Review: equation solving

Bisection:

Begin with interval bracketing zero

Choose half sub-interval which contains zero

Works always, but converges slowly: error is halved in each step

Ridder's method (secant method is similar):

Starts with two initial guesses bracketing the root.

Uses linear interpolation on a specially constructed function to derive next, better approximation to the root

Fairly fast (super-linear conv.) and does not require derivative

Newton-Raphson method:

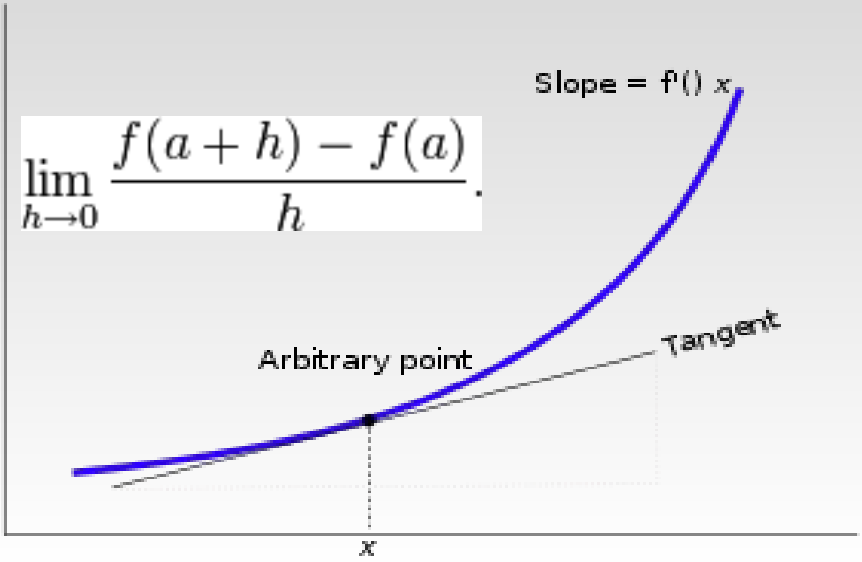
Starts at a point x (which has to be chosen carefully!)

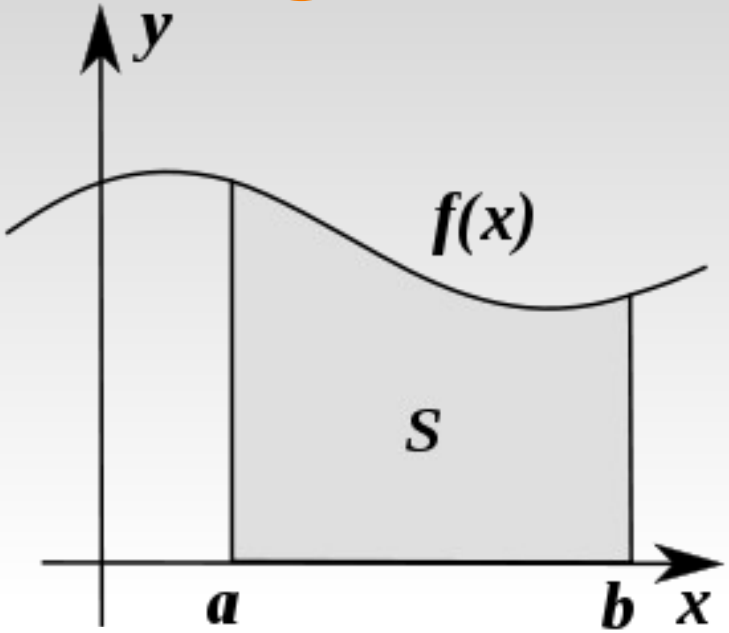
Applies linear extrapolation to zero using the first derivative at x

Very fast (quadratic convergence), but it can (and does) fail

The only method here useful for systems of algebraic equations

Differentiation and Integration





Basic operations of calculus.

Inverse to each other, since:

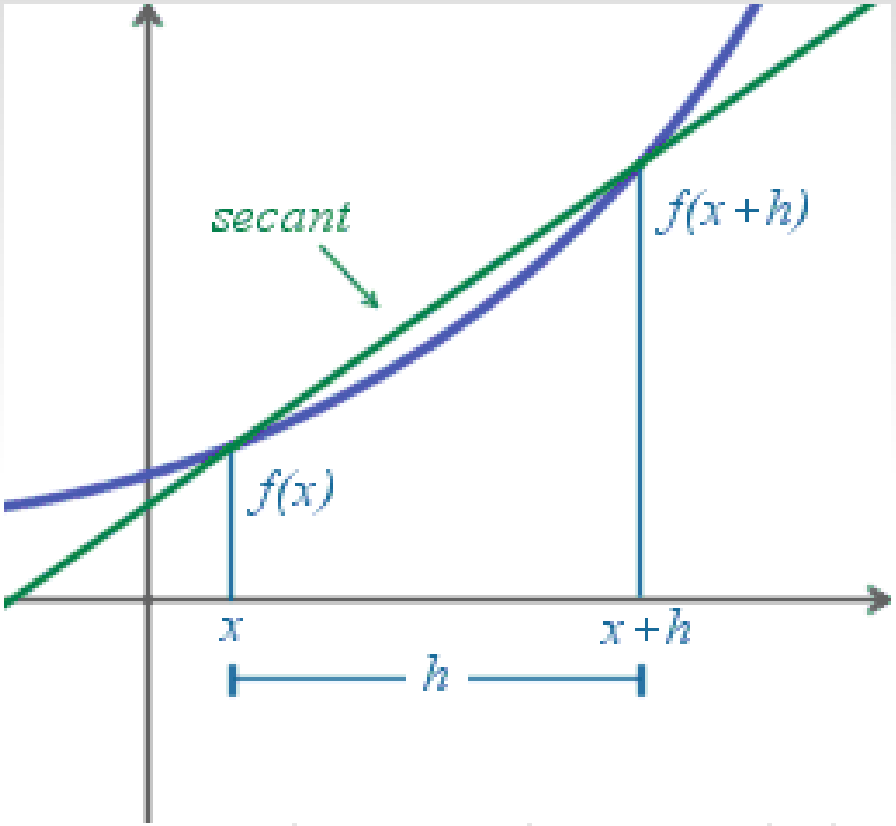
$$\frac{d}{dx}\int_a^x f(t)\,dt = f(x).$$

Both have simple geometric meaning: slope of tangent (derivative), area under curve (integral) → suggest ways to calculate integrals and derivatives numerically.

Analytically calculating derivatives is straightforward (clear rules), but integrals are difficult, typically impossible to express in a closed analytical form. Numerically the opposite is true, we will see why.

Numerical differentiation

- Problem: given a function f(x), estimate numerically its first (or higher) derivative(s).
- Function could be given at a finite number of points, or as a rule to compute at any point.
- Numerically difficult and inaccurate to compute (division by small numbers, round-off errors)



$$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}.$$

Finite-difference approximations

- Start with Taylor series expansions:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + \cdots$$

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) - \cdots$$

$$f(x+2h) = f(x) + 2hf'(x) + \frac{(2h)^2}{2!}f''(x) + \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f^{(4)}(x) + \cdots$$

$$f(x-2h) = f(x) - 2hf'(x) + \frac{(2h)^2}{2!}f''(x) - \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f^{(4)}(x) - \cdots$$

Finite-difference approximations

- Add and subtract the series, obtaining:

$$f(x+h)+f(x-h)=2f(x)+h^2 f''(x)+\frac{h^4}{12}f^{(4)}(x)+\cdots$$

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

Errors

$$f(x+2h)+f(x-2h)=2f(x)+4h^2 f''(x)+\frac{4h^4}{3}f^{(4)}(x)+\cdots$$

$$f(x+2h)-f(x-2h)=4hf'(x)+\frac{8h^3}{3}f'''(x)+\cdots$$

- Dropping high-order terms, we find, e.g.:

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

Centred FD

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

Forward FD

Applications of finite-differencing

In fact, we have already used finite differencing, for example in the secant method for solving equations:

$$x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

which is just a finite-differenced version of the Newton method for solving equations:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
## module newtonRaphson2
''' soln = newtonRaphson2(f,x,tol=1.0e-9).
    Solves the simultaneous equations f(x) = 0 by
    the Newton-Raphson method using {x} as the initial
    guess. Note that {f} and {x} are vectors.
'''
from numpy import zeros,dot
from gaussPivot import *
from math import sqrt

def newtonRaphson2(f,x,tol=1.0e-9):

    def jacobian(f,x):
        h = 1.0e-4
        n = len(x)
        jac = zeros((n,n))
        f0 = f(x)
        for i in range(n):
            temp = x[i]
            x[i] = temp + h
            f1 = f(x)
            x[i] = temp
            jac[:,i] = (f1 - f0)/h
        return jac,f0

    for i in range(30):
        jac,f0 = jacobian(f,x)
        if sqrt(dot(f0,f0)/len(x)) < tol: return x
        dx = gaussPivot(jac,-f0)
        x = x + dx
        if sqrt(dot(dx,dx)) < tol*max(max(abs(x)),1.0): return x
    print 'Too many iterations'
```

Applications of finite-differencing: Newton-Raphson method for systems

Jacobian matrix was calculated internally by finite-differencing.

First-order, forward finite difference used for calculating the Jacobian matrix

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

Higher derivatives

- Second (central) derivative:

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

- Third (central) derivative:

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

- Central differences are often preferred due to their nice properties (symmetry), but are not always appropriate, e.g. at the end of the interval, when the function is available only on one side → forward and backward ones used.

Forward and backward finite differences

- Forward and backward first derivatives:

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

- Second derivative:

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

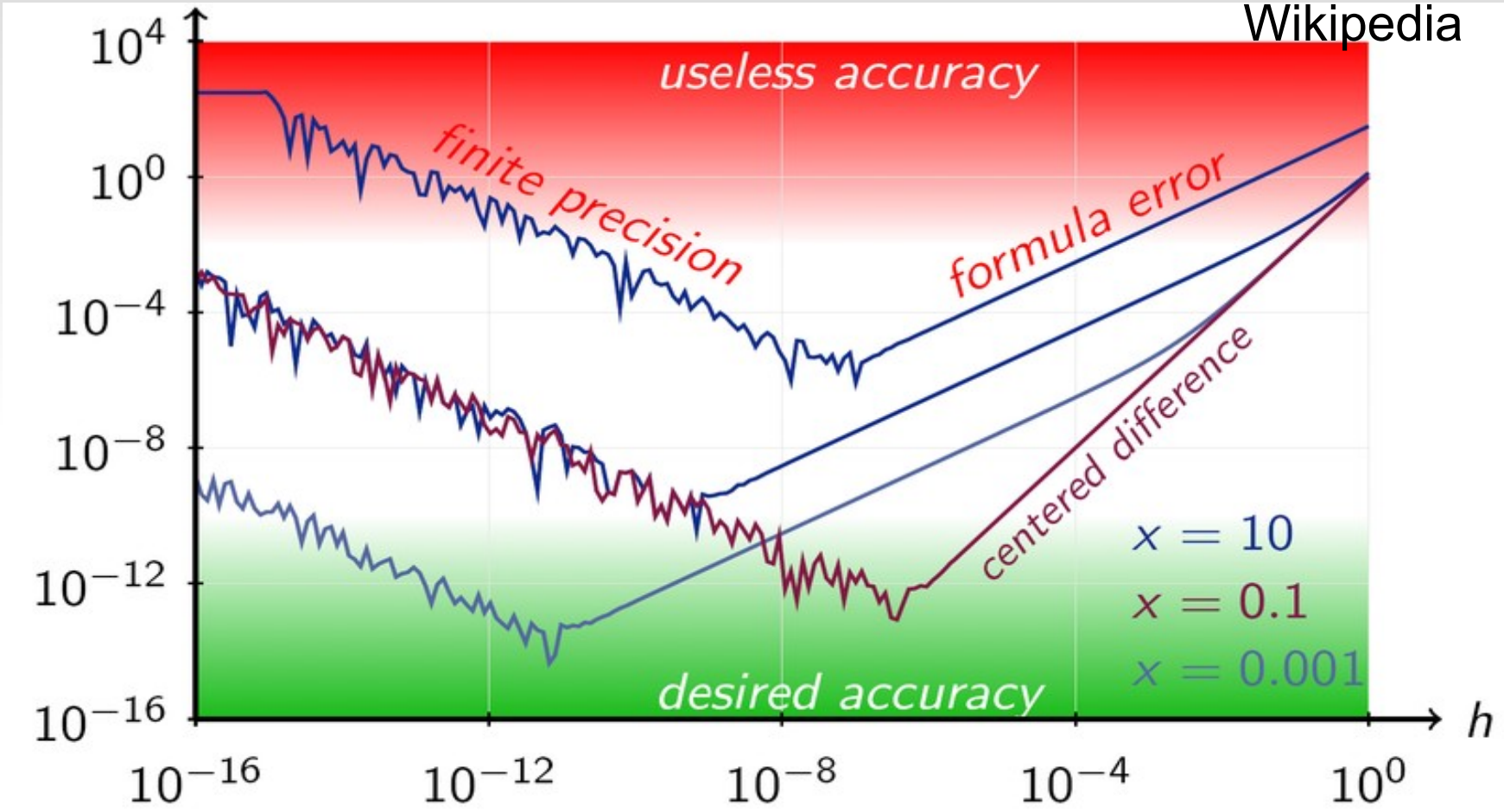
- However, we normally want errors to be smaller, at least second order in h, so we need to eliminate also the order-h terms (i.e. the f" terms), leading to e.g.:

$$f'(x) = \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} + \mathcal{O}(h^2)$$

Errors of finite-difference approximations

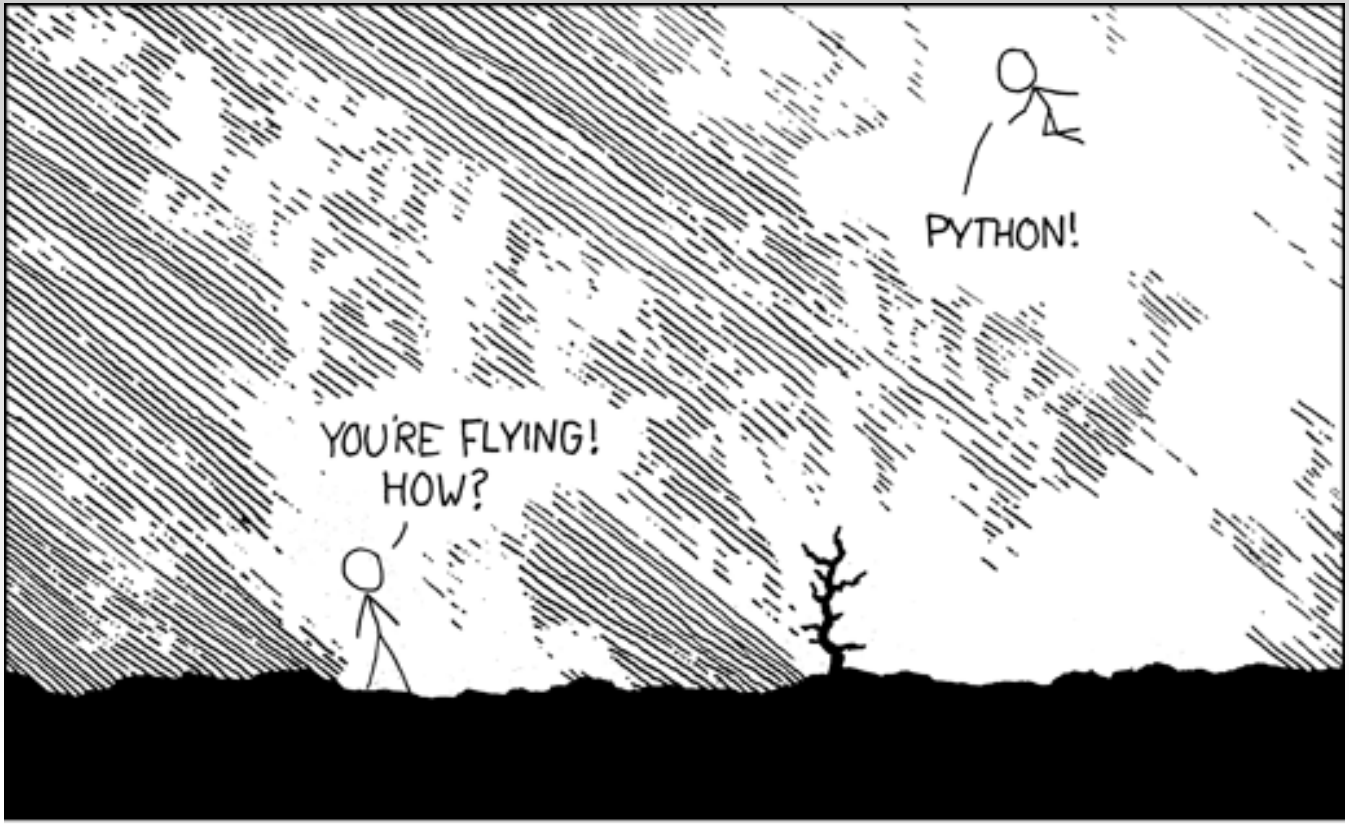
- In each finite-difference approximation the sum of the coefficients is zero (to achieve the cancellation).
- Thus, effects of round-off errors are significant. All expressions are then divided by (small) h, so roughly speaking, we divide 2 small numbers, potentially losing several significant digits in the process.
- Solution? There isn't one. However, several things help:
  - Use double-precision for calculations.
  - Use finite-difference formulae accurate to at least $O(h^2)$ (so that errors are minimized).

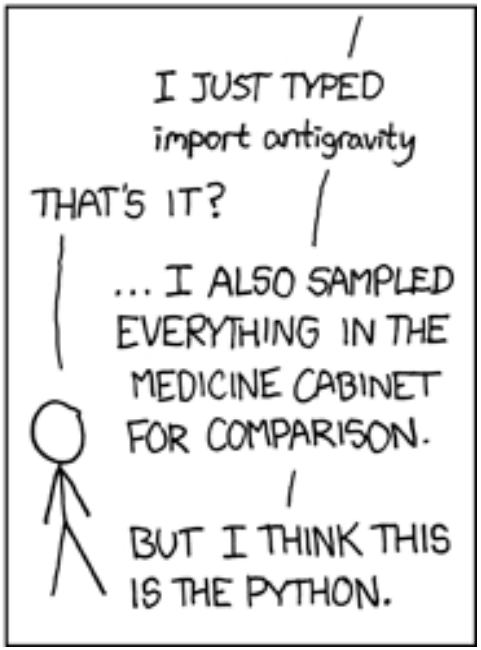
Difficulties in choosing h



Error of the numerically-calculated derivative of $f(x)=x^3$ using forward (first order) and centred (second order) expressions. Either too small or too large h is undesirable.

Why Python?





XKCD comics:
http://xkcd.com/

Richardson Extrapolation

- A method for increasing the accuracy of some numerical procedures, incl. finite-differencing.
- Assume we want to compute some quantity G(h) dependent on some parameter h.
- Let us write the quantity G as: G(h)=g(h) + E(h), where g(h) is the approximation and E(h) is the error.
- If we assume that E takes the form $E(h) = ch^p$ we can use the Richardson extrapolation method to eliminate the error.
- In practice, this eliminates the leading order of the error, making the approximation much more accurate (but not with zero error!).

Richardson Extrapolation

- First, compute g(h) for some values $h=h_1$ and $h=h_2$: $G = g(h_1) + ch_1^p$ $G = g(h_2) + ch_2^p$
- Eliminating c and solving for G, we find:

$$G = \frac{(h_1/h_2)^p g(h_2) - g(h_1)}{(h_1/h_2)^p - 1}$$

This is the Richardson extrapolation formula.

- Common choice is $h_2=h_1/2$, which yields:

$$G = \frac{2^p g(h_1/2) - g(h_1)}{2^p - 1}$$

Derivatives through interpolation

- Idea: instead of calculating the derivative by finite-differencing, we replace our function with an interpolated one and differentiate that.
- Very useful e.g. for unevenly-spaced data (varying h), in which case finite-differencing is tricky (and less accurate). For evenly-spaced data the two approaches are equivalent.
- Polynomial interpolation: OK for low-order polynomials, very bad (remember the oscillations!) for higher-order polynomials. We can use e.g. least-squares fitting to calculate the derivatives.

Derivatives through interpolation: splines

- Because of their numerical properties, splines are good (and easy) way to calculate derivatives.
- For example, first and second derivatives are:

$$f'_{i,i+1}(x) = \frac{k_i}{6}\left[\frac{3(x-x_{i+1})^2}{x_i - x_{i+1}} - (x_i - x_{i+1})\right]$$

$$-\frac{k_{i+1}}{6}\left[\frac{3(x-x_i)^2}{x_i - x_{i+1}} - (x_i - x_{i+1})\right] + \frac{y_i - y_{i+1}}{x_i - x_{i+1}}$$

$$f''_{i,i+1}(x) = k_i \frac{x - x_{i+1}}{x_i - x_{i+1}} - k_{i+1}\frac{x - x_i}{x_i - x_{i+1}}$$