Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programming Techniques for Scientific Simulations Exercise 5

HS 22
Dr. R. Käppeli

Problem 5.1 Penna Model Implementation

In your own repository, you should have the files `genome.hpp` and `animal.hpp` holding the fundamental interfaces to a Penna simulation. You can also continue with the versions from the lecture repository, but working on your own design might prove more rewarding. Implement all the functions listed in the class headers. Make sure they compile and are integrated in a build system.

Problem 5.2 Multidimensional Integration

The goal of this exercise is to implement the Simpson integration on a square $[a, b] \times [c, d]$. We can make use of Fubini's theorem

$$\int_{[a,b]\times[c,d]} f(x,y)d(x,y) = \int_a^b \int_c^d f(x,y)dydx = \int_a^b F(x)dx$$
$$F(x) = \int_c^d f(x,y)dy$$

to express the two-dimensional integral as two consecutive one-dimensional integrals.
We want to re-use the existing 1D Simpson integrator without modification. To implement the 2D integration, implement a function that takes as input

- the integration limits a, b, c, and d,
- the number of bins in the x and y direction,
- a function object representing the function to be integrated,

and returns the computed integral.
Hint: In your function, you will need to implement the functions

$$y \mapsto f(x,y) \qquad \text{and} \qquad x \mapsto \int_c^d f(x,y)dy,$$

where the integral of the second function is approximated by Simpson integration. To achieve this, either write another function object or use `std::bind` or (simplest implementation) `lambdas`.

As a test, check that integrating the unit disk

$$\phi(x,y) = \begin{cases} 1 & \text{if } x^2 + y^2 \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

yields π when integrated over $[-1, 1] \times [-1, 1]$. Then integrate the following density function:

$$\phi(x,y) = \begin{cases} e^{-(x^2+y^2)} & \text{if } x^2 + y^2 \leq R^2 \\ 0 & \text{otherwise} \end{cases}$$

for $R = 1.7606$ over the area where the function is non-zero.

Problem 5.3 Type Traits

In the previous exercise we have written a function performing Simpson integration using an approach based on function objects[1]. The solutions in the repository may be called with function pointers as well, as they also meet the concept requirements of function objects.
We have also pointed out that the templated boundaries are a potential issue if the type for the boundaries does not meet the required concept, e.g., if they were `int`s. In order to fix that, we are going to use traits.
The `integrate` routine shall be templated only on the function object `F`. The boundary types and the result type shall be deduced from the type `F` via traits

- `domain_t<F>`, defining the type for the boundaries, and
- `result_t<F>`, defining the return type for the integrate function.

Each of these traits includes a member type called `type` (accessible via e.g. `domain_t<F>::type`) that specifies the desired type.
In general, `domain_t<F>` and `result_t<F>` assume that `F` has member types called `input_t` and `output_t`, as exemplified by the function object `exp_minus_lambda_x` in the skeleton code. The trait `domain_t` shall define its `type` according to the `input_t` of `F` (i.e. `F::output_t` should be assigned to `domain_t`'s member type `type`), and the trait `result_t` according to the `output_t` of `F`.
However, the traits need to be specialised for functions without those input and ouput types, such as the function object `sin_lambda_x` in the skeleton.
Additionally, both traits also need to be specialised for function pointers with type `R(*)(T)`, where type `T` shall be used by `domain_t` and type `R` by `result_t`.

[1] https://en.cppreference.com/w/cpp/named_req/FunctionObject