Root Finding (and Optimisation)

M.Sc. in Mathematical Modelling & Scientific Computing,
Practical Numerical Analysis

Michaelmas Term 2018, Lecture 4

Root Finding

The idea of root finding is simple — we want to find $\mathbf{x}$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{0}$.

Actually finding such roots is not so simple. For example if $f(x)$ is a cubic polynomial

$$f(x) = ax^3 + bx^2 + cx + d \,,$$

then

$$x = \sqrt[3]{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right) + \sqrt{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right)^2 + \left(\frac{c}{3a} - \frac{b^2}{9a^2}\right)^3}} + \sqrt[3]{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right) - \sqrt{\left(\frac{-b^3}{27a^3} + \frac{bc}{6a^2} - \frac{d}{2a}\right)^2 + \left(\frac{c}{3a} - \frac{b^2}{9a^2}\right)^3}} - \frac{b}{3a} \,.$$

Root Finding

If $f(x)$ is a quintic polynomial

$$f(x) = ax^5 + bx^4 + cx^3 + dx^2 + ex + f \ ,$$

then there is a closed form solution for the roots iff its Galois group is solvable (Galois, 1846).

Closed form solutions to the general root-finding problem may:

- not exist;
- be unstable;
- require evaluation of special functions.

In such cases numerical methods help!

Relation to Optimisation

We know that if we want to maximise or minimise a function $g(\mathbf{x})$ then the maximum or minimum occurs at a turning point of g (or on the boundary of the domain) so that

$$f_i \quad := \quad \frac{\partial g}{\partial x_i} \quad = \quad 0 \, .$$

In other words, we have a root finding problem, $\mathbf{f}(\mathbf{x}) = \mathbf{0}$.

Univariate Root Finding

Bisection Algorithm

In 1D the problem is:

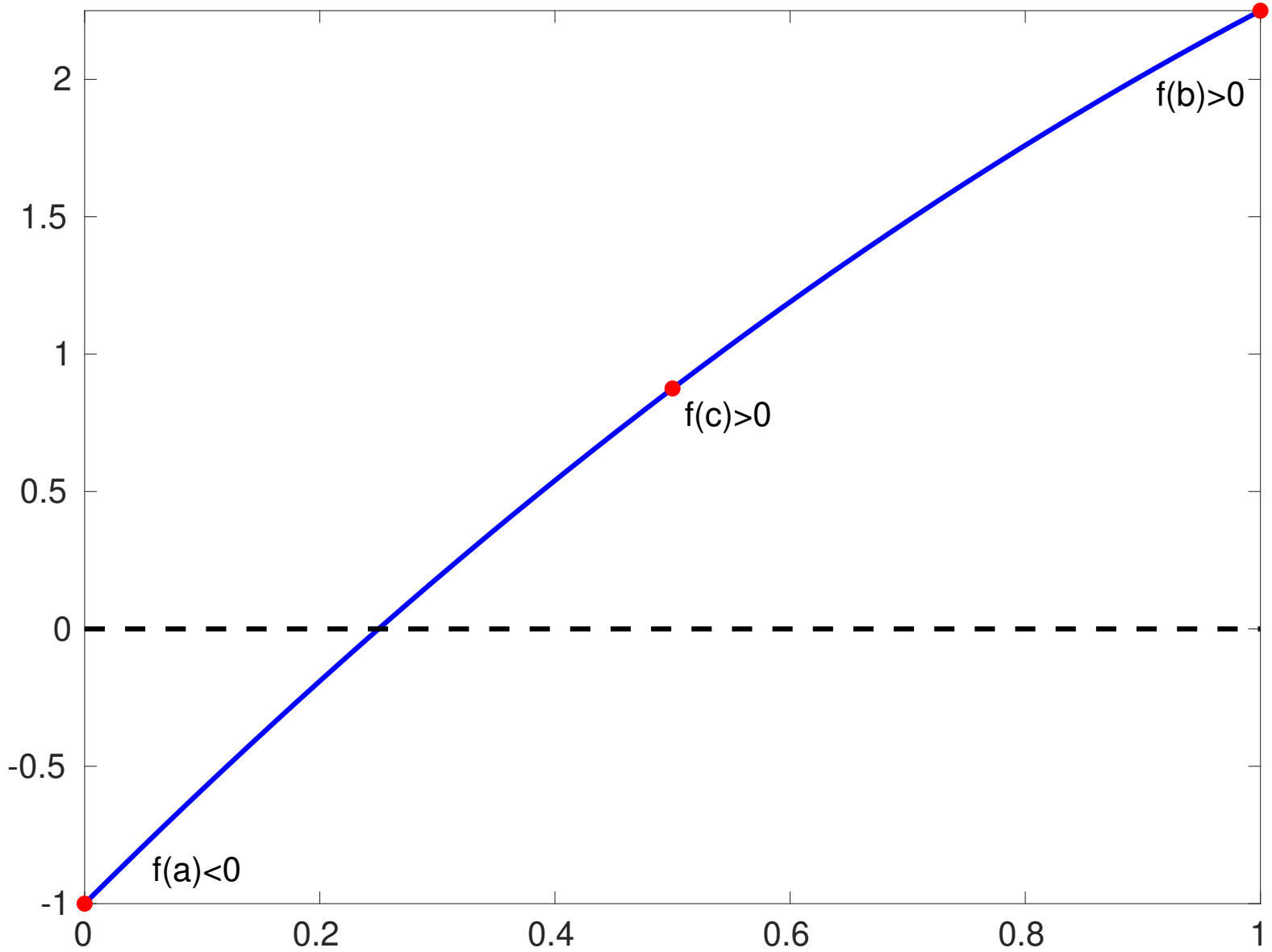
$$\text{Given } f : [a, b] \to \mathbb{R}, \text{ find } c \in [a, b] \text{ such that } f(c) = 0.$$

The bisection algorithm relies on the intermediate value theorem which states:

If $f : [a, b] \to \mathbb{R}$ is continuous and $f(a) < u < f(b)$ or $f(b) < u < f(a)$ then $\exists c \in (a, b)$ such that $f(c) = u$.

Thus, to find a root of $f(x)$ we find a and b such that $f(a)$ and $f(b)$ have opposite sign. We set $c = (a + b)/2$, compute $f(c)$ and then repeat on the half of the interval where the sign changes.

Bisection Algorithm



Bisection Algorithm

We need to decide when to terminate the algorithm — the normal choices are one of:

- $|b - a| <$ tol (solution is only changing by a very small amount);
- $|f(c)| <$ tol (residual is very small).

For convergence, since $c \in [a, b]$ the maximum error in c is $b - a$ (i.e. the width of the interval).

Since each step halves the length of the interval, the error at the nth iteration is

$$|c_n - c| \quad \leq \quad \frac{b-a}{2^n} \, .$$

Thus to achieve an accuracy of tol requires

$$n \quad = \quad \frac{\log(b-a) - \log(\mathrm{tol})}{\log(2)}$$

steps of the bisection algorithm.

Regula Falsi

The regula falsi algorithm is similar to the bisection algorithm except that it uses a potentially more intelligent guess for c.

Suppose we approximate the function f on $[a, b]$ by its linear interpolant

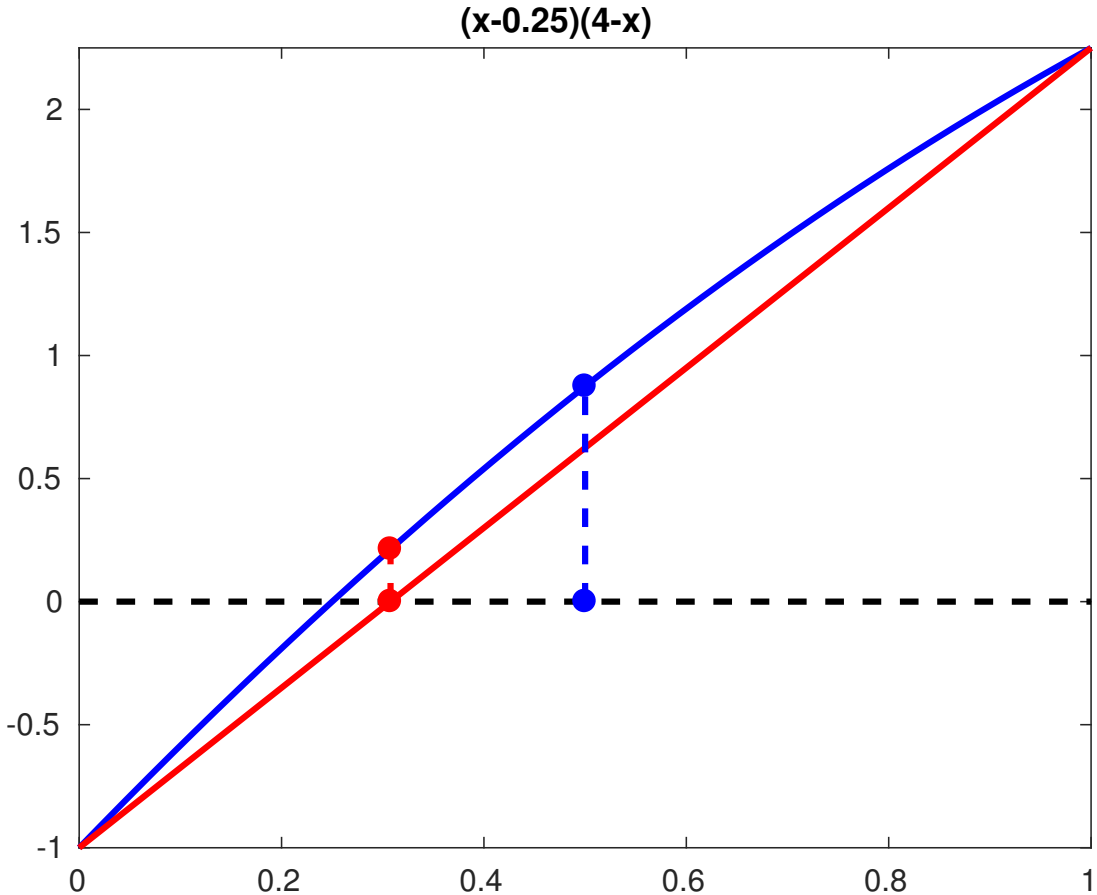
$$g(x) = \frac{x-a}{b-a} f(b) + \frac{b-x}{b-a} f(a) .$$

Then $g(x)$ has a root at

$$c = \frac{af(b) - bf(a)}{f(b) - f(a)} .$$

The regula falsi algorithm uses this as c in the bisection algorithm.

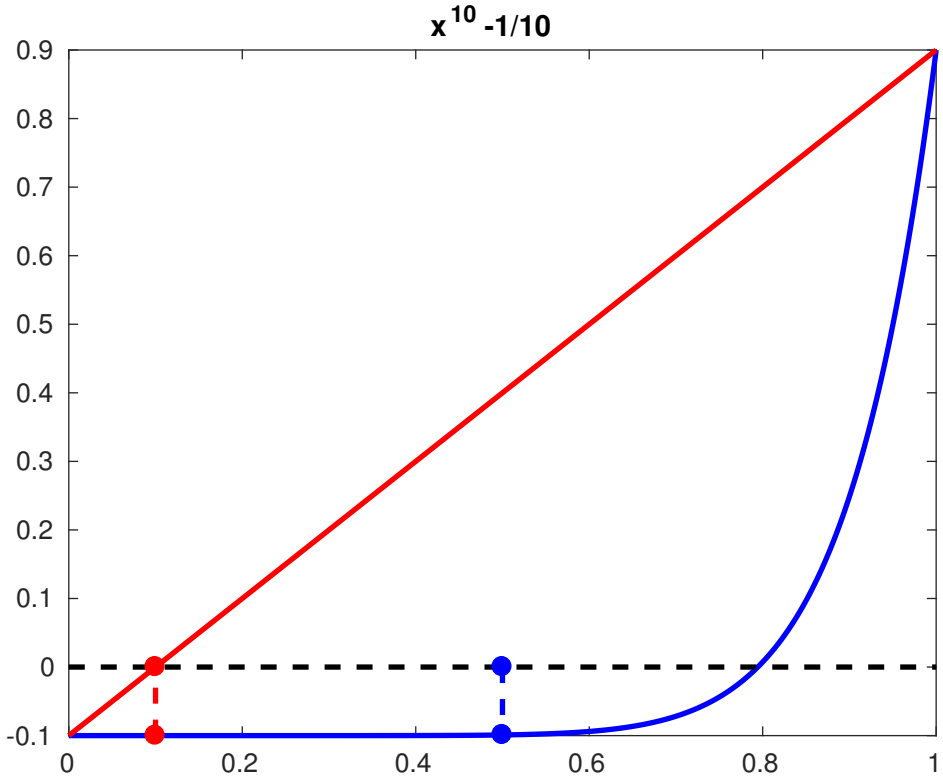
Regula Falsi — Example When it Works Well



Blue dots correspond to standard bisection algorithm, red dots to regula falsi.

Regula Falsi — Example When it Works Badly

Convergence can be slow when the curvature of f is large (when the linear approximation is not good).



Blue dots correspond to standard bisection algorithm, red dots to regula falsi.

Regula Falsi — Fix

There is a simple fix. If the algorithm goes right twice then halve $f(b)$ so

$$c \quad = \quad \frac{af(b)/2 - bf(a)}{f(b)/2 - f(a)} \, .$$

If the algorithm goes left twice then halve $f(a)$.

This method is sometimes called the **Illinois algorithm**.

Newton-Raphson

Suppose we approximate $f(x)$ by $g(x)$ which is the truncated Taylor series of f about the point x_k so

$$g(x) \quad = \quad f(x_k) + (x - x_k) f'(x_k) \; .$$

Then $g(x)$ has a root at c given by

$$c \quad = \quad x_k - \frac{f(x_k)}{f'(x_k)} \; .$$

Thus the Newton-Raphson method requires an initial guess x_0 and then iterates

$$x_{k+1} \quad = \quad x_k - \frac{f(x_k)}{f'(x_k)} \; ,$$

until convergence is achieved.

If the initial guess is "sufficiently close" to the solution, the algorithm has quadratic convergence.

Newton-Raphson Modifications

The iterate for the **secant method** is

$$x_{k+1} \quad = \quad x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \ .$$

This avoids the need to compute $f'(x_k)$.

For **damped Newton** we use

$$x_{k+1} \quad = \quad x_k - \sigma_k \frac{f(x_k)}{f'(x_k)} \ ,$$

where $\sigma_k \in (0, 1]$.

A third order variation is **Halley's method**:

$$x_{k+1} \quad = \quad x_k - \frac{2f(x_k)f'(x_k)}{2(f'(x_k))^2 - f(x_k)f''(x_k)} \ .$$

This works if the roots x_* satisfy $f'(x_*) \neq 0$ and is then Newton's method applied to the function $f(x)/\sqrt{|f'(x)|}$.

Multivariate Root Finding

Higher Dimensions

The problem is now find $\mathbf{x}$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$.

Algorithms based on the bisection algorithm are not extendable to higher dimensions.

Algorithms based on Newton's method are!

Newton's Method

Again we approximate $\mathbf{f}(\mathbf{x})$ by the first terms in its Taylor series:

$$\mathbf{f}(\mathbf{x}+\boldsymbol{\delta}\mathbf{x}) \approx \mathbf{f}(\mathbf{x}) + J(\mathbf{x})\boldsymbol{\delta}\mathbf{x}\,,$$

where $J(\mathbf{x})$ is the Jacobian evaluated at the point $\mathbf{x}$,

$$J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}.$$

In order to get the Newton iterate we solve $\mathbf{f}(\mathbf{x}+\boldsymbol{\delta}\mathbf{x}) = \mathbf{0}$, i.e.

$$J(\mathbf{x})\boldsymbol{\delta}\mathbf{x} = -\mathbf{f}(\mathbf{x})\,,$$

so that

$$J(\mathbf{x}_k)(\mathbf{x}_{k+1}-\mathbf{x}_k) = -\mathbf{f}(\mathbf{x}_k)\,.$$

Newton's Method

We can also write the Newton iterate as

$$\mathbf{x}_{k+1} \quad = \quad \mathbf{x}_k - (J(\mathbf{x}_k))^{-1}\mathbf{f}(\mathbf{x}_k) \; .$$

Thus a damped version is possible as in 1D:

$$\mathbf{x}_{k+1} \quad = \quad \mathbf{x}_k - \sigma_k(J(\mathbf{x}_k))^{-1}\mathbf{f}(\mathbf{x}_k) \; .$$

Newton's Method and Newton Fractals

One of the problems with using Newton's method is choosing an initial guess. Generally root finding problems have many solutions (think of polynomials) and different initial guesses lead to different solutions with different numbers of iterations. Newton fractals illustrate this behaviour in the complex plane.

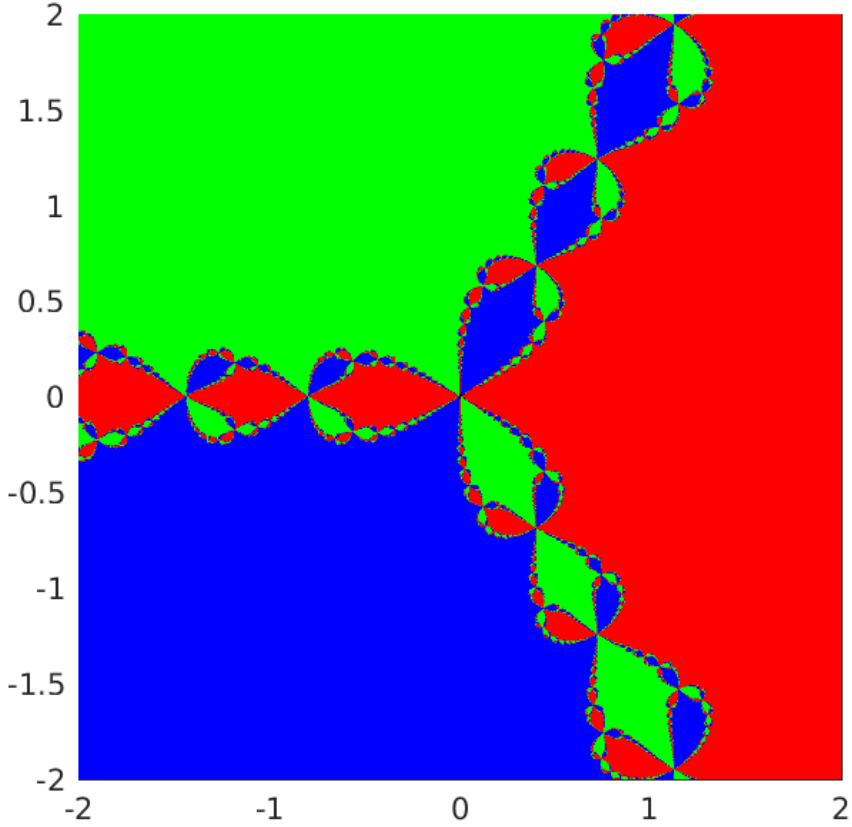
Newton Fractals

For example, suppose we consider the function $f(z) = z^3 - 1$. This has three roots at $z = 1$ and $z = (-1 \pm \sqrt{3}\imath)/2$. By writing $z = x + \imath y$ and equating real and imaginary parts we get a system

$$\begin{aligned} x^3 - 3xy^2 - 1 &= 0 \\ 3x^2y - y^3 &= 0 \end{aligned}$$

which can be solved using Newton's method. For each point $(x, y) \in [-2, 2]^2$ we compute a solution using Newton's method and record which root it converges to. A Newton fractal is a plot of this information. Shading using the number of iterations for convergence is also possible.

Newton Fractal Example



Newton fractal for $f(z) = z^3 - 1$.