



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 4, 2023

Anno Accademico 2022-2023

Abstract

Lo scopo di questa tesi e' 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 e' quello di avere un approccio ibrido e dinamico ai due mondi, dai quali e' possibile trarre il massimo vantaggio. L'approccio generale e' 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 dell'affidabilita', il modello e' 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 infettivita' R_0 ; per evitare di applicare livelli di contromisure al lato pratico insostenibili, e' stato inserito un valore di contenimento chiamato *happiness*, il quale agisce come una funzione di costo e controllo delle contomisure stesse.

Contents

1	Introduzione	1
2	Stato dell'Arte	4
2.1	Epidemiologia	4
2.2	Causalità	6
2.3	Modelli Compartmentali	7
2.3.1	Derivazioni del modello SIR	9
2.3.2	Modelli Deterministici	13
2.3.3	Modelli Stocastici	16
2.4	Modelli ad Agente	17
2.4.1	Discretizzazione	19
2.5	Julia	21
2.5.1	Agents.jl	21
2.5.2	SciML.ai	24
2.5.3	SafeBlues	35
3	Metodi e Modelli	36
3.1	Approccio con Rete Sociale	36
3.1.1	Agente	38
3.1.2	Spazio e Modello	39
3.1.3	Funzione di avanzamento agente	41
3.1.4	Funzione di avanzamento modello	42
3.1.5	Grafici	46
3.2	Monitoraggio e Intervento	53
3.2.1	SciML.ai	53
3.2.2	Grafici	56
4	Risultati Ottenuti	57
5	Sviluppi Futuri	58
5.1	Altre epidemie	58
5.1.1	Ebola	58
5.1.2	Aviaria	58
5.1.3	Influenza	58
5.2	Miglioramento integrazione SciML.ai	58
5.3	Accuratezza nella descrizione del modello	58
5.3.1	Happiness	58
5.3.2	Interventi localizzati	58
5.3.3	Migliore flessibilità generale	58
A	Approcci scartati	68
A.1	Modello ad Agente su spazio continuo	68
A.2	Modello ad Agente con spazio a grafo e modellazione singolo agente	69
A.3	Controllore Ipopt	72

List of Figures

1	Esempio di correlazione spuria da https://www.tylervigen.com/spurious-correlations	4
2	Un altro esempio di correlazione spuria da https://www.tylervigen.com/spurious-correlations	5
3	Esempio di paradosso di Simpson	6
4	Struttura modello SIR	7
5	Visualizzazione grafica di un modello SIR	8
6	Modello schematico SEIR	9
7	Esempio di modello SEIR preso dall'articolo [53]	10
8	Esempio di modello SEIR preso dall'articolo [33]	11
9	Modello SEIRS preso dall'articolo [8]	12
10	Esempio di ondate di infettività. Dati del Dipartimento di Protezione Civile Italiana	12
11	Grafico cumulativo dei dati della pandemia da COVID-19 in Italia. Dati ottenuti da Our World in Data	13
12	Esempio di modello SEIRV con stato隐式 per la condizione V	14
13	Esempio equilibrio endemico	15
14	Modello SIR stocastico	16
15	Schelling's model of segregation simulato tramite ABM	17
16	Rappresentazione schematica di un modello ad agente	18
17	Esempio di discretizzazione dei dati	19
18	Plancia di gioco di Risiko	20
19	Tabella comparativa	22
20	Rappresentazione di uno spazio a grafo	23
21	Metriche di distanza di una griglia	24
22	Esempio di Scientific Machine Learning	25
23	Esempio di funzionamento algoritmo di machine learning	26
24	Esempio di funzionamento di una rete neurale	27
25	Esempio grafico ODE	28
26	Esempio definizione ODE in Julia	29
27	Esempio risoluzione ODE in Julia	30
28	Esempio implementazione NN in Julia	31
29	Esempio implementazione ciclo di addestramento NN in Julia	31
30	Esempio implementazione NeuralODE tramite DiffEqFlux.jl	32
31	Tabella comparativa tra le varie implementazioni dei vari risolutori	33
32	Comportamento UDE nell'approssimazione di fenomeni non lineari	34
33	Esempio funzionamento SafeBlues	35
35	Esempio di grafo sociale	37
36	Codice Agente	38
37	Codice Modello	39
39	Matrice di migrazione	40
40	Definizione dei sistemi di ODE associati ad ogni agente	41
41	Comportamento agente	41
42	Funzione di avanzamento del modello	43
43	Funzione che si occupa di generare la VOC	44
44	Funzione che si occupa di simulare la ricerca di un vaccino e la sua successiva applicazione	45

45	Grafico cumulativo del risultato del modello senza intervento del controllore	46
46	Grafico delle mutazioni casuali del virus SARS-COV2 preso dall'articolo [51]	48
47	Grafico cumulativo del risultato del modello con intervento del controllore	49
48	Grafico cumulativo del risultato del modello con intervento del controllore tramite vaccino	50
49	Grafico cumulativo del risultato del modello con intervento del controllore tramite vaccino e metodi di prevenzione non farmaceutici	51
52	Definizione delle funzioni di ottimizzazione	54
53	Definizione delle funzioni di addestramento tramite due differenti ottimizzatori . . .	55
54	Esempio schematico del funzionamento di SINDy [14]	55
56	Esempio del modello modellato su spazio continuo	68
57	Esempio del comportamento delle curve nel modello continuo	69
58	Comportamento modello ABM su spazio a grafo al variare del parametro R_0	70
59	Comportamento modello SEIR al variare del parametro R_0	71
60	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	71
62	Comparison of Ipopt performance over various linear solvers using the two-dimensional partial differential equation test problem set. [75]	73
63	Definizione del controllore tramite Ipopt	74
64	Definizione regole del modello del controller	75
65	Definizione regole del modello del controller per le contromisure	75

List of Tables

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 [104] che a partire dalla metà del 14esimo secolo ritornò in Europa uccidendo venti milioni di persone in soli sei anni, l'epidemia di tifo [116] 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 [97] 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 [115], 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 [103] scoppiata 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 [57]. 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] [88]. 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 [106]. 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 [91]. 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 [109]. Dato lo scopo dell'epidemiologia, questa necessita di avvalersi di modelli matematici [73] che aiutano i ricercatori a trarre conclusioni sul sistema che analizza. Sistemi simili vengono definiti come complessi, [30] [48] 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) [86], 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 [81] [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 [47]. 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 [76] [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] [59] 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] [59].

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 [91]. Si occupa di analizzare le cause, il decorso e le conseguenze delle malattie.[30] [59]

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.

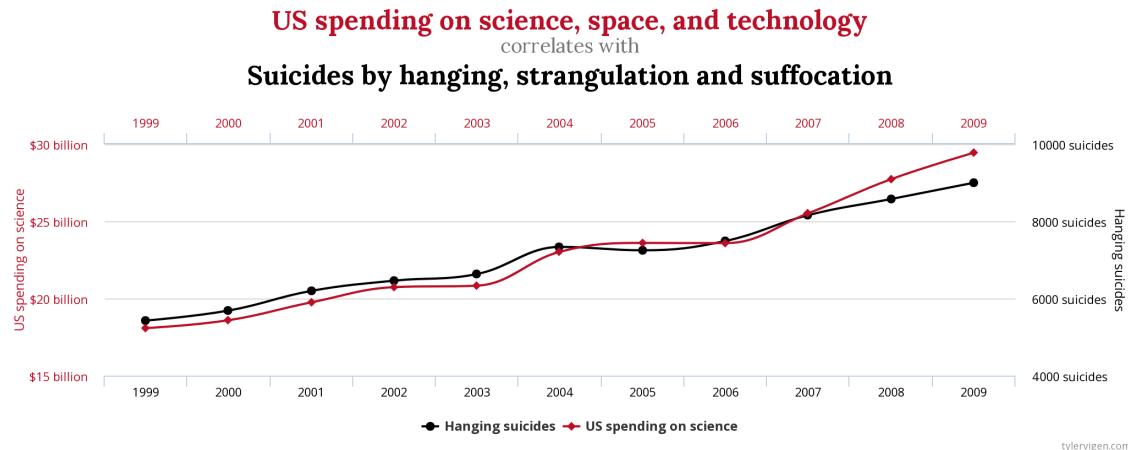


Figure 1: Esempio di correlazione spuria da <https://www.tylervigen.com/spurious-correlations>

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 respondabile di incitamento al suicidio. Ebbene non è così, questo è un caso di **relazione spuria** [114], 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 [87]. 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;

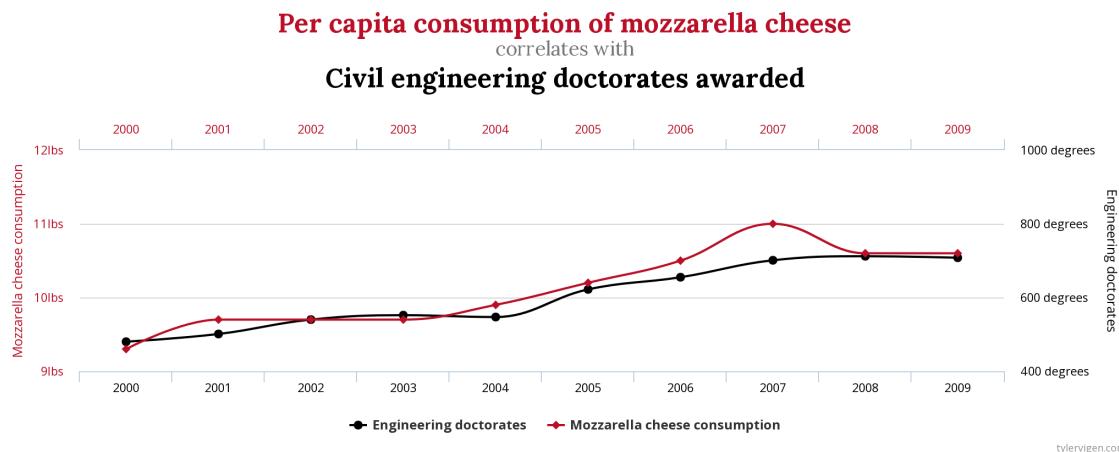


Figure 2: Un altro esempio di correlazione spuria da <https://www.tylervigen.com/spurious-correlations>

2.2 Causalità

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 [59].

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:

	Drug			No Drug		
	patients	recovered	% recovered	patients	recovered	% recovered
Men	87	81	93%	270	234	87%
Women	263	192	73%	80	55	69%
Combined data	350	273	78%	350	289	83%

Figure 3: Esempio di paradosso di Simpson

Questi risultati sembrano suggerire come la prescrizione di questa nuova medicina non aiuti i pazienti a stare meglio. Tuttavia questo risultato cade nel cosi detto *paradosso di Simpson* [108] 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.

E' 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 [70].

2.3 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 [86]. Questa tecnica di modellazione si applica anche ad altre branche della scienza, come ad esempio 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 comunque chiaramente distinti l'uno dagli altri.

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):

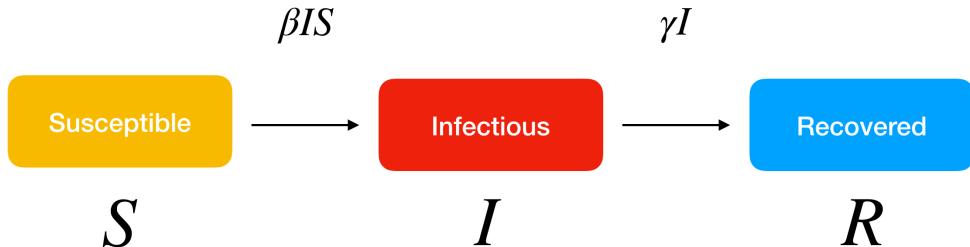


Figure 4: Struttura modello SIR

Questo modello è stato ideato all'inizio del 20esimo secolo, più precisamente nel 1917, 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.
- 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 docorso 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. Questo approccio si basa sull'utilizzo di un sistema di Equazioni Ordinarie Differenziali (ODE) [11]. Una ODE è un equazione differenziale, ovvero un equazione che lega una funzione incognita

alle sue derivate, che coinvolge una funzione di una variabile e le sue derivate di ordine qualsiasi. Questo oggetto viene utilizzato estensivamente in molti ambiti della scienza e in epidemiologia viene utilizzato per descrivere un sistema dinamico [92].

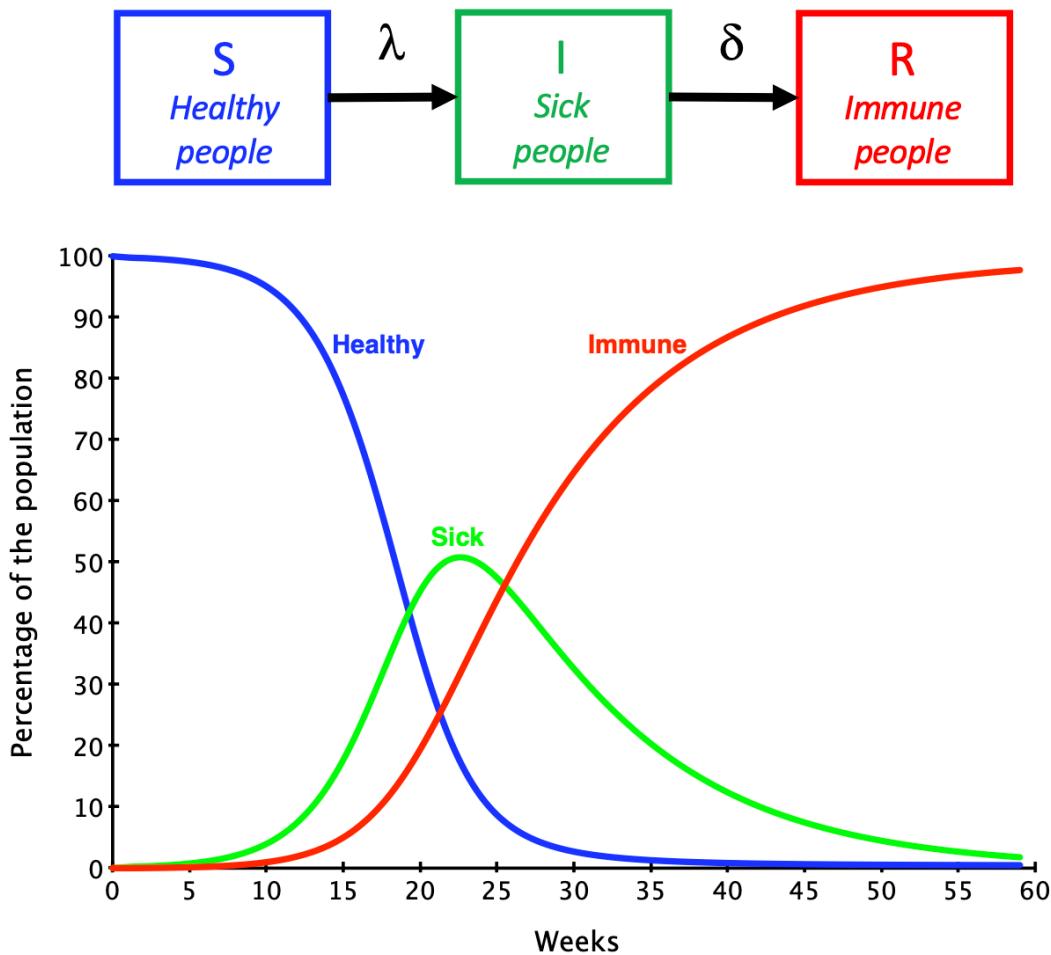


Figure 5: Visualizzazione grafica di un modello SIR

Come succede per la maggior parte di tutte le equazioni differenziali, queste non possono essere solitamente risolte in maniera esatta, e per questo ci si limita a studiarne il comportamento qualitativo della soluzione senza essere capaci di ottenere una'espressione analitica.

Nell'ambito epidemiologico tuttavia non sempre l'utilizzo di un sistema di ODE è preferito come metodo generale di modellazione di un sistema dinamico, in quanto il più delle volte questo sistema al suo interno utilizza un insieme di variabili rappresentanti processi stocastici, le quali è bene

mantenere tali. A questo scopo vengono utilizzati i sistemi di Equazioni Differenziali Stocastiche (SDE) [3].

Queste equazioni si basano sulla teoria del moto Browniano il quale descrive il movimento randomico delle particelle sospese all'interno di un medium [84]. In questo modo è possibile modellare in maniera più granulare ad esempio la diffusione di un agente patogeno tramite il medium aereo, come può essere il COVID-19.

2.3.1 Derivazioni del modello SIR

Con il tempo questo il modello SIR è stato espanso per tenere in considerazione comportamenti differenti sia della popolazione che delle malattie, andando a definire una moltitudine di modelli utili a differenti scopi. In epidemiologia il modello di riferimento maggiormente utilizzato è il modello SEIR (Susceptible, Exposed, Infectious, Recovered) con le sue varianti proprie di ogni approccio.

Questa tipologia di modello basa il proprio punto di forza sull'osservazione che da quando un individuo viene infettato tramite un agente patogeno a quando quest'ultimo diventa infettivo, passa un periodo di latenza in cui l'individuo non può ne infettare ne essere infettato. Questo periodo viene anche conosciuto come *periodo di incubazione* [96]. Con questa conoscenza pregressa è possibile sviluppare modelli e policy che tenendo conto di questo comportamento lo sfruttino per arginare l'epidemia.

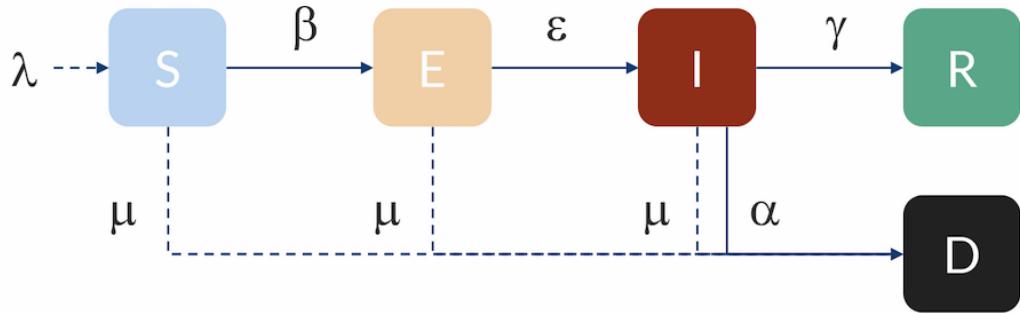


Figure 6: Modello schematico SEIR

Alcuni modelli definiscono i propri stati in maniera da considerare come stato interno al sistema anche l'agente patogeno, così da poter modellare e simulare l'andamento dell'infettività della pandemia in relazione alle contromisure prese, siano esse farmaceutiche o non. Ne è un esempio il modello proposto da [53] nel quale il modello viene proposto con l'idea di incorporare le misure di distanziamento sociale come variabili per misurare la loro efficacia contro la recente pandemia da COVID-19.

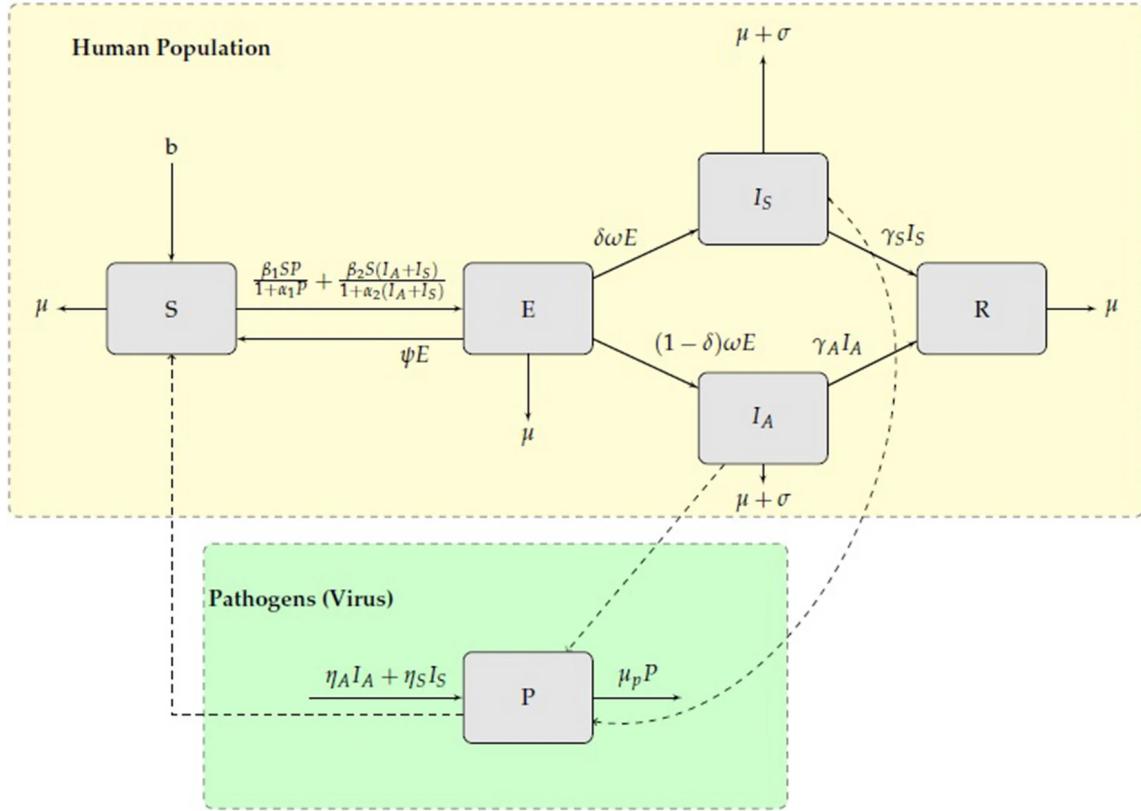


Figure 7: Esempio di modello SEIR preso dall'articolo [53]

Altri modelli, come quello proposto da [33] mantengono la stessa filosofia, ovvero quella di analizzare l'efficacia delle misure di prevenzione non farmaceutiche sull'andamento di un'epidemia, ma non modellano esplicitamente l'agente patogeno come stato del modello, bensì variando i parametri di infettività e contagio, arrivano allo stesso risultato. Un'altra differenza tra i due approcci è quella della tipologia di equazioni differenziali utilizzate, [53] hanno utilizzato delle ODE mentre [33] delle SDE.

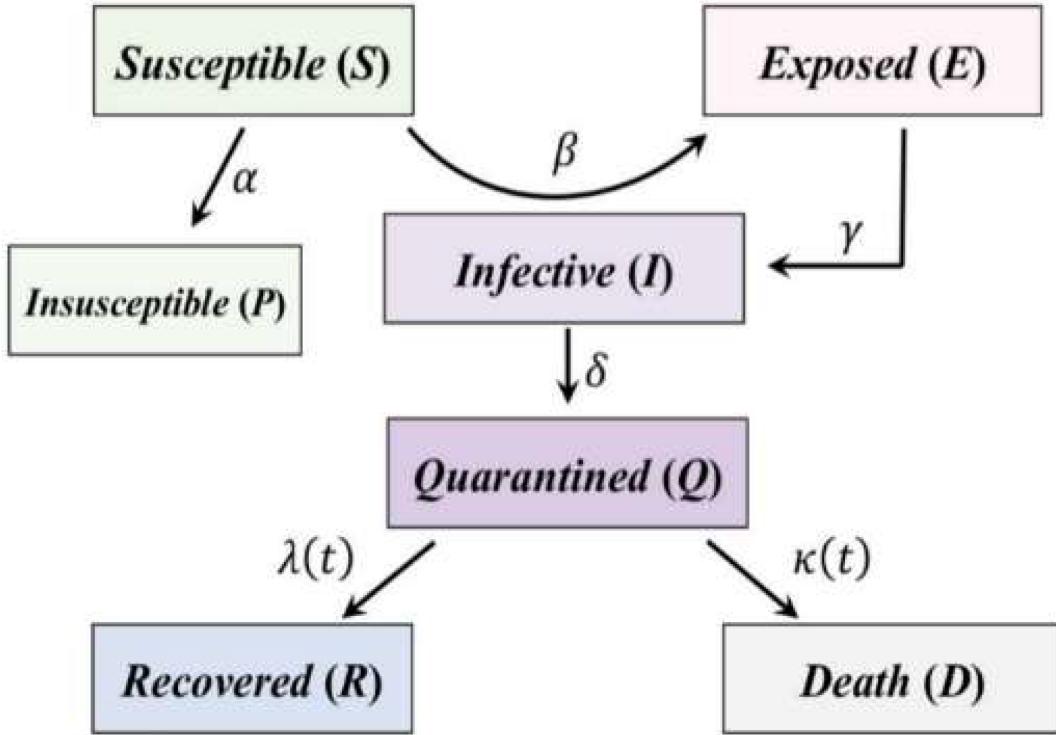


Figure 8: Esempio di modello SEIR preso dall'articolo [33]

Il motivo per cui viene utilizzato il modello SEIR come base è perché permette di modellare una caratteristica intrinseca di una malattia infettiva, ovvero il periodo di latenza che un individuo appena infettato ha prima di diventare infettivo a sua volta e mostrare i sintomi di infezione. Questo permette di osservare quanto le misure di sicurezza e prevenzione non farmaceutiche sono efficaci sulla popolazione tenendo in considerazione un tempo di ritardo intrinseco nel feedback tra l'attuamento delle misure di prevensione e i risultati positivi di queste ultime.

Una delle modifiche più utilizzate a questo modello è quella di avere un sistema ciclico, ovvero in cui gli individui che entrano nello stato R non diventano immuni alla malattia a tempo indefinito, ma perdono questa loro caratteristica di immunità dopo un dato periodo di tempo. Questo permette di modellare con più accuratezza le malattie infettive stagionali come ad esempio la comune influenza o il raffreddore, oppure mostrare l'andamento ad ondate di altre malattie che hanno la caratteristica di mutare molto velocemente, come è stato per il COVID-19 e le sue innumerevoli varianti.

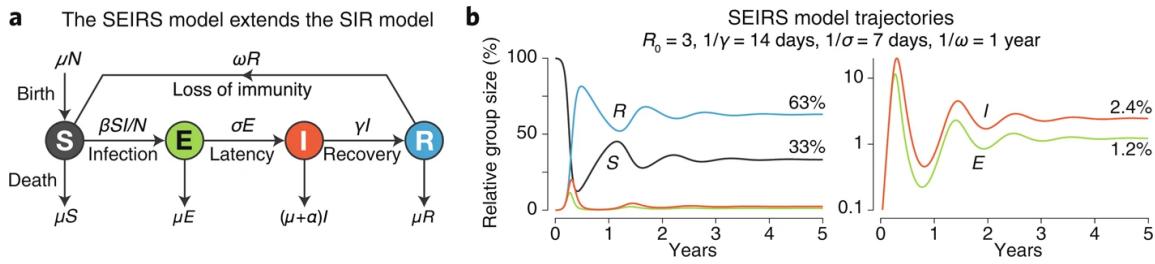


Figure 9: Modello SEIRS preso dall'articolo [8]

Questa variante denominata SEIRS permette invece di osservare, non solo l'andamento e l'efficacia delle contromisure non farmaceutiche, ma anche di quelle farmaceutiche, come ad esempio i vaccini; o più in generale l'andamento della così detta immunità di gregge [8].

Rimanendo sull'idea di voler analizzare l'efficacia di un vaccino, una modifica comune al modello SEIR è quella legata all'aggiunta dello stato V, Vaccinated, come stato esplicito oppure oppure implicito al modello. Questa variazione permette di modellare con più attenzione l'efficacia di un vaccino una volta introdotto all'interno della popolazione, ma più in generale permette di osservare l'efficacia di una politica di vaccinazione in relazione al numero di vaccinazioni effettuate in un determinato periodo di tempo. Questo viene solitamente affiancato con un modello ciclico, così da poter osservare come bisogna modificare le proprie politiche vaccinali in vista di ondate cicliche più o meno intense di infezioni.

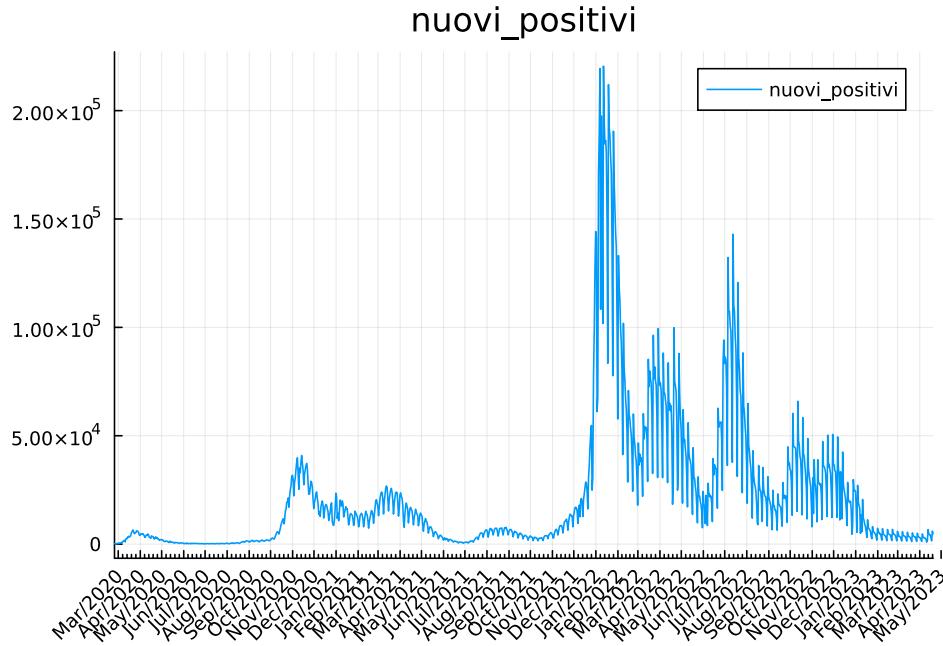


Figure 10: Esempio di ondate di infettività. Dati del Dipartimento di Protezione Civile Italiana

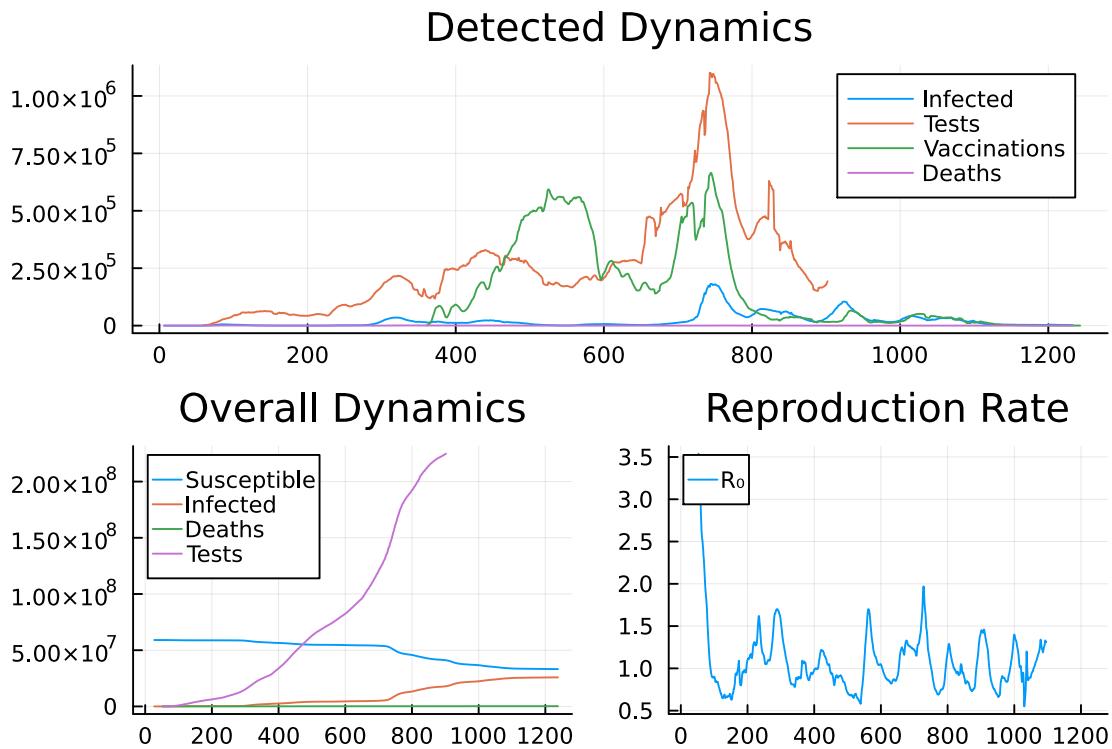


Figure 11: Grafico cumulativo dei dati della pandemia da COVID-19 in Italia. Dati ottenuti da Our World in Data

Ai fini pratici di una simulazione avere uno stato esplicitamente definito oppure ricavabile dalle probabilità di transizione degli altri stati è pressoché indifferente, e potrebbe essere richiesta una differenziazione solamente in caso in cui si avrebbe una differenza sostanziale tra lo stato R e V , ad esempio in termini di protezione dalla malattia, durata immunità etc.... .

Non essendoci un numero massimo di stati, e quindi di equazioni, utilizzabili all'interno del modello, ogni individuo è libero di definire un numero di equazioni arbitrario che rispecchia la sua idea di modellazione del sistema. Ne è un esempio il modello riportato in [32].

Come precedentemente introdotto esistono due grandi famiglie di modelli per la simulazione, e sono rispettivamente la famiglia di modelli deterministici e quella di modelli stocastici.

2.3.2 Modelli Deterministici

I modelli deterministici vengono principalmente utilizzati per la loro immediatezza e riproducibilità. Infatti un modello deterministico, una volta impostati i parametri necessari riprodurrà sempre lo stesso risultato. Questo tipo di approccio, seppur utilizzato in larga scala come ad esempio da [8] [53], [32] si basa su delle assunzioni molto forti che non sempre rispecchiano la realtà.

Infatti i modelli deterministici hanno il grosso problema di essere affidabili solamente nel caso in cui vi siano dati sufficientemente grandi, cosa che non sempre è possibile avere [86]. Essendo modelli

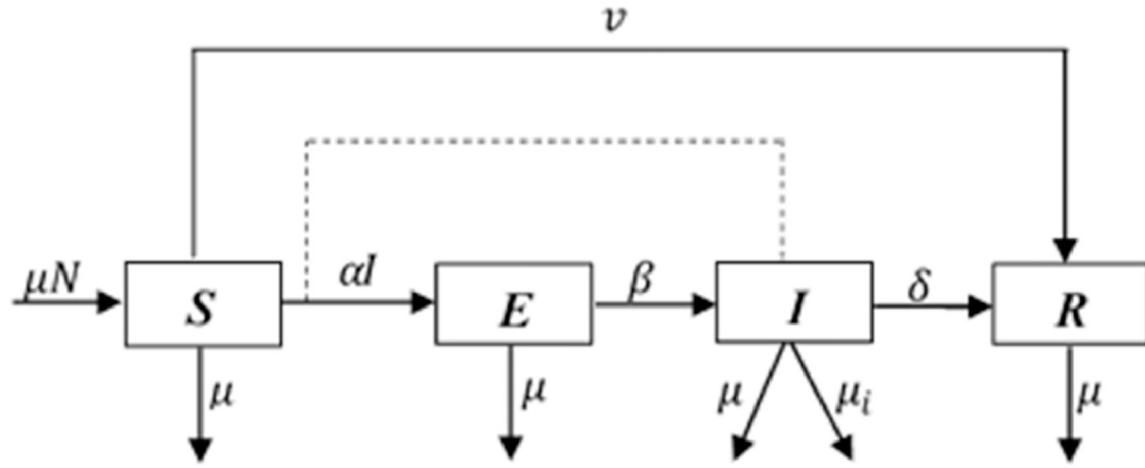


Figure 12: Esempio di modello SEIRV con stato隐式 per la condizione V

di tipo deterministico, avendo dei parametri di infettività maggiori di zero, con un numero di individui infetti anch'esso maggiore di zero, si tenderà ad avere nel lungo periodo un andamento di equilibrio endemico derivato dalle equazioni e dal modello utilizzato. Questo comportamento però non sempre rispetta la realtà, ma come precedentemente accennato, in casi in cui si hanno grandi quantità di dati legati principalmente alla popolazione, questi modelli si comportano in maniera affidabile.

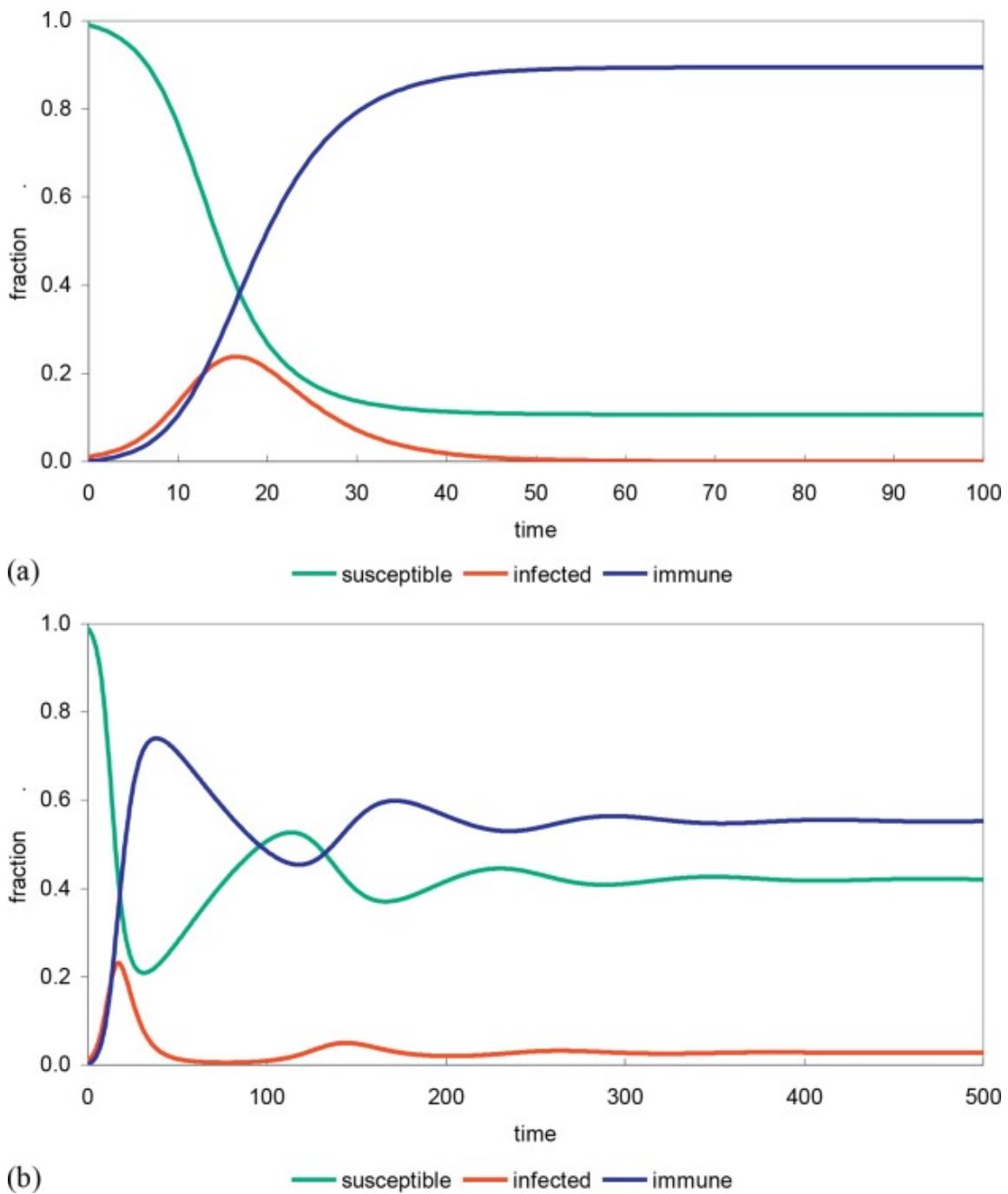


Figure 13: Esempio equilibrio endemico

2.3.3 Modelli Stocastici

I modelli stocastici, seppur più complessi e non determinabili a priori, permettono una modellazione più veritiera e simile alla realtà in quanto tengono in considerazione variazioni randomiche che possono capitare durante il decorso di una pandemia. Tuttavia questa loro caoticità richiede che per ottenere risultati robusti debbano essere eseguiti e computati molteplici volte, e la media dei loro risultati è il valore vero da tenere in considerazione. Questi modelli sono stati applicati durante la pandemia da COVID-19 come ad esempio da [33].

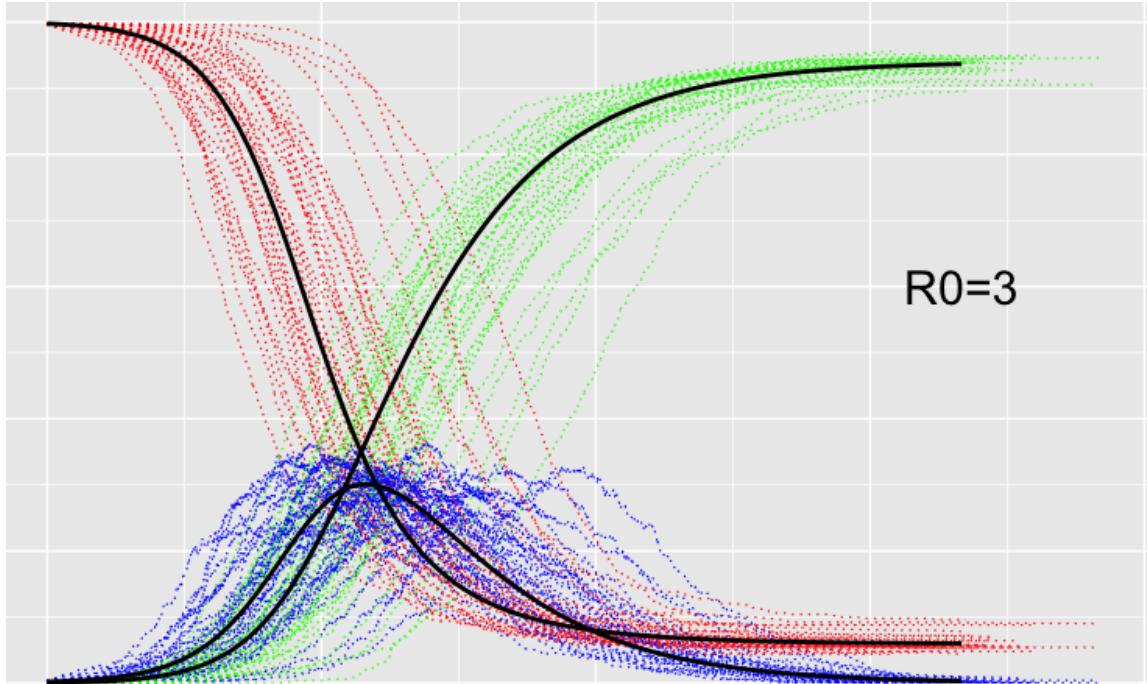


Figure 14: Modello SIR stocastico

E' immediato notare come il comportamento delle curve sia imprevedibile se preso singolarmente, e che non sempre esiste uno stato di equilibrio endemico chiaro e definito come quello ottenibile da un modello deterministico. Ciò nonostante effettuando molte simulazioni è possibile vedere come il comportamento generale del modello sia comunque simile a quello di un modello deterministico.

2.4 Modelli ad Agente

Un modello ad agente e' 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] [81].

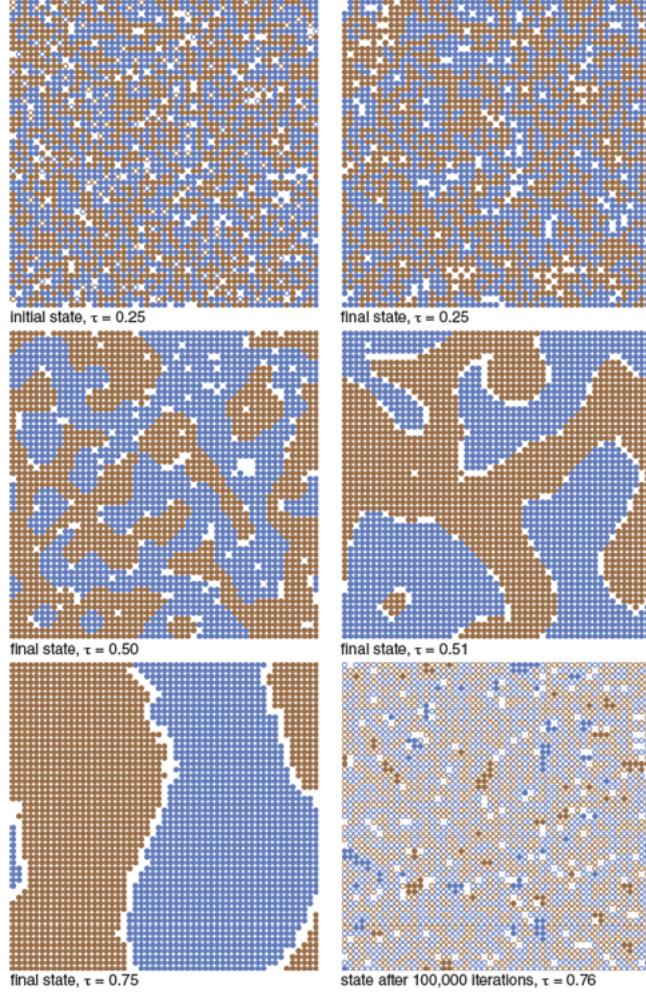


Figure 15: Schelling's model of segregation simuloato tramite ABM

L'utilizzo di modelli ad agente in epidemiologi e' una tecnica nota che da piu' di un decennio viene impiegata per simulare e comprendere le situazioni e i problemi piu' disparati, tutti pero' generalmente accomunati dal fatto che come parametro portante vi sia il comportamento umano [36] [25] [76] [7]. Uno dei parametri piu' 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 piu' specifiche ma che possono essere raggruppate come macro categoria se si scende a

compromessi.

Questa specifica e' importante in quanto uno delle sfide piu' grandi che il mondo della simulazione, e quindi quello epidemiologico devono affrontare e' 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 [74]. Non soltanto, un'altra sfida e' 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 [47].

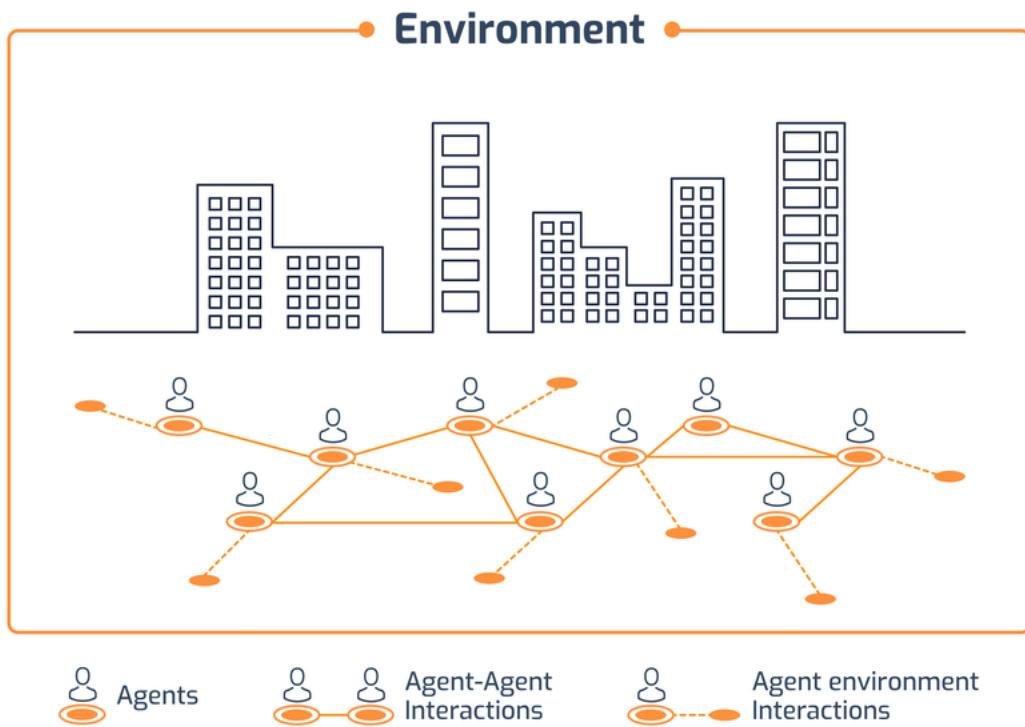


Figure 16: Rappresentazione schematica di un modello ad agente

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 ibrido [52] (ibridato con delle ODE oppure delle Partial DE), 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 piu' stocastiche e imprevedibili come il comportamento umano. L'ambiente di lavoro simulato era generalmente un ambiente controllato che poteva essere una citta' come in [60],

Un'altro obiettivo e' quello di osservare l'impatto degli interventi non farmaceutici sulla popolazione come riporta [56] [45]. Altri ancora invece utilizzano l'approccio simulativo tramite modelli ad agente per estrarre delle ODE tramite l'analisi del modello, come riportato da [54].

2.4.1 Discretizzazione

La tematica della discretizzazione e' una delle proprietà fondamentali e al contempo uno dei problemi atavici della simulazione. Il mondo in cui viviamo e' 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 realta' alla simulazione, andando inequivocabilmente a perdere informazioni nel processo [47].

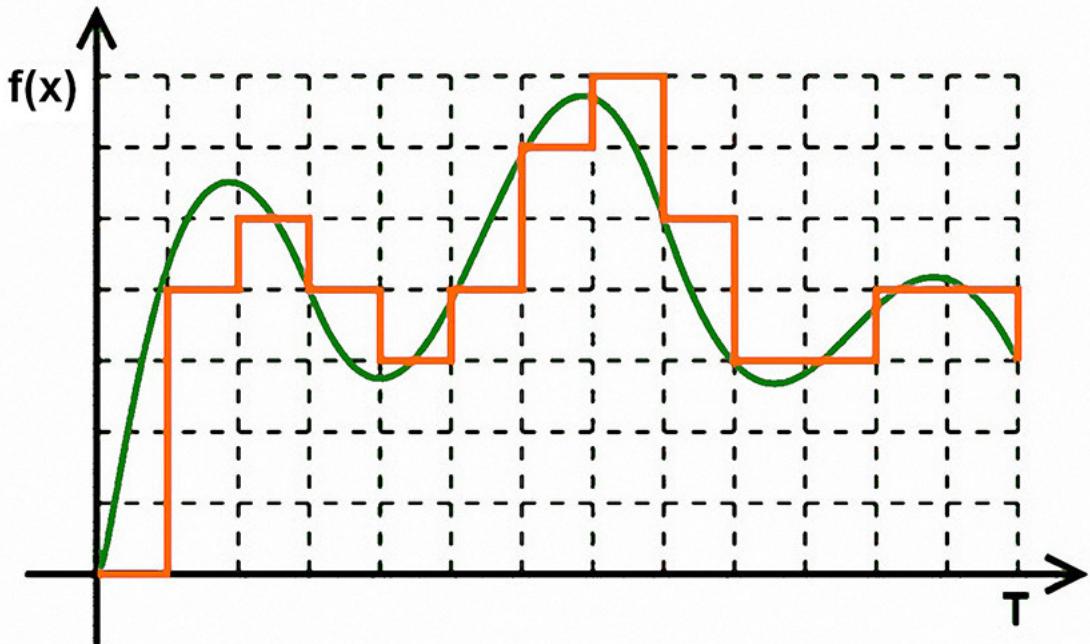


Figure 17: Esempio di discretizzazione dei dati

Il processo di discretizzazione in una simulazione puo' 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 piu' accurata un insieme di dati continuo, permettendo all'utente di utilizzare parole chiave come ad esempio `ContinuousSpace`. Quello che praticamente viene fatto e' creare un ambiente per il modello che e' il piu' preciso e granulare possibile, per cui servono allo scopo fintanto che quest'ultimo non richiede una maggiore precisione.

Il processo di discretizzazione comunque non e' sempre negativo, in quanto alcuni problemi possono essere simulati in maniera estremamente fedele anche effettuando questi accorgimenti, e anzi alle volte non e' perfino necessario avere una precisione troppo alta per la simulazione di determinati eventi.

Se ad esempio si volesse simulare tramite un agente una partita a Risiko, non e' necessario richiedere uno spazio e un tempo continuo della simulazione, in quanto questi possono essere rimpiazzati dalla loro controparte discreta che sono caselle e turni.



Figure 18: Plancia di gioco di Risiko

Se si volesse simulare il traffico aereo si potrebbe utilizzare uno spazio a grafo dove ogni nodo rappresenta un aeroporto e gli archi rappresentano la tratta in partenza o in arrivo. Anche in questo caso l'applicazione della discretizzazione non sarebbe qualcosa di problematico.

2.5 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 [98].

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

2.5.1 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].

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

Figure 19: Tabella comparativa

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 [66], branca che soprattutto grazie alla pandemia da Covid-19 ha visto un sempre piu' 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 e' la piu' concreta implementazione, nonche' 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 puo' contenere una quantita' 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.

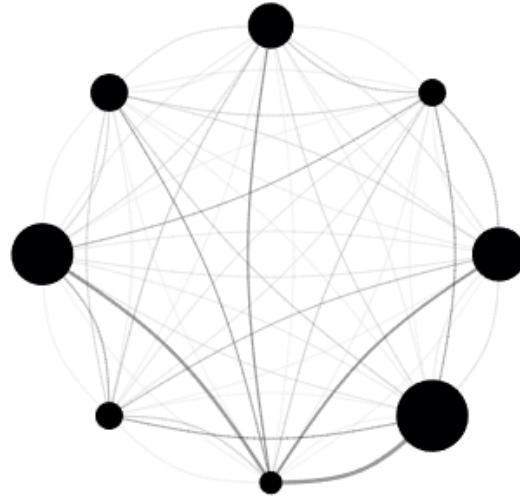


Figure 20: Rappresentazione di uno spazio a grafo

- 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*.

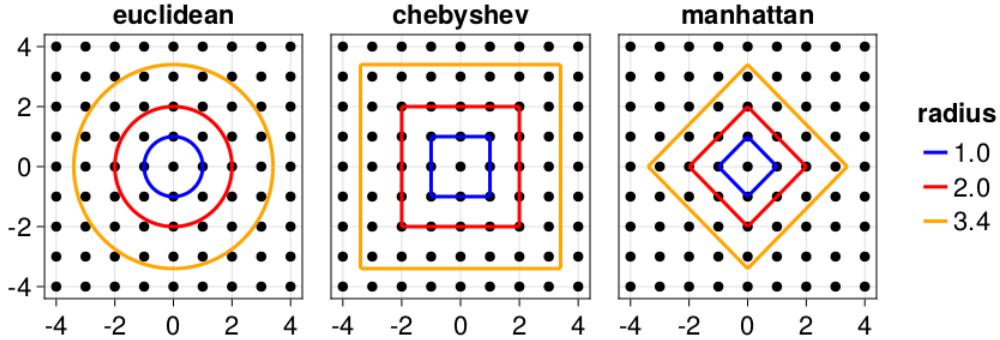


Figure 21: Metriche di distanza di una griglia

- spazio continuo: un *ContinuousSpace* rappresenta uno spazio di dimensione $D \in (0, \infty)$. E' fortemente consigliato di attribuire ad un agente all'interno di questo spazio due caratteristiche fondamentali, una posizione e una velocita'. Questa tipologia di spazio permette di rappresentare delle proprieta' 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 sara' sempre *Euclidian*. Per velocizzare il calcolo della posizione degli agenti, viene effettuata una discretizzazione implicita dello spazio, ma questa puo' essere forzata a rimanere nello spazio continuo ottenendo un calo di prestazioni.
- spazio misto: un *OpenStreetMapSpace* rappresenta una mappa come un'entita' continua che preferisce l'accuratezza alle prestazioni. La mappa viene rappresentata come un grafo connesso. I nodi non rappresentano necessariamente intersezioni.

2.5.2 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 [66] [63] [65].

Il principale utilizzo che e' 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* [66] uniti a metodi di *Machine Learning* (ML) [58] [40] [41] 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) [63] [65].

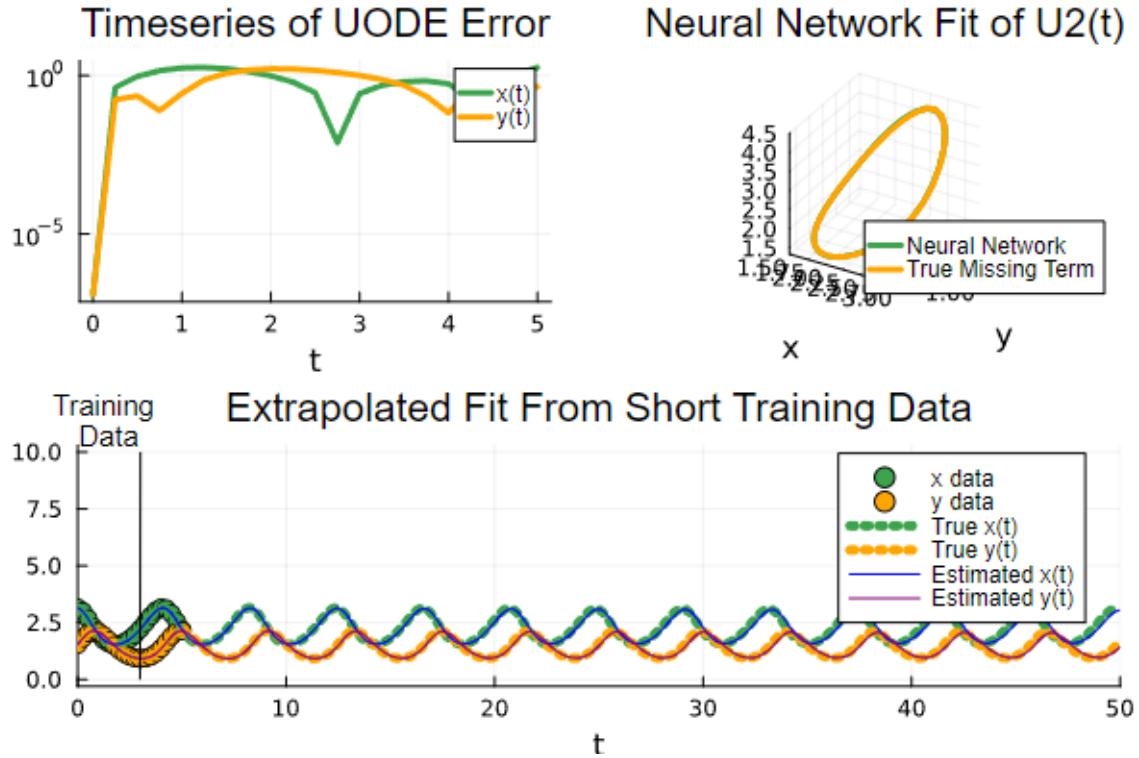


Figure 22: Esempio di Scientific Machine Learning

Equazioni Neurali Differenziali

Da quando l'articolo [16] e' 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. [46] [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 e' conosciuta a priori per cui non e' possibile applicare questo approccio. A tal proposito si puo' 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 e' generalmente

definito *modello*. L'idea alla base e' avere un periodo di allenamento (training) in cui si tenta di aggiustare i parametri (iperparametri) del modello cosi' 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 e' semplicemente un insieme di trasformazioni non lineari.

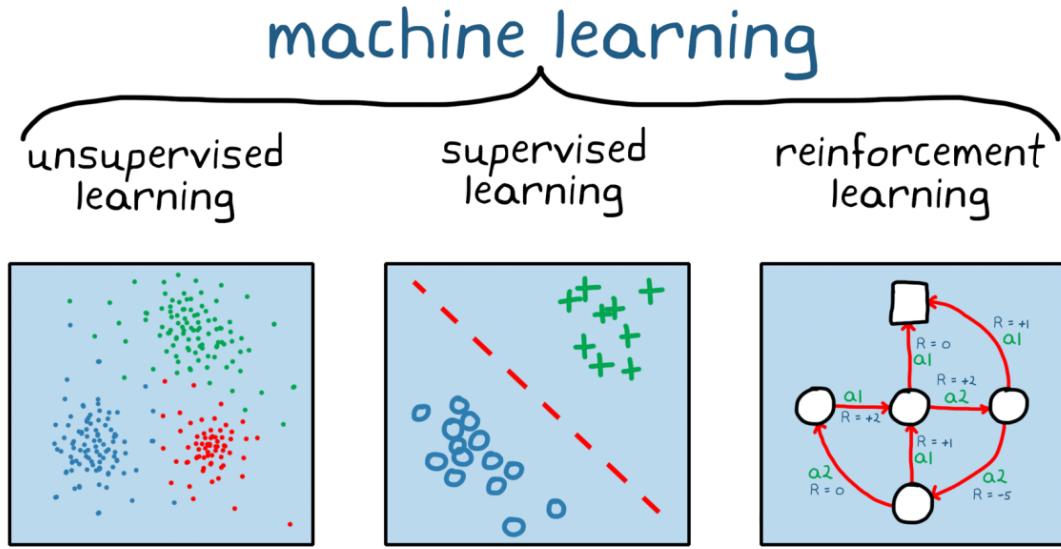


Figure 23: Esempio di funzionamento algoritmo di machine learning

L'utilizzo del machine learning e' molto interessante in quanto il concetto alla base e' estremamente semplice ma e' 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 e' 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 profondita' 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 e' 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 [117] afferma che per un numero sufficientemente grande di parametri o di strati e' possibile approssimare qualsiasi funzione non lineare in maniera sufficientemente precisa.

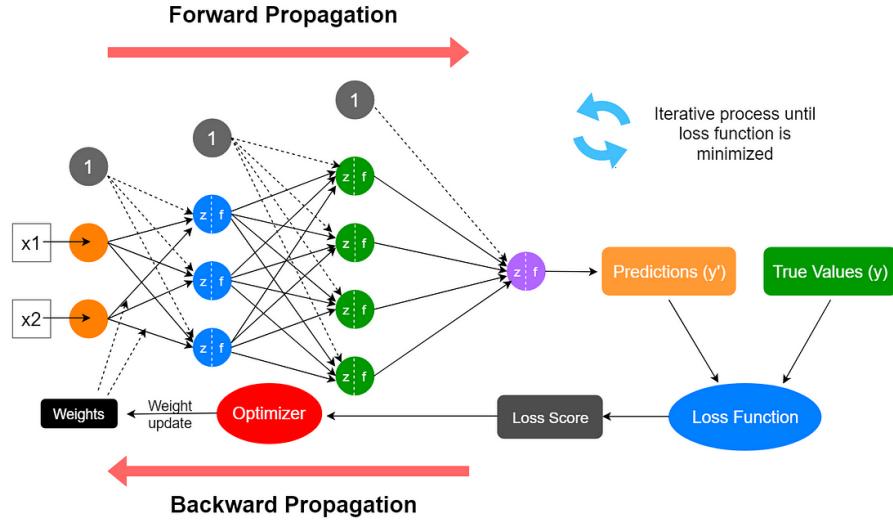


Figure 24: Esempio di funzionamento di una rete neurale

Questo approccio tuttavia necessita di apprendere ogni aspetto della trasformazione non lineare direttamente dai dati a disposizione. In molti casi tuttavia, non e' possibile conoscere l'intera equazione non lineare, ma magari se ne conosce la struttura generale. Il modo per definire matematicamente questo tipo di approccio e' tramite le equazioni differenziali. Il modo piu' immediato e' 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 e' 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 e' essenzialmente un insieme di equazioni che descrive il cambiamento delle cose e dove queste si troveranno ad un determinato periodo di tempo e' la soluzione all'equazione differenziale.

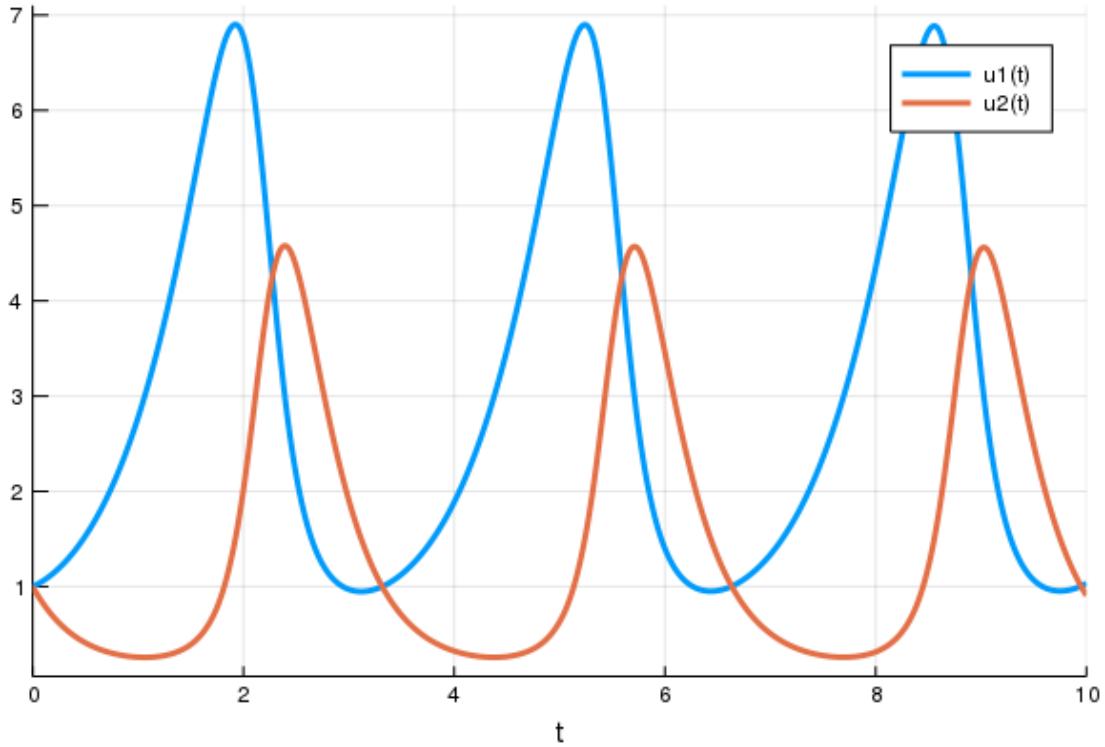


Figure 25: Esempio grafico ODE

Questo metodo e' 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 peculiarita' dei modelli di machine learning e' che sono affamati di dati e richiedono un insieme di dati su cui allenarsi molto grande, e per questo l'utilizzo delle equazioni differenziali e' divenuto un opzione molto interessante per specificare la nonlinearita' 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 e' uno dei metodi per mettere in relazione questi due mondi: quello del machine learning e quello delle equazioni differenziali. L'approccio generale e' quello di apprendere non tanto la trasformazione non lineare che vi e' tra i dati, quanto la struttura della trasformazione non lineare. Percio' invece che avere il modello $y = ML(x)$ avremo il modello $y' = ML(x)$ e si tenta successivamente di risolvere l'equazione differenziale associata.

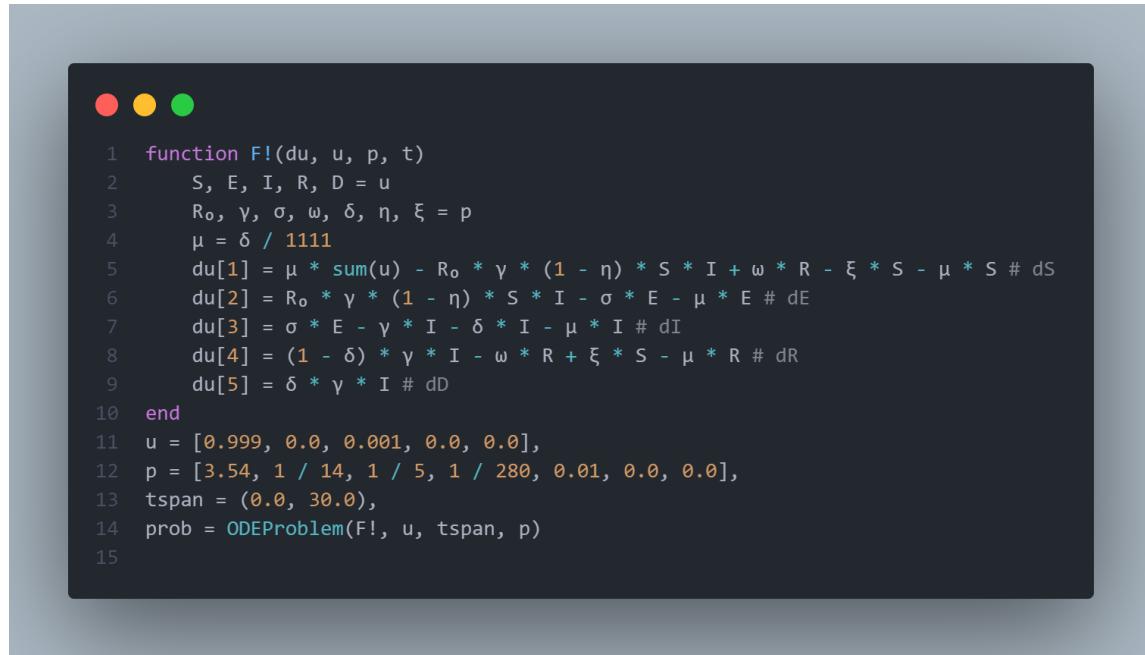
Principalmente il motivo per cui si vuole utilizzare questa tipologia di approccio e' quella per cui, definendo un modello in questo modo e utilizzando un risolutore estremamente semplice e prono ad errori come il metodo di Eulero [93] si ottengono risultati equivalenti ad una ResNet [38]. L'idea alla base e' quella che invece di modellare una rete neurale con sempre piu' strati, e quindi sempre piu'

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

Questo tipo di approccio e' *memory efficient*, ha la capacita' di gestire *dati irregolari* con forti priori sullo spazio del modello, ha un elevata capacita' 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 e' 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 $u' = f(u, p, t)$, provvedendo a fornire un insieme di condizioni iniziali u_0 , un periodo di tempo t e un insieme di parametri p .



```
1  function F!(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(F!, u, tspan, p)
15
```

Figure 26: Esempio definizione ODE in Julia

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

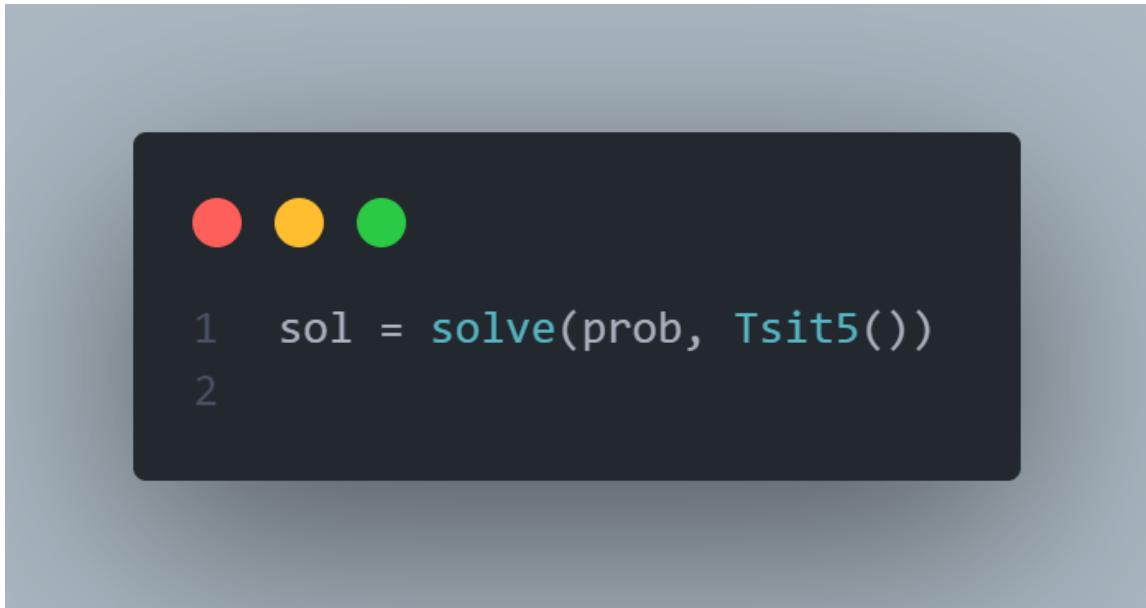


Figure 27: Esempio risoluzione ODE in Julia

Si possono specificare molteplici metodi per la risoluzione del sistema, in questo caso il metodo **Tsit5** [77] e' 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 piu' 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 e' definito. Un layer e' sostanzialmente una *funzione differenziabile* che prende come input un vettore di dimensione n e restituisce un nuovo vettore di dimensione m . Appare come i risolutori di DE rientrano anche loro nella categoria di funzioni differenziabili, il che significa che e' possibile inserirli direttamente all'interno di un programma differenziabile piu' grande. Questo programma puo' 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 parameters, 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(pred[3, :]) + sum(pred[5, :]) / sum(pred[6, :])
12 end

```

Figure 28: Esempio implementazione NN in Julia

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

```

1 losses = Float64[]
2 callback = function (p, l; loss_step=loss_step)
3     push!(losses, l)
4     if length(losses) % loss_step == 0
5         println("Current loss after $(length(losses)) iterations: $(losses[end])")
6     end
7     return false
8 end
9 adtype = Optimization.AutoZygote()
10 optf = Optimization.OptimizationFunction((x, p) -> loss(x), adtype)
11 optprob = Optimization.OptimizationProblem(optf, ComponentVector{Float64}(parameters))
12
13 res1 = Optimization.solve(
14     optprob,
15     ADAM(),
16     callback=callback,
17     maxiters=maxiters
18 )

```

Figure 29: Esempio implementazione ciclo di addestramento NN in Julia

E' importante puntualizzare come il sistema non stia apprendendo una soluzione all' equazione differenziale. Piuttosto cio' che sta imparando e' il sistema di ode da cui la soluzione e' generata.

In questo caso la Neural ODE impara una rappresentazione compatta di come la serie di dati si comporta nel tempo, e puo' facilmente estrapolare cosa potrebbe succedere con differenti condizioni iniziali. Non soltanto, il metodo e' molto flessibile per un approccio simile. 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 30: Esempio implementazione NeuralODE tramite DiffEqFlux.jl

Backpropagation tramite ODE solver

Il cuore di ogni rete neurale e' l'abilita' di propagare all'indietro [83] lungo tutta la rete le derivate, con lo scopo di calcolare il gradiente [94] della funzione di perdita rispetto ai parametri della rete. Percio' 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 piu' comune e' tramite un approccio di analisi di sensitivita' (adjoint). L'analisi di sensitivita' 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) e' 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 e' definita da un *polinomio lineare* e la sua derivata e' un'equazione dalla 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* puo' essere generalmente approssimato con la controparte lineare cosi' da ottenere una soluzione piu' 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* [66] [67] [50] [35], i risolutori sono molteplici e permettono grande dinamicita' e affidabilita' nonche' elevate prestazioni durante l'utilizzo, questo in relazione anche ai risolutori dei piu' conosciuti linguaggi di programmazione come ad esempio: **MATLAB**, **SciPy**, **R**, **C** e **C++** e altri.

La figura riportata sotto 31 [61] 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	dsolve	DifferentialEquations.jl	Sundials	Holger	ODEPACK/NELIBPAC	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	Excellent	Good	Good	Good	Good	Good	Fair	Fair	Fair	Good
Tweakability	Fair	Fair	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 and AutoDifferentiation	None	Name	Name	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	Name	Name	Excellent	None	None	None	None	None	None	Excellent	Excellent	Excellent	Excellent
Control Over Linear/Nonlinear Solvers	None	Poor	Name	Excellent	Excellent	Good	Depends on the solver	Name	Name	Name	Name	Fair	Name	Name
Built-in Parallelism	None	Name	Name	Excellent	Excellent	None	None	None	None	None	Fair	Name	Name	Name
Differential-Algebraic Equation (DAE) Solvers	Good	Name	Good	Excellent	Good	Excellent	Good	None	Fair	Good	Name	Good	Good	Good
Implicit-Defined DAE Solvers	Good	Name	Excellent	Fair	Excellent	None	Excellent	None	None	None	None	Good	None	None
Constant-Log-Delay Differential Equation (DDO) Solvers	Fair	Name	Poor	Excellent	None	Good	Fair (via DDVERK)	Name	None	None	None	None	Good	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	Name	Poor	Excellent	None	Excellent	Good	Name	None	None	None	None	Name	Excellent
Stochastic Differential Equation (SDE) Solvers	Poor	Name	Name	Excellent	None	None	Good	Name	None	None	None	None	Fair	Poor
Specialized Methods for 2nd Order ODEs and Hamiltonians (and Symplectic Integrators)	None	Name	Name	Excellent	None	Good	None	None	None	None	None	Fair	Good	Name
Boundary Value Problem (BVP) Solvers	Good	Fair	None	Good	None	None	Good	None	None	None	None	Good	Fair	Fair
GPU Compatibility	None	Name	Name	Excellent	Good	Name	Name	Name	Name	Name	Good	Name	Name	Name
Analytic Actions (Sensitivity Analysis, Parameter Estimation, etc.)	None	Name	Name	Excellent	Excellent	Name	Good (for some methods like NAPTR)	None	Fair	Good	Name	None	Excellent	Name
**** This timing includes JIT compilation with Numba, see https://github.com/JuliaDiffEq/SciPyODEEq.jl for timing details														
Score	None	Poor	Fair	Good	Excellent									
Explanation	Functionality does not exist	Functionality exists, but is probably incomplete	The basic features exist	The basic features exist and some extra tweakability exists. May include extra methods for efficiency.	The basic features exist and some extra tweakability exists. May include extra methods for efficiency.									

Figure 31: Tabella comparativa tra le varie implementazioni dei vari risolutori

Equazioni Differenziali Universali

Un *equazione differenziale universale* (UDE) e' una *equazione algebrica differenziale*[89] non triviale, ovvero un sistema di equazioni che contiene delle equazioni differenziali ed equazioni algebriche oppure e' un sistema equivalente, con la propriet'a che la sua soluzione puo' approssimare *qualsiasi* funzione continua su un qualunque intervallo $\in R$ a qualsiasi livello di precisione desiderata. [118]

Per essere precisi, una equazione differenziale (possibilmente in forma implicita) $P(y', y'', y''', \dots, y^{(n)}) = 0$ e' una UDE se, per ogni funzione a valori relative continua f e per ogni funzione continua positiva ϵ esiste una soluzione liscia[110] (una funzione e' considerabile liscia se e' differenziabile in ogni suo punto, perci' 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 puo' essere analogo all'idea di una *Macchina di Turing Universale* [119] 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.

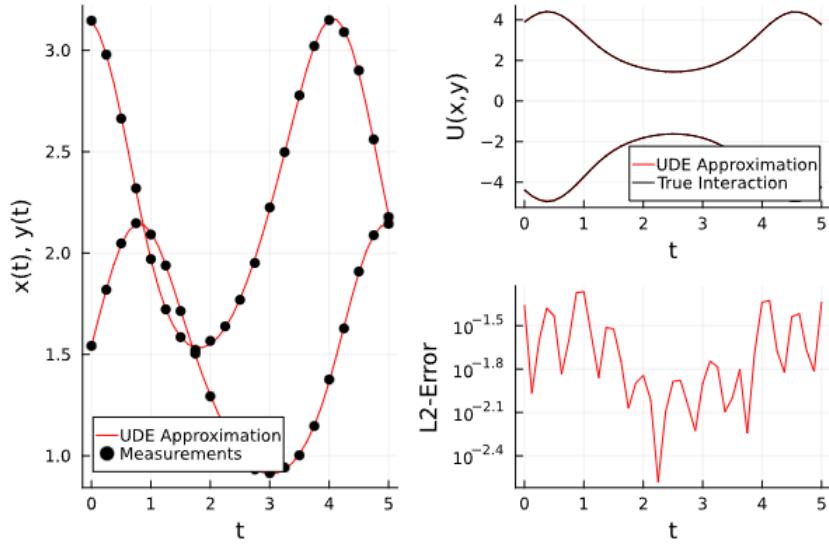


Figure 32: Comportamento UDE nell'approssimazione di fenomeni non lineari

Questo approccio viene spesso unito a tecniche *Data-Driven* [44] per l'identificazione sparsa di dinamiche non lineari. In particolare uno degli approcci utilizzati e' quello tramite l'algoritmo *SINDy* [113]. 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 e' stato largamente utilizzato nell'identificazione della *dinamica dei fluidi* cosi' come nelle *reti biologiche* e altri sistemi dinamici complessi.

2.5.3 SafeBlues

Durante la pandemia da Covid-19 il framework SciML.ai è stato utilizzato per sviluppare applicazioni le quali grazie all'utilizzo di tecniche di scientific machine learning riuscivano a prevedere in maniera molto accurata l'andamento dell'epidemia, seppur in presenza di una scarsa quantità di dati, e le stesse presentavano misure di contenimento e prevenzione che si sono dimostrate essere efficaci nel loro utilizzo [6] [20].

Un esempio può essere il modello denominato **SafeBlues** [6] [20] il quale simulando una rete bluetooth in cui gli individui potevano venire infettati da un virus e poi infettare a loro volta gli individui circostanti nella rete con una certa probabilità, aveva riprodotto fedelmente l'andamento della pandemia da Covid-19. In aggiunta questa soluzione, aveva mostrato come l'applicazione di policy per il contenimento del virus bluetooth erano perfettamente applicabili anche al caso reale della pandemia.

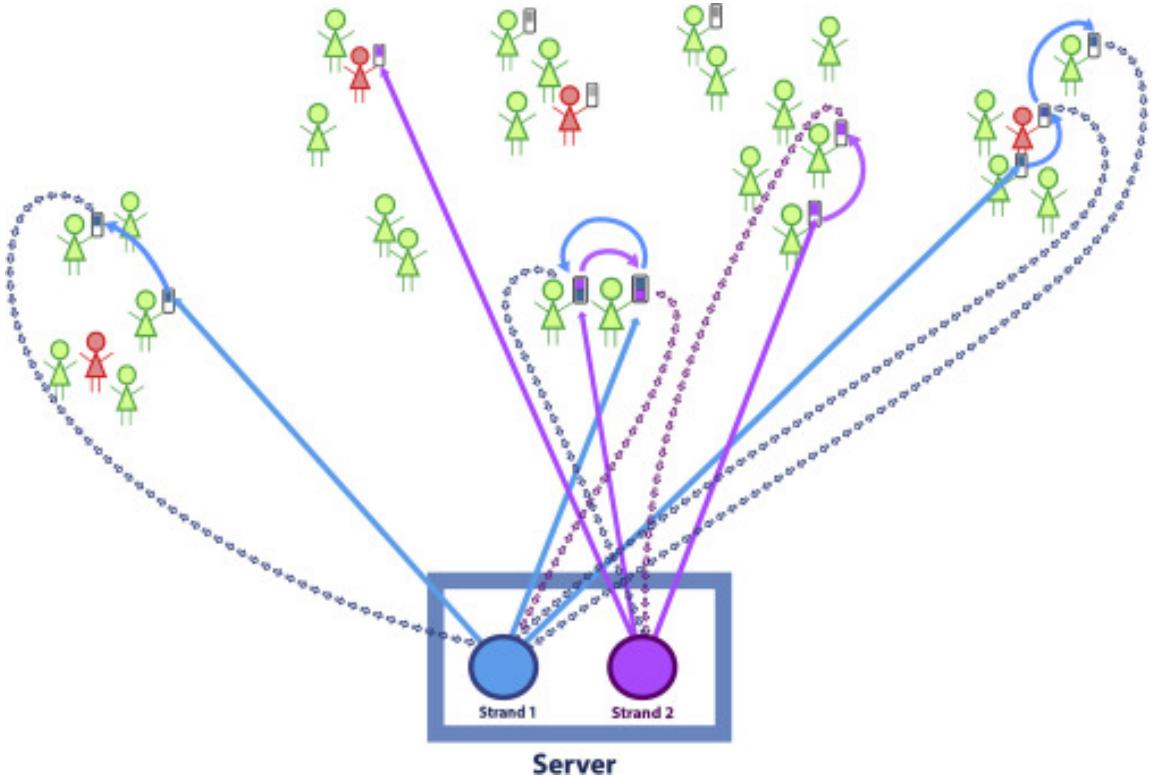


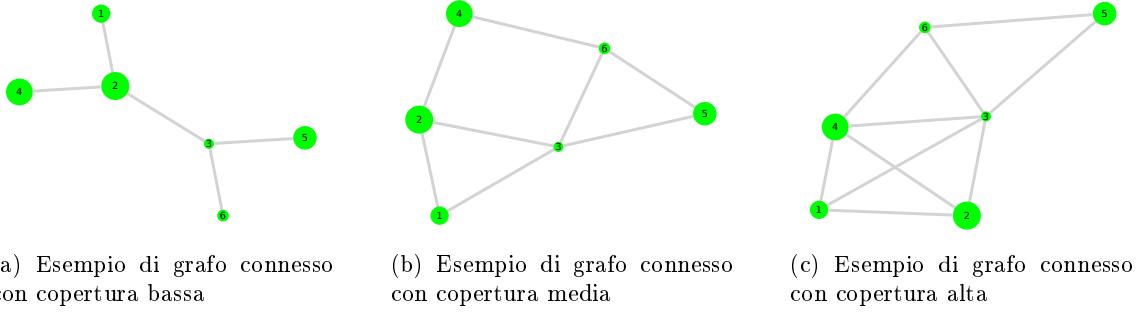
Figure 33: Esempio funzionamento SafeBlues

Questa applicazione e' stata sviluppata e rilasciata per dispositivi mobili con lo scopo di sperimentare se il sistema sviluppatto (*Safe Blues System*) potesse aiutare e migliorare i tradizionali metodi di previsione del decorso di una pandemia.

3 Metodi e Modelli

L'approccio utilizzato e' stato quello di modellare l'intero problema come se fosse una *rete sociale*. L'idea e' nata tramite dopo aver osservato e studiato attentamente il problema proposto.

Il problema si presenta come lo studio della diffusione di una pandemia a trasmissione virale in un ambiente reale, il quale e' composto principalmente di cosi' detti *Punti di Interesse* (PdI). Questi punti di interesse successivamente sono collegati tra loro tramite una rete piu' 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 piu' su un approccio di tipo *mesoscopico* tendente al *macroscopico*. Questo poiche' 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 difficolta' intrinseche ad una modellazione cosi' specifica, la quale non si adattava all'idea piu' ampia del problema. Difatti non si vuole modellare un sistema a livello microscopico per vedere le possibili interazioni tra agenti differenti, bensi' 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 una 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 e' importante ma le sue connessioni si, 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.

Un grafo sociale e' un particolare tipo di grafo che rappresenta le relazioni sociali tra entita'. Questo approccio viene largamente utilizzato per la rappresentazione delle reti sociali dove la parola grafo viene presa dalla teoria dei grafi. Generalmente un grafo sociale viene riferito come un grafo che mappa gli individui come nodi e le loro relazioni come archi. [111]

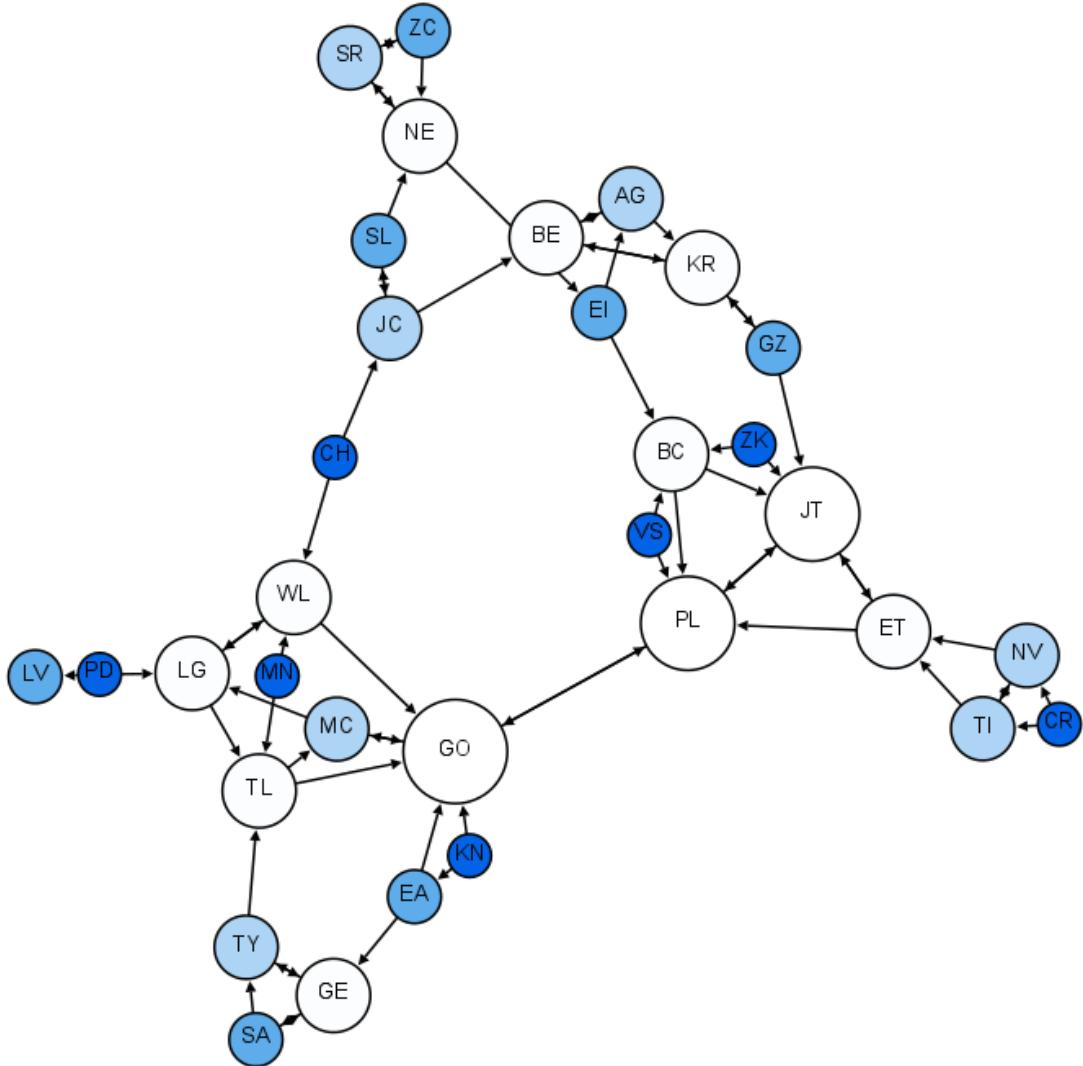


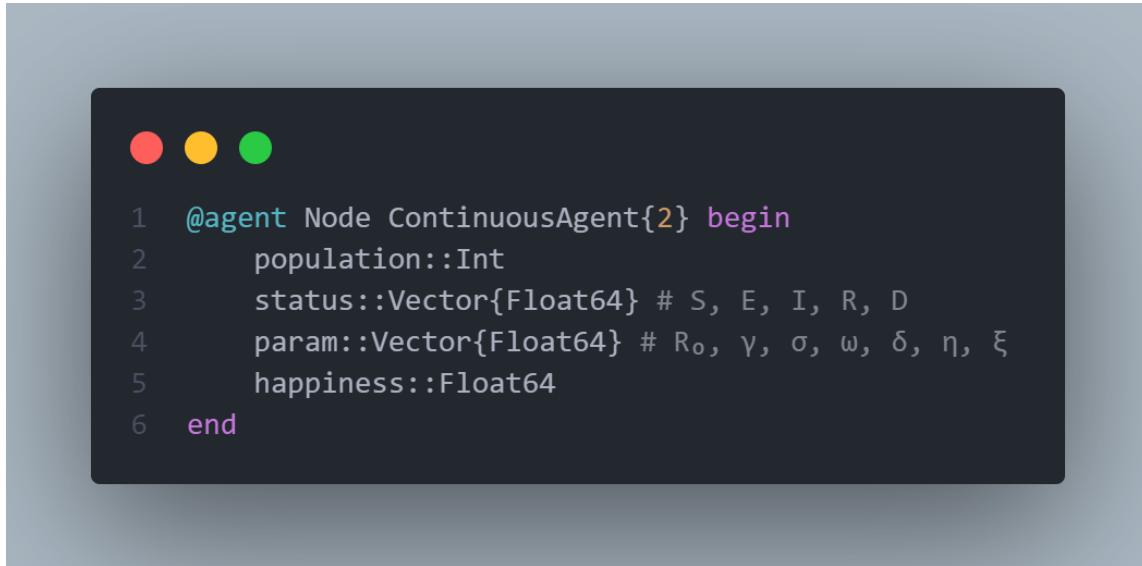
Figure 35: Esempio di grafo sociale

Il concetto è stato originariamente coniato dalla struttura dati **sociogramma** [112], 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 strettamente le relazioni interpersonali che si formano all'interno di un gruppo di individui.

La struttura dell'agente e' di tipo **ContinuousAgent** in quanto lo spazio del modello e' di tipo continuo. Questa scelta e' stata fatta in vista di possibili sviluppi futuri dell'applicazione. Successivamente si puo' osservare come il modello generato in figura 37 sia molto snello, avendo solamente

una manciata di parametri sufficienti al corretto funzionamento del modello stesso.

3.1.1 Agente



The screenshot shows a dark-themed code editor window. At the top left are three circular icons: red, yellow, and green. Below them is a block of C++ code:

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

Figure 36: Codice Agente

Come mostrato in figura 36 l'agente al suo interno e' molto minimale, descrivendo solamente gli attributi necessari e sufficienti per la corretta modellazione e interazione con se stesso. L'evoluzione di ogni agente e' completamente indipendente anche se questa puo' 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 e' nata principalmente per motivi di *type stability* in quanto le operazioni di risoluzione del sistema di ODE altrimenti potrebbero risentirne. Tuttavia e' stato fatto anche per un fattore di *facilita' di comprensione*.
- **param**: questo campo descrive tramite un vettore di tipo **Float64** il parametro 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 e' utilizzato come campo di appoggio per il bilanciamento delle contromisure applicate dal controllore. Non esprime effettivamente la misura della felicità'

che la popolazione di un nodo ha complessivamente, ma e' comunque uno stimatore similare.

3.1.2 Spazio e Modello



```
properties = @dict(
    numNodes, param, graph, migrationMatrix, step = 0, control, vaccine, integrator = nothing
)
model = ABM(
    Node,
    ContinuousSpace((100, 100); spacing=4.0, periodic=true),
    properties=properties,
    rng
)
```

Figure 37: Codice Modello

Come accennato precedentemente e mostrato in figura 37, lo spazio del modello e' di tipo **ContinuousSpace** seppure non venga effettivamente sfruttato a dovere. Tuttavia l'interesse del modello e' 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. Successivamente viene creata una matrice di flusso, denominata come **migrationMatrix** che rappresenta la percentuale di individui che dal nodo di interesse si spostano verso un nodo di destinazione. Questa matrice è salvata come **matrice sparsa** per risparmiare spazio dove le connessioni 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	.	.	.

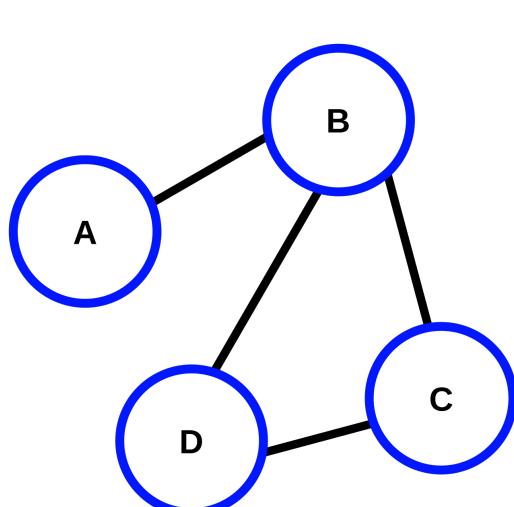
Figure 39: Matrice di migrazione

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 piu' popolosi, e minore verso i nodi meno popolosi. Questo flusso ha poi un limite massimo che puo' essere scelto e modificato dall'utente.

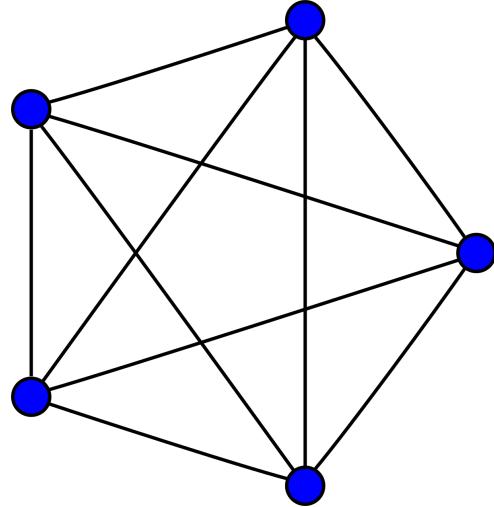
Se questa matrice fosse una matrice non sparsa, e perciò completa, si potrebbe osservare come la diagonale di questa matrice ha valori molto vicini ad 1, in relazione con il limite massimo di individui che possono migrare da ogni nodo, definito come *migration rate*.

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 piu' 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.

Applicando tale filosofia, è possibile creare un array di **ODEProblem**, oggetti che descrivono



(a) Esempio di grafo connesso



(b) Esempio di grafo completo

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 40: Definizione dei sistemi di ODE associati ad ogni agente

L'approccio scelto, ovvero di usare un integratore, puo' 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 discontinuita' nella risoluzione, ma queste vengono esplicitamente gestite se viene specificata una funzione di callback all'interno della funzione di solve.

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

3.1.3 Funzione di avanzamento agente

Un agente finche' all'interno di un modello segue ciclicamente il comportamento riportato in figura 41



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

Figure 41: 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 e' chiamato ad eseguire tre fondamentali compiti:

- **Muoversi:** un agente come entita' essendo un nodo di un grafo non puo' muoversi nel modo piu' letterale del termine, ma tramite la sua matrice di migrazione e' possibile calcolare quanti individui si spostano da il nodo sorgente, fino ad un nodo destinazione. Questo spostamento fa si che il nodo venga aggiornato con le relative percentuali di status e il totale di popolazione rimanente
- **Calcolare la felicità:** come detto in apertura della sezione, il valore di **happiness** e' un valore fantoccio che serve prevalentemente per bilanciare le contromisure del controllore. Questo valore tuttavia viene influenzato anche dalla situazione pandemica. Attualmente la funzione che si occupa di generare la felicità complessiva di un nodo si basa sull'utilizzo di una *tangente iperbolica* (\tanh) che prende come parametro il valore attuale di happiness del nodo e lo bilancia con il valore delle contromisure η associate e attualmente uso. Questo approccio e' sicuramente problematico e fallace per numerosi motivi ma attualmente adempie al suo obiettivo.
- **Chiamare il controllore:** se la proprieta' *control* del modello e' 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, andando ad utilizzare un insieme di tecniche di ottimizzazione non lineare ottenuta dalla suite **Ipopt** per stimare quale dovrebbe essere la policy migliore da applicare dato un valore massimo di costo.

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 e' un passo per ogni agente. Di default il framework Agents.jl effettua prima l'avanzamento degli agenti e successivamente quello del modello. Questo comportamento puo' 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 comunita' 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 42: Funzione di avanzamento del modello

Come e' possibile osservare dalla figura 42, 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 e' responsabile del calcolo e dell'applicazione delle misure di contromisure farmaceutiche all'interno della popolazione.

Questa sezione e' quella che piu' presta il fianco ad *assunzioni* riguardo il comportamento generale del sistema. Tutte le funzioni sopra descritte si basano su un insieme piu' o meno nutrito di assunzioni per funzionare correttamente. Questo perche' al momento in cui sto redigendo questo documento August 4, 2023, non sono ancora riuscito ad implementare metodi migliori.

La funzione che si occupa di generare la *variant of concern* (VOC) e' 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 43: 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 $8 \cdot 10^{-4}$ deriva dai seguenti articoli [51] [80] [2] i quali descrivono prevalentemente il tasso di mutazione casuale delle basi che compongono il DNA del virus SARS-COV2. Questo pero' non implica che tali mutazioni creino una VOC. Per semplicita' e' stato scelto di usare l'approccio per cui se una mutazione avviene, questa e' un a VOC. In linea di massima sembra che questo approccio semplicistico crei un numero di VOC generalmente sensato e in linea con quanto abbiamo potuto osservare durante la pandemia
- 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** [101]

La funzione che si occupa di stimare l'applicazione delle contromisure farmaceutiche, ovvero l'applicazione del vaccino, e' la seguente, e anch'essa presta il fianco a numerevoli assunzioni, le quali pero' 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)])
15             objective.param[7] = agent.param[7]
16         end
17     end
18 end

```

Figure 44: 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 puo' essere vaccinata ad ogni passo del modello. Questo approccio tende a simulare l'idea piu' o meno realistica di avere una quantita' fissata di dosi vaccinali ogni giorno che possono essere erogate alla popolazione.

Questo calcolo viene effettuato tenendo in considerazione l'approccio dell' *immunita' vaccinale di gregge* che puo' 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 immunita' 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 e' considerato come un individuo nello stato *Recovered* e segue le stesse regole di chiunque altro, indipendentemente dal modo in cui si e' entrati in questo stato.

Questa visione semplicistica comunque e' efficace nel modellare il decorso di una malattia, con e senza interventi.

3.1.5 Grafici

Grafico senza intervento

Il seguente grafico 45 mostra l'andamento delle curve del modello quando questo viene eseguito senza alcuna tipologia di intervento. Questo andamento tuttavia e' mostrato in maniera cumulativa rispetto all'andamento dei singoli agenti, il quale puo' avere comportamenti differenti in base a se appare o meno una variante, come specificato in figura 43, oppure dipendentemente dal tipo di contromisure applicate.

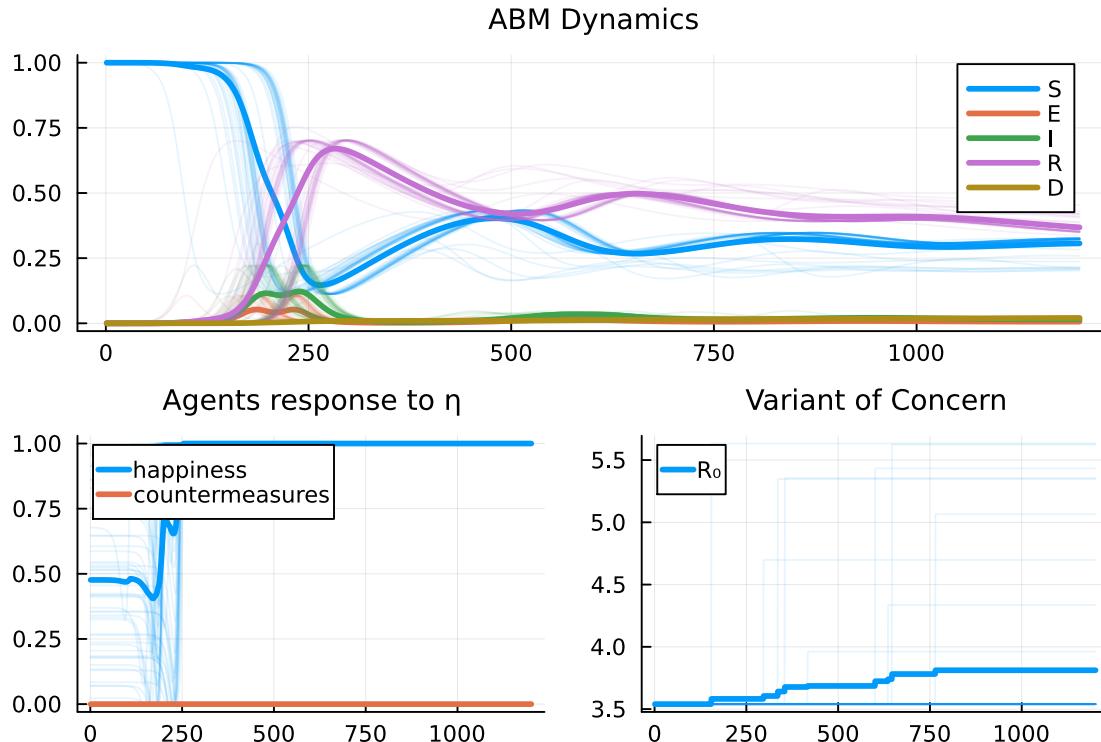


Figure 45: Grafico cumulativo del risultato del modello senza intervento del controllore

Complessivamente pero' l'andamento e' ovviamente similare ad un andamento standard di un modello SEIR, con qualche variazione dipendente dai fattori di stocasticita' del modello che pero' in questo caso non risultano troppo presenti, che vengono messi in evidenza con le curve del modello meno marcate, che rappresentano i percorsi "meno battuti" della simulazione.

Come e' 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 pero', per via di come e' stato definita la condizione di guariti, non sono immuni alle varianti del virus. Queste proprietà contribuiscono ad un andamento ciclico delle curve, in particolar modo di quelle E , I , R . La curva associata al compartimento S ha un suo andamento ciclico seppur molto sottile; questa sottigliezza e' dovuta all'assenza di misure di controllo all'interno del sistema.

Si puo' infine notare come la curva associata all'andamento degli individui nella classe *D* abbia una crescita lineare, seppur non troppo evidente.

A seguire si puo' 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 e' dovuto principalmente a come viene definita la funzione di controllo della felicità la quale essendo dominata principalmente da due fattori, η ovvero le contromisure applicate da un controllore e dalla proporzione tra il numero di individui nelle categorie **S**, **R** e gli individui nelle categorie **I**, **D** si arriva velocemente a comprenderne il funzionamento. Si osserva inoltre che la curva tende ad un *plateau* passato il periodo della "prima ondata".

Questo comportamento e' irrealistico e associato alla definizione che e' stata fatta della funzione **happiness!**. Tuttavia anche se il comportamento di questa curva non e' completamente realistico, si puo' comunque utilizzare per mantenere sotto controllo le contromisure η .

Infine si nota come, seppur la definizione della funzione associata alla creazione di una nuova *Variant of Concern (VOC)* 43 sia semplicistica e irrealistica, il grafico mostra come su un periodo di circa 3 anni, associabile alla durata del periodo covid, le voci sono al piu' una decina (ogni picco o depressione e' una voce).

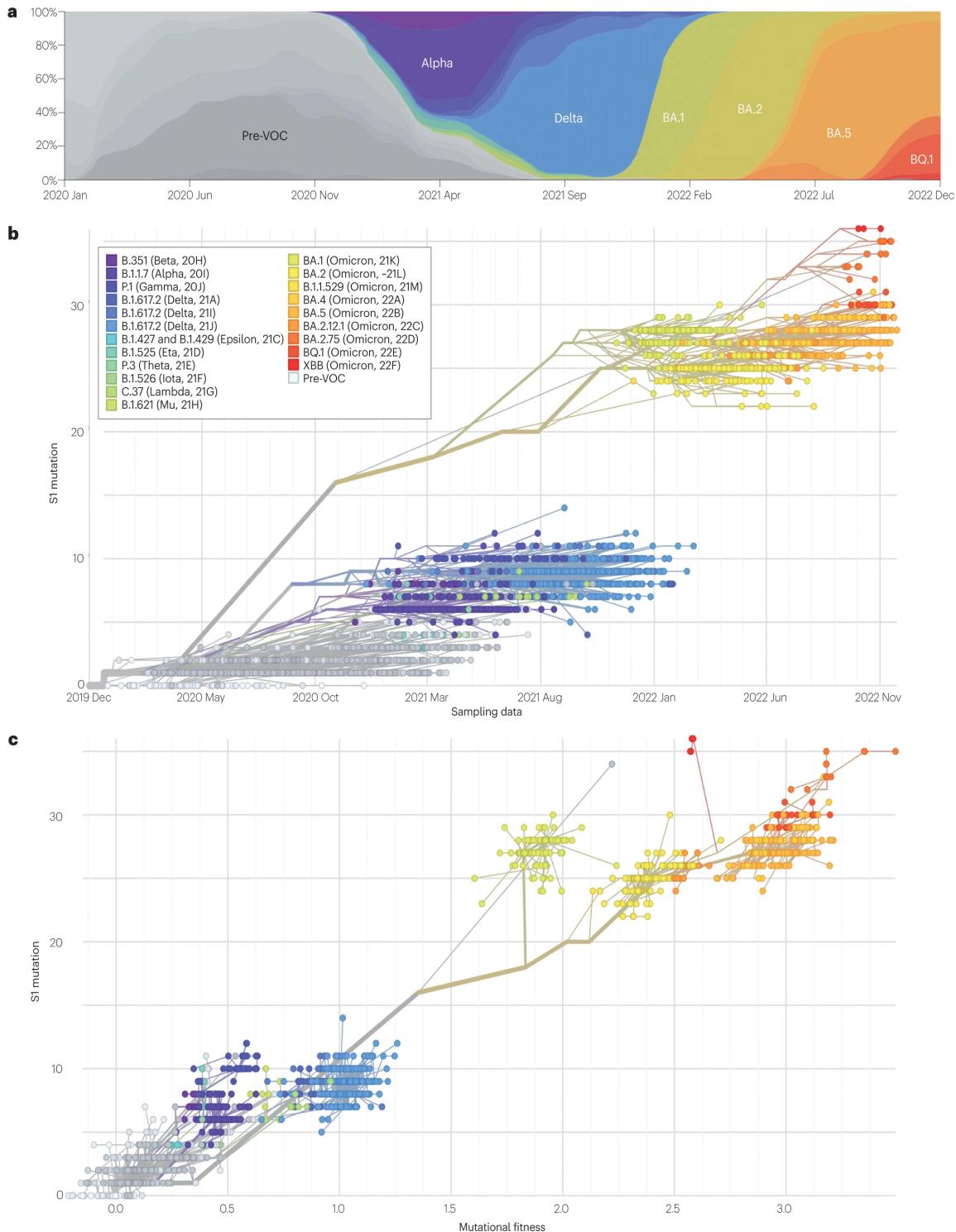


Figure 46: Grafico delle mutazioni casuali del virus SARS-CoV2 preso dall'articolo [51]

Si nota come le voci scoperte del covid siano una quantita' pressoche' similare seppur con una distribuzione all'interno della linea temporale differente. Tuttavia il comportamento semplicistico applicato nel modello sembra essere una buona approssimazione del comportamento reale del virus.

Grafico con 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. Questo andamento tuttavia e' mostrato in maniera cumulativa rispetto all'andamento dei singoli agenti, il quale puo' avere comportamenti differenti in base a se appare o meno una variante, come specificato in figura 43, oppure dipendentemente dal tipo di contromisure applicate.

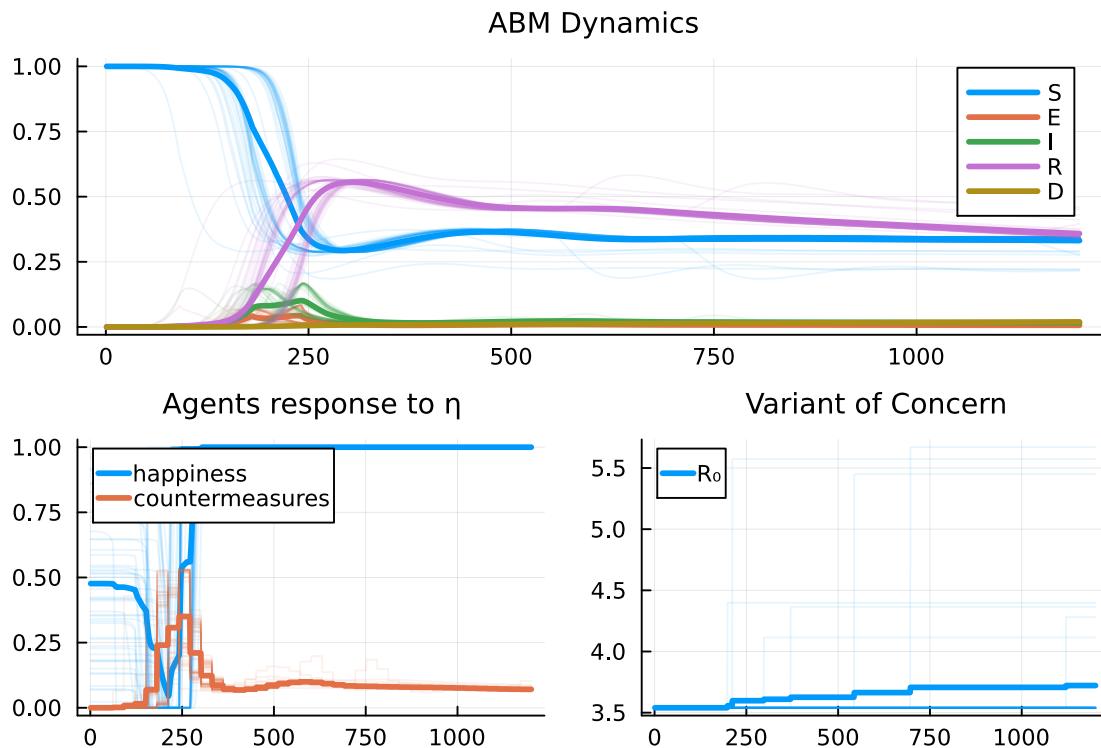


Figure 47: Grafico cumulativo del risultato del modello con intervento del controllore

Si puo' notare come le varie curve rappresentate siano molto piu' accentuate mostrando percorsi differenti ma comunque simili nell'andamento generale. La curva associata alla felicità e' altalenante e segue generalmente l'andamento delle contromisure applicate dal controllore alla popolazione. In questo caso vi sono curve molto differenti che mostrano comportamenti altrettanto differenti, ma in generale il comportamento e' relativamente contenuto e non eccessivo.

Altro dato interessante e' il numero di VOC che pare essere 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 39 andando a ridurne i valori presenti. Questo fa sì che oltre a dilazionare la diffusione della pandemia, andando ad applicare involontariamente delle contromisure simil lockdown e smart working (per prevenire il forte afflusso di individui), va anche a dilazionare il comportamento della funzione che si occupa di generare una VOC 43.

Questa infatti puo' attivarse sse nel nodo sono presenti individui infetti, in quanto non e' stata modellata la possibilita' che spontaneamente una nuova variante arrivi in un nuovo nodo senza un veicolo umano. Percio' applicare delle contromisure permette anche a contenere la diffusione di VOC nella popolazione osservata.

Grafico con 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 andamento tuttavia e' mostrato in maniera cumulativa rispetto all'andamento dei singoli agenti, il quale puo' avere comportamenti differenti in base a se appare o meno una variante, come specificato in figura 43, oppure dipendentemente dal tipo di contromisure applicate.

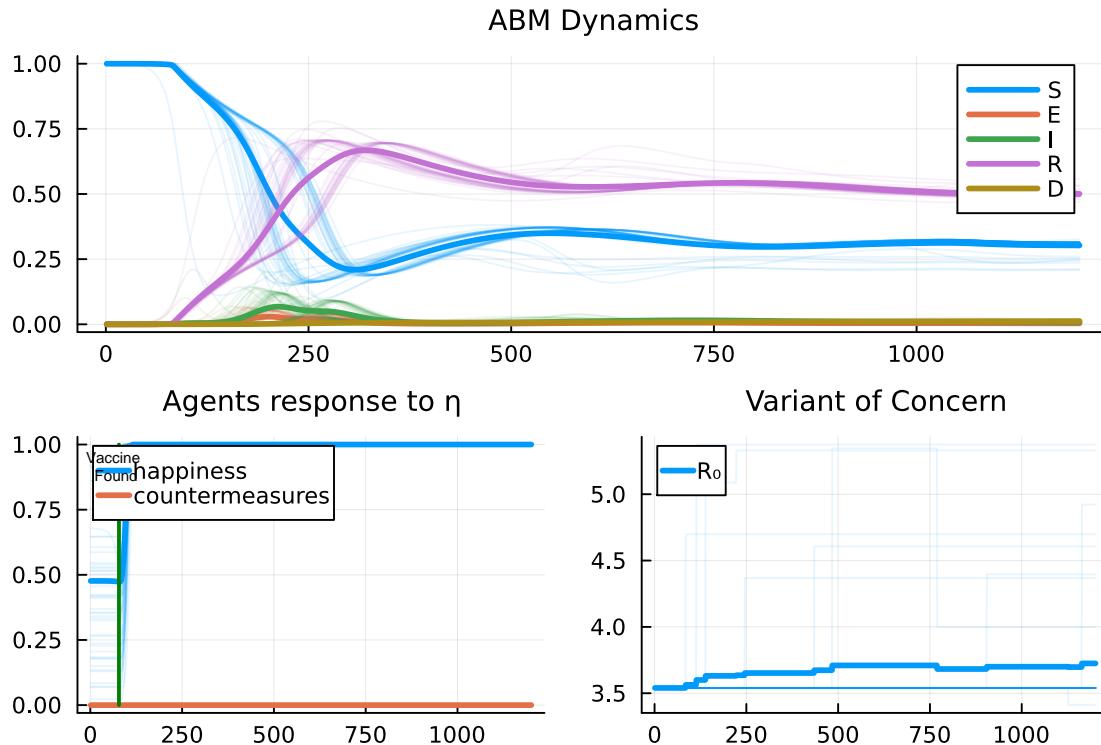


Figure 48: Grafico cumulativo del risultato del modello con intervento del controllore tramite vaccino

Come accennato con il grafico in figura 47 il grafico relativo al cambiamento del valore R_0 , quindi

indice di quante varianti compaiono in un determinato periodo di tempo, e' ulteriormente influenzato dalla presenza di un vaccino, in quanto rende la popolazione immune per un determinato periodo di tempo al virus, rendendo quest'ultimo meno influente e meno incline alla creazione di nuove varianti, le quali, una volta create paiono essere di media meno pericolose, avendo un valore R_0 medio meno alto rispetto a quelle dei grafici 47 e 45.

Questo grafico tuttavia puo' cambiare drasticamente in base a quando un vaccino viene effettivamente scoperto e rilasciato sul mercato come metodo per contrastare un'epidemia.

Grafico con intervento farmaceutico e non farmaceutico

Il seguente grafico 49 mostra l'andamento delle curve del modello quando questo viene eseguito tramite l'applicazione di una qualche tipologia di intervento farmaceutico. Questo andamento tuttavia e' mostrato in maniera cumulativa rispetto all'andamento dei singoli agenti, il quale puo' avere comportamenti differenti in base a se appare o meno una variante, come specificato in figura 43, oppure dipendentemente dal tipo di contromisure applicate.

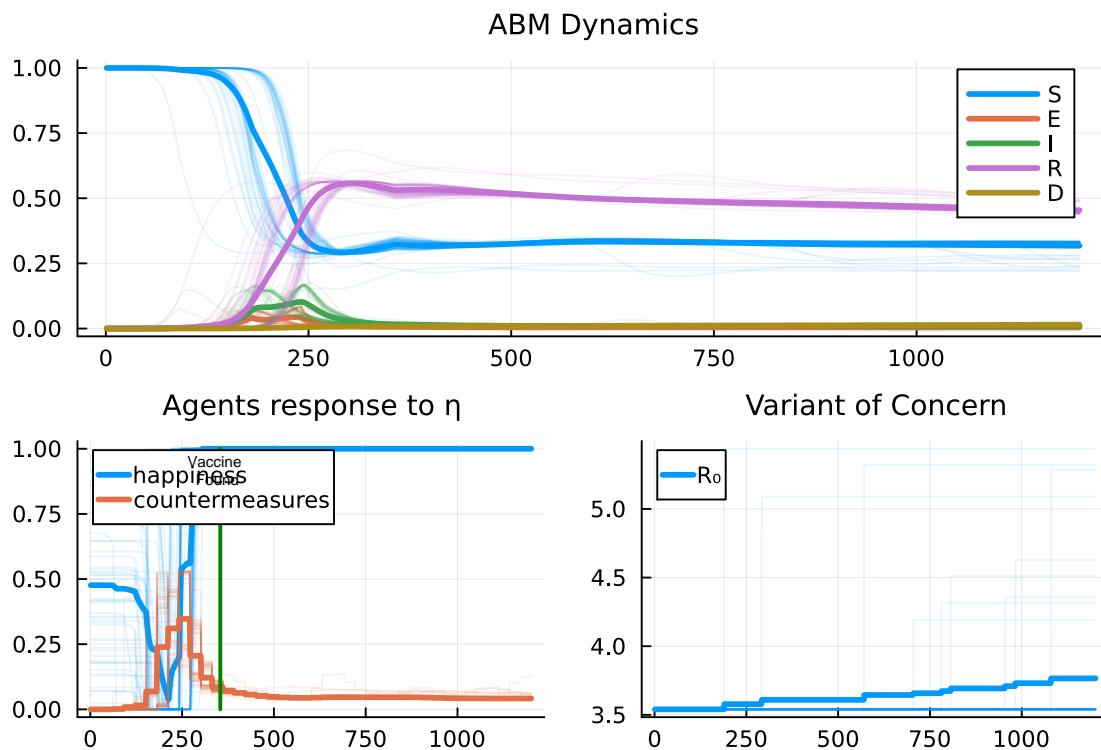


Figure 49: Grafico cumulativo del risultato del modello con intervento del controllore tramite vaccino e metodi di prevenzione non farmaceutici

In questo caso viene mostrata 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 si appiattire notevolmente la curva di infetti andando a creare velocemente un immu-

nita' 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 farmaceutiche 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.

3.2 Monitoraggio e Intervento

3.2.1 SciML.ai

L'utilizzo della suite **SciML.ai** e' stato fatto per l'integrazione di un modello di Machine Learning in un sistema di monitoraggio e possibilmente di intervento. Applicare un algoritmo di ML per predire l'andamento di un sistema non lineare dinamico senza avere una grande base di dati puo' risultare problematico e i risultati possono risultare poco affidabili. Lo sviluppo di tecniche che congiungono la modellazione puramente matematica alla flessibilita' delle reti neurali ha permesso di sviluppare approcci *data-driven domain-specific* [65] [46] [19] [16] che permettono di ottenere risultati affidabili pur avendo una base di dati scarsa.

Addestramento

Viene definito il sistema di ODE che fara' da base per l'addestramento della **Neural ODE** [16] e successivamente viene definita la rete neurale tramite la suite **Lux.jl** [58]. Questo permette di definire facilmente e velocemente una rete neurale che verra' usata in coppia con il sistema di ODE per ottenere il meglio da entrambi, andando a definire quello che viene chiamato **Scientific Machine Learning**.



```

1 function sir_ode!(du, u, p, t, p_true)
2     S, E, I, R, D = u
3     R₀, γ, σ, ω, δ = p_true
4     λ = U(u, p, st)[1]
5     μ = δ / 1111
6     du[1] = μ * sum(u) - λ[1] * S - μ * S # dS
7     du[2] = λ[1] * S - σ * E - μ * E # dE
8     du[3] = σ * E - γ * I - δ * I * μ * I # dI
9     du[4] = (1 - δ) * γ * I - ω * R - μ * R # dR
10    du[5] = δ * γ * I # dD
11 end

```

```

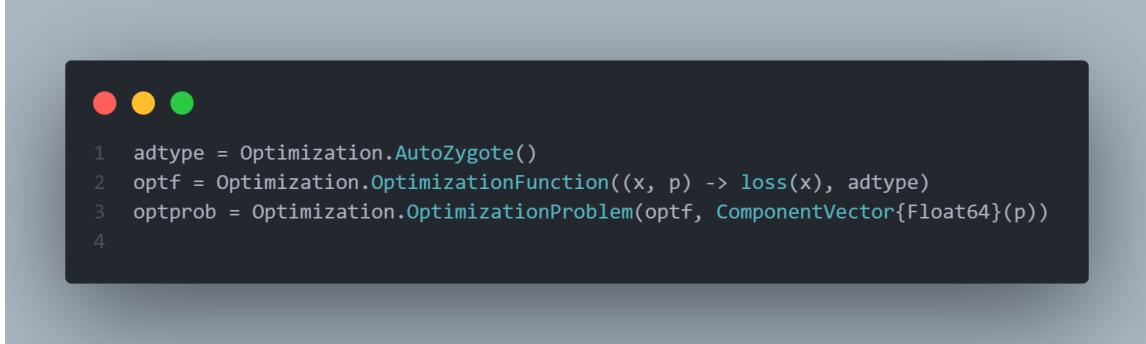
1 # Multilayer FeedForward
2 U = Lux.Chain(Lux.Dense(5, 64, relu), Lux.Dense(64, 1))
3 # Get the initial parameters and state variables of the model
4 p, st = Lux.setup(rng, U)

```

(a) Definizione del sistema di ODE per l'addestramento della rete neurale
(b) Definizione della rete neurale

Dopo aver definito queste due funzioni, vengono definite le regole di predizione per l'addestramento e le regole della funzione di loss. La funzione di loss viene definita come la loss L2 ovvero un **Mean Squared Error (MSE)** o **Errore Quadratico Medio** [100].

Utilizzando poi le regole di ottimizzazione offerte dal framework **Optimization.jl** [23] e' possibile definire una struttura di controllo tramite la creazione di un **OptimizationProblem** e una **OptimizationFunction** e la scelta di un algoritmo di ottimizzazione, generalmente lasciato automatico, e' possibile addestrare la propria *Neural ODE*.



```

1 adtype = Optimization.AutoZygote()
2 optf = Optimization.OptimizationFunction((x, p) -> loss(x), adtype)
3 optprob = Optimization.OptimizationProblem(optf, ComponentVector{Float64}(p))
4

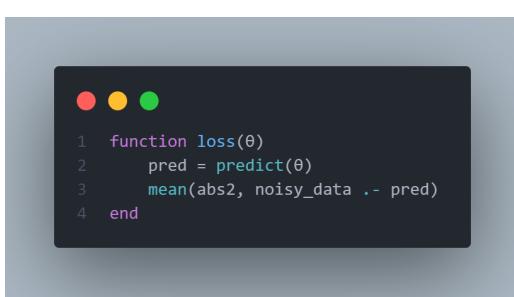
```

Figure 52: Definizione delle funzioni di ottimizzazione

Per ottenere il massimo dall'addestramento vengono effettuati due cicli di addestramento, uno usando un determinato ottimizzatore, **ADAM** e successivamente utilizzando quel risultato, fare un altro ciclo di addestramento con un ottimizzatore differente, in questo caso **BFGS** [13] [28] [34] [72].

Questo procedimento viene effettuato principalmente perche' si cerca di sfruttare al meglio le propria' dei vari ottimizzatori. In primo luogo si utilizza ADAM per la maggior parte delle iterazioni, andando a sfruttare e massimizzare la sua capacita' di trovare una buona area generale dello spazio dei parametri. Successivamente per una manciata di iterazioni viene utilizzato BFGS che velocemente riesce a cadere in un buon minimo locale.

Se si utilizzasse solamente ADAM si andrebbe probabilmente incontro ad una richiesta in termini di iterazioni molto alta, rispetto al risultato complessivo, e viceversa se si utilizzasse solo BFGS si andrebbe incontro ad un brutto minimo locale.



```

1 function loss(θ)
2     pred = predict(θ)
3     mean(abs2, noisy_data .- pred)
4 end

```

(a) Definizione della funzione di Loss



```

1 function predict(θ, X = noisy_data[:, 1], T = tspan)
2     _prob = remake(prob_nn, u₀ = X, tspan = (T[1], T[end]), p = θ)
3     Array(
4         solve(
5             _prob,
6             OrdinaryDiffEq.Vern7(; thread = OrdinaryDiffEq.ThreadType()),
7             saveat = T,
8             abstol = 1e-6,
9             reltol = 1e-6,
10            verbose = false,
11        ),
12    )
13 end

```

(b) Definizione del loop di addestramento

```

1 maxiters = 1000
2 res1 = Optimization.solve(optprob, ADAM(), callback = callback, maxiters = maxiters)
3 println("Training loss after ${length(losses)} iterations: ${losses[end]}")
4 optprob2 = Optimization.OptimizationProblem(optf, res1.u)
5 res2 = Optimization.solve(
6     optprob2,
7     Optim.LBFGS(),
8     callback = callback,
9     maxiters = trunc(Int, maxiters / 5),
10 )
11 println("Final training loss after ${length(losses)} iterations: ${losses[end]}")

```

Figure 53: Definizione delle funzioni di addestramento tramite due differenti ottimizzatori

Predizione

Viene utilizzato un approccio simile a quello denominato SINDy [113] per effettuare predizioni sul lungo termine fondendo un approccio **DataDriven** ad uno **Domain Specific**.

L'approccio **SINDy** ovvero Sparse Identification of Nonlinear Dynamics, e' un approccio algoritmico di tipo data-driven per ottenere sistemi dinamici partendo da un insieme di dati. In generale l'approccio e' quello di avere una serie di *instantanee (snapshot)* del sistema e le loro corrispondenti derivate. A questo punto e' possibile effettuare un insieme di approcci di regressione come ad esempio **LASSO** [99] per confrontare una molteplicita' di equazioni non lineari candidate con le derivate del sistema per cercare di trovare le equazioni che governano l'intero sistema.

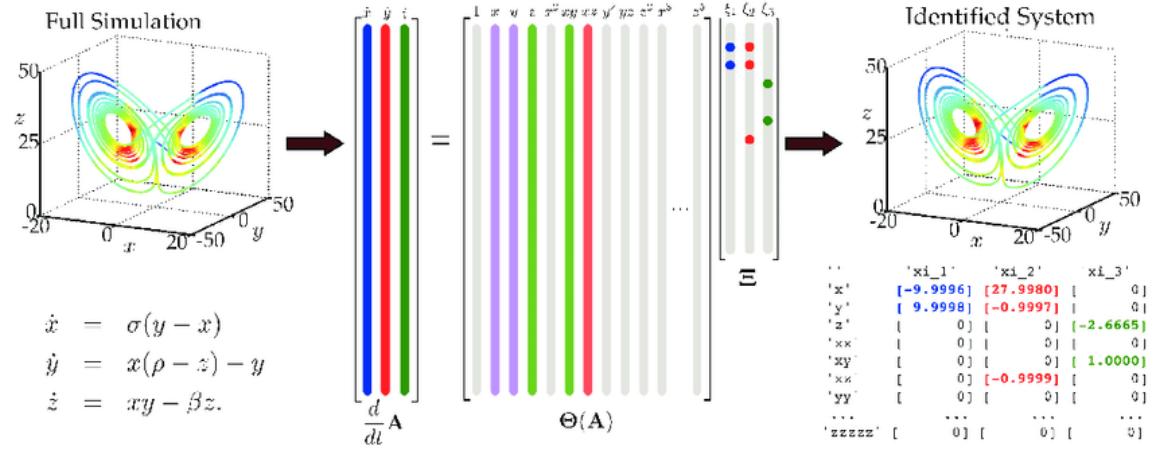


Figure 54: Esempio schematico del funzionamento di SINDy [14]

Questa procedura si basa sull'assunzione che la maggior parte di sistemi fisici hanno una manciata di equazioni dominanti che dettano l'evoluzione del sistema.

Seguendo quanto appena descritto, l'approccio si basa principalmente sull'utilizzo del modello appena creato e addestrato di Neural ODE, utilizzando due parti fondamentali di quest'ultimo

- X che rappresenta il predittore del sistema.
- Y che rappresenta la derivate del sistema, o la predizione stessa.

Questi due elementi vengono successivamente inseriti all'interno di un algoritmo data-driven per estrapolare le equazioni che governano il sistema, e successivamente vengono utilizzate queste equazioni in coppia con il vero sistema, da cui sono state ricavati i valori, che in questo caso e' conosciuto in quanto si sa che i valori sono generati da un sistema di tipo SEIR, per vedere se il sistema riesce a modellare i dati che non conosce, generalmente associati alla **FOI** o Force of Infection.

```

1 #variables u[1:3]
2 b = polynomial_basis(u, 4)
3 basis = Basis(b, u);
4
5 nn_problem = DirectDataDrivenProblem(X, Y)
6 λ = exp10(-3.0:0.01:3)
7 opt = ADMM(λ)
8
9 options = DataDrivenCommonOptions(
10     maxiters = 10_000,
11     normalize = DataNormalization(ZscoreTransform),
12     selector = bic,
13     digits = 1,
14     data_processing = DataProcessing(
15         split = 0.9,
16         batchsize = 30,
17         shuffle = true,
18         rng = rng,
19     ),
20 )
21
22 nn_res = solve(nn_problem, basis, opt, options = options)
23 nn_eqs = get_basis(nn_res)

```

(a) Codice relativo alla definizione del problema di tipo data-driven per SINDy

```

1 # Define the recovered, hybrid model
2 function recovered_dynamics!(du, u, p, t, p_true)
3     ū = nn_eqs(u, p) # Recovered equations
4     S, E, I, R, δ = ū
5     dS = -p * S * E
6     dE = p * S * E - δ * E
7     dI = δ * E - γ * I
8     dR = γ * I - μ * R
9     du[1] = p * ū[1] * ū[2] - ū[1] * S - μ * S # dS
10    du[2] = ū[1] * S - α * E - μ * E # dE
11    du[3] = α * E - γ * I - δ * I + μ * I # dI
12    du[4] = (1 - δ) * γ * I - u * R - μ * R # dR
13    du[5] = δ * γ * I # dR
14 end
15
16 recovered_dynamics!(du, u, p, t) = recovered_dynamics!(du, u, p, t, p_true)
17
18 estimation_prob = ODEProblem(recovered_dynamics!, u, tspan, get_parameter_values(nn_eqs))
19 estimate =
20     solve(estimation_prob, Tsit5(; thread = OrdinaryDiffEq.Trun)), saveat = solution.t
19

```

(b) Codice relativo alla definizione del problema di tipo data-driven per SINDy

L'approccio e' simile a quello utilizzato per la Neural ODE precedentemente, ovvero si utilizza la libreria **Optim.jl** [23] per ottenere il risultato finale che si andra' ad utilizzare.

3.2.2 Grafici

4 Risultati Ottenuti

Analisi riassuntiva dei risultati ottenuti con qualche pensiero personale

5 Sviluppi Futuri

5.1 Altre epidemie

5.1.1 Ebola

5.1.2 Aviaria

5.1.3 Influenza

5.2 Miglioramento integrazione SciML.ai

5.3 Accuratezza nella descrizione del modello

5.3.1 Happiness

5.3.2 Interventi localizzati

5.3.3 Migliore flessibilità generale

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 Angevaare, 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 Hoffman, 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] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [39] 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.
- [40] Michael Innes, Elliot Saba, Keno Fischer, Dhairyा Gandhi, Marco Concetto Rudilosso, Neethu Mariya Joy, Tejan Karmali, Avik Pal, and Viral Shah. Fashionable modelling with flux. *CoRR*, abs/1811.01457, 2018.
- [41] Mike Innes. Flux: Elegant machine learning with julia. *Journal of Open Source Software*, 2018.
- [42] 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.
- [43] Masoud Jalayer, Carlotta Orsenigo, and Carlo Vercellis. Cov-abm: A stochastic discrete-event agent-based framework to simulate spatiotemporal dynamics of covid-19, 2020.
- [44] JuliusMartensen, Christopher Rackauckas, et al. Datadrivendiffeq.jl, July 2021.

- [45] 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 Famularo, and Daniel J. Klein. Covasim: An agent-based model of covid-19 dynamics and interventions. *PLOS Computational Biology*, 17(7):1–32, 07 2021.
- [46] 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.
- [47] Garyfallos Konstantinoudis, Dominic Schuhmacher, Håvard Rue, and Ben D Spycher. Discrete versus continuous domain models for disease mapping. *Spatial and Spatio-temporal Epidemiology*, 32:100319, 2020.
- [48] James Ladyman, James Lambert, and Karoline Wiesner. What is a complex system? *European Journal for Philosophy of Science*, 3(1):33–67, Jan 2013.
- [49] 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.
- [50] 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.
- [51] 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.
- [52] 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.
- [53] 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.
- [54] 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.
- [55] 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.
- [56] 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.

- [57] World Health Organization. Who coronavirus (covid-19) dashboard.
- [58] Avik Pal. Lux: Explicit Parameterization of Deep Neural Networks in Julia, 2023. If you use this software, please cite it as below.
- [59] M Parascandola and D L Weed. Causation in epidemiology. *J Epidemiol Community Health*, 55(12):905–912, December 2001.
- [60] 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.
- [61] C. Rackauckas. A comparison between differential equation solver suites in matlab, r, julia, python, c, mathematica, maple, and fortran. *The Winnower*.
- [62] Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit. Diffeqflux.jl - a julia library for neural differential equations, 2019.
- [63] 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.
- [64] 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.
- [65] 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.
- [66] 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.
- [67] Christopher Rackauckas and Qing Nie. Confederated modular differential equation apis for accelerated algorithm development and benchmarking. *Advances in Engineering Software*, 132:1–6, 2019.
- [68] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.
- [69] 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.
- [70] 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.
- [71] Ilya Orson Sandoval, Panagiotis Petsagourakis, and Ehecatl Antonio del Rio-Chanona. Neural odes as feedback policies for nonlinear optimal control, 2022.

- [72] D. F. Shanno. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.
- [73] Constantinos I. Siettos and Lucia Russo. Mathematical modeling of infectious disease dynamics. *Virulence*, 4(4):295–306, 2013. PMID: 23552814.
- [74] 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.
- [75] Byron Tasseff, Carleton Coffrin, Andreas Wächter, and Carl Laird. Exploring benefits of linear solver parallelism on modern nonlinear optimization applications, 09 2019.
- [76] 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.
- [77] 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.
- [78] Evren Mert Turan and Johannes Jaschke. Multiple shooting for training neural differential equations on time series. *IEEE Control Systems Letters*, 6:1897–1902, 2022.
- [79] 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.
- [80] 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.
- [81] 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.
- [82] Wikipedia. Agent-based model — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Agent-based%20model&oldid=1148935023>, 2023. [Online; accessed 30-April-2023].
- [83] Wikipedia. Backpropagation — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=1166762709>, 2023. [Online; accessed 03-August-2023].
- [84] Wikipedia. Brownian motion — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Brownian%20motion&oldid=1155409035>, 2023. [Online; accessed 21-May-2023].
- [85] Wikipedia. Causality — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Causality&oldid=1150225747>, 2023. [Online; accessed 27-April-2023].
- [86] Wikipedia. Compartmental models in epidemiology — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Compartmental%20models%20in%20epidemiology&oldid=1150020798>, 2023. [Online; accessed 27-April-2023].

- [87] Wikipedia. Correlation — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Correlation&oldid=1149809031>, 2023. [Online; accessed 21-May-2023].
- [88] Wikipedia. Cycle of poverty — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Cycle%20of%20poverty&oldid=1151766525>, 2023. [Online; accessed 27-April-2023].
- [89] Wikipedia. Differential-algebraic system of equations — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Differential-algebraic%20system%20of%20equations&oldid=1159353421>, 2023. [Online; accessed 16-June-2023].
- [90] Wikipedia. Distribuzione di Poisson — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Distribuzione%20di%20Poisson&oldid=130383267>, 2023. [Online; accessed 23-July-2023].
- [91] Wikipedia. Epidemiologia — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Epidemiologia&oldid=124857046>, 2023. [Online; accessed 27-April-2023].
- [92] Wikipedia. Equazione differenziale ordinaria — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Equazione%20differenziale%20ordinaria&oldid=130359533>, 2023. [Online; accessed 21-May-2023].
- [93] Wikipedia. Euler method — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Euler%20method&oldid=1166507568>, 2023. [Online; accessed 03-August-2023].
- [94] Wikipedia. Gradiente (funzione) — Wikipedia, the free encyclopedia. [http://it.wikipedia.org/w/index.php?title=Gradiente%20\(funzione\)&oldid=133500447](http://it.wikipedia.org/w/index.php?title=Gradiente%20(funzione)&oldid=133500447), 2023. [Online; accessed 03-August-2023].
- [95] Wikipedia. Immunità di gregge — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Immunit%C3%A0%20di%20gregge&oldid=133955463>, 2023. [Online; accessed 17-June-2023].
- [96] Wikipedia. Incubation period — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Incubation%20period&oldid=1152591975>, 2023. [Online; accessed 21-May-2023].
- [97] Wikipedia. Influenza spagnola — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Influenza%20spagnola&oldid=132751127>, 2023. [Online; accessed 27-April-2023].
- [98] Wikipedia. Julia (programming language) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Julia%20\(programming%20language\)&oldid=1151847776](http://en.wikipedia.org/w/index.php?title=Julia%20(programming%20language)&oldid=1151847776), 2023. [Online; accessed 27-April-2023].
- [99] Wikipedia. Lasso (statistics) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Lasso%20\(statistics\)&oldid=1154880240](http://en.wikipedia.org/w/index.php?title=Lasso%20(statistics)&oldid=1154880240), 2023. [Online; accessed 23-July-2023].

- [100] Wikipedia. Mean squared error — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Mean%20squared%20error&oldid=1163054387>, 2023. [Online; accessed 22-July-2023].
- [101] Wikipedia. Numero di riproduzione di base — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Numero%20di%20riproduzione%20di%20base&oldid=133906348>, 2023. [Online; accessed 17-June-2023].
- [102] Wikipedia. Pandemia — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Pandemia&oldid=132968684>, 2023. [Online; accessed 27-April-2023].
- [103] Wikipedia. Pandemia di COVID-19 — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Pandemia%20di%20COVID-19&oldid=133076664>, 2023. [Online; accessed 27-April-2023].
- [104] Wikipedia. Peste nera — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Peste%20nera&oldid=132822978>, 2023. [Online; accessed 27-April-2023].
- [105] Wikipedia. Polynomial regression — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Polynomial%20regression&oldid=1113806162>, 2023. [Online; accessed 23-July-2023].
- [106] Wikipedia. Positive feedback — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Positive%20feedback&oldid=1147282594>, 2023. [Online; accessed 27-April-2023].
- [107] Wikipedia. Rectifier (neural networks) — Wikipedia, the free encyclopedia. [http://en.wikipedia.org/w/index.php?title=Rectifier%20\(neural%20networks\)&oldid=1163249828](http://en.wikipedia.org/w/index.php?title=Rectifier%20(neural%20networks)&oldid=1163249828), 2023. [Online; accessed 03-August-2023].
- [108] Wikipedia. Simpson's paradox — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Simpson%20paradox&oldid=1154565479>, 2023. [Online; accessed 21-May-2023].
- [109] Wikipedia. Simulation software — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Simulation%20software&oldid=1149041989>, 2023. [Online; accessed 27-April-2023].
- [110] Wikipedia. Smoothness — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Smoothness&oldid=1142265373>, 2023. [Online; accessed 16-June-2023].
- [111] Wikipedia. Social graph — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Social%20graph&oldid=1147955674>, 2023. [Online; accessed 16-July-2023].
- [112] Wikipedia. Sociogram — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Sociogram&oldid=1108592124>, 2023. [Online; accessed 16-July-2023].
- [113] Wikipedia. Sparse identification of non-linear dynamics — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Sparse%20identification%20of%20non-linear%20dynamics&oldid=1157228086>, 2023. [Online; accessed 16-June-2023].

- [114] Wikipedia. Spurious relationship — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Spurious%20relationship&oldid=1139468569>, 2023. [Online; accessed 30-April-2023].
- [115] Wikipedia. Storia della pandemia di AIDS — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Storia%20della%20pandemia%20di%20AIDS&oldid=132423486>, 2023. [Online; accessed 27-April-2023].
- [116] Wikipedia. Tifo esantematico — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Tifo%20esantematico&oldid=132730114>, 2023. [Online; accessed 27-April-2023].
- [117] Wikipedia. Universal approximation theorem — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Universal%20approximation%20theorem&oldid=1166603320>, 2023. [Online; accessed 03-August-2023].
- [118] Wikipedia. Universal differential equation — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Universal%20differential%20equation&oldid=1138859269>, 2023. [Online; accessed 16-June-2023].
- [119] Wikipedia. Universal Turing machine — Wikipedia, the free encyclopedia. <http://en.wikipedia.org/w/index.php?title=Universal%20Turing%20machine&oldid=1149333304>, 2023. [Online; accessed 16-June-2023].
- [120] Wikipedia. Urto elastico — Wikipedia, the free encyclopedia. <http://it.wikipedia.org/w/index.php?title=Urto%20elastico&oldid=133666418>, 2023. [Online; accessed 22-July-2023].
- [121] 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.
- [122] 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.
- [123] 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 entita' effettivamente attive all'interno dello spazio. Ci si puo' 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.

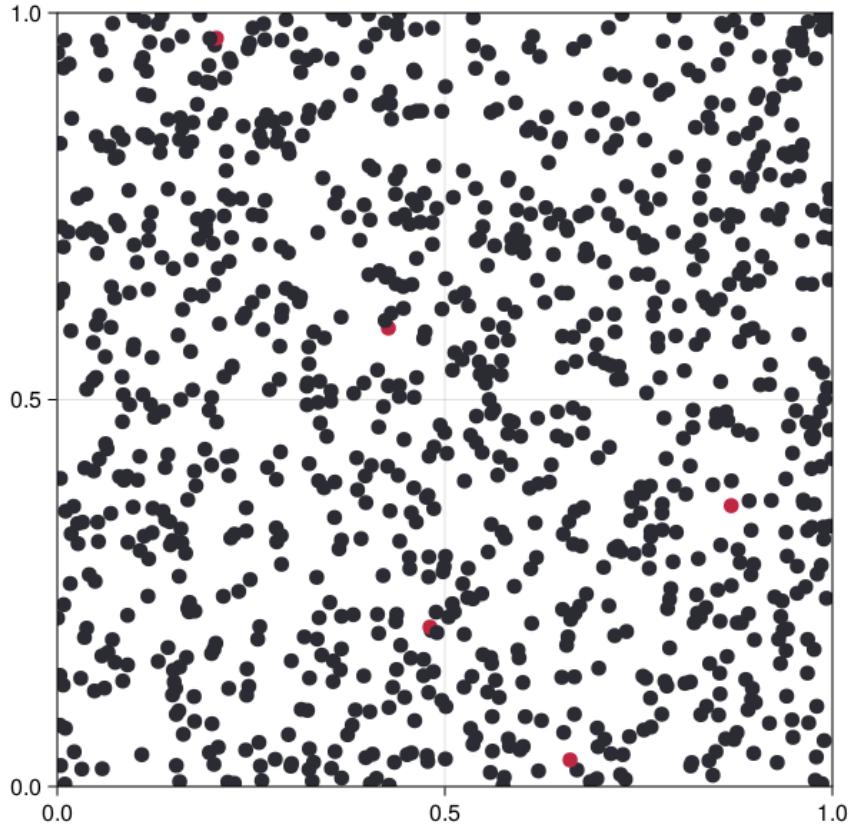


Figure 56: Esempio del modello modellato su spazio continuo

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 [120] 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.

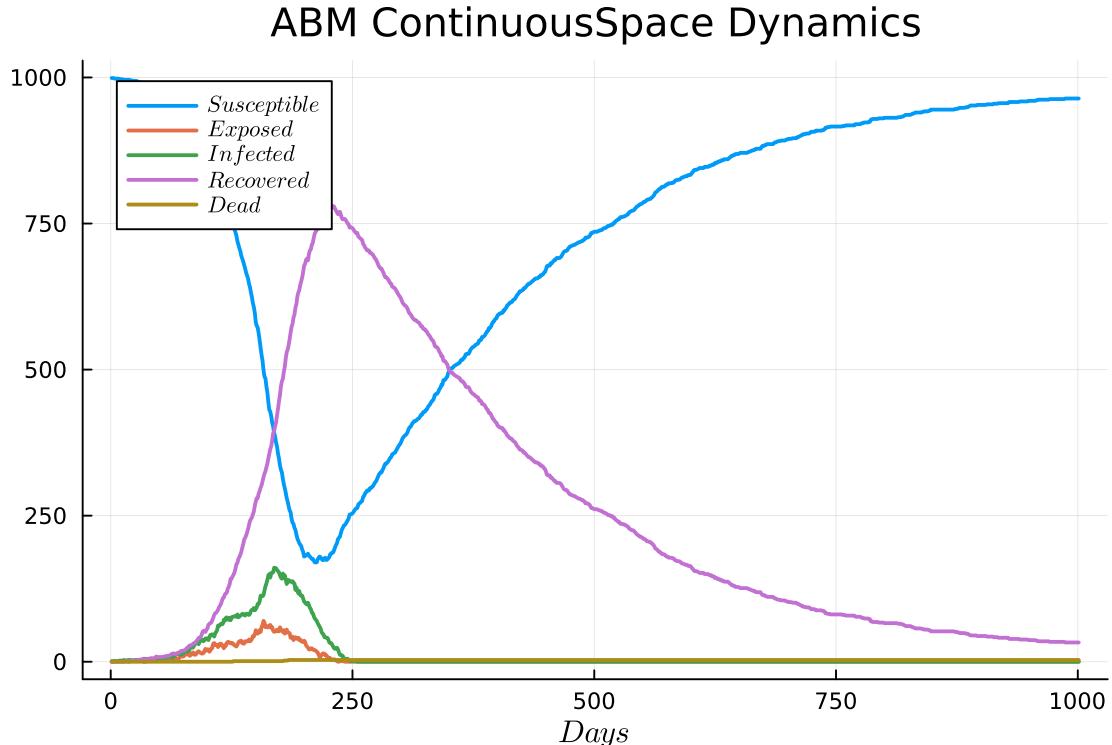


Figure 57: Esempio del comportamento delle curve nel modello continuo

Sono state poi implementate svariate proprieta' come la proprieta' 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 probabilita' di infettare ed essere infettati perdendo la capacita' di muoversi.

Questa tipologia di approccio, quella Agent oriented, richiedeva una quantita' di risorse computazionali e di tempo estremamente elevato in relazione al numero di agenti presenti nel modello. Questo approccio inoltre e' stato valutato come troppo granulare e inadatto allo scopo del progetto. Si e' deciso quindi di adottare un approccio meno granulare e piu' flessibile andando ad un livello di astrazione piu' alto.

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

L'idea e' nata per cercare di avere un controllo piu' 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.

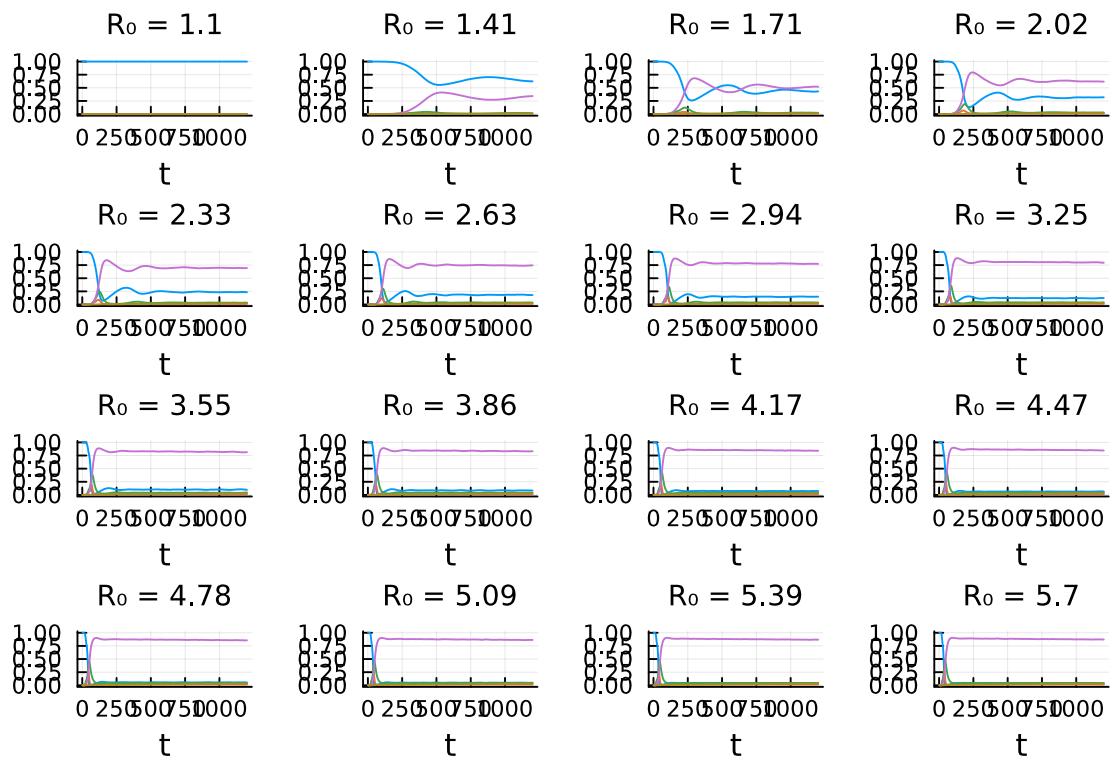


Figure 58: Comportamento modello ABM su spazio a grafo al variare del parametro R_0

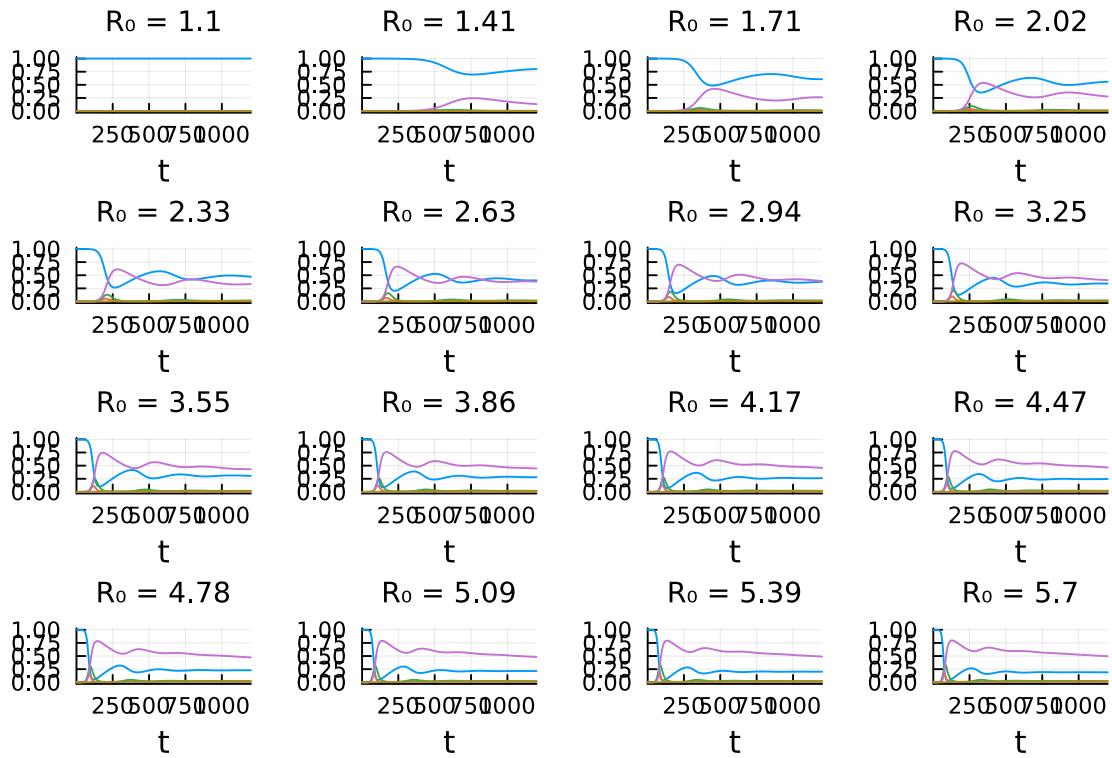


Figure 59: Comportamento modello SEIR al variare del parametro R_0

Si puo' 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 pero' puo' essere descritto a grandi linee dalla seguente formula

```

● ● ●
1 adapt_R0!(x) = return 1.1730158534328545 + 0.21570538523224972 * x
2

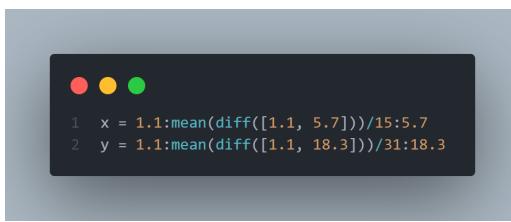
```

Figure 60: 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 e' 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** [90] con parametro $\lambda = R_0$.

In particolare la formula e' 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 e' stato calcolato **MSE** (Medium Square Error) per ogni coppia possibile di risultati per cercare quelli tra loro piu' simili, andando ad ottenere quindi un insieme di coppie. Queste poi sono state inserite in un metodo che calcolava una **polynomial fit** [105] 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 60 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 cosi' inaspettata rispetto a come dovrebbe non e' 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 e' stato anche il fatto che l'approccio con ABM era estremamente esoso in termini di risorse computazionali e tempistiche. Questo problema e' insito in ogni tipo di simulazione, tuttavia affiancato allo stravagante problema comparso e descritto sopra, e' stato decisivo per il cambio drastico di approccio.

A.3 Controllore Ipopt

Ipopt (Interior Point OPTimizer) [79] e' un pacchetto software per l'ottimizzazione non lineare su larga scala. Questo pacchetto software e' 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 $f(x) : R^n \rightarrow R$ e' 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.

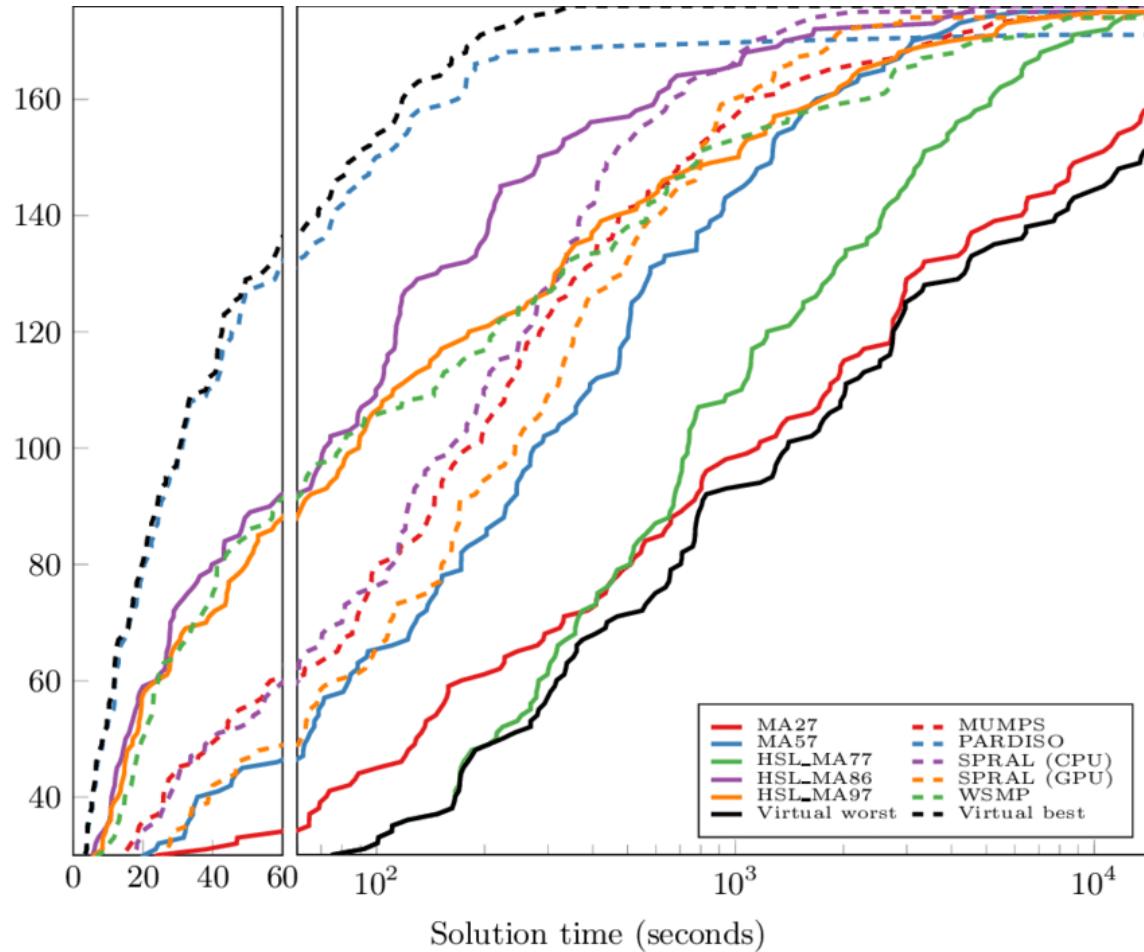


Figure 62: Comparison of Ipopt performance over various linear solvers using the two-dimensional partial differential equation test problem set. [75]

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

L'utilizzo della suite **Ipopt** e' 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)
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 63: 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.



```

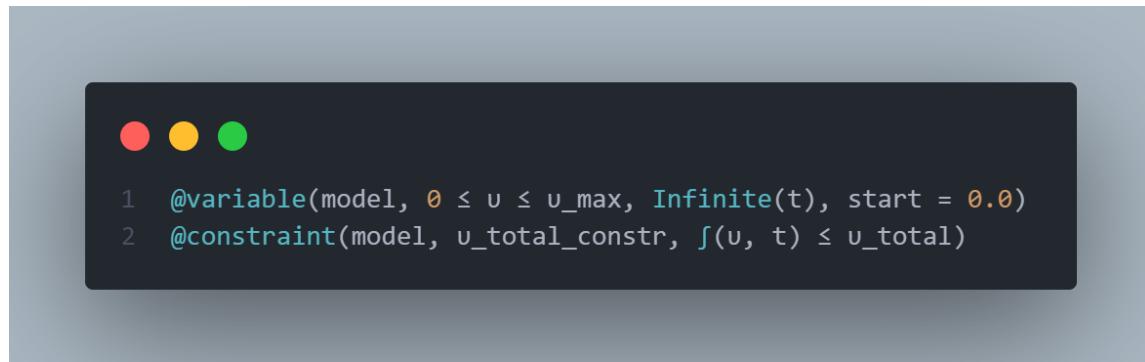
1  @constraint(model, S(0) == initial_condition[1])
2  @constraint(model, E(0) == initial_condition[2])
3  @constraint(model, I(0) == initial_condition[3])
4  @constraint(model, R(0) == initial_condition[4])
5  @constraint(model, D(0) == initial_condition[5])
6  @constraint(model, C(0) == c0)
7
8  @constraint(model, 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 64: Definizione regole del modello del controller

Essendo che le regole mostrate in figura 64 sono relative agli stati SEIR e l'idea alla base del controllore e' quella di ridurre quanto piu' possibile il numero di infetti cumulati che ci sono all'interno del sistema, e' stato aggiunto uno stato che descrive appunto questo stato aggiuntivo.

Successivamente vi sono delle regole su quanto il modello puo' 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.



```

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

```

Figure 65: Definizione regole del modello del controller per le contromisure

Infine viene ottimizzato il modello e ritornato il valore medio delle contromisure applicate quando applicate.