L2 Computational Physics
Week 1 - Introduction

- To view slide-by-slide animations in this PDF, do not view it in your web browser
  - Save it to disk
  - View “Full Screen”
    - Acrobat/Windows [CTRL+L]
    - “View / Enter Full Screen” in OS X/Previews
    - Move between pages with arrow keys

Lecture 1 Overview

- What is...
  - a Computer?
  - Computational Physics?
  - Programming?
- Course Information
  - Course Structure
  - Learning Outcomes
  - Weekly Assessments
  - Getting the most out of the lectures
- General Background
  - Languages
  - Symbolic Maths
  - Speed
  - Accuracy

Week 1

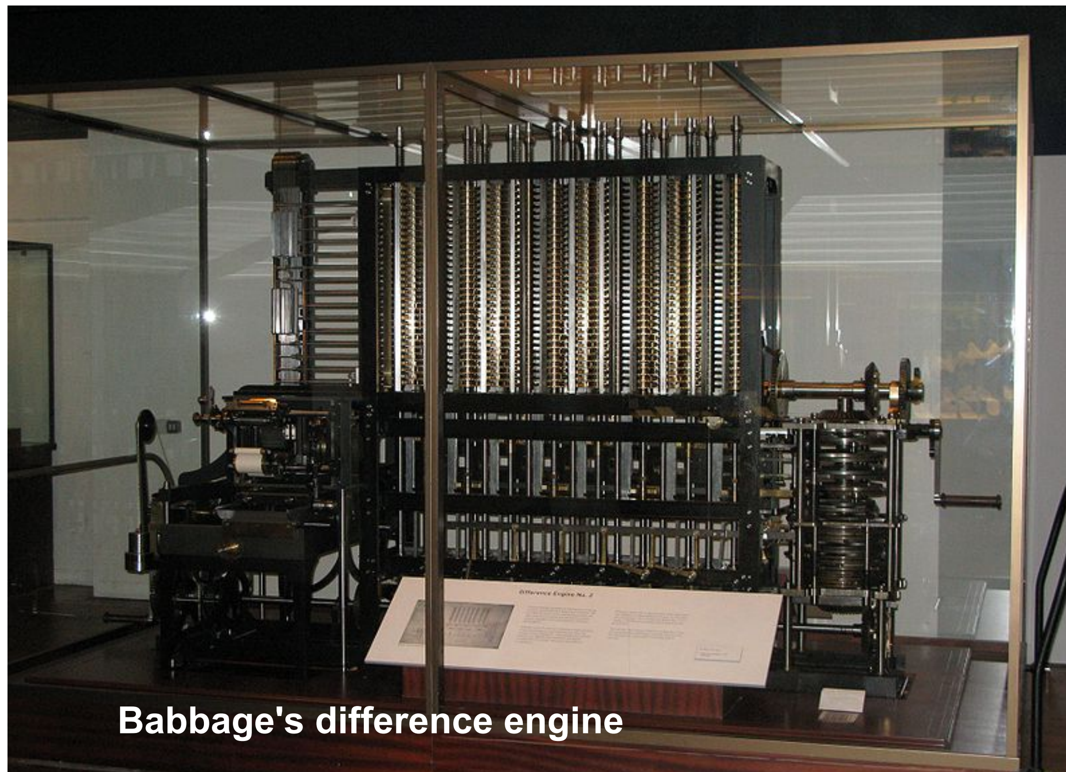
Course Overview

What is a computer?
What is Computatuional Physics?

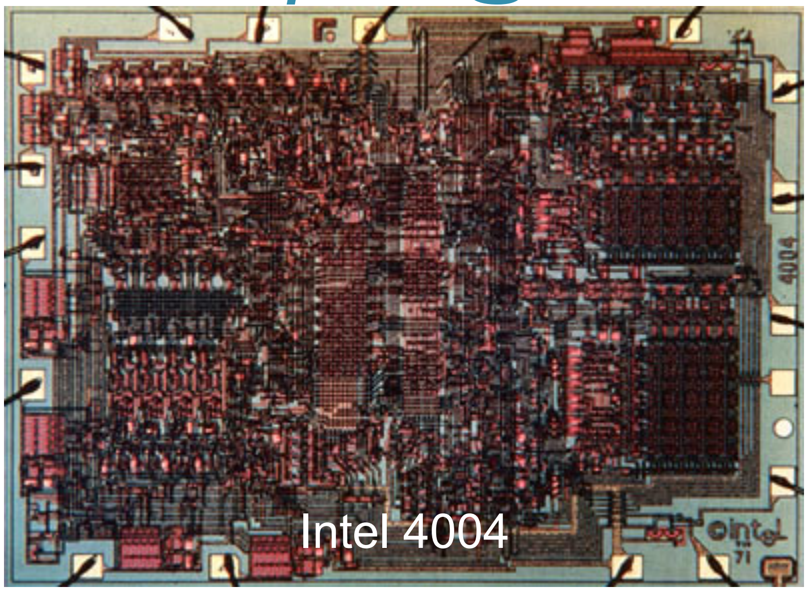
“Calculator”


Antikythera mechanism
100 BC

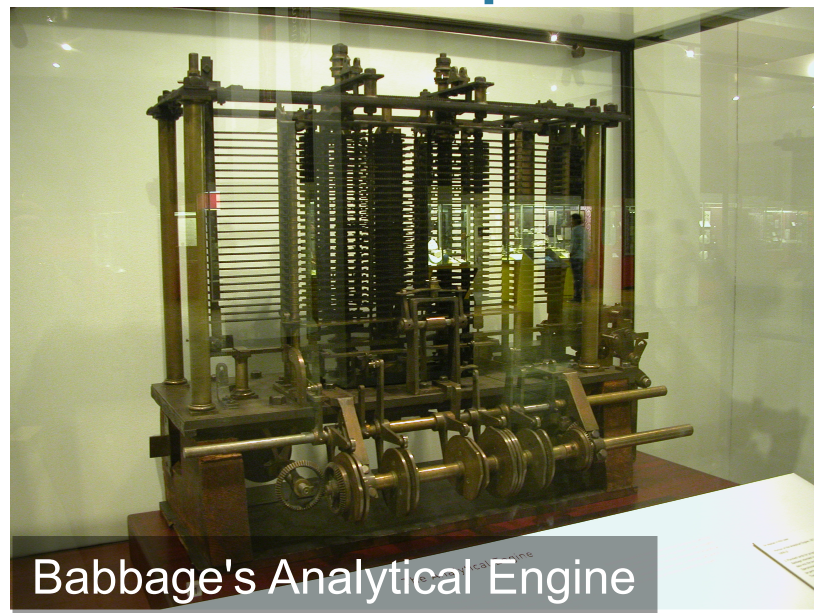

Marty McFly’s
Calculator Watch


Babbage's difference engine

A *programmable* computer

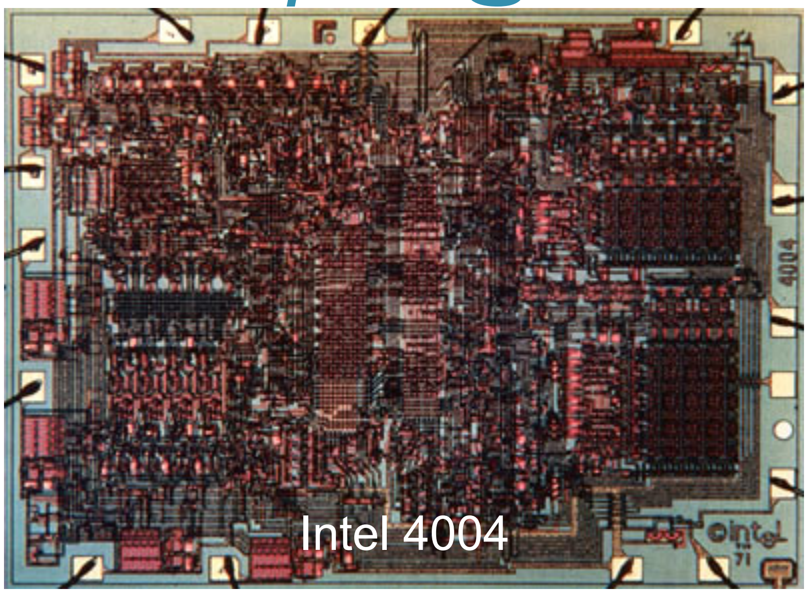


Intel 4004

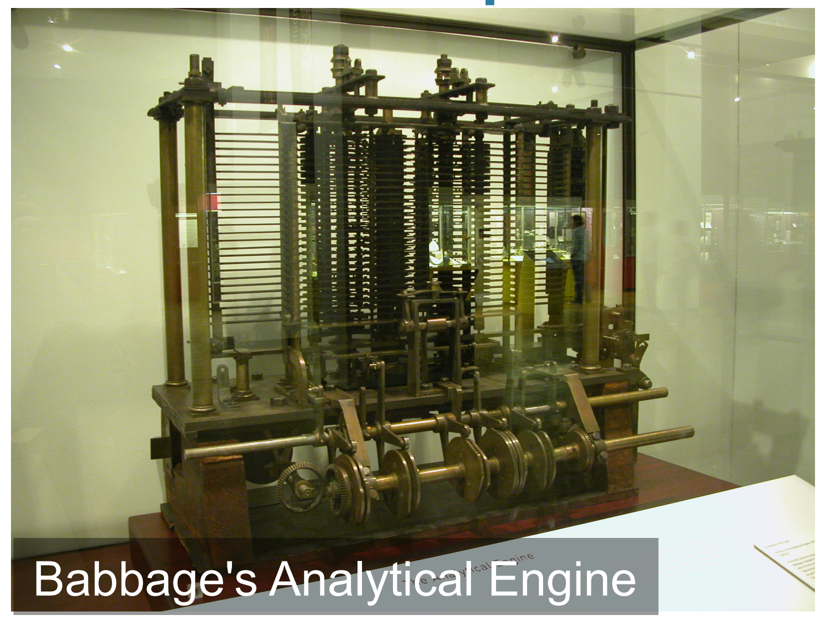


Babbage's Analytical Engine

A *programmable* computer



Intel 4004



Babbage's Analytical Engine

45 Years of the Microprocessor: Intel 4004 was 1971

4004 > 8008 > 8080 > 8086 > 80186 > 80286 > 80386 > 80486 > Pentium > PII > PIII > P4 > Core 2 > Core i3/5/7 > Xeon i5

Speed (Hz): 140,000 -> 4,600,000,000
Transistors 2,300 -> 5,500,000,000
Transistors * speed increased by 78,000,000,000

The Computer

- Lots of maths
- Controlled by logic
    - If something do this
    - Otherwise do that
- The combination of a mathematical calculator with logic based *flow control* is what makes a programmable computer
- Further reading
    - Universal Turing Machine
    - Von-Neumann architecture

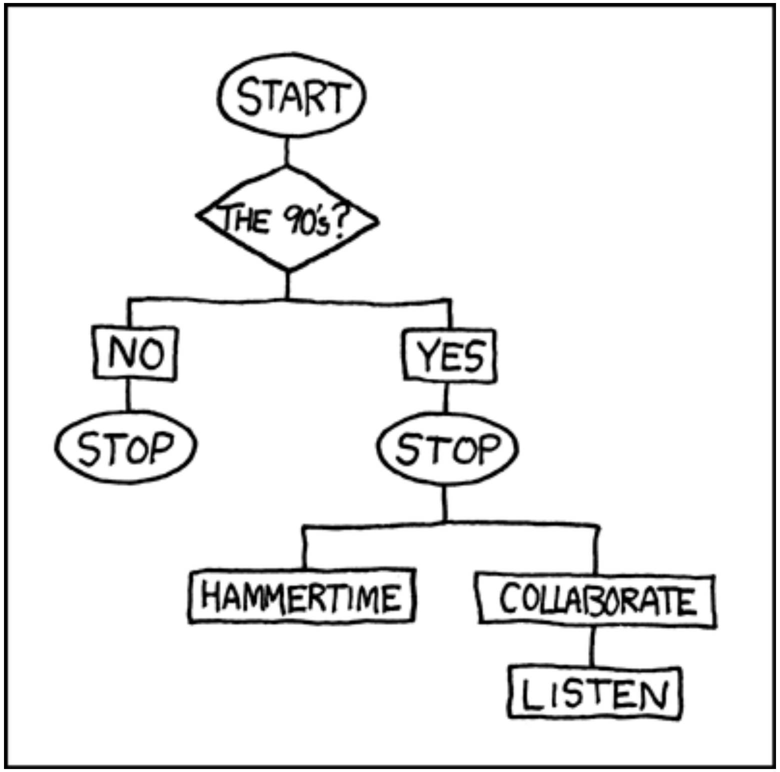


Image credit: XKCD https://xkcd.com/210/

What is Computational Physics?

Computational Physics is

Using **numerical methods**

With a computer

To solve physics problems

Everything we do in this course could be done with pen and paper, just more slowly.

Computational Physics is **not**

- Computer science
  - Mathematical basis behind computation
    - E.g. “Does this program ever finish”
  - Algorithm design
    - “Find the most efficient way of sorting these names”
  - Data structures
    - “How to store, index and retrieve patient records efficiently”

- Programming
  - Just a tool, like your pocket calculator or Excel
  - We provide support throughout the course to help with this

What is Computational Physics?

- Any arbitrary system can be described in terms of equations
  - Ground state of a hydrogen atom
  - Orbital mechanics
  - Weather
- Evaluating these equations allows us to simulate the system
  - It is through modelling / simulating many systems that we learn
  - Simulate a theory and compare to experiment – is the theory correct?
- For anything but the simplest system, an *analytical solution* is not possible
  - 2 body vs 3 body problem in gravitation
  - Ground state of hydrogen vs helium
- Instead you have to solve the equations *numerically*

Programming...

- This course is about **numerical methods** applied to Physics
- You will **program** a computer to do this
- The combination of **numerical methods** and **programming** is a key skill for many researchers in Physics

Programming...

- Hands Up time!
  1. Who has a qualification in Mathematics?
  2. Who has a qualification in Physics?
  3. Who has a qualification in Computer Programming?

- Programming is a “great divide” amongst you – massive variation in background compared to other subjects

Programming...

- With this in mind, I put a lot of effort into helping with the programming, such as:
  - Relevant examples in the lectures
    - Type them in! Learn by doing and experimenting
  - Significant skeleton code in the weekly assessments
    - Read it, think about it, type it in
  - Model Solutions
    - Read them, compare them to yours

Getting More Help

- If you are struggling with the programming:
  - Speak up! Ask at the end of the lectures. If you have a question, others almost certainly do as well
  - I will hold informal Q&As for 5 minutes at the end of lectures. Come down to the front and talk to me (we may have to move)
  - Talk to me and/or the other demonstrators in the workshops
  - Email me! If its easier to talk than write the email to explain the issue then email me to arrange a separate meeting
    - I am happy to receive groups of people

Understanding

- If you're a crack expert at programming
  - Please remember that many people are not
  - This sets the pace of the course
  - Think about how to expand on the weekly assessments, or ask me for suggestions
  - Remember: The course is about **numerical methods**
  - Make use of the workshop sessions – come along and ask the staff member to comment on your work – there is always something to learn (and we might spot a missing axis label!)

Course Information

Learning Outcomes

- An understanding of numerical methods
  - Numerical methods for
    - Differentiation,
    - Integration
    - 1st and 2nd order ODEs
  - Monte Carlo techniques, random walks
  - Function minimization and optimization
  - Fractals and Chaos

- Developing skills
  - Familiarity with programming
    - Implementing things yourself
    - Using “off the shelf” code from *scipy*
  - Graphically presenting data

Weekly Assessments

- Weekly assessments are issued for this course
  - Each problem takes the form of a small, simple Python program
  - No more than a page of code
- Problems are released on DUO
- Submission is electronic through DUO
- Your code will be printed out, marked and returned through the normal weekly problem system
- Read the style guide (on DUO)

Course Structure

- 1300 Friday Assessment released (duo)
- 1700 Monday, Tuesday, Thursday, Friday - Workshop
  - You attend one workshop/week
  - The workshop session is to provide you with help and support for the associated weekly assessment
  - Start the problem before the workshop to benefit the most
  - READ THE SHEET ALL THE WAY THROUGH BEFORE START
- 1400 Monday the next week : Assessment deadline(duo)
  - Assessments are converted into hardcopy printouts which are marked and returned through pigeon holes

Deadlines

- 1400 on Monday is a **HARD DEADLINE**
  - ZERO IF LATE!!!
- How to avoid missing a deadline
  - Plan to finish your problem a day or two early
  - If you haven't, submit your best efforts to date
  - Then submit your final version
- If you miss the deadline, your earlier version will be marked
- Repeat submissions via DUO are allowed and will automatically supersede your earlier submission(s)

Weekly problem marks

- The precise division of marks varies from problem to problem
- General guidance:

10%	Your file runs
40%	Correctness of results
20%	Answers to questions
10%	Quality of your graph
	AXES LABELS! UNITS! CAPTION! LEGEND!
20%	Quality of your code

Weekly problem marks

Check your work against the **"pre-flight"** check-list on DUO before submitting

L2/L3 Computational Physics 2014-2015

Pre-marking checklist

Open your file in an editor (e.g. IDLE) and look at the source code:

CHECK: Does your submission contain your name and CIS ID?
CHECK: Do you answer all questions asked in the assignment?
CHECK: Do you use meaningful variable names (e.g. "ix" and "iy" for index variables in 2 dimensions (xy), as opposed to "i" and "j")?

From the terminal, change in to the folder containing your script, type: "`python myScript.py`" where "`myScript.py`" is the name of your work, and press enter. Always check after even the most minor edit, in case you inadvertently broke something.

CHECK: Does your script run?
CHECK: Does a single graph appear?
CHECK: Do both graph axes have labels?
CHECK: Do axes labels include units where appropriate?
CHECK: Does the graph need a legend?

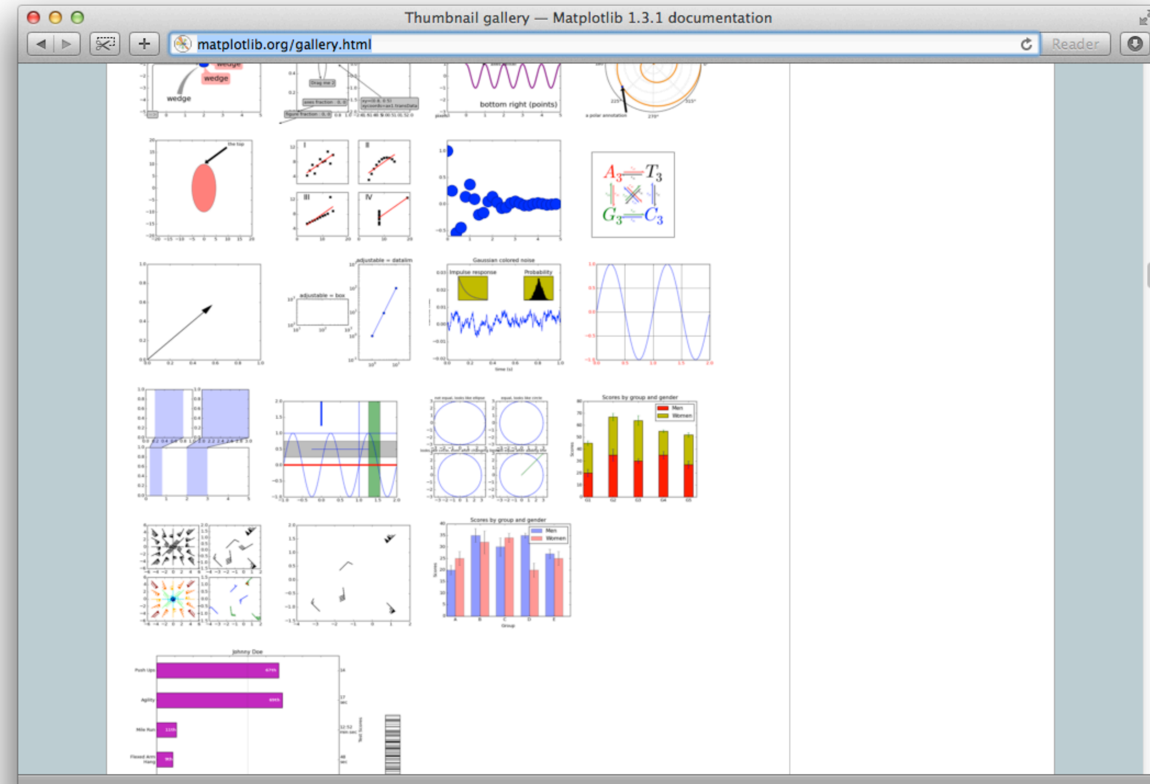
Code quality

- Follow the style guide
- Comments
  - Sparingly but meaningfully
- Variable names
  - Give them some meaning
- "Paragraphs" - Use blank lines sparingly to separate code into paragraphs. E.g.
  - Beginning – imports, set up
  - Middle – doing the maths
  - End – plotting etc.

Graphs

- Your graphs should be of suitable quality for a lab report
- Guidance is given in your assessment briefs
  - Do you want to know more?...
  - http://matplotlib.org/gallery.html
- Caption pyplot.title("...")
- Axes labels pyplot.xlabel, pyplot.ylabel
- Legend pyplot.legend("...")
- Figures in lab reports have captions, but as you do not submit a report, yours should have a title – short and descriptive

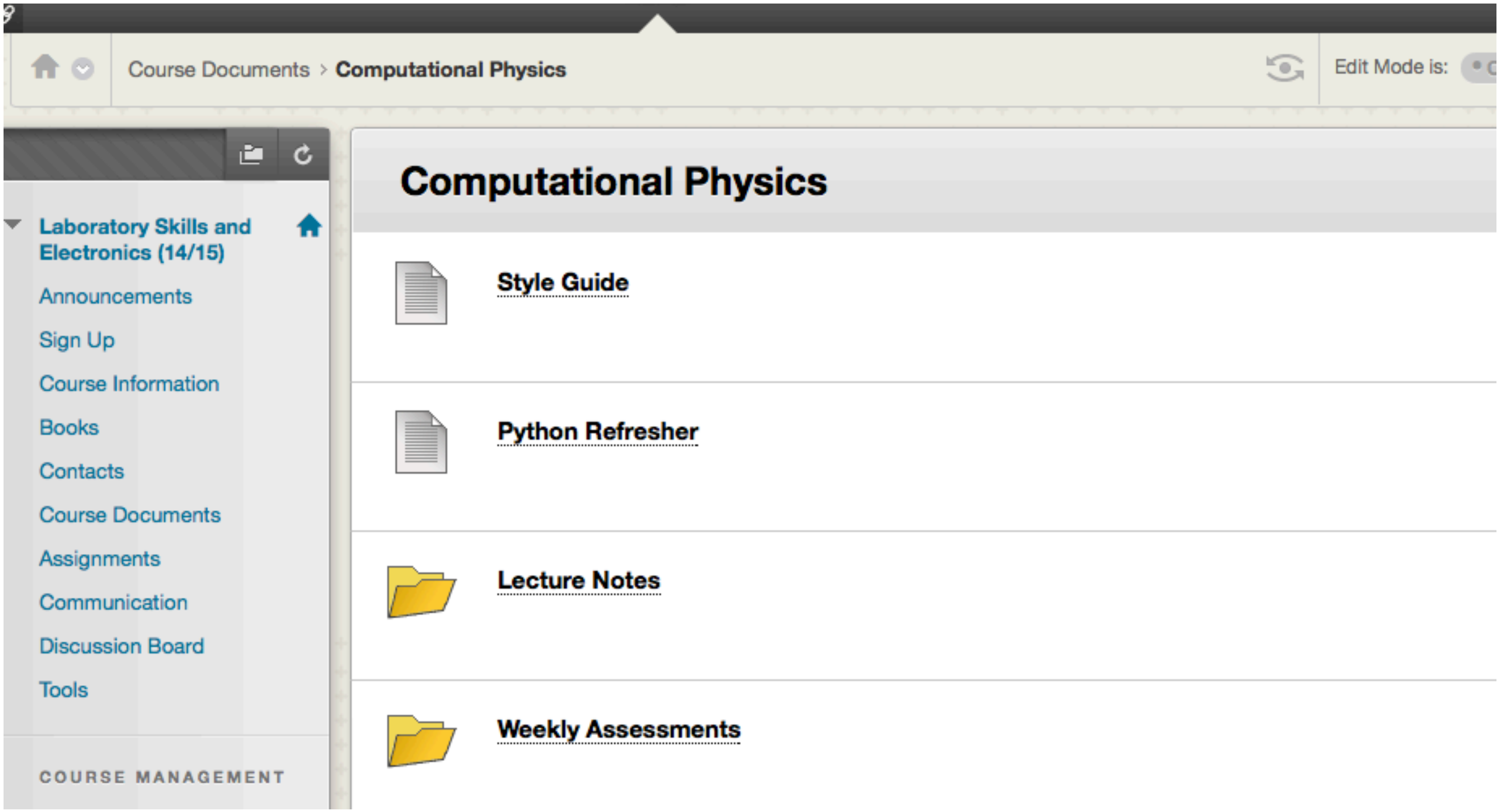
matplotlib gallery



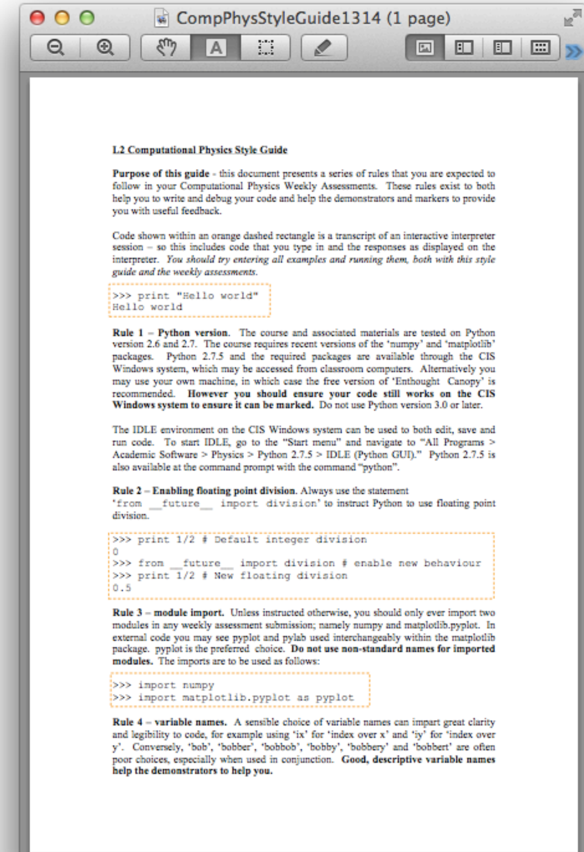
Assessments: Don't Panic

- You will be given specific guidance and example code each week READ IT AND FOLLOW STYLE
- The workshop sessions exist to give you help with both the programming and the mathematics/physics embodied by the methods
- Be prepared: Make the most of the workshop sessions – try the problem in advance and come to the workshop with questions

DUO : Laboratory Skills and Electronics (17 / 18) > Course Documents > Computational Physics



Style Guide



L2 Computational Physics Style Guide

Purpose of this guide - this document presents a series of rules that you are expected to follow in your Computational Physics Weekly Assessments. These rules exist to both help you to write and debug your code and help the demonstrators and markers to provide you with useful feedback.

Code shown within an orange dashed rectangle is a transcript of an interactive interpreter session – so this includes code that you type in and the responses as displayed on the interpreter. *You should try entering all examples and running them, both with this style guide and the weekly assessments.*

```
>>> print "Hello world"
Hello world
```

Rule 1 – Python version. The course and associated materials are tested on Python version 2.6 and 2.7. The course requires recent versions of the 'numpy' and 'matplotlib' packages. Python 2.7.5 and the required packages are available through the CIS Windows system, which may be accessed from classroom computers. Alternatively you may use your own machine, in which case the free version of 'Enthought Canopy' is recommended. **However you should ensure your code still works on the CIS Windows system to ensure it can be marked.** Do not use Python version 3.0 or later.

The IDLE environment on the CIS Windows system can be used to both edit, save and run code. To start IDLE, go to the "Start menu" and navigate to "All Programs > Academic Software > Physics > Python 2.7.5 > IDLE (Python GUI)." Python 2.7.5 is also available at the command prompt with the command "python".

Rule 2 – Enabling floating point division. Always use the statement '`from __future__ import division`' to instruct Python to use floating point division.

```
>>> print 1/2 # Default integer division
0
>>> from __future__ import division # enable new behaviour
>>> print 1/2 # New floating division
0.5
```

Rule 3 – module import. Unless instructed otherwise, you should only ever import two modules in any weekly assessment submission; namely numpy and matplotlib.pyplot. In external code you may see pyplot and pylab used interchangeably within the matplotlib package. pyplot is the preferred choice. **Do not use non-standard names for imported modules.** The imports are to be used as follows:

```
>>> import numpy
>>> import matplotlib.pyplot as pyplot
```

Rule 4 – variable names. A sensible choice of variable names can impart great clarity and legibility to code, for example using 'ix' for 'index over x' and 'iy' for 'index over y'. Conversely, 'bob', 'bobber', 'bobbob', 'bobby', 'bobbery' and 'bobbert' are often poor choices, especially when used in conjunction. **Good, descriptive variable names help the demonstrators to help you.**

- It's on DUO
- One page
- Please read and follow it
- It makes it easier for the demonstrators to read your code
- We have to read 180 programs each week!
- We do this so we can help you and provide feedback
- **Help us help you**

Getting the most out of lectures

- You all have your own learning styles
  - What works for one person may not work for another

- Full lecture notes go on DUO in advance of each lecture
  - Some of you may find it useful to go through these in advance
  - No need to take full notes
  - Think – will you benefit from making key point summary notes in the lecture?

- In some lectures I will describe a method on the whiteboard, incrementally building up a figure as I describe the method
  - Think – will you benefit from building up a copy of the figure on paper as I go?

Technical Background

Languages

- A programming language is how humans interact with computers
  - There are many types of language
  - There is a phenomenal variety in computer languages
  - The core concepts of most languages are very similar – but with different names and syntax

Types of language

- There are many paradigms
  - Many languages cannot be purely tagged with just one…

- Imperative/Procedural
- Functional
- Symbolic Maths
- Logic
- Many more

Imperative Languages

Imperative Languages

- "how, not what"
- Do this, then this, then this
- You tell the computer how to solve a problem

- ALGOL, COBOL, FORTRAN, C, C#, C++, BASIC, Python, Pascal, JavaScript, JAVA, MATLAB, IDL, Mathematica, Perl, ...

- This is the 'de facto' type of programming for almost all of the physical sciences and the wider software industry

- Arguably it's not the right way

Functional Languages

Functional Language

- When you program in a functional language you define
  - *Data*
  - *Mathematical functions that operate on the data*
- You never explicitly declare **how** to perform these functions
- In theory this frees up the computer to decide on the best way of actually manipulating the data
- LISP, Haskel, Microsoft Excel, Mathematica, Python
- Whilst functional languages have many benefits, in general they are rarely seen in the wild – why?
  - Perhaps this is because they are a poor fit to how many people think
  - They are not well suited to producing stuff like Windows or Word or games

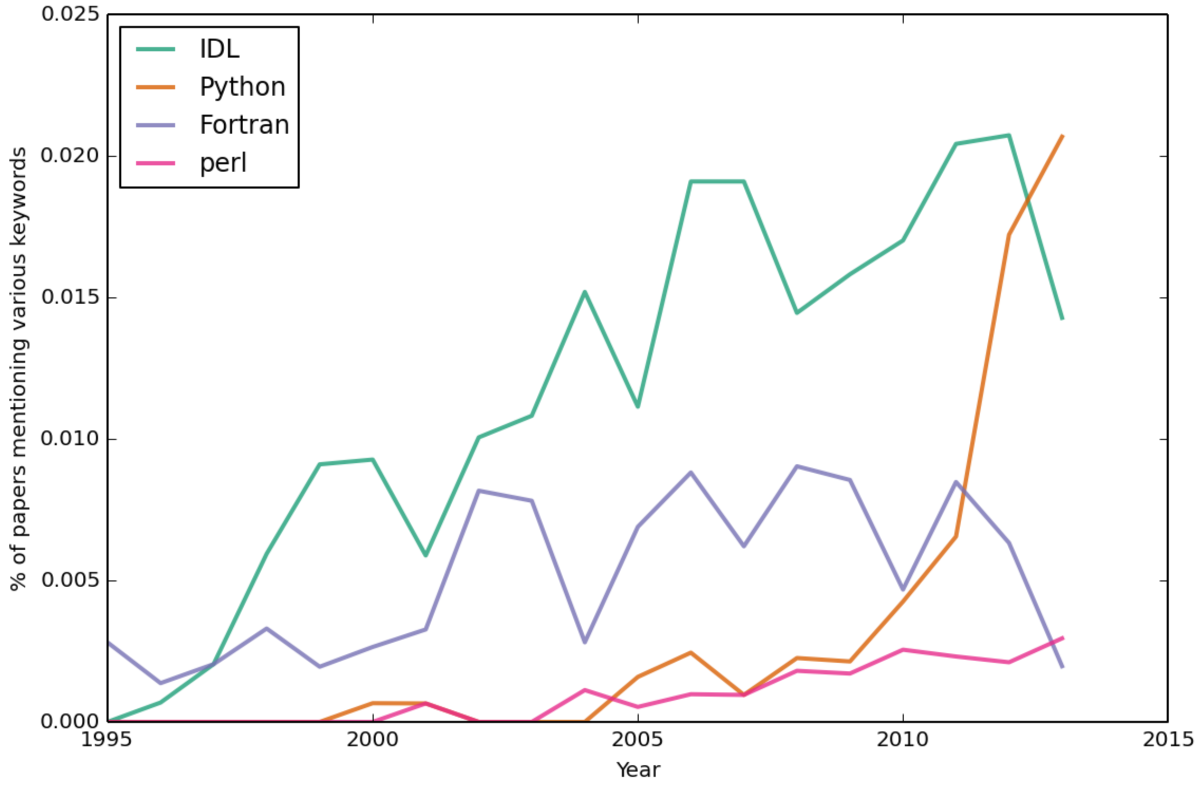
Python

About the Python programming Language

The Python language

- Origins in the early 1990s
- “Free, Open-Source”
  - No cost to buy or use it
  - The “Source Code” is freely available all
- “High level language”
  - Very approximately : it’s easier to use but potentially slower than, e.g. FORTRAN or C.
  - We will talk more about speed later
- Widely adopted by the scientific community

Languages in Astronomy



http://astrofrog.github.io/blog/2013/10/02/acknowledging-tools-services-in-papers/

Symbolic Maths

Symbolic Math

- Most languages perform numerical operations (maths) on numbers stored in *variables*
- *'Symbolic Maths'* or CAS (*Computer Algebra Systems*) perform algebra on expressions and equations defined in terms of *symbols*

Things to do with CAS

- Integration
- Differentiation
- Factorisation
- Limits
- Equation solving
- Many more

CAS Packages

- CAS is one of those areas where $,$$$ packages still sell in large quantities
- Mathematica, Maple, Mathcad, Magma
- Also plenty of open source and free packages
  - SAGE
  - SymPy (Symbolic maths in Python)
  - Plenty more

Pros

- Quick
- A good CAS will know a lot about algebra
- Analytical solutions are inherently more accurate than numerical ones

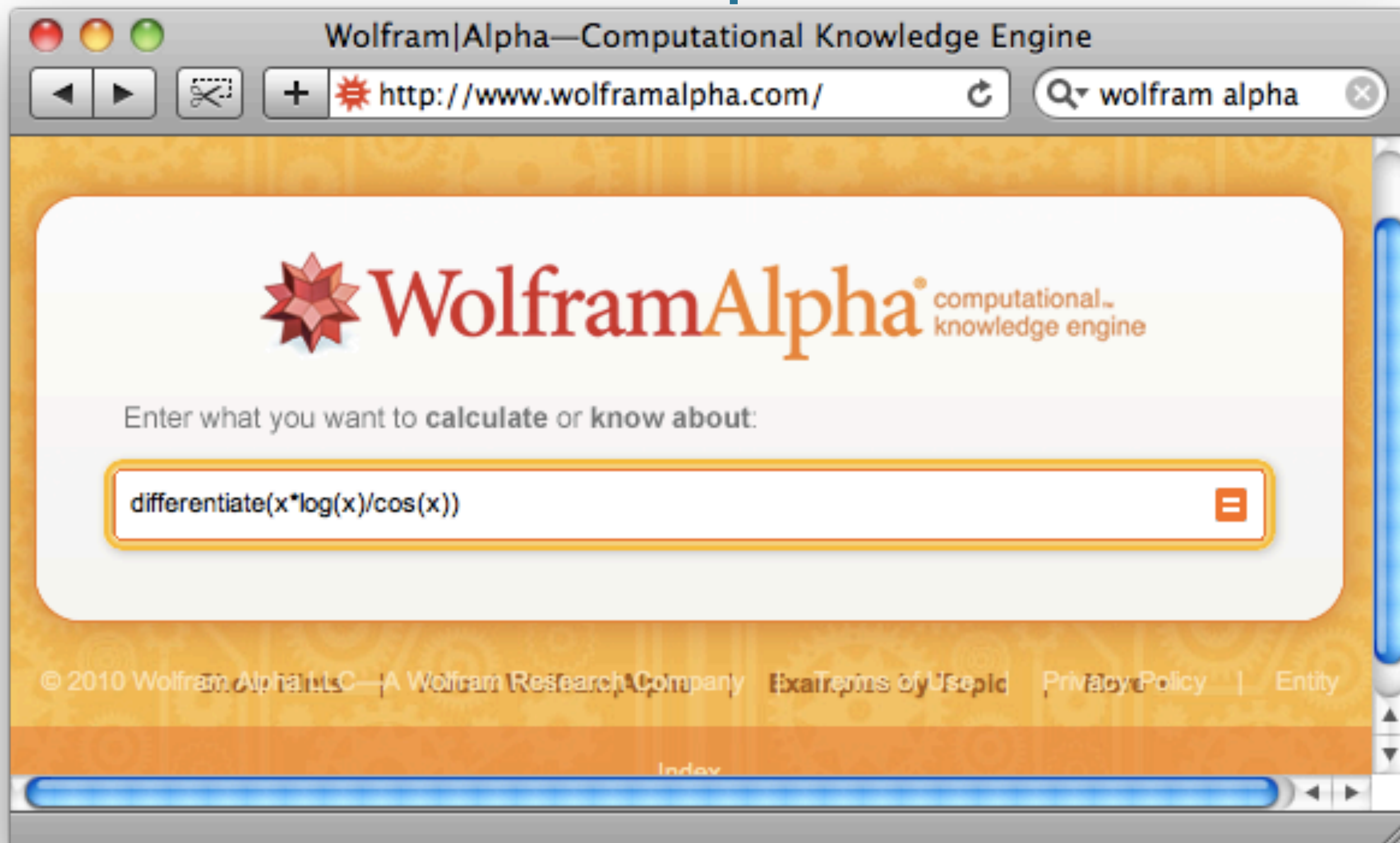
Cons

- Once you've learnt to use it...
- Brain rot!
- How did it get the answer?
- Not everything can be solved analytically and no amount of software will fix that
- Computer has **no intuition**

It's all about the Journey

- My personal philosophy:
  - The mathematical tools we learn are more than just a means to an end
  - Use of CAS hides the calculations, you just get a result
  - It is by journeying through the calculations that we come to understand the relationship between mathematics and physics, it is how we come to really understand physics
  - Learn to travel by yourself, enjoy the journey and use CAS to check that you've ended up in the right place

Example 1


Wolfram|Alpha—Computational Knowledge Engine
http://www.wolframalpha.com/
wolfram alpha
WolframAlpha computational knowledge engine
Enter what you want to calculate or know about:
differentiate(x*log(x)/cos(x))

differentiate(x*log(x)/cos(x)) – Wolfram|Alpha

http://www.wolframalpha.com/input/?i=differenti

wolfram alpha

Assuming "log" is the natural logarithm | Use the base 10 logarithm instead

Derivative:

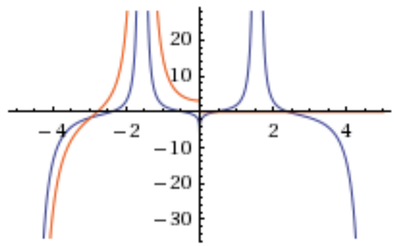
Show steps

$$\frac{d}{dx}\left(\frac{x\log(x)}{\cos(x)}\right) = \sec(x) + \log(x)\sec(x) + x\log(x)\tan(x)\sec(x)$$

$\log(x)$ is the natural logarithm »

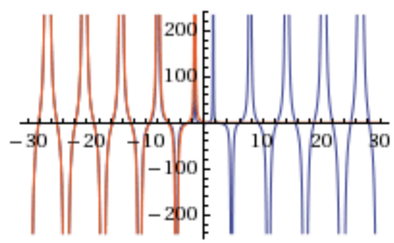
$\sec(x)$ is the secant function »

Plots:



(x from −4 to 4)

real part
imaginary part



(x from −30 to 30)

real part
imaginary part

Alternate forms:

$$\sec(x)\,(\log(x) + x\log(x)\tan(x) + 1)$$

$$\sec^2(x)\,(x\log(x)\sin(x) + (\log(x)+1)\cos(x))$$

$$\frac{2}{e^{-ix}+e^{ix}} + \frac{2\,i\,e^{-ix}\,x\log(x)}{\left(e^{-ix}+e^{ix}\right)^2} - \frac{2\,i\,e^{ix}\,x\log(x)}{\left(e^{-ix}+e^{ix}\right)^2} + \frac{2\log(x)}{e^{-ix}+e^{ix}}$$

Example 2 - sympy

- We want to compute the *indefinite integral* of

$$y = \int x^2 . \sin(x)$$

- Then we want the definite integral

Example 2 - sympy

We have to create a symbol to manipulate

```
>>> import sympy
>>> # we have to explicitly declare our symbols
>>> x = sympy.Symbol('x')
>>> # Integrate the function sin(x) * x**2
>>> sympy.integrate(sympy.sin(x) * x**2)
2*cos(x) - x**2*cos(x) + 2*x*sin(x)
>>> # Evaluate the integral over some range
>>> sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
-2 - 2*cos(2) + 4*sin(2)
>>> # sympy tries to maintain accuracy by retaining cos(2)
>>> # instead of an approximate value.  use evalf to get the value
>>> y = sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
>>> y.evalf()
2.46948338039701
```

Example 2 - sympy

Perform the integration

Note that we have to use sympy's own version of mathematical operations – it knows how to symbolically manipulate these

```
>>> import sympy
>>> # we have to explicitly declare our symbols
>>> x = sympy.Symbol('x')
>>> # Integrate the function sin(x) * x**2
>>> sympy.integrate(sympy.sin(x) * x**2)
2*cos(x) - x**2*cos(x) + 2*x*sin(x)
>>> # Evaluate the integral over some range
>>> sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
-2 - 2*cos(2) + 4*sin(2)
>>> # sympy tries to maintain accuracy by retaining cos(2)
>>> # instead of an approximate value.  use evalf to get the value
>>> y = sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
>>> y.evalf()
2.46948338039701
```

Example 2 - sympy

We can evaluate the *definite integral* between two limits

Sympy continues to work with symbols - e.g. `cos(2)`

Python Shell

File Edit Shell Debug Options Windows Help

```
>>> import sympy
>>> # we have to explicitly declare our symbols
>>> x = sympy.Symbol('x')
>>> # Integrate the function sin(x) * x**2
>>> sympy.integrate(sympy.sin(x) * x**2)
2*cos(x) - x**2*cos(x) + 2*x*sin(x)
>>> # Evaluate the integral over some range
>>> sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
-2 - 2*cos(2) + 4*sin(2)
>>> # sympy tries to maintain accuracy by retaining cos(2)
>>> # instead of an approximate value.  use evalf to get the value
>>> y = sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
>>> y.evalf()
2.46948338039701
```

Ln: 33 Col: 16

Example 2 - sympy

We ask sympy to evaluate all symbols

```
>>> import sympy
>>> # we have to explicitly declare our symbols
>>> x = sympy.Symbol('x')
>>> # Integrate the function sin(x) * x**2
>>> sympy.integrate(sympy.sin(x) * x**2)
2*cos(x) - x**2*cos(x) + 2*x*sin(x)
>>> # Evaluate the integral over some range
>>> sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
-2 - 2*cos(2) + 4*sin(2)
>>> # sympy tries to maintain accuracy by retaining cos(2)
>>> # instead of an approximate value.  use evalf to get the value
>>> y = sympy.integrate(sympy.sin(x) * x**2, (x, 0, 2))
>>> y.evalf()
2.46948338039701
```

GIGO

GIGO: Garbage In, Garbage Out

- Your model is only as accurate as the data you put in to it
  - Initial state
  - Boundary conditions
  - Physical constants
  - Assumptions
- Remember this when debugging code
  - Perhaps the problem lies with the **input data** not the **code**
- The importance of test cases!
- *Garbage In, Gospel Out*
  - Do not trust the output of a large numerical model just because the model is 1,000,000 lines of code running on a 1024 CPU supercomputer

Charles Babbage On GIGO

On two occasions I have been asked,

"Pray, Mr Babbage, if you put into the machine wrong figures, will the right answers come out?”

I am not able rightly to apprehend the kind of confusion of ideas that could provoke such a question.

Accuracy

- Try entering these numbers at the Python prompt

- 1e15 = 1,000,000,000,000,000
- 1e15 + 1 = 1,000,000,000,000,001
- 1e16 = 10,000,000,000,000,000
- 1e16 + 1 = 10,000,000,000,000,000

- Eh?

Floating Point

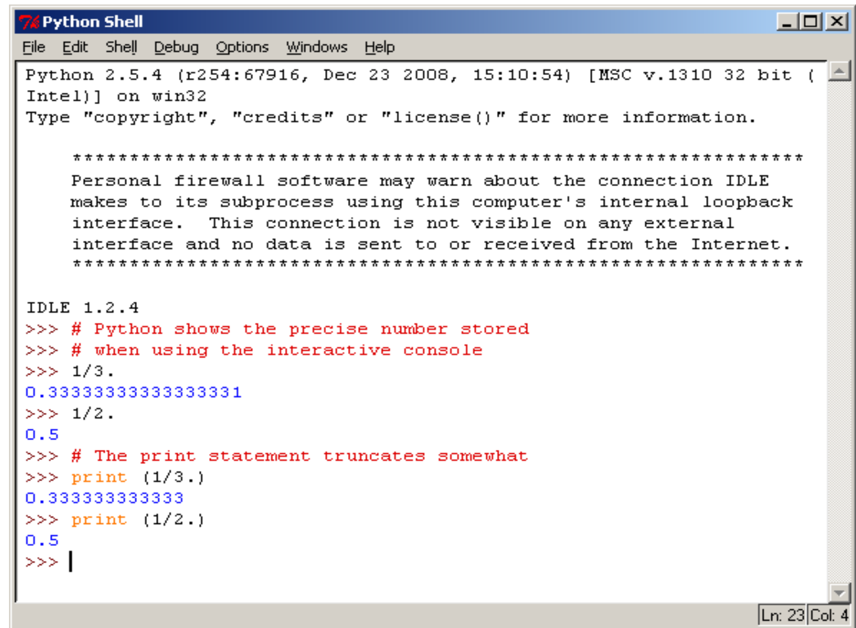
- A computer can only store a real number to a finite precision – ultimately because it has finite storage!
- The standard way of doing this is 'floating point' – a number is stored as the binary equivalent of 1.23456×10^4 for example

Floating Point

- Python normally be built using IEEE 754 standard double precision floating point
- These use 53 bits for the mantissa
  1.23456 (mantissa) x 10^4 (exponent)
- This means a maximum precision of around $1:10^{16}$ is possible
- 53 bits can store a range of 2^{53}
- $\log_{10}(2^{53})=15.95$

Fractions

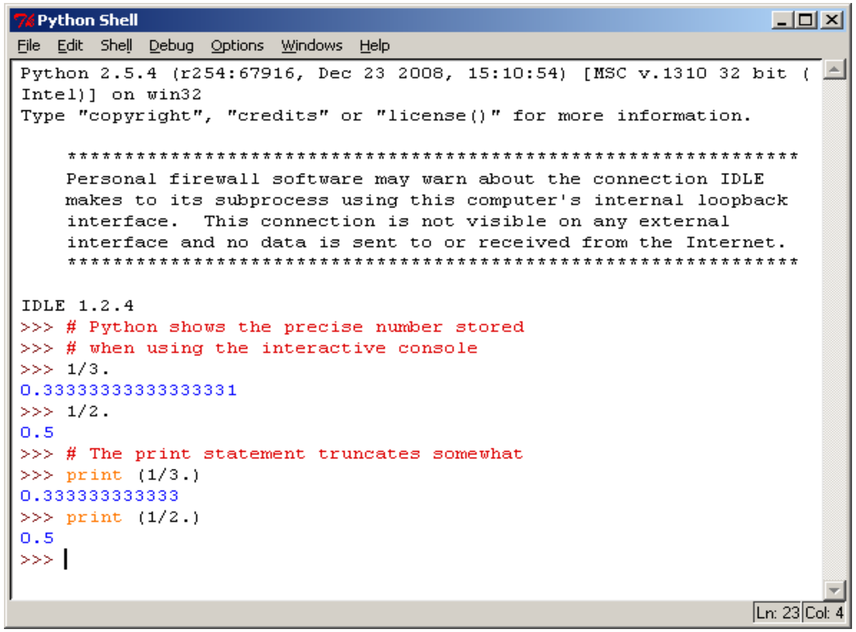
- Many fractional numbers cannot be accurately represented in binary floating point (or in decimal for that matter)



Fractions

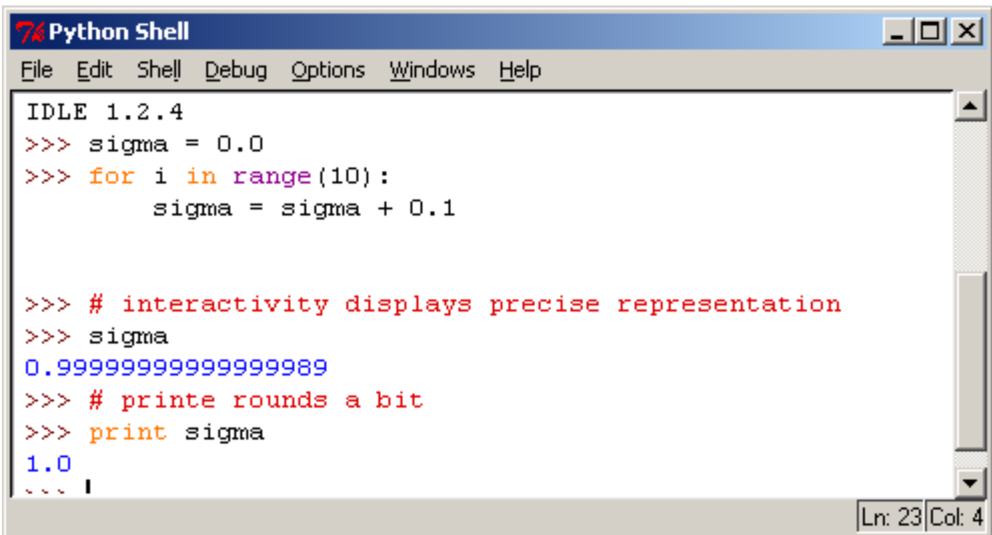
- Many languages and environments hide this by carefully displaying numbers

- Python chooses to display the details when used interactively, and to format more carefully when printing!



Rounding errors

- This also manifests in rounding errors



- This is a tiny error of $1:10^{16}$
  - Equivalent to a 4nm high bump on the Earth
  - But a very big problem if not understood

Do not test FP numbers for equality

- Often floating point numbers are **not exactly equal**
  - Due to rounding errors
  - Rounding errors depend on the **sequence of calculations**

Untitled

File Edit Format Run Options Windows Help

```
>>> a = 1.0
>>> b = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
>>> a
1.0
>>> b
0.99999999999999989
>>> a == b
False
>>> eps = 1e-10
>>> abs(a-b) < eps
True
>>>
```

Ln: 13 Col: 4

Do not test FP numbers for equality

- Can you use an index variable for your test instead?
- Or a magnitude comparison (is a > b etc.)
- Otherwise compare to a small number that is greater than rounding error

```
>>> a = 1.0
>>> b = 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
>>> a
1.0
>>> b
0.99999999999999989
>>> a == b
False
>>> eps = 1e-10
>>> abs(a-b) < eps
True
>>>
```

What’s Ahead?

1. Finite differences
2. Numerical integration – Rectangle rule, Trapezium rule, Simpson's rule
3. 1^{st} order ODEs, Euler, RK, predictor-corrector
4. 2^{nd} order ODEs, Euler-Cromer, black box solvers
5. Monte Carlo methods

What's Ahead?

6. Minimization
7. Random Walks
8. Fractals and Chaos
9. Wrap up and moving forwards

Taylor Series

Recap from Level 1

MATH156(7)1: SINGLE MATHEMATICS A(B)

MATH1061: Calculus and Probability I

Taylor Series

- Approximation for a function f(x) near a point $x=x_0$
- Expand as power series in h
- $f(x_0+h)=a_0 + a_1 h + a_2 h^2 + a_3 h^3 + \ldots$
- h=0 gives $f(x_0)=a0$
- differentiate
- $f'(x_0+h)= a_1 + 2 a_2 h + 3 a_3 h^2 + \ldots$
- h=0 gives $f'(x_0)= a_1$

$a_n=f^n(x_0)/n!$

Taylor Series

- Approximation for a function f(x) near a point $x=x_0$
- Defined in terms of the derivatives of f(x) at a

$$f(x_0+dx)=f(x_0)+\frac{f'(x_0)}{1!}dx+\frac{f''(x_0)}{2!}dx^2+\ldots$$

$$f(x_0+dx)=\sum_{i=0}^{\infty}f^{(i)}(x_0)dx^i/n!$$

- NB:

$$f'(x)=f^{(1)}(x)=\frac{df(x)}{dx}$$

$$f^{(n)}(x)=\frac{d^n f(x)}{dx^n}$$

$$0!=1$$

Taylor Series: Example

- Approximate $\sin(x_0+dx)$ at $x_0=0$

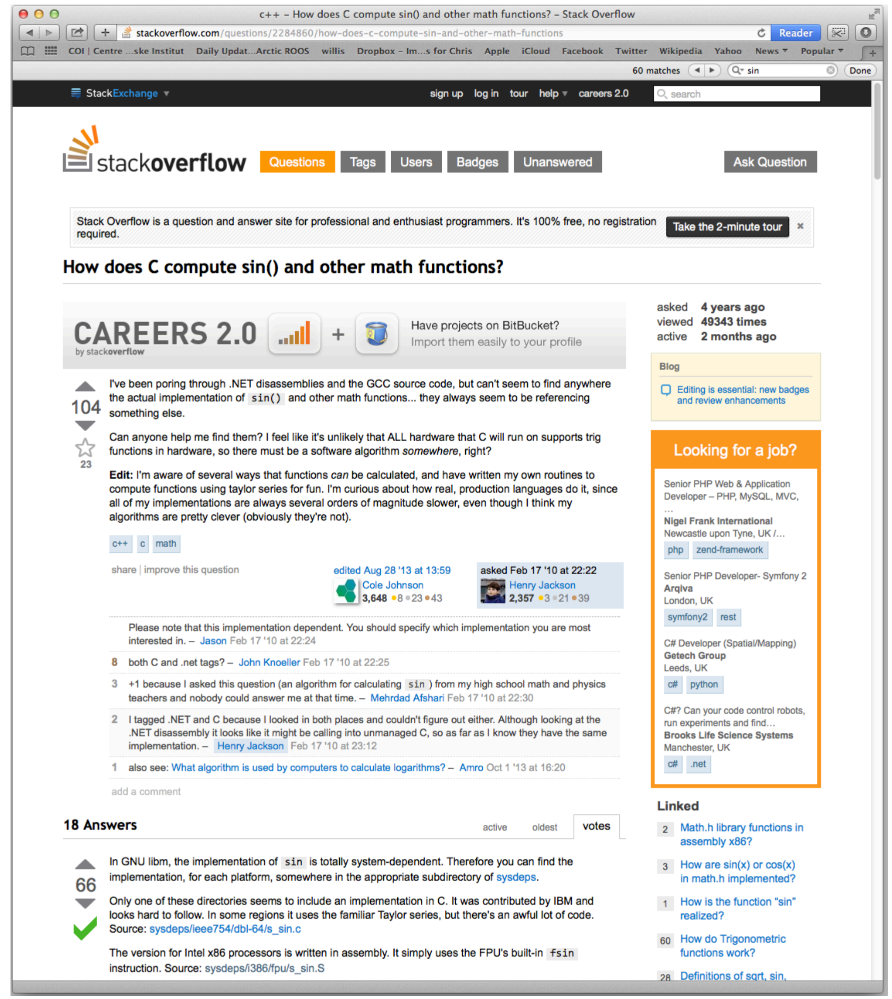
Taylor Series: Example

- Approximate $\sin(x_0+dx)$ at $x_0=0$

$$\sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Taylor Series - applications

- The Taylor series is one of the ways a computer calculates sin/cos etc.



Finite Difference Method

A numerical method to approximate the derivative of a function

Finite Difference Method

- Discarding higher order terms from Taylor series:

$$f(x+dx) \approx f(x) + f'(x)dx$$

- Re-arrange to approximate 1st derivative (gradient)

$$f'(x) \approx \frac{f(x+dx) - f(x)}{dx}$$

Classes of Method

- Forwards difference

$$f'(x) \approx \frac{f(x+dx) - f(x)}{dx}$$

- Backwards difference

$$f'(x) \approx \frac{f(x) - f(x-dx)}{dx}$$

- Central difference

$$f'(x) \approx \frac{f(x+dx/2) - f(x-dx/2)}{dx}$$

Questions to ponder...

- Taylor series
  - Does this method apply to all types of function?
- Finite difference
  - Which method(s) is most accurate?
  - Which method(s) is fastest to compute?
  - Are all methods equally useful?
  - How do you extend this to measure 2nd derivative?
  - What happens on a computer for dx << 1?