EE1103: Numerical Methods

Programming Assignment # 2

AMIZHTHNI P R K, EE21B015
Collaborators:
ANIRUDH B S, EE21B019
ANKITA HARSHA MURTHY, EE21B020

February 26, 2022

Contents

List of Figures

List of Tables

1 Problem 1

To estimate π by playing darts game.

In order to emulate throwing darts and estimating π,

- Generate random samples *x,y* where x and y are uniformly distributed in [0, 1) and *(x,y)* denote a dart's location on the square grid. Use rand() for generating random numbers and invoke srand(3141592653) as the seed for the random number gener-ator to compare results among your teammates by running your code on onlinegdb.
- Imagine a circular arc constructed using the equation

$$x^2 + y^2 \leq 1 \tag{1}$$

as shown in Fig 1.

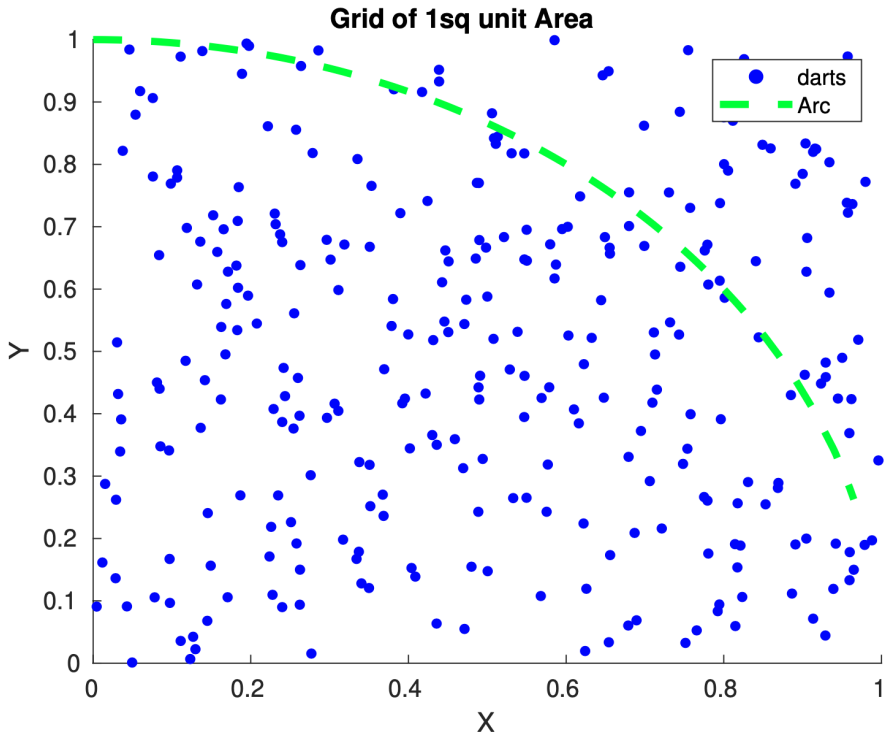

Figure 1: Points on the grid denoting landed darts ($n = 300$).

- Iterate the sample generation to simulate the n throws of the dart uniformly in the unit square.
- Let n_{arc} denote the number of darts that land inside the arc. Since the darts land uniformly, the ratio of number of samples inside the arc to those in the square will be proportional to their respective areas. π can be estimated by the following equation

$$\frac{n_{arc}}{n} \approx \frac{\text{Area of Quarter}}{\text{Area of Square}} = \frac{\pi}{4} \tag{2}$$

Let $\hat{\pi}_n$ denote the estimate of π using n throws of the dart. We have

$$\hat{\pi}_n = 4 * \frac{n_{arc}}{n} \tag{3}$$

1 (a)

(i) Estimate π using $\hat{\pi}_n$ for $n = 10^2, 10^3, ..., 10^5$. Test out the maximum value of n, you are able to reach without any errors and tabulate your results.

(ii) Plot $\hat{\pi}_n$ as a function of n ($\log_{10}$ Scale)

1 (b)

(i) Compute the absolute error % $= \frac{|\hat{\pi}_n - \pi|}{\pi} * 100$ and tabulate along with (a).

(ii) Plot the absolute error % as a function of n ($\log_{10}$ Scale)

1.1 Approach

Estimating Pi with Monte Carlo simulated dart throwing!
In this problem, we estimate π using random number generation, by throwing darts.

We use a common value for seeding the random function, and generate two random values and assign them to the co-ordinates of the dart. This can be juxtaposed with the act of throwing darts at a unit square and using the likelihood of it falling inside a circle to approximate the value of π. This creates a Continuous random variable and the probability of the dart falling in a region is proportional to its area.
The area of the quarter circle when divided by the total area of the square and multiplied by 4 gives an estimate of π.

1.2 Algorithm

The algorithm used for the Problem 1 is given below:

Algorithm 1: Approximating π as $\hat{\pi}_n$

STEP 1: We get a single input N from the user (in the log_{10}) scale for number of random variables to be sampled on

STEP 2: use rand() and srand() to create 10^N number of samples

STEP 3:

$N \leftarrow input, i \leftarrow 0, a, b \leftarrow 0, 0$

$ea = \mathbf{0}$

while $i \leq N$ **do**

  $x, y \leftarrow randomnumbers$;

  $circ \leftarrow x^2 + y^2$;

  if $circ \leq 1$ **then**

    $a \leftarrow a + 1$;

    $b \leftarrow b + 1$;

  else

    $b \leftarrow b + 1$;

  end

  $piest \leftarrow \frac{4a}{b}$;

  $ea \leftarrow \frac{piest - \pi}{\pi}$;

  $i = i + 1$

end

STEP 4: Once the for loop ends, the final value of pi estimate and error are transferred to a variable which is printed.

1.3 Results

We plot the graphs showing the Estimates of π and Approximate Errors In Figure 2 and 3. The individual values obtained for code snippet 1 are entered in google sheets against the N values to obtain the graphs.
The results of estimates of π and approximate errors are also summarized in Table 1.

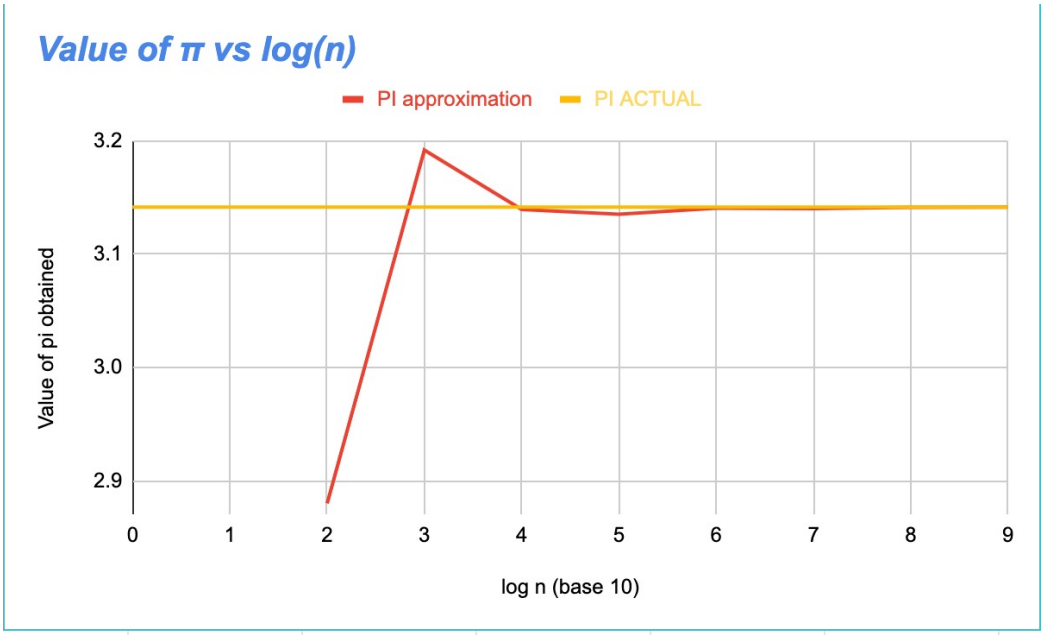

Figure 2: Number of terms considered while approximating π vs Estimates of π

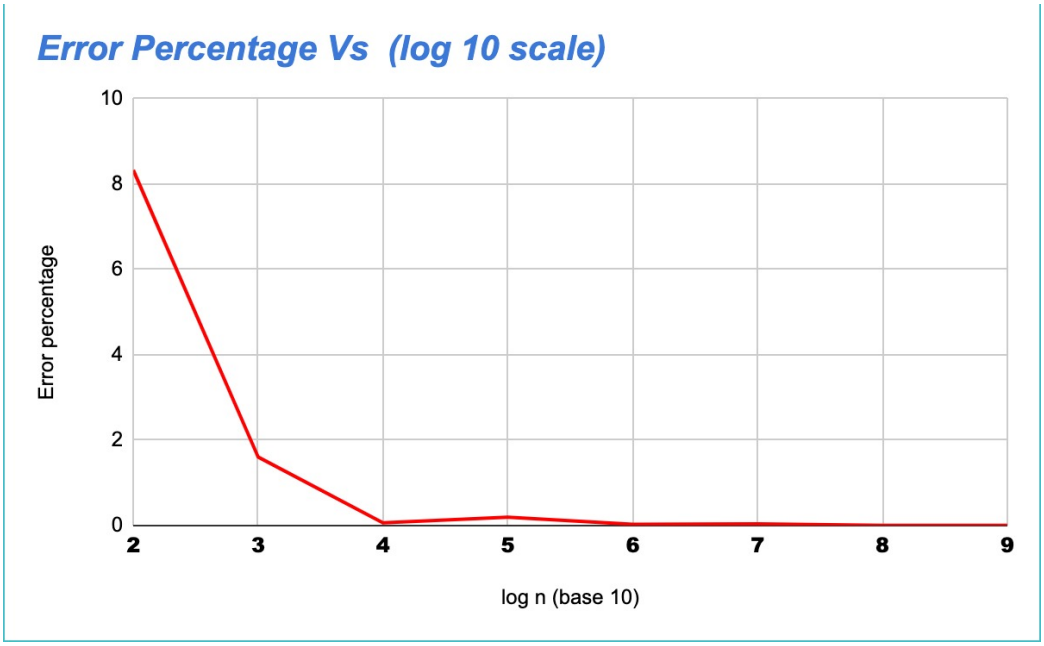

Figure 3: Number of terms considered while approximating π vs Approximate Errors

Table 1: N vs Estimates of π and Approximate Errors

N	ESTIMATES OF π	APPROXIMATE ERRORS
10^2	2.880000	8.326749
10^3	3.192000	1.604513
10^4	3.139600	0.063427
10^5	3.135440	0.195842
10^6	3.140652	0.029944
10^7	3.140396	0.038079
10^8	3.141517	0.002418
10^9	3.141575	0.000566

1.4 Inferences

We deduce the following inferences in this experiment:

- The sharp decline in the error percentage, even in the logarithmic scale, as the number of random samples is increased as shown in Figure 3 shows that the value of π is estimated to a good approximation and also *extremely fast.*
- The answers obtained depend solely on two factors: the seed taken in srand() function and the actual value of PI taken. As recommended, we have chosen MPI as the correct value of PI. Radically different answers are obtained when these factors are varied.
- Another important thing to note is that the final values are different when another `for` loop is introduced for printing the results for all values of N i.e. when we use a loop as in code snippet 2, we get a different plot as in Figure 4. Though this change is minor, we comment that the additional loop increases the time and space complexity.

- **Maximum power of 10** Outputs without error are obtained till the 9^{th} power of 10, for all values greater than which the program aborts or runs out of memory space.

1.5 Code

1(a)(i) 1(b)(i). The code used for estimating π by throwing darts is mentioned in Listing 1.

```
// C program to generate random numbers
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


// Driver program
int main (void)
{
  float x, y, circ, piest, ea, d1, d2;
  int a = 0, b = 0, j;  //a=number of darts positioned inside the circle, b=total number of darts
  printf( "Enter a value of log(N):");
  scanf("%d", &j);
  printf ("VALUE OF N\t ESTIMATE OF PI\t APPROX ERROR\n");

  srand (3141592653);   //common seeding value to ensure uniformity

  for (int i = 0; i < pow (10, j); i++) //for loop that generates the values taken by N
  {

    x = (float) rand () / (RAND_MAX); //generating random number in (0,1) for X coordinate of dart
    y = (float) rand () / (RAND_MAX); //generating random number in (0,1) for Y coordinate of dart


    circ = pow (x, 2) + pow (y, 2);
    if (circ <= 1)    //checking condition for dart within circle
      {
       a++;  //every time a dart falls inside the circle we add one to the circle dart count
       b++;  //every time a dart falls inside the circle we add one to the total dart count
      }

    else
      {
       b++;  //every time a dart falls outside the circle we add one to the total dart count
      }
```

```
    piest = (float) 4 *a / b; //approximating the value of pi for the URV
    ea = fabs (piest - M_PI) / M_PI * 100;    //Approximate error of value
    ↪ calculated from the standard value of pi
    d1 = piest;   //this variable stores the values of pi approximation in
    ↪ every iteration and prints the final value alone through the printf
    ↪ function below
    d2 = ea;  //this variable stores the values of approximate error in
    ↪ every iteration and prints the final value alone through the printf
    ↪ function below

      }
  printf ("10^%d\t\t %f \t %f\n", j, d1, d2);
  return 0;

}
```

Listing 1: Code snippet 1 used in the experiment 1.

1(a)(ii) and 1(b)(ii). The code used for graphing estimates of π versus the number of iterations is mentioned in Listing 2.

```
// C program to generate random numbers
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>


// Driver program
int main (void)
{
  float x, y, circ, piest, ea, d1, d2;
  int a = 0, b = 0, j;  //a=number of darts positioned inside the circle,
  ↪ b=total number of darts

   //common seeding value to ensure uniformity
  for (int j = 2; j < 8; j++)  //for  plotting values from 100 to 10^8
  {  srand (3141592653);
      for (int i = 0; i < pow(10,j); i++) //for loop that generates the
      ↪ values taken by N
      {

        x = (float) rand () / (RAND_MAX); //generating random number in
        ↪ (0,1) for X coordinate of dart
        y = (float) rand () / (RAND_MAX); //generating random number in
        ↪ (0,1) for Y coordinate of dart


        circ = pow (x, 2) + pow (y, 2);
        if (circ <= 1)    //checking condition for dart within circle
          {
```

```
27            a++;  //every time a dart falls inside the circle we add one to
              ↪  the circle dart count
28            b++;  //every time a dart falls inside the circle we add one to
              ↪  the total dart count
29           }
30
31         else
32           {
33            b++;  //every time a dart falls outside the circle we add one to
              ↪  the total dart count
34           }
35         piest = (float) 4 *a / b; //approximating the value of pi for the
           ↪  URV
36         ea = fabs (piest - M_PI) / M_PI * 100;    //Approximate error of
           ↪  value calculated from the standard value of pi
37         d1 = piest;   //this variable stores the values of pi approximation
           ↪  in every iteration and prints the final value alone through the
           ↪  printf function below
38         d2 = ea;  //this variable stores the values of approximate error in
           ↪  every iteration and prints the final value alone through the
           ↪  printf function below
39
40
41         }
42       FILE* data=fopen("estimate_pi.txt","a+");  //this inputs all the
         ↪  final values of approximation for a given number of tries to a
         ↪  file
43       fprintf(data,"%d\t%f\n",j,d1);
44
45       FILE* data2=fopen("aerror_pi.txt","a+");  //this inputs all the final
         ↪  approximate errors for a given number of tries to a file
46       fprintf(data2,"%d\t%f\n",j,d2);
47   }
48
49
50
51     FILE *pipe = popen("gnuplot -persist", "w");    //this plots the  graph
       ↪  using GNUPLOT for  values of approximation vs iterations
52     fprintf(pipe, "set title \"PI APPROXIMATION VS ITERATIONS\"\n");
53     fprintf(pipe, "set xlabel \"ITERATIONS EXPRESSED IN LOG_{10}
       ↪  SCALE\"\n");
54     fprintf(pipe, "set ylabel \"CORRESPONDING APPROXIMATION OF PI\"\n");
55     fprintf(pipe, "set grid\n");
56     fprintf(pipe, "plot 'estimate_pi.txt' with lines title 'pi estimates',
       ↪  3.141592653 title 'pi actual' lt  rgb 'green'\n");
57
58
59     FILE *pipe2 = popen("gnuplot -persist", "w");   //this plots the  graph
       ↪  using GNUPLOT for approximate errors vs iterations
60     fprintf(pipe2, "set title \"error percent vs ITERATIONS\"\n");
```

```
61      fprintf(pipe2, "set xlabel \"log of iterations\"\n");
62      fprintf(pipe2, "set ylabel \"error percent of pi\"\n");
63      fprintf(pipe2, "set grid\n");
64      fprintf(pipe2, "plot 'aerror_pi.txt' with lines\n");
65
66
67    return 0;
68
69  }
```

Listing 2: Code snippet 2 used in the experiment 1.

1.6 Contributions

Complete efforts were put in by me - including Coding, Plotting and Latex report preparation.

1.7 Alternate Methods

When an additional `for` loop is included, i.e. when results are not accounted for individually, we witness infinitesimal changes in the final value. Thus, for the final tabulation and plotting, the individual answers are used. Below is the output which is obtained when GNUplotting is done for Listing 2. We consolidate that this error is due to the additional looping which in turn increases the time complexity. We also take note that this requires much precision and must be done with care.

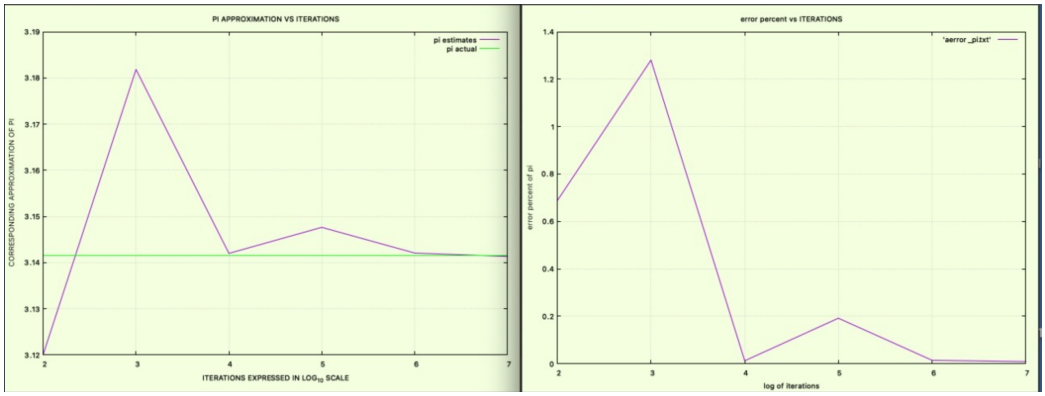

Figure 4: Plots obtained when an additional loop is considered

2 Problem 2

Generate samples from a *standard normal distribution* from uniform random samples, using the *Box-Muller transform.*

- Generate two sets of uniform distribution (say U_1 and U_2) in [0,1).
- Obtain a standard exponential random variable from U_1

$$E = -\log(U_1) \tag{4}$$

- Generate the two sets of standard normal variables using the below transform :

$$X = \sqrt{E}\cos(2\pi U_2) \tag{5}$$
$$Y = \sqrt{E}\sin(2\pi U_2) \tag{6}$$

- X and Y will be independent zero-mean unit variance Gaussian random variables (or standard Normal random variables).

2 (a)

Generate n = $100, 1000, 10000$ samples of X defined above, use srand(0) as the seed for comparison of outputs.

(i) Plot histograms for the samples generated.

2 (b)

(i) Plot the Empirical Distribution Function (EDF) of X (to approximate the Cumulative Distribution Function) for values of n = 10, 100, 1000. You can choose the x-axis range to be [4,4], since most of the probability mass of the standard normal lies in this interval.

(ii) Compute the empirical estimate for Erf(1) and Erf(2) for n = 10000 samples, as the fraction of the generated samples that are less than or equal to 1 (and 2 respectively). Equivalently, these are just the EDF values at 1 and 2 respectively.

You can use the following procedure for obtaining the EDF plot.

- Export the normal distribution values (from **a**) to Google sheets/ Microsoft excel.
- Sort the data in ascending order.
- Create a column for EDF and fill the column with increment of $\frac{1}{n}$ for every row against the sorted normal random variables.
- Plot EDF along the vertical axis and sorted values of X along the horizontal axis
- If you are using Google Sheets for plotting, you can use line plot for (b), since there is no step plot option available.

2.1 Approach

We use the *Box Muller transformation* to generate two independent zero-mean unit variance Gaussian random variables (or standard Normal random variables). We then plot one of these variables, denoted by X in the form of histograms for three values of number of samples and also their Empirical Distribution Graphs. Using the EDF, we use approximation to find the Erf.

2.2 Algorithm

The algorithm used for the above Problem 2 of the assignment is given below

Algorithm 2: Box muller method of transformations

$N \leftarrow input, i \leftarrow 0$
while $i \leq N$ **do**
  $U1, U2 \leftarrow randomnumbers\ E \leftarrow \sqrt{-log_{10}(U1)}$
  $X \leftarrow Ecos(2\pi U2)$
  $Y \leftarrow Esin(2\pi U2)$
  $i = i + 1$
end

2.3 Results

(a) The histograms for X are given below.

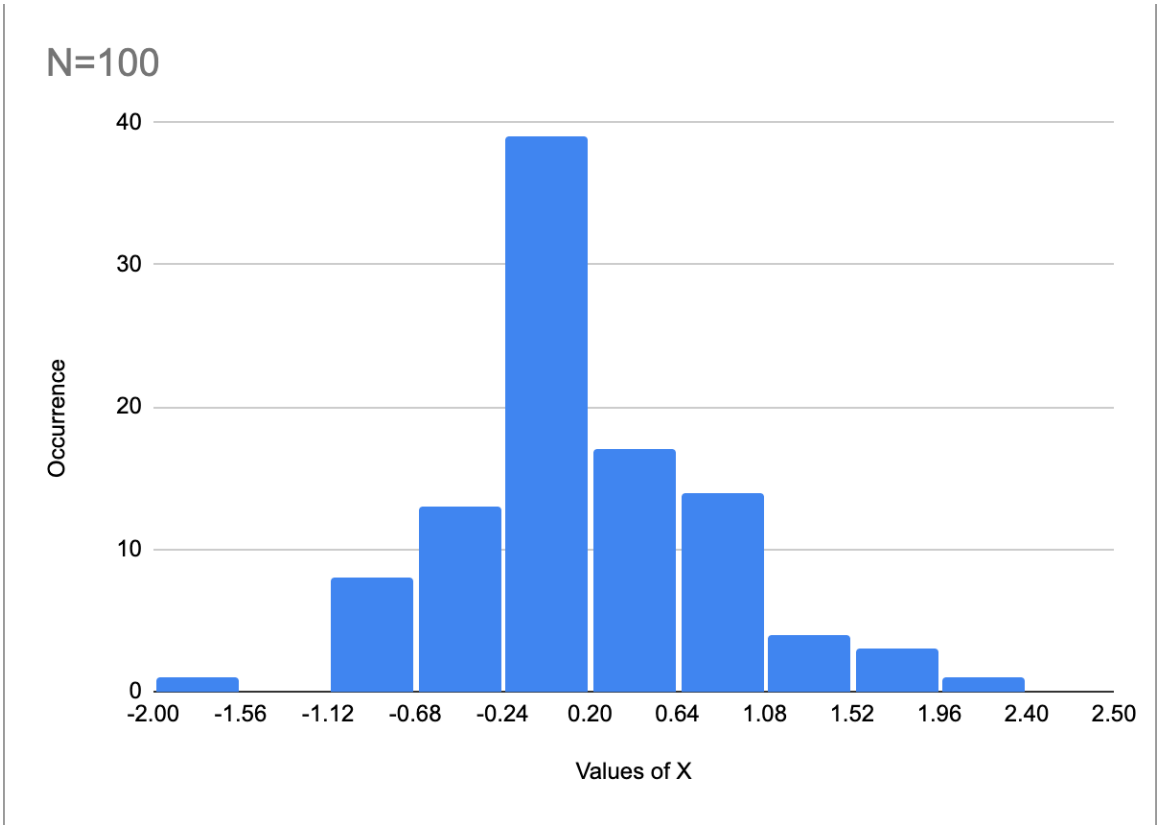

Figure 5: Histogram for N=100

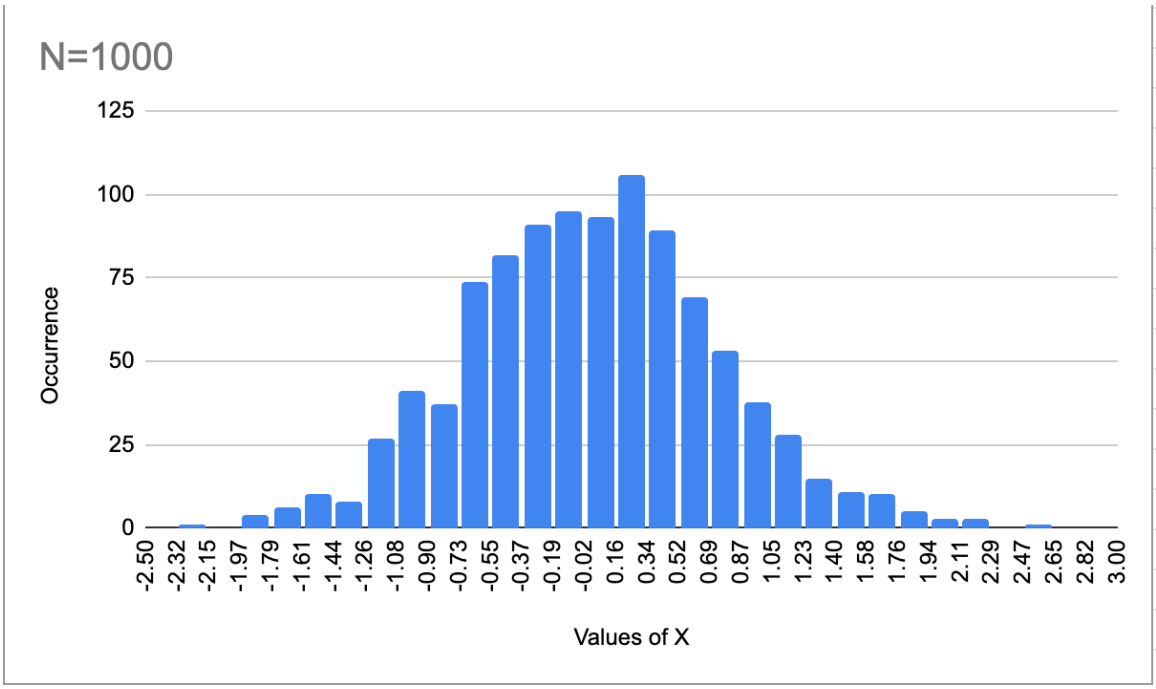

Figure 6: Histogram for N=1000

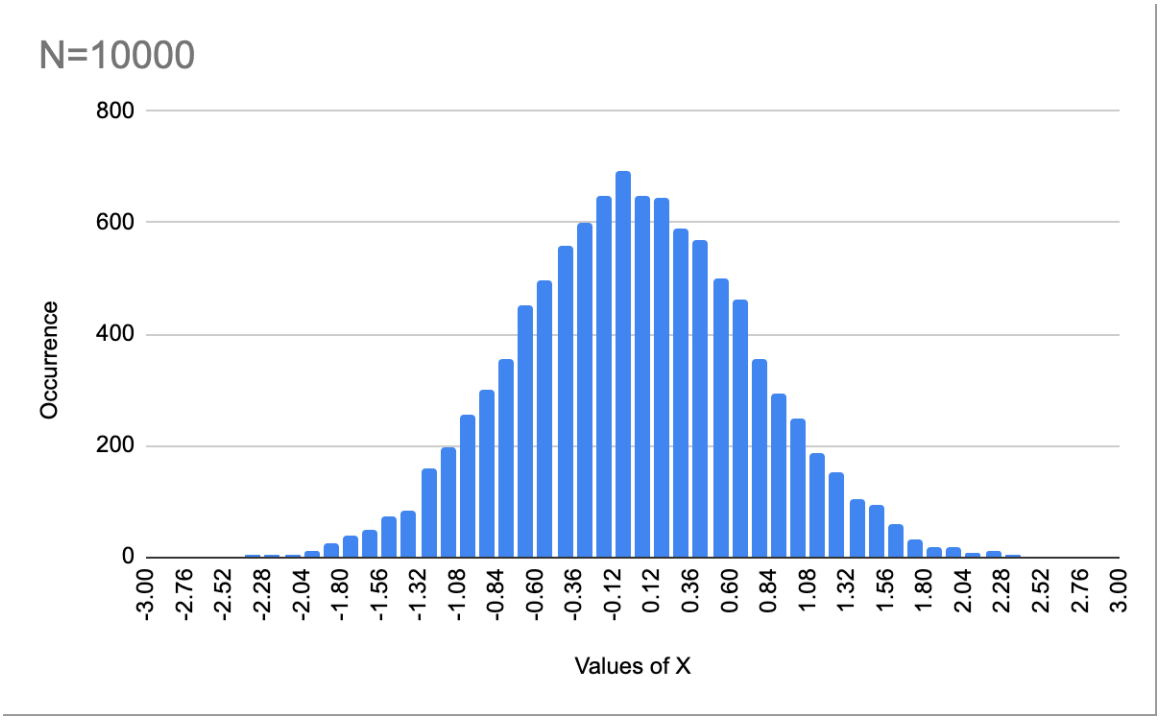

Figure 7: Histogram for N=10000

(b) (i) The Empirical Distribution Functions(EDF) of X are plotted below.

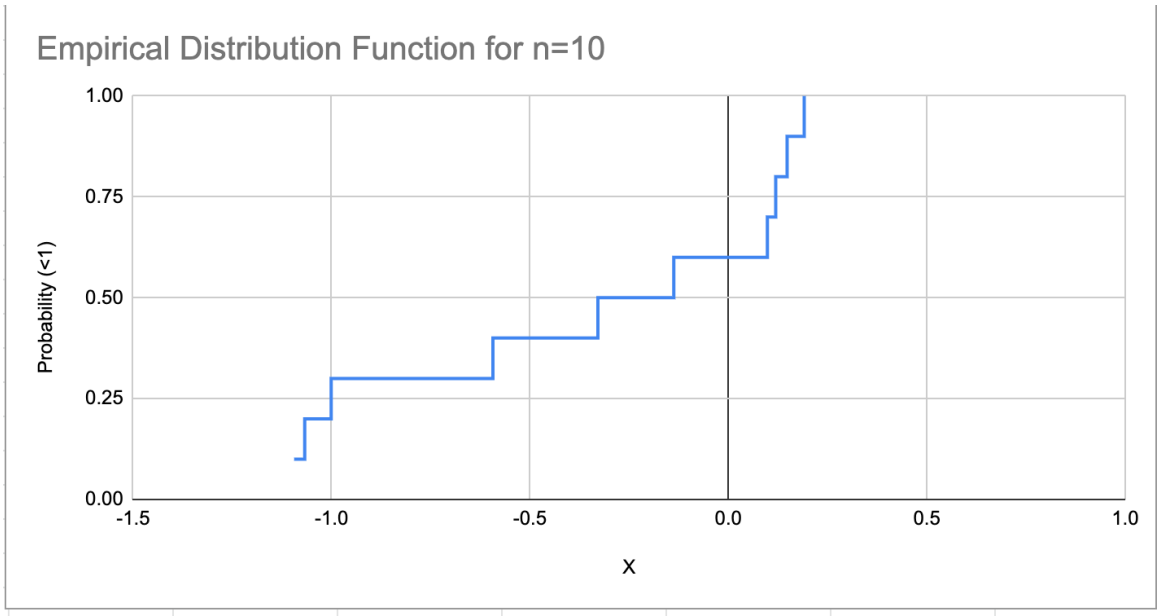

Figure 8: EDF for N=10

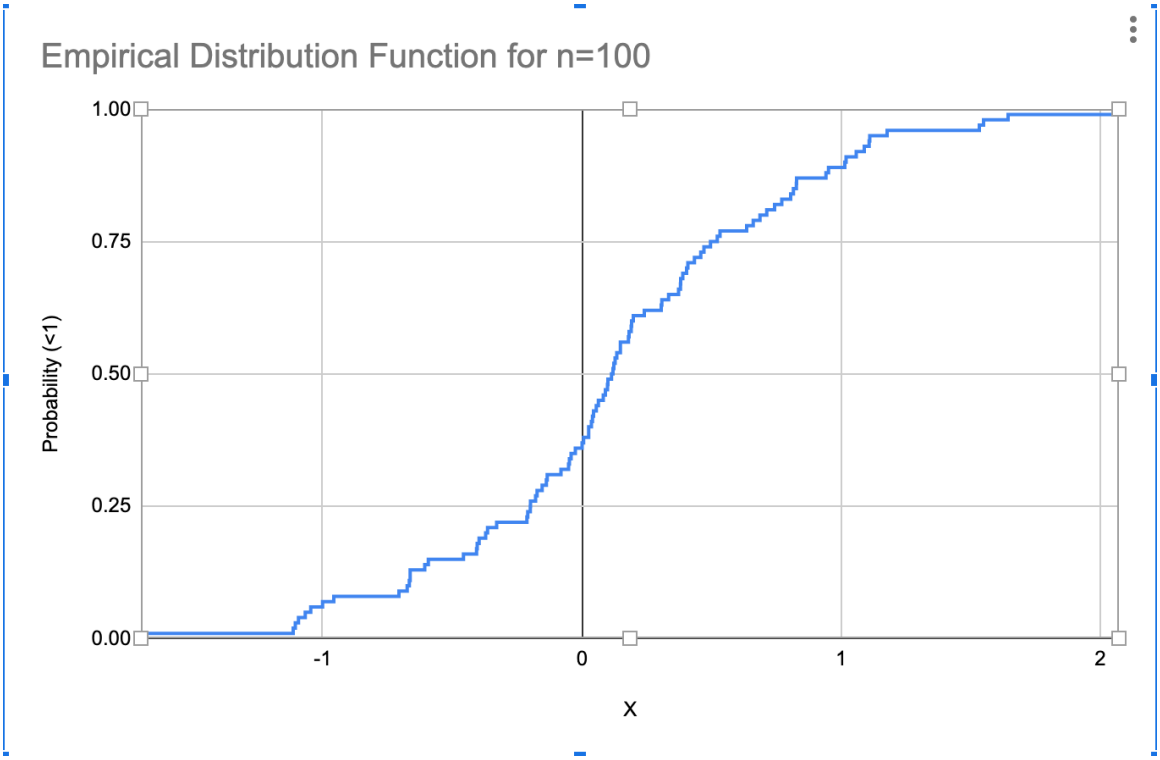

Figure 9: EDF for N=100

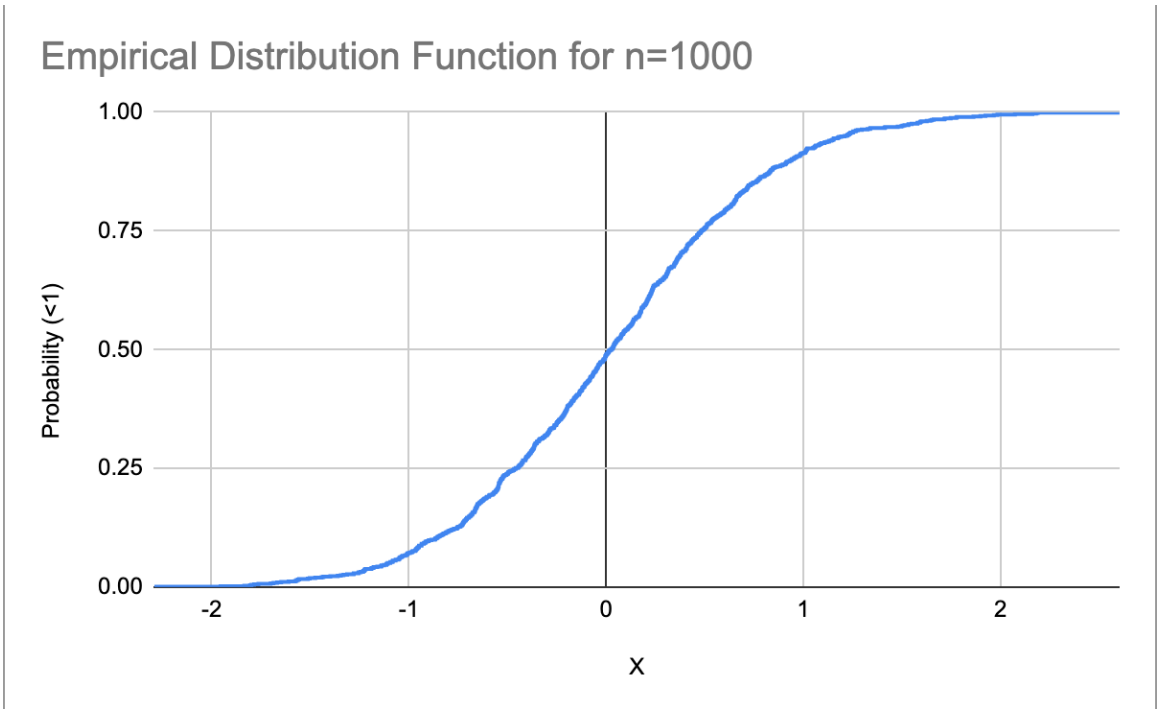

Figure 10: EDF for N=1000

(b) (ii) Erf for N=10000:

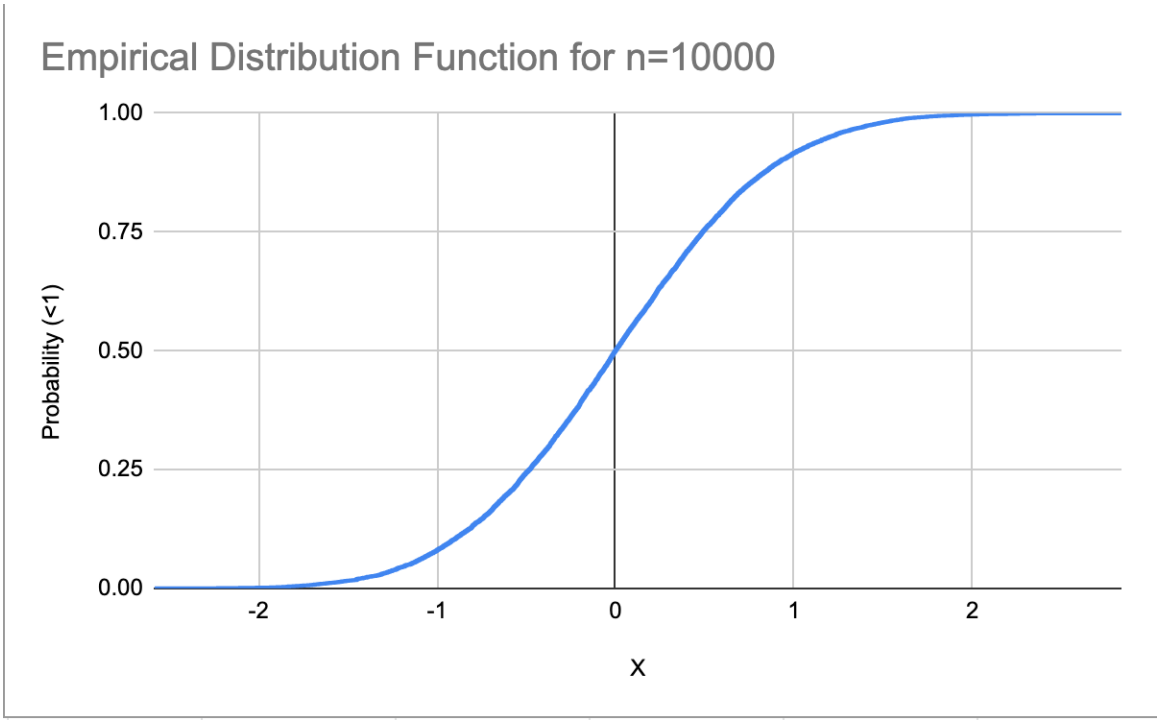

Figure 11: EDF for N=10000

Table 2: Error function tabulation

X	ErF	EDF
1	ErF(1)=EDF(1)	0.916
2	ErF(2)=EDF(2)	0.9973

2.4 Inferences

- As the number of samples is increased from 100 to 10000, the histogram obtained is more and more like a Gaussian distribution. i.e. the bell curve becomes more prominent.
- As the number of samples is increased from 100 to 10000, the EDF plot becomes closer to the ogive or the Cumulative Distribution Function (CDF). For N=10000, it is safe to say that both the curves coincide.
- The peak in the histograms is observed approximately near the midpoint which shows that maximum values tend to be in the mean range.
- Clearly, From two separate uniform random variables, we frame a standard exponential random variable and suitably alter them to obtain two independent zero-mean unit variance Gaussian random variables (or standard Normal random variables).
- For better clarity, the X range is chosen as [-2,2]. This encloses all the values of X.
- A note on Erf plotting: The Erf is precisely the scaled cum shifted graph of EDF. We know that the Erf is defined for Gaussian distributions. For N=10,000, the distribution is nearly-Gaussian. Hence the approximation EDF = Erf holds good.
- Since the exact value of Erf(1) is not available, we choose the value just greater than it and proceed.

2.5 Code

The code used for the experiments is mentioned in Listing 3.

```
// C program to generate random numbers
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
#include<math.h>


// Driver program
int main(void)
{   float U1,U2,E,X,Y; //defining variables

    srand(0);   //common seeding value

    for(int i = 0; i<pow(10,4); i++){
        U1=(float)  rand()/(RAND_MAX);  //generating random numbers in
         ↪  (0,1)
```

```
        U2=(float) rand()/(RAND_MAX);
        E=log(1/U1); //using the uniform random variable1
        X=pow(E,0.5)*cos(2*M_PI*U2); //using the uniform random variable2
        Y=pow(E,0.5)*sin(2*M_PI*U2);
        //printf(" %f \t\t %f\n ", x,y);

        printf("%f\n",X); //instead of opening a file, we copy the values
        ↪  from output to plot in the histogram and edf. this is pretty
        ↪  straightforward.
    }

    return 0;

}
```

Listing 3: Code snippet used in the experiment.

2.6 Contributions

Complete efforts were put in by me - including Coding, Algorithm, Plotting and Latex report preparation. I also confirmed the outputs with my teammates and we prepared individual reports.

2.7 Alternate Methods

I used google sheets for both histograms and EDF. Though google sheets software lacks a step-function plot, I printed all values twice and combined it with the column containing each of the $\frac{1}{n}$ values repeated twice. This resulted in a step graph which uses straight forward line graphing.
Other alternatives include gnuplotting as and when the output is generated. A noteworthy software for accomplishing exactly this and also for directly plotting the Cumulative Distribution Function is Num Excel. However, my graphing uses just elementary google sheets, which is a simpler alternative.

References

[1] Dianna Cowern, Derek Muller. Calculating Pi with Darts, 03 2015. URL https://www.youtube.com/watch?v=M34TO71SKGk.

[2] GeeksforGeeks. rand() and srand() in C/C++, 11 2020. URL https://www.geeksforgeeks.org/rand-and-srand-in-ccpp/.