Numerical Root Finding

1 Motivation

Many mathematcial problems can be written as solving

$$f(x) = c \tag{1}$$

for x. This general problem may be re-expressed as a *root-finding* problem by letting

$$g(x) = f(x) - c \tag{2}$$

and solving

$$g(x) = 0. \tag{3}$$

Root-finding algorithms are a prime example of *iterative algorithms*, where successive approximations of the solution converge on the real root.

The stopping criteria for these algorithms can be either a certain number of iterations, or when a desired accuracy is obtained (e.g. $f(x_n) = 10^{-6}$ say, in the case of a root finding problem).

In contrast to this, a *direct method* solve exactly a problem using a finite sequence of operations.

Firstly, let us look at a simple and robust example; *bracketing bisection.*

2 Bracketing Bisection

For a continuous function, $f(x)$, given $f(a) > 0$ and $f(b) < 0$, there exists $p \in [a, b]$ such that $f(p) = 0$ by the intermediate value theorem. Take c to be the mid-point of $[a, b]$.

If $f(a)$ and $f(c)$ have opposing signs then the root is in $[a, c]$. Thus, we can disregard $[c, b]$. We then begin a new search by finding the midpoint of $[a, c]$.

Similarly if $f(c)$ and $f(b)$ have opposing signs, we concentrate our attention on $[c, b]$.

Repeated iterations of this procedure will converge on the root, p.

Bracketing bisection converges linearly ($R = 1$), giving a range of values within which the root, p, is contained. After n iterations, the best estimate of the root will be at the middle of this range and will have error

$$\frac{|b-a|}{2^{n+1}} \tag{4}$$

Algotithm

1. Choose a, b such that $f(a)f(b) < 0$.
2. Find midpoint of $[a, b]$.
3. Evaluate f(c).
4. If $f(a)f(c) < 0$, set $b = c$, if $f(c)f(b) < 0$, set a=c, if $f(c) = 0$, finish.
5. Repeat $2, 3, 4$.

Example

We will use bracketing bisection to find the square root of 2.

To do this, we solve

$$f(x) = x^2 - 2 = 0. \tag{5}$$

Our initial guesses are $a = 1$ and $b = 2$. Below is a table of the early iterations of the algorithm. These values are plotted in Fig.1.

n	a	$f(a)$	b	$f(b)$	c	$f(c)$
1	1	-1	2	2	1.5	0.25
2	1	-1	1.5	0.25	1.25	-0.4375
3	1.25	-0.4375	1.5	0.25	1.375	-0.109375
4	1.375	-0.109375	1.5	0.25	1.4375	0.06640625

3 Newton Raphson

A extremely efficient root finding method is the *Newton-Raphson* algorithm. It is a typical iterative algorithm, using the approximate solution from the n^{th} iteration, x_n, to find a new, more accurate solution, x_{n+1}, to the equation, $f(x) = 0$.

The new approximation, x_{n+1}, is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \tag{6}$$

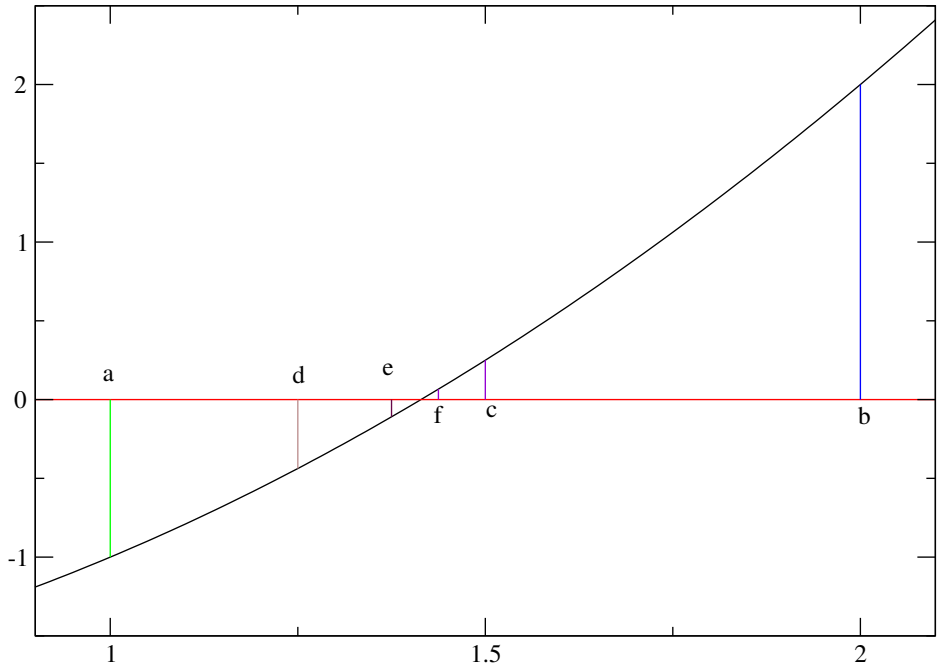


Figure 1: Early iterations of Bracketing Bisection.

3.1 Algorithm

1. Choose an appropriate starting point, x_0.
2. Evaluate $f(x_i)$, $f'(x_i)$.
3. Calculate $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$.
4. Repeat $2, 3$.

If the calculation of the derivative is complicated, or its evaluation is numerically expensive, the derivative may be approximated directly from a Newton Quotient style expression;

$$f'(x) = \frac{f(x + \Delta) - f(x)}{\Delta}. \tag{7}$$

Here, the choice of Δ is crucial. If Δ is too big, the value may differ from the actual derivative. If Δ is too small, rounding errors may occur.

The efficiency of Newton-Raphson is greatly affected by the choice of starting point. Some iteration point conditions which may cause Newton-Raphson to fail include

- Iteration point stationary ($f'(x_i) = 0$).
- Starting point enters a cycle (e.g. $x_0 \to x_1 \to x_2 = x_0 \to \ldots$).
- Derivative non-existent.
- Discontinuous derivative.

Newton-Raphson coverges quadratically with $R = 2$. At this point, let us formalise the rate of convergence, R.

3.2 Rate of Convergence

The *rate of convergence*, R, of an iterative method, measures the speed at which a convergent sequence of approximations of a solution approaches the limit, L;

$$1 = \lim_{k\to\infty} \frac{|x_{k+1} - L|}{|x_k - L|^R} \tag{8}$$

$$R = \frac{\log|x_{k+1} - L|}{\log|x_k - L|}. \tag{9}$$

Generally, the higher the rate of convergence, the fewer iterations the method will require to approach the correct answer.

Tabled below are the details, including the rate of convergence, R , of the early iterations of a Newton-Raphson procedure to solve

$$f(x) = (x^3 + x - 10)(x^3 + 3), \tag{10}$$

with a known solution at $x = 2$.

n	x_n	$f(x_n)$	$\lvert 2 - x_n\rvert$	R
0	2.209121	41.20669	2.091212×10^{-1}	–
1	2.050240	7.765416	5.023968×10^{-2}	1.911344
2	2.003623	0.5210658	3.623381×10^{-3}	1.879118
3	2.000021	2.97705×10^{-3}	2.081786×10^{-5}	1.917977
4	2	5.583022×10^{-7}	3.904211×10^{-9}	1.796081

Note how the error decreases roughly quadratically, while the rate of convergence is roughly 2.

However, a value of R will not necessarily imply numerical efficiency. For example, Newton-Raphson ($R = 2$) will require fewer iterations to converge on the solution than bracketing bisection ($R = 1$).

However, Newton-Raphson requires two function evaluations per iteration (function and derivative), whereas bracketing bisection only requires one.

Thus, though fewer recursions of Newton-Raphson are required, if the particular function is numerically expensive, this algorithm may ultimately be slower than bracketing bisection.

4 Secant Method

The *secant method* is an iterative root-finding method which uses information from the previous two iterations to find a new approximation of the solution to the equation

$$f(x) = 0. \tag{11}$$

This is done by picking two point on the function, $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$, and finding the equation of the line joining these two points. The point where this line cuts the x-axis is taken to be the new approximation, x_{i+1}.

The point x_{i-1} is then disregarded and, x_i and x_{i+1} are used to find x_{i+2}, and so on.

We must now write down an expression for x_{i+1}. Consider the line through the points $(x_{i-1}, f(x_{i-1}))$ and $(x_i, f(x_i))$. The equation of this line is

$$y - f(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}(x - x_i). \tag{12}$$

The point x_{i+1} is given by the root of this line. i.e. The line passes through the point $(x_i, 0)$. Putting this point into Eq.12, we see that

$$-f(x_i) = \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}(x_{i+1} - x_i). \tag{13}$$

This rearranges to give us

$$x_{i+i} = x_i - \frac{(x_i - x_{i-1})}{f(x_i) - f_(x_{i-1})} f(x_i) \tag{14}$$

The rate of convergence of the secant method is $R = 1.618$. It requires one function evaluation per iteration.

It is subject to failure conditions similar to Newton Raphson:

- $f(x_i) - f_(x_{i-1}) = 0$.
- Starting point enters a cycle (e.g. $x_0 \to x_1 \to x_2 = x_0 \to \ldots$).

Algorithm

1. Choose appropriate starting points, x_0 and x_1.
2. Evaluate $f(x_0)$.
3. Evaluate $f(x_i)$.
4. Calculate $x_{i+1} = x_i - \frac{(x_i - x_{i-1})}{f(x_i) - f_(x_{i-1})} f(x_i)$.
5. Repeat $3, 4$.

Muller's Algorithm

The secant method worked by approximating the function, $f(x)$, linearly, and using the root of this line to give x_{i+1} in terms of x_i and x_{i-1}. The next complication would be to approximate $f(x)$ quadratically. Following the same reasoning, we would obtain an expression for x_{i+1} in terms of x_i, x_{i-1} and x_{i-2}. This is *Muller's Algorithm*. Like the secant method, it requires one function evaluation per iteration and has rate of convergence $R = 1.86$. One additional benefit to the quadratic approximation is the possibility of finding complex roots with this method.

5 Maximising/Minimising a Function

A problem closely related to root-finding is finding the maximum or minimum of a function. Finding the global maximum or minimum of a function is, in general, a very complicated task. When we reduce this to a local problem, the situation simplifies greatly and we have a number of useful algorithms.

The most obvious to find the maximum or minimum of a function, $f(x)$, is to perform a Newton-Raphson routine on its derivative, $f'(x)$. i.e. We find the root in the derivative using

$$x_{i+1} = x_i - \frac{f'(x_i)}{f''(x_i)}. \tag{15}$$

A second derivative test will decide whether this root corresponds to a maximum or a minimum of $f(x)$.

Another approach to this problem, which is similar to bracketing bisection is the *Golden Ratio search*.

6 Golden Ratio Search

This is an iterative method for maximising/minimising a function, $f(x)$, over an interval $[a, b]$. We aasume that $f'(x) = 0$ only once over the interval $[a, b]$.

Without loss of generality, we will restrict ourselves to a discussion of the use of this method for finding the minimum x_m of $f(x)$.

We choose two interior points, c and d. These interior points should be constructed symmetrically so that $|c - a| = |d - b|$. This requires that

$$c = a + (1 - r)(b - a) \tag{16}$$
$$d = b - (1 - r)(b - a) \tag{17}$$

where $1/2 < r < 1$.

We can now use these four points to find the minimum of $f(x)$ thus;

If $f(c) < f(d)$ then $x_m \in [a, d]$. In this case, d becomes the endpoint of our new search segment $[a, d]$ and we find two new interior points.

Similarly, if $f(c) > f(d)$ then $x_m \in [c, b]$. In this case, c becomes the endpoint of our new search segment $[c, b]$ and we find two new interior points.

All that remains now is to find a value for r. We initially focus on $[a, b]$. Let us condiser the case where $f(c) < f(d)$. Thus, we restrict out attention to $[a, d]$. For each iteration, we want to perform only one functino evaluation. To do this, we must recycle both the interior points from the previous iteration. As we pointed out above, one of these interior points will become an exterior point of the next search interval (in this case d). The other interior point, c, will be maintained as an interior point for the next iteration. A new interior point, e, is placed between a and c.

Since the value of r must be the same for all iterations, we have, in this case

$$\frac{|d - a|}{|b - a|} = \frac{|c - a|}{|d - a|} \tag{18}$$

This restriction defines the value of r to be

$$r = \frac{\sqrt{5} - 1}{2} \tag{19}$$

This value is known as the *Golden Ratio*.

At each iteration, the width of the search interval decreases by a factor of 0.618. The error after n iterations will be

$$|b-a| \times 0.618^{n+1} \tag{20}$$

The Golden Section search converges at a rate $R = 1$.

The initial progression of this algorithm for the function $f(x) = x^2 - 2x + 4$, using $a = 0$ and $b = 3$, is shown in Fig.2. The relevant data are in the table below.

Initially we have

	x	$f(x)$
a	0	4
b	3	7
c	1.15	3.0225
d	1.85	3.5225

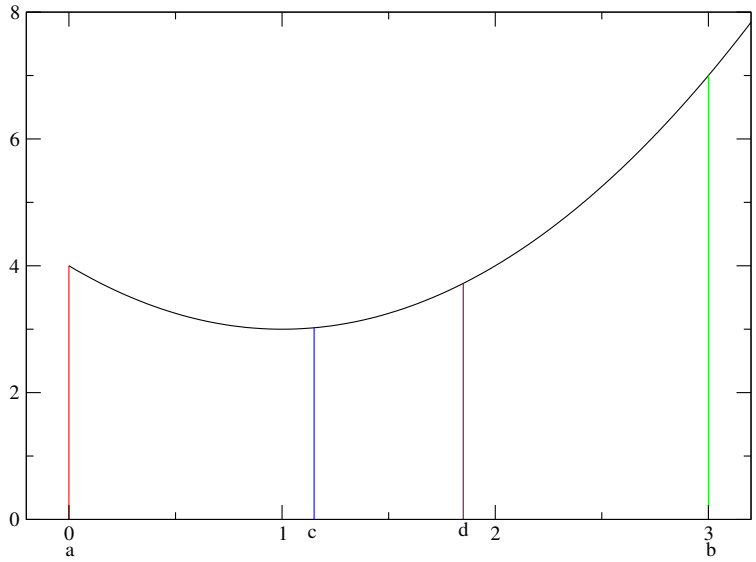
Since $f(d) > f(c)$, we restrict our attention to $[a, d]$, disregarding b. We create a new interior point, e, between a and c given by

$$e = a + (1 - r)(d - a). \tag{21}$$

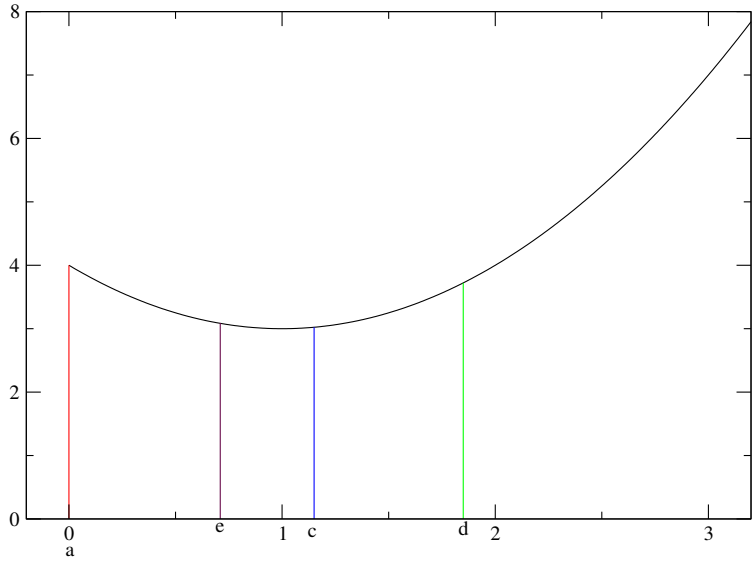
The data for this new section is given below.

	x	$f(x)$
a	0	4
d	1.85	3.5225
e	0.71	3.08
c	1.15	3.0225

In one iteration, we have reduced the width of our interval containing the minimum of $f(x)$ from 3 to $0.618 \times 3 = 1.85$.



(a) Initial set up using a, b, c and d.



(b) Disregard b and insert a new point, e, between a and c.

Figure 2: First two steps in a Golden Section Search