CISC106 Summer 2009 Lab03

- Review the code examples from class.
- Some programs below are associated with a question. **Answer the questions** using comments below your code in the m-file.
- The office hours of the TAs and the instructor are on the class website. Visit us!
- **NOTE:** Every function comment section should contain, at a minimum, **three examples** of the function being called and the result of evaluating the call (only for parts 3-7, you do not need this for parts 1 and 2). These examples must include boundary conditions (as discussed in class). Your test files must cover **at least** these exact examples (otherwise, why did you choose them?) and possibly more. Testing is important.

# Problems

1. Use the plot function to create two image files as follows. Place each of the two plot descriptions in separate, well-named script files. In each plot, have a title and labeled axes. Create and fill a matrix for the x-axis and y-axis, then use them in the plot command as shown in class and in your text. Choose data that you find interesting, perhaps related to your major or a hobby (data from a lab, sports scores or grades vs. time, etc.).

   Each script file should contain all the commands to create one of your graphic files, so that you can create the file by simply running the script.

2. At a Matlab command line, type the expression `0.3 - 0.2 - 0.1` to see the result. Are you surprised? Use the fprintf function from Chapter 2.6 (Displaying Output Data) to examine each of the numbers involved. Change width and precision fields of the format specifier as described in the chapter, so that you can see the numbers with great precision. Once you have learned how to use fprintf to do this, start a diary file and show high precision versions of all the tenths from 0.1 to 1.0. Each should print on a separate line.

3. Calculate the volume of a sphere ($volume = 4/3\pi r^3$) using the command window and then in a script M-file. Then use the editor program **emacs** to write a function file. When you are done, create a diary file "sphere.txt" that shows all three ways.

### 0.0.1 Creating a file with emacs:

(a) Open an XTerm on strauss, and type "`emacs &`". If you get a "command not found" message, you may have to type "`/opt/bin/emacs &`" until you correct your path variable.

(b) The **&** is a special command suffix to let Strauss know that you want emacs to open in a new window and let you keep using the old xterm window, too.

(c) When you first see Emacs, the paragraph of text will tell you how to start running a self-directed tutorial[1]. Do this. The option to run the tutorial appears every time you open emacs. You can run the tutorial as many times as you like until all the material is familiar or you become bored out of your gourd. I suggest you also print out the emacs reference card from the class website and learn the keystroke commands; this will make it easier to work remotely.

[1] Your Unix text also has information on using emacs.

(d) Once you are comfortable with using emacs, use emacs to create a text file called **sphereVolume.m**. In this file write a Matlab function, just as you did using the Matlab file editor before when you wrote other functions. Save the file when you are done, and then go back to your Matlab window and test out the function.

(e) Use emacs to write a script-M file to test your sphere volume function.

4. Write a function M-file using the relational operators from class and the readings (`<`, `>`, `<=`, etc) above with an **if** statement. The function will use **fprintf** (from chapter 2.2) to show a message saying whether the input parameter, n, is ten or less; greater than ten but less than twenty; twenty or greater but less than thirty; or thirty or greater.

5. If you want to return a matrix m from some function, just set the output variable to the matrix with assignment. However, Matlab also offers the ability to return multiple values. Create a function file that contains something like this:

```
function [a b] = f()
  a = 4;
  b = 5;
```

Now in a diary try:

```
x = f()
```

and then

```
[x y] = f()
```

Now look up the built-in `max` function. **Explain** in the comments of your M-file for the function above exactly what a call to max evaluates to, and when[2].

6. Given a square matrix, write a function (and test script) to use nested loops to sum the elements of the diagonal (top left to bottom right.) Now write a second function (and test script) that will sum the other diagonal.

   You should use the eye() function to generate the identity matrix like we did in class to help with testing.

If your TA requires a paper copy, be sure that you have a printed copy of your function M-files, script M-files, image files, and diary files demonstrating your testing. All must be stapled together, with your name and lab section on the top page.

Be sure that you upload a copy of all the MATLAB function, script, image, and diary files to Sakai. Then, click submit ONLY ONCE to send these to your Sakai and your TA.

**On the first page of every printed copy for this course, your name, section, and TA's name must appear.**

[2]This is, of course, in addition to the comments about the function itself.)