Lab Assignment 1

Modeling the Harvest of Bluefish on the Chesapeake Bay

Written by

Prof. Erin N. Bodine

Prepared for

Math 214: Discrete Math Modeling

Rhodes College

Last Edited: December 6, 2018

Bluefish (*Pomatomus saltatrix*) is a species of saltwater fish that can be found around the world in temperate and subtropical waters. The harvesting (fishing) of bluefish is primarily recreational as the bluefish is a prized *sport fish* since they can live up to 12 years and grow up to 40 inches long. In the 1990s the amount of bluefish present along the mid-Atlantic seaboard dropped to very low levels after several decades of over-fishing/over-harvesting.

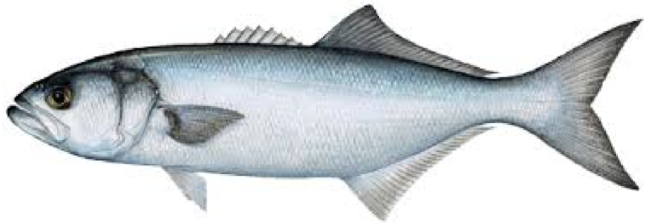


Figure 1: Illustration of a bluefish (*Pomatomus saltatrix*).

Table 1: Estimated harvest of bluefish from Chesapeake Bay from 1940 – 1990.

Year	Bluefish (lbs)
1940	15,000
1945	150,000
1950	250,000
1955	275,000
1960	270,000
1965	280,000
1970	290,000
1975	650,000
1980	1,200,000
1985	1,500,000
1990	2,750,000

In 1992, the *Daily Press* newspaper (serving southeastern Virginia) published data reporting the estimated yearly amount of harvested bluefish (in pounds) from 1940 to 1990, reported every 5 years (see Table 1 for the data). The data showed how the amount of harvested bluefish had increased exponentially over the 50 year period.

The goals of this lab assignment are to:

- Estimate the proportion by which harvesting of bluefish from Chesapeake Bay increased every 5 years.
- Model the exponential growth of bluefish harvesting from Chesapeake Bay using a discrete exponential growth model.

In this lab, we will be using Matlab to help achieve these goals. Thus, as a part of this lab you will learn how to do the following in Matlab:

- Create an m-file.
- Generate a graph.
- Write a for-loop.

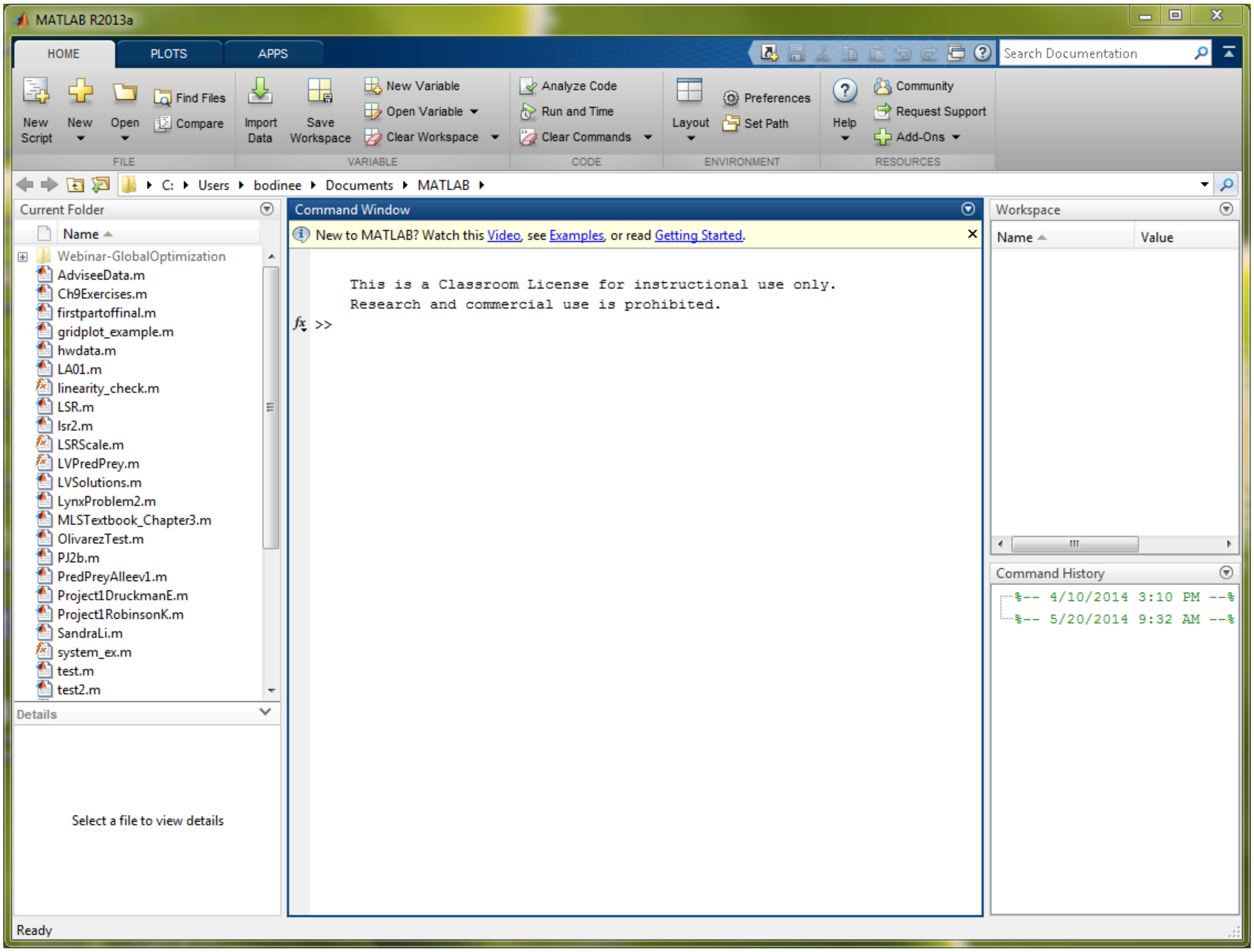
Matlab Skills

We are going to start out with a tour of Matlab. If you have never used Matlab before this will help you become familiar with the layout of the Matlab interface, and how to complete some basic tasks in Matlab.

Tour of Matlab

1. In Windows, go to the start menu and find Matlab . The icon for Matlab looks like , and the program will likely be listed as Matlab R2016a (or a more recent year if the software has been updated). Click on the program to open it. It make take several seconds before opening.
2. Once Matlab is open you should see a Matlab interface window with a menu bar across the top and several subwindows below. See Figure 2 for a sample screen shot right after opening Matlab. We will start by familiarizing ourselves with the elements contained within this Matlab interface window. Figure 3 shows the

same sample screen shot as in Figure 2 but indicates where various features of the MATLAB interface window are located.



New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
  This is a Classroom License for instructional use only.
  Research and commercial use is prohibited.

>>
```

```
%-- 4/10/2014 3:10 PM --%
%-- 5/20/2014 9:32 AM --%
```

Select a file to view details

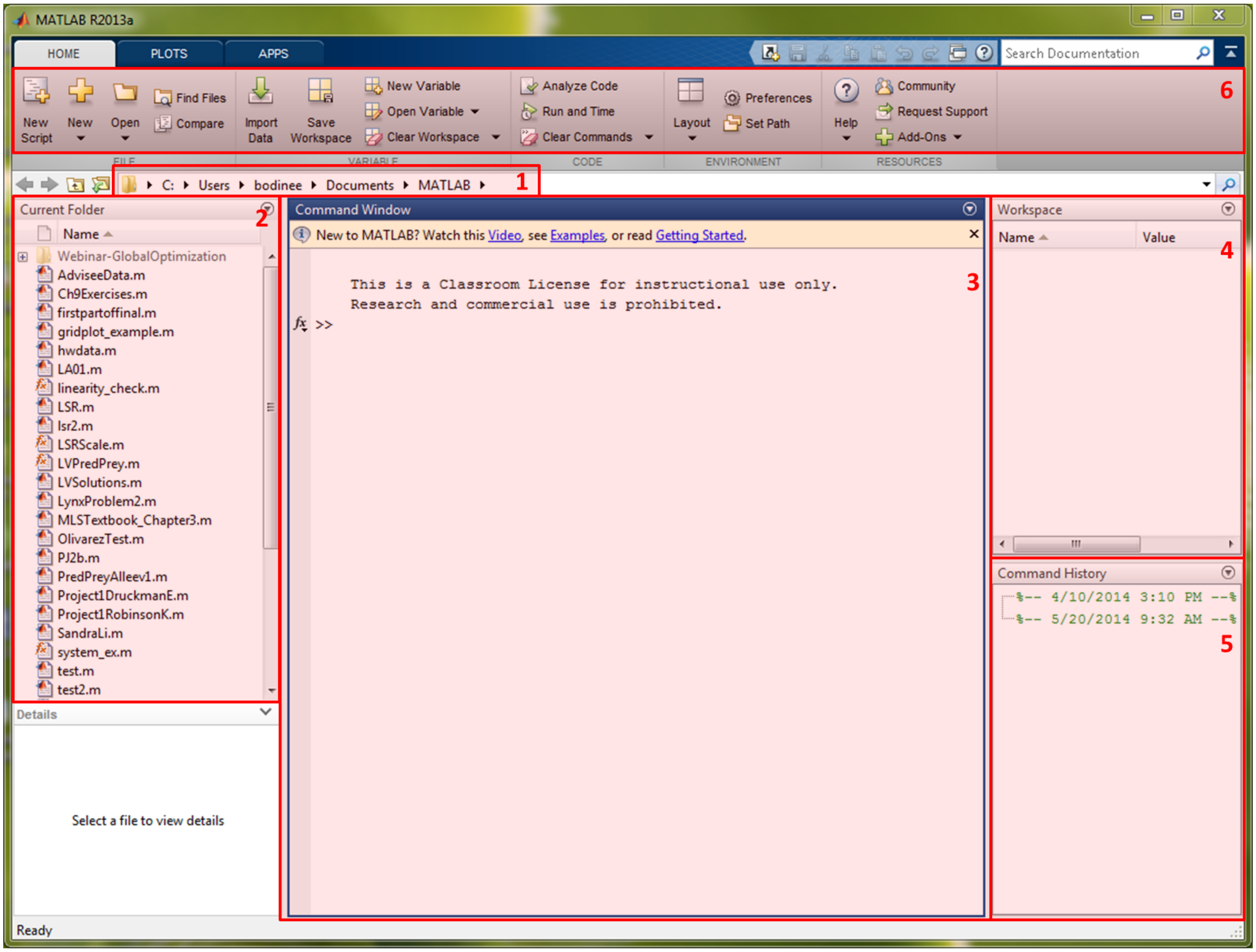
Figure 2: Screen shot of MATLAB right after it has been opened.

3. Figure 3 uses numbered red boxes to highlight several key elements of the MATLAB interface window.

   (1) *The location of the current directory or folder.* As we work with Matlab we will want to create files that store our data and execute various commands; these are called m-files (which stands for MATLAB files). In order to access the m-files we create, we need to indicate to MATLAB where these files are located. Red box (1) shows where we set the current folder. In Figure 3 the current folder is set to `C:\Users\bodinee\Documents\MATLAB`. To change the current directory click on the "Browse for Folder" icon next to the current directory location bar, and select the location you desire. Alternatively, you could click anywhere on the current directory location bar and type in the location of the folder you desire.

   (2) *The files contained within the current folder.* In the current folder shown in Figure 3, there are many files listed. Note the file naming structure, `filename.m`, where the `.m` indicates each file is an m-file. If you look closely you will see two different icons used for the m-files. This is because there are two different types of m-files: *script m-files* and *function m-files*. A ***script m-file*** executes a series of commands.

A ***function m-file*** can accept input arguments (or variables) and produce output variables (thus making it similar to a function as we know it in mathematics). In this lab assignment we will learn how to create script m-files, but later in the semester we will also learn how to create function m-files.



New to MATLAB? Watch this Video, see Examples, or read Getting Started.

```
This is a Classroom License for instructional use only.
Research and commercial use is prohibited.
fx >>
```

```
%-- 4/10/2014 3:10 PM --%
%-- 5/20/2014 9:32 AM --%
```

Figure 3: Screen shot of MATLAB right after it has been opened. The numbered red boxes show the following elements of the MATLAB interface window: (1) the location of the current directory or folder, (2) files contained within the current folder, (3) the command window, (4) the workspace, (5) the recent command history, and (6) the menu bar.

(3) *The command window.* This is the window where commands are executed. To get MATLAB to execute a command, you type the command into the command window and press "Enter", then the command is executed and any results are displayed either within the command window or in a separate window if a graph is generated. We will try typing several different commands into the command window after introducing the key elements of the MATLAB interface window.

(4) *The workspace.* The workspace shows all the variables you have created in a MATLAB session. Note, once you close out of MATLAB , all the variables in the workspace will be cleared (i.e., deleted). The variables you generate can be real numbers, or matrices of numbers (if you are not yet familiar with what a matrix is, think of it as a grid of numbers). Note, though we usually think of matrices as two-dimensional grids of numbers, MATLAB has the ability to handle multi-dimensional matrices.

(5) *The recent command history.* This subwindow will keep a running tab of all the commands you have typed into the command window. If you typed a complex command and want to execute it again, you can scroll through the command history, double click on the command you want to execute again, and it will execute in the command window again. Note, that you can also scroll through all previous commands executed in a MATLAB session by using the up and down arrows on the keyboard (when working in the command window). This is often easier, and makes the presence of the command history subwindow a bit redundant. If you would like to close the command history subwindow, click on the small circled down arrow in the top right corner of the subwindow and select close.

(6) *The menu bar.* The MATLAB menu bar has many different icons, all of which are labeled. We will use be using some of this in this lab.

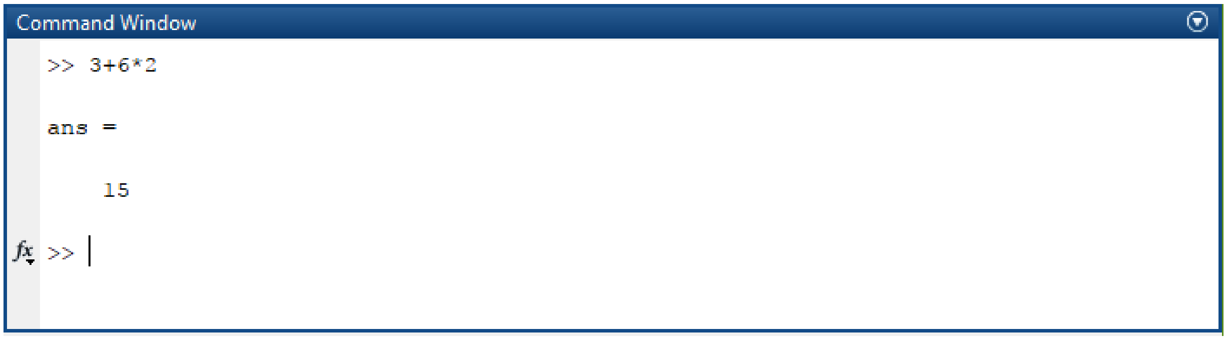
This button will open of the script editor and create a new untitled m-file. The file however is not "created" until it is first saved.

This button allows you to customize the layout of the MATLAB interface window. If you click on the button you can select from 4 different built in layouts, and select (or deselect) various subwindows for the MATLAB interface window. For example, if you removed the command history window and want to bring it back, you can select it from the Layout menu.

Clicking on this button will clear all the commands from the command window (which is sometimes useful if you have a lot of failed attempts at command that you would like to clear away). If you click on the down arrow to the right of this button, it will drop down a menu with two options: "Command Window" or "Command History" if you wanted to clear out the entire command history.

4. You will now attempt to use MATLAB to perform some basic calculations.

   (a) Click on the command window. To execute a command in Matlab you type the command into the command window and press Enter. Try typing `3 + 6 * 2`. MATLAB uses the `*` character to indicate multiplication. Notice MATLAB obeys the standard order of operations. You can type the command in with spaces around each mathematical operator (`+` and `*`) or not; it will not affect the result. Notice that when you execute the command, MATLAB returns the answer in the form `ans = 15` (though distributed over a few lines, see image below). MATLAB is storing the value 15 in a variable called `ans` (which is short for answer). If you look in the work space you will see the variable `ans` listed. The second column of the workspace shows the value of the variable, which is 15.



```
>> 3+6*2

ans =

    15

>> 
```

General arithmetic in MATLAB uses `+` for addition, `-` for substraction, `*` for multiplication, \ for division, and ^ for exponentiation. Note that spaces and adjacent parentheses do not indicate multiplication. Thus, $2(5-3)(0.1+7)$ must be entered as `2*(5-3)*(0.1+7)` where the `*` is used to indicate each multiplication.

(b) Negative numbers are indicated with a minus sign. Try typing `3 - 6 * 2` into the command window (`ans = -9`).

(c) Scientific notation requires the use of the character `e`. To use scientific notation for a number like $0.0001 = 1 \times 10^{-4}$, use `1e-4`. The value immediately after the `e`, in this case -4, will indicate the placement of the decimal. Be careful here not to place spaces between the `1`, `e`, or `-4`, as it will result in an error (try typing it into the command window and see). Try typing `5.2e-4 + 1e-5` into the command window (`ans = 5.3000e-04`). Note, the default precision of MATLAB is set to show 4 digits after the decimal (even in scientific notation).

(d) Complex numbers require the use of the character `i`. To represent the complex number $5.2 - 4.7i$, use `5.2-4.7i`. Note that a space between the `4.7` and the `i` will cause an error. Try typing the following two commands into the command window (make sure to press Enter after each line):

```
5.2-4.7i * 2.3-i
(5.2-4.7i) * (2.3-i)
```

Does each command result in the same value? Why or why not? Determine exactly what calculation is made in each command.

(e) MATLAB can only execute commands that "make sense". If your command does not make sense to MATLAB, an error message (usually in red) will be returned. Try typing `(3 + 6 * 2`. What does the error message say? What is wrong with the command that was entered?

(f) Recall that you can scroll through previous commands using the up arrow. Press the up arrow once to see the previous command `(3 + 6 * 2`. Now, use the left arrow key to navigate the cursor between the `6` and the `*`. Change the command to `(3 + 6) * 2`. Press Enter. Notice that MATLAB returned the result of this command in a form similar to the previous error-free command, and also uses the variable `ans`. In fact, if you look in the workspace, the value of `ans` has changed from 15 to 18. MATLAB has written over the old value of `ans` and replaced it with this new value, 18.

(g) If you want MATLAB to hold more than one variable (`ans`) at a time, you must specify variable names. Try typing the following two commands into the command window (make sure to press Enter after each line):

```
x = 3 + 6 * 2
y = (3 + 6) * 2
```

You may also choose to use the arrow keys, scroll through the previous commands, and edit them appropriately. Notice that two new variables have appeared in the workspace along with their corresponding values. If you want to change the values of one of the variables it is easy to do so. Try typing `x = 5` into the command window. Notice the change in the value of `x` in the workspace.

In general, variable names can use letters (upper and lower case), numbers, and the underscore character. You should note that variables in MATLAB are case sensitive, thus the variable `x` is not the same as the variable `X`. Characters and words that represent built-in constants and functions in MATLAB cannot/should not be used as variable names. For example, `pi` and `i` in MATLAB are use for the mathematical constants

π and $\sqrt{-1}$. You should strive to make variable names descriptive so that they are meaningful. However, variables can get a little unwieldy if they become too long.

(h) Once you have defined variables in Matlab you can use them in subsequent calculations. Try typing `z = y - x` into the command window. Does `z` evaluate the value you expect? Note the value of the variable `z` in the workspace.

(i) Basic scientific functions in Matlab are `sin(x)` for $\sin(x)$, `cos(x)` for $\cos(x)$, `tan(x)` for $\tan(x)$, `log(x)` for $\ln(x)$, `log10(x)` for $\log(x) = \log_{10}(x)$, and `exp(x)` for e^x. Note the the input for the sine, cosine, and tangent functions are in radians. There are many other built-in functions in Matlab which will be introduced as we need them.

Create a Script M-file

You have now successfully executed several commands in the command window. However, what if you wanted to execute the same set of commands (more than one line of commands) over and over again. It would become tedious to do this in the command window. For this, we turn to the creation of m-files. In lab we will create script m-files, but in later lab assignments we will learn how to create function m-files.

1. Open a new script file (see Tour of Matlab section).
2. We are going to write a simple script which calculates the volume of a cylinder given its height and diameter. Recall, that the volume of a cylinder with radius r and height h is $V = \pi r^2 h$. In the m-file editor window, type the following exactly:

L01Ex01.m

```matlab
% Lab 1 Example 1 by Prof Bodine
% Find the volume of a cylinder with diameter d and height h

% Enter the diameter (d) and height (h)
d = 3;
h = 7;

% Calculate the volume
V = pi * (d/2)^2 * h;

% Display the results
disp('The volume of the cylinder is ')
disp(V)
```

Once you have created the file, save it as `L01Ex01.m` (which will be our naming convention for Lab 1, Exercise 1). Make sure you take note of where you are saving it. If you save the file in a location other that the current folder, make sure you change the current folder to the location where the `L01Ex01.m` file is saved.

Now, let us take a moment to understand everything contained within this script.

(a) Anything after a `%` is a comment. This means it is NOT a command executed by the script, but a helpful note to yourself and others. These notes are helpful to yourself if you are attempting to use code you wrote quite some time ago, and helpful to others who may be looking at your code for the first time and trying to understand what it does. It may seem silly to have comments in this simple script (there are 5 lines of comments and 5 lines of actual executable code), but it will become increasingly more important as your code becomes lengthier.

The first two comments (lines 1-2) provide a label for the script, the author's name, and the purpose of the script. It is a good coding habit to always use the first few lines of a script for this purpose. The comments on lines 4, 8, and 11 describe what the immediate subsequent lines of code do.

(b) This script uses MATLAB's built-in function `disp`. The disp function will display strings (text placed in single quotes) and variables. The `disp` function only takes one argument which is why we use the `disp` function first to display a string of text (line 12), and then again to display the result of our calculation (line 13).

(c) Note that some of the lines of code end with a semicolon. In MATLAB a semicolon placed at the end of a line of code will suppress the output of that line of code. We will experiment with this shortly.

(d) Take a moment to review the code. **Do you understand what each line of code is doing?** If not, now is the time to stop and ask your instructor or another student (preferably one who has some experience with writing code).

3. Now it is time to try executing the script file. MATLAB provides a couple of ways to accomplish this: you can press F5 on your keyboard, you can click on the run button in the editor, or you can type the name of the file (without the `.m`) into the command window.

   Prof B's Personal Preference: When I am running code in MATLAB *I prefer to have the script editor window as a subwindow in the* MATLAB *interface window. To do this click on the circled down arrow in the upper right corner of the script editor window. Select "Dock Editor". If you do not like this set up you can always undo it by going through the same steps but selecting to undock.*

   Run the `L01Ex01.m` script file with 3.4 for the diameter and 4.6 for the height. Do you get 41.7643 for the volume? What appears in the workspace? Try running the script again and using different values for the diameter and height. Verify that the script is producing the correct volume each time.

4. Go back to the script editor and delete the semicolon at the end of line 6. Now run the script file again. What happens? Do you feel it is better to replace the semicolon or keep it removed?

Generate a Graph

It is often helpful to be able to display data graphically. For example, consider the cell density data from Motivating Example 1 in the reference readings (see Table 2). The table shows the density of an *E. coli* cell culture measured every 30 minutes for 3 hours. Let us plot the data shown in this table.

1. The built-in function in MATLAB for plotting is `plot(x,y)` where `x` and `y` are arrays of values which give the horizontal and vertical coordinates of points on the graph. Note, an array is simply a list of numbers. To create

Table 2: Optical Density of *E. coli* cell culture.

Time (hours)	0.0	0.5	1.0	1.5	2.0	2.5	3.0
Cell Density (OD_{600})	0.055	0.120	0.231	0.360	0.516	0.821	1.300

an array in MATLAB you can enter the numbers individually between a set of square brackets with spaces between each number. For example,

`t = [0.0 0.5 1.0 1.5 2.0 2.5 3.0],`

which is the set of time values from Table 2. If you have a lot of data to enter and it takes more than one line, place three dots (`...`) at the end of each line that is continued onto the next. However, if you have evenly spaced data (like the time data in Table 2) we can use MATLAB's colon notation. For example, `[0:10]` produces an array of integers from 0 to 10, i.e., 0, 1, 2, 3, . . . , 9, 10. However, `[0:2:10]`, produces numbers from 0 to 10 in increments of 2, that is, 0, 2, 4, 6, 8, 10. Thus, we could have created the time array by using

`t = [0:0.5:3].`

2. Open a new script file. In the m-file editor window, type the following exactly:

L01Ex02.m

```
% Lab 1 Example 2 by Prof Bodine
% Produce a Plot of the Cell Density Data

% Data for time (t) and cell density (d)
t = [0:0.5:3];
d = [0.055 0.120 0.231 0.360 0.516 0.821 1.300];

% Generate plot
plot(t,d,'ro')
xlabel('Time (hours)'); ylabel('Cell density (OD_{600})');
```

Save this script as `L01Ex02.m` in the same folder as `L01Ex01.m`.

Now, let us take a moment to understand everything contained within this script.

(a) Take a moment to read the comments. Do you understand what each section of the script is doing?

(b) Note the plot command now has an additional third input (after the `t` and `d`). This third input indicates the color (red) and marker shape (circles) of each of the points on the graph. In the command window type `help plot` (and scroll up) to see a list of the possible colors, marker shapes, and line styles that can be used with the plot command.

   (c) This script also uses MATLAB’s built-in functions `xlabel`, and `ylabel`. These commands place text along the horizontal and vertical axes, respectively.

   (d) Take another moment to review the code. **Do you understand what each line of code is doing?** If not, now is the time to stop and ask.

3. Run the script file. A window containing the graph should appear and should look like Figure 4. What does the workspace in the MATLAB interface window look like? Pay specific attention the value column.

4. Change line 9 in the code to `plot(t,d,’ro-’)` and run the script. What happens? How does the plot change? Note, the plot window may not automatically “pop up” again. You may need to relocate the window to see the changes that have been made.

5. Change line 9 in the code to `plot(t,d,’r0)` and run the script. You should get an error message. Does the error message make sense? Do you see what is now wrong with line 9 of the code?

6. Change line 9 in the code to `plot(t,m,’r0’)` and run the script. You should get an error message. Does the error message make sense? Do you see what is now wrong with line 9 of the code?

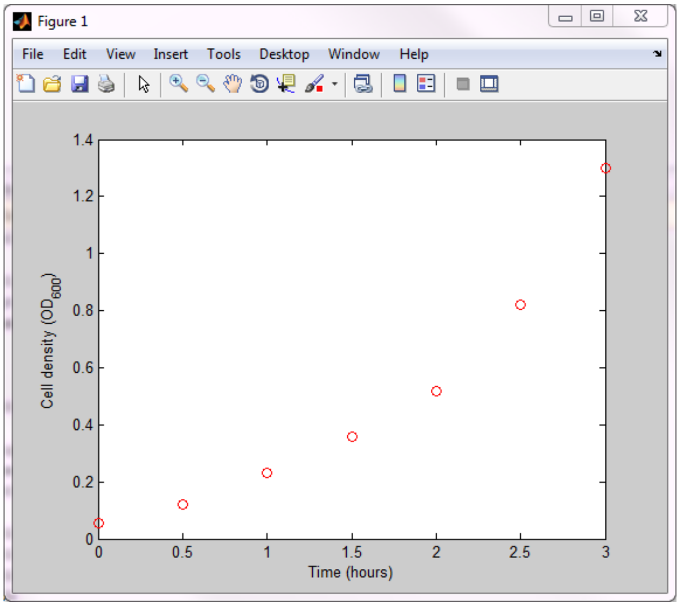


Figure 4: Plot produced from `L01Ex02.m` script.

Write a For-Loop

In Motivating Example 1 from the reference readings we took the average of ratios of cell densities at consecutive time steps to determine the value of a parameter c in the discrete exponential growth model $x_{n+1} = cx_n$. Calculating the values of these ratios was not so bad since we only had seven data points. However, this would become tedious for larger data sets. MATLAB is an excellent tool for performing the same calculation many times. When you want to repeat the same type of calculation again and again, you use computer coding construct called a ***loop***. There are two types of loops: *for-loops* and *while-loops*. ***For-loops*** execute a block of code a specified number of times, where as a ***while-loop*** executes a block of code until a (logical) condition is met which is triggered by an event in the loop.

Since we know how many ratios we need to calculate (the number of data points minus 1), we will use a for-loop. In later lab assignments we will learn how to construct while-loops.

When to use a for-loop · A for-loop is used to repeat a calculation a specified number of times. Suppose I asked you to pick a number. Then I asked you to multiply that number by 2 and write down the product. Then I asked you to multiply the resulting number by 2 and write down the product. And, then I asked you to repeat that operation 100 times. Essentially, I would be asking you to "loop" through the same procedure over and over again, and you would be iterating through a for-loop (by hand). The mathematical analogy of this is when you generate the terms of a sequence $x_{n+1} = 2x_n$ by starting with x_0 and iteratively generating the first 100 terms.

Writing a for-loop · Often when we implement a for-loop we will store values generated within each iteration of the for-loop in an array. Suppose we have an array `x` containing 5 values:

```
x = [4.3 2.1 5.6 9.3 10.2].
```

To refer to the first element of the array we use `x(1)`. To refer to the second element of the array we use `x(2)`, and so on. In general, to refer to the i^{th} element of the array we would use `x(i)`. Now, if we wanted to implement a for-loop to generate the terms in the geometric sequence defined by $x_{n+1} = 2x_n$ up to term x_{100}, assuming $x_0 = 7$, then we would use the following snippet of code:

```
x(1) = 7;
for n = 1:100
        x(n+1) = 2*x(n);
end
```

These four lines of code create a variable `x`, which will be an array containing all the terms in the sequence up to x_{100} (this will be a total of 101 terms since the first term is x_0). The first line of the code defines the first term in the sequence, $x_0 = 7$. Notice that even though we start numbering the indices of the geometric sequence with $n = 0$, in Matlab we must start the array indices at 1, and thus we have `x(1) = 7`. The next three lines are the for-loop. A for-loop is always started with `for` and ended with `end`. Right after the `for` (on the same line), we must tell Matlab the values over which the loop will iterate. In this case, we are iterating over the values in the array `1:100` which produces an array of every integer from 1 to 100 (try typing `1:100` in to the Command Window). This array represents the indices of the sequence (shifted by 1). Any code between the line starting with `for` and the line with `end` is the code that is executed one for each iteration of the for-loop.

Executing a for-loop · In our example, when the for-loop code is executed, the value of `n` is set to first value in array `1:100` (i.e., `n=1`), and then the calculation on the middle line is executed. Matlab then "loops" back and sets `n` to the next value in the array `1:100` (i.e., `n=2`), and the executes the calculation on the middle line. This process is repeated for each element in the array `1:100`, that is, this process is repeated 100 times. Each time Matlab iterates through the loop, the next term in the sequence is generated. When `n=1`, the `x(2)` term is generated. When `n=2`, the `x(3)` term is generated. When `n=3`, the `x(4)` term is generated. And so on, until lastly, when `n=100`, the `x(101)` term is generated. Note that during the first iteration through the loop, the calculation of the `x(2)` term requires Matlab to "already know" the value of `x(1)`. This is why we define the value of `x(1)` prior to the beginning of the for-loop. After the for-loop is completed (i.e., the loop has been iterated through the appropriate number of times), Matlab will "move on" and execute any code written below the for-loop.

Now, let us use a for-loop to determine the ratios of consecutive data points using the cell density data in order to determine an appropriate parameter for the discrete exponential growth model.

1. To access the i^{th} element of an array `x` in MATLAB we use `x(i)`. Your workspace should still contain the array d, if not, run `L01Ex02.m` again. In the command window type `d(3)` and press Enter. This should return the 3^{rd} element of the array `d`. Now, try `d(5)` and press Enter. This should return the 5^{th} element of the array `d`.
2. In the command window type `d(10)` and press Enter. This should return an error message. What does the error message say? Why was there an error message?
3. In the command window type `d(0)` and press Enter. This will return a different error message. Note that in MATLAB the indexes for arrays start at 1, thus all index values (the value between the parentheses) should be positive integers. Does this error message make sense?
4. Open a new script file. In the m-file editor window, type the following exactly:

L01Ex03.m

```
1   % Lab 1 Example 3 by Prof Bodine
2   % Calculate the proportion by which the cell density is increasing each
3   %   time step
4
5   % Enter the data where d is cell density
6   d = [0.055 0.120 0.231 0.360 0.516 0.821 1.300];
7
8   % Caculate the ratio of consecutive data points
9   N = length(d) - 1;
10  for i = 1:N
11     ratio(i) = d(i+1)/d(i);
12  end
13
14  % Caculate the mean ratio
15  a = mean(ratio);
16
17  % Display the % population increase
18  r = (a-1) * 100;
19  fprintf('Every 30 minutes the cell density increases by %3.1f%%\n', r)
```

Once you have created the file, save it as `L01Ex03.m` in the same folder as the other example m-files.

Now, let us take a moment to understand everything contained within this script.

(a) You should be able to understand lines 1 - 6 based on the previous examples.

(b) In line 9 we create a new variable `N` and give it the value of the length of the array `d` minus 1. Note, `length` is a built-in function in Matlab that takes an array as the input.

(c) The for-loop starts on line 10. We know it is a for-loop because it uses the reserved MATLAB word `for`. Note, you cannot use `for` as a variable name. After the `for` we define the index variable as `i` and set it to take on values 1 up to `N`, increasing by increments of 1 (notice this is the same MATLAB colon notation used for arrays). This means that for the first pass through the loop, the variable `i` will have a value of 1; the second pass through the loop, the variable `i` will have a value of 2; . . . ; the N^{th} pass through the loop, the variable `i` will have a value of `N`, which will be the last pass through the loop. After the last pass through the loop, MATLAB will move down to start executing the next line of code, in this case line 15.

(d) Let us examine line 11 more closely. During each pass through the loop, we calculate the ratio `d(i+1)/d(i)` where `i` is the loop index and `d(i)` indicates the i^{th} element of the array `d`. Each calculated ratio is then stored in the i^{th} position of a new array called `ratio`. The array `ratio` is initially created during the first pass through the loop, and a new value is appended on to the end of the array during each pass through the loop.

(e) In line 15 we use the built-in MATLAB function `mean` to calculate the mean or average of the values in the array `ratio`.

(f) You should understand line 18. The command in line 19 is similar to display, but allows a bit more flexibility in how the information is displayed. The function `fprintf` allows you to create what is called a ***format string*** which describes exactly how numbers and strings (a "string" of characters; think words and sentences) will be displayed. With `fprintf` you can display strings and variables on the same line, which you cannot do with the `disp` function. Within the string (between the single quotes) you can have text, and if you want to include the value of a variable you use the following format: %**w**.**d**x, where **w** is the total number of digits you want to display, **d** is the number of digits you want after the decimal, and x specifies what type of format your number is in

- `d` for integers,
- `f` for real numbers,
- `e` for scientific notation, and
- `g` to switch between `f` and `e` as needed.

If the total number of digits **w** can vary, then you may omit the **w**. In line 19 we use `%3.1f` which indicates that we have a real number, and we want to display 3 digits with only 1 digit after the decimal. However, we could have also used `%.1f` since the total number of digits could be larger or smaller than 3, but we always want to have the resulting value reported to the tenths place. The `%%` is how we allow the percent sign character to appear since this character normally indicates a comment in the code. The `\n` forces MATLAB to go to the next line. If we do not have the `\n`, the next thing we print or try to type into the command window will be on the same line.

Notice that the `fprintf` function in line 19 has two inputs: the string (between the single quotes) and the variable `r`. Thus, the number filled into the string where the `%3.1f` occurs is given by the value of the variable `r`. If the string contains multiple formatted values then the all of the appropriate variables are listed after the string separated by commas. For example, we could add a line between lines 11 and 12 which prints out the value of each ratio:

```
fprintf('Ratio %d is %4.3f.\n', i, ratio(i))
```

   (g) Take another moment to review the code. **Do you understand what each line of code is doing?** If not, now is the time to stop and ask.

5. Run the script file. What is printed out in the command window? What appears in the workspace?
6. Change line 19 to have `%4.2f` instead of `%3.1f` and run the script file. How does this change what is printed out in the command window?
7. Change line 9 to `N = length(d);` and run the script file. You should get an error message. What does the error message say? Does it make sense? Correct line 9.
8. Add the line `fprintf('Ratio %d is %4.3f.\n', i, ratio(i))` between the currently lines 11 and 12. How many times is the `fprintf` function executed when the script is run once?

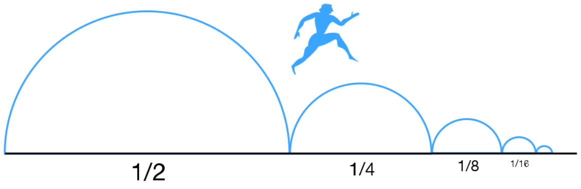
This completes the Matlab Skills section. You will review these skills in the group discussion, and put these skills to use in the Lab Assignment.

Lab Group Discussion

Week 1:

1. When should we use a semicolon in Matlab code?

2. Zeno's Dichotomy Paradox: *That which is in locomotion must arrive at the half-way stage before it arrives at the goal.* Put another way, suppose you want to move from point $x = 0$ to $x = 1$ on the number line. First, you move half the distance to $x = 1/2$. Next, you move half the remaining distance to $x = 3/4$. Then, you move half the remaining distance to $x = 7/8$. If you keep repeating this process, will you ever reach exactly $x = 1$? Theoretically, no, but physically we are able to get from point A to point B in finite time; therein lies the paradox.



The `L01Ex04.m` script calculates how far you have gone after n steps in the dichotomy example described above. Note, after $n \geq 1$ steps, $x = n/2^n$.

L01Ex04.m

```
1  % Lab 1 Example 4 by Prof Bodine
2  % Example for discussion: Zeno's paradox
3
4  n = 10;
5  for i = 1:n
6    fprintf('At step %d, x = %d/%d = %.5f\n',i,i,2^i,i/2^i)
7  end
```

   (a) How many inputs does the `fprintf` command on line 6 have?

   (b) Why is `d` used for formating three of the inputs and `f` use of another?

   (c) How many iterations does the for-loop cycle through?

   (d) `L01Ex04.m` generates two variables which are stored in the MATLAB Workspace. What are those two variable and what are their values at the end of each iteration of the for-loop?

Week 2:

1. In `L01Ex03.m` on line 6 you create an array.

   (a) What is the name of the array?
   (b) How many elements are in the array?
   (c) Does the size of this array ever change?
   (d) What is the third element of the array?
   (e) What MATLAB code would report back the third element of the array?
   (f) What MATLAB code would report back the arithmetic mean of the array?
   (g) What MATLAB code would report back the geometric mean of the array?

2. In `L01Ex03.m`, the for-loop on lines 10–12 also creates an array.
   (a) What is the name of the array?
   (b) After the for-loop has gone through one iteration, how many elements are in the array?
   (c) After the for-loop has gone through all of its iterations, how many elements are in the array?
   (d) After the for-loop has gone through all of its iterations, what is the third element of the array?
   (e) What MATLAB code would report back the third element of the array?
   (f) What MATLAB code would report back the arithmetic mean of the array?
   (g) What MATLAB code would report back the geometric mean of the array?

Checkpoint Exercise

1. Create a script m-file called `PLab01_YourLastName.m` where `YourLastName` is replaced with your last name. Your name and a description of the script's main purpose should be in the first few lines of the script as comment lines. Your script should be well commented and perform the following tasks:
   (a) Calculate the values of $x_1, x_2, \ldots, x_6$ using the model $x_{n+1} = 1.7x_n$ with $x_0 = 0.055$. Store all the values (including x_0) in an array called `x`. Recall that 0 cannot be used as an array index. For full credit, you must use a for-loop to complete this task.
   (b) Generate one plot with both the original data from Table 2 and the data generated from your model on the same graph. To accomplish this use the command `hold on;` after your first plot command. This will make the points from your second plot command appear on the same graph. After the second plot command use `hold off;`. Make sure that the axes of your graph are appropriately labeled.
   (c) Use a for-loop to calculate the difference between the actual data and your model (use $\text{data}_i - x_i$). Use an `fprintf` statement within the loop to print out the difference accurate to 0.001.

2. Run your script. Save the resulting graph as a `PLab01_YourLastName.jpg`. To do this, in the figure window select "Save as..." from the File menu, select the appropriate folder, set the filename, select "JPEG image (*.jpg) for "Save as type:", and press Save.

3. CHECKPOINT: Verify that your image file `PLab01_YourLastName.jpg` is correct with your lab instructor.

Lab Assignment

1. Create a script m-file called `Lab01_YourLastName.m`. Your name and a description of the script's main purpose should be in the first few lines of the script as comment lines. Your script should be well commented and perform the following tasks:

   (a) Enter the data for the bluefish harvest yields into two arrays: `year` containing the year data and `fish` containing the pounds of bluefish harvest for each year.

   (b) Use a for-loop to calculate the ratios of bluefish harvest yields (in lbs) at consecutive time steps. Use the `fprintf` function to print out the value of each ratio. Then calculate the arithmetic mean of the ratios (store this as the variable `a`) and the geometric mean of the ratios (store this as the variable `b`). You will need to use the `mean` and `geomean` functions in Matlab. Use `fprintf` commands to print out the values of the means to the Command Window.

   CHECKPOINT: Verify that your code is producing the correct arithmetic and geometric means with your lab instructor.

   (c) Use a for-loop to generate the values for the sequence $x_n = a^n x_0$ where $x_0 = 15,000$, and the values of the sequence $y_n = b^n y_0$ where $y_0 = 15,000$. Generate the values of sequence for $n = 1, 2, \ldots, 10$, and store all the values (including x_0 and y_0) in arrays called `x` and `y`, respectively. Recall that 0 cannot be used as an array index.

   (d) Generate one plot with both the original bluefish harvest yield data and the data generated from your two models on the same graph. Make sure the axes of your graph are appropriately labeled. Use the command `legend('data','x','y')` to create a legend.

   (e) Calculate the error the between each model and the data. Save these calculations in the variables `errorx` and `errory`. Use `fprintf` commands to print out the values of `errorx` and `errory`.

   In MATLAB , you can make these error calculations without a for-loop by taking advantage of the fact that MATLAB was designed to work easily with arrays and matrices.

   - `fish-x` will produce an array of the differences between the data and the x model sequence values.
   - `(fish-x).^2` will produce an array of the squares of the difference between the data and the x model sequence values. Make sure to include the period before the carrot, this tells MATLAB that you want square each value in the array. Without the period, MATLAB will multiply the array by itself, which is an entirely different operation (defined in classes like Linear Algebra and Multivariable Calculus).
   - Use the `sum` and `sqrt` commands to take the sum of an array and the square root of a number, respectively

2. Before running your script, create a diary file with the name `Lab01_YourLastName.out` by typing

```
>> diary Lab01_YourLastName.txt
```

in the Command Window and pressing Enter (note that you don't type the `>>`, those will already appear in the Command Window). Now, run your script, then close the diary by typing

```
>> diary off
```

in the Command Window and pressing Enter. Open the diary file to make sure that your output is visible in the file.

3. Save a copy of the graph generate by your script file as `Lab01_YourLastName.jpg`.

   Print & Bring to Class:

   - Lab Report (use the template provided on Moodle; your graph will be included in this file)
   - script file (`Lab01_YourLastName.m`)
   - output file (`Lab01_YourLastName.txt`)

   Staple these items together (in the order listed above). You are encouraged to print double-sided. *Make sure your name is on the lab report.*