Overview

Recap

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

- Two types of numbers:
  - integers (exact)
  - floats (finite precision)
- Rounding errors can occur in floating point arithmetic, especially when subtracting large numbers

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

Python distinguishes between integers and floating-point numbers, and will treat the two differently.

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

All floats in Python are double precision, unless you specifically say otherwise!

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

Numerical differentiation

Say we want to determine some function $f(x)$, its derivatives and integral
For example:

- height of a ball (or ICBM) above ground, $y(x)$
- distance from sun as function of angle, $r(\theta)$
- distance from sun as function of time, $r(t)$

Recall: Computers know nothing about the continuum!
The computer can only compute $f(x)$ at some discrete x_i, eg

$$x_i = x_0 + i\delta \qquad \text{[Python: } i = 0 \ldots N-1\text{]}$$
$$\rightarrow \qquad f_i = f(x_i) = f(x_0 + i\delta)$$

The vector f_i (together with x_i) is all we know about $f(x)$

How do we find $f'(x) = df/dx$?

Discrete derivatives

Recall the definition of the derivative:

$$\frac{df}{dx} \equiv \lim_{\delta \to 0} \frac{f(x+\delta) - f(x)}{\delta}$$

Our δ is fixed — this suggests using

$$\frac{df}{dx} \approx \frac{\Delta^F f}{\Delta x} \equiv \Delta_x^F f(x) \equiv \frac{f(x+\delta) - f(x)}{\delta}$$

This is the right or forward derivative. But there are others:

The left or backward derivative

$$\frac{df}{dx} \approx \Delta_x^B f(x) \equiv \frac{f(x) - f(x-\delta)}{\delta}$$

The centred or symmetric derivative

$$\frac{df}{dx} \approx \Delta_x^S f(x) \equiv \frac{f(x+\delta) - f(x-\delta)}{2\delta}$$

Accuracy analysis

Theorem (Taylor's theorem)

For all positive integer n there exists a $\theta \in [0,1]$ such that

$$f(x+\delta) = f(x)+\delta f'(x)+\frac{\delta^2}{2}f''(x)+\ldots+\frac{\delta^n}{n!}f^{(n)}(x)+\frac{\delta^{n+1}}{(n+1)!}f^{(n+1)}(x+\theta\delta)$$

We do not know what θ is, but we do know that

$$\frac{1}{(n+1)!}f^{(n+1)}(x+\theta\delta) \quad \text{is some finite number}$$

We write

The error term

$$\frac{\delta^{n+1}}{(n+1)!}f^{(n+1)}(x+\theta\delta) = \mathcal{O}(\delta^{n+1})$$

Error on discrete derivatives

Use Taylor expansion:

$$f(x \pm \delta) = f(x) \pm \delta f'(x) + \frac{\delta^2}{2} f''(x) + \mathcal{O}(\delta^3)$$

$$\begin{aligned}\Delta_x^F f(x) &= \frac{1}{\delta}\left[f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x) + \mathcal{O}(\delta^3) - f(x)\right] \\ &= f'(x) + \frac{\delta}{2} f''(x) + \mathcal{O}(\delta^2) = f'(x) + \mathcal{O}(\delta)\end{aligned}$$

Similarly,

$$\begin{aligned}\Delta_x^B f(x) &= \frac{1}{\delta}\left[f(x) - \left(f(x) - \delta f'(x) + \frac{\delta^2}{2} f''(x) + \mathcal{O}(\delta^3)\right)\right] \\ &= f'(x) - \frac{\delta}{2} f''(x) + \mathcal{O}(\delta^2) = f'(x) + \mathcal{O}(\delta)\end{aligned}$$

$$\begin{aligned}\Delta_x^S f(x) &= \frac{1}{2\delta}\left[f(x) + \delta f'(x) + \frac{\delta^2}{2} f''(x) - f(x) + \delta f'(x) - \frac{\delta^2}{2} f''(x) + \mathcal{O}(\delta^3)\right] \\ &= f'(x) + \mathcal{O}(\delta^2)\end{aligned}$$

Error on discrete derivatives

Forward, backward and symmetric

$$\Delta_x^F f(x) = f'(x) + \mathcal{O}(\delta)$$
$$\Delta_x^B f(x) = f'(x) + \mathcal{O}(\delta)$$
$$\Delta_x^S f(x) = f'(x) + \mathcal{O}(\delta^2)$$

The forward and backward derivatives have errors of $\mathcal{O}(\delta)$, while the symmetric derivative has errors of $\mathcal{O}(\delta^2)$.

Notation

We usually denote the accuracy by the order of the leading error term:

- Δ^F, Δ^B are accurate up to [errors of] $\mathcal{O}(\delta)$
- Δ^S is accurate up to $\mathcal{O}(\delta^2)$

Why use Δ^F, Δ^B?

Why would we ever want to use Δ^F, Δ^B?

- We have an irregular grid: $x_i - x_{i-1} \neq x_{i+1} - x_i$
- $f(x)$ is very expensive to evaluate; we have already evaluated it at x
- We cannot evaluate $f(x - \delta)$ (out of physical range)
- The symmetric derivative gives rise to unphysical artefacts, eg. spurious symmetries

Summary

- 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
- Discrete derivatives
  - Forward derivative: $\Delta^F f(x) = [f(x+\delta) - f(x)]/\delta = f'(x) + \mathcal{O}(\delta)$
  - Backward derivative: $\Delta^B f(x) = [f(x) - f(x-\delta)]/\delta = f'(x) + \mathcal{O}(\delta)$
  - Symmetric derivative:
    $\Delta^S f(x) = [f(x+\delta) - f(x-\delta)]/(2\delta) = f'(x) + \mathcal{O}(\delta^2)$
  - Accuracy analysis performed by Taylor series expansion