Computer Project: Getting Started with MATLAB **Name**_____

Purpose: To learn to create matrices and use various MATLAB commands. Examples here can be useful for reference later.

MATLAB functions: **[ ] : ; + - * ^**
size, **help**, **format**, **eye**, **zeros**, **ones**, **diag**, **rand**, **round**, **cos**, **sin**, **plot**, **axis**, **grid**, **hold**, **path**; and **randint** and **startdat** from Lay's Toolbox

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 it.

Instructions. Start MATLAB by double-clicking on its icon. The MATLAB prompt is a double arrow, >>. 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: that is, type 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 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:

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 and 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
```

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 in bracket, as a textbook would, but MATLAB does not display brackets.) 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 one entry in *E* is a decimal; this is why MATLAB displays every entry as a decimal.

2. The arrow keys. MATLAB keeps about 30 of 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. 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 this is easier: press the up arrow key on your keyboard one time to retrieve that last line typed, use the left arrow key to move the cursor so it is between 2 and 3, type a semicolon and then 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.

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. 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*. Example:

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

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

Exercise. Calculate the size of each 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**.

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 must reassign them with a command. That is, MATLAB does not work like a spreadsheet or text editor – you cannot edit things visible on the screen by selecting them and typing over.

But you can see or change a particular entry, or an entire row, or an entire column, or even a block of entries. Try the following commands to view and change various things 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 each time:

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", and its effect in **C(3, :)** and **C([1 3], :)** is to say "take all columns."

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. So when you continue this tutorial at a new MATLAB session later, type **startdat** to get *A*,*B*,*C*,*D*,*E*,*X*,**x**,**vec**—or just type in again whatever variables you need, as they are used below.

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)
ans =
     1  0  0
     0  1  0
     0  0  1
```

```
>> zeros(3)
ans =
     0  0  0
     0  0  0
     0  0  0
```

```
>> ones(size(D))
ans =
     1  1
     1  1
     1  1
```

Exercises. 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))**

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

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

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

v1 =
     1  2  3  4  5
```

```
v2 = 1:0.5:3

v2 =
    1.0  1.5  2.0  2.5  3.0
```

```
v3 = 5:-1:-2

v3 =
     5  4  3  2  1  0  -1  -2
```

Exercises. 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]

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

9. Using the semicolon to suppress printing

. 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:

```
>> y = 1:0.2:3;

>> y
y =
    1.0  1.2  1.4  1.6  1.8  2.0  2.2  2.4  2.6  2.8  3.0
```

Exercises. 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

10. The format command. This controls how things look on the screen. Type each of the following commands and record the result carefully. Notice that "e" means exponent—in fact, 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**.

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 the effect of each:

>> A,B

>> format compact, A,B

>> format, A,B

11. Matrix arithmetic. The definitions of how to multiply a scalar times a matrix, and how to add matrices of the same shape will be in your text, but you can probably figure them out from the calculations below. MATLAB uses ***** and **+** for these operations. Try the following examples, using matrices defined above. 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

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

>> 2*A - 3*D

>> x, vec, x+vec

>> A, B, A+B **>> x, X, x+X**

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**

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**

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 **randint** in Lay's Toolbox creates matrices with pseudorandom integer entries. [You will get an error message if you try to use **randint** and the file randint.m has not been installed on your computer; don't worry about that at this time.]

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 **randint** is called.

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

>> format long, P, Q

>> format, P, Q

>> randint(3) **>> randint(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 2x2 matrix with entries between 0 and 6; and **-3 + 6*rand(2)** yields a 2x2 matrix with entries between -3 and 3. It is also easy to create random integer matrices without **randint**. 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 {-5, -4, .. 4, 5}.

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.

As the examples below show, 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*')**

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)**

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
```

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 ])
```

14. Creating your own M-files.

General information:

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, etc., until it works the way you want.

An M-file must be saved with the extension .m, not .txt or .doc. This extension will be added automatically if you use the File Menu in MATLAB to open a new or existing M-file. Also, 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, the directory C:\matlab is always in the path. See Section 16 below for some details about these matters.

Using the editor inside MATLAB:

1. [This section was written for an older version of MATLAB. For a newer version, see screenshots on the following pages.] Click File on the upper left corner of the MATLAB screen. Choose New if you want to create a new M-File, or if you want to edit one that exists already, choose Open and then browse to find the file you want. Double click on the file name to open it in an Edit window. If this doesn't work inside MATLAB, use Notepad as described below.

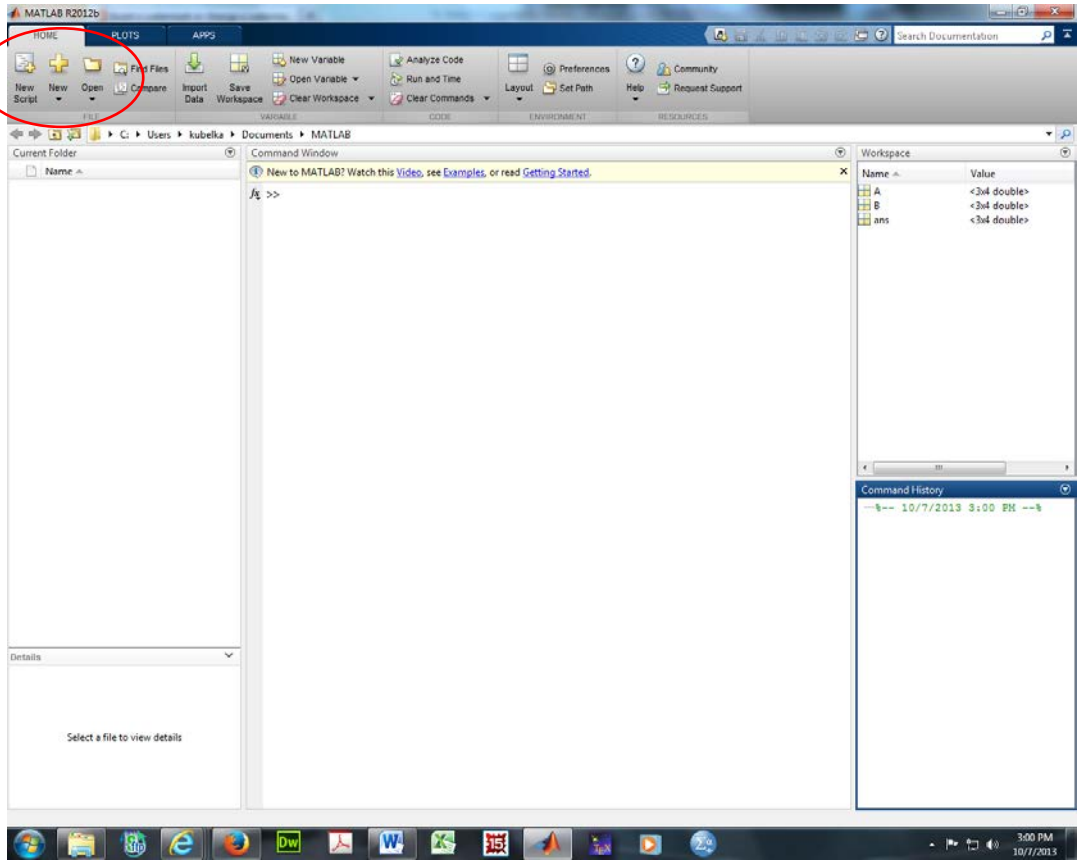
You also can search for an existing file by using the Search window or Windows Explorer. On the ribbon below your Windows screen, click on Start. Then type the name of the file you want (or part of the name) in the Search Programs and Files field and click the magnifying glass. Once you locate the file, double click on its name to open it for editing. Alternatively, right click on Start. Then choose Open Windows Explorer and look in various folders for the file you want.

2. [This section needs to be updated.] Type the commands you want in the Edit window—any valid MATLAB commands are ok. Then pull down the File menu to "Save As." Henceforth we will write **File|Save As** for this kind of menu/mouse use. In the box labeled "Save In," type the name of the folder where you want to store this file. If unsure, type C:\matlab. In the box labeled "File Name," type a name for your file. For example suppose you type play. Then click "Save." The editor will add the extension .m .

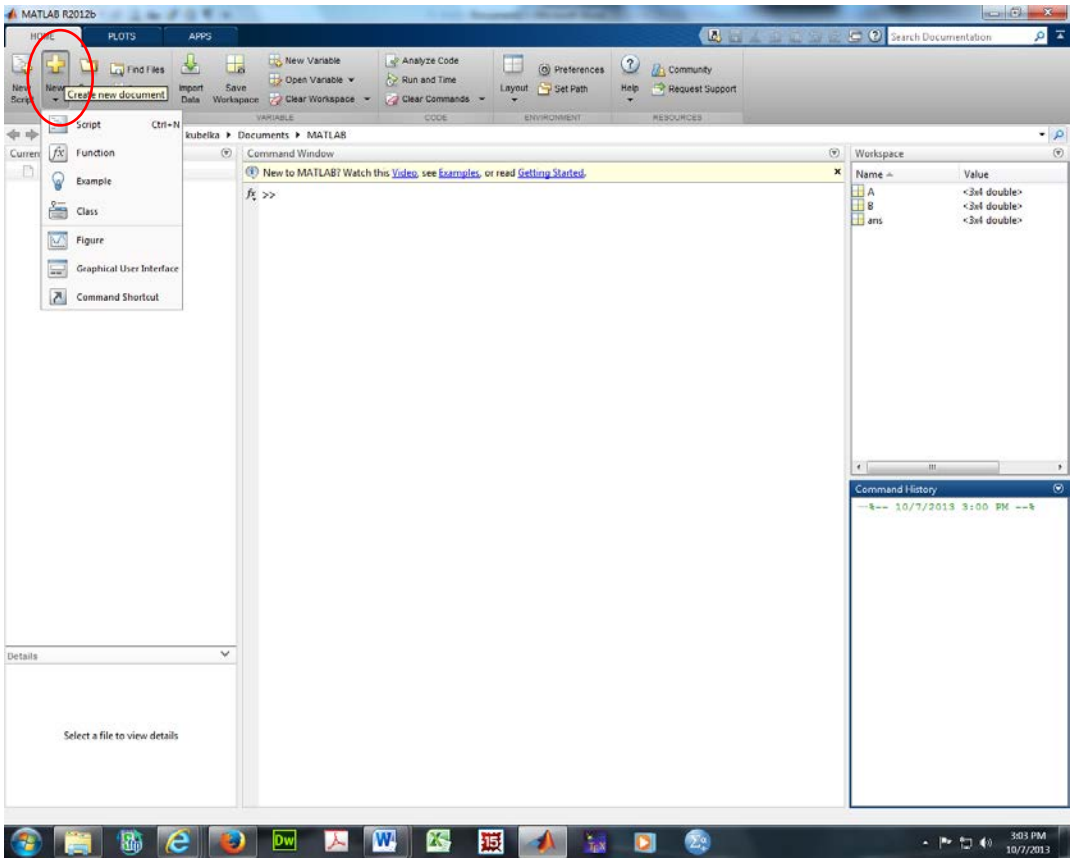
Don't close the Edit window yet; instead click on the MATLAB Command window and type **play** to execute the file. If you want to edit the file more, click on the Notebook window, make changes and save it again. Repeat this procedure until your file works satisfactorily, then close the file by clicking on the box in the upper right corner of the Edit window or by using **File|Exit**.

Using Notepad outside MATLAB:

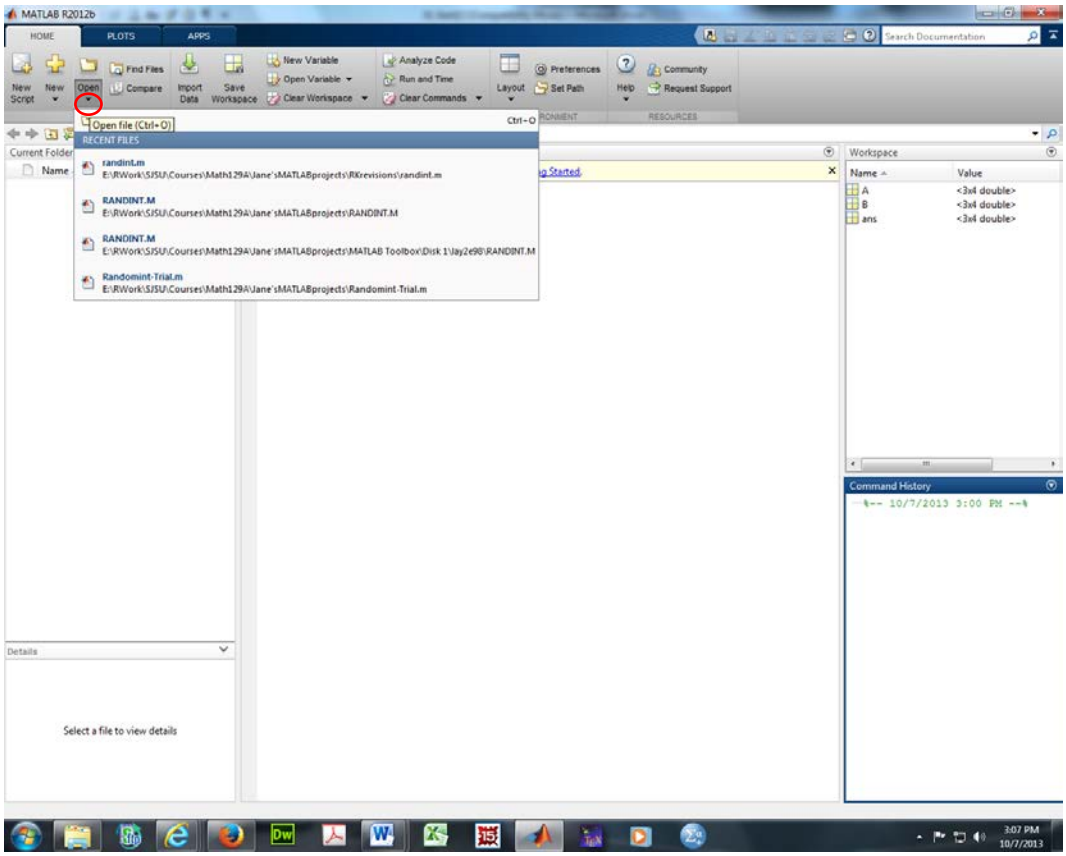
Notepad is a simple editor that comes with Windows that you will find in C:\Windows. Click on the Notepad icon to open an edit screen. You can type the commands for a new M-file here, or if you want to edit a file that already exists, choose **File|Open,** type the address of the file you want, and press [Enter]. To save a new M-file, pull down the File menu to **Save As** and type the name you want for the file. For example, you could, and type play.m as the name. Notice you must type the extension .m yourself (because Notepad doesn't know to add it).



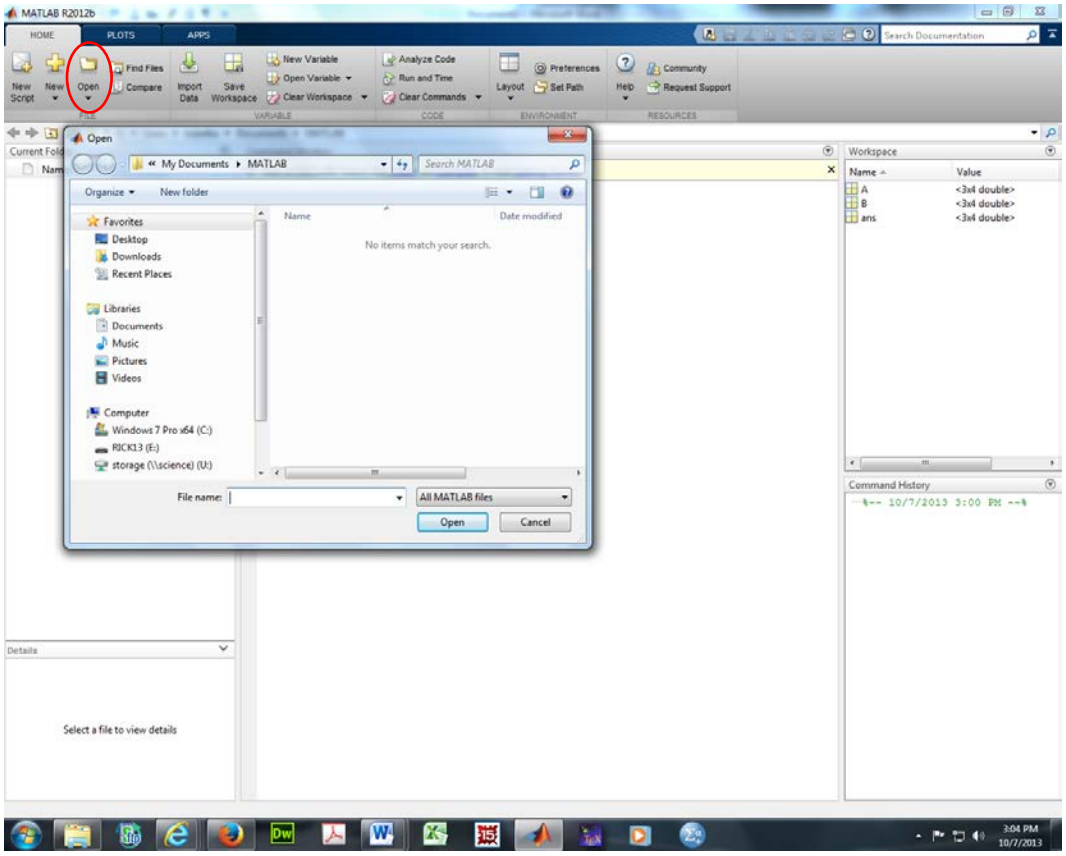
Click on the New button for a list of options.



Clicking on the black triangle in the Open button gives you a list of previously opened files.



Clicking on the Open button gives you the usual window.



ALL OF THE FOLLOWING INFORMATION NEEDS TO BE UPDATED.

15. Ways to get Lay's M-files and ATLAST M-files

The M-files in Lay's Toolbox can be downloaded from the Lay web site http://www.laylinalgebra.com .[1] This requires the password you received with your text. The files are available in the Data Downloads section.

There are versions for various platforms. Click on the version you want, and save the files in a folder on your desktop. They will be in a compressed form. Refer to the README file for information on downloading and decompressing the files.

ATLAST files can be downloaded from http://www.umassd.edu/SpecialPrograms/Atlast . When you see the ATLAST page, click on "Library of ATLAST M-files for MATLAB," then click on the platform you use: "Windows," "Unix" or "Macintosh." The files will be sent at once to your home directory. In your home directory you will see a file, atlast.zip. Move that into another folder if you wish (e.g., into your MATLAB folder), then decompress the files.

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 Lay's Toolbox, you must tell MATLAB where to look for them. For example, suppose Lay's M-files are stored on your C: drive, in a folder called laydata. 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 laydata 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 laydata 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 laydata folder.

---

[1]The author and Addison-Wesley Longman Publishing Company make no representation or warranty of any kind, expressed or implied, with regard to the programs contained herein, including without limitations any implied warranties of merchantability or fitness for particular purposes, all of which are expressly disclaimed. The author and Addison Wesley Longman Publishing Company shall not be liable for incidental or consequential damages in connection with or arising out of the use of these programs.

B. For Windows: Open your Explore window and drag the folder called laydata into your MATLAB folder. Its address is now c:\matlab\laydata . 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**.) Here are some ways to do this, for various versions of MATLAB and various operating systems.

1. If you have Student MATLAB 5.3, start it and then use **File|Set Path** to open a small screen called "Path Browser." Click the Browse button, then locate and click on laydata. Use **Path|Add to Path**. Another small screen will pop up and ask you to confirm this. Click OK. Next use **File|Save Path**. Finally, use **File|Exit Path Browser**. The method is similar in MATLAB 6.

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\laydata as well as in all the other folders which were in the path originally.

2. If instead you have Student MATLAB 5, start it and click on the Path Browser icon at the top of the screen. (This icon looks like two folders.) This opens a small screen called "MATLAB Path." Click **Add to Path**. Click the button **Browse** (which may be a button with ". . ." on it). Then locate and highlight the folder laydata. Click the button "OK," which returns you to the MATLAB Path window. Use **File|Save Path**, then click the button **Close** to close the MATLAB Path window.

3. If you have Student MATLAB 4, open Windows, then use **Start|Find|Files or Folders** and locate the file matlabrc.m . Double click on its name to open it for editing. Scroll down about 75 lines until you see lines that look like

```
';C:\MATLAB\toolbox\sigsys',...
]);
```

Insert a new line between these two, so the lines now look like

```
';C:\MATLAB\toolbox\sigsys',...
';C:\MATLAB\laydata',...
]);
```
Save the file and then close the Edit screen.

If MATLAB is currently running, type **matlabrc** at this time, to establish the new path for the current session. From now on, whenever you start MATLAB, it will look for M-files in c:\matlab\laydata 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.

D. For any operating system: Strange things happen occasionally on computers. If for some reason the instructions above do not work on your system, the following will always work, but this method has to be repeated each time you start MATLAB. Start MATLAB and type the appropriate one of the following commands:

In MATLAB 5, type **addpath 'c:\matlab\laydata'**
In MATLAB 4, type **path(path, 'c:\matlab\laydata')**
In MATLAB 3.5, type **matlabpath([matlabpath, 'c:\matlab\laydata'])**