
                  filter_shape=(20, 1, 5, 5),
                  poolsize=(2, 2), activation_fn=ReLU),
    FullyConnectedLayer(
               n_in=20*12*12, n_out=100, activation_fn=ReLU, p_dropout=0.0),
    SoftmaxLayer(n_in=100, n_out=10, p_dropout=0.5)], mini_batch_size)

n_epochs, eta = 30, 0.1
net.SGD(training_data, n_epochs, mini_batch_size, eta, \
        validation_data, test_data)
```

Figure 10.29: Run_network3.py

Chapter 11

Boundary-Value Problems of One Variable

In This Chapter:

Contents of Chapter 11

11.1. The Shooting Method

Problem 11.1. The two-point **Boundary-Value Problems** (BVPs) involve a second-order differential equation of the form

$$\begin{gathered} y'' = f(x, y, y'), \quad a \le a \le b \\ y(a) = \alpha, \quad y(b) = \beta \end{gathered} \tag{11.1}$$

Theorem 11.2. (Existence and Uniqueness)
Let

$$D = \{(x, y, y') \mid a \le a \le b, \ \textit{with} \ -\infty < y < \infty, \ -\infty < y' < \infty\}.$$

For the BVP of the form in (11.1), 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 11.3. Show that the following BVP has a unique solution.

$$\begin{gathered} y'' + e^{-xy} + sin(y') = 0, \quad 1 \le x \le 2 \\ y(1) = 0, \ \ y(2) = 0 \end{gathered}$$

Solution.

Linear Boundary-Value Problems

Definition 11.4. 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,

$$y'' = f(x, y, y') = p(x)\, y' + q(x)\, y + r(x), \tag{11.2}$$

for some p, q, r.

Corollary 11.5. *Suppose the linear BVP*

$$\begin{array}{c} y'' = f(x, y, y') = p(x)\, y' + q(x)\, y + r(x), \quad a \le a \le b \\ y(a) = \alpha, \quad y(b) = \beta \end{array} \tag{11.3}$$

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 (11.3), we first solve the problem twice, with two different *initial conditions*, obtaining solutions y_1 and y_2, say,

$$\begin{cases} y_1(a) = \alpha, & y_1'(a) = z_1 \\ y_2(a) = \beta, & y_2'(a) = z_2 \end{cases} \tag{11.4}$$

   (We will choose z_1 and z_2 appropriately later.)

2. Then we form a linear combination of y_1 and y_2:

$$y(x) \;=\; \lambda y_1(x) + (1 - \lambda) y_2(x), \tag{11.5}$$

   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

$$y(b) \;=\; \lambda y_1(b) + (1 - \lambda) y_2(b) \;=\; \beta, \tag{11.6}$$

which reads

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)}, \tag{11.7}$$

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 11.6. If the linear BVP (11.3) has a solution , then either

$$\begin{cases} y \equiv y_1, & \textbf{or} \\ y_1(b) \neq y_2(b). \end{cases} \tag{11.8}$$

For the later case, the solution is a linear combination (11.5) with (11.7).

Summary 11.7. Summary of the Linear Shooting Method

1. Solve two BVPs for y_1 and y_2:

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha, \;\; y'(a) = 0 \end{cases} \qquad \begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha, \;\; y'(a) = 1 \end{cases} \tag{11.9}$$

where $f(x, y, y') = p(x)\, y' + q(x)\, y + r(x)$.

2. Compute

$$\lambda = \frac{\beta - y_2(b)}{y_1(b) - y_2(b)}, \tag{11.10}$$

3. Get the solution of the linear BVP (11.3) as a linear combination of y_1 and y_2:

$$y(x) = \lambda y_1(x) + (1 - \lambda) y_2(x). \tag{11.11}$$

Remark 11.8. 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:

$$\begin{cases} y' = u, & y(a) = \alpha \\ u' = p\,u + q\,y + r, & u(a) = 1 \end{cases} \tag{11.12}$$

which can be solved efficiently by using an ODE solver. See **Section 5.2, p. 179, and Section 5.3 p. 188.**

Here we state one of the most popular ODE solvers, the **Fourth-order Runge-Kutta method (RK4)**.

Algorithm 11.9. (RK4: Algorithm 5.19, p. 191)

$$y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \tag{11.13}$$

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

$$\frac{h^5}{5!}y^{(5)}(\xi_n), \tag{11.14}$$

for some $\xi_n \in [x_n, x_{n+1}]$. Thus the **global error** reads, for some $\xi \in [x_0, T]$,

$$\frac{(T - x_0)h^4}{5!}y^{(5)}(\xi). \tag{11.15}$$

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 11.10. Use the RK4SYSTEM to solve

$$\begin{cases} y'' - 2y' + 2y \;=\; e^{2x}\sin(x), & 0 \le x \le 1 \\ y(0) = -0.4, \quad y'(0) = -0.6 \end{cases} \tag{11.16}$$

Solution. The IVP (11.16) reads

$$\begin{cases} y' = u, & y(0) = -0.4 \\ u' = 2u - 2y + e^{2x}\sin(x), & u(0) = -0.6 \end{cases} \tag{11.17}$$

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 11.11. 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
```

11.2. Finite Difference Methods

Recall: A linear BVP (11.3) has the form

$$\begin{aligned} y'' &= p(x)\,y' + q(x)\,y + r(x), \quad a \le a \le b \\ y(a) &= \alpha, \quad y(b) = \beta \end{aligned} \tag{11.18}$$

In order to develop **finite difference methods (FDM)** for the solution of the problem in more general environments, we in this section consider differential equations of the following form

$$\begin{cases} -y'' + p(x)\,y' + q(x)\,y = f(x), \quad a \le a \le b \\ y(a) = \alpha, \quad y'(b) = \beta \end{cases} \tag{11.19}$$

Recall:

$\boxed{\textbf{Theorem}}$ **11.12.** ***(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 $x \in [a,b]$,*

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x-x_0)^k + \mathcal{R}_n(x), \quad \mathcal{R}_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1}, \tag{11.20}$$

for some ξ between x and x_0.

Remark 1.25: Alternative Form of Taylor's Theorem

Remark 11.13. Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b). Then, for every x, $x+h \in [a,b]$,

$$f(x+h) = \sum_{k=0}^{n} \frac{f^{(k)}(x)}{k!}h^k + \mathcal{R}_n(h), \quad \mathcal{R}_n(h) = \frac{f^{(n+1)}(\xi)}{(n+1)!}h^{n+1}, \tag{11.21}$$

for some ξ between x and $x+h$. In detail,

$$f(x+h) = f(x) + f'(x)h + \frac{f''(x)}{2!}h^2 + \frac{f'''(x)}{3!}h^3 + \cdots + \frac{f^{(n)}(x)}{n!}h^n + \mathcal{R}_n(h). \tag{11.22}$$

Note: The **FD discretization procedure** consists of four steps:

1. **Partitioning**
2. **FD approximations**
3. **System of difference equations**
4. **Assembly**

11.2.1. The FD discretization procedure

Step 1: Partitioning

Let, for some $n > 0$,

$$x_i = a + ih, \quad h = \frac{b-a}{n}, \quad i = 0, 1, \cdots, n. \tag{11.23}$$

Define $g_i = g(x_i)$, for $g = y, p, q, f$; for example,

$$y_i = y(x_i). \tag{11.24}$$

The objective with finite difference methods is to approximate $\{y_i\}_{i=0}^{n}$, the solution at **discrete nodal points**.

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

$$\begin{aligned}
\text{(a)}\quad y_{i+1} &= y(x_i) + y'(x_i)h + \frac{y''(x_i)}{2!}h^2 + \frac{y'''(x_i)}{3!}h^3 + \frac{y^{(4)}(x_i)}{4!}h^4 \\
&\quad + \frac{y^{(5)}(x_i)}{5!}h^5 + \frac{y^{(6)}(x_i)}{6!}h^6 + \cdots \\
\text{(b)}\quad y_{i-1} &= y(x_i) - y'(x_i)h + \frac{y''(x_i)}{2!}h^2 - \frac{y'''(x_i)}{3!}h^3 + \frac{y^{(4)}(x_i)}{4!}h^4 \\
&\quad - \frac{y^{(5)}(x_i)}{5!}h^5 + \frac{y^{(6)}(x_i)}{6!}h^6 - \cdots
\end{aligned} \tag{11.25}$$

- **Approximation of $y''(x_i)$**: Adding (11.25)(a) and (11.25)(b) gives

$$y_{i+1} + y_{i-1} = 2y_i + 2\frac{y''(x_i)}{2!}h^2 + 2\frac{y^{(4)}(x_i)}{4!}h^4 + 2\frac{y^{(6)}(x_i)}{6!}h^6 + \cdots \tag{11.26}$$

- Solving for $y''(x_i)$ reads

$$y''(x_i) = \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} - 2\frac{y^{(4)}(x_i)}{4!}h^2 - 2\frac{y^{(6)}(x_i)}{6!}h^4 - \cdots \tag{11.27}$$

- **Approximation of $y'(x_i)$**: Subtracting (11.25)(b) from (11.25)(a) gives

$$y_{i+1} - y_{i-1} = 2y'(x_i)h + 2\frac{y'''(x_i)}{3!}h^3 + 2\frac{y^{(5)}(x_i)}{5!}h^5 + \cdots \tag{11.28}$$

- Solving for $y'(x_i)$ reads

$$y'(x_i) = \frac{y_{i+1} - y_{i-1}}{2h} - \frac{y'''(x_i)}{3!}h^2 - \frac{y^{(5)}(x_i)}{5!}h^4 - \cdots \tag{11.29}$$

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
$$-\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + p_i \cdot \frac{y_{i+1} - y_{i-1}}{2h} + q_i \cdot y_i + K_2 h^2 + \mathcal{O}(h^4) = f_i, \quad (11.30)$$
where
$$K_2 = \frac{1}{12}\big(y^{(4)}(x_i) - 2p_i y'''(x_i)\big).$$
- Note that $y_0 = y(a) = \alpha$; (11.30) is defined only for $i = 1, 2, \cdots, n$.
- When $i = n$, (11.30) reads
$$\frac{-y_{n-1} + 2y_n - y_{n+1}}{h^2} + p_n \cdot \frac{y_{n+1} - y_{n-1}}{2h} + q_n \cdot y_n + K_2 h^2 + \mathcal{O}(h^4) = f_n, \quad (11.31)$$

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 $n = x_n$,
$$\frac{y_{n+1} - y_{n-1}}{2h} - \frac{y'''(b)}{3!} h^2 + \mathcal{O}(h^4) = \beta. \quad (11.32)$$

The **second-order FDM** for the model (11.19) follows from (11.30), (11.31), and (11.32), 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 (11.19) reads
$$\frac{-v_{i-1} + 2v_i - v_{i+1}}{h^2} + p_i \cdot \frac{v_{i+1} - v_{i-1}}{2h} + q_i \cdot v_i = f_i, \quad i = 1, 2, \cdots, n, \quad (11.33)$$
where $v_0 = \alpha$ and $v_{n+1} = v_{n-1} + 2h\,\beta$.

Step 4: Assembly

- **Assembly** is to build an algebraic system from (11.33).
- Multiplying (11.33) with h^2 and simplifying the result reads

$$-c_i v_{i-1} + (2 + h^2 q_i) v_i - d_i v_{i+1} \;=\; h^2 f_i, \quad i = 1, 2, \cdots, n, \tag{11.34}$$

where

$$c_i = 1 + \frac{hp_i}{2} \quad \text{and} \quad d_i = 1 - \frac{hp_i}{2}. \tag{11.35}$$

- 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.

$$\begin{array}{rcll} (2 + h^2 q_1) v_1 - d_1 v_2 & = & h^2 f_1 + c_1 \cdot \alpha, & i = 1 \\ -c_i v_{i-1} + (2 + h^2 q_i) v_i - d_i v_{i+1} & = & h^2 f_i, & i = 2, \cdots, n-1 \\ -2 v_{n-1} + (2 + h^2 q_n) v_n & = & h^2 f_n + d_n \cdot 2h\beta, & i = n \end{array} \tag{11.36}$$

- The corresponding **algebraic system** reads

$$A\mathbf{v} = \mathbf{b}, \tag{11.37}$$

where

$$A = \begin{bmatrix} 2 + h^2 q_1 & -d_1 & & & \\ -c_2 & 2 + h^2 q_2 & -d_2 & & \\ & \ddots & \ddots & \ddots & \\ & & -c_{n-1} & 2 + h^2 q_{n-1} & -d_{n-1} \\ & & & -2 & 2 + h^2 q_n \end{bmatrix},$$
$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} h^2 f_1 \\ h^2 f_2 \\ \vdots \\ h^2 f_{n-1} \\ h^2 f_n \end{bmatrix} + \begin{bmatrix} c_1 \cdot \alpha \\ 0 \\ \vdots \\ 0 \\ d_n \cdot 2h\beta \end{bmatrix}. \tag{11.38}$$

11.2.2. Existence and uniqueness of the FDM solution

Theorem 11.14. *Suppose that p, q and f are continuous on $[a,b]$. If*

$$0 \le q(x), \quad \forall\, x \in [a,b], \tag{11.39}$$

then the linear system (11.37) has unique solution provided that

$$c_i, \;\; d_i > 0, \quad i = 1, 2, \cdots, n. \tag{11.40}$$

Proof. When the two conditions are satisfied, the matrix A diagonally dominant, i.e.,

$$a_{ii} \ge \sum_{j \ne i} |a_{ij}| =: \Lambda_i, \quad \forall\, i = 1, 2, \cdots, n. \tag{11.41}$$

Indeed,

$$\begin{array}{rcl} a_{11} & = & 2 + h^2 q_1 \ge 2 > d_1 = \Lambda_1, \\ a_{ii} & = & 2 + h^2 q_i \ge 2 = \Lambda_i, \;\; i = 2, \cdots, n. \end{array} \tag{11.42}$$

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 11.15. Let $L = \max_{x \in [a,b]} |p(x)|$. Then the conditions in (11.40) are equivalent to

$$\frac{1}{2} hL < 1. \tag{11.43}$$

That is, h must be chosen **small enough** to satisfy the inequality.

Example 11.16. Use the second-order FDM to approximate the solution of

$$\begin{aligned} &y'' = \frac{1}{x}y' + \frac{3}{x^2}y - 4x^2, \quad x \in [1,2] \\ &y(1) = 0, \;\; y'(2) = -2, \end{aligned} \tag{11.44}$$

with $h = 1/5$ and $h = 1/10$.

Note: The differential equation can be written as

$$-y'' + \frac{1}{x}y' + \frac{3}{x^2}y = 4x^2, \quad x \in [1,2]. \tag{11.45}$$

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

$$y(x) = -8/(7*\mathrm{x}) + (68/35)*\mathrm{x}^3 - (4/5)*\mathrm{x}^4. \tag{11.46}$$

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
```

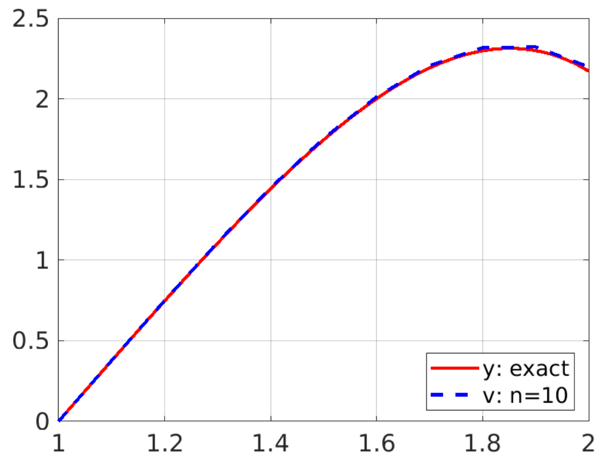
FDM_1D.m

```
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 11.17. 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. 141, for details.

11.3. Finite Element Methods

- To describe the **finite element method** (**FEM**), we consider approximating the solution to a linear two-point boundary-value problem of the form

$$\text{(D)}\quad \begin{cases} -\dfrac{d}{dx}\Big(p(x)\dfrac{du}{dx}\Big) = f(x), \quad x \in [0,1], \\ \qquad u(0)=0,\ u(1)=0, \end{cases} \tag{11.47}$$

  for which we assume $p \in C^1[0,1]$ and there is a constant $\delta > 0$ such that

$$p(x) \geq \delta > 0, \quad x \in [0,1]. \tag{11.48}$$

- The FEM is formulated with the so-called **variational principle**.

11.3.1. Variational formulation

DoiNg HeRe

11.3.2. Formulation of FEMs

11.4. FDMs for Non-constant Diffusion

CHAPTER 12

Numerical Solutions to Partial Differential Equations

In This Chapter:

Contents of Chapter 12

12.1. Parabolic PDEs in One Spatial Variable

12.2. Parabolic PDEs in Two Spatial Variables

12.3. Elliptic PDEs in Two Dimensions

Chapter 13

Projects

Finally we add projects.

Contents of Projects

P.1. Applications of Richardson Extrapolation for PDEs

Problem 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 two numerical solutions are available from 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.

- **The Issue**: The Richardson extrapolation can compute the solution of a higher-order accuracy **only on the course mesh**.
- Various **multiple coarse grid** (**MCG**) updating strategies have been considered in the literature [5].
- An MCG updating method can be expensive computationally.
- 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.**

Queisiton. Can we get higher-order solutions on finer meshes, without using a multiple coarse grid updating method?

What to do

Implement a code for Example P.2 and Algorithm P.6; report your code and results.

Example 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 $-u''(x_i)$ by 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:

$$A\mathbf{v}_h = \mathbf{b} \tag{P.1.4}$$

(d) Find $\mathbf{v}_h$, $\mathbf{v}_{h/2}$, and $\mathbf{v}_{h/4}$.

(e) Apply the Richardson Extrapolation and measure the errors.

Remark P.3. In Example P.2:

- 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.

Challenge P.4. Solutions of a sixth-order accuracy on the mid mesh of grid size $h/2$.

Remark P.5. Equation (P.1.2) can be rewritten as

$$u_i = \frac{u_{i-1}+u_{i+1}}{2} + \frac{h^2}{2}(f_i - K_2 h^2) - \frac{1}{2}K_4 h^6 - \cdots, \tag{P.1.5}$$

where

$$K_2 = \frac{u^{(4)}(x_i)}{12} \quad \text{and} \quad K_4 = \frac{u^{(6)}(x_i)}{360}. \tag{P.1.6}$$

Since $u^{(4)} = -f''$, for example, K_2 is approximated as

$$K_2 = \frac{u^{(4)}(x_i)}{12} = -\frac{1}{12}f''(x_i) = -\frac{1}{12}\frac{f_{i-1} - 2f_i + f_{i+1}}{h^2} + \mathcal{O}(h^2), \tag{P.1.7}$$

and therefore

$$f_i - K_2 h^2 = \frac{1}{12}(f_{i-1} + 10 f_i + f_{i+1}) + \mathcal{O}(h^4), \tag{P.1.8}$$

from which we have

$$u_i = \frac{u_{i-1}+u_{i+1}}{2} + \frac{h^2}{24}(f_{i-1} + 10 f_i + f_{i+1}) + \mathcal{O}(h^6). \tag{P.1.9}$$

$\boxed{\textbf{Algorithm}}$ **P.6.** In order to get a sixth-order solution on the mid mesh, we first should have *meaningful* fourth-order solutions on the mid and fine meshes.

1. Let $\mathbf{w}_h = \dfrac{1}{3}(4\mathbf{v}_{h/2} - \mathbf{v}_h)$, Richardson extrapolation on the course mesh.
2. Let $\widetilde{\mathbf{w}}_{h/2}$ be its expansion on the mid mesh: $\widetilde{\mathbf{w}}_{h/2}[0:2:\texttt{end}] = \mathbf{w}_h$.
3. Then, you should determine values $\widetilde{\mathbf{w}}_{h/2}[1:2:\texttt{end}]$ to complete the expansion. Using (P.1.9), if $i \in [1:2:2*\texttt{n}-1]$, we have

$$\widetilde{\mathbf{w}}_{h/2}(i) = \frac{\widetilde{\mathbf{w}}_{h/2}(i-1) + \widetilde{\mathbf{w}}_{h/2}(i+1)}{2} + \frac{(h/2)^2}{24}(f_{i-1} + 10f_i + f_{i+1}), \quad \textbf{(P.1.10)}$$

   where $f_i = f(i \cdot (h/2))$.
4. Repeat the above, starting with $\mathbf{w}_{h/2}$, to get its expansion $\widetilde{\mathbf{w}}_{h/4}$.
5. Then **a sixth-order solution on the mid mesh** is

$$\frac{1}{15}\big(16 * \widetilde{\mathbf{w}}_{h/4}[0:2:\texttt{end}] - \widetilde{\mathbf{w}}_{h/2}\big). \quad \textbf{(P.1.11)}$$

   Check the error.

How to find the accuracy order of a numerical scheme:

- For given h, you can measure the error: that is,

$$||\mathbf{u}_h - \mathbf{v}_h||_\infty = E_h. \tag{P.1.12}$$

- If the accuracy order is α, we may write

$$E_h = \mathcal{O}(h^\alpha) = C \cdot h^\alpha, \tag{P.1.13}$$

  for some constant $C > 0$.
- If you compute the numerical solution with $h/2$, then you can measure the error to be $E_{h/2}$, which can be written as

$$E_{h/2} = \mathcal{O}\big((h/2)^\alpha\big) = C \cdot (h/2)^\alpha, \tag{P.1.14}$$

- Dividing (P.1.13) by (P.1.14) reads

$$\frac{h^\alpha}{(h/2)^\alpha} = \frac{E_h}{E_{h/2}} \quad \Rightarrow \quad 2^\alpha = \frac{E_h}{E_{h/2}}.$$

- Now, applying the natural log to the equation results in

$$\alpha = \frac{\ln(E_h/E_{h/2})}{\ln 2} \tag{P.1.15}$$

If you use Matlab, you may start with

get_A_b.m
```
function [A,b] = get_A_b(interval,n,f,u)

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)=u(interval(1)); b(n+1)=u(interval(2));
```

Part of Richardson_extrapolation.m

```
u = @(x) sin(pi*x)+2*x;
f = @(x) pi^2*sin(pi*x);

interval=[0,1]; n0 = 10;
U = cell(3,1); V  = cell(3,1);
W = cell(2,2); WT = cell(3,1);

%%-- U & V
for i=1:3
    n = n0*2^(i-1); X =linspace(0,1,n+1)';
    U{i} = u(X);
    [A,b] = get_A_b(interval,n,f,u);
    V{i}= A\b;
end

%%-- Richardson
for i=1:2
    W{i,1} = (1/3)*(4*V{i+1}(1:2:end)-V{i});
end
W{1,2} = (1/15)*(16*W{2,1}(1:2:end)-W{1,1});

%%-- Expansion: WT{i}, i=2,3, from W{i-1,1}

```

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] 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.

[3] C. Cortes and V. N. Vapnik, *Support-vector networks*, Machine Learning, 20 (1995), pp. 273–297.

[4] G. Dahlquist, *A special stability problem for linear multistep methods*, BIT, 3 (1963), pp. 27–43.

[5] R. Dai, *Richardson extrapolation-based high accuracy high efficiency computation for partial differential equations*, Theses and Dissertations – Computer Science 20, University of Kentucky, Lexington, KY 40506, 2014.

[6] 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.

[7] S. Gerschgorin, *Über die abgrenzung der eigenwerte einer matrix*, Izv. Akad. Nauk SSSR Ser. Mat., 7 (1931), pp. 746–754.

[8] F. Golub and C. V. Loan, *Matrix Computations, 3rd Ed.*, The Johns Hopkins University Press, Baltimore, 1996.

[9] G. H. Golub and C. Reinsch, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403–420.

[10] B. Grosser and B. Lang, *An $\wr(n^2)$ algorithm for the bidiagonal svd*, Lin. Alg. Appl., 358 (2003), pp. 45–70.

[11] H. Hotelling, *Analysis of a complex of statistical variables into principal components*, Journal of Educational Psychology, 24 (1933), pp. 417–441 and 498–520.

[12] ——, *Relations between two sets of variates*, Biometrika, 28 (1936), pp. 321–377.

[13] C. JOHNSON, *Numerical Solutions of Partial Differential Equations by the Finite Element Method*, Cambridge University Press, New York, New Rochelle, Melbourne, Sydney, 1987.

[14] 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.

[15] C. KELLY, *Iterative methods for linear and nonlinear equations*, SIAM, Philadelphia, 1995.

[16] 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.

[17] P. C. NIEDFELDT AND R. W. BEARD, *Recursive ransac: multiple signal estimation with outliers*, IFAC Proceedings Volumes, 46 (2013), pp. 430–435.

[18] M. NIELSEN, *Neural networks and deep learning*. (The online book can be found at http://neuralnetworksanddeeplearning.com), 2013.

[19] K. PEARSON, *On lines and planes of closest fit to systems of points in space*, Philosophical Magazine, 2 (1901), pp. 559–572.

[20] F. ROSENBLATT, *The Perceptron: A probabilistic model for information storage and organization in the brain*, Psychological Review, (1958), pp. 65–386.

[21] 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.

[22] 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.

[23] O. TAUSSKY, *Bounds for characteristic roots of matrices*, Duke Math. J., 15 (1948), pp. 1043–1044.

[24] 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.

[25] R. VARGA, *Matrix Iterative Analysis, 2nd Ed.*, Springer-Verlag, Berlin, Heidelberg, 2000.

[26] 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