Lecture Notes on Numerical Analysis[1]

(Course code: MATH3230A)

(Academic Year: 2018/2019, First Semester)

Andrew Lam
Department of Mathematics
The Chinese University of Hong Kong
Shatin, N.T., Hong Kong

Mainly based on previous lecture notes by Prof. Jun Zou and the textbooks

Numerical Analysis: Mathematics of Scientific Computing
Brooks/Cole Publishing Co., 2009
by D. Kincaid and W. Cheney
and
Afternotes on Numerical Analysis, SIAM, 2006
by G.W. Stewart

[1] The lecture notes were prepared by Andrew Kei Fong Lam for the teaching of the course " Numerical Analysis ". Students taking this course may use the notes as part of their reading and reference materials. There might be many mistakes and typos, including English grammatical and spelling errors, in the notes. It would be greatly appreciated if those students, who will use the notes as their reading or reference material, report any mistakes and typos to the instructor Andrew Lam for improving the lecture notes.

Contents

1 Introduction

Numerical Analysis is a fundamental branch in Computational and Applied Mathematics. In this section, we list some important topics from Numerical Analysis, which will be covered in this course.

1. **Nonlinear equations of one variable**. We discuss how to solve nonlinear equations of one variable (in standard form):

   $$f(x) = 0,$$

   where $f(x)$ is nonlinear with respect to variable x.

   System of nonlinear equations. In most applications one may need to solve the following more general system of nonlinear equations:

   $$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0, \\ f_2(x_1, x_2, \ldots, x_n) &= 0, \\ &\vdots \\ f_n(x_1, x_2, \ldots, x_n) &= 0, \end{aligned}$$

   where each $f_i(x_1, x_2, \ldots, x_n)$ is a nonlinear function of n variables $x_1, \ldots, x_n$. In general, the solutions of a system of nonlinear equations are much more complicated than the solutions to a single nonlinear equation, but many methods for equations of one variable can be generalized to systems of nonlinear equations.

2. **Linear system of algebraic equations**. Linear systems are often the problems one needs to solve repeatedly, or thousands times during many mathematical modelling processes or physical/engineering simulations, e.g., in the numerical solutions of a simple population model, or the more complicated electromagnetic Maxwell system. We will study how to solve the following general system of linear algebraic equations:

   $$Ax = b$$

   where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$. While a 2×2 or a 3×3 can be solved by hand, as m and n become large we have to turn to computers for help. Furthermore, when $m \neq n$, the system may not have a solution, then how can we define "meaningful" solutions to the original problem?

3. **Floating-point arithmetic**. When computers are used for numerical computations, rounding-off errors are always present. Then how can we solve the system of linear algebraic equations $Ax = b$ or the system of nonlinear algebraic equations $F(x) = 0$, with satisfactory accuracy? How can we judge the solutions computed by computers are reliable?

4. **Interpolation**. For a given set of observation data, can we find a function to best describe/interpolate the data points? A prime example is population statistics, which is very expensive to obtain. Can we infer the population in some in-between years based on the known statistics from census years?

5. **Numerical integration**. Integration is involved in many practical applications, e.g., computing the physical masses, surface areas, volumes, fluxes, etc. But most integrals are difficult or impossible to compute exactly. For the integration of a complicated function, can we compute some good approximate value of the integral when it is impossible to calculate the integral exactly?

2 Nonlinear equation in one variable

A nonlinear equation of one variable is an equation of the form

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

where f is a nonlinear function with respect to x, and x is the only independent variable.

(1) Nonlinear equations may have **no solutions**. For example, the equation

$$f(x) = x^2 - 4x + 5 = 0$$

has no solutions in $\mathbb{R}$. In fact, $f(x) = (x-2)^2 + 1 \geq 1$ for all $x \in \mathbb{R}$.

We should first check if a nonlinear equation has solutions before solving it. If no solutions exist, then we have to look for meaningful approximate solutions to the problem, such as those that minimize some form of error.

(2) Solutions may not be **unique**. Consider the equation

$$f(x) = \sin x \cos x - \frac{1}{2} = 0.$$

It is easy to see that $x = 2n + \pi/4$ is a solution for any integer n. But most numerical methods can approximate only one solution with one initial guess. So when we construct numerical methods for a nonlinear equation, we should locate a range in which there is a unique solution to the nonlinear equation.

2.1 Iterative methods and rate of convergence

Let x^* be an exact solution to the nonlinear equation, i.e., $f(x^*) = 0$. One of the most popular and effective method to compute for x^* is the class of **iterative methods**. The idea is to start with an initial value/guess x_0, and from this initial guess we generate a sequence $x_1, x_2, \ldots, x_k, x_{k+1}, \ldots$ such that

$$\lim_{k\to\infty} x_k = x^*.$$

Many different iterative methods can be proposed to solve the same problem, and to distinguish which is **better** we use the concept of **rate of convergence**.

Definition 2.1 (Q(uotient)-Linear convergence)**.** *Given a sequence $(x_n)_{n\geq 0}$ converging to a limit x^*. If there is a constant ρ satisfying $0 \leq \rho \leq 1$ such that*

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = \rho,$$

then ρ is the **rate of convergence***, and we say that*

$$x_k \to x^* \quad \begin{cases} \textbf{\textit{Q-linearly}} \textit{ with rate } \rho & \textit{if } \rho \in (0,1), \\ \textbf{\textit{Q-superlinearly}} & \textit{if } \rho = 0, \\ \textbf{\textit{Q-sublinearly}} & \textit{if } \rho = 1. \end{cases}$$

Unfortunately, this definition does not apply to all converging sequences. For example, take

$$a_1 = a_2 = \frac{1}{4}, \quad a_3 = a_4 = \frac{1}{16}, \quad a_5 = a_6 = \frac{1}{64}, \quad \dots, \quad a_{2k-1} = a_{2k} = \frac{1}{2^{2k}}, \dots$$

which converges to zero. Then

$$\frac{a_2}{a_1} = 1, \quad \frac{a_3}{a_2} = \frac{1}{4}, \quad \frac{a_4}{a_3} = 1, \quad \frac{a_5}{a_4} = \frac{1}{4}, \quad \dots, \quad \frac{a_{2k-1}}{a_{2k-2}} = \frac{1}{4}, \quad \dots,$$

and so the limit $\lim_{n\to\infty} \frac{a_{n+1}}{a_n}$ may not exist, since the sequence oscillates between 1 and $\frac{1}{4}$. However, note that

$$|a_k| \le \frac{1}{2^k} =: \varepsilon_k \text{ for all } k \in \mathbb{N},$$

and $(\varepsilon_k)_{k\ge 0}$ converges to zero Q-linearly with rate $\rho = \frac{1}{2}$. Therefore, we can think of $(a_k)_{k\ge 0}$ converging to zero at best linearly. This motivates the following extended definition.

Definition 2.2 (R(oot)-Linear convergence)**.** *Given a sequence $(x_n)_{n\ge 0}$ converging to a limit x^*. Suppose there exists a sequence $(\varepsilon_n)_{n\ge 0}$ such that $\varepsilon_n \to 0$ as $n \to \infty$ and*

$$|x_n - x^*| \le \varepsilon_n \textit{ for all } n \in \mathbb{N}.$$

Then, we say that

$$x_k \to x^* \quad \begin{cases} \textbf{\textit{R-linearly}} & \textit{if } \varepsilon_n \to 0 \textit{ Q-linearly,} \\ \textbf{\textit{R-superlinearly}} & \textit{if } \varepsilon_n \to 0 \textit{ Q-superlinearly,} \\ \textbf{\textit{R-sublinearly}} & \textit{if } \varepsilon_n \to 0 \textit{ Q-sublinearly.} \end{cases}$$

Definition 2.3 (Convergence with order p)**.** *If there are two positive constants $p > 1$ (not necessarily an integer) and ρ such that*

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \rho, \tag{2.2}$$

then the sequence $(x_k)_{k\ge 0}$ is said to Q-converge to x^* with order p. The constant ρ is called the rate of convergence. If the order parameter $p = 2$ in (2.2), the convergence is said to be **Q-quadratic***, and if $p = 3$ in (2.2), the convergence is said to be* **Q-cubic***.*

If there exists a sequence $(\varepsilon_n)_{n\ge 0}$ such that $\varepsilon_n \to 0$ as $n \to \infty$ and

$$|x_n - x^*| \le \varepsilon_n \textit{ for all } n \in \mathbb{N},$$

then we say $(x_n)_{n\ge 0}$ R-converges to x^* with order p if $(\varepsilon_n)_{n\ge 0}$ Q-converges to zero with order p.

Lemma 2.1. *If the sequence $(x_n)_{n\ge 0}$ Q-converges to x^* with order $p > 0$, then it Q-converges to x^* with any order q for $0 < q < p$.*

Proof. Exercise. □

2.2 Absolute and relative errors

We introduce two important concepts to measure the accuracy of approximate values obtained from an iterative method.

Definition 2.4 (Absolute error). *The* **absolute error** *between the true value (solution) x^* and the approximate value x_k is the error given by*

$$|x_k - x^*|.$$

The **absolute error** between the true value (solution) x^* and the approximate value x_k is the error given by

$$|x_k - x^*|.$$

This error considers only the **distance** of x_k from x^* **without taking care of the magnitude** of the true solution x^*. This can cause problems in applications, for example, consider $\varepsilon = 10^{-5}$, and the true solution $x^* = 1$. If we stop iteration when $|x_k - x^*| \leq 10^{-5}$, then x_k has 5 accurate digits after the decimal point, i.e., $x_k = 0.99999\ldots$. But if the true solution is $x^* = 10^{-8}$, then we will stop the iteration even when $x_k = 10^{-5}$ since

$$|x_k - x^*| = 10^{-5} - 10^{-8} \leq 10^{-5},$$

this approximation x_k is 1000 times of the exact solution x^*, and so it is not accurate at all.

Definition 2.5 (Relative error). *Let x_k be an approximation to $x^* \neq 0$, then the error*

$$\rho = \frac{|x_k - x^*|}{|x^*|}$$

is called the **relative error** *of x_k.*

This allows us to write

$$x_k = x^*(1 + \varepsilon) = x^* + \varepsilon x^*, \quad |\varepsilon| = \rho,$$

and view x_k can be viewed as a small perturbation of x^*. Below we present a table of approximation to $x^* = e = 2.7182818\ldots$:

Approximation	ρ	accurate digits
2.0	$2 \times 10^{-1} \approx 10^{-1}$	1
2.7	$6 \times 10^{-3} \approx 10^{-2}$	2
2.71	$3 \times 10^{-3} \approx 10^{-3}$	3
2.718	$1 \times 10^{-4} \approx 10^{-4}$	4
2.7182	$3 \times 10^{-5} \approx 10^{-5}$	5
2.71828	$6 \times 10^{-7} \approx 10^{-6}$	6

and so

If the relative error of x_k is approximately 10^{-k}, then x_k and x^* agree to k digits, and vice versa.

2.3 Bisection method

The simplest iterative method for solving $f(x) = 0$ is the **Bisection algorithm**. It is based on the following variant of the intermediate value theorem:

Lemma 2.2. *Let f be a continuous function in the interval $[a, b]$. If the product $f(a)f(b) < 0$, then there exists at least one solution $x^* \in (a, b)$ such that $f(x^*) = 0$.*

Due to finite computational time, we usually can not obtain the exact solution x^* on a computer machine, and so we will be satisfied when we find an approximate solution $\tilde{x}^*$ such that $|f(\tilde{x}^*)| \leq \varepsilon$ or $|\tilde{x}^* - x^*| \leq \delta$ for some small tolerance parameters ε and δ. One may set ε and δ to be at different magnitude in each application.

Assume that $f(a) < 0$ and $f(b) > 0$, then there exists at least one point $x^* \in (a, b)$ such that $f(x^*) = 0$. Then, the Bisection Algorithm to find an approximate solution $\tilde{x}^*$ proceed as follows:

1. Input a and b, and a stopping tolerance parameter δ.
2. Set $a_0 := a$, $b_0 := b$ and $k := 0$.
3. While $|b_k - a_k| > \delta$, set $x_k = \frac{1}{2}(a_k + b_k)$ as the midpoint, and do the following:
   - If $f(x_k)f(a_k) > 0$ set $a_{k+1} = x_k$ and $b_{k+1} = b_k$,
   - otherwise set $a_{k+1} = a_k$ and $b_{k+1} = x_k$.
4. Output x_k if $|b_k - a_k| \leq \delta$.

Theorem 2.1 (Convergence)**.** *Let $f(x)$ be a continuous function on $[a, b]$ such that $f(a)f(b) < 0$, then the bisection algorithm* **always converges** *to a solution x^* of the equation $f(x) = 0$, and the following error estimate holds for the k^{th} approximate value x_k:*

$$|x_k - x^*| \leq \frac{1}{2}(b_k - a_k) = 2^{-(k+1)}(b - a).$$

Proof. Denote by

$$[a_0, b_0], \quad [a_1, b_1], \quad \ldots, \quad [a_k, b_k], \quad \ldots$$

the successive intervals generated from the Bisection algorithm. Then, by construction

$$a_0 \leq a_1 \leq a_2 \leq \cdots \leq a_k \leq \cdots \leq b_0, \quad b_0 \geq b_1 \geq b_2 \geq \cdots \geq b_k \geq \cdots \geq a_0,$$

and

$$b_n - a_n = \frac{1}{2}(b_{n-1} - a_{n-1}) = 2^{-n}(b_0 - a_0) \quad \text{for all } n \geq 1.$$

From these properties, the sequences $(a_k)_{k\geq 0}$ and $(b_k)_{k\geq 0}$ both converge to the same limit. Let

$$x^* = \lim_{k\to\infty} a_k = \lim_{k\to\infty} b_k,$$

then

$$f(a_k)f(b_k) < 0 \quad \Rightarrow \quad f(x^*)^2 \leq 0 \quad \Rightarrow \quad f(x^*)^2 = 0,$$

using the analysis fact (if $g_k \to g$ and $g_k < 0$, then $g \leq 0$) which can be proved by a contradiction argument. Therefore the limit x^* is a solution to the nonlinear equation $f(x) = 0$. To get the error estimate let x_k be the midpoint of $[a_k, b_k]$, then x^* lies in either $[a_k, x_k]$ or $[x_k, b_k]$ and thus

$$|x_k - x^*| \leq \frac{1}{2}(b_k - a_k) = 2^{-(k+1)}(b_0 - a_0).$$

$\square$

Notice that it is not easy to fit $|x_k - x^*| \leq 2^{-(k+1)}(b_0 - a_0)$ into the definition of Q-convergence, but using the extended definition we can easily see that the sequence $(x_k)_{k\geq 0}$ generated by the Bisection algorithm converges to x^* R-linearly.

2.4 Newton's method

The Bisection method converges under a very weak assumption on $f(x)$ (i.e., continuity) but the algorithm converges very slowly. **Newton's method/Newton-Raphson method**, also an iterative method, is one of the most powerful numerical methods for solving nonlinear equations, and one advantage is that it converges Q-quadratically if the initial guess is close to the true solution.

2.4.1 Geometric derivation

1. Start with an initial point x_0, draw the tangent line to the curve $y = f(x)$ at the point $(x_0, f(x_0))$.
2. Find the intersection point of the tangent line with the x-axis, which Newton's method takes to be the new approximation x_1.
3. Repeat the same procedure to get $x_2, x_3, \ldots$.

To obtain a formula for $x_1, x_2, \ldots$, the tangent line of the curve $y = f(x)$ at the point $(x_0, f(x_0))$ is given by

$$y - f(x_0) = f'(x_0)(x - x_0).$$

The intersection point of this line with the x-axis can be found from

$$-f(x_0) = f'(x_0)(x - x_0),$$

which gives

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

Similarly we can derive

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}, \quad \ldots, \quad x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad \ldots,$$

so that Newton's method reads as

$$\boxed{x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \quad n = 0, 1, 2, \ldots} \tag{2.3}$$

Example 2.1. *For $a \neq 0$, set $f(x) = \frac{1}{x} - a$. Then, Newton's method is given by*

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = 2x_k - ax_k^2, \quad k = 0, 1, 2, \cdots$$

As long as $x_0 \in (0, \frac{1}{a})$, the sequence $(x_k)_{k\geq 0}$ converges monotonically to the true solution $x^* = \frac{1}{a}$. The following table gives the sequence $\{x_k\}$ to approximate $1/a$ when $a = 1$ with different initial guesses:

Iteration	x_k	x_k	x_k	x_k
0	0.250000	1.750000	2.000000	$+2.100000 \times 10^0$
1	0.437500	0.437500	0.000000	-0.210000×10^0
2	0.683594	0.683594	0.000000	-4.641000×10^{-1}
3	0.899887	0.899887	0.000000	-1.143589×10^0
4	0.989977	0.989977	0.000000	-3.594973×10^0
5	0.999899	0.999899	0.000000	-2.011378×10^1
6	0.999999	0.999999	0.000000	-4.447915×10^2

We see that the choice of the initial guesses is **very important** *to the convergence of Newton's method.*

Example 2.2. *Let $a > 0$, and set $f(x) = x^2 - a$. Then, Newton's method is given by*

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = \frac{1}{2}(x_k + \frac{a}{x_k}), \quad k = 0, 1, 2, \ldots$$

The true solution is $x^* = \sqrt{a}$, and for $a = 2$ the following table indicates that Newton's method converges very rapidly with quite different initial guesses. This is a **unusual example** *as the function is a quadratic polynomial and the method converges to $x^* = \sqrt{2}$ with any initial guess.*

Iteration	x_k	x_k	x_k
0	1.000000	0.500000	6.000000
1	1.500000	2.250000	3.166667
2	1.416667	1.569444	1.899123
3	1.414216	1.421890	1.476120
4	1.414214	1.414234	1.415512
5	1.414214	1.414214	1.414214

2.5 Fixed-point iterative methods

Newton's method

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \ldots,$$

requires the computation of the derivative $f'(x_k)$ at each iteration. In some application this may not be possible due to

(1) The expression of $f(x)$ is unknown;

(2) The derivative $f'(x)$ is very expensive to compute;

(3) The value of function f may be the result of a long numerical calculation, so the derivative has no formula available.

Example 2.3. *Consider the following two-point boundary value problem:*

$$\begin{cases} x''(t) + p(t)x'(t) + q(t)x(t) = g(t), \quad a < t < b\,, \\ x(a) = \alpha, \quad x(b) = \beta \end{cases} \tag{2.4}$$

where $p(t)$, $q(t)$ and $g(t)$ are given functions. One popular way to find the solution $x(t)$ is to first solve the following initial value problem:

$$\begin{cases} x''(t) + p(t)x'(t) + q(t)x(t) = g(t), \quad t > a\,, \\ x(a) = \alpha, \quad x'(a) = z \end{cases}$$

for a given z. We write the solution as $x(t, z)$, then $x(t, z)$ will be also the solution to the boundary-value problem (2.4) *if we can find a value z such that*

$$x(b, z) = \beta.$$

Let $f(z) = x(b, z) - \beta$, then z is the solution to the nonlinear equation

$$f(z) = 0.$$

To find such solutions z, we may apply the Newton's method:

$$z_{k+1} = z_k - \frac{f(z_k)}{f'(z_k)}, \quad k = 0, 1, 2, \ldots.$$

But computing the derivative $f'(z_k)$ is complicated and can not be done directly. Then the question is can we avoid computing derivatives $f'(x_k)$ when they are difficult to do?

One possibility to get rid of the derivatives in Newton's method is to find some approximations of the derivatives $f'(x_k)$, say $g_k \approx f'(x_k)$, but g_k should be much easier to compute than $f'(x_k)$. Then we replace the Newton's method by the following:

$$x_{k+1} = x_k - \frac{f(x_k)}{g_k}, \quad k = 0, 1, 2, \ldots .$$

Such an iterative method is called a **Quasi-Newton's method**. There are many possible approximations of $f'(x_k)$, thus generating many different quasi-Newton's methods. For example, one can replace $f'(x_k)$ by the difference quotient

$$g_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}},$$

the resulting method is called the **Secant method**, which needs two initial values x_0 and x_1 to start. Another is to replace $f'(x_k)$ by a constant $g_k = g$, leading to

$$x_{k+1} = x_k - \frac{f(x_k)}{g}, \quad k = 0, 1, 2, \ldots, \tag{2.5}$$

which is called the **Constant slope method**. In particular, we might take $g = f'(x_0)$. Both Netwon's method and the above quasi-Newton's methods can be seen as special cases of fixed-point iterative methods.

Definition 2.6 (Fixed point). *For a given function $\varphi(x)$, x^* is called its fixed point if x^* satisfies $\varphi(x^*) = x^*$.*

For the Newton's method, we have

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

and for quasi-Newton's methods, we have

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

Definition 2.7. *For a given function $\varphi(x)$, the iterative method*

$$x_{k+1} = \varphi(x_k)\ ,\ k = 0, 1, 2, \ldots$$

is called a **fixed-point iteration** *associated with the function $\varphi(x)$, and $\varphi(x)$ is called the* **iterative function**.

For the fixed-point iterative method, we have the following general result regarding convergence and order of convergence.

Theorem 2.2. *If the iterative function $\varphi(x)$ satisfies the condition*

$$|\varphi'(x^*)| < 1, \tag{2.6}$$

then there exists a $\delta = \delta(x^*) > 0$ such that for any $x_0 \in [x^* - \delta, x^* + \delta]$, the fixed-point iteration converges. Furthermore,

- *If $\varphi'(x^*) \neq 0$, the convergence is Q-linear with convergence rate $\rho = |\varphi'(x^*)|$.*
- *If $\varphi'(x^*) = \varphi''(x^*) = \cdots = \varphi^{(p-1)}(x^*) = 0$ but $\varphi^{(p)}(x^*) \neq 0$, then the fixed-point iteration Q-converges with order p.*

Proof. For the first part, using the assumption $|\varphi'(x^*)| < 1$, there exists $\delta > 0$ such that

$$|\varphi'(x)| \leq \alpha < 1 \quad \forall x \in [x^* - \delta, x^* + \delta]. \tag{2.7}$$

Then, by the mean-value theorem,

$$x_1 - x^* = \varphi(x_0) - \varphi(x^*) = \varphi'(\xi_0)(x_0 - x^*)$$

for some ξ_0 lying in between x_0 and x^*. Then, if $x_0 \in [x_* - \delta, x^* + \delta]$, it holds that $|\varphi'(\xi_0)| \leq \alpha$, and so

$$|x_1 - x^*| = |\varphi'(\xi_1)(x_0 - x^*)| \leq |\varphi'(\xi_1)||x_0 - x^*| \leq \alpha\delta < \delta.$$

This yields $x_1 \in (x^* - \delta, x^* + \delta)$. Then, by the mean value theorem, for $k \in \mathbb{N}$ we have

$$x_{k+1} - x^* = \varphi(x_k) - \varphi(x^*) = \varphi'(\xi_k)(x_k - x^*),$$

where ξ_k lies between x_k and x^*. By induction, if $x_k \in (x^* - \delta, x^* + \delta)$, then $\xi_k \in (x^* - \delta, x^* + \delta)$ and $|\varphi'(\xi_k)| \leq \alpha$ with

$$|x_{k+1} - x^*| = |\varphi'(\xi_k)(x_k - x^*)| \leq \alpha|x_k - x^*| \leq \alpha\delta < \delta.$$

Hence, $x_{k+1} \in (x^* - \delta, x^* + \delta)$. Furthermore, we have the recursive estimate

$$|x_{k+1} - x^*| \leq \alpha|x_k - x^*| \leq \cdots \leq \alpha^{k+1}|x_0 - x^*|.$$

Since $\alpha < 1$ we know that $x_k \to x^*$ as $k \to \infty$, and so the iterative method converges. Furthermore, suppose $\varphi'(x^*) \neq 0$, then by the mean value theorem we find that

$$x_{k+1} - x^* = \varphi'(\xi_k)(x_k - x^*) \quad \Rightarrow \quad \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = |\varphi'(\xi_k)| \to |\varphi'(x^*)| \text{ as } k \to \infty,$$

which implies linear convergence. For the second part, we suppose

$$\varphi'(x^*) = \varphi''(x^*) = \cdots = \varphi^{(p-1)}(x^*) = 0, \quad \text{but} \quad \varphi^{(p)}(x^*) \neq 0.$$

We have by Taylor's expansion,

$$\varphi(x) = \varphi(x^*) + \frac{\varphi'(x^*)}{1!}(x - x^*) + \frac{\varphi''(x^*)}{2!}(x - x^*)^2 + \cdots + \frac{\varphi^{(p-1)}(x^*)}{(p-1)!}(x - x^*)^{p-1} + \frac{\varphi^{(p)}(\xi)}{p!}(x - x^*)^p$$

where ξ lies between x and x^*. Thus by the given assumptions,

$$x_{k+1} = \varphi(x_k) = \varphi(x^*) + \frac{\varphi^{(p)}(\xi_k)}{p!}(x_k - x^*)^p = x^* + \frac{\varphi^{(p)}(\xi_k)}{p!}(x_k - x^*)^p.$$

Since ξ_k lies between x_k and x^*, and $x_k \to x^*$ when $k \to \infty$, so we know $\xi_k \to x^*$ when $k \to \infty$. This leads to

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^p} = \frac{|\varphi^{(p)}(x^*)|}{p!} \neq 0,$$

so the convergence is of order p. □

Theorem 2.3 (Quadratic convergence of Newton's method)**.** *Let f be a function such that $f''(x)$ is continuous near x^*, $f''(x^*) \neq 0$ and $f'(x^*) \neq 0$. Then, there exists a constant $\delta = \delta(x^*) > 0$ such that if the initial guess x_0 satisfies $|x_0 - x^*| \leq \delta$, then the sequence $(x_k)_{k\geq 0}$ generated by Newton's method* (2.3) *satisfies $|x_k - x^*| \leq \delta$ and $\lim_{k\to\infty} x_k = x^*$. Moreover,*

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|^2} = \frac{|f''(x^*)|}{2|f'(x^*)|},$$

and so Newton's method converges **Q-quadratically**.

Proof. Define the iterative function

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

Then, Newton's method is given as

$$x_{k+1} = \varphi(x_k), \quad k = 0, 1, 2, \ldots.$$

Its derivative at x^* is

$$\varphi'(x^*) = 1 - \frac{f'(x^*)}{f'(x^*)} + \frac{f(x^*)f''(x^*)}{(f'(x^*))^2} = 0,$$

and so the condition (2.6) is satisfied. By Theorem 2.2 there exists $\delta = \delta(x^*) > 0$ such that if $x_0 \in [x^* - \delta, x^* + \delta]$, then Newton's method converges. Moreover, computing the second derivative at x^* yields

$$\varphi''(x^*) = \frac{f''(x^*)}{f'(x^*)} + \frac{f'''(x^*)f(x^*)}{(f'(x^*))^2} - \frac{2f(x^*)(f''(x^*))^2}{(f'(x^*))^3} = \frac{f''(x^*)}{f'(x^*)}.$$

Here we needed $f'(x^*) \neq 0$, and so $\varphi''(x^*) \neq 0$. This shows that Newton's method converges quadratically. □

Theorem 2.4 (Linear convergence of the Constant slope method)**.** *Let f be a continuous function such that $f'(s)$ is continuous near x^*, and let g be a non-zero constant such that*

$$\left|1 - \frac{f'(x^*)}{g}\right| < 1. \tag{2.8}$$

Then, there exists a constant $\delta = \delta(x^*) > 0$ such that, if the initial guess x_0 satisfies $|x_0 - x^*| \leq \delta$, the sequence $(x_k)_{k\geq 0}$ generated by the Constant slope method (2.5) *satisfies $|x_k - x^*| \leq \delta$ and $\lim_{k\to\infty} x_k = x^*$. Furthermore,*

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = \left|1 - \frac{f'(x^*)}{g}\right|.$$

Hence, the constant slope method converges **Q-linearly** *with rate of convergence $\rho = |1 - f'(x^*)/g|$.*

Proof. Define the iterative function

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

Then, its derivative at x^* is

$$\varphi'(x^*) = 1 - \frac{f'(x^*)}{g}.$$

By Theorem 2.2 and the hypothesis (2.8), $\varphi'(x^*) \neq 0$, and so the Constant slope method converges linearly with rate $\rho = 1 - \frac{f'(x^*)}{g}$. □

Remark 2.1. *At first glance, one may consider choosing the constant g to be equal to $f'(x^*)$ so that $\rho = 1 - \frac{f'(x^*)}{g} = 0$, leading to superlinear convergence. However, this requires knowledge about the true solution x^* which is usually not available.*

2.5.1 The Secant method

The Secant method arises from choosing g_k as

$$g_k = \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}, \tag{2.9}$$

in the quasi-Newton's method

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

leading to

$$\boxed{x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})} = x_k - \frac{(x_k - x_{k-1}) f(x_k)}{f(x_k) - f(x_{k-1})}, \quad k = 1, 2, \ldots.} \tag{2.10}$$

One expects that at the beginning x_k and x_{k-1} may not be very close, so g_k is not a very accurate approximation of $f'(x_k)$ and the convergence of the secant method should be slower than that of Newton's method. But as x_k gets more and more accurate, g_k approximates $f'(x_k)$ more and more accurately, then the convergence of the secant method should be close to the Newton's method.

In order to study the convergence of the secant method, we introduce the **two-point** iterative function:

$$\varphi(u,v) = u - \frac{f(u)(u-v)}{f(u)-f(v)} = \frac{vf(u)-uf(v)}{f(u)-f(v)},$$

then the secant method can be written in the form:

$$x_{k+1} = \varphi(x_k, x_{k-1}).$$

Note that φ is not defined for the case $u = v$. However, we can define

$$\varphi(u,u) = u - \frac{f(u)}{f'(u)},$$

and so the Secant method reduces to Newton's method for the case $x_k = x_{k-1}$. In particular, $\varphi(x^*, x^*) = x^*$.

Theorem 2.5 (Convergence of the secant method)**.** *Let f be a continuous function such that $f''(s)$ is continuous near x^*, $f'(x^*) \neq 0$ and $f''(x^*) \neq 0$. Then, there exists a constant $\delta = \delta(x^*) > 0$ such that, if the initial guesses x_0 and x_1) satisfy $|x_0 - x^*| \leq \delta$ and $|x_1 - x^*| \leq \delta$, the sequence $(x_k)_{k\geq 0}$ generated by the Secant method* (2.10) *satisfies $|x_k - x^*| \leq \delta$ and $\lim_{k\to\infty} x_k = x^*$ at least* **R-linearly**. *Furthermore, the order of convergence is $p = (1+\sqrt{5})/2 \approx 1.618$.*

Proof. [2] Without loss of generality, suppose $\alpha := f'(x^*) > 0$. By continuity of f', there exists $\delta > 0$ such that for $x \in [x^* - \delta, x^* + \delta]$, it holds that

$$\frac{3\alpha}{4} \leq f'(x) \leq \frac{5\alpha}{4}.$$

By the mean value theorem we can find φ_k lying in between x_k and x_{k-1} such that

$$f(x_k) - f(x_{k-1}) = f'(\varphi_k)(x_k - x_{k-1}),$$

and similarly, we can find θ_k lying in between x_k and x^* such that

$$f(x_k) - f(x^*) = f(x_k) = f'(\theta_k)(x_k - x^*).$$

[2] The convergence proof is taken from `http://www.math.usm.edu/lambers/mat772/fall10/lecture4.pdf`.

Hence, we see that

$$\begin{aligned}
x_{k+1} - x^* &= x_k - x^* - \frac{(x_k - x_{k-1})f(x_k)}{f(x_k) - f(x_{k-1})} = x_k - x^* - \frac{f(x_k)}{f'(\varphi_k)} \\
&= x_k - x^* - \frac{f'(\theta_k)(x_k - x^*)}{f'(\varphi_k)} = \Big(1 - \frac{f'(\theta_k)}{f'(\varphi_k)}\Big)(x_k - x^*).
\end{aligned}$$

Therefore, if x_{k-1} and x_k belong to $[x^* - \delta, x^* + \delta]$, then

$$|x_{k+1} - x^*| \leq \max\Big(\left|1 - \frac{5\alpha/4}{3\alpha/4}\right|, \left|1 - \frac{3\alpha/4}{5\alpha/4}\right| \Big) |x_k - x^*| \leq \frac{2}{3} |x_k - x^*| .$$

From this we deduce two things. The first is that x_{k+1} also belong to $[x^* - \delta, x^* + \delta]$, and that the error $e_{k+1} = x_{k+1} - x^*$ satisfies

$$|e_{k+1}| \leq \frac{2}{3} |e_k|$$

which implies the (R-linear) convergence of the Secant method. To improve our current estimate on the order of convergence, we use (2.10) to write

$$\begin{aligned}
e_{k+1} = x_{k+1} - x^* &= \frac{x_{k-1}f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})} - \frac{x^* f(x_k) - x^* f(x_{k-1})}{f(x_k) - f(x_{k-1})} \\
&= \frac{f(x_k)e_k - f(x_{k-1})e_{k-1}}{f(x_k) - f(x_{k-1})} \\
&= e_k e_{k-1} \frac{\left(\frac{f(x_k)}{e_{k-1}} - \frac{f(x_{k-1})}{e_k} \right)}{f(x_k) - f(x_{k-1})} \\
&= e_k e_{k-1} \frac{\left(\frac{f(x_k) - f(x^*)}{x_k - x^*} - \frac{f(x_{k-1}) - f(x^*)}{x_{k-1} - x^*} \right)}{f(x_k) - f(x_{k-1})}.
\end{aligned}$$

We define

$$F(s) = \frac{f(s) - f(x^*)}{s - x^*}.$$

Then, by the mean value theorem there exists ζ_k between x_{k-1} and x_k such that

$$F(x_k) - F(x_{k-1}) = F'(\zeta_k)(x_k - x_{k-1}),$$

with

$$F'(s) = \frac{(s - x^*)f'(s) - f(s) + f(x^*)}{(s - x^*)^2}.$$

By Taylor's theorem, there exists ν_k between x^* and ζ_k such that

$$f(x^*) = f(\zeta_k) + f'(\zeta_k)(x^* - \zeta_k) + \frac{1}{2} f''(\nu_k)(x^* - \zeta_k)^2.$$

Then, combining the above yields

$$F(x_k) - F(x_{k-1}) = \frac{2}{f''(\nu_k)}(x_k - x_{k-1}),$$

and so

$$e_{k+1} = e_k e_{k-1} \frac{F(x_k) - F(x_{k-1})}{f(x_k) - f(x_{k-1})} = \frac{2}{f''(\nu_k)} \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} = \frac{2f'(\theta_k)}{f''(\nu_k)}$$

for some θ_k in between x_k and x_{k-1}. The assumptions on f and the fact that $\{x_k\}_{k\in\mathbb{N}}$ belong in $[x^* - \delta, x^* + \delta]$ ensure that we can find a constant M such that

$$|e_{k+1}| \leq M \, |e_k| \, |e_{k-1}| \, .$$

Suppose there are constants $C_n > 0$ and $p > 0$ such that

$$|e_{n+1}| = C_n \, |e_n|^p \text{ for all } n \in \mathbb{N}, \text{ and } C_n \to C \text{ as } n \to \infty.$$

Then, we find that

$$C_k \, |e_k|^p = |e_{k+1}| \leq M \, |e_k| \; |e_{k-1}| \, ,$$

and so

$$C_k C_{k-1}^{p-1} \, |e_{k-1}|^{p(p-1)} = C_k \, |e_k|^{p-1} \leq M \, |e_{k-1}| \, .$$

Hence, comparing the exponents on $|e_{k-1}|$ shows that $p(p-1) = 1$. Taking the positive root yields $p = \frac{1}{2}(1 + \sqrt{5})$.

$\square$

3 Floating-point arithmetic

3.1 Decimal and binary numbers

The decimal system expresses any number in powers of 10:

$$538.372 = 5 \times 10^2 + 3 \times 10^1 + 8 \times 10^0 + 3 \times 10^{-1} + 7 \times 10^{-2} + 2 \times 10^{-3}.$$

Meanwhile computers adapt the binary system, which expresses any number in powers of 2. In contrast, only two digits 0 and 1 are used. For example, the real number 9.90625 in the binary form is

$$\begin{aligned}(1001.11101)_2 &= 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} \\ &\quad + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}.\end{aligned}$$

Another example is

$$2.125 = 1 \times 2^1 + 1 \times 2^{-3} = (1.001)_2.$$

Computers communicate with the human users in the decimal system but work internally in the binary system. Conversions from decimal to binary and back to decimal often take place within calculations. Since computers can only operate with a fixed number of digits, some numbers like 1/10 cannot be stored exactly in any binary computer as its binary representation has an infinite number of binary digits:

$$\frac{1}{10} = (0.0001\,1001\,1001\,1001\ \ldots)_2.$$

Hence, it is expected that some errors will occur during the binary-decimal conversion.

3.2 Normalized scientific notation

In the decimal system, one can express any real number in normalized scientific notation. For example, we can write

$$2048.6076 = 0.20486076 \times 10^4, \quad -0.0004321 = -0.4321 \times 10^{-3}.$$

Definition 3.1. *The normalized decimal form of a nonzero real number a is*

$$a = \pm r \times 10^n, \tag{3.1}$$

for some real number $r \in [0.1, 1)$ called the **significand** *or the* **mantissa***, and some integer n called the* **exponent***.*

Remark 3.1. *Some people consider r to be in the range $1 \le r < 10$, and the effect is simply reducing the exponent n by 1.*

Computers use 2 formats for numbers:

- **Fixed-point** numbers - which has a specific number of bits (or digits) reserved for the integer part (the part to the left of the decimal point) and a specific number of bits reserved for the fractional part (the right of the decimal point).
- **Floating point** numbers - which does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the mantissa/significand and a certain number of bits for the exponent.

Example 3.1. *With 8 digits, we consider the fixed point format to be (IIII.FFFF). Then, the range of numbers we can represent is* $[0000.0001, 9999.9999]$. *In contrast, if we use the format (IIIIIII.F), then the range we can represent shifts to [0000000.1, 9999999.9]. While the maximum has increased by a thousand folds, so has the minimum, and this comes as a cost of accuracy (since we only allow for 1 decimal place).*

Example 3.2. *Suppose now we allocate 2 digits for the exponent, and the remaining 6 digits for the mantissa, then the floating point format (.xxxxxx* $\times 10^{\pm xx}$*) gives us the range from* 10^{-99} *to* 10^{99}*, completely dwarfing the the scale from the fixed-point format. Therefore, floating-point representation can handle much smaller and much larger numbers compared to the fixed-point representation.*

Definition 3.2. *The binary form of a nonzero real number a is*

$$a = \pm(q)_2 \times 2^{\tilde{m}}, \tag{3.2}$$

for some binary number q denoted as the **mantissa***, and some number $\tilde{m}$ denoted as the* **exponent***.*

Example 3.3. *The decimal* 11.625 *can be written as*

$$\begin{aligned} 11.625 &= (1011.1010)_2 = 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-3} \\ &= (1.0111010)_2 \times 2^{(11)_2}, \end{aligned}$$

where $(11)_2$ *is the binary representation of the integer* 3*. Furthermore, we see that shifting the binary point 1 place to the left leads*

$$11.625 = (0.10111010)_2 \times 2^{(100)_2}, \quad (100)_2 = 4.$$

The accuracy of a number represented by a computer is determined by the word length of the computer, which is the number of bits processed by the CPU in one go. Most modern computers have a word length of 32 bits or 64 bits, and the number format for a computer with word length 32 bits is denoted as the **single-precision-floating-point format**, while for 64 bits it is the **double-precision-floating-point format**. These formats have three parts:

1. One bit reserved for the sign s;
2. n bits reserved for the exponent m, where $n = 8$ for 32 bits and $n = 11$ for 64 bits;

3. r bits reserved for the mantissa q, where $r = 23$ for 32 bits and $r = 52$ for 64 bits,

so that $1 + n + r$ is equal to the word length of the computer. Then, the storage format

$$s|m_1 \dots m_n|f_1 \dots f_r, \text{ where } s, m_1, \dots, m_n, f_1, \dots, f_r \in \{0, 1\},$$

represents the number

$$a = (-1)^s \tilde{q} \times 2^{\tilde{m}}, \quad \tilde{q} = (q)_2 = (1.f_1 \dots f_r)_2, \quad \tilde{m} = (m_1 \dots m_n)_2 - y$$

where y is an integer depending only on the word length.

Let us discuss the single-precision format in more detail, which uses a 32 bit word length with 1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa.

- The entry just before the binary point in the mantissa is set to 1. The reason is to save storage and so all 23 bits can be used for the fractional part $f_1 \dots f_r$ after the binary point. In this way, the mantissa $\tilde{q}$ belongs to the interval $[1, 2)$.

- We say that a can be expressed as a **normalized binary number** if in its mantissa, there is a 1 in front of the binary point. If the entry before the binary point is zero, then we call a a **denormalized binary number** - more on this later.

- The 8 bit number $(m_1 \dots m_8)_2$ ranges from 0 (given by $(00000000)_2$) to 255 (given by $(11111111)_2$), containing 256 possibilities. If $y = 0$, then $\tilde{m}$ can never be negative and we lost the ability to represent tiny numbers. Therefore, an offset y is needed so that there are an equal number of choices for non-negative exponents and for negative exponents.

- The cases $(m_1 \dots m_8)_2 = 0$ and $(m_1 \dots m_8)_2 = 255$, corresponding to the cases where all coefficients m_i are zero or one, are reserved for special values such as 0 and ∞. Excluding these two leaves us with 254 possibilities, and divide this number by two yields the offset $y = 127$.

- The number $m = \tilde{m} - 127$ is hence known as the **biased exponent**.

Hence, in single-precision normalized format a real number is represented as

$$a = (-1)^s (1.f_1 \dots f_{23})_2 \times 2^{(m_1 \dots m_8)_2 - 127}. \tag{3.3}$$

Example 3.4. *The number* $2.125 = (10.001)_2$ *is written as*

$$\begin{aligned}(10.001)_2 &= (-1)^0 (1.0001\underbrace{00\dots00}_{19})_2 \times 2^{128-127} \\ &= (-1)^0 (1.0001\underbrace{00\dots00}_{19})_2 \times 2^{(1000000)_2 - 127},\end{aligned}$$

so that its computer representation is:

$$0|10000000|0001\underbrace{00\dots00}_{19}.$$

In computer systems there are certain special values, such as 0, ∞ and NaN (not-a-number). The latter occurs during invalid floating-point arithmetic operations, such as the square root of a negative number, or computing $0/0$ or $\infty - \infty$. The format for these special values are given as follows:

- Zero is represented as

$$s|00000000|\underbrace{0000\ldots0000}_{23}$$

  where $s = 0$ gives $+0$ and $s = 1$ gives -0.

- Infinity is represented as

$$s|11111111|\underbrace{0000\ldots0000}_{23}$$

  where $s = 0$ gives $+\infty$ and $s = 1$ gives $-\infty$.

- NaN is represented as

$$s|11111111|f_1\ldots f_{23}$$

  where at least one of the f_i is not zero.

Remark 3.2. *From the three special values we see that there remains the possibility*

$$s|00000000|f_1\ldots f_{23}$$

where at least one of the f_i is not zero. It turns out that these formats are reserved for so-called **denormalized numbers***, which are represented as*

$$a = (-1)^s(0.f_1\ldots f_{23})_2 \times 2^{-127}.$$

The difference between normalized and denormalized numbers is that the former has 1 before the binary point and the latter has 0 before the binary point. Also, denormalize numbers always have exponent -127. These numbers exist to fill in the gaps between the zero value and the smallest normalized positive floating-point number.

In a similar fashion, for 64 bit format, i.e., double-precision, a number is stored as

$$s|m_1\ldots m_{11}|f_1\ldots f_{52}, \text{ where } s, m_1, \ldots, m_{11}, f_1, \ldots, f_{52} \in \{0,1\}.$$

Since $(m_1\ldots m_{11})_2$ has 2048 possibilities, and if we exclude the case of all zeros and the case of all one for special values, we are left with 2046 possibilities, and so the offset y should be equal to 1023 in order to have an equal number of non-negative and negative exponents. Hence, in double-precision normalized format a real number is represented as

$$a = (-1)^s(1.f_1\ldots f_{52})_2 \times 2^{(m_1\ldots m_{11})_2-1023}. \tag{3.4}$$

Again, NaN, 0, ∞ and denormalized numbers reserve the slots $(000\ldots000)_2$ and $(111\ldots111)_2$ for the exponent as described above.

Let us now consider looking at the smallest positive and largest finite normalized numbers in both single-precision and double-precision format to give a sense of their differences. In single-precision, the smallest normalized positive number is

$$\begin{aligned}&0|00000001|\underbrace{000\cdots000}_{23}\\ \mapsto&(1.\underbrace{00\cdots00}_{23})_2\times 2^{1-127}=2^{-126}\approx 1.175\times 10^{-38},\end{aligned}\tag{3.5}$$

and the largest finite normalized positive number is

$$\begin{aligned}&0|11111110|\underbrace{111\cdots111}_{23}\\ \mapsto&(1.\underbrace{111\cdots111}_{23})_2\times 2^{254-127}=(2-2^{-23})\times 2^{127}\approx 3.4\times 10^{38}.\end{aligned}\tag{3.6}$$

Meanwhile the smallest normalized positive number in double-precision format is

$$\begin{aligned}&0|\underbrace{00\cdots00}_{10}1|\underbrace{00\cdots00}_{52}\\ \mapsto&(1.\underbrace{00\cdots00}_{52})_2\times 2^{1-1023}=2^{-1022}\approx 2.225\times 10^{-308},\end{aligned}\tag{3.7}$$

while the largest finite normalized number in double-precision format is

$$\begin{aligned}&0|\underbrace{11\cdots11}_{10}0|\underbrace{11\cdots11}_{52}\\ \mapsto&(1.\underbrace{11\cdots11}_{52})_2\times 2^{2046-1023}=(2-2^{-52})\times 2^{1023}\approx 1.798\times 10^{308}.\end{aligned}\tag{3.8}$$

Definition 3.3. *A real number which can be represented as the normalized floating-point form* (3.3) (*resp.* (3.4)) *is called a* **machine number** *in the computer having a word length of 32* (*resp. 64*) *bits/binary digits.*

In particular, a machine number can be represented precisely by the computer. But this is not the case for most real numbers. If a real number x is not a machine number, it will be approximately by a most accurate machine number, usually denoted as $\mathrm{fl}(x)$.

3.3 Rounding errors

Consider a decimal x with m digits after the decimal point. For $n < m$, if a machine only has n degrees of precision, i.e., it can only represent decimals with n digits after the decimal point, then we have to obtain an approximation $\hat{x}$ of x consisting of only n decimal places. There are two ways to get an approximation:

1. **Rounding**: If the $(n+1)$th digit is 0, 1, 2, 3, or 4, then the nth digit is unchanged and all following digits are discarded. If the $(n+1)$th digit is 5, 6, 7, 8, or 9, then the nth digit is increased by one unit (carried leftwards if necessary) and the remaining digits are discarded.

2. **Chopping**: Simply discard all digits beyond the nth digit.

Example 3.5. *The following 7 digit numbers are rounded to 4 digits:*

$$\begin{aligned} 0.1735499 &\mapsto 0.1735 \\ 0.9999500 &\mapsto 1.000 \\ 0.4321609 &\mapsto 0.4322. \end{aligned}$$

If they are chopped to 4 digits then:

$$\begin{aligned} 0.1735499 &\mapsto 0.1735 \\ 0.9999500 &\mapsto 0.9999 \\ 0.4321609 &\mapsto 0.4321. \end{aligned}$$

When we perform either rounding or chopping, an error is produced. The following quantifies the relative error produced for each operation.

Lemma 3.1. *Let $x = a.a_1a_2\dots a_n\dots$ be a decimal with $a \neq 0$ (i.e., $|x| \geq 1$). Denoting the rounded number x_r with n digits after the decimal point and the chopped number x_c with n digits after the decimal point. Then, the following relative error estimate holds*

$$\frac{|x - x_r|}{|x|} \leq \frac{1}{2} \times 10^{-n}, \qquad \frac{|x - x_c|}{|x|} \leq 10^{-n}.$$

In particular the relative error for chopping is at most twice as large as the relative error for rounding.

Proof. Consider the rounded number x_r. Then if the $(n+1)$th digit of x is 0,1,2,3 or 4, then $x = x_r + \varepsilon$ for some $\varepsilon < 0.5 \times 10^{-n}$. Otherwise, if the $(n+1)$th digit of x is 5,6,7,8 or 9, then we can write $x_r = y + 10^{-n}$ for some number y of the form

$$y = a.a_1a_2\dots a_n00000\dots,$$

i.e., y agrees with x up to the nth digit and all other digits of y beyond the nth digit are zero. Then, $x = y + \delta \times 10^{-n}$ for some $\delta > 0.5$. Thus $x - x_r = (1-\delta) \times 10^{-n} \leq 0.5 \times 10^{-n}$. Since $|x| \geq 1$ (otherwise a would be zero), this leads to the relative error

$$\frac{|x - x_r|}{|x|} \leq \frac{1}{2} \times 10^{-n}.$$

Consider now the chopped number x_c, which is obtained by discarding all digits after the nth one. Then, $x = x_c + \delta \times 10^{-n}$ for some $0 \le \delta < 1$. This yields the relative error

$$\frac{|x - x_c|}{|x|} \le |\delta| \times 10^{-n} \le 10^{-n}.$$

□

For binary numbers a similar notion of rounding and chopping exist:

1. **Rounding**: If the $(n+1)$th digit is 0 then the nth digit is unchanged and all following digits are discarded. If the $(n+1)$th digit is 1, then the nth digit is increased by one unit (carried leftwards if necessary) and the remaining digits are discarded.

2. **Chopping**: Simply discard all digits beyond the nth digit.

More precisely, if $x = (x_0.x_1x_2 \dots x_n \dots)_2$, then

$$x_c = (x_0.x_1x_2 \dots x_n)_2, \text{ and } x_r = \begin{cases} (x_0.x_1x_2 \dots x_n)_2 & \text{if } x_{n+1} = 0, \\ (x_0.x_1x_2 \dots x_n)_2 + 2^{-n} & \text{if } x_{n+1} = 1. \end{cases}$$

Example 3.6. *The number* $3/5 = (0.1001100110011\dots)_2 = (0.\overline{1001})_2$ *rounded to 5 digits is* $x_r = (0.10011)_2$ *and rounded to 4 digits is* $x_r = (0.1010)_2$.

Lemma 3.2. *Let* $x = (a.a_1a_2 \dots a_n \dots)_2$ *be a binary number with* $a = 1$. *Denoting the rounded number* x_r *with n digits after the binary point and the chopped number* x_c *with n digits after the binary point. Then, the following relative error estimate holds*

$$\frac{|x - x_r|}{|x|} \le 2^{-(n+1)}, \quad \frac{|x - x_c|}{|x|} \le 2^{-n}.$$

Proof. The proof is similar to the one for decimals. For the rounded binary number x_r, if the $(n+1)$th digit is 0, then $x = x_r + \varepsilon$ with $\varepsilon < 2^{-(n+1)}$. Otherwise, writing $x_r = y + 2^{-n}$ where as before, $y = (a.a_1a_2 \dots a_n000000\dots)_2$, so that $x = y + \delta \times 2^{-(n+1)}$ for $\delta > 1$. Then, $x - x_r = (1 - \delta) \times 2^{-(n+1)} \le 2^{-(n+1)}$. These cases yield the relative error estimate for x_r. Similarly for the chopped binary number x_c, we infer that $x = x_c + \delta \times 2^{-n}$ for some $0 \le \delta < 1$. □

3.4 Machine precision

In a 32-bit representation, the restriction that the mantissa part occupies no more than 23 bits means that the machine numbers have a limited precision of roughly 6 decimal places, since the least significant bit in the mantissa represents units of 2^{-23}, approximately $1.2 \times 10^{-7} \le 10^{-6}$. Then numbers expressed with more than 6 decimal digits will be approximated by machine numbers.

Recalling (3.5), the smallest positive number that can be represented in 32-bit is $a_{\min} = 2^{-126}$, while the second smallest positive number that can be represented in 32-bit is:

$$a_{\min,2} = (1.\underbrace{00\ldots00}_{22}1)_2 \times 2^{-126},$$

so that the maximum relative error that one can made in truncating (or rounding) a number a between $a_{\min}$ and $a_{\min,2}$ is:

$$\left|\frac{a_{\min} - a}{a}\right| \leq \frac{(0.\underbrace{00\ldots00}_{22}1)_2 \times 2^{-126}}{1 \times 2^{-126}} = 2^{-23} = 10^{-6.9},$$

where in the above we used

$$|a_{\min} - a| \leq |a_{\min} - a_{\min,2}|, \quad \frac{1}{|a|} \leq \frac{1}{|a_{\min}|}.$$

This means that the number of significant digits retained is roughly equal to 7 when one truncates a very small number.

Similarly, recalling (3.6), the largest number $a_{\max}$ that can be represented in 32-bit is:

$$a_{\max} = (1.\underbrace{11\ldots1}_{23})_2 \times 2^{127} = (2 - 2^{-23}) \times 2^{127} \approx 2^{128},$$

while the second largest number that can be represented in 32-bit is:

$$a_{\max,2} = (1.\underbrace{11\ldots1}_{22}0)_2 \times 2^{127} = (2 - 2^{-22}) \times 2^{127}.$$

The difference between the two numbers is huge:

$$|a_{\max} - a_{\max,2}| = (0.\underbrace{00\ldots00}_{22}1)_2 \times 2^{127} = 2^{-23} \times 2^{127} = 2^{103} \approx 10^{31}.$$

However, the maximum relative error that one can made in truncating (or rounding) a number a between $a_{\max,2}$ and $a_{\max}$ is:

$$\left|\frac{a_{\max} - a}{a}\right| \leq \frac{(0.00\ldots001)_2 \times 2^{127}}{(1.11\ldots110)_2 \times 2^{127}} \leq \frac{2^{-23} \times 2^{127}}{(2 - 2^{-22}) \times 2^{127}} \leq 2^{-23} = 10^{-6.9},$$

where we used $2 - 2^{-22} > 1$. That again means that the number of significant digits retained is roughly equal to 7 when one truncates a very large number.

In general, the number of significant digits retained when one truncates any number in 32-bit representation is always 7 (because we use 23 bits for the mantissa and

$2^{-23} = 10^{-6.9}$). The number 2^{-23} is called **unit roundoff error** or **machine precision**[3], and usually denoted as ε_M, which also goes by the name **machine epsilon**. Roughly speaking the machine precision is an upper bound on the relative error due to rounding. We see that the unit roundoff error depends on the length of the mantissa only. In the 64-bit machines, we use 52 bits for the mantissa, and hence the accuracy is within $2^{-52} = 10^{-15.6}$, i.e. about 16 digits of accuracy.

We may also introduce the machine epsilon in a slightly different manner. Suppose we like to estimate the error involved in approximating a given positive real number x by a nearby machine number in the 32-bit system. We assume that

$$x = q \times 2^m \,, \quad \frac{1}{2} \le q < 1 \,, \quad |m| \le 127,$$

where in the above we used the unbiased exponent for m.

The real number x will be approximated by the closest machine number. Write

$$x = (1.a_1 a_2 \ldots a_{23} a_{24} a_{25} \ldots)_2 \times 2^m,$$

where each a_s is either 0 or 1. One nearby machine number is obtained by simply discarding the excess bits $a_{24}a_{25}\ldots$. The resulting number is

$$x_- = (1.a_1 a_2 \ldots a_{23})_2 \times 2^m.$$

Observe that x_- lies to the left of x on the real line. Another nearby machine number lies to the right of x, which is obtained by rounding up. That is, we drop the excess bits as before, but increase the last remaining bit a_{23} by one unit. This number is

$$x_+ = ((1.a_1 a_2 \ldots a_{23})_2 + 2^{-23})_2 \times 2^m.$$

The closer of x_- or x_+ is chosen to represent x in the computer. There are two situations. If x is represented better by x_-, then we have

$$|x - x_-| \le \frac{1}{2}|x_+ - x_-| = \frac{1}{2} \times 2^{-23} \times 2^m = 2^{m-24}.$$

In this case, the relative error is bounded as follows:

$$\left|\frac{x - x_-}{x}\right| \le \frac{2^{m-24}}{q \times 2^m} = \frac{2^{-24}}{q} \le 2^{-23}.$$

Similarly, in the second case, we have the relative error is also bounded by 2^{-23}.

In general, if x is a nonzero real number within the range of the machine, then, the machine number x^* closest to x satisfies the inequality

$$\left|\frac{x - x^*}{x}\right| \le 2^{-23}.$$

The number 2^{-23} is called unit roundoff error or machine epsilon.

[3]Note that other people may also define 2^{-24} as the machine precision

3.5 Overflow and underflow

From the definition of floating-point numbers, there are **upper** and **lower** limits for the magnitudes of the numbers that can be represented. Any attempts to create numbers

- that are too small, e.g. $\pm q \times 2^{-c}$ for $c > 127 \Rightarrow$ **underflow** errors: the default option is to set the number to zero and proceed/continue;
- that are too large, e.g. $\pm q \times 2^{c}$ for $c > 127 \Rightarrow$ **overflow** errors: these generate **fatal** errors on most computers, and computation will be halted.

Example 3.7. *Consider evaluating the function $f(x) = x^{10}$ for x near 0. With single precision arithmetic, the smallest nonzero positive number is*

$$a = 2^{-126},$$

and so $f(x)$ is set to zero if

$$x^{10} < a \quad \Leftrightarrow \quad |x| < a^{1/10} \approx 1.610 \times 10^{-4}.$$

Since overflow is usually a fatal error that causes many systems to stop with an error message, with proper scaling overflows can be eliminated at a cost of generating harmless underflows.

Example 3.8. *Consider the problem of computing*

$$c = \sqrt{a^2 + b^2}, \quad a = 10^{60}, \quad b = 1.$$

The computation may overflow while computing a^2. The remedy is to re-express c in the form

$$c = s\sqrt{\left(\frac{a}{s}\right)^2 + \left(\frac{b}{s}\right)^2},$$

where s is a suitable scaling factor, say

$$s = \max\{|a|, |b|\} = 10^{60}.$$

Then,

$$c = 10^{60}\sqrt{1^2 + \left(\frac{1}{10^{60}}\right)^2},$$

so that when $(1/10^{60})^2$ is computed, it underflows and is set to zero. The computation still goes forward and we contain $c = 10^{60}$.

3.6 Loss of significance

In general, the result of some operation of two floating-point number can not be represented by a floating-point numbers of the same size. For example, the product of two five-digit numbers (after the decimal point) will often require ten digits for its result. Another example is

$$(2.0001 \times 10^6) \times (9.0001 \times 10^2) = 18.00110001 \times 10^8.$$

Thus the result of a floating-point operation can be represented only approximately.

Let us look at a simple example. Suppose we are computing the difference $1 - 0.9999999$ in six-digit decimal arithmetic. We first align the numbers:

$$\begin{array}{rr} & 1.0000000 \\ - & 0.9999999 \\ \hline \end{array}$$

Mathematically, we have

$$\begin{array}{rr} & 1.0000000 \\ - & 0.9999999 \\ \hline & 0.0000001 \end{array}$$

Normalizing the result to the correct answer, we get

$$0.100000 \times 10^{-6}.$$

But the computer has only 6-digit decimal operation available now, so during the alignment, it would round 0.9999999 to 1.00000 and thus this leads to

$$\begin{array}{rr} & 1.00000 \\ - & 1.00000 \\ \hline & 0.00000 \end{array}$$

which gives the result zero!

This phenomenon we see is called **loss of significance error**, which often occurs when we take the difference between two nearly equal numbers. Consider another example: evaluate the function

$$f(x) = \sqrt{x+1} - \sqrt{x}$$

at $x = 100$ to 6 significant digits[4]. We have

$$\sqrt{100} = 10.0000 \quad \text{(exact)}, \qquad \sqrt{101} = 10.0499 \quad \text{(rounded)},$$

where we round $\sqrt{101}$ correctly to 6 significant digits of accuracy. Then, we get

$$\sqrt{101} - \sqrt{100} = 0.0499000$$

[4]See `https://en.wikipedia.org/wiki/Significant_figures` for the definition.

whereas the true value to 6 significant digits should be 0.0498756! In particular, four digits of accuracy (since the approximate value 0.0499000 and the true value 0.0498756 differ in 4 digits) are lost during the evaluation of $\sqrt{101} - \sqrt{100}$.

Another example: evaluate

$$f(x) = \log_{10}(x+1) - \log_{10}(x)$$

at $x = 9$ to 5 significant digits. We see that

$$\log_{10}(9+1) = 1.0000 \quad \text{(exact)}, \quad \log_{10}(9) = 0.95424 \quad \text{(rounded)},$$

and so

$$f(9) = 0.045760$$

to 5 significant digits. But the true value is 0.045757 to 5 significant digits. Hence, we have lost two significant digits of accuracy. In fact, if x is larger, this loss of significance may become more severe!

For example, take the same function $f(x)$ as above and evaluate at $x = 999$. Then, to 5 significant digits,

$$\log_{10}(999+1) = \log_{10}(1000) = 3 \quad \text{(exact)}, \quad \log_{10}(999) = 2.9995 \quad \text{(rounded)},$$

which leads to the evaluation

$$f(999) = 0.00050000$$

to 5 significant digits. But the true value correct to 5 significant digits is 0.00043451, so we have lost all significant digits in this computation.

This loss of accuracy is a by-product of

- the type of calculations involved in evaluating the function $f(x)$;
- (mainly) the finite precision decimal arithmetic being used.

The overall theme is that when two nearly equal quantities are subtracted, this leads to significant lost of accuracy. More often, the loss of significance can be subtle and difficult to detect. However, loss of significance in single precision can be avoided using double precision, as the number of bits in the mantissa is doubled. In other cases, rewriting the mathematical formula is an alternative and can make a difference. For example, writing

$$f(x) = \log_{10}\left(\frac{x+1}{x}\right) = \log_{10}\left(1 + \frac{1}{x}\right)$$

and substituting $x = 9$, we obtain to 5 significant digits

$$f(x) = 0.046757$$

which agrees with the true value to 5 significant digits. Similarly, for $x = 999$

$$f(999) = \log_{10}(1.0010) = 0.00043407$$

and we have only lost 2 significant digits, which is a huge improvement.

Remark 3.3. *Loss of significance can have real life implications*[5]. *One example is the failure of intercepting of an incoming missile which killed 28 soldiers at Dhahran, Saudi Arabia in 1991*[6].

3.7 Error analysis

The basic problem is that not all real numbers can be exactly represented as machine numbers, and during standard arithmetic operations such as $+$, $-$, $\times$, $\div$, the usual rules of arithmetic may no longer hold, i.e.,

$$a + (b + c) \neq (a + b) + c$$

due to rounding. Given a real number x, let $\mathrm{fl}(x)$ be the floating point representation of x, i.e., $\mathrm{fl}(x)$ is a machine representable number closest to x. By previous discussions, we have

$$\left|\frac{\mathrm{fl}(x) - x}{x}\right| \leq \varepsilon_M = \begin{cases} 2^{-23} \text{ for single precision,} \\ 2^{-52} \text{ for double precision.} \end{cases} \tag{3.9}$$

Then, there exists $\varepsilon = \varepsilon(x)$ such that

$$\mathrm{fl}(x) = x(1 + \varepsilon) \text{ with } |\varepsilon| \leq \varepsilon_M. \tag{3.10}$$

Assume that when 2 machine numbers are operated arithmetically, namely using the operations $+, -, \times, \div$, the operation is first correctly formed, then normalized, rounded off and stored in a machine word. To make it clear, let $\odot$ stand for any of the four basic arithmetic operations. If x and y are machine numbers, then $x \odot y$ is correctly formed, and $\mathrm{fl}(x \odot y)$ is stored. If not, then $\mathrm{fl}(x) \odot \mathrm{fl}(y)$ is correctly formed, and $\mathrm{fl}(\mathrm{fl}(x) \odot \mathrm{fl}(y))$ is stored.

Then, it is desirable to know how accurate is the value $\mathrm{fl}(x \odot y)$ or $\mathrm{fl}(\mathrm{fl}(x) \odot \mathrm{fl}(y))$ compared to the true value $x \odot y$, or alternatively, what can we say regarding an upper bound for the relative errors

$$\left|\frac{\mathrm{fl}(x \odot y) - x \odot y}{x \odot y}\right| \text{ and } \left|\frac{\mathrm{fl}(\mathrm{fl}(x) \odot \mathrm{fl}(y)) - x \odot y}{x \odot y}\right|.$$

The main idea is to use the relation (3.9). What we will perform in the following examples is **forward error analysis**, which measures the differences between the approximation $\mathrm{fl}(x \odot y)$ and the true value $x \odot y$.

Example 3.9 (Addition). *Let x and y be two positive machine numbers, then*

$$\mathrm{fl}(x + y) = [x + y](1 + \varepsilon_1) \textit{ where } |\varepsilon_1| \leq \varepsilon_M,$$

[5]See `https://www.student.cs.uwaterloo.ca/~se101/Float.pdf`.

[6]See `https://en.wikipedia.org/wiki/MIM-104_Patriot#Failure_at_Dhahran`.

and so the forward relative error is

$$\left|\frac{\mathrm{fl}(x+y)-(x+y)}{x+y}\right| \le \varepsilon_M.$$

If x and y are not machine representable, then

$$\begin{aligned}\mathrm{fl}(\mathrm{fl}(x)+\mathrm{fl}(y)) &= [\mathrm{fl}(x)+\mathrm{fl}(y)](1+\varepsilon_1)\\ &= [x(1+\varepsilon_2)+y(1+\varepsilon_3)](1+\varepsilon_1) \text{ where } |\varepsilon_i| \le \varepsilon_M \text{ for } i=1,2,3\\ &= [x+y+\varepsilon_2 x+\varepsilon_3 y](1+\varepsilon_1),\end{aligned}$$

and so

$$|\mathrm{fl}(\mathrm{fl}(x)+\mathrm{fl}(y))-(x+y)| \le 2\varepsilon_M\,|x+y|\,,$$

since the products $\varepsilon_1\varepsilon_2, \varepsilon_1\varepsilon_3$ are smaller compared to ε_1, ε_2, and ε_3, so that they most likely will underflow and be set to zero. Hence, the forward relative error is bounded by $2\varepsilon_M$, and this tells us that when we add two positive non-machine numbers, the relative error at most doubles.

Example 3.10 (Difference of squares)**.** *Consider evaluating $r = a^2 - b^2$ for two machine numbers a and b. Let $\hat{r} := \mathrm{fl}(\mathrm{fl}(a^2)-\mathrm{fl}(b^2))$, then*

$$\hat{r} = [\mathrm{fl}(a^2)-\mathrm{fl}(b^2)](1+\varepsilon_1) = (a^2(1+\varepsilon_2)-b^2(1+\varepsilon_3))(1+\varepsilon_1),$$

with $|\varepsilon_i| \le \varepsilon_M$ for $i=1,2,3$. So

$$|\hat{r}-r| = \left|a^2(\varepsilon_1+\varepsilon_2+\varepsilon_1\varepsilon_2)-b^2(\varepsilon_1+\varepsilon_3+\varepsilon_1\varepsilon_3)\right| \le 2\varepsilon_M(a^2+b^2),$$

which implies that the forward relative error is

$$\left|\frac{(\mathrm{fl}(a^2)-\mathrm{fl}(b^2))-(a^2-b^2)}{a^2-b^2}\right| \le 2\varepsilon_M\frac{a^2+b^2}{a^2-b^2} =: 2\varepsilon_M C_{a,b}.$$

We see that if $C_{a,b} = O(1)$, then the forward relative error is $O(\varepsilon_M)$ and thus the computation is accurate. However, if $C_{a,b} = O(1/\varepsilon_M)$ (which can happen if a is close to b), then the above estimate is not informative, as the forward relative error can be of order $O(1)$, rendering the computation useless for accuracy purposes.

Example 3.11 (Difference of squares - a modification)**.** *Instead of evaluating a^2-b^2, we consider evaluating $(a+b)(a-b)$, which mathematically is the same, but behaves rather differently. Then,*

$$\begin{aligned}\hat{r} &= \mathrm{fl}(\mathrm{fl}(a+b)\times\mathrm{fl}(a-b)) = [\mathrm{fl}(a+b)\times\mathrm{fl}(a-b)](1+\varepsilon_1)\\ &= [(a+b)(1+\varepsilon_2)\times(a-b)(1+\varepsilon_3)](1+\varepsilon_1)\\ &= (a+b)(a-b)(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3)\\ &= r(1+\varepsilon_1)(1+\varepsilon_2)(1+\varepsilon_3),\end{aligned}$$

where $|\varepsilon_i| \leq \varepsilon_M$ for $i = 1, 2, 3$. In particular, we have that the forward relative error is

$$\left|\frac{\hat{r} - r}{r}\right| \leq (1 + \varepsilon_M)^3 - 1 \leq 3\varepsilon_M,$$

and so the computation is always accurate regardless of the size of a and b.

Example 3.12 (Adding three numbers)**.** *Assume a, b, and c are positive machine numbers, then as the rules of arithmetic may not hold, we have two ways to construct the sum $y = a + b + c$:*

The first way is to perform the summation of a and b, and then with c, leading to

$$\begin{aligned}\hat{y}_1 := \mathrm{fl}(\mathrm{fl}(a+b)+c) &= [\mathrm{fl}(a+b)+c](1+\varepsilon_1) = [(a+b)(1+\varepsilon_2)+c](1+\varepsilon_1) \\ &= (a+b+c)\Big(1 + \frac{a+b}{a+b+c}\varepsilon_2(1+\varepsilon_1) + \varepsilon_1\Big),\end{aligned}$$

and the second way is to perform the summation of b and c, and then with a, leading to

$$\hat{y}_2 = \mathrm{fl}(a + \mathrm{fl}(b+c)) = (a+b+c)\Big(1 + \frac{b+c}{a+b+c}\varepsilon_2(1+\varepsilon_1) + \varepsilon_1\Big).$$

We neglect a third way which is to perform the summation of a and c, and then with b, as it leads to a similar relation. However, the important thing to note is that we obtain the following relative errors for the two methods:

$$\left|\frac{\hat{y}_1 - y}{y}\right| \leq \frac{a+b}{a+b+c}\varepsilon_2 + \varepsilon_1, \quad \left|\frac{\hat{y}_2 - y}{y}\right| \leq \frac{b+c}{a+b+c}\varepsilon_2 + \varepsilon_1.$$

Therefore, one obtains a smaller forward relative error if we sum the smaller numbers first.

The forward error analysis is useful as it provides the accuracy of the approximation $\hat{y}$ of an operation $y = f(x)$ for some function f, but there are certain drawbacks:

- if the true value y is unknown, then it can be difficult to compute the forward error.
- often forward error analysis gives a too pessimistic estimate on the accuracy of the operation.

We now turn to **backward error analysis**, which is concerned with the question: in the computation of the quantity $y = f(x)$, suppose we have an approximate value $\hat{y}$. Find δ such that $\hat{y} = f(x + \delta)$. In particular, we acknowledge that due to rounding errors, the calculated result $\hat{y}$ is not exactly correct, but it can be regarded as an exact solution to a nearby problem with slightly perturbed data, i.e.,

$$\begin{pmatrix}\text{approximate arithmetic} \\ \text{applied to correct data}\end{pmatrix} \Leftrightarrow \begin{pmatrix}\text{correct arithmetic} \\ \text{applied to approximate data}\end{pmatrix}.$$

Let δ be the value such that $\hat{y} = f(x+\delta)$. Then, we define

$$|\delta| \text{ as the absolute backward error,}$$
$$\frac{|\delta|}{|x|} \text{ as the relative backward error if } x \neq 0.$$

Example 3.13. *Consider $f(s) = \sqrt{s}$. For $y = \sqrt{2}$ suppose we have the approximate value $\hat{y} = 1.4$. Then, the absolute and relative forward errors are*

$$|\hat{y} - y| \approx 0.0142\ldots, \qquad \left|\frac{\hat{y}-y}{y}\right| \approx 0.01005\ldots.$$

Since $1.4 = \sqrt{1.96}$ we have that $\hat{y} = 1.4 = \sqrt{2+\delta}$ with $\delta = -0.04$. Hence, the absolute and relative backward errors are

$$|\delta| = 0.04, \qquad \left|\frac{\delta}{x}\right| = 0.02.$$

Example 3.14. [7] *Consider the evaluation of $f(s) = s^2$ at a non-machine number x. In the forward error analysis, we compute*

$$\hat{y} = \mathrm{fl}(\mathrm{fl}(x) \times \mathrm{fl}(x)) = x^2(1+\varepsilon_2)^2(1+\varepsilon_1)$$

for some $|\varepsilon_1|, |\varepsilon_2| \leq \varepsilon_M$. Then, we see that

$$\hat{y} = x^2(1 + 2\varepsilon_2 + \varepsilon_2^2 + \varepsilon_1 + 2\varepsilon_1\varepsilon_2 + \varepsilon_1\varepsilon_2^2) \quad \Rightarrow \quad \left|\frac{\hat{y}-x^2}{x^2}\right| \lesssim 3\varepsilon_M,$$

which means that the forward relative error is approximately tripled. In the backward error analysis, we start from the relation

$$\hat{y} = x^2(1 + 2\varepsilon_2 + \varepsilon_2^2 + \varepsilon_1 + 2\varepsilon_1\varepsilon_2 + \varepsilon_1\varepsilon_2^2).$$

Then, we can find δ with $|\delta| \lesssim \varepsilon_M$ such that

$$\hat{y} = x^2(1 + 2\varepsilon_2 + \varepsilon_2^2 + \varepsilon_1 + 2\varepsilon_1\varepsilon_2 + \varepsilon_1\varepsilon_2^2) = x^2(1+\delta)^2 = (x(1+\delta))^2 = f(x(1+\delta)).$$

This result says that the error in squaring a non-machine number x is no worse than accurately squaring a close approximation $x(1+\delta)$.

In summary, the forward and backward error analysis are different viewpoints describing the same computational process. The former informs us that performing floating-point arithmetic causes loss of accuracy, while the latter suggests that this should not be a concern since we have decide to use floating-point arithmetic in the first place.

[7] This example is taken from `http://people.ds.cam.ac.uk/nmm1/arithmetic/na1.pdf`.

4 Solutions of linear systems of algebraic equations

4.1 Vector and matrix norms

The simplest norm we recall for vectors $x = (x_1, \dots, x_n)^T$ is the **Euclidean norm** defined as

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^n}.$$

This satisfies all the requirements to be a norm:

(i) $\|x\| \geq 0$, and $\|x\| = 0$ if and only if $x = 0$.

(ii) $\|\alpha x\| = |\alpha| \cdot \|x\|$ for all $\alpha \in \mathbb{R}$.

(iii) (Triangle Inequality): $\|x + y\| \leq \|x\| + \|y\|$.

More general norms can be defined: for example the p-norm, for any $p \geq 1$, is defined as

$$\|x\|_p = \begin{cases} \Big(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\Big)^{1/p} & \text{for } 1 \leq p < \infty, \\ \max_{1 \leq i \leq n} |x_i| & \text{for } p = \infty. \end{cases}$$

The norm $\|\cdot\|_1$ is also called the **Manhanttan norm**, while $\|\cdot\|_\infty$ is called the **supremum norm**, while the Euclidean norm is just $\|\cdot\|_2$. One can verify that all $\|\cdot\|_p$ satisfies the three requirements to be a norm. We also have the following inequalities

$$\|x\|_\infty \leq \|x\|_q \leq \|x\|_p \leq \|x\|_1 \leq n\|x\|_\infty \quad \forall 1 \leq p \leq q \leq \infty,$$

which implies that all norms are equivalent.

For matrices, there are also notions of norms. The simplest matrix norm is to view each matrix as a sequence of numbers, and then compute the Euclidean norm of this sequence. The result is the **Frobenius norm**. More precisely, if $A \in \mathbb{R}^{m \times n}$ with (i,j)th entry a_{ij}, then the Frobenius norm of A is defined as

$$\|A\|_F := \Big(\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2\Big)^{1/2}.$$

However, as we will see later in the section for solving nonlinear systems, other types of matrix norms may also be used. For example, there are norms that are **induced** from a vector p-norm. The p-norm of a matrix is defined as

$$\|A\|_p := \max_{\|x\|_p = 1} \|Ax\|_p \quad \text{for } 1 \leq p \leq \infty,\ p \in \mathbb{N}. \tag{4.1}$$

The most common types of p-norm we will encounter are $p = 1$, $p = 2$ and $p = \infty$.

Lemma 4.1.

$$\|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}| = \text{ maximum column sum},$$

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^T A)}, \text{ where } \lambda_{\max} \text{ is the largest eigenvalue of } A^T A,$$

$$\|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}| = \text{ maximum row sum.}$$

Proof. We divide the proof into three parts.

1-norm. Let $a_1, \dots, a_n$ denote n columns of the matrix A, so that

$$A = \begin{pmatrix} | & | & \cdots & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & \cdots & | \end{pmatrix}.$$

Let e_j denote the jth standard basis vector, then $Ae_j = a_j$, and so $\|Ae_j\|_1 = \|a_j\|_1$. For any $x \in \mathbb{R}^n$ with $\|x\|_1 = 1$, we can express x as a linear combination of the standard basis:

$$x = \sum_{i=1}^{n} \alpha_i e_i,$$

and due to $\|x\|_1 = 1$ we must have

$$\sum_{i=1}^{n} |\alpha_i| = 1.$$

Suppose K is the index of the column of the matrix A with maximum 1-norm, i.e.,

$$\|a_K\|_1 \ge \|a_j\|_1 \text{ for any } 1 \le j \le n.$$

Then, together with the identity

$$Ax = \sum_{j=1}^{n} \alpha_j A e_j = \sum_{j=1}^{n} \alpha_j a_j,$$

we obtain

$$\|Ax\|_1 \le \sum_{j=1}^{n} \|\alpha_j a_j\|_1 \le \sum_{j=1}^{n} |\alpha_j|\, \|a_j\|_1 \le \|a_K\|_1 \sum_{j=1}^{n} |\alpha_j| = \|a_K\|_1.$$

Taking maximum over all such $x \in \mathbb{R}^n$ with $\|x\|_1 = 1$ yields

$$\|A\|_1 \le \|a_K\|_1.$$

On the other hand, picking $x = e_K$ gives

$$\|a_K\|_1 = \|Ae_K\|_1 \le \max_{\|x\|_1 = 1} \|Ax\|_1 = \|A\|_1$$

gives the converse inequality. This shows the claim for $\|A\|_1$.

∞-norm. Let $x \in \mathbb{R}^n$ be arbitrary with $\|x\|_\infty = \max_{1\le j\le n} |x_j| = 1$. Then,

$$\|Ax\|_\infty = \max_{1\le i\le m} \left| \sum_{j=1}^n a_{ij} x_j \right| \le \max_{1\le i\le m} \sum_{j=1}^n |a_{ij}|\, |x_j| \le \max_{1\le i\le m} \sum_{j=1}^n |a_{ij}|,$$

and so

$$\|A\|_\infty \le \max_{1\le i\le m} \sum_{j=1}^n |a_{ij}|.$$

For the converse inequality, let a_K denote the row for which the maximum row sum is achieved, i.e.,

$$A = \begin{pmatrix} --- & a_1 & --- \\ --- & a_2 & --- \\ \vdots & \vdots & \vdots \\ --- & a_m & --- \end{pmatrix} \quad \text{and} \quad \sum_{j=1}^n |a_{Kj}| \ge \sum_{j=1}^n |a_{ij}| \ \text{ for } 1 \le i \le m.$$

We consider the vector $y = (x_1, \dots, x_n)^T$ with

$$y_i = \operatorname{sign}(a_{Kj}) \text{ if } a_{Kj} \neq 0, \quad y_i = 1 \text{ if } a_{Kj} = 0.$$

Then, $\|y\|_\infty = 1$ and

$$\max_{1\le i\le m} \sum_{j=1}^n |a_{ij}| = \sum_{j=1}^n |a_{Kj}| = \|Ay\|_\infty \le \max_{\|x\|_\infty = 1} \|Ax\|_\infty = \|A\|_\infty.$$

2-norm. For a matrix $A \in \mathbb{R}^{m\times n}$, the product $B := A^T A \in \mathbb{R}^{n\times n}$ is symmetric and hence is Hermitian (self-adjoint). The eigenvalues of B are real-valued and non-negative, since B is positive semi-definite:

$$x \cdot Bx = x \cdot A^T A x = (Ax) \cdot (Ax) \ge 0.$$

We denote these eigenvalues as

$$0 \le \lambda_1 \le \lambda_2 \le \dots \le \lambda_n.$$

Corresponding to these eigenvalues are n orthonormal (hence linearly independent) eigenvectors $u_1, \dots, u_n$. Let P be a matrix whose ith column in u_i, i.e.,

$$P = \begin{pmatrix} | & | & \cdots & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & \cdots & | \end{pmatrix}.$$

Then, the orthonormality of $\{u_i\}_{i=1}^n$ implies that P is invertible, and we also have

$$P^{-1} B P = D = \operatorname{diag}(\lambda_1, \dots, \lambda_n).$$

Since $\{u_1, \dots, u_n\}$ form a basis of $\mathbb{R}^n$, any vector $x \in \mathbb{R}^n$ can be expressed as a linear combination $x = \sum_{i=1}^n \alpha_i u_i$. Then,

$$A^T A x = Bx = \sum_{i=1}^n \alpha_i B u_i = \sum_{i=1}^n \alpha_i \lambda_i u_i,$$

and so, due to the orthonormality of $\{u_i\}_{i=1}^n$, we have

$$\|Ax\|_2^2 = x \cdot A^T A x = x \cdot Bx = \sum_{i=1}^n |\alpha_i|^2 \lambda_i \le \lambda_n \sum_{i=1}^n |\alpha_i|^2 = \lambda_n \|x\|_2^2.$$

Taking maximum over all $x \in \mathbb{R}^n$ with $\|x\|_2 = 1$ yields

$$\|A\|_2 \le \sqrt{\lambda_n} = \sqrt{\text{largest eigenvalue of } A^T A}.$$

For the converse inequality, we consider $x = u_n$ and compute that

$$\|Au_n\|_2^2 = u_n \cdot A^T A u_n = u_n \cdot B u_n = |\lambda_n|^2.$$

Hence,

$$\sqrt{\lambda_n} = \|Au_n\|_2 \le \max_{\|x\|_2 = 1} \|Ax\|_2 = \|A\|_2.$$

□

Definition 4.1. *The* **spectrum** *of a square matrix $A \in \mathbb{R}^{n\times n}$ is the set of all eigenvalues of A. The* **spectral radius** *of A is defined as*

$$\rho(A) := \max_{1\le i\le n} |\lambda_i|. \tag{4.2}$$

With this definition, sometimes $\|A\|_2$ is called the **spectral norm**.

Exercise. Show that

$$\|A\|_2 \le \|A\|_F \le \sqrt{n}\|A\|_2.$$

Lemma 4.2. *The matrix p-norm $\|A\|_p$ defined in* (4.1) *satisfies all three requirements of being a norm.*

Proof. Let x be an arbitrary vector in $\mathbb{R}^n$. Now, it is clear that Ax is a vector in $\mathbb{R}^m$. By virtue of the vector p-norm being a norm, we have

(i) $\|Ax\|_p \ge 0$ and $\|Ax\|_p = 0$ if and only if $Ax = 0$.

(ii) $\|\alpha Ax\|_p = |\alpha| \cdot \|Ax\|_p$, for all $\alpha \in \mathbb{R}$.

(iii) $\|Ax + Bx\|_p \le \|Ax\|_p + \|Bx\|_p$.

Then, we see that

$$\begin{aligned}
\|A\|_p &= \max_{\|x\|_p=1} \|Ax\|_p \geq 0, \\
\|\alpha A\|_p &= \max_{\|x\|_p=1} \|\alpha Ax\|_p = \max_{\|x\|_p=1} |\alpha| \cdot \|Ax\|_p = |\alpha| \cdot \|A\|_p.
\end{aligned}$$

Meanwhile, $Ax = 0$ for arbitrary x if and only if $A = 0$, and so

$$\|A\|_p = 0 \text{ if and only if } A = 0.$$

Lastly,

$$\|A + B\|_p = \max_{\|x\|_p=1} \|(A+B)x\|_p \leq \max_{\|x\|_p=1} (\|Ax\|_p + \|Bx\|_p) = \|A\|_p + \|B\|_p.$$

□

Lemma 4.3. *For any $A \in \mathbb{R}^{n\times m}$ and $B \in \mathbb{R}^{m\times q}$, it holds that*

$$\|AB\|_p \leq \|A\|_p \|B\|_p \quad \forall 1 \leq p \leq \infty,\ p \in \mathbb{N}.$$

Proof. The key is to first show that

$$\|Ax\|_p \leq \|A\|_p \|x\|_p.$$

This comes from the fact that if $x \in \mathbb{R}^n$ is a vector whose vector p-norm isn't 1, we can define $y = \frac{x}{\|x\|_p}$. Then $\|y\|_p = 1$ and we have

$$\|Ay\|_p = \frac{1}{\|x\|_p}\|Ax\|_p.$$

Taking the maximum over y is the same as taking the maximum over x, which yields

$$\|A\|_p = \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p} \quad \Rightarrow \quad \|Ax\|_p \leq \|A\|_p \|x\|_p.$$

Now, set $y = Bx$ and using the above property gives

$$\|ABx\|_p = \|Ay\|_p \leq \|A\|_p \|y\|_p = \|A\|_p \|Bx\|_p \leq \|A\|_p \|B\|_p \|x\|_p.$$

Taking over the maximum of all x such that $\|x\|_p = 1$ yields the result. □

Exercise. Show that $\|AB\|_F \leq \|A\|_F \|B\|_F$ for the Frobenius norm.

4.2 Relative errors

Let $\tilde{x}$ be an approximation to x, then recall the relative error is defined as

$$\frac{|x-\tilde{x}|}{|x|}.$$

One can similarly define the relative error of an approximate vector $\tilde{x}$ to x as

$$\frac{\|x-\tilde{x}\|}{\|x\|},$$

where $\|\cdot\|$ can be any of the vector p-norm.

Lemma 4.4. *If the relative error of $\tilde{x}$ to x satisfies*

$$\frac{\|x-\tilde{x}\|}{\|x\|} \le \theta < 1,$$

then, the relative error of x to $\tilde{x}$ satisfies

$$\frac{\|x-\tilde{x}\|}{\|\tilde{x}\|} \le \frac{\theta}{1-\theta}.$$

Proof. We easily see

$$\frac{\|x-\tilde{x}\|}{\|\tilde{x}\|} = \frac{\|x-\tilde{x}\|}{\|x\|}\frac{\|\tilde{x}\|}{\|x\|} \le \theta\frac{\|\tilde{x}\|}{\|x\|}.$$

Using $\|x-\tilde{x}\| \le \theta\|x\|$, we derive by the (other) triangle inequality

$$\|x\| - \|\tilde{x}\| \le |\ \|x\| - \|\tilde{x}\|\ | \le \|x-\tilde{x}\| \le \theta\|x\|,$$

and so

$$(1-\theta)\|x\| \le \|\tilde{x}\| \ \Rightarrow\ \frac{\|x\|}{\|\tilde{x}\|} \le \frac{1}{1-\theta}.$$

This then gives

$$\frac{\|x-\tilde{x}\|}{\|\tilde{x}\|} \le \theta\frac{\|x\|}{\|\tilde{x}\|} \le \frac{\theta}{1-\theta}.$$

□

Let us note that $\theta/(1-\theta)$ is not too different from θ if θ is small. This shows that if the relative error of $\tilde{x}$ to x is small, so is the relative error of x to $\tilde{x}$, and we may use **any one of them** to measure the relative error. In practice, it is easier to compute and estimate

$$\frac{\|x-\tilde{x}\|}{\|\tilde{x}\|}$$

than the other, since it is usually complicated and difficult to have an accurate estimate on the exact value of x. In fact our entire task is to find an accurate estimation of the exact solution x.

4.3 Sensitivity of linear systems

Before we give methods to solve linear systems of algebraic equations, let us first discuss the issues of sensitivity. Consider solving the linear system

$$Ax = b.$$

But due to rounding errors, or observation data errors, the actual problem solved by the computer is the perturbed system:

$$\tilde{A}\tilde{x} = b.$$

The question is will $\tilde{x}$ be a good approximation to x? It is convenient to introduce the perturbation equation

$$\tilde{A} = A + E \text{ where } E = \tilde{A} - A.$$

If A is non-singular, and E is not too large in the sense that

$$\|A^{-1}E\| < 1,$$

then $\tilde{A} = A + E$ is also non-singular. Indeed, it suffices to show that for any $x \neq 0$, we have $(A + E)x \neq 0$. Since A is non-singular, we can express

$$\tilde{A}x = A(x + A^{-1}Ex).$$

Thus, $(A+E)x \neq 0$ if and only if $x+A^{-1}Ex \neq 0$. Due to the hypothesis $\|A^{-1}E\| < 1$, we obtain

$$\|x + A^{-1}Ex\| \geq \|x\| - \|A^{-1}Ex\| \geq \|x\| - \|A^{-1}E\|\|x\| > 0,$$

which shows that $x + A^{-1}Ex \neq 0$.

We now describe a fundamental perturbation theory.

Theorem 4.1. *Let A be a non-singular matrix, and $\tilde{A} = A + E$. If*

$$Ax = b, \quad \tilde{A}\tilde{x} = b$$

with $b \neq 0$, then we have

$$\frac{\|\tilde{x} - x\|}{\|\tilde{x}\|} \leq \|A^{-1}E\| = \|A^{-1}\tilde{A} - I\|.$$

So if,

$$\|A^{-1}E\| < 1,$$

then $\tilde{A}$ is non-singular with

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \frac{\|A^{-1}E\|}{1 - \|A^{-1}E\|}.$$

Proof. To prove the first result, we use $\tilde{A} = A + E$ to see that

$$b = \tilde{A}\tilde{x} = A\tilde{x} + E\tilde{x}.$$

Since $b = Ax$, we have

$$Ax = A\tilde{x} + E\tilde{x} \quad \Rightarrow \quad x - \tilde{x} = A^{-1}E\tilde{x}.$$

This implies

$$\|x - \tilde{x}\| \leq \|A^{-1}E\|\|\tilde{x}\|,$$

which is the first result. For the second result, using the previous lemma 4.4 with $\theta = \|A^{-1}E\| < 1$, we know that

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \frac{\theta}{1-\theta}.$$

The fact that $\tilde{A}$ is non-singular comes from the argument immediately preceding the theorem. □

4.4 Condition number of a matrix

The relative error of the solution to the perturbed system $\tilde{A}\tilde{x} = b$ is

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leq \frac{\|A^{-1}E\|}{1 - \|A^{-1}E\|},$$

but the right-hand side seems difficult to interpret. In the following we aim to derive a more convenient relation. Using Lemma 4.3, we see that

$$\|A^{-1}E\| \leq \|A^{-1}\| \ \|E\| = \|A^{-1}\| \ \|A\| \frac{\|E\|}{\|A\|}.$$

Defining

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

we have

$$\|A^{-1}E\| \leq \kappa(A)\frac{\|E\|}{\|A\|} \quad \Rightarrow \quad \frac{\|\tilde{x} - x\|}{\|x\|} \leq \frac{\kappa(A)\frac{\|E\|}{\|A\|}}{1 - \kappa(A)\frac{\|E\|}{\|A\|}}.$$

If $\kappa(A)\frac{\|E\|}{\|A\|}$ is small, then the fraction

$$c(E) := \frac{1}{1 - \kappa(A)\frac{\|E\|}{\|A\|}}$$

is close to 1, and thus

$$\frac{\|\tilde{x}-x\|}{\|x\|} \le c(E)\kappa(A)\frac{\|E\|}{\|A\|} = c(E)\kappa(A)\frac{\|\tilde{A}-A\|}{\|A\|}. \tag{4.3}$$

This gives us that the relative error of $\tilde{x}$ to the exact solution x can be controlled by a factor of the relative error of the perturbed matrix $\tilde{A}$ to the true matrix A.

Definition 4.2 (Condition number)**.** *The real number*

$$\kappa(A) := \|A\|\ \|A^{-1}\|$$

is called the **condition number** *of the matrix A.*

Let $\|\cdot\|$ be an induced matrix norm. Due to the chain of inequalities and equalities

$$1 \le \|I\| = \|AA^{-1}\| \le \|A\|\ \|A^{-1}\| = \kappa(A),$$

we find that

$$\boxed{\text{the condition number } \kappa(A) \text{ is } \textbf{always} \text{ greater than or equal to one.}}$$

This means that the condition number is always a magnification constant, and the bound on the error is never diminished in passing from the matrix to the solution.

The condition number is an important quantity in numerical analysis and has direct influence on the accuracy of the solution to the linear system $Ax = b$. Suppose the matrix A is rounded to the matrix $\tilde{A}$ on a computer with rounding unit ε_M (machine accuracy), so that we have

$$\tilde{a}_{ij} = a_{ij} + a_{ij}\varepsilon_{ij}, \quad |\varepsilon_{ij}| \le \varepsilon_M.$$

Then, in any of the matrix p-norms we have

$$\|\tilde{A}-A\| \le \varepsilon_M\|A\|.$$

Suppose we solve the perturbed system $\tilde{A}\tilde{x} = b$ without making any additional errors, we should obtain a solution $\tilde{x}$ satisfying

$$\frac{\|\tilde{x}-x\|}{\|x\|} \le c(E)\kappa(A)\frac{\|\tilde{A}-A\|}{\|A\|} \le \varepsilon_M\kappa(A)c(E).$$

If for instance, $\varepsilon_M = 10^{-t}$ and $\kappa(A) = 10^k$, then the solution $\tilde{x}$ can have relative error as large as 10^{k-t}. This roughly justifies the following rule:

$$\boxed{\text{If } \kappa(A) = 10^k\text{, one should expect to lose at least } k \text{ digits of accuracy in solving the system } Ax = b.}$$

4.5 Simple solution of a linear system

Consider the $n \times n$ square system $Ax = b$. A natural way to solve the system seems to be multiplying both sides of $Ax = b$ by A^{-1} to get the solution

$$x = A^{-1}b.$$

This suggests the following algorithm for solving $Ax = b$:

1. Compute $C = A^{-1}$.
2. Compute $x = Cb$.

Unfortunately, unless A has a very simple structure this algorithm is very **expensive** when the size of A is large, i.e., n is a large number, as computing A^{-1} is extremely time-consuming, and more importantly unstable. It turns out that there are more efficient algorithms for solving $Ax = b$, such as the **LU factorization** and the **Cholesky factorization**, where the basic idea is to factorize A into a product of matrices PQ such that

$$Ax = b \quad \Leftrightarrow \quad Py = b, \quad Qx = y,$$

where it is faster and easier to invert P and Q.

4.6 Solutions of triangular systems

4.6.1 Forward-substitution algorithm

Consider solving the lower triangular system

$$Lx = b$$

where L is a lower triangular matrix of order n, i.e., $l_{ij} = 0$ for all $i < j$ and

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{pmatrix}$$

Writing $Lx = b$ component-wise leads to

$$\begin{array}{lcl} l_{11}x_1 & = & b_1 \\ l_{21}x_1 + l_{22}x_2 & = & b_2 \\ & \vdots & \\ l_{i1}x_1 + l_{i2}x_2 + \cdots + l_{ii}x_i & = & b_i \\ & \vdots & \\ l_{n1}x_1 + l_{n2}x_2 + \cdots + l_{ni}x_i + \cdots + l_{nn}x_n & = & b_n \end{array}$$

then we can solve the system as follows:

$$\begin{aligned}
x_1 &= b_1/l_{11} \\
x_2 &= (b_2 - l_{21}x_1)/l_{22} \\
&\vdots \\
x_i &= (b_i - \sum_{j=1}^{i-1} l_{ij}x_j)/l_{ii}, \quad i = 3, 4, \ldots, n.
\end{aligned}$$

This algorithm is called the **forward-substitution algorithm** as we start with x_1 and end with x_n.

4.6.2 Backward-substitution algorithm

Using a similar idea, we can solve the following upper triangular system

$$Ux = b$$

where U is upper triangular of order n, i.e., $u_{ij} = 0$ for all $i > j$ and

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{pmatrix}.$$

Writing the system $Ux = b$ component-wise leads to

$$\begin{aligned}
u_{11}x_1 + u_{12}x_2 + \cdots + u_{1i}x_i + \cdots + u_{1n}x_n &= b_1 \\
u_{22}x_2 + \cdots + u_{2i}x_i + \cdots + u_{2n}x_n &= b_2 \\
&\vdots \\
u_{ii}x_i + \cdots + u_{in}x_n &= b_i \\
&\vdots \\
u_{nn}x_n &= b_n.
\end{aligned}$$

Then we can solve the system as follows:

$$\begin{aligned}
x_n &= b_n/u_{nn} \\
x_{n-1} &= (b_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1} \\
&\vdots \\
x_i &= (b_i - \sum_{j=i+1}^{n} u_{ij}x_j)/u_{ii}, \quad i = n-2, n-3, \ldots, 2, 1.
\end{aligned}$$

This algorithm is called the **backward-substitution algorithm** as we start with x_n and end with x_1.

4.6.3 Computational complexity

A good indication on whether a particular numerical method is expensive is the computational complexity. All numerical algorithms can be decomposed into the basic components of vector-vector, matrix-vector and matrix-matrix operations, which all involve the basic operations (floating-point operations aka "flop") of addition, subtraction, multiplication and division of two numbers (floating points). Note that this is only a rough estimate of the complexity of the algorithms, and is not an accurate predictor of the computational time on modern computers.

vector-vector operations. for $x, y \in \mathbb{R}^n$:

- inner product $x \cdot y$: $2n-1$ flops (or $2n$ if n is large) - n multiplications to get $\{x_i y_i\}_{i=1}^n$ and $n-1$ addition of two floating points.
- sum $x+y$, scalar multiplication αx: n flops.

matrix-vector product. $y = Ax$ with $A \in \mathbb{R}^{m\times n}$:

- $m(2n-1)$ flops (or $2mn$ if n is large) - think of computing m inner products between the rows of A and the vector x.

matrix-matrix product $C = AB$ with $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$:

- $mp(2n-1)$ flops (or $2mnp$ if n large).

The number of flops can decrease if matrices have a special structure or is sparse (i.e., a large amount of entries in the matrix are zero). Let us consider the computational costs of solving linear systems $Ax = b$ for the following cases of the matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$.

1. Diagonal matrices ($a_{ij} = 0$ if $i \neq j$): n flops as
$$x = (b_1/a_{11}, \cdots, b_n/a_{nn})^T.$$
2. Lower triangular matrices ($a_{ij} = 0$ if $i < j$), i.e., the forward-substitution algorithm: When computing x_i, we need to perform $(i-1)$ multiplications, $(i-1)$ pairwise subtraction and one last multiplication, leading to $2i-1$ flops. Hence, the total complexity is $\sum_{i=1}^n 2i - 1 = n^2$ flops.
3. Upper triangular matrices ($a_{ij} = 0$ if $i > j$), i.e., the backward-substitution algorithm: n^2 flops.

4.7 Cholesky factorization

Definition 4.3. *A matrix $A \in \mathbb{R}^{n\times n}$ is said to be* **symmetric and positive definite** *(SPD) if it satisfies*

$$A^T = A \text{ and } x^T Ax > 0 \quad \forall x \neq 0.$$

Lemma 4.5. *Let $A \in \mathbb{R}^{n\times n}$ be a SPD matrix. Then, A is invertible, and any eigenvalue of A is positive. Furthermore, if $U \in \mathbb{R}^{m\times n}$ with $m \neq n$ has linearly independent columns, then $U^TU \in \mathbb{R}^{n\times n}$ is SPD.*

Proof. Suppose for a contradiction there is some $x \neq 0$ such that $Ax = 0$, then

$$x^T Ax = 0,$$

which contradicts positive definiteness. Hence A is invertible.

Suppose λ is an eigenvalue, then there exists a corresponding eigenvector $x \neq 0$ such that

$$Ax = \lambda x.$$

Then we have

$$0 < x^T Ax = \lambda x^T x = \lambda \|x\|^2.$$

Since $x \neq 0$, we have $\|x\|^2 > 0$ and so $\lambda > 0$.

Let $U \in \mathbb{R}^{m\times n}$ be a matrix with linearly independent columns. Then, the only solution to $Ux = 0$ is $x = 0$. However,

$$x^T U^T Ux = \|Ux\|^2,$$

and so for a non-zero vector $x \in \mathbb{R}^n$, $x^TU^TUx > 0$, which implies positive definiteness. For symmetry it is clear that $(U^TU)^T = U^TU$. □

The following procedure, known as the **Cholesky factorization**, splits a SPD matrix A into U^TU where U is an upper triangular matrix. Let us write for constants α and u_{11}, vectors a and r, and submatrices A_{11} and U_{11}:

$$A = \begin{pmatrix} \alpha & a^T \\ a & A_{11} \end{pmatrix}, \quad U = \begin{pmatrix} u_{11} & r^T \\ 0 & U_{11} \end{pmatrix},$$

then

$$A = U^TU \quad \Leftrightarrow \quad \begin{pmatrix} \alpha & a^T \\ a & A_{11} \end{pmatrix} = \begin{pmatrix} u_{11} & 0 \\ r & U_{11}^T \end{pmatrix} \begin{pmatrix} u_{11} & r^T \\ 0 & U_{11} \end{pmatrix}.$$

Comparing with both sides of the equation, we obtain

1. $\alpha = u_{11}^2$

2. $a^T = u_{11} r^T$

3. $A_{11} = r \otimes r + U_{11}^T U_{11}$

or equivalently, we can write

1. $u_{11} = \sqrt{\alpha}$ (take only the positive square root).

2. $r^T = a^T / u_{11}$.

3. $U_{11}^T U_{11} = A_{11} - r \otimes r =: \hat{A}_{11}$.

To continue, we observe that step 3 is the Cholesky factorization of $\hat{A}_{11}$ if $\hat{A}_{11}$ is SPD. We check this now. By definition, $\hat{A}_{11} = A_{11} - r \otimes r$ is symmetric, and so it suffices to show that $\hat{A}_{11}$ is positive definite. Let $x^T = (x_1, \tilde{x}^T)$ for $x_1 \in \mathbb{R}$ and $\tilde{x} \in \mathbb{R}^{n-1}$, then

$$\begin{aligned} x^T \begin{pmatrix} \alpha & a^T \\ a & A_{11} \end{pmatrix} x &= \alpha x_1^2 + 2x_1(a^T\tilde{x}) + \tilde{x}^T A_{11} \tilde{x} \\ &= \alpha x_1^2 + 2x_1(a^T\tilde{x}) + \frac{1}{\alpha}(a^T\tilde{x})^2 + \tilde{x}^T \hat{A}_{11} \tilde{x} \\ &= \alpha(x_1 + (a^T\tilde{x})/\alpha)^2 + \tilde{x}^T \hat{A}_{11} \tilde{x}. \end{aligned}$$

For any non-zero $\tilde{x}$, choose $x_1 = -(a^T\tilde{x})/\alpha$, so that

$$\tilde{x}^T \hat{A}_{11} \tilde{x} = x^T \begin{pmatrix} \alpha & a^T \\ a & A_{11} \end{pmatrix} x > 0.$$

This yields the positive definiteness of $\hat{A}_{11}$. Then, we can repeat the above procedure for the submatrix $\hat{A}_{11}$. In fact the Cholesky factorization proceeds in n steps (for a $n \times n$ matrix A), with each step filtering out one row of the matrix A:

(1) At the first step, the first row of U is computed and the $(n-1) \times (n-1)$ submatrix A_{11} in the right bottom corner is modified;

(2) At the 2nd step, the second row of U is computed and the $(n-2) \times (n-2)$ submatrix in the right bottom corner is modified.

(n) The procedure continues until the n-th step, or until nothing is left in the right bottom corner.

Theorem 4.2. *A matrix A is SPD if and only if it has a Cholesky factorization (U^TU or LL^T).*

Proof. If A is SPD, then the above procedure gives a Cholesky factorization. Conversely, if $A = U^TU$, where U is a nonsingular upper triangular matrix, let $y = Ux$, then

$$x^T A x = x^T U^T U x = (Ux)^T (Ux) = y^T y \geq 0$$

with equality if and only if $y = 0$, i.e, when $x = 0$. Hence, A is positive definite. It is easy to see that $A^T = (U^TU)^T = U^TU = A$ and so A is symmetric. $\square$

Theorem 4.3. *Any two Cholesky factorization of the same matrix A differ by the sign of their columns.*

Proof. Let $A = U_1^T U_1 = U_2^T U_2$ be two Cholesky factorization with upper triangular matrices U_1 and U_2. Then, it holds that

$$I = U_1^{-T} U_2^T U_2 U_1^{-1} = (U_2 U_1^{-1})^T (U_2 U_1^{-1}).$$

Hence,

$$U_2 U_1^{-1} = (U_2 U_1^{-1})^{-T}.$$

Since U_1 is an upper triangular matrix, its inverse U_1^{-1} is also an upper triangular matrix, and thus the product $U_2 U_1^{-1}$ is also upper triangular. However, the right-hand side $(U_2 U_1^{-1})^{-T}$ is lower triangular. In order for the above equation to hold, both sides must be simultaneously upper and lower triangular, i.e., both sides are diagonal matrices. Hence, $U_2 U_1^{-1} =: D$ is a diagonal matrix and we have

$$I = (U_2 U_1^{-1})^T (U_2 U_1^{-1}) = D^T D = D^2,$$

and so D must be a diagonal matrix with entries ± 1 on its main diagonal. Then, using that $U_2 = D U_1$ we see that the two Cholesky factors U_1 and U_2 differ by signs of the columns. $\square$

4.7.1 Computational cost of the Cholesky factorization

Let us summarize the Cholesky factorization in code: we use the notation U_{k, j:n} to denote all the entries $\{U_{k,i}\}_{i=j}^n$. The Cholesky factorization of a SPD matrix $A \in \mathbb{R}^{n\times n}$ into $U^T U$ for upper triangular U can be expressed as follows

```
1  Set U = A:
2         for k = 1 to n:
3              U_{k,k:n} = U_{k,k:n}/sqrt(U_{k,k});
4              for j = k+1 to n:
5                   U_{j,j:n} = U_{j,j:n} - U_{k,j:n} U_{k,j} / U_{k,k};
6              end
7          end
```

Let us look at the innermost loop on line 5, for a fixed $j \in \{k+1, \dots, n\}$ we perform 1 division, $n - j + 1$ multiplications and $n - j + 1$ subtractions. Since j runs from $k+1$ to n, the cost for evaluating the loop from line 4 to line 6 is

- $n - k$ divisions,
- $((n+1)(n-k) - \sum_{j=k+1}^n j) = \frac{1}{2}(n-k)(n-k+1)$ multiplications,
- $((n+1)(n-k) - \sum_{j=k+1}^n j) = \frac{1}{2}(n-k)(n-k+1)$ subtractions,

where we used

$$\sum_{j=k+1}^{n} n-j+1 = (n+1)(n-k) - \sum_{j=k+1}^{n} j - (n+1)(n-k) - \sum_{j=1}^{n} j + \sum_{j=1}^{k} j$$
$$= \frac{1}{2}(n^2+n-2kn-k+k^2) = \frac{1}{2}(n-k)(n-k+1).$$

A short calculation with the basic facts

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1), \quad \sum_{k=1}^{n} k^2 = \frac{1}{6}n(n+1)(2n+1)$$

shows that

$$\sum_{k=1}^{n}(n-k)(n-k+1) = \sum_{k=1}^{n}(n^2+n-2kn-k+k^2) = \frac{1}{3}(n^3-n).$$

Then, looping over $k = 1, \ldots, n$ implies the number of flops we perform for subprogram coded in line 2, 4, 5, 6, 7 is

$$\sum_{k=1}^{n}[(n-k)+(n-k)(n-k+1)] = \frac{n^2}{2} - \frac{n}{2} + \frac{1}{3}(n^3-n). \tag{4.4}$$

Meanwhile, for fixed $k \in \{1, \ldots, n\}$, in line 2 we perform

- $n-k+1$ divisions (discounting the computation of the square root)

and so for the subprogram coded in line 2, 3, 7, the number of flops we perform is

$$\sum_{k=1}^{n} n-k+1 = \frac{1}{2}(n^2+n). \tag{4.5}$$

Adding (4.4) and (4.5) shows that the total computational complexity of the Cholesky factorization is

$$n^2 + \frac{1}{3}(n^3-n) \approx \frac{n^3}{3} \text{ for large } n.$$

Example 4.1. *Consider a* 2×2 *SPD matrix*

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}.$$

Set $u_{ij} = a_{ij}$*, then for* $k = 1$ *and for* j *from 2 to 2, we perform*

$$u_{11} \mapsto u_{11}/\sqrt{u_{11}},$$
$$u_{12} \mapsto u_{12}/\sqrt{u_{11}},$$
$$u_{22} \mapsto u_{22} - \frac{u_{12}u_{12}}{u_{11}}.$$

This yields

$$U = \begin{pmatrix} \sqrt{a_{11}} & \frac{a_{12}}{\sqrt{a_{11}}} \\ \frac{a_{12}}{\sqrt{a_{11}}} & a_{22} - \frac{a_{12}^2}{a_{11}} \end{pmatrix}$$

Then for $k = 2$ we do not have an inner loop for j, and so we perform

$$u_{22} \mapsto \frac{u_{22}}{\sqrt{u_{22}}},$$

which leads to

$$U = \begin{pmatrix} \sqrt{a_{11}} & \frac{a_{12}}{\sqrt{a_{11}}} \\ \frac{a_{12}}{\sqrt{a_{11}}} & \left(a_{22} - \frac{a_{12}^2}{a_{11}}\right)^{1/2}. \end{pmatrix}$$

For $k = 1$ we performed 1 multiplication each for updating u_{11} and u_{12}, and 1 multiplication, 1 division and 1 subtraction to update u_{22}. Then for $k = 2$, we perform 1 division for updating u_{22}. In total this is 6 flops to compute the Cholesky factorization of a SPD matrix $A \in \mathbb{R}^{2\times 2}$.

4.7.2 Examples

Example 4.2. *Find the Cholesky factorization of the following matrix*

$$A = \begin{pmatrix} 3 & -1 & 1 \\ -1 & 3 & 0 \\ 1 & 0 & 3 \end{pmatrix}. \tag{4.6}$$

In the following, the symbol $*$ means we do not care about the value at the corresponding entry.

1. *We have $u_{11} = \sqrt{3}$, $r^T = \frac{1}{\sqrt{3}}(-1, 1)$. Update the 1st row and the submatrix at the right bottom corner:*

$$\begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ * & 3 - \frac{1}{3} & 0 + \frac{1}{3} \\ * & \frac{1}{3} & 3 - \frac{1}{3} \end{pmatrix} = \begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ * & \frac{8}{3} & \frac{1}{3} \\ * & \frac{1}{3} & \frac{8}{3} \end{pmatrix}$$

2. *The matrix $\hat{A}_{11}$ is given as*

$$\hat{A}_{11} = \begin{pmatrix} \frac{8}{3} & \frac{1}{3} \\ * & \frac{8}{3} \end{pmatrix}.$$

Then, $u_{22} = \sqrt{\frac{8}{3}}$, $r = \frac{1}{3}/\sqrt{\frac{8}{3}} = \frac{1}{\sqrt{24}} = \frac{1}{2\sqrt{6}}$. Updating the 2nd row and the submatrix at the right bottom corner leads to

$$\hat{A}_{11} \to \begin{pmatrix} \sqrt{\frac{8}{3}} & \frac{1}{2\sqrt{6}} \\ * & \frac{63}{24} \end{pmatrix},$$

leading to

$$\begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ * & \frac{8}{3} & \frac{1}{3} \\ * & \frac{1}{3} & \frac{8}{3} \end{pmatrix} \rightarrow \begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ * & \frac{2\sqrt{2}}{\sqrt{3}} & \frac{1}{2\sqrt{6}} \\ * & * & \frac{63}{24} \end{pmatrix}$$

3. *Update the 3rd row and the submatrix at the right bottom corner:*

$$\begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ * & \frac{2\sqrt{2}}{\sqrt{3}} & \frac{1}{2\sqrt{6}} \\ * & * & \frac{63}{24} \end{pmatrix} \rightarrow \begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ * & \frac{2\sqrt{2}}{\sqrt{3}} & \frac{1}{2\sqrt{6}} \\ * & * & \sqrt{\frac{63}{24}} \end{pmatrix}.$$

This gives

$$U = \begin{pmatrix} \sqrt{3} & -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ 0 & \frac{2\sqrt{2}}{\sqrt{3}} & \frac{1}{2\sqrt{6}} \\ 0 & 0 & \sqrt{\frac{63}{24}} \end{pmatrix}. \tag{4.7}$$

Example 4.3. *Find the Cholesky factorization of the following matrix*

$$\begin{pmatrix} 4 & \frac{1}{2} & 1 \\ \frac{1}{2} & \frac{17}{16} & \frac{1}{4} \\ 1 & \frac{1}{4} & \frac{33}{64} \end{pmatrix}.$$

1. *Set* $u_{11} = 2$, $r^T = (\frac{1}{4}, \frac{1}{2})$ *and compute for*

$$\hat{A}_{11} = \begin{pmatrix} 1 & \frac{1}{8} \\ * & \frac{17}{64} \end{pmatrix}.$$

Update the 1st row and the submatrix at the right bottom corner:

$$\begin{pmatrix} 4 & \frac{1}{2} & 1 \\ \frac{1}{2} & \frac{17}{16} & \frac{1}{4} \\ 1 & \frac{1}{4} & \frac{33}{64} \end{pmatrix} \rightarrow \begin{pmatrix} 2 & \frac{1}{4} & \frac{1}{2} \\ * & 1 & \frac{1}{8} \\ * & \frac{1}{8} & \frac{17}{64} \end{pmatrix}$$

2. *Update the 2nd row and the submatrix at the right bottom corner:*

$$\begin{pmatrix} 1 & \frac{1}{8} \\ \frac{1}{8} & \frac{17}{64} \end{pmatrix} \rightarrow \begin{pmatrix} 1 & \frac{1}{8} \\ * & \frac{1}{4} \end{pmatrix}$$

3. *Update the third row and the submatrix at the right bottom corner:*

$$\left(\tfrac{1}{4}\right) \rightarrow \left(\tfrac{1}{2}\right)$$

this gives

$$U = \begin{pmatrix} 2 & \frac{1}{4} & \frac{1}{2} \\ 0 & 1 & \frac{1}{8} \\ 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

4.8 LU factorization and Gaussian elimination

For SPD matrices we can appeal to the Cholesky factorization, but they occur rarely in applications of mathematics. The next method we discuss is the LU factorization, and as the name suggests we factorize A into a product of a lower triangular matrix L and an upper triangular matrix U. This process can be derived from the more familiar **Gaussian elimination** that transforms a matrix A into an upper triangular matrix U.

A key concept in Gaussian elimination is the notion of elementary row/column operators. These are given as

(R1) Row switching: row r_i can be switched with row r_j.

(R2) Row multiplication: row r_i can be multiplied by a nonzero scalar $\alpha \neq 0$.

(R3) Row addition: row r_i can be added with a scalar multiple of row r_j.

The elementary column operations are analogously defined. It turns out that these row/column operations are associated to so-called elementary matrices. For example,

$$T_{i,j} = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 0 & & 1 & & \\ & & & \ddots & & & \\ & & 1 & & 0 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

is the matrix representing row switching $r_i \leftrightarrow r_j$, where the (i,j)th and (j,i)th entry is 1. In particular, the matrix product $T_{i,j}A$ is produced from A by switching rows i and j. Meanwhile, the matrix

$$D_i(\alpha) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \alpha & & & \\ & & & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

represents row multiplication with scalar $\alpha \neq 0$, and

$$L_{i,j}(\alpha) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \ddots & & & \\ & & \alpha & & 1 & & \\ & & & & & \ddots & \\ & & & & & & 1 \end{pmatrix}$$

represents adding α times row i to row j. The α value occurs at the (j, i)th entry. Notice $L_{i,j}(\alpha)$ is **always** lower triangular if $i < j$.

Exercise. Show that $T_{i,j}^{-1} = T_{i,j}$, $D_i(\alpha)^{-1} = D_i(1/\alpha)$ and $L_{i,j}(\alpha)^{-1} = L_{i,j}(-\alpha)$.

Example 4.4. *Consider the following* 2×2 *system:*

$$Ax \equiv \begin{pmatrix} 2 & 4 \\ 4 & 11 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \equiv b. \tag{4.8}$$

Eliminating x_1 *in the 2nd equation requires us to add row 1 multiplied by -2 to row 2, leading to*

$$L_{1,2}(-2)A \equiv \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 \\ 4 & 11 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 0 & 3 \end{pmatrix} \equiv U.$$

Then, the inverse $L_{1,2}(-2)^{-1}$ *is given as* $L_{1,2}(2) =: L$*, which is lower triangular, and so*

$$A = \begin{pmatrix} 2 & 4 \\ 4 & 11 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 \\ 0 & 3 \end{pmatrix} \equiv LU. \tag{4.9}$$

This gives a LU *factorization of* A*. Then, solving the system*

$$Ax = b$$

is equivalent to solving the system

$$LUx = b,$$

which can be done as follows:

$$Lc = b, \quad Ux = c.$$

Applying this process to equation (4.8)*, we have*

$$Lc = b \quad \Longleftrightarrow \quad \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix},$$

which gives

$$\begin{pmatrix} c_1 \\ c_2 \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \end{pmatrix}.$$

Then

$$Ux = c \quad \Longleftrightarrow \quad \begin{pmatrix} 2 & 4 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \end{pmatrix},$$

which gives the solution of equation (4.8)*:*

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ -1 \end{pmatrix}.$$

Example 4.5. *Consider the following* 3×3 *system:*

$$Ax \equiv \begin{pmatrix} 1 & 1 & 1 \\ 3 & 6 & 4 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ -1/3 \end{pmatrix} \equiv b. \tag{4.10}$$

Eliminating x_1 *in the 2nd equation requires us to add row 1 multiplied by -3 to row 2, i.e., multiply with* $L_{1,2}(-3)$*, and eliminating* x_1 *in the 3rd equation requires us to add row 1 multiplied by -1 to row 3, i.e., multiply with* $L_{1,3}(-1)$*, leading to*

$$L_{1,3}(-1)L_{1,2}(-3)A \equiv \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 3 & 6 & 4 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 1 & 0 \end{pmatrix} \equiv U_1.$$

Now eliminating x_2 *in the 3rd equation requires adding row 2 multiplied by* $-1/3$ *to row 3, i.e., multiplying with* $L_{2,3}(-1/3)$,

$$\begin{aligned} &L_{2,3}(-1/3)L_{1,3}(-1)L_{1,2}(-3)A \\ &\equiv \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1/3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 3 & 6 & 4 \\ 1 & 2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & -1/3 \end{pmatrix} \equiv U. \end{aligned}$$

Hence,

$$\begin{aligned} A = \begin{pmatrix} 1 & 1 & 1 \\ 3 & 6 & 4 \\ 1 & 2 & 1 \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1/3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & -1/3 \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1/3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & -1/3 \end{pmatrix} \equiv LU. \end{aligned}$$

This completes a LU factorization of A*. Then,*

$$Ax = b \quad \Longleftrightarrow \quad LUx = b,$$

which can be done as follows:

$$Lc = b, \quad Ux = c.$$

Applying this process to equation (4.10)*, we have*

$$Lc = b \quad \Longleftrightarrow \quad \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1/3 & 1 \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ -1/3 \end{pmatrix},$$

which gives

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix}.$$

Then

$$Ux = c \quad \Longleftrightarrow \quad \begin{pmatrix} 1 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & -1/3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \\ -1 \end{pmatrix},$$

which gives the solution of equation (4.10)*:*

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -8/3 \\ -1/3 \\ 3 \end{pmatrix}.$$

For general $n \times n$ systems, we can apply the previous LU factorization that **only involves** the row operation $L_{i,j}(\alpha)$. Therefore, we use the notation L_k for $k = 1, 2, \ldots$, where each L_k is a lower triangular matrix representing a particular operation $L_{i,j}(\alpha)$. The goal of the Gaussian elimination for a $n \times n$ matrix is to find a sequence of row operations, such that

$$L_{n-1} L_{n-2} \cdots L_1 A = U,$$

where U is upper triangular. Setting $L = L_1^{-1} L_2^{-1} \cdots L_{n-1}^{-1}$, which is lower triangular automatically gives the LU decomposition of A.

Definition 4.4 (Unit triangular matrix)**.** *Let $B \in \mathbb{R}^{n \times n}$ be a lower (resp. upper) triangular matrix. We say that B is a* **unit triangular** *matrix if the entries on the main diagonal of B are 1, i.e., $B_{ii} = 1$ for $i = 1, \ldots, n$.*

Exercise. Show that if L is a unit lower triangular matrix, then its inverse L^{-1} is also a unit lower triangular matrix.
If L_1 and L_2 are both unit lower triangular matrices, then $L_1 L_2$ is also unit lower triangular.

Remark 4.1. *The lower triangular matrix L in the LU decomposition of a matrix A is always a unit triangular matrix, i.e., $L_{ii} = 1$ for $i = 1, \ldots, n$, this is due to the fact that products of unit lower triangular matrices is again unit lower triangular, and the matrices $L_1, \ldots, L_N$ are by nature unit lower triangular.*

The procedure for a general $n \times n$ matrix can be summarized as follows: for the kth row, $k = 1, \ldots, n$, and any row i with $i > k$, we perform elementary row operations so that for any $j > k$,

$$\begin{array}{r} \text{row } k: \\ \text{row } i: \end{array} \begin{pmatrix} \tilde{a}_{kk} & \tilde{a}_{kj} \\ \tilde{a}_{ik} & \tilde{a}_{ij} \end{pmatrix} \to \begin{pmatrix} \tilde{a}_{kk} & \tilde{a}_{kj} \\ 0 & \tilde{a}_{ij} - \frac{\tilde{a}_{ik}}{\tilde{a}_{kk}} \tilde{a}_{kj} \end{pmatrix}$$

This is equivalent to multiplying A with the elementary matrix L_k of the form

$$\begin{array}{r} \text{row } k: \\ \text{row } i: \end{array} \begin{pmatrix} 1 & 0 \\ -\frac{\tilde{a}_{ik}}{\tilde{a}_{kk}} & 1 \end{pmatrix}.$$

Example 4.6. *Solve the following system by Gaussian elimination*

$$\begin{pmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 12 \\ 34 \\ 27 \\ -38 \end{pmatrix}.$$

Let

$$A_1 = A = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{pmatrix},$$

then we have

$$L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ -\frac{1}{2} & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix}.$$

This gives

$$A_2 = L_1 A_1 = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{pmatrix}.$$

Looking at the second row of A_2, we know

$$L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -3 & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 1 \end{pmatrix},$$

which yields

$$A_3 = L_2A_2 = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{pmatrix},$$

Further, checking the third row of A_3, we get

$$L_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -2 & 1 \end{pmatrix},$$

which helps us find

$$A_4 = L_3A_3 = \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & 5 \\ 0 & 0 & 0 & -3 \end{pmatrix}.$$

Now, we know

$$U = A_4 = L_3L_2L_1A.$$

This implies

$$\begin{aligned} A = {} & L_1^{-1}L_2^{-1}L_3^{-1}U \\ = {} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & -\frac{1}{2} & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \end{pmatrix} \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{pmatrix} \\ = {} & \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{pmatrix} \begin{pmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{pmatrix} \\ = {} & \begin{pmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{pmatrix}. \end{aligned}$$

4.8.1 Complexity of LU factorization

In this section we compute the complexity of the LU factorization. Looking the the first column of the matrix A, we perform row operations $r_j \mapsto r_j - \frac{a_{1j}}{a_{11}} r_1$ for $j = 2, \ldots, n$. Each of these row operations requires n multiplications and n subtractions.

Since we perform this for $n-1$ rows the total cost for eliminating all entries in the first column below a_{11} is $2n(n-1)$.

In a similar fashion, to remove all entries in the second column below a_{22}, we perform row operations $r_j \mapsto r_j - \frac{a_{2j}}{a_{22}} r_2$ for $j = 3, \dots, n$. Each of these operations costs $n-1$ multiplications and $n-1$ subtractions performed for $n-2$ rows, leading to a total cost of $2(n-1)(n-2)$.

By induction, we obtain a total cost to transform A into upper triangular form of

$$\begin{aligned}\sum_{i=1}^{n} 2(n-i)(n-i+1) \underbrace{=}_{j=n-i} \sum_{j=0}^{n-1} 2j(j+1) &= \sum_{j=0}^{n} 2j^2 + 2j \\ &= \frac{2}{3}(n^3 - n) \approx n^3 \text{ for large } n.\end{aligned}$$

This is **roughly twice** the computation complexity of the Cholesky algorithm, which is to be expected since we do not have the advantage of symmetry any more for general matrices.

4.8.2 Issues with LU factorization

Although the LU factorization is intuitively simple to understand, there are several technical issues that prevents its application to general matrices.

Non-uniqueness. For the matrix

$$A = \begin{pmatrix} 0 & 0 \\ 1 & 2 \end{pmatrix},$$

consider the LU factorization

$$A = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \begin{pmatrix} 1 & d \\ 0 & 1 \end{pmatrix} \Rightarrow a = 0, \quad b = 1, \quad d + c = 2.$$

This means that for any pair $(c, d) \in \mathbb{R}^2$ such that $c+d = 2$, we have a LU factorization of A, and so this shows that not every matrix has a unique LU factorization. However, if the matrix is invertible, then the LU factorization, when it exists, is unique.

Theorem 4.4. *If A is an invertible matrix with an LU decomposition. Then, the decomposition is unique.*

Proof. Let $A = L_1 U_1 = L_2 U_2$ be two LU decompositions of an invertible matrix A. Since $\det(A) \neq 0$ and by properties of the determinant:

$$\det(L_1)\det(U_1) = \det(L_2)\det(U_2),$$

the determinants of L_1, U_1, L_2 and U_2 are non-zero. In particular, L_2 and U_1 are invertible and so

$$L_2^{-1} L_1 = U_2 U_1^{-1}.$$

The inverse of a lower (resp. upper) triangular matrix is again lower (resp. upper) triangular, and the product of two lower (resp. upper) triangular matrices is lower (resp. upper) triangular. Hence, $L_2^{-1}L_1$ is lower triangular and $U_2U_1^{-1}$ is upper triangular. In order for the above identity to hold we must have $L_2^{-1}L_1$ and $U_2U_1^{-1}$ to be the same diagonal matrix. Since L_1 and L_2 are unit triangular, it holds that $L_2^{-1}L_1$ must be the identity matrix, and so $L_1 = L_2$ and consequently $U_1 = U_2$. $\square$

Non-existence. For the matrix

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

it is easy to see that A is non-singular. Suppose we write

$$A = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ 0 & u_{22} \end{pmatrix}$$

and so we need

$$l_{11}u_{11} = 0, \quad l_{11}u_{12} = 1, \quad l_{21}u_{11} = 1, \quad l_{21}u_{12} + l_{22}u_{22} = 0.$$

We immediately see a contradiction as the equations $l_{11}u_{11} = 0$ implies either one must be zero, but then no choice of u_{12} or l_{21} can then fulfill $l_{11}u_{12} = 1$ or $l_{21}u_{11} = 1$. A further problem arises when one simply applies the Gaussian elimination procedure to A. Since the first entry a_{11} is zero, we cannot apply the third row operation as we would be dividing by zero!

Below we give a criterion that ensures a matrix A has a LU factorization.

Theorem 4.5. *Let A be an $n \times n$ matrix, and for $k = 1, \dots, n-1$, let $A(k)$ denote the $k \times k$ submatrix of A consisting of the first k rows and k columns of A. If $\det(A(k)) \neq 0$ for $k = 1, \dots, n-1$, then A has a LU factorization.*

Proof. Since $A(1) = (a_{11})$ is invertible, $a_{11} \neq 0$. Then we can use the row operation $L_{i,j}(\alpha)$ to reduce the matrix A into the form

$$A' = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ 0 & a'_{22} & \cdots & a'_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a'_{n2} & \cdots & a'_{nn} \end{pmatrix}$$

Now, the submatrix $A'(2)$ is invertible, since its determinant is the same as the determinant of $A(2)$, and so a'_{22} is non-zero and allows us to reduce the matrix A' further to

$$A'' = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} \\ 0 & 0 & a''_{33} & \cdots & a''_{3n} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & a''_{n3} & \cdots & a''_{nn} \end{pmatrix}.$$

Continued in this manner allows us to reduce A into an upper triangular matrix U, also called row echelon form, where the first $n-1$ diagonal entries of U are non-zero. Furthermore, in the process of the reduction we pick up unit lower triangular matrices $L_1, \dots, L_{N-1}$, such that $L_{N-1} \cdots L_1 A = U$. Setting $L := L_1^{-1} \cdots L_{N-1}^{-1}$ yields a unit lower triangular matrix, and thus A has an LU decomposition. $\square$

Backward instability. Consider for a small constant $\varepsilon \ll 1$ the matrix

$$A = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2+\varepsilon & 5 \\ 4 & 6 & 8 \end{pmatrix}.$$

Then, the LU factorization will result in

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & \frac{2}{\varepsilon} & 1 \end{pmatrix}, \quad U = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \varepsilon & 3 \\ 0 & 0 & 4-\frac{6}{\varepsilon} \end{pmatrix}.$$

Suppose we want to solve $Ax = (1,0,0)^T$ by solving $Ly = (1,0,0)^T$ and then $Ux = y$. If ε is on the order of machine accuracy, then 4 in the entry $4 - \frac{6}{\varepsilon}$ in U is insignificant. Therefore, we have approximated matrices

$$\tilde{U} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & \varepsilon & 3 \\ 0 & 0 & -\frac{6}{\varepsilon} \end{pmatrix}, \quad \tilde{L} = L,$$

leading to

$$\tilde{L}\tilde{U} = \begin{pmatrix} 1 & 1 & 1 \\ 2 & 2+\varepsilon & 5 \\ 4 & 6 & 4 \end{pmatrix} \neq A.$$

The product is significantly different from the original matrix A. In particular, we get the solution $\tilde{x}$ for the $\tilde{L}\tilde{U}$ factorization

$$\tilde{x} = \begin{pmatrix} \frac{11}{2} - \frac{2}{3}\varepsilon \\ -2 \\ \frac{2}{3}\varepsilon - \frac{2}{3} \end{pmatrix} \approx \begin{pmatrix} \frac{11}{2} \\ -2 \\ -\frac{2}{3} \end{pmatrix}.$$

However, if we use the exact factorization LU of A, we get the exact answer

$$x = \begin{pmatrix} \frac{4\varepsilon-7}{2\varepsilon-3} \\ \frac{2}{2\varepsilon-3} \\ -2\frac{\varepsilon-1}{2\varepsilon-3} \end{pmatrix} \approx \begin{pmatrix} \frac{7}{3}\varepsilon \\ -\frac{2}{3} \\ -\frac{2}{3} \end{pmatrix}.$$

Therefore, even if $\tilde{L}$ and $\tilde{U}$ are close to L and U, the product $\tilde{L}\tilde{U}$ is not close to $LU = A$, and the computed solution $\tilde{x}$ is worthless.

These issues then motivate the following subsections on modifications of the LU factorization.

4.9 LDU factorization

As shown previously, the LU factorization of a matrix may not be unique, unless the matrix is invertible. Next we show how to ensure a unique factorization. Suppose we have obtained a factorization of A:

$$A = \tilde{L}\tilde{U},$$

where $\tilde{L}$ is unit lower triangular and $\tilde{U} = (u_{ij})_{1 \le i,j \le n}$ is upper triangular. Let

$$D = \operatorname{diag}(\tilde{U}) = \begin{pmatrix} u_{11} & 0 & \cdots & 0 \\ 0 & u_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix},$$

denote a diagonal matrix with consisting of the diagonal entries of $\tilde{U}$, and set

$$U = D^{-1}\tilde{U}, \quad U_{ij} = \begin{cases} 1 & \text{if } i = j, \\ 0 & \text{if } i > j, \\ \frac{u_{ij}}{u_{ii}} & \text{if } i < j, \end{cases}$$

which yields a unit upper triangular matrix U. Then we can further factorize A as follows:

$$A = L\,D\,U$$

where L and U are **unit** lower and upper triangular matrices, respectively, and D is a diagonal matrix. In particular, we have shown the following existence theorem, which is an immediate consequence of Theorem 4.5.

Theorem 4.6. *Let A be an $n \times n$ matrix, and for $k = 1, \ldots, n-1$, let $A(k)$ denote the $k \times k$ submatrix of A consisting of the first k rows and k columns of A. If $\det(A(k)) \neq 0$ for $k = 1, \ldots, n-1$, then A has a LDU factorization with unit triangular lower (resp. upper) matrix L (resp. U).*

The advantage of the LDU factorization is its **uniqueness**.

Theorem 4.7. *Let $\{(L_i, D_i, U_i)\}_{i=1,2}$ denote two LDU factorizations of a matrix A. Then, $L_1 = L_2$, $D_1 = D_2$ and $U_1 = U_2$.*

Proof. From the assumptions, we have

$$L_2^{-1} L_1 D_1 = D_2 U_2 U_1^{-1}.$$

Let $L \equiv L_2^{-1} L_1$ and $U \equiv U_2 U_1^{-1}$. Then we have

$$L D_1 = D_2 U. \tag{4.11}$$

It is not hard to check that the product of two unit lower (resp. upper) triangular matrices is again a unit lower (resp. upper) triangular matrix. Hence, L and U are unit lower and upper triangular matrices, respectively. Writing the matrices in (4.11) out, we have

$$LD_1 = \begin{pmatrix} (D_1)_{11} & 0 & 0 \\ * & \ddots & 0 \\ * & * & (D_1)_{nn} \end{pmatrix} = \begin{pmatrix} (D_2)_{11} & * & * \\ 0 & \ddots & * \\ 0 & 0 & (D_2)_{nn} \end{pmatrix} = D_2 U,$$

where $*$ are supposedly nonzero entries. Comparing the entries in the left and the right hand sides, we can conclude that $(D_1)_{ii} = (D_2)_{ii}$ for all i, i.e. $D_1 = D_2$. Moreover all these $*$ should be equal to zero. In particular, that would mean that both U and L are nothing but just the identity matrix I. Since $I = L \equiv L_2^{-1} L_1$, we have $L_1 = L_2$, and similarly, $U_1 = U_2$. $\square$

Example 4.7. *Find the LDU factorization of*

$$A = \begin{pmatrix} 2 & 4 \\ 4 & 11 \end{pmatrix}.$$

From (4.9)*, the LU factorization of A is*

$$A = \begin{pmatrix} 2 & 4 \\ 4 & 11 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 4 \\ 0 & 3 \end{pmatrix} \equiv \tilde{L}\,\tilde{U}.$$

Extract the diagonal entries of $\tilde{U}$ and compose the diagonal matrix

$$D = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix},$$

whilst modifying the upper triangular matrix $\tilde{U}$ to

$$U = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix},$$

we see that with $\tilde{L} = L$:

$$L\,D\,U = \begin{pmatrix} 1 & 0 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} = A.$$

We now derive the Cholesky factorization from the LDU factorization. Let A be a SPD matrix with a factorization given as

$$A = L\,D\,U$$

where L (resp. U) is an unit lower (resp. upper) triangular matrix and D is a diagonal matrix. Since A is symmetric,

$$L\,D\,U = A = A^T = U^T D L^T.$$

As L^T is an unit upper triangular matrix, by the uniqueness of the factorization, we have that $L = U^T$ and hence

$$A = U^T DU.$$

Next we claim that all the entries of the diagonal matrix D are positive. Let $\{\mathbf{e}_i\}_{i=1}^n$ be the canonical basis vectors and set $x_i = U^{-1}\mathbf{e}_i$. Then, the positive definiteness of A implies that

$$0 < x^T Ax = \mathbf{e}_i^T (U^{-1})^T U^T DUU^{-1}\mathbf{e}_i = \mathbf{e}_i^T D\mathbf{e}_i = D_{ii}.$$

Hence, $D_{ii} > 0$ for $i = 1, \ldots, n$. This in turn allows us to define the matrix square root $D^{1/2}$ of D by setting $D^{1/2}$ to be a diagonal matrix with entries $\sqrt{D_{ii}}$, i.e.,

$$D^{1/2} = \begin{pmatrix} \sqrt{D_{11}} & 0 & \cdots & 0 \\ 0 & \sqrt{D_{22}} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sqrt{D_{nn}} \end{pmatrix}.$$

Then, we see that

$$A = U^T D^{1/2} D^{1/2} U = (D^{1/2}U)^T (D^{1/2}U) =: \tilde{U}^T \tilde{U},$$

where $\tilde{U} = D^{1/2}U$ is an upper triangular matrix.

4.10 Partial pivoting

Pivoting is the action in Gaussian elimination (or any matrix algorithm) that marks an element (the pivot element) in a certain row and convert all elements above or below the pivot element into zeros. Recall at Step k of the LU factorization:

$$\begin{array}{c} k \\ i \end{array} \begin{pmatrix} \tilde{a}_{kk} & \tilde{a}_{kj} \\ \tilde{a}_{ik} & \tilde{a}_{ij} \end{pmatrix} \to \begin{pmatrix} \tilde{a}_{kk} & \tilde{a}_{kj} \\ 0 & \tilde{a}_{ij} - \frac{\tilde{a}_{ik}}{\tilde{a}_{kk}}\tilde{a}_{kj} \end{pmatrix}.$$

In this setting the coefficient $\tilde{a}_{kk}$ is known as the **pivot**. From this we see that the elimination process can continue only when the diagonal entry $\tilde{a}_{kk}$, which is the pivot element at Step k, is nonzero, as it is used as a divisor. If no diagonals are zero in all steps, the LU factorization can carry on till completion. **However**, if the diagonal entry is zero at a certain step, the algorithm cannot continue.

One example is the matrix

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

which does not possess a LU factorization. Another example is

$$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix},$$

which admits a LU factorization, but is a singular matrix. In both cases the algorithm for Gaussian elimination fails. But sometimes, even if the diagonal entry is not zero, but very small, one encounters trouble. For example, considering solving the following simple system:

$$\begin{pmatrix} 0.0001 & 0.5 \\ 0.4 & -0.3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}. \tag{4.12}$$

Let us assume that the computer (or calculator) has only 4 digits of accuracy. Then the true solution is

$$x_1 = 0.9999, \quad x_2 = 0.9998.$$

Using Gaussian Elimination to eliminate the first column, we have

$$\begin{pmatrix} 1 & 0 \\ -\frac{0.4}{0.0001} & 1 \end{pmatrix} \begin{pmatrix} 0.0001 & 0.5 \\ 0.4 & -0.3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -\frac{0.4}{0.0001} & 1 \end{pmatrix} \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}.$$

Simplifying it, we get,

$$\begin{pmatrix} 0.0001 & 0.5 \\ 0 & -2000 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.5 \\ -2000 \end{pmatrix}. \tag{4.13}$$

Notice that the $(2, 2)$-entry is obtained by:

$$-\frac{0.4}{0.0001} \cdot 0.5 - 0.3 = -2000.3 \approx 2000.$$

We see that the information of $a_{22} = -0.3$ in (4.12) is completely wiped out by the elimination. It is as if the original matrix in (4.12) starts out with $a_{22} = 0$, one will get the same matrix as in (4.13) after the elimination.

Solving the upper-triangular system (4.13), we have

$$x_2 = 1, \quad x_1 = \frac{0.5 - 0.5x_2}{0.0001} = 0.$$

We see that the solution, especially x_1, is completely wrong. Thus, if there are small pivots (i.e., the pivot element is small in magnitude), such as the (1,1)-entry 0.0001 in the current example, then Gaussian Elimination may give very inaccurate results.

The remedy is to use **partial pivoting**. Before performing Gaussian elimination in

$$\begin{array}{c} k \\ i \end{array} \begin{pmatrix} \tilde{a}_{kk} & \tilde{a}_{kj} \\ \tilde{a}_{ik} & \tilde{a}_{ij} \end{pmatrix} \rightarrow \begin{pmatrix} \tilde{a}_{kk} & \tilde{a}_{kj} \\ 0 & \tilde{a}_{ij} - \frac{\tilde{a}_{ik}}{\tilde{a}_{kk}} \tilde{a}_{kj} \end{pmatrix}$$

we permute the rows from row k onwards so that the **largest entry in magnitude** in the column becomes the pivot.

Considering system (4.12) again. Using partial pivoting, i.e., we permute the largest entry, which is 0.4, in the first column to the first row:

$$\begin{pmatrix} 0.4 & -0.3 \\ 0.0001 & 0.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.5 \end{pmatrix}. \tag{4.14}$$

Notice that we have to do it for every row when doing the permutation, as well as the right-hand side. Then the Gaussian elimination of (4.14) becomes

$$\begin{pmatrix} 1 & 0 \\ \frac{-0.0001}{0.4} & 1 \end{pmatrix} \begin{pmatrix} 0.4 & -0.3 \\ 0.0001 & 0.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{-0.0001}{0.4} & 1 \end{pmatrix} \begin{pmatrix} 0.1 \\ 0.5 \end{pmatrix}.$$

Now simplifying it, we have

$$\begin{pmatrix} 0.4 & -0.3 \\ 0 & 0.5001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0.1 \\ 0.5000 \end{pmatrix}.$$

Hence the solution is:

$$x_2 = 0.9998, \quad x_1 = \frac{0.1 + 0.3x_2}{0.4} = 0.9999.$$

We see that the solution is accurate up to the specified precision of 4 digits.

4.10.1 Full/Complete pivoting

It is known by experience that partial pivoting is sufficient for all practical problems, and is sufficient to adequately reduce round-off errors. But mathematicians have found examples where partial pivoting fails[8]. In those cases, one has to use **full pivoting** (sometimes also known as **complete pivoting**) to ensure correctness of the solutions. Full pivoting means that one permute rows and columns to bring the largest entry in absolute value in the entire submatrix that remains to be row reduced to the pivoting position. In our case above, we will be solving:

$$\begin{pmatrix} 0.5 & 0.0001 \\ -0.3 & 0.4 \end{pmatrix} \begin{pmatrix} x_2 \\ x_1 \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.1 \end{pmatrix}.$$

Unfortunately, the computations can become very difficult if one uses full pivoting. Furthermore, the improvement in numerical stability is typically outweighted by its reduced efficiency for all but the smallest matrices, and the increase in computation complexity as the maximal element has to be searched for. Most commercial software (like Matlab) only uses partial pivoting[9].

[8]See L.V. Foster, *Gaussian elimination with partial pivoting can fail in practice*, SIAM J. Matrix Anal. Appl. (1994) **15**:1354–1362

[9]In Matlab, given a matrix `A`, to see how partial pivoting and the LU factorization are done step by step, use `rrefmovie(A)`.

4.10.2 LU factorization with partial pivoting

In the last subsection we saw that when the diagonal entry is too small at certain stage of the LU factorization, it may cause some serious problem if one continues with the factorization. In this case, we should take some special strategy before continuing the factorization. The following **pivoting** is one of such strategies:

At the kth stage of the LU factorization, suppose the matrix A becomes $A_k = (a_{ij}^{(k)})$. Then, determine an index p_k for which $|a_{p_k,k}^{(k)}|$ is largest among all $|a_{jk}^{(k)}|$ for $(k \le j \le n)$, and interchange rows k and p_k before proceeding the next step of the factorization.

With the pivoting, the LU factorization process takes the form:

$$U = L_{n-1}P_{n-1} \cdots L_2P_2L_1P_1A \tag{4.15}$$

where P_k is exactly the elementary matrix T_{k,p_k} from Section 4.8 that interchanges row k and row p_k. We mention that it is entirely possible that at certain steps no pivoting is necessary, and in those situations P_k is simply the identity matrix.

Example 4.8. *Solve the following systems using the LU factorization with pivoting:*

$$\begin{pmatrix} -1 & 1 & 0 & -3 \\ 1 & 0 & 3 & 1 \\ 0 & 1 & -1 & -1 \\ 3 & 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 3 \\ 1 \end{pmatrix}$$

It can be done in the following $(n-1) = 3$ steps.

Step 1. *Permutate the rows 1 and 4 using $T_{1,4}$:*

$$\begin{pmatrix} 3 & 0 & 1 & 2 \\ 1 & 0 & 3 & 1 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 0 & -3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 3 \\ 4 \end{pmatrix}.$$

Then, do one step of the standard LU factorization:

$$\begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 0 & 3-\frac{1}{3} & 1-\frac{2}{3} \\ 0 & 1 & -1 & -1 \\ 0 & 1 & \frac{1}{3} & -3+\frac{2}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ -\frac{1}{3} \\ 3 \\ 4+\frac{1}{3} \end{pmatrix}$$

Step 2. *Permute rows 2 and 3 using $T_{2,3}$:*

$$\begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 1 & \frac{1}{3} & -\frac{7}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ -\frac{1}{3} \\ \frac{13}{3} \end{pmatrix}.$$

Then, do one step of the standard LU factorization:

$$\begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 0 & \frac{4}{3} & -\frac{4}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ -\frac{1}{3} \\ \frac{4}{3} \end{pmatrix}.$$

Step 3. *There is no need to do permutation, and so do one step of the standard LU factorization:*

$$\begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 0 & 0 & -\frac{1}{6} - \frac{4}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \\ -\frac{1}{3} \\ \frac{4}{3} + \frac{1}{6} \end{pmatrix}.$$

This gives the solution:

$$x = \begin{pmatrix} 1 \\ 2 \\ 0 \\ -1 \end{pmatrix}.$$

It is interesting to write the above Gaussian elimination process into matrix-forms. We can do as follows. Let

$$A = \begin{pmatrix} -1 & 1 & 0 & -3 \\ 1 & 0 & 3 & 1 \\ 0 & 1 & -1 & -1 \\ 3 & 0 & 1 & 2 \end{pmatrix}.$$

Step 1. *Permute rows 1 and 4 using $P_1 := T_{1,4}$:*

$$T_{1,4}A = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 1 & 0 & 3 & 1 \\ 0 & 1 & -1 & -1 \\ -1 & 1 & 0 & -3 \end{pmatrix}.$$

Then do one step of the standard LU factorization:

$$L_1(T_{1,4}A) = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 1 & -1 & -1 \\ 0 & 1 & \frac{1}{3} & -\frac{7}{3} \end{pmatrix}, \quad L_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \frac{1}{3} & 0 & 0 & 1 \end{pmatrix}.$$

Step 2. *Permute rows 2 and 3 using $P_2 := T_{2,3}$:*

$$T_{2,3}(L_1 T_{1,4} A) = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 1 & \frac{1}{3} & -\frac{7}{3} \end{pmatrix}.$$

Then do one step of the standard LU factorization:

$$L_2(T_{2,3} L_1 T_{1,4} A) = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 0 & \frac{4}{3} & -\frac{4}{3} \end{pmatrix}, \quad L_2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{pmatrix}.$$

Step 3. *. No need for permutation, i.e., $P_3 := I$. Do one step of the standard LU factorization:*

$$L_3(L_2 T_{2,3} L_1 T_{1,4} A) = \begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 0 & 0 & -\frac{9}{6} \end{pmatrix} \equiv U, \quad L_3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{2} & 1 \end{pmatrix}.$$

This gives the desired LU factorization with partial pivoting:

$$\begin{aligned} A &= T_{1,4}^{-1} L_1^{-1} T_{2,3}^{-1} L_2^{-1} L_3^{-1} U \\ &= \begin{pmatrix} -\frac{1}{3} & 0 & 0 & 1 \\ \frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} T_{2,3} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & \frac{1}{2} & 1 \end{pmatrix} U \\ &= \begin{pmatrix} -\frac{1}{3} & 0 & 0 & 1 \\ \frac{1}{3} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & \frac{1}{2} & 1 \end{pmatrix} U \\ &= \begin{pmatrix} -\frac{1}{3} & 1 & \frac{1}{2} & 1 \\ \frac{1}{3} & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 3 & 0 & 1 & 2 \\ 0 & 1 & -1 & -1 \\ 0 & 0 & \frac{8}{3} & \frac{1}{3} \\ 0 & 0 & 0 & -\frac{9}{6} \end{pmatrix} = L' U. \end{aligned}$$

Note that in the above factorization, the matrix

$$L' := T_{1,4}^{-1} L_1^{-1} T_{2,3}^{-1} L_2^{-1} L_3^{-1}$$

is not a lower triangular matrix. However, observe that the matrix

$$P := P_3 \, P_2 \, P_1 = I \, T_{2,3} \, T_{1,4}$$

is a permutation matrix exchanging row 1 with row 4, and row 2 with row 3. Then, one can check that

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

and so when we compute the product PL' we obtain

$$PL' = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} -\frac{1}{3} & 1 & \frac{1}{2} & 1 \\ \frac{1}{3} & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \frac{1}{3} & 0 & 1 & 0 \\ -\frac{1}{3} & 1 & \frac{1}{2} & 1 \end{pmatrix} =: L,$$

which is a lower triangular matrix. Therefore, we find that

$$PA = PL'U = LU,$$

and the LU factorization with partial pivoting for a matrix A actually gives the LU factorization (without pivoting) of the permuted matrix $PA = P_3P_2P_1A$.

More generally, from the formula (4.15) we can write

$$\begin{aligned} U = L_{n-1} \times P_{n-1}L_{n-2}P_{n-1}^{-1} \times P_{n-1}P_{n-2}L_{n-3}P_{n-2}^{-1}P_{n-1}^{-1} \times \cdots \\ \cdots \times (P_{n-1}\cdots P_2)L_1(P_{n-1}^{-1}\cdots P_2^{-1}) \times (P_{n-1}P_{n-2}\cdots P_2P_1)A. \end{aligned}$$

We define $P := P_{n-1}\cdots P_1$,

$$\begin{aligned} &\tilde{L}_{n-1} := L_{n-1}, \quad \tilde{L}_{n-2} := P_{n-1}L_{n-2}P_{n-1}^{-1}, \\ &\tilde{L}_{n-k} := \Big(P_{n-1}\cdots P_{n-k+1}\Big)L_{n-k}\Big(P_{n-1}^{-1}\cdots P_{n-k+1}^{-1}\Big), \end{aligned}$$

and

$$L := \tilde{L}_1^{-1}\tilde{L}_2^{-1}\cdots\tilde{L}_{n-1}^{-1}$$

so that (4.15) becomes

$$PA = LU.$$

Note that P is the permutation matrix summarising all of the interchanging of rows, while one can verify that $\tilde{L}_k$ for each $1 \le k \le n-1$ are unit lower triangular, and so L is also unit lower triangular.

4.11 General non-square linear systems

Up to now, all the methods that we have studied for solving the linear system

$$Ax = b \tag{4.16}$$

apply only when the coefficient $A \in \mathbb{R}^{n\times n}$ is a square matrix. But in many applications, we are also encountered with non-square matrices $A \in \mathbb{R}^{m\times n}$ with $m \neq n$. In this section we shall discuss how to solve such non-square systems.

4.11.1 Overdetermined systems

Assume that A is an $m \times n$ matrix, with $m > n$. Clearly the system (4.16) is unlikely to have a solution as the number of equations is larger than the number of unknowns. This is also called the **overdetermined case**. Suppose for the moment that (4.16) has a solution, and writing A column-wise as

$$A = \begin{pmatrix} | & | & \cdots & | \\ \mathbf{a_1} & \mathbf{a_2} & \cdots & \mathbf{a_n} \\ | & | & \cdots & | \end{pmatrix},$$

we can express (4.16) as

$$\mathbb{R}^m \ni b = x_1\mathbf{a_1} + x_2\mathbf{a_2} + \cdots + x_n\mathbf{a_n},$$

where note that each $\mathbf{a_i}$ is a vector in $\mathbb{R}^m$. This shows that

$$\boxed{\text{The system (4.16) has a solution only when } b \text{ lies in the subspace spanned by the column vectors of } A.}$$

Next we shall study the more general case when b does not lie in the subspace spanned by the column vectors of A. In this case the system (4.16) does not have a solution. Since the error $e_x = Ax - b$ is never zero, in physical or engineering applications, it is often **acceptable** for us to find some vector x_* that minimize the error in a certain sense, i.e.,

$$\|Ax_* - b\| \leq \|Ax - b\| \text{ for all } y \in \mathbb{R}^n.$$

If we take $\|\cdot\| = \|\cdot\|_2$ as the 2-norm, then such a solution is called the **least-squares solution**. Namely,

$$x_* = \mathrm{argmin}_{x\in\mathbb{R}^n}\|Ax - b\|_2^2 = \mathrm{argmin}_{x\in\mathbb{R}^n}\sum_{i=1}^{m}\Big(\sum_{j=1}^{n}a_{ij}x_j - b_i\Big)^2, \tag{4.17}$$

where we recall that

$$\|Ax - b\|_2^2 = \sum_{i=1}^{m}((Ax-b)_i)^2, \quad (Ax-b)_i = \sum_{j=1}^{n}a_{ij}x_j - b_i$$

Let us assume that the columns of A are **linearly independent**. Then it is easy to check that A^TA is symmetric and positive definite, and so A^TA is also invertible. In order to find x_*, we define

$$f(x) = \|Ax - b\|_2^2, \tag{4.18}$$

which is a non-negative function defined on $\mathbb{R}^n$. Then, if x_* is a minimizer of $f(x)$, necessary it satisfies for any $y \in \mathbb{R}^n$

$$\frac{d}{dt}f(x_* + ty)\Big|_{t=0} = 0.$$

Let us now compute the above derivative:

$$\begin{aligned}\frac{d}{dt}f(x_* + ty) &= \frac{d}{dt}\sum_{i=1}^{m}\Big(b_i - \sum_{j=1}^{n} a_{ij}(x_{*,j} + ty_j)\Big)^2 \\ &= \sum_{i=1}^{m} 2\Big(\sum_{j=1}^{n} a_{ij}(x_{*,j} + ty_j) - b_i\Big)\sum_{j=1}^{n} a_{ij}y_j \\ &= 2\sum_{i=1}^{m}(A(x_* + ty) - b)_i(Ay)_i = 2(A(x_* + ty) - b)\cdot Ay,\end{aligned}$$

so that

$$\frac{d}{dt}f(x_* + ty)\Big|_{t=0} = 2(Ax_* - b)\cdot Ay = 0.$$

Now, using the property that

$$Ax \cdot Ay = (A^T A)x \cdot y \text{ for } A \in \mathbb{R}^{m\times n}, \quad x, y \in \mathbb{R}^n,$$

we infer that x_* must satisfy the equation

$$A^T(Ax_* - b)\cdot y = 0 \text{ for any } y \in \mathbb{R}^n.$$

This is only possible for arbitrary vectors $y \in \mathbb{R}^n$ if x_* fulfills the **normal equation**:

$$A^T A x_* = A^T b. \tag{4.19}$$

In particular, our best approximation x_* to an acceptable solution to the linear system (4.16) is given as

$$\boxed{x_* = (A^T A)^{-1}A^T b}$$

if $(A^T A)$ is invertible.

Example 4.9. *Consider the overdetermined system*

$$\begin{aligned}2x &= 6\\ 3x &= 6\end{aligned}$$

where in matrix form we define

$$A = \begin{pmatrix}2\\3\end{pmatrix}, \quad b = \begin{pmatrix}6\\6\end{pmatrix}.$$

Notice that there is no solution to the problem $Ax = b$ for $x \in \mathbb{R}$, since the first equation gives $x_1 = 3$ and the second equation gives $x_2 = 2$. Is there a compromise between the two? We compute the normal equation to get

$$x_* = (A^T A)^{-1}A^T b = \frac{30}{13}.$$

Observe that

$$\begin{aligned}
\|Ax_1 - b\|_2 &= \|(0,3)\|_2 = 3, \\
\|Ax_2 - b\|_2 &= \|(-2,0)\|_2 = 2, \\
\|Ax^* - b\|_2 &= \|(-48/13, 12/13)\|_2 = 1.664100,
\end{aligned}$$

and so x_* has the smaller error when compared to x_1 and x_2.

4.11.2 Underdetermined systems

In the case where $m < n$, i.e., there are more unknowns than equations, the system $Ax = b$ is said to be **underdetermined**. There is a possibility that many (even infinitely many) approximate solutions exist to the minimization problem

$$\min \mapsto f(x) = \|Ax - b\|_2^2.$$

Then, it is common to seek the approximate solution with the minimum norm (typically the 2-norm), which we denote by x^*, so that

$$x^* = \operatorname{argmin}_{x \in \mathbb{R}^n \text{ with } Ax=b} \|x\|_2^2. \tag{4.20}$$

In contrast to the overdetermined case, we minimize over a subset of $\mathbb{R}^n$, namely the set $\Sigma := \{x \in \mathbb{R}^n : Ax = b\}$ of solutions to the problem $Ax = b$. However, the mathematics become more difficult when minimizing over this subspace Σ compared to minimizing over the whole space $\mathbb{R}^n$. In order to derive what equations x^* must satisfy, we introduce the Lagrangian $L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$:

$$L(x, \mu) := \|x\|_2^2 + \mu^T(b - Ax) = \sum_{i=1}^n |x_i|^2 + \sum_{j=1}^m \mu_j \Big(b_j - \sum_{k=1}^n a_{jk} x_k\Big),$$

where $\mu \in \mathbb{R}^m$ is the **Lagrange multiplier** for the constraint $Ax = b$ in (4.20). The role of the Lagrange multiplier is to relax our minimization problem from the subset $\Sigma := \{x \in \mathbb{R}^n : Ax = b\}$ to the whole space $\mathbb{R}^n$. Notice, that if x^* is a solution to (4.20), then $Ax^* = b$ and $\|x^*\| \leq \|x\|$ for any other solution $x \in \Sigma$, and thus L achieves its minimum at x^* for any vector μ. Hence, the partial derivatives of L with respect to x and μ must vanish at x^*. We compute

$$\begin{aligned}
\frac{\partial L}{\partial \mu_j}\bigg|_{x=x^*} &= (b - Ax^*)_j \quad \text{for } 1 \leq j \leq m, \\
\frac{\partial L}{\partial x_i}\bigg|_{x=x^*} &= 2x_i^* - (A^T\mu)_i \quad \text{for } 1 \leq i \leq n.
\end{aligned}$$

Setting the partial derivatives to zero gives

$$b = Ax^*, \quad x^* = \frac{1}{2}A^T\mu \quad \Rightarrow \quad b = \frac{1}{2}AA^T\mu,$$

and if (AA^T) is invertible we find that

$$\mu = 2(AA^T)^{-1}b,$$

and so

$$\boxed{x^* = \frac{1}{2}A^T\mu = A^T(AA^T)^{-1}b.}$$

Example 4.10. *Consider the equation*

$$2x_1 + 3x_2 = 5 \quad \Leftrightarrow \quad \begin{pmatrix} 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 5,$$

with $A = \begin{pmatrix} 2 & 3 \end{pmatrix} \in \mathbb{R}^{2\times 1}$ and $b = 5 \in \mathbb{R}$. Notice that there are an infinite number of solutions. In particular for any $t \in \mathbb{R}$, the pair $(t, \frac{1}{3}(5-2t))$ is a solution. Hence, we shall pick the solution x^* with the smallest 2-norm. Computing for x^* gives

$$x^* = \begin{pmatrix} 10/13 \\ 15/13 \end{pmatrix},$$

and one can check that

$$\|(t, \tfrac{1}{3}(5-2t))\|_2 = \frac{1}{3}\sqrt{(3t^2 - 20t + 25)}$$

attains its minimum value at $t = \frac{10}{13}$, and so x^* has the smallest 2-norm amongst all possible solutions to the problem.

5 Systems of nonlinear equations

5.1 Newton's method for systems

For $n \in \mathbb{N}$, we consider solving the nonlinear system

$$F(\mathbf{x}) = \begin{pmatrix} f_1(x_1, \dots, x_n) \\ f_2(x_1, \dots, x_n) \\ \vdots \\ f_n(x_1, \dots, x_n) \end{pmatrix} = \underline{0} \in \mathbb{R}^n,$$

where $\mathbf{x} = (x_1, \dots, x_n)$. We recall the **Jacobian matrix** $DF(\mathbf{x})$ is given as

$$DF(\mathbf{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix},$$

which is the generalization of the derivative $f'(x)$ to higher dimensions. Then, if we invoke the form for Newton's method in the case $n = 1$, we would write

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{F(\mathbf{x}_k)}{DF(\mathbf{x}_k)}.$$

To be more precise, $\frac{1}{DF(\mathbf{x}_k)}$ should be the inverse Jacobian matrix, which then leads to Newton's method:

For $k = 0, 1, 2, \dots$, do the following

1. Compute $\mathbf{d}_k$ by solving
$$DF(\mathbf{x}_k)\mathbf{d}_k = -F(\mathbf{x}_k).$$
2. Update $\mathbf{x}_{k+1}$ by
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k.$$

Example 5.1. *Consider solving the nonlinear system*

$$f_1(x_1, x_2) = 2x_1 + x_1x_2 - 1 = 0,$$
$$f_2(x_1, x_2) = 2x_2 - x_1x_2 + 1 = 0,$$

with Newton's method for the initial guess

$$\mathbf{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

We first compute

$$DF(\mathbf{x}) = \begin{pmatrix} 2+x_2 & x_1 \\ -x_2 & 2-x_1 \end{pmatrix}.$$

For the first iteration

$$F(\mathbf{x}_0) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad DF(\mathbf{x}_0) \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}.$$

Therefore,

$$\mathbf{x}_1 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}^{-1} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1/2 \\ -1/2 \end{pmatrix}.$$

For the second iteration:

$$F(\mathbf{x}_1) = \begin{pmatrix} -1/4 \\ 1/4 \end{pmatrix}, \quad DF(\mathbf{x}_1) = \begin{pmatrix} 3/2 & 1/2 \\ 1/2 & 3/2 \end{pmatrix}.$$

Therefore,

$$\mathbf{x}_2 = \begin{pmatrix} 1/2 \\ -1/2 \end{pmatrix} - \begin{pmatrix} 3/2 & 1/2 \\ 1/2 & 3/2 \end{pmatrix}^{-1} \begin{pmatrix} -1/4 \\ 1/4 \end{pmatrix} = \begin{pmatrix} 3/4 \\ -3/4 \end{pmatrix}.$$

It can be verified that

$$\mathbf{x}_3 = \begin{pmatrix} 0.875 \\ -0.875 \end{pmatrix}, \quad \mathbf{x}_4 = \begin{pmatrix} 0.9325 \\ -0.9325 \end{pmatrix},$$

and clearly the sequence $(\mathbf{x}_k)$ converges to the limit $\mathbf{x}^* = (1, -1)^T$.

Definition 5.1. *A sequence of vectors $(\mathbf{x}_k)$ converges with order $p > 1$ to $\mathbf{x}^* \in \mathbb{R}^n$ if there exists a constant $C > 0$ such that*

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \le C\|\mathbf{x}_k - \mathbf{x}^*\|^p \text{ for all } k.$$

If $p = 1$ and $C < 1$, then we say $(\mathbf{x}_k)$ converges to $\mathbf{x}^*$ linearly.

Think about why we need $C \in (0, 1)$ for the case $p = 1$.

Lemma 5.1 (Banach lemma)**.** *Let $A, B \in \mathbb{R}^{n\times n}$ be matrices such that $\|I - BA\| < 1$, i.e., B is almost an inverse for A. Then, both A and B are invertible with*

$$\|A^{-1}\| \le \frac{\|B\|}{1 - \|I - BA\|}, \quad \|B^{-1}\| \le \frac{\|A\|}{1 - \|I - BA\|}.$$

Proof. For $C \in \mathbb{R}^{n\times n}$, if $\|C\| < 1$ then $I - C$ is invertible. Indeed, suppose for a contradiction, $I - C$ is singular. Then, $I - C$ has an eigenvector $\mathbf{z}$ with corresponding eigenvalue $\lambda = 0$. Moreover,

$$C\mathbf{z} = (I - (I - C))\mathbf{z} = \mathbf{z},$$

and so $\mathbf{z}$ is an eigenvector for C with corresponding eigenvalue $\lambda = 1$. Hence,

$$\frac{\|C\mathbf{z}\|}{\|\mathbf{z}\|} = 1 \leq \sup_{\mathbf{y}} \frac{\|C\mathbf{y}\|}{\|\mathbf{y}\|} = \|C\|,$$

which contradicts the assumption.

Since $I - C$ is invertible, a short calculation shows that

$$(I - C)^{-1} = I + C + C^2 + C^3 + \cdots ,$$

which is also known as the **Neumann series**. Hence, if $\|I - BA\| < 1$, we know $BA = I - (I - BA)$ is invertible, and so both A and B are invertible. Indeed, we can write

$$A^{-1} = (BA)^{-1}B = (I - (I - BA))^{-1}B = (I + (I - BA) + (I - BA)^2 + \cdots)B,$$

which implies

$$\|A^{-1}\| \leq \frac{\|B\|}{1 - \|I - BA\|}.$$

The same argument also applies for the computation of B^{-1}. □

For the convergence analysis, we use the notation

$$B_\delta(\mathbf{x}) = \{\mathbf{y} \in \mathbb{R}^n;\ \|\mathbf{y} - \mathbf{x}\| < \delta\}$$

as the ball centered at $\mathbf{x}$ with radius δ. Then we have the following result.

Lemma 5.2. *Suppose there is a solution $\mathbf{x}^*$ to $F(\mathbf{x}) = \mathbf{0}$ and the Jacobian matrix $DF(\mathbf{x}^*)$ is invertible. Furthermore, assume there is a region $\Omega \subset \mathbb{R}^n$ with $\mathbf{x}^* \in \Omega$ such that the Jacobian is Lipschitz continuous, i.e.,*

$$\|DF(\mathbf{x}) - DF(\mathbf{y})\| \leq \gamma\|\mathbf{x} - \mathbf{y}\| \tag{5.1}$$

for some $\gamma > 0$ and for all $\mathbf{x}, \mathbf{y} \in \Omega$. Then, there exists $\delta > 0$ such that for all $\mathbf{z} \in B_\delta(\mathbf{x}^*)$ the following properties hold:

$$\|DF(\mathbf{x})\| \leq 2\|DF(\mathbf{x}^*)\|, \tag{5.2a}$$

$$\|DF(\mathbf{x})^{-1}\| \leq 2\|DF(\mathbf{x}^*)^{-1}\|. \tag{5.2b}$$

Proof. Let δ be small enough such that $B_\delta(\mathbf{x}^*) \subset \Omega$. For all $\mathbf{x} \in B_\delta(\mathbf{x}^*)$ the condition (5.1) implies that we have

$$\begin{aligned}\|DF(\mathbf{x})\| &\le \|DF(\mathbf{x}^*)\| + \|DF(\mathbf{x}) - DF(\mathbf{x}^*)\| \\ &\le \|DF(\mathbf{x}^*)\| + \gamma\|\mathbf{x} - \mathbf{x}^*\| \le \|DF(\mathbf{x}^*)\| + \gamma\delta,\end{aligned}$$

which implies (5.2a) once we choose δ sufficiently small so that $\gamma\delta \le \|DF(\mathbf{x}^*)\|$. For the next result (5.2b), notice that

$$\begin{aligned}\|I - DF(\mathbf{x}^*)^{-1}DF(\mathbf{x})\| &= \|DF(\mathbf{x}^*)^{-1}(DF(\mathbf{x}^*) - DF(\mathbf{x}))\| \\ &\le \|DF(\mathbf{x}^*)^{-1}\|\gamma\|\mathbf{x}^* - \mathbf{x}\| \\ &\le \gamma\delta\|DF(\mathbf{x}^*)^{-1}\| < \frac{1}{2},\end{aligned}$$

if we choose $\delta < \frac{1}{2\gamma\|DF(\mathbf{x}^*)^{-1}\|}$. Then, setting $A = DF(\mathbf{x})$ and $B = DF(\mathbf{x}^*)^{-1}$ in the Banach Lemma (Lemma 5.1), we get

$$\|DF(\mathbf{x})^{-1}\| \le \frac{\|DF(\mathbf{x}^*)^{-1}\|}{1 - \|I - DF(\mathbf{x}^*)^{-1}DF(\mathbf{x})\|} \le 2\|DF(\mathbf{x}^*)^{-1}\|.$$

□

Theorem 5.1 (Quadratic convergence of Newton's method)**.** *Suppose there is a solution $\mathbf{x}^*$ to $F(\mathbf{x}) = \mathbf{0}$ and the Jacobian matrix satisfies* (5.1) *and $DF(\mathbf{x}^*)$ is invertible. Then, there exist constants $K > 0$ and $\delta > 0$ such that for any $\mathbf{x}_0 \in B_\delta(\mathbf{x}^*)$, the sequence $\{\mathbf{x}_n\}$ generated by the Newton's method satisfies $\mathbf{x}_n \in B_\delta(\mathbf{x}^*)$, and*

$$\|\mathbf{x}_{n+1} - \mathbf{x}^*\| \le K\|\mathbf{x}_n - \mathbf{x}^*\|^2, \quad n = 0, 1, 2, \cdots.$$

So Newton's method converges quadratically.

Proof. By the fundamental theorem of calculus, we have

$$F(\mathbf{y}) = F(\mathbf{x}) + \int_0^1 DF(\mathbf{x} + t(\mathbf{y} - \mathbf{x}))(\mathbf{y} - \mathbf{x})\, dt.$$

Using the definition of Newton's method

$$\begin{aligned}&\mathbf{x}_{k+1} - \mathbf{x}^* \\ &\quad= \mathbf{x}_k - \mathbf{x}^* - DF(\mathbf{x}^*)^{-1}F(\mathbf{x}_k) \\ &\quad= DF(\mathbf{x}_k)^{-1}\int_0^1 DF(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*)\, dt - DF(\mathbf{x}_k)^{-1}F(\mathbf{x}_k) \\ &\quad= DF(\mathbf{x}_k)^{-1}\int_0^1 (DF(\mathbf{x}_k)(\mathbf{x}_k - \mathbf{x}^*) - \underbrace{F(\mathbf{x}^*)}_{=\mathbf{0}} - DF(\mathbf{x}^* + t(\mathbf{x}_k - \mathbf{x}^*))(\mathbf{x}_k - \mathbf{x}^*))\, dt \\ &\quad= DF(\mathbf{x}_k)^{-1}\int_0^1 (DF(\mathbf{x}_k) - DF(\mathbf{x}^* + t(\mathbf{x}_k - \mathbf{x}^*)))(\mathbf{x}_k - \mathbf{x}^*)\, dt.\end{aligned}$$

Then, with (5.2a) and (5.2b) we get

$$\begin{aligned}\|\mathbf{x}_{k+1} - \mathbf{x}^*\| &\leq 2\|DF(\mathbf{x}^*)^{-1}\|\gamma \int_0^1 \|(1-t)(\mathbf{x}_k - \mathbf{x}^*)\|\|\mathbf{x}_k - \mathbf{x}^*\| \, dt \\ &\leq 2\gamma\|DF(\mathbf{x}^*)^{-1}\|\|\mathbf{x}_k - \mathbf{x}^*\|^2 \int_0^1 (1+t) \, dt \leq K\|\mathbf{x}_k - \mathbf{x}^*\|^2.\end{aligned}$$

□

5.2 Broyden's method

For scalar nonlinear equations $f(x) = 0$, the computation of the derivative $f'(x_k)$ at iterate x_k may not be possible in some applications. Similarly, in Newton's method for systems of nonlinear equations, the computation and inversion of the Jacobian matrix $DF(\mathbf{x}_k)$ at each iteration can be unfeasible. This motivates replacing $DF(\mathbf{x}_k)$ with a matrix A_k that is simpler to compute, leading to the following Quasi-Newton method:

For $k = 0, 1, 2, \ldots$, do the following

1. Compute $\mathbf{d}_k$ by solving
$$A_k \mathbf{d}_k = -F(\mathbf{x}_k).$$
2. Update $\mathbf{x}_{k+1}$ by
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k.$$
3. Update A_k to A_{k+1} according to some rules.

By imposing the following properties for A_k, we arrive at **Broyden's method**, which is the topic of this section:

1. (Secant condition) $A_k(\mathbf{x}_k - \mathbf{x}_{k-1}) = F(\mathbf{x}_k) - F(\mathbf{x}_{k-1})$;
2. (Rank-one update) $A_k = A_{k-1} + \mathbf{p}_k \otimes \mathbf{d}_{k-1}$ for some vector $\mathbf{p}_k \in \mathbb{R}^n$ and $\mathbf{d}_{k-1} = -A_{k-1}^{-1} F(\mathbf{x}_{k-1})$;
3. (Orthogonal property) If $\mathbf{y} \cdot (\mathbf{x}_k - \mathbf{x}_{k-1}) = 0$, then $A_k \mathbf{y} = A_{k-1}\mathbf{y}$.

In the above $\otimes$ is the vector tensor product, and for any two vectors $\mathbf{u}$ and $\mathbf{v}$, the matrix $\mathbf{u} \otimes \mathbf{v}$ is defined as

$$\mathbf{u} \otimes \mathbf{v} = \begin{pmatrix} u_1 v_1 & u_1 v_2 & \cdots & u_1 v_n \\ u_2 v_1 & u_2 v_2 & \cdots & u_2 v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_n v_1 & u_n v_2 & \cdots & u_n v_n \end{pmatrix}.$$

Since each column of $\mathbf{u} \otimes \mathbf{v}$ is a constant multiple of the vector $\mathbf{u}$, we see that that $\mathbf{u} \otimes \mathbf{v}$ has rank one. Furthermore, the following formula will be useful:

$$(\mathbf{u} \otimes \mathbf{v})\mathbf{w} = (\mathbf{v} \cdot \mathbf{w})\mathbf{u}.$$

Let us now briefly give some reasoning for the above conditions.

1. Ideally, A_k should approximate the Jacobian $DF(\mathbf{x}_k)$. In the case $n = 1$, the Secant condition can be seen as
$$A_k = \frac{F(x_k) - F(x_{k-1})}{x_k - x_{k-1}} \approx F'(x_k),$$
but for the case $n > 1$, the above expression does not make sense, and so it is replaced by the Secant condition.

2. We wish for a simple formula to update A_{k-1} to A_k using information such as $\mathbf{d}_k$ available at the kth iteration. One simple way to do so is to consider a rank-one update, so that we only need to compute the vector $\mathbf{p}_k$ in order to obtain the new approximation matrix.

3. While the Secant condition gives information about A_k in the direction $\mathbf{d}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$, and we expect that along the direction $\mathbf{d}_k$, the matrix A_k should mimic the behaviour of the true Jacobian along the line joining $\mathbf{x}_{k-1}$ to $\mathbf{x}_k$. But for all other directions orthogonal to $\mathbf{d}_k$ there is no information about A_k. Hence, Broyden[10] suggests to use information from the previous matrix A_{k-1}, i.e., $A_k\mathbf{y} = A_{k-1}\mathbf{y}$ for all $\mathbf{y} \cdot \mathbf{d}_k = 0$. The hope is that if the initial matrix A_0 is close to the Jacobian matrix $DF(\mathbf{x}^*)$, then subsequent approximations A_k stay close.

5.2.1 Rank-one update and the "good" Broyden method

Given a matrix C and vector $\mathbf{w}$, $\mathbf{z}$ and $\mathbf{g}$, let us construct a matrix D satisfying the properties:

(1) $D\mathbf{w} = \mathbf{z}$.

(2) $D\mathbf{y} = C\mathbf{y}$ for all vector $\mathbf{y}$ orthogonal to $\mathbf{g}$, i.e., $\mathbf{y} \cdot \mathbf{g} = 0$.

The second property suggests that we should consider D of the form

$$D = C + \mathbf{p} \otimes \mathbf{g}$$

for some vector $\mathbf{p}$. Then, using that $(\mathbf{p} \otimes \mathbf{g})\mathbf{y} = (\mathbf{g} \cdot \mathbf{y})\mathbf{p} = 0$ the second property is satisfied. To identify $\mathbf{p}$ we compute using the first property

$$\mathbf{z} = D\mathbf{w} = C\mathbf{w} + (\mathbf{g} \cdot \mathbf{w})\mathbf{p} \quad \Rightarrow \quad \mathbf{p} = \frac{\mathbf{z} - C\mathbf{w}}{\mathbf{g} \cdot \mathbf{w}}.$$

[10]C.G. Broyden, On the discovery of the "good Broyden" method, Math. Program., Ser. B, 87:209–213 (2000)

Hence, we obtain the formula

$$D = C + \frac{(\mathbf{z} - C\mathbf{w}) \otimes \mathbf{g}}{\mathbf{g} \cdot \mathbf{w}}. \tag{5.3}$$

Let us consider an initial guess $\mathbf{x}_0$ and an initial invertible matrix A_0. Then, Broyden's method initializes as

1. Compute $\mathbf{d}_0 = -A_0^{-1}F(\mathbf{x}_0)$;
2. Update $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{d}_0$.

To obtain a new matrix A_1 satisfying the Secant condition and the orthogonal property that is also a rank-one update of A_0, we choose $D = A_1$, $C = A_0$, $\mathbf{g} = \mathbf{d}_0 = -A_0^{-1}F(\mathbf{x}_0)$, $\mathbf{w} = (\mathbf{x}_1 - \mathbf{x}_0) = \mathbf{d}_0$ and $\mathbf{z} = F(\mathbf{x}_1) - F(\mathbf{x}_0)$ in (5.3) to get

$$A_1 = A_0 + \frac{(F(\mathbf{x}_1) - F(\mathbf{x}_0) - A_0\mathbf{d}_0) \otimes \mathbf{d}_0}{\mathbf{d}_0 \cdot \mathbf{d}_0} = A_0 + \frac{F(\mathbf{x}_1)}{\mathbf{d}_0 \cdot \mathbf{d}_0} \otimes \mathbf{d}_0. \tag{5.4}$$

Then, we can continue with

1. Compute $\mathbf{d}_1 = -A_1^{-1}F(\mathbf{x}_1)$;
2. Update $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{d}_1$,

which leads to the "good" Broyden method:
Select $\mathbf{x}_0$ and an invertible A_0. For $k = 0, 1, 2, \ldots$, do the following:

1. Compute $\mathbf{d}_k = -A_k^{-1}F(\mathbf{x}_k)$;
2. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$;
3. Update

$$A_{k+1} = A_k + \frac{F(\mathbf{x}_{k+1}) - F(\mathbf{x}_k) - A_k\mathbf{d}_k}{\mathbf{d}_k \cdot \mathbf{d}_k} \otimes \mathbf{d}_k;$$

4. Compute A_k^{-1}.

5.2.2 The Sherman–Morrison formula and the "bad" Broyden method

One disadvantage of the "good" Broyden method is that at every iteration a matrix has to be inverted. We now introduce the Sherman–Morrison formula which gives a formula for A_k^{-1} in terms of A_{k-1}^{-1}. This means that by performing only the inversion operation on the initial matrix A_0, we can readily obtain A_1^{-1}, A_2^{-1}, $\ldots$, without inverting a matrix again.

We will make use of the equation (5.3). Firstly, consider two matrices $A, B \in \mathbb{R}^{n\times n}$ and two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^n$ such that B is the rank-one update of A:

$$B = A + \mathbf{u} \otimes \mathbf{v}. \tag{5.5}$$

We start with two computations:

1. If $A\mathbf{x} = \mathbf{u}$, then

$$B\mathbf{x} = A\mathbf{x} + (\mathbf{u}\,\mathbf{v}^T)\mathbf{x} = \mathbf{u} + (\mathbf{v}\cdot\mathbf{x})\mathbf{u} = (1 + \mathbf{v}\cdot\mathbf{x})\mathbf{u},$$

   and hence

$$B^{-1}\mathbf{u} = \frac{\mathbf{x}}{1+\mathbf{v}\cdot\mathbf{x}} = \frac{A^{-1}\mathbf{u}}{1+\mathbf{v}\cdot A^{-1}\mathbf{u}}. \tag{5.6}$$

2. If $\mathbf{x}$ is such that $\mathbf{v}\cdot A^{-1}\mathbf{x} = 0$, i.e., $\mathbf{v}$ is orthogonal to $A^{-1}\mathbf{x}$, then

$$BA^{-1}\mathbf{x} = (A + \mathbf{u}\,\mathbf{v}^T)A^{-1}\mathbf{x} = \mathbf{x} + (\mathbf{v}\cdot A^{-1}\mathbf{x})\mathbf{u} = \mathbf{x},$$

   which yields

$$B^{-1}\mathbf{x} = A^{-1}\mathbf{x}. \tag{5.7}$$

Next, we choose $D = B^{-1}$, $C = A^{-1}$, $\mathbf{g} = A^{-T}\mathbf{v}$, $\mathbf{w} = \mathbf{u}$ and $\mathbf{z} = \frac{A^{-1}\mathbf{u}}{1+\mathbf{v}\cdot A^{-1}\mathbf{u}}$ in the setting of Section 5.2.1, then the equations (5.6) and (5.7) becomes

$$D\mathbf{w} = \mathbf{z}, \quad D\mathbf{x} = C\mathbf{x} \text{ for } \mathbf{x}\cdot\mathbf{g} = 0.$$

By the formula (5.3) we see that

$$B^{-1} = A^{-1} + \frac{[(\frac{A^{-1}\mathbf{u}}{1+\mathbf{v}\cdot A^{-1}\mathbf{u}} - A^{-1}\mathbf{u})\otimes A^{-T}\mathbf{v}]}{\mathbf{v}\cdot A^{-1}\mathbf{u}} = A^{-1} - \frac{A^{-1}(\mathbf{u}\otimes\mathbf{v})A^{-1}}{1+\mathbf{v}\cdot A^{-1}\mathbf{u}}. \tag{5.8}$$

This formula that gives the inverse of a rank one update is known as the **Sherman–Morrison formula**.

Exercise. Convince yourselves that

$$A^{-1} + \frac{[(\frac{A^{-1}\mathbf{u}}{1+\mathbf{v}\cdot A^{-1}\mathbf{u}} - A^{-1}\mathbf{u})\otimes A^{-T}\mathbf{v}]}{\mathbf{v}\cdot A^{-1}\mathbf{u}} = A^{-1} - \frac{A^{-1}(\mathbf{u}\otimes\mathbf{v})A^{-1}}{1+\mathbf{v}\cdot A^{-1}\mathbf{u}}.$$

Let us consider an initial guess $\mathbf{x}_0$ and an initial invertible matrix A_0. Then, Broyden's method initializes as

1. Compute $\mathbf{d}_0 = -A_0^{-1}F(\mathbf{x}_0)$;
2. Update $\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{d}_0$.

In the "good" Broyden method, the next step is to obtain the matrix A_1 from A_0 via a rank-one update, and then compute $\mathbf{d}_1 = -A_1^{-1}F(\mathbf{x}_1)$ and update $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{d}_1$. We now use the formula (5.8). Recall the three conditions:

1. $A_1\mathbf{d}_0 = A_1(\mathbf{x}_1 - \mathbf{x}_0) = F(\mathbf{x}_1) - F(\mathbf{x}_0)$,

2. $A_1 = A_0 + \mathbf{p}_0 \otimes \mathbf{d}_0$ for $\mathbf{p}_0 = \frac{F(\mathbf{x}_1)}{\mathbf{d}_0 \cdot \mathbf{d}_0}$ from (5.4),

3. If $\mathbf{y} \cdot \mathbf{d}_0 = 0$, then $A_1\mathbf{y} = A_0\mathbf{y}$.

Compare with (5.5) we should set

$$B = A_1, \quad A = A_0, \quad \mathbf{u} = \mathbf{p}_0 = \frac{F(\mathbf{x}_1)}{\mathbf{d}_0 \cdot \mathbf{d}_0}, \quad \mathbf{v} = \mathbf{d}_0,$$

and obtain

$$A_1^{-1} = A_0^{-1} - \frac{A_0^{-1}(\frac{F(\mathbf{x}_1)}{\mathbf{d}_0 \cdot \mathbf{d}_0} \otimes \mathbf{d}_0)A_0^{-1}}{1 + \mathbf{d}_0 \cdot A_0^{-1}\frac{F(\mathbf{x}_1)}{\mathbf{d}_0 \cdot \mathbf{d}_0}} = A_0^{-1} - \frac{A_0^{-1}(F(\mathbf{x}_1) \otimes \mathbf{d}_0)A_0^{-1}}{\mathbf{d}_0 \cdot \mathbf{d}_0 + \mathbf{d}_0 \cdot A_0^{-1}F(\mathbf{x}_1)}.$$

Notice that if we have A_0^{-1}, then the above computation for A_1^{-1} only involves matrix multiplication. Furthermore, with the above choices, we see that setting $\mathbf{y} = A_0^{-1}\mathbf{x}$ in (5.7) gives

$$\text{if } \mathbf{d}_0 \cdot \mathbf{y} = 0 \text{ then } A_1^{-1}\mathbf{x} = A_0^{-1}\mathbf{x} = \mathbf{y} \quad \Rightarrow \quad A_1\mathbf{y} = \mathbf{x} = A_0\mathbf{y}.$$

Then, we can continue with $\mathbf{d}_1 = -A_1^{-1}F(\mathbf{x}_1)$ and $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{d}_1$ as before. This leads to the "bad" Broyden method:

Select $\mathbf{x}_0$ and an invertible A_0. For $k = 0, 1, 2, \cdots$, do the following:

1. Compute $\mathbf{d}_k = -A_k^{-1}F(\mathbf{x}_k)$;

2. Update $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$;

3. Compute

$$\mathbf{u}_k = A_k^{-1}F(\mathbf{x}_{k+1}), \quad c_k = \mathbf{d}_k \cdot \mathbf{d}_k + \mathbf{d}_k \cdot \mathbf{u}_k.$$

4. Compute

$$A_{k+1}^{-1} = A_k^{-1} - \frac{1}{c_k}(\mathbf{u}_k \otimes \mathbf{d}_k)A_k^{-1}.$$

Remark 5.1. *The "good" Broyden method involving the formula for A_k is named so as it seems to perform better numerically than the "bad" Broyden method when observed by Broyden and his co-workers*[11].

Example 5.2. *For the system of equations*

$$F(\mathbf{x}) = \begin{pmatrix} x^2 + y^2 + z^2 - 3 \\ x^2 + y^2 - z - 1 \\ x + y + z - 2 \end{pmatrix} = \mathbf{0}$$

[11]A. Griewank, Broyden update, the Good and the Bad!, Documenta. Math. Extra volume 301–315 (2012).

we compute the first two iterates using "bad" Broyden's method with the initial guess $\mathbf{x}_0 = (1, 0, 1)$ *and* $A_0 = DF(\mathbf{x}_0)$. *The Jacobian matrix* $DF(\mathbf{x})$ *and the first matrix* $A_0 = DF(\mathbf{x}_0)$ *are given as*

$$DF(\mathbf{x}) = \begin{pmatrix} 2x & 2y & 2z \\ 2x & 2y & -1 \\ 1 & 1 & 1 \end{pmatrix}, \quad A_0 = \begin{pmatrix} 2 & 0 & 2 \\ 2 & 0 & -1 \\ 1 & 1 & 1 \end{pmatrix}$$

We first compute $\mathbf{d}_0 = -A_0^{-1}F(\mathbf{x}_0)$*:*

$$\mathbf{d}_0 = -\begin{pmatrix} 1/6 & 1/3 & 0 \\ -1/2 & 0 & 1 \\ 1/3 & -1/3 & 0 \end{pmatrix}\begin{pmatrix} -1 \\ -1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/2 \\ -1/2 \\ 0 \end{pmatrix}$$

and update

$$\mathbf{x}_1 = \mathbf{x}_0 + \mathbf{d}_0 = \begin{pmatrix} 3/2 \\ -1/2 \\ 1 \end{pmatrix}.$$

Then, computing $\mathbf{u}_0$ *and* c_0*:*

$$\mathbf{u}_0 = A_0^{-1}F(\mathbf{x}_1) = \begin{pmatrix} 1/4 \\ -1/4 \\ 0 \end{pmatrix}, \quad c_0 = \frac{3}{4},$$

and calculate

$$A_1^{-1} = A_0^{-1} - \frac{4}{3}\begin{pmatrix} 1/8 & 1/8 & 0 \\ -1/8 & -1/8 & 0 \\ 0 & 0 & 0 \end{pmatrix} A_0^{-1} = \frac{1}{18}\begin{pmatrix} 1 & 5 & 3 \\ -7 & 1 & 15 \\ 6 & -6 & 0 \end{pmatrix}.$$

This allows us to compute $\mathbf{d}_1 = (-1/6, 1/6, 0)$ *and* $\mathbf{x}_2 = \mathbf{x}_1 + \mathbf{d}_1 = (4/3, -1/3, 0)$.

The trade-off for using an approximation of the Jacobian matrix in Broyden's method is the reduction of the convergence rate. We expect that Broyden's method may not converge quadratically as in the case of Newton's method, but we now show that Broyden's method converges linearly.

Theorem 5.2 (Convergence of Broyden's method)**.** *Let* $F(\mathbf{x})$ *be differentiable function such that its Jacobian* $DF(\mathbf{x})$ *is Lipschitz continuous with Lipschitz constant* γ *on a convex open set* $D \subset \mathbb{R}^n$, *and suppose* $\mathbf{x}^*$ *is a point such that* $F(\mathbf{x}^*) = \mathbf{0}$ *and the Jacobian* $DF(\mathbf{x}^*)$ *at* $\mathbf{x}^*$ *is invertible. Then, there exist positive constants* ε *and* δ *such that if the initial iterate* $\mathbf{x}_0$ *and matrix* A_0 *are chosen satisfying* $||\mathbf{x}_0 - \mathbf{x}^*|| < \varepsilon$ *and* $||A_0 - DF(x^*)|| \le \delta$, *then the iterates* $\mathbf{x}^n$ *and* A_n *generated by the (good) Broyden's method are all well-defined and the method converges linearly with*

$$\|\mathbf{x}_{k+1} - \mathbf{x}^*\| \le \frac{1}{2}\|\mathbf{x}_k - \mathbf{x}^*\|.$$

5.3 Steepest descent method

We recall for scalar nonlinear equations $f(x) = 0$, the only advantage of the Bisection method over Newton/Quasi-Newton method is that the initial guess does not need to be too close to the true solution x^*. In a similar way, the previous methods in this section can be rather limited if the initial guess is far away from the solution. Steepest descent method is a simple iterative method that makes **no assumptions** on the initial guess. The key idea is if $\mathbf{x}^*$ is a solution to $F(\mathbf{x}) = \mathbf{0}$, then $\mathbf{x}^*$ is a minimum of the function

$$g(\mathbf{x}) = \frac{1}{2}F(\mathbf{x})^T F(\mathbf{x}) = \frac{1}{2}\|F(\mathbf{x})\|_2^2. \tag{5.9}$$

Then, from an initial guess $\mathbf{x}_0$ we should generate a sequence

$$\mathbf{x}_1, \quad \mathbf{x}_2, \quad \cdots, \quad \mathbf{x}_k, \quad \cdots$$

such that

$$g(\mathbf{x}_0) > g(\mathbf{x}_1) > g(\mathbf{x}_2) > \cdots > g(\mathbf{x}_k) > \cdots \tag{5.10}$$

and hope that $\{\mathbf{x}_k\}_{k\in\mathbb{N}}$ converges to $\mathbf{x}^*$. To relate $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$, we consider the update formula

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k,$$

for vector $\mathbf{d}_k \in \mathbb{R}^n$ (called the **search direction**) and scalar $\alpha_k \in \mathbb{R}$ (called the **search step/step size**).

Using Taylor's expansion, we have

$$g(\mathbf{x}_{k+1}) = g(\mathbf{x}_k) + \alpha_k \nabla g(\mathbf{x}_k)\cdot \mathbf{d}_k + \frac{\alpha_k^2}{2}\mathbf{d}_k \cdot D^2 g(\mathbf{y})\mathbf{d}_k,$$

where $D^2 g$ is the Hessian matrix of g, and $\mathbf{y}$ is a point on the line connecting $\mathbf{x}_k$ and $\mathbf{x}_{k+1}$. Therefore, if α_k is sufficiently small, neglecting the Hessian term leads to

$$g(\mathbf{x}_{k+1}) \approx g(\mathbf{x}_k) + \alpha_k \nabla g(\mathbf{x}_k)\cdot \mathbf{d}_k.$$

Hence, if $\alpha_k > 0$, we should choose $\mathbf{d}_k$ such that $\nabla g(\mathbf{x}_k)\cdot \mathbf{d}_k < 0$, and one such choice is $\mathbf{d}_k = -\nabla g(x_k)$. The resulting iterative method is called **Steepest descent** (also sometimes called **Gradient descent**).

For the function g defined in (5.9), if $F(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_n(\mathbf{x}))$, then

$$g(\mathbf{x}) = \frac{1}{2}\sum_{i=1}^n |f_i(\mathbf{x})|^2 \quad \Rightarrow \quad \frac{\partial g}{\partial x_j} = \sum_{i=1}^n f_i(\mathbf{x})\frac{\partial f_i}{\partial x_j} = \sum_{i=1}^n (DF)_{ij}F_i.$$

Hence, we obtain that the gradient of g is given as

$$\nabla g(\mathbf{x}) = DF(\mathbf{x})^T F(\mathbf{x}),$$

where $DF(\mathbf{x})$ is the Jacobian matrix of $F(\mathbf{x})$, and the search direction $\mathbf{d}_k$ should be $\mathbf{d}_k = -\nabla g(\mathbf{x}_k) = -DF(\mathbf{x}_k)^T F(\mathbf{x}_k)$.

It remains choose the step size α_k. Too small values will cause the algorithm to converge slowly, while too large values will cause the algorithm to overshoot the minimum $\mathbf{x}^*$. An optimal value for α_k should satisfy

$$g(\mathbf{x}_k + \alpha_k \mathbf{d}_k) = \min_{\alpha\in\mathbb{R}} g(\mathbf{x}_k + \alpha \mathbf{d}_k),$$

i.e., the new iteration $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$ is optimal in minimizing the function g along the line $\{\mathbf{y} = \mathbf{x}_k + \alpha \mathbf{d}_k \,|\, \alpha \in \mathbb{R}\}$. With this in mind, the steepest descent algorithm is given as follows.

Steepest descent method. Select $\mathbf{x}_0$, for $k = 0, 1, 2, \ldots,$ do the following

1. Find α_k that solves the one-dimensional minimization problem

$$\min_{\alpha\geq 0} g(\mathbf{x}_k - \alpha \nabla g(\mathbf{x}_k)).$$

2. Update $\mathbf{x}_{k+1}$ by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \nabla g(\mathbf{x}_k).$$

5.3.1 Application to linear problems

We can use the steepest descent method to solve linear problems. For a given SPD matrix $A \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{b} \in \mathbb{R}^n$, instead of inverting A to find the solution $\mathbf{x}$ to $A\mathbf{x} = \mathbf{b}$, we consider minimizing the function

$$g(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}.$$

We now show that if $\mathbf{x}^*$ satisfies $g(\mathbf{x}^*) \leq g(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$, then $A\mathbf{x}^* = \mathbf{b}$. Indeed, since $\mathbf{x}^*$ is a minimum of g, we have that $\nabla g(\mathbf{x}^*) = \mathbf{0}$. Now,

$$\begin{aligned}\frac{\partial g}{\partial x_i} &= \frac{\partial}{\partial x_i}\Big(\frac{1}{2}\sum_{k,j=1}^{n} a_{kj}x_jx_k - \sum_{i=1}^{n} b_ix_i\Big) = \frac{1}{2}\Big(\sum_{j=1}^{n} a_{ij}x_j + \sum_{k=1}^{n} a_{ki}x_k\Big) - b_i \\ &= \sum_{j=1}^{n} a_{ij}x_j - b_i,\end{aligned}$$

when we use the symmetry of A. Hence,

$$\nabla g(\mathbf{x}^*) = A\mathbf{x}^* - \mathbf{b} = \mathbf{0}.$$

The search direction $\mathbf{d}_k$ is given by $\mathbf{d}_k = -\nabla g(\mathbf{x}_k) = \mathbf{b} - A\mathbf{x}_k$, and for the step size α_k, which is the minimizer of the function $h(s) = g(\mathbf{x}_k + s\mathbf{d}_k)$, should satisfy

$$h'(\alpha_k) = \nabla g(\mathbf{x}_k + \alpha_k \mathbf{d}_k) \cdot \mathbf{d}_k = 0,$$

and so

$$\Big(A(\mathbf{x}_k + \alpha_k \mathbf{d}_k) - \mathbf{b}\Big) \cdot \mathbf{d}_k = 0 \quad \Rightarrow \quad \alpha_k = \frac{(\mathbf{b} - A\mathbf{x}_k) \cdot \mathbf{d}_k}{\mathbf{d}_k \cdot A\mathbf{d}_k} = \frac{\mathbf{d}_k \cdot \mathbf{d}_k}{\mathbf{d}_k \cdot A\mathbf{d}_k}.$$

For this problem, the steepest descent method can be written more precisely:

Steepest descent method for solving $A\mathbf{x} = \mathbf{b}$. Select $\mathbf{x}_0$, for $k = 0, 1, 2, \ldots$, do the following

1. Compute

$$\mathbf{d}_k = \mathbf{b} - A\mathbf{x}_k, \quad \alpha_k = \frac{\mathbf{d}_k \cdot \mathbf{d}_k}{\mathbf{d}_k \cdot A\mathbf{d}_k}.$$

2. Update $\mathbf{x}_{k+1}$ by

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k.$$

5.3.2 Properties of the steepest descent method

Lemma 5.3. *Let $\mathbf{d}_0$, $\mathbf{d}_1$, … denote the successive search directions from the steepest descent method with optimal step sizes α_0^*, α_1^*, …, i.e.,*

$$g(\mathbf{x}_k + \alpha_k^* \mathbf{d}_k) \leq g(\mathbf{x}_k + s\mathbf{d}_k) \text{ for any } s \in \mathbb{R}.$$

Then, consecutive search directions are orthogonal:

$$\mathbf{d}_k \cdot \mathbf{d}_{k+1} = 0 \text{ for } k = 0, 1, 2, \ldots.$$

This means that the steepest descent method moves in a zig-zag direction to reach the minimum of $g(\mathbf{x})$.

Proof. To obtain the optimal step size α_k^*, we consider the function $h : \mathbb{R} \to \mathbb{R}$ defined as

$$h(s) = g(\mathbf{x}_k + s\mathbf{d}_k).$$

Then, $h'(\alpha_k^*) = 0$ as α_k^* is a minimum of h. Differentiating and the Chain rule gives

$$0 = h'(\alpha_k^*) = \nabla g(\mathbf{x}_k + \alpha_k^* \mathbf{d}_k) \cdot \mathbf{d}_k = -\mathbf{d}_{k+1} \cdot \mathbf{d}_k.$$

□

We now state a theorem for the convergence of the steepest descent method with constant step size ($\alpha_k = \alpha$ for all k).

Theorem 5.3 (Convergence of the steepest descent method with constant step size)**.** *Let $g : \mathbb{R}^n \to \mathbb{R}$ be a twice continuously differentiable function satisfying*

1. *there exists a minimum $\mathbf{x}^*$ such that $g(\mathbf{x}^*) \leq g(\mathbf{x})$ for all $\mathbf{x} \in \mathbb{R}^n$;*

2. $g(\mathbf{y}) \geq g(\mathbf{x}) + \nabla g(\mathbf{x})^T(\mathbf{y} - \mathbf{x})$ for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ (Convexity);

3. the Hessian matrix D^2g satisfies $\|D^2g\| \leq L$ for some constant $L > 0$.

Then, the steepest descent method with $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\nabla g(\mathbf{x}_k)$ for a fixed step size $\alpha \leq \frac{1}{L}$ satisfies

$$g(\mathbf{x}_k) - g(\mathbf{x}^*) \leq \frac{\|\mathbf{x}_0 - \mathbf{x}^*\|^2}{2\alpha k}.$$

This implies R-sublinear convergence of the sequence $(g(\mathbf{x}_k))_{k\in\mathbb{N}}$ to $g(\mathbf{x}^*)$.

Proof. By the assumption on the Hessian and Taylor's theorem, for some $\mathbf{z} \in \mathbb{R}^n$, we see that

$$\begin{aligned} g(\mathbf{y}) &= g(\mathbf{x}) + \nabla g(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}(\mathbf{y} - \mathbf{x})^T D^2 g(\mathbf{z})(\mathbf{y} - \mathbf{x}) \\ &\leq g(\mathbf{x}) + \nabla g(\mathbf{x})^T(\mathbf{y} - \mathbf{x}) + \frac{1}{2}L\|\mathbf{y} - \mathbf{x}\|^2. \end{aligned}$$

Setting $\mathbf{x} = \mathbf{x}_{k-1}$, $\mathbf{y} = \mathbf{x}_k = \mathbf{x}_{k-1} - \alpha\nabla g(\mathbf{x}_{k-1})$, then

$$\begin{aligned} g(\mathbf{x}_k) &\leq g(\mathbf{x}_{k-1}) + \nabla g(\mathbf{x}_{k-1})^T(\mathbf{x}_k - \mathbf{x}_{k-1}) + \frac{1}{2}L\|\mathbf{x}_k - \mathbf{x}_{k-1}\|^2 \\ &= g(\mathbf{x}_{k-1}) - \alpha\|\nabla g(\mathbf{x}_{k-1})\|^2 + \frac{1}{2}L\|\alpha\nabla g(\mathbf{x}_{k-1})\|^2 \\ &= g(\mathbf{x}_{k-1}) - (1 - \tfrac{L\alpha}{2})\alpha\|\nabla g(\mathbf{x}_{k-1})\|^2. \end{aligned}$$

If $\alpha \leq \frac{1}{L}$, then $(1 - \frac{L\alpha}{2}) \geq \frac{1}{2}$ and so

$$g(\mathbf{x}_k) \leq g(\mathbf{x}_{k-1}) - \frac{\alpha}{2}\|\nabla g(\mathbf{x}_{k-1})\|^2.$$

Unless $\mathbf{x}_{k-1} = \mathbf{x}^*$ (then $\nabla g(\mathbf{x}_{k-1}) = \mathbf{0}$), $\|\nabla g(\mathbf{x}_{k-1})\|^2$ is always positive, which means that the value of g strictly decreases along the sequence $(\mathbf{x}_k)_{k\in\mathbb{N}}$ generated by the steepest descent method.

Now, in the convexity assumption substitute $\mathbf{y} = \mathbf{x}^*$, $\mathbf{x} = \mathbf{x}_{k-1}$ to get

$$g(\mathbf{x}_{k-1}) \leq g(\mathbf{x}^*) - \nabla g(\mathbf{x}_{k-1})^T(\mathbf{x}^* - \mathbf{x}_{k-1}) = g(\mathbf{x}^*) + \nabla g(\mathbf{x}_{k-1})^T(\mathbf{x}_{k-1} - \mathbf{x}^*).$$

Then, combining the last two inequalities give

$$\begin{aligned} g(\mathbf{x}_k) &\leq g(\mathbf{x}^*) + \nabla g(\mathbf{x}_{k-1})^T(\mathbf{x}_{k-1} - \mathbf{x}^*) - \frac{\alpha}{2}\|\nabla g(x_{k-1})\|^2 \\ &= g(\mathbf{x}^*) + \frac{1}{2\alpha}\Big(2\alpha\nabla g(\mathbf{x}_{k-1})^T(\mathbf{x}_{k-1} - \mathbf{x}^*) - \alpha^2\|\nabla g(\mathbf{x}_{k-1})\|^2 \mp \|\mathbf{x}_{k-1} - \mathbf{x}^*\|^2\Big) \\ &= g(\mathbf{x}^*) + \frac{1}{2}\Big(\|\mathbf{x}_{k-1} - \mathbf{x}^*\|^2 - \|(\mathbf{x}_{k-1} - \alpha\nabla g(\mathbf{x}_{k-1})) - \mathbf{x}^*\|^2\Big) \\ &= g(\mathbf{x}^*) + \frac{1}{2}\Big(\|\mathbf{x}_{k-1} - \mathbf{x}^*\|^2 - \|\mathbf{x}_k - \mathbf{x}^*\|^2\Big). \end{aligned}$$

Using that $g(\mathbf{x}_k) < g(\mathbf{x}_{k-1}) < \cdots < g(\mathbf{x}_0)$, we obtain

$$\begin{aligned} g(\mathbf{x}_k) - g(\mathbf{x}^*) &\le \frac{1}{k}\sum_{i=1}^{k}(g(\mathbf{x}_i) - g(\mathbf{x}^*)) \le \frac{1}{2\alpha k}\sum_{i=1}^{k}\left(\|\mathbf{x}_{i-1} - \mathbf{x}^*\|^2 - \|\mathbf{x}_i - \mathbf{x}^*\|^2\right) \\ &= \frac{1}{2\alpha k}\left(\|\mathbf{x}_0 - \mathbf{x}^*\|^2 - \|x_k - \mathbf{x}^*\|^2\right) \le \frac{1}{2\alpha k}\|\mathbf{x}_0 - \mathbf{x}^*\|^2. \end{aligned}$$

□

Remark 5.2. *Sometimes the assumption on the Hessian $\|D^2 g\| \le L$ is replaced with the Lipschitz condition*

$$\|\nabla g(\mathbf{x}) - \nabla g(\mathbf{y})\| \le L\|\mathbf{x} - \mathbf{y}\| \text{ for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n. \tag{5.11}$$

We show that they are equivalent if g is any twice continuously differentiable convex function. Take the induced 2-norm for the matrix norm, then $\|D^2 g\|_2^2 = \max\{|\lambda|^2 : \lambda$ is an eigenvalue of $D^2 g\}$. For any non-zero vector $\mathbf{a}$ consider the function

$$h_a(t) = \mathbf{a} \cdot \nabla g(\mathbf{x} + t(\mathbf{y} - \mathbf{x})).$$

Since g is assumed to be twice continuously differentiable, we can apply the Mean value theorem to find a $t_a \in (0, 1)$ such that

$$\frac{h(1) - h(0)}{1 - 0} = h'(t_a) = \mathbf{a} \cdot [D^2 g(\mathbf{x} + t_a(\mathbf{y} - \mathbf{x}))(\mathbf{y} - \mathbf{x})] =: \mathbf{a} \cdot D^2 g(\mathbf{z}_a)(\mathbf{y} - \mathbf{x}).$$

Hence,

$$\mathbf{a} \cdot (\nabla g(\mathbf{y}) - \nabla g(\mathbf{x})) = h(1) - h(0) = \mathbf{a} \cdot D^2 g(\mathbf{z}_a)(\mathbf{y} - \mathbf{x}).$$

Choosing

$$\mathbf{a} = \frac{\nabla(g(\mathbf{y}) - g(\mathbf{x}))}{\|\nabla(g(\mathbf{y}) - g(\mathbf{x}))\|}$$

so that $\|\mathbf{a}\| = 1$ we see that

$$\begin{aligned} \|\nabla(g(\mathbf{y}) - g(\mathbf{x}))\| = \mathbf{a} \cdot (\nabla g(\mathbf{y}) - \nabla g(\mathbf{x})) &\le \sup_{\|\mathbf{a}\|=1} \mathbf{a} \cdot D^2 g(\mathbf{z}_a)(\mathbf{y} - \mathbf{x}) \\ &\le \sup_{\|a\|=1} \|D^2 g(\mathbf{z}_a)\|\|\mathbf{y} - \mathbf{x}\| \le L\|\mathbf{y} - \mathbf{x}\|. \end{aligned}$$

Conversely, suppose the Lipschitz condition (5.11) *holds. For fixed vectors $\mathbf{x}, \mathbf{y}$ and a constant $c > 0$, let*

$$j(t) = \nabla g(\mathbf{x} + tc\mathbf{y}).$$

Note that

$$j'(t) = cD^2 g(\mathbf{x} + tc\mathbf{y})\mathbf{y}.$$

The Mean value theorem suggests that there is some $t_c \in (0,1)$ such that

$$\nabla g(\mathbf{x} + c\mathbf{y}) - \nabla g(\mathbf{x}) = j(1) - j(0) = j'(t_c) = cD^2 g(\mathbf{x} + t_c c\mathbf{y})\mathbf{y}.$$

Taking norms and applying the Lipschitz condition gives

$$\|D^2 g(\mathbf{x} + t_c c\mathbf{y})\mathbf{y}\| = \frac{1}{c}\|\nabla g(\mathbf{x} + c\mathbf{y}) - \nabla g(\mathbf{x})\| \leq L\|\mathbf{y}\|.$$

Sending $c \to 0$ gives

$$\|D^2 g(\mathbf{x})\mathbf{y}\| \leq L\|\mathbf{y}\|.$$

If we choose $\mathbf{y}$ to be any eigenvector of $D^2 g(\mathbf{x})$ then this shows that the absolute value of any eigenvalues of $D^2(g\mathbf{x})$ must be less than or equal to L, and so $\|D^2 g\| \leq L$.

6 Polynomial interpolation

According to the its government, the population of Country X was

Year	1998	2002	2003
Population (in millions)	6.5437	6.787	6.8031

We know it is very expensive to do the population statistics every year. Then a very natural but important question to ask is:

1. (Interpolation) What was the population in the year 2000?
2. (Extrapolation) What will the population be in the year 2005?

The easiest way to answer these questions is to assume a **linear growth** of population in between the census years (i.e., 1998, 2002, and 2003). Then, the population in 2000 (let us denote this by $P(2000)$) should be the average of $P(1998)$ and $P(2002)$, and will be equal to 6.665 millions. The population in 2005, i.e., $P(2005)$, will be 6.835 millions. This kind of interpolation (or extrapolation) technique is called the **piecewise linear interpolation** method.

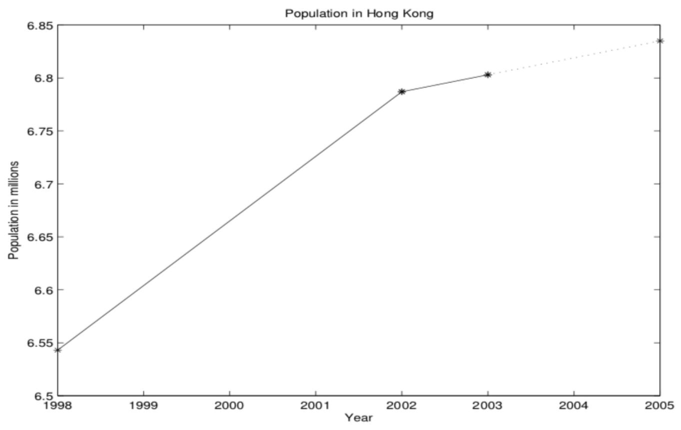

Figure 1: Interpolation using piecewise linear polynomial.

The drawback of the method is that the future forecast will only depend on the last two data $P(2002)$ and $P(2003)$. In general, we would like to involve $P(1998)$ in our forecast to reflect long-term effects. This leads to the **Vandermonde interpolation** method.

6.1 Vandermonde Interpolation

From this interpolation method, we try to get the unique **polynomial** of the highest degree that can pass through all the given data points. Since we have three given data points: $P(1998)$, $P(2002)$, and $P(2003)$, we can only determine a **quadratic** polynomial

$$p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2$$

that fits them. To obtain the coefficients α_i, $i = 0, 1, 2$, we require $p(x)$ to pass through the data points, i.e.,

$$\alpha_0 + \alpha_1 x + \alpha_2 x^2 = P(x), \qquad \text{for } x = 1998, 2002, 2003.$$

Thus in matrix notations, we have:

$$\begin{pmatrix} 1 & 1998 & 1998^2 \\ 1 & 2002 & 2002^2 \\ 1 & 2003 & 2003^2 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix} \begin{pmatrix} P(1998) \\ P(2002) \\ P(2003) \end{pmatrix}. \tag{6.1}$$

Solving the system, we get $\alpha_0 = 36115$, $\alpha_1 = 36.061$ and $\alpha_2 = -0.009$. Thus

$$p(x) = 36115 + 36.061x - 0.009x^2$$

and, in particular, $p(2005) = 6.781$ millions.

This method is simple but **naive** and it has three major drawbacks.

- First the matrix in (6.1) is a well-known **ill-conditioned** matrix called the **Vandermonde** matrix. Its condition number is about 10^{13}. Hence in the computed α_i, only 3 significant digits are accurate if we use double precision, and **none** of the digits will be correct in single precision (see the end of Section 4.4).

- The second drawback is that solving (6.1) requires Gaussian Elimination, which is an $O(n^3)$ process, where n is the degree of the interpolation polynomial. This drawback becomes more serious when n is large or when new data comes in very often, as we will have to update the coefficients as often.

- A third drawback is that if we receive a new data point, we have to reconstruct the Vandermonde matrix, which would now have an additional row and column. Inverting the new matrix cost even more computation effort than the previous one.

As an example, if we are now given the population $P(2004)$, we will have to increase the degree of the interpolation polynomial by one:

$$q(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3.$$

To get the coefficients β_i's, we will then have to solve a system similar to (6.1) but now with dimension 4:

$$\begin{pmatrix} 1 & 1998 & 1998^2 & 1998^3 \\ 1 & 2002 & 2002^2 & 2002^3 \\ 1 & 2003 & 2003^2 & 2003^3 \\ 1 & 2004 & 2004^2 & 2004^3 \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \end{pmatrix} = \begin{pmatrix} P(1998) \\ P(2002) \\ P(2003) \\ P(2004) \end{pmatrix}.$$

Note that the condition number of the 4-by-4 matrix is 10^{19}, and we have no hope of solving the system to any digit of accuracy.

6.1.1 Interpolation with polynomials of degree n

More generally, suppose we are given $n+1$ observation data :

$$\begin{array}{c|c|c|c|c|c} x & x_0 & x_1 & x_2 & \cdots & x_n \\ \hline f(x) & f_0 & f_1 & f_2 & \cdots & f_n \end{array} \tag{6.2}$$

where $x_i \neq x_j$ for all $i \neq j$, we would like to see if it is possible to determine a polynomial $p(x)$ of degree $\leq n$ such that

$$p(x_i) = f_i, \quad i = 0, 1, \ldots, n. \tag{6.3}$$

Let us write

$$p(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \cdots + \alpha_n x^n,$$

then using the conditions (6.3), we have

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix} \tag{6.4}$$

where the coefficient matrix

$$V_a = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \tag{6.5}$$

is called a **Vandermonde matrix**. Clearly, **finding a polynomial** $p(x)$ satisfying the conditions (6.3) amounts to **finding** $(n+1)$ **coefficients** $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that (6.4) is satisfied.

6.2 Lagrange interpolation

The Vandermonde method is related to the basis $\{1, x, x^2, \ldots, x^n\}$ for polynomials of degree n. A different set of basis would result in the different interpolation method. The **Lagrange interpolation** employs the basis functions

$$l_j(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{j-1})(x-x_{j+1})\cdots(x-x_n)}{(x_j-x_0)(x_j-x_1)\cdots(x_j-x_{j-1})(x_j-x_{j+1})\cdots(x_j-x_n)}, \tag{6.6}$$

for $j = 0, 1, \ldots, n$. Note that denominator is fixed and the numerator can be expanded to get a polynomial of degree n, and so $l_j(x)$ is a polynomial of degree n, and

$$l_j(x_j) = 1 \quad \text{and} \quad l_j(x_i) = 0 \quad \forall i \neq j.$$

Then, the **Lagrange interpolation** of the function f is

$$L(x) = f_0 l_0(x) + f_1 l_1(x) + \cdots + f_n l_n(x),$$

which satisfies

$$L(x_i) = f_i, \quad i = 0, 1, 2, \ldots, n.$$

Example 6.1. *Consider the data set*

x	0	1	2
$f(x)$	-2	-1	2

Then, we have

$$\begin{aligned} l_0(x) &= \frac{(x-1)(x-2)}{(-1)(-2)} = (x^2 - 3x + 2)/2, \\ l_1(x) &= \frac{x(x-2)}{(1)(-1)} = -x^2 + 2x, \\ l_2(x) &= \frac{x(x-1)}{(2)(1)} = (x^2 - x)/2, \end{aligned}$$

and so

$$L(x) = -2l_0(x) - l_1(x) + 2l_2(x) = x^2 - 2.$$

6.2.1 Cost of constructing the polynomial

Suppose we are already given $(n+1)$ data points to interpolate:

x	x_0	x_1	x_2	$\cdots$	x_n
$f(x)$	f_0	f_1	f_2	$\cdots$	f_n

The nth degree Lagrange interpolation polynomial is given by

$$\begin{aligned} L_n(x) = f_0 & \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)} \\ &+ f_1 \frac{(x-x_0)(x-x_2)\cdots(x-x_n)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_n)} \\ &+ \quad \cdots \\ &+ f_n \frac{(x-x_0)(x-x_1)\cdots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\cdots(x_n-x_{n-1})}. \end{aligned}$$

Let us denote it by

$$\begin{aligned} L_n(x) = \alpha_0 & (x-x_1)(x-x_2)\cdots(x-x_n) \\ &+ \alpha_1(x-x_0)(x-x_2)\cdots(x-x_n) \\ &+ \cdots + \alpha_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}). \end{aligned} \tag{6.7}$$

The question is **how costly** it is to compute α_i. Notice that for $0 \le i \le n$,

$$\alpha_i = \frac{f_i}{(x_i - x_0)(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}, \tag{6.8}$$

and so it requires n subtractions, $n-1$ multiplications and one division to obtain one α_i for $1 \le i \le n$. Hence the total cost of computing all $(n+1)$ coefficients will be

$$\sum_{i=i}^{n+1} 2n = (n+1)(n+2) \sim O(n^2) \text{ for large } n.$$

6.2.2 Cost of evaluating the polynomial

Suppose we have already obtained all the α_i in (6.7). Our next question is how costly it will be to compute $p_n(t)$ at arbitrary point t. Let us define:

$$P = \prod_{i=0}^{n} (t - x_i)$$

(which can be computed in $(n+1)$ subtractions and n multiplications). Then (6.7) becomes

$$L_n(t) = P \cdot \left(\frac{\alpha_0}{t - x_0} + \frac{\alpha_1}{t - x_1} + \ldots + \frac{\alpha_n}{t - x_n} \right),$$

which can now be computed in $(n+1)$ additions and $(n+2)$ multiplications. Thus $L_n(t)$ can be computed in $(n+1)+n+(n+1)+(n+2) = 4n+4 \sim O(n)$ operations for any t.

6.2.3 Cost of updating the polynomial

Suppose we are given one more data point to interpolate:

x	x_0	x_1	x_2	$\cdots$	x_n	x_{n+1}
$f(x)$	f_0	f_1	f_2	$\cdots$	f_n	f_{n+1}

The $(n+1)$th degree Lagrange interpolation polynomial is given by

$$\begin{aligned} L_{n+1}(x) = f_0 & \frac{(x - x_1)(x - x_2)\cdots(x - x_n)(x - x_{n+1})}{(x_0 - x_1)(x_0 - x_2)\cdots(x_0 - x_n)(x_0 - x_{n+1})} \\ & + f_1 \frac{(x - x_0)(x - x_2)\cdots(x - x_n)(x - x_{n+1})}{(x_1 - x_0)(x_1 - x_2)\cdots(x_1 - x_n)(x_1 - x_{n+1})} \\ & + \quad \cdots \\ & + f_n \frac{(x - x_0)(x - x_1)\cdots(x - x_{n-1})(x - x_{n+1})}{(x_n - x_0)(x_n - x_1)\cdots(x_n - x_{n-1})(x_n - x_{n+1})} \\ & + f_{n+1} \frac{(x - x_0)(x - x_1)\cdots(x - x_{n-1})(x - x_n)}{(x_{n+1} - x_0)(x_{n+1} - x_1)\cdots(x_{n+1} - x_{n-1})(x_{n+1} - x_n)}. \end{aligned}$$

It is very complicated indeed, but we can simplify it using (6.8). In fact, if we rewrite $L_{n+1}(x)$ as

$$\begin{aligned} L_{n+1}(x) = &\beta_0(x-x_1)(x-x_2)\cdots(x-x_n)(x-x_{n+1}) \\ &+\beta_1(x-x_0)(x-x_2)\cdots(x-x_n)(x-x_{n+1}) \\ &+\cdots+\beta_n(x-x_0)(x-x_1)\cdots(x-x_{n-1})(x-x_{n+1}) \\ &+\beta_{n+1}(x-x_0)(x-x_1)\cdots(x-x_{n-1})(x-x_n), \end{aligned}$$

then we can easily check that

$$\beta_i = \begin{cases} \dfrac{\alpha_i}{x_i - x_{n+1}} & \text{if } 0 \le i \le n, \\ \dfrac{f_{n+1}}{(x_{n+1}-x_0)(x_{n+1}-x_1)\cdots(x_{n+1}-x_{n-1})(x_{n+1}-x_n)} & \text{if } i = n+1. \end{cases}$$

Now it is clear that β_i, $0 \le i \le n$, can each be computed in 1 subtraction, 1 multiplication and 1 division, while β_{n+1} can be computed in $(n+1)$ subtractions, $(n+1)$ multiplications and 1 division. Thus L_{n+1} can be updated from L_n with $O(n)$ operations.

The above also demonstrates the slight inconvenience working with Lagrange interpolation when a new data point is added, as it requires re-computation of the polynomial again from scratch, which can be numerically unstable.

6.3 Newton's interpolation

In light of the difficulty that one has to recompute the coefficients of the Lagrange interpolation when a new data point appears, we turn to a different method, called Newton's interpolation, which is based on using

$$1, \quad x-x_0, \quad (x-x_0)(x-x_1), \quad \cdots, \quad (x-x_0)(x-x_1)\cdots(x-x_{n-1})$$

as a basis of the polynomials of degree $\le n$. If $p_n(x)$ be a polynomial of degree $\le n$ satisfying $p(x_i) = f_i$, then we can write

$$\begin{aligned} p_n(x) &= c_0 + c_1(x-x_0) + c_2(x-x_0)(x-x_1) + \cdots \\ &\quad + c_n(x-x_0)(x-x_1)\cdots(x-x_{n-1}) \\ &= c_0 + c_1(x-x_0) + c_2\prod_{i=0}^{1}(x-x_i) + \cdots + c_k\prod_{i=0}^{k-1}(x-x_i) + \cdots \qquad (6.9) \\ &\quad + c_n\prod_{i=0}^{n-1}(x-x_i). \end{aligned}$$

The advantage of Newton's interpolation over Lagrange's interpolation is that if we have a new data point (x_{n+1}, f_{n+1}), we get

$$p_{n+1}(x) = p_n + c_{n+1}\prod_{i=0}^{n}(x-x_i),$$

where we only need to compute the coefficient c_{n+1}, and the coefficients $c_0, \dots, c_n$ remain unchanged.

6.3.1 Divided differences

Consider a function f defined on $[a, b]$, and a set of distinct nodal points in $[a, b]$:

$$a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b.$$

Then, we can define the Divided Differences (DD) of different orders as follows.

Definition 6.1 (Divided differences)**.** *We call*

$$f[x_0] = f(x_0), \quad f[x_1] = f(x_1), \quad \cdots, \quad f[x_n] = f(x_n)$$

the **zeroth-order divided differences** *of $f(x)$. We call*

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

the **first order divided differences** *of $f(x)$; and for $k = 2, 3, \dots, n$, we call*

$$f[x_0, x_1, \cdots, x_k] = \frac{f[x_1, x_2, \cdots, x_k] - f[x_0, x_1, \cdots, x_{k-1}]}{x_k - x_0}$$

the **kth order divided differences** *of $f(x)$.*

Using these divided differences, we can now calculate the coefficients in (6.9). First, letting $x = x_0$ in (6.9), we get

$$c_0 = p(x_0) = f_0 = f[x_0].$$

taking $x = x_1$ in (6.9), we see

$$c_1 = \frac{p(x_1) - c_0}{x_1 - x_0} = \frac{p(x_1) - p(x_0)}{x_1 - x_0} = \frac{f[x_1] - f[x_0]}{x_1 - x_0} = f[x_0, x_1] \quad (= f[x_1, x_0]).$$

Similarly, taking $x = x_2$ in (6.9) we can derive

$$f(x_2) = p(x_2) = c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1)$$

and rearrange gives

$$c_2 = \left(\frac{f(x_2) - f(x_0)}{x_2 - x_0} - f[x_0, x_1] \right) / (x_2 - x_1) = \frac{f[x_0, x_2] - f[x_1, x_0]}{x_2 - x_1} = f[x_1, x_0, x_2].$$

We next show that $f[x_1, x_0, x_2] = f[x_0, x_1, x_2]$. In fact,

$$\begin{aligned}
f[x_1, x_0, x_2] &= \frac{f[x_0, x_2] - f[x_1, x_0]}{x_2 - x_1} \\
&= \frac{f(x_2) - f(x_0)}{(x_2 - x_0)(x_2 - x_1)} - \frac{f(x_1) - f(x_0)}{(x_1 - x_0)(x_2 - x_1)} \\
&= \frac{f(x_2) - f(x_1)}{(x_2 - x_0)(x_2 - x_1)} + \frac{f(x_1) - f(x_0)}{(x_2 - x_0)(x_2 - x_1)} - \frac{f(x_1) - f(x_0)}{(x_1 - x_0)(x_2 - x_1)} \\
&= \frac{f[x_1, x_2]}{x_2 - x_0} + \frac{f(x_1) - f(x_0)}{(x_2 - x_1)}\Big(\frac{1}{x_2 - x_0} - \frac{1}{x_1 - x_0}\Big) \\
&= \frac{f[x_1, x_2]}{x_2 - x_0} - \frac{f[x_0, x_1]}{x_2 - x_0} \\
&= f[x_0, x_1, x_2].
\end{aligned}$$

In general, we can deduce

$$\boxed{c_k = f[x_0, x_1, x_2, \cdots, x_k].}$$

Thus, Newton's interpolation of $f(x)$ can be written as

$$\begin{aligned}
N_n(x) = f[x_0] &+ f[x_0, x_1](x - x_0) + \cdots \\
&+ f[x_0, x_1, \cdots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1}).
\end{aligned}$$

The Newton's interpolation can be expressed in a simplified form when x_0, x_1, x_2, $\ldots$, x_n are equally spaced. In this case, let us introduce $h = x_{i+1} - x_i$ for each i and write $x = x_0 + s\,h$. Then we know $x - x_i = (s - i)h$, and the Newton form can be written as

$$\begin{aligned}
N_n(x) = N_n(x_0 + sh) &= f[x_0] + s\,h\,f[x_0, x_1] + s(s-1)h^2 f[x_0, x_1, x_2] + \cdots \\
&\quad + s(s-1)\cdots(s-n+1)h^n f[x_0, x_1, \cdots, x_n] \\
&= f[x_0] + \sum_{k=1}^{n} s(s-1)\cdots(s-k+1)h^k f[x_0, x_1, \cdots, x_k].
\end{aligned}$$

6.3.2 Derivatives and divided differences

We recall the following result, known as Rolle's theorem.

Theorem 6.1 (Rolle's theorem)**.** *Let $f : [a, b] \to \mathbb{R}$ be $n-1$-times continuously differentiable on $[a, b]$ and the nth derivative exists on (a, b), and there are n intervals given by*

$$a \le a_1 < b_1 \le a_2 < b_2 \le \cdots \le a_n < b_n \le b$$

such that $f(a_k) = f(b_k)$ for $k = 1, \ldots, n$, then there exists $\xi \in (a, b)$ such that $f^{(n)}(\xi) = 0$.

We begin with some observations:

1. If f is constant, its first order DD will vanish.
2. If f is a linear function, its first order DD is constant and its second order DD will vanish.
3. If f is a quadratic function, its first and second order DD are linear and constant, while its third order DD will vanish.

These observations tell us that divided difference acts very similar to derivatives.

Theorem 6.2. *Suppose $f \in C^n[a,b]$, and $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$ are distinct points in $[a,b]$. Then there exists some $\xi \in (a,b)$ such that*

$$f[x_0, x_1, x_2, \cdots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

Proof. Let $N_n(x)$ be the Newton's interpolation of $f(x)$ at the nodal points $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$, and set $g(x) = f(x) - N_n(x)$. Since $f(x_i) = N_n(x_i)$ for $i = 0, 1, \ldots, n$, g has $n+1$ distinct zeros in $[a,b]$. Then Rolle's theorem gives the existence of a point $\xi \in (a,b)$ such that $g^{(n)}(\xi) = 0$, which implies

$$N_n^{(n)}(\xi) = f^{(n)}(\xi).$$

But $N_n(x)$ is a polynomial of degree n with its leading coefficient $f[x_0, x_1, x_2, \cdots, x_n]$, so we have for any $x \in [a,b]$

$$N_n^{(n)}(x) = n! f[x_0, x_1, x_2, \cdots, x_n].$$

This completes the proof. □

6.3.3 Symmetry of DD

We first state an important result.

Theorem 6.3. *Let $p(x)$ and $q(x)$ be two polynomials of degree $\leq n$ such that*

$$p(x_i) = q(x_i) \text{ for } i = 0, 1, \ldots, n$$

at $n+1$ distinct points. Then, $p(x) = q(x)$ for any x.

Proof. Take the difference $r(x) = p(x) - q(x)$ which is a polynomial of degree $\leq n$. Then, r has $n+1$ roots at $x_0, \ldots, x_n$. By the fundamental theorem of algebra this is only possible if $r(x) = 0$ for any x. □

Remark 6.1. *A consequence of this is that the Lagrange's interpolation and Newton's interpolation of the function f at the same data points $(x_0, f_0), \ldots, (x_n, f_n)$ actually are the same polynomial, but with seemingly different expressions.*

In this section, we show that the divided differences are symmetric, in the sense that any permutation of its arguments does not alter its value. Clearly, $f[x_0]$ is symmetric. For the 1st order DD, we have

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

so the first order DD are symmetric.

To see the symmetry of the second order DD, namely

$$f[x_0, x_1, x_2] = f[x_1, x_0, x_2],$$

we recall that the interpolation of $f(x)$ at the point x_0, x_1, x_2 is given by

$$N_1(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1).$$

Now, if we consider the interpolation of $f(x)$ at the points x_1, x_0, x_2 (just swap the position of x_0 with x_1), we have

$$\tilde{N}_1(x) = f[x_1] + f[x_1, x_0](x - x_1) + f[x_1, x_0, x_2](x - x_1)(x - x_0).$$

Note that

$$\begin{aligned} f[x_0] + f[x_0, x_1](x - x_0) &= f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_0) = f_0 + \frac{f_1 - f_0}{x_1 - x_0}(x - x_1 + x_1 - x_0) \\ &= f_1 + \frac{f_0 - f_1}{x_0 - x_1}(x - x_1) = f[x_1] + f[x_1, x_0](x - x_1), \end{aligned}$$

and so by Theorem 6.3, **uniqueness** of the interpolation gives $N_1(x) = \tilde{N}_1(x)$, which means that

$$f[x_0, x_1, x_2](x - x_0)(x - x_1) = f[x_1, x_0, x_2](x - x_0)(x - x_1),$$

and so

$$f[x_1, x_0, x_2] = f[x_0, x_1, x_2].$$

More generally, we have the following

Theorem 6.4. *Let $f : [a, b] \to \mathbb{R}$ and consider a set of distinct nodal points $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$. Then the divided difference is a symmetric function of its arguments. Namely if $z_0, z_1, \ldots, z_n$ is a permutation of $x_0, x_1, \ldots, x_n$, then*

$$f[z_0, z_1, z_2, \cdots, z_n] = f[x_0, x_1, x_2, \cdots, x_n].$$

Proof. It is easy to see that the divided difference $f[z_0, z_1, z_2, \cdots, z_n]$ is the coefficient of x^n in the polynomial of degree n that interpolates f at the points $z_0, z_1, \ldots, z_n$, while the divided difference $f[x_0, x_1, x_2, \cdots, x_n]$ is the coefficient of x^n in the polynomial of degree n that interpolates f at the points $x_0, x_1, \ldots, x_n$. Since these two polynomials of degree n agree at $n + 1$ points, these two polynomials must be the same. $\square$

6.3.4 Computing the coefficients

Given a set of observation data:

x_0	x_1	x_2	$\cdots$	x_n
f_0	f_1	f_2	$\cdots$	f_n

how do we compute the coefficients in Newton's interpolation? We take the following simple case as an example:

Table for computing divided difference $f[x_0, x_1, x_2, x_3]$

x_0	$f[x_0] = f_0$			
		$f[x_0, x_1]$		
x_1	$f[x_1] = f_1$		$f[x_0, x_1, x_2]$	
		$f[x_1, x_2]$		$f[x_0, x_1, x_2, x_3]$
x_2	$f[x_2] = f_2$		$f[x_1, x_2, x_3]$	
		$f[x_2, x_3]$		
x_3	$f[x_3] = f_3$			

Example 6.2. *Compute the Newton form of interpolation satisfying the following conditions:*

x	3	1	5	6
$f(x)$	1	-3	2	4

We can compute as follows:

$x_0 = 3$	$f[x_0] = 1$			
		$f[x_0, x_1] = 2$		
$x_1 = 1$	$f[x_1] = -3$		$f[x_0, x_1, x_2] = -\frac{3}{8}$	
		$f[x_1, x_2] = \frac{5}{4}$		$f[x_0, x_1, x_2, x_3] = \frac{7}{40}$
$x_2 = 5$	$f[x_2] = 2$		$f[x_1, x_2, x_3] = \frac{3}{20}$	
		$f[x_2, x_3] = 2$		
$x_3 = 6$	$f[x_3] = 4$			

Thus the Newton's interpolation can be written as follows:

$$\begin{aligned} N(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_0) \\ &\quad + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\ &= 1 + 2(x - 3) - \frac{3}{8}(x - 3)(x - 1) + \frac{7}{40}(x - 3)(x - 1)(x - 5). \end{aligned} \tag{6.10}$$

Now if we add one more data point, we have the data table:

x	3	1	5	6	0
$f(x)$	1	-3	2	4	5

The calculation needs to be slightly changed as follows:

$x_0 = 3$	$f[x_0] = 1$				
		$f[x_0, x_1] = 2$			
$x_1 = 1$	$f[x_1] = -3$		$f[x_0, x_1, x_2] = -\frac{3}{8}$		
		$f[x_1, x_2] = \frac{5}{4}$		$f[x_0, x_1, x_2, x_3] = \frac{7}{40}$	
$x_2 = 5$	$f[x_2] = 2$		$f[x_1, x_2, x_3] = \frac{3}{20}$		$f[x_0, x_1, x_2, x_3, x_4] = \frac{11}{72}$
		$f[x_2, x_3] = 2$		$f[x_1, x_2, x_3, x_4] = -\frac{17}{60}$	
$x_3 = 6$	$f[x_3] = 4$		$f[x_2, x_3, x_4] = \frac{13}{30}$		
		$f[x_3, x_4] = -\frac{1}{6}$			
$x_4 = 0$	$f[x_4] = 5$				

This immediately gives the Newton's interpolation, adding only one term to the polynomial (6.10)*:*

$$\begin{aligned}
p(x) &= f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_0) \\
&\quad + f[x_0, x_1, x_2, x_3](x - x_0)(x - x_1)(x - x_2) \\
&\quad + f[x_0, x_1, x_2, x_3, x_4](x - x_0)(x - x_1)(x - x_2)(x - x_3) \\
&= 1 + 2(x - 3) - \frac{3}{8}(x - 3)(x - 1) + \frac{7}{40}(x - 3)(x - 1)(x - 5) \\
&\quad + \frac{11}{72}(x - 3)(x - 1)(x - 5)(x - 6).
\end{aligned}$$

6.3.5 Cost of constructing the polynomial

Suppose we are already given $(n + 1)$ data points to interpolate:

x	x_0	x_1	x_2	$\cdots$	x_n
$f(x)$	f_0	f_1	f_2	$\cdots$	f_n

The nth degree Newton's interpolation is given by

$$N_n(x) = c_0 + c_1(x - x_0) + \ldots + c_n(x - x_0)(x - x_1) \cdots (x - x_{n-1}). \tag{6.11}$$

The question is how costly it is to compute c_i. Recall that the c_i's are the first entries in each column of the divided difference table.

Let us then compute the cost of obtaining **all** the entries in the divided difference table. Recall that the table looks something like this:

x	0th	1st		$(n-1)$th	nth
x_0	$f[x_0]$				
		$f[x_0, x_1]$			
x_1	$f[x_1]$		$\ddots$		
		$f[x_1, x_2]$		$f[x_0, \ldots, x_{n-1}]$	
$\vdots$	$\vdots$	$\vdots$			$f[x_0, \ldots, x_n]$
		$f[x_{n-2}, x_{n-1}]$		$f[x_1, \ldots, x_n]$	
x_{n-1}	$f[x_{n-1}]$		$\vdots$		
		$f[x_{n-1}, x_n]$			
x_n	$f[x_n]$				

For the 0th level column, we don't have to do anything. We just have $f[x_i] = f_i$. In the 1st level column, each divided difference is computed in the form:

$$\frac{f[x_i] - f[x_j]}{x_i - x_j}$$

which require 2 subtractions and 1 multiplication. Notice that there are only n 1st level divided differences to compute in this column.

Next for the 2nd level divided differences. Again, each divided difference can be computed in 2 subtractions and 1 multiplication, but there are only $(n-1)$ 2nd level divided differences to compute. Repeating this argument, finally, for the nth level divided difference, there is only one to compute (namely $c_n = f[x_0, \ldots, x_n]$); and it can be computed in 2 subtractions and 1 multiplication. Thus to compute all the entries in the divided difference table (in particular those c_i's), it requires

$$3 \cdot [n + (n-1) + \ldots + 1] \sim O(n^2)$$

operations.

6.3.6 Cost of evaluating the polynomial

Suppose we have already obtained all the c_i in (6.11). Our next question is how costly it will be to compute $N_n(t)$ at arbitrary point t. For this, we use the expression

$$\begin{aligned} N_n(t) &= c_0 + c_1(t - x_0) + \cdots + c_n(t - x_0)(t - x_1)\cdots(t - x_{n-1}) \\ &= c_0 + (t - x_0)(c_1 + (t - x_1)(c_2 + (t - x_2)(c_3 + (\cdots \\ &\quad + (c_{n-2} + (t - x_{n-2})(c_{n-1} + c_n(t - x_{n-1}))\cdots))))) \end{aligned}$$

This can now be computed in $2n$ additions and n multiplications. Thus for any t, $p_n(t)$ can be computed in $O(n)$ operations.

6.3.7 Cost of updating the polynomial

Suppose we are given one more data point to interpolate:

x	x_0	x_1	x_2	$\cdots$	x_n	x_{n+1}
$f(x)$	f_0	f_1	f_2	$\cdots$	f_n	f_{n+1}

The $(n+1)$th degree Newton's interpolation is given by

$$\begin{aligned} N_{n+1}(x) &= c_0 + c_1(x - x_0) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \\ &\quad + c_{n+1}(x - x_0)(x - x_1)\cdots(x - x_{n-1})(x - x_n). \end{aligned}$$

Thus we only have to compute c_{n+1}.

To compute c_{n+1}, we have to add one entry at the bottom of each column in the divided difference table. Since each entry in the table are of the form $(a - b)/(c - d)$, each of them can be computed in 2 subtractions and 1 multiplications. Moreover, since there are $(n+1)$ columns in the table to be computed (recalling we only need to compute the 1st DD up to the $(n+1)$th DD), we only have to compute $(n+1)$ entries. We therefore can conclude that c_{n+1} can be computed in $2(n+1)$ subtractions and $(n+1)$ multiplications. Thus the update can be done in $O(n)$ operations.

6.3.8 Summary of the interpolation methods

From the previous discussions, we can briefly summarize the computational costs of three interpolation methods in the following table:

	finding the coefficients	adding one more data point
Vandermonde	solve $Ax = b$ $O(n^3)$, ill-conditioned	solve completely new system $O(n^3)$, more ill-conditioned
Lagrange	compute $\prod_{i\neq j}(x_i - x_j)$ $O(n^2)$	compute $\prod_{i=0}^{n}(x_i - x_{n+1})$ $O(n)$
Newton	table calculation $O(n^2)$	add one more row to the table $O(n)$

6.4 Error estimates of polynomial interpolations

Given a set of $n+1$ observation data at $n+1$ distinct points

x	x_0	x_1	x_2	$\cdots$	x_n
$f(x)$	f_0	f_1	f_2	$\cdots$	f_n

we know there exists a unique polynomial $p(x)$ of degree $\leq n$ satisfying the conditions

$$p(x_i) = f_i, \quad i = 0, 1, 2, \ldots, n.$$

Then, we have the following error estimate.

Theorem 6.5. *Suppose $f \in C^{n+1}[a,b]$, and $p(x)$ is the polynomial interpolation of $f(x)$ at the $n+1$ distinct points $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$. Then, for any $x \in [a,b]$, there exists a point $\xi_x \in (a,b)$ such that*

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n). \tag{6.12}$$

Proof. If $x = x_i$ for some $i \in \{0, 1, \cdots, n\}$, then the result holds obviously. Now, we consider $x \in (a,b)$ but $x \notin \{x_0, x_1, \cdots, x_n\}$ and introduce

$$\phi(t) = f(t) - p(t) - \lambda(t - x_0)(t - x_1)\cdots(t - x_n).$$

For the fixed x, choose $\lambda = \lambda(x)$ such that $\phi(x) = 0$, then we have

$$\phi(x) = 0, \ \phi(x_0) = \phi(x_1) = \cdots = \phi(x_n) = 0.$$

By the Rolle's theorem, we know

$$\phi'(t) \text{ has at least } n+1 \text{ distinct zeros,}$$

with each zero lying in between any pair in $\{x_0, x_1, \ldots, x_n, x\}$. Similarly,

$$\phi''(t) \text{ has at least } n \text{ distinct zeros.}$$

Continue this way, and we know

$$\phi^{(n+1)}(t) \text{ has at least one zero,}$$

say $\xi_x \in (a,b)$, i.e.,

$$\phi^{(n+1)}(\xi_x) = 0.$$

But notice that

$$\phi^{(n+1)}(t) = f^{(n+1)}(t) - p^{(n+1)}(t) - \lambda(n+1)!,$$

and especially

$$\phi^{(n+1)}(\xi_x) = f^{(n+1)}(\xi_x) - \lambda(n+1)! = 0.$$

This gives

$$\lambda = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}.$$

Noting λ was chosen such that $\phi(x) = 0$, we obtain

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

□

Example 6.3. *Consider the function $f(x) = \sin(4\pi x)$ approximated by a polynomial of degree 9 that interpolates f at 10 points in the interval $[0,1]$. We can apply the error estimate (6.12) with $|f^{(10)}(\xi_x)| \le (4\pi)^{10}$ and $|x - x_i| \le 1$. So for all $x \in [0,1]$,*

$$|\sin(4\pi x) - p(x)| \le \frac{(4\pi)^{10}}{10!}.$$

6.5 Chebyshev polynomials

Using the interpolation error estimate (6.12), we can estimate the accuracy of each interpolation polynomial when all the interpolating nodes

$$a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$$

are given. But with a different set of interpolating nodes, the resulting polynomial **will have a different accuracy**. This raises a natural question: can we choose the interpolating nodes so that the resulting polynomial reaches an **optimal accuracy**? In this section, we shall discuss how to find such optimal polynomial. It is easy to see from the error estimate (6.12) that the optimal polynomial can be realized if we can choose a set of interpolating nodes x_0, x_1, ..., x_n such that the resulting polynomial $(x-x_0)(x-x_1)\cdots(x-x_n)$ on the right-hand side of (6.12) is minimized among all polynomials in magnitude. An analysis of this optimization problem was first made by the mathematician Chebyshev. The process leads naturally to a system of polynomials called **Chebyshev polynomials**, which we will introduce below.

Definition 6.2 (Chebyshev polynomials)**.** *The Chebyshev polynomials are defined recursively as follows:*

$$T_0(x) = 1, \quad T_1(x) = x,$$

and for $n \geq 1$,

$$T_{n+1}(x) = 2x\,T_n(x) - T_{n-1}(x).$$

One obtains from the definition

$$\begin{aligned}
T_2(x) &= 2x^2 - 1, \\
T_3(x) &= 4x^3 - 3x, \\
T_4(x) &= 8x^4 - 8x^2 + 1, \\
T_5(x) &= 16x^5 - 20x^3 + 5x, \\
T_6(x) &= 32x^6 - 48x^4 + 18x^2 - 1,
\end{aligned}$$

and it is easy to see that the leading coefficient of $T_k(x)$ is 2^{k-1}.

Theorem 6.6. *For x in the interval $[-1, 1]$, the Chebyshev polynomials have the following closed forms for $n \geq 0$,*

$$T_n(x) = \cos(n \cos^{-1} x) \text{ for } -1 \leq x \leq 1.$$

Proof. Recall the formula

$$\cos(A + B) = \cos A \cos B - \sin A \sin B.$$

So we obtain

$$\begin{aligned}
\cos[(n+1)\theta] &= \cos\theta \cos(n\theta) - \sin\theta \sin(n\theta), \\
\cos[(n-1)\theta] &= \cos\theta \cos(n\theta) + \sin\theta \sin(n\theta),
\end{aligned}$$

adding them up, we have

$$\cos[(n+1)\theta] = 2\cos\theta \cos(n\theta) - \cos[(n-1)\theta].$$

Now let $\theta = \cos^{-1} x$ so that $x = \cos\theta$. We see from the above relation that the function

$$f_n(x) = \cos(n \cos^{-1} x)$$

satisfies the following system:

$$f_0(x) = 1, \quad f_1(x) = x,$$

and for $n \geq 1$,

$$f_{n+1}(x) = 2x\,f_n(x) - f_{n-1}(x).$$

So we have $f_n = T_n$ for all n. □

Immediately from the above closed form, we infer that

- $|T_n(x)| \leq 1$ for $-1 \leq x \leq 1$,
- $T_n\Big(\cos\frac{j\pi}{n}\Big) = (-1)^j$ for $0 \leq j \leq n$,
- $T_n\Big(\cos\frac{2j-1}{2n}\pi\Big) = 0$ for $0 \leq j \leq n$.

Definition 6.3 (Monic polynomial). *A polynomial $p(x) = a_n x^n + \cdots + a_1 x + a_0$ is called* **monic** *if $a_n = 1$, i.e., the coefficient of the term of highest order is one.*

We see that for $T_n(x)$ the term of highest order has a coefficient of 2^{n-1} for $n > 0$. Therefore $\widehat{T}_n(x) = 2^{1-n}T_n(x)$ is a monic polynomial for all $n \geq 1$. The following result shows that monic Chebyshev polynomials are optimal in some sense.

Lemma 6.1. *If p is a monic polynomial of degree k, then*

$$\|p\|_\infty = \max_{-1\leq x\leq 1} |p(x)| \geq 2^{1-k}. \tag{6.13}$$

Equality is achieve if and only if $p = \widehat{T}_k$.

Proof. For (6.13) we prove by contradiction. Suppose that

$$|p(x)| < 2^{1-k} \quad \text{for } |x| \leq 1.$$

Let $x_j = \cos(j\pi/k)$, and consider the difference $q(x) = \widehat{T}_k(x) - p(x)$, noting that as both p and $\widehat{T}_k$ are monic polynomials of degree k, the difference $q(x) = \widehat{T}_k(x) - p(x)$ must have a degree at most $k-1$. Then

$$(-1)^j p(x_j) \leq |p(x_j)| < 2^{1-k} = (-1)^j \widehat{T}_k(x_j).$$

Consequently,

$$(-1)^j[\widehat{T}_k(x_j) - p(x_j)] > 0 \quad \text{for } 0 \leq j \leq k.$$

This shows that the polynomial $q(x) = \widehat{T}_k(x) - p(x)$ oscillates in sign $k+1$ times on the interval $[-1,1]$. Therefore it must have at least k roots in $(-1,1)$. But this is contradicts that condition that $q(x)$ has degree at most $k-1$.

Finally, since $\widehat{T}_k(x) = 2^{1-k}T_k(x)$ and $\max_{x\in[-1,1]}|T_k(x)| = 1$, we obtain $\|\widehat{T}_k\|_\infty = 2^{1-k}$. $\square$

We now return to the problem of choosing a set of interpolating nodes $x_0, x_1, \ldots, x_n$, such that the right-hand side (6.12) is minimized among all polynomials in magnitude. Using the interpolation error estimate (6.12), we can deduce that

$$\max_{|x|\leq 1}|f(x) - p(x)| \leq \frac{1}{(n+1)!}\max_{|x|\leq 1}|f^{(n+1)}(x)| \max_{|x|\leq 1}|(x-x_0)(x-x_1)\cdots(x-x_n)|.$$

But by the property (6.13) of monic functions, we know

$$\max_{|x|\le 1} |(x-x_0)(x-x_1)\cdots(x-x_n)| \ge 2^{-n} = \|\widehat{T}_{n+1}(x)\|_\infty.$$

Thus, if we choose the nodes as the roots of T_{n+1}, namely

$$x_i = \cos\left(\frac{2i-1}{2n+2}\pi\right), \quad i = 1, 2, \ldots, n+1, \tag{6.14}$$

so that

$$\widehat{T}_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n),$$

then we have the following result.

Theorem 6.7. *If we choose the $(n+1)$ interpolating nodes $x_0, x_1, \ldots, x_n$ to be the roots of the Chebyshev polynomial T_{n+1}, i.e., (6.14), then the error of the resulting interpolating polynomial for a given function $f(x)$ is minimized and can be estimated by*

$$|f(x)-p(x)| \le \frac{1}{2^n(n+1)!}\max_{|t|\le 1}|f^{n+1}(t)| \text{ for any } x \in [-1,1].$$

For arbitrary interval $[a,b]$ with $a \ne -1$ and $b \ne 1$, we can use a linear transformation $l : [-1,1] \to [a,b]$, $l(x) = \frac{1}{2}(b-a) + \frac{1}{2}(b+a)x$ and compute the Chebyshev points $\{x_k\}_{k=0}^n$ in the interval $[a,b]$.

6.6 Hermite's interpolation

Given the following set of observation data regarding f and its derivative f' at points $x_0, x_1, \ldots, x_n$ with $x_i \ne x_j$ for $i \ne j$,

x	x_0	x_1	x_2	$\cdots$	x_n
$f(x)$	f_0	f_1	f_2	$\cdots$	f_n
$f'(x)$	f'_0	f'_1	f'_2	$\cdots$	f'_n

we would like to see if it is possible to determine a polynomial $p(x)$ of degree $\le 2n+1$ such that

$$p(x_i) = f_i \quad \text{and} \quad p'(x_i) = f'_i \ \text{ for } i = 0, 1, \ldots, n. \tag{6.15}$$

Below we describe two methods to construct such a polynomial. The first is based on the Lagrange interpolation, and the second is based on the Newton interpolation. It turns out that both methods are equivalent[12].

[12]See page 56 of M.J.D. Powell, Approximation Theory and Methods, Cambridge University Press, 1981

6.6.1 Lagrange form

At the heart of the Lagrange interpolation method we find special polynomials $u_i(x)$ such that

$$u_i(x_j) = \delta_{ij}$$

and so when interpolating data points $(x_i, f_i)_{0\le i\le n}$ we can use the polynomial

$$p(x) = \sum_{i=0}^{n} f_i u_i(x).$$

To deal with the additional data points from the derivatives, we can try

$$p(x) = \sum_{i=0}^{n} f_i u_i(x) + \sum_{i=0}^{n} f_i' v_i(x),$$

where

$$\begin{cases} u_i(x_j) = \delta_{ij}, \\ u_i'(x_j) = 0, \end{cases} \qquad \begin{cases} v_i(x_j) = 0, \\ v_i'(x_j) = \delta_{ij}. \end{cases}$$

Let $l_0(x), l_1(x), \dots, l_n(x)$ be the Lagrange basic functions in (6.6) associated with the set of nodal points $x_0, x_1, \dots, x_n$. Then, observe that

$$l_i^2(x_j) = \delta_{ij}, \text{ but } (l_i^2)'(x_j) \neq 0.$$

Therefore, we try

$$u_i = (a_i x + b_i) l_i^2(x).$$

Plugging in $x = x_j$ yields $u_i(x_j) = 0$ if $i \neq j$. Meanwhile if $x = x_i$ we have

$$u_i(x_i) = a_i x_i + b_i = 1. \tag{6.16}$$

Taking the derivative gives

$$u_i'(x) = a_i l_i^2(x) + 2(a_i x + b_i) l_i(x) l_i'(x).$$

For $x = x_j$ using $l_i(x_j) = 0$ if $i \neq j$ gives $u_i'(x_j) = 0$. Meanwhile if $x = x_i$ we require

$$u_i'(x_i) = a_i + 2(a_i x_i + b_i) l_i'(x_i) = 0. \tag{6.17}$$

Solving the two equations (6.16) and (6.17) leads to

$$b_i = 1 - a_i x_i, \quad a_i + 2l_i'(x_i) = 0, \quad \Rightarrow \quad a_i = -2l_i'(x_i), \quad b_i = 1 + 2x_i l_i'(x_i).$$

Hence, we have

$$u_i(x) = \Big(1 - 2l_i'(x_i)(x - x_i)\Big) l_i^2(x).$$

In a similar way, we can derive

$$v_i(x) = (x - x_i) l_i^2(x).$$

One can easily show that

1. $u_i(x)$ and $v_i(x)$ are all polynomials of degree $2n+1$;
2. $u_i(x_j) = \delta_{ij}$, and $v_i(x_j) = 0$ for any i, j;
3. $u_i'(x_j) = 0$, and $v_i'(x_j) = \delta_{ij}$ for any i, j.

Using these results, we can directly verify that the following polynomial

$$\boxed{H_{2n+1}(x) = H(x) = \sum_{i=0}^{n} f_i\, u_i(x) + \sum_{i=0}^{n} f_i'\, v_i(x)} \tag{6.18}$$

is a polynomial of degree $\leq 2n+1$ such that all the conditions in (6.15) are satisfied. This polynomial is called the **Hermite's interpolation**, and satisfies the following error estimate.

Theorem 6.8. *Suppose $f \in C^{2n+2}[a,b]$, and $H(x)$ is its Hermite's interpolation at the $n+1$ distinct points: $a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b$ such that all the conditions in* (6.15) *are satisfied. Then the following error estimate holds for some $\xi \in (a,b)$:*

$$f(x) - H(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}(x-x_0)^2(x-x_1)^2\cdots(x-x_n)^2. \tag{6.19}$$

Proof. To prove (6.19), we fix a point $x \in (a,b)$. If x is a node, i.e., $x = x_i$ for some $i = 0, \ldots, n$, the result (6.19) holds clearly. So we assume that x is not a node. Let

$$w(t) = f(t) - H(t) - \alpha \underbrace{(t-x_0)^2(t-x_1)^2\cdots(t-x_n)^2}_{=:\phi(t)}$$

where α is a constant such that $w(x) = 0$. The theorem is proved once we identify the constant α. One can easily see that $w(t)$ has at least $n+2$ zeros in (a,b): $x, x_0, x_1, \ldots, x_n$. By Rolle's theorem, $w'(t)$ has at least $n+1$ zeros in (a,b). But $w'(x)$ also vanishes at all the nodal points by construction of the Hermite's interpolation. So $w'(t)$ has at least $2n+2$ zeros in (a,b). Recursively using the Rolle's theorem, we know that $w^{(2n+2)}(t)$ has some zero $\xi \in (a,b)$, that is,

$$0 = w^{(2n+2)}(\xi) = f^{(2n+2)}(\xi) - H^{(2n+2)}(\xi) - \alpha\phi_n^{(2n+2)}(\xi)$$

Since H is a polynomial of degree $\leq 2n+1$, $H^{(2n+2)} = 0$, and since ϕ has the leading term t^{2n+2}, it follows that $\phi^{(2n+2)} = (2n+2)!$. Then, using these information we obtain

$$0 = f^{(2n+2)}(\xi) - \alpha(2n+2)!.$$

This proves the desired error estimate (6.19). □

6.6.2 Newton form

While (6.18) is a clear expression depending on the Lagrange polynomials l_i and their derivatives, it can be tedious to compute them even for small values of n. We now turn to an alternative method using Newton's interpolation with divided difference. The question is how to build a divided difference table from (6.6) now that we have an extra row of data. Consider a new sequence $z_0, \dots, z_{2n+1}$ by

$$z_{2i} = z_{2i+1} = x_i \quad \text{for } i = 0, \dots, n.$$

We then construct the divided difference table in a form that uses $z_0, \dots, z_{2n+1}$. Note that since $z_{2i} = z_{2i+1} = x_i$ for each i, we cannot define $f[z_{2i}, z_{2i+1}]$ by the divided difference formula. However, we assume, based on the divided difference formula, that a reasonable substitution in this situation is $f[z_{2i}, z_{2i+1}] = f'(z_{2i}) = f'(x_i)$. Hence, we use the entries $f'(x_0), f'(x_1), \dots, f'(x_n)$ in place of $f[z_0, z_1], f[z_2, z_3], \dots, f[z_{2n}, z_{2n+1}]$, while the remaining divided differences are produced as usual. For example, the following table shows the entries for the first three divided difference columns for three points x_0, x_1, x_2:

x	0th	1st	2nd	3rd
$z_0 = x_0$	$f[z_0] = f(x_0)$			
		$f[z_0, z_1] = f'(x_0)$		
$z_1 = x_0$	$f[z_1] = f(x_0)$		$f[z_0, z_1, z_2]$	
		$f[z_1, z_2] = f[x_0, x_1]$		$f[z_0, z_1, z_2, z_3]$
$z_2 = x_1$	$f[z_2] = f(x_1)$		$f[z_1, z_2, z_3]$	
		$f[z_2, z_3] = f'(x_1)$		$f[z_1, z_2, z_3, z_4]$
$z_3 = x_1$	$f[z_3] = f(x_1)$		$f[z_2, z_3, z_4]$	
		$f[z_3, z_4] = f[x_1, x_2]$		$f[z_2, z_3, z_4, z_5]$
$z_4 = x_2$	$f[z_4] = f(x_2)$		$f[z_3, z_4, z_5]$	
		$f[z_4, z_5] = f'(x_2)$		
$z_5 = x_2$	$f[z_5] = f(x_2)$			

We see that the rest of the table is generated in the same manner as for the Newton's divided difference table. Then, the Hermite's interpolation in divided difference form is

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

Example 6.4. *Consider the data*

x	x_0	x_1
$f(x)$	1	2
$f'(x)$	0	1

The table of divided differences looks like

x	*0th*	*1st*	*2nd*	*3rd*
$z_0 = 0$	$f[z_0] = 1$			
		$f[z_0, z_1] = f'(x_0) = 0$		
$z_1 = 0$	$f[z_1] = 1$		$f[z_0, z_1, z_2] = 1$	
		$f[z_1, z_2] = f[x_0, x_1] = 1$		$f[z_0, z_1, z_2, z_3] = -1$
$z_2 = 1$	$f[z_2] = 2$		$f[z_1, z_2, z_3] = 0$	
		$f[z_2, z_3] = f'(x_1) = 1$		
$z_3 = 1$	$f[z_3] = 2$			

and we obtain

$$H_3(x) = 1 + x^2 - x^2(x-1) = 1 - x^3 + 2x^2.$$

Exercise. For the data

x	$x_0 = -1$	$x_1 = 0$	$x_2 = 1$
$f(x)$	2	1	2
$f'(x)$	-4	0	4

Show that Hermit's interpolation gives

$$H_5(x) = 1 + x^4.$$

7 Numerical integration

Approximations of integrals are widely encountered in real applications. Many important physical quantities are represented by the integrals, e.g., mass, concentrations, heat flux, heat sources and so on. Furthermore, many partial differential equations are solved using a so-called weak formulation that involves integrals.

In this section we shall discuss how to approximate a given integral on an interval. The approximation of integrals in higher dimensions can be reduced to the integrals on many intervals. Given a function $f(x)$ on a interval $[a, b]$, we discuss how to approximate the integral

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

based on **quadrature rules**.

7.1 Simple rules

Recall that the integral

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

is the area enclosed by the curve $y = f(x)$, and the lines $x = a$, $x = b$. When a is close to b, one may approximate the area by the area of some simple geometric domains. If we use the area of the rectangle with the base $[a, b]$ and height $f(a)$ or $f(b)$, then we get the **Rectangular quadrature rules**:

$$\int_a^b f(x)\,dx \approx (b-a)\,f(a) \text{ or } \int_a^b f(x)dx \approx (b-a)\,f(b). \tag{7.1}$$

A more accurate rule is to approximate the integral by the area of the trapezoid formed by the base $[a, b]$, the lines $x = a$, $x = b$ and the line connecting $(a, f(a))$ to $(b, f(b))$. This leads immediately to the **Trapezoidal rule**:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}(f(a) + f(b)). \tag{7.2}$$

To see how good are these approximations, we compute an error estimate. For the rectangular rule, suppose $f : [a, b] \to \mathbb{R}$ satisfies $\max_{x\in[a,b]} |f'(x)| \le K$ for some $K > 0$. Then,

$$\begin{aligned}\left|\int_a^b f(x)\,dx - (b-a)f(a)\right| &= \left|\int_a^b f(x) - f(a)\,dx\right| = \left|\int_a^b f'(\xi)(x-a)\,dx\right| \\ &\le K\int_a^b x - a\,dx = \frac{K}{2}(b-a)^2,\end{aligned}$$

and analogous error estimate also holds if we use the other rectangular rule $(b-a)f(b)$ instead. Notice that if $f(x)$ is a constant function, then the rectangular rule is **exact**, in the sense that

$$\int_a^b f(x)\,dx = (b-a)f(a) \text{ for all constant functions } f:[a,b]\to\mathbb{R}.$$

For the trapezoidal rule, we first consider the Lagrange interpolation of $f(x)$ at two points $x_0 = a$ and $x_1 = b$:

$$L(x) = \frac{x-b}{a-b}f(a) + \frac{x-a}{b-a}f(b).$$

Then we have

$$\begin{aligned}\int_a^b L(x)\,dx &= \int_a^b \Big(\frac{x-b}{a-b}f(a) + \frac{x-a}{b-a}f(b)\Big)\,dx \\ &= \frac{f(a)}{a-b}\cdot\Big(-\frac{1}{2}\Big)(b-a)^2 + \frac{f(b)}{b-a}\cdot\frac{1}{2}(b-a)^2 \\ &= \frac{b-a}{2}(f(a)+f(b)).\end{aligned}$$

This is exactly the same as the trapezoidal rule. So the error of the trapezoidal rule can be transfered to the error of the Lagrange interpolation:

$$\begin{aligned}\int_a^b f(x)\,dx - \frac{b-a}{2}(f(a)+f(b)) &= \int_a^b (f(x)-L(x))\,dx \\ &= \frac{1}{2}\int_a^b f''(\xi_x)(x-a)(x-b)\,dx,\end{aligned}$$

where we used (6.12) for the last equality. We see that for any linear polynomial, the trapezoidal rule is exact. Furthermore, if $f:[a,b]\to\mathbb{R}$ satisfies $\max_{x\in[a,b]}|f''(x)| \le K$, then we obtain

$$\begin{aligned}\left|\int_a^b f(x)\,dx - \frac{b-a}{2}(f(a)+f(b))\right| &\le \frac{K}{2}\left|\int_a^b (x-a)(x-a+a-b)\,dx\right| \\ &\le \frac{K}{12}(b-a)^3.\end{aligned} \tag{7.3}$$

7.2 Composite rules

In the error estimates for the rectangular rule and the trapezoidal rule, if the size of the interval $[a,b]$ is not small, then the upper bound on the error is not useful. To derive a more accurate approximation, we can divide $[a,b]$ into n equally-spaced subintervals using the points

$$a = x_0 < x_1 < x_2 < \cdots < x_{n-1} < x_n = b.$$

Let $h = \frac{b-a}{n}$ be the length of each subinterval (or the mesh size), then we have

$$x_i = a + ih, \quad i = 0, 1, \ldots, n.$$

Now on each subinterval $[x_{i-1}, x_i]$, we can approximate $\int_{x_{i-1}}^{x_i} f(x)dx$ by the trapezoidal rule with good accuracy, i.e.,

$$\int_{x_{i-1}}^{x_i} f(x)\,dx \approx \frac{h}{2}(f(x_{i-1}) + f(x_i)).$$

Then summing gives

$$\int_a^b f(x)\,dx = \sum_{i=1}^n \int_{x_{i-1}}^{x_i} f(x)\,dx \approx \sum_{i=1}^n \frac{h}{2}(f(x_{i-1}) + f(x_i)),$$

This leads to the **composite trapezoidal rule**, which is obtained by applying an integration formula for a single interval to each subinterval of a partitioned interval:

$$\int_a^b f(x)\,dx \approx h\left(\frac{f(x_0)}{2} + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + \frac{f(x_n)}{2}\right). \tag{7.4}$$

To obtain an error estimate, we suppose $f \in C^2[a,b]$ with $\max_{x\in[a,b]} |f''(x)| \le K$, then

$$\begin{aligned}
&\left|\int_a^b f(x)\,dx - \sum_{i=1}^n \frac{h}{2}(f(x_{i-1}) + f(x_i))\right| \\
&= \left|\sum_{i=1}^n \int_{x_{i-1}}^{x_i} f(x)\,dx - \frac{h}{2}(f(x_{i-1}) + f(x_i))\right| = \left|\sum_{i=1}^n \int_{x_{i-1}}^{x_i} \frac{f''(\xi_i)}{2}(x - x_{i-1})(x - x_i)\,dx\right| \\
&\le \sum_{i=1}^n \frac{K}{2}\left|\int_{x_{i-1}}^{x_i} (x - x_{i-1})(x - x_{i-1} + x_{i-1} - x_i)\,dx\right| = \sum_{i=1}^n \frac{K}{12}(x_i - x_{i-1})^3 \\
&= \frac{nh^3K}{12} = \frac{K}{12}(b-a)h^2,
\end{aligned}$$

where $nh = (b-a)$. This shows that if we take smaller and smaller mesh sizes h, the error for the composite trapezoidal rule will tend to zero, but the cost is we have to increase the number of function evaluations.

Example 7.1. *Determine the mesh size h so that the error of the composite trapezoidal rule for computing the integral $\int_0^1 \sin(\pi x)\,dx$ is not bigger than 10^{-6}.*

Applying the error estimate

$$\left|\int_0^1 sin(\pi x)\,dx - \sum_{i=1}^n \frac{h}{2}(\sin(\pi x_{i-1}) + \sin(\pi x_i))\right| \le \frac{\pi^2}{12}h^2 \le 10^{-6}$$

implies that h must be not bigger than $\frac{2\sqrt{3}}{\pi}10^{-3}$. Hence, roughly 1,000 subregions and 1,000 function evaluations are needed.

7.3 Newton–Cotes quadrature rule

The composite trapezoidal rule is exact for linear polynomials, and we are interested in deriving quadrature rules that are exact for higher order polynomials. When defined on an equally-spaced set of points, these leads to the **Newton–Cotes quadrature rules**.

For an interval $[a,b]$, let

$$a = x_0 < x_1 < \cdots < x_n = b, \quad x_i = a + ih, \quad h = \frac{b-a}{n},$$

denote an equally-spaced partition. We want to find constants $\alpha_0, \alpha_1, \ldots, \alpha_n$ such that for any polynomial $p(x)$ of degree $\leq n$, we have

$$\int_a^b p(x)\,dx = \alpha_0 p(x_0) + \alpha_1 p(x_1) + \cdots + \alpha_n p(x_n). \tag{7.5}$$

Recalling the Lagrange interpolation for the data $(x_0, p(x_0)), \ldots, (x_n, p(x_n))$, where

$$L_n(x) = p(x_0)l_0(x) + \cdots + p(x_n)l_n(x), \quad l_i(x) = \prod_{j\neq i, j=0}^{n} \frac{x - x_j}{x_i - x_j}.$$

Since both $L_n(x)$ and $p(x)$ are polynomials of degree n that agrees at $n+1$ points, by the uniqueness Theorem 6.3, $p(x) = L_n(x)$. Hence,

$$\int_a^b p(x)\,dx = \int_a^b p(x_0)l_0(x) + \cdots + p(x_n)l_n(x)\,dx = \sum_{i=0}^{n} p(x_i) \int_a^b l_i(x)\,dx.$$

This means we should choose

$$\alpha_i = \int_a^b l_i(x)\,dx, \quad i = 0, 1, \ldots, n, \tag{7.6}$$

and motivates us to consider the Newton–Cotes rule for any function f:

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} \alpha_i f(x_i), \quad \alpha_i = \int_a^b l_i(x)\,dx. \tag{7.7}$$

It is worthwhile to investigate the special cases of $n = 1$ and $n = 2$. For $n = 1$, let $x_0 = a$ and $x_1 = b$, then

$$\alpha_0 = \int_a^b l_0(x)\,dx = \int_a^b \frac{x-b}{a-b}\,dx = \frac{1}{2}(b-a),$$
$$\alpha_1 = \int_a^b l_1(x)\,dx = \int_a^b \frac{x-a}{b-a}\,dx = \frac{1}{2}(b-a),$$

and so we get the Newton–Cotes rule for $n = 1$:

$$\int_a^b f(x)dx \approx \frac{b-a}{2}(f(a) + f(b)),$$

which is exactly the trapezoidal rule (7.2).

For $n = 2$, let $x_0 = a$, $x_1 = \frac{1}{2}(a+b)$ and $x_2 = b$, then if we set $h = b - a$ we get

$$\begin{aligned}
\alpha_0 &= \int_a^b l_0(x)\,dx = \int_a^b \frac{(x-x_1)(x-b)}{(a-x_1)(a-b)}\,dx \\
&= \frac{2}{(b-a)^2}\int_a^b (x-b+h/2)(x-b)\,dx = \frac{1}{6}(b-a), \\
\alpha_1 &= \int_a^b l_1(x)\,dx = \int_a^b \frac{(x-a)(x-b)}{(x_1-a)(x_1-b)}\,dx \\
&= -\frac{4}{(b-a)^2}\int_a^b (x-a)(x-a+a-b)\,dx = \frac{4}{6}(b-a), \\
\alpha_2 &= \int_a^b l_2(x)\,dx = \int_a^b \frac{(x-a)(x-x_1)}{(b-a)(b-x_1)}\,dx \\
&= \frac{1}{6}(b-a),
\end{aligned}$$

and so we get the Newton–Cotes rule for $n = 2$:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\Big\{f(a) + 4f((a+b)/2) + f(b)\Big\},$$

which is called the **Simpson's rule**.

One can easily see that it is very technical and lengthy to compute the coefficients α_i of the Newton–Cotes rules using (7.6) for larger n. We now give an alternative derivation of the trapezoidal rule and Simpson's rule based on the direct definition of the Newton–Cotes rule

The quadrature rule (7.5) *holds for all polynomials of degree* $\le n$.

When $n = 1$, we need two points x_0 and x_1. Let us consider $x_0 = a$ and $x_1 = b$. As the quadrature rule

$$\int_a^b f(x)\,dx = \alpha_0 f(a) + \alpha_1 f(b)$$

holds for all polynomials of degree $n \le 1$, we try with $f(x) = 1$ to obtain

$$b - a = \alpha_0 + \alpha_1,$$

and with $f(x) = x - a$ to obtain

$$\frac{1}{2}(b-a)^2 = \alpha_1(b-a).$$

This implies that

$$\alpha_0 = \frac{b-a}{2}, \quad \alpha_1 = \frac{b-a}{2}$$

and yields the trapezoidal rule:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}(f(a)+f(b)).$$

When $n = 2$, we need three points x_0, x_1 and x_2. Let us consider that $x_0 = a$, $x_1 = (a+b)/2$ and $x_2 = b$, Then using the fact that the formula

$$\int_a^b f(x)\,dx = \alpha_0 f(x_0) + \alpha_1 f(x_1) + \alpha_2 f(x_2)$$

is exact for all polynomials of degree $n \leq 2$, we obtain

$$\begin{aligned} (b-a) &= \int_a^b 1\,dx = \alpha_0 + \alpha_1 + \alpha_2, \\ \frac{1}{2}(b-a)^2 &= \int_a^b (x-a)\,dx = \alpha_1 \frac{b-a}{2} + \alpha_2(b-a), \\ -\frac{1}{6}(b-a)^3 &= \int_a^b (x-a)(x-b)\,dx = -\alpha_1 \frac{(b-a)^2}{4} \end{aligned}$$

by taking

$$f(x) = 1, \quad f(x) = x-a, \quad f(x) = (x-a)(x-b).$$

Solving the system, we derive the Simpson's rule:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\Big\{f(a) + 4f((a+b)/2) + f(b)\Big\}.$$

7.3.1 Simpson's rule

Simpson's rule is one of the most important quadrature rules due to its nice properties. First we give the error estimate for $f : [a,b] \to \mathbb{R}$ with $\max_{x\in[a,b]} |f^{(4)}(x)| \leq K$. Let

$$F(x) = \int_a^x f(t)\,dt,$$

$h = b - a$ and $\bar{x} = (a+b)/2$. Then we expand $F(b)$ at $x = a$ by Taylor series up to the 5th order to obtain:

$$\begin{aligned} F(b) &= F(a) + h\,F'(a) + \frac{h^2}{2}F''(a) + \frac{h^3}{6}F'''(a) + \frac{h^4}{24}F^{(4)}(a) + \frac{h^5}{120}F^{(5)}(\xi) \\ &= h\,f(a) + \frac{h^2}{2}f'(a) + \frac{h^3}{6}f''(a) + \frac{h^4}{24}f^{(3)}(a) + \frac{h^5}{120}f^{(4)}(\xi). \end{aligned} \tag{7.8}$$

On the other hand, we can expand each term on the right-hand side of Simpson's rule to get

$$f(a) = f(a), \tag{7.9}$$

$$f(\bar{x}) = f(a) + \frac{h}{2}f'(a) + \Big(\frac{h}{2}\Big)^2\frac{f''(a)}{2} + \Big(\frac{h}{2}\Big)^3\frac{f'''(a)}{6} + \Big(\frac{h}{2}\Big)^4\frac{f^{(4)}(\mu_1)}{24}, \tag{7.10}$$

$$f(b) = f(a) + hf'(a) + \frac{h^2}{2}f''(a) + \frac{h^3}{6}f'''(a) + \frac{h^4}{24}f^{(4)}(\mu_2), \tag{7.11}$$

for some $\mu_1 \in (a, \bar{x})$ and $\mu_2 \in (a, b)$. Using the relations (7.9)-(7.11), we deduce (recalling $h = b - a$)

$$\begin{aligned}\frac{b-a}{6}\Big\{f(a) + 4f(\bar{x}) + f(b)\Big\} = hf(a) &+ \frac{h^2}{2}f'(a) + \frac{h^3}{6}f''(a) + \frac{h^4}{24}f'''(a) \\ &+ \frac{h^5}{576}(f^{(4)}(\mu_1) + 4f^{(4)}(\mu_2)).\end{aligned}$$

Subtracting this from (7.8) yields

$$\begin{aligned}&\int_a^b f(x)\,dx - \frac{b-a}{6}\Big\{f(a) + 4f(\bar{x}) + f(b)\Big\} \\ &\quad= \frac{h^5}{120}f^{(4)}(\xi) - \frac{h^5}{576}(f^{(4)}(\mu_1) + 4f^{(4)}(\mu_2)).\end{aligned} \tag{7.12}$$

This shows that Simpson's rule is exact for polynomials of degree ≤ 3, and by the assumption $\max_{x\in[a,b]} |f^{(4)}(x)| \leq K$, we obtain

$$\left|\int_a^b f(x)\,dx - \frac{b-a}{6}\Big\{f(a) + 4f(\bar{x}) + f(b)\Big\}\right| \leq \frac{49}{2880}Kh^5 = \frac{49}{2880}K(b-a)^5.$$

This may not be such a useful error estimate if the interval $[a, b]$ is not small, and so we divide $[a, b]$ into equally-spaces subintervals

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b\ , \quad x_i = a + ih, \quad h = \frac{b-a}{n},$$

and use the Simpson's rule on each subinterval. This gives the following **composite Simpson's rule**:

$$\begin{aligned}\int_a^b f(x)\,dx &= \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} f(x)\,dx \\ &= \int_{x_0}^{x_1} f(x)\,dx + \int_{x_1}^{x_2} f(x)dx + \cdots + \int_{x_{n-1}}^{x_n} f(x)\,dx \\ &\approx \frac{h}{6}\sum_{i=1}^{n}\Big\{f(x_{i-1}) + 4f\Big(\frac{x_{i-1}+x_i}{2}\Big) + f(x_i)\Big\}.\end{aligned}$$

The error estimate of the composite Simpson rule is a direct application of the error estimate of the Simpson rule:

$$\begin{aligned} &\left| \int_a^b f(x)\,dx - \frac{h}{6}\sum_{i=1}^n \Big\{ f(x_{i-1}) + 4f\Big(\frac{x_{i-1}+x_i}{2}\Big) + f(x_i) \Big\} \right| \\ &= \left| \sum_{i=1}^n \Big[\int_{x_i-1}^{x_i} f(x)\,dx - \frac{h}{6}\Big\{ f(x_{i-1}) + 4f\Big(\frac{x_{i-1}+x_i}{2}\Big) + f(x_i) \Big\}\Big] \right| \qquad (7.13) \\ &\le \sum_{i=1}^n \frac{49}{2880} K(x_i - x_{i-1})^5 = \frac{49K}{2880} nh^5 = \frac{49K}{2880}(b-a)h^4. \end{aligned}$$

7.4 Gaussian quadrature rule

A key requirement of the Newton–Cote rule is that the nodal points $\{x_0, \dots, x_n\}$ form an equally-spaced partition of the interval $[a, b]$, and that

$$\int_a^b p(x)\,dx = \sum_{i=0}^n \alpha_i p(x_i) \text{ for all polynomials of degree } \le n.$$

In fact, there exist quadrature rules that have better accuracy, namely with the same number $n+1$ of nodal points, the integral approximation is exact for all polynomials of degree $\le 2n+1$. This comes from relaxing the condition that $x_0, \dots, x_n$ form an **equally-spaced** partition of $[a, b]$, and the subsequent quadrature rules are known as **Gaussian quadrature rules**.

We begin with an example with $n = 1$. For simplicity we take $[a, b] = [-1, 1]$, and suppose we want

$$\int_{-1}^1 f(x)\,dx = \alpha_0 f(x_0) + \alpha_1 f(x_1)$$

to hold for all polynomials of degree $\le 2n+1 = 3$, then for $f(x) = 1$, $f(x) = x$, $f(x) = x^2$ and $f(x) = x^3$, we must have

$$\int_{-1}^1 1\,dx = 2 = \alpha_0 + \alpha_1, \qquad (7.14)$$

$$\int_{-1}^1 x\,dx = 0 = \alpha_0 x_0 + \alpha_1 x_1, \qquad (7.15)$$

$$\int_{-1}^1 x^2\,dx = \frac{2}{2} = \alpha_0 x_0^2 + \alpha_1 x_1^2, \qquad (7.16)$$

$$\int_{-1}^1 x^3\,dx = 0 = \alpha_0 x_0^3 + \alpha_1 x_1^3. \qquad (7.17)$$

These are four equations for four unknowns $(\alpha_0, \alpha_1, x_0, x_1)$, but they form a nonlinear system of equations. To solve this, let us recall that

$$\alpha_0 = \int_{-1}^{1} l_0(x)\,dx = \int_{-1}^{1} \frac{x - x_1}{x_0 - x_1}\,dx = \frac{2x_1}{x_1 - x_0},$$
$$\alpha_1 = \int_{-1}^{1} l_1(x)\,dx = \int_{-1}^{1} \frac{x - x_0}{x_1 - x_0}\,dx = -\frac{2x_0}{x_1 - x_0}.$$

Adding these gives $\alpha_0 + \alpha_1 = 2$, which is (7.14). From (7.15) we get

$$\alpha_0 x_0 + \alpha_1 x_1 = 2(x_1 + x_0) = 0 \quad \Rightarrow \quad x_0 = -x_1,$$

and from (7.17) we get

$$\alpha_0 x_0^3 + \alpha_1 x_1^3 = (\alpha_1 - \alpha_0)x_1^3 = 0 \quad \Rightarrow \quad \alpha_0 = \alpha_1.$$

Hence, by (7.16) it holds that

$$\alpha_0 x_0^2 + \alpha_1 x_1^2 = 2\alpha_0 x_1^2 = \frac{4x_1^3}{2x_1} = 2x_1^2 = \frac{2}{3} \quad \Rightarrow \quad x_1 = \frac{1}{\sqrt{3}}.$$

This implies

$$x_0 = -\frac{1}{\sqrt{3}}, \quad x_1 = \frac{1}{\sqrt{3}}, \quad \alpha_0 = \alpha_1 = 1,$$

and leads to the **2 point Gaussian quadrature rule**:

$$\int_{-1}^{1} f(x)\,dx \approx f\Big(\frac{-1}{\sqrt{3}}\Big) + f\Big(\frac{1}{\sqrt{3}}\Big). \tag{7.18}$$

By the above calculations, we see that this is exact for all polynomials of degree ≤ 3. Furthermore, we immediately see that the partition

$$-1 = a < x_0 = \frac{-1}{\sqrt{3}} < x_1 = \frac{1}{\sqrt{3}} < b = 1$$

is not **equally-spaced**!

Unfortuately, the amount of effort needed to compute the points $(x_0, \dots, x_n)$ and coefficients $(\alpha_0, \cdots, \alpha_n)$ for Gaussian quadrature rules increases drastically as n increases. Take for example $n = 2$, then a **3 point Gaussian quadrature rule** is

$$\int_{-1}^{1} f(x)\,dx \approx \alpha_0 f(x_0) + \alpha_1 f(x_1) + \alpha_2 f(x_2),$$

such that for any polynomials $p(x)$ of degree $\leq 2n + 1 = 5$,

$$\int_{-1}^{1} p(x)\,dx = \alpha_0 p(x_0) + \alpha_1 p(x_1) + \alpha_2 p(x_2).$$

Proceeding as before, we take $p(x) = 1$, $p(x) = x$, $p(x) = x^2$, $p(x) = x^3$, $p(x) = x^4$ and $p(x) = x^5$ to obtain a nonlinear system of six equations for six unknowns:

$$\begin{aligned}
\alpha_0 + \alpha_1 + \alpha_2 &= 2,\\
\alpha_0 x_0 + \alpha_1 x_1 + \alpha_2 x_2 &= 0,\\
\alpha_0 x_0^2 + \alpha_1 x_1^2 + \alpha_2 x_2^2 &= \frac{2}{3},\\
\alpha_0 x_0^3 + \alpha_1 x_1^3 + \alpha_2 x_2^3 &= 0,\\
\alpha_0 x_0^4 + \alpha_1 x_1^4 + \alpha_2 x_2^4 &= \frac{2}{5},\\
\alpha_0 x_0^5 + \alpha_1 x_1^5 + \alpha_2 x_2^5 &= 0.
\end{aligned}$$

One can try to solve this nonlinear system with Newton's method or steepest descent, but suppose we choose the quadrature points $x_0, \dots, x_n$ as roots of some polynomial Q_{n+1}, then we can immediately determine the coefficients $\alpha_0, \dots, \alpha_n$ by the definition

$$\alpha_i = \int_{-1}^{1} l_i(x)\, dx.$$

It is clear that for different choice of the polynomial Q_{n+1}, we get different roots and hence different nodal points, which leads to different quadrature rules. We shall study the case where Q_{n+1} is part of the so-called Legendre polynomials.

7.4.1 Gauss–Legendre quadrature

Definition 7.1 (Legendre polynomials)**.** *The* **Legendre polynomials** $P_n(x)$ *is defined recursively by*

$$P_0(x) = 1, \quad P_1(x) = x, \quad (n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x).$$

For example,

$$P_2(x) = \frac{1}{2}(3x^2 - 1), \quad P_3(x) = \frac{1}{2}(5x^3 - 3x), \quad \cdots .$$

Then the method for deriving the Gauss–Legendre quadrature rules is as follows:

1. Let $\{x_0, \dots, x_n\}$ be the roots of $P_{n+1}(x)$.
2. Set
$$\alpha_i = \int_{-1}^{1} l_i(x)\, dx = \int_{-1}^{1} \prod_{j\neq i, j=0}^{n} \frac{x - x_j}{x_i - x_j}\, dx.$$
3. Then, the Gauss–Legendre quadrature rule is
$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=0}^{n} \alpha_i f(x_i).$$

We now provide some facts about the Legendre polynomials:

(a) $P_n(x)$ is the solution to the **Legendre differential equation**:

$$\frac{d}{dx}\Big((1-x^2)\frac{d}{dx}P_n(x)\Big) + n(n+1)P_n(x) = 0.$$

(b) Rodrigue's formula:

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}(x^2-1)^n.$$

(c) Orthogonality:

$$\int_{-1}^{1} P_m(x)P_n(x)\,dx = 0 \text{ if } m \neq n.$$

(d) Basis: any polynomial $q(x)$ of degree n can be expressed uniquely as

$$q(x) = \beta_0 P_0(x) + \beta_1 P_1(x) + \cdots + \beta_n P_n(x),$$

for constants $\beta_0, \ldots, \beta_n$.

(e) Combining properties (c) and (d) gives

$$\int_{-1}^{1} q(x)P_{n+1}(x)\,dx = 0 \text{ for any polynomial } q \text{ of degree } \leq n.$$

We give a simple proof of the orthogonality property (c): Take the Legendre differential equation for P_n and multiply by P_m, and take the Legendre differential equation for P_m and multiply by P_n, which gives

$$[((1-x^2)P_n'(x))' + n(n+1)P_n(x)]P_m(x) = 0,$$
$$[((1-x^2)P_m'(x))' + m(m+1)P_m(x)]P_n(x) = 0.$$

Subtracting one from the other and using the relation

$$((1-x^2)P_n'(x))'P_m(x) - ((1-x^2)P_m'(x))'P_n(x) = ((1-x^2)(P_n'P_m - P_m'P_n))',$$

we get

$$((1-x^2)(P_n'P_m - P_m'P_n))' + (n(n+1) - m(m+1))P_nP_m = 0.$$

Integrating over $(-1,1)$ yields

$$\underbrace{[(1-x^2)(P_n'P_m - P_m'P_n)]_{-1}^{1}}_{=0} + \underbrace{(n(n+1) - m(m+1))}_{\neq 0}\int_{-1}^{1} P_n(x)P_m(x)\,dx = 0.$$

With these properties, we have the following result.

Theorem 7.1. *The Gauss–Legendre quadrature rule defined above is exact for any polynomials of degree* $\leq 2n+1$.

Proof. Let $p(x)$ be a polynomial of degree $\leq 2n+1$ and as the monic $(n+1)$th Legendre polynomial has degree $n+1$, we can find (by factorization) two polynomials $q(x)$ and $r(x)$ of degree n such that $p(x) = q(x)P_{n+1}(x) + r(x)$. Then, as $\{x_0, \dots, x_n\}$ are the roots of $P_{n+1}(x)$, it holds that

$$p(x_i) = q(x_i)P_{n+1}(x_i) + r(x_i) = r(x_i).$$

Since r is a polynomial of degree n, by Lagrange interpolation,

$$r(x) = \sum_{i=0}^{n} r(x_i)l_i(x) = \sum_{i=0}^{n} p(x_i)l_i(x).$$

Hence,

$$\begin{aligned}\int_{-1}^{1} p(x)\,dx &= \int_{-1}^{1} q(x)P_{n+1}(x)\,dx + \int_{-1}^{1} r(x)\,dx \\ &= \int_{-1}^{1} r(x)\,dx = \int_{-1}^{1} \sum_{i=0}^{n} p(x_i)l_i(x)\,dx = \sum_{i=0}^{n} \alpha_i p(x_i),\end{aligned}$$

where in the above we used property (e) and the fact that q is a polynomial of degree n. □

We now compute the 2 point and 3 point Gauss–Legendere quadrature rule following the above procedure. First, suppose we have 2 points x_0 and x_1, i.e., the case $n = 1$. Then, we should take them to be the roots of the second Legendre polynomial $P_2(x) = \frac{1}{2}(3x^2 - 1)$, i.e.,

$$x_0 = -\frac{1}{\sqrt{3}}, \quad x_1 = \frac{1}{\sqrt{3}}.$$

Then, computing the coefficients α_0 and α_1 yields

$$\alpha_0 = \frac{2x_1}{x_1 - x_0} = 1, \quad \alpha_1 = \frac{-2x_0}{x_1 - x_0} = 1,$$

and so the **2 point Gauss–Legendre quadrature rule** is

$$\int_{-1}^{1} f(x)\,dx \approx f\Big(\frac{-1}{\sqrt{3}}\Big) + f\Big(\frac{1}{\sqrt{3}}\Big),$$

which is exactly the 2 point Gaussian quadrature rule (7.18). For the case $n = 2$, we have three points x_0, x_1 and x_2, chosen to be the roots of $P_3(x) = \frac{1}{2}(5x^3 - 3x)$, and so

$$x_0 = -\sqrt{\frac{3}{5}}, \quad x_1 = 0, \quad x_2 = \sqrt{\frac{3}{5}}.$$

For the coefficients, we get

$$\begin{aligned}
\alpha_0 &= \int_{-1}^{1} \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}\,dx = \frac{2/3 + 2x_1x_2}{(x_0-x_1)(x_0-x_2)} = \frac{5}{9},\\
\alpha_1 &= \int_{-1}^{1} \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}\,dx = \frac{2/3 + 2x_0x_2}{(x_1-x_0)(x_1-x_2)} = \frac{8}{9},\\
\alpha_2 &= \int_{-1}^{1} \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}\,dx = \frac{2/3 + 2x_0x_1}{(x_2-x_0)(x_2-x_1)} = \frac{5}{9}.
\end{aligned}$$

Hence, the **3 point Gauss–Legendre quadrature rule** is

$$\int_{-1}^{1} f(x)\,dx \approx \frac{5}{9} f\Big(-\sqrt{\frac{3}{5}}\Big) + \frac{8}{9} f(0) + \frac{5}{9} f\Big(\sqrt{\frac{3}{5}}\Big).$$

It remain to obtain an error estimate for the Gauss–Legendre quadrature rule.

7.4.2 Error estimate

Theorem 7.2. *Let $\{x_0, \ldots, x_n\}$ be the roots of $P_{n+1}(x)$, the $(n+1)$th Legendre polynomial, and define*

$$\alpha_i = \int_{-1}^{1} l_i(x)\,dx = \int_{-1}^{1} \prod_{j\neq i, j=0}^{n} \frac{x-x_j}{x_i-x_j}\,dx \text{ for } i = 0, \ldots, n.$$

Then, $\alpha_i > 0$ and if $f \in C^{2n+2}[-1,1]$ with $\max_{x\in[-1,1]} |f^{(2n+2)}(x)| \leq K$ for some $K > 0$, we have

$$\left| \int_{-1}^{1} f(x)\,dx - \sum_{i=0}^{n} \alpha_i f(x_i) \right| \leq \frac{K}{(2n+2)!} \int_{-1}^{1} (x-x_0)^2 \cdots (x-x_n)^2\,dx.$$

Proof. Fix $i \in \{0, \ldots, n\}$, and consider the polynomial

$$q(x) = (l_i(x))^2 = \prod_{j\neq i, j=0}^{n} \frac{(x-x_j)^2}{(x_i-x_j)^2}.$$

Then, q is a polynomial of degree $2n$ and

$$q(x_k) = (l_i(x_k))^2 = \begin{cases} 1 & \text{if } k = i,\\ 0 & \text{if } k \neq i. \end{cases}$$

Since q is non-negative and only zero at $x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n$, the integral of q over $(-1,1)$ is strictly positive, and so

$$0 < \int_{-1}^{1} q(x)\,dx = \sum_{k=0}^{n} \alpha_k q(x_k) = \alpha_i,$$

where the middle equality is due to the exactness of the Gauss–Legendre quadrature rule for polynomials of degree $\leq 2n+1$. For the error estimate, we consider the Hermite's interpolation $H_{2n+1}(x)$ of degree $2n+1$ such that

$$H_{2n+1}(x_i) = f(x_i), \quad H'_{2n+1}(x_i) = f'(x_i) \text{ for } i = 0, \dots, n.$$

Then, by Theorem 6.8 there exists $\xi \in (-1,1)$ such that

$$f(x) - H_{2n+1}(x) = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}(x-x_0)^2 \cdots (x-x_n)^2.$$

As $H_{2n+1}(x)$ is of degree $2n+1$, the Gauss–Legendre quadrature rule is exact and gives

$$\int_{-1}^{1} H_{2n+1}(x)\,dx = \sum_{i=0}^{n} \alpha_i H_{2n+1}(x_i) = \sum_{i=0}^{n} \alpha_i f(x_i).$$

Hence,

$$\begin{aligned}\left|\int_{-1}^{1} f(x)\,dx - \sum_{i=0}^{n} \alpha_i f(x_i)\right| &= \left|\int_{-1}^{1} f(x) - H_{2n+1}(x)\,dx\right| \\ &= \left|\int_{-1}^{1} \frac{f^{(2n+2)}(\xi)}{(2n+2)!}(x-x_0)^2 \cdots (x-x_n)^2\,dx\right| \\ &\leq \frac{K}{(2n+2)!}\int_{-1}^{1} (x-x_0)^2 \cdots (x-x_n)^2\,dx.\end{aligned}$$

□

Example 7.2. *Let $f : [-1,1] \to \mathbb{R}$ be a function with $\max_{x\in[-1,1]} |f^{(4)}(x)| \leq 1$, then for the 2 point Gauss–Legendre quadrature rule we have*

$$\int_{-1}^{1} f(x)\,dx = f\Big(\frac{-1}{\sqrt{3}}\Big) + f\Big(\frac{1}{\sqrt{3}}\Big).$$

Then, from above, the error is

$$\begin{aligned}\left|\int_{-1}^{1} f(x)\,dx - \Big(f\Big(\frac{-1}{\sqrt{3}}\Big) + f\Big(\frac{1}{\sqrt{3}}\Big)\Big)\right| &\leq \frac{1}{4!}\int_{-1}^{1} \Big(x + \frac{1}{\sqrt{3}}\Big)^2\Big(x - \frac{1}{\sqrt{3}}\Big)^2\,dx \\ = \frac{1}{24}\int_{-1}^{1}\Big(x^2 - \frac{1}{3}\Big)^2\,dx &= \frac{1}{24}\Big[\frac{x^5}{5} - \frac{2}{9}x^3 + \frac{x}{9}\Big]_{-1}^{1} = \frac{1}{135}.\end{aligned}$$

7.4.3 Quadrature rule on arbitrary intervals

While the Gauss–Legendre quadrature rules are derived in the interval $[-1,1]$, for any $f : [a,b] \to \mathbb{R}$, we can use the linear transformation

$$y = h(x) = \frac{a+b}{2} + \frac{b-a}{2}x,$$

so that

$$\int_a^b f(y)\,dy = \int_{-1}^1 f\Big(\frac{a+b}{2} + \frac{b-a}{2}x\Big)\frac{b-a}{2}\,dx =: \int_{-1}^1 g(x)\,dx.$$

Then, the 2 point Gauss–Legendre quadrature rule is

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\Big(f\Big(\frac{a+b}{2} - \frac{b-a}{2\sqrt{3}}\Big) + f\Big(\frac{a+b}{2} + \frac{b-a}{2\sqrt{3}}\Big)\Big),$$

while the 3 point Gauss–Legendre quadrature rule is

$$\begin{aligned}&\int_a^b f(x)\,dx\\ &\quad\approx \frac{b-a}{2}\Big(\frac{5}{9}f\Big(\frac{a+b}{2} - \frac{(b-a)\sqrt{3}}{2\sqrt{5}}\Big) + \frac{8}{9}f\Big(\frac{a+b}{2}\Big) + \frac{5}{9}f\Big(\frac{a+b}{2} + \frac{(b-a)\sqrt{3}}{2\sqrt{5}}\Big)\Big).\end{aligned}$$