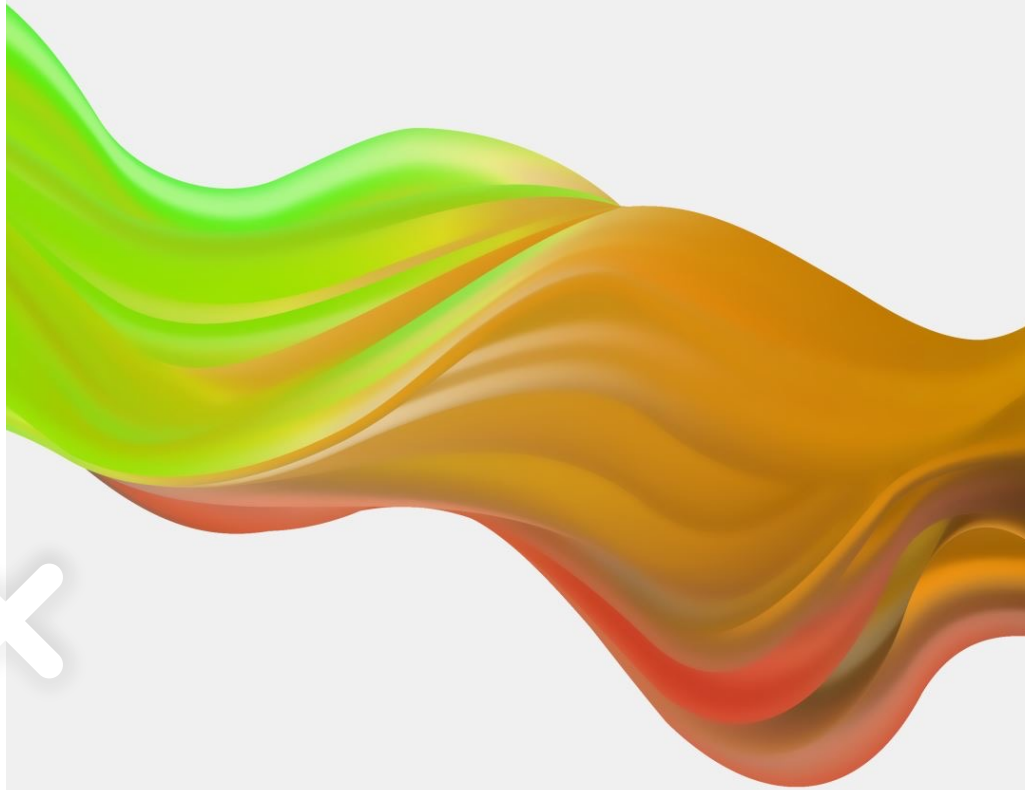
Computer Programming

MENG2020



REVIEW

Matrix Built-In Functions

MATLAB has many built-in functions for working with arrays. Some common ones are:

- `length(v)` : gives the number of elements in a vector.
- `size(A)` : gives the number of rows and columns in a matrix or a vector.
- `reshape(A,m,n)` : changes the number of rows and columns of a matrix or vector while keeping the total number of elements the same. For example, you can change a 4x4 matrix into a 2x8 matrix.

reshape Example

```
>> a = [1 2 ; 3 4]
a =
 1 2
 3 4

>> a = [a , [5 6 ; 7 8]]
a =
 1 2 5 6
 3 4 7 8

>> reshape (a, 1, 8)
ans =
 1 3 2 4 5 7 6 8
```

```
>> reshape (a, 8, 1)

ans =
 1
 3
 2
 4
 5
 7
 6
 8
```

The diag command

- `diag(v)` : makes a square matrix of zeroes with the specified vector in the main diagonal.

```
v1 = [33 66 99]; m1 = diag (v1)
 m1 = 33   0   0
       0  66   0
       0   0  99
```

- `diag(A)` : creates a vector equal to the main diagonal of the specified matrix.

```
m2 = [42 89 ; 78 31]; v2 = diag (m2)
 v2 = 42 31
```

More Built-In Functions for Arrays

- `mean(v)`: calculates the mean (average) of the specified vector.

```
v3 = [45 13 65 62]; average = mean (v3)
average = 46.2500
```

`sum(v)`: gives the sum (total) of all the elements of the specified vector.

```
v3 = [45 13 65 62]; total = sum (v3)
total = 185
```

`sort(v)`: sorts the elements in the specified vector in ascending order.

```
v3 = [45 13 65 62]; v4 = sort (v3)
v4 = 13  45  62  65
```

Creating Strings

• We create a string by typing characters within single quotes (').

• Many programming languages use the quotation mark (") for strings. Not MATLAB!

•Strings can contain letters, digits, symbols, spaces. Can be words, sentences, anything you want!

This is a string, not a number!

•Examples of strings: `'Boston'`, `'4522'`, `'{Canada1867!}'`, `'Life is good.'`

In a string variable

- Numbers are stored as an array
- A one-line string is a row vector
  - Number of elements in vector is number of characters in string

```
>> place = 'Niagara Falls';

>> size(place)

ans =

 1 13
```

String Indexes

Strings are indexed the same way as vectors and matrices

- Can read by index
- Can write by index
- Can delete by index

Ex:

```
>>word(1) = 'd';

>>word(2) = 'a';

>>word(3) = 'l';

>>word(4) = 'e';

>>word

word = dale
```

Example:

word variable is ‘dale’ from previous slide.

```
>> word(1)

ans = d

>> word(1) = 'v'

word = vale

>> word(end) = []

word = val

>> word(end+1:end+3) = 'ley'

word = valley
```

First letter is now **v**

Last letter is gone

ley added at the position following the end and the two other positions after that

MATLAB stores strings with multiple lines as an array. This means each line must have the same number of columns (characters).

```
>> names = ['Greg'; 'John']

names =

Greg

John

>> size( names )

ans =

2  4
```

Problem

4 characters 3 characters

```
>> names = [ 'Greg'; 'Jon' ]???
Error using ==> vertcat
```

vertcat means vertical concatenation

```
CAT arguments dimensions are not
consistent.
```

Must put in extra characters (usually spaces) by hand so that all rows have same number of characters

```
>> names = [ 'Greg'; 'Jon ' ]

Greg

Jon
```

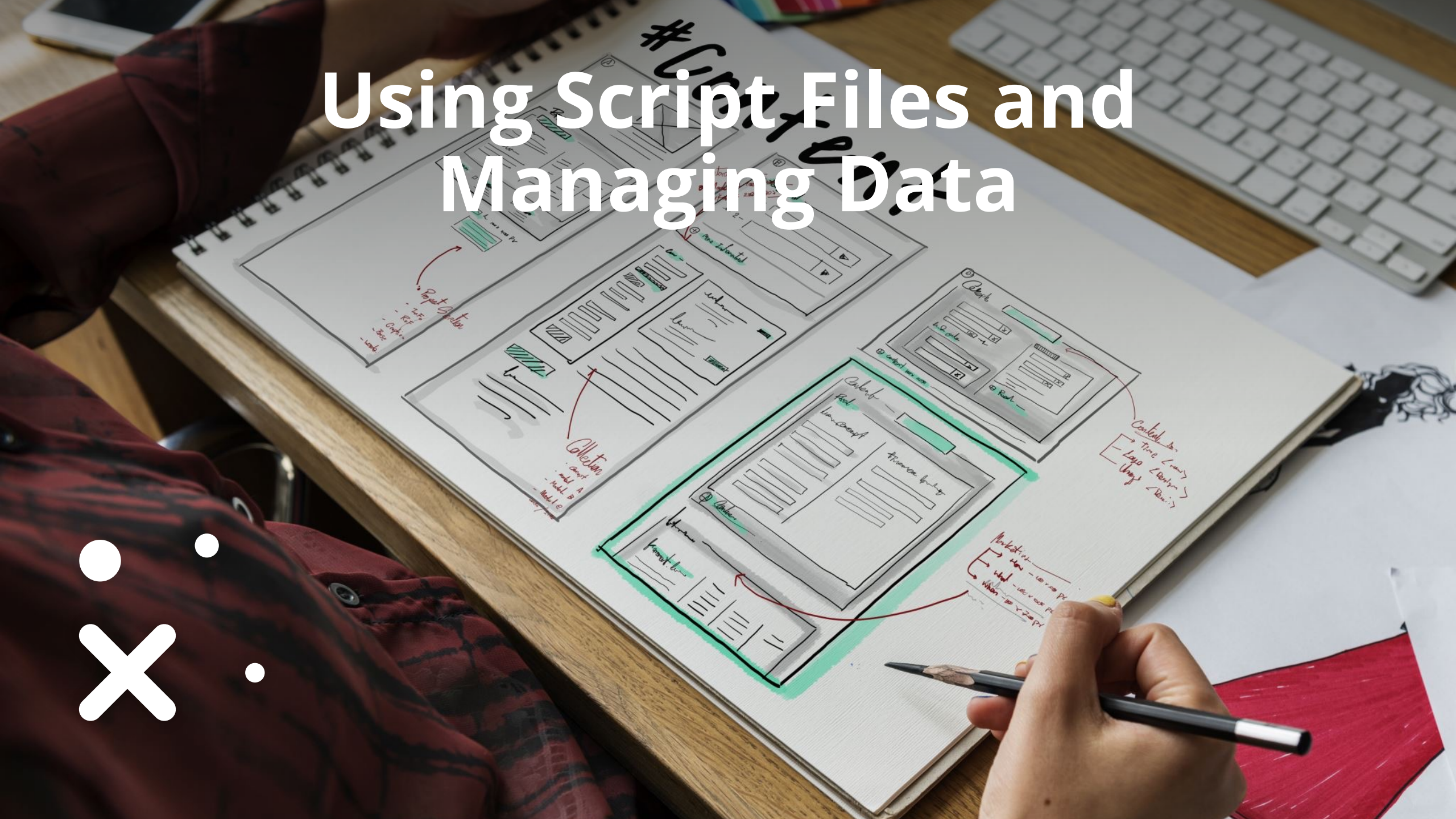
Extra space

String Padding

Making sure each line of text has the same number of characters is a big pain. MATLAB solves problem with **char** function, which *pads* each line on the right with enough spaces so that all lines have the same number of characters.

```
>> question = char('Romeo, Romeo,','Wherefore art thou', 'Romeo?')
question =

Romeo, Romeo,

Wherefore art thou

Romeo?

>> size (question)
ans =

3 18
```

Using Script Files and Managing Data



Using Variables in Simple Scripts

Method #1: You assign the values in the program itself (we saw that before remember? the = operator).

The following is an example of such a case. The script file (saved as Chapter4Example2) calculates the average points scored in three games.

```
% This script file calculates the average points scored in three games.
% The assignment of the values of the points is part of the script file.
game1=75;
game2=93;
game3=68;
ave_points=(game1+game2+game3)/3
```

The variables are assigned values within the script file.

The display in the Command Window when the script file is executed is:

```
>> Chapter4Example2

ave_points =
   78.6667
>>
```

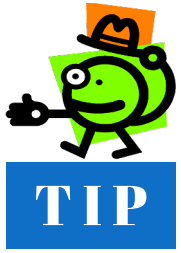
The script file is executed by typing the name of the file.

The variable `ave_points` with its value is displayed in the Command Window.

Using Variables in Simple Scripts

Method #2: Defining the variables in the command window before running the script.

- Advantage: More versatile as the script can produce a different result at every run depending on the variable values.



- *Instead of retyping entire command, use up-arrow to recall command and then edit it.*

For the previous example in which the script file has a program that calculates the average of points scored in three games, the script file (saved as Chapter4Example3) is:

```
% This script file calculates the average points scored in three games.
% The assignment of the values of the points to the variables
% game1, game2, and game3 is done in the Command Window.

ave_points=(game1+game2+game3)/3
```

Using Variables in Simple Scripts

Method #3: Assign by prompt in script file (meaning ask the user to fill the variables values).

The program asks the user to enter a value, then the script assigns that value to a variable. The MATLAB **input** command is used for that purpose.

```
variable_name = input('prompt')
```

prompt is the text that the input command will display in the Command Window (must be between single quotes).

Input Command - User Fills Variable

But what if you want to fill the variable with a string like a name or a sentence?

```
city = input ('City of birth: ')
```

If you enter just Toronto, you will get an error. You must enter the quotes to indicate that it is a string: 'Toronto'.

```
City of birth: 'Toronto'
```

But there is another way. You can use the 's' qualifier:

```
city = input ('City of birth: ', 's')
City of birth:  Toronto
```

The disp Command

- **disp (display)**: Displays variable values or text on the screen.
- Each display appears on a separate line. It doesn't show the variable name, only its value. **disp** only offers a limited control of display.
- For the best control of display we can use **fprintf**., a more advanced command.
- **disp(variable_name)** or **disp('text string')**

```
>> abc = [5  9  1;  7  2  4];
>> disp(abc)
     5     9     1
     7     2     4

>> disp('The problem has no solution.')

The problem has no solution.
>>
```

A 2 × 3 array is assigned to variable abc.

The disp command is used to display the abc array.

The array is displayed without its name.

The disp command is used to display a message.

The fprintf Command

- **fprintf** means file print formatted.
- You can write to the screen or to a file.
- You can mix numbers and text in the output.
- You have the full control of the output display but it is a little bit more complicated to use than **disp**.

The fprintf Command : A Simple Sentence

```
>> fprintf ('The quick brown fox')
The quick brown fox>>
```

Problem - The Command Window displays the command prompt (>>) at end of the text, not at start of next line! Not very pretty.

The solution is to use a `\n` *(new line) character at the end of the sentence. Looks much better!*

```
>> fprintf ('The quick brown fox\n')
The quick brown fox
>>
```

The \n Character

You can use as many \n in your fprintf as you like.

```
>> fprintf('Get\nBusy\nNow!\n\n')

Get
Busy
Now!

>>
```

fprintf Command – Printing a Variable

What if I want to print out the value of a variable? We can do that using a more advanced form of fprintf containing two parts: a string (including placeholders), and the arguments (the variable themselves). Here is a simple example with one numeric variable following a sentence.

```
>>year = 1867;
>>fprintf ('The year is %d\n', year)
The year is 1867
>>
```

This is a placeholder

You can also print without any extra text (aka labeling):

```
>>fprintf ('%d\n', year)
1867
>>
```

fprintf: Printing More Than One Variable

You can print more than one variable in the same fprintf, like this following example:

```
>> day = 25; temperature = 20;

>> fprintf ('It is October %d, and the temperature
  is %d degrees.\n', day, temperature)

It is October 25 and the temperature is 20 degrees.

>>
```

- *The number of variables and placeholders must be the same. The first placeholder formats the first variable, the second placeholder formats the second variable, and so on…*

Integer Placeholders

%d is the default integer placeholder. When used it will simply display the value ***as is*** without any padding. To add padding, to have nicely aligned columns for example, we need formatted placeholders.

%*n*d will reserve *n* places to display the number. Justification will be to the right. The negative sign takes one place.

If the value is 17 and %4d is used, then it will display 2 blank spaces followed by 17 on the screen:

##17

 # is used to represent a blank space but blank spaces are really invisible!

Integer Placeholders Rules

The number is always displayed in its entirety, even when the format is too narrow. With -1234 and a %3d placeholder, you would see -1234, therefore using 5 spaces instead of the 3 requested.

A negative number in the placeholder changes the justification to the left of the field. For example, with a value of -1234 and a %-8d placeholder, the display will be -1234###.

Three trailing blank spaces

Floating Point Placeholders

By default the %f placeholder displays the number with 6 decimal digits and no padding (may vary depending on computer system).

The formatted double placeholder has this format: *%w.df*, where **w** is the total width of the field (including sign and decimal point) and **d** the number of decimal digits.

If the value is 4.56 and the placeholder is %6.3f, then the display will be

Floating Point Placeholders Rules

With a floating point formatted print, **you always get the requested number of digits to the right of the decimal point** (even if the field is not wide enough).

You also **always** (like the integer placeholder) **get all the significant numbers** (the numbers to the left of the decimal point).

However, if there are more decimal precision in the value than are requested in the placeholder, the value is truncated to match the placeholder size and rounded up if need be.

Floating Point Placeholder Examples

If the value is -32.4573 and the placeholder is %7.3f, then you will get 3 decimal digits as requested plus the significant numbers: -32.457 for a total of 7 spaces.

If the value is -3.4578 and the placeholder is %8.3f, then you will get 3 decimal digits as requested plus the significant numbers and padding: ##-3.458. Note the rounding-up effect.

Two leading blank spaces

If the value is 187.123 and the placeholder is %8.7f it will display 187.1230000.

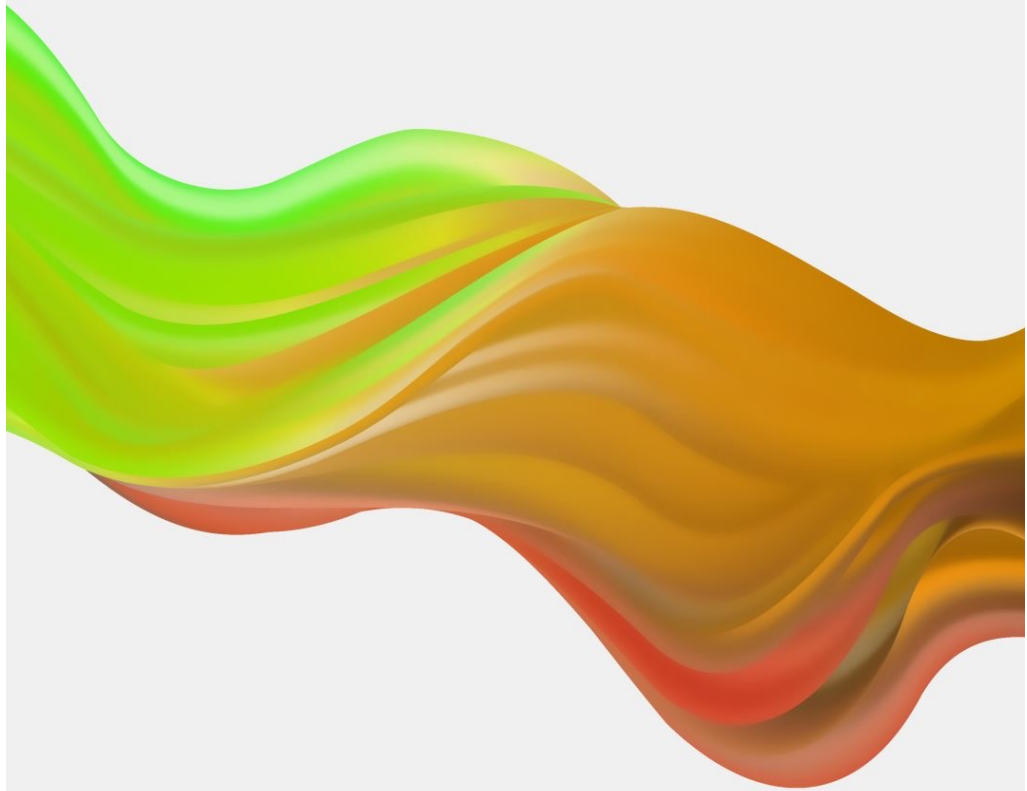
*Note: Internal values in the computer's memory are **unaffected** by placeholders (just look at the workspace).*


NEWS!

About our lecture

Computer Programming

MENG2020



The %e placeholder : Scientific Notation

%e works like %f
Here are a few examples with printing the value of pi:

```
>> fprintf( 'pi is about %f\n', pi )
   pi is about 3.141593

>> fprintf( 'pi is about %10.8f\n', pi )
 pi is about 3.14159265

>> fprintf( 'pi is about %10.8e\n', pi )
 pi is about 3.14159265e+00

>> fprintf( 'pi is about %10.2e\n', pi )
 pi is about  3.14e+00
```

Two leading blank spaces here

Escape Characters in fprintf

- What if you want to use reserved characters in an fprintf? Use escape characters to display characters used in placeholders.

- To display a percent sign, use `%%` in the text.

```
fprintf('100%%\n')
```

→ `100%`

- To display a single quote, use `''` in the text (two sequential single quotes).

```
fprintf('Martha''s Vineyard')
```

→ `Martha's Vineyard`

- To display a backslash, use `\\` in the text (two sequential backslashes).

```
fprintf('Nantucket\\ACK')
```

→ `Nantucket\ACK`

Long fprintf Command

Format strings are often long. You can break a string by:

1. Put an open square bracket ([ ) in front of the first single quote.
2. Put a second single quote where you want to stop the line.
3. Follow that quote with an ellipsis (three dots).
4. Press ENTER, which moves cursor to next line.
5. Type in the remaining text in single quotes.
6. Put a close square bracket ( ]).
7. Put the rest of the fprintf command.

Long fprintf Command : Example

```
>> weight = 178.3;
>> age = 17;
>> fprintf( ['Tim weighing %.1f lbs'...
' is %d years old'], weight, age )

 Tim weighing 178.3 lbs is 17 years old
```

fprintf and Files : Step 1

The **fprintf** command can also be used to save the output to a file instead of a display in the command window.

It takes three steps to write to a file. The first step is to open the file:

```
fid = fopen('file_name','permission')
```

- The **fid** variable is the *file identifier* to let fprintf know what file to write its output in. You may use another variable name.
- The **permission** code tells how file will be used: for reading (r), writing (w), or appending(a).

Ex:

```
>> fid = fopen('output.txt', 'w');
```

fprintf and Files : Permission Codes

Some common permission codes:

r : open file for reading only.

w : open file for writing. If the file already exists, the content is deleted. If the file doesn't exist, a new file is created.

a : same as w except that if the file already exists the written data is appended to the end of the file.

If no permission code specified, fopen uses r.

*Use command **help fopen** for more advanced codes.*

fprintf and Files : Step 2

Write to the file with fprintf. Use it exactly as before but insert fid before the format string (with placeholders):

```
fprintf(fid,'format string',variables)
```

The passed fid is how fprintf knows to write to the file instead of display on the screen.

Ex: `>> fprintf (fid, 'Age = %d\n', age);`

fprintf and Files : Step 3

- When you're done writing to the file, close it with

```
>> fclose(fid);
```

- Once you close it, you can't use that fid anymore until you get a new one by calling fopen.

- It is good form to close your files when you no longer need them. Especially important for files open in writing ('w' or 'a').

About our lecture

More on Data Files

- If the file name you give to fopen has no path, MATLAB writes it to the current directory, also called the working directory.
- You can have multiple files open simultaneously and use fprintf to write to all of them just by using different variable names as *fids*.
- Files are written in raw text. You can read the files you make with fprintf in any text editor like MATLAB's Editor/Debugger window or Notepad.

Reading from files with fscanf

To read data from a text file, we use the fscanf command (need to do fopen first of course). Example script:

```
a = 20;
b = 30;
c = 40;
out = fopen ('data.txt', 'w');
fprintf (out, '%d %d %d', a, b, c);
fclose (out);

in = fopen ('data.txt', 'r');
v = fscanf (in, '%d')
fclose(in);
```

File created and filled

OUTPUT

```
v =
    20
    30
    40
```

Exporting Data: Sharing with Other Programs

Remember that we used **save** and **load** to keep our MATLAB variables (workspace) from one session to the other? We can use a special save to export our variables to programs other than MATLAB:

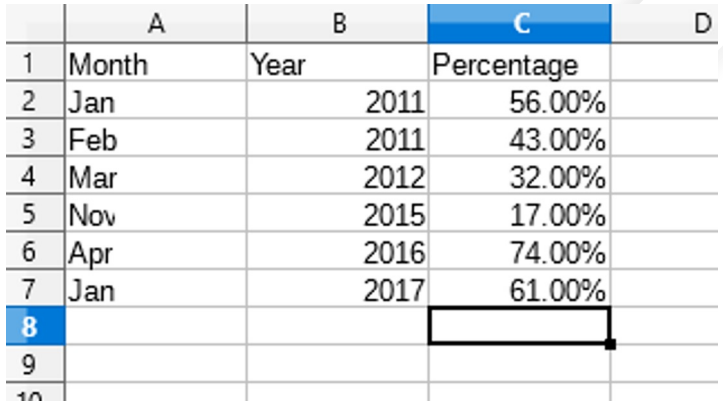
To save as formatted text (also called *ASCII text*). For example,

```
>> save filename.txt -ascii;
```

Note: It saves only the variable values, not their names!

Importing Data from a Spreadsheet

1. Let's have a spreadsheet with data created with Excel or OpenOffice or LibreOffice.



	A	B	C	D
1	Month	Year	Percentage	
2	Jan	2011	56.00%	
3	Feb	2011	43.00%	
4	Mar	2012	32.00%	
5	Nov	2015	17.00%	
6	Apr	2016	74.00%	
7	Jan	2017	61.00%	
8				
9				

2. Now export or save as your spreadsheet as an XLS file or a CSV text. Use the comma as your delimiter for CSV. The CSV file will look like this:

```
Month,Year,Percentage
Jan,2011,56.00%
Feb,2011,43.00%
Mar,2012,32.00%
Nov,2015,17.00%
Apr,2016,74.00%
Jan,2017,61.00%
```

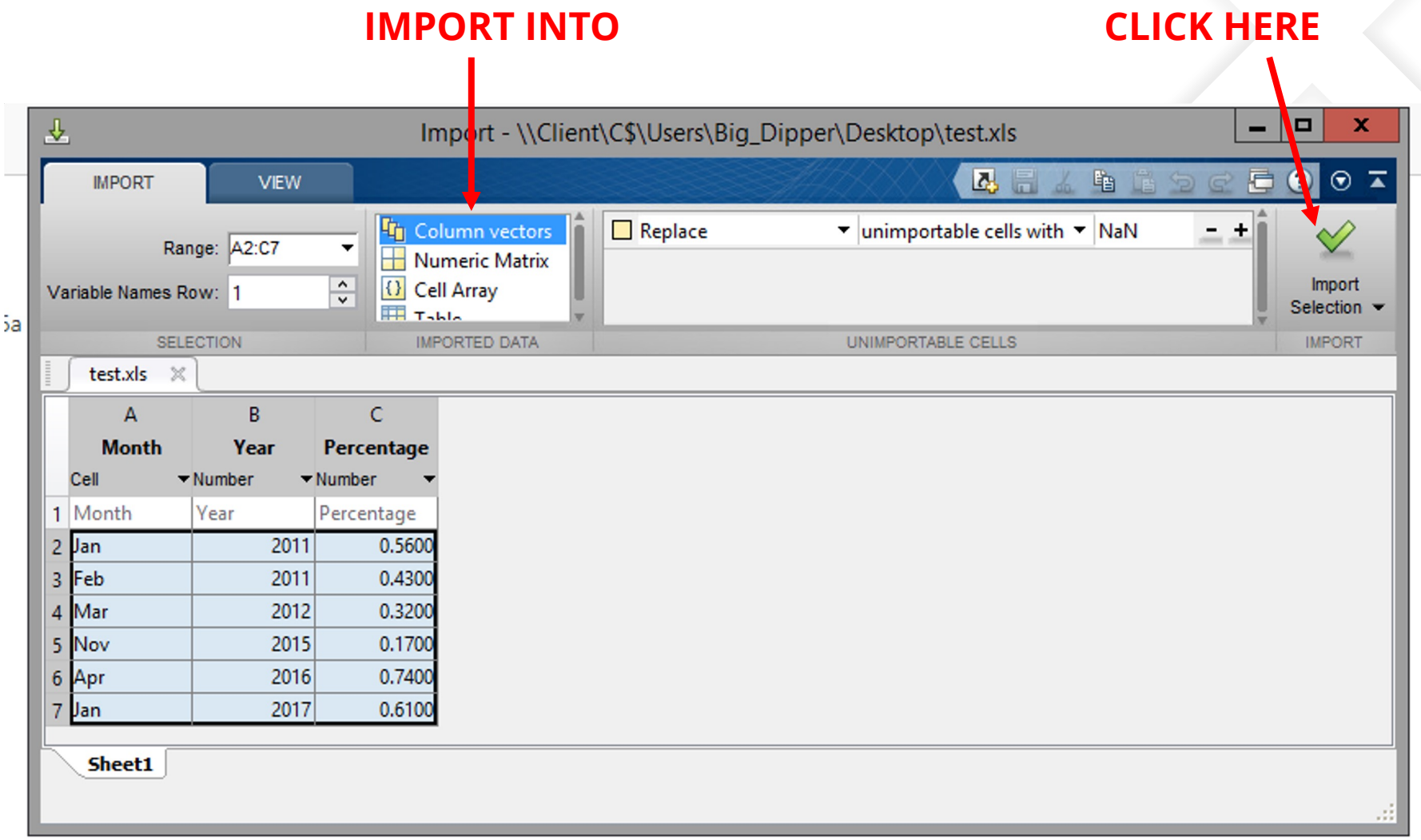
Importing Data from a Spreadsheet

3. There are two ways to import data files. The easiest way is to use the import wizard. You can import easily the XLS or CSV files. Use the Import Data button to start the wizard.

4. Locate the file you wish to import.

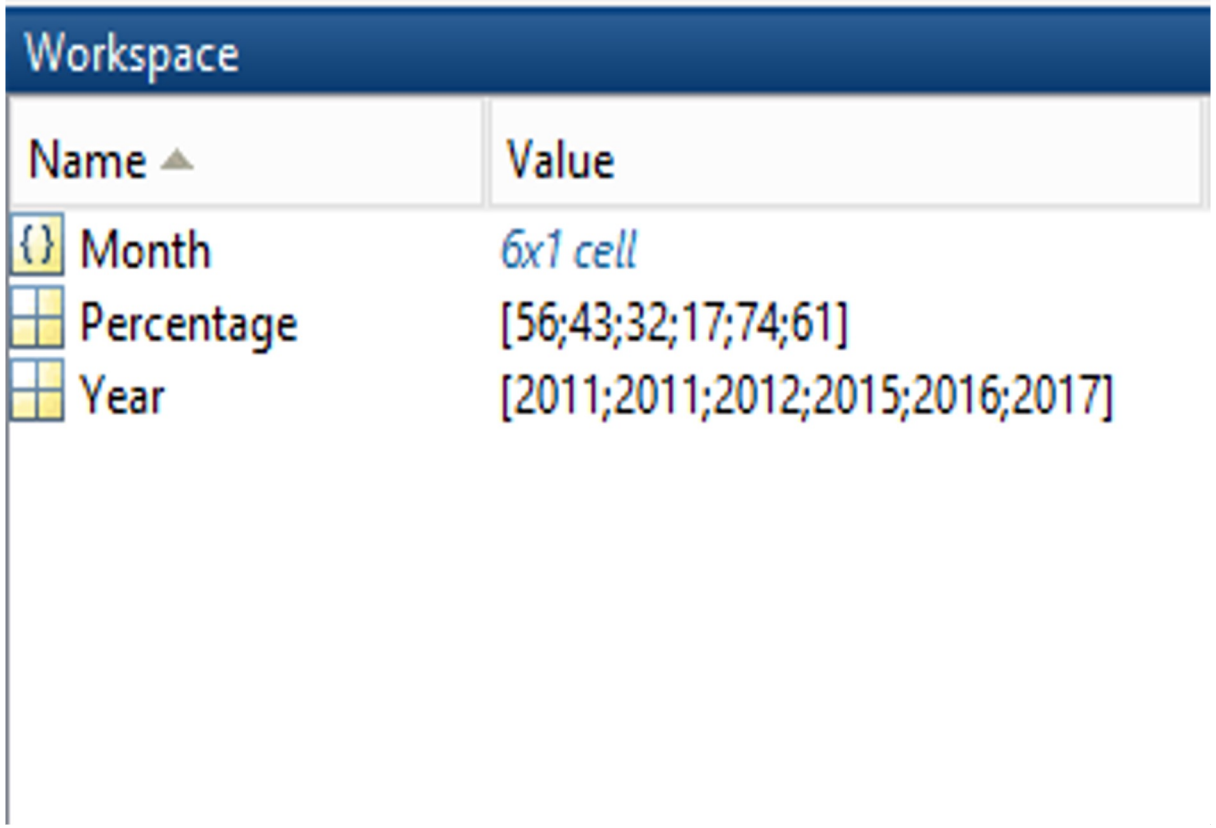
5. Next select the kind of structure you wish to put your imported date into (column vectors, numeric matrix,… and click the green check mark to import.

Importing Data from a Spreadsheet



Importing Data from a Spreadsheet

6. Close the import window and note that the workspace contains the imported variables. In this example data was imported as column vectors.



Importing Data from a Spreadsheet

- You can also import or export data by using commands in a script:
- Import from an XLS file:

```
variable_name = xlsread('filename.xls')
```

- Import from a CSV file:

```
variable_name = csvread('filename.csv')
```

- Export from MATLAB to XLS:

```
xlswrite('filename.xls', variable_name)
```

- Export from MATLAB to CSV:

```
xlswrite('filename.csv', variable_name
```

Working with Live Scripts

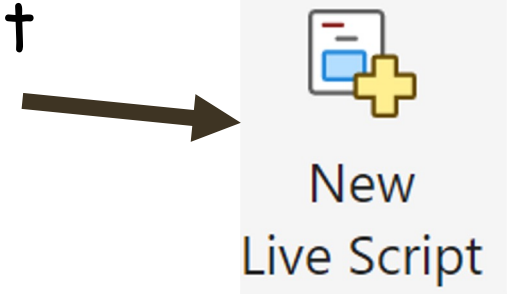
- The latest versions of MATLAB now offer a new dynamic way to write code: Live Scripts.
- A Live Script lets you work with your code dynamically and see the result alongside the code.This approach provides a less abstract method of creating MATLAB scripts.
- Live Scripts are created with the Live Editor, a different editor than the Editor window seen before.
- Live Scripts use exactly the same MATLAB commands seen before.

Live Scripts vs. Regular Scripts

- Live Scripts are saved with the .mlx extension instead of the .m extension for regular scripts.
- A Live Script provides way to document the code as a report as it provides text markup options. A separate Word or PDF report is no longer required.
- A Live Script provides results in real time. No need to wait to run the script.
- Live Scripts can be divided into sections, each one can be executed separately.

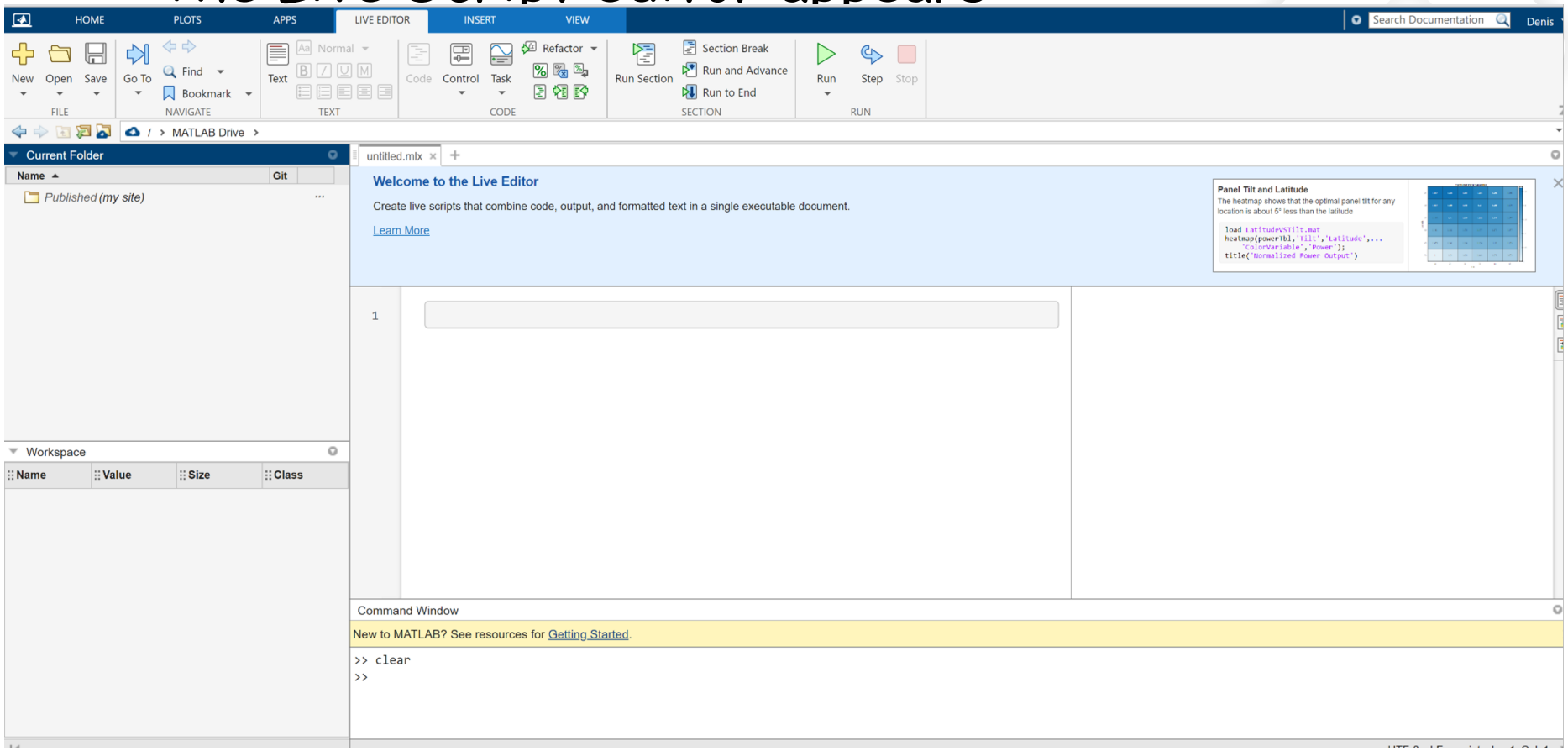
Opening the Live Editor

- Unlike regular scripts saved in .m files, live scripts saved in .mlx files are in xml format instead of raw text.
- mlx files have three types of content in them:
  - Code
  - Formatted content
  - Output
- To open



Opening the Live Editor

- The Live Script editor appears:



Welcome to the Live Editor

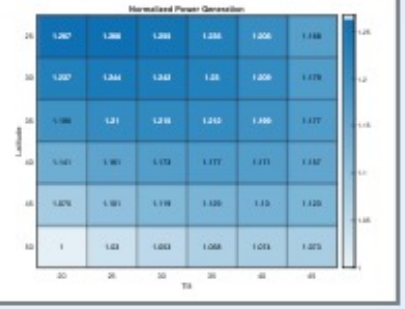
Create live scripts that combine code, output, and formatted text in a single executable document.

Learn More

Panel Tilt and Latitude

The heatmap shows that the optimal panel tilt for any location is about 5° less than the latitude

```
load    LatitudeVSTilt.mat
heatmap(powerTbl,    'Tilt'    , 'Latitude'    , ...
        'ColorVariable'    , 'Power'    );
title(    'Normalized Power Output'    )
```



```
1  3*4
2  sin (pi)
```

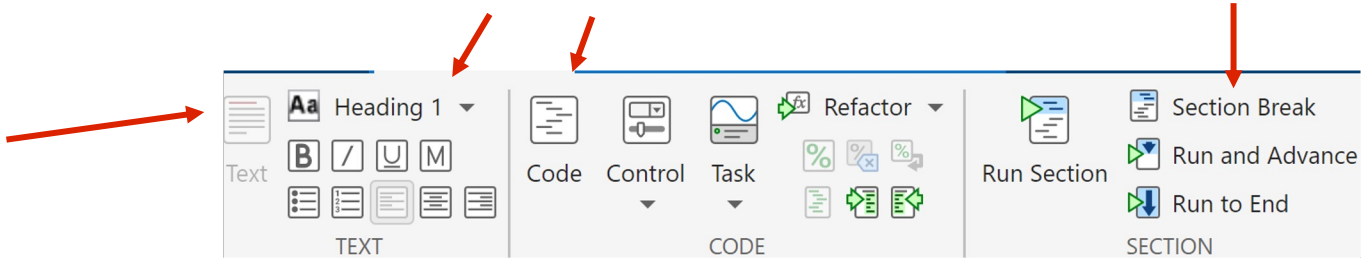
```
ans = 12
ans = 1.2246e-16
```

Working with the Output Pane

- Click **Text** in the Text section of the live editor tab. The current line of the editor changes to a text line (the line number disappears.)
- Type *This is a sample section* and then select **Heading 1** in the style choices of the Text section. Press Enter.
- Click **Code** in the Code section and enter the following code:

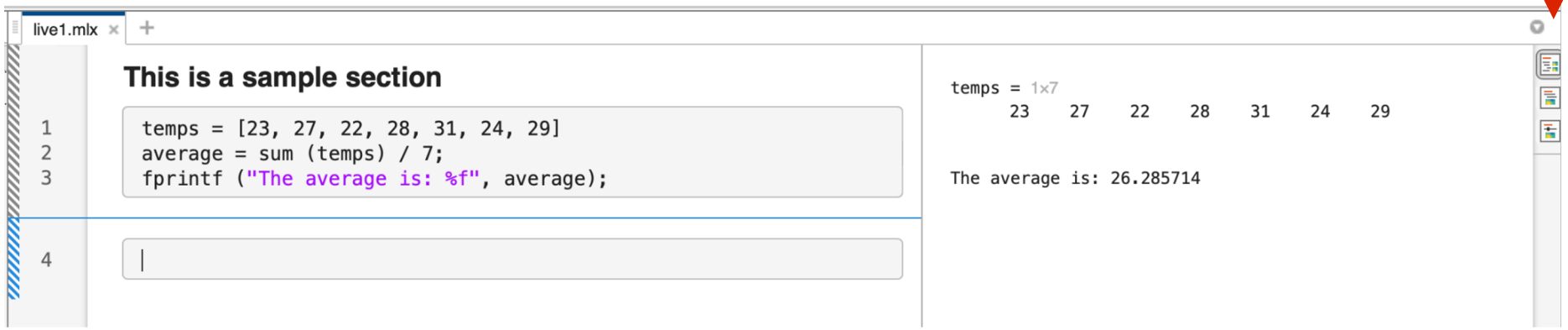
```
temps = [23, 27, 22, 28, 31, 24, 29]
average = sum (temps) / 7;
fprintf ("The average is: %f", average);
```

- Press Enter and click **Section Break** in the Section section. The editor adds a blue line.



Saving and Running a Section

- The result is shown with the code and formatted text on the left and the output of the right. You can change the display with the three buttons on the right side of the output pane.
- Save your script with the mlx extension (for example live1.mlx).
- Click inside the section code and click Run Section in the Section area.



This is a sample

```
1
2
3
4
```

```
temps = [23, 27, 22, 28, 31, 24, 29]
average = sum (temps) / 7;
fprintf ("The average is: %f", average);

```

```
temps = 1×7
      23     27     22 ...
```

```
The average is: 26.285714
```

This is a sample

```
1
2
3
4
```

```
temps = [23, 27, 22, 28, 31, 24, 29]
average = sum (temps) / 7;
fprintf ("The average is: %f", average);

```

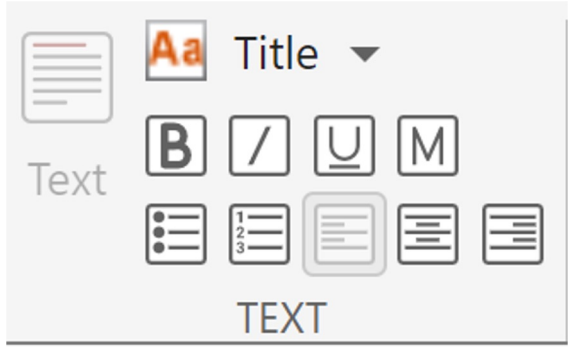
temps = 1×7

	1	2	3	4
1	23	27	22	

```
The average is: 26.285714
```

Adding Formatted Text

- You are not limited to the Heading 1 format. You have many choices of text formats and you can have as many text areas as you need.
- Each text format can be further formatted as bold, italics, underline and monospace.
- You can also create bulleted and ordered lists as well as changing the alignment.
- All this is controlled from the Text section.



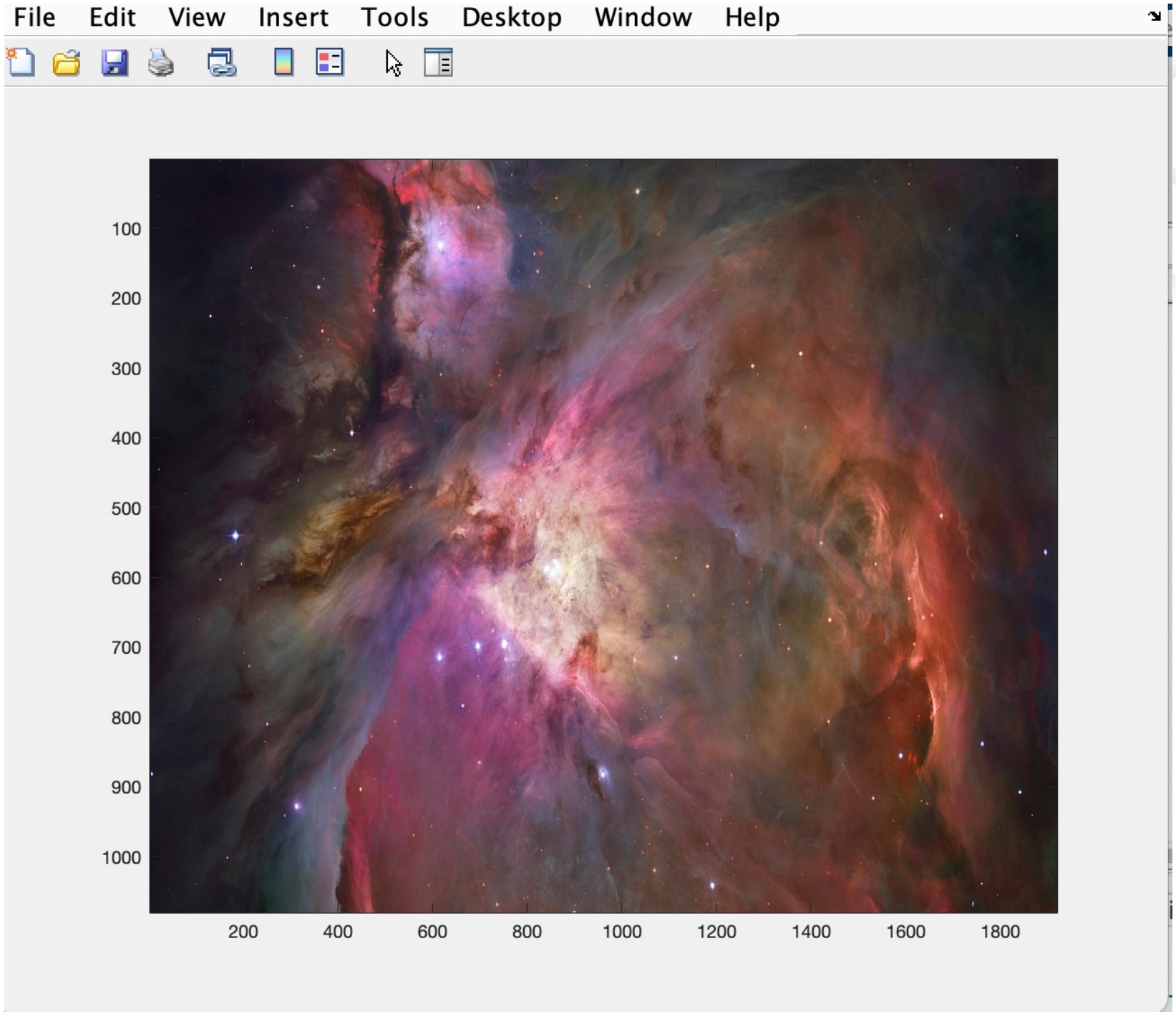
Incorporating Graphics

- It is possible to add graphics (images) into a Live Script report.
- The very common formats gif, jpeg and png are supported. Some other less popular formats like bmp, pcx or tiff are also available.
- Adding a graphic into MATLAB is a two-step process:
  - Read the graphic into a variable with the **imread** command.
  - Display the content of the variable on screen with the **image** command.

Incorporating Graphics

- This example will add a picture of the planet in spacetaken by the Hubble Space Telescope.
- Place a text with the format Title: Space.
- Place the image and add a second section break after.

```
space=imread('https://ihypress.com/wallpapers/hubble/img15.jpg ');
image (space);
```


File
Edit
View
Insert
Tools
Desktop
Window
Help
100
200
300
400
500
600
700
800
900
1000
200
400
600
800
1000
1200
1400
1600
1800

This concludes our overview of MATLAB and a taste of things to come!

Plotting Vectors

- Let's create a vector v1.

```
>> x = [0:0.04*pi:10];
>> y = sin(x);
```

- You can plot the value against the index (1,2,3...).

```
>> plot (y);
```

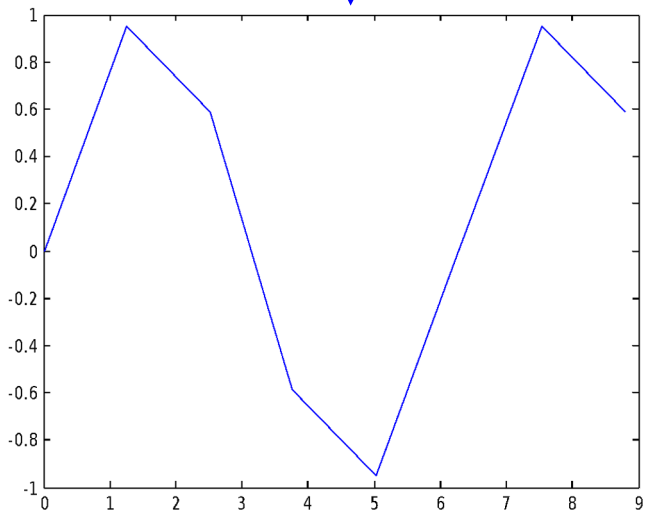
- Or you can plot against another variable (the most common usage).

```
>> plot (x,y);
```

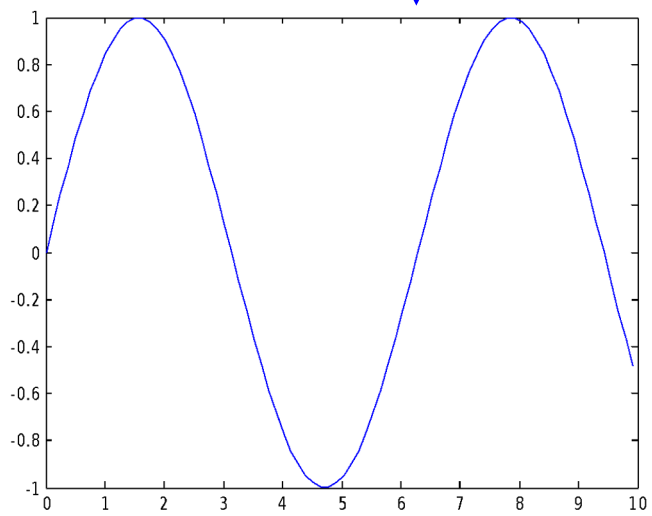
The plot Command

- **plot** generates dots at each (x,y) pair and then connects the dots with a line. To make the plot look smoother, just use more points.

```
>> x = [0:0.4*pi:10];
```

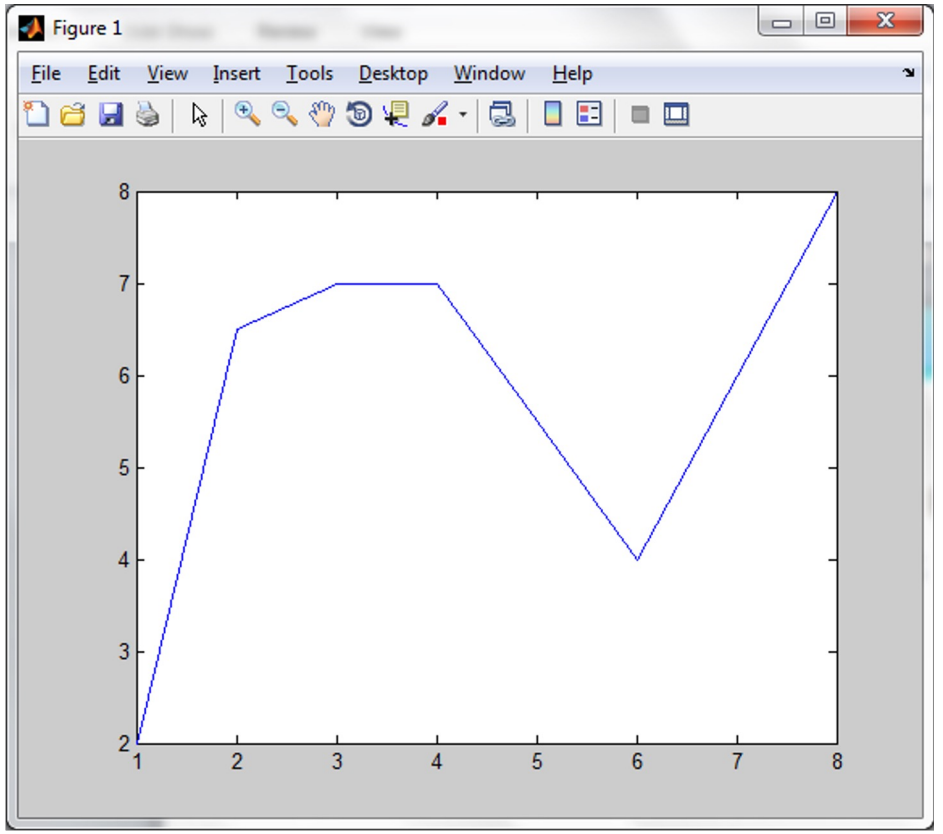



```
>> x = [0:0.04*pi:10];
```



The *plot* Command : Another Example

```
>> y=[ 2 6.5 7 7 5.5 4 6 8];
>> plot(y)
```



The plot Command

- x and y vectors must be same size or else you'll get an error.

```
>> plot ([1 2], [1 2 3]);
```

- You can change the line color, marker style, and line style by adding a string argument.
  Ex: `plot (x,y,'k.-');`

- You can plot without connecting the dots by omitting the line style argument.
  Ex: `plot (x,y,'.');`

Plot Line Specifiers

Line specifiers define the style and color of lines, and marker types.

STYLES

Line Style	Specifier
solid (default)	-
dashed	--

Line Style	Specifier
dotted	:
dash-dot	-.

DEFAULTS

COLOURS

Line Color	Specifier
red	r
green	g
blue	b
cyan	c

Line Color	Specifier
magenta	m
yellow	y
black	k
white	w

Plot Marker Types

Marker Type	Specifier		Marker Type	Specifier
plus sign	+		square	s
circle	o		diamond	d
asterisk	*		five-pointed star	p
point	.		six-pointed star	h
cross	x		triangle (pointed left)	<
triangle (pointed up)	^		triangle (pointed right)	>
triangle (pointed down)	v			

Plot Command Syntax

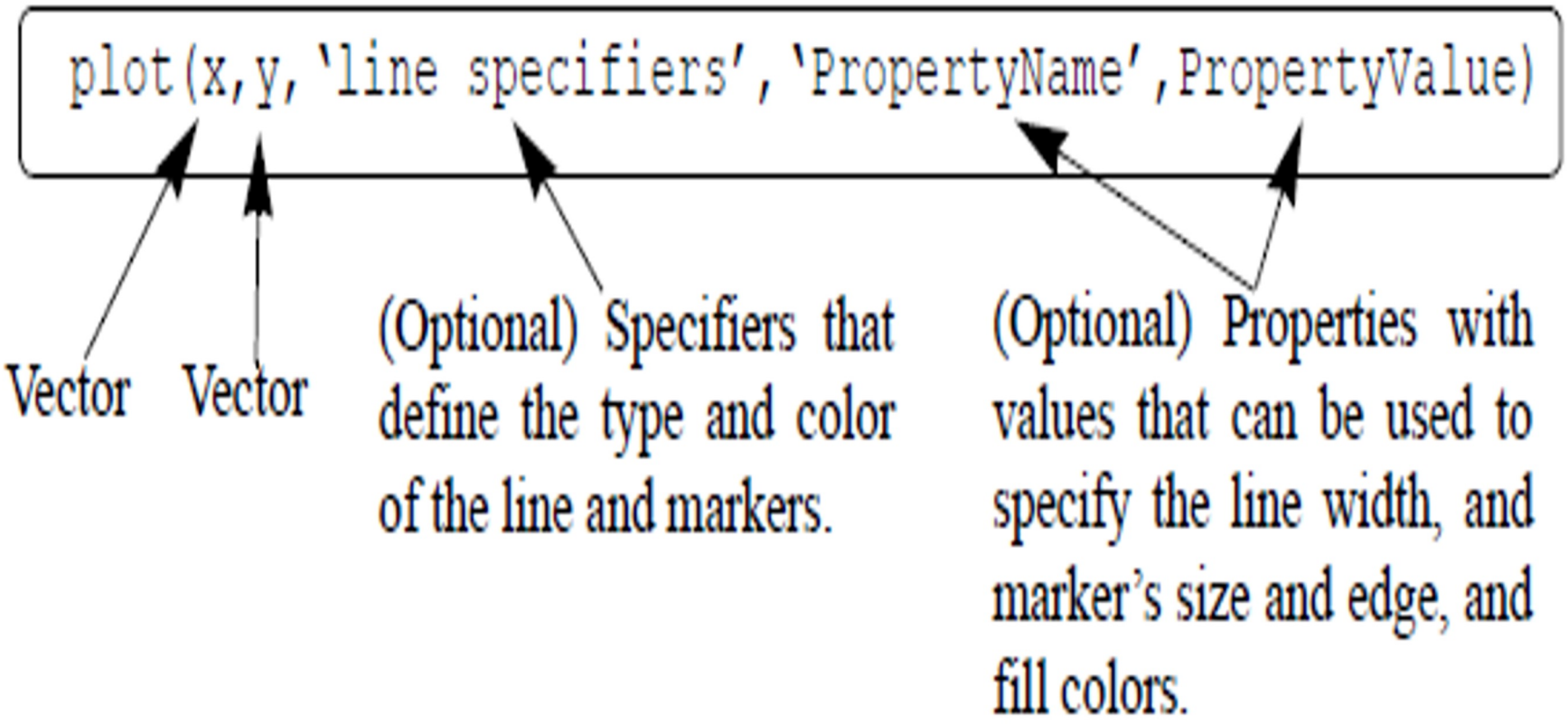
Notes about using the specifiers:

- The specifiers are typed inside the `plot` command as strings.
- Within the string the specifiers can be typed in any order.
- The specifiers are optional. This means that none, one, two, or all the three can be included in a command.

Some examples:

`plot(x,y)`	A blue solid line connects the points with no markers (default).
`plot(x,y,'r')`	A red solid line connects the points.
`plot(x,y,'--y')`	A yellow dashed line connects the points.
`plot(x,y,'*')`	The points are marked with * (no line between the points).
`plot(x,y,'g:d')`	A green dotted line connects the points that are marked with diamond markers.

The *plot* Command Generic Syntax



Plot Property Names and Values

In the *plot* command, type the property name in quotes marks, then a comma, then a value. Repeat for multiple properties.

Property Name	Description	Possible Property Values
`LineWidth` (or `linewidth`)	Specifies the width of the line.	A number in units of points (default 0.5).
`MarkerSize` (or `markersize`)	Specifies the size of the marker.	A number in units of points.
`MarkerEdgeColor` (or `markeredgecolor`)	Specifies the color of the marker, or the color of the edge line for filled markers.	Color specifiers from the table above, typed as a string.
`MarkerFaceColor` (or `markerfacecolor`)	Specifies the color of the filling for filled markers.	Color specifiers from the table above, typed as a string.

For example, the command:

```
plot(x,y,'-mo','LineWidth',2,'markersize',12,
      'MarkerEdgeColor','g','markerfacecolor','y')
```

creates a plot that connects the points with a magenta solid line and circles as markers at the points. The line width is two points and the size of the circle markers is 12 points. The markers have a green edge line and yellow filling.

EXAMPLE

YEAR	1988	1989	1990	1991	1992	1993	1994
SALES (millions)	8	12	20	22	18	24	27

```
>> yr=[1988:1:1994];
>> sle=[8  12  20  22  18  24  27];
>> plot(yr,sle,'--r*','linewidth',2,'markersize',12)
>>
```

Line Specifiers: dashed red line and asterisk marker.

Property Name and Property Value: the line width is 2 points and the markers size is 12 point.

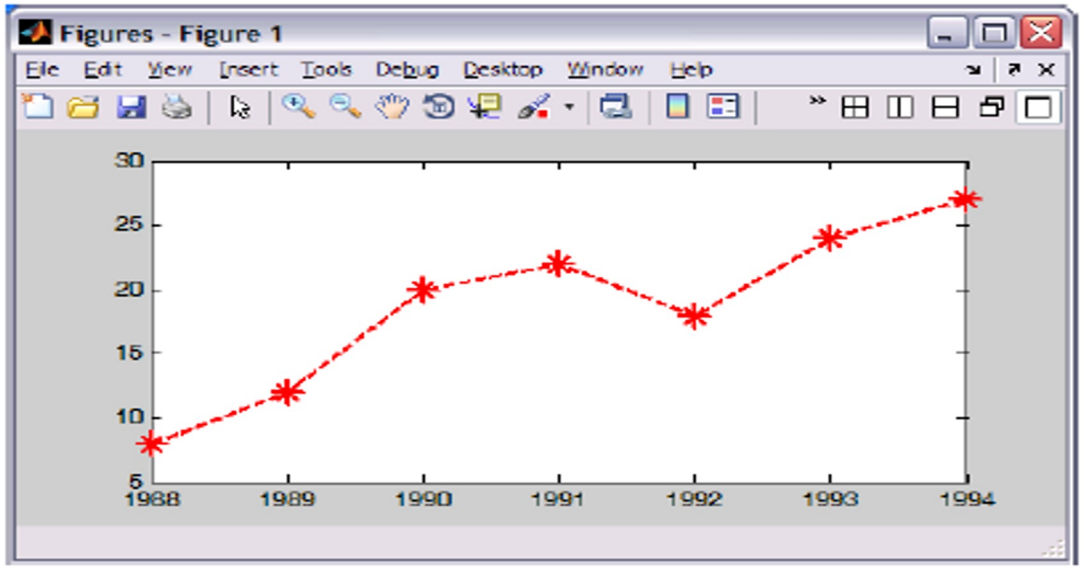


Figure 5-3: The Figure Window with a plot of the sales data.

The plot Command Hints

- Look at *help plot* for a full list of colours, markers, and line styles.
- You can customize your plot by using the menus in the figure window.
- Figures can be saved in many formats like bmp, gif, png, jpg or pdf.

Axes Options for Plots

You can turn on the grid to make it easier to read values.

```
>> grid on;
```

You can also turn off the grid.

```
>> grid off;
```

xlim sets the x axis limits.

```
>> xlim ([-pi pi]);
```

ylim sets the y axis limits.

```
>> ylim ([-1 1]);
```

Other Axes Options for Plots

```
>> axis square;
```
Axes look like a box without changing the scale.

```
>> axis tight;
```
Fits axes to data with no wasted space (default).

```
>> axis equal;
```
Makes the x and y axes with the same scale.

```
>> axis xy;
```
Places the origin (0,0) at the bottom-left corner (default).

```
>> axis ij;
```
Places the origin (0,0) at the top-left corner (good for viewing matrices).

Multiple Plots on Multiple Figures

Use the **figure** command to open a new figure or to activate an already opened figure.

```
>> figure;
%creates a new blank figure and
activates it.

>> figure(1);
%activates figure 1 or creates it
if it doesn't exist.
```

Closing Figure Windows

Use the **close** command to close figure windows.

`close`: Closes the active figure window.

`close(n)`: Closes the figure window **n**.

`close all`: Closes all the opened figure windows.

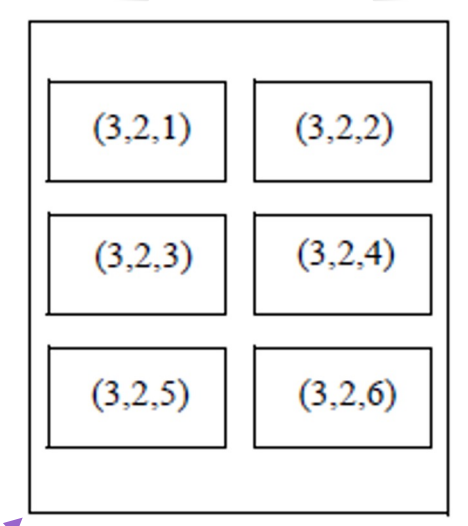
Multiple Plots on a Figure

You can have multiple plots in the same figure. For example,

```
>> subplot(3,2,1);
```

makes a figure with 3 rows and 2 columns of axes, and activates the first subplot.

*The subplots are numbered from left to right and top to bottom, with the upper left being subplot 1 and the lower right subplot m*n.*



Subplot numbers for 3x2 set of subplots

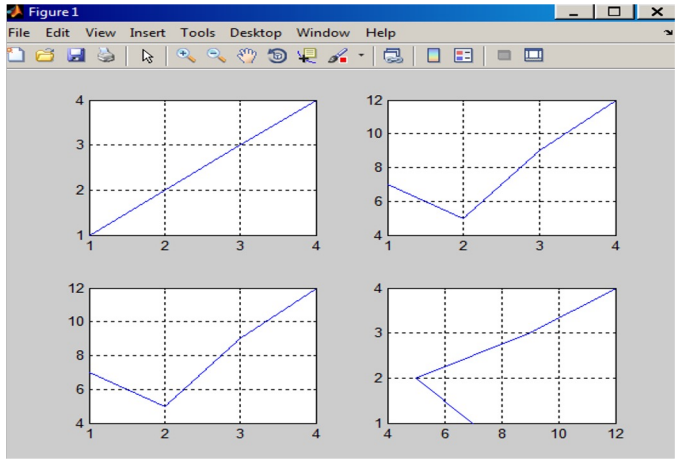
Multiple Plots on a Figure

```
x = [1 2 3 4]
y = [7 5 9 12]
subplot (2,2,1)
plot (x)
grid
subplot (2,2,2)
plot (y)
grid
subplot (2,2,3)
plot (x,y)
grid
subplot (2,2,4)
plot (y,x)
grid
```

subplot(m,n,p)

*If subplots don't exist, subplot creates them and makes subplot **p** the current subplot.*

*If subplots already exist, subplot makes subplot **p** the current one.*



Plotting a Function Against a Vector

Here is a simple way to plot a function of an independent variable:

- Create a vector *x* of values for the independent variable.
- Create a vector *y* of values as a function for every element of the independent variable vector
- Plot using **plot(x,y)**. See example next slide.

EXAMPLE

```
%  A script file that creates a plot of
%  the function: 3.5.^(-0.5*x).*cos(6x)
x=[-2:0.01:4];
y=3.5.^(-0.5*x).*cos(6*x);

plot(x,y)
```

Create vector x with the domain of the function.

Create vector y with the function value at each x.

Plot y as a function of x.

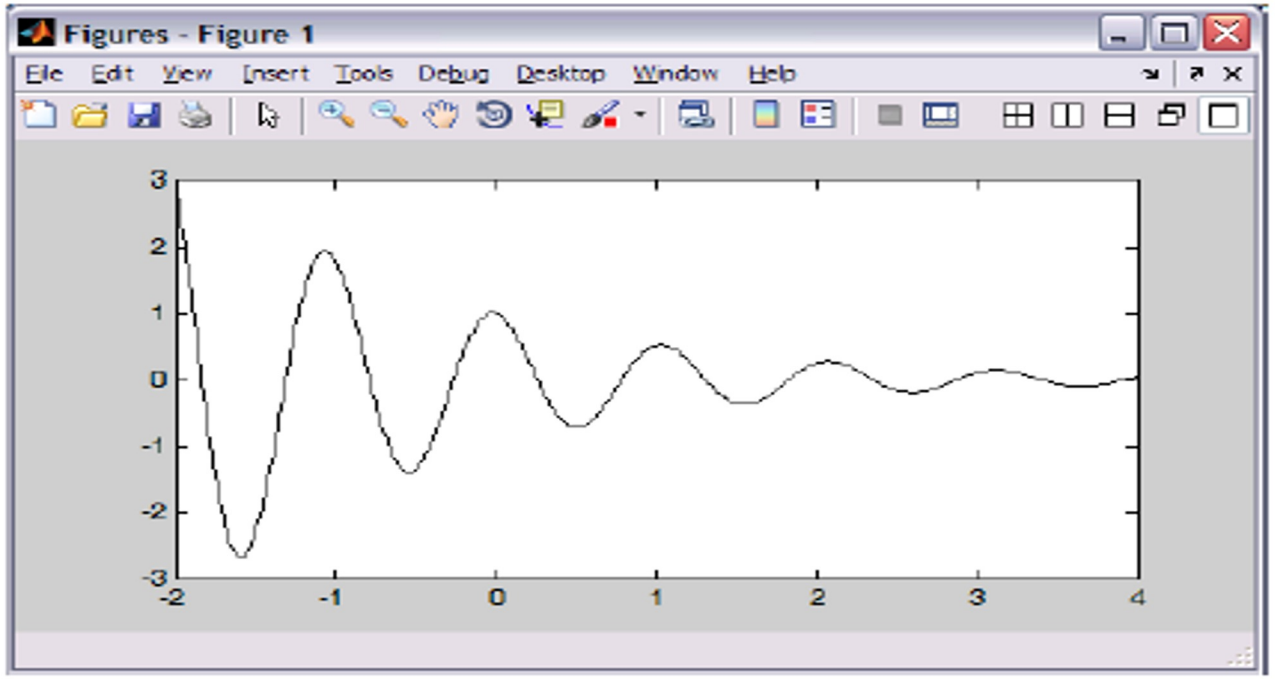


Figure 5-4: The Figure Window with a plot of the function: $y = 3.5^{-0.5x}\cos(6x)$.

Copying the Figure Window to Another Program

. To copy the entire figure window into another program (Word, PowerPoint,…), use screenshots:

. Use *Windows+Shift+s* (Windows) or *use Shift+Command+4 (Mac)*. The figure window you selected is now in the Clipboard.

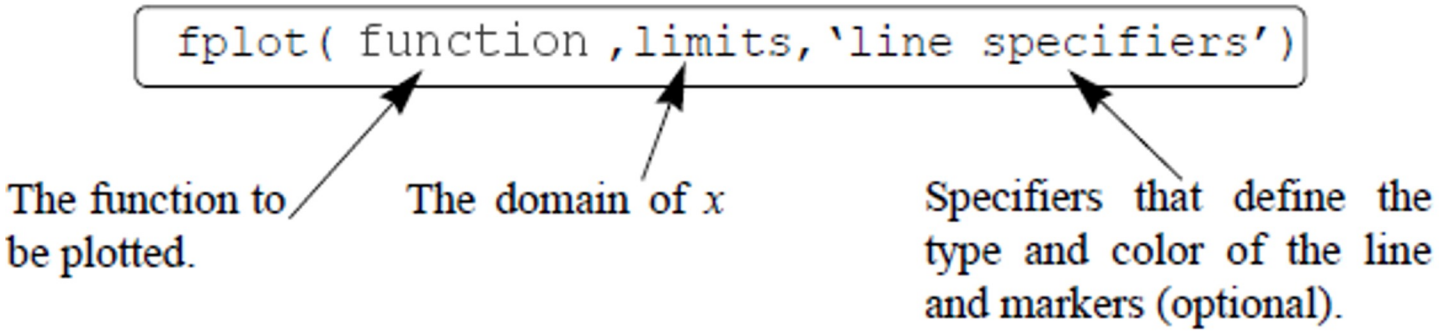
. Paste the image into the other application.

Copying a Figure to Another Program

- To copy just the plot area of the figure window into another program:
- Select *Edit/Copy Figure* in the figure window menu.
- Paste the image into the other application.
- Of course, you can save the figure to an image file (png, gif, jpg,...) as we have seen before.

Plotting a Function with *fplot*

The **fplot** command plots the curve defined by a function like $y = f(x)$ between given limits on the x axis.



The function is like a formula and is specified like this:

```
formula = @(x) x.^2 + 4.*sin(2.*x) - 1
```

The function variable

Always @(independent variable)

The formula containing the independent variable

Plotting a Function with *fplot*: *Example*

```
>> fplot(formula,[-3,3],'k')
```

function — limits of x — line specifiers (same as **plot**)

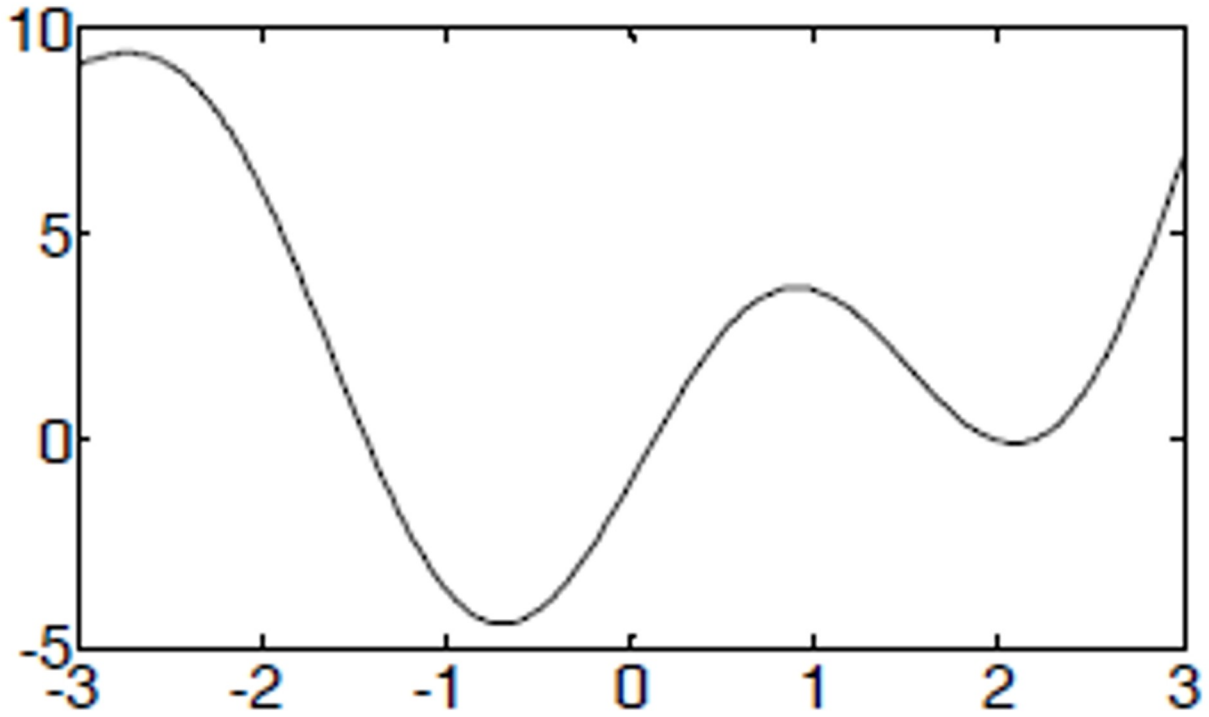


Figure 5-6: A plot of the function $y = x^2 + 4\sin(2x) - 1$.

Putting Multiple Plots on the Same Figure (This is different than subplots!)

Method #1: Multiple Pairs of Vectors

Here is an example for 3 plots on the same figure:

```
>> plot(x,y,u,v,t,h)
```

It plots **y** vs. **x**, **v** vs. **u**, and **h** vs. **t**.

The vectors of ***each*** pair must be same size b*ut it can be different than sizes in other pairs.*

It is recommended to use line specifiers to distinguish between the plots:

```
>> plot(x,y,'-b', u,v,'--r', t,h,'g:')
```

METHOD 1

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'-b',x,yd,'--r',x,ydd,':k')
```

Create vector x with the domain of the function.

Create vector y with the function value at each x.

Create vector yd with values of the first derivative.

Create vector ydd with values of the second derivative.

Create three graphs, y vs. x, yd vs. x, and ydd vs. x in the same figure.

The plot that is created is shown in Figure 5-7.

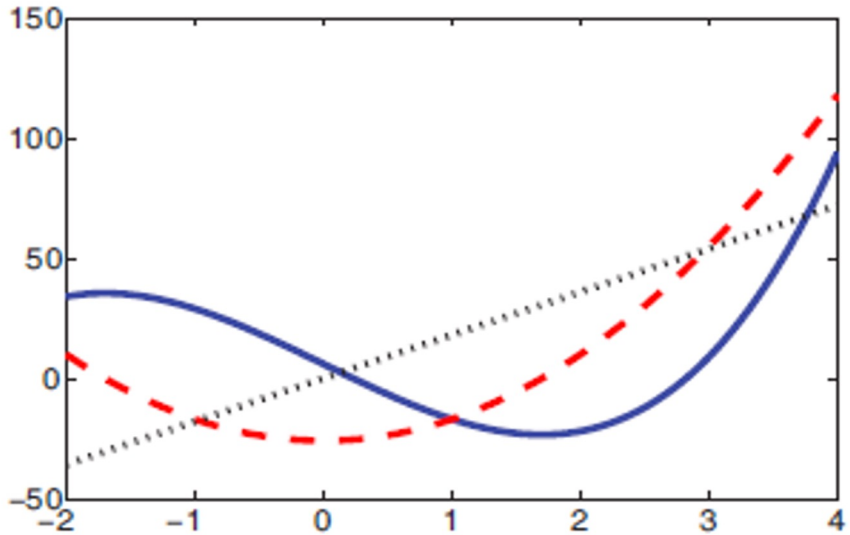


Figure 5-7: A plot of the function $y = 3x^3 - 26x + 10$ and its first and second derivatives.

Putting Multiple Plots on the Same Figure

Method #2: Use the *hold on* Command

Normally, each time you execute *plot* or *fplot* it erases the previous plot and draws a new one. ***hold on*** changes that.

1. Issue the command **hold on**.
2. Call **plot** or **fplot** for each of the remaining graphs.
3. Issue the command **hold off**.
4. Graphs drawn after **hold on** are added to the plot.
   Graphs drawn after **hold off** erase the plot.
5. Draw the first graph with **plot** or **fplot**.

METHOD 2

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'-b')
hold on
plot(x,yd,'--r')
plot(x,ydd,':k')
hold off
```

The first graph is created.

Two more graphs are added to the figure.

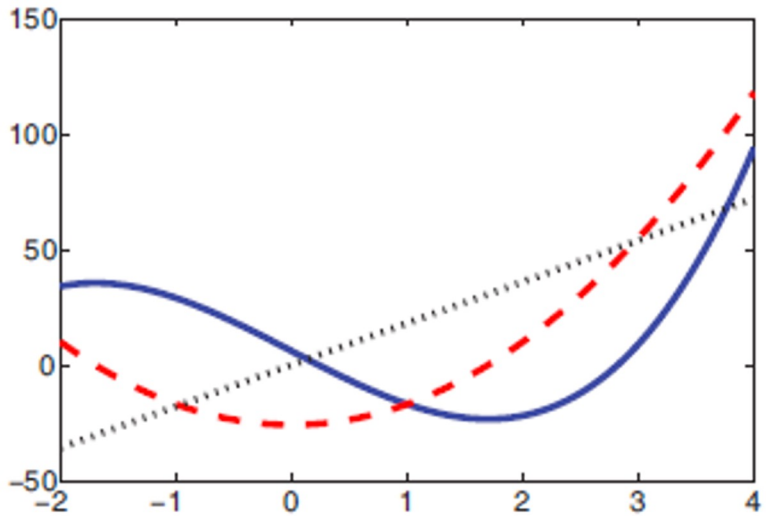


Figure 5-7: A plot of the function $y = 3x^3 - 26x + 10$ and its first and second derivatives.

Multiple Plots with the *line* Command

Method 3: Using the *line* command

The line command is another method still to add additional graphs to an existing plot.

```
line(x,y,'PropertyName','PropertyValue')
```

Example:

```
>> line(x,y,'linestyle','--','color','r','marker','o')
```

Adds a graph drawn with a dashed red line and circular markers to the current plot.

METHOD 3

```
x=[-2:0.01:4];
y=3*x.^3-26*x+6;
yd=9*x.^2-26;
ydd=18*x;
plot(x,y,'LineStyle','-','color','b')
line(x,yd,'LineStyle','--','color','r')
line(x,ydd,'linestyle',':','color','k')
```

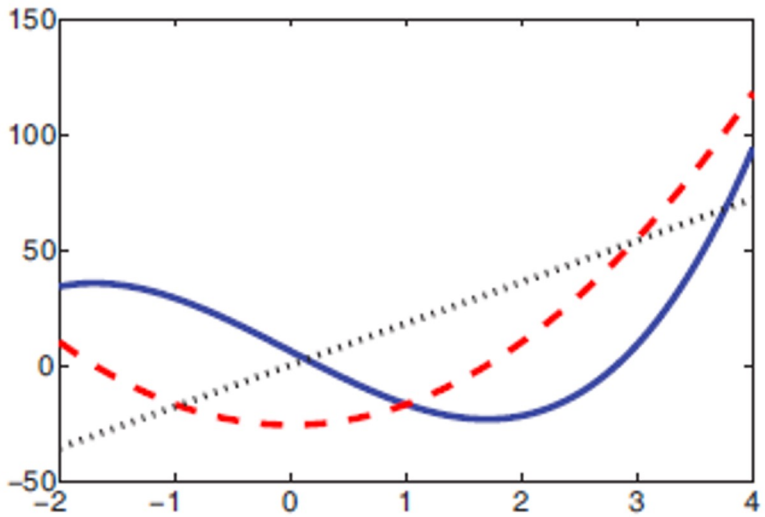


Figure 5-7: A plot of the function $y = 3x^3 - 26x + 10$ and its first and second derivatives.

This concludes our overview of MATLAB and a taste of things to come!

End of lesson