CISC106 Fall 2009 Lab04

- 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. 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. files: fact.m, factCPUTimeTest.m, factCPUTimePlot.m, myFactPlot.png

   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.

   Using your fact.m from last week's lab (Lab03 Part 1), consider the following function, and draw yourself[1] a picture showing what it does:

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

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

  for index = [1:50]
    for count = [1:100]
      x = fact(index);
    end
  end

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

[1] Do not submit.

```
end
%end of factCPUTimeTest() function
```

Examine this function 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 if it is still unclear take your drawing to your TA or professor.

This function accomplishes the first timing run of three for factorials 1 to 50 (with a count of 100). **Add two more parts** to it so that it does two additional timing runs: 2) for factorials 1 to 75; and 3) for factorials 1 to 100. (How long do you think these will take? Does your data confirm your expectation?).

Add a for loop that surrounds the 3 different timing runs so that each timing run is executed 3 times. You will have a total of 9 data points in **times**, which is a 3x3 matrix.

Once this works, create a function **factCPUTimePlot** that takes a timing matrix as an argument and plots the data 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, **do not show an average**[2]).

After your factCPUTimePlot function is working correctly, show your factCPUTimeTest function generating the data in a diary, plot the data using your factCPUTimePlot function, and *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.

2. file: fact2.m

   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).

3. files: allFactCPUTimeTests.m, allFactPlot.png

   Create a function that will plot the times of running fact, fact2, and the built-in Matlab function factorial on the same arguments. The function should 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 function generating the data in a diary, and save your plot to an image file for submission.

4. files: sumDiagLeftToRight.m, sumDiagRightToLeft.m, sumDiagLeftToRightTest.m, sumDiagRight-ToLeftTest.m

   Given a square matrix, write a function (and test function) to use nested loops to sum the elements of the diagonal (top left to bottom right.) Now write a second function (and test function) that will sum the other diagonal. Show your test function executing in a diary file.

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

5. files: isMagicSquare.m, isMagicSquareTest.m

   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[3].

   Create and submit a test function for your magic square function.

6. file: whileInput.m

   Write a function that contains a while loop. The function will prompt the user for a number; if the number is zero or positive, it will store the number in a vector and ask for another number; but if the number is negative, the function will stop and return the vector of positive numbers previously stored.

   Record a diary file demonstrating your function working for at least 3 different tests.

This assignment is due on Sakai at midnight on October 8th. Please bring a paper copy to class on Friday October 9th (if your TA requires one). Be sure that you have a printed copy of your 12 function M-files, two image files, and 5 diary files demonstrating your testing. All must be stapled together, with your name, lab section, and TA name on the top page.

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

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

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