Report Project1: Printouts and Source code
SF2565: Program construction in C++ for Scientific Computing

Marcus Klasson

September 23, 2019

Task 1: Taylor Series

Printouts from `taylor.cpp` for different number of terms n in Taylor series:

```
Computing Taylor series of sin(x) and cos(x) using 1 terms.

x   sin(x)      sinTaylor(x)    error           bound
-1  -0.841471   -0.833333       8.137651e-03    4.166667e-02
1   0.841471    0.833333        8.137651e-03    4.166667e-02
2   0.909297    0.666667        2.426308e-01    1.333333e-01
3   0.141120    -1.500000       1.641120e+00    6.750000e-01
5   -0.958924   -0.931044       2.788020e-02    1.163805e+00
10  -0.544021   0.250759        7.947804e-01    1.253796e+00

x   cos(x)      cosTaylor(x)    error           bound
-1  0.540302    0.500000        4.030231e-02    4.166667e-02
1   0.540302    0.500000        4.030231e-02    4.166667e-02
2   -0.416147   -1.000000       5.838532e-01    3.333333e-01
3   -0.989992   -3.500000       2.510008e+00    2.625000e+00
5   0.283662    0.176718        1.069445e-01    3.681619e-01
10  -0.839072   -2.293129       1.454058e+00    1.910941e+01
```

```
Computing Taylor series of sin(x) and cos(x) using 2 terms.

x   sin(x)      sinTaylor(x)    error           bound
-1  -0.841471   -0.841667       1.956819e-04    2.003968e-02
1   0.841471    0.841667        1.956819e-04    2.003968e-02
2   0.909297    0.933333        2.403591e-02    8.888889e-02
3   0.141120    0.525000        3.838800e-01    1.125000e-01
5   -0.958924   -0.960035       1.110960e-03    5.714495e-01
10  -0.544021   -0.676958       1.329368e-01    1.611805e+00

x   cos(x)      cosTaylor(x)    error           bound
-1  0.540302    0.541667        1.364361e-03    1.805556e-02
1   0.540302    0.541667        1.364361e-03    1.805556e-02
2   -0.416147   -0.333333       8.281350e-02    4.444444e-02
3   -0.989992   -0.125000       8.649925e-01    3.750000e-02
5   0.283662    0.289683        6.021163e-03    2.414028e-01
10  -0.839072   -0.485679       3.533923e-01    1.618931e+00
```

```
Computing Taylor series of sin(x) and cos(x) using 5 terms.

x   sin(x)      sinTaylor(x)    error           bound
-1  -0.841471   -0.841471       1.598285e-10    5.394045e-03
```

```
1    0.841471    0.841471        1.598285e-10    5.394045e-03
2    0.909297    0.909296        1.290863e-06    2.331529e-02
3    0.141120    0.140875        2.454139e-04    8.127380e-03
5    -0.958924   -0.958924       4.074587e-09    1.536738e-01
10   -0.544021   -0.543988       3.261118e-05    3.487106e-01

x    cos(x)      cosTaylor(x)    error           bound
-1   0.540302    0.540302        2.076253e-09    4.093199e-03
1    0.540302    0.540302        2.076253e-09    4.093199e-03
2    -0.416147   -0.416155       8.366275e-06    1.261076e-02
3    -0.989992   -0.991049       1.056611e-03    6.757153e-02
5    0.283662    0.283662        4.123022e-08    5.372389e-02
10   -0.839072   -0.839236       1.644090e-04    6.357848e-01
```

```
Computing Taylor series of sin(x) and cos(x) using 10 terms.

x    sin(x)      sinTaylor(x)    error           bound
-1   -0.841471   -0.841471       0.000000e+00    1.662986e-03
1    0.841471    0.841471        0.000000e+00    1.662986e-03
2    0.909297    0.909297        3.330669e-16    7.188122e-03
3    0.141120    0.141120        3.587630e-12    2.510040e-03
5    -0.958924   -0.958924       1.110223e-16    4.737768e-02
10   -0.544021   -0.544021       9.992007e-14    1.075141e-01

x    cos(x)      cosTaylor(x)    error           bound
-1   0.540302    0.540302        1.110223e-16    1.169486e-03
1    0.540302    0.540302        1.110223e-16    1.169486e-03
2    -0.416147   -0.416147       3.663736e-15    3.603003e-03
3    -0.989992   -0.989992       2.747003e-11    1.928557e-02
5    0.283662    0.283662        3.330669e-16    1.534969e-02
10   -0.839072   -0.839072       8.897327e-13    1.816172e-01
```

```
Computing Taylor series of sin(x) and cos(x) using 100 terms.

x    sin(x)      sinTaylor(x)    error           bound
-1   -0.841471   -0.841471       0.000000e+00    2.052068e-05
1    0.841471    0.841471        0.000000e+00    2.052068e-05
2    0.909297    0.909297        0.000000e+00    8.869896e-05
3    0.141120    0.141120        5.551115e-17    3.097303e-05
5    -0.958924   -0.958924       1.110223e-16    5.846244e-04
10   -0.544021   -0.544021       5.551115e-16    1.326687e-03

x    cos(x)      cosTaylor(x)    error           bound
-1   0.540302    0.540302        1.110223e-16    1.330728e-05
1    0.540302    0.540302        1.110223e-16    1.330728e-05
2    -0.416147   -0.416147       5.551115e-17    4.099767e-05
3    -0.989992   -0.989992       1.110223e-16    2.194457e-04
5    0.283662    0.283662        3.330669e-16    1.746602e-04
10   -0.839072   -0.839072       3.330669e-16    2.066577e-03
```

Source code

The Main program `taylor.cpp`:

```
#include <iostream>
#include <cmath>
#include <vector>
#include <stdio.h>
#include "taylor.hpp"

using namespace std;

// Reduce x to the interval [-pi, pi)
double argumentReduction(double radians) {
    return radians - 2*M_PI*round(radians / (2*M_PI) );
}

double sinTaylor(int n, double x) {

    x = argumentReduction(x);

    int sign = -1;
    double sum, t;
    t = x; // term for n=0 is x
    sum = x;
    for (int i = 1; i <= n; ++i) {
        t = t * (x * x/((2*i + 1) * (2*i)) );
        sum = sum + sign * t;
        sign = -sign;
    }
    return sum;
}

double cosTaylor(int n, double x) {

    x = argumentReduction(x);

    int sign = -1;
    double sum, t;
    t = 1.0; // term for n=0 is 1
    sum = 1.0;
    for (int i = 1; i <= n; ++i) {
        t = t * (x * x/( (2*i) * (2*i - 1)));
        sum = sum + sign * t;
        sign = -sign;
    }
    return sum;
}

int main() {

    vector<double> x = {-1, 1, 2, 3, 5, 10}; // Values of x for sin(x) and cos(x)
    int n;
    cout << "\nSelect number of terms in Taylor series: ";
    cin >> n;

    double sinx, cosx;
    double sinApprox, cosApprox;
    double error, bound;
```

```
    printf("Computing Taylor series of sin(x) and cos(x) using %d terms.\n\n", n);
    cout << "x \t sin(x) \t sinTaylor(x) \t error \t\t bound \t" << endl;
    for (int i = 0; i != x.size(); ++i) {
        sinx = sin(x[i]);
        sinApprox = sinTaylor(n, x[i]);
        error = abs(sinx - sinApprox);
        bound =  pow(-1.0, n+1) * sinApprox * (x[i] * x[i])/ ( (2*(n+1) + 1) * (2*(n+1) ) );
        printf("%.f \t %1.6f \t %1.6f \t %1.6e \t %1.6e \n",
                x[i], sinx, sinApprox, error, abs(bound));
    }

    cout << "\nx \t cos(x) \t cosTaylor(x) \t error \t\t bound \t" << endl;
    for (int i = 0; i != x.size(); ++i) {
        cosx = cos(x[i]);
        cosApprox = cosTaylor(n, x[i]);
        error = abs(cosx - cosApprox);
        bound =  pow(-1.0, n+1) * cosApprox * (x[i] * x[i])/ ( (2*(n+1)) * (2*(n+1) - 1) );
        printf("%.f \t %1.6f \t %1.6f \t %1.6e \t %1.6e \n",
                x[i], cosx, cosApprox, error, abs(bound));
    }
    return 0;
}
```

The header file `asi.hpp`:

```
#ifndef TAYLOR_HPP
#define TAYLOR_HPP

double sinTaylor(int n, double x);

double cosTaylor(int n, double x);

double argumentReduction(double radians);

#endif
```

Task 2: Adaptive Integration

Printouts from file asi.cpp:

```
Adaptive Simpson Integration of f(x) = 1 + sin(exp(3*x)):
 I          tolerance
 2.548323   1e-01
 2.505996   1e-02
 2.499857   1e-03
```

```
Adaptive Simpson Integration of f(x) = 1 + sin(exp(3*x)):
 I          tolerance
 2.500809   1e-07
```

Source code

The Main program asi.cpp:

```
#include <iostream>
#include <stdio.h>
#include <cmath>
#include <vector>
#include "asi.hpp"

using namespace std;

// target function is f(x) = 1 + sin(exp(3*x))
double target(double x) {
    return 1 + sin( exp(3*x) );
}

double simpsonRule(FunctionPointer f, double a, double b) {
    return ((b - a) / 6.0) * ( f(a) + 4 * f((a+b)/2) + f(b) );
}

double ASI(FunctionPointer f, double a, double b, double tol) {
    double I1, I2, gamma, errest;
    gamma = 0.5 * (a + b); // mid point
    I1 = simpsonRule(f, a, b);
    I2 = simpsonRule(f, a, gamma) + simpsonRule(f, gamma, b);
    errest = abs(I1 - I2);
    if (errest < 15.0*tol) {
        return I2;
    }
    return ASI(f, a, gamma, tol/2) + ASI(f, gamma, b, tol/2);
}

int main() {
    double I;
    vector<double> tol = {10e-2, 10e-3, 10e-4};

    printf("Adaptive Simpson Integration of f(x) = 1 + sin(exp(3*x)): \n");
    printf(" I \t\t tolerance \n");
    for (int i = 0; i != tol.size(); ++i) {
        I = ASI(&target, -1.0, 1.0, tol[i]);
        printf(" %3.6f \t %.e \n", I, tol[i]);
    }
    return 0;
}
```

The header file asi.hpp:

```
#ifndef ASI_HPP
#define ASI_HPP

typedef double (*FunctionPointer)(double);

double ASI(FunctionPointer f, double a, double b, double tol);

double simpsonRule(FunctionPointer f, double a, double b);

#endif
```

Matlab code `task2.m`:

```
fun = @(x) 1 + sin(exp(3*x));
I = integral(fun, -1, 1);
fprintf('Integral of f(x) = 1 + sin(exp(3*x)): %2.6f \n', I)
```