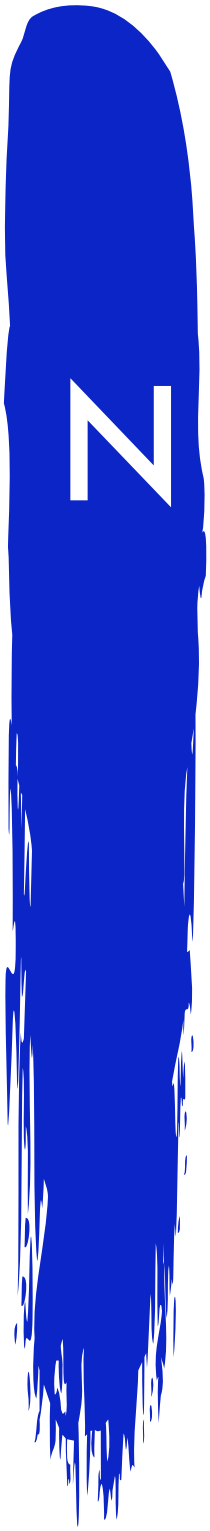


An Introduction to

Numerical Analysis

Siri Chongchitnan

2016/17

These notes accompany the course *32222: Introduction to Numerical Analysis* for second-year undergraduates reading mathematics at Hull.

The main textbook for this course is

> *Numerical Analysis*, Sauer T., 2nd ed., Pearson (2014).

In addition, I recommend the following books.

> *Numerical Mathematics and Computing*, Cheney W. and Kincaid D, 7th ed., Cengage Learning (2013).
>
> *Numerical Recipes: The Art of Scientific Computing*, Press W. H. *et. al.*, 3rd ed., Cambridge University Press (2007).

These books can all be found in the University Library.

You will also need to use MATLAB on your own computer. Send an email to `help@hull.ac.uk` to request the installation link. If you already have MATLAB at home, update it to the latest version.

It will certainly be useful for you to have access to MATLAB during lectures and tutorials. I encourage you to bring your laptops along. You can also install MATLAB on your phones or tablets.

Please send comments, questions and corrections to `s.chongchitnan@hull.ac.uk`.

Siri Chongchitnan
September 2016, Hull.

Things you should know for this course:

- Powers of 2 up to 2^{12} (write them out below)
- Basic MATLAB and MuPAD commands
- Important calculus theorems (*e.g.* MVT, IVT, Rolle's Theorem)
- Taylor's series and the remainder R_n
- Linear algebra (*e.g.* row reduction, determinant, inverse, eigenvalues, diagonalisation)

1 Computer Arithmetic

1.1 Introduction

We often take for granted that computers can make our lives easier by helping us perform complex mathematical calculations. In addition, we demand that these calculations should be done quickly and accurately, using as little resources as possible.

But how exactly does a computer achieve all this? How does a computer *compute*?

In this course, we will take a careful look at mathematics beyond pen and paper. We will study how to recast mathematical problems as numerical/logical tasks, or algorithms, that can be understood by computers, and how to optimise these algorithms for speed, accuracy and storage. These topics form the pillars of *numerical analysis*.

But first, in this chapter, we will study how computers work with numbers.

1.2 _____

A typical computer chip consists of millions of silicon transistors, each switching between two states, *charged* or *discharged*, in other words, *on* or *off*, *true* or *false*, 1 or 0. All mathematical problems must first be translated into only these two numbers. But how?

1.2.1 Integers

We are used to base−10 (_____) representation of integers, most likely because we have evolved to have 10 fingers. We don't often deal with everyday situations involving 10 or more things. For our ancestors, and even for us today, 10 can often seem like a *big* number.

Remember that we really don't have a symbol for '10': we combine symbols for 0 to 9 to represent anything bigger than 9.

Now imagine an alien with a the same basic hand structure as ours, but it only has one hand and two fingers, *i.e.* the hand only has three counting states as shown below.

In the alien's world, they don't often deal with two or more things, and the concept of 'two' seems such a *big* number to them. What symbols do you think would be sufficient for their counting needs?

Now try to think like an alien and imagine how you might create a system of counting based on these symbols, and fill in the table below.

Human system	Alien's system
0	
1	
2	
3	
4	
5	
6	

Human system	Alien's system
7	
8	
9	
10	
11	
12	
13	

The alien's system is called the _____, or _____ number system.

Each of the zero and one is called a _____, which stands for _____.
Here's another way to see the correspondence. Take our number 13:

$$\begin{aligned} &\textbf{Base 10}: \quad \mathbf{13} = \mathbf{1} \times 10^1 + \mathbf{3} \times 10^0. \\ &\textbf{Base 2}: \end{aligned} \tag{1.1}$$

(Now can you see why the word *base* is appropriate?) To prevent confusion, we will normally indicate the base of a number with the bracket notation:

$$(13)_{10} = \tag{1.2}$$

Example 1. Convert the following numbers to base 10. a) $(11011)_2$ b) $(1212)_3$.

Example 2. Find the binary representation of $(50)_{10}$. *Ans:*_____

Here is a systematic way to convert from base 10 to binary: Keep dividing by powers of 2 (to find the biggest power of 2 that would fit), keeping track of the remainder in each step until the result is 0. However, with some insight, you might not always need this algorithm.

Example 3. Convert to binary: a) $(50)_{10}$, b) $(121)_{10}$ c) $(130)_{10}$

Example 4. Without converting to base 10, write down the next 4 binary integers greater than 110010. Similarly, write down the 4 integers smaller.

The rules of binary addition and subtractions are the same as we know it, but now

$$1 + 1 =$$

1.2.2 Fractions

In analogy to the representation (1.1), can you guess what these binary numbers are in decimal?

$$(101.1)_2 =$$

$$(0.111)_2 =$$

The reverse conversion can be done as in Example 3, but now, instead of finding how many powers of 2 there are, for small numbers, we want to know how many powers of _____ there are. Hence, instead of dividing by 2 in each step, we *multiply* by 2 in each step, keeping track of the *integer part* and the *fractional part* in each stage.

Example 5. Convert to binary a) $(0.875)_{10}$, b) $(0.7)_{10}$.

Example 6. Using previous results, write down $(50.7)_{10}$ in binary.
Ans:

Example 5b shows that often we have to deal with recurring bits. Let's see how one can convert such a recurring binary number to a decimal representation.

Let's start with something you might remember from school.

Example 7. Everything is in base 10 in this question. Convert the following to fractions: a) $2.777\ldots$ b) $0.151515\ldots$ c) $0.234234234\ldots$

Note in the above examples that to shift the 'dot' by n places to the right, we multiply it by _____.

In base 2, the idea is the same, but to shift the dot in a binary number by n places to the right, we multiply it by _____.

Warning: Make sure you don't add or subtract numbers in different bases!

Example 8. Convert the following to base-10 fractions: a) $(0.\overline{1011})_2$ b) $(0.1\overline{0110})_2$.

1.3 Hex reps

Binary numbers are the building blocks of machine computations, but binary expressions are long and difficult for human to interpret. Instead, we can work with blocks of 4 bits, (*e.g.* $(0101)_2$, $(1111)_2$) and abbreviate each block with a single base _____ number.

The symbols needed for base-2 numbers are _____. The symbols needed for base-3 numbers are _____. The symbols needed for base-10 numbers are _____. You can see that for base 16 numbers (_____ numbers), we are going to need more symbols! These are:

$$0, 1, 2, 3, 4, 5, 6, 7, 8, 9, \_\_\_\_\_$$

For example, $(A)_{16} = (\_\_\_\_\_)_{10} = (\_\_\_\_\_)_2$. More about hex reps in the next section.

Example 9. Convert each block of bits to a single *hexadecimal representation*:

$$(0011)_2 = \_\_\_\_\_ \qquad (1100)_2 = \_\_\_\_\_ \qquad (1111)_2 = \_\_\_\_\_$$

Question. What's the next hexadecimal integer greater than F?

▶ **(MATLAB) Example 10.** What do you expect $(1)_{10}$ to correspond to in hexadecimal? Check this with MATLAB (change the display format to `hex`).

Your guess: _____. *MATLAB's answer:* _____.

1.4 Floats

In 1985, the Institute of Electrical and Electronics Engineers (IEEE) set out a framework for representing numbers in computers. The framework now widely used in computers is called the *IEEE Standard 754*[1]. We now take a look at this system in detail.

In base 10, we can write big numbers like 32222 in scientific notation as _____. The same idea can be done for binary numbers. For example, take $(9)_{10} = (\_\_\_\_\_)_2$. In the binary version of the scientific format, we can write

[1] See `http://ieeexplore.ieee.org/xpl/standards.jsp` for other IEEE standards.

$$(9)_{10} = +1.\boxed{001} \times 2^3,$$

where we see that such a number has 3 components:

- _____ (+ or −),
- _____ (the sequence of bits in the box),
- _____ (the power of 2). This will be stored as a binary integer in a special way — more later. It's common practice to leave the exponent in the float format as a base-10 number. Note that the exponent can be a negative integer.

When a number is represented in this way, it is called a _____ representation (or simply *float*). In this format. the leading number immediately after the sign is *always* 1 (this is called the _____ form of floats). There are special cases of unnormalised floats (for example, 0), more about these later.

Definition. The *floating-point representation* of a number x is given by

$$fl(x) = \ s1.\boxed{bbbbbb\ldots b} \times 2^c, \tag{1.3}$$

where

- the sign _____ is either + (0) or − (1),
- there are M bits for the mantissa _____.
- there are E bits for the exponent _____.

The greater the values M and E, the greater the accuracy and range of representable numbers (at the cost of memory).

Example 11. Represent these numbers in floating-point format: a) $(-50)_{10}$ b) $(0.7)_{10}$.

Note that in part (b) that even if a binary number does not terminate, $fl(x)$ must terminate because computers can only store a finite number of bits, thus there is a loss of precision going from x to $fl(x)$. This means that it's not possible to store numbers like 0.7 *exactly* on *any* binary-based computer (which is essentially all computers today).

1.5 Precision

Numbers can be stored on computers in different formats, or *precisions*. There are 3 commonly used types of precisions, with the following bit profiles.

Precision	Sign s	Mantissa M	Exponent E	Total bits
		23	8	
			11	64
		64		80

Table 1.1: Number of bits for 3 commonly used IEEE-standard precisions.

We will mainly be working with *double*-precision numbers. MATLAB uses *double* in all its calculations (although it can display in shorter-looking formats).

Example 12. Display the number 1 as a *double-precision* float.

Example 13. Find the floating-point rep of x, which is the smallest floating point-number greater than 1 that can be represented in double precision.

Write down the numerical value of $x - 1$.

Definition. The _____, $\varepsilon_{\text{mach}}$, is the distance between 1 and the smallest floating point number greater than 1. For double-precision floats,

$$\varepsilon_{\text{mach}} =$$

You might be wondering if $\varepsilon_{\text{mach}}$ is the smallest number representable as a float. Actually, there are lots of numbers smaller than $\varepsilon_{\text{mach}}$ which can be represented as floats (as we will see later), although adding such a tiny number to 1 will still give you *exactly* 1.

1.6 Error

In this section, we will show that $\varepsilon_{\text{mach}}$ is roughly the error when x is stored as $fl(x)$.

1.6.1 Types of errors

Suppose we have two quantities: *Actual* and *Estimate*. There are a few ways of quantifying the error of the estimate.

Absolute error: (1.4)

Fractional (or relative) error: (1.5)

Percentage error: (1.6)

1.6.2 Rounding-to-Nearest rules

What does the computer do with numbers like $(0.\overline{011})_2$, which cannot be represented exactly as a *double float*? In the IEEE standard, the following _____ rules are applied:

I. If bit 53 of the mantissa is 1 (and bits 54 onwards are not all zeroes), round it

Example: $\boxed{\ldots 100100}1001\ldots \rightarrow$

II. If bit 53 of the mantissa is 0 (and bits 54 onwards are not all zeroes), round it

Example: $\boxed{\ldots 010010}0100\ldots \rightarrow$

III. If bits 53 onwards are exactly $10000\ldots$ (*i.e. exactly halfway* between up and down), then round it such that bit 52 is 0.

Example: $\boxed{\ldots 010010}10000\ldots \rightarrow$

$\boxed{\ldots 101101}10000\ldots \rightarrow$

Using these rules, we can now compare the difference between x and $fl(x)$.

Example 14. Find $\left(\frac{4}{3}\right)_{10}$ in binary. Explain how it can be represented as a double-precision floating-point number.

Example 15. Show that fractional error associated with the rounding in the previous example is given by

$$\left|\frac{fl(x) - x}{x}\right| = \frac{1}{4}\varepsilon_{\text{mach}}, \tag{1.7}$$

where $x = \left(\frac{4}{3}\right)_{10}$.

In fact, we can show that under the IEEE standard, we have...

Theorem 1.1. [*Floating-point Rep Theorem*] The fractional error in representing a number x as a floating point $fl(x)$ satisfies

$$\leq$$

We won't prove it here, but you can roughly see why this is the case. From the RTN rules, rounding up or down will change x by at most 2^{-53} (the first bit out of the box).

This theorem will be important when we come to numerical calculus in Chapter 5.

1.7 Machine representation

Let's see how exactly double floats are stored in computers.

Recall from Table 1.1 that each *double* is assigned _____ bits, comprising _____ for the sign, _____ for the exponent and _____ for the mantissa. These bits are all joined into a single long string of the form

$$s\,\mathbf{e_1 e_2}\ldots\mathbf{e_{11}}\,b_1 b_2 \ldots b_{52}. \tag{1.8}$$

Note the ordering. The exponent comes *before* the mantissa. We've already dealt with the sign s (0 for _____) and the mantissa in the previous section. The exponent is the integer c in

$$\pm 1.\boxed{\ldots} \times 2^c$$

where c can be positive or negative. You might think it is easy enough to store c in the usual binary representation, but this means we need to take care of the sign of c as well, which could potentially be confusing.

Since 11 bits are assigned to the exponent, This means we can store integers from 0 up to _____. We want half of these to represent negative exponents, so roughly we only expect to cover exponents up to ±_____.

That's the rough idea, but the actual rule is a bit more fiddly: Given an exponent (positive or negative), we can store it as a *positive* integer by the following adjustment:

Actual exponent (base 10)	-1022	-1021	$\ldots$	-1	0	1	$\ldots$	1022	1023
Biased exponent (base 10)									

The rule is: given an exponent, _____, then store the binary form as $\mathbf{e_1 e_2 \ldots e_{11}}$. This is called the _____ exponent.

The extreme biased exponents **0 and 2047 are reserved for special types of numbers**, as we will see shortly.

The number 1023, is called the _____ for double floats[2]. Note: A helpful binary to remember: $(1023)_{10} =$

Example 16. Find the machine representation of 1 as a double-precision float. Give your answer in binary form.

As you can see, writing down one of these binary rep by hand is quite painful, and extremely difficult for human to interpret. This is where hex reps come in.

For human readability, we can abbreviate the machine rep by converting each block of 4 bits into one hexadecimal number. This abbreviation means that we only need to deal with _____ numbers per double float.

Example 17. Find the machine representation of 1 in hex format.

[2]The exponential bias for *single* and *long double* floats are 127 and 16383 respectively.

Warning: Computers do not do calculations in base 16!. The hex reps are used purely for humans to read codes more easily.

Example 18. Determine the machine representation of $\left(\frac{1}{3}\right)_{10}$ as a double float in hex format. Check your answer with MATLAB.
Hence, write down the hex representation of $\left(-\frac{1}{3}\right)_{10}$

You might be wondering why we write the biased exponent *before* the mantissa in machine reps (1.8). Well, if you line up positive numbers in increasing order of magnitude, then the machine reps also increases in the same order. Had we written the mantissa before the exponent, then the machine rep of 1 would appear before, say, that of $0.75=(1.1 \times 2^{-1})_2$.

Example 19. What is the magnitude of the smallest positive normalised number representable as a double float? Find its hex rep.
Check your answer in MATLAB by typing `realmin`.
[Similarly, we can find the largest double float `realmax`. See Problem Sheet 1.]

1.8 Special numbers

Let's see the special cases when the biased exponent is 0 or 2047.
Note: Another helpful binary: $(2047)_{10} =$

1.8.1 Inf

The biased exponent is 2047 is reserved for an 'effectively' infinite quantity. This includes genuine infinity *e.g.* when evaluating _____, and also numbers much[3] larger than the largest representable number `realmax`. These are all given the symbol _____ (or _____ if negative), with the following machine rep:

`Inf` = (1.9)

(*i.e.* the exponent is maxed out, everything else is 0). In hex format:

`Inf` = (1.10)

▶ **(MATLAB) Example 20.** (a) Read MATLAB's help on "`Inf`" (type `doc inf`).

(b) Write down a few quantities that would be assigned to $-$`Inf`.

(c) Find the hex rep of $-$`Inf`. Check with MATLAB.

(d) Predict what sin(`Inf`) would produce.

1.8.2 NaN

This is another naughty quantity, which results from expressions that cannot be interpreted as a unique limit, *e.g.* something like

$0/0$, `Inf/Inf`,

The result is called a _____ ('Not a Number').

[3] Numbers 'a bit' larger than `realmax` would be rounded down by RTN and represented as `realmax`.

The machine rep of a `NaN` is almost the same as that of `Inf`, except the bits in the mantissa are not all zeroes. The exact representation depends on the processor, but in MATLAB, the convention is to take the machine rep of `Inf` and change the bits to the immediate left and right of the exponent to 1. This means that the machine rep and hex rep of `NaN` are:

(1.11)

1.8.3 Subnormals

On the opposite extreme, the minimum biased exponent 0 is reserved for tiny numbers. In this case, the number is no longer assumed to be of the *normalised* form $1.\boxed{\ldots}\times 2^p$, but instead, the following unnormalised form:

$$\pm\mathbf{0}.\boxed{b_1 b_2 \ldots b_{52}} \times 2^{-1022}. \tag{1.12}$$

(note the exponent!). This kind of tiny, unnormalised float is called a _____.

Example 21. What is the magnitude of the smallest positive number representable as a double float? (let's call it `tiny`.) Find its machine rep in hex format and check with MATLAB.

Note that according to a computer operating at double precision, `tiny` is the smallest positive number in the Universe. There is *nothing* between zero and `tiny`, for the same reason there is nothing between 1 and _____

The figure below summarises the differences between `tiny`, `realmin` and $\varepsilon_{\text{mach}}$.

Note the _____ distribution of real numbers, according to computers. Without infinite storage, it's impossible for binary-based computers to understand the real line $\mathbb{R}$.

We see from the figure that the IEEE standard endows a very dense, high resolution environment around zero, which gradually spreads out, becoming less resolved the further you move away from zero. This is because in machine representation, the number of significant figures is fixed - so when you try to represent larger and larger numbers, the last significant figure corresponds to increasingly large numbers.

1.8.4 Zero

Finally, there's the most important subnormal number of all: zero. Its exponent and mantissa comprise entirely of 0 bits, though it can carry either signs depending on the calculation (for example, _____ and _____ have opposite signs). MATLAB treats $+0$ and -0 as equal; Try using `==` to test their equivalence in MATLAB.

Example 22. Find the machine reps for ± 0 in hex format.

▶ **(MATLAB) Example 23.** (Strange but true.) Use the idea of ± 0 to construct a counter-example to the following universally accepted statement.

$$\text{If } x = y, \text{ then } f(x) = f(y).$$

1.9 Float addition

▶ **(MATLAB) Example 24.** This is another MATLAB mystery. Let's define the following double-precision variables.

$$x = \frac{4}{3}, \qquad y = x - 1, \qquad z = y - \frac{1}{3}.$$

What is the correct value of z? What happens when you do this in MATLAB?

We shall try to understand this result using what we've learnt so far.

Example 25. Show that in the above working, the machine rep of z is $-\frac{1}{4}\varepsilon_{\text{mach}}$.

You can see that the main culprit is the _____.
The extra zeros supplied by the computer during the subtraction are called _____.

The RTN rules and spurious zeros are prime suspects when we see such surprising results on MATLAB. Any subtraction involving numbers not *exactly* representable as binary numbers will suffer from this problem. In long calculations, this effect could accumulate to give you a result that's far from what you expect.

This not only happens in MATLAB, but in all computers under the IEEE standard.

► **(MATLAB) Example 26.** Experiment with MATLAB and write down more 'surprising' results such as the one above.

Although such 'errors' cannot be eliminated completely, keep in mind that they are always predictable, and thus we can always work out precisely how accurate each computer calculation is. An important element of numerical analysis is to study how such tiny $\varepsilon_{\text{mach}}$-size errors propagate through long calculations, and how to avoid error amplification and minimize the total error.

1.10 Loss of significance

We end this chapter with another important difference between computer mathematics and 'real' mathematics, best illustrated through the following example.

► **(MATLAB) Example 27.** Consider two functions:

$$f(x) = \sqrt{x^2 + \alpha} - x, \qquad g(x) = \frac{\alpha}{\sqrt{x^2 + \alpha} + x} \qquad (\alpha > 0). \tag{1.13}$$

(a) Verify that $f(x) = g(x)$.

(b) Let $\alpha = 10^{-13}$. Use MATLAB to find the values of the functions when $x = 1, 10, 50$. Which expression gives a more accurate answer?

	$f(x)$	$g(x)$
$x = 1$		
10		
50		

Can you see where the problem lies?

It is actually the same problems which gave us the strange results in the last section: the rounding-to-nearest rules, and the introduction of spurious zeroes during _____.

To give a simpler analogy, suppose that instead of the usual 52 bit mantissa, our cheap computer can only store up to 3 significant digits. Now suppose we try to perform the subtraction below:

$$\sqrt{4.01} - 2.00$$

Although $\sqrt{4.01} = 2.002498\ldots$, according to our cheap computer, $\sqrt{4.01} =$ _____, so $\sqrt{4.01} - 2.00 =$ _____. We started with two numbers with 3 significant figures, but the result contains fewer significant figures. This problem is known as _____.

Similarly, in Example 27, the mantissa for $\sqrt{x^2 + \alpha}$ and x are identical except for the tail bits. The subtraction causes a flood of many spurious zeroes - hence the loss of significance.

Moral of the story: As far as possible, you must

It is sometimes difficult to see where such a subtraction might occur. Anticipation will come with experience. The rule of thumb is to try to turn potentially dangerous subtractions into additions and/or divisions by using algebraic identities or power series.

Example 28. Why could there potentially be loss of significance when each of the following expressions is calculated? How can they be avoided?

i) $1 - \cos x$, ii) $\ln x - 1$.

► **(MATLAB) Example 29.** Solve the equation for x (correct to 4 significant figures)

$$x^2 + 9^{12}x - 3 = 0.$$

MATLAB would have a hard time solving this equation, even with built-in functions. However, try the `solve` function in MuPad (see last year's worksheet).

Example 30. Evaluate the function

$$f(x) = e^x - x - 1$$

at $x = 10^{-9}$, correct to two significant figures.

2

Root finding

In this Chapter, we will study a number of *root-finding* methods, and by roots, we mean the solution(s) to

$$= \tag{2.1}$$

Root-finding is important because any equation can be put into the form $f(x) = 0$. For example, to solve $x^3 = 1 - x$, we solve for the root(s) of _____. We will be focussing on equations which are difficult or impossible to solve analytically. We shall see how to attack them numerically, not just on MATLAB but on any computer.

Here's a little preview: MATLAB actually has a built-in root-finder called `fzero`, which you can test with the command:

```
fzero(@(x) x^3+x-1, [0,1])
```

By the end of the Chapter, you will understand what MATLAB does when we issue this command. But first, let's look at a technique which will be useful when dealing with polynomials, and help you get back into the MATLAB programming.

2.1 Polynomial evaluation

Consider the polynomial $P(x) = 2x^4 + 3x^3 - 3x^2 + 5x - 1$. What is the most efficient way to evaluate $P(\frac{1}{2})$ on the computer?

Assume that we have stored $\frac{1}{2}$ in the memory. Of course we could do this in the most straightforward way.

This requires _____ multiplies and _____ adds. More cleverly, we can reduce the number of operations by storing the successive powers of $\frac{1}{2}$.

This requires _____ multiplies and _____ adds.

Better yet, we could create *layers* of simple functions:

This only requires _____ multiplies and _____ adds. It can be shown that this is the minimum number of operations achievable. This is called the method of _____ (or *Horner's method*[1]).

Saving a few microseconds on polynomial evaluations may not seem much, but in more complex tasks, we may need to evaluate high-order polynomials at hundreds of points. The time and resources saved can be better spent on more useful tasks.

Always avoid redundant calculations!

[1] *William Horner* (1786-1837), British mathematician, published this result in 1819, although the technique was known to a number of previous mathematicians.

Example 1. (do this by hand) Evaluate $P(x) = 6x^3 - 2x^2 - 3x + 7$ at $x = 0.5$ using nested multiplication. How many multiplies and adds do you need?

Example 2. What is the minimum number of multiplies and adds needed to evaluate a general polynomial degree n

$$P(x) = a_0 + a_1 x + a_2 x^2 \ldots + a_n x^n, \tag{2.2}$$

at $x = x_0$ using nested multiplication?

In fact, it can be shown that nested multiplication is *optimal*, *i.e.* no other techniques can be more efficient[2].

[2]However, it is possible to 'cheat' by recasting the polynomial in a different form (*preconditioning*) and beat down the number of multiplications further, at the cost of computing and storing new coefficients, *e.g.* it's possible to evaluate any 6th-degree polynomial in 4 multiplies and 7 adds.

▶ **(MATLAB) Example 3.** Create a MATLAB function `nest(a,x)` which takes two arguments:

- `a` is the vector of polynomial coefficients (constant first) $[a_0, a_1, a_2, \ldots a_n]$.
- `x` is the value at which to evaluate the polynomial.

and computes $a_0 + a_1x + a_2x^2 \ldots + a_nx^n$ using nested multiplication.

If you have done this right, you should find that, for example,

```
nest([7 -3 -2 6], 0.5)
```

gives the answer to Example 1

Tips and warnings

- Dig up MATLAB notes from last year.
- Annotate your code (using %) so that others can understand your thought process.
- Try a `for` loop (though there are other ways).
- Careful: The index of a MATLAB array starts from 1. For example, if `a=[4 7]`, then `a(1)=4`.
- Test, test and test your code. Codes that give the wrong answer get at most 50%.
- Codes that don't run get at most 40%.
- Codes copied wholesale from the internet or from one another will get 0%.

Save your code as `nest.m` and upload it onto *Canvas* by 2359hr, Thursday 10$^{\text{th}}$ November.

2.2 Bisection method

Example 4. Sketch the graph of $f(x) = x^3 + x - 1$.

Our preliminary investigation reveals that $x^3 + x - 1 = 0$ has only 1 real root. Note that $f(0) < 0$ and $f(1) > 0$. Since f is continuous, $\exists c \in (0, 1)$ such that _____ by the _____. We say that the root is _____ by 0 and 1.

What would then be your first guess for the root? One could first try $x_0 =$ _____. We find that $f(x_0)$_____, so it's not quite right. But what this shows is that we can narrow down our search to the interval _____, again by the virtue of the _____.

We can then repeat, or _____, our method to find new bracketing intervals that become as small as desired. This is the idea behind the *bisection method*.

Example 5. (Calculator only) Estimate the real root of $x^3 + x - 1 = 0$ using 3 iterations of bisection (Note: the first bisection is the 0th iteration. We want to find x_3).

Note that we only need the sign of $f(x_i)$, not its value, so we can save some writing.

$f(a = 0) < 0$ $\qquad$ $f(b = 1) > 0$ $\qquad$ $f(x_0 = 0.5) < 0$

In general, starting with the interval $[a, b]$, after n iterations of bisection, we know that the root lies somewhere in $[a_n, b_n]$, so our best estimate for the root is

$$x_n = \qquad \text{with error at most } \pm$$

We can simplify the error (call it e_n) a bit by noting that the length of the interval is halved with each new iteration:

$$b - a =$$

Corollary. For the bisection method starting with the interval $[a, b]$, the error of the nth iterate, x_n, satisfies the bound

(2.3)

Each iteration of bisection is *guaranteed* to halve the error. In fact, bisection is the only root-finding method with a risk-free guaranteed result (provided the root exists).

We will be meeting the error quite often, so let's define it properly:

Definition. The *error*, e_n, is the absolute difference between the actual root c, and the estimate x_n, obtained after n iteration of a root finding method, *i.e.*

$$= \tag{2.4}$$

Example 6. Let $\varepsilon > 0$ be an arbitrarily small number. Find the number of iterations, n, which will guarantee that the error from the bisection method, starting from the interval $[a, b]$, satisfies $e_n \leq \varepsilon$.

Definition. An estimate is said to be *correct to p decimal places* if $e_n < 0.5 \times 10^{-p}$.

Example 7. In Example 5, how many iterations would it take so that the estimate is guaranteed to be correct to i) 3 decimal places? ii) 8 decimal places?

▶ **(MATLAB) Example 8.** Let's now have a little help from MATLAB. Write a MATLAB function which evaluates $f(x) = x^3 + x - 1$. Continue the bisection in Example 5 until the root is guaranteed to be accurate to 3 decimal places.
[*Tip: To create a quick function without an M-file, look up "anonymous functions".*]

Example 9. Demonstrate graphically that there is only one solution to the equation

$$e^{-x} = x - 1.$$

Find the integer N such that the solution lies in the interval $[N, N + 1]$.
Hence, estimate the root using the bisection method with 4 iterations to obtain x_4.
Estimate the error of your answer.

Does bisection always work?

Note that for the bisection algorithm to start, we need to find an interval $[a, b]$ in which $f(a)$ and $f(b)$ have opposite signs, in other words,

$$f(a)f(b)$$

But this may not be possible. Take $f(x) = x^2$ for example. Think of other examples.

Even if bisection can start, it may not converge to the correct answer(s).

These examples show that we cannot simply use root-finding routines as magical 'black boxes' without understanding something about the function.

Computing is not about crunching numbers. Without mathematical insight, what we obtain will be meaningless.

2.3 Convergence

You might assume that the more iterations you do, the closer your estimate will be to the actual root. However, this may not be the case if the function is not 'well-behaved', as we saw on the previous page.

Even if the estimate does get closer and closer to the root, it may happen slowly for low-tech root-finders (like the bisection method), whilst more sophisticated methods (like `fzero`) might be faster at getting to the root. In any case, it would be good to have a way of measuring how quickly the estimate gets closer to the actual answer. This is the idea behind *convergence*.

Definition. An iterative method is said to *converge with order* k if

$$\lim_{n\to\infty} \qquad\qquad = M. \tag{2.5}$$

The constant M is called the _____.

If $k = 1$, the method is said to converge _____. The limit with $k = 1$ reminds us of the _____ from *Numbers, Sequences and Series*. For this reason, we require that _____ if $k = 1$.

If $k > 1$, the method is said to converge _____. In this case the only requirement on M is that it is *finite*. The bigger the order k, the *faster* the search converges to an answer (can you see why?).

Of course we won't know exactly what the error e_n is (otherwise you would also know the root exactly). Thus, we can only use some estimate of e_n (to the best of our knowledge) in (2.5). Using an upper bound of e_n in (2.5) will give a conservative ('worst-case scenario') estimate for k .

Example 10. Prove that for the bisection method converges linearly.

► **(MATLAB) Example 11.** Below is a MATLAB M-file for the bisection method.

```
1  function x=bisect(f,a,b,err)
2  fa = f(a);
3  fb = f(b);
4  if fa*fb>=0
5          error('Root is not bracketed.')
6  end
7  while (b-a)/2>err
8          x=(a+b)/2;
9          fx = f(x);
10           if fx==0
11              break
12           end
13           if fx*fa<0
14              b=x;
15              fb=fx;
16           else
17              a=x;
18              fa=fx;
19           end
20 end
21 x=(a+b)/2;
```

(a) Copy and save the above code as `bisect.m` in your computer/G-drive. Annotate the code (using %) thoroughly, demonstrating your understanding of the code.

(b) Modify the code so that after each iteration, the code displays the iteration number, n, and the estimate, x_n.

On Canvas, upload your modified M-file, and answer the following questions.

1. Why it is a good idea to define new variables in lines 2, 3 and 9 (as opposed to using $f(a)$, $f(b)$, $f(x)$ directly where needed)?

2. Give the command(s) needed to use the above code to find the root of $x^3 + x - 1$ to 8 decimal places. Write down the final answer obtained.

3. Are there any lines in the code above that are redundant? If so, which one(s)?

2.4 When to stop?

For iterative methods like bisection, we need to impose a _____, or your code will run forever. Ideally, we want the routine to stop when the estimate, x_n, is sufficiently close to the root, c, *i.e.* when the error $e_n = |x_n - c|$ is smaller than some given _____, ε (as in Example 6). However, some root-finding methods do not narrow down the interval containing c in such a systematic way, and so it's difficult to estimate e_n directly. So, when should we tell the root finder to stop?

(A) *Stop after N iterations.*

This is not a great idea in principle. Why?

However, it is still desirable to set the limit on the maximum number of iterations allowed. Why?

(B) *Stop when $|f(x_n)| < \varepsilon$.*

This may seem sensible since, after all, we are solving $f(x) = 0$, so we want the LHS to be as small as possible. But ...

(C) *Stop when $|x_{n+1} - x_n| < \varepsilon$.*

This gives a good idea of the improvement in having done one more iteration.

In practice, it is common to use stopping criterion (C) along with (A) as a safety net. We will implement these in the next Section.

Here is a schematic of two ways in which we can implement criteria (A) and (C) together in MATLAB.

2.5 Newton-Raphson method

This method was invented by Isaac Newton (1671) and independently by the English mathematician Joseph Raphson (1690). It is best explained with the following figure.

This method is based on the observation that the tangent lines near the root will also intersect the x axis near the root.

Here is the work flow. We start with an initial guess, x_0 close to the root.

1. Draw a tangent line at x_0.
2. Find the x-intercept of the tangent line. This is the new estimate of the root x_1.
3. Repeat the above steps to obtain x_2, x_3 *etc.* until a stopping criterion kicks in.

Example 12. Let f be a differentiable function on $\mathbb{R}$. Find the x-intercept (call it x_1) of the tangent line drawn at $x = x_0$.

Having found x_1 as the new guess, we can obtain x_2 with the same formula

$$x_2 = \tag{2.6}$$

and so on. Hence we have the following iterative scheme for the Newton-Raphson method.

x_0 = initial guess,

x_{n+1} =

When working with a calculator, try to store exact results as far as possible. Even a small round-off error can snowball to give you a rubbish final answer. If storing is not possible, at least keep one or two more decimal places than the required accuracy.

Another tip: simplifying the Newton-Raphson formula (if possible) before substituting in numbers will save a lot of time.

Example 13. (Calculator only) Starting with $x_0 = 0.5$, use 3 iterations of the Newton-Raphson method to obtain the estimate, x_3, of solution to $x^3 + x - 1 = 0$, giving your answer to 4 decimal places.

If the method converges, we will see that increasing the number of iterations produces little changes. We will deal with the convergence a bit later.

Example 14. I went to Poundland and bought a calculator which can only do addition, subtraction, multiplication and division. Ambitiously, I want to use it to calculate $\sqrt{3}$, correct to 3 decimal places. What can I do?

Example 15. Sadly, the division button on my cheap calculator is now broken. How can I use it to calculate $1/1.37$, correct to 4 decimal places?

▶ **(MATLAB) Example 16.** Create a MATLAB function

```
newton(f, df, x0, tol, Nmax)
```

which finds a root of $f(x) = 0$ using the Newton-Raphson method, where the input arguments are as follows.

- `f` is the function.
- `df` the derivative of f.
- `x0` is the initial guess.
- `tol` is tolerance, such that the code stops when $|x_{n+1} - x_n| <$ `tol` [see Criterion (C) in §2.4].
- `Nmax` is the maximum number of iterations allowed [Criterion (A)].

Your code should be:

- well-annotated to show how you break the problem down into smaller tasks. Imagine if you were to send your code to one of your classmates. You should give enough annotation for them to be able to interpret what you're trying to.
- displaying some useful information to the user as it runs (*e.g.* iteration number, current estimate, *etc.*).
- well-tested (*e.g.* see if it gives the correct answers to the Examples in this Chapter).

Submit your M-file onto Canvas.

With a bit of experience, you will find that the Newton-Raphson method often has a shaky start, with estimates going all over the place. But once converge starts to take hold, the estimates converge very rapidly (much faster than the bisection method). The following Theorem explains why.

Theorem 2.1. Let f be an infinitely differentiable function with $f(c) = 0$ and $f'(c) \neq 0$. If the Newton-Raphson method converges, then it converges *quadratically*, *i.e.*

$$= \tag{2.7}$$

where $e_n \equiv |x_n - c|$.

Proof. Recall the Taylor expansion for $f(x)$ around $x = x_n$, to linear order, with Lagrange's form of the remainder $R_2(x)$ from your Calculus course:

$$f(x) = \tag{2.8}$$

where $\xi \in$

Now set $x = c$ (the root of $f(x) = 0$) and rearrange:

Since ξ is a number between x_n and c, taking the limit as $n \to \infty$, we find that $x_n \to$ (assuming the estimates converge), and hence $\xi \to$ _____ by _____.
Hence,

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^2} =$$

For the quadratic convergence to take hold, it is essential that $f'(c) \neq 0$. Let's see what happens when this condition is not satisfied.

Example 17. Suppose we use the Newton-Raphson method to solve $x^2 = 0$, with initial guess $x_0 \neq 0$. Find the rate and order of convergence.

This shows a typical behaviour of the Newton-Raphson method at multiple roots (*i.e.* where $f(c) = f'(c) = 0$). In fact, one can show that if a root has multiplicity > 1, then the Newton-Raphson only converges linearly. There are ways to fix this, however: look up *"modified Newton's method"* if you're interested.

Example 18. Sketch some situations in which the Newton-Raphson method does not converge

2.6 Secant method

Newton's method converges faster than the bisection method because it relies on an extra piece of information: the expression for _____. But what can we do when the derivative is difficult to obtain?

The idea is to replace the tangent by a *secant*, which is just a straight line joining two points, x_0 and x_1, which will be our initial guesses, as shown in the figure below.

Note that the initial guesses do not have to bracket the root.

The work scheme for this method is as follows.

1. Draw a line joining x_0 and x_1.
2. Find the x-intercept of this line. This is the new estimate of the root x_2.
3. Repeat Step 1 using _____ and _____ to obtain x_3 *etc.* until a stopping criterion kicks in.

Example 19. Find the x-intercept of the line joining $(x_0, f(x_0))$ and $(x_1, f(x_1))$.

Having found x_2 as the new guess, we can obtain x_3 with the formula

$$x_3 = \tag{2.9}$$

and so on. Hence we have the following iterative scheme for the secant method.

x_0, x_1	=	initial guess,
x_{n+1}	=	

As you can see, the secant formula is simply an approximate version of the the Newton-Raphson formula, where the derivative is approximated by the difference quotient:

$$f'(x_n) \approx$$

Example 20. (Calculator only) Starting with $x_0 = 0$ and $x_1 = 1$, Use the secant method to obtain the estimate, x_4, for the root of $x^3 + x - 1 = 0$. Give your answer to 3 decimal places.

I like to call the secant method is the "poor man's' Newton-Raphson" (make do when we don't have derivatives).

Now let's look at the order of convergence of the secant method.

Question. (guess) Rank the bisection, Newton, and secant methods in increasing order of convergence.

Theorem 2.2. Suppose the estimates obtained from the secant method converge to the root c, with order $k > 0$, *i.e.*

$$\lim_{n\to\infty} \frac{e_{n+1}}{e_n^k} = C < \infty, \tag{2.10}$$

then $k =$ _____.
In other words, the secant method converges _____.

Proof. (informal) Assume that the convergence behaviour becomes apparent after a large number of iterations, say, $n \in \mathbb{N}$.. This means that there exist k (the order of convergence) and C (the rate of convergence) such that

(2.11)

Here we have used the approximation symbol $\approx$ quite loosely for now.

Inspired by the fact that the Newton's method converges quadratically, and the fact that each new iterate depends on *two* previous ones, we guess that there should also be a constant M such that

(2.12)

□

The proof could have been made more rigorous by dealing with inequalities rather than $\approx$, but it is a bit more tedious. I believe the heuristic proof gives us a flavour of why the Golden Ratio appears.

2.7 Comparison of methods

	Bisection	Newton-Raphson	Secant
Initial guesses			
Order of convergence			
Pros			
Cons			

3

Linear Systems

By now, you would have had plenty of exposure to linear algebra from a theoretical point of view. As usual, we will be discussing solutions to the equation

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

where A is a square matrix of numbers, $\mathbf{b}$ is a column vector of numbers, and $\mathbf{x}$ is a column vector of unknowns.

You might be tempted to write the solution as $\mathbf{x} =$ _____. However, in real applications, inverting matrices involves a large number of operations (roughly _____ $+$ and $\times$ are needed to invert an $n \times n$ matrix), and is notoriously slow for large matrices. Besides, performing more operations means more opportunities for errors to snowball.

In MATLAB, this equation is most efficiently solved using the command

$$= \tag{3.1}$$

Note the direction of the *backslash* operator \

In this Chapter, we will study ways in which linear systems can be solved efficiently on a computer. By the end of the Chapter, we will understand what exactly the backslash operator does.

▶ **(MATLAB) Example 1.** Let's try to solve $A\mathbf{x} = \mathbf{b}$, where A is a 3×3 matrix of random numbers between 0 and 1. The MATLAB command to generate A is

```
rand(3)
```

Let $\mathbf{b} = (1, 1, 1)^T$.

(a) Solve the system using the backslash operator.

(b) Solve the system using matrix inverse.

The assignment is to compare the time taken to solve the system using the 2 methods as the dimension of the problem increases from 3 to 5000. Which method is faster?

Convey this information in a graph. Imagine you have a friend who stubbornly believes that the matrix-inverse method is a good method to solve this problem. You should aim to produce a graph that will decisively convince them to change their mind.

Hint: Use the `tic` and `toc` commands to time your code.

Upload two things onto Canvas.

- the M-file used to generate the data for the graph.
- your graph **in pdf**.

Be creative.

3.1 LU factorization

Let's start with a warm-up exercise.

Example 2. Solve the system

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 3 \\ 2x_1 + x_2 - 2x_3 &= 3 \\ -3x_1 + x_2 + x_3 &= -6 \end{aligned} \tag{3.2}$$

by Gaussian elimination to *row-echelon* form.

Example 3. Describe what happens when a 3×3 matrix A is pre-multiplied by

$$\begin{pmatrix} 1 & & \\ \alpha & 1 & \\ & & 1 \end{pmatrix} \text{ (where blanks are zeroes).}$$

Hence, write down matrices L_1, L_2 and L_3 such that

$$L_3 L_2 L_1 \begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -2 \end{pmatrix}.$$

Example 4. Write down the inverse of $\begin{pmatrix} 1 & & \\ \alpha & 1 & \\ & & 1 \end{pmatrix}$. Hence, write down $L_1^{-1}, L_2^{-1}, L_3^{-1}$.

Hence, we see that the matrix in Example 2 can be written as a product of matrices:

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

Now note the following matrix identity (the ordering is important).

$$\begin{pmatrix} 1 & & \\ \alpha_1 & 1 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ \alpha_2 & & 1 \end{pmatrix} \begin{pmatrix} 1 & & \\ & 1 & \\ & \alpha_3 & 1 \end{pmatrix} = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix}. \tag{3.3}$$

Thus, we see that A can be written as a product of two 'triangular' matrices:

$$A = \begin{pmatrix} 1 & 2 & -1 \\ 2 & 1 & -2 \\ -3 & 1 & 1 \end{pmatrix} = \tag{3.4}$$

This is an example of _____.

Definition. A matrix A is said to be _____ if $A_{ij} = 0$, $i < j$, or _____ if _____.

To find the LU factorization of a matrix, simply perform row reduction as usual to get U in row-echelon form, keeping track of the row operations. Then form L with 1 along the diagonal, with the _____ of the multipliers in the lower triangle.

Example 5. Find the LU factorization of *a)* $\begin{pmatrix} 1 & -3 \\ 2 & 2 \end{pmatrix}$ *b)* $\begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix}$

Let's see how we can use LU factorization to help us solve linear systems. Here's a step-by-step guide:

- From $A\mathbf{x} = \mathbf{b}$, we LU-factorize so that _____.
- Let $\mathbf{y} =$ _____. First we solve _____ for $\mathbf{y}$ (easy).
- Now solve _____ for $\mathbf{x}$ (easy).

Example 6. Solve the system (3.2) using LU factorization.

Example 7. Solve the same system but with the RHS changed to $\mathbf{b} = \begin{pmatrix} 2 & 1 & 2 \end{pmatrix}^T$. How would you have done this with 'traditional' Gaussian elimination?

So, why do LU? It *feels* like we're doing extra work with LU factorization compared with Gaussian elimination, so why do we bother?

Firstly, as we saw in the last Example, LU factorization keeps a record of the row operations, which remains the same even if $\mathbf{b}$ is changed. There is no need to know $\mathbf{b}$ until after the factorization is complete. On the other hand, a 'traditional' Gaussian elimination would have to be redone from scratch each time $\mathbf{b}$ is changed.

Secondly, there is really no extra computational work compared with Gaussian elimination. We just felt as though there are extra steps in LU factorization simply because we only dealt with $\mathbf{b}$ separately in the final stage, whereas in Gaussian elimination, we involved $\mathbf{b}$ in our calculation from the beginning in augmented form.

We now argue that for problems involving multiple $\mathbf{b}$'s, the LU approach is *significantly* faster than naive Gaussian elimination.

3.1.1 Complexity

How many operations (*i.e.* $+$ or $\times$) does it take to reduce an $n \times n$ matrix to row-echlon form by Gaussian elimination? Here's a rough estimate.

In computing, we often speak of the _____ of an algorithm, which quantifies the computational cost. For example, one can show that the number of operations needed to obtain the row-echelon form by Gaussian elimination is at most

$$\frac{2n^3}{3} + \frac{n^2}{2} - \frac{7n}{6}. \tag{3.5}$$

When n is large, only the leading term is important. We use the _____ notation to express the leading order of an expression: We can say that Gaussian elimination is an _____ process.

On the other hand, the process of *back substitution* (*i.e.* solving for x_i given a triangular matrix) is a far less demanding process, as we now show.

Suppose we want to solve for $x_1, x_2 \ldots x_n$ from the augmented matrix

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & & \ldots & a_{1n} & b_1 \\ & a_{22} & & \ldots & a_{2n} & b_2 \\ & & & \ddots & \vdots & \vdots \\ & & a_{n-1,n-1} & & a_{n-1,n} & b_{n-1} \\ & & & & a_{nn} & b_n \end{array}\right)$$

Example 8. Show that the back-substitution process is an _____ process.

So if we have to solve $A\mathbf{x} = \mathbf{b}$ for multiple $\mathbf{b}$'s, all we have to do is *LU*-factorize *once*, then do multiple back-substitutions, costing _____ each time. This is much faster than multiple Gaussian eliminations, costing _____ each time.

Question. What is the complexity of calculating the determinant of an $n \times n$ matrix by the usual row or column expansion? *Ans:* _____.
MATLAB does not use row expansion in calculating a determinants. Instead, it first performs a version of the *LU* factorization on the matrix. Can you see why the LU factorization would help with finding the determinant?

3.2 PA=LU factorization

3.2.1 Row permutation

Example 9. Show that $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ does not have an LU factorization.

You may have noticed that the matrix in the above Example would already be in a triangular form had we *swapped the rows*. This can be achieved by pre-multiplying by a matrix

$$\begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} & & \\ & & \\ & & \end{pmatrix} \tag{3.6}$$

Example 10. Write down the all matrices which can be used to swap (permute) rows of a 3×3 matrix. How many such matrices are there with dimension $n \times n$?

These row swappers are called _____ matrices. You might have already seen them in an algebra course - indeed they form a _____ under matrix multiplication.

For our purpose of solving $A\mathbf{x} = \mathbf{b}$, where A does not have an LU factorization, all we need is to do is keep swapping by multiplying by a succession of permutation matrices, which when multiplied altogether become another permutation matrix, P. More about this in the next section.

Example 11. Given the matrix $A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$, write down an equation connecting it with $P = \begin{pmatrix} \quad & \quad \\ \quad & \quad \end{pmatrix}$, $L = \begin{pmatrix} \quad & \quad \\ \quad & \quad \end{pmatrix}$ and $U = \begin{pmatrix} \quad & \quad \\ \quad & \quad \end{pmatrix}$. *Ans:* _____

► **(MATLAB) Example 12.** To see the LU decomposition of matrix A on MATLAB, use the command `lu(A)`, and define two matrices as output. For example, `[L,U]= lu(A)`.

Similarly, where a permutation matrix is needed, use the command `[L,U,P]= lu(A)` (the ordering is important).

- Use MATLAB to confirm the factorization the above Example.
- Does MATLAB agree with our LU factorization in Example 5?
  Write down its outputs for L, U and P for part (b).

3.2.2 Swamping

Let's try to understand why MATLAB's suggestion for the LU factorization in the previous Example was not what we expected.

Example 13. Consider the following system:

$$10^{-20}x_1 + x_2 = 1$$
$$x_1 + 2x_2 = 4$$

(a) Use Gaussian elimination to find exact solutions.

(b) Estimate the numerical values of the exact solutions.

(c) What happens if these calculations were to be done on a computer with IEEE double precision?

This Example demonstrates the kind of problem that can arise if the multipliers involved in performing the row reduction are large. We see that the effect of subtracting $10^{20} \times R_1$ was to overwhelm, or _____, R_2. The information contained in R_2 is then completely dominated by R_1, and this leads to inaccurate answers due to limitation of IEEE arithmetic. This kind of inaccuracy introduced by using *large multipliers* during row reduction is called *swamping*.

Let's look at the same problem again with the rows swapped.

Example 14. Solve the following system with Gaussian elimination under IEEE double-precision arithmetic.

$$x_1 + 2x_2 = 4$$
$$10^{-20}x_1 + x_2 = 1$$

Swapping rows allowed us to keep the multiplier small, and so we retain the fragile information in both equations. The solutions aren't exactly correct, but they are very good approximations of the exact answers, certainly much better than $(x_1, x_2) = (0, 1)$.

The moral of the story : _____. One way to do this, as we have seen, is to swap rows around before doing row reduction so that the leading coefficients with the largest *magnitude* in each column rise to the top. This kind of row reduction plus vigilant row-swapping is called *partial pivoting*. In this method, multipliers are always kept small (magnitude at most 1) to avoid swamping. Here's an example.

Example 15. Apply Gaussian elimination with partial pivoting to solve the system

$$\begin{aligned} x_1 - x_2 + 3x_3 &= -3 \\ -x_1 - 2x_3 &= 1 \\ 2x_1 + 2x_2 + 4x_3 &= -1 \end{aligned} \tag{3.7}$$

Note that the magnitude of the multipliers are all _____, hence swamping is avoided.

Example 16. Write down a single matrix, P, which does *all* the row swapping in the previous Example.

We now combine partial pivoting with LU factorization. To keep track of the positions of all the multipliers (even when they are swapped), when a coefficient is eliminated, let's make a HUGE zero, inside which we write the multiplier used to eliminate the coeffcient in that position.

Example 17. Find the LU factorization of $A = \begin{pmatrix} 1 & -1 & 3 \\ -1 & 0 & -2 \\ 2 & 2 & 4 \end{pmatrix}$, with partial pivoting.

The matrices P, L and U obtained in the previous example are related to A by a single equation:

This is the so-called _____ factorization. In fact, even if a matrix can be factorized without partial pivoting, MATLAB still does $PA = LU$ to prevent potential error from swamping.

Example 18. Find the $PA = LU$ factorization of the matrix $\begin{pmatrix} 3 & 1 & 2 \\ 6 & 3 & 4 \\ 3 & 1 & 5 \end{pmatrix}$. Check your answers with MATLAB's answers in Example 12.

Example 19. Outline the steps involved in solving $A\mathbf{x} = \mathbf{b}$ using the $PA = LU$ factorization

Example 20. Solve the following system using $PA = LU$ factorization.

$$\begin{aligned} 3x_1 + x_2 + 2x_3 &= 0 \\ 6x_1 + 3x_2 + 4x_3 &= 1 \\ 3x_1 + x_2 + 5x_3 &= 3 \end{aligned} \tag{3.8}$$

3.3 Cholesky factorization

LU (or $PA = LU$) factorization can be applied to any matrix, but if a matrix has special properties (*e.g.* symmetric, or has an almost diagonal structure), we could exploit these properties to speed up the factorization process.

In this section, we will study the factorization of *symmetric positive-definite* matrices, called the Cholesky factorization[1] This method of solving $A\mathbf{x} = \mathbf{b}$ is faster than the usual LU algorithm by about a factor of 2 and requires less storage.

We will be using some concepts from last year's *Linear Algebra* course, so it's worth digging up your old notes.

3.3.1 Positive-definite matrices

Definition. A matrix A is _____ if

$$\mathbf{x}^T A\mathbf{x}$$

for all column vectors $\mathbf{x}$ ($\mathbf{x} \neq \mathbf{0}$).

Example 21. (a) Show using the definition that the following matrices are positive definite.

a) $\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$ b) $\begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}$ c) $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

(b) Show that the matrix $\begin{pmatrix} 2 & 4 \\ 4 & 5 \end{pmatrix}$ and $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 5 \end{pmatrix}$ are not positive definite.

Ans: See Lab worksheet.

Note that the *'positive'* in the phrase *'positive-definite'* refers to the expression $\mathbf{x}^T A\mathbf{x}$. Being a positive-definite matrix does **not** mean that the matrix entries are positive, as the above Examples shows.

[1] French mathematician *André-Louis Cholesky* (1875 – 1918) developed this method as a military engineer during WWI.

3.3.2 Cholesky ≈ 'square-root' for matrices

Definition. A matrix A is *symmetric* if _____.

The idea of Cholesky factorization is that if a matrix A is symmetric and positive definite, then it has a 'square-root', R, which is an *upper triangular* matrix, such that

$$A = \tag{3.9}$$

Example 22. (a) Construct an upper triangular matrix R such that $A = \begin{pmatrix} a & b \\ b & c \end{pmatrix} = R^T R$. State the assumptions you need for this factorization to work.

(b) Hence, find a Cholesky factorization of $\begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}$.

(c) Is R unique?

The previous Example gives us a flavour of the factorization method. Next we want to establish that a symmetric positive-definite matrix will automatically satisfy all the conditions needed in our ad-hoc construction of R.

3.3.3 Properties of SPD matrices

I'm getting tired of writing symmetric, positive-definite matrices, so we'll call them _____ matrices from now on. First, a little warm-up exercise.

Example 23. Find the eigenvalues and eigenvectors of $\begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}$.

Lemma 1. If A is symmetric, then it is positive-definite if and only if all its eigenvalues are positive.

Proof. Suppose that A is positive definite. Let $\mathbf{v}$ be an eigenvector of A with eigenvalue λ. Observe that

$$0 < \mathbf{v}^T A\mathbf{v} =$$

This proves the 'only if' part.

Now let's look at the 'if' part. Recall from last year's Linear Algebra that if A is symmetric, then it can be diagonalised to

$$D =$$

where the diagonal entries of the diagonal matrix D are its _____, and columns of P are its _____. These n eigenvectors, when normalized, in fact form an _____ of $\mathbb{R}^n$. Furthermore, $P^T =$ _____ (*i.e.* P is an _____ matrix).

Therefore we can write any nonzero vector $\mathbf{x}$ as

$$\mathbf{x} =$$

where $\hat{\mathbf{v}}_i$ are the normalised eigenvectors, and c_i are not all zero. Now observe that

$\mathbf{x^T} A\mathbf{x} =$

□

Lemma 2. If matrix A is SPD, then $|A| > 0$.

Proof. Since A is symmetric, can be diagonalised so that $A =$ _____ (where $P^{-1} = P^T$).

□

Next, we will need to discuss matrices of *full rank*. Let's recall what this means.

Definition. A square matrix is said to be *full rank* if its columns are _____ (and so are its rows).

Example 24. Write down a i) 3×3 ii) 2×3 iii) 5×3 matrix of full rank.

If a non-square matrix, say, 4×3, is full rank, we see that the rows can't all be linearly independent, so it's not useful to discuss the rows. We say that a 4×3 matrix is full rank if its _____ are linearly independent. Similarly, a 3×4 matrix is full rank if its _____ are linearly independent.

Definition. An $n \times m$ matrix with $n \leq m$ is said to be full rank if its _____ are linearly independent. Similarly, if $n \geq m$, then it is full rank if its _____ are linearly independent.

Lemma 3. If A is an $n \times n$ SPD matrix, and X is an $n \times m$ matrix of full rank with $n \geq m$, then $X^T AX$ is an $m \times m$ SPD matrix.

Proof. Let $M = X^T AX$. M is clearly symmetric because

$$M^T =$$

To see that M is positive definite, for any column vector $\mathbf{v} \neq \mathbf{0}$, we find

$$\mathbf{v}^T M\mathbf{v} =$$

where we assume that $X\mathbf{v} \neq 0$. Is it possible for $X\mathbf{v} = 0$ for some unlucky choice of $\mathbf{v}$?

Write X is terms of its column $X = \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 \dots \mathbf{x}_m \end{pmatrix}$ and write $\mathbf{v} = \begin{pmatrix} v_1 & v_2 \dots v_m \end{pmatrix}^T$. Hence $X\mathbf{v}$ can be expressed as:

□

Definition. A *principal* submatrix of a square matrix A is a square matrix whose diagonal comprises consecutive diagonal entries of A.

Example 25. Write down all principal submatrices of $\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$.

Example 26. Write down a matrix X such that $X^T \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} X$ equals

a) $\begin{pmatrix} 1 & 2 \\ 4 & 5 \end{pmatrix}$ b) $\begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix}$ c) $\begin{pmatrix} 9 \end{pmatrix}$

Lemma 4. Any principal submatrix of a SPD matrix is also SPD.

Proof. Let A be an $n \times n$ SPD matrix. To obtain a principal submatrix, M, of dimension $m \times m$, we sandwich A between X^T and X, as in the above Examples, where X comprises consecutive columns of the _____, and X is therefore _____.

The dimension of X is _____ with _____, and so $M = X^T AX$ is SPD by Lemma _____. □

Corollary. If A is an SPD matrix, then its diagonal entries are all _____.

Let's practise spotting SPD matrices using all the Lemmas we've proved so far.

Example 27. Are the following *symmetric* matrices positive definite?

$$A = \begin{pmatrix} 1 & 2 & 1 \\ 2 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix}, \qquad B = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 5 \end{pmatrix}, \qquad C \text{ with } |C| = -2, \qquad D \text{ with eigenvalues } 2, 3, -4.$$

Here's one final Lemma.

Lemma 5. If A and B are both upper triangular matrices, then AB is also upper triangular.

Proof. Let $C = AB$. The (i, j) entry of C is given by

$$c_{ij} = \sum$$

We want to show that $c_{ij} = 0$ if _____.

Since A is upper triangular, $a_{ik} = 0$ if _____, so we can simply sum over the nonzero entries of A:

$$c_{ij} = \sum$$

Suppose $i > j$. If $k \geq i$, then _____. But B is upper triangular, which means _____. Hence $c_{ij} = 0$ for $i > j$ and C is indeed upper triangular. □

Now we are ready to tackle the Cholesky factorization for a general SPD matrix.

3.3.4 Cholesky factorization theorem

Theorem 3.1. If A is a SPD matrix, there exists an upper triangular matrix R such that _____

Proof. Let A be an $n \times n$ SPD matrix. We can write it in *block* form as

$$A =$$

where $\mathbf{b}$ is a column vector, and C is a *symmetric* matrix of size _____. By Lemma 4, we also know that the the top left entry, a, is _____.

Now observe that we can split A into:

$$A = \begin{pmatrix} \sqrt{a} & \\ \mathbf{b}/\sqrt{a} & I \end{pmatrix} \begin{pmatrix} 1 & \\ & C - \mathbf{b}\mathbf{b}^T/a \end{pmatrix} \begin{pmatrix} \sqrt{a} & \mathbf{b}^T/\sqrt{a} \\ & I \end{pmatrix} \equiv R_1^T A_1 R_1. \tag{3.10}$$

This identity is the single key to success of Cholesky factorization. Verify it.

Next, let's work on the matrix A_1. If A_1 is SPD (we'll need to prove this later), then its submatrix _____, is also SPD by Lemma 4. Writing it in block form, we can again use the same identity to factorise it as

$$A_1 = \begin{pmatrix} 1 & & \\ & a_1 & \mathbf{b_1}^T \\ & \mathbf{b_1} & C_1 \end{pmatrix} =$$

and thus $A =$_____. We then work on the matrix A_2 and carry on until the matrix $A_n =$ _____. Hence, we have

$$A =$$

where the product of upper triangular R_i is still upper triangular (Lemma 5). This is the factorization required.

It only remains to show that the matrix A_i in each stage is SPD. Consider the expression for A_1 in Eq. 3.10.

$$A_1 =$$

R_1 is clearly a full-rank square matrix, and so is R_1^{-1}. Thus, A_1 is SPD by Lemma _____. Similarly, all the subsequent A_i's are all SPD. □

In particular, if we apply the Cholesky factorization to a 2×2 SPD matrix, we should find that $a > 0$ and $c - b^2/a > 0$ (by Lemma 4). These are in fact the "assumptions" needed in our ad-hoc factorization in Example 22. We have just shown that these assumptions are in fact automatically satisfied by SPD matrices.

Let us now see the Cholesky factorization in action. The technique is to use the identity (3.10) to construct R from the top down, layer by layer.

Example 28. Use identity (3.10) to find the Cholesky factorization of $\begin{pmatrix} 2 & 2 \\ 2 & 5 \end{pmatrix}$.

Example 29. Find the Cholesky factorization of $A = \begin{pmatrix} 4 & -2 & 2 \\ -2 & 2 & -4 \\ 2 & -4 & 11 \end{pmatrix}$.
Verify your answer using the MATLAB command `chol(A)`.

Example 30. Find the Cholesky factorization of $A = \begin{pmatrix} 1 & 2 & 0 \\ 2 & 5 & 2 \\ 0 & 2 & 5 \end{pmatrix}$.

Example 31. Use the Cholesky factorization to solve

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

Example 32. Apply Cholesky factorization to $A = \begin{pmatrix} 4 & 2 & 2 \\ 2 & 2 & 4 \\ 2 & 4 & 5 \end{pmatrix}$. Explain the result.

3.3.5 Why do Cholesky?

Why is Cholesky factorization useful? Here are some reasons.

(a) The main advantage is that the Cholesky factorization is *twice* as fast as LU factorization (since L is simply U^T).

(b) SPD matrices are not as special and rare as you might think. They occur in many areas of mathematics and physics, for example,

- covariance matrix in statistics
- metric tensor in differential geometry
- moment-of-inertia matrix in mechanics
- stiffness matrix in PDE analysis
- conductance matrix of an electronic circuit

(c) SPD matrices can be easily constructed from any matrix (even non-square ones). It's not difficult to show that if A is any full-rank matrix, then A^TA is SPD.

Consider the system $A\mathbf{x} = \mathbf{b}$ when A is skinny, *i.e.* an overdetermined system - with too many equations). There are no solutions to $A\mathbf{x} = \mathbf{b}$, but $A^TA\mathbf{x} = A^T\mathbf{b}$ has a unique set of solutions called the _____ solutions, which can be used for fitting a line/curve through data points.

(d) The Cholesky algorithm is an efficient way to test whether a symmetric matrix is positive definite (without having to calculate eigenvalues). If A is not SPD, then Cholesky factorization will fail. However, Example 32 showed that the Cholesky algorithm still yields a nice factorization of the form

$$A =$$

This factorization can be used to solve $A\mathbf{x} = \mathbf{b}$ where A is symmetric but *not* positive definite, and is still roughly twice the speed of LU factorization.

3.4 Backslash revisited

I promised to return to MATLAB's \ operator at the beginning of the Chapter, and here it is. (This section is non-examinable.)

Like `fzero`, the backslash is a *hybrid* method,

When we issue the command `A\b` to solve $A\mathbf{x} = \mathbf{b}$, MATLAB goes through the following workflow.

- If A is already upper/lower triangular, launch *back substitution*.
- If A is permuted triangular, *rearrange* then back-substitute.
- If A is symmetric with positive diagonal entries, try *Cholesky*.
- If A is just symmetric, but not positive-definite, Cholesky will factorise A into $R^T DR$ (as in Example 32), then back-substitute.
- If A is none of the above, do $PA{=}LU$, then back-substitute.
- If all of the above fail, no unique solution can be found.

I have given a simplified account above. The actual workflow is a bit more complicated. See the help documentation on `mldivide` in MATLAB.

Compare the above workflow to what happens when we try to solve $A\mathbf{x} = \mathbf{b}$ using the command `inv(A)*b`. MATLAB does the following:

- Solves $A\mathbf{c_1} = \begin{pmatrix}1\\0\\0\end{pmatrix}$, $A\mathbf{c_2} = \begin{pmatrix}0\\1\\0\end{pmatrix}$, $A\mathbf{c_3} = \begin{pmatrix}0\\0\\1\end{pmatrix}$ as above.
- Constructs `inv(A)` $= (\mathbf{c_1}\ \mathbf{c_2}\ \mathbf{c_3})$.
- Calculates `inv(A)*b`.

This process has more back-substitutions and many more operations than the backslash workflow, and generally takes a few times longer. This explains the results in the *Graph* assignment.

The main message of this Chapter is that you should never solve $A\mathbf{x} = \mathbf{b}$ by finding A^{-1}. In real appications, matrix inverses are never needed explicitly.

Nevertheless, there has been a quest to reduce the complexity of matrix inversion from $\mathcal{O}(n^3)$ by 'pre-conditioning' the matrices (repackaging and storing new variables to minimize the number of operations). The current record is $\mathcal{O}(n^{2.373})$ (*Coppersmith-Winograd* algorithm) but the steps involved are so unwieldy and the improvements are only noticable for matrices so large that they remain, for now, mathematical curiosities.

4

Interpolation

Suppose we are given the following data points on the $x - y$ plane:

(0,1), (2,2), (3,4).

Can we describe these data points with a single continuous function $y = f(x)$? Can we estimate the value of, say, $f(1)$? This is a common task in real applied mathematics – creating a *continuous* model that fits some given data.

You can see that the function $f(x)$ is not unique. You could, for instance, connect the points with straight lines, or any wiggly curve which "goes through", or _____, those 3 points. The question is, therefore, what are the requirements for $f(x)$? e.g. is it one single polynomial? of what degree? what happens at the end points? *etc.*

Here's another reason why interpolation is important. Suppose we know that the data points are obtained from a function $y = f(x)$, but it is very complicated and each function evaluation takes hours to compute. Can we find a simpler function (say, a polynomial) which is a good approximation of $f(x)$?

In this Chapter, we will tackle these interpolation problems, using the simplest kind of functions, *i.e.* _____, as building blocks. We will work mainly in 2D, but occasionally in 3D too.

Definition. A function $y = P(x)$ is said to **interpolate** points $(x_1, y_1), \ldots (x_n, y_n)$ if

$$= \tag{4.1}$$

for all $i = 1 \ldots n$.

Data points are often (but not always) listed increasing x values, with no x_i repeated (why not?). The data points (x_i, y_i) are also known as _____ (or *knots*, or *control points*).

4.1 Lagrange polynomial

Example 1. Find a function, $y = P(x)$, which interpolates between (x_1, y_1) and (x_2, y_2). Express your answer in the form $y_1 L_1(x) + y_2 L_2(x)$ (where L_i are some functions).

A fancy name for drawing a straight line through some nodes is _____.

Example 2. Using the previous Example, write down a polynomial, $y = P(x)$, which interpolates three distinct nodes $(x_1, y_1), (x_2, y_2), (x_3, y_3)$. What degree is this polynomial?

What we've found is called the _____ form of the interpolating polynomial.

Example 3. Find the Lagrange form of the interpolating polynomial given the nodes $(0,1),(2,2),(3,4)$. Hence, express your answer in the form $P(x) = ax^2 + bx + c$.
Sketch the curve and find $P(1)$.

The Lagrange polynomial can be easily generalised to higher orders: Suppose we were given 4 nodes $(x_1, y_1) \ldots (x_4, y_4)$. The Lagrange interpolating polynomial is

$$P(x) =$$

where $L_1(x) =$ $L_2(x) =$

and so on. Note that $P(x)$ is a polynomial of degree _____.

More generally, given n nodes, (x_i, y_i), $i = 1 \ldots n$, we can write down the Lagrange interpolating polynomial (degree _____) as

$$P(x) = \tag{4.2}$$

where $L_i(x) =$

At the nodes, $L_i(x)$ satisfies the condition

$$L_i(x_j) = \tag{4.3}$$

Is this polynomial *unique* as an interpolant? *Yes* (sort of)... as we now prove.

Theorem 4.1. Given distinct nodes (x_i, y_i), $i = 1 \dots n$, there exists a *unique* interpolating polynomial of degree at most $n - 1$.

Proof.

Corollary. The polynomial (4.2) is the unique polynomial of degree at most $n - 1$, interpolating n nodes.
Amongst other things, the Theorem implies that there is only one line through 2 points.

Question. Is it possible to interpolate 3 points with a line?

Question. Is it possible to interpolate 2 points (x_1, y_1) and (x_2, y_2) using a parabola?

This explains the importance of the phrase "at most" in the uniqueness result.

Example 4. (a) Find a polynomial of the lowest order which interpolates the points $(0, 2), (1, 1), (2, 0), (3, -1)$. Simplify your answer.
[*Hint: It's not supposed to be so painful.*]

(b) Write down a polynomial of degree 10 interpolating the nodes.

4.2 Divided differences

The Lagrange form of the interpolating polynomial (4.2) is, as you have seen, quite tedious to work out, so it's not often used. Here is a more efficient way to work out the same polynomial due to Newton. First let's rewrite the nodes as

$$(x_1, f(x_1)), (x_2, f(x_2)), \dots (x_n, f(x_n)). \tag{4.4}$$

In other words, we assume that the y values come from some function $y = f(x)$, which could be a function which is so complicated that it's better to work with a polynomial approximation, $P(x)$, which agrees with $f(x)$ at the nodes.

Definition. The *divided differences*, $f[x_1\ x_2 \dots\ x_n]$ is defined as the coefficient of x^{n-1} in the (unique) polynomial of order at most $n-1$, interpolating the nodes $(x_1, f(x_1)), \dots (x_n, f(x_n))$.

In other words, it's just the coefficient of the term with the highest power.

Example 5. Given the nodes (x_1, y_1) and (x_2, y_2), write down $f[x_1]$ and $f[x_1\ x_2]$.

This explains the name 'divided differences'.

Question. True or false? $f[x_1\ x_2] = f[x_2\ x_1]$.

Example 6. In Example 3, we found the interpolating polynomial for the points (0,1), (2,2), (3,4). Write down $f[0\ 2\ 3]$.
Calculate $f[0], f[2], f[3], f[0\ 2], f[2\ 3]$.

Let's write down the above divided differences as a little *tableau*

$$\begin{array}{c|ccc} x_1 & f[x_1] & & \\ & & f[x_1\ x_2] & \\ x_2 & f[x_2] & & f[x_1\ x_2\ x_3] \\ & & f[x_2\ x_3] & \\ x_3 & f[x_3] & & \end{array}$$

The next Theorem tells us how to work out $f[0\ 2\ 3]$ from the previous entries (*i.e.* a *recursive* method).

Theorem 4.2. Given distinct nodes at $x_1, x_2, \ldots x_n$, the divided differences obey the recursive relation

$$f[x_1\ x_2 \ldots\ x_n] = \frac{f[x_2\ x_3 \ldots\ x_n] - f[x_1\ x_2 \ldots\ x_{n-1}]}{x_n - x_1}.$$

Proof. Let $q(x)$ and $p(x)$ be the polynomials which interpolate the nodes $\{x_1 \ldots x_{n-1}\}$ and $\{x_2, \ldots x_n\}$ respectively. Let these polynomials be of degree at most _____.

Consider the polynomial

$$r(x) = \frac{p(x)(x - x_1) - q(x)(x - x_n)}{x_n - x_1},$$

of degree at most _____. We now show that $r(x)$ interpolates all the nodes.

Question. Check that all the divided differences in Example 6 obey the recurrence relation we just proved.

Example 7. Find all the divided differences given the nodes a) $(0,2),(1,1),(2,0)$, b) $(0,2),(1,1),(2,0),(3,-1)$.

The previous Theorem explains how to recursively obtain *all* the divided differences in the tableau. In particular, we now show that the _____of the tableau gives us the interpolating polynomial.

Theorem 4.3. Given distinct nodes at $x_1, x_2, \ldots x_n$, the interpolating polynomial can be expressed as

$$\begin{aligned} P(x) =& f[x_1] + f[x_1\ x_2](x - x_1) + \\ & f[x_1\ x_2\ x_3](x - x_1)(x - x_2) \ldots + f[\qquad\qquad\quad]\prod \end{aligned}$$

Proof. Let $P(x)$ interpolate all n nodes, and $p(x)$ interpolate all except the last node. Consider

$$Q(x) \equiv P(x) - p(x).$$

Note that $Q(x)$ is a polynomial degree at most _____, with zeroes at $x =$ _____. Therefore

$$Q(x) =$$

where A is a constant. However, $P(x) = p(x) + Q(x)$ interpolates all points, so the coefficient of the highest power (x^{n-1}) is _____ by definition. This term must be part of $Q(x)$ since $p(x)$ is of degree only at most _____. Therefore, $A =$ _____, and thus we have

$$P(x) = p(x) + \qquad (\star)$$

Let's work inductively with this relation to show that it takes the form as shown in the Theorem. Suppose we have 2 nodes ($n = 2$). By definition, $p(x)$ interpolates just the first node, and so $p(x) =$ _____. Thus, relation $(\star)$ gives

$$P_2(x) =$$

where the subscript in $P_2(x)$ indicates that we have 2 nodes.

Suppose we add one more node ($n = 3$), we can take $p(x) =$ _____ (which interpolates the first two nodes), and relation $(\star)$ becomes

$$P(x) =$$

Thus we can see that, inductively, the Theorem holds for any number of nodes. □.

The previous Theorem tells us that once we've computed the tableau, the top diagonal gives the coefficients of the interpolating polynomial without having to mess around with the Lagrange form.

Example 8. Using the tableau method, write down the interpolating polynomial given the nodes $(0, 2), (1, 1), (2, 0), (3, -1)$.

Example 9. Find the quadratic polynomial which interpolates the nodes $(0, 1), (2, 2), (3, 4)$.

The recursive nature of the algorithm means that more interpolating points can be added without having to restart from the beginning (unlike the Lagrange form).

Example 10. Add a fourth node $(1, 0)$ to the previous Example. Evaluate the new interpolating polynomial, giving your answer in a *nested* form.

Computationally, it's *not* helpful to expand the polynomial in powers of x (why?). You could go further by nesting the polynomial *completely* (see Chapter 2), but this partially nested form is sufficient for computational purposes.

Example 11 (Mock Exam 2015)**.** a) Find the equation of a cubic polynomial, $P(x)$, which interpolates the points $(-1, 0)$, $(0, 1)$, $(1, 0)$ and $(3, 16)$.
b) A quartic polynomial, $Q(x)$, interpolates the above data points, and also goes through the point $(2, 4)$.
Show that $Q(x)$ can be expressed in the form

$$Q(x) = P(x) - Cx(x^2 - 1)(x - 3),$$

where C is a constant which you should determine.

▶ **(MATLAB) Example 12.** By modifying your code, `nest.m`, from Chapter 2, or otherwise, write a MATLAB function, `interDD(X,Y,x0)`, which evaluates the interpolating polynomial, $P(x)$, in Theorem 4.3 using nested multiplication, taking 3 arguments:

- `X` is an array of x coordinates of the nodes.
- `Y` is an array of y coordinates of the nodes.
- `x0` is the value at which the polynomial is to be evaluated.

In your function, you may embed the function `DD` on the next page to help you evaluate the divided differences.
Submit the file `interDD.m` onto Canvas

By the way, you might like to see your interpolation results as a graph. For example, to plot the polynomial in Example 9 with your `interDD` function, try the commands:

```
X = [0 2 3];
x0 = linspace(0,3);                % 100 points spread evenly in [0,3]
Y = [1 2 4];
for n = 1:100
        y0(n)= interDD(X,Y,x0(n));
end
plot(X,Y,'o',x0,y0)                % plot the data points with circles,
                                   % then plot interpolant.
```

Alternatively, if you can make your `interDD` function accept an *array* `x0` (and not just a single number `x0`), then, plotting is easily done with

```
x0 = linspace(0,3);  y0 = interDD(X,Y,x0);
plot(X,Y,'o',x0,y0)
```

Here's the code for finding the divided-differences. You can simply stick this code at the bottom of your code `interDD.m` (in the same M-file, after you `end` your function). This makes `DD` is a *local function* (meaning that only the function `interDD` can use it).

```
function A = DD(X,Y)
% output A = top diagonal of the divided-differences scheme
n=length(X);
for i=1:n
    v(i,1)=Y(i) ;        % The first column f[x_i]
end

for j=2:n                % Fill in the next columns..
    for i=1:n+1-j        % ...from top to bottom
        v(i,j)=(v(i+1,j-1)-v(i,j-1))/(X(i+j-1)-X(i));   % Divided differences
    end
end

for k = 1:n
    A(k)=v(1,k);         % Output = the top diagonal
end
```

4.3 Cubic splines

The word 'spline' derives from an East-Anglian dialect word meaning a thin strip of wood or metal used by draftsmen to draw large curves (for example, in shipbuilding). Indeed, many aspects of interpolation grow out of the industrial need to model and construct curves using simple and effective mathematical tools.

A spline is a simply a _____ polynomial interpolating some given nodes, satisfying some continuity conditions (more on this later). Here's a simple example.

Example 13. Find the piecewise-linear spline which interpolates $(1, 2), (2, 1), (4, 4), (5, 3)$. Sketch the spline.

If we require smoothness (differentiability) at the node, then clearly the linear spline like the one above is not adequate. Quadratic splines would give a smooth curve, but finding x for a given y value can be tricky.

_____ splines are especially useful for interpolation because they strike a balance between flexibility, visual appearance and complexity of the calculation.

Warning: In many books (in particular, those on computer graphics), splines are treated parametrically, meaning that the cubic spline segments are a cubic function in $t \in [0, 1]$. In this module, we deal with splines in Cartesian form. The two approaches are *not* equivalent – and they generally produce different splines.

In this section, let's assume the nodes are given in increasing x ($x_1 < x_2 < x_3 \ldots$).

Definition. Given n nodes, (x_i, y_i) $(i = 1 \ldots n)$, a cubic spline is a piecewise-cubic curve, comprising _____ polynomials:

$$S_1(x) = y_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 \quad \text{on } [x_1, x_2]$$

Note that in this form, we automatically have the interpolation condition:

$$= \tag{4.5}$$

To determine the spline coefficients, let's require the spline to be a twice-differentiable (_____) function on the entire open interval (x_1, x_n).

Definition. A *cubic spline* interpolating nodes (x_i, y_i), $(i = 1 \ldots n)$, is a piecewise cubic function which is _____ on the interval (x_1, x_n).

Example 14. Find the cubic spline through $(0, 3), (1, -2), (2, 1)$.

First, let's write down the skeleton of the two cubic pieces.

Note that we have _____ unknowns to solve.

First, the continuity (_____) condition at the tail end of each piece yields:

The smoothness (_____) condition yields:

The continuity of the second derivative (_____) yields:

After 4 equations, we have run out of constraints. This is because we have not placed any constraints on the derivatives at the endpoints.

There are several ways to impose the end-point conditions. Here's one way.

Definition. A cubic spline is said to satisfy _____ *end-point conditions* if the second derivative vanishes at the endpoints, *i.e.*

and (4.6)

The resulting spline is called a *natural spline*. In our example, the natural end-point conditions read:

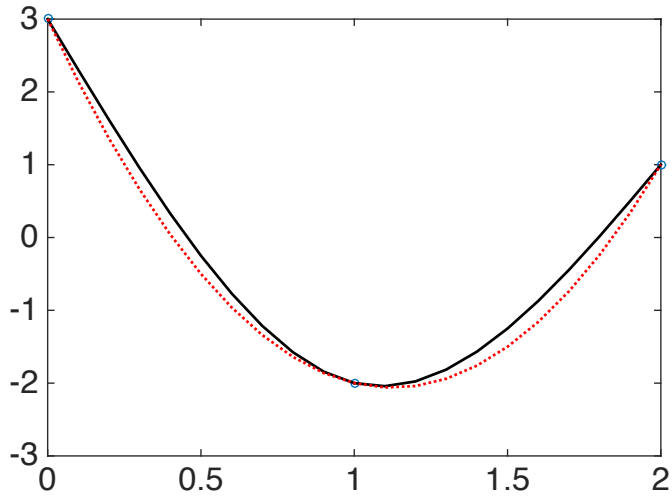
All these conditions give 6 equations necessary to solve the 6 unknowns. Let's try it with the help of MATLAB.

► **(MATLAB) Example 15.** Write down the matrix equation satisfied by $(b_1, c_1, d_1, b_2, c_2, d_2)^T$. Use MATLAB to solve for these values (don't use `inv`).

Actually there is a more humanly-achievable way to deal with this huge system – it can even be solved by hand. The idea is to break the problem down into two stages: first by writing all the b_i and d_i in terms of c_i, and then solve a linear system for c_i instead (this is a much smaller n^2 system, rather than $(3n-3)^2$). Thereafter, b_i and d_i can be recovered. The matrix equation in this method turns out to be much prettier than the one we saw earlier (it is in fact tridiagonal). However, the detail this so-called 'decoupling' method is quite tedious, though not difficult, so I'll leave it for now. Just be aware that a neat algorithm for finding spline coefficients exists, and a brute force method like what we did is not usually employed[1].

The graph below (black solid line) shows the result of our interpolation.

You may be wondering if MATLAB comes with its own spline functionality. The answer is yes: read the documentation on the commands `spline` (or, on a campus PC, look up the command `csape` which requires the curve-fitting toolbox). But MATLAB's own spline (shown in dotted line in the figure below) seems to differ from ours.



This is because MATLAB's cubic splines do not obey the natural end-point conditions. In general, MATLAB's spline[2] satisfies the following conditions at the first interior nodes:

$$S_1'''(x_2) = S_2'''(x_2) \qquad \text{and} \tag{4.7}$$

In other words, _____ condition holds at x_2 and x_{n-1} (they coincide in the above example).

[1] see §3.4 of Sauer for detail.

[2] If only 3 nodes are given, `spline` returns a quadratic interpolant, as in the figure above.

Example 16. Consider two cubic functions $S_1(x) = a + bx + cx^2 + dx^3$ and $S_2(x) = A + Bx + Cx^2 + Dx^3$. If their derivatives from 0th up to 3rd order all agree at $x = x_0$, show that $S_1(x) \equiv S_2(x)$ everywhere.

Therefore, if the MATLAB spline conditions (4.7) are satisfied, only a single cubic is needed across the first 3 nodes (same for the last 3 nodes).

This is why conditions (4.7) are known as the _____ conditions.

Example 17. Given the following cubic spline:

$$S(x) = \begin{cases} 1 + 2x + 3x^2 + 4x^3 & x \in [0, 1] \\ A + B(x-1) + C(x-1)^2 + 4(x-1)^3 & x \in [1, 2] \end{cases}$$

(a) Determine the values of the constants A, B, C. Hence find the coordinates of the nodes on which the spline interpolates.

(b) Show that the spline satisfies the not-a-knot conditions. Hence, verify that that the two cubic pieces are, in fact, identical.

Example 18. Find the constants A, B, C for the following natural spline.

$$S(x) = \begin{cases} 16 + A(x+1) + 3(x+1)^3 & x \in [-1, 0] \\ 8 + Bx + 9x^2 + Cx^3 & x \in [0, 1] \end{cases}$$

► **(MATLAB) Example 19.** Look at `data.worldbank.org/indicator/SP.POP.TOTL`
Pick any country and fill in the table below.

Year	Population
1985	
1995	
2005	
2015	

Use MATLAB to help you find the equation of the *natural* cubic spline which interpolates the data. Plot the spline along with the data points. (Use more nodes if you wish.)

Create a PDF file which nicely presents your data, your graph and the equation of the spline. Submit *only the PDF* on Canvas. Tips:

- Use `linspace` to create the x values for each piece of the spline.
- Show the data points on your spline using circles or crosses.
- *Optional extra credit*: explain any interesting trend in the graph (e.g. war, famine...)

4.4 Bézier curves

We end this Chapter with an interpolation technique that is very popular in computer design and graphics: _____ curves. In fact, these curves are used to produce the font you're reading right now.

Unlike the previous interpolation techniques, Bézier curves are _____. Let's first revisit linear interpolation from a parametric perspective.

Starting with 2 points, A and B (in any dimension). To describe any point along AB, we parametrize the line by a 'time' parameter t. It is convenient to set _____ at A and _____ at B. Let $\mathbf{a}$ and $\mathbf{b}$ be the position vectors of A and B, then the line AB can be described by the vector equation:

$$\mathbf{r}_{AB}(t) = \tag{4.8}$$

This equation forms the basis of everything that follows. Make sure you're ok with it.

Question. Let P be the point on AB for some given t. What is the ratio AP/AB?

Moving on from linear interpolation, one might now try to connect points with a curve which is quadratic in t. Just as a linear interpolation requires 2 points, a quadratic interpolation requires _____. We call these points _____ (*nodes* or *knots* are also used, but I'll make this distinction for a reason which will become clear.)

Given 3 control points A, B, C with position vectors $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, let's first try to do linear interpolation on AB and BC, using the same parameter $t \in [0, 1]$ for both lines.

$\mathbf{r}_{AB}(t) =$ $\qquad\qquad$ $\mathbf{r}_{BC}(t) =$

Now here is the neat idea behind Bézier curves: let's now *interpolate between these two lines* using the same parameter t. Let $\mathbf{B}(t)$ be the position of this point. Here is an example of $\mathbf{B}(t)$ for $t = 0, 0.25, 0.5, 0.75$, and 1.

The equation for $\mathbf{B}$(t) can be found by linear interpolation between the two lines.

$$\begin{aligned}\mathbf{B}(t) &= \\ &= \end{aligned}$$

$\mathbf{B}(t)$ is called a *Bézier curve* with control points A, B and C. Note that:

- Given 3 control points, $\mathbf{B}(t)$ is quadratic in t.
- $\mathbf{B}(t)$ begins at A ($t =$ _____) and ends at C ($t =$ _____)
- $\mathbf{B}(t)$ does **not** pass through B. Changing the coordinates of B will change the shape of the curve. This is why we call them *control points* rather than nodes.
- Let P and Q be points with parameter t on AB and BC respectively. Let R be the point with coordinates $\mathbf{B}(t)$. We have

$$\frac{AP}{AB} = \qquad = \qquad =$$

Example 20. (a) Determine the equation of the Bézier curve $\mathbf{B}(t)$ with control points at $(0, 3)$, $(2, -1)$ and $(3, 0)$.

(b) Roughly sketch the Bézier curve.

(c) Hence find (x, y) coordinates on the curve at $t = 1/2$.

(d) Find the value of t at the turning point of the curve.

Pierre Étienne Bézier (1910-1999) French engineer. At 23, he started working at Renault, where he remained for the next 42 years. He was a pioneer of computer-aided design and published work on what we now call Bézier curves (and surfaces), based on earlier work of fellow Frenchman Paul de Casteljau (who incidentally worked for rival Citroën). The font you're reading now is rendered using cubic Bézier curves.

Example 21. Find the control points of the following Bézier curve:

$$x(t) = 2t^2 + 2t - 1, \qquad y(t) = 4t - 3t^2$$

If we have more control points, we can repeat the same sequence of linear interpolations to construct a Bézier curve. The degree of the resulting polynomial in t will increase with the number of control points. For instance, given 4 control points, the x and y components of the resulting Bézier curve will be _____ polynomials in t.

Question. Conjecture the form of a Bézier curve with 4 control points: **a**, **b**, **c**, **d**.

Let's quickly prove our conjecture. We start in the same way with a linear interpolation on each of the 3 segments. As before, a linear interpolation between the adjacent segments produces two quadratic Bézier with control points **a**, **b**, **c**, and **b**, **c**, **d**. The equations of these quadratic bits are:

$$\mathbf{h}_1(t) =$$
$$\mathbf{h}_2(t) =$$

Then, we perform a linear interpolation between the two quadratic Béziers to obtain

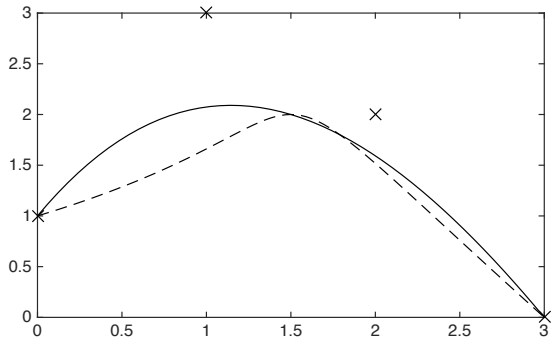
$$\mathbf{B}(t) =$$

Example 22 (Ordering matters!)**.** Write down the parametric form of the Bézier curve with control points given in the following order:

(a) $(0, 1)$, $(1, 3)$, $(2, 2)$ and $(3, 0)$.

(b) $(0, 1)$, $(2, 2)$, $(1, 3)$ and $(3, 0)$.

Calculate the coordinates of the point where $t = 0.2$ on each curve.



Note that by changing the ordering of the control points, the cubic Bézier curves can exhibit points of inflections, loops and cusps. (Roughly sketch these situations below).

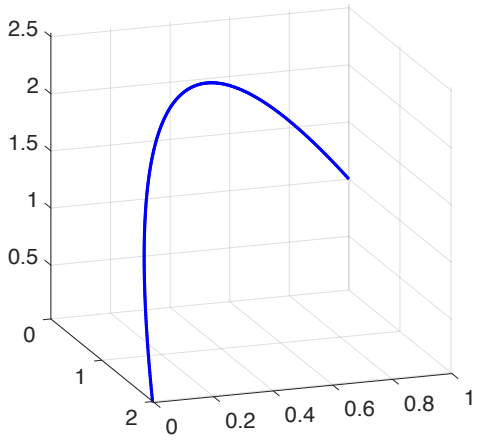
Example 23. A cubic Bézier curve has control points $\mathbf{b}_0, \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3$. The curve can also be written in the form

$$\mathbf{B}(t) = \boldsymbol{\alpha}_0 + \boldsymbol{\alpha}_1 t + \boldsymbol{\alpha}_2 t^2 + \boldsymbol{\alpha}_3 t^3.$$

Find the expression (in matrix form) for $\boldsymbol{\alpha}_i$ in terms of $\mathbf{b}_i$.
(More about this matrix in the tutorial.)

One advantage of studying Bézier curves vectorially (as we have done) is that all our results so far hold in *any* dimension. Here's an example in 3D.

Example 24. Write down the equation of the 3D Bézier curve joining $(2, 0, 0)$, $(1, 0, 4)$ and $(0, 1, 1)$. Show that the curve does not intersect the plane $z = 3$.



▶ **(MATLAB) Example 25.** Let `p` $= [x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4]$ be the list of 2D coordinates of four control points (x_i, y_i). A MATLAB code (written by a student) for plotting a Bézier curve given the array `p` is given below.

```
function bezplot(p)
t = linspace(0,1);
X=(1-t).^3*p(1) +3*t.*(1-t).^2*p(3) +3*t.^2.*(1-t)*p(5) +t.^3*p(7);
Y=(1-t).^3*p(2) +3*t.*(1-t).^2*p(4) +3*t.^2.*(1-t)*p(6) +t.^3*p(8) ;
plot(X,Y)
end
```

(a) Test the code using Example 22 to make sure it works.
Can the code be made more efficient?

(b) Modify the code so that it now accepts an $N \times 8$ matrix as the input (`p`). Each of the N row of `p` represents a single Bézier curve. The output should be a single plot containing all N Bézier curves.

Test your code on the following input data files:
`bez1.txt`
`bez2.txt`
`bez3.txt`
on Canvas. If it works, you should see some nice images.

On Canvas, submit your modified, annotated code that showed you the images.

Tip: Use `hold on` and `hold off` to plot multiple graphs.

5

Numerical Calculus

It should not surprise you that differentiation by product rule and integration by parts are quite beyond the limit of poor old IEEE binary arithmetic. *Computers do not understand calculus.* In this final chapter, we will see how to tackle differentiation and integration problems in purely *numerical* terms, involving basic arithmetic operations $(+, -, \times, \div)$.

I must emphasise that we are not going to do *symbolic* calculus. For example, we are not interested in the problems

$$\text{If } f(x) = x^2 + e^x, \text{ find } f'(x) \qquad \text{OR} \qquad \text{Find } \int \sin x \, \mathrm{d}x,$$

which are symbolic in nature, and so must be solved by specialised packages that have been taught the *rules* of calculus (*e.g.* MuPad). Instead, we are interested in problems that require numerical answers, such as

$$\text{If } f(x) = x^2 + e^x, \text{ find } f'(2) \qquad \text{OR} \qquad \text{Find } \int_0^{\pi} \sin x \, \mathrm{d}x,$$

which *can* be solved with simple binary arithmetic on a computer.

Things to recall before we proceed:

- *Taylor's Theorem* for expanding a function $f(x+h)$ as a series in h ($h > 0$).
  Full form:

  OR, if we expand up to the term in h^{n-1}, we can include the 'error' term, also called the _____ form of the remainder:

  where $\xi \in$ _____. The above equation requires $f^{(n-1)}$ to be continuous on the interval $[x, x+h]$ (*i.e.* f is a _____ function on this interval).

- *Intermediate-Value Theorem.* If f is continuous on $[a, b]$, then, for any given number y between $f(a)$ and $f(b)$,

- *Floating-Point Rep Theorem* (Theorem 1.1) The fractional error in representing a real number x as a float $fl(x)$ satisfies:

5.1 Numerical derivative

The key idea is to approximate the derivative at x by the gradient of the straight line drawn 'near' x. This idea is hardly new - recall the limit definition of the derivative:

$$f'(x) =$$

Here, we want to be more precise about the expression inside the limit. How 'near' should we go to x? and how accurate would it be? Let's now investigate.

Suppose that f is a C^2 function on $\mathbb{R}$. Let's write down the Taylor expansion of $f(x+h)$ with the error term of order h^2.

$$\implies f'(x) = \tag{5.1}$$

So we see that the error of this approximation is directly proportional to the size of h. For example, cutting h in half will also half the error (theoretically at least...).

Some terminology:

- h is called the _____.
- We say that the error in formula (5.1) is $\mathcal{O}$(_____).
- Another way to state the previous point is to say that the formula is *first-order* approximation. Note: an order-n approximation has an _____ error term.
- The formula (5.1) is known as the _____ formula for calculating the derivative. Can you see why?

Question. Write down the 2-point *backward*-difference formula for $f'(x)$.

In real applications of (5.1), it's not possible to know the exact size of the error (we don't even know $f'(x)$ exactly, let alone $f''(x)$). However, it's more useful to note how the error *scales* with h. This will allow us to minimize the error later.

Example 1. Consider $f(x) = 1/x$.

(a) Using the forward-difference formula with $h = 0.1$, estimate $f'(2)$ to 4 dec. pl.

(b) Find the actual answer and calculate the absolute the error E.

(c) Verify that the error E is within the expected bound from Taylor's theorem.

The next example shows what typically happens in real applications: there is little or no information about the exact form of f, and so we can only get a rough estimate on the size of the error.

Example 2. Estimate the derivatives below using the forward-difference formula with $h = 0.1$. Put $*$ in entries that cannot be calculated.

x	$f(x)$	$f'(x)$	$f''(x)$
-0.1	1.1052		
0	1.0000		
0.1	0.9048		
0.2	0.8187		

Note that the forward difference formula is an *asymmetric* scheme, which means that the round-off error can accumulate in one direction, becoming worse for higher-order derivatives. You might like to know that actually $f(x) = e^{-x}$ in the above example. In practice h should be much smaller (we will see how small later).

A redeeming feature of the two-point formula is that it is faster than other methods - only 2 evaluations of $f(x)$ are required per derivative. But if we can afford more evaluations, it's better to use the following *symmetric* estimate for $f'(x)$.

Theorem 5.1. If f is a C^3 function on $\mathbb{R}$, then $f'(x)$ can be estimated to *second order* using the *symmetric-difference* formula

$$f'(x) =$$

where $\xi \in (x-h, x+h)$.

Proof. Let's write down Taylor's expansions, with error term $\mathcal{O}(h^3)$,

$$f(x+h) = \tag{5.2}$$

$$f(x-h) = \tag{5.3}$$

where $\xi_1, \xi_2 \in I = (x-h, x+h)$. Subtracting gives

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6}\left[\frac{f'''(\xi_1) + f'''(\xi_2)}{2}\right] \tag{5.4}$$

Let x_m and x_M be the values of x where f''' attains the minimum and maximum on I. The final terms in the square brackets therefore satisfy the bound

$$\leq \frac{f'''(\xi_1) + f'''(\xi_2)}{2} \leq$$

Since f is a C^3 function on $\mathbb{R}$, f''' is continuous on I. By the Intermediate Value Theorem,

□

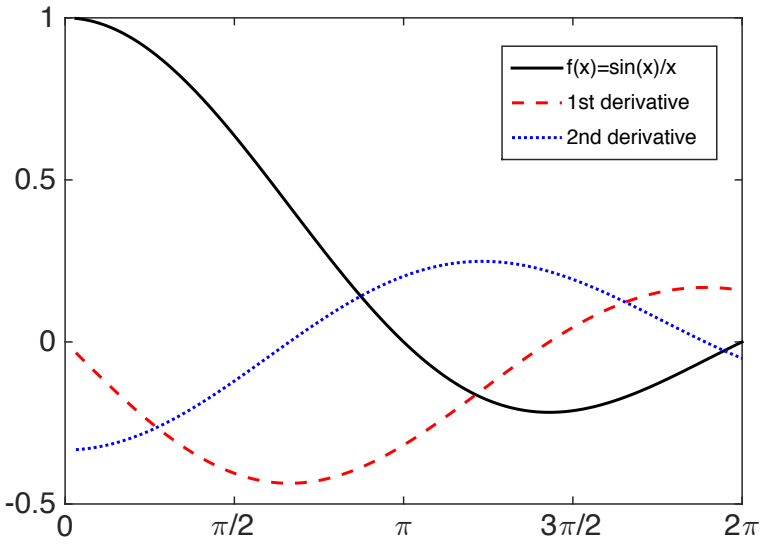
Example 3. Consider $f(x) = 1/x$. Use the symmetric-difference formula to estimate $f'(2)$, using step size $h = 0.1$, and compare the absolute error with that in Example 1.

Example 4. (Homework) Follow the derivation of Theorem 5.1 (but keep more terms) to obtain a symmetric-difference formula for $f''(x)$. You should find that this is an $\mathcal{O}(h^2)$ formula.

► **(MATLAB) Example 5.** Choose any interesting twice-differentiable function $f(x)$ on some domain. Write an M-file which, when run, produces a **beautiful plot** of $f(x)$, $f'(x)$ and $f''(x)$.

Your derivatives must be calculated using the symmetric-difference formulae (found in Theorem 5.1 and Example 4). Experiment with the step size h and use a value which you think gives a good estimate of the derivatives.

Plot everything on the same set of axes. Submit only a **single M-file** onto Canvas. Here's an example of what you should see when the M-file is run. I chose $f(x) = \dfrac{\sin x}{x}$.



Tips:

- Use different line types (*e.g.* dotted, dashed) or thicknesses to ensure that the 3 curves are distinguishable, even when printed in black and white.
- Some complicated plotting commands can be obtained by first editing the graph manually, then click on *File* ≪ *Generate Code...*

Warning: You must not differentiate anything by hand, nor should you need the symbolic functionality of MATLAB/MuPad. You should not need to type the command `syms x`.

5.2 Minimizing the error

Let's go back to the forward-difference approximation

$$f'(x) \approx \tag{5.5}$$

Theoretically, this approximation becomes more accurate as h becomes smaller. But surprisingly this is not true in practice, as we now explore.

Let's think about the sources of error associated with this formula. First of all, there is an error which comes from chopping off the Taylor series after a few terms. We have shown in Eq. 5.1 that this error has magnitude

$$E_T = \tag{5.6}$$

This is called the _____.

However, the forward-difference approximation also involves another kind of error coming from subtracting *nearly equal numbers*, leading to _____. You can imagine that this is worse when h becomes smaller and smaller. Let's estimate this second source of error (called _____).

Let the ε_1 and ε_2 be the error in the floating-point representation of $f(x)$ and $f(x+h)$ respectively, meaning that

$$\begin{aligned} \varepsilon_1 &\equiv fl(f(x)) - f(x) \\ &\equiv \end{aligned}$$

When evaluating the approximation (5.5) on a computer, we find

$$\begin{aligned} f'(x) &= \\ \\ &= \end{aligned}$$

This means that the rounding error, E_R, is given by

$$E_R = \tag{5.7}$$

(we are only interested in the magnitude of errors). Now let's try to estimate E_R. First, recall the floating-point representation theorem. The floating point rep of a quantity Q satisfies the error bound

$$\leq$$

This implies that

$$|\varepsilon_1| \leq$$

$$\leq$$

A *rough* estimate for E_R can be obtained via the triangle inequality.

$$E_R \leq$$

where we used the approximation $f(x) \approx f(x+h)$, for small h.

Therefore, the total error (truncation+rounding) is bounded above by

$$\begin{aligned} E(h) &= E_T + \max E_R \\ &\approx \end{aligned} \tag{5.8}$$

Regarding E as a function of h, we see that it's just a combination of a linear piece coming from E_T (which dominates at large h) and a h^{-1} piece coming from E_R, which dominates when h is small. This suggests that E has a minimum point.

Example 6. Estimate the value of h which minimises $E(h)$.

Since $\xi \in (x, x+h)$, for small h, we can make the approximation $f''(\xi) \approx$ _____. In conclusion, we see that the error in the forward-difference approximation is minimized if h is chosen such that

$$h \approx \tag{5.9}$$

Notice that you need different optimal h depending on *where* you are differentiating.

Example 7. If $f(x) = kx^n$ $(n > 0)$, estimate the optimal h for differentiating $f(x)$ using the forward-difference scheme.

The optimal h formula (5.9) looks great, but it is full of unknown quantities. Remember that we don't even know how to differentiate f, let alone finding f''. For practical purposes, we could assume that $f(x)$ and $f''(x)$ are roughly the same order of magnitude (for example, this works for $f(x) = \sin x$ or _____, but not for x^{1000}). If this is the case then the optimal h is *roughly*

$$h \sim \tag{5.10}$$

Here we follow the folkloric convention that $\approx$ means a *reasonable* approximation (*i.e.* estimate can be used with good confidence), whereas $\sim$ means a *rough* approximation (order-of-magnitude estimate only; hit-or-miss for some unusual functions).

So how big is this optimal h? well,

$$(\varepsilon_{\text{mach}})^{1/2} =$$

You will find that decreasing h beyond this number will result in an *increasing* error in the approximation.

Using this optimal h what is the minimum error achievable? Let's find out.

Example 8. Using the rough-approximation symbol $\sim$ in your working, estimate that the minimum error $E_{\min}$ achievable using the forward-difference formula. Express your answer in terms of $\varepsilon_{\text{mach}}$.
Make a rough sketch of the function $E(h)$, showing its turning point.

The previous Example shows that, in general, the forward-difference approximation produces derivatives that are accurate to no more than about 8 significant figures.

We can follow exactly the same procedure to arrive at the optimal h for the symmetric-difference scheme. This is so important that we will state it as a theorem.

Theorem 5.2. The error in the symmetric-difference formula for $f'(x)$ is minimized when h is chosen so that

$$h \sim (\varepsilon_{\text{mach}})^{1/3}.$$

Proof: Work on a separate sheet.

Example 9. Estimate the minimum error achievable using the symmetric-difference formula, giving your answer in terms of $\varepsilon_{\text{mach}}$.

The main results in this section can be summarised in the following graphs.

► **(MATLAB) Example 10.** Choose a function f and a value c at which f is differentiable (*i.e.* $|f'(c)| < \infty$). Compare the actual derivative with $f'(c)$ estimated using the *symmetric-difference* formula. Do this for a range of h.

Use MATLAB to produces a graph of the absolute error,

$$E(h) \equiv |\text{actual } - \text{estimate}|,$$

as a function of h. The graph should show that E has a minimum at some optimal h.

Discuss the main features of your graph, and explain whether they are consistent with the theory studied in this section.

Upload a single PDF of the graph along with your discussion (in the same PDF) onto Canvas. Here are some tips.

- To graphically display a function which fluctuates across many orders of magnitude, it's clearer to plot it on *log scale*. For example, if $y = x^2$. Plotting $Y = \ln y$ against $X = \ln x$ will result in a straight line $Y = 2X$. Read the documentation on `loglog`.
- Read about `logspace` (the partner of `linspace`). This will allow you to plot the graph over a large range of h.

▶ PRACTICAL TRICKS

Here are some industry-standard tricks and 'rules of thumb' when deploying numerical derivatives. (The material on this page is non-examinable, but the formulae and tricks discussed here are useful for real applications).

Firstly, if we approximate $f(x)$ as a polynomial of order n, then, the optimal h for the forward-difference formula is

$$h \sim \varepsilon_{\text{mach}}^{1/2}|x|.$$

(see Example 7). Clearly this estimate is not useful for calculating the derivative at $x = 0$ (why?), so we often use a more practical expression

$$h = \max\{x, 1\}\,(\varepsilon_{\text{mach}})^{1/2} \tag{5.11}$$

instead[1]. For example, to differentiate a function at some value of x, your code might look like this.

```
h = sqrt(eps)*max([x,1]);
df = (f(x+h)-f(x))/h;
```

Seems reasonable enough, but we shouldn't overlook the fact that there might also be round-off error from the expression $x+h$, since, generally $fl(x + h) \neq x + h$.

Let's suppose this round-off error is ε, *i.e.*

$$fl(x + h) = x + h + \varepsilon.$$

This means that the forward-difference ratio in the code is really

$$\texttt{df} = \frac{f(x + h + \varepsilon) - f(x)}{h}. \tag{5.12}$$

As you can see, this formula is a little forward-biased in the numerator. The result of this calculation is doomed to be inaccurate since it doesn't form a forward-difference ratio consistently for different x.

To avoid this error, we can perform the following trick.

```
h = sqrt(eps)*max([x,1]);
xph = x+h;
H = xph -x;
df = (f(xph)-f(x))/H;
```

Now we see that

$$\begin{aligned} fl(\texttt{xph}) &= x + h + \varepsilon, \\ fl(\texttt{H}) &= h + \varepsilon, \\ \implies \texttt{df} &= \frac{f(x + H) - f(x)}{H}. \end{aligned} \tag{5.13}$$

This expression is an exact forward-difference ratio at all x. Even though H isn't exactly h, (5.13) is still a superior approximation to (5.12) because we have eliminated one source of round-off error.

A similar trick can be performed for the symmetric-difference scheme to give you an improved accuracy.

[1] replace the square root by cube root for the symmetric-difference scheme.

5.3 Richardson extrapolation

You may find it disappointing that none of the derivative formulae in the previous sections has accuracy comparable to $\varepsilon_{\text{mach}}$. However, in this section, we will see how simple formulae like the forward and symmetric difference can be turbo-boosted to arbitrarily high accuracies, via a process called *Richardson extrapolation*.

First, let's see how to turbo-boost the forward-difference formula. As before, we start by considering the Taylor series for $f(x)$.

where we let $F(h) \equiv [f(x+h) - f(x]/h$ (the forward-difference estimate).

Now we use a simple trick to eliminate the next order error term ($\mathcal{O}(h)$). The trick is to *replace the step size h by $h/2$* (this is the motto of this section). This gives:

By using the original formula to eliminate the $\mathcal{O}(h)$ term, we easily obtain an $\mathcal{O}(h^2)$ (much improved) approximation.

More explicitly, this improved forward-difference formula for the derivative reads:

(5.14)

The payback for increasing the accuracy is the need to evaluate the function at more points. In addition to the function values at x and $x+h$, we also need another value at the point halfway in between.

This simple idea of reducing the error of an approximation by using more points in between is called _____. *Lewis Fry Richardson* (1881-1953) was an English mathematician and physicist whose studies on the coastline of Britain led to the discovery of fractals.

If desired, we can repeat the extrapolation trick on the new $\mathcal{O}(h^2)$ formula to obtain an $\mathcal{O}($_____$)$ formula, and so on. Often it won't be necessary to deal with the numerical constants in our working explicitly, as in the next example below.

Example 11. Use Richardson extrapolation on Eq. (5.14) to derive the formula

$$f'(x) = \frac{Cf(x) + 32f(x+\frac{h}{4}) - 12f(x+\frac{h}{2}) + f(x+h)}{3h} + \mathcal{O}(h^3)$$

where C is a constant which you should determine.

Note that there is no need to know the coefficient of the leading error term explicitly. Only the *order* is important.

Richardson extrapolation can, in fact, be applied to *any* quantity involving the step-size h (not just derivatives). In fact we will use it later for improving integration formulae.

Example 12. Suppose that a given quantity Q can be expressed as a series in h, and

$$Q = F(h) + \mathcal{O}(h^n).$$

Derive the Richardson-extrapolated formula for Q.

Now let's try Richardson extrapolation on the symmetric-difference formula for $f'(x)$. Surprisingly, each extrapolation will improve the order by not just one, but *two* orders of h ! To see this, it's worth re-deriving the formula here, without using the remainder term.

We see that the error terms in this approximation are all in even powers of h. Thus, a single extrapolation will eliminate the h^2 term, leaving us with an $\mathcal{O}(\_\_\_\_\_)$ formula.

Example 13. Use Richardson extrapolation to obtain the following symmetric-difference formula

$$f'(x) = \frac{f(x-h) - 8f(x-h/2) + 8f(x+h/2) - f(x+h)}{6h} + \mathcal{O}(h^4) \tag{5.15}$$

(No need to memorise any extrapolated formulae, but you should know *how* to obtain them.)

5.4 Numerical integration – prelude

Theorem (_____ Theorem for Integrals)**.** Let f and g be continuous on $[a, b]$, and either $g(x) \geq 0$ or $g(x) \leq 0$ on $[a, b]$ (*i.e.* $g(x)$ does not change sign) , then $\exists \xi \in (a, b)$ such that

$$\int_a^b f(x)g(x)\,\mathrm{d}x =$$

Proof. Since f is continuous, f must attain a maximum and a minimum on $[a, b]$, say, at $x = x_M$ and $x = x_m$ respectively (this is due to the _____ theorem). Thus,

$$\leq \int_a^b f(x)g(x)\,\mathrm{d}x \leq$$

where $I \equiv \int_a^b g(x)\,\mathrm{d}x$. Let's assume that $g(x) \geq 0$. This implies that _____.

- If $I > 0$, we can divide through by I and still preserve the inequality.

  Thus, $\exists \xi \in (a, b)$ such that

  $$=$$

  This is a consequence of the _____ Theorem. Multiplying both sides by I gives the required result.

- If $I = 0$ then the theorem holds for any ξ since the RHS is zero (in fact it can be shown that $g(x) \equiv 0$ in $[a, b]$ – an exercise in analysis).

To deal with the case $g(x) \leq 0$, simply replace $g(x)$ by _____ in the proof. □

Question. Interpret the above theorem graphically for the case $g(x) \equiv 1$.

Numerical integration involves problems that have *numerical* answers, *e.g.*

The key idea in numerical integration is to first approximate $f(x)$ as a _____ (why?) which matches $f(x)$ at $x = a$ and b, and possibly more *nodes* in between (the more nodes we have, the more accurate the answer will be). Thus, you can see that numerical integration relies on the technique of _____.

What's the error that results from this polynomial approximation? Here it is.

Theorem 5.3. Let $f(x)$ be a C^n function on $\mathbb{R}$, and let $P(x)$ be a polynomial such that $P(x) = f(x)$ at nodes $x_1, x_2 \dots x_n$. Then,

$$f(x) =$$

with the error given by

$$E(x) = \frac{f^{(n)}(c)}{n!}(x - x_1)(x - x_2)\dots(x - x_n),$$

where c lies between the minimum and maximum of the numbers $x, x_1, x_2 \dots, x_n$.

We will omit the proof in this course. Consult Sauer page 155 if you are interested.

Note that c depends on x (why?). To remind ourselves, we could write _____.

Even though it looks complicated, the above theorem simply says that:

$$f(x) = \text{Approximation} + \text{Error}.$$

Example 14. Write down a polynomial $P(x)$ which interpolates $f(x)$ at $x = x_0$ and $x = x_1$. Write down what Theorem 5.3 says in this case.
What happens when $f(x)$ is a straight line?
[Note: recall the shorthand $y_0 \equiv f(x_0)$, $y_1 \equiv f(x_1)$.]

5.5 Trapezium Rule

The calculation of the actual Trapezium-Rule approximation will be the same as what you've done in school, but here we will develop a deeper understanding of the error.

Consider the integral

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x,$$

where f is C^2 function on $[x_0, x_1]$. Let's try to do this integral with just one trapezoidal strip from $x = x_0$ to $x = x_1$.

First, write down $f(x)$ as a polynomial interpolating two end-nodes (x_0, y_0) and (x_1, y_1), plus the error term. Example 14 showed how these terms look like. Now let's try integrating $f(x)$.

Example 15. Evaluate $\int_{x_0}^{x_1} P(x)\,\mathrm{d}x$..
Note: it helps to express the answer in terms of the width of the strip, $h \equiv$ _____.

Example 16. Evaluate $\int_{x_0}^{x_1} E(x)\,\mathrm{d}x$. (Strategize before integrating!)

We have just proved the following theorem.

Theorem 5.4 (Basic Trapezium Rule)**.** If f is C^2 on $[x_0, x_1]$, then

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x = \tag{5.16}$$

Let's generalise this the basic (1 strip) version to the *composite* version (n strips).

Theorem 5.5 (Composite Trapezium Rule)**.** Let $a = x_0 < x_1 < x_2 \ldots < x_n = b$, where the nodes x_i are evenly spaced, with $h = (b-a)/n$. If f is C^2 on $[a, b]$, then

$$\int_{x_0}^{x_n} f(x)\,\mathrm{d}x = \frac{h}{2}\left[y_0 + y_n + 2\sum_{i=1}^{n-1} y_i\right] - \frac{(b-a)h^2}{12} f''(\xi), \quad \text{for some } \xi \in (a, b)$$

Proof: Applying the Basic Rule in the intervals $[x_0, x_1]$, $[x_1, x_2]$ $\ldots$ $[x_{n-1}, x_n]$ gives

$$\begin{aligned}
\int_{x_0}^{x_1} f(x)\,\mathrm{d}x &= \\
&= \\
&\vdots \\
&=
\end{aligned}$$

Adding these equations gives

$$\int_{x_0}^{x_n} f(x)\,\mathrm{d}x = \tag{5.17}$$

Since f is C^2 on $[a, b]$, f'' is continuous. Let f'' attain the minimum and maximum values at x_m and x_M respectively...

Corollary. The Trapezium Rule is an order-_____ approximation.

Also note that the error vanishes when f is a linear function. This is consistent with the geometric picture – area under a straight line is *exactly* a trapezium.

Example 17. Evaluate

$$\int_1^2 \ln x \, \mathrm{d}x$$

to 4 S.F. using the Trapezium Rule with 4 strips. Give an upper and a lower bound on the error in this evaluation.
Is this bound consistent with the *actual* error?

Example 18 (Tutorial discussion)**.**

(a) Evaluate

$$\int_0^{\pi} \sin^2 x \, \mathrm{d}x \qquad (\star)$$

using the Trapezium Rule with 5 strips. Give your answer to 4 S.F.

(b) Without performing the actual integral, give an upper and a lower bound on the error in your answer in part (a).

(c) How many strips are needed so that the Trapezium-Rule estimate of the integral $(\star)$ is guaranteed to be accurate to 6 decimal places.

5.6 Simpson's Rule

In Simpson's Rule[2], the interpolation in each strip is done using a _____ drawn in the interval $[x_0, x_2]$ (where $x_2 - x_1 = x_1 - x_0 \equiv$ _____). Each strip has width _____. Recall from Chapter 4 that the lowest-order polynomial $P(x)$ which interpolates $f(x)$ at x_0, x_1 and x_2 can be written in Lagrange form as:

Example 19. Show that $\displaystyle\int_{x_0}^{x_2} P(x)\,\mathrm{d}x = \frac{h}{3}\left[y_0 + y_2 + 4y_1\right]$.

Proof. Exercise □

Using a similar technique as in the Trapezium Rule (but surprisingly *much* more difficult), we can derive the integrated error term. I'll state the result here as a Theorem.

Theorem 5.6 (Basic Simpson's Rule)**.** If f is C^4 on (x_0, x_2), then

$$\int_{x_0}^{x_2} f(x)\,\mathrm{d}x =$$

where $\xi \in (x_0, x_2)$.

You can see that this is a much "better" formula compared to the Basic Trapezium Rule (why?). Now let's upgrade the Basic Rule into a composite one.

Theorem 5.7 (Composite Simpson's Rule)**.** Let $a = x_0 < x_1 < x_2 \ldots < x_{2n} = b$, where the nodes x_i are evenly spaced, with $h =$ _____. We have:

$$\int_a^b f(x)\,\mathrm{d}x = \frac{h}{3}\left[y_0 + y_{2n} + 4\sum_{i=1}^{n} y_{2i-1} + 2\sum_{i=1}^{n-1} y_{2i}\right] -$$

where $\xi \in (a, b)$.

CAUTION: Although the Simpson's formula partitions the interval $[a, b]$ using $2n + 1$ values of x, we say that only "*n strips*" were used. Be careful!

[2]Thomas Simpson (1710-1761). English self-taught mathematician. His 'rule' was already known to Newton (and a number of previous mathematicians), as Simpson himself acknowledged.

Proof: As in the Trapezium Rule, we apply the Basic version in many strips and add them all up.

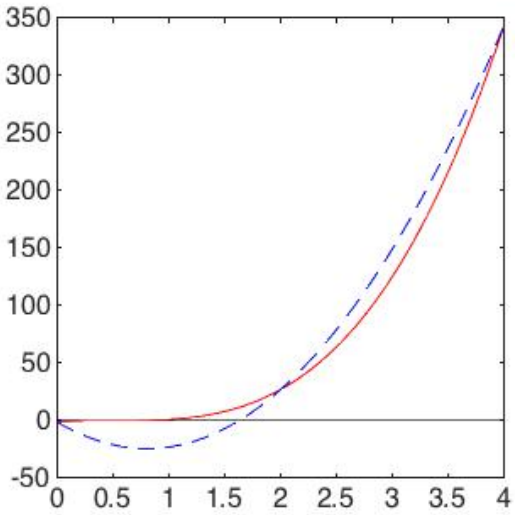
Corollary. The Simpson's Rule is an order-_____ approximation.

Example 20. A function f has the values shown below.

x	1	1.5	2	2.5	3
$f(x)$	1	$\frac{2}{3}$	$\frac{1}{2}$	$\frac{2}{5}$	$\frac{1}{3}$

Use Simpson's Rule to estimate the integral $\int_1^3 f(x)\,\mathrm{d}x$ using i) 1 strip, ii) 2 strips.

Example 21. Use Simpson's Rule with 1 strip to evaluate $\int_0^4 (2x-1)^3\,\mathrm{d}x$. Calculate the exact answer and comment on the result.



Example 22. Evaluate

$$\int_1^2 \ln x \, \mathrm{d}x$$

to 4 S.F. using the Simpson's Rule with 2 strips. Give an upper and a lower bound on the error in this evaluation. Compare the result with Example 17.
Is this bound consistent with the *actual* error?

Example 23. Estimate the number of strips, n, needed in the above Example in order to achieve an answer that is accurate to p decimal places. Give an upper and a lower bound for n when $p = 5$ and $p = 10$.

5.7 Romberg integration

We end this Chapter with the very popular and powerful *Romberg*[3] integration. The idea is to start with the usual Trapezium Rule, and improve on it iteratively via *Richardson extrapolation*, to achieve an arbitrarily high accuracy.

Previously, we have shown that Trapezium Rule is an order-_____ approximation (we always refer to the *Composite* Rule). We expect that Richardson extrapolation should eliminate the $\mathcal{O}(h^2)$ error term, and give us an order-_____ approximation.

But the good news is, actually, the next-order error is *order four*! It turns out that the error in the Trapezium Rule is actually a series in _____ powers of h, so eliminating the h^2 terms leaves us with the h^4 term (just like the symmetric-difference formula (5.15)). Let's state this very important fact as a theorem.

Theorem 5.8. The error in the Composite Trapezium Rule can be written as a series in even powers of h

$$\int_a^b f(x)\,\mathrm{d}x =$$

for some constants K_i's.

This is a surprisingly deep fact which isn't straightforward to prove, so I won't go over the proof here[4]. The exact values of the constants don't really matter to us here[5]. Only the powers of h (which determines the order of the approximation) are important.

Romberg integration starts with the calculation of Trapezium-Rule estimates.

Definition. R_{N1} denotes the Trapezium-Rule estimate of $\displaystyle\int_a^b f(x)\,\mathrm{d}x$ using 2^{N-1} strips.

For example, R_{11} uses one strip with width $h = b - a$. R_{21} uses 2 strips width _____. R_{31} uses _____ strips, width _____ *etc.* (doubling each time).

The goal of Romberg integration is to obtain the Romberg estimate R_{jk} for arbitrary j and k. Every R_{jk} is a possible answer to the integral, but differing in accuracy. The index _____ increases with the number of 'strips', and _____ increases with the 'order' of approximation. To get to higher orders, we use Richardson extrapolation...

[3] *Werner Romberg* (1909-2003), German mathematician who escaped Nazi Germany and settled in Norway. His integration method was based on previous work by Maclaurin and Huygens.

[4] see Cheney and Kincaid, page 220.

[5] The K_i's are related to the famous *Bernoulli's numbers*.

Example 24. R_{22} denotes the estimate of $I = \int_a^b f(x)\,\mathrm{d}x$ obtained by Richardson extrapolation of the order-2 estimates, R_{11} and R_{21}..

(a) Derive the expression for R_{22}. What is the order of this estimate?

(b) Deduce a similar formula for R_{32} and R_{N2}. What order are these estimates?

Remember: When the index j in R_{jk} increases by 1, the number of strips is _____. When the index k increases by 1, the order of the resulting formula increases by _____.

Example 25. Derive R_{33} in terms of R_{j2}. What is the order of this approximation?

Question. Write down a general recursive formula for the Romberg estimates R_{jk}.

(5.18)

The Romberg estimates are traditionally presented in a little *tableau*.

As we go down each column, the number of strips is doubled in each step. As we go from one column to the next, the accuracy is increased from $\mathcal{O}(h^k)$ to _____. Thus, given a finite tableau with α rows and β columns, the best estimate is _____.

In Romberg integration, always start off constructing the tableau with Trapezium Rule, then use (5.18) to get to higher orders. The formula will be provided in the exam.

Example 26. Construct the Romberg tableau up to R_{33} for $\displaystyle\int_1^3 \frac{1}{x}\,\mathrm{d}x$.
Leave all your answers as fractions.. [*Hint: For R_{j2}, check with Example 20.*]

Example 27. Let $I(h) = \int_a^b f(x)\,\mathrm{d}x$. Write down the Trapezium-Rule estimate using n strips, each of width $2h$, with the partition $a = x_0 < x_2 < \ldots < x_{2n} = b$.
Apply Richardson extrapolation to the resulting formula and comment on the result.

The diagram below summarises everything about Romberg integration and the significance of each entry.

Here is a geometric interpretation for R_{33}. The Romberg formula obtained can be shown to be equivalent to an approximation using an interpolating polynomial through _____ points (so clearly it is a _____ polynomial). It is an order-_____ estimate, known historically as *Boole's Rule*. *George Boole* (1815-1864) was an English mathematician who invented the symbolic logic which we now call *Boolean algebra*.

5.8 Epilogue

Numerical integration occupies a broad and profound part of the numerical-analysis canon – much more so than these lecture notes might suggest. By extending the basic materials presented here, it is possible to deal with the following problems numerically.

Integrals with problematic end-points: $\displaystyle\int_0^1 \frac{1}{\sqrt{x}}\,\mathrm{d}x,$

Improper integrals: $\displaystyle\int_0^\infty e^{-x^2}\,\mathrm{d}x,$

Multi-dimensional integrals: $\displaystyle\iiint_{\mathbb{R}^3} e^{-x^2-y^2-z^2}\,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z.$

Adaptive integration (used in MATLAB's `integral`): $\displaystyle\int_0^{10} \sin(x^2)\,\mathrm{d}x.$

If numerical analysis interests continues to interest you, I recommend choosing 3rd-year *Numerical Analysis* (32327), for which our module is not a pre-requisite, but will certainly give you a head start! The third-year course will be purely theoretical (no MATLAB!) and overlaps with roughly 10-20% of this course.

I hope that, at the very least, this course has given you the following:

- New insights into how computers work, and how various mathematical tasks can be optimized for accuracy and speed when implemented as codes.
- An appreciation that to understand computers, we need a deep understanding of a wide range of mathematical topics, from analysis to linear algebra.
- A vastly improved programming skill.