Exam 2 Additional Practice Problems

Disclaimer: The problems given on the exam may not be similar to those found here. The problems below are designed only to provide additional practice with topics (for example, for and while loops) most likely to be covered on the exam.

1) The ancient Babylonians discovered that the square root of any positive integer, S, can be calculated using a clever iterative sequence. If x_0 is any positive starting guess, then

$$x_1 = \frac{1}{2}\left(\frac{S}{x_0} + x_0\right)$$
$$x_2 = \frac{1}{2}\left(\frac{S}{x_1} + x_1\right)$$
$$x_3 = \frac{1}{2}\left(\frac{S}{x_2} + x_2\right)$$
$$\vdots$$
$$x_{n+1} = \frac{1}{2}\left(\frac{S}{x_n} + x_n\right)$$

As n becomes very large, $x_n \approx \sqrt{S}$.

Write a program that approximates the square root of a given positive integer:

1. Prompt the user for a positive integer whose square root will be approximated.
2. Then ask the user for a positive starting guess.
3. If the guess is negative or zero, ask the user to reenter the guess.
    a. If the user fails to enter a positive guess after 3 attempts, output a warning and take the absolute value.
    b. If the user enters 0 on the 3rd attempt, output an error and terminate the program.
4. Prompt the user to enter the desired accuracy of the approximation.
5. Using the Babylonian sequence above, approximate the square root until the approximation is within the desired accuracy. Hint: Compare your approximate result to the actual square root by finding the difference between the two. The difference should be less than (or equal to) the desired accuracy.
6. Output the final approximation and the number of iterations it took to reach that value.
7. Create a menu asking the user if they'd like to repeat the program.
    a. If the user selects yes, repeat the program. Each time the program repeats, save the integer the user entered in a vector called Integers, and save its final square root approximation in a vector called SquareRoots.
    b. If the user chooses not to repeat the program, save to a file named Results.mat the two vectors, Integers and SquareRoots. These vectors

should contain the integers and their approximated square roots for all previous times the program was run.

c. If the user closes the menu without making a selection, repeat the menu and output a message that the user must make a selection.

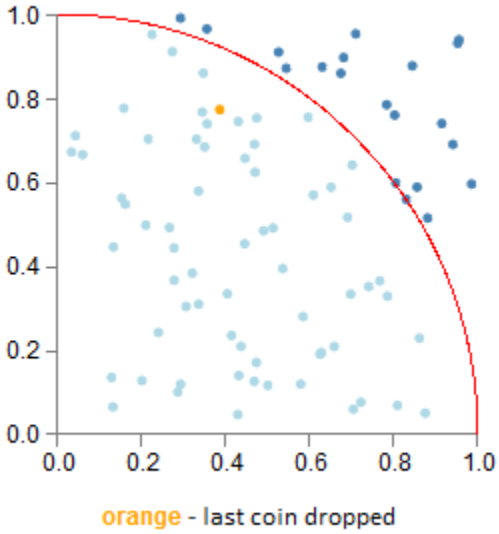
2) Write a program which accomplishes the following:
Note: the use of the functions min, max, and mean, as well as the use of logical arrays/masking is not allowed.
    1. Generate a row vector containing ten random integers with values ranging from 1 to 10.
    2. If the vector contains at least one 10 do the following:
        a. Print a statement that the array contains at least one 10
        b. Find the second highest number in the vector
        c. Find the average value of all the elements of the vector
        d. Output these values to the command window
        e. Repeat steps 1 and 2 until the vector of random integers contains no 10s
    3. If the vector contains no 10s:
        a. Print a statement that the vector contains no 10s
        b. Find the highest number in the vector
        c. Find the lowest number in the vector
        d. Output these values to the command window
        e. Output the number of times the randomly generated vector contained at least one 10
        f. Then end the program

3) Write a program that generates and manipulates prime numbers.
Note: Prime numbers are positive integers that are only divisible by 1 and themselves.
    1. Prompt the user for the index of the prime number they would like to generate (e.g. $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, $p_4 = 7$, etc.). Also, note that the first prime number is 2.
    2. If the user enters a nonpositive* number, ask the user to enter a positive number instead.
    3. If the user has not entered a positive number after 5 attempts:
        a. If the last number entered is negative, produce a warning and take the absolute value
        b. If the last number entered is zero, produce an error and end the program
    4. Given the index, i, just entered by the user, generate p_i and output it to the command window
    5. Finally, calculate and output the average of p_{i-1}, p_i, and p_{i+1}. (The use of the function mean is not allowed).

** Nonpositive means either negative or zero.*

4) Using a sequence of random numbers it is possible to estimate the irrational constant pi, using an algorithm known as a Monte Carlo simulation. (This method gets its name from the well-known casino in Monte Carlo, Monaco of James Bond fame.)

The general procedure is as follows: imagine a square grid 1 meter wide by 1 meter long with intersecting gridlines placed one millimeter apart, and define one of the corners of the square as the origin. Now, let's say we randomly drop a coin in such a way so that it has an equal chance of falling anywhere on the grid. Then measure the coin's distance from the origin, making a note of whether its distance from the origin is 1 meter or less. Now repeat this many, many times, keeping track of the total number of coin throws, n, and the number of times the coin is within 1 meter of the origin, a. After tossing the coin many, many, many times, we will find that the quantity, $4 \cdot \frac{a}{n} \approx \pi$. (As a fun exercise, think about why this is the case. Hint: the area of a quarter circle with a radius of 1 is $\frac{\pi}{4}$.)



Write a program that approximates pi to within 5 decimal places using a Monte Carlo simulation:

1. Prompt the user for the number times they'd like to run the experiment.
2. Then write a function called CoinToss which simulates the tossing of a coin onto a grid. This function takes no input and returns a single output, the distance of the coin from the origin.
3. Within this function, generate a random integer between 0 and 1000. This will be the x-coordinate of the dropped coin (in millimeters).Then generate another random integer between 0 and 1000. This is the coin's y-coordinate.
4. Determine the coin's distance from the origin. Hint: Use the distance formula, $d = \sqrt{x^2 + y^2}$.

5. Within your main script, “toss” the coin many times by using a loop. Keep a running total of the total number of coin tosses and the number of times the coin was within 1 meter of the origin.
6. Use the formula $q = 4 \cdot \frac{a}{n}$ to approximate pi. (Remember, n is the total number of tosses, and a is the number of times the coin lands within 1 meter of the origin.)
7. Stop tossing the coin only when your approximation is accurate to within 5 decimal places.
8. Output both the final value of your approximation and the number of coin tosses to reach that value.
9. Repeat the entire experiment the total number of times that the user specified at the beginning of the program. Keep track of the total number of times the coin was tossed for each experiment in a vector called NumTosses.
10. Once completed, output NumTosses to a .csv file named Results.csv.