Programming Techniques for Scientific Simulations

HS 2023
Dr. R. Käppeli,
V. Armegioiu,
I. Méndez Veiga, M. Herde,
F. Sasseli, N. Siegenheim

Exercise sheet 8

Due date: Monday, November 13, 2023, 23:59 CET (midnight).

Problem 8.1 Inheritance

Specify the output of the following code (without running it!).

```cpp
#include <iostream>

class A {
  public:
    virtual void f() const { std::cout << "A::f "; }
    void g() const { std::cout << "A::g "; }
};

class B: public A {
  public:
    void f() const { std::cout << "B::f "; }
    void g() { std::cout << "B::g "; }
};

class C: public B {
  public:
    void f() { std::cout << "C::f "; }
    void g() const { std::cout << "C::g "; }
};

void func(A const& a) {
  a.f();
  a.g();
}

int main() {
  A a;
  B b;
  C c;

  a.f();
  a.g();

  b.f();
  b.g();

  c.f();
  c.g();

  func(a);
  func(b);
  func(c);

  std::cout << std::endl;
}
```

Problem 8.2 Simpson integration with virtual functions

Implement a new version of your Simpson integration routine using virtual functions:

- Define an abstract base class with a pure virtual `operator()` for function objects in double precision.

- Write a free Simpson integrator function that takes the integrand as a reference to the abstract base class.
- Derive a concrete class from the abstract base class and implement the `operator()`.

Problem 8.3 Benchmark of Simpson integrations

Benchmark the four different versions of the Simpson integration that you have already implemented:

- hard-coded function (Exercise sheet 1),
- function pointer (Exercise sheet 2),
- function template with function objects and type traits (Exercise sheet 4 and 5),
- derived function object (virtual function) (Exercise sheet 8).

We suggest using the optimization flags `-O3 -DNDEBUG -march=native` [1]. Additionally, you can try `-funroll-loops`.

Repeat the benchmarks for functions of different computational complexity: $f_1(x) = 0$, $f_2(x) = 1$, $f_3(x) = x$, $f_4(x) = x^2$, $f_5(x) = \sin(x)$, and $f_6(x) = \sin(5x)$. Discuss the results.

Problem 8.4 Penna Model with Fishing

Read the paper by Moss de Oliveira, Penna, and Stauffer [Moss de Oliveira et al., Physica A 215, 298, 1995][2].

Implement a Penna simulation for a population of fish that is subject to systematic fishing. The goal is to observe how a slight increase in fishing may destroy an initially stable population. The following describes how you should modify your original Penna simulation:

1. At a time-step M_1, when the fish population is stable, introduce the concept of fishing. Here, each fish can die due to fishing (in addition to illness) with probability p_1. At a later time-step $M_2 > M_1$ increase the fishing probability to $p_2 > p_1$.
2. Observe that the increase in fishing at time-step M_2 destabilizes the fish population. Use simulation parameters from the paper: $M = 1$, $T = 3$, genome length $= 32$, $R = 7$, pregnancy probability $= 1$, $p_1 = 0.17$, $p_2 = 0.22$.
3. What happens if fishing is only allowed for the adult species?

As usual, you can base your code on the `Genome`, `Animal`, and `Population` classes from the lecture repository, but working on your own design might prove more rewarding.

In the paper a new parameter M_0 is introduced which you can ignore.

[1] When using `CMake`, perform the benchmarks using the `Release` build type. The build type is controlled via the variable `CMAKE_BUILD_TYPE`. By default, the `Release` build type adds `-O3 -DNDEBUG` to the compiler flags.

[2] The paper is available at https://doi.org/10.1016/0378-4371(95)00039-A. You have to be inside the ETH network in order to download it.