**DIP Lab: Introduction to MATLAB**

**Goal:**
In this lab we will become familiar with the MATLAB programming environment. You will learn the basics of MATLAB syntax and functionality. You will also write your own program to do simple manipulations to an image in MATLAB.

**Introduction:**
MATLAB is a mathematical computer package from The MathWorks. "MATLAB" stands for "Matrix Laboratory". MATLAB’s functions operate on matrices and vectors, as well as scalar numbers, so it is very useful for linear algebra applications. With it, you can perform any number of simple or complicated mathematical operations, including image processing. As a general-purpose programming language, it is easy to learn and use, but powerful enough for a wide range of science and engineering applications.

The best way to learn MATLAB is by using it, which is what we will be doing in this lab and throughout the semester. The built-in and online documentation is excellent and there is also a complete set of manuals for you to use. Please keep the manuals in the lab!

To access MATLAB, log in to one of the PCs and double-click on the MATLAB icon on the desktop (or the Windows Start Menu). The MATLAB environment will open. It is a window with several panes. The largest one is the command line. The others allow you to view the current directory, see what variables are in memory, etc. The menu bar at the top and the “Start” menu at the bottom of the window allow you to access many of MATLAB’s functions as well as help and tutorials.

For our purposes, MATLAB is ideal since an image is basically a matrix. Remember that digital images have been discretized and are composed of pixels that can be thought of as rectangular, 2-D arrays of small dots on the screen. Each dot has a particular location associated with it, its coordinates (expressed as a pair of numbers) in the rectangular array. Each of the dots also has a value associated with it (a number “stored” at that location) that specifies the brightness (or color) of that dot. These values (DN, or digital numbers) are the elements of the matrix. The matrix has a size, m x n, which represents the number of rows (m) and the number of columns (n). Each pixel also has a size related to the quantization, e.g. how many binary digits (0’s and 1’s) it takes to record the number stored. (Note that this is different than a physical or spatial size!) For example, to record grayscale between 0 and 255 (256 total possible values, in this case, shades of gray), you need to quantize the image at 8-bit \( (2^8 = 256) \). Thus, if you multiply the number of bits/pixel by the total number of pixels (m x n), you can figure out how much memory or disk space your image will fill.
Since we can treat our image as a matrix, a mathematical object, we can do all sorts of things to it. Essentially any matrix and scalar operation can be done. We can multiply, add, divide, subtract, etc. images. We can apply transformations (in shape, size, quantization, etc.). Digital images can be mathematically manipulated as we see fit.

**Instructions:**

1. Log in and start MATLAB. The first thing you should do when starting MATLAB is to navigate to your home directory on /geobase in order to set the correct working directory.

   In the following, commands to type into the MATLAB command line are in this font. Terminology you should understand the specific meaning are given in italics (see below and above)

   A note on help: If you type help followed by the name of a command, you will get a detailed description of the command, what it does, and how to use it. Note also that MATLAB is case-sensitive.

2. Create a matrix with the following line:

   \[
   A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
   \]

   Note that commas (,) separate elements and semicolons (;) separate rows.

3. Calculate the square of your matrix:

   \[
   A^{\text{square}} = A \times A
   \]

   Note that using \( \times \) (as opposed to simply \( \times \)) forces MATLAB to perform the operation element-by-element. Try the command with \( \times \) and see what happens. You will have performed a conventional matrix multiplication. I make a habit of always using \( \times \) unless I specifically want matrix multiplication. The same holds for division (\( ./ \) vs. \( / \) ) and for raising to a power (\( .^\wedge \) vs. \( ^\wedge \)). Addition (\( + \)) and subtraction (\( - \)) are always element-by-element operations.

4. As you can see, MATLAB prints the result of everything you do. This can slow things down, however, especially in a long program. You can suppress the output by placing a semicolon at the end of the line. If you type:

   \[
   B = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}
   \]

   the matrix \( B \) will not be printed. If you now type:

   \[B\]
the matrix will be printed.

5. Large matrices can be easily constructed without typing in every element. Below are some examples. Try them and see what they do. (Before doing so though, you will need to define the matrix dimensions, \( m \) and \( n \), which denote the number of rows and columns, respectively)

\[
C = \text{ones}( m , n ) \\
D = 0.5 .* \text{ones}( m , n ) \\
E = \text{zeros}( m , n )
\]

6. **Diagonal matrices** are also easy to create:

\[
F = \text{diag}( v , k )
\]

Before using this command, you need to define the vector \( v \) that you want on the diagonal of \( F \). Vector \( v \) should have at least 2 elements, and you may want to play with this value as you vary \( k \). You also need to specify \( k \) which tells MATLAB which diagonal to use: \( k = 0 \) is the main diagonal, \( k = 1 \) is 1 spot above, \( k = -1 \) is one spot below, etc. Note that MATLAB automatically adjusts the size of \( F \) size depending on how you specify \( k \).

7. To create vectors, you can type all the elements in manually, or use the tricks you learned in step 5. The following example will create a vector of 1’s, of \( m \times 1 \) dimensions:

\[
g = 0.9 .* \text{ones}( m , 1 )
\]

8. The vector, \( g \), you just made is a column vector. You can make it into a row vector by transposing it. Transposition can be done using the `'` operator or with `transpose` command:

\[
h = g'
\]

or

\[
h = \text{transpose}( g )
\]

This trick also works with matrices. Try it. (Note that we have been using lower case variable names for vectors and upper case for matrices.) If you use your matrix \( F \) (which you will note is still in MATLAB’s memory) you get a transposed matrix (e.g. \( F' \)) if vector \( v \) that you used to create it was not located on the main diagonal. So you may need to re-define \( k \) and recreate \( F \) make the result of transposing \( F \) more interesting.

9. Now, the power of MATLAB is that you can string together commands like this (MATLAB’s **built-in functions**) into pre-written **programs** (your own functions,
each stored in a simple text file) that you can invoke with a single command. You can write functions using MATLAB’s built-in text editor (which is very useful because it color-codes your program), or any other text editor. Moreover, these functions can be used on any platform that MATLAB runs on (Windows, Mac OS X, UNIX, etc.) Try writing the program below. It computes the mean and standard deviation of a series:

```matlab
function [ mean , stdev ] = stat( x )
% Comments look like this.
% They are not interpreted by MATLAB.
% They are handy for putting notes in your
% program.
% The "function" line above defines your program
% as a function that accepts input (x) and
% returns two outputs (mean and stdev).
% The name of the function is stat.
% You invoke the program by typing stat(x) at the
% command line. The program will then compute
% the mean and standard deviation of x.
% The next line returns the number of rows and
% columns in x.
[ m , n ] = size( x ) ;
% Below is a structure called an if-then-else
% statement. The double equal sign does not
% assign a value to m. It is a test of equality.
% Indenting makes your program easier to read.
if m == 1
    m = n ;
end
% What do you think these next two lines do?
mean = sum( x ) ./ m
stdev = sqrt( ( sum( ( x - mean ) .^ 2 ) ) ./ ( m - 1 ) )
% At this point the program stops and has
% returned two values, the mean and
% standard deviation.
```

When you are done typing it in, save it in your working directory as stat.m. But don't run it yet.

10. Again using MATLAB’s text editor, write a second program (see below) and call it quart.m. Save it, but don't run it yet.

```matlab
function [ quart1 , quart3 ] = quart( x )
% This function returns the first and
% third quartiles of the vector x.
ymed = median( x ) ;
```
% What do these lines do?

\[
y_1 = x( x \leq y_{\text{med}} ) ; \\
y_2 = x( x \geq y_{\text{med}} ) ; \\
quart_1 = \text{median}( y_1 ) \\
quart_3 = \text{median}( y_2 )
\]

11. In the MATLAB command window, type:

```
clear
```

This will clear all the variables currently in memory. If you type:

```
clc
```

you will clear the command line window so that you have a fresh slate. Don’t worry, though, there are ways to save variables in memory to disk, too!

12. Now create a series of numbers:

\[
x = [ 1 , 48 , 81 , 2 , 10 , 25 , 14 , 18 , 53 , 41 ]
\]

13. Run the `stat.m` program. Remember, this program is a MATLAB function, thus it needs an argument to execute the desired computations. Since we want to obtain statistics for vector \( x \), we run the command:

```
stat(x)
```

14. It turns out that MATLAB has a lot of built-in functions and toolboxes. Among these are functions that calculate the mean and the standard deviation (both part of the Statistics Toolbox). Use them to check the results of your `stat.m` program:

```
\text{mean}( x ) \\
\text{std}( x )
```

15. Another built-in statistical function is the \textit{median}:

```
\text{median}( x )
```

This should give you the number halfway between the largest and smallest in the series. An intuitive way to check the output of `median` is with:

```
y = \text{sort}( x )
```

11. In the MATLAB command window, type:

```
clear
```

This will clear all the variables currently in memory. If you type:

```
clc
```

you will clear the command line window so that you have a fresh slate. Don’t worry, though, there are ways to save variables in memory to disk, too!

12. Now create a series of numbers:

\[
x = [ 1 , 48 , 81 , 2 , 10 , 25 , 14 , 18 , 53 , 41 ]
\]

13. Run the `stat.m` program. Remember, this program is a MATLAB function, thus it needs an argument to execute the desired computations. Since we want to obtain statistics for vector \( x \), we run the command:

```
stat(x)
```

14. It turns out that MATLAB has a lot of built-in functions and \textit{toolboxes}. Among these are functions that calculate the mean and the standard deviation (both part of the Statistics Toolbox). Use them to check the results of your `stat.m` program:

```
\text{mean}( x ) \\
\text{std}( x )
```

15. Another built-in statistical function is the \textit{median}:

```
\text{median}( x )
```

This should give you the number halfway between the largest and smallest in the series. An intuitive way to check the output of `median` is with:

```
y = \text{sort}( x )
```
which will return a vector \( y \) that includes the elements of \( x \), sorted from largest to smallest. Since there are 10 elements in \( x \) (and in \( y \)), the median value is halfway between the 5\(^{th}\) and 6\(^{th}\) elements of \( y \).

16. Now try your `quart.m` program. To check the answers, look at your vector \( y \). The first quartile should be the 3\(^{rd}\) element in \( y \). The third quartile should be the 8\(^{th}\) element in \( y \).

17. Finally, here are some notes on working with an image (basically just a matrix).

   a. MATLAB has built-in functions (through the Image Processing Toolbox) for opening, displaying, and writing images. See the handout and this url for more information:

   \[ \text{http://amath.colorado.edu/courses/4720/2000Spr/Labs/Worksheets/MatlabTutorial/matlabimpr.html} \]

   However, some of the images you may work with are in file formats other than those that the functions discussed at that website can handle (e.g. ENVI format). Here is a snippet of code for opening an ENVI format image:

   ```
   fid = fopen( 'myfile', 'r', 'l' );
   myimage = fread( fid, [n, m], 'uint8' );
   fclose( fid );
   ```

   The first line opens the file containing your image. The \( \text{fid} \) is a number that MATLAB assigns to the file as a reference. The second line reads the file and puts the contents into the variable \( \text{myimage} \). The variables \( n \) and \( m \) denote the size of the image in rows and columns and \( \text{uint8} \) denotes the type of data (8-bit unsigned integers in this case). The third line closes the file.

   Similarly, to save an image in ENVI-format:

   ```
   fid = fopen( 'outputfilename', 'wb' );
   fwrite( fid, myimage, 'int16' );
   fclose( fid );
   ```

   See the MATLAB help for more information on `save`, `load`, `fopen`, `fread`, `fwrite`, and `fclose`.

   b. To display images (or any other data) as a figure, MATLAB has tons of built-in functions. Here is a snippet of code with just one example:

   ```
   figure( 1 ) \% Makes a new figure window.
   hold off \% Allows the figure to be
% overwritten.
imagesc( myimage ) % Plots the image.
colormap( pink ) % Sets the colormap.
axis image % Formats the axes.
title( ' My Image ' ) % Sets the title.
xlabel( ' column ' ) % Labels the x axis
ylabel( ' row ' ) % Labels the y axis
hold on % Prevents the figure
          % from being overwritten.

There are many, many more options and functions for plotting (e.g. plot,
subplot, image, colormap, pcolor, shading, demcmap, etc.)
Explore the help for MATLAB and the Image Processing Toolbox (and
others) for more.

c. Indexing is an important part of MATLAB. The handout has an extensive
discussion of how to do it. Here is a little example that selects only the
first row of the hypothetical matrix myimage (note that you’d need to
first define myimage for this line to do anything for you)

    firstrow = myimage( 1 , : )

The 1 denotes the first row and the colon (:) indicates all of the column
elements of that row.

d. Let’s say your image (or, in general, your matrix) is called myimage.
You can rotate it by 90 degrees with this command:

    myimage = rot90( myimage )

Note that this line will overwrite the original myimage with the rotated
one. HINT: Also note that this command and the transpose commands
you learned earlier will come in handy in transferring ENVI-format
images between MATLAB and ENVI!