CS1674: Homework 1 - Programming

Due: 9/7/2016, 11:59pm

This assignment is worth 50 points.

In addition to your code, please also submit a `log.txt` file where you write down answers whenever the instructions below ask you to "write down" something. Pay close attention to whether you are allowed to use loops, in the below tasks. If the task doesn't explicitly mention that you are, you should not use loops. Remember to use the Matlab documentation whenever needed.

Please post on Piazza if you have any questions, rather than emailing the instructor directly, unless you have a concern you wish to keep private. I will monitor Piazza daily.

Numbers, matrices and functions:

1. Generate a 1000000x1 (one million by one) vector of random numbers from a Gaussian (normal) distribution with mean of 0 and standard deviation of 5. Use Matlab's `randn` function.
2. Add 1 to every value in this list, by using a loop. To determine how many times to loop, use Matlab's `size` function. Time this operation and print the number in the code. Write that number down.
   Use Matlab's documentation to find out how to time operations.
3. Now add 1 to every value in the original random vector, without using a loop. Time this operation, print the time and write it down. Use a different way to print the number than the method you used above. (See ways to print numbers at the beginning of the Matlab tutorial script.) Write down the number.
4. Copy this in your code: `A = rand(5, 3);` Then create a single command that ensures the sum in each row is (approximately) 1, without hard-coding any numbers except to denote along which matrix dimension (rows or columns) you're operating. In other words, I should be able to run your code on another 2-dimensional matrix, of a different size which you don't know, and it should still work. Hint: Use `repmat`.
5. How many changes do you need to make to your code in the task right above to ensure that the sum in each *column* is 1?
6. Create two matrices which when added together result in a matrix containing all numbers from 1 to 100. Each matrix should only be created with a single command.
7. Plot the exponential function 2^x, for *even* values of x smaller than 100.
8. Create a function that returns the n-th number in the Fibonacci sequence.

Images:

9. Read in this image into Matlab as a matrix, and write down its dimensions.
10. Convert the image into grayscale.
11. Find the darkest pixel in the image, and write its value and [row, column] in your answer sheet. Hint: Convert to a vector first, and use Matlab's `ind2sub` function. Use Matlab's `help` to find out how to use that function.
12. Use the function `sum` and a logical operator measuring equality to a scalar, to determine and write down how many pixels in the grayscale image equal the value 6.

13. Consider a 31x31 square (a square with side equal to 31 pixels) that is centered on the darkest pixel. Replace all pixels in that square with white pixels (pixels with value 255). Do this with loops.
14. Now use the code you wrote above to find one of several pixels with value 6. Find which of those pixels are at least 15 pixels away from the border of the image in any direction (not including the 6-valued pixel itself). You can use loops. Let's call these 15-away 6-valued pixels `inds` (you don't have to call them this in your code).
15. Write code to *randomly* choose one of the `inds` pixels.
16. Now consider another 31x31 square, but this time gray (e.g. with pixel values 150). Take the image with the white square in it. Replace the randomly chosen pixel from above, and the 31x31 square in the image that's centered on this pixel, with the gray square. This time you are NOT allowed to use loops. Note that you shouldn't run into border issues because of the 15-away code you wrote above.
17. Make a new figure, display the modified image (which includes both a white square and gray square), and save it to a file using `saveas(gcf, 'new_image.png')`.

    Loops (you're allowed to use them for the below tasks):

18. Create a script that prints all the values between 1 and 100, in random order, with pauses of 1 second between each two prints.
19. Generate two random matrices `A` and `B`, and compute their product by hand, using loops. Check your code by comparing the loop-computed product with the product that you get from Matlab's `A*B`.
20. Implement a function `my_unique` that returns the number of unique rows in a matrix, and returns another matrix with any duplicate rows removed. You cannot just call Matlab's `unique`.
21. Create another script which reads in an image. It computes the scalar average pixel value along each channel (R, G, B) separately. It then subtracts the average value per channel from the corresponding channel. Finally, it writes the image to a file `mean_sub.png`.