# INTRODUCTION TO COMPUTATIONAL MATHEMATICS

## Course Notes for CM 271 / AMATH 341 / CS 371

Fall 2008

Instructor: Prof. Keith Geddes
School of Computer Science
University of Waterloo

Course notes by Prof. Hans De Sterck
Design and typesetting by
Paul Ullrich

July 3, 2007

**These notes have been funded by**

MEF Mathematics Endowment Fund

# Contents

# Preface

The goal of computational mathematics, put simply, is to find or develop algorithms that solve mathematical problems computationally (ie. using computers). In particular, we desire that any algorithm we develop fulfills three primary properties:

- **Accuracy.** An accurate algorithm is able to return a result that is numerically very close to the correct, or analytical, result.
- **Efficiency.** An efficient algorithm is able to quickly solve the mathematical problem with reasonable computational resources.
- **Robustness.** A robust algorithm works for a wide variety of inputs $\mathbf{x}$.

**These notes have been funded by...**

# Chapter 1

# Errors and Error Propagation

In examining computational mathematics problems, we shall generally consider a problem of the following form:

**Problem** Consider an arbitrary problem **P** with input **x**. We must compute the desired output $\mathbf{z} = f_P(\mathbf{x})$.

In general our only tools for solving such problems are primitive mathematical operations (for example, addition, subtraction, multiplication and division) combined with flow constructs (if statements and loops). As such, even simple problems such as evaluating the exponential function may be difficult computationally.

**Example 1.1** *Consider the problem* **P** *defined by the evaluation of the exponential function* $z = \exp(x)$*. We wish to find the approximation* $\hat{z}$ *for* $z = \exp(x)$ *computationally.*

**Algorithm A.** Recall from calculus that the Taylor series expansion of the exponential function is given by

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... = \sum_{i=0}^{\infty} \frac{x^i}{i!}. \tag{1.1}$$

Since we obviously cannot compute an infinite sum computationally without also using infinite resources, consider the truncated series constructed by only summing the first $n$ terms. This yields the expansion

$$\hat{z} = \sum_{i=0}^{n} \frac{x^i}{i!} \tag{1.2}$$

As we will later see, this is actually a poor algorithm for computing the exponential - although the reason for this may not be immediately obvious.

## 1.1 Sources of Error

Computational error can originate from several sources, although the most prominent forms of error (and the ones that we will worry about in this text) come from two main sources:

1. Errors introduced in the input $\mathbf{x}$. In particular, this form of error can be generally classified to be from one of two sources:

   a) Measurement error, caused by a difference between the "exact" value $\mathbf{x}$ and the "measured" value $\tilde{\mathbf{x}}$. The measurement error is computed as $\Delta\mathbf{x} = |\mathbf{x} - \tilde{\mathbf{x}}|$.

   b) Rounding error, caused by a difference between the "exact" value $\mathbf{x}$ and the computational or floating-point representation $\hat{\mathbf{x}} = fl(\mathbf{x})$. Since infinite precision cannot be achieved with finite resources, the computational representation is a finite precision approximation of the exact value.

   Consider, for example, the decimal number $x = 0.00012345876543$. In order to standardize the representation of these numbers we perform normalization (such that the number to the left of the decimal point is 0 and the first digit right of the decimal point is nonzero). The number $\hat{x}$ is thus normalized as follows:

   $$x = 0.\underbrace{12345876543}_{\text{mantissa}} \times 10^{-3}. \tag{1.3}$$

   However, a computer can only represent finite precision, so we are not guaranteed to retain all digits from the initial number. Let's consider a hypothetical "decimal" computer with at most 5 digits in the mantissa. The floating-point representation of $x$ obtained on this computer, after rounding, is given by

   $$\hat{x} = \texttt{fl}(x) = 0.12346 \times 10^{-3}. \tag{1.4}$$

   The process of conversion to a floating point number gives us a rounding error equal to $\Delta x = x - \hat{x} = -0.00000123457$.

2. Errors as a result of the calculation, approximation or algorithm. This form of error can again be generally classified to be from one of two sources:

   a) Truncation error. When truncating an infinite series to provide a finite approximation, the method inherently introduces an error. In example 1.1 we first considered truncating the infinite expansion as follows:

   $$z = \exp(x) = \sum_{i=0}^{\infty} \frac{x^i}{i!} = \sum_{i=0}^{n} \frac{x^i}{i!} + \sum_{i=n+1}^{\infty} \frac{x^i}{i!} = \hat{z} + \sum_{i=n+1}^{\infty} \frac{x^i}{i!} \tag{1.5}$$

   In this case, the truncation error is given by

   $$T = z - \hat{z} = \sum_{i=n+1}^{\infty} \frac{x^i}{i!} \tag{1.6}$$

b) Rounding errors in elementary steps of the algorithm. For example, consider the addition of $1.234 \times 10^7$ and $5.678 \times 10^3$ in a floating point number system where we only maintain 4 digit accuracy (in base 10). The exact result should be $1.2345678 \times 10^7$ but due to rounding we instead calculate the result to be $1.235 \times 10^7$.

In analyzing sources of error, it is often useful to provide a mathematical basis for the error analysis. In particular, there are two principal ways for providing a measure of the generated error.

**Definition 1.1** *Consider an exact result $z$ and an approximate result $\hat{z}$ generated in a specified floating point number system. Then the* ***absolute error*** *is given by*

$$\Delta z = z - \hat{z}, \tag{1.7}$$

*and the* ***relative error*** *(assuming $z \neq 0$) is given by*

$$\delta z = \frac{z - \hat{z}}{z}. \tag{1.8}$$

Of course, considering the mathematical hardships often involved in analyzing our algorithms for error, there must be some justification in bothering with error analysis. Although the errors may be first perceived as being negligible, some numerical algorithms are "numerically unstable" in the way they propagate errors. In doing error analysis, we will often run into one of the following situations in computational mathematics:

1. The algorithm may contain one or more "avoidable" steps that each greatly amplify errors.
2. The algorithm may propagate initially small errors such that the error is amplified without bound in many small steps.

Consider the following example of the first kind of generated error:

**Example 1.2** *Consider the problem* **P** *with input $x$ defined by the evaluation of the exponential function $z = \exp(x)$ (as considered in (1.1)). However, in performing the calculation, assume the following conditions:*

- *Assume "decimal" computer with 5 digits in the mantissa.*
- *Assume no measurement error or initial rounding error in $x$.*

Consider solving this problem with input $x = -5.5$; we find a numerical approximation to the exact value $z = \exp(-5.5) \approx 0.0040868$.

In solving this problem, we first apply Algorithm A, truncating the series after the first 25 values. This yields the formula $\hat{z} = \sum_{i=0}^{24} \frac{x^i}{i!}$. Performing this calculation with our floating point system yields the approximation $\hat{z}_A = 0.0057563$ which an observant reader will notice has no significant digits in common with the exact solution. We conclude that precision was lost in this calculation, and in fact notice that the relative error is $\delta z_A = \frac{1}{z}(z - \hat{z}_A) = -0.41 = -41\%$. This is a substantial error!

**Table I: Algorithm A applied to $x = -5.5$**

| $i$ | $i^{th}$ term in series | $i^{th}$ truncated sum |
|---|---|---|
| 0 | 1.0000000000000 | 1.0000000000000 |
| 1 | -5.5000000000000 | -4.5000000000000 |
| 2 | 15.1250000000000 | 10.6250000000000 |
| 3 | -27.7300000000000 | -17.1050000000000 |
| 4 | 38.1280000000000 | 21.0230000000000 |
| 5 | -41.9400000000000 | -20.9170000000000 |
| 6 | 38.4460000000000 | 17.5290000000000 |
| 7 | -30.2060000000000 | -12.6770000000000 |
| 8 | 20.7670000000000 | 8.0900000000000 |
| 9 | -12.6910000000000 | -4.6010000000000 |
| 10 | 6.9803000000000 | 2.3793000000000 |
| 11 | -3.4900000000000 | -1.1107000000000 |
| 12 | 1.5996000000000 | 0.4889000000000 |
| 13 | -0.6767600000000 | -0.1878600000000 |
| 14 | 0.2658700000000 | 0.0780100000000 |
| 15 | -0.0974840000000 | -0.0194740000000 |
| 16 | 0.0335100000000 | 0.0140360000000 |
| 17 | -0.0108420000000 | 0.0031940000000 |
| 18 | 0.0033127000000 | 0.0065067000000 |
| 19 | -0.0009589000000 | 0.0055478000000 |
| 20 | 0.0002637100000 | 0.0058115000000 |
| 21 | -0.0000690670000 | 0.0057424000000 |
| 22 | 0.0000172670000 | 0.0057597000000 |
| 23 | -0.0000041289000 | 0.0057556000000 |
| 24 | 0.0000009462300 | 0.0057565000000 |
| 25 | -0.0000002081700 | 0.0057563000000 |
| 26 | 0.0000000440350 | 0.0057563000000 |

So why does this instability occur?

Our answer to this question can be found in a related problem: Consider the subtraction of two numbers that are almost equal to one another.

Let $x_1 = 0.\underline{100134}826$ with floating-point representation $\texttt{fl}(x_1) = 0.10013$. The relative error in performing this approximation is $\delta x_1 = \frac{1}{x_1}(x_1 - \texttt{fl}(x_1)) = 4.8 \times 10^{-5} = 0.0048\%$.

Let $x_2 = 0.\underline{10012}1111$ with floating-point representation $\texttt{fl}(x_1) = 0.10012$. The relative error in performing this approximation is $\delta x_2 = \frac{1}{x_2}(x_2 - \texttt{fl}(x_2)) = 1.1 \times 10^{-5} = 0.0011\%$.

So, in general, the approximation of these two numbers to their floating-point equivalents produce relatively small errors. Now consider the subtraction $z = x_1 - x_2$. The exact solution to this is $z = 0.\underline{00001}3715$ and the computed solution using the floating-point representations is $\hat{z} = \texttt{fl}(x_1) - \texttt{fl}(x_2) = 0.00001$. The relative error in this case is $\delta\hat{z} = \frac{1}{z}(z - \hat{z}) \approx 27\%$. So what was an almost negligible error when performing rounding becomes a substantial error after we have completed the subtraction.

Thus, if possible, we will need to avoid these kind of subtractions when we are developing our algorithms. Looking back at the additions we performed in calculating $\exp(-5.5)$ (see Table I), we see that there were several subtractions performed of numbers of similar magnitude. Fortunately, we have an easy way around this if we simply take advantage of a property of the exponential, namely $\exp(-x) = (\exp(x))^{-1}$. This provides us with Algorithm B:

**Algorithm B.** Applying the truncated Taylor expansion for $\exp(5.5)$, we get the following formula for $\exp(-5.5)$:

$$\exp(x) = (\sum_{i=0}^{24} \frac{(-x)^i}{i!})^{-1} \tag{1.9}$$

This yields $\hat{z}_B = 0.00\underline{40865}$, which matches 4 out of 5 digits of the exact value.

We conclude that Algorithm B is numerically stable (largely since it avoids cancellation). Similarly, Algorithm A is unstable with respect to relative error for the input $x = -5.5$. However, it is also worth noting that for any positive value of the input we are better off using Algorithm A - Algorithm B would end up with the same cancellation issues we had originally with Algorithm A.

## 1.2 Floating Point Numbers and Operations

Floating point numbers are the standard tool for approximating real numbers on a computer. Unlike real numbers, floating point numbers only provide finite precision - effectively approximating real numbers while still attempting to provide full functionality. Consider the following definition of a floating point number system:

**Definition 1.2** *A **floating point number system** is defined by three components:*

- *The **base**, which defines the base of the number system being used in the representation. This is specified as a positive integer $b_f$.*
- *The **mantissa**, which contains the normalized value of the number being represented. Its maximal size is specified as a positive integer $m_f$, which represents the number of digits allowed in the mantissa.*
- *The **exponent**, which effectively defines the offset from normalization. Its maximal size is specified by a positive integer $e_f$, which represents the number of digits allowed in the exponent.*

*In shorthand, we write $F[b = b_f, m = m_f, e = e_f]$.*

Combined, the three components allow us to approximate any real number in the following form:

$$0.\underbrace{x_1x_2\cdots x_m}_{\text{mantissa}} \times \underbrace{b}_{\text{base}}^{\underbrace{y_1y_2\cdots y_e}_{\text{exponent}}} \tag{1.10}$$

**Example 1.3** *Consider a "decimal" computer with a floating point number system defined by base 10, 5 digits in the mantissa and 3 digits in the exponent. In shorthand, we write the system as $F[b = 10, m = 5, e = 3]$.*

*Consider the representation of the number $x = 0.000123458765$ under this system. To find the representation, we first normalize the value and then perform rounding so both the mantissa and exponent have the correct number of digits:*

| | |
|---|---|
| *input* | $x = 0.000123458765$ |
| *normalize* | $x = 0.123458765 \times 10^{-3}$ |
| *round* | $\hat{x} = \texttt{fl}(x) = 0.12346 \times 10^{-003}$ |

*Under this system, our mantissa is bounded by 99999 and the exponent is bounded by 999 (each representing the largest numbers we can display under this base in 5 and 3 digits, respectively). The largest number we can represent under this system is $0.99999 \times 10^{999}$ and the smallest positive number we can represent is $0.00001 \times 10^{-999}$.*

**Note:** Instead of rounding in the final step, we can also consider "chopping". With chopping, we simply drop all digits that cannot be represented under our system. Thus, in our example we would get $\hat{x} = \texttt{fl}(x) = 0.12345 \times 10^{-003}$ since all other digits simply would be dropped.

### 1.2.1 A Binary Computer

Instead of working in decimal (base 10), almost all computers work in binary (base 2). We normally write binary numbers as $(x)_b$ to indicate that they are represented in binary.

**Example 1.4** *Consider the binary number* $x = (1101.011)_b$. *Similar to decimal, this notation is equivalent to writing*

$$\begin{aligned} x &= 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 1 \cdot 2^{-3} \\ &= 8 + 4 + 1 + 0.25 + 0.125 \\ &= 13.375 \end{aligned}$$

Floating-point number conversions from binary work exactly the same way as decimal conversions, except for the new base:

**Example 1.5** *Consider the binary number* $x = (1101.011)_b$ *under the floating-point system* $F[b = 2, m = 4, e = 3]$.

$$\begin{aligned} &\textit{input} && x = (1101.011)_b \\ &\textit{normalize} && x = (0.1101011)_b \times 2^4 \\ &&& \quad = (0.1101011)_b \times 2^{(100)_b} \\ &\textit{round} && \hat{x} = \texttt{fl}(x) = (0.1101)_b \times 2^{(100)_b} \\ &&& \quad = (0.1101)_b \times (10)_b^{(100)_b} \end{aligned}$$

### 1.2.2 Standard floating point systems

**Single Precision Numbers.** Single precision is one of the standard floating point number systems, where numbers are represented on a computer in a 32-bit chunk of memory (4 bytes). Each number in this system is divided as follows:

| | | | |
|---|---|---|---|
| $s_m$ | m = 23 bits | $s_e$ | e = 7 bits |

Here $s_m$ is the sign bit of the mantissa and $s_e$ is the sign bit for the exponent.

Recall that we normally write floating point numbers in the form given by (1.10). The typical convention for sign bits is to use 0 to represent positive numbers and 1 to represent negative numbers.

**Example 1.6** *Consider the decimal number* $x = -0.3125$. *We note that we may write this in binary as*

$$\begin{aligned} x &= -0.3125 \\ &= -(0 \cdot 2^{-1} + 1 \cdot 2^{-2} + 0 \cdot 2^{-3} + 1 \cdot 2^{-4}) \\ &= -(0.0101)_b \\ &= -(0.101)_b \times 2^{-1} \\ &= -(0.101)_b \times 2^{-(1)_b} \end{aligned}$$

*In the single precision floating point system* $F[b = 2, m = 23, e = 7]$ *we may write* $\hat{x} = \texttt{fl}(x) = |\texttt{1}|\texttt{10100000000000000000000}|\texttt{1}|\texttt{0000001}|$.

Under the single-precision floating point number system, the largest and smallest numbers that can be represented are given as follows (without consideration for normalization, in the case of the smallest number).

$$\begin{aligned} \hat{x}_{max} &= |\texttt{0}|\texttt{11111111111111111111111}|\texttt{0}|\texttt{1111111}| \\ &= (1 - 2^{-23}) \cdot 2^{127} \\ &\approx 2^{127} \\ &\approx 1.7 \times 10^{38} \end{aligned}$$

$$\begin{aligned} \hat{x}_{min} &= |\texttt{0}|\texttt{00000000000000000000001}|\texttt{1}|\texttt{1111111}| \\ &= 2^{-23} \cdot 2^{-127} \\ &= 2^{-150} \\ &\approx 7.0 \times 10^{-46} \end{aligned}$$

Note that there are 512 ways to represent zero under the single precision system proposed here (we only require that the mantissa is zero, meaning that the signed bits and exponent can be arbitrary). Under the IEEE standard (which is used by real computers) there are some additional optimizations that take advantage of this "lost" space:

1. use of a signed integer for the exponent
2. only allow for one way to represent zero and use the free space to represent some smaller numbers
3. do not store the first 1 in the mantissa (because it is always 1 for a normalized number)

**Double Precision Numbers.** Double precision is the other standard floating point number system. Numbers are represented on a computer in a 64-bit chunk of memory (8 bytes), divided as follows:

| $s_m$ | m = 52 bits | $s_e$ | e = 10 bits |
|---|---|---|---|

In this case, the maximum and minimum numbers that can be represented are $x_{max} \approx 9 \times 10^{307}$ and $x_{min} \approx 2.5 \times 10^{-324}$, respectively.

### 1.2.3 Machine Precision

Recall that the relative error (defined by equation (1.8)) when converting a real number $x$ to a floating point number is

$$\delta x = \frac{x - \texttt{fl}(x)}{x}. \tag{1.11}$$

This motivates the question: Is there a bound on the absolute value of the relative error $|\delta x|$ for a given floating point number system?

An answer to this requires some delving into the way in which floating point numbers are represented. Consider the floating point number given by

$$\pm 0.x_1x_2\cdots x_m \times b^{\pm t} \tag{1.12}$$

with $1 \leq x_i < b$ for $i = 1, \ldots, m$. We then define the machine epsilon:

**Definition 1.3** *The **machine epsilon** $\epsilon_{mach}$ is the smallest number $\epsilon > 0$ such that* $\texttt{fl}(1+\epsilon) > 1$.

We then present the following proposition:

**Proposition 1.1** *The machine epsilon is given by*

*a)* $\epsilon_{mach} = b^{1-m}$ *if chopping is used*

*b)* $\epsilon_{mach} = \frac{1}{2}b^{1-m}$ *if rounding is used*

**Proof.** For simplicity we only prove part (a) of the proposition. If rounding is used a more complex analysis will yield the result of part (b).

Consider the following subtraction:

$$\begin{array}{lcllllllll}
1+\epsilon & = & 0. & \underbrace{1}_{b^{-1}} & \underbrace{0}_{b^{-2}} & \underbrace{0}_{b^{-3}} & \cdots & \underbrace{0}_{b^{-(m-1)}} & \underbrace{1}_{b^{-m}} & \times b^1 \\
1 & = & 0. & \underbrace{1}_{x_1} & \underbrace{0}_{x_2} & \underbrace{0}_{x_3} & \cdots & \underbrace{0}_{x_{m-1}} & \underbrace{0}_{x_m} & \times b^1 \\
\hline
\epsilon & = & 0. & 0 & 0 & 0 & \cdots & 0 & \underbrace{1}_{b^{-m}} & \times b^1 \\
\epsilon & = & b^{1-m} & & & & & & & 
\end{array}$$

**Theorem 1.1** *For any floating point system $F$ under chopping,*

$$|\delta x| = |\frac{x - \text{fl}(x)}{x}| \leq \epsilon_{mach}. \tag{1.13}$$

**Proof.** Consider the following calculation:

$$\begin{array}{rcll}
x & = & 0.d_1d_2\cdots d_md_{m+1}d_{m+2}\cdots & \cdot b^t \\
\text{fl}(x) & = & 0.d_1d_2\cdots d_m & \cdot b^t
\end{array}$$

Thus combining these two equations yields

$$\begin{array}{rcl}
\frac{x-\text{fl}(x)}{x} & = & \frac{0.00\cdots 0d_{m+1}d_{m+2}\cdots}{0.d_1d_2\cdots d_md_{m+1}d_{m+2}} \\
& = & \frac{0.d_{m+1}d_{m+2}\cdots}{0.d_1d_2\cdots d_md_{m+1}d_{m+2}\cdots} \cdot b^{-m}.
\end{array}$$

But we know the numerator is less than or equal to 1 ($1 = 1 \cdot b^0$) and the denominator is greater than or equal to 0.1 ($0.1 = 1 \cdot b^{-1}$). So we may write:

$$\begin{aligned}(x - \text{fl}(x)) \cdot x^{-1} &\leq (1) \cdot (b^{-1})^{-1} \cdot b^{-m} \\ &= b^{-m+1} \\ &= \epsilon_{mach}\end{aligned}$$

**Note.** Since $\delta x = \frac{1}{x}(x - \text{fl}(x))$, we may also write $\text{fl}(x) = x(1 - \delta x)$ with $|\delta x| \leq \epsilon_{mach}$. Hence we often say

$$\text{fl}(x) = x(1 + \eta) \qquad \text{with } |\eta| \leq \epsilon_{mach}. \tag{1.14}$$

**Single Precision.** Under single precision, $m = 23$ and so $\epsilon = 2^{-22} \approx 0.24 \times 10^{-6}$. Thus $|\delta x| \leq 0.24 \times 10^{-6}$, and so we expect 6 to 7 decimal digits of accuracy.

**Double Precision.** Under double precision, $m = 52$ and so $\epsilon = 2^{-51} \approx 0.44 \times 10^{-15}$. Thus $|\delta x| \leq 0.44 \times 10^{-15}$, and so we expect 15 to 16 decimal digits of accuracy.

### 1.2.4 Floating Point Operations

**Definition 1.4** *The symbol $\oplus$ is used to denote floating point addition, defined as*

$$a \oplus b = \text{fl}(\text{fl}(a) + \text{fl}(b)) \tag{1.15}$$

**Proposition 1.2** *For any floating point number system F,*

$$a \oplus b = (\text{fl}(a) + \text{fl}(b))(1 + \eta) \tag{1.16}$$

*with $|\eta| \leq \epsilon_{mach}$. This may also be written as*

$$a \oplus b = (a(1 + \eta_1) + b(1 + \eta_2))(1 + \eta) \tag{1.17}$$

*with $|\eta_1|, |\eta_2| and |\eta| \leq \epsilon_{mach}$. In general the operation of addition under F is not associative. That is,*

$$(a \oplus b) \oplus c \neq a \oplus (b \oplus c). \tag{1.18}$$

**Note:** There are analogous operations for subtraction, multiplication and division, written using the symbols $\ominus$, $\otimes$, and $\oslash$.

## 1.3 Condition of a Mathematical Problem

Consider a problem **P** with input $\tilde{\mathbf{x}}$ which requires the computation of a desired output $\mathbf{z} = f_P(\tilde{\mathbf{x}})$. In general, some mathematical problems of this form are very sensitive to small deviations in the input data, in which case there are a variety of problems (such as rounding in input data) which make accurate approximation on a computer difficult.

**Definition 1.5** *We say that a problem* $\mathbf{P}$ *is* ***well-conditioned*** *with respect to the absolute error if small changes* $\Delta\vec{x}$ *in* $\vec{x}$ *result in small changes* $\Delta\vec{z}$ *in* $\vec{z}$*. Similarly, we say* $\mathbf{P}$ *is* ***ill-conditioned*** *with respect to the absolute error if small changes* $\Delta\vec{x}$ *in* $\vec{x}$ *result in large changes* $\Delta\vec{z}$ *in* $\vec{z}$*.*

**Definition 1.6** *The condition number of a problem* $\mathbf{P}$ *with respect to the absolute error is given by the* ***absolute condition number*** $\kappa_A$*:*

$$\kappa_A = \| \Delta\vec{z} \| / \|\Delta\vec{x}\|. \tag{1.19}$$

*The condition number with respect to the relative error is given by the* ***relative condition number*** $\kappa_R$*:*

$$\kappa_R = \frac{\| \Delta\vec{z} \| / \| \vec{z} \|}{\|\Delta\vec{x}\|/\|\vec{x}\|} \tag{1.20}$$

If $\kappa_A$ and $\kappa_R$ are "small" we can generally infer that $\mathbf{P}$ is well-conditioned. As a guideline, if $\kappa_A$ and $\kappa_R$ are between 0.1 and 10 we can consider them to be "small." Similarly, if $\kappa_A$ and $\kappa_R$ are "large" (tend to $\infty$ or "blow up") then we can say that $\mathbf{P}$ is ill-conditioned.

**Example 1.** Consider the mathematical problem $\mathbf{P}$ defined by $z = x + y$. We wish to examine how the errors in $x$ and $y$ propagate to $z$. We define our approximations by $\Delta x = x - \hat{x}$ or $\hat{x} = x - \Delta x$, where $\Delta x$ is the error in $x$. Similarly, we have $\Delta y = y - \hat{y}$ or $\hat{y} = y - \Delta y$, where $\Delta y$ is the error in $y$. In solving the problem, we are given the approximations $\hat{x}$ and $\hat{y}$ and compute the approximation $\hat{z}$:

$$\hat{z} = \hat{x} + \hat{y} = (x + y) - (\Delta x + \Delta y).$$

and so we define our error by

$$\Delta z = z - \hat{z} = \Delta x + \Delta y. \tag{1.25}$$

a) **Condition with respect to the absolute error**
We attempt to find an upper bound for $\kappa_A$. Using equation (1.25) and the 1-norm (1.23), we can write

$$\kappa_A = \frac{|\Delta z|}{\|(\Delta x, \Delta y)\|_1} = \frac{|\Delta x + \Delta y|}{|\Delta x| + |\Delta y|}.$$

But by the triangle inequality we have $|\Delta x + \Delta y| \leq |\Delta x| + |\Delta y|$ and so yield

$$\kappa_A \leq \frac{|\Delta x| + |\Delta y|}{|\Delta x| + |\Delta y|} = 1. \tag{1.26}$$

We conclude that $\mathbf{P}$ is well-conditioned with respect to the absolute error.

## Intermission: Vector Norms

Vector norms are a useful tool for providing a measure of the magnitude of a vector, and are particularly applicable to derivations for the condition number.

**Definition 1.7** *Suppose $V$ is a vector space over $\mathbb{R}^n$. Then $\|\cdot\|$ is a* ***vector norm*** *on $V$ if and only if $\|\vec{v}\| \geq 0$, and*

*a)* $\|\vec{v}\| = 0$ *if and only if* $\vec{v} = \vec{0}$

*b)* $\|\lambda\vec{v}\| = |\lambda|\|\vec{v}\| \ \forall \ \vec{v} \in V, \ \forall \ \lambda \in \mathbb{R}$

*c)* $\|\vec{u} + \vec{v}\| \leq \|\vec{u}\| + \|\vec{v}\| \ \forall \ \vec{u}, \vec{v} \in V$ *(triangle inequality)*

There are three standard vector norms known as the 2-norm, the $\infty$-norm and the 1-norm.

**Definition 1.8** *The* ***2-norm*** *over $\mathbb{R}^n$ is defined as*

$$\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{1.21}$$

**Definition 1.9** *The* ***$\infty$-norm*** *over $\mathbb{R}^n$ is defined as*

$$\|\vec{x}\|_\infty = \max_{1 \leq i \leq n} (x_i) \tag{1.22}$$

**Definition 1.10** *The* ***1-norm*** *over $\mathbb{R}^n$ is defined as*

$$\|\vec{x}\|_1 = \sum_{i=1}^{n} |x_i| \tag{1.23}$$

Further, all vector norms that are induced by an inner product (including the 1-norm and 2-norm - but not the $\infty$-norm) satisfy the Cauchy-Schwartz Inequality, which will be of use later:

**Theorem 1.2** *Cauchy-Schwartz Inequality. Let $\|\cdot\|$ be a vector norm over a vector space $V$ induced by an inner product. Then*

$$|\vec{x} \cdot \vec{y}| \leq \|\vec{x}\|\|\vec{y}\| \tag{1.24}$$

b) **Condition with respect to the relative error**
We attempt to find an upper bound for $\kappa_R$. Consider the relative error in $z$, given by

$$|\delta z| = \frac{|\Delta z|}{|z|} = \frac{|\Delta x + \Delta y|}{|x+y|}.$$

We can clearly see here that if $x \approx -y$ then $|\delta z|$ can be very large, even though $|\Delta x|/|x|$ and $|\Delta y|/|y|$ may not be large. The relative condition number is thus

$$\kappa_R = \frac{|\Delta z|/|z|}{\|\Delta \vec{x}\|_1/\|\vec{x}\|_1} = \frac{|\Delta x + \Delta y|/|x+y|}{(|\Delta x| + |\Delta y|)/(|x| + |y|)} \leq \frac{|x| + |y|}{|x+y|}. \tag{1.27}$$

Thus we see that $\kappa_R$ can grow arbitrarily large when $x \approx -y$.

We conclude that the problem $z = x + y$ is ill-conditioned with respect to the relative error only in the case $x \approx -y$.

**Example 2.** Consider the mathematical problem **P** defined by $z = x \cdot y$. We define our approximations by $\Delta x = x - \hat{x}$ and $\Delta y = y - \hat{y}$ as in Example 1. In solving the problem, we are given the approximations $\hat{x}$ and $\hat{y}$ and compute the approximation $\hat{z}$:

$$\begin{aligned} \hat{z} = \hat{x} \cdot \hat{y} &= (x - \Delta x)(y - \Delta y) \\ &= xy - y\Delta x - x\Delta y + \overbrace{\Delta x \Delta y}^{\text{neglect}} \\ &\approx xy - y\Delta x - x\Delta y \end{aligned}$$

and so we define our error by

$$\Delta z \approx y\Delta x + x\Delta y = (x, y) \cdot (\Delta y, \Delta x). \tag{1.28}$$

The relative error in z is given by

$$\delta z \approx \frac{1}{xy}(y\Delta x + x\Delta y) = \frac{\Delta x}{x} + \frac{\Delta y}{y}. \tag{1.29}$$

a) **Condition with respect to the absolute error**
We attempt to find an upper bound for $\kappa_A$. Using equation (1.28) and the Cauchy-Schwartz Inequality (1.24), we can write

$$\kappa_A = \frac{|\Delta z|}{\|(\Delta x, \Delta y)\|} \leq \frac{\|(x,y)\|_2 \|(\Delta x, \Delta y)\|_2}{\|(\Delta x, \Delta y)\|_2} = \|(x,y)\|_2.$$

We conclude that **P** is well-conditioned with respect to the absolute error, except when $x$ or $y$ are large.

b) **Condition with respect to the relative error**
From (1.29) we have that

$$\delta z \approx \frac{\Delta x}{x} + \frac{\Delta y}{y} = \delta x + \delta y$$

and so can immediately conclude that **P** is well-conditioned with respect to the relative error. In fact, in this particular case

$$\kappa_R = \frac{|\Delta z|/|z|}{\|\Delta\vec{x}\|/\|\vec{x}\|}$$

does not easily yield a useful bound.

## 1.4 Stability of a Numerical Algorithm

Consider a problem **P** with input **x** which requires the computation of a desired output $\mathbf{z} = f_P(\mathbf{x})$. If we assume that **P** is well-conditioned, then by definition we have that

$$\|\Delta\vec{x}\| \text{ small} \quad \Rightarrow \quad |\Delta z| \text{ small}$$

$$\|\Delta\vec{x}\|/\|\vec{x}\| \text{ small} \quad \Rightarrow \quad |\Delta z/z| \text{ small}$$

For this well-conditioned problem, some algorithms may be numerically unstable, i.e. they may produce large errors $|\Delta z|$ or $|\delta z|$, while other algorithms may be stable.

**Example 1.9** *Stability with respect to the relative error. Consider the problem* **P** *defined by* $z = \exp(x)$ *with* $x = 5.5$. *The approximate values for* $x$ *and* $z$ *are denoted* $\hat{x}$ *and* $\hat{z}$ *respectively. They are related by the following formula:*

$$\hat{z} = \exp(\hat{x}) = \exp(x - \Delta x) \tag{1.31}$$

A) We investigate the condition of **P** as follows:

a) With respect to $\Delta$ we have

$$\kappa_A = \frac{|\Delta z|}{|\Delta x|} = \frac{|z - \hat{z}|}{|x - \hat{x}|} = \frac{|\exp(x) - \exp(x - \Delta x)|}{|x - \hat{x}|}. \tag{1.32}$$

We apply the Taylor series expansion given by

$$\exp(x - \Delta x) = \exp(x) - \exp(x)\Delta x + O(\Delta x^2) \tag{1.33}$$

to yield,

$$\begin{aligned} \kappa_A &= |\exp(x) - (\exp(x) - \exp(x)\Delta x + O(\Delta x^2))| \ / \ |x - \hat{x}| \\ &\approx |\exp(x)\Delta x| \ / \ |\Delta x| \\ &= |\exp(x)|, \end{aligned}$$

for small $|\Delta x|$. We conclude that the problem is well conditioned with respect to $\Delta$, except for large $x$.

## Intermission: Asymptotic Behaviour of Polynomials

We wish to consider a general method of analyzing the behaviour of polynomials in two cases: when $x \to 0$ and when $x \to \infty$. In particular, if we are only interested in the asymptotic behaviour of the polynomial as opposed to the exact value of the function, we may employ the concept of Big-Oh notation.

**Definition 1.11** *Suppose that $f(x)$ is a polynomial in $x$ without a constant term. Then the following are equivalent:*

*a)* $f(x) = O(x^n)$ *as* $x \to 0$.

*b)* $\exists\ c > 0, x_0 > 0$ *such that* $|f(x)| < c|x|^n\ \forall\ x$ *with* $|x| < |x_0|$.

*c)* $f(x)$ *is bounded from above by* $|x|^n$*, up to a constant* $c$*, as* $x \to 0$.

This effectively means that the dominant term in $f(x)$ is the term with $x^n$ as $x \to 0$, or $f(x)$ goes to zero with order $n$.

**Example 1.7** *Consider the polynomial* $g(x) = 3x^2 + 7x^3 + 10x^4 + 7x^{12}$*. We say*

$$\begin{aligned} g(x) &= O(x^2) \text{ as } x \to 0 \\ g(x) &\neq O(x^3) \text{ as } x \to 0 \\ g(x) &= 3x^2 + O(x^3) \text{ as } x \to 0 \end{aligned}$$

*We note that* $g(x) = O(x)$ *as well, but this statement is not so useful because it is not a sharp bound.*

We may also consider the behaviour of a polynomial as $x \to \infty$.

**Definition 1.12** *Suppose that $f(x)$ is a polynomial in $x$. Then the following are equivalent:*

*a)* $f(x) = O(x^n)$ *as* $x \to \infty$.

*b)* $\exists\ c > 0, x_0 > 0$ *such that* $|f(x)| < c|x|^n\ \forall\ x$ *with* $|x| > |x_0|$.

*c)* $f(x)$ *is bounded from above by* $|x|^n$*, up to a constant* $c$*, as* $x \to \infty$.

As before, this effectively means that the dominant term in $f(x)$ is the term with $x^n$ as $x \to \infty$, or $f(x)$ goes to infinity with order $n$.

**Example 1.8** *Consider the polynomial* $g(x) = 3x^2 + 7x^3 + 10x^4 + 7x^{12}$*. We say*

$$\begin{aligned} g(x) &= O(x^{12}) \text{ as } x \to \infty \\ g(x) &\neq O(x^8) \text{ as } x \to \infty \\ g(x) &= 7x^{12} + O(x^4) \text{ as } x \to \infty \end{aligned}$$

### Addition and Multiplication of Terms Involving Big-Oh

We can also add and multiply terms using Big-Oh notation, making sure to neglect higher order terms:

$$\begin{array}{rcll} f(x) & = & x + O(x^2) & \text{as } x \to 0 \\ g(x) & = & 2x + O(x^3) & \text{as } x \to 0 \\ \hline f(x) + g(x) & = & 3x + O(x^2) + O(x^3) & \text{as } x \to 0 \\ & = & 3x + O(x^2) & \text{as } x \to 0 \\ f(x) \cdot g(x) & = & 2x^2 + O(x^3) + O(x^4) + O(x^5) & \text{as } x \to 0 \\ & = & 2x^2 + O(x^3) & \text{as } x \to 0 \end{array}$$

### Applications to the Taylor Series Expansion

Recall that the Taylor series expansion for a function $f(x)$ around a point $x_0$ is given by

$$f(x_0 + \Delta x) = \sum_{n=0}^{\infty} \frac{1}{n!} f^{(n)}(x_0)(\Delta x)^n \tag{1.30}$$

We may expand this formula and write it in terms of Big-Oh notation as follows:

$$\begin{array}{rcl} f(x_0 + \Delta x) & = & f(x_0) \\ & + & f'(x_0)\Delta x \\ & + & \frac{1}{2} f''(x_0)\Delta x^2 \\ & + & \frac{1}{6} f'''(x_0)\Delta x^3 \\ & + & O(\Delta x^4) \text{ as } \Delta x \to 0 \end{array}$$

b) With respect to $\delta$ we have

$$\kappa_R = \frac{|\Delta z|/|z|}{|\Delta x|/|x|} = \left|\frac{\Delta z}{\Delta x}\right|\left|\frac{x}{z}\right| \approx |\exp(x)|\frac{|x|}{|\exp(x)|} = |x|. \tag{1.34}$$

We conclude that the problem is well conditioned with respect to $\delta$, except for large $x$.

B) Now that we know that problem P is well conditioned, we can investigate the stability of our algorithms for **P**.

Recall Algorithm A, given by (1.2). We determined that this algorithm is unstable with respect to $\delta$ for $x < 0$. This was due to a cancellation of nearly equal values in the subtraction, and so the algorithm was unstable due to ill-conditioned steps that could be avoided.

Recall Algorithm B, given by (1.9). As opposed to Algorithm A, this algorithm is stable with respect to $\delta$ for $x < 0$.

**Example 1.10** *Instability with respect to absolute error $\Delta$. Consider the problem* **P** *defined by $z = \int_0^1 \frac{x^n}{x+\alpha}dx$ where $\alpha > 0$. The approximate values for $\alpha$ and $z$ are denoted $\hat{\alpha}$ and $\hat{z}$ respectively. It can be shown that* **P** *is well-conditioned (for instance with respect to the integration boundaries 0 and 1).*

We derive an algorithm for solving this problem using a recurrence relation. In deriving a recurrence, we need to consider the base case (a) and the recursive case (b) and then ensure that they are appropriately related.

a) Consider $n = 0$:

$$\begin{aligned} I_0 &= \int_0^1 \frac{1}{x+\alpha}dx \\ &= \log(x+\alpha)\,\Big|_0^1 \\ &= \log(1+\alpha) - \log(\alpha) \end{aligned}$$

and so we get

$$I_0 = \log(\frac{1+\alpha}{\alpha}). \tag{1.35}$$

b) For general $n$ we can derive the following recurrence:

$$\begin{aligned} I_n &= \int_0^1 \frac{x^{n-1}x}{x+\alpha}dx \\ &= \int_0^1 \frac{x^{n-1}(x+\alpha-\alpha)}{x+\alpha}dx \\ &= \int_0^1 x^{n-1}dx - \alpha\int_0^1 \frac{x^{n-1}}{x+\alpha}dx \\ &= \frac{x^n}{n}\,\Big|_0^1 - \alpha I_{n-1} \end{aligned}$$

which yields the expression

$$I_n = \frac{1}{n} - \alpha I_{n-1}. \tag{1.36}$$

Thus we may formulate an algorithm using the base case and the recurrence:

**Algorithm A.**

1. Calculate $I_0$ from (1.35).
2. Calculate $I_1, I_2, \ldots, I_n$ using (1.36).

An implementation of this algorithm on a computer provides the following results, for two sample values of $\alpha$:

$$\begin{array}{rclcrcl} \alpha & = & 0.5 & \longrightarrow & I_{100} & \approx & 6.64 \times 10^{-3} \\ \alpha & = & 2.0 & \longrightarrow & I_{100} & \approx & 2.1 \times 10^{22} \end{array}$$

For $\alpha = 2.0$, we obtain a very large result! This may be a first indication that something is wrong. From the original equation we have that

$$I_{100} = \int_0^1 \frac{x^{100}}{x+\alpha} dx \leq \frac{1}{1+\alpha} \cdot 1 = 1/3. \tag{1.37}$$

Hence we conclude something is definitely amiss. We need to consider the propagation of the error in our algorithm, since we know that the algorithm is definitely correct using exact arithmetic.

Consider $I_n$ to be the exact value at each step of the algorithm, with $\hat{I}_n$ the numerical approximation. Then the absolute error at each step of the algorithm is given by $\Delta I_n = I_n - \hat{I}_n$. Thus the initial error is $\Delta I_0 = I_0 - \hat{I}_0 = I_0 - \mathrm{fl}(I_0)$.

The the exact value satisfies

$$I_n = \frac{1}{n} - \alpha I_{n-1} \tag{1.38}$$

and the numerical approximation is calculated from

$$\hat{I}_n = \frac{1}{n} - \alpha \hat{I}_{n-1} + \eta_n, \tag{1.39}$$

where $\eta_n$ is the rounding error we introduce in step $n$.

For a first analysis, we neglect $\eta_n$ and simply investigate the propagation of the initial error $\Delta I_0$ only. Then

$$\begin{array}{rcl} \Delta I_n & = & I_n - \hat{I}_n \\ & = & (\frac{1}{n} - \alpha I_{n-1}) - (\frac{1}{n} - \alpha \hat{I}_{n-1}) \\ & = & -\alpha (I_{n-1} - \hat{I}_{n-1}) \\ & = & -\alpha \Delta I_{n-1}. \end{array}$$

Applying recursion yields an expression for the accumulated error after $n$ steps:

$$\Delta I_n = (-\alpha)^n \Delta I_0. \tag{1.40}$$

**Conclusion.** From this expression, we note that there are potentially two very different outcomes:

a) If $\alpha > 1$ then the initial error is propagated in such a way that blow-up occurs. Thus Algorithm A is numerically unstable with respect to $\Delta$.

b) If $\alpha \leq 1$ then the initial error remains constant or dies out. Thus Algorithm A is numerically stable with respect to $\Delta$.

We note that further analysis would lead to the conclusion that the rounding error $\eta_n$ is also propagated in the same way as the initial error $\Delta I_0$. These results confirm the experimental results observed in the implementation.

**These notes have been funded by...**

# Chapter 2

# Root Finding

## 2.1 Introduction

The root finding problem is a classical example of numerical methods in practice. The problem is stated as follows:

**Problem** Given any function $f(x)$, find $x^*$ such that $f(x^*) = 0$. The value $x^*$ is called a root of the equation $f(x) = 0$.

If $f(x) = 0$ has a root $x^*$ there is no truly guaranteed method for finding $x^*$ computationally for an arbitrary function, but there are several techniques which are useful for many cases. A computational limitation inherent to this problem can be fairly easily seen by an observant reader: any interval of the real line contains an infinite number of points, but computationally we can solve this problem with only a finite number of evaluations of the function $f$.

Additionally, since the value $x^*$ may not be defined in our floating point number system, we will not be able to find $x^*$ exactly. Therefore, we consider a computational version of the same problem:

**Problem (Computational)** Given any $f(x)$ and some error tolerance $\epsilon > 0$, find $x^*$ such that $|f(x^*)| < \epsilon$.

We will only consider functions which are continuous in our analysis.

**Example 2.1**

1) *Consider the function* $f(x) = x^2 + x - 6 = (x - 2)(x + 3)$. *The function* $f(x) = 0$ *has two roots at x=2, -3.*

2) *With* $f(x) = x^2 - 2x + 1 = (x - 1)^2 = 0$, $x = 1$ *is a double root (i.e.* $f(x^*) = 0$ *and* $f'(x^*) = 0$*).*

*3)* $f(x) = 3x^5 + 5x^4 + \frac{1}{3}x^3 + 1 = 0$. *We have no general closed form solution for the roots of a polynomial with degree larger than 4. As a result, we will need to use numerical approximation by iteration.*

*4)* $f(x) = x^2 - \frac{1}{2}\exp(-x) = 0$. *This is naturally more difficult to solve because* $\exp(-x)$ *is a transcendental function. In fact,* $f(x) = 0$ *is called the transcendental equation and can only be solved with numerical approximation.*

**Definition 2.1** *We say that* $x^*$ *is a* ***double root*** *of* $f(x) = 0$ *if and only if* $f(x^*) = 0$ *and* $f'(x^*) = 0$.

We naturally examine this computational problem from an iterative standpoint. That is, we wish to generate a sequence of iterates $(x_k)$ such that any iterate $x_{k+1}$ can be written as some function of $x_k, x_{k-1}, \ldots, x_0$. We assume that some initial conditions are applied to the problem so that $x_p, x_{p-1}, \ldots x_0$ are either given or arbitrarily chosen. Obviously, we require that the iterates actually converge to the solution of the problem, *i.e.* $x^* = \lim_{k\to\infty} x_k$.

A natural question to ask might be, "how do we know where a root of a function may approximately be located?" A simple result from first year calculus will help in answering this:

**Theorem 2.1 (Intermediate Value Theorem)** *If* $f(x)$ *is continuous on a closed interval* $[a, b]$ *and* $c \in [f(a), f(b)]$, *then* $\exists\ x^* \in [a, b]$ *such that* $f(x^*) = c$.

Thus if we can find $[a, b]$ such that $f(a) \cdot f(b) < 0$ then by the Intermediate Value Theorem, $[a, b]$ will contain at least one root $x^*$ as long as $f(x)$ is continuous.

## 2.2 Four Algorithms for Root Finding

### 2.2.1 Bisection Method

The bisection method is one of the most simple methods for locating roots, but is also very powerful as it guarantees convergence as long as its initial conditions are met. To apply this method, we require a continuous function $f(x)$ and an initial interval $[a, b]$ such that $f(a) \cdot f(b) \leq 0$. This method effectively works by bisecting the interval and recursively using the Intermediate Value Theorem to determine a new interval where the initial conditions are met.

**Theorem 2.2** *If* $f(x)$ *is a continuous function on the interval* $[a_0, b_0]$ *such that* $f(a_0) \cdot f(b_0) \leq 0$ *then the interval* $[a_k, b_k]$, *defined by*

$$a_k = \begin{cases} a_{k-1} & if\ f((a_{k-1} + b_{k-1})/2) \cdot f(a_{k-1}) \leq 0 \\ (a_{k-1} + b_{k-1})/2 & otherwise \end{cases} \tag{2.1}$$

$$b_k = \begin{cases} b_{k-1} & if\ f((a_{k-1} + b_{k-1})/2) \cdot f(a_{k-1}) > 0 \\ (a_{k-1} + b_{k-1})/2 & otherwise \end{cases} \tag{2.2}$$

*fulfills the property* $f(a_k) \cdot f(b_k) \leq 0\ \forall\ k$.

**Algorithm: Bisection Method**
**in:** $f(x)$, $[a, b]$, tolerance $t$
**out:** $x$, an approximation for $x^*$

```
while |b-a| > t
  c = (a+b)/2
  if f(a)*f(c) <= 0
    keep a, b = c
  else
    keep b, a = c
  end if
end while
x = (a+b)/2
```

When applying the bisection method, we only require continuity of the function $f(x)$ and an initial knowledge of two points $a_0$ and $b_0$ such that $f(a_0) \cdot f(b_0) \leq 0$. We are guaranteed the existence of a root in the interval $[a_0, b_0]$ by the Intermediate Value Theorem (2.1) and are further guaranteed that the bisection method will converge to a solution.

We consider the question of "speed of convergence," namely given $a$, $b$ and $t$, how many steps does it take to reach $t$? If we suppose that $x^* = \lim_{k\to\infty} x_k$ then at each iteration the interval containing $x^*$ is halved. Thus, assuming it takes $n$ steps to fulfill $|b-a| \leq t$ we have that

$$\begin{array}{rrcl} & 2^{-n}|b-a| & \leq & t \\ \Rightarrow & n\log 2 & \geq & \log(\frac{|b-a|}{t}) \\ \Rightarrow & n & \geq & \frac{1}{\log 2}\log(\frac{|b-a|}{t}). \end{array}$$

Thus we conclude that for a given tolerance t and initial interval $[a, b]$, bisection will take

$$n \geq \frac{1}{\log 2}\log(\frac{|b-a|}{t}) \tag{2.3}$$

steps to converge.

**Example 2.2** *Given $|b-a| = 1$ and $t = 10^{-6}$, how many steps does it take to converge? From (2.3) we have that*

$$\begin{array}{rcl} n & \geq & \dfrac{1}{\log_{10}(2)}\log_{10}(10^6) \\ \Rightarrow n & \geq & 3.4 \cdot 6 \\ \Rightarrow n & \geq & 20. \end{array}$$

*(compare with $2^{20} \approx 1.05 \times 10^6$).*

### 2.2.2 Fixed Point Iteration

We note that we may rewrite the root-finding problem in an alternate way that may not be immediately obvious. Consider the real-valued function $g$, defined by $g(x) = x - f(x)$. We note that this function inherits the continuity of $f$ in an interval $[a, b]$. We can also write $f(x) = x - g(x)$ in order to obtain our original function $f(x)$. The problem of root-finding for our original function is hence equivalent to the problem of finding a solution to $g(x) = x$.

**Definition 2.2** *We say that $x^*$ is a **fixed point** of $g(x)$ if $g(x^*) = x^*$, i.e. if $x^*$ is mapped to itself under $g$.*

We note that if our function $g$ has certain desirable properties (in particular, as will be shown later, if $|g'(x^*)| < 1$ and $x_0$ is "close enough" to $x^*$), then repeated application of $g$ will actually cause us to converge to this fixed point. This implies we can write our algorithm for fixed-point iteration as follows:

**Algorithm: Fixed Point Iteration**
**in:** $g(x)$, $x_0$, tolerance $t$
**out:** $x$, an approximation for $x^*$

```
i = 0
repeat
  i = i + 1
  x[i] = g(x[i-1])
until |x[i] - x[i-1]| < t
x = x[i]
```

We note that it is not required that we limit ourselves to the choice of $g(x) = x - f(x)$ in applying this scheme. In general we can write $g(x) = x - H(f(x))$ as long as we choose $H$ such that $H(0) = 0$. Not all choices will lead to a converging method. For convergence we must also choose $H$ such that $|g'(x^*)| < 1$ (see later).

### 2.2.3 Newton's Method

Certain additional information may be available about our function that may assist in constructing a root-finding method. In particular, knowledge of the first derivative of $f(x)$ motivates the use of Newton's method. Consider the Taylor series expansion of $f(x^*)$ about an initial estimate $x_0$:

$$f(x^*) = f(x_0) + f'(x_0)(x^* - x_0) + O((x^* - x)^2). \tag{2.4}$$

If we take this sequence to leading order, we have that

$$f(x^*) \approx f(x_0) + f'(x_0)(x^* - x_0). \tag{2.5}$$

But we know that $f(x^*) = 0$ and so we find a new, often better approximation $x_1$ from $x_0$ by requiring that

$$f(x_0) + f'(x_0)(x_1 - x_0) = 0. \tag{2.6}$$

Rearranging and taking the sequence in general yields the defining equation for Newton's method:

$$x_{i+1} = x_i - f(x_i)/f'(x_i). \tag{2.7}$$

We note that we will need to look out for the case where $f'(x_i) = 0$, since this will lead to a division by zero. Otherwise, this derivation allows us to provide an algorithm for applying Newton's method numerically:

**Algorithm: Newton's Method**
**in:** $f(x)$, $f'(x)$, $x_0$, tolerance $t$
**out:** $x$, an approximation for $x^*$

```
i = 0
x[0] = x0
repeat
  i = i + 1
  if f'(x[i-1]) == 0 stop
  x[i] = x[i-1] - f(x[i-1]) / f'(x[i-1])
until |x[i] - x[i-1]| < t
x = x[i]
```

### 2.2.4 Secant Method

Newton's method provides very fast convergence, but relies on the knowledge of $f'(x)$. If this derivative is not known, not easily computable, or if $f(x)$ is not explicitly given but is the output of a computer program, then we will be unable to apply Newton's method. However, we can approximate the derivative using a numerical scheme that requires only evaluations of the function $f(x)$. From the definition of the derivative we know that

$$f'(x_i) = \lim_{\eta \to x_i} \frac{f(x_i) - f(\eta)}{x_i - \eta} \approx \frac{f(x_i) - f(\eta)}{x_i - \eta}. \tag{2.8}$$

If we choose $\eta = x_{i-1}$ then we approximate the derivative as

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}. \tag{2.9}$$

This result can be plugged into Newton's method to give the defining equation for the Secant method:

$$x_{i+1} = x_i - f(x_i)\left[\frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}\right]. \tag{2.10}$$

Note that this method actually requires the two previous values ($x_i$ and $x_{i-1}$) in order to compute $x_{i+1}$. Thus, we also need two initial values $x_0$ and $x_1$ in order to begin iteration. Also, as in Newton's method where we needed to check for $f'(x_i) = 0$ here we need to be wary of the case where $f(x_i) \approx f(x_{i-1})$, as this will potentially give undesirable results.

### 2.2.5 Stopping Criteria for Iterative Functions

For any iterative algorithm that approximates a root $x^*$ we need to consider a stopping criterion. There are several general criteria for stopping, which can be combined if necessary.

1. **Maximum number of steps.** Using this method for stopping, we impose some maximum number of steps in the iteration $i_{max}$ and stop when $i = i_{max}$. This provides a safeguard against infinite loops, but is not very efficient - *i.e.* even if we are very close to the root after 1 iteration, this method will always run for the same number of iterations.

2. **Tolerance on the size of the correction.** Under this criterion, we are given some tolerance $t$ and stop when $|x_{i+1} - x_i| \leq t$. Under the bisection method and fixed-point iteration (with "spiral-wise" convergence, see later) we actually are able to guarantee that $|x_{i+1} - x^*| \leq t$. Unfortunately, this criterion does not guarantee that $|x_{i+1} - x^*| \approx t$, in general.

   This may not work very well if one desires a small function value in the approximate root for steep functions such as $f(x) = a(x - x^*)$ with $a = 10^{11}$. Even a small error in $x$ will mean a large value for $f(x)$.

3. **Tolerance on the size of the function value.** Under this criterion, we are given some tolerance $t$ and stop when $|f(x_i)| < t$. This may not work well for a flat function such as $f(x) = a(x - x^*)$ with $a = 10^{-9}$. In this case, for $x_i$ far from $x^*$, $|f(x_i)|$ may be smaller than $t$.

In conclusion, choosing a good stopping criterion is difficult and dependent on the problem. Often trial and error is used to determine a good criterion and combines any of the aforementioned options.

## 2.3 Rate of Convergence

We wish to examine how quickly each of the four methods discussed here converges to the root $x^*$, assuming that convergence occurs. In section 2.4 we will discuss the criteria necessary for convergence of each method.

In order to lay a foundation for this discussion, we must consider how rate of convergence is measured. We first consider the error at each step of the iteration:

**Definition 2.3** *For a sequence $\{x_i\}_{i=0}^{\infty}$ and point $x^*$, the error at iteration $i$ is*

$$e_i = x_i - x^*. \tag{2.11}$$

We will define the rate of convergence by how quickly the error converges to zero (and hence how quickly $\{x_i\}_{i=0}^{\infty}$ converges to $x^*$). If $\{x_i\}_{i=0}^{\infty}$ diverges from $x^*$, then we note that $\lim_{i\to\infty} e_i = \pm\infty$.

**Definition 2.4** *The sequence $\{x_i\}_{i=1}^{\infty}$ converges to $x^*$ with order $q$ if and only if $\{x_i\}_{i=1}^{\infty}$ converges to $x^*$, $\lim_{i\to\infty} c_i = N$ when $N \in [0,\infty)$ and*

$$|e_{i+1}| = c_i|e_i|^q. \tag{2.12}$$

With these definitions in mind, we may consider the rate of convergence of each of our iteration methods. Consider the example given in Table 2.1 and Table 2.2. We wish to determine the positive root of $f(x) = x^2 - \frac{1}{2}\exp(-x)$ which has the exact value $x^* = 0.53983527690282$. We measure the value of the iterate $x_i$ in Table 2.1 and the value of the error $e_i$ in Table 2.2.

**Bisection Method.** From the derivation (2.3) we note that, on average, $|e_{i+1}| \approx \frac{1}{2}|e_i|$. But the error may increase for certain iterations, depending on the initial interval. Thus, we cannot directly apply the definition for convergence to the bisection method, but we nonetheless say that the method behaves like a linearly convergent method, with the following justification:

Consider the sequence defined by $\{L_i\}_{i=1}^{\infty}$ with $L_i = |b_i - a_i|$ the length of the interval at step $i$. We know that $L_{i+1} = \frac{1}{2}L_i$ and so the sequence $\{L_i\}$ converges to 0 linearly. We also know that $|e_i| \le L_i$, and so we say that $\{e_i\}$ converges to 0 at least linearly.

**Fixed Point Iteration.** The rate of convergence for this method is highly variable and depends greatly on the actual problem being solved. In the example at hand, we find that if we define $c_i$ by $|e_{i+1}| = c_i|e_i|$ then, on average, it appears that $\lim_{i\to\infty} c_i = 0.37$. Thus we note that fixed point iteration converges linearly as well.

**Newton's Method.** A thorough analysis of Newton's method indicates that Newton actually converges much faster than the other methods. In the example at hand, if we consider $|e_{i+1}| = c_i|e_i|^2$ we will find that we get $\lim_{i\to\infty} c_i = 0.62$. Thus Newton's method converges quadratically in this example.

**Secant Method.** With a thorough analysis of the Secant method, we find that the Secant method converges faster than fixed point iteration, but slower than Newton's method. If we consider $|e_{i+1}| = c_i|e_i|^q$ we actually find that $q = \frac{1}{2}(1+\sqrt{5}) \approx 1.618$. In the example at hand, we actually get that $\lim_{i\to\infty} c_i \approx 0.74$.

## 2.4 Convergence Theory

### 2.4.1 Fixed Point Iteration

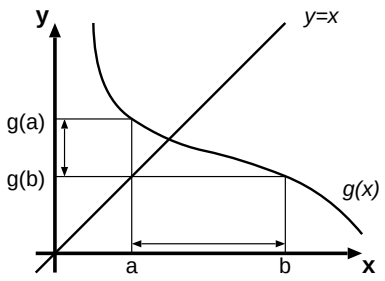
For fixed point iteration, we construct the iteration sequence $x_{i+1} = g(x_i)$ and iterate to approximate the fixed point $x^* = g(x^*)$. We demonstrated previously that this is equivalent to root-finding if we use $g(x) = x - H(f(x))$ for any function $H(x) = H(f(x))$ where $H(0) = 0$. Since $f(x^*) = 0$ by the definition of a root, we also have that $H(x^*) = H(f(x^*)) = H(0) = 0$ and so $g(x^*) = x^*$. We will show now that this method converges when $|g'(x^*)| < 1$.

The theory for the fixed point iteration method goes hand in hand with the theory behind contractions and contraction mapping from real analysis:

**Definition 2.5** *Suppose that $g$ is a real-valued function, defined and continuous on a bounded closed interval $[a, b]$ of the real line. Then, $g$ is said to be a* ***contraction*** *on $[a, b]$ if there exists a constant $L \in (0, 1)$ such that*

$$|g(x) - g(y)| \leq L|x - y| \qquad \forall x, y \in [a, b] \tag{2.16}$$

The definition of a contraction has two very important graphical interpretations:




For one, we notice $|g(a) - g(b)|$ is smaller than $|a - b|$ and so we may say that the interval $[a, b]$ has been contracted to a smaller interval $[g(a), g(b)]$.

**Table 2.1: Root-Finding Iteration $x_i$ ($f(x) = x^2 - \frac{1}{2}\exp(-x)$)**

| Bisection | Fixed Point Iteration | Newton | Secant |
|---|---|---|---|
| 1.00000000000000 | 1.00000000000000 | 2.00000000000000 | 2.00000000000000 |
| 0.50000000000000 | 0.18393972058572 | 1.03327097864435 | 0.00000000000000 |
| 0.75000000000000 | 0.56609887674714 | 0.63686054270010 | 0.22561484995794 |
| 0.62500000000000 | 0.52949890451207 | 0.54511924037555 | 0.74269471761919 |
| 0.56250000000000 | 0.54357981007437 | 0.53985256974508 | 0.49760551690233 |
| 0.53125000000000 | 0.53843372706828 | 0.53983527708914 | 0.53494633480233 |
| 0.54687500000000 | 0.54035370380524 | 0.53983527690282 | 0.53996743772003 |
| 0.53906250000000 | 0.53964266286324 | 0.53983527690282 | 0.53983487323262 |
| 0.54296875000000 | 0.53990672286905 | 0.53983527690282 | 0.53983527686958 |
| 0.54101562500000 | 0.53980875946698 | 0.53983527690282 | 0.53983527690282 |
| 0.54003906250000 | 0.53984511672844 | 0.53983527690282 | 0.53983527690282 |
| 0.53955078125000 | 0.53983162533285 | 0.53983527690282 | 0.53983527690282 |
| 0.53979492187500 | 0.53983663196232 | 0.53983527690282 | 0.53983527690282 |
| 0.53991699218750 | 0.53983477404859 | 0.53983527690282 | 0.53983527690282 |
| 0.53985595703125 | 0.53983546350813 | 0.53983527690282 | 0.53983527690282 |
| 0.53982543945312 | 0.53983520765493 | 0.53983527690282 | 0.53983527690282 |
| 0.53984069824219 | 0.53983530260020 | 0.53983527690282 | 0.53983527690282 |
| 0.53983306884766 | 0.53983526736671 | 0.53983527690282 | 0.53983527690282 |
| 0.53983688354492 | 0.53983528044160 | 0.53983527690282 | 0.53983527690282 |
| 0.53983497619629 | 0.53983527558960 | 0.53983527690282 | 0.53983527690282 |
| 0.53983592987061 | 0.53983527739014 | 0.53983527690282 | 0.53983527690282 |
| 0.53983545303345 | 0.53983527672198 | 0.53983527690282 | 0.53983527690282 |
| 0.53983521461487 | 0.53983527696993 | 0.53983527690282 | 0.53983527690282 |
| 0.53983533382416 | 0.53983527687792 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527421951 | 0.53983527691206 | 0.53983527690282 | 0.53983527690282 |
| 0.53983530402184 | 0.53983527689939 | 0.53983527690282 | 0.53983527690282 |
| 0.53983528912067 | 0.53983527690409 | 0.53983527690282 | 0.53983527690282 |
| 0.53983528167009 | 0.53983527690235 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527794480 | 0.53983527690300 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527608216 | 0.53983527690276 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527701348 | 0.53983527690284 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527654782 | 0.53983527690281 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527678065 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527689707 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527695527 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527692617 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527691162 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690434 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690070 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690252 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690343 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690298 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690275 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690286 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690281 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690283 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690282 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690281 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690282 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690282 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |
| 0.53983527690282 | 0.53983527690282 | 0.53983527690282 | 0.53983527690282 |

**Table 2.2: Root-Finding Iteration $e_i$ ($f(x) = x^2 - \frac{1}{2}\exp(-x)$)**

| Bisection | Fixed Point Iteration | Newton | Secant |
|---|---|---|---|
| 0.46016472309718 | 0.46016472309718 | 1.46016472309718 | 1.46016472309718 |
| -0.03983527690282 | -0.35589555631710 | 0.49343570174153 | -0.53983527690282 |
| 0.21016472309718 | 0.02626359984432 | 0.09702526579728 | -0.31422042694488 |
| 0.08516472309718 | -0.01033637239075 | 0.00528396347273 | 0.20285944071637 |
| 0.02266472309718 | 0.00374453317155 | 0.00001729284226 | -0.04222976000049 |
| -0.00858527690282 | -0.00140154983454 | 0.00000000018632 | -0.00488894210049 |
| 0.00703972309718 | 0.00051842690242 | 0.00000000000000 | 0.00013216081721 |
| -0.00077277690282 | -0.00019261403958 | 0.00000000000000 | -0.00000040367020 |
| 0.00313347309718 | 0.00007144596623 | 0.00000000000000 | -0.00000000003324 |
| 0.00118034809718 | -0.00002651743584 | 0.00000000000000 | 0.00000000000000 |
| 0.00020378559718 | 0.00000983982562 | 0.00000000000000 | 0.00000000000000 |
| -0.00028449565282 | -0.00000365156997 | 0.00000000000000 | 0.00000000000000 |
| -0.00004035502782 | 0.00000135505950 | 0.00000000000000 | 0.00000000000000 |
| 0.00008171528468 | -0.00000050285423 | 0.00000000000000 | 0.00000000000000 |
| 0.00002068012843 | 0.00000018660531 | 0.00000000000000 | 0.00000000000000 |
| -0.00000983744969 | -0.00000006924789 | 0.00000000000000 | 0.00000000000000 |
| 0.00000542133937 | 0.00000002569738 | 0.00000000000000 | 0.00000000000000 |
| -0.00000220805516 | -0.00000000953611 | 0.00000000000000 | 0.00000000000000 |
| 0.00000160664210 | 0.00000000353878 | 0.00000000000000 | 0.00000000000000 |
| -0.00000030070653 | -0.00000000131322 | 0.00000000000000 | 0.00000000000000 |
| 0.00000065296779 | 0.00000000048732 | 0.00000000000000 | 0.00000000000000 |
| 0.00000017613063 | -0.00000000018084 | 0.00000000000000 | 0.00000000000000 |
| -0.00000006228795 | 0.00000000006711 | 0.00000000000000 | 0.00000000000000 |
| 0.00000005692134 | -0.00000000002490 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000268331 | 0.00000000000924 | 0.00000000000000 | 0.00000000000000 |
| 0.00000002711902 | -0.00000000000343 | 0.00000000000000 | 0.00000000000000 |
| 0.00000001221785 | 0.00000000000127 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000476727 | -0.00000000000047 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000104198 | 0.00000000000018 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000082066 | -0.00000000000006 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000011066 | 0.00000000000002 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000035500 | -0.00000000000001 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000012217 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000575 | -0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000005245 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000002335 | -0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000880 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000152 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000212 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000030 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000061 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000016 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000007 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000004 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000001 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000001 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| 0.00000000000000 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000001 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000000 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000000 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |
| -0.00000000000000 | 0.00000000000000 | 0.00000000000000 | 0.00000000000000 |

## Intermission: The Golden Ratio

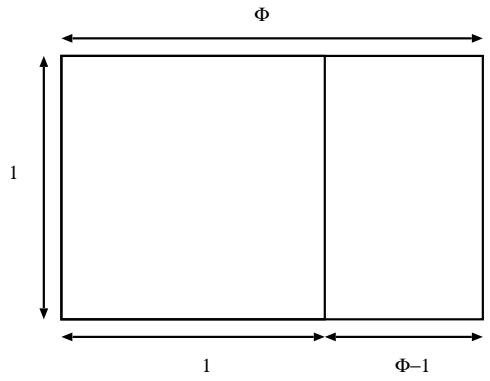



Figure 2.1: Two nested rectangles with equivalent aspect ratios

The golden ratio appears often in nature and in mathematics. It can be defined using two nested rectangles that have equivalent aspect ratios, as depicted in figure 2.1. The aspect ratio of the larger rectangle is given by $\frac{1}{\phi}$ and the smaller rectangle has the aspect ratio $\frac{\phi-1}{1}$. Equating these gives

$$\frac{1}{\phi} = \frac{1-\phi}{1}, \tag{2.13}$$

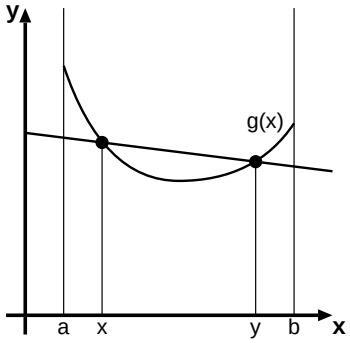
or equivalently

$$\phi^2 - \phi - 1 = 0. \tag{2.14}$$

Finally, we apply the quadratic formula and take the positive root to get

$$\phi_{1,2} = \frac{1+\sqrt{5}}{2}, \tag{2.15}$$

the golden ratio.




Also, we note that for any $x, y \in [a, b]$ such that $x \neq y$ a simple manipulation indicates that a contraction fulfills

$$\frac{|g(x) - g(y)|}{|x - y|} \leq L < 1. \tag{2.17}$$

Thus we notice that the slope of any secant line within the interval $[a, b]$ cannot exceed L in absolute value.

An observant reader might notice that this definition of a contraction appears very similar to the definition for a derivative. In fact, if we have $g(x)$ differentiable on $[a, b]$ with $|g'(x)| < 1 \; \forall \; x \in [a, b]$, then $g(x)$ is a contraction on $[a, b]$ with

$$L = \max_{x \in [a,b]} |g'(x)|.$$

The proof of this fact is left as an exercise for the reader.

The definition of a contraction leads to a very important theorem that governs the behaviour of the contraction and, in fact, gives us the result we require for fixed point iteration.

**Theorem 2.3 (Contraction Mapping Theorem)** *Let $g$ be a real-valued function, defined and continuous on a bounded closed interval $[a, b]$ of the real line, and assume that $g(x) \in [a, b]$ for all $x \in [a, b]$. Suppose, further, that $g$ is a contraction on $[a, b]$. Then,*

1. *$g$ has a unique fixed point $x^*$ in the interval $[a, b]$.*
2. *The sequence $\{x_k\}$ defined by $x_{k+1} = g(x_k)$ converges to $x^*$ as $k \to \infty$ for any starting value $x_0$ in $[a, b]$.*

**Proof: Existence of the fixed point.** The existence of a fixed point $x^*$ for $g$ is a consequence of the Intermediate Value Theorem. Define $u(x) = x - g(x)$. Then

$$u(a) = a - g(a) \leq 0 \qquad \text{and} \qquad u(b) = b - g(b) \geq 0.$$

Then by the Intermediate Value Theorem, there exists $x^* \in [a, b]$ such that $u(x^*) = 0$. Thus $x^* - g(x^*) = 0$, or equivalently $x^* = g(x^*)$ and so $x^*$ is a fixed point of $g$.

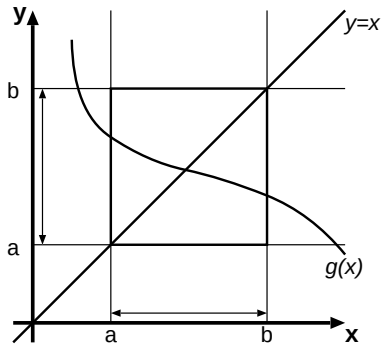



Figure 2.2: An illustration of the contraction mapping theorem.

**Uniqueness of the fixed point.** The uniqueness of this fixed point follows from (2.16) by contradiction. Suppose that $g$ has a second fixed point, $x_2^*$, in $[a, b]$ such that $g(x^*) = x^*$ and $g(x_2^*) = x_2^*$. Then,

$$|g(x^*) - g(x_2^*)| \leq L|x^* - x_2^*|, \qquad \text{(contraction property)}.$$

Using the definition of a fixed point, we have

$$|x^* - x_2^*| \leq L|x^* - x_2^*|,$$

or equivalently, $L \geq 1$. However, from the contraction property we know $L \in (0, 1)$. Thus we have a contradiction and so there is no second fixed point.

**Convergence property.** Let $x_0$ be any element of $[a, b]$. Consider the sequence $\{x_i\}$ defined by $x_{i+1} = g(x_i)$, where $x_i \in [a, b]$ implies $x_{i+1} \in [a, b]$. We note for any $x_{i-1}$ in the interval we have, by the contraction property

$$|g(x_{i-1}) - g(x^*)| \leq L|x_{i-1} - x^*|,$$

or equivalently

$$|x_i - x^*| \leq L|x_{i-1} - x^*|.$$

Using the fact that this applies for all $i$, we may use recursion to get

$$|x_i - x^*| \leq L^i|x_0 - x^*|.$$

We take the limit as $i \to \infty$ to get

$$\lim_{i\to\infty} |x_i - x^*| \leq |x_0 - x^*| \lim_{i\to\infty} L^i.$$

But since we also know that $L \in (0, 1)$ then $\lim_{i\to\infty} L^i = 0$. Thus our equation reduces to

$$\lim_{i\to\infty} |x_i - x^*| = 0,$$

or

$$\lim_{i\to\infty} x_i = x^*. \qquad \square$$

From the contraction mapping theorem, we note that it appears that convergence to the fixed point $x^*$ appears to be linear; in particular, we get from the contraction property that $e_i \leq Le_{i-1}$.

Intuitively, only one fixed point is allowed since we require a slope greater than 1 to get multiple fixed points (see Figure 2.3).

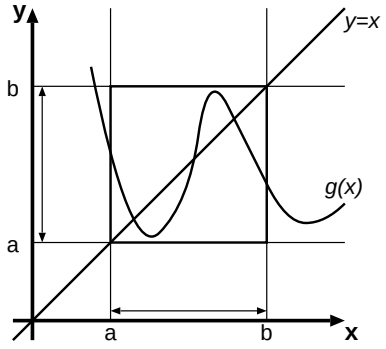



Figure 2.3: In order to get multiple fixed points, we need the slope of $g(x)$ to be greater than 1.

We can determine when a sequence converges from the following corollary to the Contraction Mapping Theorem:

**Corollary 2.1** *Let $g$ be a real-valued function, defined and continuous on a bounded closed interval $[a, b]$ of the real line, and assume that $g(x) \in [a, b]$ for all $x \in [a, b]$. Let $x^* = g(x^*)$ be a fixed point of $g(x)$ with $x^* \in [a, b]$. Assume there exists $\delta$ such that $g'(x)$ is continuous in $I_\delta = [x^* - \delta, x^* + \delta]$. Define the sequence $\{x_i\}_{i=0}^{\infty}$ by $x_{i+1} = g(x_i)$. Then:*

*I. If $|g'(x^*)| < 1$ then there exists $\epsilon$ such that $\{x_i\}$ converges to $x^*$ for $|x_0 - x^*| < \epsilon$. Further, convergence is linear with $\lim_{i\to\infty} c_i = |g'(x^*)|$.*

*II. If $|g'(x^*)| > 1$ then $\{x_i\}$ diverges for any starting value $x_0$.*

Using this corollary, we can come up with a method of choosing our form for $g(x)$ in terms of $f(x)$ depending on the derivative at the point $x^*$.

**Example 2.3** *Suppose that we somehow know that $f'(x^*) = 3/2$ where we wish to solve for the root using $f(x) = 0$. Then if we add and subtract $x$ from the equivalent equation*

*$-f(x) = 0$ we get that $x - x - f(x) = 0$. We define $g(x) = x + f(x)$ so we can apply fixed point iteration on $g(x)$ to solve $x = g(x)$. Using this definition of $g(x)$ we get that*

$$|g'(x^*)| = |1 + f'(x^*)| = |1 + 3/2| = 5/2 > 1$$

*and so from the corollary we note that we will not have convergence.*

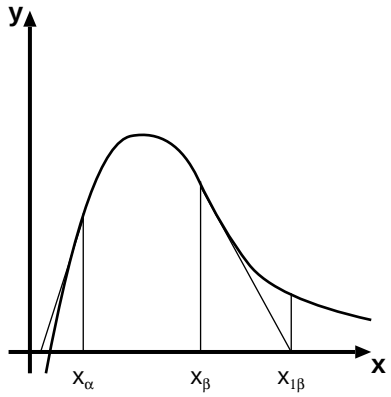
*If we instead add and subtract $x$ from $f(x) = 0$ we get that $x - x + f(x) = 0$. We define $g(x) = x - f(x)$ so we can apply fixed point iteration on $g(x)$ as before. However, with this definition of $g(x)$ we get that*

$$|g'(x^*)| = |1 - f'(x^*)| = |1 - 3/2| = 1/2 < 1$$

*and so from the corollary we can choose some $x_0$ close enough to $x^*$ to get convergence.*

### 2.4.2 Newton's Method

We need to use special care when applying Newton's method. In particular, if $x_0$ is too far away from $x^*$ we may not get convergence. Consider the following example:




If $x_0 = x_\alpha$ then we will achieve convergence. However, if $x_0 = x_\beta$ we will diverge since the function is monotone decreasing for $x \to \infty$ but remains positive.

The convergence theorem for Newton's method is not immediately obvious, but allows us to predict cases where we will achieve convergence:

**Theorem 2.4** ***Convergence Theorem for Newton's Method (Atkinson p.60)*** *If $f(x^*) = 0$, $f'(x^*) \neq 0$ and $f$, $f'$ and $f''$ are all continuous in $I_\delta = [x^* - \delta, x^* + \delta]$ with $x_0$ sufficiently close to $x^*$ then the sequence $\{x_i\}_{i=0}^{\infty}$ converges quadratically to $x^*$ with*

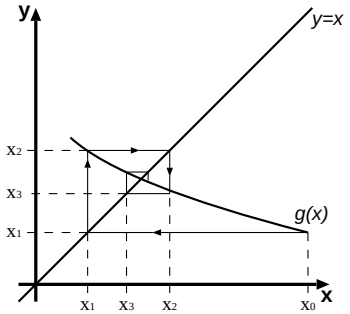
$$\lim_{i \to \infty} c_i = \frac{|f''(x^*)|}{|2f'(x^*)|}. \tag{2.18}$$

Note that in this case, the defining equation for $c_i$ is $|x_{i-1} - x^*| = c_i |x_i - x^*|^2$. We note that if $f'(x^*) = 0$ then the rate of convergence degrades to linear convergence. If the conditions for Newton's method are not met, we may not be able to achieve convergence for any starting value $x_0$.

## Convergence Behaviour of the Fixed Point Method

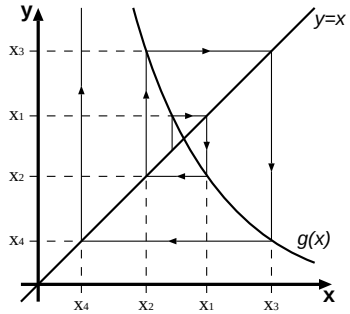
There exist four different types of behaviour when considering convergence in fixed point iteration.

**Spiral convergence**



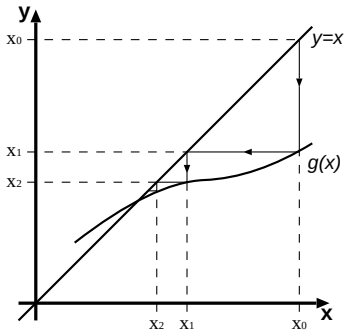

$-1 < g'(x^*) < 0$

**Spiral divergence**



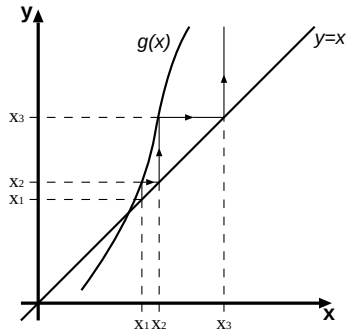

$g'(x^*) < -1$

**Staircase convergence**




$0 < g'(x^*) < 1$

**Staircase divergence**




$g'(x^*) > 1$

### 2.4.3 Secant Method

The secant method has a very similar convergence theorem to that of Newton's method. In fact, all that changes is the order of convergence of the method. Similar to Newton's method, it can be shown that if $f'(x^*) = 0$ then the rate of convergence degrades to linear.

**Theorem 2.5** ***Convergence Theorem for Secant Method (Atkinson p.67)*** *If $f(x^*) = 0$, $f'(x^*) \neq 0$ and $f$, $f'$ and $f''$ are all continuous in $I_\delta = [x^* - \delta, x^* + \delta]$ with $x_0$ sufficiently close to $x^*$ then the sequence $\{x_i\}_{i=0}^{\infty}$ converges with order $q = \frac{1}{2}(1 + \sqrt{5})$.*

### 2.4.4 Overview

Each of the root-finding methods has their own strengths and weaknesses that make them particularly useful for some situations and inapplicable in others. The following two tables provide a summary of the functionality of each root-finding method:

| Method | Does the method converge? |
|---|---|
| Bisection | yes, guaranteed |
| Fixed-point | not always; depending on $g(x)$ and $x_0$ |
| Newton | not always; depending on $f(x)$ and $x_0$ |
| Secant | not always; depending on $f(x)$, $x_0$ and $x_1$ |

| Method | Speed of convergence | Require knowledge of $f'$ |
|---|---|---|
| Bisection | slow (linear) | no |
| Fixed-point | slow (linear) | no |
| Newton | fast (quadratic) | yes |
| Secant | fast ($q \approx \frac{1}{2}(1 + \sqrt{5})$) | no |

In practice, MATLAB uses a combination of the Secant method, Bisection method and a method not discussed here called Inverse Quadratic Interpolation (the combined method is accessible through the `fzero` command). It uses the method that converges fastest if possible, defaulting to the guaranteed convergence of Bisection if necessary. Further, it requires no knowledge of the derivative. This approach allows MATLAB's general root-finding function `fzero` to be well-suited to a variety of applications.

**These notes have been funded by...**

# Chapter 3

# Numerical Linear Algebra

## 3.1 Introduction

Consider the linear system

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & \cdots & + & a_{2n}x_n & = & b_2 \\ \vdots & & \vdots & & \ddots & & \vdots & & \vdots \\ a_{n1}x_1 & + & a_{n2}x_2 & + & \cdots & + & a_{nn}x_n & = & b_n \end{array} \tag{3.1}$$

with coefficients $a_{ij}$ and unknowns $x_i$. We may rewrite this as a matrix system $A\vec{x} = \vec{b}$ where $A \in \mathbb{R}^{n\times n}$ and $\vec{x}, \vec{b} \in \mathbb{R}^n$ as follows:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}. \tag{3.2}$$

We consider the problem of solving for $\vec{x}$ computationally, where we desire

- **accuracy** (we must make sure that this is a well-conditioned problem and that we have a stable algorithm)
- **efficiency** (we wish to solve potentially large systems with limited resources)

This problem has several well-known applications. For example, the Google search engine uses linear systems in order to rank the results it retrieves for each keyword search. Here we see that efficiency is very important: the linear systems they use may contain more than three billion equations!

We may also ask if the problem of solving a given linear system is well posed, *i.e.*, can we find a single unique solution $\vec{x}$ that satisfies the linear system? We may appeal to a standard result from linear algebra to answer this question:

**Theorem 3.1** *Existence and Uniqueness Consider* $A\vec{x} = \vec{b}$.

***Case 1:*** $\det(A) \neq 0$ *(A has linearly independent rows/columns, or A is invertible) if and only if* $\vec{x} = A^{-1}\vec{b}$ *is the unique solution of* $A\vec{x} = \vec{b}$.
***Case 2:*** $\det(A) = 0$ *(recall that* $\text{range}(A) =$ *column space of A), then*
***Case 2a:*** *If* $\vec{b} \in \text{range}(A)$ *then* $A\vec{x} = \vec{b}$ *has infinitely many solutions.*
***Case 2b:*** *If* $\vec{b} \notin \text{range}(A)$ *then* $A\vec{x} = \vec{b}$ *has no solutions.*

## 3.2 Gaussian Elimination

Gaussian elimination was originally described by Carl Friedrich Gauss in 1809 in his work on celestial mechanics entitled *Theoria Motus Corporum Coelestium*. However, elimination in the form presented here was already in use well beforehand – in fact, it was even known in China in the first century B.C.

### 3.2.1 LU Factorization

Before continuing, we first consider the definition of a triangular matrix. Linear systems involving triangular matrices are very easy to solve and appear when performing Gaussian elimination.

**Definition 3.1** *A matrix* $A \in \mathbb{R}^{n\times n}$ *with components* $a_{ij}$ *is said to be* ***upper-triangular*** *if* $a_{ij} = 0$ *for all* $i > j$. *Similarly, A is said to be* ***lower-triangular*** *if* $a_{ij} = 0$ *for all* $i < j$. *A is* ***triangular*** *if it is either upper-triangular or lower-triangular.*

Gaussian elimination may be performed in two phases:

- **Phase 1:** Reduce the matrix $A$ to upper triangular form.
- **Phase 2:** Solve the reduced system by backward substitution.

We illustrate Gaussian elimination of a linear system with an example:

**Example 3.1** *Consider the system* $A\vec{x} = \vec{b}$ *with*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix}.$$

In the first step we choose the pivot element $a_{11}^{(1)} = 1$ and use it to compute $A^{(2)}$.

$$\textbf{Step i} = \textbf{1}: \quad A^{(1)} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix}$$
$$\textbf{Step i} = \textbf{2}: \quad A^{(2)} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -20 \end{pmatrix}$$

In this case $A^{(2)}$ is obtained from $A^{(1)}$ by taking linear combinations of the first row of $A^{(1)}$ with each of the other rows so as to generate zeroes in the first column. This operation may also be represented by matrix multiplication on the left with the matrix $M_1$:

$$\begin{array}{ccc} M_1 & A^{(1)} & = & A^{(2)} \\ \begin{pmatrix} 1 & 0 & 0 \\ -\frac{4}{1} & 1 & 0 \\ -\frac{7}{1} & 0 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix} & = & \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -20 \end{pmatrix}. \end{array}$$

In general, we may write

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{a_{21}^{(1)}}{a_{11}^{(1)}} & 1 & 0 \\ -\frac{a_{31}^{(1)}}{a_{11}^{(1)}} & 0 & 1 \end{pmatrix}, \qquad A^{(2)} = \begin{pmatrix} a_{11}^{(2)} & a_{12}^{(2)} & a_{13}^{(2)} \\ a_{21}^{(2)} & a_{22}^{(2)} & a_{23}^{(2)} \\ a_{31}^{(2)} & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix}.$$

We now choose $a_{22}^{(2)}$ as the pivot element. We may then compute the matrix $A^{(3)}$ using $M_2 \cdot A^{(2)} = A^{(3)}$:

$$\textbf{Step i} = \textbf{3} : M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -\frac{(-6)}{-3} & 1 \end{pmatrix}, \qquad A^{(3)} = \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -8 \end{pmatrix},$$

where we obtain $A^{(3)}$ by

$$\begin{array}{ccc} M_2 & A^{(2)} & = & A^{(3)} \\ \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} & \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & -6 & -20 \end{pmatrix} & = & \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -8 \end{pmatrix}. \end{array}$$

We note that $A^{(3)}$ is in *upper triangular* form, as desired.

**Example 3.1 Recap:**

$$\begin{array}{rclrcl} A & = & \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix} & U & = & \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 8 \end{pmatrix} \\ M_1 & = & \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{pmatrix} & M_2 & = & \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix} \end{array}$$

We may write

$$\begin{aligned} M_2 \cdot (M_1 \cdot A) &= U \\ (M_2 \cdot M_1) \cdot A &= U \\ A &= (M_2 \cdot M_1)^{-1} U \\ A &= M_1^{-1} M_2^{-1} U \end{aligned}$$

We will define a matrix $L$ by $M_1^{-1}M_2^{-1} = L$ and so may write the matrix $A$ as the product $A = LU$. We now wish to consider the properties of the matrices $M_i$ and $L$. Consider the matrix $M_1$, introduced above, and its inverse:

$$M_1 = \begin{pmatrix} 1 & 0 & 0 \\ -4 & 1 & 0 \\ -7 & 0 & 1 \end{pmatrix} \quad M_1^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 0 & 1 \end{pmatrix}.$$

We define the matrix $L_i$ as the inverse of the matrix $M_i$ (so $L_iM_i = I$). The following inversion property then follows from the structure of the matrix $M_i$:

**Inversion Property:** $L_i$ can be obtained from $M_i$ by swapping the signs of the off-diagonal elements.

**Example 3.2** *Consider the matrix $M_2$, introduced above. A simple calculation allows us to obtain the following result:*

$$L_2 = M_2^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 2 & 1 \end{pmatrix},$$

*which satisfies the inversion property.*

In addition, the structure of $L$ can be directly determined from the matrices $L_i$ using the combination property:

**Combination Property:** In general, $L = \prod_{i=1}^{n-1} L_i = L_1 \cdot L_2 \cdot \cdots \cdot L_{n-1}$. $L$ can be obtained from the $L_i$ by placing all of the off-diagonal elements of the matrices $L_i$ in the corresponding position in $L$.

We note that the matrix $L$ is a special type of lower-triangular matrix, defined as follows:

**Definition 3.2** *$L$ is called a **lower triangular matrix with unit diagonal** if and only if the matrix elements of $L$ vanish above the diagonal and are 1 on the diagonal (i.e. $\ell_{ij} = 0 \ \forall \ j > i$ and $\ell_{ii} = 1 \ \forall \ i$).*

Using these two properties, we may write the LU decomposition of $A$ as follows:

$$\begin{aligned} M_2M_1A &= U \\ A &= M_1^{-1}M_2^{-1}U \\ A &= LU, \end{aligned}$$

with $L$ unit lower triangular and $U$ upper triangular. For example 3.1, we write

$$\begin{matrix} A \\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{pmatrix} \end{matrix} = \begin{matrix} L \\ \begin{pmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{pmatrix} \end{matrix} \begin{matrix} U \\ \begin{pmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & -8 \end{pmatrix} \end{matrix}.$$

The technique discussed here may be generalized to square matrices of any size and is more generally known as LU decomposition.

**Procedure: LU Decomposition.** For any $A \in \mathbb{R}^{n\times n}$ we may compute the LU decomposition in $n$ steps as follows:

$$\begin{aligned} A^{(1)} &= A \\ A^{(2)} &= M_1 \cdot A^{(1)} \\ A^{(3)} &= M_2 \cdot A^{(2)} = M_2 M_1 A^{(1)} \\ &\vdots \\ A^{(n)} &= M_{n-1} \cdot A^{(n-1)} \end{aligned}$$

We obtain $M_i$ from

$$M_j = \begin{pmatrix} 1 & & & & 0 \\ & \ddots & & & \\ & & 1 & & \\ & & \boxed{c_{ij}} & \ddots & \\ 0 & & & & 1 \end{pmatrix}, \qquad c_{ij} = -\frac{a_{ij}^{(j)}}{a_{jj}^{(j)}}.$$

After computing the product of all $M_i$, we have

$$\underbrace{M_{n-1} \cdot M_{n-2} \cdots M_2 \cdot M_1}\cdot A = U,$$

which may be inverted to obtain

$$A = \underbrace{M_1^{-1} \cdot M_2^{-1} \cdots M_{n-2}^{-1} \cdot M_{n-1}^{-1}}\cdot U.$$

Since we defined $L_j = M_j^{-1}$, we also have

$$A = \underbrace{L_1 \cdot L_2 \cdots L_{n-2} \cdot L_{n-1}}\cdot U$$

and so obtain the LU decomposition of $A$,

$$A = L \cdot U.$$

Since $L$ and $U$ are triangular matrices, we may easily compute the solution to a linear system with either matrix $L$ or $U$. The decomposition then leads to the final step in our method of solving a general linear system:

**Procedure: Solving a Linear System by LU Decomposition.** Consider a linear system given by $A\vec{x} = \vec{b}$. Since we now have a procedure for computing the LU decomposition of a matrix $A$, we may write an algorithm for performing Gaussian Elimination computationally.

- **Phase 1:** Decompose $A = LU$ so we may write the linear system as $LU\vec{x} = \vec{b}$.
- **Phase 2:** Solve $L\vec{y} = \vec{b}$ for $\vec{y}$ by forward substitution.
- **Phase 3:** Solve $U\vec{x} = \vec{y}$ for $\vec{x}$ by backward substitution.

**Note:** Retaining $L$ and $U$ is advantageous when systems have to be solved with the same $A$ and multiple right-hand-side vectors $\vec{b}$.

### 3.2.2 Pivoting

We observe that the LU decomposition algorithm breaks down when at some step $i$ the pivot element $a_{ii}$ is equal to zero. However, this problem does not necessarily imply that the system is unsolvable.

**Example 3.3** *Consider the linear system $A\vec{x} = \vec{b}$ defined by*

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

We note that the pivot element in this example is zero in the first step, making it impossible to proceed using the LU decomposition algorithm described previously. However, we will have no problem proceeding if we swap the first and second rows before applying the LU decomposition.

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

In this case, swapping rows reduces the system to upper-triangular form, and so we may solve it very easily. By inspection, this system has the solution $\vec{x} = (1\ 1)^T$.

More formally, the operation of swapping rows can be written as multiplication on the left with a permutation matrix $P$. For example, in the previous example we may write

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

$$\implies \quad PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix}.$$

Then we have $A\vec{x} = \vec{b} \iff PA\vec{x} = P\vec{b}$.

**Definition 3.3** *$P \in \mathbb{R}^{n \times n}$ is a **permutation matrix** if and only if $P$ is obtained from the unit matrix $I_n$ by swapping any number of rows.*

**Example 3.4** *An example of a $4 \times 4$ permutation matrix is*

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

We may also (equivalently) define a permutation matrix $P$ as an $n \times n$ matrix with elements 1 and 0 such that every row and column has exactly one 1. In general, we find that the addition of the permutation matrix step is all that is needed to make the LU decomposition algorithm applicable to any square matrix:

**Theorem 3.2** *For all $A \in \mathbb{R}^{n\times n}$ there exists a permutation matrix $P$, a unit lower triangular matrix $L$ and an upper triangular matrix $U$ (all of type $\mathbb{R}^{n\times n}$) such that $PA = LU$.*

**Corollary 3.1** *If $A$ is nonsingular then $A\vec{x} = \vec{b}$ can be solved by the LU decomposition algorithm applied to $PA$.*

**Proof.** We write our linear system as $A\vec{x} = \vec{b}$ and multiply both sides by the permutation matrix $P$ (from Theorem 3.2) to obtain $PA\vec{x} = P\vec{b}$. From Corollary 3.1, we have that $PA = LU$ and so may write $LU\vec{x} = P\vec{b}$. Thus we may simply apply forward and backward substitution and so solve this system by LU decomposition. The substitution steps will not lead to divisions by zero as $L$ and $U$ do not have any vanishing diagonal elements. This follows because $\det(A) = \det(U)\det(L)/\det(P) \neq 0$, which means that $\prod_{i=1}^{n} u_{ii} \neq 0$ as $\det(P) = \pm 1$ and $\det(L) = 1$ (see Section 3.2.4.) □

### 3.2.3 Algorithm and Computational Cost

We now consider the computational implementation of Gaussian elimination using the LU decomposition. Recall that Gaussian elimination can be performed in two phases. We will assume that pivoting is not needed.

**Phase 1:** Compute the LU decomposition, $A = LU$. The pseudocode for this algorithm is as follows:

**LU-Decomposition**

```
L = diag(1)
U = A
for p = 1:n-1
  for r = p+1:n
    m = -u(r,p)/u(p,p)
    u(r,p) = 0
    for c = p+1:n
      u(r,c) = u(r,c) + m u(p,c)
    end for
    l(r,p) = -m
  end for
end for
```

Here, the variable `p` represents the row of the pivot element, `r` is the current row and `c` is the current column.

**Aside: An efficient storage method.**
We can save memory by implementing this algorithm in a clever way. Recall that the LU decomposition of $A$ is given by

$$A = LU = \begin{pmatrix} 1 & & & 0 \\ \times & \ddots & & \\ \times & \times & \ddots & \\ \times & \times & \times & 1 \end{pmatrix} \begin{pmatrix} \times & \times & \times & \times \\ \ddots & \times & \times & \times \\ & \ddots & \times & \times \\ 0 & & \ddots & \times \end{pmatrix}.$$

We may store $L$ and $U$ together as a single matrix since we know that the diagonal components of $L$ are all equal to 1. Thus the diagonal and upper-triangular portion of the new combined matrix will consist of the elements of $U$ and the lower-triangular portion will consist of the elements of $L$.

With this storage mechanism, we can also perform LU decomposition in place; *i.e.* perform all computations directly on top of the input matrix $A$. However, using this technique we lose the original contents of the matrix $A$.

**Computational Cost of Phase 1**
In order to compute the computational cost of this algorithm, we consider two counters: Let $M$ be the number of multiplications or divisions and let $A$ be the number of additions or subtractions. The following summation identities will come in handy in performing our analysis:

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

- **Count A:** We sum over all loops in the algorithm to yield

$$\begin{aligned} W &= \sum_{p=1}^{n-1} \sum_{r=p+1}^{n} \sum_{c=p+1}^{n} A \\ &= \sum_{p=1}^{n-1} (n-p)^2 A \\ &= \sum_{p=1}^{n-1} (n^2 - 2np + p^2) A \\ &= \left[ \frac{1}{6} \cdot n \cdot (n-1) \cdot (2n-1) - \frac{1}{2} \cdot 2n \cdot n \cdot (n-1) + n^2 \cdot (n-1) \right] A \\ &= \left[ (\frac{2}{6} - 1 + 1) n^3 + O(n^2) \right] A \\ &= \left[ \frac{1}{3} n^3 + O(n^2) \right] A. \end{aligned}$$

- **Count M:** Similarly we determine

$$W = \left[ \frac{1}{3} n^3 + O(n^2) \right] M.$$

  (close inspection reveals that the additional multiplication only affects the $O(n^2)$ terms.)

Thus the total number of floating point operations is

$$W = \frac{2}{3} n^3 + O(n^2) \text{ flops.} \tag{3.3}$$

(on modern CPUs, type A operations and type M operations take approximately the same amount of time.)

**Phase 2:** We solve $L\vec{y} = \vec{b}$ by forward substitution.

$$\begin{pmatrix} 1 & & 0 \\ & \ddots & \\ \boxed{L(i,j)} & & 1 \end{pmatrix} \begin{pmatrix} \\ \vec{y} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \vec{b} \\ \\ \end{pmatrix}.$$

The $i^{th}$ equation in this system is given by

$$\sum_{k=1}^{i} L(i,k) y(k) = b(i).$$

We rewrite this as

$$y(i) = b(i) - \sum_{k=1}^{i-1} L(i,k)y(k).$$

Thus we may formulate an algorithm for forward substitution by solving for each $y(i)$ in turn:

**Forward Substitution**

```
y = b
for r = 2:n
  for c = 1:r-1
    y(r) = y(r) - L(r,c) * y(c)
  end for
end for
```

Here, `r` is the current row and `c` is the current column. We may determine the computational cost of the algorithm, as with LU decomposition:

$$\begin{aligned} W &= \sum_{r=2}^{n}\sum_{c=1}^{r-1}(1M + 1A) \\ &= \sum_{r=2}^{n}(r-1)(1M + 1A) \\ &= \sum_{s=1}^{n-1} s(1M + 1A) \\ &= \frac{1}{2}n(n-1)(M + A) \\ &= n(n-1) \text{ flops} \end{aligned}$$

Thus, the total number of floating point operations required for Forward Substitution is

$$W = n^2 + O(n) \text{ flops.} \tag{3.4}$$

**Phase 3:** We solve $U\vec{x} = \vec{y}$ by backward substitution.

$$\begin{pmatrix} U(1,1) & & \boxed{U(i,j)} \\ & U(2,2) & \\ & & \ddots \\ & & U(n,n) \end{pmatrix} \begin{pmatrix} \\ \vec{x} \\ \\ \end{pmatrix} = \begin{pmatrix} \\ \vec{y} \\ \\ \end{pmatrix}.$$

The $i^{th}$ equation in this system is given by

$$\sum_{k=1}^{i} u(i,k)x(k) = y(i).$$

We rewrite this as

$$x(i) = \left[ y(i) - \sum_{k=i+1}^{n} u(i,k)x(k) \right] \frac{1}{u(i,i)}.$$

Thus we may formulate an algorithm for backward substitution by solving for each $x(i)$ in turn:

**Backward Substitution**

```
x = y
for r = n:-1:1
  for c = r+1:n
    x(r) = x(r) - U(r,c) * x(c)
  end for
  x(r) = x(r) / U(r,c)
end for
```

Here, `r` is the current row and `c` is the current column. The computational complexity will be the same as with forward substitution:

$$W = n^2 + O(n) \text{ flops.} \tag{3.5}$$

**Note:** If $u(i,i) = 0$ for some $i$, the backward substitution algorithm breaks down, but this can never happen if $\det(A) \neq 0$.

### 3.2.4 Determinants

Before continuing, we consider some of the properties of the determinant.

**Definition 3.4** *The **determinant** of a matrix $A \in \mathbb{R}^{n \times n}$ is given by*

$$\det(A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij} \det(A_{ij}), \qquad \textit{for fixed } i, \tag{3.6}$$

*with*

$$A_{ij} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{(1)(j-1)} & a_{(1)(j+1)} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{(2)(j-1)} & a_{(2)(j+1)} & \cdots & a_{2n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{(i-1)(1)} & a_{(i-1)(2)} & \cdots & a_{(i-1)(j-1)} & a_{(i-1)(j+1)} & \cdots & a_{(i-1)(n)} \\ a_{(i+1)(1)} & a_{(i+1)(2)} & \cdots & a_{(i+1)(j-1)} & a_{(i+1)(j+1)} & \cdots & a_{(i+1)(n)} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{(n)(j-1)} & a_{(n)(j+1)} & \cdots & a_{nn} \end{pmatrix}.$$

*i.e. the matrix $A_{ij}$ is an $(n-1)\times(n-1)$ matrix obtained by removing row $i$ and column $j$ from the original matrix $A$.*

This is the expansion of the determinant about row $i$ for any $1 \le i \le n$. We may also consider the expansion of the determinant about column $j$ for any $1 \le j \le n$ as follows:

$$\det(A) = \sum_{i=1}^{n} (-1)^{i+j} a_{ij} \det(A_{ij}), \qquad \text{for fixed } j. \tag{3.7}$$

**Example 3.5** *We compute the determinant of the matrix used in example 3.1 using an expansion of the first row:*

$$\det \begin{vmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 1 \end{vmatrix} = 1 \cdot \det \begin{vmatrix} 5 & 6 \\ 8 & 1 \end{vmatrix} - 2 \cdot \det \begin{vmatrix} 4 & 6 \\ 7 & 1 \end{vmatrix} + 3 \cdot \det \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} = 24.$$

The determinant satisfies several useful properties, which we may formulate in the following proposition:

**Proposition 3.1** *The following identities hold for determinants:*

1. $\det(BC) = \det(B) \cdot \det(C)$, *($B, C \in \mathbb{R}^{n\times n}$)*
2. $U \in \mathbb{R}^{n\times n}$ *upper triangular* $\Rightarrow \det(U) = \prod_{i=1}^{n} u_{ii}$
3. $L \in \mathbb{R}^{n\times n}$ *lower triangular* $\Rightarrow \det(L) = \prod_{i=1}^{n} \ell_{ii}$
4. $P \in \mathbb{R}^{n\times n}$ *permutation matrix*

$$\Rightarrow \det(P) = \begin{cases} +1 & \text{even number of row changes to obtain } P \text{ from } I_n. \\ -1 & \text{odd number of row changes to obtain } P \text{ from } I_n. \end{cases}$$

**Proof of 2.** Consider the following upper-triangular matrix $U$:

$$U = \left(\begin{array}{c|c|ccc} u_{11} & \times & \times & \cdots & \times \\ \hline 0 & u_{22} & \times & \cdots & \times \\ \hline & 0 & u_{33} & \cdots & \times \\ & & & \ddots & \vdots \\ & & & & u_{nn} \end{array}\right).$$

We expand on the first column to yield

$$\begin{aligned} \det(U) &= u_{11}\det(U^{(2)}) + \overbrace{u_{21}\det(\cdots)}^{0} + \cdots \\ &= u_{11}u_{22}\det(U^{(3)}) + \overbrace{u_{32}\det(\cdots)}^{0} + \cdots \\ &= \cdots \\ &= \prod_{i=1}^{n} u_{ii}. \qquad \square \end{aligned}$$

The proof of 1, 3, and 4 are similar and are left as an exercise for the reader.

Recall that we may solve the linear system $A\vec{x} = \vec{b}$ using Gaussian elimination as follows:

- **Phase 1:** $PA = LU$
- **Phase 2:** $L\vec{y} = P\vec{b}$
- **Phase 3:** $U\vec{x} = \vec{y}$

However, recall that the algorithm for the LU decomposition (phase 1) can only be performed if there are no divisions by zero. How can we guarantee that this will not occur?

**Proposition 3.2** *Consider a matrix $A \in \mathbb{R}^{n\times n}$. Then $\det(A) \neq 0$ if and only if the decomposition $PA = LU$ has $u_{ii} \neq 0\ \forall\ i$.*

**Proof.**

$$\begin{aligned} PA &= LU \\ \det(PA) &= \det(LU) \\ \underbrace{\det(P)}_{\pm 1}\det(A) &= \underbrace{\det(L)}_{1}\underbrace{\det(U)}_{\prod_{i=0}^{n} u_{ii}} \\ \pm\det(A) &= \prod_{i=0}^{n} u_{ii}. \end{aligned}$$

Thus we have $\det(A) \neq 0 \Leftrightarrow u_{ii} \neq 0\ \forall\ i$. $\square$

## Intermission: Cramer's Rule

Consider a set of linear equations in matrix form $A\vec{x} = \vec{b}$ as in equation (3.2). The determinant $\det(A)$ is given by

$$\det(A) = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix}. \tag{3.8}$$

If we multiply $\det(A)$ by $x_1$ and apply a property of determinants, we may take the $x_1$ inside the determinant along one of the columns.

$$x_1 \det(A) = x_1 \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} x_1 a_{11} & a_{12} & a_{13} \\ x_1 a_{21} & a_{22} & a_{23} \\ x_1 a_{31} & a_{32} & a_{33} \end{vmatrix}. \tag{3.9}$$

By another property of determinants, we may add to a column a linear combination of the other columns without changing the determinant. We write

$$x_1 \det(A) = \begin{vmatrix} x_1 a_{11} + x_2 a_{12} + x_3 a_{13} & a_{12} & a_{13} \\ x_1 a_{21} + x_2 a_{22} + x_3 a_{23} & a_{22} & a_{23} \\ x_1 a_{31} + x_2 a_{32} + x_3 a_{33} & a_{32} & a_{33} \end{vmatrix} = \begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}. \tag{3.10}$$

We define $D_i$ as the determinant of $A$ with the $i^{\text{th}}$ column replaced with $\vec{b}$. If $\det(A) \neq 0$, it follows from (3.10) that $x_i = D_i / \det(A)$. So, for our simple linear system:

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} & a_{13} \\ b_2 & a_{22} & a_{23} \\ b_3 & a_{32} & a_{33} \end{vmatrix}}{D} \qquad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 & a_{13} \\ a_{21} & b_2 & a_{23} \\ a_{31} & b_3 & a_{33} \end{vmatrix}}{D} \qquad x_3 = \frac{\begin{vmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ a_{31} & a_{32} & b_3 \end{vmatrix}}{D}$$

This procedure is known as Cramer's Rule and can be generalized to a set of $n$ equations. Using this method, we can solve our linear system by calculating the determinants of $n+1$ matrices.

Consider the general case of $A \in \mathbb{R}^{n \times n}$. In order to apply Cramer's Rule we must compute $n+1$ determinants, each of a matrix of size $n \times n$. If we use the recursive method to compute each of these determinants, for each of the $n+1$ determinants, we must compute $n$ determinants of a matrix of size $(n-1) \times (n-1)$. A short calculation reveals that we need to compute $(n+1)!$ determinants in the general case. This complexity is much higher than for any polynomial-time algorithm; in fact, it is even much worse than even an exponential time algorithm! Therefore, calculation of the determinant as in the proof of Proposition 3.2, which requires $O(n^3)$ operations, is a much better idea.

## 3.3 Condition and Stability

### 3.3.1 The Matrix Norm

The matrix norm and the vector norm are both tools designed to allow us to measure the size of either a matrix or a vector. We consider a particular set of matrix norms, namely those induced by a vector norm. These are also known as the set of "natural" matrix norms over the vector space of matrices (for matrices of the form $A \in \mathbb{R}^{n\times n}$, the set $V = \mathbb{R}^{n\times n}$ is a vector space over $\mathbb{R}$.)

**Definition 3.5** *The **natural matrix p-norm** for $p = 1,\ 2$ or $\infty$ is defined by*

$$\|A\|_p = \max_{\|\vec{x}\|\neq 0} \frac{\|A\vec{x}\|_p}{\|\vec{x}\|_p}. \tag{3.11}$$

An inequality then follows from this definition by rearranging the defining equation (3.11).

**Proposition 3.3** $\|A\vec{x}\|_p \leq \|A\|_p\|\vec{x}\|_p$.

We note that we also have the following properties associated with the matrix norm:

**Proposition 3.4** *Consider a matrix $A \in \mathbb{R}^{n\times n}$ with elements $a_{ij}$. Then*

1. $\|A\|_1 = \max_{1\leq j\leq n} \sum_{i=1}^{n} |a_{ij}|$ *(maximum absolute column sum)*
2. $\|A\|_\infty = \max_{1\leq i\leq n} \sum_{j=1}^{n} |a_{ij}|$ *(maximum absolute row sum)*
3. $\|A\|_2 = \max_{1\leq i\leq n} \lambda_i^{1/2}$ *where $\lambda_i$ are the eigenvalues of $A^TA$. We note that $\lambda_i = \sigma_i^2$, with $\sigma_i$ the singular values of $A$.*

We also note that the natural matrix norms satisfy the triangle inequality:

**Proposition 3.5** *Consider matrices $A, B \in \mathbb{R}^{n\times n}$ with $p = 1, 2, \infty$. Then*

$$\|A + B\|_p \leq \|A\|_p + \|B\|_p.$$

**Proof.** For any matrix norm $\|\cdot\| = \|\cdot\|_p$ we have

$$\|(A + B)\vec{x}\| = \|A\vec{x} + B\vec{x}\| \leq \|A\vec{x}\| + \|B\vec{x}\|,$$

from the vector triangle inequality. Then, by proposition 3.3,

$$\|(A + B)\vec{x}\| \leq \|A\|\|\vec{x}\| + \|B\|\|\vec{x}\|,$$

and so

$$\frac{\|(A + B)\vec{x}\|}{\|\vec{x}\|} \leq \|A\| + \|B\|.$$

Thus, by the definition of the matrix norm we obtain

$$\|A + B\| \leq \|A\| + \|B\|. \qquad \square$$

We can now show that the natural matrix norm satisfies the defining properties of a norm:

**Proposition 3.6** *$\|A\|_p$ is a norm:*

1. $\|A\|_p \geq 0,\ \|A\|_p = 0 \Leftrightarrow A = 0.$
2. $\|\alpha A\|_p = |\alpha| \|A\|_p.$
3. $\|A + B\|_p \leq \|A\|_p + \|B\|_p.$

Since the natural matrix p-norm satisfies the properties of a norm, it can be used as a mechanism for measuring the "size" of a matrix. This will be useful later when considering condition and stability.

### 3.3.2 Condition of the problem $A\vec{x} = \vec{b}$

The linear system problem can be reformulated as follows:

**Problem** Find $\vec{x}$ from $A\vec{x} = \vec{b}$ (*i.e.*, $\vec{x} = A^{-1}\vec{b}$).

From this statement of the problem, we may write $\vec{x} = \vec{f}(A, \vec{b})$. If we want to consider the condition of this problem, there are then two dependent variables which can be perturbed and with contribute to the condition number. We want to consider the change $\Delta\vec{x}$ if we have inputs $A + \Delta A$ and $\vec{b} + \Delta\vec{b}$.

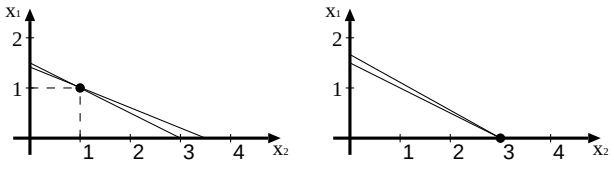
**Example 3.6** *Consider the linear system $A\vec{x} = \vec{b}$ and solution $\vec{x}$ given by*

$$\begin{pmatrix} 1 & 2 \\ 0.499 & 1.001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1.5 \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

*We perturb $A$ by a small matrix $\Delta A$ to yield a new linear system and solution given by*

$$\begin{pmatrix} 1 & 2 \\ 0.500 & 1.001 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 3 \\ 1.5 \end{pmatrix}, \qquad \vec{x} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}.$$

*Thus a small change in $A$ results in a large change in $\vec{x}$; this seems to imply that the problem is ill conditioned.*




*First equation* *Second equation*

In order to examine the condition of the initial problem **P** ($A\vec{x} = \vec{b}$) we need to consider a slight perturbation on the input data $A$ and $\vec{b}$,

$$(A + \Delta A)(\vec{x} + \Delta\vec{x}) = \vec{b} + \Delta\vec{b}. \tag{3.12}$$

We now look for bounds on $\kappa_R$. We note that it is difficult to derive a bound on $\kappa_R$ without considering a simplification of the defining equation (3.12). As a result, we consider the two most obvious simplified cases:

**Case 1:** Consider the case of no perturbation in $A$, *i.e.* $\Delta A = 0$ with $\Delta\vec{b} \neq 0$. Equation (3.12) becomes

$$A(\vec{x} + \Delta\vec{x}) = \vec{b} + \Delta\vec{b} \underbrace{\Longrightarrow}_{A\vec{x}=\vec{b}} A\Delta\vec{x} = \Delta\vec{b} \Longrightarrow \Delta\vec{x} = A^{-1}\Delta\vec{b}. \tag{3.13}$$

We take the norm of both sides of this expression to yield

$$\|\Delta\vec{x}\| = \|A^{-1}\Delta\vec{b}\| \leq \|A^{-1}\|\|\Delta\vec{b}\|. \tag{3.14}$$

We may also take the norm of both sides of $A\vec{x} = \vec{b}$ to yield

$$\|\vec{b}\| = \|A\vec{x}\| \Rightarrow \|\vec{b}\| \leq \|A\| \cdot \|\vec{x}\| \Rightarrow \|\vec{b}\| \cdot \|A\|^{-1} \leq \|\vec{x}\|. \tag{3.15}$$

From equations (3.14) and (3.15) we have that

$$\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq \|A^{-1}\|\|A\|\frac{\|\Delta\vec{b}\|}{\|\vec{b}\|}.$$

This may be rearranged to give the relative condition number

$$\kappa_R = \frac{\|\Delta\vec{x}\|/\|\vec{x}\|}{\|\Delta\vec{b}\|/\|\vec{b}\|} \leq \|A\|\|A^{-1}\|. \tag{3.16}$$

Thus we know $\|A\| \cdot \|A^{-1}\|$ is an upper bound for $\kappa_R$.

**Definition 3.6** *The condition number of a matrix $A$ is $\kappa(A) = \|A\|\|A^{-1}\|$.*

Thus if $\kappa(A)$ is small, problem **P** is well conditioned. The natural error due to rounding in $\vec{b}$ produces an error of relative magnitude $\frac{\|\Delta\vec{b}\|}{\|\vec{b}\|} \approx \epsilon_{mach}$, so it follows that there will be an error in $\vec{x}$ of relative magnitude $\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq \kappa(A) \cdot \epsilon_{mach}$.

**Case 2:** We now consider the case of no perturbation in $b$, *i.e.* $\Delta A \neq 0$ with $\Delta\vec{b} = 0$. Equation (3.12) becomes

$$(A + \Delta A)(\vec{x} + \Delta\vec{x}) = \vec{b}. \tag{3.17}$$

We expand this equation and subtract $A\vec{x} = \vec{b}$. This allows us to bound $\|\Delta\vec{x}\|$.

$$\begin{aligned} A\Delta\vec{x} &= -\Delta A(\vec{x} + \Delta\vec{x}) \\ \Delta\vec{x} &= -A^{-1}\Delta A(\vec{x} + \Delta\vec{x}) \\ \|\Delta\vec{x}\| &\leq \|A^{-1}\|\|\Delta A\|\|\vec{x} + \Delta\vec{x}\| \\ \|\Delta\vec{x}\|/\|\vec{x} + \Delta\vec{x}\| &\leq \|A^{-1}\|\|\Delta A\|\|A\|\|A\|^{-1}. \end{aligned}$$

We may apply the approximation $\|\Delta\vec{x}\|/\|\vec{x} + \Delta\vec{x}\| \approx \|\Delta\vec{x}\|/\|\vec{x}\|$ for $\Delta x$ small. This gives us an expression for the relative condition number in this case:

$$\kappa_R = \frac{\|\Delta\vec{x}\|/\|\vec{x}\|}{\|\Delta A\|/\|A\|} \leq \|A\|\|A^{-1}\|. \tag{3.18}$$

**Case 3:** In the case of perturbations in $A$ and $b$, *i.e.* $\Delta A \neq 0$ and $\Delta\vec{b} \neq 0$, it can also be shown that $\kappa_R \leq \kappa(A) = \|A\|\|A^{-1}\|$. However, the derivation is tedious and so will not be given here.

From these three cases, it appears that the condition number of a matrix $\kappa(A)$ is all we need to determine the condition number of the problem **P** defined by the linear system $A\vec{x} = \vec{b}$. In particular, we note that the 2-condition number of a matrix (defined by using the 2-norm) has a useful property that makes it unnecessary to compute the inverse $A^{-1}$:

**Proposition 3.7** *For a matrix $A \in \mathbb{R}^{n\times n}$,*

$$\kappa_2(A) = \|A\|_2\|A^{-1}\|_2 = \sqrt{\frac{\lambda_{max}(A^TA)}{\lambda_{min}(A^TA)}} = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}. \tag{3.19}$$

**Proof.** We know that $\|A\|_2 = \max_{1\leq i\leq n}(\lambda_i(A^TA))^{1/2}$, where $\lambda_i(A^TA) > 0\ \forall\ i$ and $\lambda_i$ are eigenvalues of $A^TA$. Also,

$$\begin{aligned} B\vec{x}_i &= \lambda_i\vec{x}_i \qquad (\text{if } \det(B) \neq 0) \\ \Rightarrow \vec{x}_i &= \lambda_i B^{-1}\vec{x}_i \\ \Rightarrow \lambda_i^{-1}\vec{x}_i &= B^{-1}\vec{x}_i, \end{aligned}$$

and so $\lambda_i^{-1}$ are eigenvalues of $B^{-1} = (A^TA)^{-1}$. Using this property with $B = (A^TA)^{-1}$, we obtain

$$\begin{aligned} \|A^{-1}\|_2 &= \max_{1\leq i\leq n}(\lambda_i((A^{-1})^TA^{-1}))^{1/2} \\ &= \max_{1\leq i\leq n}(\lambda_i((AA^T)^{-1}))^{1/2} \\ &= (\min_{1\leq i\leq n}(\lambda_i(AA^T))^{1/2})^{-1}. \end{aligned}$$

In addition, if $\lambda_i$ are eigenvalues of $AA^T$ then $\lambda_i$ are also eigenvalues of $A^TA$, since

$$\begin{aligned} AA^T\vec{x} &= \lambda_i\vec{x} \\ A^TAA^T\vec{x} &= \lambda_i A^T\vec{x} \\ A^TA\vec{y} &= \lambda_i\vec{y}. \end{aligned}$$

Thus the result follows from the definition of the condition number of a matrix. □

**Example 3.7** *Compute the condition number of*

$$A = \begin{pmatrix} 1 & 2 \\ 0.5 & 1.001 \end{pmatrix}.$$

The eigenvalues of $A^TA$ may be calculated as $\lambda_1 \approx 1.6 \times 10^{-7}$ and $\lambda_2 \approx 6.25$. From (3.19) we get $\kappa_2(A) = 6.25 \times 10^3$, which is a large value. We conclude the problem $A\vec{x} = \vec{b}$ of example 3.6 is ill-conditioned (or the matrix A is ill-conditioned).

### 3.3.3 Stability of the LU Decomposition Algorithm

We wish to consider a related problem, namely:

**Problem** Consider the mathematical problem defined by $z(x) = \frac{a}{x}$ with $a$ constant.

We now wish to know the absolute and relative condition number of this problem. The absolute condition number is computed from

$$\kappa_A = \left|\frac{\Delta z}{\Delta x}\right| \approx \left|\frac{dz(x)}{dx}\right| = \left|-\frac{a}{x^2}\right| = \frac{|a|}{x^2}.$$

Thus, if $x$ is small then this problem is ill-conditioned with respect to the absolute error. The relative condition number is computed from

$$\kappa_R = \frac{|\Delta z|/|z|}{|\Delta x|/|x|} \approx \left|\frac{dz(x)}{dx}\right| \left|\frac{x}{z}\right| = \left|-\frac{a}{x^2}\right| \left|\frac{x^2}{a}\right| = 1,$$

so the problem is well-conditioned with respect to the relative error. These calculations indicate that dividing by a small number (or multiplying by a large number) is ill-conditioned; this is a bad step in any algorithm because the absolute error may increase by a lot.

We conclude that the problem of linear systems will be ill-conditioned if we have many divisions by small numbers. Consider the following example, with $\delta$ small.

**Example 3.8** *Solve the linear system*

$$\begin{pmatrix} \delta & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

Applying Gaussian elimination yields

$$\begin{pmatrix} \delta & 1 \\ 0 & 1-\frac{1}{\delta} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2-\frac{1}{\delta} \end{pmatrix}.$$

We solve for $x_1$ and $x_2$ to obtain

$$x_2 = \frac{2-\frac{1}{\delta}}{1-\frac{1}{\delta}} = \frac{\frac{1}{\delta}(2\delta-1)}{\frac{1}{\delta}(\delta-1)} = \frac{2\delta-1}{\delta-1} \approx 1$$

$$x_1 = \frac{1}{\delta}(1-x_2) = \frac{1}{\delta}\left(1-\frac{2\delta-1}{\delta-1}\right) = \frac{1}{\delta}\left(\frac{\delta-1-2\delta+1}{\delta-1}\right) = \frac{1}{\delta}\left(\frac{-\delta}{\delta-1}\right) = -\frac{1}{\delta-1} \approx 1.$$

Thus for small $\delta$ we have that $(x_1\ x_2) \approx (1\ 1)$.

Under the finite precision system $F[b=10, m=4, e=5]$ with $\delta = 10^{-5}$ we have by backward substitution

$$\hat{x}_2 = \frac{\text{fl}(2-\frac{1}{\delta})}{\text{fl}(1-\frac{1}{\delta})} = \frac{\text{fl}(2-10^5)}{\text{fl}(1-10^5)} = \frac{\text{fl}(-99998)}{\text{fl}(-99999)} = 1$$

$$\hat{x}_1 = \text{fl}(\frac{1}{\delta}(1-\hat{x}_2)) = 0,$$

and so have generated a large error in $\hat{x}_1$.

Consider another approach to this problem, where we first interchange the two equations (and so use pivot 1 instead of $\delta$). We rewrite the linear system as

$$\begin{pmatrix} 1 & 1 \\ \delta & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}.$$

Thus, after applying Gaussian elimination

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

We recompute under the finite precision system

$$\hat{x}_2 = \frac{\text{fl}(1-2\delta)}{\text{fl}(1-\delta)} = \frac{\text{fl}(1-2\cdot 10^{-5})}{\text{fl}(1-10^{-5})} = \frac{\text{fl}(0.99998)}{\text{fl}(0.99999)} = 1$$

$$\hat{x}_1 = \text{fl}(2-\hat{x}_2) = \text{fl}(2-1) = 1.$$

and so avoid the large error.

We now consider the general case. Recall

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots \\ & a_{22}^{(2)} & a_{23}^{(2)} & \cdots \\ & a_{32}^{(2)} & a_{33}^{(2)} & \cdots \\ & \vdots & \vdots & \end{pmatrix}, \qquad M_2 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & \vdots & \ddots & \\ & \frac{-a_{32}^{(2)}}{a_{22}^{(2)}} & & \ddots \\ & \vdots & & & 1 \end{pmatrix}.$$

We note that we divide by the pivot element $a_{22}^{(2)}$ in $M_2$. In order to minimize the error, we should rearrange the rows in every step of the algorithm so that we get the largest possible pivot element (in absolute value). This will give us the most stable algorithm for computing the solution to the linear system since we avoid divisions by small pivot elements. This approach is called LU decomposition with partial pivoting.

## 3.4 Iterative Methods for solving $A\vec{x} = \vec{b}$

Recall that the LU decomposition requires $W = O(n^3)$ time to compute. If we specialize the type of matrix we wish to solve, there are different algorithms that may be able to solve the matrix more efficiently. This is especially true for sparse matrices, where iterative methods are very useful (these techniques are similar to Fixed-point iteration).

**Definition 3.7** *$A \in \mathbb{R}^{n\times n}$ is a **sparse matrix** if and only if the number of nonzero elements in $A$ is much smaller than $n^2$.*

For many sparse matrix applications, the number of nonzero elements per row is restricted to some small constant, *e.g.* 10. We can often store matrices of this type very efficiently (for example, compressed sparse row (CSR) format can be used). MATLAB supports a sparse storage format which can be invoked by `B = sparse(A)`.

Consider the special type of matrix:

**Definition 3.8** *$A \in \mathbb{R}^{n\times n}$ is **strictly diagonally dominant** if and only if*

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \tag{3.20}$$

*for all rows $i$.*

This type of matrix has the following useful property:

**Proposition 3.8** *A strictly diagonally dominant matrix is non-singular.*

**Proof.** Left as an exercise for the reader (this can be easily proven with the Gershgorin Circle Theorem).

**Example 3.9** *Consider the matrix $A \in \mathbb{R}^{3\times 3}$:*

$$A = \begin{pmatrix} 7 & 2 & 0 \\ 3 & 5 & -1 \\ 0 & 5 & -6 \end{pmatrix} \qquad \begin{matrix} 7 > 2 \\ 5 > 3 + 1 \\ 6 > 5 \end{matrix}$$

Clearly, $A$ is strictly diagonally dominant. In general if $A$ is strictly diagonally dominant, the transpose of $A$ does not necessarily retain the same property:

$$A^T = \begin{pmatrix} 7 & 3 & 0 \\ 2 & 5 & 5 \\ 0 & -1 & -6 \end{pmatrix} \qquad \begin{matrix} 7 > 3 \\ 5 \not> 2 + 5 \\ 6 > 1 \end{matrix}$$

### 3.4.1 Jacobi and Gauss-Seidel Methods

We wish to solve the linear system $A\vec{x} = \vec{b}$ using an iterative method. We write at each step

$$\vec{x}^{\text{old}} = \begin{pmatrix} x_1^{\text{old}} \\ x_2^{\text{old}} \\ x_3^{\text{old}} \end{pmatrix} \tag{3.21}$$

and determine $\vec{x}^{\text{new}}$ from $\vec{x}^{\text{old}}$ by either the Jacobi or Gauss-Seidel algorithm.

**Jacobi:** We assume $A \in \mathbb{R}^{3\times 3}$ and construct a system of equations as follows:

$$\begin{array}{ccccccc} a_{11}x_1^{\text{new}} & + & a_{12}x_2^{\text{old}} & + & a_{13}x_3^{\text{old}} & = & b_1 \\ a_{21}x_1^{\text{old}} & + & a_{22}x_2^{\text{new}} & + & a_{23}x_3^{\text{old}} & = & b_2 \\ a_{31}x_1^{\text{old}} & + & a_{32}x_2^{\text{old}} & + & a_{33}x_3^{\text{new}} & = & b_3. \end{array} \tag{3.22}$$

This system may be easily rearranged to solve for $\vec{x}^{\text{new}}$, giving us the defining equation for the Jacobi method:

$$x_i^{\text{new}} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1, j\neq i}^{n} a_{ij} x_j^{\text{old}} \right). \tag{3.23}$$

**Gauss-Seidel:** In the Jacobi method we may write $\vec{x}^{\text{new}} = J(\vec{x}^{\text{old}})$ for some function $J$. However, there is no reason to ignore the elements of $x_i^{\text{new}}$ derived earlier in the same step: we can indeed construct a linear system as

$$\begin{array}{ccccccc} a_{11}x_1^{\text{new}} & + & a_{12}x_2^{\text{old}} & + & a_{13}x_3^{\text{old}} & = & b_1 \\ a_{21}x_1^{\text{new}} & + & a_{22}x_2^{\text{new}} & + & a_{23}x_3^{\text{old}} & = & b_2 \\ a_{31}x_1^{\text{new}} & + & a_{32}x_2^{\text{new}} & + & a_{33}x_3^{\text{new}} & = & b_3. \end{array} \tag{3.24}$$

Rearranging yields the defining equation for the Gauss-Seidel method:

$$x_i^{\text{new}} = \frac{1}{a_{ii}} \left( b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{\text{new}} - \sum_{j=i+1}^{n} a_{ij} x_j^{\text{old}} \right). \tag{3.25}$$

For both of these methods, we must choose a starting vector $\vec{x}^{(0)}$ and generate the sequence $\vec{x}^{(1)}$, $\vec{x}^{(2)}$, $\ldots = \{\vec{x}^{(i)}\}_{i=1}^{\infty}$. We may also formulate these methods in matrix form using the decomposition $A = A_L + A_D + A_R$:

$$A = \underbrace{\begin{pmatrix} \ddots & & 0 \\ & 0 & \\ A_L & & \ddots \end{pmatrix}}_{A_L} + \underbrace{\begin{pmatrix} \ddots & & 0 \\ & A_D & \\ 0 & & \ddots \end{pmatrix}}_{A_D} + \underbrace{\begin{pmatrix} \ddots & & A_R \\ & 0 & \\ 0 & & \ddots \end{pmatrix}}_{A_R}. \tag{3.26}$$

Under the matrix decomposition, we may write the Jacobi method as

$$\vec{x}^{\text{new}} = A_D^{-1}(\vec{b} - (A_L + A_R)\vec{x}^{\text{old}}) \tag{3.27}$$

and the Gauss-Seidel method as

$$\vec{x}^{\text{new}} = A_D^{-1}(\vec{b} - (A_L\vec{x}^{\text{new}} + A_R\vec{x}^{\text{old}})). \tag{3.28}$$

These forms are equivalent to (3.23) and (3.25).

Can we guarantee convergence of the iterative methods? For some classes of matrices the answer is yes. It depends on the matrix $A$ whether the sequence of iterates converges and if so, how quickly it does. We can prove the following sufficient condition for convergence:

**Theorem 3.3** *Consider $A\vec{x} = \vec{b}$ and let $\vec{x}^{(0)}$ be any starting vector. Let $\{\vec{x}^{(i)}\}_{i=0}^{\infty}$ be the sequence generated by either Jacobi or Gauss-Seidel iterative methods. Then if $A$ is strictly diagonally dominant the sequence converges to the unique solution of the system $A\vec{x} = \vec{b}$.*

Since this theorem is only a sufficient condition, we can have a matrix $A$ that is not strictly diagonally dominant but leads to a convergent method.

In general, we note that often Gauss-Seidel will converge faster than Jacobi, but this is not always the case. For sparse matrices, we often obtain $W = O(n^2)$ for both methods.

### 3.4.2 Convergence of Iterative Methods

As a practical consideration, we need to determine when to stop iteration for a given iterative method. In particular, we need a measure of how close we are to a correct solution.

**Definition 3.9** *The residual of a linear system $A\vec{x} = \vec{b}$ for some vector $\vec{u}$ is*

$$\vec{r} = \vec{b} - A\vec{u}. \tag{3.29}$$

We write the residual at step $i$ as $\vec{r}^{(i)}$. As $\vec{r}^{(i)}$ becomes smaller, we approach convergence of the system. Hence, for a given relative tolerance $t_{rel} = 10^{-6}$ (for example), we compute the residual at each step and stop when $\|\vec{r}^{(i)}\|_2/\|\vec{r}^{(0)}\|_2 \leq t_{rel}$.

For an iterative approximation $\vec{u}$, the error is given by $\vec{e} = \vec{x} - \vec{u}$. This leads to the following relation between the error and the residual:

$$\begin{aligned} A\vec{e} &= A(\vec{x} - \vec{u}) \\ &= A\vec{x} - A\vec{u} \\ &= \vec{b} - A\vec{u} \\ &= \vec{r}. \end{aligned}$$

There are several benefits to using the residual instead of the error:

- $\vec{r}$ can be calculated easily (because calculating a matrix-vector product is "cheap" in terms of computational work compared with solving a linear system).
- $\vec{e}$ is generally unknown.

Sometimes $\vec{r}$ is small but $\vec{e}$ is large (this may happen when $\kappa(A)$ is large). Nevertheless, we will assume our linear system is well-conditioned and use $\vec{r}$ instead of $\vec{e}$ in the stopping criterion.

If we knew the error $\vec{e}$ for any approximation $\vec{u}$ we could write out the solution to the linear system directly:

$$\underbrace{\vec{x}}_{\text{exact}} = \vec{u} + (\vec{x} - \vec{u}) = \underbrace{\vec{u}}_{\text{approximate}} + \underbrace{\vec{e}}_{\text{error}}.$$

However, since the error is unknown, we can instead use the residual:

$$\vec{x} = \vec{u} + A^{-1}\vec{r}. \tag{3.30}$$

Unfortunately, inverting $A$ is an expensive operation (3 times as expensive as actually solving $A\vec{x} = \vec{b}$). Instead of using $A^{-1}$, we can choose a matrix $B$ that is easy to invert such that $B^{-1}$ is an approximation for $A^{-1}$. Then, we can obtain an approximation for $\vec{x}$ by the formula

$$\vec{x} \approx \vec{u} + B^{-1}\vec{r}. \tag{3.31}$$

Repeated application of this formula leads to a standard general form for an iterative method:

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + B^{-1}\vec{r}^{(i)},$$

or with the definition of the residual

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + B^{-1}(\vec{b} - A\vec{x}^{(i)}).$$

If $B$ is chosen appropriately then $\vec{x}^{(i)}$ will converge to $\vec{x}$.

This approach is basically fixed point iteration: we have defined a function $G$ such that $\vec{x}^{(i+1)} = G(\vec{x}^{(i)})$. If we choose $\vec{x}^{(i)} = \vec{x}$ then $\vec{x}^{(i+1)} = \vec{x}$ has to hold as well; thus $\vec{x} = G(\vec{x})$ is a fixed point of $G$. The convergence theorem for iterative methods is stated as follows:

**Theorem 3.4 (Convergence of Iterative Methods)** *Consider the iterative method*

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + B^{-1}(\vec{b} - A\vec{x}^{(i)}) \tag{3.32}$$

*with* $\det(A) \neq 0$. *If there exists a p-norm for which* $\|I - B^{-1}A\|_p < 1$ *then the iterative method will converge for any starting value* $\vec{x}^{(0)}$. *Convergence will then hold in any p-norm.*

**Proof.** Consider the error $\vec{e}^{(i)} = \vec{x} - \vec{x}^{(i)}$. By (3.32) we may write

$$\begin{aligned}
\vec{x}^{(i+1)} - \vec{x} &= \vec{x}^{(i)} - \vec{x} + B^{-1}(A\vec{x} - A\vec{x}^{(i)}) \\
-\vec{e}^{(i+1)} &= -\vec{e}^{(i)} + B^{-1}A\vec{e}^{(i)} \\
\vec{e}^{(i+1)} &= (I - B^{-1}A)\vec{e}^{(i)} \\
\|\vec{e}^{(i+1)}\|_p &= \|(I - B^{-1}A)\vec{e}^{(i)}\|_p \\
&\leq \|I - B^{-1}A\|_p \|\vec{e}^{(i)}\|_p.
\end{aligned}$$

If we apply this result recursively, we obtain

$$\|\vec{e}^{(i+1)}\|_p \leq \|I - B^{-1}A\|_p^{i+1} \|\vec{e}^{(0)}\|_p.$$

Taking the limit as $i \to \infty$ with $\|I - B^{-1}A\|_p < 1$ yields

$$\lim_{i\to\infty} \|\vec{e}^{(i+1)}\|_p = 0,$$

which is equivalent to

$$\lim_{i\to\infty} \vec{x}^{(i)} = \vec{x}. \ \square$$

We note that $\|I - B^{-1}A\|_p < 1$ means that $I \approx B^{-1}A$, or $B^{-1} \approx A^{-1}$. Recall from before that we require $B^{-1}$ to be an approximation for $A^{-1}$ in order to apply equation (3.31). The convergence theorem is very similar to Theorem 2.3 (The Contraction Mapping Theorem), which governs convergence of the fixed point method. In fact, Banach's Contraction Theorem (from real analysis) provides a generalization of both theorems.

Since (3.32) is a general form for iterative methods, we should be able to determine $B$ for the Jacobi and Gauss-Seidel methods. We rewrite the matrix form of Jacobi into standard form:

$$\begin{aligned}
\vec{x}^{(i+1)} &= A_D^{-1}(\vec{b} - (A_L + A_R)\vec{x}^{(i)}) \\
&= A_D^{-1}(\vec{b} - (A - A_D)\vec{x}^{(i)}) \qquad (from (3.26)) \\
&= A_D^{-1}A_D\vec{x}^{(i)} + A_D^{-1}(\vec{b} - A\vec{x}^{(i)}) \\
&= \vec{x}^{(i)} + A_D^{-1}(\vec{b} - A\vec{x}^{(i)}).
\end{aligned}$$

This is the standard form of an iterative method with $B^{-1} = A_D^{-1}$ (so $A^{-1}$ is approximated by $A_D^{-1}$.) We can rewrite the Gauss-Seidel method in a similar manner to recover $B^{-1} = (A_D + A_L)^{-1}$ (left as an exercise.)

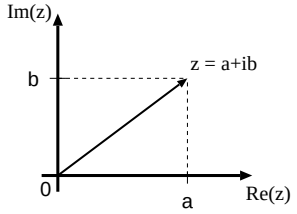
**These notes have been funded by...**

# Chapter 4

# Discrete Fourier Methods

## 4.1 Introduction

We define a complex number $z$ as $z = a + ib$, with $a$ the real part of $z$, $b$ the imaginary part, and $i = \sqrt{-1}$. As such, we may depict a complex number $z = a + ib$ as a vector in the complex plane:




Recall that for any complex number $z = a + ib$ we have:

| Term | Definition |
|---|---|
| Complex conjugate | $\bar{z} = a - ib$ |
| Real part | $\mathrm{Re}(z) = a$ |
| Imaginary part | $\mathrm{Im}(z) = b$ |
| Modulus | $r = \lvert z \rvert = \sqrt{a^2 + b^2}$ |
| Phase angle | $\theta = \arctan(b/a)$ |

We may write the complex number in terms of the modulus and phase angle

$$z = r \exp(i\theta), \tag{4.1}$$

in conjunction with the Euler formulas:

$$\exp(i\theta) = \cos(\theta) + i \sin(\theta), \tag{4.2}$$

$$\exp(-i\theta) = \cos(\theta) - i \sin(\theta). \tag{4.3}$$

We may invert these formulas in order to write sinusoids in terms of complex exponentials:

$$\cos(\theta) = \frac{1}{2}(\exp(i\theta) + \exp(-i\theta)), \tag{4.4}$$

$$\sin(\theta) = \frac{1}{2i}(\exp(i\theta) - \exp(-i\theta)). \tag{4.5}$$

Consider an arbitrary wave signal given by $y(t) = \sin(2\pi \cdot kt)$ for some constant $k$. The frequency of the wave is defined by

$$f = k, \tag{4.6}$$

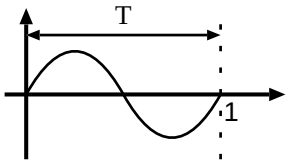
and has dimensions of oscillations per second (or Hz). The period is defined as the time required to complete one oscillation. It is related to the frequency by

$$T = \frac{1}{f} = \frac{1}{k}, \tag{4.7}$$

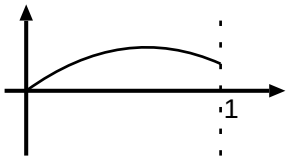
and has dimensions of seconds. The angular frequency $\omega$ is related to the frequency by

$$\omega = 2\pi\ f \tag{4.8}$$

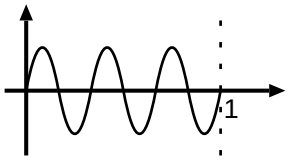
with dimensions of radians per second. These definitions can be easily generalized to any periodic function (not just sinusoids). This characterization is broadly applicable to sound waves, water waves and any other periodic behaviour. Human audible sound, for example, occurs in the frequency range from 20Hz to 20kHz.




$$\begin{cases} y(t) = \sin(2\pi t) \\ f = 1 \text{ oscillation / sec} = 1Hz \\ T = 1 \text{ sec} \\ \omega = 2\pi \text{ radians / sec} \end{cases}$$




$$\begin{cases} y(t) = \sin(2\pi \cdot \frac{1}{3}t) \\ f = \frac{1}{3} \text{ oscillation / sec} = \frac{1}{3}Hz \\ T = 3 \text{ sec} \\ \omega = \frac{2}{3}\pi \text{ radians / sec} \end{cases}$$

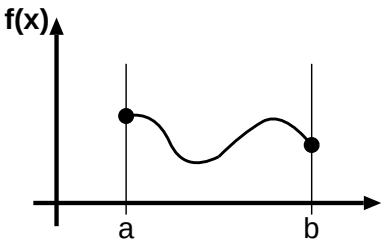



$$\begin{cases} y(t) = \sin(2\pi \cdot 3t) \\ f = 3 \text{ oscillation / sec} = 3Hz \\ T = \frac{1}{3} \text{ sec} \\ \omega = 6\pi \text{ radians / sec} \end{cases}$$

## 4.2 Fourier Series

### 4.2.1 Real form of the Fourier Series

Consider a continuous function $f(x)$ on $[a, b]$. In general, $f(x)$ can be expanded in a Fourier series as follows:




**Definition 4.1** *The **Fourier series** of $f(x)$ with $x \in [a,b]$ is*

$$g(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty} \left[ a_k \cos\left(k\frac{2\pi x}{b-a}\right) + b_k \sin\left(k\frac{2\pi x}{b-a}\right)\right], \tag{4.9}$$

*with*

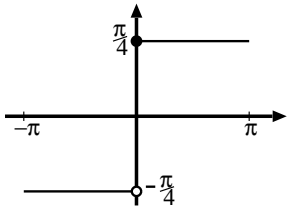
$$a_k = \frac{2}{b-a}\int_a^b f(x)\cos\left(k\frac{2\pi x}{b-a}\right)dx \tag{4.10}$$

$$b_k = \frac{2}{b-a}\int_a^b f(x)\sin\left(k\frac{2\pi x}{b-a}\right)dx. \tag{4.11}$$

**Example 4.1** *Compute the Fourier series of the function $f(x)$ defined by*

$$f(x) = \begin{cases} -\frac{\pi}{4} & x \in [-\pi, 0) \\ \frac{\pi}{4} & x \in [0, \pi] \end{cases}$$

*over the interval $[a,b] = [-\pi, \pi] \;\Rightarrow\; b - a = 2\pi$.*




We note that $f(x)$ is an odd function. Since $\cos(kx)$ is even, it follows that $f(x)\cos(kx)$ is odd. Thus,

$$a_k = \frac{2}{2\pi}\int_{-\pi}^{\pi} f(x)\cos(kx)dx = 0$$

for all $k$. We compute $b_k$ as follows:

$$\begin{aligned} b_k &= \tfrac{2}{2\pi}\textstyle\int_{-\pi}^{\pi} f(x)\sin(kx)dx \\ &= \tfrac{4}{2\pi}\textstyle\int_0^{\pi} f(x)\sin(kx)dx \quad \text{(because odd} \times \text{odd} = \text{even)} \\ &= \tfrac{1}{2}\textstyle\int_0^{\pi}\sin(kx)dx \\ &= \tfrac{1}{2}\tfrac{1}{k}\left[-\cos(kx)\right]_0^{\pi} \\ &= \tfrac{1}{2k}(-\cos(\pi k) + 1) \\ &= \tfrac{1}{2k}((-1)^{k+1} - 1). \end{aligned}$$

Thus

$$b_k = \begin{cases} \frac{1}{k} & \text{if } k \text{ is odd,} \\ 0 & \text{if } k \text{ is even.} \end{cases}$$

The complete Fourier series for $f(x)$, given by (4.9), may then be written as

$$g(x) = \sin(x) + \tfrac{1}{3}\sin(3x) + \tfrac{1}{5}\sin(5x) + \cdots .$$

The Fourier coefficients of odd and even functions simplify under the following proposition:

**Proposition 4.1** *The Fourier coefficients of a function $f(t)$ satisfy*

$$\begin{aligned} f(t) \text{ even} &\implies b_k = 0 \ \forall\ k, \\ f(t) \text{ odd} &\implies a_k = 0 \ \forall\ k. \end{aligned}$$

We may wonder: what functions can be expressed in terms of a Fourier series. The following fundamental theorem, originally proposed by Dirichlet, describes how the Fourier series $g(x)$ is related to the original function $f(x)$.

**Theorem 4.1** ***Fundamental Convergence Theorem for Fourier Series.*** *Let*

$$V = \left\{ f(x) \;\middle|\; \sqrt{\int_a^b f(x)dx} < \infty \right\}.$$

*Then for all $f(x) \in V$ there exist coefficients $a_0$, $a_k$, $b_k$ (with $1 \le k < \infty$) such that*

$$g_n(x) = \frac{a_0}{2} + \sum_{k=1}^{n} \left[ a_k \cos\left(k\frac{2\pi x}{b-a}\right) + b_k \sin\left(k\frac{2\pi x}{b-a}\right) \right]$$

*converges to $f(x)$ for $n \to \infty$ in the sense that $\sqrt{\int_a^b (f(x) - g(x))} = 0$ with $g(x) = \lim_{n\to\infty} g_n(x)$.*

Note that $g_n(x)$ is sometimes also called $S_n(x)$ (the $n^{\text{th}}$ partial sum of the Fourier series). This theorem holds for any bounded interval $[a, b]$, but for simplicity we will generally consider the interval to be $[a, b] = [-\pi, \pi]$.

$V$ is called the set of square integrable functions over $[a, b]$ and contains all polynomials, sinusoids and other nicely-behaved bounded functions. However, $V$ does not contain many common functions, including $f(x) = \tan(x)$ and $f(x) = (x - a)^{-1}$.

In addition, $V$ is a vector space of functions, *i.e.* if $f_1(x) \in V$ and $f_2(x) \in V$, then $c_1 f_1(x) + c_2 f_2(x) \in V$ ($\forall\ c_1, c_2 \in \mathbb{R}$). We define a norm over $V$ by

$$\|h(x)\|_2 = \sqrt{\int_a^b h(x)^2 dx} \qquad (L_2 \text{ norm}). \tag{4.12}$$

As a norm, this measures the "size" of the function $h(x)$. This implies a measure of the "distance" between $f(x) \in V$ and $g(x) \in V$ by

$$\|f(x) - g(x)\|_2 = \sqrt{\int_a^b (f(x) - g(x))^2 dx}. \tag{4.13}$$

We call the set of functions $V$ that are defined on an interval $[a,b]$ $L_2([a,b])$. We write

$$L_2([a,b]) = \{f(x) \mid \|f(x)\|_2 < \infty\}, \tag{4.14}$$

where $\|\cdot\|_2$ is the $L_2$ norm defined by (4.12).

### 4.2.2 Complex form of the Fourier Series

Perhaps a more natural method of representing the Fourier series is its complex form. This form of the series has only one sequence of coefficients, but each coefficient is complex-valued. We will later demonstrate that this form is equivalent to the real form of the series.

**Definition 4.2** *The **complex Fourier series** of a function $f(t)$ is*

$$h(t) = \sum_{k=-\infty}^{\infty} c_k \exp(ikt), \tag{4.15}$$

*with*

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\exp(-ikt)dt. \tag{4.16}$$

**Note:** In some books (and in MATLAB!)the complex form of the Fourier series is defined using a different sign convention, namely

$$h(t) = \sum c_k \exp(-ikt)$$

with

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\exp(ikt)dt.$$

**Relation between $c_k$ and $a_k$, $b_k$:** We may apply Euler's formula (4.3) to the complex form (4.15) to obtain

$$\begin{aligned} c_k &= \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)(\cos(-kt) + i\sin(-kt))dt \\ &= \frac{1}{2}\left(\frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\cos(kt)dt - i\frac{1}{\pi}\int_{-\pi}^{\pi} f(t)\sin(kt)dt\right). \end{aligned}$$

Comparing this expression with (4.10) and (4.11) reveals

$$c_k = \frac{1}{2}(a_k - ib_k). \tag{4.17}$$

This holds for $0 \le k$, but we can extend it to $-\infty < k < \infty$.

**Proposition 4.2** *The real and complex Fourier coefficients of a real function $f(t)$ obey*

1. $\overline{c_k} = c_{-k}$

2. $a_{-k} = a_k, \quad b_{-k} = -b_k$

3. $a_k = 2\ \mathrm{Re}(c_k), \quad b_k = -2\ \mathrm{Im}(c_k)$

4. $f(t)\ even \ \Rightarrow\ \mathrm{Im}(c_k) = 0, \quad f(t)\ odd \ \Rightarrow\ \mathrm{Re}(c_k) = 0$

5. $b_0 = 0, \quad c_0 = \frac{1}{2}a_0.$

**Proof.**

1. From (4.16) we write

$$\overline{c_k} = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(t)\exp(ikt)dt = c_{-k}.$$

   It also follows that $\mathrm{Re}(c_k) = \mathrm{Re}(c_{-k})$ (the real part of $c_k$ is even in $k$) and $\mathrm{Im}(c_k) = -\mathrm{Im}(c_{-k})$ (the imaginary part of $c_k$ is odd in $k$).

2. This result follows from (4.10) and (4.11).

3. This result follows from (4.17).

4. This result follows from Proposition 4.1 in conjunction with (4.17).

5. This result follows from (4.11) and (4.17). □

Although the complex and real forms of the Fourier series appear very different, they describe the same Fourier series. We demonstrate this result in the following theorem:

**Theorem 4.2** *The complex and real forms of the Fourier series are equivalent ($h(t) = g(t)$).*

**Proof.** By the Euler formulas (4.4) and (4.5) we may write

$$\begin{aligned} g(t) &= \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k\cos(kt) + b_k\sin(kt)) \\ &= \frac{a_0}{2} + \sum_{k=1}^{\infty}\left(\frac{a_k}{2}(\exp(ikt) + \exp(-ikt)) + \frac{b_k}{2i}(\exp(ikt) - \exp(-ikt))\right). \end{aligned}$$

We use $i^2 = -1 \implies i = -\frac{1}{i}$ to write

$$g(t) = \frac{a_0}{2} + \sum_{k=1}^{\infty}\left[\left(\frac{a_k - ib_k}{2}\right)\exp(ikt) + \left(\frac{a_k + ib_k}{2}\right)\exp(-ikt)\right].$$

Then by Proposition 4.2,

$$g(t) = c_0 + \sum_{k=1}^{\infty}\left[\left(\frac{a_k - ib_k}{2}\right)\exp(ikt) + \left(\frac{a_{-k} - ib_{-k}}{2}\right)\exp(-ikt)\right].$$

We apply the identity (4.17) and so write

$$g(t) = c_0 + \sum_{k=1}^{\infty}[c_k \exp(ikt) + c_{-k} \exp(-ikt)].$$

Thus,

$$g(t) = \sum_{k=-\infty}^{\infty} c_k \exp(ikt) = h(t),$$

which completes the proof. □

**Example 4.2** *Compute the complex Fourier series of the function $f(x)$, defined by*

$$f(x) = \begin{cases} -\frac{\pi}{4} & x \in [-\pi, 0] \\ \frac{\pi}{4} & x \in [0, \pi] \end{cases}$$

*over the interval $[a, b] = [-\pi, \pi] \;\Rightarrow\; b - a = 2\pi$.*

Recall from Example 4.1 that the real Fourier coefficients are

$$a_k = 0 \;\forall\; k, \qquad b_k = \begin{cases} \frac{1}{k}, & k \text{ odd} \\ 0, & k \text{ even.} \end{cases}$$

Then by (4.17) the complex coefficients are

$$c_k = \begin{cases} -\frac{i}{2k}, & k \text{ odd} \\ 0, & k \text{ even.} \end{cases}$$

The complex Fourier series is then

$$h(t) = \cdots + \frac{i}{6}\exp(-i3t) + \frac{i}{2}\exp(-it) - \frac{i}{2}\exp(it) - \frac{i}{6}\exp(i3t) - \cdots .$$

## 4.3 Fourier Series and Orthogonal Basis

Recall that for the vector space $V = \mathbb{R}^2$ we may define a standard basis $B = \{\vec{e}_1, \vec{e}_2\}$ with $\vec{e}_1 = (1, 0)$ and $\vec{e}_2 = (0, 1)$. The number of elements in the basis $B$ must be equal to the dimension of the vector space (in this case 2). We may then write any vector $\vec{x} \in \mathbb{R}^2$ as a linear combination of the basis vectors:

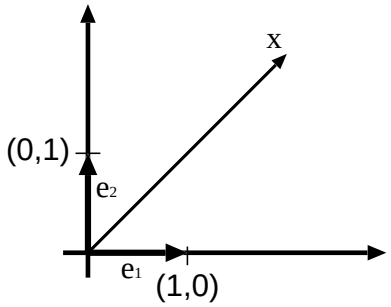
$$\vec{x} = x_1\vec{e}_1 + x_2\vec{e}_2. \tag{4.18}$$

We define the scalar product between two vectors in $V$ as

$$\vec{x} \cdot \vec{y} = x_1y_1 + x_2y_2 = <\vec{x}, \vec{y}> . \tag{4.19}$$

Any scalar product also induces a norm over the vector space by

$$\|\vec{x}\|_2 = \sqrt{<\vec{x}, \vec{x}>} = \sqrt{x_1^2 + x_2^2}. \tag{4.20}$$




In particular, we wish to focus on a particular type of basis which has some useful properties:

**Definition 4.3** *We say that a basis $B = \{\vec{e}_1, \ldots, \vec{e}_n\}$ is an **orthogonal basis** if and only if*

$$< \vec{e}_i, \vec{e}_j >= c_{ij} \text{ for } 1 \leq i, j \leq n,$$

*where $c_{ij}$ is nonzero if and only if $i = j$.*

We note that the standard basis $B = \{\vec{e}_1, \vec{e}_2\}$ defined over $V$ is orthogonal, since

$$< \vec{e}_1, \vec{e}_1 >= 1, \qquad < \vec{e}_2, \vec{e}_2 >= 1, \qquad < \vec{e}_1, \vec{e}_2 >= 0.$$

Given an orthogonal basis $\{\vec{e}_1, \vec{e}_2\}$ we can easily find the components of any vector $\vec{x}$ in the basis. Consider the scalar product of $\vec{x}$ and $\vec{e}_1$:

$$\begin{aligned} < \vec{x}, \vec{e}_1 > &= < x_1\vec{e}_1 + x_2\vec{e}_2, \vec{e}_1 > \\ &= x_1 < \vec{e}_1, \vec{e}_1 > + x_2 \underbrace{< \vec{e}_2, \vec{e}_1 >}_{=0} \\ &= x_1 < \vec{e}_1, \vec{e}_1 > . \end{aligned}$$

Thus we may write

$$x_i =< \vec{x}, \vec{e}_i > / < \vec{e}_i, \vec{e}_i > . \tag{4.21}$$

We now focus on the vector space of square integrable functions. We define our vector space by

$$V = L_2([-\pi, \pi]) = \{f(x) \,|\, \|f(x)\|_2 < \infty\}, \tag{4.22}$$

with basis

$$B = \{1, \cos(kt), \sin(kt)\} \qquad (1 \leq k < \infty). \tag{4.23}$$

This vector space has an infinite (but countable) number of linearly independent basis vectors and so has infinite (but countable) dimension. Since $B$ forms a basis for $V$, it follows that we may write any function $f(t) \in V$ as a linear combination of the basis vectors:

$$f(t) = a_0 \cdot 1 + \left[\sum_{k=1}^{\infty} a_k \cos(kt)\right] + \left[\sum_{k=1}^{\infty} b_k \sin(kt)\right]. \tag{4.24}$$

The scalar product over this vector space is given by

$$< f(t), g(t) >= \int_{-\pi}^{\pi} f(t)g(t)dt, \tag{4.25}$$

which induces the standard 2-norm

$$\|f(t)\|_2 = \sqrt{< f(t), f(t) >} = \sqrt{\int_{-\pi}^{\pi} f(t)^2 dt}. \tag{4.26}$$

In addition, we note that $B$ is an orthogonal basis:

**Proposition 4.3** *$B$ is an orthogonal basis for $V$.*

**Proof.** We note that $B$ is a basis due to Theorem 4.1. In order to prove that $B$ is orthogonal, we need to consider the scalar product of all basis vectors:

$$\begin{aligned}
< 1, 1 > &= \int_{-\pi}^{\pi} 1^2 dt = 2\pi \\
< \cos(kt), \cos(kt) > &= \int_{-\pi}^{\pi} \cos^2(kt)dt = \pi \quad (k \geq 1) \\
< \sin(kt), \sin(kt) > &= \int_{-\pi}^{\pi} \sin^2(kt)dt = \pi \quad (k \geq 1).
\end{aligned}$$

(If our basis vectors were normalized, each of these terms would equal 1). We must now show that the scalar products between different basis vectors all vanish:

$$\begin{aligned}
< 1, \cos(kt) > &= \int_{-\pi}^{\pi} \cos(kt)dt = 0 \quad (k \geq 1) \\
< 1, \sin(kt) > &= \int_{-\pi}^{\pi} \sin(kt)dt = 0 \quad (k \geq 1) \\
< \cos(kt), \sin(\ell t) > &= \int_{-\pi}^{\pi} \underbrace{\cos(kt)}_{\text{even}} \underbrace{\sin(\ell t)}_{\text{odd}} dt = 0 \quad (k, \ell \geq 1).
\end{aligned}$$

The scalar product between different cosine basis vectors requires a more extensive derivation:

$$\begin{aligned}
< \cos(kt), \cos(\ell t) > &= \int_{-\pi}^{\pi} \cos(kt)\cos(\ell t)dt = 0 \quad (k \neq \ell) \\
&= \int_{-\pi}^{\pi} \frac{1}{2}\left[\cos((k+\ell)t) + \cos((k-\ell)t)\right] dt \\
&= \frac{1}{2}\left[\frac{1}{k+\ell}\sin((k+\ell)t)\Big|_{-\pi}^{\pi} + \frac{1}{k-\ell}\sin((k-\ell)t)\Big|_{-\pi}^{\pi}\right] \\
&= 0.
\end{aligned}$$

We must also show $< \sin(kt), \sin(\ell t) >= 0$ for $(k \neq \ell \geq 1)$. This is left as an exercise for the reader. □

Since $B$ forms an orthogonal basis, we may use the projection formula (4.21) to determine the coefficients $a_k$ and $b_k$. By projection, we have

$$a_k = \frac{< f(t), \cos(kt) >}{< \cos(kt), \cos(kt) >} = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \cos(kt) dt \tag{4.27}$$

$$b_k = \frac{< f(t), \sin(kt) >}{< \sin(kt), \sin(kt) >} = \frac{1}{\pi} \int_{-\pi}^{\pi} f(t) \sin(kt) dt \tag{4.28}$$

$$\frac{a_0}{2} = \frac{< f(t), 1 >}{< 1, 1 >} = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) dt, \tag{4.29}$$

which are simply (4.10) and (4.11), the standard formulae for the Fourier coefficients.

For the complex form of the Fourier series, we can recover formula (4.16) by choosing the alternate basis

$$B = \{\exp(ikx)\} \quad (-\infty < k < \infty) \tag{4.30}$$

and using the scalar product for complex functions,

$$< f(t), g(t) >= \int_{-\pi}^{\pi} f(t) \overline{g(t)} dt. \tag{4.31}$$

Then, the scalar product between two basis vectors is

$$< \exp(ikt), \exp(i\ell t) >= \int_{-\pi}^{\pi} \exp(i(k - \ell)t) dt = 2\pi \delta_{k\ell}, \tag{4.32}$$

where $\delta_{k\ell}$ is the Kroenecker delta. Applying the projection formula yields

$$c_k = \frac{< f(t), \exp(ikt) >}{< \exp(ikt), \exp(ikt) >} = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(t) \exp(-ikt) dt, \tag{4.33}$$

which is (4.16).

## 4.4 Discrete Fourier Transform

The Fourier series is used to describe a continuous time signal $f(t)$ with $t \in [a, b]$, or $f(t)$ periodic. We wish to turn our attention now to the discrete time signal $f[n]$ over $N$ points in $0 \leq n \leq N - 1$. This type of signal may arise from sampling or digital recording of a continuous time signal, such as in music, images, stock indexes or weather data. Applying the Discrete Fourier Transform (DFT) to a discrete signal yields the frequencies present in the signal (this will be explained in more detail later.)

In order to present an expression for the discrete Fourier transform, we first require some intermediate results:

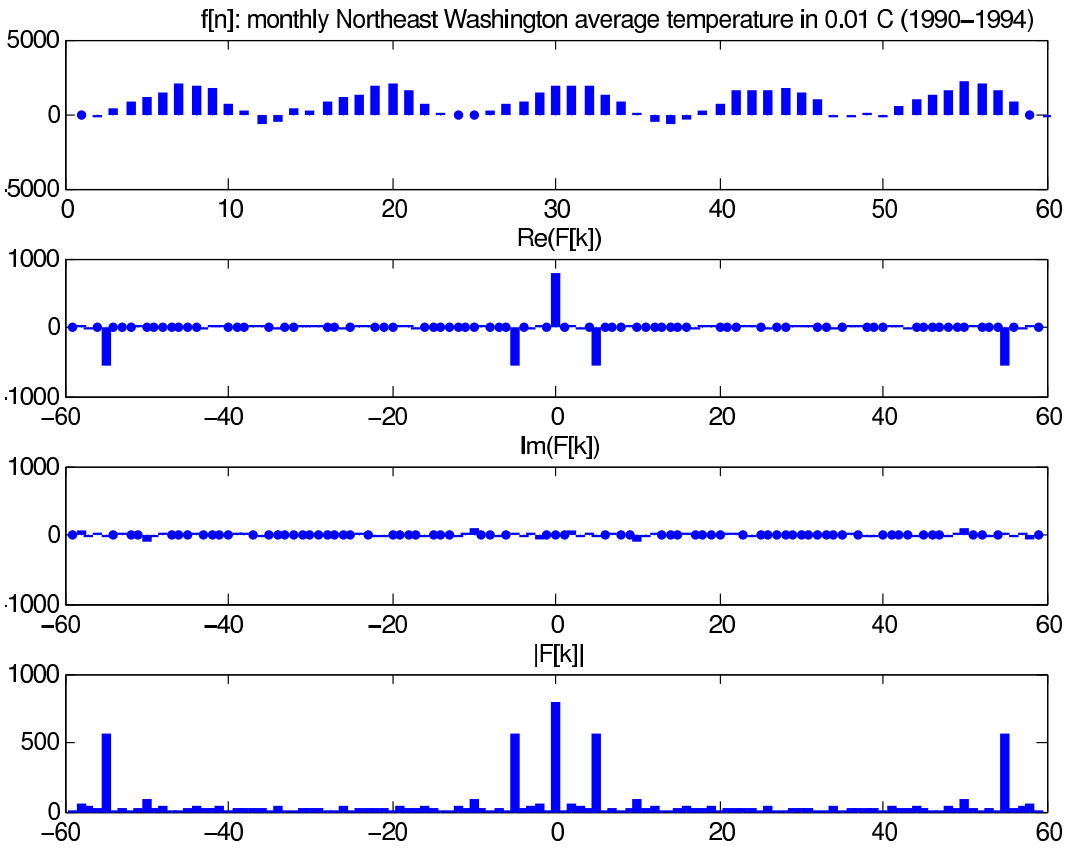



Figure 4.1: A discrete temperature profile $f[n]$ with Discrete Fourier transform $F[k]$.

**Definition 4.4** *The $N^{\text{th}}$* ***roots of unity*** *are the integer powers of*

$$W_N = \exp\left(i\frac{2\pi}{N}\right). \tag{4.34}$$

*We may write the $N$ distinct $N^{\text{th}}$ roots of unity as*

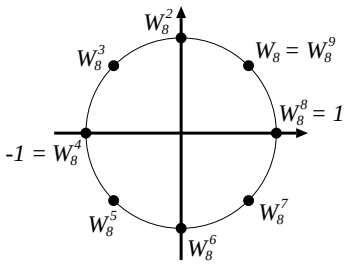
$$W_N^k = \exp\left(i\frac{2\pi k}{N}\right) \tag{4.35}$$

*for $0 \leq k < N$.*

**Example 4.3** *The $N^{\text{th}}$ roots of unity for $N = 8$ are*

$$W_8^k = \exp\left(i\frac{\pi k}{4}\right).$$

*In the complex plane, they may be depicted as*




**Proposition 4.4** *The $N^{\text{th}}$ roots of unity satisfy*

$$(W_N^k)^N = 1. \tag{4.36}$$

**Proof.** By definition,

$$(W_N^k)^N = \exp\left(i\frac{kN2\pi}{N}\right) = \exp(ik2\pi) = 1. \ \square$$

**Proposition 4.5** *The $N^{\text{th}}$ roots of unity satisfy*

$$W_N^{-k} = W_N^{N-k}. \tag{4.37}$$

**Proof.** By definition,

$$W_N^{-k} = W_N^N \cdot W_N^{-k} = W_N^{N-k}. \ \square$$

We are now prepared to define the discrete Fourier transform:

**Definition 4.5** *The* ***discrete Fourier transform*** *of a discrete time signal $f[n]$ with $0 \leq n \leq N-1$ is*

$$F[k] = DFT\{f[n]\} = \frac{1}{N}\sum_{n=0}^{N-1} f[n]\ W_N^{-kn}, \quad 0 \leq k \leq N-1. \tag{4.38}$$

**Definition 4.6** *The **inverse discrete Fourier transform** of a discrete frequency signal $F[k]$ with $0 \leq k \leq N-1$ is*

$$f[n] = IDFT\{F[k]\} = \sum_{k=0}^{N-1} F[k]\ W_N^{kn}, \quad 0 \leq n \leq N-1. \tag{4.39}$$

Note that these expressions are closely related to the complex form of the Fourier series, given in equation (4.15). Recall that when we compute the Fourier series of a function, we enforce that it is periodic outside the interval we are examining. This requirement is analogous in the discrete case; we implicitly assume that the time signal $f[n]$ is periodic when applying the discrete Fourier transform, which necessarily implies that the frequency signal is also periodic:

**Proposition 4.6** *The frequency signal $F[k]$ given by (4.38) is periodic with period $N$:*

$$F[k] = F[k + sN] \text{ with } s \in \mathbb{Z}. \tag{4.40}$$

**Proof.** This result follows from (4.38) and

$$W_N^{-(k+sN)n} = W_N^{-kn} W_N^{-nsN} = W_N^{-kn}. \ \square \tag{4.41}$$

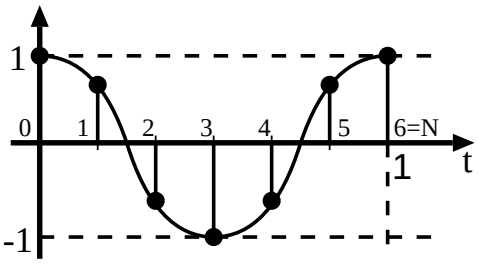
The discrete Fourier transform also leads to a set of symmetry properties for the frequency signal $F[k]$:

**Proposition 4.7** *For any real time signal $f[n]$, the frequency signal $F[k]$ satisfies*

1. $\mathrm{Re}(F[k])$ *is even in* $k$
2. $\mathrm{Im}(F[k])$ *is odd in* $k$
3. $\overline{F[k]} = F[-k]$
4. $f[n]$ *is even in* $n \ \Rightarrow \mathrm{Im}(F[k]) = 0$ *(the DFT is real)*
5. $f[n]$ *is odd in* $n \ \Rightarrow \mathrm{Re}(F[k]) = 0$ *(the DFT is purely imaginary)*

**Example 4.4** *Consider a cosine wave $f(t) = \cos(2\pi t),\ t \in [0,1]$. We sample at $N = 6$ points*

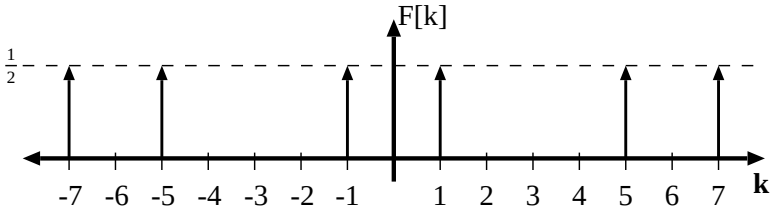
$$f[n] = \cos(2\pi t_n), \quad t_n = \frac{n}{N}, \quad 0 \leq n \leq N-1. \tag{4.42}$$



Using formula (4.4) we can rewrite $f[n]$ as

$$\begin{aligned} f[n] &= \frac{1}{2}\exp\left(i2\pi\frac{n}{N}\right) + \frac{1}{2}\exp\left(-i2\pi\frac{n}{N}\right) \\ &= \frac{1}{2}W_N^n + \frac{1}{2}W_N^{-n}. \end{aligned}$$

By (4.39) we conclude that the transformed signal is $F[1] = \frac{1}{2}$ and $F[-1] = \frac{1}{2}$ with all other $F[k]$ zero. Recall that by periodicity, $F[N-1] = F[-1]$.




We note that $\cos(2\pi t_n)$ is a low frequency wave, but we still get higher frequency components like $F[5]$. This effect is called aliasing.

### 4.4.1 Aliasing and the Sample Theorem

Consider a continuous time signal $f(t) = \cos(2\pi\ell t)$ with $t \in [0,1]$ (the frequency is $f = \ell$ in Hz). We sample $f(t)$ with $N = 6$ points at $t_n = \frac{n}{N} = \frac{n}{6}$ with $0 \le n \le 6$ to obtain a discrete time signal $f[n] = \cos(2\pi\ell\frac{n}{N})$. We consider $F[k] = DFT\{f[n]\}$ for $0 \le \ell \le 6$:

$\underline{\ell = 0:}$ The time signal is

$$\begin{aligned} f[n] &= \cos\left(2\pi \cdot 0 \cdot \tfrac{n}{N}\right) \\ &= 1 \cdot W_N^0, \end{aligned}$$

with frequency signal

$$F[0] = 1, \qquad F[6] = 1$$

due to periodicity.

$\underline{\ell = 1:}$ The time signal is

$$\begin{aligned} f[n] &= \cos\left(2\pi \cdot 1 \cdot \tfrac{n}{N}\right) \\ &= \tfrac{1}{2}\exp\left(i2\pi\tfrac{n}{N}\right) + \tfrac{1}{2}\exp\left(-i2\pi\tfrac{n}{N}\right) \\ &= \tfrac{1}{2}W_N^n + \tfrac{1}{2}W_N^{(N-1)n}, \end{aligned}$$

with frequency signal

$$F[1] = \tfrac{1}{2}, \qquad F[5] = \tfrac{1}{2}.$$

$\underline{\ell = 2:}$ The time signal is

$$\begin{aligned} f[n] &= \cos\left(2\pi \cdot 2 \cdot \tfrac{n}{N}\right) \\ &= \tfrac{1}{2}\exp\left(i2\pi\tfrac{n}{N}\cdot 2\right) + \tfrac{1}{2}\exp\left(-i2\pi\tfrac{n}{N}\cdot 2\right) \\ &= \tfrac{1}{2}W_N^{2n} + \tfrac{1}{2}W_N^{(N-2)n}, \end{aligned}$$

with frequency signal

$$F[2] = \tfrac{1}{2}, \qquad F[4] = \tfrac{1}{2}.$$

$\underline{\ell = 3:}$ The time signal is

$$\begin{aligned} f[n] &= \cos\left(2\pi \cdot 2 \cdot \tfrac{n}{N}\right) \\ &= \tfrac{1}{2}\exp\left(i2\pi\tfrac{n}{N}\cdot 3\right) + \tfrac{1}{2}\exp\left(-i2\pi\tfrac{n}{N}\cdot 3\right) \\ &= \tfrac{1}{2}W_N^{3n} + \tfrac{1}{2}W_N^{(N-3)n}, \end{aligned}$$

with frequency signal

$$F[3] = 1.$$

$\underline{\ell = 4:}$ The time signal is

$$\begin{aligned} f[n] &= \cos\left(2\pi \cdot 4 \cdot \tfrac{n}{N}\right) \\ &= \tfrac{1}{2}\exp\left(i2\pi\tfrac{n}{N}\cdot 4\right) + \tfrac{1}{2}\exp\left(-i2\pi\tfrac{n}{N}\cdot 4\right) \\ &= \tfrac{1}{2}W_N^{4n} + \tfrac{1}{2}W_N^{(N-4)n}, \end{aligned}$$

with frequency signal

$$F[4] = \tfrac{1}{2}, \qquad F[2] = \tfrac{1}{2}.$$

Examining these results, we may find it a little worrisome that the $\ell = 4$ case and the $\ell = 2$ case match. The sampling rate, or sampling frequency $f_s$ for this example is 6 samples / second, or $6Hz$. The "critical frequency" for aliasing occurs at $f = 3Hz$ (we have 2 $f = f_s$). In general, if $f_s \geq 2\ f$ then there will be no aliasing. If $f_s < 2\ f$ then aliasing will occur.

**Theorem 4.3** ***"Sampling Theorem" (loosely formulated)***
*In order to avoid aliasing error, the sampling frequency $f_s$ should be at least twice the largest frequency present in the continuous time signal.*

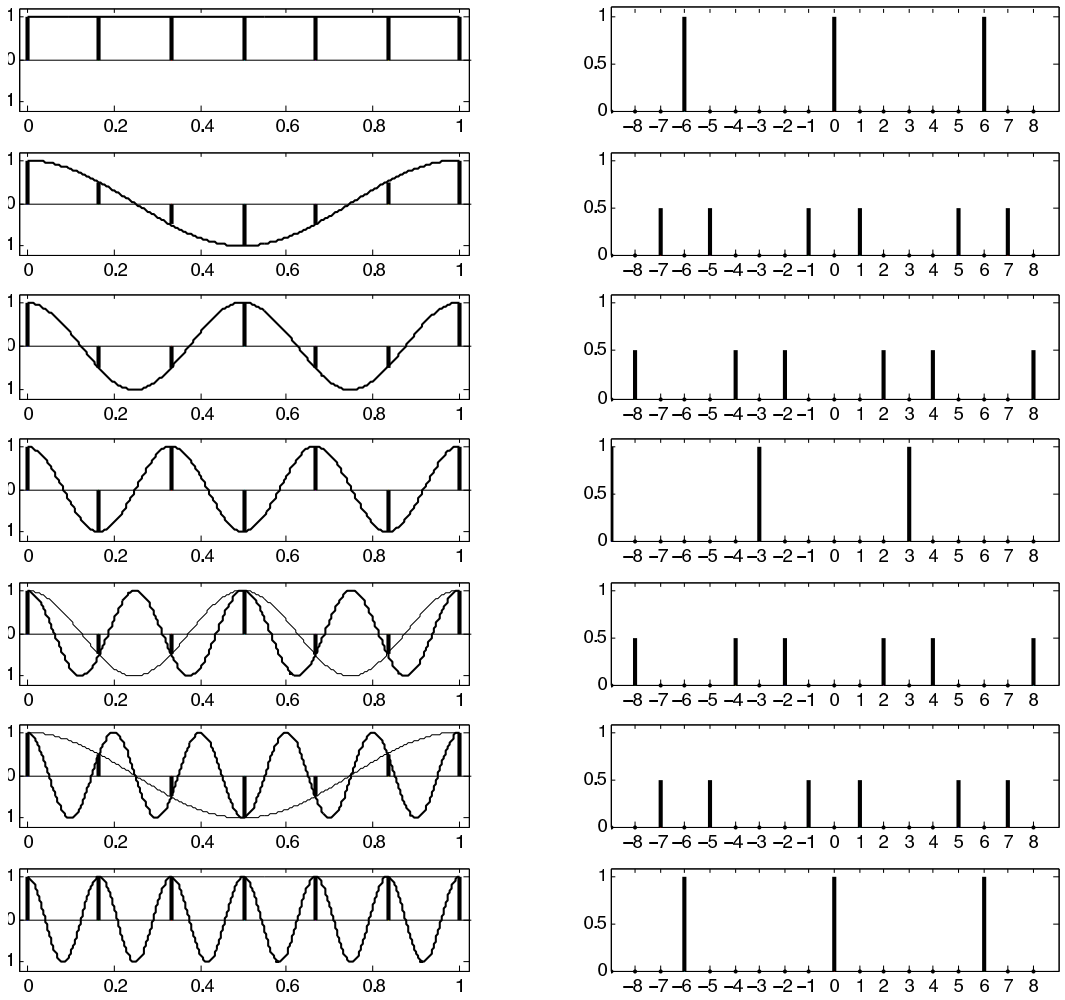



Figure 4.2: **Left:** Continuous time signal $f(t) = \cos(2\pi\ell t)$ with 6 discrete sampling points $f[n], 0 \leq \ell \leq 6$. **Right:** Discrete Fourier transforms of $f[n]$. Aliasing occurs for $\ell = 4, 5$and6: sampled high frequency signals show up as low-frequency discrete signals when the sampling rate is not high enough.

For example, human audible sound falls in the range $20Hz$ to $20kHz$. We will require a sampling frequency $f_s \geq 2 \cdot 20000Hz = 40kHz$ to avoid aliasing problems. As a result of this requirement, digital music CDs have a sampling frequency of $f_s$= $44.1kHz$.

## 4.5 Fast Fourier Transform

We wish to numerically compute the discrete Fourier transform $F[k]$ of a time signal $f[n]$. If we assume that the factors $W_N^{-kn}$ are precomputed and stored in a table (there will be $N$ factors), then we can use formula (4.38) directly to compute the $N$ coefficients $F[k]$ $(0 \leq k \leq N-1)$. The amount of work we must do per coefficient is then

$$W = (N-1)A + (N+1)M = 2N \text{ (complex) flops.}$$

The total work required to compute all coefficients is

$$W = N \cdot 2N = 2N^2 \text{ (complex) flops.}$$

This process is fairly inefficient; for a typical one minute sound file sampled at $44.1kHz$ we require $2.656 \times 10^6$ samples for the time signal and $1.4 \times 10^{13}$ (complex) flops to compute the discrete Fourier transform which is very large, even for today's fast computers. We pose the question: "how do we compute the discrete Fourier coefficients more efficiently?"

In order to answer this, we first require some intermediate results:

**Theorem 4.4** *If $N = 2^m$ for some integer $m$, then the length $N$ discrete Fourier transform $F[k]$ of discrete time signals $f[n]$ can be calculated by combining two length $\frac{N}{2}$ discrete Fourier transforms. We define*

$$g[n] = f[n] + f[n+N/2] \qquad (0 \leq n \leq N/2-1) \tag{4.43}$$
$$h[n] = (f[n] - f[n+N/2])W_N^{-n} \qquad (0 \leq n \leq N/2-1). \tag{4.44}$$

*Then*

$$F[2\ell] = \tfrac{1}{2}\, DFT\{g[n]\} \qquad \textit{(even indices)} \tag{4.45}$$
$$F[2\ell+1] = \tfrac{1}{2}\, DFT\{h[n]\} \qquad \textit{(odd indices).} \tag{4.46}$$

**Proof.** From the direct formula for the discrete Fourier transform,

$$\begin{aligned} F[k] &= \frac{1}{N}\sum_{n=0}^{N-1} f[n]W_N^{-kn} \\ &= \frac{1}{N}\sum_{n=0}^{N/2-1} f[n]W_N^{-kn} + \frac{1}{N}\sum_{n=N/2}^{N-1} f[n]W_N^{-kn} \\ &= \frac{1}{N}\sum_{n=0}^{N/2-1} f[n]W_N^{-kn} + \frac{1}{N}\sum_{\ell=0}^{N/2-1} f[\ell+N/2]W_N^{-k(\ell+N/2)} \\ &= \frac{1}{N}\sum_{n=0}^{N/2-1} (f[n]+f[n+N/2]W_N^{-kN/2})W_N^{-kn}. \end{aligned}$$

The inner root of unity may be simplified:

$$\begin{aligned} W_N^{-kN/2} &= \exp\left(-i2\pi\frac{kN/2}{N}\right) \\ &= \exp(-ik\pi) \\ &= (\exp(-i\pi))^k \\ &= (-1)^k. \end{aligned}$$

We now have two cases, based on whether $k$ is even or odd:

**Case 1 (k even):** We have $k = 2\ell$ $(0 \le \ell \le N/2-1)$. Thus,

$$\begin{aligned} F[2\ell] &= \frac{1}{N}\sum_{n=0}^{N/2-1} (f[n]+f[n+N/2])W_N^{-2\ell n} \\ &= \frac{1}{2}\left(\frac{1}{N/2}\sum_{n=0}^{N/2-1} g[n]W_{N/2}^{-\ell n}\right) \\ &= \tfrac{1}{2}\, DFT\{g[n]\}. \end{aligned}$$

**Case 2 (k odd):** We have $k = 2\ell+1$ $(0 \le \ell \le N/2-1)$. Thus,

$$\begin{aligned} F[2\ell+1] &= \frac{1}{N}\sum_{n=0}^{N/2-1} (f[n]-f[n+N/2])W_N^{-2\ell n-n} \\ &= \frac{1}{2}\left(\frac{1}{N/2}\sum_{n=0}^{N/2-1} h[n]W_{N/2}^{-\ell n}\right) \\ &= \tfrac{1}{2}\, DFT\{h[n]\}. \ \square \end{aligned}$$

If we split the discrete Fourier transform into two transforms (each of length $N/2$), the total work required will be

$$W_{tot} = 2 \cdot 2(N/2)^2 flops = N^2 flops.$$

Recall that the direct method requires $2N^2$ flops. In splitting up the Fourier transform, we only require half as much work. But we may further apply the splitting method recursively!

In order to compute the splitting, we require $N/2$ additions for $g[n]$, $N/2$ additions and multiplications for $h[n]$, and $N$ multiplications for the transform. In total, we will require $\frac{5}{2}N$ flops at each level (where $N$ is the length at each level).

**Theorem 4.5** *The fast Fourier transform (FFT) requires $O(N \log_2 N)$ flops.*

**Proof.** Assume $N = 2^m$. We tabulate the results at each step, as follows

| Level | Length per DFT | # of DFTs | Total work |
|---|---|---|---|
| $m$ | $2^m = N$ | 1 | $\frac{5}{2} \cdot N$ flops |
| $m-1$ | $2^{m-1}$ | 2 | $2 \cdot \frac{5}{2} \cdot \frac{N}{2} = \frac{5}{2} \cdot N$ flops |
| $m-2$ | $2^{m-2}$ | 4 | $4 \cdot \frac{5}{2} \cdot \frac{N}{4} = \frac{5}{2} \cdot N$ flops |
| $\vdots$ | $\vdots$ | $\vdots$ | $\vdots$ |
| 1 | 2 | $2^{m-1}$ | $2^{m-1} \cdot \frac{5}{2} \cdot \frac{N}{2^{m-1}} = \frac{5}{2} \cdot N$ flops |
| 0 | 1 | $2^m$ | None ($F[0] = f[0]$) |

We sum over the total work column:

$$\frac{5}{2}N \text{ flops per level } \cdot \log_2(N) \text{ levels} = \frac{5}{2} N \log_2 N \text{ flops},$$

which is our desired result. □

The pseudo-code for this algorithm is given as follows:

**Fast Fourier Transform**

```
function F = FastFT(f, N)
m = log N
if m = 0
  F = f
else
  build g[N]
  build h[N]
  F[even] = (1/2) FastFT(g, N/2)
  F[odd] = (1/2) FastFT(h, N/2)
end
```

We note that this algorithm can also be applied to signals where $N \neq 2^m$ by padding the signal with zeroes. In addition, this algorithm also works with complex $f[n]$, but requires that all computations be done in complex flops. Computationally, the fast Fourier transform is almost always used, due to its efficiency.

## 4.6 DFT and Orthogonal Basis

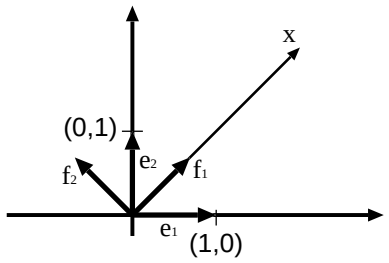
Recall that over the vector space $V = \mathbb{R}^2$ we may define an orthogonal basis

$$B = \{\vec{e}_1, \vec{e}_2\}, \qquad \text{with} < \vec{e}_i, \vec{e}_j >= \delta_{ij}$$

and an alternate orthogonal basis

$$B' = \{\vec{f}_1, \vec{f}_2\}, \qquad \text{with} < \vec{f}_i, \vec{f}_j >= \delta_{ij}.$$

Then any vector $\vec{x}$ can be expressed in terms of either basis (see figure).




Consider the discrete Fourier transform with $N = 4$, for simplicity. The time signal $f[n]$ $(0 \leq n \leq N-1 = 3)$ may be defined as a time signal vector

$$\vec{f} = \big(f[0], f[1], f[2], f[3]\big) \in \mathbb{R}^4. \tag{4.47}$$

Since $\mathbb{R}^4$ is a vector space, we may define an orthogonal basis

$$B = \{\vec{f}_0, \vec{f}_1, \vec{f}_2, \vec{f}_3\} \tag{4.48}$$

with

$$\begin{aligned} \vec{f}_0 &= (1,0,0,0), \\ \vec{f}_1 &= (0,1,0,0), \\ \vec{f}_2 &= (0,0,1,0), \\ \vec{f}_3 &= (0,0,0,1). \end{aligned}$$

Then the time signal vector may be written as

$$\vec{f} = f[0]\vec{f}_0 + f[1]\vec{f}_1 + f[2]\vec{f}_2 + f[3]\vec{f}_3. \tag{4.49}$$

From (4.39) we may also write the time signal vector as

$$\vec{f} = \left( \sum_{k=0}^{N-1} F[k] W_N^{k0}, \sum_{k=0}^{N-1} F[k] W_N^{k1}, \sum_{k=0}^{N-1} F[k] W_N^{k2}, \sum_{k=0}^{N-1} F[k] W_N^{k3} \right),$$

or

$$\vec{f} = \sum_{k=0}^{N-1} F[k] \vec{F}_k = F[0]\vec{F}_0 + F[1]\vec{F}_1 + F[2]\vec{F}_2 + F[3]\vec{F}_3, \tag{4.50}$$

where $\vec{F}_k$ is a vector in an alternate basis. We define

$$B' = \{\vec{F}_0, \vec{F}_1, \vec{F}_2, \vec{F}_3\}, \tag{4.51}$$

where

$$\begin{array}{rccccc} \vec{F}_0 = ( & W_N^{00}, & W_N^{01}, & W_N^{02}, & W_N^{03} & ) \longrightarrow k = 0 \\ \vec{F}_1 = ( & W_N^{10}, & W_N^{11}, & W_N^{12}, & W_N^{13} & ) \longrightarrow k = 1 \\ \vec{F}_2 = ( & W_N^{20}, & W_N^{21}, & W_N^{22}, & W_N^{23} & ) \longrightarrow k = 2 \\ \vec{F}_3 = ( & W_N^{30}, & W_N^{31}, & W_N^{32}, & W_N^{33} & ) \longrightarrow k = 3 \\ & \downarrow & \downarrow & \downarrow & \downarrow & \\ & n=0 & n=1 & n=2 & n=3 & \end{array} \tag{4.52}$$

We conclude from (4.50) that the discrete Fourier coefficients $F[k]$ of the time signal $f[n]$ are just the coordinates of the time signal vector $\vec{f}$ in the DFT basis $B'$. This is the same as saying that the DFT is nothing more than a basis transformation (with a basis that is useful to extract frequency information).

Is $B'$ an orthogonal basis? Consider the vector space $V = \mathbb{C}^n$ (complex vectors). We define the scalar product between two vectors $\vec{x}, \vec{y} \in \mathbb{C}^n$ as

$$< \vec{x}, \vec{y} >= \vec{x} \cdot \overline{\vec{y}} = \sum_{j=1}^{n} x_j \overline{y_j}. \tag{4.53}$$

Under this definition, the scalar product of the basis vectors in $B'$ is

$$< \vec{F}_i, \vec{F}_j >= \sum_{k=0}^{N-1} \left(W_N^{ik}\right)\left(W_N^{-jk}\right) = \sum_{k=0}^{N-1} W_N^{(i-j)k}.$$

If $i = j$ then

$$< \vec{F}_i, \vec{F}_i >= \sum_{k=0}^{N-1} 1 = N. \tag{4.54}$$

If $i \neq j$ then by the geometric series formula we may write

$$< \vec{F}_i, \vec{F}_j >= \frac{\left(W_N^{i-j}\right)^N - 1}{W_N^{i-j} - 1}.$$

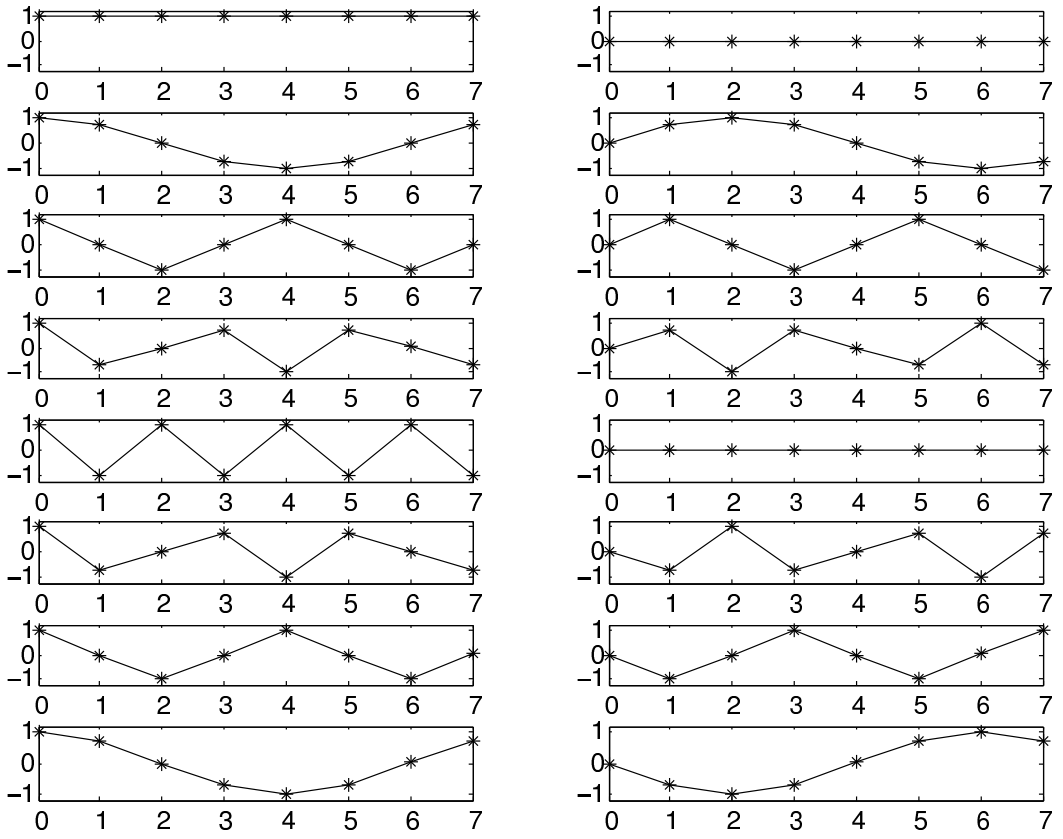



Figure 4.3: A basis for the discrete Fourier transform over $N = 8$ points. For each basis vector $\vec{F}_k$, the functions on the left represents the real component and the functions on the right represent the imaginary component.

But by the properties of $W_N$, $(W_N^{i-j})^N = (W_N{}^N)^{i-j} = 1$ so we get

$$< \vec{F}_i, \vec{F}_j >= 0,\ i \neq j, \tag{4.55}$$

and so $B'$ is an orthogonal basis!

Recall that since $B'$ is an orthogonal basis, we can use the projection formula (4.21) in conjunction with expressions (4.47) and (4.52) to find $F[k]$:

$$F[k] = \frac{< \vec{f}, \vec{F}_k >}{< \vec{F}_k, \vec{F}_k >} = \frac{1}{N} \sum_{n=0}^{N-1} f[n] \overline{W_N^{kn}} = \frac{1}{N} \sum_{n=0}^{N-1} f[n] W_N^{-kn}.$$

This is just the discrete Fourier transform formula (4.38).

## 4.7 Power Spectrum and Parseval's Theorem

Recall the Fourier series of a continuous function $f(t)$. We assume that $f(t)$ is the amplitude of a sound signal. From physics, we know that $\int_a^b f(t)^2 dt$ is proportional to the power in a sound signal on a time interval $[a, b]$. In the frequency domain, we define the concept of the power spectrum:

**Definition 4.7** *Let $F[k]$ be the complex Fourier coefficients of a discrete (or continuous) signal $f[n]$ (or $f(t)$). Then the **power spectrum** of $f[n]$ (or $f(t)$) is $|F[k]|^2$.*

Parseval's theorem then provides a connection between the power of a signal in the time domain and the summed power spectrum in the frequency domain:

**Theorem 4.6 (Parseval's Theorem.)**

*(A) (Continuous case) Let $F[k]$ be the complex Fourier coefficients of a real continuous signal $f(t)$. Then*

$$\underbrace{\frac{1}{b-a} \int_a^b f(t)^2 dt}_{\text{total power in time domain}} = \underbrace{\sum_{k=-\infty}^{\infty} |F[k]|^2}_{\text{total power in frequency domain}}. \tag{4.56}$$

*(B) (Discrete case) Let $F[k]$ be the complex Fourier coefficients of a real discrete signal $f[n]$. Then*

$$\underbrace{\frac{1}{N} \sum_{n=0}^{N-1} |f[n]|^2}_{\text{total power in time domain}} = \underbrace{\sum_{k=0}^{N-1} |F[k]|^2}_{\text{total power in frequency domain}}. \tag{4.57}$$

**Proof of (A).**

$$\begin{aligned}\frac{1}{b-a}\int_a^b f(t)^2 dt &= \frac{1}{b-a}\int_a^b f(t)\cdot f(t)dt \\ &= \frac{1}{b-a}\int_a^b \left(\sum_{k=-\infty}^{\infty} F[k]\exp\left(i2\pi\frac{kt}{b-a}\right) f(t)dt\right).\end{aligned}$$

We assume we can interchange the order of the integration and summation:

$$\begin{aligned}\frac{1}{b-a}\int_a^b f(t)^2 dt &= \sum_{k=-\infty}^{\infty} F[k]\left(\frac{1}{b-a}\int_a^b \exp\left(i2\pi\frac{kt}{b-a}\right) f(t)dt\right) \\ &= \sum_{k=-\infty}^{\infty} F[k]\cdot\overline{F[k]} \\ &= \sum_{k=-\infty}^{\infty} |F[k]|^2 . \ \square\end{aligned}$$

The proof of (B) is analogous.

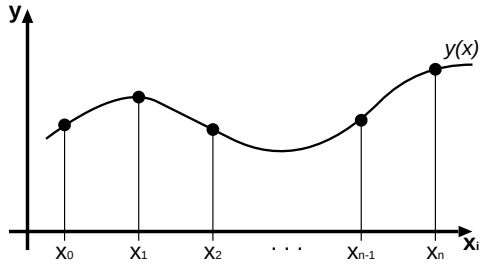
**These notes have been funded by...**

# Chapter 5

# Interpolation

We wish to now focus our attention on the problem of interpolation. If we have a set of discrete data, we may wish to determine a continuous function that interpolates all the data points. Consider the following statement of the problem:

**Problem** Given $n+1$ discrete data points $\{(x_i, f_i)\}_{i=0}^{n}$ with $x_i \neq x_j$ for $i \neq j$, determine a continuous function $y(x)$ that interpolates the data: $y(x_i) = f_i$ for $0 \leq i \leq n$.




The points $(x_i, f_i)$ could come from measurements, expensive calculations, discrete data analysis, or computer graphics (2D and 3D). There are several reasons for which we require access to a continuous function $y(x)$. For example,

- we may need to determine interpolated values at $x \neq x_i$.
- we may need to differentiate or integrate the interpolating function.

It may be impossible or infeasible to perform a continuous measurement of the function or to determine the function exactly, so interpolation is useful!

There are many ways for us to choose the function $y(x)$, some which may be very difficult and require substantial computational resources. We will focus on polynomial interpolation, which is one of the most general and widely-used methods.

## 5.1 Polynomial Interpolation

In polynomial interpolation, we wish to determine the coefficients of a polynomial that interpolates the set of points. We define the interpolating polynomial as follows:

**Definition 5.1** *Given $n+1$ discrete data points $\{(x_i, f_i)\}_{i=0}^{n}$ with $x_i \neq x_j$ for $i \neq j$, the* ***interpolating polynomial*** *is the degree $n$ polynomial*

$$y_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n, \tag{5.1}$$

*such that $y_n(x_i) = f_i$ for $0 \leq i \leq n$.*

In this problem we are given $n+1$ unknowns $a_0, a_1, \cdots, a_n$ and $n+1$ conditions $y_n(x_i) = f_i$. In order to solve for the interpolating polynomial, we must solve for the unknowns under the given conditions. There are many methods to solve for these unknowns, but we will concentrate on the Vandermonde matrix solution and the Lagrange form of the polynomial.

### 5.1.1 The Vandermonde Matrix

The Vandermonde matrix method is the most straightforward algorithm for determining an interpolating polynomial. Since all the conditions are linear, we can write them as a $(n+1) \times (n+1)$ linear system:

$$\begin{array}{ccccccccccc} a_0 & + & a_1 x_0 & + & a_2 x_0^2 & + & \cdots & + & a_n x_0^n & = & f_0 \\ a_0 & + & a_1 x_1 & + & a_2 x_1^2 & + & \cdots & + & a_n x_1^n & = & f_1 \\ & \cdots & & \cdots & & \cdots & & \cdots & & \cdots & \\ a_0 & + & a_1 x_n & + & a_2 x_n^2 & + & \cdots & + & a_n x_n^n & = & f_n, \end{array} \tag{5.2}$$

which we may also write in matrix form:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}. \tag{5.3}$$

Thus we may write $V\vec{a} = \vec{f}$, where $V$ is known as a Vandermonde matrix. We can obtain the following explicit expression for the determinant of a Vandermonde matrix:

**Proposition 5.1** *The determinant of $V$ is given by*

$$\begin{array}{rcl} \det(V) & = & (x_1 - x_0) \\ & & (x_2 - x_0)(x_2 - x_1) \\ & & (x_3 - x_0)(x_3 - x_1)(x_3 - x_2) \\ & & \cdots \\ & & (x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1}), \end{array} \tag{5.4}$$

*or, in a more compact form,*

$$\det(V) = \prod_{0 \leq i \leq j \leq n} (x_j - x_i). \tag{5.5}$$

**Proof.** Recall that we may expand $\det(V)$ about row $i$ by writing

$$\det(V) = \sum_{j=0}^{n} (-1)^{i+j} a_{ij} \det(V_{ij}),$$

where $V_{ij}$ is the $(n-1) \times (n-1)$ matrix obtained by removing row $i$ and column $j$ from matrix V:

$$V_{ij} = \begin{pmatrix} v_{00} & v_{01} & \cdots & v_{(0)(j-1)} & v_{(0)(j+1)} & \cdots & v_{0n} \\ v_{10} & v_{11} & \cdots & v_{(1)(j-1)} & v_{(1)(j+1)} & \cdots & v_{1n} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ v_{(i-1)(0)} & v_{(i-1)(1)} & \cdots & v_{(i-1)(j-1)} & v_{(i-1)(j+1)} & \cdots & v_{(i-1)(n)} \\ v_{(i+1)(0)} & v_{(i+1)(1)} & \cdots & v_{(i+1)(j-1)} & v_{(i+1)(j+1)} & \cdots & v_{(i+1)(n)} \\ \vdots & \vdots & & \vdots & \vdots & & \vdots \\ v_{n0} & v_{n1} & \cdots & v_{(n)(j-1)} & v_{(n)(j+1)} & \cdots & v_{nn} \end{pmatrix}.$$

We choose to expand $\det(V)$ about row $i = n$:

$$\begin{aligned} \det(V) &= \sum_{j=0}^{n} (-1)^{n+j} v_{nj} \det(V_{nj}) \\ &= (-1)^n \Big[ \, 1 \cdot \boxed{\det(V_{n0})} - x_n \cdot \boxed{\det(V_{n1})} + x_n^2 \cdot \boxed{\det(V_{n2})} - \\ &\quad \cdots + (-1)^n x_n^n \cdot \boxed{\det(V_{nn})} \, \Big]. \end{aligned}$$

All of the sub-determinants (indicated by boxes) are independent of $x_n$ since they do not have any elements from the $n^{\text{th}}$ row: This is essentially a polynomial of degree $n$ in $x_n$, so we may write $\det(V) = p_n(x_n)$.

We know $p_n(x_1) = 0$ because $\det(V) = 0$ when $x_n = x_1$; $V$ then has two equal rows. But we also know $p_n(x_2) = 0$, $p_n(x_3) = 0$, $p_n(x_4) = 0$, $\cdots$, $p_n(x_{n-1}) = 0$, which gives us $n$ roots of $p_n(x_n)$. From this information, we know that $p_n(x)$ and may be written as:

$$\det(V) = b(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1}), \tag{5.6}$$

with $b = (-1)^{2n} \det(V_{nn}) = \det(V_{nn})$.

We define $V^{(i)}$ as the $(i+1) \times (i+1)$ matrix formed by taking the first $i+1$ rows and $i+1$ columns of $V$. Then $V^{(n)} = V$ and $V^{(n-1)} = V_{nn}$. We may write

$$\begin{aligned} \det(V^{(n)}) &= \det(V^{(n-1)})(x_n - x_0)(x_n - x_1) \cdots (x_n - x_{n-1}) \\ \det(V^{(n-1)}) &= \det(V^{(n-2)})(x_{n-1} - x_0)(x_{n-1} - x_1) \cdots (x_{n-1} - x_{n-2}) \\ &\cdots \end{aligned}$$

Our result is obtained by repeating the decomposition (5.6) recursively to obtain the desired result. □

We may wish to consider when the interpolating polynomial is well-defined, *i.e.* when we may solve the linear system (5.3) to find a unique solution. It turns out that as long as

$x_i \neq x_j$ for $i \neq j$, we can always obtain a polynomial that interpolates the given points. This is proven in the following theorem:

**Theorem 5.1** *The interpolating polynomial $y_n(x)$ exists and is unique.*

**Proof.** We consider the system $V\vec{a} = \vec{f}$. If $x_i \neq x_j$ for $i \neq j$ then we know $\det(V) \neq 0$ from (5.5). Thus, by a standard result from linear algebra, we know the linear system has a unique solution (or, $y_n$ exists and is unique). □

We note that we rarely solve the linear system $V\vec{a} = \vec{f}$ in practice, for two reasons:

1. We note that this approach requires $W = O(n^3)$ time to solve the linear system. There are more efficient methods than this to find the interpolating polynomial.
2. $V$ is a very ill-conditioned matrix, since $\kappa_2(V)$ grows faster than exponentially as a function of $n$.

Instead, we consider a different approach which will allow us to write down the interpolating polynomial directly.

### 5.1.2 Lagrange Form

To motivate the Lagrange form of the interpolating polynomial, we ask the question: "is there a simply way to write down the interpolating polynomial without needing to solve a linear system?" Consider the most simple case of a non-constant polynomial:

**Linear Case (n=1):** We have two points $(x_0, f_0)$ and $(x_1, f_1)$. The polynomial is of the form

$$y_1(x) = a_0 + a_1 x,$$

with the conditions

$$y_1(x_0) = f_0, \qquad y_1(x_1) = f_1.$$

With a little intuition, we may think to write $y_1(x)$ as

$$y_1(x) = \frac{x - x_1}{x_0 - x_1} f_0 + \frac{x - x_0}{x_1 - x_0} f_1.$$

In this case, $y_1(x)$ takes the form

$$y_1(x) = \ell_0(x) f_0 + \ell_1(x) f_1,$$

where $\ell_0(x)$ and $\ell_1(x)$ are both degree 1 polynomials. We verify that $y_1(x)$ is a interpolating polynomial:

$$\begin{aligned} y_1(x_0) &= 1 \cdot f_0 + 0 \cdot f_1 = f_0, \qquad OK! \\ y_1(x_1) &= 0 \cdot f_0 + 1 \cdot f_1 = f_1, \qquad OK! \end{aligned}$$

By Theorem 5.1 we know that the interpolating polynomial is unique, so this must be the interpolating polynomial associated with the given points. If we collected the terms of this polynomial, we would find that this is simply another way of writing the solution we would get if we solved the Vandermonde system (5.3).

We may generalize this method for writing the interpolating polynomial to an arbitrary number of points as follows:

**Definition 5.2** *The $n+1$* ***Lagrange polynomials*** *for a set of points $\{(x_i, f_i)\}_{i=0}^n$ are the degree $n$ polynomials that satisfy the property*

$$\ell_i(x_j) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{5.7}$$

Explicitly, we may write the $i^{\text{th}}$ Lagrange polynomial as

$$\ell_i(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}. \tag{5.8}$$

Using product notation, we may also write

$$\ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x-x_j}{x_i-x_j}. \tag{5.9}$$

In general, the Lagrange form of the interpolating polynomial may be written as follows:

$$y_n(x) = \ell_0(x)f_0 + \ell_1(x)f_1 + \cdots + \ell_n(x)f_n, \tag{5.10}$$

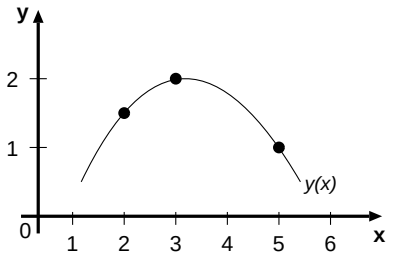
or, using summation notation,

$$y_n(x) = \sum_{i=0}^{n} \ell_i(x) f_i, \tag{5.11}$$

with the Lagrange polynomials $\ell_i(x)$ defined by (5.8). This form is an alternative way of writing the interpolating polynomial $y_n(x)$. Using this form, interpolation can be done in $O(n^2)$ time without solving a linear system!

**Example 5.1** *Write the interpolating polynomial of degree 2 for the set of points*

$$\{(2, \frac{3}{2}), (3, 2), (5, 1)\}.$$



The Lagrange form of the interpolating polynomial is

$$y_2(x) = \frac{(x-3)(x-5)}{(2-3)(2-5)}(\frac{3}{2}) + \frac{(x-2)(x-5)}{(3-2)(3-5)}(2) + \frac{(x-2)(x-3)}{(5-3)(5-3)}(1).$$

**The Lagrange Basis.** Consider the set

$$P_n(x) = \{y_n(x) | y_n(x) \text{ is a polynomial of degree} \leq n\}. \tag{5.12}$$

This is the set of all polynomials of degree less than or equal to $n$. We note that $P_n(x)$ is a vector space with the standard basis

$$B = \{1, x, x^2, \cdots, x^n\}.$$

Hence we may write any polynomial $y_n(x)$ as a linear combination of the basis vectors. In standard form, we write

$$y_n(x) = a_0 + a_1 x + \cdots + a_n x^n. \tag{5.13}$$

Similarly, the Lagrange polynomials form a different basis for the vector space $P_n(x)$:

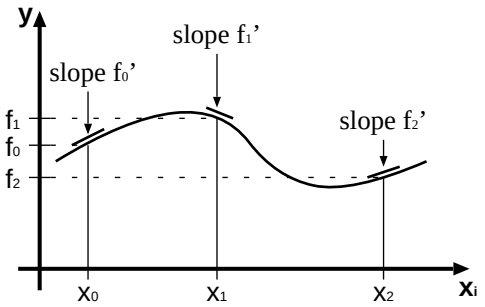
$$B' = \{\ell_0(x), \ell_1(x), \cdots, \ell_n(x)\} \tag{5.14}$$

(recall that in order to write a Lagrange basis, we require $n$ points $x_i$ $(1 \leq i \leq n)$ with $x_i \neq x_j$ for $i \neq j$.) Since this is also a basis, we may write any polynomial $y_n(x)$ as a linear combination of the Lagrange polynomials:

$$y_n(x) = \sum_{i=0}^{n} f_i \ell_i(x). \tag{5.15}$$

### 5.1.3 Hermite Interpolation

Sometimes derivatives (or slopes) are given or known at interpolation points. In this case, we can find a polynomial that interpolates both the function values and the derivatives.




**Definition 5.3** *Given* $\{(x_i, f_i, f_i')\}_{i=0}^n$, *the Hermite interpolating polynomial is the polynomial* $y(x)$ *of degree* $2n+1$ *which satisfies*

| | |
|---|---|
| $y(x_i) = f_i$ | *n+1 conditions* |
| $y'(x_i) = f_i'$ | *n+1 conditions* |
| | $2n+2$ *conditions.* |

Since there are $2n+2$ conditions, there must be $2n+2$ polynomial coefficients in the minimal degree interpolating polynomial. We conclude that $y(x)$ has degree $2n+1$.

**Example 5.2** *Consider the case of $n=1$. We have two points $(x_0, f_0, f_0')$ and $(x_1, f_1, f_1')$. The polynomial is of degree $2n+1 = 2\cdot 1+1 = 3$ (a cubic), so we may write*

$$y(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3.$$

Similar to the standard polynomial interpolation problem, we must solve for these coefficients. We consider two methods:

**Method 1: Undetermined Coefficients.** We note that we may write the polynomial and its first derivative as

$$\begin{aligned} y(x) &= a_0 + a_1 x + a_2 x^2 + a_3 x^3, \\ y'(x) &= a_1 + 2a_2 x + 3a_3 x^2. \end{aligned}$$

In matrix form, the linear system becomes

$$\begin{array}{l} y(x_0) = f_0 \\ y(x_1) = f_1 \\ y'(x_0) = f_0' \\ y'(x_1) = f_1' \end{array} \quad \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 \\ 1 & x_1 & x_1^2 & x_1^3 \\ 0 & 1 & 2x_0 & 3x_0^2 \\ 0 & 1 & 2x_1 & 3x_1^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix}.$$

Since this is simply a matrix system, we may use known techniques to solve this system. This method suffers from the same shortfalls as the Vandermonde system however, since we are still required to solve a potentially costly linear system.

**Method 2: Determine $a, b, c, d$** Similar to the idea of the Lagrange form, we can actually write the Hermite polynomial in a form that makes solving for the polynomial coefficients much easier. We write the polynomial and its derivative as

$$\begin{aligned} y(x) &= a + b(x-x_0) + c(x-x_0)^2 + d(x-x_0)^2(x-x_1) \\ y'(x) &= b + 2c(x-x_0) + 2d(x-x_0)(x-x_1) + d(x-x_0)^2. \end{aligned}$$

Substituting in the conditions yields the following linear system:

$$\begin{array}{ll} y(x_0) = f_0 & a = f_0 \\ y(x_1) = f_1 & a + b(x_1-x_0) + c(x_1-x_0)^2 = f_1 \\ y'(x_0) = f_0' & b = f_0' \\ y'(x_1) = f_1' & b + 2c(x_1-x_0) + d(x_1-x_0)^2 = f_1'. \end{array}$$

We note that $c$ and $d$ are the only coefficients we need to solve for, since $a$ and $b$ are immediately determined. We may rearrange the system to obtain

$$\begin{aligned} c &= \frac{1}{(x_1-x_0)^2}(f_1 - f_0 - f_0'(x_1-x_0)) \\ d &= \frac{1}{(x_1-x_0)^2}(f_1' - f_0' - \frac{2}{(x_1-x_0)}(f_1 - f_0 - f_0'(x_1-x_0))). \end{aligned}$$

So why does this work? Recall that we could write a basis for $P_3(x)$ as either $\{1, x, x^2, x^3\}$ (standard basis) or $\{\ell_0(x), \ell_1(x), \ell_2(x), \ell_3(x)\}$ (Lagrange basis). We may also choose the following basis for $P_3(x)$:

$$\{1, x - x_0, (x - x_0)^2, (x - x_0)^2(x - x_1)\}.$$

Writing $y(x)$ under this basis yields the form we used in method 2. This particular basis is chosen such that the calculations are somewhat simplified.

## 5.2 Piecewise Polynomial Interpolation

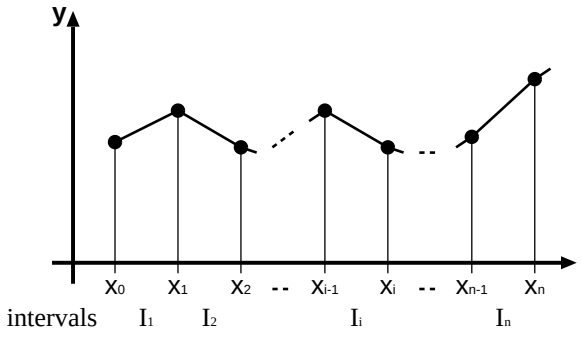
There are many problems with standard high degree polynomial interpolation, including

- strong oscillations
- quickly divergent extrapolation.

We can remedy these drawbacks by using piecewise interpolation or least-squares fitting. In the former, we break the interpolation interval into regions and generate several interpolating polynomials that are valid over each region. In the latter, we allow the data points to be 'close to' $y(x)$ instead of on $y(x)$.

### 5.2.1 Piecewise Linear Interpolation

In piecewise linear interpolation we split the domain of the function into a set of intervals (each consisting of two adjacent points) and determine the interpolating polynomial of each interval. We use a line segment to interpolate the function in each interval, since each interval contains two points.

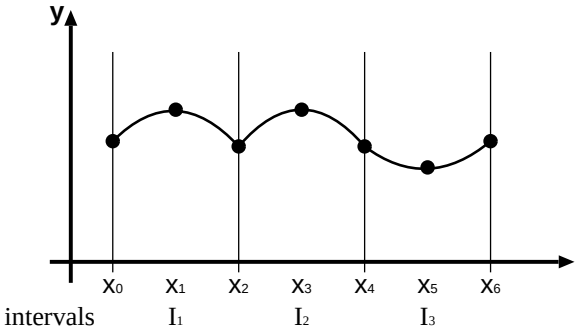



We define a set of $n$ polynomials $y_i(x)$, $1 \leq i \leq n$ where the domain of each $y_i(x)$ is $I_i = [x_{i-1}, x_i]$. We may write

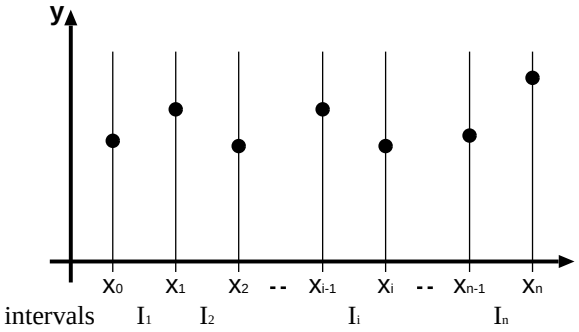
$$y_i(x) = \frac{x - x_i}{x_{i-1} - x_i} f_{i-1} + \frac{x - x_{i-1}}{x_i - x_{i-1}} f_i.$$

Then, the interpolating piecewise polynomial $y(x)$ is equal to $y_i(x)$ over the interval $I_i = [x_{i-1}, x_i]$ for all $1 \leq i \leq n$.

This method of interpolation has the drawback of not being smooth at the interpolation points (we end up with jagged edges on the interpolating curve). We may instead consider piecewise quadratic (see figure) or piecewise cubic interpolation, but these will also inevitably have jagged points at the boundaries.




### 5.2.2 Spline Interpolation




We consider a generalization of piecewise linear interpolation on a set of points $\{(x_i, f_i)\}_{i=0}^{n}$ that takes into account our desire for smoothness at the boundaries. Instead of only using an interpolation condition, we impose three types of conditions:

- interpolation conditions
- smoothness conditions
- extra boundary conditions.

This leads us to a very powerful type of interpolation, known as spline interpolation.

**Definition 5.4** *$y(x)$ is a degree $k$ spline if and only if*

1. *$y(x)$ is a piecewise polynomial of degree $k$ in each interval $I_i$. We define $y_i(x)$ as the restriction of $y(x)$ to $[x_{i-1}, x_i]$ $(1 \leq i \leq n)$.*
2. *$y_i(x_{i-1}) = f_{i-1}$ and $y_i(x_i) = f_i$ in interval $I_i$ $(1 \leq i \leq n)$ (interpolation condition).*

*3. For each interior point $j$, there are $k-1$ smoothness conditions:*

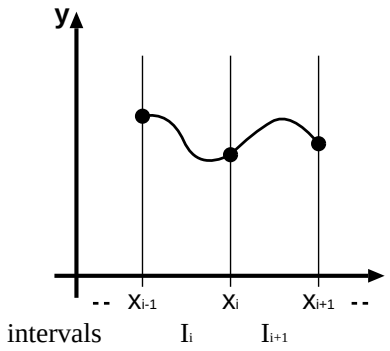
$$\left.\begin{array}{rcl} y_j'(x_j) & = & y_{j+1}'(x_j) \\ y_j''(x_j) & = & y_{j+1}''(x_j) \\ & \vdots & \\ y_j^{(k-1)}(x_j) & = & y_{j+1}^{(k-1)}(x_j) \end{array}\right\} \begin{array}{c} k-1 \text{ smoothness conditions} \\ (0 \le j \le n) \end{array}$$

Under this definition, there will be $n$ intervals and $k+1$ coefficients for each polynomial. In total, this gives $n(k+1) = nk + n$ unknowns.

We will have $2n$ interpolation conditions from part 2 and $(k-1)(n-1)$ smoothness conditions (note that the smoothness conditions only apply to the $n-1$ internal points of the spline.) Thus, in total we will have $2n + kn - k - n + 1 = kn + n - k + 1$ conditions.

Comparing the number of unknowns and the number of conditions, it is clear we need to impose $k-1$ extra conditions. These are supplied by extra boundary conditions at $x_0$ and $x_n$.

**Example 5.3** *Consider the case of $k = 3$ (a cubic spline.) We will need to impose 2 extra boundary conditions.*




There are many different types of boundary conditions we could impose. Three possible types of boundary conditions are

- "free boundary": $y_1''(x_0) = 0$, $y_n''(x_n) = 0$. A cubic spline with this boundary condition is known as a "natural" cubic spline.
- "clamped boundary": We specify the first derivatives at the ends by choosing constants $f_0'$ and $f_n'$. We then impose $y_1'(x_0) = f_0'$ and $y_n'(x_n) = f_n'$.
- "periodic boundary": If $f_0 = f_n$ we may impose that the first and second derivatives of the first and last polynomial are equal at $x_0$ and $x_n$. We obtain the conditions $y_1'(x_0) = y_n'(x_n)$ and $y_1''(x_0) = y_n''(x_n)$.

When we impose the three types of conditions, we will produce a $(nk+n) \times (nk+n)$ linear system that may be uniquely solved for the coefficients of the $y_i(x)$, $1 \le i \le n$.

### 5.2.3 Further Generalizations

Further generalizations of these techniques are possible. For example:

- Bezier Curves
- B-Splines

**These notes have been funded by...**

# Chapter 6

# Integration

The problem of numerical integration is simply stated:

**Problem** Given a continuous function $f(x)$ and an interval $[a, b]$, find a numerical approximation for $I = \int_a^b f(x)dx$.

There are several cases where numerical integration is necessary:

1. if $f(x)$ is given but no closed form solution can be found. For example, the integral of the function $f(x) = e^{-x^2}$ has no closed form solution and requires numerical integration to compute.
2. if $f(x)$ is not given, but $\{(x_i, f_i)\}_{i=0}^n$ is given.

In each case, numerical integration may be the only method of determining the integral. We consider three methods for performing numerical integration: integration of an interpolating polynomial, composite integration and Gaussian integration.

## 6.1 Integration of an Interpolating Polynomial

Recall the definition of an interpolating polynomial $y(x)$ of degree $n$ for given data points $\{(x_i, f_i)\}_{i=0}^n$. Since the interpolating polynomial provides an approximation to the function $f(x)$, we may use it to determine an approximate solution to the integral. This is advantageous because $y(x)$ may be easily integrated due to its simple form.

The exact solution, $I$ of to the integration problem is given by

$$I = \int_a^b f(x)dx, \tag{6.1}$$

and the numerical approximation $\hat{I}$, using the interpolating polynomial given by

$$\hat{I} = \int_a^b y(x)dx. \tag{6.2}$$

The truncation error $T$ is then defined as

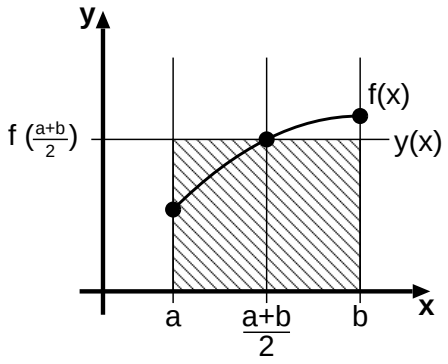
$$T = I - \hat{I}, \tag{6.3}$$

the difference between the exact solution and our approximation.

### 6.1.1 Midpoint Rule: $y(x)$ degree 0

We choose $y(x)$ constant and sample at $x_0 = \frac{a+b}{2}$. We obtain $y(x) = f_0 = f(\frac{a+b}{2})$. The numerical approximation is then

$$\hat{I}_0 = \int_a^b f\left(\frac{a+b}{2}\right) dx = (b-a) f\left(\frac{a+b}{2}\right). \tag{6.4}$$




### 6.1.2 Trapezoid Rule: $y(x)$ degree 1

We choose $y(x)$ to be a linear function between the endpoints of the interval,

$$\begin{array}{ll} (x_0 = a, & f_0 = f(x_0)) \text{ and} \\ (x_1 = b, & f_1 = f(x_1)). \end{array}$$

We may immediately write the interpolating polynomial in Lagrange form:

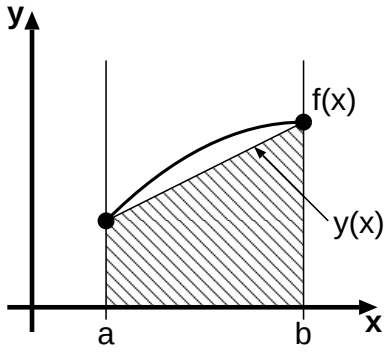
$$y(x) = \frac{(x-x_1)}{(x_0-x_1)} f_0 + \frac{(x-x_0)}{(x_1-x_0)} f_1.$$

The numerical approximation using the interpolating polynomial is then

$$\begin{aligned} \hat{I}_1 &= \int_{x_0}^{x_1} \left[ \frac{(x-x_1)}{(x_0-x_1)} f_0 + \frac{(x-x_0)}{(x_1-x_0)} f_1 \right] dx \\ &= \frac{f(a)}{a-b} \frac{(x-b)^2}{2}\bigg|_a^b + \frac{f(b)}{a-b} \frac{(x-a)^2}{2}\bigg|_a^b \\ &= \frac{f(a)}{a-b}\left(-\frac{(a-b)^2}{2}\right) + \frac{f(b)}{b-a}\left(\frac{(b-a)^2}{2}\right). \end{aligned}$$

From this we obtain the trapezoid rule

$$\hat{I}_1 = (b-a)\frac{1}{2}[f(a)+f(b)]. \tag{6.5}$$




### 6.1.3 Simpson Rule: $y(x)$ degree 2

We choose $y(x)$ to be a parabola within the interval. We interpolate the points

$$\begin{array}{ll} (x_0 = a, & f_0 = f(a)), \\ (x_1 = \frac{a+b}{2}, & f_1 = f(\frac{a+b}{2})), \text{ and} \\ (x_2 = b, & f_2 = f(b)). \end{array}$$

We may write the interpolating polynomial in Lagrange form:

$$y(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f_1 + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.$$

The numerical approximation using the interpolating polynomial is then

$$\begin{aligned} \hat{I}_2 \quad = \quad & \int_{x_0}^{x_2} \left[ \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f_0 + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f_1 + \right. \\ & \left. + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f_2 \right] dx. \end{aligned}$$

This may be written as a weighted linear combination of the $f_i$:

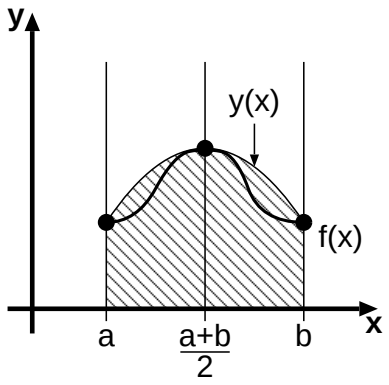
$$\hat{I}_2 = w_0 f_0 + w_1 f_1 + w_2 f_2,$$

with

$$w_0 = \int_{x_0}^{x_2} \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}dx = \frac{b-a}{6}.$$

Repeating a similar integration for $w_1$ and $w_2$ yields the Simpson rule

$$\hat{I}_2 = \frac{b-a}{6}(f_0 + 4f_1 + f_2). \tag{6.6}$$




### 6.1.4 Accuracy, Truncation Error and Degree of Precision

There are two ways to study the accuracy of the integration formulas:

**Truncation error:** We can use Taylor's theorem to compute the truncation error of each integration rule. For example, for the midpoint rule it can be shown that $T_0 = I - I_0 = \frac{(b-a)^3}{24} f''(\xi_0)$ with $\xi_0 \in (a, b)$.

**Degree of Precision:** The degree of precision of an integration formula is defined as follows:

**Definition 6.1** *The following statements are equivalent:*

- $\hat{I}$ *has* ***degree of precision*** $m$
- $T = I - \hat{I} = 0$ *for any* $f(x)$ *polynomial of degree* $\leq m$
- $\hat{I}$ *integrates any polynomial of degree* $\leq m$ *exactly.*

The three integration formulas we discussed are summarized in the following table, along with their accuracy:

| | Degree of poly. | Truncation Error | Degree of Precision |
|---|---|---|---|
| Midpoint - eq. (6.4) | 0 | $\frac{(b-a)^3}{24} f''(\xi_0)$ | 1 |
| Trapezoid - eq. (6.5) | 1 | $-\frac{(b-a)^3}{12} f''(\xi_1)$ | 1 |
| Simpson - eq. (6.6) | 2 | $-\frac{(b-a)^5}{2880} f^{(4)}(\xi_2)$ | 3 |

Clearly the Simpson rule is the most accurate approximation, but also requires the most computation. Perhaps surprising is the fact that the midpoint rule seems to provide comparable accuracy to the more computationally-intensive trapezoid rule. We consider the application of these methods to the following example:

**Example 6.1** *The error function* $\mathrm{erf}(x)$ *is defined as*

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x \exp(-t^2) dt.$$

*We can compute* $\mathrm{erf}(1)$ *to high precision using MATLAB or an advanced calculator and obtain* $\mathrm{erf} \approx 0.842701 \cdots$.

Using the three integration algorithms described above, we obtain:

**Midpoint** $\hat{I}_0 = \frac{2}{\sqrt{\pi}} \exp(-(\frac{1}{2})^2) \approx 0.\underline{8}78783$
(one correct digit)

**Trapezoid** $\hat{I}_1 = \frac{2}{\sqrt{\pi}} \left(\frac{1}{2}\right) [\exp((-0)^2) + \exp((-1)^2)] \approx 0.77173$
(zero correct digits)

**Simpson** $\hat{I}_2 = \frac{2}{\sqrt{\pi}} \left(\frac{1}{6}\right) [1 + 4\exp((-\frac{1}{2})^2) + \exp(-1)] \approx 0.\underline{843}1028$
(three correct digits)

As predicted, the Simpson rule is the most accurate. We also find, in this case, that the Midpoint rule is more accurate than the trapezoid rule.

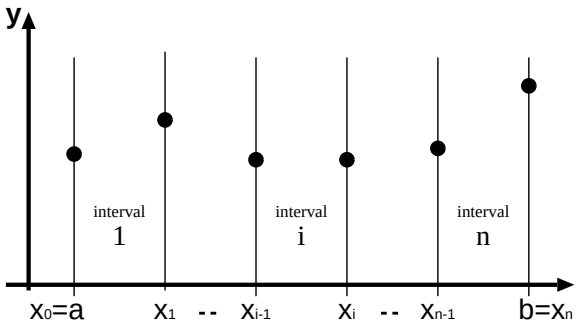
## 6.2 Composite Integration

If we wish to attain higher accuracy in the result, we may divide the integration interval into several smaller subintervals and integrate each of the subintervals individually. If we split up $[a, b]$ into $n$ subintervals of equal length, each subinterval will have length $h = \frac{b-a}{n}$. The exact integral is then written as

$$I = \int_a^b f(x) dx = \sum_{i=1}^{n} \int_{x_{i-1}}^{x_i} f(x) dx = \sum_{i=1}^{n} I_i, \tag{6.7}$$

with

$$I_i = \int_{x_{i-1}}^{x_i} f(x) dx. \tag{6.8}$$



### 6.2.1 Composite Trapezoid Rule

We may use the trapezoid rule to approximate the integral over each subinterval. We write:

$$\hat{I}_i = h\frac{f(x_{i-1}) + f(x_i)}{2}. \tag{6.9}$$

Summing over all intervals yields the complete expression for the composite trapezoid rule:

$$\hat{I} = \sum_{i=1}^{n} \hat{I}_i = \frac{h}{2}[f_0 + 2f_1 + 2f_2 + \cdots + 2f_{n-1} + f_n]. \tag{6.10}$$

**Local Truncation Error:** The local truncation error is the truncation error expected in each subinterval. We write

$$T_{loc,i} = I_i - \hat{I}_i = -\frac{1}{12}(x_i - x_{i-1})^3 f''(\xi_i), \qquad \text{with } \xi_i \in (x_{i-1}, x_i)$$

Hence, the local truncation error for the trapezoid rule is given by

$$T_{loc,i} = -\frac{1}{12}f''(\xi_i)h^3. \tag{6.11}$$

We say that the local truncation error is of the order $O(h^3)$.

**Global Truncation Error:** The global truncation error is the total truncation error over all intervals. We write

$$T_{global} = I - \hat{I} = \sum_{i=1}^{n}(I_i - \hat{I}_i) = \sum_{i=1}^{n} T_{loc,i}.$$

In order to relate $T_{global}$ to the interval length $h$, we will need the following theorem:

**Theorem 6.1** *The global truncation error for the composite trapezoid rule is $O(h^2)$.*

**Proof.** The global truncation error, in terms of the local truncation error, is given by:

$$|T_{global}| = |\sum_{i=1}^{n} T_{loc,i}| \leq \sum_{i=1}^{n} |T_{loc,i}| \text{ (by Triangle inequality).}$$

From (6.10),

$$|T_{global}| \leq \sum_{i=1}^{n} |f''(\xi_i)|\frac{h^3}{12}.$$

We define $M = \max_{a\leq x\leq b} |f''(x)|$ and so obtain

$$|T_{global}| \leq nM\frac{h^3}{12}.$$

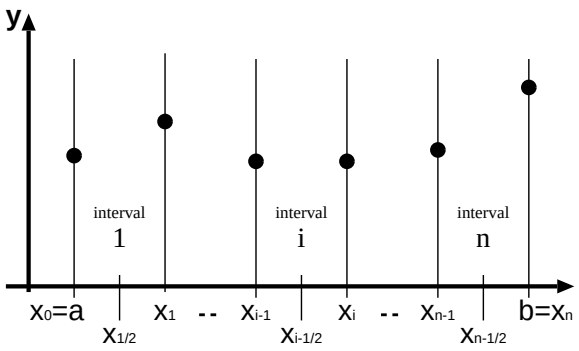
Substituting $n = \frac{b-a}{h}$ yields

$$|T_{global}| \leq (b-a)M\frac{h^2}{12}.$$

We conclude $T_{global} = O(h^2)$. □

### 6.2.2 Composite Simpson Rule

Recall that the Simpson rule is given by $\hat{I} = \frac{b-a}{6}(f(a) + 4f(\frac{a+b}{2}) + f(b))$. We define the midpoint of the subinterval $[x_{i-1}, x_i]$ as $x_{i-1/2} = \frac{x_{i-1}+x_i}{2}$ (see figure).




Using this definition, we may write the Simpson rule over one subinterval as

$$\hat{I}_i = \frac{h}{6}(f_{i-1} + 4f_{i-1/2} + f_i). \tag{6.12}$$

Summing over all intervals yields the expression for the composite Simpson rule:

$$\hat{I} = \sum_{i=1}^{n} \hat{I}_i = \frac{h}{6}[f_0 + 4f_{1/2} + 2f_1 + \cdots + 4f_{n-1/2} + f_n]. \tag{6.13}$$

Finally, we can devise a theorem for the global truncation error of this expansion:

**Theorem 6.2** *The global truncation error for the Simpson rule is $O(h^4)$.*

**Proof.** The proof is similar to the proof for the Trapezoid rule, except uses $T_{loc,i} = O(h^5)$. This is left as an exercise for the reader.

## 6.3 Gaussian Integration

Gaussian integration applies a different approach to integration than we have seen so far in this chapter. Consider the following example:

Assume we wish to determine the integral of a function $f(x)$ over the interval $[-1, 1]$. We write

$$I = \int_{-1}^{1} f(x)dx \tag{6.14}$$

and propose the following form of the numerical approximation:

$$\hat{I} = w_1 f(x_1) + w_2 f(x_2). \tag{6.15}$$

This expression has four unknowns (namely $w_1$, $w_2$, $x_1$ and $x_2$.) We want to determine these unknowns so that the degree of precision is maximal. Note that the location of function evaluation is now also a variable that will be determined optimally, in addition to the function weights.

Recall the degree of precision of an integration formula is the highest degree of polynomials that are integrated exactly. Since there are 4 unknowns in this problem, we assume we can require exact integration of polynomials up to degree 3. The four conditions imposed on the problem are then:

1. $f(x) = 1$ is integrated exactly.
2. $f(x) = x$ is integrated exactly.
3. $f(x) = x^2$ is integrated exactly.
4. $f(x) = x^3$ is integrated exactly.

(Note: these form a basis for all polynomials of degree $\leq 3$). Mathematically, these conditions are written as the following non-linear system in the four unknowns:

$$\begin{array}{lll} 1 & \int_{-1}^{1} 1 \cdot dx = w_1 + w_2 & \Rightarrow \quad \boxed{2 = w_1 + w_2} \\ 2 & \int_{-1}^{1} x \cdot dx = w_1 x_1 + w_2 x_2 & \Rightarrow \quad \boxed{0 = w_1 x_1 + w_2 x_2} \\ 3 & \int_{-1}^{1} x^2 \cdot dx = w_1 x_1^2 + w_2 x_2^2 & \Rightarrow \quad \boxed{2/3 = w_1 x_1^2 + w_2 x_2^2} \\ 4 & \int_{-1}^{1} x^3 \cdot dx = w_1 x_1^3 + w_2 x_2^3 & \Rightarrow \quad \boxed{0 = w_1 x_1^3 + w_2 x_2^3}. \end{array}$$

With some manipulation, we may solve the system for

$$x_1 = -\frac{1}{\sqrt{3}}, \qquad x_2 = \frac{1}{\sqrt{3}}, \qquad w_1 = 1, \qquad w_2 = 1.$$

Substituting these constants back into (6.15) yields

$$\hat{I} = 1 \cdot f\left(-\frac{1}{\sqrt{3}}\right) + 1 \cdot f\left(\frac{1}{\sqrt{3}}\right). \tag{6.16}$$

We conclude that this is an approximation for $\int_{-1}^{1} f(x)dx$ with degree of precision $m = 3$. By a change of integration variable, result (6.16) can be generalized as follows:

**Proposition 6.1 (Gaussian Integration.)**

$$\hat{I} = \frac{b-a}{2}\left[f((\tfrac{b-a}{2})(-\tfrac{1}{\sqrt{3}}) + (\tfrac{b+a}{2})) \; + f((\tfrac{b-a}{2})\tfrac{1}{\sqrt{3}} + (\tfrac{b+a}{2}))\right] \tag{6.17}$$

*is an approximation for* $I = \int_a^b f(x)dx$ *with degree of precision* $m = 3$.

**Proof.** We can express $x$ in terms of a new integration variable $t$ as follows:

$$x = a\frac{(1-t)}{2} + b\frac{(1+t)}{2}.$$

Note that $x = a$ when $t = -1$ and $x = b$ when $t = 1$. Also,

$$dx = \frac{b-a}{2}dt.$$

Using this substitution we get

$$\begin{aligned} I = \int_a^b f(x)dx &= \int_{-1}^{1} f((\tfrac{b-a}{2})t + \tfrac{b+a}{2})\tfrac{b-a}{2}dt \\ &= \tfrac{b-a}{2}\int_{-1}^{1} g(t)dt, \end{aligned}$$

with

$$g(t) = f\left((\tfrac{b-a}{2})t + (\tfrac{b+a}{2})\right).$$

Using (6.16) we obtain:

$$\hat{I} = \frac{b-a}{2}\left(g(-\tfrac{1}{\sqrt{3}}) + g(\tfrac{1}{\sqrt{3}})\right), \tag{6.18}$$

which leads to the desired result. □

**These notes have been funded by...**

# Appendix A

# Sample Midterm Exam

**Midterm Exam** **October 20, 2005**
**Instructor:** Professor H. De Sterck **Time:** 1.5 hours
**AIDS:** non-graphing calculator

---

1. (Errors and error propagation.) [20]

   (a) Give a brief explanation of the concept of 'condition of a mathematical problem'. Introduce the two relevant condition numbers and explain what they mean. When is a problem well-conditioned or ill-conditioned?

   (b) You are asked to calculate

   $$f(x) = \frac{1 - \cos x}{x}$$

   for $x = 0.01234567$ in the floating point system $F[b = 10, m = 4, e = 4]$ (use rounding). What is the relative error in $fl(x)$, and the relative error in the result $fl(f(x))$, if you calculate $f(x)$ using the straightforward algorithm given by the formula above? Explain. Find a different algorithm for approximating $f(x)$ that is more stable than the straightforward algorithm. Compare relative errors, and explain.

2. (Root finding.) [20]

   Consider the polynomial function $f(x) = x^2 + b\ x + b^2/4$ with $b > 0$.

   (a) Show that this function has a double root $x^*$, and locate it.

   (b) Apply the formula for Newton's method and find an expression for $x_{n+1}$ as a function of $x_n$.

(c) Determine $c_n = |x_{n+1} - x^*|/|x_n - x^*|$. What does this prove about the order of convergence? Does this contradict the convergence theorem for Newton's method? Explain.

(d) Can the Fixed Point Iteration method with $g(x) = x + f(x)$ be applied to this problem? If so, give the interval $I$ in which the initial value $x_0$ can be chosen such that convergence will occur. (Watch out, this is not an obvious question.)

3. (Root finding.) [20]

(a) Define 'contraction'. Illustrate with a graph and give one interpretation in terms of geometrical properties of the graph.

(b) Formulate the contraction mapping theorem, and prove its first part (existence and uniqueness of the fixed point).

4. (Numerical linear algebra.) [20]

Find the $LU$ decomposition of the following matrices. If necessary, determine a permutation matrix $P$ s.t. $PA = LU$.

(a)

$$A = \begin{bmatrix} 1 & 0 & 3 \\ -2 & 2 & -3 \\ 4 & -4 & 7 \end{bmatrix}.$$

(b)

$$A = \begin{bmatrix} 1 & 7 & 3 \\ 2 & 14 & 5 \\ 3 & 13 & 2 \end{bmatrix}.$$

5. (Numerical linear algebra.) [20]

(a) Describe an algorithm for calculating the determinant of an $n \times n$ matrix that has computational complexity $W = O(n^3)$ flops.

(b) Determine the number of floating point operations required for calculating the determinant of an $n \times n$ matrix using a straightforward implementation of the recursive definition of the determinant. (You can combine additions and multiplications. Hint: consider the number of flops done on each recursive level separately, and sum up over all the levels.) Find an approximation for the expression derived that is valid for large $n$. (Hint: this approximate result should be a very simple expression, and you will need $\exp x = 1 + x + x^2/2! + x^3/3! + x^4/4! + \ldots$)

# Appendix B

# Sample Final Exam

**Final Examination** **Wednesday, December 14, 2005**
**Instructor:** Professor H. De Sterck **Time:** 2.5 hours

---

1. (a) Define [15] machine epsilon of a floating point number system, and give an upper bound for the relative error $|\delta x|$ in the single precision floating point representation of a real number (assume chopping).

   (b) Give a short derivation of the Secant method (starting from a Taylor series expansion). Provide a geometrical interpretation for the Secant method.

2. (a) Find [10] the positive root of $f(x) = x^2 - 6$ using Newton's method with starting value $x_0 = 1$. Stop the iteration when the fifth digit does not change anymore in the result. (You can limit your calculations to numbers with 6 digits if you prefer to do so.)

   (b) If you were asked to find the positive root of $f(x)$ using the bisection method with initial interval $[1, 3]$, how many iterations would be required to make sure that the interval that contains the root has length smaller than $t = 0.0001$?

3. (a) Find the $LU$ decomposition of [20]

$$A = \begin{bmatrix} 3 & 3 & 3 \\ 6 & 8 & 8 \\ 6 & 10 & 11 \end{bmatrix}.$$

   (b) Consider the general form of an iterative method for solving $A\vec{x} = \vec{b}$, with $A$ non-singular. Show that if $\|I - B^{-1}A\|_p < 1$ for any $p$-norm, then the iterative method will converge to the solution for any starting value $\vec{x_0}$.

(c) Show that if $\|A\|_p < 1$ for any $p$-norm, then $I + A$ is non-singular. (Hint: assume that $I + A$ is singular, and demonstrate a contradiction.)

4. (a) Define frequency, period and angular frequency of a sine wave. [25] How are they related?

   (b) Explain how the formula for the DFT coefficients $F[k]$ of a time signal vector $\vec{f} = (f[0], f[1], \ldots, f[N-1])$ can be obtained using the general projection formula that is valid in any orthogonal basis. Briefly justify all steps in your reasoning.

   (c) Find the Fourier Series of

$$f(t) = \frac{1}{2}(\pi - |t|), \qquad t \in [-\pi, \pi].$$

   (Hint: note that $f(t)$ is an even function, and recall that $\int_a^b fg'dt = fg|_a^b - \int_a^b f'gdt$.)

   (d) Recall the general formula for calculating the length of a vector $\vec{x} = \sum_{n=0}^{N-1} x[n]\vec{e}_n$ in an N-dimensional vector space with basis $\{\vec{e}_0, \vec{e}_1, \ldots, \vec{e}_{N-1}\}$:

$$\|\vec{x}\| = \sqrt{<\vec{x}, \vec{x}>} = \sqrt{<\sum_n x[n]\vec{e}_n, \sum_l x[l]\vec{e}_l>}.$$

   Calculate the length of the time signal vector $\vec{f} = (f[0], f[1], \ldots, f[N-1])$ both using the expression for $\vec{f}$ in the time domain basis $\{\vec{f}_0, \vec{f}_1, \ldots, \vec{f}_{N-1}\}$, and the expression in the frequency domain basis $\{\vec{F}_0, \vec{F}_1, \ldots, \vec{F}_{N-1}\}$. The resulting length should of course be the same in both cases. Is it the same? What is the physical interpretation of this 'length' of time signal vector $\vec{f}$?

5. (a) Given [15] $f_0, f_0', f_1$ in points $x_0$ and $x_1$, determine the coefficients $a, b$ and $c$ of the interpolating polynomial $y(x) = a(x - x_0)^2 + b(x - x_0) + c$ that satisfies $y(x_0) = f_0, y'(x_0) = f_0'$ and $y(x_1) = f_1$.

   (b) Given $\{(x_i, f_i)\}_{i=0}^n$, you are asked to determine the coefficients $c_j$ such that $h_n(x) = \sum_{j=0}^n c_j \exp(j\,x)$ interpolates the data. Show that there is a unique solution for the coefficients $c_j$. You can assume that all the $x_i$ are different.

6. (a) Find [15] approximations for the integral

$$I = \int_0^4 \exp(\frac{1}{1+x})\, dx,$$

   using the midpoint rule, the trapezoid rule, and the Simpson rule. An accurate value for the integral is $I \approx 6.1056105$. Which of the three methods is the most accurate?

   (b) Given the general expression for the truncation error of the Simpson rule in interval $i$

$$T_{loc,i} = \frac{-h^5}{2880} f^{(4)}(\xi_i),$$

derive an upper bound for the global error $T_{global}$ of the composite Simpson rule for approximating $I = \int_a^b f(x)dx$ using $n$ subintervals of equal length $h$. What is the global order of accuracy of the composite Simpson rule?