Name: Yashika Patil Roll No: 02115374

Solutions to Homework 1

Problem 1

The files for this problem are saved as Q1_C: meaning it's Question 1 part C. Q1_D: Question 1 part d.

Part (a)

Solution:

$$\hat{I}_1(N) = V \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1 + x_i^2}$$

```
This is the latex script that I used in jupyter file:

\begin{align*}
  \hat{I}_1(N) = V \frac{1}{N} \sum_{i=1}^{N} \frac{1}{1 + x_i^2}\\
\end{align*}
```

Part (b): Pseudocode

```
N: Number of iterations
Initialize variables: sum = 0, V = 2.0, dx = 2.0 / N, func_value = 0,
integral = 0
// Monte Carlo integration loop
for i = 0 to N - 1:
    // Generate a random xi in the range [-1, 1]
    xi = (random() * 2) - 1
    // Evaluate the function value at xi
    func_value = 1 / (1 + (xi * xi))
    // Accumulate the sum
    sum = sum + func_value
// Calculate the integral approximation
integral = sum * (V / N)
// True value of the integral (for comparison)
true_integral = M_pi / 2.0
// Calculate absolute error
absolute_error = absolute_value(true_integral - integral)
// Print results
```

Part (c)

```
// C program for computing integral using Monte Carlo Integration

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#ifndef M_PI
#define M_PI (3.14159265358979323846)
#endif
int main(int argc, char **argv)
{
    int N = atoi(argv[1]);
    srand(time(NULL));

    double sum = 0;
    double V = 2.0;
    double dx = 2.0/ (double)(N);
    double func_value = 0;
    double integral = 0;
    for (int i = 0; i < N; i++)
    {
        double xi = (((double)rand() / (double)RAND_MAX) * 2) - 1;
        double func_value = 1 / (1 + (xi * xi));
        sum = sum + func_value;
        // printf("Iteration %d: xi = %f func_value = %f sum = %f\n",
i, xi, func_value, sum);
    }
    double true_integral = M_PI / 2.0;
    integral = sum * (V/N);
    double absolute_error = fabs(true_integral - integral);
    printf("%d %f\n", N, absolute_error);
    return 0;
}
```

Part (d)

The log-log plot of N vs absolute error is noisy since both I1(N) and E(N) are random variables.
The script file:

```
#!/bin/bash
i=1
while [[ i -le 30 ]]
do
    ./Q1_C $((2**i))
    ((i = i + 1))
done
```

Part (e)

The script file:

```
#!/bin/bash
i=1
while [[ i -le 30 ]]
do
    ./Q1_C.sh >> numbers$i.dat
    ((i = i+1))
done
```

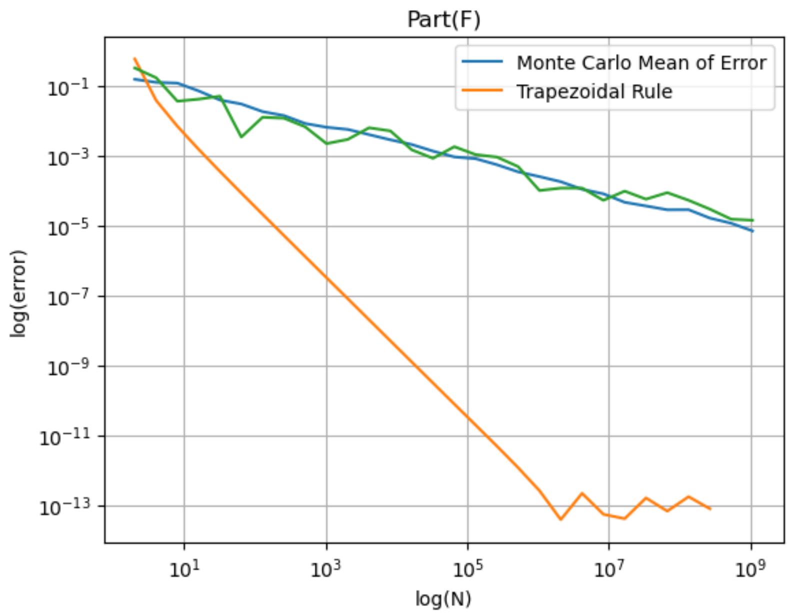
Part (g)

The script file, runtime.sh:

```
#!/bin/bash
TIMEFORMAT=%R
gcc -o pc Q1_C.c
echo "N Runtime" > new.dat
i=1
while [[ i -le 30 ]]
do
    N=$((2**i))
    echo -n "$N " >> new.dat
    { time ./pc $N; } 2>> new.dat

    ((i = i + 1))
done
```

Graph for Part (d), Part (f) and Part (g)



Problem 2

The files are saved as Q2.sh, Q2.

When N=100

Expected Average Snowfall: 2.114003
Expected Average Snowfall: 1.957884
Expected Average Snowfall: 2.253699
Expected Average Snowfall: 1.769913

N = 1000000.

Expected Average Snowfall: 2.058497
Expected Average Snowfall: 2.057429
Expected Average Snowfall: 2.063240
Expected Average Snowfall: 2.053507

When N=10000

Expected Average Snowfall: 2.035686
Expected Average Snowfall: 2.035686
Expected Average Snowfall: 2.027901
Expected Average Snowfall: 2.027901
The expected average snowfall becomes more stable as the number of samples increases since we used Monte Carlo integration, and increasing number of samples brings the estimate closer to the true value.