Solving
Nonlinear Equations

An important problem in applied mathematics is to

$$solve \;\; f(x) = 0$$

where $f(x)$ is a function of x. The values of x that make $f(x) = 0$ are called the **roots** of the equation. They are also called the **zeros** of $f(x)$.

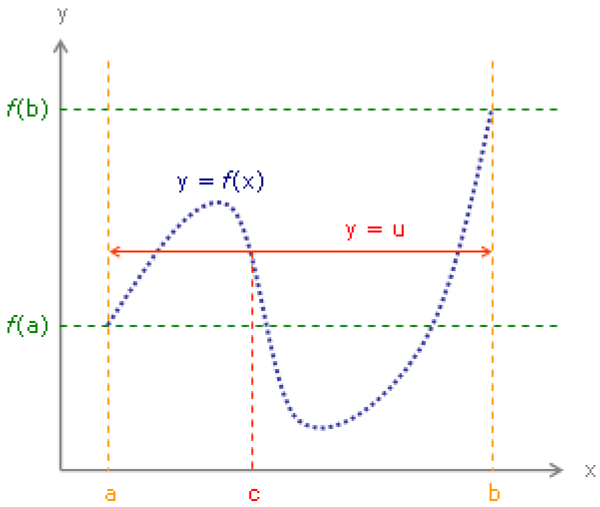
Related Methods

- **Interval Halving (Bisection) Method**
- **Newton's Method**
- **Secant Method**
- **Linear Interpolation Methods (Regula Falsi)**
- **Muller's Method**
- **Fixed-Point Iteration Method**
- **Other Methods**
- **Nonlinear Systems**

Interval Halving (Bisection) Method

Intermediate Value Theorem:

If the function $y = f(x)$ is continuous on the interval $[a,b]$ and u is any number between $f(a)$ and, $f(b)$, then there is a number c, with $c \in (a,b)$, such that $f(c) = u$.



Interval halving (bisection), an ***ancient*** but ***effective*** method for finding a zero of $f(x)$, is an excellent introduction to numerical methods. It begins with two values for x that bracket a root. It determines that they do in fact bracket a root because the function $f(x)$ changes signs at these two x-values and, if $f(x)$ is continuous, there must be at least one root between the values.

The bisection method then successively divides the initial interval in half, finds in which half the root(s) must lie, and repeats with the endpoints of the smaller interval. The test to see that $f(x)$ does change sign between points a and b is to see if

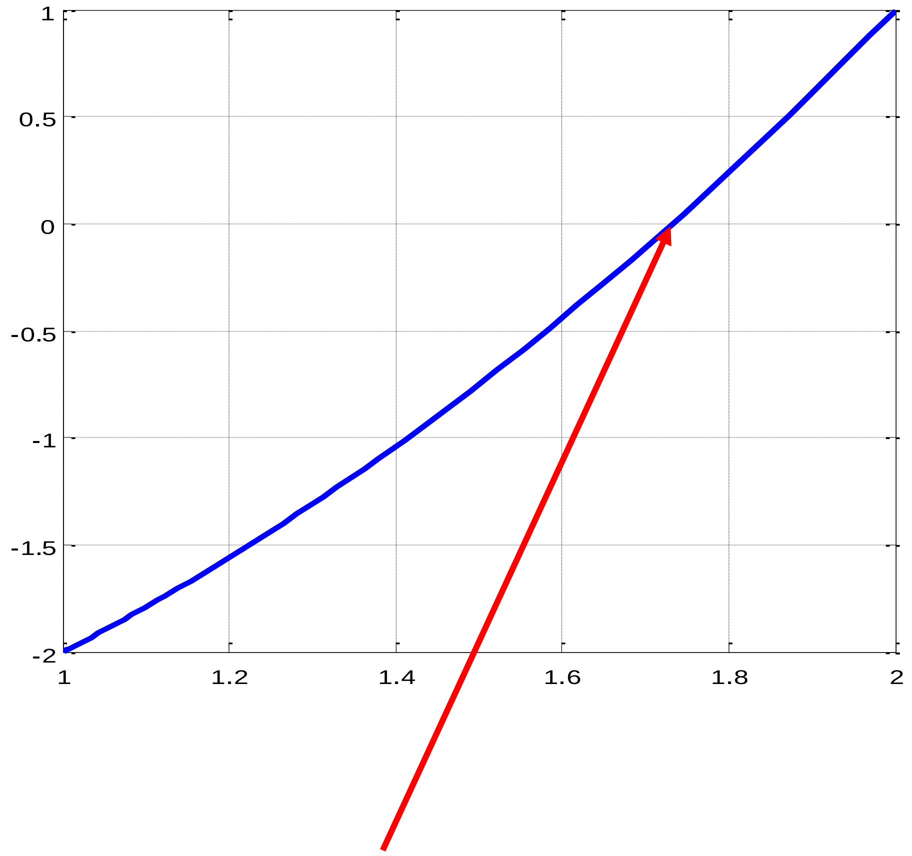
$$f(a) * f(b) < 0.$$

Example: Find a numerical approximation to $\sqrt{3}$.

We approximate the zero of

$$y = f(x) = x^2 - 3.$$

```
f=inline('x^2-3')
f =
    Inline function:
    f(x) = x^2-3
>> fplot(f,[1 2]);grid on
```



Since $f(1) = -2$ and $f(2) = 1$ we take as our starting bounds on the zero a=1 and b=2.

Our first approximation to the zero is the mid point of the interval, namely $x_1 = \frac{(1+2)}{2} = 1.5$. The value of the function $y_1 = f(x_1) = (1.5)^2 - 3 = -0.75.$ Since y(a) and y_1 have the same sign, but y_1 and y(b) have opposite signs, we know that there is a zero in the interval $[x_1,b]$.

The next Table shows the results of five iterations.

Step	a	b	x_i	y(a)	y(b)	y_i
1.	1.0000	2.0000	1.5000	-2.0000	1.0000	-0.7500
2.	1.5000	2.0000	1.7500	-0.7500	1.0000	0.0625
3.	1.5000	1.7500	1.6250	-0.7500	0.0625	-0.3594
4.	1.6250	1.7500	1.6875	-0.3594	0.0625	-0.1523
5.	1.6875	1.7500	1.7188	-0.1523	0.0625	-0.0459

An algorithm for Halving the Interval (Bisection)

To determine a root of f(x) = 0 that is accurate within a specified tolerance value(TOL), given values x_1 and x_2 such that $f(x_1)*f(x_2) < 0$.

Repeat

Set $x_3= (x_1+x_2)/2$

If $f(x_3)*f(x_1) <0$ Then

Set $x_2=x_3$

Else Set $x_1=x_3$ End if.

Until ($|x_1-x_2|$)<2*TOL).

The final value of x_3 approximates the root, and it is an error by not more than $|x_1-x_2|/2$.

Note : The method may produce a false root if f(x) is discontinuous on $[x_1,x_2]$.

Stopping Procedures

Select a tolerance ε>0 and generate $x_1,x_2,\ldots, x_n$ until one of the following conditions is met:

$$|x_n - x_{n-1}| < \varepsilon$$

$$\frac{|x_n - x_{n-1}|}{|x_n|} < \varepsilon, x_n \neq 0$$

$$|f(x_n)| < \varepsilon$$

MATLAB M-File

```
function rtn = bisec(fx, xa, xb, n)
% bisec does n bisections to approximate
%a root of fx
x=xa;
fa =eval(fx);
x=xb;
fb =eval(fx);
for i=1:n
       xc = (xa+xb)/2 ; x = xc; fc= eval(fx);
       X=[i, xa, xb, xc, fc];
       disp (X)
       if fc*fa <0
           xb=xc;
       else xa =xc;
       end % of if/else
end % of for loop
```

which we save with the name 'bisec.m'.

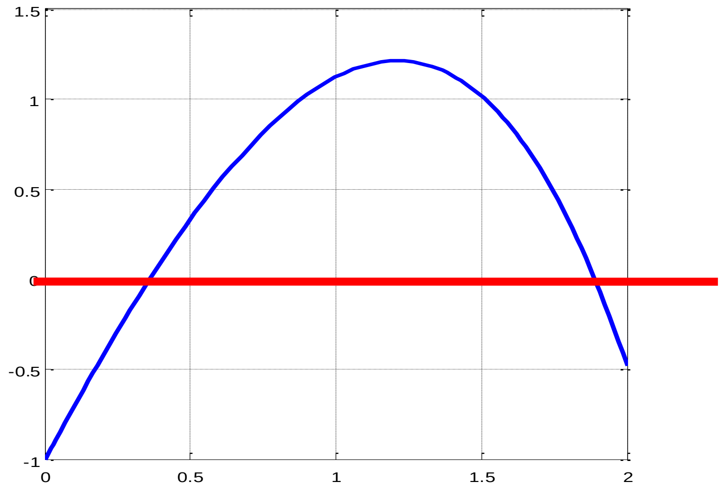
There is no stopping condition in the M-file

```
>> fx='x^2-3'
fx =
x^2-3
>>
>> bisec(fx,1,2,5)
    1.0000    1.0000    2.0000    1.5000   -0.7500
    2.0000    1.5000    2.0000    1.7500    0.0625
    3.0000    1.5000    1.7500    1.6250   -0.3594
    4.0000    1.6250    1.7500    1.6875   -0.1523
    5.0000    1.6875    1.7500    1.7188   -0.0459
bisec(fx,1,2,15)
    1.0000    1.0000    2.0000    1.5000   -0.7500
    2.0000    1.5000    2.0000    1.7500    0.0625
    3.0000    1.5000    1.7500    1.6250   -0.3594
    4.0000    1.6250    1.7500    1.6875   -0.1523
    5.0000    1.6875    1.7500    1.7188   -0.0459
    6.0000    1.7188    1.7500    1.7344    0.0081
    7.0000    1.7188    1.7344    1.7266   -0.0190
    8.0000    1.7266    1.7344    1.7305   -0.0055
    9.0000    1.7305    1.7344    1.7324    0.0013
   10.0000    1.7305    1.7324    1.7314   -0.0021
   11.0000    1.7314    1.7324    1.7319   -0.0004
   12.0000    1.7319    1.7324    1.7322    0.0004
   13.0000    1.7319    1.7322    1.7321    0.0000
   14.0000    1.7319    1.7321    1.7320   -0.0002
   15.0000    1.7320    1.7321    1.7320   -0.0001
```

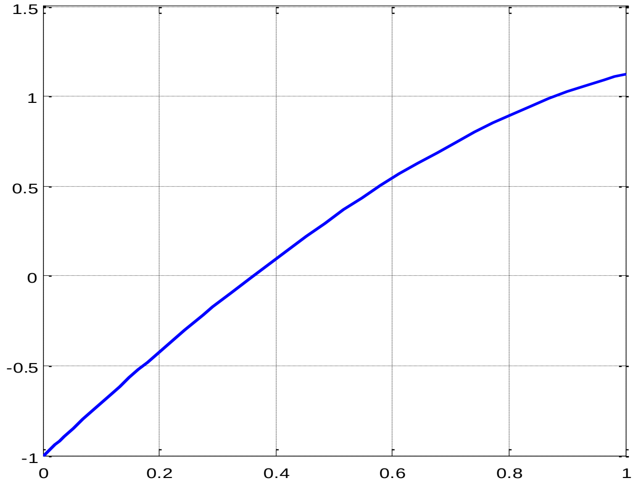
Example:

$$f(x) = 3x + \sin(x) - e^x$$

```
>> f=inline('3*x+sin(x)-exp(x)')
f =
     Inline function:
     f(x) = 3*x+sin(x)-exp(x)
>> fplot(f,[0 2]);grid on
>>
```



And we see this figure that indicates there are zeros at <u>about x=0.35 and 1.9.</u>



To obtain the true value for the root, which is needed to compute actual error, we again used MATLAB:

>> solve('3*x+sin(x)-exp(x)')

ans =0.36042170296032440136932951583028

which is really more accurate than we need.

Iteration	X_1	X_2	X_3	$F(X_3)$	Maximum Error	Actual Error
1.	0	1.0000	0.5000	0.3307	0.50000	0.13958
2.	0	0.5000	0.2500	-0.2866	0.25000	-0.11042
3.	0.2500	0.5000	0.3750	0.0363		
4.	0.2500	0.3750	0.3125	-0.1219		
5.	0.3125	0.3750	0.3438	-0.0420		
6.	0.3438	0.3750	0.3594	-0.0026		
7.	0.3594	0.3750	0.3672	0.0169		
8.	0.3594	0.3672	0.3633	0.0071		
9.	0.3594	0.3633	0.3613	0.0023		
10.	0.3594	0.3613	0.3604	-0.0002		
11.	0.3604	0.3613	0.3608	0.0010		
12.	0.3604	0.3608	0.3606	0.0004	0.00024	0.00017
13.	0.3604	0.3606	0.3605	0.0001	0.00012	0.00005

```
fx='3*x + sin(x) -exp(x)'
fx =
3*x + sin(x) -exp(x)
>> bisec(fx,0,1,13)
    1.0000         0    1.0000    0.5000    0.3307
    2.0000         0    0.5000    0.2500   -0.2866
    3.0000    0.2500    0.5000    0.3750    0.0363
    4.0000    0.2500    0.3750    0.3125   -0.1219
    5.0000    0.3125    0.3750    0.3438   -0.0420
    6.0000    0.3438    0.3750    0.3594   -0.0026
    7.0000    0.3594    0.3750    0.3672    0.0169
    8.0000    0.3594    0.3672    0.3633    0.0071
    9.0000    0.3594    0.3633    0.3613    0.0023
   10.0000    0.3594    0.3613    0.3604   -0.0002
   11.0000    0.3604    0.3613    0.3608    0.0010
   12.0000    0.3604    0.3608    0.3606    0.0004
   13.0000    0.3604    0.3606    0.3605    0.0001
```

With different parameters;

```
>> bisec(fx,0.35,0.4,20)
    1.0000    0.3500    0.4000    0.3750    0.0363
    2.0000    0.3500    0.3750    0.3625    0.0052
    3.0000    0.3500    0.3625    0.3562   -0.0105
    4.0000    0.3562    0.3625    0.3594   -0.0026
    5.0000    0.3594    0.3625    0.3609    0.0013
    6.0000    0.3594    0.3609    0.3602   -0.0007
    7.0000    0.3602    0.3609    0.3605    0.0003
    8.0000    0.3602    0.3605    0.3604   -0.0002
    9.0000    0.3604    0.3605    0.3604    0.0001
   10.0000    0.3604    0.3604    0.3604   -0.0001
   11.0000    0.3604    0.3604    0.3604    0.0000
   12.0000    0.3604    0.3604    0.3604   -0.0000
   13.0000    0.3604    0.3604    0.3604   -0.0000
   14.0000    0.3604    0.3604    0.3604    0.0000
   15.0000    0.3604    0.3604    0.3604   -0.0000
   16.0000    0.3604    0.3604    0.3604   -0.0000
   17.0000    0.3604    0.3604    0.3604   -0.0000
   18.0000    0.3604    0.3604    0.3604   -0.0000
   19.0000    0.3604    0.3604    0.3604   -0.0000
   20.0000    0.3604    0.3604    0.3604    0.0000
```

format long

\>> bisec(fx,0.35,0.40,20)

1.00000000000000	0.35000000000000	0.40000000000000	0.37500000000000	0.03628111446785
2.00000000000000	0.35000000000000	0.37500000000000	0.36250000000000	0.00519565105452
3.00000000000000	0.35000000000000	0.36250000000000	0.35625000000000	-0.01045234328072
4.00000000000000	0.35625000000000	0.36250000000000	0.35937500000000	-0.00261963457026
5.00000000000000	0.35937500000000	0.36250000000000	0.36093750000000	0.00129019064630
6.00000000000000	0.35937500000000	0.36093750000000	0.36015625000000	-0.00066417692600
7.00000000000000	0.36015625000000	0.36093750000000	0.36054687500000	0.00031314318976
8.00000000000000	0.36015625000000	0.36054687500000	0.36035156250000	-0.00017548279455
9.00000000000000	0.36035156250000	0.36054687500000	0.36044921875000	0.00006883871710
10.00000000000000	0.36035156250000	0.36044921875000	0.36040039062500	-0.00005331990898
11.00000000000000	0.36040039062500	0.36044921875000	0.36042480468750	0.00000775993651
12.00000000000000	0.36040039062500	0.36042480468750	0.36041259765625	-0.00002277985313
13.00000000000000	0.36041259765625	0.36042480468750	0.36041870117188	-0.00000750992503
14.00000000000000	0.36041870117188	0.36042480468750	0.36042175292969	0.00000012501406
15.00000000000000	0.36041870117188	0.36042175292969	0.36042022705078	-0.00000369245340
16.00000000000000	0.36042022705078	0.36042175292969	0.36042098999023	-0.00000178371915
17.00000000000000	0.36042098999023	0.36042175292969	0.36042137145996	-0.00000082935242
18.00000000000000	0.36042137145996	0.36042175292969	0.36042156219482	-0.00000035216915
19.00000000000000	0.36042156219482	0.36042175292969	0.36042165756226	-0.00000011357753
20.00000000000000	0.36042165756226	0.36042175292969	0.36042170524597	0.00000000571826

- **The <u>main advantage</u> of interval halving is that it is guaranteed to work if f(x) is continuous in [a, b] and if the values x=a and x=b actually bracket a root.**
- **<u>Another important advantage</u> that few other root-finding methods share is that the *<u>number of iterations to achieve a specified accuracy is known in advance</u>*.**
- **Because the interval [a, b] is halved each time, we can say with surely that**

$$\text{error after n iteration} < \left|\frac{(b-a)}{2^n}\right|$$

- **The major objection of interval halving has been that it is *<u>slow to converge</u>*.**

The number of n of repeated bisections needed to guarantee that the nth midpoint is an approximation to a zero and has an error less than the preassigned value δ is

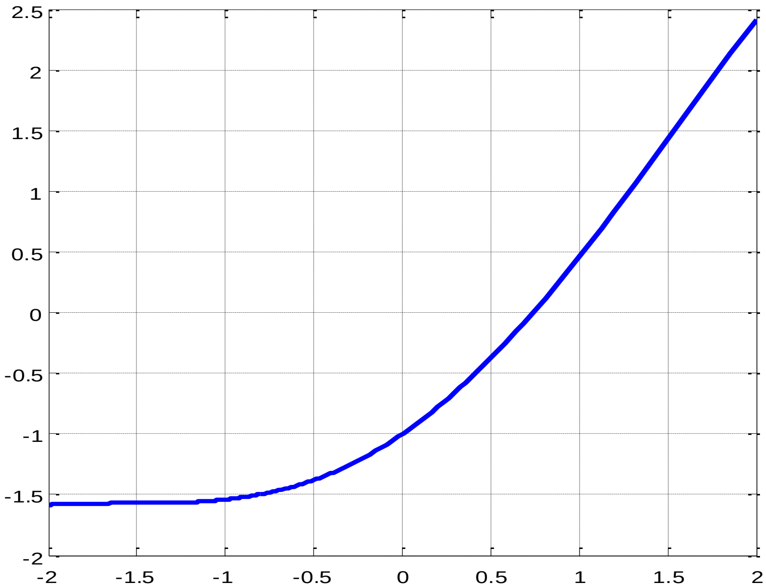
$$n = \text{int}\left(\frac{\ln(b-a)-\ln(\delta)}{\ln(2)}\right)$$

Example:

$$f(x) = x - \cos(x)$$

It is a good plan to look at a plot of the function to learn where the function crosses the x-axis. MATLAB can do it for us:

```
>> f=inline('x-cos(x)')
f =
    Inline function:
    f(x) = x-cos(x)
>> fplot(f,[-2,2]);grid on
```




```
>> solve('x-cos(x)')
 ans =
 .73908513321516064165531208767387
```

MATLAB M-File (Another file)

```
function [c,err,yc]=bisect(f,a,b,delta)
%INPUT f is the function input as string 'f'
%       a and b are the left and right endpoints
%       delta is the tolerance, n number of
iterations is computed using
%       delta and  a and b.
%OUTPUT     c is the zero
%           yc=f(c)   function value
%           err is the error estimate for c
ya=feval(f,a);
yb=feval(f,b);
if ya*yb>0 return, end
n=1+round((log(b-a)-log(delta))/log(2));
for k=1:n
    c=(a+b)/2;
    yc=feval(f,c);
    if yc==0
        a=c;
        b=c;
    elseif yb*yc>0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
    end
    if b-a < delta, break, end
end
c=(a+b)/2;
err=abs(b-a);
yc=feval(f,c);
```

which we save with the name 'bisect.m'. (before was bisec.m)

If we enter these commands

```
>> bisect(f,0,1,0.0000001)
ans =
   0.7391
```

Compare MATLAB M-Files

Bisec.m

Bisect.m

If we add

```
X=[k,a,b,c,yc]
 %       disp(X)
```

in bisect1 we can also display X values.

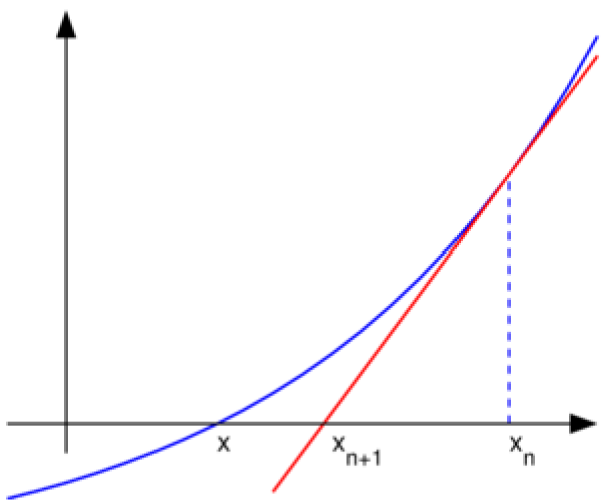
MATLAB M-File

```
function [c,err,yc]=bisect2(f,a,b,delta)
%INPUT f is the function input as string 'f'
%       a and b are the left and right endpoints
%       delta is the tolerance, n number of
iterations is computed using
%       delta and  a and b.
%OUTPUT    c is the zero
%          yc=f(c)  function value
%          err is the error estimate for c
ya=feval(f,a);
yb=feval(f,b);
if ya*yb>0 return, end
n=1+round((log(b-a)-log(delta))/log(2));
for k=1:n
    c=(a+b)/2;
    yc=feval(f,c);
    X=[k,a,b,c,yc]
 %       disp(X)
    if yc==0
        a=c;
        b=c;
    elseif yb*yc>0
        b=c;
        yb=yc;
    else
        a=c;
        ya=yc;
            end
    if b-a < delta, break, end
end
c=(a+b)/2;
err=abs(b-a);
yc=feval(f,c);
```

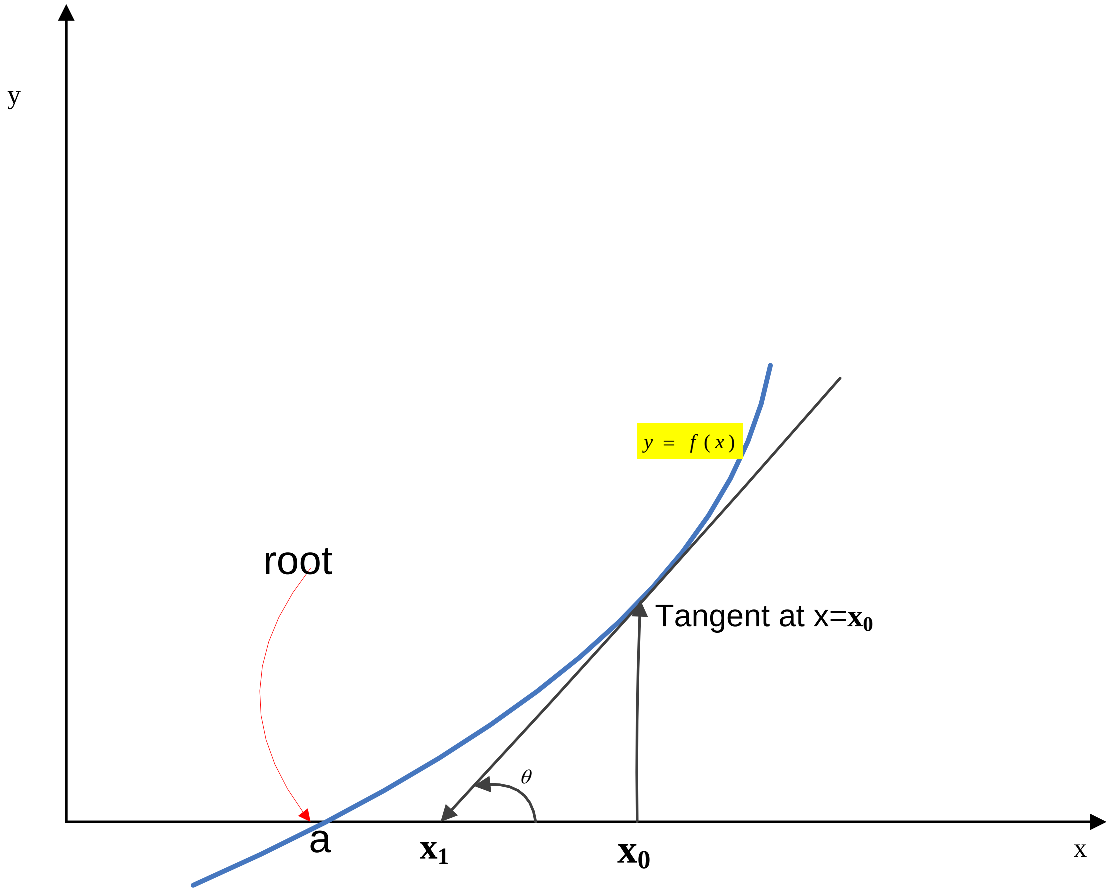
If we enter these commands

```
bisect2(f,0,1,0.0001)
X =
    1.0000         0    1.0000    0.5000   -0.3776

    2.0000    0.5000    1.0000    0.7500    0.0183
    3.0000    0.5000    0.7500    0.6250   -0.1860
    4.0000    0.6250    0.7500    0.6875   -0.0853
    5.0000    0.6875    0.7500    0.7188   -0.0339
    6.0000    0.7188    0.7500    0.7344   -0.0079
    7.0000    0.7344    0.7500    0.7422    0.0052
    8.0000    0.7344    0.7422    0.7383   -0.0013
    9.0000    0.7383    0.7422    0.7402    0.0019
   10.0000    0.7383    0.7402    0.7393    0.0003
   11.0000    0.7383    0.7393    0.7388   -0.0005
   12.0000    0.7388    0.7393    0.7390   -0.0001
   13.0000    0.7390    0.7393    0.7391    0.0001
   14.0000    0.7390    0.7391    0.7391   -0.0000
ans =
    0.7391
>>
```

Newton’s Method



One of the most widely used methods of solving equations is <u>Newton-Raphson</u> or simply Newton’s method.



From the figure, *a* is the point at which f(x) = 0 and x_0 is an estimate of a. The Newton method computes a new estimate x_1 in the following way. Construct the tangent to f(x) at $x=x_0$;

$$\tan(\theta) = \frac{f(x_0)}{x_0 - x_1}$$

But from the definition of derivative

$$\tan(\theta) = f'(x_0)$$

Thus

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1} \therefore x_0 - x_1 = \frac{f(x_0)}{f'(x_0)} ie.$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

We continue the calculation scheme by computing

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

Or, in more general terms,

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0,1,2\ldots$$

Newton's algorithm is widely used because, at least in the near neighborhood of a root, it is more rapidly convergent.

Consider the Taylor polynomial for f(x) expanded about $\bar{x}$**,**

$$f(x) = f(\bar{x}) + (x-\bar{x})f'(x-\bar{x}) + \frac{(x-\bar{x})^2}{2}f''(x-\bar{x}) + \ldots$$

Newton's method is derived by assuming that the term involving $(x-\bar{x})^2$ **is negligible and that**

$$0 \approx f(\bar{x}) + (x-\bar{x})f'(\bar{x})$$

Solving for x in this equation gives:

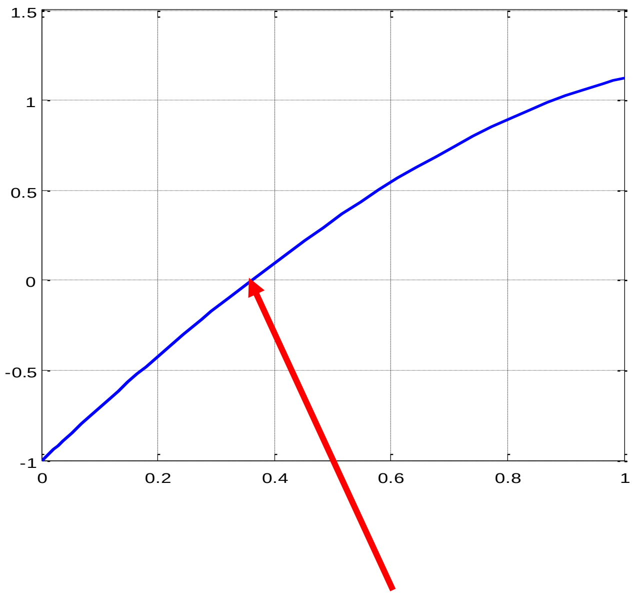
$$x \approx \bar{x} - \frac{f(\bar{x})}{f'(\bar{x})}$$

Example: **When Newton Raphson method is applied to $f(x)=3x+\sin(x)-e^{x}=0$ we have the following calculations**

$$f(x)=3x+\sin(x)-e^{x}$$
$$f'(x)=3+\cos(x)-e^{x}$$

There is little need to use MATLAB to get simple derivative, but, for practice here is how to do it:

```
>> f=inline('3*x+sin(x)-exp(x)')

f =     Inline function:

    f(x) = 3*x+sin(x)-exp(x)

>> fplot(f,[0 1]);grid on
```




```
>> fx='3*x+sin(x)-exp(x)'

fx =3*x+sin(x)-exp(x)

>> dfx=diff(fx)

 dfx = 3+cos(x)-exp(x)
```

If we begin x_0=0.0, we have

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 0.0 - \frac{-1.0}{3.0} = 0.33333;$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 0.33333 - \frac{-0.068418}{2.54934} = 0.36017;$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.36017 - \frac{-6.279.10^{-4}}{2.50226} = 0.3604217;$$

.

.

An algorithm for Newton Raphson Method

To determine a root of f(x) = 0, given x_0 reasonably close to the root,

Compute $f(x_0)$, $f^{'}(x_0)$

If ($f(x_0) \neq 0$ **and** $f^{'}(x_0) \neq 0$**) Then**

Repeat

Set $x_1 = x_0$

Set $x_0 = x_0 - \dfrac{f(x_0)}{f'(x_0)}$

Until $(|x_1 - x_0| < TOL - 1$ **or**

$(|f(x_0)| < TOL - 2$

End If.

Note: **The method may *converge* to a root *different from the* expected one *or diverge* if the starting value is not close enough to the root.**

MATLAB M-File

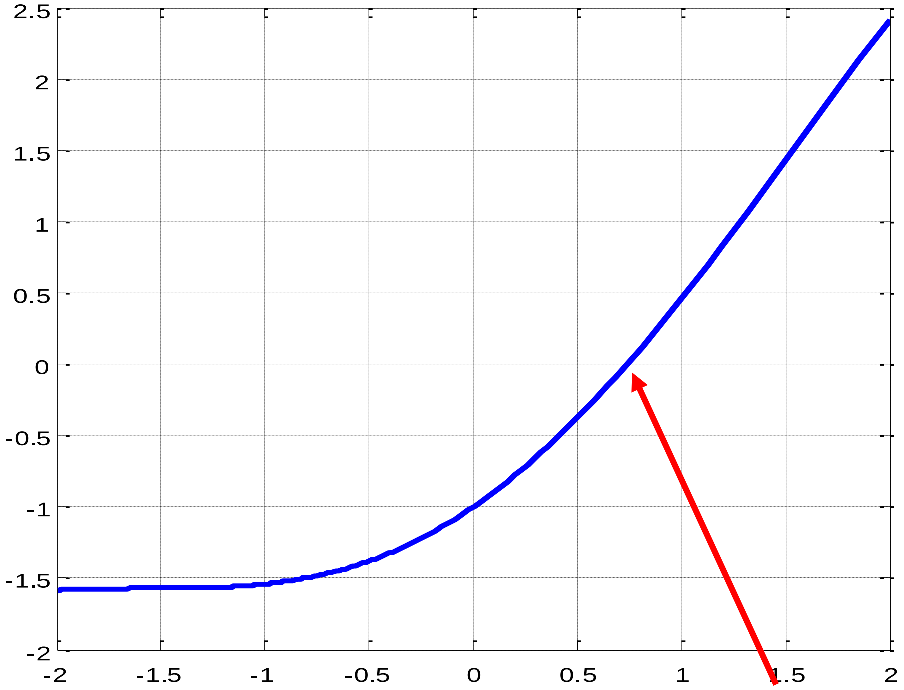
```
function
[x0,err,k,y]=Newton(f,df,x0,delta,epsilon,maxit)
%INPUT    f is object function input as a string 'f'
%      df is the derivative of f input as a string 'df'
%      x0 is the initial approximation to a zero of f
%          delta tolerance for x0
%          epsilon tolerance for function values y
%          maxit is the maximum number of iterations
%OUTPUT    result is the Newton's approx.to the zero
%          err  is the error  estimate for x0
%          k is the number of iterations
%          y is the function value f(result)
for k=1:maxit
    y=feval(f,x0);
    dy=feval(df,x0);
    x1=x0-(y/dy);
    err=abs(x1-x0);
    relerr=2*err/(abs(x1)+delta);
      X=[k,x0,y,dy]
    x0=x1;
    if
(err<delta)|(relerr<delta)|(abs(y)<epsilon),break,end
end
```

Use the MATLAB Command Window

```
>> f=inline('3*x+sin(x)-exp(x)')

f =     Inline function:

    f(x) = 3*x+sin(x)-exp(x)

>> df=inline('3+cos(x)-exp(x)')

df =     Inline function:

    df(x) = 3+cos(x)-exp(x)

>> newton(f,df,0,0.001,0.001,10)

X =     1     0    -1     3
X =    2.0000    0.3333   -0.0684    2.5493
X =    3.0000    0.3602   -0.0006    2.5023
ans =    0.3604
```

Example When Newton Raphson method is applied to $f(x) = x - \cos(x)$ we have the following calculations

$f'(x) = 1 + \sin(x)$




```
>>f=inline('x-cos(x)')
f =     Inline function:
     f(x) = x-cos(x)
>> df=diff('x-cos(x)')
 df = 1+sin(x)
 >> df=inline('1+sin(x)')
df =     Inline function:
     df(x) = 1+sin(x)
```

Suppose we choose x_0=0.75 in the example above.

The Newton Method gives:

```
>> newton(f,df,0.75,0.001,0.001,10)

X =    1.0000    0.7500    0.0183    1.6816

X =    2.0000    0.7391    0.0000    1.6736

ans =

    0.7391
```

Choose x_0=0.0

```
newton(f,df,0.0,0.001,0.001,10)

X =
     1     0    -1     1
X =    2.0000    1.0000    0.4597    1.8415
X =    3.0000    0.7504    0.0189    1.6819
X =    4.0000    0.7391    0.0000    1.6736
ans =
    0.7391
```

Convergence of Newton's Method

Newton's method

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0,1,2\ldots$$

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

Successive iterations will converge if $|g'(x)| < 1$, and, doing the differentiation, we see that the method converges if

$$|g'(x)| = \left| 1 - \frac{f'(x)f'(x) - f(x).f''(x)}{[f'(x)]^2} \right| = \left| \frac{f(x).f''(x)}{[f'(x)]^2} \right| < 1.$$

$$|g'(x)| = \left| \frac{f(x).f''(x)}{[f'(x)]^2} \right| < 1.$$

which requires that f(x) and its derivatives exist and be continuous. Newton's Method is shown to be quadratically convergent. (See: Applied Numerical Analysis, Gerald, C., WheatleyP.,p.59)

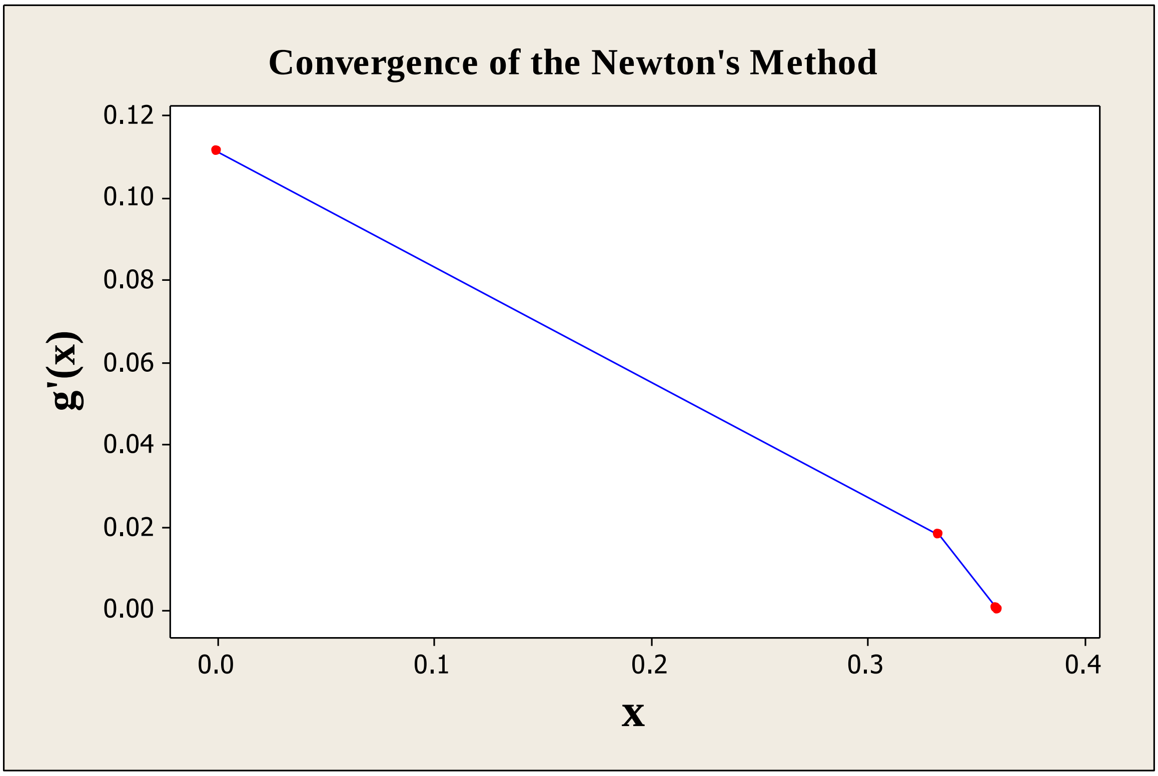
Example: (Convergence)

$$f(x) = 3x + \sin(x) - e^x$$

$$f'(x) = 3 + \cos(x) - e^x$$

$$f''(x) = -\sin(x) - e^x$$

i	x_i	$f(x_i)$	$f'(x_i)$	$f''(x_i)$	$\lvert g'(x_i) \rvert$
0	**0.000000**	**-1.00000**	**3.00000**	**-1.00000**	**0.111111**
1	**0.333333**	**-0.06842**	**2.54935**	**-1.72281**	**0.018136**
2	**0.360170**	**-0.00063**	**2.50226**	**-1.78601**	**0.000180**
3	**0.360422**	**-0.00000**	**2.50181**	**-1.78660**	**0.000000**



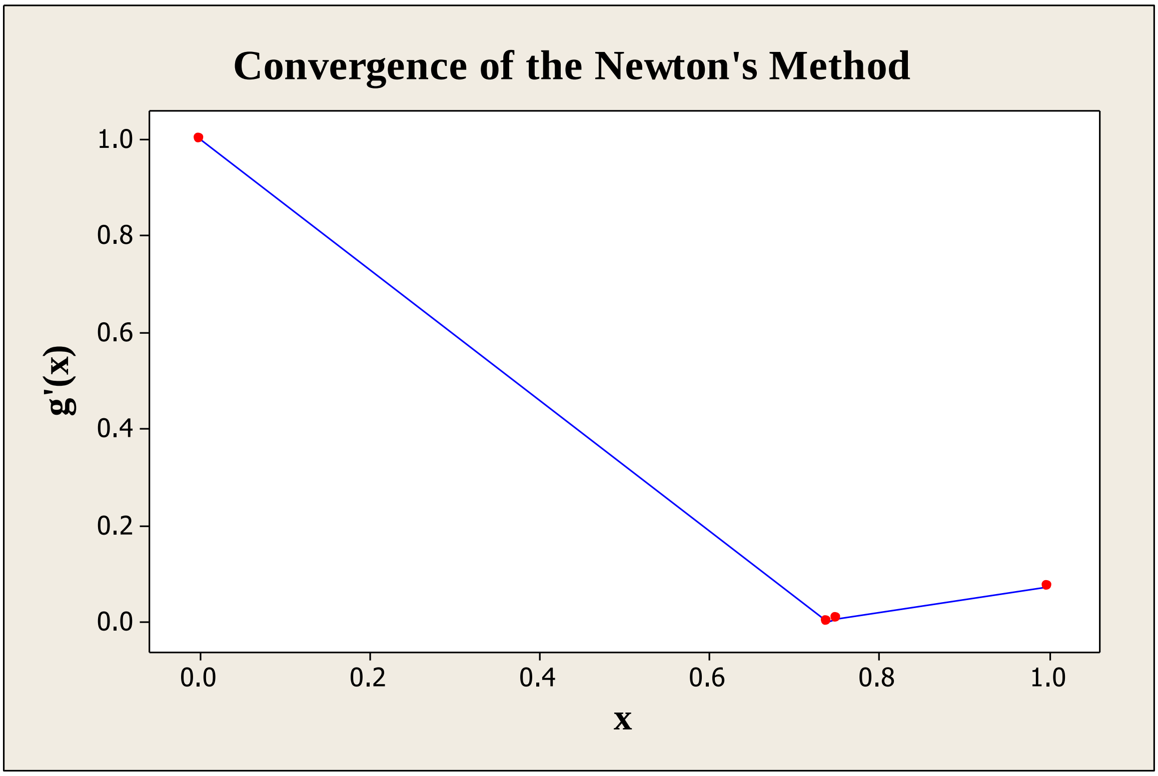
Example: (Convergence)

$$f(x) = x - \cos(x)$$

$$f'(x) = 1 + \sin(x)$$

$$f''(x) = \cos(x)$$

i	x_i	$f(x_i)$	$f'(x_i)$	$f''(x_i)$	$\lvert g'(x_i) \rvert$
0	**0.0000**	**-1.00000**	**1.00000**	**1.00000**	**1.00000**
1	**1.0000**	**0.45970**	**1.84147**	**0.54030**	**0.07325**
2	**0.7504**	**0.01898**	**1.68193**	**0.73142**	**0.00491**
3	**0.7391**	**0.00002**	**1.67362**	**0.73908**	**0.00001**



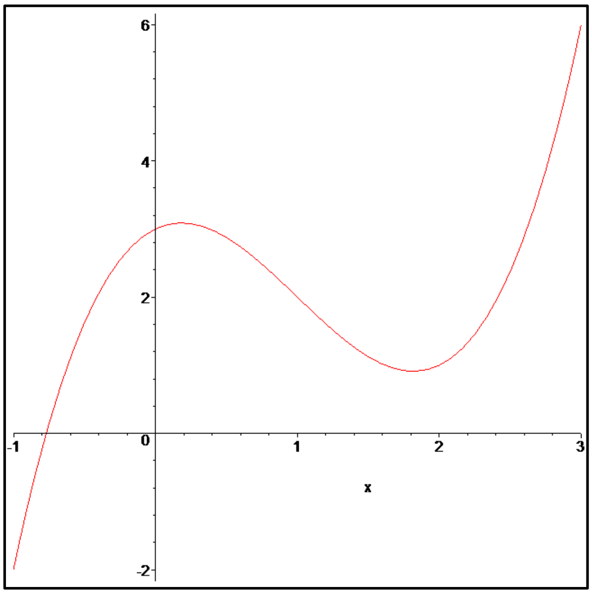
Oscillations in Newton's Method

Newton's method can give oscillatory results for some functions and some initial estimates.

Example:

Consider the cubic equation

$$f(x) = x^3 - 3x^2 + x + 3$$



The derivative is $f'(x) = 3x^2 - 6x + 1$

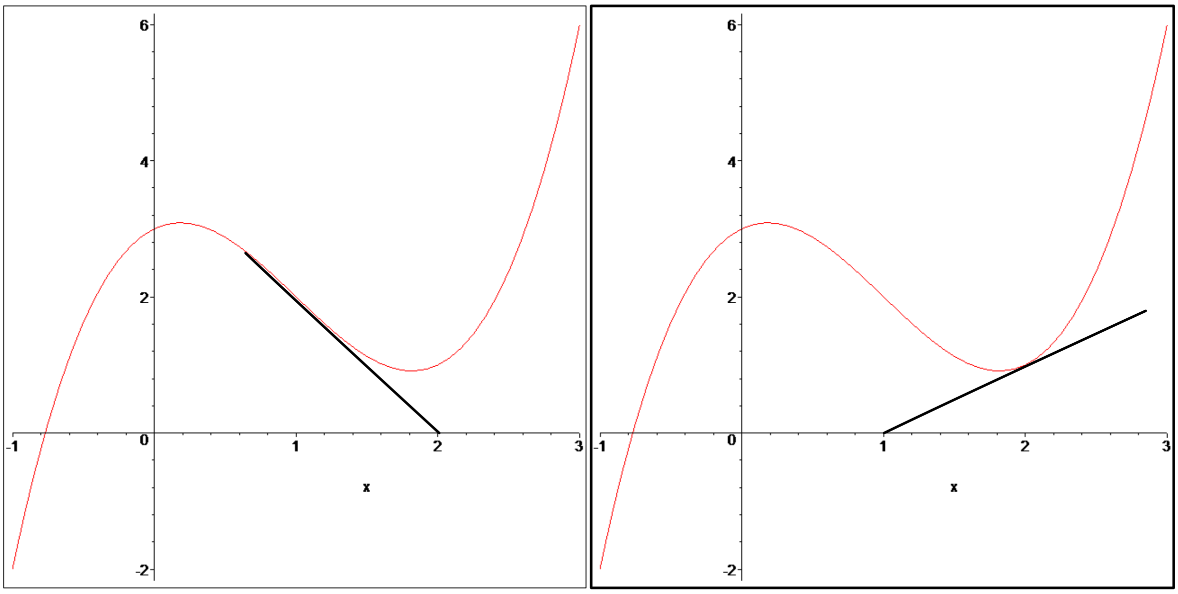
Choose x_0=1.0

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1.0 - \frac{2.0}{-2.0} = 2.0;$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 2.0 - \frac{1.0}{1.0} = 1.0;$$

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 1.0 - \frac{2.0}{-2.0} = 2.0;$$

$$x_4 = 1.0; \qquad x_5 = 2.0;$$



Oscillatory behavior of Newton's Method

Another Newton Program (derivative of the function evaluated in the M-file)

MATLAB M-File

```
function [x0,err,k,y]=newton1(f,x0,delta,epsilon,maxit)
%INPUT    f is object function input as a string 'f'
%         x0 is the initial approximation to a zero of f
%         delta tolerance for x0
%         epsilon tolerance for function values y
%         maxit is the maximum number of iterations
%OUTPUT   result is the Newton-Raphson approximation to the zero
%         err  is the error  estimate for x0
%         k is the number of iterations
%         y is the function value f(result)
g=diff(f);
h=inline(g);
ff=inline(f);
for k=1:maxit
   y=ff(x0);
   dy=h(x0);
   x1=x0-(y/dy);
   err=abs(x1-x0);
   relerr=2*err/(abs(x1)+delta);
     X=[k,x0,y,dy]
   x0=x1;
       if (err<delta)|(relerr<delta)|(abs(y)<epsilon),break,end
end
```

Example:

```
>> f=('x-cos(x)')
```
not inline

```
f =
x-cos(x)
>> newton1(f,0.75,0.001,0.001,10)
X =    1.0000    0.7500    0.0183    1.6816
X =    2.0000    0.7391    0.0000    1.6736
ans =    0.7391
```

i	x_i	$f(x_i)$	$f^{'}(x_i)$	$f^{''}(x_i)$	$\lvert g^{'}(x_i)\rvert$
0	0.0000	-1.00000	1.00000	1.00000	1.00000
1	1.0000	0.45970	1.84147	0.54030	0.07325
2	0.7504	0.01898	1.68193	0.73142	0.00491
3	0.7391	0.00002	1.67362	0.73908	0.00001

Complex Roots

Newton's method works with complex roots if we give it a complex value for the starting value.

Example:

Use Newton's method on

$$f(x) = x^3 + 2x^2 - x + 5.$$

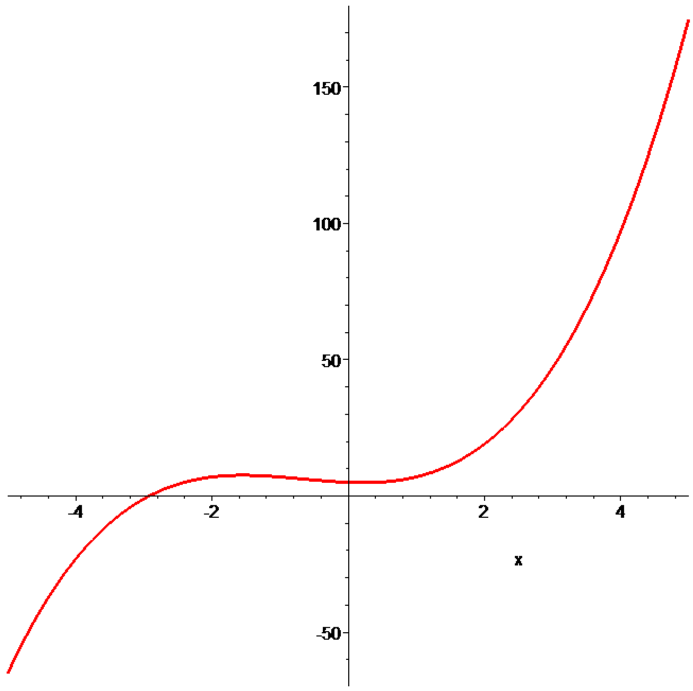


Figure shows the graph of f(x). It has a real root about x=-3, whereas the other two roots are complex because the x-axis is not crossed again.

MAPLE SOLUTION

```
> solve(x^3+2*x^2-x+5,x);
```

$$-\frac{(676+12\sqrt{3021})^{(1/3)}}{6}-\frac{14}{3\,(676+12\sqrt{3021})^{(1/3)}}-\frac{2}{3},\ \frac{(676+12\sqrt{3021})^{(1/3)}}{12}+\frac{7}{3\,(676+12\sqrt{3021})^{(1/3)}}-\frac{2}{3}+\frac{1}{2}I\sqrt{3}\left(-\frac{(676+12\sqrt{3021})^{(1/3)}}{6}+\frac{14}{3\,(676+12\sqrt{3021})^{(1/3)}}\right),$$

$$\frac{(676+12\sqrt{3021})^{(1/3)}}{12}+\frac{7}{3\,(676+12\sqrt{3021})^{(1/3)}}-\frac{2}{3}-\frac{1}{2}I\sqrt{3}\left(-\frac{(676+12\sqrt{3021})^{(1/3)}}{6}+\frac{14}{3\,(676+12\sqrt{3021})^{(1/3)}}\right)$$

MATLAB SOLUTION

solve(f)

ans = [-1/6*(676+12*3021^(1/2))^(1/3)-14/3/(676+12*3021^(1/2))^(1/3)-2/3]

[1/12*(676+12*3021^(1/2))^(1/3)+7/3/(676+12*3021^(1/2))^(1/3)-2/3+1/2*i*3^(1/2)*(-1/6*(676+12*3021^(1/2))^(1/3)+14/3/(676+12*3021^(1/2))^(1/3))]

[1/12*(676+12*3021^(1/2))^(1/3)+7/3/(676+12*3021^(1/2))^(1/3)-2/3-1/2*i*3^(1/2)*(-1/6*(676+12*3021^(1/2))^(1/3)+14/3/(676+12*3021^(1/2))^(1/3))]

If we begin Newton's method with $x_0 = 1 + i$

```
>> f=('x^3+2*x^2-x+5')
f =
x^3+2*x^2-x+5
>> newton1(f,1+i,0.001,0.001,10)
```

X =

1.0000	**1.0000 + 1.0000i**	**2.0000 + 5.0000i**	**3.0000 +10.0000i**
2.0000	**0.4862 + 1.0459i**	**1.3183 + 0.5861i**	**-1.6273 + 7.2347i**
3.0000	**0.4481 + 1.2367i**	**-0.0712 - 0.1660i**	**-3.1929 + 8.2718i**
4.0000	**0.4627 + 1.2224i**	**0.0016 - 0.0014i**	**-2.9898 + 8.2835i**

ans = 0.4629 + 1.2225i

If we begin with a real starting value –say $x_0 = -3$.

We get convergence to the REAL root

```
>> f=('x^3+2*x^2-x+5')
f =
x^3+2*x^2-x+5
>> newton1(f,-3,0.001,0.001,10)
```

X =

1	**-3**	**-1**	**14**
2.0000	**-2.9286**	**-0.0353**	**13.0153**
3.0000	**-2.9259**	**-0.0001**	**12.9785**

ans = -2.9259