Equation solving

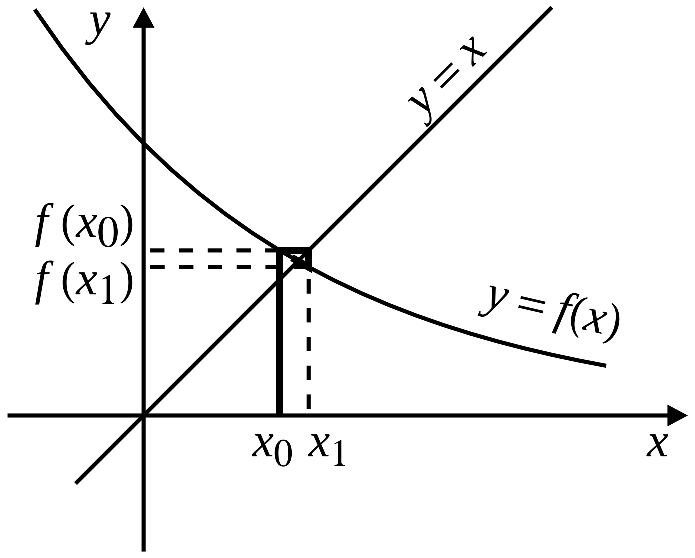
Even simple equations like $e^x = 5x$ cannot be solved analytically, or the expression of the solution can be complicated (e.g. Cardano's formula for third degree polynomial equations).

There are several methods for solving equations numerically. There is no single "best" method for all cases; the choice depends on the problem.

Direct iteration

Write the equation in the form $x = f(x)$. First we guess somehow an initial value x_0, which will then be substituted to the righthand side of the equation to get an improved solution $x_1 = f(x_0)$. This iteration is continued until the values remain the same within the required precision:

$$\begin{aligned} x_0 &= \dots \\ x_1 &= f(x_0), \\ x_2 &= f(x_1), \\ &\dots \\ x_{n+1} &= f(x_n), \\ &\dots \end{aligned}$$



In principle any of the instances of the unknown in the equation can be expressed in terms of the others; thus the equation can be written in the required form in many ways. For example, the equation $x^5 - x - 1 = 0$ can be expressed in the form $x = x^5 - 1$ or $x = (1 + x)^{0.2}$.

Using the previous form and the initial value 0.5 we'll get the following sequence of va-lues:

$$x_0 = 0.5,$$
$$x_1 = -1.853215,$$
$$x_2 = -22.85895,$$
$$x_3 = -6241392,$$

Thus the approximations do not converge.

But the other form will give the sequence

$$
\begin{aligned}
x_0 &= 0.5, \\
x_1 &= 1.158242, \\
x_2 &= 1.166326, \\
x_3 &= 1.167199, \\
x_4 &= 1.167293, \\
x_5 &= 1.167303, \\
x_6 &= 1.167304, \\
x_7 &= 1.167304
\end{aligned}
$$

In general, if the function appearing in the equation is a polynomial, the highest power of the unknown should be solved in terms of the rest. No similar rules can be given for other kinds of equations; the simplest way is just to try which choice will give a convergent sequence of approximations.

Interval halving

$$f(x) = x^5 - x - 1 = 0.$$

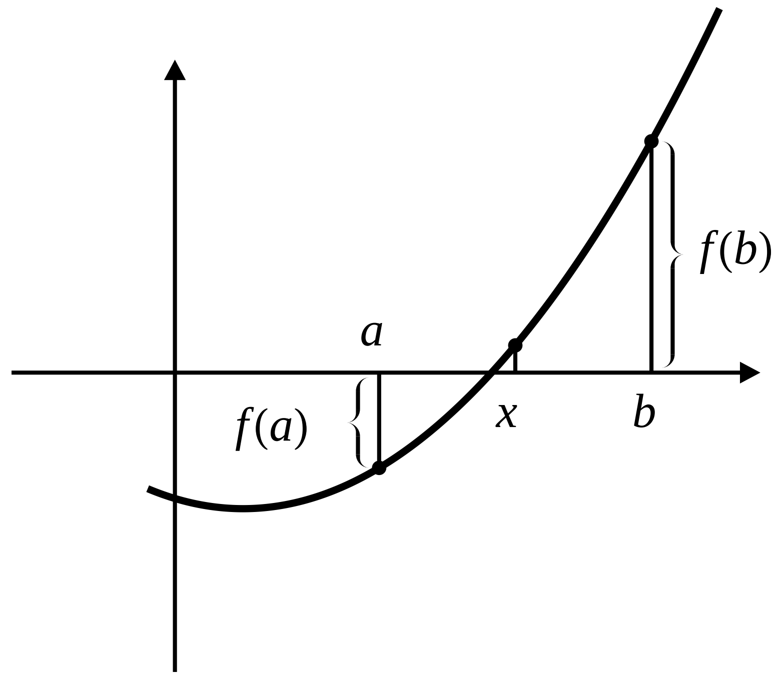
$f(1) = -1 < 0$ and $f(2) = 29 > 0$, hence the equation must have at least one solution in the range $1 < x < 2$.

Halve the interval, and at see in which half the solution must be:

$f(1.5) = 5.09 > 0$, thus the polynomial will change its sign in the range $1 < x < 1.5$.

$f(1.25) = 0.802 > 0$, thus the solution is in the range $1 < x < 1.25$.

Etc.



Due to halving the interval, each iteration gives one bit of extra information. One decimal digit is roughly equal to three binary digits. Thus to improve the solution by one significant figure about three iterations are needed.

Convergence is certain but slow.

OK, if the function evaluation is fast.

If the function is complicated (slow to calculate), this method may be too slow.

```
program halve
! solve f(x)=0 by interval halving
implicit none
real x
x = solve (1.0, 2.0)
write(*,*) x, f(x)

contains

real function f(x)
real, intent(in) :: x
  f = x**5 - x - 1
end function f

real function solve (xmin, xmax)
 real, intent(in) :: xmin, xmax
 real :: x1, y1, &     ! lower limit of interval
        x2, y2, &     ! upper limit
        x0, y0       ! midpoint

 x1=xmin; y1=f(x1)
 x2=xmax; y2=f(x2)
```

```
  x0=(x1+x2)/2 ;  y0=f(x0)

  do
    if (y1 * y0 < 0) then ! solution in (x1,x0)
      x2=x0
      y2=f(x2)
    else                  ! solution in (x0,x2)
      x1=x0
      y1=f(x1)
    end if
    x0=(x1+x2)/2
    y0=f(x0)
    if (abs(y0) < 0.0001) then ! accuracy achieved
      solve=x0
      exit
    end if
  end do
end function solve
end program halve
```

Regula falsi

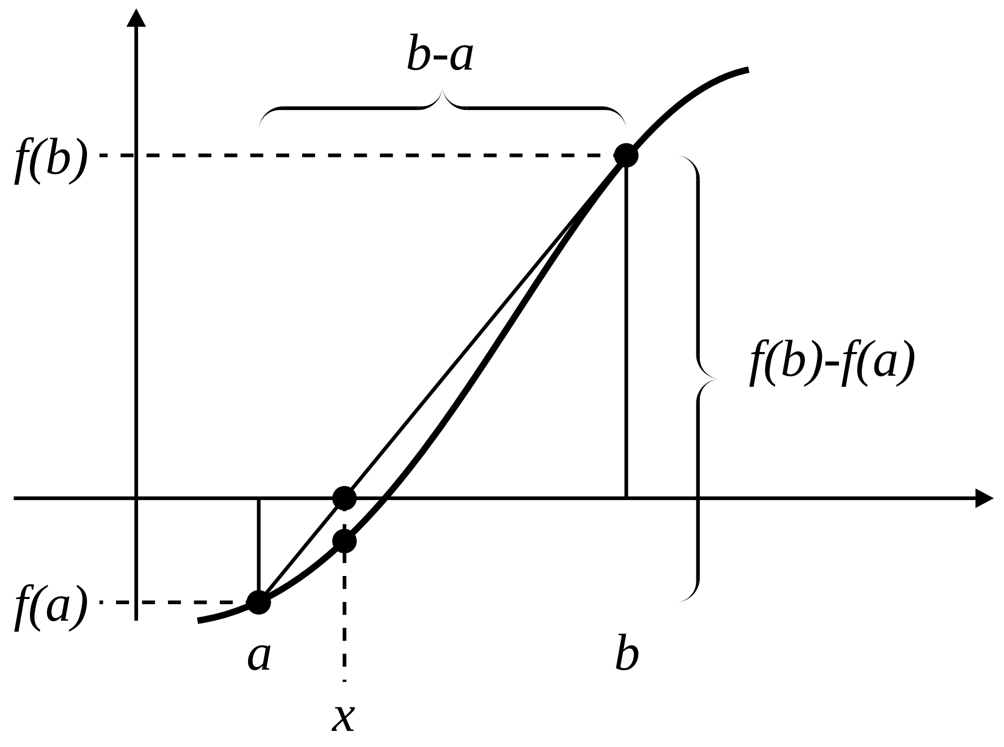
In the case of the equation $f(x) = x^5 - x - 1 = 0$ the absolute value of $f(1)$ is much smaller than $f(2)$. It seems natural that the solution has to be closer to 1 than 2.

If the interval is short the function can be approximated by a straight line. If the end-points of the interval are a and b, the equation of the line is

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

This intersects the x-axis at

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



Iteration can be continued by taking $[a, x]$ or $[x, b]$ as the new interval, depending on which interval the function will change sign.

In principle the iteration formula can be written as

$$x = \frac{af(b) - bf(a)}{f(b) - f(a)},$$

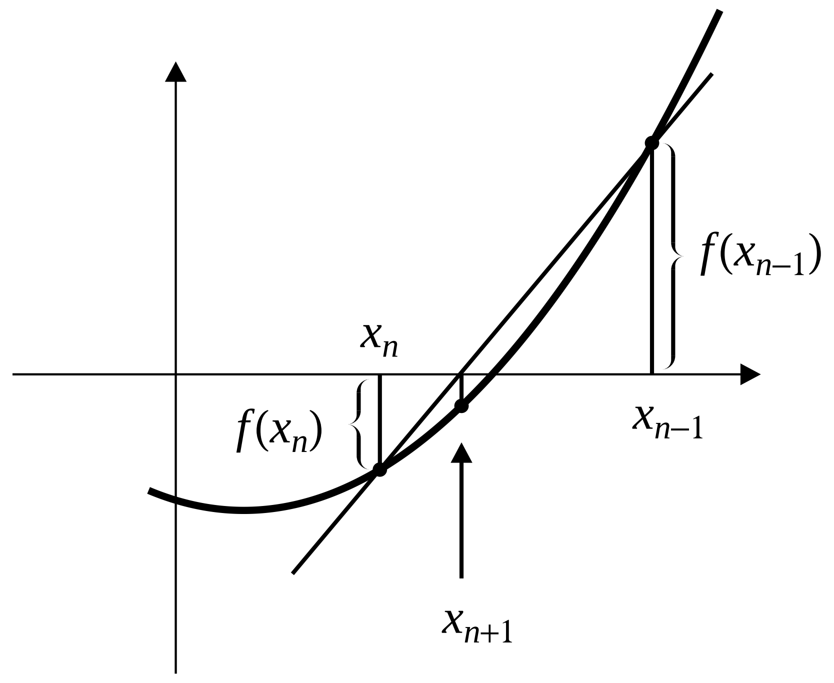
which has one floating point operation less. However, in the denominator we have a subtraction of nearly equal quantities, which is dangerous. Thus this form is not suitable for programs.

Secant method

A problem of the previous method is a slow convergence near the root.

If the last two approximations are x_n and x_{n-1}, they determine a straight line

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



The new zero point solved from this is

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

This method converges often more rapidly than the previous one.

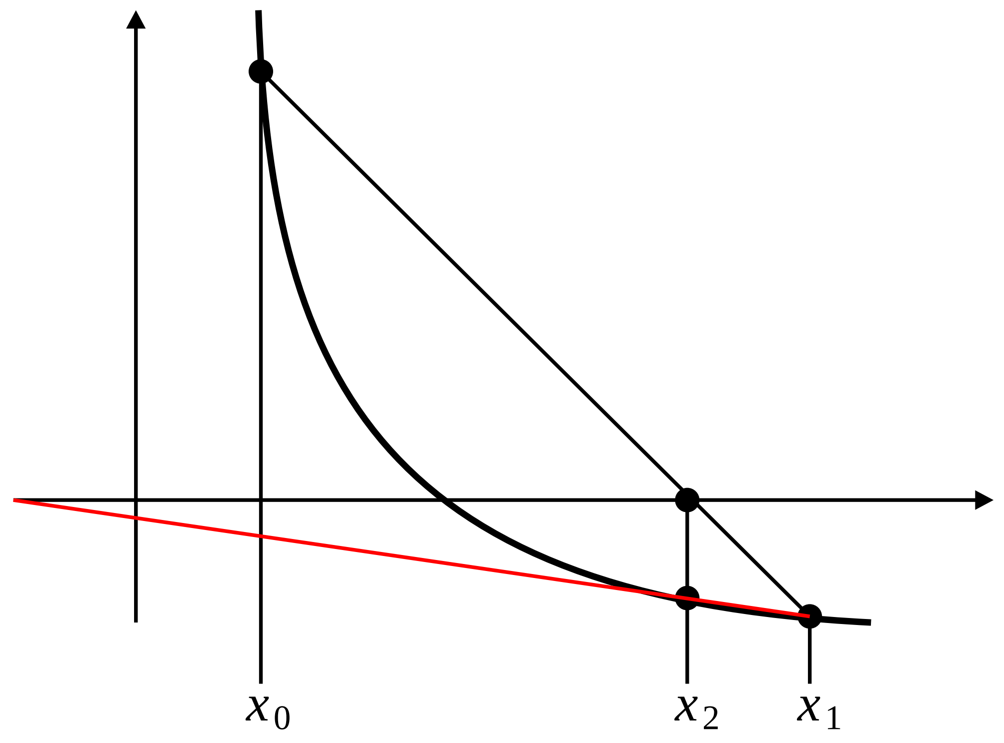
$$f(x) = \frac{1}{x^4} - 1 = 0.$$

Since $f(1/2) > 0$ and $f(2) < 0$, the solution is in the range $1/2 < x < 2$. Assume that the end points of this interval are the first two approximations.

$$\begin{aligned} x_0 =& 0.5000, \quad f(x_0) = 15, \\ x_1 =& 2.0000, \quad f(x_1) = -0.9375, \\ x_2 =& 1.9118, \quad f(x_2) = -0.9251, \\ x_3 =& -8.482. \end{aligned}$$

The sequence of successive approximations does not converge; thus the secant method does not find the solution at all.

Interval halving and regula falsi guarantee that the iterates remain all the time in a narrowing range. In the secant method there is no such range, and the sequence of iterates may even diverge.



Newton's method

Previously we had the slope

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

When the interval gets shorter this slope approaches the derivative of the function. If we approximate this expression using the derivative, the iteration step will be

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

This iteration is called either Newton's or Newton's and Raphson's method.

The derivative must be calculable. In practice, this means that the function has an analytic expression.

Newton's method suffers from the same problem as the secant method. If the derivative is small the method does not converge. The following theorem gives a criterion for the convergence of the method:

Let f be differentiable in the interval $[a, b]$, $f(a)f(b) < 0$, $f'(x) \neq 0$, and $f''(x)$ has the same size in the whole interval. Newton's method converges for an arbitrary initial value in $x_0 \in [a, b]$, if

$$\left|\frac{f(a)}{f'(a)}\right| < b - a, \quad \text{and} \quad \left|\frac{f(b)}{f'(b)}\right| < b - a.$$

Roots of a polynomial equation

All previously described methods finds one individual solution of an equation. Often that is sufficient, and possible other solutions can be found by using different initial values. Special cases are equations of the form $P_n(x) = 0$ where P_n is a polynomial of degree n. According to the fundamental theorem of algebra the equation has n solutions (real or complex), some of which may, however, be the same. For such equations there are methods that will automatically find all roots without requiring suitable initial values for each root.

The following method for finding the roots of a polynomial equation is known as the Q-D-method (Quotient–Difference). Derivation of the method is, however, too complicated to be presented here.

Assume we have the equation

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n = 0.$$

Calculate numbers q_i, $i = 1, ...n$ and e_i, $i = 0, ..., n$ in the following manner. Begin by calculating initial values

$$\begin{aligned} q_i &= 0, i = 1, ..., n-1 \\ q_n &= -a_{n-1}/a_n, \\ e_0 &= 0, \\ e_i &= a_{i-1}/a_i, i = 1, ..., n-1, \\ e_n &= 0. \end{aligned}$$

Then update the arrays q and e alternatingly:

$$\begin{aligned} q_i &= e_{i-1} - e_i + q_i, i = 1, ..., n \\ e_i &= \left(\frac{q_i}{q_{i+1}}\right) e_i, i = 0, ..., n. \end{aligned}$$

When the values of e approach zero, the values of q approach the zeros of the polynomial.

As an example, consider the equation $x^3 - 6x^2 + 11x - 6 = 0$, which has roots 1, 2 and 3.

```
program qd
! find all roots of a polynomial equation
 integer, parameter :: n=3
 real, dimension (0:n) ::  a = (/ -6, 11, -6, 1 /), &
                  e = 0, q = 0
 real, parameter :: limit = 1.0e-5
 integer :: i, kierros=0

 q(n) = -a(n-1)/a(n)
 do i=1,n-1
   e(i) = a(i-1)/a(i)
 end do
 write (6, '(8F8.4)') e, q
 ! iterate
 do
   do i=1,n
     q(i) = e(i-1)-e(i)+q(i)
   end do
   do i=0,n-1
     e(i) = q(i)/q(i+1) * e(i)
   end do
```

```
    kierros = kierros+1
    if (kierros > 100) then
      write(6,'("iteration does not converge")')
      exit
    end if
    if (maxval(abs(e)) < limit) exit
    write (6, '(8F8.4)') e, q
  end do
end program
```

The elements `e(0)` and `e(3)` are not actually needed, since they are always zeros. Omitting them would save only two words of memory but the program would be more complicated, since the first q-value `q(1)` should be handled in a different way. This is a simple example of a case in which a suitable choice of data structures the program becomes simpler.

Output of the example program:

```
e(0)   e(1)   e(2)   e(3)    q(1)   q(2)   q(3)
0.0000 -0.5455 -1.8333 0.0000   0.0000 0.0000 6.0000
0.0000 -0.2310 -0.5667 0.0000   0.5455 1.2879 4.1667
0.0000 -0.1105 -0.2556 0.0000   0.7765 1.6235 3.6000
0.0000 -0.0554 -0.1351 0.0000   0.8870 1.7686 3.3444
0.0000 -0.0283 -0.0778 0.0000   0.9424 1.8483 3.2093
0.0000 -0.0144 -0.0472 0.0000   0.9706 1.8979 3.1315
0.0000 -0.0074 -0.0295 0.0000   0.9851 1.9306 3.0843
0.0000 -0.0037 -0.0189 0.0000   0.9924 1.9528 3.0548
...

0.0000 0.0000 0.0000 0.0000   1.0000 1.9999 3.0001
0.0000 0.0000 0.0000 0.0000   1.0000 1.9999 3.0001
0.0000 0.0000 0.0000 0.0000   1.0000 1.9999 3.0001
0.0000 0.0000 0.0000 0.0000   1.0000 2.0000 3.0000
```

As we see, convergence can be quite slow. Thus the method may not be useful for finding exact values of the roots. Instead, when sufficiently distinct values have been found, they can be used as initial values for some other method.

If any of the coefficients of the polynomial is zero, the initialization will lead to division by zero. This can be avoided by changing the variable. For example, in the equation $x^3 - x + 1 = 0$ we cn substitute $x = y + 1$ to get the equation $y^3 + 3y2 + 2y + 1 = 0$. The actual roots are obtained by adding one to the solutions.

This example shows also another problem: the solution does not converge. The method will find one root, but the two other values will oscillate. This means that two of the roots are complex valued.

Sets of equations

There can be several unknowns and equations. Methods do not need essential modifications. The unknown can be taken as a vector $\mathbf{x}$, the components of which are given suitable initial values. Iteration will then give vectors that approach the solution.

In the case of several variables one has to determine how to update the solution vector:

1) Find all components of the new solution vector using only the values of the previous solution, and update the whole vector at the same time.

2) Update each component as soon as it is calculated.

Accuracy of the solution

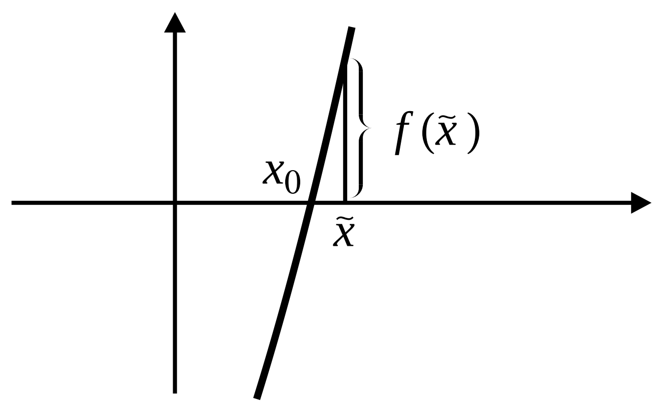
A natural requirement is that if $\tilde{x}$ is an approximate solution of the equation $f(x) = 0$ we must have

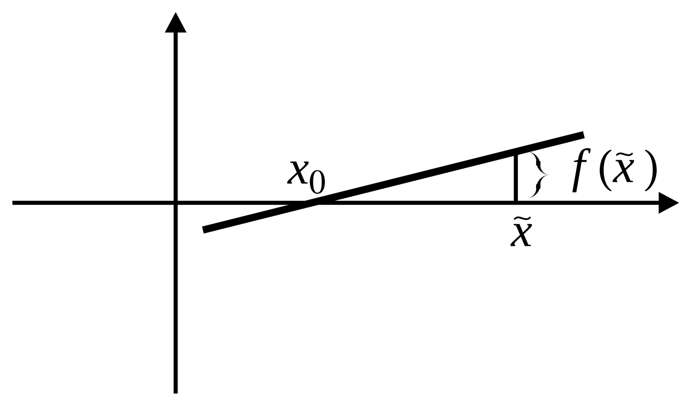
$$|f(\tilde{x})| < \epsilon,$$

where ϵ is some predefined accuracy.

If $f'(\tilde{x})$ is small $|f(\tilde{x})|$ can be small in a wide interval around the true solution.

The solution should be such that also $|x_0 - \tilde{x}|$ is small, where x_0 is the actual zero point of the function. Since x_0 is unknown, this error cannot be calculated. If the derivative is known, the error can be estimated.





Close to the zero point x_0 the function is approximately

$$f(x) \approx f'(x_0)(x - x_0),$$

whence

$$x - x_0 \approx \frac{f(x)}{f'(x_0)}.$$

When $x = \tilde{x}$,

$$\tilde{x} - x_0 \approx \frac{f(\tilde{x})}{f'(x_0)}.$$

If the derivative does not change very rapidly around the solution, we have approximately

$$\tilde{x} - x_0 \approx \frac{f(\tilde{x})}{f'(\tilde{x})}.$$

Thus the approximate solution is close to the true solution if

$$\left| \frac{f(\tilde{x})}{f'(\tilde{x})} \right| < \epsilon.$$

In addition, it is naturally required that the value of the function itself, $f(\tilde{x})$, is small.