MATLAB PRIMER

To get the MATLAB program, go to software.geneseo.edu. On the Mac or Windows side, click on "academic." Among the many programs you will find the latest version of MATLAB. For help, you may go to CIT self-help page (there is a link to it on the left side of software page). Click on CIT SELF HELP GUIDES, scroll down to software and click on Matlab. Best of all, the folks at the CIT Help Desk are ready and willing to help you install Matlab.

I. STARTING UP

Open **Matlab** from your Programs directory or by clicking on its icon.

There are several important windows that you use in Matlab: the **Command Window**, the **Command History**, the **Current Directory**, and the **Editor**. To see these three windows at once (or suppress them later), click on the **Desktop** in the Matlab tool bar. Choose **Desktop Layout**. The option labeled **Default** will display the command window, the command history and the contents of "Current Directory," which is the folder called "Work" by default.

The **Command Window** is Matlab's scrap paper. You do all your commutations here. But it is not permanent. It works very much like the screen on the TI 89 with regards to editing and recalling lines, but it is blank again every time you open a new session.

The **Command History** is where the history of your Matlab commutations are stored. Like the TI 89, it stores the most recent commands (lots of them) and sorts them by session date and time.

For a more systematic and permanent storage, you can store the command history of you session as an **M-file** for future reference. To do so, either click on the date and time of your session in the Command History to highlight your entire session or highlight the portion you want to save. Right click and choose "Create M-file." Then the **Editor** will pop up with your work entered. Name your file and save it. To reactivate your session, just type its name at the prompt in the Command Window.

Example 1

Open Matlab by clicking on the icon. If necessary, bring the Command History window and the Current Directory into view via the Desktop tab. In this example, we add 2 and 2, assign the answer to the variable *a*. Then we will compute $a^3 + 3$ and assign it to *b*. We will then change our mind and compute a^3 + 5 and assign it to *b*. Finally we take the square root of *b* and check the result by squaring our answer.

***All commands are **entered** at the last >> prompt of the Command Window and **executed** by pressing Enter. To suppress outcome (you will want to later), end your line with a semi-colon.
Important: to **stop** a command, press Crtl and C simultaneously and then press enter.

To **reuse and edit a command**, scroll up with the up arrow until you reach the command. (You can scroll down if you overshoot.) Note that the command appears at the bottom-most prompt, ready for re-use or editing.

```
>> format compact
>> format short
>> clear
%Read about the above commands in the paragraph that follows.
>> a=2+2
a =
    4
>> b=a^3+3
b =
   67
>> b=a^3+5
b =
   69
>> sqrt(b)
ans =
   8.3066
>> ans^2
ans =
   69
```

Note that sqrt(b) is assigned to the temporary variable ans. The line in bold (my edit) was not typed but re-used and edited.

The first two lines help us format our answer. The first, **format compact**, eliminates spaces between the response lines. You may want to make a habit of starting with those lines. The second, **format short,** tells Matlab to give us 5 places in our decimal answers. Other format options include **format rat**, which tells Matlab to give us a rational approximation to the given answer even though all its computations are floating point numbers. Also, we can keep lots of decimal places with **format long**. The command **clear** clears all the variables so you won't get surprised with an unintentional value. The command **clear x** clears only what you stored in *x*.

Note the entries in the **Command History**. It records each new command, but no echoes. To **SAVE** your session or part of the session, click on the date of your session in the command window. Right click and select **Create M-file**. The **Editor** will pop up with a copy of the command history. Save your file as you would save any document. Name your file and suppose I name my M file "test." For now, your file to the default folder

Work that Matlab provides. (If you are working on a school machine and want to save the file, you must either save it to a disk of jump drive or to your web space.)

Now exit Matlab and restart it. The Command Window is blank. At the prompt, you can type the name of your saved session. It will be reactivated. So if I type "test" at the prompt, all the commands in the M-file test will be executed. The big advantage of saving a session through the Editor is that you can indeed edit you work just as you do a word file and save only what you want. You can also add **comments** by beginning a comment line with the symbol **%.** Your comments (documentation) will not appear in the command window, but will be very important for you to be able to interpret your own work. So use % generously!

You can get **Help** with a command by typing **help (*command*)** e.g. entering **help clear** will download information about the command clear. Otherwise, click on Help in the tool bar and go to MATLAB Help for more extensive options. (EXPLORE!!!!!)

II. MATLAB as Calculator.

a. Arithmetic operations are computed on Matlab in the same manner as an ordinary calculator:

2+2
2*5.7
3.2^7
2/7 (the outcome is in decimal form)
(Note: Matlab also uses the backslash for division where a\b means b divided by a. So a/b = b\a. Be careful!)

Complex numbers use **i** or **j** for sqrt(-1):

```
>> (2+3i)*(5+7i)
ans =
   -11      +   29i
```

Preceding an operation by a **period** allows you to perform the operation on a vector of numbers. Let's square all the numbers in the vector v = [2 3 4] .

```
>> [2 3 4].^2
ans =
    4     9    16
```

The same calculation without the period produces an error message because we cannot square a vector. The period indicates that the operation should be performed on each component separately.

To assign a value to a variable (case sensitive), Matlab uses the equal sign. So to assign the value 10 to *a* and 20 to *A*, and then add them, write:

```
>> a=10
a =
   10
>> A=20
A =
   20
>> A+a
ans =
   30
```

If variables *a* and *b* have been assigned values, then we can compute *a*^2, and *a***b*, etc. Some of the standard functions available in Matlab are:

sin	asin (arcsin)	exp (*e*)	abs (absolute value)	sqrt (square root)
cos	acos	log (**natural** log)	floor	ceil
tan	atan			

The number π is denoted by **pi** (lower case). The number *e* is obtained by entering exp(1). You can assign it to the variable e

Note that the word log denotes **natural log.** Log_{10} (base 10) of *x* would be computed by log(*x*)/log(10).

b. Matlab allows you to create your own **functions** through M-files. First, go to File in the tool bar and click on NEW. Then select M-File. The Editor window will pop up blank. Say you want to form the function $y = f(x)$ where $f(x) = x^2 + x + 1$. On the top line of the page write:

```
function  y = f(x)
```

The first line tells Matlab that the M file contains the definition of a function, that the function's name is *f*, that it accepts one input *x* and yields one number *y* as its output.

On the next line, give the recipe for the output *y* in terms of *x*:

```
y = x.^2 + x + 1
```

(Note the period before the exponential operator. More on that later.)

Now save the M-file, naming it *f* (or whatever your function's name is).

```
function y = f(x)
y = x.^2 + x + 1
```

Now in the Command window, enter f(2). The response will be 7.

```
>> f(2)
y =
     7
ans =
     7
```

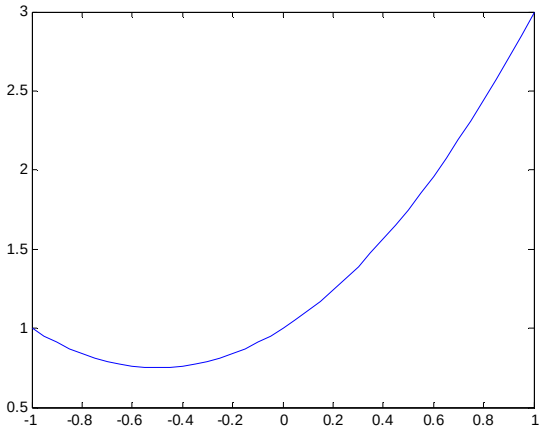
Note: in the Command Window, the answer 7 is assigned to the temporary variable *ans* and *y* remains unassigned. To assign *y* the value 7, do:

```
>> y =f(2)
y =
   7
y =
   7
```

To plot a function, we need to set a domain and an increment that determines how many points are sampled for graphing. So to graph our function *f* on the domain $-1 \leq x \leq 1$ at an increment of 0.05, we proceed as follows. Don't forget the semicolon!

```
>> x=[-1:.05:1];
>> y=f(x);
>> plot(x,y)
```

The first instruction establishes the domain x =[left endpoint: increment : right endpoint]. The resulting *x* is actually a vector of values. We compute the vector of y-values with the instruction y = f(x). (That is why we used .^ in the definition of the function. More later.) Then we instruct Matlab to plot. The plot appears in a separate window. You can save it as name.fig. Look up the plot options with "help plot."



Exercise Set 1 for parts I and II.

1. Turn on Matlab. Pull down the Desktop tab. Make it so that you only view the Command window. Then return to the default layout.

2. Perform the following calculations. Answers should display four decimal places.

i. Compute $\left(\frac{\pi}{5}\right)^3$

ii. Assign the value *e* to the variable *a* and then compute $\sqrt{a+2}$.

iii. Bring back the previous command to the prompt >> using the up arrow and assign the value obtained in part ii to the value *b*. Then compute $b^2 - 2$.

iv. Compute ln(10) and $\log_{10}(1000)$

3. Assign the vector [2 3 4] to the variable *v* and the vector [-1 0 ½] to the vector *t*. Compute *v***w*. (You should get an error message.) Now compute *v* .* *w* and *v*.^*w*. What's the difference?

4. Make a function called fred where if *y* = fred (*x*), then $y = x^3 - x$. Plot fred on the interval *x* from -2 to +2 with increments of 0.05. Then edit fred so that $y = x^3 + x$ and replot. Remember to save fred.

Using M-Files:

5. Close Matlab and reopen. This time, we will do most of our work in an M file. So open a new M-File. (Go to File, New, M File.) When the Editor pops up, the first thing to do is to save the document. I like to use something like Session(Date) or a short temporary nickname like "today." Instead of working in the Command Window, work in the M file so that you can edit much more easily and have a permanent record of your session that can be reactivated. A typical small M File might look like the following:

```
1- format short
2- format compact
3- clear
4- a=sin(pi/6)
5- b=cos(pi/15)
```

You edit it just as you would a word document. If you want to check you corrections and edits, save the document and then type its name in the Command Window. The command sequence will be executed. Your job: make and M File and do some computations with it. Experiment!!!!!

III. Matrices and Vectors.

"Matlab" is short for Matrix Lab. It's all about matrices. Every function and/or operation is performed on matrices, whether you know it or not. Mostly, you know it!

So this is a brief introduction to the most important part of Matlab. It is just the tip of the iceberg.

a. Forming matrices and learning about them.

First we enter the matrix $\begin{bmatrix} 2 & 5 & 3 \\ 5 & 9 & 0 \end{bmatrix}$ and assign it to the variable A.

```
>> A=[2 5 3;5 9 0]
A =
    2    5    3
    5    9    0
```

The entries are separated by the space bar (or by commas) and the rows are separated by the semicolon. Note that when it responds, Matlab omits the brackets. The transpose of a matrix interchanges row and columns. The transpose of A is A'.

```
>> A'
ans =
    2    5
    5    9
    3    0
```

So to obtain A, you could alternatively have entered:

```
>> B=[2 5; 5 9; 3 0]
B =
    2    5
    5    9
    3    0
A=B'
A =
    2    5    3
    5    9    0
```

Now let's find out about A and B.

i. To obtain the dimensions of A, use **size(A):**

```
>> size(A)
ans =
    2    3
```

Matlab returns a vector [rows of A, columns of A].

ii. To obtain the element in the *i*th row and *j*th column of A, use A(i, j). For instance, A(1, 2) gets the entry in the first row, second column of A.

```
>> A(1, 2)
ans =
     5
```

We can access the row and column sizes of A as follows by assigning size to a variable. The variable will be a vector i.e. a 1×2 matrix. We can then extract the components.

```
>> s=size(A)
s =
     2          3
>> s(1)
ans =
     2
```

iii. **The magic colon : is one of Matlab's most versatile features!!!!!!**

Alone, colon means "all."

So A(2, :) returns the second row and **all** the columns.

```
>> A(2,:)
ans =
     5          9          0
```

When you see **i : j,** the colon means all entries **from** the i^{th} **to** the j^{th} entry. So A(1,1:2) will extract the entries from the 1^{st} to the 2^{nd} column in the first row.

```
>> A(1,1:2)
ans =
          2          5
```

When you see **i:m:j**, the middle term *m* is the increment. So A(1, 1:2:3) will return the 1^{st} and 3^{rd} entries in the first row because we will skip by 2's.

```
>> A(1, 1:2:3)
ans =
     2          3
```

Using a increment of -1 will list elements in reverse order. So A(1, 3:-1:1) lists the first row of A in reverse order.

```
>> A(1, 3:-1:1)
ans =
     3          5          2
```

We can build new matrices from parts of old matrices.

```
>> C=A(1:2,2:3)
C =
     5     3
     9     0
```

The **empty** matrix is denoted by [], a bracket pair with nothing in it. When you set something equal to [], it gets deleted and causes the matrix to collapse to what remains. In the following, we create a 3×3 matrix M by using the command rand(3) which creates a 3×3 with random entries between 0 and 1. (Go to the help menu to find out more.) We then delete its bottom row.

>> M=rand(3) by setting the bottow row M(3, :) equal to [ ] .

```
M =
   0.4447   0.9218   0.4057
   0.6154   0.7382   0.9355
   0.7919   0.1763   0.9169
>> M(3,:)=[]
M =
   0.4447   0.9218   0.4057
   0.6154   0.7382   0.9355
```

Example: Row operations

We will perform some row operations on the matrix A:

```
0   2   3
2   4   6
3   1   5
```

i. First we swap rows 1 and 2. The trick is to first assign those rows to vectors x1 and x2 and then replace the rows of A.

```
>> x1=A(1,:)
        x1 =
             0     2     3

>> x2=A(2,:)
        x2 =
             2     4     6
>> A(1,:)=x2

A =
     2     4     6
     2     4     6
     3     1     5

>> A(2,:)=x1
A =

     2     4     6
     0     2     3
     3     1     5
```

ii. Now we **multiply the new first row** 1 by ½:

```
>> A(1,:) = 1/2*A(1,:)
        A =
             1     2     3
             0     2     3
             3     1     5
```

iii. Now we replace row 3 by row 3 -3*row(1).

```
>> A(3,:)=A(3,:)-3*(A(1,:))
       A3 =
            1     2     3
            0     2     3
            0    -5    -4
```

Other ways to build matrices and vectors.

i. We can use the colon to produce a row vector of equally spaced numbers and then use the transpose to get a column vector.

```
>> x = (3:2:9)
x =
     3     5     7     9
>> x=(3:2:9)'
x =
     3
     5
     7
     9
```

ii. We can join appropriately sized matrices and vectors. In the next example, we augment a 3×2 matrix A with a column vector.

```
>> A=[1 2;2 3;5 6]
A =
     1     2
     2     3
     5     6
>> b=[2 -1 0]'
b =
     2
    -1
     0
>> C=[A,b]
C =
     1     2     2
     2     3    -1
     5     6     0
```

Now add a row to the bottom of A. Note the use of the semicolon versus the comma.

```
>> r=[11,12]
r =
    11    12
>> D=[A;r]
D =
     1     2
     2     3
     5     6
    11    12
```

The command **rand(m,n)** will produce a matrix of the indicated size with random entries. Use the help menu to see how to control the entries.

```
>> rand(3,4)
ans =
   0.4057   0.4103   0.3529   0.1389
   0.9355   0.8936   0.8132   0.2028
   0.9169   0.0579   0.0099   0.1987
```

The command **eye(n)** produces the $n \times n$ identity matrix. (Sorry about the pun!)

```
>> eye(2)
ans =
    1    0
    0    1
```

The command **diag** is a bit tricky. If you have a matrix, it will extract its diagonal a put it into a column vector.

```
>> a=rand(3)
a =
   0.6038   0.0153   0.9318
   0.2722   0.7468   0.4660
   0.1988   0.4451   0.4186
>> diag(a)
ans =
   0.6038
   0.7468
   0.4186
```

If you have a vector, it will place it in the diagonal.

```
>> diag([1 2 3])
ans =
    1    0    0
    0    2    0
    0    0    3
```

The command does more. Please refer to the help menu.

The command **zeros(m,n)** produces an $m \times n$ matrix with all zero entries.

```
>> zeros(2,3)
ans =
    0    0    0
    0    0    0
```

IV. Matrix Operations.

We add, subtract and multiply by constants as you would expect. So let

```
a =
     1     3
     2     0
b =
    -1     5
     2     1
```

Then:

```
>> 5*a
ans =
     5    15
    10     0
>> a+b
ans =
     0     8
     4     1
```

Since a and b have compatible sizes, we can **multiply** them:

```
>> b*a
ans =
     9    -3
     4     6
```

Since a is square, we can find its **determinant** and, if it is not 0, we can find the inverse of a**:**

```
> det(a)
ans =
    -6
>> inv(a)
ans =
     0     1/2
     1/3   -1/6
```

If we let x = [1, -1], and try to multiply a*x, we get an error message because x is a row vector. So we take its transpose:

```
>> x=[1 -1]
```

```
x =
     1     -1
>> a*x
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

The last phrases were Matlab's error message. We couldn't multiply because the dimensions were not compatible. So let's correct it.

```
>> a*x'
ans =
    -2
     2
```

But we can multiply on the right:

```
>> x*a
ans =
    -1     3
```

We can raise square matrices to various powers:

```
>> a^3
ans =
    13     21
    14      6
```

Component wise operations: When preceded by a period, operations like * and ^ behave component wise. For instance, multiplies a(1,1) by b(1,1), etc. and a . ^ b will raise a(1,1) to the b(1,1) power.

```
> a.* b
ans =
    -1     15
     4      0

>> a.^b
ans =
     1    243
     4      0
```

Additional Matrix commands:

The ULTIMATE matrix command: rref (reduced row echelon form)

```
>> rref(a)
ans =
     1     0
     0     1
```

Other commands:

rank (m) computes the rank of a matrix

norm(v) computes the norm of a vector v
dot(v,w) computes the dot product of two vectors.

Exercise Set 2

1. Enter the following matrices.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 3 & 9 \\ 4 & 2 & 1 \end{bmatrix},\ B = \begin{bmatrix} 1 & 0 \\ -1 & 5 \\ 2 & 6 \end{bmatrix}$$

   i. Display the entries $A_{2,3}$ and $B_{3,2}$.
   ii. Set *b* equal to the second column of *B*.
   iii. Set *c* = to the third row of *A*.
   iv. Set *C* = *A*|*b* i.e. *A* augmented with *b*.
   v. Set *x* equal to the transpose of the first column of *B*.
   vi. Set *D* equal to the matrix *A* with the row *x* adjoined at the bottom.
   vii. Set A1 equal to the matrix *A* with the last column deleted. (Recall that [] is the empty matrix.)

2. Use the matrices found in exercise 1 to compute the following:
   i. *A* B*
   ii. *b* + *c'* (*c'* denotes the transpose of *c*.)
   iii. *B'* A*
   iv. A^4
   v. A^{-1} (if possible!)

3. i. Use **diag** to construct a matrix with diagonal [1, -1, 2, 5, 6]
   ii. Use **rand** to create a 5×6 matrix with entries between 0 and 5.
   iii. Use **zeros** to create a vector with 6 entries, all zero.
   iv. Use **eye** to create a 6×6 identity matrix.

4. Let $A2 = \begin{bmatrix} 2 & 3 \\ -1 & 5 \end{bmatrix}$ and $B2 = \begin{bmatrix} 2 & 3 \\ 3 & 2 \end{bmatrix}$.
   i. Compute *A2*B2*.
   ii. Multiply component-wise using **.***
   iii. Raise each entry in *A2* to the corresponding power in *B2*.

5. Solve the system $\begin{cases} 4x-3y+2z-w=-5 \\ 2x+y-3z=7 \\ -x+4y+z+2w=8 \end{cases}$ by augmenting the coefficient matrix by the column of constants and using **rref.** Repeat the problem using $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ as the column of constants.

V. Basic Programming.

The three basic programming tools are the "for" statement, the "while"statements, and the "if else end" statements.

i. We use the "for" statements when we know how many times we want our program to loop. The format is as follows:

for **x = array**

commands

end

The commands are executed once for each column in the array. At each step, x moves to the next value in the array. In the next example, we add together every third number numbers from 1 to 34 to obtain the sum s = 1 + 4 + 7 +…+ 34. The array will be denoted 1:3:34. First we **initialize** the variable *s*. At each step we add the value of *x* to the sum *s*. Note: the use of semicolons suppresses the view of values--a good thing for long programs! So we ask for the value of *s* after the program is executed.

```
>> s=0;
>> for x=1:3:34
s=s+x;
end
>> s
s =
    210
```

ii. The "while" statement executes commands while a condition is true. Its format is as follows:

```
while expression
    commands
end
```

In the next example, we add the values subsequent numbers until the addition of the last numbers makes the sum reach 2000 or more. Note the initialization. The last number to be added is 63. Before adding 63, the sum had not reached 2000.

```
>> s=0;i=0;
>> while s <2000
i=i+1;
s=s+i;
end
>> s
s =
      2016
>> i
i =
   63
```

iii. The "if" statement executes a statement if a condition is true and allows alternatives in other cases. It can have various formats.

```
if expression
    command executed if expression true
end
```

OR

```
if expression
    command executed if expression true
else
command executed if expression false
end
```

OR

```
if expression 1
    command executed if expression1 true
elseif  expression 2
    command executed if expression 2 true

 elseif……
            ⋮
  else
     command if none of the other expressions are true
```

end

In the following example, we will generate 100 random numbers between 0 and 1 and find out how many of them have squares greater than 0.5 and how many do not. (Okay, so there are lots of better ways to do this.) But the outcome of the program will change every time it is executed. Noted that in this program the **if** statement is nested in a **for** state. Each needs its own "end." Also note that the script is how the program appears in an M-file. The file is saved to the Matlab Work folder under the silly name "iffy." To execute it, I just type iffy in the Command Window and then ask for m and j, which are the number of times the square is greater than or and less than 0>5 respectively.

Here's the M-file program

```
m=0;j=0;k=100
 %k is the number of samples, m is how many squares are above .5 and j
is
 %how many are below.
for i=1:k
    x=rand;
if x^2 >.5
m=m+1;
else
j=j+1;
end
end
m
j
```

Here's what I write in the Command window and what Matlab returns.

```
>> iffy
k =
  100
m =
   28
j =
   72
```

That's all for now!