1. Motivation

MATLAB is a ubiquitous computing environment used by the scientific community. It is designed to manipulate, visualize and organize large amounts of data, making it especially powerful for the following tasks:

- Matrix Manipulation
- Plotting and gathering statistics about data
- Interfacing between PC and various electronic equipment, such as oscilloscopes

In 6.02 and much of the field of communications, all computation is in the digital regime. Thus, we can abstract any transmission as a vector or array of values. From this document, I hope to show that MATLAB is an intuitive and powerful way to consider problems in communications, signal processing, and other fields touched by 6.02. Much of the lab work in this course will be done in MATLAB so it is imperative that you become comfortable with the program and understand how to perform many common tasks.

2. Getting and Running MATLAB

MATLAB can be accessed by any computer running Athena. Moreover, a licensed student version can be downloaded on the MIT IS&T website at [http://web.mit.edu/ist/topics/math/](http://web.mit.edu/ist/topics/math/).

Figure 1 illustrates what MATLAB looks like when it is first opened. Note that this would not be the case on Athena machines. Instead, a simple console window will be opened instead. To reach the more user-friendly desktop view, simply enter the command `desktop` at the console.

The first thing to note when opening MATLAB is the little bar at the very top indicating the Current Directory. Although MATLAB can easily access scripts, data files or media anywhere on the PC, it is convenient to set the current directory to a stable location that you have permissions to. In the lab computers, it would be ideal to set this to your Athena home directory.

Now let’s look at the desktop and see what it has to offer.
The largest window (1) is called the *Command Window*. This is where you issue MATLAB commands and perform computations. The top left window (2) is the *Workspace*, which shows all variables initiated by the user. By pressing the above tabs, you can also open the *Current Directory Window*. This will show what files are in the current directory and allows you to open, rename or delete files. Finally, the bottom left window (3) is the *Command History*, making it easy to recall commands you have used previously.

As with most environments, there are other options and customizations that make the MATLAB desktop easier to navigate. Take a few minutes to explore it.

### 3. Creating and Manipulating Variables

#### 3.1 Creating variables, vectors and arrays

Since it is mainly used for scientific computation, MATLAB makes it very easy to create variables. In fact, MATLAB is dynamically typed, meaning you do not have to declare a specific type when creating variables.

In the command window, you should see `>>`. You can enter a command at this point. For example, I will do the following:

\[ x = 5 \]

You will immediately see that the console will echo the result.
Moreover, the variable x will now appear in the workspace and the command history. Now consider the following commands and the console response (note that the >> indicate a command and are automatically included by the console. Don’t enter the >> yourself):

```
>> x = 5
x =
   5
>> y = x
y =
   5
>> y = '6.02';
```

There are a few things to note in these three simple commands. Firstly, you can initialize variables to functions of other variables. By letting y = x, I did indeed set y = 5. Secondly, you can create strings by placing single quotation marks around the text, like when I set y = ‘6.02.’ Note that y dynamically changed from being an int type to a string type. MATLAB takes care of this so we do not have to worry about it. Finally, to suppress the echo from the console, you can add a semicolon after the command.

Creating vectors and matrices in MATLAB is just as easy. To make a vector, one can type the command

```
>> A = [6 0 2]
A =
   6   0   2
```

One can also make vectors of consecutive values easily by inputting

```
>> B = 3:6
B =
   3   4   5   6
>> C = 0:5:20
C =
   0   5  10  15  20
```
The leftmost and rightmost values indicate the upper and lower bounds of the vector. The optional step value indicates how much to increment the next term.

Finally, we can make matrices in the same fashion as vectors, except we need a semicolon to indicate when to move onto the next row. An example is given below:

```
>> D = [3 4 1; 4 1 8; 9 6 5]
```

```
D =
    3     4     1
    4     1     8
    9     6     5
```

Taking everything we have learned thus far, we can make vectors and arrays by concatenating other variables. Try the following set of commands and see if they work as expected.

```
A = [5 2:5]
B = [0 0 A]
C = [2; 3; 5; 9]
M = [A; 5:2:13; 1 2 3 4 5]
N = [M [99; 98; 97]]
```

MATLAB also allows you to consider subsets of a vector or matrix. Try the following examples to get a flavor of what you can do.

```
A(1)
A(1:3)
B(4,1)
C(1,4)
M(1,3)
M(1:3,2:3)
M(:,1)
M(12)
N(1,:)
```

Note that only one parameter is necessary, even for matrices. For this case, indices increase down the column and then to the next column. Also, the colon in MATLAB syntax indicates all the values of that particular row or column.

3.2 Mathematical Manipulation

Now that we know how to create variables, we can now perform computations on them. In most common programming languages, one usually has to create mathematical operators from scratch. For example, a factorial is commonly implemented using a recursive function.

In MATLAB, however, a large number of operators are already created for you. They are usually implemented as functions. Libraries of these functions are called toolboxes and oftentimes have very specific purposes. Some examples include the Signal Processing Toolbox, Image Processing Toolbox, and Neural Network Toolbox. Luckily,
these toolboxes are simple to install and use. The Athena machines and student license from IS&T should have almost everything you would ever need.

To start, you can do basic arithmetic on the console. Try out the following commands:

\[
\begin{align*}
    x &= 5+2 \\
    y &= 2/4 + 5/3 + x \\
    4*y + 2^2 - 4/3*2
\end{align*}
\]

Note that there is some specific ordering in which operations are being performed. Brackets can be used to make sure that you get the ordering that you want.

More complex operators are available but must be implemented as functions. Thus, you must enter in parameters when calling them. You can also access the MATLAB Help to get more information about the function. Note that oftentimes not all parameters need to be specified. Some examples are given below.

```
help sin
xsin = sin(x)
p = exp(x)*log(y)*pow(xsin, 4);
sin(log(200))
```

MATLAB’s true power is its power to handle vectors and matrices as variables. Thus, it is just as easy to operate on vectors as it is for scalars. Try the following commands:

```
x = [1:10]
y = [2:2:20];
x + y
sin(y)
ymax = max(y)
fliplr(y)
```

Some nuances exist in the multiplication and division of matrices and vectors. Those who have taken linear algebra will know that matrix multiplication is not the same as point-by-point multiplication of the terms. In MATLAB, one can do matrix (or vector) multiplication by using the * character. However, if one wants to do point-by-point multiplication (meaning multiplying the corresponding values in each matrix), then you should add a period right before the *. The following commands motivate this point:

```
X = [1 2; 4 5];
Y = [1 0; 0 1];
Z1 = X*Y;
Z2 = X.*Y;
V1 = X^2;
V2 = X.^2;
```

Try some more commands and get comfortable with creating variables. Also, think of different operators that you could apply and use the MATLAB help to find out if a function exists for it.
4. Making Plots in MATLAB

Besides being great for manipulating data, MATLAB is a powerful way to display results. Plotting is very easy, but there is also much functionality and configurability in the environment. A simple example is given:

```matlab
t = [-3:.001:3];
y = sin(5*t);
plot(t,y);
```

Note that I put a semicolon after initializing the time vector. This is because I created a very long vector (6001 terms) and did not want to display them all on the console. The plot should look like this:

![Figure 2: A plot of a sin wave](image)

This graph can be exported to many image formats to fit your needs. Other functions can allow you to customize this image. Some examples are listed below.

```matlab
plot(t, y, 'r:');
axis([-2 2 0 1]);
title('This is a sine curve');
xlabel('time');
ylabel('amplitude');
```

Use the MATLAB help documentation, especially on the plot function. There are a lot of options that can be used to make plots look nicer. It is also possible to add several curves into the same graph. One can do the following:

```matlab
t = [-3:.001:3];
x = sin(5*t);
y = cos(3*t);
plot(t, x, t, y);
```
An alternative is to use the hold command:

```
  t = [-3:.001:3];
  x = sin(5*t);
  y = cos(3*t);
  plot(t,x, 'k:');
  hold on
  plot(t, y, 'r-');
  hold off
```

You can also draw several graphs (referred to in MATLAB jargon as figures). To add a new figure, use the `figure` command. You can also add a number parameter such as `figure(3)` if you want to keep track of which figures you are accessing. A sample chunk of code is given to illustrate working with multiple figures.

```
  t = [-3:.001:3];
  x = sin(5*t);
  plot(t,x);
  figure;
  y = cos(3*t);
  plot(t,y);
  figure(50);
  z = 3*t;
  plot(t, z);
```

### 5. Writing Scripts and Functions for MATLAB

Although one can use the console to type commands, I find it often helpful to write MATLAB scripts and functions. A MATLAB script is simply a set of commands stored in a .m file, which can be edited in any text editor. Upon calling the script, MATLAB reads the commands one by one until the end of the file. Scripts are very easy to implement and makes it easier to troubleshoot problems so I recommend getting into a good habit of using them regularly. To create a script, go to the desired directory and type in `edit <scriptname>.m`. Click yes on the prompt and the MATLAB editor will open up. You can type any set of commands, separating each command with a line break.

You can then run your MATLAB script by either pressing F5 in the MATLAB editor or going into the console window and running the command `<scriptname>`.

You can also create functions in the MATLAB editor. Again, a function is a set of MATLAB commands that will be run in order. However, one can specify input parameters and/or output variables in a script. Moreover, unlike a script, the variables used in a function are local and will not show in the workspace. This is an effective way to black-box code that you might run many times.

The following is exemplifies the formatting needed for a MATLAB script:
function out = tempFunc( in1, in2 )

    y1 = in1 + in2; % this is a comment
    y2 = sin(in1+in2); % this is another comment
    y3 = 100 - 30*in2;
    out = max([in1 in2 y1 y2 y3]);

The input and output variables are optional. Note that the % denotes a comment. Anything following the % character is not run. It can be useful to document your work.

A function with parameters must be run in the console (or accessed by other scripts and functions). Again, you can simply type the function’s name as the command. To call the above function, just run

    y = tempFunc( 5, 3);

Through this course and others you will take involving MATLAB, you will be asked to write scripts and functions. Try to make a conscious effort to use them as much as possible. This will make it much easier to troubleshoot problems and help you modularize the code, which will become very important as the MATLAB tasks become more complex.

6. Some Helpful MATLAB Notes
   a. There are several MATLAB commands to clean the environment.
      • clc erases the command window.
      • clear all deletes all the variables in the workspace
      • close all closes all open figures
   b. Some useful constants are easily accessible in MATLAB, including
      • both j and i can be used to indicate imaginary numbers (i.e. 5*j)
      • pi is a preset variable
      • the mathematical constant e^n can be found using the command exp(n)
   c. One can also perform logical operations, giving binary outputs. This can be useful in many cases so it is worth trying out. An example is
      a = [2 0 3 0];
      y = a > 0
   d. Concepts that you learned in C or Java is also applicable in MATLAB. The help documentation can show you how to add for loops, while loops and case statements.
   e. For debugging purposes, especially when writing scripts or functions, it may be good to output some text to the console. Use the fprintf or display functions for this.
   f. Two very useful functions for making vectors and arrays are the zeros and ones functions. They will generate a sequence of zeros or ones depending on your input specifications.
7. One Big Example

Let's try and tie everything together by considering an example in image processing. Imagine we want to create a function that takes in an image, changes it to grayscale and makes it into a binary image (only black or white) that is set by some user-specified threshold. This is rather trivial, but allows us to consider some of the things we have learned.

Let's begin by designating the current directory to something that is stable and that you have access to. Download a picture (hopefully not too big) into that directory. Now let's create a function that performs the desired task. To begin, create a function. Do so by entering the command `edit setThres.m`.

In the MATLAB Editor, type the following:

```matlab
function setThres (im, thres)

    A = imread(im);  % imread reads the image and finds the RGB values, putting it into the 3D array A
    BW = rgb2gray(A); % change from rgb to grayscale
    figure(1)
    imshow(BW);       % take a look at the grayscale image

    OUT = (BW > thres); % set thres: if pixel > thres, then out = 1
    figure(2)
    imshow(OUT);      % take a look at the thresholded image
    imwrite(OUT,'out.jpg','jpg'); % now write to file

Now in the command window, type in following command

    setThres ('<filename>', <thres>)
```

where `<filename>` is the image you downloaded and `<thres>` is a value from 0 to 255. You should see two figures and a new jpg image with the output image. Be sure to include the single quotation marks around the filename to make it into a string.

8. Conclusion

In my previous teaching experience, I have found that MATLAB is a major stumbling block for students, who find it intimidating. However, MATLAB has a very gentle learning curve and can be a very useful tool in manipulating and visualizing data. This document is designed to introduce you to the program but is not an exhaustive overview. There are other resources on the web, including the surprisingly effective MathWorks documentation. Moreover, I and the other members of the teaching staff would be happy to discuss any questions you may have both in lab and during office hours. In the end, the best way to learn how to use MATLAB is through experience, which you will get a lot of in 6.02.