August 20, 2020 – BIOL/PHYS 6750 Matlab introduction – Prof. Joshua Weitz

This tutorial is meant to be interactive... that is *you* should be reading, typing, thinking, and asking questions. There are no source files to share, the point is for you to think about and enter the code and see the result. And then, to vary the code and see what changes. This is a more effective strategy than just *reading* in code others have written and running it. A few notes:

- If you are experienced in MATLAB, then please work on challenge problems identified in this tutorial.
- Please do talk to others via chat if you're stuck, but write your own code!
- You can do it!

1 Getting Started

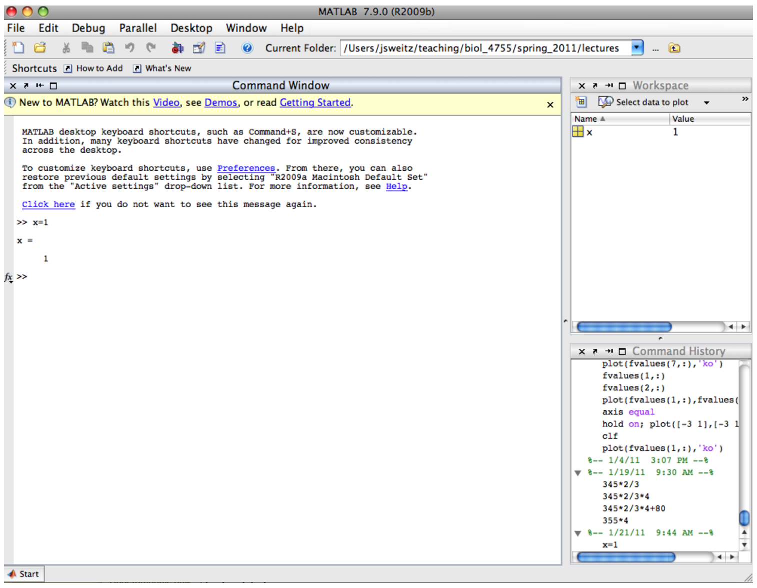
MATLAB is a numerical computing environment and is also a programming language. We will be using it in this class to simulate and model the dynamics of living systems from scales of molecules to ecosystems. MATLAB is particularly good at:

- Matrix manipulation
- Numerical methods
- Algorithms
- Graphical output

and is okay at:

- Symbolic computation
- Interfacing with other languages
- Speed of for loops (as compared to C)

It can also be used for interactive work using the MATLAB Live Editor (similar to the iPython notebook). So, let's get started! In today's class, you will gain practical experiencing using Matlab. When you open Matlab, you should see a set of windows that looks like this:


MATLAB 7.9.0 (R2009b)
File Edit Debug Parallel Desktop Window Help
Current Folder: /Users/jsweitz/teaching/biol_4755/spring_2011/lectures
Shortcuts How to Add What's New
Command Window
New to MATLAB? Watch this Video, see Demos, or read Getting Started.
Workspace
Select data to plot
Name
Value
x
1
Command History
Start

```
MATLAB desktop keyboard shortcuts, such as Command+S, are now customizable.
In addition, many keyboard shortcuts have changed for improved consistency
across the desktop.

To customize keyboard shortcuts, use Preferences. From there, you can also
restore previous default settings by selecting "R2009a Macintosh Default Set"
from the "Active settings" drop-down list. For more information, see Help.

Click here if you do not want to see this message again.

>> x=1

x =

     1

>>
```

```
    plot(fvalues(7,:),'ko')
    fvalues(1,:)
    fvalues(2,:)
    plot(fvalues(1,:),fvalues(
    axis equal
    hold on; plot([-3 1],[-3 1
    clf
    plot(fvalues(1,:),'ko')
  %-- 1/4/11  3:07 PM --%
  %-- 1/19/11  9:30 AM --%
    345*2/3
    345*2/3*4
    345*2/3*4+80
    355*4
  %-- 1/21/11  9:44 AM --%
    x=1
```

The key elements are the Command Window, Workspace and Command History. The command window is where you enter commands, and you should see a prompt that looks like this:

```
  >>
```

You can do basic math at this prompt, for example

```
>> 3+4
ans =
     7
```

and

```
>> exp(1)
ans =
    2.7183
```

Matlab has **many** functions that are built in (and you can use it like a calculator), for example, to learn more about the exponential function:

```
>> help exp
 EXP    Exponential.
    EXP(X) is the exponential of the elements of X, e to the X.
    For complex Z=X+i*Y, EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y)).

    See also expm1, log, log10, expm, expint.

    Overloaded methods:
       lti/exp
       codistributed/exp
       fints/exp

    Reference page in Help browser
       doc exp
```

Many functions have names that you expect (how do you think you should calculate cosine or sine of a value, for example)? Try it out! If you don't know the name of the function you can use the command `lookfor` or help.

Matlab is not just a calculator. It is also a programming language that can store values in memory. For example, the command

```
>> x=3
x =
     3
```

tells Matlab that the variable x has the value 3 and now every time you use "x", Matlab will substitute the value 3, for example:

```
>> y=x+1
y =
     4
```

It is very important to realize the "=" sign does not mean that Matlab checks to see if the two sides are equal to each other, but rather states that whatever is on the left should be assigned the value of that on the right. If you want to check the truth of a particular statement, for example is x equal to 3, or alternatively, is y equal to 3 then you would type:

```
>> x==3
ans =
     1
>> y==3
ans =
     0
```

The double "==" sign tells Matlab to logically compare what is on the left with that on the right and return 1 if true and 0 if false. Note that if you don't want Matlab to report back the answer every time, you can use the semicolon

```
>> y=x+1;
```

Matlab can also handle arrays of values (for example a vector or a matrix). The simplest way is to use the colon, which defines a sequential vector, for example

```
>> v=1:5
v =
     1     2     3     4     5
```

you can also modify the increments by adding a step in the middle

```
>> w=1:2:9
w =
     1     3     5     7     9
```

Any entry can be examined using the parentheses

```
>> w(3)
ans =
     5
```

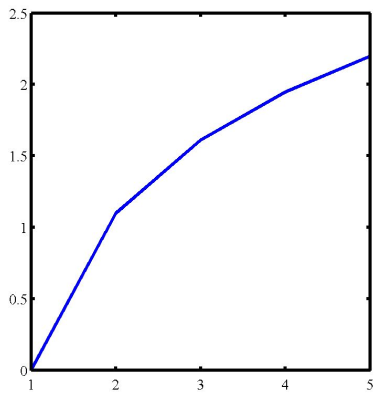
and basic math can be performed automatically on vectors (and matrices), for example

```
>> log(2*w)
ans =
    0.6931    1.7918    2.3026    2.6391    2.8904
```

Matlab can also plot graphs and surfaces and all sorts of things. To create a simple plot, use the plot command

```
>> plot(v,log(w))
```

which leads to:



2 Building "Programs" from "Scripts" and "Functions"

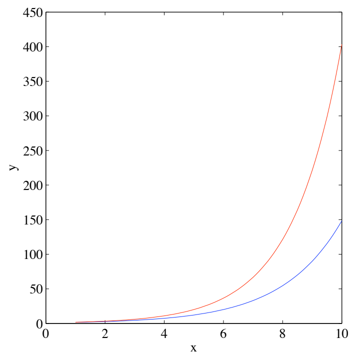
However, once you have a lot of commands, it will get exhausting typing them (especially when you make mistakes). So, Matlab includes the script concept. A script is a list of commands in a file that you can execute directly from the command window. To create a script go to the File menu and select New > M-file. Now write a few commands, such as:

```
% my_first_file.m
% Create some vectors
x =1:0.1:10;
y1 =exp(0.5*x);
y2 =exp(0.6*x);
% Plot the vectors
figure  %Pull up a new figure
hold on %Allow multiple plots on the same figure
plot(x,y1)
plot(x,y2,'r') % Use a red line
xlabel('x'), ylabel('y')  %Label the axes
print -dpdf my_first.pdf %Save the image to a file
```

Save this file as my_first_file.m in the same folder you're currently working in. Now go to the Matlab prompt and type

```
>> my_first_file
```

and it should execute the commands in the script to yield the figure:



The problem with this script is that changing the exponents in these files requires editing the script and then re-running it, instead of designating a variable change from the command window. Moreover, a script cannot return a variable or accept a variable as input. To do so requires the concept called a "function". Functions are program files that can be called from the Command window. To start one, open a new M-file and type:

```
function dNdt=logGrowth(t,N)
% function dNdt=logGrowth(t,N)
%
% logGrowth gives the growth rate of a population of size N at time t
% Usage:dNdt=logGrowth(t,N)
r =0.5;
K =100;
dNdt =r*N.*(1-N/K);
```

Now this new function can be acccessed just like one of Matlab's, for example:

```
>> vN =0:110;
>> figure
>> plot(vN, logGrowth(0,vN))
>> xlabel('N'),ylabel('dN/dt')
```

gives an upside down parabola, denoting that growth rate is positive between 0 and 100 and negative when N is greater than 100.

3 Getting Started with Core Techniques

3.1 Create loops

The two types of loops we will discuss are “for” and “while” loops. Both loop start with a keyword such as for or while and they end with the word end. The “for” loop allows us to repeat certain commands many times with a “counter” variable. Here is one example:

```
>> for j=1:4 % counting
     j+2 % the statement you want to repeat
   end
```

You can also increment your counter variable by any real number like

```
>> for j=1:-0.1:0 % counting
     1-j % the statement you want to repeat
   end
```

Or you can even use a set of arbitrary values:

```
>> for j=[1 3 5 7]
     j-1 % the statement you want to repeat
   end
```

Challenge Problem: Exercise on Matrices

Define a random matrix A of size 3-by-3. Use a double ”for loop” to calculate the square of the entries in A and store the values in another matrix B. (Hint: type ”help rand” in Matlab if you don’t know how to define a random matrix)

The second type of loop is the “while” loop. The `while` loop repeats a sequence of commands as long as some condition is met. For example, given a number n, the following code will return the smallest non-negative integer a such that $2^a \geq n$.

```
function a = smallexp(n)

a = 0;
while 2^a < n
  a = a + 1; % statement to execute if the condition is met
end

 a = smallexp(4)

a =
     2
```

Note that in the above example we used the conditional statement, $2^a < n$ to decide whether the statement within the while loop should be proceeded. Such conditional statements are also used in ”if” statements that are discussed in the next section. The relational operator used here is ”<”, which means ”less than.” Other relational operators that are available in Matlab include:

```
>  greater than
<= less than or equal
>= greater than or equal
== equal
~= not equal
```

Simple conditional statements can be combined by logical operators

```
(&&, ||, ~)
```

into compound expressing such as the following:

```
(5 > 1) && (5 == 6)
```

3.2 Make decisions

Now, let's suppose you want your code to make a decision, and the "if" statement is what you need. The general form in Matlab is as follows:

```
if expression1
  statements1
elseif expression2
  statements2
else
  statements3
end
```

Challenge Problem: Recursive Factorial

Finish the following pseudo-code that gives the factorial of a positive number n, using the recursive formula $n! = n(n-1)...1$

```
function N = factorial_recur(n)

if #
    N = 1;
else
    N = #;
end
```

3.3 Go fast, i.e., "vectorize"

Remember: `for` loops are SLOW in Matlab. One way to make your Matlab run faster is to `vectorize` the algorithm you use in the code. Vectorization can be done by converting for and while loops to equivalent vector or matrix operations. A simple example would be the following (tic and toc are used as a stop watch):

```
x = 1;
tic
for k = 1:1001
   y(k) = log10(x);
   x = x + .01;
end
toc

tic
x = 1:.01:11;
y = log10(x);
toc
```

You should find that the second command set is faster, even if it returns exactly the same output. However, for more complicated code, vectorization is not always so obvious. Some of the most commonly used functions for vectorizing can be found in the Help browser of Matlab.

3.4 Find values you want to know about

Given an array X, the command

```
>> find(X)
```

returns the indices of all nonzero elements of X. You can also use a logical expression to define X. For example,

```
>> find(X>2)
```

returns the indices corresponding to the entries of X that are greater than 2. Notice that X could also be a matrix. In this case, the function returns the linear indices as if the matrix was stored as a single column of elements (Try it!).

However, when you want to locate the entries that satisfy more than one logical expressions, the 'elementwise' logical operators (& and I) are used in place of 'short-circuit' logical operators (&& and II).

Challenge Problem: Finding Elements in Matrices

Next, build a 5-by-5 random matrix A using the command

```
>> A = rand(5)
```

Find the indices of entries whose value is smaller than 1/4 and bigger than 1/6. Check the answer with your virtual neighbors.

3.5 Save and load data

The keywords in Matlab are straightforward: save and load. Type

```
>> save filename
```

and Matlab will save all variables in the current workspace in the file filename. If you do not specify an extension to the filename, Matlab uses .mat. When you want to load all the variables from the file specified by filename, just type

```
>> load filename
```

Please keep in mind that these file formats are binary, i.e., not human-readable. If you want to save a particular file format use the help to save variables in standard, comma-separated value file format. There are also alternative ways to load and print data.

4 Numerically Integrating Differential Equations

In this class we can use Matlab to numerically solve differential equations, like the logistic growth equation, even when such solutions are not available analytically. The most-used Matlab program which does the integration is called ode45. Here is a script that integrates the logGrowth function. We will return to this multiple times in the course.

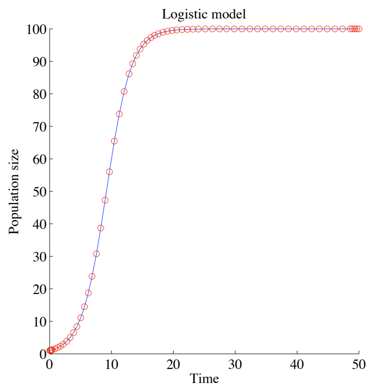
```
% Numerical solution of the logistic equation
t0 =0; % Initial time
tf =50; %Final time
N0 =1; %Initial population size
```

```
[T, vNint] = ode45(@logGrowth, [t0 tf], N0); %Numerically integrate
% Actual solution
r =0.5;
K =100;
vNact =(N0*exp(r*T))./(1+N0*(exp(r*T)-1)/K); %Actual solution
% Plot results
figure;
hold on
plot(T,vNint) % Plot numerically intregrated solution
plot(T,vNact,'ro') %Plot actual solution
xlabel('Time'), ylabel('Population size'),title('Logistic model')
print -dpdf logPlot.pdf
```

Save this script as intLog.m and run it

```
>> intLog
```

and then get the plot



Here are some important points to keep in mind:

- Matlab solves ordinary differential equations of the form

$$\frac{d\vec{y}}{dt} = \vec{f}(\vec{y})$$

  where $\vec{f}(\vec{y})$ is a vector of functions.

- Main solver you should use is: **ode45**.

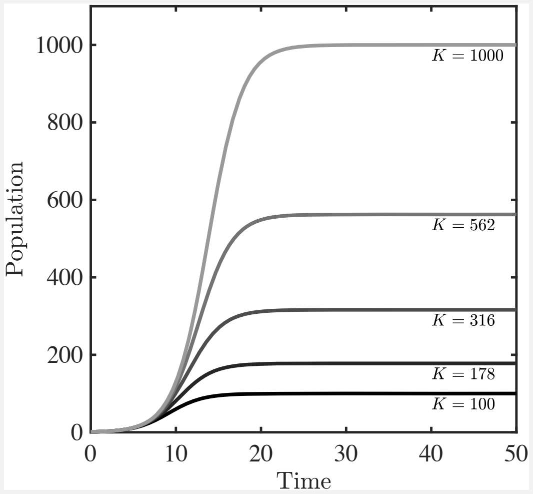
$$\text{[TOUT,YOUT]} = \text{ODE45(ODEFUN,TSPAN,Y0)}$$

  1. TOUT $\rightarrow$ time of output
  2. YOUT $\rightarrow$ value of variable
  3. ODEFUN $\rightarrow$ name of function containing the dynamics
  4. TSPAN $\rightarrow$ time limits for integration
  5. Y0 $\rightarrow$ initial conditions

- help ode45 for more information

Challenge Problem: Variables and Differential Equations

Read the `ode45` documentation and modify your logistic growth model to accept r and K as parameters. Vary the value of K over 1 order of magnitude and compare the results. If your code works, it should look something like this:



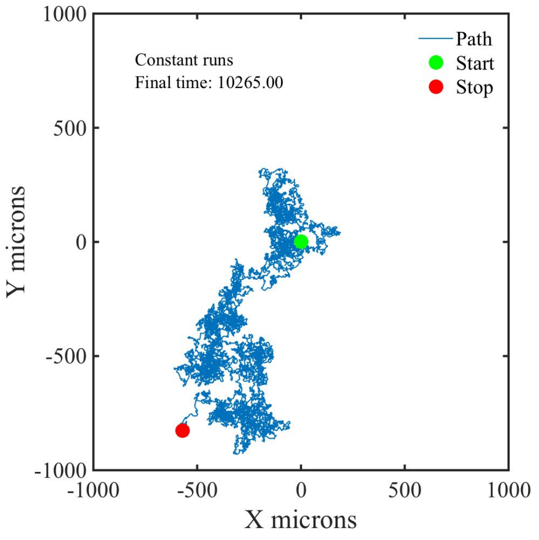
5 Advanced - Individual-Based Simulations

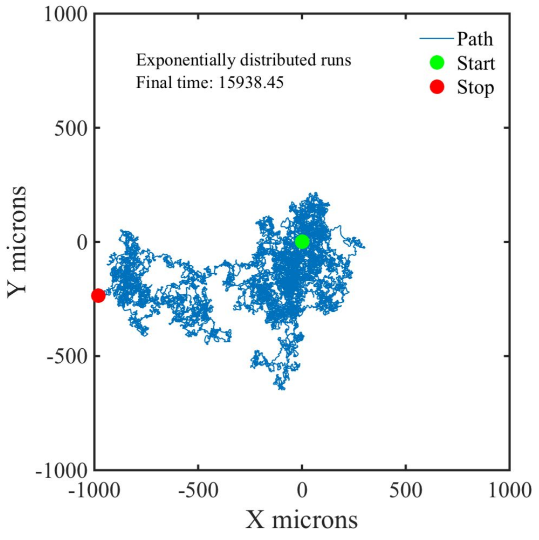
For those more experienced in MATLAB, try and develop a simulation of "diffusion" in which a bacteria swims at a speed of 10 μm/s with each "run" lasting 1 second. In this case the direction of the next run is random. How long will it take, on average, for the bacteria to travel 1mm away from its source? Recall that in class we said that the average time to travel a distance x should be:

$$T = \frac{x^2}{D} \tag{1}$$

What do we mean by *average* here?

If you develop code, can you visualize the results? If you change the length in equation, how does T change? Can you confirm the scaling and estimate D? What happens if the durations of each run is exponentially distributed, so that runs last on average 1 second, but can vary in duration? Did the answer with respect to travel time to reach 1mm change in a quantitative or qualitative way? We will get much further into these ideas later in the class. Here are examples of two runs, one with constant duration and one with exponentially distributed durations:


Path
Start
Stop
Constant runs
Final time: 10265.00
1000
500
0
-500
-1000
-1000
-500
0
500
1000
X microns
Y microns


Path
Start
Stop
Exponentially distributed runs
Final time: 15938.45
1000
500
0
-500
-1000
-1000
-500
0
500
1000
X microns
Y microns