Simulation of the interaction between two species

Predator–prey interaction model (Lotka–Volterra): numerical simulator in C++

Mattia Di Nardo Federico Albano

August 17, 2025

1 Design and implementation choices

The project consists of two header files (`simulation.hpp` and `input_utils.hpp`), two source files (`simulation.cpp` and `test_simulation.cpp`), and a `main.cpp`.

The core of the project is the *class* `Simulation`, whose private members are:

- the four parameters of the simulation A, B, C, D;
- the time step dt (fixed at 0.001): the user can choose the total simulation time at runtime, deciding how many integration steps to perform. This avoids problems of population positivity when dt is too large and the initial states are far from equilibrium;
- the equilibrium values of the two populations x_{eq} and y_{eq}, computed from the parameters;
- the "relative" values of the populations with respect to equilibrium x_{rel} and y_{rel};
- a `std::vector` containing all the states visited.

Each state is described by the `struct State`, which contains the absolute values x, y and the energy H.

The class has two constructors: the first initializes the four parameters and also the initial values of the two populations (to create a simulation to be evolved), while the second initializes only the parameters (used in tests where no evolution is needed).

Main public methods

- `compute_H(x,y)`: given a state, computes the energy H;
- `evolve()`: evolves the simulation by a single dt;
- `run(n)`: performs n integration steps;
- `get_results()`: returns the vector of visited states.

In `input_utils.hpp` a template function `read_param()` is declared and defined, used for input handling. Its purpose is to make the program fail if any input does not match the specified type.

2 Execution instructions

After configuring and building the project, the program can be executed with:

```
./build/Debug/progetto
# or
./build/Release/progetto
```

The program asks from *stdin* for the parameters $A, B, C, D > 0$, the initial conditions $x_0, y_0 > 0$, and the number of steps (integer ≥ 0). It produces a `results.txt` file in the current folder.

3 Input parameters and output format

Input Values required in order: A, B, C, D (all > 0), x_0, y_0 (positive densities), number of `steps` integer ≥ 0. If the inputs do not satisfy the positivity condition, an exception is thrown during class construction and the program terminates.

Output File `results.txt` with three columns `x y H`, representing the system states at each dt.

4 Interpretation of results

The invariant H is exactly constant in the continuous model, but the integration scheme (explicit Euler), used to discretize the differential equations, introduces a numerical error with a *drift* of H (normally small for sufficiently small dt). Therefore, H does not always remain constant, especially if the initial state is far from equilibrium.

5 Testing strategy

To verify that the results are reasonably error-free, five `TEST_CASE` were implemented, one for each of the main features:

- test for the correct initialization of a simulation and the rejection of parameters that do not satisfy the physical conditions of the model;
- test for the correct computation of energy H with different parameter combinations;
- test for the methods `evolve()` and `run()`: they preserve population positivity, generate a vector of the expected size; `run(0)` does not change the state and `run(1)` is equivalent to `evolve()`;
- test for the quasi-invariant conservation of H, which remains nearly constant within a reasonable tolerance over many iterations, starting from both near and far equilibrium initial states;
- test for the parameter input function.

6 Limitations and possible improvements

- The explicit Euler method is not *structure-preserving*: H does not remain exactly constant.
- To reduce the drift of H, one could change the integration scheme (exponential or semi-implicit integrator), modifying the formulas in `evolve()`.

7 Declaration of use of generative AI

During development, generative AI tools were used for code review and design suggestions. Implementation decisions and validation of the results are the responsibility of the authors.

8 Other useful information

The project is available on GitHub: `https://github.com/mdinardo12/lotka_volterra`.