Computational Physics 1

Lecturer: Jonivar Skullerud, jonivar@thphys.nuim.ie
Room 1.7c, Science Bldg

Lectures: Thu 9–10 Comp Phys Lab, Room 1.4, Science Bldg
Lab: Thu 2–4, or Tue 2–4, Room 1.4, Science Bldg

Outline of course

- Introduction to python and programming
- Numerical differentiation (finite differences)
- Numerical integration
- Ordinary differential equations (ODEs)
- Root finding and boundary value problems

Recommended books

- Press, Teukolsky, Vetterling, Flannery: Numerical Recipes
- Garcia: Computational Physics

Python and Linux: use online tutorials and handouts
Online course material is at
`http://www.thphys.nuim.ie/CompPhysics/` or on Moodle

Assessment

- Continuous assessment (4 quizes) 20%
- Project (Mar–May) 40%
- Exam (mostly theory) 40%

Overview

Objectives and outcomes

Module objective

To acquaint students with use of computers for solving physical problems

Learning outcomes

On completing the module, you shall be able to

- recognise and use basic programming structures (variables, expressions, functions, control statements)
- analyse the truncation error of numerical differentiation and integration schemes
- perform numerical integration of ordinary differential equations
- apply these skills to solving physical problems
- present project results in a coherent report

Why computational physics?

We want to study complex systems, because the real world is complex!
For example:

- Many-body systems (eg the solar system)
- Plasma physics (many kinds of forces)
- Tectonics and seismology
- Weather and climate
- Financial markets
- Biological systems
- Subatomic physics
- and lots more!

Why computers?

- Only very few (the very simplest) systems can be completely solved analytically
- For complex systems, “small” corrections can become big and uncontrollable.
- It is just too much work to do by hand!

Computers are new, but the methods date back a long time:
Newton, Euler, Gauss, Jacobi,. . .

Model vs theory

- A model is designed to describe phenomena
- A theory is designed to explain phenomena
- A good model may be very complicated, have lots of adjustable parameters
- A good theory is supposed to be simple

Example

- Viscous hydrodynamics is a theory
- ECMWF and HIRLAM (used by Met Éireann) are models
- Advanced models can incorporate both model and theory elements
  - ECMWF and HIRLAM incorporate fluid mechanics and thermodynamics theory and processes that cannot be easily described by theory (condensation, soil processes)

Computational physics works with all of these!

Mathematical models

Mathematical models and theories can take a number of forms:

- Multivariable functions $\longrightarrow$ minimisation, finding roots
- Ordinary differential equations
- Partial differential equations
- Matrix equations or eigenvalue problems
- Stochastic differential equations
- Multidimensional integrals

We will introduce methods to handle all (or most) of these
We will also look at methods for analysing real or simulated data

Modelling

We usually start with a simple model, describing what we think are the main features.

Example

Ball flying through the air
The main force is gravity: $m\vec{a} = m\frac{d^2\vec{x}}{dt^2} = -mg\vec{z}$
The solution to this is known: a parabola
Now we add 'complicating factors' in controlled fashion if possible:

- air drag
- spin of ball
- uneven geometry of ball
- variable atmospheric conditions
- variable gravitational force
- coriolis force
- etc.

Modelling

Do the results make sense?

Check vs empirical data if possible

- We cannot always experiment
  (cosmology, climate, stock market, nuclear reactors)
- It is not always safe or easy to measure the real system

Check if the results look sensible! (use your physical intuition)
If results are wrong or nonsensical, try to improve the model!

Predictions

Modelling can predict what happens if . . .

- ball: thrown at different angles, speeds, spin, condition of ball. . .
- climate: reduced/increased emissions

A good model has predictive power and deepens our understanding

Error analysis

No numerical study is complete without error analysis
What is the margin of error (uncertainty) of our numbers?

Types of errors

- statistical (spread in experimental or simulation data)
  - can be reduced by making more 'measurements'
- systematic (everything else!)
  - numerical: rounding errors (precision), algorithm errors
  - calibration errors or scale setting
    computers can only calculate dimensionless quantities
  - model uncertainties: factors not taken into account

We will look at the different types of numerical errors later on.
We will study them for each algorithm/method we encounter

Other numerical issues

Speed (rate of convergence)

- it is no use knowing you will get the right answer if it will take forever to get there!

Memory

- computers cannot store an infinite amount of data!

Stability

- some algorithms may "run away into outer space"
- related to error analysis

These will also be studied (where relevant) for the various algorithms we encounter.

Numerical computing — the basics

Computers are stupid!

- They can only do exactly what you have told them to do
- Basically, they only know about 0 and 1
- They are finite!

Numbers

- Computers may know about integers
  - can be constructed from zeros and ones
- They cannot know about real (irrational) numbers
  - they cannot be represented by a finite string of zeros and ones
  - can only be represented to some finite precision

Two very different types of numbers

- integers (exact) `-IMAX ... IMAX`
- floating point numbers (inexact)
  - usually represented as $\mathtt{r} = \mathtt{s} \cdot \mathtt{M} \cdot 2^{E-\varepsilon}$
    `s` = sign, `M` = mantissa, `E` = exponent
- machine precision: the smallest number that can be added to 1 to give a number different from 1
  - not the same as the smallest number that can be represented!

Limits

Since integers are exact and floats are not, computer languages and numerical software treat them separately

Precision and range

These are given by how many bits (binary digits) are used.
Typical are 32 (single) or 64 (double)

Type	Bits	Min	Max	Precision
Integer	32	-4,294,967,295	4,294,976,296	
	64	92,233,703,854,775,807		
Float	32	$\sim 10^{-38}$	$\sim 10^{38}$	$\sim 10^{-7}$
	64	$\sim 10^{-308}$	$\sim 10^{308}$	$\sim 2 \cdot 10^{-16}$

Numbers in Python

If you give a variable an integer value, Python will decide it is an integer, and use integer arithmetic on it. If you want it to behave as a floating-point number, put a decimal point on it. All floats in Python are double precision.

```
>>> clear
>>> a=13; b=3
>>> type(a)
<type 'int'>
>>> a/b
4
>>> c=3.0
>>> type(c)
<type 'float'>
>>> a/c
4.333333333333333
```

Rounding errors

- There is no point trying to get answers to greater precision than machine precision
- Critical when subtracting large numbers!

Example

Solution of $ax^2 + bx + c = 0$,

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

can be useless if $b^2 \gg ac$

- We often use increased precision when summing up lots of numbers

Other data types

- complex numbers
  - constructed as pairs of floating point numbers, eg `z = complex(0.5,1.2)` or `z = 2.0-0.7j`
- characters `'a', '*', ' '`
  - usually just a number between 0 and 255 (8 bits = 1 byte)
- strings of characters `'this is a string!'`
  - arrays (lists) of characters
  - Python has lots of functions to operate on strings
  - you can use `'single quotes'` or `"double quotes"`
- logical (boolean): false/true $\equiv$ 0/1

Functions, vectors and discretisation

In physics applications we are often interested in functions of one or more variables, for example

- the position of an object as a function of time
- the value of the electric field as a function of x and y
- the temperature, pressure and wind speed in different places
- the wave function of a quantum mechanical particle

Functions vs numbers

To a computer, a function is usually just a recipe:

$$\text{numbers in} \longrightarrow \text{function} \longrightarrow \text{numbers out}$$

In maths/physics, a function has many other properties:

- continuity (or discontinuity)
- singularities
- derivatives, integrals
- series expansions, fourier expansions, etc

The computer (usually) knows nothing about all this!

Discretisation

Computers know nothing about the continuum

— we must tell it to compute values at discrete positions or times x_i, t_i

$$\vec{r}(t) \to \vec{r}(t_i) \qquad \to (x_i, y_i, z_i)$$

$$\vec{E}(x, y) \to \vec{E}(x_i, y_j) \qquad \to (E_x^{ij}, E_y^{ij}, E_z^{ij})$$

The numbers x_i, y_i, z_i, t_i is all we know about $\vec{r}(t)$

In particular, the computer does not know that the numbers x_i, y_i, z_i have anything to do with t_i, or with each other.
We usually take the argument values t_i to be regularly spaced:

$$t_i = t_0 + i\Delta t$$

All our knowledge of a function $f(x)$ consists of

- a recipe for computing $y = f(x)$ for a given x
- rows of numbers x_i and their associated function values y_i

Summary

- Computational physics is essential for studying complex physical phenomena = the real world
- We often use models:
  - Not the full truth!
  - Start with a simple model, adjust as you go along
- Error analysis is essential in all numerical studies
- Two types of numbers:
  - integers (exact)
  - floats (finite precision)
- Rounding errors can occur in floating point arithmetic, especially when subtracting large numbers
- Computers know about numbers, not functions
  - A function is just a recipe to get from one number to another
- Computers do not know about the continuum
  - To describe a function we must compute its value at discrete points
  - Our knowledge of the function consists of rows of numbers