

EE1103: Numerical Methods

Programming Assignment # 3: Numerical Integration

Lakshmiram S, EE22B117

December 8, 2022

Contents

1	Problem 1	1
1.1	Approach	1
1.2	Algorithm	1
1.3	Code	1
1.4	Results	6
1.5	Inferences	7
1.6	Contributions	7
2	Problem 2	8
2.1	Approach	8
2.2	Algorithm	8
2.3	Code	8
2.4	Results	10
2.5	Inferences	11
2.6	Contributions	12
3	Problem 3	13
3.1	Approach	13
3.2	Algorithm	13
3.3	Code	13
3.4	Results	15
3.5	Inferences	15
3.6	Contributions	15

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 write 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
b for i from 1 to n
    sum=sum+width*y((x_i+x_{i-1})/2)
return sum
```

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(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_i)/2)+y(x_i))
return sum
```

1.3 Code

```
1 #include <stdio.h>
2 #include <math.h>
3
4 //function for integrand
5 float y(float x)
```

```

6 {
7     return 2*sin(x/2);
8 }
9 //function for midpoint
10 float midpoint(int n,float a,float b)//n: partitions,a and b are the limits
    ↪ of integration.
11 {
12     float sum=0;float width=(b-a)/n;
13     for(int i=0;i<n;i++)
14     {
15         sum+=y(a+(width*i)+(width/2))*width;
16     }
17     return sum;
18 }
19 //function for trapezoid
20 float trapezoid(int n,float a,float b)//n: partitions,a and b are the
    ↪ limits of integration.
21 {
22     float sum=0;float width=(b-a)/n;
23     for(int i=0;i<n;i++)
24     {
25         sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
26     }
27     return sum;
28 }
29 //function for simpson
30 float simpson(int n,float a,float b)//n: partitions,a and b are the limits
    ↪ of integration.
31 {
32     float sum=0;float width=(b-a)/n;
33     for(int i=0;i<n/2;i++)
34     {
35         ↪ sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
36     }
37     return sum;
38 }
39 int main()
40 {
41     float pi=3.142857;
42     float actual_value=8;
43
44     printf("printing values for midpoint method\n");
45     float
    ↪ actual1=fabs((midpoint(2,0,2*pi)-actual_value)/actual_value)*100;//actual
    ↪ error when n=2
46     printf("actual error is: %f\n",actual1);
47     //calculating actual and relative errors for powers of 2 starting from
    ↪ 4 till 1024.
48     for(int i=2;i<11;i++)

```

```

49     {
50
51         float new=midpoint(pow(2,i),0,2*pi);
52         float old=midpoint(pow(2,i-1),0,2*pi);
53         float actual_error=fabs((new-actual_value)/actual_value)*100;
54         float relative_error=fabs((new-old)/old)*100;
55         printf("iteration : %d , actual error : %f ,relative error :
↪ %f\n",i-1,actual_error,relative_error);
56
57     }
58     printf("\n\n");
59
60
61     printf("printing values for trapezoid method\n");
62     float
↪ actual2=fabs((trapezoid(2,0,2*pi)-actual_value)/actual_value)*100; //actual
↪ error when n=2
63     printf("actual error is: %f\n",actual2);
64     //calculating actual and relative errors for powers of 2 starting from
↪ 4 till 1024.
65     for(int i=2;i<11;i++)
66     {
67
68         float new=trapezoid(pow(2,i),0,2*pi);
69         float old=trapezoid(pow(2,i-1),0,2*pi);
70         float actual_error=fabs((new-actual_value)/actual_value)*100;
71         float relative_error=fabs((new-old)/old)*100;
72         printf("iteration : %d , actual error : %f ,relative error :
↪ %f\n",i-1,actual_error,relative_error);
73
74     }
75     printf("\n\n");
76
77
78
79     printf("printing values for simpson method\n");
80     float
↪ actual3=fabs((simpson(2,0,2*pi)-actual_value)/actual_value)*100; //actual
↪ error when n=2
81     printf("actual error is: %f\n",actual3);
82     //calculating actual and relative errors for powers of 2 starting from
↪ 4 till 1024.
83     for(int i=2;i<11;i++)
84     {
85
86         float new=simpson(pow(2,i),0,2*pi);
87         float old=simpson(pow(2,i-1),0,2*pi);
88         float actual_error=fabs((new-actual_value)/actual_value)*100;
89         float relative_error=fabs((new-old)/old)*100;

```

```

90     printf("iteration : %d , actual error : %f ,relative error :
↪ %f\n",i-1,actual_error,relative_error);
91
92     }
93 }
94
95

```

Listing 1: Code for 1a.

```

1  #include <stdio.h>
2  #include <math.h>
3
4  //function for integrand
5  float y(float x)
6  {
7      return 2*cos(2*x);
8  }
9  //function for midpoint
10 float midpoint(int n,float a,float b)//n: partitions,a and b are the limits
↪ of integration.
11 {
12     float sum=0;float width=(b-a)/n;
13     for(int i=0;i<n;i++)
14     {
15         sum+=y(a+(width*i)+(width/2))*width;
16     }
17     return sum;
18 }
19 //function for trapezoid
20 float trapezoid(int n,float a,float b)//n: partitions,a and b are the
↪ limits of integration.
21 {
22     float sum=0;float width=(b-a)/n;
23     for(int i=0;i<n;i++)
24     {
25         sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
26     }
27     return sum;
28 }
29 //function for simpson
30 float simpson(int n,float a,float b)//n: partitions,a and b are the limits
↪ of integration.
31 {
32     float sum=0;float width=(b-a)/n;
33     for(int i=0;i<n/2;i++)
34     {
35
↪ sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
36     }

```

```

37     return sum;
38 }
39 int main()
40 {
41     float pi=3.142857;
42     float actual_value=1;
43
44     printf("printing values for midpoint method\n");
45     float
    ↪ actual1=fabs((midpoint(2,0,pi/4)-actual_value)/actual_value)*100; //actual
    ↪ error when n=2
46     printf("actual error is: %f\n",actual1);
47     //calculating actual and relative errors for powers of 2 starting from
    ↪ 4 till 1024.
48     for(int i=2;i<11;i++)
49     {
50
51         float new=midpoint(pow(2,i),0,pi/4);
52         float old=midpoint(pow(2,i-1),0,pi/4);
53         float actual_error=fabs((new-actual_value)/actual_value)*100;
54         float relative_error=fabs((new-old)/old)*100;
55         printf("iteration : %d , actual error : %f ,relative error :
    ↪ %f\n",i-1,actual_error,relative_error);
56
57     }
58     printf("\n\n");
59
60
61
62     printf("printing values for trapezoid method\n");
63     float
    ↪ actual2=fabs((trapezoid(2,0,pi/4)-actual_value)/actual_value)*100; //actual
    ↪ error when n=2
64     printf("actual error is: %f\n",actual2);
65     //calculating actual and relative errors for powers of 2 starting from
    ↪ 4 till 1024.
66     for(int i=2;i<11;i++)
67     {
68
69         float new=trapezoid(pow(2,i),0,pi/4);
70         float old=trapezoid(pow(2,i-1),0,pi/4);
71         float actual_error=fabs((new-actual_value)/actual_value)*100;
72         float relative_error=fabs((new-old)/old)*100;
73         printf("iteration : %d , actual error : %f ,relative error :
    ↪ %f\n",i-1,actual_error,relative_error);
74
75     }
76     printf("\n\n");
77
78

```

```

79
80     printf("printing values for simpson method\n");
81     float
82     ↪ actual3=fabs((simpson(2,0,pi/4)-actual_value)/actual_value)*100; //actual
83     ↪ error when n=2
84     printf("actual error is: %f\n",actual3);
85     //calculating actual and relative errors for powers of 2 starting from
86     ↪ 4 till 1024.
87     for(int i=2;i<11;i++)
88     {
89         float new=simpson(pow(2,i),0,pi/4);
90         float old=simpson(pow(2,i-1),0,pi/4);
91         float actual_error=fabs((new-actual_value)/actual_value)*100;
92         float relative_error=fabs((new-old)/old)*100;
93         //printf("iteration : %d , actual error : %f ,relative error :
94     ↪ %f\n",i-1,actual_error,relative_error);
95         printf("%f\n",relative_error);
96     }

```

Listing 2: Code for 1b.

1.4 Results

The output from the above code has been compiled below.

Summary 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?

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

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

```
1 #include <stdio.h>
2 #include <math.h>
3 float y(float x)
4 {
5     float pi=3.142857;
6     return exp(-pow(x,2)/2)/(sqrt(2*pi));
7 }
8 //function for midpoint
9 float midpoint(int n,float a,float b)//n: partitions,a and b are the limits
   ↳ of integration.
10 {
11     float sum=0;float width=(b-a)/n;
12     for(int i=0;i<n;i++)
13     {
14         sum+=y(a+(width*i)+(width/2))*width;
15     }
16     return sum;
17 }
18 //function for trapezoid
19 float trapezoid(int n,float a,float b)//n: partitions,a and b are the
   ↳ limits of integration.
20 {
21     float sum=0;float width=(b-a)/n;
22     for(int i=0;i<n;i++)
23     {
24         sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
25     }
```

```

26     return sum;
27 }
28 //function for simpson
29 double simpson(int n,double a,double b)//n: partitions,a and b are the
    ↪ limits of integration.
30 {
31     double sum=0;double width=(b-a)/n;
32     for(int i=0;i<n/2;i++)
33     {
34         ↪ sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
35     }
36     return sum;
37 }
38 int main()
39 {
40     printf("printing values for midpoint method\n");
41     //calculating relative errors for powers of 2 starting from 4 till
    ↪ 1024.
42     for(int i=2;i<11;i++)
43     {
44
45         float new=0.5+midpoint(pow(2,i),0,1);
46         float old=0.5+midpoint(pow(2,i-1),0,1);
47         float relative_error=fabs((new-old)/old)*100;
48         //printf("iteration : %d ,relative error :
    ↪ %f\n",i-1,relative_error);
49         printf("%f\n",relative_error);
50     }
51     printf("\n\n");
52
53     printf("printing values for trapezium method\n");
54     //calculating relative errors for powers of 2 starting from 4 till
    ↪ 1024.
55     for(int i=2;i<11;i++)
56     {
57
58         float new=0.5+trapezoid(pow(2,i),0,1);
59         float old=0.5+trapezoid(pow(2,i-1),0,1);
60         float relative_error=fabs((new-old)/old)*100;
61         //printf("iteration : %d ,relative error :
    ↪ %f\n",i-1,relative_error);
62         printf("%f\n",relative_error);
63     }
64     printf("\n\n");
65
66     printf("printing values for simpson method\n");
67     //calculating relative errors for powers of 2 starting from 4 till
    ↪ 1024.
68     for(int i=2;i<11;i++)

```

```

69     {
70
71         double new=0.5+simpson(pow(2,i),0,1);
72         double old=0.5+simpson(pow(2,i-1),0,1);
73         double relative_error=fabs((new-old)/old)*100;
74         //printf("iteration : %d ,relative error :
↪ %f\n",i-1,relative_error);
75         printf("%f\n",relative_error);
76     }
77     printf("\n\n");
78 }
79

```

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.

relative errors in the three methods compared

disregarding the function value beyond $[-4,4]$ when $n=1$

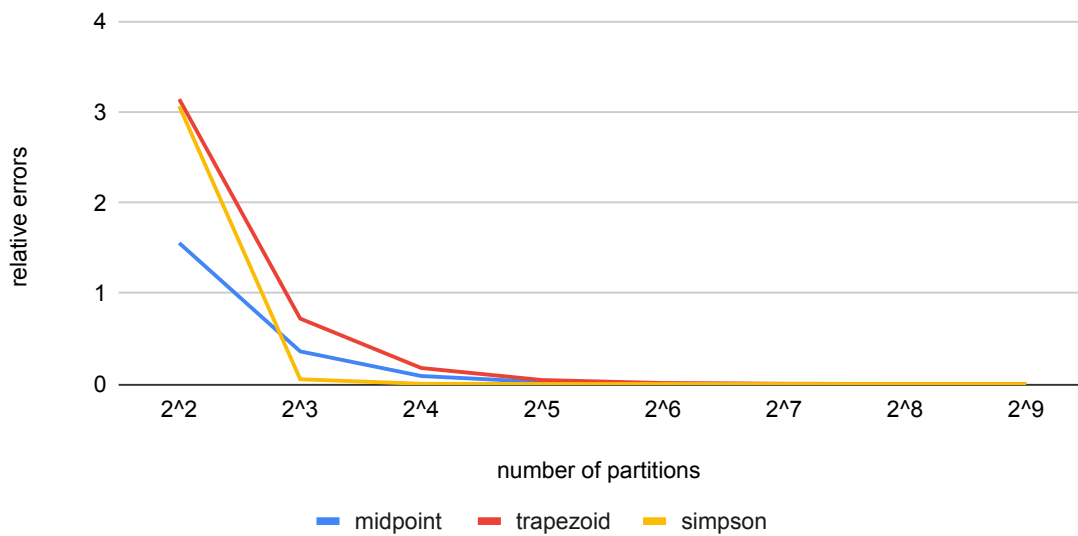


Figure 2

relative errors compared for midpoint method

n=2

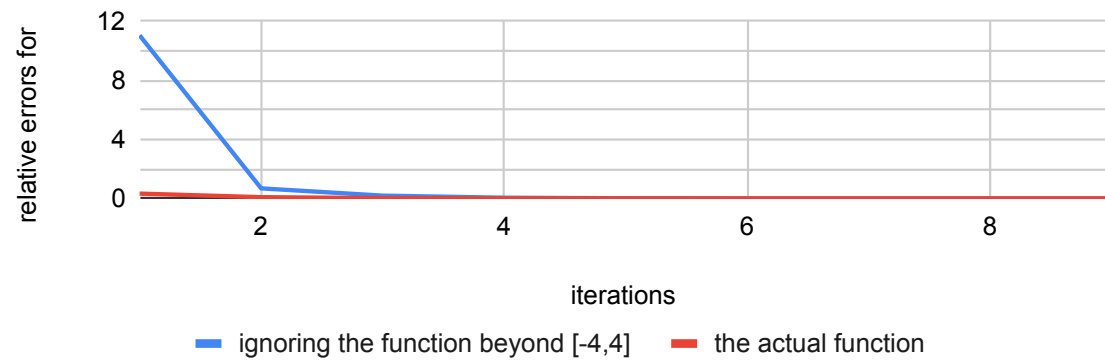


Figure 3

relative errors for trapezoid method

n=2

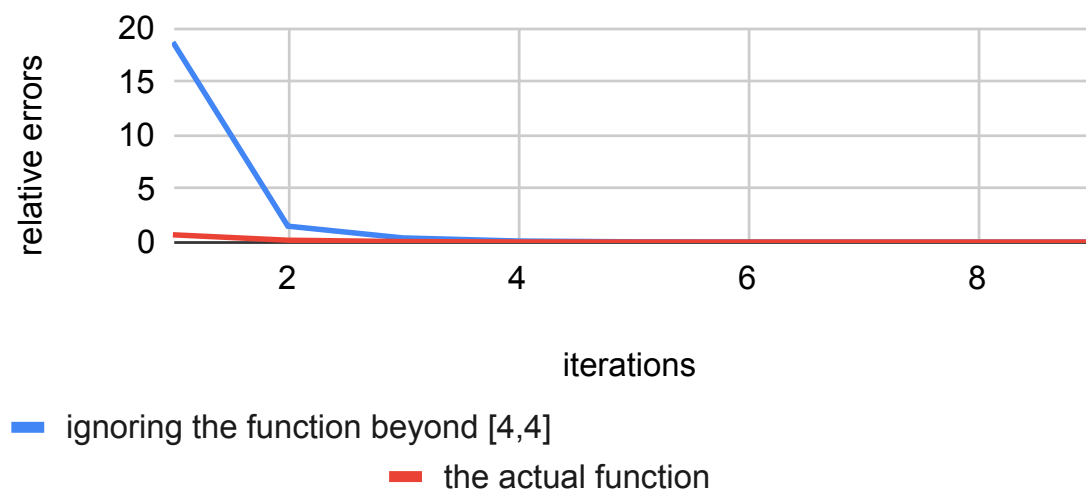


Figure 4

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.

relative errors for simpson's method

$n=2$

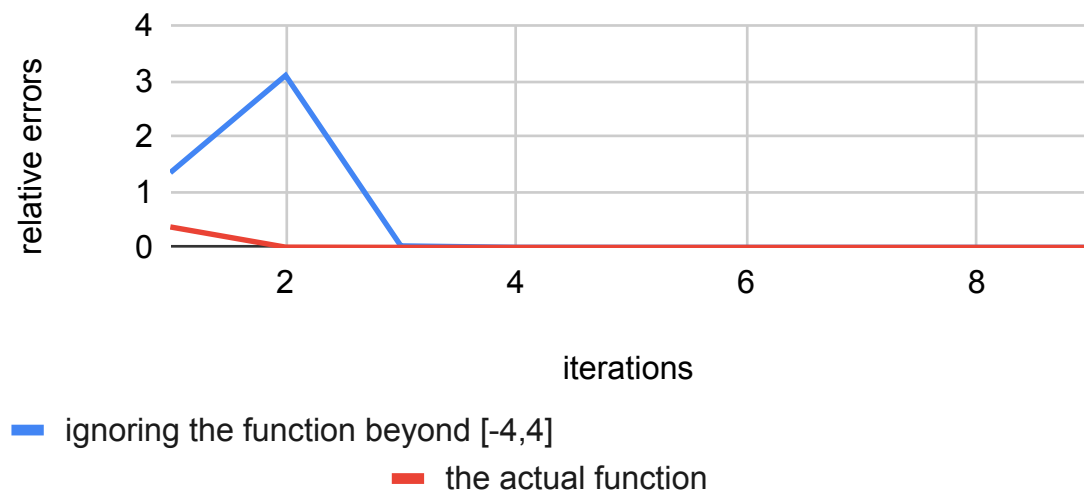


Figure 5

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.

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

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.

```
1  #include <stdio.h>
2  #include <math.h>
3  //function for integrand
4  double y(double x)
5  {
6      return pow(x, -x);
7  }
8  //function for midpoint
9  double midpoint(int n, double a, double b) //n: partitions, a and b are the
    ↪ limits of integration.
10 {
11     double sum=0; double width=(b-a)/n;
12     for(int i=0; i<n; i++)
13     {
14         sum+=y(a+(width*i)+(width/2))*width;
15     }
16     return sum;
17 }
18 //function for trapezoid
19 double trapezoid(int n, double a, double b) //n: partitions, a and b are the
    ↪ limits of integration.
20 {
21     double sum=0; double width=(b-a)/n;
22     for(int i=0; i<n; i++)
23     {
24         sum+=0.5*(y(a+width*i)+y(a+width*(i+1)))*width;
25     }
26     return sum;
27 }
28 //function for simpson
29 double simpson(int n, double a, double b) //n: partitions, a and b are the
    ↪ limits of integration.
```

```

30 {
31     double sum=0;double width=(b-a)/n;
32     for(int i=0;i<n/2;i++)
33     {
34         ↪ sum+=(width/3)*(y(a+2*i*width)+4*y(a+width*(2*i+1))+y(a+width*(2*i+2)));
35     }
36     return sum;
37 }
38 int main()
39 {
40     double sum=0;
41     for(int i=0;i<6;i++)
42     {
43         sum+=pow(i,-i);
44     }
45     double actual_value=sum; //the RHS of the equality
46
47     printf("printing values for midpoint method\n");
48     for(int i=1;i<11;i++)
49     {
50         double
51         ↪ actual_error=fabs((midpoint(pow(2,i),0,1)-actual_value)/(actual_value))*100;
52         //printf("iterations : %d ,actual error: %lf\n",i,actual_error);
53         printf("%lf\n",actual_error);
54     }
55     printf("\n\n");
56
57     printf("printing values for trapezoid method\n");
58     for(int i=1;i<11;i++)
59     {
60         double
61         ↪ actual_error=fabs((trapezoid(pow(2,i),0,1)-actual_value)/(actual_value))*100;
62         // printf("iterations : %d ,actual error: %lf\n",i,actual_error);
63         printf("%lf\n",actual_error);
64     }
65     printf("\n\n");
66
67     printf("printing values for simpson method\n");
68     for(int i=1;i<11;i++)
69     {
70         double
71         ↪ actual_error=fabs((simpson(pow(2,i),0,1)-actual_value)/(actual_value))*100;
72         //printf("iterations : %d ,actual error: %lf\n",i,actual_error);
73         printf("%lf\n",actual_error);
74     }
75     printf("\n\n");
76 }

```

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%

actual errors compared

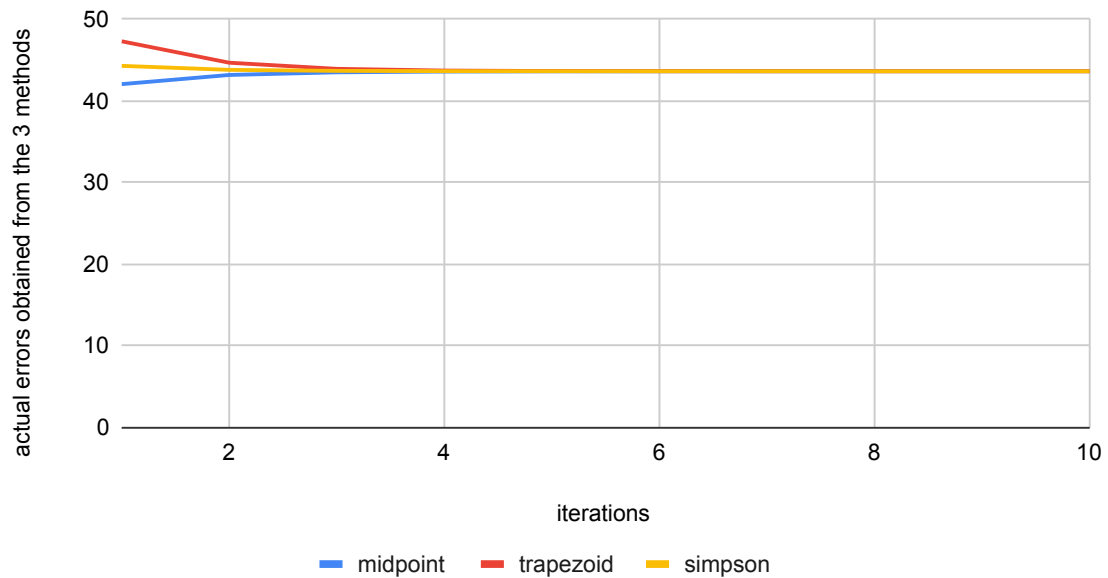


Figure 6

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.