MATLAB Project: Getting Started with MATLAB

Name________________________________

Purpose: To learn to create matrices and use various MATLAB commands for reference later
MATLAB built-in functions used: `[ ] : ; + - * ^`, `size`, `help`, `format`, `eye`, `zeros`, `ones`, `diag`, `rand`, `round`, `cos`, `sin`, `plot`, `axis`, `grid`, `hold`, `path`;
M-files used: `randomint` and `startdat` from the `Laydata5` toolbox or downloadable from pearsonhighered.com/ lay .

Introduction. This can be used as a brief tutorial and as a reference for basic operations. Use MATLAB's `help` command or see a User's Guide for more information. Some of the commands discussed here are about linear algebra topics which will not be formally introduced in your course for several weeks, so even if you go through this project early, you may want to refer back to it at various times. Write notes and questions to yourself as you work through this project.

Instructions. Open MATLAB. The MATLAB prompt is a double arrow, `>>` (or sometimes `EDU>>`). In this project each line that begins with `>>` is a command line, and the bold face words following `>>` are MATLAB commands. Try each of these for yourself by typing the bold face words and then press the key that is labeled "Return" or "Enter," to cause those commands to be executed. (In the first few sections we will write **`[Enter]`** to mean press that key, but we will omit this "carriage return" prompt later.) After you execute each line, study the result to be sure it is what you expect, and take notes. After trying the examples in each section, do the exercises. If a computation in MATLAB ever takes too long, press Ctrl-C to interrupt the process.

If you do not complete this tutorial in one session, the variables you created will be erased when you exit MATLAB. See the remark before Section 6 to find out how to get them back quickly the next time you continue work on this project.

Sections:

Copyright © 2016 Pearson Education, Inc.

1. Creating matrices. A *matrix* is a rectangular array, and in linear algebra the entries will usually be numbers. The most direct way to create a matrix is to type the numbers between square brackets, using a space or comma to separate different entries. Use a semicolon or **`[Enter]`** to create row breaks.

Examples:

```
>> A = [1 2;3 4;5 -6]    [Enter]
A =
      1  2
      3  4
      5 -6
>> B = [1 -2 3     [Enter]
      4 5 -6]      [Enter]
B =
      1   -2   3
      4    5  -6
>> x = [4;3;2]     [Enter]
x =
      4
      3
      2
>> X = [1,2,3]     [Enter]
X =
      1  2  3
```

a) To see a matrix you have created, type its name followed by **`[Enter]`**. Try each of the following and make notes how the results were displayed. Notice MATLAB is case sensitive—for example, *x* and *X* are names for different objects:

```
>> A            [Enter]


>> A,B          [Enter]


>> X,x          [Enter]
```

MATLAB will not try to execute an assignment until it is complete. For example, if you forget and press **`[Enter]`** before typing the right bracket, it will just wait for the bracket. Try this:

```
>> A = [1 2;3 4;5 -6  [Enter]

>> ]                  [Enter]
```

Exercise: If you have not done it already, create the matrices *A*, *B*, **x**, and *X* above. Then create *C*, *D*, *E*, and **vec** as shown below. (We write them with square brackets as a textbook would, but MATLAB does not display brackets.)

Copyright © 2016 Pearson Education, Inc.

b) For each matrix, record what you typed, and be sure the MATLAB display is what you expected.

$$C = \begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix} \qquad D = \begin{bmatrix} 2 & -1 \\ 1 & 3 \\ -2 & 1 \end{bmatrix} \qquad E = \begin{bmatrix} 2 & -1 \\ 0.1 & 3 \\ -2 & 1 \end{bmatrix} \qquad \mathbf{vec} = \begin{bmatrix} 3 \\ -5 \\ 1 \end{bmatrix}$$

Notice that since one entry in E is a decimal, MATLAB displays every entry as a decimal.

2. The arrow keys. MATLAB keeps your most recent command lines in memory and you can "arrow up" to retrieve a copy of any one of those. This can be useful when you want to correct a typing error, or execute a certain command repeatedly.

a) Type the following line and record the error message:

```
>> Z = [1 2 3 4;5 0]        [Enter]
```

Error message: ____

To correct such an error, you could retype the entire line, but there is an easier way. Press the up arrow key [↑] on your keyboard one time to retrieve that last line typed, and then use the left arrow key to move the cursor so it is between 2 and 3 to type a semicolon. Press **`[Enter]`** to cause the new line to execute.

You can also use the right arrow key to move to the right through a line, and if you "arrow up" too far, use the down arrow key to back up. To erase characters, use the `BackSpace` or `Delete` keys. It does not matter where the cursor is when you press **`[Enter]`** to execute the line. Another alternative is to use the Command History Window to double click on a previously entered command.

Exercise. Press the up arrow key several times to find the command line where you defined E. Change the 0.1 entry to 0.01 and press **`[Enter]`** to execute.

b) Record the new version of E:

3. The `size` command. When M is a matrix, the command **`size(M)`** returns a vector with two entries which are the number of rows and the number of columns in M.

```
>> size(A)        [Enter]
ans =
     3   2
```

Notice that `ans` is a temporary name for the last thing calculated if you did not assign a name for that result.

Exercise. Calculate the size of the other matrices you have created, B, X, **x**, C, D, E, **vec**, Z.

4. The `help` command. The command **`help`** can provide immediate assistance for any command whose name you know. For example, type **`help size`**. Also try **`help help`**. For more extensive help on a topic, type **`doc`** followed by the command name. For example, **`doc plot`** contains much more information than **`help plot`**.

5. Accessing particular matrix entries. If you want to see a matrix which you have stored, type its name. To correct entries in a stored matrix, you can reassign them with a command. MATLAB does not work like a text editor in the Command Window– you cannot edit things visible on the screen by highlighting and typing over them.

However, you can change a particular entry, an entire row, an entire column, or even a block of entries. Try the following commands to view and change various entries in the matrix *C* you created above. In each part type the first command line to see what the matrix and certain entries look like before you change them; then type the second command line to cause a change. Record the result of each command and compare the new version of *C* with the previous version to be sure you understand what happened:

a)
```
>> C, C(3,1)

>> C(3,1) = -9
```

b)
```
>> C, C(:,2)

>> C(:, 2) = [1;1;0]
```

c)
```
>> C, C([1 3], [2 3])

>> C([1 3], [2 3]) = [-2 4;6 7]
```

d)
```
>> C, C([1 3],:)

>> C([1 3],:) = C([3 1],:)
```

e)
```
>> C, C(3,:)

>> C(3,:) = [0 1 2]
```

Notice the effect of the colon in `C(:,3)` is to say "take all rows." On the other hand, the effect of the colon in `C(3,:)` and `C([1 3],:)` is to say "take all columns."

Copyright © 2016 Pearson Education, Inc.

We will assume in all the following that you have created the matrices and vectors *A*, *B*, *C*, *D*, *E*, *X*, **x**, **vec** above so they exist in your current MATLAB workspace. If you do not complete this tutorial in one session, all variables will be erased when you exit MATLAB. If you continue this tutorial at a new MATLAB session later, you will need to type in whatever variables you need. However, if the M-files in Laydata5 Toolbox have been set up for your computer, you can simply type **`startdat`** to get *A*, *B*, *C*, *D*, *E*, *X*, **x**, **vec**. Ask your instructor about these capabilities, or see Sections 15 and 16 in this project for more details.

6. Pasting blocks together. When the sizes allow it, you can create new matrices from ones that already exist in your MATLAB workspace. Using the matrices *B*, *C* ,*D* created above, try typing each of the following commands and record the result of each command in the space below it. If any error messages appear, think why.

```
[C D]               [D C]               [C;B]               [B;C]               [B C]
```

7. Some special MATLAB functions for creating matrices: `eye`, `zeros`, `ones`, `diag`. Examples:

```
>> eye(3)                          >> zeros(3)                        >> ones(size(D))
ans =                              ans =                              ans =

       1  0  0                            0  0  0                            1  1
       0  1  0                            0  0  0                            1  1
       0  0  1                            0  0  0                            1  1
```

a) Type each of the following commands and record the result:

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

```
diag([4  5  6  7])          diag([4  5  6  7], -1)       C,diag(C),diag(diag(C))
```

Copyright © 2016 Pearson Education, Inc.

b) Type commands to create the following matrices. For each, record the command you used:

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ $\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$ $\begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 0.1 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & -2 \end{bmatrix}$

8. Using the colon to create vectors with evenly spaced entries. In linear algebra, a *vector* is an ordered n-tuple. Thus, one-row and one-column matrices like those we called **x**, *X* and **vec** above would be called vectors. Frequently it is useful to be able to create a vector with evenly spaced entries (for example, in loops or to create data for plotting). This is easy to do with the colon. Examples:

```
>> v1 = 1:5              >> v2 = 1:0.5:3              >> v3 = 5:-1:-2
v1 =                     v2 =                         v3 =
    1 2 3 4 5               1.0 1.5 2.0 2.5 3.0           5 4 3 2 1 0 -1 -2
```

a) Use the colon notation to create each of the following vectors. Record the command you used for each:

[-1 0 1 2 3 4]   [9 8 7 6 5 4 3]   [4 3.5 3 2.5 2 1.5 1]

b) The numbers from 0 to 2, spaced 0.1 apart (record the command and the first and last few numbers):

9. Using the semicolon to suppress results. When you place a semicolon at the end of a command, the command will be executed but whatever it creates will not be displayed on the screen. Examples:

```
>> x = 1:0.2:3;
>> x
x =
    1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0
```

Try these commands and describe the results ("pi" is a constant in MATLAB which approximates π):

a)
```
>> A;
>> A
```

b)
```
>> w = 0:0.1:pi;
>> w
```

Copyright © 2016 Pearson Education, Inc.

10. The format command. Even when MATLAB displays only 4 to 5 digits, it is storing about 15 significant digits. The `format` command controls how things look on the screen.

a) Type each of the following commands and record the result carefully. Notice that "e" means exponent—it means multiply by some power of 10. For example, 1.2345e002 is the number 1.2345(10^2).

```
>> R = 123.125
```

```
>> format long, R
```

```
>> format short e, R
```

```
>> format short, R
```

The default mode for display of numbers is `format short`. To restore the default mode at any time, type `format`.

b) The command `format compact` is very useful. It reduces the number of blank lines on the screen, allowing you to see more of what you have done recently. Try the following and describe each effect:

```
>> A,B
```

```
>> format compact, A,B
```

```
>> format, A,B
```

11. Matrix arithmetic. You will soon see the definitions of how to multiply a scalar times a matrix, and how to add matrices of the same shape. MATLAB uses * and + for these operations. Try the following examples, using matrices defined above. You should be able to figure out what the operations are doing.

a) Type each line and record the result. If you get an error message, read it carefully and notice why the command did not work:

```
>> A, A+A, 2*A
```

```
>> 2*A - 3*D
```

```
>> A, B, A+B
```

```
>> A, D, A+D, A-D
```

```
>> x, vec, x+vec
```

```
>> x, X, x+X
```

Copyright © 2016 Pearson Education, Inc.

b) MATLAB also uses `*` for multiplication of matrices, which is a somewhat more complicated operation. You will see the definition in Section 2.1 of Lay's text. The definition requires that the "inner dimensions" of the two matrices agree. Without knowing that definition, you can still investigate some of the properties of matrix multiplication (and you may even be able to figure out what the definition is). Type the following lines and record the result of each:

```
>> A, B
```

```
>> A*B
```

```
>> B*A
```

```
>> B, C, B*C, C*B
```

```
>> C, x, C*x
```

```
>> X, C, X*C
```

c) The symbol ^ means exponent in MATLAB. For example, **`Y^2`** is a way to calculate Y^2 (which can also be calculated as **`Y*Y`** of course). Try these:

```
>> C, C*C, C^2
```

```
>> Y = 2*eye(3), Y^2, Y^3
```

If a computation in MATLAB is taking too long, press Ctrl-C to interrupt the process.

12. Creating matrices with random entries. MATLAB's function `rand` creates numbers between 0 and 1 which look very random. They are not truly random because there is a formula which generates them, but they are very close to being truly random. Such numbers are often called "pseudorandom." Similarly, the function `randomint` in the Laydata5 Toolbox creates matrices with pseudorandom integer entries mostly between -9 and 9.

a) Type the commands below and describe the result of each. Arrow up to execute the first two lines several times, to see that the numbers change each time `rand` or `randomint` is called.

```
>> P = rand(2), Q = rand(2,3)
```

```
>> format long, P, Q
```

Copyright © 2016 Pearson Education, Inc.

```
>> format, P, Q                              >> randomint(3)
```

```
>> randomint(3,4)
```

Remarks. You can scale and shift to get random entries from some interval other than (0,1). For example, `6*rand(2)` yields a 2×2 matrix with entries between 0 and 6; and `-3+6*rand(2)` yields a 2×2 matrix with entries between -3 and 3. It is also easy to create random integer matrices without `randomint`. For example, the command `round(-4.5 + 9*rand(2))` produces a 2x2 matrix with every entry chosen fairly randomly from the set of integers $\{-4,-3,\ldots,3,4\}$.

13. Plotting. The `plot` command does 2-D plotting, and when you use it, a graph will appear in a `Figure` window. If the `Figure` window obscures most of the `Command` window and you want to see both windows at once, use the mouse to resize and move them. If you cannot see a particular window at all, pull down the menu Windows and select the one you want.

a) You can specify a color and a symbol or line type when you use `plot`. To learn more, use `help plot` and the MATLAB boxes in Lay's Study Guide. Try the following examples and make a sketch or write notes to describe what happened each time. Notice we use semicolons when creating the vectors here because each vector is quite long, and there is no reason to look at them:

```
>> x = 0:0.1:2*pi; si = sin(x); co = cos(x);
```

```
>> plot(x,si)
```

```
>> plot(x,si,'r')
```

```
>> plot(x,si,'-.')
```

```
>> plot(x,si,'*')
```

```
>> plot(x, si,'b*')
```

Copyright © 2016 Pearson Education, Inc.

b) Here is one way to get more than one graph on the same axis system. Describe the result of each command:

```
>> plot(x,si,'r*',x,co,'b+')
```

```
>> P = [si;co]; plot(x,P)
```

c) Another way to get different graphs on the same axes is to use the `hold on` command. This causes the current graphics screen to be frozen so the next plot command draws on the same axis system. The command stays in effect until you release it by typing `hold off` . Try the following commands, and describe the result of each:

```
>> plot(x,co,'g--'), hold on

>> plot(x, si, 'ro')

>> hold off
```

d) It can be helpful to have grid lines displayed and to set your own limits for the axes. Try the following.

```
>> plot(x, si), grid

>> plot(x, si), axis([ -8  8  -2  2 ])
```

We defined the vector **x** on page 9. Change the vector **x** and use the last MATLAB command above to get the entire graph from –8 to 8.

Remarks: The menu on the Figure Toolbar has an assortment of options that can make editing a graph much easier. The menus and buttons on the Figure window allow you to add axes, labels, titles and other annotations to the graph. As an example, you can label the axis of a particular figure by clicking the **Insert** menu and selecting the label option that corresponds to the axis you want to label: `X label` or `Y label`. A text entry box opens along the axis for you to enter the text. Click anywhere else in the figure background to close the text entry box.

If you want to edit the figure later, save your file as a **`.fig`** file to keep your annotations. To include the figure in a professional-looking report, you may want to export the figure since what you see on the screen is not necessarily what you get in a file. Select **`Export setup`** from the figure's `File` menu. There are numerous options, and the documentation center is there to help you.

14. Creating your own script M-files. A script M-file allows you to place MATLAB commands in a text file so that you do not need to reenter the same information at the MATLAB prompt again and again. It is a good idea to have MATLAB running while you edit an M-file. This will allow you to quickly switch back and forth between the Edit screen and the MATLAB screen so you can try running your file, editing it again, running it again, until it works the way you want.

To use the M-File Editor inside MATLAB, click `New Script` on the upper left corner of the MATLAB screen. If you want to edit one that exists already, choose Open and then browse to find the file you want. The M-File Editor will open for you to edit your work. You can also edit an M-file using another text

Copyright © 2016 Pearson Education, Inc.

editor but you should realize that an M-file must be saved with the extension `.m`, not `.txt` or `.docx` whereas the `.m` extension will be added automatically if you use the M-File Editor. Type the commands you want in the `Edit` window and save your work.

In addition, you must save an M-file in some directory which is in the MATLAB path, or else MATLAB will not be able to find the file and execute it. For example, on many computers the directory `C:\matlab` is always in the path. See Section 16 below for some details about these matters.

Don't close the M-File Editor window yet. Instead click on the MATLAB `Command` window and type the name of the file. For example, if you created a file `playing.m`, type `playing` to execute the file. Continue to edit the file until you are satisfied.

As you gain more experience, you may want to experiment with script M-files to prepare homework assignments and projects. It is good practice to include comments to outline the steps in your calculations and to interpret the results of various computations. Statements in the file which begin with a percent sign are ignored by MATLAB but are greatly appreciated by others.

```
%The system is inconsistent, because 0 = 5/2 is not true.
```

See Lay's Study Guide to get you started.

15. Ways to get Laydata5 Toolbox. The M-files in Laydata5 Toolbox can be downloaded from the text's web site `http://pearsonhighered.com/lay` . Follow the on-screen directions. The README file in the zipped folder has information on installing the files. For example, if there is a Laydata or Laydata4 Toolbox from a previous version of the text, you will need to remove it.

One of the great benefits of Laydata5 Toolbox is that it contains data for most of the exercises in the text. For example, the command **`c1s2`** will retrieve the data for the exercises in Chapter 1 Section 2. Another benefit is that the Laydata5 Toolbox contains data and programs for application projects which accompany the text.

16. Installing M-files into the MATLAB path. Whenever you want to use M-files that are not part of commercial MATLAB, such as those in Laydata5 Toolbox, you must tell MATLAB where to look for them. For example, suppose the Laydata5 M-files are stored on your `C:` drive, in a folder called `laydata5`. The following procedures will work for installing any M-files, except the names of the folders may be different.

A. For Macintosh: Drag the icon for the `laydata5` folder into the MATLAB folder on your hard drive. Start MATLAB. From the `File` menu, select `Open`. Select one of the M-files in the `laydata5` folder (to open the file as if for editing), then close the file. You are done now – after this, MATLAB will always know to look in that `laydata5` folder.

B. For Windows: Open your Explore window and drag the folder called `laydata5` into your MATLAB folder. Its address is now something like `c:\matlab\laydata5` . The MATLAB command `path` outputs a long string that contains the addresses of all the folders where MATLAB looks for M-files, and you need to adjoin the address of your new folder to that string. (If you have started MATLAB, you can see the present contents of that string at any time by typing **`path`**.)

Copyright © 2016 Pearson Education, Inc.

For MATLAB R2014a and later, in the ENVIRONMENT tab at the top click on **Set Path**. Click on the `Add with Subfolders` button and browse the directory to locate and select the toolbox `laydata5`. Highlight the folder and click on **`OK`**. Click on **`Save`** to save the changes for future session and exit the dialogue box.

For earlier version of MATLAB, use **File| Set Path** (or type `pathtool` in the Command Window) to open the Set Path dialogue box. Click on the `Add with Subfolders` button and browse the directory to locate and select the toolbox `laydata5`. Highlight the folder and click on **`OK`**. Click on **`Save`** to save the changes for future session and exit the dialogue box.

From now on, the new path will be in effect. That is, whenever you use MATLAB, it will look for M-files in `c:\matlab\laydata5` as well as in all the other folders which were in the path originally.

C. For MATLAB on a network: Ask the system administrator to store your folders and adjoin their addresses to MATLAB's path. The method for doing that will depend on what version of MATLAB the network is running.

17. Other Features of MATLAB. We have only scratched the surface of the many features MATLAB provides. The focus in this project has been on the **Command Window** which is usually located in the center of the screen. You type in commands at the prompt >>. However, buttons, menus, and toolbars have much of the same functionality of the typed-in commands and are easy to use. These vary with different versions of MATLAB so we have focused on the typed commands.

When you open MATLAB, there are at least four panels and a toolbar with tabs available. In general, the default appearance includes the **Current Folder**, the **Command Window**, the **Workspace**, and **Command History** panel.

Usually at the left of the screen, the **Current Folder** lists the files in the current directory. In order for MATLAB to run a particular M-file, the file must either be in the current directory or on the list known as its **path**. By selecting the appropriate icon above the **Current Folder**, you can browse and select a new folder to be in the current directory. By double clicking on a file in the **Current Folder**, you edit that file. MATLAB is also capable of loading other common format files such as spreadsheet data. With your file in the **Current Folder** panel, you can double-click on the file and follow the resulting prompts or you can use an Import Data button on the top.

Usually at the top right, the **Workspace** panel shows you the names of any variables in use in the current work session. When you begin, the workspace is usually empty. As you define new variables, the variable name and its type will appear in the **Workspace** panel. You can change values by double clicking a variable. You can also experiment with creating graphs by right clicking or selecting variables to use with the Plots Tab on the toolbar at the top.

On the lower right, the **Command History** records the commands entered in your session. Rather than using the arrow button, you can re-enter a command by double clicking on it. You can also select multiple commands in the panel, right click, and select `Create Script` or `Create M-file` to save the typed commands to a text file. This may be helpful to save what you have done for a future session.

The layout can be customized by using the Layout menu in the Environment tab. If you lose a panel or are unhappy with the layout on the screen, go to the Layout menu on the Environment tab to select the Default layout to start again.

Considering its mathematical and numerical capabilities, MATLAB provides an appealing programming environment for numerical work and simulations. While the main emphasis here is not programming,

Copyright © 2016 Pearson Education, Inc.

MATLAB can provide a straightforward introduction to programming in this context. Its Editor/Debugger is a powerful feature for writing programs and debugging them. For example, you can run code step by step to find errors or to understand a program fully. MATLAB also has a `checkcode` command and a Code Analyzer to check MATLAB code for potential problems.

The Mathworks web site, `www.mathworks.com`, has more information about MATLAB including video tutorials at
`www.mathworks.com/academia/student_center/tutorials/launchpad.html`.

Copyright © 2016 Pearson Education, Inc.