Integration

Mitchell Miller

Physics 240

This assignment explores the properties and applications numerical integration methods. There are three different methods explored in this lab: the Trapezoid method, the Simpson method and the Gaussian Quadrature method. Each of these methods calculates the integral of a function with different summing methods. The Trapezoid method splits the area under the curve into a series of trapezoids and adds the approximate area. The Simpson method separates the area into a set of rectangular sections with parabolas at the top or bottom to match the curve. Finally, the Gaussian Quadrature method works by selecting specific points to separate the area around and weights them uniquely, giving a more dynamic area calculation to accurately calculate the integral in as few steps as possible, sometimes less than five. Each of these methods of integration will be explored using simple and complex integrals in this assignment.

1. Introduction

Integration has a massive involvement in scientific calculations. Integrals are used to solve countless problems and equations, but this can become very tedious or sometimes simply impossible to do by hand. Fortunately, there are several different methods for calculating integrals numerically with sums. Again, doing these methods by hand would also be quite tedious, however, simple computer programs can be written to perform these methods with great speed and accuracy. For both the Trapezoid and Simpson methods, increasing the divisions of area increases the accuracy. There is, however, a practical limit to this because when the number of areas becomes very large, the computing time gets very large as well. The Gaussian Quadrature method, on the other hand, provides very accurate results with very few iteration loops, which significantly decreases the computation time. This method can be much more difficult to implement with computer code due to the complex nature of the method of point mapping that it implements. These methods can be applied to different problems based on whether high accuracy or high speed is necessary, giving one a high degree of flexibility in problem solving.

2. Theory

The first task in this assignment was to find the number of particles entering a detector by integrating the following equation:

$$\int_0^1 e^{-t}dt = N(1) \qquad (1)$$

This integral can be solved analytically to find an answer to compare our numerical results to.

$$N(1) = 1 - e^{-1} = 0.63212055883\ldots \quad (2)$$

Using this analytic solution[1], the error of each method can be easily calculated.
The first method explored in this assignment is the Trapezoid method. This method takes values of the function at a set

interval and adds the area of the trapezoids formed by each region together to find an approximation for the total area[2]. For *N* points in a region beginning at *a* and ending at *b*, the length of the interval is the following:

$$h = \frac{b-a}{N-1} \tag{3}$$

for each trapezoid generated by this method the area is[3]

$$A = \frac{1}{2}(f_i + f_{i+1}) \tag{4}$$

By combining all these areas, one gets this approximation for the integral of a general function

$$\int_a^b f(x)dx = \frac{h}{2}f_1 + hf_2 + \cdots + hf_{N-1} + \frac{h}{2}f_N \tag{5}$$

It is clear from equation (5) that a large number of points means a large number of terms to generate and sum, which would cause slow performance.

The next method in this assignment is the Simpson method. This method works in a similar manner to the Trapezoid method, except the linear tops of the trapezoid section in the previous method are replaced with curved parabolas to match the function more accurately[4]. These parabolas are generated with the following approximation:

$$f(x) \approx \alpha x^2 + \beta x + \gamma \tag{6}$$

By integrating this approximation on an arbitrary region within our larger area from *x-h* to *x+h*, we find that the three constants can be related back to the original equation

$$\alpha = \frac{f(x+h) + f(x-h)}{2} \tag{7}$$

$$\beta = \frac{f(x+h) - f(x-h)}{2} \tag{8}$$

$$\gamma = f(x) \tag{9}$$

By integrating these values over two adjacent regions (*x-h – x,* and *x – x+h*) one arrives at the following approximation[5]:

$$\int_{x-h}^{x+h} f(x)dx \approx \frac{h}{3}f_{x-h} + \frac{4h}{3}f_x + \frac{h}{3}f_{x+h} \tag{10}$$

$$\int_a^b f(x)dx \approx \frac{h}{3}f_1 + \frac{4h}{3}f_2 + \frac{2h}{3}f_3 + \cdots + \frac{4h}{3}f_{N-1} + \frac{h}{3}f_N \tag{11}$$

Equation (11) is found by adding up all the areas created by equation (10). Since this method requires pairs of intervals, the total number of points *N* must be an odd number.

The final method of this assignment is the Gaussian Quadrature. This method is, by far, the most complex of the three but also the fastest and most accurate. This method works by choosing an ideal distribution of points of the region of integration[6]. By mapping and weighting each point individually, immense accuracy is achieved. The following equation is the general form for a Gaussian Quadrature method integral

$$\int_a^b f(x)dx = \int_a^b W(x)g(x)dx \approx \sum_{i=1}^{N} w_i g(x_i) \tag{12}$$

Where *W(x)* is the weighting function and *g(x)* is the mapped function.

3. Experimental Method

The first part of this lab seeks to find the integral in equation (1) using all three methods described. To do this, three separate functions were created. The first used the Trapezoid method. Using equation (5), an estimation for the integral was generated. The approximation in equation (11) was used to find the Simpson method integral. Finally, using the "gauss.c" program provided, the Gaussian Quadrature approximation was calculated. The precision in the "gauss.c" program was changed to $3x10^{-15}$.

Each of these three functions was run for an increasing number of points, from 1 to 5001. The error was calculated for each method and compared.

Finally, each integration method was applied to two more functions:

$$f_1 = \sin 1000x \, dx \qquad (13)$$
$$f_2 = \sin^x(100x)dx \qquad (14)$$

Both of these functions were integrated over the region from 0 to 2π. The error of the function in equation (13) was compared to $1x10^{-15}$.

4. Data and Analysis

The first task in this assignment is to numerically integrate equation (1). The three methods all successfully integrate that function.

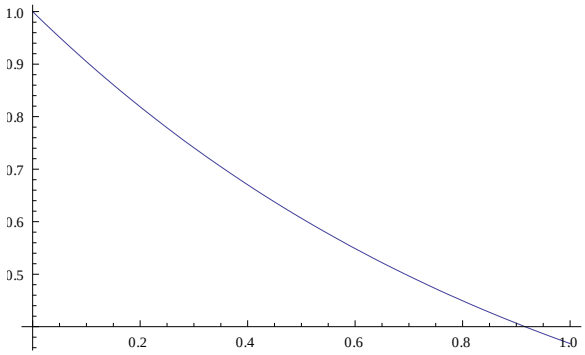


Figure 1. – This figure displays a simple plot of $f(x)=e^{-x}$ over the desired range of integration (0 to 1)

The errors of each method for this integral are displayed in Figure 2. This figure clearly illustrates the effectiveness of the Gaussian Quadrature method, with its error approaching $5x10^{-12}$ with less than 10 points. Both the Trapezoid and Simpson methods appear to approach a limit, but the number of points used in this experiment fail to reach it. This limit is unpractical, however, because the computation time for even 5000 points became quite large and based on the slope of the plot, in order to match the Quadrature error, the number of points must approach several million or more.

The final section of this lab seeks to integrate equations (13) and (14) in the region from 0 to 2π. Figure 3 and 4 display the plot of these functions. These functions proved to be more difficult to integrate numerically. The function in equation (13) has such a high frequency, that, without several thousand points or more, the shape is completely lost during the integration process. For the function in equation (14), the large discontinuities shown in Figure 4 cause the integration methods to fail completely, generating NAN's for all methods at any number of points. The error for the integration of equation (13) is

shown in Figure 5. The first thing to note when analyzing the error is the fact that the analytic solution is zero. This immediately causes problems, so the solutions were instead compared to $1x10^{-15}$. This fact could, potentially, cause the appearance of more error than there actually is, however it is important to compare the results to something. The error for this function is quite large compared to that of the first function. This could be reduced with a significant increase in the number of points used, however, as previously stated, the computation time will become increasingly long. It is possible that no matter how many points are used, a reasonably accurate result may not be found with these methods.

5. Conclusion

This lab successfully demonstrated the properties of numerical integration methods. First, the function e^{-x} was integrated using all three methods, the Trapezoid, Simpson, and Gaussian Quadrature methods. Next, two more complicate functions were integrated. These functions showed the limitations of these methods. It is clear from the results of this lab that the Gaussian Quadrature method is both the most accurate and most efficient method for integration.

6. Bibliography

[1]Wolfram Alpha. "1-e^-1." Web. <http://www.wolframalpha.com/input/?i=1-e^-1>

[2]"Trapezoidal - Quadrature - Calculus - Maths Code in C, C and Excel." *Open Source Scientific Library (C/C , .NET, Excel)*. Web. 26 Feb. 2010. <http://www.codecogs.com/d-ox/maths/calculus/quadrature/trapezoidal.php>.

[3]Wolfram Alpha. "Area of a Trapezoid." Web. <http://www.wolframalpha.com/input/?i=area+of+trapezoid>

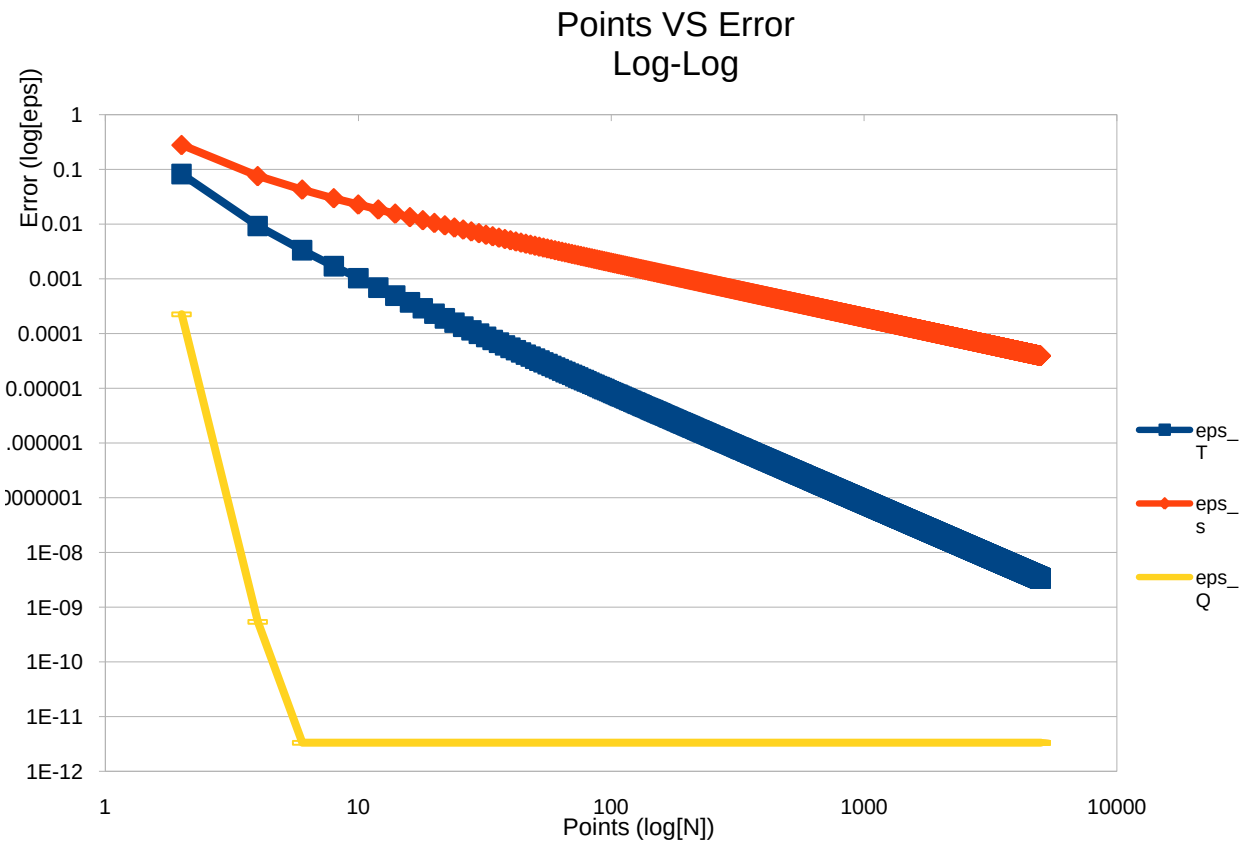
[4]"Numerical Integration." *Undergrad Mathematics*. Web. 26 Feb. 2010. <http://www.ugrad.math.ubc.ca/coursedoc/math101/notes/techniques/numerical.html>.

[5]"Numerical Integration: Intermediate." Web. 26 Feb. 2010. <http://spiff.rit.edu/classes/phys317/lectures/num_integ2/num_integ2.html>.

[6]"Gaussian Quadrature -- from Wolfram MathWorld." *Wolfram MathWorld: The Web's Most Extensive Mathematics Resource*. Web. 26 Feb. 2010. <http://mathworld.wolfram.com/GaussianQuadrature.html>.

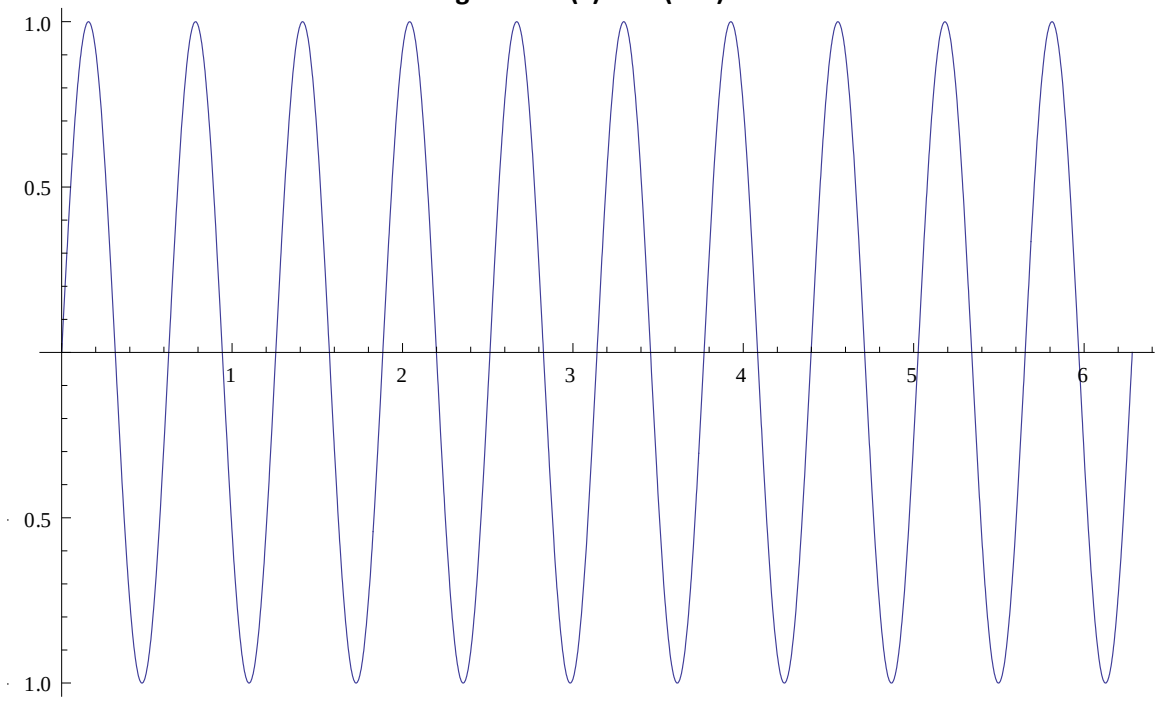
APPENDIX A: TABLES AND FIGURES

Figure 2 - Points VS Error



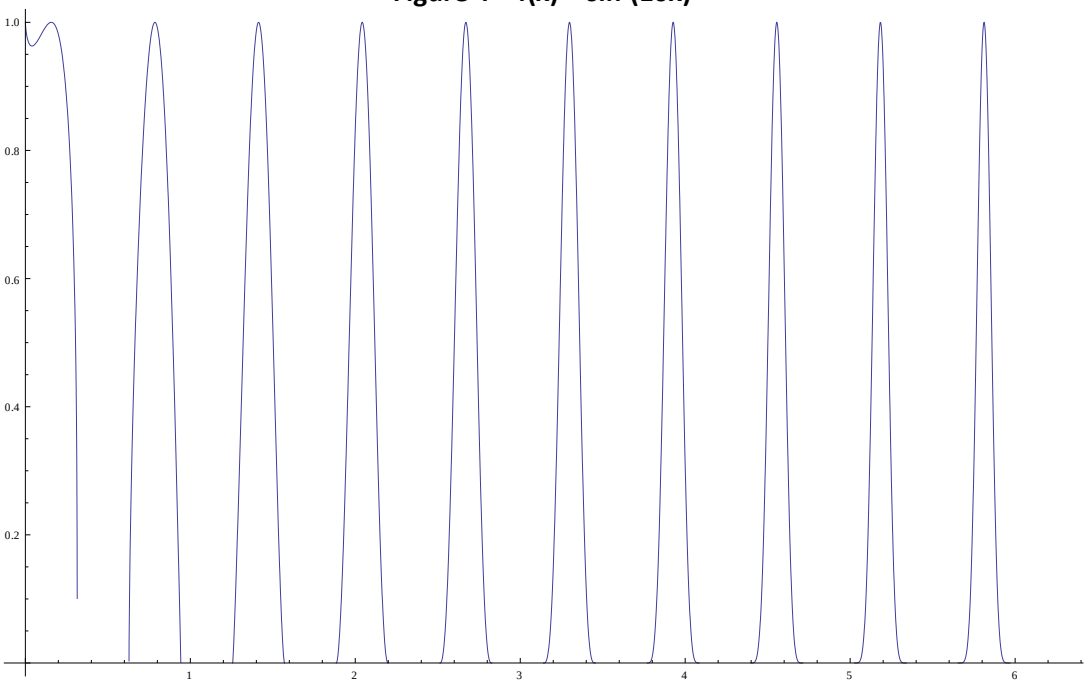
This figure displays a log-log scale plot of the error of each integration method versus the number of points for each method for the function in equation (1). The Gaussian Quadrature method clearly has the smallest accuracy and requires the fewest points.

Figure 3 – f(x) = sin(10x)



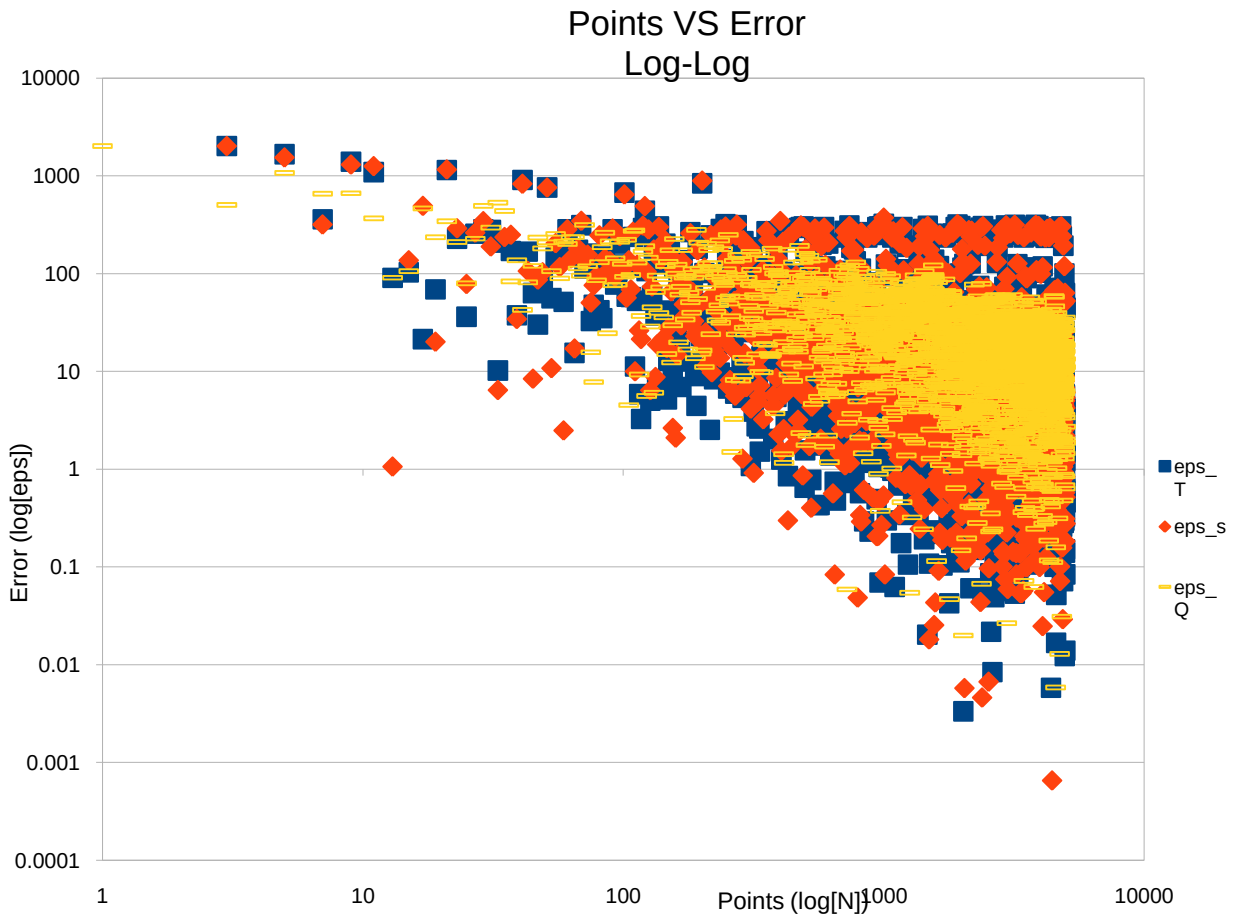
This figure displays the simple plot of sin(10x). Since increasing the value multiplied by x increases the frequency, the sin(1000x) is very difficult to view at small sizes, so this has been scaled to allow viewing.

Figure 4 – f(x) = $\sin^x(10x)$



This figure displays the plot of $\sin^x(10x)$. For the same reasons as above, this plot has been scaled down from the $\sin^x(100x)$ this is used in the assignment.

Figure 5 – Points VS Error



This figure shows error for each of the three integration methods versus the number of points for the function described in equation (13). It is clear that none of the methods are very effective for this function.

APENDIX B: SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gauss.c"
#define e 2.71828182845
#define pi 3.1415926535897932385

//Assignment 3
//Ch. 6.2.5 #1
//Mitchell Miller
//PHYS 240 2/26/10


double function(double x){
```

```
	//Specify your desired function
	return(pow(e,-x));
}

//		**TRAPEZOID METHOD**
//	|\ b		h					h
//	|  f(x)dx = ---f  + hf  + ... + hf    + ---f
//   \| a		2  1    2     N-1    2  N
//

double trapezoidIntegration(int loops,double start,double end){

	//Initialize function variables
	double sum=0,h=0,weight=0;
	int counter=0;

	h = (end-start)/(loops-1);

	//Perform trapezoid method integration sum
	for(counter=1; counter<=loops; counter++){

		if(counter==1 || counter==loops){
			weight=h/2;
		}
		else{
			weight=h;
		}

		sum += weight*function(start+(counter-1)*h);
	}

	return(sum);
}

//			**SIMPSON METHOD**
//	|\ b		h		4h	2h		4h	h
//	|  f(x)dx = ---f  + ----f  + ----f  + ... + ----f   + ---f
//   \| a		3  1    3  2    3  3		3  N-1   3  N
//

double simpsonIntegration(int loops,double start,double end){

	//Initialize function variables
	double sum=0,h=0,weight=0;
	int counter=0;

	h = (end-start)/(loops-1);
```

```
	//Perform simpson method integration sum
	for(counter=1; counter<=loops; counter++){

		if(counter==1 || counter==loops){
			weight = h/3;
		}
		if(counter!=1 && counter!=loops && counter%2==0){
			weight = 4*h/3;
		}
		if(counter!=1 && counter!=loops && counter%2==1){
			weight = 2*h/3;
		}

		sum += weight*function(start+(counter-1)*h);
	}

	return(sum);
}

//				**GAUSSIAN QUADRATURE METHOD**
//			N
//			---
//	|\ b		\	W * g( x )
//	|  f(x)dx =  /   i    i
//    \| a		---
//		i=1

double quadtratureIntegration(int loops, double start, double end){

	//Initialize variables
	int counter=0;
	double sum=0,*x,*weight;
	x=calloc(1100000,sizeof(double));
	weight=calloc(1100000,sizeof(double));

	//Generate points and weights with code provided
	gauss(loops,0,start,end,x,weight);

	//Sum integral with Gaussian Quadrature method
	for(counter=0; counter<loops; counter++){
		sum= sum + function(x[counter])*weight[counter];
	}

	return(sum);
}


int main(){
```

```
	//Initialize variables
	int loops;
	double trapSolution,simpSolution,quadSolution;
	double analyticSolution=(1-pow(e,-1));
	double trapError,simpError,quadError;
	FILE *outfile;
	outfile=fopen("625.2.error.txt","w");
	fprintf(outfile,"N\teps_T\teps_s\teps_Q\n");

	//Calculate integral for different numbers of loops
	for(loops=2;loops<5001;loops+=2){
		trapSolution = trapezoidIntegration(loops,0,1);
		simpSolution = simpsonIntegration(loops,0,1);
		quadSolution = quadtratureIntegration(loops,0,1);

		//Calculate error for given number of loops
		trapError=fabs((trapSolution-analyticSolution)/analyticSolution);
		simpError=fabs((simpSolution-analyticSolution)/analyticSolution);
		quadError=fabs((quadSolution-analyticSolution)/analyticSolution);

		//Print results
		printf("Trapezoid Method Solution: %.20lf\n",trapSolution);
		printf("Simpson Method Solution: %.20lf\n",simpSolution);
		printf("Quatrature Method Solution: %.20lf\n",quadSolution);
		printf("Analytic Solution: %.20lf\n",analyticSolution);
		printf("# of Loops = %d\n",loops);
		fprintf(outfile,"%i\t%.20lf\t%.20lf\t%.20lf\n",loops,trapError,simpError,quadError);
	}

	fclose(outfile);
}



#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "gauss.c"
#define e 2.71828182845
#define pi 3.1415926535897932385

//Assignment 3
//Ch. 6.2.5 #1
//Mitchell Miller
//PHYS 240 2/26/10
// SOURCES: http://www.codecogs.com/d-ox/maths/calculus/quadrature/trapezoidal.php
// SOURCES: http://mathworld.wolfram.com/GaussianQuadrature.html
```

```
double function(double x){
	//Specify your desired function
	return(sin(1000*x));
	//return(pow(sin(100*x),x));
}

//		**TRAPEZOID METHOD**
//	|\ b	h					h
//	|  f(x)dx = ---f  + hf  + ... + hf    + ---f
//    \| a	2  1    2     N-1    2  N
//

double trapezoidIntegration(int loops,double start,double end){

	//Initialize function variables
	double sum=0,h=0,weight=0;
	int counter=0;

	h = (end-start)/(loops-1);

	//Perform trapezoid method integration sum
	for(counter=1; counter<=loops; counter++){

		if(counter==1 || counter==loops){
			weight=h/2;
		}
		else{
			weight=h;
		}

		sum += weight*function(start+(counter-1)*h);
	}

	return(sum);
}

//			**SIMPSON METHOD**
//	|\ b	h	4h	2h	4h	h
//	|  f(x)dx = ---f  + ----f  + ----f  + ... + ----f   + ---f
//    \| a	3  1    3  2    3  3	3  N-1   3  N
//

double simpsonIntegration(int loops,double start,double end){

	//Initialize function variables
	double sum=0,h=0,weight=0;
	int counter=0;
```

```
	h = (end-start)/(loops-1);

	//Perform simpson method integration sum
	for(counter=1; counter<=loops; counter++){

		if(counter==1 || counter==loops){
			weight = h/3;
		}
		if(counter!=1 && counter!=loops && counter%2==0){
			weight = 4*h/3;
		}
		if(counter!=1 && counter!=loops && counter%2==1){
			weight = 2*h/3;
		}

		sum += weight*function(start+(counter-1)*h);
	}

	return(sum);
}

//				**GAUSSIAN QUADRATURE METHOD**
//			N
//			---
//	|\ b		\	W * g( x )
//	|  f(x)dx =	/	i	i
//   \| a		---
//		i=1

double quadtratureIntegration(int loops, double start, double end){

	//Initialize variables
	int counter=0;
	double sum=0,*x,*weight;
	x=calloc(1100000,sizeof(double));
	weight=calloc(1100000,sizeof(double));

	//Generate points and weights with code provided
	gauss(loops,0,start,end,x,weight);

	//Sum integral with Gaussian Quadrature method
	for(counter=0; counter<loops; counter++){
		sum= sum + function(x[counter])*weight[counter];
	}

	return(sum);
}
```

```
int main(){
	//Initialize variables
	int loops;
	double trapSolution,simpSolution,quadSolution;
	double analyticSolution=(1.0e-15);
	double trapError,simpError,quadError;
	FILE *outfile;
	outfile=fopen("625.3.error2.txt","w");
	fprintf(outfile,"N\teps_T\teps_s\teps_Q\n");

	//Calculate integral for different numbers of loops
	for(loops=2;loops<5001;loops+=2){
		trapSolution = trapezoidIntegration(loops,0,2*pi);
		simpSolution = simpsonIntegration(loops,0,2*pi);
		quadSolution = quadtratureIntegration(loops,0,2*pi);

		//Calculate error for given number of loops
		trapError=fabs((trapSolution-analyticSolution)/analyticSolution);
		simpError=fabs((simpSolution-analyticSolution)/analyticSolution);
		quadError=fabs((quadSolution-analyticSolution)/analyticSolution);

		//Print results
		printf("Trapezoid Method Solution: %.20lf\n",trapSolution);
		printf("Simpson Method Solution: %.20lf\n",simpSolution);
		printf("Quatrature Method Solution: %.20lf\n",quadSolution);
		printf("Analytic Solution: %.20lf\n",analyticSolution);
		printf("# of Loops = %d\n",loops);
		fprintf(outfile,"%i\t%.20lf\t%.20lf\t%.20lf\n",loops,trapError,simpError,quadError);
	}

	fclose(outfile);
}


/*    From: "A SURVEY OF COMPUTATIONAL PHYSICS"
  by RH Landau, MJ Paez, and CC BORDEIANU
  Copyright Princeton University Press, Princeton, 2008.
  Electronic Materials copyright: R Landau, Oregon State Univ, 2008;
  MJ Paez, Univ Antioquia, 2008; & CC Bordeianu, Univ Bucharest, 2008
  Support by National Science Foundation
*/
// gauss.c: Points and weights for Gaussian quadrature
// comment: this file has to reside in the same directory as integ.c
#include <math.h>
void gauss(int npts, int job, double a, double b,
                              double x[], double w[]) {
```

```
//    npts     number of points
//    job= 0  rescaling uniformly between (a,b)
//         1  for integral (0,b) with 50% pts inside (0, ab/(a+b)
//         2  for integral (a,inf) with 50% inside (a,b+2a)
//    x, w     output grid points and weights.

  int     m, i, j;
  double  t, t1, pp, p1, p2, p3;
  double  pi= 3.14159265358979323 85E0, eps= 3.e-15;
                                        // eps= accuracy to adjust
  m= (npts+1)/2;
     for (i = 1; i<= m; i++) {
       t = cos(pi*(i-0.25)/(npts+0.5));
       t1= 1;
       while((fabs(t-t1))>= eps) {
         p1= 1.0;
         p2= 0.0;
         for (j = 1; j<= npts; j++) {
           p3= p2;
           p2= p1;
           p1= ((2*j-1)*t*p2-(j-1)*p3)/j;
         }
         pp= npts*(t*p1-p2)/(t*t-1);
         t1= t;
         t = t1 - p1/pp;
       }
       x[i-1]= -t;
       x[npts-i]= t;
       w[i-1]   = 2.0/((1-t*t)*pp*pp);
       w[npts-i]= w[i-1];
   }
   if (job == 0)   {
     for (i = 0; i<npts ; i++)  {
          x[i]= x[i]*(b-a)/2.0+(b+a)/2.0;
          w[i]= w[i]*(b-a)/2.0;
     }
   }
     if (job == 1)  {
       for (i = 0; i<npts; i++)  {
         x[i]= a*b*(1+x[i]) / (b+a-(b-a)*x[i]);
         w[i]= w[i]*2*a*b*b /((b+a-(b-a)*x[i])*(b+a-(b-a)*x[i]));
       }
   }
     if (job == 2)   {
       for (i = 0; i<npts; i++)  {
         x[i]= (b*x[i]+b+a+a) / (1-x[i]);
         w[i]=  w[i]*2*(a+b)  /((1-x[i])*(1-x[i]));
```

```
    }
  }
}
```