Version 1.01

MATLAB Essentials Of a Specialist

Solving algebraic equation

 # Roots of polynomial

 # x + 4 = 0

```
>> equation = [1 4];
>> roots(equation)
ans = -4
```

 # x^2 - 4 = 0

```
>> equation = [1 0 -4]; % Ax^2 + Bx + C = 0 [A B C]
>> roots(equation)
ans =
    2.0000
   -2.0000
```

 # x^2 -3x + 9 = 0

```
>> roots([1 -3 9]);
>> ans
ans =
   1.5000 + 2.5981i
   1.5000 - 2.5981i
```

Solving System of Linear Equation

 # Ax = B

 # x + 2y = 4   3x + 4y = 5

```
>> A = [1 2; 3 4];
>> B = [4; 5];
>> x = A\B
x =
   -3.0000
    3.5000
```

 # alternate

```
>> x = inv(A)*B % or (A^-1)*B
```

 # linsolve

```
>> linsolve(A,B)
```

Solving symbolically

 # x + 3 = 2

```
>> syms x
>> solve('x+3=2',x)
ans =
-1
```

 # x^2 + 2x -3 = 12

```
>> syms x
>> solve('x^2+2*x-3=12',x)
ans =
 -5
  3
```

 # sinx = 3

```
>> syms x y
>> solve('sin(x)=3',x)
ans =
      asin(3)
 pi - asin(3)
>> double(ans)
ans =
   1.5708 - 1.7627i
   1.5708 + 1.7627i
```

 # x+2*y=-3*z % Solve for y

```
>> syms x y z
>> solve('x+2*y=-3*z',y)
ans =
- x/2 - (3*z)/2
>> pretty(ans)
  x   3 z
- - - ---
  2    2
```

 # x+2*y=-3*z % Solve for z

```
>> syms x y z
>> solve('x+2*y = -3*z',z)
ans =
- x/3 - (2*y)/3
>> solve('sin(x) + exp(z) = 3*y',y)
ans =
exp(z)/3 + sin(x)/3
>> solve('sin(x) + exp(z) = 3*y',x)
ans =
     -asin(exp(z) - 3*y)
 pi + asin(exp(z) - 3*y)
```

 # Linear equations symbolically

```
>> syms x
>> solve('x+3=2',x)
ans =
-1
```

Linear equations Solve using symbolic toolbox

```
>> syms x y z
>> [x y] = solve ('3*x - y = 2','x+y=1')
x = 3/4
y = 1/4
>> solution = solve ('3*x - y = 2','x+y=1')
solution =
    x: [1x1 sym]
    y: [1x1 sym]
>> solution.x
ans =
3/4
>> solution.y
ans =
1/4
```

 # Solve with constant in coefficients in linear equation

```
>> [x y] = solve('2*x-3*c*y=5','c*x+2*y=7')
x =
(21*c + 10)/(3*c^2 + 4)
y =
-(5*c - 14)/(3*c^2 + 4)
```

Creating Mathematical Functions

 # f(x) = x^2

```
>> f = inline('x^2','x')
f =
     Inline function:
     f(x) = x^2
>> f(2)
ans =
     4
```

 # area of a triangle

```
>> areaofTriangle = inline('.5*b*h','b','h');
>> areaofTriangle

areaofTriangle =
     Inline function:
     areaofTriangle(b,h) = .5*b*h

>> areaofTriangle(5,10)
ans =
    25
```

Algorithms

 # Bisec Rule

```
%% function
% f = 2 - 2 + log(x)

%% Initial guesses
xl = 1;
xu = 4;

fl = 2-xl + log(xl);
fu = 2-xu + log(xu);

%% check signs

flag = fl*fu;

if flag>0 %opposite sign
    error('Initial guesses should have different signs.');
end

%% iterative solution using bisecRule

err = abs(xl-xu);
xNew = (xl + xu) / 2;
fNew = 2 - xNew + log(xNew);
%%check signs of fNew*fl
if fl*fNew > 0
    xl = xNew;
    fl = fNew;
else
    xu = xNew;
    fu = fNew;
end
```

 # Newton Raphson

```
% f = 2 - x + log(x)
% f1 = - 1 + 1/x
% algorithm
% Newguess = Currentguess - (f(Currentguess)/f1(Currentguess)
%% implementing
NewGuess = 8.0;
f=1;
while(f~=0)
    Guess = NewGuess;
    f = 2 - Guess + log(Guess);
    f1 = - 1 + (1/Guess);
    NewGuess = Guess - (f/f1);
end
```

 # Gauss-Siedal Method

```
x = [1,2,2,1;2,2,4,2;1,3,2,5;2,6,5,8]; a = [1;0;2;4];
A = [x,a]; solution = x\a;
%% use A(1,1) as pivot element
for i = 2:1:length(a)
    alpha = A(i,1) / A(1,1);
    A(i,:) = A(i,:) - alpha * A(1,:);
end
%% use A(2,2) as pivot element
for i = 3:1:length(a)
    alpha = A(i,2) / A(2,2);
    A(i,:) = A(i,:) - alpha * A(2,:);
end
%%
for j = 1:1:(length(a)-1)
    for i = j+1:1:length(a)
        alpha = A(i,j) / A(j,j);
        A(i,:) = A(i,:) - alpha * A(j,:);
    end
end
```

```
# Maclaurin
n = 5;
aAll = [0.1,0.05,0.02,0.01];

vec = [1:n];
err=[];

for i=1:length(aAll)
    terms = aAll(i).^vec ./ cumprod(vec);

    expVal = 1 + cumsum(terms);

    trueVal = exp(aAll(i));

    err = [err;abs(trueVal - expVal)];
end

plot(err);
xlabel('step size');
ylabel('error');
hold on;
```

Derivatives

 # Derivative

```
>> syms x   % taking the variable function must be defined in syms
>> f = inline('x^3+x^2+3','x')
f =
     Inline function:
     f(x) = x^3+x^2+3

>> diff(f(x),x)
ans =
3*x^2 + 2*x
```

 # Derivative at a point

```
>> syms x
>> f = inline('x^2-3','x')
f =
     Inline function:
     f(x) = x^2-3
>> f1 = inline(diff(f(x),x),'x')
f1 =
     Inline function:
     f1(x) = x.*2.0
>> f1(3)
ans =
     6
```

 # Partial derivative

```
>> syms x
>> syms y
>> f = inline('sin(x)+y^3','x','y')
f =
     Inline function:
     f(x,y) = sin(x)+y^3
>> diff( f(x,y),x ) % taking der with respect to x
ans =
cos(x)
```

```
>> diff ( f(x,y),y ) % taking der with respect to y
ans =
3*y^2
```

 # counting how many times a function can be differentiated

```
syms x
f = inline('x^4 - 3*x^3 + 2*x^2 - x + 16','x');
der = f(x);
count = 0;
while(der~=0)
f = inline( diff(f(x),x), 'x' );
der = f(x);
count = count + 1;
end
>> count
5
```

 # Derivative of two polynomials product d/dx(P(x)*P(y))

```
>> p1 = [3 6 9];
>> p2 = [1 2 0];
>> polyder(p1,p2);
>> ans
ans =
    12    36    42    18
```

 # Numerical Differentiation

```
% tan inverse(a)
a = 1;
trueVal = 1/(1+a^2);
h = 0.001;
approxVal = (atan(a+h) - atan(a)) / h;
err = abs(approxVal - trueVal);
```

Limits

 # limit(function,x,approachesvalue)

```
>> syms x

>> f = inline('sin(x)/x','x')

f =

     Inline function:

     f(x) = sin(x)/x

>> limit(f(x),x,0)

ans =

1


%approaches infinity = limit((f(x),x,Inf)
```

Integration

 # Indefinite Integral

#

```
>> syms x
>> f = inline ('x^2+2*x','x')
f =
     Inline function:
     f(x) = x^2+2*x
>> int(f(x),x);
>> ans
ans =
(x^2*(x + 3))/3
```

 # Definite Integral

 # int(f(x),lowerlimit,upperlimit)

```
>> syms x
>> f = inline('x^2+2*x','x')
f =
     Inline function:
     f(x) = x^2+2*x
>> int(f(x),1,4)
ans =
36
```

```
# Numerical Integration
x=0:0.1:1; % x:lowerlimit:interval:upperlimit;
y=x.^2;
plot(x,y);

%% Trapzoid Method
avg_y = y(1:length(x)-1) + diff(y)/2;
A = sum(diff(x).*avg_y);

%% Simpson Method
A = quad('x.^2',0,1); %quad('func',lowerlimit,upperlimit);

%% Lobatto Method
A = quadl('x.^2',0,1);
```

Reading from a file

 # Reading numbers

```
>> nums = textread(numbers.txt')

nums = 123456789
```

 # Reading strings

```
>> textread('motto.txt','%s')

ans =

    'Visca'

    'El'

    'Barca'
```

 # Reading characters

```
>> cha = textread('motto.txt','%c');

>> cha

cha =

V

i

s

c

a

E

l

B

a

r

c

a
```

 # Reading images

```
>> ima = imread('img.jpg');


excel = xlsread('diode.xls', 'c4
```

 # Reading excel file

```
>> excel = xlsread('diode.xls','c4:c19');
```

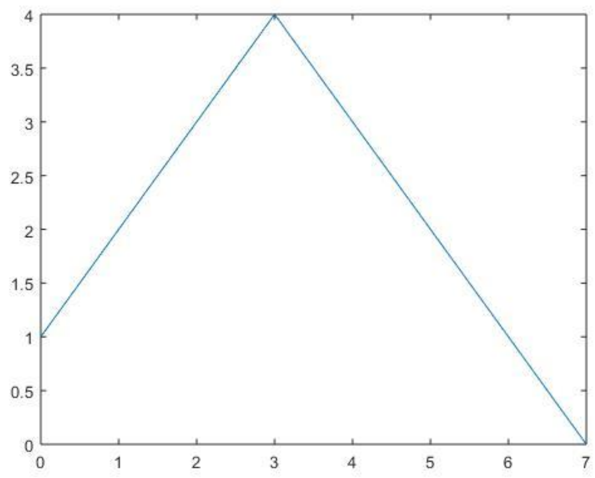
 # Reading audio file

```
>> audiofile = wavread('audio.wav');
```

Plotting

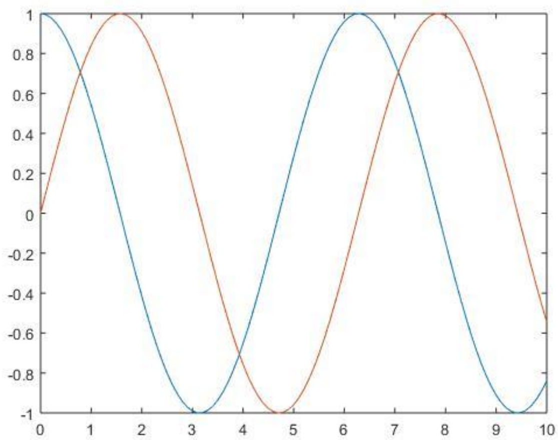
 # Basic Plot

```
>> x = [0 1 2 3 4 5 6 7];
>> y = [1 2 3 4 3 2 1 0];
>> plot(x,y)
>> title('My Favourite Plot')
>> xlabel('x-axis')
>> ylabel('y-axis')
>> plot(x,y)
```



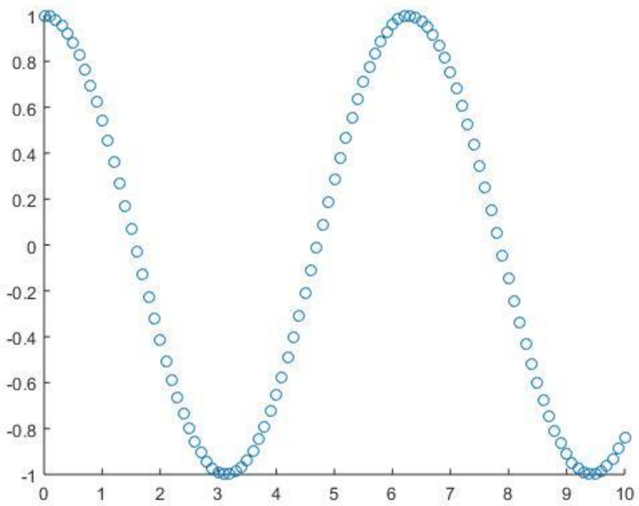
2 plots together

```
>> x = [0:0.1:10];
>> y = cos(x);
>> z = sin(x);
>> plot(x,y,x,z)
```



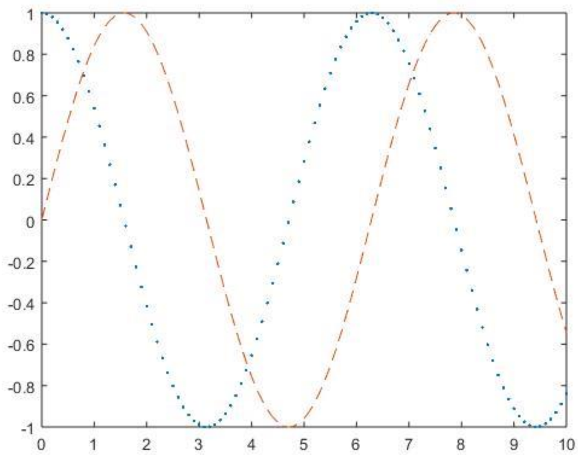
 # scatterplot

```
>> x = [0:0.1:10];
>> y = cos(x);
>> scatter(x,y)
```



 # Changing plot appearance

```
>> plot(x,y,'.',x,z,'--')
>> % legend('sinx','cosx') can be used to mention legends
```



Plot styles modification

```
plot(x,y,'ColorMarkerLine');
```

Specifier	LineStyle
'-'	Solid line (default)
'--'	Dashed line
':'	Dotted line
'-.'	Dash-dot line

Marker Specifiers

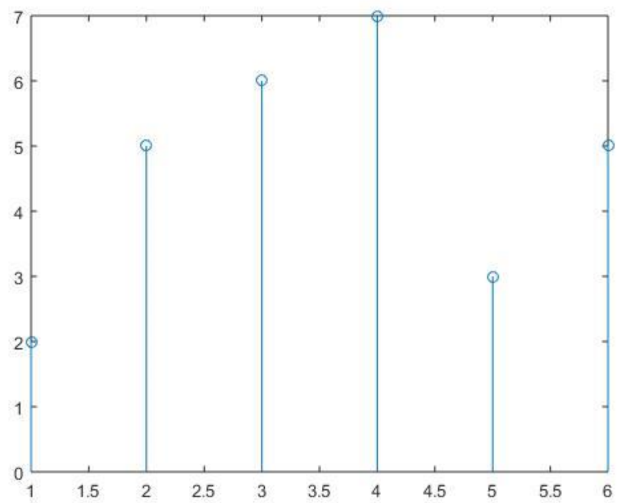
Specifier	Marker Type
'+'	Plus sign
'o'	Circle
'*'	Asterisk
'.'	Point
'x'	Cross
'square' or 's'	Square
'diamond' or 'd'	Diamond
'^'	Upward-pointing triangle
'v'	Downward-pointing triangle
'>'	Right-pointing triangle
'<'	Left-pointing triangle
'pentagram' or 'p'	Five-pointed star (pentagram)
'hexagram' or 'h'	Six-pointed star (hexagram)

Color Specifiers

Specifier	Color
r	Red
g	Green
b	Blue
c	Cyan
m	Magenta
y	Yellow
k	Black
w	White

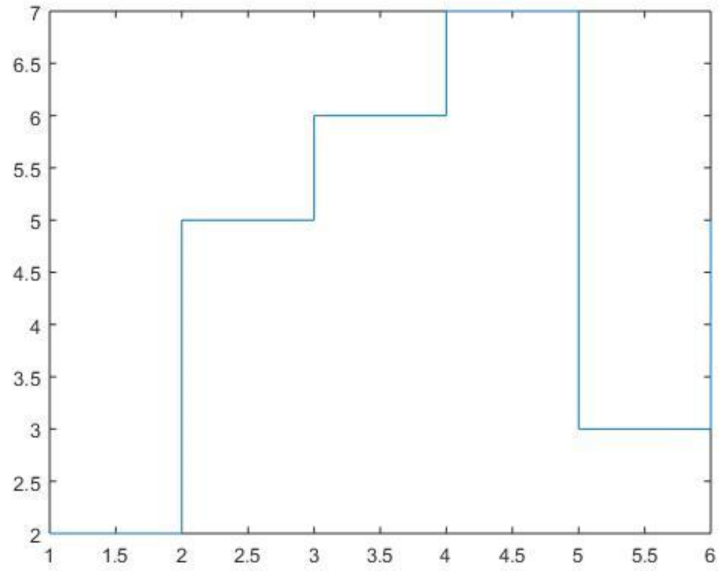
Stem plot

```
>> x = [1:6];
>> y = [2,5,6,7,3,5];
>> stem(x,y)
```



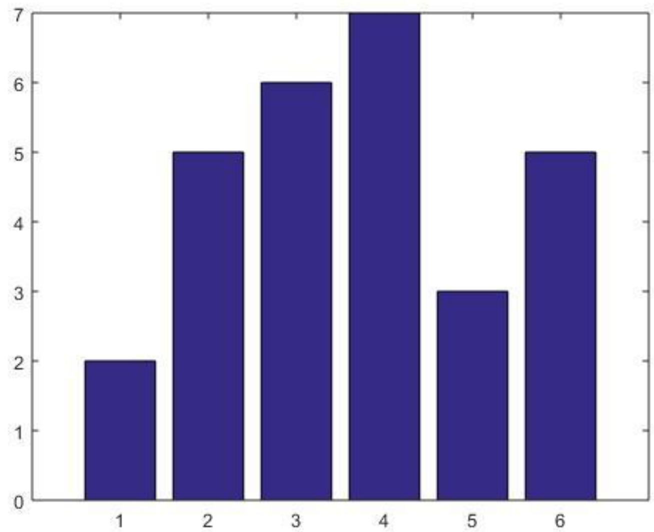
 # Stair Plot

```
>> x = [1:6];
>> y = [2,5,6,7,3,5];
>> stairs(x,y)
```



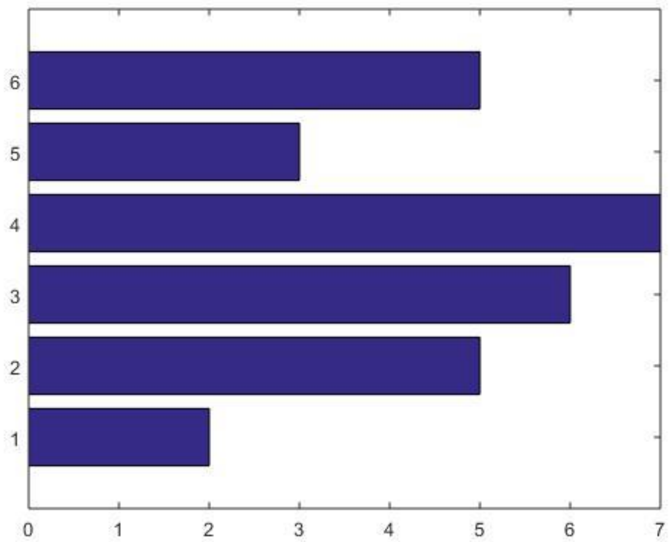
 # Bar plot

```
>> x = [1:6];
>> y = [2,5,6,7,3,5];
>> bar(x,y)
```



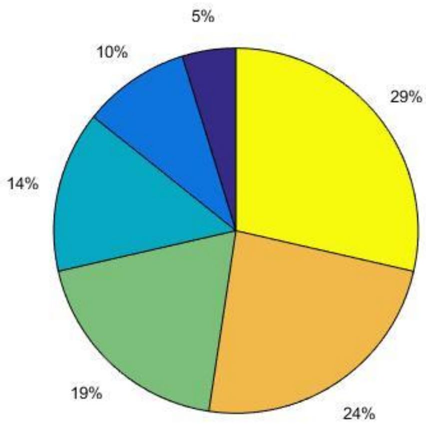
 # Horizontal Bar plot

```
>> x = [1:6];
>> y = [2,5,6,7,3,5];
>> barh(x,y)
```



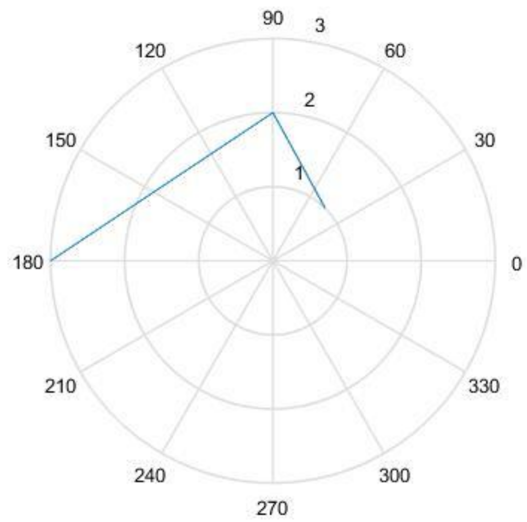
 # Pie plot

```
>> x = [1:6];
>> pie(x)
```



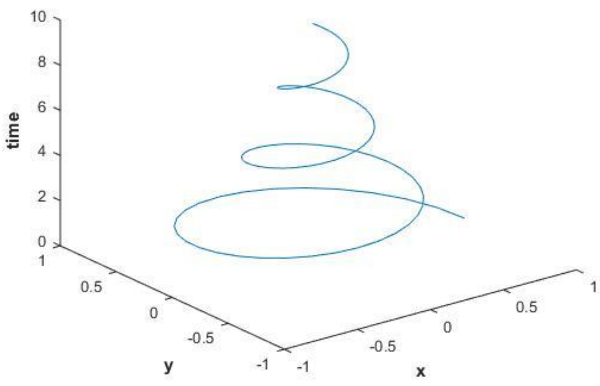
 # Polar Plot

```
>> theta = [pi/4 pi/2 pi];
>> r = [1 2 3];
>> polar(theta,r)
```



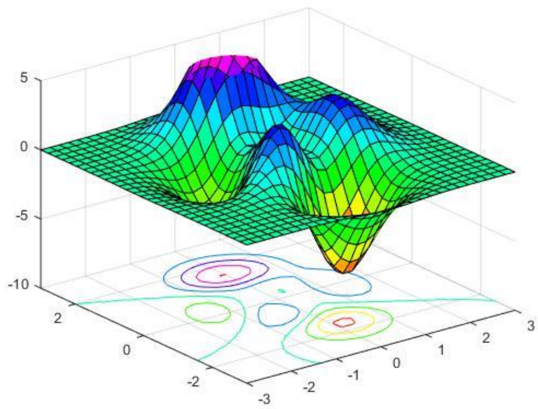
 # 3D Plot

```
>> t = [0:0.1:10];
>> x = exp(-0.2*t).*cos(2*t);
>> y = exp(-0.2*t).*sin(2*t);
>> plot3(x,y,t);
xlabel('\bf x');
ylabel('\bf y');
zlabel('\bf time');
```



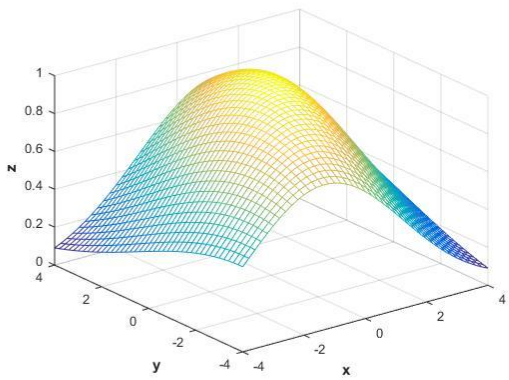
Surf

```
[x,y,z]=peaks(30);
surfc(x,y,z);
colormap hsv
axis([-3 3 -3 3 -10 5])
```



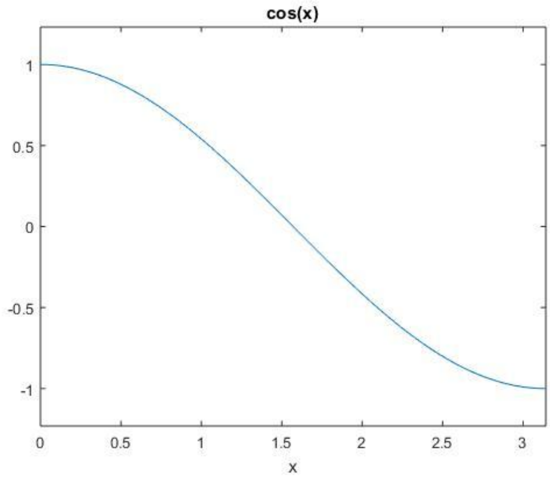
mesh

```
>> [x,y] = meshgrid(-4:0.2:4);
z = exp(-.05*(x.^2 + 0.5*(x-y).^2));
mesh (x,y,z);
xlabel('\bf x');
ylabel('\bf y');
zlabel('\bf time');
>> [x,y] = meshgrid(-4:0.2:4);
z = exp(-.05*(x.^2 + 0.5*(x-y).^2));
mesh (x,y,z);
xlabel('\bf x');
ylabel('\bf y');
zlabel('\bf z');
```



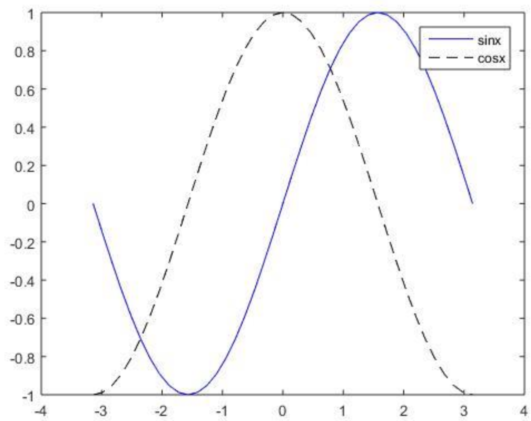
ezplot

```
>> ezplot('cos(x)',[0 pi]);
```



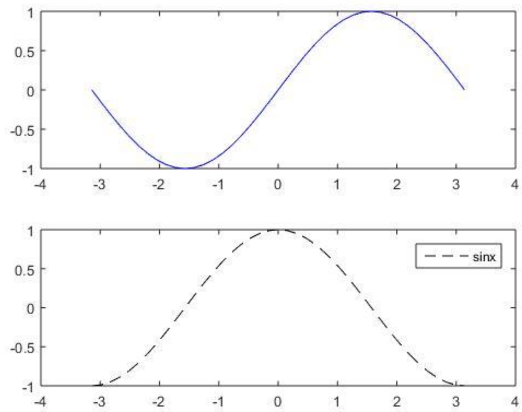
Multiple plots in same axis (again)

```
>> x = -pi:pi/20:pi;
>> y1 = sin(x);
>> y2 = cos(x);
>> plot (x,y1,'b-');
>> hold on
>> plot (x,y2,'k--');
>> legend('sinx','cosx');
```



 # Many figures in same window : Subplots

```
x = -pi:pi/20:pi;
y1 = sin(x);
y2 = cos(x);
subplot(2,1,1)
plot (x,y1,'b-');
subplot(2,1,2);
plot (x,y2,'k--');
legend('sinx','cosx');
```

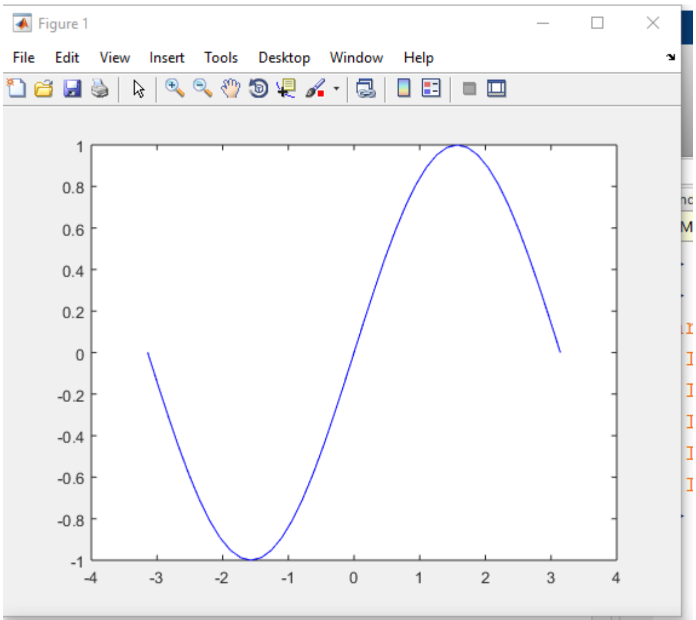


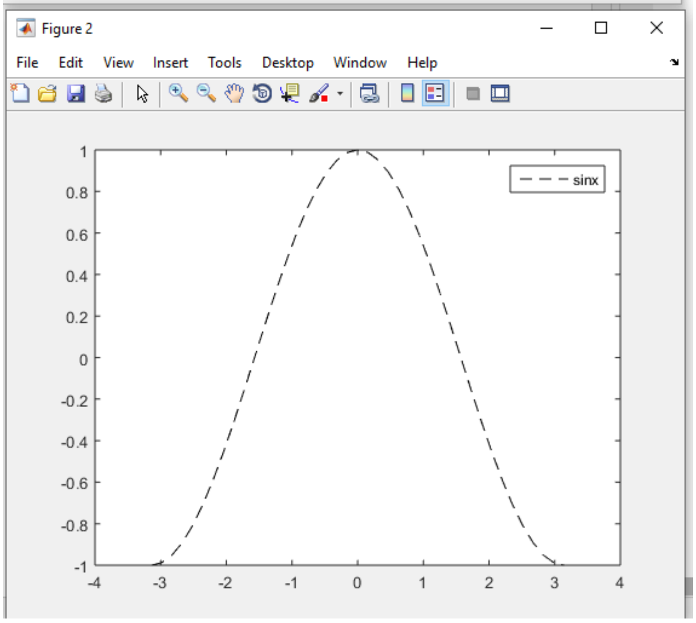
 # Enhanced Control of Plotted Lines

```
# LineWidth
# MarkerEdgeColor
# MarkerFaceColor
# MarkerSize
>> plot (x,y,'PropertyName',value,'PropertyName',value,……)
```

 # Creating Multiple Figures in different windows

```
figure(1)
x = -pi:pi/20:pi;
y1 = sin(x);
y2 = cos(x);
plot (x,y1,'b-');
figure(2)
plot (x,y2,'k--');
legend('sinx','cosx');
```





Polynomials

 # Evaluate a polynomial

```
# polyval(P,k) evauates value of x for which P(value) = k

P(x) = x^3 + x – 1

P = [1 0 1 -1]
```

 # Multiplication of Polynomials

```
P(x) = 3x + 2

Q(x) = 2x + 4

>> P = [3 2]

>> Q = [ 2 4]

>> prod = conv(P,Q)
```

 # Polynomial solution

```
>> y1 = sym('x-0.5*y+1.5*z-5');

>> y2 = sym('6*x+4*y-2*z-10');

>> y3 = sym('-x-y+z+1');

>> [x,y,z] = solve(y1,y2,y3)

x = 5.0

y = -6.0

z = -2.0
```

Transfer Function

```
#
>> num = [1 2];
>> den = [1 4 5];
>> H = tf(num,den)

H =

      s + 2
  -------------
  s^2 + 4 s + 5

Continuous-time transfer function.

>> pzmap(H)
>> grid on
>> [z,p] = tf2zp(num,den)

z =

    -2


p =

  -2.0000 + 1.0000i
  -2.0000 - 1.0000i
```

Signal Analysis

Amplitude Modulation

Mathematical Expression for AM

S_{AM} (t) = A_c * [1 + K_a * m(t)] * cos(2*(pie)*f_c*t)

K_a = Amplitude sensitivity of AM

m(t) = message signal > m(t) = Am sin(2*pie*f_m*t)

c(t) = carrier signal > c(t) = Ac cos(2*pie*f_c*t)

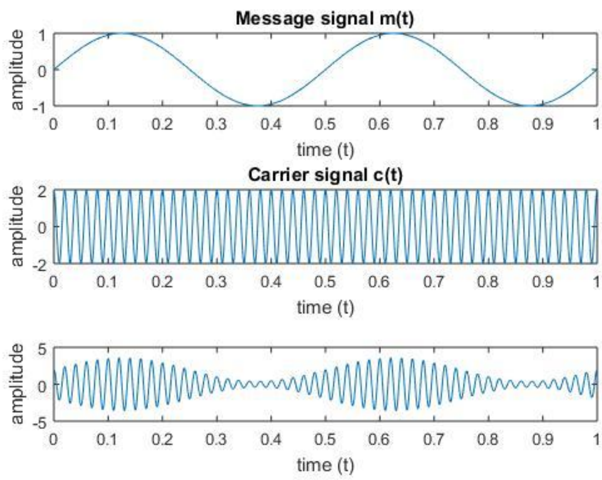
f_c = carrier frequency

f_m = message frequency

So, S_{AM} (t) = A_c * [1 + K_a * m(t)] * c(t)

Code

```
>> ka = 0.8;
>> Am = 1;
>> fm = 2;
>> t= [0:0.001:1];
>> mt = Am * sin(2*pi*fm*t);
>> Ac = 2;
>> fc = 50;
>> ct = Ac*cos(2*pi*fc*t);
>> St = (1+ka.*mt).*ct;
>> subplot(3,1,1)
>> plot(t,mt)
>> title('Message signal m(t)')
>> xlabel(' time (t) ')
>> ylabel(' amplitude ' )
>> subplot(3,1,2)
>> plot(t,ct)
>> title('Carrier signal c(t)')
>> xlabel(' time (t) ')
>> ylabel(' amplitude ' )
>> subplot(3,1,3)
>> plot(t,St)
>> xlabel(' time (t) ')
>> ylabel(' amplitude ' )
```



Frequency Modulation

Mathematical Expression for FM

S_{FM} (t) = A_c * cos [2*pie*f_c*t + beta*sin(2*pie*f_m*t)]

K_f = Frequency sensitivity of FM

m(t) = message signal > m(t) = Am cos(2*pie*f_m*t)

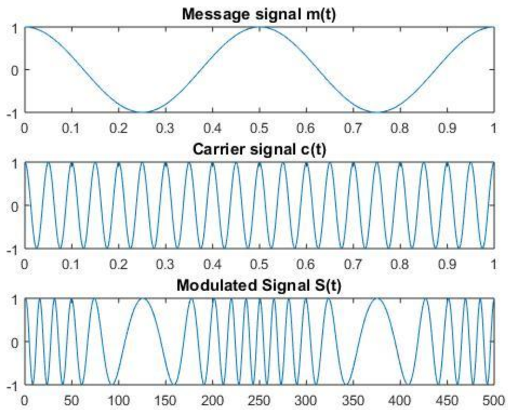
c(t) = carrier signal > c(t) = Ac sin(2*pie*f_c*t)

f_c = carrier frequency

f_m = message frequency

beta = modulation index > beta = k_f*A_m/f_m

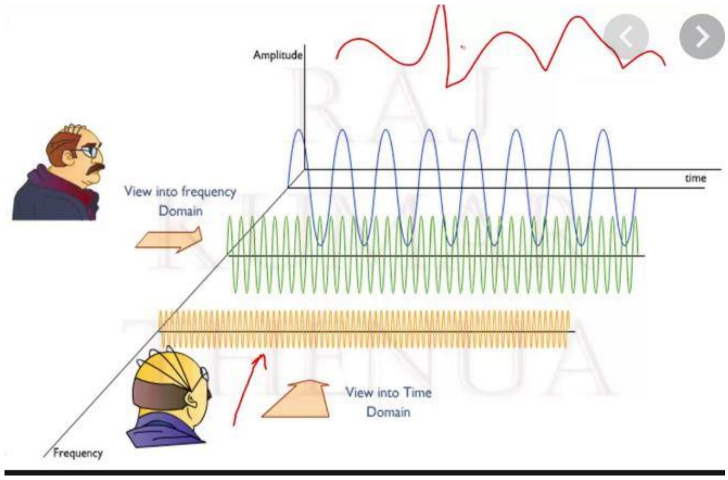
So S_{FM} (t) = A_c * cos [2*pie*f_c*t + beta*sin(2*pie*f_m*t)]



Code

```
>> Kf = 14;
>> Am = 1;
>> fm = 2;
>> beta = (Kf*Am)/fm;
>> t = linspace(0,1,500);
>> Ac = 1;
>> fc = 20;
linspace(startpoint,endpoint,no.ofpointsininterval)
>> Sfm = Ac*cos(2*pi*fc*t + beta*(sin(2*pi*fm*t)));
>> mt = Am*cos(2*pi*fm*t);
>> ct = Ac*cos(2*pi*fc*t);
>> subplot(3,1,1)
>> plot(t,mt)
>> title('Message signal m(t)')
>> subplot(3,1,2)
>> plot(t,ct) % plot(ct) same
>> title('Carrier signal c(t)')
>> subplot(3,1,3)
>> plot(Sfm);
>> title('Modulated Signal S(t)')
```

Time domain to frequency domain

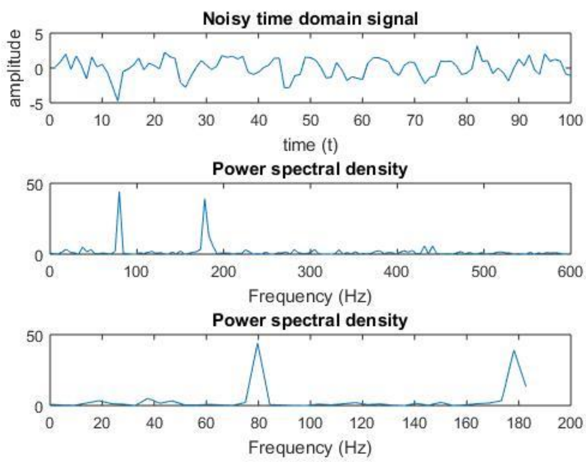


% Consider a signal sampled at 1.2kHz which is composed of 80Hz and 150Hz of sine waves and it is contaminated by a random noise.

% Write a matlab program to plot noisy signal in time domain as well as in frequency domain to get informtaion about major frequency

% components present in the signal

```
fs = 1200;
t = 0:1/fs:1;
x = sin(2*pi*80*t) + sin(2*pi*180*t);
y = x + randn (size(t));
subplot(3,1,1)
plot(y(1:100))
title('Noisy time domain signal')
xlabel('time (t)');
ylabel('amplitude');
Y = fft(y,256);
Pyy = Y.*conj(Y)/300;
f = fs/256*(0:127);
subplot(3,1,2)
plot(f,Pyy(1:128))
title('Power spectral density')
xlabel('Frequency (Hz)')
subplot(3,1,3)
plot(f(1:40),Pyy(1:40))
title('Power spectral density')
xlabel('Frequency (Hz)')
```



String

 # Declaring string

```
>> str = 'this is a test';

>> whos

  Name      Size            Bytes  Class    Attributes

  str       1x14               28  char
```

 # Converting string

```
>> x = double(str) %ascii value

x = 116   104   105   115    32   105   115    32    97    32   116   101
115   116

>> z = char(x)

z = this is a test
```

 # 2D string

```
>> name = char('Shahriar Ahmad','Student');

>> name

name =

Shahriar Ahmad

Student
```

 # Joining Strings

```
>> str1 = 'I'; str2 = ' am Shahriar'; str3 = strcat(str1,str2)

str3 =

I am Shahriar

>> str4 = strvcat(str1,str2)

str4 =

I

 am Shahriar
```

 # Comparing Strings

```
>> str1 = 'hello'; str2 = 'Hello'; str3 = 'hello';

>> strcmp(str1,str2)

ans = 0 % False

>> strcmp(str1,str3)

ans = 1 % True

>> strcmpi(str1,str2)

ans = 1

>> strcmp(str1,str2,n) %determines for first n characters
```

 # Comparing individual characters

```
>> a = 'fate';
>> b = 'cake';
>> c = a==b
c =
     0     1     0     1
```

 # Finding letters in strings

```
>> d = isletter(str)
d =
     1     1     1     1     0     0     0     1
```

 # Replace string

```
strrep(string,search,replace)
>> str = 'what is the test of this ice-cream?';
>> replacedstr = strrep(str,'test','taste')
replacedstr =
what is the taste of this ice-cream?
```

Sparse Array

 # eye to sparse array

```
>> a = eye(5)

a =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

>> as = sparse(a)

as =

   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1

>> whos
  Name      Size            Bytes  Class     Attributes

  a         5x5               200  double
  as        5x5               128  double    sparse
```

 # Alternate way: `>> a = speye(10)`

 # sparse to eye: `>> a = full(b)`

 # Functions

find() > find indices of non-zero elements

nnz() > number of non-zero elements

spy() > visualize sparsity pattern as a plot

issparse() > returns 1 for sparse matrix

Cell Arrays

 # Declaring cell arrays using assignment statements

```
>> a{1,1}=[1 3 -7; 2 0 6; 0 5 1];
>> a{1,2}='This is a test string';
>> a{2,1}=[3+4*i -5; -1*i 3-4*i];
>> a{2,2}= [ ];
>> a
a =
    [3x3 double]    'This is a test string'
    [2x2 double]                         []

>> a(1,1)
ans =
    [3x3 double]
>> a{1,1}
ans =
     1     3    -7
     2     0     6
     0     5     1
>> c=a{1,1}(2,3)
c =
     6
```

 # Alternate way to declare

```
a(1,1)={[1 3 -7; 2 0 6; 0 5 1]};
a(1,2)={'This is a test string'};
a(2,1)={[3+4*i -5; -1*i 3-4*i]};
a(2,2)= {[ ]};
```

 # Empty cell arrays

```
>> a = cell(2,2);
>> a
a =
    []    []
    []    []
```

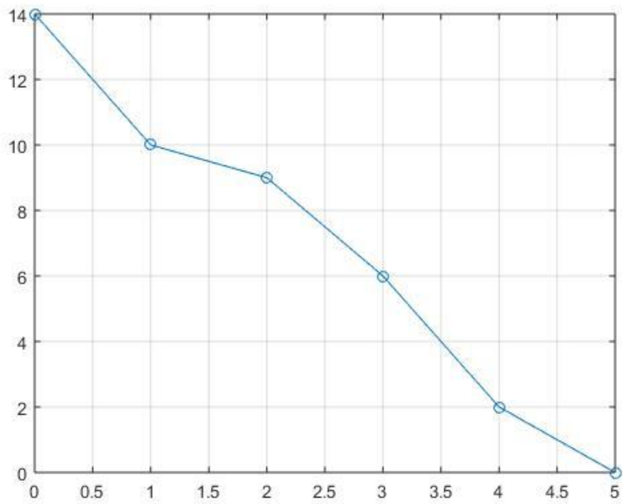
 # Plot cell array

```
>> cellplot(a)
```

Interpolation

 # interp1(x,y,new_x)

```
x = [0:5];
y= [14,10,9,6,2,0];
plot(x,y,'-o')
grid on
new_x = 3.5;
new_y = interp1(x,y,new_x)
new_y =
    4.0000
```



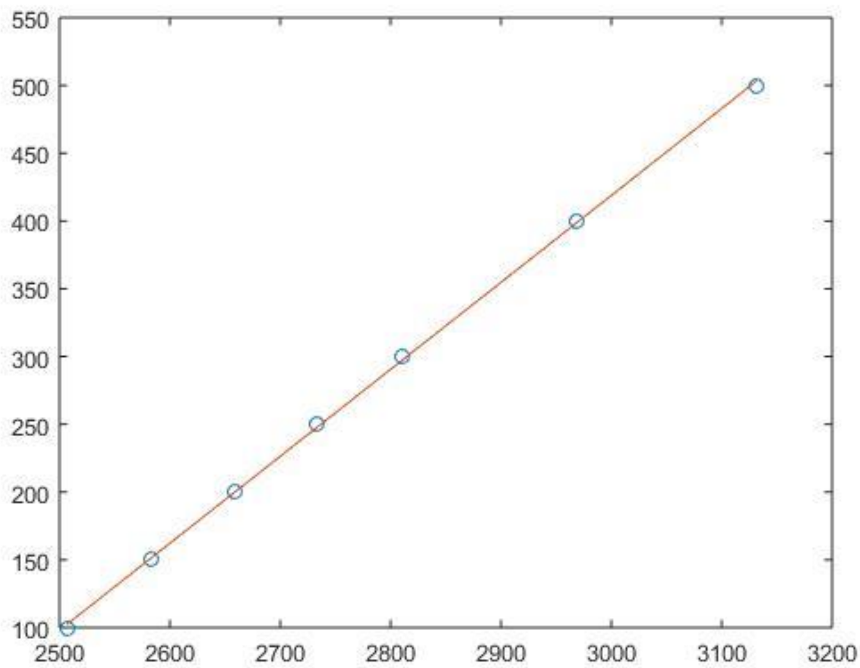
 # Methods

```
Vq = interp1(X,V,Xq,METHOD) specifies alternate methods.
    The default is linear interpolation. Use an empty matrix [] to specify
    the default. Available methods are:
       'nearest'  - nearest neighbor interpolation
      'next'     - next neighbor interpolation
      'previous' - previous neighbor interpolation
      'linear'   - linear interpolation
      'spline'   - piecewise cubic spline interpolation (SPLINE)
      'pchip'    - shape-preserving piecewise cubic interpolation
      'cubic'    - same as 'pchip'
      'v5cubic'  - the cubic interpolation from MATLAB 5, which does not
                   extrapolate and uses 'spline' if X is not equally
                   spaced.
```

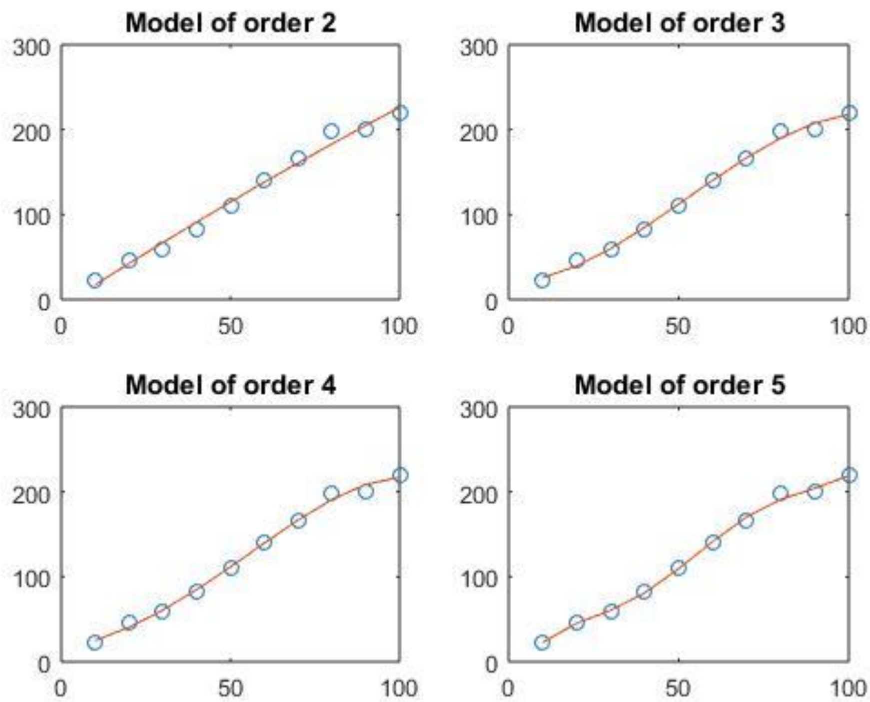
Curve Fitting

 # Linear Regression

```
T = [100,150,200,250,300,400,500];
u = [2506.7,2582.8,2658.1,2733.7,2810.4,2967.9,3131.6];

n = 1;

p = polyfit(u,T,n);
% y = ax+b
a = p(1);
b = p(2);
x= u;
ymodel = a*x+b;

plot(u,T,'o',u,ymodel);
```



```
# Polynomial Regression
x = [10:10:100];
y = [23,45,60,82,111,140,167,198,200,220];

for n=2:5
    p=polyfit(x,y,n);
    ymodel = polyval(p,x);
    subplot(2,2,n-1)
    plot(x,y,'o',x,ymodel);
    title(sprintf('Model of order %d',n));
end
```



Optimization

 # Minimum value of a polynomial

```
>> % minimum of a polynomial
>> x = -20:0.1:20;
>> y = 2.*x.^2 + 20.*x - 22;
>> plot(x,y)
>> grid on
>> i=1;
>> while (y(i) > y(i+1))
i = i + 1;
end
>> x(i)
ans =
    -5
>> y(i)
ans =
   -72
```