DSA2102 Numerical Computation

Thang Pang Ern

Reference books:

(1). M. Heath. *Scientific Computing*. 2nd edition, McGraw-Hill Professional, New York, 2001. ISBN: 9780072399103.

Contents

1 Computer Arithmetic **3**
1.1 Approximations and Errors . . . 3
1.2 Scientific Notation . . . 6
1.3 Floating Point Systems and Computing Operations . . . 13

2 Systems of Linear Equations **29**
2.1 Matrix and Vector Operations . . . 29
2.2 Systems of Linear Equations . . . 41
2.3 Forward Elimination and Backward Substitution . . . 45
2.4 LU Factorisation . . . 46
2.5 Pivoting . . . 52
2.6 Some Special Systems and the Cholesky Factorisation . . . 60

3 Linear Least Squares **71**
3.1 Least Squares Problems . . . 71
3.2 QR Factorisation . . . 76
3.3 The Householder Reflection . . . 79
3.4 The Givens Rotation . . . 83

4 Eigenvalue Problems **86**
4.1 Recap on Eigenvalues and Eigenvectors . . . 86
4.2 Singular Value Decomposition . . . 87
4.3 Root Finding Algorithms . . . 89
4.4 Power Iteration . . . 92
4.5 QR Iteration . . . 96

5 Interpolation and Approximation **97**
5.1 Polynomial Interpolation . . . 97
5.2 Piecewise Interpolation . . . 104
5.3 Orthogonal Polynomials . . . 108

6 Numerical Integration and Differentiation **113**
6.1 Newton-Cotes Quadrature Rules . . . 113
6.2 Numerical Differentiation . . . 120

Chapter 1

Computer Arithmetic

1.1 Approximations and Errors

There are several steps involved when trying to solve a problem computationally. We first develop a mathematical model, then develop algorithms to solve the equations numerically. Next, we implement them on a computer and run the simulations. Lastly, we represent the results comprehensibly, interpret and validate them.

Our focus is on the development and implementation of algorithms. First, we discuss some considerations one should have when solving a problem computationally.

Definition 1.1 (well-posed problem)**.** We say that a problem is well-posed if a solution exists, is unique, and varies continuously with the problem data.

Note that even when a problem is well-posed, it might be the case that relatively small perturbations in the inputs lead to relatively large changes in outputs. We will give a precise way to measure this sensitivity to perturbation in due course. While well-posedness is a very desirable property, many important problems are inherently *ill-posed*. It should be noted that well-posedness is a property of a mathematical problem, not of an algorithm. We wish that our algorithms are stable. That is to say (briefly), one that does not make the sensitivity of the underlying problem worse.

Example 1.1. We can consider the problem of computing the surface area A of the Earth using the formula $A = 4\pi r^2$. First, we model the Earth as a sphere. We then use an estimate $r = 6370$ km, where r refers to the radius of the Earth, based on measurements and prior computations. We will have to truncate the value of π at some point, and our computer will use rounding when making computations.

Given a quantity Q and an approximation A, the absolute error is $|Q - A|$ and the relative error is $\frac{|Q-A|}{|Q|}$. Often, the relative error is more meaningful, especially when $|Q|$ is large. For $|Q|$ near zero, the relative error may be inappropriate. It is

useful to distinguish *accuracy* from *precision*. An approximation is accurate when the error is small. Precision refers to the number of significant digits. For example, $x = 2.03048154248$ is very precise, but it is not a very accurate approximation of $\pi = 3.14159265359\ldots$.

When analysing errors, it helps to separate sources of error. Some errors are due to the data, while others arise from the computational process. Consider evaluating a function $f : \mathbb{R} \to \mathbb{R}$ at a point x. Often we only have an approximation $\widehat{x}$ to x, and we may also replace f by an approximate function $\widehat{f}$. Then,

$$\begin{aligned}\text{Total error} &= \widehat{f}\left(\widehat{x}\right) - f\left(x\right) \\ &= \left(\widehat{f}\left(\widehat{x}\right) - f\left(\widehat{x}\right)\right) + \left(f\left(\widehat{x}\right) - f\left(x\right)\right) \\ &= \text{computational error} + \text{data error}\end{aligned}$$

Example 1.2. Say we wish to compute $\sin\left(\frac{\pi}{8}\right)$. We use the approximations $\pi \approx 3$ and $\sin t \approx t$ for small t. Then, $\sin\left(\frac{\pi}{8}\right) \approx \frac{3}{8} = 0.375$. In fact, to three decimal places, we have $\sin\left(\frac{\pi}{8}\right) \approx 0.383$. Hence, the total error is -0.008, the computational error is ≈ 0.009, and the data error is ≈ -0.017.

The computational error can be further subdivided into *truncation error* and *rounding error*. That is,

$$\text{computational error} = \text{truncation error} + \text{rounding error}.$$

Truncation error results from approximations such as truncating series, discretisation, or terminating an iteration early. Rounding error comes from finite-precision arithmetic.

Example 1.3 (finite difference)**.** We give an example involving differentiation. Note that we will revisit this in Chapter 6.2. Let $f : \mathbb{R} \to \mathbb{R}$ be a differentiable function. Consider the forward difference

$$f'\left(x\right) \approx \frac{f\left(x+h\right) - f\left(x\right)}{h}.$$

By Taylor's theorem, there exists $x \leq a \leq x + h$ such that

$$f\left(x+h\right) = f\left(x\right) + f'\left(x\right)h + \frac{f''\left(a\right)}{2}h^2$$

so

$$f'\left(x\right) = \frac{f\left(x+h\right) - f\left(x\right)}{h} - \frac{f''\left(a\right)}{2}h.$$

If $|f''(t)| \leq M$ near x, then the truncation error is bounded by $\frac{M}{2}|h|$. If the error in each function value is bounded by ε, then the rounding error in the difference quotient satisfies

$$\left| \frac{f(x+h) - f(x)}{h} - \frac{\big(f(x+h) \pm \varepsilon\big) - \big(f(x) \pm \varepsilon\big)}{h} \right| \leq \frac{|\varepsilon| + |\varepsilon|}{|h|} = \frac{2\varepsilon}{|h|}.$$

Thus, the total computational error is bounded by

$$\frac{M}{2}|h| + \frac{2\varepsilon}{|h|}.$$

This illustrates a trade-off — taking $|h|$ small reduces truncation error but increases rounding error. In algebraic problems and finite-step algorithms, rounding error often dominates; in limit-based processes like derivatives and integrals, truncation error is often more significant.

We then discuss forward and backward error. Suppose we wish to solve the equation $y = f(x)$. The absolute forward error of an approximation $\widehat{y}$ is defined to be

$$|\widehat{y} - y|,$$

with the relative forward error defined analogously. Often, this is difficult to estimate directly. Instead, view $\widehat{y}$ as the exact solution to a nearby problem $\widehat{y} = f(\widehat{x})$ and define the absolute backward error as $|\widehat{x} - x|$.

Example 1.4. If $y = \sqrt{2}$ and $\widehat{y} = 1.4$, then

$$|\Delta y| = |\widehat{y} - y| \approx 0.0142 \quad \widehat{x} = \widehat{y}^2 = 1.96 \quad |\Delta x| = |\widehat{x} - x| = |1.96 - 2| = 0.04.$$

The backward error asks how much we must perturb the input to make the computed answer exact. If that perturbation is small, the solution is still *good*.

We relate forward and backward errors via the condition number, defined as

$$\text{condition number} = \frac{|\text{relative forward error}|}{|\text{relative backward error}|} = \left| \frac{\dfrac{\widehat{f}(\widehat{x}) - f(x)}{f(x)}}{\dfrac{\widehat{x} - x}{x}} \right| = \left| \frac{\Delta y / y}{\Delta x / x} \right|.$$

When we talk about conditioning of a problem, we really refer to measuring the multiplier between the size of the error one makes in the input, and the size of the

resulting error in the output. In practice, we use the differential approximation

$$\text{forward error} = f(x + \Delta x) - f(x) \approx f'(x)\,\Delta x,$$

so

$$\text{condition number} \approx \left| \frac{f'(x)\,\Delta x / f(x)}{\Delta x / x} \right| = \left| \frac{x f'(x)}{f(x)} \right|. \tag{1.1}$$

In general, a condition number of 1 is good. That is, the relative forward error is equal to the relative backward error — whatever distortion one has introduced into the input is reflected one-for-one in the output. As conditioning tells you how sensitive the true solution is to changes in the input data, it is clear that conditioning is not affected by the algorithm used to solve the problem. Stability, on the other hand, is a property of the algorithm as it tells one how much extra error the algorithm introduces due to things like rounding and finite precision.

Example 1.5. For $f(x) = \sqrt{x}$, we have $f'(x) = \frac{1}{2\sqrt{x}}$. By (1.1), we have

$$\text{condition number} \approx \left| \frac{x}{2\sqrt{x}\sqrt{x}} \right| = \frac{1}{2}.$$

Definition 1.2 (ill-conditioned problem)**.** A problem is *ill-conditioned* (or sensitive) if the condition number is much larger than 1.

Example 1.6. For $f(x) = \tan(x)$, we have $f'(x) = \sec^2 x = 1 + \tan^2 x$, so

$$\text{condition number} \approx \left| \frac{x\,(1 + \tan^2 x)}{\tan x} \right| = \left| x\,(\tan x + \cot x) \right|.$$

This becomes problematic near integer multiples of $\frac{\pi}{2}$. For instance,

$$\tan(1.57079) \approx 1.58058 \times 10^5 \quad \text{whereas} \quad \tan(1.57078) \approx 6.12490 \times 10^4.$$

1.2 Scientific Notation

We then try to understand how computers perform arithmetic so that we can see how they are a source of error. The numbers one stores into a computer are not necessarily the numbers the computer *actually stores*. For example, in single precision format, we would have the following:

(i) Input: 0.23 but output: 0.2300000042

(ii) Input: 0.25 but output 0.2500000000
(iii) Input: $\sqrt[3]{1.728} \cdot \sqrt[3]{1.728} \cdot \sqrt[3]{1.728} - 1.728$[†] but output: 1.6403199×10^{-7}
(iv) Input $\sqrt[3]{3.375} \cdot \sqrt[3]{3.375} \cdot \sqrt[3]{3.375} - 3.375$ but output: 0

On our computers, some numbers can be represented exactly, while some cannot. Our first step in understanding why is to understand different ways of representing numbers.

We are all used to the decimal system, where place values correspond to powers of 10. So,

$$833.71 = 8 \cdot 10^2 + 3 \cdot 10^1 + 3 \cdot 10^0 + 7 \cdot 10^{-1} + 1 \cdot 10^{-2}.$$

In decimal, ten is the base, while in binary, the base is two. Bases are typically integers ≥ 2, though this is not necessary. Note that if b is an integer ≥ 2, then the representation of b in base b is 10. For clarity, parentheses and subscripts to indicate which base a number is written with respect to. The subscripts are always written in base ten. For example,

$$(11.1)_2 = (3.5)_{10} \quad \text{and} \quad (11.1)_{10} = (1011.0001100110011\ldots)_2\,. \tag{1.2}$$

The second example in (1.2) already shows that things are not necessarily as simple as they appear. Some other common systems, especially in Computer Science, are octal (base eight) and hexadecimal (base sixteen). The ancient Sumerians and Babylonians used base sixty, and we see remnants of this in modern timekeeping (sixty minutes in one hour, and sixty seconds in one minute). However, none of this explains why we should care about other bases in the first place though. To understand why binary is important, we need to learn a little about computer hardware. We will present a simplified overview here.

Computer memory is made up of many small capacitors, each of which can take on one of two voltage levels. We often think of these as switches that can be either up or down, or boxes that can either be filled or blank. We call each box a bit,

[†]The number 1728 is interesting. In the theory of elliptic curves, 1728 is the value of the Klein j-invariant at $\tau = i$, corresponding to the lattice $\mathbb{Z} + i\mathbb{Z}$. This case represents elliptic curves with complex multiplication by the Gaussian integers $\mathbb{Z}[i]$, a key example in the theory of modular forms.

and a group of eight such boxes a byte. We have

$$\underbrace{\square}_{1 \text{ bit}} \quad \text{and} \quad \underbrace{\square\blacksquare\blacksquare\blacksquare\square\square\blacksquare\square}_{1 \text{ byte} = 8 \text{ bits}}$$

We can use this system to encode integers as follows:

$$(88)_{10} = (1011000)_2 = \underbrace{\square}_{\text{sign}}\underbrace{\blacksquare\square\blacksquare\blacksquare\square\square\square}_{\text{magnitude}}$$

$$(-88)_{10} = -(1011000)_2 = \underbrace{\blacksquare}_{\text{sign}}\underbrace{\blacksquare\square\blacksquare\blacksquare\square\square\square}_{\text{magnitude}}$$

To move beyond integers, we could use a fixed-point system, which has a fixed number of decimal places. That is,

$$(12.875)_{10} = (1100.111)_2 = \underbrace{\square}_{\text{sign}}\underbrace{\blacksquare\blacksquare\square\square}_{\text{integer}}\underbrace{\blacksquare\blacksquare\blacksquare}_{\text{fraction}}.$$

Note that

$$(12.875)_{10} = \left(\frac{103}{8}\right)_{10} = \left(\frac{1100111}{1000}\right)_2$$

so that dividing by eight in binary works like dividing by one thousand in decimal. In practice, we use floating point rather than fixed point systems. Due to the finite length of the memory, only finite real numbers can be represented exactly. That is, only terminating decimals can be represented exactly.

Definition 1.3. In base ten, every terminating decimal can be written in the form

$$\pm a_m a_{m-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_n$$

where all the digits a_i and b_j are in $\{0, 1, \ldots, 9\}$. Three quantities have to be recorded when representing the number, namely the sign, the digits, and the location of the decimal point.

Example 1.7. We consider a few important numbers encountered in Chemistry and Physics. For example, the melting point of ice in Kelvin is 273.15, Avogadro's number is approximately 6.022×10^{23}, the universal gravitational constant G carries a value of approximately 6.67×10^{-11}.

Note that it is more economical to use scientific notation to denote numbers. As such, using scientific notation, we denote the number

$$\pm a_m a_{m-1} \ldots a_1 a_0 . b_1 b_2 \ldots b_n$$

by

$$\pm a_m a_{m-1} \dots a_1 a_0 . b_1 b_2 \dots b_n \times 10^c.$$

It is required that $a_m \neq 0$. There are three *ingredients* of the scientific notation, which are namely the sign (either $+$ or $-$), the digits $(0, 1, \dots, 9)$, and the exponent e.

When storing a number on a computer, a certain amount of memory is assigned to each part of the scientific notation. Computer arithmetic systems based off of this notation are called floating point systems. We first consider a simple model — we assign one digit to represent the sign, three digits to represent the significant figures, and two digits to represent the exponent.

Example 1.8. For example, numbers that can be exactly represented are

$$3.50, 4.56 \times 10^{99}, -7.98 \times 10^{47}, \dots .$$

On the other hand, numbers that cannot be exactly represented are

$$3.501, 4.562 \times 10^{99}, -7.983 \times 10^{47}, \sqrt{2}, \pi, \sin 1, 4.56 \times 10^{-13}.$$

Note that indeed, 4.56×10^{-13} cannot be exactly represented because we need two digits and a sign to represent the exponent and we have only allocated two digits (implicitly, numbers like 3.50 and 4.56 have a $+$ sign in front of them so one digit is used for storing these signs).

As such, how can we get negative exponents? We discuss two methods.

- **Method 1:** Use one of the two digits to store the sign
- **Method 2:** Since two digits can be used to denote numbers from 0 to 99, we can change the interpretation of these two digits by regarding the exponent as this number minus 49, so that the exponent ranges from -49 to 50.

It turns out that method 2 can represent more possible exponents, and the representation is as follows:

$$(-1)^s \times a_0 a_1 a_2 \times 10^{e_1 e_2 - 49}$$

Here, s denotes the sign, a_0, a_1, a_2 denote the significant figures, and e_1, e_2 denote the exponent. or now, in order to avoid the different representations of the same number, we allow a_0 to be zero only when $e_1 = e_2 = 0$. Later, we will treat this case more carefully as there are some unexpected subtleties. In our simple model, we have the following:

- Smallest positive number: 0.01×10^{-49}
- Greatest negative number: -0.01×10^{-49}
- Second smallest positive number: 0.02×10^{-49}
- Second greatest negative number: -0.02×10^{-49}
- Greatest positive number: 9.99×10^{50}
- Smallest negative number: -9.99×10^{50}
- Second greatest positive number: 9.98×10^{50}
- Second smallest negative number: -9.98×10^{50}

We can represent the above (and some other numbers) using a table as shown.

$$\begin{array}{cccc}
0.00 \times 10^{-49} & 0.01 \times 10^{-49} & \dots & 0.99 \times 10^{-49} \\
1.00 \times 10^{-49} & 1.01 \times 10^{-49} & \dots & 9.99 \times 10^{-49} \\
1.00 \times 10^{-48} & 1.01 \times 10^{-48} & \dots & 9.99 \times 10^{-48} \\
\vdots & \vdots & \ddots & \vdots \\
1.00 \times 10^{49} & 1.01 \times 10^{49} & \dots & 9.99 \times 10^{49} \\
1.00 \times 10^{50} & 1.01 \times 10^{50} & \dots & 9.99 \times 10^{50}
\end{array}$$

Numbers where the first digit is allowed to be zero are said to be *subnormal* or *denormal*. On the other hand, numbers where the first digit is non-zero are called normal.

As mentioned, computers use binary numbers instead of decimal numbers. We give a few more examples.

Example 1.9. We have

$$\begin{aligned}
(10101)_2 &= \left(1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1\right)_{10} \\
&= (16 + 4 + 1)_{10} \\
&= (21)_{10}
\end{aligned}$$

So, the usual 21 in base 10 can be written as 10101 in base 2.

Example 1.10. We have

$$(101.101)_2 = \left(2^2 + 2^0 + 2^{-1} + 2^{-3}\right)_{10} = (5.625)_{10}$$

so the usual 5.625 in base 10 can be written as 101.101 in base 2.

Proposition 1.1 (addition in base 2)**.** Arithmetic with binary numbers is similar to that with decimal numbers. Since binary representations use only 1 and 0, we generally have to *borrow and carry* much more frequently. We have the following results:

(i) $(0)_2 + (0)_2 = (0)_2$

(ii) $(0)_2 + (1)_2 = (1)_2 + (0)_2 = (1)_2$

(iii) $(1)_2 + (1)_2 = (10)_2$ and $(1)_2 + (1)_2 + (1)_2 = (11)_2$

We briefly discuss Proposition 1.1. Note that **(ii)** is clear because this simply talks about the commutativity of addition in base 2, which follows from the commutativity of addition in base 10. For **(iii)**, the key idea is that binary digits work like a counter that *resets* once it reaches 2. Since the only digits are 0 and 1, adding 1 to 1 gives 0 in the current place and carries 1 to the next place (just like how adding 1 to 9 in decimal gives 0 in that place and carries 1).

Example 1.11 (addition in base 2)**.** Find the value of

$$(11.001)_2 + (1.1001)_2 .$$

Solution. We have

$$\begin{array}{r} 11.0010 \\ +\ 1.1001 \\ \hline \end{array}$$

Adding column-wise starting from the right, we have

$$\begin{array}{r} 11.0010 \\ +\ 1.1001 \\ \hline 100.1011 \end{array}$$

so the required value is $(100.1011)_2$. □

We have similar properties for subtraction.

Example 1.12 (subtraction in base 2)**.** Find the value of

$$(11.001)_2 - (10.0011)_2 .$$

Solution. We have

$$\begin{array}{r} 11.0010 \\ -\ 10.0011 \\ \hline \end{array} .$$

Subtracting column-wise starting from the right, we have

$$\begin{array}{r} 11.0010 \\ -\ 10.0011 \\ \hline 0.1111 \end{array}$$

so the required value is $(0.1111)_2$. □

For multiplication, we first ignore the decimal point and multiply the numbers as integers, after which we determine the correct location of the decimal point as normal. We illustrate the method with an example (Example 1.13).

Example 1.13 (multiplication in base 2)**.** We shall prove that

$$(1.0111)_2 \times (10.11)_2 = (11.111101)_2 .$$

As mentioned, a common trick is to ignore the binary points and multiply as if both numbers were integers. Note that $(1.0111)_2$ has 4 fractional bits, whereas $(10.11)_2$ has 2 fractional bits, so we say that the total number of fractional bits after multiplication is $4 + 2 = 6$. Performing ordinary multiplication, we have

$$(10111)_2 \times (1011)_2 = (11111101)_2 .$$

All that is left is to insert the binary point. As mentioned, there should be 6 fractional bits in the product. Starting from the right, we place the binary point 6 places left so we have the map $(11111101)_2 \mapsto (11.111101)_2$.

Alternatively, we see that

$$(1.0111)_2 = \left(2^0 + 2^{-2} + 2^{-3} + 2^{-4}\right)_{10} = \left(\frac{23}{16}\right)_{10}$$

and

$$(10.11)_2 = \left(2^1 + 2^{-1} + 2^{-2}\right)_{10} = \left(\frac{11}{4}\right)_{10} .$$

We have usual multiplication in base 10, so

$$\left(\frac{23}{16}\right)_{10} \cdot \left(\frac{11}{4}\right)_{10} = \left(\frac{253}{64}\right)_{10} .$$

Converting into base 2, the result follows.

1.3 Floating Point Systems and Computing Operations

Scientific notation for binary numbers is similar to the case for decimal numbers, but we must remember that the place values are powers of two now. Consider the number

$$(\pm a_0.b_1b_2\ldots b_n \times 10^e)_2\,.$$

If $e < n$,

$$\begin{aligned}(\pm a_0.b_1b_2\ldots b_n \times 10^e)_2 &= \left(\pm\left(a_0 + b_1\times 2^{-1} + b_2\times 2^{-2} + \ldots + b_n\times 2^{-n}\right)\times 2^e\right)_{10}\\ &= \Big(\pm\Big(a_0\times 2^e + b_1\times 2^{e-1} + b_2\times 2^{e-2} + \ldots\\ &\quad + b_{e-1}\times 2^1 + b_e + b_{e+1}\times 2^{-1} + \ldots + b_n\times 2^{e-n}\Big)\Big)_{10}\\ &= (\pm a_0b_1\ldots b_e.b_{e+1}\ldots b_n)_2\end{aligned}$$

If $e \geq n$, then

$$(\pm a_0.b_1\ldots b_n \times 10^e)_2 = \left(\pm a_0b_1\ldots b_n\underbrace{00\ldots00}_{e-n\ \text{zeros}}\right)_2\,.$$

As before, there are three components of the scientific notation, which are the sign ($+$ or $-$), significant digits $(a_0, b_1, b_2, \ldots, b_n)$, and the exponent e. Note that if we require normalisation (that is, that $a_0 \neq 0$, then we must have $a_0 = 1$). This will impact how we store binary numbers in memory. See this link for an interactive guide on a floating point calculator.

In the computer memory, every binary digit is called a bit. 1 byte is equivalent to 8 bits, 1 kilobyte is equal to 1024 bytes, 1 megabyte is 1024 kilobytes, and 1 gigabyte is 1024 megabytes.

Definition 1.4 (normalised binary number)**.** A normalised binary number is of the form

$$(\pm 1.b_1b_2\ldots b_n \times 10^e)_2 = \pm\,(1.b_1b_2\ldots b_n)_2 \times 2^e.$$

We consider a simple model of a binary system. We have one bit s for the sign (where 0 corresponds to $+$ and 1 corresponds to $-$), three bits for the significand (b_1, b_2, b_3), and two bits e_1, e_2 for the exponent e. For the exponent, we can adopt

the representation

$$00 \mapsto -1 \quad 01 \mapsto 0 \quad 10 \mapsto 1 \quad 11 \mapsto 2.$$

In this small system, we have the following expression for the binary number:

$$\left((-1)^s \times 1.b_1b_2b_3 \times 10^{e_1e_2-1}\right)_2$$

All the positive numbers that can be exactly represented by this format include

$$\begin{array}{cccc} (1.000)_2 \times 2^{-1} & (1.001)_2 \times 2^{-1} & \dots & (1.111)_2 \times 2^{-1} \\ (1.000)_2 \times 2^{0} & (1.001)_2 \times 2^{0} & \dots & (1.111)_2 \times 2^{0} \\ (1.000)_2 \times 2^{1} & (1.001)_2 \times 2^{1} & \dots & (1.111)_2 \times 2^{1} \\ (1.000)_2 \times 2^{2} & (1.001)_2 \times 2^{2} & \dots & (1.111)_2 \times 2^{2} \end{array}$$

Example 1.14. For example, the string $\boxed{0}\boxed{0}\boxed{1}\boxed{1}\boxed{0}\boxed{1}$ represents

$$\left((-1)^0 \cdot 1.101 \times 10^{01-1}\right)_2$$

Also, the string $\boxed{1}\boxed{1}\boxed{1}\boxed{0}\boxed{1}\boxed{1}$ represents

$$\left((-1)^1 \times (1.011) \times 10^{11-1}\right)_2 = (-5.5)_{10}$$

In particular, note that the number zero cannot be represented in this system, and there is a large gap between zero and the smallest positive number. We will address both of these shortcomings in due course.

In order to accommodate zero and other subnormal numbers, we introduce some special rules into our system. Consider the representation

s	e_1	e_2	b_1	b_2	b_3

.

When $e_1 = e_2 = 0$, we change the meaning of the above bits to

$$\left((-1)^s \times 0.b_1b_2b_3 \times 10^{01-1}\right)_2.$$

Note that this is equivalent to

$$\left((-1)^s \times b_1.b_2b_3 \times 10^{00-1}\right)_2.$$

Example 1.15. For example, $\boxed{1}\boxed{0}\boxed{0}\boxed{0}\boxed{1}\boxed{1}$ represents

$$\left((-1)^1 \times (0.011) \times 10^{1-1}\right)_2 = (-0.375)_{10}\,.$$

Example 1.16. Consider the Institute of Electrical and Electronics Engineers (IEEE) double precision format, which has 11 bits for storing the exponent and 52 bits for storing the mantissa. For which positive integers k can the number $5 + 2^{-k}$ be represented (with no rounding error) in binary double precision floating point arithmetic?

Solution. Note that $(5)_{10} = (101)_2$. If $k = 1$, then $2^{-k} = 2^{-1} = 0.1$ in binary. Then, $5 + 2^{-k} = (101.1)_2$, which needs 3 bits to be stored (recall Definition 1.4). If $k = 2$, then $2^{-k} = 2^{-2} = 0.01$ in binary. Then, $5 + 2^{-k} = (101.01)_2$, which needs 4 bits to be stored. Repeat, and one can conclude that $1 \leq k \leq 50$. □

In typical floating point systems, the case when all the exponent bits are equal to 1 is also reserved for special use, which again we shall discuss in due course. Using the notions earlier, we now discuss the accuracy of a floating point representation.

Definition 1.5. Any number p provided to the computer is approximated to the closest representable number, which we denote $fl\,(p)$. We can measure the absolute and relative error of this approximation, which are

$$\text{absolute error} = |p - fl\,(p)| \quad \text{and} \quad \text{relative error} = \frac{|p - fl\,(p)|}{|p|} \tag{1.3}$$

where $p \neq 0$ when computing the relative error.

Example 1.17. For example, if $p = \sqrt{2} \approx 1.414$, then $fl\,(p) = 1.375$, where we note that $(1.375)_{10} = (1.011)_2$. Recall from our earlier discussion that we allocate three bits for the significand. Hence,

$$\text{absolute error} \approx 0.0392 \quad \text{and} \quad \text{relative error} \approx 0.0277,$$

where we used (1.3).

In terms of relative errors, subnormal numbers are less accurate than normal numbers. Pictorally, if $1 < |p| < 2$, we can estimate the relative error can be estimated using

$$\frac{|p - fl\,(p)|}{|p|} \leq \frac{2^{-4}}{1} = 2^{-4}.$$

If $2 < |p| < 3.75$, we can estimate the relative error by

$$\frac{|p - fl\,(p)|}{|p|} \leq \frac{2^{-3}}{2} = 2^{-4}.$$

If $|p| < 1$, the relative error may be larger than 2^{-4}. For this system, the number 2^{-4} is referred to as the machine epsilon, and we will use the symbol $\varepsilon_{\text{mach}}$ to refer to it. Note that we always have

$$\frac{|fl\,(x) - x|}{|x|} \leq \varepsilon_{\text{mach}}.$$

We can compute the machine precision using

$$\varepsilon_{\text{mach}} = \frac{1}{2} \times (\text{base})^{1-\text{significant figures}} = \frac{1}{2} b^{1-n}.$$

In our example, we have

$$\varepsilon_{\text{mach}} = \frac{1}{2} \times 2^{1-4} = \frac{1}{16}.$$

Converting a real number to a float usually involves some rounding. Generally, we would want $fl\,(x)$ to be the float closest to x. For example, in the system described above, we have $fl\,(1.414) = 1.375$ and $fl\,(3.70) = 3.75$.

- What about a number that is equally spaced between two floats like 2.125? What about numbers that are larger than our largest float?

To address the first case, we have to adopt a rounding rule. The simplest rule is called chop rounding, where we simply leave off any bits beyond the number we are able to store. For example,

$$(2.125)_{10} = (10.001)_2 = \left(1.0001 \times 10^1\right)_2.$$

For normal numbers, we only need to store the bits after the decimal point, so under chop rounding, we would store 000 and just forget about the 1 at the end.

In practice, floating point systems usually use the round to even rule. We briefly explain this. It is helpful to think about the circumstances under which a number can be exactly halfway between two floats. This occurs only when the number has one more bit than can be stored, and that bit is a 1. Chop rounding as above, corresponds to always rounding down in this circumstance. Under round to even, we choose to round either up or down to make the final stored bit 0 (0 is even and 1 is odd).

Example 1.18 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Compute the absolute and relative error in the following approximations of x by $\hat{x}$:

(a) $x = e^{10}$ and $\hat{x} = 22000$

(b) $x = 10^{\pi}$ and $\hat{x} = 1400$

(c) $x = 8!$ and $\hat{x} = 39900$

(d) $x = 9!$ and $\hat{x} = \sqrt{18\pi}\left(\frac{9}{e}\right)^9$

Solution. Recall that

$$\text{absolute error} = \hat{x} - x \quad \text{and} \quad \text{relative error} = \frac{\hat{x} - x}{x}.$$

(a) The absolute error is 26.4657948067 and the relative error is 0.00120154522533.

(b) The absolute error is 14.544268633 and the relative error is 0.0104978227046.

(c) The absolute error is 420 and the relative error is 0.0104166666667.

(d) The absolute error is 3343.12715805 and the relative error is 0.00921276223008. □

Example 1.19 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Perform the following computations using three-digit decimal (i.e. base 10) arithmetic with rounding. That is, after any computation, your result can only have three significant digits. In each case, compute the relative errors with the exact value computed to at least five digits.

(a) $133 + 0.921$

(b) $133 - 0.499$

(c) $(121 - 0.327) - 119$

(d) $(121 - 119) - 0.327$

Solution.

(a) Answer is 134, and the relative error is 0.000589900015681.

(b) Answer is 133 and the relative error is 0.0.003766009.

(c) Answer is 2 and the relative error is 0.195457.

(d) Answer is 1.67 and the relative error is -0.00179318. □

Example 1.20 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Recall that the binomial coefficient is defined by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

and is used to determine the number of ways to choose a subset of k objects from a collection of n objects. Suppose you are working with decimal (i.e. base 10)

machine numbers of the form

$$\pm 0.d_1d_2d_3d_4 \times 10^d \quad \text{where } d \leq 15.$$

(a) What is the largest value of n for which $\binom{n}{3}$ can be computed using the definition above without causing overflow?

(b) Show that

$$\binom{n}{k} = \frac{n}{k} \cdot \frac{n-1}{k-1} \cdot \ldots \cdot \frac{n-k+1}{1}.$$

(c) Using this new definition, what is the largest value of n for which $\binom{n}{3}$ can be computed without causing overflow?

Solution.

(a) Note that

$$\pm 0.d_1d_2d_3d_4 \times 10^d = \pm d_1.d_2d_3d_4 \times 10^{d-1} \quad \text{where } d-1 \leq 14.$$

In this machine, the number of largest finite magnitude s 9.999×10^{14}. Since $n! \geq (n-3)!$, without causing overflow in the computation, one sees that $17! = 3.56 \times 10^{14}$ but $18! = 6.40 \times 10^{15}$, so the largest value of n is 17.

(b) On the right, the numerator can be written as

$$\frac{n!}{(n-k)!}$$

whereas the denominator can be written as $k!$. The result follows.

(c) By considering the fraction, we must have

$$\frac{n\,(n-1)\,(n-2)}{6} \leq 9.999 \times 10^{14}$$

One checks computationally that $n = 181710$ is the largest n that does not overflow. □

We now handle the case when numbers are too large. A more general floating point system would look like as follows:

s	e_1	e_2	$\ldots$	e_m	b_1	b_2	$\ldots$	b_{n-1}

Interpreting such strings involves some complicated rules, so we shall state an example of such a system involving the single-precision floating-point format, which has $m = 8$ and $n = 23$.

Example 1.21 (single-precision floating-point format)**.** The size is 32 bits, or 4 bytes. The minimum positive number is

$$\left(0.00\ldots01\times10^{1-1111111}\right)_2 = 2^{-149} \approx 1.40\times10^{-45}.$$

On the other hand, the maximum positive number is

$$\left(1.11\ldots11\times10^{11111110-1111111}\right)_2 = 2^{127}\times\left(2-2^{-23}\right) \approx 3.40\times10^{38}.$$

For normal numbers, the relative error is less than $2^{-24} \approx 5.96\times10^{-8}$. Normal numbers typically have 7 or 8 decimal significant digits.

As mentioned previously, subnormal normal numbers generally have less accuracy than normal numbers. In this system, we have that the greatest subnormal number is $\approx 1.1754942107\times10^{-38}$, and the smallest positive normal number is $\approx 1.1754943508\times10^{-38}$. In a floating point system, the smallest positive normal number is called the underflow level. If L is the smallest (i.e. most negative) exponent, then the underflow is equal to b^L, where b is the base. The largest representable number is called the overflow level. If U is the largest exponent, then the overflow is $b^{U+1}\left(1-b^{-n}\right)$, where b is the base and n is the number of significant digits. When the result of a computation (after rounding) is larger than the overflow level, we say that the computation overflows, and the result is stored in a special way.

The sequence

0	1	1	…	1	0	0	…	0

is interpreted as $+\infty$,

whereas the sequence

1	1	1	…	1	0	0	…	0

is interpreted as $-\infty$.

When computations overflow, they are assigned one of two special values, depending on the situation. In other cases, where the computation cannot be performed at all, we return a different special value instead. For example, the sequence

0	1	1	…	1	b_1	b_2	…	b_{n-1}

means `NaN` (not-a-number) if any of the b_k is non-zero. Some examples that would return `NaN` are $\sqrt{-1}$ and $\arcsin 1.1$. Note that computers cannot do complex

arithmetic, so in a more sophisticated system, $\sqrt{-1}$ would not be a problem. Some examples that would overflow are

$$\frac{1}{+0} = +\infty \quad \text{and} \quad \log(+0) = -\infty.$$

Also, note that there are two representations of zero, which are

0	0	0	...	0	...	0	which corresponds to $+0$ and
1	0	0	...	0	...	0	which corresponds to -0

Let us recap certain concepts. Definition 1.4 tells us that normals look like $(\pm 1.b_1b_2) \times 2^e$. For an all-zero exponent field, we drop the hidden 1 and use

$$(\pm 0.b_1b_2\ldots) \times 2^{(00\ldots01)-\text{bias}}.$$

In floating-point formats, the bias is a constant added to the true (unbiased) exponent so that the stored exponent field is unsigned. Say there are k exponent bits. Then, the bias is

$$2^{k-1} - 1.$$

For example, for a string with 1 sign bit, 3 exponent bits, 4 mantissa bits, the bias is $2^{3-1} - 1 = 3$. The layout is of the form

s	e_2	e_1	e_0	m_1	m_2	m_3	m_4

where each of the m_i's denotes the mantissa. Here are the decoding rules. If $e \neq 000, 111$, the value is

$$(-1)^s \times (1.m_1\ldots m_4)_2 \times 2^{e_2-3}.$$

If $e = 000$, the subnormal is equal to

$$(-1)^s \times (0.m_1\ldots m_4)_2 \times 2^{1-3} = (-1)^s \times (0.m)_2 \times 2^{-2}.$$

Lastly, we consider the case $e = 111$. If $m = 0000$, then we have $\pm\infty$, otherwise, we have `NaN`.

Example 1.22. Suppose you are working in a system that allocates 1 bit for the sign, 3 bits for the exponent, and 4 bits for the mantissa. Assume that normalisation is used and special values are accounted for. What values are represented by the following strings in this system:

01110001 10000001 00010001

Solution. Let the layout be

s	e_2	e_1	e_0	m_1	m_2	m_3	m_4

.

This denotes 1 sign, 3-bit exponent, 4-bit mantissa.

For the first string 01110001, the sign s contributes $(-1)^0$ and the 3-bit exponent is 111. Since there are 3 exponent bits, the bias is $2^{3-1} - 1 = 3$. Since $e = 111$ and m is not 0000, the value represented by the string is `NaN`.

For the second string 10000001, the sign contributes $(-1)^1$ and the 3-bit exponent is 000. Since $e = 000$, then the subnormal is equal to

$$(-1)^1 \times (0.0001)_2 \times 2^{1-3}.$$

We leave finding the value as a simple exercise.

Lastly, for the third string 00010001, the sign contributes $(-1)^0$ and the 3-bit exponent is 001. Then, the value is

$$(-1)^0 \times (1.0001)_2 \times 2^{0-3}.$$

Again, we leave the computation as a simple exercise. □

Example 1.23 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Consider the half-precision floating-point format, which has five bits for the exponent and ten bits for the significand. Find the decimal (i.e. base 10) number represented by the following bits:

(a) `0100110001110000`
(b) `1000001111111111`
(c) `0111110000000000`

Solution.

(a) We have

$$\left((-1)^0 \times 1.0001110000 \times 10^{10011-1111}\right)_2.$$

For the teal-coloured 1111, note that we previously catered 5 bits for the exponent. So, the number 1111 is just a string of 1s with $5 - 1 = 4$ digits. Note that

$$10011 - 1111 = 100$$

which is 4 in base 10. Hence, the answer is

$$\left(2^0 + 2^{-4} + 2^{-5} + 2^{-6}\right) \cdot 2^4 = 17.75.$$

(b) In a similar fashion, one can deduce that the answer is -6.097555×10^{-5} in base 10.

(c) Since the exponent consists of all ones, this is a special value, which is $+\infty$. □

Example 1.24 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** For which positive integers k can the number 9×2^k be represented exactly (with no rounding error) in binary single-precision floating point arithmetic?

Solution. The answer is all such k up to overflow. Binary single-precision stores numbers as $1.f \times 2^E$ with 23 fraction bits and unbiased exponent $E \in [-126, 127]$. Since $9 = (1.001_2) \times 2^3$, then

$$9 \times 2^k = 1.001_2 \times 2^{k+3}$$

which has a finite binary significand (only three fractional bits), so it is represented exactly whenever the exponent fits. We require $k + 3 \leq 127$ so $k \leq 124$. Hence, the exact representability holds for $1 \leq k \leq 124$. □

With all of this in hand, we can consider how our computer performs arithmetic computations. Suppose the single-precision floating-point format is used. Let $x = \frac{5}{7}$ and $y = \frac{1}{3}$, and consider $x + y$. When computers receive the instruction to compute $x + y$, the values of x and y have already changed to $fl\,(x)$ and $fl\,(y)$. We have

$$\begin{aligned} fl\,(x) &= (0.10110110110110110110111)_2 \\ fl\,(y) &= (0.01010101010101010101011)_2 \end{aligned}$$

Then, the value of $fl\,(x) + fl\,(y)$ is

$$(1.00001100001100001100001111)_2\,.$$

However, the computer cannot produce this result because the aforementioned number cannot be represented exactly in single-precision floating-point format — another fl has to be applied to fit the format. That is,

$$fl\,(fl\,(x) + fl\,(y)) = (1.00001100001100001100010)_2\,.$$

One can compute the absolute error, but we leave it as an exercise.

Example 1.25 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Find the smallest positive normalized number and the largest positive subnormal number that can be represented by single-precision floating-point format.

Solution. In single-precision floating point format, we allocate 1 bit for the sign, 8 bits for the exponent with bias $2^8 - 1 = 127$, and 23 bits for the mantissa.

We first find the smallest positive normalized number. The true exponent e is $1 - 127 = -126$, and the fraction is 0. Hence, the smallest positive normalized number is $1.0 \times 2^{-126} = 2^{-126}$, which is approximately 1.18×10^{-38}.

For the largest positive subnormal, the mantissa contains all 1s. Note that the significand is $(0.111\ldots1)_2 = 1 - 2^{-23}$. The largest positive subnormal number is $(1 - 2^{-23}) \cdot 2^{-126}$, which is approximately 1.18×10^{-38}. □

Example 1.26 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Suppose we are working in a binary system with 1 bit allocated to storing the sign, 2 bits allocated to storing the exponent, and 3 bits allocated to storing the mantissa. Assume that special values (e.g., `NaN`, `Inf`, etc.) are not represented in this system.

(a) How many distinct numbers can be represented in this system?

(b) What is the largest positive number that can be represented in this system?

(c) What decimal (i.e. base 10) numbers are represented by the strings `101111` and `011001`?

Solution.

(a) Consider

$$\left((-1)^s \times 1.m_1m_2m_3 \times 10^{e_1e_2-1}\right)_2.$$

We have 2 choices for the sign, 4 choices for the exponent, and 8 choices for the mantissa, so there are $2 \cdot 4 \cdot 8 = 64$ distinct binary strings. However, note that `000000` represents $+0$ whereas `100000` represents -0 — these are the two patterns with all exponent bits `00` and all mantissa bits `000`, and the sign bit decides the sign.

(b) The largest positive number is

$$\left(1.111 \times 10^{11-1}\right)_2 = \left(1.111 \times 10^{10}\right)_2 = \left(2^0 + 2^{-1} + 2^{-2} + 2^{-3}\right) \cdot 2^2 = 7.5$$

in base 10.

(c) For `101111`, we have

$$\left((-1)^1 \times 1.111 \times 10^{01-1}\right)_2 = -1.875.$$

For `011001`, we have

$$\left((-1)^0 \times 1.001 \times 10^{11-1}\right)_2 = \left(2^0 + 2^{-3}\right) \cdot 2^2 = 4.5.$$

□

Example 1.27 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Consider the IEEE single-precision format, which has 8 bits allocated for the exponent and 23 bits allocated for the mantissa.

(a) What is the maximum precision (in bits) a non-zero normalized number can have in this system?
(b) What is the minimum precision (in bits) a non-zero normalized number can have in this system?
(c) What is the maximum precision (in bits) a non-zero subnormal/denormal number can have in this system?
(d) What is the minimum precision (in bits) a non-zero subnormal/denormal number can have in this system?

Solution. We have 1 bit for the sign, 8 bits for the exponent and 23 bits for the mantissa. Note that the bias is $2^8 - 1 = 127$. Normalized numbers are of the form $(-1)^s \times 1.f \times 2^{e-127}$, where the leading 1 is the *hidden bit*. Subnormal numbers are of the form $(-1)^s \times 0.f \times 2^{1-127}$, which has no hidden 1.

(a) 24 bits. We have 23 stored fraction bits and 1 implicit leading bit.
(b) 24 bits. Every normalized number has that implicit leading 1, so you always get a full 24-bit significand, regardless of the exponent or which fraction bits are zero.
(c) 23 bits. There is no hidden 1.
(d) 1 bit. The smallest subnormal has only the least-significant fraction bit set. For example, the bit pattern `0...01` represents 2^{-149} which has just one significant bit. □

Definition 1.6. For simplicity, we define

$$\begin{aligned} x \oplus y &= fl\left(fl\left(x\right) + fl\left(y\right)\right), \\ x \ominus y &= fl\left(fl\left(x\right) - fl\left(y\right)\right), \\ x \otimes y &= fl\left(fl\left(x\right) \times fl\left(y\right)\right), \\ x \oslash y &= fl\left(fl\left(x\right) \div fl\left(y\right)\right). \end{aligned}$$

The operations in Definition 1.6 satisfy some of the familiar properties, namely

$$x \oplus y = y \oplus x \quad x \ominus y = -\left(y \ominus x\right) \quad x \otimes y = y \otimes x.$$

However, in general,

$$(x \oplus y) \oplus z \neq x \oplus (y \oplus z) \quad \text{and} \quad (x \oplus y) \otimes z \neq (x \otimes z) \oplus (y \otimes z)$$

Example 1.28. Let

$$x = 3.14159 \quad y = 2.71828 \quad z = 1000000.$$

Then, the results of $(x \oplus y) \oplus z$ and $x \oplus (y \oplus z)$ are different because

$$(x \oplus y) \oplus z = 1000005.875 \quad \text{and} \quad x \oplus (y \oplus z) = 1000005.8125.$$

Some operations may induce significant numerical error.

Example 1.29. For example, consider the sum of two numbers with significantly different magnitudes. That is,

$$10^6 \oplus 12.3456 = 1.000012375 \times 10^6.$$

Example 1.30. Also, consider subtracting two numbers which are very close to each other. For example,

$$8381.02 \ominus 123.45 \otimes 67.89 = -9.765625 \times 10^{-4}.$$

Example 1.31. We can also consider the product of two small numbers. For example, one recalls Newton's law of gravitation from Physics, which states that the force of attraction between two bodies of masses m and M and radius r apart is given by

$$F = G \cdot \frac{Mm}{r^2}.$$

We have $G = 6.674302 \times 10^{-11}$, $m = 9.109384 \times 10^{-31}$, $M = 2.18732 \times 10^{31}$, and $r = 3.248678 \times 10^{4}$. The exact value is

$$F = G \cdot \frac{mM}{r^2} = 1.260067623482\ldots \times 10^{-18}.$$

The numerical result is

$$(G \otimes m) \otimes M \oslash (r \otimes r) \approx 1.260054153269 \times 10^{-18}$$
$$G \otimes (m \otimes M) \oslash (r \otimes r) \approx 1.260067698352 \times 10^{-18}$$

because

$$G \times m \approx 6.07988 \times 10^{-41} < 1.1754942107 \times 10^{-38}.$$

As such, $G \otimes m$ is represented by a subnormal number.

Example 1.32. Suppose we are working in a floating-point format that has one bit for the sign, four bits for the exponent, and seven bits for the significand/mantissa. In this system, consider the values:

$$w = 0000010111010 \quad x = 0000001001001 \quad y = 0111110000000 \quad z = 0111110011011.$$

What are the results of the following computations? Give your answers both in the format used above and in base 10. (Hint: don't forget about special values!)

(a) $w + x$
(b) $w * x$
(c) $y - x$
(d) $w + z$
(e) $y - y$

Solution. Solution. Note that

$$w = \left(2^0 + 2^{-2} + 2^{-3} + 2^{-4} + 2^{-6}\right) \cdot 2^{-6} = \frac{93}{4096}$$

and

$$x = \left(2^{-1} + 2^{-4} + 2^{-7}\right) \cdot 2^{-6} = \frac{73}{8192}.$$

$y = +\infty$ and $z =$ `NaN`. w is normal since the exponent is strictly between 0 and 1111, x is subnormal since the exponent is 0000.

(a) We normalise to obtain

$$\left(1 + \frac{3}{256}\right) 2^{-5}$$

since $\frac{3}{256}$ is in between $\frac{1}{128}$ and $\frac{2}{128}$. The answer is 0001000000010, which in base 10 is 0.03173828125.

(b) The exact value is

$$\frac{93}{4096} \cdot \frac{73}{8192} = \frac{6789}{33554432}.$$

This lies on the subnormal range, so the result is 0000000000010, which is

$$\frac{2}{8192} = 2^{-12} = 0.000244140625.$$

(c) Since $y = \infty$ and x is finite, then $y - x = \infty$. So, the result is 0111110000000, which represents positive infinity.

(d) Since w is finite and $z = \texttt{NaN}$, as any operation with `NaN` yields `NaN`, the result is also `NaN`. In particular, we would obtain 0111110011011.

(e) Each y is ∞ and as $\infty - \infty = \texttt{NaN}$, the result is also `NaN`. □

□

We then discuss counting the number of operations for an algorithm. When we count floating-point operations, we usually only count arithmetic operations (additions, subtractions, multiplications, divisions), not assignments, indexing, or loop overhead. The idea is to measure the numerical work needed.

Example 1.33 (DSA2102 AY25/26 Sem 1 Tutorial 1)**.** Recall that the mean of a vector x is defined by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

and the variance of that vector is defined by

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

How many floating-point operations does it take to compute the variance of a vector with n components?

Solution. We first compute the mean, which requires $n-1$ additions and 1 division, so it requires n floating-point operations. To compute the variance, each $x_i - \overline{x}$ requires n subtractions. By taking the sum of squares (which is effectively multiplication), it requires n multiplications and $n-1$ additions Dividing by $n-1$ requires 1 subtraction and 1 division. Hence, we require

$$(n-1) + 1 + n + n + (n-1) + 1 + 1 = 4n + 1$$

operations in total. □

Example 1.34 (DSA2102 AY25/26 Sem 1 Homework 1)**.** The variance is a commonly encountered statistic in data science. For a data set $\{x_1, \ldots, x_n\}$ it can be computed using the *two-pass* algorithm by first computing the mean

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i,$$

and then using

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

In some situations, however, the data set is too large to keep all of it in memory, or data is being collected dynamically, so we don't have the entire data set to begin with. In these scenarios, a one-pass algorithm is preferred, since it only needs to read each data point once. Below is an example of a one-pass formula for the variance

$$\sigma^2 = \frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right].$$

Compare the number of arithmetic operations required by these two approaches.

Solution. For the two-pass algorithm, we first compute the mean. Summing all x_i requires $n-1$ additions. Dividing by n requires 1 division. For the sum of squared deviations, computing $x_i - \overline{x}$ requires n subtractions. Squaring requires n multiplications. We then take the sum of all $(x_i - \overline{x})^2$, then divide by $n-1$, which requires $n-1$ additions and 1 division. Hence, the total number of operations is

$$n - 1 + 1 + n + n + n - 1 + 1 = 4n.$$

As for the one-pass algorithm, squaring each x_i requires n multiplications. Summing x_i and x_i^2 require a total of $2\,(n-1)$ additions, squaring $\sum x_i$ requires 1 multiplication, dividing by n requires 1 division. Subtracting from $\sum x_i^2$ requires 1 subtraction, and lastly, dividing by $n-1$ requires 1 division. Hence, the total number of operations is

$$n + 2\,(n-1) + 1 + 1 + 1 + 1 = 3n + 2.$$

Both algorithms are of $\mathcal{O}(n)$ complexity, meaning to say that they can be executed in linear time. However, the one-pass formula uses $n-2$ fewer total operations. □

Chapter 2

Systems of Linear Equations

2.1 Matrix and Vector Operations

Usually, Mathematicians look for ways to reformulate a problem from another field as one in Linear Algbera as we have many tools to help us solve Linear Algebra problems. One should be familiar with concepts from MA2001 Linear Algebra. Since our goal is to understand algorithms, say we wish to examine the algorithm to compute a single entry of a matrix product. Consider the matrices $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$, $\mathbf{B} \in \mathcal{M}_{n\times p}(\mathbb{R})$, and $\mathbf{C} \in \mathcal{M}_{m\times p}(\mathbb{R})$. Here, we let

$\mathcal{M}_{n\times n}(\mathbb{R})$ denote the set of $m \times n$ matrices with real-valued entries.

Suppose $\mathbf{C} = \mathbf{AB}$. Then,

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \quad \text{for all } 1 \le i \le m \text{ and } 1 \le j \le p. \tag{2.1}$$

One can prove that the total number of multiplications required is mnp and the total number of additions required is $m(n-1)p$. Note that if $\mathbf{A}, \mathbf{B}, \mathbf{C}$ are square matrices, then $m = n = p$, so the computational complexity is $\mathcal{O}(n^3)$[†] (this refers to the big O notation, which we will briefly introduce in Definition 2.1). So, just like what we mentioned about setting $m = n = p$, if our problem has some special structure, we can exploit it to obtain a better algorithm.

A common problem in Linear Algebra is to multiply an arbitrary matrix by an upper triangular matrix (or lower triangular). Suppose $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$ and $\mathbf{B} \in \mathcal{M}_{n\times n}(\mathbb{R})$, where $\mathbf{B}$ is upper triangular. Then, recall from (2.1) that the matrix entries of the product $\mathbf{AB}$ is

$$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj} \quad \text{for all } 1 \le i \le m \text{ and } 1 \le j \le n. \tag{2.2}$$

Using this, we would end up multiplying by 0 many times, so the computational complexity should intuitively be better than $\mathcal{O}(n^3)$. However, this is not always

[†]It is conjectured (but not proven that the best computational complexity for matrix multiplication is $\mathcal{O}(n^2)$.

true since a computer does not check what a number is before performing arithmetic. That is to say, operations like adding 0, multiplying by 1 or multiplying by 0 still take up the same computational resources as any other addition or multiplication. As such, the person who designs the algorithm has to account for these cases to improve efficiency.

We lay out the entries of an upper triangular matrix $\mathbf{B} \in \mathcal{M}_{n\times n}(\mathbb{R})$. Then,

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ 0 & b_{22} & \dots & b_{2n} \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & b_{nn} \end{pmatrix} \quad \text{so} \quad b_{kj} \begin{cases} \neq 0 & \text{if } k \leq j; \\ = 0 & \text{if } k > j. \end{cases}$$

As such, our computation can be made simpler. Replacing n with j in (2.2), we have

$$c_{ij} = \sum_{k=1}^{j} a_{ik} b_{kj} \quad \text{for all } 1 \leq i \leq m \text{ and } 1 \leq j \leq n.$$

One can prove that

$$\begin{aligned} \text{the total number of multiplications} \quad &\text{is} \quad \frac{mn\,(n+1)}{2} \quad \text{and} \\ \text{the total number of additions} \quad &\text{is} \quad \frac{mn\,(n-1)}{2} \end{aligned}$$

One would find the formula for the sum to n terms of an arithmetic series useful. That is,

$$1 + 2 + \ldots + n = \frac{n\,(n+1)}{2}. \tag{2.3}$$

As expected, we can consider other interesting setups. Now, suppose both $\mathbf{A}$ and $\mathbf{B}$ are $n \times n$ lower triangular matrices. One should be convinced that

$$a_{ij} \begin{cases} \neq 0 & \text{if } i \geq k; \\ = 0 & \text{if } i < k \end{cases}.$$

A similar fact can be said for b_{kj}. Again, to save computational cost, we identify the zero entries in the matrix product and skip the computation of these entries. Similarly, for non-zero entries, avoid adding zero terms in the sum c_{ij}. Since we

are supposed to compute the sum

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \text{for } 1 \leq i \leq n \text{ and } 1 \leq j \leq i$$

where $a_{ik} \neq 0$ if $i \geq k$ and $b_{kj} \neq 0$ if $k \geq j$, then

$$c_{ij} = \sum_{k=j}^{i} a_{ik} b_{kj} \quad \text{for } 1 \leq i \leq n \text{ and } 1 \leq j \leq i.$$

One can prove that

$$\begin{aligned} \text{the total number of multiplications} \quad &\text{is} \quad \frac{n\,(n+1)\,(n+2)}{6} \quad \text{and} \\ \text{the total number of additions} \quad &\text{is} \quad \frac{n\,(n-1)\,(n+1)}{6} \end{aligned}$$

where the formula for the sum of squares of the first n positive integers is useful. As this appears somewhat more interesting than the sum of the first n positive integers (2.3), we present the former as a theorem (Theorem 2.1).

Theorem 2.1 (sum of squares of first n positive integers)**.** We have

$$1^2 + 2^2 + \ldots + n^2 = \frac{n\,(n+1)\,(2n+1)}{6}.$$

Example 2.1 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Let $\mathbf{A} = (a_{ij})_{n\times n}$ be a lower triangular matrix and $\mathbf{B} = (b_{ij})_{n\times n}$ be an upper triangular matrix. Determine the number of floating point operations needed to compute $\mathbf{C} = \mathbf{AB}$.

Solution. By definition, a lower triangular matrix $\mathbf{A} = (a_{ij})_{n\times n}$ has all entries above the main diagonal equal to zero, i.e. $a_{ij} = 0$ whenever $i < j$. Its general form is

$$\mathbf{A} = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}.$$

On the other hand, an upper triangular matrix $\mathbf{B} = (b_{ij})_{n\times n}$ has all entries below the main diagonal equal to zero, i.e. $b_{ij} = 0$ whenever $i > j$. Its general form is

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ 0 & b_{22} & b_{23} & \cdots & b_{2n} \\ 0 & 0 & b_{33} & \cdots & b_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_{nn} \end{pmatrix}.$$

We then compute

$$\mathbf{C} = \begin{pmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ a_{31} & a_{32} & a_{33} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} & \cdots & b_{1n} \\ 0 & b_{22} & b_{23} & \cdots & b_{2n} \\ 0 & 0 & b_{33} & \cdots & b_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & b_{nn} \end{pmatrix}.$$

Note that c_{11} is obtained by considering the first row of $\mathbf{A}$ and the first column of $\mathbf{B}$, which requires 1 multiplication. Next, c_{12} is obtained by considering the first row of $\mathbf{A}$ and the second column of $\mathbf{B}$, which requires 1 multiplication (b_{22} is *paired* with a 0 in $\mathbf{A}$). We repeat this up to c_{1n}, which requires 1 multiplication. The number of operations is

$$n.$$

To compute c_{21}, note that a_{22} is *paired* with a 0 in $\mathbf{B}$, so we only need 1 multiplication. Next, c_{22} requires 2 multiplications and 1 addition. We repeat this up to c_{2n}, which also requires 2 multiplications and 1 addition. The number of operations is

$$1 + 2\,(n-1) + n - 1.$$

To compute c_{31}, we need 1 multiplication. To compute c_{32}, we need 2 multiplications and 1 addition. To compute c_{33}, we need 3 multiplications and 2 additions. We repeat this, and so, to compute c_{3n}, we need 3 multiplications and 2 additions. The number of operations is

$$1 + 2 + 3\,(n-2) + 1 + 2\,(n-2).$$

In general, column j of $\mathbf{B}$ has non-zeros only in rows $1, \ldots, j$ and row i of $\mathbf{A}$ has non-zero rows only in columns $1, \ldots, i$. Repeating the above setup, we see that the

total number of multiplications is

$$n+[1+2\,(n-1)]+[1+2+3\,(n-2)]+\ldots+[1+2+\ldots+n-1+n\,(n-(n-1))]$$

which evaluates to

$$n+2\,(n-1)+3\,(n-2)+\ldots+n\,(1)+1+(1+2)+\ldots+(1+2+\ldots+n-1).$$

Note that

$$\begin{aligned} n+2\,(n-1)+3\,(n-2)+\ldots+n\,(1) &= \sum_{k=1}^{n} k\,(n-k\,(n-1)) \\ &= n\sum_{k=1}^{n} k-(n-1)\sum_{k=1}^{n} k^2 \\ &= \frac{n\,(n+1)\,(n+2)}{6} \end{aligned}$$

and

$$\begin{aligned} 1+(1+2)+\ldots+(1+2+\ldots+n-1) &= 1+3+5+\ldots+\frac{n\,(n-1)}{2} \\ &= \frac{n\,(n-1)\,(n+1)}{6} \end{aligned}$$

The total number of additions is

$$(n-1)+[1+2\,(n-2)]+[1+2+3\,(n-2)]+\ldots+[1+2+\ldots+n-2+n\,(n-1)]$$

which evaluates to

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

Hence, the total number of operations is

$$\frac{n\,(n+1)\,(n+2)}{6}+\frac{n\,(n-1)\,(n+1)}{6}+\frac{n\,(n+1)\,(2n+1)}{6}-n^2=\frac{2n^3+n}{3}.$$

□

Example 2.2 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Repeat Example 2.1 where $\mathbf{A}$ is an arbitrary $n\times n$ matrix and $\mathbf{B}$ is an $n\times n$ tridiagonal matrix, i.e., $b_{ij}=0$ if $|i-j|>1$.

Solution. For an arbitrary square matrix $\mathbf{A}$, we have

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}.$$

Next, the general form of a tridiagonal matrix $\mathbf{B}$ is

$$\mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & 0 & \cdots & 0 & 0 \\ b_{21} & b_{22} & b_{23} & \cdots & 0 & 0 \\ 0 & b_{32} & b_{33} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & b_{n-1,n-1} & b_{n-1,n} \\ 0 & 0 & 0 & \cdots & b_{n,n-1} & b_{nn} \end{pmatrix}.$$

We wish to compute the matrix product $\mathbf{C} = \mathbf{AB}$. Note that the first and last columns of $\mathbf{B}$ have 2 non-zero entries, whereas the rest have 3 non-zero entries. Thus, to compute each entry in the first and last columns of $\mathbf{C}$, we need 2 multiplications and 1 addition, while for the entries in the other columns, we need 3 multiplications and 2 additions.

Hence, the total operation count is $3n$ for the first column, $3n$ for the last column, and $5n$ for each of the other columns. This gives a total of $3n \cdot 2 + 5n\,(n-2) = 5n^2 - 4n$ operations. □

At this juncture, we also give a definition for the big O notation (Definition 2.1).

Definition 2.1 (big O notation)**.** Let $f(n)$ and $g(n)$ be functions. We say that $f(n) = \mathcal{O}(g(n))$ if there exists exists $M \in \mathbb{R}^+$ and $N \in \mathbb{R}$ such that

$$|f(n)| \leq Mg(n) \quad \text{for all } n \geq N.$$

Example 2.3 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Given two matrices $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$, $\mathbf{B} \in \mathcal{M}_{n\times p}(\mathbb{R})$, and a vector $\mathbf{x} \in \mathbb{R}^p$. To compute $\mathbf{ABx}$, which of the following sequences of computation is better: $(\mathbf{AB})\,\mathbf{x}$ or $\mathbf{A}\,(\mathbf{Bx})$? Justify your answer.

Solution. To compute $\mathbf{AB}$, we need n multiplications and $n-1$ additions when perform matrix multiplication with the first row of $\mathbf{A}$ and the first column of $\mathbf{B}$. This yields the $(1,1)$-entry of $\mathbf{AB}$. To obtain all entries of $\mathbf{AB}$, we repeat the process mp times since $\mathbf{AB} \in \mathcal{M}_{m\times p}(\mathbb{R})$. We then compute $(\mathbf{AB})\,\mathbf{x}$. First, we need p multiplications and $p-1$ additions when performing matrix multiplication with the first (and only column) of $\mathbf{x}$. Repeat the process m times since $\mathbf{AB} \in \mathcal{M}_{m\times p}(\mathbb{R}) \in \mathbb{R}^m$. The total number of operations required is

$$mp\,(n+n-1) + m\,(p+p-1) = 2mnp - m.$$

We then perform the count for $\mathbf{A}\,(\mathbf{Bx})$. First, compute $\mathbf{Bx}$ For the first entry, we need p multiplications and $p-1$ additions. Repeat for all n entries so we need np multiplications and $n(p-1)$ additions. Then, compute $\mathbf{A}(\mathbf{Bx})$. For the first entry, we need n multiplications and $n-1$ additions. Repeat for all m entries so we need mn multiplications and $m(n-1)$ additions. The total number of operations required is

$$np + mn + n\,(p-1) + m\,(n-1) = 2mn + 2np - m - n.$$

For typical dimensions, especially when $p > 1$, $\mathbf{A}(\mathbf{Bx})$ is far cheaper — it is $\mathcal{O}(np+mn)$ instead of $\mathcal{O}(mnp)$. It also avoids storing the whole $m \times p$ matrix $\mathbf{AB}$. □

Example 2.4 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** A matrix $\mathbf{M}$ is called *banded* with bandwidth w if $m_{ij} = 0$ whenever $|i-j| > w$. For example, a banded matrix with bandwidth 1 is tridiagonal, like the matrix below.

$$\begin{pmatrix} a_{11} & a_{12} & 0 & \dots & 0 & 0 \\ a_{21} & a_{22} & a_{23} & \dots & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & \dots & 0 & 0 & a_{n,n-1} & a_{nn} \end{pmatrix}$$

(a) Suppose $\mathbf{M}$ is $n \times n$ matrix with bandwidth $w = 1$. How many floating point operations are required to solve the system $\mathbf{Mx} = \mathbf{b}$?[†]

[†]This is essentially Thomas' algorithm.

(b) Suppose $\mathbf{M}$ is an $n \times n$ matrix with bandwidth $1 \leq w < n$. How many floating point operations are required to solve the system $\mathbf{Mx} = \mathbf{b}$? Your answer should be in terms of w and n.

In **(a)**, for example, consider the tridiagonal matrix

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ a_{21} & a_{22} & a_{23} \\ 0 & a_{32} & a_{33} \end{pmatrix}.$$

Let the i^{th} subdiagonal be ℓ_i, i^{th} diagonal be d_i, and the i^{th} superdiagonal be u_i. For $i = 2$, we have

$$m = \frac{\ell_2}{d_1} = \frac{a_{21}}{a_{11}}.$$

We then replace d_2 with

$$d_2 - mu_1 = a_{22} - \frac{a_{21}}{a_{11}} \cdot a_{12}.$$

Then, replace b_2 with

$$b_2 - mb_1 = b_2 - \frac{a_{21}}{a_{11}} \cdot b_1.$$

So, the matrix now becomes

$$\begin{pmatrix} a_{11} & a_{12} & 0 \\ 0 & a_{22} - \frac{a_{21}}{a_{11}} \cdot a_{12} & a_{23} \\ 0 & a_{32} & a_{33} \end{pmatrix}.$$

We then repeat the process.

As for **(b)**, suppose $n = 7$ and $w = 2$. Take

$$\mathbf{M} = \begin{pmatrix} 10 & -1 & -2 & 0 & 0 & 0 & 0 \\ -1 & 10 & -1 & -2 & 0 & 0 & 0 \\ -2 & -1 & 10 & -1 & -2 & 0 & 0 \\ 0 & -2 & -1 & 10 & -1 & -2 & 0 \\ 0 & 0 & -2 & -1 & 10 & -1 & -2 \\ 0 & 0 & 0 & -2 & -1 & 10 & -1 \\ 0 & 0 & 0 & 0 & -2 & -1 & 10 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{pmatrix}.$$

This has non-zeros only on the main diagonal, the first ± 1 diagonals, and the second ± 2 diagonals, so $w = 2$.

We first proceed with forward elimination. At pivot row k, we eliminate the entries directly below the pivot in rows $k+1, \ldots, k+w$ (if they exist). For each eliminated row, we only update the w entries to the right of the pivot, i.e. columns $k+1, \ldots, k+w$). Everything else is already zero and stays zero. We will only work with the first two pivots in detail as the rest will repeat the same pattern

For pivot $k = 1$, we need to eliminate rows 2 and 3. The general formula is $m_k = \min\{w, n-k\}$. The multipliers are

$$m_{21} = \frac{a_{21}}{a_{11}} = \frac{-1}{10} = -0.1 \quad \text{and} \quad m_{31} = \frac{a_{31}}{a_{11}} = \frac{-2}{10} = -0.2.$$

Here are the updates for each eliminated row. By construction, we have zeros below $a_{11} = 10$ in the first-column entry. We update only columns 2 and 3 (the band to the right of the pivot). For $i = 2, 3$, we replace row i with

$$i - m_{i1} \cdot \text{row 1 restricted to columns 2 and 3.}$$

The coefficient matrix now becomes

$$\begin{pmatrix} 10 & -1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 9.9 & -1.2 & -2 & 0 & 0 & 0 \\ 0 & -1.2 & 9.6 & -1 & -2 & 0 & 0 \\ 0 & -2 & -1 & 10 & -1 & -2 & 0 \\ 0 & 0 & -2 & -1 & 10 & -1 & -2 \\ 0 & 0 & 0 & -2 & -1 & 10 & -1 \\ 0 & 0 & 0 & 0 & -2 & -1 & 10 \end{pmatrix}.$$

We then update the right side $\mathbf{b}$. So, we replace b_i with $b_i - m_{i1}b_1$. That is,

$$(1, 2, 3, 4, 5, 6, 7) \mapsto (1, 2.1, 3.2, 4, 5, 6, 7)\,.$$

For every eliminated row in the coefficient matrix, we have 1 division and 2 multiplications and 2 subtractions. As for the vector $\mathbf{b}$, we have 1 multiplication and 1 subtraction. As there are 2 rows, we have

$$2 \cdot (1 + 2 + 2 + 1 + 1) = 14$$

floating point operations.

For the pivot $k = 2$, we wish to eliminate rows 3 and 4. By a formula suggested earlier, this is just $m_2 = \min\{2, 5\} = 2$ so indeed, we need to eliminate rows 3 and 4. The multipliers are

$$m_{32} = \frac{a_{32}}{a_{22}} \quad \text{and} \quad m_{42} = \frac{a_{42}}{a_{22}}.$$

We only update columns and 3 (which are to the right of pivot column 2), as well as the column vector $\mathbf{b}$. Again, one can check that the number of floating point operations is 14. One can continue this pattern and see that for pivots $k = 3, 4, 5$, each case would constitute 14 floating point operations. However, for the $k = 6$ case, only the 7th row is left. Hence, now $m_6 = 1$ (check using the formula $m_k = \min\{w, n - k\}$ where we set $k = 6$, $w = 2$ and $n = 7$). As we only touch column 7, we only have 5 floating point operations.

Thus, with $n = 7$ and $w = 2$, the number of operations required for forward elimination is $14 \cdot 5 + 5$.

We then perform backward substitution. Right before back-substitution the system is in upper banded triangular form: everything below the main diagonal is zero, and only the main diagonal plus the next w superdiagonals may be non-zero. The right side has been updated to a new vector $\widetilde{\mathbf{b}}$. For our concrete example, we obtain

$$\widetilde{\mathbf{M}} = \begin{pmatrix} 10 & -1 & -2 & 0 & 0 & 0 & 0 \\ 0 & 9.9 & -1.2 & -2 & 0 & 0 & 0 \\ 0 & 0 & 9.454545 & -1.242424 & -2 & 0 & 0 \\ 0 & 0 & 0 & 9.432692 & -1.262821 & -2 & 0 \\ 0 & 0 & 0 & 0 & 9.407860 & -1.267754 & -2 \\ 0 & 0 & 0 & 0 & 0 & 9.405107 & -1.269510 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9.403464 \end{pmatrix}$$

and

$$\widetilde{\mathbf{b}} = \begin{pmatrix} 1 \\ 2.1 \\ 3.454545 \\ 4.878205 \\ 6.383849 \\ 7.894573 \\ 9.422747 \end{pmatrix}.$$

The idea is on row i, we only subtract off products with the at most w known x's to the right (those are the only non-zeros), then divide by the diagonal. Starting with the bottom-most row (row 7), we have $x_7 = \frac{b_7}{a_{77}}$ which corresponds to 1 division. Then, x_6 is replaced with

$$\frac{b_6 - a_{67}x_7}{a_{66}}$$

which corresponds to 1 multiplication, 1 subtraction, and 1 division. As for row 5, we replace x_5 with

$$\frac{b_5 - a_{56}x_6 - a_{57}x_7}{a_{55}}$$

which corresponds to 2 multiplications, 2 subtractions, and 1 division. Repeat this from row 4 up to row 1, where each of these rows consitutes 5 floating point operations. Hence, the total number of floating point operations is

$$(5 + 14 \cdot 5) + (1 + 3 + 5 \cdot 5) = 104.$$

Solution.

(a) Let the i^{th} subdiagonal be ℓ_i, i^{th} diagonal be d_i, and the i^{th} superdiagonal be u_i. We procee with forward elimination. For $2 \leq i \leq n$, define

$$m = \frac{\ell_i}{d_{i-1}}$$

so we have 1 division. We then replace d_i with $d_i - mu_{i-1}$, which consists of 1 multiplication and 1 subtraction. Next, replace b_i with $b_i - mb_{i-1}$, which consists of 1 multiplication and 1 subtraction. For each row, we have 5 floating operations, so we have $5\,(n-1)$ operations for rows $2, \ldots, n$.

We proceed with backward substitution. Let $x_n = \frac{b_n}{d_n}$, which corresponds to 1 division. For $i = n-1, \ldots, 1$, let

$$x_i = \frac{b_i - u_i x_{i+1}}{d_i}$$

which corresponds to 1 multiplication, 1 subtraction, and 1 division. So, we have a total of $3\,(n-1)$ operations. In total, we would expect

$$5\,(n-1) + [1 + 3\,(n-1)] = 8n - 7$$

operations.

(b) We then discuss the number of floating point operations required for an arbitrary matrix of banwidth w. We say that a matrix $\mathbf{M} = (m_{ij})_{n\times n}$ has bandwidth w if all entries outside the diagonal band vanish. So, the main diagonal is always included, and the w superdiagonals (above) and w subdiagonals (below) may contain non-zero entries. Hence, the matrix has at most $2w + 1$ potentially non-zero diagonals.

For example, for a 6×6 matrix of bandwidth 2,[†] we have

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} \\ 0 & 0 & a_{53} & a_{54} & a_{55} & a_{56} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} \end{pmatrix}.$$

We first proceed with forward elimination for a general banded matrix $\mathbf{M}$ of bandwidth w. At pivot row k, we only touch the entries inside the band. Everything farther than w columns to the right is already zero and stays zero. We first ask:

How many rows below pivot do you eliminate?

The answer is up to the next w rows unless we are near the bottom. Define $m_k = \min\{w, n - k\}$. We discuss what we need to do for each of those m_k rows. First, compute the multiplier $\frac{a_{ik}}{a_{kk}}$ which corresponds to 1 division. We then update the block of entries to the right of the pivot that lie in the band. The pivot row has at most w non-zeros to the right, so that is w columns. Each updated entry costs 1 multiplication and 1 subtraction, so 2 operations per entry. Since we have m_k rows to deal with, we have $2m_k$ operations.

For the matrix $\mathbf{b}$, we update each b_i via 1 multiplication and 1 subtraction, which corresponds to 2 operations. So per eliminated row, we have $1 + 2m_k + 2$ floating point operations. Since there are m_k such rows, at pivot k, you spend

$$m_k\,(1 + 2m_k + 2) = m_k\,(2m_k + 3)$$

[†]Such a matrix is called *pentadiagonal*.

floating point operations. Now, do this for $k = 1, 2, \ldots, n-1$.

We shall spot the pattern. For the first $n - w$ pivots, we can eliminate a full w rows. The cost per pivot is $w\,(2w+3)$. We repeat this $n - w$ times. Near the bottom, m_k decreases. Those tail costs are $r\,(2r+3)$ for $r = 1$ to $r = w - 1$. Hence, the forward elimination total is

$$\sum_{k=1}^{n-1} m_k\,(2m_k + 3) = (n-w) \cdot w \cdot (2w+3) + \sum_{r=1}^{w-1} r\,(2r+3)$$

which in closed form is

$$(n-w)\left(2w^2 + 3w\right) + \frac{(w-1) \cdot w \cdot (4w+7)}{6}.$$

Lastly, we shall proceed with backward substitution. For x_i, we subtract off the dot-product with the at-most w known x's to the right, then divide by the diagonal. Define $t_i = \min\{w, n-i\}$. The cost for row i is $2t_i$. For the first $n - w$ rows from the top, as $t_i = w$, the cost is $2w + 1$ repeated $n - w$ times. In the last w rows, t_i runs along $w - 1, w - 2, \ldots, 0$ which costs $2r + 1$ for $r = 0$ to $r = w - 1$. So, the total cost for backward substitution is

$$\sum_{i=1}^{n} (2t_i + 1) = (n-w)\,(2w+1) + \sum_{r=0}^{w-1} (2r+1) = (n-w)\,(2w+1) + w^2.$$

Putting everything together, the total number of floating point operations is

$$(n-w)\left(2w^2 + 5w + 1\right) + \frac{(w-1) \cdot w \cdot (4w+7)}{6} + w^2.$$

For a sanity check, when $w = 1$ (which corresponds to the tridiagonal matrix case in **(a)**), the number of floating point operations is $8n - 7$, matching Thomas' algorithm. For the bigger picture, for some fixed $w \leq n$, the computational complexity is $\mathcal{O}\,(nw^2)$. □

2.2 Systems of Linear Equations

Recall from MA2001 the concept of solving a system of linear equations. In fact, this is one of the most fundamental problems in Applied Mathematics. In due course, we would see that many problems (in particular, involving numerical integration) can either be reduced to solving a system or involve solving a system

along the way.

Generally, a system of linear equations is given by a compact notation. That is,

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

where $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$, $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{b} = (b_1, \ldots, b_m)$. In general, if $m > n$ and the solution to the system exists and is unique, then only n equations are required to determine the solution.

Of interest is to really study how a small change in inputs affects the outputs. For example, say we consider the following system of equations

$$\begin{cases} x + y = 2 \\ x - y = 0 \end{cases}$$

which has solution $x = 1$ and $y = 1$. If we change $\mathbf{b}$ from $(2, 0)$ to $(2.01, 0)$, then the solution changes slightly — $x = 1.005$ and $y = 0.995$. On the other hand, if we consider another system of equations as follows

$$\begin{cases} x + y = 2 \\ 1.000001x + y = 2 \end{cases}$$

we see that the coefficient matrix is *nearly singular*, i.e. $\det(\mathbf{A})$ is very close to 0. As such, tiny changes in $\mathbf{b}$ may result in significant changes in x and y. As such, we wish to study the concept of continuous dependence rigorously.

For a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, we say that the output vector is $\mathbf{x}$, whereas the inputs are $\mathbf{b}$ and $\mathbf{A}$. We will consider each input separately. To proceed with our discussion, we need to figure out how to measure the *size* of $\mathbf{x}, \mathbf{b}, \mathbf{A}$.

Recall from MA2001 that the Euclidean norm of any vector $\mathbf{x} \in \mathbb{R}^n$ is

$$\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{y} \rangle} = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

Here, we used the notion of an inner product, which is a generalisation of the dot product of two vectors. In particular, in $\mathbb{R}^n$, we have $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \cdot \mathbf{y}$, where $\cdot$ denotes

the usual dot product. Several other norms are commonly used in practice[†]. For example, we have the 1-norm (can also be denoted by ℓ^1-norm or the taxicab norm or the Manhattan norm) defined by

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

and the infinity norm

$$\|\mathbf{x}\|_\infty = \max_i |x_i| \,. \tag{2.4}$$

These are examples of p-norms

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{1/p} .$$

As for the infinity norm (2.4), it is defined to be the following limit:

$$\lim_{p\to\infty} \|\mathbf{x}\|_p = \|\mathbf{x}|\,|_\infty$$

There is a nice way to visualise these p-norms. Please refer to this article by B. Chivers.

Here is a different perspective on linear systems. We can think of $\mathbf{A}$ as a map from $\mathbb{R}^n$ to $\mathbb{R}^n$ (think of it as the matrix representation of a linear transformation $T : \mathbb{R}^n \to \mathbb{R}^n$, and recall from MA2001 that indeed, the matrix is $\in \mathcal{M}_{n\times n}(\mathbb{R})$). We can represent all this information using a commutative diagram as follows (you may ignore it if you wish, but it is attached for completeness). What it says is that $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^n$, and under the map $\mathbb{R}^n \to \mathbb{R}^n$ with matrix representation $\mathbf{A}$, it sends $\mathbf{x}$ to $\mathbf{Ax} = \mathbf{b}$, making the following diagram commute:

$$\begin{array}{ccc} \mathbb{R}^n & \xrightarrow{\quad\mathbf{A}\quad} & \mathbb{R}^n \\ \ni \downarrow & & \downarrow \ni \\ \mathbf{x} & \longmapsto & \mathbf{Ax} = \mathbf{b} \end{array}$$

At this juncture, one might ask what is a good way to measure the *size* of a function. Well, for linear functions like $4x$ and $-6x$, we can take the size to be the

[†]Will encounter in courses like MA3209 Metric and Topological Spaces, MA3210 Mathematical Analysis II, MA4211 Functional Analysis, MA4262 Measure and Integration, etc.

absolute value of the coefficient of the term in x (so 4 and 6 respectively), but this feels like a *lame* and non-rigorous way of doing things. How can we make sense of the term 'size'?

Definition 2.2 (matrix norm)**.** Consider the linear system $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$. For linear functions $\mathbb{R}^n \to \mathbb{R}^n$, define the matrix norm as follows:

$$\max_{\mathbf{x}\neq 0} \frac{\|\mathbf{Ax}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|\mathbf{Ax}\| \tag{2.5}$$

Here, $\|\cdot\|$ refers to any of the vector norms previously discussed.

It is not difficult to see why the two expressions in Definition 2.5 are equivalent. Formally, we say that the norm in Definition 2.5 is an induced matrix norm. These can be computed quite easily.

(1). If $p = 1$, we define

$$\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^{m} |a_{ij}|$$

which refers to the maximum absolute value of the column sum of $\mathbf{A}$

(2). If $p = \infty$, we define

$$\|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}|$$

which refers to the maximum absolute row sum

(3). If $p = 2$, then the induced matrix norm is known as the spectral norm[†], and we define it to be

$$\|\mathbf{A}\|_2 = \sigma_{\max}(\mathbf{A}) = \text{square root of the largest eigenvalue of } \mathbf{A}^{\mathrm{T}}\mathbf{A}. \tag{2.6}$$

This refers to the largest singular value of $\mathbf{A}$[‡].

We state some properties of the induced matrix norms.

Proposition 2.1. The induced matrix norms are submultiplicative and consistent. That is to say,

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\|\,\|\mathbf{B}\| \quad \text{and} \quad \|\mathbf{Ax}\| \leq \|\mathbf{A}\|\,\|\mathbf{x}\| \quad \text{respectively.}$$

[†]Appears in singular value decomposition.

[‡]In a more general setting like for instance, over the complex numbers $\mathbb{C}$, then the transpose in (2.6) would be replaced by conjugate transpose $\mathbf{A}^*$.

Now that we have a method of measuring the sizes of the inputs and outputs to a system of linear equations, we can discuss how changes to the inputs affect the outputs. Consider a non-singular system $\mathbf{A}\mathbf{x} = \mathbf{b}$. Suppose that instead of $\mathbf{b}$, we have a perturbed $\widehat{\mathbf{b}}$ with $\Delta\mathbf{b} = \widehat{\mathbf{b}} - \mathbf{b}$. Let $\widehat{\mathbf{x}}$ be the solution to the perturbed system $\mathbf{A}\widehat{\mathbf{x}} = \widehat{\mathbf{b}}$ with $\Delta\mathbf{x} = \widehat{\mathbf{x}} - \mathbf{x}$. Then, we have

$$\mathbf{A}(\Delta\mathbf{x}) = \mathbf{A}(\widehat{\mathbf{x}} - \mathbf{x}) = \mathbf{A}\widehat{\mathbf{x}} - \mathbf{A}\mathbf{x} = \widehat{\mathbf{b}} - \mathbf{b} = \Delta\mathbf{b}.$$

Equivalently, we have $\Delta\mathbf{x} = \mathbf{A}^{-1}(\Delta\mathbf{b})$. As such,

$$\frac{\text{relative output error}}{\text{relative input error}} = \frac{\|\Delta\mathbf{x}\| / \|\mathbf{x}\|}{\|\Delta\mathbf{b}\| / \|\mathbf{b}\|} = \frac{\|\Delta\mathbf{x}\| \|\mathbf{b}\|}{\|\Delta\mathbf{b}\| \|\mathbf{x}\|} = \frac{\|\mathbf{A}^{-1}(\Delta\mathbf{b})\| \|\mathbf{A}\mathbf{x}\|}{\|\Delta\mathbf{b}\| \|\mathbf{x}\|}.$$

Since the induced matrix norm is consistent (Proposition 2.1), then

$$\frac{\text{relative output error}}{\text{relative input error}} \leq \left\|\mathbf{A}^{-1}\right\| \|\mathbf{A}\|.$$

Now, for any system $\mathbf{A}\mathbf{x} = \mathbf{b}$, we define the condition number to be $\kappa(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$. The condition number is a worst case scenario for how errors in the input are magnified. Also, there are potentially other cases, i.e. perturbing only $\mathbf{A}$, and perturbing $\mathbf{b}$ and $\mathbf{A}$ concurrently, but we will not discuss them here.

2.3 Forward Elimination and Backward Substitution

Suppose we have a square matrix $\mathbf{A}$ which denotes the coefficient matrix of a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. When performing Gaussian elimination to reduce $\mathbf{A}$ to a row-echelon form, what we really obtain is an upper triangular matrix. This part of Gaussian elimination is also known as forward elimination. From the pivot in the last row of the augmented matrix, we then perform backward substitution to obtain the solutions $x_1, \ldots, x_n$.

One can prove that for the forward elimination process,

$$\text{the number of divisions} \quad \text{is} \quad \frac{n(n-1)}{2} \quad \text{and}$$
$$\text{the number of subtractions} \quad \text{is} \quad \frac{n(n-1)(n+1)}{3}$$

For the backward substitution process,

$$\text{the number of divisions} \quad \text{is} \quad n \quad \text{and}$$
$$\text{the number of subtractions} \quad \text{is} \quad \frac{n(n-1)}{2}$$

In total, Gaussian elimination has time complexity $\mathcal{O}(n^3)$ because

$$\text{the number of divisions} \quad \text{is} \quad \frac{n(n-1)}{2} + n = \frac{n(n+1)}{2} \quad \text{and}$$

$$\text{the number of subtractions} \quad \text{is} \quad \frac{n(n-1)(n+1)}{3} + \frac{n(n-1)}{2} = \frac{n(n-1)(2n+5)}{6}$$

However, there is an important consideration ignored in the naïve form of Gaussian elimination and that is with regards to numerical stability — particularly the need for what is known as *pivoting* to avoid division by zero or by very small numbers which can lead to massive rounding errors. In Chapter 2.4, we will discuss a different way of viewing Gaussian elimination, and that is from the lens of LU factorisation.

2.4 LU Factorisation

When performing Gaussian elimination in MA2001, recall that we transform $\mathbf{A}$ to a row-echelon form, which is an upper triangular matrix $\mathbf{U}$. Note that if $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$ and say we have a system that can be solved without pivoting (see Chapter 2.5), we would need $n-1$ elementary matrices $\mathbf{E}_1, \ldots, \mathbf{E}_{n-1}$ such that

$$\mathbf{E}_{n-1} \ldots \mathbf{E}_1 \mathbf{A} = \mathbf{U}.$$

We define $\mathbf{L}^{-1} = \mathbf{E}_{n-1} \ldots \mathbf{E}_1$ (which is known to be a lower triangular matrix) so that

$$\mathbf{L} = \mathbf{E}_1^{-1} \ldots \mathbf{E}_{n-1}^{-1},$$

where we used the fact that the inverse of a lower triangular matrix is also lower triangular. This implies that $\mathbf{A} = \mathbf{LU}$ and hence the name LU factorisation, where we write $\mathbf{A}$ as the product of a lower triangular matrix and an upper triangular matrix.

We shall examine the significance of LU factorisation. Suppose $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$ and say we need to solve the following systems of linear equations:

$$\mathbf{A}\mathbf{x}_i = \mathbf{b}_i \quad \text{for all } 1 \leq i \leq p$$

Then, we define $\mathbf{X} \in \mathcal{M}_{n\times p}(\mathbb{R})$ and $\mathbf{B} \in \mathcal{M}_{n\times p}(\mathbb{R})$ as follows:

$$\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_p) \quad \text{and} \quad \mathbf{B} = (\mathbf{b}_1, \ldots, \mathbf{b}_p)$$

This is equivalent to solving the system $\mathbf{AX} = \mathbf{B}$.

Example 2.5. Say we wish to find $\mathbf{X} \in \mathcal{M}_{4\times 2}(\mathbb{R})$ such that

$$\begin{pmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{pmatrix} \mathbf{X} = \begin{pmatrix} 4 & 5 \\ 1 & 6 \\ -3 & 9 \\ 4 & -6 \end{pmatrix}.$$

Then, we need to solve two linear systems. One can verify that the forward elimination steps and coefficient matrices are the same for each system, but the right side of each augmented matrix and the backward substitutions are different.

One can extend the above-mentioned process to find the inverse of an invertible matrix. That is to say, suppose $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$ is invertible. Then, what collection of systems of linear equations must it satisfy? Clearly, we need to find $\mathbf{X} \in \mathcal{M}_{n\times n}(\mathbb{R})$ such that $\mathbf{AX} = \mathbf{I}$[†].

However, what if we wish to solve the system

$$\mathbf{A}\mathbf{x}_i = \mathbf{b}_i \quad \text{where } 1 \leq i \leq p$$

but we do not know all the $\mathbf{b}_i$ at the start? If we were to apply Gaussian elimination to each system independently, we would end up repeating a lot of work. Instead, we can separate the computation into two parts as follows:

(i) Elimination step: Transform A into an upper triangular matrix U and record the multipliers used in elimination.

(ii) Solve step: For each $\mathbf{b}_i$, apply forward elimination using the same multipliers, then perform backward substitution.

When we eliminate the first column, we compute multipliers $m_{21}, m_{31}, \ldots, m_{n1}$ and update the rows as follows:

$$R_2 \leftarrow R_2 - m_{21}R_1 \quad R_3 \leftarrow R_3 - m_{31}R_1 \quad \ldots \quad R_n \leftarrow R_n - m_{n1}R_1$$

The right side is transformed in exactly the same way. That is,

$$b_2 \leftarrow b_2 - m_{21}b_1 \quad b_3 \leftarrow b_3 - m_{31}b_1, \quad \ldots \quad b_n \leftarrow b_n - m_{n1}b_1.$$

Similarly, when we eliminate the second column using multipliers $m_{32}, m_{42}, \ldots, m_{n2}$, the updates are

$$b_3 \leftarrow b_3 - m_{32}b_2, \quad b_4 \leftarrow b_4 - m_{42}b_2 \quad \ldots \quad b_n \leftarrow b_n - m_{n2}b_2,$$

[†]Recall this procedure from MA2001.

and so on until the $(n-1)^{\text{th}}$ elimination step.

Example 2.6. Say we consider the system

$$\begin{pmatrix} 1 & 1 & 0 & 3 \\ 2 & 1 & -1 & 1 \\ 3 & -1 & -1 & 2 \\ -1 & 2 & 3 & -1 \end{pmatrix} \mathbf{x} = \mathbf{b}.$$

Applying Gaussian elimination to $\mathbf{A}$ yields

$$\begin{aligned} R_2 &\leftarrow R_2 - 2R_1 \\ R_3 &\leftarrow R_3 - 3R_1 \\ R_4 &\leftarrow R_4 + R_1 \end{aligned}$$

Then,

$$\begin{aligned} R_3 &\leftarrow R_3 - 4R_2 \\ R_4 &\leftarrow R_4 + 3R_2 \end{aligned}$$

This yields the upper triangular matrix

$$\mathbf{U} = \begin{pmatrix} 1 & 1 & 0 & 3 \\ 0 & -1 & -1 & -5 \\ 0 & 0 & 3 & 13 \\ 0 & 0 & 0 & -13 \end{pmatrix}.$$

The multipliers are

$$m_{21} = 2 \quad m_{31} = 3 \quad m_{41} = -1 \quad m_{32} = 4 \quad m_{42} = -3 \quad m_{43} = 0.$$

These can be stored in the strictly lower triangular part of $\mathbf{A}$, which yields

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 4 & 1 & 0 \\ -1 & -3 & 0 & 1 \end{pmatrix}.$$

Suppose $\mathbf{b} = (5, 6, 9, -6)$. Forward elimination yields

$$\begin{aligned} b_2 &\leftarrow 6 - 2 \cdot 5 = -4 \\ b_3 &\leftarrow 9 - 3 \cdot 5 = -6 \\ b_4 &\leftarrow -6 + 1 \cdot 5 = -1 \end{aligned}$$

and then

$$b_3 \leftarrow -6 - 4 \cdot (-4) = 10$$
$$b_4 \leftarrow -1 + 3 \cdot (-4) = -13$$

Lastly,

$$b_4 \leftarrow -13 - 0 \cdot 10 = -13.$$

In general, once the multipliers m_{ji} are known from eliminating $\mathbf{A}$, forward elimination on the right-hand side is:

$$b_i \leftarrow b_i - m_{i1}b_1 - m_{i2}b_2 - \ldots - m_{i,i-1}b_{i-1} \quad \text{for all } 2 \leq i \leq n.$$

After forward elimination, the system $\mathbf{Ux} = \mathbf{b}$ can be solved by backwards substitution, which yields

$$x_i = \frac{b_i - a_{i,i+1}x_{i+1} - \ldots - a_{in}x_n}{a_{ii}} \quad \text{for all } i = n, n-1, \ldots, 1.$$

To summarise, the multipliers m_{ji} depend only on $\mathbf{A}$, not on $\mathbf{b}$. We can precompute and store $\mathbf{L}$ (containing m_{ji}) and $\mathbf{U}$ (upper triangular form of $\mathbf{A}$). For each new $\mathbf{b}$, perform forward elimination using $\mathbf{L}$, then backwards substitution using $\mathbf{U}$. This is the basis of the LU factorisation method.

Example 2.7. Let

$$\mathbf{A} = \begin{pmatrix} 4 & -2 & 2 & 2 \\ -2 & 2 & -4 & 3 \\ 2 & -4 & 14 & 3 \\ 2 & -3 & 3 & 10 \end{pmatrix}.$$

Compute the LU factorisation of $\mathbf{A}$ via column operations.

Solution. We shall transform $\mathbf{A}$ to a lower triangular matrix with diagonal entries 1. Dividing first column by 4 yields

$$\begin{pmatrix} 4 & -2 & 2 & 2 \\ -2 & 2 & -4 & 3 \\ 2 & -4 & 14 & 3 \\ 2 & -3 & 3 & 10 \end{pmatrix} \xrightarrow{C_1 \times \frac{1}{4} \to C_1} \begin{pmatrix} 1 & -2 & 2 & 2 \\ -\frac{1}{2} & 2 & -4 & 3 \\ \frac{1}{2} & -4 & 14 & 3 \\ \frac{1}{2} & -3 & 3 & 10 \end{pmatrix}.$$

We eliminate the first three entries in the second, third and fourth columns to obtain

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & -3 & 4 \\ \frac{1}{2} & -3 & 13 & 2 \\ \frac{1}{2} & -2 & 2 & 9 \end{pmatrix}.$$

We then clear the $(2,3)$- and $(2,4)$-entries to obtain

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{2} & -3 & 4 & 14 \\ \frac{1}{2} & -2 & -4 & 17 \end{pmatrix}.$$

Make the next pivot 1 at $(3,3)$ and clear $(3,4)$ to obtain

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{2} & -3 & 1 & 0 \\ \frac{1}{2} & -2 & -1 & 31 \end{pmatrix}.$$

Finally, scale the last pivot to obtain

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ -\frac{1}{2} & 1 & 0 & 0 \\ \frac{1}{2} & -3 & 1 & 0 \\ \frac{1}{2} & -2 & -1 & 1 \end{pmatrix}.$$

Now, each pivot column has a 1 in its pivot row and zeros above it. The resulting matrix is $\mathbf{L}$ in the $\mathbf{LU}$ factorisation of $\mathbf{A}$. Thus,

$$\mathbf{U} = \mathbf{L}^{-1}\mathbf{A} = \begin{pmatrix} 4 & -2 & 2 & 2 \\ 0 & 1 & -3 & 4 \\ 0 & 0 & 4 & 14 \\ 0 & 0 & 0 & 31 \end{pmatrix}$$

which is upper triangular. □

Example 2.8 (DSA2102 AY25/26 Sem 1 Tutorial 3)**.** Compute the LU factorization of the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 4 & 5 \\ 3 & 1 & 0 \\ 4 & -2 & 2 \end{pmatrix}.$$

Solution. We wish to write $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ is lower triangular and $\mathbf{U}$ is upper triangular, where

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ \ell_{21} & 1 & 0 \\ \ell_{31} & \ell_{32} & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}.$$

By performing matrix multiplication, we have

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ \ell_{21}u_{11} & \ell_{21}u_{12} + u_{22} & \ell_{21}u_{13} + u_{23} \\ \ell_{31}u_{11} & \ell_{31}u_{12} + \ell_{32}u_{22} & \ell_{31}u_{13} + \ell_{32}u_{23} + u_{33} \end{pmatrix} = \begin{pmatrix} 2 & 4 & 5 \\ 3 & 1 & 0 \\ 4 & -2 & 2 \end{pmatrix}.$$

By comparing the first row, we have $u_{11} = 2$, $u_{12} = 4$ and $u_{13} = 5$. Updating the entries, we have

$$\begin{pmatrix} 2 & 4 & 5 \\ 2\ell_{21} & 4\ell_{21} + u_{22} & 5\ell_{21} + u_{23} \\ 2\ell_{31} & 4\ell_{31} + \ell_{32}u_{22} & 5\ell_{31} + \ell_{32}u_{23} + u_{33} \end{pmatrix} = \begin{pmatrix} 2 & 4 & 5 \\ 3 & 1 & 0 \\ 4 & -2 & 2 \end{pmatrix}.$$

Hence, $\ell_{21} = \frac{3}{2}$, and so on. One can solve for the other unknowns to obtain the LU factorisation, which is as follows:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ \frac{3}{2} & 1 & 0 \\ 2 & 2 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} 2 & 4 & 5 \\ 0 & -5 & -\frac{15}{2} \\ 0 & 0 & 7 \end{pmatrix}$$

and indeed, $\mathbf{A} = \mathbf{LU}$. □

Example 2.9 (DSA2102 AY25/26 Sem 1 Tutorial 3)**.** Consider the matrix

$$\mathbf{M} = \begin{pmatrix} 2 & 1 & 1 \\ 4 & 5 & 2 \\ 2 & -2 & 0 \end{pmatrix}.$$

(a) Compute the LU factorization of $\mathbf{M}$.

(b) Let $\mathbf{b} = (1, 2, 2)$. First solve $\mathbf{Ly} = \mathbf{b}$, and then solve $\mathbf{Ux} = \mathbf{y}$, where $\mathbf{L}$ and $\mathbf{U}$ are the factors you computed in part **(a)**.

(c) What is the solution to the system $\mathbf{Mx} = \mathbf{b}$?

Solution.

(a) The LU factorisation is

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 1 & -1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 3 & 0 \\ 0 & 0 & -1 \end{pmatrix}.$$

(b) We first wish to solve

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

which yields $\mathbf{y} = (1, 0, 1)$. We then solve

$$\begin{pmatrix} 2 & 1 & 1 \\ 0 & 3 & 0 \\ 0 & 0 & -1 \end{pmatrix} \mathbf{x} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

This yields $\mathbf{x} = (1, 0, -1)$.

(c) By the LU factorisation, we have $\mathbf{LUx} = \mathbf{b}$. Replacing $\mathbf{Ux} = \mathbf{y}$, we obtain $\mathbf{Ly} = \mathbf{b}$. Since the solution to $\mathbf{Ly} = \mathbf{b}$ is $\mathbf{y} = (1, 0, 1)$ from **(b)**, the desired solution to the system $\mathbf{Mx} = \mathbf{b}$ is $\mathbf{x} = (1, 0, -1)$. □

2.5 Pivoting

So far, we have presented Gaussian elimination in its simplest form. In practice, complications arise that require extra care, most notably pivoting.

Example 2.10 (a zero pivot problem)**.** Consider the system

$$\begin{aligned} x_1 - x_2 \quad & +2x_3 - x_4 \quad && = -8 \\ 2x_1 - 2x_2 \quad & +3x_3 - 3x_4 \quad && = -20 \\ x_1 + x_2 \quad & +x_3 \quad && = -2 \\ x_1 - 2x_2 \quad & +4x_3 + 3x_4 \quad && = 2 \end{aligned}$$

We can represent the above information as a matrix (an augmented matrix is fine as well). That is,

$$\begin{pmatrix} 1 & -1 & 2 & -1 & -8 \\ 2 & -2 & 3 & -3 & -20 \\ 1 & 1 & 1 & 0 & -2 \\ 1 & -2 & 4 & 3 & 2 \end{pmatrix}.$$

To eliminate the first variable x_1, we perform some elementary row operations to obtain the matrix

$$\begin{pmatrix} 1 & -1 & 2 & -1 & -8 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & -1 & 2 & 4 & 10 \end{pmatrix}.$$

We then attempt to eliminate x_2. The pivot in the position $(2, 2)$ is zero (where $(1, 1)$ refers to the top-left entry), making division impossible. Swapping the second and third rows, then proceeding with elimination as usual yields

$$\begin{pmatrix} 1 & -1 & 2 & -1 & -8 \\ 0 & 2 & -1 & 1 & 6 \\ 0 & 0 & -1 & -1 & -4 \\ 0 & 0 & 0 & 3 & 7 \end{pmatrix}.$$

One can then solve for x_4, x_3, x_2, x_1 in order via backward substitution.

However, note that there would be numerical issues with small pivots (see Example 2.11) — dividing by a very small number amplifies rounding errors, possibly destroying accuracy.

Example 2.11 (small pivot). Consider the system of equations

$$\begin{aligned} 0.0003x_1 + 59.147x_2 &= 59.15 \\ 5.291x_1 - 6.130x_2 &= 46.78 \end{aligned}$$

This produces the exact solution $x_1 = 10$ and $x_2 = 1$. Without pivoting, computer arithmetic produces a small but noticeable relative error. With *partial pivoting* (swapping rows so the pivot is largest in magnitude), the numerical solution matches the exact result to machine precision.

Example 2.12 (choosing the largest pivot). Suppose the coefficient matrix of a linear system is

$$\mathbf{A} = \begin{pmatrix} 0.0002 & 33.582 & 12.34 & 8.904 \\ 1.23 & -9.87 & 0.3 & 1.83 \\ 0.12 & 3.4 & 1.63 & 1.34 \end{pmatrix}.$$

So, the candidates for the pivot in the first column are 0.0002, 1.23, and 0.12. The largest magnitude is 1.23, so we need to swap R_1 with R_2. This process is repeated at each elimination stage to improve stability.

In general, row swaps can be described using *permutation matrices* (a specific type of elementary matrix) — if $\mathbf{P}_{i,j}$ is the identity matrix with rows i and j swapped, then

$$\mathbf{P}_{i,j}\mathbf{A} \quad \text{swaps rows } i \text{ and } j \text{ of } \mathbf{A}.$$

Partial pivoting at each step amounts to multiplying on the left by a sequence of permutation matrices, which yields the equation

$$\mathbf{PA} = \mathbf{LU},$$

where $\mathbf{P}$ denotes the product of all permutation matrices used. Solving $\mathbf{Ax} = \mathbf{b}$ then becomes $\mathbf{PAx} = \mathbf{Pb}$, so $\mathbf{LUx} = \mathbf{Pb}$. Since we permute the rows of $\mathbf{A}$, we must do the same for the rows of $\mathbf{b}$, so we apply $\mathbf{P}$ to reorder $\mathbf{b}$. The reordered right side becomes $\mathbf{L}\widetilde{\mathbf{b}} = \mathbf{Pb}$, then we solve this via forward substitution. Lastly, we solve $\mathbf{Ux} = \widetilde{\mathbf{b}}$ by backward substitution. Not only does this avoid division by zero, but it also significantly reduces numerical instability in Gaussian elimination.

Example 2.13 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Apply Gaussian elimination with partial pivoting to solve the linear system represented by the following augmented matrix:

$$\left(\begin{array}{ccccc|c} 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 7 & 3 & -7 \end{array}\right)$$

Solution. In the first column, the element of the largest magnitude is 1, which is in the third row. So, we swap row 1 with row 3 to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 7 & 3 & -7 \end{array}\right).$$

We then focus on the second column. Below $a_{12} = 4$, we see that the pivot is not in the second row but in the third. This pivot is trivially of the largest magnitude,

which is 1. Hence, we swap row 2 and row 3 to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 7 & 3 & -7 \end{array}\right).$$

In the third colmumn, consider the submatrix formed by the column vector $(2, 1, 1)$. Note that 2 is of the largest magnitude. By considering the third row, we perform row reduction on the fourth and fifth rows to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 0 & -\frac{3}{2} & 0 & \frac{3}{2} \\ 0 & 0 & 0 & \frac{11}{2} & 1 & -\frac{11}{2} \end{array}\right).$$

Swap rows 4 and 5 to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 0 & \frac{11}{2} & 1 & \frac{11}{2} \\ 0 & 0 & 0 & -\frac{3}{2} & 0 & -\frac{3}{2} \end{array}\right).$$

We eliminate the $a_{54} = -\frac{3}{2}$ entry to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 0 & \frac{11}{2} & 1 & \frac{11}{2} \\ 0 & 0 & 0 & 0 & \frac{3}{11} & 0 \end{array}\right).$$

Hence, $\frac{3}{11}x_5 = 0$ so $x_5 = 0$. By backward substitution, $x_4 = -1$. One can find the other unknowns so we obtain $(x_1, \ldots, x_5) = (1, 0, 0, -1, 0)$. □

Example 2.14 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Apply Gaussian elimination with partial pivoting to solve the linear system represented by the following

augmented matrix:

$$\left(\begin{array}{ccccc|c} 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 7 & 3 & -7 \end{array}\right)$$

Solution. In the first column, the element of the largest magnitude is 1, which is in the third row. So, we swap row 1 with row 3 to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 7 & 3 & -7 \end{array}\right).$$

We then focus on the second column. Below $a_{12} = 4$, we see that the pivot is not in the second row but in the third. This pivot is trivially of the largest magnitude, which is 1. Hence, we swap row 2 and row 3 to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 1 & 0 & 2 & 0 \\ 0 & 0 & 1 & 7 & 3 & -7 \end{array}\right).$$

In the third colmumn, consider the submatrix formed by the column vector $(2, 1, 1)$. Note that 2 is of the largest magnitude. By considering the third row, we perform row reduction on the fourth and fifth rows to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 0 & -\frac{3}{2} & 0 & \frac{3}{2} \\ 0 & 0 & 0 & \frac{11}{2} & 1 & -\frac{11}{2} \end{array}\right).$$

Swap rows 4 and 5 to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 0 & \frac{11}{2} & 1 & \frac{11}{2} \\ 0 & 0 & 0 & -\frac{3}{2} & 0 & -\frac{3}{2} \end{array}\right).$$

We eliminate the $a_{54} = -\frac{3}{2}$ entry to obtain

$$\left(\begin{array}{ccccc|c} 1 & 4 & 1 & 1 & 1 & 0 \\ 0 & 1 & -7 & 2 & 3 & -2 \\ 0 & 0 & 2 & 3 & 4 & -3 \\ 0 & 0 & 0 & \frac{11}{2} & 1 & \frac{11}{2} \\ 0 & 0 & 0 & 0 & \frac{3}{11} & 0 \end{array}\right).$$

Hence, $\frac{3}{11}x_5 = 0$ so $x_5 = 0$. By backward substitution, $x_4 = -1$. One can find the other unknowns so we obtain $(x_1, \ldots, x_5) = (1, 0, 0, -1, 0)$. □

Example 2.15 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Consider the linear systems represented by the following augmented matrices. Write down the augmented matrix after the elimination step without setting the eliminated coefficients to be zero. Then proceed to find the solution of the system.

(a)

$$\left(\begin{array}{cccc|c} 1 & 1 & 0 & 1 & 2 \\ 2 & 1 & -1 & 1 & 1 \\ 3 & -1 & -1 & 2 & -3 \\ -1 & 2 & 3 & -1 & 4 \end{array}\right)$$

(b)

$$\left(\begin{array}{ccccc|c} 1 & 1 & -1 & 1 & -1 & -1 \\ 3 & 1 & -3 & -2 & 3 & -6 \\ 2 & 2 & 1 & -1 & 1 & 7 \\ 4 & 1 & -1 & 4 & -5 & -7 \\ 16 & -1 & 1 & 9 & -1 & 8 \end{array}\right)$$

Solution. The idea is to use partial pivoting to convert the coefficient matrix to an upper triangular form. We omit the detailed solutions but anyway, the answer in **(a)** is $(-1, 2, 0, 1)$ and the answer in **(b)** is $(-1, 3, 3, 3, 3)$. □

Example 2.16 (DSA2102 AY25/26 Sem 1 Tutorial 2)**.** Assume that the augmented matrix

$$\left(\begin{array}{cccc|c} 1 & 0 & \frac{4}{5} & \frac{4}{5} & 0 \\ 0 & 3 & 0 & \frac{4}{7} & \frac{4}{7} \\ 5 & 0 & 6 & 1 & 0 \\ 0 & 7 & 0 & 8 & 1 \end{array}\right)$$

is obtained by Gaussian elimination with partial pivoting which only makes necessary changes of matrix entries. That is, only row labels are swapped, not the rows themselves, and eliminated entries are not set to 0. Find the original augmented matrix and the solution of the linear system.

Solution. Note that the row labels are 3, 4, 1, 2. To see why, we take a look at the first column and we see that the element of largest magnitude is in the third row, so row 3 should be labelled '1'. Next, we look at the fourth row, and it should be labelled 2 since the number 7 is of the largest magnitude in the second column consisting of the column vector $(0, 3, 7)$ (we note that the other 0 is omitted as it had already been considered in the first column). One can continue this process.

Swapping rows, we obtain

$$\left(\begin{array}{cccc|c} 5 & 0 & 6 & 1 & 0 \\ 0 & 7 & 0 & 8 & 1 \\ 1 & 0 & \frac{4}{5} & \frac{4}{5} & 0 \\ 0 & 3 & 0 & \frac{4}{7} & \frac{4}{7} \end{array}\right).$$

Solving the linear system is trivial and we leave it as an exercise. □

Example 2.17 (DSA2102 AY25/26 Sem 1 Tutorial 3)**.** Compute the $\mathbf{PA} = \mathbf{LU}$ factorization of the matrix

$$\begin{pmatrix} 1 & 2 & -1 & 0 \\ 2 & 4 & -2 & -1 \\ -3 & -5 & 6 & 1 \\ -1 & 2 & 8 & -2 \end{pmatrix}.$$

Solution. We find $\mathbf{P}$ through partial pivoting. Swapping the first and third rows, we have

$$\begin{pmatrix} -3 & -5 & 6 & 1 \\ 2 & 4 & -2 & -1 \\ 1 & 2 & -1 & 0 \\ -1 & 2 & 8 & -2 \end{pmatrix}.$$

Perform the row operations $\frac{2}{3}R_1 + R_2 \to R_2$, $\frac{1}{3}R_1 + R_3 \to R_3$, and $-\frac{1}{3}R_1 + R_4 \to R_4$ (which corresponds to three elementary matrices $\mathbf{E}_1, \mathbf{E}_2, \mathbf{E}_3$) to obtain the matrix

$$\begin{pmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{2}{3} & 2 & -\frac{1}{3} \\ 0 & \frac{1}{3} & 1 & \frac{1}{3} \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \end{pmatrix}.$$

Swapping the second and fourth rows yields

$$\begin{pmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & \frac{1}{3} & 1 & \frac{1}{3} \\ 0 & \frac{2}{3} & 2 & -\frac{1}{3} \end{pmatrix}.$$

Then, we have

$$\begin{pmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{5}{11} & \frac{6}{11} \\ 0 & 0 & \frac{10}{11} & \frac{1}{11} \end{pmatrix}.$$

Swap rows 3 and 4 to obtain

$$\begin{pmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{10}{11} & \frac{1}{11} \\ 0 & 0 & \frac{5}{11} & \frac{6}{11} \end{pmatrix}.$$

Then, we can reduce the matrix to an upper triangular one, which is

$$\mathbf{U} = \begin{pmatrix} -3 & -5 & 6 & 1 \\ 0 & \frac{11}{3} & 6 & -\frac{7}{3} \\ 0 & 0 & \frac{10}{11} & \frac{1}{11} \\ 0 & 0 & 0 & \frac{1}{2} \end{pmatrix}.$$

The permutation matrix $\mathbf{P}$ is

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}.$$

One can see that $\mathbf{L}$ is the product of some elementary matrices, for which we would eventually obtain

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 \\ -\frac{2}{3} & \frac{2}{11} & 1 & 0 \\ -\frac{1}{3} & \frac{1}{11} & \frac{1}{2} & 1 \end{pmatrix}.$$

□

2.6 Some Special Systems and the Cholesky Factorisation

Oftentimes, systems of linear equations encountered in practice have some special structure. A common example is a *banded system*, where all the non-zero entries are near the diagonal. Take for example, the following tri-diagonal matrix:

$$\begin{pmatrix} -2 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 5 & 2 & -7 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -8 & -4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 11 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 13 & -1 \end{pmatrix}$$

For the best efficiency, banded systems should be stored differently, but algorithms need only minimal adjustment, and they are much faster to work with. Banded systems are a special case of *sparse systems*, which have relatively few non-zero

entries. For example, consider the following matrix:

$$\begin{pmatrix} -2 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 1 & 0 & -3 & 0 & 0 & 0 & -1 & 0 \\ 0 & 5 & 2 & 0 & 0 & 0 & -7 & 2 \\ -8 & 0 & -1 & 0 & 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & 7 & 11 & 5 & 0 & 0 \\ 0 & -1 & 0 & -9 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 2 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 13 & -1 \end{pmatrix}$$

Again, sparse systems need to be stored differently for efficiency. Algorithms for sparse matrices need more adjustment, but large sparse matrices are common nowadays and necessitate special techniques. We will not discuss them further but instead, our focus is on two other special properties: *symmetry* and *positivity*.

Definition 2.3 (symmetric matrix). Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$. Then, $\mathbf{A}$ is symmetric if and only if it is equal to its transpose. That is, $\mathbf{A}^{\mathrm{T}} = \mathbf{A}$.

Note that only square matrices can be symmetric, the identity matrix $\mathbf{I}$ and the zero matrix $\mathbf{0}$ are symmetric, any diagonal matrix $\mathbf{D}$ is symmetric, and for any square matrix $\mathbf{A}$, the matrix $\mathbf{B} = \mathbf{A} + \mathbf{A}^{\mathrm{T}}$ is symmetric (easy to see because the transpose of the transpose of a matrix yields the original matrix).

Symmetric matrices, and even operators, arise frequently in Applied Mathematics. For example, the Laplacian operator[†] given by the divergence of the gradient of a scalar function $f : \mathbb{R}^n \to \mathbb{R}$ on a Euclidean space $\mathbb{R}^n$ is defined to be

$$\Delta f = \frac{\partial^2 f}{\partial x_1} + \ldots + \frac{\partial^2 f}{\partial x_n} = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i}. \tag{2.7}$$

[†]Will see in MA2104 Multivariable Calculus, MA4221 Partial Differential Equations, etc.

We say that the expression in (2.7) is symmetric. Also in Multivariable Calculus[‡], of interest is the Hessian matrix, defined by

$$\begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}. \tag{2.8}$$

The Hessian matrix arises in multivariable optimisation problems. Again, suppose $f : \mathbb{R}^n \to \mathbb{R}$. One can compute the gradient vector

$$\nabla f = \left\langle \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right\rangle$$

and solve the equation $\nabla f = 0$. Thereafter, compute the Hessian matrix (2.8) and classify whether the extreme point is a maximum, minimum, or a saddle (of course, just like the second derivative test for the single-variable case, we have a case which is inconclusive).

We have discussed the concept of a matrix being symmetric. Next, what does it mean for a square matrix to be *positive*? To be precise, the term used is positive definite (Definition 2.4).

Definition 2.4 (positive definite matrix)**.** Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$. Then, $\mathbf{A}$ is said to be positive definite if and only if

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0 \quad \text{for all } \mathbf{x} \neq \mathbf{0}.$$

As expected, a square matrix $\mathbf{A}$ is said to be negative definite if and only if $-\mathbf{A}$ is positive definite. Note that positive and negative definite matrices are necessarily invertible. Next, if we have $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} \geq 0$ (in contrast to > 0 in Definition 2.4), then we say that $\mathbf{A}$ is a positive semi-definite matrix. Note that matrices can be non-zero and neither positive nor negative definite.

Example 2.18. Throughout, let $\mathbf{x} = (x, y)$. Then,

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \quad \text{is positive definite.}$$

[‡]Also appears in MA3210 Mathematical Analysis II.

To see why, $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = x^2 + y^2$, which is > 0 because x and y cannot both be zero, and the square of any non-zero real number is > 0.

Next, let

$$\mathbf{B} = \begin{pmatrix} -3 & 1 \\ 1 & -3 \end{pmatrix}.$$

One can prove that $\mathbf{x}^{\mathrm{T}}\mathbf{B}\mathbf{x} < 0$, which implies that $\mathbf{B}$ is negative definite.

Lastly, let

$$\mathbf{C} = \begin{pmatrix} 2 & 1 \\ 0 & -1 \end{pmatrix}.$$

As such, $\mathbf{x}^{\mathrm{T}}\mathbf{C}\mathbf{x} = 2x^2 + xy - y^2$ which can take on positive and negative values. For example, let $f(x, y) = 2x^2 + xy - y^2$. Then, $f(1, 1) = 2$ and $f(1, -2) = -4$, so $\mathbf{C}$ is neither a positive nor definite matrix.

Of interest are matrices that are both positive definite and symmetric, and sometimes, this is included in the definition.

Example 2.19. Let $\mathbf{D} \in \mathcal{M}_{n\times n}(\mathbb{R})$ be a diagonal matrix with diagonal entries $d_1, \ldots, d_n$. Then, one sees that

$$\mathbf{x}^{\mathrm{T}}\mathbf{D}\mathbf{x} = d_1x_1^2 + \ldots + d_nx_n^2.$$

Note that $\mathbf{D}$ is positive definite if and only if $d_i > 0$ for all $1 \leq i \leq n$. Also, clearly $\mathbf{D}$ is a symmetric matrix.

Example 2.20. Consider the upper triangular matrix

$$\mathbf{U} = \begin{pmatrix} 1 & -2 & -2 \\ 0 & 2 & 4 \\ 0 & 0 & 3 \end{pmatrix}$$

and let $\mathbf{v} = (x, y, z)$. Then, one can deduce that

$$\mathbf{v}^{\mathrm{T}}\mathbf{U}\mathbf{v} = x^2 + 2y^2 + 3z^2 - 2xy - 2xz + 4yz. \tag{2.9}$$

We wish to prove that $\mathbf{U}$ is positive definite so it suffices to show that $\mathbf{v}^{\mathrm{T}}\mathbf{U}\mathbf{v}$ can be written as the sum of squares. By considering $-2xy$, we need to compensate with $x^2 + y^2$ so that $x^2 - 2xy + y^2 = (x - y)^2$ which is a perfect square. The same

can be argued for the other two coloured expressions. However, we would run into a problem because x^2 cannot be *nicely distributed* into the completed square form of $(x-y)^2$ and $(x-z)^2$.

Another way to look at the expression in (2.9) is to focus on $x^2 - 2xy - 2xz$. We can write

$$x^2 - 2xy - 2xz = x^2 - 2x\,(y+z)\,.$$

We realise that the missing term required to complete the square is $(y+z)^2$ so

$$x^2 - 2xy - 2xz = (x-(y+z))^2 - (y+z)^2\,.$$

Hence, (2.9) becomes

$$(x-(y+z))^2 - (y+z)^2 + 2y^2 + 3z^2 + 4yz = (x-y-z)^2 + y^2 + 2z^2 + 2yz$$

so again by completing the square, it becomes $(x-y-z)^2 + (y+z)^2 + z^2 > 0$, which is the sum of three squares.

In general, it is tedious to check whether a matrix is positive definite by Definition 2.4. However, we can make use of the following fact in Proposition 2.2:

Proposition 2.2. Let $\mathbb{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$. Then,

$$\mathbf{A} \text{ is positive definite} \quad \text{if and only if} \quad \mathbf{A} + \mathbf{A}^{\mathrm{T}} \text{ is positive definite.}$$

This leads to much easier tests.

Proposition 2.3. Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$ be a symmetric matrix. Then, $\mathbf{A}$ is positive definite if and only if the determinants of all its leading principal submatrices $\mathbf{A}_k$ are positive.

Example 2.21. Given

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 3 \end{pmatrix},$$

its principal submatrices are

$$\mathbf{A}_1 = \begin{pmatrix} 1 \end{pmatrix} \quad \mathbf{A}_2 = \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \quad \mathbf{A}_3 = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \\ 1 & 2 & 3 \end{pmatrix} = \mathbf{A}.$$

Note that $\det(\mathbf{A}_1) = 1$ but $\det(\mathbf{A}_2) = -3$ so by Proposition 2.3, $\mathbf{A}$ is not positive definite.

Example 2.22 (DSA2102 AY25/26 Sem 1 Tutorial 3)**.** Determine if the following symmetric matrices are positive definite:

(a)

$$\begin{pmatrix} 1 & 3 \\ 3 & 10 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 1 & 2 \\ 2 & 2 \end{pmatrix}$$

Solution.

(a) The determinant of each submatrix is positive, so the matrix is positive definite. Another way to argue is using quadratic forms, which is quite fun. Let $\mathbf{x} = (x, y)$ be a column vector and $\mathbf{A}$ be the mentioned coefficient matrix. Then,

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = x^2 + 6xy + 10y^2 = x^2 + 6xy + 9y^2 + y^2 = (x + 3y)^2 + y^2 > 0.$$

So the matrix is indeed positive definite.

(b) The determinant of the matrix is negative, so the matrix is not positive definte. □

Proposition 2.4. Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$ be a symmetric matrix. Then, $\mathbf{A}$ is positive definite if and only if its eigenvalues are all positive.

Finally, any matrix of the form $\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$ is positive semi-definite. If $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$ with $m \geq n$ and is of full rank, then $\mathbf{B}$ is positive definite. While this may seem like a special case, it is relatively common. For example, the covariance matrix in Statistics is of this form. Symmetric positive definite matrices arise in for instance linear least squares problems (see Chapter 3). If $\mathbf{A}$ is a symmetric

positive definite matrix, then we can factor $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ rather than $\mathbf{A} = \mathbf{L}\mathbf{U}$. Due to symmetry, we only need to store and work with half the matrix, and pivoting is not required for stability.

For the $n = 2$ case, say

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} \ell_{11} & 0 \\ \ell_{21} & \ell_{22} \end{pmatrix} \begin{pmatrix} \ell_{11} & \ell_{21} \\ 0 & \ell_{22} \end{pmatrix}.$$

For the $n = 3$ case, say

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{pmatrix}.$$

Note that in each case, $\mathbf{A}$ is a symmetric positive definite matrix. We leave it as a fun exercise to the reader to solve for each of the ℓ_{ij}'s, which merely requires some simple algebraic manipulation. This procedure is known as the Cholesky algorithm, and the factorisation $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathrm{T}}$ is called the Cholesky factorisation.

Consider the following problem. If $\mathbf{A}$ is an $n \times n$ symmetric positive definite matrix, then its diagonal entries must be positive. What about the converse of the statement? It turns out that the statement is true. By the real spectral theorem, $\mathbf{A}$ is symmetric implies that it is orthogonally diagonalisable. So, there exists an orthogonal matrix $\mathbf{P}$ and a diagonal matrix $\mathbf{D}$ such that $\mathbf{A} = \mathbf{P}\mathbf{D}\mathbf{P}^{\mathrm{T}}$. Let $\mathbf{x} \in \mathbb{R}^n$. Consider

$$\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{x}^{\mathrm{T}}\mathbf{P}\mathbf{D}\mathbf{P}^{\mathrm{T}}\mathbf{x} = \left(\mathbf{P}^{\mathrm{T}}\mathbf{x}\right)^{\mathrm{T}} \mathbf{D} \left(\mathbf{P}^{\mathrm{T}}\mathbf{x}\right).$$

Let $\mathbf{y} = \mathbf{P}^{\mathrm{T}}\mathbf{x} \in \mathbb{R}^n$. By definition of a positive definite matrix, we have $\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} > 0$, so $\mathbf{y}^{\mathrm{T}}\mathbf{D}\mathbf{y} > 0$. Writing the entries out,

$$\mathbf{y}^{\mathrm{T}}\mathbf{D}\mathbf{y} = d_1 y_1^2 + d_2 y_2^2 + \cdots + d_n y_n^2 > 0.$$

Since $\mathbf{P}$ is orthogonal, then $\|y_i\|^2 = 1$. Since $\mathbf{A}$ is positive definite, then every eigenvalue is positive. It follows that every diagonal entry of $\mathbf{A}$ is positive.

However, the converse is false. Let

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

which has positive diagonal entries. However, the matrix is not positive definite.

Example 2.23. Let

$$\mathbf{A} = \begin{pmatrix} 4 & -2 & 2 \\ -2 & 2 & 2 \\ 2 & -4 & 11 \end{pmatrix}.$$

Clearly, $\mathbf{A}$ is symmetric. One can use Definition 2.4 or Proposition 2.3 to prove that $\mathbf{A}$ is positive definite. We know find the Cholesky factorisation of $\mathbf{A}$. Suppose

$$\mathbf{A} = \begin{pmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{pmatrix} \begin{pmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{pmatrix} = \begin{pmatrix} \ell_{11}^2 & \ell_{11}\ell_{21} & \ell_{11}\ell_{31} \\ \ell_{11}\ell_{21} & \ell_{21}^2 + \ell_{22}^2 & \ell_{21}\ell_{31} + \ell_{22}\ell_{32} \\ \ell_{11}\ell_{31} & \ell_{21}\ell_{31} + \ell_{22}\ell_{32} & \ell_{31}^2 + \ell_{32}^2 + \ell_{33}^2 \end{pmatrix}.$$

Comparing the $(1,1)$-entry, we see that $\ell_{11}^2 = 4$. By default, we take the positive square root, so $\ell_{11} = 2$. Next, by considering the $(1,2)$-entry, we have $2\ell_{21} = -2$, so $\ell_{21} = -1$. One can slowly deduce the other entries (I do not wish to bore the reader with such calculations), thus obtaining the Cholesky factorisation of $\mathbf{A}$ being

$$\mathbf{A} = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 1 & -3 & 1 \end{pmatrix} \begin{pmatrix} 2 & -1 & 1 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{pmatrix}.$$

Example 2.24 (DSA2102 AY25/26 Sem 1 Tutorial 3)**.** Solve the system of equations by finding the Cholesky factorization of the coefficient matrices below. Make sure to verify that the matrices are symmetric and positive definite.

(a)

$$\begin{pmatrix} 1 & -1 \\ -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ -7 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -1 \\ 0 & -1 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ -7 \end{pmatrix}$$

Solution. We recall that in order to find the Cholesky factorisation of a matrix, we need it to be symmetric and positive definite. One can obtain the Cholesky factorisation easily.

(a) Let

$$\mathbf{R} = \begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix}$$

so the coefficient matrix becomes $\mathbf{R}\mathbf{R}^{\mathrm{T}}$. Solving

$$\begin{pmatrix} 1 & 0 \\ -1 & 2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 3 \\ -7 \end{pmatrix}$$

yields $\mathbf{y} = (3, -2)$. We see that the benefit of Cholesky factorisation is that in the lower triangular matrix $\mathbf{R}$, the top right entry is zero, which speeds up the computation process. We then solve

$$\begin{pmatrix} 1 & -1 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ -2 \end{pmatrix}$$

so $\mathbf{x} = (2, -1)$.

(b) Let

$$\mathbf{R} = \begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 2 \end{pmatrix}$$

so the coefficient matrix becomes $\mathbf{R}\mathbf{R}^{\mathrm{T}}$. Solving

$$\begin{pmatrix} 2 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & -1 & 2 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ -7 \end{pmatrix}$$

yields $\mathbf{y} = (0, 3, -2)$. Solving

$$\begin{pmatrix} 2 & -1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ -2 \end{pmatrix}$$

yields $\mathbf{x} = (1, 2, -1)$. □

It appears that Cholesky factorisation is faster than Gaussian elimination. One can work out that the total number of operations required for the Cholesky factorisation is $\frac{1}{6}(n^3 - n)$. The computational complexity of LU factorisation is $\mathcal{O}\left(\frac{2}{3}n^3\right)$, whereas Cholesky factorisation yields $\mathcal{O}\left(\frac{1}{3}n^3\right)$. So indeed, Cholesky factorisation is twice as fast as Gaussian elimination.

Example 2.25 (DSA2102 AY25/26 Sem 1 Tutorial 3)**.** Consider the matrix

$$\mathbf{M} = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 6 & 0 \\ -1 & 0 & 9 \end{pmatrix}.$$

(a) Compute the LU factorisation of $\mathbf{M}$.
(b) Verify that $\mathbf{M}$ is positive definite.
(c) Compute the Cholesky factorization of $\mathbf{M}$.
(d) What relationship, if any, holds between the LU factors and the Cholesky factors?

Solution.

(a) The LU factorisation yields

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & 1 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{U} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 2 & 2 \\ 0 & 0 & 6 \end{pmatrix}.$$

(b) We shall verify that $\mathbf{M}$ is positive definite. One can use the conventional approach by computing the determinant of the leading principal subminors. We omit the details.

(c) We wish to construct an upper triangular matrix $\mathbf{R}$ such that $\mathbf{M} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$. Let

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \end{pmatrix}.$$

Then,

$$\begin{pmatrix} r_{11}^2 & r_{11}r_{12} & r_{11}r_{13} \\ r_{11}r_{12} & r_{12}^2 + r_{22}^2 & r_{12}r_{13} + r_{22}r_{23} \\ r_{11}r_{13} & r_{12}r_{13} + r_{22}r_{23} & r_{13}^2 + r_{23}^2 + r_{33}^2 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 6 & 0 \\ -1 & 0 & 9 \end{pmatrix}.$$

By considering the $(1,1)$-entry, we have $r_{11}^2 = 1$ so $r_{11} = 1$. So, $r_{12} = 2$ and $r_{13} = -1$. One can then deduce the remaining entries to construct $\mathbf{R}$, which is

$$\mathbf{R} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & \sqrt{2} & \sqrt{2} \\ 0 & 0 & 2 \end{pmatrix}.$$

(d) In **(a)**, the LU factorisation yields $\mathbf{M} = \mathbf{LU}$ whereas the Cholesky decomposition in **(c)** yields $\mathbf{M} = \mathbf{R}^{\mathrm{T}}\mathbf{R}$, where $\mathbf{L}$ is lower triangular and $\mathbf{U}$ and $\mathbf{R}$ are upper triangular. Clearly, $\mathbf{U} \neq \mathbf{R}$. Having said that, observe that

$$\mathbf{L}^{\mathrm{T}} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{and} \quad \mathbf{R} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & \sqrt{2} & \sqrt{2} \\ 0 & 0 & 2 \end{pmatrix}.$$

The rows of $\mathbf{R}$ are scalar multiples of the rows of $\mathbf{L}^{\mathrm{T}}$ (equivalently, the columns of $\mathbf{R}^{\mathrm{T}}$ are scalar multiples of the columns of $\mathbf{L}$), where the scalars are given by the square roots of the diagonal entries of $\mathbf{U}$. □

Chapter 3

Linear Least Squares

3.1 Least Squares Problems

Here, we are interested in systems of linear equations $\mathbf{Ax} = \mathbf{b}$ that have no solution. Observe the following equivalent statements:

$$\mathbf{Ax} = \mathbf{b} \Leftrightarrow \mathbf{Ax} - \mathbf{b} = \mathbf{0} \Leftrightarrow \|\mathbf{Ax} - \mathbf{b}\|_2 = 0$$

So, we can minimise the norm instead. Recall from MA2001 that the method of least squares appears in problems involving linear regression, where we consider the matrix equation

$$\mathbf{X}\beta = \mathbf{y} \quad \text{or explicitly} \quad \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

This is an example of an overdetermined system of equations. We will focus on the linear regression setting, but note that the methods we discuss apply to general overdetermined systems.

Example 3.1 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.**

(a) Under what condition will a least-squares solution to an overdetermined system exist?

(b) Under what condition will it be unique?

Solution.

(a) A system of linear equations is said to be overdetermined if there are more equations than unknowns. Say we have a least squares problem $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$ and $m > n$. The least squares problem is

$$\min_{\mathbf{x}\in\mathbb{R}^n} \|\mathbf{Ax} - \mathbf{b}\|_2 .$$

So, a least squares solution exists for any $\mathbf{A}$ and $\mathbf{b}$.

(b) We consider the normal equations $\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$. The least squares problem has infinitely many solutions is $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is singular. Equivalently, for the problem to have a unique solution, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ must be non-singular. □

Consider the case when $\mathbf{A}$ is the concatenation of two column vectors $\mathbf{a}_1$ and $\mathbf{a}_2$. The existence of a solution to $\mathbf{Ax} = \mathbf{b}$ implies that $\mathbf{b} \in \text{span}\{\mathbf{a}_1, \mathbf{a}_2\}$. This span is two-dimensional so we can visualise it as a plane. On the other hand, if a solution to the linear system does not exist, then $\mathbf{b} \notin \text{span}\{\mathbf{a}_1, \mathbf{a}_2\}$. We can instead try to find the vector $\mathbf{y}$ in the span that is the closest to $\mathbf{b}$.

Note that $\mathbf{Ax} = \mathbf{b}$ does not have a solution, but $\mathbf{Ax} = \mathbf{y}$ does, and we denote this solution by $\hat{\mathbf{x}}$. Define $\mathbf{b} = \mathbf{y} + \mathbf{e}$, where $\mathbf{y}^{\mathrm{T}}\mathbf{e} = 0$ (this is just the number 0, where we recall that the dot product of two vectors can be represented using transpose). In fact, $\mathbf{a}_1^{\mathrm{T}}\mathbf{e} = \mathbf{a}_2^{\mathrm{T}}\mathbf{e} = 0$. Equivalently, $\mathbf{A}^{\mathrm{T}}\mathbf{e} = 0$. Then, one can prove that

$$\mathbf{A}^{\mathrm{T}}\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^{\mathrm{T}}\mathbf{b}.$$

These are known as the normal equations. If $\mathbf{A}$ is full rank and overdetermined, then $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ is positive definite. The Cholesky algorithm (recall from Chapter 2.6) can be used to solve the normal equations. Moreover, $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ will be invertible, so the solutions can be expressed as

$$\hat{\mathbf{x}} = \left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}\mathbf{b}.$$

The matrix $\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)^{-1}\mathbf{A}^{\mathrm{T}}$ is called the pseudoinverse of $\mathbf{A}$ and it is denoted by $\mathbf{A}^{+}$. If $\mathbf{A}$ is not overdetermined and is of full rank, we have $\mathbf{A}^{+}\mathbf{A} = \mathbf{I}$, but $\mathbf{A}\mathbf{A}^{+} \neq \mathbf{I}$.

Example 3.2 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.** Solve the normal equations directly to find the least squares solution for the following inconsistent system. Also compute the norm of the error.

(a)

$$\begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1 \\ 1 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 2 \\ 1 & 1 & 1 \\ 2 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 3 \\ 1 \\ 2 \end{pmatrix}$$

Solution.

(a) We are given that $\mathbf{Ax} = \mathbf{b}$. Consider the normal equations $\mathbf{A}^{\mathrm{T}}\mathbf{Ax} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$. So, we have $\mathbf{x} = \left(-\frac{1}{7}, \frac{10}{7}\right)$. The residual error is $\mathbf{b} - \mathbf{Ax} = \left(\frac{2}{7}, -\frac{3}{7}, \frac{1}{7}\right)$ which has norm $\frac{\sqrt{14}}{7}$.

(b) One can solve the normal equations to obtain $\mathbf{x} = (1, -1, 1)$. The residual error is $(0, 0, 0, 0)$ which has norm 0. In fact, this means that the original linear system is consistent! □

As mentioned earlier, least squares problems are commonly encountered when fitting a linear model to data. Suppose we have n observations of two variables $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$, and we think that there is a linear relationship between the two. So, we construct the following system of linear equations:

$$y_i = \beta_1 x_i + \beta_0 \quad \text{where } 1 \leq i \leq n$$

In fact, these relationships usually do not hold exactly, and we define the errors to be $\varepsilon_i = y_i - \beta_1 x_i + \beta_0$. We can express these relationships in terms of vectors — or rather, a *compact-looking* system of linear equations as follows:

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_1 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_0 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

As such, we have the problem $\mathbf{y} = \mathbf{X}\beta + \varepsilon$. On the other hand, if we have many independent variables, we still have a similar looking system of linear equations, just that β_0, β_1 *extend* to $\beta_0, \beta_1, \beta_2, \ldots, \beta_k$ and the matrix $\mathbf{X}$ has more columns.

Example 3.3 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.** Consider the data points $(-2, 4), (-1, 1), (1, 1), (2, 4)$.

(a) Find the least-squares line of best fit to this data.

(b) Find the least-squares quadratic polynomial of best fit to this data.

(c) Find the least-squares cubic polynomial of best fit to this data.

Solution.

(a) The equation of a line is $y = mx + c$. The first point yields the equation $-2m + c = 4$, and we repeat this till the 4th data point. Hence, we can construct the linear system

$$\begin{pmatrix} -2 & 1 \\ -1 & 1 \\ 1 & 1 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} m \\ c \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 1 \\ 4 \end{pmatrix}.$$

This has solution $m = 0$ and $c = 2.5$. So the least squares line of best fit is $y = 2.5$.

(b) The equation of a quadratic polynomial is $y = ax^2 + bx + c$. The first point yields the equation $4a - 2b + c = 4$, and again, we repeat this process till the 4th data point. Hence, we can construct the linear system

$$\begin{pmatrix} 4 & -2 & 1 \\ 1 & -1 & 1 \\ 1 & 1 & 1 \\ 4 & 2 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 1 \\ 4 \end{pmatrix}.$$

One can solve the system to obtain $a = 1$ and $b = c = 0$. So, the least squares quadratic polynomial of best fit is $y = x^2$.

(c) One can work out that the answer is still the same as in **(b)**, which is $y = x^2$. □

Example 3.4 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.** Find the least squares solution of the inconsistent system

$$\begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 6 \end{pmatrix}.$$

How can you interpret the result in the data-fitting context?

Solution. The normal equations yield the matrix equation

$$\begin{pmatrix} 3 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 12 \\ 0 \end{pmatrix}.$$

Hence, $x_1 = 4$ and $x_2 \in \mathbb{R}$. Hence, there are infinitely many least squares solutions. In the context of data-fitting, this means the best line for fitting the points $(0, 1)$, $(0, 5)$, and $(0, 6)$, has equation $y = 4 + cx$, where $c \in \mathbb{R}$. There are infinitely many best lines for fitting the three points; all of them have y-intercept at 4, but with different gradient. □

Example 3.5 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.** Assume that the height of a model rocket is measured at four times, and the measured times and heights are $(t, h) = (1, 135), (2, 265), (3, 385), (4, 485)$, in seconds and meters. Fit the model $h = a + bt - 4.905t^2$ to estimate the eventual maximum height of the object and when it will return to Earth.

Solution. We shall find the least squares solution to the equation

$$\begin{aligned} a + b \cdot 1 - 4.905 \cdot 1^2 &= 135 \\ a + b \cdot 2 - 4.905 \cdot 2^2 &= 265 \\ a + b \cdot 3 - 4.905 \cdot 3^2 &= 385 \\ a + b \cdot 4 - 4.905 \cdot 4^2 &= 485 \end{aligned}$$

This yields the normal equations

$$\begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 1417.15 \\ 4250.50 \end{pmatrix}.$$

This gives the least squares solution $(a, b) = (0.475, 141.525)$. Hence, the best fitted model has equation $h = 0.475 + 141.525t - 4.905t^2$. To find the maximum height, we take the derivative of h with respect to t. This yields $t = 14.426$ seconds, so the maximum height of the model rocket is 1020 m. The height of the ground is $h(0) = 0.475$. We see that the time the model rocket returns to Earth is 28.9 seconds. □

We have the system $\mathbf{Ax} = \mathbf{b}$ which has no solution. Instead, we seek to minimise $\|\mathbf{Ax} - \mathbf{b}\|_2^2$. Recall that $\|\mathbf{x}\|_2^2 = \mathbf{x}^{\mathrm{T}}\mathbf{x}$ so by some algebraic manipulation, one can deduce that

$$\|\mathbf{Ax} - \mathbf{b}\|_2^2 = \mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{Ax} - 2\mathbf{x}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}\mathbf{b} + \mathbf{b}^{\mathrm{T}}\mathbf{b}.$$

We say that this is a multivariable quadratic polynomial. In fact, any multivariable quadratic polynomial can be written as

$$\mathbf{x}^{\mathrm{T}}\mathbf{Qx} - 2\mathbf{x}^{\mathrm{T}}\mathbf{r} + s,$$

where $\mathbf{Q}$ is symmetric and positive definite. By comparison, we see that $\mathbf{Q} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$, $\mathbf{r} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$, and $s = \mathbf{b}^{\mathrm{T}}\mathbf{b}$.

Example 3.6. Let

$$p(u, v) = u^2 + 3uv - v^2 + u + v - 2.$$

Also, let $\mathbf{x} = (u, v)$. Then we see that

$$\mathbf{Q} = \begin{pmatrix} 1 & \frac{3}{2} \\ \frac{3}{2} & -1 \end{pmatrix} \quad \mathbf{r} = -\frac{1}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad s = -2.$$

We have $p(\mathbf{x}) = \mathbf{x}^{\mathrm{T}}\mathbf{Qx} - 2\mathbf{x}^{\mathrm{T}}\mathbf{r} + s$. So, the problem of minimising the norm is the same as the problem of minimising a quadratic polynomial.

3.2 QR Factorisation

For $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$ with $m \geq n$, a QR factorisation is

$$\mathbf{A} = \mathbf{QR},$$

where $\mathbf{Q} \in \mathcal{M}_{m\times n}(\mathbb{R})$ has orthonormal columns and $\mathbf{R} \in \mathcal{M}_{n\times n}(\mathbb{R})$ is upper triangular. Note that a square matrix $\mathbf{Q}$ with real entries is said to be orthogonal if $\mathbf{Q}^{\mathrm{T}}\mathbf{Q} = \mathbf{I}$. From here,

$$\|\mathbf{Qv}\|_2^2 = (\mathbf{Qv})^{\mathrm{T}}(\mathbf{Qv}) = \mathbf{v}^{\mathrm{T}}\mathbf{Q}^{\mathrm{T}}\mathbf{Qv} = \mathbf{v}^{\mathrm{T}}\mathbf{v} = \|\mathbf{v}\|_2^2$$

so orthogonal matrices preserve norms. Also, note that if $\mathbf{Q}$ is orthogonal, then so is $\mathbf{Q}^{\mathrm{T}}$. We illustrate the process for QR factorisation using the Gram-Schmidt process. Let

$$\mathbf{A} = \begin{pmatrix} \mathbf{a}_1 & \dots & \mathbf{a}_n \end{pmatrix} \quad \text{be of full column rank.}$$

Then, define $r_{11} = \|\mathbf{a}_1\|_2$ and $q_1 = \frac{\mathbf{a}_1}{\|r_{11}\|}$. For $2 \leq j \leq n$, define

$$r_{ij} = \mathbf{q}_i^{\mathrm{T}}\mathbf{a}_j \quad \text{for all } 1 \leq i \leq j-1,$$

and

$$\mathbf{u}_j = \mathbf{a}_j - \sum_{i=1}^{j-1} r_{ij}\mathbf{q}_i \quad r_{jj} = \|\mathbf{u}_j\|_2 \quad \mathbf{q}_j = \frac{\mathbf{u}_j}{r_{jj}}.$$

Then, $\mathbf{Q}$ has orthonormal columns, $\mathbf{R}$ is upper triangular, and $\mathbf{A} = \mathbf{QR}$. Equivalently, $\mathbf{R} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}$ with $r_{ij} = \mathbf{q}_i^{\mathrm{T}}\mathbf{a}_j$ for $i \leq j$.

Example 3.7 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.** Apply Gram-Schmidt orthogonalization to find the (reduced) QR factorization of the following matrices:

(a)

$$\begin{pmatrix} 4 & 8 & 1 \\ 0 & 2 & -2 \\ 3 & 6 & 7 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 4 & -4 \\ -2 & 7 \\ 4 & -5 \end{pmatrix}$$

Hence, use the QR factorization to solve the least squares problem:

(a)

$$\begin{pmatrix} 4 & 8 & 1 \\ 0 & 2 & -2 \\ 3 & 6 & 7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}$$

(b)

$$\begin{pmatrix} 4 & -4 \\ -2 & 7 \\ 4 & -5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 9 \\ 0 \end{pmatrix}$$

Solution.

(a) Perform the Gram-Schmidt process so we obtain an orthogonal matrix

$$\mathbf{Q} = \begin{pmatrix} \frac{4}{5} & 0 & -\frac{3}{5} \\ 0 & 1 & 0 \\ \frac{3}{5} & 0 & \frac{4}{5} \end{pmatrix}.$$

Since $\mathbf{A} = \mathbf{QR}$, then $\mathbf{R} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}$. So,

$$\mathbf{R} = \begin{pmatrix} 5 & 10 & 5 \\ 0 & 2 & -2 \\ 0 & 0 & 5 \end{pmatrix}$$

which is upper triangular. We then solve the mentioned least squares problem. So,

$$\begin{pmatrix} \frac{4}{5} & 0 & -\frac{3}{5} \\ 0 & 1 & 0 \\ \frac{3}{5} & 0 & \frac{4}{5} \end{pmatrix} \begin{pmatrix} 5 & 10 & 5 \\ 0 & 2 & -2 \\ 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}.$$

Hence,

$$\begin{pmatrix} 5 & 10 & 5 \\ 0 & 2 & -2 \\ 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}.$$

So, $x_3 = \frac{1}{5}$ and one can continue solving for the other unknowns. Hence, $(x_1, x_2, x_3) = \left(-\frac{1}{5}, \frac{1}{5}, \frac{1}{5}\right)$.

(b) Again, we perform the Gram-Schmidt process to obtain

$$\mathbf{Q} = \begin{pmatrix} \frac{2}{3} & \frac{14}{3\sqrt{185}} \\ -\frac{1}{3} & \frac{38}{3\sqrt{185}} \\ \frac{2}{3} & \frac{\sqrt{5}}{3\sqrt{37}} \end{pmatrix}.$$

Since $\mathbf{R} = \mathbf{Q}^{\mathrm{T}}\mathbf{A}$, we have

$$\mathbf{R} = \begin{pmatrix} 6 & -\frac{25}{3} \\ 0 & \frac{\sqrt{185}}{3} \end{pmatrix}$$

which is upper triangular. However, one can then deduce that the linear system has no solution. □

Example 3.8 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.** Consider the over-determined system of linear equations

$$\begin{pmatrix} 1 & 2 \\ 1 & 1 \\ 2 & 1 \\ 2 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \\ 3 \\ 2 \end{pmatrix}.$$

(a) Find the least squares solution to this system by solving the normal equations.

(b) Find the least squares solution to this system by factoring $\mathbf{A} = \mathbf{QR}$.

Solution.

(a) Consider $\mathbf{A}^{\mathrm{T}}\mathbf{A}\mathbf{x} = \mathbf{A}^{\mathrm{T}}\mathbf{b}$ so we obtain the system

$$\begin{pmatrix} 10 & 9 \\ 9 & 10 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 19 \\ 17 \end{pmatrix}.$$

So, the solution to the system is $(x, y) = \left(\frac{37}{19}, -\frac{1}{19}\right)$.

(b) We would obtain the same solution as in **(a)**. □

Example 3.9 (DSA2102 AY25/26 Sem 1 Tutorial 4)**.**

(a) Define $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3 \in \mathbb{R}^4$ by

$$\mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \quad \mathbf{v}_2 = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{v}_3 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Use the Gram-Schmidt process to find an orthonormal basis for the span of these vectors.

(b) Suppose that $\mathbf{v}_1, \ldots, \mathbf{v}_k$ are a linearly independent set of vectors in $\mathbb{R}^n$. How many arithmetic operations (excluding square roots) are required to apply the classical Gram-Schmidt process to these vectors?

Solution.

(a) The orthonormal basis is

$$\left\{ \frac{1}{\sqrt{3}} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\}.$$

(b) By the classical Gram-Schmidt, we really mean to find an orthonormal basis. We initialise with $\mathbf{u}_1 = \mathbf{v}_1$, then apply the recursive formula

$$\mathbf{u}_k = \mathbf{v}_k - \sum_{j=1}^{k-1} \frac{\mathbf{v}_k \cdot \mathbf{u}_j}{\|\mathbf{u}_j\|^2} \mathbf{u}_j \quad \text{for all } k \geq 2.$$

The dot product $\mathbf{v}_i \cdot \mathbf{u}_j$ requires n multiplications and $n-1$ additions. The norm squares $\|\mathbf{u}_j\|^2$ also requires n multiplications and $n-1$ additions. The division for quotient requires 1 division. So, computing one coefficient

$$\frac{\mathbf{v}_i \cdot \mathbf{u}_j}{\|\mathbf{u}_j\|^2}$$

requires $4n-1$ operations. This coefficient is a scalar.

We must compute $i-1$ projections, each with $6n-1$ operations. We then compute $\|\mathbf{u}_i\|^2$, which needs n multiplications and $n-1$ additions. Normalising needs n divisions. So, the grand total at each step is

$$(i-1)(4n-1) + (3n-1).$$

Summing over all $1 \leq i \leq k$ yields the total number of operations, which is

$$(4n-1)\frac{k(k-1)}{2} + k(3n-1)$$

□

3.3 The Householder Reflection

While the Gram-Schmidt process is the typical algorithm introduced in a Linear Algebra class (like MA2001), several other algorithms are commonly used for solving least squares problems. For example, we can perform QR factorisation via

Householder reflections or Givens rotations, or use singular value decomposition (SVD).

The Householder method is the standard implementation for the QR factorisation in most systems. Givens rotations are used for computing the QR factorisation when the matrix has special structure. They are also used in combination with Householder reflections to solve eigenvalue problems. Singular value decomposition on the other hand is the most stable but also the slowest method.

Recall that orthogonal matrices preserve the Euclidean 2-norm. We are interested in transformations in $\mathbb{R}^2$ that preserve the Euclidean length of vectors. They are namely reflections and rotations. Note that the Gram-Schmidt process proceeds via projections, which generally do not preserve norms.

To project $\mathbf{a}$ onto the span of $\mathbf{b}$, we compute

$$\frac{\mathbf{a}\cdot\mathbf{b}}{\|\mathbf{b}\|}\frac{\mathbf{b}}{\|\mathbf{b}\|}=\frac{\mathbf{b}\mathbf{b}^{\mathrm{T}}}{\mathbf{b}^{\mathrm{T}}\mathbf{b}}\mathbf{a}.$$

In general, to project $\mathbf{a}$ onto the range space of a linear transformation with matrix representation $\mathbf{M}$, we compute $\mathbf{M}\left(\mathbf{M}^{\mathrm{T}}\mathbf{M}\right)^{-1}\mathbf{M}^{\mathrm{T}}$. To see why, note that the columns of $\mathbf{M}$ span the range space. So, for any $\mathbf{v}\in R\left(\mathbf{M}\right)$, there exists $\mathbf{u}$ such that $\mathbf{M}\mathbf{u}=\mathbf{v}$. One can then show that $\mathbf{M}\left(\mathbf{M}^{\mathrm{T}}\mathbf{M}\right)^{-1}\mathbf{M}^{\mathrm{T}}\mathbf{v}=\mathbf{v}$. Moreover, if $\mathbf{w}$ is orthogonal to the range space of $\mathbf{M}$, then it is contained in the null space of $\mathbf{M}^{\mathrm{T}}$, and hence also in the null space of $\mathbf{M}\left(\mathbf{M}^{\mathrm{T}}\mathbf{M}\right)^{-1}\mathbf{M}^{\mathrm{T}}$.

Let $\mathbf{P}$ be a projection matrix. It is known that $\mathbf{P}$ is idempotent. That is, $\mathbf{P}^2=\mathbf{P}$. Also, $\mathbf{I}-\mathbf{P}$ projects onto the complementary subspace. A simple example of a projection in $\mathbb{R}^2$ is the projection onto the x-axis, which is given by

$$\mathbf{P}_{x\text{-axis}}=\begin{pmatrix}1&0\\0&0\end{pmatrix}. \tag{3.1}$$

For example, applying $\mathbf{P}_{x\text{-axis}}$ to the vector $(4,3)$ results in $(4,0)$. As such, we have *moved* the vector to the x-axis, but we have changed its length. We now motivate the Householder reflection. How can we move the vector $(4,3)$ to the x-axis without changing its length? One way to accomplish this is to reflect the

vector through the line that bisects the angle it makes with the axis. To defien the line, we need to find a vector perpendicular to it. If we choose the length of $\mathbf{v}$ carefully, then subtracting $\mathbf{v}$ from our original vector $\mathbf{x}$ will be the same as projecting onto the line, and subtracting $\mathbf{v}$ again will accomplish the reflection. We omit the full geometrical details though.

For a non-zero vector $\mathbf{v}$, the projection onto the span of $\mathbf{v}$ is

$$\frac{\mathbf{v}\mathbf{v}^{\mathrm{T}}}{\mathbf{v}^{\mathrm{T}}\mathbf{v}} \quad \text{and} \quad \text{onto the complement is } \mathbf{I} - \frac{\mathbf{v}\mathbf{v}^{\mathrm{T}}}{\mathbf{v}^{\mathrm{T}}\mathbf{v}}.$$

Define

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^{\mathrm{T}}}{\mathbf{v}^{\mathrm{T}}\mathbf{v}}. \tag{3.2}$$

Note that $\mathbf{H}$ is symmetric and orthogonal. We call $\mathbf{H}$ the Householder matrix.

Now, we use this idea to form the QR factorisation of a matrix. We first find a Householder matrix $\mathbf{H}_1$ with the property that $\mathbf{H}_1\mathbf{a}_1 = \alpha_1\mathbf{e}_1$, where $\mathbf{e}_1$ denotes the first standard basis vector. So, we obtain

$$\mathbf{H}_1\mathbf{A} = \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & * & * \\ 0 & * & * \end{pmatrix}.$$

Then, we want to zero out the entries below the diagonal in column 2, but we must not mess up the zeros we already created in column 1. As such, we con consider the Householder matrix

$$\mathbf{H}_2 = \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{S}_2 \end{pmatrix},$$

where the top-left 1 means the first row/column is unaffected, and $\mathbf{S}_2$ is a smaller Householder matrix acting only on rows 2 through m. We choose $\mathbf{S}_2$ so that it transforms the subcolumn

$$\begin{pmatrix} * \\ * \\ * \end{pmatrix} \quad \text{into} \quad \begin{pmatrix} * \\ 0 \\ 0 \end{pmatrix}.$$

After applying $\mathbf{H}_2$, the first two columns are in the desired form. We keep constructing Householder matrices $\mathbf{H}_3, \mathbf{H}_4, \ldots$, each time acting on a smaller

trailing submatrix, until we obtain

$$\text{an upper triangular matrix} \quad \mathbf{R} = \mathbf{H}_k \dots \mathbf{H}_2 \mathbf{H}_1 \mathbf{A}.$$

Since each Householder matrix $\mathbf{H}_i$ is orthogonal, then the product $\mathbf{Q} = \mathbf{H}_1 \dots \mathbf{H}_k$ is also orthogonal, and we have $\mathbf{A} = \mathbf{QR}$. It is easier to illustrate how we can apply the Householder transformation to QR factorisation with an example, so consider Example 3.10.

Example 3.10 (Householder transformation)**.** Let

$$\mathbf{A} = \begin{pmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{pmatrix}.$$

We wish to find an orthogonal matrix $\mathbf{Q}$ and an upper triangular matrix $\mathbf{R}$ such that $\mathbf{A} = \mathbf{QR}$. Note that $\|\mathbf{a}_1\|$, so to preserve norms, we need to reflect $\mathbf{a}_1$ onto $(3, 0, 0)$. Set $\mathbf{v}_1 = (3, 0, 0) - (1, 2, 2) = (2, -2, -2)$. By the definition of the Householder matrix (3.2), we have

$$\mathbf{H}_1 = \begin{pmatrix} \frac{1}{3} & \frac{2}{3} & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & -\frac{2}{3} \\ \frac{2}{3} & -\frac{2}{3} & \frac{1}{3} \end{pmatrix}.$$

One can compute

$$\mathbf{H}_1 \mathbf{A} = \begin{pmatrix} 3 & 2 \\ 0 & -3 \\ 0 & -4 \end{pmatrix}.$$

We then consider the column below the teal entry 2, which is the vector $(-3, -4)$. It has norm 5, so we set $\mathbf{v}_2 = (5, 0) - (-3, -4) = (8, 4)$. By (3.2) again, we can compute

$$\mathbf{H}_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -\frac{3}{5} & -\frac{4}{5} \\ 0 & -\frac{4}{5} & \frac{3}{5} \end{pmatrix}.$$

Then, we have

$$\mathbf{H}_2 \mathbf{H}_1 \mathbf{A} = \begin{pmatrix} 3 & 2 \\ 0 & 5 \\ 0 & 0 \end{pmatrix} = \mathbf{R}.$$

Indeed, $\mathbf{R}$ is an upper triangular matrix. One can compute

$$\mathbf{Q} = \mathbf{H}_1\mathbf{H}_2 = \begin{pmatrix} \frac{1}{3} & -\frac{14}{15} & -\frac{2}{15} \\ \frac{2}{3} & \frac{1}{3} & -\frac{2}{3} \\ \frac{2}{3} & \frac{2}{15} & \frac{11}{15} \end{pmatrix} \quad \text{which is orthogonal.}$$

3.4 The Givens Rotation

Recall the projection matrix which we mentioned in (3.1). We said that we could project the vector $(4, 3)$ but in order to preserve its length, we had to invoke what is known as the Householder transform. Another way to *move* the vector $(4, 3)$ onto the x-axis is by rotating it. Recall from MA2001 that in $\mathbb{R}^2$, a clockwise rotation[†] by θ about the origin O is given by

$$\mathbf{R}(\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

Note that $\mathbf{R}(\theta)$ is an orthogonal matrix. The proof is rather trivial and it involves the classic Pythagorean identity $\sin^2\theta + \cos^2\theta = 1$.

We can extend rotations in $\mathbb{R}^2$ to rotations in $\mathbb{R}^3$. In particular, we shall consider clockwise rotations about the x, y, z-axes as follows:

$$\mathbf{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{pmatrix}$$

$$\mathbf{R}_y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$\mathbf{R}_z(\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Note that every rotation in $\mathbb{R}^3$ can be *clearly* written as a composition of the standard rotations R_x, R_y, R_z. Now, returning to the 2-dimensional case, note that

[†]Recall the matrix representation for an anticlockwise rotation by θ, then use the fact that $\sin\theta$ is an odd function but $\cos\theta$ is an even function.

applying a rotation to the vector (x_1, x_2) leads to the matrix equation

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix}. \tag{3.3}$$

The rotation preserves lengths, i.e. $y^2 = x_1^2 + x_2^2$, and (3.3) can also be written as

$$\begin{pmatrix} x_1 & x_2 \\ x_2 & -x_1 \end{pmatrix} \begin{pmatrix} \cos\theta \\ \sin\theta \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix}.$$

Considering the inverse of the coefficient matrix, one can deduce that

$$\sin\theta = \frac{x_2}{\sqrt{x_1^2 + x_2^2}} \quad \text{and} \quad \cos\theta = \frac{x_1}{\sqrt{x_1^2 + x_2^2}}.$$

Say we wish to rotate $(4, 3)$ onto the x-axis, so we have $\sin\theta = \frac{3}{5}$ and $\cos\theta = \frac{4}{5}$. So, $(4, 3)$ is mapped to the vector $(5, 0)$, thus preserving lengths.

We can extend the idea of rotation to higher dimensions as follows. We wish to use one component of a vector to zero out another component. For example, consider the following rotation in $\mathbb{R}^5$:

$$\mathbf{R}(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \cos\theta & 0 & \sin\theta & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} x_1 \\ \alpha \\ x_3 \\ 0 \\ x_5 \end{pmatrix} \tag{3.4}$$

Then,

$$\cos\theta = \frac{x_2}{\sqrt{x_2^2 + x_4^2}} \quad \text{and} \quad \sin\theta = \frac{x_4}{\sqrt{x_2^2 + x_4^2}}.$$

Similar to the 2-dimensional case, we can rotate any arbitrary vector in $\mathbb{R}^5$, say $(5, 3, -2, 4, 1)$, which gets mapped to $(5, 5, -2, 0, 1)$ (of course, one needs to substitute the values of $\sin\theta$ and $\cos\theta$ in the rotation matrix in (3.4)).

Now, we use this idea to form the QR factorisation of a matrix. If $\mathbf{A}$ is an $m \times n$ matrix with $m \geq n$, we can apply a sequence of what are called *Givens rotations* (one can think of this as a generalisation of rotation matrices) to transform $\mathbf{A}$ into an upper triangular matrix. Note that we need one Givens rotation for each

entry to be zeroed out. In the first column, we use the first entry to zero out the remaining $m-1$ entries, and in the second column, we use the second entry to zero out the remaining $m-2$ entries, and so on. Thus, we obtain

$$\mathbf{G}_k \dots \mathbf{G}_2\mathbf{G}_1\mathbf{A} = \mathbf{R} \quad \text{or equivalently} \quad \mathbf{A} = \mathbf{G}_1^{\mathrm{T}} \dots \mathbf{G}_k^{\mathrm{T}}\mathbf{R} = \mathbf{Q}\mathbf{R}.$$

Example 3.11 (Givens rotation). Let

$$\mathbf{A} = \begin{pmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{pmatrix}.$$

Recall that

$$\cos\theta = \frac{x_1}{\sqrt{x_1^2 + x_2^2}} \quad \text{and} \quad \sin\theta = \frac{x_2}{\sqrt{x_1^2 + x_2^2}}.$$

We zero out the 3 entries below the diagonal one at a time. By some trial and error, we construct the Givens rotation

$$\mathbf{G}_1 = \begin{pmatrix} \frac{1}{\sqrt{5}} & \frac{2}{\sqrt{5}} & 0 \\ -\frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{so} \quad \mathbf{G}_1\mathbf{A} = \begin{pmatrix} \sqrt{5} & \frac{2}{\sqrt{5}} \\ 0 & \frac{11}{\sqrt{5}} \\ 2 & 2 \end{pmatrix}.$$

Thereafter, consider the Givens rotation

$$\mathbf{G}_2 = \begin{pmatrix} \frac{\sqrt{5}}{3} & 0 & \frac{2}{3} \\ 0 & 1 & 0 \\ -\frac{2}{3} & 0 & \frac{\sqrt{5}}{3} \end{pmatrix} \quad \text{so} \quad \mathbf{G}_2\mathbf{G}_1\mathbf{A} = \begin{pmatrix} 3 & 2 \\ 0 & \frac{11}{\sqrt{5}} \\ 0 & \frac{2}{\sqrt{5}} \end{pmatrix}.$$

Lastly (and as expected), we consider the Givens rotation

$$\mathbf{G}_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{11}{5\sqrt{5}} & \frac{2}{5\sqrt{5}} \\ 0 & -\frac{2}{5\sqrt{5}} & \frac{11}{5\sqrt{5}} \end{pmatrix} \quad \text{so} \quad \mathbf{G}_3\mathbf{G}_2\mathbf{G}_1 = \begin{pmatrix} 3 & 2 \\ 0 & 5 \\ 0 & 0 \end{pmatrix}$$

which is an upper triangular matrix.

There is another method that is slower than the QR factorisation, but more stable, and it also works even when $\mathbf{A}$ is not full rank. This method is known as singular value decomposition (SVD), which we will discuss in Chapter 4.2.

Chapter 4

Eigenvalue Problems

4.1 Recap on Eigenvalues and Eigenvectors

Definition 4.1 (eigenvalue and eigenvector)**.** Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$. We say that $\mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$ is an eigenvector of $\mathbf{A}$ corresponding to an eigenvalue $\lambda \in \mathbb{C}$ if the equation $\mathbf{Ax} = \lambda\mathbf{x}$ is satisfied.

From here, one defines the characteristic polynomial of a matrix $\mathbf{A}$, which is $p_{\mathbf{A}}(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I})$. Note that every root of $p_{\mathbf{A}} = 0$ is an eigenvalue of $\mathbf{A}$, and every eigenvalue of $\mathbf{A}$ is a root of $p_{\mathbf{A}} = 0$. From MA2001, recall that for any diagonal matrix $\mathbf{D}$, its diagonal entries are its eigenvalues and the associated eigenvectors are the standard basis vectors; for any upper triangular matrix $\mathbf{U}$, the diagonal entries are also its eigenvalues, but the associated eigenvectors are not as obvious as in the diagonal case.

Recall the fundamental theorem of algebra, which states that every polynomial with complex coefficients has a root over the complex numbers. As a consequence, we can factor polynomials over the complex numbers as follows:

$$p(z) = c_n z^n + \ldots + c_1 z + z_0 = c_n(z - \lambda_1)\ldots(z - \lambda_n)$$

where $\lambda_1, \ldots, \lambda_n$ are the eigenvalues of the matrix $\mathbf{A}$ (and recall that p denotes its corresponding characteristic polynomial).

Definition 4.2 (multiplicity)**.** Let $p_{\mathbf{A}}(\lambda)$ denote the characteristic polynomial of $\mathbf{A}$. The algebraic multiplicity is the multiplicity of a root of p, and the geometric multiplicity is the dimension of the eigenspace $\operatorname{null}(\mathbf{A} - \lambda\mathbf{I})$.

Note that the geometric multiplicity of an eigenvalue is always $\leq$ its algebraic multiplicity. If the geometric multiplicity is strictly less than the algebraic multiplicity, we say that the matrix is defective. Note that non-defective matrices are diagonalisable. That is to say,

$$\mathbf{AP} = \mathbf{PD} \quad \text{or equivalently} \quad \mathbf{A} = \mathbf{PDP}^{-1}.$$

Here, $\mathbf{P}$ is an invertible matrix whose columns are the eigenvectors of $\mathbf{A}$. We say that $\mathbf{A}$ is diagonalisable if and only if it has n linearly independent eigenvectors (another condition for a matrix $\mathbf{A}$ to be diagonalisable is that it has n distinct eigenvalues).

If an $n \times n$ matrix $\mathbf{A}$ has n orthogonal eigenvectors, then we say that it is orthogonally diagonalisable. Equivalently, there exists an orthonormal basis of $\mathbb{R}^n$ consisting of eigenvectors of $\mathbf{A}$.

Theorem 4.1 (real spectral theorem)**.** Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$. Then, $\mathbf{A}$ is orthogonally diagonalisable if and only if $\mathbf{A} = \mathbf{A}^{\mathrm{T}}$.

Theorem 4.2 (complex spectral theorem)**.** Let $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{C})$. Then, $\mathbf{A}$ is orthogonally diagonalisable if and only if it is a normal matrix. That is, $\mathbf{A}\mathbf{A}^* = \mathbf{A}^*\mathbf{A}$, where $\mathbf{A}^*$ denotes the conjugate transpose of $\mathbf{A}$.

4.2 Singular Value Decomposition

We now introduce the concept of singular value decomposition. For $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$, we wish to write

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^{\mathrm{T}},$$

where $\mathbf{U} \in \mathcal{M}_{m\times m}(\mathbb{R})$ and $\mathbf{V} \in \mathcal{M}_{n\times n}(\mathbb{R})$ are orthogonal, and $\Sigma \in \mathcal{M}_{m\times n}(\mathbb{R})$ is diagonal. The matrices $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ and $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ are real symmetric and hence orthogonally diagonalisable by the real spectral theorem (Theorem 4.1). Moreover, they are positive semi-definite so their eigenvalues are non-negative. In fact, $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ and $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ share the same non-zero eigenvalues.

The matrices $\mathbf{U}$ and $\mathbf{V}$ come from the respective spectral decompositions

$$\mathbf{A}\mathbf{A}^{\mathrm{T}} = \mathbf{U}\mathbf{D}_1\mathbf{U}^{\mathrm{T}} \quad \text{and} \quad \mathbf{A}^{\mathrm{T}}\mathbf{A} = \mathbf{V}\mathbf{D}_2\mathbf{V}^{\mathrm{T}}$$

The columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ and the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$. The diagonal entries of Σ are called the singular values of $\mathbf{A}$, and they are the square roots of the eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ or $\mathbf{A}\mathbf{A}^{\mathrm{T}}$.

Example 4.1. We find the singular value decomposition of

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

Note that

$$\mathbf{A}\mathbf{A}^{\mathrm{T}} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad \text{and} \quad \mathbf{A}^{\mathrm{T}}\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

The eigenvalues of $\mathbf{A}\mathbf{A}^{\mathrm{T}}$ are 3 and 1 with corresponding eigenvectors $(1,1)$ and $(1,-1)$; the eigenvalues of $\mathbf{A}^{\mathrm{T}}\mathbf{A}$ are 3, 1, and 0, and the corresponding orthogonal eigenvectors are $(1,2,1)$, $(1,0,-1)$, $(1,-1,1)$. As such,

$$\Sigma = \begin{pmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \mathbf{U} = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \quad \mathbf{V} = \begin{pmatrix} \frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \\ \frac{2}{\sqrt{6}} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} \end{pmatrix}.$$

One important application of singular value decomposition is to solve least squares problems. Let $\mathbf{A} \in \mathcal{M}_{m\times n}(\mathbb{R})$ with $m > n$ and consider

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad \text{or equivalently} \quad \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}\mathbf{x} = \mathbf{b}.$$

One can show that

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 = \left\|\Sigma\mathbf{V}^{\mathrm{T}}\mathbf{x} - \mathbf{U}^{\mathrm{T}}\mathbf{b}\right\|_2^2.$$

Set $\mathbf{y} = \mathbf{V}^{\mathrm{T}}\mathbf{x}$ and $\mathbf{z} = \mathbf{U}^{\mathrm{T}}\mathbf{b}$ so that we get

$$\left\|\Sigma\mathbf{V}^{\mathrm{T}}\mathbf{x} - \mathbf{U}^{\mathrm{T}}\mathbf{b}\right\|_2^2 = \|\Sigma\mathbf{y} - \mathbf{z}\|_2^2 = \sum_{i=1}^{m} (\sigma_i y_i - z_i)^2.$$

Only the first n components of $\Sigma\mathbf{y}$ are non-zero, so

$$\sum_{i=1}^{m} (\sigma_i y_i - z_i)^2 = \sum_{i=1}^{n} (\sigma_i y_i - z_i)^2 + \sum_{i=n+1}^{m} z_i^2.$$

We have no control over the second sum, but we can choose y_i so that $\sigma_i y_i = z_i$. Using the definition of $\mathbf{y}$ and $\mathbf{z}$, this is equivalent to choosing $\mathbf{x} = \mathbf{V}\Sigma^{+}U^{\mathrm{T}}\mathbf{b}$. Recall that for an $m \times n$ diagonal matrix Σ, we define Σ^{+} by taking the reciprocals of the non-zero elements.

For a general $\mathbf{A}$ with $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}}$, we take $\mathbf{A}^{+} = \mathbf{V}\mathbf{\Sigma}^{+}\mathbf{U}^{\mathrm{T}}$. Given a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$, we have the following properties:

(i) $\mathbf{x} = \mathbf{A}^{+}\mathbf{b}$ is a solution if any exist

(ii) If there are infinitely many solutions, then $\mathbf{x} = \mathbf{A}^{+}\mathbf{B}$ has minimal norm among them all

(iii) $\mathbf{x} = \mathbf{A}^{+}\mathbf{b}$ is the least squares approximation if no solution exists

(iv) If the least squares approximation is not unique, then $\mathbf{x} = \mathbf{A}^{+}\mathbf{b}$ is the least-norm least squares approximation

4.3 Root Finding Algorithms

We now return to our goal of computing the eigenvalues and eigenvectors of a square matrix. Using the methods learnt in Linear Algebra, we might try to proceed by forming the characteristic polynomial $p_{\mathbf{A}}(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I})$ and computing its roots. For polynomals of degree 4 or smaller, the orots can be found directly via the quadratic, cubic, or quartic formula (the latter two are quite complicated though). For polynomials of degree 5 and higher, a brilliant Norwegian Mathematician by the name of Niels Henrik Abel proved that it is impossible to derive a similar formula. This result is known as the Abel-Ruffini theorem, and he gave a proof of it in 1823 — he was only 21 years old. In practice, we use iterative methods to find roots. In fact, polynomial root finding is a special case of the more general problem of solving non-linear equations, including systems of non-linear equations.

Some examples of non-linear equations pop up in Physics — for example, in the ideal gas law and Newton's law of gravitation. The ideal gas law states that

$$pV = nRT,$$

where p denotes the pressure (Pa), V denotes the volume (m^3), n denotes the amount of substance (mol), R denotes the universal gas constant, and T denotes the absolute temperature (K). Mathematically, we can express the law as the zero set of a smooth map. That is,

$$F : \mathbb{R}^4_{>0} \to \mathbb{R} \quad \text{where} \quad F(p, V, n, T) = pV - nRT.$$

As such, the law corresponds to finding the pre-image of the singleton set $\{0\}$ (formally, we call this the *fibre*), i.e.

$$F^{-1}(\{0\}) = \{(p, V, n, T) : pV - nRT = 0\}.$$

In general, we see that such problems involve considering $f : \mathbb{R}^n \to \mathbb{R}^m$ where $f(\mathbf{x}) = \mathbf{0}$. For example, if we wish to solve the equation $x^2 = 4 \sin x$, we can set $f(x) = x^2 - 4 \sin x$ and solve $f(x) = 0$.

The existence of solutions to non-linear equations is much more difficult to characterise as compared to linear equations. Having said that, some possible conditions for existence include the intermediate value theorem, the inverse function theorem, and the contraction mapping theorem. Even when solutions exist, the typical behaviour of non-linear equations is non-uniqueness. For example, polynomials can have multiple or isolated roots. Consider the equation $f(x) = x^2 - 2x + 1 = 0$, which has a repeated root at $x = 1$ of multiplicity 2. When we compute $f'(x)\, 2 = (x - 1)$, we see that the root of $f(x) = 0$, $x = 1$, is also a root of the derivative.

Our first method for solving non-linear equations, known as the bisection method, relies on the intermediate value theorem. The method returns an interval rather than a number, but this makes sense when working on a computer because solutions to $f(x) = 0$ may not exist in finite-precision arithmetic even when they exist in $\mathbb{R}$. Recall that one version of the intermediate value theorem states that if $f : [a, b] \to \mathbb{R}$ is a continuous function and changes sign on the interval $[a, b]$, then it must have a root in that interval. We start with an initial interval $[a_0, b_0]$, where the polarities of $f(a_0)$ and $f(b_0)$ are different. We then evaluate f at the midpoint m_0, where $m_0 = \frac{a_0 + b_0}{2}$. We again check the signs — if $f(a_0)$ and $f(m_0)$ have different signs, then $[a_0, m_0]$ becomes our new interval, and we choose $[m_0, b_0]$ otherwise. We continue this process of narrowing the interval.

One alternative to the bisection method is fixed-point iteration. We define x to be a fixed point of a function g if $g(x) = x$. Some problems $f(x) = 0$ can be recast as $g(x) = x$. For example, if

$$f(x) = x^2 - x - 2 = 0,$$

then we can set $x^2 - 2 = x$, so we can take $g(x) = x^2 - 2$. Alternatively, we can set $x^2 = x + 2$ so $x = 1 + \frac{2}{x}$, which implies we can take $g(x) = 1 + \frac{2}{x}$. From here, we observe that a given problem may have many different interpretations as a fixed point problem.

For the fixed-point iteration approach, we first choose some initialisation x_0, then let $x_{k+1}g\left(x_k\right)$. Define the error at step k to be

$$e_{k+1} = x_{k+1} - x = g\left(x_k\right) - g\left(x\right).$$

By the mean value theorem, there exists θ_k between x_k and x such that

$$g\left(x_k\right) - g\left(x\right) = g'\left(\theta_k\right)\left(x_k - x\right)$$

so the error is $e_{k+1} = g'\left(\theta_k\right)e_k$. If $\left|g'\left(x\right)\right| < 1$, then we have convergence. Note that if $g'\left(x\right) = 0$, then by Taylor's theorem, we have

$$g\left(x_k\right) - g\left(x\right) = g''\left(\zeta_k\right) \cdot \frac{\left(x_k - x\right)^2}{2} \quad \text{where } \zeta_k \text{ is between } x_k \text{ and } x.$$

This would lead to more rapid convergence.

There is a more advanced version of fixed-point iteration known as Newton's method. This method can also be extended to functions of several variables. The intuition behind Newton's method is as follows. For a fixed x, we can approximate f using a linear function of h via

$$f\left(x + h\right) \approx f\left(x\right) + f'\left(x\right)h.$$

We are looking for the roots of this function, so

$$h = -\frac{f\left(x\right)}{f'\left(x\right)}.$$

Our procedure is to make an initial guess x_0, compute h, set $x_1 = x_0 - h$ and repeat. Newton's method is a special type of fixed-point iteration with $g\left(x\right) = x - h$. So, $g\left(x\right) = x$ if and only if $f\left(x\right) = 0$ with the assumption that $f'\left(x\right) \neq 0$. Note that this method requires us to compute derivatives. For Newton's method, we have

$$g\left(x\right) = x - \frac{f\left(x\right)}{f'\left(x\right)} \quad \text{so} \quad g'\left(x\right) = \frac{f\left(x\right)f''\left(x\right)}{\left[f'\left(x\right)\right]^2}.$$

If a is a simple root, then $f\left(a\right) = 0$ and $f'\left(a\right) \neq 0$ so $g'\left(a\right) = 0$. By Taylor's theorem, this implies that convergence to simple roots should be quadratic.

The three methods we discussed, namely bisection, fixed-point iteration, and the

Newton-Raphson method, allow us to only find one root of a function. What if we want to find all the zeros of a polynomial? We could use one of the methods from before to find a root a, then consider the function $\frac{p(x)}{x-a}$ and repeat the process. Alternatively, and what is actually done in practice, is that we form the *companion matrix* (Definition 4.3) and solve the associated eigenvalue problem. For example, let $p(x) = x^3 - 6x^2 + 11x - 6$. Then, the associated companion matrix is

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & 6 \\ 1 & 0 & -11 \\ 0 & 1 & 6 \end{pmatrix}.$$

The eigenvalues of $\mathbf{C}$ are 2, 3, and 1, which are the roots of the polynomial equation $p(x) = 0$. This is used in practice because instead of finding roots one-by-one and performing polynomial division repeatedly (which can accumulate numerical errors), constructing the associated companion matrix allows one to use robust eigenvalue algorithms to find all roots simultaneously, even for higher degree polynomials. We give a definition for the companion matrix of a polynomial (Definition 4.3).

Definition 4.3 (companion matrix)**.** Let

$$p(x) = x^n + c_{n-1}x^{n-1} + \ldots + c_1 x + c_0$$

be a monic polynomial of degree n over $\mathbb{R}$. The companion matrix associated with $p(x)$ is

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 & -c_0 \\ 1 & 0 & 0 & \ldots & 0 & -c_1 \\ 0 & 1 & 0 & \ldots & 0 & -c_2 \\ 0 & 0 & 1 & \ldots & 0 & -c_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & -c_{n-1} \end{pmatrix}.$$

The eigenvalues of $\mathbb{C}$ are the roots of $p(x) = 0$.

4.4 Power Iteration

Our first eigenvalue algorithm is used to compute the largest in magntiude eigenvalue of a square matrix. Despite its limitations and simplicity, this algorithm

is used in practice in important applications, such as Google's PageRank algorithm.

We first introduce a way to compute an eigenvalue if we already know an eigenvector. Let $\mathbf{x}$ be an $n \times 1$ matrix and consider the overdetermined system $\lambda\mathbf{x} = \mathbf{A}\mathbf{x}$. Pre-multiplying both sides by $\mathbf{x}^{\mathrm{T}}$, we can deduce that

$$\lambda = \frac{\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}}{\mathbf{x}^{\mathrm{T}}\mathbf{x}}. \tag{4.1}$$

The fraction in (4.1) is called a Rayleigh quotient. Alternatively, we can write

$$\frac{\langle \mathbf{A}\mathbf{x}, \mathbf{x}\rangle}{\langle \mathbf{x}, \mathbf{x}\rangle} = \lambda.$$

The power iteration algorithm proceeds by applying our matrix to a random initial vector repeatedly. That is to say, we choose an initial vector $\mathbf{v}_0$, and recursively compute $\mathbf{v}_k = \mathbf{A}\mathbf{v}_{k-1}$ for $k = 1, 2, \ldots$. Suppose $\mathbf{A}$ is a non-defective matrix (meaning to say assuming $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$, then $\mathbf{A}$ has n linearly independent eigenvectors) with eigenvalues

$$|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_n|.$$

Then, there exists a basis of eigenvectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$, even though we do not know what these vectors are. So for $\mathbf{v}_0 \in \mathbb{R}^n$, we note that $\mathbf{v}_0 = c_1\mathbf{x}_1 + \ldots + c_n\mathbf{x}_n$ for some real coefficients $c_1, \ldots, c_n$. Then,

$$\mathbf{A}\mathbf{v}_0 = c_1\lambda_1\mathbf{x}_1 + \ldots + c_n\lambda_n\mathbf{x}_n.$$

Continuing to apply $\mathbf{A}$, we obtain

$$\mathbf{A}^k\mathbf{v}_0 = \sum_{i=1}^{n} c_i\lambda_i^k\mathbf{x}_i = \lambda_1^k\left(c_1\mathbf{x}_1 + \sum_{i=1}^{n} c_i\left(\frac{\lambda_i}{\lambda_1}\right)^k\mathbf{x}_i\right).$$

That is, under our current setup, we have $\mathbf{v}_k \mapsto c_1\lambda_1^k\mathbf{x}_1$, and we can recover the eigenvalue using the Rayleigh quotient. An even better approach is to take

$$\mathbf{w}_k = \mathbf{A}\mathbf{v}_{k-1} \quad \text{and} \quad \mathbf{v}_k = \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|_\infty}.$$

This keeps the components of our vectors to a reasonable magnitude.

Example 4.2 (power iteration)**.** Let

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 5 \end{pmatrix}.$$

Choose $\mathbf{v}_0 = (1, 1, 1)$. Then, one can compute the following:

$$\begin{aligned} \mathbf{A}\mathbf{v}_0 &= (4, 4, 5) \\ \mathbf{A}^2\mathbf{v}_0 &= (17, 17, 25) \\ \mathbf{A}^3\mathbf{v}_0 &= (76, 76, 125) \\ \mathbf{A}^4\mathbf{v}_0 &= (353, 353, 625) \\ \mathbf{A}^5\mathbf{v}_0 &= (1684, 1684, 3125) \end{aligned}$$

Using the method where we divide by the maximum entry at each step, we eventually obtain $\mathbf{A}^{10}\mathbf{v}_0 = (0.503, 0.503, 1)$ which eventually stabilises.

The power iteration algorithm can be adapted in various ways to find other eigenvalues and eigenvectors. For example, if λ is an eigenvalue of $\mathbf{A}$, then $\frac{1}{\lambda}$ is an eigenvalue of $\mathbf{A}^{-1}$. Thus, applying power iteration to $\mathbf{A}^{-1}$ will produce the eigenvector corresponding to the smallest (in magnitude) eigenvalue of $\mathbf{A}$. Similarly, for some scalar σ, the smallest (in magnitude) eigenvalue of $\mathbf{A} - \sigma\mathbf{I}$ is $\lambda - \sigma$, where λ is the closest eigenvalue to σ. These two observations are combined to form the inverse iteration algorithm, defined as follows:

$$\mathbf{v}_{k+1} = (\mathbf{A} - \sigma\mathbf{I})^{-1}\,\mathbf{v}_k$$

Rather than explicitly computing the inverse of $\mathbf{A} - \sigma\mathbf{I}$, we compute the LU factorisation and solve

$$(\mathbf{A} - \sigma\mathbf{I})\,\mathbf{v}_{k+1} = \mathbf{v}_k \text{ at each step.}$$

At the end, we once again make use of the Rayleigh quotient to compute the eigenvalue. Note that the closer σ is to an eigenvalue of $\mathbf{A}$, the faster inverse iteration will converge. We can update the shift σ at each stage to speed up convergence. We do this by combining inverse iteration with Rayleigh quotients in an algorithm known as the Rayleigh quotient iteration.

Theorem 4.3 (Rayleigh quotient iteration). Suppose we are given $\mathbf{A} \in \mathcal{M}_{n\times n}(\mathbb{R})$. For $k = 0, 1, 2, \ldots$, we first normalise $\mathbf{v}_k$ to obtain

$$\mathbf{v}_k \mapsto \frac{\mathbf{v}_k}{\|\mathbf{v}_k\|_\infty}.$$

The Rayleigh quotient is defined to be

$$\sigma_{k+1} = \frac{\mathbf{v}_k^{\mathrm{T}} \mathbf{A} \mathbf{v}_k}{\mathbf{v}_k^{\mathrm{T}} \mathbf{v}_k}.$$

We then solve the shifted system

$$(\mathbf{A} - \sigma_k \mathbf{I})\, \mathbf{w}_{k+1} = \mathbf{v}_k \quad \text{and} \quad \text{take } \mathbf{v}_{k+1} = \frac{\mathbf{w}_k}{\|\mathbf{w}_k\|_\infty}.$$

Example 4.3. Consider the matrix

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 3 & 1 \\ 0 & 0 & 5 \end{pmatrix}.$$

Choose $\sigma = -1$ and $\mathbf{v}_0$ at random. Then,

$$\mathbf{A} - \sigma\mathbf{I} = \begin{pmatrix} 3 & 1 & 1 \\ 0 & 4 & 1 \\ 0 & 0 & 6 \end{pmatrix} \quad \text{so} \quad (\mathbf{A} - \sigma\mathbf{I})^{-1} = \begin{pmatrix} 0.33333 & -0.08333 & -0.04167 \\ 0.00000 & 0.25000 & -0.04167 \\ 0.00000 & 0.00000 & 0.16667 \end{pmatrix}.$$

Let $\mathbf{B} = (\mathbf{A} - \sigma\mathbf{I})^{-1}$. Note that because $\mathbf{A}$ is upper triangular, its eigenvalues are its diagonal entries 2, 3, and 5, and because $\sigma = -1$ is closest to 2, our recursive algorithm should converge to the $\lambda = 2$ eigenvector. Choose $\mathbf{v}_0 = (1, 0.9, 1)$ and normalise it by $\|\cdot\|_\infty$. So,

$$\mathbf{w}_1 = \mathbf{B}\mathbf{v}_0 = \begin{pmatrix} 0.21766 \\ 0.21333 \\ 0.16666 \end{pmatrix} \quad \text{so} \quad \mathbf{v}_1 = \frac{\mathbf{w}_1}{\|\mathbf{w}_1\|_\infty} = \begin{pmatrix} 1 \\ 0.84615 \\ 0.76923 \end{pmatrix}.$$

Let

$$\sigma_1 = \frac{\mathbf{v}_1^{\mathrm{T}} \mathbf{A} \mathbf{v}_1}{\mathbf{v}_1^{\mathrm{T}} \mathbf{v}_1} = 4.0615.$$

Repeat this process to compute values of σ_k for $k = 2, 3, 4, \ldots$. For example, $\sigma_2 = 3.7226$, $\sigma_3 = 3.3827$, $\sigma_4 = 3.0840$. One can deduce that the Rayleigh quotient σ_k tends towards 2, as expected.

4.5 QR Iteration

In practice, many techniques are used to speed up the convergence of this algorithm. One important technique is to first convert the matrix to a special form. A matrix is called *upper Hessenberg* if all of its entries below the first subdiagonal are 0. That is,

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ 0 & a_{32} & a_{33} & a_{34} & a_{35} \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix}.$$

So, an upper Hessenberg matrix is almost upper-triangular. A symmetric upper Hessenberg matrix is *tridiagonal*. That is, of the form

$$\begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} \\ 0 & 0 & 0 & a_{54} & a_{55} \end{pmatrix}.$$

We need a way to transform a matrix to upper Hessenberg form that preserves its eigenvalues. This can be accomplished with Householder reflections.

Example 4.4. Consider the matrix

$$\mathbf{B} = \begin{pmatrix} 2 & 1 & 1 \\ 3 & 2 & 5 \\ 4 & 4 & 1 \end{pmatrix}.$$

We know that the following matrix will eliminate the last entry of the first column:

$$\mathbf{H} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{3}{5} & \frac{4}{5} \\ 0 & \frac{4}{5} & -\frac{3}{5} \end{pmatrix}$$

Moreover, $\mathbf{HBH}$ has the same eigenvalues as $\mathbf{B}$ and

$$\mathbf{HBH} = \begin{pmatrix} 2.00 & 1.40 & 0.20 \\ 5.00 & 5.68 & 1.24 \\ 0 & 2.24 & -2.68 \end{pmatrix}$$

Once a matrix is in upper Hessenberg form, the QR factorisation can be computed via Givens rotation.

Chapter 5

Interpolation and Approximation

5.1 Polynomial Interpolation

Interpolation involves fitting a function to some data points. In contrast to least squares regression, in interpolation problems, we want our function to match the data exactly. For example, say we are given the population of Singapore in $1960, 1965, 1970, \ldots, 2020$, how can we estimate the population of Singapore in 1997?

The general interpolation problem can be phrased as follows. For an unknown function $f(x)$, given some data points on the graph of $f(x)$ say $(x_0, y_0), \ldots, (x_n, y_n)$, how can we recover the original function $f(x)$? Alternatively, given x, how can we guess the value of $f(x)$?

In contrast to predicting the population of a city, we take a look at another example. Consider a specific case of the gamma function

$$g(x) = \int_x^\infty e^{-t} t^{-1/2}\, dt \quad \text{and} \quad \text{its approximation } \sqrt{\pi} - \sum_{k=0}^{N} \frac{(-1)^k\, x^{k+\frac{1}{2}}}{k!\left(k+\frac{1}{2}\right)}.$$

The approximation is inefficient for large x. Instead, one can try to interpolate the function.

To really have a fruitful discussion on polynomial interpolation, we first need to formulate the problem more precisely. The approximation is usually a function with a number of parameters, say

$$(ax+b)\sin(cx+d)\exp\left(\frac{ex+f}{gx+h}\right).$$

There are several criteria we would like this to satisfy, namely the formula should be determined by our prior knowledge of the function, the parameters are to be determined by the data points, the function must not be difficult to evaluate (the idea of simplicity), and the formula must be able to approximate a sufficiently

wide range of functions (approximability).

Some common choices for approximating functions are as follows:

(i) Trigonometric functions[†] which have the general form

$$a_0 + \sum_{k=1}^{n} \left[a_k \cos(kx) + b_k \sin(kx)\right]$$

(ii) Rational functions which have the general form

$$\frac{a_0 + a_1 x + \ldots + a_m x^m}{b_0 + b_1 x + \ldots + b_n x^n}$$

(iii) Polynomials which have the general form $a_0 + a_1 x + \ldots + a_m x^m$

Here, we will only focus on polynomial interpolation. We say that a polynomial of degree m is of the form

$$P_m(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m = \sum_{k=0}^{m} a_k x^k. \tag{5.1}$$

When evaluating a polynomial function on a computer, a naïve approach requires m additions and $\frac{m(m+1)}{2}$ multiplications. We discuss Horner's method, which would only yield m additions and m multiplications. To execute this, take a polynomial P_m as in (5.1) and rewrite it as

$$P_m(x) = a_0 + x\left(a_1 + x\left(a_2 + x\left(a_3 + \ldots + x\left(a_{m-1} + x a_m\right)\ldots\right)\right)\right).$$

Horner's method is expected to have only m additions and m multiplications. We can express this method iteratively as follows:

$$p_m = a_m \quad \text{and} \quad p_{m-1} = a_{m-1} + x p_m \text{ and so on.}$$

We then introduce Weierstrass' approximation theorem (Theorem 5.1).

Theorem 5.1 (Weierstrass approximation theorem)**.** Let f be a continuous function on $[a, b]$. For any $\varepsilon > 0$, there exists a polynomial $p(x)$ such that

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

[†]Related to Fourier series.

What Theorem 5.1 is saying is that polynomials can approximate any continuous function defined on a finite closed interval up to any precision. To get higher accuracy, we usually require a polynomial of a higher degree. Unfortunately, the theorem does not guarantee whether $f(x)$ and $P(x)$ agree on any points.

The interpolation problem can be formally stated as follows:

> For an unknown function $f(x)$, suppose the values of $f(x_0), \ldots, f(x_n)$ are given. Find a polynomial $P_d(x)$ of degree d such that $P_d(x_i) = f(x_i)$ for all $0i \leq n$. The points $x_0, \ldots, x_n$ are called nodes and the polynomial P_d is the interpolant or the interpolating polynomial.

Assume that $P_d(x) = a_0 + a_1 x + \ldots + a_d x^d$. We wish to find $a_0, a_1, \ldots, a_d$ such that

$$\begin{aligned} a_0 + a_1 x_0 + \ldots + a_d x_0^d &= f(x_0) \\ a_0 + a_1 x_1 + \ldots + a_d x_1^d &= f(x_1) \\ \vdots &= \vdots \\ a_0 + a_1 x_n + \ldots + a_d x_n^d &= f(x_n) \end{aligned}$$

The above system can be written as $\mathbf{Xa} = \mathbf{f}$, where $\mathbf{a} = (a_0, \ldots, a_d)$, $\mathbf{f} = (f(x_0), \ldots, f(x_n))$, and $\mathbf{X}$ denotes the following Vandermonde matrix in (5.2). Note that it is said to be Vandermonde because the terms in each row form a geometric progression.

$$\mathbf{X} = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^d \\ 1 & x_1 & x_1^2 & \ldots & x_1^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \ldots & x_n^d \end{pmatrix}. \tag{5.2}$$

Given the equation for the coefficients of the interpolant $\mathbf{Xa} = \mathbf{f}$, we would first like to know if the solution exists. If it does, is the solution unique? Next, can we find the vector $\mathbf{a}$ efficiently, and are there any other representations of the interpolating polynomial?

For existence and uniqueness, recall that 2 points uniquely determine a line, 3 points determine a unique quadratic, 4 points determine a unique cubic, and so on. As such, if we are trying to fit a polynomial of degree d to a set of points, a solution

exists as long as there are no more than $d+1$ points, and the solution is unique as long as there are at least $d+1$ points. General questions of existence, uniqueness, and conditioning depend on the basis matrix $\mathbf{X}$. As is typical with vector spaces, there are many possible choices of bases for the collection of polynomials. We begin our discussion with the most familiar basis, also known as the monomial basis.

Let $M_k(x) = x^k$. Note that the functions $M_0(x), \ldots, M_d(x)$ span the space of polynomials of degree at most d. So, we write

$$p(x) = a_0 + a_1 x + \ldots + a_d x^d = a_0 M_0 + a_1 M_1 + \ldots + a_d M_d.$$

The $(n+1) \times (d+1)$ Vandermonde matrix is

$$\mathbf{X} = \begin{pmatrix} M_0(x_0) & M_1(x_0) & \ldots & M_d(x_0) \\ M_0(x_1) & M_1(x_1) & \ldots & M_d(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ M_0(x_n) & M_1(x_n) & M_2(x_n) & \ldots M_d(x_n) \end{pmatrix}.$$

Note that computing $\mathbf{Xa}$ is polynomial evaluation, whereas solving $\mathbf{Xa} = \mathbf{f}$ is polynomial interpolation. If $d = n$, then $\mathbf{Xa} = \mathbf{0}$ implies that the polynomial has $d+1$ roots, but it is only degree d, so $\mathbf{X}$ must be invertible. We illustrate with an example (see Example 5.1).

Example 5.1. Consider the data points $(-2, -27)$, $(0, -1)$ and $(1, 0)$. We wish to fit a quadratic polynomial $p(x) = a_0 + a_1 x + a_2 x^2$. So, we have

$$\begin{pmatrix} 1 & x_0 & x_0^2 \\ 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} \quad \text{so} \quad \begin{pmatrix} 1 & -2 & 4 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -27 \\ -1 \\ 0 \end{pmatrix}.$$

The coefficient matrix is invertible so solving the matrix equation yields the polynomial $p(x) = -1 + 5x - 4x^2$.

We then discuss the method of Lagrange interpolation. To begin our discussion, we consider an alternative set of basis functions. Let $x_0, x_1, \ldots, x_n$ be the interpolation nodes. Assume the functions $\varphi_0(x), \ldots, \varphi_n(x)$ satisfy $\varphi_i(x_j) = 1$ if $i = j$, and 0 otherwise, for all $1 \leq i, j \leq n$. Explicitly, we have

$$\begin{matrix} \varphi_0(x_0) = 1 & \varphi_0(x_1) = 0 & \ldots & \varphi_0(x_n) = 0 \\ \varphi_1(x_0) = 0 & \varphi_1(x_1) = 1 & \ldots & \varphi_1(x_n) = 0 \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_n(x_0) = 0 & \varphi_n(x_1) = 0 & \ldots & \varphi_n(x_n) = 1 \end{matrix}.$$

Then, the function

$$I(x) = \sum_{k=0}^{n} f(x_k)\,\varphi_k(x) \tag{5.3}$$

is an interpolating function for the data points $(x_k, f(x_k))$. We can choose a set of polynomial basis functions with this property, which is known as the Lagrange basis. For $x_0 = 1$, $x_1 = 2$, $x_2 = 3$, and $x_3 = 4$, the following polynomials satisfy the property in (5.3):

$$\begin{aligned}
L_0(x) &= \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} \\
L_1(x) &= \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} \\
L_2(x) &= \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} \\
L_3(x) &= \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)}
\end{aligned}$$

These functions are called the Lagrange basis polynomials.

Definition 5.1 (Lagrange interpolating polynomial)**.** We consider a more general setup. Let $x_0, x_1, \ldots, x_n$ be $n+1$ distinct real numbers. For $k = 0, 1, \ldots, n$, the k^{th} Lagrange basis polynomial $L_k(x)$ is a polynomial of degree n defined by

$$L_k(x) = \prod_{\substack{j=0 \\ j\neq k}}^{n} \frac{x - x_j}{x_k - x_j} = \frac{(x-x_0)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_n)}{(x_k-x_0)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n)}. \tag{5.4}$$

The interpolating polynomial is given by

$$P_n(x) = \sum_{k=0}^{n} f(x_k)\,L_k(x).$$

The basis matrix in this case is the $(n+1)\times(n+1)$ identity matrix $\mathbf{I}$.

Example 5.2 (Lagrange interpolation)**.** Consider the data points $(-2, -27)$, $(0, -1)$ and $(1, 0)$. We seek a polynomial

$$p(x) = y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x).$$

By (5.4), we have

$$p(x) = y_0 \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2 \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.$$

Substituting $x_0, x_1, x_2, y_0, y_1, y_2$ yields $p(x) = -4x^2 + 5x - 1$.

Example 5.3. For a more interesting example, we consider the gamma function

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1}\, dt.$$

Using integration by parts, one can deduce that $\Gamma(n) = (n-1)!$. As such, we shall consider the data points $\Gamma(1) = 1$, $\Gamma(2) = 1$, $\Gamma(3) = 2$, and $\Gamma(4) = 6$. We shall fit a polynomial using the Lagrange basis, so

$$\begin{aligned}
L_0(x) &= \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)} = -\frac{1}{6}(x-2)(x-3)(x-4) \\
L_1(x) &= \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)} = \frac{1}{2}(x-1)(x-3)(x-4) \\
L_2(x) &= \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)} = -\frac{1}{2}(x-1)(x-2)(x-4) \\
L_3(x) &= \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)} = \frac{1}{6}(x-1)(x-2)(x-3)
\end{aligned}$$

So, the Lagrange interpolating polynomial is

$$P_3(x) = L_0(x) + L_1(x) + 2L_2(x) + 6L_3(x).$$

The third common set of basis functions is called the Newton basis. Say we have a set of points (x_i, y_i) where $0 \le i \le n$. The Newton basis functions are defined to be

$$N_i(x) = \prod_{k=0}^{i-1} (x - x_k).$$

The first few basis functions are

$$\begin{aligned}
N_0(x) &= 1 \\
N_1(x) &= x - x_0 \\
N_2(x) &= (x-x_0)(x-x_1)
\end{aligned}$$

and so on. The basis matrix is the lower triangular matrix

$$\begin{pmatrix} 1 & 0 & \dots & 0 \\ 1 & x_1 - x_0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x_0 & \dots & (x_n - x_0) \dots (x_n - x_{n-1}) \end{pmatrix}$$

Example 5.4. Consider the data points $(-2, -27)$, $(0, -1)$, and $(1, 0)$. We seek a polynomial of the form

$$p(x) = a_0 N_0(x) + a_1 N_1(x) + a_2 N_2(x),$$

thus forming the matrix equation

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & x_1 - x_0 & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix}.$$

Hence,

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 3 & 3 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{pmatrix} -27 \\ -1 \\ 0 \end{pmatrix}.$$

The coefficient matrix is invertible so one can deduce that

$$p(x) = -27 + 13(x + 2) - 4(x + 2)x = -1 + 5x - 4x^2.$$

One should note that the Newton basis functions have a nice *incremental* property — if we add new data points, we do not need to redo what we have done earlier. For example, if we start with $(-2, -27)$, our interpolating polynomial is

$$p_0(x) = a_0 N_0(x) = -27.$$

Adding the point $(0, -1)$, our interpolating polynomial is

$$p_1(x) = p_0(x) + a_1 N_1(x) = -27 + 13(x + 2)$$

and one can recursively perform the aforementioned step to new data points in order to obtain new interpolating polynomials.

What happens when the basis matrix is not square? If $m > n$, the equations can be written as

$$\begin{aligned} a_0 + a_1 x_0 + \ldots + a_n x_0^n &= b_0 - a_{n+1} x_0^{n+1} - \ldots - a_m x_0^m \\ a_0 + a_1 x_1 + \ldots + a_n x_1^n &= b_1 - a_{n+1} x_1^{n+1} - \ldots - a_m x_1^m \\ \vdots &= \vdots \\ a_0 + a_1 x_n + \ldots + a_n x_n^n &= b_n - a_{n+1} x_n^{n+1} - \ldots - a_m x_n^m \end{aligned}$$

We can solve for $a_0, a_1, \ldots, a_n$ in terms of $a_{n+1}, \ldots, a_m$. To conclude, the system has infinite solutions.

If $m < n$, the equations can be written as

$$\begin{aligned} a_0 + a_1 x_0 + \ldots + a_m x_0^m &= b_0 \\ a_0 + a_1 x_1 + \ldots + a_m x_1^m &= b_1 \\ \vdots &= \vdots \\ a_0 + a_1 x_m + \ldots + a_m x_m^m &= b_m \\ a_0 + a_1 x_{m+1} + \ldots + a_m x_{m+1}^m &= b_{m+1} \\ \vdots &= \vdots \\ a_0 + a_1 x_n + \ldots + a_m x_n^m &= b_n \end{aligned}$$

The first m equations have already determined the solution. The solution of the complete system exists only if the solution satisfies the last $n - m$ equations.

5.2 Piecewise Interpolation

So far, we have only asked that the interpolating polynomial match the function values at the nodes. What if we also want the polynomial to match the slope of the function at the nodes? Then, we will also need to know $f'(x_0), \ldots, f'(x_n)$. What degree polynomial do we need? We have $2n + 2$ conditions, so the degree of the polynomial would need to be $2n + 1$.

Recall from MA2002 Calculus that given an infinitely differentiable function f, its Taylor series at $x = a$ is

$$f(a) + f'(a)(t - a) + \frac{f''(a)}{2!}(t - a)^2 + \ldots + \frac{f^{(k)}(a)}{k!}(t - a)^k + \ldots.$$

We define the n^{th} Taylor polynomial of f at a to be

$$p_n(t) = f(a) + +f'(a)(t-a) + \frac{f''(a)}{2!}(t-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(t-a)^n .$$

The Taylor polynomial, which can be interpreted as a finite series, has the property that $p(a) = f(a)$, $p'(a) = f'(a)$, $p''(a) = f''(a)$ and so on. Note that we need a polynomial of degree n to match the function value at a and the values of the first n derivatives at a. In what follows, we will use the idea of putting constraints on the derivatives, though we will generally only look at first and second derivatives.

Suppose we are given the data points (x_k, y_k) for $k = 0, 1, \ldots, n$. We would like to find a continuous piecewise linear function $f(x)$ such that $f(x_k) = y_k$ for all $1 \leq k \leq n$ and $f(x)$ is a linear function on (x_{k-1}, x_k) for all $1 \leq k \leq n$.

We first discuss interpolation using piecewise linear functions. Recall that the equation of a line through the points (a, b) and (c, d) is

$$y = b + \frac{d-b}{c-a}(x-a) .$$

As such, we have

$$f(x) = \begin{cases} y_0 + \dfrac{y_1 - y_0}{x_1 - x_0}(x - x_0) & \text{if } x \in [x_0, x_1] ; \\ y_1 + \dfrac{y_2 - y_1}{x_2 - x_1}(x - x_1) & \text{if } x \in (x_1, x_2] ; \\ \vdots & \vdots \\ y_{n-1} + \dfrac{y_n - y_{n-1}}{x_n - x_{n-1}}(x - x_{n-1}) & \text{if } x \in (x_{n-1}, x_n] \end{cases} \tag{5.5}$$

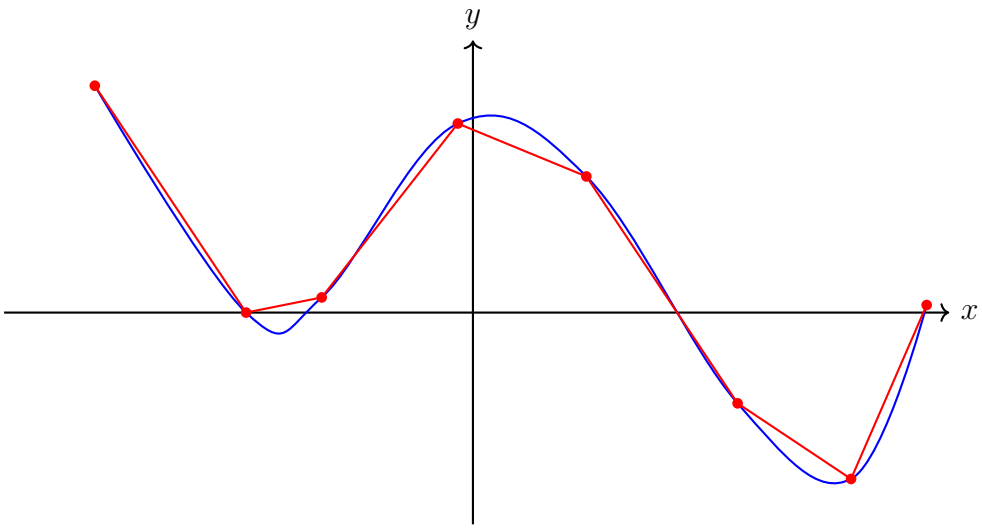

Figure 1: Interpolation using piecewise linear functions

So far, we have only asked that the interpolating polynomial match the function values at the nodes. What if we also want the polynomial to match the slope of the function at the nodes? Then, we will also need to know $f'(x_0), \ldots, f'(x_n)$. What degree polynomial do we need? We have $2n+2$ conditions, so the degree of the polynomial would need to be $2n+1$.

We then discuss piecewise constant interpolation. Given data points $(x_0, y_0), \ldots, (x_n, y_n)$ with nodes ordered increasingly, define for each i

$$a_i = \frac{x_i - x_{i-1}}{2} \quad \text{and} \quad b_i = \frac{x_{i+1} - x_i}{2},$$

and set $f(x) = y_i$ for $a_i \leq x < b_i$. This interpolant is, in general, not continuous.

Example 5.5. For the data $(0,1), (2,3), (4,-2), (6,0)$, we have

$$f(x) = \begin{cases} 1 & \text{if } x < 1; \\ 3 & \text{if } 1 \leq x < 3; \\ -2 & \text{if } 3 \leq x < 5; \\ 0 & \text{if } x \geq 5. \end{cases}$$

This is not a continuous function.

We now consider a slightly different interpolant in contrast to (5.5). Given $(x_0, y_0), \ldots, (x_n, y_n)$, on each interval $[x_i, x_{i+1}]$ use the secant line

$$f(x) = y_{i+1}\frac{x - x_i}{x_{i+1} - x_i} + y_i\frac{x_{i+1} - x}{x_{i+1} - x_i}, \quad \text{where } x_i \le x \le x_{i+1}.$$

This interpolant is continuous but generally not differentiable at the nodes.

Example 5.6. For $(0,1), (2,3), (4,-2), (6,0)$, we have

$$f(x) = \begin{cases} 1 + x & \text{if } 0 \le x \le 2; \\ 3 - \frac{5}{2}(x-2) & \text{if } 2 \le x \le 4; \\ -2 + (x-4) & \text{if } 4 \le x \le 6. \end{cases}$$

We then discuss piecewise quadratic interpolation. The idea is to fit parabolas

$$f_j(x) = a_j x^2 + b_j x + c_j$$

to triples $(x_i, y_i), (x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2})$ via

$$y_i = a_j x_i^2 + b_j x_i + c_j \quad y_{i+1} = a_j x_{i+1}^2 + b_j x_{i+1} + c_j \quad y_{i+2} = a_j x_{i+2}^2 + b_j x_{i+2} + c_j.$$

This is still generally not differentiable at the junctions and is often no better than the piecewise linear interpolant between nodes. Note that to fit n quadratics without extra constraints, one needs $2n + 1$ points.

Example 5.7. For $(0,1), (2,3), (4,-2), (6,0)$, one convenient choice is

$$f_1(x) = 1 + \frac{11}{4}x - \frac{7}{8}x^2 \quad \text{and} \quad f_2(x) = 15 - \frac{31}{4}x + \frac{7}{8}x^2.$$

As for differentiable piecewise quadratic interpolation, the idea is to use two adjacent points per quadratic but enforce derivative matching at the shared node. Let

$$f_i(x) = a_i x^2 + b_i x + c_i.$$

Impose, for consecutive intervals $[x_i, x_{i+1}]$ and $[x_{i+1}, x_{i+2}]$,

$$\begin{aligned} y_i &= a_i x_i^2 + b_i x_i + c_i \\ y_{i+1} &= a_i x_{i+1}^2 + b_i x_{i+1} + c_i \\ y_{i+1} &= a_{i+1} x_{i+1}^2 + b_{i+1} x_{i+1} + c_{i+1} \\ y_{i+2} &= a_{i+1} x_{i+2}^2 + b_{i+1} x_{i+2} + c_{i+1} \\ 2a_i x_{i+1} + b_i &= 2a_{i+1} x_{i+1} + b_{i+1} \end{aligned}$$

We have six unknowns and five equations; add the condition that second derivatives match at the midpoint (equivalently $a_i = a_{i+1}$). We now discuss piecewise cubic spline interpolation. The idea is to fit cubics

$$f_j(x) = a_j x^3 + b_j x^2 + c_j x + d_j,$$

to quadruples by solving

$$\begin{aligned} y_i &= a_j x_i^3 + b_j x_i^2 + c_j x_i + d_j \\ y_{i+1} &= a_j x_{i+1}^3 + b_j x_{i+1}^2 + c_j x_{i+1} + d_j \\ y_{i+2} &= a_j x_{i+2}^3 + b_j x_{i+2}^2 + c_j x_{i+2} + d_j \\ y_{i+3} &= a_j x_{i+3}^3 + b_j x_{i+3}^2 + c_j x_{i+3} + d_j \end{aligned}$$

Same degrees-of-freedom issues as the quadratic case; instead, we can use fewer points per cubic and impose smoothness conditions.

Example 5.8 (unique cubic through four points)**.** For $(0,1), (2,3), (4,-2), (6,0)$, we have

$$f(x) = 1 + \frac{61}{12}x - \frac{21}{8}x^2 + \frac{7}{24}x^3.$$

As for the method of natural cubic splines, we seek $n-1$ cubics $p_i(x)$ on $[x_i, x_{i+1}]$ of the form

$$p_i(x) = y_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3.$$

The conditions are as follows. First, we must interpolate the polynomial so $p_i(x_{i+1}) = y_{i+1}$. As the first and second derivatives must be continuous, then $p'_{i-1}(x_i) = p'_i(x_i)$ and $p''_{i-1}(x_i) = p''_i(x_i)$. Lastly, we must fit the natural endpoints, so $p''(x_0) = p''(x_n) = 0$. A straightforward formulation yields a $4(n-1) \times 4(n-1)$ linear system, but it can be reduced to a tridiagonal system. We omit the details.

5.3 Orthogonal Polynomials

We then introduce the Chebyshev polynomials of the first kind, denoted by $T_n(x)$. These satisfy the recurrence relation

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

They are also orthogonal. That is to say,

$$\int_{-1}^{1} \frac{T_n(x)\, T_m(x)}{\sqrt{1-x^2}}\, dx = 0 \quad \text{for } m \neq n.$$

We now explore the idea of orthogonality of functions in greater detail, and to do so, we need to introduce what is called an *inner product* (Definition 5.2).

Definition 5.2 (inner product)**.** Let V be a vector space over a field F. An inner product is a function $\langle \cdot, \cdot \rangle : V \times V \to F$ that satisfies the following properties:

(i) $\langle \mathbf{v}, \mathbf{v} \rangle \geq 0$ for all $\mathbf{v} \in V$
(ii) $\langle \mathbf{v}, \mathbf{v} \rangle = 0$ if and only if $\mathbf{v} = \mathbf{0}$
(iii) $\langle \mathbf{u} + \mathbf{v}, \mathbf{w} \rangle$ for all $\mathbf{u}, \mathbf{v}, \mathbf{w} \in V$
(iv) $\langle \alpha \mathbf{u}, \mathbf{v} \rangle = \alpha \langle \mathbf{u}, \mathbf{v} \rangle$ for all $\alpha \in F$ and $\mathbf{u}, \mathbf{v} \in V$
(v) $\langle \mathbf{u}, \mathbf{v} \rangle = \overline{\langle \mathbf{v}, \mathbf{u} \rangle}$ for all $\mathbf{u}, \mathbf{v} \in V$

Example 5.9. Some examples of inner products are as follows:

(i) We have the dot product on $\mathbb{R}^n$. For any $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, we have

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^{\mathrm{T}} \mathbf{y} = \sum_{i=1}^{n} x_i y_i.$$

(ii) We have the dot product on $\mathbb{C}^n$. For $\mathbf{w}, \mathbf{z} \in \mathbb{C}^n$, we have

$$\langle \mathbf{w}, \mathbf{z} \rangle = \mathbf{z}^{*} \mathbf{w} = \sum_{i=1}^{n} w_i \overline{z_i}.$$

Here, $\mathbf{z}^*$ denotes the complex conjugate of $\mathbf{z}$.

(iii) We can also have an inner product on $\mathcal{M}_{m \times n}\, \mathbb{R})$. For any matrices $\mathbf{A}, \mathbf{B} \in \mathcal{M}_{m \times n}(\mathbb{R})$, define

$$\langle \mathbf{A}, \mathbf{B} \rangle = \operatorname{tr}\left(\mathbf{A}^{\mathrm{T}} \mathbf{B}\right).$$

We can also have inner products on random variables and functions. For example,

$$\langle X, Y \rangle = \mathbb{E}(XY) \quad \text{where } \mathbb{E}(X) \text{ denotes the expectation of } X.$$

Also,

$$\langle f, g \rangle = \int f(x)\, g(x)\ dx.$$

Recall that a basis for a finite-dimensional vector space V is a set of vectors in V that spans V and is linearly independent. If $\{\mathbf{v}_1, \ldots, \mathbf{v}_n\}$ is a basis for V, then every $\mathbf{v} \in V$ has a unique representation

$$\mathbf{v} = \alpha_1 \mathbf{v}_1 + \ldots + \alpha_n \mathbf{v}_n.$$

A set of vectors $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ is said to be orthonormal if $\langle \mathbf{e}_i, \mathbf{e}_j \rangle = 1$ for $i = j$ and 0 if $i \neq j$. If $\{\mathbf{e}_1, \ldots, \mathbf{e}_n\}$ is an orthonormal basis for V, then every $\mathbf{v} \in V$ has a unique representation

$$\mathbf{v} = \langle \mathbf{v}, \mathbf{e}_1 \rangle \mathbf{e}_1 + \ldots + \langle \mathbf{v}, \mathbf{e}_n \rangle \mathbf{e}_n.$$

Note that representing vectors in terms of an orthonormal basis is more computationally efficient than representing them in terms of non-orthonormal bases.

We then extend the idea of inner products and orthogonality to functions.

Definition 5.3 (inner product for continuous functions)**.** For continuous functions f and g on $[a, b]$, define the inner product

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

One can also integrate with respect to a certain weight function $w(x) > 0$, thus yielding the inner product formula

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

Note that we will not consider integrating over $\mathbb{R}$ because for most polynomials, this is either infinity or undefined.

Using the inner product defined in Definition 5.3, we can apply the Gram-Schmidt process to polynomials. Say we consider the monomial basis $\{1, x, x^2, \ldots, x^n\}$ and the inner product

$$\langle p, q \rangle = \int_{-1}^1 p(x)\, q(x)\ dx.$$

One can apply the Gram-Schmidt process to obtain a list of orthogonal polynomials $q_0, \ldots, q_n$ as follows:

$$q_0(x) = \sqrt{\frac{1}{2}} \quad q_1(x) = \sqrt{\frac{3}{2}}x \quad q_2(x) = \sqrt{\frac{5}{8}}\left(3x^2 - 1\right) \quad q_3(x) = \sqrt{\frac{7}{8}}\left(5x^3 - 3x\right)$$

We then *orthonormalise* the polynomials. Instead of requiring $\|q_i\| = 1$ in the usual sense for vectors in $\mathbb{R}^n$, it is typical to require $q_i(1) = 1$. As such,

$$q_0(x) = 1 \quad q_1(x) = x \quad q_2(x) = \frac{1}{2}\left(3x^2 - 1\right) \quad q_3(x) = \frac{1}{2}\left(5x^3 - 3x\right)$$

In general, one can prove that

$$q_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}\left(x^2 - 1\right)^n \tag{5.6}$$

In fact, the polynomials $q_n(x)$ are known as the Legendre polynomials, and the equation in (5.6) is known as Rodrigues' formula.

Proposition 5.1. In general, the Legendre polynomials satisfy the recurrence relation

$$nq_n(x) = (2n-1)\,xq_{n-1}(x) - (n-1)\,q_{n-2}(x).$$

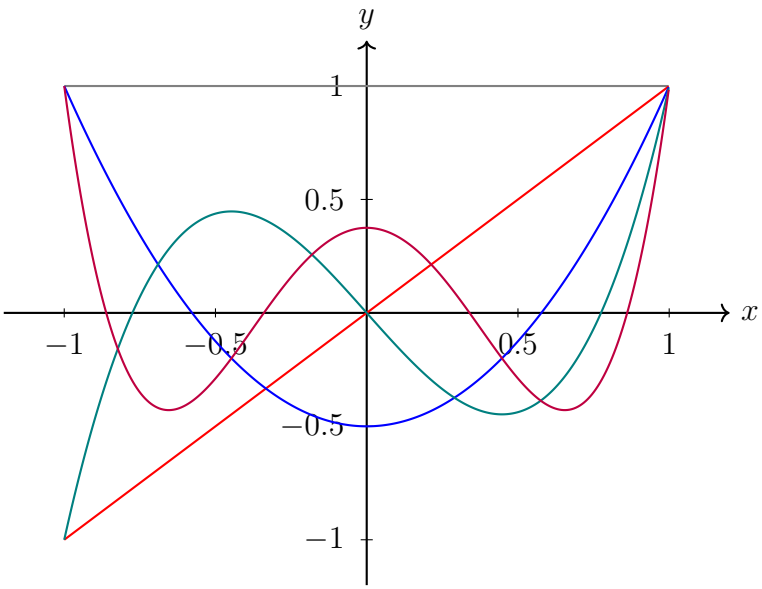

Figure 2: Graphs of $P_0(x)$, $P_1(x), P_2(x), P_3(x), P_4(x)$ on $[-1, 1]$

The Chebyshev polynomials of the first kind, defined by the inner product

$$\langle p, q\rangle = \int_{-1}^{1} \frac{p(x)\,q(x)}{\sqrt{1-x^2}}\,dx,$$

are also interesting. Consider the orthogonal polynomials produced by the Gram-Schmidt process equipped with the mentioned inner product, thus producing the polynomials $T_0, \ldots, T_n$, where

$$T_0(x) = 1 \quad T_1(x) = x \quad T_2(x) = 2x^2 - 1 \quad T_3(x) = 4x^3 - 3x.$$

The Chebyshev polynomials of the second kind can also be defined by

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

Proposition 5.2. The Chebyshev polynomials of the first kind satisfy the recurrence relation

$$T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x).$$

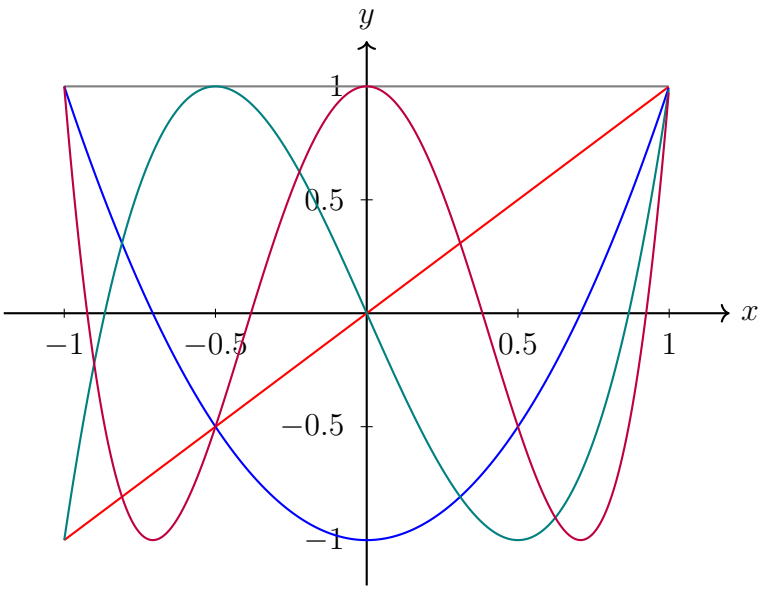

Figure 3: Graphs of $T_0(x), T_1(x), T_2(x), T_3(x), T_4(x)$

In Numerical Analysis, the most commonly encountered orthogonal polynomials are the Legendre polynomials, Chebyshev polynomials, Laguerre polynomials, and the Hermite polynomials. All orthogonal polynomials satisfy a three-term recurrence relation. Orthogonal polynomials have advantages for least squares polynomial fitting, and they are applied in numerous branches of Mathematics such as Differential Equations, Probability Theory, Physics, etc.

Chapter 6

Numerical Integration and Differentiation

6.1 Newton-Cotes Quadrature Rules

Now, our discussion shifts to the evaluation of the integral

$$\int_a^b f(x)\ dx. \tag{6.1}$$

Integrals of the form in (6.1) can be evaluated using series expansion. However, the series is not always available. Even worse, sometimes the function $f(x)$ is provided as a *black box*, meaning to say that we can access its values but not have its analytical expression. In this case, a commonly-used technique is to find several function values $f(x)$ and then approximate $f(x)$ by interpolation. Afterwards, we integrate the interpolating function and use the result as the numerical approximation of the integral.

One should recall the two fundamental theorems of Calculus, which we shall not state. Many problems in Applied Mathematics involve computing definite integrals, and here we present a few that are particularly relevant to Data Science. In Probability Theory, if X and Y are continuous random variables and X has density f, we can compute the probability that X lies in some interval using

$$P(a \leq X \leq b) = \int_a^b f(x)\ dx.$$

We can also compute the expected value of X using

$$\mathbb{E}(X) = \int_{\mathbb{R}} xf(x)\ dx.$$

If X and y have joint density function g, then the marginal distributions are computed using

$$g_X(x) = \int_{\mathbb{R}} g(x,y)\ dy \quad \text{and} \quad g_Y(y) = \int_{\mathbb{R}} g(x,y)\ dx.$$

For distributions whose densities are *not very nice*, or even unknown, these integrals must be computed numerically rather than analytically. For example, the Laplace transform

$$\mathcal{L}(f)(t) = \int_0^\infty e^{-xt} f(x)\ dx$$

has applications in designing circuits and signal processing. Furthermore, in Probability Theory, the moment generating function of a random variable is the Laplace transform of the probability density function.

The Fourier transform

$$\mathcal{F}(f)(t) = \int_{-\infty}^\infty e^{-2\pi itx} f(x)\ dx$$

has applications in differential equations, signal processing and Quantum Mechanics. Moreover and again in Probability Theory, the characteristic function of a random variable is the Fourier transform of the probability density function[†].

Certain important functions do not have closed form representations in terms of other elementary functions, and are instead defined by integrals. For example, we have the gamma function

$$\Gamma(t) = \int_0^\infty e^{-x} x^{t-1}\ dx,$$

the beta function

$$B(a,b) = \int_0^1 x^{a-1}(1-x)^{b-1}\ dx,$$

and the error function

$$\Phi(t) = \frac{2}{\sqrt{\pi}} \int_0^t e^{-x^2}\ dx.$$

These have applications in Probability Theory too.

Recall from MA2002 Calculus that one standard way of defining the definite integral is using Riemann sums. On the interval $[a,b]$, we define $h = \frac{b-a}{n}$ and set $x_k = a + hk$ for $k = 0, 1, \ldots, n$. We then define the left and right Riemann sums to be

$$L_n = \sum_{k=0}^{n-1} hf(x_k) \quad \text{and} R_n = \sum_{k=1}^{n} hf(x_k) \text{ respectively.}$$

[†]In fact, this is the typical way of proving the central limit theorem

If

$$\lim_{n\to\infty} R_n = \lim_{n\to\infty} L_n = I, \tag{6.2}$$

we say that f is Riemann integrable on $[a,b]$ and we write

$$\int_a^b f(x)\ dx = I.$$

(6.2) is *informally* known as the Riemann integrability criterion. There is a more mathematically rigorous way to state it, but it is taught in MA3210 Mathematical Analysis II. Anyway, this suggests one way of approximating the integral: take relatively large n and compute the Riemann sums.

Example 6.1. For example, we consider the function $f(x) = \sqrt{1+x^2}$ on $[-1,1]$. One can use a Pythagorean identity to evaluate

$$\int_{-1}^1 f(x)\ dx \approx 2.2956$$

and in fact obtain the exact value. As mentioned, we can obtain an approximation. We leave it to the reader to compute L_n and R_n for $n = 1,2,3$ but it is apparent that for larger n, L_n and R_n will converge to the true value. However, it will converge relatively slowly in practice, so we can attempt to use other methods.

We shall introduce other methods of numerical integration but before that, we discuss the existence, uniqueness, and conditioning of the problem. First, if f is bounded on $[a,b]$ and continuous at all but countably many points on $[a,b]$, then the Riemann integral exists[†]. Since the integral is defined using a limit and limits are unique when they exist, uniqueness is built into the definition. For condition, we need a way to measure the *size* of a function. Define

$$\|f\|_\infty = \max_{x\in[a,b]} |f(x)| \quad \text{and} \quad \int_a^b f(x)\ dx = I(f).$$

If $\widetilde{f}$ is a perturbation of f, then we have

$$\left|I(f) - I\left(\widetilde{f}\right)\right| \le (b-a)\left\|f - \widehat{f}\right\|_\infty.$$

That is, the error is proportional to the size of the interval. Many numerical integration methods are referred to as quadrature. This means that we use quadrilaterals to approximate integrals.

[†]This is the Lebesgue-Vitali theorem.

Definition 6.1 (quadrature rule). An n point quadrature rule is of the form

$$Q_n(f) = \sum_{i=1}^{n} w_i f(x_i)$$

with coefficients or weights w_i, and nodes or abscissas x_i.

As mentioned before, there is a connection between numerical integration and interpolation. Given points $(x_i, f(x_i))$ where $i = 0, 1, \ldots, n$, we can fit a polynomial to these data, and definite integrals of polynomials can be evaluated easily. Let $\ell_1, \ldots, \ell_n$ be the Lagrange basis functions and interpolate

$$p(x) = \sum_{i=1}^{n} f(x_i)\, \ell_i(x).$$

Then, one can easily show that

$$\int_a^b p(x)\ dx = \sum_{i=0}^{n} w_i f(x_i). \tag{6.3}$$

We have thus obtained a quadrature rule. Alternatively, we can develop another quadrature rule as follows. Suppose we seek a quadrature rule that integrates polynomials of degree n and below exactly. That is, we want the equation (6.3) to be true for polynomials of degree n and below. In particular, if we apply this to the monomial basis $\{1, x, x^2, \ldots, x^n\}$, we have

$$\begin{aligned}
\int_a^b dx &= b - a = w_0 \cdot 1 + \ldots + w_n \cdot 1 \\
\int_a^b x\ dx &= \frac{b^2 - a^2}{2} = w_0 \cdot x_0 + \ldots + w_n \cdot x_n \\
&\vdots = \vdots \\
\int_a^b x^n\ dx &= \frac{b^{n+1} - a^{n+1}}{n+1} = w_0 \cdot x_0^n + \ldots + w_n \cdot x_n^n
\end{aligned}$$

This leads to the system of equations

$$\begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_0 & x_1 & \ldots & x_n \\ \vdots & \vdots & \ddots & \vdots \\ x_0^n & x_1^n & \ldots & x_n^n \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} b - a \\ \frac{b^2 - a^2}{2} \\ \vdots \\ \frac{b^{n+1} - a^{n+1}}{n+1} \end{pmatrix}$$

The transpose of the coefficient matrix is a Vandermonde matrix. When we choose the nodes $x_0, \ldots, x_n$ to be equally spaced, we call the resulting quadrature a Newton-Cotes rule. We will investigate quadrature rules with $n = 0, 1, 2, 3$. Note that n is the degree of the rule, not the number of nodes. Here, the degree refers to the degree of the underlying interpolating polynomial. Also, closed Newton-Coates rules include the endpoints a and b, whereas open rules do not.

Proposition 6.1 (midpoint rule)**.** When we sample at only one point $x_0 = \frac{b-a}{2}$, this corresponds to interpolation by a constant polynomial. Our example of the integral is given by

$$M(f) = (b-a) f\left(\frac{a+b}{2}\right).$$

This is known as the midpoint rule.

We then introduce the trapezium rule (Proposition 6.2), which involves sampling at $x_0 = a$ and $x_1 = b$. This corresponds to interpolation by a linear polynomial. Note that the line through the points $(a, f(a))$ and $(b, f(b))$ is given by

$$p(x) = f(a)\frac{x-b}{a-b} + f(b)\frac{x-a}{b-a}.$$

Integrating and simplifying yields Proposition 6.2.

Proposition 6.2 (trapezium rule)**.** We have

$$T(f) = \frac{b-a}{2}(f(a) + f(b)).$$

When we sample at $x_0 = a$, $x_1 = \frac{a+b}{2}$ and $x_2 = b$, this corresponds to interpolation by a quadratic polynomial. The formula is given by

$$p(x) = f(a)\frac{(x-m)(x-b)}{(a-m)(a-b)} + f(m)\frac{(x-a)(x-b)}{(m-a)(m-b)} + f(b)\frac{(x-a)(x-m)}{(b-a)(b-m)}.$$

Integrating and simplifying yields Proposition 6.3.

Proposition 6.3 (Simpson's rule)**.** We have

$$S(f) = \frac{b-a}{6}(f(a) + 4f(m) + f(b))$$

where $m = \frac{a+b}{2}$.

Recall the identity

$$S(f) = \frac{2}{3}M(f) + \frac{1}{3}T(f).$$

Lastly, when we sample at $x_0 = a$, $x_1 = \frac{2a+b}{3}$, $x_2 = \frac{a+2b}{3}$, and $x_3 = b$, this corresponds to interpolation by a cubic polynomial. This yields Simpson's 3/8 rule (Proposition 6.4).

Proposition 6.4 (Simpson's 3/8 rule)**.** We have

$$\Theta(f) = \frac{b-a}{8}\left(f(a) + 3f\left(\frac{2a+b}{3}\right) + 3f\left(\frac{a+2b}{3}\right) + f(b)\right).$$

Example 6.2. On $[0,1]$, $\int_0^1 e^x dx = e - 1 \approx 1.71828$.

$$M(e^x) = e^{1/2} \approx 1.64872, \quad T(e^x) = \tfrac{1}{2}(1+e) \approx 1.85914,$$

$$S(e^x) = \tfrac{1}{6}\left(1 + 4e^{1/2} + e\right) \approx 1.71886, \quad \Theta(e^x) = \tfrac{1}{8}\left(1 + 3e^{1/3} + 3e^{2/3} + e\right) \approx 1.71854.$$

Absolute errors are about 0.0696, 0.1409, 0.00058, and 0.00026, respectively.

We now discuss error expansions via a midpoint Taylor series. We expand about $m = \frac{a+b}{2}$ to obtain

$$f(x) = f(m) + f'(m)(x-m) + \frac{f''(m)}{2}(x-m)^2 + \frac{f^{(3)}(m)}{6}(x-m)^3 + \frac{f^{(4)}(m)}{24}(x-m)^4 + \dots.$$

By symmetry, the odd powers integrate to 0. Writing $h = \frac{b-a}{2}$, one obtains

$$\int_a^b f(x)\,dx = (b-a)f(m) + \frac{f''(m)}{24}(b-a)^3 + \frac{f^{(4)}(m)}{1920}(b-a)^5 + \dots$$

Hence the *single-panel* error formulas are

$$\int_a^b f - M(f) = \frac{f''(m)}{24}(b-a)^3 + \frac{f^{(4)}(m)}{1920}(b-a)^5 + \dots T(f) - \int_a^b f \quad = \frac{f''(m)}{12}(b-a)^3 + \frac{f^{(4)}(m}{480}$$

Consequently, if $|f''(x)| \le c$ on $[a,b]$, then

$$\left|\int_a^b f - M(f)\right| \le \frac{c}{24}(b-a)^3 \quad \text{and} \quad \left|\int_a^b f - T(f)\right| \le \frac{c}{12}(b-a)^3.$$

If $\left|f^{(4)}(x)\right| \leq c$ on $[a,b]$, then

$$\left|\int_a^b f - S(f)\right| \leq \frac{c}{2880}(b-a)^5 \quad \text{and} \quad \left|\int_a^b f - \Theta(f)\right| \leq \frac{c}{6480}(b-a)^5.$$

We can generalise Propositions 6.1, 6.2, 6.3 using composite Newton-Cotes rules. Partition $[a,b]$ into k panels of length $h = \frac{b-a}{k}$, with nodes $x_j = a + jh$. The composite midpoint rule states that

$$M_k(f) = h\sum_{j=1}^{k} f\left(\frac{x_{j-1}+x_j}{2}\right).$$

The composite trapezium rule states that

$$T_k(f) = \frac{h}{2}\left(f(a) + f(b) + 2\sum_{j=1}^{k-1} f(x_j)\right).$$

Lastly, the composite Simpson's rule with $2k$ panels states the following, where we let $h = \frac{b-a}{2k}$:

$$S_{2k}(f) = \frac{h}{3}\left(f(a) + f(b) + 4\sum_{j=1}^{k} f(x_{2j-1}) + 2\sum_{j=1}^{k-1} f(x_{2j})\right)$$

Example 6.3. On $[0,\pi]$ with two panels for midpoint and trapezium:

$$M_2(\sin x) = \frac{\pi}{2}\left(\tfrac{\sqrt{2}}{2} + \tfrac{\sqrt{2}}{2}\right) = \frac{\sqrt{2}\pi}{2} \approx 2.22, \quad T_2(\sin x) = \frac{\pi}{4}(0+0+2) = \frac{\pi}{2} \approx 1.57,$$

while $\int_0^\pi \sin x dx = 2$.

Let

$$Q_n(f) = \sum_{i=0}^{n} w_i f(x_i)$$

with equally spaced x_i. If all $w_i > 0$, then

$$\sum_{i=0}^{n} |w_i| = \sum_{i=0}^{n} w_i = b - a.$$

For sufficiently large n, at least one weight is negative and

$$\lim_{n\to\infty} \sum_{i=0}^{n} |w_i|,$$

reflecting instability on equally spaced nodes. A practical alternative is *Clenshaw–Curtis* quadrature, which samples at Chebyshev extrema and is equivalent to

$$\int_{-1}^{1} f(x)\,dx = \int_0^\pi f(\cos\theta)\sin\theta\,d\theta.$$

6.2 Numerical Differentiation

Numerical differentiation is often simpler than numerical integration, but it is typically *ill-conditioned*: small perturbations in data can be amplified, and catastrophic cancellation is hard to avoid when subtracting nearby values.

Recall that the derivative at x is

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

We use numerical differentiation when we only have discrete samples, when there is no closed form for f, or when the exact formula is more costly than an approximation. From the definition, we have

$$\begin{aligned} \text{forward difference } f'(x) &\approx \frac{f(x+h) - f(x)}{h} \\ \text{backward difference } f'(x) &\approx \frac{f(x) - f(x-h)}{h} \\ \text{central difference } f'(x) &\approx \frac{f(x+h) - f(x-h)}{2h} \end{aligned}$$

Many handheld calculators use the central formula with $h = 0.001$. Applying backward to forward difference yields the three-point second derivative:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

Taylor's theorem on $[x, x+h]$ gives

$$f(x+h) = f(x) + f'(x)\,h + \frac{f''(a)}{2}h^2 \quad \text{for some } a \in (x, x+h).$$

Solving for $f'(x)$ yields the forward difference with an explicit remainder

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{f''(a)}{2}h.$$

More generally, with higher smoothness,

$$\begin{aligned} \text{Forward:} \quad f'(x) &= \frac{f(x+h) - f(x)}{h} - \frac{f''(\xi_+)}{2}h, \\ \text{Backward:} \quad f'(x) &= \frac{f(x) - f(x-h)}{h} + \frac{f''(\xi_-)}{2}h, \\ \text{Central:} \quad f'(x) &= \frac{f(x+h) - f(x-h)}{2h} - \frac{f^{(3)}(\eta)}{6}h^2, \end{aligned}$$

for some ξ_+, ξ_-, η between $x-h$ and $x+h$. Thus forward and backward are first order in h, while central is second order.

Proposition 6.5 (balancing truncation and rounding)**.** If $|f''(t)| \leq M$ near x, the forward truncation error is at most $\frac{1}{2}Mh$. If each function value has absolute error at most ε, the subtraction in $(f(x+h) - f(x))$ contributes rounding error $\leq 2\varepsilon/h$. The total bound

$$E(h) \leq \frac{1}{2}Mh + \frac{2\varepsilon}{h}$$

is minimised at $h_* = 2\sqrt{\varepsilon/M}$.