Exercise Sheet 6

Exercise 1: C++ Quiz (5 points)

Here are some additional questions from https://cppquiz.org for you to answer:

Question 1: https://cppquiz.org/quiz/question/162 (unqualified name lookup)

Question 2: https://cppquiz.org/quiz/question/44 (dynamic polymorphism)

Question 3: https://cppquiz.org/quiz/question/29 (virtual during con-/destruction)

Question 4: https://cppquiz.org/quiz/question/224 (abstract base classes)

Question 5: https://cppquiz.org/quiz/question/133 (diamond pattern)

Question 1 is about name lookup rules, and questions 2 to 5 have a look at dynamic polymorphism, virtual methods, and virtual inheritance. Do the results match with your expectations? Discuss.

Exercise 2: Interfaces: Integration (20 points)

In the lecture the concept of interfaces was introduced, using virtual methods and abstract base classes. This exercise considers numerical integration as an example application.

The goal is to write a code library that allows, via numerical integration, to determine the integral of an arbitrary function $f(t)$ on a given interval $t \in [A, B]$ $(A, B \in \mathbb{R})$. The interval $[A, B]$ is divided in N subintervals of uniform length Δt.

$$\Delta t = \frac{B - A}{N}, \quad t_i = A + \Delta t \cdot i, \quad i = 0, \dots, N$$

On each of the subintervals the integral can be estimated using an appropriate quadrature rule. Summing up these estimates produces a *composite quadrature rule*:

$$\int_A^B f(t)dt = \sum_{i=0}^{N-1} \left(Q_{t_i}^{t_{i+1}}(f) + E_{t_i}^{t_{i+1}}(f) \right)$$

Here Q_a^b refers to the result of the quadrature rule on the interval $[a, b]$ and E_a^b to the error. Quadrature rules are normally based on polynomial interpolation. This means it is possible to specify the maximum degree of polynomials that are integrated exactly and the order of the error in the case of functions that can't be integrated exactly.

In this exercise, we want to limit ourselves to two quadrature rules:

- The trapezoidal rule:

$$Q_{\text{Trapez}}{}_a^b(f) = \frac{b-a}{2}(f(a) + f(b))$$

  It is exact for polynomials up to first order, and the error of the resulting composite rule is $O(\Delta t^2)$.

- The Simpson rule:

$$Q_{\text{Simpson}}{}_a^b(f) = \frac{b-a}{6} \cdot \left(f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right)$$

  It is exact for polynomials up to degree two, and the error of the resulting composite rule is $O(\Delta t^4)$.

A good test for such a numerical integration is to study the order of convergence. For a problem with a known solution, calculate the error E_N for N subintervals and for $2N$ subintervals. Doubling the number of subintervals cuts the interval width Δt in half, reducing the error. The quotient

$$EOC = \frac{\log(E_N/E_{2N})}{\log(2)}$$

is an indication of the order of convergence of the error (e.g., for the trapezoidal rule you should observe order 2).

(a) Design interfaces for the different concepts. To this end, proceed as follows:

- First, identify the abstract concepts in our problem.
  - We want to evaluate integrals of various functions; so we have the *function* as an abstract concept.
  - An integral is represented as a sum of integrals of subintervals.
  - On each subinterval a suitable *quadrature formula* is applied.
  - The quadrature formula evaluates the function to be integrated at certain points in order to determine an approximate solution.
  - A *composite quadrature rule* sums the results.
  - A suitable *test problem* would contain
    - ∗ a function to integrate
    - ∗ a choice of quadrature rule and composite rule
    - ∗ the interval bounds
    - ∗ the true resulting integral value
- How are these abstract concepts mapped to interfaces? Usually, one introduces a base class as an interface description for each concept.
  - What is a suitable interface for a function?
  - A quadrature rule evaluates a function on a given interval, how would a suitable virtual method look like?
  - Each quadrature rule can also report the degree of exactly integrable polynomials, and report the order of convergence of the integration error that results.
  - The actual integration (i.e., composite quadrature rule) is parameterized with a quadrature rule and a function. It evaluates the integral of the function on an interval that has to be specified, dividing the interval into N subintervals. In addition, the quadrature formula that is to be used must be specified.
  - Apart from the equidistant composite quadrature used here, there may be other approaches (see second exercise). In anticipation of extensions, also introduce the abstract concept of a composite quadrature rule, although right now there is only one actual implementation.
  - What would be a suitable interface for test problems?

(b) Describe your interfaces and explain the design decisions. What are the reasons for your specific choice of interfaces?

(c) Implement the interfaces using abstract base classes and dynamic polymorphism. Write implementations for all of the above variants of abstract concepts (trapezoidal rule, Simpson's rule, equidistant composite rule). Your functions and classes should be as general as possible, i.e., use interface classes for arguments and return values. Make use of the best practices introduced in the lecture.

(d) Test your implementation with the integration of $2t^2 + 5$ on the interval $[-3, 13]$ and $\frac{t}{\pi}\sin(t)$ on the interval $[0, 2\pi]$. To do this, implement appropriate test problems for each combination of test function and quadrature rule. Write a free function that expects a test problem and a number of steps to perform, and then

- reports the true integral value and expected convergence order
- starts with one big interval without subdivisions
- doubles the number of subintervals after each step
- applies the composite rule in each step
- reports in each step:
  - the current estimated value
  - the resulting error
  - the estimated order of convergence (EOC)
  - the deviation from the expected order

Discuss the results.

Exercise 3: Adaptive Integration (10 points (bonus))

An improvement over the integration using equidistant intervals is adaptive integration. Here one tries to only use small intervals Δt where the error is actually large.

If you are in need of additional points or not challenged enough `;-)`, you may write a class for adaptive integration that complements the equidistant one from the previous exercise. This new version should use the same interface for functions and quadrature rules and fit into the test infrastructure you have written.

One can estimate the error in one of the subintervals by comparing the result of quadrature with the result of an interval that has been refined once, i.e., the local error is estimated through hierarchical refinement. The goal is to have the “error density”, i.e., error per unit length of interval, to be about the same in all subintervals. Subintervals with large errors are divided again, iteratively or recursively. One starts with a small number of subintervals and repeats the error estimation and refinement until the desired number of subintervals is reached.

To avoid having to constantly reevaluate the integrals of the individual subintervals, it is helpful to store the subintervals, integrals and estimated errors in appropriate data structures: note that the refinement used to estimate the local error will immediately become part of the set of subintervals if this error proves to be large. The containers of the C++ Standard Library, e.g., `std::map` and `std::deque`, might prove useful. The same holds for function evaluations: note that one half resp. one third of the function evaluations in the previous exercise were actually unnecessary and quite wasteful, because interval bounds were generally evaluated twice.

You are free to design more or less efficient algorithms for the local error estimation and for deciding which intervals should be bisected and in which order. Your primary concern is a program that produces good approximations w.r.t. the number of intervals used, thoughts about runtime efficiency are a good idea but not mandatory.

The points of this exercise don’t count towards the total number of points that can be achieved during the semester. Consequently, you can do this exercise instead of another one later on or in addition to the other exercises to achieve a higher point percentage.