

# B1 Numerical Algorithms

4 lectures, MT 2023

Introduction

Wes Armour

23<sup>rd</sup> October 2023

# Course Overview

- 4 Lectures and 1 computational class.
- The focus of this course will be on the use of numerical methods in engineering science.
- Supporting material from courses:
  - P101 Calculus 1
  - P102 Calculus 2
  - P104 ODEs
  - A102 PDEs

# Acknowledgements

- Special thanks to Prof Ron Daniel who delivered this course up to 2012.  
*The lecture slides you have include some material prepared by Prof Daniel.*
- Special thanks to Professor Vicente Grau who lectured this course 2013-2017.  
*The lecture slides that I will present borrow some content (plots) from Prof Grau.*
- Special thanks to Professor Stephen Roberts who lectured “Engineering Computation”.  
*The lecture slides that I will present borrow some content from Prof Roberts.*

# Course Material

PDF copies of:

- These slides.
- Tutorial sheet.
- OLD lecture notes – use as a guide only!

Bonus videos on Panopto – playlist for each lecture on Canvas.

Useful material – weblinks on Canvas.

MATLAB codes used are also available on canvas.

If something is *really* not clear, and you are *really* stuck, email [wes.armour@eng.ox.ac.uk](mailto:wes.armour@eng.ox.ac.uk)


# Course Material

# Numerical representations and details

*Panopto bonus video*

## B1 Numerical Algorithms

4 Lectures, MT 2022  
Lecture one.  
*Bonus video – Rounding errors and storing numbers digitally*  
Wes Armour



Chris/Tom/Quentin re: [info@wesaudio.com](mailto:info@wesaudio.com)

### Bits, Bytes and nibbles

Before we think about storing continuous numbers digitally, lets look at how an integer number is stored.

Current computer hardware uses the **bit** as a mechanism to store numbers. It can take on two values: **0** or **1**.

A **byte** is a collection of **8 bits**, using an **octet**.

4 bytes (32 bits) are combined to form an **int**.

8 bytes (64 bits) are combined to form a **long int**.

To understand how this combining works, lets take a look at a **nibble** – a 4 bit integer.

*Note that computer programming languages and hardware sometimes employ different representations to those above. See here: [https://en.cppreference.com/ko/cpp/integer\\_computer\\_representation\\_common\\_integer\\_bits\\_bytes](https://en.cppreference.com/ko/cpp/integer_computer_representation_common_integer_bits_bytes)*

### The nibble

A nibble is a combination of 4 bits. Each bit can be either **0** or **1**, stored.

So how many different things can a nibble represent?


1 bit = things:  $1 + 1 = 2$   
3 bits = things:  $2^3 = 8$   
4 bits = things:  $2^4 = 16$

How do we use these combinations to represent integer numbers?

$$N_{bits} = b_{n-1} \cdot 2^{n-1} + b_{n-2} \cdot 2^{n-2} + \dots + b_1 \cdot 2^1 + b_0 \cdot 2^0$$

$$= 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 0 + 0 + 1 = 1$$

$$= 5$$

$$14_{10} \rightarrow 127_{10}, 17_{10}$$


### Storing continuous numbers digitally

All mainstream modern computing hardware implement the IEEE Standard for Floating Point Arithmetic (**IEEE754**).

This standard describes that everyone agrees on how to store a continuous number, how to perform arithmetic operations, how to round numbers (for example when adding them together, along with other things).

A floating point number is an approximate representation of a continuous number.

The term floating point reflects the fact that the **decimal point** in the representation can **float**. It can be placed anywhere within the significant digits of the number.

Let's look at this in some more detail.

1 \* 01:10

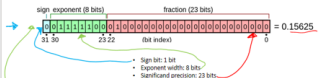
2 \* 01:28

3 \* 04:17

4 \* 01:23

### Floating points

A single precision floating point number is represented by 32 bits of information. It looks like:



- Sign bit: 1 bit
- Exponent width: 8 bits
- Significand precision: 23 bits


*IEEE 754 has different versions of "floats":*

- **Half precision** (16 bits: 1 sign, 5 exponent, 10 significand)
- **Single precision** (32 bits: 1 sign, 8 exponent, 23 significand)
- **Double precision** (64 bits: 1 sign, 11 exponent, 52 significand)

5 \* 01:32

### Floating points – example

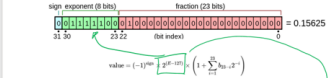
Let's think about our example below – how do we arrive at the number 0.15625?



6 \* 00:53

### Floating points – example

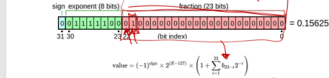
Let's think about our example below – how do we arrive at the number 0.15625?



7 \* 02:29

### Floating points – example

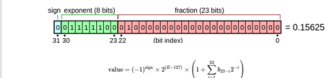
Let's think about our example below – how do we arrive at the number 0.15625?



8 \* 02:18

### Floating points – example

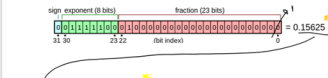
Let's think about our example below – how do we arrive at the number 0.15625?



9 \* 01:24

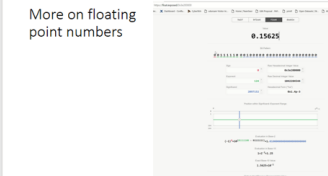
### Gapping

What is the number "next to" 0.15625?



10 \* 02:09

### More on floating point numbers



11

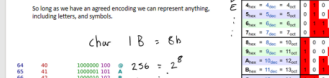
### What about letters, symbols...?

As previously mentioned, combinations of bits can represent "things". So long as we have an agreed encoding we can represent anything, including letters, and symbols.

char 1B = 8b

256 = 2^8

ASCII



12 \* 01:20

# Reading

- Wikipedia has some excellent material that can be extremely helpful.
- G. Wheatley, “Applied Numerical Analysis – 7th Editions”, Pearson International Edition.
- R.L Burden, J.D. Faires, “Numerical Analysis – 8th Edition”, Thomson International Student Edition.
- F B Hildebrand, “Introduction to Numerical Analysis – 2nd Edition”, Dover reprint.

## *The Classics...*

- ❑ *Press, Teukolsky, et. al., “Numerical Recipes 3rd Edition: The Art of Scientific Computing” (2007).*
- ❑ *Froberg, “Introduction to Numerical Analysis” (2nd edition): Addison Wesley (1973).*

## *A hands on text...*

- ❖ *Kiusalaas, “Numerical Methods in Engineering with Matlab/Python...”, Cambridge (2016).*

# Learning Outcomes

- Finite difference approximations (*numerical differentiation...*) (Lecture 1)
  - forward, backward and central difference equations,
  - formulae for more than one variable,
  - formulae for higher order derivatives.
- Numerical quadrature (*also known as integration...*) (Lecture 1)
  - trapezium rule,
  - Simpson's rule,
  - sampling methods.
- Sources of error and convergence (Lecture 1)
  - round-off error,
  - truncation error,
  - order of an algorithm,
  - Taylor series analysis,

# Learning Outcomes

- Function fitting and approximation (Lecture 2)
  - Linear regression,
  - polynomial regression,
  - condition number,
  - Richardson's method,
  - differentiation and noise.
- Numerical integration of ODEs (Lecture 3)
  - predictor-corrector methods,
  - modified Euler method,
  - Runge-Kutta methods,
  - adaptive step-size control (*Adams-Bashforth / Adams-Moulton*).
- Linear PDE solution by finite differences (Lecture 4)
  - Elliptic (*Laplace*),
  - Parabolic (*diffusion*),
  - Hyperbolic (*wave*).



# Finally...

Remember, if something is *really* not clear, and you are *really* stuck, email [wes.armour@eng.ox.ac.uk](mailto:wes.armour@eng.ox.ac.uk)

I do hope you find the course interesting and enjoyable!