Advanced Programming - Exam 06 Jun 2025 - Part 2

Objective

Implement a C++ program that estimates definite integrals using the **Monte Carlo method**. The program should use **inheritance, polymorphism, and templates** to support various integrand functions and floating-point types.

Mathematical description

Given a function $f(x)$ defined on an interval $[a, b]$, the definite integral is:

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

Using the **Monte Carlo method**, we approximate this integral by:

$$I \approx \frac{b-a}{N} \sum_{i=1}^{N} f(x_i)$$

where $x_i \sim \mathcal{U}(a, b)$ are *uniform random* samples in $[a, b]$, and N is the number of samples.

Exercise instructions

Overview

Your task is to build a Monte Carlo integrator using C++ and apply object-oriented design principles:

- **Inheritance**: Create a base class `Integrator` and derive specific implementations.
- **Polymorphism**: Use virtual methods to dispatch between integration strategies.
- **Templates**: Make the integrator work with different numeric types (`float`, `double`).

Tasks

1. **(1 point) Define a base class `Integrator`**
   - Include a pure virtual function:
     ```
     virtual double integrate(std::function<double(double)> f,
                              double a, double b, std::size_t N) = 0;
     ```
   - Optionally include methods for setting a random seed.
2. **(2 points) Implement `MonteCarloIntegrator`**
   - Override `integrate()` using Monte Carlo random sampling over $[a, b]$.
   - Use the C++ standard library's `<random>` for number generation.
   - Ensure reproducibility with optional seed control.
3. **(1 point) Use polymorphism**
   - In `main()`, define a pointer or reference to the base class `Integrator` and dispatch to the concrete implementation at runtime.
4. **(2 points) Test your implementation**
   - Evaluate the integral of:
     - $f(x) = x^2$ on $[0, 1] \rightarrow$ Exact: $1/3$
     - $f(x) = \sin(x)$ on $[0, \pi] \rightarrow$ Exact: 2
   - Compare estimated results with exact values for various N.
5. **(2 points) Templatize your code**
   - Allow the integrator to work with both `float` and `double` types.
   - Test both versions in the same program.
6. **(2 points) Configuration and compilation**
   - Write a `CMakeLists.txt` to compile your code into a library and a test executable.
   - Provide clear build and usage instructions.
7. **(5 points) Python bindings using pybind11**
   - Expose your C++ class and methods to Python using `pybind11`.
   - Allow the user to pass Python functions to be integrated.
   - Demonstrate the binding with a Python script.
   - Compare your implementation with `scipy.integrate.quad`.

Evaluation criteria

- Code design and modularity.
- Correct use of inheritance, polymorphism, and templates.
- Accuracy of the Monte Carlo estimates.
- Memory and exception safety.
- Clean Python interface and working comparison with SciPy.