Section 2 – Roots of Equations

In this section, we will look at finding the roots of functions. The basic root-finding problem involves many concepts and techniques that will be useful in more advanced topics.

Algebraic and Transcendental Functions

A function of the form $y = f(x)$ is algebraic if it can be expressed in the form:

$$f_n y^n + f_{n-1} y^{n-1} + f_{n-2} y^{n-2} + \ldots + f_1 y + f_0 = 0$$

where f_i is an *i*th-order polynomial in *x*. Polynomials are a simple class of algebraic functions that are represented by

$$f_n(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n$$

Where *n* is the order of the polynomial and the a_i are constants. For example,

$$f_2(x) = 1 - 2.37x + 7.5x^2$$
$$f_6(x) = 5x^2 - x^3 + 7x^6$$

A transcendental function is one that is not algebraic. These types of functions include trigonometric, logarithmic, exponential or other functions. Examples include

$$f(x) = \ln x^2 - 1$$
$$f(x) = e^{-0.2x} \sin(3x - 0.5)$$

There are two distinct areas when it comes to finding the root of functions:

1. Determination of the real roots of algebraic and transcendental functions, and usually only a single root, given its approximate location
2. Determination of all of the real and complex roots of polynomials

2.1 Graphical Methods

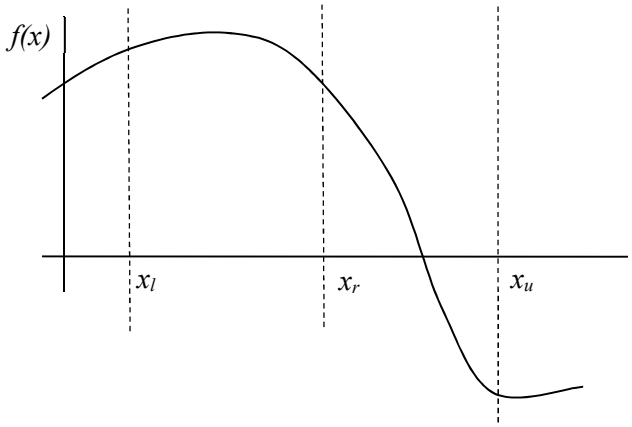
Graphical methods are straightforward – simply graph the function *f(x)* and see where it crosses the *x*-axis. This method will immediately yield a rough approximation of the value of the root, which can be refined through finer and more detailed graphs. It is not necessarily precise, but it is very useful in order to determine a starting point for more sophisticated methods.

2.2 Closed Methods

The following methods work on "closed" or bounded domains, defined by upper and lower values that bracket the root of interest.

2.2.1 Bisection Method

If $f(x)$ is real and continuous in the interval from x_l to x_u, and $f(u_l)$ and $f(x_u)$ have opposite signs, then there must be at least one real root between x_l and x_u.



The *bisection method* (or *binary chopping*, *interval halving* or *Bolzano's Method*) divides the interval between the upper and lower bound in half to find the next approximate root x_r,

$$x_r = \frac{x_l + x_u}{2}$$

which replaces the bound of the interval, either x_l or x_u, whose function value has the same sign as $f(x_r)$. The method proceeds until the termination criterion is met

$$\varepsilon_a = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right|$$

Pseudocode – Bisection method

```
FUNCTION Bisection(xl, xu, xr, ea, imax)
      DIM iter, es, fxl, fxu, fxr, xrold
      iter=0
      fxl=f(xl)
      fxu=f(xu)
      xrold=xl+(xu-xl)/3
      DO
            iter = iter+1
            xr = (xu + xl)/2  ' Bisection method
            fxr = f(xr)
            IF xr = 0 then
                  es = ABS(xr - xrold)
            ELSE
```

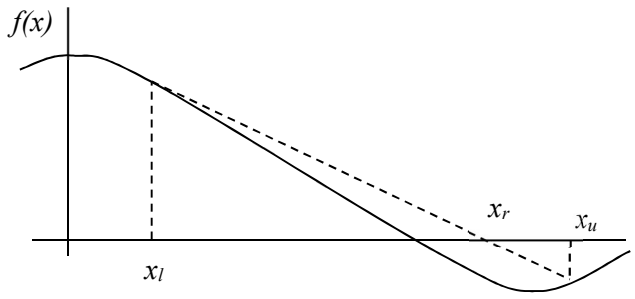
```
                es = ABS((xr - xrold)/xr)
            END IF
' if fxr and fxu have different signs, replace lower bound
            IF fxr*fxu < 0 THEN
                xl = xr
                fxl = fxr
            ELSE // replace upper bound
                xu = xr
                fxu = fxr
            END IF
            xrold = xr
        UNTIL iter ≥ imax OR es ≤ ea
        Bisection = xr
END Bisection
```

Examples:

1. Find all of the real roots of
   a. $f(x) = \sin(10x) + \cos(3x)\ ;\ 0 \le x \le 5$
   b. $f(x) = -0.6x^2 + 2.4x + 5.5$
   c. $f(x) = x^{10} - 1\ ;\ 0 \le x \le 1.3$
   d. $f(x) = 4x^3 - 6x^2 + 7x - 2.3$
   e. $f(x) = -26 + 85x - 91x^2 + 44x^3 - 8x^4 + x^5$

2.2.2 False Position Method

The bisection method works fairly well, but convergence can be improved if the root lies close to one of the bounds. Consider the figure shown. By similar triangles,



$$\frac{f(x_l)}{x_r - x_l} = \frac{f(x_u)}{x_r - x_u}$$

Solving for x_r gives

$$x_r = \frac{x_u f(x_l) - x_l f(x_u)}{f(x_l) - f(x_u)} = x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$$

This new root estimate replaces the bound x_u or x_l whose function value has the same sign as $f(x_r)$. The termination criterion is the same as for the bisection method.

The false position method is generally more efficient than bracketing, but not always (consider, for example, the function $f(x) = x^{10}-1$ between $x = 0$ and $x = 1.3$). The false position method can tend to be one-sided, leading to slow convergence. If this appears to be a problem, try the *modified false position method*. In this technique, if one bound is fixed for two successive iterations, bisect the interval once and proceed with the false position method.

2.3 Open Methods

The bracketing and false position methods are "closed" methods, that is, they "close" an interval and converge on the root from both ends of that interval. Open methods require only one (sometimes two) starting values that do not bracket the root, making them self-starting and more efficient. However, they can diverge and even move away from the root that is sought.

2.3.1 Simple Fixed-Point Iteration

Some functions can be manipulated to be of the form $x = g(x)$, either algebraically or by adding x to both sides of $f(x)=0$. If this is the case, one can converge on a root by iterating

$$x_{i+1} = g(x_i)$$

with termination criterion

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right|$$

While this method is easy to implement, it has several drawbacks. Convergence can be slow; at best it is linear. Also, the method can diverge, with convergence determined by the sign of the first derivative of $g(x)$: if $|g'(x)| < 1$ then the method converges, if $|g'(x)| > 1$ then fixed-point iteration diverges.

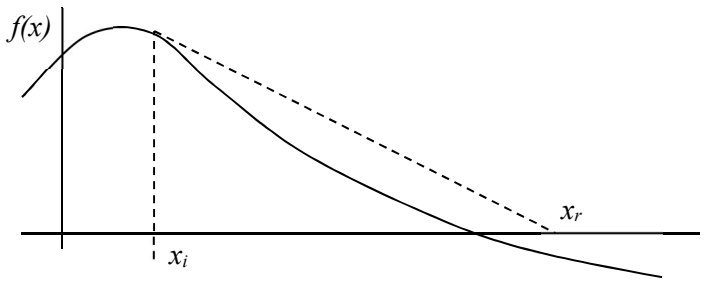
Pseudocode – Fixed Point Iteration

```
FUNCTION FixedPoint(x0, es, imax, iter, ea)
      xr = x0
      iter = 0
      DO
            iter = iter + 1
            xr = g(xrold) ' fixed point iteration
            IF iter>1 then
                  IF xr = 0 then
                        es = ABS(xr - xrold)
```

```
                ELSE
                      es = ABS((xr - xrold)/xr)
                END IF
            END IF
            xrold=xr
        END DO
        FixedPoint = xr
END FixedPoint
```

2.3.2 Newton-Raphson Method

Newton-Raphson is the most widely used method of the root-finding formulas. The tangent to the curve at the point x_i, $f(x_i)$ is used to determine the next estimate for the root. The slope of the curve at the point x_i can be written as



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

so that

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

with termination criterion

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right|$$

Newton-Raphson is quadratically convergent, that is, $E_{i+1} \approx E_i^2$. The method is very fast and very efficient. Care must be taken, however, since

- N-R can diverge if the tangent to the curve takes it away from the root
- N-R can converge slowly if multiple roots exist. Two methods exist to deal with multiple roots:

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

where m is the multiplicity of the root, or

$$x_{i+1} = x_i - \frac{f(x_i)f'(x_i)}{\left[f'(x_i)\right]^2 - f(x_i)f''(x_i)}$$

- It must be noted that Newton-Raphson method needs an analytical function to work since the derivatives must be explicitly determined.

2.3.3 Secant Method

This method is similar to Newton-Raphson, substituting a backward finite-difference approximation for the derivative:

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

So that

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

The secant method requires two points to start, x_{i-1} and x_i. It also may diverge, similar to the Newton-Raphson method.

2.3.4 Modified Secant Method

Instead of using a finite difference approximation of the derivative in Newton-Raphson, estimate the derivative using a small perturbation of the independent variable:

$$f'(x_i) \approx \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

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

2.3.5 Multiple Roots

Multiple roots, for example *f(x)=(x-a)(x-a)(x-b)* cause difficulties when searching for roots. Bracketing methods do not work with multiple roots (why?). In addition, *f'(x)=0* at the root, causing problems for the Newton-Raphson and the Secant methods.

2.3.6 Multivariate Methods

Given a set of equations $\boldsymbol{f(x)} = 0$

$$\begin{gathered} f_1(x_1, \dots x_N) = f_1(\boldsymbol{x}) = 0 \\ f_2(x_1, \dots x_N) = f_2(\boldsymbol{x}) = 0 \\ \dots \\ f_N(x_1, \dots x_N) = f_N(\boldsymbol{x}) = 0 \end{gathered}$$

The first-order Taylor expansion can be written as

$$\boldsymbol{f}(\boldsymbol{x} + \delta\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{J}(\boldsymbol{x})\delta\boldsymbol{x}$$

where the *Jacobian* of $\boldsymbol{f(x)}$ is

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_N}{\partial x_1} & \cdots & \frac{\partial f_N}{\partial x_N} \end{bmatrix}$$

Solving for $\delta\boldsymbol{x}$, the multivariate Newton-Raphson can be expressed as

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - [\boldsymbol{J}(\boldsymbol{x}_i)]^{-1}\boldsymbol{f}(\boldsymbol{x}_i)$$

Example:

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{cases} 3x_1 - \cos(x_2 x_3) - \frac{3}{2} = 0 \\ 4x_1^2 - 625x_2^2 + 2x_3 - 1 = 0 \\ 20x_3 + e^{-x_1 x_2} + 9 = 0 \end{cases}$$

$$\boldsymbol{J}(\boldsymbol{x}) = \begin{bmatrix} 3 & x_3\sin(x_2 x_3) & x_2\sin(x_2 x_3) \\ 8x_1 & -1250x_2 & 2 \\ -x_2 e^{-x_1 x_2} & -x_1 e^{-x_1 x_2} & 20 \end{bmatrix}$$

If the analytical derivatives are not available, it is possible to approximate the Jacobian from two consecutive iterations (multivariate Secant method)

$$J_{ij} = \frac{\partial f_i(x_i, \cdots, x_N)}{\partial x_j} = \frac{f_i\left(x_1^{(n)}, \cdots, x_j^{(n+1)}, \cdots x_N^{(n)}\right) - f_i\left(x_1^{(n)}, \cdots, x_j^{(n)}, \cdots x_N^{(n)}\right)}{x_j^{(n+1)} - x_j^{(n)}}$$

2.4 Roots of Polynomials

Finding all of the roots of a polynomial is a common problem in numerical analysis. Before delving into the methods, let's first examine efficient ways to evaluate and manipulate polynomials.

Evaluation of Polynomials

Consider the following polynomial:

$$f_3(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

Evaluating the function as it is written involves 6 multiplications and three additions. However, if it is written

$$f_3(x) = ((a_3x + a_2)x + a_1)x + a_0$$

it can be evaluated with only three multiplications and three additions. In pseudocode, given a vector of coefficients `a(j)`,

```
DO FOR j=n to 0 STEP -1
      df = df * x + p
      p = p * x + a(j)
END DO
```

Note that in the pseudocode above, the derivative of the polynomial, `df`, is evaluated at the same time as the function.

Polynomial Deflation

Recall that polynomials can be divided in a manner similar to basic arithmetic, sometimes referred to as *synthetic division*:

$$
\begin{array}{r}
x+6 \\
(x-4)\overline{)\,x^2+2x-24} \\
\underline{-(x^2-4x)} \\
6x-24 \\
\underline{-(6x-24)} \\
0
\end{array}
$$

So that $(x^2 + 2x - 24) = (x - 4)(x + 6)$. In the example here, if $(x - 4)$ was not a factor of the polynomial, there would have been a remainder.

Using this idea, once we find a root of an *n*th-order polynomial we can divide it out (*deflating* the polynomial) and continue work with a new polynomial of order *n*-1. However, this process is very sensitive to round-off error. *Forward deflation* is where the roots are found from smallest to largest, *backward deflation* is where the roots are found and the polynomial deflated from largest to smallest. *Root polishing* is a technique where the polynomial is deflated as the roots are found, and then those roots are used as better initial guesses for a second attempt, often in the opposite direction.

Conventional Methods

Since the roots of polynomials are often complex, this has to be a consideration for any root-finding method applied. Bracketing methods do not work at all for complex roots. Newton-Raphson (and its alternative methods) works well if complex arithmetic is implemented, with all of the same divergence possibilities already discussed.

2.4.1 Müller's Method

Similar to the Secant Method, which projects a line through two function values, Müller's Method projects a parabola through three values to estimate the root. Fit a parabola of the form

$$f(x) = a(x - x_2)^2 + b(x - x_2) + c$$

where x_2 is the root estimate, to intersect three points: $[x_0, f(x_0)]$, $[x_1, f(x_1)]$ and $[x_2, f(x_2)]$

$$
\begin{aligned}
f(x_0) &= a(x_0 - x_2)^2 + b(x_0 - x_2) + c \\
f(x_1) &= a(x_1 - x_2)^2 + b(x_1 - x_2) + c \\
f(x_2) &= a(x_2 - x_2)^2 + b(x_2 - x_2) + c = c
\end{aligned}
$$

now let

$$h_0 = x_1 - x_0$$
$$h_1 = x_2 - x_1$$
$$\delta_0 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$
$$\delta_1 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

so that

$$a = \frac{\delta_1 - \delta_0}{h_1 - h_0}$$
$$b = ah_1 + \delta_1$$
$$c = f(x_2)$$

To find the new root estimate, x_3, apply the alternate form of the quadratic formula:

$$x_3 - x_2 = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

or

$$x_3 = x_2 + \frac{-2c}{b \pm \sqrt{b^2 - 4ac}}$$

which yields either two real roots or a complex conjugate pair. By convention, the sign taken to be the same sign as b, which always yields the root estimate closer to x_2. Then

- If only real roots are considered, for the next iteration choose the two points closest to the new root estimate x_3 and apply the method again to refine the root estimate.
- If complex roots are possible then proceed in sequence, that is, $x_1 \rightarrow x_0$, $x_2 \rightarrow x_1$, $x_3 \rightarrow x_2$ and go through the method again to determine a better root estimate.

Pseudocode – Müller's Method

```
SUB Muller(xr, h, eps, maxit)
x2 = xr
x1 = xr + h*xr
x0 = xr - h*xr
DO
      iter = iter +1
```

```
        h0 = x1 - x0
        h1 = x2 - x1
        d0 = (f(x1) - f(x0)) / h0
        d1 = (f(x2) - f(x1)) / h1
        a = (d1 - d0) / (h1 + h0)
        b = a*h1 + d1
        c = f(x2)
        rad = SQRT(b*b - 4*a*c)
        IF |b+rad| > |b-rad| THEN
              den = b + rad
        ELSE
              den = b - rad
        END IF
        dxr = -2*c / den
        xr = x2 + dxr
        PRINT iter, xr
        IF (|dxr| < eps*xr OR iter > maxit) EXIT
        x0 = x1
        x1 = x2
        x2 = xr
END DO
END Muller
```

2.4.2 Bairstow's Method

If we have a general polynomial

$$f_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

that is divided by a factor (x-t), it yields a polynomial that is one order lower

$$f_{n-1}(x) = b_0 + b_1 x + b_2 x^2 + \ldots + b_n x^{n-1}$$

where

$$b_n = a_n$$
$$b_i = a_i + b_{i+1} t$$

and $i = n$-1 to 0. If t is a root of the original polynomial, then $b_0 = 0$.

Bairstow's Method divides the polynomial by a quadratic factor, ($x^2 - rx - s$) to yield

$$f_{n-2}(x) = b_2 + b_3 x + b_4 x^2 + \ldots + b_n x^{n-2}$$

with remainder

$$R = b_1(x - r) + b_0$$

and

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} + rb_n \\ b_i &= a_i + rb_{i+1} + sb_{i+2} \end{aligned}$$

where $i = n$-2 to 0. The idea behind Bairstow's Method is to drive the remainder to zero. To do this, both b_1 and b_0 must be zero. Expand both in first-order Taylor series:

$$\begin{aligned} b_1(r + \Delta r, s + \Delta s) &= b_1 + \frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s \\ b_0(r + \Delta r, s + \Delta s) &= b_0 + \frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s \end{aligned}$$

so that

$$\begin{aligned} \frac{\partial b_1}{\partial r}\Delta r + \frac{\partial b_1}{\partial s}\Delta s &= -b_1 \\ \frac{\partial b_0}{\partial r}\Delta r + \frac{\partial b_0}{\partial s}\Delta s &= -b_0 \end{aligned}$$

Now let

$$\begin{aligned} c_n &= b_n \\ c_{n-1} &= b_{n-1} + rc_n \\ c_i &= b_i + rc_{i+1} + sc_{i+2} \end{aligned}$$

where $c_1 = \partial b_0 / \partial r$, $c2 = \partial b_0 / \partial s = \partial b_1 / \partial r$, $c_3 = \partial b_1 / \partial s$, etc., so that

$$\begin{aligned} c_2\Delta r + c_3\Delta s &= -b_1 \\ c_1\Delta r + c_2\Delta s &= -b_0 \end{aligned}$$

Solve these two equations for Δr and Δs, then use them to improve the initial guesses of r and s. At each step, the approximate errors are

$$\varepsilon_{a,r} = \left|\frac{\Delta r}{r}\right|$$
$$\varepsilon_{a,s} = \left|\frac{\Delta s}{s}\right|$$

When both of these error estimates fall below a specified value, then the root can be identified as

$$x = \frac{r \pm \sqrt{r^2 + 4s}}{2}$$

and the deflated polynomial with coefficients b_i remains. Three possibilities exist:

1. The polynomial is third-order or higher. In this case, apply the method again to find the root(s).
2. The remaining polynomial is quadratic – solve for the two remaining roots with the quadratic formula.
3. The polynomial is linear. In this case, the last root is $x = -s/r$

Pseudocode – Bairstow's Method

```
SUB Bairstow(a, nn, es, rr, ss, maxit, re, im, ier)
DIMENSION b(nn), c(nn)
r= rr
s = ss
n = nn
ier = 0
ea1 = 1
ea2 = 1
DO
      IF n<3 OR iter>= maxit EXIT
      iter = 0
      DO
            iter = iter +1
            b(n) = a(n)
            b(n-1) = a(n-1) + r*b(n)
            c(n) = b(n)
            c(n-1) = b(n-1) + r*c(n)
            DO i = n-2, 0, -1
                  b(i) = a(i) + r*b(i+1) + s*b(i+2)
                  c(i) = b(i) + r*c(i+1) + s*c(i+2)
```

```
            END DO
            det = c(2)*c(2) - c(3)*c(1)
            IF det <> 0 THEN
                  dr = (-b(1)*c(2) + b(0)*c(3))/det
                  ds = (-b(0)*c(2) + b(1)*c(1))/det
                  r = r + dr
                  s = s + ds
                  IF r<>0 THEN ea1 = ABS(dr/r)*100
                  IF s<>0 THEN ea2 = ABS(ds/s)*100
            ELSE
                  r = r + 1
                  s = s + 1
                  iter = 0
            END IF
            IF ea1 <= es AND ea2 <=es OR iter >= maxit EXIT
      END DO
      CALL QuadRoot(r, s, r1, i1, r2, i2)
      re(n) = r1
      im(n) = i1
      re(n-1) = r2
      im(n-1) = i2
      n = n-2
      DO i = 0, n
            a(i) = b(i+2)
      END DO
END DO
IF iter < maxit THEN
      IF n = 2 THEN
            r = -a(1)/a(2)
            s = -a(0)/a(2)
            CALL Quadroot(r, s, r1, i1, r2, i2)
            re(n) = r1
            im(n) = i1
            re(n-1) = r2
            im(n-1) = i2
      ELSE
            re(n) = -a(0)/a(1)
            im(n) = 0
      END IF
ELSE
      ier = 1
END IF
End Bairstow

SUB Quadroot(r, s, r1, i1, r2, i2)
disc = r*r + 4*s
IF disc > 0 THEN
      r1 = (r + SQRT(disc))/2
      r2 = (r - SQRT(disc))/2
```

```
      i1 = 0
      i2 = 0
ELSE
      r1 = r/2
      r2 = r1
      i1 = SQRT(ABS(disc))/2
      i2 = -i1
END IF
END Quadroot
```