Project 1

SF2568 Program construction in C++ for Scientific Computing

Aron Andersson aronande@kth.se Harsha HN harshahn@kth.se

Task 1 - Taylor Expansion using Horner's Scheme

Taylor series of the sine function and the cosine function are shown as below

$$sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

$$cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}$$

These equations are computationally expensive, thus the sums are computed using Horner's scheme. Horner's method uses only n multiplications and n additions, therefore the calculations can be carried-out nominally 13 times faster[1].

$$p(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_n x^n$$

$$p(x) = c_0 + x(c_1 + x(c_2 + \cdots (c_{n-1} + x c_n)))$$

In order to compute sine and cosine using this method, corresponding coefficients need to be computed. These coefficients involve calculating $(-1)^n$ and factorial. Firstly, the $(-1)^n$ is computed using modulo concept, which saved the expensive pow() operation. Secondly, the factorials are avoided by implementing them as a multiplying factor within Horner's method i.e., using just the index at each c_i. These are implemented in three files namely main.cpp, taylor.cpp, taylor.hpp. Results are shown in the below table namely the absolute difference between the cmath implementation and the Taylor's expansion.

N	func	$x = -1$	$x = 1$	$x = 2$	$x = 3$	$x = 5$	$x = 10$
5	sin	2.48e-08	2.48e-08	5.00e-05	0.00419	1.04855	1448.82
5	cos	2.73e-07	2.73e-07	0.00027	0.01521	2.24474	1459.78
10	sin	0	0	4.08e-14	2.01e-10	8.89e-06	16.2678
10	cos	0	0	4.27e-13	1.40e-09	3.71e-05	33.5995
50	sin	0	0	0	1.38e-16	3.33e-16	2.02e-13
50	cos	0	0	5.55e-17	2.22e-16	8.32e-16	2.16e-13
100	sin	0	0	0	1.38e-16	3.33e-16	2.02e-13
100	cos	0	0	5.55e-17	2.22e-16	8.32e-16	2.16e-13

Table 1: Absolute difference between cmath implementation and Taylor's series over a range of x (angles) and N terms using Horner's method.

Based on the observation from the table 1, we conclude that for more number of terms(N) we have accurate results regardless of the value of x. However, for less number of terms(N) we notice significant error margin even for x values as low as 3.

Task 2 - Adaptive Simpson Integration (ASI)

ASI function takes four arguments. They are function pointer, limits of the integral and the error tolerance. ASI function is a recursive evaluation of the integral using the Simpson rule until the acceptable observed error. ASI function is implemented as per the given algorithm and definition. Simpson integral takes the function pointer and the integral limits.

The source code for task 2 is implemented in three files namely *main.cpp, asi.cpp* and *asi.hpp*. *asi.hpp* is a header file intended for abstraction wherein function pointer, Simpson rule and ASI function are declared. *asi.cpp* holds the definition of these two functions. *main.cpp* is comprises of user function, allowed error tolerances and the user interface.

ASI recursively splits the integral interval into 2 subintervals and computes using Simpson rule until the error tolerance is met. Simpson rule is implemented as per its definition.
It is defined as below

$$I(a,b) = \frac{b-a}{6}\left(f(a) + 4f(\frac{a+b}{2}) + f(b) \right)$$

Function to be evaluated is $\int_{-1}^{1}(1 + sin\ e^{3x})dx$

Matlab integral function *vpaintegral* [2] with a tolerance of 10^{-8} yields the value of *2.5008091*.

```
syms x;
vpaintegral((1+ sin(exp(3*x))), [-1 1], 'RelTol', 1e-8, 'AbsTol', 0);
```

Range of tolerance considered are $\{10^{-2}, 10^{-3}, 10^{-4}\}$. The results of our implementation is compared against matlab integral function with tolerance of 1e-8 and shown below.

tolerance	ASI value	error w.r.t matlab func
0.01	2.505996157	0.00520
0.001	2.499856554	9.525e-4
0.0001	2.500809667	5.670e-7

Table 2: Absolute difference between ASI implementation and matlab implementation for a set of tolerance value.

As observed from the table, we can deduce that integral value is more accurate with the finer tolerance value. This can be validated from comparison with the matlab integral function.

References:

1. https://en.wikipedia.org/wiki/Horner%27s_method (last visited 22 sept 2019)
2. https://se.mathworks.com/help/symbolic/vpaintegral.html#d117e233229 (last visited 28 sept 2019)