We have to find values of x such that the non-linear equation, $f(x) = 0$ is satisfied. When we say $f(x)$ is a non-linear function of x, it means that $f(x)$ cannot be written as $ax + b$, where a and b are constants. When we say that $f(x)$ is an algebraic equation, it means that $f(x)$ does not involve any differentials of the form $\frac{d^n y}{dx^n}$.

Some examples of non-linear are:

(1) $f_1(x) = x^2 + 5x + 6$

(2) $f_2(x) = \sin(x) + \cos(x)$

(3) $f_3(x) = x^3 - 3x^2 + 3x - 1 - e^x$

The roots cannot be obtained through analytical means except for a few simple cases, e.g. for $f_1(x)$. So our aim is to learn numerical methods which will evaluate the roots approximately.

Consider **Fig.1**. This is a plot of $f(x) = x^2 - 3$ over the range $-1 \leq x \leq 3$. The root, i.e., solution to the non-linear equation is indicated where $f(x) = 0$.

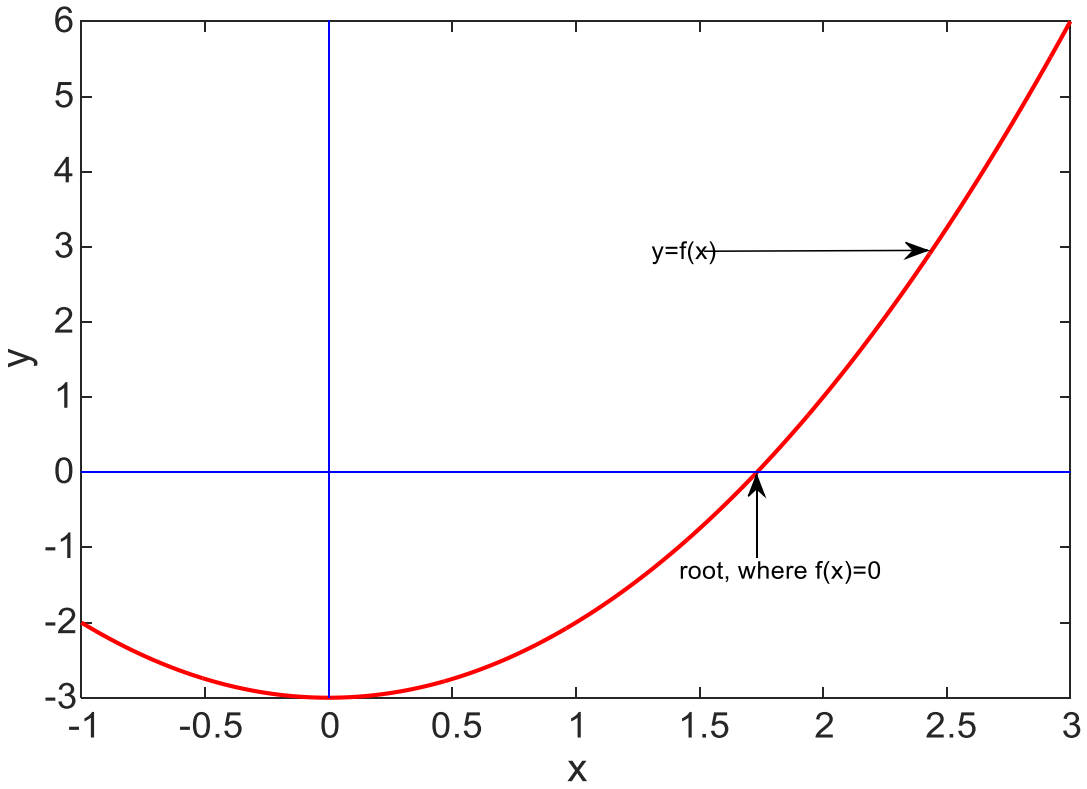

Fig.1

1. Bracketing Methods: Bisection Method and False Position Method

In these methods, the approximate location of the root is first determined by finding two values that bracket the root. These two values are the two initial guesses. But how do we know that these two values bracket the root?

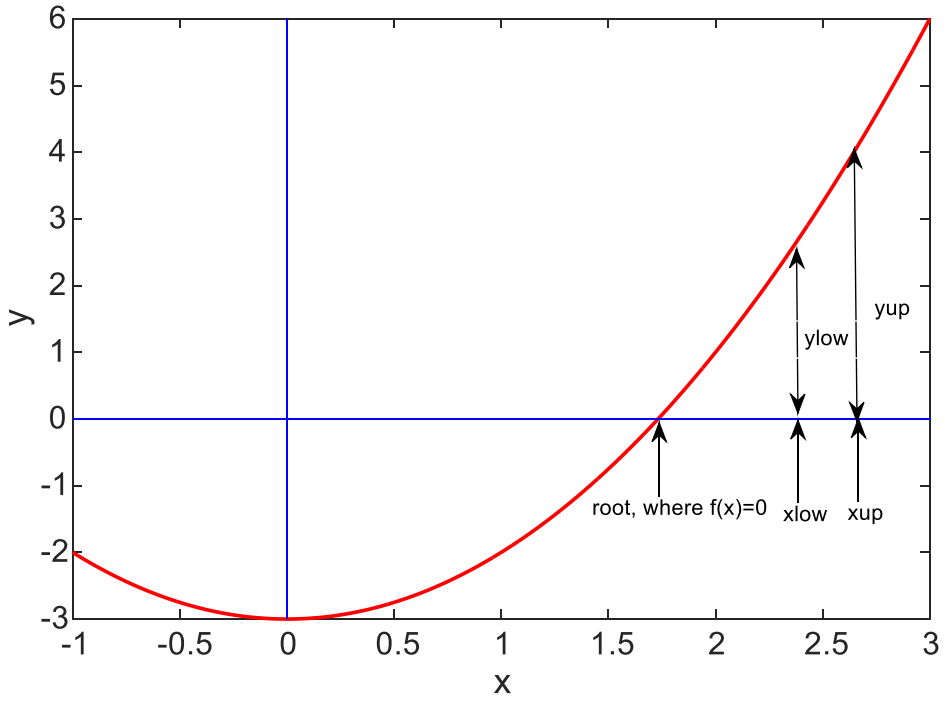

Fig. 2a

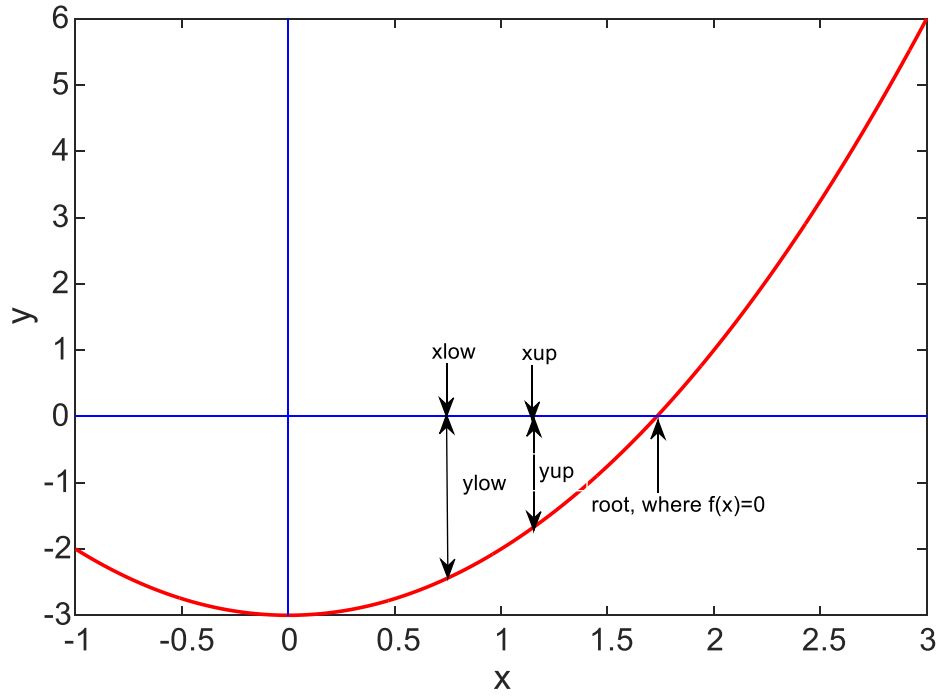

Fig.2b

In **Fig. 2a**, both the initial guesses are to the right side of the root and in **Fig. 2b**, both of them are to the left side of the root. In both cases, the initial guesses do not bracket the root and, y_{low} and y_{up} are of same sign.

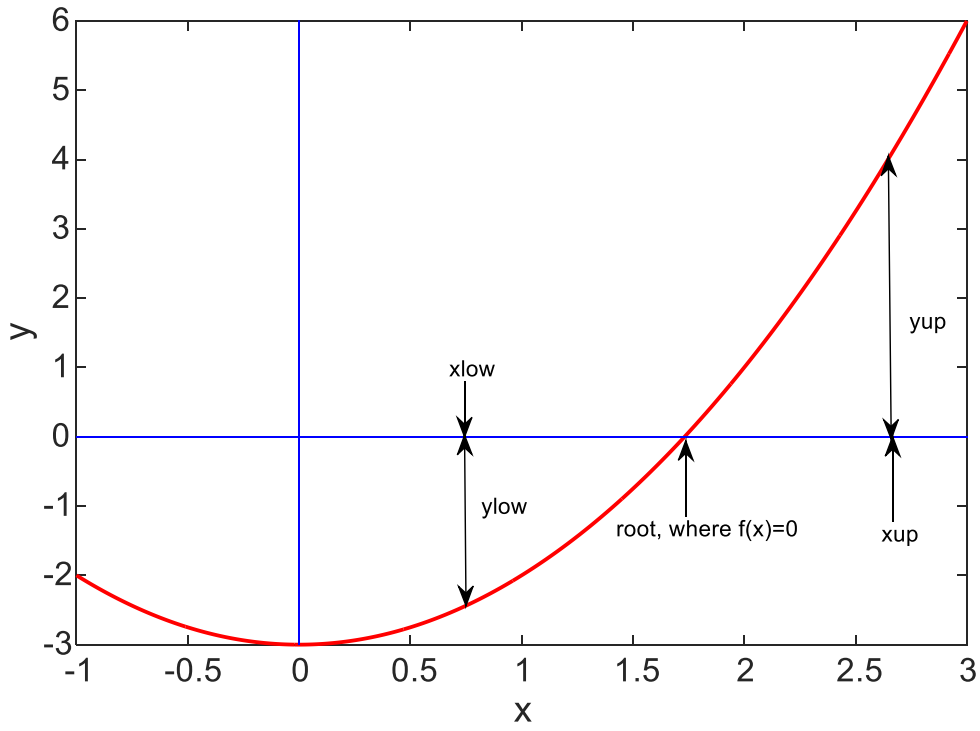

Fig.3

Consider **Fig.3**. Now, the two initial guesses bracket the root, one being to the left of the root and the other being to the right. Note that, y_{low} and y_{up} are of opposite sign. That's how we conclude that: ***a root is bracketed or enclosed if the function changes signs at the endpoints.***

Once we know that, the root is enclosed by these two initial guesses, we can find a third guess which is closer to the root than the original two values. Then we have to check to see if the new value is a root. Otherwise a new pair of bracket is generated from these three values, and the procedure is repeated.

1.1 Bisection Method

Steps:

1. Give the following as inputs: two initial guesses, x_{low} and x_{up}, and a tolerance on difference between x_{up} and x_{low}, namely, ε.
2. Check if the root is in this interval by computing $y_{low} = f(x_{low})$ and $y_{up} = f(x_{up})$, and finally calculating $y_{low} * y_{up}$.
   If $y_{low} * y_{up} > 0$, the root is most likely not in this interval and so quit saying that this method is not applicable.
   Else if $y_{low} * y_{up} < 0$, at least one root is in the interval, and proceed as follows.
3. Check if $(x_{up} - x_{low})$ is already less than or equal to ε. Use the initial guesses x_{low} and x_{up} to generate a third value that is closer to the root. The new point is calculated as mid-point between x_{low} and x_{up}. Let's call it x_{mid}.
   $$x_{mid} = \frac{x_{low} + x_{up}}{2}$$
   The method gets its name from this bisecting of two values. It is also known as interval halving method. Why? It will be evident when we will do some calculations.
4. Test whether x_{mid} is a root of $f(x)$ by evaluating the function at x_{mid}, e.g. finding value of $y_{mid} = f(x_{mid})$ and see if y_{mid} is indeed zero.
5. If x_{mid} is the root, the job is done and no further step is required.
   Else if x_{mid} is not a root, i.e., $y_{mid} \neq 0$, then there are two possible cases:
   (a) If $y_{mid} * y_{low} < 0$, then the root is in the left half of the interval and we should update x_{up} to x_{mid}.
   (b) If $y_{mid} * y_{low} > 0$, then the root is in the right half of the interval and we should update x_{low} to x_{mid}.
6. Continue steps 3 through 5 until interval width $(x_{up} - x_{low})$ has been reduced to a size $\leq \varepsilon$.

The next page illustrates a step-by-step calculation for the bisection process.

Bisection Method

Consider the following equation: $y = f(x) = x^2 - 3$

And the tolerance of difference between x_{up} and x_{low}, ε = 0.01 and $x_{low} = 1, x_{up} = 2$

The root is indeed in this interval, as $y_{up} = f(x_{up}) = 1$ and $y_{low} = f(x_{low}) = -2$, and hence, $y_{low} * y_{up} < 0$

x_{low}	x_{up}	$x_{up} - x_{low}$	$y_{low} = f(x_{low})$	$y_{up} = f(x_{up})$	$x_{mid} = \frac{x_{up} + x_{low}}{2}$	$y_{mid} = f(x_{mid})$	$y_{mid} * y_{low}$	Need to Update
1	2	1	-2	1	1.5000	-0.7500	>0	x_{low}
1.5000	2	0.5000	-0.7500	1	1.7500	0.0625	<0	x_{up}
1.5000	**1.7500**	0.2500	-0.7500	0.0625	1.6250	-0.3594	>0	x_{low}
1.6250	1.7500	0.1250	-0.3594	0.0625	1.6875	-0.1523	>0	x_{low}
1.6875	1.7500	0.0625	-0.1523	0.0625	1.7188	-0.0459	>0	x_{low}
1.7188	1.7500	0.0313	-0.0459	0.0625	1.7344	0.0081	<0	x_{up}
1.7188	**1.7344**	0.0156	-0.0459	0.0081	1.7266	-0.0190	<0	x_{up}
1.7188	**1.7266**	0.0078<0.01	-	-	-	-	-	-

The root in this method is the last value of $x_{mid} = 1.7266$. 7 iterations are required to reach this value. A lower value of ε would yield a better approximation of root, but would require more iterations.

Note that, the interval $(x_{up} - x_{low})$ is halved in each iteration. This shows why it is called interval halving method.

*The values that are updated are bold-faced.

Consider the following code for bisection method to do the same using MATLAB:

```
clc; clear all; close all;
f=inline('x^2-3'); %function declared as a numerical function
tol=0.01;%value of tolerance of the interval (xup-xlow)
%two initial guesses
xlow=1;
xup=2;
ylow=f(xlow);
yup=f(xup);
%checking whether a root is in this interval
if ylow*yup>0
    disp('Root is not likely in this interval');
else disp('Root is in this interval');
    iter=0; %calculating iterations
    while (xup-xlow>=tol) %assuming xup>xlow
        iter=iter+1;
        xmid=(xlow+xup)/2;%calculating xmid
        ymid=f(xmid);
        if ymid==0
            break; %if xmid=root, break the loop
        else %otherwise update xup or xlow
            ylow=f(xlow);
            yup=f(xup);
            if ymid*ylow>0
                xlow=xmid;
            else xup=xmid;
            end
        end
    end
    root=xmid;
    disp(root);
    disp(iter);
end
```

This code will find a root for the given function. But will it work for all functions? If not, for which functions will it fail?

Also, the given function has two roots? What do you have to change to obtain the other root? What modification has to be done to find all the roots in a given interval? Will that method be general? Can you find the imaginary roots in this method?

1.2 False Position Method

A shortcoming of the bisection method is that, in dividing the interval from x_{low} to x_{up} into equal halves, no account is taken of $f(x_{low})$ and $f(x_{up})$. Consider **Fig. 4**. Here, $f(x_{low})$ is closer to zero than $f(x_{up})$. So, it is most likely that the root is closer to x_{low} than to x_{up}. An alternative method that exploits this graphical insight is to join $f(x_{low})$ and $f(x_{up})$ by a straight line. The intersection of this line with the x-axis represents a better estimate of the root. The replacement of the curve by a straight line gives the false position of the root, hence, giving it the name, the False Position Method, or in Latin, Regula Falsi. It is also called linear interpolation method.

Consider **Fig.4**. The interpolated line is a straight line from (x_{low}, y_{low}) to (x_{up}, y_{up}).

The equation of such a straight line is,

$$\frac{x - x_{up}}{y - y_{up}} = \frac{x_{low} - x_{up}}{y_{low} - y_{up}}$$

$$or\ x = x_{up} + \frac{(y - y_{up})(x_{low} - x_{up})}{y_{low} - y_{up}}$$

At $x = x_m, y = 0$. So, it becomes,

$$x_m = x_{up} - \frac{y_{up}(x_{low} - x_{up})}{y_{low} - y_{up}} = x_{up} - \frac{f(x_{up})(x_{low} - x_{up})}{f(x_{low}) - f(x_{up})}$$

This x_m is a better approximation of the root.

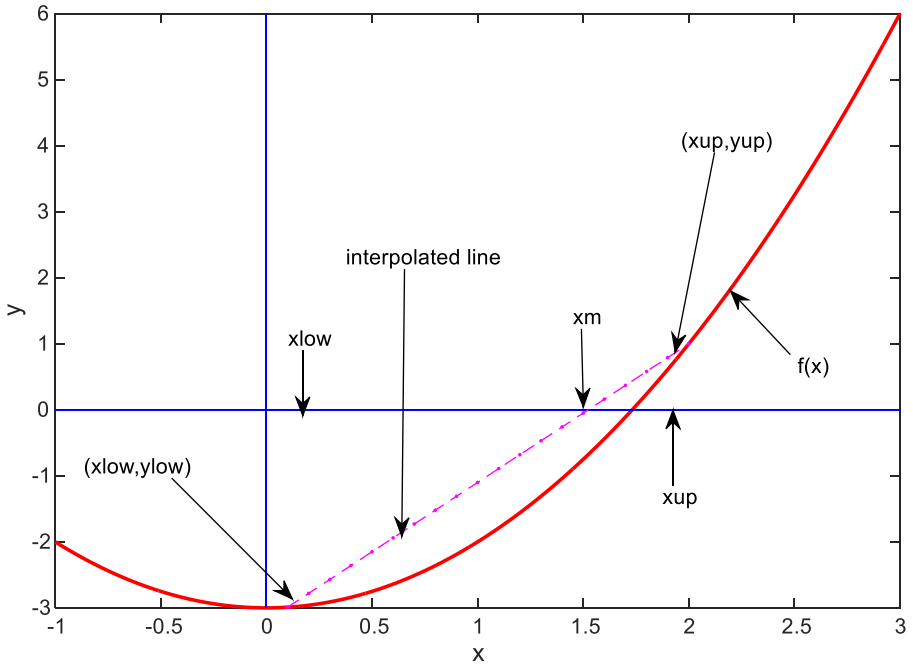

Fig. 4

Steps:

1. Give the following as inputs: two initial guesses, x_{low} and x_{up}, and a tolerance on y_m , ε. (Note that, tolerance is on y now. Why?).
2. Check if the root is in this interval by computing $y_{low} = f(x_{low})$ and $y_{up} = f(x_{up})$, and finally calculating $y_{low} * y_{up}$.
   If $y_{low} * y_{up} > 0$, the root is most likely not in this interval and so quit saying that this method is not applicable.
   Else if $y_{low} * y_{up} < 0$, at least one root is in the interval, and proceed as follows.
3. The new point is calculated between x_{low} and x_{up}. Let's call it x_m.
$$x_m = x_{up} - \frac{f(x_{up})(x_{low} - x_{up})}{f(x_{low}) - f(x_{up})}$$
4. Check if $y_m = f(x_m)$ is already less than or equal to ε. If yes, set x_m to be root and stop. If not, then there are two possible cases:
   (c) If $y_m * y_{low} < 0$, then the root is in the left half of the interval and we should update x_{up} to x_m.
   (d) If $y_m * y_{low} > 0$, then the root is in the right half of the interval and we should update x_{low} to x_m.
5. Continue steps 3 through 4 until $y_m = f(x_m)$ has been reduced to a value $\leq \varepsilon$.

The next page illustrates a step-by-step calculation for the false position method.

False Position Method

Consider the following equation: $y = f(x) = x^2 - 3$

And the tolerance of $f(root)$ or y_m: 0.001 and $x_{low} = 1, x_{up} = 2$

The root is indeed in this interval, as $y_{up} = f(x_{up}) = 1$ and $y_{low} = f(x_{low}) = -2$, and hence, $y_{low} * y_{up} < 0$

$$x_m = x_{up} - \frac{f(x_{up})(x_{low} - x_{up})}{f(x_{low}) - f(x_{up})}$$

x_{low}	x_{up}	$x_{up} - x_{low}$	$y_{low} = f(x_{low})$	$y_{up} = f(x_{up})$	x_m	$y_m = f(x_m)$	$\lvert y_m \rvert$	$y_m * y_{low}$	Need to Update
1	2	1	-2	1	1.6667	-0.2222	0.2222>0.001	>0	x_{low}
1.6667	2.0000	0.3333	-0.2222	1.0000	1.7273	-0.0165	0.0165>0.001	>0	x_{low}
1.7273	2.0000	0.2727	-0.0165	1.0000	1.7317	-0.0012	0.0012>0.001	>0	x_{low}
1.7317	2.0000	0.2683	-0.0012	1.0000	1.7320	-0.0001	0.0001<0.001	>0	x_{low}
1.7320	2.0000	0.2680	-	-	-	-	-	-	-

The root in this method is the last value of $x_m = 1.7320$. 4 iterations are required to reach this value. A lower value of ε would yield a better approximation of root, but would require more iterations.

Note that, the interval $(x_{up} - x_{low})$ does not reduce as shown in bisection method. This shows why we did not use tolerance on interval, rather chose to apply tolerance on y_m. Also tolerance is on absolute value of y_m, because we can reach to 0 from both negative values and positive values.

*The values that are updated are bold-faced.

Consider the following code for false position method to do the same using MATLAB:

```
clc; clear all; close all;
f=inline('x^2-3'); %function declared as a numerical function
tol=0.001;%value of tolerance on f(root)or ym
%two initial guesses
xlow=1;
xup=2;
ylow=f(xlow);
yup=f(xup);
%checking whether a root is in this interval
if ylow*yup>0
    disp('Root is not likely in this interval');
else disp('Root is in this interval');
    xm=xup-f(xup)*(xlow-xup)/(f(xlow)-f(xup));%calculating xm
    ym=f(xm);
    iter=1; %calculating iterations
    while (abs(ym)>=tol) %assuming xup>xlow
        ylow=f(xlow);
        yup=f(xup);
        if ym*ylow>0
            xlow=xm;
        else xup=xm;
        end
        xm=xup-f(xup)*(xlow-xup)/(f(xlow)-f(xup));%calculating xm
        ym=f(xm);
        iter=iter+1;
    end
    root=xm;
    disp(root);
    disp(iter);
end
```

2 Newton Raphson Method

If $f(x)$, $f'(x)$ and $f''(x)$ are continuous near a root x, then this extra information regarding the nature of $f(x)$ can be used to develop algorithms that will produce a root faster than bisection or false position method. Newton-Raphson method relies on the continuity of $f'(x)$ and $f''(x)$.

This method attempts to find the root by repeatedly approximating $f(x)$ with a linear function at each step. If the initial guess at root is x_k, a tangent can be extended from the point $(x_k, f(x_k))$. The point where the tangent crosses the $x-axis$ usually represents an improved estimate of the root. At the point where the tangent touches the curve, the slope of the curve and the slope of the tangent is the same. So, from **Fig. 5**,

$$\frac{f(x_k) - 0}{x_k - x_{k+1}} = f'(x_k)$$

Rearranging this, we find: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} = x_k + dx$

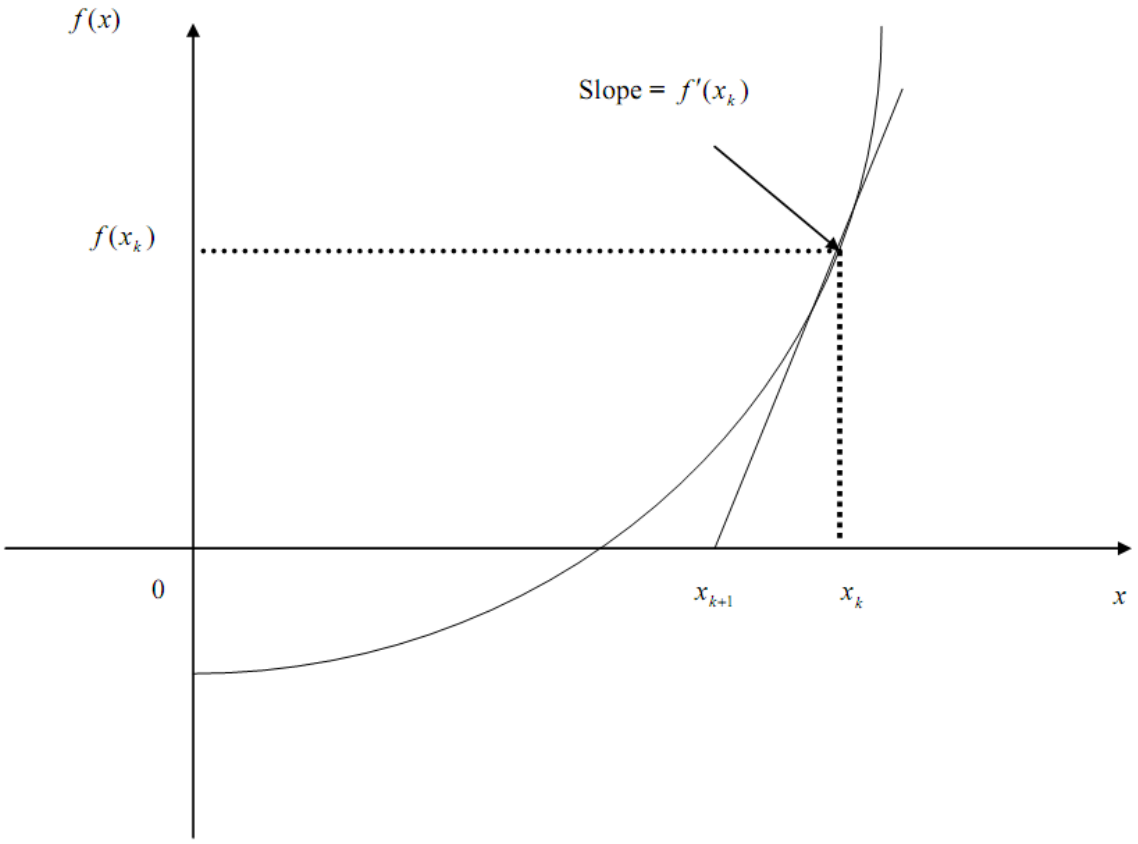

Fig. 5

Steps:

1. Give the following as inputs: an initial guesses, x_k, and a tolerance on $y_k = f(x_k)$, ε. (Note that, you may give a tolerance on dx, too, as it decreases in each iteration). You have to know both the original function, $f(x)$ and its first derivative, $f'(x)$.
2. Check if $y_k = f(x_k)$ is close to zero, i.e. check if $y_k \leq \varepsilon$. If not, calculate a new approximation of root, x_{k+1}:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

3. Go to step 2 and repeat until the solution is reached.

The next page illustrates a step-by-step calculation for the Newton-Raphson method.

Newton-Raphson Method

Consider the following equation: $y = f(x) = x^2 - 3$

The first derivative of the function is, $f'(x) = 2x$

And the tolerance of $f(root)$ or y_k: 0.001 and $x_k = 2$

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

x_k	$y_k = f(x_k)$	$\lvert y_k \rvert$	**Need to Calculate x_{k+1}?**	$f'(x_k)$	x_{k+1}	$dx = x_{k+1} - x_k$
2	1	1	Yes	4	1.75	-0.25
1.75	0.0625	0.0625	Yes	3.5	1.7321	-0.0179
1.7321	0.00017	0.00017<0.001	No	-	-	-

The root in this method is the last value of $x_k = 1.7321$. 2 iterations are required to reach this value. A lower value of ε would yield a better approximation of root, but would require more iterations.

Note that, the interval ($dx = x_{k+1} - x_k$) reduces in each step, too. So, a tolerance may be set on that, too.

*If $f'(x_k)$ is zero at some point, x_k, we would be in trouble. Why?

Consider the following code for Newton-Raphson method to do the same using MATLAB:

```
clc;clear all;close all;
syms x; %declaring x as a symbol
fs=x^2-3;%f is a symbolic function
f=matlabFunction(fs);%converting f to numerical function
fderiv=matlabFunction(diff(fs));%differentiating f as symbolic
function and
%converting symbolic function to numerical function
xk=2;%initial guess of root, xk
tol=0.001;%tolerance on f(root)=yk
iter=0;%number of iterations
while abs(f(xk))> tol %check if root has been reached
    xkp1=xk-f(xk)/fderiv(xk);%calculating new approximation of
root
    xk=xkp1;%updating xk
    iter=iter+1;
end
root=xk;
disp(root)
disp(iter)
```

3 Secant Method

The Newton-Raphson method requires two function evaluations per iteration, $f(x)$ and $f'(x)$. The calculation of a derivative could involve considerable effort and many functions have non-elementary forms, such as integrals, sums etc. It is desirable to have a method for finding a root that does not depend on the computation of a derivative. The secant method does not need a formula for a derivative and it can be coded so that only one new function evaluation is required per iteration.

The secant method requires two initial guesses x_{k-1} and x_k. The points $(x_{k-1}, f(x_{k-1}))$ and $(x_k, f(x_k))$ both lie on the graph of $f(x)$. Connecting these two points gives the secant line,

$$y - y_k = \frac{y_{k-1} - y_k}{x_{k-1} - x_k}(x - x_k)$$

Since we want $f(x) = 0$, we set $y = 0$, solve for x, and use that as our next approximation. So,

$$x_{k+1} = x_k - \frac{x_{k-1} - x_k}{y_{k-1} - y_k} y_k$$

Note that, the formula for secant method and false position method are the same, but the logical decisions are different. Also note that, in secant method, the initial guesses need not bracket the root. **Fig. 6** illustrates this.

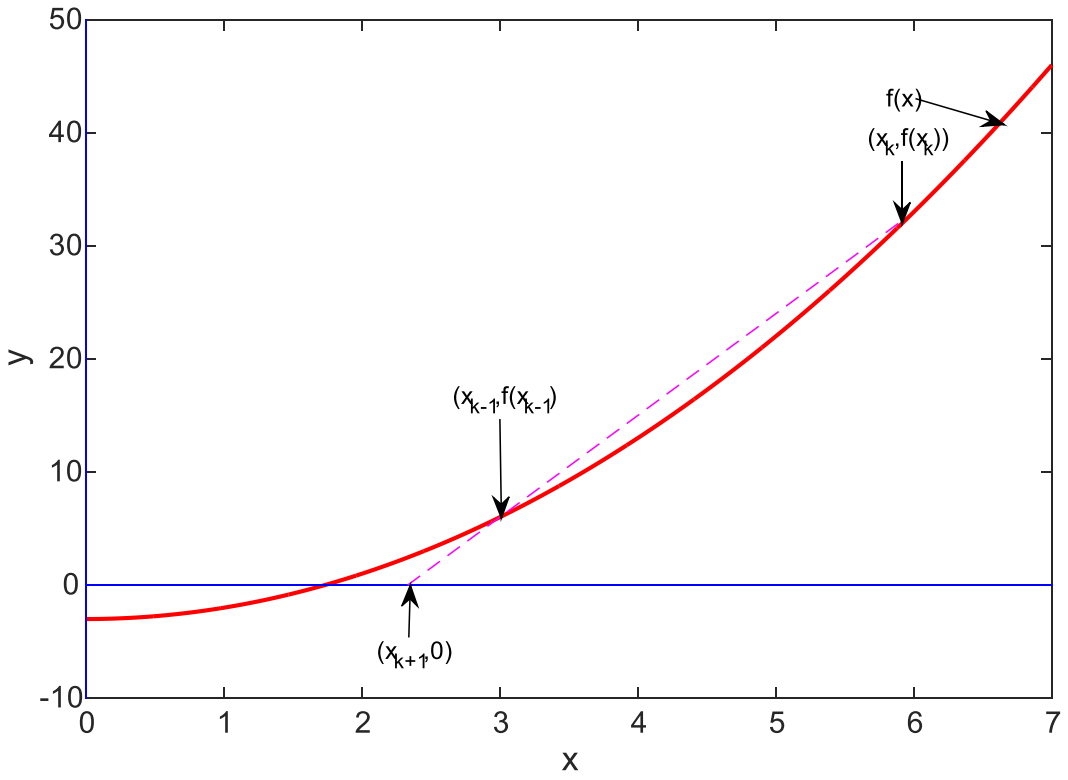

Fig.6

Steps:

1. Give the following as inputs: two initial guesses, x_k, and x_{k-1} and a tolerance on $y_k = f(x_k)$, ε. (Note that, you may give a tolerance on $dx = x_k - x_{k-1}$, too, as it decreases in each iteration).
2. Check if $y_k = f(x_k)$ is close to zero, i.e. check if $y_k \leq \varepsilon$. If not, calculate a new approximation of root, x_{k+1}:

$$x_{k+1} = x_k - \frac{x_{k-1} - x_k}{y_{k-1} - y_k} y_k$$

3. Make new guesses for next iteration, by setting:
   $x_{k-1} = x_k$ and $x_k = x_{k+1}$
4. Go to step 2 and repeat until the solution is reached.

The next page illustrates a step-by-step calculation for the Secant method.

Secant Method

Consider the following equation: $y = f(x) = x^2 - 3$

And the tolerance of $f(root)$ or y_k: 0.001 and $x_{k-1} = 3$ and $x_k = 2$

$$x_{k+1} = x_k - \frac{x_{k-1} - x_k}{y_{k-1} - y_k} y_k$$

x_{k-1}	x_k	$dx = x_k - x_{k-1}$	$y_{k-1} = f(x_{k-1})$	$y_k = f(x_k)$	$\lvert y_k \rvert$	Need to Calculate x_{k+1}?	x_{k+1}
3	2	-1	6	1	1	Yes	1.8
2	1.8	-0.2	1	0.24	0.24	Yes	1.7368
1.8	1.7368	-0.0632	0.24	0.0164	0.0164	Yes	1.7321
1.7368	1.7321	-0.0047	0.0164	0.00017	0.00017 <0.001	No	-

The root in this method is the last value of $x_k = 1.7321$. 3 iterations are required to reach this value. A lower value of ε would yield a better approximation of root, but would require more iterations.

Note that, the interval ($dx = x_k - x_{k-1}$) reduces in each step, too. So, a tolerance may be set on that, too.

Consider the following code for Secant method to do the same using MATLAB:

```
clc;clear all;close all;
xk=2; %initial guess1
xkminus1=3;%initial guess2
tol=0.001; %tolerance on f(root)=yk
f=@(x) x^2-3; %anonymous function
yk=f(xk);%evaluating yk
ykminus1=f(xkminus1);%evaluating ykminus1
iter=0;%number of iterations
while abs(yk)> tol%checking if a solution has been reached
    xkplus1=xk-(xkminus1-xk)/(ykminus1-yk)*yk;%new approximation
    ykplus1=f(xkplus1);
    %creating new pair of guesses
    xkminus1=xk;
    ykminus1=yk;
    xk=xkplus1;
    yk=ykplus1;
    iter=iter+1;
end
root=xk;
disp(iter);
disp(root)
```