



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

August 31, 2023

Anno Accademico 2022-2023

Abstract

Lo scopo di questa tesi è quello di studiare il comportamento di modelli di simulazione e di intervento utilizzando il linguaggio di programmazione Julia, e i suoi framework Agents.jl, SciML.ai. In particolare viene proposto l'utilizzo del framework Agents.jl come metodo per la simulazione di sistemi complessi e il framework SciML.ai come base per lo sviluppo di un controllore tramite tecniche ibride di machine learning, come ad esempio l'utilizzo delle Neural ODE. Lo scopo principale è quello di avere un approccio ibrido e dinamico ai due mondi, dai quali è possibile trarre il massimo vantaggio. L'approccio generale è stato modellato come un problema complesso che lavora con una struttura dati a grafo, simulando una rete sociale, in cui i nodi del grafo caratterizzano gli agenti del modello. Per ottenere un miglioramento delle prestazioni e della stabilità, il modello è stato ibridato con un sistema di ODE. Il controllore si basa sull'idea di una Neural ODE che controlla in maniera autonoma e automatica il livello di contromisure applicate. Queste vengono identificate come una riduzione dell'indice di infettività R_0 ; per evitare di applicare livelli di contromisure al lato pratico insostenibili, è stato inserito un valore di contenimento chiamato *happiness*, il quale agisce come una funzione di costo e controllo delle contromisure stesse.

Contents

1	Introduzione	1
2	Stato dell'Arte	4
2.1	Epidemiologia	4
2.2	Modelli Compartmentali	7
2.2.1	Il modello Kermack-McKendrick	7
2.3	Modelli ad Agente	12
2.3.1	Discretizzazione	14
2.4	Julia	16
2.4.1	Agents.jl	16
2.4.2	SciML.ai	19
3	Metodi e Modelli	30
3.1	Approccio con Rete Sociale	30
3.1.1	Agente	32
3.1.2	Spazio e Modello	33
3.1.3	Funzione di avanzamento agente	36
3.1.4	Funzione di avanzamento modello	38
3.2	Monitoraggio e Intervento	43
4	Risultati Ottenuti	51
4.1	Nessun intervento	52
4.2	Intervento non farmaceutico	54
4.3	Intervento farmaceutico	56
4.4	Intervento farmaceutico e non farmaceutico	57
4.5	Analisi di sensitività	58
4.5.1	Analisi comportamento dato un numero di nodi infetti iniziali variabile	60
4.5.2	Analisi comportamento dato il valore di migrazione variabile	61
4.5.3	Analisi comportamento dato il numero di nodi della rete variabile	62
4.5.4	Analisi comportamento data la copertura dei nodi variabile	63
5	Sviluppi Futuri	65
5.1	Perfezionamento Controllore	65
5.2	Miglioramento funzione happiness	65
5.3	Perfezionamento generale modello	65
5.4	Ottimizzazione generale	65
A	Approcci scartati	72
A.1	Modello ad Agente su spazio continuo	72
A.2	Modello ad Agente con spazio a grafo e modellazione singolo agente	74
A.3	Controllore Ipopt	77

List of Figures

1	Esempio di correlazione spuria	4
2	Un altro esempio di correlazione spuria	5
3	Esempio della struttura del modello SIR	8
4	Esempio di modello SEIRS [8]	10
5	Esempio di modello SEIR stocastico	11
6	Agent-Based Epidemiological Modeling of COVID-19 in Localized Environment [17]	12
7	Rappresentazione schematica di un modello ad agente	13
8	Esempio di comportamento emergente nella simulazione degli stormi di uccelli	14
9	Esempio di differenti tipologie di discretizzazione, siano esse nel tempo o nei valori	15
10	Tabella comparativa [1]	17
11	Esempio di rappresentazione di uno spazio a grafo con archi pesati	18
12	Esempio di metriche per il calcolo della distanza su uno spazio a griglia	19
13	Esempio applicazione UODE e Neural ODE [63] [66]	20
14	Differenti approcci di apprendimento di machine learning	21
15	Esempio di funzionamento di una rete neurale	22
16	Esempio grafico ODE rappresentante la famosa equazione di Lotka-Volterra	23
17	Esempio definizione ODE in Julia	24
18	Esempio implementazione di una rete neurale in Julia	25
19	Esempio implementazione NeuralODE tramite DiffEqFlux.jl	26
20	Tabella comparativa tra le varie implementazioni dei vari risolutori [62]	27
21	Esempio comparativo tra funzionamento Machine Learning e Deep Learning	28
22	Comportamento UDE nell'approssimazione di fenomeni non lineari [66]	29
24	Esempio di grafo sociale	31
25	Codice Agente	32
26	Codice Modello	33
28	Matrice di migrazione dato un MigrationRate di 0.01	34
29	Funzione che crea la matrice di migrazione data la topologia di un grafo	35
30	Definizione dei sistemi di ODE associati ad ogni agente	36
31	Comportamento agente	36
32	Funzione atta a calcolare lo spostamento di agenti da un nodo all'altro del grafo	37
33	Funzione atta a calcolare la felicità degli agenti	38
34	Funzione di avanzamento del modello	39
35	Funzione che si occupa di generare la VOC	40
36	Funzione che si occupa di simulare la ricerca di un vaccino e la sua successiva applicazione	41
38	Definizione controller all'interno del modello ad agente	44
39	Funzione per il calcolo del limite superiore per le contromisure non farmaceutiche	45
40	Definizione controllore tramite Neural ODE	46
41	Definizione funzioni di appoggio del controllore	47
42	Definizione funzioni di addestramento del controllore	48
43	Esempio di utilizzo congiunto di due ottimizzatori differenti in uno stesso ciclo di addestramento	49
44	Funzione di attivazione Swish	50
45	Grafico cumulativo del modello senza alcun tipo di intervento	52

46	Grafico delle mutazioni del virus SARS-COV2 preso da Our World in Data	53
47	Grafico cumulativo del modello con intervento non farmaceutico del controllore	54
48	Grafico cumulativo del modello con intervento del controllore tramite interventi farmaceutici come ad esempio un vaccino	56
49	Grafico cumulativo del modello con intervento del controllore tramite vaccino e metodi di prevenzione non farmaceutici	57
50	Grafico rappresentante l'analisi di sensitività del modello	58
51	Parametri usati per l'analisi di sensitività del modello ad agente	59
56	Esempio della funzione atta alla creazione di un grafo dato un determinato livello di copertura	64
57	Esempio del modello modellato su spazio continuo	72
58	Esempio del comportamento delle curve nel modello continuo	73
59	Comportamento modello ABM su spazio a grafo al variare del parametro R_0	74
60	Comportamento modello SEIR al variare del parametro R_0	75
61	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	75
63	Comparison of Ipopt performance over various linear solvers using the two-dimensional partial differential equation test problem set. [76]	77
64	Definizione del controllore tramite Ipopt	78
65	Definizione regole del modello del controller	79
66	Definizione regole del modello del controller per le contromisure	79

1 Introduzione

L'utilizzo di metodi e tecniche sempre più sofisticati da parte della comunità scientifica, in particolar modo da parte di epidemiologi e medici, è sempre stato argomento di grande dibattito e interesse. Negli ultimi anni il mondo si è espanso esponenzialmente divenendo sempre più connesso, incrementando drasticamente la probabilità che un virus affligga a livello mondiale la popolazione creando un disastro senza precedenti.

La storia dell'uomo è costellata di epidemie, e solamente alcune tra loro si sono guadagnate il privilegio di essere ricordate e tra queste solamente una piccola parte ha ottenuto il primato di essere ricordata come una catastrofe. Forse è proprio questo che le ha rese così salde nell'immaginario comune aumentando la loro già imponente aurea di terrore.

Per fare degli esempi possiamo citare: la peste nera che a partire dalla metà del 14esimo secolo ritornò in Europa uccidendo venti milioni di persone in soli sei anni, l'epidemia di tifo che non solo fu fatale durante il periodo delle crociate, ma anche durante la seconda guerra mondiale all'interno dei campi di concentramento nazisti, oppure le varie influenze, prima tra tutte quella spagnola la quale nel periodo del primo dopo guerra, ovvero tra il 1918 e il 1920, uccise 50 milioni di persone in tutto il mondo, oppure l'epidemia di AIDS, che ha all'attivo dal 1981 oltre 75 milioni di casi e 35 milioni di morti.

Se ci si sofferma un attimo pensando proprio a questo tipo di pandemia, quella influenzale, ci viene da tirare un sospiro di sollievo, in quanto oramai come cittadini del primo mondo l'idea di influenza non ci fa più così paura. Eppure dovrebbe, e sfortunatamente lo abbiamo ricordato nel peggior modo possibile.

La pandemia di SARS-CoV-2 scoppia negli ultimi mesi del 2019 condizionò l'intera umanità per circa 3 anni, e nel momento che sto scrivendo queste righe continua a condizionarla. Questa pandemia si è macchiata di aver mietuto, allo stato attuale delle cose, quasi 7 milioni di vite accertate. Questa tragedia rimarrà impressa nella memoria umana in quanto capace di aver messo in crisi l'intero sistema governativo mondiale, creando uno stato di allarme, panico e alle volte perfino isteria, che pochi altri avvenimenti sono stati in grado di fare.

Osservando le statistiche proprie di questa epidemia, ciò che balza subito all'occhio è sicuramente il numero associato alle morti e agli infetti: quasi 7 milioni di morti e più di 700 milioni di infetti [58]. Numeri che da soli basterebbero a mettere a disagio qualsiasi lettore, eppure altri dati, nascosti a prima vista, possono fornire ulteriori macabre informazioni. Un esempio tra tutti è l'effetto che un'epidemia del genere ha avuto sull'economia [37], portando disagi generalizzati ovunque.

Legato al disagio economico vi è un altro dato preoccupante definito come poverty trap [10]. Questo fenomeno nasce in quegli ambienti in cui le condizioni di povertà economica e la prevalenza di malattie possono imprigionare una società in uno stato persistente di bassa sanità e sempre maggiore povertà; questo fenomeno ciclico si auto sostiene e prende il nome di positive feedback. Per ultimo, ma non per questo meno importante, l'effetto di una pandemia può portare ad instabilità governative le quali possono portare a un arretramento del sistema sanitario e di welfare [24], ricadendo come sopra descritto all'interno del fenomeno di positive feedback.

Questi sono solamente alcuni esempi dei problemi che possono sorgere, e che sono sorti con lo scoppio di una pandemia globale come è stata quella del COVID-19. Per questo la comunità scientifica, in

particolare gli epidemiologi cercano soluzioni sempre più efficaci e accurate per prevenire, arginare e contrastare avvenimenti del genere.

L'epidemiologia è una disciplina nata di recente evolutasi insieme alle esigenze della società ogni qualvolta una nuova emergenza sanitaria faceva irruzione nella quotidianità. La prima definizione di epidemiologia è stata data da Lilienfeld [29] nel 1978, e cita:

l'epidemiologia è un modo di ragionare riguardo le malattie, e si occupa di effettuare inferenza biologica derivata dall'osservazione di fenomeni patologici all'interno di una popolazione.

Con il tempo questa definizione ha subito molti cambiamenti, derivati anche e soprattutto dall'espansione degli ambiti relazionati all'epidemiologia; con l'aggiunta ad esempio della farmacoepidemiologia, dell'epidemiologia molecolare e dell'epidemiologia genetica. Non solo, ambiti come etica, filosofia ed epistemologia sono estremamente importanti ed influenti nella crescita e sviluppo di questa materia [29].

Attualmente con il termine epidemiologia si intende la disciplina biomedica che studia la distribuzione e la frequenza delle malattie ed eventi di rilevanza sanitaria nella popolazione. L'epidemiologia si occupa di analizzare le cause, il decorso e le conseguenze delle malattie. Secondo Last et al (1998) l'epidemiologia è definita come:

lo studio della distribuzione e dei determinanti delle situazioni o degli eventi collegati alla salute in una specifica popolazione, e l'applicazione di questo studio al controllo dei problemi di salute.

Uno degli strumenti più utilizzati in epidemiologia sono le simulazioni software. Dato lo scopo dell'epidemiologia, questa necessita di avvalersi di modelli matematici [74] che aiutano i ricercatori a trarre conclusioni sul sistema che analizza. Sistemi simili vengono definiti come complessi, [30] [49] ovvero sistemi dinamici a multicomponenti che tipicamente interagiscono tra loro, e che sono descrivibili tramite modelli matematici.

I primi modelli chiamati compartmentali [8] basano il proprio funzionamento sullo studio di gruppi di individui disgiunti che interagiscono tra loro, analizzabili tramite un sistema di equazioni ordinarie differenziali (ODE) [11]. Questo approccio è stato teorizzato da Kermack e McKendrick nel 1927 applicando una modellazione matematica al comportamento delle malattie infettive su un gruppo di individui, tenendo in considerazione la variabile del tempo. Da qui è nato il famoso modello Susceptible-Infectious-Recovered (SIR), il quale tuttora viene utilizzato ampiamente insieme alle sue varianti. Successivamente con l'unione di svariate discipline come: la teoria dei giochi, i sistemi complessi, i comportamenti emergenti, la sociologia computazionale, i sistemi multiagente e la programmazione evoluzionaria sono nati i modelli ad agente [82] [7], modelli computazionali autonomi per la simulazione di sistemi complessi.

Grazie alla loro teorizzazione negli anni 1940 ma soprattutto grazie al loro uso concreto dagli anni 1990 il mondo della simulazione ha avuto un grande balzo in avanti. Ciò che rende questi sistemi molto flessibili e potenti è la capacità di far emergere spontaneamente dei comportamenti complessi da un insieme di regole semplici. Ovviamente però la richiesta di risorse e capacità computazionale è estremamente elevata rispetto alla controparte di modelli puramente matematici.

Ogni approccio ha dei pro e dei contro e la vera potenza di questi modelli sta principalmente nella acutezza di chi deve poi farne uso. In base ai compromessi e le assunzioni fatte durante la fase

di modellazione ogni approccio può rivelarsi vincente. Uno dei compromessi maggiori che viene generalmente applicato a questi modelli è quello della discretizzazione dell'ambiente [48]. La realtà è a tutti gli effetti un insieme continuo di eventi, ma essendo tutti i dispositivi di calcolo discreti è impossibile simulare avvenimenti continui (siano essi nel tempo o nello spazio) in maniera diretta e perciò bisogna fare dei compromessi, trasformando il proprio spazio di lavoro in uno più adatto alle macchine che lo devono simulare.

Un'altra assunzione generalista e approssimativa per definizione, ma necessaria per la costruzione di un modello che analizzi il decorso di un'epidemia all'interno di una società, è quella legata alla tipologia di comportamento che verrà mostrato dagli esseri umani in condizioni di pericolo [77] [25]. Queste sono solo alcune delle assunzioni e compromessi che bisogna fare quando ci si approccia al mondo della simulazione. Tuttavia applicare delle assunzioni, alle volte anche forti e controiduttive, non sempre è sinonimo di errore. Alle volte tramite lo studio della causalità degli eventi [30] [60] è possibile astrarre un set minimale di assunzioni che se applicate danno la capacità al modello di rappresentare molto bene il comportamento desiderato.

Certamente ci saranno alcune discrepanze soprattutto in casi estremi, ma è un'eventualità che viene tenuta in considerazione ogni qualvolta si parla di simulazione, e che non è possibile eliminare del tutto.

La causalità o causa effetto, è quell'influenza per cui un evento, un processo, uno stato o un oggetto contribuiscono nella produzione di un nuovo evento, processo, stato o effetto, dove la causa è parzialmente responsabile dell'effetto e l'effetto è parzialmente dipendente dalla causa.

A prima vista non sembra una tematica molto complessa o di difficile approccio, complice il fatto che in quanto esseri umani siamo una specie che si è evoluta per trovare una correlazione tra gli eventi, ma correlazione non significa, e soprattutto non implica, causalità [4]. Questo tema si riferisce all'incapacità legittima di dedurre la relazione di causa - effetto tra due eventi o variabili solamente sulla base di una osservazione della loro associatività o correlazione.

Diventa chiaro come una delle parti più complesse dell'epidemiologia sia proprio quella di stabilire le cause di un dato fenomeno e comprendere come un determinato intervento su di esse influenzi quest'ultimo [30] [60].

Questa rapida introduzione all'epidemiologia e alla simulazione di sistemi complessi tramite l'utilizzo in particolare di modelli ad agente sarà l'argomento cardine dell'intero elaborato. Nelle sezioni successive verrà proposta un'analisi dello stato dell'arte dell'attuale panorama epidemiologico e simulativo, con alcune rapide digressioni sul problema della discretizzazione, problema assai sentito nell'ambito della simulazione.

Successivamente verranno portati alla luce alcuni esempi e modelli di sistemi ad agente che si occupano di modellare differenti tipi di epidemie, tutte caratterizzate però dal fatto di essere epidemie infettive a diffusione principalmente aerea, come ad esempio l'ebola, l'aviaria e l'influenza.

Successivamente vi sarà un'analisi più dettagliata riguardo la pandemia da COVID-19 che recentemente ci ha colpito. Infine verranno analizzati alcuni metodi di monitoraggio di queste simulazioni con un focus su alcuni metodi di intervento.

2 Stato dell'Arte

2.1 Epidemiologia

L'epidemiologia è la disciplina biomedica che studia la distribuzione e la frequenza delle malattie ed eventi di rilevanza sanitaria nella popolazione. Si occupa di analizzare le cause, il decorso e le conseguenze delle malattie.[30] [60]

L'epidemiologia è una disciplina molto pratica, che visto l'obiettivo che si pone, ovvero quello di trovare le cause relative ad un dato effetto, non può esentarsi dagli svariati problemi che gravitano e definiscono questo nobile obiettivo, primo tra tutti: cosa vuol dire che un evento è causa di un altro e come definisco questo tipo di rapporto in maniera inequivocabile?

Questi interrogativi possono sembrare banali in quanto come specie ci siamo evoluti per trovare una correlazione di causalità tra eventi anche quando questi non ne hanno. Ad esempio se fossimo in un bosco, al buio e soli con l'unico rumore ad accompagnarcici che è quello di una piacevole brezza estiva, se percepissimo un rumore tra i cespugli, molto probabilmente penseremmo che c'è qualcosa che non va, che ci sia un pericolo in agguato, un predatore, anche se magari la motivazione è la suddetta brezza.

Questo adattamento evolutivo ci ha permesso di sopravvivere in situazioni di pericolo, ma sfortunatamente quando si parla di scienza e di dati, non sempre l'istinto è qualcosa a cui affidarsi, in quanto i dati di per loro non dicono assolutamente nulla, siamo noi in quanto individui dotati di intelletto, tecniche e metodi a dover estrapolare dei significati che verifichiamo essere corretti e inequivocabili.

Se ci soffermiamo su questa ultima affermazione, possiamo essere facilmente tratti in inganno. Prendiamo per esempio il seguente grafico che mostra in maniera *inequivocabile* come la spesa da parte degli USA sulla ricerca aerospaziale sia direttamente collegata al numero di suicidi per strangolamento.

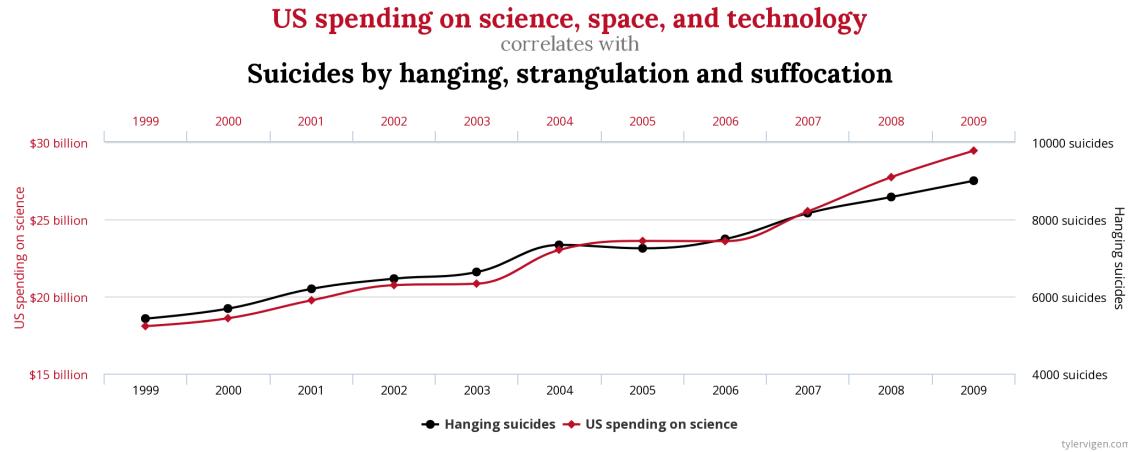


Figure 1: Esempio di correlazione spuria
<https://www.tylervigen.com/spurious-correlations>

Affidandoci solamente al grafico e ai dati riportati in figura, ci verrebbe da pensare che le due categorie siano in qualche modo correlate, e che il governo americano debba essere fermato, in quanto responsabile di incitamento al suicidio. Ebbene non è così, questo è un caso di **relazione spuria**, ovvero che due o più variabili sono associate ma non causalmente correlate.

Associatività e causalità infatti non sono la stessa cosa, ed è bene quando si studia l'una, non confondersi con l'altra. In statistica, una correlazione tra dati è una qualsiasi relazione che vi è tra due o più variabili, sia essa di tipo causale oppure non. Nell'esempio di prima la correlazione tra le due variabili potrebbe essere banalmente il tempo, infatti con il passare del tempo la spesa media per la ricerca aerospaziale è continuata a salire per via del sempre più alto interesse e investimento nel settore e sfortunatamente con il passare degli anni si è registrato un aumento costante del numero di suicidi.

Il **bias** che abbiamo verso la ricerca di un intreccio tra gli eventi, in maniera che questi siano sempre contigui l'uno con l'altro, in maniera che si possa tracciare una chiara e distinta linea dal primo all'ultimo ci trae in inganno quando questi sembrano esserlo ma in realtà non lo sono. Molto spesso la risposta più semplice è anche quella meno interessante, benché corretta, ovvero che due eventi completamente slegati tra loro possono avere andamenti simili, così come differenti addendi possono portare lo stesso risultato;

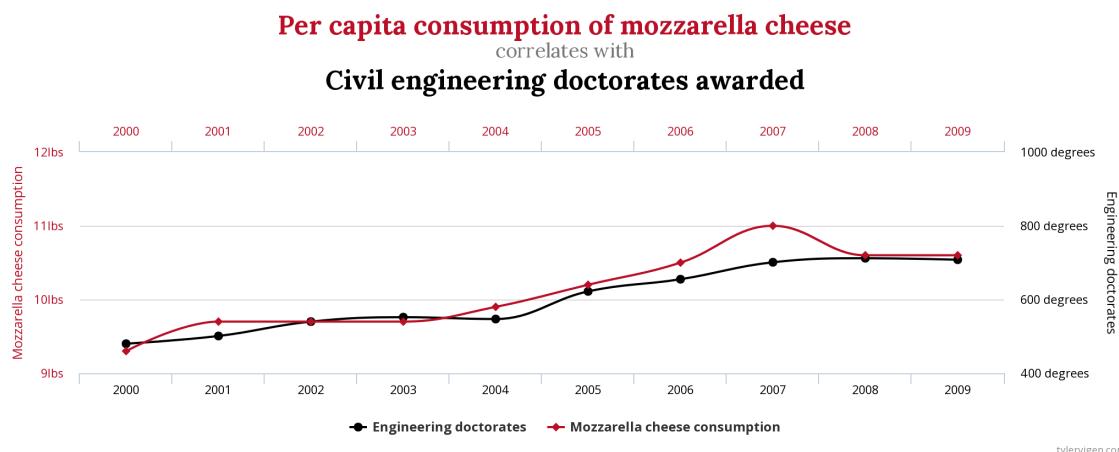


Figure 2: Un altro esempio di correlazione spuria
<https://www.tylervigen.com/spurious-correlations>

Il problema della causalità non è da prendere sotto gamba ed è uno dei problemi cardine quando si vogliono determinare e applicare degli interventi all'interno di una popolazione per cercare di mitigare ad esempio la diffusione di un agente patogeno [60].

Come è stato già introdotto precedentemente, i dati di per loro non dicono nulla, siamo noi che dobbiamo imparare a capire il significato di ciò che esprimono. Un esempio estremamente calzante di come pur avendo bene in mente il problema sopra citato delle correlazioni *spurie*, si possa comunque essere ingannati dai dati, è il seguente:

Poniamo di essere un medico e di dover decidere se prescrivere o meno ad un paziente un determinato medicinale. Per aiutarci nella decisione abbiamo la storia clinica del paziente e i risultati di uno studio su una nuova medicina che attesta di curare il malessere del paziente. Questa nuova medicina è stata testata su un gruppo di settecento persone divise in due pari sottogruppi in cui 350 pazienti decidevano autonomamente se prendere o meno la medicina e 350 decidevano autonomamente il contrario. I risultati sono i seguenti:

Simpson's paradox						
Category	Patients	Recovered	% Recov- ered	Patients	Recovered	% Recov- ered
Men	87	81	93%	270	234	87%
Women	263	192	73%	80	55	69%
Combined data	350	273	78%	350	289	83%

Questi risultati sembrano suggerire come la prescrizione di questa nuova medicina non aiuti i pazienti a stare meglio. Tuttavia questo risultato cade nel così detto *paradosso di Simpson* per cui i dati aggregati relativi ad uno specifico trattamento sembrano descrivere una sua perdita di efficacia relativamente ai singoli dati delle singole categorie in esame, portando un lettore non attento a cadere nell'inganno di pensare che ci sia una perdita di efficacia. Questo esempio pone l'accento sul fatto che non sempre estrapolare informazioni da dei dati aggregati può risultare efficace, e anzi, alle volte questi possono ingannare. In questi casi bisogna estrapolare le informazioni di causalità dai dati singoli.

è chiaro che non sia così semplice comprendere le cause di un determinato effetto, o insieme di effetti. Conoscere l'agente patogeno, o quanto meno la sua natura può aiutare, ma non sempre è sufficiente. L'utilizzo di modelli di Machine Learning per l'estrapolazione di dati, di correlazioni e successivamente per la definizione di policy di intervento può risultare in un rischio non indifferente ma al contempo descrive un potente alleato per la definizione di policy di intervento all'interno di settori estremamente delicati come quelli sanitari [71].

2.2 Modelli Compartmentali

In epidemiologia i modelli compartmentali sono una tecnica di modellezione generica che si pre-dispone molto bene allo studio complessivo del comportamento di una malattia infettiva. Questa tecnica di modellazione si applica anche ad altre branche della scienza, una tra tutte la finanza.

Questa tecnica di modellazione matematica basa il proprio funzionamento sull'assunzione che, data una popolazione di individui, questi vengano etichettati in maniera differente, in base allo stato di progressione della malattia che hanno, o non hanno, contratto. Così facendo si vanno a definire dei compartimenti ben separati che possono interagire tra loro, ma che rimangono chiaramente distinti l'uno dagli altri.

Equazioni Ordinarie Differenziali

Un equazione ordinaria differenziale (ODE) è un equazione che coinvolge alcune derivate ordinarie di una funzione. Generalmente data una derivata di una funzione si vuole studiare la funzione di partenza e per fare questo bisogna trovare l'antiderivata (o integrale) della funzione in esame.

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

Generalmente però risolvere una ODE è più complicato di risolvere un semplice integrale, e pur sapendo che il principio alla base rimane la risoluzione di un integrale, la parte difficile è quella di determinare quale tipo di integrazione abbiamo bisogno per ricavare la nostra soluzione.

Fintanto che le equazioni associate a $\frac{dx}{dt}$ dipendono esclusivamente dalla variabile t e non dalla funzione $x(t)$ allora la loro risoluzione risulta molto più facile di quanto non sembri. Tuttavia nel caso in cui si avesse una funzione che dipende da $x(t)$ questo ragionamento si fa più complicato. Nel caso in cui $\frac{dx}{dt}$ dipende da $x(t)$ non è possibile applicare una semplice integrazione e successivamente il teorema fondamentale del calcolo, ma anzi necessitiamo di effettuare alcune manipolazioni, in particolare quelle che prendono il nome di *chain rules* o *u-substitution*.

2.2.1 Il modello Kermack-McKendrick

Secondo la definizione precedente possiamo separare la popolazione soggetta di studio in categorie, o compartimenti, e assumere una relazione temporale che descriva il passaggio di individui da un compartimento all'altro. Malattie che garantiscono un'immunità avranno una struttura compartmentale differenti da malattie che non la garantiscono.

Il modello che tutt'ora viene usato come riferimento e come base per lo studio e modellazione è il così detto modello **Susceptible, Infectious, Recovered** (SIR)

Questo modello è stato ideato all'inizio del 20esimo secolo, più precisamente nel 1927, da Kermack e McKendrick. Come introdotto questo modello si basa sull'assunzione che all'interno di una popolazione durante il decorso di una malattia vi possano esistere solamente tre stadi in cui un individuo può essere inserito:

- Susceptible: Questo stadio rappresenta lo stato iniziale per la maggior parte degli individui all'interno di una popolazione. Rappresenta il numero di persone che possono contrarre la malattia.

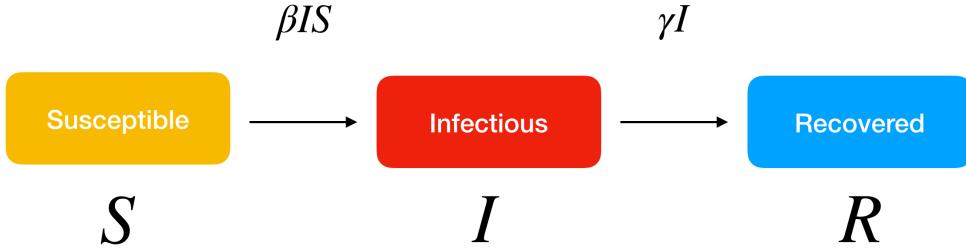


Figure 3: Esempio della struttura del modello SIR

- Infectious: Questo stadio rappresenta tutti quegli individui che dallo stato di Susceptible, dopo essere venuti in contatto con un individuo infetto, diventano a loro volta individui infetti.
- Recovered: Questo stadio rappresenta una duplice categoria, quella degli individui che alla fine del docoro della malattia sopravvivono ad essa, e quelli che invece muoiono a causa di questa. Generalmente questo stato viene anche definito come Removed.

Da questa semplice idea poi si è andato a sviluppare un modello matematico per descrivere come queste 3 categorie separate ma che si influenzano vicendevolmente, cambiano nel corso del tempo. La variabile indipendente del modello è il tempo, indicato come t e il tasso di trasferimento tra compartimenti, il quale è espresso matematicamente come derivate rispetto al tempo e alla dimensione del comparto, e come risultato questo modello è stato inizialmente formulato tramite l'utilizzo delle equazioni differenziali [11].

Durante la formulazione del modello in termini di derivate della dimensione di ogni comparto si assume che il numero di individui di ogni comparto sia rappresentabile come una funzione differenziabile nel tempo. Questa assunzione può essere una ragionevole approssimazione nel caso in cui vi siano molti individui per comparto, altrimenti potrebbe risentire di essere sospetta in caso contrario. Nel caso in cui si utilizzino le equazioni ordinarie differenziali come modello si assume che il comportamento della popolazione sia determinato completamente da una relazione deterministica. Questa assunzione può essere una ragionevole approssimazione, tuttavia esistono approcci di tipo *stocastico* che aggirano questo tipo di assunzione introducendo concetti probabilistici per la modellazione del modello.

Il modello in figura 3 è un caso speciale del modello proposto da Kermack e McKendrick nel loro articolo del 1927, ma è divenuto il modello base per la strutturazione di questo tipo di modelli. Questa tipologia di modello descrive il seguente sistema di equazioni ordinarie differenziali:

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

Questo sistema modella le seguenti assunzioni:

- In media un individuo della popolazione effettua sufficienti contatti con altri individui tale per cui è possibile far evolvere l'infezione con la seguente relazione $\beta \cdot N$ con N numero di individui totali.
- Gli infetti abbandonano il compartimento che determina gli infetti con un tasso pari a $\gamma \cdot I$.
- Non esistono entrate o uscite di individui dalla popolazione totale eccezione fatta per possibili uscite tramite morte data dalla malattia stessa.

Data la descrizione sopra fatta, in accordo con il primo punto, la probabilità di un contatto casuale da un individuo infetto con un individuo suscettibile, a cui è possibile trasmettere l'infezione, è di $\frac{S}{N}$, dove il numero di nuovi infetti è dato da $(\beta \cdot N) \cdot (\frac{S}{N}) \cdot I = \beta \cdot S \cdot I$. Questo approccio è criticabile in quanto si può argomentare come il tasso di contatti dovrebbe essere governato dal numero di infetti e non dei suscettibili, andando così a descrivere la seguente formula $(\beta \cdot N) \cdot (\frac{I}{N}) \cdot S = \beta \cdot S \cdot I$. In entrambi i casi si ottiene lo stesso tasso di infezione; tuttavia possono esistere approcci dove una scelta sia più adatta dell'altra.

La seconda assunzione non ha una chiara controparte con il mondo epidemiologico. Consideriamo l'insieme dei membri che sono stati infettati in un dato periodo di tempo e consideriamo $u(s)$ come il numero di questi che sono ancora infettivi dopo s unità di tempo. Se definiamo α come la frazione di infetti che abbandona la classe di appartenenza in un determinato periodo di tempo, otteniamo la seguente formula $= -\alpha u$, la cui soluzione è $u(s) = u(0)e^{-\alpha s}$. Con questo la frazione di infetti che rimangono infettivi dopo un periodo s è $e^{-\alpha s}$, per cui il periodo di infettività viene approssimato come una *distribuzione esponenziale* con media $\int_0^\infty e^{-\alpha s} ds = \frac{1}{\alpha}$.

Questa assunzione, per cui il tasso di contatto è proporzionale alla dimensione della popolazione N con una costante di proporzionalità β e una distribuzione esponenziale del tasso di ripresa, è generalmente troppo semplicistica. Tuttavia pur essendo un modello che descrive un andamento estremamente semplicistico e irrealistico, è stato osservato come modelli molto più complessi e realistici hanno dimostrato comportamenti estremamente simili a questo, facendo sì che questo modello, seppur semplice, sia un buon modello che approssima bene la realtà.

Il problema definito precedentemente non è possibile da risolvere analiticamente, però da ciò è possibile imparare il comportamento generale del sistema. Questo approccio permette di inserire un numero variabile di infetti all'interno di una popolazione per studiare se e come un'epidemia si propaghi. La quantità $\beta \cdot \frac{S(0)}{\alpha}$ è indicata come una soglia generalmente conosciuta come *basic reproduction number* e viene indicato come R_0 , il quale determina se può o meno avvenire un'epidemia. Se $R_0 < 1$ allora l'infezione andrà a morire prima di diventare una epidemia, mentre se fosse $R_0 > 1$ vi sarà un outbreak epidemico.

L'indice R_0 definisce il numero di infezioni secondarie causate da un singolo individuo durante il proprio periodo infettivo all'interno di una popolazione pienamente suscettibile di dimensione $K \approx S(0)$ durante l'intero decorso dell'infezione. In questa situazione un individuo infetto effettua $\beta \cdot K$ contatti per unità di tempo, ognuno di essi con un individuo suscettibile che produce nuove infezioni durante un periodo medio infettivo di $\frac{1}{\alpha}$; con questo si può ottenere che il valore di R_0 è $\beta \cdot \frac{K}{\alpha}$. Generalmente è difficile stimare il rateo di contatto β in una popolazione in quanto questo dipende fortemente dalla tipologia di malattia studiata ma anche dal comportamento sociale degli individui. Tuttavia esistono molteplici approssimazioni che permettono di modellare l'andamento di una malattia infettiva in presenza di pochi dati.

Il numero di infetti è stato osservato crescere esponenzialmente e questo può essere approssimato dalla seguente formula $\frac{dS}{dt} = (\beta \cdot K - \alpha) \cdot I$ e l'iniziale crescita è definibile come $r = \beta \cdot K - \alpha = \alpha \cdot (R_0 - 1)$. Il valore di r può essere stimato sperimentalmente all'inizio di una pandemia. Successivamente, data la possibilità di misurare sia K che α , β può essere calcolato come $\beta = \frac{r+\alpha}{K}$, tuttavia questo approccio è sensibile alla presenza di incompletezza dei dati e alla segnalazione parziale dei casi effettivi, andando ad avere una rappresentazione non accurata. Questa sensibilità nell'accuratezza è ulteriormente accentuata quando si studia l'outbreak di una malattia precedentemente sconosciuta o in cui i primi casi sono facilmente proni a diagnosi errate.

Derivazione modello SIR: modello SEIR

In molteplici malattie infettive esiste un periodo definito di *esposizione* in cui un individuo una volta infetto non è immediatamente infettivo per cui vi è un ritardo nella trasmissione effettiva della malattia. Questo periodo è irrilevante quando è estremamente breve. Tuttavia un periodo di esposizione lungo può portare a differenze significative nel modello. Generalmente per modellare questa variante è sufficiente inserire un nuovo comparto **E** il quale va a modificare il sistema di equazioni differenziali nel seguente modo:

$$\begin{cases} \frac{dS}{dt} = -\beta(N) \cdot S \cdot I \\ \frac{dE}{dt} = \beta(N) \cdot S \cdot I - \kappa \cdot E \\ \frac{dI}{dt} = \kappa \cdot E - \gamma \cdot I \\ \frac{dR}{dt} = \gamma \cdot I \end{cases}$$

Questo modello permette di integrare l'evenienza di casi *asintomatici* i quali sono infettivi e non totalmente associabili al comparto **E**.

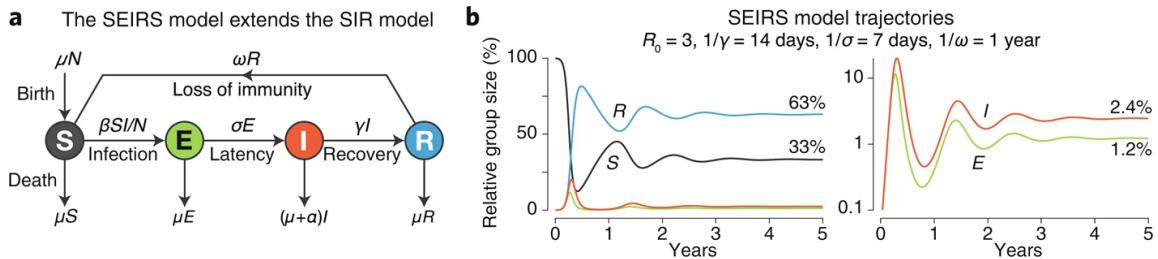


Figure 4: Esempio di modello SEIRS [8]

Modello SIR stocastico

Una delle assunzioni del modello presentato precedentemente, ovvero della tipologia **deterministica** è quella per cui la dimensione dei compatti è abbastanza grande da permettere di avere un mix omogeneo di membri al suo interno. Questa assunzione è sufficientemente ragionevole quando un'epidemia è già avviata, ma all'inizio la situazione può risultare estremamente differente. All'inizio di una epidemia la maggior parte della popolazione ricade all'interno della categoria suscettibile, ovvero che non è stata (ancora) infettata, e il numero associato agli individui infetti è relativamente piccolo. Il rateo di trasmissione della malattia dipende fortemente dal pattern dei contatti tra individui e servirebbe utilizzare una descrizione di questo pattern. Poiché il numero di

infetti è piccolo una descrizione del modello che utilizza una assunzione di *mass action* dovrebbe essere sostituito da un modello che incorpora degli effetti stocastici.

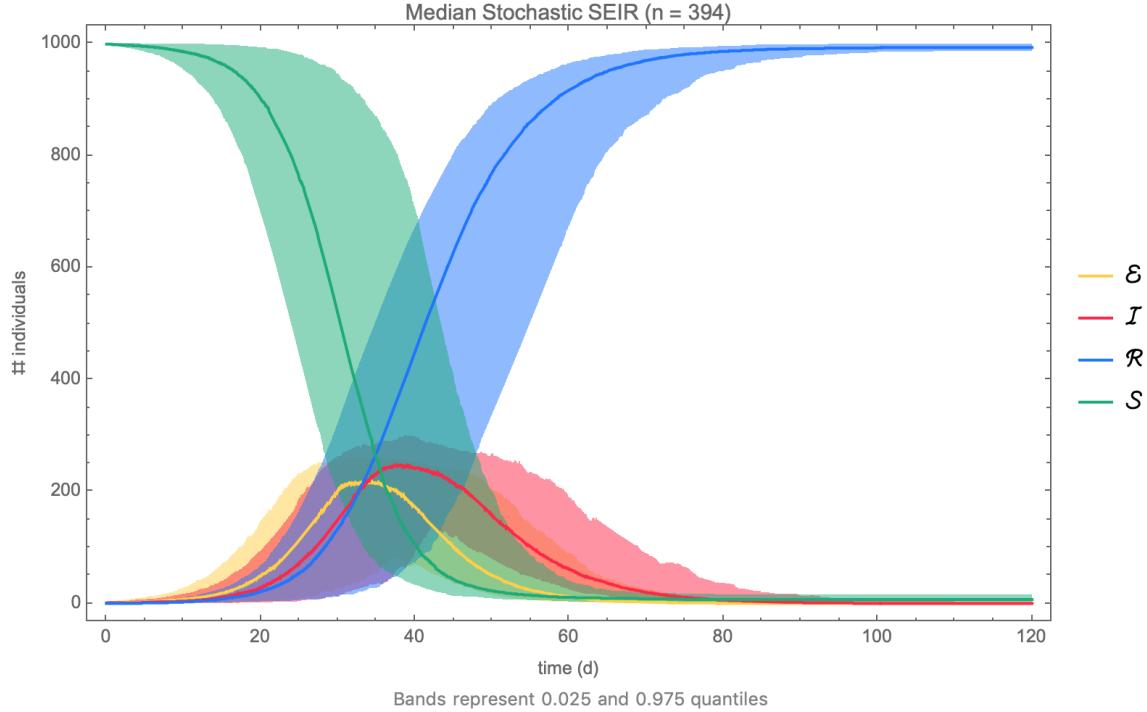


Figure 5: Esempio di modello SEIR stocastico

Esistono differenti approcci che permettono di descrivere questo comportamento stocastico, il primo e sicuramente più intuitivo è quello di descrivere un modello stocastico dell'epidemia. Altrimenti è possibile utilizzare un modello stocastico della pandemia per una malattia che può essere comunicata da applicare fintanto che il numero di infetti rimane relativamente piccolo, andando a distinguere due differenti categorie: un *minor disease outbreak* confinato negli stadi iniziali della pandemia e un *major disease outbreak* che occorre nel caso in cui il numero di infetti inizia a crescere esponenzialmente.

Una volta che la pandemia è iniziata, è possibile passare ad un modello di tipo deterministico, in questo caso si parla di **network models**.

2.3 Modelli ad Agente

Un modello ad agente è un modello computazionale per la simulazione delle azioni e interazioni di un insieme di agenti autonomi, siano essi individui o gruppi di individui, con l'obiettivo di comprendere il comportamento del sistema e la relazione che vige con i suoi risultati [82].

Nella modellazione basata su agenti, un sistema viene modellato come un insieme di entità decisionali autonome chiamate *agenti*. Ogni agente valuta individualmente la propria situazione e prende decisioni sulla base di un insieme di regole ben precise. Gli agenti possono eseguire diversi comportamenti appropriati al sistema che rappresentano. Interazioni competitive e ripetute tra agenti sono peculiarità di un modello ad agente che si basa sulla potenza computazionale di un computer per esplorare dinamiche che altrimenti non sarebbero esplorabili tramite la classica modellazione matematica. Nella sua rappresentazione più semplice, un modello ad agente consiste in un sistema di agenti e relazioni tra loro, e questo permette perfino a un modello estremamente semplice di mostrare comportamenti complessi e permette di ottenere informazioni preziose riguardo le dinamiche del mondo reale che modellano.

Generalmente un agente può essere capace di *evolvere* nel tempo, permettendo di osservare comportamenti precedentemente inaccessibili, denominati **comportamenti emergenti**. I modelli più sofisticati incorporano generalmente reti neurali, algoritmi evolutivi o altre tecniche di apprendimento per permettere alla simulazione di essere quanto più realistica possibile.

L'utilizzo di modelli ad agente in epidemiologia è una tecnica nota che da più di un decennio viene impiegata per simulare e comprendere i problemi più disparati, tutti però generalmente accomunati dal fatto che come parametro portante vi sia il comportamento umano [36] [25] [77] [7]. Uno dei parametri più caratterizzanti che da sempre sono stati tenuti in considerazione e assunti come ristretti ad una piccola cerchia, sono le interazioni sociali tra individui. Questo sovrainsieme di parametri racchiude molteplici sottoinsiemi di parametri che descrivono delle interazioni più specifiche ma che possono essere raggruppate come macro categoria se si scende a compromessi.

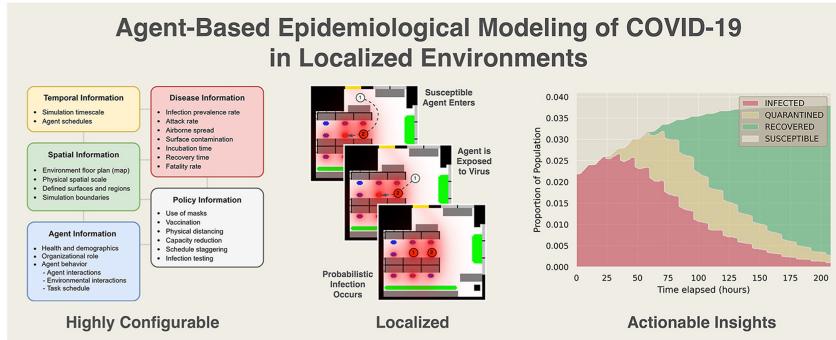


Figure 6: Agent-Based Epidemiological Modeling of COVID-19 in Localized Environment [17]

Questa specifica è importante in quanto uno delle sfide più grandi che il mondo della simulazione, e quindi quello epidemiologico devono affrontare è proprio quello di trovare un modo efficiente e soprattutto realistico di simulare le interazioni sociali tra individui, in quanto queste possono influenzare notevolmente i risultati di una simulazione definendola utile oppure inutile [75]. Non soltanto, un'altra sfida è il modo con cui si decide di rappresentare lo spazio (e il tempo) all'interno della simulazione. In base al tipo di discretizzazione effettuata una simulazione potrebbe essere

utile in un campo ma totalmente inutile in un altro [48].

I vantaggi di utilizzare una tecnica di simulazione tramite modello ad agente sono principalmente i seguenti:

- I modelli ad agente riescono a catturare molto bene i comportamenti emergenti
- I modelli ad agente forniscono una descrizione naturale del sistema che rappresentano
- i modelli ad agente sono *flessibili*

è chiaro come principalmente la scelta dell'utilizzo di un modello di simulazione viene preferito sopra le altre tecniche di simulazione è proprio per via della possibilità di catturare i comportamenti emergenti del sistema.

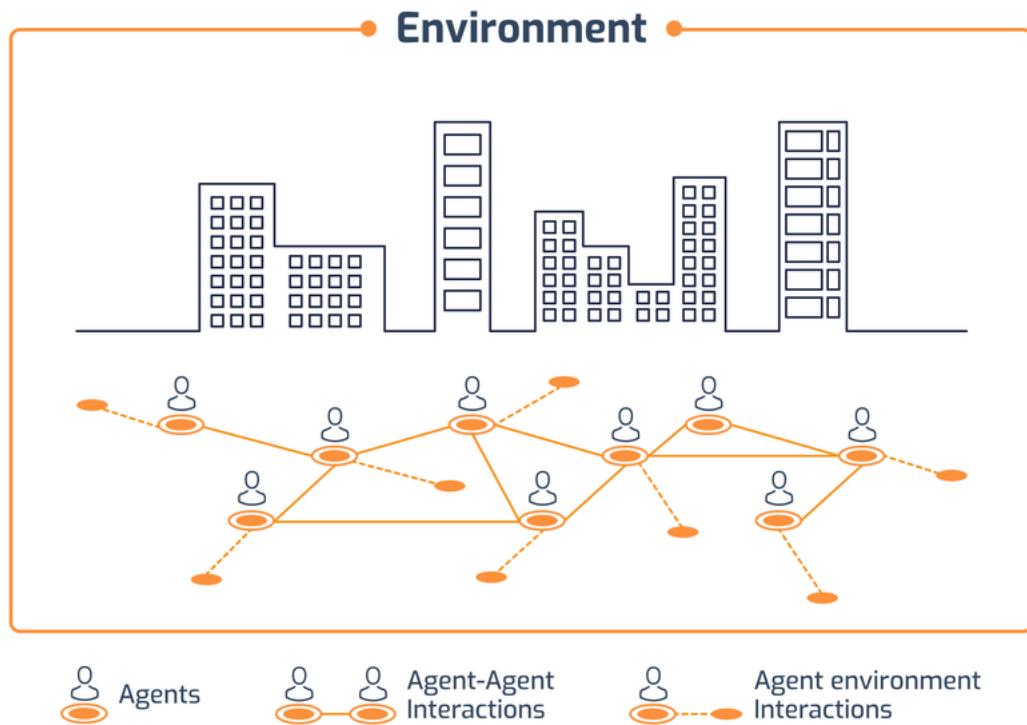


Figure 7: Rappresentazione schematica di un modello ad agente
<https://www.researchgate.net/profile/Marco-Galbiati/publication/356133149/figure/fig1/AS:1098840425922562@1638995381948/Schematic-representation-of-an-agent-based-model-ABM.ppm>

Con l'arrivo della pandemia da COVID-19 molti ricercatori hanno focalizzato la propria attenzione sull'idea di sviluppare un modello ad agente puro oppure ibridato [53] con delle equazioni differenziali, con l'obiettivo di trovare un modello di simulazione in grado di simulare in maniera affidabile il

decorso di una pandemia tenendo in considerazione le variabili più stocastiche e imprevedibili come il comportamento umano, e magari ottenere un intuizione riguardo il comportamento emergente del sistema.

Comportamento emergente

I fenomeni emergenti sono il risultato dell’interazione di entità individuali. Per definizione non possono essere ridotte ad una parte del sistema in quanto l’insieme del sistema è maggiore della somma delle sue singole parti, questo poichè si deve tenere in considerazione le interazioni tra le parti. Un comportamento emergente può avere proprietà che sono separate dalle proprietà intrinseche delle parti. Per questo motivo i comportamenti emergenti rendono complesso lo studio e l’analisi del sistema, in particolare rendono difficile predirne il comportamento, in quanto questi comportamenti possono essere controintuitivi.

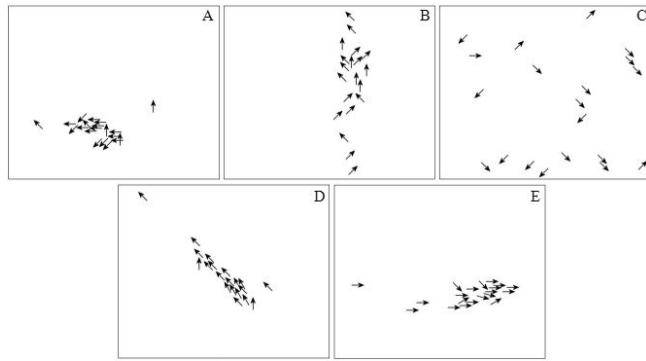


Figure 8: Esempio di comportamento emergente nella simulazione degli stormi di uccelli
è interessante notare come un modello ad agente generi un comportamento emergente dal basso verso l’alto ovvero dai singoli agenti fino all’intero gruppo, e questo alza un interrogativo su cosa possa costituire un comportamento emergente.

2.3.1 Discretizzazione

La tematica della discretizzazione è una delle proprietà fondamentali e al contempo uno dei problemi atavici della simulazione. Il mondo in cui viviamo è un mondo continuo, ma gli strumenti che attualmente abbiamo per simularlo sono discreti, per cui ogni qualvolta che vogliamo simulare un evento dobbiamo decidere in che modo adattare la realtà alla simulazione, andando inequivocabilmente a perdere informazioni nel processo [48].

Il processo di discretizzazione in una simulazione può prendere principalmente due macro aree che sono: lo spazio e il tempo. Molti framework di simulazione implementano al loro interno diversi trucchetti per simulare in maniera quanto più accurata un insieme di dati continuo, permettendo all’utente di utilizzare parole chiave come ad esempio **ContinuousSpace**. Quello che praticamente viene fatto è creare un ambiente per il modello che è il più preciso e granulare possibile in maniera tale da avere una descrizione quanto più accurata (anche se imprecisa) del mondo.

Il processo di discretizzazione comunque non è sempre negativo, in quanto alcuni problemi possono essere simulati in maniera estremamente fedele anche effettuando questi accorgimenti, e anzi alle

volte non è perfino necessario avere una precisione troppo alta per la simulazione di determinati eventi.

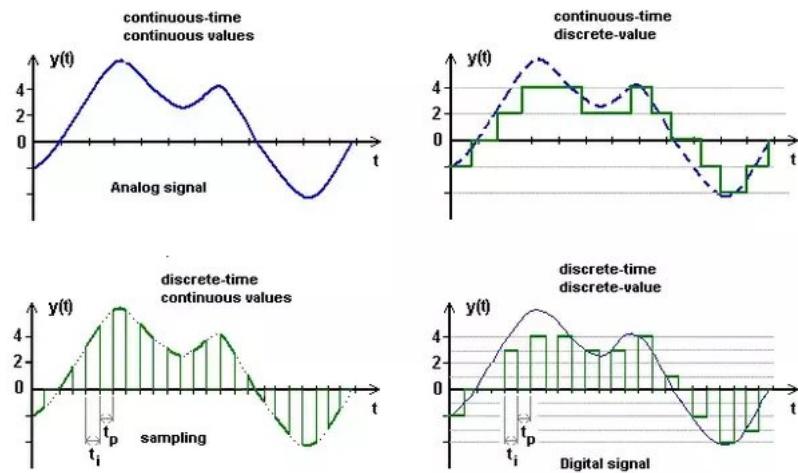


Figure 9: Esempio di differenti tipologie di discretizzazione, siano esse nel tempo o nei valori

2.4 Julia

Julia è un linguaggio di programmazione ad alto livello, multi-paradigma e open-source ideato per compiere analisi numerica ed effettuare operazioni di computer science in maniera rapida e stabile. Julia è nato ufficialmente come linguaggio di programmazione nell'anno 2012 con lo scopo di fornire uno strumento potente, robusto e veloce tanto se non più dei linguaggi considerati in questo ambito lo stato dell'arte, ovvero C e Fortran; ma anche facile da approcciare, al contrario dei linguaggi sopra citati. Julia è un linguaggio di programmazione scritto in C++ e Scheme, ma gran parte della sua composizione è scritta in Julia stesso.

le caratteristiche principali di questo linguaggio sono principalmente:

- Alte performance: lo scopo per cui Julia è nato è stato quello di offrire un linguaggio estremamente performante con la capacità di poter compilare programmi in codice nativo per molteplici piattaforme grazie all'utilizzo di LLVM
- Dinamico: la scelta di rendere Julia un linguaggio dinamicamente tipizzato lo rende di facile utilizzo in quanto rende molto più semplice il suo approccio anche a chi non ha una base solida di programmazione, in quanto ritorna la stessa sensazione di immediatezza di un linguaggio di scripting. Inoltre questo permette un alto supporto per l'uso interattivo
- Ambiente riproducibile: lo scopo del linguaggio è quello di poter permettere all'utente di ricreare le stesse condizioni ogni volta su ogni macchina su cui un programma viene eseguito. Questo può essere ottenuto tramite l'utilizzo di file binari pre compilati
- Componibile: Julia utilizza l'approccio multiple dispatch come paradigma, permettendo una grande flessibilità nell'esprimere una elevata quantità di pattern di programmazione, dall'object-oriented al funzionale
- General Purpose: lo scopo del linguaggio è quello di creare un ecosistema in grado di poter soddisfare qualsiasi esigenza di un utente, permettendo la creazione di applicativi e microservizi senza dover ricorrere ad integrazioni con codice non nativo Julia
- Open source: Julia abbraccia la filosofia open source, e il codice sorgente dell'intero linguaggio, così come di tutte le librerie è disponibile sulla piattaforma GitHub sotto la licenza MIT. Questo permette una crescita eterogenea grazie al contributo di più di 1000 utenti che si impegnano a migliorare il linguaggio

2.4.1 Agents.jl

Seguendo la filosofia propria del linguaggio di programmazione in cui è sviluppata, la libreria Agents.jl [21] viene sviluppata con l'obiettivo di essere facile da imparare e usare ed estendibile, con forte attenzione sulla creazione ed evoluzione di modelli veloci e soprattutto scalabili. Molteplici esempi comparativi sono stati effettuati mostrando come il framework sviluppato permetta di avere un notevole guadagno prestazionale rispetto ai maggiori competitor attualmente presenti sul mercato (Mesa, NetLogo, MASON) [1].

	Extreme-scale	Repast HPC	MATSIM, PDES-MAS, Swarm
Computational Modelling Strength / Models' Scalability Level	High / Large-scale	Altrevia Adaptive Modeler, SeSAM	AnyLogic (2D/3D), AOR Simulation, CloudSim, CybelePro, FLAME, LSD (2D/3D), MASS, Pandora, UrbanSim
Medium-scale	NetLogo (2D/3D)	Ascape, CRAFTY, GAMA (2D/3D), SimEvents (MATLAB®), Simio (2D/3D), Simul8 (2D/3D)	MASON (2D/3D)
Light-weight / Small-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	
	AgentScript, Framsticks (2D/3D), JAMEL, JCASim (1D/2D/3D), jES, MOBIDYC, PedSim, PS-I, Scratch (2D/3D), Simlr, SimSketch, SOARS, StarLogo, StarLogoTNG (3D), Sugarscape, VisualBots	SEAS (2D/3D)	
	Simple/Easy	Moderate	Complex/Hard
	Model Development Effort →		

Figure 10: Tabella comparativa [1]

La facilità di interazione con questa libreria non è da confondersi con una mancanza di opzioni durante lo sviluppo, in quanto nativamente Agents.jl permette l'integrazione con altre librerie che in maniera altrettanto semplice e veloce offrono all'utente la possibilità di addentrarsi nel mondo

del machine learning, in particolar modo il mondo del Scientific Machine Learning [67], branca che soprattutto grazie alla pandemia da Covid-19 ha visto un sempre più crescente interesse.

Agents.jl offre molteplici opzione di configurazione, ma principalmente quello su cui si basa sono i seguenti principi:

- definizione di un tipo di agente, generalmente viene raccomandato di estendere la tipologia *StandardABM* la quale è la più concreta implementazione, nonchè l'implementazione di default, di un costruttore generico di un **AgentBasedModel**.
- definizione di una tipologia di spazio, esistono principalmente 2 tipologie di spazio da poter utilizzare come base e si basano sull'utilizzo di uno spazio *discreto* oppure *continuo*.
 - spazio discreto a grafo: un *GraphSpace* rappresenta uno spazio del modello rappresentato da un grafo arbitrario in cui ogni nodo può contenere una quantità di agenti arbitraria. Per funzionare correttamente questa tipologia di spazio richiede che gli agenti implementino al loro interno specifici attributi per rappresentare la loro posizione all'interno dello spazio. Questa tipologia di spazio si appoggia alla libreria **Graphs.jl** [26] per gestire tutte le operazioni relative alla struttura dati del grafo.

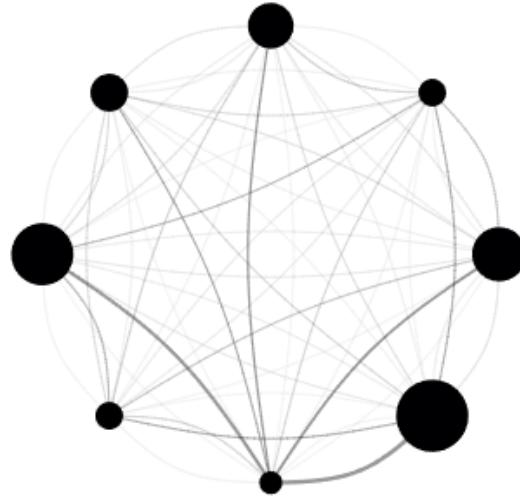


Figure 11: Esempio di rappresentazione di uno spazio a grafo con archi pesati

- spazio discreto a griglia: un *GridSpace* rappresenta uno spazio del modello rappresentato da una griglia di dimensione $D \geq 1$. Questa tipologia di spazio richiede l'utilizzo di una metrica per la definizione della distanza tra celle di una griglia. Ci sono attualmente tre tipologie di metriche supportate e sono: *Euclidean*, *Manhattan* e *Chebyshev*.

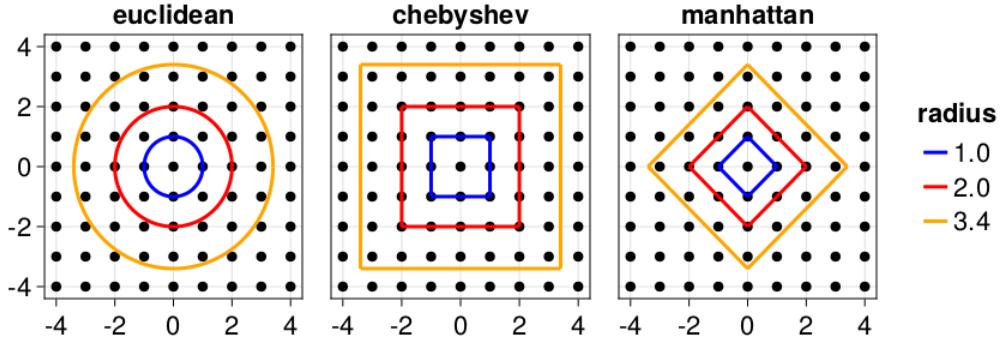


Figure 12: Esempio di metriche per il calcolo della distanza su uno spazio a griglia

- spazio continuo: un *ContinuousSpace* rappresenta uno spazio di dimensione $D \in (0, \infty)$. è fortemente consigliato di attribuire ad un agente all'interno di questo spazio due caratteristiche fondamentali, una posizione e una velocità. Questa tipologia di spazio permette di rappresentare delle proprietà spaziali tramite valori finiti oppure tramite *funzioni*, i quali rappresentano una discretizzazione di valori spaziali che potrebbero non essere disponibili in maniera analitica. Utilizzando questa tipologia di spazio la metrica di distanza utilizzata sarà sempre *Euclidian*. Per velocizzare il calcolo della posizione degli agenti, viene effettuata una discretizzazione implicita dello spazio, ma questa può essere forzata a rimanere nello spazio continuo ottenendo un calo di prestazioni.
- spazio misto: un *OpenStreetMapSpace* rappresenta una mappa come un'entità continua che preferisce l'accuratezza alle prestazioni. La mappa viene rappresentata come un grafo connesso. I nodi non rappresentano necessariamente intersezioni.

2.4.2 SciML.ai

SciML.ai è una collezione di librerie dedite all'analisi numerica e al calcolo scientifico. Questo framework permette di avere tutti gli strumenti per poter utilizzare facilmente, velocemente e in maniera robusta tecniche di analisi numerica molto avanzata, così da poter sviluppare applicazioni complesse in maniera semplice e concreta [67] [64] [66].

Il principale utilizzo che è stato fatto di queste librerie si concentra principalmente sull'implementazione di metodi di analisi numerica, come ad esempio l'utilizzo di risolutori per sistemi di *Equazioni Ordinarie Differenziali* (ODE) che si possono trovare nel package *OrdinaryDiffEq.jl* [67] uniti a metodi di *Machine Learning* (ML) [59] [41] [42] per lo sviluppo di un modello di *Scientific Machine Learning* che si appoggia sui costrutti denominati **Neural Differential Equation** (Neural DE) e **Universal Differential Equation** (UDE) [64] [66].

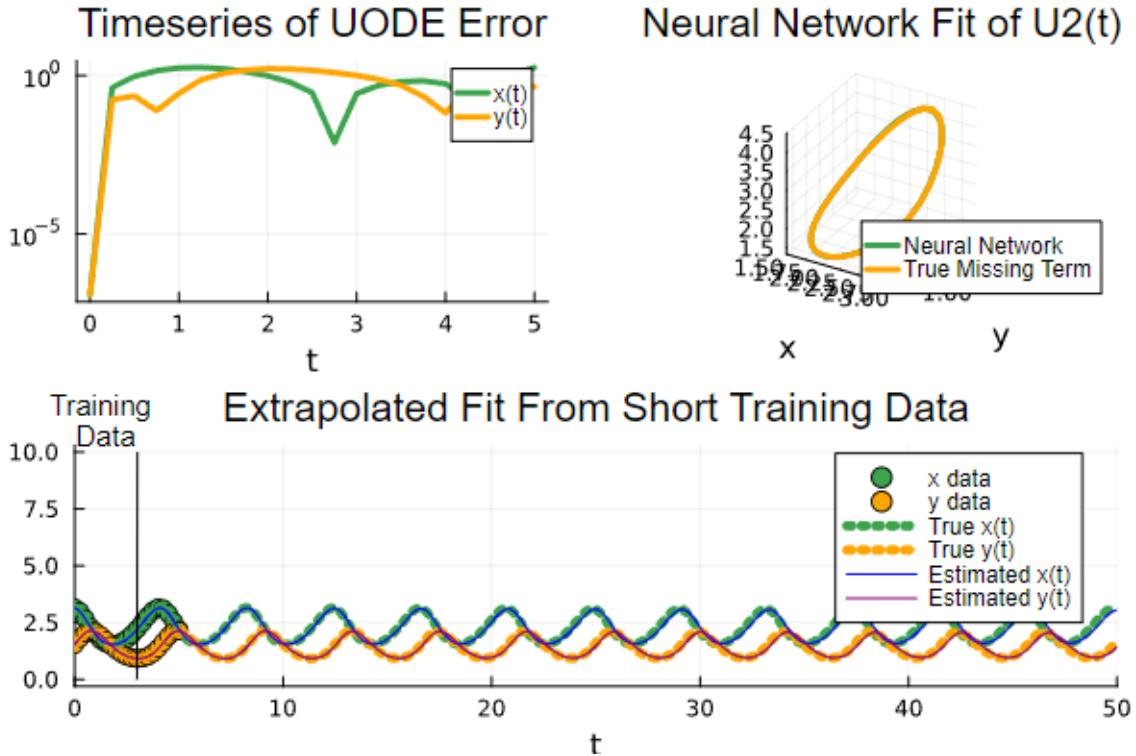


Figure 13: Esempio applicazione UODE e Neural ODE [63] [66]

Equazioni Neurali Differenziali

Da quando l'articolo [16] è stato pubblicato questa tecnica ha ricevuto molta attenzione facendo sì che si iniziasse ad ibridare due paradigmi di modellazione come le ODE e le reti neurali (NN) per cercare di ottenere il massimo da entrambe minimizzando gli effetti indesiderati. [47] [16]

Un'equazione differenziale è un modo per specificare una trasformazione non lineare arbitraria codificando matematicamente le ipotesi strutturali a priori del sistema. A riguardo esistono 3 approcci comuni per definire una trasformazione non lineare:

- **modellazione diretta**
- **machine learning**
- **equazioni differenziali**

Il primo approccio, ovvero la modellazione diretta, funziona solamente se si conosce l'esatta funzione che mette in correlazione l'input con l'output. Tuttavia nella generalità dei casi questa relazione non è conosciuta a priori per cui non è possibile applicare questo approccio. A tal proposito si può utilizzare l'approccio del machine learning.

In un generico problema di machine learning dato un insieme di dati x si vuole predire un insieme y di dati correlati da una funzione sottostante. Questa generazione di dati y da x è generalmente

definito *modello*. L'idea alla base è avere un periodo di allenamento (training) in cui si tenta di aggiustare i parametri (iperparametri) del modello così da assicurarsi di avere un modello in grado di generare previsioni accurate. Successivamente questo modello verrà utilizzato per effettuare inferenza su dati x mai visti. Questo approccio è semplicemente un insieme di trasformazioni non lineari.

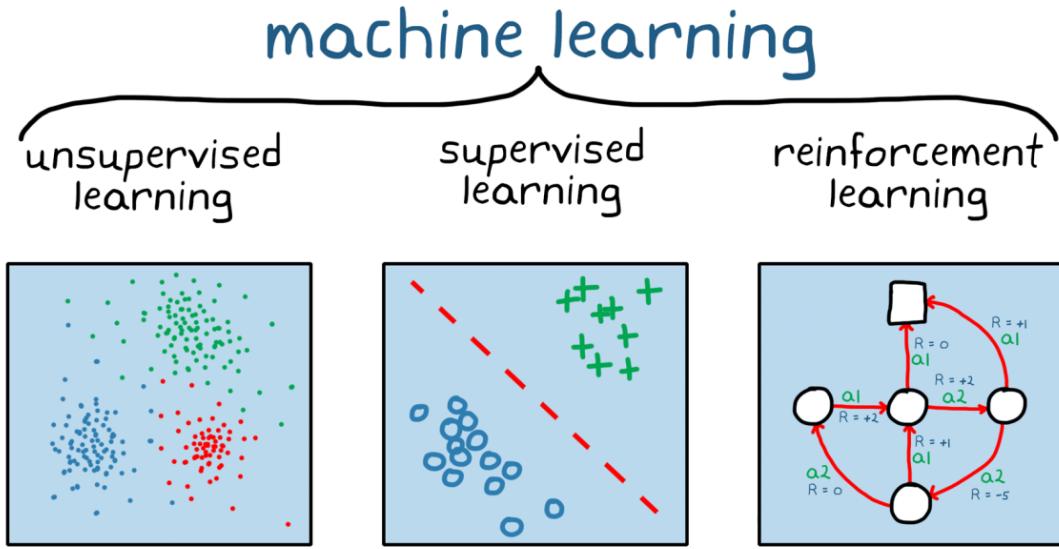


Figure 14: Differenti approcci di apprendimento di machine learning
https://it.mathworks.com/discovery/reinforcement-learning/_jcr_content/mainParsys3/discoverysubsection/mainParsys/image.adapt.full.medium.png

L'utilizzo del machine learning è molto interessante in quanto il concetto alla base è estremamente semplice ma è molto potente in quanto riesce ad adattarsi ai dati forniti in maniera molto buona. Questa soluzione si espande successivamente all'idea di rete neurale (o Neural Network, NN) la quale non è altro che un insieme di moltiplicazioni tra matrici seguite dall'applicazione di una funzione di attivazione. Per esempio una semplice rete neurale da 3 strati di profondità viene 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 dei parametri apprendibili. Successivamente lo scopo è quello di scegliere W tale che $ML(x) = y$ si comporti in maniera ragionevole simile alla funzione incognita che vogliamo adattare. Grazie all'applicazione del teorema di approssimazione universale afferma che per un numero sufficientemente grande di parametri o di strati è possibile approssimare qualsiasi funzione non lineare in maniera sufficientemente precisa.

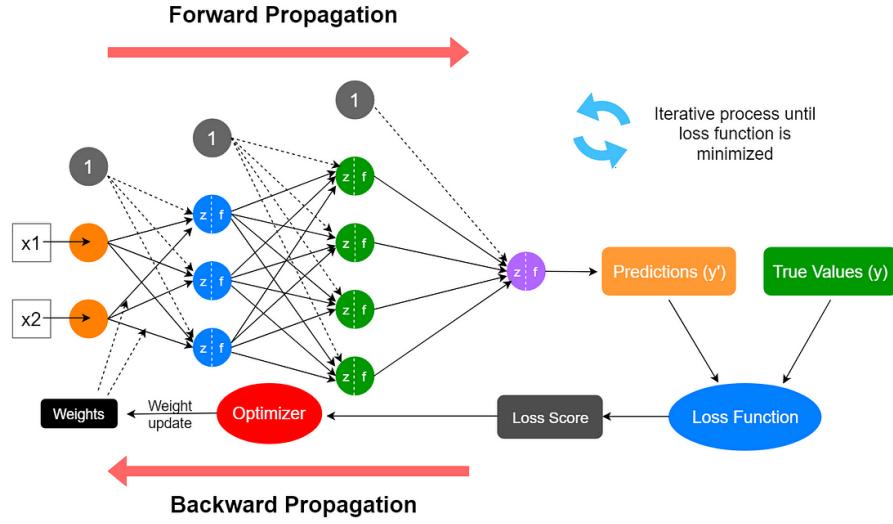


Figure 15: Esempio di funzionamento di una rete neurale
https://miro.medium.com/v2/resize:fit:4800/format:webp/1*ZXAOUqmlyECgfvA81Sr6Ew.png

Questo approccio tuttavia necessita di apprendere ogni aspetto della trasformazione non lineare direttamente dai dati a disposizione. In molti casi tuttavia, non è possibile conoscere l'intera equazione non lineare, ma magari se ne conosce la struttura generale. Il modo per definire matematicamente questo tipo di approccio è tramite le equazioni differenziali. Il modo più immediato è quello di definire un modello matematico in cui si tenta di apprendere una costante associata all'andamento di un insieme di dati.

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

Tramite questo approccio non è necessario conoscere la soluzione dell'equazione differenziale per convalidare che il modello sia corretto. Infatti la struttura del modello e la matematica stessa viene codificata all'interno del modello stesso il quale produce successivamente una soluzione. Questa tipologia di modelli è essenzialmente un insieme di equazioni che descrive il cambiamento delle cose e dove queste si troveranno ad un determinato periodo di tempo è la soluzione all'equazione differenziale.

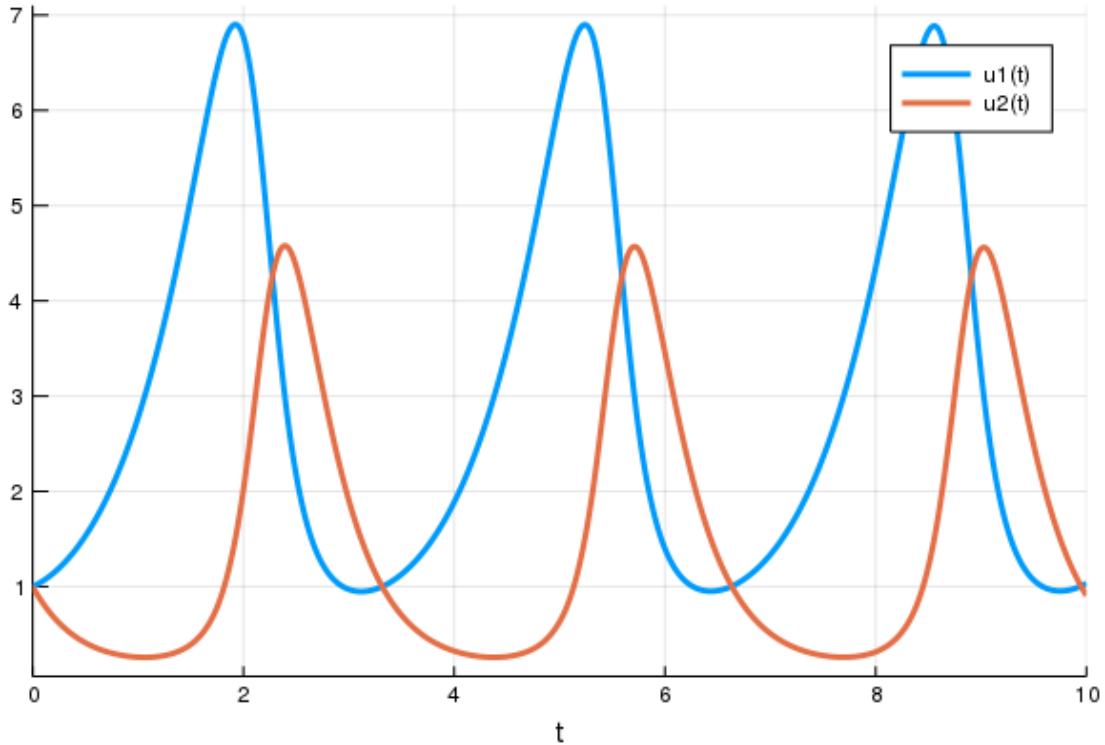


Figure 16: Esempio grafico ODE rappresentante la famosa equazione di Lotka-Volterra

Questo metodo è stata la soluzione utilizzata prevalentemente nel mondo della scienza e recentemente ha visto il proprio utilizzo ulteriormente espanso con l'ibridazione tramite strutture matematica aggiuntive che hanno permesso di modellare sistemi complessi come ad esempio la modellazione di sistemi farmacologici o biologici.

La peculiarità dei modelli di machine learning è che sono affamati di dati e richiedono un insieme di dati su cui allenarsi molto grande, e per questo l'utilizzo delle equazioni differenziali è divenuto un'opzione molto interessante per specificare la nonlinearietà in maniera apprendibile (tramite parametri) in maniera limitata. Questo permette principalmente di incorporare una conoscenza specifica di un dominio nelle relazioni strutturali tra input e output del modello.

Una equazione differenziale neurale è uno dei metodi per mettere in relazione questi due mondi: quello del machine learning e quello delle equazioni differenziali. L'approccio generale è quello di apprendere non tanto la trasformazione non lineare che vi è tra i dati, quanto la struttura della trasformazione non lineare. Perciò invece che avere il modello $y = ML(x)$ avremo il modello $y' = ML(x)$ e si tenta successivamente di risolvere l'equazione differenziale associata.

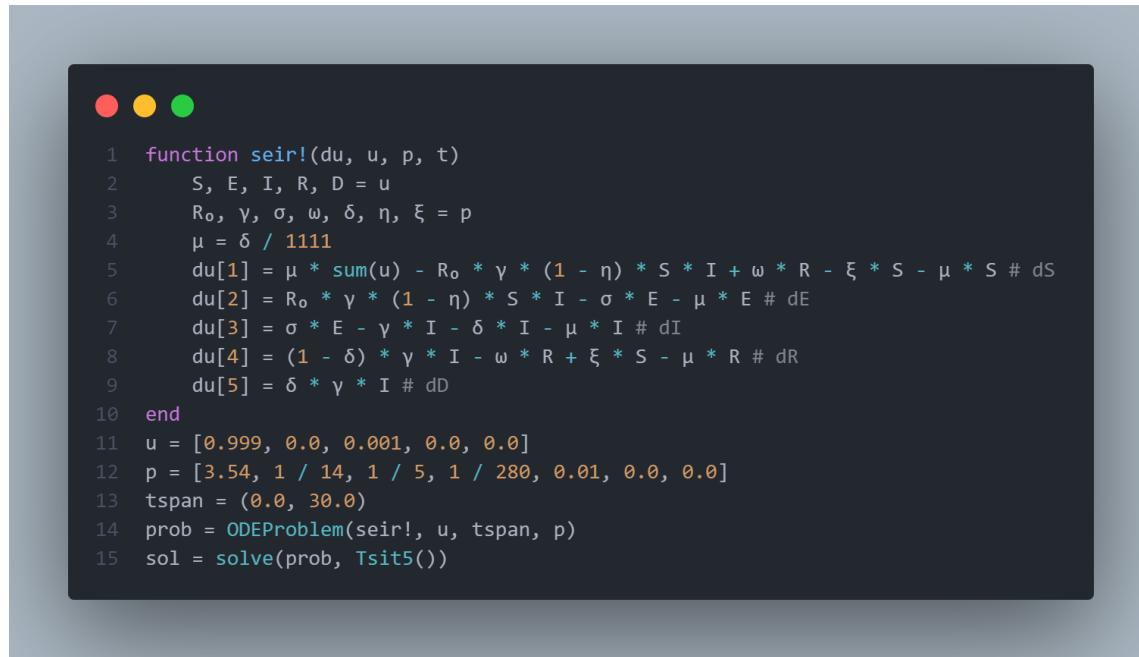
Principalmente il motivo per cui si vuole utilizzare questa tipologia di approccio è quella per cui, definendo un modello in questo modo e utilizzando un risolutore estremamente semplice e prono ad errori come il metodo di Eulero si ottengono risultati equivalenti ad una ResNet [39]. L'idea alla base è quella che invece di modellare una rete neurale con sempre più strati, e quindi sempre più

profonda, è sufficiente modellare il sistema di equazioni differenziali direttamente e successivamente risolverla tramite un risolutore specifico.

Questo tipo di approccio è *memory efficient*, ha la capacità di gestire *dati irregolari* con forti priori sullo spazio del modello, ha un elevata capacità di approssimare funzioni lineari e non lineari e si poggia su solide basi teoriche che pesca da entrambi i lati.

Risolvere una ODE in Julia

L'idea alla base è quella di definire un oggetto **ODEProblem** tramite la definizione di una funzione che ha al suo interno la descrizione del sistema di ODE tramite la specificazione delle derivate delle equazioni nella forma $= f(u, p, t)$, provvedendo a fornire un insieme di condizioni iniziali u_0 , un periodo 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 17: Esempio definizione ODE in Julia

Successivamente è possibile risolvere il sistema di ODE tramite la chiamata della funzione **solve**

Si possono specificare molteplici metodi per la risoluzione del sistema, in questo caso il metodo **Tsit5** [78] è un metodo di quinto ordine esplicito di tipologia **Runge-Kutta** con uno stimatore di errore integrato di tipo **Tsitouras**. L'interfaccia proposta dal framework DifferentialEquations.jl permette di definire molteplici parametri aggiuntivi, utili per un approccio più capillare.

Inserire una ODE all'interno di una NN

Per comprendere meglio cosa significa inserire una ODE all'interno di una NN, bisogna osservare come un layer di una NN è definito. Un layer è sostanzialmente una *funzione differenziabile* che prende come input un vettore di dimensione n e resituisce un nuovo vettore di dimensione m .

Appare come i risolutori di DE rientrano anche loro nella categoria di funzioni differenziabili, il che significa che è possibile inserirli direttamente all'interno di un programma differenziabile più grande. Questo programma può essere nel nostro caso una rete neurale.



```

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 18: Esempio implementazione di una rete neurale in Julia

Successivamente è possibile addestrare per un determinato numero di integrazioni la nostra rete neurale per ottenerne i risultati.

è importante puntualizzare come il sistema non stia apprendendo una soluzione all' equazione differenziale. Piuttosto ciò che sta imparando è il sistema di ODE da cui la soluzione è generata. In questo caso la Neural ODE impara una rappresentazione compatta di come la serie di dati

si comporta nel tempo, e può facilmente estrapolare cosa potrebbe succedere con differenti condizioni iniziali. La suite **DiffEqFlux.jl** [16] permette di avere un comodo wrapper per definire una **NeuralODE**.

```

● ● ●
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 19: Esempio implementazione NeuralODE tramite DiffEqFlux.jl

Backpropagation tramite ODE solver

Il cuore di ogni rete neurale è l'abilità di propagare all'indietro lungo tutta la rete le derivate, con lo scopo di calcolare il gradiente della funzione di perdita rispetto ai parametri della rete. Perciò la sfida diventa trovare un metodo per applicare lo stesso comportamento anche quando si utilizzano dei risolutori ODE all'interno della rete neurale.

Esistono svariati approcci, il più comune è tramite un approccio di analisi di sensitività (adjointed). L'analisi di sensitività definisce una nuova ODE la cui soluzione fornisce il gradiente per la funzione di costo rispetto ai parametri, e successivamente viene risolta questa seconda ODE.

Equazioni Ordinarie Differenziali

Nell'ambito matematico, una *equazione ordinaria differenziale* (ODE) è un'equazione differenziale (DE) dipendente da un singolo valore indipendente, generalmente il tempo. All'interno di questa grande famiglia di equazioni, il gruppo delle *equazioni lineari differenziali* gioca un ruolo predominante in quanto la maggior parte dei fenomeni fisici e di matematica applicata possono essere descritti dalla soluzione di questo tipo di equazioni.

Una equazione lineare differenziale è definita da un *polinomio lineare* e la sua derivata è un'equazione della forma:

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

dove $\alpha_0(x), \dots, \alpha_n(x)$ e $b(x)$ sono funzioni differenziabili arbitrarie che non richiedono di essere lineari, e $y', \dots, y^{(n)}$ sono le successive derivate della funzione incognita y della variabile x .

L'utilizzo di *equazioni non lineari differenziali* può essere generalmente approssimato con la controparte lineare così da ottenere una soluzione più semplice.

La suite di SciML.ai offre un'ampia varietà di framework per la risoluzione di sistemi di equazioni lineari differenziali i quali si possono prevalentemente trovare all'interno della libreria *DifferentialEquations.jl* [67] [68] [51] [35], i risolutori sono molteplici e permettono grande dinamicità e affidabilità nonché elevate prestazioni durante l'utilizzo, questo in relazione anche ai risolutori dei più conosciuti linguaggi di programmazione come ad esempio: **MATLAB**, **SciPy**, **R**, **C** e **C++ e altri**.

La figura riportata sotto 20 [62] mostra una comparativa tra le varie implementazioni dei metodi di risoluzione di sistemi di ODE di vario genere utilizzati dalla maggior parte dei linguaggi di programmazione focalizzati sull'analisi numerica e scientifica.

Comparison Of Differential Equation Solver Software														
Subject/Item	MATLAB	SciPy	deSolve	DifferentialEquations.jl	Sundials	Homer	ODSPACK/Netlib/NAG	JCODE	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	Fair	Excellent	Good	Fair	Good	Poor	Poor	Fair	Poor	Fair	Fair	Fair
Efficiency*	Poor	Poor***	Poor**	Excellent	Good	Good	Good	Good	Good	Fair	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	Fair	None	None	None	Good	Good
Symbolic Calculation of Jacobians/Autodifferentiation	None	None	None	Excellent	None	None	None	None	None	None	None	None	Excellent	Excellent
Complex Numbers	Excellent	Good	Fair	Good	None	None	None	None	None	None	Good	Excellent	Excellent	Excellent
Arbitrary Precision Numbers	None	None	None	Excellent	None	None	None	None	None	None	Excellent	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	None	Excellent	Excellent	Good	Depends on the solver	None	None	None	None	Fair	None	None
Built-in Parallelism	None	None	None	Excellent	Excellent	None	None	None	None	None	None	Fair	None	None
Differential-Algebraic Equation (DAE) Solvers	Good	None	Good	Excellent	Good	Excellent	Good	None	Fair	Good	None	None	Good	Good
Implicitly-Defined DAE Solvers	Good	None	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	Good	None	None
Constant-Log Delay Differential Equation (DDE) Solvers	Fair	None	Poor	Excellent	None	Good	Fair (via DDVERK)	Fair	None	None	None	Good	Excellent	Excellent
Stochastic Differential Equations (SDE) Solvers	Poor	None	Poor	Excellent	None	Excellent	Good	None	None	None	None	None	Excellent	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	None	None	Excellent	None	None	None	Good	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonian Systems (e.g., Symplectic Integrators)	None	None	None	Excellent	None	Good	None	None	None	None	None	Fair	Good	None
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	Good	Fair	None
GPU Compatibility	None	None	None	Excellent	Good	None	None	None	None	None	None	Good	None	None
Analysis Add-ons (Sensitivity Analysis, Parameter Estimation, etc.)	None	None	None	Excellent	Excellent	None	Good (for some methods like DASKR)	None	Fair	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														
Score	None	Poor	Fair	Good	Excellent									
Explanation	Functionality does not exist	Functionality exists, but is incomplete	The basic features exist	The basic features exist and some extra tweakability exists. May include extra methods or efficiency.	The basic features exist and some extra tweakability exists. May include extra methods or efficiency.									

Figure 20: Tabella comparativa tra le varie implementazioni dei vari risolutori [62]

Equazioni Differenziali Universali

Recentemente gli avanzamenti nel mondo del machine learning sono stati dominati dalle tecniche di deep learning le quali necessitano di una quantità di dati enormi, generalmente chiamati *big data*, per risolvere problemi definiti precedentemente come difficili e complessi, come ad esempio problemi di *image recognition* o *natural language processing*.

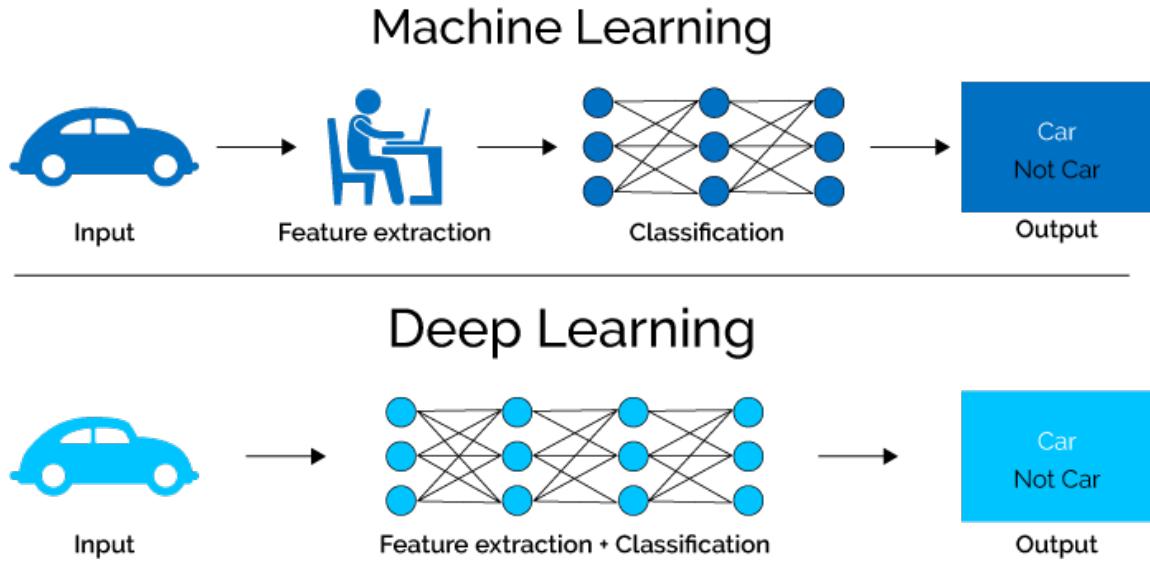


Figure 21: Esempio comparativo tra funzionamento Machine Learning e Deep Learning

Tuttavia se molteplici ambiti della scienza sono riusciti a generare una mole di dati sufficiente per l'utilizzo di queste tecniche, alcuni ambiti specialmente legati alla medicina e a tutto ciò che ne concerne, faticano estremamente ad avere un insieme di dati sufficientemente grande e variegato per applicare le tecniche sopra introdotte. In questo ambito i modelli meccanicistici rimangono quelli più utilizzati. L'utilizzo di questi modelli tuttavia impone dei limiti quali l'essere modelli predittivi che dipendono fortemente dalle conoscenze pregresse del modello, andando a creare quello che può essere assimilabile ad una specie di bias induttivo, mentre l'approccio data-driven dei modelli di machine learning può consentire di essere molto più flessibile permettendo di abbandonare le ipotesi semplificative necessarie per derivare i modelli teorici.

Un *equazione differenziale universale* (UDE) è un insieme di equazioni differenziali definiti pienamente, o in parte, da un *approssimatore universale*. Un approssimatore universale è un oggetto parametrico capace di rappresentare qualsiasi funzione dato una determinata dimensione dei parametri. Degli approssimatori universali, in uno spazio a basse dimensioni includono la serie di Fourier o di Chebyshev, mentre un approssimatore universale in uno spazio ad alte dimensioni includono le reti neurali e altri modelli utilizzati all'interno del machine learning. Matematicamente parlando però, nella sua forma più generale, una UDE è una equazione differenziale parziale (PDE) a ritardo stocastico forzato definita con approssimatori universali incorporati:

$$\mathcal{N}[u(t), u(\alpha(t)), W(t), U_\theta(u, \beta(t))] = 0$$

dove $\alpha(t)$ è una funzione a ritardo e $W(t)$ è un processo Wiener.

Un *equazione differenziale universale* (UDE) è una *equazione algebrica differenziale* non triviale, ovvero un sistema di equazioni che contiene delle equazioni differenziali ed equazioni algebriche oppure è un sistema equivalente, con la proprietà che la sua soluzione può approssimare *qualsiasi* funzione continua su un qualunque intervallo $\in R$ a qualsiasi livello di precisione desiderata.

Per essere precisi, una equazione differenziale (possibilmente in forma implicita) $P(y', y'', y''', \dots, y^{(n)}) = 0$ è una UDE se, per ogni funzione a valori relative continua f e per ogni funzione continua positiva ϵ esiste una soluzione liscia (una funzione è considerabile liscia se è differenziabile in ogni suo punto, perciò continua) y di $P(y', y'', y''', \dots, y^{(n)}) = 0$ con $|y(x) - f(x)| < \epsilon(x) \forall x \in R$.

Il concetto di UDE può essere analogo all'idea di una *Macchina di Turing Universale* con la differenza che le UDE non dettano l'evoluzione di un sistema, ma si limitano a imporre determinate regole che ogni sistema che si evolve deve soddisfare. Questo permette di avere un modello robusto per l'analisi di dati e la predizione dell'interazione che hanno vari fenomeni tra loro.

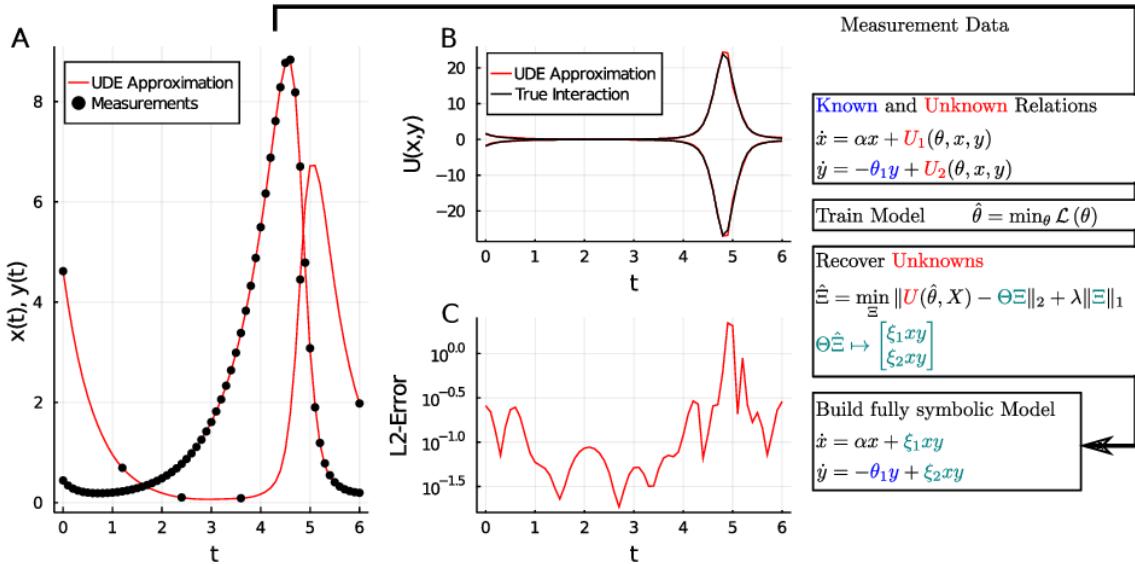


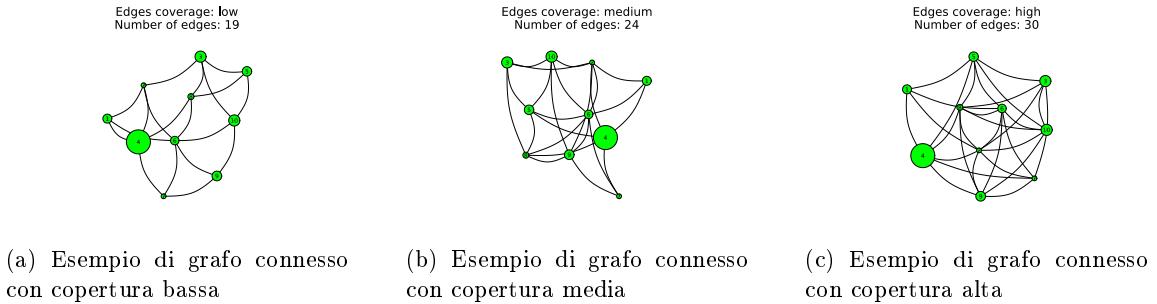
Figure 22: Comportamento UDE nell'approssimazione di fenomeni non lineari [66]

Questo approccio viene spesso unito a tecniche *Data-Driven* [45] per l'identificazione sparsa di dinamiche non lineari. In particolare uno degli approcci utilizzati è quello tramite l'algoritmo *SINDy* (*Sparse Identification of Non-linear Dynamics*). Questo algoritmo performa una serie di operazioni di regressione come ad esempio **LASSO** su una libreria di funzioni candidate non lineari ottenute da uno snapshot del sistema dinamico che si sta analizzando e delle sue derivate, con l'obiettivo di trovare le equazioni che lo governano. Questo procedimento si basa sull'assunzione che molti sistemi fisici hanno solamente una manciata di termini che ne dettano le dinamiche e l'evoluzione. Questo metodo è stato largamente utilizzato nell'identificazione della *dinamica dei fluidi* così come nelle *reti biologiche* e altri sistemi dinamici complessi.

3 Metodi e Modelli

L'approccio utilizzato è stato quello di modellare l'intero problema come se fosse una *rete sociale*.

Il problema si presenta come lo studio della diffusione di una pandemia a trasmissione virale in un ambiente reale, il quale è composto principalmente di così detti *Punti di Interesse* (PdI). Questi punti di interesse successivamente sono collegati tra loro tramite una rete più o meno fitta di collegamenti. Questa costruzione ricorda in maniera molto stretta una struttura dati largamente utilizzata, ovvero il **Grafo**.



Con questo approccio la modellazione del problema e soprattutto la dinamica intera del modello verte più su un approccio di tipo *mesoscopico* tendente al *macroscopico*. Questo poichè per quanto l'idea di modellare un sistema ad agente estremamente granulare fosse di non poco interesse, al lato pratico ci si sarebbe scontrati con delle difficoltà intrinseche ad una modellazione così specifica, la quale non si adattava all'idea più ampia del problema. Difatti non si vuole modellare un sistema a livello microscopico per vedere le possibili interazioni tra agenti differenti, bensì si vuole vedere la risposta collettiva di un agente astratto all'utilizzo di interventi mirati e specifici per contrastare l'epidemia in maniera localizzata.

3.1 Approccio con Rete Sociale

Il modello utilizzato sfrutta le proprietà del framework **Agents.jl** e definisce un modello ad agente di tipo **ABM**. Essendo che in questo caso la struttura spaziale del modello non è importante ma le sue connessioni sì, vengono definiti gli agenti come nodi veri e propri del grafo in cui al loro interno viene simulato il ciclo di vita della pandemia tramite un modello di tipo **SEIR** deterministico.

La struttura dell'agente è di tipo **ContinuousAgent** in quanto lo spazio del modello è di tipo continuo. Questa scelta è stata fatta in vista di possibili sviluppi futuri dell'applicazione. Successivamente si può osservare come il modello generato in figura 26 sia molto snello, avendo solamente una manciata di parametri sufficienti al corretto funzionamento del modello stesso.

Grafo sociale

Un grafo sociale è un particolare tipo di grafo che rappresenta le relazioni sociali tra entità. Questo approccio viene largamente utilizzato per la rappresentazione delle reti sociali dove la parola *grafo* viene presa dalla teoria dei grafici. Generalmente un grafo sociale viene riferito come un grafo che mappa gli individui come nodi e le loro relazioni come archi.

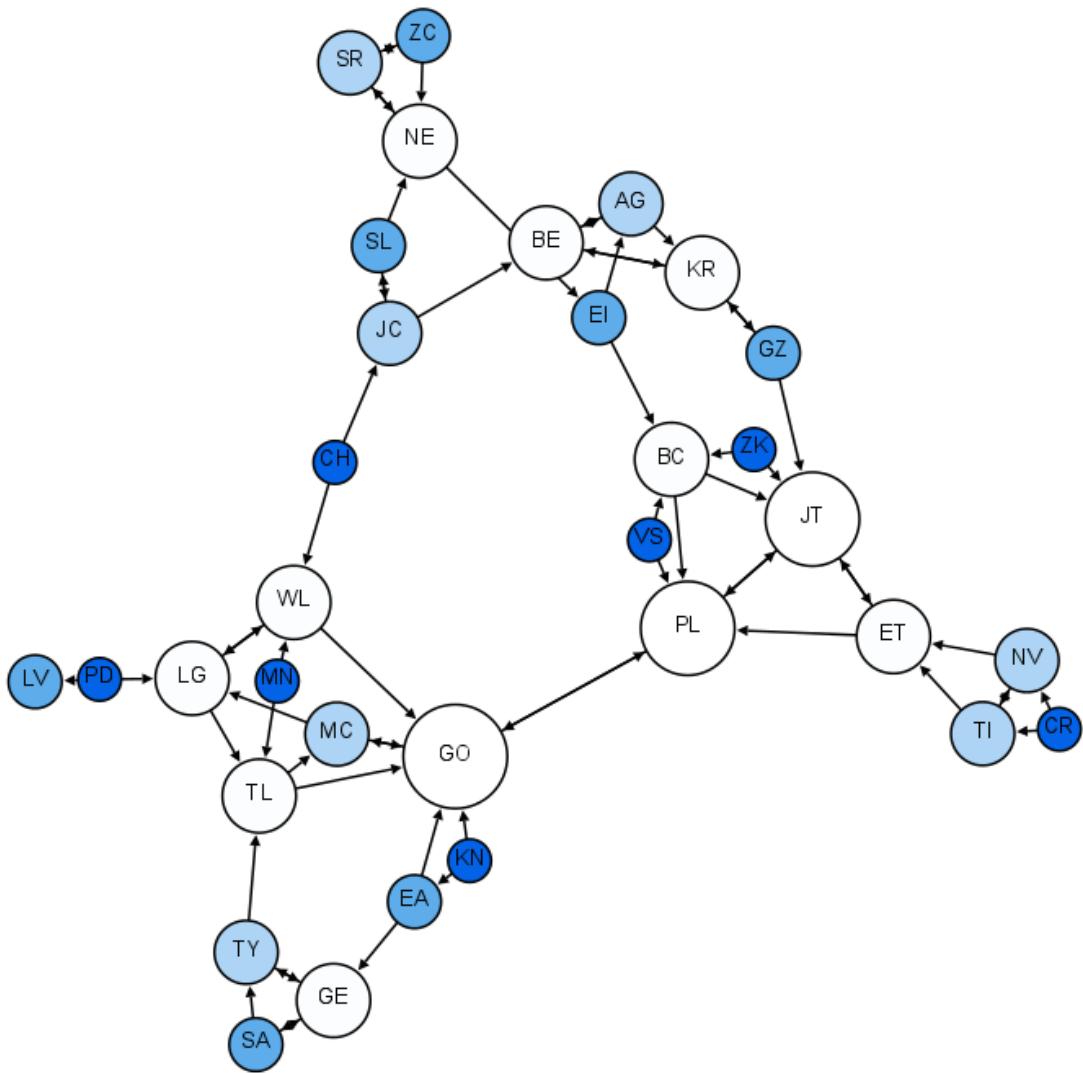
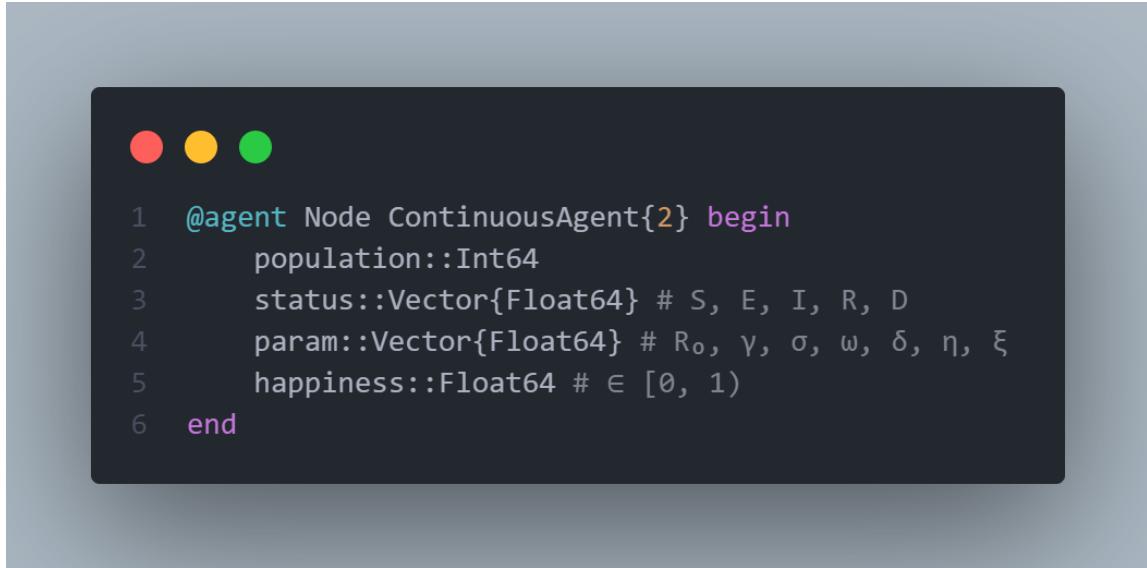


Figure 24: Esempio di grafo sociale
[https://en.wikipedia.org/wiki/Sociogram#media/File:
 Moreno_Sociogram_3rd_Grade.png](https://en.wikipedia.org/wiki/Sociogram#media/File:Moreno_Sociogram_3rd_Grade.png)

Il concetto è stato originariamente coniato dalla struttura dati **sociogramma**, una struttura a grafo che rappresenta i collegamenti sociali che hanno gli individui da esso modellato, ovvero le persone. Questa struttura dati si occupa di strettare le relazioni interpersonali che si formano all'interno di un gruppo di individui.

3.1.1 Agente



```
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 25: Codice Agente

Come mostrato in figura 25 l'agente al suo interno è molto minimale, descrivendo solamente gli attributi necessari e sufficienti per la corretta modellazione e interazione con l'ambiente, il modello e gli altri agenti, e se stesso. L'evoluzione di ogni agente è completamente indipendente dall'evoluzione degli altri, ma può essere influenzata da altri agenti della rete, modificando l'altrimenti determinismo del modello SEIR al suo interno.

I campi principali dell'agente sono i seguenti:

- **population**: questo campo descrive tramite un valore di tipo **Int** il numero totale di individui che ad ogni step del modello si trovano all'interno di uno specifico nodo.
- **status**: questo campo descrive tramite un vettore di tipo **Float64** lo stato della popolazione all'interno del nodo. Questo stato viene descritto come percentuale di individui, per cui i valori all'interno del vettore saranno tutti compresi tra 0 e 1. La scelta di utilizzare questo approccio è nata principalmente per motivi di *type stability* in quanto le operazioni di risoluzione del sistema di ODE altrimenti potrebbero risentirne. Tuttavia è stato fatto anche per un fattore di *facilità di comprensione*.
- **param**: questo campo descrive tramite un vettore di tipo **Float64** i parametri del modello SEIR specifici del nodo in questione. Questi parametri sono comprensivi non solo dei valori relativi all'epidemia, ma anche dei valori relativi alle contromisure, in particolare le contromisure **non farmaceutiche** rappresentate dal valore η e quelle **farmaceutiche** rappresentate dal valore ξ .
- **happiness**: questo campo è utilizzato come campo di appoggio per il bilanciamento delle contromisure applicate dal controllore.

3.1.2 Spazio e Modello



```

● ● ●

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

```

Figure 26: Codice Modello

Come accennato precedentemente e mostrato in figura 26, lo spazio del modello è di tipo **ContinuousSpace** seppure non venga effettivamente sfruttato a dovere. Tuttavia l'interesse del modello è il fatto che utilizzi una struttura come un grafo in maniera astratta per modellare le interazioni tra i suoi nodi, ovvero gli agenti, in maniera rapida ed efficiente.

Per prima cosa viene creato il modello tramite la creazione di un grafo *connesso*, il cui numero di archi dipende da quanto l'utente vuole che siano connessi i nodi. Più alto è il desiderio di copertura più archi verranno creati. Tuttavia ci sono dei limiti superiori e inferiori sul numero di nodi creabili.

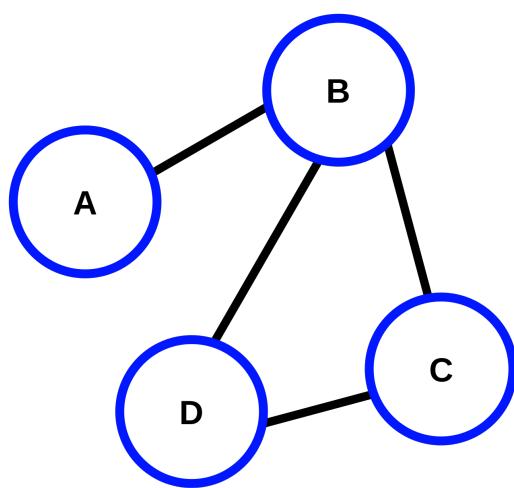
Il grafo che si va a creare è *non orientato*, per cui il numero minimo di archi necessari per costruire un grafo connesso, ovvero un grafo in cui da ogni nodo è sempre possibile raggiungere tutti gli altri nodi tramite un percorso che li collega, diventa il seguente: $numEdges = numNodes - 1$ e questo pone il limite inferiore. Quello superiore è dato dalla costruzione di un grafo completo, un grafo per cui ogni nodo è collegato con tutti gli altri nodi appartenenti al grafo. In questo caso il numero di archi necessari è: $numEdges = \frac{numNodes*(numNodes-1)}{2}$.

All'interno di questi due limiti, viene creato il grafo secondo le esigenze dell'utente come mostrato in figura 56. Successivamente viene creata una matrice di flusso, denominata **migrationMatrix** che rappresenta la percentuale di individui che dal nodo di interesse si spostano verso un nodo obiettivo. Questa matrice è salvata come **matrice sparsa** per risparmiare spazio dove le connessione non sono presenti tra i nodi.

8x8 SparseArrays.SparseMatrixCSC{Float64, Int64} with 28 stored entries:							
.	0.0015721	0.0023041	0.0013949	0.00107337	.	0.00304883	.
0.0018289	0.00340724	0.00365843
0.00134269	0.00183501	0.00213331	0.00225913
0.00213854	.	.	.	0.00118995	0.00343377	.	.
0.00388786	.	.	0.00281135	.	.	0.00880203	0.00958409
.	.	0.00158361	0.00112637
0.00117835	0.00113197	0.00141489	.	0.000939218	.	.	.
.	0.00110731	0.00136506	.	0.0009317	.	.	.

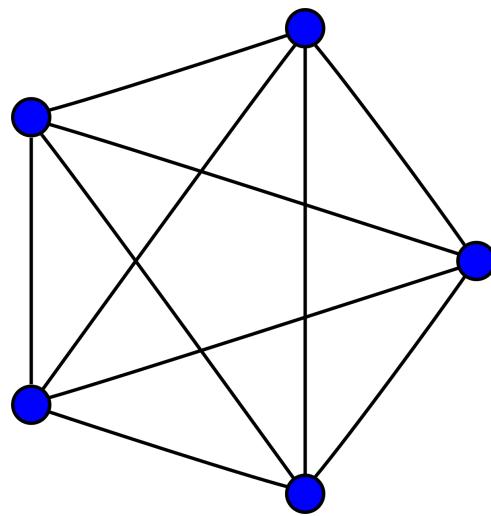
Figure 28: Matrice di migrazione dato un MigrationRate di 0.01

Questa matrice viene esplicitamente creata in base alla topologia del grafo che viene inizializzato in precedenza e si occupa di pesare gli archi di quest'ultimo seguendo la filosofia per cui si ha un flusso maggiore verso i nodi più popolosi, e minore verso i nodi meno popolosi.



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

```

Figure 29: Funzione che crea la matrice di migrazione data la topologia di un grafo

Se questa matrice fosse una matrice non sparsa, e perciò completa, si potrebbe osservare come la diagonale di questa matrice ha valori molto vicini ad 1, in relazione con il limite massimo di individui che possono migrare da ogni nodo, definito come *migration rate*; andando a delineare come la maggior parte della popolazione non effettui in genere spostamenti tra nodi ma che sia più probabile che rimanga nel suo nodo di origine. Questa assunzione può variare nel tempo ad esempio nei periodi di alto flusso migratorio come ad esempio i periodi estivi o più in generale i periodi strettamente collegati alle vacanze.

Successivamente, dopo avere popolato ogni agente (o nodo) del modello con i corretti valori, dato che la struttura interna su cui viene effettuata la computazione del modello è basata su un sistema di ODE, viene istanziato un array di **ODEProblem**. L'idea di spezzettare un sistema di ODE in più sottosistemi di ODE collegate tra loro tramite uno scambio continuo di informazioni (in questo caso individui), è stata pubblicata nel seguente articolo [22] in cui si modella l'outbreak da COVID-19 tenendo in considerazione il flusso migratorio dei voli aerei da e verso differenti nazioni.

Applicando tale filosofia, è possibile creare un array di **ODEProblem**, oggetti che descrivono un sistema di equazioni differenziali ordinarie, ai quali viene messo a corredo un **integratore**. Quest'ultimo fa le veci della classica funzione **solve** che si occupa di risolvere l'intero sistema.



```

1 prob = [OrdinaryDiffEq.ODEProblem(seir!, a.status, tspan, a.param) for a in allagents(model)]
2 integrator = [OrdinaryDiffEq.init(p, OrdinaryDiffEq.Tsit5(); advance_to_tstop=true) for p in prob]
3 model.properties[:integrator] = integrator

```

Figure 30: Definizione dei sistemi di ODE associati ad ogni agente

L'approccio scelto, ovvero di usare un integratore, può essere sostituito tramite l'uso di funzioni di **callback** le quali permettono quando vengono soddisfatte delle condizioni di attivazione specifiche, di andare a modificare i parametri del sistema che si sta risolvendo. Queste modifiche tuttavia, possono portare delle discontinuità nella risoluzione, ma queste vengono esplicitamente gestite se viene specificata una funzione di callback all'interno della funzione di solve, altrimenti l'integratore deve essere notificato che i parametri e i valori al suo interno possono essere stati modificati.

I due approcci sono completamente equivalenti e la scelta di uno piuttosto che dell'altro ricade solamente su un fattore di comodità d'uso.

3.1.3 Funzione di avanzamento agente

Un agente finché all'interno di un modello segue ciclicamente il comportamento riportato in figura 31



```

1 function agent_step!(agent, model::ABM)
2     migrate!(agent, model)
3     happiness!(agent)
4     model.control ? controller!(agent, model) : nothing
5 end

```

Figure 31: Comportamento agente

Ad ogni passo del modello, vengono attivati tutti gli agenti presenti al suo interno e fatti aggiornare seguendo le regole di scheduling preposte.

Ad ogni passo, un agente è chiamato ad eseguire tre fondamentali compiti:

- **Muoversi:** un agente come entità essendo un nodo di un grafo non può muoversi nel modo

più letterale del termine, ma tramite la sua matrice di migrazione è possibile calcolare quanti individui si spostano da un nodo sorgente, fino ad un nodo destinazione. Questo spostamento fa sì che il nodo venga aggiornato con le relative percentuali di status e il totale di popolazione rimanente



```

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

```

Figure 32: Funzione atta a calcolare lo spostamento di agenti da un nodo all'altro del grafo

- **Calcolare la felicità:** come detto in apertura della sezione, il valore di **happiness** è un valore fantoccio che serve prevalentemente per bilanciare le contromisure del controllore. Questo valore tuttavia viene influenzato anche dalla situazione pandemica.



```

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

```

Figure 33: Funzione atta a calcolare la felicità degli agenti

Questo approccio è sicuramente problematico e fallace per numerosi motivi ma attualmente adempie al suo obiettivo di gestore del controllore, in quanto non è mai stato inteso di realizzare uno stimatore affidabile per l'umore generale di una popolazione durante una situazione estrema come può essere una pandemia.

- **Chiamare il controllore:** se la proprietà *control* del modello è impostata su *true*, viene chiamato il controllore che si occupa di stimare, data la situazione attuale del nodo quanto stringenti devono essere le policy da applicare al nodo in questione per cercare di minimizzare il numero di infetti in un determinato numero di passi.

3.1.4 Funzione di avanzamento modello

Ogni passo di avanzamento del modello segna un passo di avanzamento dell'intero sistema. Questo vuol dire che all'interno di un passo del modello vi è un passo per ogni agente. Di default il framework Agents.jl effettua prima l'avanzamento degli agenti e successivamente quello del modello. Questo comportamento può essere modificato quando viene chiamata la funzione di **step!** che si occupa di far girare l'intero sistema.

Il modello si occupa ad ogni passo di effettuare tutte quelle operazioni che non sono legate ad un singolo agente ma che sono legate o alla comunità di agenti oppure al modello in quanto sistema complesso.



```

1 function model_step!(model::ABM)
2     agents = [agent for agent in allagents(model)]
3     for agent in agents
4         # notify the integrator that the condition may be altered
5         model.integrator[agent.id].u = agent.status
6         model.integrator[agent.id].p = agent.param
7         OrdinaryDiffEq.u_modified!(model.integrator[agent.id], true)
8         OrdinaryDiffEq.step!(model.integrator[agent.id], 1.0, true)
9         agent.status = model.integrator[agent.id].u
10    end
11    voc!(model)
12    model.vaccine ? vaccine!(model) : nothing
13    model.step += 1
14 end

```

Figure 34: Funzione di avanzamento del modello

Come è possibile osservare dalla figura 34, il sistema si occupa principalmente di aggiornare notificare l'integratore di ogni agente e aggiornarlo indicandogli che le condizioni precedenti possono essere state alterate, e successivamente effettuare un passo del sistema di integrazione. Successivamente il modello si preoccupa di generare una nuova *Variant Of Concern* (VOC) e infine di chiamare la funzione **vaccine!** la quale è responsabile del calcolo e dell'applicazione delle misure di contromisure farmaceutiche all'interno della popolazione.

Questa sezione è quella che più presta il fianco ad *assunzioni* riguardo il comportamento generale del sistema. Tutte le funzioni sopra descritte si basano su un insieme più o meno nutrito di assunzioni per funzionare correttamente. La funzione che si occupa di generare la VOC è la seguente

```

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 35: Funzione che si occupa di generare la VOC

Si possono notare molte assunzioni:

- la condizione di attivazione della funzione si basa su un valore che sembra totalmente randomico. Quel valore, ovvero $8e - 3$ deriva dai seguenti articoli [52] [81] [2] i quali descrivono prevalentemente il tasso di mutazione casuale delle basi che compongono il DNA del virus SARS-COV2. Questo però non implica che tali mutazioni creino una VOC. Per semplicità è stato scelto di usare l'approccio per cui se una mutazione avviene, questa è un a VOC e, in linea di massima, sembra che questo approccio semplicistico descriva un numero di VOC generalmente sensato e in linea con quanto abbiamo potuto osservare durante la pandemia, facendo ipotizzare che la scelta di usare uno stimatore così semplicistico possa comunque portare a dei benefici
- la distribuzione dei parametri associati alla pandemia viene calcolata seguendo una distribuzione **Normale**, tranne per la distribuzione dell'indice R_0 il quale segue una distribuzione di tipo **Uniforme**. Questa scelta non ha alcun riscontro in letteratura in quanto non vi sono studi che attestano in che modo una variante influenzi i parametri stessi del virus, tanto meno se questa influenza si distribuisce seguendo un andamento normale. Ciò nonostante questa brutale semplificazione è stata applicata per dare un senso all'altrimenti insensata implementazione di una variante.

La funzione che si occupa di stimare l'applicazione delle contromisure farmaceutiche, ovvero l'applicazione del vaccino, è la seguente, e anch'essa presta il fianco a numerose assunzioni, le quali però empiricamente parlando effettuano una buona stima di come sono evolute le cose nel mondo reale.



```

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 36: Funzione che si occupa di simulare la ricerca di un vaccino e la sua successiva applicazione

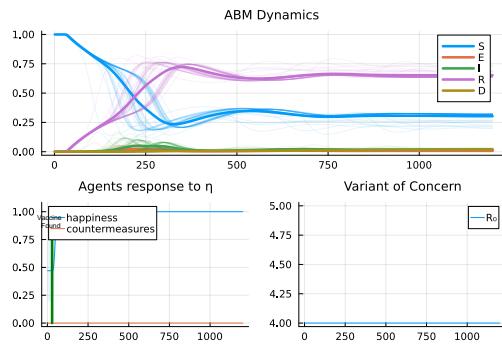
Si nota come in generale la funzione si basa sul generare un valore fissato, generalmente $\in [0, 1]$, che rappresenta la percentuale di popolazione che può essere vaccinata ad ogni passo del modello. Questo approccio tende a simulare l'idea più o meno realistica di avere una quantità fissata di dosi vaccinali ogni giorno che possono essere erogate alla popolazione.

Questo calcolo viene effettuato tenendo in considerazione l'approccio dell' *immunità vaccinale di gregge* che può essere ottenuta seguendo la formula

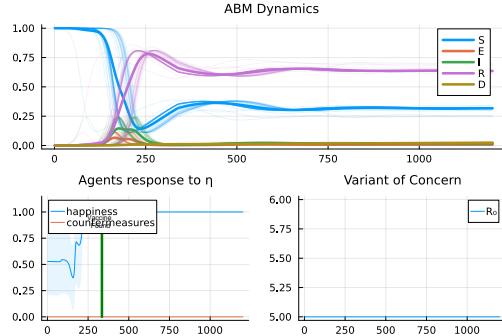
$$V_c = \frac{1 - \frac{1}{R_0}}{\text{VaccineEfficiency}}$$

Successivamente calcolo questo valore come un obiettivo da raggiungere entro un tempo ω che equivale al periodo di immunità che un individuo ha dopo aver contratto la malattia (o aver effettuato il vaccino). Questa idea viene utilizzata per modellare le curve del modello in quanto un individuo vaccinato è considerato come un individuo nello stato *Recovered* e segue le stesse regole di chiunque altro, indipendentemente dal modo in cui si è entrati in questo stato.

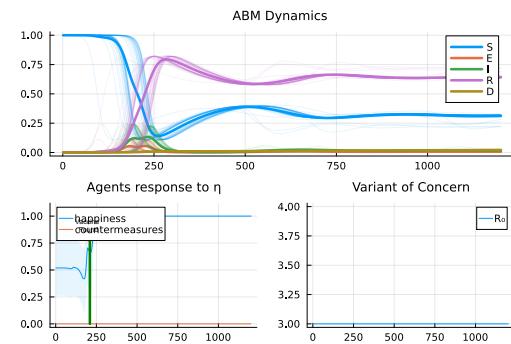
Questa visione semplicistica comunque è efficace nel modellare il decorso degli interventi farmaceutici per le contromisure contro una specifica epidemia, mostrando come l'applicazione di un vaccino e quindi la sua efficacia dipenda principalmente da quando inizia la campagna vaccinale, oltre che dal numero di persone a cui viene somministrato.



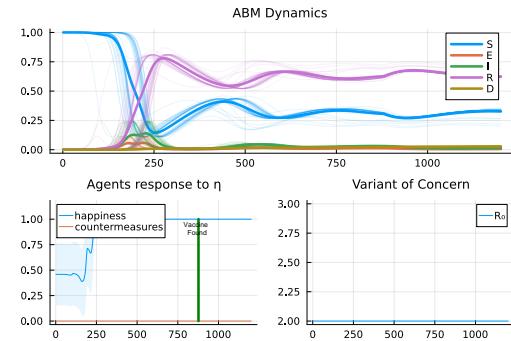
(a) Grafico per la comparazione sul periodo di inizio della campagna vaccinale. Immediata



(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

3.2 Monitoraggio e Intervento

Il sistema di controllo e intervento scelto segue l'idea del controllo autonomo tramite l'utilizzo di una Neural ODE. Questo approccio permette di specificare un sistema di ODE che descrive il sistema reale con le sue relazioni e inserire chirurgicamente una rete neurale al suo interno che controlli il dosaggio delle contromisure. [15] [43] [72]

Questa tecnica permette di ottenere ottimi risultati, i quali tuttavia dipendono fortemente dalla tipologia di modellazione del sistema scelto e dalla tipologia di funzione di costo o di controllo applicata per addestrare il modello. La sfida in questo caso ricade sull'attenta modellazione del sistema di partenza e sulla comprensione dei risultati offerti dal controllore. Questo controllore infatti ritorna dei valori nell'ambito $\in [0, 1]$, i quali possono essere difficilmente interpretabili. Perciò è necessario a priori avere ben chiaro cosa significhi il range di valori che può ritornare il controllore, e capire a cosa è associato nel mondo reale.

Inizialmente la funzione di controllo inserita all'interno del modello ad agente è definita come riportato in figura 38. è possibile osservare come vengano inseriti dei valori di controllo per gestire la chiamata al controllore vero e proprio.

I valori di controllo sono generalmente associati a quadri bisogna realisticamente parlando, chiamare la funzione di controllo. Da un lato servono sia a ridurre il numero di chiamate, e quindi ridurre l'impatto prestazionale sia a rendere più realistica l'idea di avere accesso ad un sistema di controllo solamente in determinati punti del tempo e non sempre. Questo va ad imitare anche il ritardo tra la raccolta dati, la formulazione di una contromisura adeguata e l'applicazione della stessa.

Successivamente vengono definite ulteriori regole di attivazione, questa volta legate alla prima attivazione del controllore. A posteriori è chiaro come avere delle regole di prevenzione già in atto possa essere di enorme aiuto nel combattimento contro il dilagare di un'epidemia, ma alle volte vi è un ritardo nel riconoscere il pericolo, e questo è dato dal valore di *tolerance* il quale non attiva il controllore fintanto che la situazione sembra rimanere sotto i minimi di soglia.



```
1  function control!(  
2      agent,  
3      model::ABM;  
4      tolerance::Float64=1e-3,  
5      dt::Float64=30.0,  
6      maxiters::Int=100  
7  )  
8      if agent.status[3] ≥ tolerance && model.step % dt == 0  
9          agent.param[6] = controller(  
10              agent.status,  
11              vcat(agent.param[1:5], agent.param[7]),  
12              agent.happiness,  
13              (0.0, dt),  
14              maxiters;  
15              loss_step=Int(maxiters / 10),  
16              u_max=Distributions.cdf(Distributions.Beta(2, 5), agent.status[3]),  
17              rng=model.rng  
18          )  
19      end  
20  end
```

Figure 38: Definizione controller all'interno del modello ad agente

Altri valori definiscono principalmente i parametri che servono alla funzione di controllo per operare correttamente. Tra questi vi è un parametro opzionale v il quale descrive quanto potrà essere il valore massimo delle contromisure applicabili in relazione a quanti infetti sono presenti all'interno di un dato nodo. Questo valore viene calcolato come un esponenziale nel numero degli infetti. Questa assunzione è utile come valore di appoggio ulteriore per il controllore nel caso in cui dovesse ritornare un valore troppo alto rispetto a quello che di buon senso sarebbe adatto.

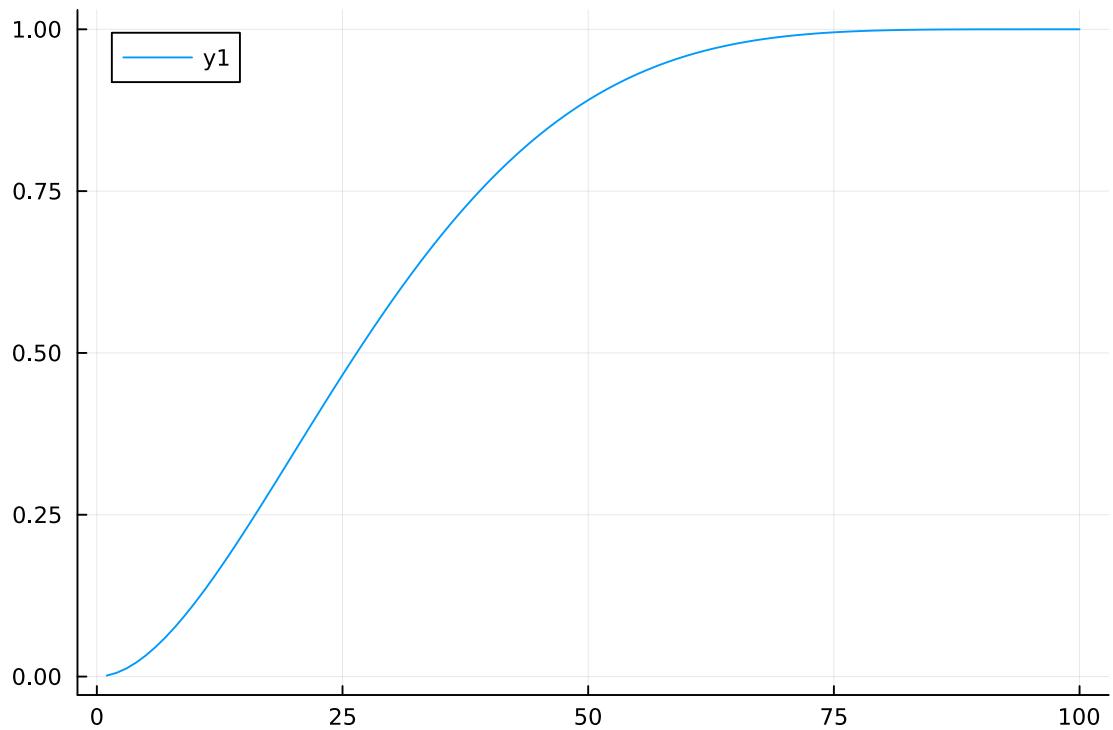
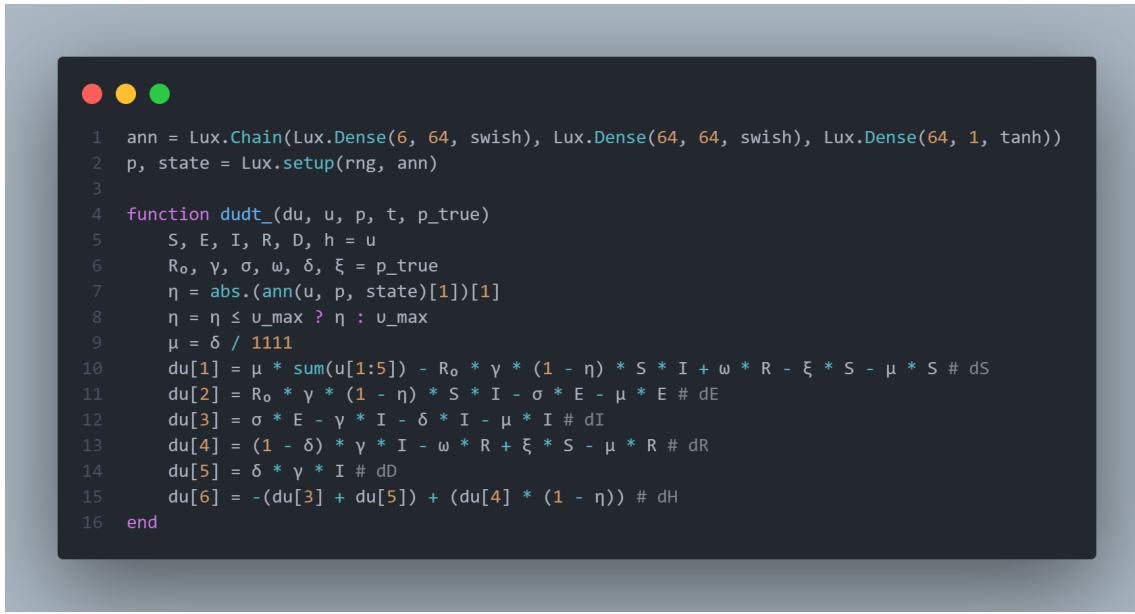


Figure 39: Funzione per il calcolo del limite superiore per le contromisure non farmaceutiche

Come si può osservare, la funzione descritta in figura 39 descrive la *funzione di distribuzione cumulativa* per la distribuzione *beta* con parametri $\alpha = 2$ e $\beta = 5$. Questo tipo di funzione è stata scelta in quanto descrive abbastanza bene il valore di pericolo che, soprattutto durante i primi mesi di pandemia, è stato attribuito al numero di infetti che si riscontravano. Questo valore cresce rapidamente e poi tende a rallentare abbastanza bruscamente una volta superata la soglia del cinquanta per cento di infetti.

Questo può essere anche relazionato all'idea che una pandemia abbia un punto critico dopo il quale le speranze decadono. Questo ovviamente in relazione alla pandemia stessa. Questa funzione potrebbe anche alterarsi nel corso del tempo, modellando perciò una sorta di abitudine alla pandemia.

Il controllore vero e proprio si articola principalmente come già accennato nella sezione relativa alle Neural ODE.



```

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

```

Figure 40: Definizione controllore tramite Neural ODE

Viene definita una rete neurale tramite l’ausilio del framework **Lux.jl** [59], e questa viene utilizzata esclusivamente per il guess del valore di controllo η durante la fase di addestramento, il quale verrà alla fine utilizzato come valore di controllo per il modello. Successivamente viene definito il sistema di ODE che governa il fenomeno che vogliamo analizzare, nello specifico il sistema in questione è un sistema **SEIR** che modella la perdita di immunità con il tempo.

Successivamente viene istanziato il problema con l’ausilio dell’interfaccia **ODEProblem** e da qui in avanti il modello potrà iniziare a essere allenato. Innanzitutto vengono definite delle funzioni per la predizione e il calcolo della loss del modello, e infine anche una funzione ausiliaria di callback utile per osservare l’andamento dell’apprendimento del modello.



```

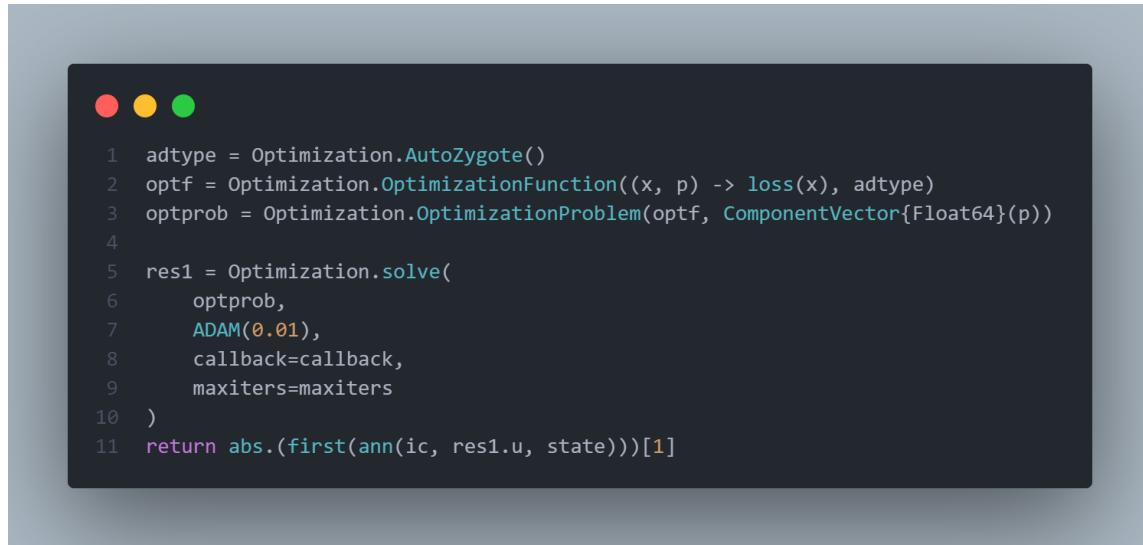
1 function predict(p)
2     _prob = remake(prob, u0=ic, tspan=timeframe, p=p)
3     Array(
4         solve(
5             _prob,
6             Tsit5(),
7             saveat=ts,
8             abstol=1e-10,
9             reltol=1e-10,
10            verbose=false
11        )
12    )
13 end
14
15 function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

```

Figure 41: Definizione funzioni di appoggio del controllore

Come si può osservare in figura 41 la funzione di *callback* oltre ad essere una funzione di appoggio nella stampa dei dati relativi all’addestramento, è anche una utile funzione per implementare un controllo sulla tipologia di apprendimento del modello. Infatti per ridurre ancora di più il numero di iterazioni necessarie per avere dei buoni risultati è stato scelto di inserire un controllo simile al metodo di **early stopping** per prevenire l’addestramento eccessivo quando si trova un valore di ottimo. Questo approccio è puramente *empirico* in quanto è stato riscontrato come il valore di loss generale del sistema tende a decrescere o crescere stabilmente e una volta trovato il valore ottimale, se rimangono delle iterazioni da fare, questo non migliora per cui una volta che i valori rimangono identici è un buon azzardo terminare il periodo di addestramento del modello.

La parte di addestramento vero e proprio del modello avviene con 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
5 res1 = Optimization.solve(
6     optprob,
7     ADAM(0.01),
8     callback=callback,
9     maxiters=maxiters
10 )
11 return abs.(first(ann(ic, res1.u, state)))[1]
```

Figure 42: Definizione funzioni di addestramento del controllore

Il ciclo di addestramento può essere scomposto idealmente in due parti, ma per quanto osservato, non sono entrambe necessarie. La prima parte, riportata in figura 42 si propone di utilizzare un ciclo di addestramento con l'utilizzo dell'ottimizzatore **ADAM**. Se i risultati dovessero esser soddisfacenti, allora è possibile terminare. Tuttavia in questi casi, è buona pratica utilizzare due differenti cicli di addestramento per migliorare i risultati.

Più nello specifico vengono associati due differenti cicli di addestramento con due differenti ottimizzatori per ottenere il massimo da entrambi. Questo approccio serve per massimizzare le peculiarità di ognuno degli ottimizzatori. È pratica comune utilizzare per la buona parte dell'addestramento un ottimizzatore in grado di trovare un buono spazio degli iperparametri e successivamente utilizzare un ottimizzatore come ad esempio **BFGS** [13] [34] [28] [73] il quale è estremamente utile nel trovare rapidamente un minimo locale all'interno dello spazio dei parametri.

L'utilizzo di questi due ottimizzatori in successione è utile in quanto utilizzando solamente **ADAM** si potrebbe incorrere nell'avere un numero di iterazioni troppo elevate per raggiungere lo stesso risultato, mentre utilizzando solamente **BFGS** si potrebbe incorrere in un brutto minimo locale.

La mia scelta di non utilizzare l'approccio sopra descritto, ovvero di utilizzare due differenti ottimizzatori per ottenere il massimo risultato da entrambi, è dato dal fatto che generalmente l'utilizzo di ADAM con un numero di iterazioni molto contenuto, dato anche dall'utilizzo di una tecnica per ridurre ulteriormente le iterazioni di addestramento se non vi è un guadagno di accuratezza 41, permette di arrivare al miglior risultato in breve tempo senza l'utilizzo di un secondo ottimizzatore, il quale se comunque utilizzato non porta alcun miglioramento di sorta, impegnando solamente 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 43: Esempio di utilizzo congiunto di due ottimizzatori differenti in uno stesso ciclo di addestramento

Funzione di attivazione rete neurale

La scelta della funzione di attivazione all'interno delle reti neurali pone un grande impatto sulle dinamiche di apprendimento. Attualmente la più utilizzata e di grande successo è la **ReLU** (Rectified Linear Unit) la quale è della forma $f(x) = \max(0, x)$. Tuttavia svariate alternative sono state proposte, ma nessuna di queste è riuscita a spodestare il primato della ReLU, principalmente a causa di guadagni prestazionali inconsistenti.

La funzione di attivazione che ho deciso di utilizzare si chiama **Swish** ed è stata proposta dal *Google Brain Team* [69], la quale è la seguente funzione $f(x) = x \cdot \text{sigmoid}(x)$. Nei loro esperimenti

è stato osservato come sostituire la funzione di attivazione ReLU con Swish permetteva di avere un aumento prestazionale sui task di *top-1 classification* sul dataset **ImageNet** del 0.9% utilizzando **NASNetA** e dello 0.6% utilizzando **Inception-ResNet-v2**.

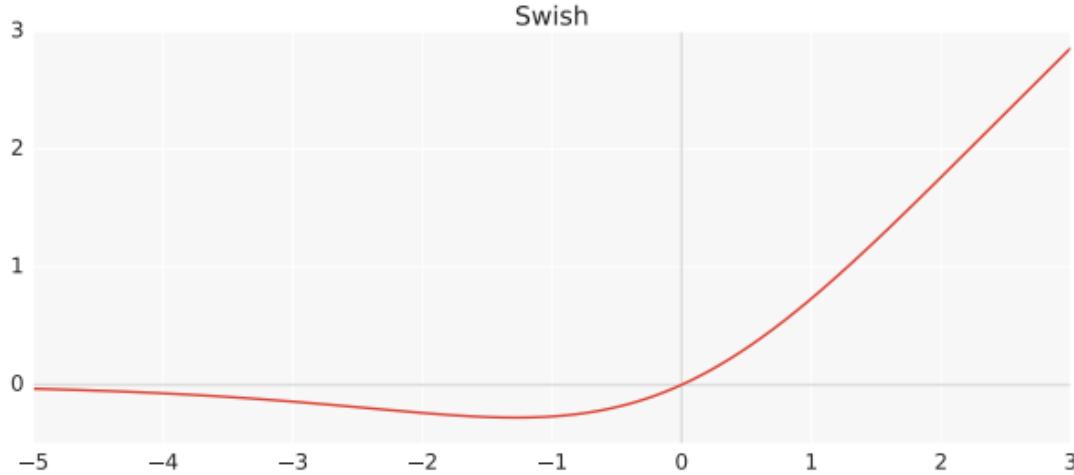


Figure 44: Funzione di attivazione Swish

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

Il punto forte di Swish è la sua semplicità e forte somiglianza con ReLU, la quale la rende estremamente facile da implementare al suo posto. La principale problematica con l'utilizzo di ReLU è il fatto che il valore della derivata è 0 per metà dei valori in input x .

è stato dimostrato come, utilizzando il dataset **MNIST**, le prestazioni di Swish e ReLU siano equiparabili nei fintanto che le reti avevano un numero di layer all'incirca fino a 40. Successivamente Swish riusciva ad ottenere prestazioni migliori di ReLU in maniera considerevole, soprattutto quando il numero di layer si attestava tra i 40 e i 50, casi in cui l'ottimizzazione tendenzialmente diventa più complessa. Questo suggerì, e venne poi osservato, come in reti molto profonde, Swish tendeva ad avere prestazioni migliori nei test di accuracy rispetto alla controparte presa in esame. Entrambe comunque hanno esperienziato un calo delle prestazioni all'aumentare della dimensioni dei batch; in ogni caso comunque Swish otteneva dei risultati migliori.

Alla luce di questi dati ho deciso di utilizzare questa funzione come funzione di attivazione all'interno della rete neurale, anche se tendenzialmente essendo essa poco profonda, i risultati e soprattutto il guadagno di accuratezza non dovrebbe essere troppo sensibile 40.

4 Risultati Ottenuti

I risultati, che verranno dettagliatamente mostrati nelle sezioni successive, possono essere riassunti generalmente come dei risultati positivi, sia in fatto di performance che di policy applicate. Specificatamente al tempo e' possibile mostrare come il tempo generale della simulazione dipenda principalmente dall'applicazione del controllore e quindi dal calcolo della Neural ODE e tutte le operazioni che ne conseguono.

Tempo Impiegato (HH:MM:SS)					
Tipologia di Test	Nessun Intervento	Inter- vento Farmaceutico	Intervento Non Farmaceutico	Intervento Far- maceutico	Intervento Far- maceutico e Non
Singlerun (50 PdI)	0:00:14		0:09:08	0:00:13	0:02:54
Ensemblerun (5 ripetizioni, 50 PdI)	0:00:34		00:41:07	0:00:26	0:54:21

Tempo Impiegato (HH:MM:SS)				
Tipologia di Test	Nodi Infetti In- iziali Variabili	Valore di Mi- grazione Vari- abile	Nodi della Rete Variabile	Copertura dei Nodi Variabile
Paramscan	0:00:13	0:00:33	0:00:33	0:00:13

Queste lunghe tempistiche derivano dal calcolo delle contromisure non farmaceutiche applicabili alla popolazione, e per questo, le contromisure farmaceutiche, se applicate con il giusto tempismo, oltre a dare un notevole apporto alla riduzione della diffusione della pandemia 37a 37b 37c 37d, permettono una notevole riduzione dei tempi computazionali.

4.1 Nessun intervento

Il seguente grafico 45 mostra l'andamento delle curve del modello quando questo viene eseguito senza alcuna tipologia di intervento. Questo andamento è mostrato in maniera cumulativa rispetto all'andamento dei singoli agenti, i quali possono mostrare comportamenti differenti tra loro.

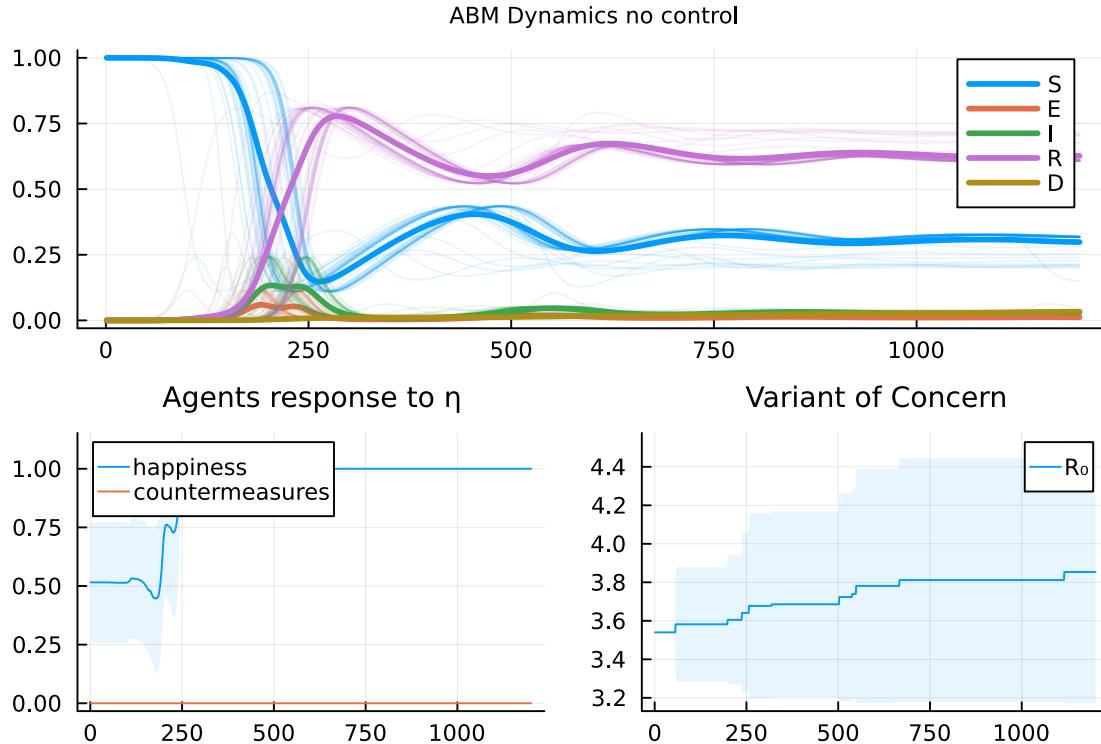


Figure 45: Grafico cumulativo del modello senza alcun tipo di intervento

Complessivamente l'andamento del modello è similare all'andamento standard di un modello di tipo SEIR, con qualche variazione dipendente dai fattori di stocasticità intrisechi del modello; che in questo caso non sono troppo presenti. Il grafico mostra le traiettorie più comuni delle curve cumulate del modello, dove vengono messe in evidenza i percorsi più utilizzati.

Come è possibile notare, il numero di individui suscettibili crolla drasticamente per via della diffusione rapida e simil esponenziale che ha il virus. Questa viene emulata dall'altrettanto rapida crescita di individui guariti (recovered) che però, per via di come è stato definita la condizione di guariti, non sono immuni alle varianti del virus, permettendo di modellare una possibile ciclicità dell'epidemia data dalla perdita di immunità della popolazione. Queste proprietà contribuiscono ad un andamento ciclico delle curve. Si può notare come la curva associata all'andamento degli individui nella classe D abbia una crescita lineare, seppur non troppo evidente.

A seguire si può osservare come la curva associata alla variabile di happiness del modello, valore che serve per bilanciare la durezza delle misure di controllo per evitare di cadere in un ciclo funzionale ma insostenibile, mostra un comportamento alquanto bizzarro. Questo è dovuto principalmente a

come viene definita la funzione di controllo della felicità 33. Si osserva inoltre che la curva tende ad un *plateau* passato il periodo della "prima ondata".

Questo comportamento principalmente irrealistico e associato alla definizione che è stato fatto della funzione **happiness** 33. Il comportamento di questa curva non è completamente realistico, ma è comunque utilizzabile per lo scopo di mantenere sotto controllo le contromisure η .

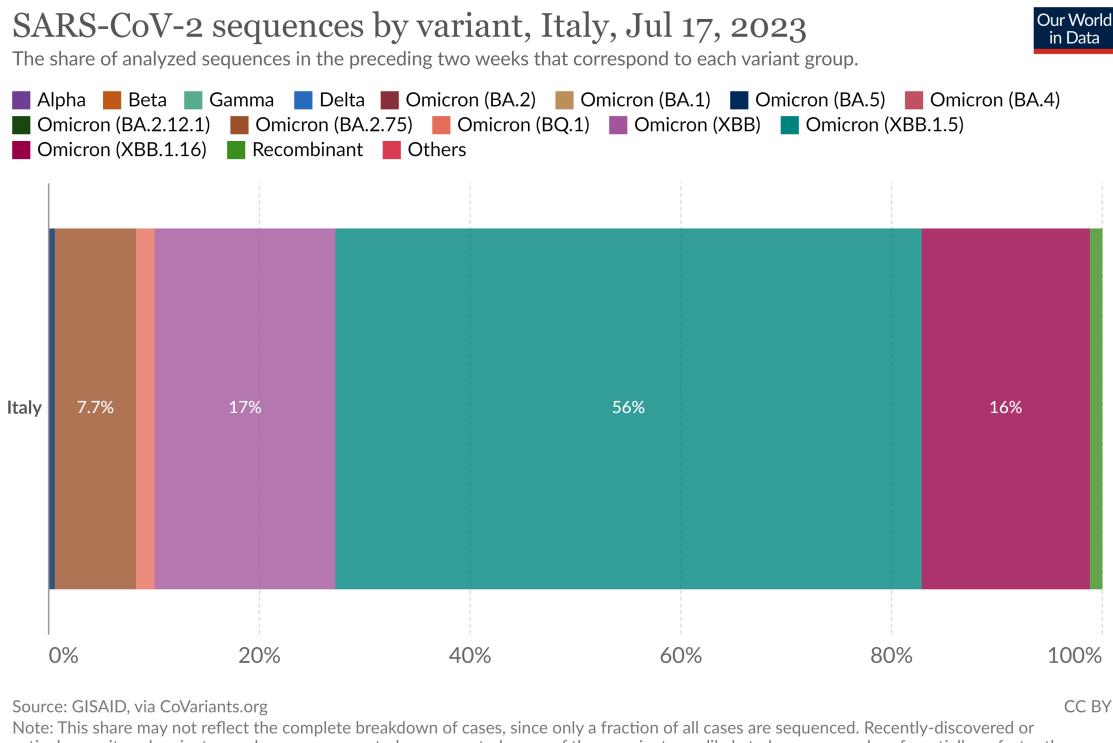


Figure 46: Grafico delle mutazioni del virus SARS-COV2 preso da Our World in Data
<https://ourworldindata.org/coronavirus>

Infine si nota come, seppur la definizione della funzione associata alla creazione di una nuova *VOC* 35 sia semplicistica e irrealistica, il grafico mostra come su un periodo di circa 3 anni, associabile alla durata del periodo covid, il numero di VOC sia pressoché sovrapponibile con quanto osservato dai dati raccolti durante la pandemia 46.

4.2 Intervento non farmaceutico

Il seguente grafico 47 mostra l'andamento delle curve del modello quando questo viene eseguito tramite l'applicazione di una qualche tipologia di intervento non farmaceutico. Le contromisure sono rappresentate come un insieme di valori appartenenti all'intervallo $[0, 1]$ le quali rappresentano cumulativamente un insieme di metriche differenti non esplicite che vanno a definire un insieme di misure di controllo. Le misure di controllo cercano di mimicare quelle associate al progetto **OxCGR**T [38] il quale ha lo scopo di misurare la durezza delle contromisure applicate in un determinato paese; questo valore è composto di 9 metriche: *chiusura delle scuole; chiusura dei luoghi di lavoro; cancellazione di eventi pubblici; restrizioni agli assembramenti pubblici; chiusura dei trasporti pubblici; obbligo di rimanere a casa; campagne di informazione pubblica; restrizioni agli spostamenti interni e controlli sui viaggi internazionali.*

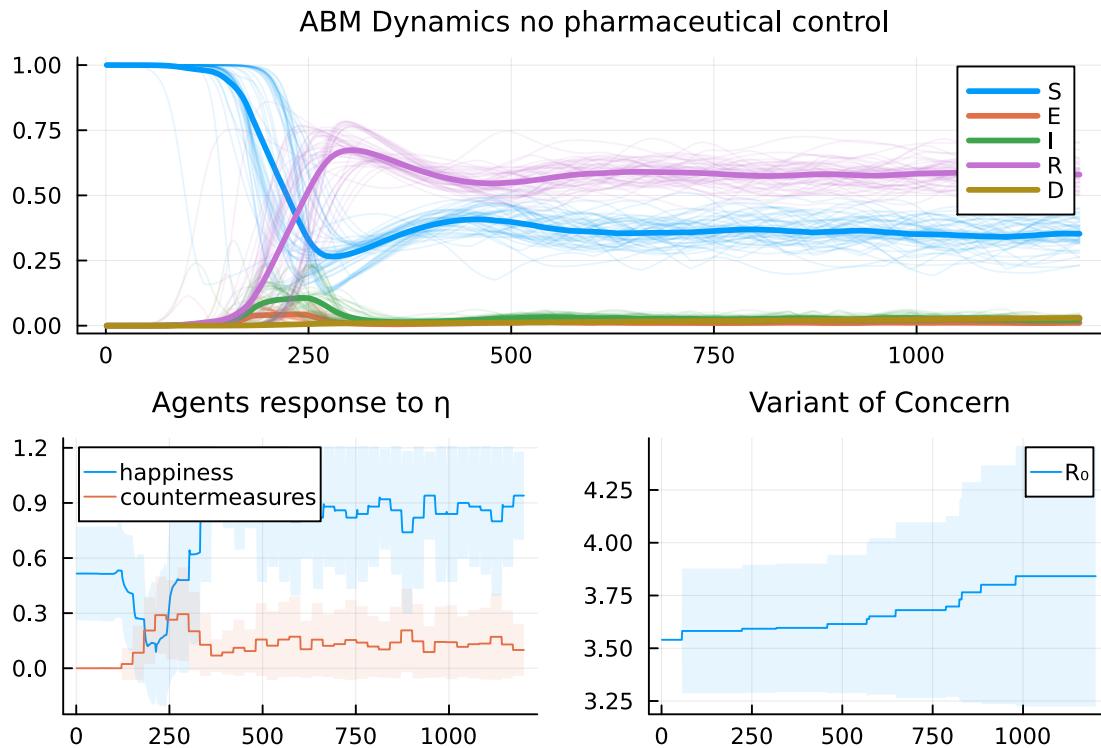


Figure 47: Grafico cumulativo del modello con intervento non farmaceutico del controllore

Queste metriche sono metriche reali ma che nel modello vengono viste come un insieme unico, associabile ad una somma di medie dei valori di ogni metrica. Questa scelta porta a non avere un indice molto chiaro ma permette comunque di avere un'idea generale estremamente immediata dell'effetto che queste hanno sul modello.

Si nota come con l'applicazione di un insieme di contromisure, più o meno stringenti a seconda del periodo osservato, possiamo osservare come le curve epidemiologiche tendano ad appiattirsi, soprattutto quelle associate ai compartimenti **E**, **I**, le quali influenzano le curve **S**, **R**. In questo

caso la ciclicità del modello viene meno.

Altro dato interessante è il numero di VOC che è diminuito rispetto al modello senza intervento. Questo dipende generalmente dal comportamento delle contromisure. Infatti queste quando vengono applicate, oltre a influenzare il parametro **happiness**, vanno ad influenzare anche la matrice di flusso 28 andando a ridurne i valori presenti. Questo fa sì che oltre a dilazionare la diffusione della pandemia, va a dilazionare il comportamento della funzione che si occupa di generare una VOC 35.

Questa infatti può attivarsi se nel nodo sono presenti individui infetti, in quanto non è stata modellata la possibilità che spontaneamente una nuova variante arrivi in un nuovo nodo senza un veicolo umano. Perciò applicare delle contromisure permette anche a contenere la diffusione di VOC nella popolazione osservata.

Si può osservare come la variabile di controllo **happiness** segua molto attentamente l'andamento sia delle contromisure che della pandemia, arrivando all'inizio della pandemia quando le contromisure sono più stringenti a diventare pressoché nulla. Successivamente, pur mantenendo un livello di contromisure sostenuto, la media del valore di controllo tende ad alzarsi, seguendo il valore del comportamento **R**. Questo approccio sembra imitare il generale andamento della risposta che la popolazione italiana ha avuto alle prime misure di contenimento applicate realmente durante la pandemia da COVID-19.

4.3 Intervento farmaceutico

Il seguente grafico 48 mostra l'andamento delle curve del modello quando questo viene eseguito tramite l'applicazione di una qualche tipologia di intervento farmaceutico. Questo grafico dipende enormemente da quando viene trovato e successivamente applicato un intervento farmaceutico alla popolazione. Prima viene applicato un intervento farmaceutico più le curve si modificano e sono differenti dall'andamento senza intervento 45, mentre più tempo passa più il comportamento tende ad assomigliarsi.

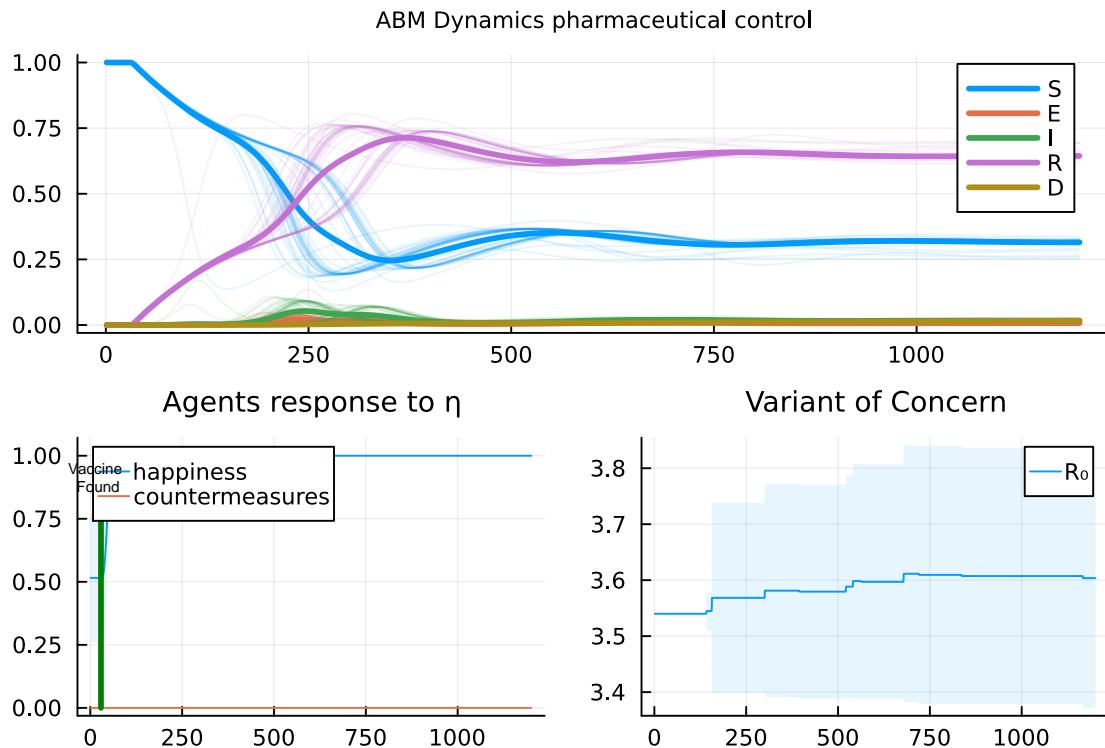


Figure 48: Grafico cumulativo del modello con intervento del controllore tramite interventi farmaceutici come ad esempio un vaccino

Il grafico mostra come se disponibili e applicate repentinamente, l'utilizzo di contromisure farmaceutiche permette di ridurre repentinamente le curve senza andare ad intaccare la curva di happiness in maniera troppo sensibile.

4.4 Intervento farmaceutico e non farmaceutico

Il seguente grafico 49 mostra l'andamento delle curve del modello quando questo viene eseguito tramite l'applicazione sia di un intervento di tipo farmaceutico come ad esempio l'utilizzo di un vaccino, che di contromisure non farmaceutiche.

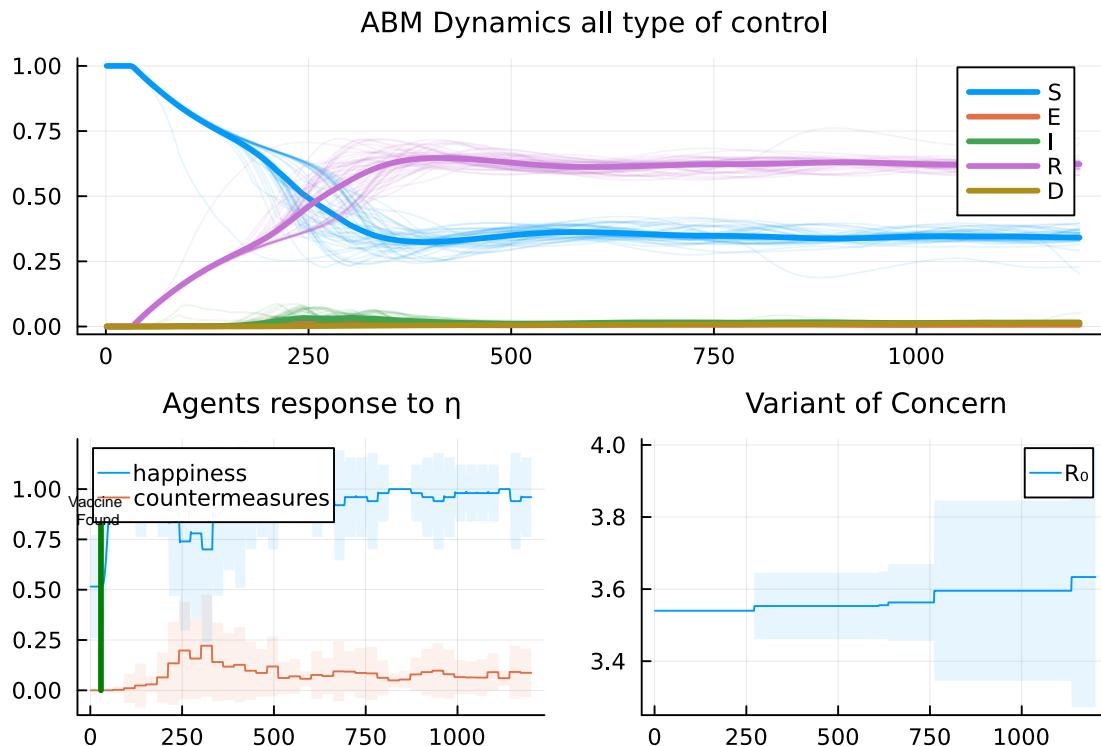


Figure 49: Grafico cumulativo del modello con intervento del controllore tramite vaccino e metodi di prevenzione non farmaceutici

In questo caso viene mostrato l'andamento delle curve tenendo in considerazione l'utilizzo di ogni mezzo per prevenire e contrastare l'epidemia. L'utilizzo combinato di mezzi farmaceutici e non permettono di appiattire notevolmente la curva di infetti andando a creare velocemente un'immunità di gruppo che rende la popolazione meno suscettibile alle varianti. Questo inoltre fa sì che la *happiness* media del modello sia generalmente più alta anche quando vengono applicate delle contromisure che vanno ad intaccare la felicità della popolazione (ad esempio un lockdown).

Queste contromisure non farmaceutiche sono in genere meno stringenti e meno prolungate, permettendo quindi alla popolazione di non avere un calo drastico della *happiness* generale come in figura 47. Tuttavia dipendono fortemente da quando le contromisure farmaceutiche (il vaccino) vengono applicate e con che efficacia questo riesce ad entrare in circolazione.

Tuttavia questo dimostra come l'utilizzo di un vaccino a priori sia un metodo molto efficace per contrastare una epidemia, e che ovviamente l'efficacia di questo dipenda molto e soprattutto da quando viene applicato alla popolazione. Successivamente si può notare come le misure non farma-

ceutiche di prevenzione e contrasto dell'epidemia sono un mezzo efficace per controllare la diffusione dell'infezione, ma queste hanno un costo in termini sia di generale qualità della vita, che anche economico, non indifferente come mostrato dai grafici precedenti e dall'esperienza diretta che si è avuto con la pandemia da COVID-19.

4.5 Analisi di sensitività

Osservando il modello SEIR definito come segue 17 si può osservare come quest'ultimo potrebbe essere sensibile al mutamento di specifici parametri al suo interno, i quali potrebbero portare il modello ad avere comportamenti peculiari e soprattutto imprevisti o inattesi.

Perciò è stato deciso di effettuare un analisi di sensitività del sistema tramite l'utilizzo di metodi di libreria offerti dalla suite *SciML.ai*. In questo modo è stato possibile osservare a quali parametri il modello fosse più sensibile.

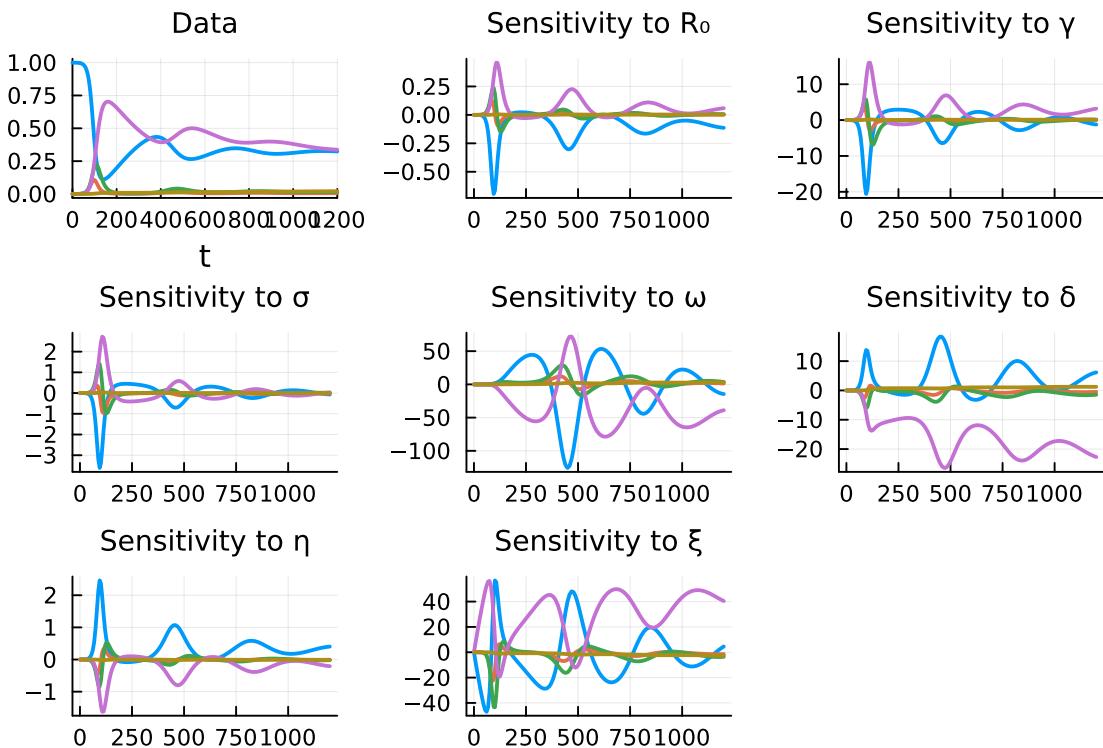


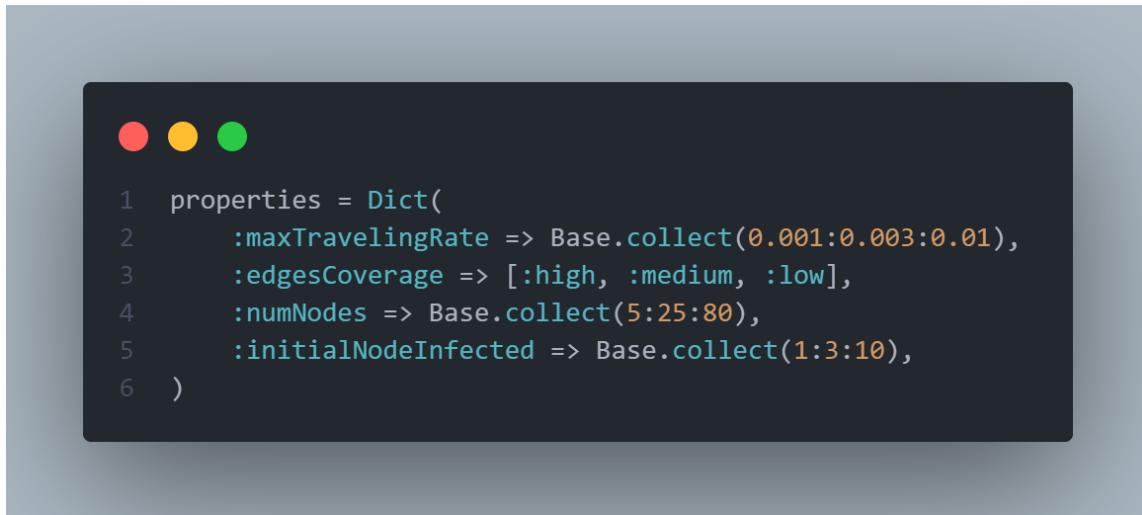
Figure 50: Grafico rappresentante l'analisi di sensitività del modello

Come osservato in figura 50, il modello ha una sensitività abbastanza uniforme per i parametri R_0 , γ e σ , il che era atteso in quanto sono tutti responsabili dell'avanzamento del contagio. La sensitività ai parametri ω e δ era comunque attesa, anche se non così accentuata come mostrata in figura. Infine la sensitività relativa al parametro n responsabile delle contromisure è pressocchè inversa a quella del parametro R_0 il che è relativamente atteso in quanto è un parametro direttamente collegato alle contromisure per la diffusione del contagio. La sensitività al parametro ξ invece ha andamento

singolare e pare essere legato alla sensitività del parametro ω seppur in modo specchiato. Questo probabilmente perché essendo i parametri ω e ξ legati al compartimento **R** del modello, che gestisce gli individui sia guariti che vaccinati, il loro comportamento è sensato che sia collegabile.

Successivamente è stata applicato un controllo di sensitività riguardo i parametri relativi al modello ad agente in se, tramite l'utilizzo della funzione **paramscan** offerta dalla libreria **Agents.jl**.

è stato scelto un insieme di parametri del modello che vengono modificati all'interno di un range definito e in base alle varie configurazioni ottenute si va ad osservare come il comportamento del modello può o meno cambiare. I valori che sono stati scelti in quanto idealmente responsabili di un comportamento differente se modificati sono i seguenti:



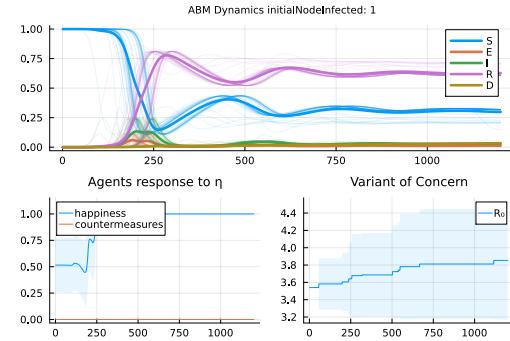
```
● ● ●  
1 properties = Dict(  
2     :maxTravelingRate => Base.collect(0.001:0.003:0.01),  
3     :edgesCoverage => [:high, :medium, :low],  
4     :numNodes => Base.collect(5:25:80),  
5     :initialNodeInfected => Base.collect(1:3:10),  
6 )
```

Figure 51: Parametri usati per l'analisi di sensitività del modello ad agente

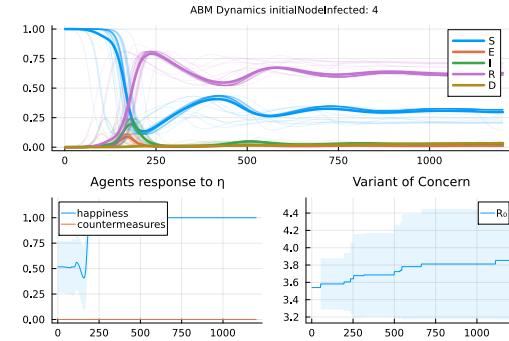
Questi parametri sono stati scelti in come possibili parametri con un comportamento latente. I parametri relativi al controllore e al vaccino non sono stati inclusi per ovvi motivi, in quanto è stato già dimostrato come il loro variare può portare a cambiamenti significativi all'interno della simulazione del modello 45 47 48 49.

4.5.1 Analisi comportamento dato un numero di nodi infetti iniziali variabile

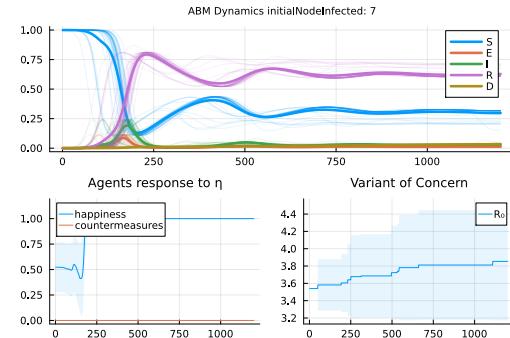
In questo confronto si nota come il numero di nodi infetti iniziali influenzi principalmente la velocità con cui il picco di infetti arriva a compimento. Si nota come nel caso in cui si inizi con un numero di nodi infetti maggiore, la curva degli infetti tenderà a essere prona verso l'inizio della linea temporale piuttosto che la fine, arrivando al proprio picco prima. Oltre a questo tuttavia non vi sono altri comportamenti degni di nota. Nel complesso è un comportamento atteso.



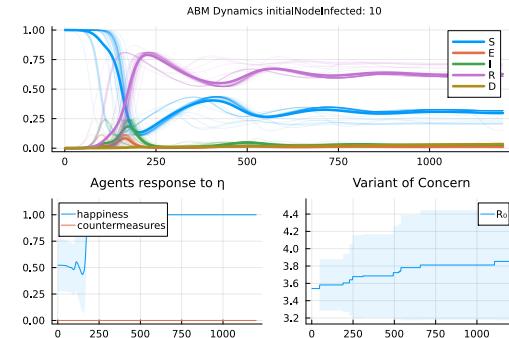
(a) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 1



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



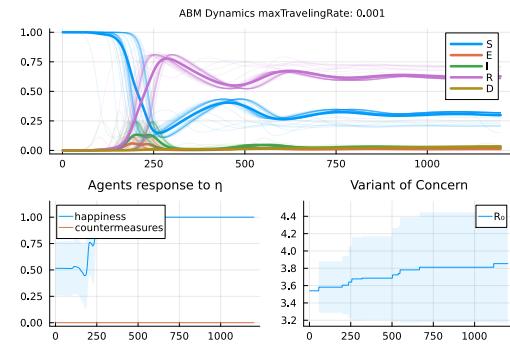
(c) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 7



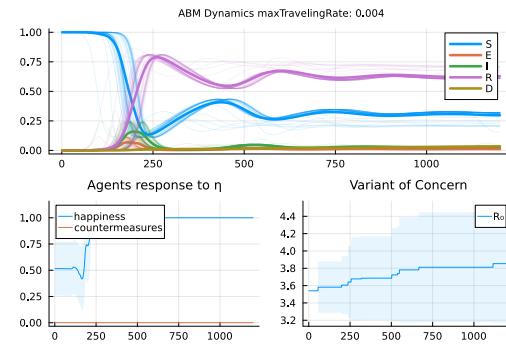
(d) Grafico per la comparazione sul numero di nodi infetti di partenza. Numero nodi infetti iniziali 10

4.5.2 Analisi comportamento dato il valore di migrazione variabile

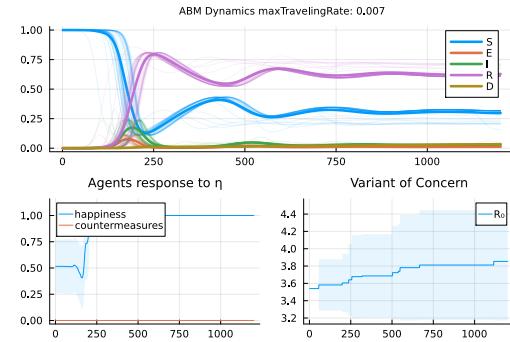
Anche in questo caso si può osservare come la curva epidemiologica tenda a raggiungere un valore di picco più alto in un periodo di tempo minore nel caso in cui si ha un rateo di migrazione più alto. Anche questo comportamento era atteso e non sorprende.



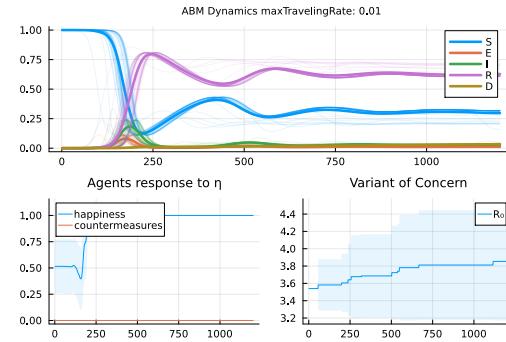
(a) Grafico per la comparazione sul valore di migrazione. Valore di 0.001



(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

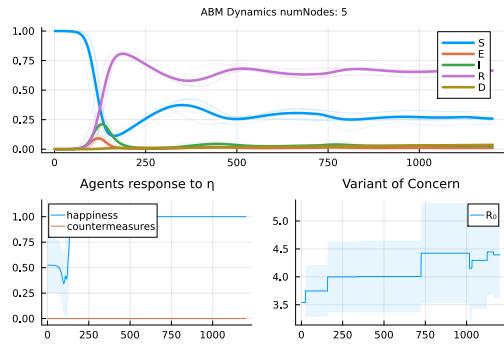


(d) Grafico per la comparazione sul valore di migrazione. Valore di 0.01

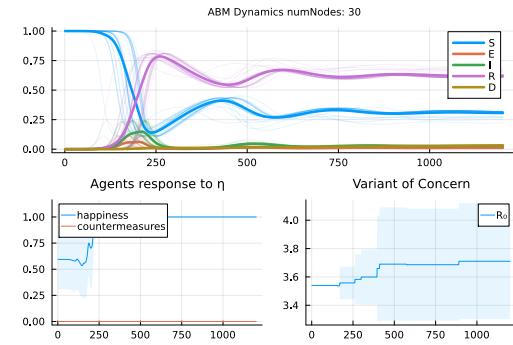
Questo comportamento dipende anche da cosa il parametro **migrationRate** sta ad indicare. Questo parametro indica il massimo numero di individui che possono spostarsi da un nodo di partenza verso un nodo di destinazione. Questo valore viene calcolato quando viene creata la matrice di migrazione 28 e generalmente è un valore *sempre minore* di quello specificato come migrationRate.

4.5.3 Analisi comportamento dato il numero di nodi della rete variabile

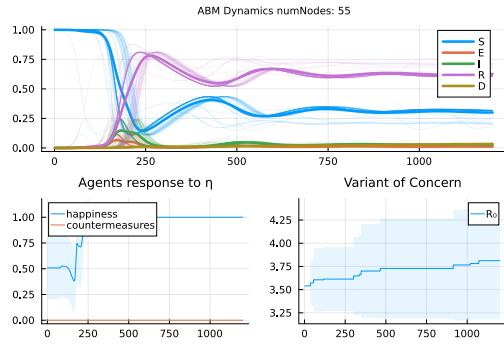
In questo confronto è lapalissiano come il numero di nodi della rete influenzi l'andamento delle curve epidemiologiche. I grafici mostrano come in genere un numero di nodi maggiore comporta ad una diffusione più lenta e con picchi meno ripidi seppur più prolungati nel tempo. Ovviamente questo dipende da molteplici fattori, primo tra tutti la topologia delle connessioni del grafo e in secondo luogo dove il primo focolare ha inizio.



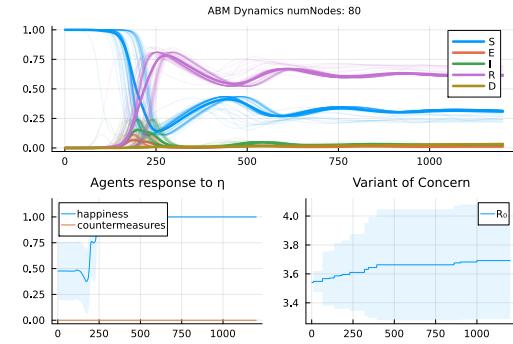
(a) Grafico per la comparazione sul numero di nodi della rete. Numero nodi 5



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



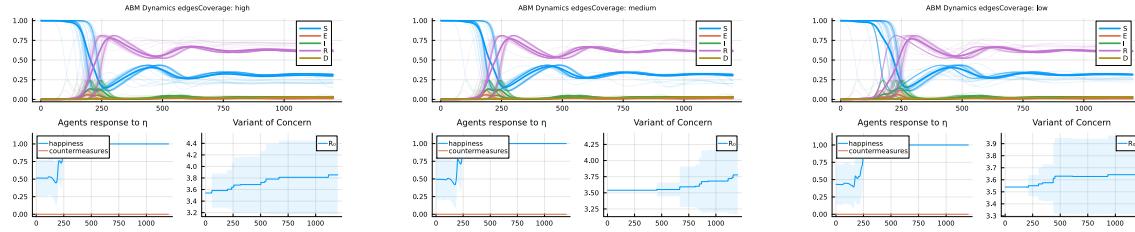
(c) Grafico per la comparazione sul numero di nodi della rete. Numero nodi 55



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

4.5.4 Analisi comportamento data la copertura dei nodi variabile

In questa figura è osservabile come la copertura della rete, data dal numero di archi che la compongono, va ad influenzare in maniera sensibile l'andamento della pandemia, soprattutto nel caso in cui la copertura è bassa 55c. Tendenzialmente è ipotizzabile che con una bassa copertura, quindi con una possibilità più limitata di spostarsi da un nodo all'altro, la diffusione del virus subisce un arresto, e questo è osservabile con i dati ottenuti durante la pandemia COVID-19 dove una delle prime contromisure applicate è stata quella del **lockdown generalizzato** in cui la capacità di spostamento degli individui è calata drasticamente.



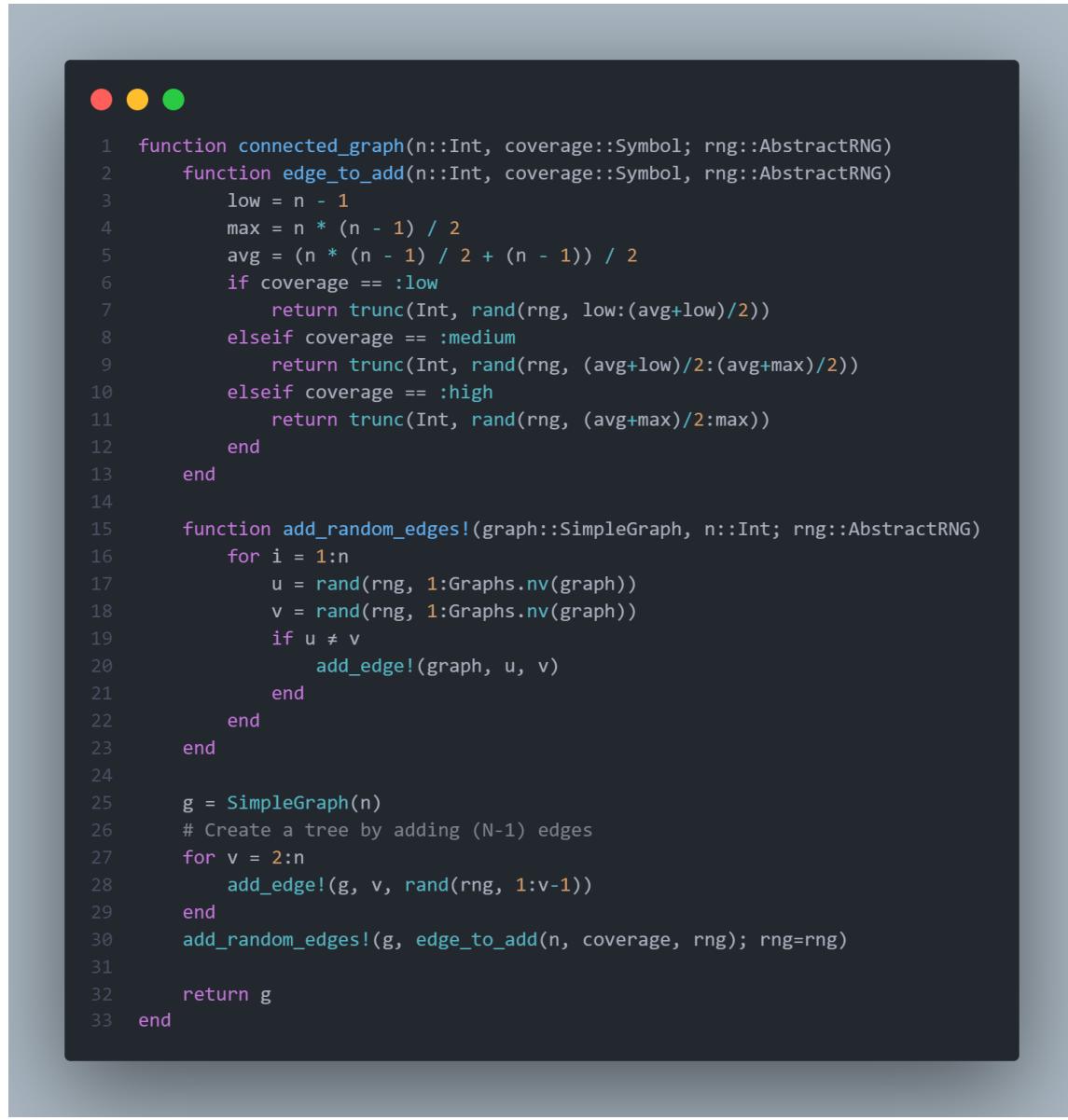
(a) Grafico per la comparazione sulla copertura della rete. Copertura alta

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

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

è accurato assumere come, dato il modo con cui viene creata la copertura degli archi, questa possa influenzare la diffusione del virus. In particolar modo la diffusione nel caso di alta e media copertura è molto simile. Questo è dovuto con molta probabilità al modo in cui viene creata la copertura; modo che può essere osservato in figura 56

Ciò nonostante, quello che può essere ulteriormente osservato è il comportamento sempre più "incerto" delle curve epidemiologiche al diminuire della copertura del grafo. Discorso similare si può osservare nella comparazione sul numero di nodi della rete, in cui le curve epidemiologiche tendono ad assumere un comportamento sempre più incerto all'aumentare dei nodi presenti in rete. Anche il numero di **VOC** pare aumentare con il numero di nodi, questo suggerisce che essendoci più individui la possibilità di mutazioni aumenta, così come diminuisce con il diminuire della copertura, stando ad indicare come seppur il numero di nodi possa essere elevato, ma se non vi è una possibilità o veicolo di infezione, le mutazioni difficilmente possono accadere; diminuendo la pericolosità dell'agente patogeno.



```
1 function connected_graph(n::Int, coverage::Symbol; rng::AbstractRNG)
2     function edge_to_add(n::Int, coverage::Symbol, rng::AbstractRNG)
3         low = n - 1
4         max = n * (n - 1) / 2
5         avg = (n * (n - 1) / 2 + (n - 1)) / 2
6         if coverage == :low
7             return trunc(Int, rand(rng, low:(avg+low)/2))
8         elseif coverage == :medium
9             return trunc(Int, rand(rng, (avg+low)/2:(avg+max)/2))
10        elseif coverage == :high
11            return trunc(Int, rand(rng, (avg+max)/2:max))
12        end
13    end
14
15    function add_random_edges!(graph::SimpleGraph, n::Int; rng::AbstractRNG)
16        for i = 1:n
17            u = rand(rng, 1:Graphs.nv(graph))
18            v = rand(rng, 1:Graphs.nv(graph))
19            if u ≠ v
20                add_edge!(graph, u, v)
21            end
22        end
23    end
24
25    g = SimpleGraph(n)
26    # Create a tree by adding (N-1) edges
27    for v = 2:n
28        add_edge!(g, v, rand(rng, 1:v-1))
29    end
30    add_random_edges!(g, edge_to_add(n, coverage, rng); rng=rng)
31
32    return g
33 end
```

Figure 56: Esempio della funzione atta alla creazione di un grafo dato un determinato livello di copertura

5 Sviluppi Futuri

5.1 Perfezionamento Controllore

Uno sviluppo futuro può essere il perfezionamento del controllore sia nelle sue capacità di individuazione e applicazione delle policy, sia nella chiarezza di ritornare delle policy *human-understandable* così da essere facilmente tradotte e comprese anche da un operatore umano. Questo sviluppo apre le porte allo studio di causalità, elemento molto importante trattato in apertura dell'elaborato in quanto bisognerà cercare di capire una relazione che vi è tra l'applicazione di una specifica contromisura e il suo apporto alla diminuzione di una pandemia.

Questo approccio è veramente complesso ma di estremo interesse in quanto potrebbe risultare in un balzo tecnologico non indifferente.

5.2 Miglioramento funzione happiness

Un'altro grande scoglio definito come assunzione è la funzione di happiness, responsabile del controllo della funzione Controllore. Questa funzione attualmente serve solamente come brutale asticella per evitare di cadere all'interno di un ciclo disfunzionale di contromisure insostenibili nella vita reale. Tuttavia se venisse migliorato tenendo conto di quanto veramente ogni topologia di contromisura possa impattare sulla vita delle persone, sia sul breve che sul lungo periodo, questo potrebbe essere una pietra miliare per l'accuratezza delle policy sviluppate in quanto si avrebbe un valore di riscontro molto più accurato.

Tuttavia anche questa sezione richiede un attenta analisi della causalità degli eventi, in particolare stimare quanto in generale una contromisura possa impattare l'umore di una popolazione. Umore il quale varia in maniera sia per via delle contromisure che anche per stimoli esterni alle volte neanche immaginabili. Ad aiuto sarebbe comodo avere a corredo molteplici *benchtest* caratterizzati da differenti modelli di simulazione; partendo da quello più granulare per comprendere il comportamento emergente relativo al livello generale di soddisfazione di una popolazione dato un ambiente dinamico e auto influenzante, fino ad arrivare a mettere tutto quanto ottenuto insieme per avere una simulazione macroscopica in grado di offrire risultati affidabili.

5.3 Perfezionamento generale modello

Il modello implementato si basa su una serie di assunzioni relativamente ragionevoli, date dall'osservazione empirica di quanto si voleva andare a modellare. L'osservazione e i dati raccolti durante tutto il periodo COVID-19 sono stati utili per ottenere un modello che possa essere abbastanza veritiero. Ciò nonostante questo modello attualmente non si presta molto bene alla generalizzazione, ponendo il fianco a delle rigidità strutturali che ne compromettono l'uso.

è quindi buona pratica quella di avere un modello che ben si adatti e generalizzi ai vari problemi cui viene sottoposto.

5.4 Ottimizzazione generale

Il codice sviluppato sicuramente ha nampio margine di miglioramento soprattutto sotto l'aspetto delle performance generali. In futuro è un nobile obiettivo quello di renderlo più efficiente, scalabile e ottimizzato possibile.

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 Approcci scartati

A.1 Modello ad Agente su spazio continuo

L'idea alla base era quella di modellare una popolazione tramite l'utilizzo di uno spazio del modello di tipo continuo. Gli agenti sarebbero poi stati modellati come individui singoli e reali, in quanto entità effettivamente attive all'interno dello spazio. Ci si può immaginare lo spazio del modello come una grande griglia definita da coordinate *continue* di una certa dimensione $N \times M$. All'interno di questa griglia vengono posizionati randomicamente gli agenti, i quali vengono rappresentati graficamente come dei punti di differenti colori.

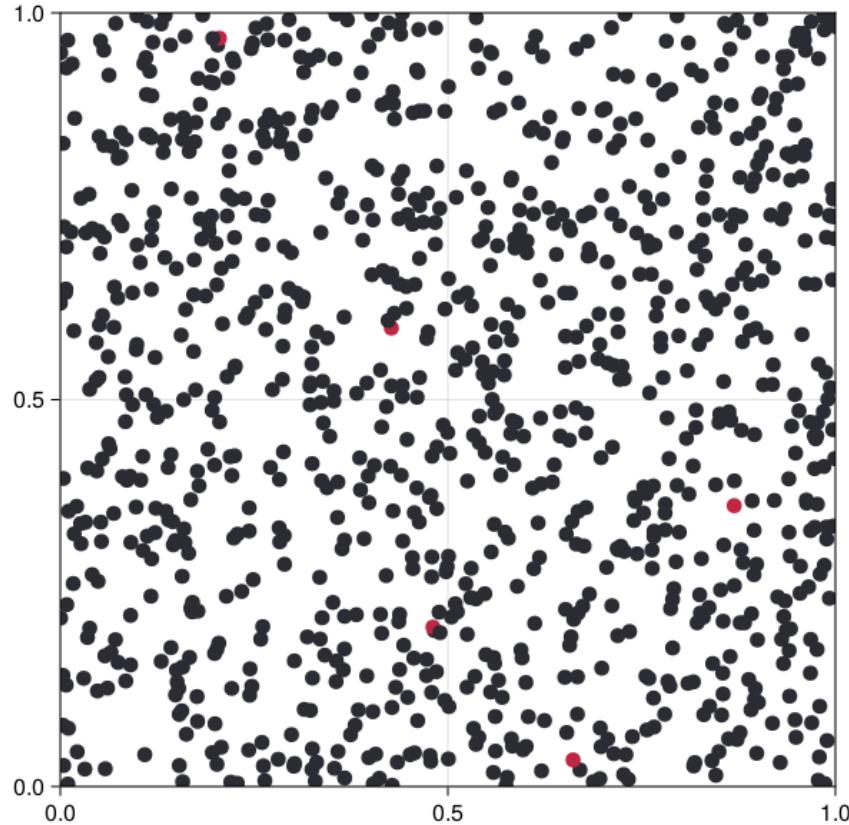


Figure 57: Esempio del modello modellato su spazio continuo

Questo approccio utilizzava una metodologia similare a quella delle palline da biliardo per modellare l'interazione tra agenti all'interno dello spazio del modello. Ogni agente poteva infettare in maniera casuale un suo vicino se i due avevano un interazione. Ogni interazione era modellata similmente ad un urto elastico tra corpi. Questo non era tuttavia un simulatore di urti elastici affidabili, bensì un approccio che prendeva spunto da esso, in particolare dal gioco del biliardo.

In questo modo il comportamento complessivo del modello poteva permettere l'osservarsi di un

andamento simile a quanto ci si aspetterebbe da un sistema simile.

ABM ContinuousSpace Dynamics

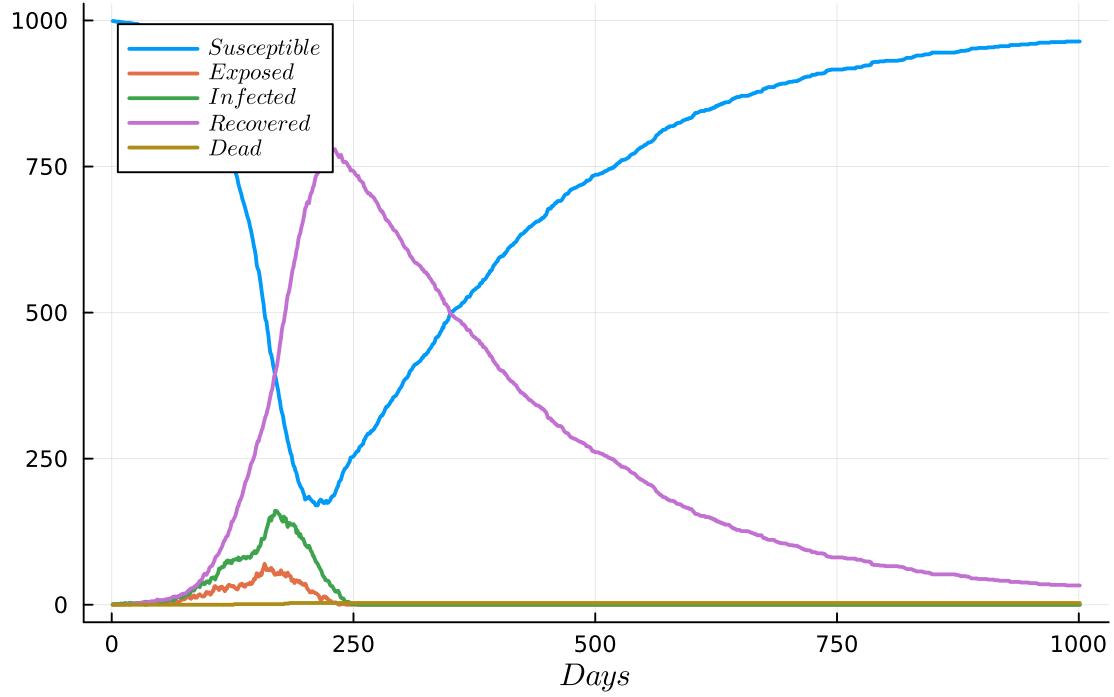


Figure 58: Esempio del comportamento delle curve nel modello continuo

Sono state poi implementate svariate proprietà come la proprietà di essere individuati dopo un periodo di latenza come individui infetti e quindi essere confinati in quarantena, la quale era definita come una diminuzione nella probabilità di infettare ed essere infettati perdendo la capacità di muoversi.

La tipologia di approccio del tipo Agent-oriented richiede una quantità di risorse computazionali e di tempo estremamente elevato in relazione al numero di agenti presenti nel modello. Questo approccio inoltre è stato valutato come troppo granulare e inadatto allo scopo del progetto. Si è deciso quindi di adottare un approccio meno granulare e più flessibile andando ad un livello di astrazione più alto.

A.2 Modello ad Agente con spazio a grafo e modellazione singolo agente

L'idea è nata per cercare di avere un controllo più granulare sullo spazio del modello e sulla sua evoluzione locale, non tanto degli agenti. Sono nati molteplici problemi, primo tra tutti quello del tempo impiegato e successivamente quello del comportamento delle curve epidemiologiche che non rispettavano assolutamente il comportamento descritto dal modello deterministico SEIR, ma non in un modo ragionevole, bensì in un modo completamente alieno.

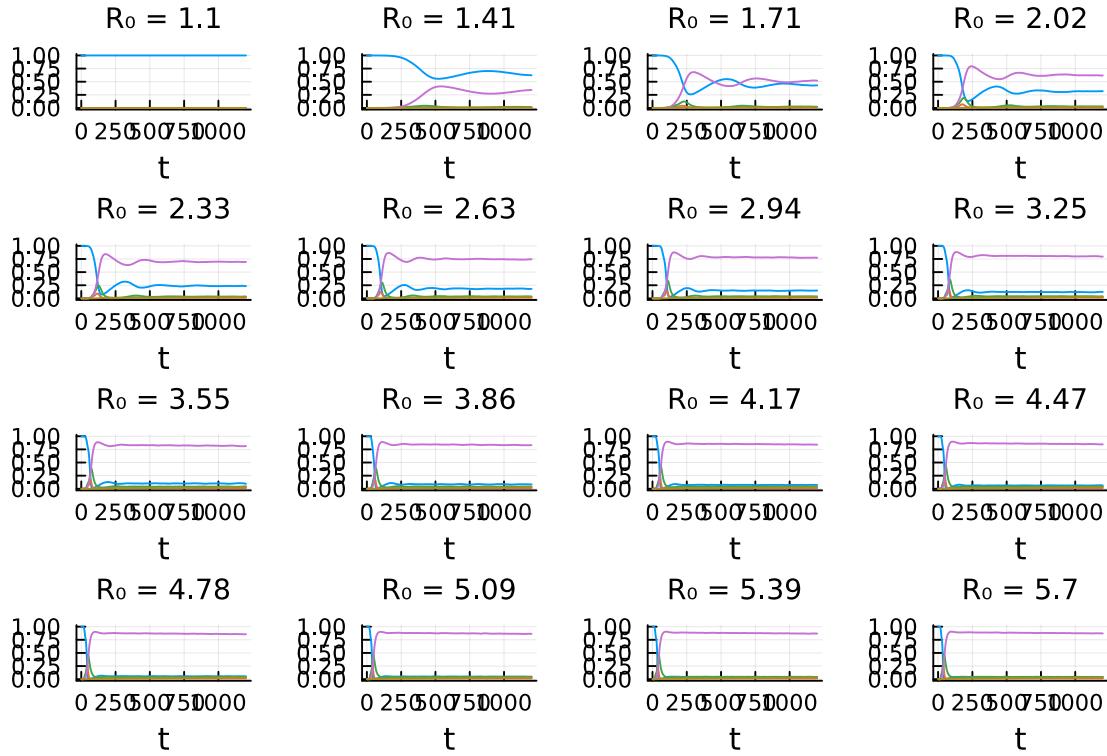


Figure 59: Comportamento modello ABM su spazio a grafo al variare del parametro R_0

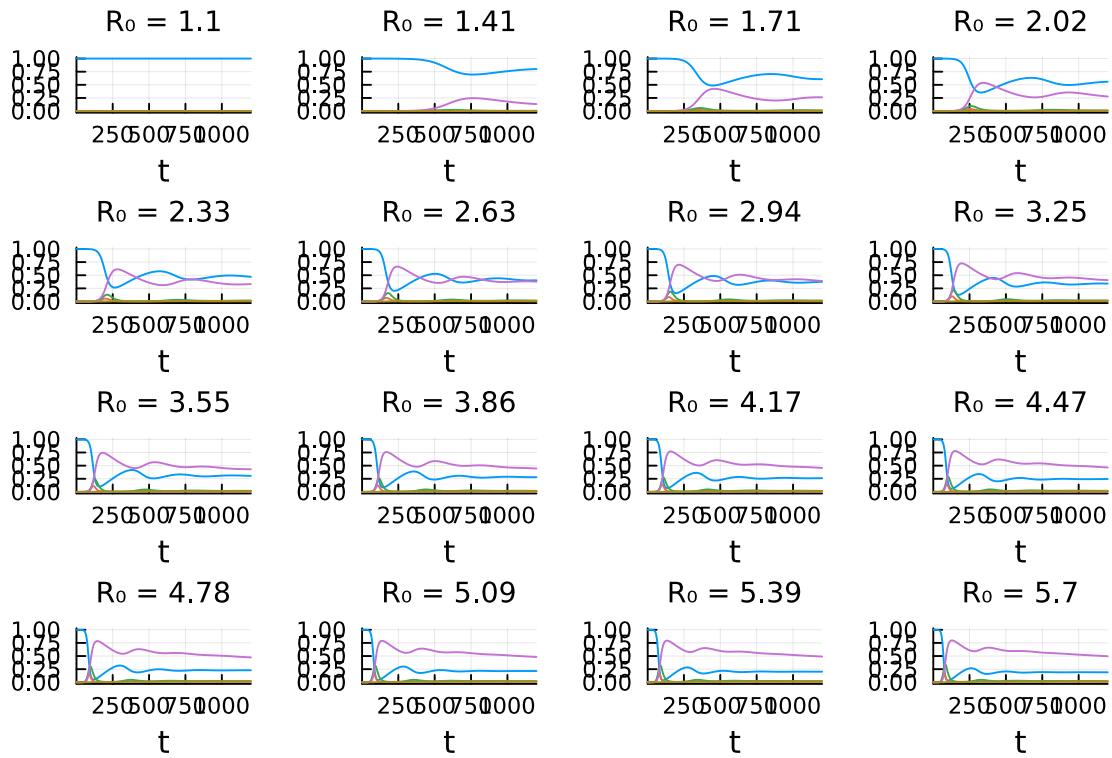


Figure 60: Comportamento modello SEIR al variare del parametro R_0

Si può facilmente osservare come il comportamento delle curve prenda un comportamento anomalo fin dalle prime variazioni del parametro R_0 per culminare con risultati completamente alieni. Il motivo alla base rimane tutt'ora ignoto e sconosciuto, tuttavia esecuzioni differenti hanno mostrato dei comportamenti differenti, seppur altrettanto alieni.

Questo comportamento però può essere descritto a grandi linee dalla seguente formula

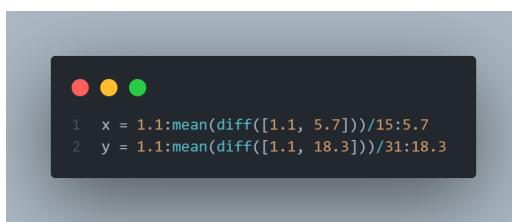


Figure 61: Formula che si occupa di descrivere il rapporto tra il comportamento del modello scartato e del modello SEIR. In particolare questa formula descrive il rapporto tra gli R_0

Bisogna tuttavia precisare come run successive abbiano portato a risultati differenti dei plot relativi

al comportamento del modello ad agente. Il motivo rimane tutt'ora ignoto ma è possibile che uno degli attori in gioco possa essere la funzione **random** associata al campionamento degli individui che va a fare da mimica per il numero di contatti che avvengono ad ogni passo per ogni individuo infetto. Questo campionamento si basa principalmente sul calcolo di una distribuzione di **Poisson** con parametro $\lambda = R_0$.

In particolare la formula è stata ricavata dopo aver fatto un confronto tra molteplici valori di R_0 e i risultati ottenuti sia dal modello SEIR che dal modello ABM. Successivamente è stato calcolato **MSE** (Medium Square Error) per ogni coppia possibile di risultati per cercare quelli tra loro più simili, andando ad ottenere quindi un insieme di coppie. Queste poi sono state inserite in un metodo che calcolava una **polynomial fit** tra tutte le coppie di valori per vedere quale poteva essere la formula che governava la differenza di risultati. Da tenere a mente che in un caso normale questa differenza non dovrebbe esistere, o quanto meno se esiste dovrebbe essere trascurabile.



(a) figure

Range di valori di R_0 testati

```

1 for i = 1:length(res_abm)
2     temp = []
3     for j = 1:length(res_ode)
4         push!(temp, mean(abs2, res_abm[i] .- res_ode[j]))
5     end
6     push!(res_a, x[i])
7     push!(res_b, y[findmin(temp)[2]])
8 end

```

(b) Calcolo MSE risultati SEIR - ABM

Da qui si ottiene la formula in figura 61 la quale descrive approssimativamente, in base al grado del polinomio che si vuole andare a creare, la relazione che esiste tra le coppie di valori analizzate.

La motivazione per cui il modello ad agente si comporta in maniera così inaspettata rispetto a come dovrebbe non è stato chiaro, e in letteratura non sembra esserci alcun articolo che ne parli in maniera approfondita, e per questo ho deciso di abbandonare l'approccio. Punto a favore è stato anche il fatto che l'approccio con ABM era estremamente esoso in termini di risorse computazionali e temporistiche. Questo problema è insito in ogni tipo di simulazione, tuttavia affiancato allo stravagante problema comparso e descritto sopra, è stato decisivo per il cambio drastico di approccio.

A.3 Controllore Ipopt

Ipopt (Interior Point OPTimizer) [80] è un pacchetto software per l'ottimizzazione non lineare su larga scala. Questo pacchetto software è realizzato per trovare delle soluzioni (locali) a problemi di ottimizzazione matematica nella forma: $\min_{x \in R^n} f(x)$ tale che $g_L \leq g(x) \leq g_U$ e $x_L \leq x \leq x_U$, dove $f(x) : R^n \rightarrow R$ è la funzione obiettivo e $g(x) : R^n \rightarrow R^m$ sono le funzioni di vincolo.

I vettori g_L e g_U denotano i limiti inferiore e superiore sui vincoli e i vettori x_L e x_U sono i limiti delle variabili x . Le funzioni $f(x)$ e $g(x)$ possono essere sia non lineari che non convesse, ma la loro derivata seconda deve esistere e deve essere continua.

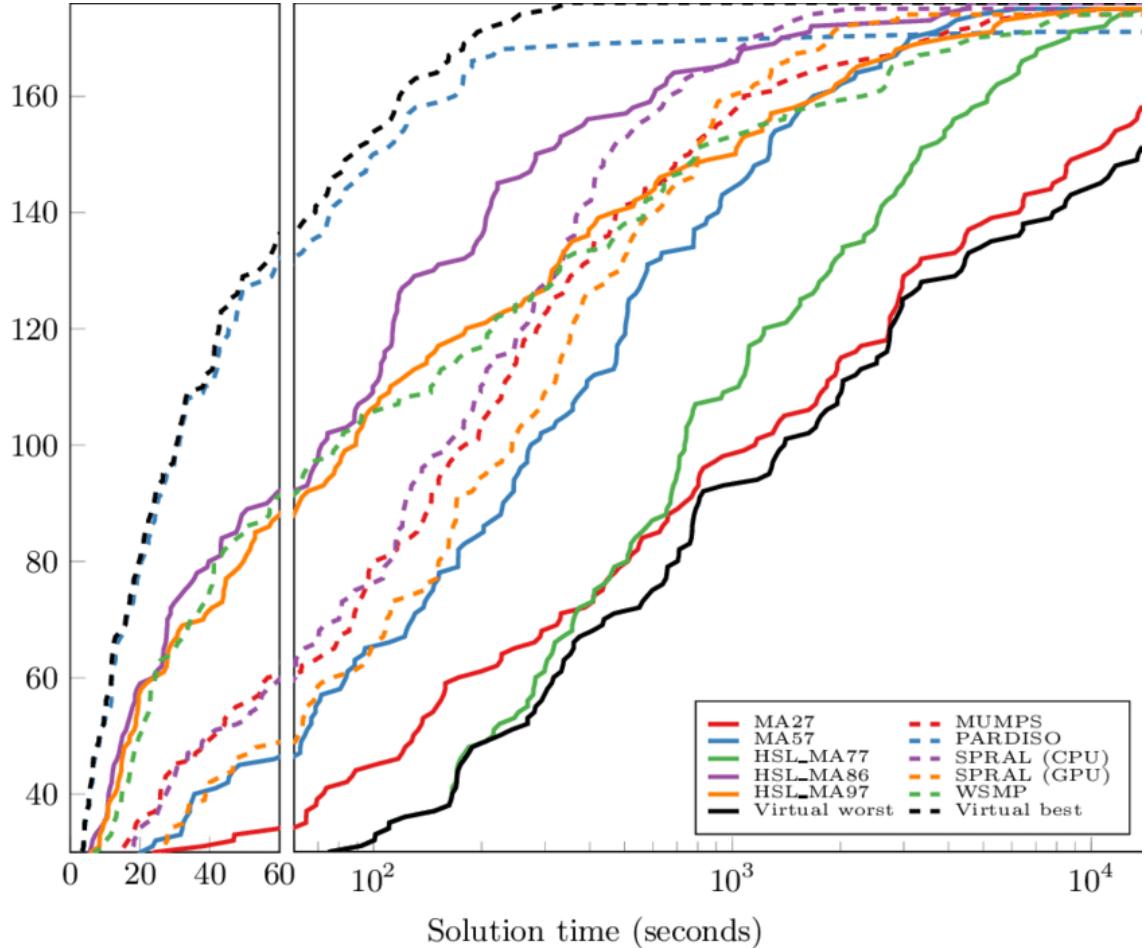
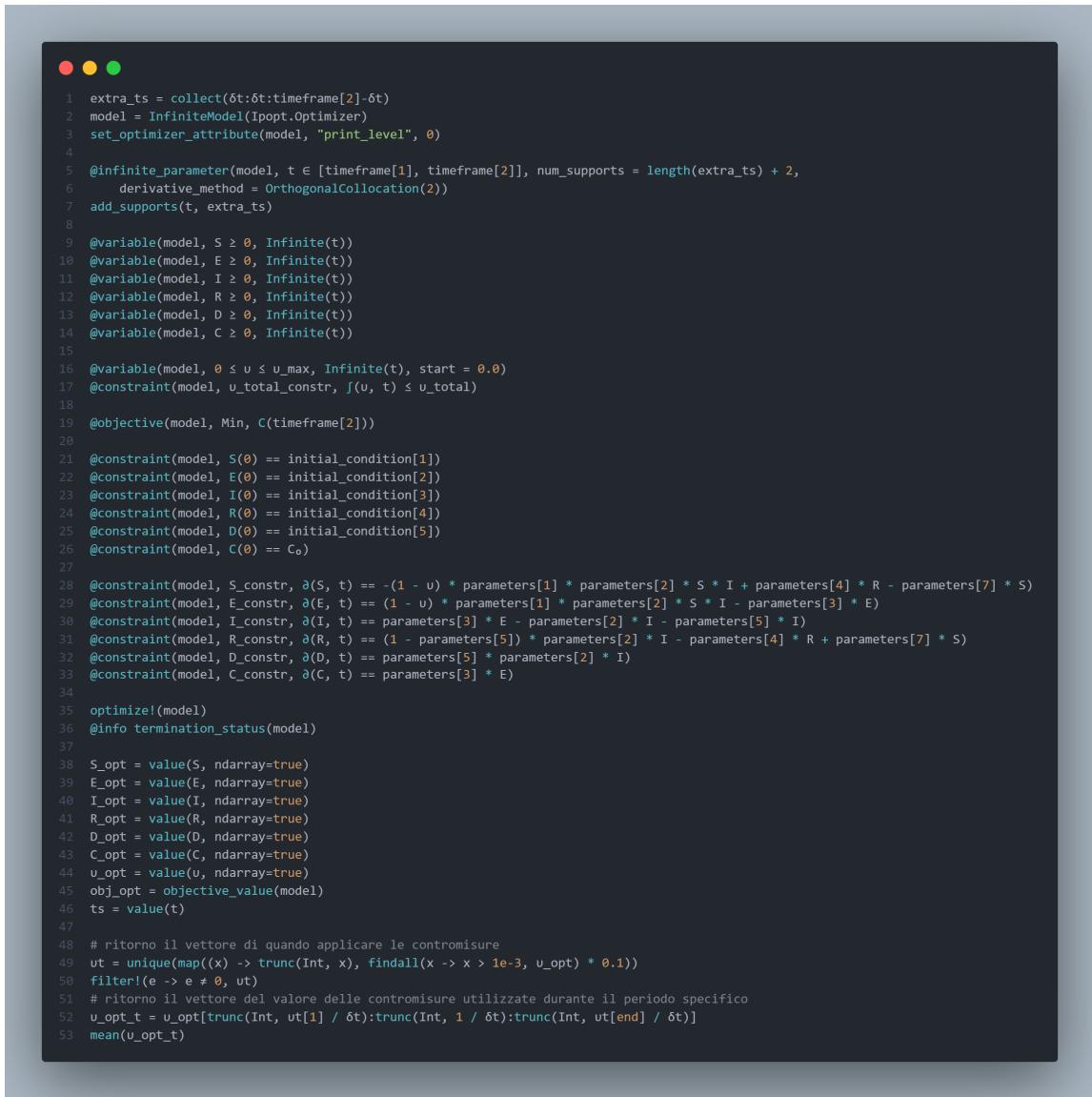


Figure 63: Comparison of Ipopt performance over various linear solvers using the two-dimensional partial differential equation test problem set. [76]

Ipopt è scritto in C++ ed è stato rilasciato come software open source sotto la licenza **Eclipse Public License (EPL)**.

L'utilizzo della suite **Ipopt** è stato fatto per l'applicazione di un sistema di monitoraggio e intervento all'interno del modello di simulazione. Successivamente l'idea di utilizzare un'integrazione con la suite **SciML.ai** verrà sfruttata per aggiungere algoritmi di Machine Learning per rendere più realistico e consistente il modello nelle sue predizioni e scelte.



```

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) == C₀)
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 - parameters[4] * R + parameters[7] * S)
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 64: Definizione del controllore tramite Ipopt

L'approccio generale è stato semplice ma efficace, in quanto viene definito un modello per raccogliere tutte le informazioni relative e necessarie per l'algoritmo di ottimizzazione e successivamente vengono definite le regole che governano il comportamento del modello.

Le regole in questione sono principalmente le stesse che sono usate per descrivere il modello SEIR che viene utilizzato all'interno del modello ad agente.



```

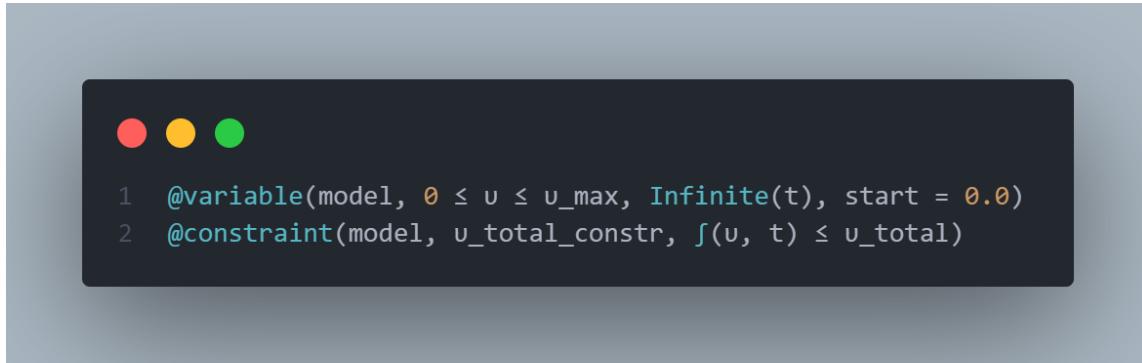
1  @constraint(model, S(0) == initial_condition[1])
2  @constraint(model, E(0) == initial_condition[2])
3  @constraint(model, I(0) == initial_condition[3])
4  @constraint(model, R(0) == initial_condition[4])
5  @constraint(model, D(0) == initial_condition[5])
6  @constraint(model, C(0) == C0)
7
8  @constraint(model, 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 65: Definizione regole del modello del controller

Essendo che le regole mostrate in figura 65 sono relative agli stati SEIR e l'idea alla base del controllore è quella di ridurre quanto più possibile il numero di infetti cumulati che ci sono all'interno del sistema, è stato aggiunto uno stato che descrive appunto questo stato aggiuntivo.

Successivamente vi sono delle regole su quanto il modello può impiegare in termini di risorse, le quali sono le nostre contromisure con relativo costo, dato dall'integrale del valore della nostra contromisura applicata nel tempo. Infine viene ottimizzato il modello e ritornato il valore medio delle contromisure applicate quando applicate.



```

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

```

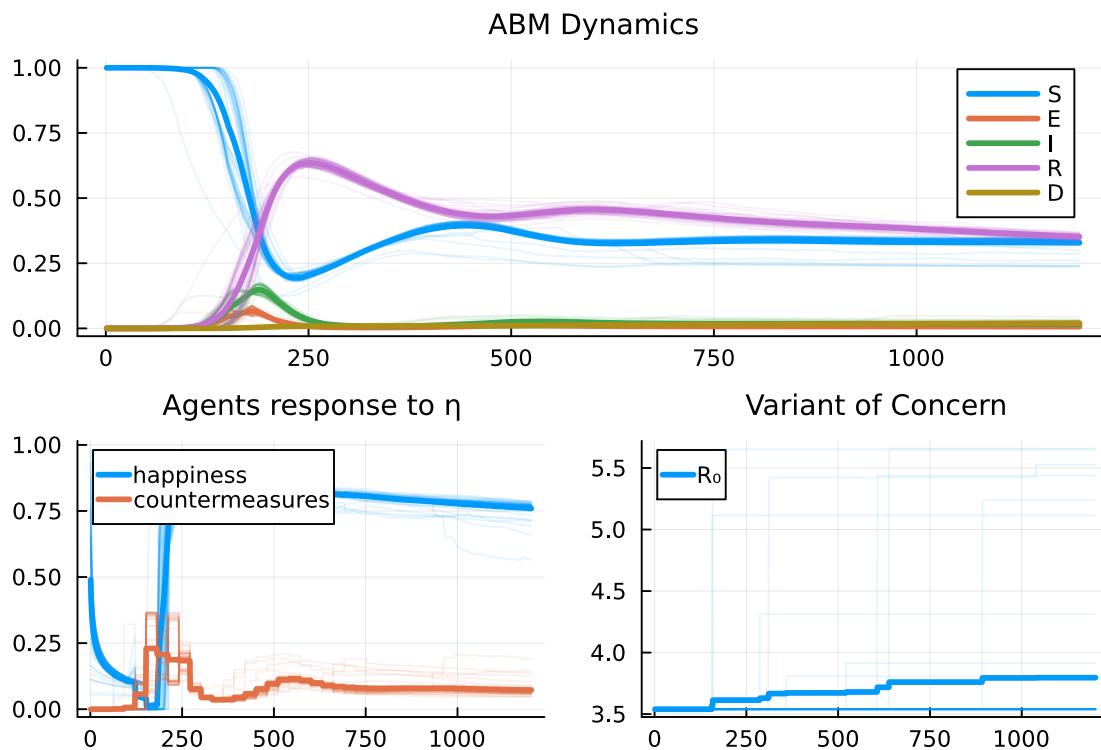
Figure 66: Definizione regole del modello del controller per le contromisure

I risultati ottenuti 67a 67b non si discostano troppo da quelli ottenuti tramite l'utilizzo del controllore con NeuralODE 40 implementato come mostra la figura 47 49. Si può notare come i risultati siano molto simili, e per questo è stato deciso di mantenere l'utilizzo dell'implementazione custom su quella della libreria Ipopt, in quanto questa scelta sembrava portare un aumento prestazionale non troppo significativo, ma sicuramente incoraggiante.

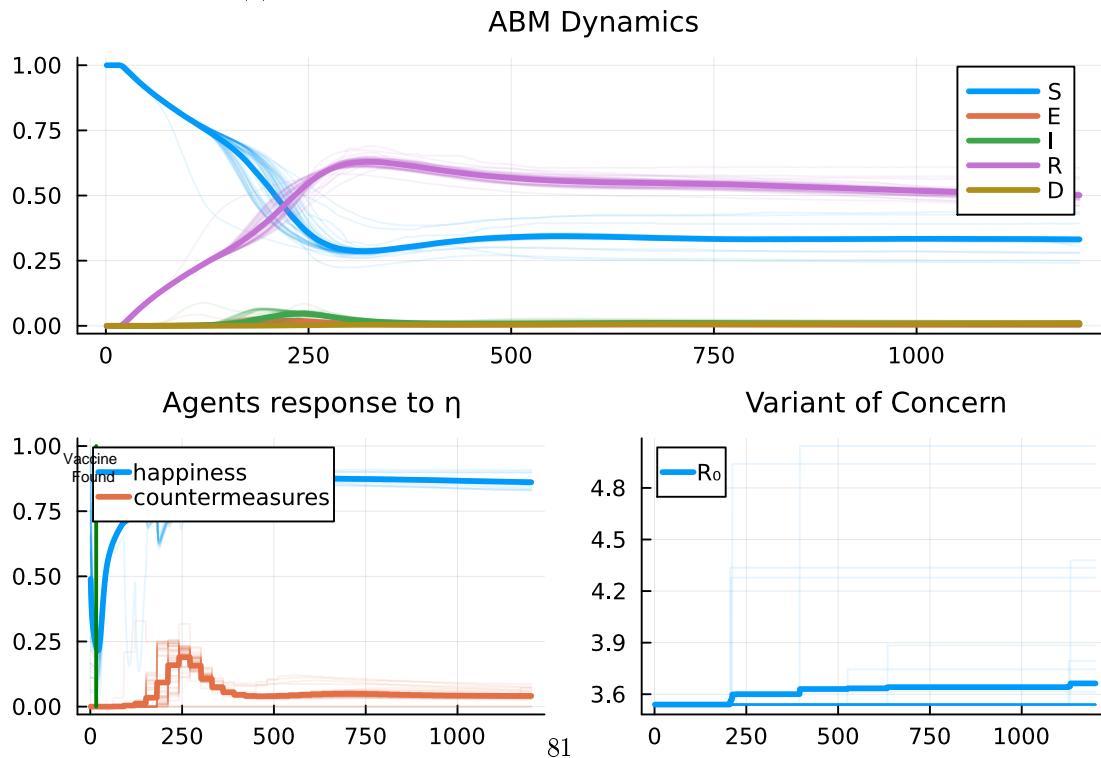
Da notare come la possibilità di convertire il codice da **CPU based** a **GPU based** tramite l'ausilio di poche semplici istruzioni, sia qualcosa che è stato preponderante nella scelta di non utilizzare la

libreria esterna Ipopt, soprattutto in vista di possibili sviluppi futuri legati al miglioramento delle performance del codice in termini di risorse computazionali e soprattutto temporali [19].

La scelta di utilizzare un approccio ibrido come controllore piuttosto che un approccio definito tramite una suite di ottimizzazione non lineare, è stato optato anche in vista del fatto che la definizione del controllore diveniva più semplice rispetto alla controparte Ipopt. Questa semplicità tuttavia è data dal fatto che non si ha diretto controllo sulla funzione da ottimizzare in quanto è mascherata da una rete neurale, cosa che con il controllore esplicito di Ipopt era possibile fare definendo vari vincoli sul sistema in questione.



(a) Risultato applicazione controllore tramite la suite Ipopt



(b) Risultato applicazione controllore tramite la suite Ipopt