Solution of nonlinear algebraic equations

Consider the following problem.

Find x such that

$$f(x) = 0$$

for a given function f. (Nonlinear means that f is not simply of the form $ax + b$).

We will examine various methods for finding the solution.

Method 1. The bisection method

This method is based on the intermediate value theorem (see **theorems.pdf**):

Suppose that a continuous function f defined on an interval $[a, b]$ is such that $f(a)$ and $f(b)$ have opposite signs, i.e. $f(a)f(b) < 0$. Then there exists a number p with $a < p < b$ for which $f(p) = 0$.

For simplicity we assume there is only one root in $[a, b]$.
The algorithm is as follows:

Bisection method algorithm

Set $a_1 = a;\ b_1 = b;\ p_1 = (a_1 + b_1)/2$.
If $f(p_1) = 0$ then $p = p_1$ and we are finished.
If $f(a_1)f(p_1) > 0$ then $p \in (p_1, b_1)$ and we set $a_2 = p_1, \quad b_2 = b_1$.
If $f(a_1)f(p_1) < 0$ then $p \in (a_1, p_1)$ and we set $a_2 = a_1, \quad b_2 = p_1$.
We now repeat the algorithm with a_1 replaced by a_2 and b_1 replaced by b_2.
We carry on until sufficient accuracy is obtained.

The last statement can be interpreted in different ways:
Suppose we have generated a sequence of iterates $p_1, p_2, p_{3,\ldots}$

Do we stop when:
(i) $|p_n - p_{n-1}| < \varepsilon$ (absolute error)
or (ii) $|f(p_n)| < \varepsilon$
or (iii) $|p_n - p_{n-1}| \,/\, |p_n| < \varepsilon$ (relative error)?

The choice of stopping criterion can often be very important.

Let's see how this algorithm can be programmed in Matlab (**bisection.m**) and see how we can compute the root to a polynomial using this method.

Clearly the bisection method is slow to converge (although it will always get there eventually!). Also, a good intermediate approximation may be discarded. To see this consider the solution of

$$\cos\left[\pi(x - 0.01)\right] = 0 \quad \text{over the range } 0 < x < 1.$$

We will illustrate this example in Matlab (**bisection.m**).

Can we find a faster method?

Fixed point iteration

We write $f(x) = x - g(x)$ and solve

$$x = g(x).$$

A solution of this equation is said to be a **fixed point** of g.
Before proceeding we state two theorems in connection with this method.

Theorem 1

Let g be continuous on $[a, b]$ and let $g(x) \in [a, b]$ for all $x \in [a, b]$. Then g has a fixed point in $[a, b]$. Suppose further that $|g'(x)| \le k < 1$ for all $x \in (a, b)$. Then g has a unique fixed point p in $[a, b]$.

Proof

(i) Existence

If $g(a) = a$ or $g(b) = b$ the existence of a fixed point is clear. Suppose not.
Since $g(x) \in [a, b]$ for all $x \in [a, b]$ then $g(a) > a$ and $g(b) < b$.
Define $h(x) = g(x) - x$. Then h is continuous on $[a, b]$, $h(a) > 0,\ h(b) < 0$.
Thus by the intermediate value theorem there exists a $p \in (a, b)$ such that $h(p) = 0$.
p is therefore a fixed point of g.

(ii) Uniqueness

Suppose we have two fixed points p and q in $[a, b]$ with $p \neq q$.
Then $|p - q| = |g(p) - g(q)| = |p - q|\,|g'(\tau)|$ by the mean value theorem with $\tau \in (p, q)$.
Since $|g'(\tau)| < 1$ we have $|p - q| < |p - q|$, which is a contradiction. Hence we have uniqueness.

To find the fixed point of $g(x)$ we choose an initial approximation p_0 and define a sequence p_n by

$$p_n = g(p_{n-1}), \quad n = 1, 2, 3, \ldots$$

This procedure is known as **fixed point or functional iteration**. Let's see fixed point iteration in action (**fixedpoint.m**).

Theorem 2

Let g be continuous in $[a, b]$ and suppose $g(x) \in [a, b]$ for all $x \in [a, b]$. Suppose further that $|g'(x)| \le k < 1$ for all $x \in (a, b)$.
Then if p_0 is any number in $[a, b]$ the sequence defined by

$$p_n = g(p_{n-1}), \quad n \ge 1$$

converges to the unique fixed point p in $[a, b]$.

Proof

Suppose that $a \le p_{n-1} \le b$. Then we have $a \le g(p_{n-1}) \le b$ and hence $a \le p_n \le b$. Since $a \le p_0 \le b$ it follows by induction that all successive iterates p_n remain in $[a, b]$.
Now suppose the exact solution is $p = \alpha$, i.e. $g(\alpha) = \alpha$. Then

$$\alpha - p_{n+1} = g(\alpha) - g(p_n) = (\alpha - p_n)g'(c_n),$$

for some $c_n \in (\alpha, p_n)$ using the Mean-Value theorem (see **theorems.pdf**). Since $|g'(c_n)| \le k$ it follows that

$$|\alpha - p_{n+1}| \le k\ |\alpha - p_n|$$

and hence

$$|\alpha - p_n| \le k^n\,|\alpha - p_0|\,.$$

The right hand side tends to zero as $n \to \infty$ (since $k < 1$) and so we have $p_n \to \alpha$ as $n \to \infty$, as required.

How do we choose $g(x)$? For some choices of g the scheme may not converge! One way of choosing g is via **Newton's** (or **Newton-Raphson**) method.
Newton's method is derived as follows:

Newton's Method

Suppose that the function f is twice continuously differentiable on $[a, b]$. We wish to find p such that $f(p) = 0$. Let x_0 be an approximation to p such that

$$f(x_0) \neq 0 \quad \text{and } |x_0 - p| \quad \text{is `small'}.$$

A Taylor series expansion about p gives

$$0 = f(p) = f(x_0) + (p - x_0)f'(x_0) + \frac{(p - x_0)^2}{2} f''(\tau),$$

where $\tau \in (p, x_0)$. Here we have used the Lagrange form of the remainder for Taylor series (see **theorems.pdf**).
Newton's method arises by assuming that if $|p - x_0|$ is small then $(p - x_0)^2 f''(\tau)/2$ can be neglected. So we are then left with

$$0 = f(p) \simeq f(x_0) + (p - x_0)f'(x_0).$$

Solving this equation for p we have

$$p \simeq x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Applying this result successively gives the Newton-Raphson method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}. \quad \boxed{\textbf{NEWTON'S METHOD}}$$

Note that this is of the form

$$x_{n+1} = g(x_n), \quad \text{with } g(x) = x - \frac{f(x)}{f'(x)}.$$

Geometrical interpretation:

At the current value x_n find the tangent to the curve.
Extend this until it cuts the x-axis - this is the value x_{n+1}.
Continue procedure.

Advantages of Newton's method

Can converge very rapidly.
Works also for $f(z) = 0$ with z complex.

Disadvantages

May not converge.
Evaluation of $f'(x)$ may be expensive.
Can be slow if $f(x)$ has a multiple root, i.e $f(p) = f'(p) = 0$.
If roots are complex numbers then we need a complex initial guess.

Order of convergence of Newton's method

It can be shown that if $|p - x_n|$ is sufficiently small then $|p - x_{n+1}| = \lambda \, |p - x_n|^2$, i.e. Newton's method is **quadratically convergent**.

Secant method

A method which does not require the evaluation of the derivative $f'(x)$ is the **secant method**.
In this we make the approximation

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

Substituting into Newton's method we have

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}. \quad \boxed{\textbf{SECANT METHOD}}$$

Note that this method needs two initial approximations x_0 and x_1. It requires less work than Newton since we do not need to compute $f'(x)$.

Geometrical interpretation

Fit a straight line through the last two values $(x_n, f(x_n)),\ (x_{n-1}, f(x_{n-1}))$.
Then x_{n+1} is where this line crosses the x-axis.

Rate of convergence of secant method

This is difficult to analyze but it can be shown that if $|p - x_n|$ is sufficiently small then

$$|p - x_{n+1}| = \lambda \, |p - x_n|^{\alpha}, \quad \text{where } \alpha = \frac{1}{2}(1 + \sqrt{5}) \simeq 1.618,$$

whereas $\alpha = 2$ for Newton. So once we are close to the root the secant method converges more slowly than Newton, but faster than the bisection method.

Generalization to systems of equations

Suppose we wish to solve the simultaneous equations

$$f(x,y)=0,\;\; g(x,y)=0$$

for the values x and y, where f, g are known functions.
First we write this in vector form by introducing

$$q=\begin{pmatrix} x \\ y \end{pmatrix},\;\; F=\begin{pmatrix} f \\ g \end{pmatrix}$$

so that we have to solve

$$F(q)=0.$$

It can be shown that the generalization of Newton's method is then

$$\left(\frac{\partial F}{\partial q}\right)_{q=q_n}(q_{n+1}-q_n)=-F(q_n).$$

MULTI-DIMENSIONAL NEWTON METHOD

Here $\partial F/\partial q$ is a matrix (the Jacobian) consisting of partial derivatives.

$$\frac{\partial F}{\partial q}=\begin{pmatrix} \partial f/\partial x & \partial f/\partial y \\ \partial g/\partial x & \partial g/\partial y \end{pmatrix}.$$

Example of 2D Newton iteration

Consider the system

$$\begin{aligned} x^2-y^2-2\cos x &= 0, \\ xy+\sin x-y^3 &= 0. \end{aligned}$$

Applying this method we have

$$\begin{pmatrix} 2x_n+2\sin x_n & -2y_n \\ y_n+\cos x_n & x_n-3y_n^2 \end{pmatrix}\begin{pmatrix} x_{n+1}-x_n \\ y_{n+1}-y_n \end{pmatrix}=\begin{pmatrix} -x_n^2+y_n^2+2\cos x_n \\ -x_ny_n-\sin x_n+y_n^3 \end{pmatrix},$$

at each step of the iteration. We need initial guesses for x and y to start this off.
Let's see how we could program this in Matlab (**newton2d.m**).

Finding the zeros of a polynomial

It is well-known that a polynomial of degree n has n roots (counted to multiplicity). However when $n > 4$ there is no exact formula for the roots. So we need to find them numerically. One method is known as deflation.

The method of deflation

Suppose we have a polynomial $p_n(x)$ of degree n. We find a zero of this polynomial using Newton's method (say). Suppose this root is α. We can then divide out the root:

$$p_{n-1}(x) = p_n(x)/(x-\alpha),$$

so that we now have a polynomial of degree $n-1$. We now apply our root finding algorithm to the new polynomial. By repeating this method we can find all n roots.

Disadvantage of deflation

Usually we will only be finding an approximation to each root so that the reduced polynomial p_{n-1} is only an approximation to the actual polynomial. Suppose at the first stage we compute the approximate root $\overline{\alpha}$, while the true root is α. Then the perturbed polynomial is

$$\overline{p}_{n-1}(x) = p_n(x)/(x-\overline{\alpha}).$$

The crucial question to ask is the following:

Given a polynomial $p_n(x)$ and a small perturbation of this, $\overline{p}_n(x)$, can the zeros change by a large amount?
The answer is *yes*, as may be illustrated by the following example.
Consider the polynomial

$$\begin{aligned} p_{20}(x) &= (x-1)(x-2)(x-3)\cdots(x-20) \\ &= x^{20} - 210\,x^{19} + 20615\,x^{18} + \cdots \end{aligned}$$

Suppose we change the coefficient 210 to $210 - 10^{-7}$ and leave all other coefficients unchanged. Let's see in Matlab what happens to the roots (see **polynomial.m)**.

Matlab shows us that deflation is sometimes not an accurate process.