SF2656/FSF3565 Assignment 1

Björn Wehlin

Problem 1 – Adaptive integration

Problem 1a (4pts)

Consider the computation of the definite integral

$$I = \int_a^b f(x)\,dx$$

for a smooth function $f\colon [a,b] \to \mathbb{R}$. The task consists of computing an approximation to the integral with a prescribed tolerance τ. We will use adaptive Simpson quadrature. You have learned about it in the basic course in Numerical Analysis. Let

$$I(\alpha,\beta) = \frac{\beta-\alpha}{6}(f(\alpha) + 4f((\alpha+\beta)/2) + f(\beta))$$

be the Simpson rule applied to evaluating the integral $\int_\alpha^\beta f(x)\,dx$. Then, it holds for the error

$$I(\alpha,\beta) - \int_\alpha^\beta f(x)\,dx = \frac{(\beta-\alpha)^5}{2880} f^{(4)}(\xi_1)$$

for some $\xi_1 \in (\alpha,\beta)$. If we introduce the midpoint $\gamma = \frac{1}{2}(\alpha+\beta)$ and $I_2(\alpha,\beta) := I(\alpha,\gamma) + I(\gamma,\beta)$, this equation leads to

$$I_2(\alpha,\beta) - \int_\alpha^\beta f(x)\,dx = \frac{(\beta-\alpha)^5}{46080} f^{(4)}(\xi_2)$$

for some $\xi_2 \in (\alpha,\beta)$. These two equations can be used in order to estimate the error of the Simpson rule applied to f on $[\alpha,\beta]$: If we assume that $f^{(4)}(\xi_1) \approx f^{(4)}(\xi_2)$, then

$$I_2(\alpha,\beta) - \int_\alpha^\beta f(x)\,dx \approx \frac{1}{15}(I_2(\alpha,\beta) - I(\alpha,\beta)).$$

Thus, the error of the numerical approximation $I_2(\alpha,\beta)$ can be estimated by the right-hand term

$$\texttt{errest} = \frac{1}{15}(I_2(\alpha,\beta) - I(\alpha,\beta)).$$

So we can design the following algorithm:

```
Algorithm 1: ASI($f, a, b, \tau$) (Adaptive Simpson Integration)
  Data: Function $f$, bounds $a < b$, tolerance $\tau$
  $i_1 \leftarrow I(a, b)$;
  $i_2 \leftarrow I_2(a, b)$;
  errest $\leftarrow |i_1 - i_2|$;
  if errest $< 15\tau$ then
      return $i_2$;
  end
  return ASI($f, a, (a + b)/2, \tau/2$) + ASI($f, (a + b)/2, b, \tau/2$);
```

Implement the algorithm! Apply your method to approximate the integral

$$\int_0^\pi [x + \cos(x^5)]\, dx$$

with a tolerance τ of $10^{-2}, 10^{-3}, 10^{-4}$! Compare your results with the value provided by MATLAB's integration functions (with an absolute tolerance of 10^{-8}).

Note To pass f to your integration function, you can use an argument of type `std::function<double(double)>` (you'll need `#include <functional>`). This will let your pass f as a variable to the integration function, and then call the variable as if it was the original function. See `https://en.cppreference.com/w/cpp/utility/functional/function` for details (the example on storing a free function may be of particular interest).

Acknowledgment Problem 1a is from Michael Hanke. Some slight modifications have been made.

Problem 1b (2pts)

For numerical algorithms that do function evaluations, it is important to understand how many times this happens. Add (optional) tracking of the number of times f is called during the integration.

You should not rely on global variables for this (maybe a pointer can help?). It should be possible to call your integration routine several times during the same execution of the program.

Report how many evaluations of f are done when the tolerance is, as before, $\tau = 10^{-2}, 10^{-3}$ and 10^{-4}.

Problem 2 – Unit testing

Problem 2a (3 pts)

An important part of writing robust scientific software is to add testing. In this exercise, you will learn some basic techniques for unit testing in C++. We will use the program we wrote in Problem 1 and add unit tests to it.

In unit testing, we test small parts of the code (see, e.g., `https://en.wikipedia.org/wiki/Unit_testing`). There are several unit testing frameworks available for C++, for example Google Test and Catch2. We will use utest.h, which has a similar interface to Google Test but is much easier to include as part of a project.[1]

Unit tests should be simple, small and self-contained (and not rely on outside resources such as disks or network, etc.).

Start by downloading `utest.h` from Canvas or `https://github.com/sheredom/utest.h` and putting the header file in the same directory as your project.

For unit testing, one typically builds a separate executable to run the tests. To accomplish this without code duplication, it may be a good idea to break your project into separate files where your integration routines can be used both from the executable for Problem 1, and the unit testing executable.

- Read the documentation at `https://github.com/sheredom/utest.h`.
- Integrate `utest.h` in your build
- Create a unit test that expects/asserts that $\int_0^1 x\,dx = 1/2$ (of course, using your adaptive Simpson integrator)

Problem 2b (1pt)

In your adaptive Simpson integrator, add a check that $\tau > 0$. If not, throw an exception and add a unit test that an exception does indeed get thrown in this case.

(In a proper unit testing methodology, you would start by first writing the test that the exception is thrown, run the program to see that the new test is failing. Only then should you modify the code to actually throw the exception and rerun to check that the test is now succeeding.)

Submission

The programming exercises should be done individually, or in groups of two. Hand in a report containing:

- Comments and explanations that you think are necessary for understanding your program. (But do not comment excessively.)

[1] If you are brave enough, you may use Google Test or Catch2 instead of utest.h for this part of the exercise. It's a good exercise in putting builds together!

- The output of your program according to the tasks. Don't forget to draw conclusions!
- Printout of the source code in PDF format. (We need this to be able to comment on your code, points will be deducted for missing to do this.) You can either put the source code as an appendix to your written report, or you can hand in separate PDF(s).

In addition, all source code for your program(s) in .zip/.tar format and instructions for how to compile/run.

Good luck!