Implementazione modello SIHD con logiche basate su equazioni differenziali e su un automa cellulare

Autrice progetto:

Lunedei Nicole

IL PROBLEMA PROPOSTO

Questo programma propone due modelli per studiare e rappresentare l'evoluzione di una pandemia a cui è soggetta una popolazione.

La popolazione viene suddivisa in 4 categorie: i Suscettibili, ovvero gli individui sani che sono sensibili al virus e quindi possono infettarsi; gli Infettati, ovvero gli individui che hanno contratto il virus e che possono infettare; i Guariti, sono coloro che sono guariti dal virus, e non possono né ammalarsi di nuovo né infettare; infine, ci sono i Morti, coloro che non sono sopravvissuti a causa del virus.

L'evoluzione della pandemia è caratterizzata da parametri probabilistici, rispetto a quello che può succedere in una giornata:

- probabilità di infezione, ovvero quanto è probabile che un suscettibile venga infettato
- probabilità di guarigione
- probabilità di morte

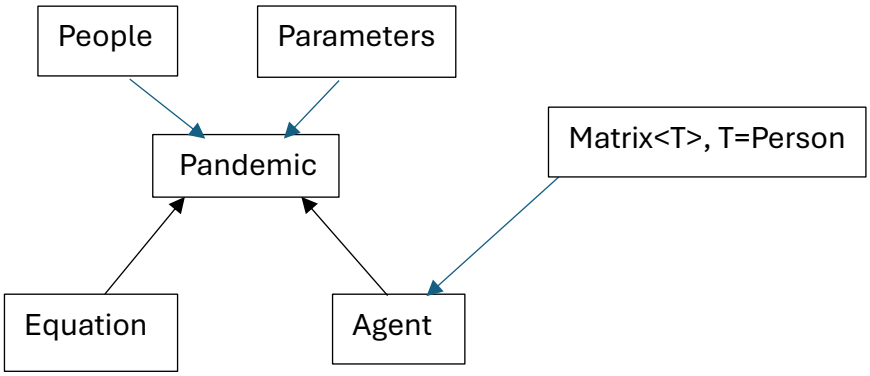
C'è un ultimo parametro che rappresenta la tendenza della popolazione a vaccinarsi. Introdurre l'opzione del vaccino determinerà uno "splitting" tra le persone suscettibili vaccinate e non vaccinate che seguiranno evoluzioni separate con la stessa logica, ma con parametri diversi.

CONCETTI DI C++

I concetti utilizzati all'interno del codice sono:

- User-defined type: class e struct.
    - Overloading di operatori (<<,>>,=,==,!=..)
    - Ereditarietà tra classi
- Funzioni esterne
- Funzioni lambda
- Template: sia in ambito di funzioni che di user-defined type
- Smart-pointer: in particolare, gli std::unique_ptr<T> che hanno permesso l'utilizzo dell'allocazione dinamica, azzerando il rischio di memory leaks durante l'esecuzione

INTRODUZIONE DEI TIPI DEFINITI



La struttura ereditaria di Pandemic (Base) con Equation e Agent (Derivate) è rappresentata dalle frecce nere; il riutilizzo di tipi in Pandemic, di Parameters e di People, e in Agent di Matrix<Person> è, invece, rappresentato dalle frecce azzurre.

Analisi delle classi

Pandemic è il tipo di un oggetto in grado di descrivere e registrare l'evoluzione della pandemia. Gli attributi privati sono i seguenti:

- Un vettore di oggetti di tipo People (**std::vector<People>**): è un vettore di tracciamento, per registrare il decorso della diffusione, e la sua dimensione rappresenta il numero dei giorni.
  People rappresenta la popolazione suddivisa nelle quattro categorie: Susceptible, Infected, Healed e Dead.

Oggetto People

S_ [0](int)	S_ [1] (int)
I_ [0] (int)	I_ [1] (int)
H_(int)	
D_(int)	

Lo struct People è costituito da:
-3 costruttori: Default, Parametrico e di Copia;
- 4 overloading di operatori: operator=, operator==, operator>> e operator<<;
- il distruttore.

Gli array bidimensionali dei primi due elementi concretizzano lo "splitting" dei suscettibili e degli infettati: i non vaccinati e i vaccinati sono indicati rispettivamente da [0] e [1].

- Un oggetto di tipo Parameters(**par**_): rappresenta l'insieme dei parametri probabilistici che danno forma alla dinamica dell'evoluzione, citati in precedenza.

Oggetto Parameters

β [0](double)	β [1](double)
γ [0](double)	γ [1](double)
Ω[0](double)	Ω[1](double)
vax(double)	

Lo struct Parameters è costituito da:
-3 costruttori: Default, Parametrico e di Copia;
- 2 operatori overloaded: operator=, operator==;
- il distruttore.
Come nello struct, People gli array bidimensionali servono a distinguere i vaccinati dai non vaccinati.

- Un oggetto integer(**N**_): rappresenta il numero totale della popolazione
- Generazione casuale(static):
    - std::mt19937(**gen**): un motore di generazione di numeri pseudo-casuali (PRNG, Pseudo Random Number Generator) basato sull'algoritmo Mersenne Twister.
    - std::uniform_real_distribution<> (**dis**): una distribuzione probabilistica inclusa nella libreria standard di C++ (<random>)
- Un array static bidimensionale di oggetti double (**std::array<double,2> efficacy**): rappresenta l'efficacia del vaccino sulla diffusione del virus e sulla morte.
  Per la spiegazione dei metodi vedere l'APPENDICE 1

  Perchè set_parameters e introduce_vacc?

  L'obbiettivo è di riprodurre una situazione reale, ma che permetta una manipolazione chiara dell'oggetto: nella realtà un vaccino non è pronto nello stesso istante della comparsa del virus, perciò l'oggetto viene creato privo di vaccino e ci sono i controlli per prevenirlo sia nel costruttore parametrico che in set_Parameters. Il metodo introduce_vacc rappresenta un contributo che l'uomo può dare alla situazione per cercare di migliorarla e lo decide lui se e quando introdurlo, come può fare l'utente durante la simulazione, ma può farlo solo una volta.

La dichiarazione e la definizione della classe sono presenti nei file pandemic.hpp e pandemic.cpp.

Tutto ciò viene ereditato dalle classi derivate Equation e Agent. Infatti, queste classi si specializzano in termini di logica che governa l'evoluzione e, di conseguenza, anche di modalità di visualizzazione dell'evoluzione.

EQUATION CLASS

La logica si basa su un sistema di equazioni differenziali che descrivono il modello della diffusione di un virus:

- $\frac{dS}{dt} = -\beta \frac{S}{N} I$
- $\frac{dI}{dt} = +\beta \frac{S}{N} I - \gamma I - \omega I$
- $\frac{dH}{dt} = +\gamma I$
- $\frac{dD}{dt} = +\omega I$

La modalità di visualizzazione è la stampa dei valori aggiornati giornalmente, sia facendo un conteggio totale sia separando i vaccinati e i non vaccinati.
Per la spiegazione dei metodi vedere l'APPENDICE 2.
La dichiarazione e la definizione della classe sono presenti nei file equation.hpp e equation.cpp

AGENT CLASS

La logica di questa classe si basa puramente sulla probabilità. Si può immaginare proprio come il "lancio di un dado" ogni volta che avviene un incontro tra un suscettibile e un infetto oppure un infetto che può morire o guarire.

La visualizzazione dell'evoluzione avviene tramite la realizzazione di un automa cellulare rappresentato da una matrice (Matrix<T>, APPENDICE 4) avente un numero di elementi pari al numero della popolazione; ogni elemento rappresenta una persona a cui è associato uno stato, che evolve
durante la simulazione, questo tipo di elemento è un'enumerazione che può assumere i seguenti stati: Suscettibile, Suscettibile_v, Infetto, Infetto_v, Guarito e Morto.
Per la spiegazione dei metodi vedere l'APPENDICE 3.

La dichiarazione e la definizione della classe sono presenti nei file agent.hpp e agent.cpp.

FUNZIONI TEMPLATE

Sono state utilizzate per definire in maniera più efficiente i metodi e sono definite nel file pandemic.hpp.

Un esempio è in Equation class, dove queste sono state molto utili per collegare e manipolare gli oggetti all'interno dei metodi evolve () e evolve_vaccine (). Infatti, la combinazione della logica e della rappresentazione dei dati ha generato un problema, cioè come adattare oggetti double (equazioni differenziali) con oggetti integer (oggetto People).

L'idea è stata di considerare la parte decimale dei dati risultanti e valutare quale di questi avesse la parte decimale più grande, in seguito aggiungergli la somma delle parti decimali. Per realizzare ciò è stata adottata la seguente logica:

- Un nuovo oggetto People, viene dato come argomento a update_situation (), che svolge due azioni: la prima è codificare il tipo People in std::array<T,N> (= transform_Array<T,N>, T = double e N= 6) e la seconda è l'aggiornamento;
- il risultato di update_situation () è un contenitore di sei oggetti double, il passo successivo è individuare l'elemento con la parte decimale più grande (=maximum_dec()); si procede sommando le parti decimali e aggiungerle all'oggetto restituito da maximum_dec; l'array di oggetti double modificato viene trasformato in un array di oggetti integer (= integer_part()) che viene trasformato in un oggetto People. Questo è ciò che succede nel metodo fix.
- Infine, fix restituisce un oggetto People pronto per essere aggiunto al vettore di tracciamento population_

Strategie di testing

Le classi e lo struct sono stati testati con l'utilizzo della libreria "doctest.h".
Per verificare il corretto funzionamento del codice, è stato seguito il seguente ordine di verifiche:

- Costruzione degli oggetti con i costruttori di default, parametrico e di copia
- Modifiche di dati con setters e funzionalità generali

- Test sui metodi che gestiscono l'evoluzione:
    - Per Equation: sono stati testati i metodi update_situation e fix individualmente, e poi in evolve e evolve vaccine.
    - Per Agent: grazie all'utilizzo di inside_matrix, è stata testata ogni cella della matrice verificando che il funzionamento (change_state, data_colection e evolve) non andasse in contrasto con le regole stabilite. Inoltre, è stato necessario effettuare la verifica di alcuni metodi stampando su terminale, ad esempio per infected_neighbours.

Sono stati utilizzati sia valori che rispettano i requisiti delle classi, sia valori che non li rispettano provocando il lancio di eccezioni previste.

SIMULAZIONI

In entrambi i programmi viene chiesto all'utente se vuole avviare una simulazione con dati di default oppure con dati personalizzati.

Gli oggetti vengono allocati dinamicamente con gli smart pointers (std::unique_ptr<Equation> e std:: unique_ptr <Agent>). Questa è una scelta vantaggiosa, non solo per il risparmio di tempo e spazio di memoria nello stack al compile time, ma anche per rendere visibile l'oggetto nell'intera scope e inizializzarlo all'interno del blocco try/catch, localizzato all'interno di un ciclo while (righe 274-303 , di main_ag_simulation.cpp).Questi cicli permettono di richiedere i dati finché questi non rispettano i requisiti delle classi fino a un massimo di volte, stabilito dalla variabile MAX_IT.

I dati vengono chiesti in input da terminale, in particolare, il numero della popolazione, le probabilità e la situazione iniziale.

Per l'avvio della simulazione è necessario fare attenzione alla combinazione delle probabilità per rispettare la condizione $R_0 > 1$(con $R_0 = \frac{\beta}{\gamma+\omega}$) e nell'impostazione della situazione iniziale, gli infettati non devono essere nulli, mentre i guariti e i morti sì. Nel caso in cui la somma degli infettati e dei suscettibili sia minore del numero della popolazione non è un problema perché viene risolto dal metodo set_initial_condition, ma non può essere superiore o verrà lanciato un errore.

Importante notare l'utilizzo di due metodi dell'oggetto std::cin che garantisco la pulizia del flusso di dati d'entrata ogni volta che viene immesso un dato:

- std::cin.clear: rimuove lo stato d'errore del flusso
- std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n'): ha la funzione di svuotare il buffer associato al flusso

Ci sono due modalità per far terminare la simulazione: automatica, cioè finché ci sono ancora persone infette la simulazione continua (metodo **terminate** e ciclo while), oppure l'utente può scegliere il numero di giorni della durata della pandemia (variabile T e ciclo for).

Equation

Nella parte di default viene avviata una simulazione automatica; giunta al termine viene chiesto se si vuole confrontare la simulazione precedente con una avente gli stessi dati, ma con l'aggiunta l'opzione di vaccino.

Nella parte personalizzata, dopo l'inserimento dei dati input, si chiede la modalità della durata se automatica o con la scelta dei giorni.

- Con la scelta automatica, come nella parte di default, alla fine chiede se si vuole confrontare la simulazione appena eseguita con una avente l'opzione di vaccino.
- Con la scelta dei giorni, al termine del ciclo, c'è un controllo che verifica se siano rimaste persone infette, in caso affermativo viene chiesto all'utente se desidera terminare la simulazione finché le persone infette non si annullano. Contemporaneamente, è possibile introdurre il vaccino.

  Alla fine, vengono stampati i dati importanti della pandemia:

- Numero dei giorni
- Valore di soglia critica
- (in caso di confronto) i dati degli ultimi giorni di entrambe le simulazioni

Per un esempio vedere **APPENDICE 5**.

Agent

La novità in questa simulazione è che vengono utilizzati anche gli oggetti di sfml, per la visualizzazione grafica della matrice dell'oggetto Agent. Per questo motivo, prima dell'inizio della simulazione viene definita una funzione paint, che ha come unico scopo quello si tradurre un elemento Person in una figura circolare di un certo colore (Suscettibili = verde, Suscettibili_v = acquamarina, Infettati = rossi, Infettati_v = arancioni, Guariti = blu e Morti = Magenta).

Nella parte di default viene avviata una simulazione con la scelta dei giorni; al termine, del ciclo viene chiesto se si vuole terminare la simulazione in maniera automatica.

Nella parte personalizzata, dopo l'inserimento dei dati input, si chiede la modalità della durata automatica o con la scelta dei giorni.

- Con la scelta automatica, la simulazione viene gestita dal ciclo while d'apertura dell'oggetto sf::RenderWindow.
  Durante la visualizzazione della finestra è possibile interagire con essa con i seguenti tasti:

- ⇨ “Space Bar”: avvio della simulazione;
- ⇨ “P”: mettere in pausa la simulazione;
- ⇨ “V”: mettere in pausa la simulazione e introdurre il vaccino;

- Con la scelta dei giorni: al termine del ciclo, ci sono due controlli successivi, sull’ultimo elemento di population_:
  - ⇨ un controllo che verifica se siano rimaste persone infette, in caso affermativo viene chiesto all’utente se desidera continuare in maniera automatica
  - ⇨ un controllo che verifichi se ci siano ancora persone suscettibili al virus, in caso affermativo viene chiesto di introdurre l’opzione del vaccino per terminare l’evoluzione

Alla fine, vengono stampati i dati importanti della pandemia:

- Numero dei giorni
- Valore di soglia critica
- I dati dell’ultimo giorno

Alla fine compaiono problemi di memory leak dovuti,però, all’utilizzo di oggetti sfml:

```
==21159==ERROR: LeakSanitizer: detected memory leaks

Direct leak of 128 byte(s) in 1 object(s) allocated from:
    #0 0x7f6daaf8cc3e in __interceptor_realloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:163
    #1 0x7f6da5126efd  (<unknown module>)

Direct leak of 64 byte(s) in 1 object(s) allocated from:
    #0 0x7f6daaf8e587 in operator new(unsigned long) ../../../../src/libsanitizer/asan/asan_new_delete.cc:104
    #1 0x7f6da59840cd  (/usr/lib/x86_64-linux-gnu/libasan.so.5.0.0+0x4d00cd)

Indirect leak of 56 byte(s) in 1 object(s) allocated from:
    #0 0x7f6daaf8ca06 in __interceptor_calloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:153
    #1 0x7f6da5ceb905  (/usr/lib/x86_64-linux-gnu/libasan.so.5.0.0+0x837905)

Indirect leak of 56 byte(s) in 1 object(s) allocated from:
    #0 0x7f6daaf8ca06 in __interceptor_calloc ../../../../src/libsanitizer/asan/asan_malloc_linux.cc:153
    #1 0x7f6da5ceb8ee  (/usr/lib/x86_64-linux-gnu/libasan.so.5.0.0+0x8378ee)

SUMMARY: AddressSanitizer: 304 byte(s) leaked in 4 allocation(s).
```

RIFERIMENTI

Il programma e le istruzioni CMake, per eseguirlo (file README), sono presenti al link https://github.com/nicoleLunedei/repo_progetto.

==========

APPENDICE 1

Metodi della classe Pandemic:

- 3 Costruttori: Default,Parametrico e Copia
- Distruttore
- Setters: **set_Parameters, introduce_vacc e set_initial_condition**
- Getters: **get_evolution** (che restituisce l’oggetto population_), **get_Parameters**, **get_number_population, get_days** (numero dei giorni), **get_situation_day**(situazione del giorno indicato);

- Checking methods: **check_R0**(viene controllato che il valore di soglia critica sia maggiore di 1), **check_normalization** (controllache la proprietà di normalizzazione venga rispettata, in caso contrario viene risolto il problema)
- Funzionalità generali: **change_after_vacc**(aggiorna le probabilità per le persone vaccinate),**generate**(genera un numero casuale: gen funge da seme per la distribuzione di probabilità),**add_data** (aggiunge un elemento People al vettore population_), **is_vaccinated**(esegue una sorta di raccolta dati tra le persone suscettibili in riferimento a chi decide di vaccinarsi e chi no), **calculate_R0**(calcola il valore di soglia critica associato alla diffusione del virus), **terminate** (informa quando non ci sono più persone infette).

APPENDICE 2

Metodi della classe Equation:

- 3 Costruttori: Default, Parametrico e Copia
- Distruttore
- Raccolta dati riguardanti il vaccino, cioè viene chiesto ad ogni suscettibile se vuole vaccinarsi o no, e vengono separati in suscettibili vaccinati e non vaccinati: **sorting**;
- Funzionalità per l'evoluzione: **update_situation**(si limita ad aggiornare la situazione in base alla impostazioni date), **fix**(permette di convertire i valori double in valori interi rispettando i tipi di People e il numero della popolazione totale), **evolve** e **evolve_vaccine** sono i metodi che fanno avvenire l'evoluzione vera e propria, rispettivamente senza e con il vaccino;
- Funzionalità di visualizzazione: **calculate,** prende i dati di una giornata e li restituisce sostto forma di array a 4 dimensioni, somma i suscettibili e gli infatti; Print specifica automatizza la stampa del risultato di calculate.

APPENDICE 3

Metodi della classe Agent:

- 3 Costruttori: Default, Parametrico e Copia
- Distruttore
- Getters: **get_matrix**; **show_cell,** restituisce il valore di uno specifico elemento della matrice garantendo una struttura totoidale; **get_side,** restituisce il valore del lato della matrice
- Setters: **draw_matrix**, imposta la situazione iniziale della simulazione e la "disegna" sulla matrice in maniera casuale;
- Funzionalità d'evoluzione: **change_state**, riconosce lo stato di una particella e lo cambia in base alle regole date; **data_collection,** traduce la situazione della matrice aggiornata in dati in "formato " People; **evolve,** fa avvenire l'evoluzione vera e propria aggiornando la matrice e il vettore population_.
- Funzionalità generali: **throwing_dices,** lmatrice;nte "tirare i dadi", viene generato un numero random e confrontato una valore double che rappresenta la probabilità data in argomento; **infected_neighbours,** effettua un conteggio di quante persone infettate ci sono attorno ad una suscettibile; **sorting,** raccoglie dati riguardanti il vaccino, cioè viene chiesto ad ogni suscettibile

se vuole vaccinarsi o no, e vengono separati in suscettibili vaccinati e non vaccinati, sia in population_ che nella matrice.

APPENDICE 4

Matrix<T> è costituito da:

- 3 Costruttori: Default, Parametrico e di Copia
- Overloading di 5 operatori: operator=, operator==, operator!=, operator<< e operator[];
- Metodi per interagire singolarmente con tutti gli elementi della matrice per applicare una determinata un'operazione: **inside_matrix** e **each_cell**.
  Questi due metodi sono stati molto utili per evitare la ridondanza di doppi cicli for e ciò ha permesso, al momento di specificare l'operazione, una completa concentrazione su di essa (definita tramite una funzione lambda) e, quindi, una lettura più chiara e semplice del codice.
- Funzionalità generali: **add**(aggiunge un elemento alla fine dell'ultima riga), **sum**(somma di tutti gli elementi), **modify**(modifica un elemento specifico) e **read**(restituisce un elemento specifico)

File matrix.hpp

APPENDICE 5

Esempio di Equation con simulazione personalizzata e modalità automatica.
N = 4000
prob = {{0.75, 0.}, {0.23,0.}, {0.36,0.},0.}
people: Suscettibili = 3000, Infettati = 1000, Guariti = 0, Morti = 0
prob_v = 0.68
Modifica di prob: = {{0.75, 0.2175}, {0.23, 0.464}, {0.36, 0.126},0.68}

Secondo Giorno della simulazione con il vaccino
Day N° :2
Suscepttible not vaccinated: 791|| Suscepttible vaccinated: 1917
Infected not vaccinated: 592|| Infected not vaccinated: 110
Healed: 230
Dead: 360

Output ottenuti:
| Simulation with vaccine |
Numbers of days :18
S = 2257 || I = 0 || H = 801 || D = 941

| Simulation without vaccine |
Numbers of days :19
S = 1119 || I = 0 || H = 1122 || D = 1757

R_0 =1.27119