Live Lecture 1

Computational Physics 301

Overview

- Introduction to the unit
- Simultaneous Equations
  - Computational efficiency
  - Algorithmic complexity
- Eigenproblems
  - Iteration
  - Finite Difference Equations

Computational Physics 301

- **Topics**
  - Linear Algebra
  - Partial Differential Equations
  - Monte Carlo methods
- **Video lectures**
  - Focus on the mathematical & algorithmic background
- **Jupyter notebooks**
  - Practical examples
  - Programming techniques
  - Algorithm implementation & testing/validation
- **Exercises**
  - Solving physics problems using numerical algorithms

Schedule

TB1

Week	1	2	3	4	5	6	7	8	9	10	11	12
Content/Lectures							Linear Algebra					
Drop-in sessions								Mon/Fri	Mon/Fri	Mon/Fri		
Assignment								Ex1	Ex1	Ex1	deadline Thurs 12:30pm	
Feedback												

TB2

Week	13	14	15	16	17	18	19	20	21	22	23	24
Content/Lectures	Partial Differential Equations					Monte Carlo Methods						
Drop-in sessions	Fri	Mon/Fri	Mon/Fri	Mon		Fri	Mon/Fri	Mon/Fri	Mon			
Assignment	Ex2	Ex2	Ex2	deadline Thurs 12:30pm		Ex3	Ex3	Ex3	deadline Thurs 12:30pm			
Feedback	Ex1					Ex2				Ex3		

Coursework

- Unit is assessed by coursework
  - One assignment per topic, released alongside topic content
  - Each assignments consist of a **Jupyter notebook**
  - Each assignment carries equal weight

- **“Assessment Info”** video on Blackboard
  - Details of the marking & moderation process
  - Marking criteria
  - Tips on how to submit a good assignment

Getting Help

- **Drop-in sessions**
  - Teaching assistants on hand to give help
  - Programming issues, conceptual problems, debugging, etc.
- **Teams**
  - *“PHYS38012: Computational Physics 301 2022/23 (TB-4, A)”*
  - For help during and outside drop-in session times
  - Please feel free to post questions
- **Email**
  - Alan Reynolds & Henning Flaecher during TB1
  - Jim Brooke during TB2

Linear Algebra

Simultaneous Equations

- A large class of problems… often written as a matrix equation

$$ax_1 + bx_2 = y_1$$
$$cx_1 + dx_2 = y_2$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix} \qquad A\underline{x} = \underline{y}$$

- Numerical methods are unsurprisingly useful for large matrices

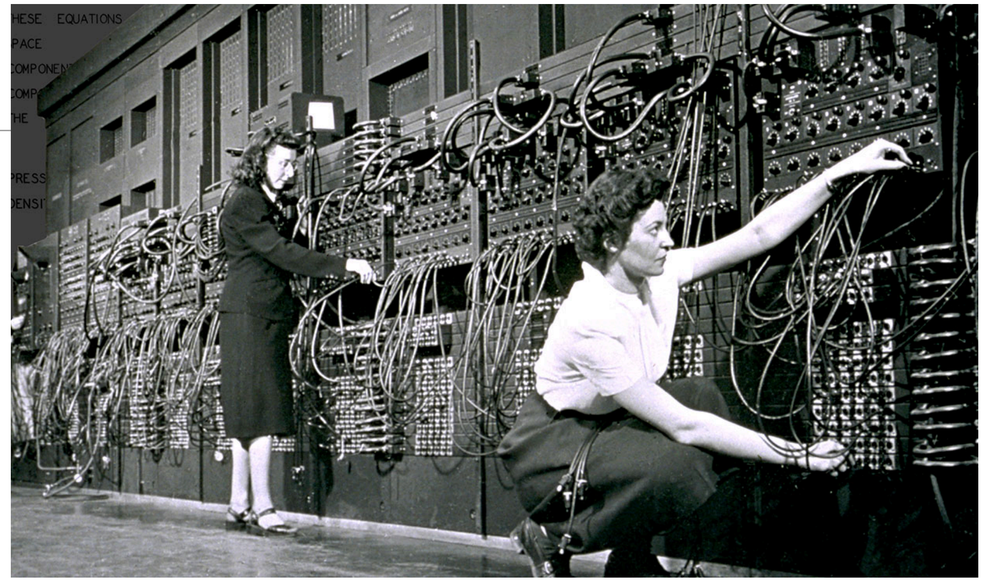
- Lecture 1 introduces several methods for solving this kind of problem :
  - Matrix inversion
  - Gaussian elimination
  - LU decomposition
  - SVD decomposition

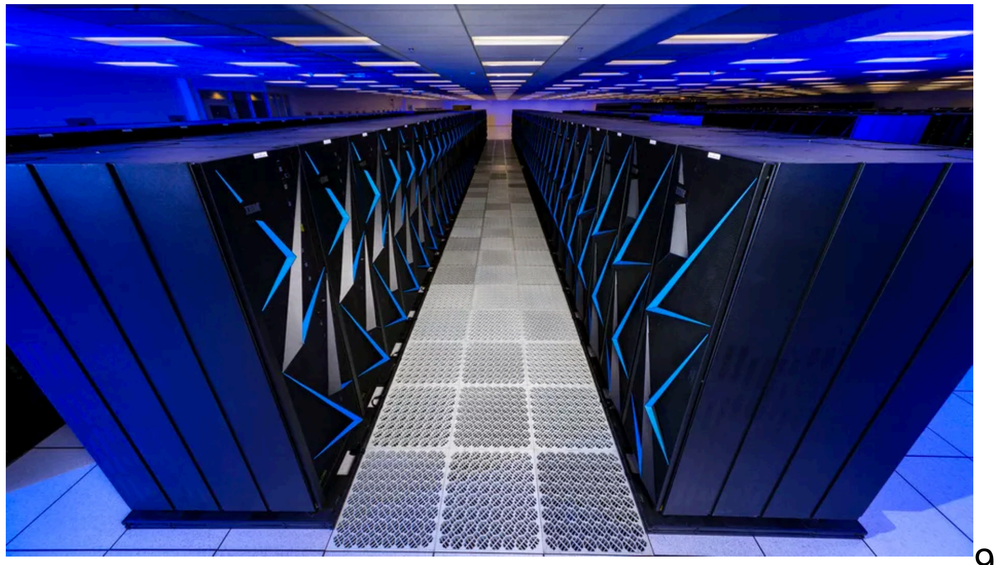
Linear Algebra 1 notebook examines differences in terms of numerical output

Here we will look at differences in terms of computation time

Computation Time

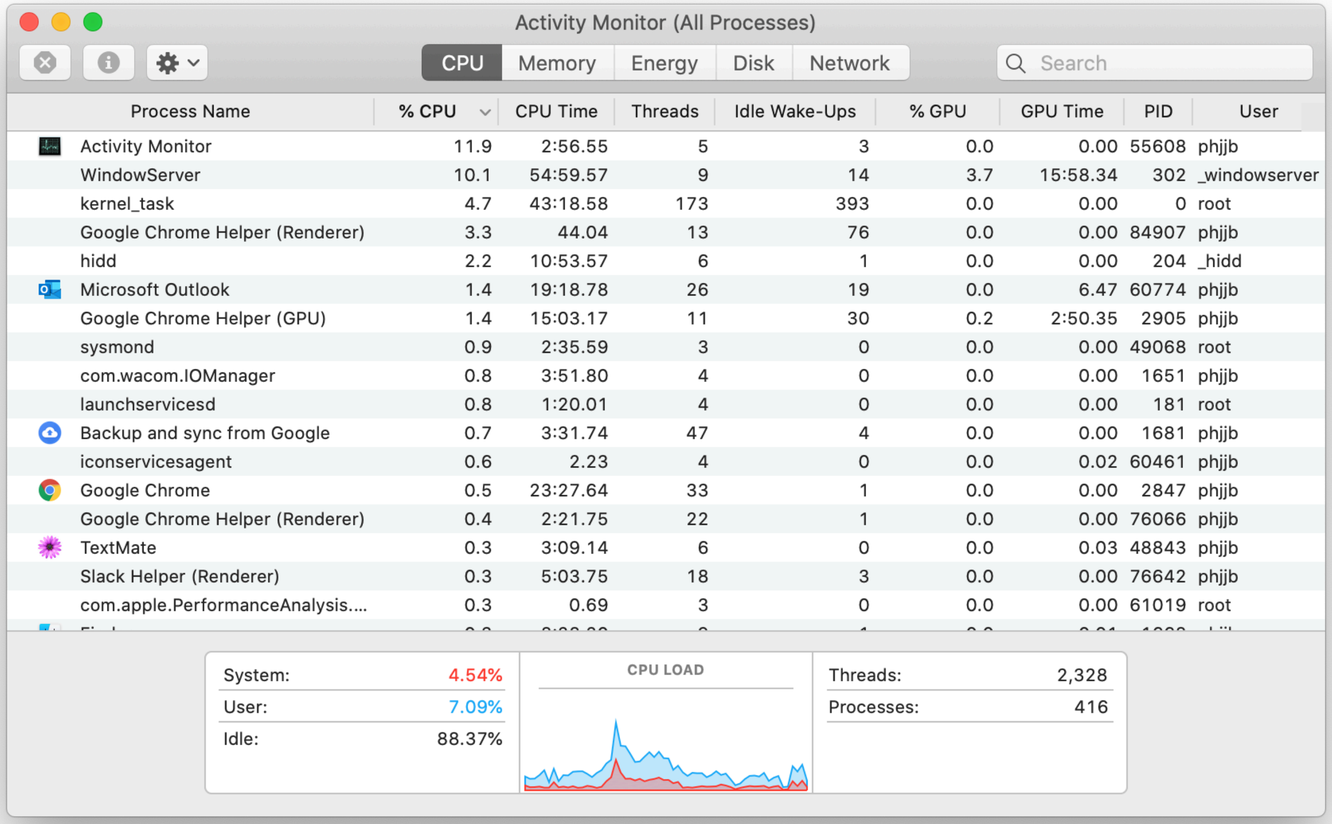
- Time taken to perform a calculation depends on several factors
  - Complexity of our algorithm
  - Efficiency of our code
  - Computer power/architecture
- Measuring computation time is not always straightforward





Measuring Execution Time

- **Computer has an internal clock**
  - We can use this to estimate the time required by a function, but…
- **A typical cpu will be running multiple threads in parallel**
  - Using the ‘wall clock’ time does not account for time the cpu spent doing other things
- **Clock resolution is not that great**
  - Around 0.1s, but machine dependent
  - Need to measure time intervals that are >> resolution



What my cpu was doing when I wrote this slide (not very much, to be fair…)

Algorithmic Complexity

- Consider a simple algorithm : *sum over an N-element array*
- Implement this by storing the sum in an accumulator, iterating over every element of the array, and adding it to the accumulator

$$s = \sum_{i=0}^{N} a_i$$

- Try to break this down into individual cpu operations
  - Each run through the loop requires it to :
    - Increment ***i***
    - Retrieve a_i from memory
    - Do an addition $s+a_i$
    - Store accumulator ***s***

```
1  def sum(a):
2      s=0
3      for i in range(N):
4          s += a[i]
5      return s
```

- The total number of cpu operations is something like $4N$

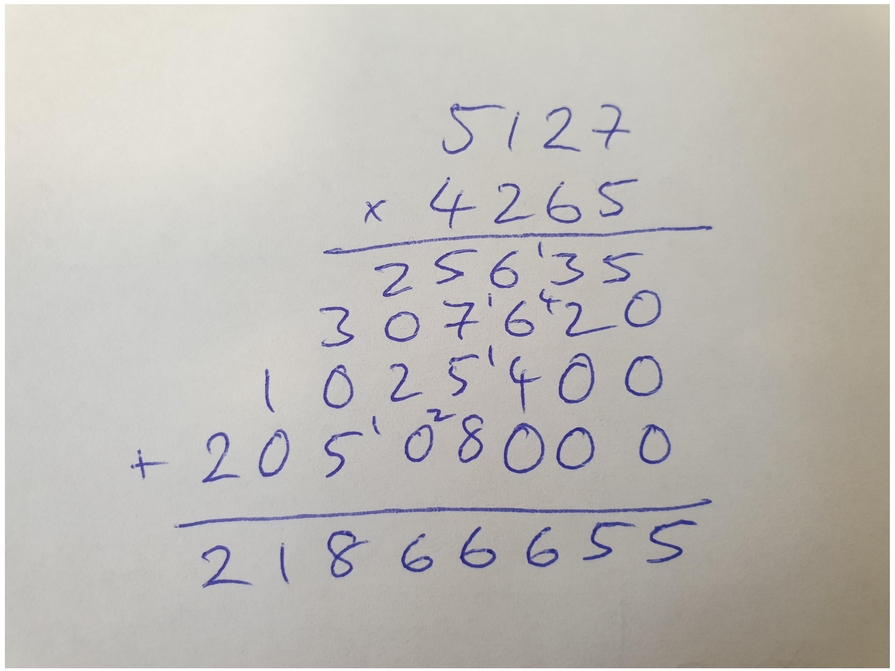
The number of operations is linear with N

The "complexity" is $\mathcal{O}(N)$

(But note that we didn't consider the number of digits here…)

Algorithmic Complexity

- Addition is straightforward
  - Two *n*-digit numbers can be added in $\mathcal{O}(n)$ operations
- Multiplication is more complicated
  - Consider multiplying two *n*-digit numbers using “long multiplication”
  - This requires $\mathcal{O}(n^2)$ operations
- But there are better algorithms!
  - eg. Karatsuba multiplication is $\mathcal{O}(n^{1.585})$
  - An example of ‘divide and conquer’



Efficient multiplication algorithms is really just the beginning…

Algorithmic Complexity

- What is the complexity of Cramer's rule ? $A^{-1} = \frac{1}{\det A} C^T$
  - Recall that $\det_{N \times N}$ depends on N determinants : $\det_{N-1 \times N-1}$
  - So the determinant term requires $\mathcal{O}(N!)$ operations
- What about LU and SV decomposition ?
  - The calculation is beyond the scope of this unit, but you can look it up
  - Both require $\mathcal{O}(N^3)$ operations

Eigenproblems

- Goal is to find the solutions (ie. eigenvalues λ and eigenvectors $\underline{u}$) of

$$(A - \lambda I)\underline{u} = 0$$

- Linear Algebra 2 lecture & notebook introduce some methods :
  - Power iteration
  - Rayleigh quotient iteration
  - Hotelling's deflation

- Two classes of problem :
  - Eigenfunction form is known - see example in Linear Algebra 2 notebook
  - Eigenfunction form is unknown - need to use numerical approximation, eg. Exercise 1

Iteration

- Any method which involves repeatedly apply an equation
- Typically, iteration is defined using some state, X, and a function $f(X)$:

$$X_{i+1} = f(X_i)$$

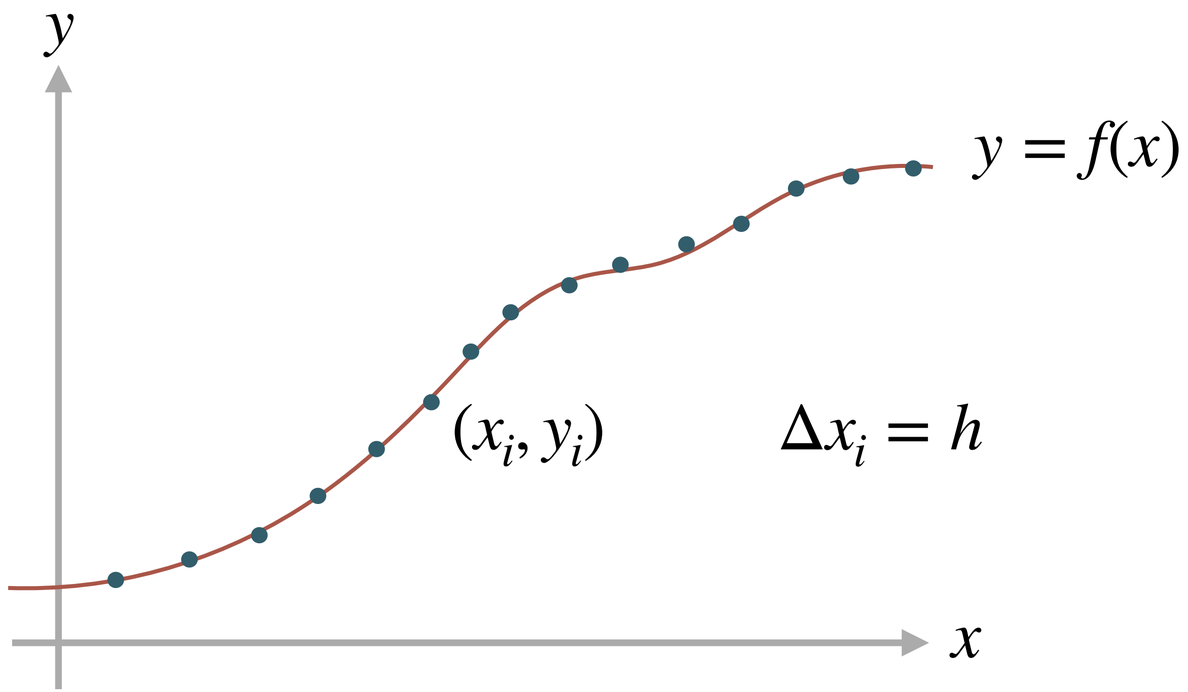
- The iteration process :
  - Start with a guess for X (possibly an educated one)
  - Apply $f(X)$ repeatedly
  - Test for convergence, eg. $\Delta X = X_{i+1} - X_i < Y$ and stop
    - But must also include some mechanism to handle divergence !

Finite Difference Methods

- **This will be covered in more detail in the next topic (PDEs)**
  - This is a very brief introduction, since they are used in Ex 2
- **The eigenproblem in the Linear Algebra 1 notebook uses a particular basis**
  - Assume solutions can be written as linear combinations on the basis
  - Matrix order corresponds to the number of degrees of freedom
- **The eigenproblems in Exercise 2 used Finite Difference Methods**
  - Assume solutions can be approximated on a discrete set of points
  - Matrix order corresponds to the number of points

Finite Difference Methods

- Approximate a solution as set of points on a grid
  - Here, define a regular spaced set of x values, x_i, and estimate values of the solution at each point, y_i



Finite Difference Equations using Taylor Expansion

- Take a Taylor expansion of $f(x)$ about $(x+h)$:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \ldots$$

- Re-arranging gives a ‘forward difference’ approximation :

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h^2)$$

- Similarly, we can obtain the backward difference approximation

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \mathcal{O}(h^2)$$

- These are both *1st order* approximations, since error is proportional to h^2

Finite Difference Equations using Taylor Expansion

- For 2nd derivative, we sum Taylor expansions about $(x+h)$ and $(x-h)$ to obtain

$$f(x-h)+f(x+h)=2f(x)+h^2f''(x)+\mathcal{O}(h^4)$$

- Since the odd-powered terms in h cancel

- This gives

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

Finite Difference Equations

- Of course, $x_i + h = x_{i+1}$ and $x_i - h = x_{i-1}$, so the 1st derivatives can be written :

$$f'(x_i) = \frac{f(x_{i+1}) - f(x_i)}{h} + \mathcal{O}(h^2) \quad \text{and} \quad f'(x_i) = \frac{f(x_i) - f(x_{i-1})}{h} + \mathcal{O}(h^2)$$

- And the 2nd derivative can be written :

$$f''(x_i) = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} + \mathcal{O}(h^4)$$

Used in Ex 2 to construct the Hamiltonian

Finite Difference Equations

- Useful technique for numerical approximation to unknown functions
  - Especially when analytic solutions cannot be found !
- Approximate derivatives using Taylor expansion
  - Choice of precision - how many terms in the expansion to include
  - Typically have some choices about whether to use forward/backward differences - can have an impact depending on the problem/technique
- Method can be extended to expressions with multiple dimensions/variables