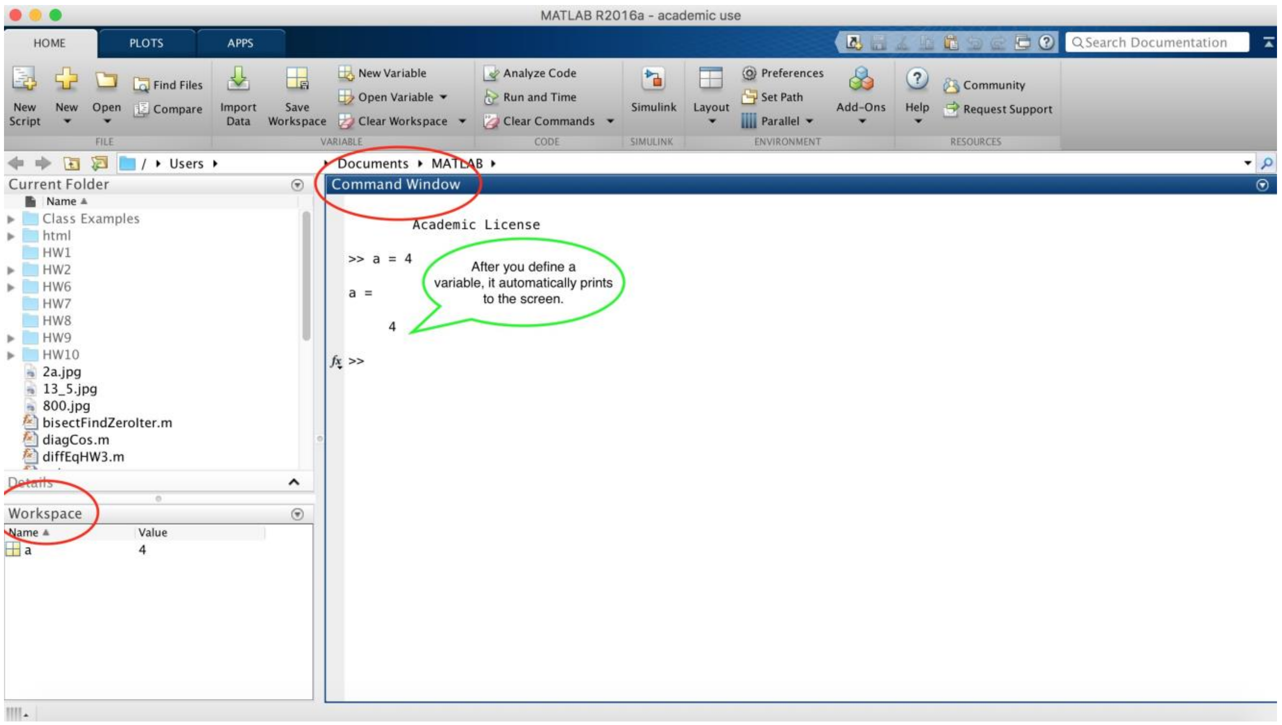
BIS20Q Lab 1

Introduction to MATLAB

1. First Steps—The Environment and the Command Line

MATLAB is a scientific programming language with an easy-to-use interface. The name stands for "matrix laboratory," because the program is designed to handle matrices quickly and efficiently with many built-in functions for performing common linear-algebra operations. MATLAB is frequently used as a tool in many areas of biological research, and it is a great starter language if you are new to programming.



```
  Academic License

>> a = 4

a =

     4

fx >>
```

When you open MATLAB for the first time, you are presented with the MATLAB *environment*, which has three re-sizable windows, the most important of which is the Command Window. This is where you define variables, call functions, or anything else you would like to do. Try each of the following:

```
>> 2+5
>> 2/5
>> 2*5
>> 2+5*3
>> 2^5
```

Variables in a programming language are similar to mathematical variables—in MATLAB the variables are vectors or matrices, which will turn out to be useful as we move through these labs. Let's define a variable by entering the following command in the Command Window:

```
>> a = 4
```

You might notice that variable a now shows up in the Workspace window. You can reassign variables and perform arithmetic in the command line. Note that this will overwrite the previous value of a. For example:

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

A useful trick is to add a semicolon at the end of a line in order to suppress its output (this will be useful when you are working with a huge array of numbers, for example). Try multiplying two variables together as shown below—what is the value of c?

```
>> b = 7;
>> c = a * b;
>> c
```

49

Two other useful commands are clc and clear. Try both and explain what they do below (and be careful not to confuse them down the line!):

clc clears the command window (visually). clear clears MATLAB's memory, deleting any variables currently stored in memory. You definitely don't want to type clear when you meant to use clc! Additionally, you can clear specific variables, leaving the rest of the memory with commands such as "clear a" (that will only delete the variable a).

2. Mathematical Functions

There are many mathematical functions that are already included in MATLAB. Try each of the following to execute a square root, an exponential (e^2), and the sine function:

```
>> sqrt(2)
>> exp(2)
>> sin(90)
```

Note that the sine of 90 may not have given you the answer you expected. Try each of the following to get more information about the sine function in MATLAB, and confirm why it didn't.

```
>> help sin
>> doc sin
```

If you needed to figure out how to use a new function that were not familiar with, which of the two above commands would you use and why?

You should use whichever one you are more comfortable with. I find the interface with doc to be easier to read and navigate.

3. Vectors Store Multiple Values in a Single Variable

A really simple definition of a *vector* is that it is a list of numbers. To define the vector (2, 7, 3) in MATLAB, write:

```
>> my_vector = [2, 7, 3];
```

Since vectors are used so often in MATLAB, there are a number of tricks that are available to easily define common types. For example, suppose that you wanted a variable that contains all of the times (in seconds) between zero and ten minutes at ten-second intervals. This might correspond to an experimental data set that you have. The MATLAB syntax for this is:

```
>> timeList = (first number:increment:last number);
```

Go ahead and create this vector. This vector is pretty long, so checking it by hand could be tedious. Instead, let's check the first five values that it contains, and let's define those five elements to be a new vector:

```
>> five_elements = timeList (0:5)
```

Did that work? If not, what happened?

No. It triggers an error.

Accessing one or more elements in a vector as you did in your last command is called "indexing." Although it's fairly common among different programming languages that vectors be "zero-indexed" (i.e., the first element of my vector is my_vector(0)), MATLAB is "one- indexed." What is the correct command to define five_elements as the first five elements of the vector times?

five_elements = times(1:5)

Just as you did with the simple variables earlier, you can perform arithmetic on vector variables, too. This can be extremely useful, but you have to be a bit more careful to make sure that the computer is carrying out the operations that you think it is!

Start by defining two vectors of your choosing, each containing three elements. What does "vector1 + vector2" do?

It creates a new vector whose elements are the sum of the corresponding elements in vector1 and vector2.

4. Using vectors for math and plotting can be more efficient

Supposing you have data for miles traveled and gallons of gasoline used for ten different cars on two different days, and you would like to compute the average miles per gallon for each car. You can copy and paste the lines below to create variables in the workspace containing this data (the data for the cars are listed in the same order for each variable):

```
>> miles1=[153 177 89 144 99 101 210 93 134 111];  % milage on day 1

>> miles2=[109 111 86 95 96 139 136 94 235 178];  % mileage on day 2

>> gallons1=[5.1 6.8 5.5 7.1 3.9 4.5 6.8 5.1 3.3 5.2];  % gallons used on day 1

>> gallons2=[3.8 4.4 5.2 4.8 3.7 6.3 4.4 5.1 5.9 8.1];  % gallons used on day 2
```

You could compute the miles per gallon for each car by adding its two mileage measurements and dividing by the sum of the gallons measurements, but doing this one-by-one would be very tedious. Fortunately, vector computation in MATLAB makes this much easier. The key is to use *elementwise arithmetic*, in which you add a "." in front of the arithmetic operator. For example, what does "miles1 ./ gallons1" do?
It creates a new vector in which each element is the corresponding element in miles1 divided by the corresponding element in gallons1.

Write a single command that would return the miles per gallon for each car as a vector (using the data from both days combined).
(miles1 + miles2) ./ (gallons1 + gallons2)

Which car (provide its number from the list) is the most efficient?
The 9th

Now try using functions on vectors, and making our first graphs.

```
>>  x = 0:0.1:10;

>>  y = sin(x);

>>  plot(x,y, 'go');

>>  hold on;

>>  plot(x,sin(2*x), 'blue');

>>  plot(x,sin(x+1), 'red');
```

What was the purpose of the 4th line (hold on)?
It causes the following plots to be overlaid on top of the first one, instead of clearing the previous

plot and replacing it with the latest one.

5. Matrices are 2 (or higher) dimensional arrays of numbers

A matrix is just like a vector, except that it is of higher dimension. In fact, you can also think of vectors as just a special 1-dimensional case of matrices. We can define and access the elements of a matrix just as we do for vectors, except that we additionally use a semicolon to indicate the end of a row in the matrix. To define a simple matrix with three rows, write:

```
>>  mat  =  [2   7   3; 1  4  6;  7  8  9];
```

Try each of the following, and note the results:

```
>>  mat + 1

>>  mat + mat

>>  mat * mat

>>  mat .* mat
```

Why do the last two lines give different results?
The first command does matrix multiplication, and the second does elementwise multiplication.

You can create a new matrix of a desired size in one step, using built-in functions. Try each of the following:

```
>>  mat  =  ones(3,4)
>>  mat  =  ones(5,5)
```

You can also retrieve or change the elements of a matrix one element at a time:

```
>>  mat(3,4)=7
```

One row at a time:

```
>>  mat(3,:)=3
>>  mat(3,:)=1:5
```

Or one column at a time:

```
>>  mat(:,2)=2
>>  mat(:,4)=(1:5)'
```

To understand the last command, try the following at the command line:

```
>> (1:10)
>> (1:10) '
```

What is the difference between the outputs of these two commands, and why did we need to include the apostrophe in our assignment to column 4 of mat above?
The apostrophe turns a row vector into a column vector. More generally, it transposes a matrix,

which means that it flips the rows and columns.

6. Loops and Logic

Up until this point, MATLAB has probably just seemed like an expensive calculator, so we're going to introduce some slightly more advanced functionality, which you can still enter by hand in the command line. *Loops* and *logical statements* are the most basic building blocks of any complex program. We'll start with "for" loops.

"For" loops are the right choice any time you want the computer to perform some operation repeatedly— and you tell the program exactly how many times you would like that operation to be carried out. Suppose—for a very mathematical example—that you needed to compute the sum

$$S = \sum_{n=1}^{10} \frac{1}{n}$$

The following "for" loop handles this easily—implement it from the command line using the steps below. In the spaces provided under each line, explain what you think the preceding command is doing.

```
>> S = 0;
```

This step initializes the value of S to zero.

```
>> for n = 1:10
```

This step defines the variable n as the counter variable for the loop, and defines that the loop will iterate for values of n beginning at 1 and stepping up by steps of 1 to a final value of n=10.

```
S = S + 1 / n;
```

In each iteration of the loop, the value of S will be increased by 1/n (using the value of n for that loop iteration).

```
end
```

This signals the end of the loop. The program will then jump back up to the "for" line and advance to the next value for n, unless the iterations for all values of n have been completed.

What was the final value of S?

2.929

In the third command, why is the semicolon important?

If the semicolon was not there, the value of S would be printed to the command value for each step in the loop. This would clutter up the command window.

Something that you may have noticed in this third command is that a variable S was re-assigned using itself! This is allowed in MATLAB (although it's sometimes not allowed in other programming languages).

An "if" statement is a simple example of a logical statement, and it allows you to create a condition under which a certain operation occurs. Consider the following command:

```
>> 2 > 5
```

This returns a "logical"-type variable with value 0. Logical variables are an example of a binary variable, which can take on one of two values—either one or zero, where 0 means false and 1 means true.

Now, you're going to analyze a more complicated set of commands. Here are a few pieces of useful information: "=="" means "is equal to", "||" means "or", and "else" can be translated to "otherwise."

```
>> a = 1;
>> if (a == 2 || a > 3)
>> b = 1;
>> else b = 2;
>> end
```

What is the final value of b, and why?
The final value is 2, because the statement (a==2 || a>3) was false. For this reason, the "else" block was executed instead of the "if" block.

7. Putting it together

Using the for loops and commands that you learned about above, generate a new variable "bigmat" storing a 10 x 10 matrix, in which the first row contains the numbers from 1 to 10, the second row contains the numbers from 2 to 11, and so on.

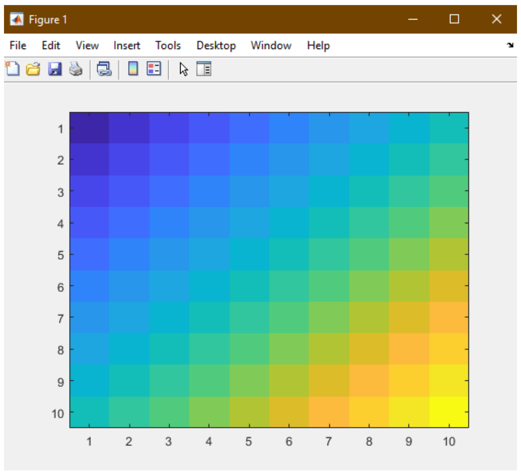
Copy and paste the commands you used below.

```
bigmat = zeros(10,10); for i=1:10; bigmat(i,:)=(1:10)+(i-1); end
```

(note that other answers could also work. You could write a separate command for filling in each row, but that is much more tedious than writing a for loop. You could also have written "bigmat(i,:)=i:(i+9);" for line inside the for loop, or several other equivalent expressions).

You can use the "imagesc" function in Matlab to get a visual picture of the matrix. Use the function and paste a screenshot image of the result below:

```
>> imagesc(bigmat)
```



Finally, you would like to create a 10 x 10 logical matrix in which each value is true if the corresponding value in bigmat is greater than or equal to 10. **You should be able to do this in a single line, by performing an operation on the matrix bigmat**. You can store the resulting logical matrix in a variable called "triangle." Paste your code below:

```
triangle = (bigmat >= 10);
```

Use the function imagesc to get a visual picture of the matrix, and paste a screenshot of the result below.

```
>> imagesc(triangle)
```

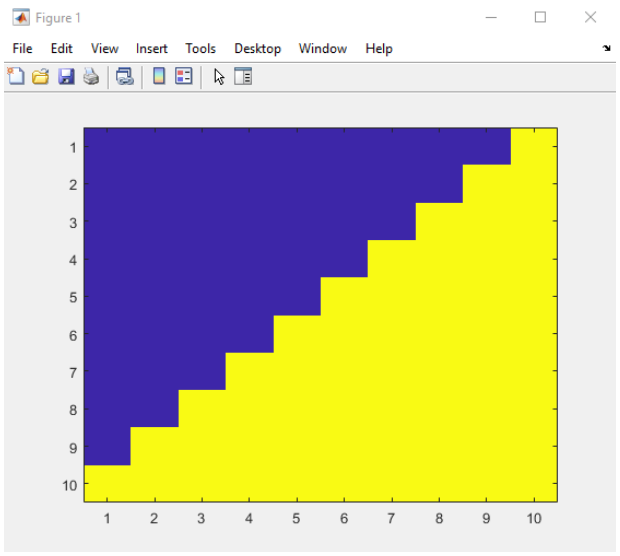


Figure 1
File Edit View Insert Tools Desktop Window Help
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10