Exercise set 0

Computational Methods for Engineering Applications II
D-MATH, HS-2015
Lecturer: Prof. Dr. Siddhartha Mishra

This is a warm-up exercise sheet. You should complete this exercise to make sure you are familiar with the programming tools used in the course. There is no new theory introduced in this exercise.

Problem 0.1 Midpoint rule

In this exercise we let $a < b$ be two real numbers and $f : [a, b] \to \mathbb{R}$ be a smooth function. Our goal is to compute a numerical approximation to the integral

$$I(f) := \int_a^b f(x)\,\mathrm{d}x.$$

Recall that for a given number of subintervals n, the *midpoint rule* $I_n(f)$ is given as

$$I_n(f) := \frac{b-a}{n}\left[\sum_{k=0}^{n-1} f\left(a + (k+1/2)\frac{b-a}{n}\right)\right].$$

It can be checked that the error scales as $\mathcal{O}(n^{-2})$, in other words

$$|I(f) - I_n(f)| \leq Cn^{-2}.$$

0.1a)

Write a function in C++ that computes and returns (as `double`) the midpoint rule, given the parameters:

- the endpoints a and b (as `double`),
- the number of subintervals n (as `int`),
- a function pointer to f.

You should use the following signature:

```
#pragma once
///
/// This is the type of a function taking as parameter a double, and
/// return a double
///
typedef double(*FunctionPointer)(double);

///
/// Computes the midpoint rule to approximate the integral
///
/// \param a the left endpoint
/// \param b the right endpoint
/// \param n the number of subintervals to use
/// \param f the function to compute the integral over
///
double midpoint_rule(double a, double b, int n, FunctionPointer f);
```

template_code/series0_1_a.hpp

and the following template:

```cpp
#include "series0_1_a.hpp"

double midpoint_rule(double a, double b, int n, FunctionPointer f) {
    // Insert your code here.
}
```

template_code/series0_1_a.cpp

0.1b)

For the rest of the problem, we set

$$a = 0.2,\ b = 1.3 \text{ and } f(x) = \sin(\pi x).$$

Compute the exact integral $I(f)$.

0.1c)

Write a C++ program that computes and prints $I_n(f)$ for $n = 100$. You may use the following template:

```cpp
// To use our previously written midpoint rule function
#include "series0_1_a.hpp"

// For printing to the terminal
#include <iostream>

// On some platforms we need to add this in order
// to get M_PI defined
#define _USE_MATH_DEFINES

// for our usual math functions and constants
#include <math.h>


// We use these two to set the precision of our output.
#include <limits>
#include <iomanip>

double f(double x) {
    // Compute f(x) here
}

int main() {
    // TODO: Compute the proper approximation here:
    const double In = 0;


    // Set high precision for output, easier to see what is going on:
    std::cout << std::setprecision(std::numeric_limits<long double>::digits10 + 1);
    // We print out the value of the midpoint rule here:
    std::cout << "In = " << In << std::endl;

    return 0;
}
```

template_code/series0_1_c.cpp

0.1d)

In this exercise we will investigate the experimental order of convergence for the midpoint rule. Write a C++ program that computes the difference

$$|I(f) - I_n(f)|,$$

for

$$n = 2^k \qquad k = 4, \ldots, 11.$$

Store the output to file and plot the results in MATLAB/Python using log scales on both axes. How does this plot agree with the error bound

$$|I(f) - I_n(f)| \leq Cn^{-2}?$$

You may use the following template code:

```cpp
#include "series0_1_a.hpp" // To use our library
#include "writer.hpp" // This is the output function to write to file

// We store our results in a vector
#include <vector>

// On some platforms we need to add this in order
// to get M_PI defined
#define _USE_MATH_DEFINES

// for our usual math functions and constants
#include <math.h>

double f(double x) {
    // Define f here
}


int main() {
    const double a = 0.2;
    const double b = 1.3;
    double exact = (cos(M_PI*a) - cos(M_PI * b)) / M_PI;

    // We store the errors in this vector
    std::vector<double> errors;

    // We store the number of subintervals we have used here
    std::vector<int> numberOfSubintervals;

    for(int k = 4; k <= 11; k++) {
        int n = 1 << k;

        // Compute the midpoint rule here.

        // Compute the correct error:
        double error = 0;
        errors.push_back(error);
        numberOfSubintervals.push_back(n);
    }

    // Write result to disk
    writeToFile("series0_1_d_errors.txt", errors);
    writeToFile("series0_1_d_numbers.txt", numberOfSubintervals);
}
```

template_code/series0_1_d.cpp

For plotting in MATLAB, you may use the following script:

```
errors = load('series0_1_d_errors.txt');
numberOfIntervals = load('series0_1_d_numbers.txt');

loglog(numberOfIntervals, errors, '*-');
title('Error plot for ex. 0.1d)')
xlabel('n')
ylabel('|I(f)-I_n(f)|')

grid on
grid minor
```

template_code/plot_error.m

and in python you may use

```
import numpy
import pylab
errors = numpy.loadtxt('series0_1_d_errors.txt')
numberOfIntervals = numpy.loadtxt('series0_1_d_numbers.txt')

pylab.loglog(numberOfIntervals, errors, '-*')
pylab.title('Error plot for ex. 0.1d)')
pylab.xlabel('n')
pylab.ylabel('|I(f)-I_n(f)|')
pylab.grid('on')
pylab.show()
```

template_code/plot_error.py

All templates are available at:
`https://www2.math.ethz.ch/education/bachelor/lectures/hs2015/other/cmea2/series/template_code0.zip`