EE1103: Numerical Methods

Programming Assignment # 3

Amizhthni PRK, EE21B015
Collaborators:
Ankita Harsha Murthy, EE21B020
Anirudh BS, EE21B019

March 6, 2022

Contents

List of Figures

List of Tables

1 Problem 1

Integrate the sine wave from [0,π] using Midpoint, Trapezoidal, and Simpson's methods. Evaluate the integral for different n (the number of subintervals) and Tabulate the absolute error for different experiments and compare the efficiency of the methods. Plot the relevant graphs to check the behaviour of the absolute error for each method as a function of n.

$$f(x) = sin(x)$$

Find

$$\int_0^{\pi} f(x)dx$$

1.1 Approach

In this problem, we calculate the integral of sin(x) from 0 to π using the Midpoint, Trapezoidal and Simpson's methods. We use simple nested loops to iterate till the value of the integral is obtained to a good accuracy.

1.2 Algorithm

The algorithms for each of the three numerical integration methods are presented below.

1.2.1 Algorithm for the Midpoint method of numerical integration

The pseudocode for the Midpoint method of numerical integration is provided below

Algorithm 1: Approximating $\int_0^{\pi} sin(x)\, dx$ by Midpoint method of numerical integration

```
for j = 2 to 11 do
    n = 2^j, a ← 0
    Δx ← π/2 × n
    for i = 1 to 2 × n do
        a ← a + sin(i × Δx)
        i = i + 2
    end
    sum ← (a/n) × π
    Absolute error = |2 − sum|;
    j = j + 1
end
```

1.2.2 Algorithm for Trapezoidal method of numerical integration

The pseudocode for the Trapezoidal method of numerical integration is provided below.

Algorithm 2: Approximating $\int_0^\pi sin(x)\,dx$ by Trapezoidal method of numerical integration

---

```
for j = 2 to 11 do
    n = 2^j, a ← 0
    Δx ← π/n
    for i = 1 to n do
        if i is 0 then
            a ← a + sin(i × Δx)
        end
        else
            a ← a + sin(i × Δx) × 2
        end
        i = i + 1
    end
    sum ← a/n × π × 0.5
    Absolute error = |2 − sum|;
    j = j + 1
end
```

1.2.3 Algorithm for Simpson's method of numerical integration

The pseudocode for the Simpson's method of numerical integration is provided below.

Algorithm 3: Approximating $\int_0^\pi sin(x)\,dx$ by Simpson's method of numerical integration

---

```
for j = 2 to 11 do
    n = 2^j, a ← 0
    Δx ← π/n
    for i = 1 to n do
        if i is 0 then
            a ← a + sin(i × Δx)
        end
        if i is even then
            a ← a + sin(i × Δx) × 2
        end
        if i is odd then
            a ← a + sin(i × Δx) × 4
        end
        i = i + 1
    end
    sum ← a/(3n) × π
    Absolute error = |2 − sum|;
    j = j + 1
end
```

1.3 Results

The values of the integral obtained are tabulated in table 1 and the absolute errors are tabulated under table 3. Further, we plot the absolute error versus the number of iterations for different methods.

Table 1: Value of numeric integral in each method vs number of intervals

n	**MIDPOINT APPROXIMATION**	**TRAPEZOIDAL APPROXIMATION**	**SIMPSON APPROXIMATION**
4	2.052344	1.896119	2.004560
8	2.012909	1.974232	2.000269
16	2.003216	1.993570	2.000017
32	2.000803	1.998393	2.000001
64	2.000201	1.999598	2.000000
128	2.000050	1.999900	2.000000
256	2.000013	1.999975	2.000000
512	2.000003	1.999994	2.000000
1024	2.000001	1.999998	2.000000

Table 2: Absolute error in each method vs number of intervals

n	**MIDPOINT ERROR**	**TRAPEZOIDAL ERROR**	**SIMPSON ERROR**
4	0.052344305954	0.10388102063	0.004559754984
8	0.012909085599	0.025768398054	0.000269169948
16	0.003216378168	0.006429656228	0.000016591048
32	0.000803416310	0.001606639030	0.000001033369
64	0.000200811728	0.000401611360	0.000000064530
128	0.000050200286	0.000100399816	0.000000004032
256	0.000012549906	0.000025099765	0.000000000252
512	0.000003137466	0.000006274929	0.000000000016
1024	0.00000784366	0.000001568732	0.000000000001

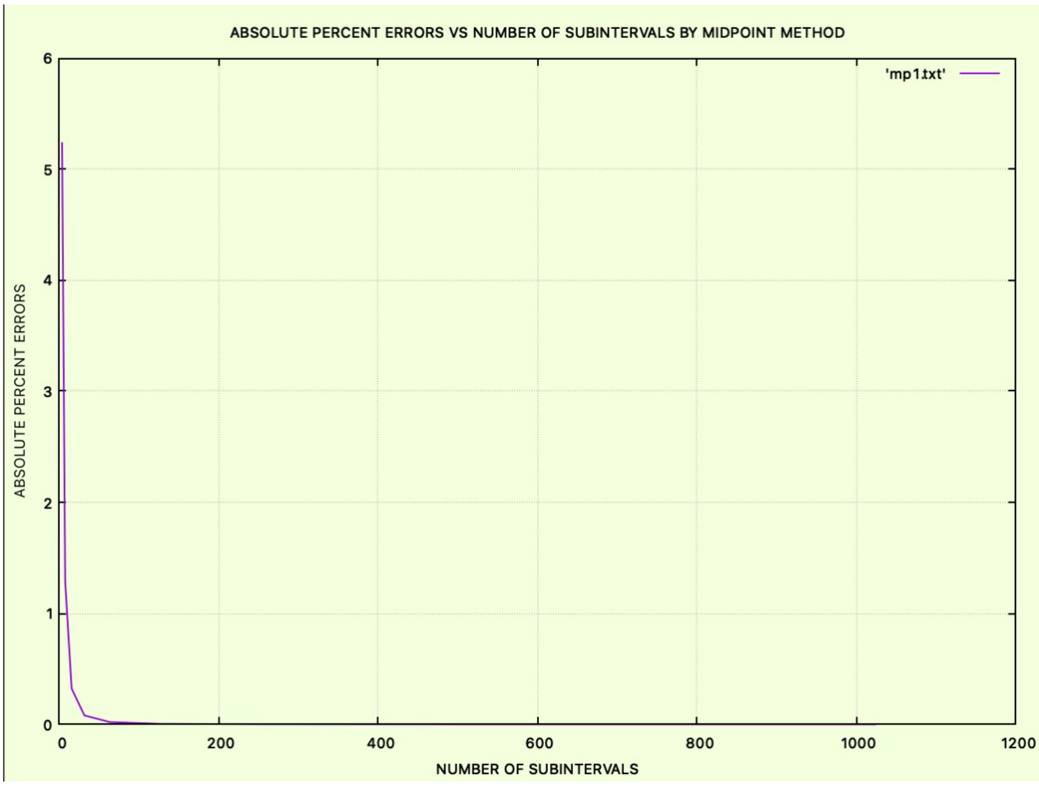


Figure 1: Error in MIDPOINT APPROXIMATION method vs number of subintervals

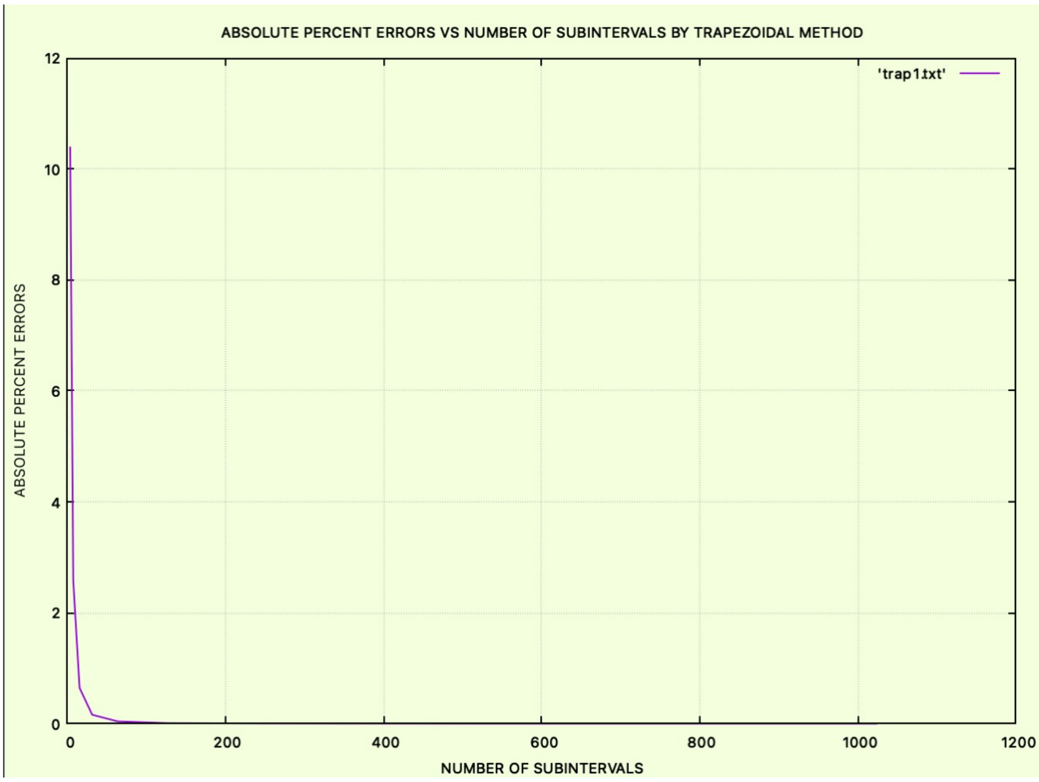


Figure 2: Error in Trapezoidal method vs number of subintervals

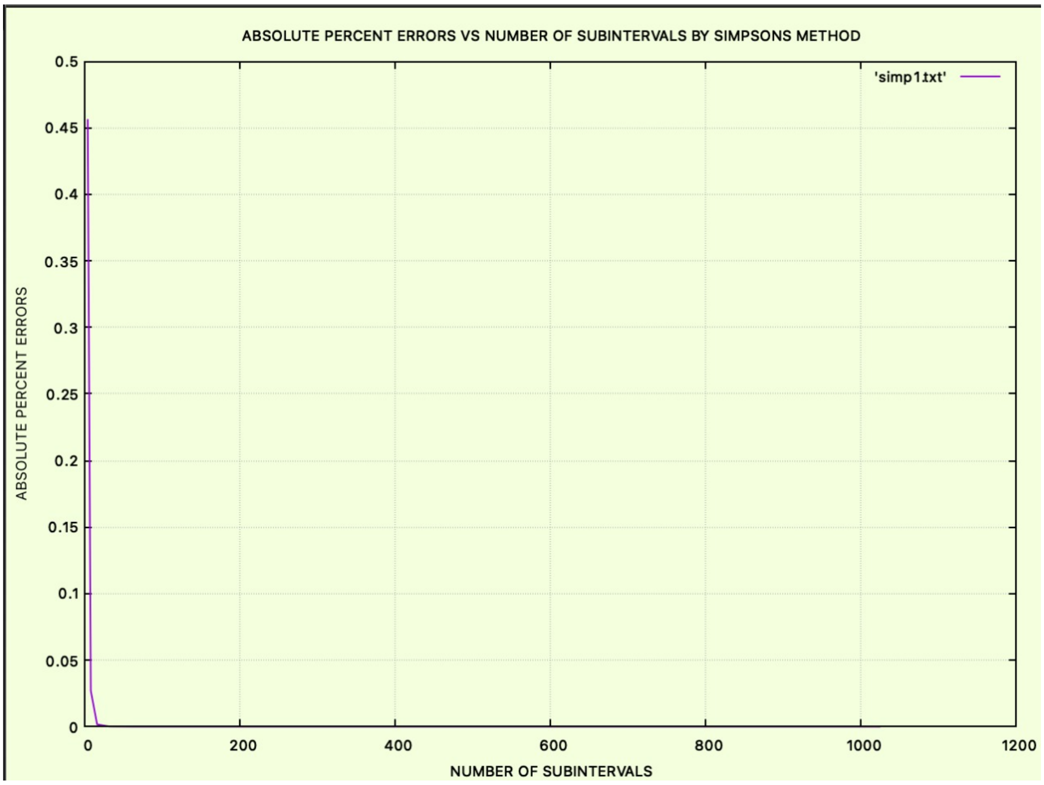


Figure 3: Error in Simpson's method vs number of subintervals

1.4 Inferences

We deduce the following inferences in this experiment:

- The error is high initially and rapidly decreases as n increases exponentially.
- Simpson's method gives the highest accuracy as seen from the graph, followed by the Midpoint method and then the Trapezoidal method.
- We know that the true value of the integral is equal to 2. We observe that Simpson's method and the Midpoint method give us values which are slightly greater than the true value of the integral. On the other hand, the trapezoidal method gives us values which are slightly less than the true value.

1.5 Code

The code used is given below.

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

void mp(); //function definitions
void trap();
void simps();
int main()

{
```

```
    printf("METHOD \t\tN\t ESTIMATE \t ABSOLUTE PERCENT ERROR\n");
    mp();
    trap(); //FUNCTION CALL
    simps();

    FILE *pipei = popen("gnuplot -persist", "w"); //for plotting all the midpoint method results
    fprintf(pipei, "set title \"ABSOLUTE PERCENT ERRORS VS NUMBER OF SUBINTERVALS BY MIDPOINT METHOD\"\n");
    fprintf(pipei, "set xlabel \"NUMBER OF SUBINTERVALS\"\n");
    fprintf(pipei, "set ylabel \"ABSOLUTE PERCENT ERRORS\"\n");
    fprintf(pipei, "set grid\n");
    fprintf(pipei, "plot 'mp1.txt' with lines\n");

    FILE *pipe = popen("gnuplot -persist", "w");  //for plotting all the trapezoidal method results
    fprintf(pipe, "set title \"ABSOLUTE PERCENT ERRORS VS NUMBER OF SUBINTERVALS BY TRAPEZOIDAL METHOD\"\n");
    fprintf(pipe, "set xlabel \"NUMBER OF SUBINTERVALS\"\n");
    fprintf(pipe, "set ylabel \"ABSOLUTE PERCENT ERRORS\"\n");
    fprintf(pipe, "set grid\n");
    fprintf(pipe, "plot 'trap1.txt' with lines\n");

    FILE *pipeu = popen("gnuplot -persist", "w"); //for plotting all the simpsons method results
    fprintf(pipeu, "set title \"ABSOLUTE PERCENT ERRORS VS NUMBER OF SUBINTERVALS BY SIMPSONS METHOD\"\n");
    fprintf(pipeu, "set xlabel \"NUMBER OF SUBINTERVALS\"\n");
    fprintf(pipeu, "set ylabel \"ABSOLUTE PERCENT ERRORS\"\"\n");
    fprintf(pipeu, "set grid\n");
    fprintf(pipeu, "plot 'simp1.txt' with lines\n");


}

void mp() //midpoint method function
{
    double a, s, ae;
    for (int j=2; j<11; j++) //POWER OF 2
    {
        int n=pow(2,j); //NUMBER OF INTERVALS
        a=0; //VALUE OF COUNTER HAS TO BE UPDATED EVERY ITERATION
        double x=M_PI/(n*2);
        for (int i=1; i<2*n; i+=2)
        {

            a=a+sin(i*x); //UPDATING VALUE OF COUNTER

        }
    s=a/n*M_PI;
```

```
    ae=fabs((2-s));
    printf("MIDPOINT\t%d\t %lf\t% lf\n",n,s,ae);
    FILE* data1=fopen("mp1.txt","a+");
    fprintf(data1,"%d\t%f\n",n,ae);

    }

}


void trap() //trapezoidal method
{
    double a2, s2,ae2;
    int  n2, j2=2;

    while (j2<11) //POWER OF 2
    {
        n2=pow(2,j2); //NUMBER OF INTERVALS
        a2=0; //VALUE OF COUNTER HAS TO BE UPDATED EVERY ITERATION
        double x2=M_PI/n2;
        for (int i=0; i<n2; i++)
        {

            if (i==0 || i==M_PI)
            {
                a2=a2+sin(i*x2); //UPDATING VALUE OF COUNTER
            }
            else
            {
                a2=a2 + 2*sin(i*x2); //UPDATING VALUE OF COUNTER
                ↪ ALTERNATIVELY
            }
        }
        s2=a2/n2*M_PI/2;
        ae2=fabs((2-s2));
        printf("TRAPEZOID  \t%d\t %lf\t %lf\n",n2,s2,ae2);
        FILE* data2=fopen("trap1.txt","a+");
        fprintf(data2,"%d\t%f\n",n2,ae2);

        j2++;
    }

}

void simps() //simpsons method
{
    double a3, s3,ae3;
    int  n3, j3=2;

    while (j3<11) //POWER OF 2
```

```
    {
        n3=pow(2,j3); //NUMBER OF INTERVALS
        a3=0; //VALUE OF COUNTER HAS TO BE UPDATED EVERY ITERATION
        double x3=M_PI/n3;
        for (int i=0; i<n3; i++)
        {

            if (i==0 || i==M_PI)
            {
                a3=a3+sin(i*x3); //UPDATING VALUE OF COUNTER FOR ENDING VALUES
            }
            else if (i%2==0)
            {
                a3=a3 + 2*sin(i*x3); //UPDATING VALUE OF COUNTER FOR EVEN VALUES OF INTERVALS
            }
            else if (i%2!=0)
            {
                a3=a3 + 4*sin(i*x3); //UPDATING VALUE OF COUNTER FOR ODD VALUES OF INTERVALS
            }
        }
        s3=a3/n3*M_PI/3;
        ae3=fabs((2-s3));
        printf("SIMPSONS  \t%d\t %lf\t %lf\n",n3,s3,ae3);
        FILE* data3=fopen("simp1.txt","a+");
        fprintf(data3,"%d\t%f\n",n3,ae3);

        j3++;
    }

}
```

Listing 1: Code for all 3 methods.

1.6 Contributions

Complete coding for the first question including GNUPlotting and Latex typing was done by me.

2 Problem 2

Integrate the standard Gaussian pdf to estimate Erf(1) and Erf(2) using Midpoint,Trapezoidal, and Simpson's rules.
Note: Assume the Gaussian PDF has 0 value outside the range [-4,4].

(a) Tabulate the absolute error for different experiments and compare the efficiency of the methods.

(b) Plot the absolute error vs n and explain if there is any anomalous behaviour. Is neglecting the region outside [-4,4] a good choice for calculating the integral with 0.1% accuracy?

(c) Compare the values of Erf(1) and Erf(2) obtained by integration to those obtained using the empirical distribution in the previous assignment.

$$\text{Erf}(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$$

2.1 Approach

In this problem, we calculate the value of Erf(x) from 0 to π using the Midpoint, Trapezoidal and Simpson's methods.
We first define a function that returns the value of $e^{\frac{-x^2}{2}}$. We then call this function in 3 subsequent functions for the 3 different methods. We use simple nested loops to iterate till the value of the integral is obtained to a good accuracy. While doing so, we neglect the region outside [-4,4]. Finally, we plot the errors vs n using GNU plot.

2.2 Algorithm

The algorithms for the 3 methods are given below.

2.2.1 Algorithm for Midpoint method

The pseudocode for the Midpoint method of numerical integration is provided in Algorithm 4.

Algorithm 4: Approximating $f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$ using Midpoint method of numerical integration

---

Defining function f(x)
Defining function for Midpoint method
for $j = 2$ *to* 11 **do**
  $n = 2^j, a \leftarrow 0, ub \leftarrow 1 or 2$
  $\Delta x \leftarrow \frac{ub+4}{n}$
  for $i = 1$ *to* $2 \times n$ **do**
    $a \leftarrow a + f(i \times \Delta x - 4)$
    $i = i + 2$
  end
  $sum \leftarrow \frac{a}{\Delta x \sqrt{2 \times \pi}}$
  $Absolute\ error = |\text{Exact integral depending on ub} - sum|$;
  $j = j + 1$
end

2.2.2 Algorithm for Trapezoidal method of numerical integration

The pseudocode for the Trapezoidal method of numerical integration is provided in Algorithm 5.

Algorithm 5: Approximating $f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}}\, dt$ using Trapezoidal method of numerical integration

---

```
Defining function f(x)
Defining function for Trapezoidal method
for j = 2 to 11 do
    n = 2^j, a ← 0, ub ← 1or2
    Δx ← (ub+4)/n
    for i = 1 to n do
        if i is 0 then
            a ← a + f(i × Δx − 4)
        end
        else
            a ← a + f(i × Δx − 4) × 2
        end
        i = i + 1
    end
    sum ← a / (2Δx√(2×π))
    Absolute error = |Exact integral depending on ub − sum|;
    j = j + 1
end
```

2.2.3 Algorithm for Simpson's method

The pseudocode for the Simpson's method of numerical integration is provided in Algorithm 6.

Algorithm 6: Approximating $f(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$ using Simpson's method of numerical integration

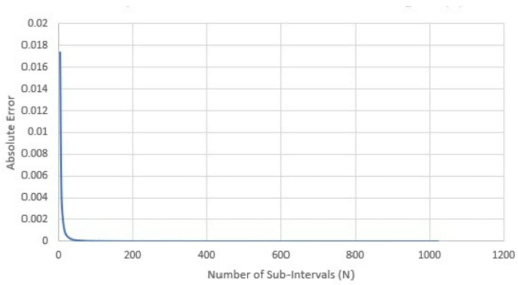
Defining function f(x),Defining function for Simpson's method
for $j = 2$ *to* 11 **do**
  $n = 2^j, a \leftarrow 0, ub \leftarrow 1 or 2$
  $\Delta x \leftarrow \frac{ub+4}{n}$
  for $i = 1$ *to* n **do**
    if i *is* 0 **then**
      $a \leftarrow a + f(i \times \Delta x - 4)$
    end
    if i *is even* **then**
      $a \leftarrow a + 2 \times f(i \times \Delta x - 4)$
    end
    if i *is odd* **then**
      $a \leftarrow a + 4 \times f(i \times \Delta x - 4)$
    end
    $i = i + 1$
  end
  $sum \leftarrow \frac{a}{3\Delta x\sqrt{2\times\pi}}$
  $Absolute\ error = |\text{Exact integral depending on ub} - sum|$;
  $j = j + 1$
end

2.3 Results

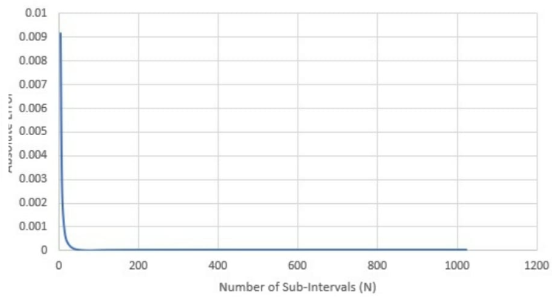
We plot the graphs showing the absolute error percentage as a function of the number of subintervals considered while approximating $Erf(1)$ and $Erf(2)$. The values obtained are also summarized in the tables given below.

Table 3: ERF calculated by EDF

X	ERF
1	0.837
2	0.969

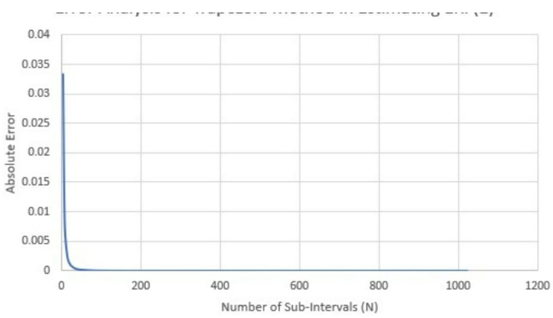


(a) Error graph for **ERF(1) Midpoint method**.

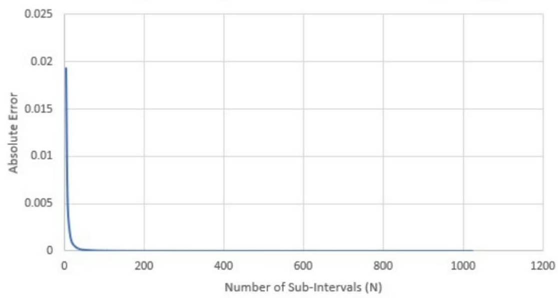


(b) Error graph for **ERF(2) Midpoint method**.

Figure 4: Absolute Error vs number of sub-intervals while calculating using Midpoint Method

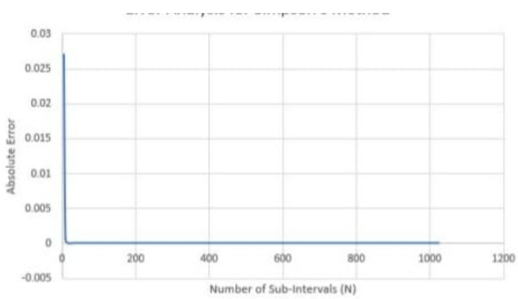


(a) Error graph for **ERF(1) Trapezoidal method**.

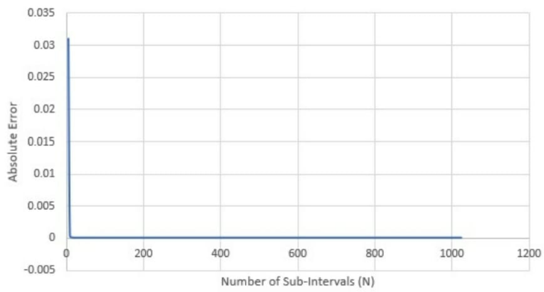


(b) Error graph for **ERF(2) Trapezoidal method**.

Figure 5: Absolute Error vs number of sub-intervals while calculating using Trapezoid Method



(a) Error graph for **ERF(1) Simpson's method**.



(b) Error graph for **ERF(2) Simpson's method**.

Figure 6: Absolute Error vs number of sub-intervals while calculating using Simpson's Method.

n	Error
4	0.017370
8	0.004007
16	0.000960
32	0.000215
64	0.000030
128	0.000017
256	0.000028
512	0.000031
1024	0.000032

Table 4: Absolute Error while approximating **ERF(1)** using Midpoint Method of Numerical Integration

n	Error
4	0.009185
8	0.002462
16	0.000601
32	0.000127
64	0.000008
128	0.000022
256	0.000029
512	0.000031
1024	0.000032

Table 5: Absolute Error while approximating **ERF(2)** using Midpoint Method of Numerical Integration

n	Error
4	0.033431
8	0.008030
16	0.002012
32	0.000526
64	0.000155
128	0.000063
256	0.000040
512	0.000034
1024	0.000032

Table 6: Absolute Error while approximating **ERF(1)** using Trapezoid Method of Numerical Integration

n	Error
4	0.019310
8	0.005062
16	0.001300
32	0.000350
64	0.000111
128	0.000052
256	0.000037
512	0.000033
1024	0.000033

Table 7: Absolute Error while approximating **ERF(2)** using Trapezoid Method of Numerical Integration

n	Error Percent
4	0.027066
8	0.000436
16	0.000005
32	0.000030
64	0.000031
128	0.000031
256	0.000031
512	0.000031
1024	0.000031

Table 8: Error while approximating **ERF(1)** using Simpson Method of Numerical Integration

n	Error
4	0.030970
8	0.000313
16	0.000046
32	0.000031
64	0.000031
128	0.000031
256	0.000031
512	0.000031
1024	0.000031

Table 9: Error while approximating **ERF(2)** using Simpson Method of Numerical Integration

2.4 Inference

We deduce the following inferences in this question:

- Calculating the integral taken in the interval [-4,4] gives an accurate value, with an error percent of under 0.01%.
- The error is initially quite high. However it decreases rapidly as we exponentially increase n.
- The accuracy is highest in Simpson's method, followed by the Midpoint method and then the Trapezoidal method. Using Simpson's method, the error converges to a nearly constant value.
- The values of Erf(1) and Erf(2) obtained from the:
  a) previous assignment are 0.837 and 0.969 respectively.
  b) Simpson's method are 0.841313 and 0.977218 respectively.
  c) Wolfram Alpha (true value used in calculations) are 0.841345 and 0.97725 respectively.
  We therefore observe that there is a good agreement between the Erf values obtained from the previous assignment and those obtained using Simpson's method of Integration. We also note that the errors obtained in each iteration for Erf(1) and Erf(2) are very close to each other.

2.5 Codes

The individual codes used for the experiments are mentioned below.
The code for Midpoint method of numerical integration is given under listing 2, Trapezoidal method of numerical integration is given under listing 3 and Simpson's method of numerical integration is given under listing 4.

```c
#include <stdio.h>//for x=1 we change values similarly for x=2
#include <math.h>
double error (double t) //function for error absolute
{
   double e;
   e = fabs(t - 0.841345) ;
   return e;
}
double error2 (double t) //function for error absolute
{
   double e;
   e = fabs(t - 0.97725) ;
   return e;
}

double midpoint(double a, double b) //function to calculate the midpoint of
 ↪  interval
{
   double bisect;
   bisect = (a+b)/2.0;
   return bisect;
}
```

```
double f(double x) //function definition
{
   double y;
   y = exp((-1)*x*x/2);
   return y;
}
int main()
{
   printf("Erf(1)\n");
   double i,a,b,c;
   double x;
   x = 1.0000;
   double dx;
   for(int j = 4;j<=1024;j=j*2){ //for different values of n
   dx = (x+4.000000)/j;
   double area = 0.0000;
   i = -4.000000;
   for(int k =1;k<=j;k++) //for loop to calculate the integral following the algorithm

   {
      a = i;
      b = i +dx;
      c = midpoint(a,b);
      area = area + dx*f(c);
      i = i + dx;
   }
   area = area*(1/sqrt(2*(M_PI)));
   printf("The area enclosed is %lf\n",area);
   double et;
   et = error(area);
   printf("The error is %lf\n",et);
   }
   printf("Erf(2)\n");


   x = 2.0000;

   for(int j = 4;j<=1024;j=j*2){ //for different values of n
   double dx = (x+4.000000)/j;
   double area = 0.0000;
   double i = -4.000000;
   for(int k =1;k<=j;k++) //for loop to calculate the integral following the algorithm

   {
      a = i;
      b = i +dx;
      c = midpoint(a,b);
      area = area + dx*f(c);
```

```
        i = i + dx;
    }
    area = area*(1/sqrt(2*(M_PI)));
    printf("The area enclosed is %lf\n",area);
    double et;
    et = error2(area);
    printf("The error is %lf\n",et);
    }
}
```

Listing 2: Code for MIDPOINT method

```
#include <stdio.h>
#include <math.h>
//trapezoid
double f(double x) //defining function
{
    //returns f(x)
    double y=exp(-x*x/2); //just inner integrand
    return y;
}
int main()
{
    int n;   //number of intervals
    double x=1;  //value of x as entered by the user

    double s=0.841345;//correct value of erf1
    printf("ERF(1)\n");
    printf("NUMBER OF SUBINTERVALS  ABSOLUTE ERROR   value\n");

     //x1 and x2 are the lower and upper limit, respectively, of each
     ↪  Riemann interval
    double fx1,fx2,fx;
    for (int j=2; j<11; j++)
    {
        n=pow(2,j);
        double i=(x+4)/n,sum=0;  //i=size of each interval
        double x1=-4.0, x2=x1+i;
        while(x2<=x)
        {
            fx1=f(x1); //function value of lower bound
            fx2=f(x2);  //function value of upper bound
            fx=(fx1+fx2)/2*i;  //area of trapezium
            sum+=fx; //updating counter value
            x1=x2; //continuation condition
            x2=x1+i;
        }
    sum = sum /pow(2*M_PI,0.5); //FINAL VALUE OF ERF

    double ae=fabs(sum-s); //ABSOLUTE ERROR CALCULATION
```

```
    printf("%d\t\t\t %lf \t%lf\n",n,ae,sum);

    }
    x=2;  //value of x as entered by the user

    s=0.97725;//correct value of erf1
    printf("ERF(2)\n");
    printf("NUMBER OF SUBINTERVALS  ABSOLUTE ERROR   value\n");


    for (int j=2; j<11; j++)
    {
        n=pow(2,j);
        double i=(x+4)/n,sum=0;  //i=size of each interval
        double x1=-4.0, x2=x1+i;
        while(x2<=x)
        {
            fx1=f(x1); //function value of lower bound
            fx2=f(x2);  //function value of upper bound
            fx=(fx1+fx2)/2*i;  //area of trapezium
            sum+=fx; //updating counter value
            x1=x2; //continuation condition
            x2=x1+i;
        }
    sum = sum /pow(2*M_PI,0.5); //FINAL VALUE OF ERF

    double ae=fabs(sum-s); //ABSOLUTE ERROR CALCULATION

    printf("%d\t\t\t %lf \t%lf\n",n,ae,sum);

    }

}
```

Listing 3: Code for TRAPEZOIDAL method.

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

double simps(); //definition

double f(double x)
{
    return exp(-x*x/2);//function
}

void main()
{
```

```
    printf("METHOD \t\tN\t ESTIMATE \t ABSOLUTE PERCENT ERROR\n"); //tabulation printing headlines

    simps();

}

double  simps()
{
    double a3, s3,ae3,x3;
    int  n3, j3=2;
    printf("Erf(1)\n");
    while (j3<11)//powers of 2 initialisation
    {

        n3=pow(2,j3); //number of subintervals
        a3=0;
        double x=1; //higher bound
        double r=0.841345;
        a3=a3+f(-4)+f(x); //summing the function for beginnning and end
        x3=(x+4)/n3; //size of interval
        for (int i=1; i<n3; i++)
        {
            if (i%2==0)   //  even values
            {
                a3=a3 + 2*f(-4+i*x3);
            }
            else if (i%2!=0) //odd values
            {
                a3=a3 + 4*f(-4+i*x3);
            }
        }
        s3=a3*x3/3/pow(2*M_PI,0.5);//summing up
        ae3=fabs((r-s3));//errors ABSOLUTE
        printf("SIMPSONS  \t%d\t %0.12lf\t %0.12lf\n",n3,s3,ae3);//final print
        j3++;
    }
    printf("Erf(2)\n");
    j3=2;
    while (j3<11)//powers of 2 initialisation
    {

        n3=pow(2,j3); //number of subintervals
        a3=0;
        double x=2; //higher bound
        double r=0.97725;
        a3=a3+f(-4)+f(x); //summing the function for beginnning and end
        x3=(x+4)/n3; //size of interval
        for (int i=1; i<n3; i++)
```

```
        {
            if (i%2==0)   //  even values
            {
                a3=a3 + 2*f(-4+i*x3);
            }
            else if (i%2!=0) //odd values
            {
                a3=a3 + 4*f(-4+i*x3);
            }
        }
        s3=a3*x3/3/pow(2*M_PI,0.5);//summing up
        ae3=fabs((r-s3));//errors ABSOLUTE
        printf("SIMPSONS  \t%d\t %0.12lf\t %0.12lf\n",n3,s3,ae3);//final print
        j3++;
    }

}
```

Listing 4: Code for SIMPSON's method.

2.6 Contributions

Code for Simpson- Amizhthni
Graphs - Anirudh
Code for Trapezoidal- Ankita
Code for Midpoint- Anirudh
Complete Latex work was done by me.