



UNIVERSITÀ DEGLI STUDI DI MILANO - BICOCCA

Scuola di Scienze

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

AGENT-BASED MODELING AND LEARNING FOR EPIDEMIOLOGICAL STUDIES

Relatore: Prof. Antoniotti Marco

Correlatore: Prof.

Relazione della prova finale di:

Matteo Stievano

Matricola 829535

September 3, 2023

Anno Accademico 2022-2023

Abstract

Lo scopo del presente lavoro è l'analisi e lo studio del comportamento di modelli di simulazione e di intervento nell'ambito della gestione di eventi pandemici, utilizzando il linguaggio di programmazione Julia e alcuni dei suoi framework, tra cui Agents.jl, SciML.ai e Lux.jl.

Nello specifico, si propone l'adozione del framework Agents.jl come strumento principale per la simulazione di sistemi complessi e l'integrazione di SciML.ai e Lux.jl per lo sviluppo di un controllore in grado di analizzare e interpretare i dati in ingresso. Questo controllore è finalizzato all'apprendimento di strategie di intervento e alla loro applicazione mediante tecniche di machine learning ibrido, come l'utilizzo di Neural ODE.

L'obiettivo è sviluppare un approccio dinamico e ibrido che combini le vantaggiose caratteristiche dei modelli di simulazione e dei modelli matematici. Poiché il problema affrontato è intrinsecamente complesso, è stato modellato come un grafo sociale, cercando di emulare il comportamento di una rete sociale in cui i nodi del grafo rappresentano gli agenti del modello.

Per migliorare le prestazioni e la stabilità del modello, è stato introdotto un sistema di equazioni differenziali ordinarie (ODE) che gestisce e simula l'andamento epidemico di ciascun nodo (agente) del grafo. Il controllore si basa sull'idea di una Neural ODE, che regola in modo autonomo e automatico il livello di contromisure applicate. Tali contromisure sono identificate come una riduzione dell'indice di infettività R_0 . Per evitare l'applicazione di contromisure praticamente insostenibili, è stato introdotto un parametro di "felicità", che agisce come una funzione di costo aggiuntiva per il controllore.

Contents

1	Introduzione	1
2	Panoramica dello Stato dell'Arte	3
2.1	Epidemiologia	3
2.2	Modelli Compartmentali	6
2.2.1	Il modello Kermack-McKendrick	6
2.3	Modelli basati su Agenti	10
2.3.1	Comportamento Emergente	11
2.3.2	Discretizzazione	11
2.4	Julia	13
2.4.1	Agents.jl	13
2.4.2	Benchmarking e confronto con altri linguaggi	14
2.5	Equazioni Neurali Differenziali	17
2.5.1	Inserire un'ODE all'interno di una NN	21
2.5.2	Backpropagation tramite ODE solver	22
2.5.3	Equazioni Differenziali Universali	23
3	Metodi e Modelli	26
3.1	Approccio con Rete Sociale	26
3.2	Agente	27
3.3	Spazio e Modello	29
3.4	Funzione di Avanzamento Agente	29
3.5	Funzione di Avanzamento Modello	31
3.5.1	Funzione per la Generazione delle Varianti del Virus (VOC)	32
3.5.2	Funzione per la Simulazione della Campagna di Vaccinazione	32
3.6	Monitoraggio e Intervento	35
4	Analisi di Sensibilità	43
4.1	Analisi del Comportamento in Base al Numero Iniziale di Nodi Infetti	45
4.2	Analisi del Comportamento in Base al Tasso di Migrazione	46
4.3	Analisi del Comportamento in Base al Numero di Nodi della Rete	47
4.4	Analisi del Comportamento in Base alla Copertura della Rete	48
5	Risultati Ottenuti	49
5.1	Nessun Intervento	50
5.2	Intervento Non Farmaceutico	51
5.3	Intervento Farmaceutico	52
5.4	Intervento Farmaceutico e Non Farmaceutico	53
6	Conclusione	54
7	Sviluppi Futuri	55
7.1	Perfezionamento del Controllore	55
7.2	Miglioramento della Funzione di Happiness	55
7.3	Perfezionamento Generale del Modello	55

7.4 Ottimizzazione Generale	55
A Altri Approcci	62
A.1 Modello ad Agente su Spazio Continuo	62
A.2 Modello ad Agente con Spazio a Grafo e Modellazione Singolo Agente	64
A.3 Controllore Ipopt	66

List of Figures

1	Esempio di correlazione spuria	3
2	Un altro esempio di correlazione spuria	4
3	Esempio della struttura del modello SIR	7
4	Esempio di modello SEIRS [8]	8
5	Esempio di modello SEIR stocastico	9
6	Agent-Based Epidemiological Modeling of COVID-19 in Localized Environment [17]	10
7	Esempio di comportamento emergente nella simulazione degli stormi di uccelli	11
8	Esempio di differenti tipologie di discretizzazione, siano esse nel tempo o nei valori	12
9	Esempio di metriche per il calcolo della distanza su uno spazio a griglia [21]	14
10	Tabella comparativa [1]	15
11	Differenti approcci di apprendimento di machine learning	17
12	Esempio di funzionamento di una rete neurale	18
13	Esempio grafico ODE rappresentante la famosa equazione di Lotka-Volterra	19
14	Esempio definizione ODE in Julia	20
15	Esempio implementazione di una rete neurale in Julia	21
16	Esempio implementazione NeuralODE tramite DiffEqFlux.jl	22
17	Tabella comparativa tra le varie implementazioni dei vari risolutori [62]	23
18	Esempio comparativo tra funzionamento Machine Learning e Deep Learning	24
19	Comportamento UDE nell'approssimazione di fenomeni non lineari [66]	25
21	Moreno's Sociogram of a 3rd grade class	27
22	Codice Agente	28
23	Codice Modello	29
25	Funzione atta a calcolare lo spostamento di agenti da un nodo all'altro del grafo	30
26	Funzione che crea la matrice di migrazione data la topologia di un grafo	31
27	Funzione atta a calcolare la felicità degli agenti	31
28	Funzione che si occupa di generare la VOC	32
29	Funzione che si occupa di simulare la ricerca di un vaccino e la sua successiva applicazione	33
31	Definizione del controllore all'interno del modello ad agente	36
32	Funzione per il calcolo del limite superiore per le contromisure non farmaceutiche	37
33	Definizione del controllore mediante Neural ODE	38
34	Definizione delle funzioni di supporto del controllore	39
35	Definizione delle funzioni di addestramento del controllore	40
36	Esempio di utilizzo congiunto di due ottimizzatori differenti in uno stesso ciclo di addestramento	41
37	Funzione di attivazione Swish	42
38	Grafico rappresentante l'analisi di sensitività del modello	43
39	Parametri usati per l'analisi di sensitività del modello ad agente	44
44	Grafico cumulativo del modello senza alcun tipo di intervento	50
45	Grafico cumulativo del modello con intervento non farmaceutico	51
46	Grafico cumulativo del modello con intervento farmaceutico	52
47	Grafico cumulativo del modello con intervento farmaceutico e non farmaceutico combinato	53
48	Esempio del modello modellato su spazio continuo	62

49	Esempio del comportamento delle curve nel modello continuo	63
50	Comportamento modello ABM su spazio a grafo al variare del parametro R_0	64
51	Comportamento modello SEIR al variare del parametro R_0	65
52	Formula che si occupa di descrivere il rapporto tra il comportamento del modello scartato e del modello SEIR. In particolare questa formula descrive il rapporto tra gli R_0	65
53	Comparison of Ipopt performance over various linear solvers using the two-dimensional partial differential equation test problem set. [76]	66
54	Definizione del controllore tramite Ipopt	67
55	Definizione regole del modello del controller	68
56	Definizione regole del modello del controller per le contromisure	68

1 Introduzione

L'impiego di metodologie e tecniche sempre più avanzate è stato oggetto di discussione e interesse nella comunità scientifica, in particolare tra epidemiologi e medici. Negli ultimi anni, il mondo è diventato notevolmente interconnesso, aumentando significativamente la probabilità di diffusione globale di virus e di conseguenti catastrofi senza precedenti.

La storia umana è segnata da epidemie, ma solo alcune di esse hanno lasciato un'impronta duratura nella memoria collettiva a causa delle loro conseguenze catastrofiche. Tra queste, alcune delle più note includono la peste nera che nel XIV secolo mieté venti milioni di vite in Europa in soli sei anni, l'epidemia di tifo durante le crociate e la Seconda Guerra Mondiale, l'influenza spagnola che causò 50 milioni di morti tra il 1918 e il 1920, e l'epidemia di AIDS, che ha colpito oltre 75 milioni di persone e causato 35 milioni di decessi dal 1981.

Oggi, l'influenza stagionale non suscita più lo stesso timore nei cittadini dei paesi sviluppati, ma la pandemia di COVID-19, iniziata alla fine del 2019, ha condizionato l'umanità per tre anni e continua a farlo, causando finora quasi 7 milioni di vittime accertate. Questa pandemia rimarrà impressa nella memoria umana poiché ha messo in crisi l'intero sistema governativo globale, causando allarmi, panico e, talvolta, isteria, come pochi altri eventi sono stati in grado di fare.

Esaminando le statistiche di questa epidemia, i numeri relativi ai decessi e agli infetti (quasi 7 milioni di morti e oltre 700 milioni di infetti) da soli sono sufficienti a preoccupare qualsiasi lettore. Tuttavia, ci sono altri dati meno evidenti che forniscono informazioni altrettanto inquietanti, come l'impatto economico globale e il concetto di "poverty trap", un circolo vizioso in cui la povertà e le malattie perpetuano un ciclo di bassa salute e crescente povertà.

Questi sono solo alcuni dei problemi che possono emergere durante una pandemia globale come quella del COVID-19, e quindi la comunità scientifica, in particolare gli epidemiologi, cerca costantemente soluzioni più efficaci ed accurate per prevenire, contenere e gestire eventi di questo genere.

L'epidemiologia è una disciplina relativamente giovane che si è evoluta per affrontare emergenze sanitarie. La sua definizione originale risale al 1978, ma nel corso del tempo è cresciuta e si è adattata alle esigenze della società. Attualmente, l'epidemiologia è definita come lo studio della distribuzione e dei determinanti delle condizioni o eventi legati alla salute in una specifica popolazione, con l'applicazione di questo studio al controllo dei problemi di salute.

Uno strumento ampiamente utilizzato in epidemiologia è la simulazione tramite software, che si basa su modelli matematici per trarre conclusioni sui sistemi analizzati. Questi sistemi, spesso definiti complessi, coinvolgono molteplici componenti che interagiscono tra loro e possono essere descritti mediante modelli matematici.

I primi modelli epidemiologici erano compartmentali, basati su gruppi di individui separati che interagivano tra loro e potevano essere descritti mediante equazioni differenziali ordinarie (ODE). Uno dei modelli più noti è il modello Suscettibile-Infetto-Ricoverato (SIR) sviluppato da Kermack e McKendrick nel 1927. Successivamente, sono emersi i modelli ad agenti, che sono autonomi e consentono la simulazione di sistemi complessi tramite l'interazione di entità autonome.

La simulazione ha fatto notevoli progressi dagli anni '90 grazie all'espansione delle risorse computazionali. Tuttavia, i modelli di simulazione richiedono compromessi e semplificazioni, ad esempio la discretizzazione degli ambienti di simulazione. Inoltre, è necessario considerare il comportamento

umano in situazioni di pericolo, il che può essere complesso da modellare.

L'identificazione delle cause di un fenomeno e la comprensione di come un intervento influenzi tale fenomeno rappresentano una delle sfide più complesse dell'epidemiologia. La causalità, ossia la relazione di causa-effetto tra eventi o variabili, non è sempre evidente e richiede un'analisi rigorosa. La correlazione tra eventi non implica necessariamente causalità.

In questa introduzione, abbiamo fornito una panoramica sull'epidemiologia, la simulazione di sistemi complessi e la sfida della causalità. Le sezioni successive analizzeranno lo stato dell'arte dell'epidemiologia e della simulazione, con un focus sulla pandemia da COVID-19 e sui metodi di monitoraggio e intervento nelle simulazioni epidemiologiche.

2 Panoramica dello Stato dell'Arte

2.1 Epidemiologia

L'epidemiologia rappresenta una disciplina biomedica di fondamentale importanza, focalizzata sull'analisi della distribuzione e dell'incidenza delle malattie e degli eventi sanitari rilevanti all'interno di una popolazione. Questo campo di studio si dedica all'approfondimento delle cause, dei pattern temporali e delle conseguenze delle patologie [30] [60].

L'epidemiologia si caratterizza per la sua natura essenzialmente pratica, poiché il suo obiettivo principale consiste nel determinare le cause sottese a un determinato effetto sanitario. Ciò nonostante, questa disciplina si trova ad affrontare una serie di sfide complesse che definiscono il suo percorso. Tra queste sfide, la più rilevante è l'identificazione precisa delle relazioni di causalità tra eventi.

Tali interrogativi possono sembrare elementari, dato che, come specie, abbiamo sviluppato un istinto innato nell'individuare correlazioni causali tra eventi, anche quando queste non esistono effettivamente. Ad esempio, se ci trovassimo in un bosco buio, soli e circondati solo dal sussurro di una leggera brezza estiva e dovessemmo udire un rumore provenire dai cespugli, è probabile che lo associeremmo a un pericolo imminente, come la presenza di un predatore, sebbene il rumore sia causato dalla brezza stessa.

Questo adattamento evolutivo ci ha permesso di sopravvivere in situazioni di pericolo, ma purtroppo, quando si tratta di scienza e dati, l'istinto non è sempre una guida affidabile. I dati, per loro natura, sono neutri e non trasmettono automaticamente informazioni significative; spetta a noi, in quanto individui dotati di intelligenza, tecniche e metodi, estrarre significato da questi dati in modo accurato ed inequivocabile.

Un'osservazione chiave è che ciò che sembra ovvio può essere fuorviante. Per esempio, il grafico seguente sembra dimostrare in modo "inequivocabile" una relazione diretta tra la spesa degli Stati Uniti per la ricerca aerospaziale e il numero di suicidi per strangolamento:

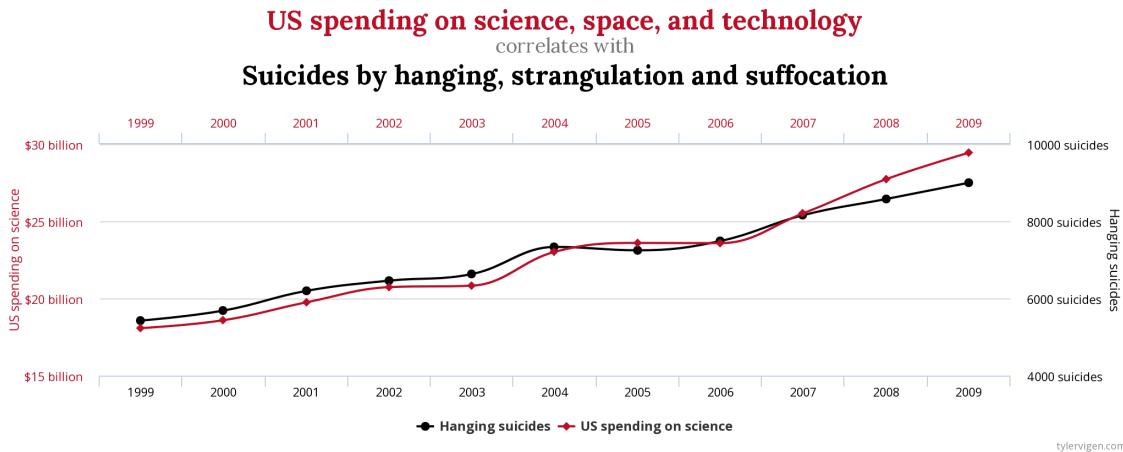


Figure 1: Esempio di correlazione spuria
<https://www.tylervigen.com/spurious-correlations>

Sulla base di questo grafico e dei dati presentati, si potrebbe erroneamente concludere che le due categorie sono in qualche modo correlate e che il governo degli Stati Uniti debba essere accusato di incitare al suicidio. Tuttavia, questa è un esempio di una "relazione spuria", in cui due o più variabili sono associate ma non sono causalmente collegate.

È importante sottolineare che l'associazione e la causalità non sono la stessa cosa, e quando si studia una, è fondamentale non confonderla con l'altra. In statistica, una correlazione tra dati rappresenta qualsiasi tipo di relazione tra due o più variabili, indipendentemente dalla sua natura causale o non causale. Nel caso precedente, la correlazione tra le due variabili potrebbe essere semplicemente dovuta al passare del tempo: nel corso degli anni, la spesa media per la ricerca aerospaziale è continuata a crescere a causa di un interesse e di investimenti sempre maggiori in quel settore, mentre nel tempo si è verificato un costante aumento del numero di suicidi.

Il nostro pregiudizio verso la ricerca di collegamenti tra eventi, in modo che sembrino sempre collegati in modo tangibile e che si possa tracciare una chiara e distinta linea causale dall'inizio alla fine, può ingannarci quando tali collegamenti sembrano evidenti ma non lo sono. Spesso, la spiegazione più semplice è anche la meno interessante, sebbene sia corretta: due eventi completamente svincolati tra loro possono avere andamenti simili, e differenti fattori possono portare allo stesso risultato.

Un altro esempio illustrativo di correlazione spuria è il seguente:

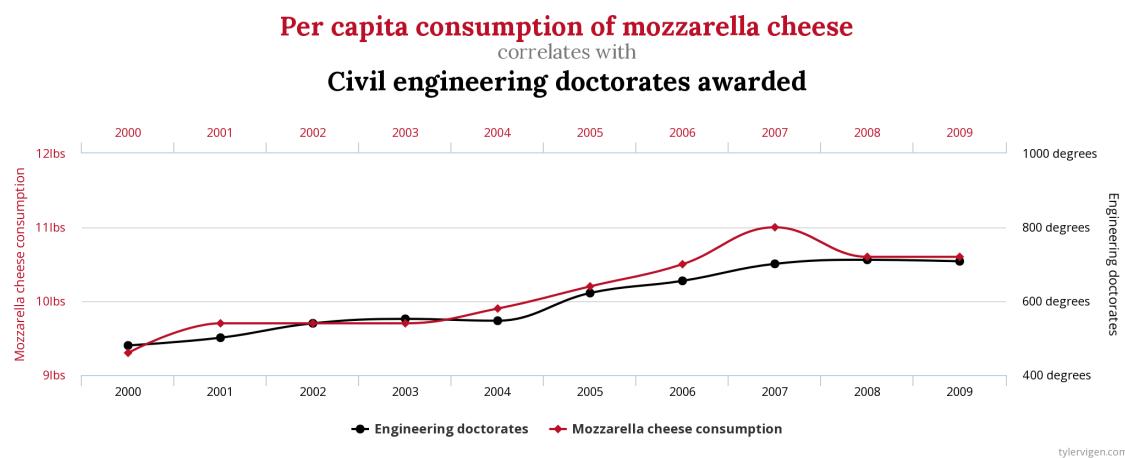


Figure 2: Un altro esempio di correlazione spuria
<https://www.tylervigen.com/spurious-correlations>

Il problema della causalità è di estrema importanza e rappresenta una delle sfide principali quando si cercano di sviluppare e applicare interventi all'interno di una popolazione per mitigare, ad esempio, la diffusione di un agente patogeno [60].

Come già accennato, i dati per se stessi sono privi di significato; è il nostro compito imparare a interpretare il loro significato. Un esempio eloquente di come, nonostante la consapevolezza del problema delle correlazioni spurie, i dati possano comunque trarre in inganno è il seguente:

Supponiamo di essere medici e di dover decidere se prescrivere un certo farmaco a un paziente. Per

prendere questa decisione, abbiamo a disposizione la storia clinica del paziente e i risultati di uno studio su un nuovo farmaco che sembra promettente nel trattamento della sua malattia. Questo farmaco è stato testato su un gruppo di 700 persone, suddivise in due sottogruppi di 350 pazienti ciascuno. In uno di questi sottogruppi, i pazienti hanno scelto autonomamente se assumere o meno il farmaco, mentre nell'altro gruppo, i pazienti hanno fatto l'opposto. Ecco i risultati:

Paradosso di Simpson						
Categoria	Pazienti	Guariti	% Guariti	Pazienti	Guariti	% Guariti
Uomini	87	81	93%	270	234	87%
Donne	263	192	73%	80	55	69%
Dati combinati	350	273	78%	350	289	83%

Questi risultati sembrano suggerire che la prescrizione di questo nuovo farmaco non abbia un impatto positivo sulla guarigione dei pazienti. Tuttavia, questo risultato è in realtà un esempio del cosiddetto "paradosso di Simpson", in cui i dati aggregati relativi a un trattamento specifico sembrano indicare una perdita di efficacia, mentre i dati delle singole categorie mostrano risultati opposti. Questo esempio sottolinea il fatto che l'interpretazione di dati aggregati non sempre può essere affidabile, e talvolta può ingannare. In questi casi, è necessario estrarre le informazioni sulla causalità dai dati individuali.

È evidente che comprendere le cause di un determinato effetto o insieme di effetti non è un compito banale. Anche conoscendo l'agente patogeno o almeno la sua natura, non sempre è sufficiente per spiegare la complessità delle interazioni. L'utilizzo di modelli di apprendimento automatico per l'analisi dei dati, la ricerca di correlazioni e la successiva formulazione di politiche di intervento può rappresentare un rischio, ma al contempo offre un alleato potente nella definizione di politiche di intervento in settori estremamente delicati come quello della sanità [71].

2.2 Modelli Compartmentali

In campo epidemiologico, i modelli compartmentali rappresentano una metodologia di modellazione generica che si presta bene all'analisi globale del comportamento di una malattia infettiva. Questa tecnica di modellazione trova applicazione anche in altri ambiti scientifici, tra cui la finanza.

L'approccio matematico dei modelli compartmentali si basa sull'assunzione che, dato un gruppo di individui, sia possibile suddividerli in diverse categorie in base al loro stato di esposizione o infezione da una particolare malattia. Questo processo crea compartimenti ben distinti che possono interagire tra loro, ma rimangono chiaramente separati gli uni dagli altri.

Equazioni Ordinarie Differenziali

Un'equazione ordinaria differenziale (ODE) è un tipo di equazione che coinvolge derivate ordinarie di una funzione rispetto a una variabile indipendente, di solito il tempo. In generale, quando abbiamo una derivata di una funzione, desideriamo studiare la funzione stessa e, per farlo, dobbiamo trovare l'antiderivata (o integrale) della funzione data. Ad esempio:

$$\frac{dx}{dt}(t) = \cos(t) \Rightarrow x(t) = \sin(t) + C$$

Tuttavia, risolvere un'ODE può essere più complesso rispetto a calcolare un semplice integrale. Anche se il principio fondamentale è lo stesso (risolvere un integrale), la difficoltà principale consiste nel determinare quale tipo di integrazione sia necessario per ottenere la soluzione desiderata.

Nel caso in cui le equazioni associate a $\frac{dx}{dt}$ dipendano solo dalla variabile t e non dalla funzione $x(t)$, la loro soluzione risulta molto più semplice di quanto possa sembrare. Tuttavia, quando $\frac{dx}{dt}$ dipende da $x(t)$, la situazione diventa più complessa e richiede spesso manipolazioni matematiche, come l'applicazione delle regole di derivazione o la sostituzione delle variabili (*chain rules* o *u-substitution*).

2.2.1 Il modello Kermack-McKendrick

Secondo la definizione sopra riportata, è possibile suddividere la popolazione oggetto di studio in categorie o compartimenti, stabilendo una relazione temporale che descrive il passaggio degli individui da un compartimento all'altro. Malattie che conferiscono immunità avranno una struttura compartmentale diversa rispetto a malattie che non la conferiscono.

Il modello più utilizzato come riferimento per lo studio e la modellazione epidemiologica è il modello "Susceptible, Infectious, Recovered" (SIR), ideato nel 1927 da Kermack e McKendrick. Questo modello si basa sull'assunzione che, durante lo sviluppo di una malattia, gli individui possano trovarsi in uno dei tre stati:

- **Susceptible (S):** Rappresenta lo stato iniziale per la maggior parte degli individui all'interno di una popolazione, indicando il numero di persone che possono contrarre la malattia.
- **Infectious (I):** Questo stato rappresenta gli individui che, partendo dallo stato suscettibile, diventano infetti dopo essere venuti in contatto con individui infetti.

- **Recovered (R):** Questo stato rappresenta gli individui che, al termine della malattia, sopravvivono o muoiono a causa di essa. A volte, questo stato è chiamato anche "Removed".

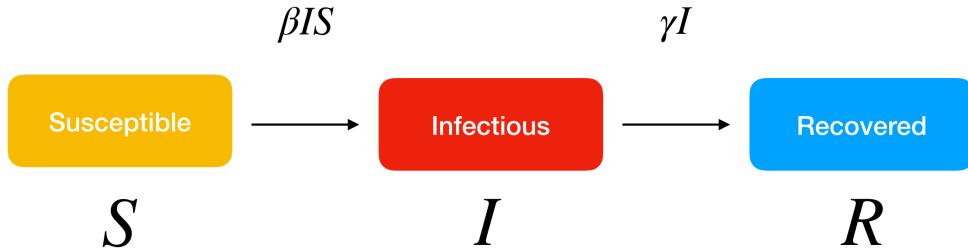


Figure 3: Esempio della struttura del modello SIR

Da questa idea di base, è stato sviluppato un modello matematico che descrive come queste tre categorie, sebbene separate, interagiscano nel tempo. Il tempo è considerato la variabile indipendente del modello, denotata come t , e i tassi di transizione tra i compartimenti sono espressi come derivate rispetto al tempo e alle dimensioni dei compartimenti. Di conseguenza, questo modello è stato inizialmente formulato utilizzando equazioni differenziali ordinarie [11].

Il sistema di equazioni differenziali che descrive il modello SIR è il seguente:

$$\begin{aligned}\frac{dS}{dt} &= -\beta \cdot S \cdot I \\ \frac{dI}{dt} &= \beta \cdot S \cdot I - \gamma \cdot I \\ \frac{dR}{dt} &= \gamma \cdot I\end{aligned}$$

Questo sistema modella le seguenti assunzioni:

- In media, un individuo nella popolazione ha abbastanza contatti con gli altri da permettere la diffusione dell'infezione, con un tasso di infezione rappresentato da $\beta \cdot N$, dove N è il numero totale di individui nella popolazione.
- Gli individui infetti lasciano il compartimento degli infetti a un tasso di $\gamma \cdot I$.
- Non ci sono entrate o uscite di individui dalla popolazione totale, tranne per le possibili uscite dovute alla morte causata dalla malattia stessa.

L'indice R_0 rappresenta il numero di infezioni secondarie causate da un singolo individuo durante il suo periodo infettivo in una popolazione completamente suscettibile di dimensione $K \approx S(0)$. In questa situazione, un individuo infetto effettua $\beta \cdot K$ contatti per unità di tempo, ognuno dei quali può trasmettere l'infezione a un individuo suscettibile, producendo nuove infezioni durante il suo periodo medio di infettività di $\frac{1}{\gamma}$. Pertanto, il valore di R_0 è $\beta \cdot \frac{K}{\gamma}$.

Il valore di R_0 è cruciale nel determinare se si verificherà o meno un'epidemia. Se $R_0 < 1$, l'infezione si estinguereà prima di diventare un'epidemia. Se $R_0 > 1$, ci sarà un'epidemia.

La modellazione epidemiologica può diventare più complessa in presenza di una fase di esposizione, in cui gli individui infetti non sono immediatamente infettivi. In questo caso, è possibile aggiungere un compartimento "Exposed (E)" al modello, che influenzera il sistema di equazioni differenziali. Questo tipo di modello è noto come SEIR (Susceptible, Exposed, Infectious, Recovered).

Il sistema di equazioni differenziali che descrive il modello SEIR è il seguente:

$$\frac{dS}{dt} = -\beta(N) \cdot S \cdot I$$

$$\frac{dE}{dt} = \beta(N) \cdot S \cdot I - \kappa \cdot E$$

$$\frac{dI}{dt} = \kappa \cdot E - \gamma \cdot I$$

$$\frac{dR}{dt} = \gamma \cdot I$$

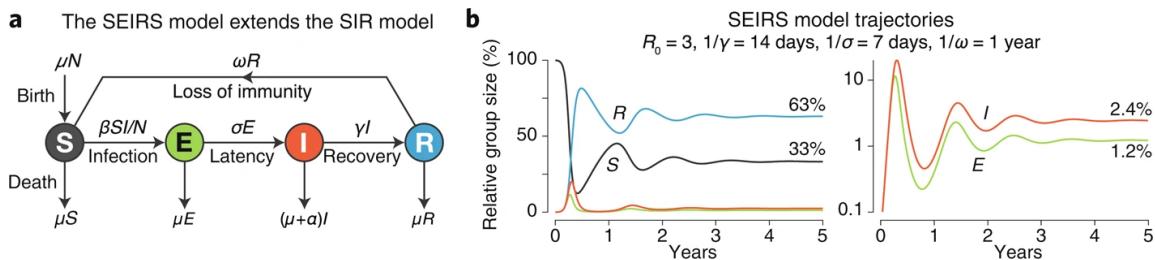


Figure 4: Esempio di modello SEIRS [8]

Un altro sviluppo importante è l'analisi dei modelli epidemiologici in forma stocastica, che tiene conto della variabilità individuale e delle piccole dimensioni dei compartimenti all'inizio di un'epidemia. Questi modelli stocastici considerano gli effetti casuali nei contatti tra gli individui e possono essere particolarmente rilevanti nelle prime fasi di un'epidemia, quando il numero di infetti è ancora relativamente basso. In queste situazioni, è necessario utilizzare approcci stocastici, come i modelli basati su reti (network models) per catturare la dinamica epidemica in modo più accurato.

In generale, i modelli compartmentali sono strumenti essenziali per la modellazione e la comprensione della diffusione delle malattie infettive, consentendo agli epidemiologi di studiare scenari diversi e sviluppare strategie di controllo più efficaci.

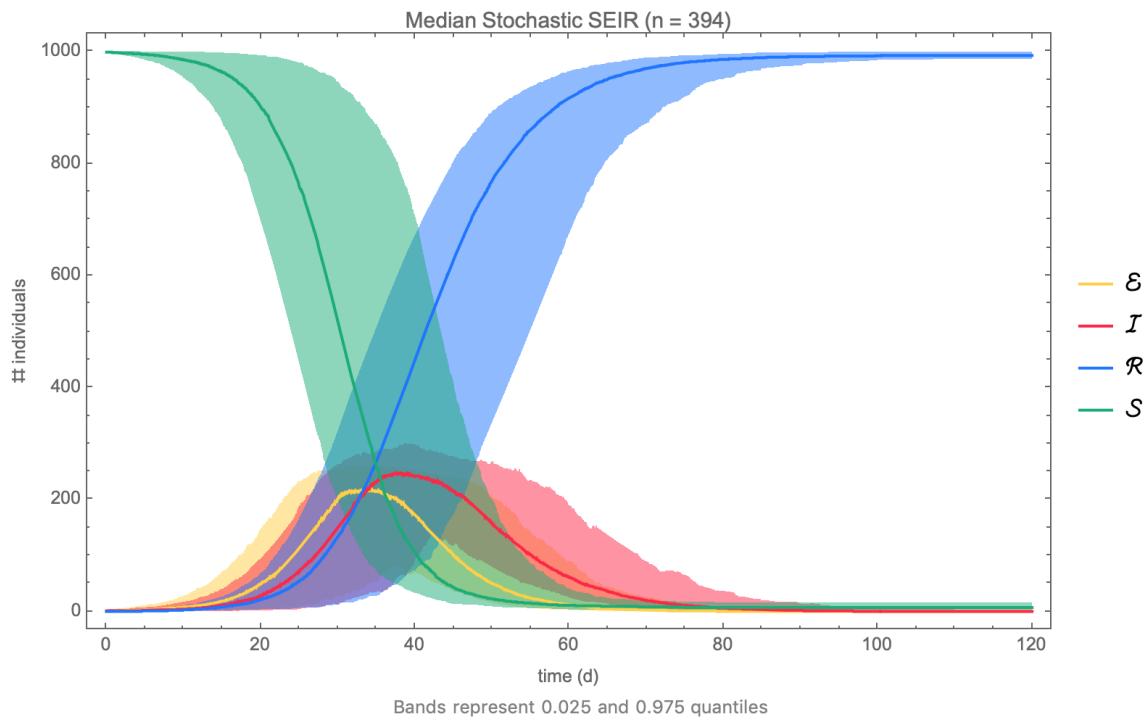


Figure 5: Esempio di modello SEIR stocastico
https://community.wolfram.com/c/portal/getImageAttachment?filename=stochastic_SEIR.png&userId=228444

2.3 Modelli basati su Agenti

I modelli basati su agenti rappresentano una metodologia computazionale utilizzata per simulare le azioni e le interazioni di un insieme di agenti autonomi, che possono essere individui o gruppi di individui. Lo scopo principale di tali modelli è comprendere il comportamento di un sistema e le relazioni che influenzano i suoi risultati [82].

Nella modellazione basata su agenti, un sistema viene suddiviso in un insieme di entità decisionali autonome chiamate "agenti". Ciascun agente valuta autonomamente la propria situazione e prende decisioni basate su un insieme di regole specifiche. Gli agenti possono eseguire una serie di comportamenti che sono appropriati per il sistema che rappresentano. Le interazioni competitive e ripetute tra gli agenti costituiscono una caratteristica distintiva di un modello basato su agenti, sfruttando la potenza computazionale dei computer per esplorare dinamiche altrimenti difficilmente accessibili attraverso la modellazione matematica tradizionale.

In una forma di base, un modello basato su agenti consiste in un sistema di agenti e delle relazioni tra di essi. Anche un modello estremamente semplice può rivelare comportamenti complessi e offrire informazioni preziose sulle dinamiche del mondo reale che stanno modellando.

Generalmente, gli agenti possono "evolvere" nel tempo, consentendo di osservare comportamenti precedentemente inaccessibili, noti come "comportamenti emergenti". I modelli più sofisticati spesso incorporano reti neurali, algoritmi evolutivi o altre tecniche di apprendimento per rendere la simulazione il più realistica possibile.

L'uso di modelli basati su agenti in epidemiologia è noto da diversi decenni ed è stato utilizzato per simulare e comprendere una vasta gamma di problemi, spesso centrati sul comportamento umano come parametro chiave [36] [25] [77] [7]. Uno dei parametri più rilevanti che è stato preso in considerazione è stato il comportamento sociale degli individui. Questo insieme di parametri comprende diverse interazioni specifiche, che possono essere raggruppate in macrocategorie se necessario.

Con l'avvento della pandemia di COVID-19, molti ricercatori si sono concentrati sulla creazione di modelli basati su agenti, sia puri che ibridi con equazioni differenziali, al fine di sviluppare simulazioni affidabili del decorso di una pandemia, tenendo conto di variabili stocastiche e imprevedibili come il comportamento umano. Questo approccio ha lo scopo di ottenere intuizioni sul comportamento emergente del sistema.

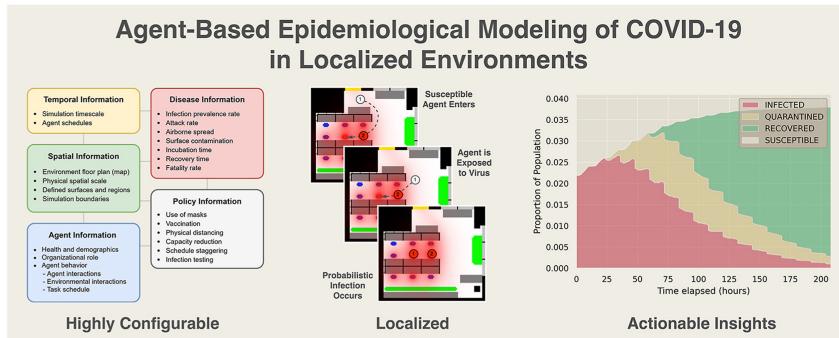


Figure 6: Agent-Based Epidemiological Modeling of COVID-19 in Localized Environment [17]

2.3.1 Comportamento Emergente

I fenomeni emergenti sono il risultato delle interazioni tra singole entità. Non possono essere spiegati completamente attraverso le proprietà intrinseche delle singole parti, poiché dipendono dalle interazioni tra tali parti. Il comportamento emergente può manifestare proprietà separate dalle singole parti, rendendo la comprensione e la previsione del comportamento del sistema complesse, spesso contrarie all'intuito.

Un modello basato su agenti genera comportamenti emergenti dal basso verso l'alto, cioè dai singoli agenti all'intero gruppo. Ciò solleva domande sull'essenza stessa del comportamento emergente.

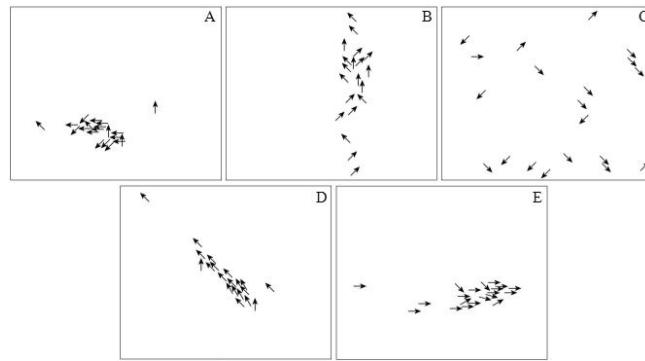


Figure 7: Esempio di comportamento emergente nella simulazione degli stormi di uccelli
<https://www.jasss.org/13/2/8/Figure6b.jpg>

2.3.2 Discretizzazione

La discretizzazione è una delle sfide fondamentali nella simulazione, in quanto il mondo reale è continuo mentre gli strumenti di simulazione attuali operano su dati discreti. Durante la simulazione di eventi, è necessario decidere come adattare la realtà alla simulazione, con il rischio di perdere informazioni nel processo.

La discretizzazione in una simulazione può riguardare principalmente due aspetti: lo spazio e il tempo. Molti framework di simulazione offrono la possibilità di specificare come gestire la discretizzazione, ad esempio attraverso l'uso di parole chiave come "ContinuousSpace". Tuttavia, la scelta della discretizzazione può influenzare significativamente l'adeguatezza della simulazione per un dato contesto.

Va notato che la discretizzazione non è sempre negativa e che alcune simulazioni possono fornire risultati accurati anche con una discretizzazione significativa. La scelta della discretizzazione dipende spesso dalla specifica applicazione e dall'obiettivo della simulazione.

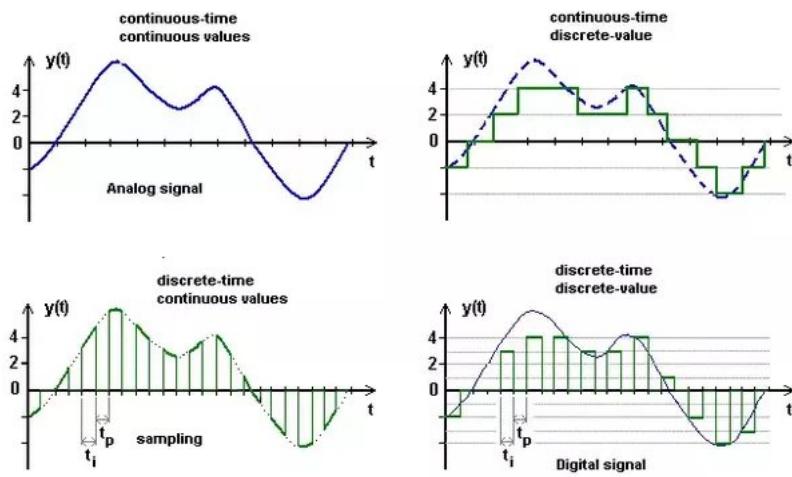


Figure 8: Esempio di differenti tipologie di discretizzazione, siano esse nel tempo o nei valori
<https://qph.cf2.quoracdn.net/main-qimg-100fc99cfe855462247225a07e1dfb7e-pj1q>

2.4 Julia

Il linguaggio di programmazione Julia rappresenta un sistema di alto livello, multi-paradigma e open-source concepito per l'analisi numerica e l'esecuzione efficiente di operazioni di computer science. Julia è emerso come linguaggio di programmazione ufficiale nel 2012 con l'obiettivo di fornire un potente strumento che rivalesse, se non superasse, i linguaggi di riferimento nel settore, come C e Fortran, in termini di velocità e affidabilità. Tuttavia, Julia è stato progettato anche per essere di facile accesso anche a coloro che non possiedono una solida base di programmazione. Il linguaggio è stato sviluppato principalmente in C++ e Scheme, ma gran parte del suo ecosistema è stato scritto in Julia stesso.

Le principali caratteristiche di Julia sono:

- **Elevate prestazioni:** Julia è stato creato con l'obiettivo di offrire prestazioni eccezionali, con la capacità di compilare programmi in codice nativo per diverse piattaforme, sfruttando LLVM.
- **Dinamismo:** La scelta di adottare una tipizzazione dinamica rende Julia più accessibile a chiunque, anche a coloro che non hanno una solida base di programmazione. Ciò offre anche un alto supporto per l'interazione in tempo reale.
- **Ambiente riproducibile:** Julia mira a garantire che le condizioni di esecuzione di un programma siano ricreabili su qualsiasi macchina. Questo obiettivo è raggiunto attraverso l'uso di file binari pre-compilati.
- **Componibilità:** Julia fa uso dell'approccio del "multiple dispatch" come paradigma di programmazione, consentendo una grande flessibilità nella rappresentazione di una vasta gamma di pattern di programmazione, dall'orientato agli oggetti a quello funzionale.
- **General-purpose:** Julia mira a creare un ecosistema in grado di soddisfare qualsiasi esigenza dell'utente, consentendo la creazione di applicazioni e microservizi senza la necessità di ricorrere a integrazioni con codice non nativo di Julia.
- **Open source:** Julia abbraccia la filosofia open source, con il codice sorgente del linguaggio e di tutte le librerie disponibili su GitHub sotto la licenza MIT. Questo favorisce una crescita eterogenea grazie al contributo di oltre 1000 utenti impegnati nel migliorare il linguaggio.

2.4.1 Agents.jl

In linea con la filosofia di sviluppo di Julia, la libreria Agents.jl [21] è stata progettata con l'obiettivo di essere facilmente apprendibile, estendibile e di offrire modelli di simulazione veloci e scalabili. Numerosi confronti hanno dimostrato che questa libreria offre significativi vantaggi prestazionali rispetto ai principali competitor presenti sul mercato, come Mesa, NetLogo e MASON [1].

La facilità d'uso di Agents.jl non va confusa con una mancanza di opzioni di sviluppo, poiché la libreria consente l'integrazione con altre librerie in modo altrettanto semplice e rapido. In particolare, offre un supporto per l'uso di machine learning, in particolare nel campo del Scientific Machine Learning [67], un campo che ha suscitato un crescente interesse, soprattutto a causa della pandemia da Covid-19.

Agents.jl offre diverse opzioni di configurazione, ma si basa principalmente su due principi:

- **Definizione del tipo di agente:** È generalmente consigliato estendere la tipologia "StandardABM", che rappresenta l'implementazione predefinita di un modello basato su agenti. Questo tipo consente la creazione di un "AgentBasedModel".
- **Definizione del tipo di spazio:** Sono supportati principalmente due tipi di spazio: spazio discreto a grafo [26] e spazio discreto a griglia. Entrambi richiedono attributi specifici per rappresentare la posizione degli agenti nello spazio.

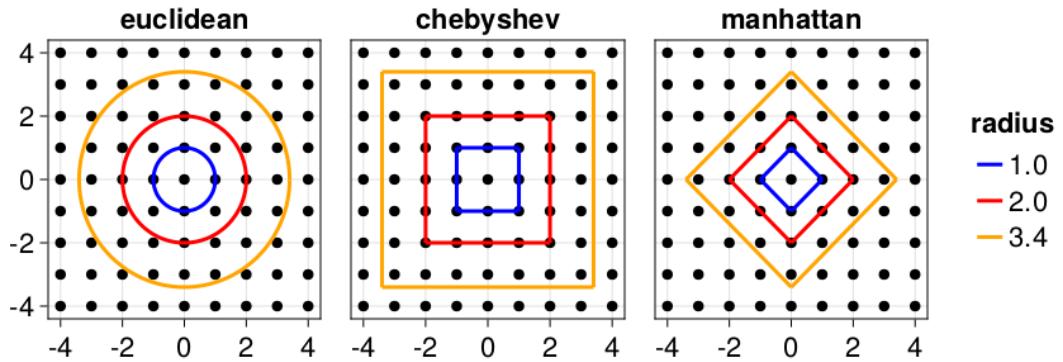


Figure 9: Esempio di metriche per il calcolo della distanza su uno spazio a griglia [21]

Inoltre, è possibile utilizzare uno spazio continuo, che rappresenta uno spazio di dimensione variabile con attributi di posizione e velocità. Infine, esiste uno spazio misto, chiamato "OpenStreetMapSpace", che rappresenta mappe come entità continue.

2.4.2 Benchmarking e confronto con altri linguaggi

Uno degli aspetti di maggior rilevanza nel contesto di Julia è la sua eccellente performance rispetto ad altri linguaggi di programmazione. Questo è stato dimostrato in diversi benchmark, in cui Julia ha superato linguaggi come Python, R e MATLAB [1] in termini di velocità computazionale. Questo vantaggio è particolarmente importante nelle simulazioni basate su agenti, in cui la velocità di esecuzione può fare la differenza tra la fattibilità e l'inapplicabilità di un modello.

	Extreme-scale	Repast HPC	MATSIM, PDES-MAS, Swarm
High /Large-scale	Altreva Adaptive Modeler, SeSAM	AnyLogic (2D/3D), AOR Simulation, CloudSim, CybelePro, FLAME, LSD (2D/3D), MASS, Pandora, UrbanSim	Agent Cell (2D/3D), Brahms, BSim (2D/3D), D-OMAR, Echo, Ecolab, FLAME GPU (3D), GridABM, HLA_Agent, HLA_RePast, Repast-J or Repast-3, Repast Symphony (2D/3D)
	NetLogo (2D/3D)	Ascape, CRAFTY, GAMA (2D/3D), SimEvents (MATLAB®), Simio (2D/3D), Simul8 (2D/3D)	MASON (2D/3D)
Medium-scale	JAS, VSEdit	Agent Factory, Breve (3D), Cormas, Envision, GALATEA, IDEA, JAMSIM, Janus, JASA, JAS-mine, MACSimJX, Mathematica® (Wolfram), Mimosa, MIMOSE, Mobility Testbed, Modgen, OBEUS, SimAgent, SimBioSys, TerraME, Xholon (2D/3D)	DigiHive, MASyV (2D/3D)
	AgentSheets, BehaviourComposer (2D/3D), FlexSim (2D/3D)	Eve, ExtendSim (2D/3D), GROWLab, Insight Maker, Mesa	
Light-weight /Small-scale	AgentScript, Framsticks (2D/3D), JAMEL, JCASim (1D/2D/3D), jES, MOBIDYC, PedSim, PS-I, Scratch (2D/3D), SimI, SimSketch, SOARS, StarLogo, StarLogoTNG (3D), Sugarscape, VisualBots	SEAS (2D/3D)	
	Simple/Easy	Moderate	Complex/Hard
	Model Development Effort →		

Figure 10: Tabella comparativa [1]

Il confronto con Python è particolarmente rilevante, dato che Python è ampiamente utilizzato in campo scientifico e computazionale. Mentre Python offre una vasta gamma di librerie e strumenti, il suo utilizzo in simulazioni basate su agenti può essere limitato dalla sua lentezza. Julia, gra-

zie alla sua architettura e alle sue librerie ottimizzate, riesce a superare questa limitazione senza compromettere la facilità d'uso [67] [64] [66] [59] [41] [42].

Julia è emerso come un linguaggio di programmazione di alto livello, versatile ed efficiente, che ha trovato applicazioni in vari campi, tra cui la simulazione basata su agenti. La libreria Agents.jl ha semplificato notevolmente lo sviluppo di modelli di simulazione basati su agenti in Julia, offrendo prestazioni eccezionali e una vasta gamma di funzionalità.

Il confronto con altri linguaggi, in particolare con Python, ha dimostrato che Julia può offrire un notevole vantaggio in termini di prestazioni computazionali, senza sacrificare la facilità d'uso e la versatilità.

In conclusione, Julia e la libreria Agents.jl rappresentano strumenti potenti per la creazione di modelli di simulazione basati su agenti che richiedono elevate prestazioni computazionali. La combinazione di un linguaggio di alto livello, come Julia, con una libreria specializzata come Agents.jl, offre un ambiente ideale per la progettazione e lo sviluppo di simulazioni avanzate in una vasta gamma di settori.

2.5 Equazioni Neurali Differenziali

Sin dalla pubblicazione dell'articolo di Chen et al. (2019) [16], la tecnica delle Equazioni Neurali Differenziali (END) ha attirato notevole attenzione, portando all'ibridazione di due paradigmi di modellazione distinti: le Equazioni Differenziali Ordinarie (ODE) e le reti neurali (NN). Questo sforzo mira a sfruttare al meglio entrambi i paradigmi, minimizzando gli effetti indesiderati [47] [16].

Un'Equazione Differenziale è un metodo per specificare una trasformazione non lineare arbitraria, codificando matematicamente le ipotesi strutturali a priori del sistema. Esistono tre approcci comuni per definire tale trasformazione non lineare:

- **Modellazione diretta**
- **Machine learning**
- **Equazioni differenziali**

L'approccio di modellazione diretta funziona solo quando si conosce la funzione esatta che collega l'input con l'output. Tuttavia, nella maggior parte dei casi, questa relazione è sconosciuta a priori, rendendo impossibile applicare questo metodo. In questi casi, l'approccio di machine learning diventa una soluzione valida.

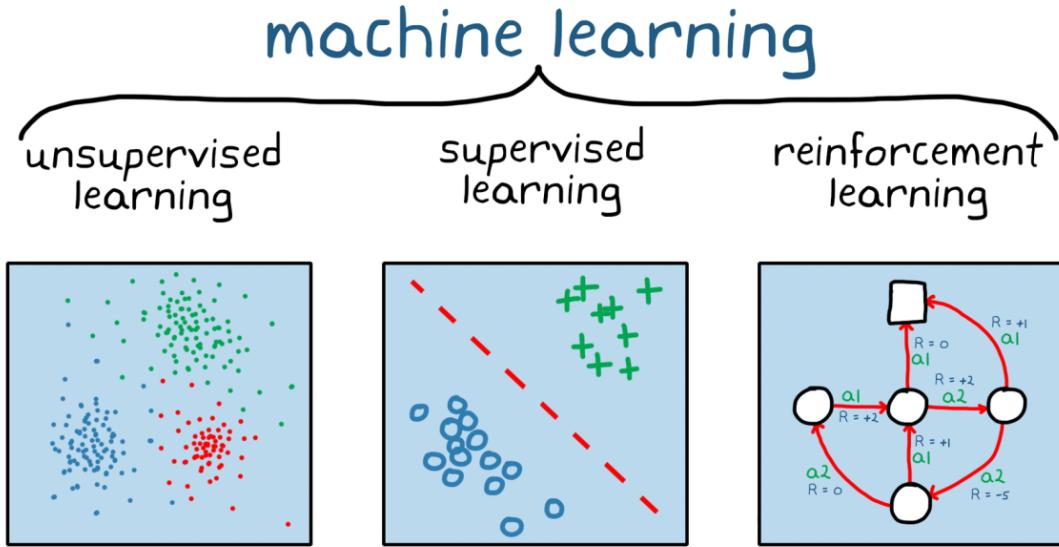


Figure 11: Differenti approcci di apprendimento di machine learning
https://it.mathworks.com/discovery/reinforcement-learning/_jcr_content/mainParsys3/discoverysubsection/mainParsys/image.adapt.full.medium.png

Nel contesto generale del machine learning, dato un insieme di dati x , l'obiettivo è predire un insieme y di dati correlati utilizzando una funzione sottostante. Questa funzione viene comunemente chiamata "modello". Si inizia con una fase di addestramento in cui si cercano di regolare i parametri

(iperparametri) del modello per ottenere un modello in grado di generare previsioni accurate. Successivamente, il modello viene utilizzato per inferire dati x mai visti in precedenza. In sostanza, questo approccio consiste in una serie di trasformazioni non lineari.

L'uso del machine learning è affascinante perché si basa su un concetto estremamente semplice ma potente: adattare il modello ai dati forniti in modo efficace. Questa idea si estende ulteriormente alle reti neurali (NN), che sono essenzialmente insiemi di moltiplicazioni tra matrici seguite dall'applicazione di una funzione di attivazione. Ad esempio, una semplice rete neurale a tre strati è definita come:

$$ML(x) = \sigma(W_3 \cdot \sigma(W_2 \cdot \sigma(W_1 \cdot x)))$$

Dove W_1 , W_2 , e W_3 sono parametri apprendibili. L'obiettivo è selezionare questi parametri in modo che $ML(x) = y$ si comporti in modo simile alla funzione incognita che si desidera adattare. Grazie all'applicazione del teorema di approssimazione universale, si afferma che con un numero sufficientemente grande di parametri o strati, è possibile approssimare qualsiasi funzione non lineare in modo sufficientemente preciso.

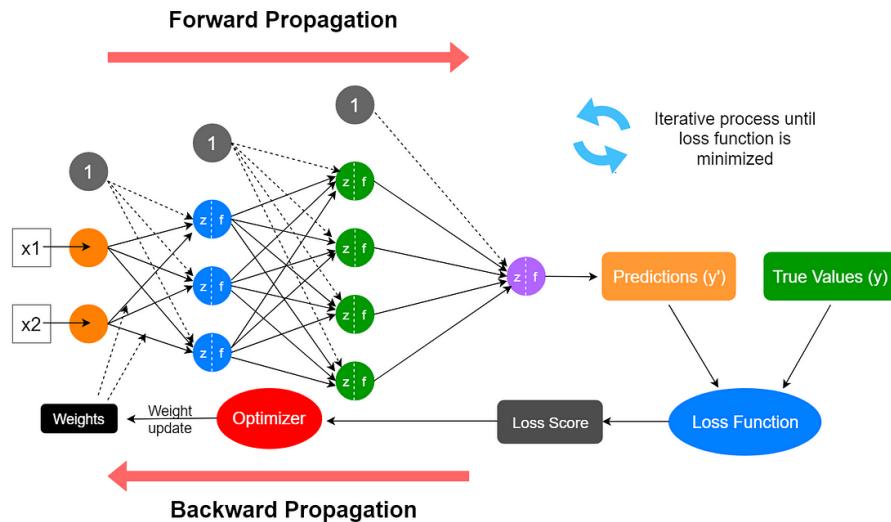


Figure 12: Esempio di funzionamento di una rete neurale
https://miro.medium.com/v2/resize:fit:4800/format:webp/1*ZXAOUqmlECgfvA81Sr6Ew.png

Questo approccio richiede l'apprendimento di ogni aspetto della trasformazione non lineare direttamente dai dati disponibili. In molti casi, però, non è possibile conoscere l'intera equazione non lineare, ma forse se ne conosce la struttura generale. Un modo per definire matematicamente questo tipo di approccio è attraverso le equazioni differenziali. Un metodo immediato è quello di definire un modello matematico in cui si cerca di apprendere una costante associata al comportamento di un insieme di dati:

$$y'(t) = \alpha \cdot y(t)$$

Questo approccio non richiede la conoscenza della soluzione dell'equazione differenziale per convalidare la correttezza del modello. La struttura del modello e la matematica stessa sono incorporate nel modello stesso, il quale successivamente produce una soluzione. Questo tipo di modelli è essenzialmente un insieme di equazioni che descrivono il cambiamento delle variabili nel tempo, con la soluzione dell'equazione differenziale come risultato finale.

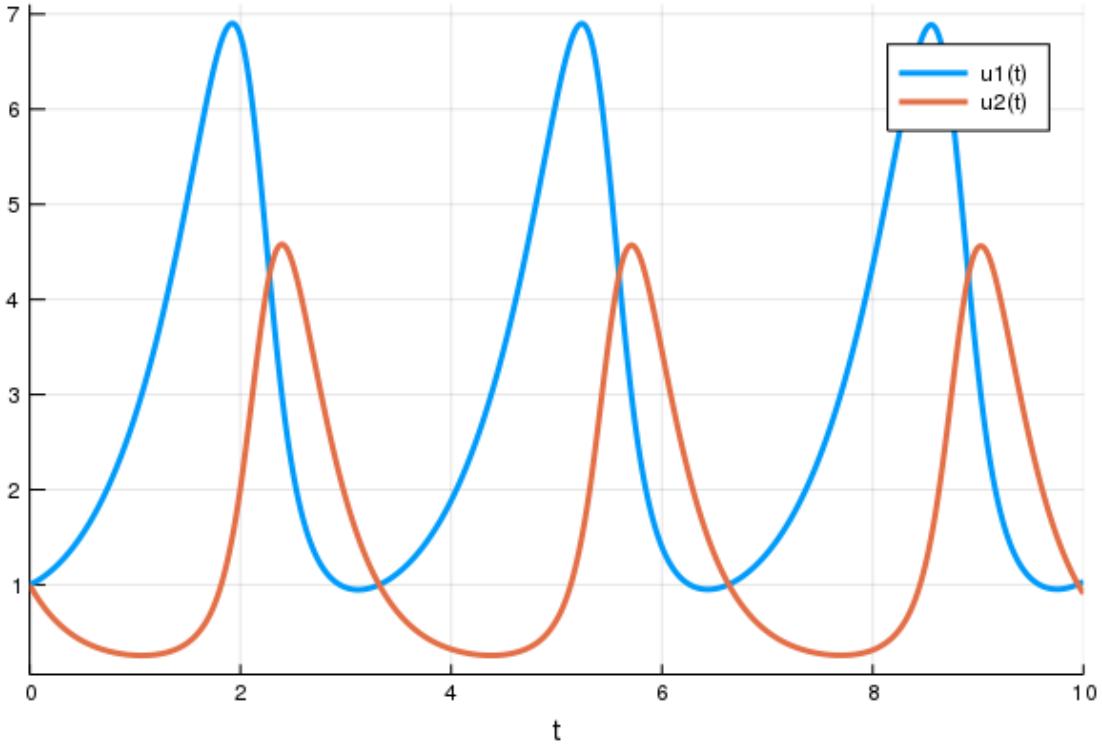


Figure 13: Esempio grafico ODE rappresentante la famosa equazione di Lotka-Volterra

Questo metodo è stato ampiamente utilizzato in campi scientifici, e recentemente ha visto ulteriori sviluppi tramite l'integrazione di strutture matematiche aggiuntive che permettono di modellare sistemi complessi come quelli in ambito farmacologico o biologico.

Ciò che rende i modelli di machine learning interessanti è la loro fame di dati e la necessità di un grande insieme di dati di allenamento. In questo contesto, l'uso delle equazioni differenziali è diventato una scelta interessante per specificare la non linearità in modo apprendibile (tramite parametri) in modo più efficiente. Questo permette di incorporare la conoscenza specifica di un dominio nelle relazioni strutturali tra input e output del modello.

Un'Equazione Differenziale Neurale (Neural ODE) rappresenta un modo per collegare i mondi del machine learning e delle equazioni differenziali. L'approccio generale è quello di non apprendere solo la trasformazione non lineare tra i dati, ma anche la struttura stessa della trasformazione non

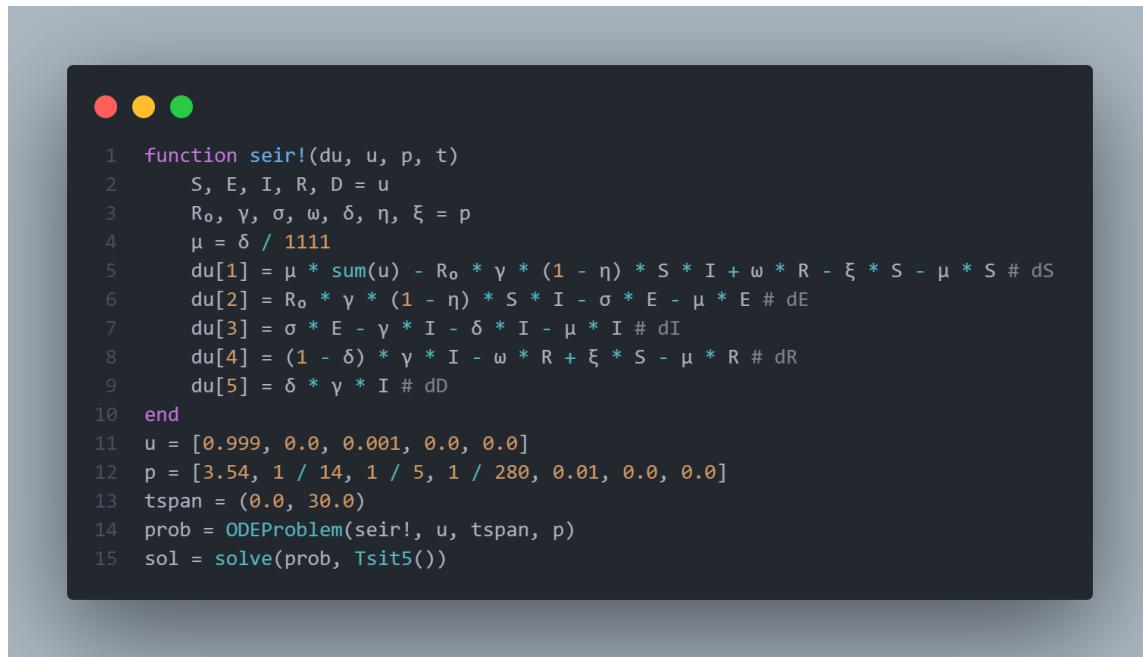
lineare. Invece di avere il modello $y = ML(x)$, si ha il modello $y' = ML(x)$, e successivamente si tenta di risolvere l'equazione differenziale associata.

L'approccio fondamentale qui è che definendo il modello in questo modo e utilizzando un risolutore semplice e propenso agli errori come il metodo di Euler, è possibile ottenere risultati equivalenti a quelli ottenuti con una ResNet [39]. L'idea di base è che invece di modellare una rete neurale con sempre più strati, diventando sempre più profonda, è sufficiente modellare direttamente il sistema di equazioni differenziali e risolverlo con un risolutore specifico.

Questo approccio è efficiente dal punto di vista della memoria, è in grado di gestire dati irregolari, ha una forte conoscenza a priori dello spazio del modello, è in grado di approssimare sia funzioni lineari che non lineari e si basa su solide basi teoriche derivanti da entrambi i lati.

Risolvere un'ODE in Julia

L'idea di base è quella di definire un oggetto "ODEProblem" specificando una funzione che descrive il sistema di ODE attraverso la specifica delle derivate delle equazioni nella forma $u' = f(u, p, t)$, fornendo un set di condizioni iniziali u_0 , un intervallo di tempo t , e un insieme di parametri p .



```

1  function seir!(du, u, p, t)
2      S, E, I, R, D = u
3      R₀, γ, σ, ω, δ, η, ξ = p
4      μ = δ / 1111
5      du[1] = μ * sum(u) - R₀ * γ * (1 - η) * S * I + ω * R - ξ * S - μ * S # dS
6      du[2] = R₀ * γ * (1 - η) * S * I - σ * E - μ * E # dE
7      du[3] = σ * E - γ * I - δ * I - μ * I # dI
8      du[4] = (1 - δ) * γ * I - ω * R + ξ * S - μ * R # dR
9      du[5] = δ * γ * I # dD
10 end
11 u = [0.999, 0.0, 0.001, 0.0, 0.0]
12 p = [3.54, 1 / 14, 1 / 5, 1 / 280, 0.01, 0.0, 0.0]
13 tspan = (0.0, 30.0)
14 prob = ODEProblem(seir!, u, tspan, p)
15 sol = solve(prob, Tsit5())

```

Figure 14: Esempio definizione ODE in Julia

Successivamente, è possibile risolvere il sistema di ODE utilizzando la funzione "solve". È possibile specificare diversi metodi per la risoluzione del sistema. Ad esempio, il metodo "Tsit5" è un metodo esplicito di quinto ordine di tipo Runge-Kutta con un stimatore di errore integrato di tipo Tsitouras [78]. La libreria DifferentialEquations.jl offre una vasta gamma di parametri aggiuntivi per un approccio più dettagliato [67].

2.5.1 Inserire un'ODE all'interno di una NN

Per capire meglio cosa significhi inserire un'ODE all'interno di una rete neurale (NN), è necessario esaminare come è definito un layer di una NN. Un layer è essenzialmente una funzione differenziabile che accetta un vettore di dimensione n come input e restituisce un nuovo vettore di dimensione m come output. Questo si allinea con l'idea che i risolutori di DE rientrano nella categoria delle funzioni differenziabili, il che significa che possono essere incorporati direttamente in un programma più grande basato su differenziazione automatica, come una rete neurale. [59] [41]



```

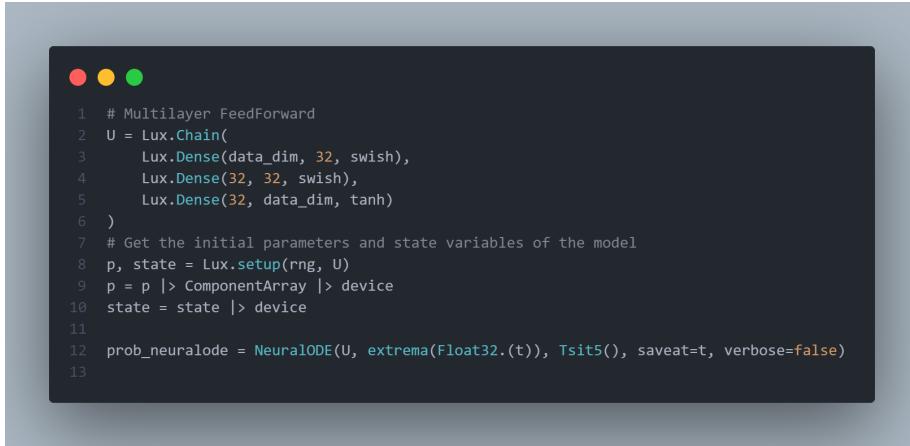
1 ann = Lux.Chain(Lux.Dense(6, 16, swish), Lux.Dense(16, 16, swish), Lux.Dense(16, 1, tanh))
2 p, state = Lux.setup(rng, ann)
3
4 function predict(p)
5     _prob = remake(prob, u0=ic, tspan=timeframe, p=p)
6     Array(solve(_prob, Tsit5()), saveat=ts, abstol=1e-10, reltol=1e-10, verbose=false))
7 end
8
9 function loss(p)
10    pred = predict(p)
11    sum(abs2, pred[3, :]) + sum(abs2, pred[5, :]) / sum(abs2, pred[6, :])
12 end
13
14 losses = Float64[]
15 callback = function (p, l; loss_step=loss_step)
16     push!(losses, l)
17     if length(losses) > 1 && (losses[end-1] - losses[end]) == 0.0
18         # exit early if not improving
19         return true
20     end
21     if length(losses) % loss_step == 0
22         @debug "Current loss after $(length(losses)) iterations: $(losses[end])"
23     end
24     return false
25 end
26 adtype = Optimization.AutoZygote()
27 optf = Optimization.OptimizationFunction((x, p) -> loss(x), adtype)
28 optprob = Optimization.OptimizationProblem(optf, ComponentVector{Float64}(p))
29
30 res1 = Optimization.solve(
31     optprob,
32     ADAM(0.01),
33     callback=callback,
34     maxiters=maxiters
35 )

```

Figure 15: Esempio implementazione di una rete neurale in Julia

Successivamente, è possibile addestrare la NN per un numero specifico di integrazioni per ottenere

i risultati desiderati. È importante sottolineare che il sistema non sta apprendendo una soluzione all'equazione differenziale, ma piuttosto sta apprendendo il sistema di equazioni differenziali da cui è generata la soluzione. La NN sta effettivamente apprendendo una rappresentazione compatta del comportamento della serie di dati nel tempo e può estrapolare facilmente cosa potrebbe accadere con diverse condizioni iniziali. La suite DiffEqFlux.jl offre un wrapper comodo per definire una "NeuralODE" [64].



```

1 # Multilayer FeedForward
2 U = Lux.Chain(
3     Lux.Dense(data_dim, 32, swish),
4     Lux.Dense(32, 32, swish),
5     Lux.Dense(32, data_dim, tanh)
6 )
7 # Get the initial parameters and state variables of the model
8 p, state = Lux.setup(rng, U)
9 p = p |> ComponentArray |> device
10 state = state |> device
11
12 prob_neuralode = NeuralODE(U, extrema(Float32.(t)), Tsit5(), saveat=t, verbose=false)
13

```

Figure 16: Esempio implementazione NeuralODE tramite DiffEqFlux.jl

2.5.2 Backpropagation tramite ODE solver

Il cuore di ogni rete neurale è la capacità di propagare all'indietro le derivate lungo l'intera rete per calcolare il gradiente della funzione di perdita rispetto ai parametri della rete. La sfida sta nel trovare un modo per applicare lo stesso principio quando si utilizzano risolutori di ODE all'interno di una rete neurale.

Esistono vari approcci per affrontare questo problema, ma il più comune è attraverso l'analisi di sensitività (adjoint). Questo approccio definisce una nuova ODE il cui obiettivo è calcolare il gradiente della funzione di costo rispetto ai parametri e successivamente risolvere questa seconda ODE.

Equazioni Differenziali Ordinarie

Nel campo matematico, un'Equazione Differenziale Ordinaria (ODE) è un tipo di equazione differenziale che coinvolge una sola variabile indipendente, solitamente il tempo. Tra le diverse categorie di equazioni differenziali, le Equazioni Differenziali Lineari occupano un ruolo predominante poiché molti fenomeni fisici e matematici possono essere descritti tramite la soluzione di tali equazioni.

Un'ODE lineare è definita da un polinomio lineare, e la sua forma generale è:

$$\alpha_0(x)y + \alpha_1(x)y' + \alpha_2(x)y'' + \dots + \alpha_n(x)y^{(n)} + b(x) = 0$$

Dove $\alpha_0(x), \dots, \alpha_n(x)$ e $b(x)$ sono funzioni differenziabili arbitrarie (non necessariamente lineari), e $y', \dots, y^{(n)}$ rappresentano le derivate successive della funzione incognita y rispetto alla variabile x .

L'utilizzo di equazioni differenziali non lineari può spesso essere approssimato con equazioni lineari per ottenere soluzioni più semplici. La libreria SciML.ai offre un ampio set di framework per la risoluzione di sistemi di equazioni differenziali, principalmente attraverso la libreria DifferentialEquations.jl, con vari metodi di risoluzione e prestazioni elevate [62].

Comparison Of Differential Equation Solver Software														
Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Holger	ODEPACK/NWlib /NAG	JACODE	PyDSTool	FATODE	GSL	BOOST	Mathematica	Maple
Language	MATLAB	Python	R	Julia	C++ and Fortran	Fortran	Fortran	Python	Python	Fortran	C	C++	Mathematica	Maple
Selection of Methods for ODEs	Fair	Poor	Poor***	Fair	Excellent	Good	Fair	Good	Poor	Poor	Fair	Poor	Fair	Fair
Efficiency*	Poor	Poor****	Poor***	Excellent	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Poor	Good	Excellent	Excellent	Good	Good	Fair	Fair	Fair	Fair	Fair	Good	Fair
Event Handling	Good	Good	Fair	Excellent	Good**	None	Good**	None	None	None	None	None	Good	Good
Symbolic Calculation of Jacobians and Auto-differentiation	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	Good	Excellent	Excellent	
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	None	Excellent	Excellent	
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	Fair	None	
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	Good	Good	
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	Good	None	
Constrained Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DSVERK)	Fair	None	None	None	Good	Excellent	
State-Dependent DDE Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	Excellent	
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	Fair	Poor	
Specialized Methods for 2nd Order ODEs and Hamiltonian Systems (Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	Poor	Good	Good	None	Good	None	None	None	None	Good	Fair	
GPU Compatibility	None	None	None	Excellent	Good	None	None	None	None	None	None	Good	None	None
Analysis Addons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Excellent	None	Good (for some methods like DAERK)	Poor	Good	None	None	Excellent	None	
* Efficiency takes into account not only the efficiency of the implementation, but the features of the implemented methods (advanced timestepping controls, existence of methods which are known to be more efficient, Jacobian handling).														
** Event handling needs to be implemented yourself using basic rootfinding functionality.														
*** There is a way to write your own C/Fortran code for the derivative, in which case it nearly matches Julia's speed.														
**** This timing includes JIT compilation with Numba, see https://github.com/JuliaDiffEq/SciPyDiffEq.jl for timing details.														
Scale	None	Poor	Fair	Good	Excellent									
Explanation	Functionality does not exist	Functionality exists, but its feature incomplete	The basic features exist	The basic features exist and some extra functionality exist	The basic features exist and some extra functionality exist. May include extra methods for efficiency.									

Figure 17: Tabella comparativa tra le varie implementazioni dei vari risolutori [62]

2.5.3 Equazioni Differenziali Universali

Recentemente, gli sviluppi nel campo del machine learning sono stati guidati dalle tecniche di deep learning, che richiedono grandi quantità di dati, noti come "big data", per risolvere problemi precedentemente considerati difficili e complessi, come il riconoscimento di immagini o il processing del linguaggio naturale.

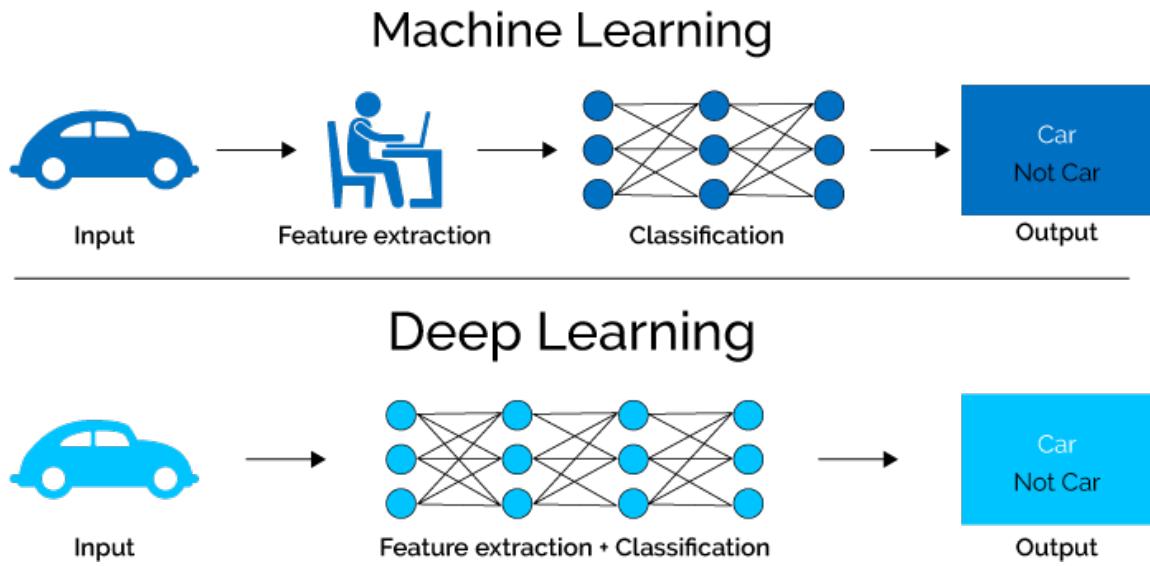


Figure 18: Esempio comparativo tra funzionamento Machine Learning e Deep Learning
https://goodboychan.github.io/images/copied_from_nb/image/fe.png

Tuttavia, in molti settori, specialmente nella medicina e in altre discipline correlate, è difficile ottenere un insieme di dati sufficientemente ampio e diversificato per applicare queste tecniche. In questi contesti, i modelli meccanicistici rimangono la scelta principale. L'approccio data-driven dei modelli di machine learning, tuttavia, offre maggiore flessibilità e la possibilità di evitare ipotesi semplificative richieste dai modelli teorici.

Un'Equazione Differenziale Universale (UDE) è definita da un "approssimatore universale", un oggetto parametrico in grado di rappresentare qualsiasi funzione dati un certo numero di parametri. Gli approssimatori universali includono, ad esempio, le serie di Fourier o di Chebyshev per spazi a basse dimensioni e le reti neurali per spazi ad alta dimensione [66].

Un'UDE è un sistema di equazioni differenziali o algebriche non triviali che può approssimare qualsiasi funzione continua su un intervallo a qualsiasi livello di precisione desiderato. In altre parole, un'UDE è in grado di rappresentare qualsiasi sistema dinamico continuo con qualsiasi grado di dettaglio richiesto.

L'approccio delle UDE può essere accoppiato con tecniche data-driven, come l'algoritmo SINDy (Sparse Identification of Non-linear Dynamics), per l'identificazione sparsa di dinamiche non lineari in sistemi complessi come dinamiche dei fluidi o reti biologiche [45].

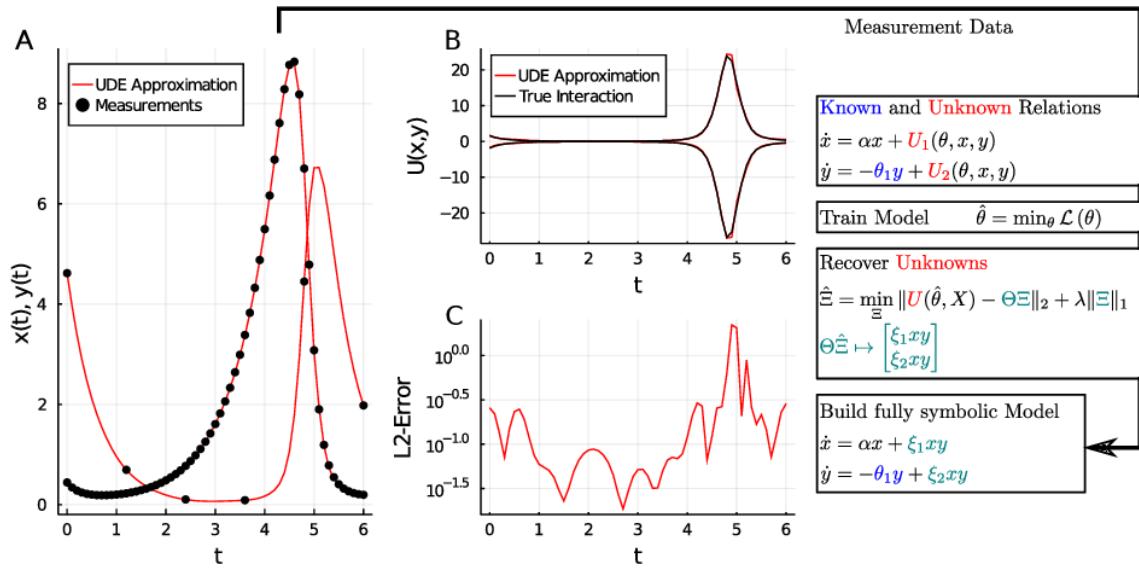
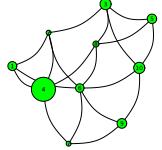


Figure 19: Comportamento UDE nell'approssimazione di fenomeni non lineari [66]

3 Metodi e Modelli

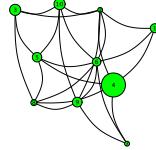
L'approccio utilizzato in questa ricerca si è basato sulla modellazione del problema nell'ambito di una rete sociale. L'obiettivo è quello di studiare la diffusione di una pandemia virale in un ambiente reale, composto principalmente da punti di interesse (PdI) collegati tra loro tramite una rete di collegamenti, che assume la forma di un grafo.

Edges coverage: low
Number of edges: 19



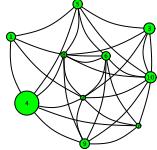
(a) Esempio di grafo connesso con copertura bassa

Edges coverage: medium
Number of edges: 24



(b) Esempio di grafo connesso con copertura media

Edges coverage: high
Number of edges: 30



(c) Esempio di grafo connesso con copertura alta

Questo approccio si avvicina a una prospettiva di tipo mesoscopico, orientata al macroscopico. La scelta di tale approccio deriva dalla necessità di modellare il sistema in maniera più ampia, concentrandosi sulla risposta collettiva di un agente astratto agli interventi mirati per contrastare l'epidemia in modo localizzato.

3.1 Approccio con Rete Sociale

Il modello sviluppato fa uso del framework "Agents.jl" [21] e definisce un modello ad agenti (ABM). In questo contesto, la struttura spaziale del modello non è rilevante, ma le connessioni tra gli agenti lo sono. Gli agenti vengono considerati come nodi all'interno di un grafo, in cui viene simulato il ciclo di vita della pandemia attraverso un modello SEIR deterministico. Gli agenti sono definiti come oggetti di tipo "ContinuousAgent", poiché lo spazio del modello è considerato continuo.

La struttura dell'agente è essenziale e include i seguenti campi principali:

- **population:** un intero che rappresenta il numero totale di individui presenti nel nodo.
- **status:** un vettore di numeri decimali che rappresenta lo stato della popolazione nel nodo, espressa come percentuale di individui.
- **param:** un vettore di numeri decimali che contiene i parametri specifici del modello SEIR per il nodo.
- **happiness:** un campo di supporto per il bilanciamento delle contromisure applicate dal controllore.

Grafo Sociale

Un grafo sociale è un tipo di grafo che rappresenta le relazioni sociali tra le entità. In questo contesto, i nodi del grafo rappresentano gli individui e gli archi rappresentano le loro relazioni sociali. Questo approccio è spesso utilizzato per modellare le reti sociali, dove gli individui sono i nodi e le loro connessioni sono gli archi.

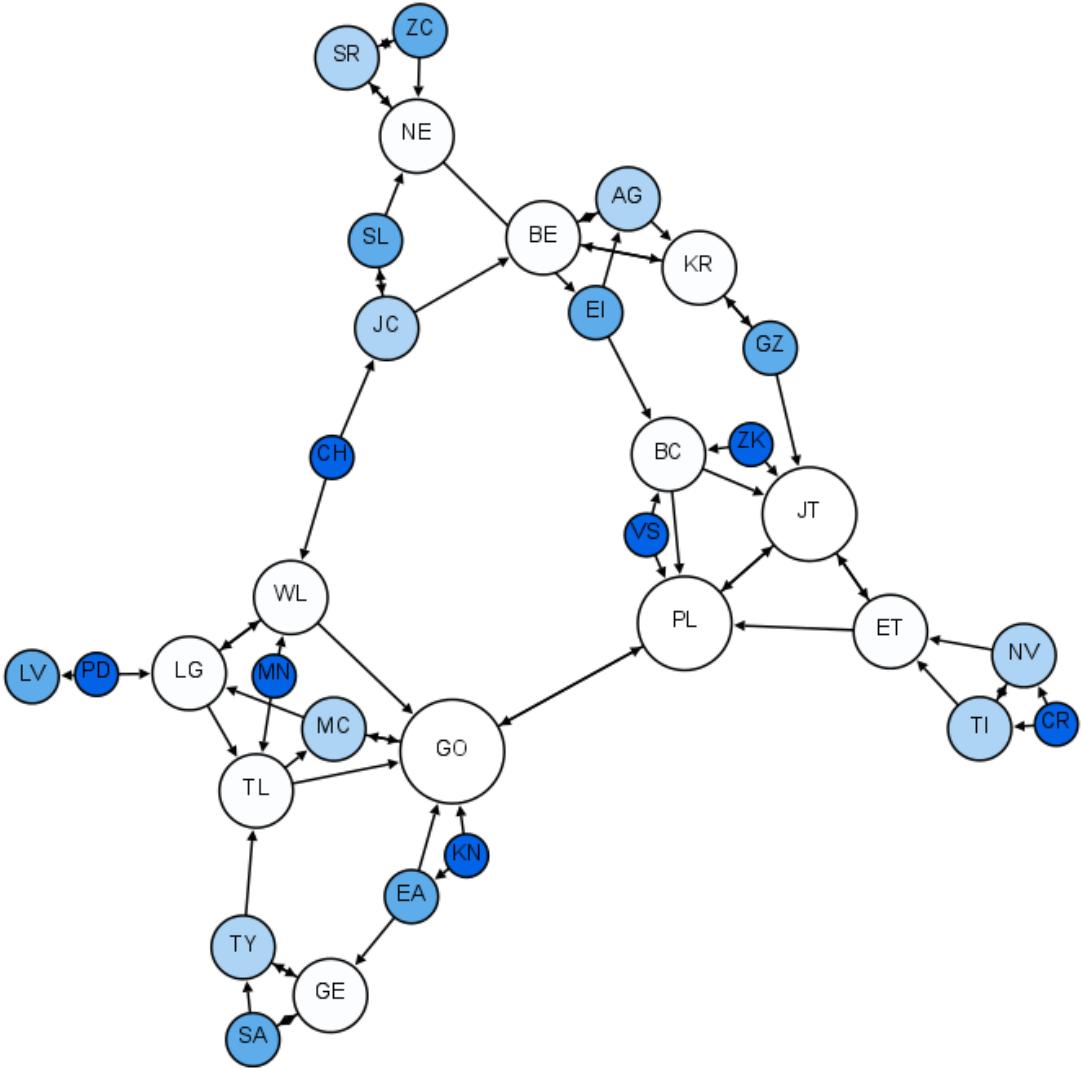


Figure 21: Moreno's Sociogram of a 3rd grade class
https://en.wikipedia.org/wiki/Sociogram#/media/File:Moreno_Sociogram_3rd_Grade.png

3.2 Agente

L'agente è stato implementato in modo minimale e contiene solo gli attributi essenziali per il funzionamento del modello e le interazioni con l'ambiente e gli altri agenti. L'evoluzione di ogni agente è indipendente dagli altri, ma può essere influenzata da agenti nella stessa rete.



```
1 @agent Node ContinuousAgent{2} begin
2     population::Int64
3     status)::Vector{Float64} # S, E, I, R, D
4     param)::Vector{Float64} # R₀, γ, σ, ω, δ, η, ξ
5     happiness::Float64 # ∈ [0, 1)
6 end
```

Figure 22: Codice Agente

3.3 Spazio e Modello



```
1 properties = @dict(numNodes,
2                     param,
3                     graph,
4                     migrationMatrix,
5                     step=0,
6                     control,
7                     vaccine,
8                     control_options,
9                     integrator=nothing)
10
11 model = ABM(Node,
12              ContinuousSpace((100, 100); spacing = 4.0, periodic = true);
13              properties = properties,
14              rng)
```

Figure 23: Codice Modello

Lo spazio del modello è basato su un grafo connesso, dove il numero di archi dipende dalla copertura desiderata degli archi. La topologia del grafo è creata in base alle preferenze dell'utente, con un limite inferiore dato dalla connessione minima e un limite superiore dato dalla connessione completa.

3.4 Funzione di Avanzamento Agente

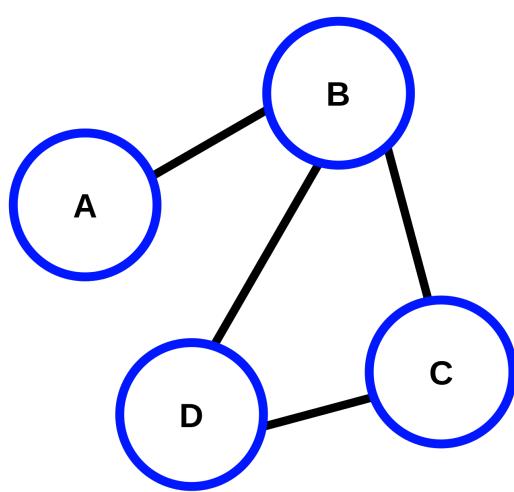
Ogni agente segue un ciclo di comportamento che include tre compiti principali: spostarsi tra i nodi del grafo [22], calcolare il proprio "livello di felicità" e chiamare il controllore per valutare le misure di intervento necessarie.

```

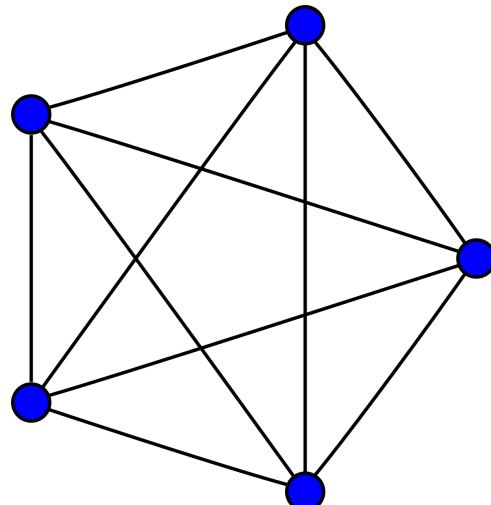
● ○ ●
1 function migrate!(agent, model::ABM)
2     network = model.migrationMatrix[agent.id, :]
3     tidxs, tweights = findnz(network)
4
5     for i in 1:length(tidxs)
6         ap = deepcopy(agent.population)
7         as = deepcopy(agent.status)
8
9         out = as .* tweights[i] .* (1 - deepcopy(agent.param[6]))
10        outp = out .* ap
11        new_status = as - out
12        new_population = sum(new_status .* ap)
13        agent.status = new_status .* ap ./ new_population
14        agent.population = round(Int64, new_population)
15
16        objective = filter(x -> x.id == tidxs[i], [a for a in allagents(model)])[1]
17        os = deepcopy(objective.status)
18        op = deepcopy(objective.population)
19
20        new_status = (os .* op) + outp
21        new_population = sum(new_status)
22        objective.status = new_status ./ new_population
23        objective.population = round(Int64, new_population)
24    end
25 end

```

Figure 25: Funzione atta a calcolare lo spostamento di agenti da un nodo all'altro del grafo



(a) Esempio di grafo connesso
https://it.wikipedia.org/wiki/Grafo_connesso#/media/File:CPT-Graphs-undirected-unweighted-ex1.svg



(b) Esempio di grafo completo
https://en.wikipedia.org/wiki/Complete_graph#/media/File:4-simplex_graph.svg

```

1  function get_migration_matrix(g::SimpleGraph,
2      population::Vector{Int},
3      maxTravelingRate::Float64)
4      numNodes = Graphs_nv(g)
5      migrationMatrix = zeros(numNodes, numNodes)
6
7      for n in 1:numNodes
8          for m in 1:numNodes
9              migrationMatrix[n, m] = (population[n] + population[m]) / population[n]
10         end
11     end
12
13     migrationMatrix = (migrationMatrix .* maxTravelingRate) ./ maximum(migrationMatrix)
14     migrationMatrix[diagind(migrationMatrix)] .= 1.0
15     mmSum = sum(migrationMatrix, dims = 2)
16
17     for c in 1:numNodes
18         migrationMatrix[c, :] ./= mmSum[c]
19     end
20
21     return migrationMatrix .* adjacency_matrix(g)
22 end

```

Figure 26: Funzione che crea la matrice di migrazione data la topologia di un grafo

```

1  function happiness!(agent)
2      agent.happiness = agent.happiness - (agent.status[3] + agent.status[5]) +
3          (agent.status[4] * (1 - agent.param[6])) - agent.param[6])
4      agent.happiness = agent.happiness < 0.0 ? 0.0 :
5          agent.happiness > 1.0 ? 1.0 : agent.happiness
6  end

```

Figure 27: Funzione atta a calcolare la felicità degli agenti

3.5 Funzione di Avanzamento Modello

Il modello coordina l'avanzamento degli agenti e si occupa di operazioni che coinvolgono l'intero sistema invece di singoli agenti. Queste operazioni includono l'aggiornamento delle equazioni dif-

ferenziali ordinarie (ODE) associate a ciascun agente e la gestione delle varianti del virus di interesse.

3.5.1 Funzione per la Generazione delle Varianti del Virus (VOC)

La generazione delle VOC è basata su alcune assunzioni, tra cui un tasso di mutazione casuale delle basi del virus e una distribuzione dei parametri pandemici. Questa semplice implementazione rappresenta un'approssimazione del comportamento delle varianti. [52] [81] [2]



```
1  function voc!(model::ABM)
2      if rand(model.rng) ≤ 8e-3
3          agent = random_agent(model)
4          if agent.status[3] ≠ 0.0
5              agent.param[1] = rand(model.rng, Uniform(3.3, 5.7))
6              agent.param[2] = rand(model.rng, Normal(agent.param[2], agent.param[2] / 10))
7              agent.param[3] = rand(model.rng, Normal(agent.param[3], agent.param[3] / 10))
8              agent.param[4] = rand(model.rng, Normal(agent.param[4], agent.param[4] / 10))
9              agent.param[5] = rand(model.rng, Normal(agent.param[5], agent.param[5] / 10))
10         end
11     end
12   end
```

Figure 28: Funzione che si occupa di generare la VOC

3.5.2 Funzione per la Simulazione della Campagna di Vaccinazione

La simulazione della campagna di vaccinazione si basa su una percentuale fissa di popolazione vaccinata a ciascun passo del modello. Questa percentuale tiene conto dell'immunità di gregge da raggiungere entro un periodo specifico.



```

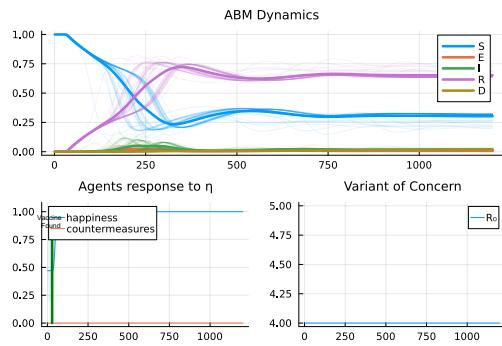
1  function vaccine!(model::ABM)
2      if rand(model.rng) < 1 / 365
3          R = mean([agent.param[1] for agent in allagents(model)])
4          vaccine = (1 - (1 / R)) / rand(model.rng, Normal(0.83, 0.083))
5          vaccine *= mean([agent.param[4] for agent in allagents(model)])
6          agent = random_agent(model)
7          agent.param[7] = vaccine
8      end
9      for agent in allagents(model)
10         if agent.param[7] > 0.0
11             network = model.migrationMatrix[agent.id, :]
12             tidxs, tweights = findnz(network)
13             id = sample(model.rng, 1:length(tidxs), Weights(tweights))
14             objective = filter(x -> x.id == tidxs[id], [a for a in allagents(model)][1])
15             objective.param[7] = agent.param[7]
16         end
17     end
18 end

```

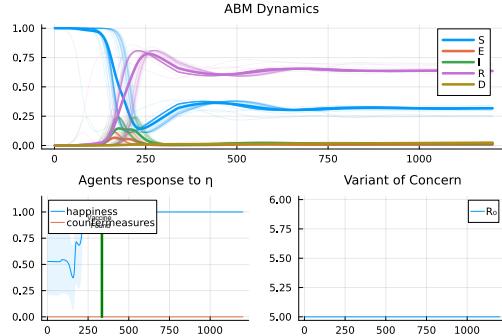
Figure 29: Funzione che si occupa di simulare la ricerca di un vaccino e la sua successiva applicazione. Queste funzioni sono soggette a diverse assunzioni, tra cui l'effetto delle mutazioni del virus sulla distribuzione dei parametri pandemici e il meccanismo di vaccinazione basato su dosi fisse. Le assunzioni semplificano il modello, ma possono non rappresentare completamente il comportamento del mondo reale.

Grafici di Confronto

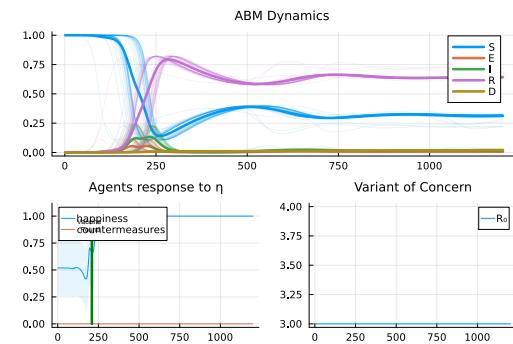
I grafici di confronto mostrano l'impatto del momento di inizio della campagna di vaccinazione su diversi scenari. Questi grafici illustrano come l'efficacia della campagna vaccinale possa dipendere dal suo avvio, oltre al numero di persone vaccinate.



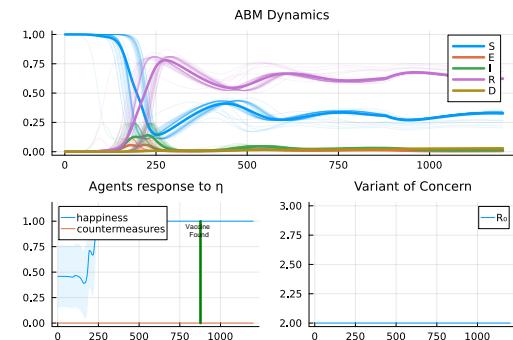
(a) Grafico per la comparazione sul periodo di inizio della campagna vaccinale. Immediata



(c) Grafico per la comparazione sul periodo di inizio della campagna vaccinale. Poco dopo la prima ondata



(b) Grafico per la comparazione sul periodo di inizio della campagna vaccinale. Durante prima ondata



(d) Grafico per la comparazione sul periodo di inizio della campagna vaccinale. In ritardo

3.6 Monitoraggio e Intervento

Il sistema di controllo e intervento adottato segue il principio del controllo autonomo mediante l'impiego di una Neural ODE. Questa strategia permette di definire un sistema di Equazioni Differenziali Ordinarie (ODE) che descriva il sistema del mondo reale con le sue relazioni, e di inserire all'interno di esso in modo mirato una rete neurale che controlli il dosaggio delle contromisure [15] [43] [72].

Questa tecnica ha dimostrato di produrre risultati eccellenti, ma è fortemente influenzata dalla corretta modellazione del sistema sottostante e dalla selezione della funzione di costo o di controllo utilizzata per addestrare il modello. La sfida principale in questo contesto è garantire una modellazione accurata del sistema di partenza e una comprensione approfondita dei risultati generati dal controllore. Questo perché il controllore restituisce valori nel range $\in [0, 1]$, i quali possono risultare difficilmente interpretabili. Pertanto, è fondamentale definire chiaramente in anticipo il significato di questo intervallo di valori nel contesto del mondo reale.

Inizialmente, la funzione di controllo incorporata nel modello agente è definita come illustrato in figura 31. È possibile notare come siano inseriti dei parametri di controllo per gestire l'invocazione effettiva del controllore.

Questi parametri di controllo sono principalmente associati a quando è necessario, dal punto di vista pratico, richiamare la funzione di controllo. Da un lato, servono per ridurre il numero di chiamate al controllore, contribuendo così a migliorare le prestazioni, e dall'altro lato, rendono più realistica l'idea di avere accesso a un sistema di controllo solo in determinati momenti temporali, anziché in modo continuo. Ciò riflette anche il ritardo tra la raccolta dei dati, la formulazione di una contromisura appropriata e la sua attuazione.

In seguito, vengono definite ulteriori regole di attivazione, questa volta legate alla prima attivazione del controllore. Retrospettivamente, è evidente come avere regole di prevenzione già attive possa essere di grande aiuto nella gestione di una pandemia. Tuttavia, a volte può esserci un ritardo nel riconoscere il pericolo, determinato dal valore di "tolerance", il quale evita l'attivazione del controllore fintanto che la situazione sembra rimanere al di sotto della soglia minima.

```

1  function control!(agent,
2      model::ABM;
3      tolerance = 1e-3,
4      dt = 30.0,
5      step = 7.0,
6      maxiters::Int = 100,
7      loss = missing,
8      u_max = missing)
9      if agent.status[3] ≥ tolerance && model.step % dt == 0
10         u_max = u_max === missing ?
11             Distributions.cdf(Distributions.Beta(2, 5), agent.status[3]) : u_max
12         agent.param[6] = controller(agent.status,
13             vcat(agent.param[1:5], agent.param[7]));
14         h = agent.happiness,
15         timeframe = (0.0, dt),
16         step = step,
17         maxiters = maxiters,
18         loss_step = Int(maxiters / 10),
19         loss_function = loss,
20         u_max = u_max,
21         rng = model.rng)
22     end
23 end

```

Figure 31: Definizione del controllore all'interno del modello ad agente

Altri parametri definiscono principalmente i fattori necessari affinché la funzione di controllo operi correttamente. Tra questi c'è un parametro opzionale v , che descrive quanto potrebbe essere il valore massimo delle contromisure applicabili in relazione al numero di individui infetti in un determinato nodo. Questo valore è calcolato come un'esponenziale rispetto al numero di infetti. Questa assunzione è utile come ulteriore valore di riferimento per il controllore nel caso in cui restituiscia un valore troppo elevato rispetto a quanto sia sensato.

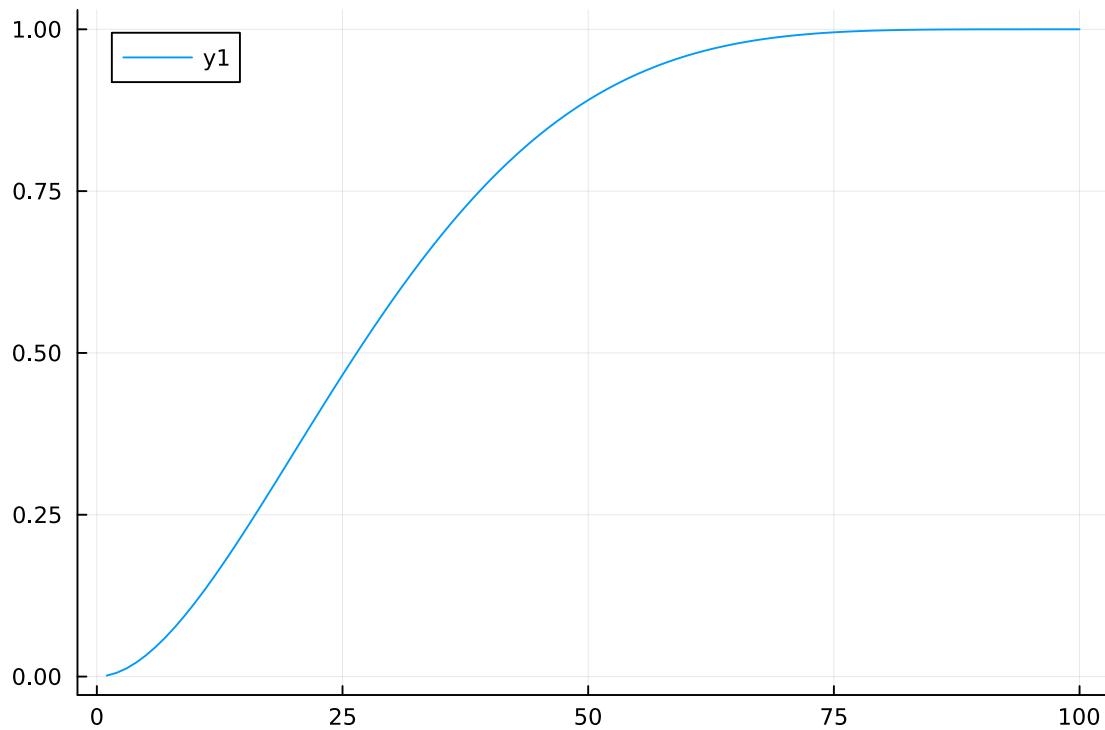
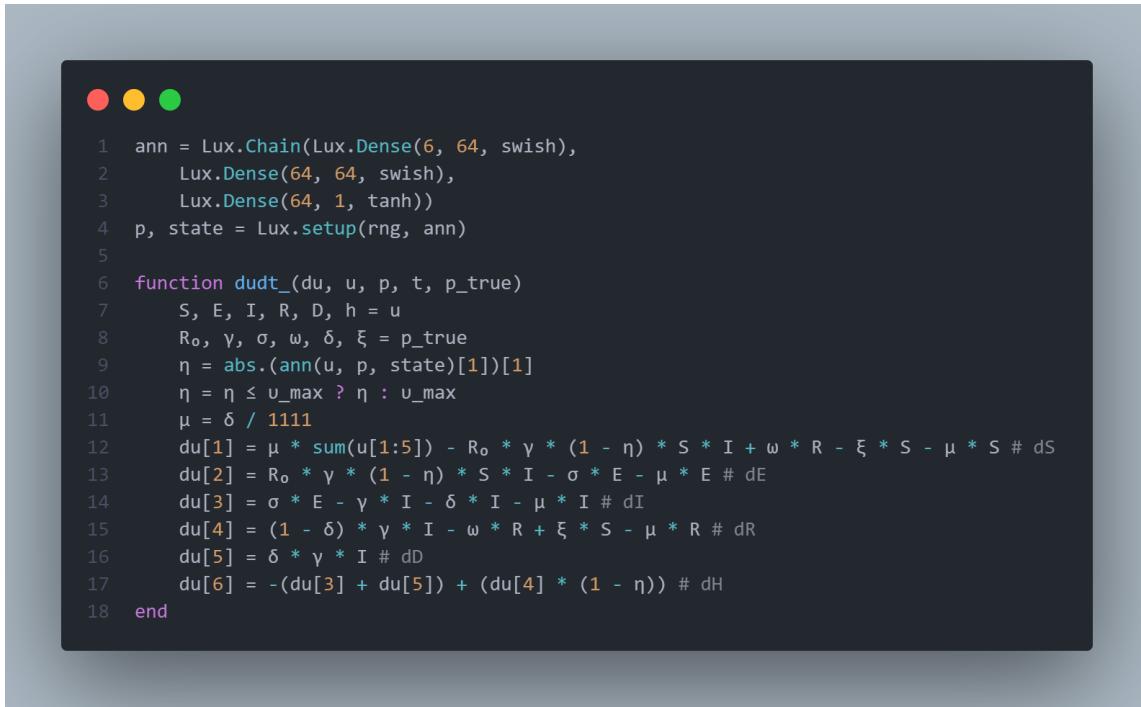


Figure 32: Funzione per il calcolo del limite superiore per le contromisure non farmaceutiche

Come evidenziato nella figura 32, la funzione rappresenta la "funzione di distribuzione cumulativa" per la distribuzione "beta" con parametri $\alpha = 2$ e $\beta = 5$. Questo tipo di funzione è stato scelto poiché approssima abbastanza bene il concetto di pericolo, soprattutto durante le prime fasi di una pandemia, quando il numero di infetti cresce rapidamente e poi tende a stabilizzarsi una volta superato il cinquanta percento di infetti.

Questa rappresentazione può anche essere correlata all'idea di un punto critico in una pandemia, oltre il quale le speranze di controllo diminuiscono drasticamente. È importante notare che questa funzione potrebbe evolversi nel tempo, riflettendo una sorta di adattamento alla pandemia.

Il controllore effettivo è principalmente strutturato come già accennato nella sezione relativa alle Neural ODE.



```

1 ann = Lux.Chain(Lux.Dense(6, 64, swish),
2                 Lux.Dense(64, 64, swish),
3                 Lux.Dense(64, 1, tanh))
4 p, state = Lux.setup(rng, ann)
5
6 function dudt_(du, u, p, t, p_true)
7     S, E, I, R, D, h = u
8     R₀, γ, σ, ω, δ, ξ = p_true
9     η = abs.(ann(u, p, state)[1])[1]
10    η = η ≤ u_max ? η : u_max
11    μ = δ / 1111
12    du[1] = μ * sum(u[1:5]) - R₀ * γ * (1 - η) * S * I + ω * R - ξ * S - μ * S # dS
13    du[2] = R₀ * γ * (1 - η) * S * I - σ * E - μ * E # dE
14    du[3] = σ * E - γ * I - δ * I - μ * I # dI
15    du[4] = (1 - δ) * γ * I - ω * R + ξ * S - μ * R # dR
16    du[5] = δ * γ * I # dD
17    du[6] = -(du[3] + du[5]) + (du[4] * (1 - η)) # dH
18 end

```

Figure 33: Definizione del controllore mediante Neural ODE

Una rete neurale è definita utilizzando il framework **Lux.jl** [59] e viene utilizzata esclusivamente per fare una stima del valore di controllo η durante la fase di addestramento, il quale sarà alla fine utilizzato come valore di controllo per il modello. Successivamente, viene definito il sistema di ODE che governa il fenomeno che si desidera analizzare, nello specifico, il sistema è un modello **SEIR** che tiene conto della perdita di immunità nel tempo.

Successivamente, il problema viene istanziato utilizzando l'interfaccia **ODEProblem**, a partire dalla quale il modello può iniziare ad essere addestrato. Prima di procedere con l'addestramento effettivo, vengono definite funzioni per la predizione, il calcolo della perdita del modello (loss) e una funzione di callback ausiliaria utile per monitorare l'andamento dell'apprendimento del modello.

```

● ○ ●
1 function predict(p)
2     _prob = remake(prob, u0 = ic, tspan = timeframe, p = p)
3     Array(solve(_prob,
4                 Tsit5(),
5                 saveat = ts,
6                 abstol = 1e-10,
7                 reltol = 1e-10,
8                 verbose = false))
9 end
10
11 function l(x)
12     loss_function === missing ? sum(abs2, x[3, :]) / sum(abs2, x[end, :]) :
13     loss_function
14 end
15
16 loss(p) = l(predict(p))
17
18 losses = Float64[]
19 callback = function (p, l; loss_step = loss_step)
20     push!(losses, l)
21     ## Exit early if not improving...
22     if length(losses) > 1 && (abs(losses[end - 1] - losses[end])) < eps()
23         return true
24     end
25     if length(losses) % loss_step == 0
26         @debug "Current loss after $(length(losses)) iterations: $(losses[end])"
27     end
28     return false
29 end

```

Figure 34: Definizione delle funzioni di supporto del controllore

Come mostrato nella figura 34, la funzione di callback svolge un doppio ruolo: da un lato, fornisce un supporto nella visualizzazione dei dati relativi all'addestramento, e dall'altro, implementa una forma di "early stopping" per evitare l'addestramento eccessivo una volta che il modello ha raggiunto un risultato ottimale. Questo approccio si basa sull'osservazione che il valore della loss tende a stabilizzarsi una volta che il modello ha raggiunto una buona soluzione.

La fase di addestramento effettiva del modello avviene attraverso le seguenti funzioni:



```

1 adtype = Optimization.AutoZygote()
2 optf = Optimization.OptimizationFunction((x, p) -> loss(x), adtype)
3 optprob = Optimization.OptimizationProblem(optf, ComponentVector{Float64}(p))
4 res1 = Optimization.solve(optprob, ADAM(0.01), callback = callback, maxiters = maxiters)
5 return abs.(first(ann(ic, res1.u, state)))[1]

```

Figure 35: Definizione delle funzioni di addestramento del controllore

Il ciclo di addestramento può essere idealmente suddiviso in due parti, ma in pratica, entrambe potrebbero non essere necessarie. La prima parte, come mostrato nella figura 35, impiega un ciclo di addestramento con l'uso dell'ottimizzatore **ADAM**. Se i risultati sono soddisfacenti, è possibile concludere l'addestramento. Tuttavia, è comune utilizzare due cicli di addestramento differenti per massimizzare le prestazioni. In particolare, si associano due cicli di addestramento con due ottimizzatori diversi al fine di sfruttare al massimo le peculiarità di ognuno. Questo approccio è utile per massimizzare il processo di ottimizzazione dei parametri. Solitamente, si utilizza un ottimizzatore per la maggior parte dell'addestramento, in quanto è in grado di trovare rapidamente un buon spazio di iperparametri. Successivamente, si passa a un secondo ottimizzatore, come ad esempio **BFGS**, che eccelle nel trovare rapidamente un minimo locale all'interno dello spazio dei parametri.

La mia scelta di non adottare questo approccio, ovvero di utilizzare un solo ottimizzatore, deriva dal fatto che l'uso di **ADAM** con un numero limitato di iterazioni, combinato con una tecnica per ridurre ulteriormente le iterazioni quando non si verifica un miglioramento significativo, consente di raggiungere un risultato ottimale in tempi brevi, senza la necessità di utilizzare un secondo ottimizzatore, il quale non apporterebbe ulteriori miglioramenti e comporterebbe solo un aumento delle risorse computazionali.



```

1  res1 = Optimization.solve(
2      optprob,
3      ADAM(),
4      callback=callback,
5      maxiters=400
6  )
7
8  optprob2 = remake(optprob, u0=res1.u)
9  res2 = Optimization.solve(
10     optprob2,
11     Optim.BFGS(initial_stepnorm=0.01),
12     callback=callback,
13     maxiters=100
14 )

```

Figure 36: Esempio di utilizzo congiunto di due ottimizzatori differenti in uno stesso ciclo di addestramento

Funzione di Attivazione della Rete Neurale

La scelta della funzione di attivazione all'interno delle reti neurali ha un notevole impatto sulle dinamiche di apprendimento. Attualmente, la funzione più comunemente utilizzata e di successo è la funzione **ReLU** (**R**ectified **L**inear **U**nity), definita come $f(x) = \max(0, x)$. Tuttavia, sono state proposte diverse alternative, ma nessuna di esse è riuscita a superare la popolarità della ReLU, principalmente a causa di risultati prestazionali variabili.

La funzione di attivazione che ho scelto di utilizzare è chiamata **Swish** ed è stata proposta dal *Google Brain Team* [69]. La funzione Swish è definita come $f(x) = x \cdot \text{sigmoid}(x)$. In esperimenti

condotti, è stato osservato che sostituire la funzione di attivazione ReLU con Swish ha portato a miglioramenti nelle prestazioni sui task di classificazione "top-1" su dataset come **ImageNet**, con guadagni fino al 0.9% utilizzando **NASNetA** e fino al 0.6% utilizzando **Inception-ResNet-v2**.

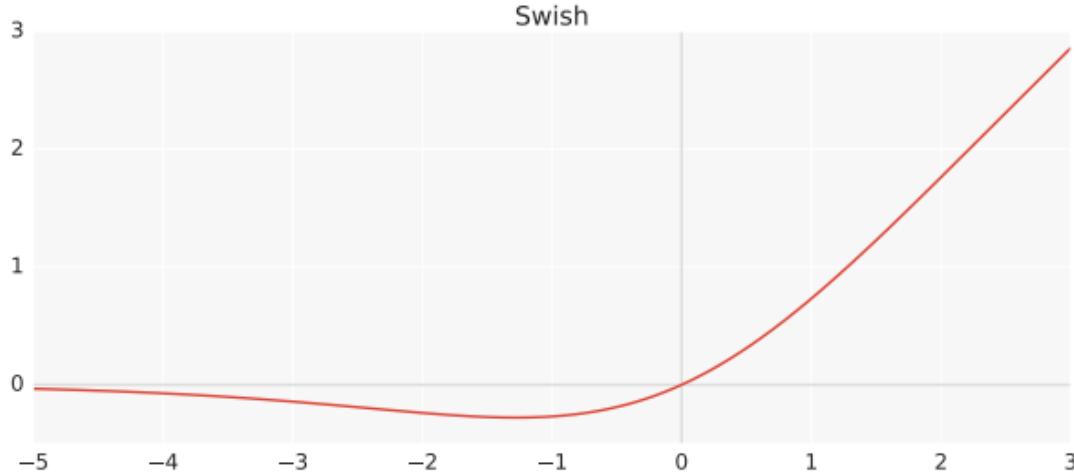


Figure 37: Funzione di attivazione Swish

<https://lazyprogrammer.me/wp-content/uploads/2017/10/Screen-Shot-2017-10-18-at-2.39.55-PM.png>

La peculiarità distintiva di Swish risiede nella sua semplicità e nella notevole somiglianza con la funzione di attivazione ReLU, il che la rende estremamente agevole da implementare come sua alternativa. Un'importante problematica associata all'utilizzo di ReLU è rappresentata dal fatto che il valore della sua derivata è pari a zero per la metà dei valori di input x .

È stato dimostrato, mediante l'impiego del dataset **MNIST**, che le prestazioni di Swish e ReLU sono sostanzialmente comparabili quando si utilizzano reti neurali con un numero di strati (layer) che si aggira intorno a 40. Tuttavia, emerge chiaramente che Swish supera in modo significativo ReLU in termini di prestazioni, specialmente quando il numero di strati si avvicina ai 40-50, situazione in cui l'ottimizzazione tende a diventare più complessa. Ciò suggerisce, ed è stato successivamente confermato, che in reti neurali molto profonde, Swish mostra prestazioni superiori nei test di accuratezza rispetto alla funzione ReLU. Tuttavia, è importante notare che entrambe le funzioni sperimentano un declino delle prestazioni all'aumentare delle dimensioni dei batch, ma in ogni caso, Swish ottiene risultati migliori.

Alla luce di questi dati e considerando la struttura relativamente poco profonda del modello, ho preso la decisione di utilizzare Swish come funzione di attivazione all'interno della rete neurale. È importante notare che questa scelta è attesa per contribuire positivamente alle prestazioni complessive, anche se ci si aspetta che il guadagno di accuratezza non sia eccessivamente sensibile, dato il contesto dell'applicazione illustrata nella figura 33.

4 Analisi di Sensibilità

L'analisi di sensitività è stata condotta sul modello SEIR, come illustrato in Figura 14. Questa analisi mirava a valutare come il modello reagisse a variazioni specifiche dei suoi parametri, le quali potrebbero condurre a comportamenti inattesi o peculiari.

Per effettuare l'analisi di sensitività, sono stati impiegati i metodi forniti dalla suite SciML.ai. Ciò ha permesso di identificare i parametri ai quali il modello è più sensibile.

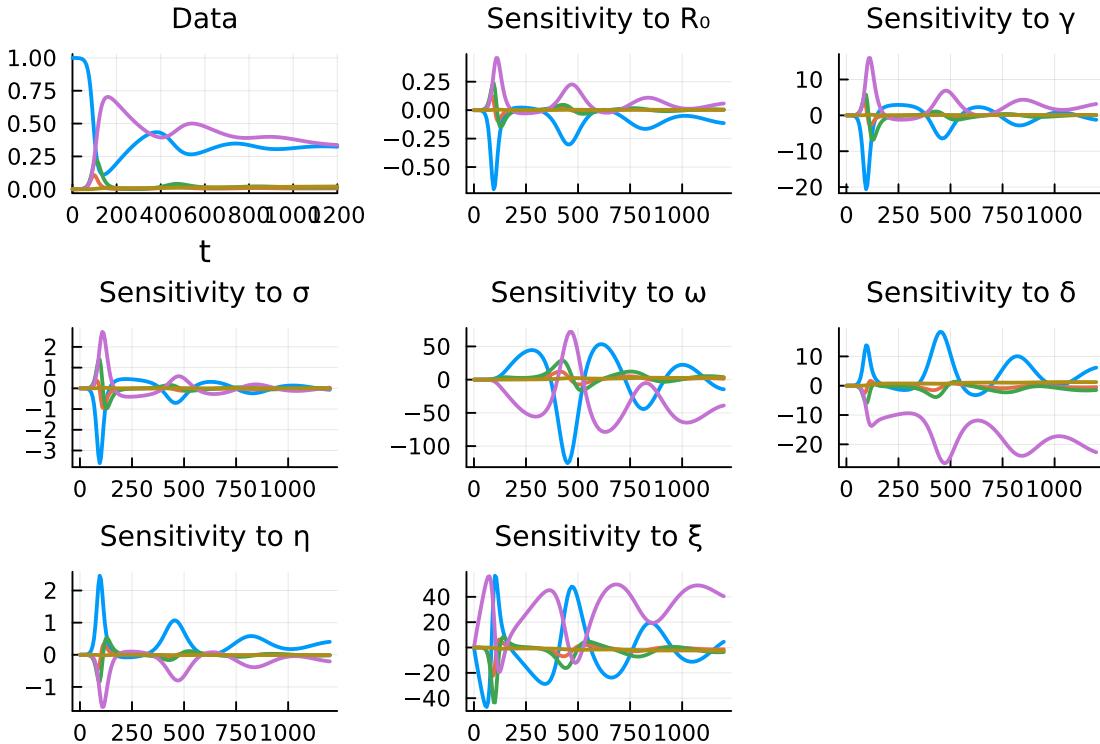


Figure 38: Grafico rappresentante l'analisi di sensitività del modello

Come evidenziato nella Figura 38, il modello dimostra una sensitività uniforme rispetto ai parametri R_0 , γ , e σ , il che è coerente, dato che essi influiscono sull'andamento dell'epidemia. La sensitività ai parametri ω e δ era attesa, ma è risultata più marcata di quanto previsto. La sensitività al parametro η , che influisce sulle contromisure, è inversamente correlata a quella del parametro R_0 , come previsto, poiché η è direttamente legato alle misure di contenimento. La sensitività al parametro ξ mostra un comportamento singolare, probabilmente dovuto alla sua correlazione con ω , entrambi legati al compartimento "R" del modello, che gestisce gli individui guariti e vaccinati.

Successivamente, è stata condotta un'analisi di sensitività sui parametri specifici del modello ad agenti utilizzando la funzione **paramscan** fornita dalla libreria **Agents.jl**. Sono stati selezionati parametri potenzialmente influenti sul comportamento del modello:



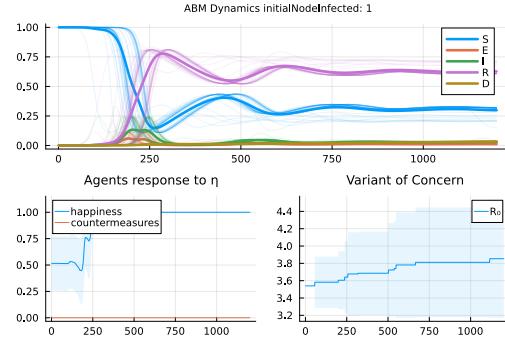
```
1 properties = Dict(
2     :maxTravelingRate => Base.collect(0.001:0.003:0.01),
3     :edgesCoverage => [:high, :medium, :low],
4     :numNodes => Base.collect(5:25:80),
5     :initialNodeInfected => Base.collect(1:3:10),
6   )
```

Figure 39: Parametri usati per l'analisi di sensitività del modello ad agente

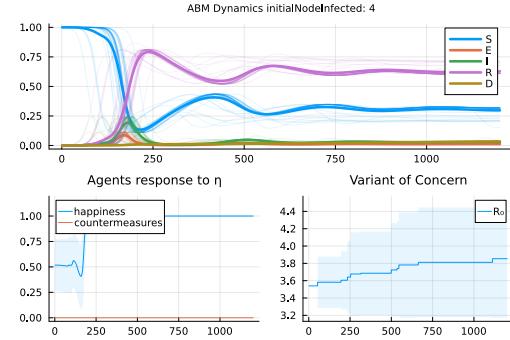
Questi parametri sono stati scelti in quanto potenziali agenti di cambiamenti significativi nel comportamento del modello. I parametri relativi al controllore e al vaccino sono stati esclusi, poiché è stato dimostrato che le loro variazioni possono comportare cambiamenti significativi nella simulazione.

4.1 Analisi del Comportamento in Base al Numero Iniziale di Nodi Infetti

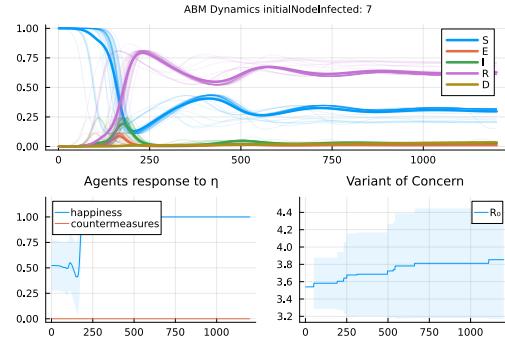
L'analisi ha esaminato come il numero iniziale di nodi infetti influenzi il comportamento del modello. Si è osservato che un numero maggiore di nodi infetti iniziali porta a un raggiungimento più rapido del picco di infetti, ma non ha comportato altri comportamenti significativi. Questo risultato era atteso e coerente con le dinamiche epidemiologiche.



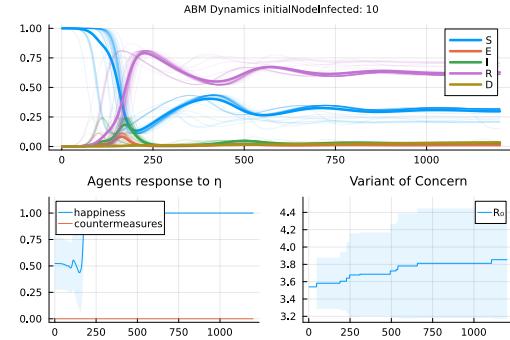
(a) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 1



(b) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 4



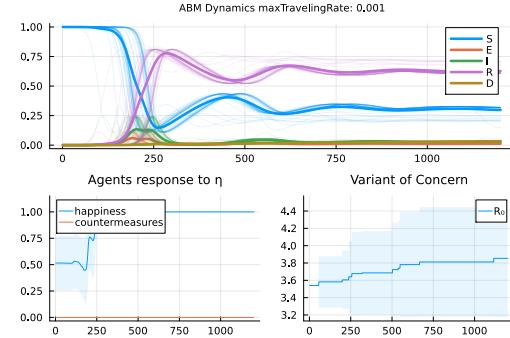
(c) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 7



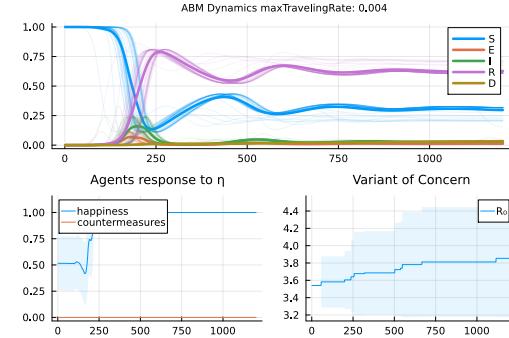
(d) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 10

4.2 Analisi del Comportamento in Base al Tasso di Migrazione

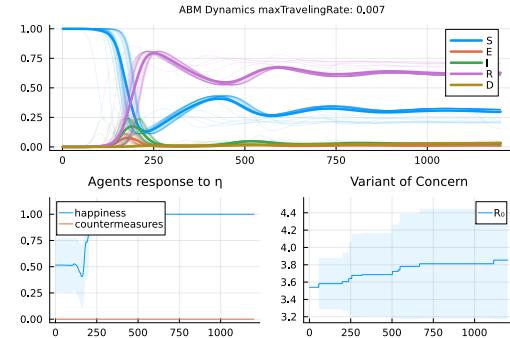
L'analisi ha esaminato come il tasso di migrazione influenzi il comportamento del modello. Si è notato che un tasso di migrazione più alto ha portato a un picco di infetti più alto in un periodo di tempo più breve. Questo risultato è coerente con l'effetto della mobilità sulla diffusione delle malattie infettive.



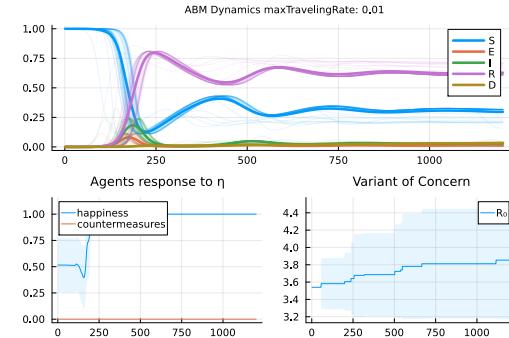
(a) Grafico per la comparazione sul valore di migrazione. Valore di 0.001



(b) Grafico per la comparazione sul valore di migrazione. Valore di 0.004



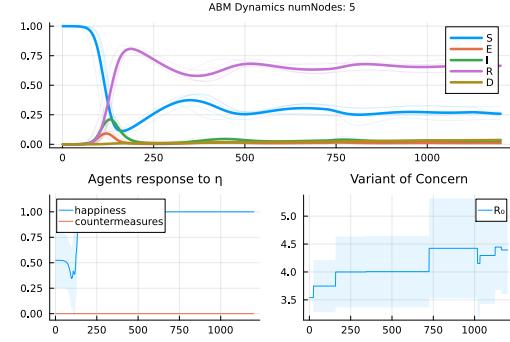
(c) Grafico per la comparazione sul valore di migrazione. Valore di 0.007



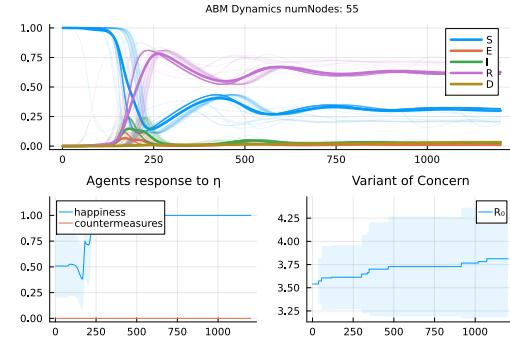
(d) Grafico per la comparazione sul valore di migrazione. Valore di 0.01

4.3 Analisi del Comportamento in Base al Numero di Nodi della Rete

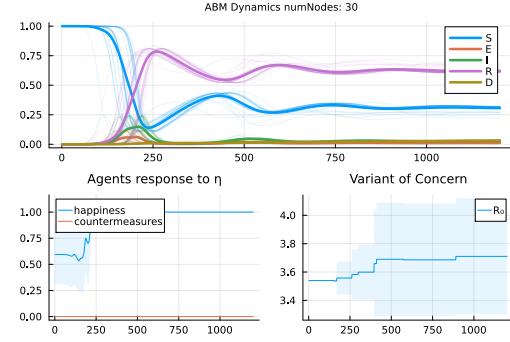
L'analisi ha esaminato come il numero di nodi della rete influenzi il comportamento del modello. Si è osservato che un numero maggiore di nodi ha portato a una diffusione più lenta e a picchi meno ripidi ma più prolungati nel tempo. Questo risultato è influenzato dalla topologia delle connessioni e dalla posizione del focolaio iniziale.



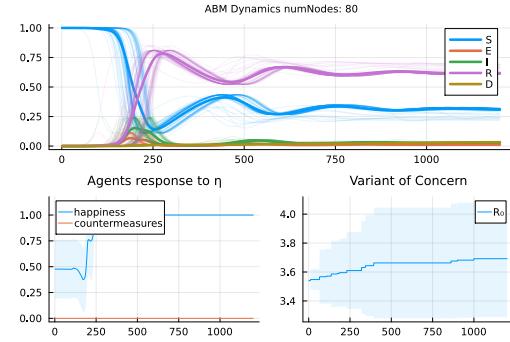
(a) Grafico per la comparazione sul numero di nodi della rete. Numero nodi 5



(c) Grafico per la comparazione sul numero di nodi della rete. Numero nodi 55



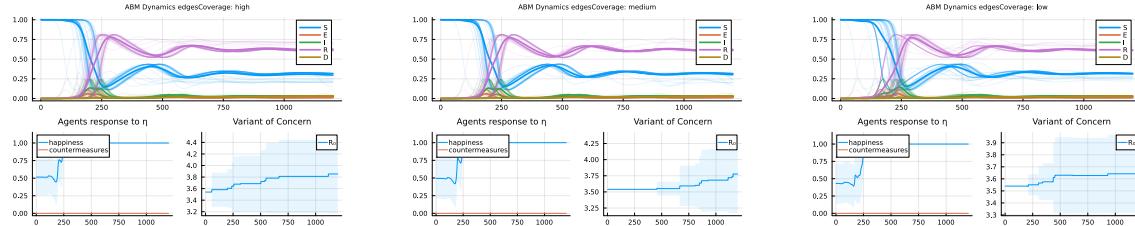
(b) Grafico per la comparazione sul numero di nodi della rete. Numero nodi 30



(d) Grafico per la comparazione sul numero di nodi della rete. Numero nodi 80

4.4 Analisi del Comportamento in Base alla Copertura della Rete

L'analisi ha esaminato come la copertura della rete, rappresentata dal numero di archi, influenzi il comportamento del modello. Si è notato che una copertura bassa ha comportato un rallentamento della diffusione del virus, simile all'effetto di un lockdown. Questo risultato è coerente con l'importanza della mobilità nella diffusione delle malattie infettive e con il ruolo delle misure di contenimento.



(a) Grafico per la comparazione sulla copertura della rete. Copertura alta

(b) Grafico per la comparazione sulla copertura della rete. Copertura media

(c) Grafico per la comparazione sulla copertura della rete. Copertura bassa

Inoltre, è stato osservato che il numero di varianti del virus (VOC) tende ad aumentare con il numero di nodi nella rete, suggerendo che più individui possono portare a un aumento delle mutazioni, ma la presenza di una bassa copertura può limitare la loro diffusione.

In sintesi, l'analisi ha evidenziato come vari parametri e condizioni possano influenzare il comportamento del modello epidemiologico, fornendo importanti informazioni per la comprensione e la gestione delle epidemie.

5 Risultati Ottenuti

I risultati ottenuti, che verranno dettagliatamente presentati nelle sezioni successive, possono essere riassunti in modo generale come risultati positivi, sia in termini di performance che di politiche applicate. In particolare, è possibile notare come il tempo generale della simulazione dipenda principalmente dall'applicazione del controllore e quindi dal calcolo della Neural ODE e tutte le operazioni correlate.

Table 1: Tempo Impiegato (HH:MM:SS)

Tipologia di Test				
	Nessun Intervento	Intervento Non Farmaceutico	Intervento Farmaceutico	Intervento Farmaceutico e Non
Singlerun (50 PdI)	0:00:15	0:06:52	0:00:13	0:02:52
Ensemblerun (5 ripetizioni, 50 PdI)	0:00:34	0:06:22	0:00:24	0:02:55

Table 2: Tempo Impiegato (HH:MM:SS)

Tipologia di Test				
	Nodi Infetti Iniziali Variabili	Valore di Migrazione Variabile	Nodi della Rete Variabile	Copertura dei Nodi Variabile
Paramscan	0:00:13	0:00:21	0:00:32	0:00:12

Questi lunghi tempi di calcolo derivano dalla computazione delle contromisure non farmaceutiche applicate alla popolazione. Le contromisure farmaceutiche, se applicate con il giusto tempismo, oltre a contribuire notevolmente alla riduzione della diffusione della pandemia (vedi Figura 30a, Figura 30b, Figura 30c, Figura 30d), consentono una significativa riduzione dei tempi di calcolo.

5.1 Nessun Intervento

Il seguente grafico (Figura 44) mostra l'andamento delle curve del modello quando questo viene eseguito senza alcuna tipologia di intervento. Questo andamento è rappresentato in modo cumulativo rispetto all'andamento dei singoli agenti, i quali possono mostrare comportamenti differenti tra loro.

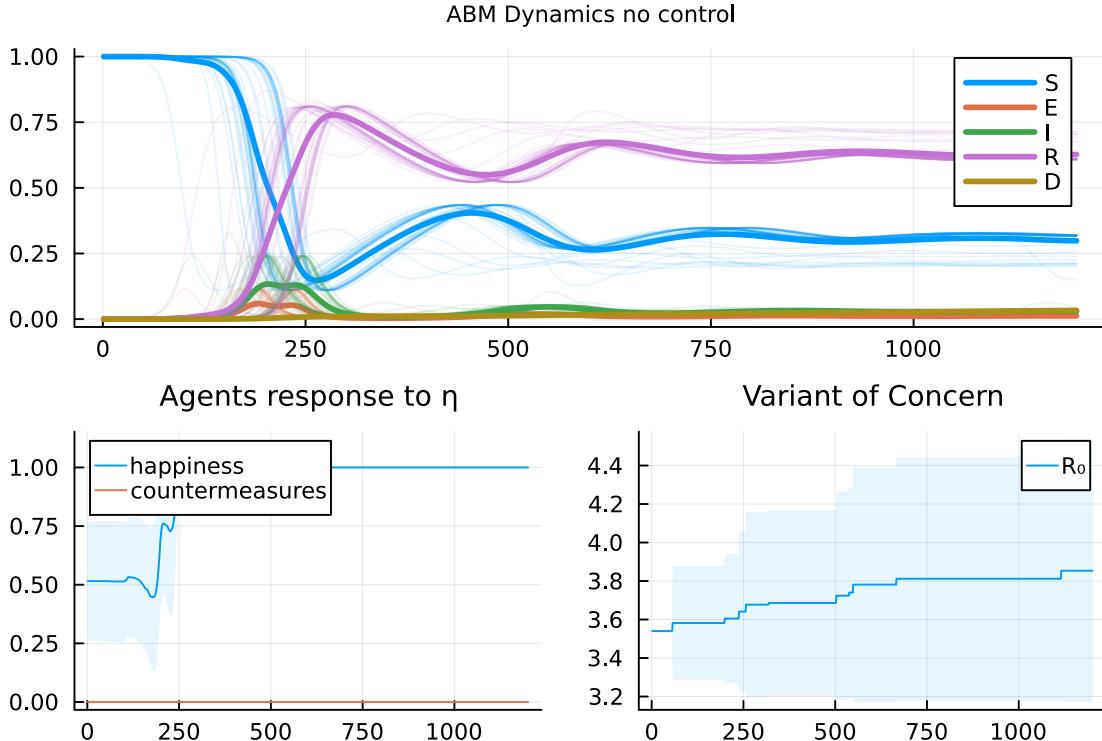


Figure 44: Grafico cumulativo del modello senza alcun tipo di intervento

Complessivamente, l'andamento del modello è simile a quello standard di un modello di tipo SEIR, con alcune variazioni dovute ai fattori di stocasticità intrinseci del modello. Il grafico mostra le traiettorie più comuni delle curve cumulative del modello, mettendo in evidenza i percorsi più frequenti.

Come è possibile notare, il numero di individui suscettibili diminuisce drasticamente a causa della rapida diffusione esponenziale del virus. Questo è accompagnato da una crescita altrettanto rapida del numero di individui guariti (recovered). Tuttavia, a causa della definizione delle condizioni per i guariti, questi individui non sono immuni alle varianti del virus, il che permette di modellare una possibile ciclicità dell'epidemia dovuta alla perdita di immunità della popolazione. Queste proprietà contribuiscono all'andamento ciclico delle curve. Inoltre, si nota che la curva associata all'andamento degli individui nella classe D ha una crescita lineare, sebbene non molto evidente.

La curva associata alla variabile di "happiness" del modello, utilizzata per bilanciare la severità

delle misure di controllo per evitare un ciclo insostenibile, mostra un comportamento piuttosto peculiare. Questo è principalmente dovuto alla definizione della funzione di controllo della felicità (vedi Figura 27). Inoltre, la curva tende ad oscillare in modo ciclico a causa della definizione delle misure di controllo.

In generale, il modello senza interventi mostra un'epidemia in rapida crescita seguita da periodi ciclici di riduzione della diffusione del virus.

5.2 Intervento Non Farmaceutico

Nel seguente grafico (Figura 45) è rappresentato l'andamento delle curve quando viene applicato un intervento non farmaceutico, come il distanziamento sociale, l'uso di maschere, ecc. L'obiettivo di questo tipo di intervento è quello di rallentare la diffusione del virus senza utilizzare farmaci o vaccini.

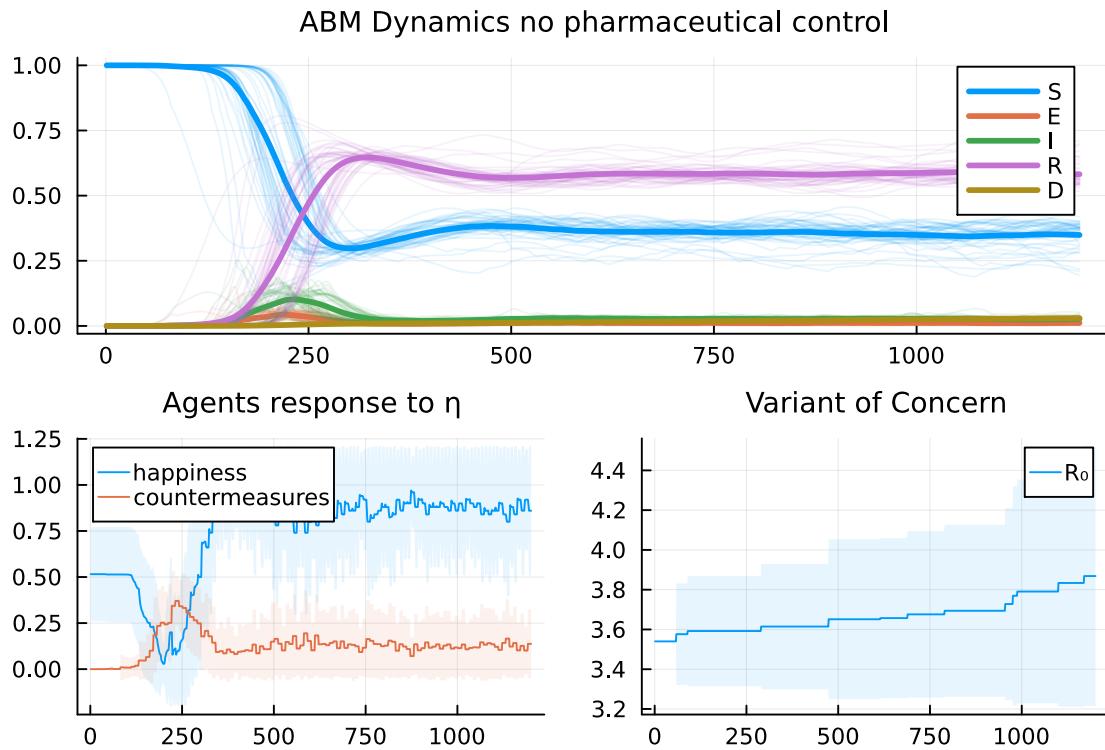


Figure 45: Grafico cumulativo del modello con intervento non farmaceutico

Come si può vedere dal grafico, l'intervento non farmaceutico ha un impatto significativo sulla diffusione del virus. In particolare, il numero di individui suscettibili rimane più stabile nel tempo rispetto al caso senza interventi, e il picco dell'epidemia è notevolmente ridotto. Questo è dovuto all'efficacia delle misure di distanziamento sociale e al fatto che esse riducono il tasso di contatto tra gli individui infetti e suscettibili. Tuttavia, il numero di individui guariti è ancora ciclico a causa della perdita di immunità, e la curva di felicità mostra comunque oscillazioni dovute alle misure di

controllo cicliche.

In generale, l'intervento non farmaceutico riesce a contenere l'epidemia e a ridurne l'entità, ma non riesce a eliminarla completamente.

5.3 Intervento Farmaceutico

Il seguente grafico (Figura 46) rappresenta l'andamento delle curve quando viene applicato un intervento farmaceutico, come la somministrazione di vaccini o farmaci antivirali. L'obiettivo di questo tipo di intervento è quello di ridurre significativamente la diffusione del virus e, idealmente, di eliminare l'epidemia.

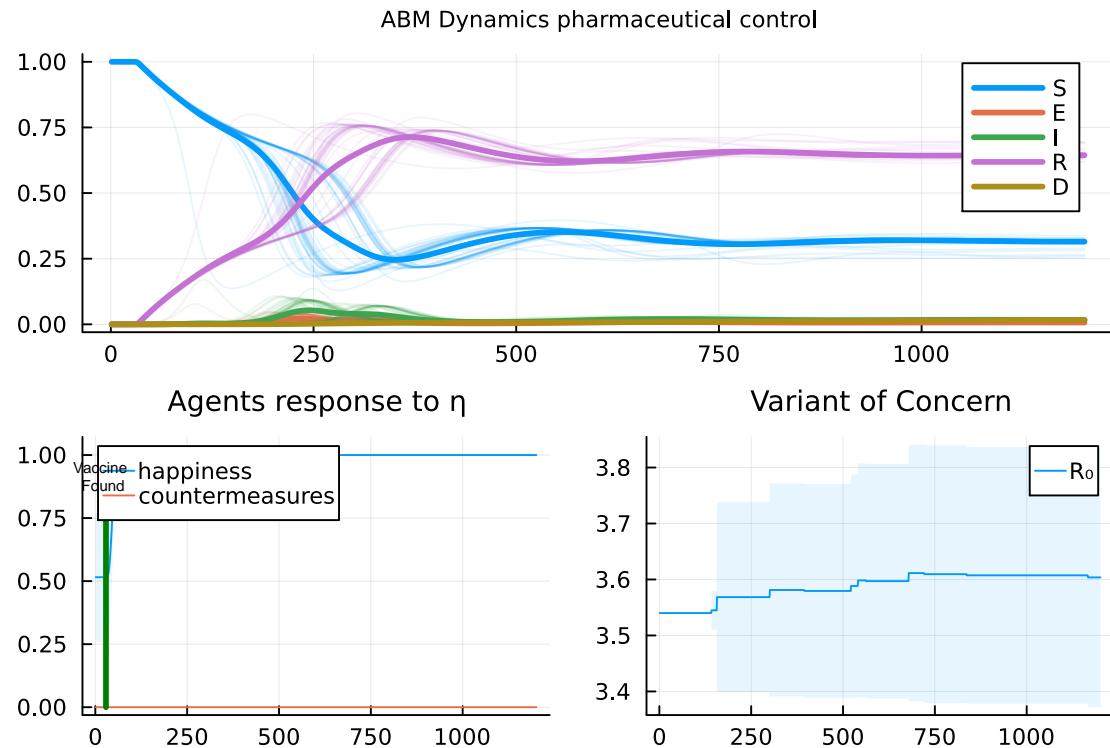


Figure 46: Grafico cumulativo del modello con intervento farmaceutico

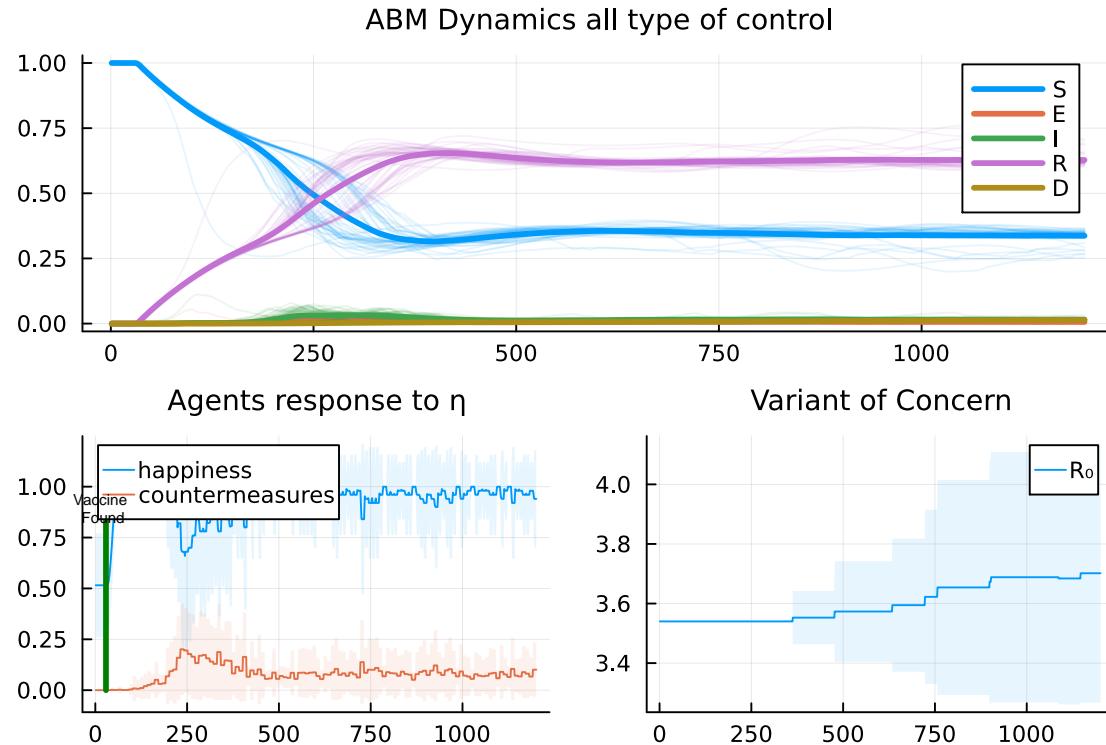
Come si può notare, l'intervento farmaceutico ha un impatto molto positivo sulla diffusione del virus. Il numero di individui suscettibili rimane stabile nel tempo, e l'epidemia viene rapidamente contenuta. Questo è dovuto alla somministrazione di farmaci o vaccini, che riducono la capacità di trasmissione del virus e proteggono gli individui suscettibili. Inoltre, il numero di individui guariti non mostra più l'andamento ciclico, poiché i soggetti guariti attraverso l'intervento farmaceutico acquisiscono immunità duratura.

La curva di felicità mostra ancora alcune oscillazioni dovute alle misure di controllo cicliche, ma queste oscillazioni sono meno pronunciate rispetto ai casi precedenti.

In generale, l'intervento farmaceutico è estremamente efficace nel contenere e ridurre l'epidemia.

5.4 Intervento Farmaceutico e Non Farmaceutico

Il seguente grafico (Figura 47) rappresenta l'andamento delle curve quando vengono applicati sia un intervento farmaceutico che un intervento non farmaceutico. L'obiettivo di questa combinazione di interventi è quello di massimizzare la riduzione della diffusione del virus e di contenere l'epidemia in modo ottimale.



Come mostrato nel grafico, l'effetto combinato di intervento farmaceutico e non farmaceutico è estremamente efficace nel contenere l'epidemia. Il numero di individui suscettibili rimane stabile nel tempo, e l'epidemia viene rapidamente ridotta a livelli molto bassi. La curva di individui guariti mostra un aumento costante nel tempo, poiché l'intervento farmaceutico garantisce immunità duratura.

La curva di felicità mostra ancora alcune oscillazioni dovute alle misure di controllo cicliche, ma queste oscillazioni sono molto meno pronunciate rispetto ai casi precedenti.

In generale, la combinazione di intervento farmaceutico e non farmaceutico è altamente efficace nel contenere e ridurre l'epidemia in modo ottimale.

6 Conclusione

In conclusione, i risultati ottenuti dimostrano che l'applicazione di misure di controllo, sia farmaceutiche che non farmaceutiche, è estremamente efficace nel contenere e ridurre la diffusione del virus. L'intervento farmaceutico, in particolare, ha dimostrato di essere altamente efficace nel contenere l'epidemia e prevenire il ciclo insostenibile di infezioni. Tuttavia, l'intervento farmaceutico da solo potrebbe non essere sufficiente a eliminare completamente l'epidemia, mentre la combinazione di intervento farmaceutico e non farmaceutico offre una protezione ottimale.

I tempi di calcolo delle simulazioni sono significativamente influenzati dall'applicazione delle misure di controllo non farmaceutiche, ma questo è un compromesso necessario per ottenere risultati accurati e realistici. Inoltre, l'uso di misure di controllo cicliche può portare a oscillazioni nelle curve di felicità, ma queste oscillazioni sono gestibili.

In generale, il modello di simulazione dell'epidemia su una rete sociale offre una piattaforma utile per studiare gli effetti delle misure di controllo e per sviluppare strategie ottimali per la gestione delle epidemie.

7 Sviluppi Futuri

7.1 Perfezionamento del Controllore

Un possibile sviluppo futuro di rilevanza è il perfezionamento del controllore, mirando a migliorarne sia le capacità di individuazione che l'applicazione delle policy. Inoltre, si potrebbe cercare di rendere le policy generate dal controllore più comprensibili per gli operatori umani, in modo da agevolare la loro traduzione e comprensione. Questa evoluzione apre la porta a un'approfondita analisi di causalità, che è un elemento di estrema importanza, come precedentemente discusso. In particolare, sarà necessario esplorare la relazione tra l'applicazione di specifiche contromisure e il loro impatto sulla riduzione di una pandemia.

Tale approccio si presenta come una sfida complessa ma altamente promettente, poiché potrebbe rappresentare un notevole avanzamento tecnologico.

7.2 Miglioramento della Funzione di Happiness

Un'altra importante area di sviluppo futuro riguarda la funzione di happiness, che gioca un ruolo cruciale nel controllo del controllore. Attualmente, questa funzione è utilizzata in modo semplice per evitare l'attivazione di contromisure insostenibili nella vita reale. Tuttavia, è possibile migliorarla ulteriormente, tenendo conto dell'effettivo impatto che ciascuna tipologia di contromisura può avere sulla vita delle persone, sia nel breve che nel lungo periodo. Questo miglioramento potrebbe rappresentare una pietra miliare per l'accuratezza delle policy sviluppate, fornendo un quadro di valutazione più preciso.

Tuttavia, anche questa area di sviluppo richiede un'analisi attenta della causalità degli eventi, in particolare per stimare l'effetto delle contromisure sull'umore complessivo di una popolazione. L'umore può variare a causa di numerosi fattori, comprese le contromisure e gli stimoli esterni, e l'analisi richiederà il supporto di diversi benchmark con modelli di simulazione diversificati.

7.3 Perfezionamento Generale del Modello

Il modello attualmente implementato si basa su assunzioni relativamente realistiche, derivanti da osservazioni empiriche. Tuttavia, il modello potrebbe beneficiare di un maggiore grado di generalizzazione e flessibilità per adattarsi a una varietà di problemi. Ciò comporterebbe la necessità di superare alcune rigidità strutturali attuali che ne limitano l'utilità in contesti diversi.

7.4 Ottimizzazione Generale

Un ulteriore sviluppo importante riguarda l'ottimizzazione generale del codice. Attualmente, il codice potrebbe essere ulteriormente migliorato in termini di efficienza, scalabilità e ottimizzazione delle performance. Questo sforzo mirerebbe a rendere il codice più performante e adattabile alle esigenze future.

References

- [1] Sameera Abar, Georgios K. Theodoropoulos, Pierre Lemarinier, and Gregory M.P. O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33, 2017.
- [2] Mohammad Abavisani, Karim Rahimian, Bahar Mahdavi, Samaneh Tokhanbigli, Mahsa Mollapour Siasakht, Amin Farhadi, Mansoor Kodori, Mohammadamin Mahmanzar, and Zahra Meshkat. Mutations in sars-cov-2 structural proteins: a global analysis. *Virology Journal*, 19(1):220, Dec 2022.
- [3] Linda J. S. Allen. *An Introduction to Stochastic Epidemic Models*, pages 81–130. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [4] Naomi Altman and Martin Krzywinski. Association, correlation and causation. *Nature Methods*, 12(10):899–900, Oct 2015.
- [5] Justin Angevare, Zeny Feng, and Rob Deardon. Pathogen.jl: Infectious disease transmission network modeling with julia. *Journal of Statistical Software*, 104(4):1–30, 2022.
- [6] Azam Asanjarani, Aminath Shausan, Keng Chew, Thomas Graham, Shane G. Henderson, Hermanus M. Jansen, Kirsty R. Short, Peter G. Taylor, Aapeli Vuorinen, Yuvraj Yadav, Ilze Ziedins, and Yoni Nazarathy. Emulation of epidemics via bluetooth-based virtual safe virus spread: Experimental setup, software, and data. *PLOS Digital Health*, 1(12):1–23, 12 2022.
- [7] Keith R. Bissett, Jose Cadena, Maleq Khan, and Chris J. Kuhlman. Agent-based computational epidemiological modeling. *Journal of the Indian Institute of Science*, 101(3):303–327, Jul 2021.
- [8] Ottar N. Bjørnstad, Katriona Shea, Martin Krzywinski, and Naomi Altman. The seirs model for infectious disease dynamics. *Nature Methods*, 17(6):557–558, Jun 2020.
- [9] Pierre-Alexandre Bliman and Michel Duprez. How best can finite-time social distancing reduce epidemic final size? *Journal of Theoretical Biology*, 511:110557, 2021.
- [10] Matthew H Bonds, Donald C Keenan, Pejman Rohani, and Jeffrey D Sachs. Poverty trap formed by the ecology of infectious diseases. *Proc Biol Sci*, 277(1685):1185–1192, December 2009.
- [11] Fred Brauer. *Compartmental Models in Epidemiology*, pages 19–79. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.
- [12] Tom Britton and Lasse Leskelä. Optimal intervention strategies for minimizing total incidence during an epidemic. *SIAM Journal on Applied Mathematics*, 83(2):354–373, mar 2023.
- [13] C. G. BROYDEN. The Convergence of a Class of Double-rank Minimization Algorithms 1. General Considerations. *IMA Journal of Applied Mathematics*, 6(1):76–90, 03 1970.
- [14] Steven L. Brunton and J. Nathan Kutz. *7 Data-driven methods for reduced-order modeling*, pages 307–344. De Gruyter, Berlin, Boston, 2021.

- [15] Lucas Böttcher and Thomas Asikis. Near-optimal control of dynamical systems with neural ordinary differential equations. *Machine Learning: Science and Technology*, 3(4):045004, oct 2022.
- [16] Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations, 2019.
- [17] P. Ciunkiewicz, W. Brooke, M. Rogers, and S. Yanushkevich. Agent-based epidemiological modeling of covid-19 in localized environments. *Computers in Biology and Medicine*, 144:105396, 2022.
- [18] Fernando Córdova-Lepe and Katia Vogt-Geisse. Adding a reaction-restoration type transmission rate dynamic-law to the basic seir covid-19 model. *PLOS ONE*, 17(6):1–21, 06 2022.
- [19] Raj Dandekar, Karen Chung, Vaibhav Dixit, Mohamed Tarek, Aslan Garcia-Valadez, Krishna Vishal Vemula, and Chris Rackauckas. Bayesian neural ordinary differential equations, 2022.
- [20] Raj Dandekar, Shane G. Henderson, Hermanus M. Jansen, Joshua McDonald, Sarat Moka, Yoni Nazarathy, Christopher Rackauckas, Peter G. Taylor, and Aapeli Vuorinen. Safe blues: The case for virtual safe virus spread in the long-term fight against epidemics. *Patterns*, 2(3):100220, 2021.
- [21] George Datseris, Ali R. Vahdati, and Timothy C. DuBois. Agents.jl: a performant and feature-full agent-based modeling software of minimal code complexity. *SIMULATION*, 0(0):003754972110688, January 2022.
- [22] Xiaoye Ding, Shenyang Huang, Abby Leung, and Reihaneh Rabbany. Incorporating dynamic flight network in seir to model mobility between populations. *Applied Network Science*, 6(1):42, Jun 2021.
- [23] Vaibhav Kumar Dixit and Christopher Rackauckas. Optimization.jl: A unified optimization package, March 2023.
- [24] Bernhard Ebbinghaus, Lukas Lehner, and Elias Naumann. Welfare state support during the covid-19 pandemic: Change and continuity in public attitudes towards social policies in germany. *European Policy Analysis*, 8(3):297–311, 2022.
- [25] Abdulrahman M El-Sayed, Peter Scarborough, Lars Seemann, and Sandro Galea. Social network analysis and agent-based modeling in social epidemiology. *Epidemiol Perspect Innov*, 9(1):1, February 2012.
- [26] James Fairbanks, Mathieu Besançon, Schöll Simon, Júlio Hoffiman, Nick Eubank, and Stefan Karpinski. Juliagraphs/graphs.jl: an optimized graphs package for the julia programming language, 2021.
- [27] Peter G. Fennell, Sergey Melnik, and James P. Gleeson. Limitations of discrete-time approaches to continuous-time contagion dynamics. *Phys. Rev. E*, 94:052125, Nov 2016.
- [28] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 01 1970.

- [29] Mathilde Frérot, Annick Lefebvre, Simon Aho, Patrick Callier, Karine Astruc, and Ludwig Serge Aho Glélé. What is epidemiology? changing definitions of epidemiology 1978-2017. *PLOS ONE*, 13(12):1–27, 12 2018.
- [30] Sandro Galea, Matthew Riddle, and George A Kaplan. Causal thinking and complex system approaches in epidemiology. *Int J Epidemiol*, 39(1):97–106, October 2009.
- [31] David J Gardner, Daniel R Reynolds, Carol S Woodward, and Cody J Balos. Enabling new flexibility in the SUNDIALS suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 2022.
- [32] Giulia Giordano, Franco Blanchini, Raffaele Bruno, Patrizio Colaneri, Alessandro Di Filippo, Angela Di Matteo, and Marta Colaneri. Modelling the covid-19 epidemic and implementation of population-wide interventions in italy. *Nature Medicine*, 26(6):855–860, Jun 2020.
- [33] Alberto Godio, Francesca Pace, and Andrea Vergnano. Seir modeling of the italian epidemic of sars-cov-2 using computational swarm intelligence. *International Journal of Environmental Research and Public Health*, 17(10), 2020.
- [34] Donald Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.
- [35] Shashi Gowda, Yingbo Ma, Valentin Churavy, Alan Edelman, and Christopher Rackauckas. Sparsity programming: Automated sparsity-aware optimizations in differentiable programming. 2019.
- [36] Elizabeth R. Groff, Shane D. Johnson, and Amy Thornton. State of the art in agent-based modeling of urban crime: An overview. *Journal of Quantitative Criminology*, 35(1):155–193, Mar 2019.
- [37] World Bank Group. Wdr 2022 chapter 1. introduction, Mar 2023.
- [38] Thomas Hale, Noam Angrist, Rafael Goldszmidt, Beatriz Kira, Anna Petherick, Toby Phillips, Samuel Webster, Emily Cameron-Blake, Laura Hallas, Saptarshi Majumdar, and Helen Tatlow. A global panel database of pandemic policies (oxford covid-19 government response tracker). *Nature Human Behaviour*, 5(4):529–538, Apr 2021.
- [39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [40] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. *ACM Transactions on Mathematical Software (TOMS)*, 31(3):363–396, 2005.
- [41] Michael Innes, Elliot Saba, Keno Fischer, Dhairya Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.
- [42] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018.
- [43] Mike Innes, Alan Edelman, Keno Fischer, Chris Rackauckas, Elliot Saba, Viral B Shah, and Will Tebbutt. A differentiable programming system to bridge machine learning and scientific computing, 2019.

- [44] Masoud Jalayer, Carlotta Orsenigo, and Carlo Vercellis. Cov-abm: A stochastic discrete-event agent-based framework to simulate spatiotemporal dynamics of covid-19, 2020.
- [45] Julius Martensen, Christopher Rackauckas, et al. Datadrivendiffeq.jl, July 2021.
- [46] Cliff C. Kerr, Robyn M. Stuart, Dina Mistry, Romesh G. Abeysuriya, Katherine Rosenfeld, Gregory R. Hart, Rafael C. Núñez, Jamie A. Cohen, Prashanth Selvaraj, Brittany Hagedorn, Lauren George, Michał Jastrzębski, Amanda S. Izzo, Greer Fowler, Anna Palmer, Dominic Delport, Nick Scott, Sherrie L. Kelly, Caroline S. Bennette, Bradley G. Wagner, Stewart T. Chang, Assaf P. Oron, Edward A. Wenger, Jasmina Panovska-Griffiths, Michael Famulare, and Daniel J. Klein. Covasim: An agent-based model of covid-19 dynamics and interventions. *PLOS Computational Biology*, 17(7):1–32, 07 2021.
- [47] Suyong Kim, Weiqi Ji, Sili Deng, Yingbo Ma, and Christopher Rackauckas. Stiff neural ordinary differential equations. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 31(9):093122, sep 2021.
- [48] Garyfallos Konstantoudis, Dominic Schuhmacher, Håvard Rue, and Ben D Spycher. Discrete versus continuous domain models for disease mapping. *Spatial and Spatio-temporal Epidemiology*, 32:100319, 2020.
- [49] James Ladyman, James Lambert, and Karoline Wiesner. What is a complex system? *European Journal for Philosophy of Science*, 3(1):33–67, Jan 2013.
- [50] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. Jump 1.0: Recent improvements to a modeling language for mathematical optimization. *Mathematical Programming Computation*, 2023.
- [51] Yingbo Ma, Vaibhav Dixit, Michael J Innes, Xingjian Guo, and Chris Rackauckas. A comparison of automatic differentiation and continuous sensitivity analysis for derivatives of differential equation solutions. In *2021 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–9, 2021.
- [52] Peter V. Markov, Mahan Ghafari, Martin Beer, Katrina Lythgoe, Peter Simmonds, Nikolaos I. Stilianakis, and Aris Katzourakis. The evolution of sars-cov-2. *Nature Reviews Microbiology*, 21(6):361–379, Jun 2023.
- [53] Sadegh Marzban, Renji Han, Nóra Juhász, and Gergely Röst. A hybrid PDE-ABM model for viral dynamics with application to SARS-CoV-2 and influenza. *R Soc Open Sci*, 8(11):210787, November 2021.
- [54] Samuel Mwalili, Mark Kimathi, Viona Ojiambo, Duncan Gathungu, and Rachel Mbogo. Seir model for covid-19 dynamics incorporating the environment and social distancing. *BMC Research Notes*, 13(1):352, Jul 2020.
- [55] John T Nardini, Ruth E Baker, Matthew J Simpson, and Kevin B Flores. Learning differential equation models from stochastic agent-based model simulations. *J R Soc Interface*, 18(176):20200987, March 2021.
- [56] Hiroshi Nishiura. Correcting the actual reproduction number: a simple method to estimate $r(0)$ from early epidemic growth data. *Int J Environ Res Public Health*, 7(1):291–302, January 2010.

- [57] Aleksandar Novakovic and Adele H. Marshall. The cp-abm approach for modelling covid-19 infection dynamics and quantifying the effects of non-pharmaceutical interventions. *Pattern Recognition*, 130:108790, 2022.
- [58] World Health Organization. Who coronavirus (covid-19) dashboard.
- [59] Avik Pal. Lux: Explicit Parameterization of Deep Neural Networks in Julia, 2023. If you use this software, please cite it as below.
- [60] M Parascandola and D L Weed. Causation in epidemiology. *J Epidemiol Community Health*, 55(12):905–912, December 2001.
- [61] Sunny Prakash Prajapati, Rahul Bhaumik, and Tarun Kumar. An intelligent abm-based framework for developing pandemic-resilient urban spaces in post-covid smart cities. *Procedia Computer Science*, 218:2299–2308, 2023. International Conference on Machine Learning and Data Engineering.
- [62] C. Rackauckas. A comparison between differential equation solver suites in matlab, r, julia, python, c, mathematica, maple, and fortran. *The Winnower*.
- [63] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl - a julia library for neural differential equations, 2019.
- [64] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl-a julia library for neural differential equations. *arXiv preprint arXiv:1902.02376*, 2019.
- [65] Christopher Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl - A julia library for neural differential equations. *CoRR*, abs/1902.02376, 2019.
- [66] Christopher Rackauckas, Yingbo Ma, Julius Martensen, Collin Warner, Kirill Zubov, Rohit Supekar, Dominic Skinner, and Ali Ramadhan. Universal differential equations for scientific machine learning. *arXiv preprint arXiv:2001.04385*, 2020.
- [67] Christopher Rackauckas and Qing Nie. Differentialequations.jl—a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1):15, 2017.
- [68] Christopher Rackauckas and Qing Nie. Confederated modular differential equation apis for accelerated algorithm development and benchmarking. *Advances in Engineering Software*, 132:1–6, 2019.
- [69] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.
- [70] Elisabeth Roesch, Christopher Rackauckas, and Michael P. H. Stumpf. Collocation based training of neural ordinary differential equations. *Statistical Applications in Genetics and Molecular Biology*, 20(2):37–49, 2021.
- [71] Pedro Sanchez, Jeremy P. Voisey, Tian Xia, Hannah I. Watson, Alison Q. O’Neil, and Sotirios A. Tsaftaris. Causal machine learning for healthcare and precision medicine. *Royal Society Open Science*, 9(8):220638, 2022.

- [72] Ilya Orson Sandoval, Panagiotis Petsagourakis, and Ehecatl Antonio del Rio-Chanona. Neural odes as feedback policies for nonlinear optimal control, 2022.
- [73] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.
- [74] Constantinos I. Siettos and Lucia Russo. Mathematical modeling of infectious disease dynamics. *Virulence*, 4(4):295–306, 2013. PMID: 23552814.
- [75] Eric Silverman, Umberto Gostoli, Stefano Picascia, Jonatan Almagor, Mark McCann, Richard Shaw, and Claudio Angione. Situating agent-based modelling in population health research. *Emerging Themes in Epidemiology*, 18(1):10, Jul 2021.
- [76] Byron Tasseff, Carleton Coffrin, Andreas Wächter, and Carl Laird. Exploring benefits of linear solver parallelism on modern nonlinear optimization applications, 09 2019.
- [77] Melissa Tracy, Magdalena Cerdá, and Katherine M Keyes. Agent-Based modeling in public health: Current applications and future directions. *Annu Rev Public Health*, 39:77–94, January 2018.
- [78] Ch. Tsitouras. Runge-kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Comput. Math. Appl.*, 62(2):770–775, jul 2011.
- [79] Evren Mert Turan and Johannes Jasicke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 6:1897–1902, 2022.
- [80] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, Mar 2006.
- [81] Shihang Wang, Xuanyu Xu, Cai Wei, Sicong Li, Jingying Zhao, Yin Zheng, Xiaoyu Liu, Xiaomin Zeng, Wenliang Yuan, and Sihua Peng. Molecular evolutionary characteristics of sars-cov-2 emerging in the united states. *Journal of Medical Virology*, 94(1):310–317, 2022.
- [82] Christopher W. Weimer, J. O. Miller, and Raymond R. Hill. Agent-based modeling: An introduction and primer. In *2016 Winter Simulation Conference (WSC)*, pages 65–79, 2016.
- [83] Nana Wu, Keven Joyal-Desmarais, Paula A B Ribeiro, Ariany Marques Vieira, Jovana Stojanovic, Comfort Sanuade, Doro Yip, and Simon L Bacon. Long-term effectiveness of COVID-19 vaccines against infections, hospitalisations, and mortality in adults: findings from a rapid living systematic evidence synthesis and meta-analysis up to december, 2022. *Lancet Respir. Med.*, 11(5):439–452, May 2023.
- [84] Hualei Xin, Yu Li, Peng Wu, Zhili Li, Eric H Y Lau, Ying Qin, Liping Wang, Benjamin J Cowling, Tim K Tsang, and Zhongjie Li. Estimating the Latent Period of Coronavirus Disease 2019 (COVID-19). *Clinical Infectious Diseases*, 74(9):1678–1681, 09 2021.
- [85] J H Zaccai. How to assess epidemiological studies. *Postgrad Med J*, 80(941):140–147, March 2004.

A Altri Approcci

A.1 Modello ad Agente su Spazio Continuo

L'approccio iniziale prevedeva la modellazione di una popolazione attraverso l'utilizzo di uno spazio di modello continuo. Gli agenti sarebbero stati rappresentati come individui reali, con un'effettiva presenza all'interno dello spazio. Questo spazio di modello continuo poteva essere visualizzato come una griglia di dimensioni $N \times M$ in cui gli agenti venivano posizionati in modo casuale e rappresentati come punti colorati.

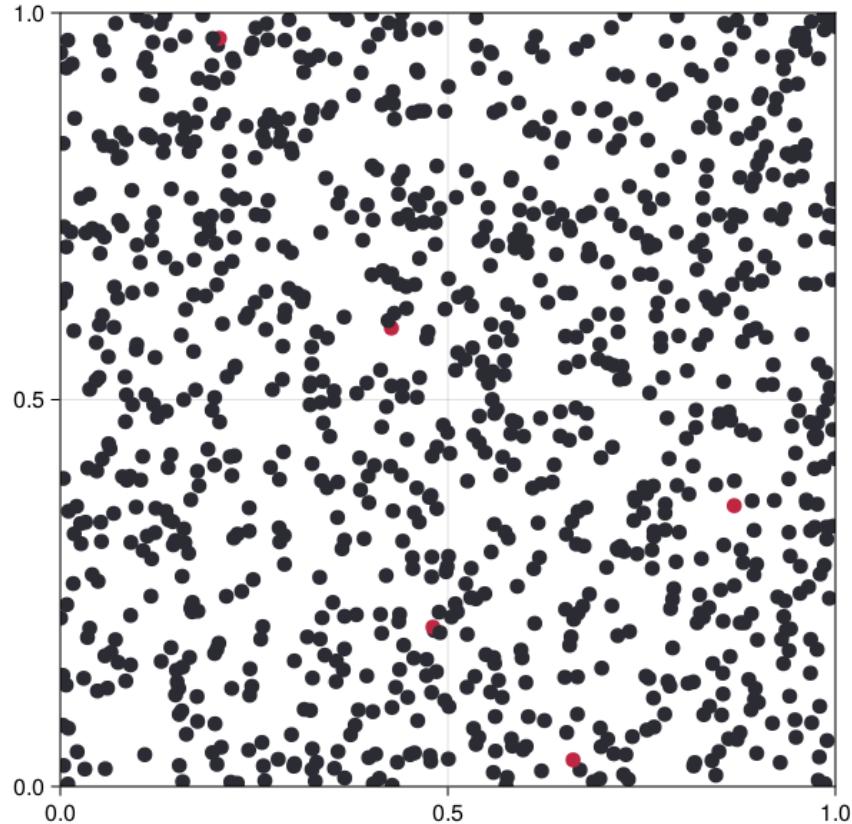


Figure 48: Esempio del modello modellato su spazio continuo

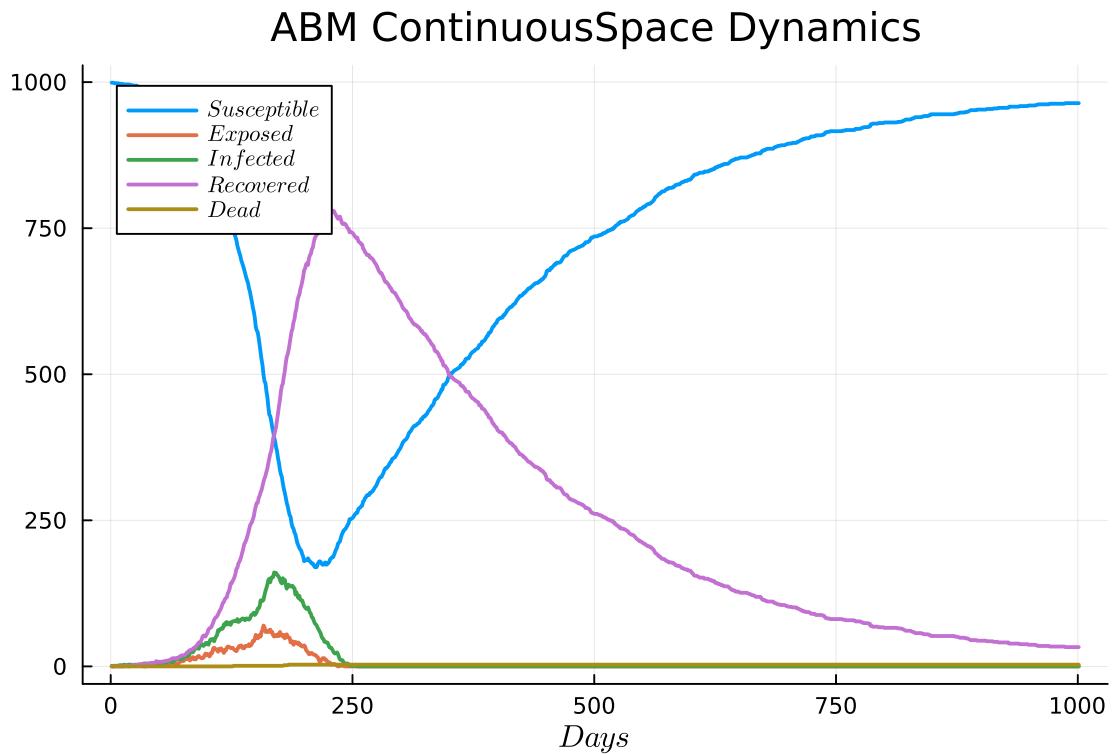


Figure 49: Esempio del comportamento delle curve nel modello continuo

Questo approccio prendeva spunto dalla fisica delle palline da biliardo per modellare le interazioni tra gli agenti all'interno dello spazio del modello. Ogni agente poteva casualmente infettare un vicino solo se interagivano, e questa interazione veniva modellata come un urto elastico tra corpi, anche se non rappresentava un vero simulatore di urti elastici, bensì ne prendeva spunto.

Tuttavia, questo approccio è risultato eccessivamente granulare e poco adatto agli obiettivi del progetto. Inoltre, richiedeva una quantità considerevole di risorse computazionali e tempo, il che ha portato alla sua sostituzione con un approccio più astratto e flessibile.

A.2 Modello ad Agente con Spazio a Grafo e Modellazione Singolo Agente

Un secondo approccio è stato progettato per consentire un maggiore controllo sullo spazio di modello e la sua evoluzione locale, piuttosto che sugli agenti individuali. Tuttavia, questo approccio ha incontrato problemi significativi, tra cui tempi di esecuzione estremamente lunghi e un comportamento delle curve epidemiologiche completamente divergente rispetto al modello SEIR deterministico. Questo comportamento anomalo è stato osservato in presenza di variazioni nel parametro R_0 , e la causa rimane sconosciuta.

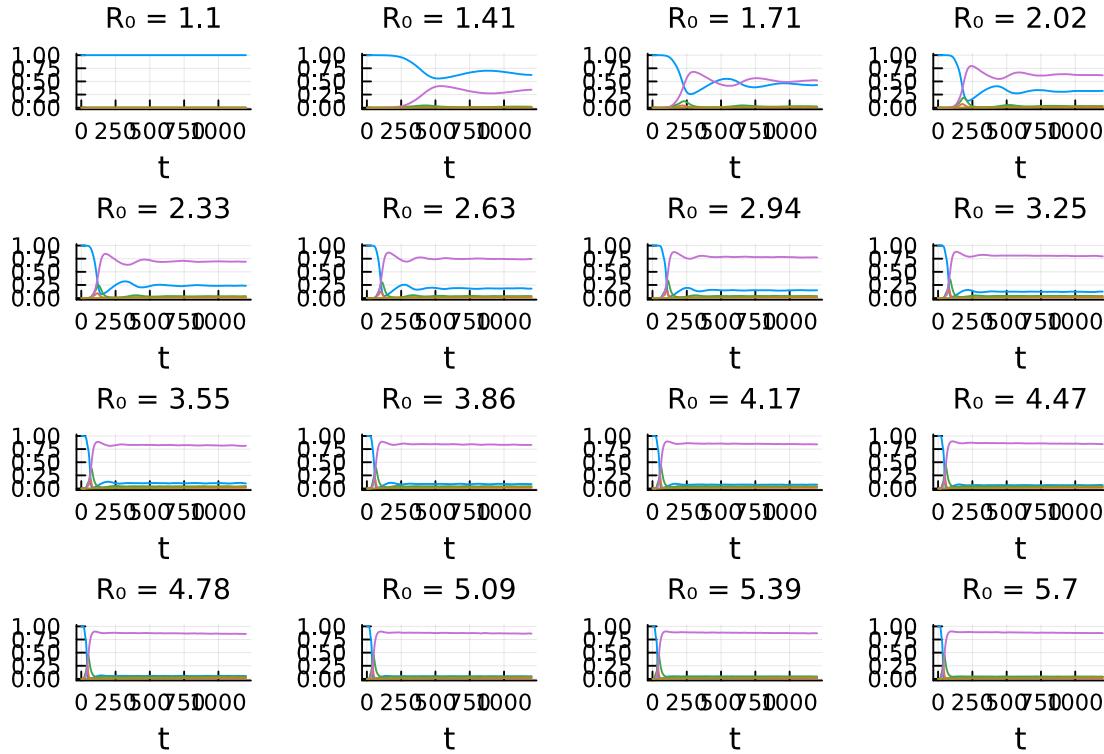


Figure 50: Comportamento modello ABM su spazio a grafo al variare del parametro R_0

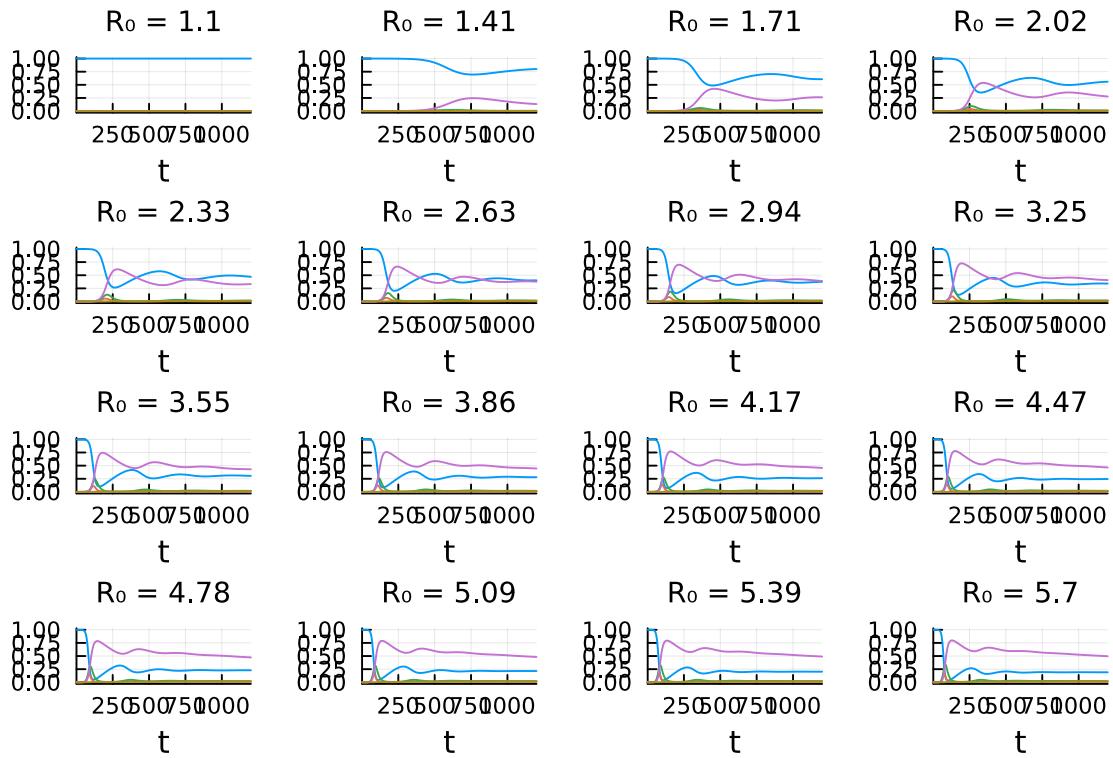


Figure 51: Comportamento modello SEIR al variare del parametro R_0

Anche se è stata formulata una formula approssimativa per descrivere la relazione tra i risultati del modello ABM e SEIR, questo comportamento rimane inspiegabile.



Figure 52: Formula che si occupa di descrivere il rapporto tra il comportamento del modello scartato e del modello SEIR. In particolare questa formula descrive il rapporto tra gli R_0

L'uso di un approccio basato su un grafo e la modellazione del singolo agente ha portato a risultati incoerenti e ha reso necessario abbandonare questa metodologia a favore di alternative più promettenti.

A.3 Controllore Ipopt

È stato esplorato un terzo approccio basato su Ipopt (Interior Point OPTimizer), un pacchetto software per l'ottimizzazione non lineare su larga scala. Ipopt è progettato per trovare soluzioni locali a problemi di ottimizzazione matematica, con funzioni obiettivo e vincoli non lineari. [80]

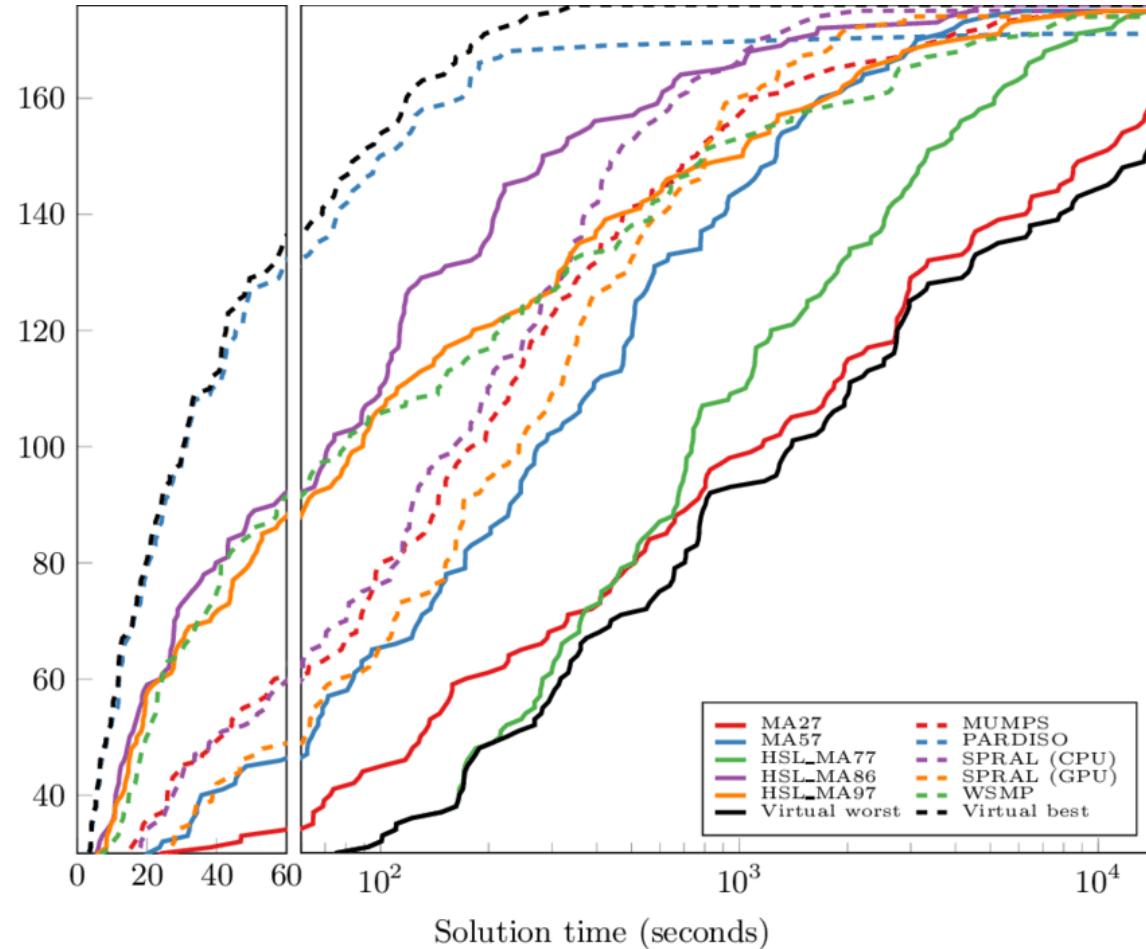


Figure 53: Comparison of Ipopt performance over various linear solvers using the two-dimensional partial differential equation test problem set. [76]

L'approccio ha previsto l'applicazione di un sistema di monitoraggio e intervento all'interno del modello di simulazione, con l'obiettivo di ridurre il numero di individui infetti. Le regole per il comportamento del modello sono state definite, inclusa una rappresentazione degli stati SEIR e un nuovo stato per il controllo delle risorse.

```

● ● ●
1 extra_ts = collect(δt:δt:timeframe[2]-δt)
2 model = InfiniteModel(Ipopt.Optimizer)
3 set_optimizer_attribute(model, "print_level", 0)
4
5 @infinite_parameter(model, t ∈ [timeframe[1], timeframe[2]], num_supports = length(extra_ts) + 2,
6 derivative_method = Orthogonalcollocation(2))
7 add_supports(t, extra_ts)
8
9 @variable(model, S ≥ 0, Infinite(t))
10 @variable(model, E ≥ 0, Infinite(t))
11 @variable(model, I ≥ 0, Infinite(t))
12 @variable(model, R ≥ 0, Infinite(t))
13 @variable(model, D ≥ 0, Infinite(t))
14 @variable(model, C ≥ 0, Infinite(t))
15
16 @variable(model, 0 ≤ u ≤ u_max, Infinite(t), start = 0.0)
17 @constraint(model, u_total_constr, ∫(u, t) ≤ u_total)
18
19 @objective(model, Min, C(timeframe[2]))
20
21 @constraint(model, S(0) == initial_condition[1])
22 @constraint(model, E(0) == initial_condition[2])
23 @constraint(model, I(0) == initial_condition[3])
24 @constraint(model, R(0) == initial_condition[4])
25 @constraint(model, D(0) == initial_condition[5])
26 @constraint(model, C(0) == Co)
27
28 @constraint(model, S_constr, ̈(S, t) == -(1 - u) * parameters[1] * parameters[2] * S * I + parameters[4] * R - parameters[7] * S)
29 @constraint(model, E_constr, ̈(E, t) == (1 - u) * parameters[1] * parameters[2] * S * I - parameters[3] * E)
30 @constraint(model, I_constr, ̈(I, t) == parameters[3] * E - parameters[2] * I - parameters[5] * I)
31 @constraint(model, R_constr, ̈(R, t) == (1 - parameters[5]) * parameters[2] * I - parameters[4] * R + parameters[7] * S)
32 @constraint(model, D_constr, ̈(D, t) == parameters[5] * parameters[2] * I)
33 @constraint(model, C_constr, ̈(C, t) == parameters[3] * E)
34
35 optimize!(model)
36 @info termination_status(model)
37
38 S_opt = value(S, ndarray=true)
39 E_opt = value(E, ndarray=true)
40 I_opt = value(I, ndarray=true)
41 R_opt = value(R, ndarray=true)
42 D_opt = value(D, ndarray=true)
43 C_opt = value(C, ndarray=true)
44 u_opt = value(u, ndarray=true)
45 obj_opt = objective_value(model)
46 ts = value(t)
47
48 # ritorno il vettore di quando applicare le contromisure
49 ut = unique(map((x) -> trunc(Int, x), findall(x -> x > 1e-3, u_opt) * 0.1))
50 filter!(e -> e ≠ 0, ut)
51 # ritorno il vettore del valore delle contromisure utilizzate durante il periodo specifico
52 u_opt_t = u_opt[trunc(Int, ut[1] / δt):trunc(Int, 1 / δt):trunc(Int, ut[end] / δt)]
53 mean(u_opt_t)

```

Figure 54: Definizione del controllore tramite Ipopt

```

1  @constraint(model, S(0) == initial_condition[1])
2  @constraint(model, E(0) == initial_condition[2])
3  @constraint(model, I(0) == initial_condition[3])
4  @constraint(model, R(0) == initial_condition[4])
5  @constraint(model, D(0) == initial_condition[5])
6  @constraint(model, C(0) == C0)
7
8  @constraint(model, S_constr, ̈(S, t) == -(1 - u) * parameters[1] * parameters[2] * S * I + parameters[4] * R - parameters[7] * S)
9  @constraint(model, E_constr, ̈(E, t) == (1 - u) * parameters[1] * parameters[2] * S * I - parameters[3] * E)
10 @constraint(model, I_constr, ̈(I, t) == parameters[3] * E - parameters[2] * I - parameters[5] * I)
11 @constraint(model, R_constr, ̈(R, t) == (1 - parameters[5]) * parameters[2] * I - parameters[4] * R + parameters[7] * S)
12 @constraint(model, D_constr, ̈(D, t) == parameters[5] * parameters[2] * I)
13 @constraint(model, C_constr, ̈(C, t) == parameters[3] * E)

```

Figure 55: Definizione regole del modello del controller

```

1  @variable(model, 0 ≤ u ≤ u_max, Infinite(t), start = 0.0)
2  @constraint(model, u_total_constr, ∫(u, t) ≤ u_total)

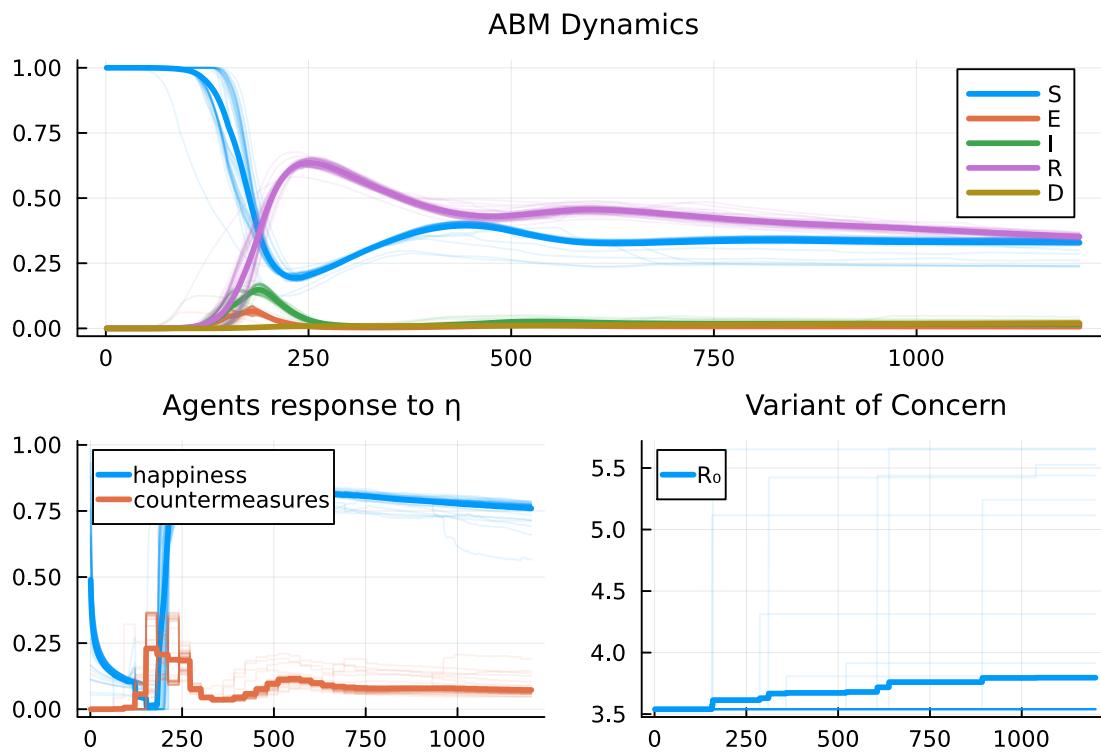
```

Figure 56: Definizione regole del modello del controller per le contromisure

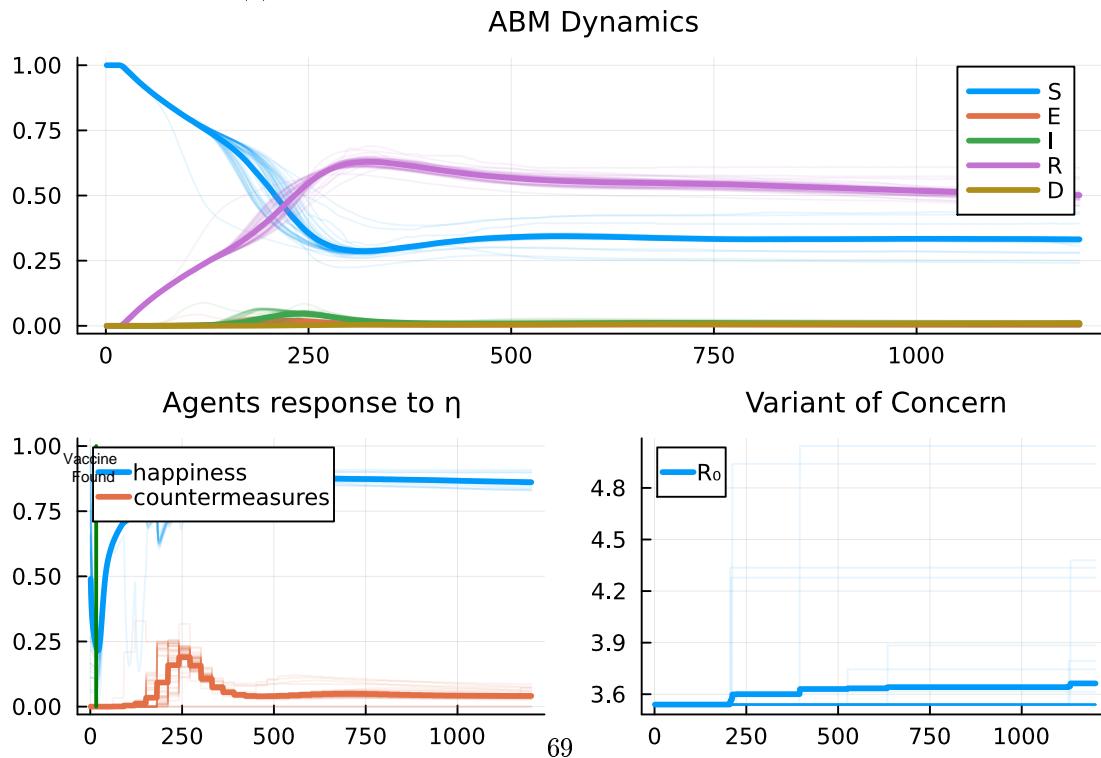
Sono state stabilite anche regole per il consumo di risorse, rappresentando le contromisure con il loro costo associato. L'ottimizzazione del modello è stata eseguita e sono stati restituiti i valori medi delle contromisure applicate.

I risultati ottenuti con Ipopt sono stati confrontati con quelli dell'implementazione personalizzata, rivelando somiglianze significative. Tuttavia, la possibilità di convertire facilmente il codice da CPU a GPU è stata determinante nella decisione di utilizzare l'implementazione personalizzata, in previsione di futuri miglioramenti delle prestazioni del codice.

L'approccio ibrido del controllore è stato preferito per la sua semplicità e la possibilità di mascherare la funzione obiettivo tramite una rete neurale, semplificando notevolmente la definizione delle regole rispetto all'implementazione esplicita di Ipopt.



(a) Risultato applicazione controllore tramite la suite Ipopt



(b) Risultato applicazione controllore tramite la suite Ipopt