MATLAB

Brief Introduction to MATLAB

Brief Introduction to Matlab

1) Script language
   - Work like a calculate
   - No need to compile

2) Platform independent
   - PC, Linux, Luix, Mac
   - Same lines of code

Brief Introduction to Matlab

3) Powerful graphic and Animation utilities

4) Many tool boxes available
   - Partial differential equations
   - Optimization
   - Statistics
   - Financing, etc

Brief Introduction to Matlab

5) Popular in Scientific and Engineering
Communities
- User groups

6) Easy to implement
- No declaration of variable type, size, etc (not necessary)
- Vectorized

Vector Formats

Define variables

- X= [1 3 5] % x is a row vector (1x3)

- Y= [1;2;3] % y is a column vector (3x1)

It shows:

- X=

  1 3 5
- Y=

  1

  3

  5

Want A and B are 2x2 martrix:

- A= [1 2;3 4] %2x2 matrix
- B= [5 6;7 8] %2x2 matrix

Display:

- A=

  1 2

  3 4

- B=

  5 6

  7 8

Entries of a matrix

- A(2,1) %second row first column entry of A=3
- B(2,2) %second row second column entry of B=8’
- C=[A B] % C is a 2x4

Matlab expressions are:

- Ans=
  3
- Ans =
  8
- Ans=
  1 3
  2 4
- C=
  1 2 5 6
  3 4 7 8

Matrix Transpose & Concatenation

- Transpose(A) is the transpose of A
- A’ is the complex conjugate transpose of A
- When A is a real matrix, transpose(A)=A’
- C=[A’ B’] % C is a 2 x 4
- D=[A;B]

C=

1 3 5 7
2 4 6 8

D=

1 2
3 4
5 6
7 8

Some Matlab built in functions

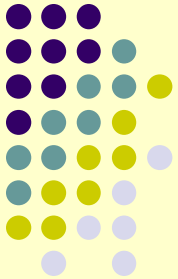
```
>>length(x) % length of x

Length(y) % length of y

Size(x) % dimensions of x

Size(x,1) % number of rows in x

Size(x,2) % number of column in
    x
```

- >> length(x)
  ans =
  3
- >> length(y)
  ans =
  3
- >> size(x)
  ans =
  1 3
- >> size(x,1)
  ans =
  1
- >> size(x,2)
  ans =
  3

If you want to know more: type

- Type help who
- Type help whos
- Type help size
- Type help save
- Save result x y % save x and y into a file called result.mat
- Clear all % clear all variables from workspace
- Load result % load all variables in file called result.mat
- F5 saves and runs
- F9 executes the selected commands

Functions

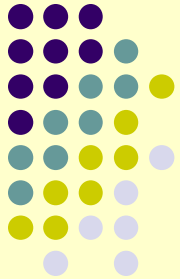


Like C++ or JAVA, MATLAB allows you to write your own functions.

This is done in MATLAB's *Editor.*

Take, for example, the following code:

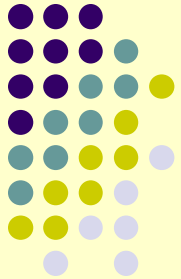
Functions



```
function y = myfunc(x)     %declare functions using the keyword function.
x = [2 4 6];               %declaring x to be the 1x3 matrix.
y = 2 * x;                 %multiply matrix x by 2.
```

Now you can save the function and run it from the Command Window by calling the function. Typing *myfunc* in the command window or using the F5 shortcut will result in the following output:

Functions

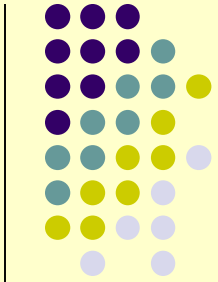


```
function y = myfunc(x)     %declare functions using the keyword function.
x = [2 4 6];               %declaring x to be the 1x3 matrix.
y = 2 * x;                 %multiply matrix x by 2.
```

Output:

```
ans =

     4     8    12
```

Functions



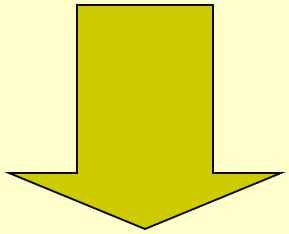
Try substituting y = x * x for y = 2 * x . . .

```
function y = myfunc(x)     %declare functions using the keyword function.
x = [2 4 6];               %declaring x to be the 1x3 matrix.
y = x * x;                 %multiply matrix x by itself.
```

Running this function would result in the following output:

Functions

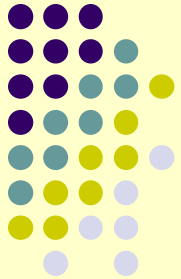
```
function y = myfunc(x)     %declare functions using the keyword function.
x = [2 4 6];               %declaring x to be the 1x3 matrix.
y = x * x;                 %multiply matrix x by itself.
```



??? Error using ==> mtimes

Inner matrix dimensions must agree

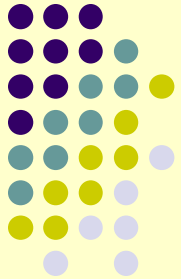
Functions



```
function y = myfunc(x)     %declare functions using the keyword function.
x = [2 4 6];               %declaring x to be the 1x3 matrix.
y = x * x;                 %multiply matrix x by itself.
```

We are asking our function to multiply a 1x3 matrix by a 1x3 matrix. Is this possible? NO. As we know from Linear Algebra, two matrices A and B can be multiplied together if A is of dimension mxn and B is of dimension nxp.

Functions

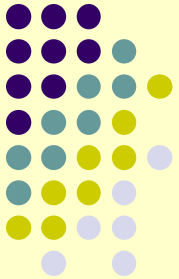


MATLAB provides us with *component multiplication* denoted as:

.*

So, we can write x.*x to multiply *component by component*.

Functions

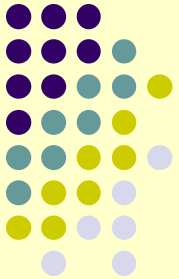


If, for example, we have

$$x = \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix} \quad \text{and} \quad y = \begin{bmatrix} y1 \\ y2 \\ y3 \end{bmatrix}$$

Then, x .* y = $\begin{bmatrix} x1y1 \\ x2y2 \\ x3y3 \end{bmatrix}$

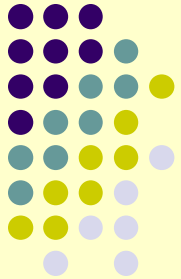
Functions



Similarly, we have

.^	*component power*
./	*component division*

Functions



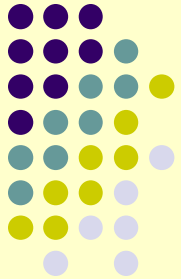
Suppose we wanted our function *myfunc*, to compute

f(x) = 1/(1+x^2)

The correct way to write this would be:

y = 1 ./ (1 + x .^ 2)

Functions



In our function example we obtain an output containing the x-value.

However, we can also have multiple outputs, as in the following example:

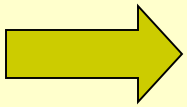
Functions

```
function [y, z] = myfunc(x)
y = 1 ./ (1 + x .^ 2);
z = x .^ 3;
```

Back in the Command Window, we can set x=linspace(0,2*pi,3).

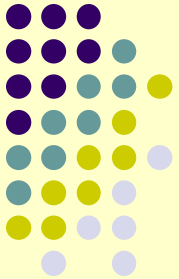
Now we can call our function and observe our “two” outputs:

[y, z] = myfunc(x)



```
y =
    1.0000    0.0920    0.0247

z =
         0   31.0063  248.0502
```

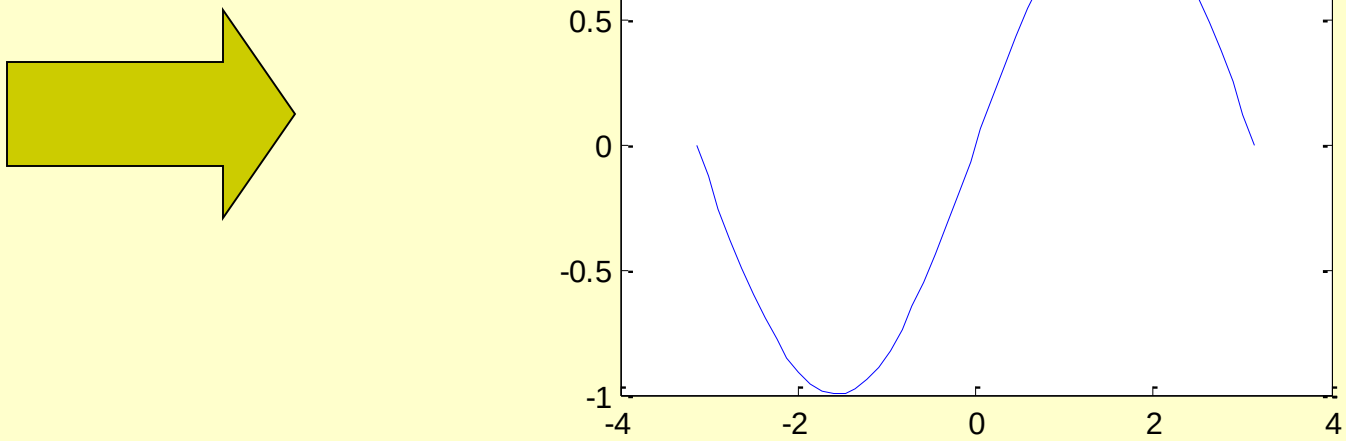
Plots & Subplots



Let’s take a look at some examples of code and the outputs to demonstrate *plotting* in MATLAB.

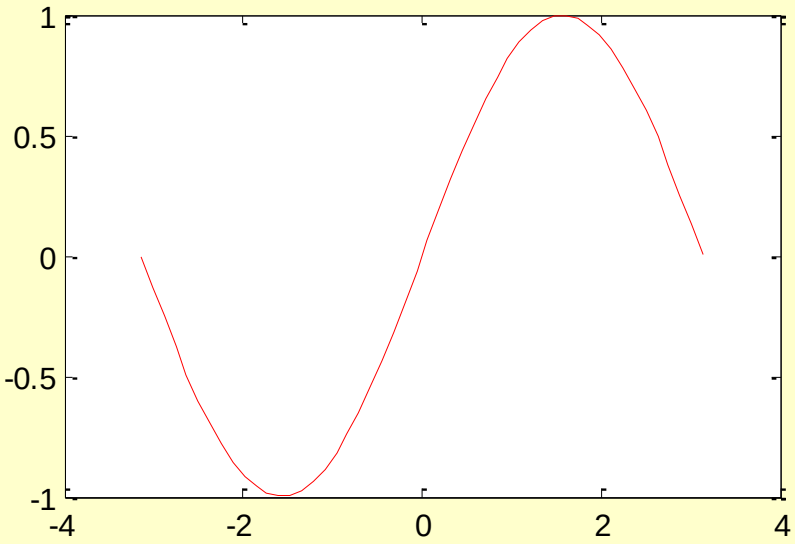
Plots & Subplots

```
x = linspace(-pi, pi, 50);      %x axis will be from –pi to pi, graphing
                                %50 points.
y = sin(x);                     %sine function
z = cos(x);                     %cosine
plot(x, y);                     %will plot the sine function
```



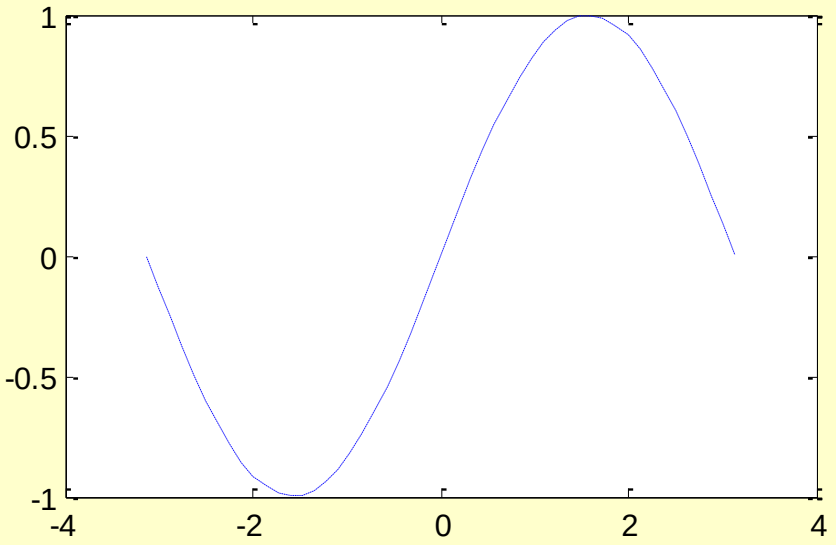
Plots & Subplots

```
x = linspace(-pi, pi, 50);      %x axis will be from –pi to pi, graphing
                                %50 points.
y = sin(x);                     %sine function
z = cos(x);                     %cosine function
plot(x, y, 'r');                %your graph can be in a color of your
                                %choice using the letters r, b, y, g, k,
                                %m, c, w
```



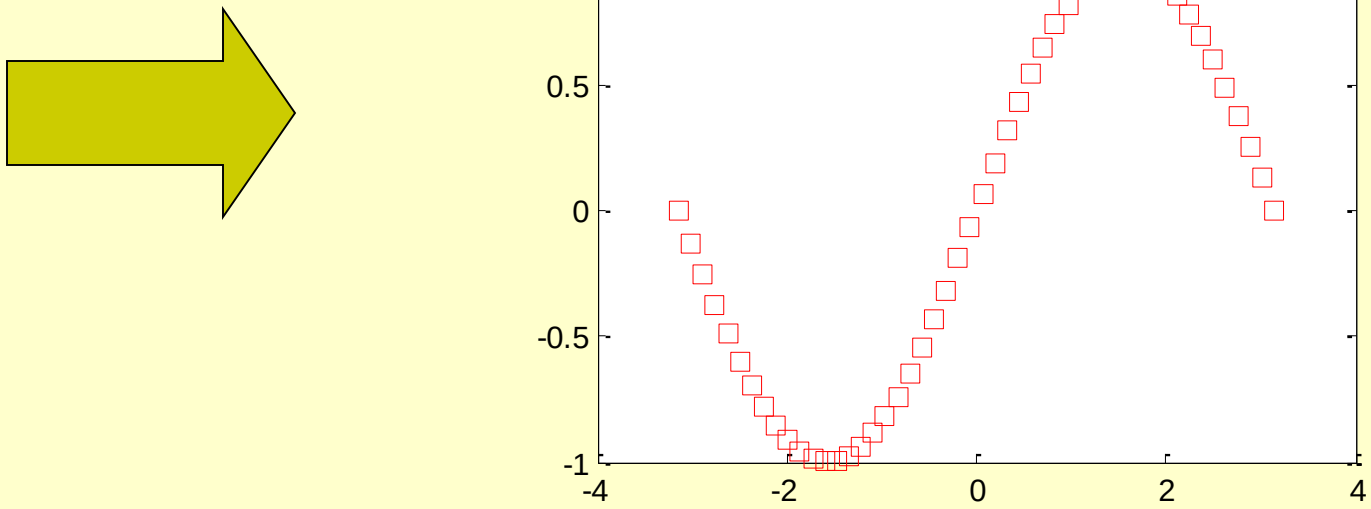
Plots & Subplots

```
x = linspace(-pi, pi, 50);      %x axis will be from –pi to pi, graphing
                                %50 points.
y = sin(x);                     %sine function
z = cos(x);                     %cosine function
plot(x, y, '-.');               %you can change the "line style" of your
                                %graph using -, --, -., o(circle), s(square),
                                %d(diamond), ^, <, >, p(pentagon/star), h
```



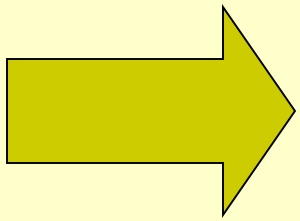
Plots & Subplots

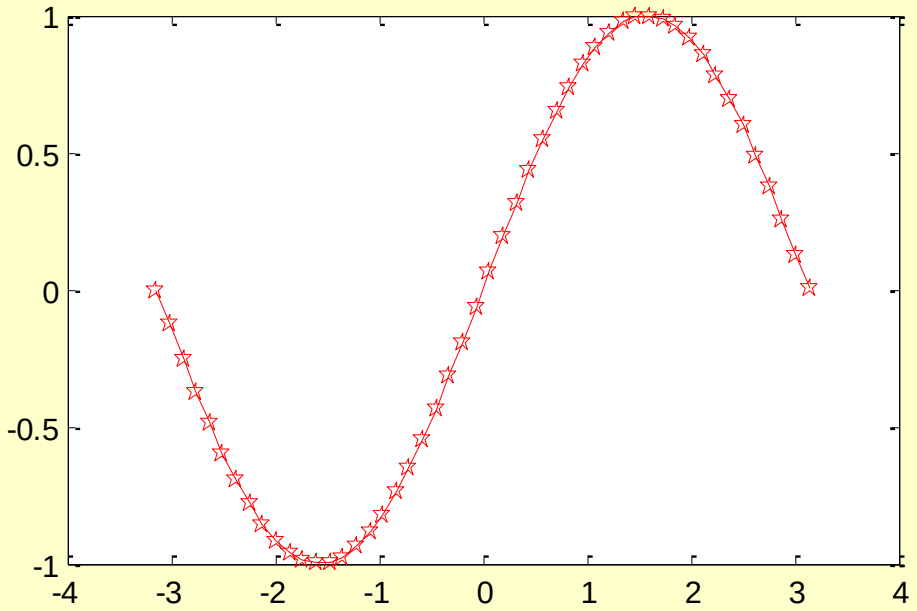
```
x = linspace(-pi, pi, 50);      %x axis will be from –pi to pi, graphing
                                %50 points.
y = sin(x);                     %sine function
z = cos(x);                     %cosine
plot(x, y, 'rs');               %you can
function
have a combination of color
%and line style (i.e. red squares)
```



Plots & Subplots

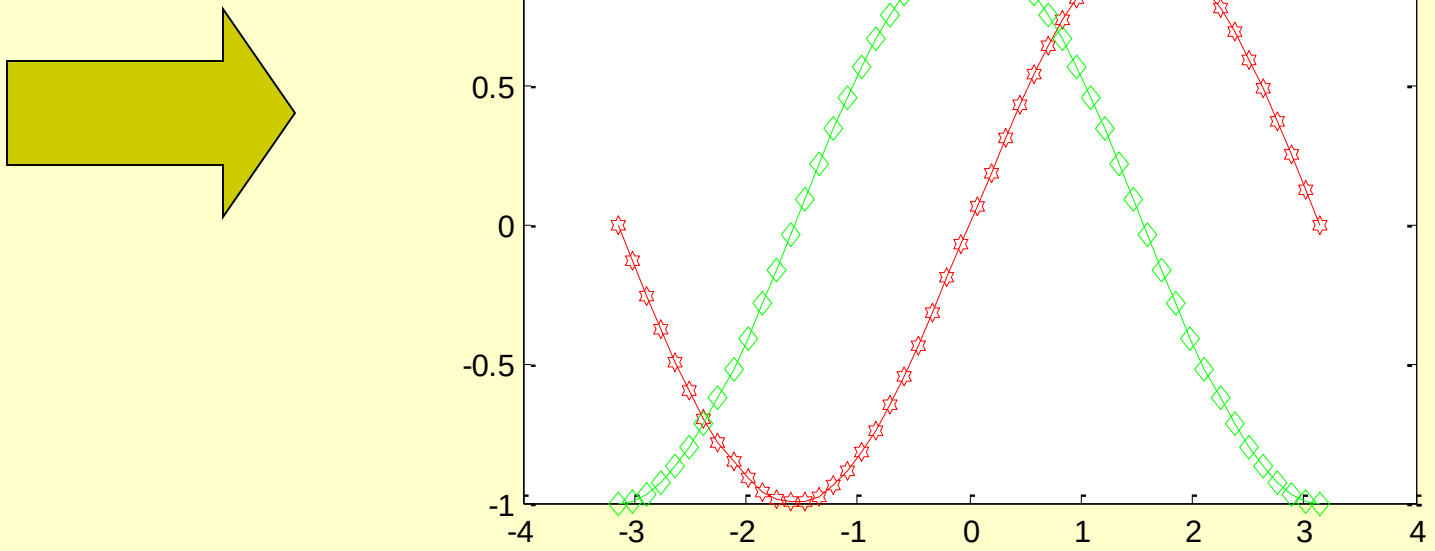
```
x = linspace(-pi, pi, 50);      %x axis will be from –pi to pi, graphing
                                %50 points.
y = sin(x);                     %sine function
z = cos(x);                     %cosine function
plot(x, y, '-rp');              %red stars with line going through
```





Plots & Subplots

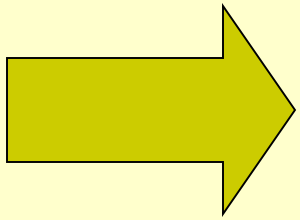
```
x = linspace(-pi, pi, 50);      %x axis will be from –pi to pi, graphing
                                %50 points.
y = sin(x);                     %sine function
z = cos(x);                     %cosine function
plot(x, y, '-rh'); hold on      %"hold on" will keep this graph and plot the next
plot(x, z, '-gd'); hold off     %graph with it.  Otherwise, by default, the next
                                %graph will overwrite the previous graph.
```

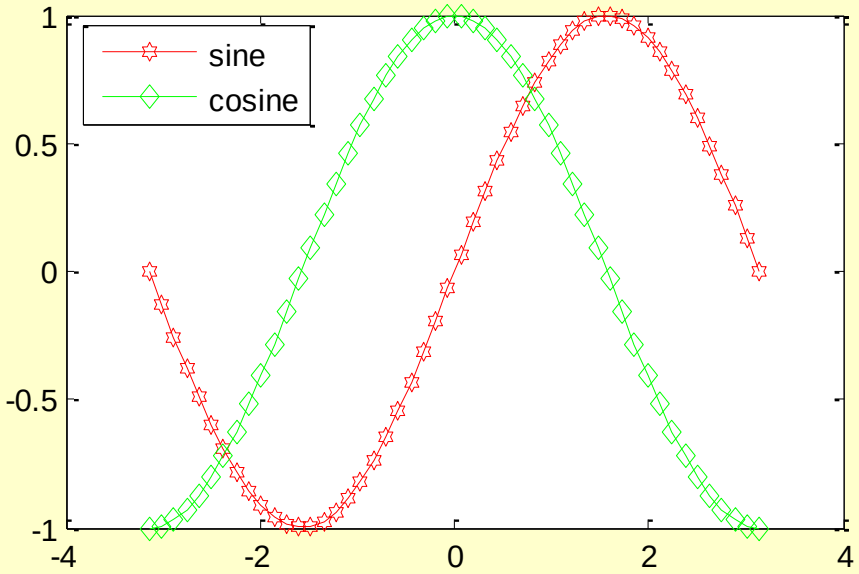


Plots & Subplots

```
x = linspace(-pi, pi, 50);
y = sin(x);
z = cos(x);
plot(x, y, '-rh'); hold on
plot(x, z, '-gd'); hold off
legend('sine', 'cosine', 2)
```

%places legend in position 2



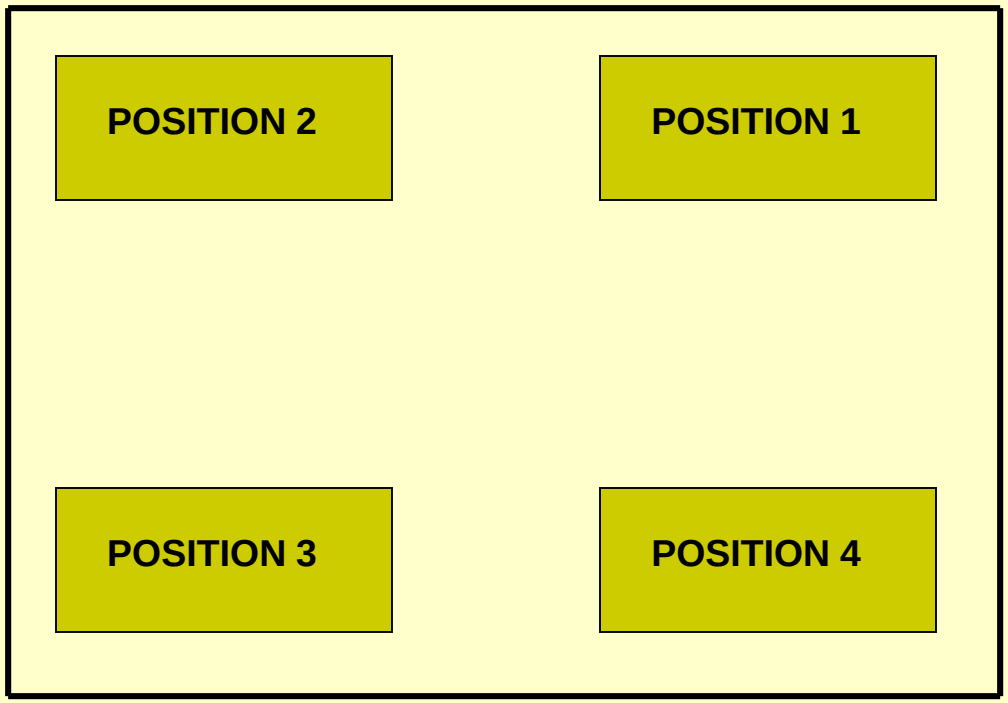


Plots & Subplots

Remark:

If you do not place a position number for the legend, it will, by default, place the legend in position 1. Position 0 tells MATLAB to place the legend in the best location for that graph.

The positions are as followed:



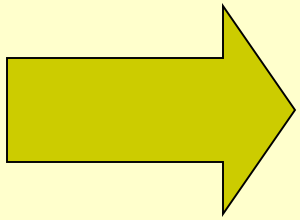
Plots & Subplots

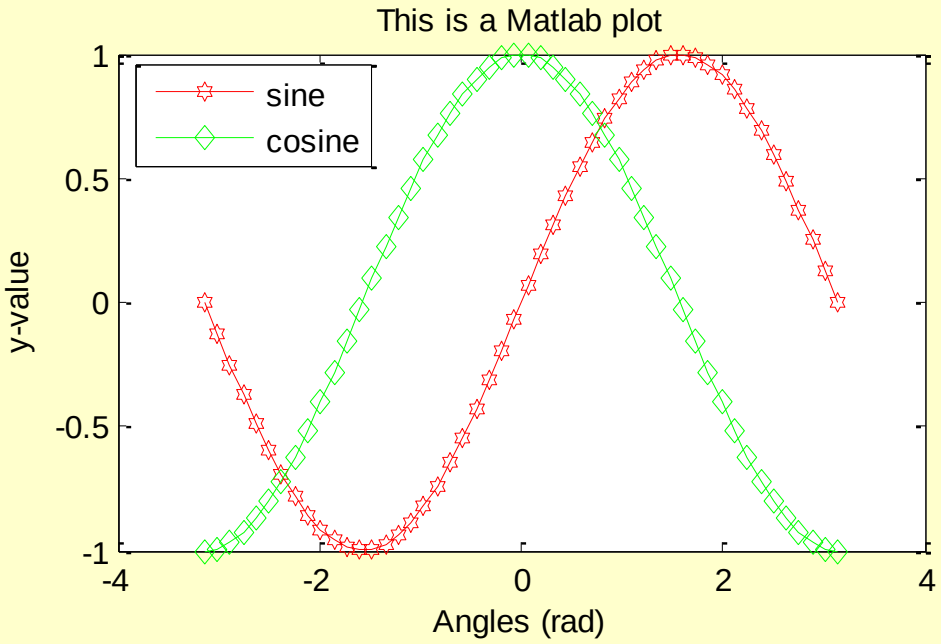
```
x = linspace(-pi, pi, 50);
y = sin(x);
z = cos(x);
plot(x, y, '-rh'); hold on
plot(x, z, '-gd'); hold off
legend('sine', 'cosine', 2)
xlabel('Angles (rad)')
ylabel('y-value')
title('This is a Matlab plot')
```

%labeling the x-axis (x-axis is in radians)

%labeling the y-axis

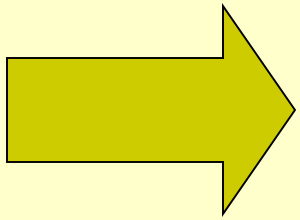
%giving the graph a title

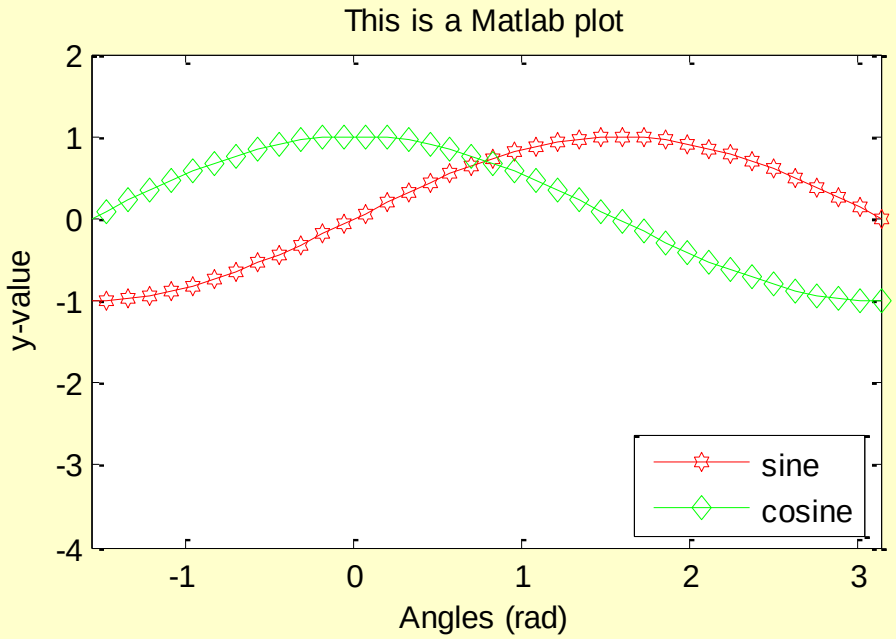




Plots & Subplots

```
x = linspace(-pi, pi, 50);
y = sin(x);
z = cos(x);
plot(x, y, '-rh'); hold on
plot(x, z, '-gd'); hold off
legend('sine', 'cosine', 0) %will place legend in best place
xlabel('Angles (rad)')
ylabel('y-value')
title('This is a Matlab plot')
xlim([-pi/2, pi])          %x-axis from –pi/2 to pi
ylim([-4, 2])              %y-axis from -4 to 2
```





Plots & Subplots

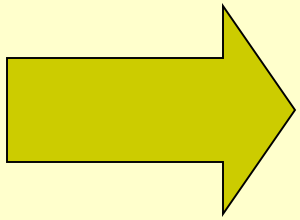
```
x = linspace(-pi, pi, 50);
y = sin(x);
z = cos(x);
subplot(2, 1, 1)
```

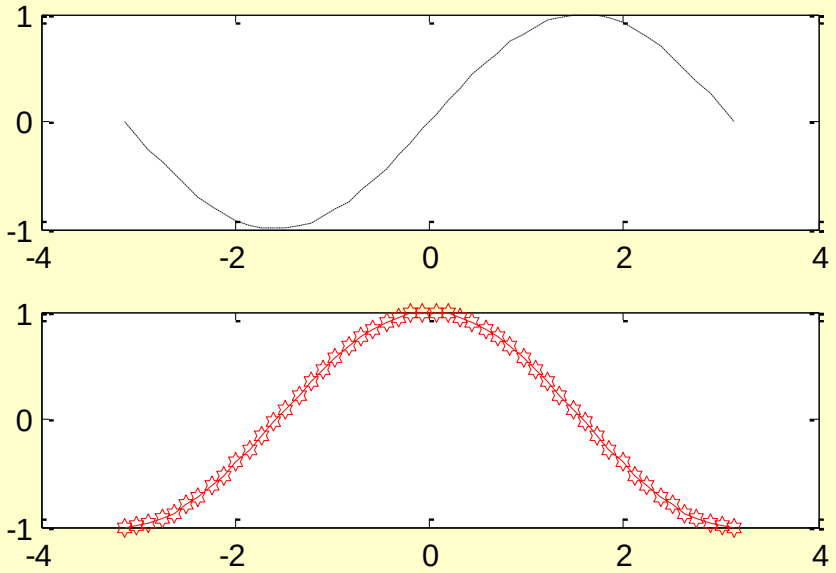
%2 rows, 1 column of plots, will plot the following graph in location 1

```
plot(x, y, '-.k')
subplot(2, 1, 2)
```

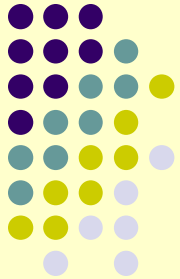
%2 rows, 1 column of plots, will plot the following graph in location 2

```
plot(x, z, '-rh')
```





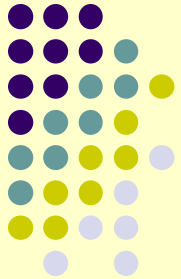
Plots & Subplots



Remark:

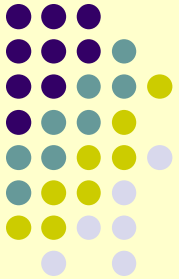
Subplots: When dealing with titles, axis labels, etc., the most recent graph will be effected. If, for example, you want to title the first subplot, you must specify title('title 1').

The following class assignment assigned illustrates subplots, titling subplots, and the other topics discussed in this lecture.



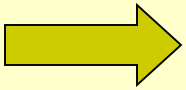
Handling Strings

String Concatenation

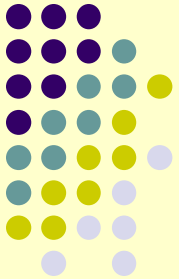


Let s1 = ‘My name is ’

and s2 = ‘Charles’.

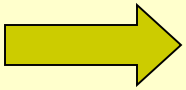
Then s3 = [s1, s2] 

String Concatenation

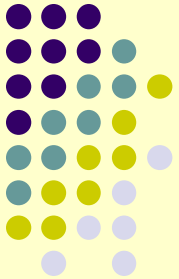


Let s1 = ‘My name is ’

and s2 = ‘Charles’.

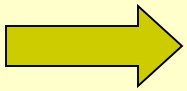
Then s3 = [s1, s2]  My name is Charles

String Concatenation

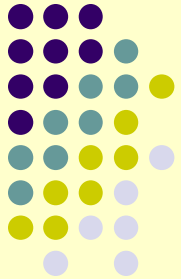


Let s1 = ‘My name is ’

and s2 = ‘Charles’.

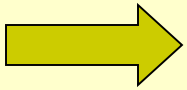
s4 = [s1, ‘Linda’] 

String Concatenation

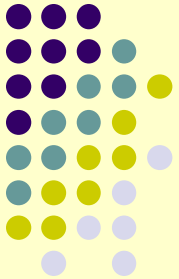


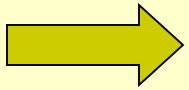
Let s1 = ‘My name is ’

and s2 = ‘Charles’.

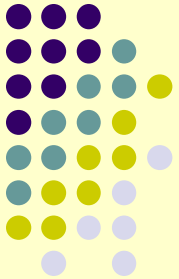
s4 = [s1, ‘Linda’]  My name is Linda

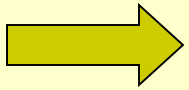
String Concatenation



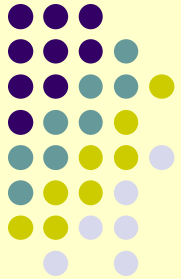
s5 = [‘This is’ ‘a test’] 

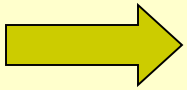
String Concatenation



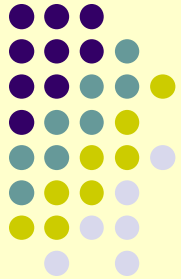
s5 = [‘This is’ ‘a test’]  This is a test

String Concatenation

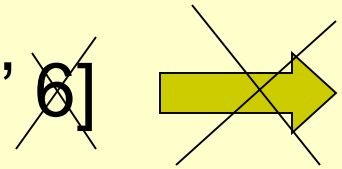


s6 = [‘This is number’ 6] 

String Concatenation

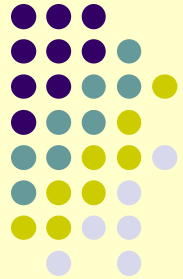


s6 = [‘This is number’ 6]

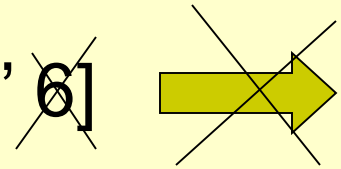


This statement would result in an error because of the “6”. You can not just have a number. String concatenation deals with *strings.*

String Concatenation

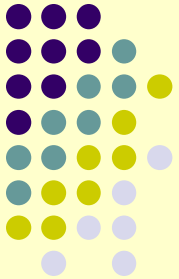


s6 = [‘This is number’ 6]



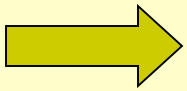
To correct this, the built in MATLAB function, num2str, **is used.** num2str **converts a number into a string.**

String Concatenation

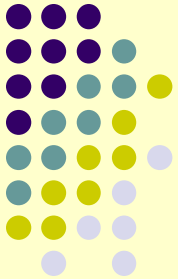


So we have,

s6 = [‘This is number ’ num2str(6)]



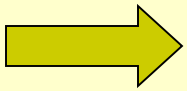
String Concatenation



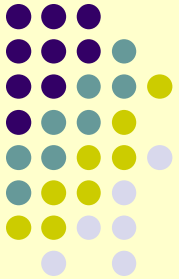
So we have,

s6 = [‘This is number ’ num2str(6)]

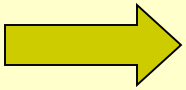
This is number 6



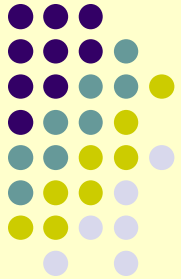
String Concatenation



Let s1 = ‘My answer is ’

Then s2 = [s1, ‘pi’] 

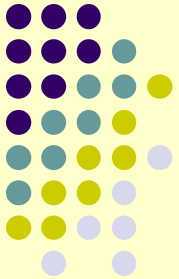
String Concatenation



Let s1 = ‘My answer is ’

Then s2 = [s1, ‘pi’] ⇒ My answer is pi

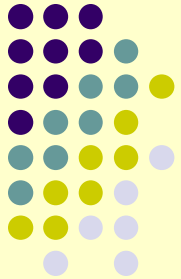
String Concatenation



Let s1 = ‘My answer is ’

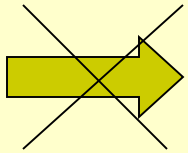
Then s3 = [s1, 4]

String Concatenation



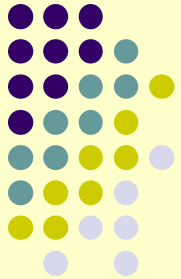
Let s1 = ‘My answer is ’

Then s3 = [s1, 4]



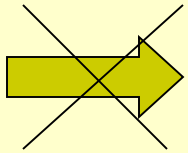
error

String Concatenation



Let s1 = ‘My answer is ’

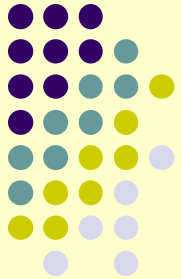
Then s3 = [s1, 4]



error

s3 = [s1, num2str(4)] → My answer is 4

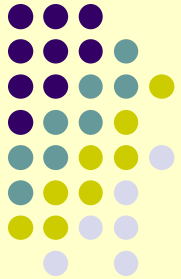
String Concatenation



Let s1 = ‘My answer is ’

Then s4 = [s1, num2str(pi)]

String Concatenation



Let s1 = ‘My answer is ’

Then s4 = [s1, num2str(pi)] → My answer is 3.1416

Class Assignment

Given $f(x) = e^x$ the Taylor approximation for x near 0 is

$$P_N(x) \cong 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^N}{N!}$$

Use the FOR loop and MATLAB subplot command

to plot e^x and $P_N(x) \cong 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^N}{N!}$, for N=1, …, 6 and $-1 < x < 1$. Put the mathematical expression of its approximation on the title of each subplot.