Solving Nonlinear Algebraic Equations: Functions of One Variable

Selis Önel, PhD

Numerical Iterative Methods

- Graphical method
- Bisection method
- Secant-type methods
  - Regula falsi
  - Secant method
- Newton’s method
- Muller’s Method
- Minimization
  - Golden-section search
  - Brent’s method

Exact Zeros of Nonlinear Algebraic Equations

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

- Exact zeros (roots) of f(x) can be found by the quadratic formula
- However, no such method exists for most nonlinear functions
- Formula for finding the zeros of a cubic function is very complicated
- Niels Henrik Abel (1802-1829): In 1824 **Abel** proved the impossibility of solving algebraically the general equation of the fifth and higher degrees

Abel, Niels Henrik (1802-1829)

Norwegian mathematician: Born on August 5, 1802 in the small village of Findoe, Norway

Abel's life was spent in poverty, caused by the large size of his family (he had six brothers and his father died when he was only eighteen) and the difficult economic situation in Norway at that time. Abel died of tuberculosis at the age of 26 after being forced to live in miserable conditions because of his inability to obtain a university post.

- At the age of 16, Abel gave a proof of the **binomial theorem** valid for all numbers, extending Euler's result which had only held for **rationals**. At age 19, he showed there is no general algebraic solution for the roots of a quintic equation, or any general polynomial equation of degree greater than four, in terms of explicit algebraic operations. To do this, he invented (independently of Galois) an extremely important branch of mathematics known as group theory, which is invaluable not only in many areas of mathematics, but for much of physics as well. Among his other accomplishments, Abel wrote a monumental work on elliptic functions which, however, was not discovered until after his death. When asked how he developed his mathematical abilities so rapidly, he replied "by studying the masters, not their pupils."
- Abel sent a paper on the unsolvability of the quintic equation to Gauss, who proceeded to discard without a glance what he believed to be the worthless work of a crank. In 1825, the Norwegian government funded Abel on a scholarly visit to France and Germany. Abel then traveled to Paris, where he gave an important paper revealing the double periodicity of the elliptic functions, which Legendre later described to Cauchy as "a monument more lasting than bronze" (borrowing a famous sentence by the Roman poet Horatius). However, Cauchy proceeded to misplace the manuscript. In Berlin, Abel met and was befriended by August Crelle, an amateur mathematician who had founded the famous *Journal für die reine und angewandte Mathematik* (Journal for pure and applied mathematics), which had published several papers by Abel.
- However, an offer of a professorship in Berlin was not forthcoming for four years, by which time it was too late. A letter from Crelle arrived two days after Abel's death, informing him that he had been offered professorship at the University of Berlin.

Ref: http://scienceworld.wolfram.com/biography/Abel.html

Approximate Zeros of Nonlinear Functions

- Bisection method is a systematic searching technique
- Secant, false-position and Newton’s methods use straight-line approximation to the function whose zero is sought
- More powerful methods use a quadratic approximation to the function or a combination of these techniques
- Each approach produces a succession of approximations

Choosing the Right Technique

Answer the following questions:

- Does the approximation approach the desired solution?
- How rapidly is the solution approached?
- How much computational effort is required?

Ex: Finding the Square Root Using an Iterative method: Babylonian Method

$y=x^2-3$

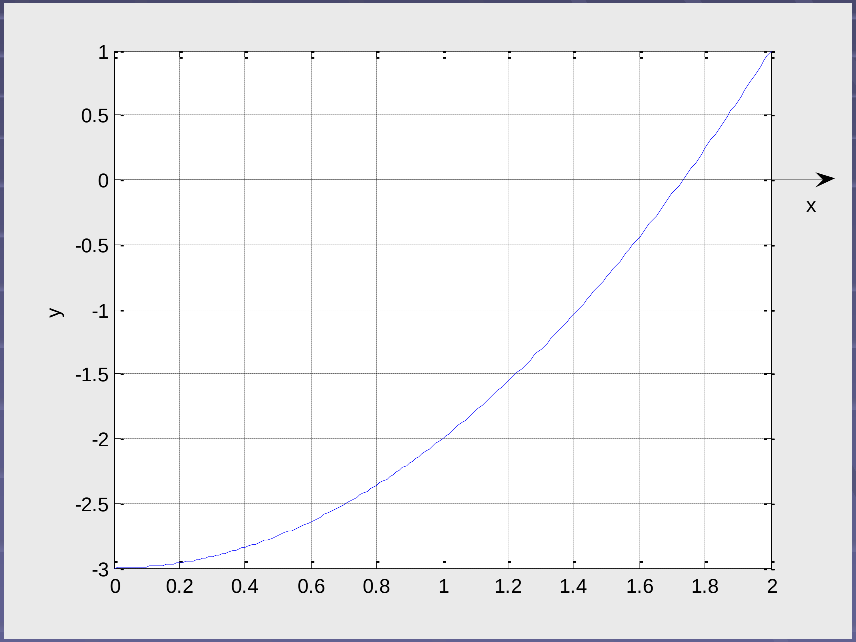
To find the square root of a positive number c, write $x^2=c$ in implicit form as:

$x=1/2(x+c/x)$

Then use an iterative algorithm to use the RHS of the equation to generate an updated estimate for the desired value of x

Remember "fixed-point iteration"

```
>> x=0:0.01:2;  y=x.^2-3; plot(x,y);
xlabel('x'); ylabel('y'); grid on
```



$x_1=1/2(x_0+c/x_0)$

$x_2=1/2(x_1+c/x_1)$

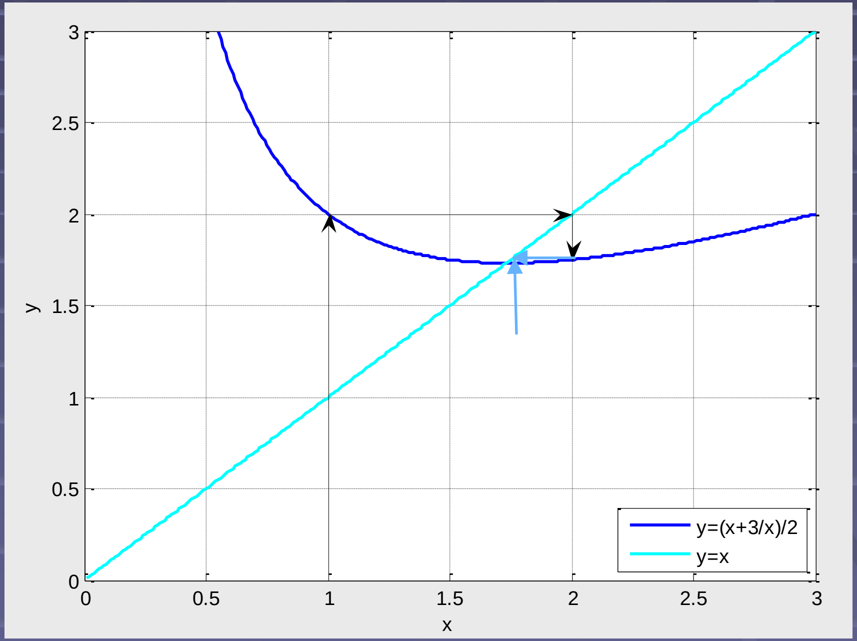
$x_k=1/2(x_{k-1}+c/x_{k-1})$

Ex: Finding the Square Root Using an Iterative method: Babylonian Method

Fixed point iterations to find a root of $x^2=3$:

```
%Calculate the roots of y=x^2-3, i.e. sqrt(3)
%using fixed point iteration
tol=1;
k=1;
x(k)=1;
while tol>.0001,
    k=k+1;
    x(k)=(x(k-1)+3/x(k-1))/2;
    tol=abs(x(k)-x(k-1));
end
k, x(k), x(k-1)
```

```
k =      5
x(5) =    1.73205081001473
x(4) =    1.73214285714286
X(3) =    1.75000000000000
X(2) =    2.0
```



x1=1/2(1+3/1)=2

x2=1/2(2+3/2)=7/4

…

xk=1/2(xk-1+c/xk-1)

Bisection Method

Bisection method is a systematic search technique for finding a zero of a continuous function

- If a root is known to exist for a function f(x)=0 in an interval, subdivide the interval into equal intervals and determine which subinterval contains the root
- Simple and intuitive method

Bisection Method

- Suppose that an interval [a,b] is determined since the function changes sign between a and b.
- The approximate solution is the midpoint of the interval: m=(a+b)/2
- A zero must lie in either [a,m] or [m,b]
- Determine the appropriate subinterval by testing the sign of the function in the interval [a,m]
- If the sign of f(x) changes on [a,m], search continues on [a,m]
- Otherwise, it continues on [m,b]

Remember: Intermediate Value Theorem

- If $f(x)$ is a continuous function on $[a,b]$ with $f(a) \neq f(b)$, and N is any number between $f(a)$ and $f(b)$, then there is a number m such that $a<m<b$ and $f(m) = N$.
- If $f(a)<0$ and $f(b)>0$ (or vice versa), then $f(x)$ has a zero in the interval (a,b)

Ex1: Bisection Method - Square root

Find the numerical approximation to the zero of $y=f(x)=x^2-3$

- We know that f(1)=-2 and f(2)=1
- So start with boundary values a=1 and b=2
- First approximation to the zero is:

m=(a+b)/2 = (1+2)/2= 1.5

$y(m)=y(1.5)=(1.5)^2-3= -0.75$

- y(a) and y(m) have the same sign, but y(m) and y(b) have opposite signs, so the zero should be in the interval [m,b]

Ex1: Bisection Method - Square root

```
%Bisection method to find the squareroot
a=1; b=2;
tol=0.0001; it=0;
f=inline('x^2-3','x');
ya=feval(f,a);   yb=feval(f,b);
disp('it     a      b     m     ya       yb       ym      err');
while ya*yb<0
    it=it+1;   m=(a+b)/2; ym=feval(f,m);
    err = abs(b-a)/2;
out=[it, a, b, m, ya, yb, ym, err];  disp(out)
     if ym==0, a=m; b=m; break,
     elseif ym*ya<0, b=m; yb=ym;
     else a=m; ya=ym;
     end
     if abs(b-a)<tol, break, end
end
format short
```

Remember: inline functions

- An inline function is one for which the compiler copies the code from the function definition directly into the code of the calling function rather than creating a separate set of instructions in memory.
- Instead of transferring control to and from the function code segment, a modified copy of the function body may be substituted directly for the function call. In this way, the performance overhead of a function call is avoided.
- A function is declared inline by using the **inline** function specifier or by defining a member function within a class or structure definition. The **inline** specifier is only a suggestion to the compiler that an inline expansion can be performed; the compiler is free to ignore the suggestion.

- In MATLAB®:

inline(expression) constructs an inline function object from the MATLAB expression contained in the string expression. The input arguments are automatically determined by searching expression for variable names (symvar). If no variable exists, 'x‘ is used.

symvar determines the symbolic variables in an expression.
Ex: symvar('cos(pi*x - beta1)') returns {'beta1';'x'}

Ex1: Bisection Method - Square root

it	a	b	m	ya	yb	ym	err
1.0000	1.0000	2.0000	1.5000	-2.0000	1.000	-0.7500	0.5000
2.0000	1.5000	2.0000	1.7500	-0.7500	1.0000	0.0625	0.2500
3.0000	1.5000	1.7500	1.6250	-0.7500	0.0625	-0.3594	0.1250
4.0000	1.6250	1.7500	1.6875	-0.3594	0.0625	-0.1523	0.0625
5.0000	1.6875	1.7500	1.7188	-0.1523	0.0625	-0.0459	0.0313
6.0000	1.7188	1.7500	1.7344	-0.0459	0.0625	0.0081	0.0156
7.0000	1.7188	1.7344	1.7266	-0.0459	0.0081	-0.0190	0.0078
8.0000	1.7266	1.7344	1.7305	-0.0190	0.0081	-0.0055	0.0039
9.0000	1.7305	1.7344	1.7324	-0.0055	0.0081	0.0013	0.0020
10.0000	1.7305	1.7324	1.7314	-0.0055	0.0013	-0.0021	0.0010
11.0000	1.7314	1.7324	1.7319	-0.0021	0.0013	-0.0004	0.0005
12.0000	1.7319	1.7324	1.7322	-0.0004	0.0013	0.0004	0.0002
13.0000	1.7319	1.7322	1.7321	-0.0004	0.0004	0.0000	0.0001
14.0000	1.7319	1.7321	1.7320	-0.0004	0.0000	-0.0002	0.0001

Ex2: Bisection Method: Floating Cork

Finding the floating depth for a cork ball with:

radius=1, density=1/4 of density of water

Find the zero (between 0 and 1) of

$y=f(x)=x^3-3x^2+1$

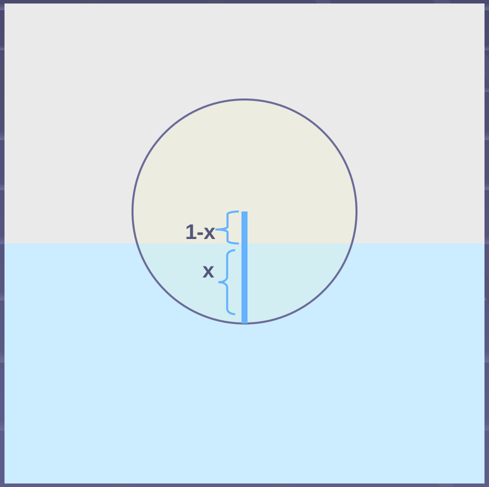
Remember:

Archimedes rule: if a solid that is lighter than a fluid is paced in the fluid, the solid will be immersed to such a depth that the weight of the solid is equal to the weight of the displaced fluid.

For ex. A spherical ball of unit radius will float in water at a depth x (the distance from the bottom of the ball to the water line) determined by ρ , the specific gravity of the ball, where ρ<1.

Ex2: Bisection Method

- Volume of the submerged segment of sphere: $V=Pi*x/6(3r^2+x^2)$
- r and x are related by the Pythagorean theorem $r^2+(1-x)^2=1 \rightarrow r^2=1-(1-x)^2$
- To find the depth at which the ball floats, solve the equation stating that the vol. of the submerged section is ρ times the vol. of the entire sphere:
- $V = Pi*x/6(3r^2+x^2) = 4\rho Pi/3$
- $V = Pi*x/6(3(1-(1-x)^2)+x^2) = 4\rho Pi/3$
- $x^3-3x^2+4*\rho=0$
- (ρ of cork is 0.25)



Ex2: Bisection Method, continued

```
%Bisection method
%Ex adopted from Fausett, 2nd ed. p.52
format short
f=inline('x.^3-3*x.^2+1');
a=0; b=1; kmax=30; tol=10^-5;
ya=f(a); yb=f(b);
if sign(ya)==sign(yb),
    error('function has the same sign at the end points'),
end
disp('   step      a        b        m       ym          bound')
for k=1:kmax,
    m=(a+b)/2; ym=f(m); iter=k; bound=(b-a)/2;
    out=[iter, a, b m, ym, bound]; disp(out)
    if abs(ym)<tol, disp('bisection has converged');break; end
    if sign(ym)~=sign(ya)
        b=m; yb=ym;
    else a=m; ya=ym;
    end
    if (iter>=kmax), disp('zero not found to desired tolerance'), end
end
```

Ex2: Bisection Method, continued

```
     step        a          b          m         ym        bound
   1.0000        0     1.0000     0.5000     0.3750     0.5000
   2.0000   0.5000     1.0000     0.7500    -0.2656     0.2500
   3.0000   0.5000     0.7500     0.6250     0.0723     0.1250
   4.0000   0.6250     0.7500     0.6875    -0.0930     0.0625
   5.0000   0.6250     0.6875     0.6563    -0.0094     0.0313
   6.0000   0.6250     0.6563     0.6406     0.0317     0.0156
   7.0000   0.6406     0.6563     0.6484     0.0112     0.0078
   8.0000   0.6484     0.6563     0.6523     0.0009     0.0039
   9.0000   0.6523     0.6563     0.6543    -0.0042     0.0020
  10.0000   0.6523     0.6543     0.6533    -0.0016     0.0010
  11.0000   0.6523     0.6533     0.6528    -0.0003     0.0005
  12.0000   0.6523     0.6528     0.6526     0.0003     0.0002
  13.0000   0.6526     0.6528     0.6527    -0.0000     0.0001
  14.0000   0.6526     0.6527     0.6526     0.0001     0.0001
  15.0000   0.6526     0.6527     0.6527     0.0001     0.0000
  16.0000   0.6527     0.6527     0.6527     0.0000     0.0000
  17.0000   0.6527     0.6527     0.6527     0.0000     0.0000
bisection has converged
```

Ex3: Bisection Method

```
function [c,yc,err,P] = bisect(f,a,b,delta)
  %---------------------------------------------------------------------------
  %BISECT   The bisection method is used to locate a root.
  % Sample calls   [c,yc,err] = bisect('f',a,b,delta) or [c,yc,err,P] = bisect('f',a,b,delta)
  % Inputs
  %   f      name of the function
  %   a      left endpoint
  %   b      right endpoint
  %   delta   convergence tolerance
  % Return
  %   c      solution: the root
  %   yc      solution: the function value
  %   err     error estimate in c
  %   P      History vector of the iterations
  % NUMERICAL METHODS: MATLAB Programs, (c) John H. Mathews 1995
  % Methods for Locating a Root, Page 61
  %---------------------------------------------------------------------------
  P = [a b];   ya = feval(f,a);  yb = feval(f,b);
  if ya*yb > 0, break, end
  max1 = 1 + round((log(b-a)-log(delta))/log(2));
  for k=1:max1,     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
    P = [P;a b];
    if b-a < delta, break, end
  end,         c  = (a+b)/2;  yc = feval(f,c);  err = abs(b-a)/2;
  echo on; clc;
  %---------------------------------------------------------------------------
  % This program implements the bisection method.
  %  Define and store f(x) in the M-file  f.m
  function y = f(x)
  y = x.*sin(x) - 1;
```

Secant Type Methods

- Bisection does not make use of the information about the shape of function y=f(x) to find its zeros
- To incorporate such information: Straight-line approximation to the function
  - Find the point where the straight-line approximation to f(x) crosses the x-axis
  - Use two initial values of the independent variable x that bracket the searched zero, i.e. xЄ[a,b] such that a<x<b
  - f(a) and f(b) should have opposite signs, i.e., the straight line through the initial points should cross the x-axis → new guess value

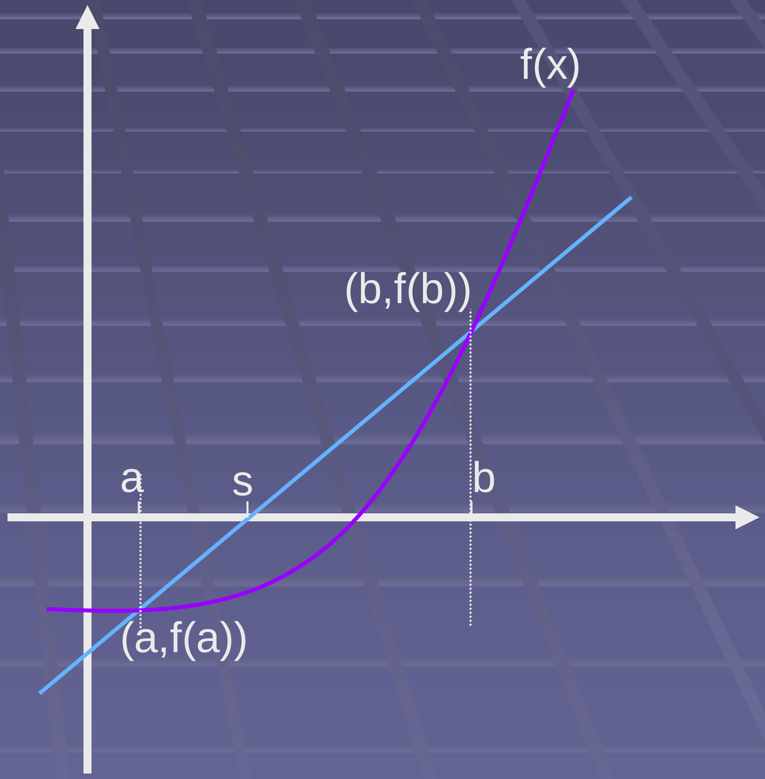
$$s=b-yb*(b-a)/(yb-ya)$$

Secant Type Methods

- Bisection does not make use of the information about the shape of function y=f(x) to find its zers
- To incorporate such information: Straight-line approximation to the function
  - Find the point where the straight-line approximation to f(x) crosses the x-axis
  - Use two initial values of the independent variable x that bracket the searched zero, i.e. xЄ[a,b] such that a<x<b
  - f(a) and f(b) should have opposite signs, i.e., the straight line through the initial points should cross the x-axis

→ new guess value

s=b-yb*(b-a)/(yb-ya)



Secant Type Methods

Regula-falsi and secant methods:

Algebraically solve the equation for the line f(x) that passes through the points (a,f(a)) and (b,f(b)), to find the point (s,0)

Equation for the line:

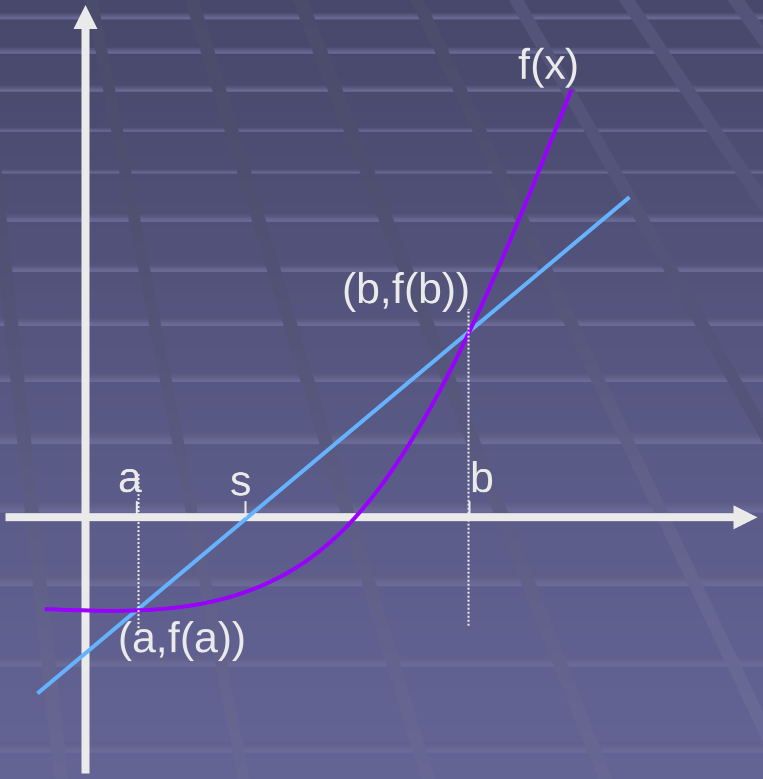
(f(x)-f(b))/(f(a)-f(b))=(x-b)/(a-b)

f(x)=f(b)+(f(a)-f(b))/(a-b)*(x-b)

Substituting x=s and f(x)=0 gives:

the new guess value

s=b-f(b)*(b-a)/(f(b)-f(a))



Regula Falsi (False Position) Method

- Proceeds as the bisection method to find the subinterval [a,s] or [s,b] in which a zero exists.
- It tests whether ya and ys have different signs
- If there is a zero in the interval [a,s],
  - Leave the value of a unchanged
  - Set b=s
- If there is no zero in the interval [a,s],
  - The zero must be in the interval [s,b]
  - Set a=s
  - Leave b unchanged

Regula Falsi (False Position) Method

```
%Function for regula falsi method
%Adopted from Fausett, 2nd ed., pg. 55
function [s,ys]=funfalsi(f,a,b,tol,itmax)
%f is an inline function
ya=f(a); yb=f(b);
if sign(ya)==sign(yb)
    error('function has the same sign at the end points')
end
disp('  step          a           b             s           ys')
for it=1:itmax
    s=b-yb*(b-a)/(yb-ya);      ys=f(s);
    out=[it,a,b,s,ys];  disp(out),
    if abs(ys)<tol, disp('regula falsi has converged'); break, end
    if sign(ys)~=sign(ya),         b=s;  yb=ys;
    else                           a=s;  ya=ys;
    end
    if (it>=itmax),  disp('zero not found to desired tolerance'), end
end
format short
```

Regula Falsi (False Position) Method Ex1: Square root

```
>> f=inline('x.^2-3'); a=1; b=2; itmax=10;
>> f
f = Inline function:
     f(x) = x.^2-3
>> [s,ys]=funfalsi(f,a,b,.00001,itmax)
      step          a          b              s           ys
    1.0000    1.0000    2.0000    1.6667   -0.2222
    2.0000    1.6667    2.0000    1.7273   -0.0165
    3.0000    1.7273    2.0000    1.7317   -0.0012
    4.0000    1.7317    2.0000    1.7320   -0.0001
    5.0000    1.7320    2.0000    1.7320   -0.0000
regula falsi has converged
s =    1.7320
ys = -6.1342e-006
```

Ex2: Regula Falsi – Floating Cork

```
>> [s,ys]=funfalsi(inline('x.^3-3*x.^2+1'),0,1,0.00001,10);
     step         a           b           s          ys
   1.0000         0      1.0000      0.5000      0.3750
   2.0000    0.5000      1.0000      0.6364      0.0428
   3.0000    0.6364      1.0000      0.6513      0.0037
   4.0000    0.6513      1.0000      0.6526      0.0003
   5.0000    0.6526      1.0000      0.6527      0.0000
   6.0000    0.6527      1.0000      0.6527      0.0000

regula falsi has converged
```

Secant Method

- Does not require its zeros to be bracketed at each step of the calculations
- Does not guarantee that there is a zero in the interval between two successive approximations
- Forms the next approximation from the previous two points

  $x_2=x_1-(x_1-x_0)/(y_1-y_0)y_1$

  At the kth stage

  $x_{k+1}=x_k-(x_k-x_{k-1})/(y_k-y_{k-1})y_k$
- Does not require any testing of the signs
- Converges more rapidly than regula falsi
- But does not guarantee convergence as it does not bracket the zero at each iteration

Secant Method

```
%Function for secant method
%Adopted from Fausett, 2nd ed., pg. 59
function [xx,yy]=funsecant(f,a,b,tol,itmax)
%f is an inline function
y(1)=f(a); y(2)=f(b); x(1)=a; x(2)=b;
Dx(1)=0;  Dx(2)=0; %Difference between successive approximations
disp('  step         x(k-1)          x(k)           x(k+1)          y(k+1)         Dx(k+1)')
for k=2:itmax
   x(k+1)=x(k)-y(k)*(x(k)-x(k-1))/(y(k)-y(k-1));
   y(k+1)=f(x(k+1));
   Dx(k+1)=x(k+1)-x(k);
   it=k-1;
   out=[it,x(k-1),x(k),x(k+1),y(k+1),Dx(k+1)]; disp(out)
   xx=x(k+1); yy=y(k+1);
   if abs(y(k+1))<tol;
       disp('secant method has converged'); break;
   end
   if (it>=itmax)
       disp('zero not found to desired tolerance')
   end
end
format short
```

Secant Method

```
>> [xx,yy]=funsecant(inline('x.^3-3*x.^2+1'),0,1,10^-8,10);
     step      x(k-1)      x(k)     x(k+1)    y(k+1)   Dx(k+1)
   1.0000         0      1.0000    0.5000    0.3750   -0.5000
   2.0000    1.0000    0.5000    0.6364    0.0428    0.1364
   3.0000    0.5000    0.6364    0.6539   -0.0033    0.0176
   4.0000    0.6364    0.6539    0.6527    0.0000   -0.0012
   5.0000    0.6539    0.6527    0.6527    0.0000    0.0000
   6.0000    0.6527    0.6527    0.6527   -0.0000    0.0000

secant method has converged
```

Ex: Secant Method
Central Angle of an Elliptical Orbit

- At any time t, what is the central angle θ of an elliptical orbit with a period of revolution P=100 days and eccentricity e=0.5?
- The function is

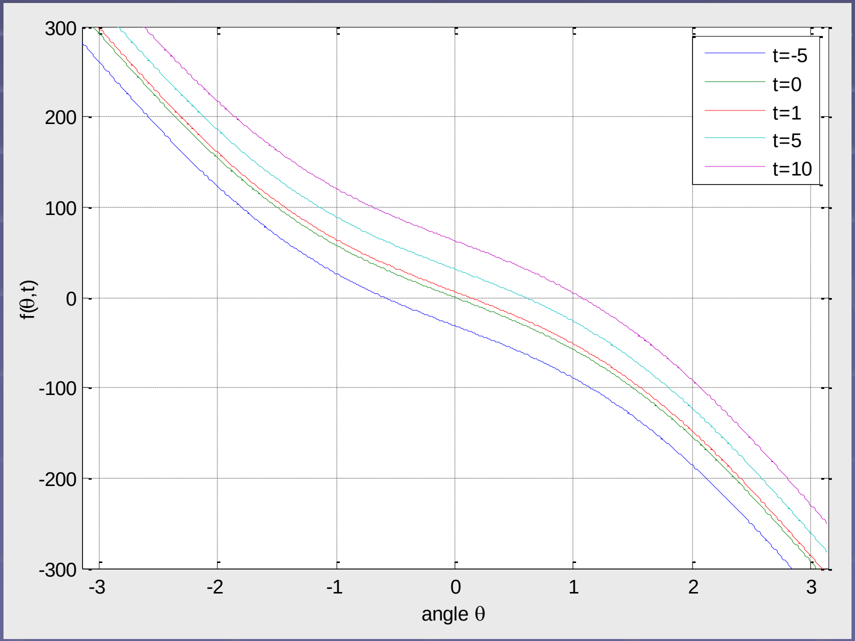
$$f(\theta,t)=2*\pi*t-P*\theta+P*e*\sin(\theta)$$

$$f(\theta,t)=2*\pi*t-100*\theta+50*\sin(\theta)$$

Ex: Secant Method
Central Angle of an Elliptical Orbit

- The function is $f(\theta,t)=2*\pi*t-P*\theta+P*e*\sin(\theta)$

```
syms t;
Per=100; ec=0.5;
angle=-pi:0.01:pi;
f=2*pi*t-Per*angle+Per*ec*sin(angle);
plot(angle,subs(f,-5),angle,subs(f,0),angle,subs(f,1),angle,subs(f,5),angle,subs(f,10) );
xlabel('angle \theta');
ylabel('f(\theta,t)');
xlim([-pi pi]);
ylim([-300 300]);
grid on;
legend('t=-5','t=0','t=1','t=5','t=10')
```

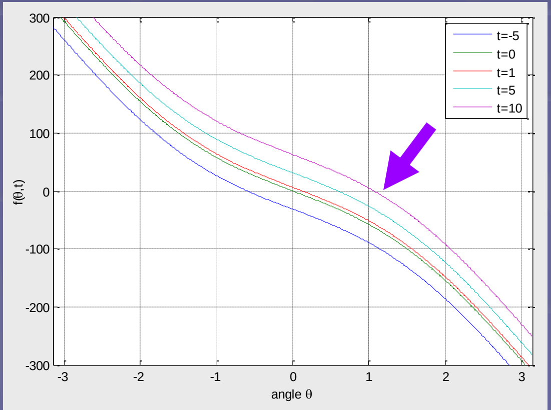


Ex: Secant Method
Central Angle of an Elliptical Orbit

- The function is f(θ,t)=2*π*t-P*θ+P*e*sin(θ)
- For t=10, the angle θ can be found using the secant method for 1< θ<2

```
>>ff=inline('20*pi-100*x+50*sin(x)');
>>[xx,yy]=funsecant(ff,1,2,10^-4,10)
```

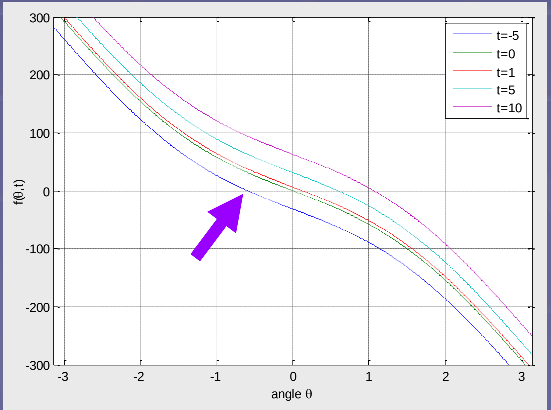
```
   step     x(k-1)     x(k)      x(k+1)     y(k+1)    Dx(k+1)
  1.0000    1.0000    2.0000    1.0508    1.1447   -0.9492
  2.0000    2.0000    1.0508    1.0625    0.2622    0.0117
  3.0000    1.0508    1.0625    1.0660   -0.0012    0.0035
  4.0000    1.0625    1.0660    1.0659    0.0000   -0.0000
secant method has converged
xx =    1.0659
yy =    1.1503e-006
```



Ex: Secant Method
Central Angle of an Elliptical Orbit

- The function is $f(\theta,t)=2*\pi*t-P*\theta+P*e*\sin(\theta)$
- For t=-5, the angle θ can be found using the secant method for $-1<\theta<0$

```
>>ff=inline('-10*pi-100*x+50*sin(x)');
>>[xx,yy]=funsecant(ff,-1,0,10^-4,10)

step        x(k-1)        x(k)       x(k+1)     y(k+1)    Dx(k+1)
   1.0000   -1.0000         0        -0.5423   -2.9889   -0.5423
   2.0000         0       -0.5423   -0.5994    0.3147   -0.0570
   3.0000   -0.5423   -0.5994   -0.5939   -0.0038    0.0054
   4.0000   -0.5994   -0.5939   -0.5940   -0.0000   -0.0001
secant method has converged
xx =   -0.5940
yy =   -4.9177e-006
```



Convergence Theorem for the Secant Method

If

- $f(x)$, $f'(x)$ and $f''(x)$ are continuous on

$$I=[x^*-e,x^*+e]$$

- $f'(x^*)\neq 0$
- the initial estimates x_0 and x_1 (in I) are sufficiently close to x^*

Then the secant method will converge

Convergence Theorem for the Secant Method

The requirements for "sufficiently close":

For all x in the interval I:

$M=\max|f''| / (2\min|f'|)$

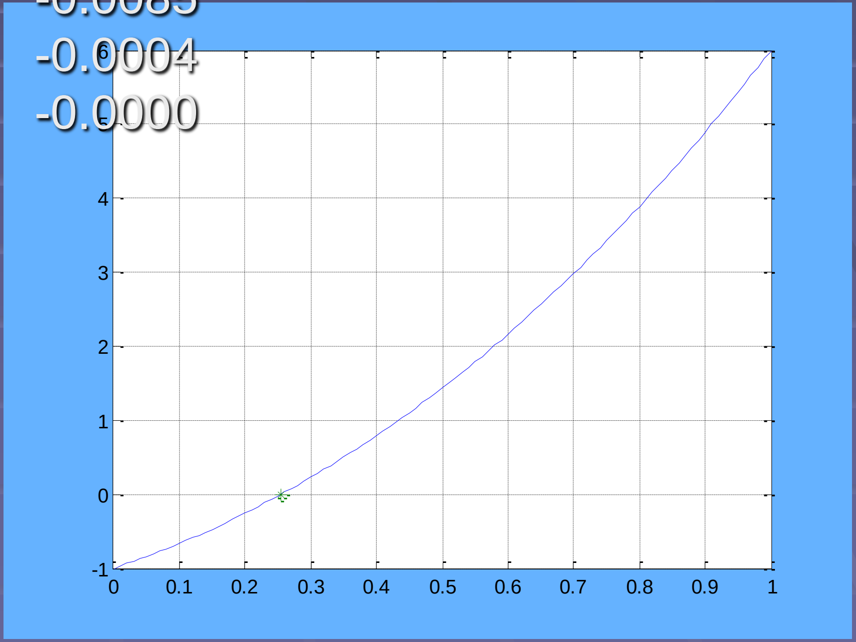
Secant method will converge if

$\max(M|x^{*}-x_0|, M|x^{*}-x_1|)<1$

- This method may converge for starting values that do not satisfy this inequality
- In general, the larger the M, the closer the starting values are to the zero!

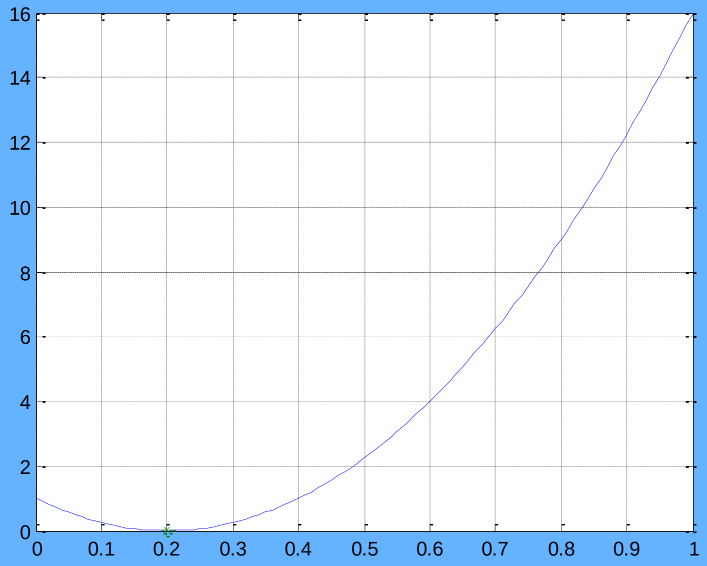
Ex: Secant Method – Fast Convergence

```
>> [xx,yy]=funsecant(inline('3*exp(x)-4*cos(x)'),1,.9,.00001,50); plot([0:.01:1],
   subs(f,[0:.01:1]),xx,yy,'*');grid on;
   step      x(k-1)     x(k)      x(k+1)    y(k+1)    Dx(k+1)
   1.0000    1.0000    0.9000    0.4558    1.1403    -0.4442
   2.0000    0.9000    0.4558    0.3207    0.3384    -0.1350
   3.0000    0.4558    0.3207    0.2638    0.0438    -0.0570
   4.0000    0.3207    0.2638    0.2553    0.0022    -0.0085
   5.0000    0.2638    0.2553    0.2549    0.0000    -0.0004
   6.0000    0.2553    0.2549    0.2549    0.0000    -0.0000

secant method has converged
```



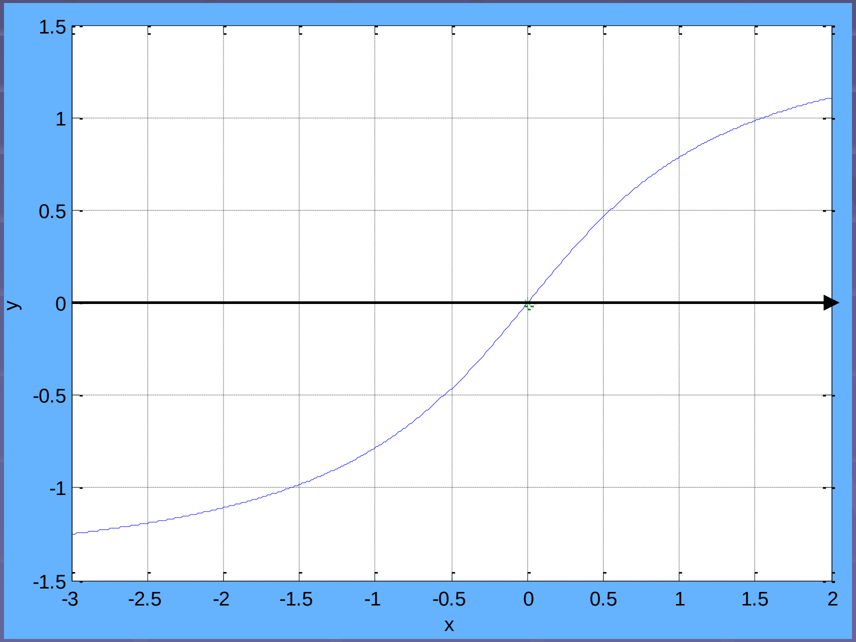
Ex: Secant Method – Slow Convergence

```
>> [xx,yy]=funsecant(inline('1-10*x+25*x.^2'),1,.9,.00001,50);
   plot([0:.01:1], subs(inline('1-10*x+25*x.^2'),[0:.01:1]),xx,yy,'*');  grid on;
   step      x(k-1)    x(k)      x(k+1)    y(k+1)    Dx(k+1)
   1.0000    1.0000    0.9000    0.5733    3.4844    -0.3267
   2.0000    0.9000    0.5733    0.4435    1.4820    -0.1299
   3.0000    0.5733    0.4435    0.3474    0.5429    -0.0961
   4.0000    0.4435    0.3474    0.2918    0.2107    -0.0556
   5.0000    0.3474    0.2918    0.2566    0.0800    -0.0352
   6.0000    0.2918    0.2566    0.2350    0.0306    -0.0216
   7.0000    0.2566    0.2350    0.2216    0.0117    -0.0134
   8.0000    0.2350    0.2216    0.2134    0.0045    -0.0083
   9.0000    0.2216    0.2134    0.2083    0.0017    -0.0051
  10.0000    0.2134    0.2083    0.2051    0.0007    -0.003
  11.0000    0.2083    0.2051    0.2032    0.0002    -0.0019
  12.0000    0.2051    0.2032    0.2019    0.0001    -0.0012
  13.0000    0.2032    0.2019    0.2012    0.0000    -0.0007
  14.0000    0.2019    0.2012    0.2007    0.0000    -0.0005
  15.0000    0.2012    0.2007    0.2005    0.0000    -0.0003

secant method has converged
```



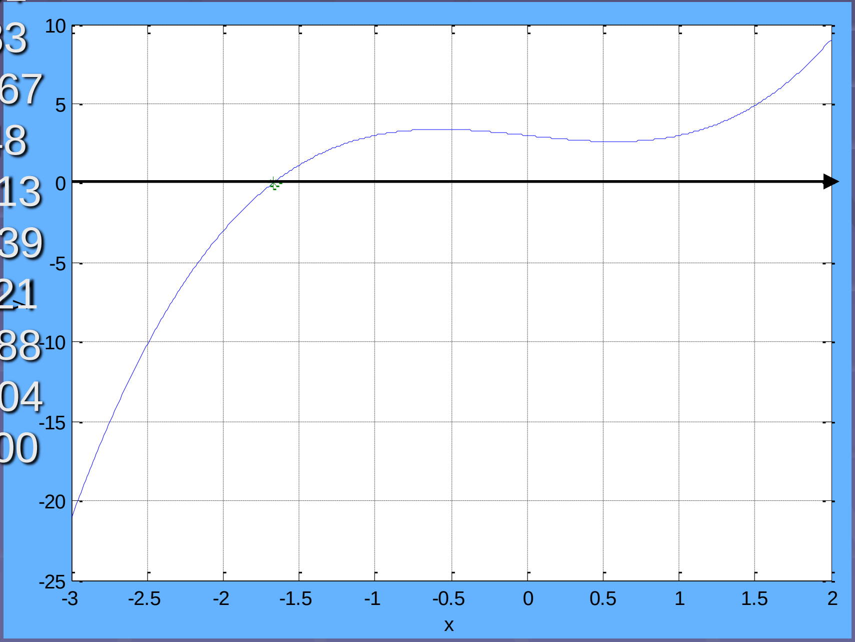
Ex: Secant Method – Oscillatory Convergence

```
>> [xx,yy]=funsecant(inline('atan(x)'),1.35,1.3,.00001,50);

     step      x(k-1)       x(k)       x(k+1)    y(k+1)    Dx(k+1)
    1.0000    1.3500    1.3000    -1.2214   -0.8847   -2.5214
    2.0000    1.3000   -1.2214     0.0180    0.0180    1.2394
    3.0000   -1.2214    0.0180    -0.0067   -0.0067   -0.0248
    4.0000    0.0180   -0.0067     0.0000     0.0000    0.0067
secant method has converged
```



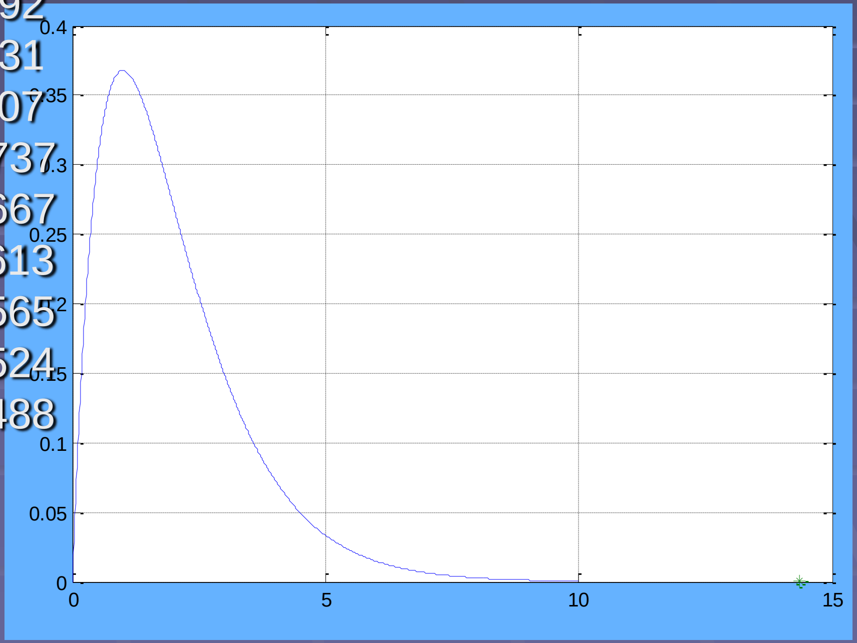
Ex: Secant Method – Convergence

```
>> [xx,yy]=funsecant(inline('3-x+x.^3'),2,1,.00001,50);

   step      x(k-1)     x(k)      x(k+1)      y(k+1)    Dx(k+1)
  1.0000    2.0000    1.0000    0.5000    2.6250   -0.5000
  2.0000    1.0000    0.5000   -3.0000  -21.0000   -3.5000
  3.0000    0.5000   -3.0000    0.1111    2.8903    3.1111
  4.0000   -3.0000    0.1111   -0.2653    3.2466   -0.3764
  5.0000    0.1111   -0.2653    3.1639   31.5077    3.4292
  6.0000   -0.2653    3.1639   -0.6592    3.3727   -3.8231
  7.0000    3.1639   -0.6592   -1.1175    2.7219   -0.4583
  8.0000   -0.6592   -1.1175   -3.0342  -21.9005   -1.9167
  9.0000   -1.1175   -3.0342   -1.3294    1.9799    1.7048
 10.0000   -3.0342   -1.3294   -1.4708    1.2894   -0.1413
 11.0000   -1.3294   -1.4708   -1.7347   -0.4850   -0.2639
 12.0000   -1.4708   -1.7347   -1.6625    0.0673    0.0721
 13.0000   -1.7347   -1.6625   -1.6713    0.0028   -0.0088
 14.0000   -1.6625   -1.6713   -1.6717   -0.0000   -0.0004
 15.0000   -1.6713   -1.6717   -1.6717    0.0000    0.0000

secant method has converged
```



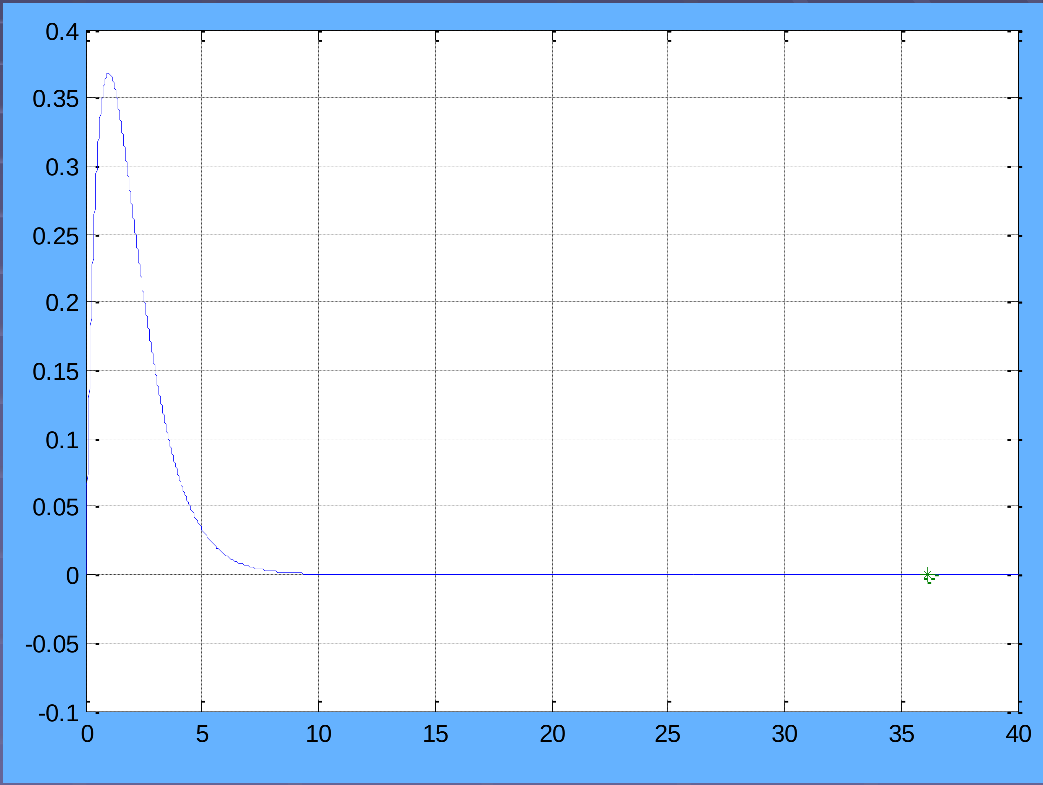
Ex: Secant Method – Nonconvergence

```
>> [xx,yy]=funsecant(inline('exp(-x)*x'),1.5,2,.00001,50);

     step      x(k-1)     x(k)     x(k+1)     y(k+1)    Dx(k+1)
   1.0000    1.5000    2.0000    4.1138    0.0672    2.1138
   2.0000    2.0000    4.1138    4.8125    0.0391    0.6987
   3.0000    4.1138    4.8125    5.7841    0.0178    0.9716
   4.0000    4.8125    5.7841    6.5949    0.0090    0.8108
   5.0000    5.7841    6.5949    7.4281    0.0044    0.8332
   6.0000    6.5949    7.4281    8.2272    0.0022    0.7992
   7.0000    7.4281    8.2272    9.0203    0.0011    0.7931
   8.0000    8.2272    9.0203    9.8010    0.0005    0.7807
   9.0000    9.0203    9.8010   10.5746    0.0003    0.7737
  10.0000    9.8010   10.5746   11.3413    0.0001    0.7667
  11.0000   10.5746   11.3413   12.1026    0.0001    0.7613
  12.0000   11.3413   12.1026   12.8591    0.0000    0.7565
  13.0000   12.1026   12.8591   13.6115    0.0000    0.7524
  14.0000   12.8591   13.6115   14.3603    0.0000    0.7488

secant method has converged
```



Ex: Secant Method – Nonconvergence

```
>> [xx,yy]=funsecant(inline('exp(-x)*x'),1.5,2,10^-14,50);
   plot([0:.01:40], subs(inline('exp(-x)*x'),[0:.01:40]),xx,yy,'*');grid on;
 step    x(k-1)   x(k) x(k+1) y(k+1) Dx(k+1)
  1.0000   1.5000   2.0000   4.1138   0.0672   2.1138
  2.0000   2.0000   4.1138   4.8125   0.0391   0.6987
  3.0000   4.1138   4.8125   5.7841   0.0178   0.9716
  4.0000   4.8125   5.7841   6.5949   0.0090   0.8108
  5.0000   5.7841   6.5949   7.4281   0.0044   0.8332
  6.0000   6.5949   7.4281   8.2272   0.0022   0.7992
  7.0000   7.4281   8.2272   9.0203   0.0011   0.7931
  8.0000   8.2272   9.0203   9.8010   0.0005   0.7807
  9.0000   9.0203   9.8010  10.5746   0.0003   0.7737
 10.0000   9.8010  10.5746  11.3413   0.0001   0.7667
 11.0000  10.5746  11.3413  12.1026   0.0001   0.7613
 12.0000  11.3413  12.1026  12.8591   0.0000   0.7565
 13.0000  12.1026  12.8591  13.6115   0.0000   0.7524
 14.0000  12.8591  13.6115  14.3603   0.0000   0.7488
 15.0000  13.6115  14.3603  15.1060   0.0000   0.7457
 16.0000  14.3603  15.1060  15.8488   0.0000   0.7428
 17.0000  15.1060  15.8488  16.5892   0.0000   0.7403
 18.0000  15.8488  16.5892  17.3272   0.0000   0.7381
 19.0000  16.5892  17.3272  18.0632   0.0000   0.7360
 20.0000  17.3272  18.0632  18.7974   0.0000   0.7342
 21.0000  18.0632  18.7974  19.5298   0.0000   0.7324
 22.0000  18.7974  19.5298  20.2607   0.0000   0.7309
 23.0000  19.5298  20.2607  20.9901   0.0000   0.7294
 24.0000  20.2607  20.9901  21.7182   0.0000   0.7281
 25.0000  20.9901  21.7182  22.4451   0.0000   0.7269
 26.0000  21.7182  22.4451  23.1708   0.0000   0.7257
 27.0000  22.4451  23.1708  23.8954   0.0000   0.7246
 28.0000  23.1708  23.8954  24.6191   0.0000   0.7236
 29.0000  23.8954  24.6191  25.3418   0.0000   0.7227
 30.0000  24.6191  25.3418  26.0636   0.0000   0.7218
 31.0000  25.3418  26.0636  26.7846   0.0000   0.7210
 32.0000  26.0636  26.7846  27.5048   0.0000   0.7202
 33.0000  26.7846  27.5048  28.2242   0.0000   0.7195
 34.0000  27.5048  28.2242  28.9430   0.0000   0.7188
 35.0000  28.2242  28.9430  29.6611   0.0000   0.7181
 36.0000  28.9430  29.6611  30.3785   0.0000   0.7175
 37.0000  29.6611  30.3785  31.0954   0.0000   0.7169
 38.0000  30.3785  31.0954  31.8117   0.0000   0.7163
 39.0000  31.0954  31.8117  32.5274   0.0000   0.7158
 40.0000  31.8117  32.5274  33.2427   0.0000   0.7152
 41.0000  32.5274  33.2427  33.9574   0.0000   0.7147
 42.0000  33.2427  33.9574  34.6717   0.0000   0.7143
 43.0000  33.9574  34.6717  35.3855   0.0000   0.7138
 44.0000  34.6717  35.3855  36.0989   0.0000   0.7134
secant method has converged
```



Convergence

- An iterative numerical method converges with order p if:

$$|x_k - x^*| \sim \lambda |x_{k-1} - x^*|^p$$

or

$$\lim_{(k\to\infty)} (|x_k - x^*| \,/\, |x_{k-1} - x^*|^p) = \lambda$$

for large values of k, where the true zero is x^* and $\lambda > 0$ is the asymptotic error constant (rate of convergence).

- If $p=1$ and $\lambda < 1$, the method is called linearly convergent
- If $p>1$ and $\lambda > 0 \rightarrow p=2$ quadratic , $p=3$ cubic convergence

Convergence of Regula Falsi

- If a function does not change its concavity on the interval $[a_k,b_k]$, then either the point $(b_k,y(b_k))$ will not change during the regula falsi iterations (k+1,…), or the point $(a_k,y(a_k))$ does not change. Then the convergence is linear.
- Some graphical analysis is necessary to find the appropriate subinterval to bracket the zero
- A choice of the subinterval must be made at each iteration.

Convergence of Secant Method

- Does not bracket the zero at each iteration, so does not guarantee convergence
- When it converges, it is faster than bisection or regula falsi
- Rate of convergence is $p=(1+5^{1/2})/2\sim1.62$, i.e. convergence is faster than linear, but less than quadratic
- Asymptotic error constant is:

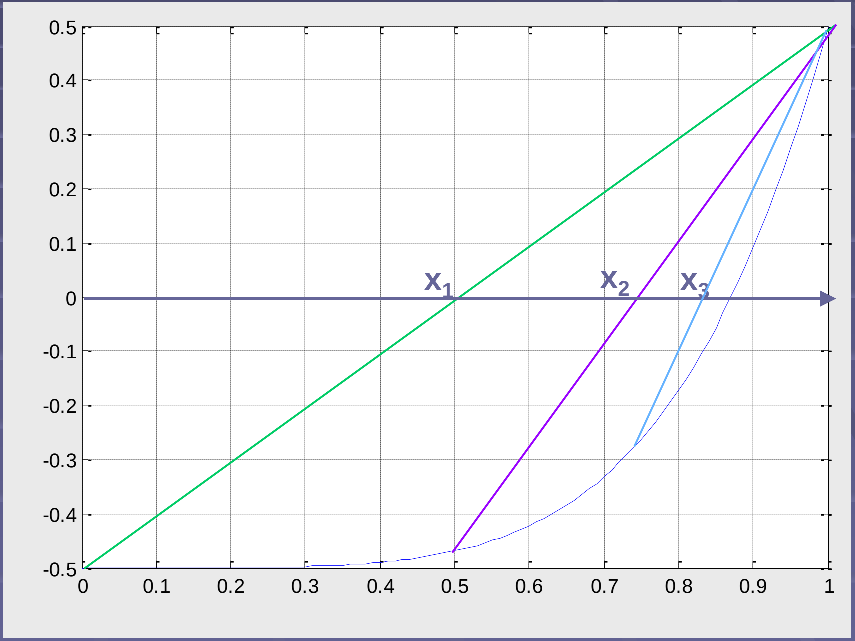
$\lambda=|f''(x^*)/(2f'(x^*)|^b$, where $b=(5^{1/2}-1)/2$

Challenging Problem: Regula Falsi If a function is relatively flat near a zero

$y=x^5-0.5$ Find the positive real zero

```
>> ff=inline('x^5-0.5');
>>[xx,yy]=funfalsi(ff,0,1,10^-4,10)

 step         a          b          s          ys
1.0000        0       1.0000     0.5000    -0.4688
2.0000     0.5000     1.0000     0.7419    -0.2752
3.0000     0.7419     1.0000     0.8335    -0.0976
4.0000     0.8335     1.0000     0.8607    -0.0276
5.0000     0.8607     1.0000     0.8680    -0.0073
6.0000     0.8680     1.0000     0.8699    -0.0019
7.0000     0.8699     1.0000     0.8704    -0.0005
8.0000     0.8704     1.0000     0.8705    -0.0001
9.0000     0.8705     1.0000     0.8705    -0.0000
 regula falsi has converged
 xx =     0.8705
 yy =   -3.1687e-005
```

```
>> x=0:0.01:1; plot(x,x.^5-0.5); grid on
```



First three approximations of x for y using the Regula Falsi method

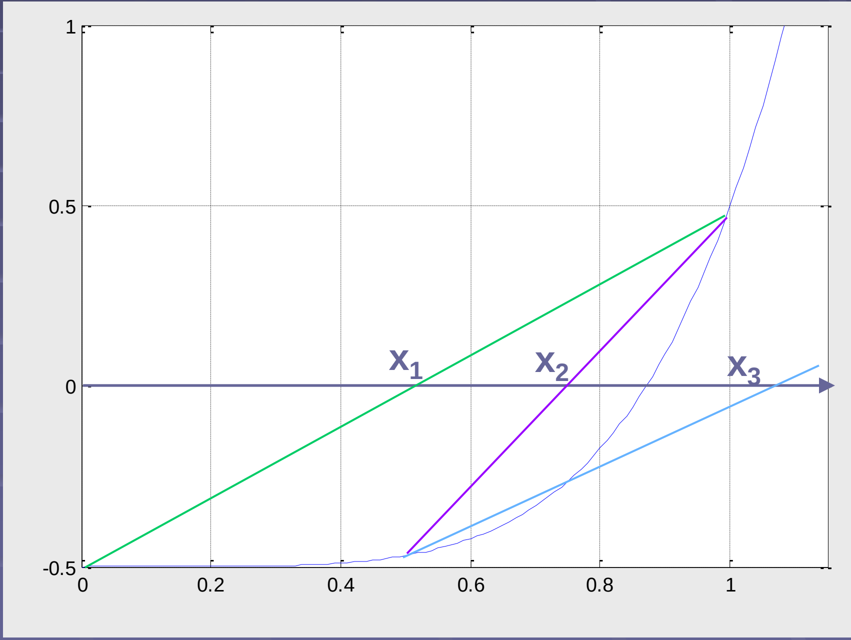
Challenging Problem: Secant Method If a function is relatively flat near a zero

$y=x^5-0.5$ Find the positive real zero

```
>> ff=inline('x^5-0.5');
>>[xx,yy]=funsecant(ff,0,1,10^-4,10)
```

```
step      x(k-1)     x(k)      x(k+1)    y(k+1)    Dx(k+1)
1.0000         0    1.0000    0.5000   -0.4688   -0.5000
2.0000    1.0000    0.5000    0.7419   -0.2752    0.2419
3.0000    0.5000    0.7419    1.0859    1.0098    0.3439
4.0000    0.7419    1.0859    0.8156   -0.1391   -0.2703
5.0000    1.0859    0.8156    0.8483   -0.0607    0.0327
6.0000    0.8156    0.8483    0.8736    0.0089    0.0253
7.0000    0.8483    0.8736    0.8704   -0.0005   -0.0032
8.0000    0.8736    0.8704    0.8705   -0.0000    0.0002

secant method has converged

xx =    0.8705
yy =   -3.2437e-006
```

```
>> x=0:0.01:1; plot(x,x.^5-0.5);
grid on; xlim([0 1.15]); ylim([-.5 1])
```



First three approximations of x for y using the Regula Falsi method

Challenging Problem: Secant Method
If a function is relatively flat near a zero

$y=x^5-0.5$ Find the positive real zero

The third approximation is outside the original interval that contains the root.

Yet after 8 iterations method has converged to a solution.

If starting values are changed:

$x_0=1$ and $x_1=0$

Then $x_2=0.5$ but x_3 is based on a straight line with almost zero slope, the line determined by points (0,-0.5) and (0.5,-0.46875)

There are extreme oscillations before the method converges to the correct value

```
>> [xx,yy]=funsecant(ff,1,0,10^-4,20)
   step      x(k-1)     x(k)    x(k+1)    y(k+1)   Dx(k+1)
  1.0000    1.0000        0    0.5000   -0.4688    0.5000
 1.0e+004 *
  0.0002         0    0.0001    0.0008    3.2767    0.0008
  3.0000    0.5000    8.0000    0.5001   -0.4687   -7.4999
  4.0000    8.0000    0.5001    0.5002   -0.4687    0.0001
  5.0000    0.5001    0.5002    1.9981   31.3459    1.4979
  6.0000    0.5002    1.9981    0.5223   -0.4611   -1.4758
  7.0000    1.9981    0.5223    0.5437   -0.4525    0.0214
  8.0000    0.5223    0.5437    1.6643   12.2690    1.1206
  9.0000    0.5437    1.6643    0.5835   -0.4323   -1.0808
 10.0000    1.6643    0.5835    0.6203   -0.4081    0.0368
 11.0000    0.5835    0.6203    1.2410    2.4432    0.6207
 12.0000    0.6203    1.2410    0.7092   -0.3206   -0.5318
 13.0000    1.2410    0.7092    0.7709   -0.2278    0.0617
 14.0000    0.7092    0.7709    0.9223    0.1672    0.1514
 15.0000    0.7709    0.9223    0.8582   -0.0346   -0.0641
 16.0000    0.9223    0.8582    0.8691   -0.0040    0.0110
 17.0000    0.8582    0.8691    0.8706    0.0001    0.0014
 18.0000    0.8691    0.8706    0.8706   -0.0000   -0.0000
secant method has converged
xx =    0.8706    yy =   -3.7642e-007
```

Newton’s (Newton-Raphson) Method

- Uses straight line approximation to the function (similar to regula falsi and secant methods)
- The line is a ***tangent*** to the curve (not a secant)
- The next approximation to the zero is the value of x where the tangent crosses the x axis
- Derivative of the function should be known

Newton's (Newton-Raphson) Method

- Finds a root of a nonlinear equation
- Applicable on the complex domain to find a complex root
- Derived from Taylor expansion:

First order truncated Taylor expansion of $f(x)$ about an initial estimate x_0 is:

$f(x) \sim f(x_0)+f'(x_0)(x-x_0)$

Then an approximation for the root is:

$x_1=(f(x_1)-f(x_0))/f'(x_0)+x_0 \rightarrow x_1=x_0-f(x_0)/f'(x_0)$

In general:

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

Newton's Method
Finding the square root of 3/4

```
>> x=[0:.01:1];  plot(x,4*x.^2-3, x,4*x-4),  grid on
```

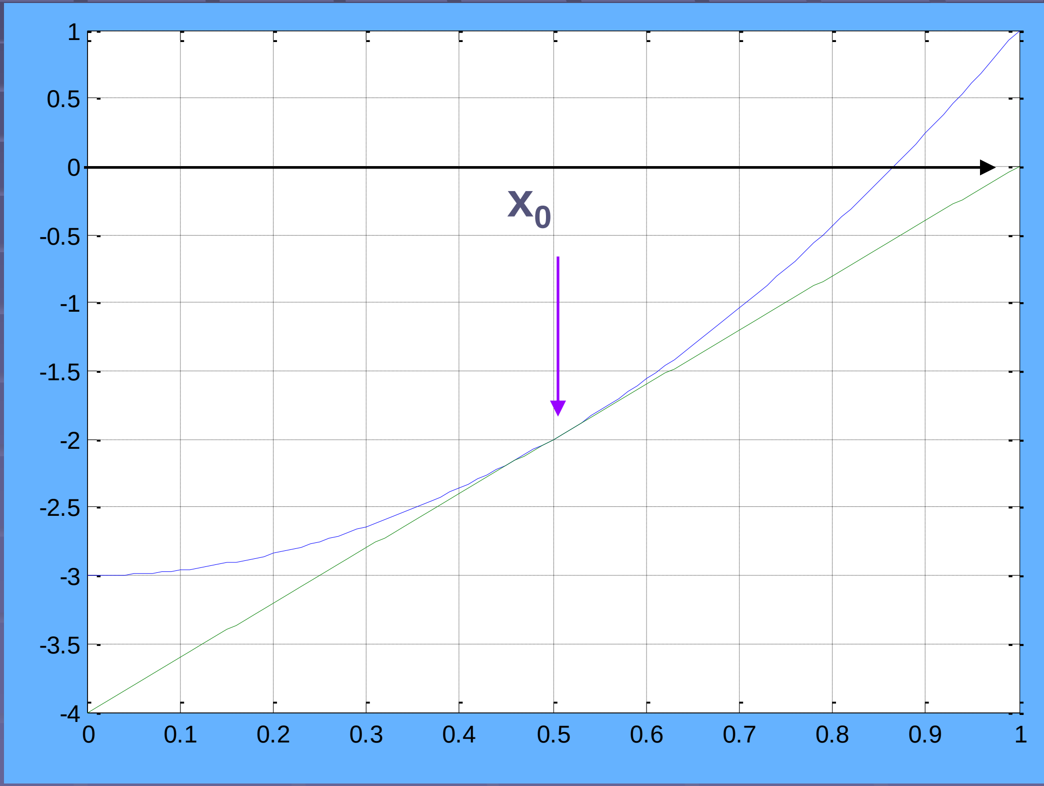
Graph of $f(x)=4x^2-3$ and the tangent line at $x=0.5$

Tangent line at (x_0,y_0):

$y-y_0=f'(x_0)(x-x_0)$

Next approximation to the zero of $f(x)$ is the value of x where the tangent crosses the x-axis.

Substitute $(x_1,0)$ in this equation and solve for x_1



Ex: Newton's Method

```
%Newton's solution: Iterative method to solve for the zeros of a nonlinear
    equation
%Adopted from Fausett 2nd ed. p.66
function [x,y]=funNewton(fun,fund,x1,tol,kmax)
%fun: Inline function or m-file function
%dfun: derivative function (inline or m-file)
%x1: starting guess value
x(1)=x1;   y(1)=feval(fun,x(1));   yd(1)=feval(fund,x(1));
for k=2:kmax
   x(k)=x(k-1)-y(k-1)/yd(k-1);      y(k)=feval(fun,x(k));
   if abs(x(k)-x(k-1))<tol
      disp('Newton method has converged'); break;
   end
   yd(k)=feval(fund,x(k));
end
if k>=kmax, disp('zero not found to desired tolerance'); end
n=length(x);  k=1:n;
out=[k' x' y'];  disp('   step            x            y'), disp(out)
```

Ex: Newton's Method

```
>> [x,y]=funNewton(inline('4*x.^2-3'),inline('8*x'),0.5,10^-5,30)

Newton method has converged
       step          x           y
    1.0000    0.5000   -2.0000
    2.0000    1.0000    1.0000
    3.0000    0.8750    0.0625
    4.0000    0.8661    0.0003
    5.0000    0.8660    0.0000
    6.0000    0.8660   -0.0000

x =    0.5000    1.0000    0.8750    0.8661    0.8660    0.8660
y =   -2.0000    1.0000    0.0625    0.0003    0.0000   -0.0000
```

Convergence of Newton’s Method

- **Theorem: Let x* be a root of f(x)=0, where f is a continuous function on an interval containing x*, and we suppose that | f’(x*)| > 0, then Newton's iteration will converge to x* if the starting point x_0 is close enough to x***

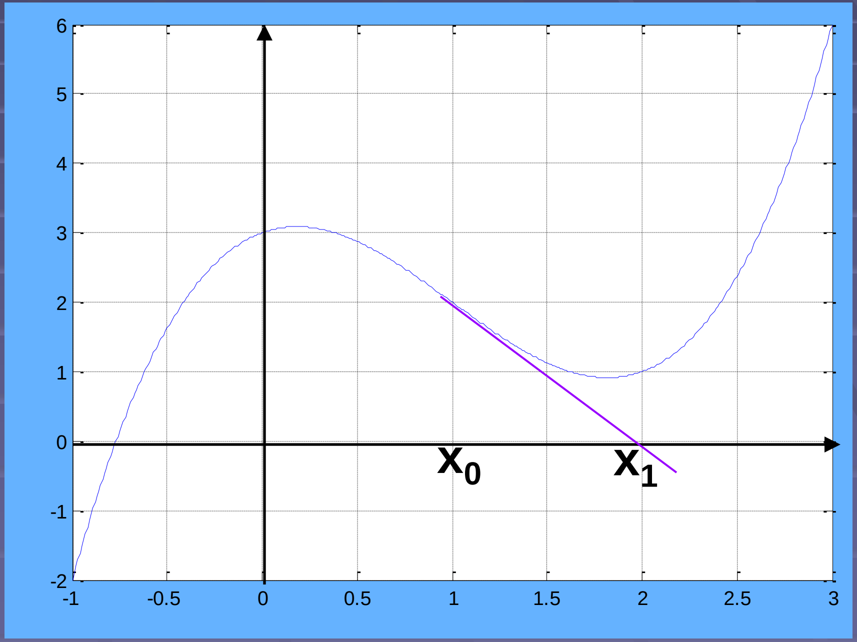
- **This theorem insures that Newton's method will always converge if the initial point is sufficiently close to the root and if this root is not singular (that is f’(x*) is non zero). This process has the local convergence property**

Oscillations in Newton's Method

- **Newton's method can give oscillatory results for some functions and some initial estimates**
- **Ex: $y=x^3-3x^2+x+3$**

- **If we do not consider the graph and guess $x_0=1$, the derivative line will cross the x axis at $x_1=2$ …**

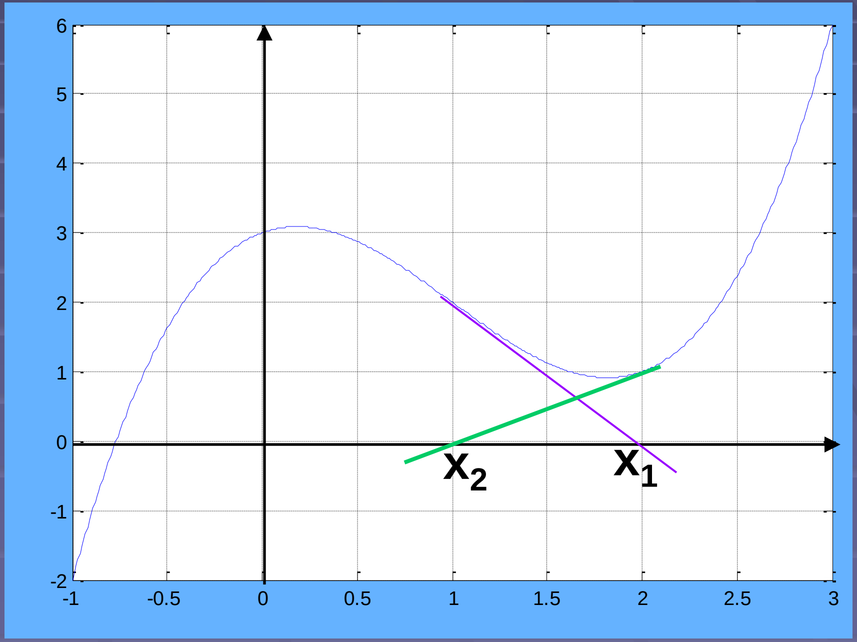
```
>> x=[-1:.01:3];
>> plot(x,x.^3-3*x.^2+x+3); grid on
```



Oscillations in Newton's Method

- … and if we take $x_1=2$, and draw the tangent line to the curve, it will cross the x axis at $x_2=1$

```
>> x=[-1:.01:3];
>> plot(x,x.^3-3*x.^2+x+3); grid on
```



```
>> f=inline('x.^3-3*x.^2+x+3');
   fd=inline('3*x.^2-6*x+1');
>> [x,y]=funNewton(f,fd,1,.001,5);
zero not found to desired tolerance
   step   x    y
     1    1    2
     2    2    1
     3    1    2
     4    2    1
     5    1    2
```

The process will oscillate between 1 and 2 !

Newton’s Method: Repeated Roots

- Loses its quadratic convergence in the case of a repeated root
- Can be modified if the order of the repeated root is known (or can be estimated)
- For an m^{th} order root:

$x = g(x) = x - f(x)/f'(x) \rightarrow x = g(x) = x - mf(x)/f'(x)$

Newton's Method: Repeated Roots

- Proof:

For an mth order root:

$x = g(x) = x - f(x)/f'(x) \rightarrow x = g(x) = x - mf(x)/f'(x)$

Because

$g'(x) = f(x)f''(x)/(f'(x))^2$

At x^*, if $f'(x^*) \neq 0$, then $g'(x) = 0$

But for a double root ($m=2$), $f(x) = (x - x^*)^2 h(x)$,

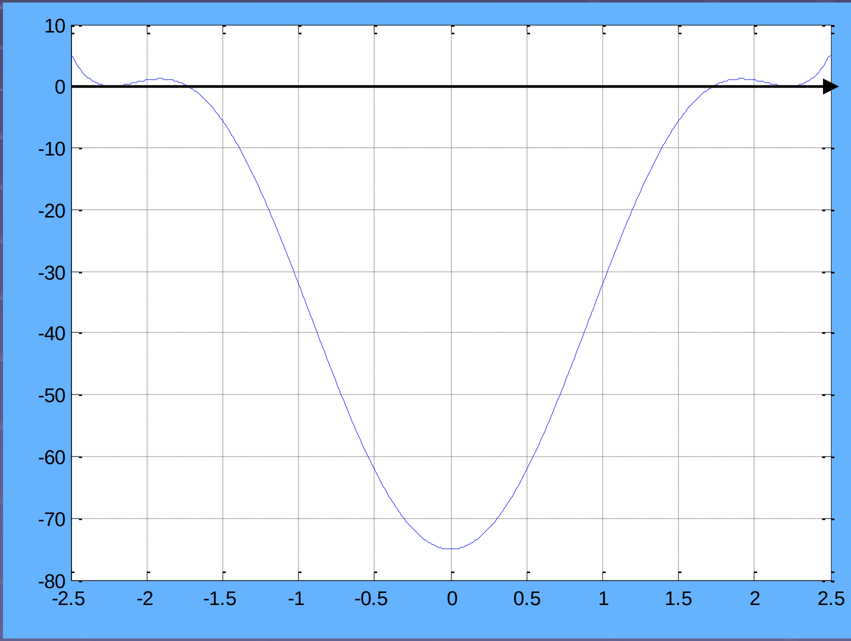
$g'(x) = 1 - 1/2 \neq 0$

Newton’s Method: Repeated Roots

```
%Newton's solution: Iterative method
%to solve for a zero of a nonlinear equation with double roots
%Adopted from Fausett 2nd ed. p.66
function [x,y]=funNewtond(f,fd,m,x0,tol,kmax)
%fun: Inline function or m-file function
%dfun: derivative function (inline or m-file)
%x1: starting guess value
x(1)=x0;     y(1)=f(x(1));
yd(1)=fd(x(1)); C(1)=0;
disp('    step  x(k)   y(k)   Dx  C')
for k=1:kmax
    x(k+1)=x(k)-m*y(k)/yd(k);
    y(k+1)=f(x(k+1));     yd(k+1)=fd(x(k+1));
    Dx(k)=x(k+1)-x(k);
    if k>1,   C(k)=Dx(k)/Dx(k-1)^2;   end
    if abs(Dx(k))<tol,   disp('Newton method has converged');  break;   end
   iter=k-1; out=[iter, x(k), y(k), Dx(k), C(k)]; disp(out);
   if iter>=kmax+1, disp('zero not found to desired tolerance'); end
end
```

Ex: Newton's Method-Repeated Roots

```
>> f=inline('(x.^2-5).^2*(x.^2-3)');
  fd=inline('4*x*(x.^2-5)*(x.^2-3)-2*x*(x.^2-
    5).^2');

>> [x,y]=funNewtond(f,fd,2,2,.0001,10);

    step      x(k)      y(k)      Dx        C
     0       2.0000    1.0000    0.1667     0
  1.0000     2.1667    0.1582    0.0647   2.3285
  2.0000     2.2313    0.0009    0.0047   1.1235

Newton method has converged
```



```
>> x=[-2.5:.01:2.5];

>> plot(x,((x.^2-5).^2.*(x.^2-3))), grid on
```

Muller’s Method

- Is an extension to the methods that use a linear approximation to the function to find its zero
- Is a quadratic approximation to the function
- Can give complex zeros even if the initial guess value (estimate) is real
- Less sensitive to starting values than Newton’s method
- Formulas are relatively more complicated compared to the previous methods:
  - Use 3 points on the function (or 2 points that bracket the zero and the midpoint of the interval)
  - Find the quadratic equation that passes through these points
  - Find the zeros of the quadratic

Muller's Method

- Parabola passing though three points (x_1,y_1), (x_2,y_2), (x_3,y_3) :

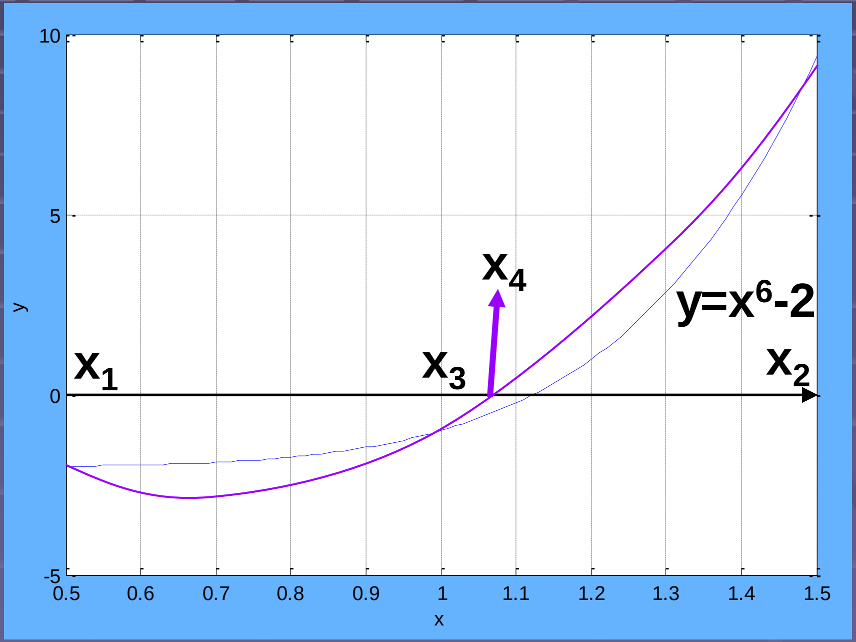
$y=y_3+c_2(x-x_3)+d_1(x-x_3)(x-x_2)$

Find the coefficients:

$c_1=(y_2-y_1)/(x_2-x_1)$

$c_2=(y_3-y_2)/(x_3-x_2)$

$d_1=(c_2-c_1)/(x_3-x_1)$



Let $s=c_2+d_1(x_3-x_2)$

Solving for the zero closest to x_3

gives the next approximation to the desired zero

$x_4=x_3-2y_3/[s+\text{sign}(s)(s^2-4y_3d_1)^{1/2}]$

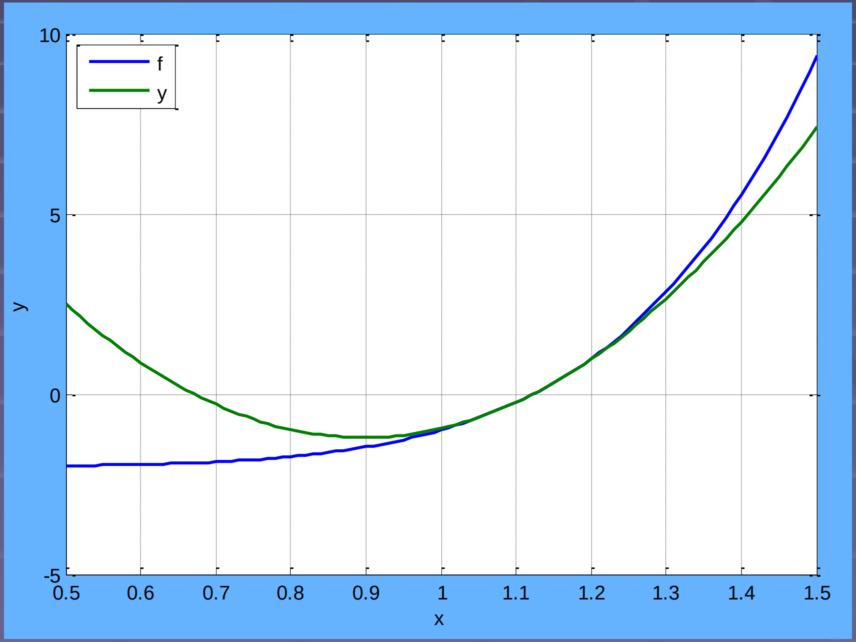
Ex: Muller's Method

```
% Muller's iterative solution to find the zeros of a nonlinear equation
% f: String containing name of function
% x1,x2 Starting estimates, x3=(x1+x2)/2

function [x,y]=funMuller(f,x1,x2,tol,kmax)
x(1)=x1;    x(2)=x2;     x(3)=(x(2)+x(1))/2;
y(1)=feval(f,x1);    y(2)=feval(f,x2);    y(3)=feval(f,x(3));
c(1)=(y(2)-y(1))/(x(2)-x(1));
disp('    step         x         y');
disp([1 x(1) y(1); 2 x(2) y(2)]);
for k=3:kmax
    c(k-1)=(y(k)-y(k-1))/(x(k)-x(k-1));
    d(k-2)=(c(k-1)-c(k-2))/(x(k)-x(k-2));
    s=c(k-1)+(x(k)-x(k-1))*d(k-2);
    x(k+1)=x(k)-2*y(k)/(s+sign(s)*sqrt(s^2-4*y(k)*d(k-2)));
    y(k+1)=feval(f,x(k+1));
    out=[k, x(k),  y(k)]; disp(out);
    if abs(x(k+1)-x(k))<tol,   disp('Muller method has converged'); break; end
end
if k>=kmax, disp('zero not found to desired tolerance'); end
z=.5:.01:1.5;     fz=z.^6-2;      yz=y(k)+c(k-1).*(z-x(k))+d(k-2).*(z-x(k)).*(z-x(k-1));
plot(z,fz,z,yz); grid on; xlabel('x'); ylabel('y'); legend('f','y',2);  ylim([-5 10]);
```

Ex: Muller's Method

```
>> f=inline('x^6-2');   x1=0.5; x2=1.5;  tol=10^-5; kmax=50;
    [x,y]=funMuller(f,x1,x2,tol,kmax);
```

```
  step         x         y
1.0000    0.5000   -1.9844
2.0000    1.5000    9.3906
3         1         -1
4.0000    1.0779   -0.4317
5.0000    1.1170   -0.0576
6.0000    1.1225    0.0008
7.0000    1.1225   -0.0000
```



Muller method has converged

Minimization of a Function of One Variable:

- Finding the minimum using methods that
  1. Require only function evaluations
  2. Require information on the derivative of the function

Simplest minimization approach without using derivative of function:

Golden-Section method

(similar to bisection method)

Golden-Section Search

- Finds the minimum of the function in an interval
- To bracket a minimum point, we need:

Bracketing triplet: (a,t,b)

where $a<t<b$ s.t. $f(t)<f(a)$ and $f(t)<f(b)$

then f(x) has a minimum on interval (a,b)

Golden-Section Search

A new point t_2 is either in $[a, t_1]$ or $[t_1,b]$

Compare function values $f(t_1)$ and $f(t_2)$

Reduce the width of the interval by

$(1-r)\sim 0.618 = 1/r^*$

Golden ratio $= r^* = (1+\sqrt{5})/2 = 1.6180$

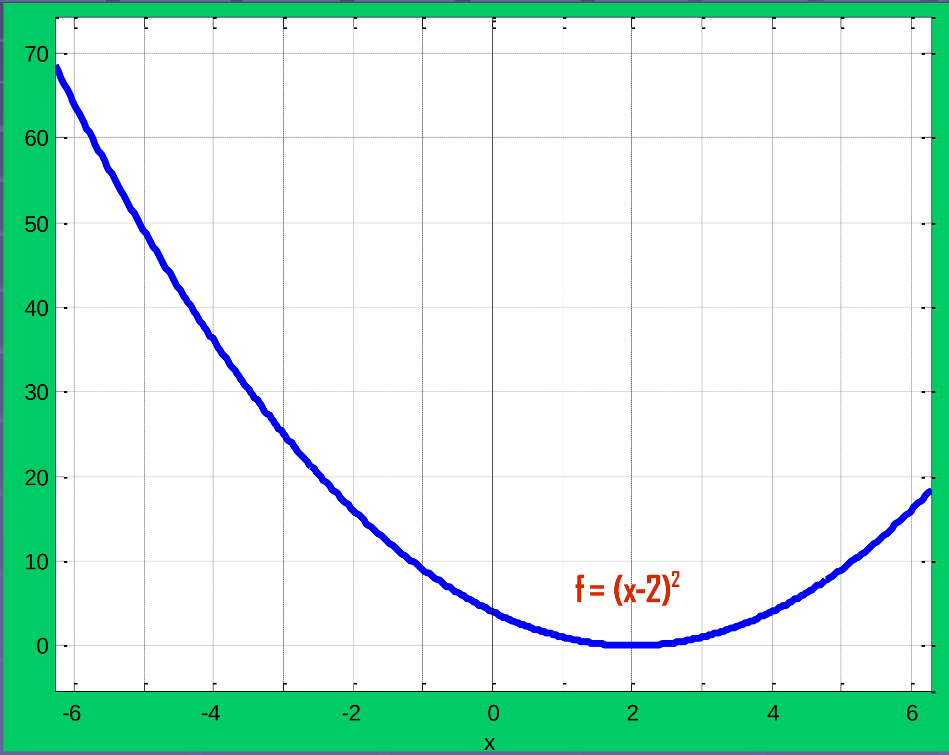
Ex: Golden-section search

```
>> ezplot('(x-2)^2'), grid on
```

f → min on (1,3)

Starting triplet:

$(a,t_1,b) \rightarrow (1,1.5,3)$



Ex: Golden-section search

```
%Golden-Section search method application: Adopted from
%http://www-users.math.umd.edu/~jec/working/workingmfiles/golden.m
f = inline('(x-2).^2');
N=10;  a0=1;  b0=3;  r=(sqrt(5)-1)/2;  alist=zeros(N,1);    blist=zeros(N,1);
a=a0;    b=b0;     t1=a+(1-r)*(b-a);    t2=b-(1-r)*(b-a);    f1=f(t1);    f2=f(t2);
for  n=1:N
        if  f1 < f2
        B=t2;    t2=t1;     t1=a+(1-r)*(b-a);    f2=f1;    f1=f(t1);
        else   a=t1;   t1=t2;   t2=b-(1-r)*(b-a);   f1=f2;    f2= f(t2);
        end
alist(n)=a;    blist(n)=b;
end
disp(' a   b   f(a)   f(b)    b-a ')    disp(' ')
alist=[a0;alist];    blist=[b0; blist];
[alist, blist, f(alist), f(blist), blist-alist]
```