Lab 01: Matlab tutorial

EE 20 Fall 2014
University of California, Berkeley

1 Introduction & creating scalar variables

Welcome to EE 20! In this first lab, you will be introduced to the basics of Matlab. This will be the foundation for future labs.

First, Start up Matlab on your computers.

When you start up Matlab, it should look like Figure 1. The *command window* is where you will type your code. Let's try something very simple: create a variable, say a, and set it to the value 1. All you need to do is type the following: `a = 1`

You should realize that as soon as you do so, in the *Workspace* window, there is now a variable a, with the value 1. Also note that Matlab printed `a = 1` in the command window. You may not need this; for example, if you are creating a vector with 1000 entries, you may not need Matlab to print out the 1000 entries.

To suppress the printing, use the semicolon. Try typing `b = 2;`(note the semicolon)

You should see a variable `b` pop up in the Workspace window, and if you call it by typing `b`, you will get the value. But there is no automatic printing when you created the variable.

You may remember from previous math classes that this type of variable is called a scalar, as opposed to the vector/matrix variables we will discuss below.

2 Vectors/Matrices

One of the things Matlab allows you to do easily is mathematical operations on vectors and matrices. Let's look at the basic commands.

2.1 Creating

creating vectors

Try the following commands and see what they do:

```
u = [2 4 5]
v = [2; 4; 5]
v = [2 4 5]'
w = 2:5
u = 1:2:7
```

the colon

The important thing to notice here is the colon operator.

`start:step:end` creates a vector that goes from `start` to `end`, with a step size of `step`.

If you do not specify `step`, the default step size is 1.

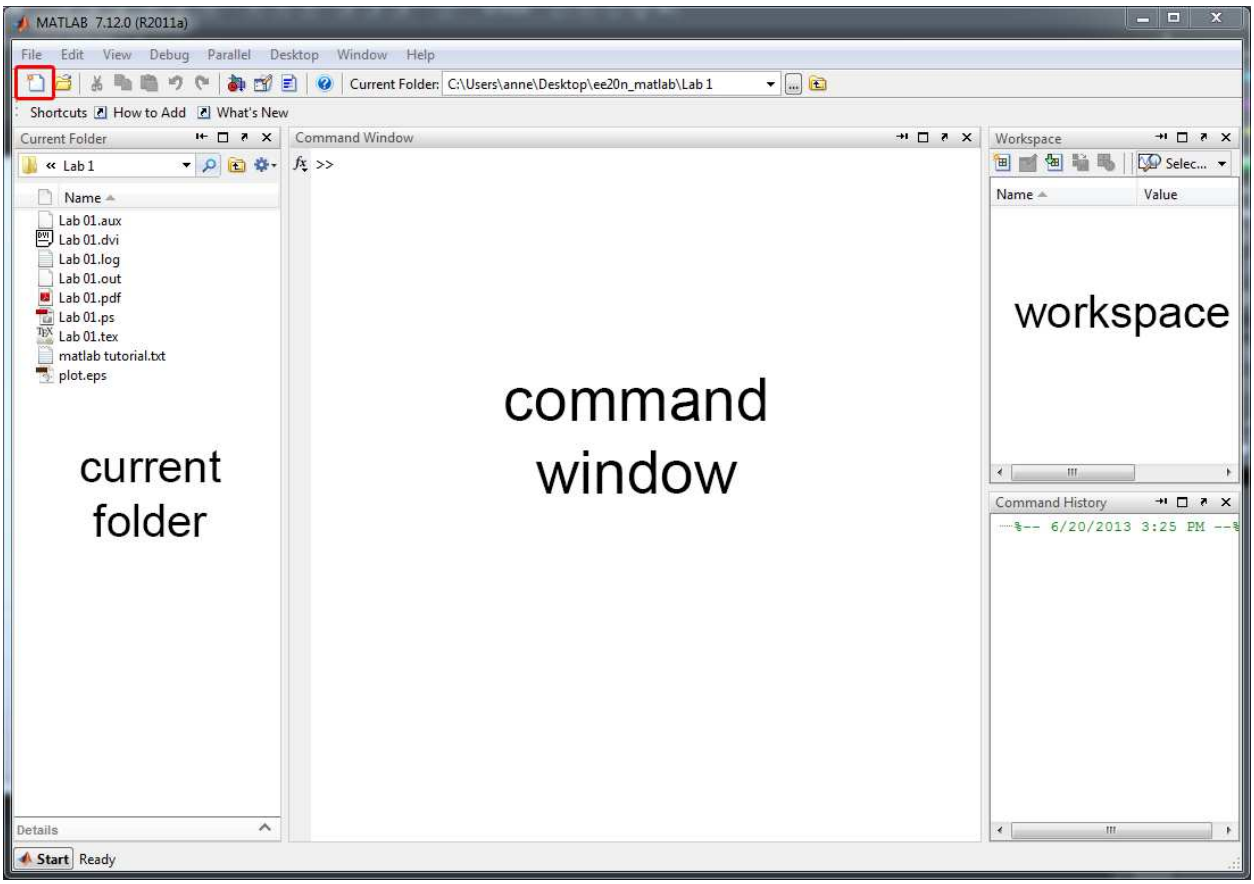


Figure 1: Matlab window on startup.

creating matrices

A = [1 2 3; 4 5 6] Creates a 2×3 matrix. The semicolon indicates the change in row.

2.2 Change entries/indexing

You can change the entries in a matrix or vector. Try the following:

```
A
A(1,1)
A(2,3)
A(2,:)
A(:,2)
A(2,2) = 4;
```

Which index is the row index? Which is the column index? What does the colon mean here? What did the last line do?

You can also try to cut a sub-matrix of a matrix. Try the following:

```
B = [1 2 3 4; 5 6 7 8; 9 10 11 12];
B(1:2,2:4)
```

What do each of the 4 numbers in the second line specify?

2.3 Special matrices

Matlab has many built-in functions, and some of them create special matrices so you don't have to type out all the entries. Two important matrices that have built-in Matlab commands are the all 1 and all 0 matrices, created by `ones` and `zeros`. Try the following and see what they do:

```
ones(2,3)
zeros(5,4)
```

2.4 Matrix operations

You can easily perform matrix operations in Matlab. For instance, you can transpose a matrix with `'`, and invert a Matrix with `inv`. Another useful command is `size()`, which tells you the dimensions of a given matrix. Try the following commands:

```
A = [1 2; 3 4];
A'
inv(A)
B = [1 2 3; 4 5 6];
size(A)
size(B)
```

2.5 Mathematical operations

Of course, just having vectors/matrices is no fun, we can do mathematical operations on them! Type the following code:

```
u = 1:0.5:3;
v = ones(1,5);
u+v
```

Addition is easy, but what about multiplication? You know that to do matrix multiplication, the inner dimensions of the two matrices must match. Try the following:

```
A = [1 2 3; 4 5 6];
B = [7 8; 9 10; 11 12];
A*B
```

We can also do point-wise multiplication:

```
u = 1:5
v = 6:10
u.*v
```

The dot (period) means that instead of treating this as two vectors u and v being multiplied, just multiply the first entry of u with the first entry of v, 2nd entry of u with 2nd entry of v...and so on.

Similarly, we can do other point-wise operations, like exponentiation:

```
u = [1 2 3; 4 5 6]
u.^2
```

This squares each entry of u.

2.6 How is this applicable to signals and systems?

We use Matlab to run various simulations of signals and how they are transformed by systems, but there are a couple of limitations on we can do. Since computers have finite memory, they cannot keep track of continuous time signals or infinitely long signals. This limits us to finite-length, discrete-time signals when using Matlab. (You can approximate continuous-time signals in Matlab by using a sufficiently large number of samples, but this will be covered later). We use vectors to represent discrete-time signals, with each component of the vector representing the value the signal takes on at a particular time. In fact, since a vector has a finite number of components, it can only represent discrete-time signals over a finite duration. For example, a signal $x(n) = \sin(\frac{\pi}{3}n)$ is an infinite duration signal, which we could represent in Matlab over a finite duration with

```
n = 0:0.1:6;
x = sin(pi/3 * n);
```

The vector x only has a finite number of values from $x(n)$.

3 Utilize the built-in help!

Before we keep going, there is one command that you should definitely know, and that is the `help` command. For example, for the ones function that you just saw, you can look up the Matlab documentation on it by typing:

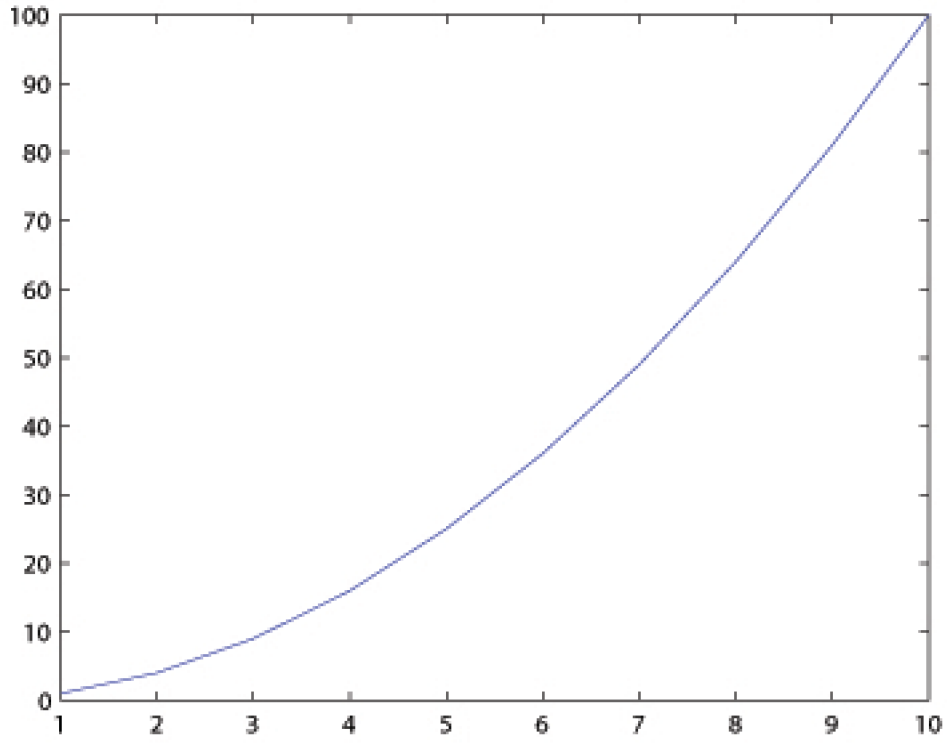
```
help ones
```

4 Plotting

4.1 Plot

There are many things Matlab can plot. For now, it is sufficient for you to know how to plot simple 2D plots to get started. The command to do so is `plot(x, y)`, which will plot the values of the vector `y` against the values of the vector `x`. Obviously, the two vectors `x` and `y` must have the same number of entries or Matlab will be mad.

```
x = 1:10;
y = x.^2;
plot(x, y)
```



4.2 Labeling

You can label both axes and the title of the graph, using the following commands:

```
xlabel(['x axis label'])
ylabel(['y axis label'])
title(['graph title'])
```

4.3 Axis control

By default, `plot` will give you a figure where the min/max x and y values are the smallest/largest values in the two vectors you plot. If you want to change the default behavior, use the `axis` command.

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

4.4 New figure and hold

What happens when you want to plot multiple graphs? If you want to plot it on a separate plot, you don't need any additional commands. Except there's a slight problem. Try the following code:

```
x = -5:0.1:5;
y1 = x.^2;
y2 = x.^3
```

```
plot(x, y1)
plot(x, y2)
```

What happened? You only get the 2nd graph! Matlab actually plotted the first graph, but when it plots the 2nd one, it overwrote the first one. To solve this, you can plot the 2nd graph on a new figure:

```
x = -5:0.1:5;
y1 = x.^2;
y2 = x.^3
figure()
plot(x, y1)
figure()
plot(x, y2)
```

Now there are two figures, one containing each plot, where the figures are created by the command `figure()`. What about when you want to plot two lines on the same figure? You need to use the `hold` command. `hold on` and `hold off` turns holding on and off. Using `hold all` will automatically also hold the color of the plot, so your new plot is automatically a different color. Try the following code:

```
x = -5:0.1:5;
y1 = x.^2;
y2 = x.^3
figure()
hold on
plot(x, y1)
plot(x, y2)
```

5 Saving/loading and .m files

5.1 save/load

To save the data you have in your workspace, use `save('filename')`. This will save all the variables in your workspace. If you want to save only some of them, use `save('filename', 'variable name 1', 'variable name 2')` and so on. Try saving the variables x and y1 in your workspace into a file named abc. Note that you need to be in a directory that you have permission to save to. Change the directory you are in using the current folder panel on the left of the command window.

To load the variables you have saved from the file, use `load('filename')` However, before doing so, let's clean our workspace using the following command: `clear all`

This clears all the variables saved in your workspace.

Now try `load('abc')`

You should now see the x and y1 vectors back in your workspace!

5.2 .m files

What happens when you wish to save your code? This is where `.m` files come into play. You can create a new `.m` file by clicking on the new script button on the upper left corner (highlighted with a red border in Figure 1), or use the keyboard shortcut `ctrl+N`.

To save your file, simply use `file->save` or `ctrl+S`.

Create a `.m` file, and copy your code for the last two line from the command history and paste it into the new `.m` file. Then add the line

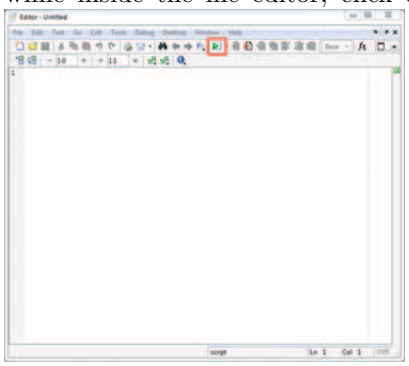
```
clc; clear all; close all;
```

to the top of the file. What do these lines do, and why is this a good idea when executing a script?

Executing a .m file

To run a `.m` file, you can do one of the following:

- while inside the file editor, click on the Run button, outlined in red in the following screen shot.



- while inside the file editor, use the keyboard shortcut `F5` or `ctrl+Enter`
- while inside the Command window, type the name of the `.m` file, *without the .m*

5.3 comments

To comment in Matlab, use the `%` symbol.

```
% This is a comment
```

Comments can help you remember what your code is doing in plain english when you come back to it later.

6 Exercises

Now that you've got a handle of the basics of Matlab, complete the following exercises to get checked off!

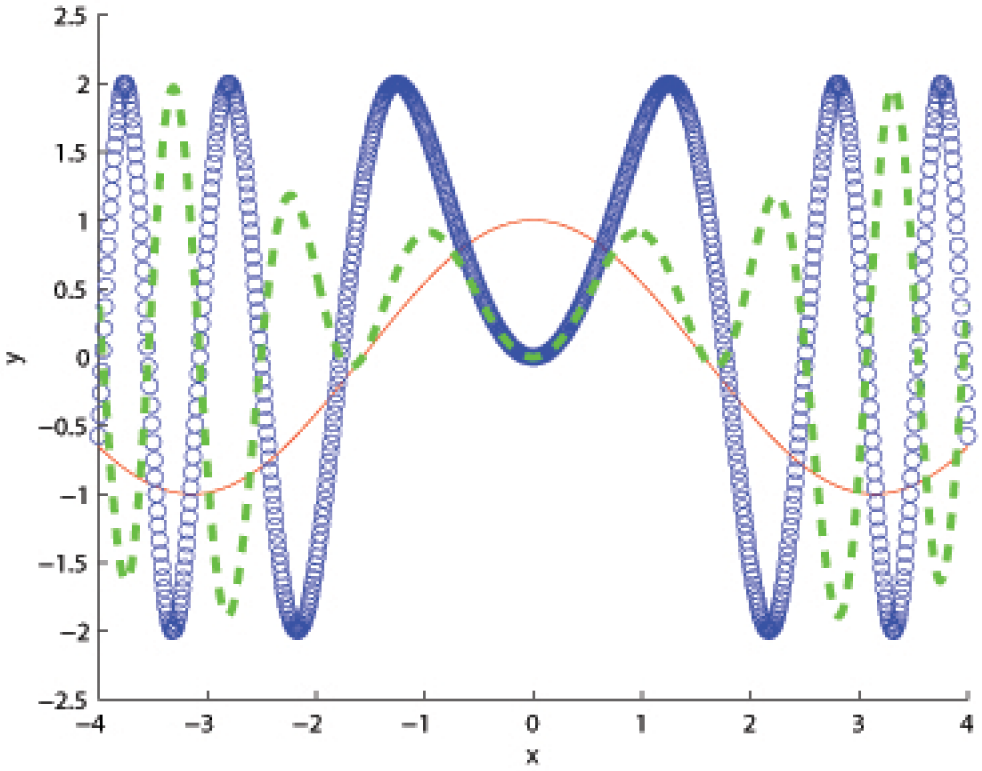
1. Create a `.m` file, save it in the Lab 01 folder and name it `exercises.m`
2. Why is it a good idea to put the commands `clc; clear; close all;` in your script?
3. Generate an array (row vector), call it A, with 10 entries: 3, 1, 4, 1, 5, 9, 2, 6, 5, 3
4. Generate two new arrays, `EvenEntries` and `OddEntries`, using the array A, such that `EvenEntries` has the entries with even indices in array A and `OddEntries` has the entries with odd indices in array A. Do this without using any `for`/`while` loops.
5. Create the following matrix
$$B = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix}$$
6. Using matrix B, create the following matrices:
$$C = \begin{pmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \end{pmatrix}$$

$$D = \begin{pmatrix} 2 & 4 \\ 6 & 8 \\ 10 & 12 \\ 14 & 16 \end{pmatrix}$$

$$E = \begin{pmatrix} 6 & 7 & 8 \\ 14 & 15 & 16 \end{pmatrix}$$

Again you should be able to do this without loops! (Hint: did you understand how the colon worked in the matrix indexing?)

7. Create the following plot.



The red line is $\cos(x)$, the blue plot is $2\sin(x^2)$, and the green plot is $2\cos(x)\sin(x^2)$. Because it is incredibly difficult for you to actually count the number of blue circles to figure out the step size, you have to trust me when I tell you that there are 100 circles between x=0 and x=1 (what does this mean? What's the step size?).

How can you figure out how to change the color and style of the plot? (Asking your GSI would be a way, but is there a helpful function that you can use to look up the answer right within Matlab?)