Homework (30pts) chapter 5 _ text book (Lent)

- Write programs and their outputs. Format each program as instructed in chapter 1.

Q 1(5pts) 5.4

Q 2(5pts) 5.7

Q 3 (5pts)-5.9

Q4 (5pts)- 5.20

Q5(5pts) – Write a program to calculate a factorial of any number N

Q6 (5pts)- Write a program to determine the cost of automobile insurance based on drivers age and number of accidents that the driver had. The basic insurance charge is $500. There is a surcharge of $100 if the driver is under 25 and with an additional sub charge for accidents:

Number of accidents	Accidents sub charge
1	50
2	125
3	225
4	375
5	575
6	No insurance

Homework (10pts) chapter 4 _ text book (Lent)

Q 1(5pts) 4.3

Q 2(5pts) 4.5

4. **Fibonacci numbers.** The Fibonacci numbers are the numbers in the sequence

$$0, 1, 1, 2, 3, 5, \ldots,$$

where the first two Fibonacci numbers are defined to be 0 and 1 and each subsequent number is obtained by finding the sum of the two previous Fibonacci numbers. These numbers are important in biology as well as in computing algorithms. Starting from `fib(1)=0` and `fib(2)=1`, write a program, `fibonacci.m`, that computes the first N Fibonacci numbers and finds their sum.

5. **Toss a coin.** Write a program, `CoinToss.m`, that asks the user to call a coin toss in the air (h or t). The program should then honestly (randomly) generate a head or tail using `randi`, and declare whether the user won or lost. Allow the user to continue to play until the user wants to stop.

6. **Partial sum of integers.** Write a program, `SumUpTo.m`, that uses a `for` loop to calculate the sum of the positive integers up to N. Set the value of N near the beginning of the program. ($N = 100$ was Gauss's famous trick.) That is, find and print:

$$P(n) = \sum_{k=1}^{n} k$$

7. **Finding prime numbers.** Write a program, `FindPrimes.m`, that examines each integer from 1 to N to see if the number is a prime. If it is prime (and only then), print out the number. Use the MATLAB function `isprime(k)`, which returns the value `true` if k is prime. The program should print out all the primes less than or equal to N. Use $N = 100$ as a test case.

```
>> FindPrimes
N= 100
Found Prime: 2
Found Prime: 3
Found Prime: 5
Found Prime: 7
Found Prime: 11 (etc.)
```

8. **Print laughing.** Write a program, `Laughing.m`, that types out a line with N Ha!s separated by spaces. Use a `for` loop to construct the string. It should work with any positive value of N.

```
>> Laugh
Ha! Ha! Ha! Ha! Ha! Ha! Ha! Ha!
```

9. **First π approximation.** Write a program, `PiSeries1.m`, that calculates an approximation to π by using the Euler solution to the Basel problem:

$$\frac{\pi^2}{6} \approx \sum_{k=1}^{N} \frac{1}{k^2}$$

Compute the sum for $N = 100$, 1000, 1×10^6, and 1×10^7.

10. **Second π approximation.** Write a program, `PiSeries2.m`, that calculates an approximation to π by using the Gregory-Leibniz series:

$$\frac{\pi}{4} \approx \sum_{k=0}^{N} \frac{(-1)^k}{2k+1}$$

Compute the sum for $N = 100$, 1000, 1×10^6, and 1×10^7.

11. **Calculate absolute value.** Write a program, `absvalue.m`, that gets a number from a user, then finds and displays its absolute value. Use an `if-then-else` construction rather than just the built-in `abs` function.

12. **Plot piecewise function.** Write a program, `plotPiecewise.m`, that plots the piecewise function $p(x)$ defined as follows, in the range $x = [-2, 2]$.

$$p(x) = \begin{cases} 2x+3 & \text{if } x \in [-3/2, -1] \\ -sin(\pi x/2) & \text{if } x \in (-1, 1) \\ 2x-3 & \text{if } x \in [1, 3/2] \\ 0 & \text{otherwise} \end{cases}$$

13. **Rolling dice (improved).** Write a program, `ThrowDice2.m`, that prints the results of rolling two fair dice (use `randi`). Also print out "boxcars" for double sixes and "snake eyes" for double ones.

14. **Drawing cards.** Write a program, `DrawCard.m`, that randomly selects a single card (from a full deck). Print the result as ace-spades, 5-hearts, king-diamonds, jack-clubs, etc.

15. **Number guessing game.** Write a program, `guessing.m`, that does the following: The computer generates a random integer between 1 and 20 (inclusive). The computer displays "I am thinking of a number between 1 and 20." It prompts the user to input a guess. While the guess is not equal to the computer's number, it tells the user whether the guess was too low or too high, and prompts for another guess. When the user guesses the right number, the computer congratulates the user and ends the game. (Hint: Use a `while` loop.) A run of the program (for which the user guessed 10 and 15 before correctly guessing 13) might look something like this:

```
>> guessing
I am thinking of a number
 between 1 and 20.
Enter your guess: 10
Too low
Enter your guess: 15
Too high
Enter your guess: 13
Right!
```

16. **Improved guessing game.** Write a program, `ImprovingGuessing.m`, in which the guessing game automatically repeats over and over until the user quits. Add the ability for the user to quit the program by inputting a `"q"` or a `"0."` At that point, print out the *average* number of guesses the user needed. In the previous example, the user only needed three guesses. If they have adopted a clever strategy, what is the maximum number of guesses the user should need on average?

17. **Counting by ones.** Write a program, `Countem.m`, that types out a line with N integers starting with n0, separated by commas and spaces. Use a `for` loop to construct the string.

```
>> Countem
N0= 5, N= 10
5, 6, 7, 8, 9, 10, 11, 12, 13, 14
```

18. **Rock-paper-scissors.** Write a program, `rps.m`, which plays the game rock-paper-scissors with the user. The user enters an `"r," "p," "s,"` or `"q"` to quit the game. The computer honestly (randomly) selects a choice, then tells the user who won, and prompts for another input. The computer keeps a running total of human wins and losses. Develop the program by writing some simpler versions first.

    a. **Version One.** Write a program, `rps1.m` that runs a single turn of the game. Generate the computer's choice using `rand`. Prompt the user for the choice `"r," "p,"` or `"s."` Decide who won the turn (human or computer) and display the results.

    b. **Version Two.** Write a program, `rps2.m`, by adding to the user's options the choice `"q"` to quit. The program now plays the game over and over until the user enters a `q`.

    c. **Version Three.** Write a program, `rps3.m`, adding the feature that the program keeps a running total of wins and losses by human and computer and reports the cumulative results at the end of every turn.

19. **Hangman.** Write a program, `hangman.m`, that runs a game of hangman between two computer users. With the other player looking away, the first user should input a word, which is then cleared from the Command window. On each guess, the player should have the choice of guessing a word or a letter. Before each guess, the player should see a display of the word with asterisks in place of the letters the player has not guessed and the correct guesses so far filled in, and a list of all the letters the player has guessed up to that point.

20. **Monte Carlo integration.** One way to estimate the value of definite integrals in spaces with many dimensions is the Monte Carlo integration method. We illustrate the method here in two-dimensions. Consider the task of estimating the area of a circle of radius R. Inscribe the circle in a square with sides of length $2R$. To estimate the area, randomly pick N points in the square (use `rand`). Find how many points N_{in} are in the circle. The estimate for the area is then the fraction of all the points that are in the circle times the area of the square, $A \approx A_{MC} \equiv (N_{in}/N)4R^2$. Write a program, `MonteCarloCircle.m`, that implements this algorithm. Tabulate A_{MC} for $N = 10^k$ where $k = \{1, 2, \ldots, 7\}$. Make a graphical representation of the points "raining down" on the square with points inside the circle being a different color than points outside the circle.

21. **Pick a number.** Write a program `pickaNumber` that implements the following game. The program asks the user to mentally pick a small whole number n, but not report it. The program randomly picks 5 integers between 2 and 6: a, b, c, d, and e (don't disclose them to the user yet). The program then asks the user to do the following steps.

    1. Multiply the number n by a.
    2. Add b to the result.
    3. Multiply the result by c.
    4. Add d to the result.