ROSE-HULMAN INSTITUTE OF TECHNOLOGY
Department of Electrical and Computer Engineering

EC 300 Signals and Systems Spring 2006

MATLAB Scripts and Functions

Lab 2

Mark A. Yoder and Bruce A. Ferguson

Objectives

The purpose of this lab is to learn to write program files in MATLAB. While working through this lab you should learn many of the commands we will be using throughout the quarter. In addition, you will learn to create MATLAB programs, which we will use, reuse, and modify throughout the term. Use the `help` command often and early. Specifically, you should achieve the following:

- Learn the purpose of MATLAB scripts and functions (aka “m-files”)
- Learn to create a script file for performing repeated operations
- Learn to create functions for custom operations
- Learn to properly graph data from various sources

Pre-Lab

1. Create a working directory for you MATLAB work in this class. You should create a MATLAB directory with folders for ECE 300 Class and ECE 300 Lab.
2. Read the following notes, and write a good definition of the “`path`” function/variable used in MATLAB. Type `helpwin path` for details about the current path setting.
3. Indicate how directories are added to the path. Add the directories you have created to the `path` variable. Enter this information into your lab notebook, and turn a photocopy in as instructed.

Notes on Using MATLAB

MATLAB can be used in two different ways – through the command line interface and using MATLAB scripts. This second lab focuses on the use of MATLAB programs.

The second method of interfacing is by writing MATLAB programs called scripts or “m-files”. These are programs written in MATLAB’s command language, the same as is used in the command window. In the last lab, you typed a sequence of commands in a specific order to achieve some more complex result (such as plotting sinusoids). Each command had a specific purpose, and the order was important. However, one big problem with the method used in that lab was that you could not save the sequence of commands that you used. MATLAB programs allow you to save a sequence of commands for reuse and modification.

The usual structure of a MATLAB program is similar to that of other programming languages: a main program is written (*filename*.m), and this program can call functions saved in other files (*function_name*.m). The main program is where you organize your work into a coherent flow. Functions are used to program repeated command sequences, so that your main program can be neater and smaller. Every command you type in the command line interface window is actually a MATLAB function. Both the main program and functions are called m-files, but they have slightly different structures.

It is suggested that you use the MATLAB script editor to create and modify your m-files. The editor can be accessed from inside MATLAB ("File → New → M-File"). Be sure to save the file before you run it. If you do not, MATLAB will use the previous version of the file which it has stored in memory. Simply type the filename (without the ".m") from the command line to run the program. MATLAB references the script (program) only by its filename, so choose a name that has meaning. (Seventeen files named "testx" make for a confusing lab exercise!)

The files you create and save are stored in the directory shown in the "`current directory`" window just above the command window. Be sure to create a directory for your ECE 300 labwork, and navigate to that directory using the browse button (ellipses - "…") to the right of the `current directory` window.

MATLAB can access any file stored in its *path* variable. The path is a list of directories that MATLAB has been told to look into to find commands and files. When you type a command `burgerflip` in the command window, MATLAB looks through its path to find a file named "`burgerflip.m`", and then executes it.

Finally, when you save a function, MATLAB will know the function only by its filename. Even if your function definition in the first line of the m-file has something different listed as the command name, the filename in which you store the function is MATLAB's official reference. So when you call the function from either the command window or an m-file, you must use the name of the file the function is stored in. It is obviously good practice to make both the file name and the function definition line use the same name.

Procedure: There are three exercises for today's lab

1. Script file basics

(a) Use the MATLAB editor ("File → New → M-File") to create a program file called `sintest.m` containing the following lines:

```
t = -2:0.05:3;
x = sin(2*pi*0.789*t); % plot a sinusoid
plot(t,x), grid on
title('Test Plot of Sinusoid')
xlabel('Time (s)');
```

Save the file. (MATLAB cannot access your programs or functions until they are saved from the editor.)

Creating a program file in this fashion allows for easy storage of a number of commands as a MATLAB program. The program file is also called an M-File or a script. The program will be stored in the current directory indicated in the MATLAB window.

Run your function from MATLAB by typing its name (`sintest`) at the MATLAB prompt (note that `sintest` must reside in MATLAB's search path). **Verify** that the program produces the expected results.

(b) You have now created your first MATLAB program! Type `type sintest` at the command prompt to see the file you have created.

(c) Edit `sintest.m` to add a line containing the "`hold on`" function (type help hold for more information), followed by another plot command to add a plot of `0.5*cos(2*pi*0.789*t)` to the plot created above. Add a final line to your program containing "`hold off`". Note that you must save sintest.m in order for MATLAB to access the updated program.

Instructor Verification (see last page)

2. MATLAB Functions

MATLAB commands typed at the command line interface are simply functions. In this exercise we will learn to create various types of functions.

(a) *Create a simple function with no arguments.* Type the following lines and save them in a file called `greeting.m`. Save the file, and test it at the command line. This function is just a small program similar to your main program, with one major difference.

```
function greeting
disp ('Hello world!')
test_variable = 3.14159
```

Note that the first line of the function file begins with the word `function` followed by the function name. This first line differentiates a function from a script (regular program) – a script does not contain the `function` line. It is important that you make the function name the same as the filename you save it in.

An important difference between a script and a function is how they handle "local variables". For example, note that running your `sintest` script causes the variables `t` and `x` to appear in the base workspace (see them listed in the upper left pane of the MATLAB window), but your `greeting` function does not cause `test_variable` to appear there, even though it prints to the command window. Read the help for the global function, and **explain in your lab notebook** the difference between local variables, global variables, and the base workspace.

This type of function simply performs an operation. It does not have any inputs, and produces no outputs. Next, a function with an input is investigated.

(b) *Create a simple function of one argument.* Some functions are programs which require inputs. The inputs to a function are called arguments. Functions such as `cos` or `abs` have a single argument. Other functions have multiple arguments. The following function skeleton is provided for you to create your own function m-file. Your function should accept one argument, hopefully a number, and determine whether it is even, odd, or neither (e.g. non-integer). The `if`statement and `mod` command will be useful for this operation. The function should print “The input is even”, “The input is odd”, “The input is not an integer”, or “The input is not a number” as a result of your evaluation. You probably want to use the help command to lookup the built in commands *if* and *isnumeric.*

```
function even_odd(input)
 % Insert comments here describing how function is to be used
 result = mod(???);
 if (result==??)
```

Experiment with your new function by typing `even_odd (3)` etc. at the command prompt. This function has a single argument passed to it, in this case the number 3. At the command line or in another M-file, the argument used in the call of the function may be a number or a variable. While MATLAB is evaluating the function, the argument value is assigned to the local variable `input`. The variable input does not have meaning outside of the function. You have now learned how to pass arguments to a function.

(c) *Create a simple function of one argument, returning two values.* A third type of function returns a value to the command interface or program calling the function. The function definition line has a different form from the examples shown above. A function definition line is provided below to get your started.

Create a function which accepts a numeric input and evaluates how many times the number two goes into that input. The function should return the number of times 2 goes into the number and the remainder to the main program in the form of two variables: `two_fac` and `remainder`. For example, if the input is 33, the result should be 16 and 1. The returned value `two_fac` is the result of the *integer division* of `input` by 2.

You should be able to perform this operation using only two command lines in the function. The function `floor` will prove helpful.

```
function [two_fac,remainder] = two_div(input)
 % Insert comments here describing how function is to be used
 % Insert function commands
```

The function definition line contains a number of important elements. First, there is the familiar function definition (which should be the same as the filename it is saved in). A single argument (`input`) is passed to the function from the main program. There are also two defined outputs (or returned values) of the function (`2fac` and `remainder`). The syntax is important, so it is a good idea to understand how function definitions are constructed.

To call this function,type the following line at the command prompt (the same form is used to call the function from an M-file):

```
[two_fac,remainder] = two_div(input)
test = two_fac*2 + remainder
```

The first line tells MATLAB to call the function `two_div` with the argument `input`, and place the returned values into the variables `two_fac` and `remainder`. The second line verifies the returned values by calculating a variable `test`, which should be the same as the value store in `input`. Test your function to be sure it is operating properly. Demonstrate this to your instructor.

Instructor Verification (see last page)

3. Presenting Data Graphically

Often in lab you will have three types of data that you will want to plot on the same plot, Analytic (from a mathematical analysis), Simulation (from MATLAB, SIMULINK or PSpice), and Lab Data (measured in lab). Suppose a process is to be modeled, and the model verified. Your analysis says the data should fit:

$$x(t) = \sin(2\pi 440t + \pi/2)$$

Next suppose you run a simulation of the process being modeled, and the result of this simulation is the data in the file “simdata.txt” available on the course webpage. Within the file, the first line is a comment line. In the subsequent lines, the first value is *x*, the second is the time (in seconds) that *x* was measured. Type `help fileformats` to learn how to read in data from this type of file.

Finally you measure the following data in lab:

Lab Data	Time (ms)
+1.07	-4.17
-0.38	-3.55
-0.82	-2.92
+0.74	-2.30
+0.72	-1.67
-0.72	-1.05

(a) Create a single plot showing all of your data using a MATLAB script. Your script should perform the steps listed below. Save your script as dataplot.m.

(b) First, evaluate the equation at the correct times and store the results in an array called, e.g., `xxAna` (for Analytic).

(c) Read in the data from the simulation results file, and store the data in the array `test`. Note that the array `test` contains two vectors. We will access the two vectors by breaking test into `x` and `t` arrays using:

```
xxSim = test(:,1);    % Be sure you understand this notation!
ttSim = test(:,2);
```

(d) Store the lab data in two arrays, e.g. `xxLab` and `ttLab`, by typing it in by hand.

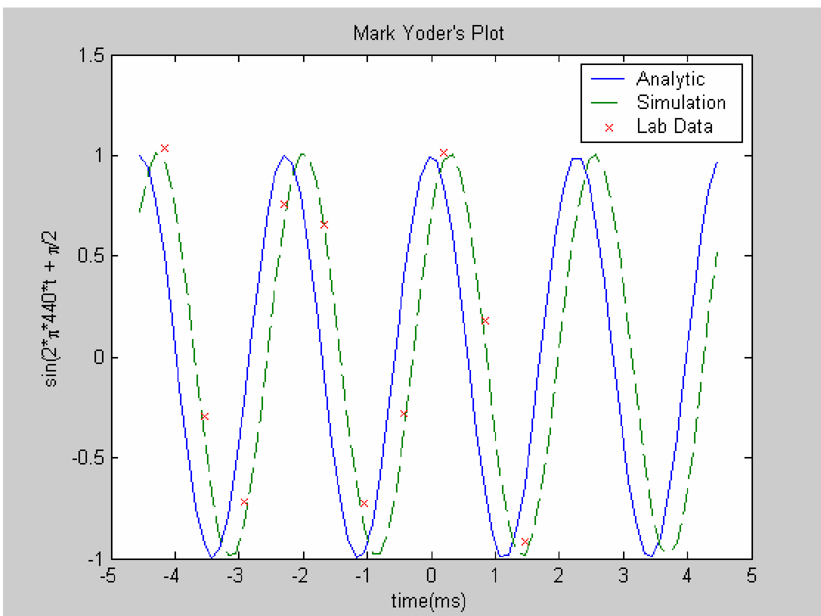
(e) Finally, plot all three on the same plot using:

```
plot(ttAna, xxAna, ttSim, xxSim, '--', ttLab, xxLab, 'x')
```

(f) Add a legend using:

```
legend('Analytic', 'Simulation', 'Lab Data');
```

(g) Label your plot using `xlabel`, `ylabel`, and `title`. (You can get π by entering "\pi".) Put your name in the title. Your plot should look something like:



Note the time axis is in ms, not in seconds!

Instructor Verification (see last page)

Note: We have shown you the above format because it is an industry standard to plot Analytic data as a solid line, Simulation data as a dashed line, and to plot individual Lab Data as points, not as a continuous line. **You are expected to follow this convention in later labs and courses.**

Report

Record the results of all of your work in your lab notebook. Tape any printout (graphs, for example) into the notebook as specified in the Lab Manual for the course. Be sure that all members of your lab group sign the lab notebook, and hand in one notebook per group at the end of lab (both members must keep up-to-date lab notebooks).

Lab 02 Introduction to MATLAB
Instructor Verification Sheet

Paste this page in your laboratory notebook along with your other lab data and observations.

Name ______________________________________ Date of Lab: ________________

Part 1(c) Make sure your plot is properly presented.

Verified: ____________________________________ Date/Time: ________________

Part 2(c) Verify operation of your function.

Verified: ____________________________________ Date/Time: ________________

Part 3(g) Present your plot. Tape your resultant plot in your lab notebook.

Verified: ____________________________________ Date/Time: ________________