Simulazione dell'interazione tra due specie

Modello di interazione predatori–prede (Lotka–Volterra): simulatore numerico in C++

Mattia Di Nardo Federico Albano

17 agosto 2025

Scelte progettuali e implementative

Il progetto è composto da due file header (`simulation.hpp` e `input_utils.hpp`), due file sorgente (`simulation.cpp` e `test_simulation.cpp`) e un `main.cpp`.

Il cuore del progetto è la *classe* `Simulation`, che ha come membri privati:

- i quattro parametri della simulazione A, B, C, D;
- il passo temporale dt (fissato a 0,001): l'utente può scegliere il tempo totale della simulazione a runtime, decidendo quanti step di integrazione eseguire. In questo modo si evitano problemi di positività delle popolazioni per dt troppo grandi e stati iniziali molto lontani dall'equilibrio;
- i valori di equilibrio delle due popolazioni x_{eq} e y_{eq}, calcolati a partire dai parametri;
- i valori "relativi" delle popolazioni rispetto all'equilibrio x_{rel} e y_{rel};
- un `std::vector` che contiene tutti gli stati attraversati.

Per descrivere ogni stato si usa la `struct State`, che contiene i valori assoluti x, y e l'energia H.

La classe ha due costruttori: il primo inizializza i quattro parametri e anche i valori iniziali delle due popolazioni (per creare una simulazione che si vuole far evolvere), mentre il secondo inizializza solo i parametri (usato nei test in cui non serve evolvere).

Metodi pubblici principali

- `compute_H(x,y)`: dato uno stato, calcola l'energia H;
- `evolve()`: fa evolvere la simulazione di un singolo dt;
- `run(n)`: esegue n passi di integrazione;
- `get_results()`: restituisce il vettore degli stati visitati.

In `input_utils.hpp` è dichiarata e definita una funzione template `read_param()`, usata nella ricezione degli input. Il suo scopo è far fallire il programma se uno degli input non rispetta il tipo specificato.

Librerie esterne da installare

I test usano `doctest.h` (header-only) incluso nel repository; non servono pacchetti esterni.

Istruzioni per l'esecuzione

Dopo aver configurato e buildato il progetto, si può eseguire il programma con:

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

Il programma chiede da *stdin* i parametri $A, B, C, D > 0$, le condizioni iniziali $x_0, y_0 > 0$ e il numero di passi (intero ≥ 0). Produce un file `results.txt` nella cartella corrente.

Parametri di input e formato di output

Input Valori richiesti in ordine: A, B, C, D (tutti > 0), x_0, y_0 (densità positive), numero di `steps` intero ≥ 0. Se gli input non rispettano la condizione di positività, al momento della costruzione della classe viene lanciata un'eccezione e il programma termina.

Output File `results.txt` con tre colonne `x y H`, rappresentanti gli stati del sistema ogni dt.

Interpretazione dei risultati

L'invariante H è esattamente costante nel modello continuo, ma lo schema di integrazione (Euler esplicito), usato per discretizzare le equazioni differenziali, introduce un errore numerico con *drift* di H (di norma piccolo per dt sufficientemente piccolo). Perciò H non rimane sempre costante, in particolare se lo stato iniziale è molto lontano dall'equilibrio.

Strategia di test

Per verificare che quanto ottenuto sia ragionevolmente esente da errori, sono stati implementati cinque `TEST_CASE`, uno per ciascuna delle principali funzionalità:

- test per la corretta inizializzazione di una simulazione e il rifiuto dei parametri che non rispettano le condizioni fisiche del modello;
- test per il corretto calcolo dell'energia H con diverse combinazioni di parametri;
- test per i metodi `evolve()` e `run()`: preservano la positività delle popolazioni, generano un vettore della dimensione attesa; `run(0)` non genera cambiamenti allo stato e `run(1)` equivale a `evolve()`;
- test per la conservazione quasi-invariante di H, che resta quasi costante entro una tolleranza ragionevole su molte iterazioni, a partire da stati iniziali sia vicini sia lontani dall'equilibrio;
- test per la funzione di ricezione dei parametri.

Limitazioni e possibili miglioramenti

- Il metodo Euler esplicito non è *structure-preserving*: H non resta esattamente costante.
- Per ridurre il drift di H, si potrebbe cambiare schema di integrazione (integratore esponenziale o semi-implicito), modificando le formule in `evolve()`.

Dichiarazione di uso di IA generativa

Durante lo sviluppo sono stati utilizzati strumenti di IA generativa per code review e suggerimenti di design. Le decisioni implementative e la validazione dei risultati sono a cura degli autori.