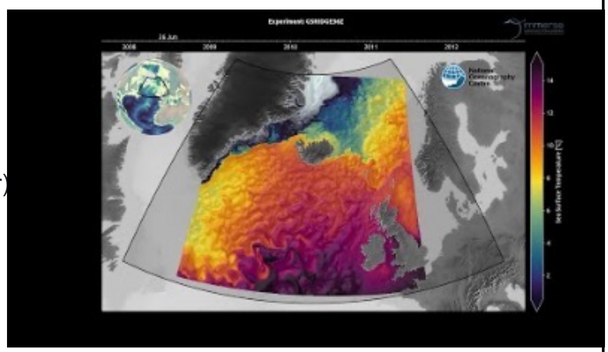
Numerical Algorithms for HPC 2022/23

Course introduction

1

Introduction to the course

- Course covers most common algorithms or computational patterns in scientific computing
- Many algorithms common across different applications
- Which, when, why (why not)
- Serial and in parallel
- Algorithmic complexity
- Errors
- Verification



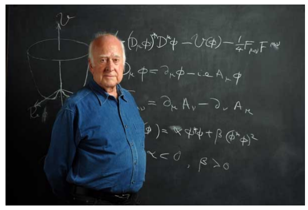
Sea surface temperature (colour) and ice concentration (grey shades) in the Greenland-Scotland Ridge region.
Dr Mattia Almansi, Marine Systems Modelling - National Oceoanography Centre

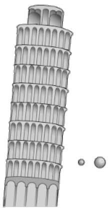
2

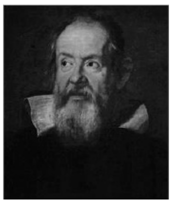
2

The scientific method

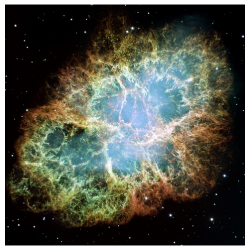
Theory: Mathematical equations provide a description or model





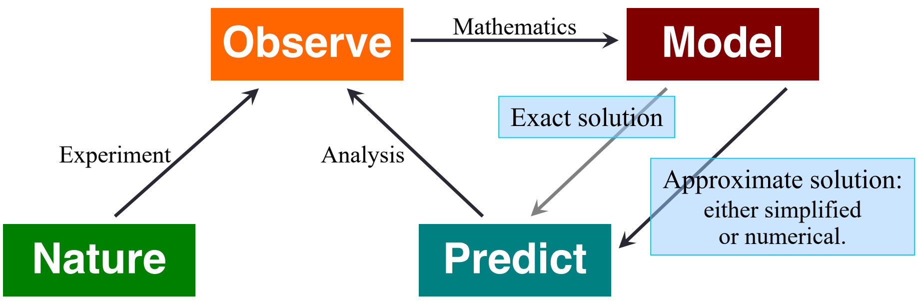


- Experiment
  - Inference from data
  - Test hypothesis



- Computer simulation
  - Too big, small, difficult or dangerous for standard methods
  - Explore the space of solutions

Computer Simulation in Science



What is tractable?

- How big a problem can we tackle?
  - On standard computers
    - Run out of time
    - Run out of space
  - On state-of-the-art machines
    - Run out of time
    - Run out of space
    - Run out of precision

Scientific computations are limited by available computing resources

Numerical algorithms in parallel?

- Majority of scientific problems can be considered 'numerical algorithms'
  - Designing buildings, cars, aeroplanes, pipes, efficient use of materials, weather, membrane simulation, material modelling, chemical reactions
- Solve equations and systems of equations
  - Matrices (Gaussian elimination, LU decomposition etc.)
  - Iterative methods (Jacobi, Gauss-Seidel, successive over-relaxation, conjugate gradient)
  - Eigenvalue decomposition [Not covered in course]
- *Parallelisation* allows you to do more
  - Increases your memory and flop limit
  - Introduces additional complexity and overhead
  - Allows us to solve same, or bigger, problems faster

Parallel and serial algorithms

- Majority of the course is based on serial algorithms
  - But usually look at what needs to be considered in parallel
  - Usually worth considering parallel aspects at the algorithm design stage
- Will get a chance to try some exercises in parallel
- Course was previously known as Parallel Numerical Algorithms (PNA)
  - But emphasis not on parallel aspects
  - Now known as Numerical Algorithms for HPC (NAHPC)
  - You may occasionally see references to PNA
  - Also past exams will all be from Parallel Numerical Algorithms (PNA)
    - Much earlier ones from Applied Numerical Algorithms (ANA)

epcc

7

7

Example maths in the course

- Significant mathematical content in slides…
- …but always as simple as possible to understand numerical algorithms
- For example, will see summations: $\sum_{i=1}^{n} a_i$
  - Just means $a_1 + a_2 + \ldots + a_n$
  - In computing terms, think of a loop:

```
ans = 0.0
do i = 1 to n
        ans = ans + a[i]
end do
print ans
```

Discrete Fourier transform:

$F[n]$ $f[k]$

$$F_n = \sum_{k=0}^{N-1} f_k e^{2\pi ikn/N}$$

k = loop variable

epcc

8

8

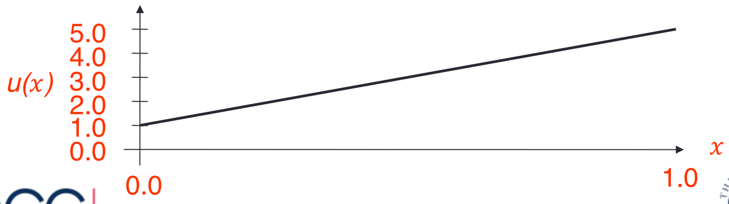
Second example: slide from PDE lectures

- Differential equation involving density of pollution $u(x)$ at point x is given by:

$$-\frac{d^2}{dx^2}u(x) = 0$$

$\frac{d}{dx}$ means find "gradient of…"
$\frac{d^2}{dx^2}$ means find "gradient of gradient of…"

- Its solution is $u(x) = mx + c$ [given in lecture – you don't have to derive this!]
- This results in a straight line, e.g.



9

Course structure

- ~11 weeks – 2 lectures per week + 1 practical class
  - Monday at 11:10-12:00 - Lecture
  - Tuesday at 12:10-13:00 – Practical
  - Thursday at 14:10-15:00 – Lecture
- Lectures
  - Slides available in Learn before class
    - Also recorded
  - Download if desired and annotate and elaborate
- Exercise classes
  - Available in Learn before class
  - You are expected to start exercises in each practical session
    - Complete the exercises in your own time
    - Help available during and after the sessions from lecturers and tutors
- Assessment is
  - One class test on Tuesday 18 October (week 5) worth 15%.
  - Exam worth 85% at end of semester 1 with 3 questions, equally weighted.

Learn

10

10

Lecturers

- Three lecturers
  - William Lucas (Course organiser, Matrices, PDEs, FFTs)
    - w.lucas@epcc.ed.ac.uk
  - Arno Proeme (Molecular Dynamics, Many body systems)
    - a.proeme@epcc.ed.ac.uk
  - David Henty (Random numbers - Monte Carlo)
    - d.henty@epcc.ed.ac.uk
- Class tutor
  - Eleanor Broadway
    - e.broadway@epcc.ed.ac.uk
- All approachable and friendly!
  - Happy to discuss material in and out of lectures

epcc

Timetable

- Full timetable in Learn
- Always check latest version!

- Introduction/Floating Point numbers/errors:
  - 3 lectures, 1 exercise (William L)
- Dense linear algebra
  - 2 lectures, 1 exercise (William L)
- PDEs
  - 3 lectures, 2 exercises (William L)
- Sparse linear algebra/Krylov subspaces
  - 4 lectures, 2 exercises (William L)
- Fast Fourier Transforms
  - 3 lectures, 1 exercise (William L)
- N-body methods/Molecular Dynamics
  - 3 lectures, 1 exercise (Arno P)
- Monte Carlo/random numbers
  - 2 lectures, 1 exercise (David H)

epcc

Exercise class problems

- Problems can be done in C or Fortran
  - Only some can be done in Java
  - competent Java programmer should be able to work in C
  - demonstrate and develop ideas essential to the course
- Mathematical proofs are not examined for this course, but you are expected to perform simple algebraic manipulation
  - See previous exams
- Work on the exercises
  - Exam is based on material from lectures AND exercises!

```fortran
Subroutine times_vector_CSR(M, x, y)
  ! This routine calculates y = M*x for CSR-matrix m and vectors x and y
  type(csr), intent(in)                                :: M
  real(kind=16), dimension(M%ncol), intent(in)         :: x
  real(kind=16), dimension(M%nrow), intent(out)        :: y
  integer                                              :: row, col, ctr

  ! zero the vector before multiplying
  y = 0.0
  ctr = 1

  do row = 1,M%nrow
     do col = M%rowStart(row), M%rowStart(row+1)-1
        y(row) = y(row) + M%value(ctr)*x(M%colIndex(ctr))
        ctr = ctr + 1
     end do
  end do

end Subroutine times_vector_CSR
```

epcc

Representing Numbers of a Computer, Part 1

How computers store real numbers
and the problems that result

epcc

Overview

- Integers
- Reals, floats, doubles, etc.
- Arithmetical operations and rounding errors

- In a code we may write:

```
x = sqrt(2.0)
```

  - but how is this stored?

epcc

Mathematics vs Computers

- Mathematics is an ideal world
  - integers and real numbers can be as large as you want
  - real numbers can be as large or as small as you want
  - can represent every number exactly:

$$1, -3, 1/3, 10^{36237}, 10^{-232322}, \sqrt{2}, \pi, ...$$

- In mathematics numbers range from $-\infty$ to $+\infty$
  - there is an infinite number of real numbers in any interval
- This not true on a computer
  - numbers have a limited range (integers and real numbers)
  - limited precision (real numbers)

epcc

Integers

- In the real world, we like to use **base 10**
  - we only write the 10 characters 0,1,2,3,4,5,6,7,8,9
  - use *position* to represent each power of 10

$$125 = \mathbf{1} * 10^2 + \mathbf{2} * 10^1 + \mathbf{5} * 10^0$$
$$= \mathbf{1}*100 + \mathbf{2}*10 + \mathbf{5}*1 = \mathbf{125}$$

  - represent positive or negative using a leading "+" or "-"
- Computers are binary machines
  - can only store ones and zeros
  - minimum storage unit is 8 bits = 1 byte
- Use **base 2**

$$1111101 = \mathbf{1}*2^6 + \mathbf{1}*2^5 + \mathbf{1}*2^4 + \mathbf{1}*2^3 + \mathbf{1}*2^2 + \mathbf{0}*2^1 + \mathbf{1}*2^0$$
$$= \mathbf{1}*64 + \mathbf{1}*32 + \mathbf{1}*16 + \mathbf{1}*8 + \mathbf{1}*4 + \mathbf{0}*2 + \mathbf{1}*1$$
$$= \mathbf{125}$$

epcc 17

17

Long addition

- Binary arithmetic works the same as decimal (but simpler!)
- E.g. 'long' addition (assume 3 "digits" or "bits" available)

```
      538
   +  695
     1233
      1 1
Overflow digit   Carry digits
```

```
      111
   +  101
     1100
     1 11
Overflow bit   Carry bits
```

- If only 3 digits available can only store 0-999 so 1233 above would cause overflow
  - Likewise 1100 for binary if only 3 bits available
  - We need to decide what to do when we get overflow…
- Subtraction, multiplication, division, etc. all work the same as decimal

epcc 18

18

Storage and Range

- Assume we reserve 1 byte (8 bits) for positive integers
  - minimum value `0`
  - maximum value $2^8 - 1 = 255$
  - if result is out of range we will overflow and get wrong answer!

- Standard storage is 4 bytes = 32 bits
  - minimum value `0`
  - maximum value $2^{32} - 1 = 4294967295 \approx 4G \approx 4$ billion
- Is this a problem?
- If so, can use 8 bytes (64-bit integers)
  - minimum value `0`
  - maximum value $2^{64} - 1 = 18446744073709551616 \approx 10^{19}$
- Not directly related to whether you have 32-bit or 64-bit operating system

epcc 19

19

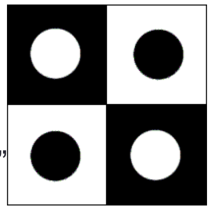
Negative numbers – signed bit

- To store -ve numbers could simply reserve 1st bit for the sign
  - 0 for a positive number
  - 1 for a negative number
- E.g. using 3 bits
  - 1st bit for sign
  - Next 2 bits for the magnitude
  - Decimal number 3 stored as binary 011
  - Decimal number -3 stored as binary 111
- For 8 bits, range is now: -127 to +127
  - Was 0 to +255 when only allowing positive numbers
  - Now includes 2 zeros (+0 and -0) but only one kind of zero in real world!
- Simple, easy to see magnitude of number…
- …but not most efficient scheme for addition

epcc 20

20

Two's Complement (not examinable)



- Use "two's complement" representation
  - flip all ones to zeros and zeros to ones
  - then add one (ignoring overflow)
- Result: negative integers have the first bit set to "1"
  - As with signed bit, for 8 bits, range is now: -128 to + 127
  - normal addition (ignoring overflow) gives the correct answer

```
00000011 = 3
11111100
00000001
11111101 = -3
```

flip the bits

add 1

```
125 - 3 =
125 + (-3) = 01111101 + 111111101 = 01111010  = 122
```

21

21

Example binary to decimal with 2 bits

Binary number	Straightforward Decimal equivalent	Decimal equivalent with shift (offset)	Sign bit	Two's complement
00	0	-1	+0	0
01	1	0	+1	1
10	2	1	-0	-2
11	3	2	-1	-1
Range	[0,3]	[-1,2]	[-1,1]	[-2,1]
	Straightforward but only gives positive integers (e.g. unsigned integers)	Used later for exponent in FP numbers	Note you get two kinds of 0. Again used for FP numbers	Addition just works! Used for integers

- Turns out there are several ways to map!
  - Most common ones shown above
- Each of these mappings are used at some point

22

22

Integer Arithmetic

- Computers are brilliant at integer maths
- Integers can be added, subtracted and multiplied with complete accuracy…
  - …as long as the final result is not too large in magnitude
- But what about division?
  - 4/2 = 2, 27/3 = 9, but 7/3 = 2 (instead of 2.3333333333333…).
  - what do we do with numbers like that?
  - how do we store real numbers?

epcc 23

23

Fixed-point Arithmetic

- Can use an integer to represent a real number.
  - we have 8 bits stored in X 0-255.
  - represent real number a between 0.0 and 1.0 by dividing by 256
  - We're simply saying
    - Smallest number 0 represents $0/256 = 0$
    - 1 represents $1/256$, 2 represents $2/256$, 3 represents $3/256$, etc.
    - Largest number 255 represents $255/256 = .99609375 \approx 1$
  - e.g. $a = 5/9 = 0.55555$ represented as $X = 142$
    - $142/256 = 0.5546875$
- Operations now treat integers as fractions:
  - If you want to calculate $c = a \times b$ where a, b, and c are all fractions between 0.0 and 1.0
  - Set X = integer($a \times 256$), Y = integer($b \times 256$)
  - Result Z = integer(c x 256)
    - E.g. $c = a \times b$ becomes $c = (256a \times 256b)/256$, i.e. $Z = X \times Y/256$
  - Between the upper and lower limits (0.0 to 1.0), we have a uniform grid of possible 'real' numbers.

epcc 24

24

Problems with Fixed Point

- This arithmetic is very fast
  - but does not cope with large ranges
  - E.g. above, cannot represent numbers < 0 or numbers >= 1
- Can adjust the range
  - but at the cost of precision

epcc 25

25

Scientific Notation (in Decimal)

- How do we store 4261700.0 and 0.042617
  - in the same storage scheme?
- In last 2 slides, decimal point was *fixed*
  - now let it *float* as appropriate
- E.g. in *scientific notation* above becomes 4.2617×10^6 and 4.2617×10^{-2}
  - or sometimes written as $4.2617\text{E+}6$ or $4.2617\text{E-}2$
- We will shift the decimal place so that it is always at the start but just after the decimal point
  - i.e. 0.42617 (42617 here is known as the **mantissa** m)
- Remember how many places we have to shift
  - i.e. $+7$ or -1 (+7 or -1 known as the the **exponent** e)
- Actual number is of the form $0.mmmm \times 10^e$
  - i.e. 0.4262×10^7 or 0.4262×10^{-1}
    - here we have 4 digits available for $mmmm$ and 1 for e
    - always use all 5 numbers - don't waste space storing leading zero!
    - automatically adjusts to the magnitude of the number being stored
    - could have chosen to use 2 spaces for e to cope with very small/large numbers

epcc 26

26

Floating-Point Numbers

$$0 \bullet \boxed{m}\ \boxed{m}\ \boxed{m}\ \boxed{m} \times 10^{\boxed{e}}$$

- Decimal point "floats" left and right as required
  - fixed-point numbers have constant absolute error, e.g. ± 0.00001
  - floating-point have a constant relative error, e.g. $\pm 0.001\%$
- Computer storage of real numbers directly analogous to scientific notation
  - except using binary representation not decimal
  - ... with a few subtleties regarding sign of m and e
- All modern processors are designed to deal with floating-point numbers *directly in hardware*

epcc 27

27

The IEEE 754 Standard

- Mantissa made positive or negative:
  - the first bit indicates the sign: 0 = positive and 1 = negative.
- General binary format is:

Highest Bit	1	10010101	10000010000011101000100	Lowest Bit
	Sign	Exponent	Mantissa	

- Exponent made positive or negative using a "biased" or "shifted" representation:
  - If the stored exponent, c, is X bits long, then the actual exponent is $c - \mathit{bias}$ where the offset *bias* = $(2^X/2 - 1)$. e.g. $X = 3$:

Stored (c,binary)	000	001	010	011	100	101	110	111
Stored (c,decimal)	0	1	2	3	4	5	6	7
Represents ($c - 3$)	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**

epcc 28

28

IEEE – The Hidden Bit

- In base 10 exponent-mantissa notation:
  - we chose to standardise the mantissa so that it always lies in the range $0.0 \leq m < 1.0$
  - the first digit is always 0, so there is no need to write it.
  - However, first non-zero digit could be any digit between 1 and 9
- The FP mantissa is "normalised" to lie in the **binary** range:

  $1.0 \leq m < 10.0$ i.e. decimal range $1.0 \leq m < 2.0$

  - The first bit is therefore **always one** **so** there is no need to store it!
  - We only store the variable part, called the significand (f).
  - the mantissa m = 1.f (in binary), and the 1 is called "The Hidden Bit":
  - However, this does mean that zero requires special treatment
    - having f and e as all zeros is defined to be (+/-) zero.

epcc 29

29

Binary Fractions: what does 1.f mean?

- Whole numbers are straightforward
  - base 10: $109 = 1*10^2 + 0*10^1 + 9*10^0 = 1*100 + 0*10 + 9*1 = 109$
  - base 2: $1101101 = 1*2^6+1*2^5+0*2^4+1*2^3+1*2^2+0*2^1+1*2^0$

    $= 1*64 + 1*32 + 0*16 + 1*8 + 1*4 + 0*2 + 1*1$

    $= 64 + 32 + 8 + 4 + 1 = 109$
- Simple extension to fractions

  $109.625 = 1*10^2 + 0*10^1 + 9*10^0 + 6*10^{-1} + 2*10^{-2} + 5*10^{-3}$

  $= 1*100 + 0*10 + 9*1 + 6*0.1 + 2*0.01 + 5*0.001$

  $1101101.101 = 109 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}$

  $= 109 + 1*(1/2) + 0*(1/4) + 1*(1/8)$

  $= 109 + 0.5 + 0.125$

  $= 109.625$

epcc 30

30

Summary

- Looking at how to store integers
  - Various options for negative numbers
- Real numbers stored in floating-point format
- Floating point numbers conform to IEEE 754 standard
- Lots of issues to be aware of… see next lecture!

epcc

31

31