3. Solution of Nonlinear Equations

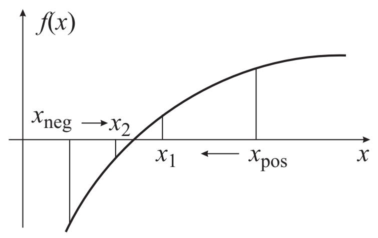
NUMERICAL ANALYSIS. Prof. Y. Nishidate (323-B, nisidate@u-aizu.ac.jp)
http://web-int.u-aizu.ac.jp/∼nisidate/na/

The zeroes of a function are the values of the function's variable for which the value of the function is zero. Mathematically, given the function $f(x)$, z is a zero when $f(z) = 0$. The problem of finding the values at which a function takes a maximum or minimum value is called searching for the extremes of a function. This problem can be transformed into a zero-finding problem if the derivative of the function can be easily computed. The extremes are the zeroes of the function's derivative.

The techniques for the solution of nonlinear equations are all *iterative* in nature. This means that we begin our solution process with a guess and then refine this guess according to some specific rules in an iteration loop. Iterations are terminated when a *convergence* is reached that is the error of the solution is smaller then the specified error tolerance.

Bisection Method

Let assume that one knows two values of x for which the function takes values of opposite sign. Let us call x_{pos} the value such that $f(x_{pos}) > 0$ and x_{neg} the value such that $f(x_{neg}) < 0$. If the function is continuous between x_{pos} and x_{neg}, at least one zero of the function exists in the interval $[x_{pos}, x_{neg}]$. This case is illustrated below.



If the function f is not continuous over the interval where the sign of the function changes, then the presence of a zero cannot be guaranteed. The continuity requirement is essential for the application of the bisection algorithm.

The values x_{pos} and x_{neg} are the initial values of the bisection algorithm. The algorithm goes as follows:

```
do
    Compute $x = (x_{pos} + x_{neg})/2$
    if $f(x) > 0$ then $x_{pos} = x$
    else $x_{neg} = x$
while $|x_{pos} - x_{neg}| > \varepsilon$
```

Here ε is the error tolerance of the solution. The error at each iteration is $|x_{pos} - x_{neg}|$ since the zero of a function is always inside the interval defined by x_{pos} and x_{neg}.

For a given pair of initial values, x_{pos} and x_{neg}, the number of iterations required to obtain a root with an error less than ε is given by

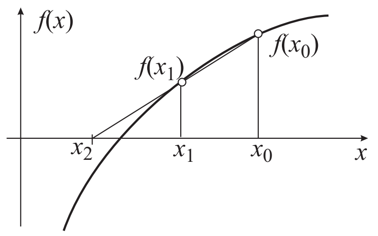
$$n = \left\lceil \log_2 \frac{|x_{pos} - x_{neg}|}{\varepsilon} \right\rceil$$

For example, if the distance between the two initial values is 1, the number of iterations required to attain a precision of 10^{-8} is 30. It shows that the bisection algorithm is rather slow.

Knowledge of the initial values, x_{pos} and x_{neg}, is essential for starting the algorithm. Methods to define them must be supplied. A possible procedure is to sample the function randomly over a given range to find each initial value.

Secant Method

Almost every function can be approximated by a straight line over a small interval. We begin from a value x_0 that is not far from the root z. Suppose we assume that $f(x)$ is linear in the vicinity of the root z. Now we choose another point, x_1, which is near to x_0 and also near to z (which we don't know yet), and we draw a straight line through the two points.



If $f(x)$ was truly linear, the straight line would intersect the x-axis at the root. But $f(x)$ is not exactly linear because we solve a nonlinear equation. That means that the intersection of the line with the x-axis is not at $x = z$ but that it should be close to it. From the obvious similar triangles we can write

$$\frac{x_0 - x_2}{f(x_0)} = \frac{x_0 - x_1}{f(x_0) - f(x_1)}$$

Solving for x_2 we get:

$$x_2 = x_0 - f(x_0)\frac{x_0 - x_1}{f(x_0) - f(x_1)}$$

Since $f(x)$ is not exactly linear, x_2 is not equal to z, but it should be closer than either of the two points we began with. We can continue to get better estimates of the root if we do this repeatedly, always choosing the two x -values nearest to z for drawing the straight line. Since each newly computed value should be nearer to the root, we can do this easily after the second iteration has been computed, by always using the last two computed points. But after the first iteration there aren't "two last computed points." So we make sure to start with x_1 closer to the root than x_0 by testing $f(x_0)$ and $f(x_1)$ and swapping if the first function value is smaller. The net effect of this rule is to set $x_0 = x_1$and $x_1 = x_2$ after each iteration.

The technique we have described is known as the secant method because the line through two points on the curve is called the secant line. Task is to determine a root of $f(x) = 0$, given two values, x_0 and x_1, that are near the root. Here is pseudocode for the secant method algorithm:

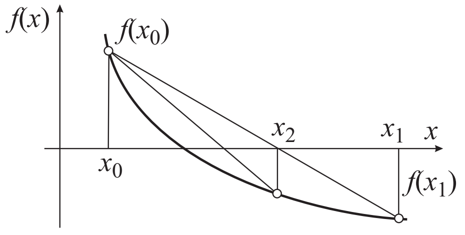
```
if $|f(x_0)| < |f(x_1)|$ then swap $x_0$ with $x_1$
do
    $x_2 = x_0 - f(x_0)(x_0 - x_1)/(f(x_0) - f(x_1))$
    $x_0 = x_1$
    $x_1 = x_2$
while $|x_0 - x_1| > \varepsilon$
```

Here ε is the error tolerance of the solution.

In pathological cases the secant method has no convergence. A way to avoid pathological cases is to ensure that the root is bracketed between the two starting points and remain between the successive pairs. This can be done with *linear interpolation* or *false position* method.

Linear Interpolation (False Position)

The technique known as the *false position method* is similar to bisection method except the next value of the unknown root is taken at the intersection of a line between the pair of x-values and the x-axis rather than midpoint. Doing so gives faster convergence than does bisection method. Here is the algorithm of the false position method.

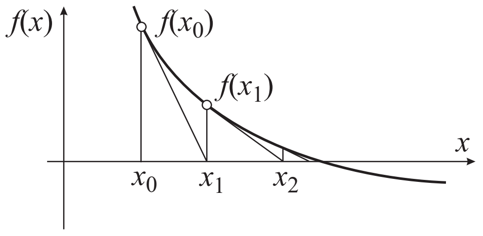


To determine a root of $f(x) = 0$, given two values, x_0 and x_1, that bracket the root, that is, $f(x_0)$ and $f(x_1)$ are of opposite sign:

```
$x = x_0$
if $|f(x_0)| < |f(x_1)|$ then swap $x_0$ with $x_1$
do
    $x_n = x$
    $x = x_0 - f(x_0)(x_0 - x_1)/(f(x_0) - f(x_1))$
    if $f(x)$ is of opposite sign to $f(x_0)$
    then $x_1 = x$
    else $x_0 = x$
while $|x - x_n| > \varepsilon$
```

Newton's Method

One of the most widely used methods of solving nonlinear equations is *Newton's method*. Like the previous methods, Newton's algorithm is also based on a linear approximation of the function, but does so using a tangent to the curve.



The main relation of Newton's method can be derived from the first-order Taylor expansion of f about the iterate x_n :

$$f(x) = f(x_n) + f'(x_n)(x - x_n)$$

where $f'(x)$ is the first derivative of $f(x)$. Setting this approximation to zero gives the next iterate x_{n+1}as the solution of

$$f(x_n) + f'(x_n)(x - x_n) = 0$$

Thus the algorithm of Newton's method can be expressed as follows:

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

Starting from a single initial estimate x_0 that is not too far from the root, we move along the tangent to its intersection with the x-axis, and take that as the next approximation. The iteration procedure is continued until the successive x-values are sufficiently close (or the value of the function is sufficiently close to zero).

To determine a root of $f(x) = 0$, given a value x_0 reasonably close to the root, the following Newton's algorithm is used:

```
compute $f(x_0)$ and $f'(x_0)$
$x_1 = x_0$
if $f(x_0) \neq 0$ and $f'(x_0) \neq 0$
    do
        $x_0 = x_1$
        $x_1 = x_0 - f(x_0)/f'(x_0)$
    while $|x_0 - x_1| > \varepsilon$
```

Newton's algorithm is widely used because in the neighborhood of a root, it is more rapidly convergent than any of the considered here methods. It is possible to show that Newton's method has quadratic rate of convergence which means the error of each step approaches a constant K times the square of the error of the previous step.

Newton's Method for Systems of Nonlinear Equations

In many cases of practical computational modeling, systems of nonlinear equations arise. Assume that $f_1,\ f_2,\ ...\ f_m$ are functions of the independent variables $x_1,\ x_2,\ ...\ x_m$ and we seek a solution for the system of nonlinear equations:

$$\begin{array}{l} f_1(x_1,\ x_2,\ ...,\ x_m) = 0 \\ f_2(x_1,\ x_2,\ ...,\ x_m) = 0 \\ ... \\ f_m(x_1,\ x_2,\ ...,\ x_m) = 0 \end{array}$$

Let us introduce the following vector notation for unknowns and functions:

$$\mathbf{x} = \left\{ \begin{array}{c} x_1 \\ x_2 \\ ... \\ x_m \end{array} \right\}, \quad \mathbf{f} = \left\{ \begin{array}{c} f_1 \\ f_2 \\ ... \\ f_m \end{array} \right\}$$

In our notation, vector means column matrix. Using vector notation, we can rewrite the system of nonlinear equations as follows:

$$\mathbf{f}(\mathbf{x}) = \mathbf{0}$$

First-order Tailor expansion around the iterate $\mathbf{x}^{(n)}$ can be written in the form:

$$\begin{array}{l} f_1(\mathbf{x}) = f_1(\mathbf{x}^{(n)}) + \sum\limits_{i=1}^{m} \frac{\partial f_1(\mathbf{x}^{(n)})}{\partial x_i}(x_i - x_i^{(n)}) \\ ... \\ f_m(\mathbf{x}) = f_m(\mathbf{x}^{(n)}) + \sum\limits_{i=1}^{m} \frac{\partial f_m(\mathbf{x}^{(n)})}{\partial x_i}(x_i - x_i^{(n)}) \end{array}$$

This expansion can be rewritten using matrix-vector notation:

$$\mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{x}^{(n)}) + \mathbf{J}(\mathbf{x}^{(n)})(\mathbf{x} - \mathbf{x}^{(n)})$$

Here $\mathbf{J}(\mathbf{x})$ is the Jacobian matrix:

$$\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \left[\begin{array}{ccc} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & ... & \frac{\partial f_1(\mathbf{x})}{\partial x_m} \\ ... & ... & ... \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & ... & \frac{\partial f_m(\mathbf{x})}{\partial x_m} \end{array} \right]$$

By setting to zero $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ in the above relation the iteration procedure of the Newton method can be formulated for systems of nonlinear equations:

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} - (\mathbf{J}(\mathbf{x}^{(n)}))^{-1}\mathbf{f}(\mathbf{x}^{(n)})$$

Here $(\mathbf{J}(\mathbf{x}^{(n)}))^{-1}$ is the inverse of the Jacobian matrix. Since the matrix inverse is computationally expensive operation then instead of matrix inversion with subsequent multiplication, the solution of the linear equation system is performed:

$$\mathbf{J}(\mathbf{x}^{(n)})\Delta\mathbf{x}^{(n)} = -\mathbf{f}(\mathbf{x}^{(n)})$$

where $\Delta\mathbf{x}^{(n)} = \mathbf{x}^{(n+1)} - \mathbf{x}^{(n)}$.

Choice of the Method

The Newton's algorithm is faster than other methods for most functions. However Newton's algorithm will fail if it encounters a value for which the derivative of the function is very small. In this case the algorithm jumps far away from the solution.

On the other hand, the bisection method is very reliable and will always converge over an interval where the function has no singularity. Thus the combination of Newton's and bisection methods can be fast and reliable.