

INTRODUCTION TO MATLAB
Short Course on Mathematics of Biological Complexity
L. J. Gross - June 2000

This is a very basic introduction to the elements of MATLAB. MATLAB is a mathematics package that allows you to easily solve many of the quantitative problems that arise in the life sciences. This document briefly describes some of the key elements in using MATLAB to (i) do descriptive statistics; (ii) matrix algebra; (iii) probability, and (iv) discrete difference equations. These are all topics that will be covered in some detail in the short course and this document is designed to just aid you in solving problems in these areas using MATLAB.

MATLAB can be used with a command line interface, or it can be used to read and execute sequences of commands (essentially a program) that have been previously saved as a file. The typical file is simply a text collection of commands, using a .m extension. A variety of such files will be used in different sections of this course, and are available on the drives of the machines in the lab, in q:\cclab\mathbio.

DESCRIPTIVE STATISTICS, REGRESSION, and CURVE FITTING:

Handling lists of data:

In many laboratory and field experiments, you collect lists of data associated with measurements of experimental outcomes. Typically, the ordering of the data matters in that it makes a difference as to what measurement is made first, second, etc. For example, if you are measuring leaf lengths and widths, you would want to maintain the order in which you collected the data so that you know what length to associate with what width. We do this mathematically by putting the data in a list in which the order matters - we call such a list a "vector". Suppose we have the following data:

```
Student number: 1  2  3  4  5  6
grade           : 60 80 100 90 70 10
```

In MATLAB you type:

```
SNUMBER=[1 2 3 4 5 6]
```

and MATLAB will then print out:

```
SNUMBER =
 1  2  3  4  5  6
```

then you type:

```
SGRADE=[60 80 100 90 70 10]
```

and MATLAB will print out:

```
SGRADE =
60  80 100  90  70 10
```

Then the vector SNUMBER contains the number of each student and the vector SGRADE contains the grade of the corresponding student. We can then use a variety of MATLAB commands to make calculations from these data. For example, if we want to find the arithmetic mean value of the above grades then all we have to do is type:

```
m=mean(SGRADE)
```

and MATLAB will print out:

```
m =  
68.3333
```

If we want to find the standard deviation of the grades then we type:

```
sdev=std(SGRADE)
```

and MATLAB will print out:

```
sdev =  
31.8852
```

NOTE: MATLAB is case sensitive (i.e small or capital letters make a difference - so be careful). Note as well that certain names in MATLAB are reserved for its use - so you would not want to name a variable "mean" or "std" - the program will not allow this.

In both the above cases, MATLAB has now assigned the above numerical values to the variables m and sdev, and you can use MATLAB to make any standard calculations using these - for example to square m and subtract 3 times sdev type:

```
m^2 - 3*sdev
```

and MATLAB will print out:

```
ans =  
4.5738e+03
```

You can use all the standard mathematical operators: + - * / ^

We have learned so far how to set up our data in vectors on MATLAB and how to do some simple statistics commands. Now we are going to see how we can do a linear regression using our data.

Another example: Here are some data of length and width of *Acer saccharum* leaves

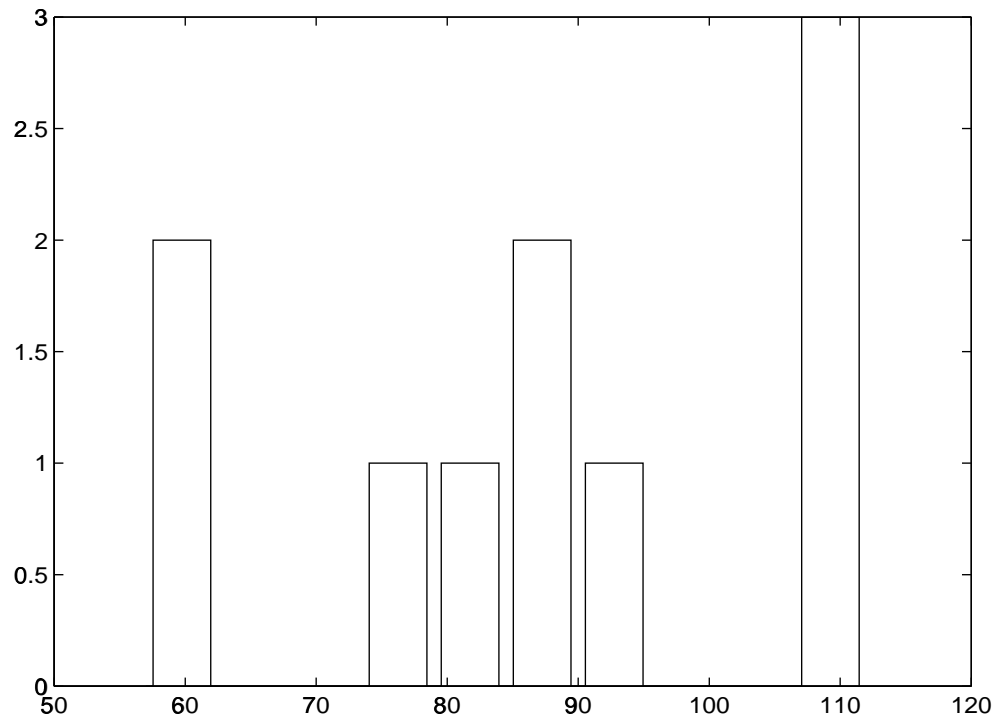
```
Length: 57 95 61 110 85 80 78 112 87 112  
Width : 70 105 78 120 99 89 99 125 105 123
```

let us store our data in vectors :

```
L=[57 95 61 110 85 80 78 112 87 112]  
W=[70 105 78 120 99 89 99 125 105 123]
```

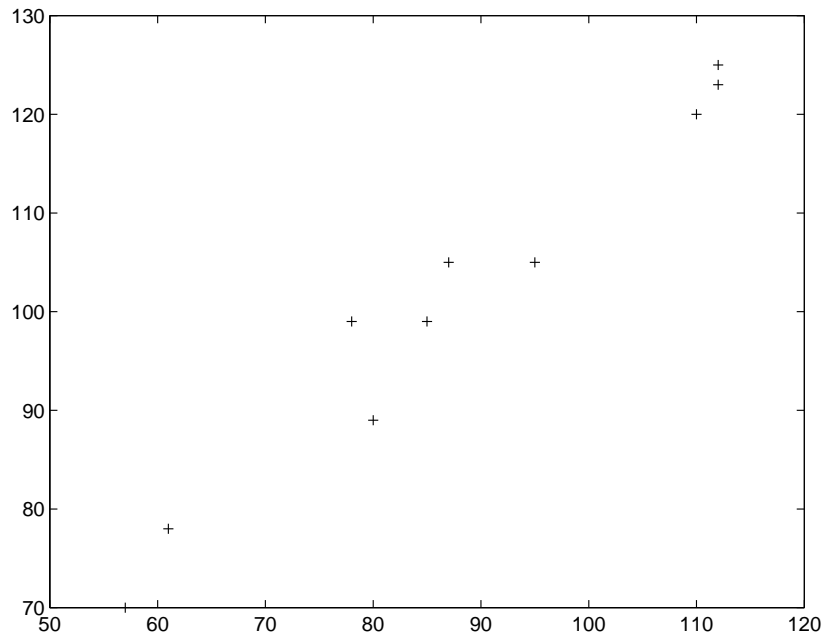
If we want the histogram of the length we type in:

```
hist(L) and hit the return key and you will see the following histogram.
```



To plot these points, with length on the horizontal axis and width on the vertical axis, using a + symbol for each of the data points we type in:

`plot(L,W,'+')` then hit enter and you will see the following graph:



Let us do a linear regression for the above data. We first calculate the regression coefficients:

```
C=polyfit(L,W,1) hit the return key and you will see
```

```
C =
```

```
0.9162 20.9460
```

This means that MATLAB has created a vector $C=[0.9162\ 20.9460]$ with the first number in C being the slope and the second number in C being the y-intercept of the regression line. To graph the above line we need to find at least two values for the polynomial $P(L)=0.9162*L+20.9460$ corresponding to any two values of L . Here $P(L)$ is the equation of the line with slope and y-intercept given by C .

Let us for example choose the following two values of L : 57 112 and let us store them in a vector in MATLAB:

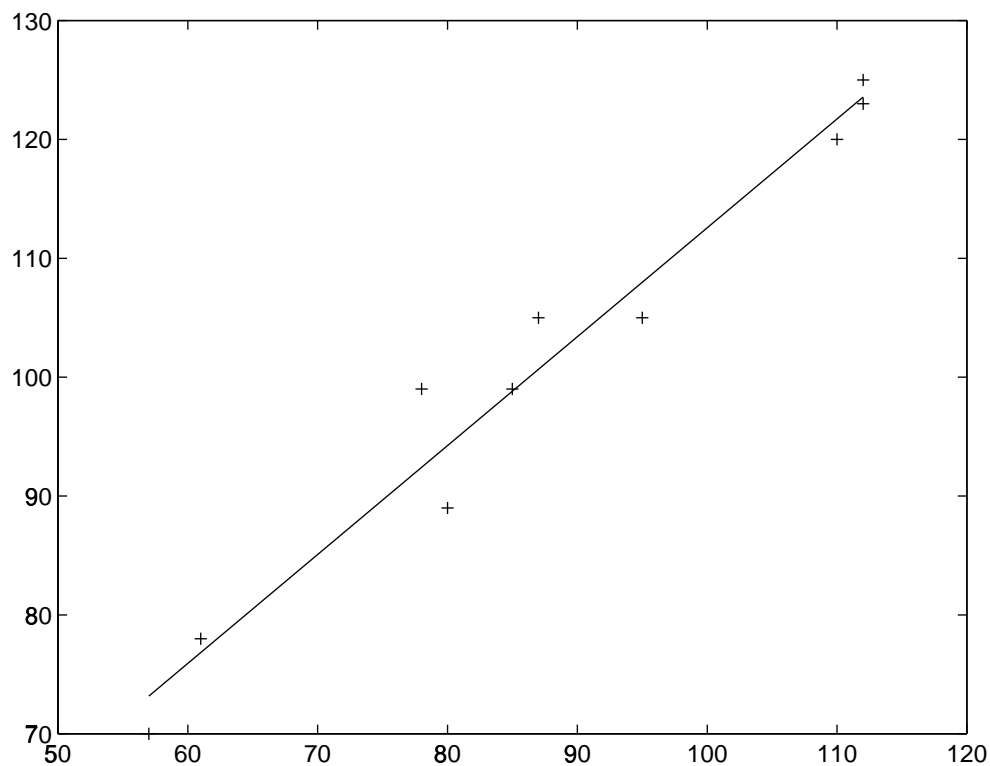
```
LTEMP=[57 112]
```

Then to find the values of $P(L)$ that correspond to those numbers we use the following command:

```
Y=polyval(C,LTEMP)
```

then type in:

```
plot(LTEMP,Y,L,W,'+') hit enter and you will see the following graph.
```



If we need to fit a higher degree polynomial to our data we change the number 1 to the degree of the polynomial we need - in general the command is `polyfit(L,W,n)` where n is the degree of the polynomial. For example, to fit a quadratic equation to a set of points, you would use `polyfit(L,W,2)`.

Saving data in MATLAB:

Before quitting MATLAB make sure that you save the data you need for future use. For example

suppose we want to save the data L and W. Then place a floppy in the drive a: and type in the following:

```
save a:myleafs.dat L W
```

then hit enter. This will save our data in a file called myleafs.dat on a floppy, then to quit MATLAB type in:

```
quit
```

In the future when you start MATLAB again, if you need to use the data L , W that are stored in the file math151.dat you need to load this file first. To do that place your floppy that contains the data from last time in drive a: and type in the following:

```
load a:myleafs.dat
```

then hit enter. To see L or W you just type in L or W and hit enter, then you will see the data in L or W. At any time you can see what variables MATLAB has stored by typing who

Using .m files:

MATLAB allows you to save sets of commands so that you don't have to retype them if you want to apply the commands to several different data sets. For example, you may want to look at histograms of leaf length and width using leaf data from several different species of trees. The sets of MATLAB commands are saved as files with a .m extension, and may be executed when you are in MATLAB by just typing the name of the file, without including the .m

For example, the below file is called lesson1.m - to use it you would first load in a file of leaf length and width data that you have saved on a floppy by typing:

```
load a:leafstat.dat
```

and then to run the program that is also on your floppy you type:

```
cd a:\
```

```
lesson1
```

where lesson1.m is the below file saved on your floppy.

Note that you can type dir to list the contents of your current directory.

```
%Lesson 1
%First read in leaf length and width data from a file
% which contains the data in the variables l and w
% load a:leafstat.dat
%if the file is on a floppy in the a: drive
%Then read in a command file that is
%this file of commands for MATLAB which you have
%saved in a file called lesson1.m on your floppy.
%First tell MATLAB to use the a: drive
% cd a:\
%then tell it to run lesson1
% lesson1
%If the file lesson1.m is not on your floppy but
%rather is on the computers in the lab, use instead
```

```

% cd q:\cclab\mathbio
%and then type lesson1
%
who,pause
plot(l,w,'+w')
title('l versus w'),pause
hist(l)
title('l'),pause
hist(w)
title('w'),pause
meanl=mean(l),meanw=mean(w),pause
plot(l,w,'+w',mean(l),mean(w),'or')
title('l versus w with mean'),pause
stdl=std(l),stdw=std(w),pause

```

The above program does the following after you have loaded in the file of leaf data. It pauses after each action and waits until you hit the return key.:

Plot the data
 Show a histogram of l
 Show a histogram of w
 Compute the means of l and w
 Plot the means of l and w as a point along with the data
 Compute the standard deviations of l and w

MATRICES IN MATLAB:

Defining a Matrix:

We saw above how to enter a set of data into MATLAB. In a similar way we can define a set of data that has more than one row which we call a matrix. You can think of a matrix as just a table of numbers, in which the position of a number in the table does matter.

Example: suppose we have the following matrix

$$M = \begin{bmatrix} 2 & 1 \\ 3 & 4 \end{bmatrix}$$

to define this matrix in MATLAB type in:

```
M=[2 1;3 4]
```

(make sure that each row except the last ends up with a ;) then hit return and you will see the following:

```
M= 2  1
    3  4
```

Suppose you want to form another matrix :

$$B = \begin{bmatrix} 1 & 3 & 2 \\ 1 & 5 & 0 \end{bmatrix}$$

then we type in:

```
B=[1 3 2;1 5 0]
```

(notice again that each row but the last ends up with a ;) hit the return key and you will see the following:

```
B= 1   3   2
    1   5   0
```

Saving Your Matrix:

Suppose we had to quit MATLAB and go home but we want to use the above matrices later. First insert a floppy in drive a: and type in the following

```
save a:filename M B
```

for example:

```
save a:test M B
```

then hit return and your matrices M and B will be saved on your floppy in the file called test. The next time you need to use these matrices you start MATLAB and you insert your floppy in a: and type in:

```
load a:test
```

hit return and this will load your file test into MATLAB. To actually see one of the above two matrices (for example to see M) type in:

```
M
```

hit return and you will see the following:

```
M=  2  1
    3  4
```

Matrix Operations:

Most algebraic operations work on matrices in MATLAB just as they do on scalar quantities (e.g. single numbers). Indeed MATLAB is designed to be a very powerful matrix manipulation program. Type in:

```
C=M+M
```

then hit enter and you will see the following:

```
C=4  2
    6  8
```

which is obtained by adding each element of M to itself. The same thing can be achieved by multiplying each element of M by 2, this can be done by typing in the following:

```
C=2*M
```

then hit enter you will see the same thing as above. To do matrix multiplication, type

```
D=M*B
```

hit enter and you will see that D contains the multiplication of M with B i.e:

```
D= 3  11  4
    7  29  6
```

similarly if we type in :

```
E=M-C
```

then E will contain the subtraction of each element of C from the corresponding element of M:

```
E= -2  -1
```

Finding Eigenvalues and Eigenvectors of a Matrix:

To find the eigenvalues and eigenvectors of the matrix M we type in the following:

```
[evec,eval]=eig(M)
```

hit return and you will see the following:

```
evec= -0.7071 -0.3162  
      0.7071 -0.9487
```

```
eval= 1    0  
      0    5
```

where each column of evec is an eigenvector of M and each diagonal element of eval is an eigenvalue of M. (Note: the names evec and eval are optional - you can choose any names you like instead of these).

Creating a .m file:

Instead of typing everything in MATLAB as we did above we could have created the following file in any editor which does exactly what we did above:

```
M=[2 1;3 4]  
B=[1 3 2;1 5 0]  
save a:test M B % make sure you have floppy in drive a:  
C=M+M  
C=2*M  
D=M*B  
E=M-C  
[evec,eval]=eig(M)
```

then save the above file as filename.m, for example go.m Then when you are in MATLAB and your floppy is in drive a: type in:

```
cd a:\  
go
```

when you hit enter you will see all the above commands being executed.

Help:

Help is available on any command of MATLAB and it is very useful, for example if you type in:

```
help eig
```

and hit return you will see all explanations you need on how to use the eig command. So if you are not sure on how to use a command in MATLAB use the help command for a brief explanation.

Note: Suppose A is an n x m matrix. How can we extract a column or a row from A and store it in a vector x?

```
example: let A= 1 0 5 4
                2 8 8 9
                1 0 1 9
```

Suppose we want a vector x to contain the second row of A, then type in:

```
x=A(2,:)
```

hit enter and you will see:

```
x= 2 8 8 9 (displayed as a row vector)
```

if we want x to contain the second column type in :

```
x=A(:,2)
```

hit return you will see:

```
x= 0 8 0 (displayed as a column vector)
```

The above command might be used when you have a matrix of data and say each row includes the measurements on leaves of a single tree and the different rows correspond to different trees. If you want the mean for the leaves of only the first tree then you will need the data in the first row only, so you'll need to use the above command to put the first row in a vector. Similarly, you could put the first column of the matrix in a vector, and the mean of it would give the mean for the first measurement of leaves on all trees (for example if the leaves were measured at different times during development, the mean of each column gives the average size of leaves at that developmental stage).

USING MATLAB TO CALCULATE PROBABILITIES BY SIMULATION:

You can use MATLAB to simulate coin tosses, dice tosses, card games, and so on through the use of its built-in function rand. The function rand numerically generates random numbers coming from one of several distributions. The simplest one is a uniform distribution - you can think of this as tossing a dart at a line segment from 0 to 1, with the dart being equally likely to hit any point in the interval [0,1]. This is easy to use to simulate coin tosses by setting the probability of getting a head to be p, calling the function rand, and if rand gives a number less than p, a head is said to occur, while if rand gives a number greater than p then a tail is said to occur. An exactly similar method applies to tossing a single die. In this case the cut-off values (for a fair die) would be 1/6, 2/6, ..., 5/6. This method of simulating probabilistic events is often called Monte-Carlo simulation.

NOTE: Calling rand(1,1) will return a single pseudo-random number between 0 and 1.

EXAMPLE: MATLAB code to simulate n tosses of a die, with probability of getting Heads on a single toss = p.

```
%coins.m
% This MATLAB code simulates n tosses of a coin with probability p
% of getting a Head and 1-p of getting a Tail. You are prompted to supply
% the probability p and the number of tosses. At the end, output will
% be the number of Heads and Number of Tails which occurred, stored
% in a vector out.
%
n=input('how many times to toss the coin: ')
```

```

p=input('probability of heads on a single toss: ')
% nh will count the number of heads, nt the number of tails
nh=0;
nt=0;
x=[]
for i=1:n
a=rand(1,1);
if a <= p
    nh=nh+1;
    x=[x 1];
else
    nt=nt+1;
    x=[x 0];
end;
end;
per=nh/n;
out=[nh nt];
disp('number and percentage of Heads is')
nh,per,pause
disp('number and percentage of Tails is')
nt,1.-per,pause
hist(x,2)
title('histogram of H (1) and T (0)'),pause

```

DIFFERENCE EQUATIONS IN MATLAB:

Difference equations are used to analyze situations in which the variables of interest (such as population size) are observed at regular, fixed time intervals, such as every month, year, generation, etc. The general first-order difference equation is $x(n+1) = f(x(n))$ where $x(n)$ gives the population size at generation n , n is an integer, and the function f specifies how the population size next generation depends upon the current population size. You will see in this course the properties of simple linear difference equations, but in general, if the function f is non-linear, there is no way to get an explicit solution - that is there is no way to get a formula that tells you what $x(n)$ is for any given n . Rather, what you have to do in most cases is use a computer to calculate $x(n+1)$ and then repeat this again and again - this is called iteration and MATLAB does it very easily. An example is given below for the case of the discrete logistic equation:

```

%Logistic.m - this MATLAB file solves the
%discrete logistic equation  $x(i+1)=r*x(i)*(1-x(i))$ 
%and the user is prompted to read in the value
% of r to use as well as the initial value
% x0 which must be in (0,1)
% and the time interval over which to run the

```

```

% simulation.
s=1;
while s>0;
r=input('input growth rate r: ')
x0=input('input initial population x0: ')
n=input('end of time interval b: ')
x=zeros(n+1,1);
t=zeros(n+1,1);
x(1)=x0;
for i=1:n
t(i)=i-1;
x(i+1)=r*x(i)*(1-x(i));
end
t(n+1)=n;
plot(t,x,t,x,'o'),pause
s=input('Do you want to stop - if so enter 0')
end

```

Here the line

```
for i=1:n
```

tells MATLAB to iterate the equation n times.