RELAZIONE SUL PROGETTO D'ESAME

Corso di Programmazione per la Fisica

Autori: Francesco Galasso, Lorenzo Tomassoni

Sommario

1. SCOPO DEL PROGETTO

Lo scopo del progetto è lo sviluppo in C++ di una simulazione della diffusione di un'epidemia in una popolazione. In una prima parte del lavoro svolto si è puntato a implementare le equazioni del modello teorico SIR (Susceptibles, Infectious, Removed), descritto più approfonditamente nella sezione "Descrizione generale e scelte implementative". In seguito è stata sviluppata una seconda parte con l'obiettivo di rappresentare graficamente (con la libreria grafica esterna SFML) la diffusione epidemica in una popolazione in movimento all'interno di una griglia 40x40, con la possibilità di avere individui vaccinati più resistenti al contagio.

2. DESCRIZIONE GENERALE E SCELTE IMPLEMENTATIVE

2.1. Modello SIR

Il modello SIR è uno dei più semplici, ma più usati, modelli teorici per la diffusione epidemica. In questo modello la popolazione studiata viene considerata come composta da:

- Suscettibili (S), le persone che ancora non hanno contratto la malattia
- Infetti (I), le persone che hanno contratto la malattia e che possono contagiare le altre
- Rimossi (R), le persone morte (per la malattia) o guarite e immuni

Per ogni individuo gli unici trasferimenti possibili sono:
Suscettibile → Infetto → Rimosso
Il modello, nella sua versione più semplice, è regolato da tre equazioni differenziali:

$$\frac{dS}{dt} = -\beta \cdot \frac{S}{N} \cdot I \qquad \frac{dI}{dt} = \beta \cdot \frac{S}{N} \cdot I - \gamma \cdot I \qquad \frac{dR}{dt} = \gamma \cdot I$$

Sono presenti due parametri, β e γ, entrambi con valore compreso tra 0 e 1:

- β rappresenta la probabilità di contagio in seguito a contatti tra Infetti e Suscettibili (S → I)
- γ rappresenta la probabilità di rimozione di Infetti (I → R)

In fase di implementazione si è scelto di tenere tali parametri costanti. Per ottenere predizioni più accurate, è possibile considerarli come variabili, in base al comportamento della popolazione e a misure di mitigazione messe in atto.
Notiamo che le tre equazioni differenziali rispettano sempre la condizione $S + I + R = N$ (N è il numero totale di persone). Inoltre l'epidemia sarà in espansione (cioè $\frac{dI}{dt} > 0$) se $\beta \cdot \frac{S}{N} > \gamma$, in contrazione (cioè $\frac{dI}{dt} < 0$) se $\beta \cdot \frac{S}{N} < \gamma$. Se quindi definiamo $R_0 = {}^{\beta}/_{\gamma}$ e consideriamo $S \approx N$, le fasi di espansione saranno caratterizzate da $R_0 > 1$, quelle di contrazione da $R_0 < 1$.

2.2. Prima parte

Per l'implementazione delle equazioni differenziali del modello SIR si è utilizzata la discretizzazione:

$$S_i = S_{i-1} - \beta \cdot \frac{S_{i-1}}{N} \cdot I_{i-1} \qquad I_i = I_{i-1} + \beta \cdot \frac{S_{i-1}}{N} \cdot I_{i-1} - \gamma \cdot I_{i-1} \qquad R_i = R_{i-1} + \gamma \cdot I_{i-1}$$

Con i che indicizza le iterazioni compiute e (ovviamente) $S, I, R \in \mathbb{N}$.
In first_part.cpp si può trovare il main() della funzione, descritto di seguito.
Si prendono inizialmente in input i parametri β e γ, la composizione iniziale della popolazione in S, I, R e la durata della simulazione (vale a dire quante iterazioni al massimo verranno compiute). Questo è fatto attraverso le funzioni insert_parameter() e insert_people(), che prendono dall'utente i valori richiesti attraverso standard input, stampano un messaggio di errore qualora i valori inseriti non fossero coerenti con la richiesta fatta (per esempio in caso di inserimento di

caratteri non numerici o nel caso di valori negativi) e ritornano il valore correttamente inserito oppure sollevano un'eccezione in caso di improvvisa terminazione dell'input.
I valori dei parametri β e γ vengono tenuti in una struct "Parameters", mentre S, I, R sono gli attributi privati di un oggetto di tipo SIR, classe implementata in sir.hpp, con metodi per ottenere e impostare i tre attributi e un operatore "==".
Nella fase di calcolo, in ogni iterazione, si ottengono inizialmente i valori "floating point" per S, I, R, poi, per mantenere il numero di persone per ogni categoria intero, li si tronca all'unità e si "distribuisce" la somma di tutte le parti decimali, mantenendo l'interezza di S, I, R e il fatto che la loro somma deve rimanere costante.
Si conserva in memoria anche una copia del precedente risultato in modo che se 2 iterazioni consecutive danno lo stesso risultato si esca da loop di calcolo, perché le successive iterazioni forniranno sempre gli stessi risultati. L'utente viene informato attraverso standard output dell'avvenuta "convergenza".
Dopo aver stampato a schermo i risultati, questi vengono anche scritti in un file "results .txt".

2.3. Seconda parte

Nella seconda parte del progetto è stata implementata una visualizzazione della diffusione dell'epidemia tramite SFML: un'interfaccia di programmazione per lo sviluppo di applicazioni grafiche. In particolare la popolazione sotto studio viene raffigurata come un insieme di quadratini in una griglia 40x40 larga 720x720 pixel. Sotto la griglia è presente un contatore che tiene conto dell'evoluzione del numero di persone Suscettibili, Infette e Rimosse (la finestra è quindi grande 720x775 pixel).

Ogni "persona" nella popolazione viene individuata come un oggetto di tipo Person (classe dichiarata in person.hpp) ed è caratterizzata dagli attributi (privati):

- x, cioè la "cella" occupata nella griglia 40x40 (le celle sono indicizzate da 0 a 1599, da sinistra verso destra, dall'alto verso il basso)
- px e py, cioè le 2 componenti della quantità di moto, che possono prendere valori 0 o ±1 (con gli assi "orientati" verso destra e verso l'alto)
- s, lo stato di "salute", implementato con un enum Status che può essere o Status::s (Suscettibile) oppure Status::i (Infetto)
- is_vaccinated, un bool che indica lo stato di vaccinazione della persona ("false" di default)
- dot, un sf::RectangleShape che sarà la rappresentazione grafica della persona (come si può capire dal namespace, è una classe di SFML), se la persona è Suscettibile il colore sarà verde, se Infetta rosso, se Suscettibile e vaccinata blu

Oltre ai metodi per ottenere o modificare alcuni attributi è definito un metodo vaccine() che fa diventare "true" l'attributo is_vaccinated e imposta il colore blu per il dot, un metodo set_dot_color() che imposta liberamente il colore del dot e un metodo update_dot_position(), che pone il dot nel posto corretto della griglia a partire dalla sua posizione x.
Dato che con grandi popolazioni le persone tendono a sovrapporsi nelle celle della griglia, per evitare (almeno in parte) questo fenomeno, le dimensioni del dot sono minori di quelle della cella e in update_dot_position() il dot viene spostato ai margini della cella a seconda della direzione del suo moto (per esempio se la persona "arriva" in una cella da destra, il dot viene spostato al

margine destro, se “arriva” in una cella da sinistra, il dot viene spostato al margine sinistro, eccetera…).

La popolazione invece è individuata come un oggetto di tipo “Population” (classe dichiarata all’interno di population.hpp) e ha gli attributi (privati):

- people, un std::vector di oggetti Person, il “contenitore” per tutte le persone
- beta, il parametro β di cui si è parlato sopra
- gamma, il parametro γ di cui si è parlato sopra
- vaccination_campaign, un valore tra 0 e 1, che indica la frazione di persone che devono diventare Rimosse prima di far partire la campagna vaccinale
- total_susceptibles, il valore (costantemente aggiornato) di Suscettibili nella popolazione (utile per il contatore a schermo di S, I, R)
- original_size, la quantità di persone presenti nel vector people all’inizializzazione (utile per il contatore a schermo di S, I, R e per le vaccinazioni)

I metodi più rilevanti della classe Population sono: evolve(), infection(), vaccination(), death(std::vector<int>).
Il metodo evolve() è quello che viene chiamato ad ogni iterazione del loop di gestione dell’intefaccia grafica nel main() (in second_part.cpp). Questo metodo assegna alle componenti della quantità di moto di ogni singola persona dei valori casuali tra 0 e ±1 (si potrebbero quindi considerare le singole persone come dei “random walkers”), aggiorna gli attributi di ogni singola persona (anche chiamando update_dot_position()) e infine chiama i metodi infection() e vaccination(). La generazione avviene con un std::default_random_engine (della libreria di numeri random, definita nell’header <random>) a cui ogni volta viene fornito un seed dato da std::random_device; si è scelto, data la semplicità del problema, di usare una distribuzione uniforme (std::uniform_int_distribution<>).
Il metodo infection() è quello che si occupa di gestire i contagi tra persone che occupano la stessa cella della griglia. Innanzi tutto, con un loop su tutta la popolazione, si stabilisce, tramite estrazione casuale di un numero tra 0 e 1, quali persone Infette diventano Rimosse: i loro indici nel vector people vengono aggiunti a un vector di interi che viene passato al metodo death() (il quale rimuoverà da people le persone indicate). In seguito, con due for-loops “annidati”, si cercano le persone che occupano la stessa cella e, anche qui con estrazione casuale di un numero tra 0 e 1, si stabilisce se avvenga o no il contagio tra due persone e, se necessario, si pone lo stato di salute di una delle due da Status::s a Status::i, si cambia il colore del relativo dot e si aggiorna total_susceptibles. Se una persona è vaccinata, si moltiplica il numero “estratto” (da confrontare con β) per 1.3, questo vuol dire che ogni persona vaccinata non avrà probabilità β di ammalarsi, ma $\beta/_{1.3}$. Si può quindi dire che vaccinare persone fa abbassare il coefficiente R_0 (infatti una popolazione di soli vaccinati avrà $R_0' = \beta/_{1.3 * \gamma} < \beta/_{\gamma} = R_0$).
Il coefficiente 1.3, stabilito arbitrariamente dagli autori, comporta una riduzione della probabilità di infezione per i vaccinati di $\sim 23\%$; nonostante questa possa essere in alcuni casi una sottostima, gli effetti della vaccinazione (meglio se tempestiva) anche così sono apprezzabili.
Il metodo vaccination() si occupa di “vaccinare”, ogni volta che viene chiamato, l’1% della popolazione, indipendentemente dallo stato di salute. Si noti che si vaccinano persone solo se si è raggiunta una certa quota di Rimossi, stabilita da vaccination_campaign, numero compreso tra 0 e

1 (è necessario che i Rimossi cioè $original\_size - actual\_size$ siano più della quota $vaccination\_campaign * original\_size$). Se non si vogliono vaccinazioni è quindi sufficiente porre vaccination_campaign = 1.
Il metodo death prende in input un vector di interi e rimuove le corrispondenti persone infette da people. Il loop che esegue questo compito scorre la lista degli indici delle persone da rimuovere al contrario, per evitare di rimuovere la persona sbagliata o di accedere a elementi di memoria fuori dal vector (perché, rimuovendo una persona, le successive hanno tutte l'indice minorato di uno).

Il main() della seconda parte (in second_part.cpp) inizia prendendo in input i parametri β e γ e i valori di Suscettibili e Infetti, grazie alle funzioni insert_parameter() e insert_people(). Ci si assicura che il numero totale di persone non superi 1600 utilizzando un while-loop che porta al reinserimento dei valori in input in caso si superi il limite. Si chiede quindi all'utente se voglia visualizzare la griglia e se voglia avere le vaccinazioni nella simulazione, la risposta viene presa in standard input con una funzione insert_y_n() (definita in insert_function.cpp), molto simile a quelle già presentate. Nel caso in cui si voglia mostrare la griglia si istanzia un vettore di sf::VertexArray che contenga tutte le linee che comporranno la griglia (una linea, larga un pixel, in SFML è ottenuta con un sf::VertexArray con 2 vertici).
Nel caso in cui si voglia tenere conto dei vaccini, verrà chiesto all'utente di inserire il valore della percentuale di rimossi necessaria per avviare la vaccinazione, assegnando alla variabile vax_pct un valore compreso tra 0 e 1.
Viene quindi generata una popolazione attraverso la funzione (definita sempre in second_part.cpp) random_population_generator(int, int, double, double, double). Questa funzione prende in input i parametri inseriti dall'utente (Suscettibili, Infetti, β, γ e vax_pct), genera un vector di Person che abbia il numero richiesto di Suscettibili e di Infetti in posizioni casuali ($x \in [0, 1599]$) e con quantità di moto casuali ($px, py \in [-1, 1]$); ritorna quindi un oggetto Population con il vector generato come "people" e i valori forniti di β, γ e vax_pct come "beta", "gamma" e "vaccination_campaign".
Dopo aver inizializzato la finestra in cui verrà mostrata l'evoluzione del programma (di tipo sf::RenderWindow), si entra in un while-loop dedicato alla gestione della finestra. Per ogni iterazione si controlla che la finestra non sia stata chiusa e poi si chiama il metodo evolve() sulla popolazione studiata; viene chiamato il metodo (della finestra) clear(), che rimuova il contenuto del "frame" precedente, si disegnano (draw()) tutte le persone e, dopo aver ottenuto i valori "attuali" di suscettibili infetti e rimossi con i metodi get_susceptibles(), get_infectious() e get_removed() si scrivono a schermo questi valori.
La scrittura avviene con oggetti di tipo sf::Text, da inizializzare con un font (sf::Font): l'utente può trovare nella cartella con i file scaricati anche un file helvetica.otf, che permette di scrivere con il font Helvetica. Si chiama infine il metodo display() che mostra effettivamente a schermo ciò che è stato disegnato.

3. ISTRUZIONI SU MODALITÀ DI COMPILAZIONE, TESTING ED ESECUZIONE

La compilazione avviene tramite CMake, un software libero e open-source: viene fornito con il progetto un file CMakeLists.txt, utile per questo scopo.
Nel file CMakeLists.txt si stabiliscono i flag per la compilazione, nel nostro caso si tratta di "-Wall -Wextra -fsanitize=address,undefined", si definiscono gli eseguibili che verrano creati (e a partire da quali translation unit) e dove necessario si effettua il linking con la libreria grafica SFML.
Alla fine si otterranno gli eseguibili "sir" e "visual" rispettivamente per prima e seconda parte del progetto e gli eseguibili "sir.t", "evolve.t", "vaccination.t", "infection.t" per i test.
Dapprima è necessario creare un'area di compilazione a partire dalla directory "attuale" (si assume che l'utente si trovi nella stessa directory dei file .cpp e di CMakeLists.txt): questo viene fatto con il comando
cmake -S . -B build -DCMAKE_BUILD_TYPE=Release
Questo comando specifica di utilizzare come "sorgente" la directory attuale ("-S .") e di creare l'area di compilazione in una cartella "build2 generata appositamente ("-B build"). Il flag "-DCMAKE_BUILD_TYPE=Release" serve per specificare che la modalità di compilazione deve essere quella di release (invece che quella di debug).
In seguito si compila effettivamente il programma con il comando
cmake –build build
A questo punto gli eseguibili si troveranno nella cartella "build" e sarà possibile farli partire con comandi come ./build/sir oppure ./build/visual eccetera…
Si segnala che la libreria grafica potrebbe produrre memory leak: l'address sanitizer li metterà in evidenza alla chiusura della finestra di visualizzazione.
Per il testing si è usato doctest (si può trovare il file doctest.h nella cartella dei file da scaricare). Per quanto riguarda la prima parte si è testata la corretta costruzione di oggetti di tipo sir e il corretto funzionamento dei metodi in sir.test.cpp, invece in evolve.test.cpp viene definita una funzione che riprende i tratti fondamentali del main() di first_part.cpp e si controlla il corretto sviluppo del sistema per casi semplici (in modo che gli autori potessero calcolare la stessa cosa in modo indipendente e fare il confronto dei risultati). Per quanto riguarda la seconda parte ci si è concentrati sul corretto funzionamento dei metodi innfection() e vaccination(): data la natura probabilistica dei due metodi abbiamo ripetuto gli stessi semplici passi per 100000 volte e abbiamo appurato che statisticamente il programma si comportava correttamente, ad esempio un singolo infetto, con $\gamma = 0.25$, dopo una chiamata a infection() effettivamente "moriva" in un caso su quattro.

4. CONCLUSIONI E ANALISI DEI RISULTATI

Per quanto riguarda la prima parte del progetto, il programma rispecchia i risultati previsti dalle equazioni differenziali del modello SIR, in particolare, si può notare come con un piccolo γ la diffusione dell'epidemia sia facilitata. Questo perché il tasso di rimozione γ è inversamente proporzionale alla vita media di un individuo infetto, che, rimanendo in vita più a lungo, riesce a contagiare un maggior numero di persone prima di essere rimosso.
Per quanto riguarda invece la seconda parte del progetto, la grandezza della popolazione gioca un ruolo rilevante a causa del comportamento dinamico della simulazione: un minor numero di persone nella griglia implica una minore probabilità che due persone occupino la stessa cella e quindi possano contagiarsi. Con l'implementazione del vaccino la diffusione epidemica è ostacolata da una minore probabilità di infezione, inoltre vediamo che più è tempestivo l'intervento, migliori sono i risultati.

Riportiamo alcuni risultati ottenuti:

- 80 Suscettibili, 10 Infetti, β=0.4, γ=0.1 ($R_0 = 4$), no vaccinazioni
  → S = 80, I = 0, R = 10
- 800 Suscettibili, 100 Infetti, β=0.4, γ=0.1 ($R_0 = 4$), no vaccinazioni
  → S = 238, I = 0, R = 662
- 800 Suscettibili, 100 Infetti, β=0.4, γ=0.1 ($R_0 = 4$), vaccinazioni che iniziano con il 15% di Rimossi
  → S = 255, I = 0, R = 645
- 800 Suscettibili, 100 Infetti, β=0.4, γ=0.1 ($R_0 = 4$), vaccinazioni che iniziano con il 3% di Rimossi
  → S = 271, I = 0, R = 629

Va notato che il modello non è in accordo con le equazioni usate nella prima parte, visto che il modello prevede contagi solo tra persone nella stessa cella, mentre il modello SIR prevede a ogni iterazione "contatti" tra tutte le persone nella popolazione. Infatti si trova che

- 800 Suscettibili, 100 Infetti, β=0.4, γ=0.1 ($R_0 = 4$), no vaccinazioni
  → S = 13, I = 5, R = 882

5. LINK UTILI

- Documentazione SFML: www.sfml-dev.org/tutorials/2.6/
- GitHub del progetto: https://github.com/FraGalasso/SIR-model-23