CISC106 Fall 2008 Lab05

- 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. Your test files must cover **at least** these exact examples and possibly more. Testing is important.
- Review the **plot** command in your Chapman, section 2.11.

Problems

1. Write a recursive function using the following definition:

   Write a recursive MATLAB function **fact** that takes a positive integer `n` as argument, and returns `fact(n)` as defined:

   $$\text{fact(n)} = \begin{cases} \text{n, if } n <= 2 \\ \text{the product of n and fact(n-1) otherwise.} \end{cases}$$

   Thus `fact(4)` evaluates to 24.

   After you write fact, write a script that tests it. Demonstrate.

2. An important tool for computer users, as scientists, is the ability to compare how fast programs run. Though we think of computers as fast, there are many problems which machines could do better if our programs ran faster (e.g. weather tracking and prediction, automated interpretation of x-rays, optimizing internet traffic in a dense city network).

   NOTE: The timing runs for this lab must all be run **in Matlab on Strauss**. You may do so remotely, but do not submit timing runs from a Matlab running directly on your own computer.

   If you are interested in how long something takes, do not time it only once. Time it multiple times and record all of your results. Matlab makes this easy.

   Consider the following program, and draw yourself[1] a picture showing what it does:

```
%Create a vector to hold the times of three runs
times = zeros(1,3);

%set a baseline time for a timing run
start = cputime();

for index = [1:100]
  x = fact(index);
```

[1] Do not submit.

```
end

%calculate elapsed time and store
times(1) = cputime() - start;
%end of one timing run

disp(times);
```

Examine this program and run it to see how it works. If you don't understand it, first remove some semi-colons so you can see what is being assigned; then it is still unclear take your drawing to your TA or professor.

This program accomplishes the first timing run of three. **Add two more parts** to it so that it does two more timing runs: 2) calculating factorials from 1 to 200; and 3) calculating factorials from 1 to 300. (How long do you think these will take? Does your data confirm your expectation?). You will have a total of 9 data points.

Once this works, add a plot command, with title and axis labels. The x-axis should be the number of factorials computed; the y-axis should be time in seconds. At each tick on the x-axis you should have three data points (one for each run). **Always** show all your data points, not averages of your data[2].

After your script is working correctly[3], show your script generating the data in a diary, and plot *and* save the figure as a picture with the Matlab print command:

```
> print -dpng myFactPlot.png
```

You can look at your image file on Strauss using a browser to be sure it is correct.

3. Write a different factorial function, named fact2. Function fact2 will take one parameter, and will compute the same answer as fact, but in a different way. Inside the function, use a **for loop** to multiply all the numbers required (instead of using a recursive call).

   After you test your code with a test script (will this be difficult to write?) use a loop in the interpreter to see how many times you have to run the function to get a measurable time (something more than .1 seconds).

4. Create a script that will plot the times of running fact, fact2, and the built-in Matlab function factorial on the same arguments. Use the number of run times you determined in part 3. Time each function on three arguments, running the same number of times. Do three timing runs for each function on each argument. Now plot the performance of all three functions on the same graph (this will look a lot like your timing graph for fact, but will also have three times as many data points). Show your script generating the data in a diary, and save your plot to an image file for submission.

   Now create a second plot from the same data, but this time do not use the plot command, use the command from page 64 of your Chapman text that will provide a better view of the data. If you aren't sure which one you need, experiment. If you still aren't sure, bring the images from your experiments to your TA or professor.

[2] Sometimes points with deviation bars are acceptable, but a single average point is useless to a scientist.

[3] You can save yourself a lot of time by putting the plot commands into your (already working) script!

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()
```

   Note that I'm leaving off the semi-colons (why?).

   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[4].

6. Evaluate the following in the interpreter:

```
>> a = 'asdf'
>> b = 'wer'
>> c = [a;b]
>> c = [a;a]
```

   Do you understand the error? Read section 6.2 through 6.2.4 of your text. Then show in your diary how to correctly create the matrix that caused the error above.

7. 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.

8. Using your own diagonal functions, write a new boolean function to test whether a matrix is a magic square (see Wikipedia.org). Your new function will take a matrix as a parameter. You may write or use other supporting functions as needed. To be as efficient as possible, this function will stop as soon as it finds an incorrect sum[5].

9. Copy the image http://www.udel.edu/CIS/106/chandrak/08F/labs/fruits-picture.jpg into your directory. Copy the file lab05_image.m from the same www. Run it to see what it does (note: it is the same example covered in class). Now, modify the program and get input from the user for shrink factor, s. 's' can be 1, 2, 3, 4, 5, or 6. The program must take the input and shrink the image based on the input 's'. Run the program for atleast three shrink factors, include the corresponding shrunk images in your submission, along with the main program itself.

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

[5] You may NOT use break statements in this class. Learn to write loops instead. ;)

If your TA requires a paper copy, be sure that you have a printed copy of your (at least) nine function M-files, four script M-files, five (or more) image files, and diary files demostrating 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, imasge, 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.