EE1103: Numerical Methods

Programming Assignment # 4

Lakshmiram S, EE22B117

December 12, 2022

Contents

1 Problem 1

Estimating π using darts :
Here, we try to use probability to determine the value of π. this can be achieved as follows: We randomly throw darts on a square board and calculate the number of those darts that lie inside a circle whose diameter is the same as that of the square's side.
Now π is evaluated as follows:

$$\frac{number\ of\ darts\ inside\ circle}{total\ number\ of\ darts} = \frac{\pi}{4}$$

(a) Estimate the value of π using the above process. Find out the maximum value of n for which the process can be carried out without precision errors.Also plot π_n vs log(n).

(b) Plot the absolute error vs log(n).

(c) Plot a histogram of the quantity $\sqrt{n}|(\pi_n - \pi)|$ and comment on the shape you observe.

1.1 Approach

We simulate the dart throwing process by using a random number generator and later we calculate the probability.
This can be done by using simple loops.

1.2 Algorithm

Algorithm 1: throwing darts to calculate π.

```
//function to generate a random float number between two numbers a and b
num=a+(b-a)*(random number/max limit)
//generate a random x and y coordinate
x=random number between 0 and 1
y=random number between 0 and 1
//condition
if x^2+y^2<1
    points inside circle+
//probability
\pi=(points inside circle)/(total points)*4
```

1.3 Code

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

//function for generating a random number between a and b both inclusive
float random_number(float a,float b)
{
    return a+(b-a)*((float)rand()/(float)RAND_MAX);
}
//function for pi_n
float pi_n(float n)
```

```
{
    float inside_circle=0;
    for(int i=0;i<n;i++)
    {
        float x=random_number(0.00,1.00);
        float y=random_number(0.00,1.00);
        if(pow(x,2)+pow(y,2)<=1)
        {
            inside_circle++;
        }
    }
    return (inside_circle/n)*4;
}
//function to return the madhava series
float madhava_n(float n)
{
    float sum=0;
    for(int i=1;i<=n;i++)
    {
        sum+=pow(-1,i+1)/(2*i-1);
    }
    return sum*4;
}
int main()
{
    srand(200);
    float pi = M_PI;
    for(int i=1;i<8;i++)
    {
        float approx_pi=pi_n(pow(10,i));
        float absolute_error=fabs((approx_pi-pi)/pi)*100;
        float
↪   absolute_error_madhava=fabs((madhava_n(pow(10,i))-pi)/pi)*100;
        //printf("approximate pi : %f absolute error :
↪   %f\n",approx_pi,absolute_error);
        printf("%f\n",absolute_error_madhava);


    }
}
```

Listing 1: Code snippet used in the experiment.

1.4 Results

The output obtained has been compiled below as tables and graphs.

value of n	approximate value of pi	absolute error
10^1	**2.8**	**10.873236**
10^2	**3.16**	**0.585924**
10^3	**3.184**	**1.349865**
10^4	**3.1436**	**0.063893**
10^5	**3.14372**	**0.06771**
10^6	**3.140364**	**0.039114**
10^7	**3.140386**	**0.038424**
10^8	**0.671089**	**78.63858**

Figure 1: approximate values of pi and absolute errors

1.5 Inferences

- π can be approximated well upto 10^7 darts with good accuracy. beyond that precision errors fall in.
- As for the histogram, one can expect a uniform normal distribution. But we see this this is not the case. We can't exactly apply the central limit theorem here because it applies only when the number of observations per sample and the number of sample sets both tend to ∞.
- The absolute error in the Madhava series converges faster than the one using darts. Even if dart throwing can be simulated in a computer, it still requires the compiler to generate n random numbers and later check some conditions, which on the whole is more complicated than calculating the sum of a series upto n terms.

1.6 Contributions

I got to work on this assignment independently. But I did get timely help from my team mater Hrushikesh.

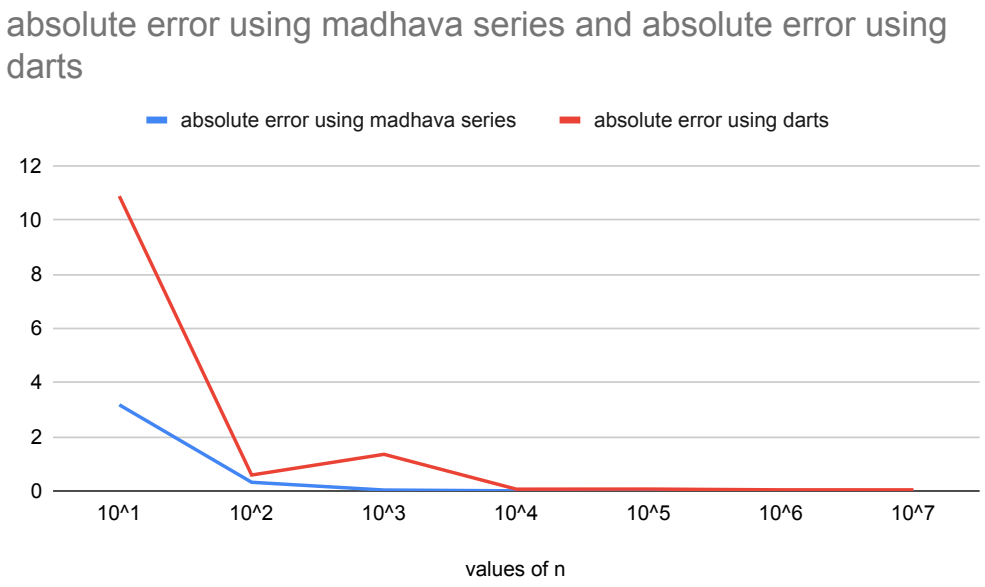

Figure 2

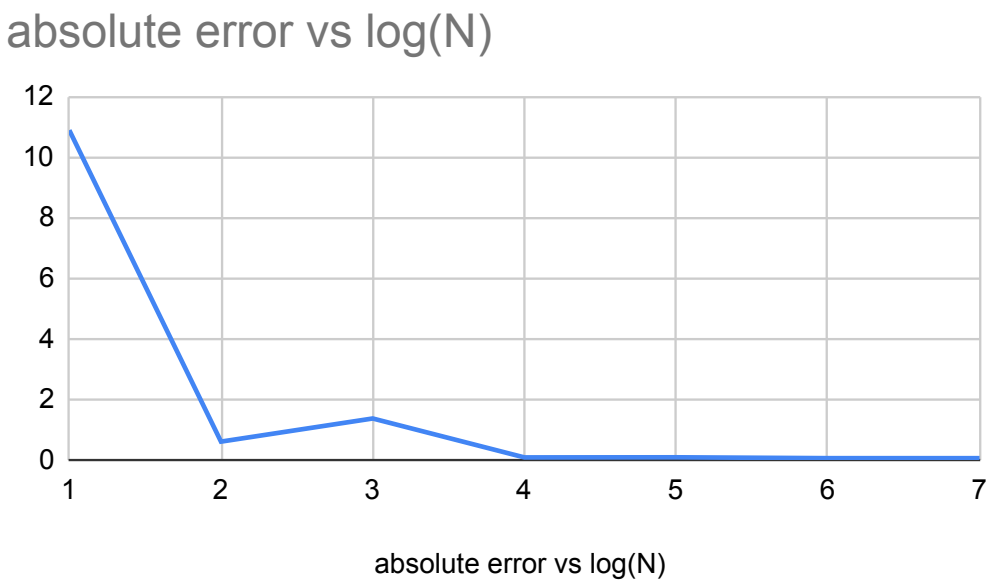

Figure 3

2 Problem 2

Generating independent zero-mean unit variance Gaussian random variables (or standard Normal random variables). :
Here we first produce two sets of n random numbers(call them U_1 and U_2. Now we find an exponential random variable using U_1 by

$$E = -2log(U_1)$$

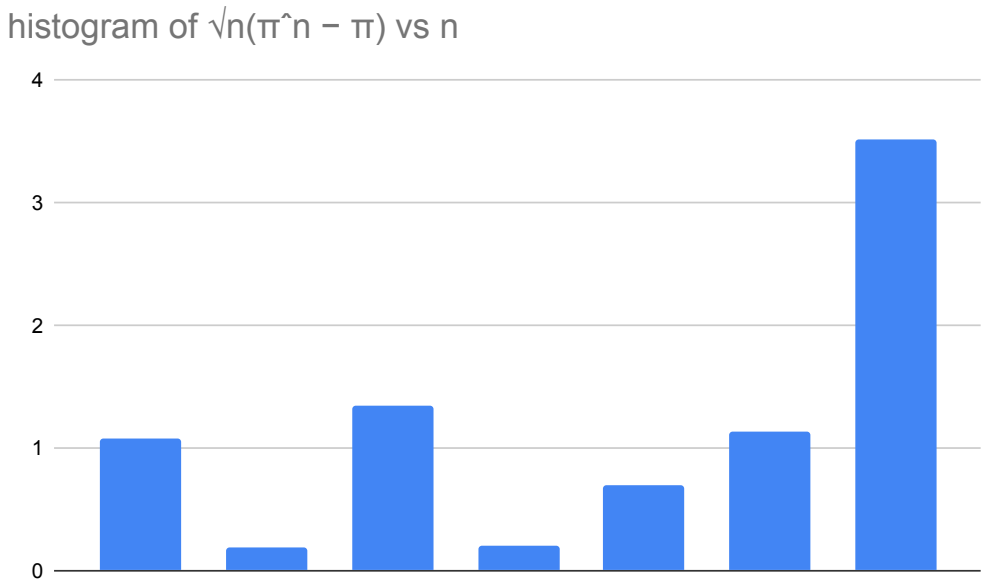

Figure 4

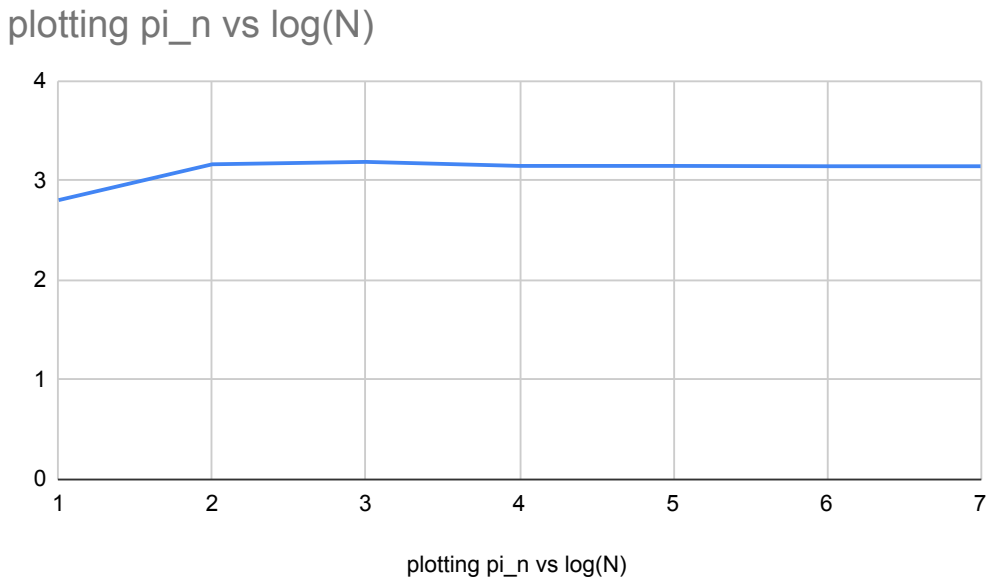

Figure 5

. The next step is to employ the Box-Muller transform to generate Normal random variables as follows:

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

and

$$X = \sqrt{E}\sin(2\pi U_2)$$

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

(b) Plot the absolute error vs log(n).

(c) 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.

(d) Compute Monte-Carlo estimates 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.

2.1 Approach

We use the same approach as before-use a random number generator to find X This,again,can be done by using simple loops.

2.2 Algorithm

Algorithm 2: Normal Random Variables

```
//function to generate a random float number between two numbers a and b
num=a+(b-a)*(random number/max limit)
Ei,U1_i and U2_i are the ith elements of the corresponding RV sample sets.
//find the exponential RV using the formula
Ei = 2  log(U1_i)
//Find X
X = \sqrt{E} cos(2U2_i)
print X
```

2.3 Code

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
//function to generate n random variables between 0 and 1
float random_number(float a,float b)
{
    return a+(b-a)*((float)rand()/(float)RAND_MAX);
}
//function to print the n values of the set X
//let u1_i denote the ith element of the set U1
//let u2_i denote the ith element of the set U2
void X(int n)
{
    for(int i=0;i<n;i++)
    {
        float u1=random_number(0.00, 1.00);
        float u2=random_number(0.00, 1.00);
        printf("%f\n",sqrt(fabs(-2*log(u1)))*cos(2*M_PI*u2));

    }
}
```

```
22
23
24 //planning on directly evaluating x_i using the formulas in hand
25 int main()
26 {
27     srand(0);
28     X(10000);
29 }
```

Listing 2: Code snippet used in the experiment.

2.4 Results

The output obtained has been compiled below as tables and graphs.

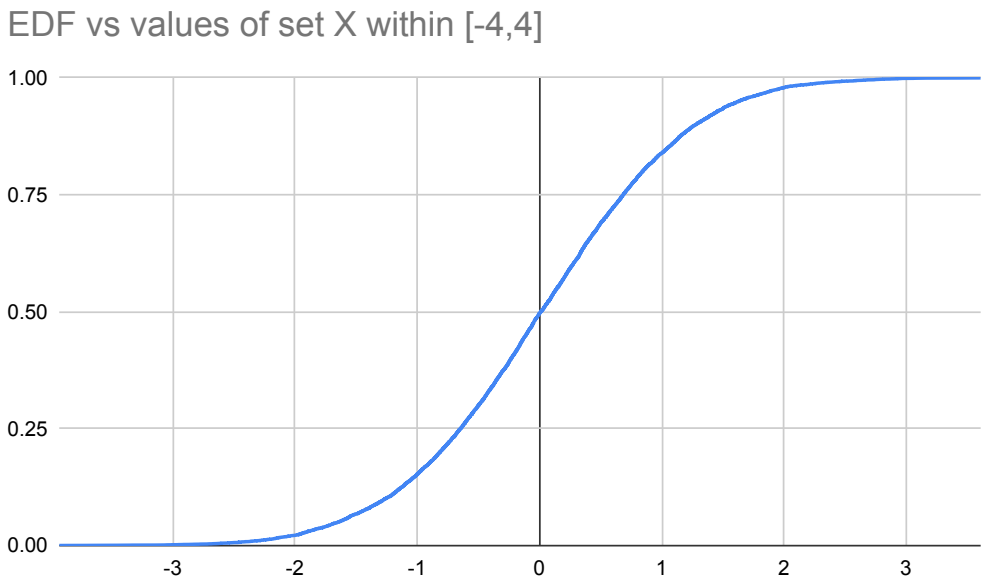

Figure 6: approximate values of pi and absolute errors

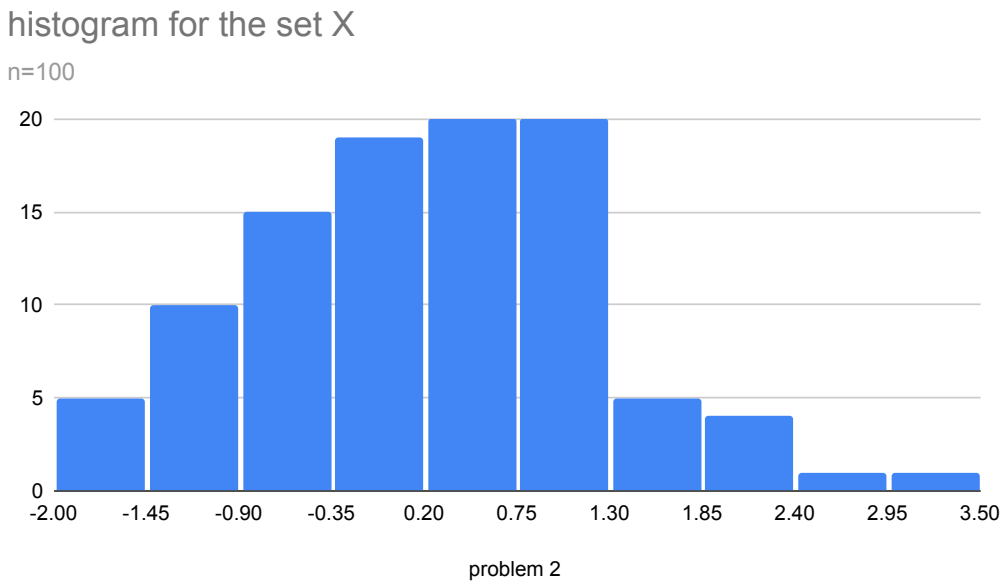

Figure 7

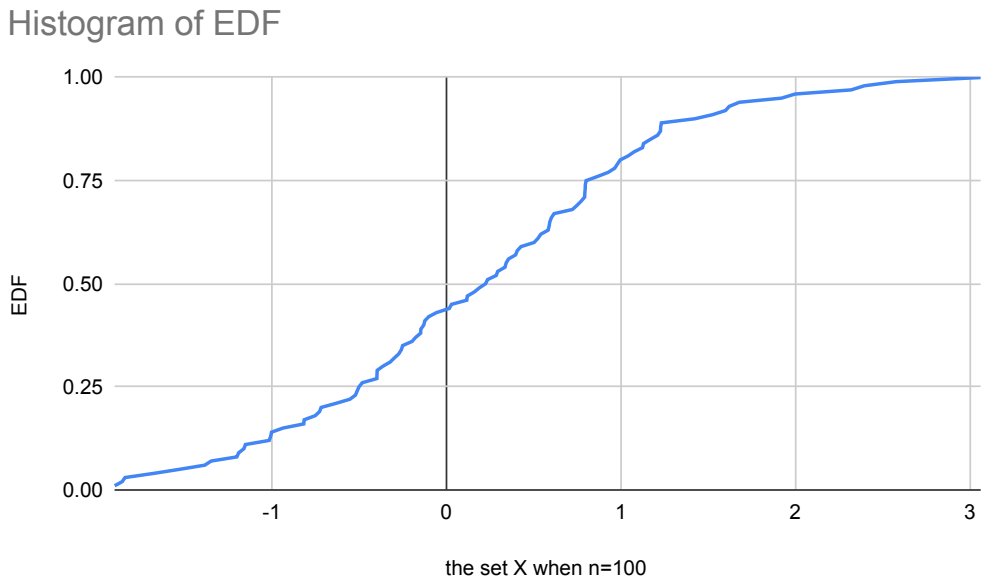

Figure 8

2.5 Inferences

- The histograms for n become more and more approximate to a bell curve as the value of n increases.
- The EDF values show that the fraction of numbers that are less than 1 is 0.849 and that of 2 is 0.9803, which is pretty close to the values obtained by numerically integrating the Gaussian PDF.

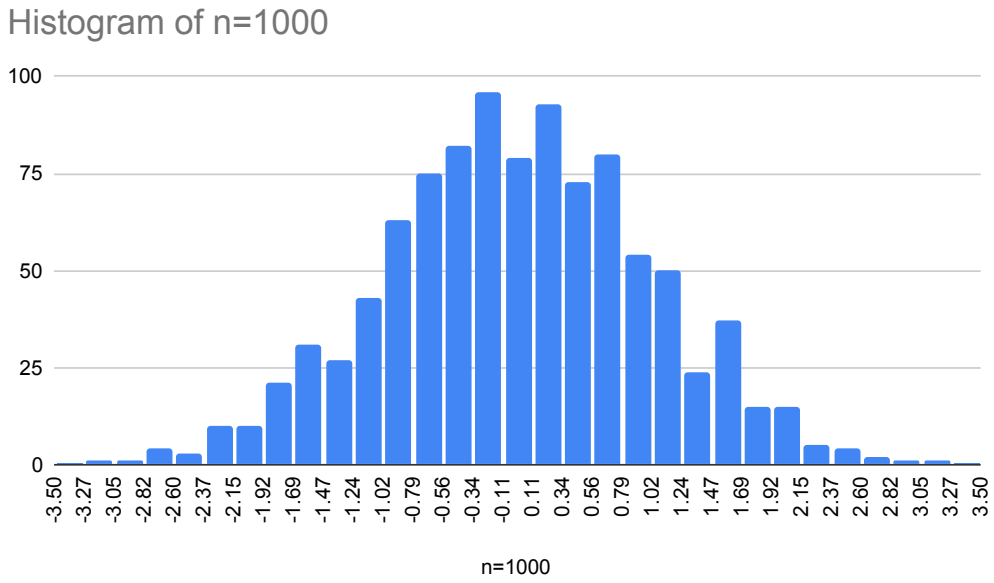

Figure 9

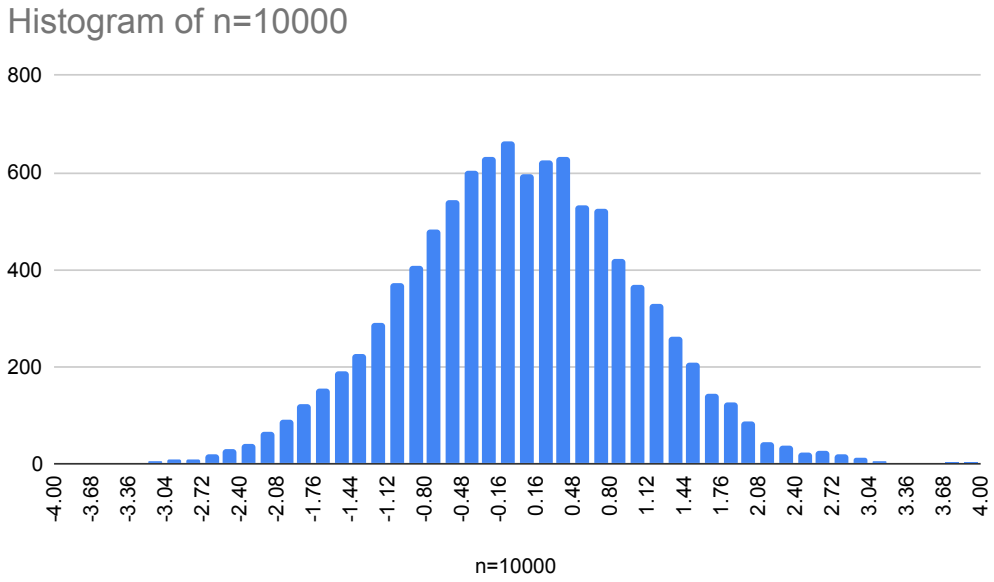

Figure 10

2.6 Contributions

I got to work on this assignment independently. But I did get timely help from my team-mate Hrushikesh.