Lab: Introduction to Programming

1 M- files

This section was created by Professor Gilchrist (Department of Biology at the College of William and Mary).

A program in MATLAB is created with a text editor and saved as a text file. (MATLAB includes an excellent text editor with many useful features, but one could use an external editor if one wished.) It must be saved as a text-only file. It must have a name that ends with .m such as "filename.m", and as such, may be called an "m-file."

The most simple way to work with m-files is to keep these files in the "work" folder under the MATLAB directory. You may also create a directory somewhere in your home directory where you keep your MATLAB work. When you open MATLAB, you need to navigate to that directory to access m-files that you have already created. You can do this by selecting that directory as your Current Directory in the tool bar of the main MATLAB window.

When you create programs with a text editor, save it as a text file with the .m suffix. To run the program start up MATLAB, and, from its command line, type the name of the program you have previously saved (not including the .m suffix.) Note that MATLAB does NOT like spaces in file names!!!

2 Programming

The goal of this lesson is to introduce you to programming logic and language. To start programming use the built in editor in MATLAB. To do this go through the **Desktop** menu then choose **Editor**. Now open a new file using the **File** menu in the editor and choose the **M-File** option under **New**. We will begin to start our first MATLAB program.

2.1 Getting Started

The very first item on the top of the code should be your name! Since your name is not part of the code the computer needs to see we will write it as a comment in the program. To do this the comment must begin with a percentage sign (%). Anything you type behind the percent sign will not be read as part of the code. Good programmers comment their code, so they can remind themself (and any else who may read their code such as a professor) what they did and why they did that. In the editor, beginning at the top left, type:

```
% Your Name
% Today's Date
% Introduction to MATLAB programming
```

The first few actual code lines of each program should start with the following items:

- the function name (which is the same as the filename)
- clearing the memory
- clearing the work space

It is VERY IMPORTANT that the filename used in the function command is exactly the same as the file name used to save the file. The only difference will be that the saved file will have a ".m" on the end of it to denote it is an m-file. We will both save and label todays's program as "example1". In order to save this file use the editor's built in file menu. Use the **Save As** option and type: example1.m for the file name. Now underneath your heading press the **return** or **enter** key a few times and then type:

```
function[] = example1()
clear all
clc
```

The first line denotes the beginning of a program/function. (It is possible to have functions nested inside of one another.) The **clear all** command erases the memory and **clc** clears your workspace.

2.2 Defining Parameters and Variables

There are many techniques to defining parameters and/or initial conditions, the most simple being to list the parameters and their assigned values just as you would working with the MATLAB prompt. Remember, if you do not want to see the output use semicolons. A good programmer will label the sections of their code as well as their parameters. (Students who want help with their code will also use labels!) For this program define the following parameters:

```
% Parameters
a = 3;
n = 50;
intX = 100;
```

The next method is used if you are sharing data, parameters and/or variables between two programs. In order to do this the parameters and variables being used in both codes must be labeled as **global**. This is similar to having a home phone with only local access and having long distance. We will need to use this option when we work with simulations of continuous systems of nonlinear equations. In order to define this you would use:

```
global a;
global n;
```

```
global intX;


% Parameters
a = 3;
n = 50;
intX = 100;
```

Although it is not necessary to assign variables, wise programmers do, especially those who are learning how to program. In MATLAB your variables will be recorded as vectors, providing a natural check to make sure you are recording the correct number of steps or iterations. One straight forward way to define variables is to use the **zeros** command to make a "dummy" vector which the program will later fill in for you.

```
% Variables
pred = zeros(1,50);
prey = zeros(1,50);
year = zeros(1,50);
```

These are just a few basic methods for defining terms.

2.3 Running the Program

To run this program (which will not show you anything unless you remove some of the semicolons), type the file name without the ".m" suffix.

```
>> example1
```

Next time you wish to run your program use the up arrow at the MATLAB prompt. This should recall the last item you typed and then hit **return** or **enter**. In fact, the up arrow remembers many of your previous commands! Now let us work with some programming logic.

2.4 Loops

In order to simulate equations we will often need to repeat steps time after time. This is very tedious by hand, but is fairly straightforward to do with loops. We will learn about "for" loops below (note that there are also "while" loops. See **help** for more details.)

Typically in programming we use the letters i and j as index numbers which allow us to step through the programs. However, in MATLAB the letters i and j have already been assigned as imaginary numbers. You may overwrite this built in MATLAB functions, but then you will be unable to use i and j for imaginary notation. I suggest using other letters to denote indexes, such as **k**, **ii** or even **jj**.

Often times we want a computer to do a set of instructions over and over again. This requires a programming structure called a "for" loop. You define a counter, say k, which goes from some lower

limit to an upper limit, usually in steps of one. For each count of the counter, you do a bunch of commands. When you hit the upper limit on the counter, the program stops. Look at this example where we begin at the value 2 and end at 5. In your program type:

```
% For Loop practice
for k = 2:1:5
    k ∧ 2
end
```

The first line is read by the computer as "repeat the following commands starting with k=2, incrementing k by one each time and continue until k= 5. Now, let's do that again and put the results into a matrix (solution vector) and define"n" to be the number of iterations we wish to take.

```
% Second Loop practice
n = 5                 % Number of desired itterations
x = zeros(1,n);       % Solution vector
for k = 1:1:n         % Loop from ii=1, steps of 1, to ii=n which in this case is 5
    x(k) = k ∧ 2;
end
x                     % To see the resulting vector
```

Occasionally, we will want to use our previous values to find the next set of values. This will also cause some subtleties in the indexing. Since matrix indexing begins with the number 1 and sometimes we wish to begin our simulation or program from time = 0, we have to amend our indexing. The first fix is with vector lengths (in the example below this can be seen through "n" and "nn"). Here we want the program to run for n numbers of years, but we also want to include the initial condition for the zeroth year (time=0). This means the solution vectors will have one more entry than the number of iterations. The second indexing problem is due to the fact that we need the previous result to calculate the next result. For example, using last year's population to predict this year's population. In this example, we let "k" be the previous solution (or time) while "kk" is the next solution (or time).

In this example, let us work with a population of bacteria that doubles every hour. We will begin with an initial population of 100 bacteria and want to know how many bacteria there are every hour for the next 12 hours. (Okay, we could do this in our heads, but lets use MATLAB instead.) This means we will actually have 13 values to find including the initial bacteria colony at time zero. We can look at the solution vector ("bacteria") in the workspace or even graph it!

```
% Parameters
n = 12;
nn = n+1;
initial = 100;

% Variables
bacteria = zeros(1,nn);
hours = zeros(1,nn);
```

```
% Setting the initial conditions
bacteria(1) = initial;

% Equation/Simulation
for k = 1:1:n
    kk = k+1;
    bacteria(kk) = bacteria(k)*2;
    hours(kk) = k;
end


% To see the solution in the workspace
hours
bacteria


% To graph the solution
figure(1)
clf
hold on
set(gca,'fontsize',20)
plot(hours, bacteria, 'b','linewidth',3)
title('Bacteria Population')
xlabel('Hour')
ylabel('Number of Bacteria')
hold off
```

We will be using this type of loop throughout the course.

3 Conditional Statements: If-Else

These statements can be used to alter equations, parameters or variables dependent on specified conditions. Considering the absolute value function, if the number of interest is positive or zero it remains unchanged. However, the number of interest is negative then this number is multiplied by -1 in order to make it positive. This is an example of where the input determines the manipulation used on it.

In order to test whether or not our number of interest is positive, zero or negative, we need to use logic statements that will either be true or false. If this condition is true, then the statements that follow "if" will be evaluated. If this condition is false, then this command will evaluate the "else" portion. There are many different conditions that can be used. Here is a list of the typical conditions:

If the statement is true then MATLAB returns the value"1" and if the statement is false MATLAB returns the value "0". Occasionally this fact comes in handy! For now, we should check to make sure

Condition	True when:
$x < y$	True if x is less than y
$x > y$	True if x is greater than y
$x <= y$	True if x is less than or equal to y
$x >= y$	True if x is greater than or equal to y
$x == y$	True is x exactly equals y
$x \sim= y$	True if x does not equal y

this is correct.

```
>> 1<2
>> 1>2
>> 2<=2
```

Let us assume you may want a command only to be performed in certain circumstances. For example, dividing a number by zero is not wise in mathematics. Here is an if-else statement which will do the division if x is not zero and will return the value **nan** (not a number) if x is zero.

```
x = 2
if x ~= 0
  y = 1/x;
else
  y = nan;
end
```

Now try it again by changing `x = 2` to `x = 0` and see what happens.

```
x = 0
if x ~= 0
  y = 1/x;
else
  y = nan;
end
```

The "if-else" statements can be linked together if you have more alternatives by using the **elseif** command. For this next example any negative x value will be turned into a positive, any zero x values will be added to 3 and any positive x values will be used in a division. Mathematically this is know as a piecewise function and written as:

$$y = \begin{cases} -x, & \text{if } x < 0; \\ 3 + x & \text{if } x = 0; \\ \frac{1}{x} & \text{otherwise.} \end{cases}$$

In MATLAB this function would be typed in as the following:

```
if x<0
  y = -x;
elseif x==0
  y = 3+x;
else
  y = 1/x;
end
```

Let us use this function to evaluate the x at -4, -1, 0, 1, and 2 and store them in a solution vector labeled "y". We will need to use both a "for" loop and an "if-else" statement. You may want to cut and paste in your program since we have previously used all of these commands.

```
x = [-4 -1 0 1 2];
y = zeros(1,5);
for k=1:1:5
   newx = x(k);
   if newx<0
     y(k) = -newx;
   elseif newx ==0
    y(k) = 3+newx;
   else
    y(k) = 1/newx;
   end % end if statement
end % end for loop
```

Remember:

- Once you have created a program (m-file) you can run the program by typing in the filename without the ".m" at the MATLAB prompt. You must save your file each time you change parameter values, before you run the programs.
- File names and function names must match:
  - file name = assignment1.m
  - function name = assignment1()
- Use the "help" command to find good names for your variables!

4 Practice Problems

1. For the following values of x:

-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5

write a program to perform and store the results of the equation:

$$y = 2x^2 - 3x + 1$$

then plot the resulting function

2. For the same values of x in the previous problem, write a program to perform, store and plot the following function.

$$y = \begin{cases} x^2, & \text{if } x < 0; \\ 3 & \text{if } x = 0; \\ \sqrt{x} & \text{otherwise.} \end{cases}$$