Matlab Tutorial

January 23, 2019

Instructor: Prof. Michael Rahaim

Outline

- Introduction to Matlab
- Data Types and Conversions
- Data Visualization
- Matlab Scripts
- Structures, Functions, and Classes
- Additional Material
  - *Data Import / Export*
  - *Matrices & Matrix Manipulation*
  - *Matlab Reports*
  - *Graphical User Interfaces*
  - *Matlab Compiler / Matlab Live / Matlab Profiler*

Note: *Thorough understanding of the basic usage will make it much easier to work on more elaborate problems!*

We will cover how to do a variety of things…

BUT, the real objective is to learn how to figure things out on your own.

This lecture is highly interactive

- Follow along with activities
- Try out what you see on the slides
- Ask questions!

Introduction

Discussion Topics:

- *Matlab Overview*
- *Development Environment*
- *Matlab Help and Documentation*

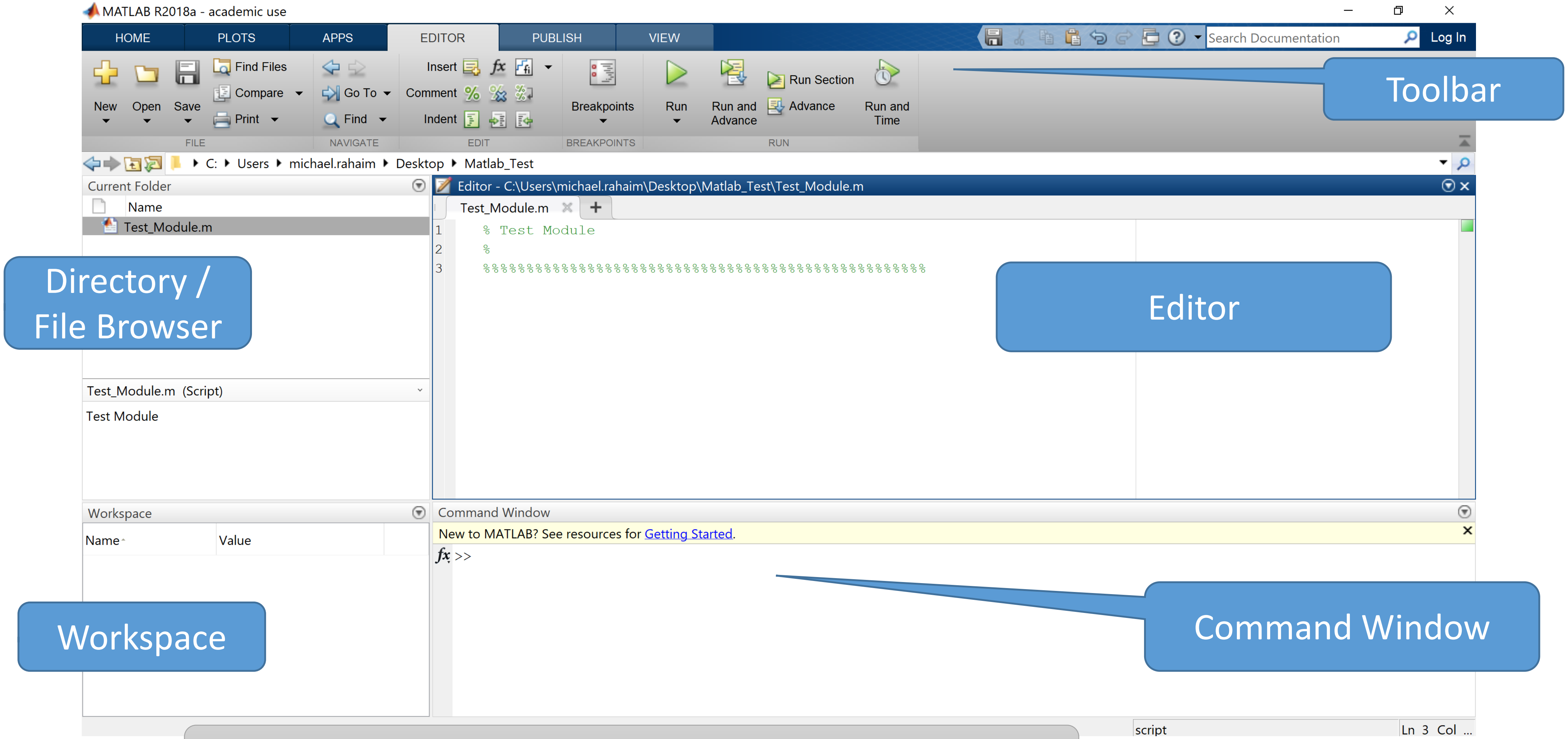
Matlab Overview

- What is Matlab?
  - Why do you care?
  - Why are you here???

- Benefits of Matlab
  - Scripting Language (not compiled like C, C++, etc.)
  - Highly optimized for Matrix manipulation
  - Highly integrated with many aspects of academic research
  - Large community of users and contributors to Matlab Central
  - Excellent documentation and “help” resources

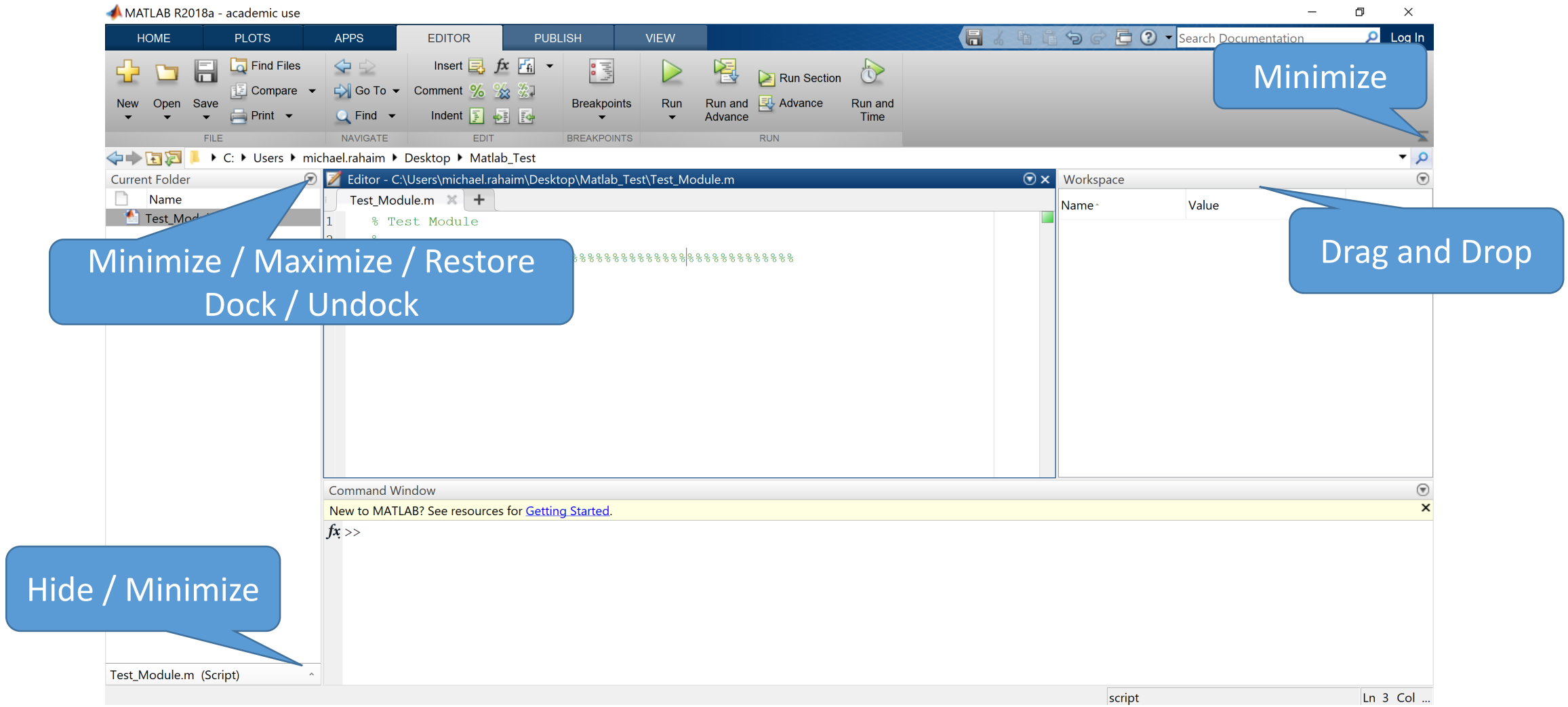
Development Environment

Note: *Toolbar layout often changes between Matlab versions*



Appearance may differ in different operating systems

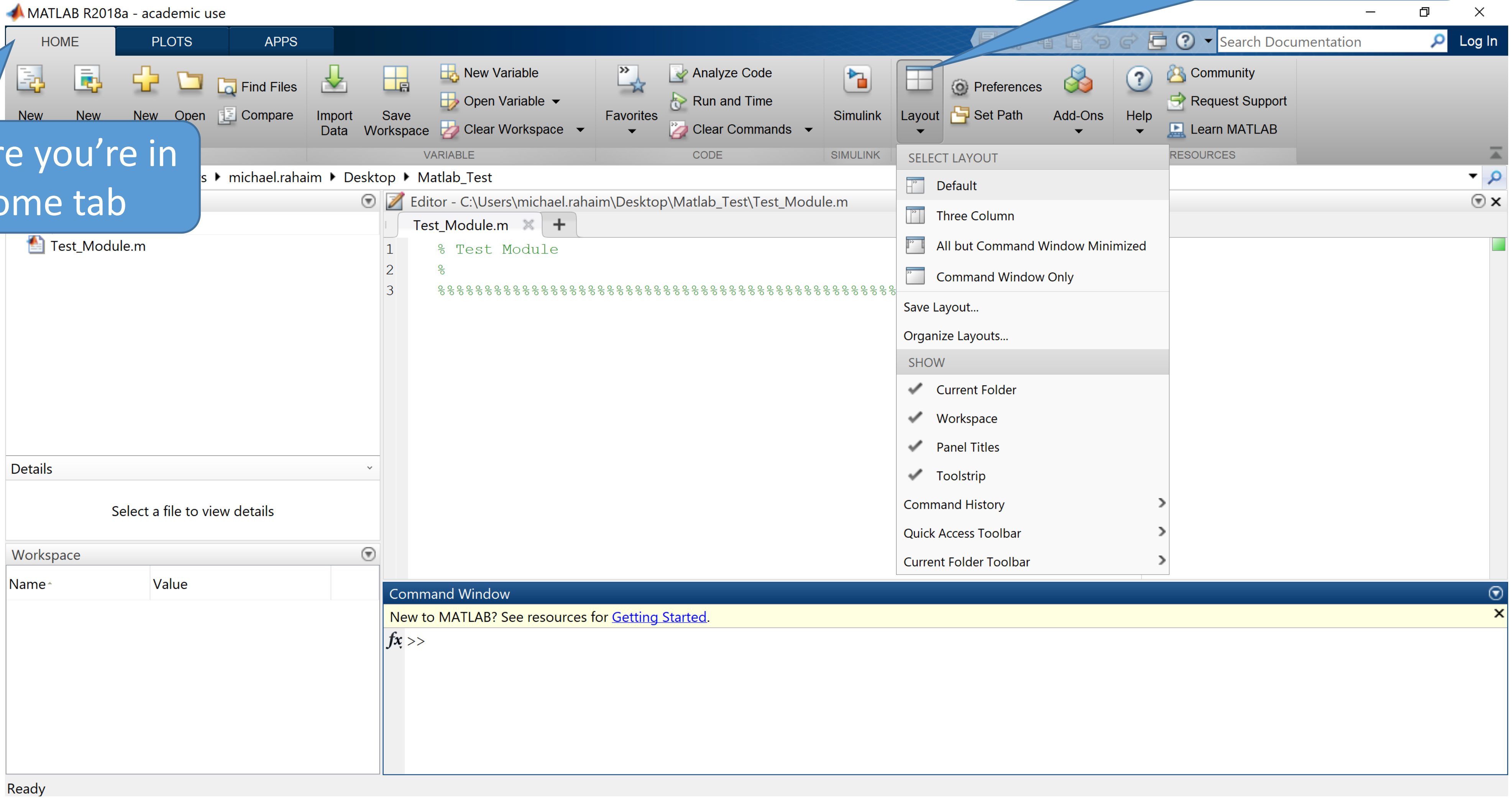
Organizing the Environment



Organizing the Environment

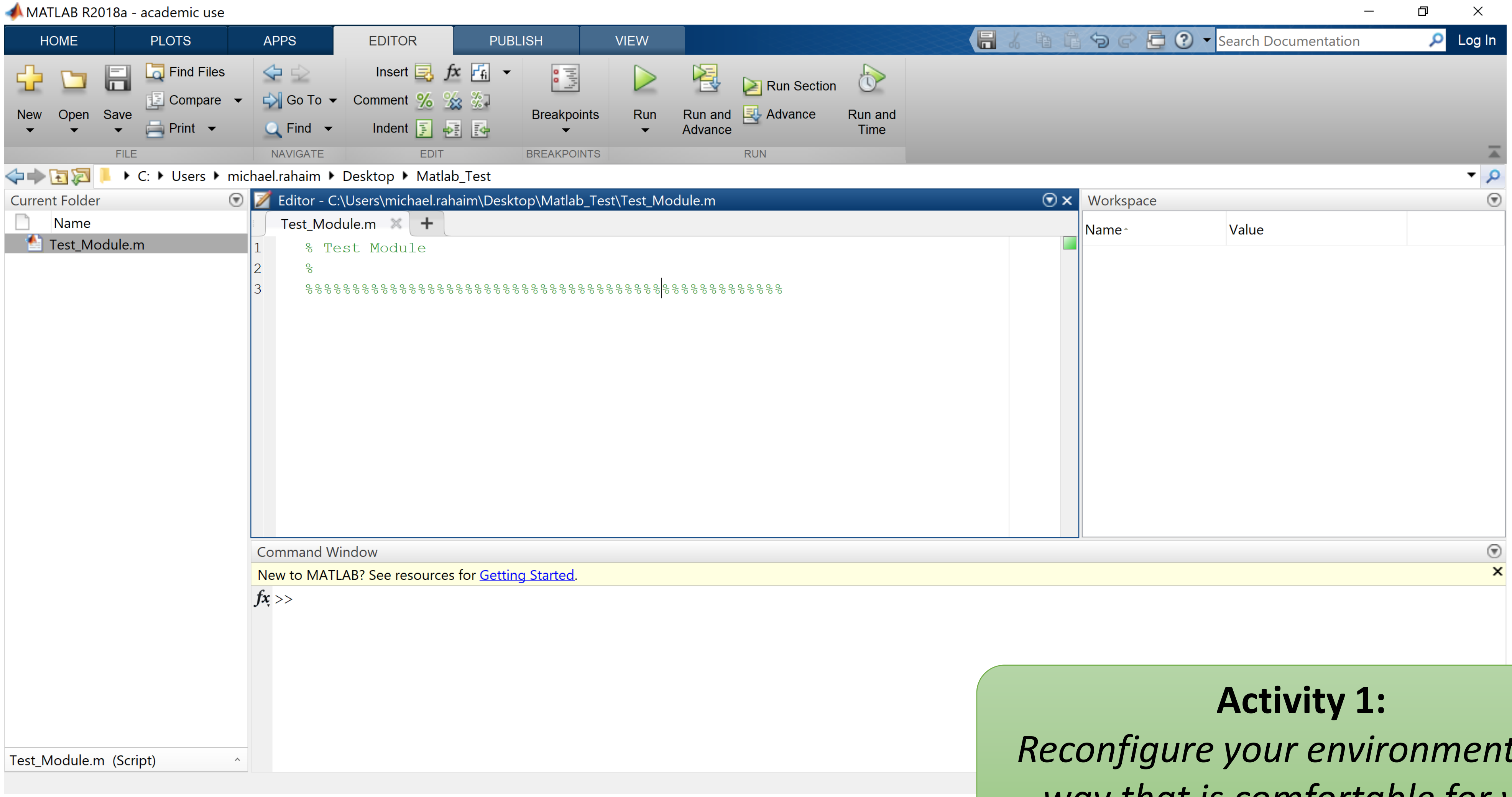
You can always return to default configuration

Make sure you're in the home tab



Organizing the Environment

I will be using this configuration for most of the lecture.



Activity 1:
Reconfigure your environment in a way that is comfortable for you

Using the Command Line

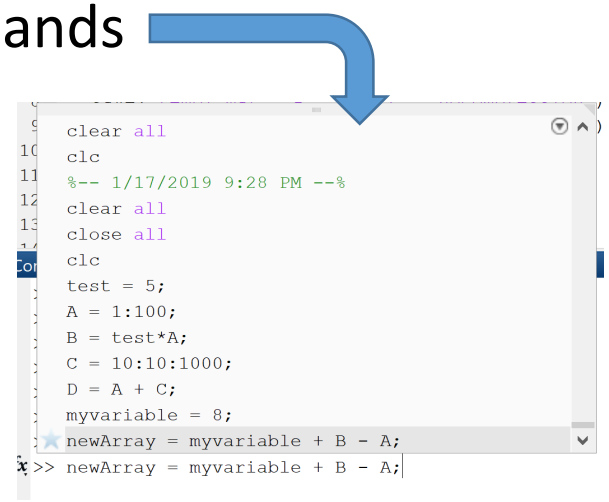
We will discuss debugging later; but the command line is useful for testing commands as you step through a program or script

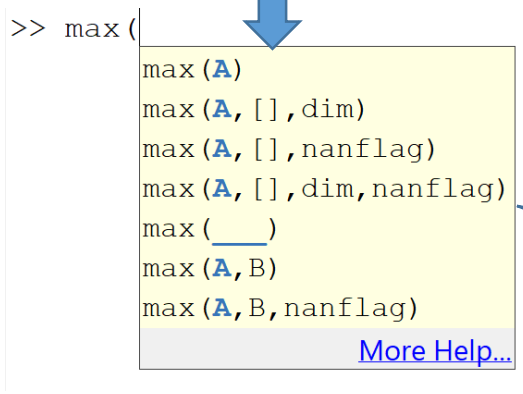
- Command line is a simple command / response interface
- Type a command, get a response:

```
>> a = 5
>> b = a+2
```

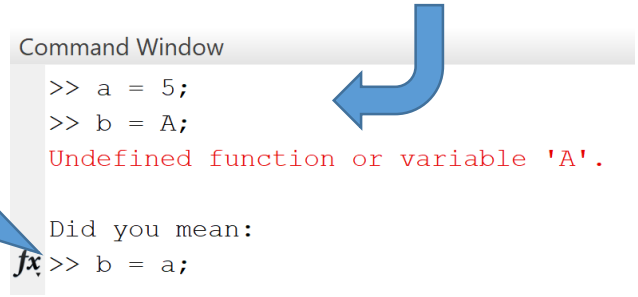
Response might be an action (e.g, storing variable to workspace)

- Using the Command line:
  - Use Arrows to navigate, find, edit, and reuse previous commands
  - Use Tab completion for longer variable names and functions
  - Suggested input to functions
  - Get warnings and errors





(Pause for a second after typing a parenthesis)



Matlab is case sensitive!

Using Matlab's Workspace

Useful Commands

```
clear all
clc
```

- **Minimalist View:** "Matlab is a fancy calculator"

Ok… so let's make some variables and do some simple math manipulations!

- Define Variables (Example): `a = 5;` versus `a = 5`

Activity 2:

1. *Use the layout options to minimize everything except for the command window*
2. *Define the variables a, b, c, and d*
3. *Use the variables to define x and y*
4. *Determine z*

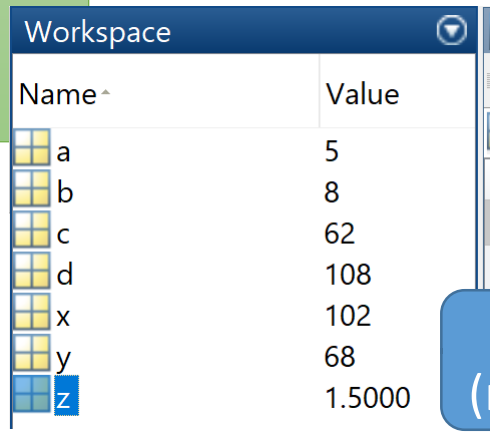
$$a = 5,\ b = 8,\ c = 62,\ d = 108$$

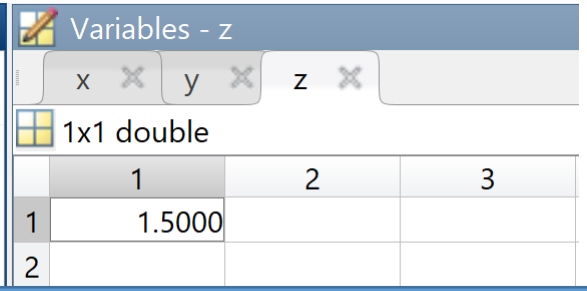
$$x = c + ab,\ y = d - ab$$

$$z = x/y$$

Now, look at the workspace to see all the values you've set

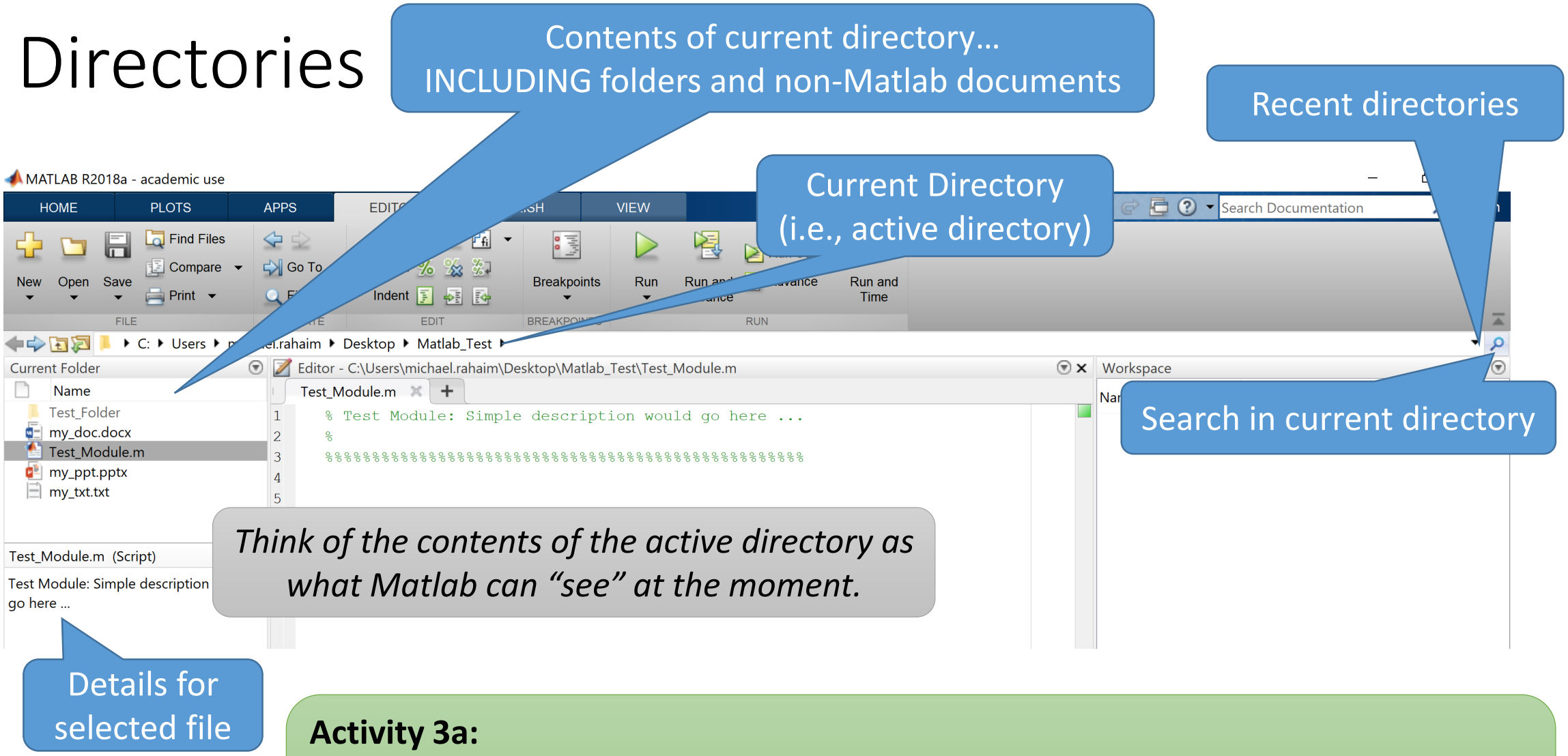
What happens when you double click a value in the workspace?





This view is more relevant later (matrices, structures, debugging etc.)

Directories



Activity 3a:

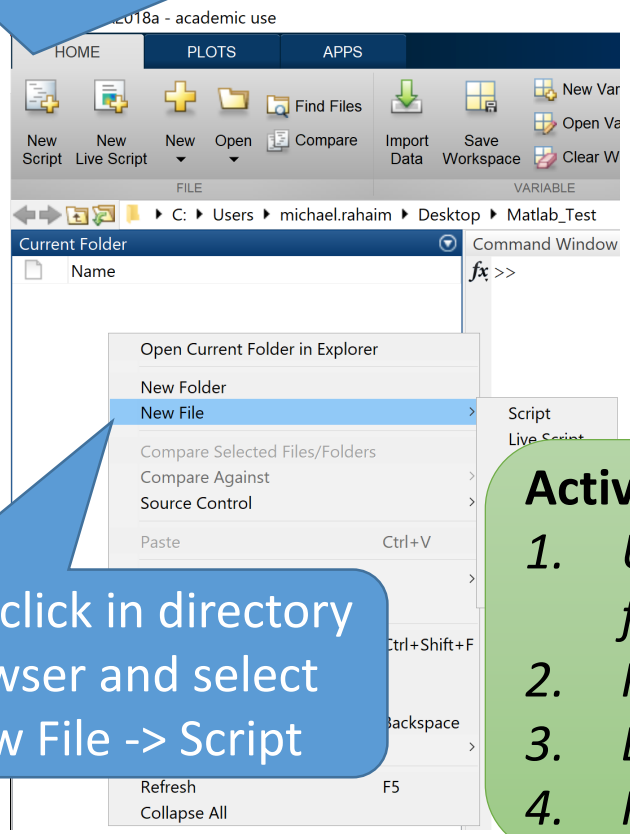
1. *In the Matlab file browser, navigate to your Desktop and create a new folder.*
2. *Rename the folder Matlab_Tutorial*
3. *Make Matlab_Tutorial your active directory*

Editor – Matlab Scripts

We will discuss scripts in more detail later

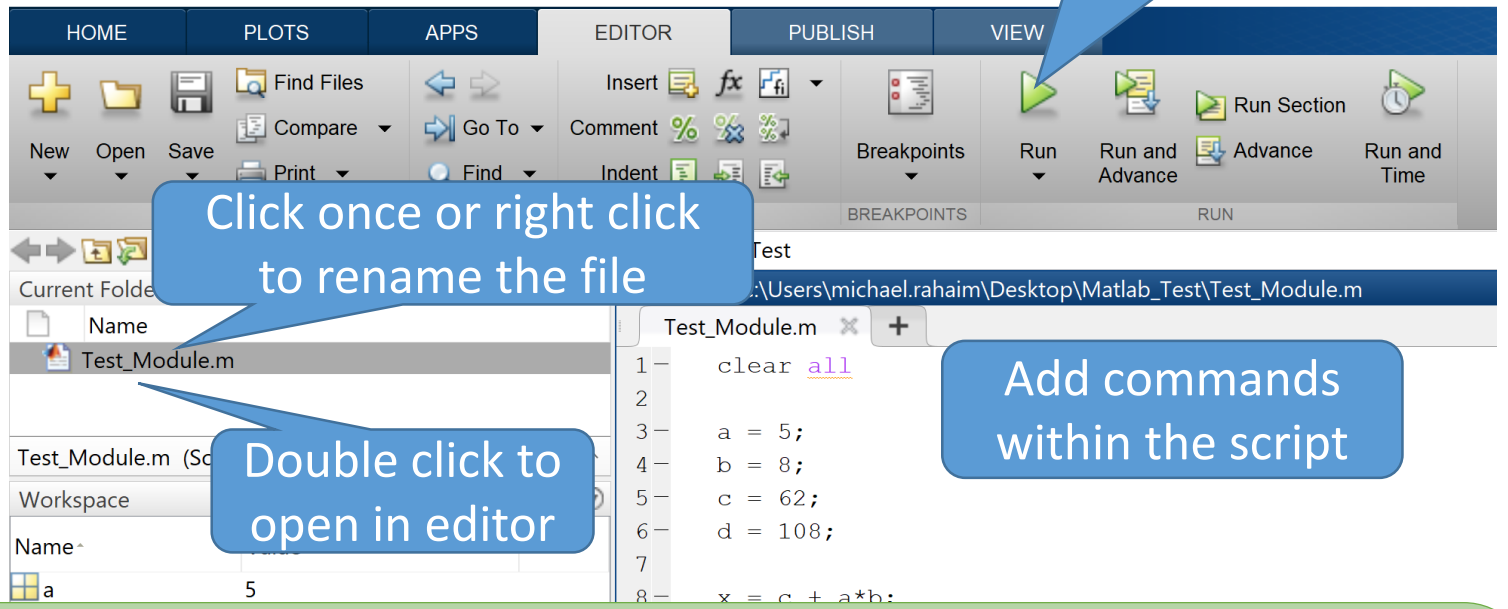
- Scripts are a simple way to specify a series of commands
  - Create a new script

Or, select New File -> Script in HOME tab



Right click in directory browser and select New File -> Script

Select "Run" to execute script



Click once or right click to rename the file

Double click to open in editor

Add commands within the script

Activity 3b:

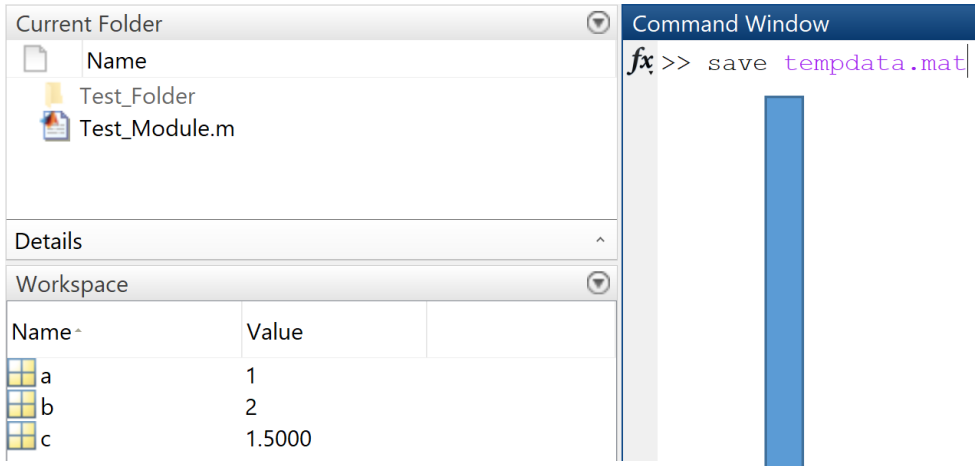
1. *Using the Matlab file browser, find ACTIVITY03.m in the shared OneDrive folder, copy it, and paste it into the Matlab_Tutorial folder on your desktop.*
2. *Run the script by typing "ACTIVITY03" in your command window*
3. *Double click ACTIVITY03.m to open it in the editor*
4. *Run the script again, this time using the "Run" button under the Editor tab*

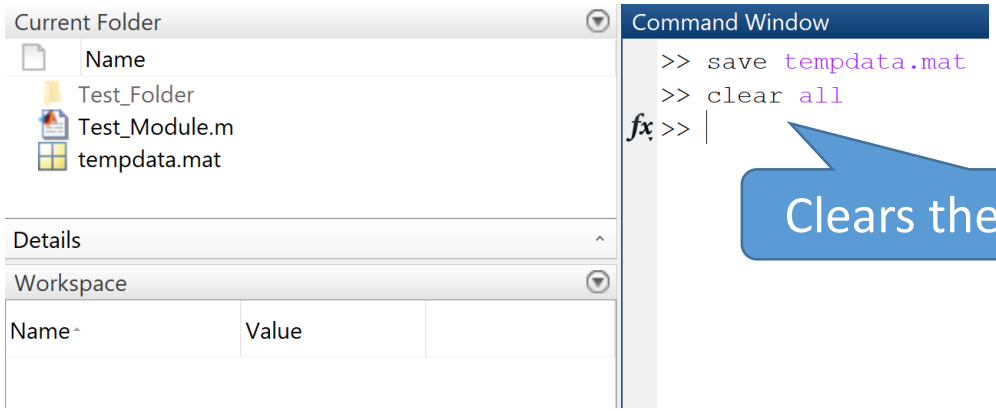
Saving / Loading .mat Files

Useful Commands

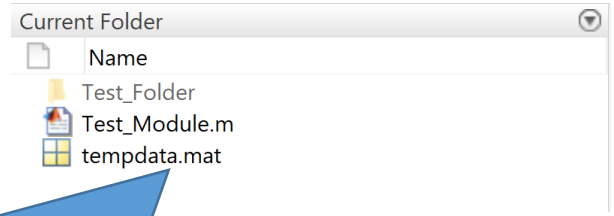
```
save
load
```

- You can save your current workspace to work with in the future

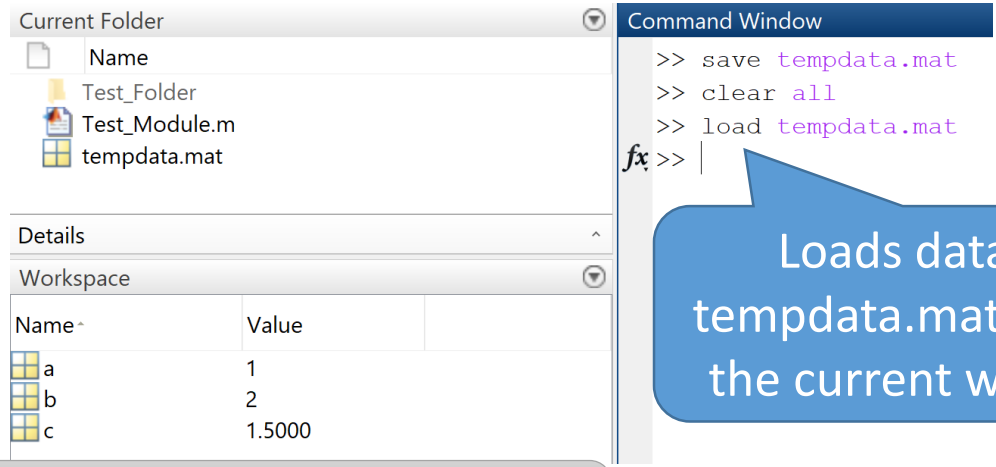




Clears the workspace



Variables in current workspace are saved to a specified .mat file in current directory



Loads data from tempdata.mat back into the current workspace

You can also double-click on the mat file from Matlab's File Browser to open it

Matlab Help and Documentation

Useful Commands

```
max()
min()
mean()
```

- Ok… basic algebra… but Matlab can do SO much more!
  - Vast library of existing functions

- How do we know how to use a specific function?
  - One of Matlab's best features is the documentation and help functionality

1. **Command Line:** `help function_name`
2. **In the dev environment:** Home tab, *Help* → *Documentation*
3. **Online Documentation:** https://www.mathworks.com/help/matlab/

Activity 4:

1. *Use the 'help' command to determine how to use the* `max()` *function*
2. *Use the online documentation to learn how to use the* `mean()` *function*

Basic Data Types and Conversions

Discussion Topics:

- *Basic Types*
- *Arrays and Cells*
- *Type Conversions*
- *Characters and Strings*
- *Operators and Basic Operations*

Basic Data Types

Useful for validating input when your program takes input from somewhere external

Useful Commands

```
isfloat()
isnumeric()
isreal()
isfinite()
isnan()
```

- **Data Types:**
  - https://www.mathworks.com/help/matlab/data-types.html
- **Numeric Data Types:**
  - https://www.mathworks.com/help/matlab/numeric-types.html
- **Examples:**
  - **double:** double precision floating point
  - **unit8:** 8-bit integer values
  - **logical:** Boolean (i.e., true or false value)

We will see later that logical values are useful for selectively operating on a subset of values in large arrays.

Activity 5:

- Create two variables:
  a=10; b=int8(10);
- Observe the values in the variable editor.
  What's the difference?
- In the variable editor, Change values to 1.5.
  What do you see?
- Change the values to 500. *What do you see?*

Type casting can be useful for large data sets when memory size (and execution time) needs to be considered

Arrays (i.e., Vectors)

- Defining Arrays
  - A = [1,2,3,4]
  - Using colon notation:
    - B = 1:10
    - C = 1:2:10
    - D = 10:-5:-20
  - Array definition functions:
    - X = zeros(1,20)
    - Y = ones(1,10)
    - Z = 5*ones(1,5)
  - Array Concatenation
    - temp = [X,Y,Z]

Take a look at this in the variable viewer

Useful Commands

```
zeros()
ones()
```

Row Arrays vs. Column Arrays
When combining arrays you will need to pay attention to the primary dimension

Compare the following:
x1 = [1,2,3,4]
x2 = [1;2;3;4]

Take a look at these in the variable viewer

You can use ’ to Transpose:
X3 = x1’;

Activity 6:
1. Load *ACTIVITY06.mat* from OneDrive
2. Find the min value in ARRAY1.
3. Find the ***index*** of the max value in ARRAY2.
4. Concatenate ARRAY3 and ARRAY4, find mean

Cells and Cell Arrays

Combining cell arrays is more tedious than combining arrays of the same data type

- Cells act as “containers” for variables of different data types / lengths

- Cells, and cell arrays, are created with { } rather than [ ]
  - mycell1 = {};
  - mycell2 = {1,2,3};

- Cell arrays can hold different data types AND sizes
  - mycell3 = {1,int8(2)}
  - mycell4 = {[1,2,3], [1.5], 10:15};

Take a look at this in the variable viewer

(You can double click on specific cells in the variable viewer to see their contents)

Type Conversions

Useful Commands

`cast()`
`typecast()`

- You can convert between certain data types using the cast function

```
a = 2;
b = cast(a,'int8');
```

```
c = 5:10;
d = cast(c,'int8')
```

- Why would we want to do this?

The saved files should be around the same size on disk!! This is because the save function uses compression by default

Activity 7:

1. Create array of 10000 doubles: `A = 1:10000;`
2. Cast as 16 bit unsigned integers: `B = cast(A,'uint16');`
3. Compare memory size by typing `whos` in the Matlab command window
4. Save each array as a separate .mat file
   - Method 1: Right click on the variable in your workspace. Select “Save As”
   - Method 2: Use the save function to save a single variable (use *help save* to find out how)
5. Look at the files in windows explorer. *How do the file sizes compare?*

Try saving A and B this way:

```
save('temp1.mat','A','-v7.3','-nocompression');
save('temp2.mat','B','-v7.3','-nocompression');
```

Characters / Char Arrays

Useful Commands

`num2str()`

- *Characters* allow for text storage
  - Defining characters: `myChar1 = 'a';  myChar2 = '1';`
  - Casting numbers as characters: `a = 1;  b = cast(a,'char');`
  - Using ***num2str*** as an alternative: `b = num2str(1);`

Char representation of number 1

Does this give what you expect?

Try `b = cast(a+48,'char');`

Offset for ASCII value

- *Char Arrays*
  - Defining char arrays: `charArray = 'Hello World';`
  - *num2str* for numeric char arrays: `a = 3.1415; b = num2str(a);`
  - Concatenate Char Arrays: `a = 'Hello'; b = 'World'; c = [a, ' ', b];`
  - Concatenate with *num2str*:

This is VERY useful when labeling plots (as we will see later)

```
myVal = 2.7183; myChars = ['myVal = ', num2str(myVal)];
```

Strings

- Storing multiple char arrays in matrix form has challenges:

```
myChars = ['ab'; 'cd'; 'ef'];
```

This is OK

```
myChars = ['Hello'; 'World'; 'yay!'];
```

This is NOT valid

Different size char arrays can NOT be combined this way

- Cell Arrays

```
myChars = {'Hello', 'yo'; 'Foo', 'Bar'};
```

The cell array holds multiple char arrays

```
temp1 = myChars{3};   temp2 = myChars{2,2};
```

Each char array can be addressed

- Strings versus Character Arrays
  - Defining Strings: `myStr = "Hello World";`
  - String Arrays: `myStrings = ["Hello", "World"; "Foo", "Bar"];`

You can see the difference in your variable editor

Strings have more functionality and are better for working with text as data

Operators and Basic Operations

If you're working with large arrays and you aren't sure how a command works, try it with smaller arrays to visualize the result!

- Some of the basic scalar operators
  - Arithmetic: (+, -, *, /, ^)
  - Logical: (==, >, <, >=, <=, ~=, &, |)

Note: Be aware of what happens when trying to operate on both Row and Col Arrays:

```
C = A + B';
```

- With multiple arrays, we often want to operate element-wise:

```
A = [1,2,3,4];
B = [2,2,3,3];
```

```
X1 = A + B;
X2 = A - B;
```

```
X3 = A.*B;
X4 = A./B;
X5 = A.^B;
```

- Selecting values from an array:
  - Specific Values: `A = 101:110;  B = A([1,1,2,6]);`
  - Range Selection: `A = 101:110;  C = A(4:6);`
  - Logical Selection: `A = 101:110;  D = A(logical([0,0,1,1,0,0,0,1,1,0]));`

Need to cast the array of 0s and 1s as a logical array!

Operators and Basic Operations

Useful Commands

`sum()`

- Multi-Array Logical Operators
  - Array/Value Logical Comparison

Note: X and Y are *logical* arrays here!

```
A = 101:110;        X = A > 103;        Y = (A>103) & (A<=108);
```

  - Multi-Array Logical Comparison

```
A = [1,2,3,4];   B = [2,2,3,3];   X = (A == B);
```

  - Logical Value selection

```
A = 101:110;   B = A((A>103) & (A<=108));
```

```
A = 101:110;   B = 0:9;   C = A((B>3) & (B<8));
```

Useful for selecting values from one array based on values in another array

Activity 8:

1. Load *my_vals* and *my_time* from ACTIVITY08.mat
2. Create a new array with the values from *my_vals* that correspond to values in *my_time* from 25 to 50 and from 75 to 80.
3. Use sum() function to find the sum of all values in your new array.

Data Visualization

Discussion Topics:

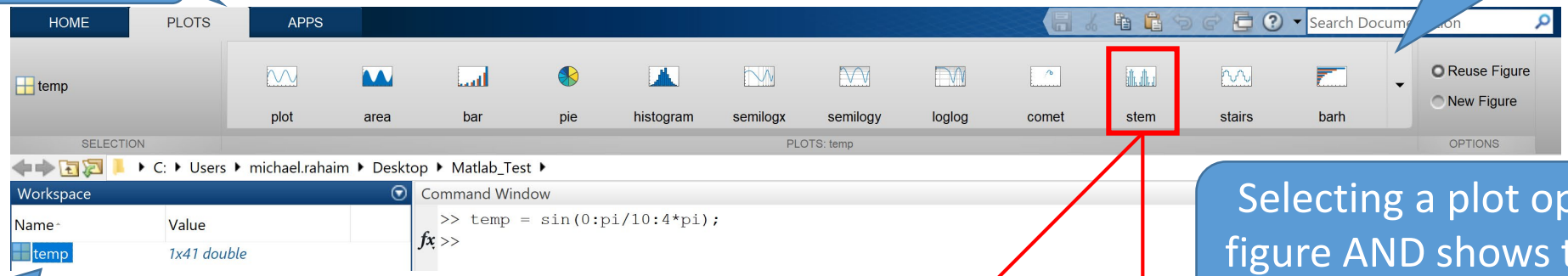
- *Plotting (2D)*
- *Plotting (3D)*
- *Videos / movies*

Two Dimensional Plots

- Matlab has MANY options for displaying data…
  - You can use the Matlab environment to explore plot options

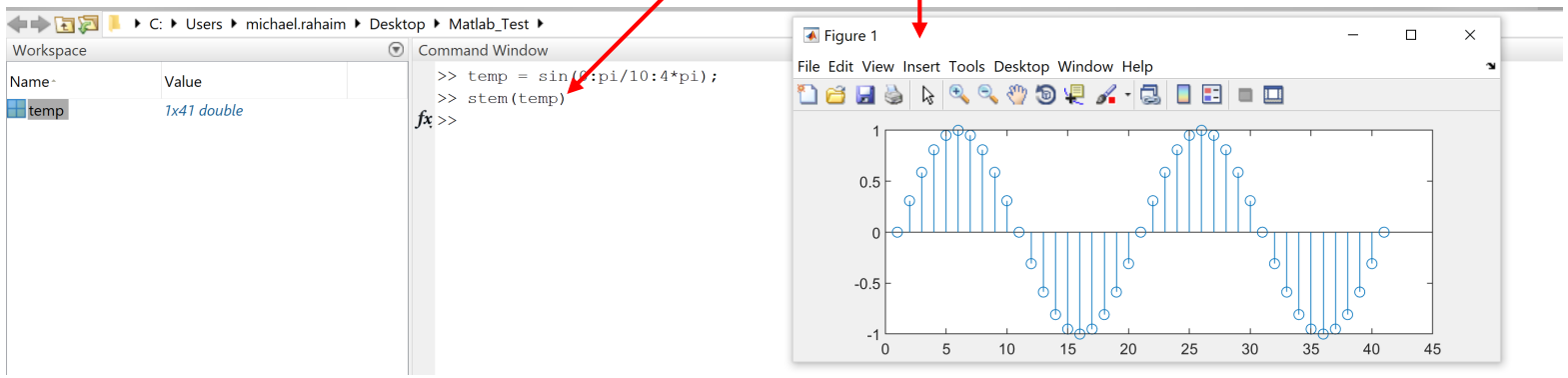
Look in the plots tab

More plot options for the specific variable



Selecting a plot option opens a figure AND shows the command line call for the desired plot

Make sure the desired variable is selected

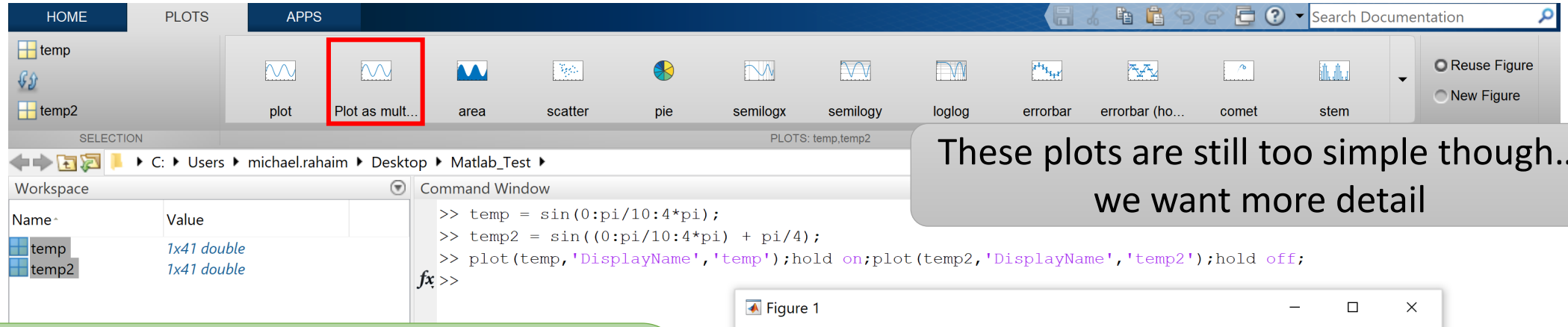


Two Dimensional Plots

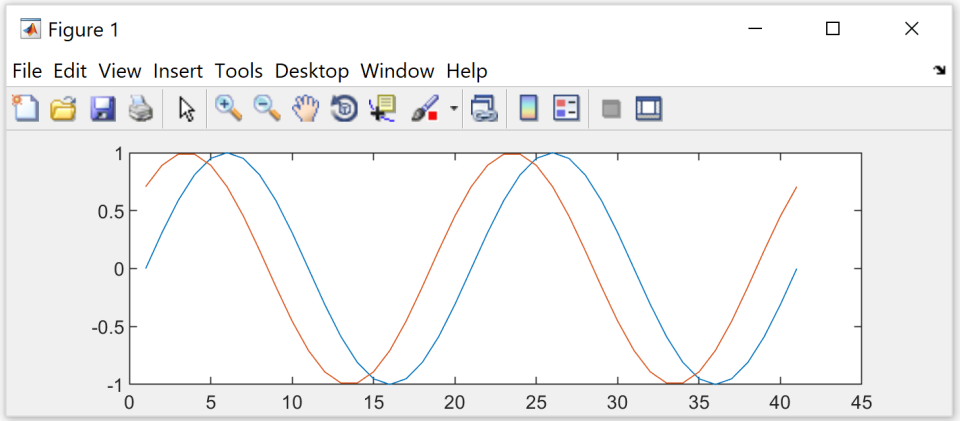
Useful Commands

`legend()`

- Matlab has MANY options for displaying data…
  - You can use the Matlab environment to explore plot options
  - You also use the environment for plotting multiple variables!



These plots are still too simple though… we want more detail



Activity 9a:

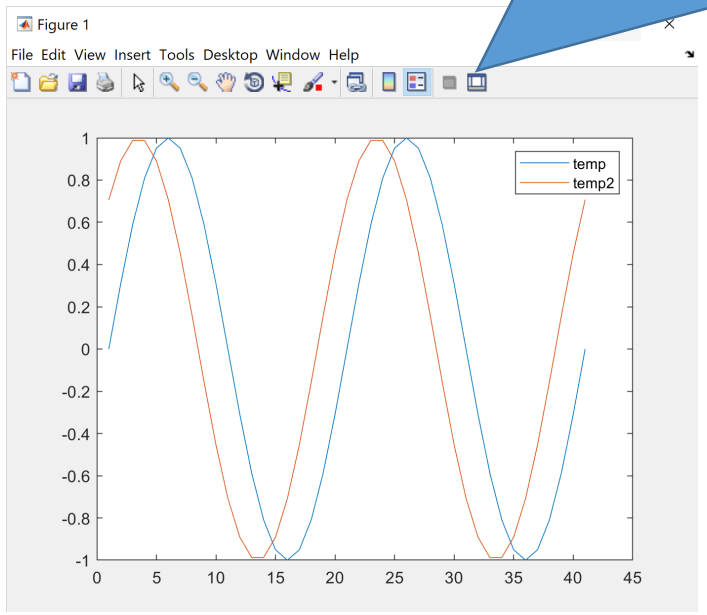
1. Create the sinusoids temp1 and temp2
2. Plot the two sinusoids on the same plot
3. Add a legend by typing `legend()` in the command window

Two Dimensional Plots

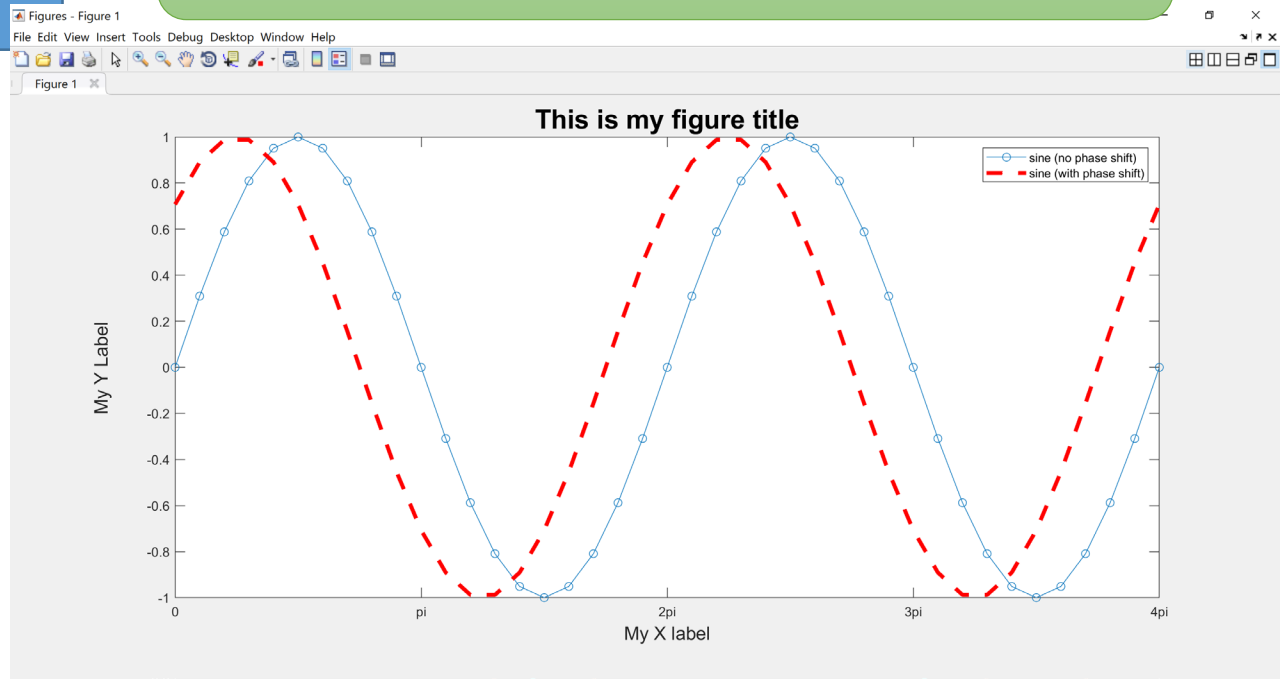
The graphical option is a nice way to edit a single figure; but the command line is much better for repeatability and creating many figures

- **Editing Figures**
  - We can use commands like `legend()` to specify the figure's appearance
  - We can also edit the appearance graphically with Matlab's plot tools

The plot tools interface lets you modify figure properties (e.g., line width and style, titles and axis labels, fonts, axis range and values, etc.)

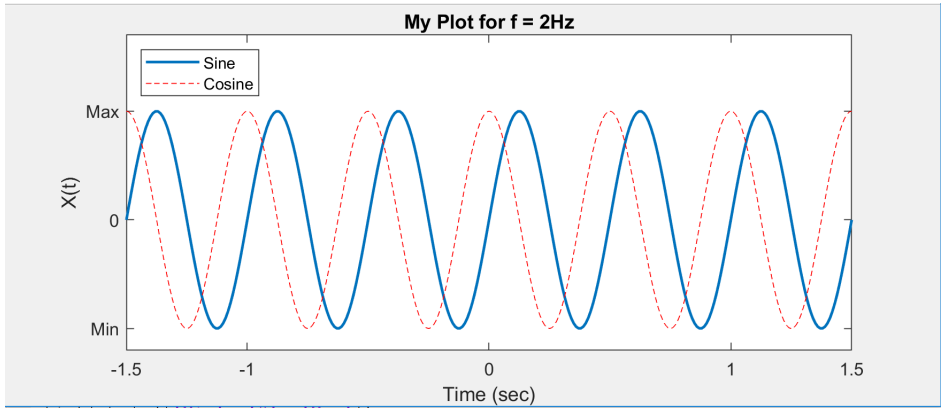


Activity 9b:
Use plot tools and try to recreate this figure



Two Dimensional Plots

- Command line Editing
  - Example:




```
f = 2;
t = -1.5:0.01:1.5;
x1 = sin(2*pi*f*t);
x2 = cos(2*pi*f*t);
figure('Position',[100,100,700,300]);
plot(t,x1,'LineWidth',1.5); hold on
plot(t,x2,'LineStyle','--','Color','r');
title(['My Plot for f = ' num2str(f) 'Hz']);
xlabel('Time (sec)');
ylabel('X(t)');
xticks([-1.5,-1,0,1,1.5]);
yticks(-1:1)
ylim([-1.2,1.7]);
yticklabels({'Min', '0', 'Max'});
legend('Sine','Cosine','Location','NorthWest');
```

Specify figure size and location

Specify line properties https://www.mathworks.com/help/matlab/ref/matlab.graphics.primitive.line-properties.html

Plotting array as a function of another array

Use variable in label

Modify labels and axes

We will later see how these hard coded values can be more cleanly specified in a script

Three Dimensional Plots

Useful Commands

`meshgrid()`

- Matlab can also plot in a 3rd dimension
  - Here you need X, Y, and Z values
  - The `meshgrid` function is an easy way to get X and Y values for a grid
    - Example: `[x,y] = meshgrid(-3:3, 0:0.5:2);`

Note that the function puts out 2 resulting variables

See what this gives you (observe the workspace and variable viewer)

  - Plot Examples

```
res = pi/8;
range = -2*pi:res:2*pi;
[x,y] = meshgrid(range,range);
z = sin(x) + cos(y);
figure()
surf(x,y,z);
figure()
surf(range,range,z);
```

Using variables here makes making changes easier.

Two ways to get the same resulting plot

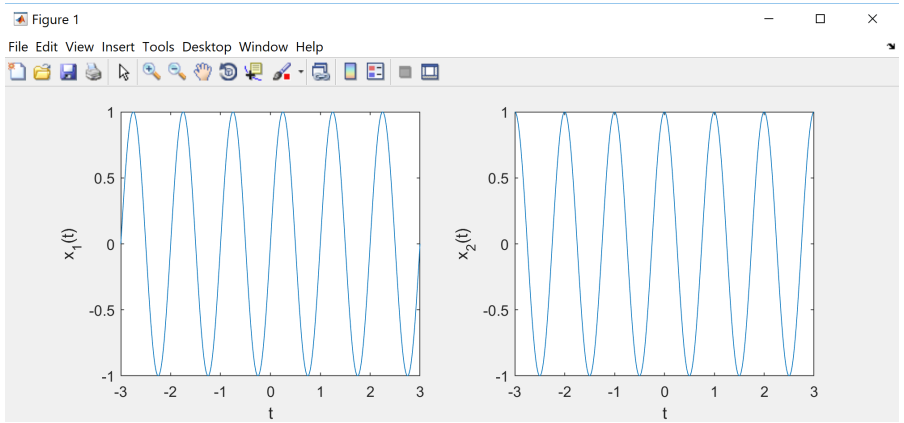
Some other useful 3D plot functionality

```
figure()
surf(x,y,z);
colorbar
rotate3d on
shading interp
```

Multi-Plot Figures

- You can also organize plots with subplots in the same figure.
  - Use `subplot(nRows,nCols,N)` to specify grid arrangement and specific plot

```
t = -3:0.01:3;
x1 = sin(2*pi*t);
x2 = cos(2*pi*t);
figure('Position',[100,100,800,300]);
subplot(1,2,1);
plot(t,x1);
xlabel('t');
ylabel('x_1(t)');
subplot(1,2,2);
plot(t,x2);
xlabel('t');
ylabel('x_2(t)');
```



Text underscore displays as subscript

Activity 10:

1. Load ACTIVITY10.mat
2. Create a figure with multiple plots:
   a) surf plot of `X`, `Y` and `Z`
   b) 2D plot of `x_t` versus `t`
   c) stem plot of `y_t` versus `t` showing ONLY the range where `x_t` is greater than 0

Try using logical operators

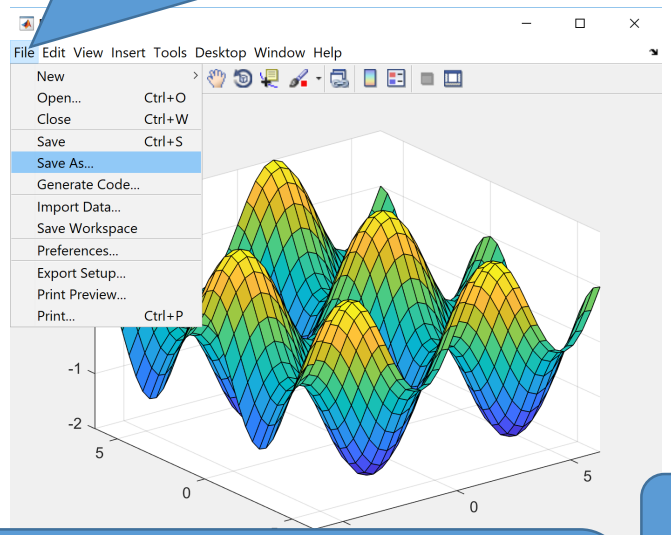
Saving Figures

Useful Commands

```
savefig()
saveas()
```

- I'm using screenshots to show the way figures look when created…
  - But GENERALLY, avoid screenshots and save your figures and plots directly

Method 1: Save in the figure editor



When you need to save 20, 50 100, etc. images, you will be happy to have this option rather than trying to screen shot each figure!

Method 2: Command Line

```
res = pi/8;
range = -2*pi:res:2*pi;
[x,y] = meshgrid(range,range);
z = sin(x) + cos(y);
h = figure();
surf(x,y,z);
savefig(h,'myfig.fig','compact');
saveas(h,'myfig.jpg');
```

h is a "handle" that contains information about the figure

savefig() saves figure that can be reloaded in Matlab later

saveas() saves a graphics file of the specified figure

If you want to preserve the quality of complex images, save in a vector graphics format (e.g., pdf, eps, …)

Videos / Movies

Useful Commands

```
getframe
movie()
VideoWriter()
```

- Figures can be combined as frames of a movie
  - **Step 1:** collect frames by using `M(k)=getframe` after each plot
  - **Step 2:** Play the movie with `movie(M,n)`

For each frame, specify the value of k (i.e., k=1,2,3,…

M is the collection of frames, n is how many times to repeat

- To save the movie as a video file:

Create a VideoWriter object

```
v = VideoWriter('myvideo.avi');
open(v);
writeVideo(v,M(1));
writeVideo(v,M(2));
writeVideo(v,M(3));
writeVideo(v,M(4));
writeVideo(v,M(5));
close(v);
```

Write frame M(k) to the file

This is not a practical way to implement larger movies. We will come back to this when discussing scripts / loops.

Activity 11:

1. Open the script ACTIVITY11.m
2. Execute the script to play the movie
3. Add code to the script to save the movie as a .avi file


BREAK
TIME !!

Matlab Scripts

Discussion Topics:

- *Scripting – Best Practices*
- *Flow Control*
  - Conditional Statements
  - Loops
- *Debugging Scripts*

Matlab Scripts - Overview

- Scripts improve repeatability and allow you to create larger programs
- Best practices for scripting:

Good commenting also helps format reports (discussed later)

1. **Comments**
   - Single line comments vs. multiline comments
   - Comment Section Separations

```
A = 1:10000; % This is an array...
% A = 1:10000;
```

```
%{
A = [1,2,3,4];
B = [2,2,3,3];
X1 = A + B;
X2 = A - B;
%}
```

```
29 -  A = [1,2,3,4];
30 -  B = [2,2,3,3];
31    %% Section 1
32 -  X1 = A + B;
33 -  X2 = A - B;
34    %% Section 2
35 -  X3 = A.*B;
36 -  X4 = A./B;
37 -  X5 = A.^B;
```

Section headers help you navigate the file (file viewer / details)

```
29 -  A = [1,2,3,4];
30 -  B = [2,2,3,3];
31    %{
32    X1 = A + B;
33    X2 = A - B;
34    %}
35 -  X3 = A.*B;
36 -  X4 = A./B;
37 -  X5 = A.^B;
```

```
29 -  A = [1,2,3,4];
30 -  B = [2,2,3,3];
31    %{...%}
35 -  X3 = A.*B;
36 -  X4 = A./B;
37 -  X5 = A.^B;
```

Activity 12a:

1. Open and execute ACTIVITY12.m
2. Add section headers throughout the script
3. Observe the file details in the File Browser

Matlab Scripts - Overview

- Scripts improve repeatability and allow you to create larger programs
- Best practices for scripting:
  2. **Writing commands on multiple lines (…)**
     - Improves readability of your script

```
myChars1 = {'This is a cell array', 'of different char arrays', 'that is very long', 'and difficult to read'};
```

```
myChars2 = {'This is a cell array', ...
            'of different char arrays', ...
            'that is very long', ...
            'and and easier to read'};
```

Activity 12b:

1. Continue working with your modified version of ACTIVITY12.m
2. Use multiple lines for commands that extend beyond the single page

Matlab Scripts - Overview

- Scripts improve repeatability and allow you to create larger programs
- Best practices for scripting:
  3. **Use meaningful variable names**
     - Find a convention that you like, and stick to it!

When you change a variable name in a script, you can change all instances of the variable with ***shift + enter***

```
temp = 0:0.01:2;
temp1 = 0.5;
temp2 = 5;
temp3 = pi/4;
temp4 = temp1*cos(2*pi*temp2*temp + temp3);
plot(temp, temp4);
```

```
t = 0:0.01:2;
A = 0.5;
f = 5;
phi = pi/4;
x_t = A*cos(2*pi*f*t + phi);
plot(t, x_t);
```

```
temp = 0:0.01:2;
temp1 = 0.5;
temp2 = 5;
temp3 = pi/4;
temp4 = temp1*cos(2*pi*temp2*temp + temp3);
plot(temp, temp4);
```

Selecting a variable in a script highlights all instances of the variable

Activity 12c:

1. Continue working with ACTIVITY12.m
2. Change variables “temp1” and “temp2” to give them meaning

Matlab Scripts - Overview

- Scripts improve repeatability and allow you to create larger programs
- Best practices for scripting:
  4. **Avoid *hard coding*!**
     - Makes it easier to manipulate values in large scriptss
     - Helps avoid mismatched values

```
%% Plot a 2Hz sin and cos with Amplitude 5
t = 0:0.01:2;
x1_t = 5*sin(2*pi*2*t);
x2_t = 5*cos(2*pi*2*t);
```

I want to see how they compare at 1Hz…

```
t = 0:0.01:2;
x1_t = 5*sin(2*pi*1*t);
x2_t = 5*cos(2*pi*2*t);
```

Easy to miss one of the hard coded values!

```
t = 0:0.01:2;
A = 5;
f = 2;
% Plot a sin and cos with
% Amplitude A and frequency f
x1_t = A*sin(2*pi*f*t);
x2_t = A*cos(2*pi*f*t);
```

Easier to modify!

Activity 12d:

1. Continue working with ACTIVITY12.m
2. Change parameters and execute again
3. Open ACTIVITY12_2.m and notice the parametrized values.
4. Change parameters in ACTIVITY12_2.m and execute

Flow Control – Conditional Statements

Useful Commands

`listdlg()`
`find()`

- Conditional Statements
  - *If / end*
  - *If / else / end*
  - *If / elseif / end*
  - *switch / case / otherwise*

Useful when selecting one of many options (rather than using if / elseif / elseif / … / etc.)

```
if (a == 1)
    % Do something here...
end
```

```
if (strcmp(temp_string,'desired string'))
    % Do something here...
else
    % Do something else here...
end
```

```
if (strcmp(temp_string,'desired string'))
    % Do something here...
elseif (a ~= 1)
    % Do something else here...
else
    % Otherwise do this...
end
```

Activity 13:

1. Open and execute ACTIVITY13.m
2. Add conditional statements so that the script only plots the figures selected by the user
   Hint: Try using the `find()` function

Flow Control – Loops

- For loop
  - Iterate through defined values
- While loop
  - Iterate until something occurs

```
t = 0:0.005:1;
for i = 1:2:5
    x = sin(2*pi*i*t);
    plot(t,x,'DisplayName', ...
          ['f=' num2str(i) 'Hz']);
    hold on;
end
legend();

answer = 'Yes';
while (strcmp(answer,'Yes'))
    answer = questdlg('Continue?');
end
```

Useful Commands

`randn()`

Activity 14:

1. *Create a new script that uses the randn() function to make an array of 1000 random numbers and plot a histogram representing the frequency of occurrence of different numbers.*
2. *Add a for loop that iterates from 2 to 100 with a step size of 2. On each iteration, create a random array of length equal to the iteration number and plot the histogram of the array.*
3. *Add code to save a movie frame of the histograms from each iteration*
4. *Play the movie at the end of your script.*
5. *Add another loop that goes through the frames and saves a video file in .avi format*

Debugging

- As your programs / scripts become more complex, you will inevitably encounter errors in your code…
  - Debugging with the Matlab debugger will help you resolve your errors!

- In the development environment, you can:
  - Set ***break points*** in your script
  - ***Step*** through your script one line at a time
  - ***Step into*** functions (e.g., see content of Matlab Function)
  - Use the workspace observe variables while stepping though script
  - Use the command line to see why something isn't working correctly

Activity 15:

1. Open and execute ACTIVITY15.m
2. Set a break point and step through the script to see why it isn't working as specified.

Functions, Structures, and Classes

Discussion Topics:

- *Matlab Functions*
- *Structures*
- *Classes*

Functions

- Matlab has MANY built-in functions
  - *Matlab Core Functionality*
  - *Toolboxes*
  - *Matlab Central / File Exchange*

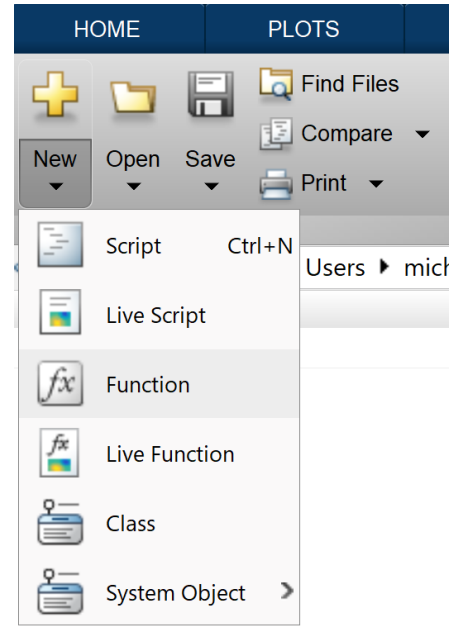
- Functions with multiple input arrangements
  - Revisit the `mean()` function
  - Specifications / Parameters as input (e.g., `plot()` function)

- Functions with multiple output values:

```
myMax = max(A);     [myMax,myIndex] = max(A);     [~,myIndex] = max(A);
```

Creating Functions

- Creating functions in separate Function files
  - From home menu or by right clicking in file browser



Needs to have same name as the file name

Editor - C:\Users\michael.rahaim\Desktop\Matlab_Test_mike\myF

myFunction.m

```
1  function [outputArg1,outputArg2] = myFunction(inputArg1,inputArg2)
2  %MYFUNCTION Summary of this function goes here
3  %   Detailed explanation goes here
4  outputArg1 = inputArg1;
5  outputArg2 = inputArg2;
6  end
7
```

No specification for input data type

Returns the current values of the output at the completion of the function

Note: In order to call the function, Matlab must be able to find it! Matlab will look in the current active directory and in the ***Matlab Path***

Creating Functions

Note: You can add to Matlab's search path: https://www.mathworks.com/help/matlab/matlab_env/what-is-the-matlab-search-path.html

- Creating functions in same file functions

myFunction.m

```
1   function [outputArg1,outputArg2] = myFunction(inputArg1,inputArg2)
2   %MYFUNCTION Summary of this function goes here
3   %   Detailed explanation goes here
4   outputArg1 = inputArg1;
5   outputArg2 = internal_function(inputArg2,inputArg1);
6   end
7
8   function [outputArg1] = internal_function(inputArg1,inputArg2)
9       outputArg1 = inputArg1/inputArg2;
10  end
```

This is a local function… it can't be called outside the file

Activity 16:

1. Create the function file shown above
2. Determine the output when `inputArg1 = 10` and `inputArg2 = 5`
3. Set a break point on line 4 and call the function by typing `[a,b] = myFunction(10,5);`
4. Run the program and use *step* and *step in* to see the values of `inputArg1` and `inputArg2` at different points in the function call.
5. Change your active directory and try calling the function. *What happens?*

Structures and Classes

- **Structures** (https://www.mathworks.com/help/matlab/ref/struct.html)
  - Somewhat like a cell array with meaningful indices…
  - Example:

```
>> myStruct.name = 'My Structure';
>> myStruct.val1 = 3;
>> myStruct.array = zeros(1,3);
```

Activity 17:

1. Create the structure shown here.
2. Open and observe the value in the workspace variable viewer.
3. Create an array with `myArray(5) = myStruct;` and observe in the variable viewer

- **Classes** (https://www.mathworks.com/help/matlab/object-oriented-programming.html)
  - Classes take it further, allowing objects to have functions and parameters.
  - Classes are created in a new file (similar to functions)

Additional Material

Discussion Topics:

- *Data import / export*
- *Matrices*
- *Graphical User Interfaces (GUIs)*
- *Matlab Live*
- *Matlab Reports*
- *Matlab Profiler*
- *Matlab Compiled Code*

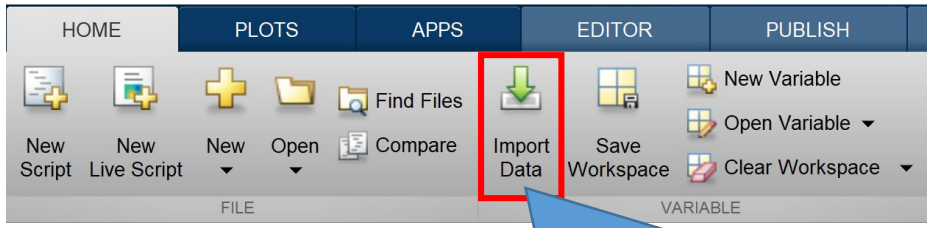
Data Import / Export

e.g., signal collected from scope and saved as .csv or .xls file

Useful Commands

```
xlsread()
xlswrite()
```

- We often want to analyze / observe data collected externally
- **Import Tool**



Graphical Interface for importing data

Activity 18:

1. Select the *Import Data* option on Matlab's Home tab.
2. Find and open the **ACTIVITY18.xls** spreadsheet
3. Select all data in rows 2-5
4. Select *Variable Names Row* as row 1
5. Select Column Vectors as Output type
6. Select Cell Array of Char Vectors for text options
7. Select Import Data from the Import Selection drop down list

- **Command line**
  - More efficient / repeatable to import within a script…
  - Repeat activity 18; but select *generate script* from the import selection drop down list for step 7. Run this script and step through with the debugger to see what is happening.

Can also export to excel with `xlswrite()` or import / export .csv with `csvread()` / `csvwrite()`

Other Interesting Material

- Matlab Matrices
  - https://www.mathworks.com/help/matlab/learn_matlab/matrices-and-arrays.html
- Matlab Graphical User Interface
  - https://www.mathworks.com/discovery/matlab-gui.html
- Matlab Live
  - https://www.mathworks.com/products/matlab/live-editor.html

Other Interesting Material

- Matlab Reports
  - https://www.mathworks.com/products/ML_reportgenerator.html
- Matlab Profiler
  - https://www.mathworks.com/help/matlab/ref/profile.html
- Matlab Compiler
  - https://www.mathworks.com/products/compiler.html