# EE1103: Numerical Methods

# Programming Assignment # 3: Numerical Integration

Lakshmiram S, EE22B117

December 8, 2022

## Contents

## 1 Problem 1

Use the midpoint, trapezoidal and Simpson-1/3 integration methods to
Find the arc length of a cycloid generated by a unit circle.
Calculate the area under the Lemniscate of Bernoulli

$$r^2 = a^2 \cos 2\theta$$

### 1.1 Approach

We first writs c codes for the midpoint, trapezoid and the simpson's method. This can be achieved by simple for-loops.

### 1.2 Algorithm

**Algorithm 1:** midpoint method

```
limits of range=[a,b],number of partitions=n,width of partition=(b-a)/n\\
y(x) is the function
sum=0
for i from 1 to n
    sum=sum+width*y((x_i+x_i-1)/2)
return sum
```

b

**Algorithm 2:** trapezoid method

```
limits of range=[a,b],number of partitions=n,width of partition=(b-a)/n\\
y(x) is the function
sum=0
for i from 1 to n
    sum=sum+0.5*(y(x_i)+y(y(x_i-1))*width
return sum
```

**Algorithm 3:** simpson's method

```
limits of range=[a,b],number of partitions=n,width of partition=(b-a)/n\\
y(x) is the function
sum=0
for i from 1 to n/2
    sum=sum+1/3*width*(y(x_i-1)+4y((x_i-1+x_1)/2)+y(x_i)
return sum
```

### 1.3 Code

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

//function for integrand
float y(float x)
```

```c
{
    return 2*sin(x/2);
}
//function for midpoint
float midpoint(int n,float a,float b)//n: partitions,a and b are the limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=y(a+(width*i)+(width/2))*width;
    }
    return sum;
}
//function for trapezoid
float trapezoid(int n,float a,float b)//n: partitions,a and b are the limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
    }
    return sum;
}
//function for simpson
float simpson(int n,float a,float b)//n: partitions,a and b are the limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n/2;i++)
    {
        sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
    }
    return sum;
}
int main()
{
    float pi=3.142857;
    float actual_value=8;

    printf("printing values for midpoint method\n");
    float actual1=fabs((midpoint(2,0,2*pi)-actual_value)/actual_value)*100;//actual error when n=2
    printf("actual error is: %f\n",actual1);
    //calculating actual and relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=midpoint(pow(2,i),0,2*pi);
        float old=midpoint(pow(2,i-1),0,2*pi);
        float actual_error=fabs((new-actual_value)/actual_value)*100;
        float relative_error=fabs((new-old)/old)*100;
        printf("iteration : %d , actual error : %f ,relative error : %f\n",i-1,actual_error,relative_error);

    }
    printf("\n\n");


    printf("printing values for trapezoid method\n");
    float actual2=fabs((trapezoid(2,0,2*pi)-actual_value)/actual_value)*100;//actual error when n=2
    printf("actual error is: %f\n",actual2);
    //calculating actual and relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=trapezoid(pow(2,i),0,2*pi);
        float old=trapezoid(pow(2,i-1),0,2*pi);
        float actual_error=fabs((new-actual_value)/actual_value)*100;
        float relative_error=fabs((new-old)/old)*100;
        printf("iteration : %d , actual error : %f ,relative error : %f\n",i-1,actual_error,relative_error);

    }
    printf("\n\n");



    printf("printing values for simpson method\n");
    float actual3=fabs((simpson(2,0,2*pi)-actual_value)/actual_value)*100;//actual error when n=2
    printf("actual error is: %f\n",actual3);
    //calculating actual and relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=simpson(pow(2,i),0,2*pi);
        float old=simpson(pow(2,i-1),0,2*pi);
        float actual_error=fabs((new-actual_value)/actual_value)*100;
        float relative_error=fabs((new-old)/old)*100;
```

```c
            printf("iteration : %d , actual error : %f ,relative error : %f\n",i-1,actual_error,relative_error);

    }
}


```

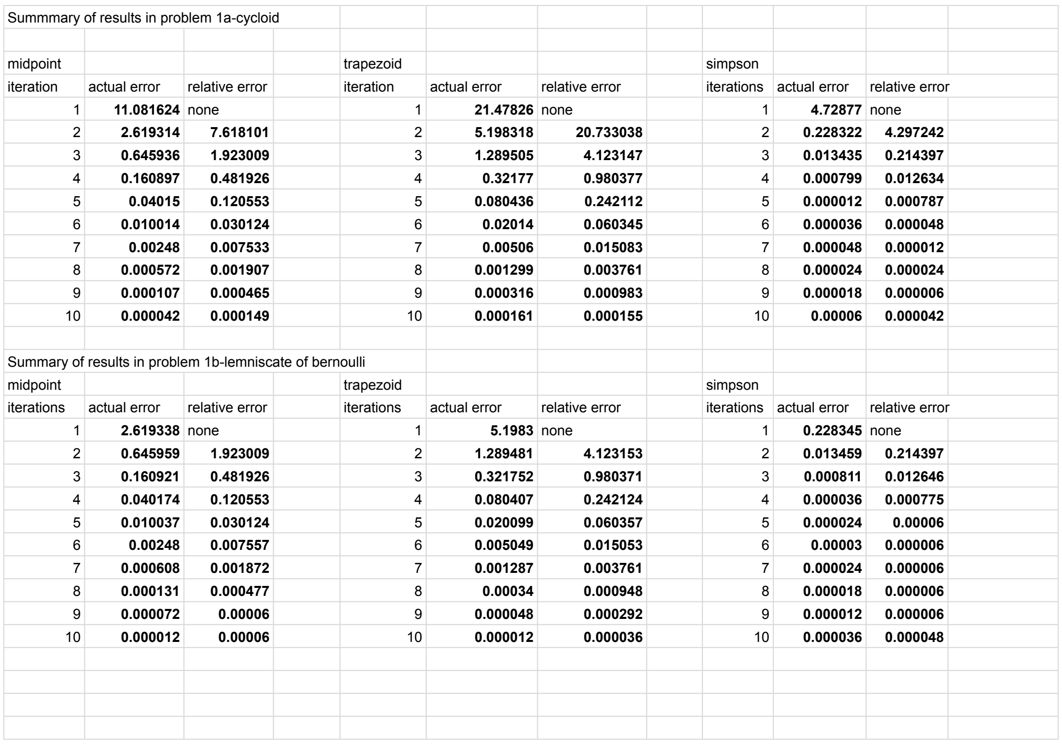
Listing 1: Code for 1a.

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

//function for integrand
float y(float x)
{
    return 2*cos(2*x);
}
//function for midpoint
float midpoint(int n,float a,float b)//n: partitions,a and b are the limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=y(a+(width*i)+(width/2))*width;
    }
    return sum;
}
//function for trapezoid
float trapezoid(int n,float a,float b)//n: partitions,a and b are the limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
    }
    return sum;
}
//function for simpson
float simpson(int n,float a,float b)//n: partitions,a and b are the limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n/2;i++)
    {
        sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
    }
    return sum;
}
int main()
{
    float pi=3.142857;
    float actual_value=1;

    printf("printing values for midpoint method\n");
    float
↪   actual1=fabs((midpoint(2,0,pi/4)-actual_value)/actual_value)*100;//actual
↪   error when n=2
    printf("actual error is: %f\n",actual1);
    //calculating actual and relative errors for powers of 2 starting from
↪   4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=midpoint(pow(2,i),0,pi/4);
        float old=midpoint(pow(2,i-1),0,pi/4);
        float actual_error=fabs((new-actual_value)/actual_value)*100;
        float relative_error=fabs((new-old)/old)*100;
        printf("iteration : %d , actual error : %f ,relative error :
↪   %f\n",i-1,actual_error,relative_error);


    }
    printf("\n\n");


    printf("printing values for trapezoid method\n");
    float
↪   actual2=fabs((trapezoid(2,0,pi/4)-actual_value)/actual_value)*100;//actual
↪   error when n=2
    printf("actual error is: %f\n",actual2);
    //calculating actual and relative errors for powers of 2 starting from
↪   4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=trapezoid(pow(2,i),0,pi/4);
        float old=trapezoid(pow(2,i-1),0,pi/4);
        float actual_error=fabs((new-actual_value)/actual_value)*100;
        float relative_error=fabs((new-old)/old)*100;
        printf("iteration : %d , actual error : %f ,relative error :
↪   %f\n",i-1,actual_error,relative_error);

    }
    printf("\n\n");



    printf("printing values for simpson method\n");
    float actual3=fabs((simpson(2,0,pi/4)-actual_value)/actual_value)*100;//actual error when n=2
    printf("actual error is: %f\n",actual3);
    //calculating actual and relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=simpson(pow(2,i),0,pi/4);
        float old=simpson(pow(2,i-1),0,pi/4);
        float actual_error=fabs((new-actual_value)/actual_value)*100;
        float relative_error=fabs((new-old)/old)*100;
        //printf("iteration : %d , actual error : %f ,relative error : %f\n",i-1,actual_error,relative_error);
        printf("%f\n",relative_error);
    }
}


```

Listing 2: Code for 1b.

## 1.4 Results

The output from the above code has been compiled below.



Summmary of results in problem 1a-cycloid

| midpoint | | | trapezoid | | | simpson | | |
|---|---|---|---|---|---|---|---|---|
| iteration | actual error | relative error | iteration | actual error | relative error | iterations | actual error | relative error |
| 1 | **11.081624** | none | 1 | **21.47826** | none | 1 | **4.72877** | none |
| 2 | **2.619314** | **7.618101** | 2 | **5.198318** | **20.733038** | 2 | **0.228322** | **4.297242** |
| 3 | **0.645936** | **1.923009** | 3 | **1.289505** | **4.123147** | 3 | **0.013435** | **0.214397** |
| 4 | **0.160897** | **0.481926** | 4 | **0.32177** | **0.980377** | 4 | **0.000799** | **0.012634** |
| 5 | **0.04015** | **0.120553** | 5 | **0.080436** | **0.242112** | 5 | **0.000012** | **0.000787** |
| 6 | **0.010014** | **0.030124** | 6 | **0.02014** | **0.060345** | 6 | **0.000036** | **0.000048** |
| 7 | **0.00248** | **0.007533** | 7 | **0.00506** | **0.015083** | 7 | **0.000048** | **0.000012** |
| 8 | **0.000572** | **0.001907** | 8 | **0.001299** | **0.003761** | 8 | **0.000024** | **0.000024** |
| 9 | **0.000107** | **0.000465** | 9 | **0.000316** | **0.000983** | 9 | **0.000018** | **0.000006** |
| 10 | **0.000042** | **0.000149** | 10 | **0.000161** | **0.000155** | 10 | **0.00006** | **0.000042** |

Summary of results in problem 1b-lemniscate of bernoulli

| midpoint | | | trapezoid | | | simpson | | |
|---|---|---|---|---|---|---|---|---|
| iterations | actual error | relative error | iterations | actual error | relative error | iterations | actual error | relative error |
| 1 | **2.619338** | none | 1 | **5.1983** | none | 1 | **0.228345** | none |
| 2 | **0.645959** | **1.923009** | 2 | **1.289481** | **4.123153** | 2 | **0.013459** | **0.214397** |
| 3 | **0.160921** | **0.481926** | 3 | **0.321752** | **0.980371** | 3 | **0.000811** | **0.012646** |
| 4 | **0.040174** | **0.120553** | 4 | **0.080407** | **0.242124** | 4 | **0.000036** | **0.000775** |
| 5 | **0.010037** | **0.030124** | 5 | **0.020099** | **0.060357** | 5 | **0.000024** | **0.00006** |
| 6 | **0.00248** | **0.007557** | 6 | **0.005049** | **0.015053** | 6 | **0.00003** | **0.000006** |
| 7 | **0.000608** | **0.001872** | 7 | **0.001287** | **0.003761** | 7 | **0.000024** | **0.000006** |
| 8 | **0.000131** | **0.000477** | 8 | **0.00034** | **0.000948** | 8 | **0.000018** | **0.000006** |
| 9 | **0.000072** | **0.00006** | 9 | **0.000048** | **0.000292** | 9 | **0.000012** | **0.000006** |
| 10 | **0.000012** | **0.00006** | 10 | **0.000012** | **0.000036** | 10 | **0.000036** | **0.000048** |

Figure 1: summary of results from various methods

## 1.5 Inferences

- no particular deductions can be made from the values presented in the table. No comments can be given about the initial error percentage nor the rate at which the error converges.
- what is curious though is the fact that the simpson's integral can be expressed as the weighted mean of the midpoint and the trapezoidal methods as:

$$S_{2n} = \frac{2}{3}M_n + \frac{1}{3}T_n$$

## 1.6 Contributions

I got to work on this assignment independently.As for the additional software employed for generating tables and plots, Google sheets was used for the job.

## 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?

$$Erf(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-x^2/2} dx$$

### 2.1 Approach

The same three methods are used for this question as well.

### 2.2 Algorithm

The algorithm is the same as the previous question.

### 2.3 Code

```c
#include<stdio.h>
#include<math.h>
float y(float x)
{
    float pi=3.142857;
    return exp(-pow(x,2)/2)/(sqrt(2*pi));
}
//function for midpoint
float midpoint(int n,float a,float b)//n: partitions,a and b are the limits
    of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=y(a+(width*i)+(width/2))*width;
    }
    return sum;
}
//function for trapezoid
float trapezoid(int n,float a,float b)//n: partitions,a and b are the
    limits of integragtion.
{
    float sum=0;float width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
    }
    return sum;
}
//function for simpson
double simpson(int n,double a,double b)//n: partitions,a and b are the limits of integragtion.
{
    double sum=0;double width=(b-a)/n;
    for(int i=0;i<n/2;i++)
    {
        sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
    }
    return sum;
}
int main()
{
    printf("printing values for midpoint method\n");
    //calculating relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=0.5+midpoint(pow(2,i),0,1);
        float old=0.5+midpoint(pow(2,i-1),0,1);
        float relative_error=fabs((new-old)/old)*100;
        //printf("iteration : %d ,relative error : %f\n",i-1,relative_error);
        printf("%f\n",relative_error);
    }
    printf("\n\n");

    printf("printing values for trapezium method\n");
    //calculating relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        float new=0.5+trapezoid(pow(2,i),0,1);
        float old=0.5+trapezoid(pow(2,i-1),0,1);
        float relative_error=fabs((new-old)/old)*100;
        //printf("iteration : %d ,relative error : %f\n",i-1,relative_error);
        printf("%f\n",relative_error);
    }
    printf("\n\n");

    printf("printing values for simpson method\n");
    //calculating relative errors for powers of 2 starting from 4 till 1024.
    for(int i=2;i<11;i++)
    {

        double new=0.5+simpson(pow(2,i),0,1);
        double old=0.5+simpson(pow(2,i-1),0,1);
        double relative_error=fabs((new-old)/old)*100;
        //printf("iteration : %d ,relative error :
 ↪  %f\n",i-1,relative_error);
        printf("%f\n",relative_error);
    }
    printf("\n\n");
}

```

Listing 3: Code for 2.

### 2.4 Results

Erf(1) obtained by using the actual Gaussian PDF is 0.841276 The output from the above code has been compiled below.

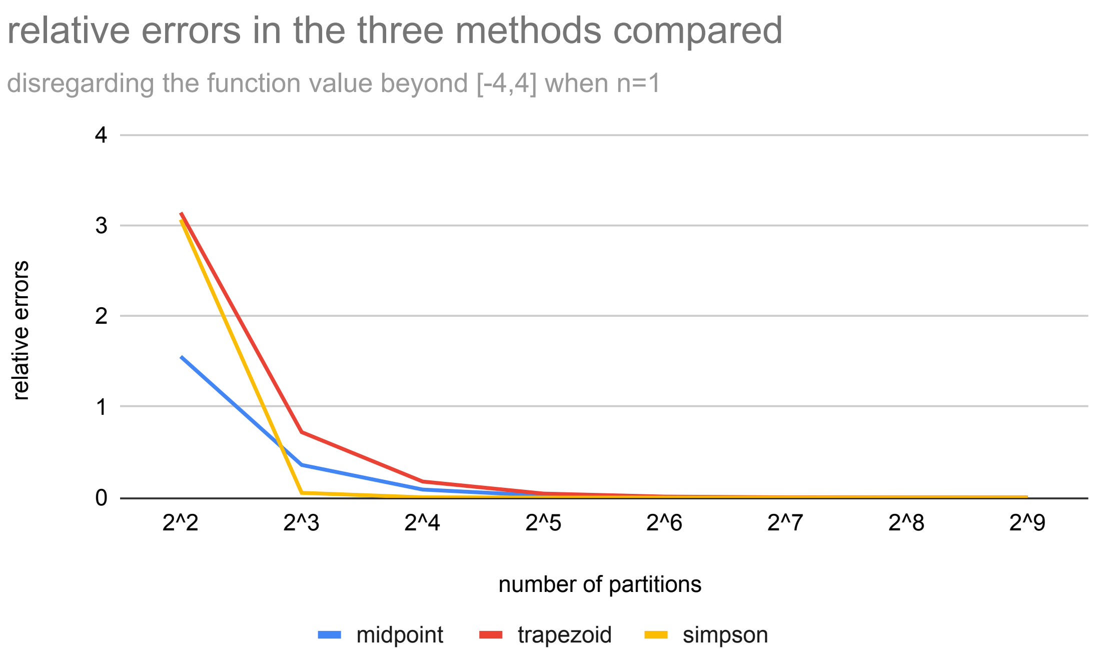


Figure 2

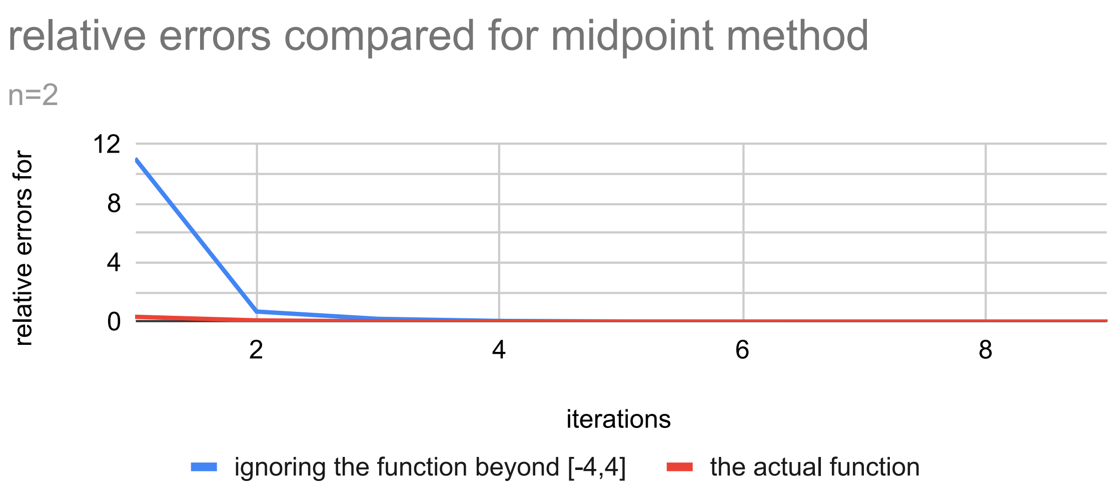


Figure 3

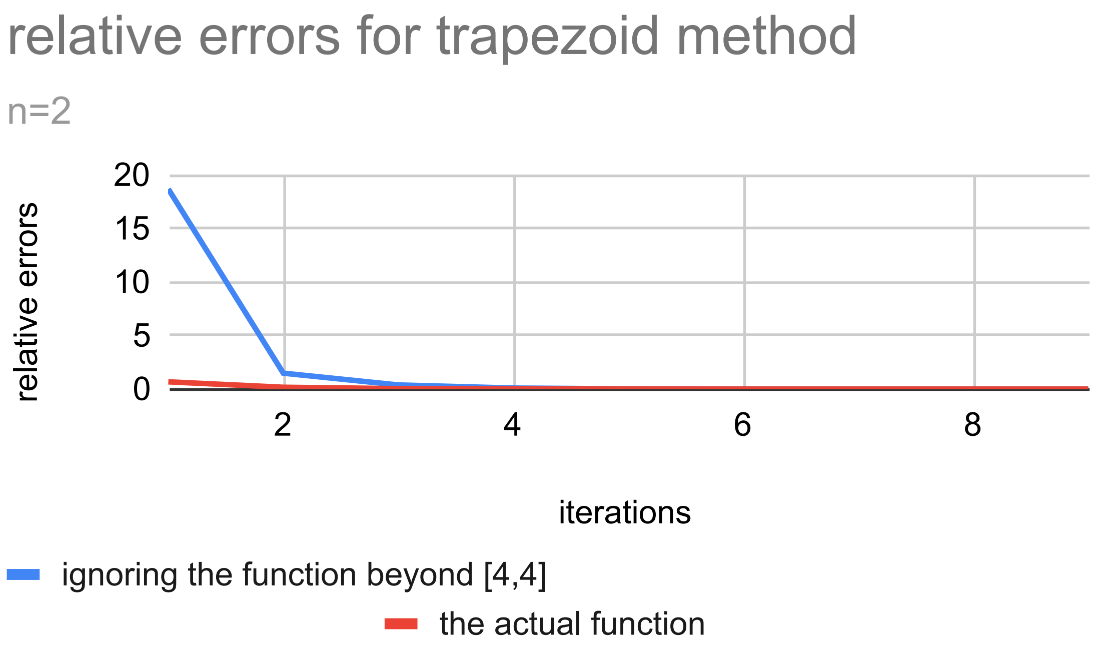


Figure 4

### 2.5 Inferences

- calculating the actual error in this case is not possible because the given integral cannot be evaluated exactly.
- Based on the graphs, it is very clear that using the actual function is much better than ignoring it beyond [-4,4] with regards to how fast the relative error reduces.

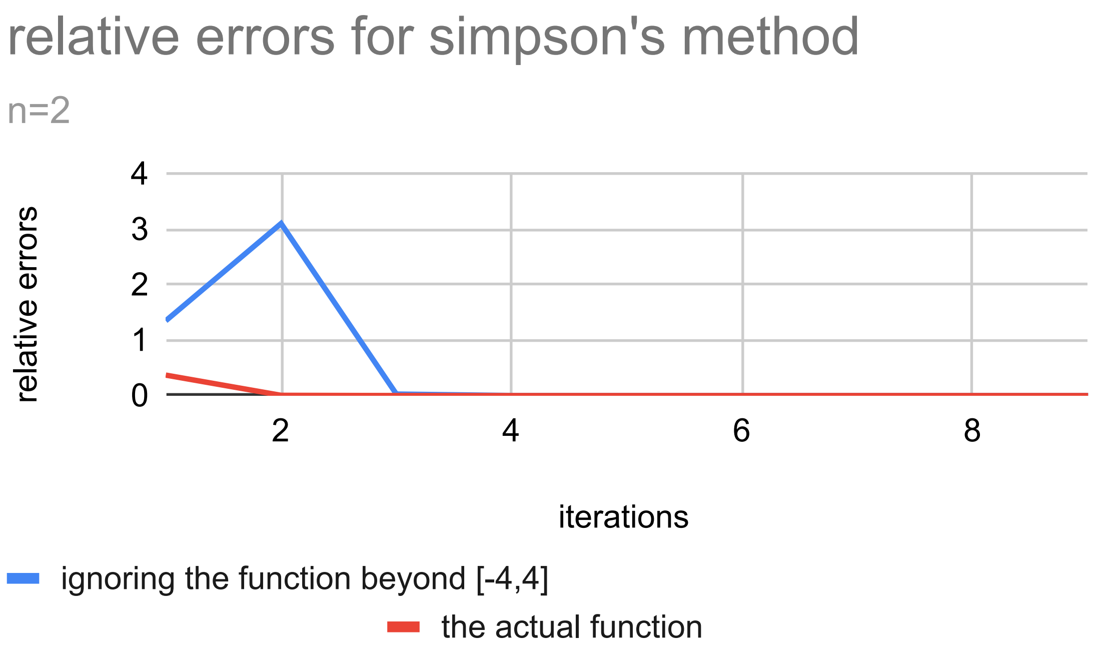


Figure 5

### 2.6 Contributions

I got to work on this assignment independently.As for the additional software employed for generating tables and plots, Google sheets was used for the job.

## 3 Problem 3

Using numerical integration, verify the following result,known as the sophomore's dream, in the best manner you can with supporting error analysis.

$$\int_0^1 x^{-x} dx = \sum_{i=1}^{\infty} n^{-n}$$

### 3.1 Approach

The same three methods are used for this question as well.

### 3.2 Algorithm

The algorithm is the same as the previous question.

### 3.3 Code

The code used for the experiments is mentioned in Listing 4.

```c
#include <stdio.h>
#include <math.h>
//function for integrand
double y(double x)
{
    return pow(x,-x);
}
//function for midpoint
double midpoint(int n,double a,double b)//n: partitions,a and b are the
    limits of integragtion.
{
    double sum=0;double width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=y(a+(width*i)+(width/2))*width;
    }
    return sum;
}
//function for trapezoid
double trapezoid(int n,double a,double b)//n: partitions,a and b are the
    limits of integragtion.
{
    double sum=0;double width=(b-a)/n;
    for(int i=0;i<n;i++)
    {
        sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
    }
    return sum;
}
//function for simpson
double simpson(int n,double a,double b)//n: partitions,a and b are the
    limits of integragtion.
```

```
{
    double sum=0;double width=(b-a)/n;
    for(int i=0;i<n/2;i++)
    {
↪   sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
    }
    return sum;
}
int main()
{
    double sum=0;
    for(int i=0;i<6;i++)
    {
        sum+=pow(i,-i);
    }
    double actual_value=sum;//the RHS of the equality

    printf("printing values for midpoint method\n");
    for(int i=1;i<11;i++)
    {
        double
↪   actual_error=fabs((midpoint(pow(2,i),0,1)-actual_value)/(actual_value))*100;
        //printf("iterations : %d ,actual error: %lf\n",i,actual_error);
        printf("%lf\n",actual_error);
    }
    printf("\n\n");

    printf("printing values for trapezoid method\n");
    for(int i=1;i<11;i++)
    {
        double
↪   actual_error=fabs((trapezoid(pow(2,i),0,1)-actual_value)/(actual_value))*100;
       // printf("iterations : %d ,actual error: %lf\n",i,actual_error);
        printf("%lf\n",actual_error);
    }
    printf("\n\n");

    printf("printing values for simpson method\n");
    for(int i=1;i<11;i++)
    {
        double
↪   actual_error=fabs((simpson(pow(2,i),0,1)-actual_value)/(actual_value))*100;
        //printf("iterations : %d ,actual error: %lf\n",i,actual_error);
        printf("%lf\n",actual_error);
    }
    printf("\n\n");
}
```

Listing 4: Code for 3.

## 3.4 Results

We limit the calculation of the summation to n=6, because the data types defined in c lose precision beyond a certain extent.
But we observe significant error percentages in all the three methods, approximately 45%

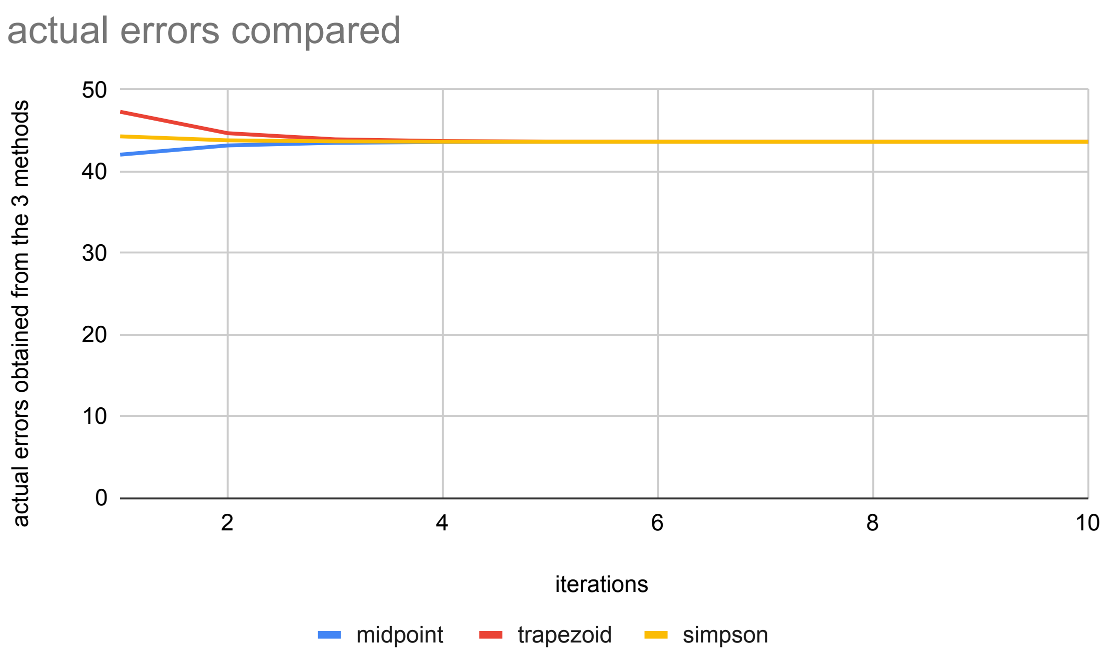


Figure 6

## 3.5 Inferences

The error values would be much lower if we could tend the value of n to infinity, but such a process is not practically viable due to precision problems.

## 3.6 Contributions

I got to work on this assignment independently.As for the additional software employed for generating tables and plots, Google sheets was used for the job.