Numerical methods: the basic tools

Andrea Ciardi

Laboratory for the Study of Radiation and Matter in Astrophysics
Sorbonne Université and Observatoire de Paris

Content

Numbers, approximations and errors

It is better to be vaguely right than exactly wrong

Carveth Read, "Logic: Deductive and Inductive" (1898)

Double precision, floating point numbers

Most scientific computations represent real numbers using double precision floating-point numbers:

- Single precision $\rightarrow$ 32 bits $=$ 4 Bytes
- Double precision $\rightarrow$ 64 bits $=$ 8 Bytes

This is a binary version of the scientific or engineering notation:

- $c = +2.99792458 \times 10^8$ m s^{-1} scientific notation
- $c = +0.299792458 \times 10^9$ or $+0.299792458E09$ m s^{-1} engineering notation, **use this second notation to initialize your variables!**

Floating-point numbers are represented as

$$x_f = (-1)^s \times 1.f \times 2^{e-b}$$

Double precision numbers are stored contiguosly (2 words of 32 bits):

$\rightarrow$ The sign s is stored as a single bit

$\rightarrow$ The exponent e is stored in the next 11 bits ($0 \leq e \leq 2047$). The actual exponent is $e - b$, with the bias $b = 1023$

$\rightarrow$ The fractional part of the mantissa f is stored in the remaining 52 bits

Double precision numbers have approximately 16 decimal places of precision and magnitudes in the range

$$4.9 \cdots \times 10^{-324} \rightarrow 1.8 \cdots \times 10^{308}$$

Representation scheme for IEEE doubles

Number name	Values of *s*, *e*, and *f*	Value of double
Normal	$0 < e < 2047$	$(-1)^s \times 2^{e-1023} \times 1.f$
Subnormal	$e = 0,\ f \neq 0$	$(-1)^s \times 2^{-1022} \times 0.f$
Signed zero	$e = 0,\ f = 0$	$(-1)^s \times 0.0$
$+\infty$	$s = 0,\ e = 2047,\ f = 0$	+INF
$-\infty$	$s = 1,\ e = 2047,\ f = 0$	−INF
Not a number	$s = u,\ e = 2047,\ f \neq 0$	NaN

Figure 1: IEEE representation

Exercise

Write a program that determines the underflow and overflow limits.

$\rightarrow$ Take a number, keep dividing or multiplying until it becomes 0 or INF

Machine precision

The floating-point representation used to store numbers in a computer is of limited precision.

Machine precision, ϵ_m is the maximum positive number that can be added to the number stored as 1 without changing it:

$$1_{comp} + \epsilon_m := 1_{comp}$$

For example $1.0 + 10^{-16} \rightarrow 1.0$ for a double precision number.

The computer representation of a number can then be written as:

$$x_c = x(1 \pm \epsilon_x)$$

where the error $|\epsilon_x| \leq \epsilon_m$

Exercise

Write a program to determine the machine precision (for double precision numbers)

Subtractive Cancelation

Let's consider the subtraction of two numbers $a = b - c$. In the computer representation this can be approximately written as

$$a_c \simeq b_c - c_c \simeq b(1+\epsilon_b) - c(1+\epsilon_c) \simeq a\left(1 + \frac{b}{a}\epsilon_b - \frac{c}{a}\epsilon_c\right)$$

→ For example, if $b \gg c$ then $a \sim b$ and $a_c \sim a(1+\epsilon_b)$

→ But if $b \sim c$

$$a_c \simeq a\left[1 + \frac{b}{a}(\epsilon_b - \epsilon_c)\right]$$

The errors do not necessarily cancel out and we cannot assume any sign for the errors. Moreover, the error is multiplied by the large number $\frac{b}{a}$ (remember that $b \sim c \to a \sim 0$).

Exercise

Explore subtractive cancelation for the function

$$f(x) = \frac{1 - \sqrt{1 - t(x)^2}}{t(x)}$$

where

$$t(x) = e^{-\pi x}$$

Interpolation

Introduction

$\rightarrow$ Data (mathematical functions, etc.) are represented as discrete sets of points.

$\rightarrow$ It is often the case that we need information on the data or a function (or its derivatives) at intermediate points, where we do not have the actual values.

$\rightarrow$ Interpolation is a method to obtain new data points from the discrete set of known data points.

It can be proven that given $n+1$ data points it is always possible to find a polynomial of order/degree n to pass through/reproduce the $n+1$ points.

Direct Method
Given $n+1$ data points the direct method assumes the following polynomial:

$$y = f(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$$

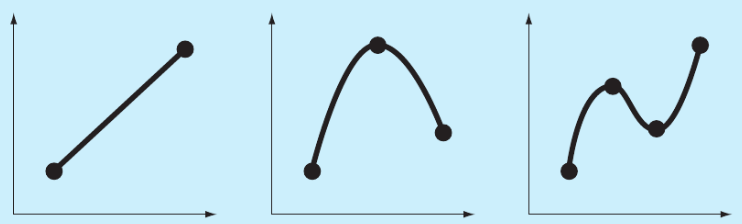

Figure 2: Examples of interpolating polynomials

Direct Method ii

With $n+1$ values for x and the $n+1$ corresponding values for y (i.e. the data set), we can solve for coefficients $a_0, a_1, ..., a_n$ by solving the $n+1$ simultaneous liner equations.

For example, given two data points (x_0, y_0) and (x_1, y_1), we can use the polynomial (linear function)

$$y = f(x) = a_0 + a_1 x$$

to pass through the two data points:

$$y_0 = a_0 + a_1 x_0$$

$$y_1 = a_0 + a_1 x_1$$

Solving for a_0 and a_1 gives the analytical expression of the function $f(x)$, which can be used as the basis for interpolation - estimating the missing data points y in-between x_0 and x_1.

Newton's divided-difference interpolating polynomial i

Given a set of $n+1$ data points

$(x_0, y_0), \ldots, (x_j, y_j), \ldots, (x_k, y_k)$

where no two x_j are the same, the Newton interpolation polynomial is a linear combination of Newton basis polynomials:

$$N(x) := \sum_{j=0}^{k} a_j n_j(x)$$

with the Newton basis polynomials defined as

$$n_j(x) := \prod_{i=0}^{j-1} (x - x_i)$$

for $j > 0$ and $n_0(x) \equiv 1$.

Newton's divided-difference interpolating polynomial ii

The coefficients of the polynomial are defined as

$$a_j := [y_0, \ldots, y_j]$$

where

$$[y_0, \ldots, y_j]$$

is the notation for **divided differences**.

Example of divided differences

$$
\begin{aligned}
[y_0] &= y_0 \\
[y_0, y_1] &= \frac{y_1 - y_0}{x_1 - x_0} \\
[y_0, y_1, y_2] &= \frac{[y_1, y_2] - [y_0, y_1]}{x_2 - x_0} = \frac{\frac{y_2 - y_1}{x_2 - x_1} - \frac{y_1 - y_0}{x_1 - x_0}}{x_2 - x_0} = \frac{y_2 - y_1}{(x_2 - x_1)(x_2 - x_0)} - \frac{y_1 - y_0}{(x_1 - x_0)(x_2 - x_0)} \\
[y_0, y_1, y_2, y_3] &= \frac{[y_1, y_2, y_3] - [y_0, y_1, y_2]}{x_3 - x_0}
\end{aligned}
$$

Newton's divided-difference interpolating polynomial iii

Thus Newton polynomial can be written as

$$N(x) = [y_0] + [y_0, y_1](x - x_0) + \cdots + [y_0, \ldots, y_k](x - x_0)(x - x_1)\cdots(x - x_{k-1})$$

Given two data points Newton's polynomial takes the form

$$y = f(x) = a_0 + a_1(x - x_0)$$

with the coefficients

$$a_0 = y_0$$

$$a_1 = \frac{y_1 - y_0}{x_1 - x_0}$$

The first derivative of the function at $x = x_0$ is $f'(x_0) = a_1 = \frac{y_1 - y_0}{x_1 - x_0}$, which is the *forward* divided difference.

This is another form for the interpolating polynomial and it is essentially a reformulation of Newton polynomial in order to avoid the computation of divided differences.

The Lagrange form specifies the interpolation polynomial as:

$$f_n(x) := \sum_{i=0}^{n} L_i(x) f(x_i)$$

$$L_i(x) := \prod_{\substack{0 \le m \le n \\ m \ne i}} \frac{x - x_m}{x_i - x_m} = \frac{(x - x_0)}{(x_i - x_0)} \cdots \frac{(x - x_{i-1})}{(x_i - x_{i-1})} \frac{(x - x_{i+1})}{(x_i - x_{i+1})} \cdots \frac{(x - x_n)}{(x_i - x_n)}$$

where n is the order of the polynomial.

Lagrange interpolating polynomial ii

Example

For two data points, the polynomial of order $n = 1$ is

$$\begin{aligned} f_1(x) &= L_0 f(x_0) + L_1 f(x_1) \\ &= \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1) \end{aligned}$$

The first derivative also matches that of the divided difference method:

$$\begin{aligned} f'(x) &= \frac{f(x_0)}{x_0 - x_1} + \frac{f(x_1)}{x_1 - x_0} \\ &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \end{aligned}$$

Note that $L_i(x) = 1$ at $x = x_i$ and zero for all other sample points.

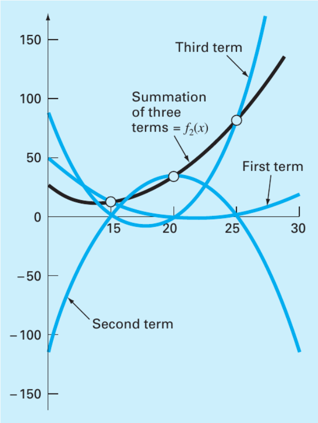

Figure 3: Example of a second order polynomial in Lagrange form: $f_2(x) = L_0(x)f(x_0) + L_1(x)f(x_1) + L_2(x)f(x_2)$

Discussion on interpolation i

- Direct method requires more computational effort for large number of data points $n+1$ and can yield inaccurate results
- Newton and Lagrange methods have similar computational effort. Lagrange is easier to program.
- Use Newton if the order of the polynomial is not known a priori, Lagrange otherwise.

Discussion on interpolation ii

Fitting high-order polynomials can lead to oscillatory behaviour

$\longrightarrow$ need for piecewise interpolation or other methods

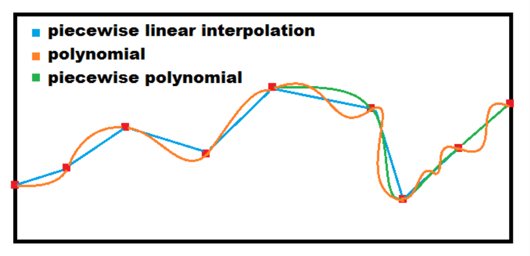

Figure 4: Example of piecewise interpolation

See jupyter notebook

You can explore polynomial interpolation and Runge's phenomenon in the jupyter nobook `Runge_phenomenon.ipynb`

A commonly used technique is spline interpolation. In this case the function used to interpolate is a piecewise polynomial called a spline.

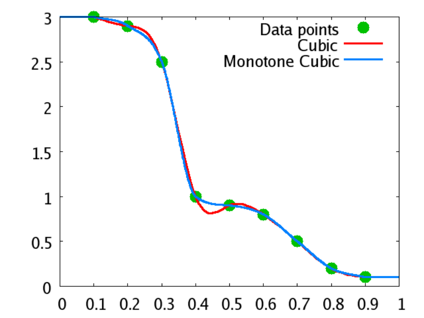

Figure 5: Monotonic cubic interpolation using cubic Hermite splines

Least-square regression

Introduction

Fitting a polynomial to a set of data is not always a good idea, especially experimental data!

A better strategy is to derive an approximating function that fits the shape or general trend of the data.

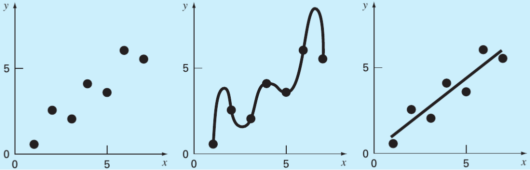

Figure 6: Least-squares fit vs polynomial interpolation

The simplest is to fit a straight line $f(x) = a_0 + a_1 x$ to a set of data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$:

$$y_i = f(x_i) + \epsilon_i = a_0 + a_1 x_i + \epsilon_i$$

where ϵ, the error or residual, is the discrepancy between the true value of y_i (the data) and the model $f(x)$:

$$\epsilon_i = y_i - (a_0 + a_1 x_i)$$

One strategy that provides the "best fit" to the data is to minimize the sum of the squares of the residuals:

$$S(a_0, a_1) = \sum_{i=1}^{n} (\epsilon_i)^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 = \sum_{i=1}^{n} [y_i - a_0 - a_1 x_i]^2$$

We can minimize the function $S(a_0, a_1)$ by setting the gradient to zero:

$$\frac{\partial S}{\partial a_0} = -2 \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i) = 0$$

$$\frac{\partial S}{\partial a_1} = -2 \sum_{i=1}^{n} [(y_i - a_0 - a_1 x_i) x_i] = 0$$

Linear regression ii

Let's rewrite the first equation as:

$$-\sum_{i=1}^{n} y_i + \sum_{i=1}^{n} a_0 + \sum_{i=1}^{n} a_1 x_i = 0 \;\rightarrow\; -\sum_{i=1}^{n} y_i + n a_0 + \sum_{i=1}^{n} a_1 x_i = 0 \;\rightarrow\; a_0 n + a_1 \sum_{i=1}^{n} x_i = \sum_{i=1}^{n} y_i$$

and the second as:

$$-\sum_{i=1}^{n} y_i x_i + \sum_{i=1}^{n} a_0 x_i + \sum_{i=1}^{n} a_1 x_i^2 = 0 \;\rightarrow\; a_0 \sum_{i=1}^{n} x_i + a_1 \sum_{i=1}^{n} x_i^2 = \sum_{i=1}^{n} x_i y_i$$

The unknown a_0 and a_1 are then given by:

$$a_1 = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$a_0 = \frac{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n \sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}$$

.

Using a more compact notation, which makes it easier for programming the method, let's define:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

$$S_1 = \sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}$$

$$S_2 = \sum_{i=1}^{n} x_i^2 - n\bar{x}^2$$

then

$$a_1 = \frac{S1}{S2}$$

$$a_0 = \bar{y} - a_1\bar{x}$$

.

Exercise

Write a program that performs a fit to data using linear regression and use it to study the solar wind data from the ULYSSE satellite.

Solar wind exercise ii

For a steady-state, spherically symmetric freely expanding solar wind, conservation of mass and momentum give:

$$4\pi r^2 n v = K_1$$

$$4\pi r^2 n v^2 = K_2$$

which give:

$$v = const.$$

$$n \propto r^{-2}$$

Therefore we want to fit a power law

$$n = \alpha r^{\beta}$$

taking the logarithm we have

$$\ln n = \ln \alpha + \beta \ln r$$

Solar wind exercise iii

so we have to perform our **linear** regression

$$y' = a_0 + a_1 x'$$

on the variable y' and x' defined as

$$y' = \ln n$$

$$x' = \ln r$$

The fitting parameters are linked to the power law parameters by:

$$\beta = a_1$$

$$\alpha = e^{a_0}$$

Finding roots

Introduction i

The roots or zeros of a function $f(x)$ are the values x such that

$$f(x) = 0$$

or equivalently, finding the roots of $f(x) - g(x) = 0$ is the same as solving the equation

$$f(x) = g(x)$$

For example, the equation of state of real gases can be approximated by the van der Waals equation:

$$\left(p + \frac{an^2}{V^2}\right)\left(\frac{V}{n} - b\right) = RT$$

finding the volume for a range of temperatures, pressures, and number of moles requires a root finding algorithm.

Many root finding algorithms exist and most are iterative schemes, that is, they approach the "solution" by subsequent improved approximations.

- Bracketing methods
  - Bisection method
  - False-position method
  - ...
- Open methods
  - Newton-Raphson method
  - Secant method
  - ...

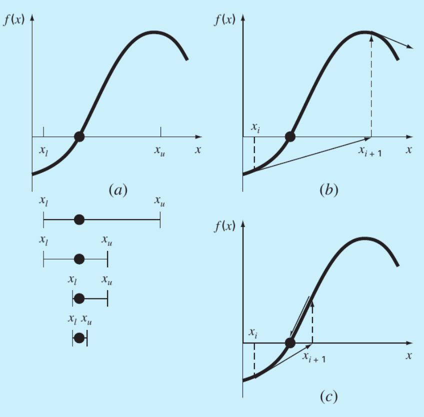

Figure 7: Comparison of bisection and open methods

Bisection method i

The bisection method starts with an interval $[a, b]$ (i.e. two guesses). If the function is continuous and $f(a)f(b) < 0$ then the root can be found, but it may be slow to get there...

The algorithms proceeds

$\rightarrow$ by cutting in half the size of the interval and

$\rightarrow$ finding which sub-interval satisfies $f(x_1)f(x_2) < 0$.

$\rightarrow$ The procedure (cut and check) is repeated until the error is "small enough!". The numerical root is the mid-point of the last interval

Exercise

Use the bisection method to find the root of the function

$$f(x) = \frac{1}{2} - e^{-x}$$

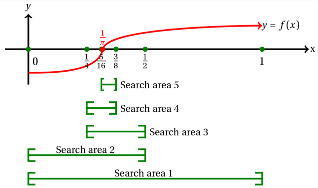

Figure 8: The bisection method

False-position method

The bisection is a "brute-force" method that does not take into account the properties of the function it is trying to find the roots of.

The *regula falsi* or false-position or linear interpolation method uses the relative values of the function at the end points to find an improved estimate of the root, which is given by:

$$x_r = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

$\rightarrow$ the new sub-interval, either $[x_l, x_r]$ or $[x_r, x_u]$, is chosen as in the bisection method.

Exercise

Use the false-position method to find the root of the function

$$f(x) = \frac{1}{2} - e^{-x}$$

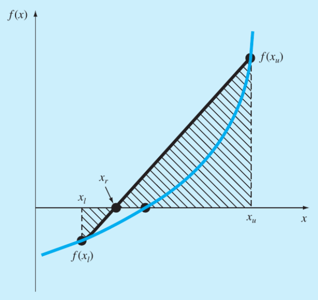

Figure 9: The regula falsi method

Newton-Raphson Method

This is one of the most widely used root finding algorithms. It requires:

$\rightarrow$ one initial guess value for the root

$\rightarrow$ the first derivative of the function

The improved estimate of the root is given by:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

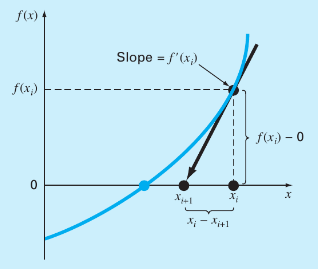

Figure 10: Newton-Raphson method

Newton-Raphson Method

This is one of the most widely used root finding algorithms. It requires:

→ one initial guess value for the root

→ the first derivative of the function

The improved estimate of the root is given by:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

But in some cases it can fail!

Backtracking is a technique to stop the search of the zero and change the guess value.

If a new guess $x_i + \delta x$, where $\delta x = -f(x)/f'(x)$, increases the magnitude of the function

$$|f(x_i + \delta x)| > |f(x_i)|$$

then we go back to x_i and decrease the size of δx by a factor α and keep doing it until

$$|f(x_i + \delta x)| < |f(x_i)|$$

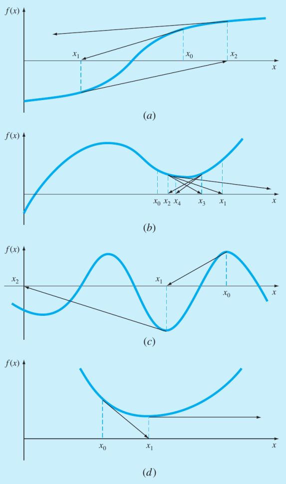

Figure 10: Examples of poor convergence of the Newton-Raphson method

The Secant Method

The secant method uses the numerically evaluated derivative. This is needed when the analytical expression of the derivative of the function $f(x)$ is not known or easily evaluated.

In the secant method the derivative is approximated as a *backward finite divided difference*:

$$f'(x) \simeq \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

The iterative equation for the improved estimate of the root is then given by:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

This approach requires two initial estimates of x

$\rightarrow$ the secant method and the false position method are similar, but different in important ways.

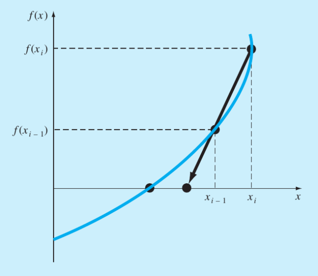

Figure 11: The secant method

The secant method uses the numerically evaluated derivative. This is needed when the analytical expression of the derivative of the function $f(x)$ is not known or easily evaluated.

In the secant method the derivative is approximated as a *backward finite divided difference*:

$$f'(x) \simeq \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

The iterative equation for the improved estimate of the root is then given by:

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

This approach requires two initial estimates of x

$\rightarrow$ the secant method and the false position method are similar, but different in important ways.

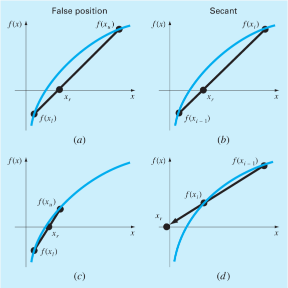

Figure 11: Comparison of the false position and secant methods

The modified Secant Method

Instead of using two values to estimate the initial derivative, we can use a small perturbation δx_i to calculate the derivative. In this case the iterative equation is:

$$x_{i+1} = x_i - \frac{\delta x_i f(x_i)}{f(x_i + \delta x_i) - f(x_i)}$$

Exercise

Use the open methods discussed (Newton-Raphson, secant and modified secant) to find the root of the function

$$f(x) = \frac{1}{2} - e^{-x}$$

- compare the various methods (speed, number of iterations, accuracy, ...) and discuss your results