
SF2565 - Assignement I

Author:

Cecilia Battinelli 950904-0722

cbat@kth.se

Sakina Huseynzade 960201-0143

sakinah@kth.se

September 25, 2018

Contents

1	Task 1	2
1.1	Task 1 Background and Problem formulation	2
1.2	Task 1 Implementation Results	5
2	Task 2	8
2.1	Task 2 Background and Problem formulation	8
2.2	Task 2 Implementation Results	9
A	Task 1	11
B	Task 2	14

1 Task 1

1.1 Task 1 Background and Problem formulation

The task is to formulate the Taylor expansions of $\cos x$ and $\sin x$ in $C++$ using Horner's Scheme to evaluate the polynomial, hence without explicitly using neither the factorial and power functions. Later, for some given x and N the polynomials are evaluated and compared with the mathematical in-built functions values for the sine and cosine from `cmath` library. Then, considerations about accuracy are made.

The Horner's scheme allows a reduction of operations that have to be performed. Furthermore, the grouping in this case is facilitated thanks to the following factorial property. Note that the basic property $n! = n(n-1)!$ is used.

$$\frac{1}{n!} + \frac{1}{(n+2)!} = \frac{1}{n!} + \frac{1}{(n+2)(n+1)!} = \frac{1}{n!} + \frac{1}{(n+2)(n+1)n!} = \frac{1}{n!} \left(1 + \frac{1}{(n+2)(n+1)} \right) \quad (1)$$

The general Horner's scheme can be written as follows: ¹

General polynomial

$$P_N(x) = a_0 + a_1x + a_2x^2 + \dots + a_{N-1}x^{N-1} + a_Nx^N$$

Horner's scheme polynomial

$$P_N(x) = a_0 + x(a_1 + x(a_2 + \dots + x(a_{N-1} + a_Nx)))$$

The general expressions of Taylor series are given by:

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots + (-1)^n \frac{x^{2n+1}}{(2n+1)!} \quad (2)$$

$$\cos(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} \quad (3)$$

By Horner's scheme, equations (2) and (3) - thanks to property in eq. (1) - can be written as follows:

$$\sin(x) = x - \frac{x^3}{3!} \left(1 - \frac{x^2}{5 \cdot 4} \left(1 - \dots \left(1 - \frac{x^2}{(2n+1)(2n)} \right) \right) \right) \quad (4)$$

¹Wikipedia - Horner's Method

$$\cos(x) = 1 - \frac{x^2}{2!} \left(1 - \frac{x^2}{5 * 4} \left(1 - \dots \left(1 - \frac{x^2}{(2n)(2n-1)} \right) \right) \right) \quad (5)$$

We can see that terms Taylor expansions of $\sin(x)$ and $\cos(x)$ can be grouped by using (1) and Horner's scheme polynomial representation. Thus, to evaluate coefficients in of polynomial we go backwards, i.e:

For $\cos x$:

$$b_N = 1 - \frac{x \cdot x}{2N(2N-1)}$$

$$b_n = 1 - \frac{x \cdot x}{2n(2n-1)} b_{n+1}, \quad n = N-1, N-2, \dots, 2, 1$$

For $\sin x$:

$$b_N = 1 - \frac{x \cdot x}{(2N+1)(2N)}$$

$$b_n = 1 - \frac{x \cdot x}{(2n+1)(2n)} b_{n+1}, \quad n = N-1, N-2, \dots, 2, 1$$

A remark is that, if property shown in equation (1) was not used, than the coefficient a would stil have a factorial function as denominator. Thus, this property allows a more efficient grouping.

The Taylor expansions are implemented in `double SinTaylor(int N, double x)` and `double CosTaylor(int N, double x)`.

Thus, the Horner's scheme together with factorial property led to this code formulation:

```
double sinTaylor(int N, double x){  
    int i;  
    double sum = 1;  
    for (i = N ; i > 0; i--) {  
        sum = 1 - x*x*sum/(double)(2*i*(2*i+1));  
    }  
    sum = x*sum;  
    return sum;  
}
```

Figure 1: Code snippet showing how the Taylor series function is implemented. This is also valid for the cosine with the exception of the line before `return`. This is because the polynomial in cosine has even powers whereas the sine has odd powers.

1.2 Task 1 Implementation Results

Performance of the Taylor series implementation shown for selected values of x and N . The results are given below in Table 1. The `realsin` and `realcos` are the `sin(x)` and `cos(x)` functions from `<cmath>` library, while `sinTaylor(N,x)` and `cosTaylor(N,x)` are the functions results with implemented algorithms of Taylor expansion.

Table 2 shows the error and absolute value for $(N + 1) - st$ term for both `cos(x)` and `sin(x)` functions.

Table 1: Taylor series implementation

x	N	realsin	sinTaylor(N,x)	realcos	cosTaylor(N,x)
-1	1	-0.841471	-0.833333	0.540302	0.5
1	1	0.841471	0.833333	0.540302	0.5
2	1	0.909297	0.666667	-0.416147	-1
3	1	0.14112	-1.5	-0.989992	-3.5
5	1	-0.958924	-15.8333	0.283662	-11.5
10	1	-0.544021	-156.667	-0.839072	-49
-1	5	-0.841471	-0.841471	0.540302	0.540302
1	5	0.841471	0.841471	0.540302	0.540302
2	5	0.909297	0.909296	-0.416147	-0.416155
3	5	0.14112	0.140875	-0.989992	-0.991049
5	5	-0.958924	-1.13362	0.283662	-0.162747
10	5	-0.544021	-1056.94	-0.839072	-1296.8
-1	10	-0.841471	-0.841471	0.540302	0.540302
1	10	0.841471	0.841471	0.540302	0.540302
2	10	0.909297	0.909297	-0.416147	-0.416147
3	10	0.14112	0.14112	-0.989992	-0.989992
5	10	-0.958924	-0.958924	0.283662	0.283664
10	10	-0.544021	2.76109	-0.839072	6.66456
-1	20	-0.841471	-0.841471	0.540302	0.540302
1	20	0.841471	0.841471	0.540302	0.540302
2	20	0.909297	0.909297	-0.416147	-0.416147
3	20	0.14112	0.14112	-0.989992	-0.989992
5	20	-0.958924	-0.958924	0.283662	0.283662
10	20	-0.544021	-0.544021	-0.839072	-0.839072

Table 2: Errors and $(N + 1) - st$ term absolute value in implementation

x	N	SinError	AbsValue for sin	cosError	AbsValue for cos
-1	1	0.00813765	0.00833333	0.0403023	0.0416667
1	1	0.008137651	0.00833333	0.0403023	0.0416667
2	1	0.242631	0.266667	0.583853	0.666667
3	1	1.64112	2.025	2.51001	3.375
5	1	14.8744	26.0417	11.7837	26.0417
10	1	156.123	833.333	48.1609	416.667
-1	5	1.59828e-10	1.6059e-10	2.07625e-09	2.08768e-09
1	5	1.59828e-10	1.6059e-10	2.07625e-09	2.08768e-09
2	5	1.29086e-06	1.31556e-06	8.36627e-06	8.55112e-06
3	5	0.000245414	0.0002560335	0.00105661	0.00110948
5	5	0.174693	0.196033	0.446409	0.509686
10	5	1056.4	1605.9	1295.96	2087.68
-1	10	0	0	0	0
1	10	0	0	0	0
2	10	2.22045e-16	2.22045e-16	3.83027e-15	3.9968e-15
3	10	3.5876e-12	3.64175e-12	2.74702e-11	2.7919e-11
5	10	4.42572e-07	4.61122e-07	2.02867e-06	2.12116e-06
10	10	3.30511	3.86817	7.50364	8.89679
-1	20	0	0	0	0
1	20	0	0	0	0
2	20	0	0	1.66533e-16	0
3	20	1.38778e-16	0	2.22045e-162	0
5	20	1.88738e-15	0	7.77156e-16	0
10	20	1.57616e-10	1.65961e-10	6.75881e-10	7.11792e-10

As illustrated in Table 2, mostly all errors are bounded by the $(N + 1) - st$ term in the corresponding Taylor series. Since the bounding is verified using the inequality $< \text{sign}$, when both error and the $(N + 1) - st$ term are null then the bounding is not properly verified.

As we see from Table 2 errors $|\text{real sin}(x) - \text{sinTaylor}(N, x)|$ and $|\text{real cos}(x) - \text{cosTaylor}(N, x)|$ are **not bounded** by $(N + 1) - st$ term in the corresponding Taylor series in the following cases:

- 1) when expansion order $N = 20$ and $x = -1; 1; 2; 3; 5;$
- 2) when $N = 10$ and $x = -1; 1$

3) when $N = 10$ and $x = 2$ the error for cos is bounded, while error $|\text{real sin}(x) - \text{sinTaylor}(N, x)|$ is not.

A remark concerns low values of N . As expected, as N increases the error decreases and gets closer to zero; this is because the Taylor polynomial will better interpolate the real function. To confirm this, by mathematical analysis it is known that the 21st order of Taylor Series for sine will interpolate the function from 0 to 2π , hence it will cover an entire period; this also explain why for larger value of x a remarkable accuracy is achieved for larger N , i.e. for $x = 10$ (10 radians $\approx 572,9$ degrees) a negligible error is achieved for $N=20$. As a matter of fact, instead for $x = 1$ (hence 57,2 degrees) a noteworthy accuracy is definitely achieved already for $N=5$.

The code's output provides all together the values of the approximated functions, real functions and bounding because two for loops are done to evaluate each x for each N :

```
for (int n=0; n < 4; n++) {
    int N;
    N = Ng[n];
    for (int y=0; y < 6; y++){
        double x;
        x = xg[y];
        double realcos = cos(x);
        double realsin = sin(x);
        cout << "The current N value is: " << N << endl;
        cout << "The current x value is: " << x << endl;
        cout << "The approximated sine value is: " << sinTaylor(N,x) << endl;
        cout << "The mathematical sine real value is: " << realsin << endl;
        cout << "The approximated cosine value is: " << cosTaylor(N,x) << endl;
        cout << "The mathematical cosine real value is: " << realcos << endl;
        double sinbound = abs(sinTaylor(N+1,x) - sinTaylor(N,x));
        double cosbound = abs(cosTaylor(N+1,x) - cosTaylor(N,x));

        cout << "The error for the sine is: " << abs(realsin - sinTaylor(N,x)) << endl;
        cout << "The abs of N+1 term for the sine is: " << sinbound << endl;
        cout << "The error for the cosine is: " << abs(realcos - cosTaylor(N,x)) << endl;
        cout << "The abs of N+1 term for the cosine is: " << cosbound << endl;

        if (abs(realsin - sinTaylor(N,x)) < abs(sinbound)) {
            cout << "The error for the sine is bounded " << endl;
        };
        if (abs(realcos - cosTaylor(N,x)) < abs(cosbound)) {
            cout << "The error for the cosine is bounded " << endl;
        };
    };
};
```

Figure 2: Code snippet showing the two for loops for the relevant N and given x . The snippet illustrates also how the output will be shown.

The general output snippet - for a given pair (N,x) - is:


```
The current N value is: 5
The current x value is: 1
The approximated sine value is: 0.841471
The mathematical sine real value is: 0.841471
The approximated cosine value is: 0.540302
The mathematical cosine real value is: 0.540302
The error for the sine is: 1.59828e-010
The abs of N+1 term for the sine is: 1.6059e-010
The error for the cosine is: 2.07625e-009
The abs of N+1 term for the cosine is: 2.08768e-009
The error for the sine is bounded
The error for the cosine is bounded
```

Figure 3: Output snippet for N=5 and x=1

2 Task 2

2.1 Task 2 Background and Problem formulation

This task aims to solve an integral of a smooth function using the Adaptive Simpson quadrature, i.e. adaptive integration.

Let the function be $f(x)$ such that $f : [a, b] \rightarrow R$; the ASI does not integrate the function but rather the 2-nd order interpolating polynomial evaluated at $x_0 = a$, $x_1 = \frac{(a+b)}{2}$ and $x_2 = b$. The function to be studied is $f(x) = 1 + \sin e^{3x}$; the interval is $[a, b] = [-1, 1]$, then the integral is

$$\int_{-1}^1 1 + \sin e^{3x} dx$$

The Simpson rule is:

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

This is implemented in I function. Later, introducing the midpoint γ it is possible to define IT such as

$$I_2(a, b) = I(a, \gamma) + I(\gamma, b)$$

Given these two definitions, the error can be defined as follows: $|I - I_2|$. An appropriate value for tolerance is introduced; accordingly to ASI, if the error is lower than 15 times the tolerance, then I_2 is returned as value for the integral.

Finally the ASI algorithm is defined as recursive algorithm. This is done in ASI as here illustrated:

```
double ASI(double a, double b, double tol){
    double I1, I2;
    I1 = I(a,b,f);
    I2 = IT(a,b);

    double errset;
    errset = abs(I1 - I2);
    if (errset < 15*tol) {return I2;}
    else {return ASI(a, midpoint(a,b), tol/2) + ASI(midpoint(a,b), b, tol/2);};
}
```

Figure 4: ASI algorithm code's snippet.

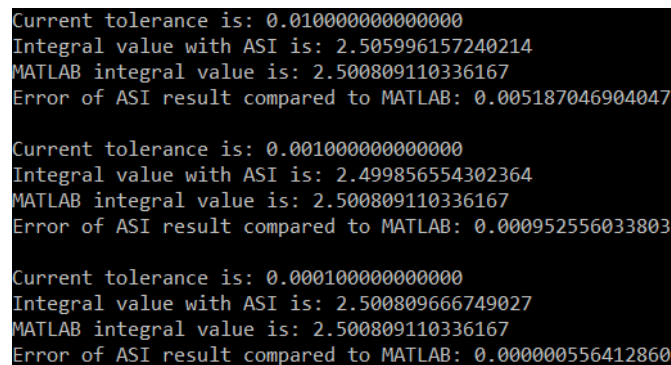
As Figure 4 shows, if the error is not lower than 15 times the tolerance then the interval will be split using its midpoint and for each half interval the tolerance is halved too.

As it can be seen in Figure 4 a further function is defined to compute the midpoint of two given points: `double midpoint(double a, double b)`.

2.2 Task 2 Implementation Results

The integral is both solved using MATLAB and the ASI algorithm. In MATLAB the `integral` inbuilt function is used; the relative tolerance is set to 10^{-8} . It is chosen to set the relative tolerance to this bounding value since it is a measure of the error relative to the size of each component solution.²

The output is here reported both as snippet of it and in Table 3:



```
Current tolerance is: 0.010000000000000
Integral value with ASI is: 2.505996157240214
MATLAB integral value is: 2.500809110336167
Error of ASI result compared to MATLAB: 0.005187046904047

Current tolerance is: 0.001000000000000
Integral value with ASI is: 2.499856554302364
MATLAB integral value is: 2.500809110336167
Error of ASI result compared to MATLAB: 0.000952556033803

Current tolerance is: 0.000100000000000
Integral value with ASI is: 2.500809666749027
MATLAB integral value is: 2.500809110336167
Error of ASI result compared to MATLAB: 0.00000556412860
```

Figure 5: ASI algorithm output compared to MATLAB result

²Mathworks - Absolute and Relative tolerance definitions

tol	Integral Value
10^{-2}	2.505996157240214
10^{-3}	2.499856554302364
10^{-4}	2.500809666749027

Table 3: Values for the Integral computed using ASI for different tolerances

As clear from the output snippet, a comparison to the MATLAB result is done, specifically calculating the absolute value of difference (error). This is here summarized in Table 4:

tol	Integral Value	MATLAB	Error
10^{-2}	2.505996157240214	2.500809110336167	0.005187046904047
10^{-3}	2.499856554302364	2.500809110336167	0.000952556033803
10^{-4}	2.500809666749027	2.500809110336167	0.000000556412860

Table 4: MATLAB and ASI values comparison

It can be seen that as the tolerances decreases, the error done decreases too. It was found empirically testing further tolerance that when the tolerance is set to 10^{-14} , the error is 0.0.

```
Current tolerance is: 0.00000000000010
Integral value with ASI is: 2.500809110336167
MATLAB integral value is: 2.500809110336167
Error of ASI result compared to MATLAB: 0.00000000000000
```

Figure 6: Empirical Result of Zero Error for ASI compared to MATLAB

A Task 1

```
1 // Task 1
2
3 #include <iostream>
4 #include <cmath>
5 #include <iomanip>
6
7 using namespace std;
8
9
10 // Functions declaration
11
12 double sinTaylor(int N, double x);
13 double cosTaylor(int N, double x);
14
15
16
17 int main()
18 {
19     int Ng [5] = {1,5,10,20}; // Number of terms
20     double xg [6] = {-1,1,2,3,5,10}; // Values of x
21
22
23     for (int n=0; n < 4; n++) {
24         int N;
25         N = Ng[n];
26         for (int y=0; y < 6; y++){
27             double x;
28             x = xg[y];
29             double realcos = cos(x);
30             double realsin = sin(x);
31             cout << "The current N value is: " << N << endl;
32             cout << "The current x value is: " << x << endl;
33             cout << "The approximated sine value is: " <<
34                 sinTaylor(N,x) << endl;
35             cout << "The mathematical sine real value is: "
36                 << realsin << endl;
37             cout << "The approximated cosine value is: " <<
38                 cosTaylor(N,x) << endl;
```

```
36         cout << "The mathematical cosine real value is:
           " << realcos << endl;
37         double sinbound = abs(sinTaylor(N+1,x) -
           sinTaylor(N,x));
38         double cosbound = abs(cosTaylor(N+1,x) -
           cosTaylor(N,x));
39
40         cout << "The error for the sine is: " << abs(
           realsin - sinTaylor(N,x)) << endl;
41         cout << "The abs of N+1 term for the sine is: "
           << sinbound << endl;
42         cout << "The error for the cosine is: " << abs(
           realcos - cosTaylor(N,x)) << endl;
43         cout << "The abs of N+1 term for the cosine is:
           " << cosbound << endl;
44
45
46         if (abs(realsin - sinTaylor(N,x)) < abs(sinbound
           )) {
47             cout << "The error for the sine is bounded "
               << endl;
48         };
49         if (abs(realcos - cosTaylor(N,x)) < abs(cosbound
           )) {
50             cout << "The error for the cosine is bounded
               " << endl;
51         };
52
53         cout << "
           " << endl;
54
55     }
56 }
57
58
59     return 0;
60
61 }
62
63 // Functions definition using Horner's scheme for polynomial
   evaluation
```

```
64
65 double sinTaylor(int N, double x){
66
67     int i;
68     double summ = 1;
69
70     for (i = N ; i > 0; i--) {
71
72         summ = 1 - x*x*summ/(double)(2*i*(2*i+1));
73
74     }
75
76     summ = x*summ;
77     return summ;
78 }
79
80
81 double cosTaylor(int N, double x){
82
83     int i;
84     double summ = 1;
85
86
87     for (i = N ; i > 0; i--) {
88
89         summ = 1 - x*x*summ/(double)((2*i)*(2*i - 1));
90     }
91     return summ;
92 }
```

B Task 2

```
1 //Task 2
2
3 #include <iostream>
4 #include<iomanip>
5 #include<cmath>
6
7 using namespace std;
8
9 // Functions declaration
10 double f(double x);
11 double midpoint(double a, double b);
12 double I(double a, double b, double f(double x));
13 double IT(double a, double b);
14 double ASI(double a, double b, double tol);
15
16
17 int main(int argc, char *argv[])
18 {
19     double a, b, matlab;
20     a = -1;
21     b = 1;
22     matlab = 2.500809110336167;
23     double tol [3] = {1e-2, 1e-3, 1e-4};
24
25     cout << fixed << setprecision(15);
26
27     for (int i = 0; i < 3; i++){
28         double tolerance;
29         tolerance = tol[i];
30         cout << "Current tolerance is: " << tol[i] << "
31             " << endl;
32         cout << "Integral value with ASI is: " << ASI(a,b,
33             tolerance) << endl;
34         cout << "MATLAB integral value is: 2.500809110336167
35             " << endl;
36         cout << "Error of ASI result compared to MATLAB: "
37             << abs(matlab - ASI(a,b,tolerance)) << endl;
38         cout << endl;
```

```
35     }
36
37     return 0;
38 }
39
40
41 // Functions definition
42
43 double ASI(double a, double b, double tol){
44     double I1, I2;
45     I1 = I(a,b,f);
46     I2 = IT(a,b);
47
48     double errset;
49     errset = abs(I1 - I2);
50     if (errset < 15*tol) {return I2;}
51     else {return ASI(a, midpoint(a,b), tol/2) + ASI(midpoint
52               (a,b), b, tol/2);};
53 }
54
55
56 double I(double a, double b, double f(double x)){
57     double value;
58     value = (b-a)/6*(f(a) + 4*f(midpoint(a,b)) + f(b));
59     return value;
60 }
61
62 double IT(double a, double b){
63     double value2;
64     double gamma;
65     gamma = midpoint(a,b);
66     value2 = I(a,gamma,f) + I(gamma,b,f);
67     return value2;
68 }
69
70 double f(double x){ return (1+sin(exp(3*x)));}
71
72 double midpoint (double a, double b){
73     double gamma;
```



```
74     gamma = (a+b)*0.5;  
75     return gamma;  
76 }
```