# Numerical Analysis

Cesar O. Aguilar
Department of Mathematics
State University of New York at Geneseo

# Contents

# 1

# Introduction

## 1.1 Taylor's Theorem

Many problems in science and engineering cannot be solved with explicit formulas yielding exact results, i.e. solved analytically. Numerical analysis is concerned with developing algorithms that produce approximate solutions and establish estimates on the accuracy of the solution.

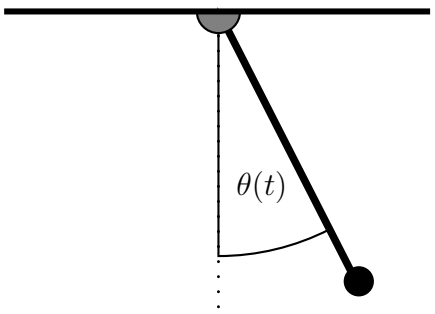



Figure 1.1: Pendulum swinging under the action of gravity

**Example 1.1.** Consider a pendulum free to hang from one of its end and under the force of gravity, see Figure 1.1. Let $\theta(t)$ denote the angle the rod makes with the vertical measured counter clockwise. Then applying Newton's Law's, we obtain the differential equation

$$\theta''(t) = -\sin(\theta(t)).$$

For every initial condition $\theta(0)$ and $\theta'(0)$ there exists a unique function $\theta(t)$ defined for all $t \in \mathbb{R}$ and solving the above differential equation. Analytic expression for $\theta(t)$ in terms of elementary functions from calculus are unknown.

Consider the problem of computing $\cos(0.01)$. Recall that if $(p, q)$ are the Cartesian coordinates of the point on the unit circle such that the line from the origin $(0, 0)$ to $(p, q)$

makes an angle $x \in [0, 2\pi)$ with the $x$-axis then $p = \cos(x)$. However, in general there are no closed form solutions for $\cos(x)$, except in very few cases such as $\cos(0) = 1$, $\cos(\pi/2) = 0$, or $\cos(\pi/3) = 1/2$. However, we can use Taylor's Theorem to approximate $\cos(0.01)$ and also give a bound on the error in the approximation.

**Theorem 1.1: Taylor's Theorem**

Let $f : [a, b] \to \mathbb{R}$ be such that $f, f^{(1)}, \ldots, f^{(n)}$ are continuous on $[a, b]$ and that $f^{(n+1)}$ exists on $(a, b)$. Let $x_0 \in [a, b]$. Then for any $x \in [a, b]$ there exists a number $\xi(x)$ between $x_0$ and $x$ such that

$$f(x) = P_n(x) + R_n(x)$$

where

$$P_n(x) = f(x_0) + f^{(1)}(x_0)(x - x_0) + \frac{1}{2!}f^{(2)}(x_0)(x - x_0)^2 + \cdots + \frac{1}{n!}f^{(n)}(x_0)(x - x_0)^n$$

and

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

The term $P_n(x)$ is called the **$n$-th order Taylor polynomial of $f$ centered at** $x_0$ and $R_n(x)$ is called the **remainder/truncation term**. If $f(x_0), f^{(1)}(x_0), \ldots, f^{(n)}(x_0)$ are easy to compute and $x$ is close to $x_0$, for example $|x - x_0| < 1$, then $R_n(x)$ will be small provided that $n$ is large enough and $f^{(n+1)}(\xi(x))$ is not too big. In fact, if for instance $|f^{(n+1)}(z)| \leq M$ for all $z \in [a, b]$ then

$$\begin{aligned} |f(x) - P_n(x)| &= \left| \frac{1}{(n+1)!}f^{(n+1)}(\xi(x))(x - x_0)^{n+1} \right| \\ &= \frac{1}{(n+1)!}|f^{(n+1)}(\xi(x))||x - x_0|^{n+1} \\ &\leq \frac{1}{(n+1)!}M|x - x_0|^{n+1} \end{aligned}$$

Hence, if $|x - x_0|$ is small then the error term $|R_n(x)| = |f(x) - P_n(x)|$ becomes small as $n$ increases. The next example illustrates the use of Taylor's theorem.

**Example 1.2.** Let $f(x) = \cos(x)$ and let $x_0 = 0$.

(a) Determine $P_2(x)$ at $x_0$ and use it to approximate $\cos(0.01)$. Determine an upper bound for $|\cos(0.01) - P_2(0.01)|$.

(b) Repeat with $P_3(x)$.

*Solution.* We have that $f^{(1)} = -\sin(x)$, $f^{(2)}(x) = -\cos(x)$, $f^{(3)}(x) = \sin(x)$, and $f^{(4)}(x) = \cos(x)$. Hence $f^{(1)}(0) = 0$, $f^{(2)}(0) = -1$, and $f^{(3)}(0) = 0$. Hence,

$$P_2(x) = 1 - \frac{1}{2}x^2.$$

Thus $P_2(0.01) = 1 - \frac{1}{2}(0.01)^2 = 0.99995$ is an approximation to $\cos(0.01)$. To estimate the error in the approximation we determine a bound on $R_2(0.01)$. Now, $R_2(x) = \frac{1}{3!}\sin(\xi(x))(x-0)^3$, where $\xi(x)$ is in between $x$ and $x_0 = 0$, and since $|\sin(z)| \leq 1$ for all $z \in \mathbb{R}$ it follows that

$$\begin{aligned}
|\cos(0.01) - P_2(0.01)| &= |R_2(0.01)| \\
&= \left|\frac{1}{6}\sin(\xi(0.01))0.01^3\right| \\
&= \left|\frac{1}{6}|\sin(\xi(0.01))||0.01|^3\right| \\
&\leq \frac{1}{6}0.01^3 \\
&= \frac{1}{6}10^{-6}
\end{aligned}$$

Hence, $P_2(0.01) = 0.99995$ approximates $\cos(0.01)$ to within six decimal places. We can actually do better. Indeed, since $|\sin(z)| \leq |z|$ for all $z \in \mathbb{R}$ and $0 < \xi(0.01) < 0.01$ it follows that

$$\begin{aligned}
|\cos(0.01) - P_2(0.01)| &= |R_2(0.01)| \\
&= \left|\frac{1}{6}\sin(\xi(0.01))0.01^3\right| \\
&= \frac{1}{6}|\sin(\xi(0.01))||0.01|^3 \\
&\leq \frac{1}{6}|\xi(0.01)||0.01|^3 \\
&\leq \frac{1}{6}(0.01)^{-4} \\
&= \frac{1}{6}10^{-8}
\end{aligned}$$

Hece, $P_2(0.01)$ approximates $\cos(0.01)$ to within eight decimal places. Now we consider $P_3(x)$. It is straightforward to compute that

$$P_3(x) = 1 - \frac{1}{2}x^2 \quad \text{and} \quad R_3(x) = \frac{1}{4!}\cos(\xi(x))x^4.$$

Thus, $P_3(0.01) = 0.999995$. Now, $|\cos(z)| \leq 1$ for all $z \in \mathbb{R}$ and therefore

$$\begin{aligned} |\cos(0.01) - P_3(0.01)| &= \left| \frac{1}{4!} \cos(\xi(0.01)) 0.01^4 \right| \\ &\leq \frac{1}{24} 0.01^4 \\ &< 0.42 \cdot 10^{-9}. \end{aligned}$$

Hence, 0.99995 approximates $\cos(0.01)$ to within nine decimal places. □

The previous example illustrates two main objectives with numerical analysis, namely:
(a) Find an approximation to the solution of a given problem.
(b) Determine a bound for the error of the approximation.

**Example 1.3.** Let $f(x) = \sqrt{x}$. Find $P_3(x)$ centered at $x_0 = 1$ and use it to approximate $\sqrt{1.25}$. Give an upper bound for $|f(1.25) - P_3(1.25)|$.

*Solution.* We compute that $f'(x) = 1/2x^{-1/2}$, $f^{(2)}(x) = -1/4x^{-3/2}$, $f^{(3)}(x) = 3/8x^{-5/2}$, and $f^{(4)}(x) = -15/16x^{-7/2}$. Hence,

$$P_3(x) = 1 + \frac{1}{2}(x-1) - \frac{1}{8}(x-1)^2 + \frac{1}{16}(x-1)^3 \quad \text{and} \quad R_3(x) = -\frac{15}{16 \cdot 4!}(\xi(x))^{-7/2}(x-1)^4$$

Then $P_3(1.25) = 1.1181640625$. Now since $1 < \xi(1.25) < 1.25$ we have that $(\xi(x))^{-7/2} < 1$ and therefore

$$\begin{aligned} |\sqrt{1.25} - P_3(1.25)| &= \frac{15}{16} |\xi(1.25)|^{-7/2} \frac{0.25^4}{4!} \\ &\leq \frac{0.25^4}{4!} \\ &\approx 0.16 \cdot 10^{-3} \end{aligned}$$

Hence, $P_3(1.25) = 1.1181640625$ approximates $\sqrt{1.25}$ to within three decimal places. In fact, $\sqrt{1.25} = 1.1180339887\ldots$. □

## 1.2 Sequences and Round-Off Error

Recall that by a **sequence of real numbers** we mean an infinite list $(x_1, x_2, x_3, \ldots)$ such that each number $x_i \in \mathbb{R}$. We use the short-hand notation $\{x_n\}_{n=1}^{\infty}$ or $(x_n)$ to denote a sequence. The $n$-th term of the sequence $(x_n)$ is $x_n$. Technically speaking, a sequence is a function from $\mathbb{N}$ to $\mathbb{R}$ but we side-step this technicality.

**Definition 1.2**

The sequence $(x_n)$ **converges to the number** $L$ if for any arbitrary number $\varepsilon > 0$ there exists a term $x_N$ in the sequence such that $|x_N - L| < \varepsilon$, $|x_{N+1} - L| < \varepsilon$, $|x_{N+2} - L| < \varepsilon$, and the same holds for all subsequent terms. In other words, $|x_n - L| < \varepsilon$ for all $n \geq N$. In this case, we write that

$$\lim_{n\to\infty} x_n = L \quad \text{or} \quad x_n \to L.$$

**Example 1.4.** Consider the sequence $x_n = \frac{1}{n^2} + 2$. We claim that $x_n \to 2$. Let $\varepsilon > 0$ be arbitrary. When does $|x_n - 2| < \varepsilon$? That is, when does

$$|x_n - 2| = \left|\frac{1}{n^2}\right| < \varepsilon \ ?$$

Now, $\frac{1}{n^2} < \varepsilon$ when $\frac{1}{\sqrt{\varepsilon}} < n$. Hence, if we choose $N$ to be the greatest integer such that $\frac{1}{\sqrt{\varepsilon}} < N$ then $|x_n - 2| < \varepsilon$ for all $n \geq N$. For instance, if $\varepsilon = 0.001$ then $\frac{1}{\sqrt{0.001}} \approx 31.6$, and thus if $N = 32$ then $|x_n - 2| < 0.001$ for all $n \geq 32$.

Sequences arise naturally in numerical analysis when solving problems using iteration or recursion. At this point, we will use sequences to understand the potential undesirable side-effects of numerical **round-off** error. In numerical computations with a computer, round-off error is unavoidable but it is important to understand if round-off error will yield a useless approximation. Roughly speaking, round-off error occurs when we use computers to store numbers with finite precision and use them to perform computations. Every real number $x$ can be written in the form

$$x = \pm 0.d_1 d_2 \cdots d_k \cdots \times 10^n$$

where $1 \leq d_1 \leq 9$ and $0 \leq d_i \leq 9$ for each $i = 2, 3, \ldots$. Since a computer has a finite amount of memory, the sequence of numbers $d_1, d_2, \ldots$ cannot all be stored and some sort of approximation must be done to represent $x$. The **floating-point form** of $x$, denoted by $fl(x)$, is obtained by terminating the sequence $d_1, d_2, \ldots$ at some $k$ decimal place. There are two common ways that the termination is done. The first is called **chopping** and consists of simply discarding the digits $d_{k+1}, d_{k+2}, \ldots$. This produces the floating-point form

$$fl(x) = 0.d_1 d_2 \cdots d_k \times 10^n.$$

The other method, called **rounding**, is obtained by adding $5 \times 10^{n-(k+1)}$ to $x$ and then chopping. If $d_{k+1} < 5$ then this amounts to rounding-down at the $k$th decimal plane, that is, we simply discard all digits after $d_k$. If $d_{k+1} \geq 5$ then we add 1 to $d_k$ and then chop at the $k$th decimal place.

**Example 1.5.** Consider $x = 3.102399654\ldots = 0.3102399654\ldots \times 10^1$. The seven-digit floating-point form of $x$ obtained by chopping is

$$fl(x) = 0.3102399 \times 10^1 = 3.102399$$

whereas the seven-digit floating-point form via rounding is

$$fl(x) = 0.3102400 \times 10^1 = 3.102400$$

since $d_7 = 6 \geq 5$.

When we measure error, it is important to take into account the overall magnitude of the quantity being approximated. For example, an error of $\varepsilon = 0.5$ in approximating say $p = 3558$ is much more acceptable than the same error in approximating $p = 3$.

**Definition 1.3**

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

**Example 1.6.** Let $x = 5/7 = 0.\overline{714285}$ and let $u = 0.714251$. Suppose that we use 5-digit chopping to store numbers. Then $fl(x) = 0.71428 \times 10^0$ and $fl(u) = 0.71425 \times 10^0$. Consider the error in computing $p = x - u$ with 5-digit arithmetic and floating-point representation from chopping. We have that $fl(x) - fl(u) = 0.00003 = 0.3 \times 10^{-4}$ and thus

$$p^* = fl(fl(x) - fl(u)) = 0.3 \times 10^{-4}.$$

Now

$$p = x - u = 0.347143\ldots \times 10^{-4}$$

so that

$$|p - p^*| = 0.47143\ldots \times 10^{-5}.$$

Although the absolute error is small the relative error is

$$\frac{|p - p^*|}{|p|} = \left| \frac{0.47143 \times 10^{-5}}{0.347143 \times 10^{-4}} \right| \approx 0.136.$$

Considering the magnitudes of $p$ and $p^*$, this is a large relative error.

**Example 1.7.** Use three-digit rounding arithmetic to compute $\frac{2}{9} \times \frac{9}{7}$. Compute the absolute and relative error with the exact value determined to at least five decimal places.

*Solution.* Let $x = \frac{2}{9} = 0.\overline{2222}$ and let $y = \frac{9}{7} = 1.285714286\ldots = 0.128571\ldots \times 10^1$. Then using three-digit rounding, $fl(x) = 0.222$ and $fl(y) = 1.29$. Then $fl(x) \times fl(y)) = 0.28638$, and therefore $p^* = fl(fl(x) \times fl(y)) = 0.286$ is our approximation of $p = xy = 0.28571\ldots$. The absolute error is $|p - p^*| = 0.286 \times 10^{-3}$ and the relative error is $\frac{|p-p^*|}{|p|} = 10^{-3}$. □

The following example illustrates how finite digit representation can lead to serious round-off error.

**Example 1.8.** Consider the sequence of numbers $(p_0, p_1, p_2, \ldots)$ defined recursively as

$$p_n = \frac{10}{3}p_{n-1} - p_{n-2}, \text{ for } n \geq 2$$

where $p_0, p_1$ are arbitrarily set. It can be shown that

$$p_n = c_1 \left(\frac{1}{3}\right)^n + c_2 3^n$$

satisfies the recursion equation, where the numbers $c_1$ and $c_2$ are determined by $p_0$ and $p_1$. If we set $p_0 = 1$ and $p_1 = \frac{1}{3}$ then $c_1 = 1$ and $c_2 = 0$. Hence, in this case $p_n = \left(\frac{1}{3}\right)^n$ and thus the true solution converges to zero exponentially.

Now suppose that floating-point representation and arithmetic up to five digits (using chopping) is used to compute $p_n$ recursively. Then $\hat{p}_0 = 1.0000$ and $\hat{p}_1 = 0.33333$ and then $\hat{c}_1 = 1.0000$ and $\hat{c}_2 = -0.12500 \times 10^{-5}$. The sequence generated is then

$$\hat{p}_n = 1.0000 \left(\frac{1}{3}\right)^n - 0.12500 \times 10^{-5}(3)^n.$$

The absolute round-off error is then

$$p_n - \hat{p}_n = 0.12500 \times 10^{-5}(3^n).$$

Hence the error grows exponentially as $n \to \infty$. In fact, using 16-digit arithmetic to compute $p_n$ we obtain the data shown in Table 1.1.

| $n$ | $\hat{p}_n$ |
|---|---|
| 10 | $1.6935 \times 10^{-5}$ |
| 20 | $3.0309 \times 10^{-8}$ |
| 30 | $1.7727 \times 10^{-3}$ |
| 40 | $1.0468 \times 10^{2}$ |
| 50 | $2.0604 \times 10^{6}$ |

Table 1.1: Instability due to round-off error

## 1.3 Algorithms and Stability

An **algorithm** is a well-defined and unambiguous computational procedure that takes a set of **inputs** and produces a set of **outputs**. An important property of an algorithm is the effect on the output it produces under small changes to its input. An algorithm is called **stable** if a small change in the initial input produces a small change in the output, otherwise, it is called **unstable**. Stability of an algorithm can be illustrated by how an approximation error propagates through a computation. Suppose an algorithm is executing and at some point of the computation an error $E_0 > 0$ is introduced. Let $E_n$ represent the magnitude of the error after $n$ subsequent operations. If

$$E_n \approx CnE_0$$

where $C$ is a constant, then the growth of the error is said to be **linear**. If

$$E_n \approx C^n E_0$$

where $C > 1$ then the growth of the error is called **exponential**. Algorithms whose error is linear are considered stable and those whose error is exponential are considered unstable.

**Example 1.9.** Recall that the solution to the recursion

$$p_n = \frac{10}{3}p_{n-1} - p_{n-2}, \text{ for } n \geq 2$$

is

$$p_n = c_1\left(\frac{1}{3}\right)^n + c_2 3^n$$

where the numbers $c_1$ and $c_2$ are determined by $p_0$ and $p_1$. If we set $p_0 = 1$ and $p_1 = \frac{1}{3}$ then $p_n = \left(\frac{1}{3}\right)^n$. Using five-digit floating-point representation we have that the computed approximation to $p_n$ is

$$\hat{p}_n = 1.0000\left(\frac{1}{3}\right)^n - 0.12500 \times 10^{-5}(3)^n.$$

An error is introduced in the computation at $\hat{p}_2 = 0.33333$. This error propagates as we perform the arithmetic operations defining the recursion. As shown before, the absolute round-off error is

$$|p_n - \hat{p}_n| = 0.12500 \times 10^{-5}(3^n).$$

Hence the error grows exponentially as $n \to \infty$.

**Example 1.10.** Consider the recursion

$$p_n = 2p_{n-1} - p_{n-2},\ n \geq 2.$$

With $p_0 = 1$ and $p_1 = 1/3$ the solution is $p_n = 1 - \frac{2}{3}n$. Using floating-point representation with five-digit accuracy, we compute that

$$\hat{p}_n = 1.0000 - 0.66667n$$

and therefore

$$|p_n - \hat{p}_n| = (0.66667 - 0.\overline{666666})n.$$

Hence the error propagates linearly when making a small change to the value $p_2$.

To give an overview of the fundamental steps of an algorithm we can use **pseudocode**, as the next examples illustrates.

**Example 1.11.** Use pseudocode to compute the average of the numbers $x_1, x_2, \ldots, x_N$, given that $N$ and the numbers $x_1, x_2, \ldots, x_N$ are the inputs.

*Solution.* The pseudocode is shown in Algorithm 1.1. Notice that the variable avg needs to be initialized before it can be used in the `for` loop. □

**Algorithm 1.1** Average

INPUT: $x_1, x_2, \ldots, x_N$

OUTPUT: The average value of $x_1, x_2, \ldots, x_N$, avg $= \frac{1}{N}\sum_{i=1}^{N} x_i$

1: **set** avg $= 0$
2: **for** $i = 1, 2, \ldots, N$ **do**
3: **set** avg $=$ avg $+ x_i$
4: **set** avg $= \frac{1}{N}$avg
5: **output** avg

**Example 1.12.** Consider the sequence $x_n = \frac{1}{\sqrt{n}}$. Clearly $\lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0$. Use pseudocode to determine the minimum $N$ such that $|x_n - 0| < \varepsilon$ for all $n \geq N$ for a given $\varepsilon > 0$.

*Solution.* The pseudocode is shown in Algorithm 1.2. Notice that we are using the fact that the sequence $x_n = \frac{1}{\sqrt{n}}$ is strictly decreasing so that if $\frac{1}{\sqrt{N}} < \varepsilon$ then $\frac{1}{\sqrt{n}} < \varepsilon$ for all $n \geq N$. □

**Algorithm 1.2** Sequence $x_n = \frac{1}{\sqrt{n}}$

INPUT: $\varepsilon > 0$

OUTPUT: Minimum $N$ such that $\frac{1}{\sqrt{n}} < \varepsilon$ for $n \geq N$

1: **set** $N = 1$

2: **while** $\frac{1}{\sqrt{N}} \geq \varepsilon$ **do**
    **set** $N = N + 1$

3: **output** $N$

**Example 1.13.** Let $P_N(x)$ be the Taylor polynomial of $e^x$ centered at $x_0 = 0$. Write pseudocode for an algorithm that returns the minimum value of $N$ required for

$$|e^{-0.5} - P_N(-0.5)| < 10^{-8}$$

given that $e^{-0.5}$ is known with infinite precision.

*Solution.* Recall that $P_N(x) = \sum_{k=0}^{N} \frac{1}{k!} x^k$. The pseudocode is shown in Algorithm 1.3.

**Algorithm 1.3** Maclaurin Polynomial of $e^x$

INPUT: $e^{-0.5}$, $10^{-8}$

OUTPUT: minimum $N$ such that $|e^{-0.5} - P_N(-0.5)| < 10^{-8}$

1: **set** $N = 0$, $P = 1$

2: **while** $|e^{-0.5} - P| \geq \varepsilon$ **do**
    **set** $N = N + 1$
    **set** $P = P + \frac{(-0.5)^N}{N!}$ (adds the next Taylor polynomial term)

3: **output** $N$

□

# 2

# Root-Finding in One Variable Equations

The point $p \in [a, b]$ is called a **zero** of the function $f : [a, b] \to \mathbb{R}$ if $f(p) = 0$. A zero $p$ of $f$ is also called a **root** of the equation $f(x) = 0$. In this section, we consider the very important problem of finding/approximating a zero of a function. Finding zeros/roots of functions/equations is one of the oldest problems in mathematics. For instance, the roots of the equation $ax^2 + bx + c = 0$ are given by the well-known quadratic formula $p = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$. In general, however, there are no explicit formulas for the roots of polynomials of degree five or higher in terms of the coefficients of the polynomial and using the usual algebraic operations and radicals. Hence, in general one must resort to numerical methods to find approximations to the roots in these cases and in cases involving more complex equations.

## 2.1 Bisection Method

The first root-finding method we consider is the Bisection method which is an application of the Intermediate Value Theorem.

**Theorem 2.1: Intermediate Value Theorem**

Let $f : [a, b] \to \mathbb{R}$ be a continuous function. Suppose that $f(a)$ and $f(b)$ are non-zero and have opposite signs, that is $f(a)f(b) < 0$. Then there exists $p \in (a, b)$ such that $f(p) = 0$.

Suppose that $f : [a, b] \to \mathbb{R}$ is continuous and $f(a)f(b) < 0$. Suppose for simplicity that $f$ has a unique zero $p$ in $(a, b)$. Consider the midpoint $p_1 = \frac{a+b}{2}$ of the interval $[a, b]$. If $f(p_1) = 0$ then we have found a zero of $f$, otherwise if $f(p_1) \neq 0$ then $f(p_1)$ has opposite sign with either $f(a)$ or $f(b)$. Suppose that without loss of generality that $f(a)$ and $f(p_1)$ have opposite sign, that is, that $f(a)f(p_1) < 0$. Then necessarily $p \in (a, p_1)$ by the IVT. Now let $a_2 = a$ and $b_2 = p_1$ and consider the midpoint $p_2 = \frac{a_2+b_2}{2}$. If $f(p_2) = 0$ then we have found a zero of $f$, otherwise if $f(p_2) \neq 0$ then $f(p_2)$ has opposite sign with either $f(a_2)$ or

$f(b_2)$. Suppose that $f(p_2)f(b_2) < 0$. Then necessarily $p \in (p_2, b_2)$ by the IVT. Let $a_3 = p_2$ and $b_3 = b_2$. We repeat this procedure with $[a_3, b_3]$ so that either $f(p_3) = 0$ or $p \in (p_3, b_3)$ or $p \in (a_3, p_3)$. Continuing in this way we obtain either $p = p_N$ for some $N$, or a sequence of distinct points $(p_1, p_2, p_3, \ldots)$ and a sequence of intervals $[a_n, b_n]$ such that $p \in (a_n, p_n)$ or $p \in (p_n, b_n)$ and thus

$$|p - p_n| < \frac{b_n - a_n}{2}. \tag{2.1}$$

For notational consistency, we have set $a_1 = a$ and $b_1 = b$. Now $b_2 - a_2 = \frac{b_1 - a_1}{2}$, $b_3 - a_3 = \frac{b_2 - a_2}{2} = \frac{b-a}{2^2}$, and by induction

$$b_n - a_n = \frac{b-a}{2^{n-1}}$$

for $n \geq 1$. Therefore,

$$|p - p_n| < \frac{b_n - a_n}{2} = \frac{b-a}{2^{n-1} \cdot 2} = \frac{b-a}{2^n}.$$

It follows that $\lim_{k\to\infty} p_n = p$ and thus we have generated a sequence $(p_n)$ that converges to the zero $p$ of $f$. We say that the sequence $(p_1, p_2, p_3, \ldots)$ is generated by the **Bisection method** for the function $f : [a, b] \to \mathbb{R}$. In summary, we have the following.

**Theorem 2.2**

Suppose that $f : [a, b] \to \mathbb{R}$ is a continuous function and $f(a)f(b) < 0$. Let $(p_1, p_2, p_3, \ldots)$ be the sequence generated by the Bisection method for $f$. Then $(p_n)$ converges to a zero $p \in (a, b)$ of the function $f$. In fact, for $n \geq 1$ it holds that

$$|p - p_n| < \frac{b-a}{2^n}.$$

Hence, according to inequality (2.1), if $n$ is such that

$$\frac{b_n - a_n}{2} = \frac{b-a}{2^n} < \varepsilon \tag{2.2}$$

then $|p - p_n| < \varepsilon$. Condition (2.2) can therefore be used to test when $p_n$ approximates $p$ to within $\varepsilon$. The pseudocode for an algorithm that implements the Bisection method in shown in Algorithm 2.1.

The condition

$$\frac{b_n - a_n}{2} < \varepsilon$$

in STEP 4 of Algorithm 2.1 guarantees that after a successful run of the algorithm we have $|p - p_n| < \varepsilon$. The next example illustrates the Bisection method.

**Example 2.1.** Consider the function $f(x) = x^3 + 3x^2 - 8$. Use the Bisection method to compute an approximation $p^*$ to a zero $p$ of $f$ so that $|p - p^*| < 1 \times 10^{-5}$.

**Algorithm 2.1** Bisection Method

INPUT: $a, b, \varepsilon, N_{\max}$

OUTPUT: Approximate zero $p^*$ of $f$ or message of failure

**set** $a_n = a$, $b_n = b$
**for** $n = 1, 2, \ldots, N_{\max}$ **do**
    **set** $p = \frac{b_n + a_n}{2}$
    **if** $f(p) = 0$ **or** $\frac{b_n - a_n}{2} < \varepsilon$ **do**
        **return** $p$
        **break**
    **else**
        **if** $f(a_n)f(p) < 0$ **do**
            **set** $b_n = p$
        **else**
            **set** $a_n = p$
**print**("Maximum number of iterations reached.")

*Solution.* We observe that $f(0) = -8$ and $f(2) = 12$. Hence, $f$ has a zero in the interval $[0, 2]$. Applying the Bisection method we obtain Table 2.1. We have that

$$\frac{1}{2}|b_{18} - a_{18}| = 0.762939 \times 10^{-5}$$

and therefore

$$|p - p_{18}| < \frac{1}{2}|b_{18} - a_{18}| = 0.762939 \times 10^{-5} < 1 \times 10^{-5}.$$

In fact, using Python's root finding function in the `numpy.polynomial.polynomial` module, we obtain that $|p - p_{18}| = 0.714 \times 10^{-5}$. However, one finds that the error $|p - p_{17}|$ is smaller: $|p - p_{17}| = 0.49 \times 10^{-6}$ but the algorithm does not stop at $N = 17$ since $\frac{1}{2}(b_{17} - a_{17}) = 1.5259 \times 10^{-5} > 1 \times 10^{-5}$. Hence, our stopping criteria for accuracy, namely $\frac{1}{2}(b_n - a_n) < \varepsilon$, may miss approximations to $p$ that are more accurate than what is actually produced. However, the Bisection method has the important property that our sequence $(p_n)$ converges to a zero of $f$ with error bound $|p - p_n| < \frac{b-a}{2^n}$.

□

We introduce some notation regarding the order of convergence of a sequence $(x_n)$ to $L$.

**Definition 2.3**

Suppose that $(\beta_n)$ is such that $\beta_n \to 0$ and $(x_n)$ converges to $L$. If there exists a number

| $n$ | $a_n$ | $b_n$ | $p_n$ | $\lvert p - p_n\rvert$ |
|---|---|---|---|---|
| 1 | 0.00000000 | 2.00000000 | 1.00000000 | 0.35530140 |
| 2 | 1.00000000 | 2.00000000 | 1.50000000 | 0.14469860 |
| 3 | 1.00000000 | 1.50000000 | 1.25000000 | 0.10530140 |
| 4 | 1.25000000 | 1.50000000 | 1.37500000 | 0.01969860 |
| 5 | 1.25000000 | 1.37500000 | 1.31250000 | 0.04280140 |
| 6 | 1.31250000 | 1.37500000 | 1.34375000 | 0.01155140 |
| 7 | 1.34375000 | 1.37500000 | 1.35937500 | 0.00407360 |
| 8 | 1.34375000 | 1.35937500 | 1.35156250 | 0.00373890 |
| 9 | 1.35156250 | 1.35937500 | 1.35546875 | 0.00016735 |
| 10 | 1.35156250 | 1.35546875 | 1.35351562 | 0.00178577 |
| 11 | 1.35351562 | 1.35546875 | 1.35449219 | 0.00080921 |
| 12 | 1.35449219 | 1.35546875 | 1.35498047 | 0.00032093 |
| 13 | 1.35498047 | 1.35546875 | 1.35522461 | 0.00007679 |
| 14 | 1.35522461 | 1.35546875 | 1.35534668 | 0.00004528 |
| 15 | 1.35522461 | 1.35534668 | 1.35528564 | 0.00001575 |
| 16 | 1.35528564 | 1.35534668 | 1.35531616 | 0.00001476 |
| 17 | 1.35528564 | 1.35531616 | 1.35530090 | 0.00000049 |
| 18 | 1.35530090 | 1.35531616 | 1.35530853 | 0.00000714 |

Table 2.1: Bisection method for $f(x) = x^3 + 3x^2 - 8$.

$K > 0$ such that for large enough $n$ we have

$$|x_n - L| \leq K|\beta_n|$$

then we say that $(x_n)$ **converges to** $L$ **with order of convergence** $O(\beta_n)$. In this case we write that $x_n = L + O(\beta_n)$.

The expression $O(\beta_n)$ is read "big-oh of $\beta_n$". Almost always, one seeks to show order of convergence $O(\beta_n)$ with

$$\beta_n = \frac{1}{n^p}$$

for some $p > 0$ or that

$$\beta_n = c^n$$

for some $0 < c < 1$.

**Example 2.2.** Consider the sequences $x_n = \frac{n+1}{n^2}$ and $y_n = \frac{n+3}{n^3}$. Show that

$$x_n = O\left(\frac{1}{n}\right) \quad \text{and} \quad y_n = O\left(\frac{1}{n^2}\right)$$

*Solution.* To see this notice we have

$$x_n = \frac{n+1}{n^2} \leq \frac{n+n}{n^2} = 2\frac{1}{n}$$

and

$$y_n = \frac{n+3}{n^3} \leq \frac{n+3n}{n^3} = 4\frac{1}{n^2}$$

□

When applied to the sequence generated by the Bisection method we obtain the following.

**Corollary 2.4**

The sequence $(p_n)$ generated by the Bisection method for $f : [a, b] \to \mathbb{R}$ converges to a zero $p \in (a, b)$ of $f$ with order of convergence $O\left(\frac{1}{2^n}\right)$.

*Proof.* We showed that

$$|p - p_n| < \frac{b-a}{2^n}.$$

□

Although the sequence $(p_n)$ satisfies $|p - p_n| < \frac{b-a}{2^n}$, we may be interested in the behavior of $f(p_n)$. For this we introduce the following definition.

**Definition 2.5**

A function $f : [a, b] \to \mathbb{R}$ is called **Lipschitz** if there exists a constant $K > 0$ such that for every $x, y \in [a, b]$, we have $|f(x) - f(y)| \leq K|x - y|$.

**Proposition 2.6**

If $f : [a, b] \to \mathbb{R}$ is Lipschitz on $[a, b]$ then $f$ is continuous on $[a, b]$.

Suppose that $f : [a, b] \to \mathbb{R}$ is Lipschitz on $[a, b]$ with constant $K > 0$. Suppose that $p \in (a, b)$ is a zero of $f$ and $(p_n)$ is the sequence generated by the Bisection method. Then since $p, p_n \in [a, b]$ we have

$$|f(p) - f(p_n)| \leq K|p - p_n| < K\frac{b-a}{2^n}$$

But $f(p) = 0$ and therefore

$$|f(p_n)| < K\frac{b-a}{2^n}.$$

Hence, $f(p_n) = O\left(\frac{1}{2^n}\right)$. Hence, when $f$ is Lipschitz we are guaranteed that the convergence of $f(p_n)$ is also exponential. The following gives a sufficient condition for the Lipschitz condition.

**Proposition 2.7**

Suppose that $f : [a, b] \to \mathbb{R}$ has a derivative that is bounded on $[a, b]$ by $K > 0$, that is, $|f'(x)| \leq K$ for all $x \in [a, b]$. Then $f$ is Lipschitz on $[a, b]$ with constant $K$.

*Proof.* Let $x, y \in [a, b]$. Then by Taylor's theorem

$$f(x) = f(y) + f'(\xi(x))(x - y)$$

where $\xi(x)$ lies in between $x$ and $y$. Now since $|f'(z)| \leq K$ for all $z \in [a, b]$ it follows that

$$|f(x) - f(y)| = |f'(\xi(x))(x - y)| \leq K|x - y|.$$

Since $x, y$ are arbitrary, this proves that $f$ is Lipschitz. □

The previous proposition can be combined with the following.

**Proposition 2.8: Extreme Value Theorem**

Let $h : [a, b] \to \mathbb{R}$ be a continuous function. Then there exists $x_1, x_2 \in [a, b]$ such that $f(x_1) \leq f(x) \leq f(x_2)$ for all $x \in [a, b]$, and in particular $h$ is bounded on $[a, b]$. In particular, $|f(x)| \leq \max\{|f(x_1)|, |f(x_2)|\}$.

Now, if $f'$ is continuous on $[a, b]$ then by the Extreme Value Theorem $f'$ is bounded on $[a, b]$. Consequently, $f$ is Lipschitz with constant $K = \max_{x \in [a,b]} |f'(x)|$.

**Example 2.3.** Consider $f(x) = x^3 + 3x^2 - 8$ on the interval $[0, 2]$. We have that $f'(x) = 3x^2 + 6x$ and therefore $|f'(x)| \leq 24$ and the maximum of $|f'(x)|$ occurs at $x = 2$. Therefore, for all $x, y \in [0, 2]$ we have $|f(x) - f(y)| \leq K|x - y|$ with $K = 24$. Now, from our previous analysis,

$$|f(p_n)| < K\frac{b - a}{2^n} = 24\frac{1}{2^{n-1}} = 3\frac{1}{2^{n-4}}.$$

In Figure 2.1, we plot the sequence $|f(p_n)|$ and the error estimate $K(b - a)2^{-n}$ for $n = 1, 2, \ldots, 10$. Notice that the estimate is very conservative for $n$ small.

**Example 2.4.** Let $f(x) = \sin(x)$ and let $g(x) = 1 - x^2$ and consider the problem of finding $p$ such that $f(p) = g(p)$, i.e., where the graphs of $f$ and $g$ intersect. We are interested in approximating $p$ with $p^*$ so that $|f(p^*) - g(p^*)| < 10^{-6}$. To this end, let

$$h(x) = f(x) - g(x) = \sin(x) - 1 + x^2.$$

Then $p$ is a zero of $h$ if and only if it is a root of the equation $f(x) = g(x)$. Now $h(0) = -1$ and $h(1) = \sin(1) > 0$. Hence, $h$ has a zero in the interval $[0, 1]$. Now, $h'(x) = \cos(x) + 2x$

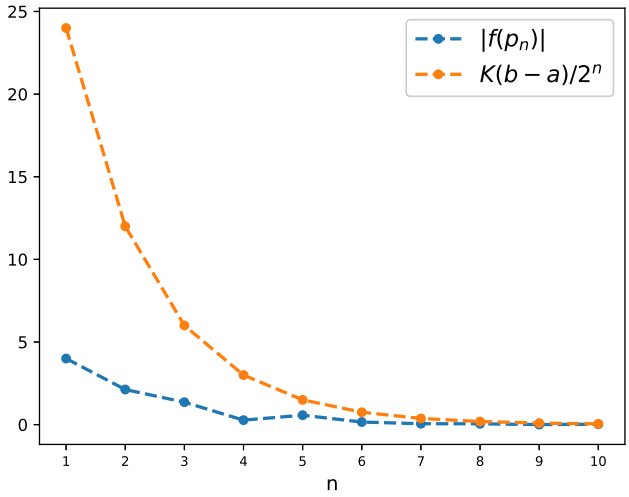



Figure 2.1: Error estimage $K(b-a)2^{-n}$ and $|f(p_n)|$ from Example 2.3

and since $h''(x) = -\sin(x) + 2 > 0$ for all $x$ then $h'(x)$ is an increasing function. Therefore, the maximum value of $h'(x)$ on $[0,1]$ is $h'(1) = \cos(1) + 2 < 3$. Hence, let $K = 3$. Therefore, if $(p_n)$ is the Bisection method sequence applied to $h$ on the interval $[0,1]$ then

$$|h(p_n)| < 3(1-0)\frac{1}{2^n} = \frac{3}{2^n}.$$

Now, $\frac{3}{2^n} < 10^{-6}$ if and only if

$$\frac{\ln(3 \cdot 10^6)}{\ln(2)} < n.$$

Now $\frac{\ln(3\cdot 10^6)}{\ln(2)} \approx 21.52$ and thus if $n \geq 22$ then $|h(p_n)| = |f(p_n) - g(p_n)| < 10^{-6}$. In fact, one computes that $h(p_{20}) = 0.839956 \times 10^{-6}$.

## 2.2 Fixed-Point Iteration

A point $p$ is a **fixed point** of the function $g$ if $g(p) = p$. In this case, it is easy to see that the graph of $g$ intersects the graph of $h(x) = x$ at $p$.

Fixed points and zeros of functions are related in the following way. Suppose that $f(p) = 0$. Then the point $p$ is a fixed point of the function $g(x) = x - f(x)$ since $g(p) = p - f(p) = p$. In fact, for any function $\phi(x)$, if $g(x) = x - \phi(x)f(x)$ then $g(p) = p - \phi(p)f(p) = p$. Conversely, if $g$ has a fixed point at $p$ then $f(x) = x - g(x)$ has a zero at $p$ since $f(p) = p - g(p) = p - p = 0$.

In this section, we will introduce the fixed-point iteration method for finding fixed points of functions. Finding fixed points is an alternative, and sometimes more powerful, method

for finding the zeros of a function. In fact, if $\phi$ is chosen so that $\phi(p) \neq 0$, then $g(x) = x - \phi(x)f(x)$ has a fixed point at $p$ if and only if $p$ is a zero of $f$. It might be easier to find a fixed point of $g$ then to find a zero of $f$ directly, and we will see that understanding the fixed-point problem can be useful in finding zeros of functions. Before giving the Fixed-Point Theorem, we need the following.

**Theorem 2.9: Mean Value Theorem**

Suppose that $f : [a, b] \to \mathbb{R}$ is continuous and $f'(x)$ exists for all $x \in (a, b)$. Then there exists $c \in (a, b)$ such that

$$\frac{f(b) - f(a)}{b - a} = f'(c).$$

**Theorem 2.10: Fixed-Point Theorem**

Let $g : [a, b] \to \mathbb{R}$ be a continuous function.
(a) If $g(x) \in [a, b]$ for all $x \in [a, b]$ then $g$ has a fixed point in $[a, b]$.
(b) In addition, if $g$ is Lipschitz with constant $0 < K < 1$ then $g$ has a unique fixed point in $[a, b]$.

*Proof.* (a) If $g(a) = a$ or $g(b) = b$ we are done, so suppose that $a < g(a)$ and $g(b) < b$. Consider $h(x) = x - g(x)$. Then $h(a) = a - g(a) < 0$ and $h(b) = b - g(b) > 0$. Hence, by the IVT, there exists $p \in (a, b)$ such that $h(p) = 0$, that is $p = g(p)$. Hence $g$ has a fixed point at $p$.

(b) Suppose that $g$ has two distinct fixed points $p$ and $q$ in $[a, b]$. Then,

$$|p - q| = |g(p) - g(q)| \leq K|p - q|$$

and therefore $K \geq 1$. Hence, if $K < 1$ then $g$ has a unique fixed point.

□

**Example 2.5.** Apply Theorem 2.10 to $g(x) = (x^2 - 1)/3$ on the interval $[-1, 1]$ to show that $g$ has a unique fixed point in $[-1, 1]$.

*Solution.* We need to show that $g([-1, 1]) \subset [-1, 1]$. Since $g$ is continuous, it's maximum and minimum values are either at the end-points of $[-1, 1]$ or where $g'(x) = 0$. Now, $g'(x) = \frac{2}{3}x$ and thus $g'(x) = 0$ at $x = 0$. Now $g(-1) = 0$, $g(1) = 0$, and $g(0) = -1/3$. Hence, $-\frac{1}{3} \leq g(x) \leq 0$ for all $x \in [-1, 1]$. Hence, $g$ has a fixed point in $[-1, 1]$. Now for $x \in [-1, 1]$ we have that $|g'(x)| = \frac{2}{3}|x| \leq \frac{2}{3} < 1$. Hence, in fact $g$ has a unique fixed point on $[-1, 1]$. In fact, the fixed point $p$ satisfies $p = \frac{p^2-1}{3}$ or $p^2 - 3p - 1 = 0$. Thus $p = \frac{1}{2}(3 - \sqrt{13}) \in [-1, 1]$.

Now, on the interval $[3,4]$, $g$ also has a fixed point, namely $p = \frac{1}{2}(3+\sqrt{13})$, however we cannot use Theorem 2.10 to conclude the existence nor uniqueness of $p$ in $[3,4]$ since $g(4) = 5$ and $g'(4) = \frac{8}{3} > 1$. This shows that the conditions in Theorem 2.10 are not necessary for existence and uniqueness of fixed points. □

We now describe how **iteration** can be used to determine a fixed point of a function $g(x)$. Before we can do that we need the following theorem.

**Theorem 2.11**

Let $f : [a,b] \to \mathbb{R}$ be a continuous function and suppose that $(p_n)$ is a sequence contained entirely in $[a,b]$ and converging to $p \in [a,b]$. Then

$$\lim_{n\to\infty} f(p_n) = f(p) = f\left(\lim_{n\to\infty} p_n\right).$$

Let $p_0$ be an initial condition and define iteratively $p_n = g(p_{n-1})$ for $n \geq 1$. This generates a sequence $(p_n)$. *If* $(p_n)$ converges, say to $p = \lim_{n\to\infty} p_n$ then since $g$ is continuous we have

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

Hence, $p$ is a fixed point of $g$. This technique of finding fixed points is called **fixed-point iteration**. Below we write pseudocode to implement fixed point iteration.

**Algorithm 2.2** Fixed-Point Iteration

INPUT: $p_0, \varepsilon, N_{\max}$

OUTPUT: Approximate fixed-point $p$ of $g$ or message of failure

1: **set** $p_{\text{old}} = p_0$
2: **for** $n = 1, 2, \ldots, N_{\max}$ **do**
3: **set** $p_{\text{new}} = g(p_{\text{old}})$
4: **if** $|p_{\text{new}} - p_{\text{old}}| < \varepsilon$ **do**
5: **return** $p_{\text{new}}$
**break**
6: **else**
7: **set** $p_{\text{old}} = p_{\text{new}}$
8: **print**("Maximum number of iterations reached.")

Note that the tolerance condition in STEP 4. of Algorithm 2.2 is equivalent to $|g(p_{\text{old}}) - p_{\text{old}}| < \varepsilon$, i.e., it checks whether $p_{\text{old}}$ is a fixed point of $g$ with error no more than $\varepsilon$. Also note that we return $p_{\text{new}}$ since if $|p_{\text{new}} - p_{\text{old}}| < \varepsilon$ then

$$|g(p_{\text{new}}) - p_{\text{new}}| = |g(p_{\text{new}}) - g(p_{\text{old}})| \leq K|p_{\text{new}} - p_{\text{old}}| < K\varepsilon$$

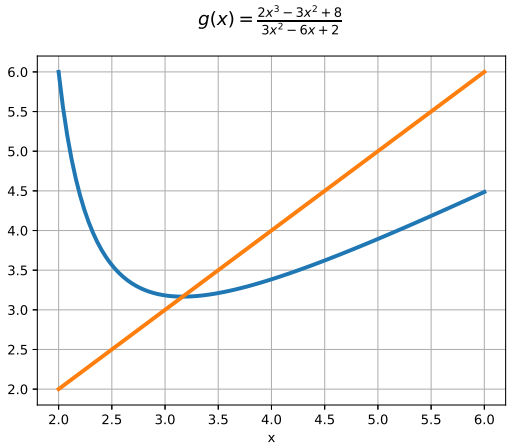



Figure 2.2: Example 2.6

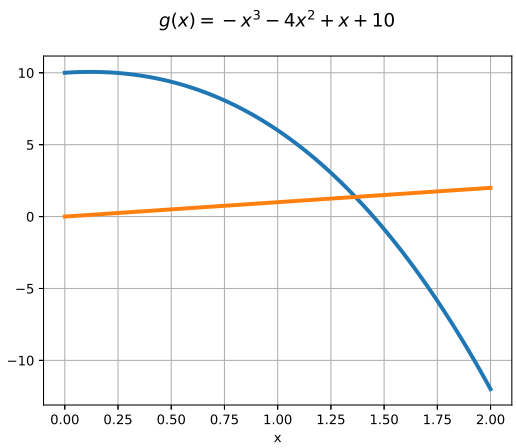



Figure 2.3: Example 2.7

and since $K < 1$ then $|g(p_{\text{new}}) - p_{\text{new}}| < \varepsilon$.

**Example 2.6.** Consider $g(x) = \frac{2x^3-3x^2+8}{3x^2-6x+2}$ on the interval $[2, 6]$. The graph of $g$ and $h(x) = x$ is shown in Figure 2.6. We see that $g$ has a unique fixed point in $[2, 6]$. Suppose we set $p_0 = 5$ and compute $(p_n)$ via fixed-point iteration. We compute that

$$\begin{aligned} p_1 &= 3.893617021276595 \\ p_2 &= 3.340747402373615 \\ p_3 &= 3.179810015624071 \\ p_4 &= 3.166402441514069 \\ p_5 &= 3.166312751395192 \\ p_6 &= 3.166312747397789 \\ p_7 &= 3.166312747397789 \\ p_8 &= 3.166312747397789 \end{aligned}$$

If we run the fixed-point iteration algorithm we obtain that for $N = 6$ we have $|g(p_6)-p_6| < \varepsilon$, or $|p_7 - p_6| < \varepsilon$, for $\varepsilon = 10^{-10}$. Hence, fixed-point iteration seems to generate a sequence $(p_n)$ converging to a fixed point of $g$.

**Example 2.7.** Consider $g(x) = -x^3 - 4x^2 + x + 10$ on the interval $[0, 2]$. In Figure 2.7 we show the graph of $g$ and $h(x) = x$, and observe that $g$ has a unique fixed point in $[0, 2]$. Let

$p_0 = 1.5$. We compute using $p_n = g(p_{n-1})$ that

$$\begin{aligned}
p_1 &= -0.875000000000000 \\
p_2 &= 6.732421875000000 \\
p_3 &= -4.697200120016932 \times 10^2 \\
p_4 &= 1.027545551873851 \times 10^8 \\
p_5 &= -1.084933870531746 \times 10^{24} \\
p_6 &= 1.277055591444378 \times 10^{27} \\
p_7 &= -2.082712908581027 \times 10^{216} \\
p_8 &= \text{NaN}
\end{aligned}$$

Here NaN stands for Not-a-Number and arises in computer computations as a result of mathematically undefined operations like $\frac{1}{0}$ or $\infty - \infty$. In this case, $-p(7)^3 \approx 8 \times 10^{648}$ and $4p(7)^2 \approx 1.6 \times 10^{432}$. On a computer using 64 bits to represent numbers, the largest number that can be stored is approximately $1 \times 10^{308}$. Any computation resulting in a number larger that this number is set to $\infty = \text{Inf}$ and this is an instance of **overflow**. Hence, the computation $-p(7)^3 - 4p(7)^2$ yields $\infty - \infty$, which is undefined. In any case, based on these computations, it seems as though fixed-point iteration is unsuccessful in generating a sequence $(p_n)$ converging to the fixed point of $g$.

It is important then to establish sufficient conditions when fixed-point iteration yields a sequence converging to a fixed-point.

**Theorem 2.12: Fixed-Point Iteration**

Suppose that $g : [a, b] \to \mathbb{R}$ satisfies the properties in Theorem 2.10, that is, $g([a, b]) \subset [a, b]$ and $g$ is Lipschitz with constant $0 < K < 1$. Then for any $p_0 \in [a, b]$, the sequence

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

converges to the unique fixed point of $g$ in $[a, b]$.

*Proof.* By Theorem 2.10, $g$ has a unique fixed point $p \in [a, b]$. Now, since $g([a, b]) \subset [a, b]$ it follows that $p_n$ is well-defined and moreover $p_n \in [a, b]$. Now,

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

By induction,

$$|p_n - p| \leq K^n|p_0 - p|.$$

Now since $0 < K < 1$,then $\lim_{n\to\infty} K^n|p_0 - p| = 0$ and therefore $(p_n)$ converges to $p$. □

**Corollary 2.13**

Suppose that $g : [a, b] \to \mathbb{R}$ satisfies the properties in Theorem 2.10. Then

$$|p_n - p| \leq K^n \max\{p_0 - a, b - p_0\}$$

and

$$|p_n - p| \leq \frac{K^n}{1 - K}|p_1 - p_0|, \quad n \geq 1.$$

*Proof.* For the first claim, $|p - p_0| \leq \max\{p_0 - a, b - p_0\}$ since $p \in [a, b]$. The second claim is left as an exercise. □

Hence, Theorem 2.12 determines when fixed point iteration yields a sequence converging to a fixed point. However, the conditions are not necessary. In the case that $g(x) = \frac{2x^3-3x^2+8}{3x^2-6x+2}$ on the interval $[2, 6]$, we do have that $g([2, 6]) \subset [2, 6]$ but $g'(x)$ has range $[-12, 0.61]$.

**Example 2.8.** Consider the function $g(x) = \sqrt{\frac{10}{4+x}}$ on the interval $[1, 2]$. It is clear that $g$ is decreasing on $[1, 2]$. Now $1 < g(1) = \sqrt{2} < 2$ and $1 < g(2) = \sqrt{10/6} < 2$. Hence, $g([1, 2]) \subset [1, 2]$. Now, for all $x \in [1, 2]$, we have

$$\begin{aligned} |g'(x)| &= \left| \frac{-5}{\sqrt{10}(4 + x)^(3/2)} \right| \\ &\leq \frac{5}{\sqrt{10}(5)^(3/2)} \\ &= \frac{1}{\sqrt{10}\sqrt{5}} \\ &< 1. \end{aligned}$$

In fact, $|g'(x)| < 0.15$. Hence, according to Theorem 2.12, for any $p_0 \in [1, 2]$, the sequence $p_n = g(p_{n-1})$ converges to the fixed point of $g$. For instance, with $p_0 = 1$ one finds that $|g(p_{11}) - p_{11}| < 10^{-10}$ and $p_{11} = 1.365230013468209$ is thus a good approximation to $p$.

## 2.3 Newton's Method

Newton's method is a powerful root-finding method implemented as a fixed-point iteration. Suppose that $f''(x)$ is continuous on $[a, b]$ and we are interested in finding a zero $p \in [a, b]$ of $f$. Suppose that $p_0 \in [a, b]$ is close to $p$, so that in particular, $|p - p_0|$ is small, and $f'(p_0) \neq 0$. From Taylor's theorem,

$$f(p) = f(p_0) + f'(p_0)(p - p_0) + \frac{f''(\xi(p))}{2}(p - p_0)^2$$

where $\xi(p)$ is in between $p$ and $p_0$. Now since $f(p) = 0$, we have

$$0 = f(p_0) + f'(p_0)(p - p_0) + \frac{f''(\xi(p))}{2}(p - p_0)^2.$$

Since $|p - p_0|$ is small, the term $(p - p_0)^2$ is small and we have

$$0 \approx f(p_0) + f'(p_0)(p - p_0)$$

or

$$p \approx p_0 - \frac{f(p_0)}{f'(p_0)}.$$

Hence, the number $p_1 = p_0 - \frac{f(p_0)}{f'(p_0)}$ is an approximation to $p$ provided $\frac{f''(\xi(p))}{2}(p - p_0)^2$ is small. We can repeat this process with $p_1$ to obtain that

$$p \approx p_1 - \frac{f(p_1)}{f'(p_1)}$$

and thus $p_2 = p_1 - \frac{f(p_1)}{f'(p_1)}$ is an approximation to $p$. We can repeat this iterative process and obtain a sequence $(p_n)$ defined by

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

which is well-defined provided that $f'(p_n) \neq 0$ for all $n$. Below is the algorithm for Newton's method.

**Algorithm 2.3** Newton's Method

INPUT: $p_0, \varepsilon, N_{\max}$

OUTPUT: Approximate zero $p$ of $f$ or message of failure

1: **set** $p_{\text{old}} = p_0$
2: **for** $n = 1, 2, \ldots, N_{\max}$ **do**
3: **set** $p_{\text{new}} = p_{\text{old}} - f(p_{\text{old}})/f'(p_{\text{old}})$
4: **if** $|p_{\text{new}} - p_{\text{old}}| < \varepsilon$ **do**
5: **return** $p_{\text{new}}$
**break**
6: **else**
7: **set** $p_{\text{old}} = p_{\text{new}}$
8: **print**("Maximum number of iterations reached.")

Notice that Newton's method is a fixed-point iteration with

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

Clearly, Newton's method cannot proceed if $f'(p_n) = 0$ for some $n$.

| $n$ | Fixed-Point | $\lvert f(p_n)\rvert$ | Newton | $\lvert f(p_n)\rvert$ | Bisection | $\lvert f(p_n)\rvert$ |
|---|---|---|---|---|---|---|
| 0 | 0.785398 | 7.829138e-02 | 0.785398 | 7.829138e-02 | 0.785398 | 7.829138e-02 |
| 1 | 0.707107 | 5.313782e-02 | 0.739536 | 7.548747e-04 | 0.392699 | 5.311805e-01 |
| 2 | 0.760245 | 3.557712e-02 | 0.739085 | 7.512987e-08 | 0.589049 | 2.424210e-01 |
| 3 | 0.724667 | 2.405240e-02 | 0.739085 | 6.661338e-16 | 0.687223 | 8.578706e-02 |
| 4 | 0.748720 | 1.615904e-02 | 0.739085 | 1.110223e-16 | 0.736311 | 4.640347e-03 |
| 5 | 0.732561 | 1.090337e-02 | 0.739085 | 0.000000e+00 | 0.760854 | 3.660739e-02 |

Table 2.2: Comparison of the bisection method, direct fixed-point iteration, and Newton's method to find the zero of $f(x) = x - \cos(x)$. Newton's method displays very rapid convergence.

**Example 2.9.** Consider the problem of finding a zero of $f(x) = x - \cos(x)$ on the interval $[0, \pi/2]$. A zero of $f$ is a fixed point of $g_1(x) = \cos(x)$. Hence, we can generate a fixed-point iteration sequence with $g_1(x)$. Alternatively, we can generate a fixed-point iteration sequence using Newton's method with

$$g_2(x) = x - \frac{f(x)}{f'(x)} = x - \frac{x - \cos(x)}{1 + \sin(x)}.$$

For the initial condition $p_0 = \frac{\pi}{4}$, we include the results in Table 2.2 for both cases and also using the bisection method. The numerical results indicate that Newton's method has very fast convergence.

The previous example shows that Newton's method can converge very quickly and we will see why this is the case in the next section. For now, the next theorem gives sufficient conditions for when Newton's method will converge. The main point of the theorem is that if the initial approximation $p_0$ is close to $p$ then convergence is guaranteed.

**Theorem 2.14**

Suppose that $f$ is such that $f''$ is continuous on $[a, b]$ and $p \in (a, b)$ is such that $f(p) = 0$ and $f'(p) \neq 0$. Then there exists $\delta > 0$ such that Newton's method generates a sequence $(p_n)$ that converges to $p$ for all initial conditions $p_0 \in [p - \delta, p + \delta]$.

*Proof.* The approach is to apply the fixed-point iteration theorem (Theorem 2.12) to the function $g(x) = x - \frac{f(x)}{f'(x)}$. Choose any $0 < K < 1$. Since $f'(x)$ is continuous and $f'(p) \neq 0$, there exists $\delta_1 > 0$ such that $g'(x)$ is well-defined on $[\delta_1 - p, p + \delta_1]$. Now,

$$g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}$$

and since $f^{(2)}$ is continuous, it follows that $g'(x)$ is continuous on $[\delta_1 - p, p + \delta_1]$. Since $g'(p) = 0$ there exists $\delta < \delta_1$ such that $|g'(x)| \leq K$ for all $x \in I = [p - \delta, \delta + p]$. This proves that $g$ is Lipschitz on the interval $I$ with constant $K < 1$. It remains to show that $g(I) \subset I$. For any $x \in I$, we have

$$|g(x) - p| = |g(x) - g(p)| \leq K|x - p| \leq |x - p| \leq \delta$$

Hence $g(x) \in I$. Theorem 2.12 applied to $g$ completes the proof. □

The above theorem/proof shows that under reasonable assumptions on $f$, if $p_0$ is close to $p$ then Newton's method converges to a zero of $f$. Moreover, the bound $|g'(x)| \leq K$ controls the convergence of Newton's method. In particular, if $\frac{f''(x)}{[f'(x)]^2}$ is not too large then since $f(p) = 0$, we should see convergence of order $p_n = p + O(K^n)$.

## 2.4 Error Analysis of General Iterative Methods

In this section, we introduce a way to measure the speed of convergence of a sequence and use this measure to describe the convergence of fixed-point iteration methods. Recall that a sequence $(x_n)$ converges to $L$ with order of convergence $O(\beta_n)$ if

$$|x_n - L| \leq K|\beta_n|$$

where $K > 0$ and $(\beta_n) \to 0$. This definition of order of convergence is of a comparison type because it says that $(x_n)$ converges to $L$ at least as fast as $(\beta_n)$ converges to 0. The following notion of order of convergence, however, is meant to describe the speed in which a sequence converges relative to how subsequent elements of the sequence are approaching the limit.

**Definition 2.15**

Suppose that $(p_n)$ converges to $p$. If there are positive numbers $\alpha$ and $K$ such that

$$|p_{n+1} - p| \leq K|p_n - p|^\alpha \tag{2.3}$$

then we say that $(p_n)$ **converges to $p$ of order $\alpha$.** If $\alpha = 1$ then the convergence is said to be at least **linear** and if $\alpha = 2$ then the convergence is said to be at least **quadratic**.

Note that (2.3) is equivalent to

$$\frac{|p_{n+1} - p|}{|p_n - p|^\alpha} \leq K$$

which is sometimes more useful to work with.

**Example 2.10.** Analyze the order of convergence of the following sequences.

(a) Suppose that $k$ is a positive integer and consider the sequence $p_n = \frac{1}{n^k}$, which converges to $p = 0$. Then

$$p_{n+1} = \frac{1}{(n+1)^k} = \frac{1}{n^k + \cdots + 1} \leq \frac{1}{n^k} = p_n.$$

Hence, $p_{n+1} \leq p_n$ and therefore $(p_n)$ converges to $p = 0$ at least linearly. Let $\alpha$ be a positive number. Then

$$\frac{p_{n+1}}{p_n^\alpha} = \frac{\frac{1}{(n+1)^k}}{n^{\alpha k}} = \frac{n^{\alpha k}}{(n+1)^k}.$$

If $\alpha > 1$ then the ratio $\frac{p_{n+1}}{p_n}$ is unbounded as $n \to \infty$. Therefore, it does not hold that

$$\frac{p_{n+1}}{p_n^\alpha} \leq K$$

for some $K > 0$.

(b) Consider the sequence $p_n = C^n$ where $0 < C < 1$, which converges to $p = 0$. Then

$$p_{n+1} = C^{n+1} = Cp_n.$$

Hence, $(p_n)$ converges to $p = 0$ linearly.

(c) Let $0 < C < 1$ and consider the sequence $q_n = C^{2^n}$, which converges to $p = 0$. Then

$$q_{n+1} = C^{2^{n+1}} = C^{2^n \cdot 2} = (C^{2^n})^2 = q_n^2.$$

Thus, $(q_n)$ converges to $p = 0$ quadratically.

As shown in the previous example, the sequence $p_n = C^n$ converges to $p = 0$ linearly and the sequence $q_n = C^{2^n}$ converges quadratically. In Figure 2.4 and Table 2.3, we compare the convergence speed of these two sequences. Notice that for the quadratically convergent sequence, the value of $q_{11}$ is smaller than the smallest number that can be stored on a computer using 64 bits to represent numbers (approximately $1 \times 10^{-308}$).

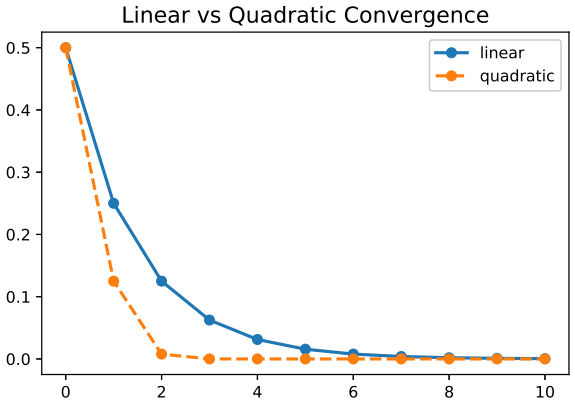



Figure 2.4: $p_n = C^n$ (linear) and $q_n = C^{2^n}$ (quadratic) for $C = 0.5$.

| $n$ | linear $p_n = C^n$ | quadratic $q_n = C^{2^n}$ |
|---|---|---|
| 1 | 5.0000e-01 | 5.0000e-01 |
| 2 | 2.5000e-01 | 1.2500e-01 |
| 3 | 1.2500e-01 | 7.8125e-03 |
| 4 | 6.2500e-02 | 3.0518e-05 |
| 5 | 3.1250e-02 | 4.6566e-10 |
| 6 | 1.5625e-02 | 1.0842e-19 |
| 7 | 7.8125e-03 | 5.8775e-39 |
| 8 | 3.9062e-03 | 1.7272e-77 |
| 9 | 1.9531e-03 | 1.4917e-154 |
| 10 | 9.7656e-04 | 1.1125e-308 |
| 11 | 4.8828e-04 | 0.0000e+00 |

Table 2.3: Numerical values of $p_n$ and $q_n$.

We now analyze the expected convergence speed of a sequence generated by the fixed-point iteration method.

**Theorem 2.16**

Suppose that $g : [a, b] \to \mathbb{R}$ has a continuous derivative $g'$ on $[a, b]$ and $g(x) \in [a, b]$ for all $x \in [a, b]$. Assume further that there exists a constant $0 < K < 1$ such that $|g'(x)| \leq K$ for all $x \in [a, b]$. Let $p \in [a, b]$ denote the unique fixed point of $g$. If $g'(p) \neq 0$ then for any $p_0 \in [a, b]$ (not equal to $p$) the sequence $(p_n)$ generated by fixed-point iteration using $g$ converges only linearly to $p$.

*Proof.* Since $p_n, p \in [a, b]$, by the Mean Value theorem we have

$$\frac{g(p_n) - g(p)}{p_n - p} = g'(\xi_n)$$

where $\xi_n$ is in between $p_n$ and $p$. Therefore,

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

and thus $(p_n)$ converges to $p$ at least linearly. Now suppose that $\alpha > 1$ and consider the ratio

$$\frac{|p_{n+1} - p|}{|p_n - p|^\alpha}.$$

We have that

$$\frac{|p_{n+1}-p|}{|p_n-p|^\alpha} = \frac{|g'(\xi_n)(p_n-p)|}{|p_n-p|^\alpha} = \frac{|g'(\xi_n)||p_n-p|}{|p_n-p|^\alpha} = \frac{|g'(\xi_n)|}{|p_n-p|^{\alpha-1}}.$$

Since $\xi_n$ is in between $p_n$ and $p$, and $(p_n)$ converges to $p$, then by the Squeeze theorem $(\xi_n)$ also converges to $p$. Thus, since $g'$ is continuous we have $\lim_{n\to\infty} g'(\xi_n) = g'(p) \neq 0$. Therefore, since $\lim_{n\to\infty} |p_n-p|^{\alpha-1} = 0$, we have

$$\lim_{n\to\infty} \frac{|p_{n+1}-p|}{|p_n-p|^\alpha} = \lim_{n\to\infty} \frac{|g'(p_n)|}{|p_n-p|^{\alpha-1}} = \infty$$

and thus $(p_n)$ converges only linearly. □

The previous theorem suggests that in order for the fixed-point iteration sequence $(p_n)$ to converge quadratically we need $g'(p) = 0$. The following theorem provides further sufficient conditions for quadratic convergence.

**Theorem 2.17: Quadratic Convergence**

Suppose that $g(p) = p$, that $g'(p) = 0$, and $g^{(2)}$ is continuous on a closed interval $I$ containing $p$ in its interior. Then there exists a $\delta > 0$ such that for any $p_0 \in [p-\delta, p+\delta]$, the sequence $(p_n)$ obtained by fixed-point iteration with $g$ converges at least quadratically to $p$.

*Proof.* Let $0 < K < 1$. Then as in the proof of Theorem 2.14, there exists a $\delta > 0$ such that $|g'(x)| \leq K$ and $g(x) \in [p-\delta, p+\delta]$ for all $x \in [p-\delta, p+\delta]$. Moreover, for any $p_0 \in [p-\delta, p+\delta]$ the sequence $(p_n)$ converges to $p$. By making $\delta$ smaller, we can assume that $[p-\delta, p+\delta] \subset I$ and thus $g^{(2)}$ is continuous on $[p-\delta, p+\delta]$. For notational simplicity, let $[a, b]$ denote the interval $[p-\delta, p+\delta]$. By Taylor's theorem, we have

$$g(x) = g(p) + g'(p)(x-p) + \frac{g^{(2)}(\xi)}{2}(x-p)^2$$

for all $x \in [a, b]$, where $\xi$ is in between $x$ and $p$. Therefore,

$$p_{n+1} = g(p_n) = p + \frac{g^{(2)}(\xi_n)}{2}(p_n-p)^2.$$

Since $g^{(2)}$ is continuous on $[a, b]$, there exists $M > 0$ such that $|g^{(2)}(x)| \leq M$ for all $x \in [a, b]$. Therefore

$$|p_{n+1}-p| = \frac{1}{2}|g^{(2)}(\xi_n)(p_n-p)^2| \leq \frac{M}{2}|p_n-p|^2.$$

This proves that $(p_n)$ converges at least quadratically to $p$. □

We can apply the previous theorem to obtain conditions for when Newton's method converges at least quadratically.

**Corollary 2.18**

Suppose that $f$ is such that $f^{(3)}$ is continuous on $[a, b]$ and $p \in (a, b)$ is such that $f(p) = 0$ and $f'(p) \neq 0$. Then there exists $\delta > 0$ such that Newton's method applied to $f$ generates a sequence $(p_n)$ that converges to $p$ at least quadratically for all initial conditions $p_0 \in [p - \delta, p + \delta]$.

*Proof.* Recall that in Newton's method, fixed point iteration is performed with

$$g(x) = x - \frac{f(x)}{f'(x)}$$

and therefore $g(p) = p$. A direct calculation yields that

$$g'(x) = \frac{f(x)f''(x)}{[f'(x)]^2}.$$

Hence, $g'(p) = 0$. Moreover, since $f^{(3)}$ is continuous then $g''$ is continuous. Theorem 2.17 is therefore applicable to $g$ and the result follows. □

# 3

# Interpolation

## 3.1 Lagrange Polynomials

The problem of interpolation is the following. Given data points $(x_0, y_0)$, $(x_1, y_1), \ldots, (x_n, y_n)$ find a function $P : [a, b] \to \mathbb{R}$ such that $y_j = P(x_j)$ for all $j = 0, 1, \ldots, n$. We say that $P$ **interpolates** the data points $(x_0, y_0), \ldots, (x_n, y_n)$. The $y$-values of the data points may represent the values of an unknown and sought out function $f : [a, b] \to \mathbb{R}$. The interpolating function $P$ could then be used to estimate $f$ at all points $x \in [a, b]$. The $x$-values $x_0, x_1, \ldots, x_n$ are called the **nodes**.

The most straightforward interpolating function $P(x)$ that is continuous is a piecewise linear interpolation. Assuming the nodes are ordered $x_0 < x_1 < \cdots < x_n$, for each subinterval $[x_k, x_{k+1}]$, one uses the line from $(x_k, y_k)$ to $(x_{k+1}, y_{k+1})$ to interpolate. This results in a continuous interpolating function $P(x)$ that is differentiable at all points in $[a, b]$ except (possibly) at the nodes $x_0, x_1, \ldots, x_n$. For instance, the function $f : [-2, 4] \to \mathbb{R}$ defined by

$$f(x) = 3\cos(2x) - \sin(0.5x) + 3\sin(3.3x) + 0.5\sin(10x)$$

with data points $x_0 = -2, x_1 = -1.5, x_2 = -1, \ldots, x_{11} = 3.5, x_{12} = 4$ and a linear interpolation $P(x)$ is shown in Figure 3.1.

Usually, interpolation is done when the function $f$ is unknown, but sometimes interpolation is done even when $f$ is known explicitly. For instance, if $f$ is too costly to evaluate and $f$ will be evaluated repeatedly at many points, we may evaluate $f$ at a few points $x_0, x_1, \ldots, x_n$ and then compute a simple interpolating function $P$ that can be used to approximate $f(x)$ at other points $x \in [a, b]$. A particularly simple and useful class of functions, both computationally and to analyze, are the polynomials:

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

Polynomials are relatively easy to evaluate, differentiate, integrate, etc., and thus are good

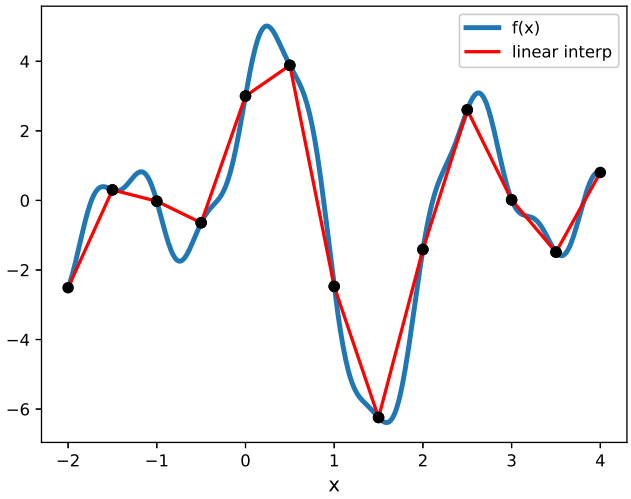



Figure 3.1: Interpolation by connecting data points with lines

candidates to use for interpolation. Moreover, the following theorem shows that polynomials can approximate continuous functions to any desired degree of accuracy.

**Theorem 3.1: Weierstrass Approximation Theorem**

Let $f : [a, b] \to \mathbb{R}$ be a continuous function. For each $\varepsilon > 0$ there exists a polynomial $P(x)$ such that $|f(x) - P(x)| < \varepsilon$ for all $x \in [a, b]$.

In Figure 3.2, we illustrate the Weierstrass Approximation Theorem. The black curve is a continuous function $f(x)$, the top and bottom red curves are $f(x) + \varepsilon$ and $f(x) - \varepsilon$, respectively, and the green curve is the graph of a function $P(x)$ such that $|f(x) - P(x)| < \varepsilon$ for all $x \in [0, 3]$.

We have already encountered the use of polynomials to approximate functions, namely the Taylor polynomials:

$$P_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(c)}{n!}(x - c)^k.$$

One of the potential problems with using Taylor polynomials to approximate $f$ over an **entire** interval $[a, b]$ is that, in general, the approximation $P_n(x)$ is accurate only for points $x$ nearby the base point $c$. Although it is true that for some functions on an interval $[a, b]$, with $c \in (a, b)$, there exists $n \in \mathbb{N}$ such that $|f(x) - P_n(x)| < \epsilon$ for all $x \in [a, b]$, this is not always the case, even for infinitely differentiable functions. For example, consider the function $f(x) = \frac{1}{x^2}$ on the interval $[\frac{1}{2}, 3]$. One can compute that for $c = 1$ we have

$$P_n(x) = \sum_{k=0}^{n}(-1)^k(k+1)(x-1)^k = 1-2(x-1)+3(x-1)^2-4(x-1)^3+\cdots+(-1)^n(n+1)(x-1)^n.$$

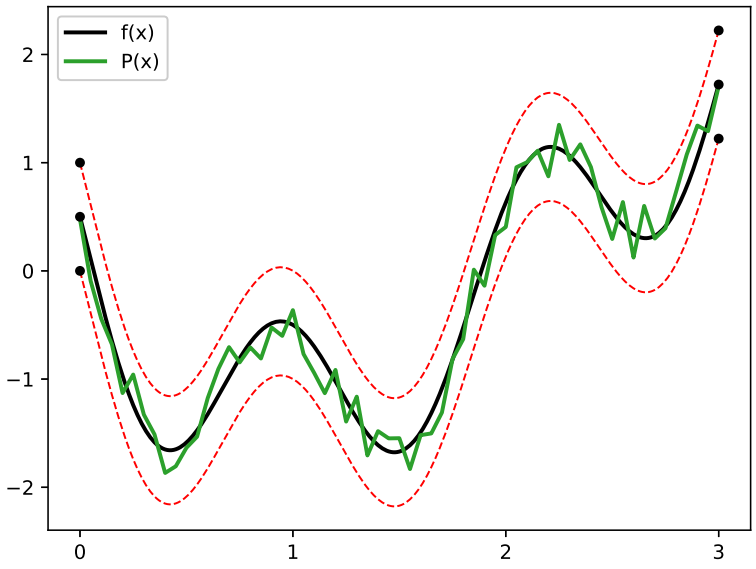



Figure 3.2: $P(x)$ approximating $f(x)$ within $\varepsilon$ at every $x \in [0, 3]$

At $x = 2$ one finds that

$$P_1(2) = 1,\ P_2(2) = -1,\ P_3(2) = 2,\ P_4(2) = -2,\ P_5(2) = 3,\ P_6(2) = -3, \ldots$$

Thus, as $n \to \infty$, $P_n(2)$ becomes an increasingly worse approximation to $f(2) = \frac{1}{4}$. Although Taylor polynomials can suffer from this phenomenon, they are still useful in deriving error estimates and developing numerical algorithms.

We now construct a class of polynomials that can be used to interpolate a set of data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$. The first question we need to answer is whether for a given set of data points there exists a polynomial that interpolates that data.

**Theorem 3.2: Unique Interpolating Polynomial**

Suppose that $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ is a given set of data points with distinct nodes. Then there exists a unique polynomial $P(x)$ of order at most $n$ such that $P(x_k) = y_k$ for $k = 0, 1, \ldots, n$.

*Proof.* We will show how to explicitly construct such a polynomial

$$P(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0.$$

The condition that $P$ interpolates that data results in the following $(n+1)$ equations

$$\begin{aligned}
y_0 &= c_n x_0^n + c_{n-1} x_0^{n-1} + \cdots + c_1 x_0 + c_0 \\
y_1 &= c_n x_1^n + c_{n-1} x_1^{n-1} + \cdots + c_1 x_1 + c_0 \\
y_2 &= c_n x_2^n + c_{n-1} x_2^{n-1} + \cdots + c_1 x_2 + c_0 \\
\vdots &= \qquad\qquad\qquad \vdots \\
y_n &= c_n x_n^n + c_{n-1} x_n^{n-1} + \cdots + c_1 x_n + c_0
\end{aligned}$$

These equations are equivalent to the linear system

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix}}_{c} = \underbrace{\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}}_{y}$$

The matrix $A$ is known as a **Vandermonde matrix**. The linear system $Ac = y$ is uniquely solvable for $c$ if and only if $A$ is an invertible matrix. Invertibility of $A$ is equivalent to $\det(A) \neq 0$. Using induction, one can show that

$$\det(A) = \prod_{1 \leq i < j \leq n} (x_i - x_j).$$

Since all nodes are distinct we have $\det(A) \neq 0$ and thus $A$ is indeed invertible. The unique coefficients are $c = A^{-1}y$. □

Although the previous theorem gives an explicit method to compute the interpolating polynomial $P$ of a given data set, there is a more interesting way to construct $P$. Assume that the nodes $x_0, x_1, \ldots, x_n$ are all distinct. For each $k \in \{0, 1, \ldots, n\}$ we define the $n$th order polynomial $L_k(x)$ by

$$L_k(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0)(x_k - x_1) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)},$$

or more succinctly,

$$L_k(x) = \prod_{i \neq k} \frac{(x - x_i)}{(x_k - x_i)}.$$

It is not too hard to see that $L_k(x_i) = 0$ for all $i \neq k$ and $L_k(x_k) = 1$. The polynomial $L_k(x)$ is called the $k$th **Lagrange polynomial** at the nodes $x_0, x_1, \ldots, x_n$. We then define

$$P(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x) = \sum_{k=0}^{n} y_k L_k(x).$$

Notice that $P(x_k) = y_k$ for all $k = 0, 1, \ldots, n$ and thus $P$ is indeed the unique interpolating polynomial for the given data. We have the following.

**Theorem 3.3**

Consider the set of data points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$, with distinct nodes $x_0$, $x_1$, …, $x_n$. Then the polynomial

$$P(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x) \tag{3.1}$$

is the unique polynomial of degree at most $n$ such that $y_k = P(x_k)$.

The decomposition of the form (3.1) of the unique interpolation polynomial for a given set of data points is said to be in **Lagrange form**.

**Example 3.1.** Consider the data

| $x$ | $y$ |
|---|---|
| 1.0 | 1.0 |
| 1.75 | 16/49 |
| 2.25 | 16/81 |
| 3.0 | 1/9 |

(a) Find the 3rd order Lagrange polynomial $P$ interpolating the data.
(b) If the given data if obtained by sampling a function $f$ at the nodes, estimate $f(2)$.

*Solution.* (a) The nodes are $x_0 = 1.0$, $x_1 = 1.75$, $x_2 = 2.25$, and $x_3 = 3.0$. Compute:

$$L_0(x) = \frac{(x-1.75)(x-2.25)(x-3)}{(1-1.75)(1-2.25)(1-3)} = -\frac{8}{15}(x-1.75)(x-2.25)(x-3)$$

$$L_1(x) = \frac{(x-1)(x-2.25)(x-3)}{(1.75-1)(1.75-2.25)(1.75-3)} = \frac{32}{15}(x-1)(x-2.25)(x-3)$$

$$L_2(x) = \frac{(x-1)(x-1.75)(x-3)}{(2.25-1)(2.25-1.75)(2.25-3.0)} = -\frac{32}{15}(x-1)(x-1.75)(x-3)$$

$$L_3(x) = \frac{(x-1)(x-1.75)(x-2.25)}{(3.0-1)(3.0-1.75)(3.0-2.25)} = \frac{8}{15}(x-1)(x-1.75)(x-2.25)$$

Therefore,

$$\begin{aligned} P(x) &= y_0L_0(x) + y_1L_1(x) + y_2L_2(x) + y_3L_3(x) \\ &= 1.0L_0(x) + \frac{16}{49}L_1(x) + \frac{16}{81}L_2(x) + \frac{1}{9}L_3(x) \end{aligned}$$

and after expanding and simplifying we obtain

$$P(x) = -\frac{2368}{11907}x^3 + \frac{17936}{11907}x^2 - \frac{46252}{11907}x + \frac{14197}{3969}$$

(b) We find that $P(2.0) = 0.24246$. The given data was actually obtained by evaluating $f(x) = \frac{1}{x^2}$ at the nodes. Here, $f(2) = 0.25$ and so $P(2.0)$ is a good estimate. The graph of $P$ and $f$ are shown in Figure 3.3.

□

**Example 3.2.** Consider again the function

$$f(x) = 3\cos(2x) - \sin(0.5x) + 3\sin(3.3x) + 0.5\sin(10x)$$

on the interval $[-2, 4]$. We construct the 6th, the 12th, and 14th degree Lagrange polynomials. For each case, we divide the interval $[-2, 4]$ into $n = 6$, $n = 12$, and $n = 14$ equal parts yielding the nodes $-2 = x_0 < x_1 < \cdots < x_n = 4$ that are equally spaced at a distance $h = \frac{4-(-2)}{n} = \frac{6}{n}$. In Figure 3.4, we plot $f$ and the Lagrange polynomials. Notice that near the end-points, the interpolating polynomials vary significantly away from $f(x)$, especially as the order $n$ increases. A reason for this is that polynomials of very high degree can vary wildly even in small intervals. For this example, the 51st Lagrange polynomial does very well in approximating $f$ on the interval $[-2, 4]$; the maximum error is $\max_{-2\le x\le 4}|f(x) - P(x)| \approx 0.1514$. However, as the order of the polynomial increases, the error actually increases.

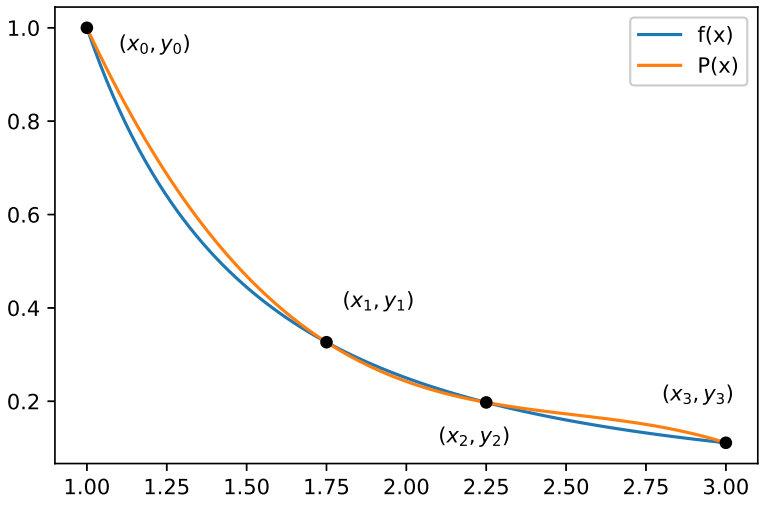



Figure 3.3: Graph of $f(x) = \frac{1}{x^2}$ and the 3rd order Lagrange interpolating polynomial through the nodes $x_0, x_1, x_2, x_3$ for Example 3.1.

In Algorithm 3.1, we implement Lagrange interpolation; the input are data points $(x_0, y_0)$, …, $(x_n, y_n)$ and points $u_1$, $u_2$, …, $u_N$, and the output are the values $P(u_1)$, $P(u_2)$, …, $P(u_N)$ where $P$ is $n$th Lagrange polynomial interpolating the $(x, y)$ data points.

**Algorithm 3.1** Lagrange Polynomial Interpolation

INPUT: $x = (x_0, x_1, \ldots, x_n)$, $y = (y_0, y_1, \ldots, y_n)$, $(u_0, u_1, \ldots, u_N)$

OUTPUT: The values $[v_0, v_1, \ldots, v_N]$ obtained by evaluating the $n$-order interpolating polynomial of the input data, that is, $v_i = P(u_i)$, where $P$ is the $n$th order Lagrange polynomial

1: **set** $v = \text{zeros}(1, N+1)$
2: **for** $k = 0, 1, 2, \ldots, n$ **do**
3: **for** $j = 0, 1, 2, \ldots, N$ **do**
4: **set** $L = 1$
5: **for** $i = 0, 1, 2, \ldots, k-1, k+1, \ldots, n$ **do**
6: **set** $L = \dfrac{u_j - x_i}{x_k - x_i} \cdot L$
7: **set** $v_j = v_j + y_k L$
8: **output** $[v_0, v_1, v_2, \ldots, v_N]$

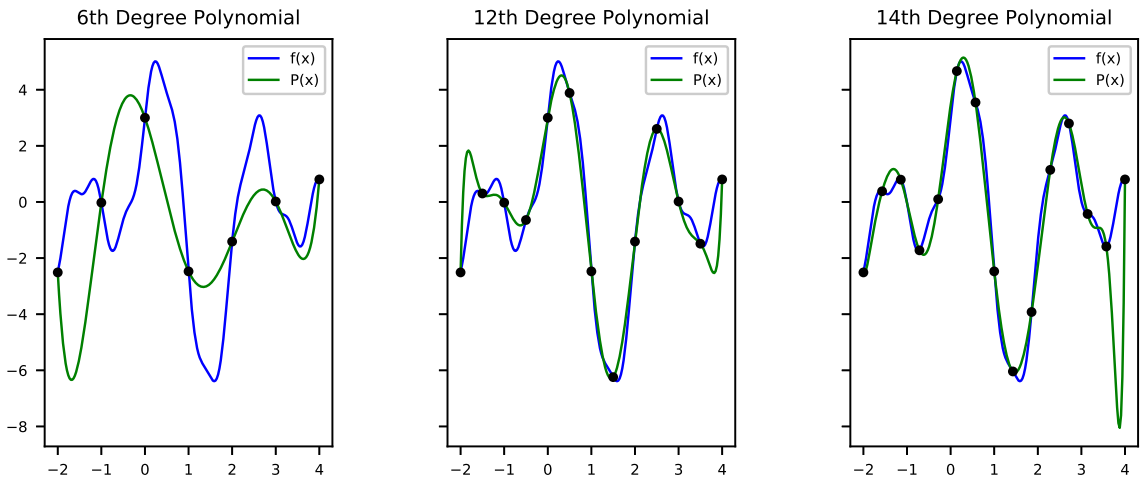



Figure 3.4: As the order of a Lagrange polynomial increases, the error in the interpolation in the inner nodes decreases but near the boundary nodes the error can vary wildly. In this example, the Lagrange polynomial of order $n = 14$ exhibits a large error near the last node point.

In Algorithm 3.1, the Lagrange polynomial

$$P(x) = \sum_{k=0}^{n} y_k L_k(x) = \sum_{k=0}^{n} y_k \prod_{i \neq k} \frac{x - x_i}{x_k - x_i}$$

is evaluated at $u_j$ by first computing for each $k$ the value $L = L_k(u_j) = \prod_{i \neq k} \frac{u_j - x_i}{x_k - x_i}$ and then updating the running sum of $v_j = v_j + y_k L$.

The following theorem will be useful to obtain an error bound for $|f(x) - P(x)|$ for each $x \in [a, b]$.

**Theorem 3.4**

Suppose that $x_0, x_1, \ldots, x_n$ are distinct and contained in the interval $[a, b]$ and $f^{(n+1)}$ is continuous on $[a, b]$. Let $P(x)$ be the $n$th Lagrange polynomial for $f$ at nodes $x_0, x_1, \cdots, x_n$. Then for each $x \in [a, b]$ there exists a number $\xi(x) \in (a, b)$ such that

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x - x_0)(x - x_1) \cdots (x - x_n).$$

*Proof.* When $x = x_j$ for some $j$ then both sides of the equation are zero regardless of the value of $\xi \in (a, b)$. Hence, suppose that $x \neq x_j$, and let $w(x) = \prod_{j=0}^{n}(x - x_j)$ and define the function $g : [a, b] \to \mathbb{R}$ by

$$g(t) = f(t) - P(t) - \frac{f(x) - P(x)}{w(x)} w(t).$$

Since $f^{(n+1)}$ is continuous, and $P$ and $w$ are polynomials, $g^{(n+1)}$ is also continuous. Moreover, $g$ has distinct roots $x, x_0, x_1, \ldots, x_n$. Thus, $g$ has $n+2$ distinct roots in the interval $[a,b]$. By applying Rolle's theorem successively to $g, g^{(1)}, \ldots, g^{(n)}$, we deduce that there exists $\xi(x) \in (a,b)$ such that $g^{(n+1)}(\xi(x)) = 0$. Since $P(t)$ is a polynomial of degree at most $n$ and $w^{(n+1)}(t) = (n+1)!$ (because $w(t)$ is monic of degree $n+1$) we obtain

$$0 = f^{(n+1)}(\xi(x)) - \frac{f(x) - P(x)}{w(x)}(n+1)!$$

and the result follows. □

Using the previous theorem, we will now determine an error bound for the remainder $|f(x) - P(x)|$ for the case of equally spaced nodes $a = x_0 < x_1 < \cdots < x_n = b$. Hence, let $h = \frac{b-a}{n}$ be the length of each subinterval $[x_j, x_{j+1}]$. Then, for $j = 0, 1, \ldots, n$ we have

$$x_j = a + jh.$$

For any $x \in (a,b)$, not equal to one of the nodes, we have that $x = a + sh$ for some non-integer value $s \in (0,n)$. Thus,

$$\prod_{j=0}^{n}(x - x_j) = \prod_{j=0}^{n}(a + hs - a - jh) = \prod_{j=0}^{n} h(s-j) = h^{n+1}\prod_{j=0}^{n}(s-j).$$

Therefore,

$$|f(x) - P(x)| \leq \frac{h^{n+1}}{(n+1)!}|f^{(n+1)}(\xi(x))|\prod_{j=0}^{n}|s-j|$$

To proceed we need the following technical lemma.

**Lemma 3.5**

Let $n \in \mathbb{N}$ and let $s \in (0,n)$. Then $\prod_{j=0}^{n}|s-j| \leq \frac{1}{4}n!$.

*Proof.* The proof is by induction. The base case $n = 1$ gives the polynomial $\prod_{j=0}^{1}|s-j| = s(s-1)$ which achieves a maximum absolute value of $\frac{1}{4}$ at $s = \frac{1}{2}$. Assume the claim holds for $n$ and consider $\prod_{j=0}^{n+1}|s-j|$. If $s \in (0,n)$ then $|s-(n+1)| \leq n+1$ and thus by the induction hypothesis

$$\prod_{j=0}^{n+1}|s-j| = |s-(n+1)|\prod_{j=0}^{n}|s-j| \leq (n+1)\frac{1}{4}n! = \frac{1}{4}(n+1)!.$$

If on the other hand $s \in (n, n+1)$ then let $r = (n+1-s) \in (0,1)$ and thus

$$\begin{aligned}\prod_{j=0}^{n+1} |s-j| &= \prod_{j=0}^{n+1} |s-(n+1)+(n+1)-j| \\ &= \prod_{j=0}^{n+1} |r+j-(n+1)| \\ &= \prod_{k=0}^{n+1} |r-k| \\ &\leq \frac{1}{4} n!(n+1)\end{aligned}$$

where in the last line we applied the induction hypothesis since $r \in (0,1) \subset (0,n)$ and also used $|r-(n+1)| < (n+1)$. □

Hence, by the previous lemma,

$$|f(x) - P(x)| \leq h^{n+1} \frac{1}{(n+1)!} \frac{1}{4} n! |f^{(n+1)}(\xi(x))|$$

and then finally

$$|f(x) - P(x)| \leq h^{n+1} \frac{1}{4(n+1)} \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)|.$$

Hence, if $f$ is such that $\max_{x\in[a,b]} |f^{(n+1)}(x)|$ does not grow to fast as $n$ grows, then $P$ will become a better approximation to $f$ as the order of the interpolating polynomial increases. The following examples illustrates this point.

**Example 3.3.** Consider $f(x) = xe^x$ on the interval $[a,b] = [0,2]$. Let $P$ be the $n$th order interpolating polynomial with equally-spaced nodes $x_0, x_1, \ldots, x_n$. Estimate the maximum error $|f(x) - P(x)|$ on the given interval when $n = 10$.

*Solution.* One can show that

$$f^{(n)}(x) = ne^x + xe^x = e^x(n+x)$$

For each $x \in [0,2]$ we have

$$|f^{(n)}(x)| = |e^x(n+x)| = e^x|n+x| \leq e^b(n+b).$$

Therefore, for all $x \in [a,b]$, we obtain

$$|f(x) - P(x)| \leq \frac{h^{n+1}}{4(n+1)} e^b(n+b) = \frac{e^b}{4} \left(\frac{b-a}{n}\right)^{n+1} (n+b+1)$$

When $n = 10$ we obtain

$$\frac{e^b}{4}\left(\frac{b-a}{n}\right)^{n+1}(n+b+1) \approx 4.9181 \times 10^{-7}$$

Therefore, for all $x \in [0, 2]$ it holds that

$$|f(x) - P(x)| \leq 4.9181 \times 10^{-7}$$

□

**Example 3.4.** Consider the function $f(x) = 3\sin(2x)$ on the interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. Let $P$ be the $n$th order interpolating polynomial with equally-spaced nodes $x_0, x_1, \ldots, x_n$. Estimate the maximum error $|f(x) - P(x)|$ on the given interval when $n = 20$.

*Solution.* We have proved that for all

$$|f(x) - P(x)| \leq \frac{h^{n+1}}{4(n+1)} \max_{\xi \in [a,b]} |f^{(n+1)}(\xi)|.$$

Compute the first few derivatives of $f$:

$$\begin{aligned} f^{(1)}(x) &= 3 \cdot 2\cos(2x) \\ f^{(2)}(x) &= -3 \cdot 2^2 \sin(2x) \\ f^{(3)}(x) &= -3 \cdot 2^3 \cos(2x) \\ f^{(4)}(x) &= 3 \cdot 2^4 \sin(2x) \end{aligned}$$

We can see a pattern emerging. When $n$ is odd, $f^{(n)}(x) = \pm 3 \cdot 2^n \cos(2x)$, and when $n$ is even $f^{(n)}(x) = \pm 3 \cdot 2^n \sin(2x)$. In either case, $|f^{(n)}(x)| \leq 3 \cdot 2^n$ for all $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$. Therefore, we have the estimate

$$|f(x) - P(x)| \leq \frac{h^{n+1}}{4(n+1)} \max_{x \in [a,b]} |f^{(n+1)}(x)| \leq \frac{h^{n+1}}{4(n+1)} 3 \cdot 2^{n+1}.$$

Now $h = \frac{b-a}{n} = \frac{\frac{\pi}{2} - (-\frac{\pi}{2})}{n} = \frac{\pi}{n}$ and therefore

$$|f(x) - P(x)| \leq \frac{3}{4(n+1)}\left(\frac{\pi}{n}\right)^{n+1} 2^{n+1} = \frac{3}{4(n+1)}\left(\frac{2\pi}{n}\right)^{n+1}$$

Now since $\lim_{n\to\infty} \frac{2\pi}{n} = 0$ then

$$\lim_{n\to\infty} \frac{3}{4(n+1)}\left(\frac{2\pi}{n}\right)^{n+1} = 0$$

Therefore, even though the magnitude of the derivatives of $f$ grow unboundedly, the interpolating polynomials $P$ become more accurate estimates to $f$ as $n \to \infty$. When $n = 20$ we obtain

$$\frac{3}{4(n+1)}\left(\frac{2\pi}{n}\right)^{n+1} \approx 9.840 \times 10^{-13}$$

Therefore, the maximum error of $|f(x) - P(x)|$ for all $x \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ is no bigger than approximately $9.840 \times 10^{-13}$. □

The following example generalizes the previous example.

**Example 3.5.** Consider using $n+1$ equally spaced nodes on the interval $[a, b]$ to interpolate the value of the function $f(x)$ at any $x \in [a, b]$. By the previous theorem, if $P$ is the $n$th order interpolating polynomial then

$$|f(x) - P(x)| \leq h^{n+1}\frac{1}{4(n+1)} \max_{\xi \in [0,\pi]} |f^{(n+1)}(\xi)|$$

Suppose that there exists some $M > 0$ and $C > 0$ such that $|f^{(n+1)}(x)| \leq CM^{n+1}$ for all $x \in [a, b]$. Then

$$|f(x) - P(x)| \leq \left(\frac{b-a}{n}\right)^{n+1} \frac{4}{(n+1)} CM^{n+1}$$

To simplify the notation, let $K = b - a$ and thus

$$|f(x) - P(x)| \leq \frac{4C}{(n+1)}\left(\frac{MK}{n}\right)^{n+1}$$

We will show that the sequence $p_n = \frac{4C}{(n+1)}\left(\frac{MK}{n}\right)^{n+1}$ converges to zero of order $O(a^{n+1})$ for every $0 < a < 1$. Let $0 < a < 1$ be arbitrary. Since $\lim_{n\to\infty} \frac{MK}{n} = 0$, there exists a natural number $N$ such that $\frac{MK}{n} < a$ for all $n \geq N$. Therefore, for all $n \geq N$ we have

$$\left(\frac{MK}{n}\right)^{n+1} < a^{n+1}$$

and then multiplying both sides of the last inequality by $\frac{4C}{n+1}$ we obtain

$$\left(\frac{MK}{n}\right)^{n+1} \frac{4C}{(n+1)} < a^{n+1}\frac{4C}{(n+1)} < 4Ca^{n+1}$$

This proves that $p_n = O\left(a^{n+1}\right)$. Therefore,

$$\lim_{n\to\infty} |f(x) - P(x)| = 0$$

with order of convergence $O\left(a^{n+1}\right)$ for any $0 < a < 1$.

**Example 3.6.** The following example, discovered by C. Runge (1901), shows that interpolation with equally spaced nodes does not always result in a decrease in the maximum error $|f(x) - P(x)|$ as the order of the interpolating polynomial $P$ increases, i.e., as we add more node points. In particular, as $n$ increases, the interpolating polynomial could exhibit wild oscillations near the boundary points of the interval. This is known as Runge's phenomenon. Consider the function

$$f(x) = \frac{1}{1 + 25x^2}$$

on the interval $[-1, 1]$. In Figure 3.5, we plot the interpolating polynomial for $n \in \{6, 12, 14\}$. Notice how near the end-points of the interval, the interpolating polynomial is a poor approximation to $f$. In fact, one can show that

$$\lim_{n\to\infty} \left( \max_{x\in[-1,1]} |f(x) - P(x)| \right) = \infty,$$

that is, the maximum error between $f$ and $P$ increases without bound as $n \to \infty$.

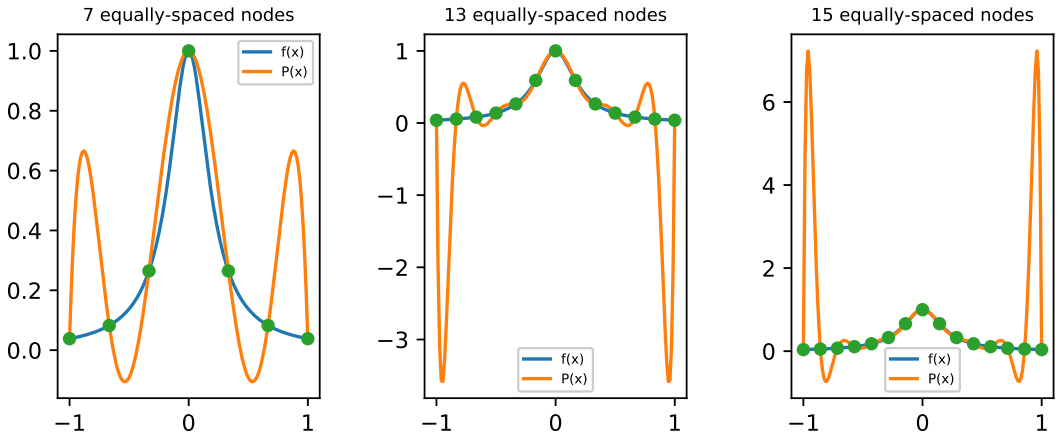



Figure 3.5: Equally spaced nodes does not always yield a good interpolating polynomial.

## 3.2 Chebyshev Polynomials and Interpolation Error Minimization

We introduce a set of polynomials whose roots give nodes $x_0, x_1, \ldots, x_n$ that minimize the error $|f(x) - P(x)|$ over all possible choices of the nodes. These polynomials are called the **Chebyshev polynomials** and are defined as follows. For $x \in [-1, 1]$ and an integer $n \geq 0$ define

$$T_n(x) = \cos(n \arccos(x)).$$

It is clear that $|T_n(x)| \leq 1$. What is not clear is that $T_n(x)$ is a polynomial for each $n$. To see how, we note that $T_0(x) = 1$ and $T_1(x) = x$. Now, using the identity $\cos(\alpha + \beta) = \cos(\alpha)\cos(\beta) - \sin(\alpha)\sin(\beta)$, we have

$$T_{n+1}(x) = \cos(n \arccos(x) + \arccos(x)) = xT_n(x) - \sin(n \arccos(x)) \sin(\arccos(x)).$$

Similarly, and using the fact that cosine is an odd function and sine is an even function, we obtain

$$T_{n-1}(x) = xT_n(x) + \sin(n \arccos(x)) \sin(\arccos(x)).$$

Therefore,

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x).$$

Since $T_0(x) = 1$ and $T_1(x) = x$, it follows that $T_n(x)$ is an $n$th order polynomial with leading coefficient $2^{n-1}$, that is,

$$T_n(x) = 2^{n-1}x^n + \cdots.$$

The first ten Chebyshev polynomials are

$$\begin{aligned}
T_0(x) &= 1 & T_5(x) &= 16x^5 - 20x^3 + 5x \\
T_1(x) &= x & T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1 \\
T_2(x) &= 2x^2 - 1 & T_7(x) &= 64x^7 - 112x^5 + 56x^3 - 7x \\
T_3(x) &= 4x^3 - 3x & T_8(x) &= 128x^8 - 256x^6 + 160x^4 - 32x^2 + 1 \\
T_4(x) &= 8x^4 - 8x^2 + 1 & T_9(x) &= 256x^9 - 576x^7 + 432x^5 - 120x^3 + 9x
\end{aligned}$$

Figure 3.6 shows the first few Chebyshev polynomials plotted over the interval $[-1, 1]$.

From the definition of Chebyshev polynomials, the $(n + 1)$ roots of $T_{n+1}(x)$ are readily available. For $j \in \{0, 1, \ldots, n\}$ let

$$x_j = \cos\left(\frac{2j + 1}{2n + 2}\pi\right) \tag{3.2}$$

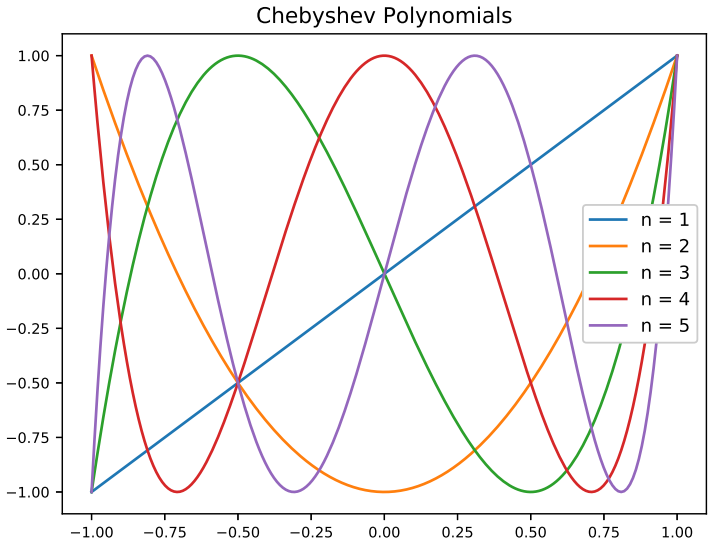



Figure 3.6: The first few Chebyshev polynomials.

and then

$$\begin{aligned} T_{n+1}(x_j) &= \cos\left((n+1)\frac{2j+1}{2n+2}\pi\right) \\ &= \cos((2j+1)\tfrac{\pi}{2}) \\ &= 0. \end{aligned}$$

Hence, $x_0, x_1, \ldots, x_n$ are the roots of the polynomial $T_{n+1}(x)$. We will call the roots (3.2) the $(n+1)$ **Chebyshev nodes** on the interval $[-1, 1]$. We also note that

$$T_{n+1}\left(\cos\left(\frac{j\pi}{(n+1)}\right)\right) = (-1)^j$$

for $j = 0, 1, \ldots, n+1$, and thus $T_{n+1}$ achieves its maximum and minimum value of 1 and $-1$, respectively, on the interval $[-1, 1]$. In Figure 3.7, we plot the Chebyshev nodes in the interval $[-1, 1]$ for $n = 20$.

Now since $T_{n+1}(x) = 2^n x^{n+1} + \cdots$, then if we set $Q_{n+1}(x) = 2^{-n} T_{n+1}(x)$, we wee that $Q_{n+1}(x)$ is a monic polynomial whose roots are the Chebyshev nodes $x_0, x_1, \ldots, x_n$. We may therefore write

$$Q_{n+1}(x) = (x - x_0)(x - x_1)\cdots(x - x_n).$$

Now, $|Q_{n+1}(x)| = |2^{-n} T_{n+1}(x)| \leq 2^{-n}$ for all $x \in [-1, 1]$ and thus

$$\max_{-1 \leq x \leq 1} |Q_{n+1}(x)| = 2^{-n}.$$

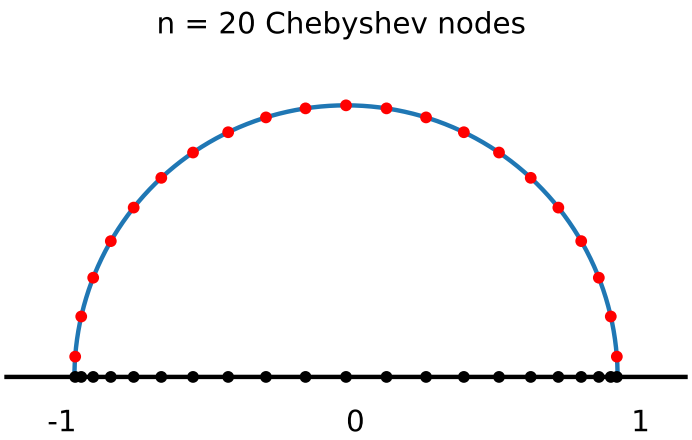



Figure 3.7: Chebyshev nodes (black) for $n = 20$. Notice how the nodes clutter near the boundary points of the interval. The red nodes are located at the angles $\theta_j = \frac{2j+1}{2n+2}\pi$

The following lemma states that of all the monic polynomials, this is the least maximum value on the interval $[-1, 1]$.

**Lemma 3.6: Optimality of Chebyshev Nodes**

Let $\tilde{Q}(x)$ be a monic polynomial of degree $n+1$. Then $\max_{-1\le x\le 1} |\tilde{Q}(x)| \ge 2^{-n}$. Moreover, if $\max_{-1\le x\le 1} |\tilde{Q}(x)| = 2^{-n}$ then $\tilde{Q}(x) = Q_{n+1}(x)$.

As a consequence, we obtain the following.

**Theorem 3.7**

et $f : [-1, 1] \to \mathbb{R}$ be such that $f^{(n+1)}$ is continuous on $[-1, 1]$. Let $x_0, x_1, \ldots, x_n$ be the $n+1$ Chebyshev nodes:

$$x_j = \cos\left(\frac{2j+1}{2n+2}\pi\right)$$

Let $P(x)$ be the Lagrange interpolating polynomial of the data set $(x_0, f(x_0)), \ldots, (x_n, f(x_n))$. Then for all $x \in [-1, 1]$ we have

$$|f(x) - P(x)| \le \frac{1}{2^n(n+1)!} \max_{x\in[-1,1]} |f^{(n+1)}(x)| \tag{3.3}$$

and this is the best possible bound among all $n$th order interpolating polynomials with nodes in $[-1, 1]$.

If $f$ is defined on $[a, b]$ (i.e., not necessarily $[-1, 1]$), then the zeros of the polynomial

$T_{n+1}(x)$ can be shifted to the interval $[a, b]$ by means of the following:

$$x_j = \frac{(a+b)}{2} + \frac{(b-a)}{2}\cos\left(\frac{2j+1}{2n+2}\pi\right) \tag{3.4}$$

for $j = 0, 1, \ldots, n$. We will call the points (3.4) the **Chebyshev nodes** on the interval $[a, b]$. In this case, we obtain the following.

**Theorem 3.8**

Let $f : [a, b] \to \mathbb{R}$ be such that $f^{(n+1)}$ is continuous on $[a, b]$. Let $x_0, x_1, \ldots, x_n$ be the $n + 1$ Chebyshev nodes in the interval $[a, b]$:

$$x_j = \frac{(a+b)}{2} + \frac{(b-a)}{2}\cos\left(\frac{2j+1}{2n+2}\pi\right)$$

Let $P(x)$ be the Lagrange interpolating polynomial of the data set $(x_0, f(x_0)), \ldots, (x_n, f(x_n))$. Then for all $x \in [a, b]$ we have

$$|f(x) - P(x)| \leq \left(\frac{b-a}{2}\right)^{n+1} \frac{1}{2^n(n+1)!} \max_{x\in[a,b]} |f^{(n+1)}(x)| \tag{3.5}$$

and this is the best possible bound among all $n$th order interpolating polynomials with nodes in $[a, b]$.

**Example 3.7.** Consider again the function

$$f(x) = 3\cos(2x) - \sin(0.5x) + 3\sin(3.3x) + 0.5\sin(10x)$$

on the interval $[-2, 4]$. We compute the $n$th order Lagrange polynomials for $n = 6, 12, 14$ at the Chebyshev nodes (3.4) and compare the results when the nodes are equally spaced. In Figure 3.8 we plot the error $|f(x) - P(x)|$ on $[-2, 4]$ for each case of $n$ using equally spaced and the Chebyshev nodes. Notice how with the Chebyshev points, the large error near the end point $b = 4$ is eliminated. We now prove that as $n \to \infty$, the polynomial $P(x)$ using the Chebyshev points approximates $f(x)$ uniformly. The function $f$ takes the form

$$f(x) = \sum_{k=1}^{N} A_k \cos(\omega_k x + \theta_k).$$

By induction, it is not hard to show that $f^{(n)}(x) = \sum_{k=1}^{N}(\pm 1)\omega_k^n A_k \cos(\omega_k x + \theta_k)$ when $n$ is even and $f^{(n)}(x) = \sum_{k=1}^{N}(\pm 1)\omega_k^n A_k \sin(\omega_k x + \theta_k)$ when $n$ is odd. In either case, since $|\sin(x)| \leq 1$ and $|\cos(x)| \leq 1$ for all $x \in \mathbb{R}$, we have

$$|f^{(n)}(x)| \leq \sum_{k=1}^{N} |\omega_k|^n |A_k| \leq AN\omega^n$$

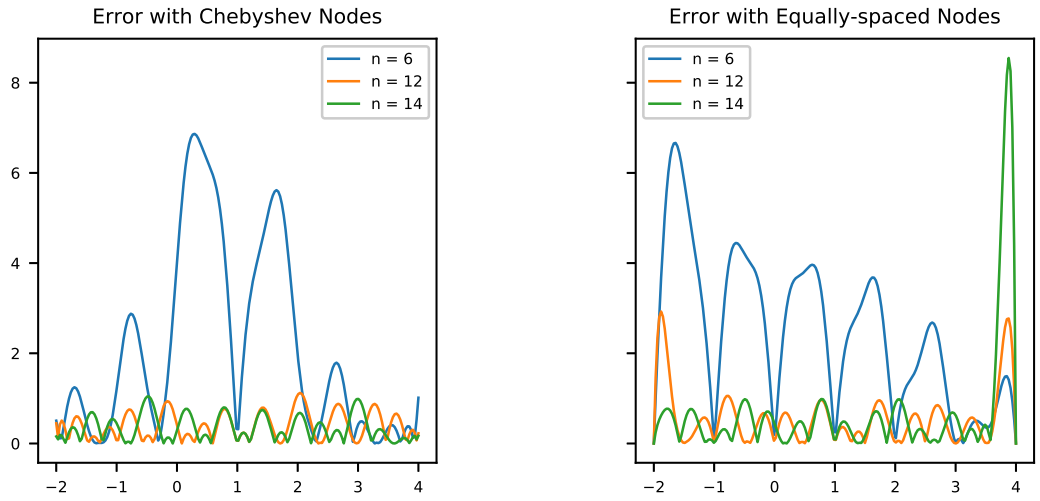



Figure 3.8: Error comparison with equally spaced and Chebyshev nodes. Notice that for $n = 14$, the error with the Chebyshev nodes does not exhibit the large error at the boundary points like with equally-spaced nodes.

where $\omega = \max |\omega_k|$ and $A = \max |A_k|$. It follows from (3.5) that when Chebyshev points are used for the polynomial $P(x)$ then

$$|f(x) - P(x)| \leq \frac{AN}{2^n(n+1)!}\omega^{n+1}.$$

Since $\lim_{n\to\infty} \frac{\omega^n}{n!} = 0$ for any $\omega \geq 0$ it follows that for any $\varepsilon > 0$ there exists sufficiently large $n$ such that $|f(x) - P(x)| < \varepsilon$ **for all** $x \in [a, b]$.

## 3.3 Newton's Divided Differences

In this section, we present an alternative algebraic representation of the interpolating polynomial of a given data set $\{(x_j, f(x_j))\}_{j=0}^{n}$. The method is attributed to Sir Isaac Newton and is called *Newton's Divided-Difference Formula*. A major advantage of the divided-difference representation of the interpolating polynomial is that very little work is required to compute an interpolating polynomial for the nodes $x_0, x_1, \ldots, x_{n+1}$ if we have the interpolating polynomial for the nodes $x_0, x_1, \ldots, x_n$. This makes it easy to add new nodes to, for example, increase the accuracy of the interpolating polynomial without having to recompute the polynomial from scratch.

In the divided-difference representation of the interpolating polynomial $P(x)$, we seek a representation of the form

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_0)\cdots(x - x_{n-1})$$

or more compactly

$$P(x) = a_0 + \sum_{k=1}^{n} a_k(x - x_0)(x - x_1)\cdots(x - x_{k-1}).$$

The goal then is to compute the *divided-difference coefficients* $a_0, a_1, a_2, \ldots, a_n$. To that end, we proceed by imposing the constraint that $P(x_j) = f(x_j)$ and solving for the coefficients one at a time. To begin, we have

$$P(x_0) = a_0$$

and therefore

$$a_0 = f(x_0).$$

Next, we have

$$P(x_1) = a_0 + a_1(x_1 - x_0)$$

and therefore if $P(x_1) = f(x_1)$ then after rearranging, and using $a_0 = f(x_0)$, we obtain

$$a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Next, we have

$$P(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1)$$

and therefore if $P(x_2) = f(x_2)$, and using the previously computed expressions for $a_0$ and $a_1$, then after some rearranging we obtain

$$a_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}. \tag{3.6}$$

We continue in this manner and find the remaining coefficients $a_3, \ldots, a_n$. At this point, we make the observation that the coefficient $a_k$ depends only on the nodes $x_0, x_1, \ldots, x_k$, and therefore if a new node $x_{n+1}$ is added, we need only compute the coefficient $a_{n+1}$. This is one of the advantages with the divided-difference representation of $P(x)$. In particular, if $P_n(x)$ is the interpolating polynomial for the nodes $x_0, x_1, \ldots, x_n$ and $P_{n+1}(x)$ is the interpolating polynomial for the nodes $x_0, x_1, \ldots, x_n, x_{n+1}$ then

$$P_{n+1}(x) = P_n(x) + a_{n+1}(x - x_0)(x - x_1)\cdots(x - x_n)$$

and thus if $P_n(x)$ is known then we need only compute $a_{n+1}$ to obtain $P_{n+1}(x)$. This point is llustrated in Example 3.9 below.

In practice, the coefficients $a_1, a_2, \ldots, a_n$ are computed using a recurrence relation involving what are called *divided-differences* of $f$ with respect to the nodes $x_0, x_1, \ldots, x_n$. For nodes $x_i$ and $x_{i+1}$ the divided-difference $f[x_i, x_{i+1}]$ is defined as

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Using this notation, the coefficient $a_1$ can be written as

$$a_1 = f[x_0, x_1] = \frac{f(x_1) - f(x_0)}{x_1 - x_0}.$$

Similarly, for nodes $x_i, x_{i+1}, x_{i+2}$, the divided difference $f[x_i, x_{i+1}, x_{i+2}]$ is defined as

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

With this notation, from (3.6) the coefficient $a_2$ can be written as

$$a_2 = f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

In general, given nodes $x_i, x_{i+1}, \ldots, x_j$, where $0 \le i < j \le n$, we define the divided difference $f[x_i, x_{i+1}, \ldots, x_j]$ by

$$f[x_i, x_{i+1}, \ldots, x_j] = \frac{f[x_{i+1}, \ldots, x_j] - f[x_i, \ldots, x_{j-1}]}{x_j - x_i}.$$

With this notation, it can be shown by induction that the coefficient $a_k$ for $1 \le k \le n$ is given by

$$a_k = f[x_0, x_1, \ldots, x_k].$$

There is a very efficient computational way to compute the coefficients $a_1, a_2, \ldots, a_n$ that is best explained with a table. Suppose that we want to compute the divided-differences

| $i =$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $x$ | $f[\cdot]$ | $f[\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot,\cdot]$ |
| $x_0$ | $f[x_0]$ | $\star$ | $\star$ | $\star$ | $\star$ |
| $x_1$ | $f[x_1]$ | $f[x_0,x_1]$ | $\star$ | $\star$ | $\star$ |
| $x_2$ | $f[x_2]$ | $f[x_1,x_2]$ | $f[x_0,x_1,x_2]$ | $\star$ | $\star$ |
| $x_3$ | $f[x_3]$ | $f[x_2,x_3]$ | $f[x_1,x_2,x_3]$ | $f[x_0,x_1,x_2,x_3]$ | $\star$ |
| $x_4$ | $f[x_4]$ | $f[x_3,x_4]$ | $f[x_2,x_3,x_4]$ | $f[x_1,x_2,x_3,x_4]$ | $f[x_0,x_1,x_2,x_3,x_4]$ |

Table 3.1: Divided-difference table with $n = 4$

representation of $P(x)$ given the nodes $x_0, x_1, x_2, x_3, x_4$ and the function values $f(x_0)$, $f(x_1)$, $f(x_2)$, $f(x_3)$, $f(x_4)$. To compute the coefficients $a_k = f[x_0, x_1, \ldots, x_k]$ we must compute all the divided-differences shown in Table 3.1. Notice that the divided-difference coefficients $a_k = f[x_0, x_1, \ldots, x_k]$ are contained along the diagonal of the table.

Alternatively, we can view the table as a $(n+1) \times (n+1)$ matrix:

$$\mathbf{F} = \begin{pmatrix} f[x_0] & 0 & 0 & 0 & 0 \\ f[x_1] & f[x_0,x_1] & 0 & 0 & 0 \\ f[x_2] & f[x_1,x_2] & f[x_0,x_1,x_2] & 0 & 0 \\ f[x_3] & f[x_2,x_3] & f[x_1,x_2,x_3] & f[x_0,x_1,x_2,x_3] & 0 \\ f[x_4] & f[x_3,x_4] & f[x_2,x_3,x_4] & f[x_1,x_2,x_3,x_4] & f[x_0,x_1,x_2,x_3,x_4] \end{pmatrix}$$

Notice that the divided-difference coefficients $a_k = f[x_0, x_1, \ldots, x_k]$ are the diagonal entries of $\mathbf{F}$. The matrix $\mathbf{F}$ is created one column at a time, starting with the first. The first column $\mathbf{F}[:, 0]$ is simply the given data values $f(x_0)$, $f(x_1), \ldots, f(x_n)$. Then, once the entries of $\mathbf{F}$ in the $i-1$ column have been computed, to compute the entry of $\mathbf{F}$ in the $i$th column and $j$th row (using zero-based indexing), where $1 \leq i \leq n$ and $i \leq j \leq n$, we use

$$\mathbf{F}(j,i) = \frac{\mathbf{F}(j,i-1) - \mathbf{F}(j-1,i-1)}{x_j - x_{j-i}}.$$

For example, consider $f[x_1, x_2, x_3] = \frac{f[x_2,x_3]-f[x_1,x_2]}{x_3-x_1}$. We have that $\mathbf{F}(3,2) = f[x_1, x_2, x_3]$,

$\mathbf{F}(3,1) = f[x_2, x_3]$, and $\mathbf{F}(2,1) = f[x_1, x_2]$, and therefore

$$\mathbf{F}(3,2) = \frac{\mathbf{F}(3,1) - \mathbf{F}(2,1)}{x_3 - x_1}.$$

In Algorithm 3.2, we write an algorithm that computes $a_0, a_1, \ldots, a_n$ using this approach.

**Algorithm 3.2** Divided Differences Coefficients

INPUT: $x = [x_0, x_1, \ldots, x_n]$, $y = [f(x_0), f(x_1), \ldots, f(x_n)]$

OUTPUT: The divided-differences coefficients $a_k = f[x_0, x_1, \ldots, x_k]$, $0 \le k \le n$

1: **set** $F = \text{zeros}(n+1, n+1)$

2: **set** $F[:, 0] = y$ # fills the first column with $y$ data

3: **for** $i = 1, 2, \ldots, n$ **do**

4: **for** $j = i, i+1, \ldots, n$ **do**

5: **set** $F[j, i] = \dfrac{\mathbf{F}[j, i-1] - \mathbf{F}[j-1, i-1]}{x_j - x_{j-i}}$.

6: **output** $[F[0,0], F[1,1], F[2,2], \ldots, F[n,n]]$

Once the divided-difference coefficients have been computed, we can evaluate the interpolating polynomial at any given points $u = (u_0, u_1, \ldots, u_N)$; this is done in Algorithm 3.3.

**Algorithm 3.3** Divided-Difference Interpolation

INPUT: $a = [a_0, a_1, \ldots, a_n]$ (divided-differences coefficients), $u = [u_0, u_1, \ldots, u_N]$

OUTPUT: The values $[v_0, v_1, \ldots, v_N]$ obtained by evaluating the $n$-order interpolating polynomial of the input data, that is, $v_i = P(u_i)$, where $P$ is the $n$th order interpolating polynomial in divided-difference form

1: **set** $v = [a_0, a_0, \ldots, a_0]$ # of length $N+1$

2: **for** $k = 1, 2, \ldots, n$ **do**

3: $s = [a_k, a_k, \ldots, a_k]$ # of length $N+1$

4: **for** $j = 0, 1, \ldots, k-1$

5: $s = (u - x_j) * s$ # vectorized version, $u$ and $s$ must be NUMPY arrays!

6: $v = v + s$

7: **output** $v$

**Example 3.8.** Given the following data, compute the divided-difference table and $P(3)$, where $P(x)$ is the interpolating polynomial of the data.

$$x = (1, 2, 4, 5, 7)$$
$$y = (1, 3, 3, 4, 2)$$

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $x$ | $f[\cdot]$ | $f[\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot,\cdot]$ |
| 1 | 1 | $\star$ | $\star$ | $\star$ | $\star$ |
| 2 | 3 | 2 | $\star$ | $\star$ | $\star$ |
| 4 | 3 | 0 | $-\frac{2}{3}$ | $\star$ | $\star$ |
| 5 | 4 | 1 | $\frac{1}{3}$ | $\frac{1}{4}$ | $\star$ |
| 7 | 2 | $-1$ | $-\frac{2}{3}$ | $-\frac{1}{5}$ | $-\frac{3}{40}$ |

Table 3.2: Divided difference table for Example 3.8

*Solution.* The divided-difference table is shown in Table 3.2. The divided-differences coefficients are $a = (1, 2, -2/3, 1/4, -3/40)$ and therefore the interpolating polynomial is

$$P(x) = 1+2(x-1)-\frac{2}{3}(x-1)(x-2)+\frac{1}{4}(x-1)(x-2)(x-4)-\frac{3}{40}(x-1)(x-2)(x-4)(x-5).$$

Therefore,

$$P(3) = 1 + 2(2) - \frac{2}{3}(2)(1) + \frac{1}{4}(2)(1)(-1) - \frac{3}{40}(2)(1)(-1)(-2) = \frac{43}{15}.$$

□

**Example 3.9.** Suppose that we add the data point $(x_5, y_5) = (6, 5)$ to the data set in the previous example, so that now

$$x = (1, 2, 4, 5, 7, 6)$$
$$y = (1, 3, 3, 4, 2, 5).$$

Update the divided difference table and recompute $P(3)$ using the new higher-order interpolating polynomial.

*Solution.* The updated divided-difference table is shown in Table 3.3. Therefore, using the result from the previous example, we obtain

$$P(3) = \frac{43}{15} - \frac{1}{120}(3-1)(3-2)(3-4)(3-5)(3-7) = 3.$$

□

We end this section by presenting an interesting connection between $a_k = f[x_0, x_1, \ldots, x_k]$ and the $k$th derivative of $f$.

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $x$ | $f[\cdot]$ | $f[\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot,\cdot]$ |
| 1 | 1 | $\star$ | $\star$ | $\star$ | $\star$ | $\star$ |
| 2 | 3 | 2 | $\star$ | $\star$ | $\star$ | $\star$ |
| 4 | 3 | 0 | $-\frac{2}{3}$ | $\star$ | $\star$ | $\star$ |
| 5 | 4 | 1 | $\frac{1}{3}$ | $\frac{1}{4}$ | $\star$ | $\star$ |
| 7 | 2 | $-1$ | $-\frac{2}{3}$ | $-\frac{1}{5}$ | $-\frac{3}{40}$ | $\star$ |
| 6 | 5 | $-3$ | $-2$ | $-\frac{2}{3}$ | $-\frac{7}{60}$ | $-\frac{1}{120}$ |

Table 3.3: Divided difference table for Example 3.9

**Theorem 3.9**

Let $f : [a, b] \to \mathbb{R}$ be such that $f^{(k)}$ is continuous on $[a, b]$ and let $x_0, x_1, \ldots, x_k$ be distinct nodes contained in the interval $[a, b]$. Then there exists a point $\xi \in [a, b]$ such that

$$f[x_0, x_1, \ldots, x_k] = \frac{f^{(k)}(\xi)}{k!}.$$

*Proof.* Let $P(x)$ be the interpolating polynomial for the the data $y_j = f(x_j)$, $j = 0, 1, \ldots, k$, and let

$$g(x) = f(x) - P(x).$$

Then $g(x_j) = 0$ for $j = 0, 1, \ldots, k$, and thus $g$ has $k+1$ zeros in $[a, b]$. Since $f^{(k)}$ is continuous on $[a, b]$, and $P$ is a polynomial, $g^{(k)}$ is continuous, and therefore by Rolle's theorem, $g^{(k)}$ has a zero $\xi \in [a, b]$. Therefore,

$$0 = f^{(k)}(\xi) - P^{(k)}(\xi).$$

Using Newton's divided difference formula, we have

$$P(x) = f(x_0) + \sum_{j=1}^{k} f[x_0, x_1, \ldots, x_j](x - x_0) \cdots (x - x_{j-1}) = f[x_0, x_1, \ldots, x_k]x^k + \cdots$$

and therefore $P^{(k)}(x) = f[x_0, x_1, \ldots, x_{k-1}]k!$. Therefore,

$$0 = f^{(k)}(\xi) - f[x_0, x_1, \ldots, x_{k-1}]k!$$

and this ends the proof. □

**Example 3.10.** Fill in the missing entries in the divided-difference Table 3.4

| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $x$ | $f[\cdot]$ | $f[\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot,\cdot]$ | $f[\cdot,\cdot,\cdot,\cdot,\cdot,\cdot]$ |
| $-2$ | $-1$ | $\star$ | $\star$ | $\star$ | $\star$ | $\star$ |
| $-1$ | $0$ | $1$ | $\star$ | $\star$ | $\star$ | $\star$ |
| $0$ | $-1$ | $-1$ | $-1$ | $\star$ | $\star$ | $\star$ |
| $1$ | $3$ | $\#$ | $\frac{5}{2}$ | $\frac{7}{6}$ | $\star$ | $\star$ |
| $2$ | $0$ | $-3$ | $\#$ | $\#$ | $\#$ | $\star$ |
| $3$ | $1$ | $1$ | $2$ | $\frac{11}{6}$ | $\frac{23}{24}$ | $\#$ |

Table 3.4: Divided difference table for Example 3.10

**Example 3.11.** Compute the divided-difference coefficients of the interpolating polynomial $P(x)$ for the given data and then write $P(x)$ in the standard monomial basis.

$$x = (-1, 0, 1, 2)$$
$$y = (1, 0, -1, 4)$$

**Example 3.12.** Use the Lagrange form and the divided-difference form of the interpolating polynomial for the data $\{(x_j, f(x_j))\}_{j=0}^n$ to show that

$$f[x_0, x_1, \ldots, x_n] = \sum_{k=0}^{n} \frac{f(x_k)}{\prod_{j\neq k}(x_k - x_j)} = \sum_{k=0}^{n} \frac{f(x_k)}{w_n'(x_k)}.$$

where $w_n(x) = \prod_{j=0}^n (x - x_j)$.

*Solution.* The divided-difference form of the polynomial is

$$P(x) = a_0 + \sum_{j=1}^{n} f[x_0, x_1, \ldots, x_j](x - x_0)(x - x_1)\cdots(x - x_{j-1})$$

The number $f[x_0, x_1, \ldots, x_n]$ is the coefficient of the monomial $x^n$ in $P(x)$. On the other hand, the Lagrange form of the polynomial is

$$P(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \cdots + f(x_n)L_n(x)$$

where $L_k(x) = \prod_{j\neq k} \frac{(x-x_j)}{x_k - x_j}$. Each of $f(x_k)L_k(x)$ is an $n$th order polynomial, whose coefficient is

$$\frac{f(x_k)}{\prod_{j\neq k}(x_k - x_j)}$$

Therefore, the coefficient of $P(x)$ in the Lagrange form is

$$\sum_{k=0}^{n} \frac{f(x_k)}{\prod_{j\neq k}(x_k - x_j)}$$

Hence,

$$f[x_0, x_1, \ldots, x_n] = \sum_{k=0}^{n} \frac{f(x_k)}{\prod_{j\neq k}(x_k - x_j)}$$

and this proves the firs equality. Now if $w_n(x) = \prod_{j=0}^{n}(x - x_j)$ then by the product rule of differentiation we have

$$w_n'(x) = \sum_{j=0}^{n} \prod_{j\neq k} (x - x_j)$$

and on evaluating at $x_k$ we obtain

$$w_n'(x_k) = \prod_{j\neq k}(x_k - x_j).$$

and thus

$$f[x_0, x_1, \ldots, x_n] = \sum_{k=0}^{n} \frac{f(x_k)}{w_n'(x_k)}.$$

□

## 3.4 Hermite Polynomials

Let $x_0, x_1, \ldots, x_n \in [a, b]$ be a given set of nodes and let $f : [a, b] \to \mathbb{R}$ be a given function. Thus far, we have considered interpolation when only $f(x_0), f(x_1), \ldots, f(x_n)$ are known but we may also know the derivatives of $f$ at the nodes. That is, given non-negative integers $m_0, m_1, \ldots, m_n$, we may know the values

$$\begin{gathered} f(x_0),\ f^{(1)}(x_0), \ldots,\ f^{(m_0)}(x_0) \\ f(x_1),\ f^{(1)}(x_1), \ldots,\ f^{(m_1)}(x_1) \\ \cdots \\ f(x_n),\ f^{(1)}(x_n), \ldots,\ f^{(m_n)}(x_n) \end{gathered}$$

with $m_0, m_1, \ldots, m_n$ not all necessarily the same. We may therefore be interested in computing the polynomial $P$ that agrees with $f$ and its derivatives at the given nodes.

**Definition 3.10**

Let $x_0, x_1, \ldots, x_n \in [a, b]$ be distinct nodes, let $m_0, m_1, \ldots, m_n$ be non-negative integers and let $m = \max\{m_0, m_1, \ldots, m_n\}$. The **osculating polynomial** of $f \in C^m([a, b])$ at the given nodes $x_0, x_1, \ldots, x_n$ and orders $m_0, m_1, \ldots, m_n$, is the polynomial $P(x)$ of least

degree such that

$$P^{(k)}(x_i) = f^{(k)}(x_i)$$

for all $k = 0, 1, \ldots, m_i$ and all $i = 0, 1, \ldots, n$

Osculating polynomials generalize both the Taylor polynomials and the Lagrange polynomials. When $n = 0$, i.e., only one node $x_0$ is given, then $P$ is the Taylor polynomial of $f$ of order $m_0$ centered at $x_0$. And when $m_i = 0$ for all $i$ then $P$ is the Lagrange polynomial at the nodes $x_0, x_1, \ldots, x_n$.

When $m_i \geq 1$ then an osculating polynomial will have the same shape as $f$ at the node $x_i$. For instance, if $m_i \geq 1$ then both $f$ and $P$ have the same tangent line at $x_i$, and if $m_i \geq 2$ then $f$ and $P$ curve in the same direction. The word *osculate* originates from the Latin word *osculum* that means "kiss".

Given the nodes $x_0, \ldots, x_n$ and the numbers $m_0, m_1, \ldots, m_n$, the number of conditions on $f$ to compute the osculating polynomial is $(n+1) + \sum_{i=0}^{n} m_i$. Hence, the order of the osculating polynomial is at most $M = n + \sum_{i=0}^{n} m_i$.

A special case of the osculating polynomials is when $m_i = 1$ for all $i$, and these are called the **Hermite polynomials**. Hence, in this case we are given nodes $x_0, x_1, \ldots, x_n$ and the data $f(x_0), f'(x_0), f(x_1), f'(x_1), \ldots, f(x_n), f'(x_n)$. Thus, the order of the osculating polynomial is at most $2n + 1$. The following gives an exact formula for the osculating polynomial in this case.

**Theorem 3.11**

Let $f \in C^1([a, b])$ and let $x_0, x_1, \ldots, x_n \in [a, b]$. The Hermite polynomial $H(x)$ of $f$ at the nodes $x_0, x_1, \ldots, x_n$ is the osculating polynomial of degree at most $2n + 1$ given by

$$H(x) = \sum_{k=0}^{n} f(x_k) H_k(x) + \sum_{k=0}^{n} f'(x_k) \hat{H}_k(x)$$

where

$$H_k(x) = [1 - 2(x - x_k) L_k'(x)] \, L_k^2(x)$$

$$\hat{H}_k(x) = (x - x_k) L_k^2(x).$$

In addition, if $f \in C^{2n+2}([a, b])$ then for $x \in [a, b]$ we have

$$f(x) - H(x) = \frac{(x - x_0)^2 (x - x_1)^2 \cdots (x - x_n)^2}{(2n + 2)!} f^{(2n+2)}(\xi(x))$$

for some $\xi(x) \in (a, b)$.

*Proof.* It is clear that $H_k(x_j) = \hat{H}_k(x_j) = 0$ if $j \neq k$ and $H_k(x_k) = 1$ and $\hat{H}_k(x_k) = 0$. Hence, $H(x_j) = f(x_j)$ for all $j = 0, 1, \ldots, n$. We compute

$$H_k'(x) = [-2L_k'(x) - 2(x - x_k)L_k''(x)]L_k^2(x) + 2[1 - 2(x - x_k)L_k'(x)]L_k(x)L_k'(x)$$

$$\hat{H}_k'(x) = L_k^2(x) + 2(x - x_k)L_k(x)L_k'(x).$$

Now since $L_k(x_j) = 0$ if $j \neq k$ then $H_k'(x_j) = \hat{H}_k'(x_j) = 0$. Now,

$$H_k'(x_k) = -2L_k'(x_k) + 2L_k(x_k)L_k'(x_k) = 0.$$

Hence, $H_k'(x_j) = 0$ for all $j = 0, 1, \ldots, n$. Similarly,

$$\hat{H}_k'(x_k) = 1.$$

Therefore,

$$H'(x_j) = \sum_{k=0}^{n} [f(x_k)H_k'(x_j) + f'(x_k)\hat{H}_k'(x_j)] = f'(x_j)$$

This proves the first claim. To prove the error formula, we use the same procedure as in the Lagrange interpolation case using the function

$$g(t) = f(t) - H(t) - \frac{(t - x_0)^2 \cdots (t - x_n)^2}{(x - x_0)^2 \cdots (x - x_n)^2} [f(x) - H(x)].$$

The details are left to the reader. □

There is a way to compute the Hermite polynomials $H(x)$ using a divided-difference table. Suppose that $x_0, x_1, \ldots, x_n$ are distinct nodes, and $f$ and $f'$ are given at the nodes. Define a new sequence of nodes $z_0, z_1, \ldots, z_{2n}, z_{2n+1}$ by setting

$$z_0 = z_1 = x_0$$

$$z_2 = z_3 = x_1$$

$$\vdots$$

$$z_{2n} = z_{2n+1} = x_n$$

In other words,

$$z_{2i} = z_{2i+1} = x_i$$

for $i = 0, 1, \ldots, n$. Construct the divided-difference table with the data $z_0, z_1, \ldots, z_{2n}, z_{2n+1}$ and $f(z_0), f(z_1), \ldots, f(z_{2n}), f(z_{2n+1})$. Since $z_{2i} = z_{2i+1}$, the divided-differences $f[z_{2i}, z_{2i+1}]$ cannot be computed because we obtain

$$f[z_{2i}, z_{2i+1}] = \frac{f[z_{2i+1}] - f[z_{2i}]}{z_{2i+1} - z_{2i}} = \frac{0}{0}.$$

It is reasonable to use the estimate

$$f[z_{2i}, z_{2i+1}] \approx f'(z_{2i}) = f'(x_i)$$

Hence, in the column of the divided-difference table containing the terms $f[\cdot,\cdot]$, we replace $f[z_{2i}, z_{2i+1}]$ with $f'(x_i)$. The other terms of the divided-difference table are computed as usual. The Hermite polynomial is then given by

$$H(z) = f[z_0] + \sum_{k=1}^{2n+1} f[z_0, z_1, \ldots, z_k](z - z_0)(z - z_1)\cdots(z - z_{k-1}).$$

**Example 3.13.** Consider the data

$$x = (-1, 0, 1)$$
$$y = (-2, 3, 2)$$
$$y' = (14, -1, 2)$$

where $y_i = f(x_i)$ and $y_i' = f(x_i)$, for $i = 0, 1, 2$. Find the Hermite polynomial $H(z)$ for the given data and then find $H(-2)$.

**Example 3.14.** Write a Python function called `HermiteInterp` that takes as input arrays $x = (x_0, x_1, \ldots, x_n)$, $y = (y_0, y_1, \ldots, y_n)$, $y' = (y_0', y_1', \ldots, y_n')$, and $u = (u_0, u_1, \ldots, u_N)$, and outputs $v = (v_0, v_1, \ldots, v_N)$ where $v_i = H(u_i)$ for $i = 0, 1, \ldots, N$ and where $H$ is the Hermite polynomial for the given data. The $y$ and $y'$ arrays are assumed to contain the values of $f$ and $f'$, respectively, at the given nodes.

## 3.5 Piecewise Interpolation with Cubic Splines

In the past sections, we have considered interpolation of a set of data points contained in the interval $[a, b]$ using a single polynomial. We saw, however, that polynomials can deviate significantly in between interpolating points. Although we were able to remedy this using Chebyshev nodes, one needs to increase the order of the interpolating polynomial to get better accuracy. In this section, we take a different approach and use multiple polynomials for interpolation on an interval $[a, b]$. Specifically, for each subinterval $[x_{j-1}, x_j]$ of $[a, b]$, we use one polynomial to interpolate the data points $(x_{j-1}, y_{j-1})$ and $(x_j, y_j)$, and use a different polynomial for the next interval $[x_j, x_{j+1}]$. The resulting interpolating function is a **piecewise-polynomial** and since it interpolates the data points it is continuous. The simplest and easiest to compute piecewise-polynomial approximation is a **piecewise-linear approximation**. The line connecting the points $(x_{j-1}, y_{j-1})$ and $(x_j, y_{j+1})$ is

$$\ell_j(x) = y_{j-1} + \frac{(y_j - y_{j-1})}{(x_j - x_{j-1})}(x - x_{j-1})$$

and thus the piecewise-linear approximation interpolating $(x_0, x_1), \ldots, (x_n, y_n)$ is

$$F(x) = \begin{cases} \ell_1(x), & x_0 \le x < x_1 \\ \ell_2(x), & x_1 \le x < x_2 \\ \vdots & \vdots \\ \ell_n(x), & x_{n-1} \le x \le x_n \end{cases}$$

An example of piecewise-linear interpolation is shown in Figure 3.1. As the figure shows, in general, a piecewise-linear interpolating function is not differentiable at the nodes, and thus one needs to add derivative constraints at the inner nodes so that the derivatives at these nodes are equal for interpolating polynomials on adjacent subintervals. If one uses quadratic polynomials there are not enough coefficients to specify the derivatives at the end-points of the interval, in addition to the interpolating conditions at the end-points. For this reason, we need to use cubic polynomials on each subinterval and thereby construct a **piecewise-cubic polynomial** better known as **cubic splines**. In the general construction of a cubic spline, the derivatives of the interpolating function at the inner nodes do not necessarily have to equal the derivatives of the function being approximated.

**Definition 3.12**

Let $f : [a, b] \to \mathbb{R}$ be a given function and let $a = x_0 < x_1 < \cdots < x_n = b$ be a given set of nodes. A **cubic spline interpolant** $S$ for $f$ is a piecewise function that satisfies the following:

(a) $S(x)$ is a cubic polynomial, denoted $S_j(x)$, on the interval $[x_{j-1}, x_j]$ for $j = 1, \ldots, n$
(b) $S_j(x_{j-1}) = f(x_{j-1})$ and $S_j(x_j) = f(x_j)$ for $j = 1, \ldots, n$
(c) $S'_j(x_j) = S'_{j+1}(x_j)$ for $j = 1, \ldots, n-1$
(d) $S''_j(x_j) = S''_{j+1}(x_j)$ for $j = 1, \ldots, n-1$
(e) At the boundary nodes, one of the conditions is satisfied:
  (i) $S''(x_0) = S''(x_n) = 0$ (**natural** or **free** boundary condition)
  (ii) $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$ (**clamped boundary** condition)

**Remark 3.1.** The word **natural** is used in part e(i) because the shape of the approximating function at the end-points is the shape of a line after it passes through the end-points.

The total number of coefficients in the polynomials $S_1(x), S_2(x), \ldots, S_n(x)$ is $4n$ since each polynomial contains 4 coefficients. We seek the following form for each cubic polynomial $S_j(x)$:

$$S_j(x) = a_j + b_j(x - x_{j-1}) + c_j(x - x_{j-1})^2 + d_j(x - x_{j-1})^3$$

and thus the $4n$ unknowns are $\{a_j\}_{j=1}^n, \{b_j\}_{j=1}^n, \{c_j\}_{j=1}^n, \{d_j\}_{j=1}^n$. Now, condition (b) generates $2n$ equations. Each of condition (c) and (d) generates $n-1$ equations. The running

total is therefore $4n-2$. Condition (e) generates 2 more equations for a grand total of exactly $4n$ equations. The resulting set of equations in the $4n$ unknowns is therefore a square linear system.

**Example 3.15.** Construct a natural cubic spline through the points $(1,2)$, $(2,3)$, and $(3,5)$.

*Solution.* There are two intervals, namely, $[x_0,x_1]=[1,2]$ and $[x_1,x_2]=[2,3]$. Let $S_1(x)$ and $S_2(x)$ denote the approximating cubic polynomials on each interval, respectively. Thus,

$$S(x)=\begin{cases} S_1(x), 1\le x<2 \\ \\ S_2(x), 2\le x\le 3 \end{cases}$$

where

$$S_1(x)=a_1+b_1(x-1)+c_1(x-1)^2+d_1(x-1)^3$$

$$S_2(x)=a_2+b_2(x-2)+c_2(x-2)^2+d_2(x-2)^3.$$

There are 8 coefficients so we must impose 8 conditions for a unique solution. The interpolating conditions are:

$$\begin{aligned} 2&=f(1)=S_1(1)=a_1 \\ 3&=f(2)=S_1(2)=a_1+b_1+c_1+d_1 \\ 3&=f(2)=S_2(2)=a_2 \\ 5&=f(3)=S_2(3)=a_2+b_2+c_2+d_2 \end{aligned}$$

From the conditions $S_1'(2)=S_2'(2)$ and $S_1''(2)=S_2''(2)$ we obtain

$$b_1+2c_1+3d_1=b_2, \qquad 2c_1+6d_1=2c_2$$

The natural boundary conditions $S_1''(1)=0$ and $S_2''(3)=0$ give

$$2c_1=0, \qquad 2c_2+6d_2=0$$

This is a linear system consisting of 8 equations in the 8 unknown coefficients. Solving the linear system yields

$$S(x)=\begin{cases} 2+\frac{3}{4}(x-1)+\frac{1}{4}(x-1)^3, & 1\le x<2 \\ \\ 3+\frac{3}{2}(x-2)+\frac{3}{4}(x-2)^2-\frac{1}{4}(x-2)^3, & 2\le x\le 3. \end{cases}$$

□

We now describe a general procedure for solving for the unknown coefficients in the polynomials $S_1(x), S_2(x), \ldots, S_n(x)$. First of all, evaluating $S_j$ at $x_{j-1}$ we obtain

$$S_j(x_{j-1}) = a_j$$

and then from condition (b) we must have

$$a_j = f(x_{j-1})$$

for $j = 1, 2, \ldots, n$. Hence, the $a$ coefficients $(a_1, a_2, \ldots, a_n)$ are simply the values of the function at the first $n$ nodes, namely $(f(x_0), f(x_1), \ldots, f(x_{n-1}))$. For notational consistency, we define $a_{n+1} = f(x_n)$. Recall that condition (b) produces $2n$ equations and we have already solved for $n$ of the coefficients (the $a$'s); we can therefore reduce the $2n$ equations from (b) into $n$ equations. Now $S_j(x_j) = f(x_j)$ and also $S_{j+1}(x_j) = f(x_j)$ and thus $S_j(x_j) = S_{j+1}(x_j)$, which is equivalent to

$$a_{j+1} = a_j + b_j(x_j - x_{j-1}) + c_j(x_j - x_{j-1})^2 + d_j(x_j - x_{j-1})^3$$

for $j = 1, 2, \ldots, n-1$. Since the quantity $x_j - x_{j-1}$ is used repeatedly, we set $h_j = x_j - x_{j-1}$, and thus

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \tag{3.7}$$

for $j = 1, 2, \ldots, n-1$. Now, since $f(x_n)$ is known and $f(x_n) = S_n(x_n) = a_n + b_n h_n + c_n h_n^2 + d_n h_n^3$ and we defined $a_{n+1} = f(x_n)$, then (3.7) actually holds for all $j = 1, 2, \ldots, n$. Hence, (3.7) are the $n$ equations produced by condition (b).

Now, similarly from the first order conditions supplied by (c) we have

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2 \tag{3.8}$$

for $j = 1, 2, \ldots, n-1$, and from the second order conditions supplied in (d) we have

$$c_{j+1} = c_j + 3d_j h_j$$

for $j = 1, 2, \ldots, n-1$. We can solve the last equation for $d_j$:

$$d_j = \frac{c_{j+1} - c_j}{3h_j} \tag{3.9}$$

for $j = 1, 2, \ldots, n-1$. Now, from the natural condition $0 = S_n''(x_n) = 2c_n + 6d_n h_n$, and therefore if we set $c_{n+1} = 0$ then

$$d_n = \frac{c_{n+1} - c_n}{3h_n}.$$

Hence, (3.9) holds for $j = 1, 2, \ldots, n$. Thus, $d_1, d_2, \ldots, d_n$ are all in terms of $c_1, c_2, \ldots, c_n$. Now, substituting (3.9) into (3.7) and (3.8) we obtain

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1}) \tag{3.10}$$

for all $j = 1, 2, \ldots, n$, and

$$b_{j+1} = b_j + h_j(c_j + c_{j+1})$$

for $j = 1, 2, \ldots, n-1$, or equivalently

$$b_j = b_{j-1} + h_{j-1}(c_{j-1} + c_j) \tag{3.11}$$

for $j = 2, \ldots, n$. Now, solving for $b_j$ from (3.10) we obtain

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}) \tag{3.12}$$

for $j = 1, 2, \ldots, n$. Therefore, using (3.11) and (3.12) we have

$$\frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1}) = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j) + h_{j-1}(c_{j-1} + c_j)$$

for $j = 2, 3, \ldots, n$, which can be simplified to

$$h_{j-1}c_{j-1} + 2(h_j + h_{j-1})c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1}) \tag{3.13}$$

for $j = 2, 3, \ldots, n$. From (3.13) we obtain $n-1$ equations for the $n$ unknowns $c_1, c_2, \ldots, c_n$. However, recall that the natural condition at $x_0$ is $0 = S_1''(x_0) = 2c_1$ and therefore we have that $c_1 = 0$. Hence the $n-1$ equations (3.13) uniquely determine the coefficients $c_2, c_3, \ldots, c_n$ provided a solution exists. We note also that we defined $c_{n+1} = 0$ for notational consistency. Once we solve the linear system (3.13) for the unknowns $c_1, c_2, \ldots, c_n$ we can obtain the $b$'s and the $d$'s using (3.12) and (3.9), respectively.

**Example 3.16.** Suppose that $n = 5$. Then the $n-1$ equations (3.13) are (recall that $j = 2, 3, \ldots, n$)

$$h_1 c_1 + 2(h_2 + h_1)c_2 + h_2 c_3 = \frac{3}{h_2}(a_3 - a_2) - \frac{3}{h_1}(a_2 - a_1)$$

$$h_2 c_2 + 2(h_3 + h_2)c_3 + h_3 c_4 = \frac{3}{h_3}(a_4 - a_3) - \frac{3}{h_2}(a_3 - a_2)$$

$$h_3 c_3 + 2(h_4 + h_3)c_4 + h_4 c_5 = \frac{3}{h_4}(a_5 - a_4) - \frac{3}{h_3}(a_4 - a_3)$$

$$h_4 c_4 + 2(h_5 + h_4)c_5 + h_5 c_6 = \frac{3}{h_5}(a_6 - a_5) - \frac{3}{h_4}(a_5 - a_4)$$

Recall that we also have that $c_1 = 0$ and $c_6 = 0$.

The linear system (3.13) is an example of a **tridiagonal linear system** whose coefficient matrix is **strictly diagonally dominant**. A matrix $\mathbf{A}$ with entries $a_{i,j}$ is called strictly diagonally dominant if

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}|$$

One can show that strictly diagonally dominant matrices are non-singular and thus a unique solution for $c_2, c_3, \ldots, c_n$ exist. Once the $c_1, c_2, \ldots, c_n$ are known, we can compute $d_1, d_2, \ldots, d_n$ from (3.9) and $b_1, b_2, \ldots, b_n$ from (3.12). In summary, we have the following.

**Theorem 3.13**

Let $f : [a, b] \to \mathbb{R}$ be a given function and let $a = x_0 < x_1 < \cdots < x_n = b$ be distinct nodes. Then $f$ has a unique natural cubic spline interpolant $S$ on $[a, b]$.

The linear system (3.13) for the coefficients $\mathbf{c} = (c_1, c_2, \ldots, c_{n+1})$ written in vector form $\mathbf{Ac} = \mathbf{w}$ has coefficient matrix

$$\mathbf{A} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & 0 & 0 & \cdots & \cdots & 0 \\ \boldsymbol{h_1} & \mathbf{2(h_1 + h_2)} & \boldsymbol{h_2} & 0 & \cdots & \cdots & 0 \\ 0 & \boldsymbol{h_2} & \mathbf{2(h_2 + h_3)} & \boldsymbol{h_3} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \boldsymbol{h_{n-2}} & \mathbf{2(h_{n-2} + h_{n-1})} & \boldsymbol{h_{n-1}} & 0 \\ 0 & \cdots & \cdots & 0 & \boldsymbol{h_{n-1}} & \mathbf{2(h_{n-1} + h_n)} & \boldsymbol{h_n} \\ 0 & 0 & \cdots & \cdots & 0 & \mathbf{0} & \mathbf{1} \end{pmatrix}$$

and

$$\mathbf{w} = \left(0,\ \frac{3}{h_2}(a_3 - a_2) - \frac{3}{h_1}(a_2 - a_1),\ \cdots,\ \frac{3}{h_n}(a_{n+1} - a_n) - \frac{3}{h_{n-1}}(a_n - a_{n-1}),\ 0\right). \quad (3.14)$$

In Algorithm 3.4, we compute the $4n$ coefficients in a natural cubic spline given the data $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$.

We end this section with an example.

**Example 3.17.** In this example we compare cubic spline interpolation with Lagrange polynomial interpolation for the Runge function

$$f(x) = \frac{1}{1 + 25x^2}$$

**Algorithm 3.4** Natural Cubic Spline Coefficients

INPUT: $\mathbf{x} = (x_0, x_1, \ldots, x_n)$, $\mathbf{y} = (y_0, y_1, \ldots, y_n)$

OUTPUT: The coefficients $\mathbf{a} = (a_1, \ldots, a_n, a_{n+1})$, $\mathbf{b} = (b_1, \ldots, b_n)$, $\mathbf{c} = (c_1, \ldots, c_n, c_{n+1})$, $\mathbf{d} = (d_1, \ldots, d_n)$ used to construct the cubic spline interpolating the input data with natural boundary conditions

1: **set** $\mathbf{a} = (y_0, y_1, \ldots, y_n)$
2: **create** $\mathbf{h} = (h_1, h_2, \ldots, h_n)$
3: **create** coefficient matrix $\mathbf{A}$ of size $(n+1) \times (n+1)$
4: **create** vector $\mathbf{w}$ using (3.14)
5: **solve** the linear system $\mathbf{Ac} = \mathbf{w}$ for $\mathbf{c} = (c_1, c_2, \ldots, c_{n+1})$
6: **create** $\mathbf{d} = (d_1, d_2, \ldots, d_n)$ using (3.9)
7: **create** $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ using (3.12)
8: **output** $\mathbf{a}[0:n]$, $\mathbf{b}$, $\mathbf{c}[0:n]$, $\mathbf{d}$

on the interval $[-1, 1, ]$ using both Chebyshev nodes and equally spaced nodes. In Figure 3.9, we use Chebyshev nodes to compare cubic spline interpolation with Lagrange polynomial interpolation. In Figure 3.10, we use equally spaced nodes to compare cubic spline interpolation with Lagrange interpolation. Notice that in both cases, the Lagrange polynomial approximation experiences relatively large deviations within the nodes.

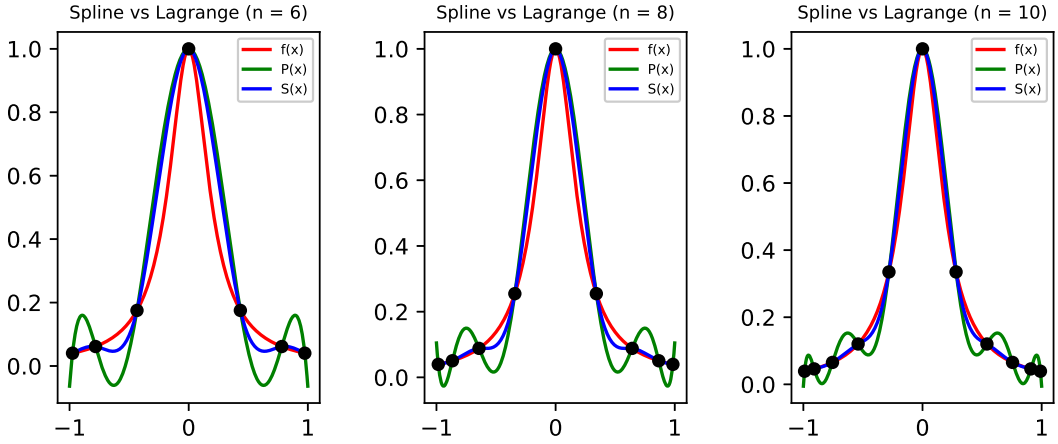



Figure 3.9: Cubic spline vs Lagrange interpolation using Chebyshev nodes: $P(x)$ Lagrange and $S(x)$ spline

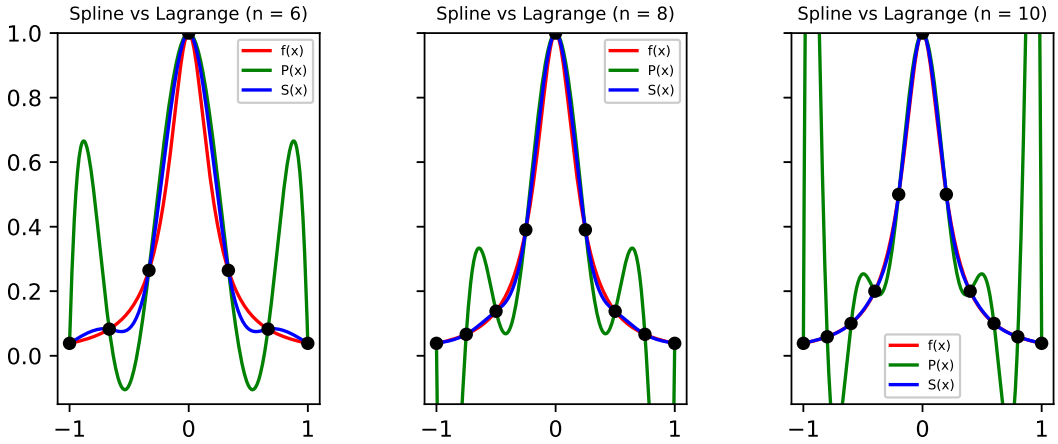



Figure 3.10: Cubic spline vs Lagrange interpolation using equally spaced nodes: $P(x)$ Lagrange and $S(x)$ spline

# 4

# Numerical Differentiation and Integration

## 4.1 Numerical Differentiation

Suppose that $f : [a, b] \to \mathbb{R}$ is differentiable on $[a, b]$. In this section, we consider the problem of approximating $f'(x)$ for a given point $x \in [a, b]$ using only the values of $f$ near $x$. Recall that by definition

$$f'(x) = \lim_{h \to \infty} \frac{f(x+h) - f(x)}{h},$$

and therefore a reasonable way to estimate $f'(x)$ is to use

$$\frac{f(x+h) - f(x)}{h} \tag{4.1}$$

for small $h$. We will see that (4.1) can be obtained by using a linear Lagrange polynomial $P(x)$. We will generalize our derivation of (4.1) to higher-order Lagrange polynomials and obtain more accurate estimates of $f'(x)$ using multiple points near $x$.

Let $x_0 \in [a, b]$ and suppose we want to estimate $f'(x_0)$ using the value of $f$ and $x_0$ and at a nearby point $x_1 = x_0 + h$ be in $[a, b]$. The number $h$ is allowed to be negative in which case $x_1$ is to the left of $x_0$. Then from our previous work on interpolation (see Theorem 3.4), we have

$$f(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + \frac{w(x)}{2!} f^{(2)}(\xi(x))$$

where

$$\begin{aligned} w(x) &= (x - x_0)(x - x_1) \\ L_0(x) &= (x - x_1)/(x_0 - x_1) \\ L_1(x) &= (x - x_0)/(x_1 - x_0), \end{aligned}$$

and $\xi(x)$ is in between $x_0$ and $x_1$. Differentiating with respect to $x$ yields

$$f'(x) = f(x_0)L_0'(x) + f(x_1)L_1'(x) + \frac{(2x - x_0 - x_1)}{2} f^{(2)}(\xi(x)) + \frac{w(x)}{2} \frac{d}{dx} \left( f^{(2)}(\xi(x)) \right).$$

For arbitrary $x \in [a,b]$, the quantity $\frac{d}{dx}\left(f^{(2)}(\xi(x))\right)$ is unknown. However, since $w(x_0) = 0$ we have

$$\begin{aligned} f'(x_0) &= f(x_0)L_0'(x_0) + f(x_1)L_1'(x_0) - \frac{h}{2}f^{(2)}(\xi(x_0)) \\ &= f(x_0)\frac{1}{-h} + f(x_1)\frac{1}{h} - \frac{h}{2}f^{(2)}(\xi(x_0)) \\ &= \frac{1}{h}(f(x_0+h) - f(x_0)) - \frac{h}{2}f^{(2)}(\xi(x_0)). \end{aligned}$$

Hence, for small $h$, we can use $\frac{1}{h}(f(x_0+h) - f(x_0))$ to estimate $f'(x_0)$ with error term bounded by $\frac{M}{2}h$, where $M > 0$ satisfies $|f'(x)| \leq M$ for all $x \in [x_0, x_1]$. The estimate (4.1) is known as a **forward-difference** if $h > 0$ and a **backward-difference** if $h < 0$.

The above derivation can be generalized using an $n$th order Lagrange polynomial with nodes $x_0, x_1, \ldots, x_n$. Suppose then that $f \in C^{n+1}([a,b])$ and $x_0, x_1, \ldots, x_n \in [a,b]$. Then

$$f(x) = P_n(x) + \frac{1}{(n+1)!}w(x)f^{(n+1)}(\xi(x))$$

where

$$P_n(x) = \sum_{k=0}^{n} f(x_k)L_k(x) \quad \text{(interpolating polynomial)}$$

$$w(x) = (x-x_0)(x-x_1)\cdots(x-x_n),$$

$$L_k(x) = \prod_{\substack{i=1 \\ i\neq k}}^{n} \frac{(x-x_i)}{(x_k-x_i)}, \quad k = 0, 1, \ldots, n$$

and $\xi(x)$ is in the interval $[a,b]$. Differentiating with respect to $x$ yields

$$f'(x) = P_n'(x) + \frac{1}{(n+1)!}w'(x)f^{(n+1)}(\xi(x)) + \frac{1}{(n+1)!}w(x)\frac{d}{dx}f^{(n+1)}(\xi(x)).$$

The term $\frac{d}{dx}f^{(n+1)}(\xi(x))$ is unknown and may be difficult to estimate. However, when we evaluate at one of the nodes $x_j$ the term $w(x)$ vanishes, that is, $w(x_j) = 0$, and thus the unknown term $\frac{d}{dx}f^{(n+1)}(\xi(x))$ is eliminated. Therefore,

$$f'(x_j) = P_n'(x_j) + \frac{1}{(n+1)!}w'(x_j)f^{(n+1)}(\xi(x_j))$$

For convenience we introduce the following notation for the error term:

$$e_n(x_j) = \frac{1}{(n+1)!}w'(x_j)f^{(n+1)}(\xi(x_j))$$

and thus

$$f'(x_j) = P_n'(x_j) + e_n(x_j).$$

If $w'(x_j)$ is small and $f^{(n+1)}(\xi(x_j))$ is not too big then the error term $e_n(x_j)$ will be negligible and thus $P_n'(x_j) = \sum_{k=0}^{n} f(x_k)L_k'(x_j)$ can be taken as an estimate for $f'(x_j)$. We therefore obtain the following approximation

$$f'(x_j) = \sum_{k=0}^{n} f(x_k)L_k'(x_j) + e_n(x_j) \tag{4.2}$$

which is called a $(n+1)$**-point formula**.

The most commonly used point formulas are 3- and 5-point formulas. For the 3-point formula we have

$$f'(x_j) = f(x_0)L_0'(x_j) + f(x_1)L_1'(x_j) + f(x_2)L_2'(x_j) + e_2(x_j)$$

for $j = 0, 1, 2$. One can easily show that

$$L_0'(x) = \frac{2x - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}$$

$$L_1'(x) = \frac{2x - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}$$

$$L_2'(x) = \frac{2x - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}$$

Therefore,

$$\begin{aligned} f'(x_j) = f(x_0)\left(\frac{2x_j - x_1 - x_2}{(x_0 - x_1)(x_0 - x_2)}\right) + f(x_1)\left(\frac{2x_j - x_0 - x_2}{(x_1 - x_0)(x_1 - x_2)}\right) \\ + f(x_2)\left(\frac{2x_j - x_0 - x_1}{(x_2 - x_0)(x_2 - x_1)}\right) + e_2(x_j) \end{aligned} \tag{4.3}$$

The 3-point formula (4.3) takes on a very simple form for equally spaced nodes $x_0, x_1, x_2$ where $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$. Here $h$ is called the **step-size** and is allowed to be negative. If $h > 0$ then $x_1$ and $x_2$ are to the right of $x_0$, and if $h < 0$ then $x_1$ and $x_2$ are to the left of $x_0$. In any case, substituting $x_1$ and $x_2$ in the general 3-point formula (4.3) and evaluating at $x_j = x_0$ we obtain

$$f'(x_0) = \frac{1}{2h}\left[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h)\right] + \frac{h^2}{3}f^{(3)}(\xi_0) \tag{4.4}$$

Hence, $f'(x_0)$ can be estimated using the values $f(x_0), f(x_0 + h)$, and $f(x_0 + 2h)$. Hence, if $h > 0$ then only information from $f$ at $x_0$ and in a forward direction is needed, and if $h < 0$

then only information from $f$ at $x_0$ and in a backward direction is needed. Using (4.3) again but with $x_j = x_1$ yield

$$f'(x_1) = \frac{1}{2h}\left(f(x_0 + 2h) - f(x_0)\right) - \frac{h^2}{6}f^{(3)}(\xi_1) \tag{4.5}$$

which can be written as

$$f'(x_1) = \frac{1}{2h}\left(f(x_1 + h) - f(x_1 - h)\right) - \frac{h^2}{6}f^{(3)}(\xi_1).$$

In this case, $f'(x_1)$ can be estimated using values of $f$ after $f(x_1)$, namely $f(x_1 + h)$, and using values of $f$ before $f(x_1)$, namely $f(x_1 - h)$. Lastly, using (4.3) with $x_j = x_2$ we have

$$f'(x_2) = \frac{1}{2h}\left[f(x_0) - 4f(x_0 + h) + 3f(x_0 + 2h)\right] + \frac{h^2}{3}f^{(3)}(\xi_2) \tag{4.6}$$

which can be written as

$$f'(x_2) = \frac{1}{2h}\left[f(x_2 - 2h) - 4f(x_2 - h) + 3f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_2)$$

In this case, $f'(x_2)$ can be estimated using values before $f(x_2)$, namely $f(x_2 - 2h)$ and $f(x_2 - h)$ provided $h > 0$. Notice that the formula for $f'(x_2)$ can be obtained from $f'(x_0)$ by replacing $h$ with $-h$ in $f'(x_0)$.

To eliminate any possible confusion as to whether $h$ is taken to be positive or negative, we summarize the previous derivations assuming that $h > 0$. Then, to estimate $f'$ at the points $x_0 < x_1 < x_2$, with $x_1 = x_0 + h$ and $x_2 = x_0 + 2h$, we we have

$$f'(x_0) \approx \frac{1}{2h}\big(-3f(x_0) + 4f(x_1) - f(x_2)\big) \tag{F}$$

$$f'(x_1) \approx \frac{1}{2h}\big(f(x_2) - f(x_0)\big) \tag{M}$$

$$f'(x_2) \approx \frac{1}{2h}\big(f(x_0) - 4f(x_1) + 3f(x_2)\big) \tag{B}$$

The estimate (F) is a forward 3-point estimate, (M) is a mid-point 3-point estimate, and (B) is a backward 3-point estimate. Notice that the error with the mid-point estimate is approximately half of the error of the two end-point formulas (F) and (B). This seems plausible since the mid-point estimate uses the values of $f$ on the left and right from where $f'$ is being estimated, whereas the end-point formulas uses the values of $f$ only on one side (forward or backward) from where $f'$ is being estimated.

Using a similar analysis, we can derive the 5**-point midpoint formula**

$$f'(x_0) = \frac{1}{12h}\left[f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)\right] + \frac{h^4}{30}f^{(5)}(\xi)$$

where $\xi \in (x_0 - 2h, x_0 + 2h)$, and the **5-point endpoint formula**

$$f'(x_0) = \frac{1}{12h}\left[-25f(x_0) + 48f(x_0 + h) - 36f(x_0 + 2h) + 16f(x_0 + 3h) - 3f(x_0 + 4h)\right] + \frac{h^4}{5}f^{(5)}(\xi)$$

where $\xi \in (x_0, x_0 + 4h)$. We now consider two examples.

**Example 4.1.** Consider the data

| $x$ | $f(x)$ |
|---|---|
| 1.0 | $-1.0$ |
| 1.1 | $-2.0$ |
| 1.2 | 1.0 |
| 1.3 | 1.0 |
| 1.4 | 3.0 |

Use the given values to estimate $f'(x)$ for each $x \in \{1.0, 1.1, 1.2, 1.3, 1.4\}$ using all applicable 3-point formulas.

*Solution.* The step-size is $h = 0.1$. We consider each $x$ separately:

1. $x = 1.0$: In this case, only the end-point formulas can be applied. The 3-point formula gives
$$f'(1.0) \approx \frac{1}{2h}(-3f(1) + 4f(1.1) - f(1.2)) = -30.0$$

2. $x = 1.1$: We can use 3-point end-point (forward) and 3-point midpoint. For end-point
$$f'(1.1) \approx \frac{1}{2h}(-3f(1.1) + 4f(1.2) - f(1.3)) = 4.5$$
and for mid-point
$$f'(1.1) \approx \frac{1}{2h}(f(1.2) - f(1.0)) = 10.0$$

3. $x = 1.2$: We can use all three 3-point formulas (foward/backward end-point, and midpoint). For forward end-point:
$$f'(1.2) \approx \frac{1}{2h}(-3f(1.2) + 4f(1.3) - f(1.4)) = -10.0$$
and for backward end-point:
$$f'(1.2) \approx \frac{1}{2h}(f(1) - 4f(1.1) + 3f(1.2)) = 50$$
With midpoint:
$$f'(1.2) \approx \frac{1}{2h}(f(1.3) - f(1.1)) = 15$$

4. $x = 1.3$: We can use mid-point and backward end-point formula. For mid-point:

$$f'(1.3) \approx \frac{1}{2h}(f(1.4) - f(1.2)) = 10.0$$

and with the backward end-point formula:

$$f'(1.3) \approx \frac{1}{2h}(f(1.1) - 4f(1.2) + 3f(1.3)) = -15.0$$

5. $x = 1.4$: Only the backward end-point 3-point. For the 3-point formula:

$$f'(1.4) \approx \frac{1}{2h}(f(1.2) - 4f(1.3) + 3f(1.4)) = 30$$

□

**Example 4.2.** Consider again the data

| $x$ | $f(x)$ |
|---|---|
| 1.0 | −1.0 |
| 1.1 | −2.0 |
| 1.2 | 1.0 |
| 1.3 | 1.0 |
| 1.4 | 3.0 |

Use the given values to estimate $f'(x)$ for each $x \in \{1.0, 1.1, 1.2, 1.3, 1.4\}$ using all applicable 5-point formulas.

*Solution.* Again we have $h = 0.1$. In this case, we can only compute $f'(x)$ using 5-point formulas for $x = 1.0$, $x = 1.2$, and $x = 1.4$. We consider each $x$ separately:

1. $x = 1.0$: The 5-point forward formula gives

$$f'(1.0) \approx \frac{1}{12h}(-25f(1) + 48f(1.1) - 36f(1.2) + 16f(1.3) - 3f(1.4)) = \cdots$$

2. $x = 1.2$: The 5-point mid-point formula gives

$$f'(1.2) \approx \frac{1}{12h}(f(1.0) - 8f(1.1) + 8f(1.3) - f(1.4)) = \cdots$$

3. $x = 1.4$: The 5-point backward formula:

$$f'(1.4) \approx -\frac{1}{12h}(-3f(1) + 16f(1.1) - 36f(1.2) + 48f(1.3) - 25f(1.4)) = \cdots$$

□

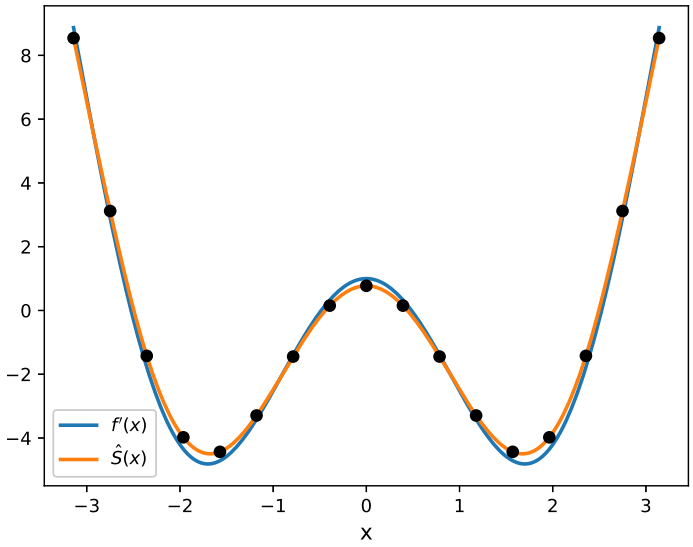



Figure 4.1: Figure for Example 4.3; $\hat{S}(x)$ interpolates through the estimates of $f'(x_j)$

**Example 4.3.** Suppose that the function $f(x) = x\cos(x) - x^2\sin(x)$ is sampled at equally spaced points $-\pi = x_0 < x_1 < \cdots < x_{n-1} < x_n = \pi$, where $n = 16$. Using the 3-point formulas to estimate $f'(x_j)$ for $j = 0, 1, \ldots, n$ we obtain the values $\hat{y}' = (\hat{y}'_0, \hat{y}'_1, \ldots, \hat{y}'_n)$. We then use a natural cubic spline $\hat{S}(x)$ to interpolate the data $\{(x_j, \hat{y}'_j\}_{j=0}^n$. The graph of $f'(x)$ and $\hat{S}(x)$ are shown in Figure 4.1.

**Example 4.4.** Write a Python function called `ThreePointDiff` that takes as input a step-size $h$ and an array $y = (y_0, y_1, \ldots, y_n)$ and returns the numerical derivative of the $y$-values using the 3-point formulas. Use the mid-point estimates whenever they are applicable. Algorithm 4.1 shows the pseudocode for `ThreePointDiff`.

**Algorithm 4.1** ThreePointDiff

INPUT: $y = (y_0, y_1, \ldots, y_n)$, step-size $h > 0$

OUTPUT: $\hat{y}' = (\hat{y}'_0, \hat{y}'_1, \ldots, \hat{y}'_n)$ where $\hat{y}'_j$ is a 3-point estimate of $f'(x_j)$

1: **set** $\hat{y}' = \text{zeros}(n+1)$

2: **compute** $\hat{y}'_0$ # using forward 3-point formula

3: **compute** $\hat{y}'_n$ # using backward 3-point formula

4: **for** $k = 1, 2, \ldots, n-1$ **do**

5: **compute** $\hat{y}'_k$ # using mid-point 3-point formula

6: **output** $\hat{y}'$

## 4.2 Numerical Integration

Before we begin with numerical integration, we introduce the Weighted Mean Value Theorem (WMVT) for integrals that will be used in this section.

**Theorem 4.1: Weighted Mean Value Theorem for Integrals**

Suppose that $f$ and $g$ are continuous on $[a, b]$ and $g$ does not change sign on $[a, b]$. Then there exists a number $c \in (a, b)$ such that

$$\int_a^b f(x)g(x)\,dx = f(c)\int_a^b g(x)\,dx.$$

We note that when $g(x) = 1$ for all $x \in [a, b]$ then the WMVT reduces to the standard Mean Value theorem for Integrals:

$$\int_a^b f(x)\,dx = f(c)(b-a).$$

In this section, we consider the problem of numerically computing definite integrals

$$\int_a^b f(x)\,dx.$$

The process of computing exact areas is termed **quadrature**, and the process of using numerical methods to compute definite integrals is called **numerical quadrature**. In its simplest form, the approach is to partition the interval $[a, b]$ using nodes $a = x_0 < x_1 < \cdots < x_n = b$, compute the $n$th order Lagrange interpolating polynomial

$$P(x) = \sum_{i=0}^{n} f(x_i)L_i(x)$$

and then use

$$\begin{aligned}\int_a^b f(x)\,dx \approx \int_a^b P(x)\,dx &= \int_a^b \sum_{i=0}^{n} f(x_i)L_i(x)\,dx \\ &= \sum_{i=0}^{n} f(x_i)\int_a^b L_i(x)\,dx \\ &= \sum_{i=0}^{n} f(x_i)s_i\end{aligned}$$

where

$$s_i = \int_a^b L_i(x)\,dx$$

for $i = 0, 1, \ldots, n$. Hence, the estimates we will obtain to $\int_a^b f(x)\,dx$ will be of the form

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

As we have seen, however, using a high-order interpolating polynomial over an entire interval $[a, b]$ can be problematic due to the oscillatory behavior of high-order polynomials. Hence, we may want to take a piecewise approach to numerical integration and use many low-order polynomials instead of one high-order polynomial, similar to what we did with cubic splines and numerical differentiation. In other words, we will partition $[a, b]$ into smaller subintervals $[a_1, b_1], [a_2, b_2], \ldots, [a_r, b_r]$, and for each subinterval $[a_j, b_j]$ use an $n$th order Lagrange polynomial to estimate $\int_{a_j}^{b_j} f(x)\,dx$, where $n$ is small, and then add the estimates to approximate $\int_a^b f(x)\,dx$.

We now described two methods to estimate $\int_a^b f(x)\,dx$. The simplest and non-trivial estimate of $\int_a^b f(x)\,dx$ is obtained by using a *linear* Lagrange polynomial with nodes $x_0 = a$ and $x_1 = b$. The linear Lagrange polynomial is

$$P(x) = f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0}.$$

Then $f(x) = P(x) + \frac{1}{2}f^{(2)}(\xi(x))(x - x_0)(x - x_1)$ (where $\xi(x)$ is in between $x_0$ and $x_1$), and therefore

$$\begin{aligned}
\int_{x_0}^{x_1} f(x)\,dx &= \int_{x_0}^{x_1} P(x)\,dx + \frac{1}{2}\int_{x_0}^{x_1} f^{(2)}(\xi(x))(x - x_0)(x - x_1)\,dx \\
&= \left[\frac{f(x_0)}{x_0 - x_1}(x - x_1)^2 + \frac{f(x_1)}{x_1 - x_0}(x - x_0)^2\right]_{x_0}^{x_1} + \frac{1}{2}\int_{x_0}^{x_1} f^{(2)}(\xi(x))(x - x_0)(x - x_1)\,dx \\
&= \frac{(x_1 - x_0)}{2}[f(x_0) + f(x_1)] + \frac{1}{2}\int_{x_0}^{x_1} f^{(2)}(\xi(x))(x - x_0)(x - x_1)\,dx
\end{aligned}$$

Since $g(x) = (x - x_0)(x - x_1)$ does not change sign in $[x_0, x_1]$, the WMVT applies and after some simplification we obtain

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{(x_1 - x_0)}{2}[f(x_0) + f(x_1)] - f^{(2)}(c)\frac{(x_1 - x_0)^3}{12}$$

where $c$ is a number in $(x_0, x_1)$. Setting $h = x_1 - x_0$ we obtain

$$\int_{x_0}^{x_1} f(x)\,dx = \frac{h}{2}\left[f(x_0) + f(x_1)\right] - \frac{h^3}{12}f^{(2)}(c) \tag{4.7}$$

and this is commonly called the **Trapezoidal rule** since when $f$ is non-negative on $[a, b]$ the number $\frac{h}{2}(f(x_0) + f(x_1))$ is the area of a trapezoid with heights $f(x_0)$ and $f(x_1)$ and of

base $h$. Equivalently, $\frac{h}{2}(f(x_0)+f(x_1))$ is the average of the left-end and right-end Riemann sum approximations.

The next natural estimate of $\int_a^b f(x)\,dx$ is to use a quadratic Lagrange polynomial with nodes $a=x_0<x_1<x_2=b$. However, the estimate of the error will be of order $O(h^4)$ and it turns out that if we use a cubic Taylor polynomial based at $x_1$ then the error estimate is of order $O(h^5)$, where $h=x_1-x_0=x_2-x_1$. Hence, using a cubic Taylor polynomial based at $x_1$ we obtain

$$f(x)=\sum_{k=0}^{3}\frac{1}{k!}f^{(k)}(x_1)(x-x_1)^k+\frac{1}{4!}f^{(4)}(\xi)(x-x_1)^4$$

where $\xi$ is in between $x$ and $x_1$, and therefore

$$\int_a^b f(x)\,dx=\int_a^b\sum_{k=0}^{3}\frac{1}{k!}f^{(k)}(x_1)(x-x_1)^k\,dx+\frac{1}{4!}\int_a^b f^{(4)}(\xi(x))(x-x_1)^4\,dx.$$

Applying the WMVT to the second integral we obtain

$$\int_a^b f(x)\,dx=\int_a^b\sum_{k=0}^{3}\frac{1}{k!}f^{(k)}(x_1)(x-x_1)^k\,dx+\frac{1}{60}f^{(4)}(\xi)h^5 \tag{4.8}$$

Now, recalling that $a=x_0$ and $b=x_2$, we obtain

$$\begin{aligned}\int_a^b\sum_{k=0}^{3}\frac{1}{k!}f^{(k)}(x_1)(x-x_1)^k\,dx&=\sum_{k=0}^{3}\frac{f^{(k)}(x_1)}{(k+1)!}(x-x_1)^{k+1}\Big|_{x_0}^{x_2}\\&=f(x_1)2h+\frac{1}{2!}f^{(1)}(x_1)(h^2-(-h)^2)+\frac{1}{3!}f^{(2)}(x_1)(h^3-(-h)^3)\\&\quad+\frac{1}{4!}f^{(3)}(x_1)(h^4-(-h)^4)\\&=f(x_1)2h+f^{(2)}(x_1)\frac{h^3}{3}\end{aligned}$$

Using techniques from numerical differentiation, it can be shown that

$$f^{(2)}(x_1)=\frac{1}{h^2}[f(x_0)-2f(x_1)+f(x_2)]-\frac{h^2}{12}f^{(4)}(\xi_1)$$

where $\xi_1\in(x_0,x_2)$. Therefore,

$$\begin{aligned}\int_a^b\sum_{k=0}^{3}\frac{1}{k!}f^{(k)}(x_1)(x-x_1)^k\,dx&=f(x_1)2h+f^{(2)}(x_1)\frac{h^3}{3}\\&=2hf(x_1)+\frac{h^3}{3}\left[\frac{1}{h^2}\left(f(x_0)-2f(x_1)+f(x_2)\right)-\frac{h^2}{12}f^{(4)}(\xi_1)\right]\\&=\frac{h}{3}[f(x_0)+4f(x_1)+f(x_2)]-\frac{h^5}{36}f^{(4)}(\xi_1)\end{aligned}$$

Combining the last equation with (4.8) we finally obtain

$$\int_a^b f(x)\,dx = \frac{h}{3}[f(x_0) + 4f(x_1) + f(x_2)] - \frac{h^5}{90}f^{(4)}(\xi) \tag{4.9}$$

where $\xi$ lies in $(a, b)$. The above estimate is known as **Simpson's rule**.

Both the trapezoidal and Simpson's rule can be used in a piecewise approach to numerical integration. We demonstrate this via an example using Simpson's rule.

**Example 4.5.** Approximate $\int_0^4 e^x\,dx$ using Simpson's rule in three different ways:

(a) Apply Simpson's rule on the interval $[0, 4]$.

(b) Partition $[0, 4]$ into $r = 2$ subintervals $[0, 2]$ and $[2, 4]$, apply Simpson's rule on each subinterval and add the two results.

(c) Partition $[0, 4]$ into $r = 4$ subintervals $[0, 1], [1, 2], [2, 3], [3, 4]$, apply Simpson's rule on each subinterval and add the four results.

Compare all three with the exact value $\int_0^4 e^x\,dx = e^4 - 1 = 53.598150033144236\ldots$

*Solution.* (a) In this case $h = 2$ and we obtain the estimate

$$\int_0^4 e^x\,dx \approx \frac{h}{3}\left(e^0 + 4e^2 + e^4\right) = \frac{2}{3}(1 + 4e^2 + e^4) = 56.76958$$

The magnitude of the error is 3.17143.

(b) In this case $h = 1$ and we obtain the estimate

$$\begin{aligned}\int_0^4 e^x\,dx &= \int_0^2 e^x\,dx + \int_2^4 e^x\,dx \\ &\approx \frac{h}{3}\left(e^0 + 4e^1 + e^2\right) + \frac{h}{3}(e^2 + 4e^3 + e^4) \\ &= 53.86385\end{aligned}$$

The magnitude of the error is 0.26570.

(c) In this case $h = \frac{1}{2}$ and we obtain the estimate

$$\begin{aligned}\int_0^4 e^x\,dx &= \int_0^1 e^x\,dx + \int_1^2 e^x\,dx + + \int_2^3 e^x\,dx + \int_3^4 e^x\,dx \\ &\approx \frac{h}{3}(e^0 + 4e^{0.5} + e^1) + \frac{h}{3}(e^1 + 4e^{1.5} + e^2) \\ &\quad + \frac{h}{3}(e^2 + 4e^{2.5} + e^3) + \frac{h}{3}(e^3 + 4e^{3.5} + e^4) \\ &= 53.61622\end{aligned}$$

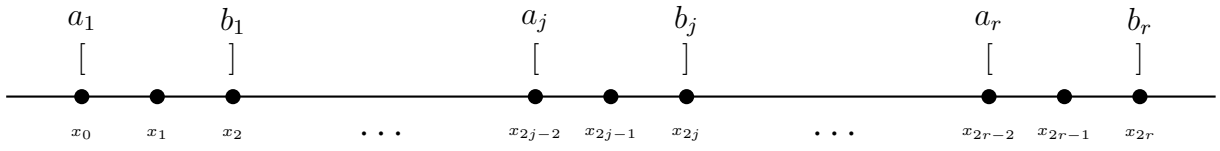
The magnitude of the error is 0.01807.

□

The previous example illustrates the general approach to piecewise numerical integration. To approximate $\int_a^b f(x)\,dx$, we partition $[a,b]$ into $r$ non-overlapping and equal length subintervals $[a_1,b_1],[a_2,b_2],\ldots,[a_r,b_r]$, and estimate $\int_{a_j}^{b_j} f(x)\,dx$ using either the Trapezoidal or Simpson's rule (or any other estimate). If, for instance, we used Simpson's rule, for each sub-interval $[a_j,b_j]$ we use three equally-spaced nodes $x_{2j-2} < x_{2j-1} < x_{2j}$, where $x_{2j-2} = a_j$, $x_{2j} = b_j$, and $x_{2j-1}$ is the midpoint between $a_j$ and $b_j$, and use the estimate

$$\int_{a_j}^{b_j} f(x)\,dx \approx \frac{h}{3}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})).$$

Below is a diagram illustrating the partitioning of $[a,b]$ into subintervals $[a_j,b_j]$ and the resulting nodes $x_0, x_1, x_2, \ldots, x_{2r}$.




We then add up all the estimates on each subinterval and obtain the following estimate for $\int_a^b f(x)\,dx$:

$$\int_a^b f(x)\,dx \approx \sum_{j=1}^{r} \frac{h}{3}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})).$$

The estimate (4.10) is known as the **Composite Simpson's rule** on $[a,b]$ with $r$ subintervals. The process of partitioning $[a,b]$ into the $r$ subintervals $[a_j,b_j]$ produces the nodes $x_0, x_1, \ldots, x_n$ with spacing $h = (b-a)/n$, where we have set $n = 2r$.

In practice, given $[a,b]$ one introduces equally-spaced nodes $a = x_0 < x_1 < \cdots < x_n = b$ where $n$ is **even** and $h = \frac{b-a}{n}$ is the step-size of the nodes. The values $y_j = f(x_j)$ are computed, for $j = 0, 1, \ldots, n$, and on each subinterval $[x_{2j-1}, x_{2j}]$ we obtain the estimate

$$\int_{x_{2j-2}}^{x_{2j}} f(x)\,dx \approx \frac{h}{3}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})).$$

and add these estimates

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\sum_{j=1}^{n/2}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})). \tag{4.10}$$

Based on our experience with working with Lagrange polynomials, we would expect that this piecewise approach would yield better approximations than using a single Lagrange polynomial with $n+1$ equally spaced nodes.

**Example 4.6.** Consider the function

$$f(x) = 3\cos(2x) - \sin(0.5x) + 3\sin(3.3x) + 0.5\sin(10x)$$

on the interval $[a, b] = [-2, 4]$. Let $n = 16$ and let $h = (b - a)/n = 0.375$. We obtain the nodes $x_0, x_1, \ldots, x_{16}$ with $x_j = a + jh$, $j = 0, 1, \ldots, n$. The first three nodes $x_0, x_1, x_2$ are used to compute the estimate

$$\int_{x_0}^{x_2} f(x)\,dx \approx \frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2)) = -1.2188.$$

We then use $x_2, x_3, x_4$ to compute the estimate

$$\int_{x_2}^{x_4} f(x)\,dx \approx \frac{h}{3}(f(x_2) + 4f(x_3) + f(x_4)) = -0.6562.$$

We continue in this way until we finally compute

$$\int_{x_{14}}^{x_{16}} f(x)\,dx \approx \frac{h}{3}(f(x_{14}) + 4f(x_{15}) + f(x_{16})) = 2.7188.$$

Then

$$\int_{-2}^{4} f(x)\,dx \approx \sum_{j=1}^{n/2} \frac{h}{3}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})) = -1.5688.$$

Repeating this process with increasing values of $n$ produces the table of values

| $n$ | $S$ |
|---|---|
| 16 | −1.568761 |
| 32 | −1.372813 |
| 64 | −1.378821 |
| 128 | −1.379072 |
| 256 | −1.379087 |
| 512 | −1.379088 |

The actual value is $\int_{-2}^{4} f(x)\,dx = -1.379087621\ldots$.

We now do some further analysis of using the Composite Simpson's rule. From (4.9), on each interval $[x_{2j-2}, x_{2j}]$ we have

$$\int_{x_{2j-2}}^{x_{2j}} f(x)dx = \frac{h}{3}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})) - \frac{h^5}{90}f^{(4)}(\xi_j)$$

where $\xi_j \in (x_{2j-2}, x_{2j-1})$. Therefore,

$$\begin{aligned}\int_a^b f(x)\,dx &= \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)dx \\ &= \sum_{j=1}^{n/2} \left[\frac{h}{3}(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})) - \frac{h^5}{90}f^{(4)}(\xi_j)\right]\end{aligned}$$

Now, the even values $f(x_{2j})$ for $j = 1, 2, \ldots, n-1$ appear twice in the above sum, and therefore the Composite Simpson's rule can be written as

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right] - \frac{h^5}{90}\sum_{j=1}^{n/2} f^{(4)}(\xi_j) \quad (4.11)$$

Further analysis of the error terms produces the following result.

**Theorem 4.2: Composite Simpson's Rule**

Suppose that $f \in C^4[a,b]$ and let $n$ be even. Let $h = \frac{b-a}{n}$ and let $x_j = a + jh$ for $j = 0, 1, \ldots, n$. There exists $\xi \in (a,b)$ such that

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right] - \frac{(b-a)}{180}h^4 f^{(4)}(\xi).$$

For notational convenience, we denote

$$S(f,a,b,n) = \frac{h}{3}\left[f(x_0) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + f(x_n)\right] \quad (4.12)$$

and call it the **Simpson approximation** of $\int_a^b f(x)\,dx$. Thus, with this notation, from the Composite Simpson's Rule Theorem we obtain

$$\left|\int_a^b f(x)\,dx - S(f,a,b,n)\right| = \frac{(b-a)}{180}h^4 \max_{x\in[a,b]} |f^{(4)}(x)|.$$

Using (4.12), it is straightforward to write the pseudocode for the Composite Simpson rule, see Algorithm 4.2.

**Example 4.7.** Let $f(x) = \sin(x)$. Determine the value of $n$ that will guarantee that $\left|\int_0^\pi \sin(x)\,dx - S(f,0,\pi,n)\right| < \varepsilon$ where $\varepsilon = 1 \times 10^{-12}$.

*Solution.* According to the Composite Simpson's Rule theorem (and using the fact that $f^{(4)}(x) = \sin(x)$) we have

$$\left|\int_0^\pi \sin(x)\,dx - S(f,0,\pi,n)\right| = \frac{\pi}{180}h^4|\sin(\xi)|$$

where $\xi \in (0,\pi)$. Since $|\sin(x)| \le 1$ for all $x$ we can write

$$\left|\int_0^\pi \sin(x)\,dx - S(f,0,\pi,n)\right| = \frac{\pi}{180}h^4|\sin(\xi)| \le \frac{\pi}{180}h^4$$

**Algorithm 4.2** Composite Simpson Rule

INPUT: $h > 0$, data points $y_0, y_1, y_2, \ldots, y_n$, where $n$ is even

OUTPUT: Estimate $\sum_{j=1}^{n/2} S_j$ where $S_j = \frac{h}{3}(y_{2j-2} + 4y_{2j-1} + y_{2j})$

1: **set** $S_{\text{even}} = 0$
2: **set** $S_{\text{odd}} = 0$
3: **for** $j = 1, 2, \ldots, (\frac{n}{2} - 1)$ **do**
4: **set** $S_{\text{even}} = S_{\text{even}} + y_{2j}$
5: **for** $j = 1, 2, \ldots, \frac{n}{2}$ **do**
6: **set** $S_{\text{odd}} = S_{\text{odd}} + y_{2j-1}$
7: **set** $S = \frac{h}{3}(y_0 + 2S_{\text{even}} + 4S_{\text{odd}} + y_n)$
8: **output** $S$

If we set $\frac{\pi}{180}h^4 < \varepsilon$ and use $h = \frac{\pi}{n}$ we obtain that

$$h < \left(\frac{180\varepsilon}{\pi}\right)^{1/4}$$

which is equivalent to

$$\frac{\pi}{\left(\frac{180\varepsilon}{\pi}\right)^{1/4}} < n$$

With the given $\varepsilon = 1 \times 10^{-12}$ we obtain

$$n > 1141.876.$$

We must therefore choose $n = 1142$ and with this choice one computes that

$$S(f, 0, \pi, n) = 2.0000000000006364 = 2 + 6.364 \times 10^{-13}$$

The actual value of the integral is $\int_0^\pi \sin(x)\,dx = 2$ and we are within $\varepsilon$ as predicted. □

# 5

# Direct Methods for Linear Systems

## 5.1 Gaussian Elimination with Partial Pivoting

To solve the linear system $\mathbf{Ax} = \mathbf{b}$ for the unknown vector $\mathbf{x}$, we form the augmented matrix $\tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{A} & \mathbf{b} \end{pmatrix}$ and perform elementary row operations, also known as Gaussian elimination.

## 5.2 LU Decomposition

We begin, as usual, with definitions.

**Definition 5.1: Triangular Matrices**

The matrix $\mathbf{M} = (m_{i,j})$ is *lower-triangular* if all entries above the main diagonal of $\mathbf{M}$ are zero, that is, $m_{i,j} = 0$ whenever $j > i$. We say that $\mathbf{M}$ is *upper-triangular* if all entries below the main diagonal of $\mathbf{M}$ are zero, that is, $m_{i,j} = 0$ whenever $i > j$. If $\mathbf{M}$ is either upper- or lower-triangular then we say that $\mathbf{M}$ is *triangular*. If $\mathbf{M}$ is both upper- and lower-triangular then $\mathbf{M}$ is called a *diagonal* matrix.

**Example 5.1.** Prove that if $\mathbf{M}$ is a triangular matrix then $\det(\mathbf{M}) = \prod_{i=1}^{n} m_{i,i}$.

**Example 5.2.** If $\mathbf{M}$ is lower-triangular then what can we say about $\mathbf{M}^T$?

We now introduce the main notion of this section.

**Definition 5.2: *LU*-Decomposition**

The matrix $\mathbf{A}$ has a *LU-decomposition* or *LU-factorization* if there exists a lower-triangular matrix $\mathbf{L}$ and an upper-triangular matrix $\mathbf{U}$ such that $\mathbf{A} = \mathbf{LU}$.

The Gaussian elimination algorithm used to solve the linear system $\mathbf{Ax} = \mathbf{b}$ may seem like a very elementary process used for a specific problem (solving a linear system) but in this section we will see that the algorithm is more sophisticated than at first glance and useful to

**Algorithm 5.1** Gaussian Elimination with Partial Pivoting

INPUT: $\mathbf{A}$ an $n \times n$ matrix, $\mathbf{b}$ a $n \times 1$ vector

OUTPUT: Solution $\mathbf{x}$ to the linear system $\mathbf{Ax} = \mathbf{b}$, or error message that no unique solution exists

**set** $\varepsilon = 1 \times 10^{-14}$ (or a suitably small tolerance)

**set** $\tilde{\mathbf{A}} = \begin{pmatrix} \mathbf{A} & \mathbf{b} \end{pmatrix}$

**for** $i = 1, 2, \ldots, n$ **do** (main loop performing row reduction)

  **find** $k \in \{i, i+1, \ldots, n\}$ such that $|\tilde{a}_{k,i}| = \max\{|\tilde{a}_{i,i}|, |\tilde{a}_{i+1,i}|, \ldots, |a_{n,i}|\}$

  **if** $|a_{k,i}| < \varepsilon$

    **print**( "No unique solution exists!" )

    **return** 0

  **if** $k \neq i$

    **swap** row $R_i$ and row $R_k$ in $\tilde{\mathbf{A}}$

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

    $$\left( R_j - \frac{\tilde{a}_{j,i}}{\tilde{a}_{i,i}} R_i \right) \longrightarrow R_j$$

**set** $\mathbf{y}$ equal to the last column of $\tilde{\mathbf{A}}$

**set** $\mathbf{x} = (0, 0, \ldots, 0)$

**set** $x_n = \dfrac{y_n}{\tilde{a}_{n,n}}$

**for** $i = n-1, n-2, \ldots, 2, 1$ **do**

  **set** $x_i = \dfrac{1}{\tilde{a}_{i,i}} \left( y_i - \tilde{a}_{i,i+1} x_{i+1} - \tilde{a}_{i,i+2} x_{i+2} - \cdots - \tilde{a}_{i,n} x_n \right)$

**return** $\mathbf{x}$

solve other linear-algebraic problems regarding $\mathbf{A}$. Specifically, Gaussian elimination in fact produces a $LU$-decomposition and such a factorization has several applications other than solving linear systems.

**Example 5.3.** Below we present two matrices that have a $LU$-decomposition:

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

$$\mathbf{A} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Notice that the first matrix is non-singular while the latter is singular.

**Example 5.4.** If a matrix has a $LU$-factorization, it may not necessarily be unique. For example, $\mathbf{A} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 2 \end{smallmatrix}\right)$ has the following $LU$-factorizations:

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 3 \end{pmatrix}$$

**Example 5.5.** Not every matrix has a $LU$-factorization. For example, consider $\mathbf{A} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. Then if

$$\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \mathbf{LU} = \begin{pmatrix} \ell_{1,1} & 0 \\ \ell_{1,2} & \ell_{2,2} \end{pmatrix} \begin{pmatrix} u_{1,1} & u_{1,2} \\ 0 & u_{2,2} \end{pmatrix}$$

then we must have $\ell_{1,1}u_{1,1} = 0$ and therefore $\det(\mathbf{LU}) = \det(\mathbf{L})\det(\mathbf{u}) = \ell_{1,1}\ell_{2,2}u_{1,1}u_{2,2} = 0$. However, $\det(\mathbf{A}) = -1$.

Let us now show how Gaussian elimination can be used to determine a $LU$-decomposition of a given matrix. Since Gaussian elimination is an iterative process that changes the coefficient matrix $\mathbf{A}$ and the vector $\mathbf{b}$, we introduce some notation to denote these objects after each step of the elimination process. Hence, let $\mathbf{A}^{(1)} = \mathbf{A}$, let $\mathbf{b}^{(1)} = \mathbf{b}$, and let $a_{i,j}^{(1)}$ denote the $(i, j)$ entry of $\mathbf{A}^{(1)}$. Using this notation the linear system is therefore $\mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}$. If $a_{1,1}^{(1)} \neq 0$ then we perform the elementary row operation

$$R_j - \frac{a_{j,1}^{(1)}}{a_{1,1}^{(1)}} R_1 \longrightarrow R_j$$

to each row $j = 2, 3, \ldots, n$. To make the notation less cumbersome, let

$$m_{j,1} = \frac{a_{j,1}^{(1)}}{a_{1,1}^{(1)}}$$

for $j = 2, 3, \ldots, n$, and thus the elementary row operations can be written more compactly as

$$R_j - m_{j,1}R_1 \longrightarrow R_j. \tag{5.1}$$

Denote the new coefficient matrix by $\mathbf{A}^{(2)}$ and the new $\mathbf{b}$-vector by $\mathbf{b}^{(2)}$. Hence, the augmented matrix after the first step of the elimination process is

$$\left(\mathbf{A}^{(2)} \quad | \quad \mathbf{b}^{(2)}\right) = \left(\begin{array}{ccccc|c} a_{1,1}^{(2)} & a_{1,2}^{(2)} & a_{1,3}^{(2)} & \cdots & a_{1,n}^{(2)} & b_1^{(2)} \\ 0 & a_{2,2}^{(2)} & a_{2,3}^{(2)} & \cdots & a_{2,n}^{(2)} & b_2^{(2)} \\ 0 & a_{3,2}^{(2)} & a_{3,3}^{(2)} & \cdots & a_{3,n}^{(2)} & b_3^{(2)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a_{n,2}^{(2)} & a_{n,3}^{(2)} & \cdots & a_{n,n}^{(2)} & b_n^{(2)} \end{array}\right)$$

and the equivalent linear system is $\mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$. It turns out that the elementary row operation (5.1) can be accomplished by multiplying both sides of the equation $\mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}$ on the left by the matrix

$$\mathbf{M}^{(1)} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ -m_{2,1} & 1 & 0 & \cdots & 0 \\ -m_{3,1} & 0 & 1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ -m_{n,1} & 0 & 0 & \cdots & 1 \end{pmatrix}$$

In other words, $\mathbf{A}^{(2)} = \mathbf{M}^{(1)}\mathbf{A}^{(1)}$ and $\mathbf{b}^{(2)} = \mathbf{M}^{(1)}\mathbf{b}^{(1)}$ (try this on a $3 \times 3$ example). Hence, the original equation $\mathbf{Ax} = \mathbf{b}$ is multiplied on the left by $\mathbf{M}^{(1)}$:

$$\mathbf{M}^{(1)}\mathbf{Ax} = \mathbf{M}^{(1)}\mathbf{b}.$$

The next step in Gaussian eliminiation proceeds if $a_{2,2}^{(2)} \neq 0$ and then performing the elementary row operations

$$R_j - m_{j,2}R_2 \longrightarrow R_j$$

for $j = 3, \ldots, n$ and where

$$m_{j,2} = \frac{a_{j,2}^{(2)}}{a_{2,2}^{(2)}}.$$

Denote the new coefficient matrix by $\mathbf{A}^{(3)}$ and the new $\mathbf{b}$-vector by $\mathbf{b}^{(3)}$. Hence, the aug-

mented matrix after the second step of the elimination process is

$$\left(\mathbf{A}^{(3)} \quad | \quad \mathbf{b}^{(3)}\right) = \left(\begin{array}{ccccc|c} a_{1,1}^{(3)} & a_{1,2}^{(3)} & a_{1,3}^{(3)} & \cdots & a_{1,n}^{(3)} & b_1^{(3)} \\ 0 & a_{2,2}^{(3)} & a_{2,3}^{(3)} & \cdots & a_{2,n}^{(3)} & b_2^{(3)} \\ 0 & 0 & a_{3,3}^{(3)} & \cdots & a_{3,n}^{(3)} & b_3^{(3)} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n,3}^{(3)} & \cdots & a_{n,n}^{(3)} & b_n^{(3)} \end{array}\right)$$

and the equivalent linear system is $\mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$. This series of elementary row operations is equivalent to multiplying $\mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$ on the left by the matrix

$$\mathbf{M}^{(2)} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & -m_{3,2} & 1 & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & -m_{n,2} & 0 & \cdots & 1 \end{pmatrix}.$$

Hence,

$$\mathbf{A}^{(3)} = \mathbf{M}^{(2)}\mathbf{M}^{(1)}\mathbf{A}$$

and

$$\mathbf{b}^{(3)} = \mathbf{M}^{(2)}\mathbf{M}^{(1)}\mathbf{b}$$

In general, the $k$th step in Gaussian elimination takes the linear system $\mathbf{A}^{(k)}\mathbf{x} = \mathbf{b}^{(k)}$ and, provided $a_{k,k}^{(k)} \neq 0$, we multiply on the left by the matrix

$$\mathbf{M}^{(k)} = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \\ & & & 1 & & \\ & & & -m_{k+1,k} & \ddots & \\ & & & \vdots & & \ddots \\ & & & -m_{n,k} & & & 1 \end{pmatrix}$$

to obtain

$$\mathbf{A}^{(k+1)} = \mathbf{M}^{(k)}\mathbf{A}^{(k)} = \mathbf{M}^{(k)}\mathbf{M}^{(k-1)}\cdots\mathbf{M}^{(1)}\mathbf{A}$$

and

$$\mathbf{b}^{(k+1)} = \mathbf{M}^{(k)}\mathbf{b}^{(k)} = \mathbf{M}^{(k)}\mathbf{M}^{(k-1)}\cdots\mathbf{M}^{(1)}\mathbf{b}$$

The Gaussian elimination process terminates with the coefficient matrix

$$\mathbf{A}^{(n)} = \begin{pmatrix} a_{1,1}^{(n)} & a_{1,2}^{(n)} & \cdots & a_{1,n}^{(n)} \\ 0 & a_{2,2}^{(n)} & \cdots & a_{2,n}^{(n)} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n,n}^{(n)} \end{pmatrix} \tag{5.2}$$

which is obtained as the product

$$\mathbf{A}^{(n)} = \mathbf{M}^{(n-1)}\mathbf{M}^{(n-2)} \cdots \mathbf{M}^{(1)}\mathbf{A}.$$

Notice that $\mathbf{A}^{(n)}$ is an *upper-triangular matrix*; every entry under the main diagonal is zero. To obtain the $LU$-factorization of $\mathbf{A}$, we first note that for each $k$ the matrix $\mathbf{M}^{(k)}$ is invertible and its inverse is

$$[\mathbf{M}^{(k)}]^{-1} = \mathbf{L}^{(k)} = \begin{pmatrix} 1 & & & & & \\ & 1 & & & & \\ & & \ddots & & & \\ & & & 1 & & \\ & & & m_{k+1,k} & \ddots & \\ & & & \vdots & & \ddots \\ & & & m_{n,k} & & 1 \end{pmatrix}$$

(verify this on a $3 \times 3$ case). Thus $\mathbf{L}^{(k)}\mathbf{M}^{(k)} = \mathbf{I}$ for each $k = 1, 2, \ldots, n-1$. Therefore,

$$\mathbf{L}^{(1)}\mathbf{L}^{(2)} \cdots \mathbf{L}^{(n-1)}\mathbf{L}^{(n-1)}\mathbf{M}^{(n-1)}\mathbf{M}^{(n-2)} \cdots \mathbf{M}^{(2)}\mathbf{M}^{(1)}\mathbf{A} = \mathbf{A}$$

Let

$$\mathbf{L} = \mathbf{L}^{(1)}\mathbf{L}^{(2)} \cdots \mathbf{L}^{(n-1)}\mathbf{L}^{(n-1)}$$

and let

$$\mathbf{U} = \mathbf{A}^{(n)} = \mathbf{M}^{(n-1)}\mathbf{M}^{(n-2)} \cdots \mathbf{M}^{(2)}\mathbf{M}^{(1)}\mathbf{A}.$$

and therefore

$$\mathbf{A} = \mathbf{LU}.$$

We have already seen that $\mathbf{U} = \mathbf{A}^{(n)}$ is upper-triangular and one can verify that

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ m_{2,1} & 1 & 0 & \cdots & 0 \\ m_{3,1} & m_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ m_{n,1} & m_{n,2} & \cdots & m_{n,n-1} & 1 \end{pmatrix} \tag{5.3}$$

and thus $\mathbf{L}$ is *lower-triangular*; all entries above the main diagonal are zero. To summarize:

**Theorem 5.3: Gauss Elimination**

Let $\mathbf{A}$ be a $n \times n$ matrix. If Gaussian elimination can be performed without row interchanges on a linear system with $\mathbf{A}$ as the coefficient matrix then $\mathbf{A}$ has a $LU$-decomposition.

In summary, to obtain a $LU$-decomposition of $\mathbf{A}$, we simply perform Gaussian elimination as usual which produces $\mathbf{U} = \mathbf{A}^{(n)}$ and we also keep track of the numbers

$$m_{j,i} = \frac{a_{j,i}^{(i)}}{a_{i,i}^{(i)}}$$

in order to construct $\mathbf{L}$. We emphasize that a matrix $\mathbf{A}$ admits a $LU$-decomposition if Gaussian elimination can be performed without any row interchanges; this seems like a very strong restriction but many matrices that arise in applications admit such a decomposition, and besides, we will soon see how to deal with the case when row interchanges are necessary.

**Example 5.6.** Find a $LU$-decomposition of the matrix

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{pmatrix}.$$

The $LU$-decomposition of a matrix can be used to solve a linear system as follows. Suppose that $\mathbf{A} = \mathbf{LU}$ and we want to solve $\mathbf{Ax} = \mathbf{b}$, or equivalently $\mathbf{LUx} = \mathbf{b}$. Introduce the vector $\mathbf{y} = \mathbf{Ux}$, so that $\mathbf{Ly} = \mathbf{b}$. Since $\mathbf{L}$ is lower-triangular, we can easily solve for $\mathbf{y}$ using *forward substitution* by setting

$$y_1 = b_1$$

and then for $i = 2, \ldots, n$ we have

$$y_i = b_i - \sum_{j=1}^{i-1} m_{i,j} y_j.$$

Now that $\mathbf{y}$ is known, we solve for $\mathbf{x}$ from the equation $\mathbf{Ux} = \mathbf{y}$. Since $\mathbf{U}$ is upper-triangular, we perform back-substitution by first solving for

$$x_n = \frac{y_n}{u_{n,n}}$$

and then for $i = n-1, \ldots, 2, 1$ we have

$$x_i = \frac{1}{u_{i,i}} \left( y_i - \sum_{j=i+1}^{n} u_{i,j} x_j \right).$$

If the coefficient matrix $\mathbf{A}$ will be used to solve many linear systems, then we can first compute the $LU$-decomposition of $\mathbf{A}$ and then use the above procedure to solve the various linear systems.

As we have seen, a matrix $\mathbf{A}$ has a $LU$-decomposition if Gaussian elimination can be performed without any row interchanges, which holds whenever $a_{i,i}^{(i)} \neq 0$ for all $i = 1, 2, \ldots, n-1$. We now want to give conditions on $\mathbf{A}$ that imply that $a_{i,i}^{(i)} \neq 0$. To that end, we introduce the following.

**Definition 5.4: Leading Principal Submatrices**

Let $\mathbf{A}$ be a $n \times n$ matrix and let $k \in \{1, 2, \ldots, n\}$ be arbitrary. The $k \times k$ matrix obtained from $\mathbf{A}$ by keeping only the first $k$ rows and first $k$ columns is a called a *leading principal submatrix* of $\mathbf{A}$ and we denote this matrix by $\mathbf{A}_k$. We call $\mathbf{A}_k$ the $k$th leading principal submatrix of $\mathbf{A}$.

**Example 5.7.** The leading principal submatrices of

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & -1 & 0 \\ 3 & 2 & 0 & -1 \\ -2 & 1 & 0 & 1 \\ -1 & 1 & 2 & 7 \end{pmatrix}$$

are

$$\mathbf{A}_1 = \begin{pmatrix} 1 \end{pmatrix}, \qquad \mathbf{A}_2 = \begin{pmatrix} 1 & 0 \\ 3 & 2 \end{pmatrix}, \qquad \mathbf{A}_3 = \begin{pmatrix} 1 & 0 & -1 \\ 3 & 2 & 0 \\ -2 & 1 & 0 \end{pmatrix}, \qquad \mathbf{A}_4 = \mathbf{A}$$

**Theorem 5.5: $LU$ Existence and Uniqueness**

Let $\mathbf{A}$ be a non-singular $n \times n$ matrix. If $\mathbf{A}_k$ is non-singular for all $k = 1, 2, \ldots, n$ then $\mathbf{A}$ has a unique $LU$-decomposition $\mathbf{A} = \mathbf{LU}$ in which $\mathbf{L}$ is unit lower-triangular. Conversely, if $\mathbf{A}$ has a $LU$-decomposition then $\mathbf{A}_k$ is non-singular for all $k = 1, 2, \ldots, n$.

*Proof.* If $n = 1$ and $\mathbf{A} = (a)$ then $a \neq 0$ because $\mathbf{A}$ is non-singular. Then $\mathbf{A} = (1)(\frac{1}{a})$ and this $LU$-decomposition in which $\mathbf{L}$ is unit lower-triangular is unique. By induction, suppose that the claim is true for some $n \geq 1$ and let $\mathbf{A}$ be a $(n+1) \times (n+1)$ matrix such that $\mathbf{A}_k$ is non-singular for each $k = 1, 2, \ldots, n$. Block partition $\mathbf{A}$ as follows

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & a \end{pmatrix}$$

where $\mathbf{A}_{11}$ is the $n$th leading principal submatrix of $\mathbf{A}$, $\mathbf{A}_{12}$ is a $n \times 1$ column vector, $\mathbf{A}_{21}$ is a $1 \times n$ row vector, and $a$ is a scalar. By hypothesis, $\mathbf{A}_{11}$ and its leading principal submatrices are non-singular, and therefore by induction, $\mathbf{A}_{11}$ has a unique $LU$-decomposition $\mathbf{A}_{11} = \mathbf{L}_{11}\mathbf{U}_{11}$ in which $\mathbf{L}_{11}$ is unit lower-triangular. Since $\mathbf{A}_{11}$ is non-singular, its rows form a basis of $\mathbb{R}^n$ and therefore there exists a unique vector $\mathbf{b}$ such that $\mathbf{A}_{21} = \mathbf{b}^T\mathbf{A}_{11} = \mathbf{b}^T\mathbf{L}_{11}\mathbf{U}_{11}$. To construct a $LU$-decomposition of $\mathbf{A}$ set

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & a \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{L}_{21} & \ell \end{pmatrix} \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{0} & u \end{pmatrix}$$

where $\ell = 1$. Expanding the right-hand side and equating both sides of the equation yields $\mathbf{U}_{12} = \mathbf{L}_{11}^{-1}\mathbf{A}_{12}$, $\mathbf{L}_{21} = \mathbf{b}^T\mathbf{L}_{11}$, and $u = a - \mathbf{L}_{21}\mathbf{U}_{12}$, and uniqueness follows by uniqueness of $\mathbf{L}_{11}$ and $\mathbf{b}$.

To prove the converse, suppose that $\mathbf{A}$ has a $LU$-decomposition, say $\mathbf{A} = \mathbf{L}\mathbf{U}$. Then $\det(\mathbf{A}) = \det(\mathbf{L})\det(\mathbf{U})$, and since $\mathbf{A}$ is non-singular, we have $\det(\mathbf{L}) \neq 0$ and $\det(\mathbf{U}) \neq 0$. Therefore, since $\mathbf{L}$ and $\mathbf{U}$ are triangular, the leading principal submatrices of $\mathbf{L}$ and $\mathbf{U}$ are also non-singular, i.e., $\det(\mathbf{L}_k) \neq 0$ and $\det(\mathbf{U}_k) \neq 0$ for all $k = 1, 2, \ldots, n$. Since $\mathbf{L}$ is lower-triangular and $\mathbf{U}$ is upper-triangular, we have $\mathbf{A}_k = \mathbf{L}_k\mathbf{U}_k$ for all $k = 1, 2, \ldots, n$, that is, the $k$th leading principal submatrix of $\mathbf{A}$ is the product of the $k$th leading principal submatrices of $\mathbf{L}$ and $\mathbf{U}$. Therefore, $\det(\mathbf{A}_k) = \det(\mathbf{L}_k)\det(\mathbf{U}_k) \neq 0$. This proves that $\mathbf{A}_k$ is non-singular. □

## 5.3 LU-Decomposition with Row Interchanges

There are several classes of matrices that admit a $LU$-decomposition and we will consider some of them in the next section. Right now, however, we confront the case when row interchanges are necessary. This occurs when we reach a pivot $a_{i,i}^{(i)}$ that is zero, or if we are implementing partial pivoting (and we are), is very nearly equal to zero. At this step in Gaussian elimination, we find $k \in \{i, i+1, \ldots, n\}$ such that

$$|a_{k,i}^{(i)}| = \max\{|a_{i,i}^{(i)}|, |a_{i+1,i}^{(i)}|, \ldots, |a_{n,i}^{(i)}|\}$$

and we then swap row $R_i$ and row $R_k$ and proceed with the elementary row operations as usual. As the algorithm proceeds and we encounter pivots nearly equal to zero, we perform the necessary row interchanges. If the matrix $\mathbf{A}$ is non-singular (i.e., it has an inverse), the Gaussian elimination algorithm would complete successfully and we would obtain a solution to the given linear system. We would also obtain a $LU$-decomposition but not of the original matrix $\mathbf{A}$ but of a matrix obtained from $\mathbf{A}$ by performing the row interchanges. Hence, if we knew the row interchanges in advance before beginning Gaussian elimination, we could first

perform these row interchanges on $\mathbf{A}$ and then proceed with Gaussian elimination without the need of performing row interchanges. The process of performing row interchanges on a matrix $\mathbf{A}$ is equivalent to multiplying $\mathbf{A}$ on the right by a **permutation matrix P**. Let us review this concept now.

Recall that a **permutation** on the set $\{1, 2, \ldots, n\}$ is a bijective function

$$\sigma : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$$

that is, $\sigma$ is an invertible function. A permutation $\sigma$ can be thought of as a rearrangement of the list $(1, 2, \ldots, n)$. For example, if $n = 7$ and $\sigma$ is the permutation $\sigma(1) = 3, \sigma(2) = 4, \sigma(3) = 1$, $\sigma(4) = 2$, $\sigma(5) = 5$, $\sigma(6) = 7$, and $\sigma(7) = 6$, then $\sigma$ can be represented as

$$\sigma = (3, 4, 1, 2, 5, 7, 6).$$

The set of all permutations on the set $\{1, 2, \ldots, n\}$ will be denoted by $S_n$ and is called the *symmetric group* or the *permutation group* on $n$ symbols. The number of ways to rearrange the ordered list $(1, 2, \ldots, n)$ is $n!$, and therefore $|S_n| = n!$. The group $S_n$ is one of the most important groups in all of mathematics (Permutation group).

For the permutation $\sigma \in S_n$ define the $n \times n$ matrix $\mathbf{P}$ as follows. Let $\mathbf{e}_1, \ldots, \mathbf{e}_n$ denote the standard basis vectors in $\mathbb{R}^n$ thought of as column vectors. Define the matrix $\mathbf{P}$ as

$$\mathbf{P} = \begin{pmatrix} \mathbf{e}_{\sigma(1)}^T \\ \mathbf{e}_{\sigma(2)}^T \\ \mathbf{e}_{\sigma(3)}^T \\ \vdots \\ \mathbf{e}_{\sigma(n)}^T \end{pmatrix}.$$

where $\mathbf{e}^T$ denotes the transpose of $\mathbf{e}$ (recall that $\mathbf{e}$ is a column vector and thus $\mathbf{e}^T$ is a row vector). For example, for the permutation $\sigma = (3, 6, 5, 2, 1, 4)$ the matrix $\mathbf{P}$ is

$$\mathbf{P} = \begin{pmatrix} \mathbf{e}_{\sigma(1)}^T \\ \mathbf{e}_{\sigma(2)}^T \\ \mathbf{e}_{\sigma(3)}^T \\ \mathbf{e}_{\sigma(4)}^T \\ \mathbf{e}_{\sigma(5)}^T \\ \mathbf{e}_{\sigma(6)}^T \end{pmatrix} = \begin{pmatrix} \mathbf{e}_3^T \\ \mathbf{e}_6^T \\ \mathbf{e}_5^T \\ \mathbf{e}_2^T \\ \mathbf{e}_1^T \\ \mathbf{e}_4^T \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}. \tag{5.4}$$

The matrix $\mathbf{P}$ is called the **permutation matrix** associated to $\sigma$. The columns of any permutation matrix $\mathbf{P}$ form an *orthonormal basis* of $\mathbb{R}^n$ since the columns of $\mathbf{P}$ are just the standard basis vectors of $\mathbb{R}^n$ (in a rearranged order). Hence, permutation matrices are **orthogonal matrices**, in other words $\mathbf{P}^T\mathbf{P} = \mathbf{P}\mathbf{P}^T = \mathbf{I}$. Hence, $\mathbf{P}^{-1} = \mathbf{P}^T$, and this implies that $\det(\mathbf{P}) = \pm 1$ for any permutation matrix $\mathbf{P}$.

For any matrix $\mathbf{A}$ and a permutation matrix $\mathbf{P}$ associated to $\sigma$, multiplying $\mathbf{A}$ on the left by $\mathbf{P}$ amounts to permuting the rows of $\mathbf{A}$ as specified by the permutation $\sigma$. For example, if $\sigma = (3, 1, 4, 2)$ and

$$\mathbf{A} = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix}$$

then

$$\mathbf{PA} = \begin{pmatrix} a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \end{pmatrix}$$

Let us now return to the case where Gaussian elimination requires row interchanges on a matrix $\mathbf{A}$. The row interchanges can be fully described by a permutation $\sigma$ and performing these row interchanges on $\mathbf{A}$ amounts to the operation $\mathbf{PA}$ where $\mathbf{P}$ is the permutation matrix associated to $\sigma$. The matrix $\mathbf{PA}$ now has a $LU$-decomposition and therefore

$$\mathbf{PA} = \mathbf{LU}.$$

We emphasize that $\mathbf{LU}$ is the $LU$-decomposition of $\mathbf{PA}$ and not of $\mathbf{A}$. In any case, since $\mathbf{P}$ is invertible and in fact $\mathbf{P}^{-1} = \mathbf{P}^T$, we obtain the following decomposition of $\mathbf{A}$:

$$\mathbf{A} = \mathbf{P}^T\mathbf{LU} = (\mathbf{P}^T\mathbf{L})\mathbf{U}.$$

The matrix $\mathbf{U}$ is still upper-triangular since it is obtained using Gaussian elimination but $\mathbf{P}^T\mathbf{L}$ is not lower-triangular unless $\mathbf{P} = \mathbf{I}$ (clearly, this occurs when no row interchanges were needed on the original $\mathbf{A}$). We will call $\mathbf{A} = \mathbf{P}^T\mathbf{LU}$ a $PLU$-decomposition of $\mathbf{A}$.

In practice, the permutation matrix $\mathbf{P}$ is not constructed but only the associated permutation $\sigma$. To see how to compute $\sigma$, we begin with the identity permutation

$$\sigma^{(0)} = (1, 2, \ldots, n).$$

Gaussian elimination begins with the pivot $a_{1,1}^{(1)}$. If we need to interchange row $R_k$ with row $R_1$ then define a new permutation $\sigma^{(1)}$ by interchanging entries $k$ and 1 in $\sigma^{(0)}$ and leaving all other entries fixed. If no row interchange is needed then $\sigma^{(1)} = \sigma^{(0)}$. Iteratively, suppose that we have computed $\sigma^{(i-1)}$ and we are at pivot $a_{i,i}^{(i)}$. If we need to interchange row $R_k$ with row $R_i$, then define a new permutation $\sigma^{(i)}$ by interchanging entries $k$ and $i$ in $\sigma^{(i-1)}$ and leaving all other entries fixed. If Gaussian elimination proceeds to the last pivot $a_{n,n}^{(n)}$ and $a_{n,n}^{(n)}$ is not zero then no row interchange is necessary and Gaussian elimination ends, and thus the sought out permutation is $\sigma = \sigma^{(n-1)}$.

**Example 5.8.** Suppose that Gaussian elimination is performed with partial pivoting on a linear system of size $n = 8$. Suppose that Gaussian elimination begins by interchanging row $R_4$ with $R_1$. Then $\sigma^{(1)} = (4, 2, 3, 1, 5, 6, 7, 8)$. Suppose next that no row interchange is necessary with pivots $a_{2,2}^{(2)}$ and $a_{3,3}^{(3)}$. Then $\sigma^{(2)} = \sigma^{(1)}$ and $\sigma^{(3)} = \sigma^{(2)}$. Suppose that it is necessary to interchange row $R_4$ and $R_6$. Then $\sigma^{(4)} = (4, 2, 3, 6, 5, 1, 7, 8)$. Next, suppose that no row interchange is needed with pivot $a_{5,5}^{(5)}$ and thus $\sigma^{(5)} = \sigma^{(4)}$. Next, suppose that the row interchange $R_6$ and $R_7$ is needed. Then $\sigma^{(6)} = (4, 2, 3, 6, 5, 7, 1, 8)$. Finally, suppose that Gaussian elimination proceeds without any further row interchanges. Then $\sigma = \sigma^{(6)}$.

We now present the pseudocode for $LU$-decomposition treating the general case of row interchanges and returning $\sigma, \mathbf{L}$, and $\mathbf{U}$.

**Algorithm 5.2** $LU$-Decomposition with row interchanges

INPUT: $\mathbf{A}$ an $n \times n$ matrix

OUTPUT: $(\sigma, \mathbf{L}, \mathbf{U})$ where $\mathbf{A} = \mathbf{P}^T\mathbf{L}\mathbf{U}$ is the $LU$-decomposition with permutation matrix $\mathbf{P}$ associated to $\sigma$

**set** $\varepsilon = 1 \times 10^{-14}$ (or a suitably small tolerance)
**set** $\sigma = (1, 2, \ldots, n)$ (not using zero-based indexing)
**set** $\tilde{\mathbf{A}} = \mathbf{A}$ (a copy of $\mathbf{A}$ that will change)
**set** $\mathbf{L} = \mathbf{I}_{n\times n}$ ($n \times n$ identity matrix)
**for** $i = 1, 2, \ldots, n$ **do** (main loop performing row reduction)
    **find** $k \in \{i, i+1, \ldots, n\}$ such that $|\tilde{a}_{k,i}| = \max\{|\tilde{a}_{i,i}|, |\tilde{a}_{i+1,i}|, \ldots, |\tilde{a}_{n,i}|\}$
    **if** $|\tilde{a}_{k,i}| < \varepsilon$
        **print**( "$LU$-decomposition does not exist!" )
        **return** 0
    **if** $k \neq i$
        **swap** row $i$ and row $k$ in $\tilde{\mathbf{A}}$
        **swap** $\sigma(i)$ and $\sigma(k)$
        **swap** row $i$ and row $k$ of $\mathbf{L}$ for columns from 1 to $i - 1$
    **for** $j = i + 1, i + 2, \ldots, n$ **do**
        **set** $\mathbf{L}_{j,i} = \tilde{a}_{j,i}/\tilde{a}_{i,i}$
        $(R_j - \mathbf{L}_{j,i}R_i) \longrightarrow R_j$
**return** $(\sigma, \mathbf{L}, \tilde{\mathbf{A}})$

## 5.4 Diagonally Dominant and Positive Definite Matrices

We begin with the definition of diagonally dominant matrices.

**Definition 5.6: Diagonally Dominant**

The $n \times n$ matrix $\mathbf{A}$ is said to be **diagonally dominant** if for each $i \in \{1, 2, \ldots, n\}$

$$|a_{i,i}| \geq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|.$$

The matrix $\mathbf{A}$ is **strictly diagonally dominant** if for each $i \in \{1, 2, \ldots, n\}$

$$|a_{i,i}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{i,j}|.$$

Clearly, a strictly diagonally dominant matrix is diagonally dominant.

**Example 5.9.** Match the matrix on the left with the property on the right.

$$\begin{pmatrix} -3 & 2 & -1 \\ 0 & 2 & 1 \\ -1 & 4 & 7 \end{pmatrix}$$ not diagonally dominant

$$\begin{pmatrix} -4 & -2 & -1 \\ 0 & 2 & 1 \\ -1 & 4 & 6 \end{pmatrix}$$ diagonally dominant but not strictly

$$\begin{pmatrix} 3 & 2 & -1 \\ 0 & 2 & 1 \\ -1 & 4 & 2 \end{pmatrix}$$ strictly diagonally dominant

**Example 5.10.** The coefficient matrix of the linear system used to solve for the coefficients

in a natural cubic spline is strictly diagonally dominant:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 0 & \cdots & \cdots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & \cdots & \cdots & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & 0 \\ 0 & \cdots & \cdots & 0 & h_{n-1} & 2(h_{n-1}+h_n) & h_n \\ 0 & 0 & \cdots & \cdots & 0 & 0 & 1 \end{pmatrix}$$

**Example 5.11.** Laplacian and signless Laplacian matrices of graphs are diagonally dominant matrices but are not strictly diagonally dominant.

Strictly diagonally dominant matrices have nice properties, for example:

**Theorem 5.7**

Let $\mathbf{A}$ be a strictly diagonally dominant matrix. Then $\mathbf{A}$ is invertible and $\mathbf{A}$ has a $LU$-decomposition.

Before we prove Theorem 5.7, it is useful to introduce the following now.

**Theorem 5.8: Gershgorin Circle Theorem**

Let $\mathbf{A} = (a_{i,j})$ be a $n \times n$ matrix, let

$$r_k = \sum_{\substack{j=1 \\ j \neq k}}^{n} |a_{k,j}|,$$

and let $D_k$ be the closed disc

$$D_k = \{z \in \mathbb{C} \ : \ |z - a_{k,k}| \leq r_k\}$$

in the complex plane. Then the eigenvalues of $\mathbf{A}$ are contained in the union $D_1 \cup D_2 \cup \cdots \cup D_n$.

*Proof.* Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ be an eigenvector of $\mathbf{A}$ with corresponding eigenvalue $\lambda$. Let $k$ be such that $|x_k| = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$ and note that $|x_k| \neq 0$ because $\mathbf{x}$ is not the

zero vector. The $k$th entry of both sides of the equation $\mathbf{Ax} = \lambda\mathbf{x}$ is

$$\lambda x_k = \sum_{j=1}^{n} a_{k,j}x_j$$

and therefore

$$(\lambda - a_{k,k})x_k = \sum_{\substack{j=1 \\ j\neq k}}^{n} a_{k,j}x_j.$$

By the triangle inequality we have

$$|\lambda - a_{k,k}||x_k| \leq \sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{k,j}||x_j|$$

$$\leq |x_k| \sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{k,j}|$$

and dividing by $|x_k|$ on both sides we obtain

$$|\lambda - a_{k,k}| \leq \sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{k,j}|.$$

This proves that $\lambda \in D_k$ and the proof is complete. □

Now we prove Theorem 5.7.

*Proof of Theorem 5.7.* By the Gershgorin Circle theorem, if $\lambda$ is an eigenvalue of $\mathbf{A}$ then $\lambda$ is contained in some Gershgorin disc, say it is $D_k = \{z \in \mathbb{C} \ : \ |z - a_{k,k}| \leq r_k\}$ for some $k \in \{1, 2, \ldots, n\}$. Since $|a_{k,k}| > \sum_{\substack{j=1 \\ j\neq k}}^{n} |a_{k,j}|$ then $D_k$ does not intersect the origin and therefore $\lambda \neq 0$. It is a basic fact from linear algebra that a matrix is invertible if and only if it does not contain zero as an eigenvalue. This proves the first statement.

Let $k \in \{1, 2, \ldots, n\}$ and consider the leading principal submatrix $\mathbf{A}_k$. It is clear that $\mathbf{A}_k$ is strictly diagonally dominant and therefore $\mathbf{A}_k$ is non-singular, i.e., $\det(\mathbf{A}_k) \neq 0$. By Theorem 5.5, we conclude that $\mathbf{A}$ has a $LU$-decomposition. □

We now introduce another important class of matrices.

**Definition 5.9: Positive Definiteness**

Let $\mathbf{A}$ be a symmetric matrix. We say that $\mathbf{A}$ is **positive definite** if $\mathbf{x}^T\mathbf{Ax} > 0$ for all non-zero vectors $\mathbf{x}$. We say that $\mathbf{A}$ is **positive semi-definite** if $\mathbf{x}^T\mathbf{Ax} \geq 0$ for all non-zero vectors $\mathbf{x}$.

We emphasize that we have defined positive definiteness for symmetric matrices, although symmetry is not required for $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for $\mathbf{x} \neq 0$, for example $\mathbf{A} = \begin{pmatrix} 1 & \frac{1}{3} \\ \frac{1}{4} & 1 \end{pmatrix}$. However, symmetric matrices will be the class of matrices that will be interest to us.

**Example 5.12.** Consider the matrix

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

Then if $\mathbf{x} = (x_1, x_2, x_3)$ we have

$$\begin{aligned} \mathbf{x}^T\mathbf{A}\mathbf{x} &= \begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \begin{pmatrix} 2x_1 - x_2 \\ -x_1 + 2x_2 - x_3 \\ -x_2 + 2x_3 \end{pmatrix} \\ &= 2x_1^2 - x_1x_2 - x_1x_2 + 2x_2^2 - x_2x_3 - x_2x_3 + 2x_3^2 \\ &= x_1^2 + (x_1^2 - 2x_1x_2 + x_2^2) + (x_2^2 - 2x_2x_3 + x_3^2) + x_3^2 \\ &= x_1^2 + (x_1 - x_2)^2 + (x_2 - x_3)^2 + x_3^2 \end{aligned}$$

It is clear that $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for all $\mathbf{x} \neq 0$ and therefore $\mathbf{A}$ is positive definite.

Below we prove two useful properties of positive definite matrices.

**Theorem 5.10**

Suppose that $\mathbf{A} = (a_{i,j})$ is a positive definite $n \times n$ matrix. Then $\mathbf{A}$ is invertible and $a_{i,i} > 0$ for all $i = 1, 2, \ldots, n$.

*Proof.* Suppose that $\mathbf{A}\mathbf{x} = \mathbf{0}$. Then $\mathbf{x}^T\mathbf{A}\mathbf{x} = 0$. Since $\mathbf{A}$ is positive definite then $\mathbf{x} = \mathbf{0}$. This proves that $\mathbf{A}$ has a trivial kernel and is therefore invertible. Let $\mathbf{e}_i$ denote the $i$th standard basis vector of $\mathbb{R}^n$. One computes that $\mathbf{e}_i^T\mathbf{A}\mathbf{e}_i = a_{i,i}$ and since $\mathbf{A}$ is positive definite then $a_{i,i} > 0$. □

**Lemma 5.11**

Let $\mathbf{A}$ be a symmetric matrix. Then $\mathbf{A}$ is positive definite if and only if every leading principal submatrix of $\mathbf{A}$ is positive definite.

*Proof.* Suppose that $\mathbf{A}$ is a $n \times n$ positive definite matrix and let $\mathbf{A}_k$ be the $k$th leading principal submatrix of $\mathbf{A}$. Let $\tilde{\mathbf{x}} \in \mathbb{R}^k$ be a non-zero vector and let $\mathbf{x} = (\tilde{\mathbf{x}}, 0, \ldots, 0) \in \mathbb{R}^n$ be an extension of $\tilde{\mathbf{x}}$. Then

$$\tilde{\mathbf{x}}^T\mathbf{A}_k\tilde{\mathbf{x}} = \mathbf{x}^T\mathbf{A}\mathbf{x} > 0.$$

This proves that $\mathbf{A}_k$ is positive definite. The other direction is trivial. □

We now have the following characterization of positive definite matrices.

**Theorem 5.12: Positive Definiteness and Eigenvalues**

Let $\mathbf{A}$ be a symmetric matrix. Then $\mathbf{A}$ is positive definite if and only if $\mathbf{A}$ has positive eigenvalues.

*Proof.* Suppose that $\mathbf{A}$ is positive definite. If $\mathbf{x}$ is an eigenvector of $\mathbf{A}$ with eigenvalue $\lambda$ then $\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T(\lambda\mathbf{x}) = \lambda\mathbf{x}^T\mathbf{x} = \lambda\|\mathbf{x}\|^2$. Positive definiteness of $\mathbf{A}$ implies that $\lambda\|\mathbf{x}\|^2 > 0$ and since $\|\mathbf{x}\|^2 > 0$ we must have $\lambda > 0$.

To prove the converse, we note that since $\mathbf{A}$ is symmetric, there exists an orthonormal basis $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ of $\mathbb{R}^n$ consisting of eigenvectors of $\mathbf{A}$. Let $\lambda_i$ be the eigenvalue corresponding to $\mathbf{x}_i$. Then our assumption is that $\lambda_i > 0$. For any vector $\mathbf{x} \in \mathbb{R}^n$ there exists constants $c_1, c_2, \ldots, c_n$ such that $\mathbf{x} = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_n\mathbf{x}_n$. Then one can show that

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \lambda_1 c_1^2 + \lambda_2 c_2^2 + \cdots + \lambda_n c_n^2.$$

If $\mathbf{x} \neq \mathbf{0}$ then at least one $c_i$ is non-zero and then $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$. □

We now come to the main result for positive definite matrices and their $LU$-decompositions.

**Theorem 5.13: Positive Definiteness and $LU$-Decomposition**

Let $\mathbf{A}$ be a symmetric matrix. Then $\mathbf{A}$ is positive definite if and only if $\mathbf{A}$ has a $LU$-decomposition with all pivots positive. In this case, Gaussian elimination is stable with respect to round-off error.

*Proof.* We prove only the forward direction. Suppose that $\mathbf{A}$ is positive definite. Then by Lemma 5.11, all the leading principal submatrices $\mathbf{A}_k$ of $\mathbf{A}$ are positive definite. This implies that all leading principal submatrices $\mathbf{A}_k$ are non-singular and therefore $\mathbf{A}$ has a $LU$-decomposition, say $\mathbf{A} = \mathbf{L}\mathbf{U}$. Now, $\det(\mathbf{A}_k) = u_{1,1}u_{2,2}\cdots u_{k,k}$. Since $\mathbf{A}_k$ is positive definite then $\det(\mathbf{A}_k) > 0$. Hence $u_{1,1} > 0$. By induction this implies that $u_{k,k} > 0$ for all $k = 1, 2, \ldots, n$. Hence, all pivots are positive. □

According to Theorem 5.13, if $\mathbf{A}$ is a symmetric positive definite matrix then $\mathbf{A}$ has a $LU$-decomposition, say $\mathbf{A} = \mathbf{L}\mathbf{U}$ where

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{2,1} & 1 & 0 & \cdots & 0 \\ \ell_{3,1} & \ell_{3,2} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ \ell_{n,1} & \ell_{n,2} & \cdots & \ell_{n,n-1} & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} u_{1,1} & u_{1,2} & \cdots & u_{1,n} \\ 0 & u_{2,2} & \cdots & u_{2,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & u_{n,n} \end{pmatrix}$$

and all pivots $u_{k,k}$ are positive. Therefore, we can write

$$\mathbf{U} = \begin{pmatrix} u_{1,1} & & & & \\ & u_{2,2} & & & \\ & & \ddots & & \\ & & & \ddots & \\ & & & & u_{n,n} \end{pmatrix} \begin{pmatrix} 1 & \frac{u_{1,2}}{u_{1,1}} & \frac{u_{1,3}}{u_{1,1}} & \cdots & \frac{u_{1,n}}{u_{1,1}} \\ & 1 & \frac{u_{2,3}}{u_{2,2}} & \cdots & \frac{u_{2,n}}{u_{2,2}} \\ & & \ddots & & \vdots \\ & & & \ddots & \vdots \\ & & & & 1 \end{pmatrix} = \mathbf{D}\widetilde{\mathbf{U}}$$

Hence, the $LU$-decomposition can be further decomposed as $\mathbf{A} = \mathbf{L}\mathbf{D}\widetilde{\mathbf{U}}$. Since $\mathbf{A}$ is symmetric we have

$$\mathbf{L}\mathbf{D}\widetilde{\mathbf{U}} = \mathbf{A} = \mathbf{A}^T = \widetilde{\mathbf{U}}^T\mathbf{D}\mathbf{L}^T = \widetilde{\mathbf{U}}^T(\mathbf{D}\mathbf{L}^T).$$

Now $\widetilde{\mathbf{U}}^T$ is a unit lower triangular matrix and $(\mathbf{D}\mathbf{L}^T)$ is upper triangular. By uniqueness of the decomposition $\mathbf{A} = \mathbf{L}\mathbf{U}$ (Theorem 5.5) we must have $\widetilde{\mathbf{U}}^T = \mathbf{L}$. Hence,

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$$

We have proved the following.

**Corollary 5.14: LDL-Factorication**

Let $\mathbf{A}$ be a symmetric and positive definite matrix. There exists a unique unit lower-triangular matrix $\mathbf{L}$ and unique diagonal matrix $\mathbf{D}$ with positive diagonal entries such that $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$.

Suppose that $\mathbf{A}$ is a symmetric and positive definite matrix and let $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$ be the decomposition from Corollary 5.14. Since all diagonal entries of $\mathbf{D}$ are positive we can write

$$\mathbf{D} = \sqrt{\mathbf{D}}\sqrt{\mathbf{D}}$$

where $\sqrt{\mathbf{D}}$ is the diagonal matrix with diagonal entires $\sqrt{u_{k,k}}$. Therefore,

$$\mathbf{A} = \mathbf{L}\sqrt{\mathbf{D}}\sqrt{\mathbf{D}}\mathbf{L}^T = (\mathbf{L}\sqrt{\mathbf{D}})(\sqrt{\mathbf{D}}\mathbf{L}^T) = (\mathbf{L}\sqrt{\mathbf{D}})(\mathbf{L}\sqrt{\mathbf{D}})^T$$

Because $\sqrt{\mathbf{D}}$ is diagonal and $\mathbf{L}$ is a unit lower-triangular matrix it follows that $\widetilde{\mathbf{L}} = \mathbf{L}\sqrt{\mathbf{D}}$ is a lower-triangular matrix with positive entries along the diagonal. We have proved the following.

**Corollary 5.15: Cholesky Factorization**

Let $\mathbf{A}$ be a symmetric and positive definite matrix. There exists a unique lower-triangular matrix $\widetilde{\mathbf{L}}$ with positive entries along the diagonal such that $\mathbf{A} = \widetilde{\mathbf{L}}\widetilde{\mathbf{L}}^T$.

# 6

# Iterative Techniques for Solving Linear Systems

In this section, we will study iterative techniques for solving a system of linear equations $\mathbf{Ax} = \mathbf{b}$. The methods we will consider are similar to the fixed-point iteration techniques for finding the solutions to a scalar equation such as $f(x) = 0$. Compared to Gaussian elimination, iterative techniques are generally more efficient for large linear systems with a sparse coefficient matrix. Before we study iterative techniques for linear systems, and to motivate what is to come, we review the main ingredients in fixed-point iteration on $\mathbb{R}$.

Fixed-point iteration on $\mathbb{R}$ is concerned with finding a fixed-point of a function $g$, i.e., a point $p^* \in \mathbb{R}$ such that $g(p^*) = p^*$. We begin with an initial guess $p_0$ and compute $p_k = g(p_{k-1})$ for $k \geq 1$. Under certain conditions on $g$, the sequence $(p_k)$ converges to $p^*$, that is, given any $\varepsilon > 0$ there exists $N \in \mathbb{N}$ such that $|p_k - p^*| < \varepsilon$ for all $k \geq N$. In practice, we iterate until $|p_k - g(p_k)| < \varepsilon$ for some given tolerance $\varepsilon$. In order to generalize the notion of fixed-point iteration on $\mathbb{R}^n$, we need to introduce (1) sequences in $\mathbb{R}^n$, (2) a notion of distance in $\mathbb{R}^n$, and (3) convergence of sequences in $\mathbb{R}^n$. To do all of this, all we need is to equip $\mathbb{R}^n$ with a generalization of the absolute value on $\mathbb{R}$.

## 6.1 Vector and Matrix Norms

We begin with the definition of a **norm** on $\mathbb{R}^n$ which generalizes the absolute value on $\mathbb{R}$. Norms are used to measure distance from the origin in $\mathbb{R}^n$.

**Definition 6.1: Vector norm**

A **norm** on $\mathbb{R}^n$ is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ with the following properties:
(i) $\|\mathbf{x}\| \geq 0$
(ii) $\|\mathbf{x}\| = 0$ if and only if $\mathbf{x} = \mathbf{0}$
(iii) $\|\alpha\mathbf{x}\| = |\alpha|\|\mathbf{x}\|$ for every $\alpha \in \mathbb{R}$
(iv) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$ (Triangle inequality)

The first norm we will introduce and denoted by $\|\cdot\|_\infty$ is called the $\ell_\infty$**-norm**, and defined as

$$\|\mathbf{x}\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}.$$

One can verify that $\|\cdot\|_\infty$ is indeed a norm by verifying that properties (i)-(iv) are satisfied for $\|\cdot\|_\infty$. For instance, to verify that $\|\cdot\|_\infty$ satisfies the triangle inequality:

$$\begin{aligned}
\|\mathbf{x} + \mathbf{y}\|_\infty &= \max\{|x_1 + y_1|, |x_2 + y_2|, \ldots, |x_n + y_n|\} \\
&\leq \max\{|x_1| + |y_1|, |x_2| + |y_2|, \ldots, |x_n| + |y_n|\} \\
&\leq \max\{|x_1|, |x_2|, \ldots, |x_n|\} + \max\{|y_1|, |y_2|, \ldots, |y_n|\} \\
&= \|\mathbf{x}\|_\infty + \|\mathbf{y}\|_\infty
\end{aligned}$$

The next norm we introduce is related to the **standard Euclidean inner product** on $\mathbb{R}^n$ defined as

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{n} x_i y_i = \mathbf{x}^T \mathbf{y}.$$

The inner product $\langle \cdot, \cdot \rangle$ satisfies the following properties:

(a) $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$, and $\langle \mathbf{x}, \mathbf{x} \rangle = 0$ if and only if $\mathbf{x} = \mathbf{0}$
(b) $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$ (symmetry)
(c) $\langle \mathbf{x} + \mathbf{y}, \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{z} \rangle + \langle \mathbf{y}, \mathbf{z} \rangle$ (additivity)
(d) $\langle \alpha\mathbf{x}, \mathbf{y} \rangle = \alpha \langle \mathbf{x}, \mathbf{y} \rangle$ (homogeneity)

From the symmetry and the additivity properties of $\langle \cdot, \cdot \rangle$ we have $\langle \mathbf{x}, \mathbf{y} + \mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle$. We say that two vectors $\mathbf{x}$ and $\mathbf{y}$ in $\mathbb{R}^n$ are **orthogonal** if $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.

Using the inner product $\langle \cdot, \cdot \rangle$ we define the **the Euclidean norm**, or the $\ell_2$**-norm**, as

$$\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle} = \left(x_1^2 + x_2^2 + \cdots + x_n^2\right)^{1/2}.$$

We say that the norm $\|\cdot\|_2$ is **induced** by the inner product $\langle \cdot, \cdot \rangle$. It is straightforward to verify that the defining properties (i)-(iii) of a norm do hold for $\|\cdot\|_2$ but to prove the triangle inequality we use the following.

**Theorem 6.2: Cauchy-Schwarz**

For any vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ it holds that

$$\langle \mathbf{x}, \mathbf{y} \rangle \leq \|\mathbf{x}\|_2 \cdot \|\mathbf{y}\|_2$$

We now use the Cauchy-Schwarz inequality to prove that the $\ell_2$-norm satisfies the triangle inequality:

$$\begin{aligned}
\|\mathbf{x}+\mathbf{y}\|_2^2 &= \langle \mathbf{x}+\mathbf{y}, \mathbf{x}+\mathbf{y} \rangle \\
&= \langle \mathbf{x}, \mathbf{x} \rangle + 2\langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle \\
&\leq \langle \mathbf{x}, \mathbf{x} \rangle + 2\|\mathbf{x}\|_2\|\mathbf{y}\|_2 + \langle \mathbf{y}, \mathbf{y} \rangle \\
&= \|\mathbf{x}\|_2^2 + 2\|\mathbf{x}\|_2\|\mathbf{y}\|_2 + \|\mathbf{y}\|_2^2 \\
&= (\|\mathbf{x}\|_2 + \|\mathbf{y}\|_2)^2
\end{aligned}$$

and the proof is complete by taking the square root of both sides of $\|\mathbf{x}+\mathbf{y}\|_2^2 \leq (\|\mathbf{x}\|_2+\|\mathbf{y}\|_2)^2$.

**Example 6.1.** Let $\|\cdot\|$ denote the Euclidean norm on $\mathbb{R}^n$. Prove that for any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ it holds that

$$\|\mathbf{x}+\mathbf{y}\| \cdot \|\mathbf{x}-\mathbf{y}\| \leq \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$$

What can you say if $\mathbf{x}$ and $\mathbf{y}$ are orthogonal?

*Solution.* We first note that

$$\|\mathbf{x}-\mathbf{y}\| = \sqrt{\langle \mathbf{x}-\mathbf{y}, \mathbf{x}-\mathbf{y} \rangle} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle - 2\langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle}$$

$$\|\mathbf{x}+\mathbf{y}\| = \sqrt{\langle \mathbf{x}+\mathbf{y}, \mathbf{x}+\mathbf{y} \rangle} = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle + 2\langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle}$$

Let $a = \langle \mathbf{x}, \mathbf{x} \rangle$, $b = \langle \mathbf{x}, \mathbf{y} \rangle$, and $c = \langle \mathbf{y}, \mathbf{y} \rangle$. Hence,

$$\|\mathbf{x}+\mathbf{y}\| \cdot \|\mathbf{x}-\mathbf{y}\| = \sqrt{(a-2b+c)(a+2b+c)}$$

Now $(a-2b+c)(a+2b+c) = (a+c)^2 - 4b^2$, and therefore $(a-2b+c)(a+2b+c) \leq (a+b)^2$. Therefore,

$$\|\mathbf{x}+\mathbf{y}\| \cdot \|\mathbf{x}-\mathbf{y}\| = \sqrt{(a+b)^2 - 4b^2} \leq \sqrt{(a+b)^2} = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{y}, \mathbf{y} \rangle$$

which is equivalent to $\|\mathbf{x}+\mathbf{y}\| \cdot \|\mathbf{x}-\mathbf{y}\| \leq \|\mathbf{x}\|^2 + \|\mathbf{y}\|^2$. □

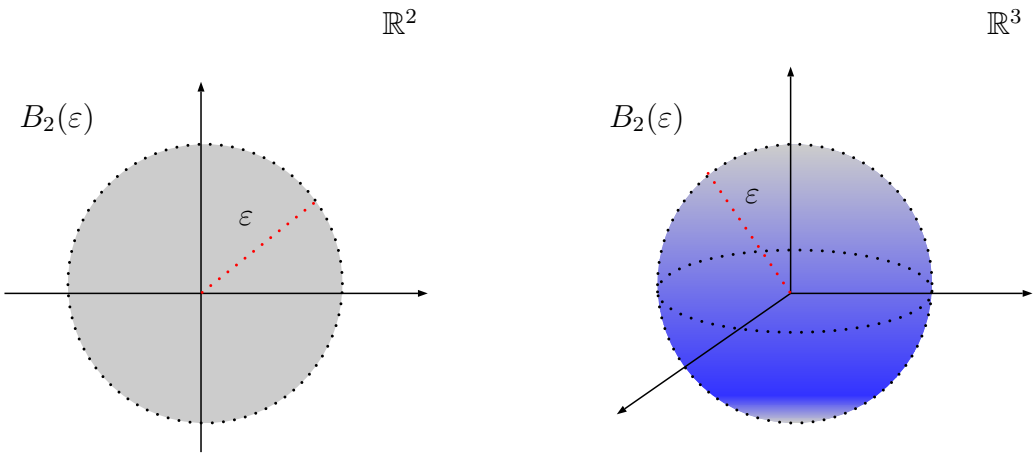



Figure 6.1: Euclidean balls in $\mathbb{R}^2$ (left) and $\mathbb{R}^3$ (right)

As we mentioned above, vector norms generalize the absolute value on $\mathbb{R}$. For instance, if $\varepsilon > 0$, then the set of numbers $x \in \mathbb{R}$ such that $|x| < \varepsilon$ are the numbers whose distance to 0 is less than $\varepsilon$, or equivalently the open interval $(-\varepsilon, \varepsilon)$. Analogously, the set of vectors $\mathbf{x}$ such that $\|\mathbf{x}\|_2 < \varepsilon$ are the vectors whose Euclidean distance to the origin is less than $\varepsilon$. Geometrically, the set $B_2(\varepsilon) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_2 < \varepsilon\}$ is an open ball centered at the origin of radius $\varepsilon$. The cases $n = 2$ and $n = 3$ are illustrated in Figure 6.1. In the case of the $\ell_\infty$-norm, the sets $B_\infty(\varepsilon) = \{\mathbf{x} \in \mathbb{R}^n : \|\mathbf{x}\|_\infty < \varepsilon\}$ are open cubes centered at the origin with sides of length $2\varepsilon$. The cases $n = 2$ and $n = 3$ are illustrated in Figure 6.2.

**Example 6.2.** Find $\|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_\infty$ if $\mathbf{x} = (-1, 0, 2, -3, 1)$.

*Solution.* We compute

$$\|\mathbf{x}\|_2 = \sqrt{(-1)^2 + 0^2 + 2^2 + (-3)^2 + 1^2} = \sqrt{15}$$

and

$$\|\mathbf{x}\|_\infty = \max\{|-1|, |0|, |2|, |-3|, |1|\} = 3$$

□

There is an important relationship between the $\ell_2$-norm and the $\ell_\infty$-norm as described below.

**Theorem 6.3**

For any $\mathbf{x} \in \mathbb{R}^n$,

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{x}\|_\infty$$

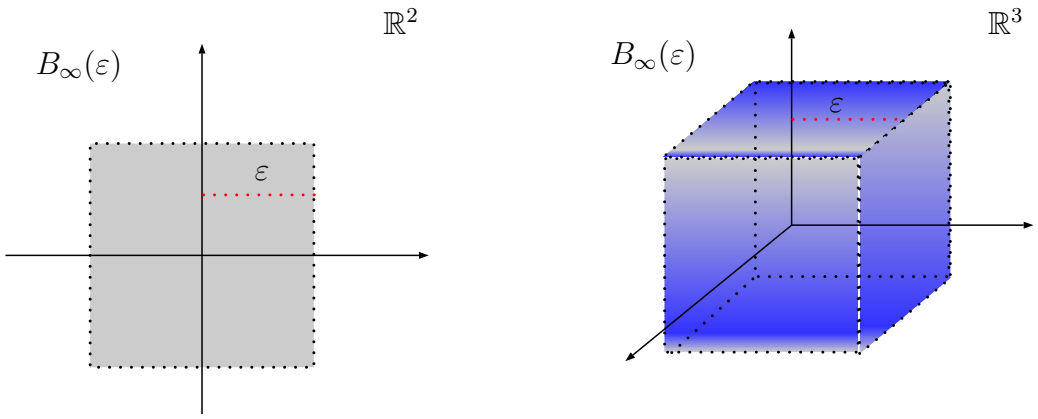



Figure 6.2: $\ell_\infty$-balls in $\mathbb{R}^2$ (left) and $\mathbb{R}^3$ (right)

*Proof.* Let $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ and let $j \in \{1, 2, \ldots, n\}$ be such that $|x_j| = \|\mathbf{x}\|_\infty = \max\{|x_1|, |x_2|, \ldots, |x_n|\}$. Then

$$\|\mathbf{x}\|_\infty^2 = |x_j|^2 \leq \sum_{i=1}^{n} |x_i|^2 = \|\mathbf{x}\|_2^2$$

and therefore $\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\|_2$. On the other hand,

$$\|\mathbf{x}\|_2^2 = \sum_{i=1}^{n} |x_i|^2 \leq \sum_{i=1}^{n} |x_j|^2 = n|x_j|^2 = n\|\mathbf{x}\|_\infty^2$$

and therefore $\|\mathbf{x}\|_2 \leq \sqrt{n}\|\mathbf{x}\|_\infty$. □

Given a norm $\|\cdot\|$ on $\mathbb{R}^n$, the **distance** between $\mathbf{x}$ and $\mathbf{y}$ is

$$\|\mathbf{x} - \mathbf{y}\|.$$

By the properties of a norm, if $\|\mathbf{x} - \mathbf{y}\| = 0$ then $\mathbf{x} - \mathbf{y} = \mathbf{0}$ or equivalently $\mathbf{x} = \mathbf{y}$. Having defined a notion of distance between vectors in $\mathbb{R}^n$, we can now define convergence of sequences in $\mathbb{R}^n$. An **infinite sequence** in $\mathbb{R}^n$ is an infinite list of vectors in $\mathbb{R}^n$ such as $(\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \ldots,)$ where $\mathbf{x}^k \in \mathbb{R}^n$ (technically a sequence is a function from $\mathbb{N}$ to $\mathbb{R}^n$ but we usually ignore this formality and think of a sequence as an infinite list). We will denote the sequence $(\mathbf{x}^1, \mathbf{x}^2, \mathbf{x}^3, \ldots,)$ by $(\mathbf{x}^k)$.

**Definition 6.4: Convergence of sequences**

Let $\|\cdot\|$ be a norm on $\mathbb{R}^n$. A sequence $(\mathbf{x}^{(k)})$ in $\mathbb{R}^n$ is said to converge to $\mathbf{x}^*$ with respect

to $\|\cdot\|$ if for any $\varepsilon > 0$ there exists $N \in \mathbb{N}$ such that

$$\|\mathbf{x}^k - \mathbf{x}^*\| < \varepsilon$$

for all $k \geq N$. In this case we write $\lim_{k\to\infty} \mathbf{x}^k = \mathbf{x}^*$.

Convergence of a sequence in $\mathbb{R}^n$ can be written in terms of convergence of a sequence in $\mathbb{R}$. Let $(\mathbf{x}^k)$ be a sequence in $\mathbb{R}^n$ and let $a_k = \|\mathbf{x}^k - \mathbf{x}^*\|$. Then $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ if and only if $\lim_{k\to\infty} a_k = \lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\| = 0$.

Theorem 6.3 has an important application with regards to convergence of a sequence with respect to $\|\cdot\|_2$ and $\|\cdot\|_\infty$.

**Theorem 6.5**

Let $(\mathbf{x}^k)$ be a sequence in $\mathbb{R}^n$ and let $\mathbf{x}^* \in \mathbb{R}^n$. Then $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ with respect to $\|\cdot\|_2$ if and only if $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ with respect to $\|\cdot\|_\infty$.

*Proof.* By Theorem 6.3,

$$\|\mathbf{x}^k - \mathbf{x}^*\|_\infty \leq \|\mathbf{x}^k - \mathbf{x}^*\|_2 \leq \sqrt{n}\|\mathbf{x}^k - \mathbf{x}^*\|_\infty.$$

If $\lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\|_2 = 0$ then $\lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\|_\infty = 0$ and if $\lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\|_\infty = 0$ then $\lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\|_2 = 0$. □

**Remark 6.1.** Theorem 6.5 is a special case of a more general result. In fact, given *any* two norms $\|\cdot\|$ and $\|\cdot\|'$, a sequence $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ with respect to $\|\cdot\|$ if and only if $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ with respect to $\|\cdot\|'$. Another way of saying this is that *all norms on* $\mathbb{R}^n$ *are equivalent with respect to convergence.*

We now establish a criteria for convergence of sequences in $\mathbb{R}^n$.

**Theorem 6.6**

The sequence $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ with respect to $\|\cdot\|_\infty$ if and only if $\lim_{k\to\infty} x_i^k = x_i^*$ for all $i = 1, 2, \ldots, n$.

*Proof.* Suppose that $\lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\|_\infty = 0$. For any $i = 1, 2, \ldots, n$ we have

$$0 \leq |x_i^k - x_i^*| \leq \|\mathbf{x}^k - \mathbf{x}^*\|_\infty$$

and therefore $\lim_{k\to\infty} |x_i^k - x_i^*| = 0$. Conversely, suppose that $\lim_{k\to\infty} |x_i^k - x_i^*| = 0$ for all $i = 1, 2, \ldots, n$. Then $\lim_{k\to\infty} \sum_{i=1}^n |x_i^k - x_i^*| = 0$. Now

$$0 \leq \|\mathbf{x}^k - \mathbf{x}^*\|_\infty = \max\{|x_1^k - x_1^*|, |x_2^k - x_2^*|, \ldots, |x_n^k - x_n^*|\} \leq \sum_{i=1}^n |x_i^k - x_i^*|$$

from which it follows that $\lim_{k\to\infty} \|\mathbf{x}^k - \mathbf{x}^*\|_\infty = 0$. □

Combining Theorem 6.5 and Theorem 6.6, we obtain the following corollary.

**Corollary 6.7**

The sequence $(\mathbf{x}^k)$ converges to $\mathbf{x}^*$ with respect to $\|\cdot\|_2$ if and only if $\lim_{k\to\infty} x_i^k = x_i^*$ for all $i = 1, 2, \ldots, n$.

A norm on $\mathbb{R}^n$ provides a distance measure between points in $\mathbb{R}^n$. We may want to do this on the space of $n \times n$ matrices $M_n(\mathbb{R})$. For example, if $\mathbf{A}$ has a $LU$-decomposition and a numerical algorithm is used to compute the factors $\mathbf{L}$ and $\mathbf{U}$ then it is natural to measure the distance between $\mathbf{A}$ and $\mathbf{LU}$ since we should not expect $\mathbf{A} = \mathbf{LU}$ exactly due to, for instance, round-off error. The idea is then to define a norm $\|\cdot\|$ on $M_n(\mathbb{R})$ just as we did in $\mathbb{R}^n$ and compute $\|\mathbf{A} - \mathbf{LU}\|$.

**Definition 6.8**

A **matrix norm** on $M_n(\mathbb{R})$ is a function $\|\cdot\| : M_n(\mathbb{R}) \to \mathbb{R}$ that satisfies the following:

(i) $\|\mathbf{A}\| \geq 0$
(ii) $\|\mathbf{A}\| = 0$ if and only if $\mathbf{A} = \mathbf{0}$
(iii) $\|\alpha\mathbf{A}\| = |\alpha|\|\mathbf{A}\|$ for all $\alpha \in \mathbb{R}$
(iv) $\|\mathbf{A} + \mathbf{B}\| \leq \|\mathbf{A}\| + \|\mathbf{B}\|$
(v) $\|\mathbf{AB}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{B}\|$

Notice that properties (i)-(iv) are exactly the same as for vector norms on $\mathbb{R}^n$. The new condition is (v) and is there because $M_n(\mathbb{R})$ has a natural multiplication property and therefore it is frequently useful to relate the norm of $\mathbf{AB}$ with the norms of $\mathbf{A}$ and $\mathbf{B}$. If $\|\cdot\|$ is a matrix norm on $M_n(\mathbb{R})$ then the **distance** between $\mathbf{A}$ and $\mathbf{B}$ with respect to this norm is $\|\mathbf{A} - \mathbf{B}\|$.

There are various ways to construct a matrix norm. Given a matrix $\mathbf{A}$, one such way is to consider the relationship between $\|\mathbf{x}\|$ and $\|\mathbf{Ax}\|$ as $\mathbf{x}$ varies. The idea is that the ratio $\frac{\|\mathbf{Ax}\|}{\|\mathbf{x}\|}$, for $\mathbf{x} \neq \mathbf{0}$, is a measure of the "size" of $\mathbf{A}$.

**Theorem 6.9: Natural Matrix Norms**

Let $\|\cdot\|$ by any matrix norm on $\mathbb{R}^n$. Then $\|\cdot\| : M_n(\mathbb{R}) \to \mathbb{R}$ defined by

$$\|\mathbf{A}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|$$

is a matrix norm.

*Proof.* The only property of a matrix norm that is non-trivial is the multiplicative property. When $\|\mathbf{AB}\| = 0$ the result is trivial hence assume that $\|\mathbf{AB}\| \neq 0$. Hence, there exists

some $\mathbf{x}^*$ with norm one such that $\mathbf{B}\mathbf{x}^* \neq \mathbf{0}$ and $\|\mathbf{AB}\| = \max_{\|\mathbf{x}\|=1} \|\mathbf{ABx}\| = \|\mathbf{ABx}^*\|$. Let $\alpha = \|\mathbf{Bx}^*\|$. Then

$$\begin{aligned}\|\mathbf{AB}\| &= \|\mathbf{ABx}^*\| \\ &= \|\mathbf{ABx}^* \tfrac{1}{\alpha}\| \cdot \|\mathbf{Bx}^*\| \\ &\leq \|\mathbf{A}\|\|\mathbf{Bx}^*\| \\ &\leq \|\mathbf{A}\|\|\mathbf{B}\|\end{aligned}$$

□

Notice that the same symbol $\|\cdot\|$ is used for both the norm on $\mathbb{R}^n$ and the norm on $M_n(\mathbb{R})$. The matrix norm on $M_n(\mathbb{R})$ defined in Theorem 6.9 is called the **natural matrix norm** associated to the vector norm on $\mathbb{R}^n$. From now on, we only consider natural matrix norms, i.e., those induced by vector norms.

**Corollary 6.10**

Let $\|\cdot\|$ be a matrix norm. For any matrix $\mathbf{A}$ and any vector $\mathbf{y}$,

$$\|\mathbf{Ay}\| \leq \|\mathbf{A}\| \cdot \|\mathbf{y}\|$$

*Proof.* The case $\mathbf{y} = \mathbf{0}$ is trivial and we thus assume $\mathbf{y} \neq \mathbf{0}$. Let $\mathbf{x} = \frac{1}{\|\mathbf{y}\|}\mathbf{y}$ and thus $\|\mathbf{x}\| = 1$. Therefore, by definition of $\|\mathbf{A}\|$ we have $\|\mathbf{Ax}\| \leq \|\mathbf{A}\|$. By the homogeneity property of norms, $\|\mathbf{Ax}\| = \frac{1}{\|\mathbf{y}\|}\|\mathbf{Ay}\|$ and therefore

$$\frac{1}{\|\mathbf{y}\|}\|\mathbf{Ay}\| \leq \|\mathbf{A}\|.$$

□

By the previous corollary, an equivalent definition of a natural matrix norm is

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

In view of Theorem 6.9, we define the $\ell_\infty$-norm and the $\ell_2$-norm of a matrix $\mathbf{A}$ as

$$\|\mathbf{A}\|_\infty = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|_\infty$$

and

$$\|\mathbf{A}\|_2 = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\|_2$$

respectively. The following theorem gives a direct method to compute $\|\mathbf{A}\|_\infty$.

**Theorem 6.11**

If $\mathbf{A} \in M_n(\mathbb{R})$ then

$$\|\mathbf{A}\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{i,j}|.$$

In words, the previous theorem states that $\|\mathbf{A}\|_\infty$ is the maximum absolute row sum of $\mathbf{A}$.

**Example 6.3.** According to Theorem 6.11, for

$$\mathbf{A} = \begin{pmatrix} 3 & -2 & -3 & 0 \\ 4 & 0 & 0 & 1 \\ -4 & 3 & -1 & 3 \\ 5 & 0 & 1 & 1 \end{pmatrix}$$

we have $\|\mathbf{A}\|_\infty = \max_{1\leq i\leq 4} \sum_{j=1}^{4} |a_{i,j}| = \max\{8, 5, 11, 7\} = 11.$

## 6.2 Eigenvalues and Convergent Matrices

Let $\mathbf{A}$ be a $n \times n$ matrix. Recall that a number $\lambda$ is called an **eigenvalue** of $\mathbf{A}$ if there exists a non-zero vector $\mathbf{x} \in \mathbb{R}^n$ such that

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}.$$

and in this case we say that $\mathbf{x}$ is an **eigenvector** of $\mathbf{A}$ associated to $\lambda$. The pair $(\lambda, \mathbf{x})$ will be called an **eigenpair** of $\mathbf{A}$. If $\lambda$ is an eigenvalue of $\mathbf{A}$ we denote by

$$E_\lambda = \{\mathbf{x} \in \mathbb{R}^n \,:\, \mathbf{A}\mathbf{x} = \lambda\mathbf{x}\}$$

the set of all eigenvectors of $\mathbf{x}$ associated to $\lambda$ (plus the zero vector) and call it the **$\lambda$-eigenspace**. It is an easy exercise to show that $E_\lambda$ is a subspace of $\mathbb{R}^n$.

If $(\lambda, \mathbf{x})$ is an eigenpair of $\mathbf{A}$ then the eigenvalue equation $\mathbf{A}\mathbf{x} = \lambda\mathbf{x}$ can be written as $(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0}$. Since $\mathbf{x}$ is non-zero, this implies that $(\mathbf{A} - \lambda\mathbf{I})$ is a singular matrix and therefore $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$. Hence, $\lambda$ is an eigenvalue of $\mathbf{A}$ if and only if $\lambda$ is a zero of the function

$$p(x) = \det(\mathbf{A} - x\mathbf{I}).$$

The function $p(x)$ is a monic polynomial in $x$ of order $n$ and is called the **characteristic polynomial** of $\mathbf{A}$. Hence, $\mathbf{A}$ has at most $n$ distinct eigenvalues. In general, the eigenvalues of a matrix $\mathbf{A}$ are complex numbers even if $\mathbf{A}$ contains only real entries. Recall that for a complex number $z = x + iy \in \mathbb{C}$, the modulus of $z$ is $|z| = \sqrt{x^2 + y^2}$. If $z$ is a real number its modulus is simply its absolute value.

**Definition 6.12: Spectral Radius**

The **spectral radius** of the matrix $\mathbf{A}$, denoted by $\rho(\mathbf{A})$, is defined as the maximum modulus of all the eigenvalues of $\mathbf{A}$, that is,

$$\rho(\mathbf{A}) = \max\{|\lambda| \, : \, \lambda \text{ is an eigenvalue of } \mathbf{A}\}$$

There is a close relationship between the matrix norm of a matrix and its spectral radius.

**Theorem 6.13: Eigenvalues and spectral radius**

Let $\mathbf{A}$ be a $n \times n$ matrix. The following hold:
(i) $\|\mathbf{A}\|_2 = \sqrt{\rho(\mathbf{A}^T\mathbf{A})}$
(ii) $\rho(\mathbf{A}) \leq \|\mathbf{A}\|$ for any matrix norm $\|\cdot\|$

*Proof.* By definition,

$$\|\mathbf{A}\|_2^2 = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|^2 = \max_{\|\mathbf{x}\|=1} (\mathbf{x}\mathbf{A})^T\mathbf{A}\mathbf{x} = \max_{\|\mathbf{x}\|=1} \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x}$$

The matrix $\mathbf{B} = \mathbf{A}^T\mathbf{A}$ is symmetric and therefore there exists a basis of $\mathbb{R}^n$ consisting of orthonormal eigenvectors of $\mathbf{B}$, call such a basis $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, with corresponding eigenvalues $\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2$. Since $\mathbf{B}$ is positive semi-definite, and symmetric, the eigenvalues $\sigma_i^2$ of $\mathbf{B}$ are non-negative real numbers. Assume without loss that $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_n^2$, and thus $\rho(\mathbf{A}^T\mathbf{A}) = \sigma_1^2$. Let $\mathbf{x} \in \mathbb{R}^n$ be a unit vector and let $\mathbf{x} = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_n\mathbf{x}_n$, for some $c_1, \ldots, c_n \in \mathbb{R}$. Since $\|\mathbf{x}\| = 1$, and $\mathbf{x}_1, \ldots, \mathbf{x}_n$ are orthonormal, then $c_1^2 + c_2^2 + \cdots + c_n^2 = 1$ and therefore

$$\mathbf{x}^T(\mathbf{A}^T\mathbf{A})\mathbf{x} = c_1^2\sigma_1^2 + c_2^2\sigma_2^2 + \cdots + c_n^2\sigma_n^2 \leq (c_1^2 + c_2^2 + \cdots + c_n^2)\sigma_1^2 = \sigma_1^2$$

This proves that $\|\mathbf{A}\|_2 \leq \sqrt{\sigma_1^2} = \sigma_1$. On the other hand, $\mathbf{x}_1^T(\mathbf{A}^T\mathbf{A})\mathbf{x}_1 = \sigma_1^2\mathbf{x}^T\mathbf{x} = \sigma_1^2$ and thus $\|\mathbf{A}\|_2 \geq \sigma_1$. Hence, we conclude $\|\mathbf{A}\|_2 = \sqrt{\rho(\mathbf{A}^T\mathbf{A})} = \sigma_1$ as claimed.

To prove (ii), let $\mathbf{x}$ be a unit eigenvector of $\mathbf{A}$ with corresponding eigenvalue $\lambda$. Then

$$|\lambda| = |\lambda| \cdot \|\mathbf{x}\| = \|\lambda\mathbf{x}\| = \|\mathbf{A}\mathbf{x}\| \leq \|\mathbf{A}\|\|\mathbf{x}\| = \|\mathbf{A}\|.$$

Therefore, $|\lambda| \leq \|\mathbf{A}\|$ for any eigenvalue $\lambda$ of $\mathbf{A}$ and thus $\rho(\mathbf{A}) = \max|\lambda| \leq \|\mathbf{A}\|$. □

**Example 6.4.** Show that if $\mathbf{A}$ is symmetric then $\rho(\mathbf{A}) = \|\mathbf{A}\|_2$.

*Solution.* The eigenvalues of $\mathbf{A}^2$ are the squares of the eigenvalues of $\mathbf{A}$ and therefore $\rho(\mathbf{A}^2) = \rho(\mathbf{A})^2$. Then since $\mathbf{A}^T = \mathbf{A}$,

$$\begin{aligned}\|\mathbf{A}\|_2 &= \sqrt{\rho(\mathbf{A}^T\mathbf{A})} \\ &= \sqrt{\rho(\mathbf{A}^2)} \\ &= \sqrt{\rho(\mathbf{A})^2} \\ &= \rho(\mathbf{A}).\end{aligned}$$

and the proof is complete. □

Lastly, we consider the notion of convergent matrices which will be used when we consider iterative methods for linear systems.

**Definition 6.14: Convergent Matrices**

A $n \times n$ matrix $\mathbf{A} = (a_{i,j})$ is said to be **convergent** if $\lim_{k\to\infty} (\mathbf{A}^k)_{i,j} = 0$ for all $1 \leq i, j \leq n$. In this case we write that $\lim_{k\to\infty} \mathbf{A}^k = \mathbf{0}$.

We now give several characterizations of convergent matrices.

**Theorem 6.15**

The following statements are equivalent:
(a) $\mathbf{A}$ is a convergent matrix
(b) $\rho(\mathbf{A}) < 1$
(c) $\lim_{k\to\infty} \mathbf{A}^k\mathbf{x} = \mathbf{0}$ for every vector $\mathbf{x} \in \mathbb{R}^n$
(d) $\lim_{k\to\infty} \|\mathbf{A}^k\| = 0$ for some natural norm $\|\cdot\|$
(e) $\lim_{k\to\infty} \|\mathbf{A}^k\| = 0$ for all natural norms $\|\cdot\|$

*Proof.* Suppose that $\lim_{k\to\infty} \mathbf{A}^k = \mathbf{0}$. If $(\lambda, \mathbf{x})$ is an eigenpair then from $\mathbf{Ax} = \lambda\mathbf{x}$ we have $\mathbf{A}^k\mathbf{x} = \lambda^k\mathbf{x}$. Since $\mathbf{A}^k \to \mathbf{0}$ then $\lim_{k\to\infty} \lambda^k\mathbf{x} = \mathbf{0}$ and this happens only if $|\lambda| < 1$. Hence, $\rho(\mathbf{A}) < 1$. To prove that $\rho(\mathbf{A}) < 1$ implies that $\mathbf{A}$ is convergent, assume for simplicity that $\mathbf{A}$ is diagonalizable. Then $\mathbf{A} = \mathbf{QDQ}^{-1}$ where $\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix whose diagonal entries are the eigenvalues of $\mathbf{A}$. Then $\mathbf{A}^k = \mathbf{QD}^k\mathbf{Q}^{-1}$. Now $\mathbf{D}^k = \text{diag}(\lambda_1^k, \lambda_2^k, \ldots, \lambda_n^k)$ and since $\rho(\mathbf{A}) < 1$ then $\mathbf{D}$ is convergent and thus $\mathbf{A}$ is convergent. □

## 6.3 Iterative Methods and Convergence

For many problems, solving a linear system $\mathbf{Ax} = \mathbf{b}$ for the unknown vector $\mathbf{x}$ is best handled using the $LU$-decomposition of $\mathbf{A}$ (or one of its variants) and using forward- and back-substitution. In many cases that arise in practice, however, it is too costly to perform $LU$-decomposition both computationally and with regards to memory storage. Moreover, it may not be necessary to obtain a highly accurate solution to the linear system but only a reasonably good approximation, and in some cases we might have a good initial guess to the solution of the system and can use the initial estimate to speed up the computation of a more accurate approximation.

Suppose that $\mathbf{x}^*$ is a solution to the linear system $\mathbf{Ax} = \mathbf{b}$, that is, $\mathbf{Ax}^* = \mathbf{b}$. Iterative techniques convert the problem of finding $\mathbf{x}^*$ to that of finding a fixed point of a carefully constructed function $\mathbf{G} : \mathbb{R}^n \to \mathbb{R}^n$, i.e., $\mathbf{G}(\mathbf{x}^*) = \mathbf{x}^*$ if and only if $\mathbf{Ax}^* = \mathbf{b}$. Then, to find $\mathbf{x}^*$, we use fixed-point iteration

$$\mathbf{x}^{(k+1)} = \mathbf{G}(\mathbf{x}^{(k)})$$

starting with an initial guess $\mathbf{x}^0$. The construction of $\mathbf{G}$ is made to guarantee that $\mathbf{x}^{(k)}$ converges to $\mathbf{x}^*$. Before we discuss specific ways to construct $\mathbf{G}$, we consider a general approach and give conditions for convergence.

Suppose that we have the decomposition $\mathbf{A} = \mathbf{M} - \mathbf{N}$ where $\mathbf{M}$ is a non-singular matrix. Then $\mathbf{Ax} = \mathbf{b}$ is equivalent to the linear system

$$\mathbf{Mx} = \mathbf{Nx} + \mathbf{b}.$$

Since $\mathbf{N} = \mathbf{M} - \mathbf{A}$ then $\mathbf{M}^{-1}\mathbf{N} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$, and therefore

$$\mathbf{x} = \mathbf{x} + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{Ax}).$$

Then $\mathbf{Ax}^* = \mathbf{b}$ if and only if

$$\mathbf{x}^* = \mathbf{x}^* + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{Ax}^*).$$

To find $\mathbf{x}^*$ using fixed-point iteration, start with an initial guess $\mathbf{x}^0$ and compute for $k \geq 0$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{Ax}^{(k)}). \tag{6.1}$$

or equivalently

$$\mathbf{x}^{(k+1)} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{x}^{(k)} + \mathbf{M}^{-1}\mathbf{b}. \tag{6.2}$$

Hence, fixed-iteration is being performed with the function $\mathbf{G}(\mathbf{x}) = \mathbf{x} + \mathbf{M}^{-1}(\mathbf{b} - \mathbf{Ax})$. The vector $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{Ax}^{(k)}$ is called the **residual** and measures how close $\mathbf{x}^{(k)}$ is to being a solution to $\mathbf{Ax} = \mathbf{b}$. In particular, $\lim_{k\to\infty} \mathbf{r}^{(k)} = \mathbf{0}$ if and only if $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$.

We now analyze the convergence property of the fixed-point iteration (6.1). Let $\mathbf{e}^{(k)} = \mathbf{x}^* - \mathbf{x}^{(k)}$ denote the **error**. Then $\lim_{k\to\infty} \mathbf{e}^{(k)} = \mathbf{0}$ if and only if $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$. We can obtain an expression for the error from the equations

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

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

and subtracting, yielding

$$\mathbf{e}^{(k+1)} = (\mathbf{I} - \mathbf{M}^{-1}\mathbf{A})\mathbf{e}^{(k)}.$$

Introduce the matrix

$$\mathbf{T} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$$

and thus $\mathbf{e}^{(k+1)} = \mathbf{T}\mathbf{e}^{(k)}$. Therefore,

$$\mathbf{e}^{(k)} = \mathbf{T}^k\mathbf{e}^{(0)}.$$

Therefore, $\lim_{k\to} \mathbf{e}^{(k)} = \mathbf{0}$ if and only if $\lim_{k\to\infty} \mathbf{T}^k\mathbf{e}^{(0)} = \mathbf{0}$. Suppose that $\|\cdot\|$ is any matrix norm. Then

$$\|\mathbf{e}^{(k)}\| = \|\mathbf{T}^k\mathbf{e}^{(0)}\| \leq \|\mathbf{T}\|^k \cdot \|\mathbf{e}^{(0)}\|.$$

If $\|\mathbf{T}\| < 1$ then clearly $\lim_{k\to\infty} \|\mathbf{T}\|^k = 0$ and therefore $\mathbf{e}^{(k)}$ converges to zero. It turns out that the convergence of the error $\mathbf{e}^{(k)}$ depends solely on the spectral radius of $\mathbf{T}$.

**Theorem 6.16: Convergence of Iteration**

Suppose that the linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ has a unique solution $\mathbf{x}^*$. For any $\mathbf{x}^{(0)}$, the sequence $(\mathbf{x}^{(k)})$ generated by the iteration (6.1) converges to $\mathbf{x}^*$ if and only if $\rho(\mathbf{T}) < 1$.

The quickest way to prove this theorem is to use the following two facts.

**Lemma 6.17**

Let $\mathbf{X}$ be square matrix and suppose that $\varepsilon > 0$. Then there exists a matrix norm $\|\cdot\|$ such that $\|\mathbf{X}\| < \rho(\mathbf{X}) + \varepsilon$.

By Theorem 6.15, $\rho(\mathbf{X})$ is a lower-bound for $\|\mathbf{X}\|$ for *all* matrix norms and thus Lemma 6.17 implies that

$$\rho(\mathbf{A}) = \inf\{\|\mathbf{A}\| \,:\, \|\cdot\| \text{ is a matrix norm}\}.$$

**Lemma 6.18**

Every matrix norm on $M_n$ is induced by a vector norm.

We now prove Theorem 6.16.

*Proof of Theorem 6.16.* Let $\mathbf{x}^*$ denote the unique solution to $\mathbf{Ax} = \mathbf{b}$. If for any $\mathbf{x}^{(0)}$ the sequence $(\mathbf{x}^{(k)})$ converges to $\mathbf{x}^*$ then $\lim_{k\to\infty} \mathbf{T}^k\mathbf{e}^{(0)} = \mathbf{0}$ for any $\mathbf{e}^{(0)}$. By Theorem 6.15, this implies that $\rho(\mathbf{T}) < 1$. Now suppose that $\rho(\mathbf{T}) < 1$. There exists $\varepsilon > 0$ such that $\rho(\mathbf{T}) + \varepsilon < 1$. By Lemma 6.17, there exists a matrix norm $\|\cdot\|$ such that $\|\mathbf{T}\| < \rho(\mathbf{T}) + \varepsilon < 1$, that is, $\|\mathbf{T}\| < 1$. By the discussion above, this implies that $\lim_{k\to\infty} \mathbf{e}^{(k)} = \mathbf{0}$, that is, $\lim_{k\to\infty} \mathbf{x}^{(k)} = \mathbf{x}^*$. □

## 6.4 Jacobi and Gauss-Siedel Methods

In component form, the linear system $\mathbf{Ax} = \mathbf{b}$ is

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n = b_i$$

for $i = 1, 2, \ldots, n$. If $a_{i,i} \neq 0$ then we can isolate for $x_i$:

$$x_i = \frac{1}{a_{i,i}} \left[ -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{i,j}x_j + b_i \right]$$

The idea of the Jacobi iteration is to use the right-hand side of the above equation as our iteration function. Hence, the Jacobi iteration scheme is

$$x_i^{(k+1)} = \frac{1}{a_{i,i}} \left[ -\sum_{\substack{j=1 \\ j\neq i}}^{n} a_{i,j}x_j^{(k)} + b_i \right]$$

for $i = 1, 2, \ldots, n$. We now write this in matrix form. Let $\mathbf{D} = \operatorname{diag}(a_{1,1}, a_{2,2}, \ldots, a_{n,n})$. Then in matrix form the Jacobi scheme is

$$\begin{aligned} \mathbf{x}^{(k+1)} &= \mathbf{D}^{-1}\left[-(\mathbf{A} - \mathbf{D})\mathbf{x}^{(k)} + \mathbf{b}\right] \\ &= \mathbf{D}^{-1}(\mathbf{D} - \mathbf{A})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b} \\ &= (\mathbf{I} - \mathbf{D}^{-1}\mathbf{A})\mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{b}. \end{aligned}$$

Hence, the Jacobi scheme is the iteration (6.1) with $\mathbf{M} = \mathbf{D}$. In practice, Jacobi iteration is done by first computing the residual $\mathbf{r}^{(k)} = \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$, scaling the $i$th entry of $\mathbf{r}^{(k)}$ by $\frac{1}{a_{i,i}}$ and then adding $\mathbf{x}^{(k)}$:

$$\begin{aligned}\mathbf{r}^{(k)} &= \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}\\ \mathbf{x}^{(k+1)} &= \mathbf{x}^{(k)} + \mathbf{D}^{-1}\mathbf{r}^{(k)}\end{aligned}$$

**Remark 6.2.** In Python (and most numerical software), it is not efficient to compute $\mathbf{D}^{-1}\mathbf{r}$ as a matrix-vector multiplication because multiplying a vector by a diagonal matrix is equivalent to the **Hadamard product** of two vectors:

$$\mathbf{D}^{-1}\mathbf{r} = \begin{bmatrix}\frac{1}{a_{1,1}}\\ \frac{1}{a_{2,2}}\\ \vdots\\ \frac{1}{a_{n,n}}\end{bmatrix} \circ \begin{bmatrix}r_1\\ r_2\\ \vdots\\ r_n\end{bmatrix} = \begin{bmatrix}\frac{r_1}{a_{1,1}}\\ \frac{r_2}{a_{2,2}}\\ \vdots\\ \frac{r_n}{a_{n,n}}\end{bmatrix}$$

If $\mathbf{d} = (\frac{1}{a_{1,1}}, \frac{1}{a_{2,2}}, \ldots, \frac{1}{a_{n,n}})$ and $\mathbf{r}$ are defined as Numpy arrays then $\mathbf{D}^{-1}\mathbf{r}$ can be computed as $\mathbf{d} * \mathbf{r}$. The operator $*$ performs multiplication component-wise on matrices/vectors of the same size. Alternatively, if $\mathbf{d} = a_{1,1}, a_{2,2}, \ldots, a_{n,n})$ then $\mathbf{D}^{-1}\mathbf{r} = \mathbf{r}/\mathbf{d}$.

The Gauss-Siedel method is based on the idea that if $x_1^{(k+1)}, \ldots, x_i^{(k+1)}$ have been computed (for $i > 1$) then instead of using $x_1^{(k)}, \ldots, x_i^{(k)}$ to compute $x_{i+1}^{(k+1)}$ we can use $x_1^{(k+1)}, \ldots, x_i^{(k+1)}$. Therefore, In component form, Gauss-Siedel iteration scheme is

$$x_i^{(k+1)} = \frac{1}{a_{i,i}}\left[\sum_{1\le j<i} a_{i,j}x_j^{(k+1)} - \sum_{i+1<j\le n} a_{i,j}x_j^{(k)} + b_i\right]$$

To write this in matrix form, decompose $\mathbf{A}$ as $\mathbf{A} = \mathbf{A}_L + \mathbf{D} + \mathbf{A}_U$ where $\mathbf{A}_L$ is a lower-triangular matrix (with zeros on the diagonal) and $\mathbf{A}_U$ is an upper-triangular matrix (with zeros along the diagonal). For example,

$$\mathbf{A} = \begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix} = \underbrace{\begin{pmatrix}0 & 0 & 0\\ 4 & 0 & 0\\ 7 & 8 & 0\end{pmatrix}}_{\mathbf{A}_L} + \underbrace{\begin{pmatrix}1 & 0 & 0\\ 0 & 5 & 0\\ 0 & 0 & 9\end{pmatrix}}_{\mathbf{D}} + \underbrace{\begin{pmatrix}0 & 2 & 3\\ 0 & 0 & 6\\ 0 & 0 & 0\end{pmatrix}}_{\mathbf{A}_U}$$

Then in matrix form, the Gauss-Siedel method is

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left[-\mathbf{A}_L\mathbf{x}^{(k+1)} - \mathbf{A}_U\mathbf{x}^{(k)} + \mathbf{b}\right]$$

and after isolating for $\mathbf{x}^{(k+1)}$ and straightforward matrix algebra we obtain

$$\mathbf{x}^{(k+1)} = \left[\mathbf{I} - (\mathbf{A}_L + \mathbf{D})^{-1}\mathbf{A}\right]\mathbf{x}^{(k)} + (\mathbf{A}_L + \mathbf{D})^{-1}\mathbf{b}.$$

Thus, the Gauss-Siedel iteration scheme takes $\mathbf{M} = \mathbf{A}_L + \mathbf{D}$ in (6.1).

We now prove that the Jacobi and Gauss-Siedel methods converge when $\mathbf{A}$ is strictly diagonally dominant. We begin with noting that for any $\mathbf{M}$, the eigenvalues of $\mathbf{T} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A}$ are of the form $1 - \lambda$ where $\lambda$ is an eigenvalue of $\mathbf{M}^{-1}\mathbf{A}$. By Theorem 6.16, convergence follows if and only if $\rho(\mathbf{T}) < 1$ if and only if $|\lambda - 1| < 1$ for all eigenvalues $\lambda$ of $\mathbf{M}^{-1}\mathbf{A}$.

**Theorem 6.19: Convergence of Jacobi and Gauss-Siedel Methods**

Consider the linear system $\mathbf{Ax} = \mathbf{b}$. If $\mathbf{A}$ is strictly diagonally dominant then both the Jacobi and Gauss-Siedel iteration schemes converge to the unique solution $\mathbf{x}^*$ of the system for any initial condition $\mathbf{x}^{(0)}$.

*Proof.* We first note that if $\mathbf{A}$ is strictly diagonally dominant then $\mathbf{A}$ is invertible and therefore $\mathbf{Ax} = \mathbf{b}$ has a unique solution. To prove convergence for either method, we must prove that $\rho(\mathbf{T}) < 1$. We note that since $\mathbf{A}$ is strictly diagonally dominant, for all $i = 1, 2, \ldots, n$ we have

$$|a_{i,i}| > \sum_{j \neq i} |a_{i,j}| \geq 0$$

which is equivalent to

$$\sum_{j \neq i} \frac{|a_{i,j}|}{|a_{i,i}|} < 1.$$

First consider the Jacobi scheme. The product matrix $\mathbf{D}^{-1}\mathbf{A}$ is obtained by multiplying row $i$ of $\mathbf{A}$ by $\frac{1}{a_{i,i}}$ for each $i \in \{1, 2, \ldots, n\}$. Hence, the $i$th row of $\mathbf{D}^{-1}\mathbf{A}$ is

$$\begin{pmatrix} \frac{a_{i,1}}{a_{i,i}} & \frac{a_{i,2}}{a_{i,i}} & \cdots & \frac{a_{i,i-1}}{a_{i,i}} & 1 & \frac{a_{i,i+1}}{a_{i,i}} & \cdots & \frac{a_{i,n}}{a_{i,i}} \end{pmatrix}$$

and therefore

$$\|\mathbf{T}\|_\infty = \|\mathbf{I} - \mathbf{D}^{-1}\mathbf{A}\|_\infty = \max_{1 \leq i \leq n} \sum_{j \neq i} \frac{|a_{i,j}|}{|a_{i,i}|} < 1.$$

Therefore, $\rho(\mathbf{T}) \leq \|\mathbf{T}\|_\infty < 1$ and convergence follows.

Now consider the Gauss-Siedel scheme. In this case, $\mathbf{M} = (\mathbf{A}_L + \mathbf{D})$ and therefore,

$$\mathbf{T} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A} = \mathbf{I} - (\mathbf{A}_L + \mathbf{D})^{-1}(\mathbf{A}_L + \mathbf{D} + \mathbf{A}_U) = -(\mathbf{A}_L + \mathbf{D})^{-1}\mathbf{A}_U.$$

Let $(\lambda, \mathbf{z})$ be an eigenpair of $\mathbf{T}$. Then $\mathbf{Tz} = \lambda\mathbf{z}$ implies that

$$\lambda(\mathbf{A}_L + \mathbf{D})\mathbf{z} = -\mathbf{A}_U\mathbf{z} \tag{6.3}$$

Let $i \in \{1, 2, \ldots, n\}$ be such that $\|\mathbf{z}\|_\infty = |z_i|$. Then from (6.3)

$$\lambda \sum_{j=1}^{i} a_{i,j} z_j = -\sum_{j=i+1}^{n} a_{i,j} z_j$$

and therefore

$$\lambda a_{i,i} z_i = -\sum_{j \le i-1} \lambda a_{i,j} z_j - \sum_{j \ge i+1} a_{i,j} z_j.$$

Hence, by the triangle inequality

$$|\lambda||a_{i,i}||z_i| \le \sum_{j \le i-1} |\lambda||a_{i,j}||z_j| + \sum_{j \ge i+1} |a_{i,j}||z_j| \le |z_i| \left( \sum_{j \le i-1} |\lambda||a_{i,j}| + \sum_{j \ge i+1} |a_{i,j}| \right)$$

and thus

$$|\lambda||a_{i,i}| \le \sum_{j \le i-1} |\lambda||a_{i,j}| + \sum_{j \ge i+1} |a_{i,j}|$$

Hence,

$$|\lambda| \left( 1 - \sum_{j \le i-1} \frac{|a_{i,j}|}{|a_{i,i}|} \right) \le \sum_{j \ge i+1} \frac{|a_{i,j}|}{|a_{i,i}|}$$

and finally

$$|\lambda| \le \frac{\sum_{j \ge i+1} \frac{|a_{i,j}|}{|a_{i,i}|}}{1 - \sum_{j \le i-1} \frac{|a_{i,j}|}{|a_{i,i}|}}$$

We note that by the strictly diagonally dominant assumption on $\mathbf{A}$,

$$1 > \sum_{j \ne i} \frac{|a_{i,j}|}{|a_{i,i}|} = \sum_{j \le i-1} \frac{|a_{i,j}|}{|a_{i,i}|} + \sum_{j \ge i+1} \frac{|a_{i,j}|}{|a_{i,i}|}$$

and consequently

$$1 > \frac{\sum_{j \ge i+1} \frac{|a_{i,j}|}{|a_{i,i}|}}{1 - \sum_{j \le i-1} \frac{|a_{i,j}|}{|a_{i,i}|}}$$

This proves that $|\lambda| < 1$ for all eigenvalues of $\mathbf{T}$ and thus $\rho(\mathbf{T}) < 1$. □

Another condition for convergence of the Gauss-Siedel method is the following.

**Theorem 6.20: Convergence of Gauss-Siedel Method**

Consider the linear system $\mathbf{Ax} = \mathbf{b}$. If $\mathbf{A}$ is symmetric and positive definite then the Gauss-Siedel iteration schemes converges to the unique solution $\mathbf{x}^*$ of the system for any initial condition $\mathbf{x}^{(0)}$.

*Proof.* As before, we have

$$\mathbf{T} = \mathbf{I} - \mathbf{M}^{-1}\mathbf{A} = \mathbf{I} - (\mathbf{A}_L + \mathbf{D})^{-1}(\mathbf{A}_L + \mathbf{D} + \mathbf{A}_U) = -(\mathbf{A}_L + \mathbf{D})^{-1}\mathbf{A}_U$$

Let $(\lambda, \mathbf{z})$ be an eigenpair of $\mathbf{T}$. Then $\mathbf{Tz} = \lambda\mathbf{z}$ implies that

$$\lambda(\mathbf{A}_L + \mathbf{D})\mathbf{z} = -\mathbf{A}_U z = -\mathbf{A}_L^T \mathbf{z}$$

Therefore,

$$\lambda \mathbf{z}^T(\mathbf{A}_L + \mathbf{D})\mathbf{z} = -\mathbf{z}\mathbf{A}_L^T\mathbf{z} = -\mathbf{z}^T\mathbf{A}_L\mathbf{z},$$

that is,

$$\lambda(\mathbf{z}^T\mathbf{A}_L\mathbf{z} + \mathbf{z}^T\mathbf{D}\mathbf{z}) = -\mathbf{z}^T\mathbf{A}_L\mathbf{z}$$

Let $a = \mathbf{z}^T\mathbf{A}_L\mathbf{z}$ and let $b = \mathbf{z}^T\mathbf{D}\mathbf{z} > 0$ (a positive definite matrix has positive diagonal entries). Then

$$\lambda(a + b) = -a$$

Now $(a + b) \neq 0$, otherwise $a < 0$ and thus $-a \neq 0$. Thus,

$$\lambda^2 = \frac{a^2}{a^2 + 2ab + b^2}.$$

Now since $\mathbf{A}$ is positive definite, $\mathbf{z}^T\mathbf{A}\mathbf{z} = a+b+a > 0$ and therefore $2a > -b$ and consequently $2ab + b^2 > 0$. Therefore,

$$\lambda^2 = \frac{a^2}{a^2 + 2ab + b^2} < 1$$

and then $|\lambda| < 1$. Hence, $\rho(\mathbf{T}) < 1$ and convergence follows. □

# 7

# Approximating Eigenvalues

In this section we consider methods to compute and approximate eigenvalues and eigenvectors of matrices.

## 7.1 The Power Method

The Power Method is used to approximate the largest eigenvalue (in magnitude) and corresponding eigenvector of a matrix. Assume that $\mathbf{A}$ has $n$ linearly independent eigenvectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, with corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. Assume that

$$|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|,$$

that is, the eigenvalues are ordered with respect to magnitude from largest to smallest and only $\lambda_1$ satisfies $\rho(\mathbf{A}) = |\lambda_1|$. We will call $\lambda_1$ the **dominant eigenvalue** of $\mathbf{A}$ and a corresponding eigenvector a **dominant eigenvector**. Let $\mathbf{v} \in \mathbb{R}^n$ be fixed. Then since $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ form a basis of $\mathbb{R}^n$, there exists constants $c_1, c_2, \ldots, c_n$ such that $\mathbf{v} = c_1\mathbf{x}_1 + c_2\mathbf{x}_2 + \cdots + c_n\mathbf{x}_n$. Then

$$\mathbf{A}\mathbf{v} = c_1\mathbf{A}\mathbf{x}_1 + c_2\mathbf{A}\mathbf{x}_2 + \cdots + c_n\mathbf{A}\mathbf{x}_n = c_1\lambda_1\mathbf{x}_1 + c_2\lambda_2\mathbf{x}_2 + \cdots + c_n\lambda_n\mathbf{x}_n$$

$$\mathbf{A}^2\mathbf{v} = c_1\lambda_1^2\mathbf{x}_1 + c_2\lambda_2^2\mathbf{x}_2 + \cdots + c_n\lambda_n^2\mathbf{x}_n$$

and in general for any $k \in \mathbb{N}$ we have

$$\mathbf{A}^k\mathbf{v} = c_1\lambda_1^k\mathbf{x}_1 + c_2\lambda_2^k\mathbf{x}_2 + \cdots + c_n\lambda_n^k\mathbf{x}_n$$

Equivalently,

$$\mathbf{A}^k\mathbf{v} = \lambda_1^k\left[c_1\mathbf{x}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)\mathbf{x}_2 + \cdots + c_n\left(\frac{\lambda_n}{\lambda_1}\right)\mathbf{x}_n\right]$$

Since $|\lambda_j/\lambda_1| < 1$, for $j = 2, \ldots, n$, then $\lim_{k\to\infty}\frac{\lambda_j}{\lambda_1} = 0$ for $j = 2, \ldots, n$, and therefore when $k$ is large we have

$$\mathbf{A}^k\mathbf{v} \approx c_1\lambda_1^k\mathbf{x}_1.$$

Hence, when $k$ is large, the vector $\mathbf{A}^k\mathbf{v}$ is approximately a scalar multiple of $\mathbf{x}_1$ (provided $c_1 \neq 0$) and thus $\mathbf{A}^k\mathbf{v}$ is approximately an eigenvector of $\mathbf{A}$ with eigenvalue $\lambda_1$. We could therefore use $\mathbf{A}^k\mathbf{v}$ as an approximation to an eigenvector of $\mathbf{A}$. To obtain an estimate for $\lambda_1$, compute $\mathbf{A}^{k+1}\mathbf{v}$ and then

$$\mathbf{A}^{k+1}\mathbf{v} = \mathbf{A}(\mathbf{A}^k\mathbf{v}) \approx \lambda_1\mathbf{A}^k\mathbf{v}$$

and we can extract a non-zero component of $\mathbf{A}^{k+1}\mathbf{v}$ and $\mathbf{A}^k\mathbf{v}$, and solve for $\lambda_1$:

$$\lambda_1 \approx \frac{(\mathbf{A}^{k+1}\mathbf{v})_i}{(\mathbf{A}^k\mathbf{v})_i}$$

To summarize, the idealized Power Method is to first choose a non-zero vector $\mathbf{v}^{(0)}$ and then compute iteratively the vector $\mathbf{v}^{(k+1)} = \mathbf{A}\mathbf{v}^{(k)}$. For large $k$, we obtain an estimate of $\lambda_1$ as

$$\lambda_1 \approx \frac{v_i^{(k+1)}}{v_i^{(k)}}$$

provided $v_i^{(k)} \neq 0$ and $\mathbf{v}^{(k+1)}$ is an approximate eigenvector.

In practice, numerical difficulties will arise with the above implementation of the Power Method. For instance, if $|\lambda_1| < 1$ then $\lim_{k\to\infty} \mathbf{v}^{(k)} = \mathbf{0}$ and thus both $v_i^{(k+1)}$ and $v_i^{(k)}$ are close to zero and the ratio $\frac{v_i^{(k+1)}}{v_i^{(k)}}$ is likely to be dominated by round-off error. On the other hand, if $|\lambda_1| > 1$ then the sequence of vectors $\mathbf{v}^{(k)}$ is unbounded and numerical over-flow might occur so that $v_i^{(k+1)}$ and $v_i^{(k)}$ are no longer numerically useful to approximate $\lambda_1$. To circumvent these numerical difficulties, we normalize the vectors $\mathbf{v}^{(k)}$ at each iteration step using the $\infty$-norm. We begin with a unit norm vector $\mathbf{v}^{(0)}$ such that there exists an index $p_0 \in \{1, 2, \ldots, n\}$ such that $v_{p_0}^{(0)} = 1 = \|\mathbf{v}^{(0)}\|_\infty$ . Then compute $\mathbf{w}^{(1)} = \mathbf{A}\mathbf{v}^{(0)}$ and then set

$$\mathbf{v}^{(1)} = \frac{\mathbf{w}^{(1)}}{w_{p_1}^{(1)}}$$

where $p_1 \in \{1, 2, \ldots, n\}$ is the least integer such that $|w_{p_1}^{(1)}| = \|\mathbf{w}^{(1)}\|_\infty$. Then $\|\mathbf{v}^{(1)}\|_\infty = v_{p_1}^{(1)} = 1$. Inductively, having computed $\mathbf{v}^{(k)}$ of unit norm with $v_{p_k}^{(k)} = 1$ for some $p_k \in \{1, 2, \ldots, n\}$, compute $\mathbf{w}^{(k+1)} = \mathbf{A}\mathbf{v}^{(k)}$ and then set

$$\mathbf{v}^{(k+1)} = \frac{\mathbf{w}^{(k)}}{w_{p_{k+1}}^{(k+1)}}$$

where $p_{k+1} \in \{1, 2, \ldots, n\}$ is the least integer such that $|w_{p_{k+1}}^{(k+1)}| = \|\mathbf{w}^{(k+1)}\|_\infty$. Then $\|\mathbf{v}^{(k+1)}\|_\infty = v_{p_{k+1}}^{(k+1)} = 1$. From our analysis above, when $k$ is large, $\mathbf{v}^{(k)}$ is a good approximation to an eigenvector of $\mathbf{A}$, that is,

$$\mathbf{w}^{(k+1)} = \mathbf{A}\mathbf{v}^{(k)} \approx \lambda_1\mathbf{v}^{(k)}$$

Now, the maximum entry (in absolute value) of the vector $\lambda_1 \mathbf{v}^{(k)}$ is at $p_k$ since $v_{p_k}^{(k)} = 1$. Hence, $p_{k+1} = p_k$ and $w_{p_{k+1}}^{(k+1)} \approx \lambda_1$. Therefore,

$$\mathbf{v}^{(k+1)} = \frac{\mathbf{w}^{(k+1}}{w_{p_{k+1}}^{(k+1)}} \approx \mathbf{v}^{(k)}.$$

Hence, once $\|\mathbf{v}^{(k+1)} - \mathbf{v}^{(k)}\|_\infty$ is within tolerance, we can use $\mathbf{v}^{(k+1)}$ as an approximation to an eigenvector of $\mathbf{A}$ with eigenvalue $\lambda_1$, and $\lambda_1 \approx w_{p_k}^{(k+1)}$. Below is the pseudocode for the Power Method.

**Algorithm 7.1** Power Method

INPUT: $\mathbf{A}$, $\mathbf{w}$ (initial guess), $N_{\max}$, $\varepsilon > 0$

OUTPUT: Approximations $(\tilde{\lambda}, \tilde{\mathbf{v}})$ to dominating eigenpair of $\mathbf{A}$

**find** smallest integer $p$ such that $|w_p| = \|\mathbf{w}\|_\infty$
**set** $\mathbf{v} = \mathbf{w}/w_p$
**for** $k = 1, \ldots, N_{\max}$
    **set** $\mathbf{w} = \mathbf{A}\mathbf{v}$
    **set** $\lambda = w_p$
    **find** smallest integer $p$ such that $|w_p| = \|\mathbf{w}\|_\infty$
    **if** $|w_p| < \varepsilon$ **then**
        **print**("$\mathbf{A}$ has zero as an eigenvalue; choose new $\mathbf{w}$ and restart")
    **if** $\|\mathbf{v} - \mathbf{w}/w_p\| < \varepsilon$
        **print**("Power method converged!")
        **return** $\lambda, \mathbf{w}/w_p$
    **else**
        $\mathbf{v} = \mathbf{w}/w_p$
**print**("Power method did not converge!")
**return** $\lambda, \mathbf{v}$

**Example 7.1.** Perform three iterations of the Power method for

$$\mathbf{A} = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix}$$

with initial condition $\mathbf{w}^{(0)} = (1, 1, 1)$, and produce approximations to the dominant eigenpair of $\mathbf{A}$.

*Solution.* The eigenvalues of $\mathbf{A}$ are $\lambda_1 = 2.80193774$, $\lambda_2 = 1.44504187$, and $\lambda_3 = -0.2469796$. The initial vector $\mathbf{w}^{(0)}$ is of unit $\infty$-norm and the first entry is 1. Hence, $\mathbf{v}^{(0)} = \mathbf{w}^{(0)}$. Compute

$$\mathbf{w}^{(1)} = \mathbf{A}\mathbf{v}^{(0)} = (3, 2, 3)$$

Then $\mathbf{v}^{(1)} = \mathbf{w}^{(1)}/3 = (1, 2/3, 1)$. One computes that $\|\mathbf{v}^{(0)} - \mathbf{v}^{(1)}\|_\infty = 1/3$. Next, compute

$$\mathbf{w}^{(2)} = \mathbf{A}\mathbf{v}^{(1)} = (3, 2, 8/3)$$

Then $\mathbf{v}^{(2)} = \mathbf{w}^{(2)}/3 = (1, 2/3, 8/9)$. One computes that $\|\mathbf{v}^{(2)} - \mathbf{v}^{(1)}\|_\infty = 1/9$. Next, compute

$$\mathbf{w}^{(3)} = \mathbf{A}\mathbf{v}^{(2)} = (26/9, 17/9, 22/9) \approx (2.8888, 1.8888, 2.4444)$$

Then $\mathbf{v}^{(3)} = \mathbf{w}^{(3)}/(26/9) = (1, 16/26, 22/26)$. One computes that $\|\mathbf{v}^{(2)} - \mathbf{v}^{(1)}\|_\infty \approx 0.0740$. An approximation to $\lambda_1$ is $\lambda_1 \approx \frac{26}{9} = 2.88888$ and an approximation to a dominant eigenvector is $\mathbf{v}^{(3)}$. □

## 7.2 Symmetric Power Method

If $(\lambda, \mathbf{v})$ is an eigenpair of $\mathbf{A}$ with $\mathbf{v}$ a unit eigenvector (unit in the $\|\cdot\|_2$ norm) then

$$\mathbf{v}^T\mathbf{A}\mathbf{v} = \mathbf{v}^T(\lambda\mathbf{v}) = \lambda(\mathbf{v}^T\mathbf{v}) = \lambda.$$

Conversely, if $\mathbf{v}$ is a unit vector (again, in the 2-norm) then if

$$\mathbf{A}\mathbf{v} - (\mathbf{v}^T\mathbf{A}\mathbf{v})\mathbf{v} = \mathbf{0}$$

then $\mathbf{v}$ is an eigenvector of $\mathbf{A}$ with eigenvalue $\mathbf{v}^T\mathbf{A}\mathbf{v}$. Thus, given any unit vector $\mathbf{w}$, if

$$\mathbf{A}\mathbf{w} - (\mathbf{w}^T\mathbf{A}\mathbf{w})\mathbf{w} \approx \mathbf{0}$$

then $\mathbf{w}$ is an approximate eigenvector of $\mathbf{A}$ with approximate eigenvalue $\mathbf{w}^T\mathbf{A}\mathbf{w}$. This forms the basis for the symmetric Power method described below.

Suppose that $\mathbf{v}^{(0)}$ is a unit vector in the 2-norm. Compute

$$\mathbf{w}^{(1)} = \mathbf{A}\mathbf{v}^{(0)}$$

$$\mu_1 = (\mathbf{v}^{(0)})^T\mathbf{w}^{(1)} = (\mathbf{v}^{(0)})^T\mathbf{A}\mathbf{v}^{(0)}$$

From the discussion above, if

$$\|\mathbf{w}^{(1)} - \mu_1\mathbf{v}^{(0)}\|_2 < \varepsilon$$

then $\mathbf{v}^{(0)}$ is a good approximation to an eigenvector of $\mathbf{A}$ with eigenvalue $\mu_1$. Otherwise, set

$$\mathbf{v}^{(1)} = \frac{1}{\|\mathbf{w}^{(1)}\|_2}\mathbf{w}^{(1)}.$$

and repeat the process with $\mathbf{v}^{(1)}$. Thus compute

$$\mathbf{w}^{(2)} = \mathbf{A}\mathbf{v}^{(1)}$$

$$\mu_2 = (\mathbf{v}^{(1)})^T\mathbf{w}^{(2)} = (\mathbf{v}^{(1)})^T\mathbf{A}\mathbf{v}^{(1)}$$

and if

$$\|\mathbf{w}^{(2)} - \mu_2\mathbf{v}^{(1)}\|_2 < \varepsilon$$

then $\mathbf{v}^{(1)}$ is a good approximation to an eigenvector of $\mathbf{A}$ with eigenvalue $\mu_2$. Below is the pseudocode for the Power method applied to a symmetric matrix.

**Algorithm 7.2** Symmetric Power Method

INPUT: $\mathbf{A}$, $\mathbf{w}$ (initial guess), $N_{\max}$, $\varepsilon > 0$

OUTPUT: Approximations $(\tilde{\lambda}, \tilde{\mathbf{v}})$ to dominating eigenpair of symmetric $\mathbf{A}$

```
set v = w/||w||_2
for k = 1, ..., N_max
    set w = Av
    set λ = v^T w
    if ||w − λv||_2 < ε then
        print("Power method converged!")
        return λ, v
    else
        v = w/||w||_2
print("Power method did not converge!")
return λ, v
```

**Example 7.2.** Perform three iterations of the symmetric power method for the matrix

$$\mathbf{A} = \begin{pmatrix} -4 & 1 \\ 1 & 2 \end{pmatrix}$$

with initial condition $\mathbf{w} = (1, 1)$.

## 7.3 PageRank Algorithm

Finding the dominant eigenvector of a matrix has applications in several ranking problems, the most widely known is the PageRank algorithm used by Google to rank webpages. The ranking of the webpages is done to determine the order that webpages are displayed when a user types in a search query. The ranking of the webpages is obtained by finding the dominant eigenvector of the Google matrix $\mathbf{G}$ associated to the network and the individual entries of the dominant eigenvector are ordered which produces the desired ranking. We now describe how we model a network using a matrix and how to create the Google matrix for a given network.

The story begins by first modeling the internet as a collection of vertices and arcs. Each vertex represents an internet website and an arc exists from website $w_i$ to website $w_j$ if in the website $w_i$ there is a link to website $w_j$. To be concrete, consider the directed network shown in Figure 7.1.

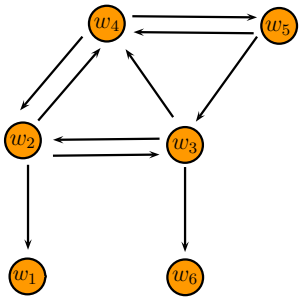



Figure 7.1: This is a tiny network

The **adjacency** matrix for this directed network is

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}.$$

In the adjacency matrix, each column corresponds to the out-going links for the corresponding vertex. For example, vertex $w_2$ links to vertices $\{w_1, w_3, w_4\}$ and thus column $i = 2$ of $\mathbf{A}$ has 1's in positions $\{1, 3, 4\}$ and 0's elsewhere. The rows of $\mathbf{A}$ correspond to the in-links for the corresponding vertex. For example, the non-zero entries of row $j = 3$ are $\{2, 5\}$ because vertex $w_3$ has in-links from vertices $\{w_2, w_5\}$. Vertices that have no out-going links are called **dangling nodes** and the columns corresponding to these nodes are zero columns. For example, in the above tiny network, nodes $w_1$ and $w_6$ are dangling nodes. Dangling nodes can be identified by computing the **out-degree vector**

```
d = np.sum(A, axis = 0)
```

and the entries of **d** that are zero correspond to dangling nodes since **d** records the number of out-going links of each vertex. The out-degree vector for the tiny network above is $\mathbf{d} = (0, 3, 3, 2, 2, 0)$.

We now create what is called the **hyperlink matrix H** of the network. To that end, each non-zero column of **A** is normalized so that its sum is equal to one. For example, if column $\mathbf{A}[:, i]$ is non-zero then the $i$th column of **H** is

$$\mathbf{H}[:, i] = \frac{1}{\mathbf{d}[i]}\mathbf{A}[:, i].$$

For the tiny network above, the hyperlink matrix is

$$\mathbf{H} = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \end{bmatrix}$$

We now make one further modification to the hyperlink matrix **H** to create a new matrix $\overline{\mathbf{H}}$. The matrix $\overline{\mathbf{H}}$ is obtained by replacing each zero column of **H** with the vector $\frac{1}{n}\mathbf{e}$ where $\mathbf{e} = (1, 1, 1 \ldots, 1)$ is the all ones vector. This modification of **H** fixes the **dangling node** problem (recall that a dangling node corresponds to a zero column of **H**). It is straightforward to compute $\overline{\mathbf{H}}$ using a `for` loop but we do not actually want to compute or store $\overline{\mathbf{H}}$. Instead we want to decompose $\overline{\mathbf{H}}$ in the form

$$\overline{\mathbf{H}} = \mathbf{H} + \mathbf{X}$$

where **X** has two types of columns: if vertex $w_i$ is a dangling node then $\mathbf{X}[:, i] = \frac{1}{n}\mathbf{e}$ and, if $w_i$ is not a dangling node then $\mathbf{X}[:, i] = \mathbf{0}$. To see how **X** can be computed, first define the **dangling node vector a** as:

$$\mathbf{a}[i] = \begin{cases} 1, & \text{if } w_i \text{ is a dangling node} \\ 0, & \text{otherwise.} \end{cases}$$

Hence, **a** identifies which nodes are dangling nodes. For example, if $\mathbf{d} = (3, 8, 0, 4, 11, 0, 9, 0)$ then

```
a = [0, 0, 1, 0, 0, 1, 0, 1]
```

because the nodes $\{w_3, w_6, w_8\}$ are the dangling nodes. Then

$$\mathbf{X} = \tfrac{1}{n}\mathbf{e} \cdot \mathbf{a}^T$$

and therefore

$$\overline{\mathbf{H}} = \mathbf{H} + \tfrac{1}{n}\mathbf{e} \cdot \mathbf{a}^T \tag{7.1}$$

For example, for the hyperlink matrix $\mathbf{H}$ of the tiny network above

$$\overline{\mathbf{H}} = \begin{bmatrix} \frac{1}{6} & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & \frac{1}{3} & \frac{1}{2} & 0 & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & 0 & 0 & \frac{1}{2} & \frac{1}{6} \\ \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{2} & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{6} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{H}} + \underbrace{\begin{bmatrix} \frac{1}{6} & 0 & 0 & 0 & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 & 0 & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 & 0 & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 & 0 & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 & 0 & 0 & \frac{1}{6} \\ \frac{1}{6} & 0 & 0 & 0 & 0 & \frac{1}{6} \end{bmatrix}}_{\frac{1}{n}\mathbf{e}\cdot\mathbf{a}^T}$$

We finally create the **Google** matrix associated to the network:

$$\mathbf{G} = \alpha\overline{\mathbf{H}} + (1-\alpha)\tfrac{1}{n}\mathbf{J} \tag{7.2}$$

where $\alpha$ is a constant parameter such that $0 < \alpha < 1$ and $\mathbf{J}$ is the $n \times n$ all ones matrix (every entry in $\mathbf{J}$ is 1). We will choose the value $\alpha = 0.85$. From the definition (7.2) of the Google matrix, equation (7.1), and the fact that $\mathbf{J} = \mathbf{e} \cdot \mathbf{e}^T$ we have that

$$\begin{aligned} \mathbf{G} &= \alpha\overline{\mathbf{H}} + (1-\alpha)\tfrac{1}{n}\mathbf{J} \\ &= \alpha(\mathbf{H} + \tfrac{1}{n}\mathbf{e} \cdot \mathbf{a}^T) + (1-\alpha)\tfrac{1}{n}\mathbf{e} \cdot \mathbf{e}^T \\ &= \alpha\mathbf{H} + \tfrac{\alpha}{n}\mathbf{e} \cdot \mathbf{a}^T + (1-\alpha)\tfrac{1}{n}\mathbf{e} \cdot \mathbf{e}^T \\ &= \alpha\mathbf{H} + \mathbf{e} \cdot \left(\tfrac{\alpha}{n}\mathbf{a}^T + (1-\alpha)\tfrac{1}{n}\mathbf{e}^T\right) \end{aligned}$$

Thus, given any vector $\mathbf{v} \in \mathbb{R}^n$ we have

$$\mathbf{G}\mathbf{v} = \alpha\mathbf{H}\mathbf{v} + \left(\tfrac{\alpha}{n}\mathbf{a}^T\mathbf{v} + (1-\alpha)\tfrac{1}{n}\mathbf{e}^T\mathbf{v}\right)\mathbf{e} \tag{7.3}$$

Notice that the coefficient of $\mathbf{e}$ in (7.3) is the scalar

$$\beta(\mathbf{v}) = \left(\tfrac{\alpha}{n}\mathbf{a}^T\mathbf{v} + (1-\alpha)\tfrac{1}{n}\mathbf{e}^T\mathbf{v}\right)$$

and thus

$$\mathbf{Gv} = \alpha\mathbf{Hv} + \beta(\mathbf{v})\mathbf{e}.$$

The above decomposition of $\mathbf{G}$ makes computing $\mathbf{Gv}$ very efficient. Specifically, to compute $\mathbf{Gv}$ all we need to compute is the vector $\mathbf{Hv}$, and the two scalar quantities $\mathbf{a}^T\mathbf{v}$ and $\mathbf{e}^T\mathbf{v}$ appearing in $\beta$. In Python the latter computations can be done as follows:

$$\mathbf{a} = \text{np.where}(\ \mathbf{d} < 1)[0]$$

$$\mathbf{a}^T\mathbf{v} = \text{np.sum}(\ \mathbf{v}[\ \mathbf{a}\ ]\ )$$

$$\mathbf{e}^T\mathbf{v} = \text{np.sum}(\ \mathbf{v}\ )$$

So, what properties does the Google matrix satisfy? The ones we are interested in are below and can be proved using a well-known result in matrix analysis called the Perron-Frobenius theorem.

**Theorem 7.1: PageRank**

For the Google matrix $\mathbf{G}$ the following hold:

(i) The dominant eigenvalue of $\mathbf{G}$ is $\lambda_1 = 1$ is and it is a simple eigenvalue.

(ii) There is a unique positive vector $\mathbf{v}^*$ such that $\mathbf{Gv}^* = \mathbf{v}^*$ and the entries of $\mathbf{v}^*$ sum to one.

(iii) For any vector $\mathbf{w}$ whose entries sum to one it holds that

$$\lim_{k\to\infty} \mathbf{G}^k\mathbf{w} = \mathbf{v}^*$$

The vector $\mathbf{v}^*$ in the PageRank theorem is known as the **PageRank vector** of $\mathbf{G}$ and the PageRank theorem states that we can perform the power method with $\mathbf{G}$ that will generate a sequence converging to $\mathbf{v}^*$. Below we present pseudocode implementing the PageRank algorithm which, in addition to returning the PageRank vector $\mathbf{v}^* = (v_1^*, v_2^*, \ldots, v_n^*)$, returns the **PageRank indices** $I = (i_1, i_2, i_3, \ldots, i_n)$ such that

$$v_{i_1}^* \geq v_{i_2}^* \geq v_{i_3}^* \geq \cdots \geq v_{i_n}^*.$$

The ordering $I = (i_1, i_2, \ldots, i_n)$ is used to rank the importance of the vertices in the network. Specifically, the larger the numerical value of $v_j^*$ the larger the ranking of vertex $w_j$. We define the **PageRank** of vertex $w_j$ in the network as the integer $r_j \in \{1, 2, \ldots, n\}$ such that $i_{r_j} = j$. In other words, $r_j$ is the location of $j$ in the list $I = (i_1, i_2, \ldots, i_n)$.

**Example 7.3.** For the tiny network in Figure 7.1, performing the PageRank algorithm with tolerance $\varepsilon = 1 \times 10^{-12}$, after $k = 31$ iterations the PageRank vector is computed as

$$\mathbf{v}^* = \begin{pmatrix} 0.11527 & 0.20696 & 0.18139 & 0.23278 & 0.15557 & 0.10803 \end{pmatrix}$$

The PageRank index is therefore $I = (4, 2, 3, 5, 1, 6)$. The rankings are therefore $\mathbf{r} = (5, 2, 3, 1, 4, 6)$. Note that in Python, $\mathbf{r} =$ np.argsort(I).

Below is the pseudocode for the PageRank algorithm.

**Algorithm 7.3** PageRank

INPUT: $\mathbf{A}$, $\varepsilon > 0$, $N_{\max}$

OUTPUT: Approximate PageRank vector of Google matrix and PageRank index $I$

**create** degree vector $\mathbf{d}$
**create** vector $\mathbf{a}$ containing the indices of the dangling nodes
**create** diagonal matrix $\mathbf{D}$ such that $\mathbf{D}[i,i] = \frac{1}{\mathbf{d}[i]}$ if $\mathbf{d}[i] \neq 0$
    and $\mathbf{D}[i,i] = 0$ if $\mathbf{d}[i] = 0$
$\mathbf{H} = \alpha\mathbf{A}\mathbf{D}$ (this is technically not $\mathbf{H}$)
$\mathbf{v}_{\text{old}} = \frac{1}{n}\mathbf{e}$
$\beta = \frac{\alpha}{n}\mathbf{a}^T\mathbf{v}_{\text{old}} + (1-\alpha)\frac{1}{n}\mathbf{e}^T\mathbf{v}_{\text{old}}$
$\mathbf{v} = \mathbf{H}\mathbf{v}_{\text{old}} + \beta\mathbf{e}$
$k = 0$
**while** $\|\mathbf{v} - \mathbf{v}_{\text{old}}\|_2 > \varepsilon$ **and** $k < N_{\max}$
    $\mathbf{v}_{\text{old}} = \mathbf{v}$
    $\beta = \frac{\alpha}{n}\mathbf{a}^T\mathbf{v}_{\text{old}} + (1-\alpha)\frac{1}{n}\mathbf{e}^T\mathbf{v}_{\text{old}}$
    $\mathbf{v} = \mathbf{H}\mathbf{v} + \beta\mathbf{e}$
    $k = k + 1$
**end**
**find** PageRank index set $I = (i_1, i_2, \ldots, i_n)$
**return** $\mathbf{v}$, $I$

## 7.4 Singular Value Decomposition

The singular value decomposition of a matrix is closely related to the eigenvector-eigenvalue decomposition of a symmetric matrix. Recall that if $\mathbf{A}$ is a $n \times n$ symmetric matrix then there exists an orthogonal matrix $\mathbf{V}$ (meaning that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$) and a diagonal matrix $\mathbf{D}$ such that

$$\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T.$$

The columns of $\mathbf{V} = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n\end{bmatrix}$ are orthonormal eigenvectors of $\mathbf{A}$ and the entries of $\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ are the corresponding eigenvalues of $\mathbf{A}$, that is $\mathbf{A}\mathbf{v}_i = \lambda_i\mathbf{v}$ for

$i = 1, 2, \ldots, n$. The decomposition $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$ is called the **spectral decomposition** of $\mathbf{A}$. The decomposition $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^T$ can also be written as

$$\begin{aligned}\mathbf{A} &= \mathbf{V}\mathbf{D}\mathbf{V}^T \\ &= \begin{bmatrix}\lambda_1\mathbf{v}_1 & \lambda_2\mathbf{v}_2 & \cdots & \lambda_n\mathbf{v}_n\end{bmatrix}\mathbf{V}^T \\ &= \lambda_1\mathbf{v}_1\mathbf{v}_1^T + \lambda_2\mathbf{v}_2\mathbf{v}_2^T + \cdots + \lambda_n\mathbf{v}_n\mathbf{v}_n^T\end{aligned}$$

Hence, $\mathbf{A}$ is the sum of the spectral components $\lambda_i\mathbf{v}_i\mathbf{v}_i^T$. This expansion is called the *outer product expansion*.

Geometrically, orthogonal matrices are generalizations of rotation and reflection matrices on $\mathbb{R}^2$. This is because orthogonal matrices preserve the Euclidean norm of a vector:

$$\|\mathbf{V}\mathbf{x}\|_2 = \sqrt{(\mathbf{V}\mathbf{x})^T\mathbf{V}\mathbf{x}} = \sqrt{\mathbf{x}^T\mathbf{V}^T\mathbf{V}\mathbf{x}} = \sqrt{\mathbf{x}^T\mathbf{x}} = \|\mathbf{x}\|_2.$$

With this interpretation of orthogonal matrices, the spectral decomposition factors a symmetric matrix $\mathbf{A}$ as a series of three transformations; first a rotation/reflection $\mathbf{V}^T$, second a scaling $\mathbf{D}$, and third a rotation/reflection $\mathbf{V}$.

When $\mathbf{A}$ is not symmetric or (even worse) not a square matrix, say that $\mathbf{A}$ is $m \times n$, then $\mathbf{A}$ does not generally have a spectral decomposition, and when it is not square it does not even have eigenvalues/eigenvectors. However, the singular value decomposition is rooted in the idea of decomposing $\mathbf{A}$ as a sequence of a reflection/rotation, then a scaling, and finally another reflection/rotation. Finding such a decomposition is equivalent to finding an orthonormal basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ of $\mathbb{R}^n$, an orthonormal basis $\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m$ of $\mathbb{R}^m$, and positive constants $\sigma_1, \sigma_2, \ldots, \sigma_r$ such that $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ for $i = 1, 2, \ldots, r$, where $r = \text{rank}(\mathbf{A}) \leq \min\{n, m\}$. Having found such data, then

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where $\mathbf{U} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m\end{bmatrix}$, $\mathbf{V} = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_m\end{bmatrix}$, and $\mathbf{\Sigma}$ is a $m \times n$ diagonal matrix whose first $k$ diagonal entries are $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ and all other entries are zero. This is called the **singular value decomposition** (SVD) of $\mathbf{A}$. Before we prove that *any* matrix has a SVD, we need the following.

**Lemma 7.2**

The matrices $\mathbf{A}$ and $\mathbf{A}^T\mathbf{A}$ have the same nullspace and consequently $\text{rank}(\mathbf{A}) = \text{rank}(\mathbf{A}^T\mathbf{A})$ and $\text{nullity}(\mathbf{A}) = \text{nullity}(\mathbf{A}^T\mathbf{A})$.

*Proof.* The nullspace of a matrix $\mathbf{M}$ will be denoted by $\ker(\mathbf{M})$. Suppose that $\mathbf{v} \in \ker(\mathbf{A})$. Then from $\mathbf{A}\mathbf{v} = \mathbf{0}$ we have $\mathbf{A}^T\mathbf{A}\mathbf{v} = \mathbf{0}$ and thus $\mathbf{v} \in \ker(\mathbf{A}^T\mathbf{A})$. Now suppose that

$\mathbf{v} \in \ker(\mathbf{A}^T\mathbf{A})$. Then from $\mathbf{A}^T\mathbf{A}\mathbf{v} = \mathbf{0}$ we have $\mathbf{v}^T\mathbf{A}^T\mathbf{A}\mathbf{v} = 0$ and thus $\|\mathbf{A}\mathbf{v}\|_2^2 = 0$. Therefore, $\mathbf{A}\mathbf{v} = \mathbf{0}$ and this proves that $\mathbf{v} \in \ker(\mathbf{A})$. This proves that $\ker(\mathbf{A}) = \ker(\mathbf{A}^T\mathbf{A})$. The rank and nullity relationships follow from the Rank theorem. □

We now prove that any matrix has a singular value decomposition.

**Theorem 7.3: Singular Value Decomposition**

For any $m \times n$ matrix $\mathbf{A}$, there exists an $m \times m$ orthogonal matrix $\mathbf{U}$, an $n \times n$ orthogonal matrix $\mathbf{V}$, and a $m \times n$ diagonal matrix $\mathbf{\Sigma}$ such that

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

*Proof.* The matrix $\mathbf{A}^T\mathbf{A}$ is a $n \times n$ symmetric matrix and therefore there exists an orthonormal basis $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ of $\mathbb{R}^n$ consisting of eigenvectors of $\mathbf{A}^T\mathbf{A}$. The matrix $\mathbf{A}^T\mathbf{A}$ is positive semi-definite and thus its eigenvalues are all non-negative. Label these eigenvalues as $\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2$ in non-increasing order and consistent with the ordering of the eigenvectors, that is, $(\mathbf{A}^T\mathbf{A})\mathbf{v}_i = \sigma_i^2\mathbf{v}_i$. Let $r \leq n$ be such that $\sigma_r^2 > 0$ and $\sigma_{r+1}^2 = \cdots = \sigma_n^2 = 0$. Then $(\mathbf{A}^T\mathbf{A})\mathbf{v}_i = \mathbf{0}$ for $i = r+1, \ldots, n$ and thus $r = \operatorname{rank}(\mathbf{A}^T\mathbf{A}) = \operatorname{rank}(\mathbf{A}) \leq \min\{m, n\}$. Let $\mathbf{V} = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n\end{bmatrix}$ and note that $\mathbf{V}$ is orthogonal. Now define, for $i = 1, 2, \ldots, r$, the vector

$$\mathbf{u}_i = \frac{1}{\sigma_i}\mathbf{A}\mathbf{v}_i.$$

Then for $1 \leq i, j \leq r$, and using the fact that $\mathbf{v}_1, \ldots, \mathbf{v}_n$ are orthonormal, we have

$$\mathbf{u}_i^T\mathbf{u}_j = (\sigma_i^{-1}\sigma_j^{-1})\mathbf{v}_i^T\mathbf{A}^T\mathbf{A}\mathbf{v}_j = (\sigma_i^{-1}\sigma_j^{-1})\mathbf{v}_i^T(\sigma_j^2\mathbf{v}_j) = \frac{\sigma_j^2}{\sigma_i\sigma_j}\mathbf{v}_i^T\mathbf{v}_j = \begin{cases}1, & \text{if } i = j\\ 0, & \text{if } i \neq j\end{cases}$$

Therefore, $\mathbf{u}_1, \ldots, \mathbf{u}_r$ are orthonormal. We then extend the set $\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ to an orthonormal basis $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\}$ of $\mathbb{R}^m$ (say using the Gram-Schmidt procedure). Let $\mathbf{U} = \begin{bmatrix}\mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_m\end{bmatrix}$. Then,

$$\begin{aligned}\mathbf{A}\mathbf{V} &= \begin{bmatrix}\mathbf{A}\mathbf{v}_1 & \mathbf{A}\mathbf{v}_2 & \cdots & \mathbf{A}\mathbf{v}_r & \mathbf{A}\mathbf{v}_{r+1} & \cdots & \mathbf{A}\mathbf{v}_n\end{bmatrix}\\ &= \begin{bmatrix}\sigma_1\mathbf{u}_1 & \sigma_2\mathbf{u}_2 & \cdots & \sigma_r\mathbf{u}_r & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}\\ &= \mathbf{U}\mathbf{\Sigma}\end{aligned}$$

and this completes the proof. □

The numbers $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ are the **singular values** of $\mathbf{A}$, and as the proof above showed, they are found by taking the (positive) square roots of the non-zero eigenvalues of $\mathbf{A}^T\mathbf{A}$. The proof above also showed the following.

**Corollary 7.4**

The number of singular values of $\mathbf{A}$ is the rank of $\mathbf{A}$. In other words, the number of non-zero eigenvalues of $\mathbf{A}^T\mathbf{A}$ is the rank of $\mathbf{A}$.

Although we will not spend much time computing by hand the SVD of specific matrices, we make the following observation on computing the matrix $\mathbf{U}$. The vectors $\{\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m\}$ are eigenvectors of the matrix $\mathbf{A}\mathbf{A}^T$ since

$$\mathbf{A}\mathbf{A}^T = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)(\mathbf{V}\mathbf{\Sigma}^T\mathbf{U}^T) = \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^T\mathbf{U}^T$$

For this we also deduce that the non-zero eigenvalues of $\mathbf{A}\mathbf{A}^T$ are $\sigma_1^2 \geq \cdots \geq \sigma_r^2$ (the same as those of $\mathbf{A}^T\mathbf{A}$). Moreover, by definition $\mathbf{u}_i = \frac{1}{\sigma_i}\mathbf{A}\mathbf{v}_i$ for $i = 1, 2, \ldots, r$, and therefore $\mathbf{A}\mathbf{v}_i$ are eigenvectors of $\mathbf{A}\mathbf{A}^T$ aligned with $\mathbf{u}_i$ since $\sigma_i > 0$. Hence, an alternative method to compute $\mathbf{U}$ is to determine an orthonormal set of eigenvectors of $\mathbf{A}\mathbf{A}^T$, say $\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2, \ldots, \tilde{\mathbf{u}}_m$, with corresponding eigenvalues $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_r^2 > \sigma_{r+1}^2 = \cdots = \sigma_m^2 = 0$. Then, for $i = 1, 2, \ldots, r$, if $\langle \tilde{\mathbf{u}}_i, \mathbf{A}\mathbf{v}_i \rangle > 0$ then set $\mathbf{u}_i = \tilde{\mathbf{u}}_i$ otherwise set $\mathbf{u}_i = -\tilde{\mathbf{u}}_i$, and for $i > r$ set $\mathbf{u}_i = \tilde{\mathbf{u}}_i$.

**Example 7.4.** Let $\mathbf{A} = \begin{pmatrix} 1 & \frac{3}{2} \\ 0 & 1 \end{pmatrix}$. Then $\mathbf{A}$ is not symmetric and also not diagonalizable. Compute

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 1 & \frac{3}{2} \\ \frac{3}{2} & \frac{13}{4} \end{pmatrix}$$

One finds that the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are $\sigma_1^2 = 4$ and $\sigma_2^2 = 1/4$, with corresponding orthonormal eigenvectors $\mathbf{v}_1 = \frac{1}{\sqrt{5}}\begin{pmatrix} 1 \\ 2 \end{pmatrix}$ and $\mathbf{v}_2 = \frac{1}{\sqrt{5}}\begin{pmatrix} -2 \\ 1 \end{pmatrix}$. Then one computes

$$\mathbf{u}_1 = \frac{1}{\sigma_1}\mathbf{A}\mathbf{v}_1 = \frac{1}{\sqrt{5}}\begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

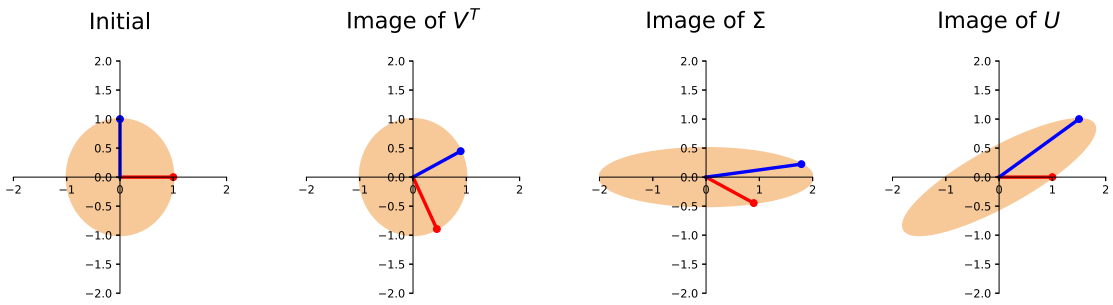
and

$$\mathbf{u}_2 = \frac{1}{\sigma_2}\mathbf{A}\mathbf{v}_2 = \frac{1}{\sqrt{5}}\begin{pmatrix} -1 \\ 2 \end{pmatrix}$$

Therefore, the SVD of $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \begin{pmatrix} \frac{2}{\sqrt{5}} & -\frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \end{pmatrix}\begin{pmatrix} 2 & 0 \\ 0 & \frac{1}{2} \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} \\ -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \end{pmatrix}$$

Below is a figure illustration the sequence of linear transformations $\mathbf{V}^T$, $\mathbf{\Sigma}$, and $\mathbf{V}$:




**Example 7.5.** Let

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

Then

$$\mathbf{A}\mathbf{A}^T = \begin{pmatrix} 2 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 2 \end{pmatrix}$$

and

$$\mathbf{A}^T\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

The eigenvalues of $\mathbf{A}\mathbf{A}^T$ are $\sigma_1^2 = 5$, $\sigma_2^2 = 2$, $\sigma_3^2 = 1$, $\sigma_4^2 = 0$, and $\sigma_5^2 = 0$, and the eigenvalues of $\mathbf{A}^T\mathbf{A}$ are $\sigma_1^2 = 5$, $\sigma_2^2 = 2$, $\sigma_3^2 = 1$. Hence,

$$\mathbf{\Sigma} = \begin{pmatrix} \sqrt{5} & 0 & 0 \\ 0 & \sqrt{2} & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

An orthogonal eigenvector matrix for $\mathbf{A}^T\mathbf{A}$ is

$$\mathbf{V} = \begin{pmatrix} \frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{6}}{3} & -\frac{\sqrt{3}}{3} & 0 \\ \frac{\sqrt{6}}{6} & \frac{\sqrt{3}}{3} & \frac{\sqrt{2}}{2} \end{pmatrix}$$

and an orthogonal eigenvector matrix for $\mathbf{A}\mathbf{A}^T$ is

$$\mathbf{U} = \begin{pmatrix} \frac{\sqrt{30}}{15} & \frac{\sqrt{6}}{3} & 0 & -\frac{\sqrt{7}}{7} & \frac{\sqrt{70}}{35} \\ \frac{\sqrt{30}}{15} & -\frac{\sqrt{6}}{6} & 0 & -\frac{2\sqrt{7}}{7} & -\frac{3\sqrt{70}}{70} \\ \frac{\sqrt{30}}{10} & 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{7}}{7} & -\frac{\sqrt{70}}{35} \\ \frac{\sqrt{30}}{15} & -\frac{\sqrt{6}}{6} & 0 & 0 & \frac{\sqrt{70}}{10} \\ \frac{\sqrt{30}}{10} & 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{7}}{7} & -\frac{\sqrt{70}}{35} \end{pmatrix}$$

One can verify that $\langle \mathbf{u}_i, \mathbf{A}\mathbf{v}_i \rangle > 0$ for $i = 1, 2, 3$. Therefore,

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T.$$

We make the following observation that is a by-product of the SVD and can be used to compute the a basis for the nullspace and range of a matrix. Let $\mathbf{A}$ have singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$ and suppose that $\mathbf{A}$ has rank $r$. Then by construction of the SVD, we have that $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}$ for $i = 1, 2, \ldots, r$ and $\mathbf{A}\mathbf{v}_j = \mathbf{0}$ for $j = r+1, \ldots, n$. Therefore, the set $\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$ is a basis for the range of $\mathbf{A}$ and $\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}$ is a basis for the nullspace of $\mathbf{A}$:

$$\text{nullspace}(\mathbf{A}) = \text{span}\{\mathbf{v}_{r+1}, \ldots, \mathbf{v}_n\}$$

$$\text{range}(\mathbf{A}) = \text{span}\{\mathbf{u}_1, \ldots, \mathbf{u}_r\}$$

## 7.5 SVD and Image Processing

We now consider an application of the SVD to image processing. The SVD when expanded can be written as

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T. \tag{7.4}$$

Recall that the singular values are ordered from largest to smallest, that is, $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. If for some $k < r$ the singular values $\sigma_{k+1}, \ldots, \sigma_r$ are small relative to $\sigma_1, \sigma_2, \ldots, \sigma_k$, then we can approximate $\mathbf{A}$ with

$$\mathbf{A} \approx \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_r \mathbf{u}_r \mathbf{v}_k^T.$$

Although we have discarded the terms from $k+1, \ldots, r$, the approximation might be reasonably good if the discarded terms are indeed small. To quantify "small" in a reasonable way, consider

$$\omega_j = \frac{\sigma_j}{\sum_{i=1}^{r} \sigma_i}.$$

Hence, $\omega_j$ is the contribution of the singular value $\sigma_j$ to the overall sum $\sum \sigma_i$. If $\omega_j < \alpha$ for some chosen tolerance $0 < \alpha < 1$, then we could declare that $\sigma_j$ is small enough that we can discard $\sigma_j \mathbf{u}_j \mathbf{v}_j^T$ from the expansion (7.4). Since the singular values are ordered from largest to smallest, we find the smallest $k$ such that $\omega_k > \alpha$ and discard $\sigma_{k+1}, \ldots, \sigma_r$.

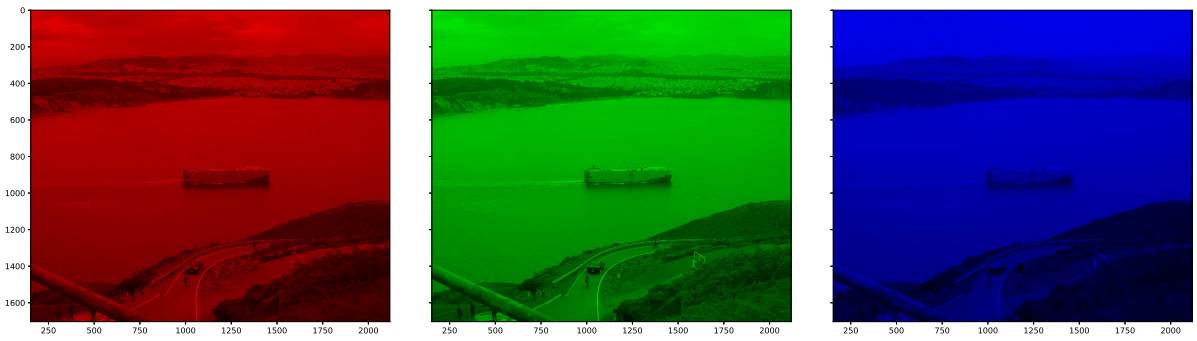
To see how the above approximation technique can be used, consider an image containing $m \times n$ pixels. Each pixel $p_{i,j}$ in the image has a RED $R_{i,j}$, GREEN $G_{i,j}$, and BLUE $B_{i,j}$ numerical intensity value each ranging between 0 and 1, that is, $0 \leq R_{i,j} \leq 1$, $0 \leq G_{i,j} \leq 1$, and $0 \leq B_{i,j} \leq 1$. If $R_{i,j} = 0$ then the color black is assigned and if $R_{i,j} = 1$ then the red color is assigned, and similarly with $G_{i,j}$ and $B_{i,j}$ with red replaced with green and blue, respectively. The resulting color at pixel $p_{i,j}$ is the superposition of the three RGB colors: $p_{i,j} = R_{i,j} + G_{i,j} + B_{i,j}$. Hence, an image $\mathbf{P}$ is the superposition (the sum) of three $m \times n$ matrices $\mathbf{R}$, $\mathbf{G}$, and $\mathbf{B}$:

$$\mathbf{P} = \mathbf{R} + \mathbf{G} + \mathbf{B}$$

As an example, consider the image below of a cargo ship leaving the San Francisco bay:


The image is the superposition of the following R, G, B images:
In this case, the image has $m \times n = 1704 \times 2272$ pixels. For each $\mathbf{R}$, $\mathbf{G}$, and $\mathbf{B}$ matrix, we compute the singular value decomposition and keep only the terms in the expansion (7.4) such that $\omega_j > \alpha$ for various values of $\alpha$ and we display the results in Figure 7.2.

Consider the case of $\alpha = 0.0011$. In all cases of the matrices $\mathbf{R}, \mathbf{G}$, and $\mathbf{B}$, the number of singular values is $r = m = 1704$, that is, all matrices $\mathbf{R}, \mathbf{G}$, and $\mathbf{B}$ have maximum rank.



The amount of data points needed to reconstruct the image is $m \times n \times 3 = 11,614,464$. For the value of $\alpha = 0.0011$, one computes that the number of singular values $\sigma_j$ such that $\omega_j > \alpha$ is $k_R = 109$ for $\mathbf{R}$, $k_G = 105$ for $\mathbf{G}$, and $k_B = 83$ for $\mathbf{B}$. Hence, the number of data points kept in the SVD expansion of $\mathbf{R}$ is $(m+n) \times k_R = 433,384$, for $\mathbf{G}$ it is $(m+n) \times k_G = 471,480$, and for $\mathbf{B}$ it is $(m+n) \times k_B = 330,008$. Hence, the approximation requires $(m+n) \times (k_R + k_G + k_B) = 1,180,872$ data points. This results in a **compression ratio** of

$$\kappa = \frac{m \times n \times 3}{(m+n) \times (k_R + k_G + k_B)} = \frac{11,614,464}{1,180,872} \approx 10$$

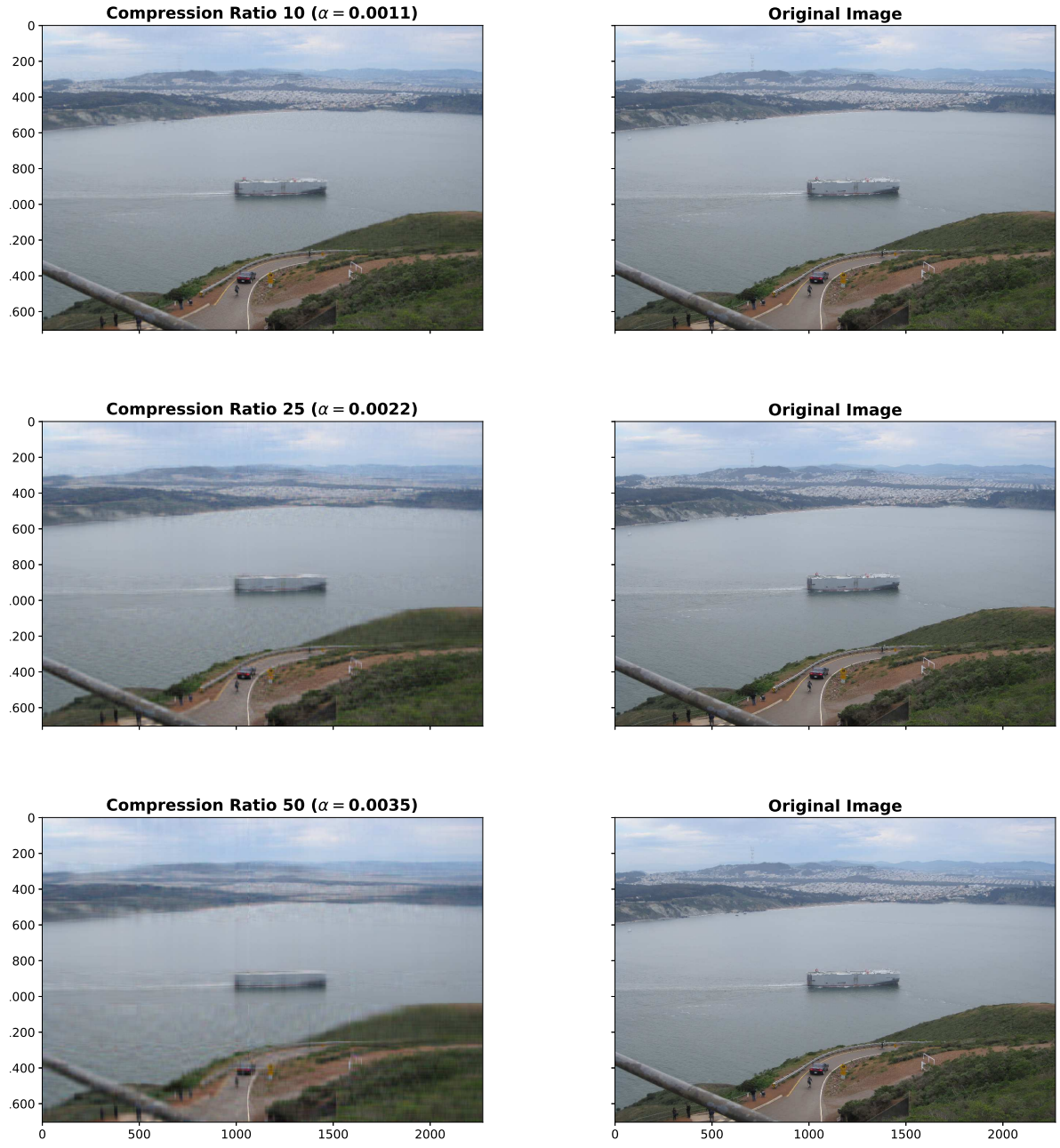



Figure 7.2: Approximation of image for various values of $\alpha$; as $\alpha$ increases we demand that the weight $\omega_j$ of the singular value $\sigma_j$ contribute more to the total sum $\sum \sigma_i$ and thus less of the singular values will be included in the approximation

# 8

# Numerical Solutions to Ordinary Differential Equations

## 8.1 Ordinary differential equations

A one-dimensional **ordinary differential equation** (ODE) is an equation of the form

$$\frac{dx}{dt} = f(x(t)) \tag{8.1}$$

where $x(\cdot)$ and $f(\cdot)$ is are real-valued functions of a single variable. The function $f$ is given and the problem is to find $x(t)$ satisfying (8.1); such a function is called a **solution** or **trajectory** of (8.1). In the theory of ODEs, the Newtonian notation for differentiation $\dot{x} := \frac{dx}{dt}$ is more widely used, and we will follow this convention. The ODE (8.1) is said to be **linear** if $f$ is a linear function, that is, $f(x) = ax$ for some $a \in \mathbb{R}$, and is said to be **nonlinear** otherwise.

**Example 8.1.** Show that $x(t) = \frac{1}{1-t}$ is a solution to the differential equation

$$\dot{x} = x^2.$$

*Solution.* Here $f(x) = x^2$. Now, if $x(t) = \frac{1}{1-t}$ then

$$\dot{x} = (-1)(1-t)^{-2}(-1) = \left(\frac{1}{1-t}\right)^2 = f(x(t)).$$

Notice that the function $x(t) = \frac{1}{1-t}$ is not defined at $t = 1$. □

**Example 8.2.** Show that $x(t) = \tan(t + c)$, where $c \in \mathbb{R}$ is an arbitrary constant, is a solution to the nonlinear ODE

$$\dot{x} = 1 + x^2.$$

*Solution.* Here $f(x) = 1 + x^2$, which is indeed nonlinear. We compute

$$\dot{x} = \frac{d}{dt}(\tan(t+c)) = \sec^2(t+c) = 1 + \tan^2(t+c) = 1 + x(t)^2 = f(x(t)).$$

This shows that $x(t) = \tan(t+c)$ is a solution to the ODE. □

In the previous example, the constant $c$ is a **constant of integration** and thus we have actually produced a family of solutions and not a particular one. We must therefore specify an additional condition on $x(t)$ in order to distinguish one particular solution of the ODE. This condition is commonly called an **initial condition** and is of the form

$$x(t_0) = x_0. \tag{8.2}$$

The ODE (8.1) with initial condition (8.2) is called an **initial value problem** (IVP).

**Example 8.3.** Consider the initial value problem

$$\dot{x} = 1 + x^2$$
$$x(0) = 1.$$

It has solution $x(t) = \tan(t + \pi/4)$ since $x(0) = \tan(\pi/4) = 1$. On the other hand, the initial value problem

$$\dot{x} = 1 + x^2$$
$$x(0) = 0$$

has solution $x(t) = \tan(t)$. Notice that for both cases, the solution $x(t)$ is defined only on a finite interval $(a, b)$ containing $t_0 = 0$; for $x(t) = \tan(t + \pi/4)$ it is defined for $(\frac{-3\pi}{4}, \frac{\pi}{4})$ and for $x(t) = \tan(t)$ it is defined for $(-\frac{\pi}{2}, \frac{\pi}{2})$. Hence, in general, a solution to a IVP may exist only on a finite interval $(a, b)$ and moreover, it is possible that as $t \to a$ and/or $t \to b$, $x(t)$ becomes unbounded in finite time.

**Example 8.4.** Consider the IVP

$$\dot{x} = 3x^{2/3}$$
$$x(0) = 0.$$

Here $f(x) = 3x^{2/3}$. The constant function $x(0) = 0$ clearly solves the IVP. On the other hand, the function $x(t) = t^3$ is also a solution since

$$\dot{x} = 3t^2 = 3(t^3)^{2/3} = f(x(t)).$$

This example shows that, in general, solutions to ODEs are not unique, even if $f(x)$ is continuous.

The previous examples show that even when $f$ is a well-behaved function, solutions to ODEs may not be so well-behaved globally. However, the next theorem states that under some fairly mild assumptions on $f$, solutions will indeed exist (for a short while, at the very least) and they will be unique.

**Theorem 8.1: Existence and Uniqueness**

Consider the initial value problem

$$\begin{aligned} \dot{x} &= f(x) \\ x(t_0) &= x_0 \end{aligned}$$

and suppose that $f$ and $f'$ are continuous on an open interval $I \subset \mathbb{R}$. If $x_0 \in I$ then there exists an interval $(a, b)$ containing $t_0$ and a unique solution $x : (a, b) \to \mathbb{R}$ to the initial value problem.

From now on, we implicitly assume that $f$ satisfies the conditions of Theorem 8.1 and thus consider only initial value problems that have a unique solution on some interval.

In most interesting cases, unlike Example 8.3 and 8.4, it is not possible to determine an explicit closed-form formula for the solution of an initial value problem, at least not using the familiar functions from calculus.

Now we consider general $n$-dimensional differential equations. To do this, we introduce the notion of a vector field and a curve in $\mathbb{R}^n$. A **vector field** $\mathbf{F}$ in $\mathbb{R}^n$ is a mapping $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ that assigns to each point $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathbb{R}$ a vector $\mathbf{F}(\mathbf{x}) \in \mathbb{R}^n$. In component form, a vector field $\mathbf{F}(\mathbf{x})$ can be written as

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} F_1(\mathbf{x}) \\ F_2(\mathbf{x}) \\ \vdots \\ F_n(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} F_1(x_1, x_2, \ldots, x_n) \\ F_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ F_n(x_1, x_2, \ldots, x_n) \end{bmatrix}.$$

An example of a vector field in $\mathbb{R}^2$ is

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} -x_1 + \sin(x_1 x_2) \\ x_2 + x_1^2 \end{bmatrix}$$

and one in $\mathbb{R}^3$ is

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_2 - x_1 \\ x_1 - x_2 - x_1 x_3 \\ x_1 x_2 - x_3 \end{bmatrix}.$$

In $\mathbb{R}^2$, we can visualize the behavior of a vector field by drawing at each $\mathbf{x} \in \mathbb{R}^2$ the vector $\mathbf{F}(\mathbf{x})$ with tail based at $\mathbf{x}$. An example of the type of plots obtained by doing this for a two-dimensional vector field is shown in Figure 8.1.

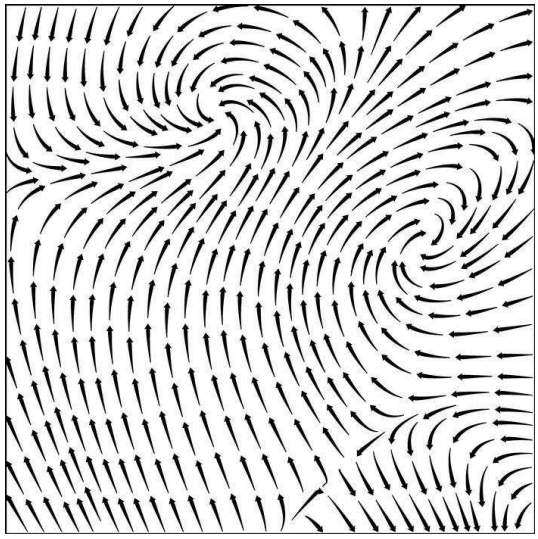


Figure 8.1: A vector field in $\mathbb{R}^2$

A **curve** in $\mathbb{R}^n$ is a function $\mathbf{x} : I \to \mathbb{R}^n$, where $I \subset \mathbb{R}$ is an interval. If in component form we have

$$\mathbf{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{bmatrix}$$

then the **derivative** of $\mathbf{x}(t)$ is

$$\frac{d\mathbf{x}}{dt} = \dot{\mathbf{x}}(t) = \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \vdots \\ \dot{x}_n(t) \end{bmatrix}$$

provided the individual derivatives

$$\dot{x}_i(t) = \lim_{h \to 0} \frac{x_i(t+h) - x_i(t)}{h}$$

exist. In this case, we say that $\mathbf{x}(t)$ is a **differentiable curve** on $I$. For example,

$$\mathbf{x}(t) = \begin{bmatrix} t^3 \sin(t) - 1 \\ \cos(t^2) + t^3 \\ e^{t^2} - 3t \end{bmatrix}$$

is a differentiable curve on $I = \mathbb{R}^3$ with derivative

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} 3t^2 \sin(t) + t^3 \cos(t) \\ -2t \sin(t^2) + 3t^2 \\ 2te^{t^2} - 3 \end{bmatrix}$$

An $n$-**dimensional ordinary differential equation** (ODE) is an equation of the form

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}(t)) \tag{8.3}$$

where $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ is an $n$-dimensional vector field and $\mathbf{x} : I \to \mathbb{R}^n$ is an unknown differentiable curve. An **initial value problem** is an ODE with an **initial condition**:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{F}(\mathbf{x}(t)) \\ \mathbf{x}(t_0) &= \mathbf{x}_0 \end{aligned}$$

where $\mathbf{x}_0 \in \mathbb{R}$ is a given fixed vector.

**Example 8.5.** Consider the initial value problem

$$\begin{aligned} \dot{x}_1 &= -x_1 - 3x_2 \\ \dot{x}_2 &= 2x_2 \\ \mathbf{x}(0) &= (3, 2). \end{aligned}$$

Here $\mathbf{F}(\mathbf{x}) = \begin{bmatrix} -x_1 - 3x_2 \\ 2x_2 \end{bmatrix}$. Show that the curve

$$\mathbf{x}(t) = \begin{bmatrix} 3e^{-t} + 2(e^{-t} - e^{2t}) \\ 2e^{2t} \end{bmatrix}$$

is a solution to the IVP.

*Solution.* First of all,

$$\mathbf{x}(0) = \begin{bmatrix} 3e^0 + 2(e^0 - e^0) \\ 2e^0 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Now,

$$\dot{\mathbf{x}} = \begin{bmatrix} -3e^{-t} + 2(-e^{-t} - 2e^{2t}) \\ 4e^{2t} \end{bmatrix} = \begin{bmatrix} -5e^{-t} - 4e^{2t} \\ 4e^{2t} \end{bmatrix} \tag{8.4}$$

On the other hand,

$$\begin{aligned} \mathbf{F}(\mathbf{x}(t)) &= \begin{bmatrix} -(3e^{-t} + 2(e^{-t} - e^{2t})) - 3(2e^{2t}) \\ 2(2e^{2t}) \end{bmatrix} \\ &= \begin{bmatrix} -5e^{-t} + 2e^{2t} - 6e^{2t} \\ 4e^{2t} \end{bmatrix} \\ &= \begin{bmatrix} -5e^{-t} - 4e^{2t} \\ 4e^{2t} \end{bmatrix} \end{aligned} \tag{8.5}$$

Comparing (8.4) and (8.5), we see that $\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}(t))$, and therefore $\mathbf{x}(t)$ solves the IVP. The curve $\mathbf{x}(t)$ is plotted in Figure 8.2. □

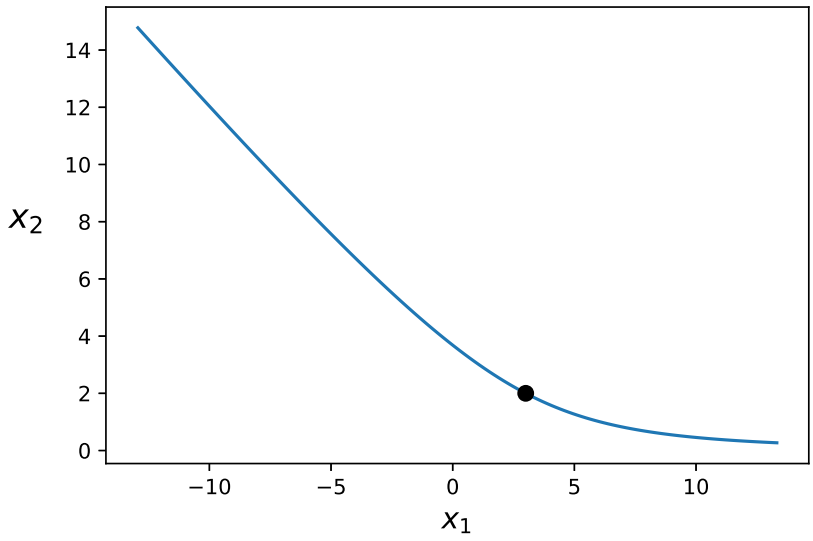



Figure 8.2: Solution of ODE in Example 8.5

**Example 8.6.** Verify that

$$\mathbf{x}(t) = \begin{bmatrix} e^{-t}(2\cos(2t) - 9\sin(2t)) \\ e^{-t}(4\cos(2t) - \sin(2t)) \end{bmatrix}$$

is a solution to the initial value problem

$$\begin{aligned} \dot{x}_1 &= -5x_2 \\ \dot{x}_2 &= x_1 - 2x_2 \\ \mathbf{x}(0) &= (2, 4) \end{aligned}$$

*Solution.* This is left as an exercise. □

To end this section, we now introduce the derivative of a vector field $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$.

**Definition 8.2**

Let $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ be a vector field with component functions $F_1(\mathbf{x}), F_2(\mathbf{x}), \ldots, F_n(\mathbf{x})$. We say that $\mathbf{F}$ is **continuously differentiable** if for all $i, j = 1, 2, \ldots, n$,

$$\frac{\partial F_i}{\partial x_j}(x_1, x_2, \ldots, x_n)$$

exist and are continuous on $\mathbb{R}^n$. In this case, the **derivative** of $\mathbf{F}$ at $\mathbf{p}$ is the $n \times n$

matrix

$$\mathbf{DF}(\mathbf{p}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial F_1}{\partial x_n} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix}$$

where all $\frac{\partial F_i}{\partial x_j}$ are evaluated at $\mathbf{p}$. The matrix $\mathbf{DF}$ is called the **Jacobian matrix** of $\mathbf{F}$.

**Example 8.7.** Find the derivative of the vector field $\mathbf{F} : \mathbb{R}^2 \to \mathbb{R}^2$ given by

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_1 \sin(3x_2) - x_1^3 x_2 \\ \ln(x_1 + 1) - x_1 e^{5x_2} \end{bmatrix}$$

at the point $\mathbf{p} = (1, 2)$.

*Solution.* By definition, for any $\mathbf{x} = (x_1, x_2)$:

$$\mathbf{DF}(\mathbf{x}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} \sin(3x_2) - 3x_1^2 x_2 & 3x_1 \cos(3x_2) - x_1^3 \\ \frac{1}{x_1+1} - e^{5x_2} & -5x_1 e^{x_2} \end{bmatrix}$$

Therefore,

$$\mathbf{DF}(\mathbf{p}) = \begin{bmatrix} \sin(6) - 9 & 3\cos(6) - 1 \\ \frac{1}{2} - e^5 & -5e^2 \end{bmatrix}$$

□

**Example 8.8.** Find the derivative $\mathbf{DF}(\mathbf{x})$ if $\mathbf{x} = (x_1, x_2, x_3)$ and

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_1 x_2 - x_2^3 x_3 - 7x_1 \\ x_3^2 - x_1^2 \\ x_1 x_3 - x_2 x_3 \end{bmatrix}$$

*Solution.* By definition,

$$\mathbf{DF}(\mathbf{x}) = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \frac{\partial F_1}{\partial x_3} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \frac{\partial F_2}{\partial x_3} \\ \frac{\partial F_3}{\partial x_1} & \frac{\partial F_3}{\partial x_2} & \frac{\partial F_3}{\partial x_3} \end{bmatrix} = \begin{bmatrix} x_2 - 7 & x_1 - 3x_2^2 x_3 & -x_2^2 \\ -2x_1 & 0 & 2x_3 \\ x_3 & -x_3 & x_1 - x_2 \end{bmatrix}.$$

□

In many models of physical phenomenon, there is variable of interest $x(t)$ (position, temperature, concentration, etc.) that is known to satisfy a high-order differential equation. For example, let $x(t)$ denote the position of a body of mass $m$ acted upon by a force $F$ in the direction of motion. Then Newton's 2nd law of motion states that

$$F = m\ddot{x}(t).$$

This equation is a 2-nd order differential equation for the variable $x(t)$. In practice, it is convenient to introduce a system of differentiable equations instead of directly solving a high-order equation. For this example, we introduce the vector $\mathbf{x} = (x_1, x_2)$ and let $x_1(t) = x(t)$ (the position of the body) and let $x_2(t) = \dot{x}(t)$ (the velocity of the body). Then

$$\begin{aligned}\dot{x}_1 &= x_2\\ \dot{x}_2 &= \frac{F}{m}\end{aligned}$$

The vector field associated to this 2-dimensional ODE is therefore

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_2 \\ \\ \frac{F}{m} \end{bmatrix}$$

If $\mathbf{x}(t) = (x_1(t), x_2(t)$ is a trajectory of the system of equations, then the position of the body is simply $x_1(t)$ by definition.

**Example 8.9.** Suppose that the function $w(t)$ satisfies the differential equation

$$2w^{(4)}(t) - \sin(\dot{w}) + 7\frac{w^{(3)}(t)}{1+\ddot{w}(t)} = 0$$

Write an equivalent 4-dimensional system of differential equations.

*Solution.* Let $x_1 = w$, let $x_2 = \dot{w}$, let $x_3 = \ddot{w}$, let $x_4 = w^{(3)}$, and let $\mathbf{x} = (x_1, x_2, x_3, x_4)$. Then

$$\begin{aligned}\dot{x}_1 &= \dot{w} = x_2\\ \dot{x}_2 &= \ddot{w} = x_3\\ \dot{x}_3 &= w^{(3)} = x_4\\ \dot{x}_4 &= \frac{1}{2}\left(\sin(\dot{w}) - 7\frac{w^{(3)}}{1+\ddot{w}}\right)\\ &= \frac{1}{2}\left(\sin(x_2) - 7\frac{x_4}{1+x_3}\right)\end{aligned}$$

is the equivalent 4-dimensional system of ODEs. The associated vector field is then

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} x_2 \\ x_3 \\ x_4 \\ \frac{1}{2}\left(\sin(x_2) - 7\frac{x_4}{1+x_3}\right) \end{bmatrix}$$

□

## 8.2 Euler's Method

Suppose that $x(t)$ is a solution of the one-dimensional ODE $\dot{x} = f(x(t))$ with initial condition $x(t_0) = x_0$, and assume that $x(t)$ is defined for $a \leq t \leq b$ where $a = t_0$. By Taylor's theorem, for $t \in [a, b]$ we have

$$x(t) = x(t_0) + \dot{x}(t_0)(t - t_0) + \frac{\ddot{x}(\xi(t_0))}{2!}(t - t_0)^2$$

where $\xi(t_0)$ is in between $t_0$ and $t$. Now, if $t$ is close to $t_0$, and in particular so that $|t - t_0| < 1$, then $|t - t_0|^2$ will be small compared to $|t - t_0|$. We therefore take as an approximation

$$x(t) \approx x(t_0) + \dot{x}(t_0)(t - t_0).$$

Since $\dot{x}(t_0) = f(x(t_0)) = f(x_0)$ we have

$$x(t) \approx x_0 + f(x_0)(t - t_0). \tag{8.6}$$

The function $\ell(t) = x_0 + f(x_0)(t - t_0)$ is the tangent line to the graph of $x(t)$ passing through the point $(t_0, x_0)$. The approximation (8.6) is the basis for **Euler's method** which we now describe.

Suppose that $x(t)$ is defined on $[a, b]$. Partition the interval $[a, b]$ into equally spaced points $a = t_0 < t_1 < \cdots < t_N = b$, which we call **mesh points**. The step-size is therefore $h = \frac{b-a}{N}$. Since $t_0 = a$ we have

$$t_i = t_{i-1} + h = a + ih$$

for $i = 1, \ldots, N$. At the mesh point $t_i$ we will approximate $x(t_i)$ with $w_i$ computed as follows. We first set $w_0 = x(t_0) = x_0$. To compute the approximation $w_1$ to $x(t_1)$ we use (8.6):

$$x(t_1) \approx x(t_0) + f(x_0)(t_1 - t_0) = w_0 + f(w_0)h$$

Hence, we set $w_1 = w_0 + f(w_0)h$. Now,

$$x(t_2) \approx x(t_1) + f(x(t_1))(t_2 - t_1) \approx w_1 + f(w_1)h$$

and thus we set $w_2 = w_1 + f(w_1)h$. Similarly,

$$x(t_3) \approx x(t_2) + f(x(t_2))(t_3 - t_2) \approx w_2 + f(w_2)h$$

and thus we set $w_3 = w_2 + f(w_2)h$. Continuing in this way, we obtain

$$w_i = w_{i-1} + f(w_{i-1})h$$

for $i = 1, \ldots, N$. At the end of this process, we obtain the points $(t_0, w_0), (t_1, w_1), \ldots, (t_N, w_N)$ that approximate the exact values $(t_0, x(t_0)), (t_1, x(t_1)), \ldots, (t_N, x(t_N))$.

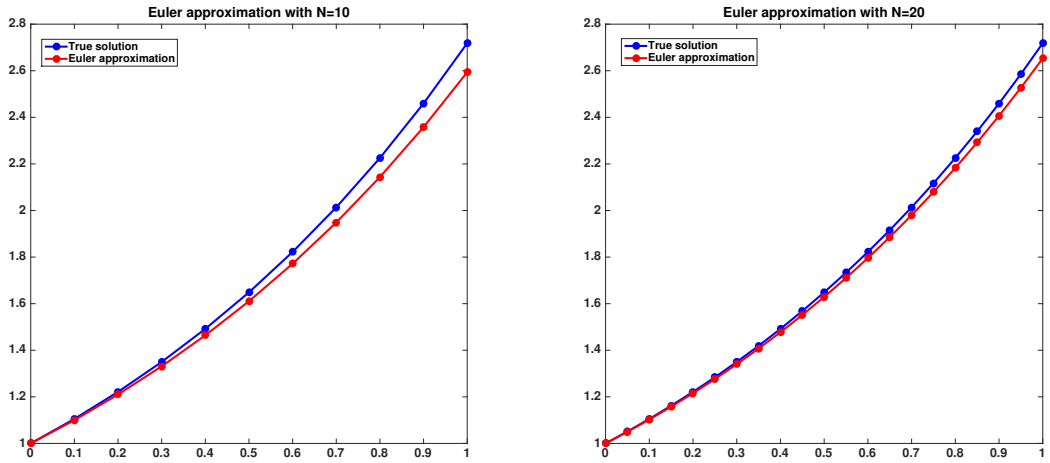



Figure 8.3: Euler approximation to the solution of $\dot{x} = x$, $x(0) = 1$ with $N \in \{10, 20\}$

**Example 8.10.** Consider the initial value problem

$$\dot{x} = x$$

$$x(0) = 1$$

The solution is $x(t) = e^t$. We apply Euler's method on the interval $[0, 1]$ with $N = 10$ and $N = 20$ and compute $w_0, w_1, \ldots, w_N$ for each case. We plot the true solution $x(t) = e^t$ and the approximate values $x(t_i) = w_i$ in Figure 8.3 for both cases of $N$.

Euler's method in the $n$-dimensional case is identical to the 1-dimensional case. Given the initial value problem

$$\dot{\mathbf{x}} = \mathbf{F}(\mathbf{x}(t))$$

$$\mathbf{x}(t_0) = \mathbf{x}_0$$

we seek to approximate the solution $\mathbf{x}(t)$, assumed to be defined on the interval $[a, b]$, at discrete points $a = t_0 < t_1 < \cdots < t_N = b$. As before, we consider the mesh points $t_i = t_{i-1} + h$, for $i = 1, 2, \ldots, N$, where $t_0 = a$, and seek to approximate $\mathbf{x}(t_i)$ with $\mathbf{w}_i$. We set $\mathbf{w}_0 = \mathbf{x}_0$ and define

$$\mathbf{w}_i = \mathbf{w}_{i-1} + \mathbf{F}(\mathbf{w}_{i-1})h$$

for $i = 1, 2, \ldots, N$, where $h = \frac{b-a}{N}$ is the step-size.

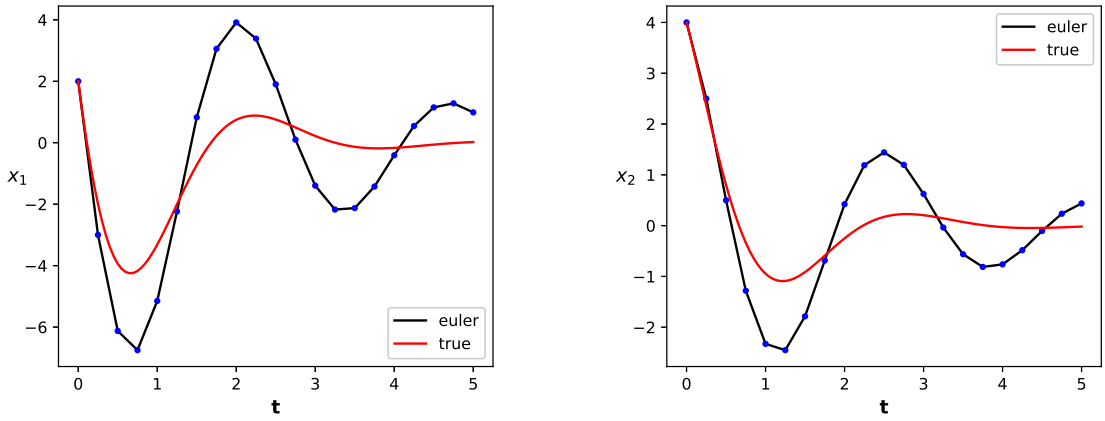



Figure 8.4: True solution and Euler approximation with $N = 20$

**Example 8.11.** Consider the initial value problem

$$\begin{aligned} \dot{x}_1 &= -5x_2 \\ \dot{x}_2 &= x_1 - 2x_2 \\ \mathbf{x}(0) &= (2, 4) \end{aligned}$$

As shown in Example 8.6, the solution is

$$\mathbf{x}(t) = \begin{bmatrix} e^{-t}(2\cos(2t) - 9\sin(2t)) \\ e^{-t}(4\cos(2t) - \sin(2t)) \end{bmatrix}.$$

Let $\mathbf{w}_0 = \mathbf{x}_0 = (2, 4)$ and compute $\mathbf{w}_1, \ldots, \mathbf{w}_N$, for $N = 20$. The results are shown in Figures 8.4. We also plot the trajectory $\mathbf{x}(t) = (x_1(t), x_2(t))$ and the approximation vectors $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_N$ on the $(x_1, x_2)$-plane in Figure 8.5.

We now analyze the error in using Euler's method. To that end, we need the following lemma.

**Lemma 8.3**

Let $\alpha, \beta > 0$ and let $(a_0, a_1, \ldots, a_k)$ be a sequence of numbers such that $a_0 \geq -\beta/\alpha$, and

$$a_{i+1} \leq (1 + \alpha)a_i + \beta$$

for each $i = 1, \ldots, k$. Then

$$a_i \leq e^{i\alpha}\left(a_0 + \frac{\beta}{\alpha}\right) - \frac{\beta}{\alpha}.$$

The following result describes the error between the estimates $w_i$ obtained using Euler's method and the true value $x(t_i)$.

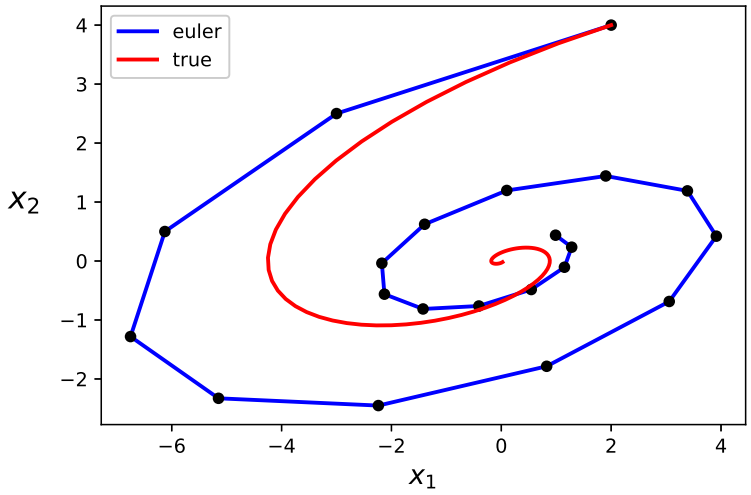



Figure 8.5: True solution and Euler approximation with $N = 20$ in the phase space

**Theorem 8.4**

Consider the initial value problem

$$\dot{x} = f(x)$$
$$x(a) = x_0,$$

and let $x : [a, b] \to \mathbb{R}$ denote its unique solution. Assume that $|x''(t)| \leq M$ for all $t \in [a, b]$. Suppose further that $f$ is Lipschitz on $\mathbb{R}$ with constant $L$, that is, $|f(x) - f(y)| \leq L|x - y|$ for all $x, y \in \mathbb{R}$. Let $w_0, w_1, \ldots, w_N$ denote the sequence obtained from Euler's method. Then, for each $i = 0, 1, \ldots, N$,

$$|x(t_i) - w_i| \leq \frac{hM}{2L}\left(e^{ihL} - 1\right).$$

and consequently

$$\max_{1 \leq i \leq N} |x(t_i) - w_i| \leq \frac{hM}{2L}(e^{(b-a)L} - 1).$$

*Proof.* From Taylor's theorem

$$x(t_{i+1}) = x(t_i) + hf(x(t_i)) + \frac{h^2}{2}x''(\xi_i)$$

for some $t_i < \xi_i < t_{i+1}$. For simplicity, let $x_i = x(t_i)$. Now, by definition,

$$w_{i+1} = w_i + hf(w_i)$$

Therefore,

$$|x_{i+1} - w_{i+1}| \leq |x_i - w_i| + h|f(x_i) - f(w_i)| + \frac{h^2}{2}|x''(\xi_i)|$$

$$= (1 + hL)|x_i - w_i| + \frac{h^2}{2}M$$

Applying Lemma 8.3 with $\alpha = hL$, $\beta = h^2M/2$, and $a_i = |x_i - w_i|$ we have

$$|x_{i+1} - w_{i+1}| \leq e^{(i+1)hL}\left(|x_0 - w_0| + \frac{h^2M}{2hL}\right) - \frac{h^2M}{2hL},$$

and since $w_0 = x_0$, we have

$$|x_{i+1} - w_{i+1}| \leq \frac{hM}{2L}\left(e^{(i+1)hL} - 1\right).$$

for $i = 0, 1, \ldots, N-1$. Note that since $ih \leq hN$ for $i = 1, \ldots, N$, and $b = a + hN$, we have

$$|x_i - w_i| \leq \frac{hM}{2L}(e^{(b-a)L} - 1).$$

and the second claim follows. □

Hence, the above inequality implies that the error in using Euler's method will decrease linearly as a function of the step-size $h$.

## 8.3 Taylor Methods

Euler's method is a special case of high-order Taylor methods. Consider the initial value problem

$$\dot{x} = f(x)$$
$$x(a) = x_0$$

and let $x(t)$ denote the unique solution on the interval $[a, b]$. Let $a = t_0 < t_1 < \cdots < t_N = b$ be equally spaced mesh-points in the interval $[a, b]$ and let $h = t_{i+1} - t_i$ be the step-size. Let $x_i = x(t_i)$ for $i = 0, 1, \ldots, N$. If $x(t)$ is $(k+1)$ continuously differentiable then by Taylor's theorem,

$$x(t_{i+1}) = x(t_i) + x'(t_i)h + \frac{h^2}{2!}x^{(2)}(t_i) + \frac{h^3}{3!}x^{(3)}(t_i) + \cdots + \frac{h^n}{k!}x^{(k)}(t_i) + \frac{h^{k+1}}{(k+1)!}x^{(k+1)}(\xi_i)$$

where $t_i < \xi_i < t_{i+1}$. The idea now is to expand the derivatives $x^{(j)}(t_i)$ appearing in the Taylor expansion all in terms of $f$ and the derivatives of $f$. For example, since $x'(t) = f(x(t))$, then by the chain rule

$$x^{(2)}(t) = f'(x(t))x'(t) = f'(x(t))f(x(t))$$

and similarly

$$x^{(3)}(t) = f''(x(t))x'(t)f(x(t))+f'(x(t))f'(x(t))x'(t) = f^{(2)}(x(t))(f(x(t))^2+(f'(x(t)))^2f(x(t)).$$

This, obviously, becomes very tedious and non-trivial as higher-order derivatives $x^{(j)}(t)$ are computed. However, this is the basis of the $n$th-order Taylor method. For example, the 1st order Taylor method is Euler's method:

$$w_{i+1} = w_i + hf(w_i).$$

The 2nd order Taylor method is

$$w_{i+1} = w_i + hf(w_i) + (f'(w_i)f(w_i))\,\frac{h^2}{2}$$

and the 3rd order Taylor method is

$$w_{i+1} = w_i + hf(w_i) + (f'(w_i)f(w_i))\frac{h^2}{2} + \left(f^{(2)}(w_i)f(w_i)^2 + f'(w_i)^2f(w_i)\right)\frac{h^3}{3!}$$

for $i = 0, 1, \ldots, N-1$, and where $w_0 = x_0$. In general, the **$k$th order Taylor method** is

$$w_{i+1} = w_i + hf(w_i) + \frac{d}{dt}[f(x(t))]\Big|_{x(t)=w_i}\frac{h^2}{2!} + \cdots + \frac{d^{k-1}}{dt^{k-1}}[f(x(t))]\Big|_{x(t)=w_i}\frac{h^k}{k!}$$

where

$$\frac{d^j}{dt^j}[f(x(t))]\Big|_{x(t)=w_i}$$

is obtained by computing the $j$th derivative of $f(x(t))$ using the chain rule and then substituting $w_i$ for $x(t)$. The main disadvantage with high-order Taylor methods is that one needs to differentiate $f(x(t))$, and for this reason Taylor methods are hardly used in practice. However, one expects that high-order Taylor methods are more accurate than Euler's method.

**Example 8.12.** Consider the IVP

$$\dot{x} = x$$
$$x(0) = \tfrac{1}{2}$$

the solution of which is $x(t) = \frac{1}{2}e^t$. In this case, $f(x) = x$ and thus $f'(x) = 1$ and $f^{(2)}(x) = 0$. Hence, the 3rd order Taylor method is

$$w_{i+1} = w_i + hf(w_i) + (f'(w_i)f(w_i))\frac{h^2}{2!} + \left(f^{(2)}(w_i)f(w_i)^2 + f'(w_i)^2f(w_i)\right)\frac{h^3}{3!}$$

$$= w_i + hw_i + w_i\frac{h^2}{2} + w_i\frac{h^3}{6}$$

| $t_i$ | Euler | 3rd Taylor |
|---|---|---|
| 0.00 | 0.0000e+00 | 0.0000e+00 |
| 0.20 | 1.0701e-02 | 3.4712e-05 |
| 0.40 | 2.5912e-02 | 8.4793e-05 |
| 0.60 | 4.7059e-02 | 1.5535e-04 |
| 0.80 | 7.5970e-02 | 2.5298e-04 |
| 1.00 | 1.1498e-01 | 3.8623e-04 |
| 1.20 | 1.6707e-01 | 5.6607e-04 |
| 1.40 | 2.3601e-01 | 8.0661e-04 |
| 1.60 | 3.2661e-01 | 1.1259e-03 |
| 1.80 | 4.4493e-01 | 1.5470e-03 |
| 2.00 | 5.9866e-01 | 2.0994e-03 |

Table 8.1: Error $|x_i - w_i|$ for $N = 10$

| $t_i$ | Euler | 3rd Taylor |
|---|---|---|
| 0.00 | 0.0000e+00 | 0.0000e+00 |
| 0.10 | 2.5855e-03 | 2.1257e-06 |
| 0.20 | 5.7014e-03 | 4.6985e-06 |
| 0.30 | 9.4294e-03 | 7.7890e-06 |
| 0.40 | 1.3862e-02 | 1.1478e-05 |
| 0.50 | 1.9106e-02 | 1.5856e-05 |
| 0.60 | 2.5279e-02 | 2.1028e-05 |
| 0.70 | 3.2518e-02 | 2.7113e-05 |
| 0.80 | 4.0976e-02 | 3.4245e-05 |
| 0.90 | 5.0828e-02 | 4.2577e-05 |
| 1.00 | 6.2270e-02 | 5.2283e-05 |
| 1.10 | 7.5525e-02 | 6.3560e-05 |
| 1.20 | 9.0844e-02 | 7.6630e-05 |
| 1.30 | 1.0851e-01 | 9.1747e-05 |
| 1.40 | 1.2885e-01 | 1.0920e-04 |
| 1.50 | 1.5222e-01 | 1.2930e-04 |
| 1.60 | 1.7903e-01 | 1.5242e-04 |
| 1.70 | 2.0974e-01 | 1.7898e-04 |
| 1.80 | 2.4487e-01 | 2.0944e-04 |
| 1.90 | 2.8499e-01 | 2.4433e-04 |
| 2.00 | 3.3078e-01 | 2.8423e-04 |

Table 8.2: Error $|x_i - w_i|$ for $N = 20$

We compare the error $|x_i - w_i|$ using both the Euler method and the 3rd order Taylor method for two cases of $N$ on the interval $[0, 2]$. The results are shown in Table 8.1-8.2. As expected, for both cases of $N$, the Taylor method of third order produces a smaller error.

For an $n$-dimensional differentiable equation, the 2nd order Taylor method is

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)h + \mathbf{DF}(\mathbf{w}_i)\mathbf{F}(\mathbf{w}_i)\frac{h^2}{2}.$$

and the 3rd order Taylor method is

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)h + \mathbf{DF}(\mathbf{w}_i)\mathbf{F}(\mathbf{w}_i)\frac{h^2}{2} + \left[\mathbf{D}^2\mathbf{F}(\mathbf{F}(\mathbf{w}_i), \mathbf{F}(\mathbf{w}_i)) + \mathbf{DF}(\mathbf{w}_i)\mathbf{DF}(\mathbf{w}_i)\mathbf{F}(\mathbf{w}_i)\right]\frac{h^3}{6}.$$

## 8.4 Runge-Kutta Methods

In Euler's method, given $\mathbf{w}_i$ we compute $\mathbf{w}_{i+1}$ by following the line with direction vector $\mathbf{F}(\mathbf{w}_i)$ along a time interval of length $h$ starting from $\mathbf{w}_i$:

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)h$$

We are therefore using the value of $\mathbf{F}$ at the beginning of the interval $[t_i, t_i + h]$ to obtain the new estimate at the end-point of the interval. To improve Euler's method, we could seek an improved "average" value of $\mathbf{F}$ along the entire interval. As a first attempt, we could instead follow the line $t \to \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)t$ only to the mid-point of the interval $[0, h]$, and thus arrive at the point $\mathbf{y} = \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)\frac{h}{2}$, and then use $\mathbf{F}(\mathbf{y})$ as an estimate for the average value of $\mathbf{F}$ along the entire interval. The hope is that $\mathbf{F}(\mathbf{y})$ is a better estimate of the average change

$$\frac{\mathbf{x}(t_i + h) - \mathbf{x}(t_i)}{h}.$$

Using this approach, our estimate $\mathbf{w}_{i+1} \approx \mathbf{x}(t_i + h)$ is then computed in **two-stages** and is known as the **modified Euler method** or the **mid-point method**:

**Algorithm 8.5: Modified Euler Method**

Set $\mathbf{w}_0 = \mathbf{x}_0$ and for $i = 0, 1, \ldots, N-1$ compute

$$\begin{aligned} \mathbf{y} &= \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)\frac{h}{2} \\ \mathbf{w}_{i+1} &= \mathbf{w}_i + \mathbf{F}(\mathbf{y})h \end{aligned}$$

Notice that the computation of $\mathbf{F}(\mathbf{y})$ requires nested evaluations of $\mathbf{F}$:

$$\mathbf{F}(\mathbf{y}) = \mathbf{F}(\mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)\tfrac{h}{2})$$

Another attempt to improve Euler's method is to use the average of $\mathbf{F}(\mathbf{w}_i)$ and $\mathbf{F}(\mathbf{y})$ as the final direction in computing the new estimate $\mathbf{w}_{i+1}$ from $\mathbf{w}_i$. The result is another **two-stage** method known as **Heun's method**:

**Algorithm 8.6: Heun's Method**

Set $\mathbf{w}_0 = \mathbf{x}_0$ and for $i = 0, 1, \ldots, N-1$ compute

$$\begin{aligned} \mathbf{y} &= \mathbf{w}_i + \mathbf{F}(\mathbf{w}_i)h \\ \mathbf{w}_{i+1} &= \mathbf{w}_i + \frac{1}{2}\Big[\mathbf{F}(\mathbf{w}_i) + \mathbf{F}(\mathbf{y})\Big]h \end{aligned}$$

Both the mid-point method and Heun's method seem rather ad-hoc. We will show that in fact there is a systematic way to obtain both methods (and other methods), and that they agree with the 2nd order Taylor method up to order $h^2$. We begin by noticing that all methods we have considered thus far can be written in the form

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \mathbf{\Phi}(\mathbf{w}_i, h)h.$$

for a suitably defined function $\mathbf{\Phi}(\mathbf{w}_i, h)$. Explicitly: We now show that the mid-point and

| **Method** | $\mathbf{\Phi}(\mathbf{w}, h)$ |
|---|---|
| Euler | $\mathbf{F}(\mathbf{w})$ |
| 2nd-Order Taylor | $\mathbf{F}(\mathbf{w}) + \mathbf{DF}(\mathbf{w})\mathbf{F}(\mathbf{w})\frac{h}{2}$ |
| Mid-point | $\mathbf{F}(\mathbf{w} + \mathbf{F}(\mathbf{w})\frac{h}{2})$ |
| Heun | $\frac{1}{2}\big[\mathbf{F}(\mathbf{w}) + \mathbf{F}(\mathbf{w} + \mathbf{F}(\mathbf{w})h)\big]$ |

Table 8.3: Numerical methods considered thus far

Hen's method agree with the 2nd-order Taylor method up to order $h^2$. To keep the notation less cumbersome, we will consider the 1-dimensional case. The starting point is to consider the general method

$$\mathbf{\Phi}(w, h) = a_1 f(w) + a_2 f(w + \mu f(w)h)$$

for yet to be determined constants $a_1, a_2, \mu$. The idea is to expand $\mathbf{\Phi}$ in a Taylor series with respect to $h$:

$$\begin{aligned}\mathbf{\Phi}(w, h) &= a_1 f(w) + a_2 \left[f(w) + f'(w)\mu h f(w) + \tfrac{1}{2!} f''(w)\mu^2 h^2 (f(w))^2 + \cdots\right] \\ &= (a_1 + a_2) f(w) + a_2 \mu f'(w) f(w) h + O(h^2)\end{aligned}$$

In order for $\mathbf{\Phi}(w, h)$ to agree with the 2nd order Taylor method up to order $h^2$, we need

$$\begin{aligned} a_1 + a_2 &= 1 \\ a_2 \mu &= \frac{1}{2} \end{aligned}$$

One possible solution is $a_1 = a_2 = \frac{1}{2}$ and $\mu = 1$ which gives

$$\mathbf{\Phi}(w, h) = \frac{1}{2} f(w) + \frac{1}{2} f(w + f(w)h)$$

and from Table 8.3 we observe that this is Heun's method. Another possible solution is $a_1 = 0$, $a_2 = 1$, and $\mu = \frac{1}{2}$ which gives

$$\mathbf{\Phi}(w, h) = f(w + f(w)\tfrac{h}{2})$$

and from Table 8.3 we observe that this is the Mid-point method. In general, we observe that we obtain a family of methods parametrized by say $\mu \neq 0$ all of which agree with the 2nd order Taylor method up to order $h^2$:

$$a_1 = 1 - \frac{1}{2\mu}$$

$$a_2 = \frac{1}{2\mu}$$

The punchline of this approach is that we can achieve high-order accuracy by using nested evaluation of $f$ instead of explicitly computing derivatives of $f$ as needed in direct Taylor methods.

Following a similar approach of using nested evaluations, one can derive a 4th order method, known as the **4th order Runge-Kutta (RK4) method**, or simply the **Runge-Kutta method**:

**Algorithm 8.7: RK4**

Set $\mathbf{w}_0 = \mathbf{x}_0$ and for $i = 0, 1, \ldots, N-1$ compute

$$\mathbf{y}_1 = \mathbf{F}(\mathbf{w}_i)$$

$$\mathbf{y}_2 = \mathbf{F}\left(\mathbf{w}_i + \mathbf{y}_1 \tfrac{h}{2}\right)$$

$$\mathbf{y}_3 = \mathbf{F}\left(\mathbf{w}_i + \mathbf{y}_2 \tfrac{h}{2}\right)$$

$$\mathbf{y}_4 = \mathbf{F}(\mathbf{w}_i + \mathbf{y}_3 h)$$

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \left(\mathbf{y}_1 + 2\mathbf{y}_2 + 2\mathbf{y}_3 + \mathbf{y}_4\right)\frac{h}{6}$$

## 8.5 Convergence

In this section we settle the issue of convergence of a numerical scheme

$$\mathbf{w}_{i+1} = \mathbf{w}_i + \mathbf{\Phi}(\mathbf{w}_i, h)h \tag{8.7}$$

that produces discrete values $\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_N$ that approximate $\mathbf{x}(t_0), \mathbf{x}(t_1), \ldots, \mathbf{x}(t_N)$ where $\mathbf{x}(t)$ is the unique solution to the initial value problem

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{F}(\mathbf{x}(t)) \\ \mathbf{x}(a) &= \mathbf{x}_0 \end{aligned} \tag{8.8}$$

defined on the interval $[a, b]$. Here as before $t_i = a + ih$ for $i = 0, 1, \ldots, N$ and $h = \frac{b-a}{N}$. We define the **global discretization error** as

$$\mathbf{e}_i = \mathbf{x}(t_i) - \mathbf{w}_i$$

for $i = 0, 1, \ldots, N$. We say that the scheme $\mathbf{\Phi}$ is **convergent** if

$$\lim_{N\to\infty} \left( \max_{0\le i\le N} \|\mathbf{e}_i\| \right) = 0$$

An important notion needed in a typical proof of convergence of numerical methods for ODEs is that of the **local truncation error**.

**Definition 8.8: Local Truncation Error**

Let $\mathbf{x} : [a, b] \to \mathbb{R}^n$ be the unique solution to the IVP (8.8). The **local truncation error** of the method $\Phi$ is

$$\tau(t, h) = \frac{\mathbf{x}(t + h) - \mathbf{x}(t)}{h} - \Phi(\mathbf{x}(t), h)$$

for $t, t + h \in [a, b]$.

The local truncation error of a method $\mathbf{\Phi}$ is a measure of how well the exact solution $\mathbf{x}(t)$ satisfies the method $\mathbf{\Phi}$, or equivalently it is a measure of the accuracy of the method at a specific step assuming the exact solution is known at the previous step. Consider for example the mid-point method $\mathbf{\Phi}(w, h) = f(w + f(w)\frac{h}{2})$, where for simplicity we consider a 1-dimensional ODE (the $n$-dimensional is conceptually the same just more notation). Then as previously shown

$$\mathbf{\Phi}(\mathbf{w}, h) = f(w) + f'(w)f(w)\frac{h}{2} + f''(\xi)f(w)^2\frac{h^2}{8}$$

where $\xi$ is in between $w$ and $w + f(w)\frac{h}{2}$. On the other hand,

$$x(t + h) = x(t) + f(x(t))h + f'(x(t))f(x(t))\frac{h^2}{2} + x^{(3)}(s)\frac{h^3}{6}$$

where $t < s < t+h$. Note that the $x^{(3)}(s)$ depends on up to the second derivatives of $f$ since $\dot{x}(t) = f(x(t))$. Therefore,

$$\begin{aligned}
\tau(t, h) &= \frac{x(t + h) - x(t)}{h} - \mathbf{\Phi}(x(t), h) \\
&= x^{(3)}(s)\frac{h^2}{6} - f''(\xi)f(w)^2\frac{h^2}{8} \\
&= \left[\frac{x^{(3)}(s)}{6} - \frac{f''(\xi)f(x(t))^2}{8}\right] h^2
\end{aligned}$$

Therefore if the second order derivative of $f$ are continuous then

$$|\tau(t,h)| \leq Mh^2$$

for some constant $M > 0$. We say then that the local truncation error is $O(h^2)$. A similar computation shows that the local truncation error of Heun's method is $O(h^2)$ and the RK4 method has local truncation error $O(h^4)$, provided of course that $f$ has continuous derivatives of sufficiently high order. The next theorem gives a sufficient condition for convergence of a method $\mathbf{\Phi}$ whose local truncation error is of order $O(h^p)$ and that satisfies a Lipschitz condition.

**Theorem 8.9: Convergence**

Consider the IVP (8.8) and let $\mathbf{x} : [a,b] \to \mathbb{R}^n$ be its unique solution. Consider the numerical method (8.7) where $h = \frac{b-a}{N}$ and $t_i = a + ih$ for $i = 0, 1, \ldots, N$. Suppose that there constants $M, L, h_0 > 0$ such that the local truncation error satisfies

$$\|\tau(t, \mathbf{x}(t), h)\| \leq Mh^p$$

for all $t, t+h \in [a,b]$ for some $p > 0$ and

$$\|\mathbf{\Phi}(\mathbf{y}, h) - \mathbf{\Phi}(\mathbf{z}, h)\| \leq L\|\mathbf{y} - \mathbf{z}\|$$

for all $\mathbf{y}, \mathbf{z} \in \mathbb{R}^n$, and all $h \leq h_0$. Then the discretization error $\mathbf{e}_i = \mathbf{x}(t_i) - \mathbf{w}_i$ satisfies

$$\|\mathbf{e}_i\| \leq \frac{Mh^p}{L}\left(e^{ihL} - 1\right)$$

for all $i = 0, 1, \ldots, N$. Consequently, the method $\mathbf{\Phi}$ is convergent.

*Proof.* The discretization error can be written as

$$\begin{aligned}
\mathbf{e}_{i+1} &= \mathbf{x}(t_{i+1}) - \mathbf{w}_{i+1} \\
&= \mathbf{x}(t_i) + \left(\frac{\mathbf{x}(t_{i+1}) - \mathbf{x}(t_i)}{h}\right) h - (\mathbf{w}_i + \mathbf{\Phi}(\mathbf{w}_i, h)h) \\
&= \mathbf{e}_i + \left(\frac{\mathbf{x}(t_{i+1}) - \mathbf{x}(t_i)}{h} - \mathbf{\Phi}(\mathbf{w}_i, h)\right) h \\
&= \mathbf{e}_i + \left(\left(\frac{\mathbf{x}(t_{i+1}) - \mathbf{x}(t_i)}{h}\right) - \mathbf{\Phi}(\mathbf{x}(t_i), h) + \mathbf{\Phi}(\mathbf{x}(t_i), h) - \mathbf{\Phi}(\mathbf{w}_i, h)\right) h \\
&= \mathbf{e}_i + (\tau(t_i, \mathbf{x}(t_i), h) + \mathbf{\Phi}(\mathbf{x}(t_i), h) - \mathbf{\Phi}(\mathbf{w}_i, h))\, h
\end{aligned}$$

By the triangle inequality and the assumption on the local truncation error and the Lipschitz condition on $\mathbf{\Phi}$:

$$\begin{aligned}\|\mathbf{e}_{i+1}\| &\leq \|\mathbf{e}_i\| + \|\tau(t_i, \mathbf{x}(t_i), h)\|h + \|\mathbf{\Phi}(\mathbf{x}(t_i), h) - \mathbf{\Phi}(\mathbf{w}_i, h)\|h \\ &\leq \|\mathbf{e}_i\| + Mh^{p+1} + L\|\mathbf{x}(t_i) - \mathbf{w}_i\|h \\ &= (1 + hL)\|\mathbf{e}_i\| + Mh^{p+1}\end{aligned}$$

Then by Lemma 8.3 with $\alpha = hL$ and $\beta = Mh^{p+1}$, and since $\|\mathbf{e}_0\| = 0$, we have

$$\|\mathbf{e}_i\| \leq \frac{Mh^p}{L}\left(e^{ihL} - 1\right)$$

This proves the first claim. Since $ih \leq Nh = (b - a)$ we obtain that

$$\|\mathbf{e}_i\| \leq \frac{Mh^p}{L}\left(e^{(b-a)L} - 1\right)$$

and therefore

$$\max_{1\leq i\leq N} \|\mathbf{e}_i\| \leq \frac{Mh^p}{L}\left(e^{(b-a)L} - 1\right)$$

Hence, since $h = \frac{b-a}{N}$ we have that

$$\lim_{N\to\infty}\left(\max_{1\leq i\leq N} \|\mathbf{e}_i\|\right) = 0$$

proving that the method is convergent. □

We have simplified the proof by assuming that $\mathbf{\Phi}$ satisfies a global Lipschitz condition. This global Lipschitz condition can be relaxed significantly but the idea of the proof remains the same.

**Example 8.13.** Consider the mid-point method $\Phi(w, h) = f(w + f(w)\frac{h}{2})$. Suppose that $f : \mathbb{R}^n \to \mathbb{R}^n$ satisfies a Lipschitz constant, say that $\|f(y) - f(z)\| \leq K\|y - z\|$.
(a) Show that if $h_0 > 0$, then $\Phi$ satisfies a Lipschitz condition on the set

$$\{(w, h) \,|\, w \in \mathbb{R}^n,\ 0 < h \leq h_0\}.$$

(b) Under what conditions will the local truncation error satisfy the assumptions of the convergence theorem.

*Solution.* (a) Suppose that $\|f(y) - f(z)\| \leq K\|y - z\|$ for some constant $K > 0$. Then for $z, y \in \mathbb{R}^n$:

$$\begin{aligned}
\|\Phi(y,h) - \Phi(z,h)\| &= \|f(y + f(y)\tfrac{h}{2}) - f(z + f(z)\tfrac{h}{2})\| \\
&\leq K\|(y + f(y)\tfrac{h}{2}) - (z + f(z)\tfrac{h}{2})\| \\
&= K\|(y - z) + (f(y) - f(z))\tfrac{h}{2}\| \\
&\leq K\left(\|y - z\| + \|f(y) - f(z)\|\tfrac{h}{2}\right) \qquad \text{(triangle inequality)} \\
&= K\|y - z\| + K\|f(y) - f(z)\|\tfrac{h}{2} \\
&\leq K\|y - z\| + K^2\|y - z\|\tfrac{h}{2} \\
&= (K + K^2\tfrac{h}{2})\|y - z\| \\
&\leq (K + K^2\tfrac{h_0}{2})\|y - z\|.
\end{aligned}$$

Hence, if $L = K + K^2\frac{h_0}{2}$, then $\|\Phi(y,h) - \Phi(z,h)\| \leq L\|y - z\|$ for all $y, z \in \mathbb{R}^n$.

(b) We have previously shown that

$$\mathbf{\Phi}(w,h) = f(w) + f'(w)f(w)\frac{h}{2} + f''(\xi)f(w)^2\frac{h^2}{8}$$

where $\xi$ is in between $w$ and $w + f(w)\frac{h}{2}$ and that

$$x(t+h) = x(t) + f(x(t))h + f'(x(t))f(x(t))\frac{h^2}{2} + x^{(3)}(s)\frac{h^3}{6}$$

where $t < s < t + h$. Therefore,

$$\begin{aligned}
\tau(t,h) &= \frac{x(t+h) - x(t)}{h} - \Phi(x(t),h) \\
&= x^{(3)}(s)\frac{h^2}{6} - f''(\xi(t))f(x(t))^2\frac{h^2}{8} \\
&= \left(x^{(3)}(s)\tfrac{1}{6} - f''(\xi(t))f(x(t))^2\tfrac{1}{8}\right)h^2
\end{aligned}$$

If $f''$ is continuous then $x^{(3)}$ is continuous and then $\|x^{(3)}(s)\frac{1}{6} - f''(\xi(t))f(x(t))^2\frac{1}{8}\| \leq M$ for some $M > 0$. In this case, $\|\tau(t,h)\| \leq Mh^2$, and the assumption needed in the convergence theorem is satisfied.

□

The 4th order Runge-Kutta method requires four function evaluations per step whereas the midpoint method requires two and Euler's method only requires one. Hence, to compare the accuracy of the methods we must ensure the computational effort for each method is the same. Thus if a step size of $h$ is used for RK4 then we should use a step-size of $\frac{h}{4}$ for Euler and $\frac{h}{2}$ for the midpoint method. If RK4 is to be a superior method then it should give a smaller global discretization error with a larger step-size.

**Example 8.14.** Consider the IVP

$$\dot{x} = 1 + x^2, \quad x(0) = 0.$$

The solution is $x(t) = \tan(t)$. Compare the global discretization error using Euler's method with $N_1 = 40$, midpoint method with $N_2 = 20$, and RK4 with $N_3 = 10$, on the interval $[0, 10]$.