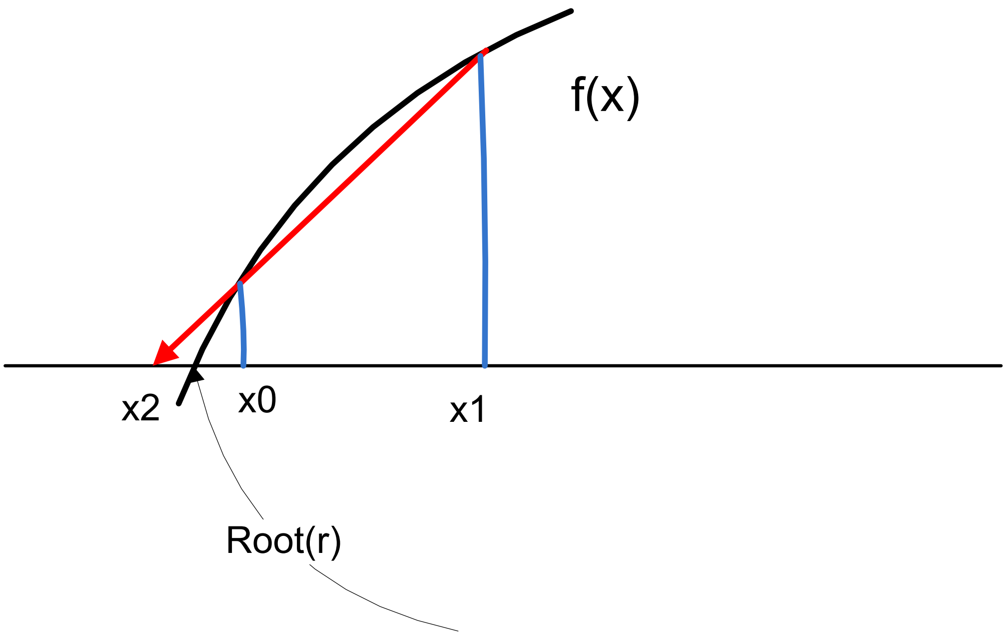
The Secant Method

The Secant method begins by finding two points on the curve of f(x), hopefully near to the root we seek.



The intersection of the line with the x-axis is not at x = r but that it should be close to it. From the obvious <u>similar triangles we can write</u>

$$\frac{(x_1 - x_2)}{f(x_1)} = \frac{(x_1 - x_0)}{f(x_1) - f(x_0)}$$

And from this solve for x_2:

$$x_2 = x_1 - f(x_1)\frac{(x_1 - x_0)}{f(x_1) - f(x_0)}$$

Because f(x) is not exactly linear, x_2 is not equal the root, but it should be closer than either of the two points we began with.

If we repeat this, we have:

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

The technique is known as the secant method because the line through two points on the curve is called the secant line.

Newton Method

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

Approximation to the derivative gives Secant Method

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

An algorithm for the Secant Method

To determine a root of f(x) = 0, given two values, x_0 and x_1, that are near the root .

If $|f(x_0)| < |f(x_1)|$ Then

Swap x_0 with x_1.

Repeat

Set $x_2 = x_1 - f(x_1) * \dfrac{x_0 - x_1}{f(x_0) - f(x_1)}$

Set $x_0 = x_1$

Set $x_1 = x_2$

Until $|f(x_2)| < ToleranceValue.$

End If.

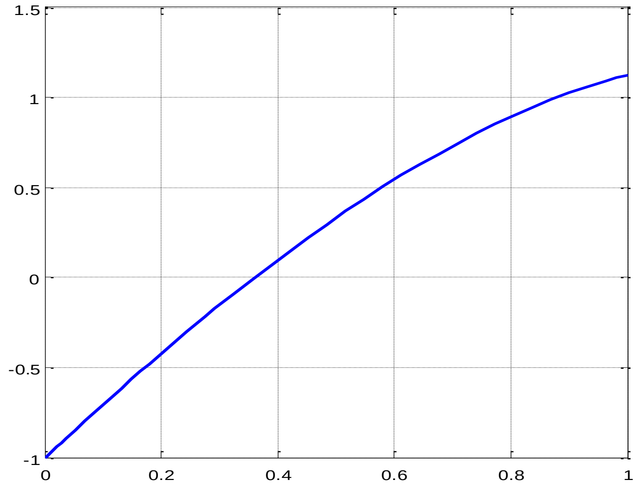
Note: If f(x) is not continuous, the method may fail.

MATLAB M-File

```
function
[x1,err,k,y]=secant(f,x0,x1,delta,epsilon,maxit)
%Input    - f is the object function
%              - x0 and x1 are the initial
approximations to a zero of f
%              - delta is the tolerance for x2
%              - epsilon is the tolerance for the
function values y
%              - maxit is the max. number of iterations
%Output - p1 is the secant method approximation to the
zero
%              - err is the error estimate for p1
%              - k is the number of iterations
%              - y is the function value f(p1)
for k=1:maxit
    x2=x1-f(x1)*(x1-x0)/(f(x1)-f(x0));
    err=abs(x2-x1);
    relerr=2*err/(abs(x2)+delta);
    x0=x1;
    x1=x2;
    y=f(x1);
    X=[k,x0,x1,x2,y]
    if
(err<delta)|(relerr<delta)|(abs(y)<epsilon),break,end
end
```

Example:

Secant Method on $f(x) = 3x + \sin(x) - e^x = 0$.



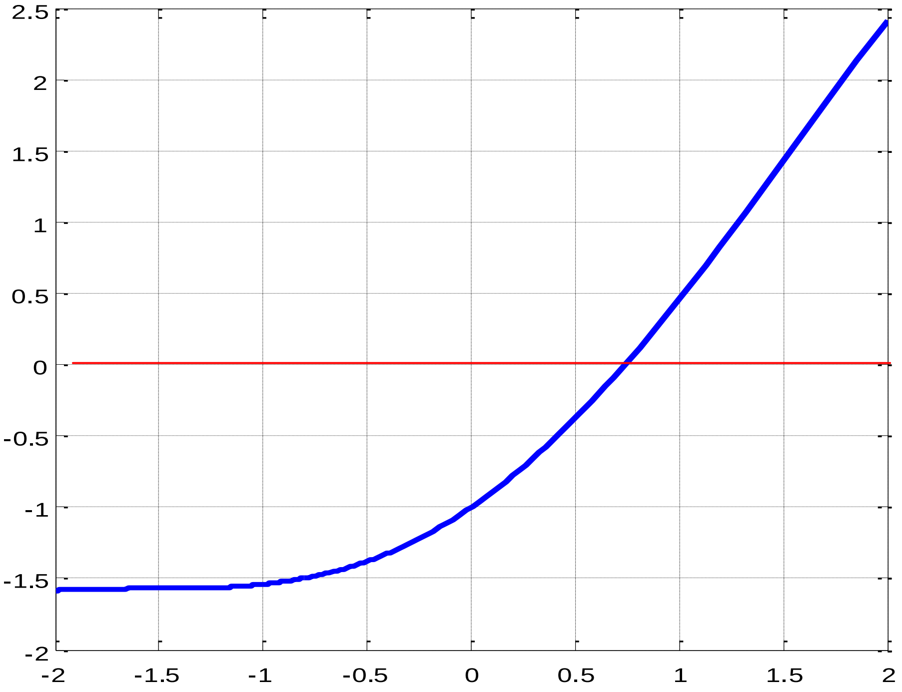
Step	x_0	x_1	x_2	$f(x_2)$
1	1	0	0.4709896	0.2651558
2	0	0.4709896	0.3722771	2.953367E-02
3	0.4709896	0.3722771	0.3599043	-1.294787E-03
4	0.3722771	0.3599043	0.3604239	5.552969E-06
5	0.3599043	0.3604239	0.3604217	3.554221E-08

```
>> f=inline('3*x+sin(x)-exp(x)')
f =  Inline function:
     f(x) = 3*x+sin(x)-exp(x)
>> secant(f,1,0,0.001,0.001,10)
X =    1.0000        0       0.4710    0.4710    0.2652
X =    2.0000    0.4710    0.3723    0.3723    0.0295
X =    3.0000    0.3723    0.3599    0.3599   -0.0013
X =    4.0000    0.3599    0.3604    0.3604    0.0000
ans =    0.3604
```

Example

When Secant method is applied to

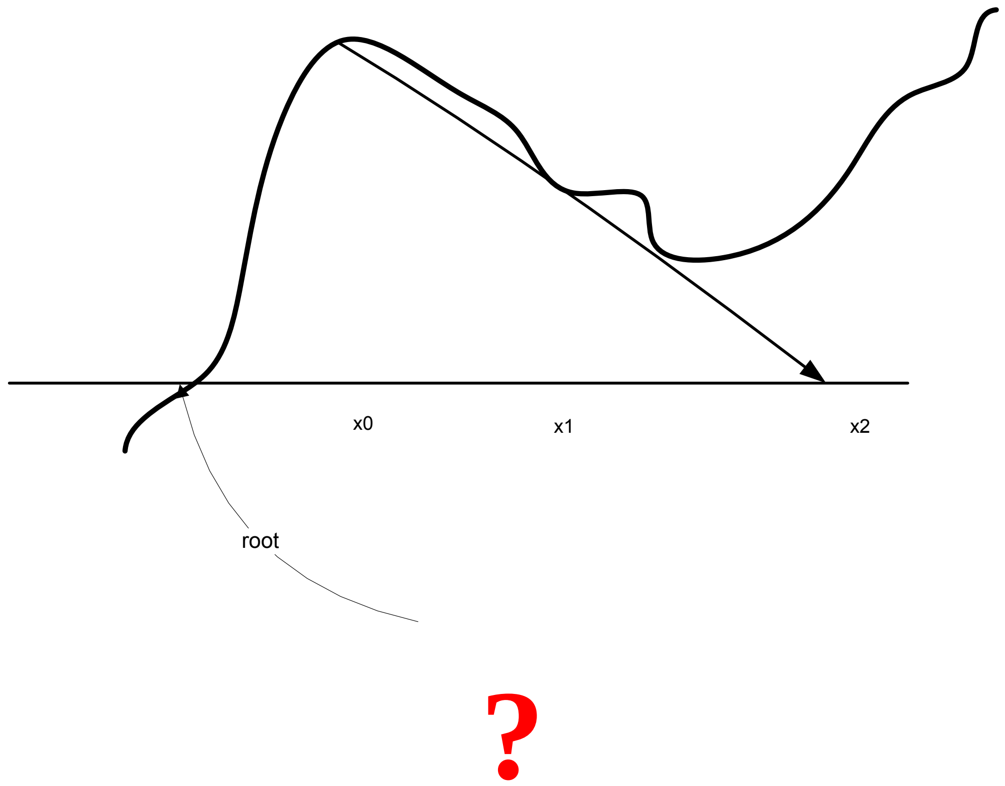
$$f(x) = x - \cos(x)$$



>> f=inline('x-cos(x)')
f =   Inline function:
**   f(x) = x-cos(x)**
>> secant(f,0,1,0.001,0.001,10)

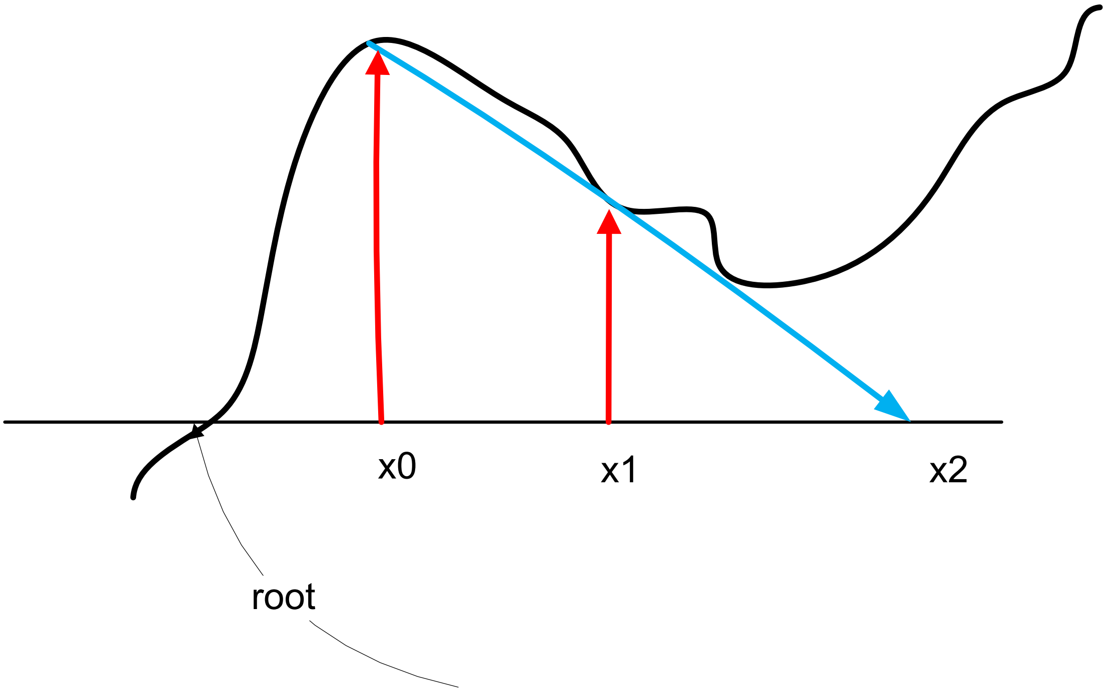
X =	**1.0000**	**1.0000**	**0.6851**	**0.6851**	**-0.0893**
X =	**2.0000**	**0.6851**	**0.7363**	**0.7363**	**-0.0047**
X =	**3.0000**	**0.7363**	**0.7391**	**0.7391**	**0.0001**

ans = 0.7391

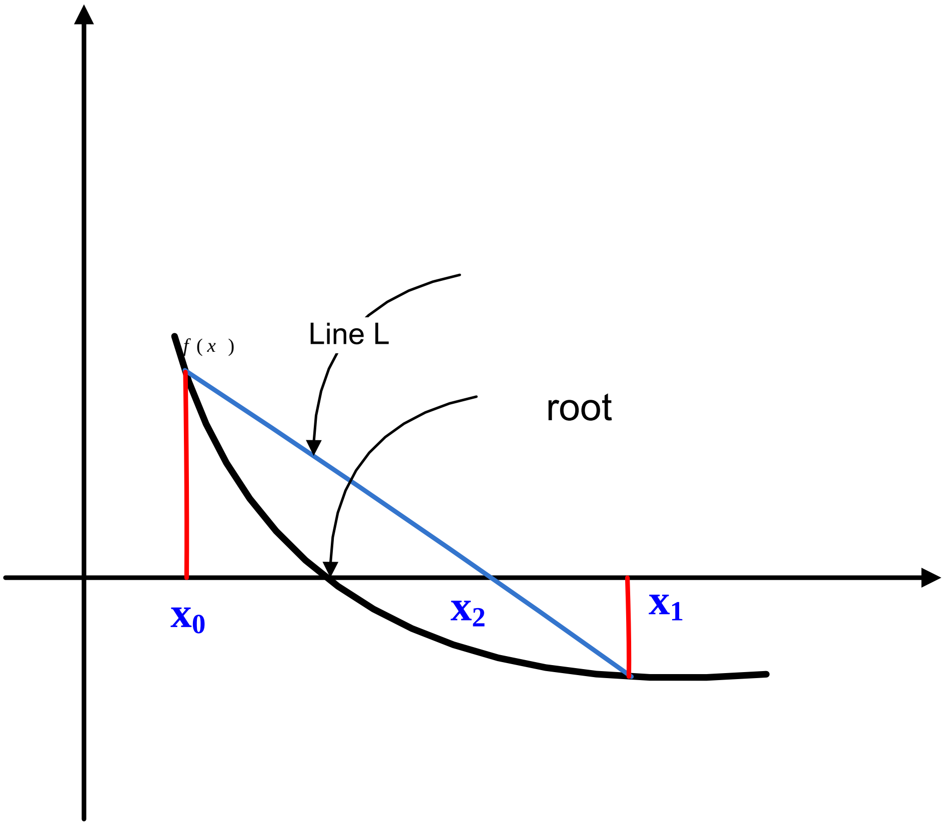

x0
x1
x2
root
?

Linear Interpolation Method (False Position-Regula Falsi)

The Secant method begins by finding two points on the curve of f(x), hopefully near to the root we seek.



A way to avoid such pathology is to ensure that the root is <u>bracketed between the two starting values and remains</u> between the successive pairs. When this is done, the method is known as *<u>linear interpolation, or more often, as the method of false position</u>*.



This technique is similar to bisection except the next iterate is taken at the intersection of a line between the pair of x-values and the x-axis rather than at the mid point.

We assume that $f(x_0)$ and $f(x_1)$ have opposite signs the bisection method used the mid point of the interval $[x_0, x_1]$ as the next iterate.

A better approximation is obtained if we find the point $(x_2, 0)$ where the secant line L joining the points $((x_0, f(x_0)), (x_1, f(x_1))$ crosses the x-axis.

To find the value of x_2, we write down two versions of the slope m of the line L:

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

Where the points $((x_0, f(x_0))$ and $(x_1, f(x_1))$ are used, and

$$m = \frac{f(x_2) - f(x_1)}{x_2 - x_1} = \frac{0 - f(x_1)}{x_2 - x_1}$$

Where the points $((x_1, f(x_1)$ and $(x_2\ f(x_2))$ are used.

Equating the slopes

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{0 - f(x_1)}{x_2 - x_1}$$

which is easily solved for x_2 to get

$$x_2 = x_1 - \frac{(x_1 - x_0)}{f(x_1) - f(x_0)} f(x_1)$$

or

$$x = b - \frac{(b - a)}{f(b) - f(a)} f(b)$$

The regula falsi method, or the rule of false position, proceeds as the in bisection to find the subinterval $[x_0, x_2]$ or $[x_2, x_1]$ that contains the zero by testing for a change of sign of the function, i.e., testing whether

$$f(x_0)f(x_2) < 0$$

Or

$$f(x_1)f(x_2) < 0.$$

The three possibilities

1. **If $f(x_0)$ and $f(x_2)$ have opposite signs a zero lies in $[x_0, x_2]$.**
2. **If $f(x_2)$ and $f(x_1)$ have opposite signs a zero lies in $[x_2, x_1]$.**
3. **If $f(x_2) = 0$ then the zero is x_2.**

$$[x_0, x_1]$$

$$f(x_0)\,f(x_1) < 0$$

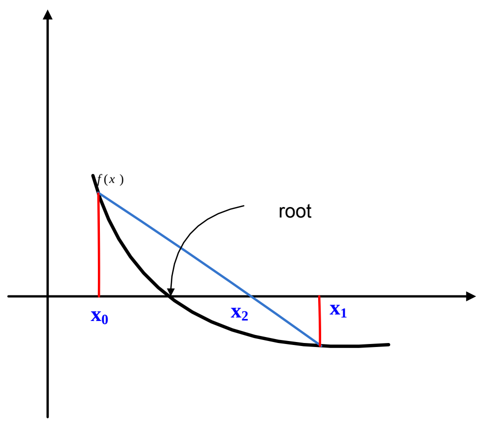
$$x_2 = x_1 - \frac{(x_1 - x_0)}{f(x_1) - f(x_0)} f(x_1)$$

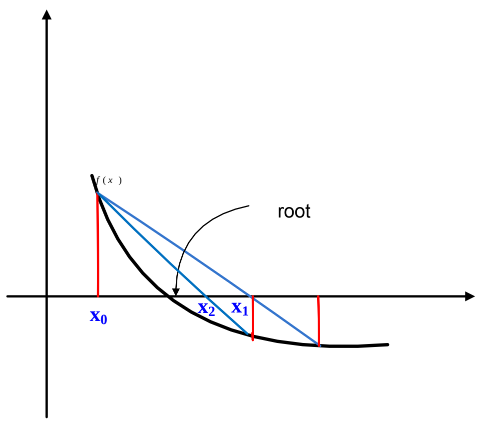
$$f(x_0)\,f(x_2) < 0$$

$$f(x_1)\,f(x_2) < 0$$

$$[x_0, x_2]$$

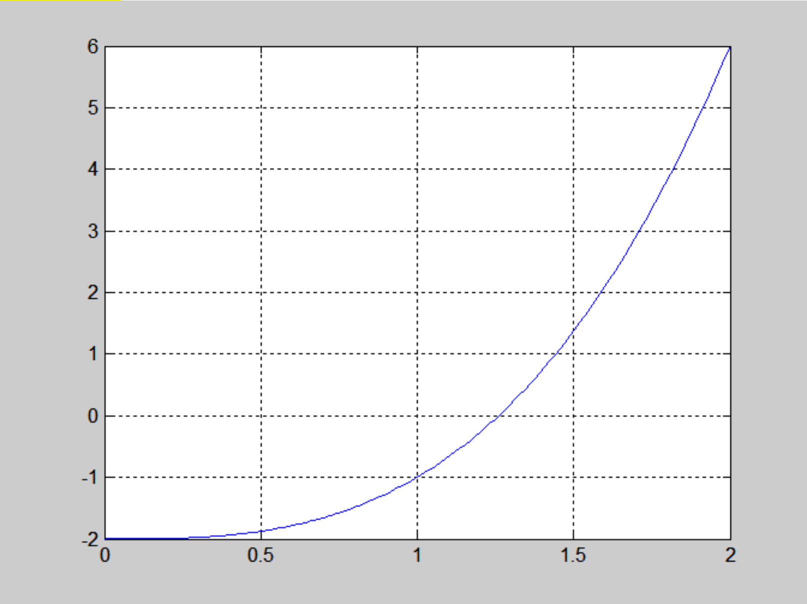
$$[x_2, x_1]$$





Example:

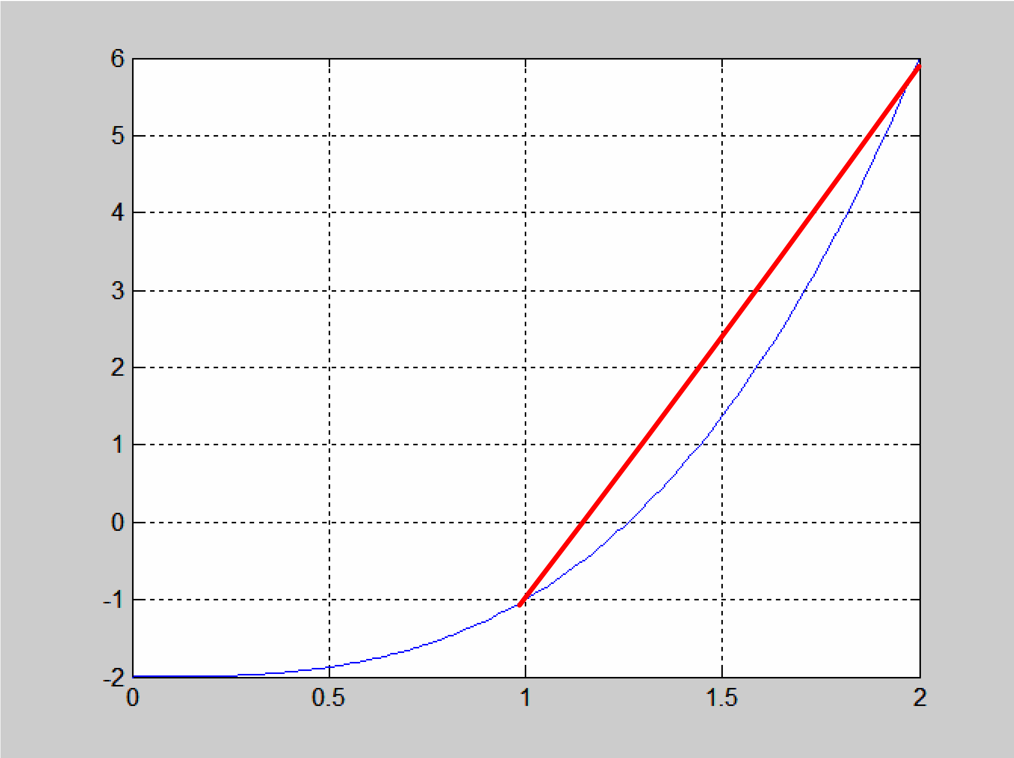
To find a numerical approximation to $\sqrt[3]{2}$, we find the zero of $f(x)=x^3-2$. Since $f(1)=-1$ and $f(2)=6$ we take as our starting bounds on the zero $x_0=1$ $x_1=2$.

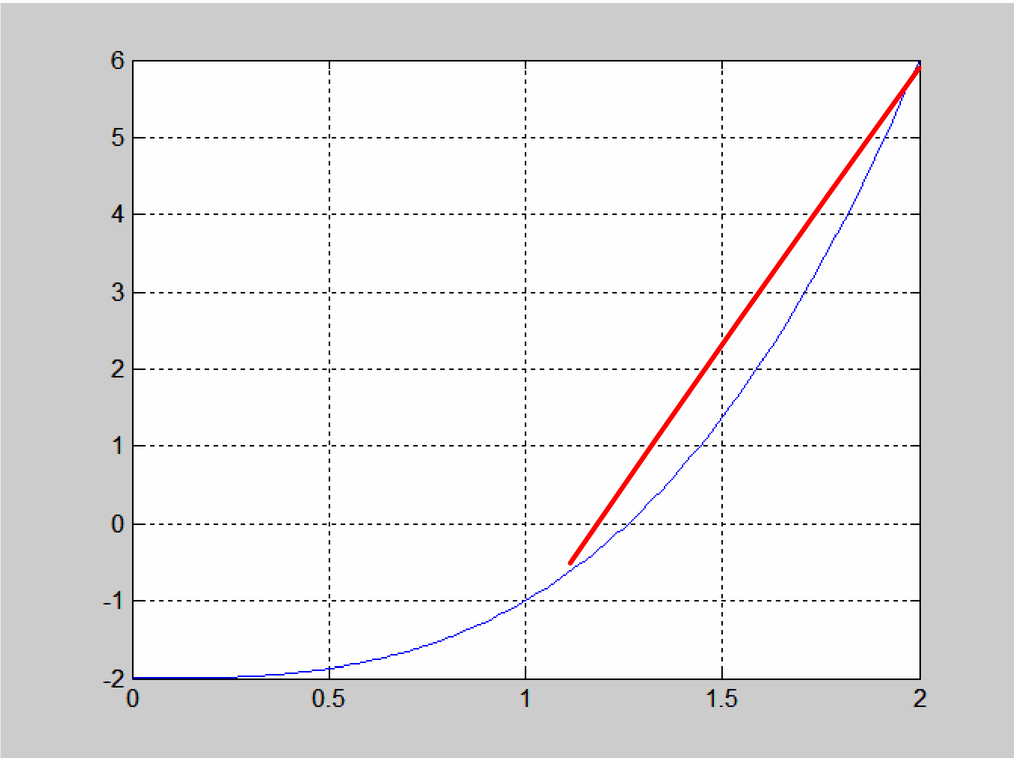


Calculations of $\sqrt[3]{2}$ using regula falsi

Step	x_0	x_1	x_2	$f(x_2)$
1	**1.0000**	**2.0000**	**1.1429**	**-0.50729**
2	**1.1429**	**2.0000**	**1.2097**	**-0.22986**
3	**1.2097**	**2.0000**	**1.2388**	**-0.098736**
4	**1.2388**	**2.0000**	**1.2512**	**-0.041433**
5	**1.2512**	**2.0000**	**1.2563**	**-0.017216**
6	**1.2563**	**2.0000**	**1.2584**	**-0.0071239**
7	**1.2584**	**2.0000**	**1.2593**	**-0.0029429**
8	**1.2593**	**2.0000**	**1.2597**	**-0.0012148**
9	**1.2597**	**2.0000**	**1.2598**	**-0.00050134**
10	**1.2598**	**2.0000**	**1.2599**	**-0.00020687**

$$x_2=2-\frac{(2-1)}{6+1}(6)=2-\frac{6}{7}=\frac{8}{7}=1.1429$$


6
5
4
3
2
1
0
-1
-2
0
0.5
1
1.5
2


6
5
4
3
2
1
0
-1
-2
0
0.5
1
1.5
2

An algorithm for the Method of False Position (regula falsi)

To determine a root of f(x) = 0, given two values, x_0 and x_1, that is $f(x_0)$ and $f(x_1)$ are of opposite sign.

Set $$x_2 = x_1 - \frac{(x_1 - x_0)}{f(x_1) - f(x_0)} f(x_1)$$

If $f(x_2)$ is opposite sign to $f(x_0)$

Set $x_1=x_2$

Else

Set $x_0=x_2$

Until $|f(x_2)| < ToleranceValue.$

End If.

Note: If f(x) is not continuous, the method may fail.

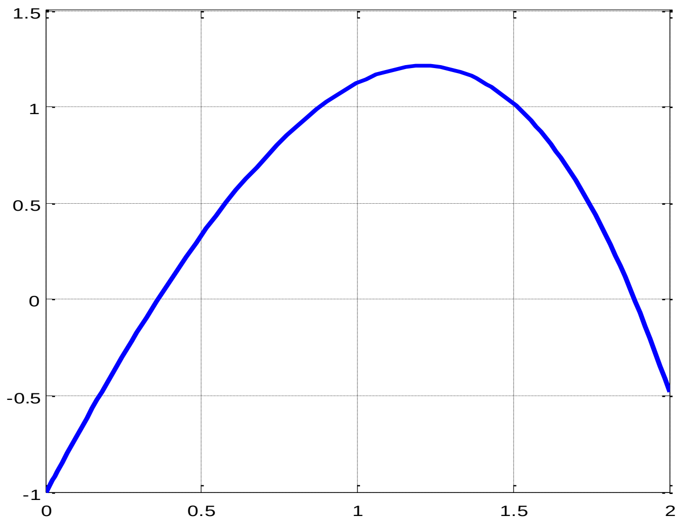
MATLAB M-File

```
function [c,err,yc]=regulafalsi(f,x0,x1,delta,epsilon,max1)
%Input    - f is the function
%           - x0 and x1 are the left and right endpoints
%           - delta is the tolerance for the zero
%           - epsilon is the tolerance for the value of f at the zero
%           - max1 is the maximum number of iterations
%Output - x2 is the zero
%           - yc=f(x2)
%           - err is the error estimate for x2
%If f is defined as an M-file function use the @ notation
% call [c,err,yc]=regula(@f,a,b,delta,epsilon,max1)
%If f is defined as an anonymous function use the
% call [c,err,yc]=regulafalsi(f,a,b,delta,epsilon,max1)
ya=f(x0);
yb=f(x1);
if ya*yb>0
        disp('Note: f(x0)*f(x1) >0'),
        return,
end
for k=1:max1
        dx=yb*(x1-x0)/(yb-ya);
        x2=x1-dx;
        ac=x2-x0;
        yc=f(x2);
        if yc==0,break;
        elseif yb*yc>0
                x1=x2;
                yb=yc;
        else
                x0=x2;
                ya=yc;
        end
        dx=min(abs(dx),ac);
        if abs(dx)<delta,break,end
        if abs(yc)<epsilon, break,end
    X=[k,x0,x1,x2,ya,yb,yc]
end
x2
err=abs(x1-x0)/2
yc=f(x2)
```

Example:

Regula Falsi Method on $f(x) = 3x + \sin(x) - e^x = 0$

```
>> 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
```



With x_0=0 and x_1=1

```
regulafalsi(f,0,1,0.001,0.001,10)
X =
   1.0000        0    0.4710    0.4710   -1.0000    0.2652    0.2652
   2.0000        0    0.3723    0.3723   -1.0000    0.0295    0.0295
   3.0000        0    0.3616    0.3616   -1.0000    0.0029    0.0029
x2 =    0.3605
err =    0.1803
yc =  2.8945e-004
```

With x_0=1 and x_1=2

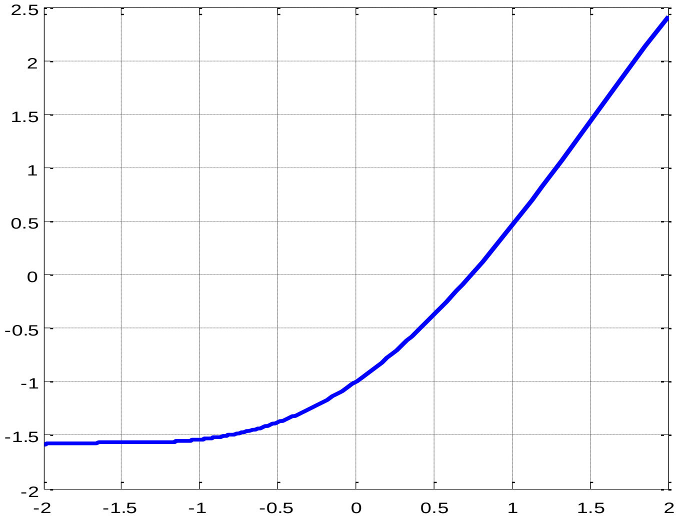
```
>> regulafalsi(f,1,2,0.001,0.001,10)
X =
   1.0000    1.7007    2.0000    1.7007    0.6159   -0.4798    0.6159
   2.0000    1.8689    2.0000    1.8689    0.0813   -0.4798    0.0813
   3.0000    1.8879    2.0000    1.8879    0.0083   -0.4798    0.0083
x2 =    1.8898
err =    0.0551
yc =  8.1430e-004
```

We obtain Second ROOT

Example

When Regula Falsi Method is applied to

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

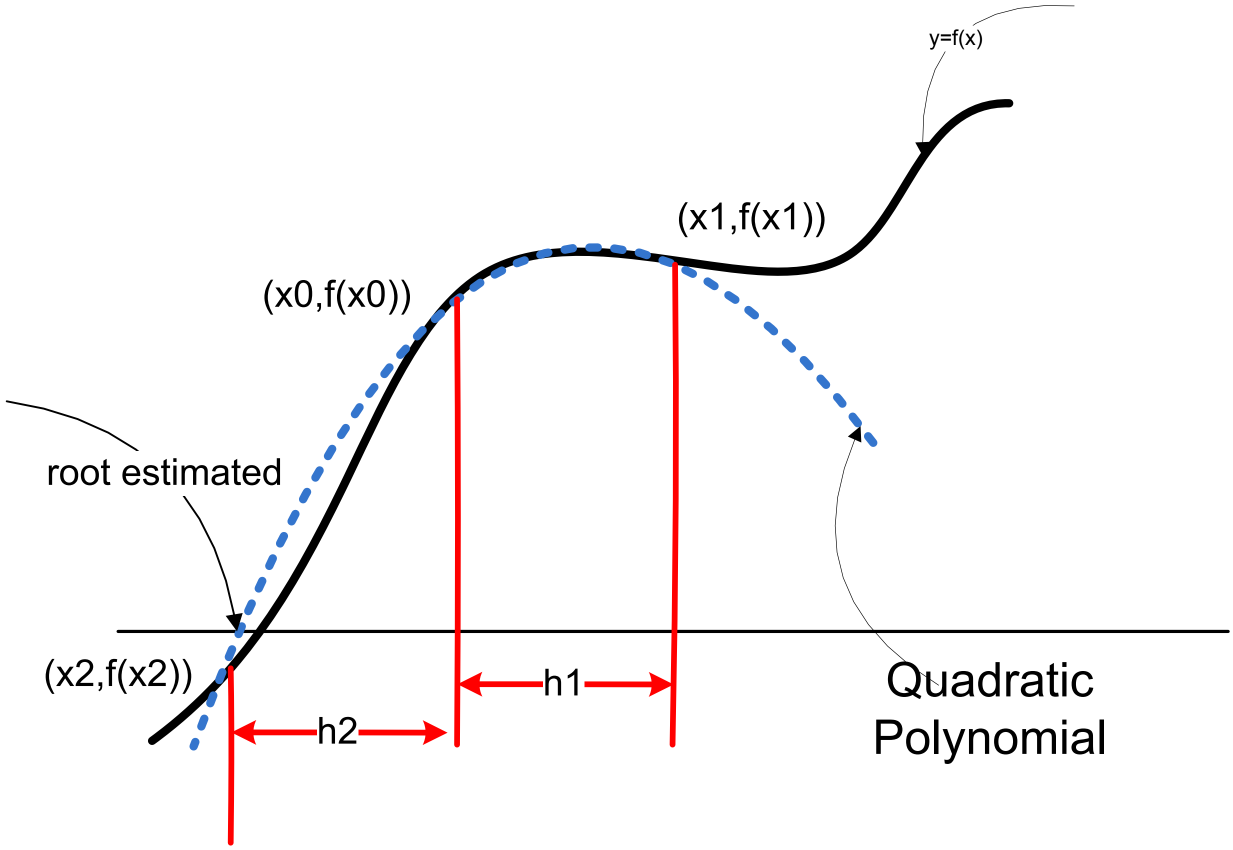


With $x_0=0$ and $x_1=1$

```
>> f=inline('x-cos(x)')
f =
     Inline function:
     f(x) = x-cos(x)
>> regulafalsi(f,0.0,1.0,0.001,0.001,10)
X =
    1.0000    0.6851    1.0000    0.6851   -0.0893    0.4597   -0.0893
    2.0000    0.7363    1.0000    0.7363   -0.0047    0.4597   -0.0047
x2 =   0.7389
err =    0.1305
```

Muller's Method

Most of the roots finding methods that we have considered so far have approximated the function in the neighbored of the root by a straight line. Muller's Method is based on approximating the function in the neighborhood of the root by a quadratic polynomial. This gives a much closer match to the actual curve.



A second degree polynomial is made to fit three points near a root, at x_0, x_1, x_2 with x_0 between x_1 and x_2.

The procedure for Muller's method is developed by writing a quadratic equation that first through three points in the *around of the root*, in the form

$$av^2 + bv + c.$$

The development is simplified if we transform axes to pass through the middle point, by letting $v = x - x_0$.

Let

$$h_1 = x_1 - x_0 \quad \textbf{and} \quad h_2 = x_0 - x_2.$$

We evaluate the coefficients by evaluating quadratic polynomial $P_2(v)$ at three points:

$$\begin{aligned}
v &= 0: & a(0)^2 + b(0) + c &= f_0;\\
v &= h_1: & ah_1^2 + bh_1 + c &= f_1;\\
v &= -h_2: & ah_2^2 - bh_2 + c &= f_2.
\end{aligned}$$

From the first equation $c = f_0$,

Letting $\dfrac{h_2}{h_1} = \lambda$ **we can solve the other two equations for a and b:**

$$a = \frac{\lambda f_1 - f_0(1+\lambda) + f_2}{\alpha h_1^2(1+\lambda)}, \qquad b = \frac{f_1 - f_0 - ah_1^2}{h_1},$$

After computing a, b and c, we solve for the root of $av^2 + bv + c = 0$ by the quadratic formula, choosing the root nearest to the middle point x_0 this value is

$$root = x_0 - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}$$

with the sign in the denominator taken to give the largest absolute value of the denominator

that is, if b > 0, choose plus

if b < 0,choose minus

if b = 0 choose either.

We always reset the subscripts to make x_0 be in the middle of the three values.

Algorithm for Muller's Method

Given the points x_2, x_0 and x_1 in increasing value.

Evaluate the corresponding function values: f_2, f_0 and f_1.

Repeat

(Evaluate the coefficients of the parabola, $ax^2 + bx + c$, determined by the three points. $\{(x_2, f(x_2)), (x_0, f(x_0)), (x_1, f(x_1))\}$;

Set $h_1 = x_1 - x_0$; $h_2 = x_0 - x_2$; $\lambda = \dfrac{h_2}{h_1}$

Set c=f_0

Set $a = \dfrac{\lambda f_1 - f_0(1+\lambda) + f_2}{\lambda h_1^2(1+\lambda)}$, $\quad b = \dfrac{f_1 - f_0 - a h_1^2}{h_1}$,

(next compute the roots of the polynomial.)

Set $root = x_0 - \dfrac{2c}{b \pm \sqrt{b^2 - 4ac}}$

Choose root x_r, closest to x_0 **by making he denominator as large as possible; i.e. if b> 0, choose plus; otherwise choose minus**

If $x_r > x_0$

Then rearrange to x_0, x_1 **and the root**

Else rearrange to x_0, x_2 **and the root**

End If.

(In either case; reset subscripts so that x_0 **is in the middle.)**

Until $|f(x_r)| < TOLERANCE$

Example:

Find a root between 0 and 1 of the same transcendental function as before $f(x) = 3x + \sin(x) - e^x$.

Let

$$
\begin{array}{lll}
x_0 = 0.5, & f(x_0) = 0.330704 & h_1 = 0.5 \\
x_1 = 1.0, & f(x_1) = 1.123489 & h_2 = 0.5 \\
x_2 = 0, & f(x_2) = -1 & \lambda = 1.0
\end{array}
$$

x_2, x_0 and x_1 in increasing value

$$c = f(x_0) = 0.33074$$

$$a = \frac{(1.0)(1.123189) - 0.330704(2.0) + (-1)}{1.0(0.5)^2(2.0)} = -1.07644,$$

$$b = \frac{1.123189 - 0.33074 - (11.07644)(0.5)^2}{0.5} = 2.12319$$

and

$$root = 0.5 - \frac{2(0.330704)}{2.12319 + \sqrt{(2.12319)^2 - 4(-1.07644)(0.330704)}}$$
$$= 0.354914$$

RULE:

x_0 ***be in the middle of the three values***

$x_0 = 0.5, \quad x_1 = 1.0, \quad x_2 = 0.0$ and root $= 0.354914$

For the next step

If $root(x_r) > x_0$

Then rearrange to

x_0, x_1 **and the root**

Else rearrange to x_0, x_2 **and the root**

For the next iteration we have ($x_r < x_0$ **)**

$$0.5, \quad 0.0, \quad 0.354914$$

Set middle number as x_0

$$x_0 = 0.354914, \quad x_1 = 0.5, \quad x_2 = 0.0$$

$$\begin{array}{lll} x_0 = 0.354914, & f(x_0) = -0.0138066 & h_1 = 0.145086 \\ x_1 = 0.5, & f(x_1) = 0.330704 & h_2 = 0.354914 \\ x_2 = 0, & f(x_2) = -1 & \lambda = 2.44623. \end{array}$$

Then

$$a=\frac{(2.44623)(0.330704)-(-0.0138066)(3.44623)+(-1)}{2.44623(0.145086)^2(3.44623)}$$
$$=-0.808314,$$
$$b=\frac{0.330704-(-0.0138066)-(-0.808314)(0.145086)^2}{0.145086}$$
$$=2.49180,$$
$$c=-0.0138066.$$

and

$$root=0.354914-\frac{2(-0.0138066)}{2.49180+\sqrt{(2.49180)^2-4(-0.808314)(-0.013866)}}$$
$$=0.360465.$$

$$x_0=0.354914,\ x_1=0.5,\quad x_2=0,\quad \text{root}=0.360465$$

For the next iteration we have ($x_r > x_0$)

0.354914 0.5 0.360465

Complete the following and compute the new root

$x_0=$................., $f(x_0)=$......................... $h_1=$.......................
$x_1=$.................., $f(x_1)=$......................... $h_2=$.......................
$x_2=$.................., $f(x_2)=$....................... $\lambda=$.......................

After a third iteration, we get 0.3604217 as the value for the root, which is identical to that from Newton's method after three iterations.

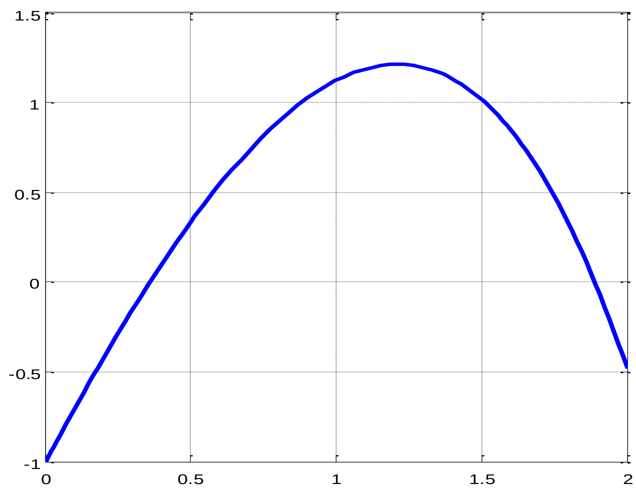
MATLAB M-File (MULLER METHOD)

```
function [x,y,err]=muller(f,x0,x1,x2,delta,epsilon,maxit)
%Input   - f is the object function
%              - x0, x1, and x2 are the initial approximations
%         - delta is the tolerance for x0, x1, and x2
%              - epsilon the the tolerance for the function values y
%         - max1 is the maximum number of iterations
%Output- p is the Muller approximation to the zero of f
%         - y is the function value y = f(x)
%         - err is the error in the approximation of x.
%If f is defined as an M-file function use the @ notation
% call [p,y,err]=muller(@f,x0,x1,x2,delta,epsilon,maxit).
%If f is defined as an anonymous function use the
% call [p,y,err]=muller(f,x0,x1,x2,delta,epsilon,maxit).
%Initalize the matrices P and Y
P=[x0 x1 x2];
Y=f(P);
for k=1:maxit
  h0=P(1)-P(3);h1=P(2)-P(3);e0=Y(1)-Y(3);e1=Y(2)-Y(3);c=Y(3);
  denom=h1*h0^2-h0*h1^2;
  a=(e0*h1-e1*h0)/denom;
  b=(e1*h0^2-e0*h1^2)/denom;
    %Suppress any complex roots
  if b^2-4*a*c > 0
    disc=sqrt(b^2-4*a*c);
  else
    disc=0;
  end
    %Find the smallest root of (17)
  if b < 0
    disc=-disc;
  end
  z=-2*c/(b+disc);
  x=P(3)+z;
   X=[k,a,b,c,x]
  %Sort the entries of P to find the two closest to p
  if abs(x-P(2))<abs(x-P(1))
    Q=[P(2) P(1) P(3)];
    P=Q;
    Y=f(P);
  end
  if abs(x-P(3))<abs(x-P(2))
    R=[P(1) P(3) P(2)];
    P=R;
    Y=f(P);
  end
    %Replace the entry of P that was farthest from p with p
  P(3)=x;
  Y(3) = f(P(3));
  y=Y(3);
    err=abs(z);
  relerr=err/(abs(x)+delta);
  if (err<delta)|(relerr<delta)|(abs(y)<epsilon)
    break
  end
end
```

Example:

Muller Method on $f(x) = 3x + \sin(x) - e^x = 0$

```
>> 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
```



With x_0=0.5 and x_1=1.0 and x_2=0.0

```
>>f=inline('3*x+sin(x)-exp(x)')
f =
     Inline function:
     f(x) = 3*x+sin(x)-exp(x)
>> muller(f,0.5,1.0,0.0,0.001,0.001,10)
X =
      I          a          b          c        root
   1.0000   -1.0764    3.1996   -1.0000    0.3549
   2.0000   -0.8083    2.4918   -0.0138    0.3605
ans = 0.3605
```

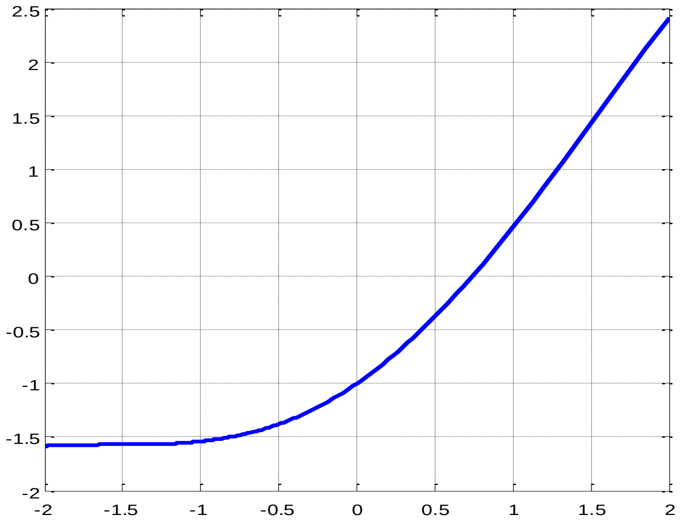
With x_0=0.5 and x_1=1.0 and x_2=0.0 and high tolerance

```
>> muller(f,0.5,1.0,0.0,0.00001,0.00001,10)
X =
      I          a          b          c        root
   1.0000   -1.0764    3.1996   -1.0000    0.3549
   2.0000   -0.8083    2.4918   -0.0138    0.3605
   3.0000   -0.9471    2.5014    0.0001    0.3604
ans =   0.3604
yc =  8.1430e-004
```

Example

When Muller is applied to

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




```
f=inline('x-cos(x)')
f =
    Inline function:
    f(x) = x-cos(x)
>> muller(f,0.5,1.0,0.0,0.00001,0.00001,10)
X =
       I          a          b          c        root
    1.0000    0.4297    1.0300   -1.0000    0.7415
    2.0000    0.3649    1.6684    0.0040    0.7391
    3.0000    0.3943    1.6735   -0.0000    0.7391
ans =    0.7391
```