Numerical Analysis using Matlab and Maple

Lectures on YouTube:
https://www.youtube.com/channel/UCmRbK4vlGDht-joOQ5g0J2Q

Seongjai Kim

Department of Mathematics and Statistics

Mississippi State University

Mississippi State, MS 39762 USA

Email: skim@math.msstate.edu

Updated: June 26, 2021

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. Any questions, suggestions, comments will be deeply appreciated.

Seongjai Kim
June 26, 2021

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.

1.1. Review of Calculus

1.1.1. 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.

Answer: 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*

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

1.1.2. Differentiability

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.

$a := 0:$

$b := 2:$

$f := x \to x + \sin(x):$

$AS := \dfrac{f(b) - f(a)}{b - a} = 1 + \dfrac{1}{2}\sin(2)$

$evalf(\%) = 1.454648713$

$EQN := f'(x) = \dfrac{f(b) - f(a)}{b - a}$

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

$c := solve(EQN, x)$

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

$evalf(\%) = 1.098818559$

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

Figure 1.1: A **maple** implementation.

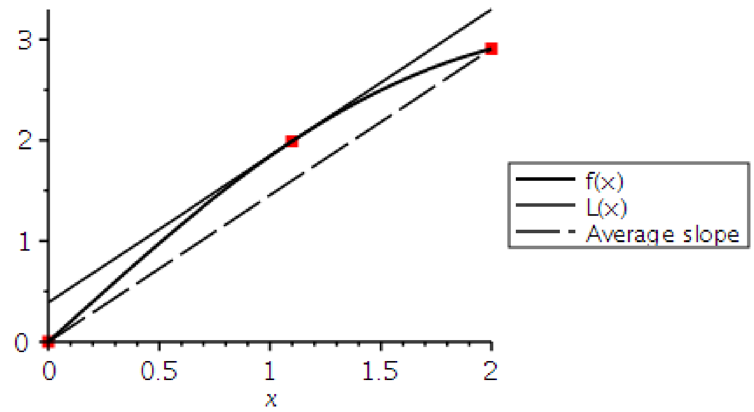


Figure 1.2: The resulting figure, from the implementation in Figure 1.1.

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

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

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

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

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

```

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]);
```

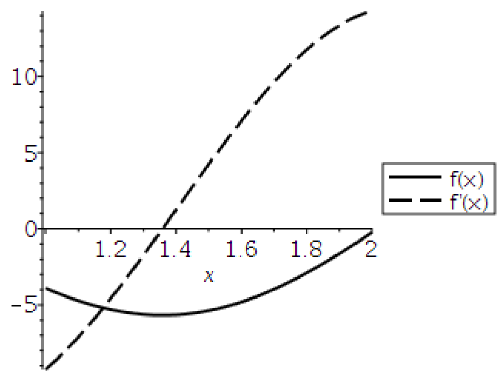


Figure 1.3: The figure from the Maple-code.

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

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 $f^{(n)}(c) = 0$.*

1.1.3. Integration

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

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

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. $F' = f$.*

Weighted Mean Value Theorem on Integral (WMVT)

Theorem 1.18. *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*

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

Remark 1.19. 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 $[a, b]$:

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

1.1.4. Taylor's Theorem

Theorem 1.20. ***(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 1.21. Let $f(x) = cos(x)$ and $x_0 = 0$. Determine the second and third Taylor polynomials for f about x_0.

Maple-code

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

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

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

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

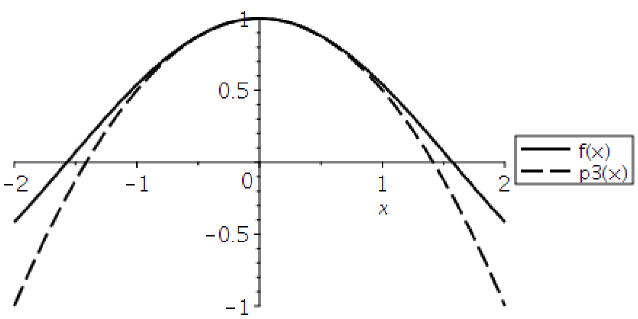


Figure 1.4: $f(x) = \cos x$ and its third Taylor polynomial $P_3(x)$.

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

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

which is the **Mean Value Theorem**.

Theorem 1.22. ***(Taylor's Theorem with Integral Remainder)***: *Suppose* $f \in C^n[a,b]$ *and* $x_0 \in [a,b]$. *Then, for every* $x \in [a,b]$,

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

where

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

Alternative Form of Taylor's Theorem

Remark 1.23. 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,

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

Theorem 1.24. ***(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*

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

where

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

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 1.25. 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. Vectors

Definition 1.26. Let $\mathbf{u} = [u_1, u_2, \cdots, u_n]^T$ and $\mathbf{v} = [v_1, v_2, \cdots, v_n]^T$ are vectors in $\mathbb{R}^n$. Then, the **inner product** (or **dot product**) of $\mathbf{u}$ and $\mathbf{v}$ is given by

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

Definition 1.27. The **length** (**Euclidean norm**) of $\mathbf{v}$ is nonnegative scalar $\|\mathbf{v}\|$ defined by

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

Definition 1.28. For $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$, the **distance** between $\mathbf{u}$ and $\mathbf{v}$ is

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

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

Example 1.29. Let $\mathbf{u} = \begin{bmatrix} 1 \\ -2 \\ 2 \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} 3 \\ 2 \\ -4 \end{bmatrix}$. Find $\mathbf{u} \bullet \mathbf{v}$, $\|\mathbf{u}\|$, and $dist(\mathbf{u}, \mathbf{v})$.

Solution.

Definition 1.30. Two vectors $\mathbf{u}$ and $\mathbf{v}$ in $\mathbb{R}^n$ are **orthogonal** if $\mathbf{u}\bullet\mathbf{v} = 0$.

Theorem 1.31. ***Pythagorean Theorem**: Two vectors $\mathbf{u}$ and $\mathbf{v}$ are orthogonal if and only if*

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

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

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

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

Example 1.32. Let $\mathbf{u} = \begin{bmatrix} 1 \\ \sqrt{3} \end{bmatrix}$ and $\mathbf{v} = \begin{bmatrix} -1/2 \\ \sqrt{3}/2 \end{bmatrix}$. Use (1.21) to find the angle between $\mathbf{u}$ and $\mathbf{v}$.

Solution.

1.2.2. 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 1.33. 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 1.34. 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 1.35. Find the parabola $y = a_0 + a_1 x + a_2 x^2$ that passes through $(1, 1)$, $(2, 2)$, and $(3, 5)$.

Solution.

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

1.2.3. Invertible (nonsingular) matrices

Definition 1.36. 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 1.37. (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 1.38. *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 1.39. Find the inverse of $A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 3 \\ 4 & -3 & 8 \end{bmatrix}$, if it exists.

Solution.

Theorem 1.40.

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 1.41. (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. $\det A \neq 0$.

1.2.4. Determinants

Definition 1.42. 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 1.43. 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.

Answer: c. 12

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

Definition 1.44. Let A_{ij} be the *submatrix* of A obtained by deleting row i and column j of A. Then the (i,j)-**cofactor** of $A = [a_{ij}]$ is the scalar C_{ij}, given by

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

Definition 1.45. For $n \geq 2$, the **determinant** of an $n \times n$ matrix $A = [a_{ij}]$ is given by the following formulas:

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

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

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

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

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

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

Example 1.46. 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.

Answer: -2

1.2.5. Eigenvectors and eigenvalues

Definition 1.47. Let A be an $n \times n$ matrix. An **eigenvector** of A is a *nonzero* vector $\mathbf{x}$ such that $A\mathbf{x} = \lambda\mathbf{x}$ for some scalar λ. In this case, a scalar λ is an **eigenvalue** and $\mathbf{x}$ is the *corresponding* **eigenvector**.

Definition 1.48. 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 1.49. Find the characteristic polynomial and all eigenvalues of
$$A = \begin{bmatrix} 1 & 1 & 0 \\ 6 & 0 & 5 \\ 0 & 0 & 2 \end{bmatrix}$$

Solution.

Remark 1.50. Let A be an $n \times n$ matrix. Then the characteristic equation of A is of the form

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

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

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

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

Theorem 1.51. *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,

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

  which is a contradiction.

□

1.2.6. Vector and matrix norms

Definition 1.52. A **norm** (or, **vector norm**) on $\mathbb{R}^n$ is a function that assigns to each $x \in \mathbb{R}^n$ a nonnegative real number $\|x\|$ such that the following three properties are satisfied: for all $\mathbf{x}$, $\mathbf{y} \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$,

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

Example 1.53. The most common norms are

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

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

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

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

Note: Two of frequently used p-norms are

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

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

Example 1.54. One may consider the infinity-norm as the limit of p-norms, as $p \to \infty$.

Solution.

Definition 1.55. 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 1.56. $\|A\|_F \equiv \Big(\sum_{i,j} |a_{ij}|^2\Big)^{1/2}$ is called the **Frobenius norm**.

Definition 1.57. *Once a vector norm* $\|\cdot\|$ *has been specified, the* ***induced matrix norm*** *is defined by*

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

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

Theorem 1.58.

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

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

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

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

d. $\|A\|_2 \equiv \max_{\mathbf{x} \ne 0} \frac{\|A\mathbf{x}\|_2}{\|\mathbf{x}\|_2} = \sqrt{\lambda_{\max}(A^T A)}$,
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 1.59. *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 1.60. Let $A = \begin{bmatrix} 1 & 2 & -2 \\ 0 & 4 & 1 \\ 1 & -2 & 2 \end{bmatrix}$. Then, we have

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

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 1.61. 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 1.62. The number p^* is said to approximate p to **t-significant digits** (or figures) if t is the largest nonnegative integer for which

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

1.3.1. Computational algorithms

Definition 1.63. 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 1.64. 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 1.65. (**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 1.66. 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 1.67. 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 1.68.

- 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

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

  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

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

  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 1.69. 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 1.70. 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 1.71. 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 1.72. 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 1.73. 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.

Answer: $k = 3$.

Self-study 1.74. Let $f(h) = \dfrac{1}{h}(1 + h - e^h)$. What are the limit and the rate of convergence of $f(h)$ as $h \to 0$?

Solution.

Self-study 1.75. 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. $\sin x \cos x = o(1)$, as $x \to 0$

Solution.

Example 1.76. 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'
```

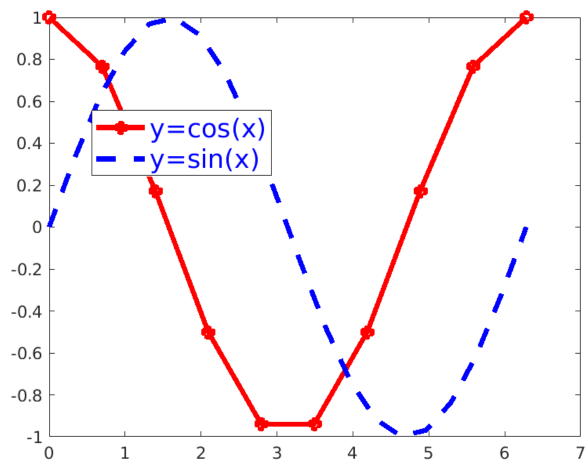


Figure 1.5: fig_cos_sin.png: `plot` of $y = \cos x$ and $y = \sin x$.

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

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

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

(b) Use the Taylor's Theorem (Theorem 1.20) 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.

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

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*

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

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

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 22.

***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 Equations in One Variable

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.

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 IVP).
3) There is a single solution in $[a, b]$.

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) · 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)
```

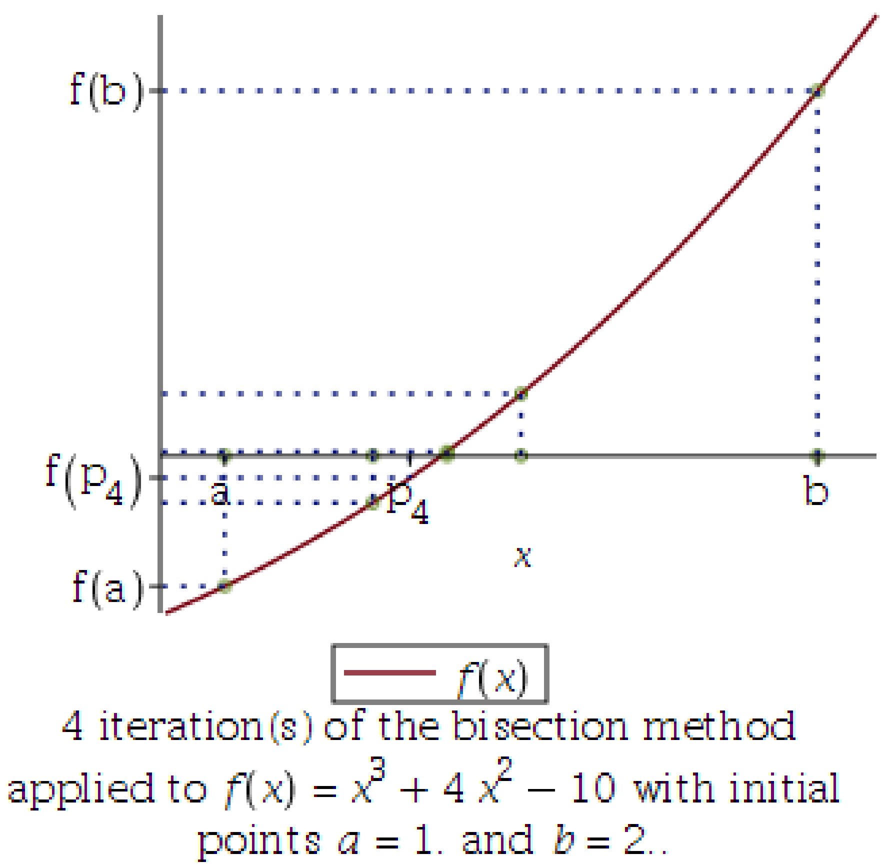


Figure 2.1: The bisection method.

```
> 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

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

Answer: $n = 10$

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.

Answer: $n \ge \ln(10^8/3)/\ln(2)$. 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.

Answer: no

2.2. Fixed-Point Iteration

Definition 2.9. A number p is a **fixed point** for a given function 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.

Remark 2.10. 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**.

Example 2.11. Find fixed points of $g(x) = x^2 - 2$.

Solution.

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

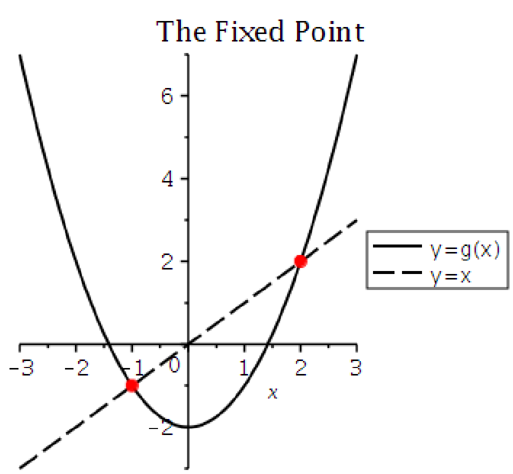


Figure 2.3: Fixed points of $g(x) = x^2 - 2$.

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*

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

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

Proof.

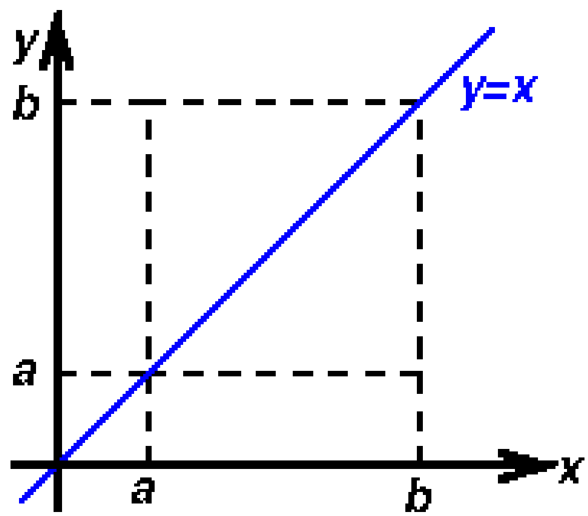


Figure 2.4: Illustration of the existence-and-uniqueness theorem.

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

Solution.

2.2.2. Fixed-point iteration

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

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

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

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

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

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

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

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

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

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

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

(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
```

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

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

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 $p \in [a,b]$.

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.

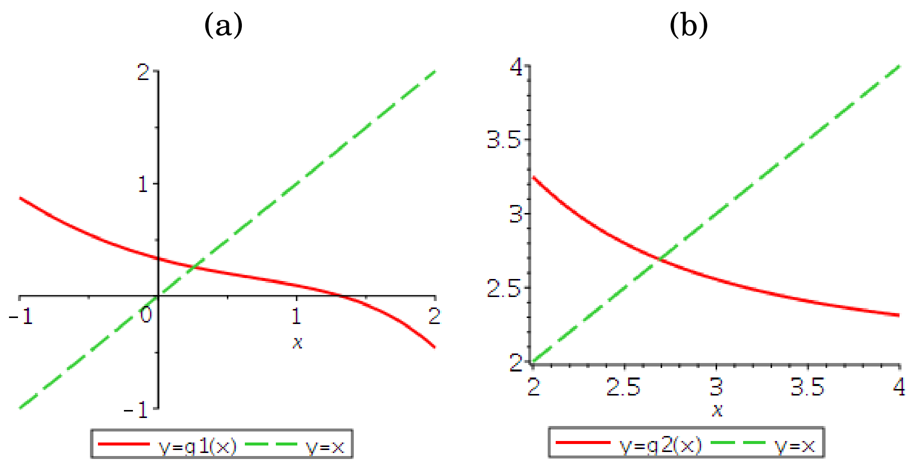


Figure 2.5: Visualization of the functions.

Answer: (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*** $\mathbf{f(x) = 0}$***).*** *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).

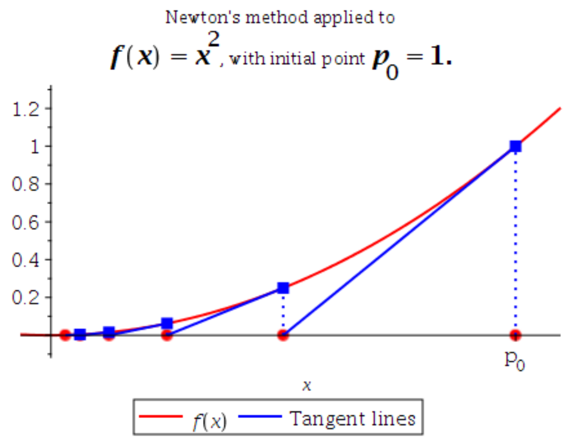


Figure 2.6: Graphical interpretation of the Newton's method.

An Example of Divergence

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

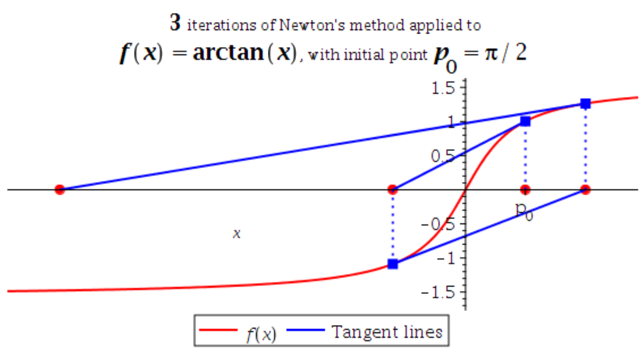


Figure 2.7: An example of divergence.

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. 27.)

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,*

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

can be expressed as

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

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

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

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

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

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

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

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

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

Solution.

Procedure NewtonRaphsonSYS.mw

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

Result

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

2.3.3. The secant method

Recall: The Newton's method, defined as

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

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

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

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

Thus, the resulting algorithm reads

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

Note:

- Two initial values (p_0, p_1) must be given, which however is not a drawback.
- It requires only one new evaluation of f per step.
- The graphical interpretation of the secant method is similar to that of Newton's method.
- 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`.

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);
```

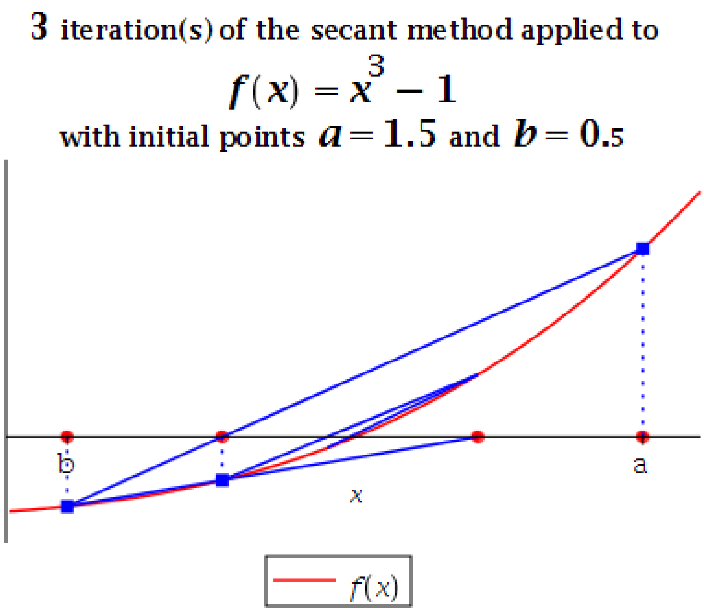


Figure 2.8: Graphical interpretation of the secant method.

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

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

Solution.

Answer: $p_2 = 5.0$

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

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

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

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

  else, set

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

  endif

Graphical interpretation

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

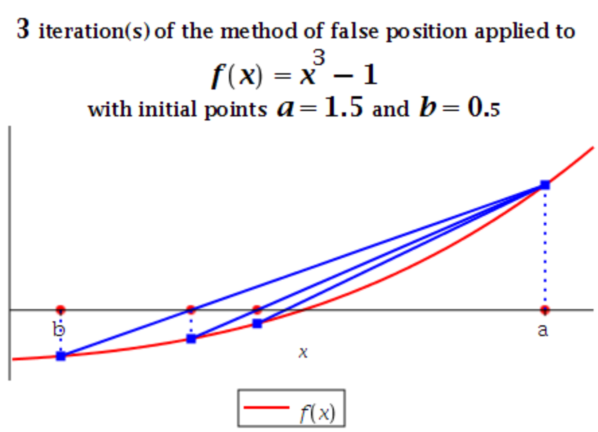


Figure 2.9: Graphical interpretation of the false position method.

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

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

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

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

- 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}	$\cdots$	$P(x_0) = b_0$

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

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

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

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

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

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

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

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

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

How to evaluate $P'(x)$: The derivative $P'(x)$ can be evaluated by using the Horner's method with the same efficiency. Indeed, differentiating (2.50) reads

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

Thus

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

That is, 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:

	1	−4	7	−5	−2
3		3	−3	12	21
	1	−1	4	7	$19 = P(3)$
3		3	6	30	
	1	2	10	$37 = Q(3) = P'(3)$	

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)
  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)
  newton_horner: x0=3; x=2, in 7 iterations
```

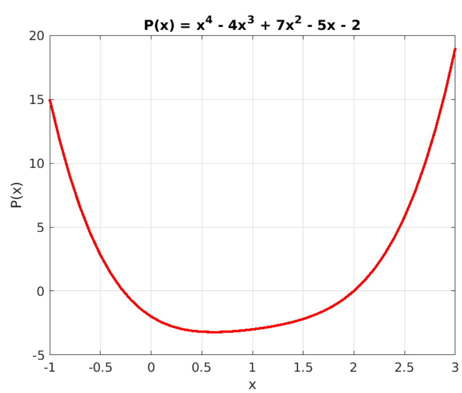


Figure 2.10: Polynomial $P(x) = x^4 - 4x^3 + 7x^2 - 5x - 2$. Its two zeros are -0.275682 and 2.

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

2.4.3. Bairstow's method

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

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

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

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

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

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

  Here J is the **Jacobian matrix**.

Question: How to compute the Jacobian matrix

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 2.11: Bairstow's method.

Run Bairstow

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

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

Deflation

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

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

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

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

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

Remark 2.43.

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

Exercises for Chapter 2

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

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

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

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

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

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

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

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

Consider the following equations defined on the given intervals:

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

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

For each of the above equations,

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

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

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

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

(a) Introduce two different fixed-point forms which are convergent for $x \in [1, 2]$.

(b) Perform five iterations for each of the iterations with $p_0 = 1.5$, and measure $|p - p_5|$.

(c) Rank the associated iterations based on their apparent speed of convergence with $p_0 = 1.5$. Discuss why one is better than the other.

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

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

(a) Show that for each $y \in [0, \pi]$, there exists an x satisfying the equation.

(b) Interpret this as a fixed-point problem.

(c) C Find x's for $y = 1, \pi/2, 2$, using the fixed-point iteration. Set $\varepsilon = 1/2$.

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

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

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

Find C and s such that

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

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

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

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

Hint: *Define* $f_1(x, y) = 4x^2 - y^2$, $f_2(x, y) = 4xy^2 - x - 1$. *Then try to use (2.37)-(2.38), p.59. Note that* $J(x, y) = \begin{bmatrix} 8x & -2y \\ 4y^2 - 1 & 8xy \end{bmatrix}$. *Thus, for example,* $J(x_0, y_0) = \begin{bmatrix} 0 & -2 \\ 3 & 0 \end{bmatrix}$ *and* $\begin{bmatrix} f_1(x_0, y_0) \\ f_2(x_0, y_0) \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$. *Now you can find the correction vector to update iterate and get* $\mathbf{x}_1$. *Do it once more for* $\mathbf{x}_2$.

2.7. C Consider the polynomial

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

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

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

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

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

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
Rational interpolation	Interpolation of rough data with minimum oscillation

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*

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

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

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

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

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

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

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

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

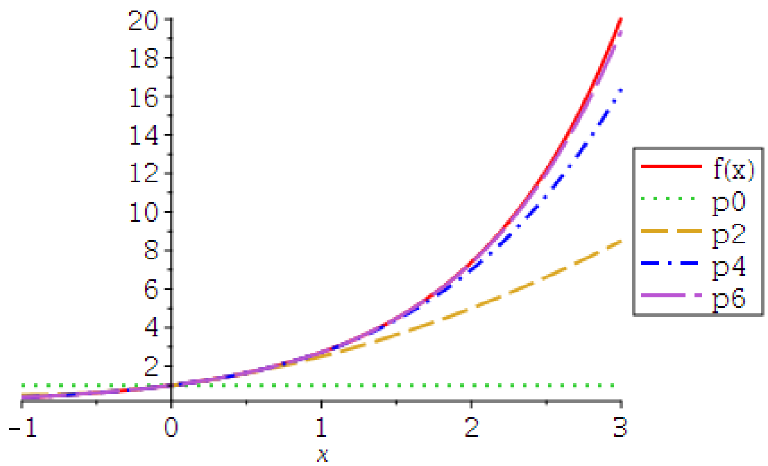


Figure 3.1: Polynomial approximations for $f(x) = e^x$.

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

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

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

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

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

$$p_n \equiv q_n.$$

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

- For $n = 0$, the existence is obvious since we may choose the constant function
$$p_0(x) = y_0. \tag{3.4}$$
- Now suppose that we have obtained a polynomial p_{k-1} of degree $\le k-1$ with
$$p_{k-1}(x_i) = y_i, \quad \text{for } 0 \le i \le k-1. \tag{3.5}$$
- We try to construct p_k in the form
$$p_k(x) = p_{k-1}(x) + c_k(x - x_0)(x - x_1)\cdots(x - x_{k-1}) \tag{3.6}$$
for some c_k.
  (a) Note that (3.6) is unquestionably a polynomial of degree $\le k$.
  (b) Furthermore, p_k interpolates the data that p_{k-1} interpolates:
$$p_k(x_i) = p_{k-1}(x_i) = y_i, \quad 0 \le i \le k-1. \tag{3.7}$$

- Now we determine the constant c_k to satisfy the condition

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

which leads to

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

This equation can certainly be solved for c_k:

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

because the denominator is not zero. (Why?)

□

3.1.1. Newton form of the interpolating polynomials

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

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

The compact form of this reads

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

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

The first few cases of (3.12) are

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

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

Illustration of Newton's interpolating polynomials

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

A Maple implementation

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

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

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

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

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

```

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

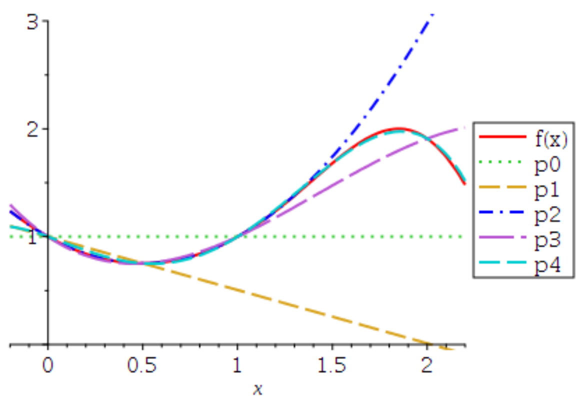


Figure 3.2: Illustration of Newton's interpolating polynomials, with $f(x) = \sin(x \cdot (x-1)) + 1$, at [0, 0.5, 1.0, 2.0, 1.5].

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

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

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

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

The computation of c_k, using Horner's algorithm

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

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

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

Example 3.7. Let

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

Four values of this function are given as

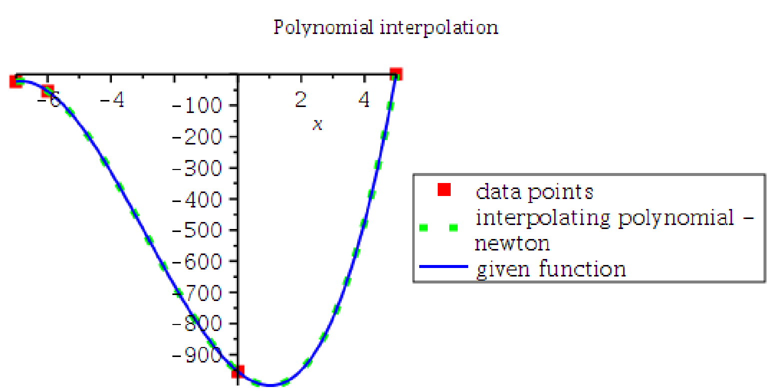
x_i	5	-7	-6	0
y_i	1	-23	-54	-954

Construct the Newton form of the polynomial from the data.

Solution.

Maple-code

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



```
DividedDifferenceTable(N);
```

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

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

x_i	2	-1	1
y_i	1	4	-2

Solution.

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

3.1.2. Lagrange Form of Interpolating Polynomials

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

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

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

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

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

- Then,

$$p_n(x_j) = \sum_{k=0}^{n} y_k L_{n,k}(x_j) = L_{n,i}(x_j). \tag{3.17}$$

- On the other hand, the polynomial p_n interpolating the data must satisfy $p_n(x_j) = \delta_{ij}$, where δ_{ij} is the ***Kronecker delta***

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

- Thus all the basis polynomials must satisfy

$$L_{n,i}(x_j) = \delta_{ij}, \quad \text{for all } 0 \le i, j \le n. \tag{3.18}$$

  Polynomials satisfying such a property are known as the **cardinal functions**.

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

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

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

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

and therefore

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

Hence, we have

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

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

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

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

x	x_0	x_1
y	y_0	y_1

Solution.

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

Solution.

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

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

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

Solution.

Maple-code

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

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

3.1.3. Polynomial interpolation error

Theorem 3.15. ***(Polynomial Interpolation Error Theorem).*** *Let $f \in C^{n+1}[a,b]$, and let P_n be the polynomial of degree $\le n$ 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 1.20.** (**Taylor's Theorem with Lagrange Remainder**), page 8. Suppose $f \in C^n[a,b]$, $f^{(n+1)}$ exists on (a,b), and $x_0 \in [a,b]$. Then, for every $x \in [a,b]$,

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

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

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

Answer: $n = 10$

3.1.4. Chebyshev polynomials

In the Polynomial Interpolation Error Theorem, **there is a term that can be optimized 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:

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

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

T6 := x -> simplify(ChebyshevT(6, x)): T6(x)
                          6       4       2
                      32 x  - 48 x  + 18 x  - 1
```

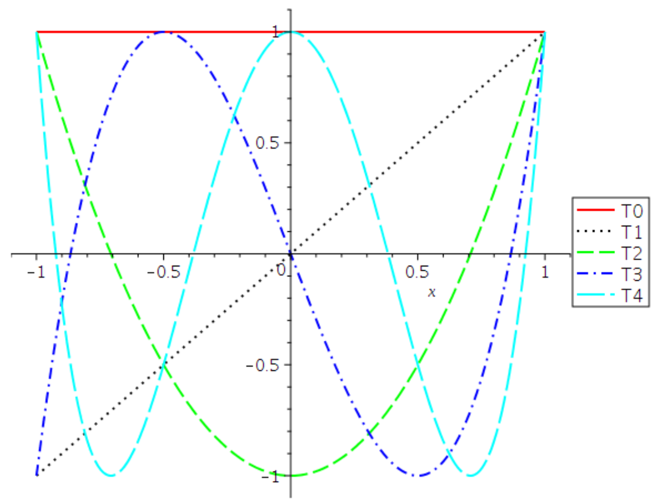


Figure 3.3: Chebyshev polynomials.

Theorem 3.21. (Properties of Chebyshev polynomials)*:*

(a) For $x \in [-1, 1]$*, the Chebyshev polynomials have this closed-form expression:*

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

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

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

and its minimum value will be attained if

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

(c) 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*

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

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

Solution. From Example 3.17, we know that

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

Thus

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

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

Accuracy comparison between uniform nodes and Chebyshev nodes:

Maple-code

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

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

```

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

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

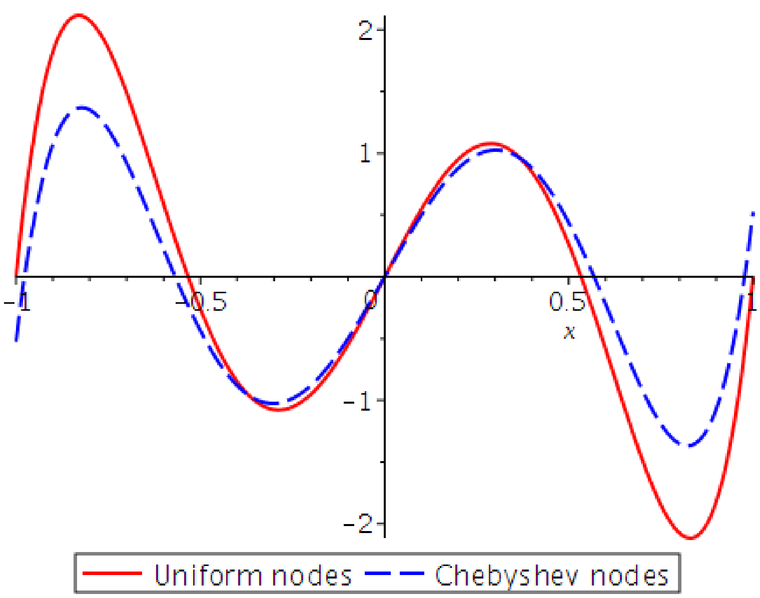


Figure 3.4: Accuracy comparison between uniform nodes and Chebyshev nodes.

3.2. Divided Differences

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

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

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

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

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

(a) The coefficient c_1 is determined to satisfy

$$y_1 = p_1(x_1) = c_0 + c_1(x_1 - x_0). \tag{3.41}$$

Note $c_0 = y_0$. Thus, we have

$$y_1 - y_0 = c_1(x_1 - x_0) \tag{3.42}$$

and therefore

$$c_1 = \frac{y_1 - y_0}{x_1 - x_0}. \tag{3.43}$$

(b) Now, since

$$y_2 = p_2(x_2) = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1),$$

it follows from the above, (3.42), and (3.43) that

$$\begin{aligned} c_2 &= \frac{y_2 - y_0 - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(y_2 - y_1) + (y_1 - y_0) - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} \\ &= \frac{(y_2 - y_1) + c_1(x_1 - x_0) - c_1(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)} = \frac{(y_2 - y_1)/(x_2 - x_1) - c_1}{x_2 - x_0}. \end{aligned} \tag{3.44}$$

Definition 3.25. (Divided differences):

- The **zeroth divided difference** of the function f with respect to x_i, denoted $f[x_i]$, is the value of at x_i:

$$f[x_i] = f(x_i) \tag{3.45}$$

- The remaining divided differences are defined recursively. The **first divided difference** of f with respect to x_i, x_{i+1} is defined as

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}. \tag{3.46}$$

- The **second divided difference** relative to x_i, x_{i+1}, x_{i+2} is defined as

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}. \tag{3.47}$$

- In general, the k**th divided difference** relative to $x_i, x_{i+1}, \cdots, x_{i+k}$ is defined as

$$f[x_i, x_{i+1}, \cdots, x_{i+k}] = \frac{f[x_{i+1}, \cdots, x_{i+k}] - f[x_i, \cdots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{3.48}$$

Note: It follows from Remark 3.24 that the coefficients of the Newton interpolating polynomials read

$$c_0 = f[x_0], \quad c_1 = f[x_0, x_1], \quad c_2 = f[x_0, x_1, x_2]. \tag{3.49}$$

In general,

$$c_k = f[x_0, x_1, \cdots, x_k]. \tag{3.50}$$

Newton's Divided Difference Table

x	f[x]	DD1 (f[,])	DD2 (f[, ,])	DD3 (f[, , ,])
x_0	$f[x_0]$			
x_1	$f[x_1]$	$f[x_0, x_1]$ $= \frac{f[x_1]-f[x_0]}{x_1-x_0}$		
x_2	$f[x_2]$	$f[x_1, x_2]$ $= \frac{f[x_2]-f[x_1]}{x_2-x_1}$	$f[x_0, x_1, x_2]$ $= \frac{f[x_1,x_2]-f[x_0,x_1]}{x_2-x_0}$	
x_3	$f[x_3]$	$f[x_2, x_3]$ $= \frac{f[x_3]-f[x_2]}{x_3-x_2}$	$f[x_1, x_2, x_3]$ $= \frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$	$f[x_0, x_1, x_2, x_3]$ $= \frac{f[x_1,x_2,x_3]-f[x_0,x_1,x_2]}{x_3-x_0}$

(3.51)

Pseudocode 3.26. (**Newton's Divided Difference Formula**):

Input: (x_i, y_i), $i = 0, 1, \cdots, n$, saved as $F_{i,0} = y_i$
Output: $F_{i,i}$, $i = 0, 1, \cdots, n$
Step 1: For $i = 1, 2, \cdots, n$
For $j = 1, 2, \cdots, i$

$$F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$$

Step 2: Return $(F_{0,0}, F_{1,1}, \cdots, F_{n,n})$

Example 3.27. Determine the Newton interpolating polynomial for the data:

x	0	1	2	5
y	1	-1	3	-189

Solution.

Answer: $f(x) = 1 - 2x + 3x(x - 1) - 4x(x - 1)(x - 2)$

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 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 $0 \le i < j \le n$,*

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

which is the polynomial interpolating f at $x_i, x_{i+1}, \cdots, x_j$.

Note: The above theorem implies that the interpolating polynomial can be generated recursively. For example,

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

and so on. They are generated in the manner shown in the following table, where each row is completed before the succeeding rows are begun.

x_0	$y_0 = P_0$			
x_1	$y_1 = P_1$	$P_{0,1}$		
x_2	$y_2 = P_2$	$P_{1,2}$	$P_{0,1,2}$	
x_3	$y_3 = P_3$	$P_{2,3}$	$P_{1,2,3}$	$P_{0,1,2,3}$

(3.62)

For simplicity in computation, we may try to avoid multiple subscripts by defining the new variable

$$Q_{i,j} = P_{i-j,i-j+1,\cdots,i}$$

Then the above table can be expressed as

x_0	$P_0 = Q_{0,0}$			
x_1	$P_1 = Q_{1,0}$	$P_{0,1} = Q_{1,1}$		
x_2	$P_2 = Q_{2,0}$	$P_{1,2} = Q_{2,1}$	$P_{0,1,2} = Q_{2,2}$	
x_3	$P_3 = Q_{3,0}$	$P_{2,3} = Q_{3,1}$	$P_{1,2,3} = Q_{3,2}$	$P_{0,1,2,3} = Q_{3,3}$

(3.63)

Example 3.34. Let $x_0 = 2.0$, $x_1 = 2.2$, $x_2 = 2.3$, $x_3 = 1.9$, $x_4 = 2.15$. Use Neville's method to approximate $f(2.1) = \ln(2.1)$ in a four-digit accuracy.

Solution.

Maple-code
```
with(Student[NumericalAnalysis]):
x0 := 2.0:
x1 := 2.2:
x2 := 2.3:
x3 := 1.9:
x4 := 2.15:
xy := [[x0,ln(x0)], [x1,ln(x1)], [x2,ln(x2)], [x3,ln(x3)], [x4,ln(x4)]]:
P := PolynomialInterpolation(xy, method = neville):

Q := NevilleTable(P, 2.1)
  [[0.6931471806, 0,            0,            0,            0           ],
   [0.7884573604, 0.7408022680, 0,            0,            0           ],
   [0.8329091229, 0.7440056025, 0.7418700461, 0,            0           ],
   [0.6418538862, 0.7373815030, 0.7417975693, 0.7419425227, 0           ],
   [0.7654678421, 0.7407450500, 0.7418662324, 0.7419348958, 0.7419374382]]
```

Note that

$$\begin{aligned} |Q_{3,3} - Q_{2,2}| &= |0.7419425227 - 0.7418700461| = 0.0000724766 \\ |Q_{4,4} - Q_{3,3}| &= |0.7419374382 - 0.7419425227| = 0.0000050845 \end{aligned}$$

Thus $Q_{3,3} = 0.7419425227$ is already in a four-digit accuracy.

Check: The real value is $\ln(2.1) = 0.7419373447$. The absolute error: $|\ln(2.1) - Q_{3,3}| == 0.0000051780$. □

Pseudocode 3.35.

Input: $\begin{cases} \text{the nodes } x_0, x_1, \cdots, x_n; \text{the evaluation point } x; \text{the tolerance } \varepsilon; \\ \text{and values } y_0, y_1, \cdots, y_n \text{ in the 1st column of } Q \in \mathbb{R}^{(n+1)\times(n+1)} \end{cases}$

Output: Q

Step 1: For $i = 1, 2, \cdots, n$

For $j = 1, 2, \cdots, i$

$$Q_{i,j} = \frac{(x - x_{i-j})Q_{i,j-1} - (x - x_i)Q_{i-1,j-1}}{x_i - x_{i-j}}$$

if $(|Q_{i,i} - Q_{i-1,i-1}| < \varepsilon)$ $\{i_0 = i;$ break;$\}$

Step 2: Return $(Q,\ i_0)$

Example 3.36. Neville's method is used to approximate $f(0.3)$, giving the following table.

$x_0 = 0$	$Q_{0,0} = 1$			
$x_1 = 0.25$	$Q_{1,0} = 2$	$Q_{1,1} = 2.2$		
$x_2 = 0.5$	$\mathbf{Q_{2,0}}$	$Q_{2,1}$	$Q_{2,2}$	
$x_3 = 0.75$	$Q_{3,0} = 5$	$Q_{3,1}$	$Q_{3,2} = 2.12$	$Q_{3,3} = 2.168$

(3.64)

Determine $Q_{2,0} = f(x_2)$.

Solution.

Answer: $Q_{2,2} = 2.2$; $Q_{2,1} = 2.2$; $Q_{2,0} = 3$

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

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

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

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

Example 3.41. (Continuation): Find a quartic polynomial p_4 that takes values given in the preceding example and, in addition, satisfies $p_4(2) = 44$.

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

$$|f(x) - P(x)| < \varepsilon, \quad \text{for all } x \in [a,b]. \tag{3.77}$$

Interpolation at equidistant points is a natural and common approach to construct approximating polynomials. **Runge's phenomenon** demonstrates, however, that interpolation can easily result in divergent approximations.

Example 3.42. (**Runge's phenomenon**): Consider the function

$$f(x) = \frac{1}{1+25x^2}, \quad x \in [-1,1]. \tag{3.78}$$

Runge found that if this function was interpolated at equidistant points

$$x_i = -1 + i\,\frac{2}{n}, \quad i = 0, 1, \cdots, n,$$

the resulting interpolation p_n oscillated toward the end of the interval, i.e. close to -1 and 1. It can even be proven that the interpolation error tends toward infinity when the degree of the polynomial increases:

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

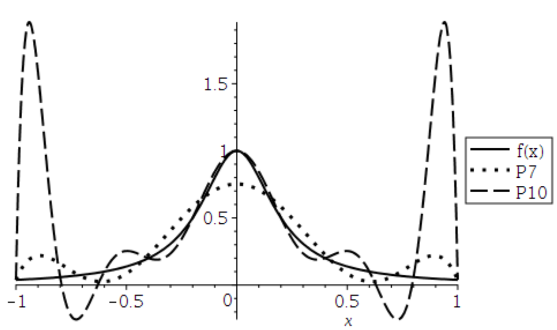


Figure 3.5: Runge's phenomenon.

Mitigation to the problem

- **Change of interpolation points**: e.g., Chebyshev nodes
- **Constrained minimization**: e.g., Hermite-like higher-order polynomial interpolation, whose first (or second) derivative has minimal norm.
- **Use of piecewise polynomials**: e.g., Spline interpolation

Definition 3.43. A **partition** of the interval $[a, b]$ is an ordered sequence $\{x_i\}_{i=0}^n$ such that

$$a = x_0 < x_1 < x_2 < \cdots < x_n = b.$$

The numbers x_i are known as **knots** or **nodes**.

Definition 3.44. A function S is a **spline of degree** k on $[a, b]$ if

1) The domain of S is $[a, b]$.
2) There exits a partition $\{x_i\}_{i=0}^n$ of $[a, b]$ such that on each subinterval $[x_{i-1}, x_i]$, $S \in \mathbb{P}_k$.
3) $S, S', \cdots, S^{(k-1)}$ are continuous on (a, b).

3.5.2. Linear splines

A **linear spline** is a continuous function which is linear on each subinterval. Thus it is defined entirely by its values at the nodes. That is, given

x	x_0	x_1	$\cdots$	x_n
y	y_0	y_1	$\cdots$	y_n

the linear polynomial on each subinterval is defined as

$$L_i(x) = y_{i-1} + \frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}), \quad x \in [x_{i-1}, x_i]. \tag{3.80}$$

Example 3.45. Find the linear spline for

x	0.0	0.2	0.5	0.8	1.0
y	1.3	3.0	2.0	2.1	2.5

Solution.

The linear spline can be easily computed as

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

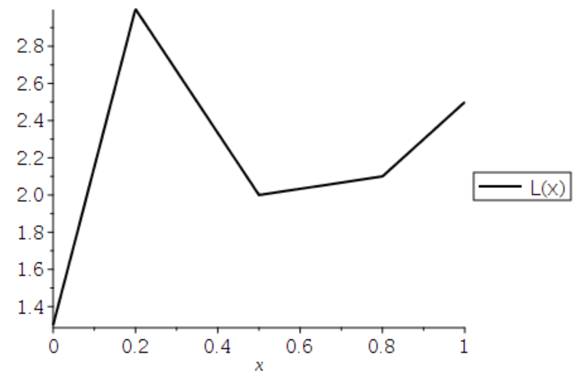


Figure 3.6: Linear spline.

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 $Q_i = Q|_{[x_{i-1}, x_i]}$. Then

$$Q_i(x) = a_i x^2 + b_i x + c_i, \quad x \in [x_{i-1}, x_i], \quad i = 1, 2, \cdots, n. \tag{3.84}$$

  Thus there are **3n parameters** to define $Q(x)$.

- For each of the **n subintervals**, the data (x_i, y_i), $i = 1, 2, \cdots, n$, gives two equations regarding $Q_i(x)$:

$$Q_i(x_{i-1}) = y_{i-1} \quad \text{and} \quad Q_i(x_i) = y_i, \quad i = 1, 2, \cdots, n. \tag{3.85}$$

  This is $2n$ equations. The continuity condition on Q' gives a single equation for each of the **$(n-1)$ internal nodes**:

$$Q_i'(x_i) = Q_{i+1}'(x_i), \quad i = 1, 2, \cdots, n-1. \tag{3.86}$$

  This **totals $(3n-1)$ equations**, but $3n$ unknowns.

- Thus **an additional user-chosen condition** is required, e.g.,

$$Q'(a) = f'(a), \quad Q'(a) = 0, \quad \text{or} \quad Q''(a) = 0. \tag{3.87}$$

  Alternatively, the additional condition can be given at $x = b$.

Algorithm 3.48. (Construction of quadratic splines):

(0) Define

$$z_i = Q'(x_i), \quad i = 0, 1, \cdots, n; \tag{3.88}$$

suppose that the additional condition is given by specifying z_0.

(1) Because $Q_i' = Q'|_{[x_{i-1}, x_i]}$ is a *linear function* satisfying

$$Q_i'(x_{i-1}) = z_{i-1} \quad \text{and} \quad Q_i'(x_i) = z_i, \quad (\textit{continuity of } Q') \tag{3.89}$$

we have

$$Q_i'(x) = z_{i-1} + \frac{z_i - z_{i-1}}{x_i - x_{i-1}}(x - x_{i-1}), \quad x \in [x_{i-1}, x_i]. \tag{3.90}$$

(2) By integrating it and using $Q_i(x_{i-1}) = y_{i-1}$ (*left edge value*)

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

(3) In order to determine z_i, $1 \le i \le n$, we use the above at x_i (*right edge value*):

$$y_i = Q_i(x_i) = \frac{z_i - z_{i-1}}{2(x_i - x_{i-1})}(x_i - x_{i-1})^2 + z_{i-1}(x_i - x_{i-1}) + y_{i-1}, \tag{3.92}$$

which implies

$$\begin{aligned} y_i - y_{i-1} &= \frac{1}{2}(z_i - z_{i-1})(x_i - x_{i-1}) + z_{i-1}(x_i - x_{i-1}) \\ &= (x_i - x_{i-1})\frac{(z_i + z_{i-1})}{2}. \end{aligned}$$

Thus we have

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

Note:

a. You should first decide z_i using (3.93) and then finalize Q_i from (3.91).

b. When z_n is specified, Equation (3.93) can be replaced by

$$z_{i-1} = 2\frac{y_i - y_{i-1}}{x_i - x_{i-1}} - z_i, \quad i = n, n-1, \cdots, 1. \tag{3.94}$$

Example 3.49. Find the quadratic spline for the same dataset used in Example 3.45, p. 114:

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]=0
z[1]=17
z[2]=-23.6667
z[3]=24.3333
z[4]=-20.3333
```

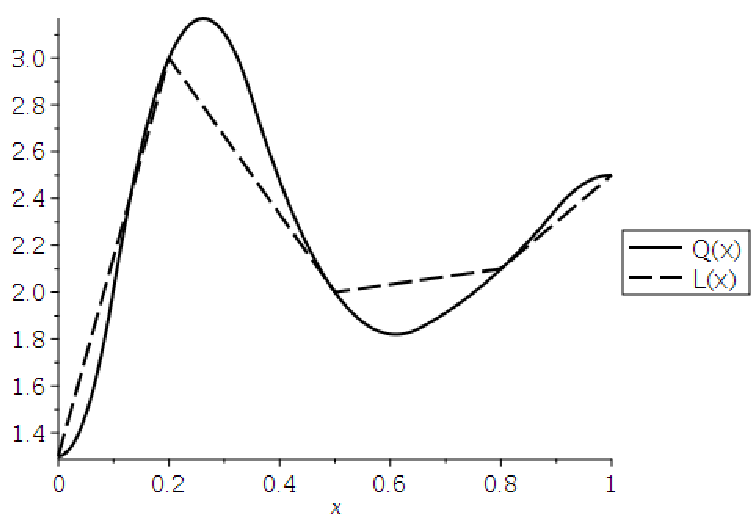


Figure 3.7: The graph of $Q(x)$ is superposed over the graph of the linear spline $L(x)$.

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.

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 $S_i'' = S''|_{[x_{i-1}, x_i]}$ is a *linear function* satisfying

$$S_i''(x_{i-1}) = z_{i-1} \quad \text{and} \quad S_i''(x_i) = z_i, \quad (\textit{continuity of } S'') \tag{3.98}$$

and therefore is given by the straight line between z_{i-1} and z_i:

$$S_i''(x) = \frac{z_{i-1}(x_i - x)}{h_i} + \frac{z_i(x - x_{i-1})}{h_i}, \quad h_i = x_i - x_{i-1}, \quad x \in [x_{i-1}, x_i]. \tag{3.99}$$

(2) If (3.99) is integrated twice, the result reads

$$S_i(x) = \frac{z_{i-1}(x_i - x)^3}{6h_i} + \frac{z_i(x - x_{i-1})^3}{6h_i} + C(x - x_{i-1}) + D(x_i - x). \tag{3.100}$$

In order to determine C and D, we use $S_i(x_{i-1}) = y_{i-1}$ and $S_i(x_i) = y_i$ (*left and right edge values*):

$$S_i(x_{i-1}) = \frac{z_{i-1}}{6}h_i^2 + Dh_i = y_{i-1}, \quad S_i(x_i) = \frac{z_i}{6}h_i^2 + Ch_i = y_i. \tag{3.101}$$

Thus (3.100) becomes

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

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

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

Construction of cubic splines (continue):

Then substitution of $x = x_i$ and simplification lead to

$$S_i'(x_i) = \frac{h_i}{6} z_{i-1} + \frac{h_i}{3} z_i + \frac{y_i - y_{i-1}}{h_i}. \tag{3.104}$$

Analogously, after obtaining S_{i+1}', we have

$$S_{i+1}'(x_i) = -\frac{h_{i+1}}{3} z_{i-1} - \frac{h_{i+1}}{6} z_{i+1} + \frac{y_{i+1} - y_i}{h_{i+1}}. \tag{3.105}$$

When the right sides of (3.104) and (3.105) are set equal to each other, the result reads

$$h_i\, z_{i-1} + 2(h_i + h_{i+1})\, z_i + h_{i+1}\, z_{i+1} = \frac{6(y_{i+1} - y_i)}{h_{i+1}} - \frac{6(y_i - y_{i-1})}{h_i}, \tag{3.106}$$

for $i = 1, 2, \cdots, n-1$.

(4) **Two additional user-chosen conditions** are required to determine $(n+1)$ unknowns, $z_0, z_1, \cdots, z_n$. There are two popular approaches for the choice of the two additional conditions.

Natural Cubic Spline :	$z_0 = 0,\ \ z_n = 0$
Clamped Cubic Spline :	$S'(a) = f'(a),\ \ S'(b) = f'(b)$

Natural Cubic Splines: Let $z_0 = z_n = 0$. Then the system of linear equations in (3.106) can be written as

$$A \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} b_2 - b_1 \\ b_3 - b_2 \\ \vdots \\ b_n - b_{n-1} \end{bmatrix}, \tag{3.107}$$

where

$$A = \begin{bmatrix} 2(h_1+h_2) & h_2 & & & \\ h_2 & 2(h_2+h_3) & h_3 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ & & & h_{n-1} & 2(h_{n-1}+h_n) \end{bmatrix} \quad \text{and} \quad b_i = \frac{6}{h_i}(y_i - y_{i-1}).$$

Clamped Cubic Splines: Let $f'(a)$ and $f'(b)$ be prescribed. Then the two extra conditions read

$$S'(a) = f'(a), \quad S'(b) = f'(b). \tag{3.108}$$

Since $a = x_0$ and $b = x_n$, utilizing Equation (3.103), the conditions read

$$\begin{aligned} 2h_1 z_0 + h_1 z_1 &= \frac{6}{h_1}(y_1 - y_0) - 6f'(x_0) \\ h_n z_{n-1} + 2h_n z_n &= 6f'(x_n) - \frac{6}{h_n}(y_n - y_{n-1}) \end{aligned} \tag{3.109}$$

Equation (3.106) and the above two equations clearly make $(n+1)$ conditions for $(n+1)$ unknowns, $z_0, z_1, \cdots, z_n$. It is a good exercise to compose an algebraic system for the computation of clamped cubic splines.

Example 3.52. Find the natural cubic spline for the same dataset

x	0.0	0.2	0.5	0.8	1.0
y	1.3	3.0	2.0	2.1	2.5

Solution.

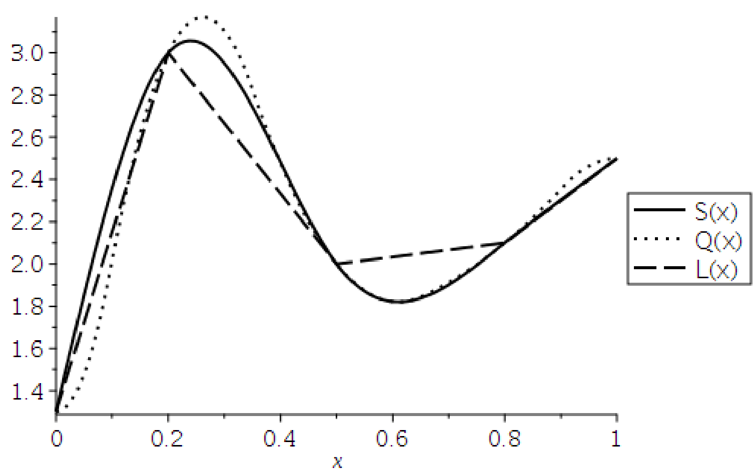


Figure 3.8: The graph of $S(x)$ is superposed over the graphs of the linear spline $L(x)$ and the quadratic spline $Q(x)$.

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 = 'natural'):
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  /
```

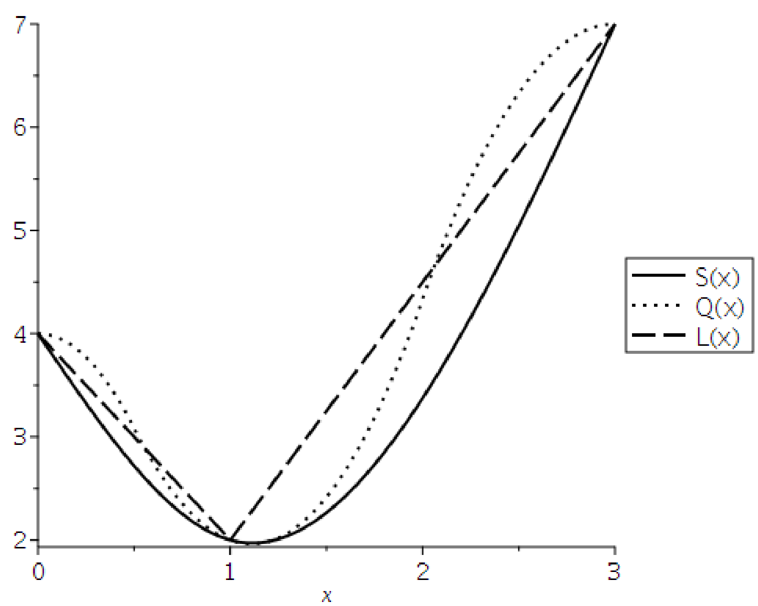


Figure 3.9

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*

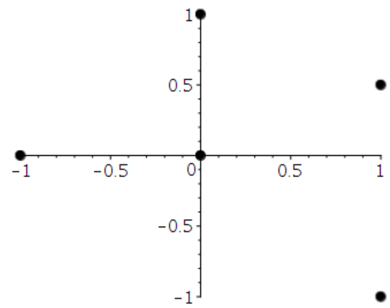
$$\int_a^b [S''(x)]^2\, dx \le \int_a^b [f''(x)]^2\, dx. \tag{3.110}$$

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)
```

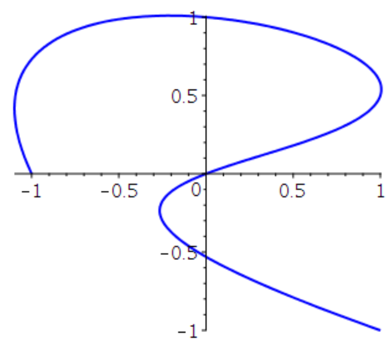


Figure 3.10

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

$$\begin{aligned} x(0) &= x_0, \quad y(0) = y_0, \quad \frac{dy}{dx}(t=0); \\ x(1) &= x_1, \quad y(1) = y_1, \quad \frac{dy}{dx}(t=1); \end{aligned} \tag{3.111}$$

  Only six conditions are specified, while the cubic polynomials $x(t)$ and $y(t)$ each have four parameters, for a total of eight.

- This provides flexibility in choosing the pair of cubic polynomials to specify the conditions.
- Notice that the natural form for determining $x(t)$ and $y(t)$ requires to specify

$$\begin{aligned} &x(0), \quad x(1), \quad x'(0), \quad x'(1); \\ &y(0), \quad y(1), \quad y'(0), \quad y'(1); \end{aligned} \tag{3.112}$$

- On the other hand, 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

$$\begin{aligned} \frac{dy}{dx}(t=0) &= \frac{(y_0+\beta_0)-y_0}{(x_0+\alpha_0)-x_0} = \frac{\beta_0}{\alpha_0} = \frac{y'(0)}{x'(0)}, \\ \frac{dy}{dx}(t=1) &= \frac{y_1-(y_1-\beta_1)}{x_1-(x_1-\alpha_1)} = \frac{\beta_1}{\alpha_1} = \frac{y'(1)}{x'(1)}. \end{aligned} \tag{3.114}$$

- Therefore, we may specify

$$x'(0) = \alpha_0, \quad y'(0) = \beta_0; \qquad x'(1) = \alpha_1, \quad y'(1) = \beta_1. \tag{3.115}$$

Formula. **(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 3.59. 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
    ```

    ⇒ `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`

    $\Rightarrow$ `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`

    $\Rightarrow$ `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)`

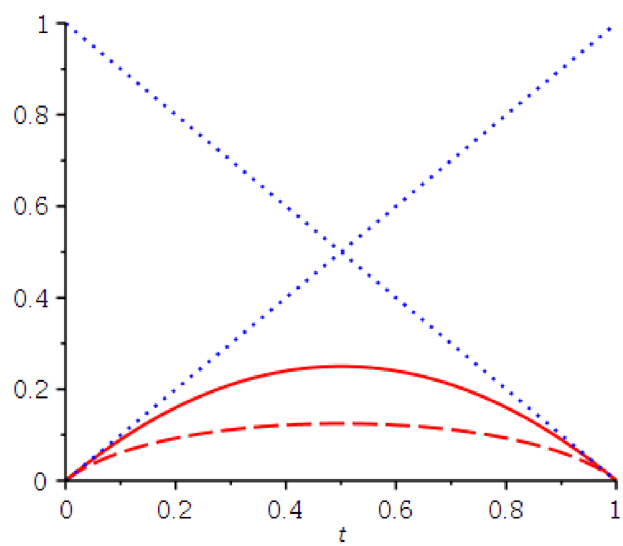


Figure 3.11: The parametric curves: $H_1(t)$ and $H_2(t)$.

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.)

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.

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.)

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.122.)

(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.

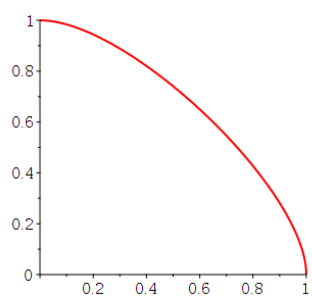
3.9. Construct the **piecewise cubic Hermite interpolating polynomial** for

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

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-point rules Five-point rules Richardson extrapolation	 Combination of low-order differences, to get higher-order accuracy
Numerical Integration	$f(x) \approx P_n(x)$ piecewisely $\Rightarrow \int_a^b f(x) \approx \int_a^b P_n(x)$
Trapezoid rule Simpson's rule Simpson's Three-Eights rule Romberg integration	 Newton-Cotes formulas
Gaussian Quadrature	Method of undetermined coefficients & orthogonal polynomials
Legendre polynomials	

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. **(Two-Point Difference Formulas)**: Let $x_1 = x_0 + h$ and $P_{0,1}$ be the first **Lagrange polynomial** interpolating f and 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 4.1. 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 4.2. 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. **(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

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

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 4.3. 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. **(Three-Point Difference Formulas** $(n = 2)$**)**: For convenience, let

$$x0, \quad x_1 = x_0 + h, \quad x_2 = x_0 + 2h, \qquad h > 0.$$

Recall

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

Thus, the **three-point endpoint formulas** and the **three-point midpoint formula** read

$$\begin{aligned}
f'(x_0) &= f(x_0)L'_{2,0}(x_0) + f(x_1)L'_{2,1}(x_0) + f(x_2)L'_{2,2}(x_0) + \frac{f^{(3)}(\xi(x_0))}{3!} \prod_{k=0,k\neq 0}^{2} (x_0 - x_k) \\
&= \frac{\mathbf{-3f(x_0) + 4f(x_1) - f(x_2)}}{\mathbf{2h}} + \frac{h^2}{3} f^{(3)}(\xi_0), \\
f'(x_1) &= \frac{\mathbf{f(x_2) - f(x_0)}}{\mathbf{2h}} - \frac{h^2}{6} f^{(3)}(\xi_1), \\
f'(x_2) &= \frac{\mathbf{f(x_0) - 4f(x_1) + 3f(x_2)}}{\mathbf{2h}} + \frac{h^2}{3} f^{(3)}(\xi_2).
\end{aligned} \tag{4.11}$$

Formula. **(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 4.4. (Numerical Differentiation, the $(n+1)$-point difference formulas):

1. $f(x) = P_n(x) + R_n(x), \quad P_n(x) \in \mathbb{P}_n$
2. $f'(x) = P_n'(x) + \mathcal{O}(h^n)$,
   $f''(x) = P_n''(x) + \mathcal{O}(h^{n-1})$, *and so on.*

Second-Derivative Midpoint Formula

Example 4.5. We can see from the above summary that when the three-point ($n = 2$) difference formula is applied for the approximation of f'', the accuracy reads $\mathcal{O}(h)$. Use the *Taylor expansion* to derive the formula

$$f''(x_0) = \frac{f_{-1} - 2f_0 + f_1}{h^2} - \frac{h^2}{12} f^{(4)}(\xi). \tag{4.13}$$

Solution.

Example 4.6. 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 difference results while using low-order formulas.

Example 4.7. Derive the *three-point midpoint formulas*:

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

Solution.

Observation 4.8. The equations can be written as

$$M = N(h) + K_2h^2 + K_4h^4 + K_6h^6 + \cdots, \tag{4.15}$$

where M is the desired (unknown) quantity, $N(h)$ s an approximation of M using the parameter h, and K_i are values independent of h.

How can we take advantage of the observation?

Strategy 4.9. (Richardson extrapolation):

1. Let's first write out (4.15) with h replaced by $h/2$:

$$M = N(h/2) + K_2 h^2/4 + K_4 h^4/16 + K_6 h^6/64 + \cdots . \tag{4.16}$$

Then the leading term in the error series, $K_2 h^2$, can be eliminated as follows:

$$\begin{array}{rcll} M &=& N(h) & +K_2 h^2 + K_4 h^4 + K_6 h^6 + \cdots \\ 4M &=& 4N(h/2) & +K_2 h^2 + K_4 h^4/4 + K_6 h^6/16 + \cdots \\ \hline 3M &=& 4N(h/2) - N(h) & -\dfrac{3}{4}K_4 h^4 - \dfrac{15}{16}K_6 h^6 - \cdots \end{array} \tag{4.17}$$

Thus we have

$$M = \frac{1}{3}[4N(h/2) - N(h)] - \frac{1}{4}K_4 h^4 - \frac{5}{16}K_6 h^6 - \cdots . \tag{4.18}$$

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 (4.18) as

$$M = N_2(h) - \frac{1}{4}K_4 h^4 - \frac{5}{16}K_6 h^6 - \cdots . \tag{4.19}$$

Then, similarly,

$$M = N_2(h/2) - \frac{1}{64}K_4 h^4 - \frac{5}{2^{10}}K_6 h^6 - \cdots . \tag{4.20}$$

Subtract (4.18) from 16 times (4.20) to produce a new $\mathcal{O}(h^6)$ formula:

$$M = \frac{1}{15}[16N_2(h/2) - N_2(h)] + \frac{1}{64}K_6 h^6 + \cdots . \tag{4.21}$$

Algorithm 4.10. (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

$$D(i,0) = N(h/2^i), \quad i = 0, 1, \cdots, n. \tag{4.22}$$

2. Compute additional quantities using the formula

for $i = 1, 2, \cdots, n$ do
  for $j = 1, 2, \cdots, i$ do

$$D(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot D(i,j-1) - D(i-1,j-1)\right] \tag{4.23}$$

  end do
end do

Note:

(a) One can prove that

$$D(i,j) = M + \mathcal{O}(h^{2(j+1)}). \tag{4.24}$$

(b) The second step in the algorithm can be rewritten for a column-wise computation:

for $j = 1, 2, \cdots, i$ do
  for $i = j, j+1, \cdots, n$ do

$$D(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot D(i,j-1) - D(i-1,j-1)\right] \tag{4.25}$$

  end do
end do

Example 4.11. 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 4.12. Let $f(x) = \ln x$, as in the previous example, Example 4.11. 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	$D_{2,0} = -1.001252088$	$D_{2,1} = -0.9999915873$	$D_{2,2} = -1.000000260$

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

$$P_n(x) = \sum_{i=0}^{n} f(x_i) L_{n,i}(x), \tag{4.26}$$

which interpolates the function f. Then, as just mentioned, we simply approximate

$$\int_a^b f(x)\,dx \approx \int_a^b P_n(x)\,dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_{n,i}(x)\,dx. \tag{4.27}$$

Definition 4.13. In this way, we obtain a formula which is a *weighted sum* of the function values:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} A_i\, f(x_i), \tag{4.28}$$

where

$$A_i = \int_a^b L_{n,i}(x)\,dx. \tag{4.29}$$

The formula of the form in (4.28) 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,

$$f(x) = \sum_{i=0}^{1} f(x_i) L_{1,i}(x) + \frac{f''(\xi_x)}{2!}(x - x_0)(x - x_1), \tag{4.30}$$

and its integration reads

$$\int_{x_0}^{x_1} f(x)\, dx = \sum_{i=0}^{1} f(x_i) \int_{x_0}^{x_1} L_{1,i}(x)\, dx + \int_{x_0}^{x_1} \frac{f''(\xi_x)}{2!}(x - x_0)(x - x_1)\, dx. \tag{4.31}$$

Derivation 4.14. *Terms in the right-side of (4.31) must be verified to get a formula and its error bound. Note that*

$$\begin{aligned} A_0 &= \int_{x_0}^{x_1} L_{1,0}(x)\, dx = \int_{x_0}^{x_1} \frac{x - x_1}{x_0 - x_1} dx = \frac{1}{2}(x_1 - x_0), \\ A_1 &= \int_{x_0}^{x_1} L_{1,1}(x)\, dx = \int_{x_0}^{x_1} \frac{x - x_0}{x_1 - x_0} dx = \frac{1}{2}(x_1 - x_0), \end{aligned} \tag{4.32}$$

and

$$\begin{aligned} \int_{x_0}^{x_1} \frac{f''(\xi_x)}{2!}(x - x_0)(x - x_1)\, dx &= \frac{f''(\xi)}{2!} \int_{x_0}^{x_1} (x - x_0)(x - x_1)\, dx \\ &= -\frac{f''(\xi)}{12}(x_1 - x_0)^3. \end{aligned} \tag{4.33}$$

(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 4.15. The corresponding quadrature formula is

$$\int_{x_0}^{x_1} f(x)\, dx = \frac{h}{2}[f(x_0) + f(x_1)] - \frac{h^3}{12} f''(\xi), \quad \text{(Trapezoid)} \tag{4.34}$$

which is known as the **trapezoid rule**.

Graphical interpretation:

```
with(Student[Calculus1]):
f := x^3 + 2 + sin(2*Pi*x):
ApproximateInt(f, 0..1, output = animation, partition = 1,
    method = trapezoid, refinement = halve,
    boxoptions = [filled = [color=pink,transparency=0.5]]);
```

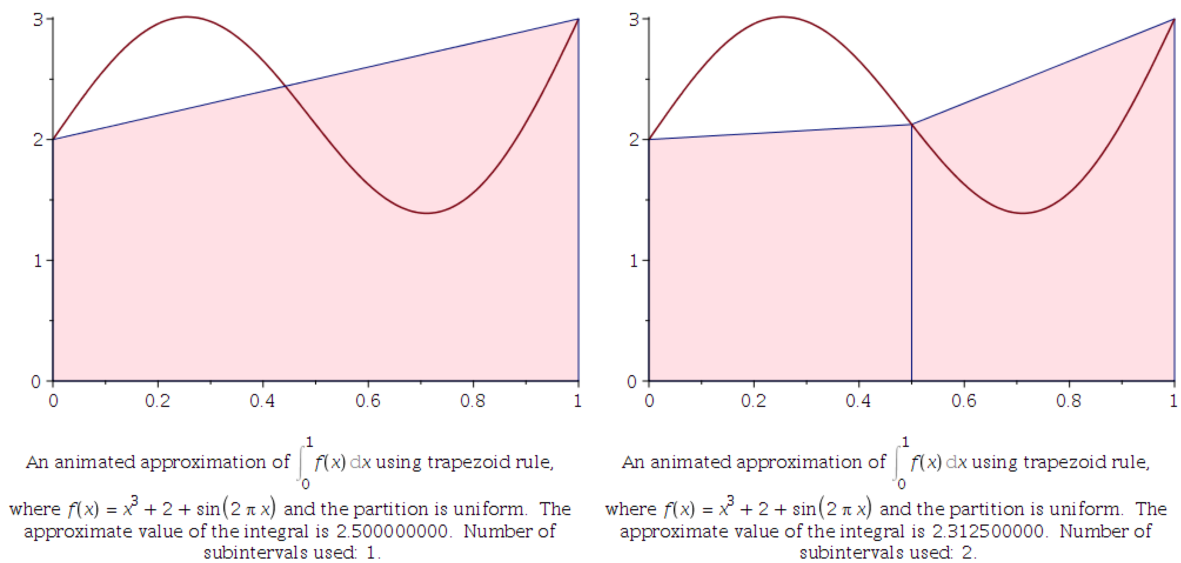


Figure 4.1: Trapazoid rule.

Composite trapezoid rule: Let the interval $[a, b]$ be partitioned as

$$a = x_0 < x_1 < \cdots < x_n = b.$$

Then the trapezoid rule can be applied to each subinterval. Here the nodes are not necessarily uniformly spaced. Thus, we obtain the **composite trapezoid rule** reads

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x)\,dx \approx \sum_{i=1}^{n} \frac{h_i}{2}\left(f(x_{i-1}) + f(x_i)\right), \quad h_i = x_i - x_{i-1}. \tag{4.35}$$

With a uniform spacing:

$$x_i = a + i\,h, \quad h = \frac{b-a}{n},$$

the composite trapezoid rule takes the form

$$\int_a^b f(x)\,dx \approx h \cdot \left[\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_i)\right], \tag{4.36}$$

for which the **composite error** becomes

$$\sum_{i=1}^{n}\left(-\frac{h^3}{12}f''(\xi_i)\right) = -f''(\xi)\sum_{i=1}^{n}\frac{h^3}{12} = -f''(\xi)\frac{h^3}{12}\cdot n = -f''(\xi)\frac{(b-a)h^2}{12}, \tag{4.37}$$

where we have used $\left(h = \dfrac{b-a}{n} \Rightarrow n = \dfrac{b-a}{h}\right)$.

Example 4.16.

```
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]]);
```

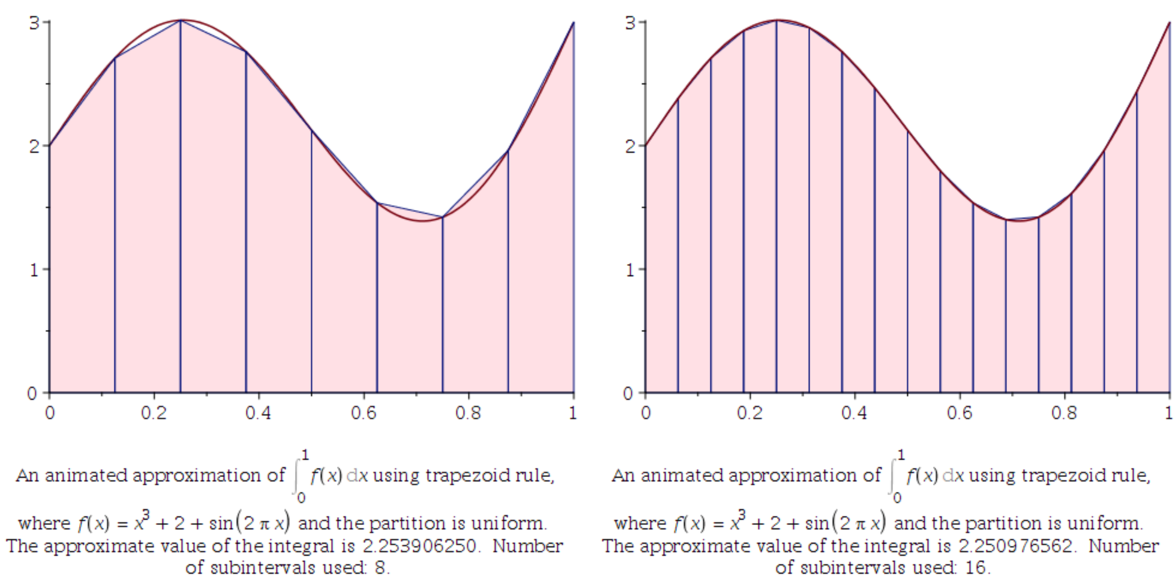


Figure 4.2: Composite trapazoid rule.

4.3.2. Simpson's rule

Simpson's rule results from integrating over $[a,b]$ the **second Lagrange polynomial** with three equally spaced nodes:

$$x_0 = a,\quad x_1 = a+h,\quad x_2 = a+2h = b. \qquad \left(h = \frac{b-a}{2}\right)$$

Definition 4.17. The **elementary Simpson's rule** reads

$$\int_a^b f(x)\,dx = \int_{x_0}^{x_2} f(x)\,dx \approx \int_{x_0}^{x_2} \sum_{i=0}^{2} f(x_i)L_{2,i}(x), \tag{4.38}$$

which is reduced to

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{2h}{6}\Big[f(x_0) + 4f(x_1) + f(x_2)\Big]. \tag{4.39}$$

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]]);
```

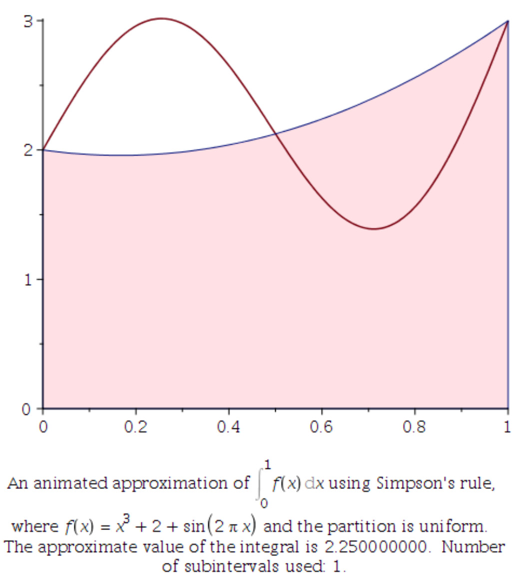


Figure 4.3: The elementary Simpson's rule, which is **exact** for the given problem.

Error analysis for the elementary Simpson's rule:

- The error for the elementary Simpson's rule can be analyzed from

$$\int_{x_0}^{x_2} \frac{f'''(\xi)}{3!}(x-x_0)(x-x_1)(x-x_2)\,dx, \tag{4.40}$$

which must be in $\mathcal{O}(h^4)$.

- However, by approximating the problem in another way, one can show **the error is in $\mathcal{O}(h^5)$**. (See Example 4.5, p.135.)
- 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

$$f(x) = f(x_1)+f'(x_1)(x-x_1)+\frac{f''(x_1)}{2!}(x-x_1)^2+\frac{f'''(x_1)}{3!}(x-x_1)^3+\frac{f^{(4)}(\xi)}{4!}(x-x_1)^4. \tag{4.41}$$

By integrating the terms over $[x_0, x_2]$, we have

$$\begin{aligned}\int_{x_0}^{x_2} f(x)\,dx &= \Big| f(x_1)(x-x_1) + \frac{f'(x_1)}{2}(x-x_1)^2 + \frac{f''(x_1)}{3!}(x-x_1)^3 \\ &\quad + \frac{f'''(x_1)}{4!}(x-x_1)^4\Big|_{x_0}^{x_2} + \int_{x_0}^{x_2} \frac{f^{(4)}(\xi)}{4!}(x-x_1)^4\,dx.\end{aligned} \tag{4.42}$$

The last term can be easily computed by using the *Weighted Mean Value Theorem on Integral*:

$$\int_{x_0}^{x_2} \frac{f^{(4)}(\xi)}{4!}(x-x_1)^4\,dx = \frac{f^{(4)}(\xi_1)}{4!}\int_{x_0}^{x_2}(x-x_1)^4\,dx = \frac{f^{(4)}(\xi_1)}{60}h^5. \tag{4.43}$$

Thus, Equation (4.42) reads

$$\int_{x_0}^{x_2} f(x)\,dx = 2h\,f(x_1) + \frac{h^3}{3}f''(x_1) + \frac{f^{(4)}(\xi_1)}{60}h^5. \tag{4.44}$$

See (4.14), p.137, to recall that

$$f''(x_1) = \frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2} - \frac{h^2}{12}f^{(4)}(\xi_2). \tag{4.45}$$

Plugging this to (4.44) reads

$$\int_{x_0}^{x_2} f(x)\,dx = 2h\,f(x_1) + \frac{h^3}{3}\Big(\frac{f(x_0) - 2f(x_1) + f(x_2)}{h^2}\Big) - \frac{h^3}{3}\Big(\frac{h^2}{12}f^{(4)}(\xi_2)\Big) + \frac{f^{(4)}(\xi_1)}{60}h^5, \tag{4.46}$$

and therefore

$$\int_{x_0}^{x_2} f(x)\,dx = \frac{2h}{6}\big[f(x_0) + 4f(x_1) + f(x_2)\big] - \frac{h^5}{90} f^{(4)}(\xi_3). \tag{4.47}$$

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

$$\int_a^b f(x)\,dx = \sum_{i=1}^{n/2} \int_{x_{2i-2}}^{x_{2i}} f(x)\,dx \approx \frac{2h}{6} \sum_{i=1}^{n/2} \big[f(x_{2i-2}) + 4f(x_{2i-1}) + f(x_{2i})\big]. \tag{4.48}$$

Example 4.18. Show that the error term for the composite Simpson's rule becomes

$$-\frac{(b-a)h^4}{180} f^{(4)}(\xi). \tag{4.49}$$

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 4.19. *When* three equal subintervals *are combined, the resulting integration formula is called the* ***Simpson's three-eights rule****:*

$$\int_{x_0}^{x_3} f(x)\,dx = \frac{3h}{8}\Big[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\Big] - \frac{3h^5}{80} f^{(4)}(\xi). \tag{4.50}$$

Example 4.20. 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.

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 4.21. Consider

$$\int_0^2 \frac{x}{x^2+1}dx,$$

of which the true value is $\frac{\ln 5}{2} \approx 0.80471895621705018730$. Use 6 equally spaced points ($n = 5$) to approximate the integral using

(a) the trapezoid rule, and

(b) a combination of the Simpson's rule and the Simpson's three-eights rule.

Solution.

(a) 0.7895495781.

(so, the error = **0.0151693779**)

(b) $\int_0^{0.8} f(x)dx + \int_{0.8}^{2} f(x)dx \approx 0.2474264468 + 0.5567293981 = 0.8041558449.$

(so, the error = **0.0005631111**)

4.4. Romberg Integration

In the previous section, we have found that the *Composite Trapezoid rule* has a truncation error of order $\mathcal{O}(h^2)$. Specifically, we showed that for

$$x_k = a + k\,h, \quad h = \frac{b-a}{n}, \qquad (0 \le k \le n)$$

we have

$$\int_a^b f(x)\,dx = T(n) - f''(\xi)\frac{(b-a)h^2}{12}, \quad \text{(Composite Trapezoid)} \tag{4.51}$$

where $T(n)$ is the Trapezoid quadrature obtained over n equal subintervals:

$$T(n) = h\left[\frac{1}{2}f(a) + \sum_{k=1}^{n-1} f(x_k) + \frac{1}{2}f(b)\right]. \tag{4.52}$$

4.4.1. Recursive Trapezoid rule

Let us begin with an effective computation technique for the Composite Trapezoid rule when $n = 2^i$.

Example 4.22. 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 (4.52), we have

$$\begin{aligned}
T(1) &= 1 \cdot \left[\frac{1}{2}f(0) + \frac{1}{2}f(1)\right] \\
T(2) &= \frac{1}{2} \cdot \left[\frac{1}{2}f(0) + f\left(\frac{1}{2}\right) + \frac{1}{2}f(1)\right] \\
T(4) &= \frac{1}{4} \cdot \left[\frac{1}{2}f(0) + f\left(\frac{1}{4}\right) + f\left(\frac{1}{2}\right) + f\left(\frac{3}{4}\right) + \frac{1}{2}f(1)\right] \\
T(8) &= \frac{1}{8} \cdot \left[\frac{1}{2}f(0) + f\left(\frac{1}{8}\right) + f\left(\frac{1}{4}\right) + f\left(\frac{3}{8}\right) + f\left(\frac{1}{2}\right)\right. \\
&\qquad \left. + f\left(\frac{5}{8}\right) + f\left(\frac{3}{4}\right) + f\left(\frac{7}{8}\right) + \frac{1}{2}f(1)\right]
\end{aligned} \tag{4.53}$$

Remark 4.23. It is clear that if $T(2n)$ is to be computed, then we can take advantage of the work already done in the computation of $T(n)$. For example, from the preceding example, we see that

$$\begin{aligned} T(2) &= \frac{1}{2}T(1) + \frac{1}{2}\cdot\left[f\left(\frac{1}{2}\right)\right] \\ T(4) &= \frac{1}{2}T(2) + \frac{1}{4}\cdot\left[f\left(\frac{1}{4}\right) + f\left(\frac{3}{4}\right)\right] \\ T(8) &= \frac{1}{2}T(4) + \frac{1}{8}\cdot\left[f\left(\frac{1}{8}\right) + f\left(\frac{3}{8}\right) + f\left(\frac{5}{8}\right) + f\left(\frac{7}{8}\right)\right] \end{aligned} \tag{4.54}$$

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

$$T(2n) = \frac{1}{2}T(n) + h\,[f(a+h) + f(a+3h) + \cdots + f(a+(2n-1)h)], \tag{4.55}$$

or

$$T(2n) = \frac{1}{2}T(n) + h\left(\sum_{k=1}^{n} f(x_{2k-1})\right). \tag{4.56}$$

Now, if there are 2^i uniform subintervals, Equation (4.55) provides a **recursive Trapezoid rule**:

$$T(2^i) = \frac{1}{2}T(2^{i-1}) + h_i\left(\sum_{k=1}^{2^{i-1}} f(a+(2k-1)h_i)\right), \tag{4.57}$$

where

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

4.4.2. The Romberg algorithm

By an alternative method (Taylor series method), it can be shown that if $f \in C^\infty[a,b]$, the Composite Trapezoid rule (4.52) can also be written with an error term in the form

$$\int_a^b f(x)\,dx \;=\; T(n) + K_2h^2 + K_4h^4 + K_6h^6 + \cdots\,, \tag{4.58}$$

where K_i are constants independent of h. Since

$$\int_a^b f(x)\,dx \;=\; T(2n) + K_2h^2/4 + K_4h^4/16 + K_6h^6/64 + \cdots\,, \tag{4.59}$$

as for *Richardson extrapolation*, we have

$$\int_a^b f(x)\,dx \;=\; \frac{1}{3}\big[4T(2n) - T(n)\big] - \frac{3}{4}K_4h^4 - \frac{15}{16}K_6h^6 - \cdots\,. \tag{4.60}$$

Algorithm 4.24. (**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 (4.57):

$$\begin{aligned} R(0,0) &= \frac{b-a}{2}[f(a)+f(b)],\\ R(i,0) &= \frac{1}{2}R(i-1,0) + h_i\Big(\sum_{k=1}^{2^{i-1}} f(a+(2k-1)h_i)\Big). \end{aligned} \tag{4.61}$$

2. Then, evaluate higher-order approximations recursively using

for $i = 1, 2, \cdots, n$ do
  for $j = 1, 2, \cdots, i$ do

$$R(i,j) = \frac{1}{4^j - 1}\left[4^j \cdot R(i,j-1) - R(i-1,j-1)\right] \tag{4.62}$$

  end do
end do

Example 4.25. 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
```

$j = 0 : \mathcal{O}(h^2)$	$j = 1 : \mathcal{O}(h^4)$	$j = 2 : \mathcal{O}(h^6)$	$j = 3 : \mathcal{O}(h^8)$
$R_{0,0} = 0$			
$R_{1,0} = 1.570796327$	$R_{1,1} = 2.094395103$		
$R_{2,0} = 1.896118898$	$R_{2,1} = 2.004559755$	$R_{2,2} = 1.998570731$	
$R_{3,0} = 1.974231602$	$R_{3,1} = 2.000269171$	$R_{3,2} = 1.999983131$	$R_{3,3} = 2.000005551$

(4.63)

Self-study 4.26. The true value for the integral: $\int_0^{\pi} \sin x \, dx = 2$. Use the table in (4.63) 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:

- In Section 4.3, we saw how to create quadrature formulas of the type

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} w_i\, f(x_i), \tag{4.64}$$

that are exact for polynomials of degree $\le n$, which is the case if and only if

$$w_i = \int_a^b L_{n,i}(x)\,dx = \int_a^b \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\,dx. \tag{4.65}$$

- 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 coefficients were determined uniquely form the requirement that Formula (4.64) must be an equality for $f \in \mathbb{P}_n$.

4.5.1. The method of undetermined coefficients

Example 4.27. Find w_0, w_1, w_2 with which the following formula is exact for all polynomials of degree ≤ 2:

$$\int_0^1 f(x)\,dx \;\approx\; w_0\, f(0) + w_1\, f(1/2) + w_2\, f(1). \tag{4.66}$$

Solution. Formula (4.66) must be exact for some low-order polynomials. Consider trial functions $f(x) = 1,\ x,\ x^2$. Then, for each of them,

$$\begin{aligned}
1 &= \int_0^1 1\,dx &&= w_0 \cdot 1 + w_1 \cdot 1 + w_2 \cdot 1 &&= w_0 + w_1 + w_2 \\
\frac{1}{2} &= \int_0^1 x\,dx &&= w_0 \cdot 0 + w_1 \cdot \frac{1}{2} + w_2 \cdot 1 &&= \frac{1}{2} w_1 + w_2 \\
\frac{1}{3} &= \int_0^1 x^2\,dx &&= w_0 \cdot 0 + w_1 \cdot \left(\frac{1}{2}\right)^2 + w_2 \cdot 1 &&= \frac{1}{4} w_1 + w_2
\end{aligned} \tag{4.67}$$

The solution of this system of three simultaneous equations is

$$(w_0, w_1, w_2) = \left(\frac{1}{6}, \frac{2}{3}, \frac{1}{6}\right). \tag{4.68}$$

Thus the formula can be written as

$$\int_0^1 f(x)\,dx \approx \frac{1}{6}\big[f(0) + 4f(1/2) + f(1)\big], \tag{4.69}$$

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 (4.69) is the *elementary Simpson's rule* with $h = 1/2$.

Key Idea 4.28. Gaussian quadrature chooses the points for evaluation 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

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i\, f(x_i). \tag{4.70}$$

- 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 ($x_1, x_2, \cdots, x_n$ and $w_1, w_2, \cdots, w_n$) to choose. 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 4.29. Determine x_1, x_2 and w_1, w_2 so that the integration formula

$$\int_{-1}^{1} f(x)\,dx \;\approx\; w_1 f(x_1) + w_2 f(x_2) \tag{4.71}$$

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* can be used to determine the nodes and weights for formulas that give exact results for high-order polynomials, but an alternative method obtained them mote easily. The alternative is related to *Legendre orthogonal polynomials*.

4.5.2. Legendre polynomials

Definition 4.30. 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

$$\int_{-1}^{1} Q(x)P_k(x)\,dx = 0, \;\; \forall\, Q(x) \in \mathbb{P}_{k-1}. \tag{4.72}$$

Such orthogonal polynomials can be formulated by a certain three-term recurrence relation.

Definition 4.31. The **Legendre polynomials** obey the three-term recurrence relation, known as *Bonnet's recursion formula*:

$$(k+1)P_{k+1}(x) = (2k+1)xP_k(x) - kP_{k-1}(x), \tag{4.73}$$

beginning with $P_0(x) = 1$, $P_1(x) = x$. A few first Legendre polynomials are

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= \frac{3}{2}(x^2 - \frac{1}{3}) \\ P_3(x) &= \frac{5}{2}(x^3 - \frac{3}{5}x) \\ P_4(x) &= \frac{35}{8}(x^4 - \frac{6}{7}x^2 + \frac{3}{35}) \\ P_5(x) &= \frac{63}{8}(x^5 - \frac{10}{9}x^3 + \frac{5}{21}x) \end{aligned} \tag{4.74}$$

Theorem 4.32. *The Legendre polynomials satisfy*

$$\begin{aligned} &|P_k(x)| \le 1, \quad \forall\, x \in [-1, 1], \\ &P_k(\pm 1) = (\pm 1)^k, \\ &\int_{-1}^{1} P_j(x)P_k(x)\,dx = 0, \quad j \neq k; \qquad \int_{-1}^{1} P_k(x)^2\,dx = \frac{1}{k+1/2}. \end{aligned} \tag{4.75}$$

Note: The Chebyshev polynomials, defined in Definition 3.20, are also orthogonal polynomials; see page 95. Frequently-cited classical orthogonal polynomials are: Jacobi polynomials, Laguerre polynomials, Chebyshev polynomials, and Legendre polynomials.

4.5.3. Gauss integration

Theorem 4.33. ***(Gauss integration)****: Suppose that* $\{x_1, x_2, \cdots, x_n\}$ *are the roots of the* n**th Legendre polynomial** P_n *and* $\{w_1, w_2, \cdots, w_n\}$ *are obtained by*

$$w_i = \int_{-1}^{1} \prod_{j=1,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \left(= \int_{-1}^{1} L_{n-1,i}(x)\, dx \right) \tag{4.76}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i) \;\; \textit{is exact}, \;\; \forall f \in \mathbb{P}_{2n-1}. \tag{4.77}$$

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

$$\sum_{j=1}^{n} (x_j)^i w_j = \int_{-1}^{1} x^i\, dx, \quad i = 0, 1, \cdots, n-1. \tag{4.78}$$

Gauss-integration

```
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 4.34. (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*:

$$T : [-1, 1] \to [a, b], \quad x = T(t) = \frac{b-a}{2}t + \frac{a+b}{2}. \tag{4.79}$$

Using it, we have

$$\int_a^b f(x)\,dx = \int_{-1}^1 f\Big(\frac{b-a}{2}t + \frac{a+b}{2}\Big)\frac{b-a}{2}\,dt. \tag{4.80}$$

Example 4.35. Find the Gaussian Quadrature for $\int_0^{\pi} \sin(x)\,dx$, with $n = 2, 3, 4$.

Solution.

Gaussian-Quadrature

```
a := 0:
b := Pi:
f := x -> sin(x):
nmax := 4:
T := t -> 1/2*(b - a)*t + 1/2*a + 1/2*b:
T(t)
                         1        1
                         - Pi t + - Pi
                         2        2
trueI := int(f(x), x = a..b)
                               2

g := t -> 1/2*f(T(t))*(b - a):
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
```

4.5.4. Gauss-Lobatto integration

Theorem 4.36. (***Gauss-Lobatto integration***): *Let* $x_0 = -1$, $x_n = 1$, *and* $\{x_1, x_2, \cdots, x_{n-1}\}$ *are the roots of the first-derivative of the nth Legendre polynomial,* $P_n'(x)$. *Let* $\{w_0, w_1, w_2, \cdots, w_n\}$ *be obtained by*

$$w_i = \int_{-1}^{1} \prod_{j=0,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \left(= \int_{-1}^{1} L_{n,i}(x)\, dx \right) \tag{4.81}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=0}^{n} w_i\, f(x_i) \;\; \textit{is exact}, \;\; \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.82}$$

Recall: **Theorem 4.33 (Gauss integration)**: Suppose that $\{x_1, x_2, \cdots, x_n\}$ are the roots of the n***th Legendre polynomial*** P_n and $\{w_1, w_2, \cdots, w_n\}$ are obtained by

$$w_i = \int_{-1}^{1} \prod_{j=1,\, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}\, dx. \quad \left(= \int_{-1}^{1} L_{n-1,i}(x)\, dx \right) \tag{4.83}$$

Then,

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i) \;\; \text{is exact}, \;\; \forall\, f \in \mathbb{P}_{2n-1}. \tag{4.84}$$

Remark 4.37.

- 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.
- Once the nodes are determined, the weights $\{w_0, w_1, w_2, \cdots, w_n\}$ an also be found by using the *method of undetermined coefficients*, as for Gauss integration; the weights are the solution of the linear system

$$\sum_{j=0}^{n} (x_j)^i w_j = \int_{-1}^{1} x^i\, dx, \quad i = 0, 1, \cdots, n. \tag{4.85}$$

Self-study **4.38.** Find the Gauss-Lobatto Quadrature for $\int_0^\pi \sin(x)\,dx$, with $n = 2, 3, 4$.

Exercises for Chapter 4

4.1. Use the most accurate three-point formulas to determine the missing entries.

x	$f(x)$	$f'(x)$	$f''(x)$
1.0	2.0000		6.00
1.2	1.7536		
1.4	1.9616		
1.6	2.8736		
1.8	4.7776		
2.0	8.0000		
2.2	12.9056		52.08

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:

$$f'(1.6) = 6.784, \quad f''(1.6) = 24.72.$$

4.3. Let a numerical process be described by

$$M = N(h) + K_1 h + K_2 h^2 + K_3 h^3 + \cdots \tag{4.86}$$

Explain how Richardson extrapolation will work in this case. (Try to introduce a formula described as in (4.23), page 139.)

4.4. 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.

4.5. 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

How long is the track?

4.6. 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 $R[3,3]$.

4.7. C Find the Gaussian Quadrature for $\displaystyle\int_0^{\pi/2} \cos^2 x\,dx$, with $n = 2, 3, 4$.

Chapter 5

Numerical Solution of Ordinary Differential Equations

In this chapter:

Topics	**Applications/Properties**
Elementary Theory of IVPs	Existence and uniqueness of 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	

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

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

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*

$$y(x + h) = y(x) + hy'(x) + \frac{h^2}{2}y''(x) + \cdots. \tag{5.6}$$

- Letting $x = x_0$ and 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

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

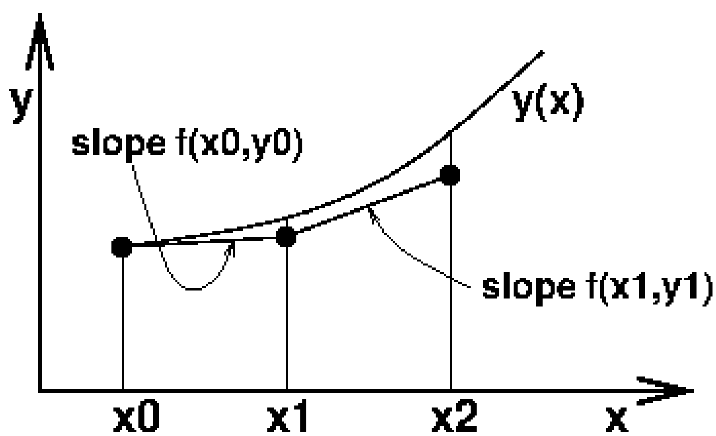


Figure 5.1: The Euler method.

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 Figure 5.1. For example, y_1 is determined by moving the point (x_0, y_0) by the length of h with the slope $f(x_0, y_0)$.

Convergence of the Euler method

Theorem 5.9. *Let f satisfy the Lipschitz condition in its second variable, i.e., there is $\lambda > 0$ such that*

$$|f(x, y_1) - f(x, y_2)| \le \lambda |y_1 - y_2|, \quad \forall\, y_1, y_2. \tag{5.10}$$

Then, the Euler method is convergent; more precisely,

$$|y_n - y(x_n)| \le \frac{C}{\lambda} h[(1 + \lambda h)^n - 1], \quad n = 0, 1, 2, \cdots. \tag{5.11}$$

Proof. The true solution y satisfies

$$y(x_{n+1}) = y(x_n) + hf(x_n, y(x_n)) + \mathcal{O}(h^2). \tag{5.12}$$

Thus it follows from (5.9) and (5.12) 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 (5.10), we have

$$|e_{n+1}| \le (1 + \lambda h)|e_n| + Ch^2. \tag{5.13}$$

Here we will prove (5.11) by using (5.13) 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 5.10. Consider

$$\begin{aligned} y' &= y - x^3 + x + 1, \quad 0 \le x \le 3, \\ y(0) &= 0.5. \end{aligned} \tag{5.14}$$

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:

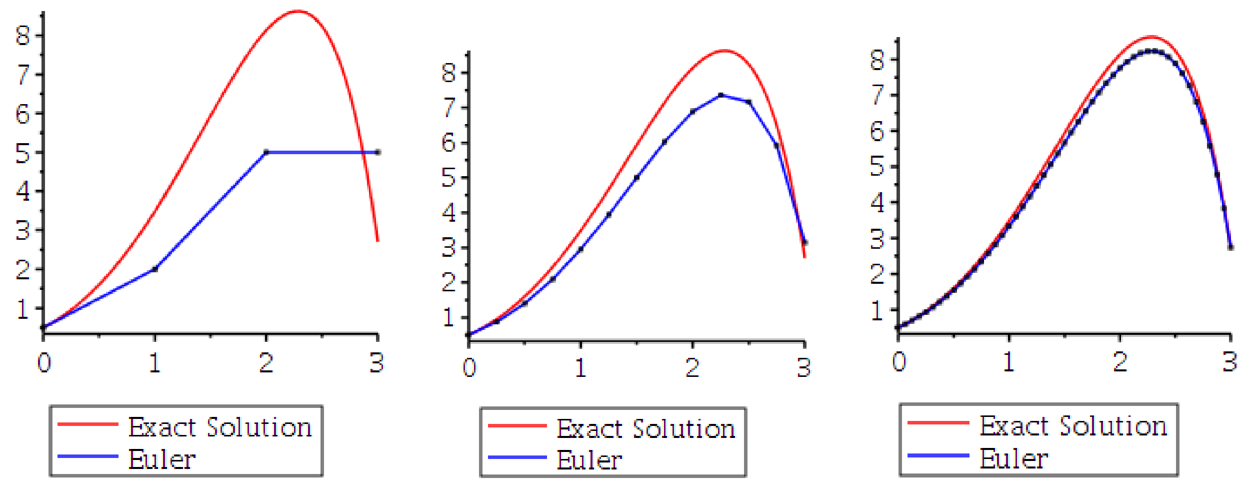


Figure 5.2: The Euler method, with $h = 1 \to \frac{1}{4} \to \frac{1}{16}$.

Example 5.11. 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 5.12. **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

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + hy'(x_n) + \frac{h^2}{2!}y''(x_n) + \cdots + \frac{h^m}{m!}y^{(m)}(x_n) \\ &\quad + \frac{h^{m+1}}{(m+1)!}y^{(m+1)}(\xi_n). \end{aligned} \tag{5.15}$$

- Successive differentiation of the solution, $y(x)$, gives

$$y'(x) = f(x, y(x)), \quad y''(x) = f'(x, y(x)), \quad \cdots ,$$

and generally,

$$y^{(k)}(x) = f^{(k-1)}(x, y(x)). \tag{5.16}$$

- Thus, we have

$$\begin{aligned} y(x_{n+1}) &= y(x_n) + hf(x_n, y(x_n)) + \frac{h^2}{2!}f'(x_n, y(x_n)) + \cdots + \frac{h^m}{m!}f^{(m-1)}(x_n, y(x_n)) \\ &\quad + \frac{h^{m+1}}{(m+1)!}f^{(m)}(\xi_n, y(\xi_n)) \end{aligned} \tag{5.17}$$

Algorithm 5.13. The **Taylor method of order m** corresponding to (5.17) is obtained by deleting the remainder term involving ξ_n:

$$y_{n+1} = y_n + h\,T_m(x_n, y_n), \tag{5.18}$$

where

$$T_m(x_n, y_n) = f(x_n, y_n) + \frac{h}{2!}f'(x_n, y_n) + \cdots + \frac{h^{m-1}}{m!}f^{(m-1)}(x_n, y_n). \tag{5.19}$$

Remark 5.14.

- $m = 1 \Rightarrow y_{n+1} = y_n + hf(x_n, y_n)$
  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 5.15. Consider the initial-value problem

$$y' = y - x^3 + x + 1, \quad y(0) = 0.5. \tag{5.20}$$

(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

$$\begin{aligned} T_3(x, y) &= f(x, y) + \frac{h}{2}f'(x, y) + \frac{h^2}{6}f''(x, y) \\ &= y - x^3 + x + 1 + \frac{h}{2}(y - x^3 - 3x^2 + x + 2) \\ &\quad + \frac{h^2}{6}(y - x^3 - 3x^2 - 5x + 2) \end{aligned} \tag{5.21}$$

For m large, the computation of T_m is time-consuming and cumbersome.

Part (b):

Maple-code

```
T3 := y - x^3 + x + 1 + 1/2*h*(-x^3 - 3*x^2 + x + y + 2)
      + 1/6*h^2*(-x^3 - 3*x^2 - 5*x + y + 2):
h := 1/2:
x0 := 0: y0 := 1/2:
y1 := y0 + h*eval(T3, [x = x0, y = y0])
                              155
                              ---
                              96
y2 := y1 + h*eval(T3, [x = x0 + h, y = y1])
                             16217
                             -----
                             4608
evalf(%)
                          3.519314236

exacty := x -> 4 + 5*x + 3*x^2 + x^3 - 7/2*exp(x):
exacty(1)
                              7
                         13 - - exp(1)
                              2
evalf(%)
                          3.486013602
#The absolute error:
evalf(abs(exacty(1) - y2))
                          0.033300634
```

5.3. Runge-Kutta Methods

Note: What we are going to do is to solve the *initial value problem* (IVP):

$$\begin{cases} y' &= f(x, y), \\ y(x_0) &= y_0. \end{cases} \quad \text{(IVP)} \tag{5.22}$$

- The Taylor-series method of the preceding section has the drawback of requiring the **computation of derivatives of $f(x, y)$**. This is a tedious and time-consuming procedure for most cases, which makes the Taylor methods seldom used in practice.

Definition 5.16. 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 slopes, as follows:

$$y_{n+1} = y_n + h\,(w_1 K_1 + w_2 K_2 + \cdots + w_m K_m)\,, \tag{5.23}$$

where

(a) $w_j \geq 0$ and $w_1 + w_2 + \cdots + w_m = 1$

(b) K_j are recursive evaluations of the slope $f(x, y)$

(c) Need to determine w_j and other parameters to satisfy

$$w_1 K_1 + w_2 K_2 + \cdots + w_m K_m \approx T_m(x_n, y_n) + \mathcal{O}(h^m) \tag{5.24}$$

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:

$$y_{n+1} = y_n + h\,(w_1 K_1 + w_2 K_2) \tag{5.25}$$

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

$$w_1 K_1 + w_2 K_2 \;=\; T_2(x_n, y_n) + \mathcal{O}(h^2) \;=\; f(x_n, y_n) + \frac{h}{2} f'(x_n, y_n) + \mathcal{O}(h^2). \tag{5.26}$$

Derivation: For the left-hand side of (5.25), 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 $y'' = f_x + f_y y' = f_x + f_y f$,

$$y(x+h) = y(x) + h\,f + \frac{h^2}{2}(f_x + f_y f) + \mathcal{O}(h^3). \tag{5.27}$$

On the other hand, the right-side of (5.25) 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

$$y + h(w_1 K_1 + w_2 K_2) \;=\; y + (w_1 + w_2) h\,f + h^2 (w_2 \alpha f_x + w_2 \beta f_y f) + \mathcal{O}(h^3). \tag{5.28}$$

The comparison of (5.27) and (5.28) drives the following result, for the second-order Runge-Kutta methods.

Results:

$$w_1 + w_2 = 1, \quad w_2\,\alpha = \frac{1}{2}, \quad w_2\,\beta = \frac{1}{2}. \tag{5.29}$$

Common Choices:

Algorithm 5.17.

I. $w_1 = w_2 = \frac{1}{2}$, $\alpha = \beta = 1$:
Then, the algorithm (5.25) becomes

$$y_{n+1} = y_n + \frac{h}{2}(K_1 + K_2), \tag{5.30}$$

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 (5.25) reads

$$y_{n+1} = y_n + h\,f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2} f(x_n, y_n)\right) \tag{5.31}$$

which is also known as the **Modified Euler method**.

It follows from (5.27) and (5.28) that the **local truncation error** for the above Runge-Kutta methods are $\mathcal{O}(h^3)$. Thus the **global error** becomes

$$\mathcal{O}(h^2). \tag{5.32}$$

5.3.2. Fourth-order Runge-Kutta method

Formulation:

$$y_{n+1} = y_n + h\,(w_1 K_1 + w_2 K_2 + w_3 K_3 + w_4 K_4) \tag{5.33}$$

where

$$\begin{aligned}
K_1 &= f(x_n, y_n)\\
K_2 &= f(x_n + \alpha_1 h, y_n + \beta_1 h\, K_1)\\
K_3 &= f(x_n + \alpha_2 h, y_n + \beta_2 h\, K_1 + \beta_3 h\, K_2)\\
K_4 &= f(x_n + \alpha_3 h, y_n + \beta_4 h\, K_1 + \beta_5 h\, K_2 + \beta_6 h\, K_3)
\end{aligned}$$

Requirement: Determine w_j, α_j, β_j such that

$$w_1 K_1 + w_2 K_2 + w_3 K_3 + w_4 K_4 = T_4(x_n, y_n) + \mathcal{O}(h^4)$$

The most common choice:

Algorithm 5.18. Fourth-order Runge-Kutta method (RK4): The most commonly used set of parameter values yields

$$y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \tag{5.34}$$

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

$$\frac{h^5}{5!} y^{(5)}(\xi_n), \tag{5.35}$$

for some $\xi_n \in [x_n, x_{n+1}]$. Thus the **global error** reads, for some $\xi \in [x_0, T]$,

$$\frac{(T - x_0)h^4}{5!} y^{(5)}(\xi). \tag{5.36}$$

Example 5.19. Use RK4 to solve the initial-value problem

$$y' = y - x^3 + x + 1, \quad y(0) = 0.5. \tag{5.37}$$

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 (5.37):

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 5.20.

- **Accuracy** of numerical methods can be improved **by decreasing the step size**.
- Decreasing the step size ≈ 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**:

$$\begin{aligned} &y''(t) + \frac{\kappa}{m}y = \frac{F_0}{m}\cos(\mu t), \\ &y(0) = c_0, \quad y'(0) = 0, \end{aligned} \tag{5.38}$$

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 (5.38):

$$y(t) = A\cos(\omega t) + \frac{F_0}{m(\omega^2 - \mu^2)}\cos(\mu t), \tag{5.39}$$

  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 (5.38) as

$$\begin{aligned} y_1' &= -\omega y_2, & y_0(0) &= c_0, \\ y_2' &= \omega y_1 - \frac{F_0}{m\omega}\cos(\mu t), & y_2(0) &= 0. \end{aligned} \tag{5.40}$$

  We will deal with details of *High-Order Equations & Systems of Differential Equations* in § 5.6 on page 194.
- The motion is periodic only if μ/ω is a rational number. We choose

$$m = 1,\ \ F_0 = 40,\ \ A = 1,\ \ \omega = 4\pi,\ \ \mu = 2\pi. \quad (\Rightarrow c_0 \approx 1.33774) \tag{5.41}$$

  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 (5.40)-(5.41).

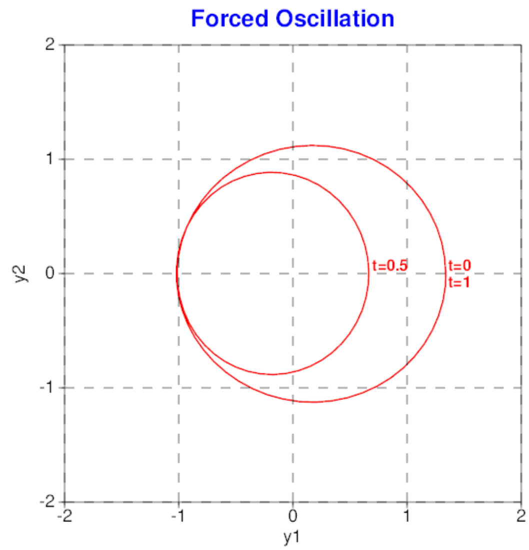


Figure 5.3: The trajectory of the mass satisfying (5.40)-(5.41).

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
$$|\mathbf{y}^h_{N_t} - \mathbf{y}(1)| = \left(\left[y^h_{1,N_t} - y_1(1)\right]^2 + \left[y^h_{2,N_t} - y_2(1)\right]^2\right)^{1/2}, \tag{5.42}$$
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
$$\alpha := \frac{\ln(E(2h)/E(h))}{\ln 2}, \tag{5.43}$$
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 5.21. (Order of Convergence): Let's assume that the algorithm under consideration produces error in $\mathcal{O}(h^\alpha)$. Then, we may write

$$E(h) = C\,h^\alpha, \tag{5.44}$$

where h is the grid size. When the grid size is ph, the error will become

$$E(ph) = C\,(ph)^\alpha. \tag{5.45}$$

It follows from (5.44) and (5.45) that

$$\frac{E(ph)}{E(h)} = \frac{C\,(ph)^\alpha}{C\,h^\alpha} = p^\alpha. \tag{5.46}$$

By taking a logarithm, one can solve the above equation for the **order of convergence** α:

$$\alpha = \frac{\ln(E(ph)/E(h))}{\ln p}. \tag{5.47}$$

When $p = 2$, the above becomes (5.43).

5.5. Multi-step Methods

The problem: The first-order initial value problem (IVP)

$$\begin{cases} y' = f(x, y), \\ y(x_0) = y_0. \end{cases} \quad \text{(IVP)} \tag{5.48}$$

Numerical Methods:

- **Single-step/Starting methods**: Euler's method, Modified Euler's, Runge-Kutta methods
- **Multi-step/Continuing methods**: Adams-Bashforth-Moulton

Definition 5.22. 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

$$\begin{aligned} y_{n+1} &= a_1 y_n + a_2 y_{n-1} + \cdots + a_m y_{n+1-m} \\ &\quad + h[b_0 f(x_{n+1}, y_{n+1}) + b_1 f(x_n, y_n) + \cdots + b_m f(x_{n+1-m}, y_{n+1-m})]. \end{aligned} \tag{5.49}$$

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 5.23. (Fourth-order multi-step methods):

Let $y_i' = f(x_i, y_i)$.

- **Adams-Bashforth method** (explicit)

$$y_{n+1} = y_n + \frac{h}{24}(55y_n' - 59y_{n-1}' + 37y_{n-2}' - 9y_{n-3}') \tag{5.50}$$

- **Adams-Moulton method** (implicit)

$$y_{n+1} = y_n + \frac{h}{24}(9y_{n+1}' + 19y_n' - 5y_{n-1}' + y_{n-2}') \tag{5.51}$$

- **Adams-Bashforth-Moulton method** (predictor-corrector)

$$\begin{aligned} y_{n+1}^* &= y_n + \frac{h}{24}(55y_n' - 59y_{n-1}' + 37y_{n-2}' - 9y_{n-3}') \\ y_{n+1} &= y_n + \frac{h}{24}(9{y'}_{n+1}^* + 19y_n' - 5y_{n-1}' + y_{n-2}') \end{aligned} \tag{5.52}$$

  where ${y'}_{n+1}^* = f(x_{n+1}, y_{n+1}^*)$

Remark 5.24.

- 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 [1].

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)

$$\begin{cases} y'' = f(x, y, y'), & x \in [x_0, T] \\ y(x_0) = y_0, \quad y'(x_0) = u_0, \end{cases} \tag{5.53}$$

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:

$$\begin{cases} y' = u, & y(x_0) = y_0, \\ u' = f(x, y, u), & u(x_0) = u_0, \end{cases} \quad x \in [x_0, T] \tag{5.54}$$

Remark 5.25.

- The right-side of the DEs involves no derivatives.
- The system (5.54) can be solved by one of the numerical methods (we have studied), after modifying it for vector functions.

Example 5.26. 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
```

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's 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:

$$\begin{cases} y' &= 1 + (x-y)^2, \quad 2 \le x \le 3, \\ y(2) &= 1, \end{cases} \tag{5.55}$$

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 $y(3) = 2.5$.

5.4. C Now, solve the problem in the preceding exercise, (5.55), by implementing

(a) RK4
(b) Adams-Bashforth-Moulton method

Use $h = 0.05$ and compare the accuracy.

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

$$\begin{cases} u_1' &= u_2 - u_3 + t, \quad u_1(0) = 1, \\ u_2' &= 3t^2, \qquad\qquad u_2(0) = 1, \qquad (0 \le t \le 1) \\ u_3' &= u_2 + e^{-t}, \qquad u_3(0) = -1, \end{cases} \tag{5.56}$$

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.

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.

6.1. Systems of Linear Equations

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

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

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 1.36) 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 1.41) *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. *$\det A \neq 0$.*

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 1.50.*

6.1.2. Numerical solutions of differential equations

Consider the following differential equation:

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

where

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

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's series expansion** 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

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

- 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

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

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.

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

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

and

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

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

Dirichlet Boundary Condition:

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

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

- Then, the corresponding algebraic system can be formulated as

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

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

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

and

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

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 $i = 1, \cdots, n$.*

6.2.1. 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
   if L(i,i)==0, set error flag, exit
   b(i) = b(i)/L(i,i)
end
```
(6.19)

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

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

6.2.2. 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

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

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);
    b(1:i-1)=b(1:i-1)-U(1:i-1,i)*x(i);
end
```
(6.23)

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

6.3. Gauss Elimination

— a very basic algorithm for solving $A\mathbf{x} = \mathbf{b}$

The algorithms developed here produce (in the absence of rounding errors) the unique solution of $A\mathbf{x} = \mathbf{b}$, whenever $A \in \mathbb{R}^{n\times n}$ is nonsingular.

Strategy 6.12. (Gauss elimination):

- First, transform the system $A\mathbf{x} = \mathbf{b}$ to an equivalent system $U\mathbf{x} = \mathbf{y}$, where U is upper-triangular;
- then Further transform the system $U\mathbf{x} = \mathbf{y}$ to get $\mathbf{x}$.
  - It is convenient to represent $A\mathbf{x} = \mathbf{b}$ by an augmented matrix $[A|\mathbf{b}]$; each equation in $A\mathbf{x} = \mathbf{b}$ 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 $[\hat{A}\,|\,\hat{\mathbf{b}}]$ is obtained from $[A\,|\,\mathbf{b}]$ by elementary row operations (EROs), then systems $[A\,|\,\mathbf{b}]$ and $[\hat{A}\,|\,\hat{\mathbf{b}}]$ 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):

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

Figure 6.1

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.

Example 6.17. Let $A = \begin{bmatrix} 1 & -2 & 1 \\ 2 & -2 & 3 \\ -3 & 2 & 0 \end{bmatrix}$.

a) Reduce it to an echelon matrix, using *replacement operations*.
b) Express the replacement operations as elementary matrices.
c) Find their inverse.

Algorithm 6.18. (An LU Factorization Algorithm) *The derivation introduces an LU factorization: Let $A \in \mathbb{R}^{m\times n}$. Then*

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

Remark 6.19. Let E_1 and E_2 be elementary matrices that correspond to replacement operations occurred in the LU Factorization Algorithm. Then $E_1 E_2 = E_2 E_1$ and $E_1^{-1} E_2^{-1} = E_2^{-1} E_1^{-1}$.

Example 6.20. Find the LU factorization of

$$A = \begin{bmatrix} 4 & 3 & -5 \\ -4 & -5 & 7 \\ 8 & 8 & -7 \end{bmatrix}.$$

Theorem 6.21. (**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

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

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

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

Here we have used the assumption that U is nonsingular and so is U_{11}.

(1) $\Rightarrow$ **(2)**: It can be proved by induction on n. □

Example 6.22. Find the LU factorization of $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$.

Solution. (Practical Implementation):

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

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

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

Note: it is easy to verify that $A = LU$.

Example 6.23. *Find the LU factorization of*

$$A = \begin{bmatrix} 2 & -1 \\ 6 & 5 \\ -10 & 3 \\ 12 & -2 \end{bmatrix}.$$

Solution.

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

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

Note: U has the same size as A, that is, its size is 4×2. L is a square matrix and is a unit lower triangular matrix.

Remark 6.24. For an $n \times n$ **dense matrix** A (with most entries nonzero) with n moderately large.

- Computing an LU factorization of A takes about $2n^3/3$ flops† ($\sim$ row reducing $[A\ \mathbf{b}]$), while finding A^{-1} requires about $2n^3$ flops.
- Solving $L\mathbf{y} = \mathbf{b}$ and $U\mathbf{x} = \mathbf{y}$ requires about $2n^2$ flops, because any $n \times n$ triangular system can be solved in about n^2 flops.
- Multiplying $\mathbf{b}$ by A^{-1} also requires about $2n^2$ flops, but the result may not as accurate as that obtained from L and U (due to round-off errors in computing A^{-1} & $A^{-1}\mathbf{b}$).
- If A is **sparse** (with mostly zero entries), then L and U may be sparse, too. On the other hand, A^{-1} is likely to be **dense**. In this case, a solution of $A\mathbf{x} = \mathbf{b}$ with LU factorization is much faster than using A^{-1}.

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

6.3.2. 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

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

In the following couple of examples, LU is given.

Example 6.25. Let $A = \begin{bmatrix} 1 & 4 & -2 \\ 2 & 5 & -3 \\ -3 & -18 & 16 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -12 \\ -14 \\ 64 \end{bmatrix}$;

$$A = LU \stackrel{\Delta}{=} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 4 & -2 \\ 0 & -3 & 1 \\ 0 & 0 & 8 \end{bmatrix}.$$

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:

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$;
(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$.

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

$$[L \,\vdots\, \mathbf{b}] = \begin{bmatrix} 1 & 0 & 0 & \vdots & -12 \\ 2 & 1 & 0 & \vdots & -14 \\ -3 & 2 & 1 & \vdots & 64 \end{bmatrix} \to \cdots \to \begin{bmatrix} 1 & 0 & 0 & \vdots & -12 \\ 0 & 1 & 0 & \vdots & 10 \\ 0 & 0 & 1 & \vdots & 8 \end{bmatrix} = [I \,\vdots\, \mathbf{y}]. \tag{6.31}$$

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

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

Thus, $\mathbf{x} = [2, -3, 1]^T$. □

Example 6.26. Let $A = \begin{bmatrix} 5 & 4 & -2 & -3 \\ 15 & 13 & 2 & -10 \\ -5 & -1 & 28 & 3 \\ 10 & 10 & 8 & -8 \end{bmatrix}$, $\mathbf{b} = \begin{bmatrix} -10 \\ -29 \\ 30 \\ -22 \end{bmatrix}$;

$$A = LU \overset{\Delta}{=} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ -1 & 3 & 1 & 0 \\ 2 & 2 & -2 & 1 \end{bmatrix} \begin{bmatrix} 5 & 4 & -2 & -3 \\ 0 & 1 & 8 & -1 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 6 \end{bmatrix}.$$

Use the LU factorization of A to solve $A\mathbf{x} = \mathbf{b}$.

Solution.

(1) Solve $L\mathbf{y} = \mathbf{b}$ for $\mathbf{y}$:

$$[L \,\vdots\, \mathbf{b}] = \begin{bmatrix} 1 & 0 & 0 & 0 & \vdots & -10 \\ 3 & 1 & 0 & 0 & \vdots & -29 \\ -1 & 3 & 1 & 0 & \vdots & 30 \\ 2 & 2 & -2 & 1 & \vdots & -22 \end{bmatrix} \to \cdots \to \begin{bmatrix} 1 & 0 & 0 & 0 & \vdots & -10 \\ 0 & 1 & 0 & 0 & \vdots & 1 \\ 0 & 0 & 1 & 0 & \vdots & 17 \\ 0 & 0 & 0 & 1 & \vdots & 30 \end{bmatrix} = [I \,\vdots\, \mathbf{y}].$$

(2) Solve $U\mathbf{x} = \mathbf{y}$ for $\mathbf{x}$:

$$[U \,\vdots\, \mathbf{y}] = \begin{bmatrix} 5 & 4 & -2 & -3 & \vdots & -10 \\ 0 & 1 & 8 & -1 & \vdots & 1 \\ 0 & 0 & 2 & 3 & \vdots & 17 \\ 0 & 0 & 0 & 6 & \vdots & 30 \end{bmatrix} \to \cdots \to \begin{bmatrix} 1 & 0 & 0 & 0 & \vdots & 3 \\ 0 & 1 & 0 & 0 & \vdots & -2 \\ 0 & 0 & 1 & 0 & \vdots & 1 \\ 0 & 0 & 0 & 1 & \vdots & 5 \end{bmatrix} = [I \,\vdots\, \mathbf{x}].$$

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

6.3.3. Gauss elimination with pivoting

Definition 6.27. 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 6.28. Show that if P is permutation matrix, then $P^T P = PP^T = I$. Thus P is nonsingular and

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

Solution.

Lemma 6.29. Let P and Q be $n \times n$ **permutation matrices** and $A \in \mathbb{R}^{n \times n}$. Then

(a) PA is A with its rows permuted
AP is A with its columns permuted.

(b) $det(P) = \pm 1$.

(c) PQ is also a permutation matrix.

Example 6.30. Let $A \in \mathbb{R}^{n\times n}$, and let $\widehat{A}$ be a matrix obtained from scrambling the rows of A. Show that there is a unique permutation matrix $P \in \mathbb{R}^{n\times n}$ such that $\widehat{A} = PA$.

Hint: Consider the row indices in the scrambled matrix $\widehat{A}$, say $\{k_1, k_2, \cdots, k_n\}$. (This means that for example, the first row of $\widehat{A}$ is the same as the k_1-th row of A.) Use the index set to define a permutation matrix P.

Proof. (Self-study)

Theorem 6.31. ***Gauss elimination with partial pivoting***, *applied to $A \in \mathbb{R}^{n\times n}$, produces a unit lower-triangular matrix L with $|\ell_{ij}| \le 1$, an upper-triangular matrix U, and a permutation matrix P such that*

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

or, equivalently,

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

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

Algorithm 6.32. Solving $A\mathbf{x} = \mathbf{b}$ using *Gauss elimination with partial pivoting*:

1. Factorize A into $A = P^T LU$, where
   P = permutation matrix,
   L = unit lower triangular matrix
   (i.e., with ones on the diagonal),
   U = nonsingular upper-triangular matrix.

2. Solve $P^T LU\mathbf{x} = \mathbf{b}$
   (a) $LU\mathbf{x} = P\mathbf{b}$ (permuting $\mathbf{b}$)
   (b) $U\mathbf{x} = L^{-1}(P\mathbf{b})$ (forward substitution)
   (c) $\mathbf{x} = U^{-1}(L^{-1}P\mathbf{b})$ (back substitution)

In practice:

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

Theorem 6.33. *If A is nonsingular, then there exist permutations P_1 and P_2, a unit lower triangular matrix L, and a nonsingular upper-triangular matrix U such that*

$$P_1 A P_2 = LU.$$

Only one of P_1 and P_2 is necessary.

Remark 6.34. $P_1 A$ reorders the rows of A, AP_2 reorders the columns, and $P_1 A P_2$ reorders both. Consider

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

- 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 6.35. 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 6.36. Find the LU factorization of $A = \begin{bmatrix} 3 & -1 & 1 \\ 9 & 1 & 2 \\ -6 & 5 & -5 \end{bmatrix}$, which is considered in Example 6.22.

Solution. **(Without pivoting)**

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

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

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

(With partial pivoting)

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

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

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

$PA = LU$

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

6.3.4. Calculating A^{-1}

Algorithm 6.37. (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.)

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

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

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

(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.)

Bibliography

[1] G. DAHLQUIST, *A special stability problem for linear multistep methods*, BIT, 3 (1963), pp. 27–43.

[2] F. GOLUB AND C. V. LOAN, *Matrix Computations, 3rd Ed.*, The Johns Hopkins University Press, Baltimore, 1996.

Index