Week 1: Matlab Basics
APPM 2460

1 Introduction

In this lab we'll get acquainted with the basics of Matlab. This will be review if you've done any sort of programming before. The goal here is to get everyone on the same page with regards to some basic concepts.

2 The Matlab Environment

First we'll launch Matlab. Once it is up and running, you will see three windows; the **Command Window**, the **Current Folder** window, and the **Workspace** window. For now, we'll focus on the command window.

2.1 The Command Window

The **command window** (sometimes referred to as the command line) is where you enter commands directly. *All commands are typed in this window at the Matlab prompt '* $\gg$*'.*

- When you first launch Matlab, you will automatically go to this window
- Usually, the command line is best for single-line commands.

Example: To make sense of how it works, type the following:

- `disp('Hello world!')`
- `disp(sqrt(2))`
  - The `disp` command tells Matlab to display something. This something can be a string (sequence of alphanumeric characters, such as a word or sentence) or a number, among other things.
- `x = 2*6`
  - After entering the command (`x=2*6`), you should see a variable `x` appear in your workspace. This means you've **assigned** a value to the variable, a concept we'll describe in the next section.
- `x = 5`
  - After entering the command (`x=5`), you should see a variable `x` in the workspace changes value. The variable x was "overwritten" or "reassigned."
- `X = 4*8`
  - After entering the command (`X = 4*8`), you should see a variable `X` appear in your workspace but that it did not replace `x`. This demonstrates that *Matlab is case sensitive.*

2.2 The Workspace

The **workspace** is defined as the collection of variables you have assigned. As you accumulate variables or structures, you will see these listed alphabetically. The workspace window allows you to check out what variables are currently in use so that you do not try to reuse a variable that has been assigned a different value.

Your workspace is emptied each time you start Matlab. You can also force Matlab to delete the variables in the workspace using the command `clear all`.

- Note: It is best to clear the workspace at the beginning of each script so we know we are not accidentally using leftover values from old work.
- The workspace also lists the size and type of each variable.

2.3 The Current Folder Window

The **current folder** tells you which folder you are working in (that is, where you are saving files or where Matlab will look for files that you want to use). If you try to use a file that is not in the current folder, Matlab will not be able to open/run the file and will give an error message.

The current folder can be used to navigate between folders or to run files that are listed.

2.4 Writing Scripts

You may want to write multiple lines worth of commands. For this, we use a **script**. We can make a new script by clicking the **New Script** button on the top left of the Matlab window.

A new window should have opened, named **Editor**. It also shows the name of the script you're editing, which right now is likely "Untitled." Type the following lines into the editor window:


```
clear all
x = 2;
disp(x)
x = x+10;
disp(x)
disp('Go Buffs')
```

Now, hit the button labeled **Run** in the bar above the editor window. It will prompt you to save the script, so give it name and hit save. Then, the script should run.

You'll notice that the exact same thing happened as would have if you had entered each line of this in order. To reiterate:

A script is just a sequence of commands that are executed in order.

Let's make some changes to the script and run it again:

- First, remove the lines that start with `disp`, and run the script. You should see no output, although if you look in your workspace, you should see that, indeed, our variable `x` has the value 12.
- Now, remove the semicolons, and run the script. You should see output for each of the lines that begin with `x = ...`. We say that semicolons "suppress output," and this is usually what we want. As we change the value of things (in this case, the variable `x`) we don't want to see the value each time.

Scripts are a sequence of commands. Put a semicolon at the end of every line that has an equals sign in it so that Matlab doesn't print stuff you don't want.

2.5 Publishing Scripts in Matlab

In this class you will need to submit scripts of your work for homework. However, you will also need to demonstrate that the submitted script correctly executes a command or performs a calculation. To do this, you will "publish" the scripts that you write. When Matlab publishes a script, it will run the code and print the results of any commands or calculations at the bottom of the script (or at the bottom of that section of the script).

To publish a script, follow these steps:

1. At the top of the script, there are three tabs ("EDITOR", "PUBLISH", and "VIEW"). Click the middle tab "PUBLISH".
2. A new menu will appear under "PUBLISH." On the right-hand side of this menu, click the downward facing arrow under the file labeled "Publish"
3. Select "edit publishing options..."
4. Under "Output settings" change the "Output file format" from `html` to `pdf`. Also, delete `\html` from the "Output folder" name.
5. In the bottom right corner, click "Publish."

There are a few things that you need to keep in mind when you publish your Matlab scripts.

- You cannot publish (i) `m`-file functions or (ii) anything that requires a separate input.
- Matlab will overwrite any pdf with the same name as the `m`-file that you are publishing. If you do not want a file overwritten, change its name *before* publishing the `m`-file.
- If you would like to comment out several consecutive lines of code (or if you would like to include a multi-line comment in the code), use the following syntax:

```
%{
    The code you want to comment out
    or the comment you want to write.
%}
```

- Often your code will include multiple parts. It can be helpful to print the result(s) of each chunk of code directly beneath those lines of code when publishing the script. This is accomplished by divide the code into sections using the command `%%`:

```
Section 1 of the code here

%%

Section 2 of the code here
```


3 Variables and Assignment

3.1 Pseudo-Code

Programming a computer can seem daunting, and overwhelming when it's not working. The key to understanding what the computer is doing is to use *pseudo-code*. This is a way to write, by hand, what you want the computer to do at each step. Then, you can follow the instructions yourself and see what the computer should know at each step.

3.2 Assignment of Variables

Matlab does not think of variables the same way that we people do in math classes. You can think of Matlab as storing variables inside a filing cabinet, with an note card for each variable. If I enter the command `x = 2`, Matlab will create a new note card in the filing cabinet, label it `x`, and write the value 2 on the back.

When we want to save a value to a variable (write a number on the note card), that is called *assigning* the value. In the example above, we are assigning the value 2 to the variable `x`. In pseud-code, we will sometimes use a backwards arrow ($\leftarrow$) to denote assignment. We use $=$ (in pseud-code) for something else, which we will cover later in the semester. The arrow helps us to remember that Matlab starts on the right side of the assignment, computes a value, then saves it to the variable on the left. Consider this line of pseudo code:

$$\texttt{x} \leftarrow 2 + 3$$

Here, the program starts on the right, adds 2 and 3 together, gets 5, then writes 5 to (the note card associated with) `x`.

3.3 Assignment is not Equality

In math classes, an equation such as $x = 2 + y$ makes perfect sense: x depends on y, and in particular is equal to $2 + y$. However, to Matlab, this is gibberish. Try entering `x = 2+y` in the command window; you will get an error. This is because Matlab works in terms of concrete values, and it does not know what you mean by `y`. The first thing Matlab will do is go to the variable (note card) `y`, and retrieve the value, which it will then add to 2. However, there is no such variable! We need to tell Matlab what `y` is first. These lines of pseudo-code do make sense:

$$y \leftarrow 3$$
$$x \leftarrow 2 + y$$

Now, in the second line, the program would go the variable `y`, retrieve the value 3, add it to 2, get 5, then store it in the variable `x`.

Let's look at another situation where Matlab treats equals signs differently than we would in class. In a math class, $x = x + 2$ is an impossibility. However, Matlab can handle this expression:

$$x \leftarrow 3$$
$$x \leftarrow x + 2$$

Let's go through this one step at a time:

- In the first line, Matlab creates a variable `x` and stores the value 3 in it.

(Remember that we always start on the right side of an assignment.)

- In the second line, Matlab retrieves the value of `x` (3), adds this value (3) to 2, and gets 5. Then, it stores it in the variable on the left side of the assignment, which happens to be `x`.
- There is no issue with the fact that we used `x` on both sides of the assignment. The final value stored in the variable `x` simply changed from 3 to 5.

4 Vectors

Since this class deals with vectors, we will certainly want to be able to work with them in Matlab. Nearly everything we do this semester will use vectors.

Using the note card visualization, a vector variable will have a row of numbers on the back of the note card. We use brackets to tell Matlab that we are creating a vector, with commas between each of the numbers, known as *elements*. For example, I could create a vector of the numbers 1 through 4 using the following pseudo-code:

$$\texttt{x} \leftarrow [1, 2, 3, 4]$$

What if I wanted to create a vector of the numbers between 1 and 1000, or 1000000? This method would require that I spend a LOT of time typing. Instead, we can create a whole vector all at once using the following template:

$$\texttt{x} \leftarrow \text{first number} : \text{increment} : \text{last number}.$$

This is often stated in more succinct terms:

$$\texttt{x} \leftarrow \text{start} : \text{step} : \text{stop}.$$

So, for the above example, the following pseudo-code generates the vector $[1, 2, 3, 4]$:

$$\texttt{x} \leftarrow 1 : 1 : 4$$

The first number in the vector is 1, we want to increment by 1, and the last number is 4. If I wanted the vector $[2, 4, 6, 8]$ I would use

$$\texttt{x} \leftarrow 2 : 2 : 8$$

If I wanted a vector of all of the multiples of 10 under 10000, I could type:

$$\texttt{x} \leftarrow 0 : 10 : 9999$$

Give it a try yourself! Enter the command `x = 2:2:8` into the command window and see what happens. What if we do `x = 2:3:8`?

4.1 Indexing a vector

Now that we can create a vector, we want to be able to use, and manipulate them. Suppose I have a vector which contains the temperatures in Boulder on Jan 14 for a period of 5 years. It might look something like this, where the first element is last year's temperature, and the last is from 2010:

$$\texttt{temps} \leftarrow [30, 40, 24, 55, 18]$$

Now, what if I wanted to use the temperature from 2011? Then we would want to *index* the vector. We do this using parenthesis next to the variable name of the vector, and the number corresponding to the location of the element we want. 2011 is in the 4th element, so we can type:

$$\texttt{2011temp} \leftarrow \texttt{temps(4)}$$

Now the variable `2011temp` would contain the value 55.

What if we wanted to go in reverse? That is, instead of retrieving one of the elements, we wanted to change the temperature we have recorded for 2012. We would want to assign a value to `temps(3)`, so it would go on the left side of the arrow, and the value we want to save on the right.

$$\texttt{temps(3)} \leftarrow 51$$

What if we wanted to change the temperatures for both 2012 and 2011 at once? We can use the colon once again. When indexing, the colon reads like *through*, so that `1:6` is one through six. Then `x(1:3)` would index the first through the third entries of the vector `x`. Thus, to change the temperatures for 2011 and 2012, we could type

```
temps(2:3) = [30, 33]
```

Then, the vector temps would be $[30, 30, 33, 55, 18]$. Try this yourself, and make sure that the value of `temps` is what you expect!

APPM 2460 Homework Week 1

Submit a published pdf of your script and any other supporting code needed to solve the following problem to Canvas by Monday, August 31 at 11:59 p.m.

1. This problem is to familiarize you with Matlab. Divide a Matlab script into sections using the command `%%`. Make one section per calculation and use a comment to label each section of the script `(a)-(j)`. In each section, fill in the Matlab command that performs the desired operation. Do not supress the outputs of these calculations.

Part	Desired Operation
(a)	$x \leftarrow 8$
(b)	$y \leftarrow x/2$
(c)	$x \leftarrow x * y$
(d)	$z \leftarrow x - y * 5$
(e)	$\text{vec} \leftarrow [1, 4, 3, 8]$
(f)	Set A equal to the third element of "vec"
(g)	$A \leftarrow A + \text{vec}(4)/\text{vec}(2)$
(h)	$\text{newvec} \leftarrow y : z : x$
(i)	$\text{vec}(1) \leftarrow \text{newvec}(2)$
(j)	$\text{vec}(2:4) \leftarrow [x, y, A]$

2. (In the same script, make a new section.) The following questions will allow me to get to know you better. Answer each of the following questions, writing the answers into the script as comments.

   (a) What year are you at CU? What is your major?

   (b) Do you have experience with Matlab or any other programming language?

   (c) What do you hope to learn from this class?