Comp. Methods in Mech. Eng.

MCG 4127

Assignment # 5

uOttawa

Name: Usama Tariq

Student Number: 7362757

Professor: Catherine Mavriplis

TA: Fabien Giroux

For this assignment I used Simpsons numerical integration method and Gauss Quadrature to compare results and observe which method requires the least number of iterations to converge to the solution.

First, we shall explore Simpsons method which has the general equation

$$\int_a^b f(x)dx \approx \frac{\Delta x}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \cdots 4f(x_{n-1}) + f(x_n)\right)$$

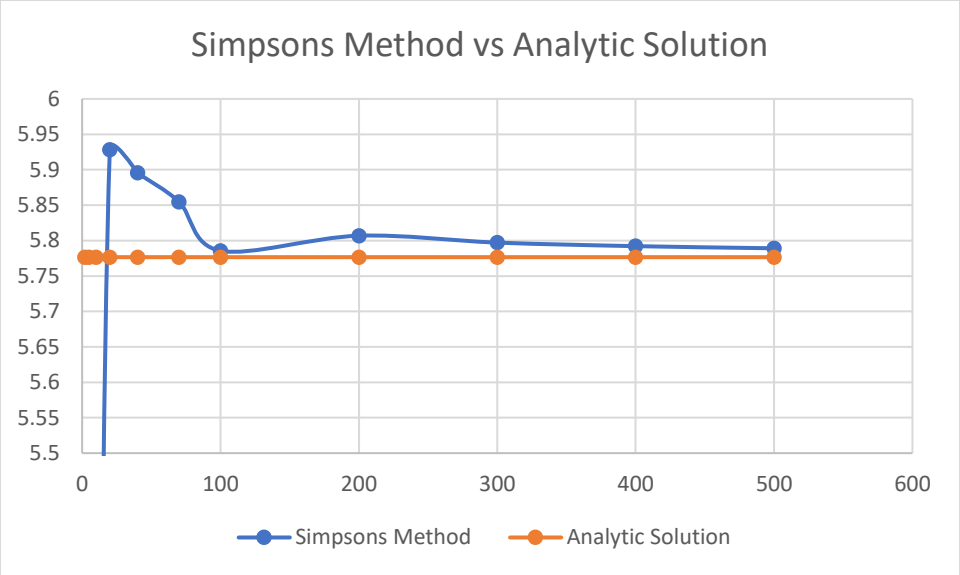
Where $\Delta x = \frac{b-a}{n}$ and $x_i = a + i\Delta x$

A test integral was initially studied to test the validity of both methods by comparing it to the analytic solution. After programming both methods in C++, the following results are obtained.

The test integral used was $\int_0^3 x^2 \sin(x)\, dx$

n	Simpsons Method	Analytic Solution	Percent Error (%)
2	-16.6198	5.7766	387.709
5	1.62235	5.7766	71.91514
10	4.21819	5.7766	26.97798
20	5.92829	5.7766	2.625939
40	5.89581	5.7766	2.063671
70	***5.85471***	***5.7766***	***1.352179***
100	5.78564	5.7766	0.156493
200	5.80689	5.7766	0.524357
300	5.79716	5.7766	0.355919
400	5.79216	5.7766	0.269363
500	5.78913	5.7766	0.21691

As can be seen in the table above around 70 iterations are required to reach a percent error of 1%.



Now we can compare Simpsons Method with Gaussian Quadrature which is represented by

$$\int_a^b f(x)dx = \sum_{i=1}^{\infty} w_i f(x_i) \approx \sum_{i=1}^{n} w_i f(x_i)$$

To use this method, the integration has to be converted such that it can be defined on the interval $[-1,1]$. This can be done by the following equation:

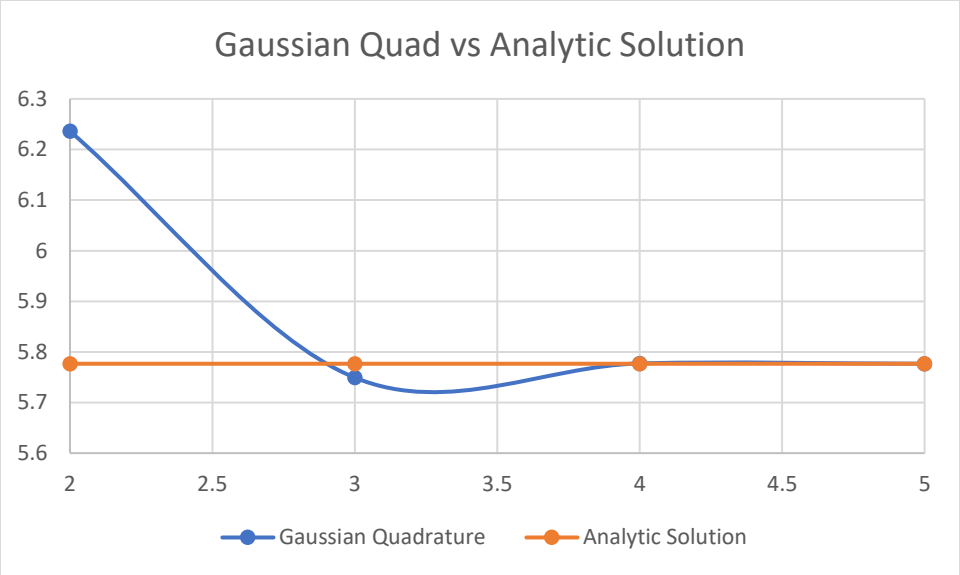
$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^{1} f\left(\frac{b-a}{2}x_i + \frac{b+a}{2}\right)dx$$

So, in the case of our test integral, now it takes the following form

$$\int_0^3 x^2 \sin(x)dx = 1.5 \int_{-1}^{1} (1.5x + 1.5)^2 \sin(1.5x + 1.5)dx$$

The weights of the Gauss Quadrature were hardcoded in the C++ program and then studied; the results are as follows

n	Gaussian Quadrature	Analytic Solution	Percent Error (%)
2	6.23612	5.7766	7.954852
3	5.74918	5.7766	0.474674
4	5.77723	5.7766	0.010906
5	5.77666	5.7766	0.001039



It is quite evident that Gaussian Quadrature requires far less iterations to get a very accurate solution, with the 5th giving a percent error of 0.

Heat Transfer

The total number of heat added in a 1kg of material from temperature range of -100 to 200 C. The general heat equation is

$$Q = Cm\Delta T$$

The value of the heat capacity is a function of temperature and in order to find the total heat, we compute the following integral of the heat capacity

$$\int C(T)dT = \int 0.132 + 1.56 \times 10^{-4}T + 2.64 \times 10^{-7}T^2\, dT$$

$$Q = (1kg)(300\ K)(\int 0.132 + 1.56 \times 10^{-4}T + 2.64 \times 10^{-7}T^2\, dT)$$

Using both Simpsons method (1000 iterations) we find that the heat required is 42.73 kJ and using Gaussian Quadrature (5 iterations) the result comes out to be 48.99. Since the Gaussian Quadrature is more accurate, that result is more trustworthy.

Code

```cpp
#include <iostream>
#include <cmath>
#include <vector>
#include <fstream>
#include <math.h>
#include <functional>
#include <ctime>
#include <chrono>

using namespace std;
// Using typedefs for all functions
using function_type = function<double(double)>;

auto gauss_weights(int n){
  vector<pair<double,double>> w;
  if (n == 2){
    w.push_back(make_pair(-0.5773502691896257, 1.0000000000000000));
    w.push_back(make_pair(0.5773502691896257, 1.0000000000000000));
  }

  if (n == 3){
    w.push_back(make_pair(0.0000000000000000, 0.8888888888888888));
    w.push_back(make_pair(-0.7745966692414834, 0.5555555555555556));
    w.push_back(make_pair(0.7745966692414834, 0.5555555555555556));
  }

  if (n == 4){
    w.push_back(make_pair(-0.3399810435848563, 0.6521451548625461));
    w.push_back(make_pair(0.3399810435848563, 0.6521451548625461));
    w.push_back(make_pair(-0.8611363115940526, 0.3478548451374538));
    w.push_back(make_pair(0.8611363115940526, 0.3478548451374538));
  }

  if (n == 5){
    w.push_back(make_pair(0.0000000000000000, 0.5688888888888889));
    w.push_back(make_pair(-0.5384693101056831, 0.4786286704993665));
    w.push_back(make_pair(0.5384693101056831, 0.4786286704993665));
    w.push_back(make_pair(-0.9061798459386640, 0.2369268850561891));
    w.push_back(make_pair(0.9061798459386640, 0.2369268850561891));
  }

  if (n == 6){
    w.push_back(make_pair(0.6612093864662645, 0.3607615730481386));
    w.push_back(make_pair(-0.6612093864662645, 0.3607615730481386));
    w.push_back(make_pair(-0.2386191860831969, 0.4679139345726910));
    w.push_back(make_pair(0.2386191860831969, 0.4679139345726910));
    w.push_back(make_pair(-0.9324695142031521, 0.1713244923791704));
    w.push_back(make_pair(0.9324695142031521, 0.1713244923791704));
  }

  if (n == 7){
    w.push_back(make_pair(0.0000000000000000, 0.4179591836734694));
    w.push_back(make_pair(0.4058451513773972, 0.3818300505051189));
    w.push_back(make_pair(-0.4058451513773972, 0.3818300505051189));
    w.push_back(make_pair(-0.7415311855993945, 0.2797053914892766));
    w.push_back(make_pair(0.7415311855993945, 0.2797053914892766));
    w.push_back(make_pair(-0.9491079123427585, 0.1294849661688697));
    w.push_back(make_pair(0.9491079123427585, 0.1294849661688697));
```

```
59.   }
60.
61.   if (n == 8){
62.     w.push_back(make_pair(-0.1834346424956498, 0.3626837833783620));
63.     w.push_back(make_pair(0.1834346424956498, 0.3626837833783620));
64.     w.push_back(make_pair(-0.5255324099163290, 0.3137066458778873));
65.     w.push_back(make_pair(0.5255324099163290, 0.3137066458778873));
66.     w.push_back(make_pair(-0.7966664774136267, 0.2223810344533745));
67.     w.push_back(make_pair(0.7966664774136267, 0.2223810344533745));
68.     w.push_back(make_pair(-0.9602898564975363, 0.1012285362903763));
69.     w.push_back(make_pair(0.9602898564975363, 0.1012285362903763));
70.   }
71.
72.   if (n == 9){
73.     w.push_back(make_pair(0.0000000000000000, 0.3302393550012598));
74.     w.push_back(make_pair(-0.8360311073266358, 0.1806481606948574));
75.     w.push_back(make_pair(0.8360311073266358, 0.1806481606948574));
76.     w.push_back(make_pair(-0.9681602395076261, 0.0812743883615744));
77.     w.push_back(make_pair(0.9681602395076261, 0.0812743883615744));
78.     w.push_back(make_pair(-0.3242534234038089, 0.3123470770400029));
79.     w.push_back(make_pair(0.3242534234038089, 0.3123470770400029));
80.     w.push_back(make_pair(-0.6133714327005904, 0.2606106964029354));
81.     w.push_back(make_pair(0.6133714327005904, 0.2606106964029354));
82.   }
83.
84.   if (n == 10){
85.     w.push_back(make_pair(-0.1488743389816312, 0.2955242247147529));
86.     w.push_back(make_pair(0.1488743389816312, 0.2955242247147529));
87.     w.push_back(make_pair(-0.4333953941292472, 0.2692667193099963));
88.     w.push_back(make_pair(0.4333953941292472, 0.2692667193099963));
89.     w.push_back(make_pair(-0.6794095682990244, 0.2190863625159820));
90.     w.push_back(make_pair(0.6794095682990244, 0.2190863625159820));
91.     w.push_back(make_pair(-0.8650633666889845, 0.1494513491505806));
92.     w.push_back(make_pair(0.8650633666889845, 0.1494513491505806));
93.     w.push_back(make_pair(-0.9739065285171717, 0.0666713443086881));
94.     w.push_back(make_pair(0.9739065285171717, 0.0666713443086881));
95.   }
96.
97.   return w;
98.
99. }
100.
101.       double simposons_method (function_type f, double a, double b, double del_x, doub
    le n){
102.         double sum = f(a);
103.
104.         for (int i = 0; i < n+1 ; i++){
105.
106.           a = a + del_x;
107.
108.           if (((i-1)%2)+1 == 1){
109.             sum = sum + 4*f(a);
110.           }
111.
112.           if (((i-1)%2)+1 == 2){
113.             sum = sum + 2*f(a);
114.           }
115.
116.         }
117.
118.         sum = sum + f(b);
```

```
119.         return (del_x/3)*sum;
120.       }
121.
122.       double guass_quad (function_type f, double iter){
123.         double sum = 0;
124.
125.         auto w = gauss_weights(iter);
126.
127.         for (int i = 0; i < iter; i++){
128.           sum = sum + f(w[i].first)*w[i].second;
129.         }
130.
131.         sum = sum*150;
132.         return sum;
133.
134.       }
135.
136.
137.       int main() {
138.
139.         // The constants
140.         double a = -100;
141.         double b = 200;
142.         double n = 9000;
143.         double del_x = (b-a)/n;
144.         double num_of_iter = 5;
145.
146.
147.           // Functions for temprature
148.           auto c = [](double t){
149.             return 0.132 + 1.56*pow(10,-4)*t + 2.64*pow(10,-7)*pow(t,2);
150.           };
151.
152.           auto c_gauss = [](double t1){
153.             return 0.132 + 1.56*pow(10,-4)*(150*t1+150) + 2.64*pow(10,-
    7)*pow( (150*t1 + 150), 2);
154.           };
155.
156.           // auto f = [](double x){
157.           //   return pow(x,2)*sin(x);
158.           // };
159.
160.           // auto f_gauss = [](double x1){
161.           //   return pow((1.5*x1 + 1.5), 2)*sin((1.5*x1 + 1.5));
162.           // };
163.
164.           cout << "-----------------------------------------------------\n";
165.           cout << "-----------------------------------------------------\n";
166.           cout << "|                  SIMPSONS METHOD                  |\n";
167.           cout << "-----------------------------------------------------\n";
168.           cout << "-----------------------------------------------------\n";
169.
170.           double ans = simposons_method(c, a, b, del_x, n);
171.
172.           cout << ans << '\n';
173.
174.           cout << "-----------------------------------------------------\n";
175.           cout << "-----------------------------------------------------\n";
176.           cout << "|             GUASS-QUADRATURE METHOD               |\n";
177.           cout << "-----------------------------------------------------\n";
178.           cout << "-----------------------------------------------------\n";
```

```
179.
180.           double ans_2 = guass_quad(c_gauss, num_of_iter);
181.           cout << ans_2 << '\n';
182.
183.       }
```