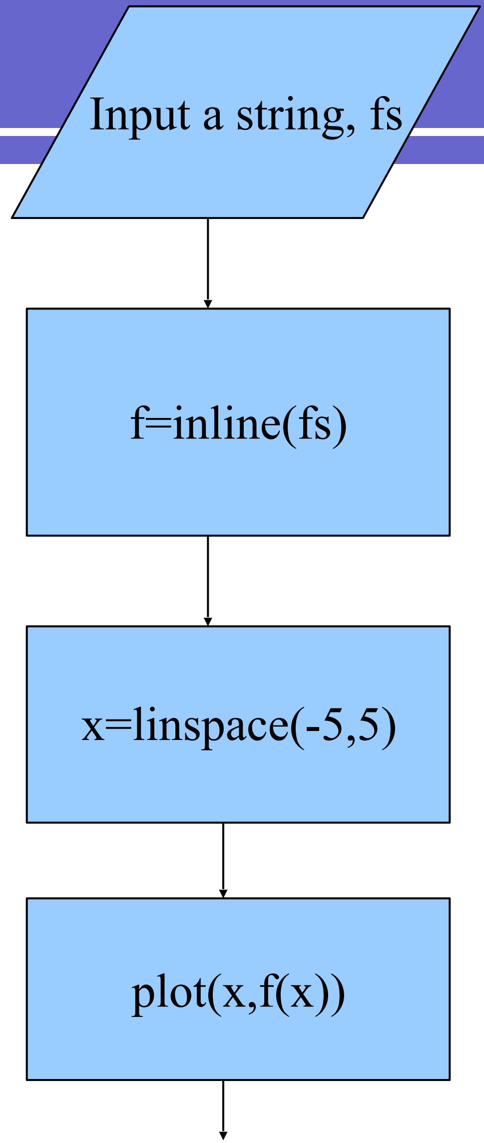
Lecture 3

- Inline function
- Symbolic differentiation
- Taylor series
- Root finding
- Newton method
- Bisection method

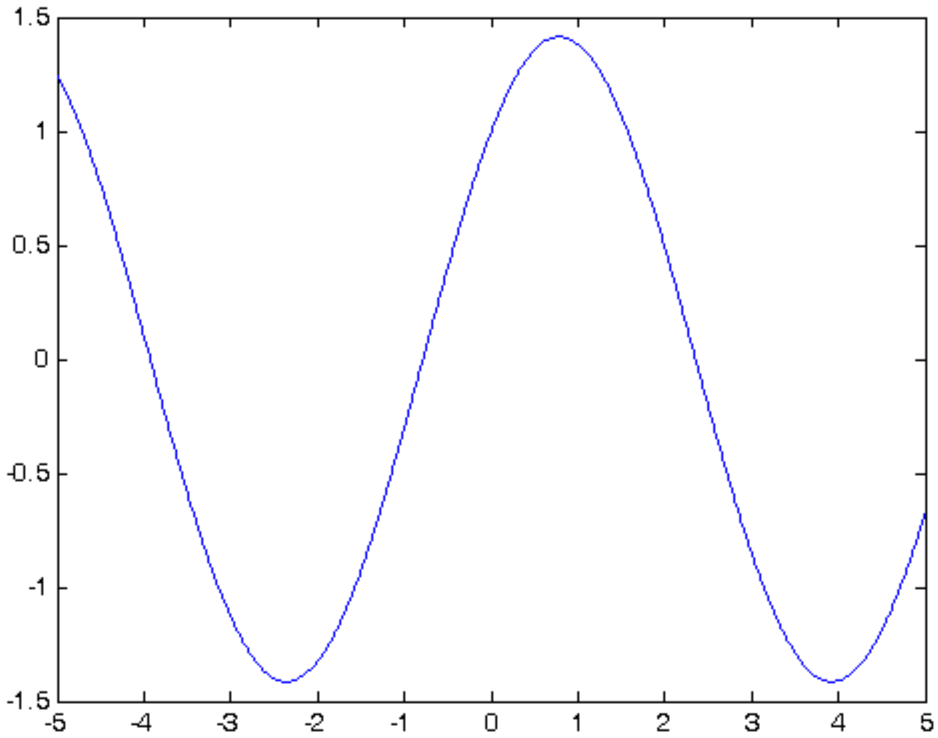
```
>> g=inline('x^2+1')

g =

     Inline function:
     g(x) = x^2+1

>> g(2)

ans =

     5
```

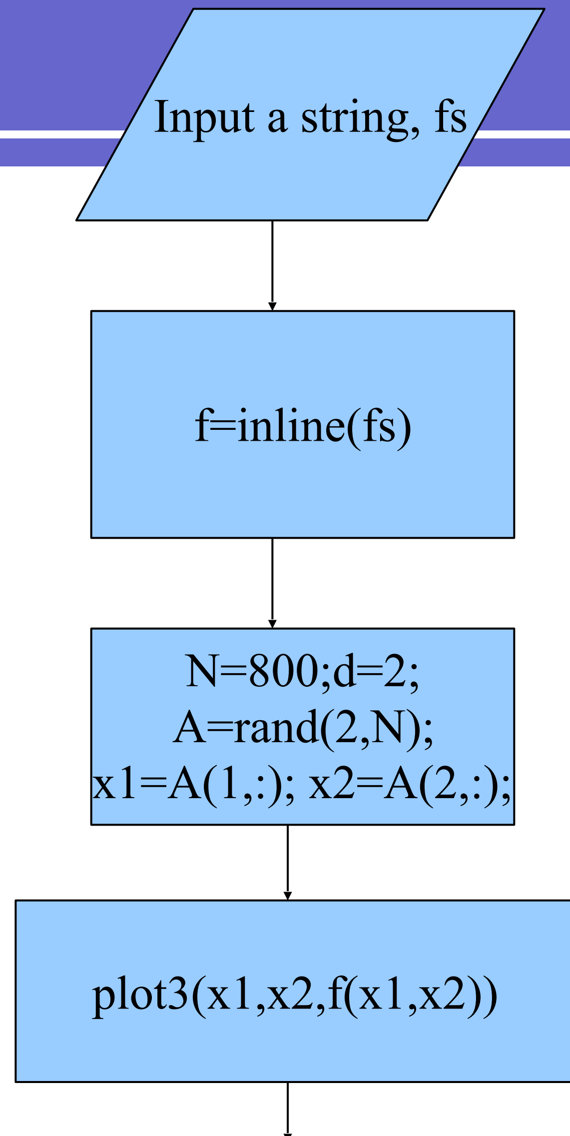

Input a string, fs
f=inline(fs)
x=linspace(-5,5)
plot(x,f(x))

Demo 1

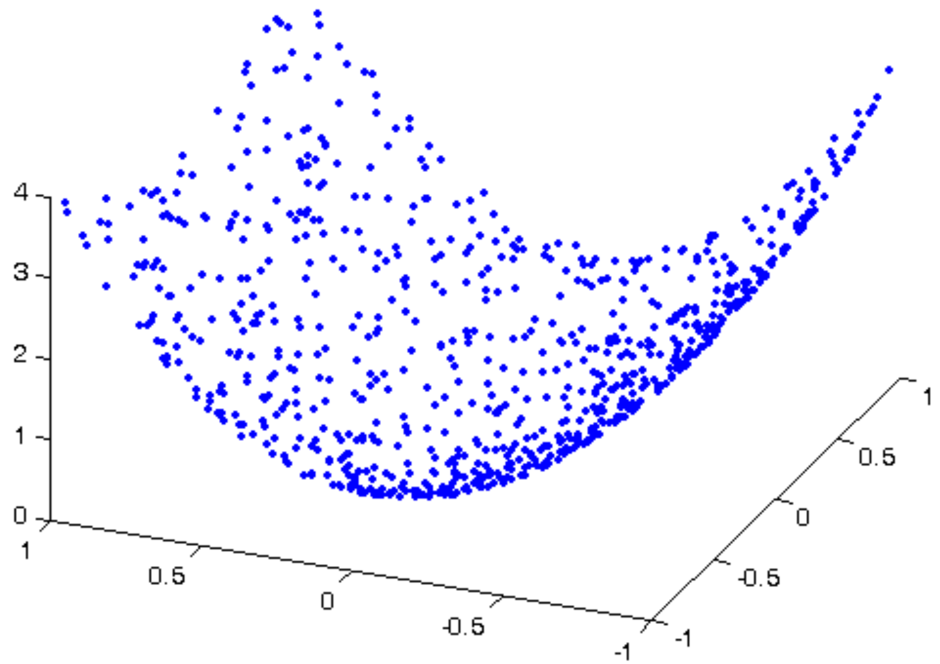
```
% Create an inline function
fs= 'tanh(x)';
f=inline(fs);
range=5;
x=-range:0.01:range;
y=f(x);
plot(x,y);
```

cos(x)




Input a string, fs
f=inline(fs)
N=800;d=2;
A=rand(2,N);
x1=A(1,:); x2=A(2,:);
plot3(x1,x2,f(x1,x2))

A sample from function surface



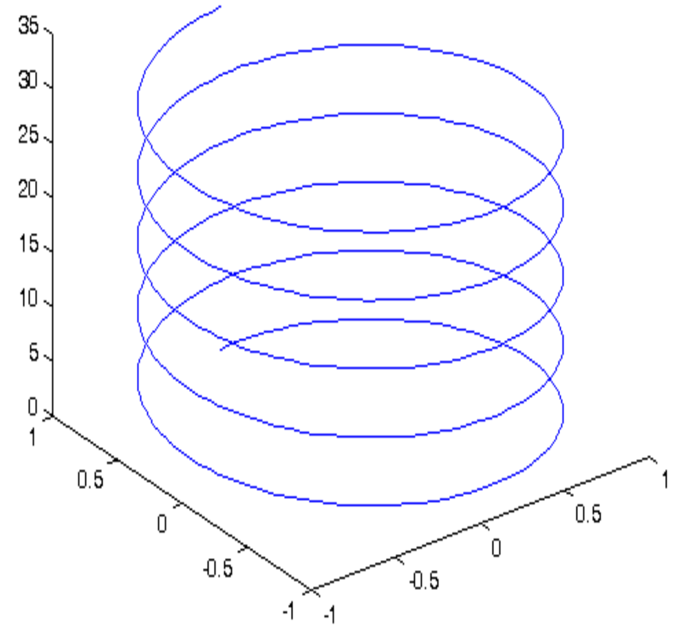
Demo 2

```
% Inline function creation
% The created inline function has two input arguments
% Plot a sample from the function surface
fs='x.^2+3*y.^2';
fxy=inline(fs);
a=rand(2,800)*2-1;
x=a(1,:);y=a(2,:);
plot3(x,y,fxy(x,y),'.');
```

3D plot

- 3d plot

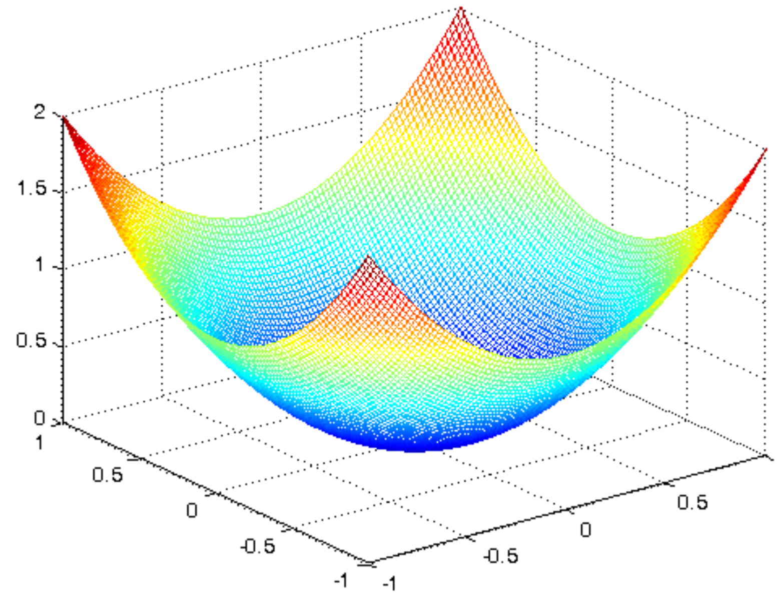
```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t);
```



mesh

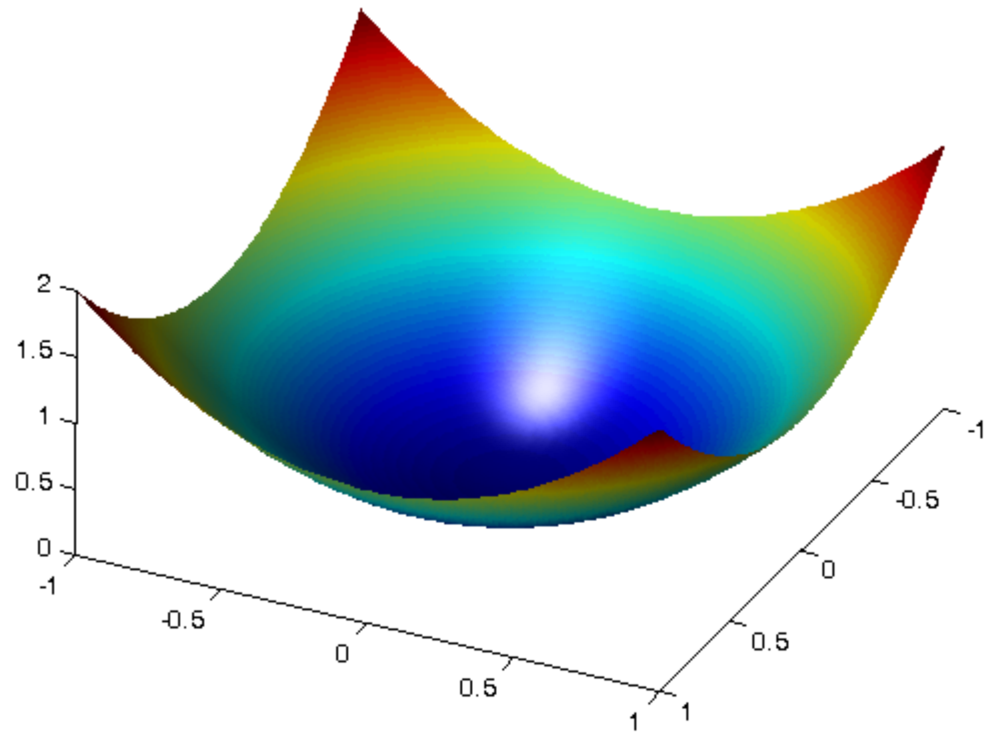
x=linspace(0,1);

```
x=linspace(-1,1);
y=linspace(-1,1)';
X=repmat(x,100,1);
Y=repmat(y,1,100);
mesh(x,y,X.^2+Y.^2)
```



surface

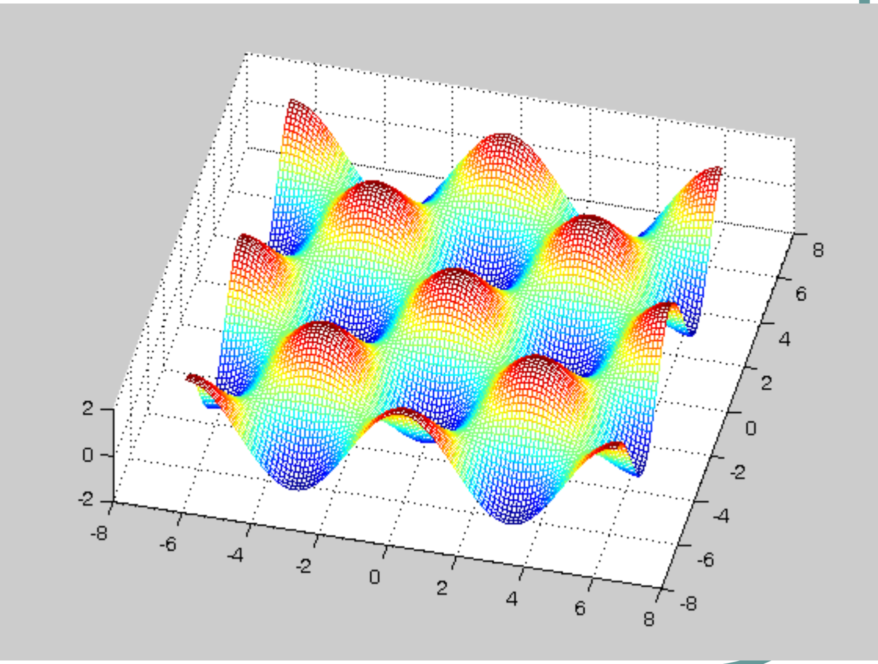
```
x=linspace(-1,1);
y=linspace(-1,1)';
X=repmat(x,100,1);
Y=repmat(y,1,100);
surface(x,y,X.^2+Y.^2)
shading interp
light
lighting phong
```



demo mesh

- demo_mesh.m

Key in a 2D function:cos(x+y)



Taylor series

$$f(x) \approx f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \boxed{?}$$

$$= \sum_{k=0}^{\infty} \frac{f^k(c)}{k!}(x-c)^k$$

$$\textit{if} \;\; f^k(c) \text{ exists for k} = 0, 1, \boxed{?}$$

Taylor's Theorem

$$f(x) \approx f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + ?$$

$$= \sum_{k=0}^{n} \frac{f^{k}(c)}{k!}(x-c)^k + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-c)^{n+1}$$

$if\ f^{n+1}$ exists at interval [a, b]

x, c∈[a , b]

$\xi(x)$∈(x, c)

Taylor's Theorem

- Approximate f(x) by first n+1 terms of Taylor series

$$\sum_{k=0}^{n} \frac{f^{k}(c)}{k!}(x-c)^{k}$$

- Truncating error:

$$\frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-c)^{n+1}$$

Taylor series expansion

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2)$$

$$f'(x) = \frac{df(x)}{dx} = -2xf(x)$$

$$f''(x) = -2f(x) - 2xf'(x) = -2f(x) + 4x^2 f(x)$$

$$= f(x)(4x^2 - 2)$$

$$f'''(x) = 4xf(x) + 8xf(x) - 8x^3 f(x)$$

$$= f(x)(12x - 8x^3)$$

c=1,k=3

$$f(x) = \frac{1}{\sqrt{2\pi}} \exp(-x^2)$$

$$f(x) = \sum_{k=0}^{3} \frac{f^k(c)}{k!} (x-c)^k$$

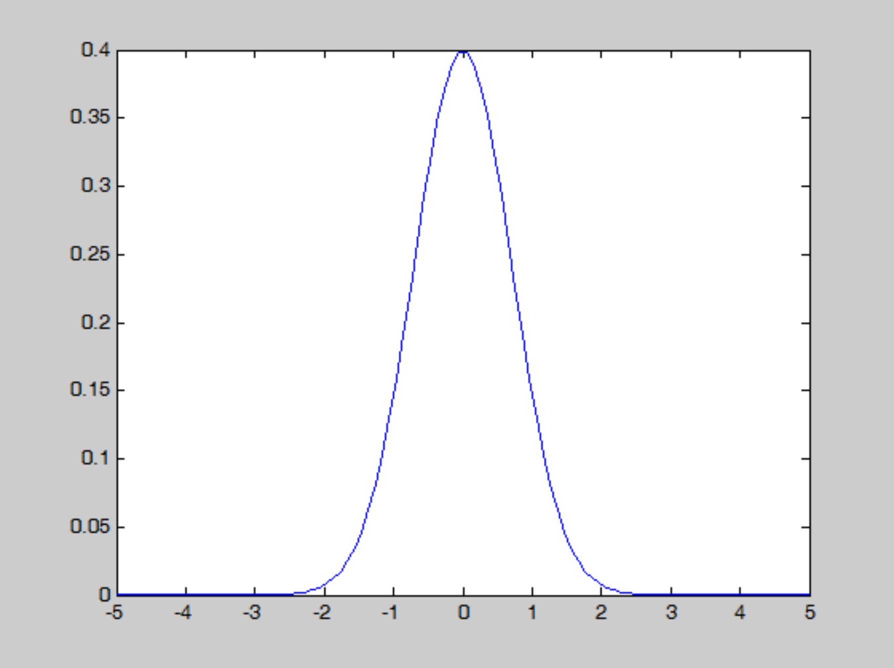
$$= f(c) + f'(c)(x-c) + \frac{f''(c)}{2!}(x-c)^2 + \frac{f'''(c)}{3!}(x-c)^3$$

Normal function

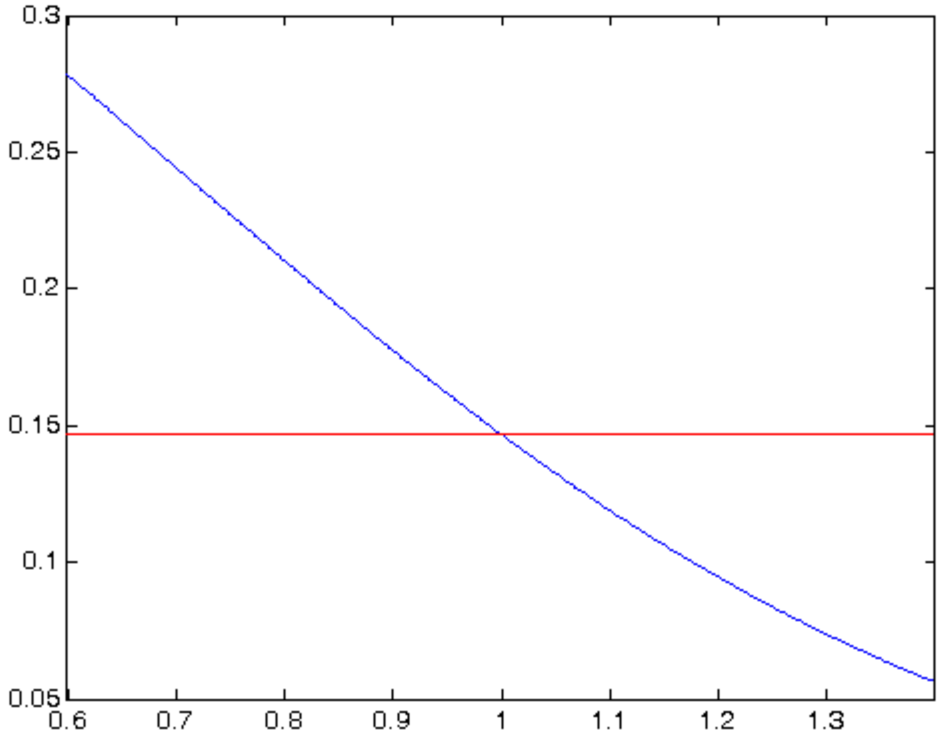
source code

```
function y=f_normal(x)
    y=1/sqrt(2*pi)*exp(-x.^2);
return
```

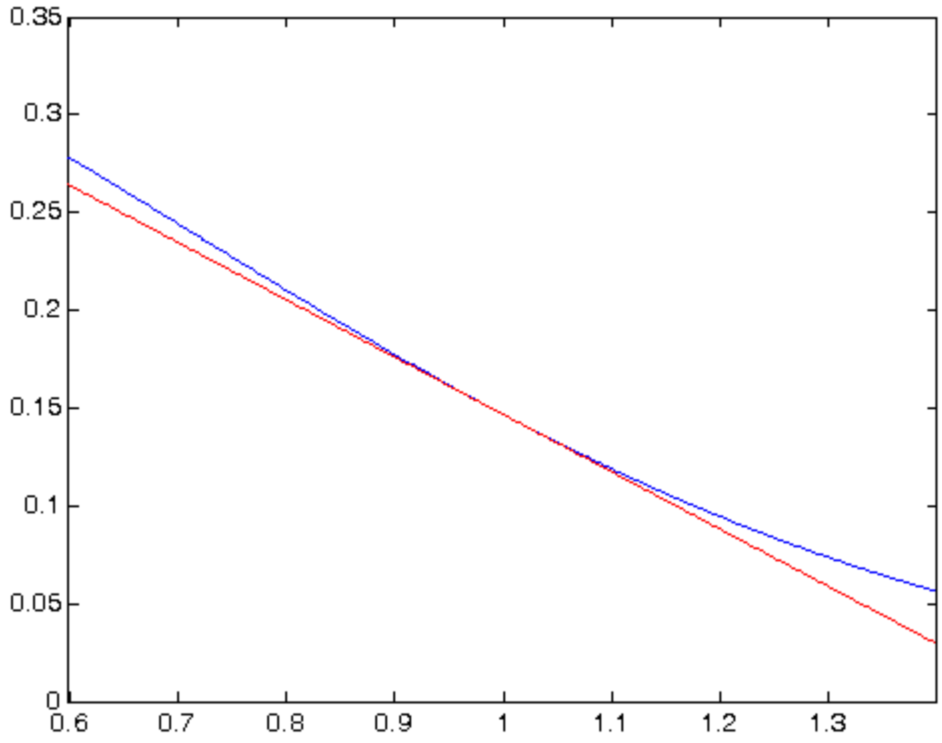
```
x=linspace(-5,5);
>> plot(x,f_normal(x))
```



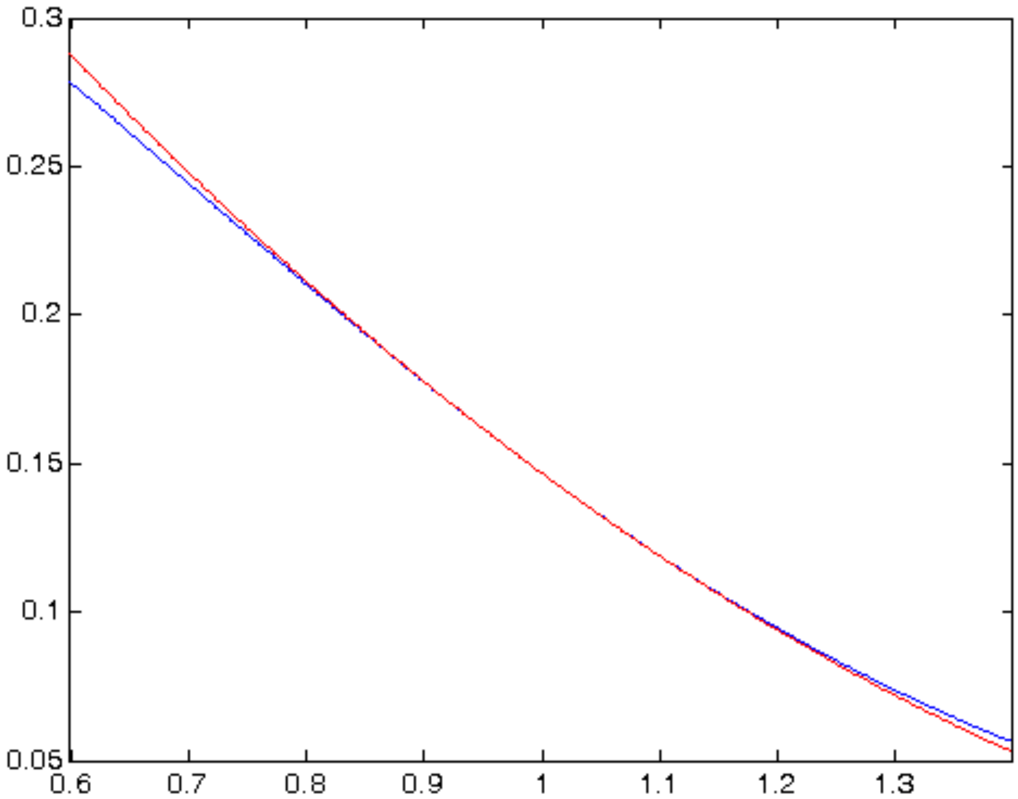
Taylor expansion at c=1 with k=0



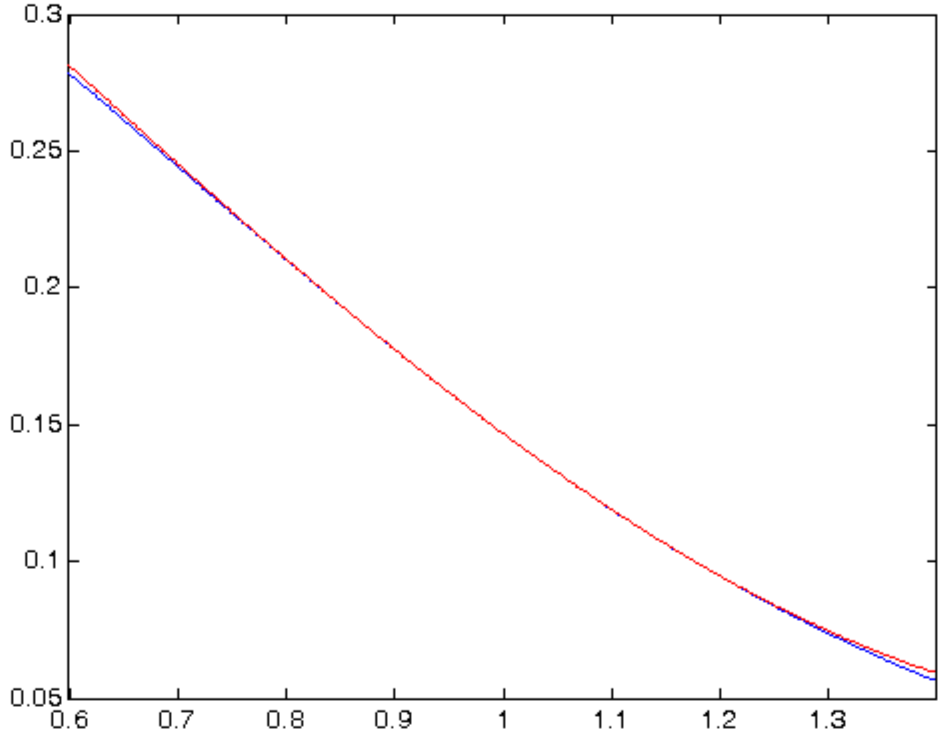
Taylor expansion at c=1 with k ≤1



Taylor expansion at c=1 with k ≤2



Taylor expansion at c=1 with k $\leq$3



Taylor expansion

Source code Taylor_nor

```
% Taylor expansion
% f(x)=1/sqrt(2pi)*exp(-x^2);
x=linspace(0.6,1.4,500);
n=length(x);
c=1;
for i=1:n
    y1(i)=f_nm(c);
    y2(i)=-2*c*f_nm(c)*(x(i)-c);
    y3(i)=(4*c^2-2)*f_nm(c)/2*(x(i)-c)^2;
    y4(i)=(12*c-8*c^3)*f_nm(c)/6*(x(i)-c)^3;
end
plot(x,f_nm(x));hold on;plot(x,y1,'r'); title('one term');
figure; plot(x,f_nm(x));hold on;plot(x,y1+y2,'r');title('two terms');
figure; plot(x,f_nm(x));hold on;plot(x,y1+y2+y3,'r');title('three terms');
figure; plot(x,f_nm(x));hold on;plot(x,y1+y2+y3+y4,'r');title('four terms');
```

```
>> x=sym('x');
>> diff(x.^2)

ans =

2*x
```

```
x=sym('x')
ss='x.^2';
inst=['diff(' ss ')']
eval(inst)

ans =

2*x
```

First derivative

```
% input a string to specify a function
% find its derivative
ss='tanh(x)'
fx=inline(ss);
ss=['diff(' ss ')'];
ss1=eval(ss);
fx1=inline(ss1)
```

2nd derivative

```
x= sym('x')
f=inline(tanh(x));
s1= diff(tanh(x))
f1 = inline(s1)
s2=diff(s1)
f2 = inline(s2)
s3=diff(s2)
f3 = inline(s3)
```

Example

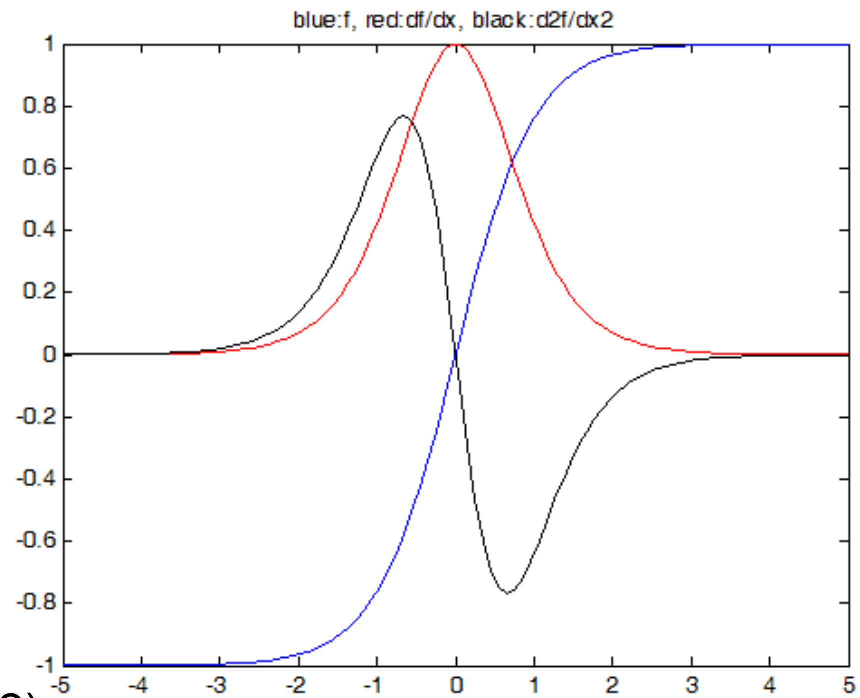
>> f_diff3
function of x:tanh(x)

fx1 =

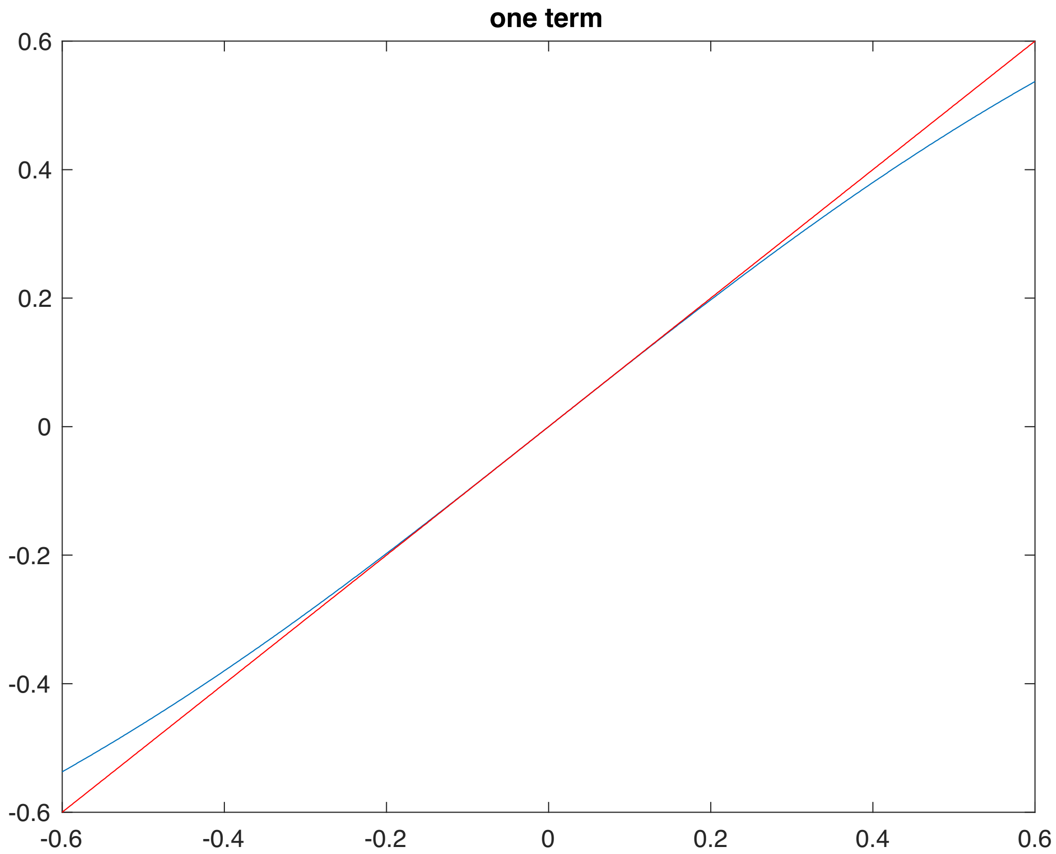
Inline function:
fx1(x) = 1-tanh(x).^2

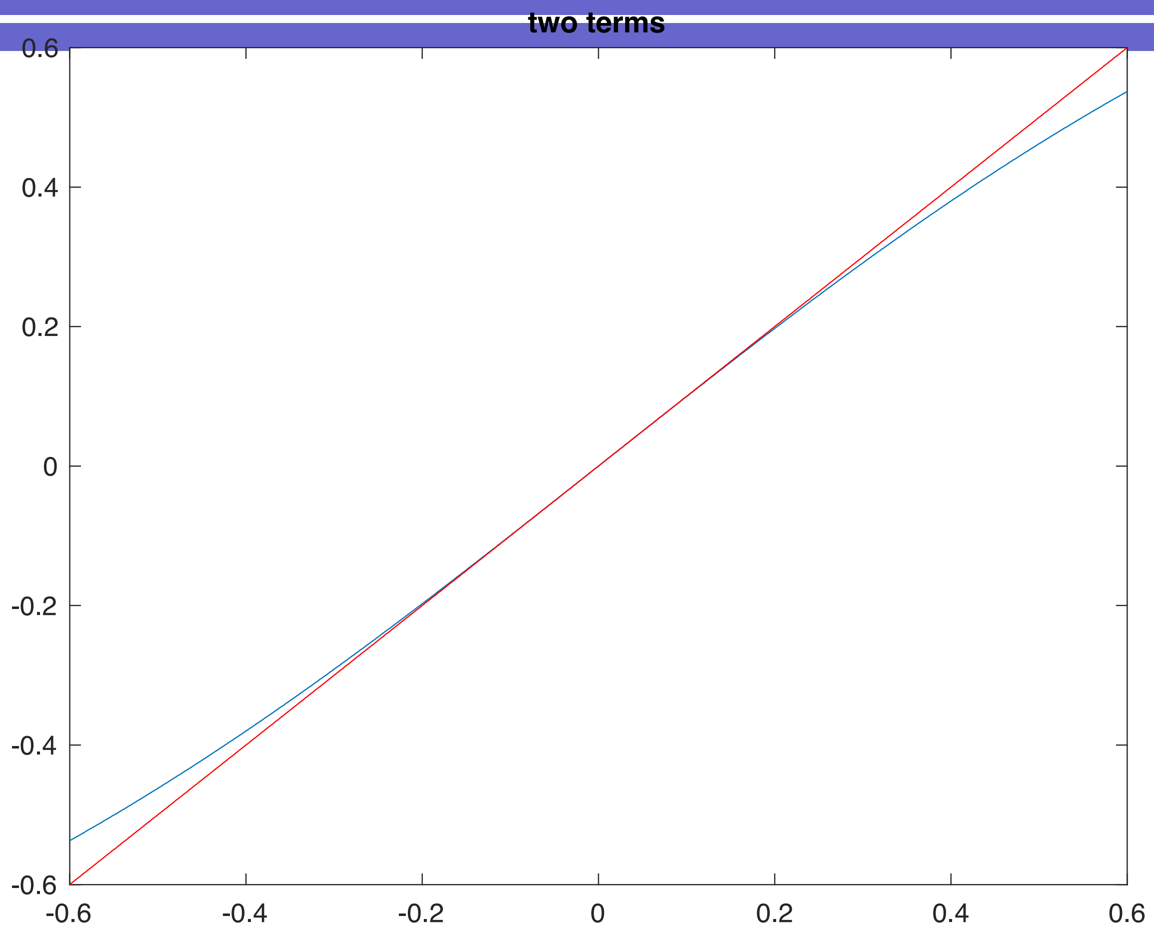
fx2 =

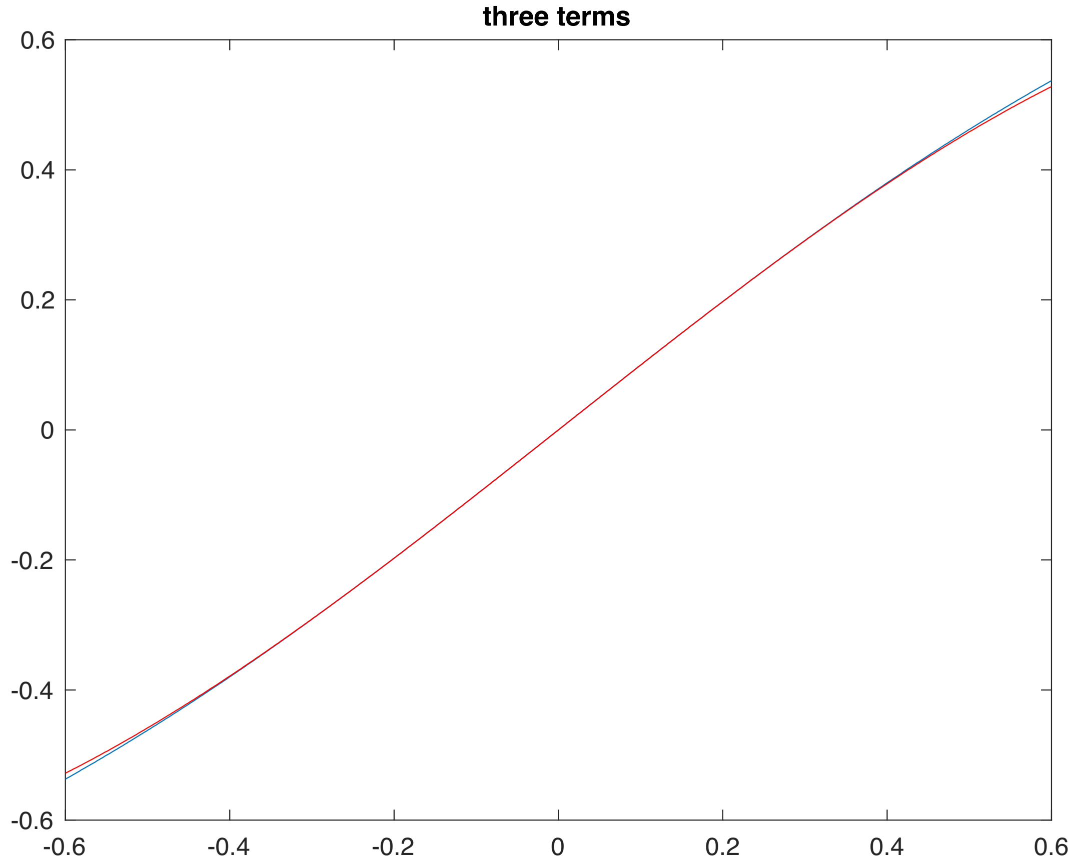
Inline function:
fx2(x) = -2.*tanh(x).*(1-tanh(x).^2)

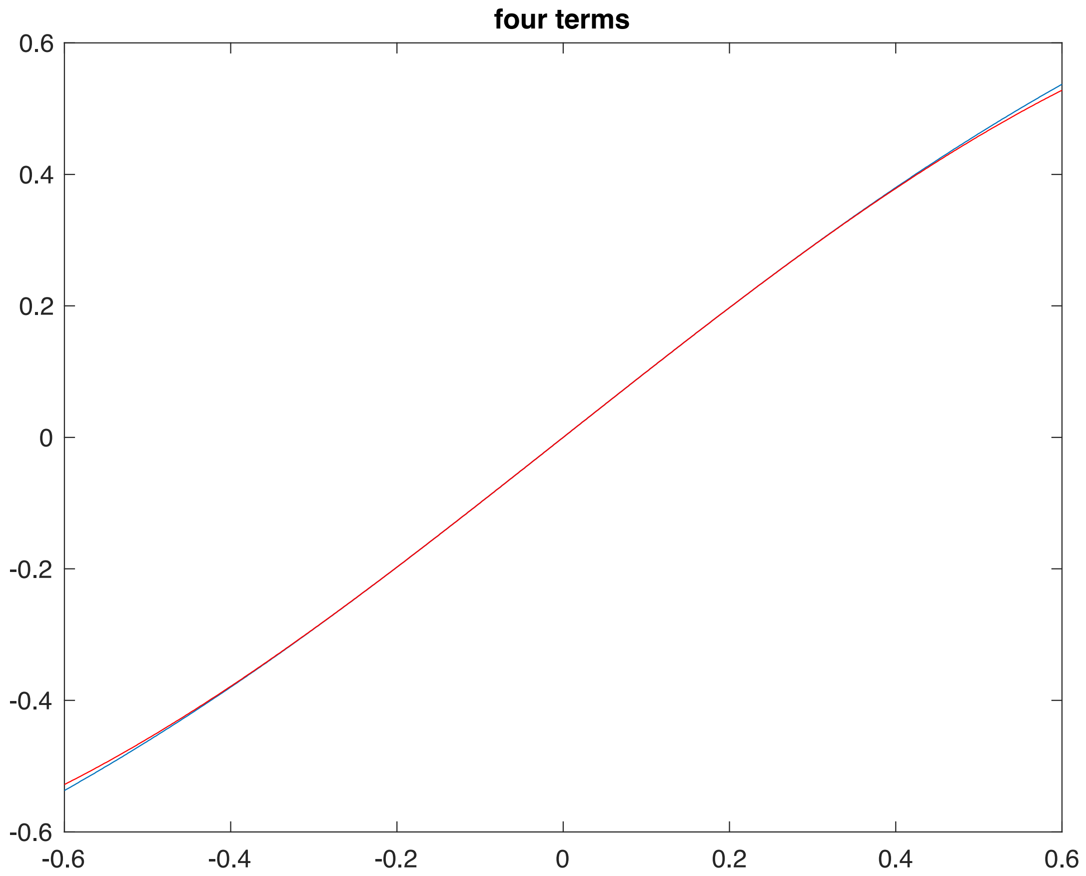


```
x=linspace(-0.6,0.6,500);
n=length(x);
c=0;
for i=1:n
    y(i)=f (c);
    y1(i)=f1(c)*(x(i)-c);
    y2(i)=f2(c)/2*(x(i)-c)^2;
    y3(i)=f3(c)/6*(x(i)-c)^3;
end
plot(x,f (x));hold on;plot(x,y1,'r'); title('one
term');
figure; plot(x,f (x));hold
on;plot(x,y1+y2,'r');title('two terms');
figure; plot(x,f (x));hold
on;plot(x,y1+y2+y3,'r');title('three terms');
```


one term
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.6
-0.4
-0.2
0
0.2
0.4
0.6


two terms
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

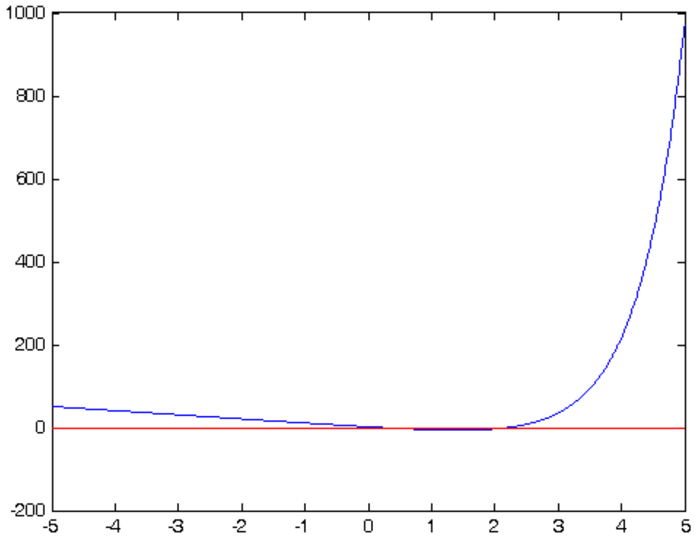

three terms
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.6
-0.4
-0.2
0
0.2
0.4
0.6


four terms
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.6
-0.4
-0.2
0
0.2
0.4
0.6

Problem: **Root finding**

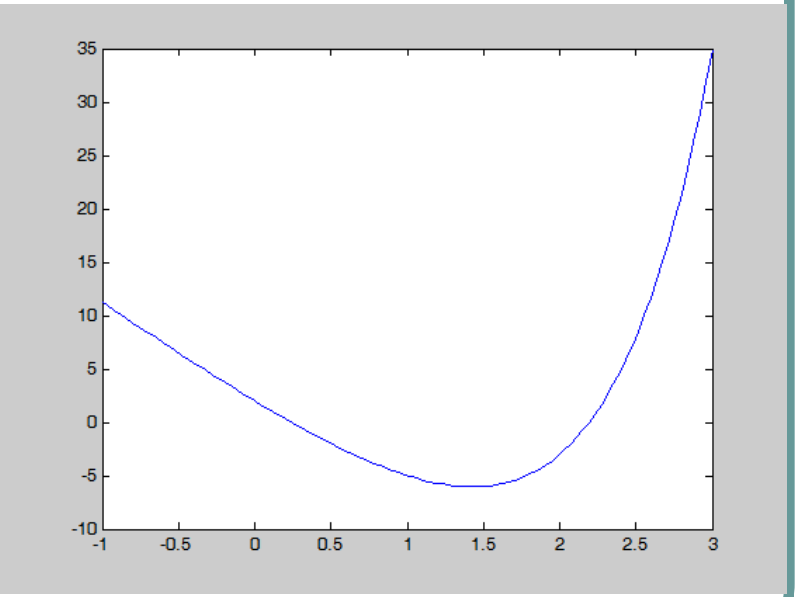
- Find x such $f(x)=0$ for a given f
- f denotes an arbitrary function

$$f(x) = 2^{x^2} - 10x + 1$$



>> x=linspace(-1,3);

>> plot(x,2.^x.^2-10*x+1)



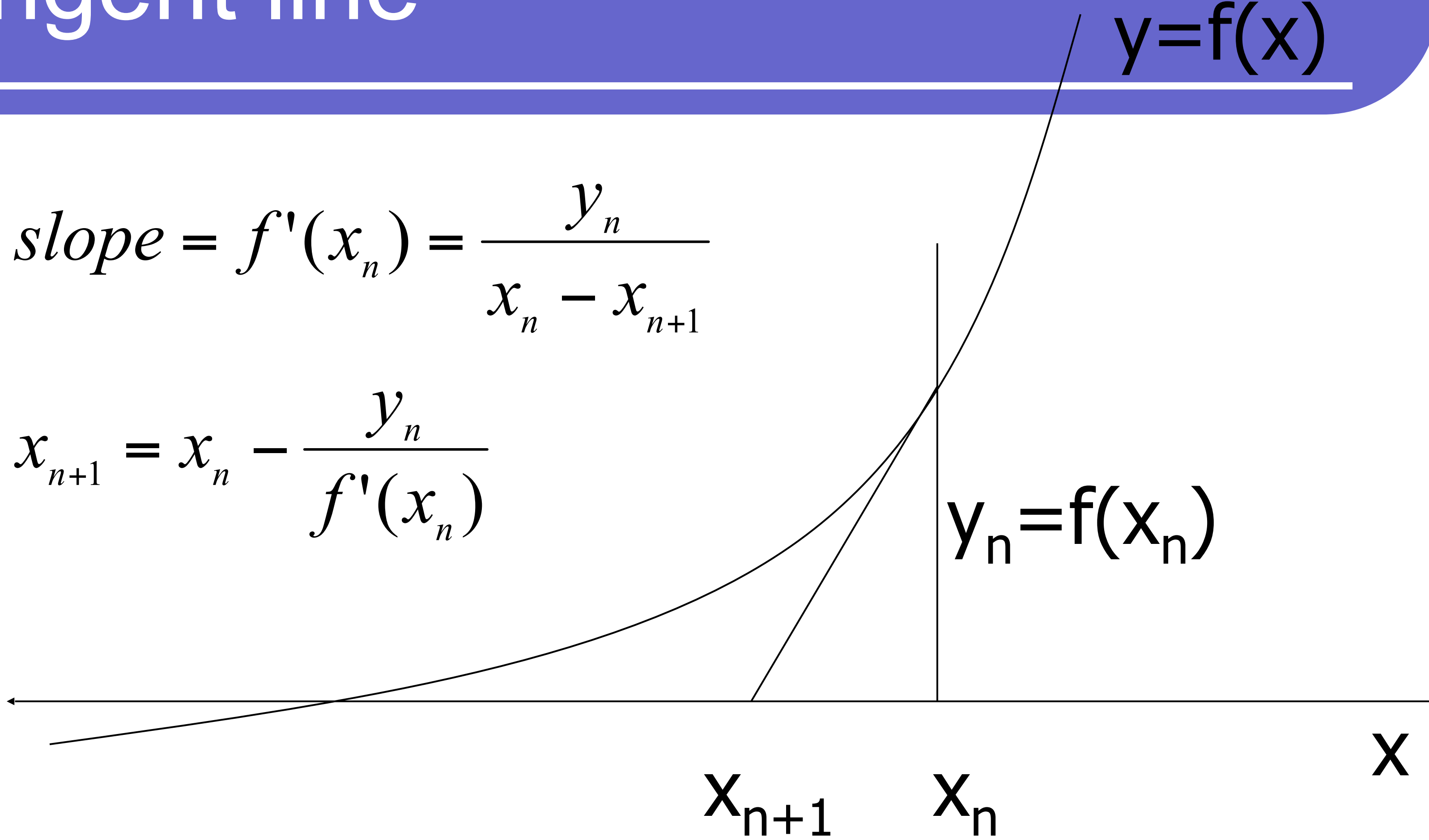
Root finding

- Input: a string that specifies a function
- Plot the given function
- Find at least a root

Tangent line

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

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



Iterative approach

- Start at a random guess
- Refine current guess by
  - Find the tangent line that passes $(x_n, f(x_n))$
  - Set x_n to intersection of the tangent line to horizontal axis

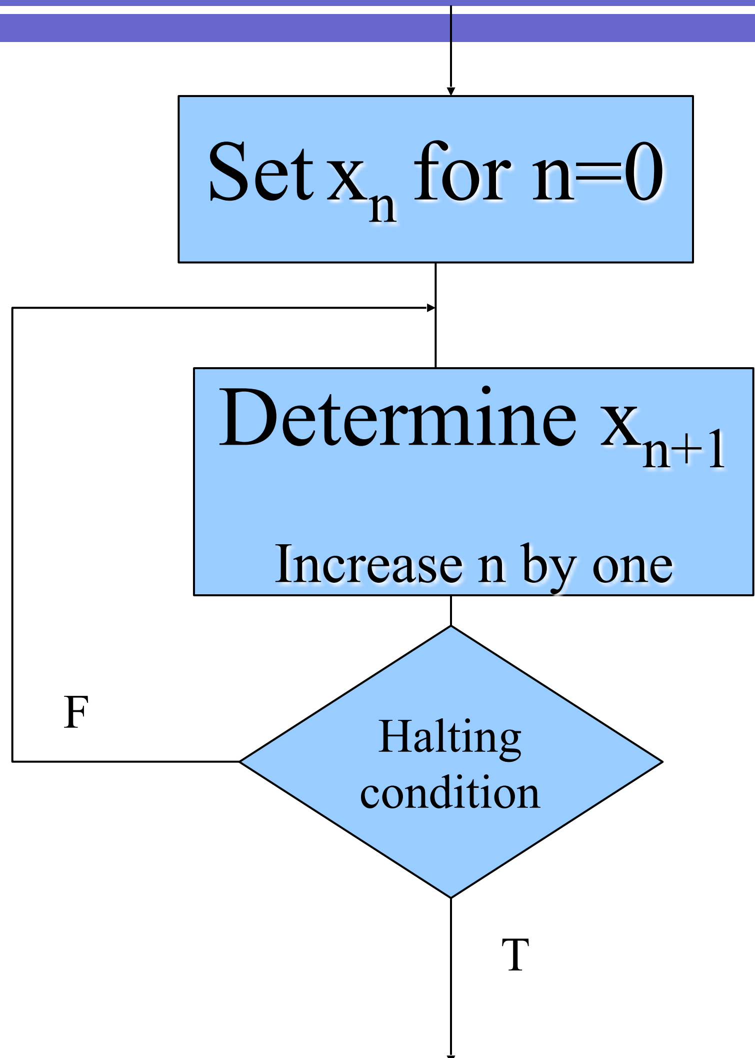
Updating rule

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

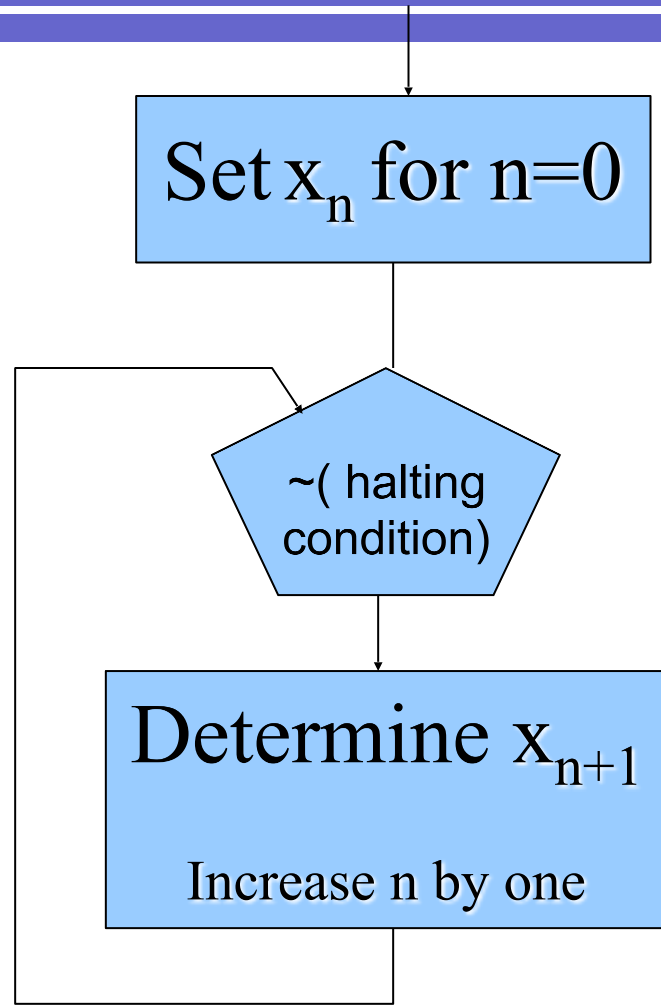
Newton method

- Iterative approach
  - Random guess
  - Refine current guess according to an updating rule
  - If halting condition holds, go to step 2 otherwise
    exit

Iterative approach

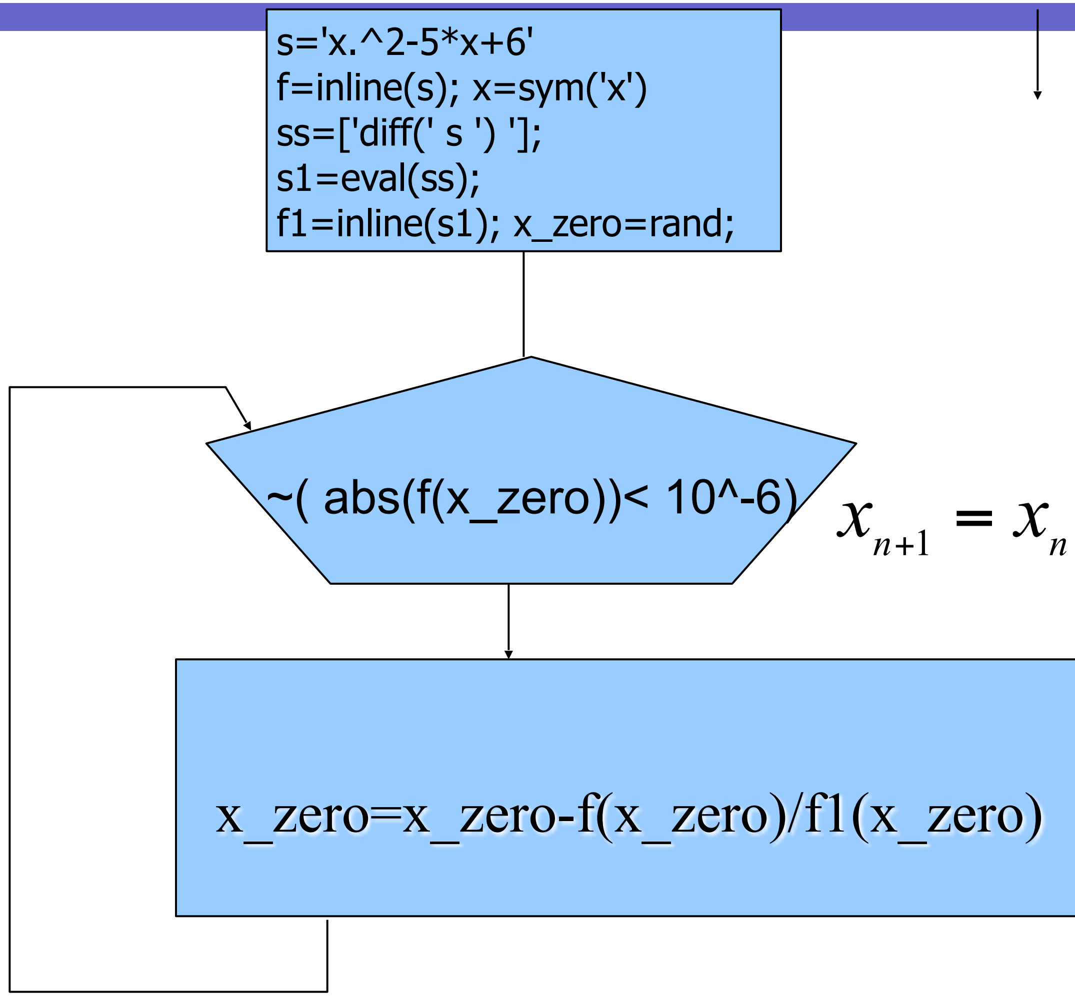


Iterative approach



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

Iterative approach



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

Halting condition

- Let x_n denote the guess at iteration n
- The absolute value of $f(x_n)$ is expected close to zero for some n
- Halting Condition

  abs($f(x_n)$) < epslon

  - epslon denotes a predefined positive small number

Initialization

- Random initialization
- Input from users

Demo_newton

source code demo_newton

```
fstr=input('input a function:','s');
x_ini=input('guess its zero:');
x_zero=newton(fstr,x_ini);
```

newton.m

Main Program

```
fstr=input('input a function:','s');
x_ini=input('guess its zero:');
range=3;
x_zero=newton(fstr,x_ini);
fx=inline(fstr);x=linspace(-range,range);plot(x,fx(x));hold on;
plot(x_zero,fx(x_zero),'ro');
plot([-range range],[0 0],'r');
```

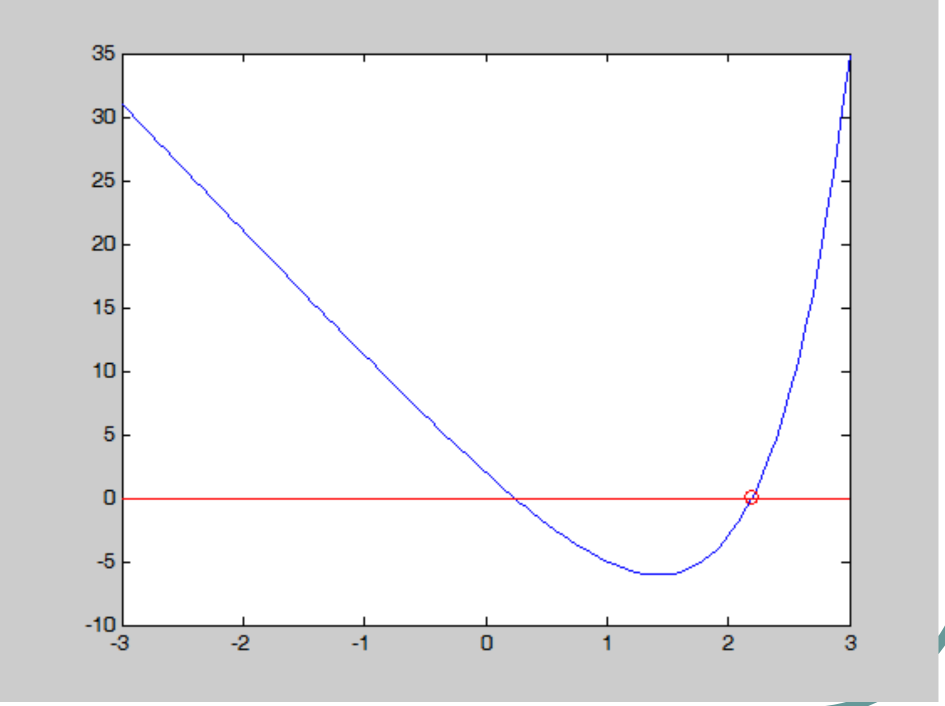
Newton method

```
ss=['diff(' fstr ')'];
x=sym('x');
fstr1=eval(ss);
fx1=inline(fstr1);
xzero=x_ini;
it=0;
while abs(fx(xzero)) > ep
    it=it+1;
    if abs(fx1(xzero)) < ep
        fprintf('zero derivative\n');
        return
    end
    xzero=xzero-fx(xzero)/fx1(xzero);
    fprintf(' iter=%d x=%f fx=%f\n',it,xzero,fx(xzero));
end
```

```
>> demo_newton
input a function:cos(x)
guess its zero:2
 iter=1 x=1.542342 fx=0.028450
 iter=2 x=1.570804 fx=-0.000008
 iter=3 x=1.570796 fx=0.000000
>>
```

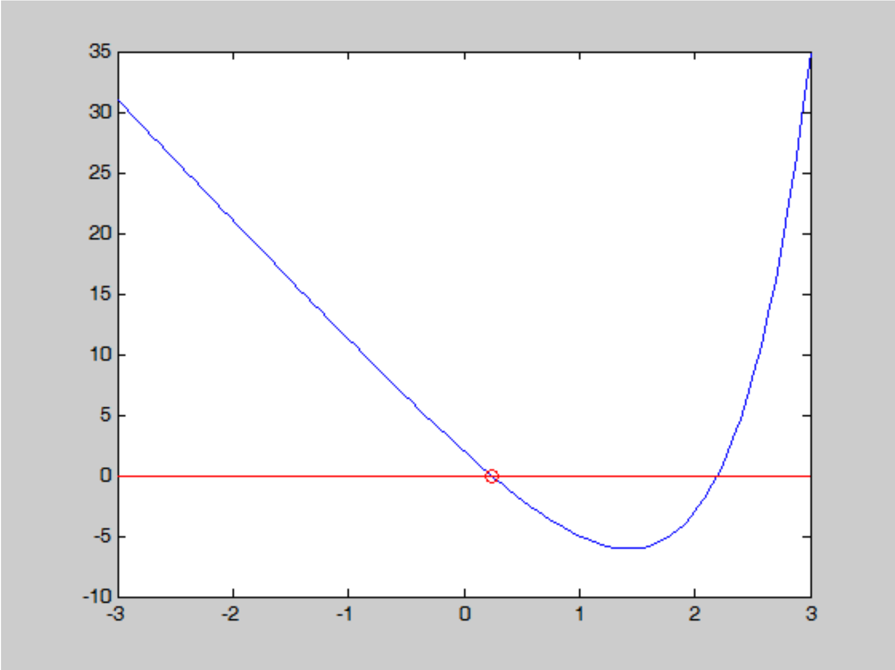
Example

demo_newton
input a function:2.^x.^2-10*x+1
guess its zero:1.5



Example

>> demo_newton
input a function:2.^x.^2-10*x+1
guess its zero:0.5



Unconstrained Optimization

- Given a differentiable function, y=f(x), unconstrained optimization aims to find the minimum of f(x)
- Let x denote a minimum and $g(x) = \frac{df(x)}{dx}$

Then

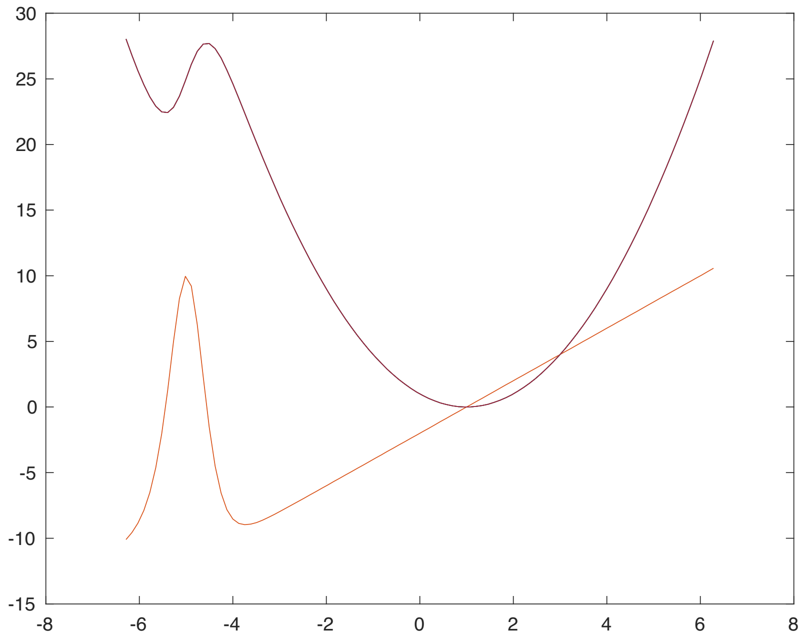
$$g(x) = 0$$

Optimization by Newton method

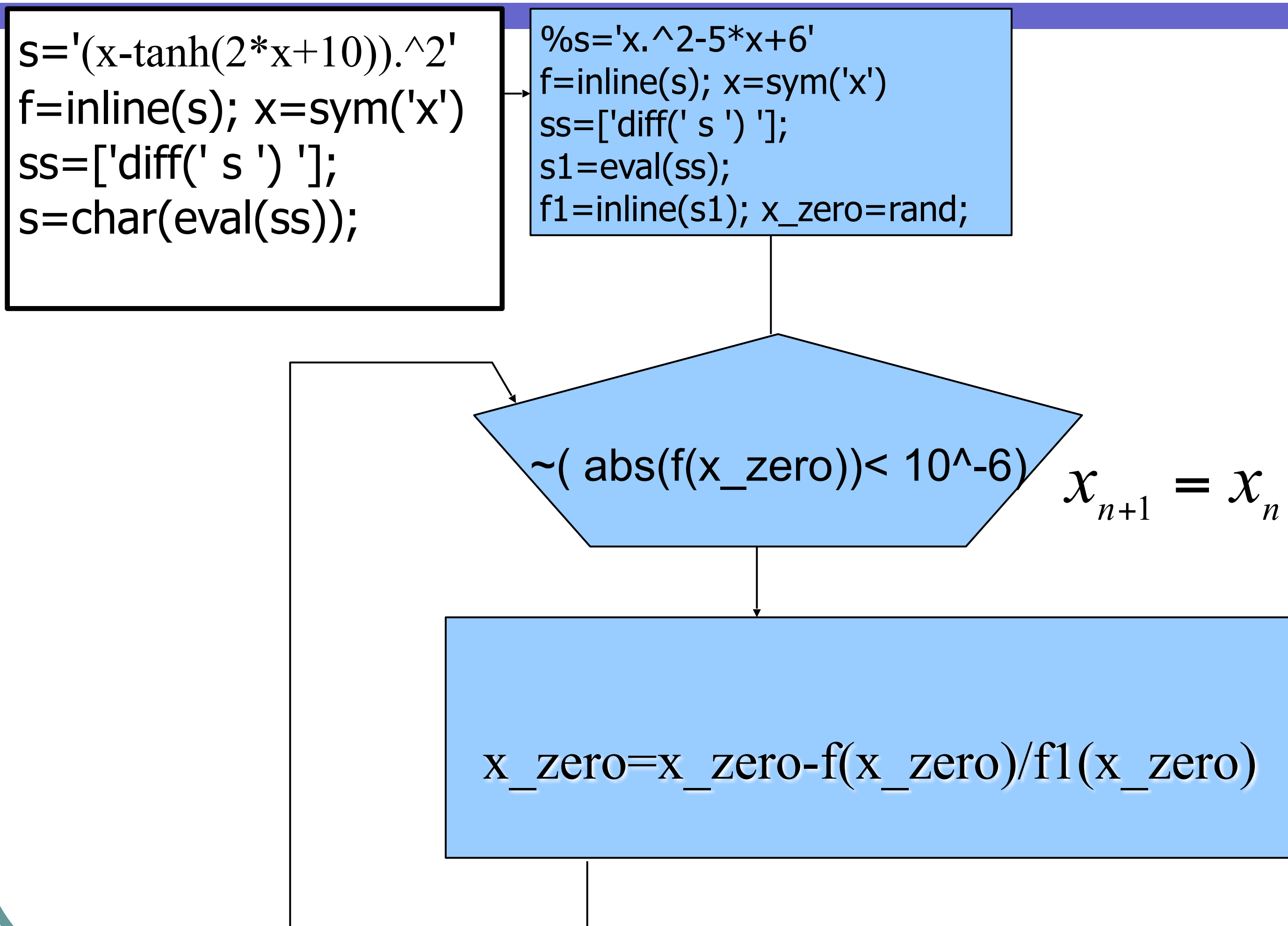
- Use symbolic differentiation to find the first derivative of a given function
- Apply the Newton method to find zeros of the first derivative

First derivative and second derivative

```
s='(x-tanh(2*x+10)).^2'
f=inline(s); x=sym('x')
ss=['diff(' s ') '];
S=eval(ss);
f1=inline(s);
z=linspace(-2*pi,2*pi);
plot(z,f(z));hold on;
plot(z,f1(z));
```

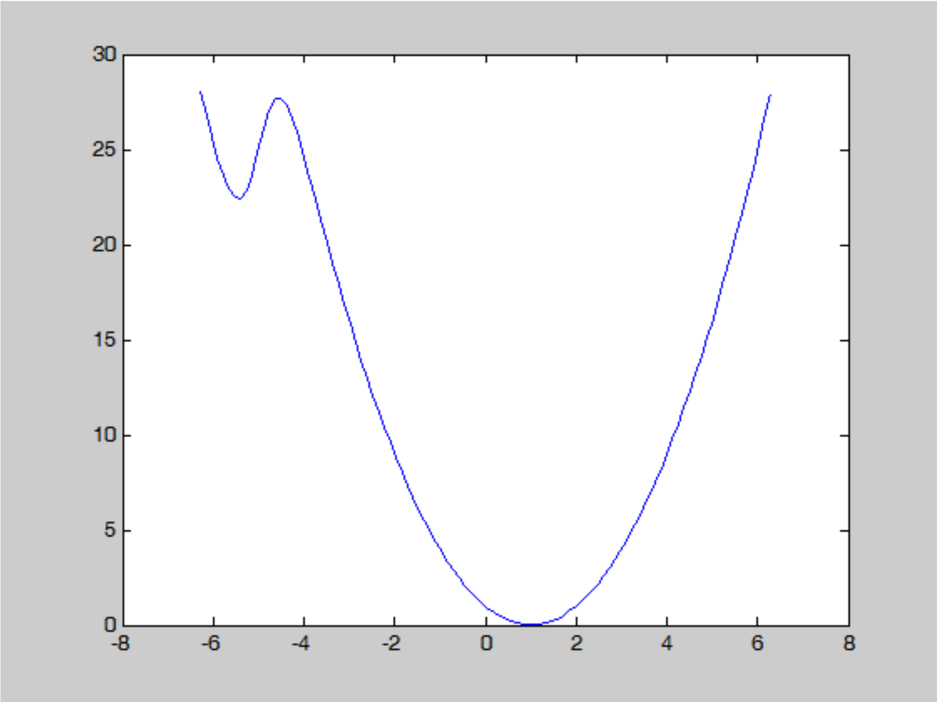


Iterative approach



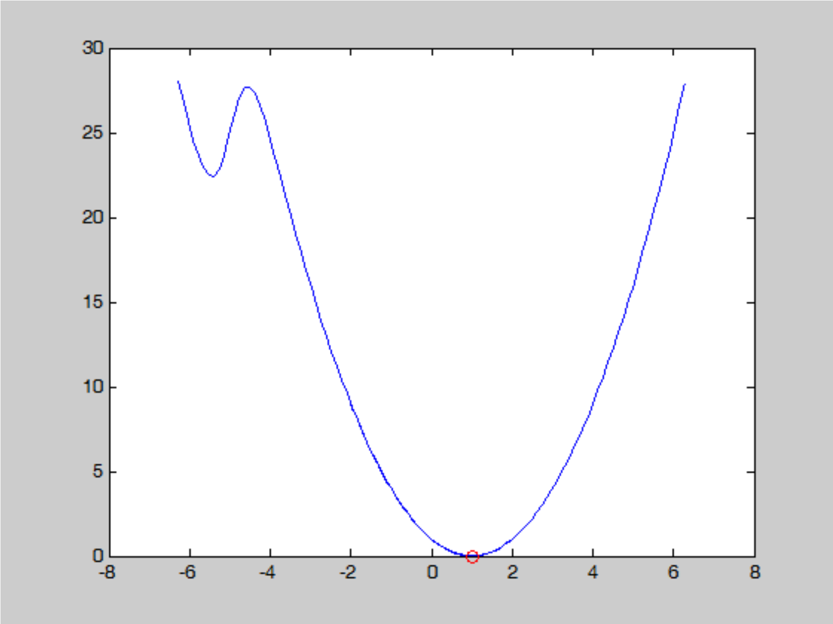
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

```
>> x=linspace(-2*pi,2*pi);
>> plot(x,(x-tanh(2*x+10)).^2)
```



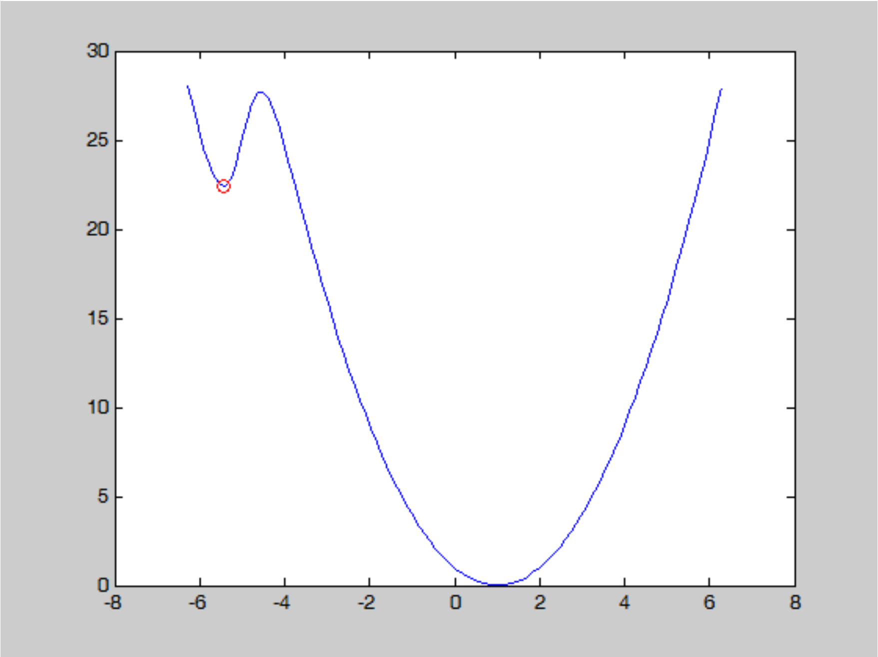
Local minimum

```
>> demo_min
input a function:(x-tanh(2*x+10)).^2
guess its minimum:4
```

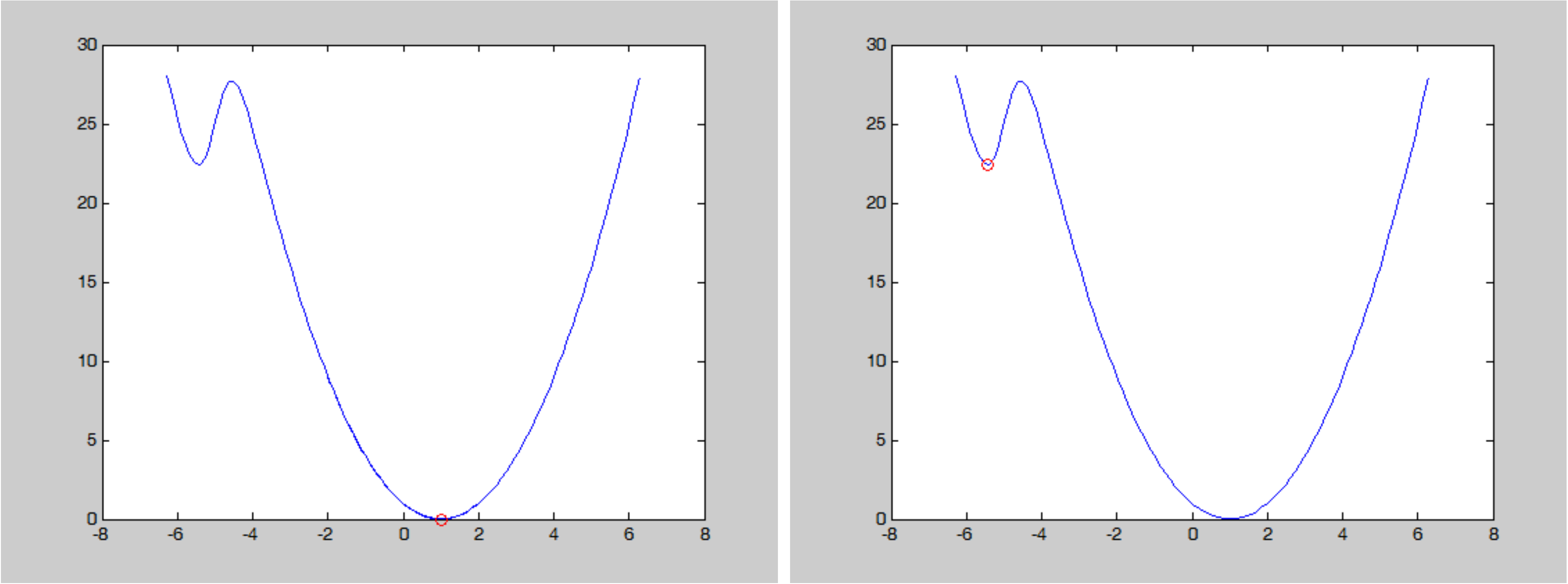


Local Minimum

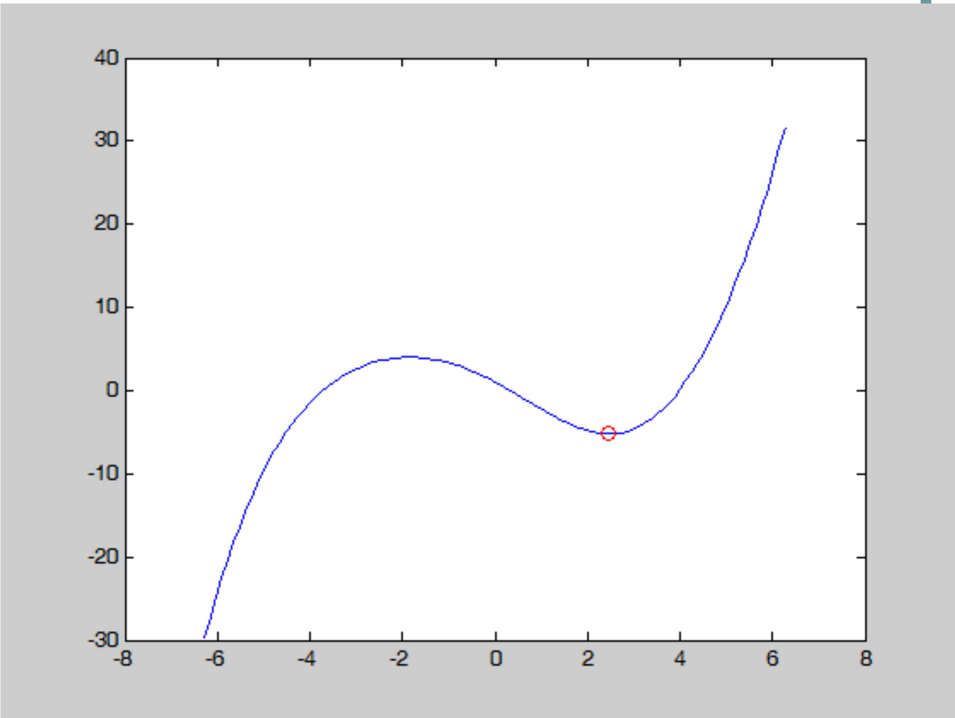
```
>> demo_min
input a function:(x-tanh(2*x+10)).^2
guess its minimum:-4
```



Global minimum



```
>> demo_min
input a function:0.2*x.^3-3*x+cos(x)
guess its minimum:2
```



Exercise

- ex3.pdf