Simulazione di un'epidemia secondo il modello SIR ed automa cellulare

Sara Maiano
Giovanna Mariapia Monaldi
Federico Tronchi

Agosto 2023

Introduzione

Il programma qui presentato è stato ideato come progetto d'esame del corso di Programmazione per la Fisica presso l'università di Bologna, anno accademico 2022/23. Scritto interamente in C++, il suo obiettivo è quello di ricreare, graficamente e non, con due diversi approcci, l'andamento di un'epidemia in un sistema chiuso.

Indice

1 Presentazione del programma

1.1 Parte I: Il modello SIR

La prima parte in particolare sviluppa su due file, `sir.hpp` e `sir.cpp`, il modello matematico SIR per l'evoluzione di una epidemia in una popolazione chiusa. Il nome di questo modello è l'acronimo dei tre stati che vengono considerati per ogni individuo della popolazione:

- S : "Susceptible", le persone soggette a una possibile infezione;
- I : "Infectious", le persone che hanno già contratto l'infezione, diventando così contagiose;
- R : "Removed", le persone guarite, messe in isolamento oppure morte.

In questo modello quindi si esclude la possibilità di riprendere l'infezione, e lo stato di una persona può proseguire solamente in una direzione precisa: da suscettibile ad infetto e da infetto a rimosso.

Nell'*header file* `sir.hpp` si include innanzitutto `<vector>` che permette l'utilizzo del vettore della *Standard Library*. Si definisce in seguito una struttura `Param` che contiene in sé i parametri del modello SIR:

- beta : $\beta \epsilon [0, 1]$, che misura la probabilità di contagio, per cui il passaggio per l'individuo dallo stato di suscettibile a quello di infetto;
- gamma : $\gamma \epsilon [0, 1]$, che misura invece la probabilità di guarigione o decesso, dunque regola il passaggio per l'individuo da infetto a rimosso.

Viene anche definita una struttura `State` che contiene gli stati, sopra citati, in cui gli individui possono trovarsi.

La dichiarazione della classe SIR rappresenta invece il modello SIR stesso e vi sono stati inclusi due metodi pubblici:

```
get_total ed evolve
```

Il primo metodo viene utilizzato per ottenere il numero totale degli individui mentre il secondo metodo per far evolvere il modello per un certo numero di giorni.
Nel file `sir.cpp` vengono invece implementati i costruttori e definiti i metodi dichiarati nell'*header file* ed in particolare, nella funzione `evolve`, troviamo le

tre equazioni che regolano l'andamento dell'epidemia secondo il modello matematico.
Queste tre equazioni sono ricavate a partire dalle tre equazioni differenziali proprie del modello SIR. Queste vengono discretizzate e considerando $\Delta T = 1$ si ottengono le seguenti equazioni impiegate per calcolare lo sviluppo dell'epidemia:

$$S_i = S_{i-1} - \beta \frac{S_{i-1}}{N} I_{i-1} \tag{1}$$

$$I_i = I_{i-1} + \beta \frac{S_{i-1}}{N} I_{i-1} - \gamma I_{i-1} \tag{2}$$

$$R_i = R_{i-1} + \gamma I_{i-1} \tag{3}$$

Queste tre equazioni sono accompagnate da due condizioni essenziali che devono essere rispettate durante l'intera evoluzione dell'epidemia:

- il numero totale degli individui deve rimanere costante :

$$S + I + R = N$$

- il valore assegnato ad ogni gruppo di individui (suscettibili, infetti o rimossi) deve essere un numero naturale:

$$S, I, R \epsilon N$$

Per fa sì che la prima condizione venga rispettata, il programma calcola dopo ogni evoluzione il numero totale degli individui e se questa somma è maggiore di N riduce il numero dei suscettibili, mentre se questa somma è minore di N aumenta il numero di infetti.

Il controllo della seconda condizione avviene invece nel main quando vengono presi in input da utente i valori per s, i ed r. Per maggiori informazioni si rimanda alla sezione 1.3.

1.2 Parte II: Infezione del singolo individuo

1.2.1 Schematizzazione degli individui come celle

Per la seconda parte ci si è discostati dal primo modello matematico utilizzato, che guardava alla popolazione nella sua interezza, e ci si è concentrati a definire lo stato di ogni singolo individuo. In `cell.hpp` vengono dichiarate:

- un enumeration `Condition`: che rappresenta gli stati possibili per una cella (Susceptible, Infectious, Removed, Void). Si nota la prima differenza con la prima parte, dove non esisteva lo stato "nullo" definito nella seconda parte come "void". È usata una *scoped enumeration*, dichiarata quindi come `enum class` per evitare conversioni implicite.

- Una classe `Cell`: in cui vengono definiti due costruttori, rispettivamente il primo per inizializzare la condizione di una cella come vuoto di default, e il secondo per attribuire ad ogni cella una delle altre condizioni. Vengono utilizzati inoltre due metodi:
  - `get_condition`: utilizzato per ottenere lo stato attuale di una cella;
  - `set_condition`: utilizzato per assegnare un nuovo stato ad una cella.

I costruttori di `Cell` e le definizioni dei metodi sopra elencati vengono implementati in `cell.cpp`.

1.2.2 Rappresentazione delle celle in una griglia

Per la rappresentazione dello stato di contagio di una popolazione si è pensato di utilizzare una griglia in cui ogni cella rappresenta un individuo o rimane vuota. Nel file `grid.hpp` vengono dichiarati tutti i dati membri, i metodi, i costruttori e una funzione implementati successivamente nel file `grid.cpp`.
È stata creata una classe `Grid` contenente i dati membri: `m_side`, che indica il lato della griglia e `m_cells`, di tipo `std::vector<Cell>` contenente propriamente gli individui. Essa contiene poi i seguenti metodi pubblici:

- `get_side`: per ottenere la dimensione del lato della griglia;
- `get_cell`: per ottenere una referenza modificabile o una referenza costante[1], ad una cella specifica indicata dalla posizione `cell_pos`;
- `count_s, count_i, count_r, count_voids` : che contano all'interno della griglia quante celle ci sono per ogni stato definito, utilizzando l'algoritmo `std::accumulate`;
- `valid_coord`: per verificare che le coordinate di ogni cella, riga e colonna, siano positive e rientrino nelle dimensioni della griglia stessa;
- `fill`: un metodo che assegna a un tot (determinato dalla variabile s) di celle lo stato di suscettibili e a un tot (determinato dalla variabile i) di celle lo stato di infetti, e che successivamente mescola le celle nella griglia utilizzando la funzione `random_value` che genera valori casuali compresi tra 0 e 1;
- `inf_neigh`: che conta per ogni cella individuata da una precisa posizione, il numero di celle infette a confine con essa;
- `evolution`: che simula l'evoluzione della griglia e quindi di un'infezione guidata dai parametri beta e gamma.

[1] La possibilità di ottenere una referenza costante o modificabile è data dal fatto che sono state dichiarate due funzioni `get_cell` entrambe con lo stesso nome.

Nel file `main.grid.cpp` la funzione main gestisce l'evoluzione vera e propria della griglia e utilizza la libreria esterna SFML per la risposta grafica all'utente. In questa parte di codice infatti vengono richiesti in input da terminale all'utente i parametri necessari all'avviamento del programma. In risposta l'utente riceverà un determinato output, ma per maggiori informazioni a riguardo rimandiamo alla sezione 4.3 .

1.2.3 Parte grafica

La parte grafica prende vita nei due file `graphics.hpp` e `graphics.cpp`. All'interno di questi due file viene innanzitutto inclusa una libreria grafica esterna `SFML`, viene poi definita una classe `Graph` in cui viene dichiarato un metodo pubblico `draw_grid`. La funzione `draw_grid` calcola le dimensioni di ogni cella in relazione alle dimensioni della griglia indicate precedentemente dall'utente e in relazione alle dimensioni della finestra grafica, in questo caso 800x600. Utilizzando le coordinate di riga e colonna per ogni cella viene riconosciuto poi lo stato da assegnare e in funzione di quest'ultimo viene assegnato un colore specifico, in particolare:

- per i suscettibili viene impostato il colore verde;
- per gli infetti viene impostato il colore rosso;
- per i rimossi il colore blu;
- per le celle vuote viene impostato il colore bianco.

1.3 Validazione degli input

Per controllare che i valori inseriti come input da utente rispecchiassero le condizioni necessarie, si è scelto di creare un file `validate_input.hpp` in cui sono state dichiarate le funzioni poi definite nel file `validate_input.cpp`.

Per il controllo della casistica in cui il parametro beta è minore del parametro gamma si è implementata una funzione `valid_R0` che attraverso un ciclo *while* continua a chiedere all'utente di inserire un valore di beta che sia maggiore di gamma e quando viene inserito un valore corretto, commenta all'utente che se beta è maggiore di gamma allora la pandemia può partire.
Allo stesso modo, con una funzione `valid_SIR`, è stata controllata la condizione per cui, la somma dei suscettibili e degli infetti che l'utente deve inserire per la seconda parte, fosse necessariamente minore della dimensione della griglia *side*side*.

Per controllare invece che l'utente inserisse da terminale dei valori corretti per i parametri dei suscettibili, degli infetti, dei rimossi, di beta e gamma, dei giorni di durata dell'epidemia e della lunghezza del lato della griglia, si è utilizzata una strategia differente.

Per ognuno dei parametri è stata implementata una funzione che lavora attraverso un ciclo *while* e attraverso un *try-catch*. Nel ciclo *while* si verifica che vengano inseriti numeri corretti, ed in caso questi non lo fossero, ad opera dell'*else* parte un messaggio d'errore che permette all'utente di correggere il valore da assegnare al parametro secondo le giuste indicazioni. Il *try-catch* entra in gioco invece quando vengono inseriti caratteri non numerici per cui anche in questo caso viene stampato un messaggio d'errore che permette all'utente di correggersi.
Entrambi i tipi di funzione sono stati utilizzati con lo scopo di non bloccare il programma nel momento in cui vengono inseriti valori non validi per i parametri, ma di permettere all'utente di correggersi e proseguire da quel punto.

2 Strumenti di sviluppo

2.1 CMake

CMake è un sistema di build che ha permesso di gestire gli eseguibili, le librerie incluse e i file di test presenti nel progetto. È stato creato il file di configurazione CMakeLists.txt in cui sono contenute le istruzioni che permettono la compilazione dell'intero programma. In particolare in questo file:

- vengono creati gli eseguibili delle due parti del progetto, 'sir' e 'grid';
- viene abilitato il supporto per i test;
- vengono aggiunte le opzioni `-Wall` e `-Wextra` che fanno comparire degli avvertimenti (*warning*) durante la compilazione se vengono individuati errori;
- viene richiesta la libreria esterna SFML;
- viene abilitato l'*Address Sanitizer*, strumento che individua eventuali problemi di memoria.

2.2 Clang format

Clang Format è uno strumento di formattazione utilizzabile per diversi linguaggi tra cui C++. Esso aiuta quindi a rendere omogenea l'organizzazione visiva del codice e fa sì che ci sia coerenza anche se più persone lavorano sugli stessi file. Per formattare il codice dei file componenti il progetto è stata usata una configurazione di default di Clang Format, presente nel file `.clang-format`.

2.3 Git e GitHub

Lavorando in gruppo, è stato opportuno avvalersi di uno strumento che tenesse traccia delle modifiche apportate al codice e che facilitasse la collaborazione tra i membri. La scelta è ricaduta su Git, un sistema di version control, ed è stato utilizzato in concomitanza a GitHub, una piattaforma dove possono essere conservate *repository* Git.

Dopo aver fatto le modifiche su una *repository* locale, si procedeva con la richiesta di approvazione di tali modifiche e un successivo caricamento di queste ultime sulla *repository* remota di GitHub. Queste operazioni sono state fatte sfruttando coniugando l'uso comandi di Git quali `-commit` e azioni di GitHub quali le *Pull Requests*.
La *repository* GitHub utilizzata durante lo sviluppo del progetto è accessibile con il seguente link: repository

3 Test

Per entrambe le parti del progetto sono stati eseguiti dei test per verificare che le funzioni svolgessero i compiti attesi, si tratta quindi di *unit tests*. Lo strumento usato a tal proposito è stato **doctest**, un framework di testing *single-header*. I file dedicati ai test includono il file `doctest.h` e sono organizzati in `TEST_CASE` e `SUBCASE`.

3.1 Test per la I parte

Per la prima parte del progetto i test si trovano nel file `sir.test.cpp`. In questo file viene come primo test verificato che la funzione `get_total` funzioni come dovrebbe. Poi viene testato il corretto funzionamento dell'operatore di comparazione sia nel caso in cui debba restituire un'ugualianza rispettata che nel caso contrario. Infine vengono testati i valori generati dalla funzione principale che simula l'evoluzione del modello SIR per zero giorni di epidemia, un giorno, due giorni e tre.

3.2 Test per la II parte

I test per la seconda parte del progetto si trovano invece nel file `grid.test.cpp`. Viene testato il corretto funzionamento delle funzioni di cui si è parlato nel dettaglio nella sezione 1.2.1 .

4 Istruzioni per l'utilizzo

4.1 Istruzioni per la compilazione

Essendo il progetto composto da più file sorgente, è necessario che quest'ultimo venga compilato utilizzando CMake (di cui si è riportata una spiegazione dettagliata nella sezione 2.1). Per la compilazione del progetto quindi devono essere utilizzati i seguenti due comandi che utilizzano CMake in modalità di debug:

1. `cmake -S . -B build -DCMAKE_BUILD_TYPE=Debug`
2. `cmake --build build`

Se dopo la compilazione si volessero far partire i test di cui si parla nella sezione 3, sarà sufficiente mandare il seguente comando:

`cmake --build build --target test`.

4.2 Guida per la I parte

Se dopo aver compilato si desidera proseguire avviando la prima parte del programma, il comando da eseguire sarà il seguente :

`./build/sir`

Tutti i parametri necessari al programma sono presi in input da terminale, per cui all'utente verranno stampate a schermo una serie di richieste in cui gli verrà chiesto di assegnare un valore per i seguenti parametri :

- il parametro beta : *beta*
- il parametro gamma : *gamma*
- il numero dei suscettibili : s
- il numero degli infetti : i
- il numero dei rimossi : r
- il numero dei giorni per cui si vuole far evolvere l'epidemia : *epidemic duration*.

In caso di inserimenti non corretti per i valori richiesti, l'utente riceverà dei messaggi di warning di cui si è parlato nella sezione 1.3 . Quando i valori necessari saranno tutti corretti la risposta di output per l'utente verrà stampata interamente su terminale, restituendo lo stato dell'epidemia giorno per giorno. Un esempio di inserimento dati potrebbe essere il seguente :

```
insert beta:  0.6
insert gamma:  0.3
Since beta > gamma, R0 > 1, the epidemic starts .
insert susceptibles:  130
insert infectious:  20
insert removed:  0
insert epidemic duration in days:  2
```

Con un set di dati come quello d'esempio, la risposta in output all'utente sarà la seguente :

```
total:  150
day:  0
s:  130
i:  20
r:  0
total:  150
-----
```

```
day:  1
s:  120
i:  24
r:  6
-----
day:  2
s:  108
i:  29
r:  13
total:  150
-----
the epidemic lasts for:  2 days
```

4.3 Guida per la II parte

Se dopo la compilazione, si desidera proseguire avviando la seconda parte del programma, il comando da eseguire sarà il seguente:

`./build/grid`

Come per la prima parte, anche in questo caso verrà richiesto all'utente di inserire in input da terminale i parametri necessari :

- la dimensione del lato della griglia : *grid side*
- il numero dei suscettibili : s
- il numero degli infetti : i
- il parametro beta : *beta*
- il parametro gamma : *gamma*
- il numero dei giorni per cui si vuole far evolvere l'epidemia : *epidemic duration*

Nel caso di inserimenti non corretti per i valori richiesti, l'utente riceverà dei messaggi di *warning* di cui si è parlato nella sezione 1.3. Quando i valori da inserire in input saranno tutti corretti, la risposta all'utente verrà stampata in output sia da terminale, dove verrà mostrato giorno per giorno lo stato epidemico, così come veniva fatto nella prima parte; sia tramite una finestra grafica che mostrerà l'evoluzione generale dell'epidemia sequenziando giorno dopo l'altro la griglia in evoluzione.

Un esempio di inserimento dati potrebbe essere il seguente :

grid side: 15
s: 220
i: 5
beta: 0.5
gamma: 0.35
epidemic duration : 2

per cui all'utente la schermata apparirà in questo modo :

grid side: 15
The grid has a total of 225 cells
s: 220
i: 5
Since (S + I) < (side*side) the epidemic starts
Initial number of S: 220
Initial number of I: 5
Initial number of R: 0
Initial number of Void: 0
beta: 0.5
gamma: 0.35
Since beta > gamma, R0 > 1, the epidemic starts
epidemic duration : 2

La risposta in output che dovrebbe apparire sul terminale con un set di dati come quello d'esempio è la seguente :

day: 1
Number of S: 200
Number of I: 24
Number of R: 1
Number of Void: 0
day: 2
Number of S: 152
Number of I: 61
Number of R: 12
Number of Void: 0

Quando il programma avrà terminato l'evoluzione dell'epidemia all'utente potrebbero apparire sul terminale degli *allert sanitizer*, di cui non dovrà curarsi in quanto propri della libreria esterna SFML utilizzata nell'esecuzione del programma.

L'utente potrà notare inoltre utilizzando lo stesso set di dati per la prima e la seconda parte, che queste producono risultati diversi, in quanto l'evoluzione dell'epidemia nei due casi è regolata da schemi logici differenti.
Nella prima parte sono le equazioni di cui si è parlato nella sezione 1.1 a regolare l'evoluzione dell'epidemia per la popolazione nel suo complesso. Per la seconda parte invece l'automa cellulare mostra l'andamento dell'epidemia nel complesso solo in relazione all'evoluzione dell'infezione per i singoli individui che rispondono unicamente della situazione a loro strettamente circostante.