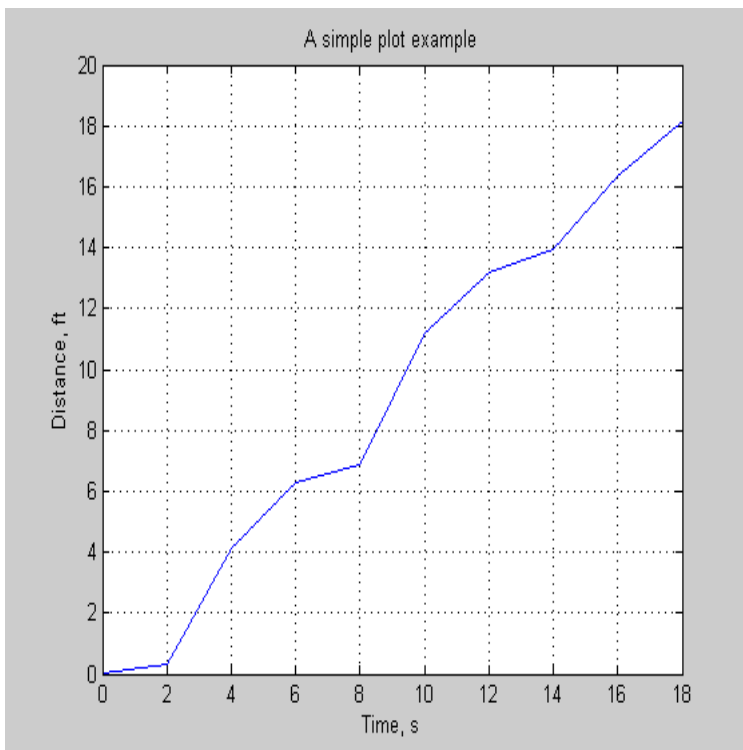

Lect02 - Matlab graphics & some useful functions

Simple x-y plot from given data

Example

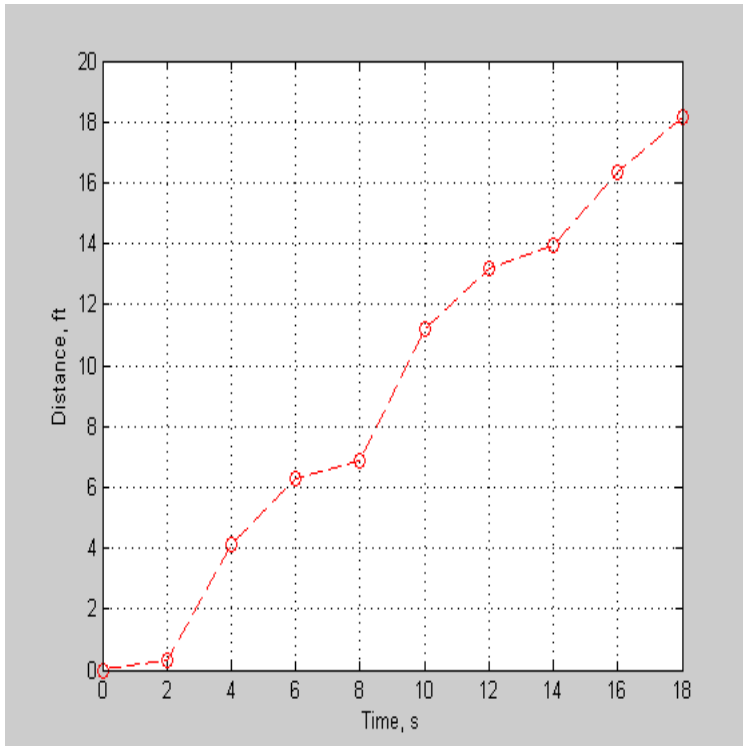
| time, sec | Distance, Ft |
|-----------|--------------|
| 0 | 0 |
| 2 | 0.33 |
| 4 | 4.13 |
| 6 | 6.29 |
| 8 | 6.85 |
| 10 | 11.19 |
| 12 | 13.19 |
| 14 | 13.96 |
| 16 | 16.33 |
| 18 | 18.17 |

```
time = [0:2:18];  
dist = [0, 0.33, 4.13, 6.29, 6.85, 11.19, 13.19, 13.96, 16.33, 18.17];  
plot(time, dist)  
xlabel('Time, s')  
ylabel('Distance, ft')  
title('A simple plot example')  
grid on
```



Controlling colors and line styles

```
plot(time, dist, '--or')
xlabel('Time, s')
ylabel('Distance, ft')
grid on
```



| Line Type | Indicator | Point Type | Indicator | Color | Indicator |
|-----------|-----------|----------------|-----------|---------|-----------|
| solid | - | point | . | blue | b |
| dotted | : | circle | o | green | g |
| dash-dot | -. | x-mark | x | red | r |
| dashed | -- | plus | + | cyan | c |
| | | star | * | magenta | m |
| | | square | s | yellow | y |
| | | diamond | d | black | k |
| | | triangle down | v | | |
| | | triangle up | ^ | | |
| | | triangle left | < | | |
| | | triangle right | > | | |

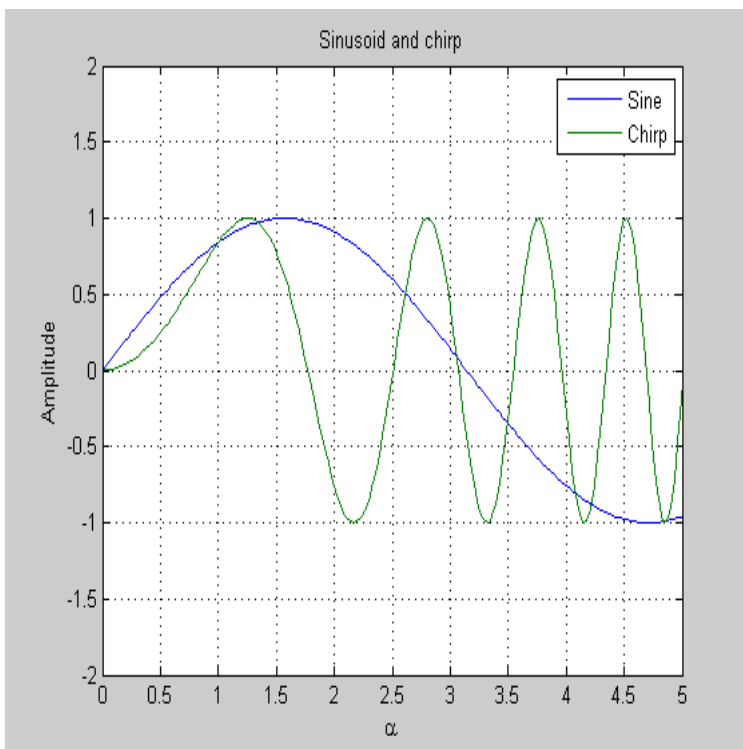
Plots with multiple lines on the same graph

Matlab automatically scales each plot to completely fill the graph. If you want to specify a different axis – use the axis command.

```
axis([xmin, xmax, ymin, ymax])
```

You can use Greek letters in your labels by putting a backslash (\) before the name of the letter. For example: title('\alpha \beta \gamma') creates the plot title $\alpha \beta \gamma$. To create a superscript use '^' in curly brackets title('x^{21}') gives x^{21} . To create a subscript use '_' (underscore) in curly brackets title('This is plot of \alpha_{21}') gives "This is plot of α_{21} ".

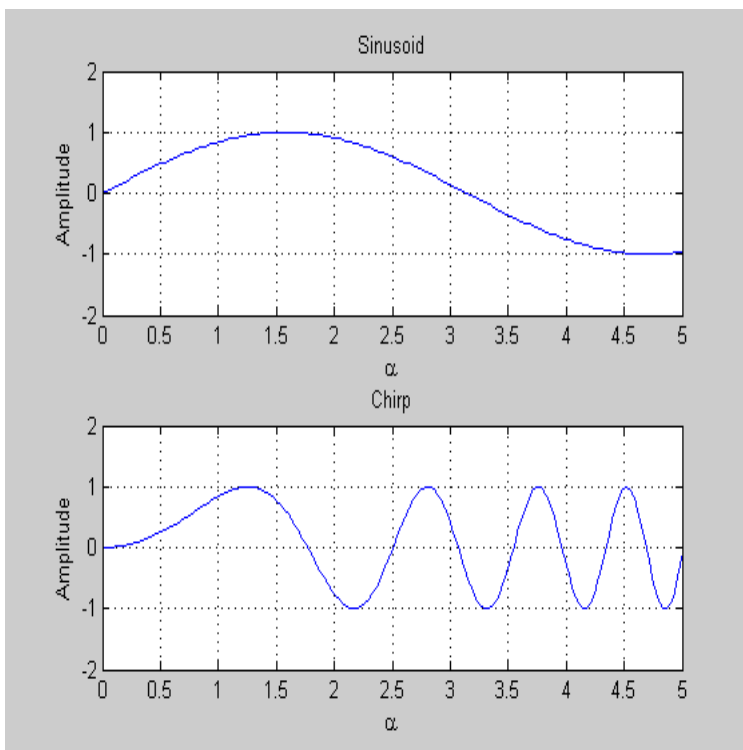
```
alpha = [0:pi/100:2*pi];
plot(alpha, sin(alpha), alpha, sin(alpha.^2))
xlabel('\alpha'); ylabel('Amplitude');
axis([0 5 -2 2]); grid on;
title('Sinusoid and chirp') legend('Sine', 'Chirp')
```



Subplots

An array of graphs can be created by dividing the figure window into several regions using the subplot function. For example in the subplot(2,1,1) the first two entries indicate that the figure window is divided into two rows and one column. The third entry specifies that the next graph is to be shown in the first row.

```
subplot(2,1,1)
plot(alpha, sin(alpha))
xlabel('\alpha'); ylabel('Amplitude');
axis([0 5 -2 2]); grid on;
title('Sinusoid')
subplot(2,1,2)
plot(alpha, sin(alpha.^2))
xlabel('\alpha'); ylabel('Amplitude');
axis([0 5 -2 2]); grid on;
title('Chirp')
```



Class Activity 1

Plot the following polynomial and its first derivative between $x = -2$ and $x = 2$ using 20 points.

$$f(x) = x^4 + 3x - 1; \quad f'(x) = 4x^3 + 3$$

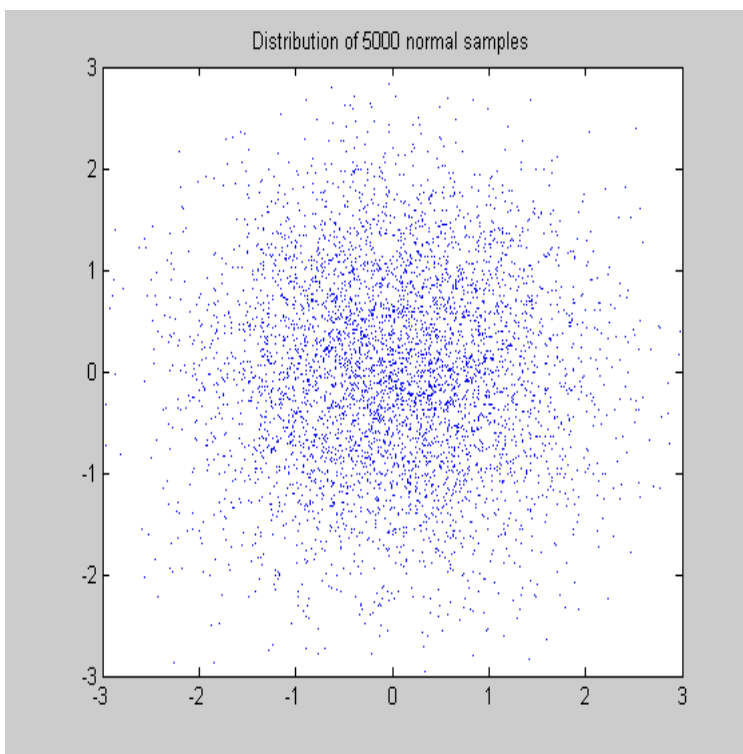
Show the two plots in different styles or as subplots. Show legend indicating which line represents f and which its derivative. Create appropriate axes labels, legend, and plot title.

Plot of large sample data

Matlab function **rand(n)** returns an $n \times n$ matrix of uniformly distributed random numbers between 0 and 1. The function **randn(n)** does the same thing except the numbers are normally distributed with mean 0 and standard deviation 1.

B = reshape(A,m,n) returns the m -by- n matrix **B** whose elements are taken column-wise from **A**. An error results if **A** does not have $m*n$ elements.

```
n=100; data=randn(n);  
xy=reshape(data,2,n*n/2);  
plot(xy(1,:),xy(2,:),'.','MarkerSize',1)  
axis([-3 3 -3 3])  
title('Distribution of 5000 normal samples')
```



Plots of functions of two variables

`[X, Y] = meshgrid(x, y)` transforms the domain specified by vectors `x` and `y` into arrays `X` and `Y`, which can be used to evaluate functions of two variables and three-dimensional mesh/surface plots. The rows of the output array `X` are copies of the vector `x`; columns of the output array `Y` are copies of the vector `y`.

```
x=[1, 2, 3, 4, 5]
```

```
x =
     1     2     3     4     5
```

```
y=[-1, 0, 1]
```

```
Y =
    -1     0     1
```

```
[X,Y]=meshgrid(x,y)
```

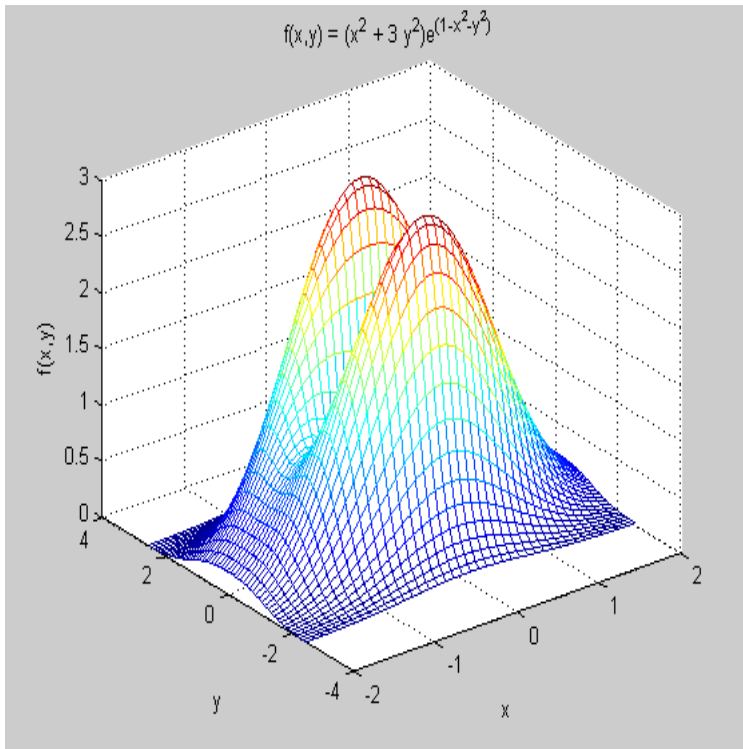
```
X =
     1     2     3     4     5
     1     2     3     4     5
     1     2     3     4     5
Y =
    -1    -1    -1    -1    -1
     0     0     0     0     0
     1     1     1     1     1
```

`mesh(X,Y,Z)` draws a wireframe mesh with color determined by `Z` so color is proportional to surface height.

Example: Plot the following function in the area defined by $-2 \leq x \leq 2$ and $-2.5 \leq y \leq 2.5$

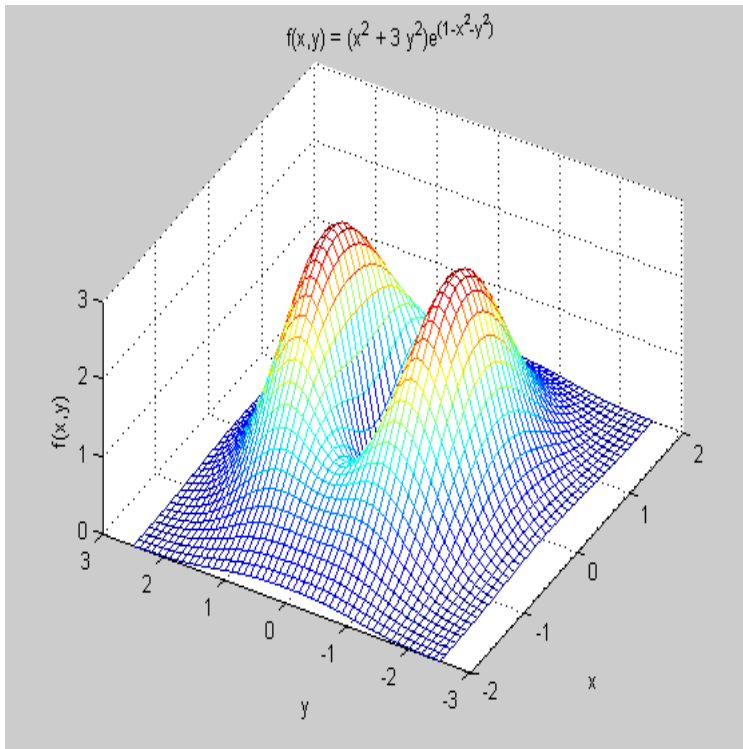
$$f(x, y) = (x^2 + 3y^2)e^{(1-x^2-y^2)}$$

```
x=[-2:0.1:2]; y=[-2.5:.1:2.5];
[X, Y] = meshgrid(x,y);
fxy = (X.^2 + 3*(Y.^2)) .* exp(1-X.^2 -Y.^2)
mesh(x, y, fxy)
xlabel('x'); ylabel('y'); zlabel('f(x,y)');
title('f(x,y) = (x^2 + 3 y^2)e^{(1-x^2-y^2)}')
```



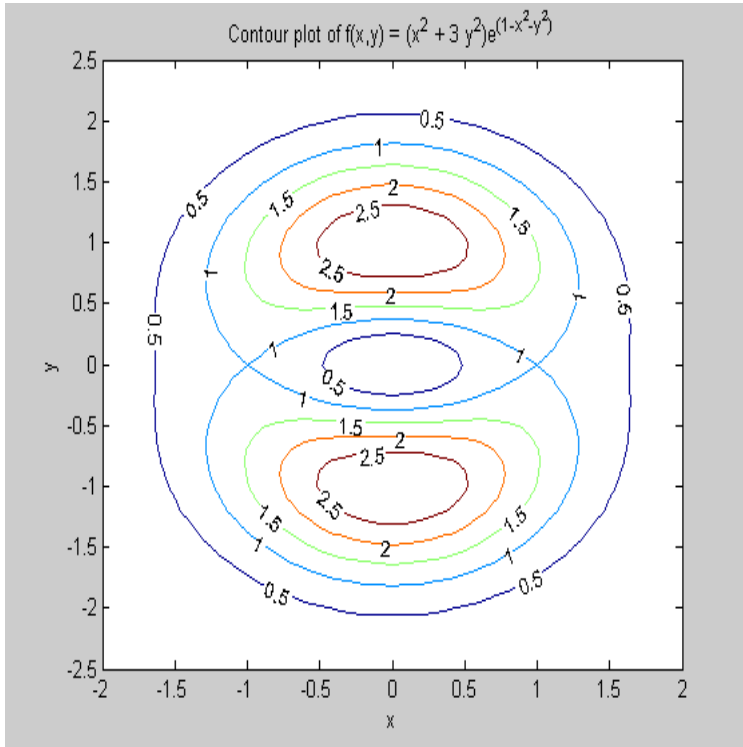
view(az,el) sets the viewing angle for a three-dimensional plot. The azimuth, az, is the horizontal rotation about the z-axis as measured in degrees from the negative y-axis. Positive values indicate counterclockwise rotation of the viewpoint. el is the vertical elevation of the viewpoint in degrees. Positive values of elevation correspond to moving above the object; negative values correspond to moving below the object.

```
mesh(x, y, fxy)
view(-60, 50)
xlabel('x'); ylabel('y'); zlabel('f(x,y)');
title('f(x,y) = (x^2 + 3 y^2)e^{(1-x^2-y^2)}')
```



contour(X,Y,Z) draws contour plots of Z. X and Y specify the x- and y-axis limits. **contour**(X,Y,Z, n) specifies to draw n contour levels.

```
contour(x, y, fxy, 5, 'ShowText', 'on')
xlabel('x'); ylabel('y');
title('Contour plot of f(x,y) = (x^2 + 3 y^2)e^{(1-x^2-y^2)}')
```



Class Activity 2

Create mesh and contour plots of the following function.

$$f(x, y) = 3x e^y - x^3 - e^3 y; \quad -\infty \leq x \leq \infty, \quad -\infty \leq y \leq \infty$$

Since the domain is infinite use the following transformation to compress the entire plane to a finite square

$$x = \tan(s) \text{ and } y = \tan(t)$$

and plot

$$\tan^{-1}(f(\tan(s), \tan(t))); \quad -\pi/2 \leq s \leq \pi/2, \quad -\pi/2 \leq t \leq \pi/2$$

Some useful functions

sum & prod functions

$B = \mathbf{sum}(A)$ returns sums along different dimensions of an array. If A is a vector, $\mathbf{sum}(A)$ returns the sum of the elements. If A is a matrix, $\mathbf{sum}(A)$ treats the columns of A as vectors, returning a row vector of the sums of each column.

$B = \mathbf{prod}(A)$ returns the products along different dimensions of an array. If A is a vector, $\mathbf{prod}(A)$ returns the product of the elements. If A is a matrix, $\mathbf{prod}(A)$ treats the columns of A as vectors, returning a row vector of the products of each column.

```
a = [1 2 3; 4 5 6; 7 8 9]
```

```
a =  
    1     2     3  
    4     5     6  
    7     8     9
```

```
sum(a)
```

```
ans =  
    12    15    18
```

```
prod(a)
```

```
ans =  
    28    80   162
```

zeros & ones functions

B = zeros(n) returns an n-by-n matrix of zeros. An error message appears if n is not a scalar. **B = zeros(m,n)** or **B = zeros([m n])** returns an m-by-n matrix of zeros.

Y = ones(n) returns an n-by-n matrix of 1s. An error message appears if n is not a scalar. **Y = ones(m,n)** or **Y = ones([m n])** returns an m-by-n matrix of ones.

```
zeros(5)
```

```
ans =  
    0    0    0    0    0  
    0    0    0    0    0  
    0    0    0    0    0  
    0    0    0    0    0  
    0    0    0    0    0
```

```
ones(3,5)
```

```
ans =  
    1    1    1    1    1  
    1    1    1    1    1  
    1    1    1    1    1
```

linspace & logspace functions

$y = \mathbf{linspace}(a,b,n)$ generates a row vector y of n points linearly spaced between and including a and b .

$y = \mathbf{logspace}(a,b,n)$ generates n points between decades 10^a and 10^b .

```
linspace(-2,2,5)
```

```
ans =  
    -2    -1     0     1     2
```

```
logspace(-2,2,5)
```

```
ans =  
    0.0100    0.1000    1.0000   10.0000  100.0000
```

Rounding Functions

fix(x): rounds the elements of x toward zero, resulting in an array of integers

floor(x): rounds the elements of x to the nearest integers less than or equal to x.

ceil(x): rounds the elements of x to the nearest integers greater than or equal to x.

round(x): rounds the elements of x to the nearest integers.

```
x=exp(rand(1,8))
```

```
x =
  Columns 1 through 5
    1.3128    1.2200    1.0154    2.1102    1.5606
  Columns 6 through 8
    2.5391    1.5936    1.5199
```

```
fix(x)
```

```
ans =
     1     1     1     2     1     2     1     1
```

```
floor(x)
```

```
ans =
     1     1     1     2     1     2     1     1
```

```
ceil(x)
```

```
ans =
     2     2     2     3     2     3     2     2
```

```
round(x)
```

```
ans =
     1     1     1     2     2     3     2     2
```

mod & rem function

The function mod returns modulus after division. It performs the operation floor on number and divisor and returns the remainder of the floor operation. $M = \mathbf{mod}(X,Y)$ returns $X - n.*Y$ where $n = \text{floor}(X./Y)$.

The function rem returns remainder after division. It performs the operation truncate on number and divisor and returns the remainder of the truncate operation. $R = \mathbf{rem}(X,Y)$ returns $X - n.*Y$ where $n = \text{fix}(X./Y)$. The inputs X and Y must be real arrays of the same size, or real scalars.

```
[mod(10,3), rem(10,3), mod(9,3), rem(9,3)]
```

```
ans =
     1     1     0     0
```

```
y=2*ones(1,5);
x=exp(rand(1,5))
```

```
x =
    2.2737    1.5600    1.8505    2.2077    2.5138
```

```
mod(x,y)
```

```
ans =
    0.2737    1.5600    1.8505    0.2077    0.5138
```

```
rem(x,y)
```

```
ans =
    0.2737    1.5600    1.8505    0.2077    0.5138
```

Data analysis functions

```
max(x)
min(x)
mean(x)
median(x)
sort(x)
std(x)
var(x)

x=exp(rand(1,5))

x =
    1.2198    1.8290    1.3128    1.2200    1.0154

max(x)

ans =
    1.8290

min(x)

ans =
    1.0154

mean(x)

ans =
    1.3194

median(x)

ans =
    1.2200

sort(x)

ans =
    1.0154    1.2198    1.2200    1.3128    1.8290

standard deviation

std(x)

ans =
    0.3050

variance

var(x)

ans =
    0.0930
```

disp & fprintf functions

disp(X) displays an array, without printing the array name. If X contains a text string, the string is displayed.

```
farmData = [[0.2113, 0.0820, 0.7599, 0.0087, 0.8096]', [0.8474, 0.4524, ...
    0.8075, 0.4832, 0.6135]', [0.2749, 0.8807, 0.6538, 0.4899, 0.7741]];
disp('    Corn    Oats    Hay')
disp(farmData)
```

| Corn | Oats | Hay |
|--------|--------|--------|
| 0.2113 | 0.8474 | 0.2749 |
| 0.0820 | 0.4524 | 0.8807 |
| 0.7599 | 0.8075 | 0.6538 |
| 0.0087 | 0.4832 | 0.4899 |
| 0.8096 | 0.6135 | 0.7741 |

fprintf(obj, 'format', 'cmd') writes the string using the format specified by format. format is a C language conversion specification. Conversion specifications involve the % character and the conversion characters d, i, o, u, x, X, f, e, E, g, G, c, and s.

```
A = [6 12 6 91 13 6]
[theMax, theIndex] = max(A);
fprintf('The max value in A is %d at %d\n', theMax, theIndex);
```

```
A =
    6    12     6    91    13     6
the max value in A is 91 at 4
```

Control Structures

For creating your own functions it usually is necessary to control the flow of calculations using if/else family of commands, switch/case structure and for and while loops. All these controls require relational and logical operators. The Matlab function **find** is particularly useful for these logical operations.

Relational Operators

| | |
|----|--------------------------|
| < | Less than |
| <= | Less than or equal to |
| > | Greater than |
| >= | Greater than or equal to |
| == | Equal to |
| ~= | Not equal to |

Recall that an equal sign (=) is the assignment operator, and can not be used for comparisons. Matlab like most computer programs use the number 1 for true and 0 for false.

```
x=5; y=1;
[x<y, x>y, x~=y, x==y]
```

```
ans =
     0     1     1     0
```

For comparisons involving arrays Matlab compares corresponding elements and determines if the result is true or false for each.

```
x=[1:5]; y=[1, 5, 3, 2, 10];
[x<y; x>y; x~=y; x==y]
```

```
ans =
     0     1     0     0     1
     0     0     0     1     0
     0     1     0     1     1
     1     0     1     0     0
```

Logical Operators

& (and):

a & b - True if both a and b are true

~ (not): Changes the state - ~(True) = False.

~(False) = True.

| (or):

a | b - True if either a or b or both are true

xor (exclusive or):

a xor b - True if one of a or b is true, but not both

In order for Matlab to decide a comparison is true for an entire matrix, it must be true for every element in the matrix.

```
x = [1:5]; y = [-2, 0, 2, 4, 6]; z = 8*ones(1,5);
[z>x & z>y; z>x | z>y; xor(z>x, z>y); ~(z>x)]
```

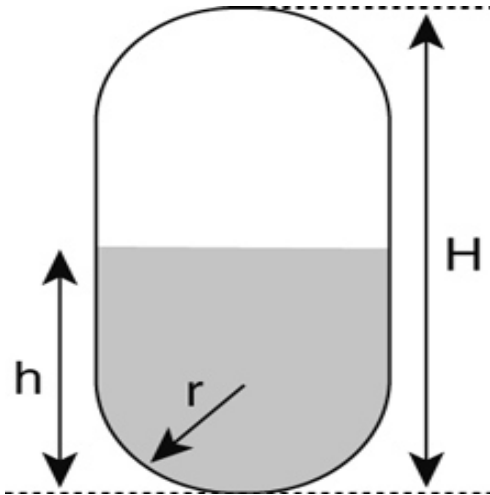
ans =

```
 1     1     1     1     1
 1     1     1     1     1
 0     0     0     0     0
 0     0     0     0     0
```

if/else structure

The simple 'if' triggers the execution of a block of code if a condition is true. If it is false that block of code is skipped, and the program continues without doing anything.

Example: Volume of the liquid in a tank.



If $h < r$ then the volume is that of the partially filled hemisphere given by

$$V = \frac{1}{3} \pi h^2 (3r - h)$$

If $h > r$ but less than $H - r$ then the volume is that of the hemisphere and partially filled cylinder given by

$$V = \frac{2}{3} \pi r^3 + \pi r^2 (h - r)$$

If $h > H - r$ but less than H then the volume is that of the full sphere and cylinder minus the empty portion of the top hemisphere given by

$$V = \frac{4}{3} \pi r^3 + \pi r^2 (H - 2r) - \frac{1}{3} \pi (H - h)^2 (3r - H + h)$$

H = 10; r = 2; h = 4;

if h < r

v = (1/3)*pi*h.^2.*(3*r-h);

elseif h < H-r

v = (2/3)*pi*r^3 + pi*r^2*(h-r);

elseif h <= H

v = (4/3)*pi*r^3 + pi*r^2*(H-2*r) ...
- (1/3)*pi*(H-h)^2*(3*r-H+h);

else

disp('liquid level must be less than the tank height')
continue

end

v

v =
41.8879

switch/case structure

This structure allows you to choose between multiple outcomes, based on some criterion, which must be exactly true. This structure is an alternative to the if/else/elseif structure. The code is generally easier to read with switch/case.

Example:

```
leapYear = false;
month = 2;
switch month
    case {9, 4, 6, 11}
        % Sept, Apr, June, Nov
        days = 30;
    case 2 % Feb
        if leapYear
            days = 29;
        else
            days = 28;
        end
    case {1, 3, 5, 7, 8, 10, 12}
        % other months
        days = 31;
    otherwise
        error('bad month index')
end
days
days =
    28
```

for loop

```
for index = [matrix]
    commands to be executed
end
```

The loop is executed once for each element of the index matrix identified in the first line.

```
A = [6 12 6 91 13 6];
theMax = A(1);
theIndex = 1;
for index = 1:length(A)
    x = A(index);
    if x > theMax
        theMax = x;
        theIndex = index;
    end
end
fprintf('The max value in A is %d at %d\n',theMax, theIndex);
```

The max value in A is 91 at 4

while loop

While loops are very similar to for loops. The big difference is the way Matlab decides how many times to repeat the loop. While loops continue until some criterion is met.

```
while criterion
    commands to be executed
end

A = [6 12 6 91 13 6];
theMax = A(1);
theIndex = 1;
index = 1;
while index <= length(A)
    x = A(index);
    if x > theMax
        theMax = x;
        theIndex = index;
    end
    index = index + 1;
end
fprintf('The max value in A is %d at %d\n', theMax, theIndex);
```

The max value in A is 91 at 4

Watch out for infinite loops. If you accidentally create a loop that just keeps running you can exit the calculation manually by typing **ctrl+c**.