Project 1 Key

>> 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 with square braces as a textbook would, but MATLAB does not display braces.) For each matrix, record what you typed and be sure the MATLAB display is what you expected.

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

C = [-4,8,-1;5,0,3;6,2,10] E = [2,-1;0.1,3;-2,1]

D = [2,-1;1,3;-2,1] vec = [3;-5;1]

Notice that since one entry in *E* is a decimal, 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: Error using vertcat , CAT arguments dim not consistent

To correct such an error, you could retype the entire line. 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*:

$$E = \begin{bmatrix} 2.0000 & -1.0000 \\ 0.0100 & 3.0000 \\ -2.0000 & 1.0000 \end{bmatrix}$$

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*.

size(B) = 1 6 size(C) = 3 3 size(vec) = 3 1

size(X) = 1 3 size(D) = 3 2 size(Z) = 3 2

size(x) = 3 1 size(E) = 3 2

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 darkening 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)**

$$\begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix}$$

>> C(3,1) = -9

$$\begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ -9 & 2 & 10 \end{bmatrix}$$

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

$$\begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ -9 & 2 & 10 \end{bmatrix}$$

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

$$\begin{bmatrix} -4 & 1 & -1 \\ 5 & 1 & 3 \\ -9 & 0 & 10 \end{bmatrix}$$

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

$$\begin{bmatrix} -4 & 1 & -1 \\ 5 & 1 & 3 \\ -9 & 0 & 10 \end{bmatrix}$$

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

$$\begin{bmatrix} -4 & -2 & 4 \\ 5 & 1 & 3 \\ -9 & 6 & 7 \end{bmatrix}$$

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

$$\begin{bmatrix} -4 & -2 & 4 \\ 5 & 1 & 3 \\ -9 & 6 & 7 \end{bmatrix}$$

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

$$\begin{bmatrix} -9 & 6 & 7 \\ 5 & 1 & 3 \\ -4 & -2 & 4 \end{bmatrix}$$

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

$$\begin{bmatrix} -9 & 6 & 7 \\ 5 & 1 & 3 \\ -4 & -2 & 4 \end{bmatrix}$$

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

$$\begin{bmatrix} -9 & 6 & 7 \\ 5 & 1 & 3 \\ 0 & 1 & 2 \end{bmatrix}$$

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. 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 the Laydata 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]

$$\begin{bmatrix} -9 & 6 & 7 & 2 & -1 \\ 5 & 1 & 3 & 1 & 3 \\ 0 & 1 & 2 & -2 & 1 \end{bmatrix}$$

[D C]

$$\begin{bmatrix} 2 & -1 & -4 & 8 & -1 \\ 1 & 3 & 5 & 0 & 3 \\ -2 & 1 & 6 & 2 & 10 \end{bmatrix}$$

[C;B]

$$\begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \\ 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$$

[B;C]

$$\begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \\ -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix}$$

[B C]

Error using => horzcat
b/c B has 2 rows
& C has 3 rows

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)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

zeros(3,5)

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

zeros(3)

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

ones(2,3)

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

ones(2)

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

diag([4 5 6 7])

$$\begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 7 \end{bmatrix}$$

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

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 \end{bmatrix}$$

C, diag(C), diag(diag(C))

$$\begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix}, \quad \begin{bmatrix} -4 \\ 0 \\ 10 \end{bmatrix}, \quad \begin{bmatrix} -4 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 10 \end{bmatrix}$$

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

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

eye(2)

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

zeros(2)

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix}$$

ones(3,2)

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

diag([7 0.1 6 -2])

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] -1:4

[9 8 7 6 5 4 3] 9:-1:3

[4 3.5 3 2.5 2 1.5 1] 4:-.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):

0:0.1:2

0 .1 .2 .3 .4 ... 1.8 1.9 2.0

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:

```
>> 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
```

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

a) >> A; no printout
>> A

$$= \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & -6 \end{bmatrix}$$

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

= 1 1.1 1.2 1.4 1.5
...
2.9 3.0 3.1

10. The format command. This command 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 = 123.125 123.1250

>> format long, R 1.231250000000000e002

>> format short e, R ~~123.1250~~ 1.2313e+002

>> format short, R 123.1250

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 $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & -6 \end{bmatrix}$ $B = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$ close to each other

>> format compact, A,B

looks the same as above

>> format, A,B

extra spacing b/t the matrices in the printout

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. 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

$\begin{bmatrix} 2 & 4 \\ 6 & 8 \\ 10 & -12 \end{bmatrix}$

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

$\begin{bmatrix} 3 & 1 \\ 4 & 7 \\ 3 & -5 \end{bmatrix}$ $\begin{bmatrix} -1 & 3 \\ 2 & 1 \\ 7 & -7 \end{bmatrix}$

>> 2*A - 3*D

$\begin{bmatrix} -4 & 7 \\ 3 & -1 \\ 16 & -15 \end{bmatrix}$

>> x, vec, x+vec

$\begin{bmatrix} 7 \\ -2 \\ 3 \end{bmatrix}$

>> A, B, A+B

error not same size

>> x, X, x+X

error not same size

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 $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & -6 \end{bmatrix}$ $B = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$

>>A*B

$\begin{bmatrix} 9 & 8 & -9 \\ 19 & 14 & -15 \\ -19 & -40 & 51 \end{bmatrix}$

>> B*A

$\begin{bmatrix} 10 & -24 \\ -11 & 64 \end{bmatrix}$

>> B, C, B*C, C*B $B = \begin{bmatrix} 1 & -2 & 3 \\ 4 & 5 & -6 \end{bmatrix}$ $C = \begin{bmatrix} -4 & 8 & -1 \\ 5 & 0 & 3 \\ 6 & 2 & 10 \end{bmatrix}$ $B \cdot C = \begin{bmatrix} 4 & 14 & 23 \\ -27 & 20 & 49 \end{bmatrix}$ $C \cdot B =$ Error

>> C, x, C*x

$\begin{bmatrix} 6 \\ 26 \\ 50 \end{bmatrix}$

>> X, C, X*C

$\begin{bmatrix} 24 & 14 & 35 \end{bmatrix}$

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

$$\begin{bmatrix} 50 & -34 & 18 \\ -2 & 46 & 25 \\ 46 & 68 & 50 \end{bmatrix}$$

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

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}, \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix}, \begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 8 \end{bmatrix}$$

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 Laydata Toolbox creates matrices with pseudorandom integer entries.

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)

$$P = \begin{bmatrix} .8147 & .1270 \\ .9058 & .9134 \end{bmatrix} \qquad Q = \begin{bmatrix} .6324 & .2785 & .9575 \\ .0975 & .5469 & .9649 \end{bmatrix}$$

>> format long, P, Q

$$P = \begin{bmatrix} .814723686393179 & .126986816293506 \\ .905791937075619 & .913575856139019 \end{bmatrix}$$

$$Q = \begin{bmatrix} Q \text{ with lots more decimal places} \\ 15 \end{bmatrix}$$

>> format, P, Q

back to 4 decimal places

>> randomint(3)

$$\begin{bmatrix} 0 & 1 & 8 \\ -5 & 5 & 1 \\ -6 & 3 & 6 \end{bmatrix}$$

>> randomint(3,4)

$$\begin{bmatrix} 3 & -2 & 2 & 1 \\ -2 & 2 & 0 & -4 \\ 2 & -2 & 0 & 0 \end{bmatrix}$$

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 **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 {-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. It is not possible to see both windows at once in MATLAB 3.5.

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)

standard plot of $\sin x$ on $[0, 2\pi]$

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

standard plot of $\sin x$ on $[0, 2\pi]$
line is red

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

line is —·—·—

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

line is a string of *
(default color)

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

looks the same as last 1
(line color is blue)

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+')

std $\sin x$ on $[0, 2\pi]$ in red & line is ***
std $\cos x$ on $[0, 2\pi]$ in blue & line is +++
both graphs in same window

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

both $\sin x$ & $\cos x$ plotted over $[0, 2\pi]$
lines green & blue — smooth line

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**

std graph of cos(x) on $[0, 2\pi]$
line green & dashed

plot(x, si, 'ro')
hold off

std graph of sin x on $[0, 2\pi]$
line red & connected circles

both graphs on same figure

It can be helpful to have grid lines displayed, and to set your own limits for the axes. Try the following. (If using MATLAB 3.5, type the **plot** commands last instead of first in each of the following lines -- e.g., **grid, plot(x, si)** .)

>> plot(x, si), grid

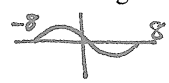
std plot of sin x on $[0, 2\pi]$
w/ grid lines

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

std plot of sin x on $[0, 2\pi]$
graphing window forced to $[-8, 8] \times [-2, 2]$

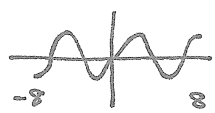
We defined the vector **x** on the top of page 8. Change the vector **x** and use the last MATLAB command above to get the entire graph from –8 to 8.

way 1: plot($\frac{8}{\pi}$x-8, si)



14. Creating your own M-files.

way 2: x = -8:.1:8
plot(x, sin(x))



General information:

An 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, 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, on many computers the directory C:\matlab is always in the path. See Section 16 below for some details about these matters.