A nonlinear equation as defined here has the form

$$f(x) = 0 \tag{1}$$

where $f : \mathbb{R} \to \mathbb{R}$ is a continuous function. Throughout these notes, we will call a (relevant) solution $\tilde{x}$.

An iterative method is initiated with a starting guess x_0. An iterative method generates a sequence $x_0, x_1, x_2, \ldots$ with some algorithm.

For a given problem and initial guess, the iterative method is called **convergent** if $x_i \to X$ for $i \to \infty$ where X is a fixed value (not necessarily a solution). If we also have $x_i \to \tilde{x}$ for $i \to \infty$, we say that the method is **convergent to a solution**.

If a method converges to a solution for any problem and from any initial guess x_0, we say that the method is **globally convergent**. Only the methods using bracketing has this property

1.1 Important properties

The goal of any method is to find an approximation to a solution with a desired accuracy (distance to the solution). When choosing a solution, the following properties should be considered

- **Robustness:** Is there a guaranteed convergence to the solution. Some methods are robust for all problems (when the method is globally convergent to a solution). Some method are robust for specific types of problems.
- **Efficiency:** Simply computed as the number of $f(x)$-computations needed to converge to the solution with the desired accuracy.

Convergence, order and convergence constant

If a method converges to a solution $\tilde{x}$, we write the error after k steps as

$$\epsilon_k = x_k - \tilde{x}$$

If then

$$\frac{|\epsilon_{k+1}|}{|\epsilon_k|^\alpha} \to C$$

we say that the convergence has **order** α with **convergence constant** C.

A given method has the same order for all root finding problems (except in singular cases), whereas the convergence constant is problem dependent.

2.2 False Position (Regula Falsi)

False position is a bracketing variant of the Secant method inspired by Bisection. The algorithm is

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

If $f(x_{i+1})f(y_i) < 0$

$$y_{i+1} = y_i$$

Else

$$y_{i+1} = x_i$$

2.2.1 Convergence properties

An analysis (which we shall again skip) shows that if f is twice continuous differentiable and $f''(\tilde{x}) \neq 0$, we will have that from some k that

$$y_m = y_{m+1} = y_{m+2} = \dots$$

We then get that the Regula falsi method has

$$\begin{aligned} \alpha &= 1 \\ C &= 1 - f'(\tilde{x}) \frac{y_m - \tilde{x}}{f(y_m)} \end{aligned}$$

For some function, the False Position method generates the following output for the first 20 iterations:

i	xi	xi-x{i-1}
3.	1.25057	0.282152
4.	1.4553	0.204722
5.	1.59944	0.144141
6.	1.69738	0.0979444
7.	1.76192	0.0645371
8.	1.80347	0.0415478
9.	1.82978	0.0263173
10.	1.84627	0.0164908
11.	1.85654	0.0102615
12.	1.86289	0.00635714
13.	1.86682	0.00392742
14.	1.86924	0.00242216
15.	1.87073	0.00149222
16.	1.87165	0.000918711
17.	1.87222	0.000565388
18.	1.87257	0.000347861
19.	1.87278	0.000213992
20.	1.87291	0.000131628

What is the error on the approximation x20=1.87291 ?

Assume that we have used an iterative method that generates a sequence $x_0, x_1, \ldots, x_k$ and that we write the unknown exact solution as:

$$\tilde{x} \quad \text{meaning that} \quad f(\tilde{x}) = 0 \tag{1}$$

We write the error, ϵ_k, for iteration k:

$$\epsilon_k = x_k - \tilde{x}$$ Unknown

And we write the change at the k'th iteration d_k:

$$d_k = x_k - x_{k-1}$$ Computable from the output

$$x_{k+1} = x_k + d_{k+1}; \quad x_{k+2} = d_{k+2} + x_{k+1}; \ldots x_{k+n} = x_{k+n-1} + d_{k+n}$$

$$x_{k+n} = x_k + d_{k+1} + d_{k+2} \cdots + d_{k+n} = x_k + \sum_{i=1}^{n} d_{k+i}$$

$$\tilde{x} = x_k + \sum_{i=1}^{\infty} d_{k+i}$$

$$\epsilon_k = x_k - \tilde{x} = -\sum_{i=1}^{\infty} d_{k+i} = -\left[d_{k+1} + d_{k+2} + d_{k+3} + \ldots\right]$$

Unfortunately also unknown as the d_k values are only known up to the last computed iteration

$$\epsilon_k = x_k - \tilde{x} = -\sum_{i=1}^{\infty} d_{k+i} = -\left[d_{k+1} + d_{k+2} + d_{k+3} + ...\right]$$

i	xi	xi-x{i-1}	(xi-x{i-1})/(x{i-1}-x{i-2})^k
3.	1.25057	0.282152	0.723922
4.	1.4553	0.204722	0.725574
5.	1.59944	0.144141	0.704082
6.	1.69738	0.0979444	0.679504
7.	1.76192	0.0645371	0.658915
8.	1.80347	0.0415478	0.643783
9.	1.82978	0.0263173	0.633421
10.	1.84627	0.0164908	0.626616
11.	1.85654	0.0102615	0.622258
12.	1.86289	0.00635714	0.619511
13.	1.86682	0.00392742	0.617796
14.	1.86924	0.00242216	0.616731
15.	1.87073	0.00149222	0.616072
16.	1.87165	0.000918711	0.615665
17.	1.87222	0.000565388	0.615415
18.	1.87257	0.000347861	0.61526
19.	1.87278	0.000213992	0.615165
20.	1.87291	0.000131628	0.615107

First order methods with (False Position, Bisection).

If a method satisfies that

$$\frac{d_k}{d_{k-1}} \quad \rightarrow \quad C \quad \text{as} \quad k \rightarrow \infty \qquad \text{where } |C| < 1$$

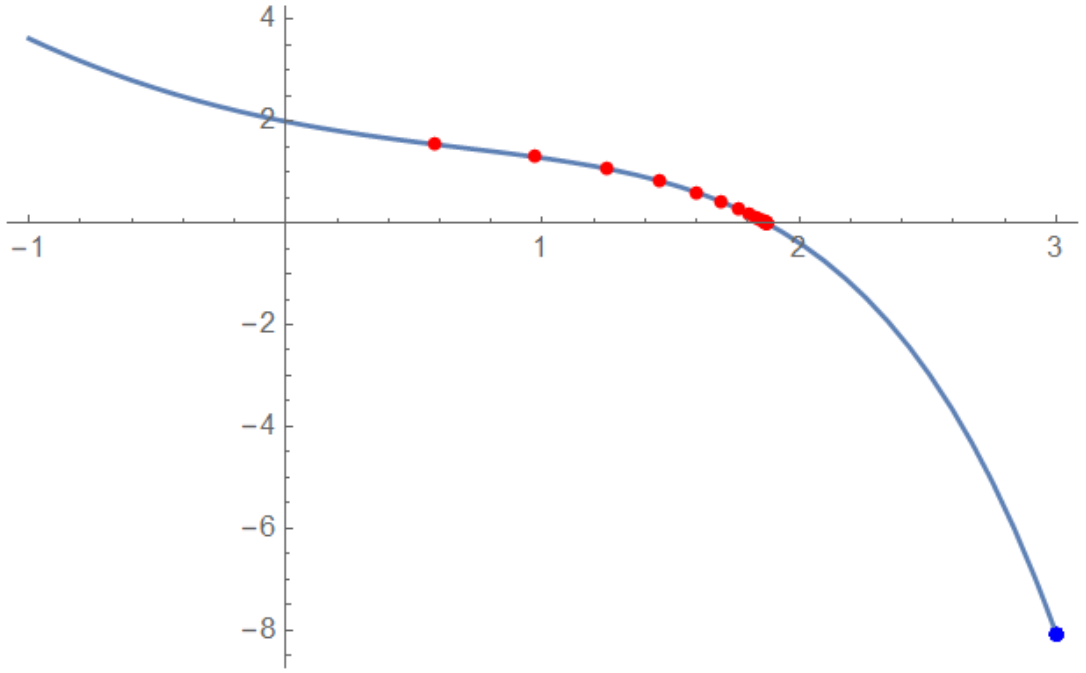
(False position).

$$\epsilon_k \simeq -d_k \left[C + C^2 + C^3 + ...\right] = \frac{-C}{1-C} d_k$$

Now we can do the error estimate for False Position…

f(x)=x^2-exp(x)+3

xExact= 1.8731225477130433



False Position generates the following output for the first 20 iterations:

i	xi	xi-x{i-1}	(xi-x{i-1})/(x{i-1}-x{i-2}
3.	1.25057	0.282152	0.723922
4.	1.4553	0.204722	0.725574
5.	1.59944	0.144141	0.704082
6.	1.69738	0.0979444	0.679504
7.	1.76192	0.0645371	0.658915
8.	1.80347	0.0415478	0.643783
9.	1.82978	0.0263173	0.633421
10.	1.84627	0.0164908	0.626616
11.	1.85654	0.0102615	0.622258
12.	1.86289	0.00635714	0.619511
13.	1.86682	0.00392742	0.617796
14.	1.86924	0.00242216	0.616731
15.	1.87073	0.00149222	0.616072
16.	1.87165	0.000918711	0.615665
17.	1.87222	0.000565388	0.615415
18.	1.87257	0.000347861	0.61526
19.	1.87278	0.000213992	0.615165
20.	1.87291	0.000131628	0.615107

$$\frac{d_k}{d_{k-1}} \quad \to \quad C \quad \text{as} \quad k \to \infty \qquad \text{where } |C| < 1$$

$$\epsilon_k \simeq -d_k \left[C + C^2 + C^3 + ...\right] = \frac{-C}{1-C} d_k$$

e20=(-0.615/(1-0.615))0.000131628

= -0.00021026

1.1 Newtons method

The most well known iterative method for root finding is Newtons method (sometimes called the Newton-Raphson method). To use Newtons method, the function $f(x)$ must be continuous differentiable. The algorithm is then

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

1.1.1 Convergence properties

An analysis (which we shall skip) shows that if f is twice continuous differentiable, Newtons method has

$$\begin{aligned} \alpha &= 2 \\ C &= \frac{1}{2}\frac{f''(\tilde{x})}{f'(\tilde{x})} \end{aligned}$$

$$\epsilon_k = x_k - \tilde{x} = -\sum_{i=1}^{\infty} d_{k+i} = -\left[d_{k+1} + d_{k+2} + d_{k+3} + ...\right]$$

Higher order methods (Newton, Secant, Ridders)

If we with the convergence order α have that

$$\frac{|d_{k+1}|}{|d_k|^{\alpha}} \leq C \quad \text{for all } k > k_0$$

Then it can be shown that

$$|\epsilon_k| \leq C|d_k|^{\alpha}$$

Now we can do the error estimate for Newton…

Newtons method example:

i	xi	xi-x{i-1}	|xi-x{i-1}|/|x{i-1}-x{i-2}|^2
3.	1.8959	-0.154309	1.09289
4.	1.87354	-0.0223633	0.939192
5.	1.87312	-0.000416959	0.833724
6.	1.87312	-1.41936×10^{-7}	0.816408
7.	1.87312	-1.64313×10^{-14}	0.815614

Newtons method generates the following output for the first iterations:

i	xi	xi-x{i-1}	|xi-x{i-1}|/|x{i-1}-x{i-2}|^2	|xi-xExact|
3.	1.8959	-0.154309	1.09289	0.0227804
4.	1.87354	-0.0223633	0.939192	0.000417101
5.	1.87312	-0.000416959	0.833724	1.41936×10^{-7}

$$|\epsilon_k| \leq C|d_k|^{\alpha}$$

e5<=0.85(0.000417)^2
=1.5x10^-7

Due to roundoff error. Can often make it difficult to estimate the convergence constant C for higher order methods.

2.3 Ridders method

A faster bracketing method is achieved by combining Bisection and False Position to Ridders method:

$$z_i = (x_i + y_i)/2$$

$$a = \frac{f(z_i)+\text{sign}[f(y_i)]\sqrt{f(z_i)^2-f(x_i)f(y_i)}}{f(y_i)} \text{ (not needed in the algorithm)}$$

$$x_{i+1} = z_i + (z_i - x_i)\frac{\text{sign}[f(x_i)-f(y_i)]f(z_i)}{\sqrt{f(z_i)^2-f(x_i)f(y_i)}}$$

If $f(x_{i+1})f(z_i) < 0$

$$y_{i+1} = z_i$$

Else if $f(x_{i+1})f(y_i) < 0$

$$y_{i+1} = y_i$$

Else

$$y_{i+1} = x_i$$

Ridders method generates the following output for the first iterations:

i	xi	xi-x{i-1}	(xi-x{i-1})/(x{i-1}-x{i-2})^3	xi-xExact
3.	1.8732	0.00865275	0.00426018	0.0000807271
4.	1.87312	-0.0000807272	-124.611	-1.76952×10^{-10}
5.	1.87312	1.76952×10^{-10}	-336.352	0.

$$|\epsilon_k| \leq C|d_k|^{\alpha}$$

e3<=0.00426(0.00865)^3
=2.8x10^-9 !!!!!!!!!!!
Far too optimistic. In general, it can be very hard to estimate errors for Ridders method because roundoff errors sets in before any convergence constant C has settled.

Same problem can occur with Newtons method and sometimes also for the Secant method.

The robustness and extremely fast convergence more than compensates for the lack of a good error estimate. To get a fair estimate, use a very conservative value for C, here for example C=1000 (even that is not 100 percent safe).

We then get

e3<=1000(0.00865)^3
=0.00065
e4<=1000(0.0000807)^3
=5.3x10^-10

Summary of error estimates

Method	Expected Order	Estimate of C	Estimate of $\lvert\epsilon_k\rvert$
Newton	2	$\dfrac{\lvert d_k\rvert}{\lvert d_{k-1}\rvert^2}$	$C\lvert d_k\rvert^2$
Secant	$\dfrac{1}{2}(1+\sqrt{5}) \simeq 1.62$	$\dfrac{\lvert d_k\rvert}{\lvert d_{k-1}\rvert^{1.62}}$	$C\lvert d_k\rvert^{1.62}$
Bisection	1	$\dfrac{1}{2}$	$\lvert d_k\rvert$
False Position	1	$\dfrac{d_k}{d_{k-1}}$	$\dfrac{-C}{1-C}d_k$
Ridders	3	$\dfrac{\lvert d_k\rvert}{\lvert d_{k-1}\rvert^3}$	$C\lvert d_k\rvert^3$

For the higher order methods (Secant, Newton, Ridders), it can be difficult to estimate the convergence constant as convergence is so fast. Hence, a supremum may be guessed as outlined above. For first order methods, it is usually easy to estimate C.

Consider the simple problem

f(x)=x-cos(x)=0

where there must be a solution X between 0 and Pi/2.

Implement Newton, Secant, Bisection, Regula Falsi and Ridder for this problem.

Each method produces a sequence x_0, x_1, x_2,… that is supposed to converge to the solution.

You will be asked to organize outputs of the method in a table where each line corresponds to one iteration. The table will contain the following info

i x_i x_i-x_{i-1} |x_i-x_(i-1)|/ |x_(i-1)-x_(i-2)|^k

where k is the convergence order of the method.

Implement a method that tests for ”convergence of the convergence constant” and estimates the error. Here, you need to decide in your program if convergence is achieved and pick your estimate of the convergence constant.

For False Position, the convergence constant should well defined. $\frac{d_k}{d_{k-1}} \rightarrow C$

For higher order methods, you need to pick your estimate of the supremum of the convergence constant. For Ridders, you may choose the bracket size as your (conservative) error estimate. $\frac{|d_{k+1}|}{|d_k|^\alpha} \leq C \quad \text{for all } k > k_0$

It is desirable to automate this procedure as 1D root finding is often called from bigger problems.