2.1. **Iterative methods for roots of a single nonlinear equation.** Suppose $f(x)$ is a continuous function on the interval $[a, b]$ and satisfies $f(a)f(b) < 0$. Then the Intermediate Value Theorem says there is at least one number s, with $a < s < b$, such that $f(s) = 0$.

Example: $f(x) = x^3 - x - 1 = 0$. Now $f(1) = -1$ and $f(2) = 5$ so there is at least one root in the interval $(1, 2)$. In some cases, it is possible to show there is only one root in an interval. For example, $f'(x) = 3x^2 - 1$ and on the interval $[1, 2]$, $f'(x) \geq 2 > 0$. Hence, $f(x)$ is strictly increasing on $[1, 2]$, so there is only one root.

Example: $f(x) = x^3 - 3x - 1 = 0$. Now $f(-2) = -3$, $f(2) = 1$, so there is at least one root in $(-2, 2)$. In fact, there are three roots in this interval. Note that $f(-1) = 1$, so there is at least one root in $(-2, -1)$. $f(0) = -1$, so there is a second root in $(-1, 0)$. The third root is in the interval $(0, 2)$. Note that we cannot draw any conclusion about whether there are roots in the interval $[-1, 2]$ from the fact that both $f(-1)$ and $f(2)$ are positive.

The simplest scheme to find a root s is to use the *method of bisection*.

Bisection Algorithm: Set $a_0 = a$ and $b_0 = b$. For $n = 0, 1, \ldots,$
Set $x_n = (a_n + b_n)/2$.
If $f(a_n)f(x_n) < 0$, set $a_{n+1} = a_n$, $b_{n+1} = x_n$.
If $f(x_n)f(b_n) < 0$, set $a_{n+1} = x_n$, $b_{n+1} = b_n$.
Then at each stage of the iteration, the root lies in the interval spanned by a_n and b_n. Hence

$$|s - x_n| \leq |b_n - a_n|/2.$$

Now since $|b_n - a_n| = |b_{n-1} - a_{n-1}|/2$, we easily get that

$$|s - x_n| \leq |b - a|/2^{n+1}, \quad n \geq 0.$$

Thus, we can achieve any desired accuracy by taking n sufficiently large.

Example: Given a tolerance ϵ, find a number of iterations n that guarantees that $|s - x_n| \leq \epsilon$. We choose n so that $|b - a|/2^{n+1} \leq \epsilon$. This will guarantee that $|s - x_n| \leq \epsilon$. To do this, we need to find a value of n such that $2^{n+1} \geq |b - a|/\epsilon$. Hence, we choose n so that

$$(n + 1) \ln 2 \geq \ln[|b - a|/\epsilon], \quad \text{i.e.,} \quad (n + 1) \geq \frac{\ln[|b - a|/\epsilon]}{\ln 2}.$$

Implementation in a *Matlab* computer code.

```
% Bisection
% a = left end point of interval containing the root
% b = right  end point of interval containing the root
% tolx = error tolerance in x
% tolf = error tolerance in the function value
% N = the current iteration number
% Nmax = maximum number of iterations
% fcn.m is the name of the file containing the function
format long
a=1;
```

```
b=4;
N=1;
Nmax = 50;
tolx = .001;
tolf = 0.000001;
fa = feval('fcn',a);
fb = feval('fcn',b);
m =(a+b)/2;
fm = feval('fcn',m);
while (abs(b-a) > tolx) & (abs(fm) > tolf) & (N < Nmax)
[N,a,b,m,fm]
 if fa*fm <=0;
  b = m;
  fb = fm;
 else a = m;
 fa= fm;
end
 m = (a+b)/2;
 fm = feval('fcn', m);
 N= N+1;
end
```

To use this program, first create a *Matlab m-file* with the name `fcn.m`. Note that such a file must have the extension *.m* and must be placed in the directory from which you are running *Matlab*. For example, for the function $f(x) = x - \cos x$, the contents of the file `fcn.m` would be:

```
function f = fcn(x)
f = x - cos(x);
```

Method of False Position: Instead of choosing x_n as the midpoint of the points bracketing the root, we choose it as the weighted average of these points, with the weights depending on the size of the function values. This can be done by choosing x_n as the point where the secant line joining the points $(a_n, f(a_n)$ and $(b_n, f(b_n))$ crosses the x axis. This line is given by

$$y - f(a_n) = \frac{f(b_n) - f(a_n)}{b_n - a_n}(x - a_n).$$

When $y = 0$, we get that

$$x \equiv x_n = a_n - \frac{b_n - a_n}{f(b_n) - f(a_n)} f(a_n)$$

To maintain the root bracketing property, we could then proceed as in the bisection algorithm, i.e.,
If $f(a_n)f(x_n) < 0$, set $a_{n+1} = a_n$, $b_{n+1} = x_n$.
If $f(x_n)f(b_n) < 0$, set $a_{n+1} = x_n$, $b_{n+1} = b_n$.
However, although the method of false position produces a point at which $|f(x)|$ is small

somewhat faster than the bisection method, it does not give a small interval in which the root is known to lie.

Consider the following example: $f(x) = x^2 - 2$ on the interval $[0, 2]$.

Then all the iterates x_n lie to the left of the root s, so although $s \in [x_n, b]$, $|b - x_n| \geq |b - s|$ for all n, i.e., the size of the interval in which the root is know to lie is not converging to zero.

Secant method: This method is similar to the method of false position, except that we drop the requirement that the root be bracketed. This, starting from two values x_0 and x_1 which do bracket the root, we simply define the sequence $\{x_n\}$ by

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

When it converges, this method converges faster than the method of false position.

The Dekker-Brent method This method combines the secant method and the method of bisection to produce an algorithm that has the advantages of both methods, i.e., it retains the root-bracketing property of the bisection method, but uses the secant method when appropriate to attain a higher rate of convergence. This method has been implemented in the Matlab routine *fzero* to find the root of a single nonlinear equation.

Newton's method: Geometrically, starting from an initial guess x_0, we define at each step a new approximation x_{n+1} as the position where the tangent line to the curve $y = f(x)$ at $x = x_n$ crosses the x-axis. Since the equation of this tangent line is given by

$$y - f(x_n) = f'(x_n)(x - x_n),$$

we get that when $y = 0$,

$$x = x_{n+1} = x_n - f(x_n)/f'(x_n).$$

Another way to think of Newton's method is that it is the approximation given by truncating the Taylor series expansion, i.e., we have

$$0 = f(s) = f(x_n) + f'(x_n)(s - x_n) + f''(\xi)(s - x_n)^2.$$

If x_n is close to s, then $s - x_n$ is small, so that $(s - x_n)^2$ is even smaller. Discarding this last term, we define x_{n+1} as the approximation to s which restores equality to this equation, i.e.,

$$f(x_n) + f'(x_n)(x_{n+1} - x_n) = 0.$$

Solving for x_{n+1}, we recover Newton's method.

One convenient way to write Newton's method is in the form $x_{n+1} = x_n + \delta_n$, where the increment δ_n is the solution of $f'(x_n)\delta_n = -f(x_n)$.