5. Solution of Nonlinear Equations

The concern here is to find the roots of a nonlinear equation or a nonlinear system of equations.

An equation is called a *nonlinear equation* if it is not linear in each of the unknowns. The following is an example of a nonlinear equation.

$$3x^2 + 4x + 5 = 0$$

A system of equations is called a *nonlinear system of equations* if at least one of the equations is not linear in one of the unknowns. An example of a nonlinear system is shown below.

$$\begin{cases} 4x_1^2 + 9x_2^2 - 10 = 0 \\ x_1x_2 - 2x_1 - 1 = 0 \end{cases}$$

4.1 Solving a Single Nonlinear Equation

Given a continuous and real-valued function $f(x)$, we will three methods to find an $r$ such that $f(r) = 0$. The first method is the *bisection method*.

(1). Bisection Method

If $f(x)$ changes sign at $x = a$ and $x = b$ then, by continuity, the interval $[a, b]$ contains at least one root of $f(x)$ (see Figure 4.1 for two examples).

![Figure 4.1](image)

Figure 4.1 $f(x)$ has (a) one root, and (b) five roots between $a$ and $b$.

To find a root of $f(x)$ between $a$ and $b$, we compute the value of $f(x)$ at $c = (a+b)/2$. If $f(c) = 0$ then we output $x = c$ as the root of $f(x)$. If $f(c) \neq 0$ and the sign of $f(c)$ is different from the sign of $f(a)$ then we continue the search on the interval on the interval $[a, c]$. Otherwise, continue the search on the interval $[c, b]$. We continue this bisecting process until a stop criterion is satisfied. A typical criterion is when the difference of the values of $f(x)$ at the endpoints of the new interval is smaller than some pre-set error bound $\varepsilon$. When this condition is
satisfied, the search process stops and the midpoint of the new interval is output as an approximation to a root of \( f(x) \). As algorithm, called BISECT, is shown below for this process.

---

**Algorithm: BISECT**

Input: a function \( f(x) \), an interval \([a_0, b_0]\) such that \( \text{sign } f(a_0) \neq \text{sign } f(b_0) \), and an error bound \( \varepsilon \).

Output: an approximation to a root of \( f(x) \) between \( a_0 \) and \( b_0 \).

\[
\begin{align*}
\text{var } & \ a, b, c : \text{real}; \\
1. & \ a := a_0; \ b := b_0; \ c := a + (b-a)/2; \\
2. & \text{while } (|f(a) - f(b)| > \varepsilon) \text{ do begin } \\
& \quad \text{if } (f(c) = 0) \text{ then stop } \\
& \quad \text{else begin } \\
& \quad \quad \text{if } (\text{sign } f(c) \neq \text{sign } f(a)) \text{ then } b := c; \\
& \quad \quad \text{else } a := c; \\
& \quad \quad c := a + (b-a)/2; \\
& \quad \end{end}; \\
3. & \text{return } c. 
\end{align*}
\]

---

Another stop criterion frequently used is to test if the distance between \( a \) and \( b \) is smaller than a given error bound \( \varepsilon \). In this case, in \( n \) steps, an approximate root will be computed with an error smaller than or equal to

\[
\frac{(b-a)}{2^{n+1}}
\]

Therefore, to make the error smaller than or equal to \( \varepsilon \), the iteration in the bisection method should be repeated at least the following times.

\[
-1 + \frac{\log(b-a)}{\log2} \leq \frac{\varepsilon}{\log2}
\]

(2). Regula Falsi (Rule of false position)
This method is a modification of the bisection method. In algorithm BISECT, the midpoint of the current interval is always used for \( c \) to form a new interval. Therefore, the size of the new interval is always one half of that of the current interval. One way to improve the performance of this algorithm is to reduce the size of each new interval so that one can get an equally good approximation of the root in less iteration steps. This can be achieved by choosing a point close to where the root of the function is more likely to be. Note that if \( f(x) \) has different sign at \( a \) and \( b \) and the absolute value of \( f(a) \) is bigger than the absolute value of \( f(b) \), then it is in general true that the root of \( f(x) \) is closer to \( b \) than to \( a \). Therefore, in this case, a better choice for \( c \) would be the intersection point of the line segment that connects the end points of the curve (Figure 4.2), instead of the midpoint of \([a, b]\). Consequently, in this method, \( c \) is computed using the following approach:

\[
c = a + \frac{f(a)}{f(a)-f(b)}(b-a)
\]

instead of \( c = a + (b-a)/2 \).

![Figure 4.2 A better choice for \( c \).](image)

(3). Newton-Raphson Method

With an initial guess \( x_0 \), this approach constructs a sequence of points \( \{ x_k \mid k = 1, 2, 3, \cdots \} \) so that, hopefully, this sequence of points would converge to a root of \( f(x) \).

![Figure 4.3 Newton-Raphson method.](image)
The basic idea of this approach is as follows (see Figure 4.3). If \( x_k \) is close enough to a root \( r \) of \( f(x) \), then by constructing the tangent line of \( f(x) \) at \( x = x_k \) and find its intersection with the \( x \)-axis, we would get a point \( x_{k+1} \) closer to \( r \) than \( x_k \). If we repeat the same process for \( x_{k+1} \), we would get a point \( x_{k+2} \) even closer to \( r \). Therefore, the sequence of points constructed this way, \{ \( x_k \) \}, would converge to \( r \) when \( k \) tends to infinity.

From Figure 4.3, it is easy to see that the equation of the tangent line of \( f(x) \) at \( x = x_k \) is

\[
y = f(x_k) + (x - x_k)f'(x_k)
\]

where \( f'(x_k) \) is the derivative of \( f(x) \) at \( x_k \). The \( x \) intercept of this equation with the \( x \)-axis is \((x_{k+1}, 0)\) with

\[
x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}
\] (4.1)

With an initial guess \( x_0 \), the Newton-Raphson method computes a sequence of points \( x_1, x_2, x_3, \ldots \) using (4.1). The computation process stops when the difference between two consecutive points is smaller than a given error bound \( \varepsilon \).

This method is convergent only when we are sufficiently close to a root \( r \) of \( f(x) \). If the slope of \( f(x) \) is small in a region, we usually can not guarantee the convergence of this method (see Figure 4.4 for an example). If the explicit representation of \( f'(x) \) is not provided, we can use the following divided difference of \( f(x) \) to approximate \( f'(x_k) \).

\[
f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}
\]

![Figure 4.4](image.png)

Figure 4.4 An example where the Newton-Raphson method is not convergent.

### 4.2 The Special Case of Polynomials

Given a polynomial \( f(x) \) defined as follows,

\[
f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n
\] (4.2)

how would we find a root of \( f(x) \)? If the Newton-Raphson method is to be used, we need to
make a initial guess first, and then apply (4.1) repeatedly to construct a sequence of points. The evaluation of (4.1) involves the evaluation of both \( f(x) \) and \( f'(x) \). In this section, we will develop efficient algorithms for the evaluation of a polynomial and its derivative at a given point so that the evaluation of (4.1) can be performed efficiently.

A polynomial can be uniquely represented in a computer by its degree \( n \) and its \( n+1 \) coefficients. To evaluate the value of a polynomial at a given point \( x = s \), we use Horner's rule. The polynomial is nested as follows and then evaluated outward.

\[
f(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + a_n x)))
\]

The algorithm that does this is shown below:

\[
b_n := a_n \;
\]

\[
\text{for } i = n-1 \text{ downto } 0 \text{ do } \\
\quad b_i := a_i + b_{i+1} \times s \; ;
\]

When the process stops, \( b_0 = f(s) \). For instance, the evaluation of \( f(x) = 12 - 2x - 6x^2 + x^3 \) at \( x = 2 \) is done as follows.

\[
b_3 := 1 ; \\
b_2 := -6 + b_3 \times 2 = -4 ; \\
b_1 := -2 + b_2 \times 2 = -10 ; \\
b_0 := 12 + b_1 \times 2 = -8 ;
\]

The value of \( f(2) \) is \(-8\).

To evaluate the value of \( f'(x) \) at \( x = s \), note that if \( b_n, b_{n-1}, \ldots, b_1 \) and \( b_0 \) are defined as in (4.3) and (4.4), i.e.,

\[
b_n = a_n
\]

and

\[
b_i = a_i + b_{i+1} \times x ; \quad i = n-1, \ldots, 2, 1, 0
\]

where \( a_i \) are the coefficients of \( f(x) \) defined in (4.2), then each \( a_i \) with \( 0 \leq i \leq n-1 \) can be expressed as

\[
a_i = b_i - b_{i+1} \times s
\]

and we have

\[
f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \\
= (b_0 - b_1 s) + (b_1 - b_2 s)x + (b_2 - b_3 s)x^2 + \cdots + (b_{n-1} - b_n s)x^{n-1} + b_n x^n
\]
Therefore $f(x)$ can be written in the following form

$$f(x) = (x-s)g(x) + b_0 \quad (4.5)$$

with $b_0 = f(s)$ and

$$g(x) = b_1 + b_2x + b_3x^2 + \cdots + b_nx^{n-1}. \quad (4.6)$$

$(4.5)$ is a very important observation. $(4.5)$ implies that

$$f'(x) = g(x) + (x-s)g'(x).$$

Therefore, if $x = s$, we have

$$f'(s) = g(s) \quad (4.7)$$

That is, to compute the value of $f'(x)$ at $x=s$, one only needs to evaluate $g(x)$ at $x=s$. Since we get the coefficients of $g(x)$ when we evaluate $f(s)$, the evaluation process of $g(s)$ can be integrated with the evaluation process of $f(s)$, as follows.

$$b_n := a_n ;$$
$$c_n := b_n ;$$

for $i = n-1 \text{ downto } 1$ do

$$b_i := a_i + b_{i+1}*s ;$$
$$c_i := b_i + c_{i+1}*s ;$$
$$b_0 := a_0 + b_1*s ;$$

When the process stops, we not only have the value of $f(s)$, but the value of $f'(s)$ as well. The value of $f(s)$ is in $b_0$ and the the value of $f'(s)$ is in $c_1$. For instance, in the above example of computing the value of $f(x) = 12-2x-6x^2+x^3$ at $x = 2$, we can simultaneously compute $f(2)$ and $f'(2)$, as follows.

$$b_3 := 1 ; \quad c_3 := 1 ;$$
$$b_2 := -6 + b_3*2 = -4 ; \quad c_2 := -4 + c_3*2 = -2 ;$$
$$b_1 := -2 + b_2*2 = -10 ; \quad c_1 := -10 + c_2*2 = -14 ;$$
$$b_0 := 12 + b_1*2 = -8 ;$$
The value of $f'(2)$ is $-14$. 