EE1103: Numerical Methods

Programming Assignment # 2

Shreya .S. Ramanujam, EE21B126

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 generator 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 Figure 1

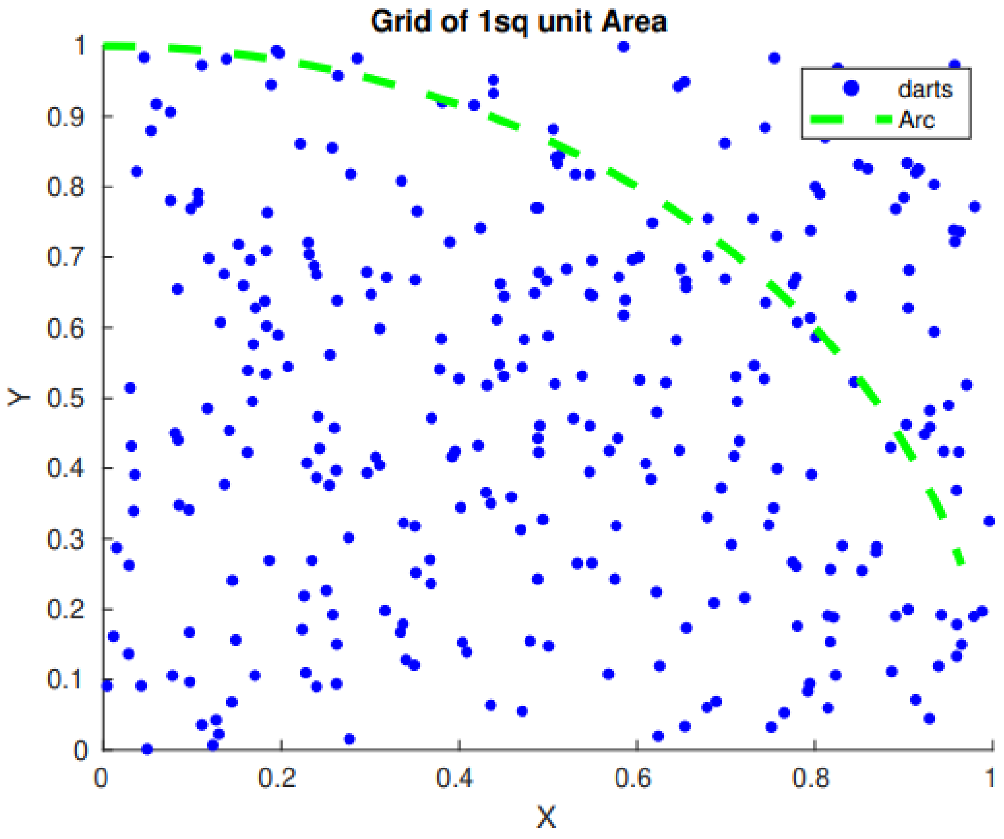

Figure 1: Points on the grid denoting landed darts ($n = 300$)

- 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)

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

1(b)

1. Compute the absolute error $\% = \frac{|\hat{\pi}_n - \pi|}{\pi} * 100$ and tabulate along with (a).
2. Plot the absolute error % as a function of n ($\log_{10} scale$).

1.1 Approach

In this problem, we use a nested `for` loop.

The outer `for` loop iterates over each value of "n" (which is the total no. of dart throws considered for that iteration).

In each outer loop iteration (thus "n" is fixed for this iteration), the inner `for` loop generates "n" number of random dart landing coordinates (x, y), and counts the number of these darts that land inside the circle $x^2 + y^2 < 1$.

We then apply

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

to get our estimate of π,

$$\text{absolute error\%} = \frac{|\hat{\pi}_n - \pi|}{\pi} * 100 \tag{5}$$

to calculate absolute error %, and then display tabulated result.

1.2 Algorithm

The pseudocode for estimating $\hat{\pi}_n$ and the absolute error % is provided in Algorithm 1.

Algorithm 1: Estimating the value of π ($\hat{\pi}_n$) by throwing darts

```
Set seed for random generator
for $n \leftarrow 100, 1000, \ldots 1000000000$ do
    $n_{arc} \leftarrow 0$;
    for $i \leftarrow 0, 1, \ldots (n-1)$ do
        $x \leftarrow rand()/(RAND\_MAX)$;
        $y \leftarrow rand()/(RAND\_MAX)$;
        if $x^2 + y^2 < 1$ then
        $n_{arc} \leftarrow n_{arc} + 1$;
        end
    end
    $\hat{\pi}_n \leftarrow 4 * \frac{n_{arc}}{n}$;
    $abserror \leftarrow \frac{|\hat{\pi}_n - \pi|}{\pi} * 100$;
    $display\ n,\ \hat{\pi}_n,\ abserror$;
end
```

1.3 Results

The results of Problem 1 have been tabulated below in Table 1.

Table 1: Results of Problem 1

No. of Dart Throws (n)	$log_{10}(n)$	$\hat{\pi}_n$	Absolute Error %
100	2	3.200000	1.859165
1000	3	3.160000	0.585927
10000	4	3.124000	0.559989
100000	5	3.148240	0.211594
1000000	6	3.143180	0.050523
10000000	7	3.141957	0.011591
100000000	8	3.141496	0.003086
1000000000	9	3.141528	0.002054

The plots of *"Estimated value of pi ($\hat{\pi}_n$) v/s No. of dart throws considered (log_{10} scale)"* and *"Absolute Error % v/s No. of dart throws considered (log_{10} scale)"* have also been shown below in Figure 2 and 3 respectively.

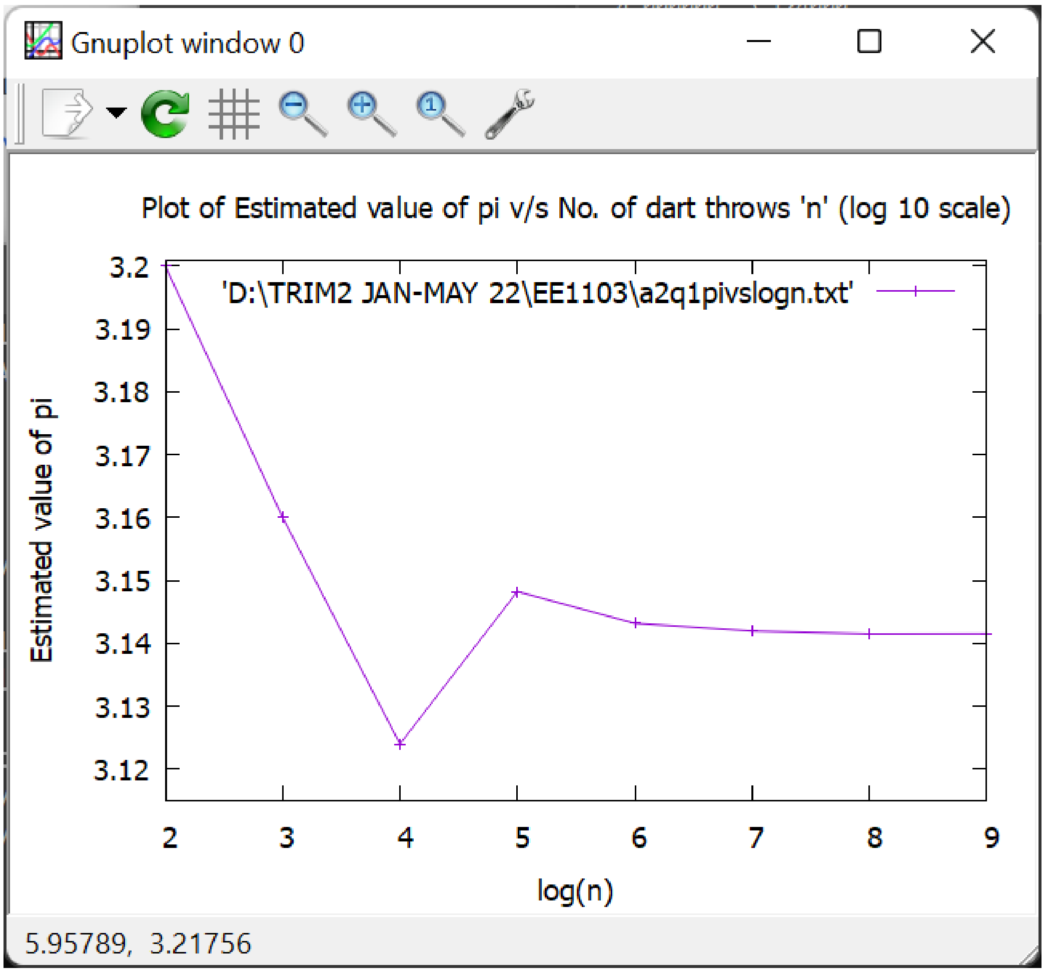

Figure 2: Estimated value of π ($\hat{\pi}_n$) v/s No. of dart throws considered ($\log_{10}$ scale).

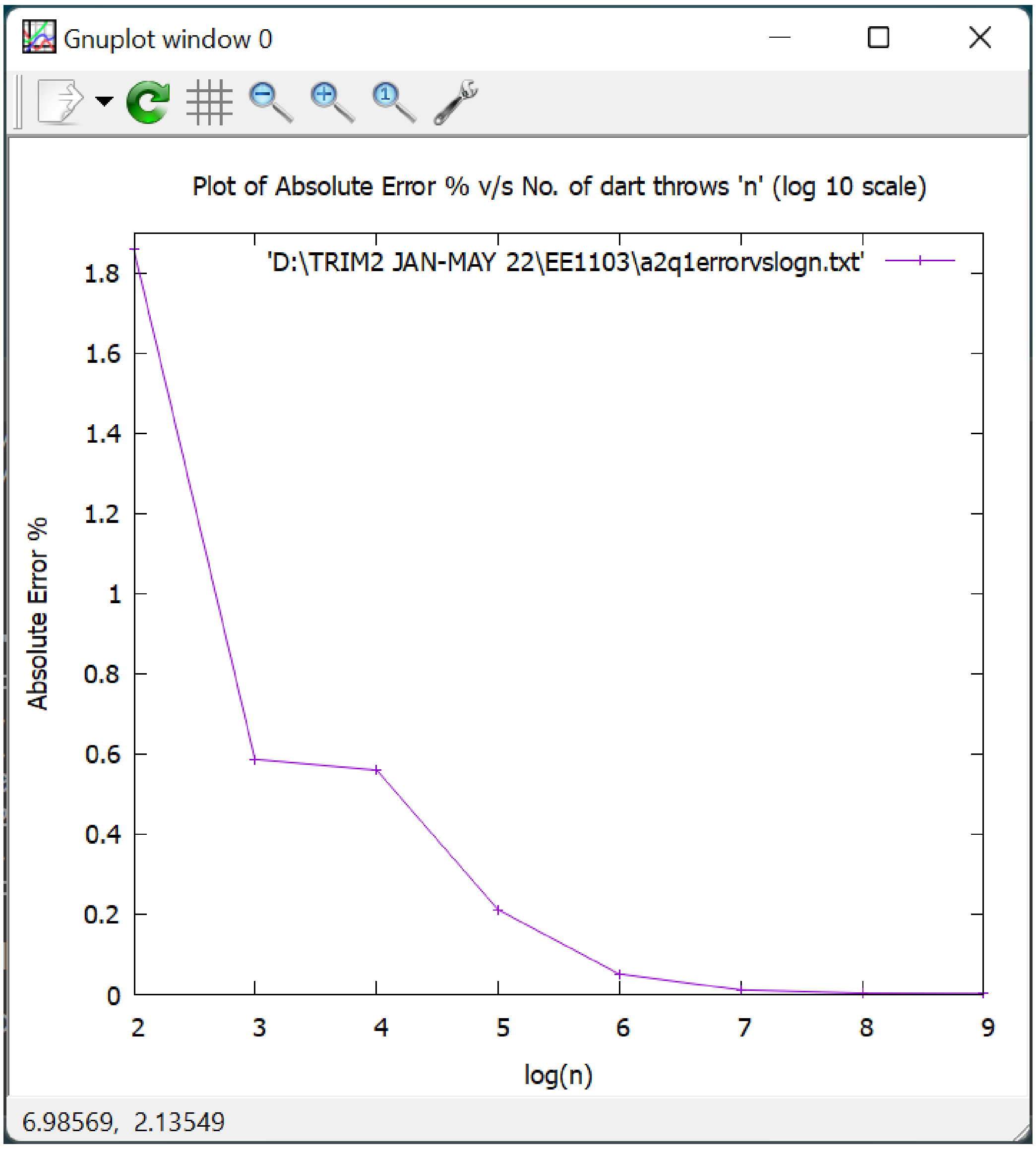

Figure 3: Absolute Error % v/s Number of dart throws considered while approximating π ($\log_{10}$ scale)

1.4 Inferences

We deduce the following inferences from Problem 1:

- There is a sharp decline in the absolute error % as a function of $log_{10}(n)$, as shown in Figure 3. So, the more sample dart throws we consider in our estimation, the closer our estimate of $\hat{\pi}_n$ will be to the real value of π.

- However, the maximum dart throws we can accommodate is 10^9, since beyond this (i.e. for $10^{10}, 10^{11} \dots$ dart throws) we get an error message of *"warning: overflow in implicit constant conversion [-Woverflow]"*.

- This error message disappears and the program compiles if we take n as *long long* instead *int*, but then the program takes an extremely long time to execute for $n = 10^{10}$ itself (I was not able to get an output at all; the program got "KILLED" after 10 minutes of executing with no output).

- Thus, taking these constraints into consideration, the most accurate estimate of $\hat{\pi}_n$ that we can get is ***3.141528* for 10^9 dart throws, with an absolute error of *0.002054%***.
- Using *double* variables instead of *float* variables in the program gives the same value for most accurate estimate of $\hat{\pi}_n$ that we got using *float*, namely $\hat{\pi}_n$ = ***3.141528* for 10^9 dart throws**.
- Thus, there is no real benefit of using *double* over *float* (in terms of getting a more accurate value of $\hat{\pi}_n$), so I have used *float* in my program to save memory space.

1.5 Code

The code used for Problem 1 is mentioned in Listing 1.

Listing 1: Code snippet used in Problem 1.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main (void)
{
    //declaring necessary variables
    int j, n;
    float x , y, pihat, abserror, xaxis;

    //setting the seed for rand()
    srand(3141592653);

    printf("Number of dart throws \t\t log(n)\t\t\tEstimate for pi
    ↪  \tAbsolute Error %% \n");

    //outer loop for iterating over number of dart throws considered in
    ↪  each case
    for(n = 100; n <= 1000000000 ; n = n * 10)
    {
        //initializing counter variable "j", which counts no. of darts
        ↪  inside the circle, to zero
        j = 0;

        //inner loop for generating random coordinates of the dart and
        ↪  counting the no. of darts that land inside the circle
        for (int i = 0; i < n ; i++)
        {
            x = (float)rand()/(RAND_MAX);
            y = (float)rand()/(RAND_MAX);

            //checking if dart landed inside circle and incrementing
            ↪  counter j for each dart that lands inside circle
            if ((x*x + y *y) < 1)
            j++;
        }

        //calculating our estimate of pi
        pihat = 4.0 * ((float)j/n);

        //calculating absolute error % in estimated value of pi
        abserror = fabs(((M_PI - pihat)/M_PI)*100);

        //calculating x coordinate for plotting (ie log(n))
        xaxis = log10f(n);

        //tabulate result for each value of n (no. of darts)
        printf("%d\t\t\t\t%f\t\t%f\t\t%f\n", n, xaxis, pihat, abserror);
    }

    return(0);
}
```

1.6 Contributions

Individual submission. All items included in this subsection were done by me.

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

$$E = -\log(U_1). \tag{6}$$

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

$$X = \sqrt{E}\cos(2\pi U_2) \tag{7}$$

$$Y = \sqrt{E}\sin(2\pi U_2) \tag{8}$$

- 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. (onlinegdb)

(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

In this problem, we use a nested `for` loop to generate required number of samples of X.

The outer loop iterates over $n = 100, 1000, 10000$, with "n" representing the total number of random samples we need to generate in that iteration of the outer loop.

The inner loop then generates and displays the specified "n" number of random samples.

2.2 Algorithm

The pseudocode for generating Gaussian random variable X is provided in Algorithm 2.

Algorithm 2: Generating samples of Gaussian random variable X

```
Set seed for random generator
for $n \leftarrow 100, 1000, 10000$ do
    for $i \leftarrow 1, 2, \ldots n$ do
        $u1 \leftarrow rand()/(RAND\_MAX)$;
        $u2 \leftarrow rand()/(RAND\_MAX)$;
        $e \leftarrow -\log_{10}(u1)$;
        $x \leftarrow \sqrt{E} * \cos(2 * \pi * u_2)$;
        display $x$
    end
end
```

2.3 Results

The histograms for the $n = 100, 1000$ and 10000 samples of Gaussian Random Variable X have been plotted below in Figures 4, 5 and 6 respectively.

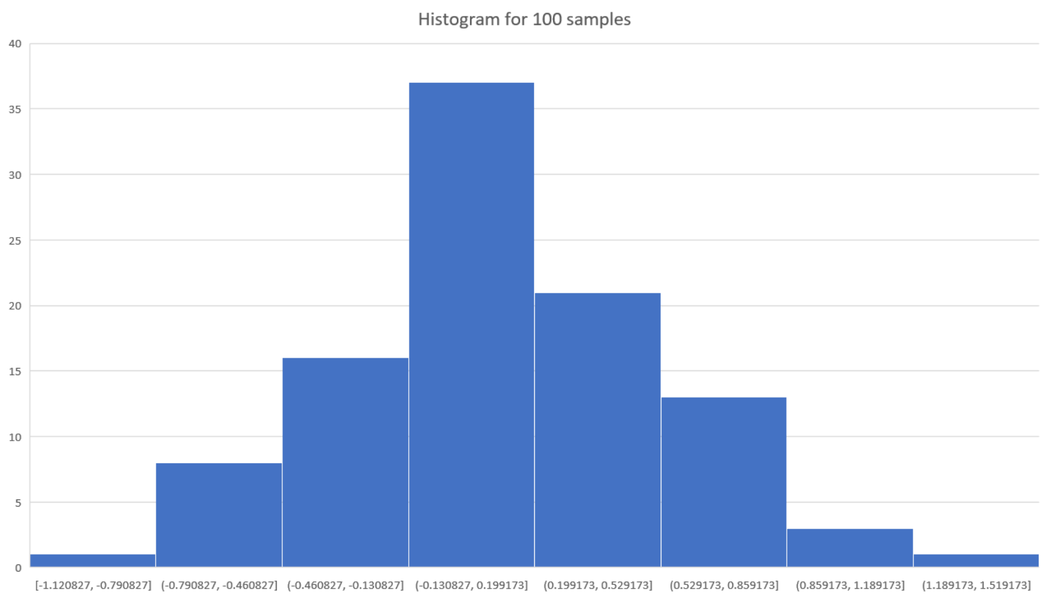

Figure 4: Histogram for 100 samples of Gaussian random variable X

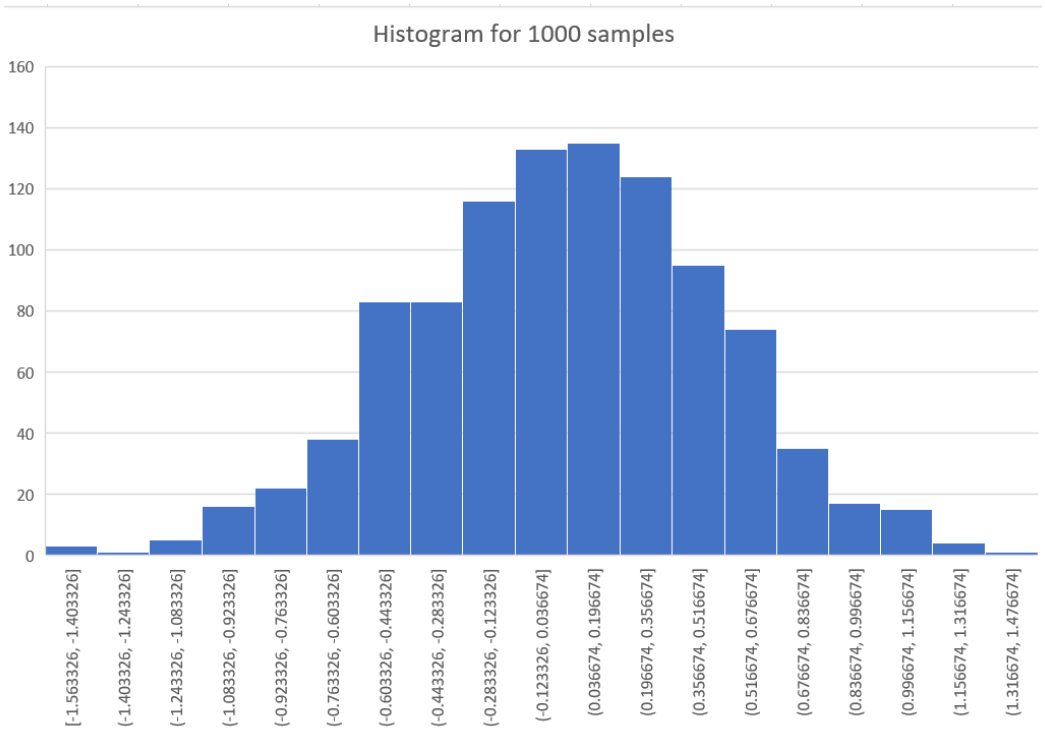

Figure 5: Histogram for 1000 samples of Gaussian random variable X

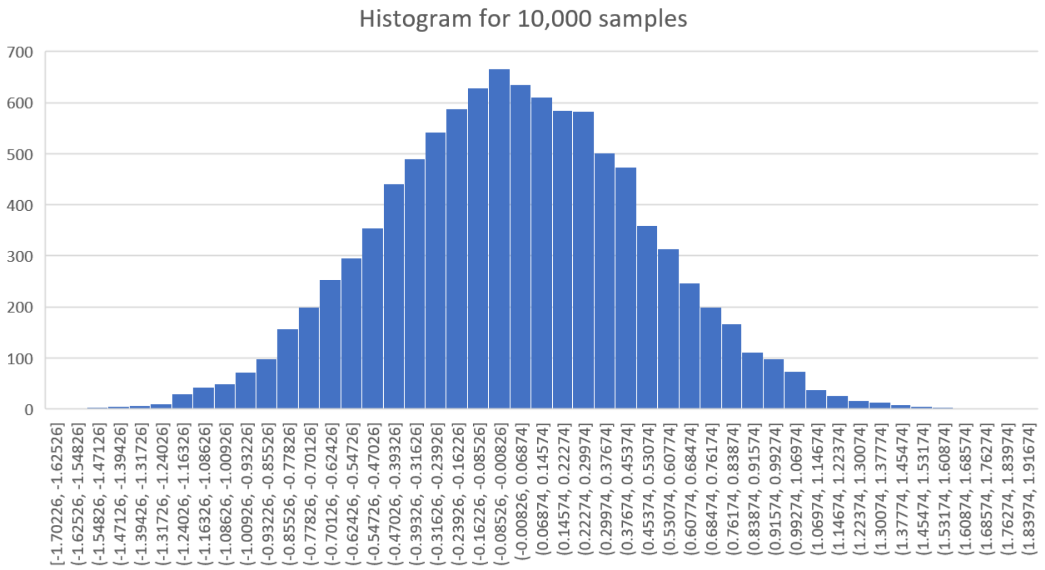

Figure 6: Histogram for 10000 samples of Gaussian random variable X

The Empirical Distribution Functions (EDF) of X for $n = 10, 100$ and 1000 have also been plotted below in Figures 7, 8 and 9 respectively.

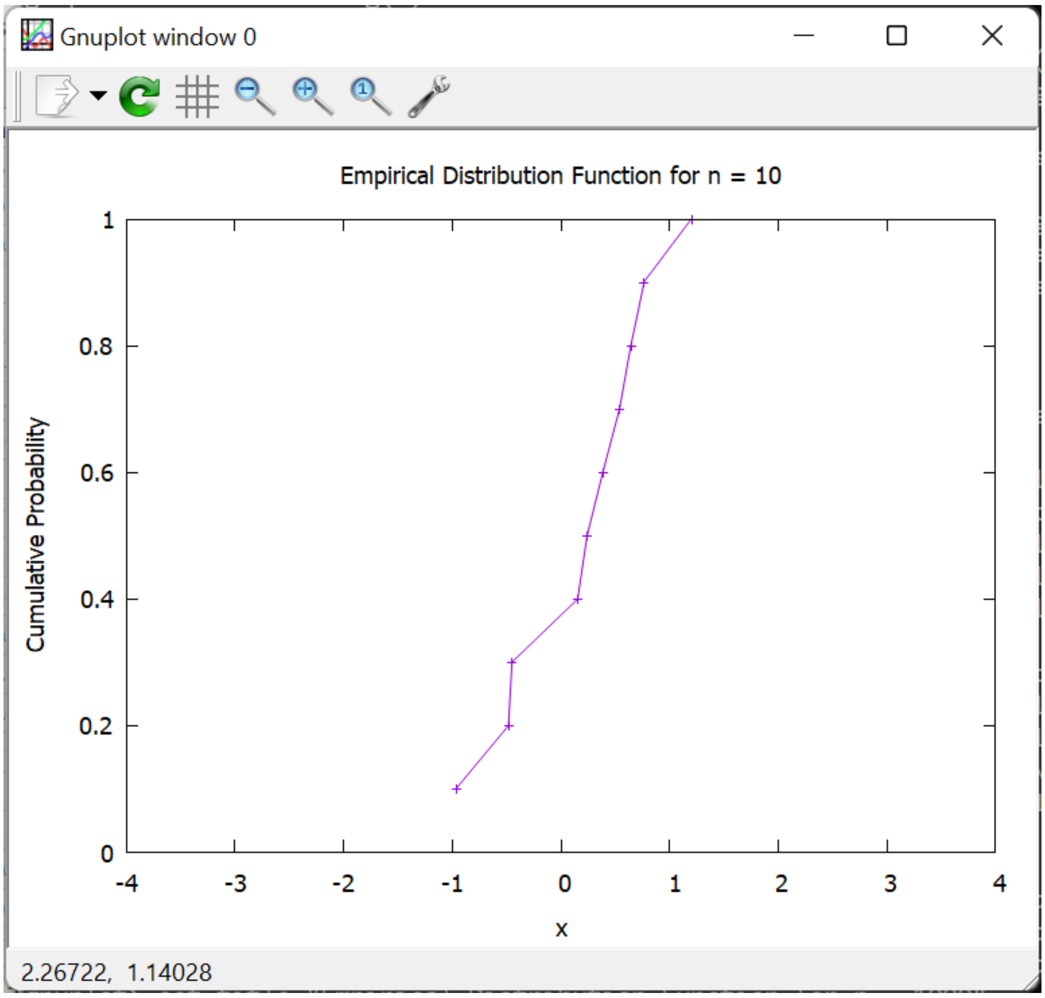

Figure 7: Empirical Distribution Function (EDF) of Gaussian random variable X for $n = 10$ samples

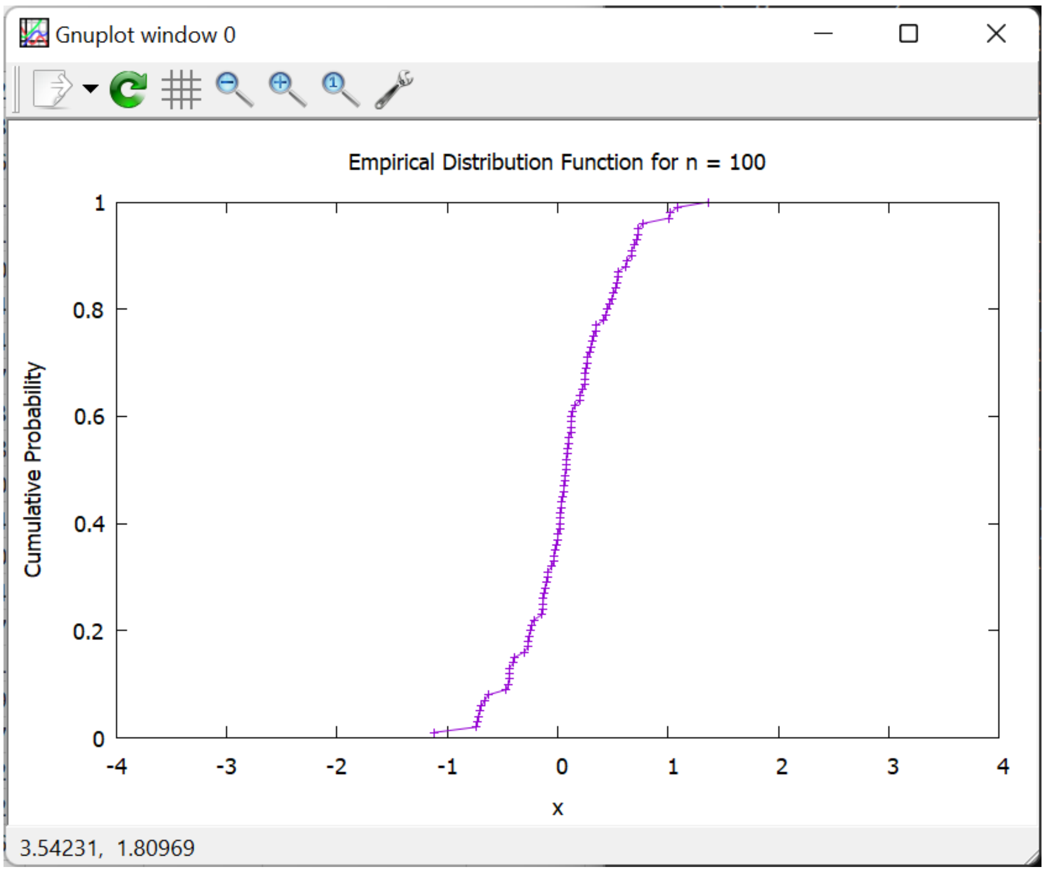

Figure 8: Empirical Distribution Function (EDF) of Gaussian random variable X for $n = 100$ samples

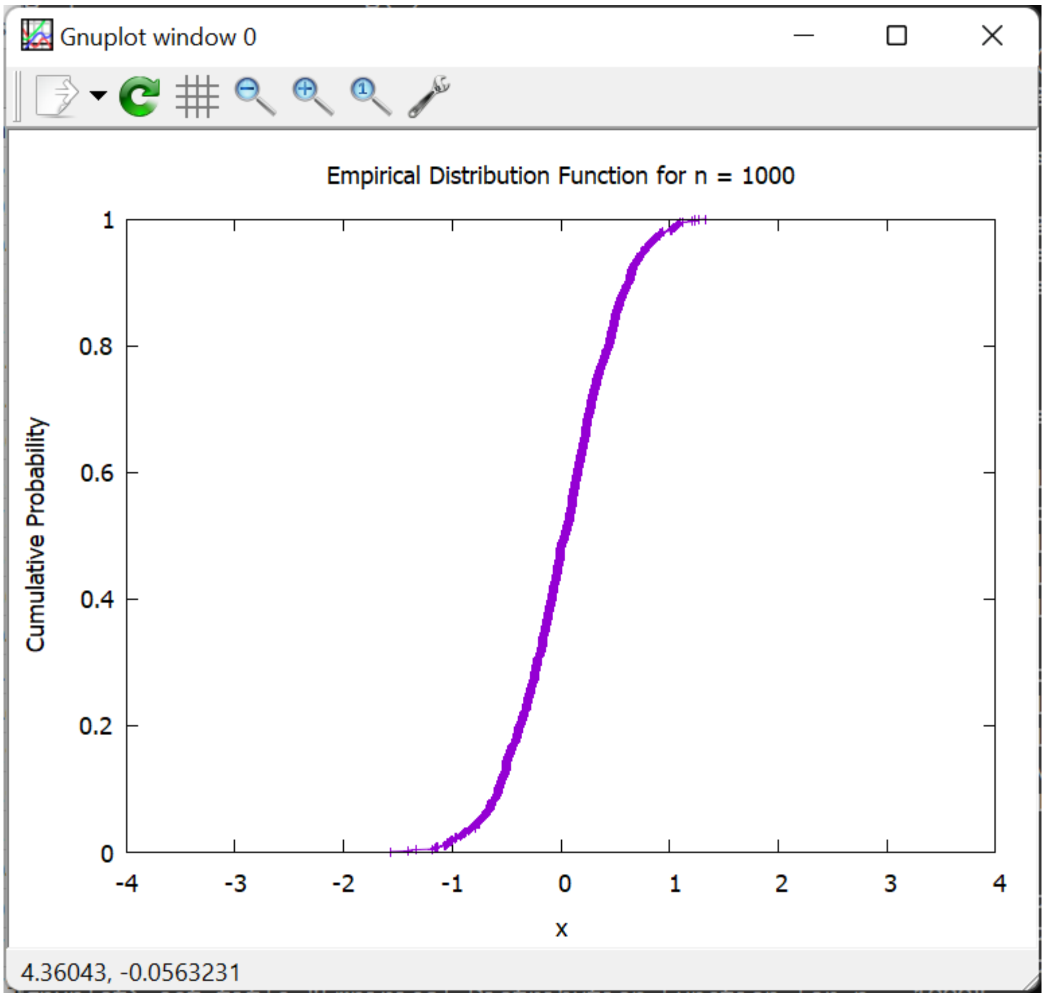

Figure 9: Empirical Distribution Function (EDF) of Gaussian random variable X for $n = 1000$ samples

2.4 Inferences

We deduce the following inferences from Problem 2:

- *The Empirical Estimates for Erf(1) and Erf(2) for* $n = 10000$ *samples come out to be* $\frac{9819}{10000}$ *and* $\frac{10000}{10000}$ *respectively.*
- This implies that out of the 10,000 random samples of X generated, 9819 of them had a value ≤ 1, and all 10,000 of them had a value ≤ 2.
- The histograms generated for Gaussian random variable X are approximately Gaussian-curves (with zero-mean), and their shape becomes more pronounced as the no. of samples (n) is increased, as can be seen from Figures 4, 5 and 6 (for $n = 100, 1000$ and 10000 respectively), where the shape of the histogram progressively becomes more bell-shaped.
- Similarly, the shape of the EDF plot also becomes much more pronounced and smooth curve-like as we increase the no. of samples, n, as seen from Figures 7, 8 and 9 for $n = 10, 100$ and 1000 respectively.

2.5 Code

The code used for the experiments is mentioned in Listing 2.

Listing 2: Code snippet used in Problem 2: Generating samples of Gaussian random variable X.

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

int main (void)
{
    //setting seed for rand() function
    srand(0);

    //declaring variables used in calculations
    float u1, u2, e, x;

    //control variables of outer and inner loops respectively.
    int n, i;

    // n specifies total no. of samples of x to be generated
    for(n = 100; n <= 10000; n = n * 10)
    {
        printf("Samples of X generated for n = %d\n", n);

        //inner loop generates "n" samples of x
        for(i = 1; i <= n; i++)
        {
            //calculations for generaing a sample of Gaussian random
             ↪  variable x
            u1 = (float)rand()/RAND_MAX;
            u2 = (float)rand()/RAND_MAX;
            e = -log10f(u1);
            x = powf(e, 0.5) * cosf(2 * M_PI * u2);

            //printing each generated sample
            printf("%f\n", x);
        }

        //formatting output
        printf("\n");
    }
    return(0);
}

```

2.6 Contributions

Individual submission. All items included in this subsection were done by me.