CSYS5051 Report

Shing Hin (John) Yeung

November 30, 2020

Chapter 1

Introduction

The outbreak of COVID-19 is an ongoing pandemic that affects worldwide. This is caused by a corona virus SARS-COV-2 which attacks the respiratory system. Human-to-human transmission is possible. While human who infected may become asymptomatic, where they may not show symptoms at the initial phase of infection. The epidemic starts from late 2019 from Wuhan, People's of Republic China. It eventually spread to most countries of the world, with 62 million cases has been confirmed and more than 1.44 million deaths worldwide (World Health Organisation 2020a) as at the end of November 2020. The World Health Organisation has declared this as a public health emergency (World Health Organisation 2020b).

While the impact of the pandemic is devastating. There is an international effort to discover medication and, more often is a vaccine that prevents the spread of the epidemic. Vaccination is one of the breakthroughs in public health. The modern vaccine was invented by Edward Jenner in 1798 to protect the population from smallpox (Centers for Disease Control and Prevention 2016). It eradicates the smallpox pandemic in 1970 (World Health Organization 2010). At the time of writing, there are yet any vaccine and anti-viral medications are available on market. There are few vaccines, including Pfizer (Pfizer 2020), Moderna (Moderna 2020) and the University of Oxford (in collaboration with AstraZeneca) (Oxford University 2020) has positive outcomes from their Phase

III trial.

The aim of this study is to identify the behavioural factors that contributes to this pandemic, where this report proposes several factors using a wide range of quantitative tools. Furthermore, this study lays the policy implications for optimal vaccine adoption. For example, the main horsepower in modelling behaviour are using game-theoretical setting upon human decisions. Game theory is a general framework where multiple decision makers as players, while they make a decision they will get the corresponding payoff due to how others decision. The are famous examples such as the prisoners' dilemma. Where two players are required to confess or defect the other in a police interrogation. When the players decided to defect each other, which they may serve the least prison sentence if the other decided to confess or defect. In fact, when both players defect each other they will serve more sentence because of their mutual decision. This is also called a Nash equilibrium when one player decides the most beneficial decision for them, and no matter what other move will not change this person's decision.

Epidemiology and game theory can be connected when we observe one person commit something that may or may not cause infection. For example, we may speculate perform social distancing while outside can reduce the probability to be infected. While social distancing is not legally mandatory, it depends on where the decision maker think this is necessary. So they will decide based on the rewards provided by each decision. This study considers several behavioural aspect and therefore predict the epidemic curve. Apart from social distancing, travelling overseas, personal hygiene are prominent examples and there are some discussions about their effect. Vaccination is an important public health policy to eradicate epidemics, and to vaccinate or not becomes a decision to individuals and the government authorities.

Social distancing is one of the main ideas talked about in public health policies during the COVID-19 pandemic. It has been shown that COVID-19 can be transmitted by respiratory droplets from coughs and sneezes. Humans positioned within a range of about 1.8 metres may receive the infected droplets

(Centers for Disease Control and Prevention 2020). Daily routine involves close face-to-face contact and it is a risk of transmitting COVID-19. For example, meeting work clients and teaching students but also living with family or housemates are common practices. However, this does not fixated the people we meet everyday. Longitudinal social network considers the contact updates by network evolution. In simple words, there is a well-defined network update rules driving this change. Throughout the pandemic, there is a much emphasis to reduce close contacts by imposing social distancing. Quantitatively there are less nodes linked to each other.

Within the personal layer, individual opinions drive the consensus outcome. Opinion dynamics is one of the successful areas in sociophysics. It analyses how human preferences influence the group behaviour. With simple models and assumptions, it can capture the essence of opinion settlement in real world settings (e.g. Borghesi, Raynal, and Bouchaud 2012 and Fernández-Gracia et al. 2014). This study follows from the local majority model by Galam (2012). The population form local groups of equal size, and they positioned based on the local majority consensus. Cheon and Morimoto (2016) had noted where the "minority dominance over majority" in Galam and Jacobs (2007, cited in Cheon and Morimoto 2016) and the indecisional hung vote when there was nearly 50:50 bipartisan opinion in Galam (2004, cited in Cheon and Morimoto 2016) and Borghesi and Galam (2006, cited in Cheon and Morimoto 2016). These have lead to the idea of heterogeneous personalities composition such as inflexibles and balancers. It is common to encounter people whom are inflexible upon their position. These people holds their position for long time and do not change in debates or social communications. Balancers act as either devil's advocate and contrarians. They often purposed to stimulate broader discussion, occasionally targeting the inflexibles. This paper includes new findings from Cheon and Morimoto (2016) with their new type of balancers, where they tries to oppose the majority decision. They have claimed with the inclusion of balancers, this will prevent where minority opinion overkill the majority and the opinion space become stagnated at 50:50.

Both the longitudinal social networks and opinion dynamics may require agent based modelling. Agent based modelling considers the heterogeneity of agents, for example, people in this case. When they interact, thus they will change states or perform tasks. This type of modelling produce unique results when they consider the feedback loop from heterogeneous agents and non-linear results cannot be produced by mean field modelling. This study combines all tools within an agent based model. Each parts of the society can be viewed as separate factors and the computer program may model them individually.

1.1 SIRV Model

The model is based on a compartment model which describes the transition of epidemic states. We follow from Abou-Ismail (2020) to define a stochastic compartment model to define the movement of COVID-19 patients. This is shown in fig. 1.1 below.

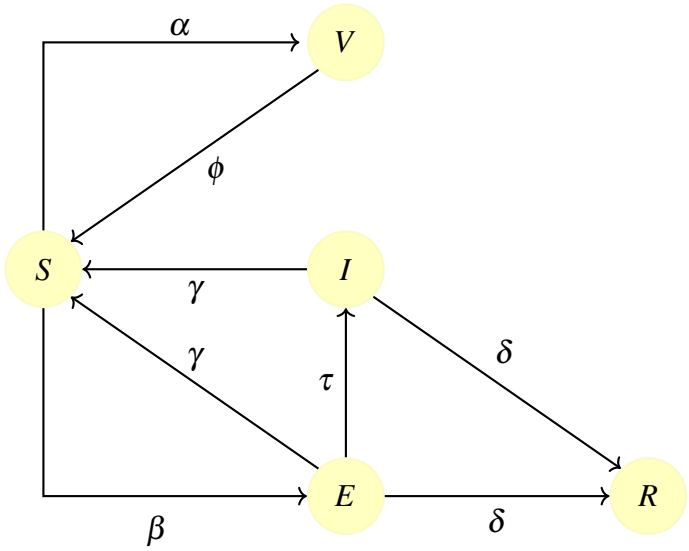


Figure 1.1: SIRV Model

Initially most of the population of N people are intact to the infection, and the simulation starts when the infection is introduced to the population. The simulation uses a set number to start with. For example, in New South Wales state of Australia, it started with 4 cases initially (NSW Health 2020). This study

uses this as base number for all simulations. Each susceptible person is subject to a probabilistic transition between $S \rightarrow E$. Where β is the individual transmission rate among each contact. It can be customised depends on the situation. The compartment model is featured with two infection compartments: E (Exposed, asymptomatic) or I (infected and symptomatic). Patients within the E compartment may infect others by contact, and they will transfer from $E \rightarrow I$ either after fourteen days or if they have conducted a COVID-19 testing. Either infection compartments are subjected to a transition to S state under the probability γ. The model considers patients may probabilistically removed to the R component, and there are no further transitions from there. This is defined by a removal probability δ. To encounter their lifestyle, for example, people may need to spare time to visit their GP or their "laziness" to vaccinate. The probability α, namely the adoption probability, describes this lifestyle factor hence delaying to take vaccine. The higher the value, the likely the population is taking the vaccine immediately. The vaccine itself may wear off due to its nature, and this is determined by a factor of ϕ.

1.1.1 Demographics

The key of COVID-19 transmission is the close contacts among humans. Social life is an integral part of human life, and it differs between people. This model incorporates the social contact through a negative binomial distribution $\mathrm{NB}(c, \beta)$. It is used to model independent and identical Bernoulli trials (i.e. "yes" or "no") until a specified number of failures c occurs. Unlike other common probability distributions, it allows the mean and variance to be different. Which in epidemiology it is used to model when transmission rate differs between different settings. In this model, the transmission rate differs between people living in the city and in rural environment.

For example, in urban environment some people may contact large number of people. Front line service workers are requires to take care of many patients, including whom infected with COVID-19. At the same time, not everyone in

urban environment meets such large amount of people and a large variance of social contacts among individuals should be considered. In rural environment, while there are low migration and outgoing travelling rates. It is expected where c is low.

Gender are also an important factor in modelling COVID-19. It is shown that women are twice likely to be infected than men, while men are more likely to die from COVID-19 (Wenham, Smith, and Morgan 2020). The model incorporates optional infection and removal rates for men and women.

Age is an important factor in this pandemic. For example, elderly population is more likely to die from COVID-19. When they are infected, elderlies are subjected to a higher rate of removal rate δ_e. The implementation of vaccine incorporates prioritised allocation to these vulnerable people. For example, when there are less supplies than demand. These vulnerable people are chosen into the pool of prioritised allocation, with a high value of α. The rest of the population then subject to a lower adoption rate α.

Apart from age relates to death rate, people whom have chronic diseases are subjected to higher rate of removal rate δ_c. The same implementation may apply to these people.

1.2 Game Theoretical Model

1.2.1 Overseas Travel

People travelling overseas may contact with people whom infected. Thus when they come back, they will pass on the infection to local population. We model a person whom is planning to travel overseas will gain the following:

$$E_1 = -\mathbf{A}\beta_{\text{dest}} r_{I,i} \tag{1.1}$$

Where $\mathbf{A} \in [0, 1]$ is the constant that a person is aware the pandemic occurred in the destination. Their reward is to be infected $r_{I,i}$. The utility gained from

not travelling is

$$E_0 = -r_0 \tag{1.2}$$

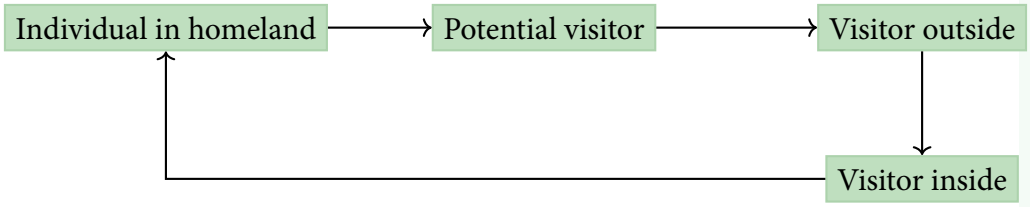


Figure 1.2: Visitor's states of travelling overseas. (Zhao, Bauch, and He 2018)

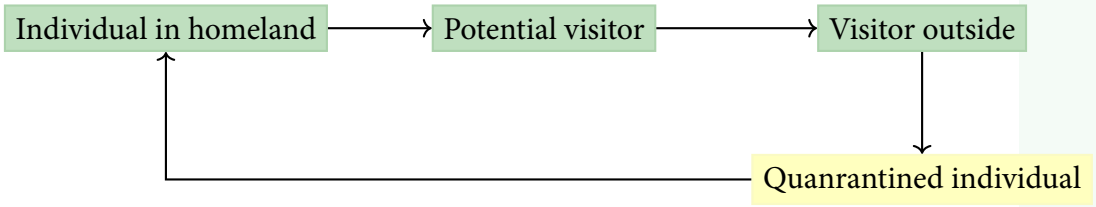


Figure 1.3: Visitor's states of travelling overseas under lockdown.

During government lockdown, a limited number of citizens may still able to travel overseas. The utility function has an extra term of further cost. When they return, they are required to self-isolated. The amended utility function becomes

$$E_1 = -\mathbf{A}\beta_{\text{dest}} r_{I,i} - r_\ell \varepsilon \tag{1.3}$$

However, this depends on each person's situation and so there is a externality factor $\varepsilon \sim N(0, 1)$ that simulates the noise parameter.

1.2.2 Vaccine Adoption under Bounded Rationality

People decides realistically based on their belief and imperfect information. Thus payoffs rarely it arrives to Nash equilibrium but in the middle between rational and random choices. This is called bounded rationality. From the perspective of modelling, we use a logit function for a person make decisions. The normal form is (Kasthurirathna, Harre, and Piraveenan 2016)

$$P_i(X) = \frac{e^{\lambda \langle U(s_{i,X}, P(X)) \rangle}}{\sum_i e^{\lambda \langle U(s_{i,X}, P(X)) \rangle}} \tag{1.4}$$

Where λ is the rationality parameter.

In this model, each person should decide if they should vaccinate or not. Their decision will lead to a different payoff r_s. This is

$$r_s = \begin{cases} r_V & \text{non-vacinnated} \\ r_{\neg V} & \text{vacinnated} \end{cases} \tag{1.5}$$

This means the probability of a person to implement a strategy is

$$P_i(X) = \frac{e^{\alpha\lambda r_s}}{\sum_s e^{\alpha\lambda_i r_s}} = \frac{e^{\alpha\lambda r_s}}{e^{\alpha\lambda_i r_V} + e^{\alpha\lambda_i r_{\neg V}}} \tag{1.6}$$

1.2.3 Vaccine Availability and Type

This model incorporates different situations when the vaccine is available. For example, the vaccine may provide permanent immunity (i.e. one off), seasonal vaccine or as a chemoprophylaxis. These are differed by the wear-off probability ϕ:

- **One off** $\phi = 0$
- **Chemoprophylaxis** $\phi = \text{const.}$
- **Seasonal** $\phi \propto \beta$

The vaccine may also caused the person to immune (i.e. $\beta = 0$ if contact with this person) or reduce symptoms (i.e. higher γ or lower δ).

1.2.4 Intimacy Game in Vaccine Adoption

We separate the population into two groups: People who vaccinates and people who do not. Individuals may switch between strategies. Suppose X_i represents the strategy of the i-th person to vaccinate or not, then we have

$$X_i = \begin{cases} 0 & \text{non-vacinnate} \\ 1 & \text{vacinnates} \end{cases} \tag{1.7}$$

Where the payoff from choosing to vaccinate or not would be

$$\begin{aligned} P(X = 0) &= r_{I,i}\theta_i \\ P(X = 1) &= r_{V,i} \end{aligned} \tag{1.8}$$

Where θ_i is the perceived probability of infection (Bhattacharyya, Vutha, and Bauch 2019). This is defined as

$$\theta_i \equiv \rho \frac{\#\text{ local infection}}{\#\text{ all infection}} + (1 - \rho)\frac{\#\text{ global infection}}{\#\text{ all infection}} \tag{1.9}$$

Where ρ is the relative importance of local and global information. We also assume that the person switch strategies based on Fermi-Dirac function, which is

$$P(X_i) = \frac{1}{1 + \mathrm{e}^{-\lambda \Delta P_i}} \tag{1.10}$$

Where $\Delta P \equiv P(1) - P(0)$ is the utility of the node i given by $\Delta P = r_{V,i} - r_{I,i}\theta_i$.

1.3 Contact Networks

This study considers the contact network as a Babarasi-Albert network. This type of network is a scale free network constructed by preferential attachment: A new node will connect to a fixed number of nodes, under the condition where the new links are form probabilistically from the degree (to all the degrees in the network) of the target node. It has an implication to reality where people prefers to contact with popular people. Thus the popular people are always more popular.

The longitudinal social network refers to the update at the end of each time step. To maintain the degree distribution, Xulvi-Brunet–Sokolov algorithm is used for rewiring the nodes. The algorithm finds 4 nodes that located either the sides of 2 links. Then they may either

- The nodes with highest degrees rewire together, and vice versa. Therefore

the network will become more assortative.

- The nodes with highest degrees rewire with node with low degrees. Therefore the network will become more disassortative.

There is another layer on top of the rewiring method, which is the probability to rewire. It is said that high probability will move the assortative or dissortative rewiring to more assortative or dissortative network respectively.

1.4 Opinion Dynamics and Vaccination

An individual would express their opinions to vaccinate (or not), and this is influenced by their local information group. The simulation considers this as an information network. This simulation considers 2 components:

- Opinion states: Pro-vaccine (i.e. $o = 1$) and anti-vaccine (i.e. $o = 0$)
- Epidemic states: Susceptible (S), Infected (E, I) or Vaccinated (V) individuals.

This is encoded in each agents, thus the simulation would update the 2 layers together.

All susceptible agents are prone to be infected by means of interactions. In this study, each agent will interact with another agent within the population. If one of the agents within the interaction has already been infected, there is a transmission probability β which addresses if such interaction causes the disease pass from to one another. Figure 1.4 shows this transition between $S \rightarrow E$ epidemic states differs from the two parties. For the case of pro-vaccination groups, this transition is βI. Each infected agents will recover upon on a recovery probability γ. Thus the agent will transfer from $E \rightarrow S$ epidemic state. However, not all vaccine has permanent immunity, but wear off dependent to their health conditions and how the infected proportion. This is modelled as a probability of ϕ, hence the agents will transfer from $V \rightarrow S$ epidemic state. When $\phi = 0$ the vaccine has no effect to protect the population, where $\phi = 1$

represents permanent immunity. The connection between the opinion and epidemic layers are connected when a person is pro-vaccine, hence these agents may take the vaccine governed by α. The biggest different between the people whom are pro-vaccine or not, in terms of the simulation, is that people whom expressed to be anti-vaccine will not be considered to take vaccine at that time step. This may change through updates in the local information groups.

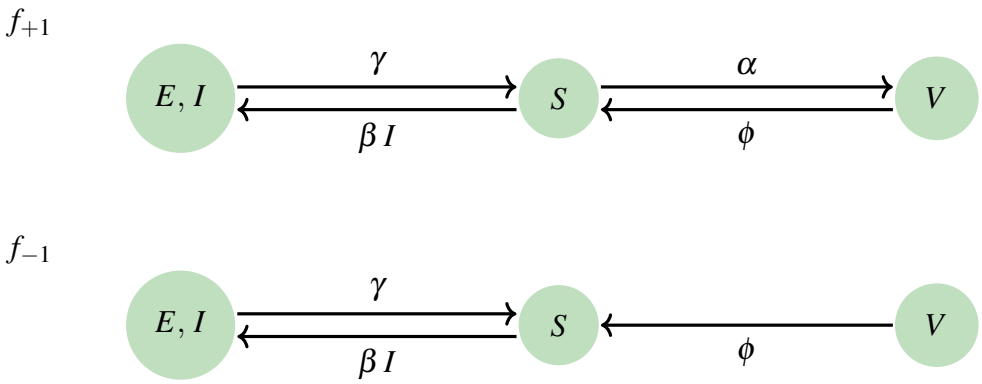


Figure 1.4: Transitions between epidemic status for anti-vaccine population f_{-1} and pro-vaccine population f_{+1}.

Chapter 2

Model and Methods

A series of Monte Carlo simulations are proceed. This is created a software package written in Python. The code is located in the appendix. This chapter serves to report on the computational model to the theory introduced in the Introduction chapter.

To start, the baseline parameters are tuned. This study proceeds by tuning from different values determining the basic reproductive ratio. Basic reproductive ratio R_0 is commonly used in epidemiology where it measures the secondary infection into an uninfected population. It is commonly defined as a ratio of the infection rate β and recovery rate γ. This means

$$R_0 = \frac{\beta}{\gamma} \tag{2.1}$$

In this study, we consider $R_0 = 2.6$ and where $\beta = 0.14$ and $\gamma = 0.05$ is used.

2.1 Opinion Dynamics

Consider a population composed of N agents. Each agents holds either opinions of pro-vaccination or anti-vaccination at time t. From the simulation point of view, pro-vaccination has opinion value of 1 while the opposite represents 0. This study assigns the opinions to all agents at the start and assign each to a group sizes of r based on their ID. Each agent has opinion updated at the end

of each time step.

The population may be composed by different personalities. This study considers the following personalities:

1. **Conformer**: This agent updates its opinions based on the local group majority decision.

2. **Inflexible**: This agent hold opinion invariant to time steps. They are positioned either to be pro-vaccine or anti-vaccination.

3. **Balancer**: This agent is a contrarian to the local majority decision. Once all local groups have formed their opinions at time step t, they will oppose the local majority opinion at time step $t + 1$.

The plain situation means the population is composed by conformers only. In simple words, conformers are memoryless, but simply follow where the majority opinion of the local group they located. For each local groups, all members will update their opinions when the total of their pro-vaccination opinions a_t (at each time step) is over half. This is defined as

$$a_t = \sum_{n=1}^{r} a_{t,n} \tag{2.2}$$

where when a_t is greater than $\lfloor \frac{a_t}{2} \rfloor$, all members of the local group become pro-vaccination.

The inflexibles will retain their opinion from the start of the simulation, and after each time step. The balancers opposes the local majority consensus after each update. Therefore, it is predicted that some of the local groups does not have unanimous support or against to vaccination.

2.2 Simulation Method

At the start of the simulation, a set number of people is set as seeds. In all simulations, these are 4 people as from the initial number of infections in the state of New South Wales in Australia (NSW Health 2020).

Initial opinions are subjected to binomial distribution Bin(N, p) where p is the probability of people support vaccination. Therefore, it is not guaranteed where exact proportion of pro-vaccination corresponds to the specified p. Agents will have their opinions and epidemic states updated at each time step. Specifically, this involves with the following steps (according to the working code):

1. Epidemic component

    a) For all vaccinated people, they are subjected to a wear-off probability ϕ. The higher the value of ϕ, the less likely the vaccinated population will wear off their immunity. This value of ϕ can be pre-defined or subjected by the preset values of the vaccine type.

    b) Infected people in the E compartment will subject to a test rate. Which includes a sure positive testing if tested. Then they may transfer to the I component.

    c) If any people are pro-vaccine at that time step when information network is activated or otherwise, they are eligible to be vaccinated. This is subjected to the adoption probability α. All vaccinated agents are not probable to be infected prior their immunity wore off.

    d) Alternatively, any infected agents at that time step are possible to recover subject to the probability γ.

    e) Each people is infected by a fixed infection probability β. Depends on the options (mode) activated, they may alter the infection process differently. For example, the process under network is depicted below.

    f) Every infected people may able to recover based on a probability of γ. For each person recovered within the immune period, they will not be infected again.

    g) While they may also subject a removal probability δ.

2. Contact network component

    a) For each edges on the network, only the edge that has either side were infected will considered for the Monte Carlo simulation. Otherwise, this edge is skipped. This is the same when one edge is in the I component. Where they are infected with symptoms shown. We assume these people are quarantined.

    b) The susceptible person is therefore infected under a probability β. This will replace the relevant step in the epidemic layer itself.

    c) After the epidemic layer has done all their procedures. The contact network will undergo a rewiring update. In this study we use the Xulvi-Brunet Sokolov algorithm to control the rewiring.

3. Opinion component

    a) For any agents who are inflexibles, their memory are stored temporarily before any updates.

    b) All agents will follow the local majority rule. Thus, all agents at this point of the time step, their opinion will follow the local group consensus.

    c) For any agents who are inflexibles or balancers, their opinions are updated based on their personality. For instance,

        - The inflexibles will restore their opinions now.
        - The balancers will change their opinions now, which is guaranteed to contrary to the local majority consensus.

    d) At the end of the time step, each agents will rotate their groups. Algorithmically, this means their group number is reassigned.

These time steps repeats upon each iteration.

Chapter 3

Literature Review

There are many studies in regards to explaining the mobility patterns, demographics of transmitting SARS-COV-2. This may due to the wide spread nature of the pandemic, it may also be fortunate where quantitative insights are available to discover and predict the infection curve. This chapter serves the review of current literature and how they contribute to the findings of epidemic modelling during this time.

3.1 The Importance of Game Theoretical Modelling in Epidemiology

Using game theoretical modelling considers behaviour aspects into compartment modelling. This approach consider a person benchmarking their decisions and choose the option that brings them with the maximum utility among all alternatives. Zhao, Bauch, and He (2018) modelled agent's options to travel overseas (or not) and observed that government implementations of travel restrictions may reduce the severity of 2003 SARS outbreak. In their study, they have incorporated utilities from travelling overseas (or not) as the factor of the transition probabilities with the reward companioned. Thus it can be very easily used in compartment model, for example, people whom travelled are subject to the transmission probability and may be infected.

There is another aspect where game theory stands out from modelling behaviours from Zhao, Bauch, and He (2018). In each games, each person made decisions subjected by others. The famous example is the prisoners dilemma, when one person choose a strategy (either confess or defect), their sentence differs because of other's choice. In terms of epidemic models, decision alternatives are characterised by additional terms due to their altered choice (e.g. Chang et al. (2020)). For example, Chang et al. (2020) has listed when modelling choices to vaccination. An additional term can be used for encountering the perceived probability to be infected ("perceived transmissibility") from proportion of infected and deaths. The earlier example is the case where agents self-learn from their experience, while the latter captures information by imitating from the environment. This study encounters several socio-economical factors relates to the transmission of SARS-COV-2. Each agents learn from their own experiences and generate the complex behaviours. For instance, Freire et al. (2020) proposed a social decision making game upon several decisions that affects the COVID-19 transmission. While they have considered a classic matrix-form games. Unlike Zhao, Bauch, and He (2018) or the literatures discussed in Chang et al. (2020), the authors initiated an agent based modelling with Control-based Reinforcement Learning, where the agents learn to adapt the environment better. Our approach takes some elements of the literatures mentioned, where the merit of agent based modelling is taken to encounter the diversity within the population, the use of game theory becomes the core component to drive the decision making. In this study, the scope remains where the tools discussed by Chang et al. (2020) are used. This is for exploratory purpose and with the tools mentioned have a mature development, it will be easier to develop a model from here.

3.2 The Importance of Agent-based Modelling in Epidemiology

Using agent-based modelling captures a non-linear dynamics into compartment modelling. Traditionally, simulating collective actions was done using equation based modelling or mean field theorem. Where it was criticised when such methodology ignores the feedback due to heterogeneity of humans and complex systems agents (Stauffer 2013). With the significant improvement in computational powers. There is a rising trend upon using agent based models.

Opinion dynamics is a new area that studies how opinions influenced. There are several base models in this area, such as the voter model (Sznajd-Weron, Szwabiński, and Weron 2014), or the Sznajd model (Sznajd-Weron and Sznajd 2000). One of the cornerstone model is the local majority theorem by Galam (2012). Which states that people form local groups, such as family, friends group or casual social groups. At the end of the social gathering, all members express the majority opinion and move to a new group. This simple model is powerful in describing how opinion forms and dissipates. There are many further additions (e.g. Crokidakis, Blanco, and Anteneodo 2014 with three options) and applications in political science (e.g. Ramos et al. 2015) and social networks (e.g. Zhang et al. 2014 and Alvarez-Zuzek et al. 2017) since its discovery.

Epidemiology is one of key applications. For example, Pires and Crokidakis (2017) approached the problem by combining the two layers together. They changed the epidemic parameters (i.e. α, β, γ, ϕ as in this study) and found that the disease will eradicate when majority are pro-vaccination. While high social engagement will lead to the eradicate of disease even when minority supports vaccination. Xia and Liu (2013) had combined opinion dynamics and social impact theory. Where the theory assess how individuals can be source or objects of influence in adapting vaccination. Voinson, Billiard, and Alvergne (2015) considered a cognitive feedback loop with human biases and claimed

that would never eradicate the infection. Rather, it creates oscillations in vaccination coverage depending on individual opinions. Each of the findings based on either the SIS or SIR model from epidemiology, they sought social parameters that are irrelevant to other studies. The diverse approach on customising the local majority model provides an exciting insight to understand epidemics. It is also the best method when society is parameterised by multitude factors. Since this is a new research area, it is hard to find the comparative studies to link between the examples. Rather this study builds on the model from Pires and Crokidakis (2017). Where the parameters are measurable from the macroscopic point of view. However, the model provided by Pires and Crokidakis (2017) is prone to situations like hung vote as described in Galam (2004, cited in Cheon and Morimoto 2016). When the social issue is highly controversial, it is more realistic to see opinions oscillate due to intense debates. It may therefore be advantageous to consider work from Cheon and Morimoto (2016) with heterogeneous personalities among the population. They have referenced from Galam's earlier models and introduced the concepts of inflexibles (i.e. in Galam (2004)), while created the idea of balancers based on the addition of personalities.

Anti-vaccination movement remains as a long going social issue, and it should attract more research to understand how their opinion influence the population. This literature review highlights the need of opinion dynamics to form more holistic approach to improve our health system. Furthermore, several relevant literatures have been compared. Their approaches diverged from their modelling and the parameters taken in account. This study takes route by making additions to the model from Pires and Crokidakis (2017) and Cheon and Morimoto (2016).

The literature of COVID-19 and opinion dynamics are rather not enriched, compared to literature regarding to game theoretical modelling. This possibly due to the emerging nature of the outbreak. It also shows some opportunity to develop from the literature and apply the COVID-19 situation into vaccine adoption.

Chapter 4

Result

The following presents the simulation results.

4.1 Immunity Time

Figure 4.1 shows the epidemic due to different immunity time. In this study we present the results when immunity time are 0 days, 60 days, 180 days and 210 days. The curve corresponding to no immunity after recovery has a very slow drop of the curve, while they have the tallest infected peak among all immunity periods. The period of immunity seems to have effect upon the shape. For example, the curve represents 60 days of immunity behaves as a decaying oscillatory behaviour. While the 210 days period of immunity shows the disease may pause until a second wave comes back after 210 days since the epidemic started. The curve corresponds to 180 days appears the second wave earlier than the curve corresponding to 210 days immune time. The time where the second wave comes back is appeared to start after the specified immune period has over, from the start of the simulation.

4.2 Transmission on the Longitudinal Network

Close physical contacts through longitudinal networks has effect on disease spread. Figure 4.2 shows different number of new links formed in a longitudi-

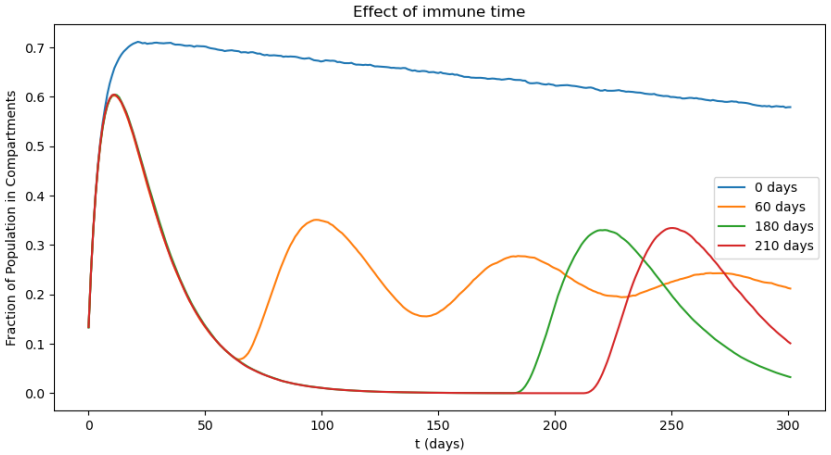


Figure 4.1: Results of transmission under different immunity time. Patients are immune to COVID-19 under a fixed, and after that they may prone to re-infection again.

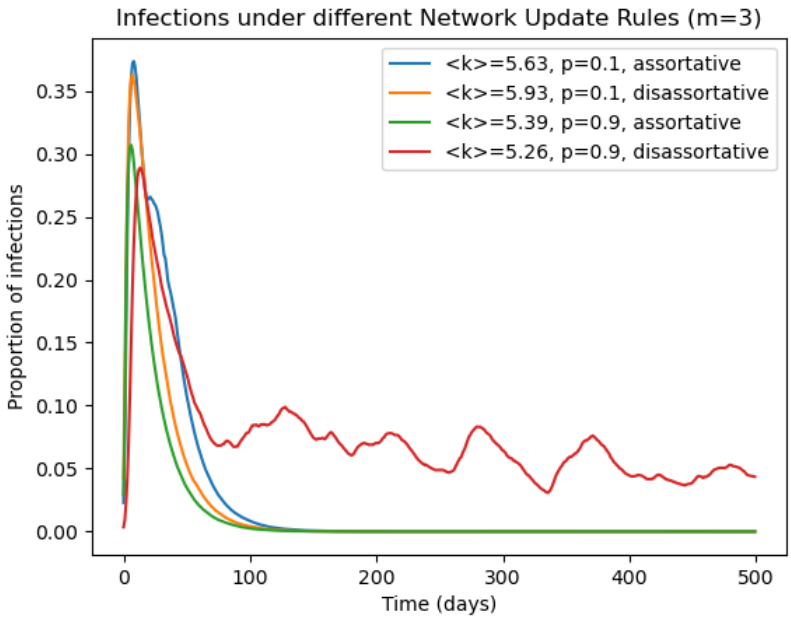
nal network has effect in the peak and sometimes its shape. For example, in most cases the epidemic has one peak and it will be eradicated afterwards. In fig. 4.2a, the case where the rewiring probability is often (i.e. $p = 0.9$) then the peak is lower than low rewiring probability. In addition to that, when the contact network uses disassortative update, the epidemic does not eradicate but fluctuates after some point.

Figure 4.2b shows the results when 20 number of new links formed in each new nodes. Most cases in the figure has one peak and drops to 0 afterwards. The only cases flattens at 0 all the time is when the rewiring probability is $p = 0.9$ and using dissasortative wiring. The average degree and assortativity during the network updates are shown in fig. 4.3 and fig. 4.4.

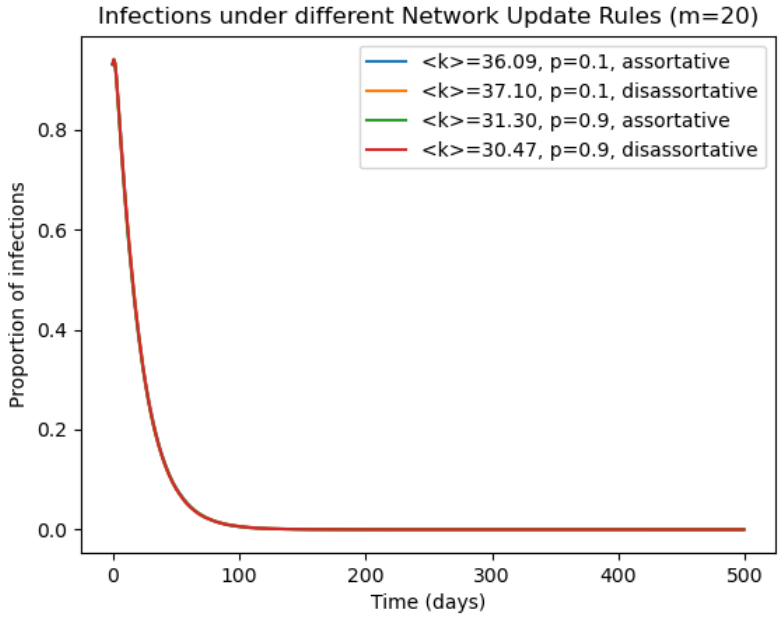
4.3 Bounded Rationality

Using eq. (1.6), we obtain the probability to adapt vaccine under different rationality parameters λ. This is shown in fig. 4.5. Under the consideration of bounded rationality, the minimum adoption probability $P(X)$ is 0.5. This is where $\lambda = 0$ represents.

When we consider the effect or reward (cost) due to infection r_I, which may



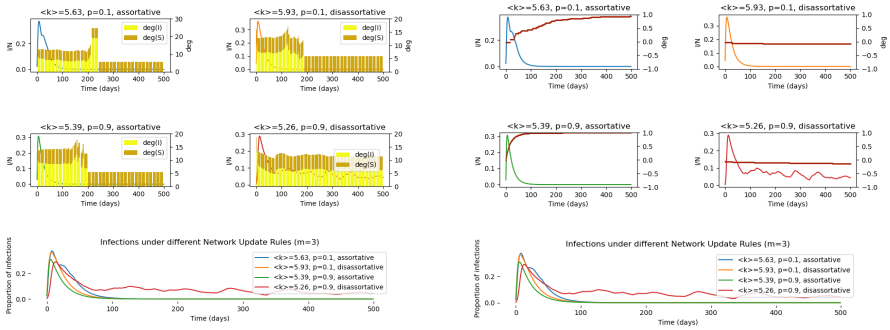
(a) $m = 3$



(b) $m = 20$

Figure 4.2: Results of transmission under different number of new links formed in contact network. ($N = 10000, T = 500, \alpha = 0, \beta = 0.14, \gamma = 0.05, \delta = 0.00005$)

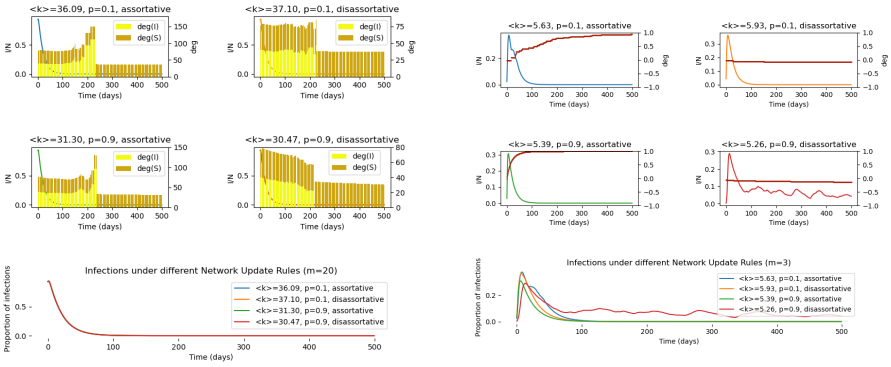
happen when one choose not to vaccinate. We can see that the higher the value of r_I, the likely that a person to vaccinate even with low reward from vaccinate



(a) Average degree

(b) Assortativity change

Figure 4.3: Average degree and assortativity at each time step for the contact (longitudinal social) network shown in fig. 4.2a.



(a) Average degree

(b) Assortativity change

Figure 4.4: Average degree and assortativity at each time step for the contact (longitudinal social) network shown in fig. 4.2b.

itself. This is shown in fig. 4.6, where the different lines represents a different ratio between r_V and r_I. The higher the reward (cost) from infection, the higher the probability to take vaccine. The ratio between r_V and r_I has effect on vaccinate. Figure 4.7 and fig. 4.5 shows a comparison when there is a higher reward (cost) due to infection.

When we apply the theoretical results into simulation. In this study we choose $r_V = 1$ and $r_I = -10$ and compared the vaccination rate at two different adoption rate α. It shows that the higher the adoption rate, the more likely the population may vaccinate. The effect of rationality parameter λ is proportional to higher probability to vaccinate. This is shown in fig. 4.8.

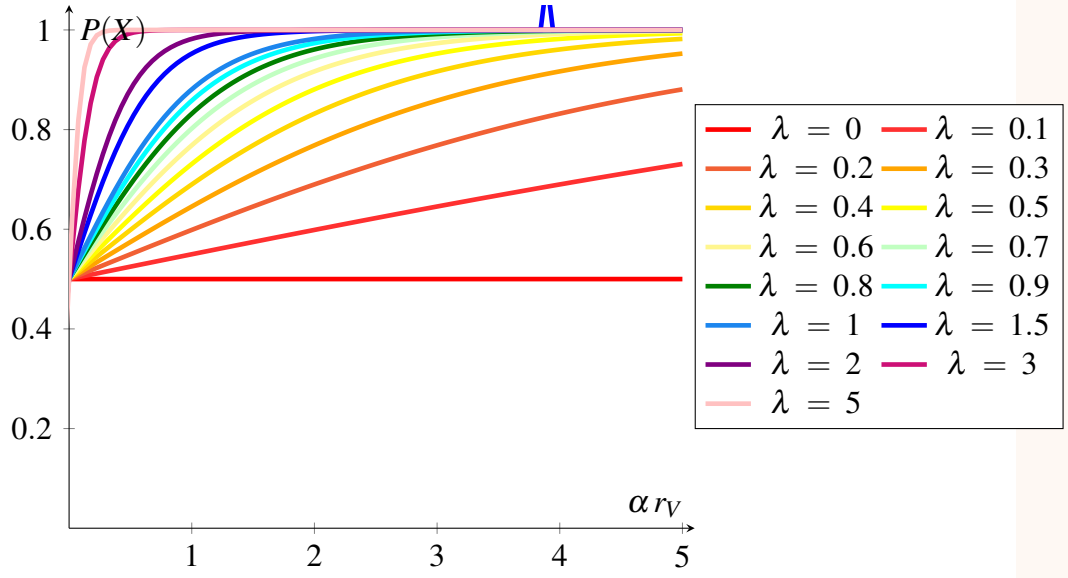


Figure 4.5: Rationality parameter λ and probability to take vaccine. Assume $r_V = 1$ and $r_I = -1$.

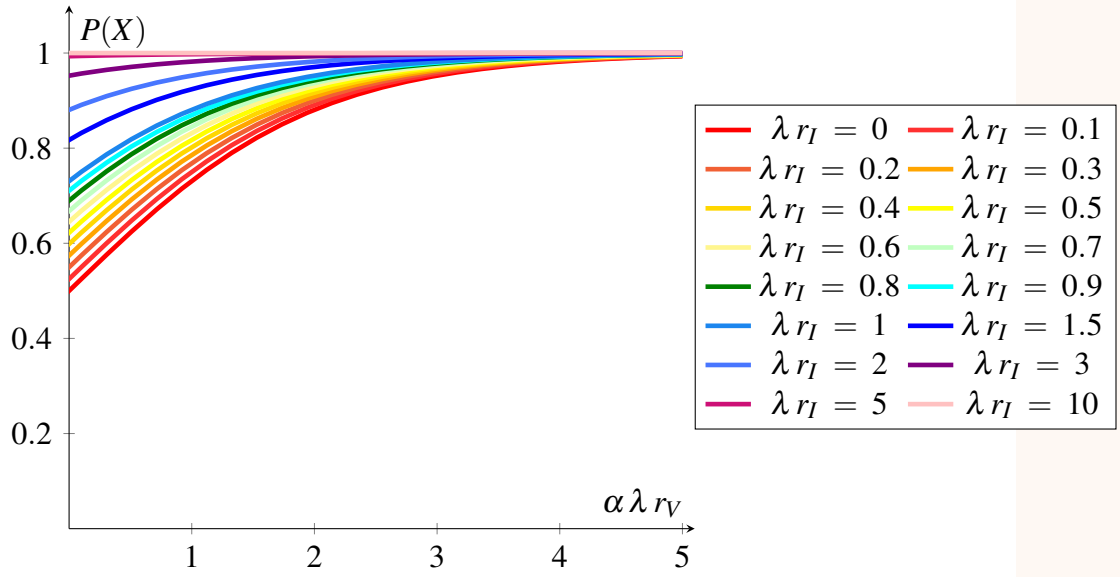


Figure 4.6: Rewards r_I and probability to take vaccine when bounded rationality applies.

4.4 Opinion Dynamics

The proportion of inflexibles and balancers associates in different proportions of pro-vaccination f_{+1}. In particular, in fig. 4.9, there is an inflexion point around 0.1 to 0.2 of balancers. The higher the proportion of balancers, the likely the f_{+1} converges to 0.5 of the total population. While the proportion of bal-

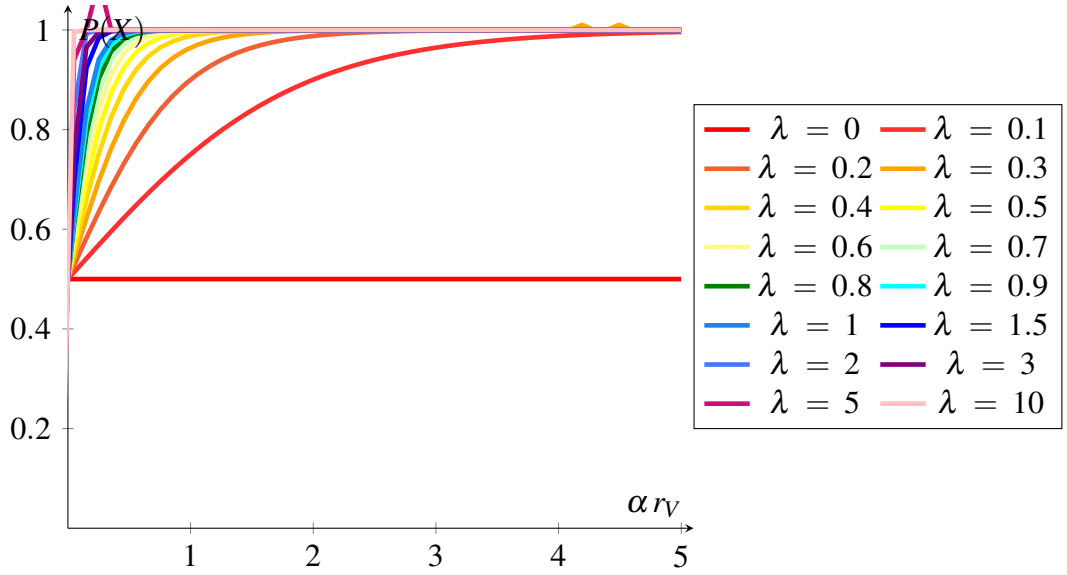
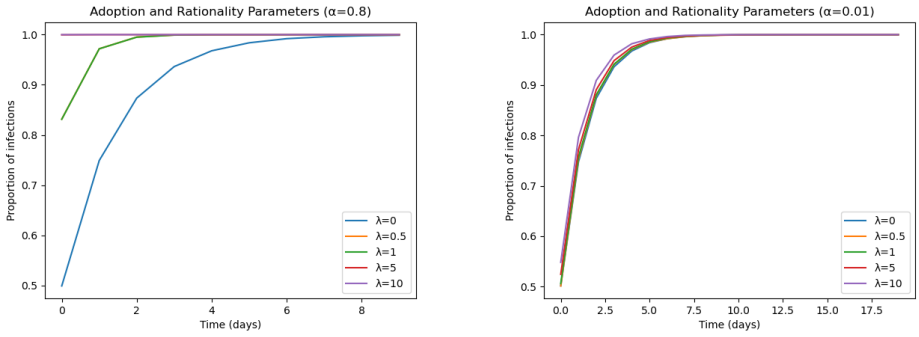


Figure 4.7: Rationality parameter λ and probability to take vaccine. Assume $r_V = 1$ and $r_I = -10$.



(a) $\alpha = 0.8$ (b) $\alpha = 0.01$

Figure 4.8: Results of transmission under different rationality parameters. Using $r_V = 1$ and $r_I = -1$.

ancers becomes higher and so as inflexibles, they associate with a lower f_{+1} in the long run. As shown in fig. 4.10, high fluctuations only shown when balancers dominates, and over 0.8 or above. The proportion of inflexibles are low as 0.1. At other mixes of balancers and inflexibles, the standard deviation of f_{+1} remains low. Which means the value fluctuates at a stable level.

The proportion of inflexibles and balancers have effect on the long term infected number and vaccination adoption. Figure 4.11a shows the higher proportion of inflexibles, the epidemic will sustain at a higher level. While the higher

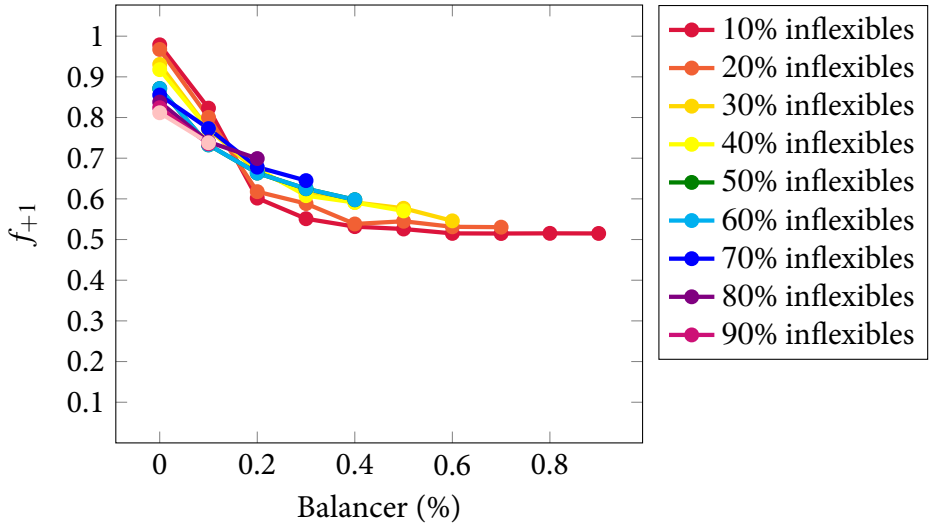


Figure 4.9: f_{+1} at different inflexibles and balancer levels (at x-axis). Initially $f_{+1} = 0.8$.

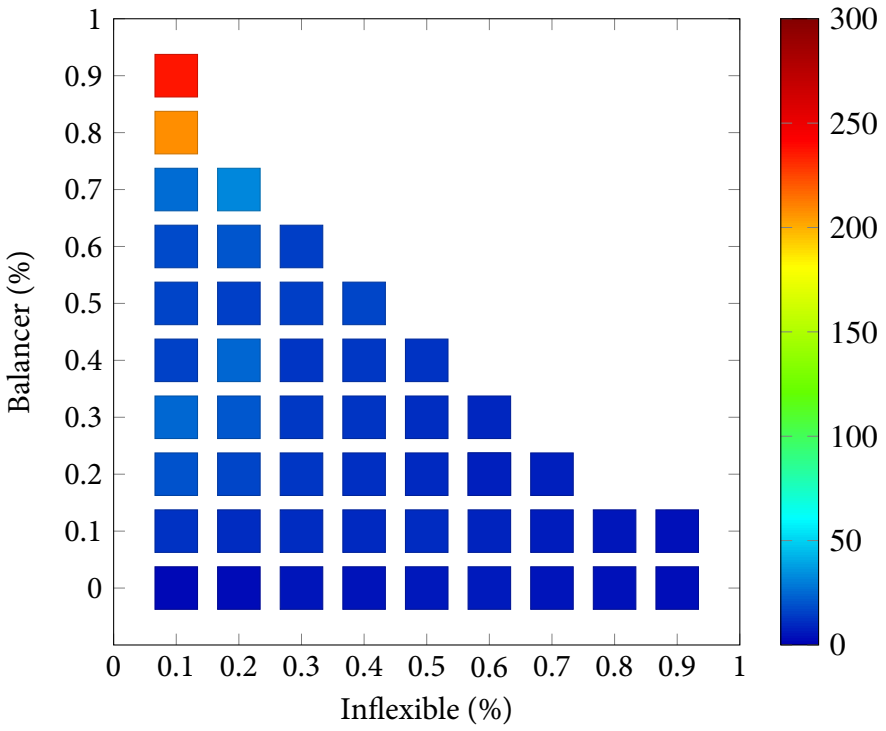
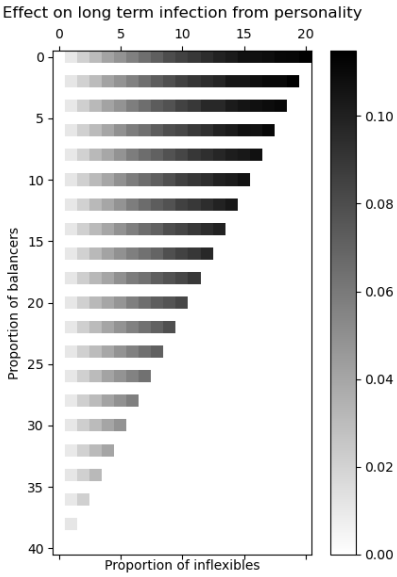
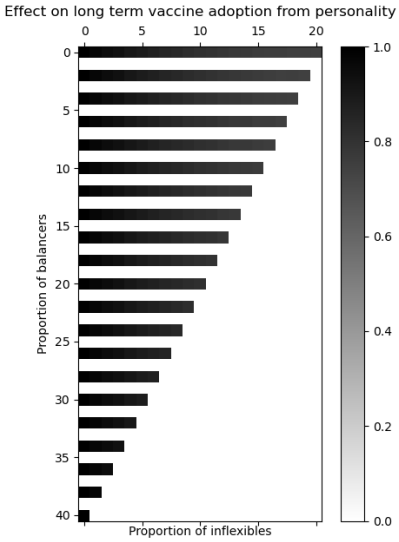


Figure 4.10: Standard deviation of f_{+1} at different inflexibles and balancer levels (at x-axis). Initially $f_{+1} = 0.8$.

the inflexible levels, there are a weak disproportion of vaccination adoption (In fig. 4.11b). However, the overall adoption has been high at long time after the vaccine is available.



(a) Effect on infection



(b) Effect on vaccination

Figure 4.11: Results of infection and vaccinated proportion at time $t = 500$.

Chapter 5

Discussion

The broad approach to predict the COVID-19 pandemic is that it produces some interesting results. Firstly, the inclusion of immune period states that its existence contributes the epidemic curve in reality. In reality, the pandemic curve has a bell shape, where the simulation results shows so. The situation where immunity period is absent, the epidemic curve is slowly falling down. This may be because where any person whom recovered, they are likely to infected again.

The more realistic pattern occurred to be a sufficiently elongated immune period. It remains an outstanding finding where when immune period is equipped in this simulation, the curves starts with a common bell shape. This initial bell shape elapses for 60 days, and if the immune period is longer, we can see the curve has sufficient time to go down to minimal level until the second wave. While the curve corresponds to 60 days in fig. 4.1, it seems where the population are able to contact the disease again, the curve rises from that level of proportion. It explains why such curve behaves as a decaying oscillation.

It is however, at this stage there are no vaccine available on the market. It may be hard to infer the vaccine adoption rate at this point.

Apart from the immune time, social contacts is one of the most mentioned factors in transmitting COVID-19. The findings here coincide with the intuition where the more social contacts will cause the intensity of the infection,

but not necessary the period of the epidemic. The continuum of epidemic relies on an eligible person (i.e. susceptible but not immune) to have contacts with someone whom infected but not quarantined. Without further introduction of the disease from outside, it is hard for the epidemic to elongated. Thus the only peak in fig. 4.2.

The average degree may have some implications to the epidemic. The larger the average degree, the higher the epidemic peak. Which is shown in fig. 4.2a and fig. 4.2b. The immune period is connected to this result when a person recovers, then it becomes a blockage to spread the disease. The fact that a person is quarantined are also important as well. These two factors may explains why in a highly connected network, when the average degree of the infected is high but the epidemic did not survive.

This study also considers bounded rationality when one considers to vaccinate. Under this framework, the adoption is very quick with very low utility to vaccinate. The lowest bound of vaccinate is 0.5 where it presents random decisions. Where the higher the rationality parameter, we can see more adoption is likely. It mean lead to fast full adoption from the population as in fig. 4.8.

It is however, rationality does not imply information available. In this study we assume perfect information so that everyone are aware of the reward or costs from vaccination r_V and infection r_I. The ratio between r_V and r_I shows how quick to adopt vaccine. This speed is proportional to r_I to r_V ratio.

The role of inflexibles and balancers add the fluctuation into the time series of f_{+1}. In general, the levels of inflexibles and/ or balancers would generate the similar level of fluctuations, as shown in fig. 4.10 (except in extreme balancer levels). The levels of inflexibles and/ or balancers does not create a uniform association to f_{+1} accross all levels of inflexibles and/ or balancers. Figure 4.9 illustrates an inflexion point occurs when the level of balancers are between 0.1 to 0.2. This adds the second layer of complexity to intervene our society upon adapting vaccinations: When majority are pro-vaccination, low levels of balancers would be beneficial with low level of inflexibles. While high levels of balancers would be beneficial with high level of inflexibles. This means implying

that when inflexibles stay still in their position, balancers will try to oppose the group's decision and improve the social outcome. By symmetry, this is the opposite with minority of pro-vaccination initially.

Chapter 5 Discussion

Chapter 6

Conclusion and Outlook

This report presents the proposal to model SARS-COV-2 vaccine adoption and to predict the pandemic under different considerations of behavioural factors. For example, the inclusion of immune time makes the simulation realistic and it produces the second wave. The inclusion of longitudinal social networks shows implications of the social distancing and infection. While bounded rationality and opinion dynamics has implications to vaccine adoption from the individual or close social contact level.

One outstanding work from here is to simulate the combined factors to predict the epidemic. In this way we can see a more realistic picture when these are integral part of the society. For example, there is a possibility where the opinion dynamics and game theory can be combined: While opinion dynamics concerns how people form decisions. Game theory concerns the impact made by decisions, and more interestingly how mutual consensus leads to non-trivial observations. Another consideration that is not part of this report is the crossover of demographic profiles, as outlined from this report age and gender are few of the prominent examples that characterises this epidemic. The simulation has well considered the heterogeneity of the society and did not assimilate this diversity. The agent based modelling also considers the feedback loop based on this consideration.

In review of this study, there are few point that should note as caveat, if mov-

ing on. The real value of game theoretical rewards should be investigated. For example, the reward of vaccination and the cost that the infection brings. In this study the ration of both is highlighted when discussing the results from bounded rationality. The (realistic) convertible value to reality is important because we generate a deeper understanding of this pandemic up to an individual level. For example, the meaning of infection differs between people. While the young generation is more likely to recover, they may value such infection lesser.

In addition to that, the bounded rationality model used has a lower bound of vaccinate probability is 0.5. This restricts the option when there is a pessimistic demand of vaccine.

The other issue is where the contact network applies to the simulation, no external infection can be made to the population. This contributes to the results in the longitudinal network with one epidemic peaks. One idea to replicate the subsequent waves is to model the infection brought from overseas. The model is outlines in the Introduction section and further work is required to produce the findings. Overall, the work has been produced in this report considers most of behavioural and medical factors that causes this pandemic, and it can be encoded within one simulation package. When there is a need of adding more layers to the simulation, the code is flexible to include those.

Bibliography

Abou-Ismail, A. (2020). "Compartmental Models of the COVID-19 Pandemic for Physicians and Physician-Scientists." en. In: **SN comprehensive clinical medicine**, pp. 1–7. ISSN: 2523-8973. URL: http://search.proquest.com/docview/2437118399/.

Alvarez-Zuzek, L., C. La Rocca, J. Iglesias, and L. A. Braunstein (2017). "Epidemic spreading in multiplex networks influenced by opinion exchanges on vaccination.(Research Article)(Report)". en. In: **PLoS ONE** 12.11, e0186492. ISSN: 1932-6203.

Bhattacharyya, S., A. Vutha, and C. T. Bauch (2019). "The impact of rare but severe vaccine adverse events on behaviour-disease dynamics: a network model." en. In: **Scientific reports** 9.1, p. 7164. ISSN: 2045-2322. URL: http://search.proquest.com/docview/2231910970/.

Borghesi, C. and S. Galam (2006). "Chaotic, staggered, and polarized dynamics in opinion forming: The contrarian effect". In: **Physical Review E** 73.6, p. 066118.

Borghesi, C., J.-C. Raynal, and J.-P. Bouchaud (2012). "Election Turnout Statistics in Many Countries: Similarities, Differences, and a Diffusive Field Model for Decision-Making". In: **PLOS ONE** 7.5, pp. 1–12. DOI: 10.1371/journal.pone.0036289. URL: https://doi.org/10.1371/journal.pone.0036289.

Centers for Disease Control and Prevention (2016). **History of Smallpox**. en. [Online; accessed 29-November-2020]. URL: https://www.cdc.gov/smallpox/history/history.html.

Bibliography

Centers for Disease Control and Prevention (2020). **How COVID-19 Spreads**. en. [Online; accessed 29-November-2020]. URL: https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/how-covid-spreads.html?CDC_AA_refVal=https%3A%2F%2Fwww.cdc.gov%2Fcoronavirus%2F2019-ncov%2Fprepare%2Ftransmission.html.

Chang, S., M. Piraveenan, P. Pattison, and M. Prokopenko (2020). "Game theoretic modelling of infectious disease dynamics and intervention methods: a review". eng. In: **Journal of Biological Dynamics** 14.1, pp. 57–89. ISSN: 1751-3758. URL: http://www.tandfonline.com/doi/abs/10.1080/17513758.2020.1720322.

Cheon, T. and J. Morimoto (2016). "Balancer effects in opinion dynamics". en. In: **Physics Letters A** 380.3, pp. 429, 434. ISSN: 0375-9601.

Crokidakis, N., V. H. Blanco, and C. Anteneodo (2014). "Impact of contrarians and intransigents in a kinetic model of opinion dynamics". In: **Phys. Rev. E** 89 (1), p. 013310. DOI: 10.1103/PhysRevE.89.013310. URL: https://link.aps.org/doi/10.1103/PhysRevE.89.013310.

Fernández-Gracia, J., K. Suchecki, J. J. Ramasco, M. San Miguel, and V. M. Eguíluz (2014). "Is the Voter Model a Model for Voters?" In: **Phys. Rev. Lett.** 112 (15), p. 158701. DOI: 10.1103/PhysRevLett.112.158701. URL: https://link.aps.org/doi/10.1103/PhysRevLett.112.158701.

Freire, I., C. Moulin-Frier, M. Sanchez-Fibla, X. D. Arsiwalla, P. F. Verschure, and J. Vyrastekova (2020). "Modeling the formation of social conventions from embodied real-time interactions". en. In: **PloS one** 15.6, e0234434–e0234434. ISSN: 1932-6203.

Galam, S. (2004). "Contrarian deterministic effects on opinion dynamics:"the hung elections scenario"". In: **Physica A: Statistical Mechanics and its Applications** 333, pp. 453–460.

– (2012). **Sociophysics : A Physicist's Modeling of Psycho-political Phenomena**. en. 1st ed. Bibliographic Level Mode of Issuance: Monograph. Springer. ISBN: 9781461420323.

Galam, S. and F. Jacobs (2007). "The role of inflexible minorities in the breaking of democratic opinion dynamics". In: **Physica A: Statistical Mechanics and its Applications** 381, pp. 366–376.

Kasthurirathna, D., M. Harre, and M. Piraveenan (2016). "Optimising influence in social networks using bounded rationality models". en. In: **Social network analysis and mining** 6.1, pp. 1–14. ISSN: 1869-5450.

Moderna (2020). **Moderna's COVID-19 Vaccine Candidate Meets its Primary Efficacy Endpoint in the First Interim Analysis of the Phase 3 COVE Study**. en. [Online; accessed 29-November-2020]. URL: https://investors.modernatx.com/news-releases/news-release-details/modernas-covid-19-vaccine-candidate-meets-its-primary-efficacy/.

NSW Health (2020). **Find the facts about COVID-19**. en. [Online; accessed 27-November-2020]. URL: https://www.nsw.gov.au/covid-19/find-facts-about-covid-19.

Oxford University (2020). **Oxford University breakthrough on global COVID-19 vaccine**. en. [Online; accessed 29-November-2020]. URL: https://www.research.ox.ac.uk/Article/2020-11-23-oxford-university-breakthrough-on-global-covid-19-vaccine.

Pfizer (2020). **PFIZER AND BIONTECH ANNOUNCE VACCINE CANDIDATE AGAINST COVID-19 ACHIEVED SUCCESS IN FIRST INTERIM ANALYSIS FROM PHASE 3 STUDY**. en. [Online; accessed 29-November-2020]. URL: https://www.pfizer.com/news/press-release/press-release-detail/pfizer-and-biontech-announce-vaccine-candidate-against.

Pires, M. and N. Crokidakis (2017). "Dynamics of epidemic spreading with vaccination: Impact of social pressure and engagement". In: **Physica A: Statistical Mechanics and its Applications** 467, pp. 167–179. ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2016.10.004. URL: http://www.sciencedirect.com/science/article/pii/S0378437116306914.

Ramos, M., J. Shao, S. D. S. Reis, C. Anteneodo, J. S. Andrade, S. Havlin, and H. A. Makse (2015). "How does public opinion become extreme?" en. In: **Scientific Reports (Nature Publisher Group)** 5, p. 10032.

Stauffer, D. (2013). "A Biased Review of Sociophysics". en. In: **Journal of Statistical Physics** 151.1, p.9–20. ISSN: 1572-9613. DOI: 10.1007/s10955-012-0604-9. URL: https://doi.org/10.1007/s10955-012-0604-9.

Sznajd-Weron, K. and J. Sznajd (2000). "Opinion evolution in closed community". In: **International Journal of Modern Physics C** 11.06, pp. 1157–1165.

Sznajd-Weron, K., J. Szwabiński, and R. Weron (2014). "Is the Person-Situation Debate Important for Agent-Based Modeling and Vice-Versa?" In: **PLOS ONE** 9.11, pp. 1–7. DOI: 10.1371/journal.pone.0112203. URL: https://doi.org/10.1371/journal.pone.0112203.

Voinson, M., S. Billiard, and A. Alvergne (2015). "Beyond Rational Decision-Making: Modelling the Influence of Cognitive Biases on the Dynamics of Vaccination Coverage". In: **PLOS ONE** 10.11, pp. 1–17. DOI: 10.1371/journal.pone.0142990. URL: https://doi.org/10.1371/journal.pone.0142990.

Wenham, C., J. Smith, and R. Morgan (2020). "COVID-19: the gendered impacts of the outbreak". en. In: **The Lancet (British edition)** 395.10227, pp. 846–848. ISSN: 0140-6736.

World Health Organisation (2020a). **Coronavirus disease (COVID-19) Weekly Epidemiological Update and Weekly Operational Update**. en. [Online; accessed 29-November-2020]. URL: https://www.who.int/emergencies/diseases/novel-coronavirus-2019/situation-reports.

– (2020b). **WHO Director-General's statement on IHR Emergency Committee on Novel Coronavirus (2019-nCoV)**. en. [Online; accessed 29-November-2020]. URL: https://www.who.int/dg/speeches/detail/who-director-general-s-statement-on-ihr-emergency-committee-on-novel-coronavirus-(2019-ncov).

World Health Organization (2010). **Statue commemorates smallpox eradication**. en. [Online; accessed 29-November-2020]. URL: https://www.who.int/csr/disease/smallpox/en/.

Xia, S. and J. Liu (2013). "A Computational Approach to Characterizing the Impact of Social Influence on Individuals' Vaccination Decision Making". In:

PLOS ONE 8.4, pp. 1–11. DOI: 10.1371/journal.pone.0060373. URL: https://doi.org/10.1371/journal.pone.0060373.

Zhang, H., M. Gao, W. Wang, and Z. Liu (2014). "Evolutionary prisoner's dilemma game on graphs and social networks with external constraint". en. In: **Journal of Theoretical Biology** 358, pp. 122–131. ISSN: 0022-5193.

Zhao, S., C. Bauch, and D. He (2018). "Strategic decision making about travel during disease outbreaks: a game theoretical approach". In: **Journal of the Royal Society Interface** 15.146. ISSN: 1742-5689.

Code

The following simulation code are written in Python. All simulations are started from the interface `main.py`. It controls the simulation through `simulation.py` which updates the epidemic states of each agents. Auxiliary files imports the classes to model the people and the epidemics. This chapter listed the following files:

- `simulation.py` is the wrapper of the simulation run.
- `epidemic.py` defines the epidemic and updates to the agents.
- `contact.py` defines the longitudinal social network.
- `group.py` defines the local majority rule and the information groups.
- `person.py` defines the agents.
- `mode.py` customised the simulation.
- `write.py` to export simulation data.

The code and its subsequence updates are also published on https://github.com/lt-shy-john/covid19-vaccine-game-theory.

Listing 6.1: `main.py`

```
1 # Import libraries
2 import sys
3 import time
4
5 # Import class files
```

Code

```
6  from person import Person
7  from group import Group
8  from simulation import Simulation
9  import mode
10 from contact import ContactNwk
11
12 '''
13 Main code
14
15 - cmd functions
16 - main loop
17 '''
18 def setting(N, T, alpha, beta, gamma, phi, delta,
       alpha_V, alpha_T, phi_V, phi_T, test_rate,
       immune_time, group_size, verbose_mode):
19     info = input('Information␣about␣the␣parameters?
           ␣[y/n]␣').lower()
20     print()
21     if info == 'y':
22         info_summ()
23     print('Leave␣blank␣if␣not␣changing␣the␣value(s)
           .')
24     N_temp = input('N␣>>>␣')
25     N = set_correct_para(N_temp, N, pos=True)
26     T_temp = input('T␣>>>␣')
27     T = set_correct_para(T_temp, T, pos=True)
28     alpha_temp = input('alpha␣>>>␣')
29     alpha = set_correct_epi_para(alpha_temp, alpha)
30     beta_temp = input('beta␣>>>␣')
31     beta = set_correct_epi_para(beta_temp, beta)
```

```
32      gamma_temp = input('gamma␣>>>␣')
33      gamma = set_correct_epi_para(gamma_temp, gamma)
34      phi_temp = input('phi␣>>>␣')
35      phi = set_correct_epi_para(phi_temp, phi)
36      delta_temp = input('delta␣>>>␣')
37      delta = set_correct_epi_para(delta_temp, delta)
38      cmd = input('Other␣parameters?␣[y/n]')
39      if cmd == 'y':
40          N, T, alpha, beta, gamma, phi, delta,
               alpha_V, alpha_T, phi_V, phi_T, test_rate
                = setting_other(N, T, alpha, beta, gamma
               , phi, delta, alpha_V, alpha_T, phi_V,
               phi_T, test_rate, group_size,
               verbose_mode)
41      population = Person.make_population(N)
42      return N, T, alpha, beta, gamma, phi, delta,
           alpha_V, alpha_T, phi_V, phi_T, test_rate,
           immune_time, group_size, verbose_mode
43
44  def setting_other(N, T, alpha, beta, gamma, phi,
       delta, alpha_V, alpha_T, phi_V, phi_T, test_rate,
        immune_time, group_size, verbose_mode):
45      print('Adoption␣parameters␣\n')
46      alpha_V_temp = input('Vaccine:␣')
47      alpha_V = set_correct_epi_para(alpha_V_temp,
           alpha_V)
48      alpha_T_temp = input('Treatment:␣')
49      alpha_T = set_correct_epi_para(alpha_T_temp,
           alpha_T)
50      print('\nTransmission␣parameters␣\n')
```

Code

```
51      pass
52      print('\nWear-off␣parameters␣\n')
53      phi_V_temp = input('Vaccine:␣')
54      phi_V = set_correct_epi_para(phi_V_temp, phi_V)
55      phi_T_temp = input('Treatment:␣')
56      phi_T = set_correct_epi_para(phi_T_temp, phi_T)
57      print('\nInfection␣related')
58      immune_time_temp = input('Immune␣time␣(days):␣'
           )
59      immune_time = set_correct_epi_para(
           immune_time_temp, immune_time)
60      print('\nTesting␣parameters␣\n')
61      test_rate_temp = input('COVID-19:␣')
62      test_rate = set_correct_epi_para(test_rate_temp
           , test_rate)
63      if 1 in modes:
64          print('\nYou␣have␣initiated␣mode␣1␣\n')
65      if 51 in modes or 52 in modes or 53 in modes or
            54 in modes:
66          print('\nYou␣have␣created␣contact␣network␣\
               n')
67          print('\nNetwork␣parameters␣\n')
68      if 21 in modes:
69          group_size_temp = input('Group␣size:␣')
70          group_size = set_correct_para(
               group_size_temp, group_size, pos=True)
71      cmd = input('Verbose␣mode?␣[y/n]')
72      if cmd == 'y':
73          verbose_mode = True
74      elif cmd == 'n':
```

```
75          verbose_mode = False
76      return N, T, alpha, beta, gamma, phi, delta,
            alpha_V, alpha_T, phi_V, phi_T, test_rate,
            immune_time, group_size, verbose_mode
77
78  def summary():
79      print('N: {}'.format(N))
80      print('T: {}'.format(T))
81      print('===== SIR Rate =====')
82      print('alpha: {}'.format(alpha))
83      print('beta: {}'.format(beta))
84      print('gamma: {}'.format(gamma))
85      print('phi: {}'.format(phi))
86      print('delta: {}'.format(delta))
87      cmd = input('Show other epidemic paremeters? [y
            /n] ')
88      if cmd == 'y':
89          print('alpha_V = {}'.format(alpha_V))
90          print('alpha_T = {}'.format(alpha_T))
91          print('immune time = {}'.format(immune_time
                ))
92          print('test rate: {}'.format(test_rate))
93      print()
94      info = input('Information about the parameters?
            [y/n] ').lower()
95      if info == 'y':
96          info_summ()
97      print()
98      if len(modes) > 0:
99          info = input('There are customised settings
```

Code

```
            . View them? [y/n] ')
        if info == 'y':
            for mode in modes.values():
                print(mode.__dict__)
    print()

def show_nwk():
    if contact_nwk.network != None:
        contact_nwk.show_nwk()
    else:
        print('Topology is not set, use command "
            MODE" to initiate them.')

def info_summ():
    print('N - Number of simulated agents.')
    print('T - Time steps/ period of simulation.')
    print('Alpha - Adoption of vaccination/ PrEP (
        willingness).')
    print('Beta - Infection rate.')
    print('Gamma - Recovery rate.')
    print('Delta - Removal rate.')
    print('Phi - Protection wear off rate.')
    print('Delta - Removal rate.')
    print('Tau - COVID-19 Testing rate. ')

def help():
    print('LOOK - View partner network.')
    print('MODE - Change mode settings.')
    print('RUN/ START - Start the simulation.')
    print('SETTING - Set simulation settings.')
```

```
127     print('OTHER␣SETTING␣-␣Set␣auxiliary␣simulation
            ␣parameters.')
128     print('SUMMARY␣-␣Print␣the␣simulation␣
            parameters.')
129     print('QUIT/␣Q␣-␣Quit␣the␣software.')
130
131 def usage():
132     print('Usage:␣python3␣main.py␣[(N)␣(T)␣(alpha)␣
            (beta)␣(gamma)␣(phi)␣(delta)]␣...\n\
133 ␣␣␣␣[-m␣␣<modes_config>]␣[-f␣(filename)]␣[-verbose␣
        |␣--v]␣[run]\n')
134     print('-immune_time␣\t␣Immune␣time␣after␣
            recovered,␣in␣days.')
135     print('-test_rate␣\t␣COVID-19␣testing␣rate.')
136     print('-m␣\t\t␣Mode')
137     print('␣␣␣␣--1␣\t␣Living␣in␣city/␣rural.')
138     print('␣␣␣␣--2␣\t␣Travelled␣back␣from␣overseas.
            ')
139     print('␣␣␣␣--4␣\t␣Bounded␣rationality␣of␣
            vaccine.')
140     print('␣␣␣␣--5␣\t␣Edit␣contact␣network.')
141     print('␣␣␣␣--7␣\t␣Age␣distribution.')
142     print('␣␣␣␣--8␣\t␣Gender␣distribution.')
143     print('␣␣␣␣--10␣\t␣Type␣of␣vaccine.')
144     print('␣␣␣␣--11␣\t␣Stop␣transmissability/␣
            reduce␣severity.')
145     print('␣␣␣␣--12␣\t␣Cost␣of␣vaccine.')
146     print('␣␣␣␣--13␣\t␣Accessibility␣to␣vaccine.')
147     print('␣␣␣␣--14␣\t␣Side␣effects␣of␣vaccine.')
148     print('␣␣␣␣--20␣\t␣Intimacy␣game.')
```

Code

```
149     print('␣␣␣␣--21␣\t␣Local␣majority␣rule.')
150     print('␣␣␣␣--22␣\t␣Stubbon␣to␣take␣vaccine.')
151     print('␣␣␣␣--23␣\t␣Stubbon␣to␣against␣vaccine.'
            )
152     print('␣␣␣␣--24␣\t␣Contrary␣to␣social␣groups.')
153     print('␣␣␣␣--31␣\t␣Medication␣incorporated.')
154     print('␣␣␣␣--41␣\t␣Moral␣hazard␣of␣social␣
            distancing.')
155     print('␣␣␣␣--42␣\t␣Moral␣hazard␣of␣treatment.')
156     print('␣␣␣␣--51␣\t␣Erdos-Renyi␣topology.')
157     print('␣␣␣␣--52␣\t␣Preferential␣attachment.')
158     print('␣␣␣␣--53␣\t␣Small␣world␣network.')
159     print('␣␣␣␣--54␣\t␣Lattice␣network.')
160     print('-f␣\t\t␣Export␣file␣name.')
161     print('-h␣\t\t␣Usage.')
162     print('run␣\t\t␣Run␣simulation,␣last␣argument.'
            )
163
164 def correct_para(p, pos=False):
165     '''
166     Convert the parameters into integers.
167
168     Parameters
169     p -- input.
170     - pos: If the parameter is positive number.
171     '''
172     try:
173         p_num = int(p)
174         if pos == True and p_num < 1:
175             p_num = 1
```

```
176         return p_num
177     except ValueError:
178         p_num = 1
179         return p_num
180 
181 def set_correct_para(p, P, pos=False):
182     '''
183     Convert the parameters into integers. If input
        is blank then do nothing.
184 
185     Parameters:
186     p -- string input.
187     P -- original value.
188     pos -- If the parameter is positive number.
189     '''
190     if p == '':
191         return P
192     else:
193         return correct_para(p, pos=False)
194 
195 def correct_epi_para(p):
196     '''
197     Convert epidemic parameters into floats.
198 
199     Parameters
200     - p: Epidemic rate, positive decimal less than
        1.
201     '''
202     try:
203         p_num = float(p)
```

Code

```
204         if p_num < 0 or p_num > 1:
205             p_num = 0
206             print('Please␣check␣your␣inputs␣and␣
                    change␣them␣in␣SETTING.')
207         return p_num
208     except ValueError:
209         p_num = 0
210         print('Please␣check␣your␣inputs␣and␣change␣
                them␣in␣SETTING.')
211         return p_num
212
213 def set_correct_epi_para(p, P):
214     '''
215     Convert the parameters into integers. If input
            is blank then do nothing.
216
217     Parameters:
218     p -- string input.
219     P -- original value.
220     pos -- If the parameter is positive number.
221     '''
222     if p == '':
223         return P
224     else:
225         return correct_epi_para(p)
226
227 def set_mode(mode):
228     cmd = ''
229     while cmd != 'y':
230         print('Select␣the␣following␣options:')
```

```
231         print('01: Living in city/ rural [{}]'.
                format(mode01.flag))
232         print('02: Travel from overseas [{}]'.
                format(mode02.flag))
233         print('04: Bounded rationality of vaccine
                [{}]'.format(mode04.flag))
234         print('05: Edit contact network')
235         print('07: Age distribution [{}]'.format(
                mode07.flag))
236         print('08: Gender population [{}]'.format(
                mode08.flag))
237         print('10: Type of vaccine [{}]'.format(
                mode10.flag))
238         print('11: Stop transmissability/ reduce
                severity [{}]'.format(mode11.flag))
239         print('12: Cost of vaccine []')
240         print('13: Accessibility to vaccine []')
241         print('14: Side effects of vaccine []')
242         print('20: Imitation game [{}]'.format(
                mode20.flag))
243         print('21: Local majority rule [{}]'.format
                (mode21.flag))
244         print('22: Stubbon to take vaccine[{}]'.
                format(mode22.flag))
245         print('23: Stubbon to against vaccine[{}]'.
                format(mode23.flag))
246         print('24: Contrary to social groups[{}]'.
                format(mode24.flag))
247         print('31: Medication incorporated [{}]'.
                format(mode31.flag))
```

Code

```
248         print('41: Moral hazard of social
                distancing []')
249         print('42: Moral hazard of treatment []')
250         print('51: Erdos-Renyi topology [{}]'.
                format(mode51.flag))
251         print('52: Preferential attachment [{}]'.
                format(mode52.flag))
252         print('53: Small world topology [{}]'.
                format(mode53.flag))
253         print('54: Lattice network [{}]'.format(
                mode54.flag))
254         print('Input number codes to change the
                options.')
255         mode_input = input('> ')
256         print(mode_input)
257         mode = mode_settings(mode_input, mode)
258         cmd = input('Return to main menu? [y/n] ')
259     return mode
260
261 def mode_settings(cmd, mode=None):
262     cmd = cmd.split(' ')
263     if cmd == ['']:
264         # If empty response, then leave prematurely
                .
265         return mode
266     rv_modes = []
267     if '-dp' in cmd:
268         removal_idx = cmd.index('-dp')
269     print('Adding: ')
270     if '-dp' in cmd:
```

```
271         print(cmd[:removal_idx])
272         print('Removing')
273         print(cmd[removal_idx+1:])
274 
275         rv_modes = cmd[removal_idx+1:]
276         cmd = cmd[:removal_idx]
277     else:
278         print(cmd)
279     if len(cmd) > 0 and '-dp' not in cmd:
280         for i in range(len(cmd)):
281             try:
282                 int(cmd[i])
283             except ValueError:
284                 print('Wrong␣data␣type␣for␣mode,␣
                       please␣check␣your␣inputs.')
285                 continue
286             if int(cmd[i]) == 1:
287                 mode01()
288                 if mode01.flag == 'X':
289                     mode[1] = mode01
290                 else:
291                     mode.pop(1)
292             elif int(cmd[i]) == 2:
293                 mode02.set_proportion()
294                 mode02()
295                 if mode02.flag == 'X':
296                     mode[2] = mode02
297                 else:
298                     mode.pop(2)
299             elif int(cmd[i]) == 4:
```

Code

```
300                 mode04()
301                 if mode04.flag == 'X':
302                     mode[4] = mode04
303                 else:
304                     mode.pop(4)
305             elif int(cmd[i]) == 5:
306                 mode05()
307                 if mode05.flag == 'X':
308                     mode[5] = mode05
309                 else:
310                     mode.pop(5)
311             elif int(cmd[i]) == 7:
312                 mode07()
313                 if mode07.flag == 'X':
314                     mode[7] = mode07
315                 else:
316                     mode.pop(7)
317             elif int(cmd[i]) == 8:
318                 mode08()
319                 if mode08.flag == 'X':
320                     mode[8] = mode08
321                 else:
322                     mode.pop(8)
323             elif int(cmd[i]) == 10:
324                 mode10()
325                 if mode10.flag == 'X':
326                     mode[10] = mode10
327                 else:
328                     mode.pop(10)
329             elif int(cmd[i]) == 11:
```

```
330                 mode11()
331                 if mode11.flag == 'X':
332                     mode[11] = mode11
333                 else:
334                     mode.pop(10)
335             elif int(cmd[i]) == 20:
336                 mode20()
337                 if mode20.flag == 'X':
338                     mode[20] = mode20
339                 else:
340                     mode.pop(21)
341             elif int(cmd[i]) == 21:
342                 mode21()
343                 if mode21.flag == 'X':
344                     mode[21] = mode21
345                 else:
346                     mode.pop(21)
347             elif int(cmd[i]) == 22:
348                 if check_main_mode_opinion(modes
                    ,22) == True:
349                     mode22()
350                 if mode22.flag == 'X':
351                     mode[22] = mode22
352                 else:
353                     mode.pop(22)
354             elif int(cmd[i]) == 23:
355                 if check_main_mode_opinion(modes
                    ,23) == True:
356                     mode23()
357                 if mode23.flag == 'X':
```

Code

```
358                     mode[23] = mode23
359                 else:
360                     mode.pop(23)
361             elif int(cmd[i]) == 24:
362                 if check_main_mode_opinion(modes
                    ,24) == True:
363                     mode24()
364                 if mode24.flag == 'X':
365                     mode[24] = mode24
366                 else:
367                     mode.pop(24)
368             elif int(cmd[i]) == 31:
369                 mode31()
370                 if mode31.flag == 'X':
371                     mode[31] = mode31
372                 else:
373                     mode.pop(31)
374             elif int(cmd[i]) == 51:
375                 if (mode52.flag == 'X'):
376                     print('Mode␣52␣has␣been␣
                        activated.␣Mode␣51␣unable␣to␣
                        start.')
377                     break
378                 mode51()
379                 if mode51.flag == 'X':
380                     mode[51] = mode51
381                 else:
382                     mode.pop(51)
383             elif int(cmd[i]) == 52:
384                 if (mode51.flag == 'X'):
```

```
385                     print('Mode 51 has been
                           activated. Mode 52 unable to
                           start.')
386                     break
387                 mode52()
388                 if mode52.flag == 'X':
389                     mode[52] = mode52
390                 else:
391                     mode.pop(52)
392     # Remove modes (Check if the modes itself
        overwrites basic settings)
393     if len(rv_modes) > 0:
394         for mode_opt in rv_modes:
395             try:
396                 print(mode[int(mode_opt)].flag)
397                 mode[int(mode_opt)].drop_flag()
398             except KeyError:
399                 continue
400             except ValueError:
401                 continue
402     return mode
403
404 def check_main_mode_opinion(modes, code):
405     if 21 not in modes:
406         print(f'Warning: Mode 21 is requried for
              activating mode {code}.')
407         print('Please return to settings to
              activate this mode first. ')
408         return False
409     return True
```

Code

```
410
411 def find_mode(code, mode_master_list):
412     for mode in mode_master_list:
413         if mode.code == code:
414             return mode
415
416 def export(filename):
417     print('Coming␣soon')
418
419 print('␣␣=========================================
    ␣␣\n\n')
420 print('␣␣Agent␣Based␣Modelling:␣COVID-19␣SUEP␣Model
    ␣␣\n\n')
421 print('␣␣=========================================
    ␣␣')
422 print()
423 # Express mode: Call usage information
424 if len(sys.argv) == 2 and (sys.argv[1] == '-help'
    or sys.argv[1] == '-h'):
425     usage()
426     quit()
427
428 if len(sys.argv) == 1:
429     N = input('Number␣of␣people␣(N):␣')
430     N = correct_para(N, pos=True)
431     T = input('Simulation␣time␣(T):␣')
432     T = correct_para(T)
433     alpha = input('Adoption␣rate␣(alpha):␣')
434     alpha = correct_epi_para(alpha)
435     beta = input('Infection␣rate␣(beta):␣')
```

```
436     beta = correct_epi_para(beta)
437     gamma = input('Recovery␣rate␣(gamma):␣')
438     gamma = correct_epi_para(gamma)
439     phi = input('Rate␣to␣resuscept␣(phi):␣')
440     phi = correct_epi_para(phi)
441     delta = input('Removal␣rate␣(delta):␣')
442     delta = correct_epi_para(delta)
443 elif len(sys.argv) > 1:
444     print('Using␣pre-defined␣inputs.␣')
445     try:
446         N = correct_para(sys.argv[1], pos=True)
447         T = correct_para(sys.argv[2])
448         alpha = correct_epi_para(sys.argv[3])
449         beta = correct_epi_para(sys.argv[4])
450         gamma = correct_epi_para(sys.argv[5])
451         phi = correct_epi_para(sys.argv[6])
452         delta = correct_epi_para(sys.argv[7])
453     except:
454         print('Exception␣encountered.␣Leaving␣
              program...')
455         print('Usage:␣python3␣main.py␣[(N)␣(T)␣(
              alpha)␣(beta)␣(gamma)␣(phi)␣(delta)]␣...\
              n[-m␣<modes_config>]␣[-f␣(filename)]␣[run
              ]\n')
456         quit()
457 print()
458
459 '''
460 Set initial variables
461 '''
```

Code

```
462 alpha_V = alpha
463 alpha_T = alpha
464 beta_SS = 0.0
465 beta_II = 0.0
466 beta_RR = 0.01
467 beta_VV = 0.0
468 beta_IR = 0.01
469 beta_SR = 0.01
470 beta_SV = 0.01
471 beta_PI = 0.01
472 beta_IV = 0.01
473 beta_RV = 0.01
474 beta_SI2 = beta
475 beta_II2 = 0.0
476 beta_RI2 = beta_IR
477 beta_VI2 = beta_IV
478 phi_V = phi
479 phi_T = 0.95
480 test_rate = 0.5
481 immune_time = 60
482 group_size = 3
483
484 verbose_mode = False  # Need to put here for
       initiating other objects (nwk and person if
       needed).
485 population = Person.make_population(N)
486 contact_nwk = ContactNwk(population, verbose_mode)
487 info_nwk = Group(population, group_size)
488 filename = ''  # Default file name to export (.csv)
       . Change when use prompt 'export' cmd.
```

```
489
490 mode_master_list = []
491 # All objects should add into mode_master_list
492 mode01 = mode.Mode01(population)
493 mode02 = mode.Mode02(population)
494 mode04 = mode.Mode04(population, alpha)
495 mode05 = mode.Mode05(population, contact_nwk)
496 mode07 = mode.Mode07(population, beta, delta)
497 mode08 = mode.Mode08(population, beta, delta)
498 mode10 = mode.Mode10(population, phi, beta)
499 mode11 = mode.Mode11(population)
500 mode20 = mode.Mode20(population, contact_nwk, beta)
501 mode21 = mode.Mode21(population, info_nwk)
502 mode22 = mode.Mode22(population, info_nwk)
503 mode23 = mode.Mode23(population, info_nwk)
504 mode24 = mode.Mode24(population, info_nwk)
505 mode31 = mode.Mode31(population)
506 mode51 = mode.Mode51(population, contact_nwk)
507 mode52 = mode.Mode52(population, contact_nwk)
508 mode53 = mode.Mode53(population, contact_nwk)
509 mode54 = mode.Mode54(population, contact_nwk)
510
511 mode_master_list = [mode01, mode02, mode04, mode05,
        mode07, mode08,
512 mode10, mode11,
513 mode20, mode21, mode22, mode23, mode24,
514 mode31,
515 mode51, mode52, mode53, mode54]
516
517
```

Code

```
518 modes = {}
519
520 '''
521 Express mode
522
523 Loads the settings prior to the run. Optional
      keyword 'run' to run the simulation automatically
      .
524 '''
525
526 # Check if mode exists
527 for i in range(len(sys.argv)):
528     try:
529         if sys.argv[i] == '-immune_time':
530             immune_time_temp = sys.argv[i+1]
531             immune_time = set_correct_para(
                  immune_time_temp, immune_time, pos=
                  True)
532         elif sys.argv[i] == '-test_rate':
533             test_rate_temp = sys.argv[i+1]
534             test_rate = set_correct_epi_para(
                  test_rate_temp, test_rate)
535         elif sys.argv[i] == '-verbose' or sys.argv[
              i] == '--v':
536             verbose_mode = True
537         elif sys.argv[i] == '-m':
538             for j in range(i+1,len(sys.argv)):
539                 # Skip at other options
540                 if sys.argv[j][:2] == '--':
541                     mode_flag = int(sys.argv[j
```

```
                    ][2:])
542                 print('Loading␣mode:␣{}'.format
                    (mode_flag))
543
544                 # Activate modes with no
                    options needed
545                 if mode_flag == 21:
546                     info_nwk.set_roster()
547                     info_nwk.set_population()
548                     mode21.raise_flag()
549                     if mode21.flag == 'X':
550                         modes[21] = mode21
551                     else:
552                         mode.pop(21)
553                 elif mode_flag == 51:
554                     if 52 in modes:
555                         print('Mode␣52␣has␣been
                            ␣activated.␣Ignore␣
                            mode␣51.␣')
556                         break
557                     elif 53 in modes:
558                         print('Mode␣53␣has␣been
                            ␣activated.␣Ignore␣
                            mode␣52.␣')
559                         break
560                     elif 54 in modes:
561                         print('Mode␣54␣has␣been
                            ␣activated.␣Ignore␣
                            mode␣52.␣')
562                         break
```

Code

```
563                         mode51()
564                         if mode51.flag == 'X':
565                             modes[51] = mode51
566                         else:
567                             modes.pop(51)
568                     # elif mode_flag == 52:
569                     #     if 51 in modes:
570                     #         print('Mode 51 has
                            been activated. Ignore mode
                            52. ')
571                     #         break
572                     #     mode52()
573                     #     if mode52.flag == 'X':
574                     #         modes[52] = mode52
575                     #     else:
576                     #         mode.pop(52)
577
578                     # Loop through config values
579                     for k in range(j+1,len(sys.argv
                        )):
580                         if sys.argv[k][0] == '-'
                            and sys.argv[k][1].
                            isalpha():
581                             break
582                         if sys.argv[k][:2] == '--':
583                             break
584
585                         # Set up individual modes
586                         if mode_flag == 1:
587                             # Placeholder
```

```
588                         if sys.argv[k][:3] == '
                               *b=':
589                             mode01_b_config =
                                   sys.argv[k][3:].
                                   split(',')
590                             b_c = float(
                                   mode01_b_config
                                   [0])
591                             b_r = float(
                                   mode01_b_config
                                   [1])
592                             mode01.set_beta(0,
                                   b_c)
593                             mode01.set_beta(1,
                                   b_r)
594                         elif sys.argv[k][:3] ==
                               '*p=':
595                             mode01_p_config =
                                   sys.argv[k][3:].
                                   split(',')
596                             w_c = float(
                                   mode01_p_config
                                   [0])
597                             w_r = float(
                                   mode01_p_config
                                   [1])
598                             mode01.set_weight(
                                   w_c, w_r)
599
600                         mode01.assign_regions()
```

Code

```
601                             mode01.raise_flag()
602                             if mode01.flag == 'X':
603                                 modes[1] = mode01
604                             else:
605                                 mode.pop(1)
606                         elif mode_flag == 4:
607                             if sys.argv[k][:3] == '
                                   *l=':
608                                 lambda_BR =
                                       population[0].
                                       lambda_BR
609                                 mode04_l_config =
                                       sys.argv[k][3:]
610                                 lambda_BR =
                                       set_correct_para(
                                       mode04_l_config,
                                       lambda_BR, pos=
                                       True)
611                                 mode04.set_lambda(
                                       lambda_BR)
612                             mode04.QRE()
613                             mode04.raise_flag()
614                             if mode04.flag == 'X':
615                                 modes[4] = mode04
616                             else:
617                                 mode.pop(4)
618                         elif mode_flag == 7:
619                             if sys.argv[k][:3] == '
                                   *b=':
620                                 mode07_b_config =
```

```
                                sys.argv[k][3:].
                                split(',')
621                         mode07.beta_age = [
                                float(x) for x in
                                 mode07_b_config]
622                     elif sys.argv[k][:3] ==
                            '*d=':
623                         mode07_d_config =
                                sys.argv[k][3:].
                                split(',')
624                         mode07.delta_age =
                                [float(x) for x
                                in
                                mode07_d_config]
625                     mode07.set_population()
626                     mode07.raise_flag()
627                     if mode07.flag == 'X':
628                         modes[7] = mode07
629                     else:
630                         modes.pop(7)
631                 elif mode_flag == 8:
632                     if sys.argv[k][:3] == '
                           *b=':
633                         mode08_b_config =
                                sys.argv[k][3:].
                                split(',')
634                         mode08.beta_gender
                                = [float(x) for x
                                 in
                                mode08_b_config]
```

Code

```
                elif sys.argv[k][:3] ==
                    '*d=':
                    mode08_d_config =
                       sys.argv[k][3:].
                       split(',')
                    mode08.delta_gender
                        = [float(x) for
                       x in
                       mode08_d_config]
                mode08.set_population()
                mode08.raise_flag()
                if mode08.flag == 'X':
                    modes[8] = mode08
                else:
                    modes.pop(8)
            elif mode_flag == 10:
                if sys.argv[k][:6] == '
                   *mode=':
                    modes[10].type =
                       modes[10].
                       check_input(sys.
                       argv[k][6:])
                if mode10.flag == 'X':
                    modes[10] = mode10
                else:
                    modes.pop(10)
            elif mode_flag == 11:
                if sys.argv[k][:6] == '
                   *mode=':
                    modes[11].type =
```

```
                modes[11].
                check_input(sys.
                argv[k][6:])
654         elif sys.argv[k][:3] ==
            '*b=':
655             mode11_b_config =
                sys.argv[k][3:]
656             mode11.beta_V =
                set_correct_para(
                mode11_b_config,
                mode11.beta_V)
657             mode11.check_beta()
658         elif sys.argv[k][:3] ==
            '*g=':
659             mode11_g_config =
                sys.argv[k][3:]
660             mode11.gamma_V =
                set_correct_para(
                mode11_g_config,
                mode11.gamma_V)
661             mode11.check_gamma
                ()
662         elif sys.argv[k][:3] ==
            '*d=':
663             mode11_d_config =
                sys.argv[k][3:]
664             mode11.delta_V =
                set_correct_para(
                mode11_d_config,
                mode11.delta_V)
```

Code

```
665                                 mode11.check_delta
                                        ()
666                             if mode11.flag == 'X':
667                                 modes[11] = mode11
668                             else:
669                                 modes.pop(11)
670                         elif mode_flag == 21:
671                             if sys.argv[k][:3] == '
                                    *+=':
672                                 mode21_pro_config =
                                        sys.argv[k][3:]
673                                 mode21.set_pro(
                                        mode21_pro_config
                                        )
674                             elif sys.argv[k][:3] ==
                                    '*-=':
675                                 mode21_ag_config =
                                        sys.argv[k][3:]
676                                 mode21.set_ag(
                                        mode21_ag_config)
677                             if mode21.propro !=
                                    None and mode21.agpro
                                    != None:
678                                 mode21.set_opinion
                                        ()
679                                 mode21.
                                        set_personality()
680                                 mode21.raise_flag()
681                             if mode21.flag == 'X':
682                                 modes[21] = mode21
```

```
683                             else:
684                                 modes.pop(21)
685                         elif mode_flag == 22:
686                             if sys.argv[k][:3] == '
                                   *p=':
687                                 mode22_pro_config =
                                        sys.argv[k][3:]
688                                 mode22.
                                       assign_personality
                                       (
                                       mode22_pro_config
                                       )
689                                 mode22.raise_flag()
690                             if mode22.flag == 'X':
691                                 modes[22] = mode22
692                             else:
693                                 modes.pop(22)
694                         elif mode_flag == 23:
695                             if sys.argv[k][:3] == '
                                   *p=':
696                                 mode23_pro_config =
                                        sys.argv[k][3:]
697                                 mode23.
                                       assign_personality
                                       (
                                       mode23_pro_config
                                       )
698                                 mode23.raise_flag()
699
700                             if mode23.flag == 'X':
```

Code

```
701                                 modes[23] = mode23
702                             else:
703                                 modes.pop(23)
704                         elif mode_flag == 24:
705                             if sys.argv[k][:3] == '
                                    *p=':
706                                 mode24_pro_config =
                                        sys.argv[k][3:]
707                                 mode24.
                                        assign_personality
                                        (
                                        mode24_pro_config
                                        )
708                                 mode24.raise_flag()
709
710                             if mode24.flag == 'X':
711                                 modes[24] = mode24
712                             else:
713                                 modes.pop(24)
714                         elif mode_flag == 52:
715                             if 51 in modes:
716                                 print('Mode␣51␣has␣
                                        been␣activated.␣
                                        Ignore␣mode␣52.␣'
                                        )
717                                 break
718                             elif 53 in modes:
719                                 print('Mode␣53␣has␣
                                        been␣activated.␣
                                        Ignore␣mode␣52.␣'
```

```
                                    )
720                                 break
721                             elif 54 in modes:
722                                 print('Mode␣54␣has␣
                                        been␣activated.␣
                                        Ignore␣mode␣52.␣'
                                        )
723                                 break
724
725                             if sys.argv[k][:3] == '
                                    *m=':
726                                 mode52_m_config =
                                        int(sys.argv[k
                                        ][3:])
727                                 mode52.set_m(
                                        mode52_m_config)
728                             elif sys.argv[k][:3] ==
                                    '*p=':
729                                 contact_nwk.
                                        update_rule = '
                                        XBS'
730                                 mode52_p_config =
                                        float(sys.argv[k
                                        ][3:])
731                                 mode52.set_pupdate(
                                        mode52_p_config)
732                             elif sys.argv[k][:3] ==
                                    '*a=':
733                                 contact_nwk.
                                        update_rule = '
```

Code

```
                                    XBS'
                                mode52_assort = int
                                    (sys.argv[k][3:])
                                if mode52_assort ==
                                     1:
                                     contact_nwk.
                                         assort = True
                                elif mode52_assort
                                    == 0:
                                     contact_nwk.
                                         assort =
                                         False
                            elif sys.argv[k][:3] ==
                                '*l=':
                                contact_nwk.
                                    update_rule = '
                                    random'
                                mode52_l_config = [
                                    int(x) for x in
                                    sys.argv[k][3:].
                                    split(',')]
                                mode52.set_l0(
                                    mode52_l_config
                                    [0])
                                mode52.set_l1(
                                    mode52_l_config
                                    [1])
                            mode52.set_network()
                            mode52.raise_flag()
                            if mode52.flag == 'X':
```

```
747                                 modes[52] = mode52
748                             else:
749                                 mode.pop(52)
750
751
752                         # elif mode_flag == 999:
753                         #     # There are 3 args with last one characters
754                         #     seven_config(*[int(data) if data.isnumeric() else data for data in config])
755                         continue
756                     if sys.argv[j][0] == '-' and sys.argv[j][1].isalpha():
757                         break
758                     if sys.argv[j] == 'run':
759                         break
760                     # print(mode_flag, '*'+str(argv[j]))
761         except ValueError:
762             print('Invalid input. Check your arguments. ')
763             continue
764         except IndexError:
765             break
766
767         # Check file name to export
768         try:
769             if sys.argv[i] == '-f':
```

Code

```
770             if sys.argv[i+1] == 'run':
771                 raise ValueError
772             filename = sys.argv[i+1]
773     except ValueError:
774         print('No file name provided. Please check
               your inputs.')
775         continue
776     except IndexError:
777         break
778
779 if sys.argv[-1] == 'run':
780     print('===== Simulation Running =====')
781     current_run = Simulation(population, T,
           population, contact_nwk, info_nwk, alpha,
           beta, gamma, phi, delta, filename, alpha_V,
           alpha_T, beta_SS, beta_II, beta_RR, beta_VV,
           beta_IR, beta_SR, beta_SV, beta_PI, beta_IV,
           beta_RV, beta_SI2, beta_II2, beta_RI2,
           beta_VI2, test_rate, immune_time,
           verbose_mode)
782     # Load modes
783     current_run.load_modes(modes)
784     if len(modes) > 0:
785         print('\nMode objects loaded.\n')
786     # Run
787     current_run()
788     print('=====  Simulation Ended  =====')
789     print('\nSee you!')
790     quit()
791
```

```
'''
Normal mode
'''

while True:
    cmd = input('>>> ').lower()
    if cmd == 'setting':
        N, T, alpha, beta, gamma, phi, delta,
           alpha_V, alpha_T, phi_V, phi_T, test_rate
           , immune_time, group_size, verbose_mode =
            setting(N, T, alpha, beta, gamma, phi,
           delta, alpha_V, alpha_T, phi_V, phi_T,
           test_rate, immune_time, group_size,
           verbose_mode)
        population = Person.make_population(N)
    elif cmd == 'other setting':
        print('Leave blank if not changing the
           value(s).')
        N, T, alpha, beta, gamma, phi, delta,
           alpha_V, alpha_T, phi_V, phi_T, test_rate
           , immune_time, group_size, verbose_mode =
            setting_other(N, T, alpha, beta, gamma,
           phi, delta, alpha_V, alpha_T, phi_V,
           phi_T, test_rate, immune_time, group_size
           , verbose_mode)
    elif cmd == 'summary':
        summary()
    elif cmd == 'look':
        show_nwk()
    elif cmd == 'help':
```

Code

```
809         help()
810     elif cmd == 'start' or cmd == 'run':
811         print('=====␣Simulation␣Running␣=====')
812         current_run = Simulation(population, T,
                population, contact_nwk, info_nwk, alpha,
                 beta, gamma, phi, delta, filename,
                alpha_V, alpha_T, beta_SS, beta_II,
                beta_RR, beta_VV, beta_IR, beta_SR,
                beta_SV, beta_PI, beta_IV, beta_RV,
                beta_SI2, beta_II2, beta_RI2, beta_VI2,
                test_rate, immune_time, verbose_mode)
813         # Load modes
814         current_run.load_modes(modes)
815         if len(modes) > 0:
816             print('\nMode␣objects␣loaded.\n')
817         # Run
818         current_run()
819         print('=====␣␣Simulation␣Ended␣␣=====')
820     elif cmd == 'mode':
821         modes = set_mode(modes)
822     elif cmd == 'export':
823         filename = input('File␣name:␣')
824     elif cmd == 'thank␣you':
825         print('␣␣
                ==========================================
                ␣␣\n\n')
826         print('␣␣Agent␣Based␣Modelling:␣COVID-19␣
                SEIP␣Model␣␣\n\n')
827         print('␣␣
                ==========================================
```

```
            ␣␣')
828         time.sleep(3)
829         print('\n\nAuthor:␣Shing␣Hin␣(John)␣Yeung\n
                ')
830         time.sleep(1)
831         print('This␣software␣code␣comes␣from␣
                Masters␣in␣Complex␣Systems␣a␣Capstone␣
                Project.␣\n')
832         time.sleep(1)
833         print('It␣aims␣for␣modelling␣human␣choice␣
                upon␣vaccine␣adoption␣and␣therefore␣
                predict␣the␣epidemic.␣\n\n')
834         time.sleep(1)
835         print('The␣author␣would␣like␣to␣thank␣you␣
                to␣many␣people␣who␣conrtibuted␣to␣this␣
                project.␣\n')
836         time.sleep(1)
837         print('Dr␣Shailendra␣Sawleshwarkar')
838         print('A/␣Prof␣Iryna␣Zablotska-Manos')
839         print('Dr␣Samit␣Bhattacharyya')
840         time.sleep(3)
841         print('\n\nPrimary␣supervisor')
842         print('Dr␣Mahendrarajah␣Piraveenan\n')
843         time.sleep(3)
844         print('This␣study␣dedicates␣to␣the␣humanity
                ␣that␣strives␣in␣the␣COVID-19␣pandemic.␣\
                n\n\n')
845         time.sleep(1)
846         print('====␣Thank␣you␣====')
847     elif cmd == 'quit' or cmd == 'q':
```

Code

```
848             print('See you!')
849             quit()
850         else:
851             print('Invalid input. Please check your
                    command again.')
852             cmd = input('Commands [y/n]')
853             if cmd == 'y':
854                 usage()
855         print('')
```

Listing 6.2: simulation.py

```
1  from person import Person
2  from epidemic import Epidemic
3  from contact import ContactNwk
4  import write
5
6  import random
7
8  class Simulation:
9      def __init__(self, N, T, people, contact_nwk,
           info_nwk, alpha, beta, gamma, phi, delta,
           filename, alpha_V, alpha_T, beta_SS, beta_II,
            beta_RR, beta_VV, beta_IR, beta_SR, beta_SV,
            beta_PI, beta_IV, beta_RV, beta_SI2,
           beta_II2, beta_RI2, beta_VI2, tau,
           immune_time, verbose_mode, groups_of=3):
10             self.N = N
11             self.groups_of = groups_of
12             self.people = people   # List of people
                   objects
13             self.contact_nwk = contact_nwk
```

```
14          self.info_nwk = info_nwk
15          self.groups = None
16          self.T = T
17
18          # Adoption rate
19          self.alpha = alpha
20          self.alpha_V = alpha_V
21          self.alpha_T = alpha_T
22
23          # Infection rate
24          self.beta = beta
25          self.beta_SS = beta_SS
26          self.beta_II = beta_II
27          self.beta_RR = beta_RR
28          self.beta_VV = beta_VV
29          self.beta_IR = beta_IR
30          self.beta_SR = beta_SR
31          self.beta_SV = beta_SV
32          self.beta_PI = beta_PI
33          self.beta_IV = beta_IV
34          self.beta_RV = beta_RV
35          self.beta_SI2 = beta_SI2
36          self.beta_II2 = beta_II2
37          self.beta_RI2 = beta_RI2
38          self.beta_VI2 = beta_VI2
39
40          # Recovery rate
41          self.gamma = gamma
42          self.immune_time = immune_time
43
```

Code

```
44          # Wear off rate
45          self.phi = phi
46
47          # Testing rate
48          self.test_rate = tau
49
50          # Removal rate
51          self.delta = delta
52
53          # Auxillary parameters
54          self.verbose_mode = verbose_mode
55          self.filename = filename
56          self.modes = {}
57
58      def load_modes(self,modes):
59          '''Load mode objects into epidemic class,
               as defined in the main code.
60
61          parameters
62          ----------
63
64          modes - dict:
65              Keys are integer mode code with the
                   corresponding mode objects
66          '''
67          self.modes = modes
68
69      def __call__(self, modes=None, start=True):
70          FILENAME_STATES = ''
71          epidemic = Epidemic(self.alpha, self.beta,
```

```
            self.gamma, self.phi, self.delta, self.
            people, self.test_rate, self.immune_time,
             self.contact_nwk, self.verbose_mode,
            start)
72          epidemic.set_other_alpha_param(self.alpha_V
            , self.alpha_T)
73          epidemic.set_other_beta_param(self.beta_SS,
             self.beta_II, self.beta_RR, self.beta_VV
            , self.beta_IR, self.beta_SR, self.
            beta_SV, self.beta_PI, self.beta_IV, self
            .beta_RV, self.beta_SI2, self.beta_II2,
            self.beta_RI2, self.beta_VI2)
74          epidemic.load_modes(self.modes)
75          print('beta = {}, alpha = {}, gamma = {}, 
            phi = {}, lambda = {}'.format(epidemic.
            infection, epidemic.vaccinated, epidemic.
            recover, epidemic.resus, epidemic.
            test_rate))
76          print('========== t = 0 ==========\n')
77          print('N = {}'.format(len(self.people)))
78          print('S = {}, I = {}, V = {}, R = {}'.
            format(epidemic.S, epidemic.I, epidemic.V
            , epidemic.R))
79          epidemic.get_states()
80          if self.filename != '':
81              write.WriteStates(epidemic, self.
                filename)
82          for t in range(self.T):
83              print('========== t = {} ==========\
                n'.format(t+1))
```

Code

```
84              print('N␣=␣{}'.format(len(self.people))
                    )
85              print('S␣=␣{},␣I␣=␣{},␣V␣=␣{},␣R␣=␣{}'.
                    format(epidemic.S, epidemic.I,
                    epidemic.V, epidemic.R))
86              # Info network update
87              if 21 in self.modes:
88                  if any(i in self.modes for i in
                        [22, 23]):
89                      self.info_nwk.
                            inflexible_prework()
90                  if self.verbose_mode == True:
91                      print('Opinion␣(before)')
92                      for group_no, group in self.
                            info_nwk.roster.items():
93                          print(f'{group_no}:', [x.
                                opinion for x in group])
94                  self.info_nwk.update(self.
                        verbose_mode)
95                  if any(i in self.modes for i in
                        [22, 23]):
96                      self.info_nwk.inflexible()
97                  if 24 in self.modes:
98                      self.info_nwk.balance(self.
                            verbose_mode)
99                  if self.verbose_mode == True:
100                     print('Opinion␣(after)')
101                     for group_no, group in self.
                            info_nwk.roster.items():
102                         print(f'{group_no}:', [x.
```

```
                                opinion for x in group])
103                 if any(i in self.modes for i in
                        [22, 23, 24]) and self.filename
                        != '':
104                     write.WriteOpinionPersonality(
                            self, self.filename)
105                 elif self.filename != '':
106                     write.WriteOpinion(self, self.
                            filename)
107                 # Permutate members into groups
108                 self.info_nwk.update_group(self.
                        verbose_mode)
109             # Epidemic network update
110             epidemic.next(self.filename)
111
112         print('\n==========␣Result␣==========\n'
                )
113         print('There␣are␣{}␣people␣infected.'.
                format(epidemic.I))
114         print('There␣are␣{}␣people␣vaccinated.'.
                format(epidemic.V))
115         print()
116         if self.filename != '':
117             print('Data␣stored␣in␣\'{}.csv\''.
                    format(self.filename))
118             write.WriteCompartmentHistory(self,
                    self.filename)
119             print('Compartment␣history␣exported␣in␣
                    \'{}-compartment.csv\''.format(self.
                    filename))
```

Code

```
120             write.WriteTestingHistory(self, self.
                    filename)
121             print('COVID-19␣testing␣records␣
                    exported␣in␣\'{}-testing.csv\''.
                    format(self.filename))
122             if any(i in self.modes for i in [22,
                    23, 24]):
123                 print('Population␣personality␣and␣
                        information␣network␣details␣
                        exported␣in␣\'{}-opinion.csv\''.
                        format(self.filename))
124             elif 21 in self.modes:
125                 print('Information␣network␣details␣
                        exported␣in␣\'{}-opinion.csv\''.
                        format(self.filename))
126             if any(i in self.modes for i in [51,
                    52, 53, 54]) and self.contact_nwk.
                    update_rule != None:
127                 print('Average␣degree␣history␣
                        exported␣in␣\'{}-nwk-deg.csv\',␣
                        \'{}-nwk-deg_I.csv\'␣and␣\'{}-nwk
                        -deg_S.csv\''.format(self.
                        filename,self.filename,self.
                        filename))
128                 print('Assortativity␣history␣
                        exported␣in␣\'{}-assort-deg.csv\'
                        '.format(self.filename))
129             write.WriteSummary(self, self.filename)
130             print('Summary␣exported␣in␣\'{}-summary
                    .txt\''.format(self.filename))
```

```
131             print()
132
133         # Return any data
```

Listing 6.3: epidemic.py

```
1  import random
2  import numpy as np
3
4  from contact import ContactNwk
5  import person
6  import write
7
8  class Epidemic:
9
10     def __init__ (self, vaccinated, infection,
           recover, resus, remove, people, test_rate,
           immune_time, contact_nwk, verbose_mode, start
           =True):
11         '''Initial elements
12
13         Attributes
14         ----------
15
16         epidemic - int
17             Flag epidemic starts or ends.
18
19         people - People
20             Agents for simulation
21         '''
22         self.epidemic = 0   # Whether an epidemic
              occured or not.
```

Code

```
        self.people = people
        self.contact_nwk = contact_nwk
        self.mode = {}  # Dict of modes loaded.
           Values are mode objects

        try:
            if vaccinated >= 0 and vaccinated <= 1:
                self.vaccinated = vaccinated   #
                   Probability to get vaccinated
            else:
                raise ValueError
            if infection >= 0 and infection <= 1:
                self.infection = infection
            else:
                raise ValueError
            if recover >= 0 and recover <= 1:
                self.recover = recover  # Recovery
                   rate
            else:
                raise ValueError
            if remove >= 0 and remove <= 1:
                self.remove = remove  # Recovery
                   rate
            else:
                raise ValueError
            if resus >= 0 and resus <= 1:
                self.resus = resus
            else:
                raise ValueError
            if test_rate >= 0 and test_rate <= 1:
```

```
49                 self.test_rate = test_rate
50             else:
51                 raise ValueError
52
53             self.immune_time = immune_time
54
55             # Customised lifestyle rate. Call
                   set_other_alpha_param() to set values
                   .
56             self.alpha_V = self.vaccinated
57             self.alpha_T = self.vaccinated
58
59             # Customised infection rate
60             # self.infection_SS = 0.0
61             # self.infection_II = 0.0
62             # self.infection_II2 = 0.0
63             # self.infection_RR = 0.01
64             # self.infection_VV = 0.0
65             # self.infection_IR = 0.01
66             # self.infection_SR = 0.01
67             # self.infection_SV = 0.01
68             # self.infection_PI = 0.01
69             # self.infection_IV = 0.01
70             # self.infection_RV = 0.01
71             # self.infection_SI2 = self.infection
72             # self.infection_RI2 = 0.01
73             # self.infection_VI2 = 0.01
74             # self.infection_condom = 0.01
75             # self.check_beta()
76
```

Code

```
77              # Auxillary parameter
78              self.verbose_mode = verbose_mode
79
80
81              '''Compartment statics
82
83              Number of agents within a compartment.
84
85              Attributes
86              ----------
87              S: int
88                  Number of people not infected (
                        susceptible).
89              U: int
90                  Number of people with COVID-19
                        symptoms observed.
91              E: int
92                  Number of people with COVID-19
                        symptoms observed, assumed
                        quarantined.
93              I: int
94                  Sum of people in E and U. Number of
                         infected agents.
95              V: int
96                  Number of people taken PrEP
97              R: int
98                  Number of people removed.
99              Pro: int
100                 Number of agents willing to accept
                        vaccine
```

```
101             Ag: int
102                 Number of agents against of taking
                        vaccine
103             dS: int
104                 Difference of suceptible
                        compartment at different times
105             dI: int
106                 Difference of infected compartment
                        at different times
107             dV: int
108                 Difference of vaccinated (PrEP)
                        compartment at different times
109             '''
110             self.S = len(self.people)
111             self.U = 0
112             self.E = 0
113             self.I = self.U + self.E
114             self.V = 0
115             self.R = 0
116             self.Pro = 0
117             self.Ag = 0
118             self.dS = -self.vaccinated*self.S*self.
                    Pro - (1-self.vaccinated)*self.
                    infection*self.S*self.I*self.Pro -
                    self.infection*self.S*self.I*self.Ag
                    + self.recover*self.I + self.resus*
                    self.V
119             self.dI = (1-self.vaccinated)*self.
                    infection*self.S*self.I*self.Pro +
                    self.infection*self.S*self.I*self.Ag
```

Code

```
                - self.recover*self.I
120             self.dV = self.vaccinated*self.S*self.
                    Pro - self.resus*self.V
121
122             if start == True:
123                 self.set_epidemic(1)
124                 # Write longitudinal social network
                        data
125                 if (51 in self.mode or 52 in self.
                        mode or 53 in self.mode or 54 in
                        self.mode) and self.contact_nwk.
                        update_rule != None:
126                     if filename != '':
127                         write.WriteNetworkAvgDegree
                                (self.contact_nwk.
                                nwk_graph, filename)
128                         write.
                                WriteNetworkAvgDegree_I(
                                self.contact_nwk.
                                nwk_graph, filename)
129                         write.
                                WriteNetworkAvgDegree_S(
                                self.contact_nwk.
                                nwk_graph, filename)
130                         write.
                                WriteNetworkAssortativity
                                (self.contact_nwk.
                                nwk_graph, filename)
131
132
```

```
133         except ValueError:
134             print('Check your parameters if they
                are probabilities.')
135
136     def set_other_alpha_param(self, alpha_V,
            alpha_T):
137         self.alpha_V = alpha_V
138         self.alpha_T = alpha_T
139
140     def set_other_beta_param(self, beta_SS, beta_II
            , beta_RR, beta_VV, beta_IR, beta_SR, beta_SV
            , beta_PI, beta_IV, beta_RV, beta_SI2,
            beta_II2, beta_RI2, beta_VI2):
141         self.infection_SS = beta_SS
142         self.infection_II = beta_II
143         self.infection_RR = beta_RR
144         self.infection_VV = beta_VV
145         self.infection_IR = beta_IR
146         self.infection_SR = beta_SR
147         self.infection_SV = beta_SV
148         self.infection_PI = beta_PI
149         self.infection_IV = beta_IV
150         self.infection_RV = beta_RV
151
152
153
154     def get_states(self):
155         '''
156             Get number of people who are in S, I or
                V state.
```

Code

```
157         '''
158         self.S = 0
159         self.I = 0
160         self.U = 0
161         self.E = 0
162         self.V = 0
163         self.R = 0
164         for i in range(len(self.people)):
165             if self.people[i].vaccinated == 1:
166                 self.V += 1
167                 continue
168             elif self.people[i].suceptible == 1 and
                    self.people[i].removed == 0:
169                 self.U += 1
170                 continue
171             elif self.people[i].exposed == 1 and
                   self.people[i].removed == 0:
172                 self.E += 1
173                 continue
174             elif self.people[i].removed == 1:
175                 self.R += 1
176                 continue
177             self.S += 1
178         self.I = self.U + self.E
179         return self.S, self.I, self.U, self.E, self
               .V, self.R
180
181     def write_history(self):
182         '''Write compartment history of everyone.
               '''
```

```
183         for i in range(len(self.people)):
184             if self.people[i].vaccinated == 1:
185                 self.people[i].compartment_history.
                       append('V')
186                 continue
187             elif (self.people[i].suceptible == 0
                   and self.people[i].vaccinated == 0):
188                 self.people[i].compartment_history.
                       append('S')
189                 continue
190             elif (self.people[i].suceptible == 1
                   and self.people[i].exposed == 0):
191                 self.people[i].compartment_history.
                       append('E')
192                 continue
193             elif (self.people[i].suceptible == 1
                   and self.people[i].exposed == 1):
194                 self.people[i].compartment_history.
                       append('I')
195                 continue
196             elif self.people[i].removed == 1:
197                 self.people[i].compartment_history.
                       append('R')
198                 continue
199
200     def set_epidemic(self, mode):
201         '''
202         Set either the environment to be disease-
               free or not.
203         '''
```

Code

```
204         try:
205             if mode > 1 or mode < 0:
206                 raise ValueError
207         except ValueError:
208             print('Mode must be either 1 or 0')
209             pass
210         if mode == 1:
211             self.epidemic = 1
212             Epidemic.start_epidemic(self)
213         else:
214             self.epidemic = 0
215             Epidemic.kill_epidemic(self)
216 
217     def start_epidemic(self, initial_infection=4):
218         '''
219         Start an epidemic
220         '''
221         if len(self.people) < initial_infection:
222             initial_infection = len(self.people)
223         # Pick first 4 people/ random number of
              people (1-5) infected initially
224         for i in range(initial_infection):
225             self.people[i].suceptible = 1
226 
227     def kill_epidemic(self):
228         for i in range(len(self.people)):
229             self.people[i].suceptible = 0
230 
231     def load_modes(self, modes):
232         self.mode.update(modes)
```

```
233
234     def set_pro_ag(self):
235         '''
236         Return the proportion of people who pro or
               against vaccination.
237         '''
238         pro = 0
239         ag = 0
240         for i in range(len(self.people)):
241             if self.people[i].opinion == 1:
242                 pro += 1
243             else:
244                 ag += 1
245         self.Pro = pro/(len(self.people))
246         self.Ag = ag/(len(self.people))
247
248         # Resume temp variables
249         ag = 0
250         pro = 0
251
252     def vaccinate(self):
253         for i in range(len(self.people)):
254             if 4 in self.mode:
255                 if self.verbose_mode == True:
256                     print(self.mode[4].P_Alpha[i])
257                 seed = random.randint(0,10000)
                       /10000
258                 if seed < self.mode[4].P_Alpha[i]
                       and self.people[i].vaccinated ==
                       0:
```

Code

```
259                     if self.verbose_mode == True:
260                         print(f'{self.people[i].id}
                                ␣has␣decided␣to␣take␣
                                vaccine.␣')
261                     self.people[i].vaccinated = 1
262                 continue
263             if 20 in self.mode:
264                 theta = self.mode[20].
                    set_perceived_infection(self.I/
                    len(self.people))
265                 continue
266             if 21 in self.mode:
267                 person = self.people[i]
268                 seed = random.randint(0,10000)
                    /10000
269                 if person.opinion == 1 and seed <
                    self.alpha_V:
270                     if self.verbose_mode == True:
271                         print(f'***,␣{seed}␣<=␣{
                                self.alpha_V}')
272                     person.vaccinated = 1
273                 continue
274             if self.people[i].suceptible == 1:
275                 continue
276
277     def removed(self):
278         '''
279         A person is removed from population.
280         '''
281         delta_pp = np.ones(len(self.people))
```

```
282         for i in range(len(self.people)):
283             if any(i in self.mode for i in [7, 8]):
284                 # Fetch all parameters:
285                 if 7 in self.mode:
286                     delta_pp[i] = np.multiply(self.
                        mode[7].delta_age[int(self.
                        people[i].age//10)], delta_pp
                        [i])
287                 if 8 in self.mode:
288                     delta_pp[i] = np.multiply(self.
                        mode[8].delta_gender[self.
                        people[i].gender], delta_pp[i
                        ])
289                 if 11 in self.mode:
290                     if self.people[i].vaccinated ==
                         1:
291                         delta_pp[i] = np.multiply(
                            self.mode[11].delta_V,
                            delta_pp[i])
292                 if self.verbose_mode == True:
293                     print(f'Delta␣for␣{self.people[
                        i].id}␣is␣{beta_pp[i]}.␣')
294
295
296         for i in range(len(self.people)):
297             if self.people[i].suceptible != 1:
298                 continue
299             seed = random.randint(0,1000)/1000
300             if any(i in self.mode for i in [7, 8]):
301                 if seed < delta_pp[i]:
```

Code

```
302                     self.people[i].removed = 1
303
304             if seed < self.remove:
305                 self.people[i].removed = 1
306
307     def infect(self):
308         '''
309         Mechanism of infection.
310         '''
311
312         # Network contact controlled by Epidemic.
               social_contact()
313         if (51 in self.mode) or (52 in self.mode)
               or (53 in self.mode) or (54 in self.mode)
               :
314             if self.verbose_mode == True:
315                 print('Social␣contact␣applies␣to␣
                       infection.␣')
316             self.social_contact()
317             return
318         # Creating customised infection parameter
               for each person.
319         beta_pp = np.ones(len(self.people))
320         for i in range(len(self.people)):
321             if any(i in self.mode for i in [1, 7,
                   8, 11]):
322                 # Fetch all parameters:
323                 if 1 in self.mode:
324                     beta_pp[i] = np.multiply(self.
                           mode[1].betas[self.people[i].
```

```
                        location], beta_pp[i])
325                 if 7 in self.mode:
326                     beta_pp[i] = np.multiply(self.
                           mode[7].beta_age[int(self.
                           people[i].age//10)], beta_pp[
                           i])
327                 if 8 in self.mode:
328                     beta_pp[i] = np.multiply(self.
                           mode[8].beta_gender[self.
                           people[i].gender], beta_pp[i
                           ])
329                 if 11 in self.mode:
330                     if self.people[i].vaccinated ==
                            1:
331                         beta_pp[i] = np.multiply(
                               self.mode[11].beta_V,
                               beta_pp[i])
332
333         for i in range(len(self.people)):
334             '''
335             Infect (or not)
336             '''
337             if self.people[i].suceptible == 1 or
                   self.people[i].removed == 1 :
338                 if self.verbose_mode == True:
339                     print(f'{self.people[i].id} 
                           will not be infected. ')
340                 continue  # Skip
341
342             if 11 not in self.mode and self.people[
```

Code

```
                i].vaccinated == 1:
343                 if self.verbose_mode == True:
344                     print(f'{self.people[i].id} is
                            vaccinated and will not be
                            infected. ')
345                 continue
346
347             seed = random.randint(0,1000)/1000
348             '''
349             Customised infection from modes (excl.
                network contact)
350             '''
351             if any(i in self.mode for i in [1, 7,
                8]):
352                 if self.verbose_mode == True:
353                     print(f'Beta for {self.people[i
                            ].id} is {beta_pp[i]}. ')
354                 if seed < beta_pp[i]:
355                     self.people[i].suceptible = 1
356                 continue
357             # Normal infection event
358             if seed < self.infection:
359                 self.people[i].suceptible = 1
360
361     def social_contact(self):
362         '''
363         Simulate social contacts.
364         '''
365         for edge in self.contact_nwk.network:
366             # This is edge of People objects.
```

```
367             # Conditions where disease will not
                    spread (SS, VV, RR)
368             if self.verbose_mode == True:
369                 print('{}/␣{}'.format(self.
                        contact_nwk.network.index(edge),
                        len(self.contact_nwk.network)))
370             if edge[0].suceptible == 0 and edge[1].
                    suceptible == 0:
371                 if self.verbose_mode == True:
372                     print(f'Both␣ends␣are␣not␣
                            infected.␣Skip␣({edge[0].id},
                            ␣{edge[1].id}).␣')
373                 continue
374             # The following will not apply if mode
                    11 has been activated, skips this
                    condition.
375             if (edge[0].vaccinated == 1 and edge
                    [1].vaccinated == 1) and 11 not in
                    self.mode:
376                 if self.verbose_mode == True:
377                     print(f'Both␣ends␣are␣
                            vaccinated.␣Skip␣({edge[0].id
                            },␣{edge[1].id}).␣')
378                 continue
379             elif edge[0].vaccinated == 1 and 11 not
                    in self.mode:
380                 if self.verbose_mode == True:
381                     print(f'{edge[0].id}␣are␣
                            vaccinated.␣Skip␣({edge[0].id
                            },␣{edge[1].id}).␣')
```

Code

```
382                 continue
383             elif edge[1].vaccinated == 1 and 11 not
                    in self.mode:
384                 if self.verbose_mode == True:
385                     print(f'{edge[1].id} are
                        vaccinated. Skip ({edge[0].id
                        }, {edge[1].id}). ')
386                 continue
387             if edge[0].removed == 1 or edge[1].
                removed == 1:
388                 if self.verbose_mode == True:
389                     print(f'One of the contacts are
                        removed. Skip ({edge[0].id},
                        {edge[1].id}). ')
390                 continue
391             if edge[0].exposed == 1 or edge[1].
                exposed == 1:
392                 if self.verbose_mode == True:
393                     print(f'One of the contacts are
                        quarantined. Skip ({edge[0].
                        id}, {edge[1].id}). ')
394                 continue
395             if self.verbose_mode == True:
396                 print(edge[0].id, edge[1].id)
397                 print('  ', edge[0].suceptible,
                    edge[1].suceptible)
398
399             # Infect (or not)
400             seed = random.randint(0,100000)/100000
401             if seed < self.infection:
```

```
402                 edge[0].suceptible = 1
403                 edge[1].suceptible = 1
404                 if self.verbose_mode == True:
405                     print(f'{edge[0].id}-{edge[1].
                        id} pair is infected.')
406
407     def infection_clock(self, i):
408         if self.people[i].infection_clock > 14:
409             self.people[i].exposed = 1
410
411     def infected(self):
412         '''
413         Once a person was infected for 14 days,
               their symptoms are exposed.
414
415         If the person is tested, we may put them
               into E compartment.
416         '''
417         for i in range(len(self.people)):
418             if self.people[i].suceptible == 1 and
                   self.people[i].removed == 0:
419                 self.people[i].infection_clock += 1
420             else:
421                 self.people[i].infection_clock = 0
422
423             self.infection_clock(i)
424
425     def recovery(self):
426         for i in range(len(self.people)):
427             seed = random.randint(0,100000)/100000
```

Code

```
428             if 11 in self.mode:
429                 if seed < self.mode[11].gamma_V:
430                     self.people[i].suceptible = 0
431                     self.people[i].exposed = 0
432             if seed < self.recover:
433                 self.people[i].suceptible = 0
434                 self.people[i].exposed = 0
435 
436     def immune(self):
437         '''
438         Assume there is a period of immunity since
               recovery.
439         '''
440         if self.immune_time == 0:
441             return
442         for i in range(len(self.people)):
443             recent = self.people[i].
                   compartment_history[-self.immune_time
                   :]
444             for j in range(len(recent)-1):
445                 if len(recent) < 2:
446                     continue
447                 if recent[j] == 'I' and recent[j+1]
                       == 'S':
448                     self.people[i].suceptible = 0
449                     self.people[i].exposed = 0
450                     continue
451 
452     def wear_off(self):
453         '''
```

```
454         Vaccine may wear off.
455         '''
456         if 10 in self.mode:
457             if self.mode[10].type == 1:
458                 return # Patients will not have
                        their vaccine wear-off.
459             elif self.mode[10].type == 2:
460                 for i in range(len(self.people)):
461                     # See people are unlikely to
                            leave V compartment.
462                     recent = self.people[i].
                            compartment_history[-366//4:]
463                     for j in range(len(recent)-1):
464                         if len(recent) < 2:
465                             continue
466                         if recent[j] == 'S' and
                                recent[j+1] == 'V':
467                             # Maintain V state and
                                    iterate to next
                                    person
468                             self.people[i].
                                    vaccinated = 1
469                             continue
470                     # Else
471                     seed = random.randint(0,100000)
                            /100000
472                     if self.people[i].vaccinated ==
                            1 and seed <= self.resus:
473                         self.people[i].vaccinated =
                                0
```

Code

```
474         for i in range(len(self.people)):
475             seed = random.randint(0,100000)/100000
476             if self.people[i].vaccinated == 1 and
                    seed <= self.resus:
477                 self.people[i].vaccinated = 0
478
479     def testing(self):
480         '''
481         COVID-19 testing and people who are in the
                E compartment will become I.
482         '''
483         for i in range(len(self.people)):
484             seed = random.randint(0,100000)/100000
485             if seed < self.test_rate:
486                 if self.people[i].suceptible == 1:
487                     self.people[i].exposed = 1
488                 self.people[i].test_history.append
                        (1)
489                 continue
490             self.people[i].test_history.append(0)
491
492     def __iter__(self):
493         return self
494
495     def next(self, filename):
496         '''
497         At each iteration, there will be:
498         * Calculate S, I, V and proportion of pro
                and against vaccine.
499         * Each person interacts with another.
```

```
500         * Write the data files.
501 
502         Parameter:
503         - filename: File name for csv output.
504         '''
505 
506         self.get_states()
507         self.set_pro_ag()
508         self.wear_off()
509         self.testing()
510         self.infect()
511         self.removed()
512         self.recovery()
513         self.vaccinate()
514         self.infected()
515         self.immune()
516         if 51 in self.mode or 52 in self.mode or 53
                in self.mode or 54 in self.mode:
517             if self.contact_nwk.update_rule == '
                   random':
518                 self.contact_nwk.update_random_nwk
                       ()
519                 if filename != '':
520                     write.WriteNetworkAvgDegree(
                           self.contact_nwk.nwk_graph,
                           filename)
521                     write.WriteNetworkAvgDegree_I(
                           self.contact_nwk.nwk_graph,
                           filename)
522                     write.WriteNetworkAvgDegree_S(
```

Code

```
                        self.contact_nwk.nwk_graph,
                        filename)
523                 write.WriteNetworkAssortativity
                        (self.contact_nwk.nwk_graph,
                        filename)
524             elif self.contact_nwk.update_rule == '
                    XBS':
525                 self.contact_nwk.
                        update_xulvi_brunet_sokolov()
526                 if filename != '':
527                     write.WriteNetworkAvgDegree(
                            self.contact_nwk.nwk_graph,
                            filename)
528                     write.WriteNetworkAvgDegree_I(
                            self.contact_nwk.nwk_graph,
                            filename)
529                     write.WriteNetworkAvgDegree_S(
                            self.contact_nwk.nwk_graph,
                            filename)
530                     write.WriteNetworkAssortativity
                            (self.contact_nwk.nwk_graph,
                            filename)
531             self.contact_nwk.update_nwk()
532
533         self.get_states()
534         self.write_history()
535         if filename != '':
536             write.WriteStates(self, filename)
```

Listing 6.4: contact.py

```
1 import random
```

```
2  import networkx as nx
3  from matplotlib import pyplot as plt
4
5  import person
6
7  class ContactNwk:
8
9      def __init__(self, people, verbose_mode):
10         self.people = people
11         self.group_no = None
12         self.network = None  # Graph to show
               partner topology
13         # From now network is defined by modes 50 -
               59.
14         self.nwk_graph = nx.Graph(self.network)
15
16         self.speed_mode = False
17         self.verbose_mode = verbose_mode
18         self.update_rule = None
19
20         # Probability to change bonds
21         self.l0 = 0.5
22         self.l1 = 0.5
23         self.assort = True
24         self.PUpdate = 0.5 # For Contact.
               update_xulvi_brunet_sokolov()
25
26     def set_default_edge_list(self):
27         '''
28         Generate edge list and set Contact.network.
```

Code

```
            By default edge list is in disjoint
            pairs.
29          '''
30          temp_roster = self.people
31          random.shuffle(temp_roster)
32          self.network = list(zip(temp_roster[:len(
               temp_roster)//2],temp_roster[len(
               temp_roster)//2:]))
33          if len(self.people) % 2 == 1:
34              self.network.append((self.people[-1],
                   None))
35
36      def show_nwk(self):
37          pos = nx.random_layout(self.nwk_graph)
38          labels = {}
39          for node in self.nwk_graph.nodes:
40              if type(node) == person.Person:
41                  labels[node] = node.id
42          nx.draw(self.nwk_graph, pos=pos,with_labels
               =False)
43          nx.draw_networkx_labels(self.nwk_graph,pos=
               pos,labels=labels,font_size=12)
44          plt.show()
45
46      def update_nwk(self):
47          '''
48          Basic functionality of network updates.
49          '''
50          # Remove people whom removed
51          to_be_removed = []
```

```
52          for node in self.nwk_graph.nodes:
53              if node.removed == 1:
54                  to_be_removed.append(node)
55          for node in to_be_removed:
56              self.nwk_graph.remove_node(node)
57
58          # Add edge list to contact_nwk.network
59          self.network = [e for e in self.nwk_graph.
               edges]
60
61      def update_xulvi_brunet_sokolov(self):
62          '''
63          Update the network. In ContactNwk().
64          '''
65          deg_ls = dict(self.nwk_graph.degree)  #
               Need this in the loop.
66          if self.verbose_mode == True:
67              print('Degree␣of␣all␣nodes␣loaded.␣')
68
69          seed = random.randint(0,10000)/10000
70          if seed > self.PUpdate:
71              return
72
73          if self.verbose_mode == True:
74              print('Proceeding␣to␣updating␣network
                   ...␣\n')
75          tmp_edge_ls = self.network
76          random.shuffle(tmp_edge_ls)
77          if self.verbose_mode == True:
78              print('Edge␣list␣shuffled,␣repairing␣
```

Code

```
                them␣now.␣')
79          edge_pairs_idx = 0
80          while edge_pairs_idx < len(tmp_edge_ls):
81              if self.verbose_mode == True:
82                  print(f'Pairing␣edges␣{
                        edge_pairs_idx}␣and␣{
                        edge_pairs_idx␣+␣1}␣out␣of␣{len(
                        tmp_edge_ls)}.␣')
83              if edge_pairs_idx == len(tmp_edge_ls)
                    -1:
84                  break
85              pair_nodes = [*tmp_edge_ls[
                    edge_pairs_idx], *tmp_edge_ls[
                    edge_pairs_idx+1]]
86              edge_pairs_idx += 2
87
88              # Sort by degree
89              pair_nodes_dict = dict([(x, deg_ls[x])
                    for x in pair_nodes])
90              pair_nodes_sorted = sorted(
                    pair_nodes_dict, key=pair_nodes_dict.
                    get, reverse=True)
91
92              if self.speed_mode != True:
93                  if self.assort == True:
94                      # Rebond then debond
95                      self.nwk_graph.remove_edge(
                            pair_nodes[0], pair_nodes[1])
96                      self.nwk_graph.remove_edge(
                            pair_nodes[2], pair_nodes[3])
```

```
97                      if len(pair_nodes_sorted) == 4:
98                          self.nwk_graph.add_edge(
                               pair_nodes_sorted[0],
                               pair_nodes_sorted[1])
99                          self.nwk_graph.add_edge(
                               pair_nodes_sorted[2],
                               pair_nodes_sorted[3])
100                     else:
101                         self.nwk_graph.add_edge(
                               pair_nodes_sorted[0],
                               pair_nodes_sorted[1])
102                         self.nwk_graph.add_edge(
                               pair_nodes_sorted[1],
                               pair_nodes_sorted[2])
103                 else:
104                     # Rebond then debond
105                     self.nwk_graph.remove_edge(
                           pair_nodes[0], pair_nodes[1])
106                     self.nwk_graph.remove_edge(
                           pair_nodes[2], pair_nodes[3])
107                     if len(pair_nodes_sorted) == 4:
108                         self.nwk_graph.add_edge(
                               pair_nodes_sorted[0],
                               pair_nodes_sorted[3])
109                         self.nwk_graph.add_edge(
                               pair_nodes_sorted[1],
                               pair_nodes_sorted[2])
110                     else:
111                         self.nwk_graph.add_edge(
                               pair_nodes_sorted[0],
```

Code

```
                                pair_nodes_sorted[2])
112
113         # Add edge list to contact_nwk.network
114         self.network = [e for e in self.nwk_graph.
                edges]
115
116
117     def update_random_nwk(self):
118         '''
119         At each time, contact clusters changed.
120         '''
121         for s_node in self.nwk_graph.nodes():
122             for t_node in self.nwk_graph.nodes():
123                 seed = random.randint(0,10000)
                        /10000
124                 # Bond
125                 if seed < self.l1 and ((s_node.id,
                        t_node.id) not in self.network or
                         (t_node.id,s_node.id) not in
                        self.network):
126                     self.nwk_graph.add_edge(s_node.
                            id,t_node.id)
127
128                 # De-bond
129                 if seed < self.l0:
130                     if (s_node.id,t_node.id) in
                            self.network:
131                         self.nwk_graph.remove_edge(
                                s_node.id,t_node.id)
132                     elif (t_node.id,s_node.id) in
```

```
                        self.network:
133                      self.nwk_graph.remove_edge(
                            t_node.id,s_node.id)
134
135         # Add edge list to contact_nwk.network
136         self.contact_nwk.network = [e for e in self
               .contact_nwk.nwk_graph.edges]
```

Listing 6.5: group.py

```
 1 import random
 2
 3 import write
 4
 5
 6 class Group:
 7     def __init__(self, people, group_size=3):
 8         self.people = people
 9         self.size = group_size
10         self.roster = {}
11
12         self.set_population()
13         self.set_roster()
14
15     def set_population(self):
16         '''
17         Initially we assign group numbers according
               to their id.
18         '''
19         for i in range(len(self.people)):
20             self.people[i].group_no = i // self.
                   size
```

Code

```
21
22      def set_roster(self):
23          for i in range((len(self.people) // self.
                size)+1):
24              self.roster[i] = []
25          for i in range(len(self.people)):
26              self.roster[self.people[i].group_no].
                    append(self.people[i])
27
28      def update_group(self, verbose_mode=False):
29          '''
30          Permutate everyone into another group.
31          '''
32          # print(self.roster)
33          for group_no, members in self.roster.items
                ():
34              for member in members:
35                  if verbose_mode == True:
36                      print(f'{member.id} is in Group
                            {group_no}')
37                  self.roster[group_no].remove(member
                        )
38                  dest_group = random.randint(0,len(
                        self.people) // self.size)
39                  if verbose_mode == True:
40                      print(f'They should be moved to
                            Group {dest_group}')
41                      print(f'This group has
                            currently have members: {[p.
                            id for p in self.roster[
```

```
                    dest_group]]}. ')
42                  self.roster[dest_group].append(
                        member)
43                  if verbose_mode == True:
44                      print(f'This group has
                            currently now have members:
                            {[p.id for p in self.roster[
                            dest_group]]}. ')
45                  member.group_no = dest_group
46
47                  swap_out = self.roster[dest_group][
                        random.randint(0,len(self.roster[
                        dest_group])-1)]
48                  if verbose_mode == True:
49                      print(f'Group {dest_group} will
                            remove {swap_out.id}. ')
50                  self.roster[dest_group].remove(
                        swap_out)
51                  self.roster[group_no].append(
                        swap_out)
52                  if verbose_mode == True:
53                      print(f'Group {dest_group} now
                            have {[p.id for p in self.
                            roster[dest_group]]}. \n\n')
54                  swap_out.group_no = dest_group
55
56      def update(self, verbose_mode=False):
57          '''
58          Update opinion using majority Rule
59          '''
```

Code

```
60          # self.inflexible_prework()
61          for group_no, members in self.roster.items
                ():
62              total_opinion = 0
63              for member in members:
64                  total_opinion += member.opinion
65              if verbose_mode == True:
66                  print(f'Group␣{group_no}␣has␣total␣
                        opinion␣of␣{total_opinion}.␣')
67              if total_opinion > len(members)//2:
68                  for member in members:
69                      member.opinion = 1
70              else:
71                  for member in members:
72                      member.opinion = 0
73          # Further update due to group concensus.
74          # self.inflexible()
75          # self.balance()
76
77
78
79      def inflexible_prework(self):
80          '''
81          Store opinions of inflexibles.
82          '''
83          for i in range(len(self.people)):
84              if self.people[i].personality == 1:
85                  self.people[i].meta_opinion = 1
86              elif self.people[i].personality == 2:
87                  self.people[i].meta_opinion = 0
```

```
88                  if verbose_mode == True:
89                      print(f'{self.people[i].id}: {
                            self.people[i].meta_opinion}'
                            )
90
91      def inflexible(self):
92          '''
93          Restore opinions of inflexibles.
94          '''
95          for i in range(len(self.people)):
96              if self.people[i].meta_opinion == 1:
97                  self.people[i].opinion = 1
98              elif self.people[i].meta_opinion == 0:
99                  self.people[i].opinion = 0
100                 if verbose_mode == True:
101                     print(f'   *{self.people[i].id
                            }: {self.people[i].opinion}')
102             self.people[i].meta_opinion = None  #
                    They will forget their deep believe
                    until next round.
103
104
105     def balance(self, verbose_mode=False):
106         for i in range(len(self.people)):
107             if self.people[i].personality == 3:
108                 if verbose_mode == True:
109                     print('Before: {} - o:{}'.
                            format(self.people[i].id,
                            self.people[i].opinion))
110                 self.people[i].swap_opinion()
```

Code

```
111                 if verbose_mode == True:
112                     print('After: {} - o:{}'.format
                            (self.people[i].id, self.
                            people[i].opinion))
113         # print('')
```

Listing 6.6: person.py

```
1  '''
2  Model Opinion Dynamics and separate them into
       groups of 3
3  '''
4
5  import numpy as np
6  import random
7
8  class Person:
9      id = 0 # Initial population
10
11     def __init__(self):
12         Person.id += 1  # Name of the person.
13         self.id = Person.id
14
15         self.location = None
16         '''
17         0 - City
18         1 - Rural
19         '''
20
21         # Bounded rationality
22         self.rV_BR = 1
23         self.rI_BR = -1
```

```
24          self.lambda_BR = 0.5
25
26          self.occupation = 0
27          '''
28          0 - Not specified
29          1 - Essential workers
30          '''
31
32          self.wealth = 1000
33
34          self.group_no = None
35
36          # Personality
37          '''
38          0 - Normal
39          1 - Inflexible (Pro)
40          2 - Inflexible (Against)
41          3 - Balancer
42          '''
43          self.personality = None
44          self.opinion = None #random.choices([0, 1],
                weights = [2, 8], k = 1)[0]
45          self.meta_opinion = None
46
47          # Epidemic state
48          self.suceptible = 0 #int(round(random.
               uniform(0, 1), 0))   # 0 means without
               disease, 1 means infected
49          self.exposed    = 0
50          self.vaccinated = 0 # Assume all 0 (None of
```

Code

```
            them took vaccine).
51          self.removed    = 0 # 0 means not in R
               compartment, 1 is.
52
53          self.infection_clock = 0
54
55          # Travelling overseas
56          self.overseas = None
57          self.A = None
58          self.rS_overseas = None
59          self.rI_overseas = None
60
61          # Demographics
62          self.age = None
63          self.gender = None  # 0 means male, 1 means
                female
64
65          self.compartment_history = []
66          self.vaccine_history = []
67          self.test_history = []
68
69      def make_population(N):
70          population = []
71          for i in range(N):
72              population.append(Person())
73          return population
74
75      def set_age(self):
76          age = random.choices(np.linspace(0, 90, 10)
               , weights = [14.5, 16.2, 15.5, 18.9,
```

```
            14.1, 10.1, 7.4, 2.2, 1, 0.1], k = 1)[0]
77          age += random.randint(0,9)
78          self.age = age
79
80      def set_gender(self):
81          self.gender = random.choices([0, 1],
                weights = [5, 5], k = 1)[0]
82
83      def swap_opinion(self):
84          if self.opinion == 0:
85              self.opinion = 1
86          else:
87              self.opinion = 0
```

Listing 6.7: mode.py

```
1  from person import Person
2
3  import random
4  import networkx as nx
5  import numpy as np
6
7  class Mode:
8      def __init__(self, people, code):
9          self.code = code
10         # Flag to alert setting has been loaded.
11         self.flag = '␣'   # If loaded then has
               value 'X'.
12         # Population objects
13         self.people = people
14
15     def raise_flag(self):
```

Code

```
        '''
        If loaded then has value 'X'.
        '''
        self.flag = 'X'

    def drop_flag(self):
        '''
        If settings unloaded then mute the flagged
           icon.
        '''
        self.flag = '␣'

    def correct_para(self, p, pos=False):
        '''
        Convert the parameters into integers.

        Parameters
        p: int
            input.
        pos: boolean
            If the parameter is positive number.
        '''
        try:
            p_num = int(p)
            if pos == True and p_num < 1:
                p_num = 1
            return p_num
        except ValueError:
            p_num = 1
            return p_num
```

```
45
46      def set_correct_para(self, p, P, pos=False):
47          '''
48          Convert the parameters into integers. If
               input is blank then do nothing.
49
50          Parameters:
51          p -- string input.
52          P -- original value.
53          pos -- If the parameter is positive number.
54          '''
55          if p == '':
56              return P
57          else:
58              return correct_para(p, pos=False)
59
60      def correct_epi_para(self, p):
61          '''
62          Convert epidemic parameters into floats.
63
64          Parameters
65          - p: Epidemic rate, positive decimal less
               than 1.
66          '''
67          try:
68              p_num = float(p)
69              if p_num < 0 or p_num > 1:
70                  p_num = 0
71                  print('Please␣check␣your␣inputs␣and
                       ␣change␣them␣in␣SETTING.')
```

Code

```
72              return p_num
73          except ValueError:
74              p_num = 0
75              print('Please␣check␣your␣inputs␣and␣
                    change␣them␣in␣SETTING.')
76              return p_num
77
78      def set_correct_epi_para(self, p, P):
79          '''
80          Convert the parameters into integers. If
               input is blank then do nothing.
81
82          Parameters:
83          p -- string input.
84          P -- original value.
85          pos -- If the parameter is positive number.
86          '''
87          if p == '':
88              return P
89          else:
90              return self.correct_epi_para(p)
91
92  '''
93  ============================================================


94
95  Individual mode settings
96
97  ============================================================
```

```
98 '''
99
100
101 '''
102 01: Living in city/ rural
103 '''
104 class Mode01(Mode):
105     '''
106     Attributes
107     ----------
108     weight: {city, rural}
109         Proportion of residents in city and rural
            respectively.
110     betas: {city, rural}
111         The infection rate while living in city or
            rural environment.
112     '''
113
114     def __init__(self, people, betas=[0.5,0.5]):
115         super().__init__(people,1)
116         self.weight = [4,6]
117         self.betas = betas
118
119     def set_weight(self, c, r):
120         self.weight = [c,r]
121         self.check_weight_integrity()
122
123     def check_weight_integrity(self):
124         if sum(self.weight) > 1:
125             print('Warning:␣Weights␣too␣much.␣Set␣
```

Code

```
                uniform␣proportion␣for␣city␣and␣
                suburban␣proportion.␣')
126             self.weight[0] = self.weight[1] = 5
127         elif sum(self.weight) < 1:
128             print('Warning:␣Weights␣too␣less.␣
                Proportion␣of␣rural␣residents␣is␣set␣
                to␣complement␣of␣city␣proportion.␣')
129             self.weight[1] = 1 - self.weight[0]
130
131     def assign_regions(self):
132         for person in self.people:
133             person.location = random.choices(list(
                range(2)), weights = self.weight, k
                =1)[0]
134
135     def __call__(self):
136         '''
137         When mode 1 is created.
138         '''
139         beta_city, beta_rural = self.betas[0], self
                .betas[1]
140         prop_city, prop_rural = self.weight[0],
                self.weight[1]
141
142         print('-------------------------')
143         print('You␣are␣creating␣mode␣1.␣')
144         print('-------------------------\n')
145         print('Please␣set␣infection␣parameter␣below
                .␣')
146         beta_city_temp = input('City␣>>>␣')
```

```
147         beta_city = super().set_correct_epi_para(
                beta_city_temp, beta_city)
148         self.set_beta(0, beta_city)
149         beta_rural_temp = input('Rural >>> ')
150         beta_rural = super().set_correct_epi_para(
                beta_rural_temp, beta_rural)
151         self.set_beta(1, beta_rural)
152 
153         print('\nPlease set proportional parameter
                below. ')
154         prop_city_temp = input('City >>> ')
155         prop_city = super().set_correct_epi_para(
                prop_city_temp, prop_city)
156         prop_rural_temp = input('Rural >>> ')
157         prop_rural = super().set_correct_epi_para(
                prop_rural_temp, prop_rural)
158         prop_city, prop_rural = self.weight[0],
                self.weight[1]
159         print('{}: {}, {}: {}'.format(self.betas
                [0], self.betas[1], self.weight[0], self.
                weight[1]))
160         print('We are assigning the population to
                regions.')
161         self.assign_regions()
162         self.raise_flag()
163         print('\nMode 1 equipped. \n')
164 
165     def set_beta(self, idx, value):
166         '''
167         Set infection rate for each region.
```

Code

```
168
169         Arguments
170         ---------
171         idx: int
172             The index according to Mode01.betas.
173         '''
174         self.betas[idx] = value
175
176
177     def infect_01(self, idx, p):
178         '''
179         Model different infection rate due to
               residence.
180         '''
181         if self.people[idx].location == 0 and p <
               self.betas[0]:
182             self.people[idx].suceptible = 1
183         elif self.people[idx].location == 1 and p <
               self.betas[1]:
184             self.people[idx].suceptible = 1
185
186
187 '''
188 02: Travel from overseas
189 '''
190 class Mode02(Mode):
191     def __init__(self, people):
192         super().__init__(people,2)
193         self.overseas = {'Some␣Places': 0}
194         self.rS = 1
```

```
195         self.rI = 1
196 
197     def __call__(self):
198         print('-------------------------')
199         print('You␣are␣creating␣mode␣2.␣')
200         print('-------------------------\n')
201         print('Please␣set␣the␣parameters␣below.␣')
202         self.raise_flag()
203 
204     def create_setting(self):
205         '''
206         Assign values to population
207         '''
208         pass
209 
210     def make_decision(self):
211         '''
212         Make decision based on circumstances in
               each time step.
213         '''
214         pass
215 
216     def get_Mode02E1(self, i):
217         '''
218         Utility function for someone (i-th person)
               decides to travel
219         '''
220         return -self.people[i].A * self.beta * self
               .rI
221 
```

Code

```
222     def get_Mode02E0(self, i):
223         '''
224         Utility function for someone (i-th person)
               decides not to travel
225         '''
226         return -self.rS
227 
228 '''
229 04: Bounded rationality of vaccine
230 '''
231 class Mode04(Mode):
232     def __init__(self, people, alpha):
233         super().__init__(people,4)
234         self.alpha = alpha
235         self.P_Alpha = []
236 
237         # Other parameters are stored within the
               person
238 
239     def __call__(self):
240         print('-------------------------')
241         print('You␣are␣creating␣mode␣4.␣')
242         print('-------------------------\n')
243         if self.alpha == 0:
244             print('Warning:␣Adoption␣parameter␣is␣
                   0,␣mode4␣will␣not␣work␣under␣this.␣
                   Please␣reset␣adoption␣parameter␣first
                   .␣')
245             return
246         print('Please␣set␣rationality␣parameter␣
```

```
            below.␣')
247         lambda_BR = self.people[0].lambda_BR
248         lambda_BR_temp = input('Lambda␣>>>␣')
249         lambda_BR = super().set_correct_para(
                lambda_BR_temp, lambda_BR, pos=True)
250         self.set_lambda(lambda_BR)
251         print('Assigning␣parameters␣to␣population.␣
                ')
252         self.QRE()
253         self.raise_flag()
254         print('\nMode␣4␣equipped.␣\n')
255
256     def set_lambda(self, lambda_input):
257         '''
258         Set rationality parameter for each person.
259
260         parameter
261         ---------
262         lambda_input: float
263             A initial value fixed for all people.
264         '''
265         for person in self.people:
266             person.lambda_BR = lambda_input
267
268     def QRE(self):
269         '''
270         Return a list of probability to adaopt
                vaccine with size of population.
271         '''
272         utility_fn = [self.alpha * person.lambda_BR
```

Code

```
            * person.rV_BR for person in self.people
            ]
273         disutility_fn = [self.alpha * person.
            lambda_BR * person.rI_BR for person in
            self.people]
274         self.P_Alpha = np.divide(np.exp(utility_fn)
            ,(np.add(np.exp(utility_fn), np.exp(
            disutility_fn))), out=np.ones_like(
            utility_fn), where=(utility_fn!=np.inf))
275         # print('QRE: ')
276         # print('U =',utility_fn)
277         # print('-U =',disutility_fn)
278         # print('p =',self.P_Alpha)
279
280
281 '''
282 05: Edit partner network
283 '''
284 class Mode05(Mode):
285     def __init__(self, people, g):
286         super().__init__(people,5)
287         self.g = g   # Import from ContactNwk
            object. Graph of social network
288         self.data = None # User requests to change
            social network topology
289
290
291     def view_network(self):
292         '''
293         Import graph from main.py and view them.
```

```
294         '''
295         try:
296             self.g.show_nwk()
297         except NameError:
298             print('Topology␣will␣be␣generated␣after
                ␣the␣first␣run.')
299             pass
300 
301     def read_data(self):
302         # Parse data stream
303         self.data = self.data.split()
304         for i in range(len(self.data)):
305             self.data[i] = self.data[i].split('-')
306             # print(self.data[i][0],self.data[i
                ][1])
307         tmp_container = []
308         for i in range(len(self.data)):
309             # print(self.data[i])
310             for j in range(len(self.people)):
311                 if self.people[j].id == int(self.
                    data[i][0]) or self.people[j].id
                    == int(self.data[i][1]):
312                     tmp_container.append(self.
                        people[j])
313                     # print(tmp_container)
314                 if len(tmp_container) == 2:
315                     if self.g.network == None:
316                         self.g.network = []
317                     self.g.network.append(tuple(
                        tmp_container))
```

Code

```
318                     self.g.nwk_graph.add_edges_from
                            ([tuple(tmp_container)])
319                     tmp_container = []
320                     print('Added')
321                     break
322         self.data = None
323
324     def __call__(self):
325         print('-------------------------')
326         print('You␣are␣editing␣mode␣5.␣')
327         print('-------------------------\n')
328         cmd = None
329         while cmd != 'y':
330             print('Please␣review␣the␣contact␣
                    network.')
331             self.view_network()
332
333             print('Input␣the␣agents␣you␣wished␣to␣
                    connect...␣')
334             print('Agents␣are␣linked␣by␣"-"␣and␣
                    pairs␣separated␣by␣space.')
335             self.data = input('>␣')
336             self.read_data()
337             if self.data != '':
338                 self.raise_flag()
339
340             cmd = input('Do␣you␣want␣to␣leave?␣[y/n
                    ]')
341             if cmd == 'y':
342                 return
```

```
343
344 '''
345 07: Age distribution
346 '''
347 class Mode07(Mode):
348     '''
349     Attributes
350     ----------
351     beta: iterable of floats (0 to 1)
352         Transmission rate of different age brackets
            .
353
354     Note
355     ----
356     Age brackets: 0 - 9, 10 - 19, 20 - 29, 30 - 39,
        40 - 49, 50 - 59, 60 - 69, 70 - 79, 80 - 89,
        90 - 99.
357
358     '''
359
360     def __init__(self, people, beta, delta):
361         super().__init__(people,7)
362         self.beta_age = [beta for x in range(10)]
363         self.delta_age = [delta for x in range(10)]
364
365     def set_population(self):
366         '''
367         Set age of a population.
368
369         parameter
```

Code

```
370         ---------
371         input: iterable, optional
372             Define frequency and their condom use.
373 
374         '''
375         for person in self.people:
376             person.set_age()
377 
378 
379 
380     def __call__(self):
381         print('-------------------------')
382         print('You␣are␣creating␣mode␣7.␣')
383         print('-------------------------\n')
384         # 0 - 9, 10 - 19, 20 - 29, 30 - 39, 40 -
              49, 50 - 59, 60 - 69, 70 - 79, 80 - 89,
              90 - 99
385 
386         # Infection
387         print('Please␣set␣infection␣parameter␣for␣
              each␣age␣brackets␣below.␣')
388         beta0_temp = input('0␣-␣9␣>>>␣')
389         self.beta_age[0] = super().
              set_correct_epi_para(beta0_temp, self.
              beta_age[0])
390         beta1_temp = input('10␣-␣19␣>>>␣')
391         self.beta_age[1] = super().
              set_correct_epi_para(beta1_temp, self.
              beta_age[1])
392         beta2_temp = input('20␣-␣29␣>>>␣')
```

```
393         self.beta_age[2] = super().
               set_correct_epi_para(beta2_temp, self.
               beta_age[2])
394         beta3_temp = input('30␣-␣39␣>>>␣')
395         self.beta_age[3] = super().
               set_correct_epi_para(beta3_temp, self.
               beta_age[3])
396         beta4_temp = input('40␣-␣49␣>>>␣')
397         self.beta_age[4] = super().
               set_correct_epi_para(beta4_temp, self.
               beta_age[4])
398         beta5_temp = input('50␣-␣59␣>>>␣')
399         self.beta_age[5] = super().
               set_correct_epi_para(beta5_temp, self.
               beta_age[5])
400         beta6_temp = input('60␣-␣69␣>>>␣')
401         self.beta_age[6] = super().
               set_correct_epi_para(beta6_temp, self.
               beta_age[6])
402         beta7_temp = input('70␣-␣79␣>>>␣')
403         self.beta_age[7] = super().
               set_correct_epi_para(beta7_temp, self.
               beta_age[7])
404         beta8_temp = input('80␣-␣89␣>>>␣')
405         self.beta_age[8] = super().
               set_correct_epi_para(beta8_temp, self.
               beta_age[8])
406         beta9_temp = input('90␣-␣99␣>>>␣')
407         self.beta_age[9] = super().
               set_correct_epi_para(beta9_temp, self.
```

Code

```
            beta_age[9])
408
409         # Removal
410         print('Please␣set␣removal␣parameter␣for␣
                each␣age␣brackets␣below.␣')
411         delta0_temp = input('0␣-␣9␣>>>␣')
412         self.delta_age[0] = super().
                set_correct_epi_para(delta0_temp, self.
                delta_age[0])
413         delta1_temp = input('10␣-␣19␣>>>␣')
414         self.delta_age[1] = super().
                set_correct_epi_para(delta1_temp, self.
                delta_age[1])
415         delta2_temp = input('20␣-␣29␣>>>␣')
416         self.delta_age[2] = super().
                set_correct_epi_para(delta2_temp, self.
                delta_age[2])
417         delta3_temp = input('30␣-␣39␣>>>␣')
418         self.delta_age[3] = super().
                set_correct_epi_para(delta3_temp, self.
                delta_age[3])
419         delta4_temp = input('40␣-␣49␣>>>␣')
420         self.delta_age[4] = super().
                set_correct_epi_para(delta4_temp, self.
                delta_age[4])
421         delta5_temp = input('50␣-␣59␣>>>␣')
422         self.delta_age[5] = super().
                set_correct_epi_para(delta5_temp, self.
                delta_age[5])
423         delta6_temp = input('60␣-␣69␣>>>␣')
```

```
424         self.delta_age[6] = super().
               set_correct_epi_para(delta6_temp, self.
               delta_age[6])
425         delta7_temp = input('70␣-␣79␣>>>␣')
426         self.delta_age[7] = super().
               set_correct_epi_para(delta7_temp, self.
               delta_age[7])
427         delta8_temp = input('80␣-␣89␣>>>␣')
428         self.delta_age[8] = super().
               set_correct_epi_para(delta8_temp, self.
               delta_age[8])
429         delta9_temp = input('90␣-␣99␣>>>␣')
430         self.delta_age[9] = super().
               set_correct_epi_para(delta9_temp, self.
               delta_age[9])
431
432         print('You␣may␣edit␣the␣proportion␣of␣each␣
               brackets␣in␣person.py.␣')
433         self.set_population()
434         self.raise_flag()
435         print('\nMode␣7␣equipped.␣\n')
436
437 '''
438 08: Gender distribution
439 '''
440 class Mode08(Mode):
441     '''
442     Attributes
443     ----------
444     beta: iterable of floats (0 to 1)
```

Code

```
445         Transmission rate of different age brackets
                .
446
447     Note
448     ----
449     0 is male, 1 is female.
450
451     '''
452
453     def __init__(self, people, beta, delta):
454         super().__init__(people,8)
455         self.beta_gender = [beta for x in range(2)]
456         self.delta_gender = [delta for x in range
                (2)]
457
458     def set_population(self):
459         '''
460         Set age of a population.
461
462         parameter
463         ---------
464         input: iterable, optional
465             Define frequency and their condom use.
466
467         '''
468         for person in self.people:
469             person.set_gender()
470
471
472     def __call__(self):
```

```
473         print('-------------------------')
474         print('You␣are␣creating␣mode␣8.␣')
475         print('-------------------------\n')
476         print('Please␣set␣infection␣parameter␣for␣
                each␣age␣brackets␣below.␣')
477         beta0_temp = input('Male␣>>>␣')
478         self.beta_gender[0] = super().
                set_correct_epi_para(beta0_temp, self.
                beta_gender[0])
479         beta1_temp = input('Female␣>>>␣')
480         self.beta_gender[1] = super().
                set_correct_epi_para(beta1_temp, self.
                beta_gender[1])
481         print('Please␣set␣removal␣parameter␣for␣
                each␣age␣brackets␣below.␣')
482         delta0_temp = input('Male␣>>>␣')
483         self.delta_gender[0] = super().
                set_correct_epi_para(delta0_temp, self.
                delta_gender[0])
484         delta1_temp = input('Female␣>>>␣')
485         self.delta_gender[1] = super().
                set_correct_epi_para(delta1_temp, self.
                delta_gender[1])
486         print('You␣may␣edit␣the␣proportion␣of␣each␣
                brackets␣in␣person.py.␣')
487         self.set_population()
488         self.raise_flag()
489         print('\nMode␣8␣equipped.␣\n')
490
491 '''
```

Code

```
10: Type of vaccine (One-off/ Seasonal/
   Chemoprophylaxis)
'''
class Mode10(Mode):
    def __init__(self, people, phi, beta):
        super().__init__(people,10)
        self.types = ['One-off', 'Seasonal', '
           Chemoprophylaxis']
        self.type = None

    def __call__(self):
        print('-------------------------')
        print('You␣are␣creating␣mode␣10.␣')
        print('-------------------------\n')
        print('Please␣set␣infection␣parameter␣below
           .␣')
        for i in range(len(self.types)):
            print(f'{i+1}.␣{self.types[i]}')
        cmd = input('Please␣choose␣one␣option:␣')
        if cmd == '1':
            self.type = 1
        elif cmd == '2':
            self.type = 2
        elif cmd == '3':
            self.type = 3
        self.raise_flag()
        print('\nMode␣10␣equipped.␣\n')

    def check_input(self, cmd):
        '''
```

```
519         Check from express mode if user has input
               an integer the corresponds to an existing
                mode.
520         '''
521         try:
522             cmd = int(cmd)
523             if cmd > 0 and cmd <= 3:
524                 return cmd
525         except ValueError:
526             print('Invalid␣vaccine␣type␣specified.␣
                   ')
527
528
529 '''
530 11: Stop transmissability/ reduce severity
531 '''
532 class Mode11(Mode):
533     def __init__(self, people):
534         super().__init__(people,11)
535         self.types = ['Stop␣transmissability', '
               Reduce␣severity']
536         self.type = None
537
538         self.beta_V = None
539         self.gamma_V = None
540         self.delta_V = None
541
542     def __call__(self, beta, gamma, delta):
543         print('-------------------------')
544         print('You␣are␣creating␣mode␣11.␣')
```

Code

```
545         print('-------------------------\n')
546         print('Please␣set␣infection␣parameter␣below
                .␣')
547         for i in range(len(self.types)):
548             print(f'{i+1}.␣{self.types[i]}')
549         cmd = input('Please␣choose␣one␣option:␣')
550         if cmd == '1':
551             self.type = 1
552             new_beta_temp = input('Beta␣>>>␣')
553             self.beta_V = super().
                    set_correct_epi_para(new_beta_temp,
                    self.beta_V)
554         elif cmd == '2':
555             self.type = 2
556             new_gamma_temp = input('Gamma␣>>>␣')
557             self.gamma_V = super().
                    set_correct_epi_para(new_gamma_temp,
                    self.gamma_V)
558             new_delta_temp = input('Delta␣>>>␣')
559             self.delta_V = super().
                    set_correct_epi_para(new_delta_temp,
                    self.delta_V)
560         self.raise_flag()
561         print('\nMode␣11␣equipped.␣\n')
562
563     def check_input(self, cmd):
564         '''
565         Check from express mode if user has input
              an integer the corresponds to an existing
               mode.
```

```
566         '''
567         try:
568             cmd = int(cmd)
569             if cmd > 0 and cmd <= 2:
570                 return cmd
571         except ValueError:
572             print('Invalid␣vaccine␣type␣specified.␣
                ')
573 
574     def check_beta(self, beta):
575         if beta < self.beta_V:
576             print('Warning:␣Your␣setting␣implies␣
                vaccine␣may␣cause␣higher␣
                tranmissibility.␣')
577     def check_gamma(self, gamma):
578         if gamma > self.gamma_V:
579             print('Warning:␣Your␣setting␣implies␣
                vaccine␣may␣cause␣lower␣effectiveness
                .␣')
580     def check_delta(self, delta):
581         if delta > self.delta_V:
582             print('Warning:␣Your␣setting␣implies␣
                vaccine␣may␣cause␣higher␣death␣rate.␣
                ')
583 
584 '''
585 20: Intimacy game
586 '''
587 class Mode20(Mode):
588     def __init__(self, people, contact_nwk, beta):
```

Code

```
589         super().__init__(people,20)
590         self.contact_nwk = contact_nwk
591         self.rV = 1
592         self.rI = -1
593         self.beta = beta
594         self.local_infection_p = np.ones(len(self.
               people))  # The proportion, not number of
                cases.
595         self.theta = np.ones(len(self.people))
596 
597         # Weights on local and global pereption
598         self.rho = 1
599 
600         self.ProbV = np.ones(len(self.people))
601 
602     def set_perceived_infection(self,
           global_infection):
603         # Clear objects
604         self.theta = np.ones(len(self.people))
605         self.local_infection_p = np.ones(len(self.
               people))
606 
607         # Start
608         local_infection = np.zeros(len(self.people)
               )
609         if self.people[0].location != None:
610             for i in range(len(self.people)):
611                 if self.people[i].location == 0:
612                     pass
613                 else:
```

```
614                     pass
615             return
616         if self.contact_nwk.network != None:
617             for i in range(len(self.people)):
618                 for neighbour in self.contact_nwk.
                        nwk_graph.neighbours(self.people[
                        i]):
619                     local_infection[i] += 1
620             print(f'There␣are␣{local_infection[i]}␣
                    people␣infected␣around␣{self.people[i
                    ].id}.␣')
621             self.local_infection_p =
                    local_infection/len(self.people)
622             self.theta = np.add(self.
                    local_infection_p * self.rho, np.ones
                    (len(self.people)) * global_infection
                     * (1-self.rho))
623         else:
624             pass
625
626 '''
627 21: Local Majority Rule
628 '''
629 class Mode21(Mode):
630     def __init__(self, people, info_nwk):
631         super().__init__(people,21)
632         self.info_nwk = info_nwk
633         self.propro = None
634         self.agpro = None
635
```

Code

```
636     def __call__(self):
637         print('-------------------------')
638         print('You␣are␣creating␣mode␣21.␣')
639         print('-------------------------\n')
640         print('Please␣set␣proportion␣of␣initial␣
                opinion␣below.␣')
641         propro_temp = input('Pro␣>>>␣')
642         self.propro = super().correct_para(
                propro_temp)
643         agpro_temp = input('Ag␣>>>␣')
644         self.agpro = super().correct_para(
                agpro_temp)
645         self.set_opinion()
646         print('All␣population␣has␣been␣assigned␣
                with␣their␣opinion.␣')
647         self.set_personality()
648         print('All␣population␣has␣been␣assigned␣
                with␣default␣personality.␣')
649         # Roster has been set already.
650         self.raise_flag()
651         print('\nMode␣21␣equipped.␣\n')
652 
653     def get_prop(self):
654         return self.propro/(self.propro+self.agpro)
655 
656     def set_pro(self, propro_temp):
657         self.propro = super().correct_para(
                propro_temp)
658 
659     def set_ag(self, ag_temp):
```

```
660         self.agpro = super().correct_para(ag_temp)
661 
662     def set_opinion(self):
663         for person in self.people:
664             seed = random.randint(0,1000)/1000
665             if seed < self.get_prop():
666                 person.opinion = 1
667             else:
668                 person.opinion = 0
669 
670     def set_personality(self):
671         for person in self.people:
672             person.personality = 0
673 
674 '''
675 22: Stubbon to take vaccine
676 '''
677 class Mode22(Mode):
678     def __init__(self, people, info_nwk):
679         super().__init__(people,22)
680         self.info_nwk = info_nwk
681         self.InflexProProportion = None
682 
683     def __call__(self):
684         print('-------------------------')
685         print('You␣are␣creating␣mode␣22.␣')
686         print('-------------------------\n')
687         print('Please␣set␣proportion␣of␣stubbon␣of␣
              pro-vaccine␣below.␣')
688         propro_temp = input('Pro␣>>>␣')
```

Code

```
689         self.assign_personality(propro_temp)
690         self.raise_flag()
691         print('\nMode␣22␣equipped.␣\n')
692 
693     def assign_personality(self, p):
694         '''
695         Assign some people with stubbon to take
               vaccine personality.
696         '''
697         p = super().set_correct_epi_para(p, 0)
698         for person in self.people:
699             if person.personality == 0:
700                 seed = random.randint(0,1000)/1000
701                 if seed < p:
702                     person.personality = 1
703                     person.opinion = 1
704 
705 '''
706 23: Stubbon to against vaccine
707 '''
708 class Mode23(Mode):
709     def __init__(self, people, info_nwk):
710         super().__init__(people,23)
711         self.info_nwk = info_nwk
712         self.InflexAgProportion = None
713 
714     def __call__(self):
715         print('-------------------------')
716         print('You␣are␣creating␣mode␣23.␣')
717         print('-------------------------\n')
```

```
718         print('Please␣set␣proportion␣of␣stubbon␣of␣
               anti-vaccine␣below.␣')
719         agpro_temp = input('Pro␣>>>␣')
720         self.assign_personality(agpro_temp)
721         self.raise_flag()
722         print('\nMode␣23␣equipped.␣\n')
723 
724     def assign_personality(self, p):
725         '''
726         Assign some people with stubbon to against
               vaccine personality.
727         '''
728         p = super().set_correct_epi_para(p, 0)
729         for person in self.people:
730             if person.personality == 0:
731                 seed = random.randint(0,1000)/1000
732                 if seed < p:
733                     person.personality = 2
734                     person.opinion = 0
735 
736 '''
737 24: Contrary to social groups
738 '''
739 class Mode24(Mode):
740     def __init__(self, people, info_nwk):
741         super().__init__(people,24)
742         self.info_nwk = info_nwk
743         self.BalancerProportion = None
744 
745     def __call__(self):
```

Code

```
        print('-------------------------')
        print('You␣are␣creating␣mode␣24.␣')
        print('-------------------------\n')
        print('Please␣set␣proportion␣of␣contrarian␣
           below.␣')
        balpro_temp = input('Pro␣>>>␣')
        self.assign_personality(balpro_temp)
        self.assign_personality()
        self.raise_flag()
        print('\nMode␣24␣equipped.␣\n')

    def assign_personality(self, p):
        '''
        Assign people with balancer personality.
        '''
        p = super().set_correct_epi_para(p, 0)
        for person in self.people:
            if person.personality == 0:
                seed = random.randint(0,1000)/1000
                if seed < p:
                    person.personality = 3

'''
31: Medication incorporated
'''
class Mode31(Mode):
    def __init__(self, people):
        super().__init__(people,31)
```

```
775     def __call__(self):
776         print('-------------------------')
777         print('You␣are␣creating␣mode␣31.␣')
778         print('-------------------------\n')
779         self.raise_flag()
780 
781 '''
782 51: Erdos-Renyi topology
783 '''
784 class Mode51(Mode):
785     def __init__(self, people, contact_nwk):
786         super().__init__(people,51)
787         # Initially set partner living in the same
              region.
788         self.contact_nwk = contact_nwk
789         self.p = 0.1  # Pairing probability
790 
791     def set_network(self):
792         self.contact_nwk.nwk_graph = nx.generators.
              random_graphs.erdos_renyi_graph(len(self.
              people), self.p)
793 
794         # Relabel nodes to People objects
795         mapping = {node: self.people[node] for node
               in self.contact_nwk.nwk_graph}
796         self.contact_nwk.nwk_graph = nx.
              relabel_nodes(self.contact_nwk.nwk_graph,
               mapping)
797 
798         # Random pair people with no partners with
```

Code

```
            other partners
799         for node in self.contact_nwk.nwk_graph.
               nodes:
800             if self.contact_nwk.nwk_graph.degree(
                   node) == 0:
801                 random_node = random.choice(list(
                       self.contact_nwk.nwk_graph.nodes
                       ()))
802                 self.contact_nwk.nwk_graph.add_edge
                       (node, random_node)
803
804         # Add edge list to contact_nwk.network
805         self.contact_nwk.network = [e for e in self
               .contact_nwk.nwk_graph.edges]
806
807     def set_p(self, p):
808         if p > 1:
809             self.p = 1
810         elif p < 0:
811             self.p = 0
812         else:
813             self.p = p
814
815     def set_pupdate(self, p):
816         '''
817         Set probability to update network
818         '''
819         if p > 1:
820             self.contact_nwk.PUpdate = 1
821         elif p < 0:
```

```
822             self.contact_nwk.PUpdate = 0
823         else:
824             self.contact_nwk.PUpdate = p
825 
826     def set_l0(self, l0):
827         '''
828         Set probability to debond
829         '''
830         if l0 > 1:
831             self.contact_nwk.l0 = 1
832         elif l0 < 0:
833             self.contact_nwk.l0 = 0
834         else:
835             self.contact_nwk.l0 = l0
836 
837     def set_l1(self, l1):
838         '''
839         Set probability to rebond
840         '''
841         if l0 > 1:
842             self.contact_nwk.l1 = 1
843         elif l0 < 0:
844             self.contact_nwk.l1 = 0
845         else:
846             self.contact_nwk.l1 = l1
847 
848     def __call__(self):
849         print('-------------------------')
850         print('You␣are␣creating␣mode␣51.␣')
851         print('-------------------------\n')
```

Code

```
852         print('Please␣set␣infection␣parameter␣below
                .␣')
853         try:
854             p_temp = float(input('p␣>>>␣'))
855         except ValueError:
856             print('Invalid␣data␣type␣for␣p,␣set␣p␣
                    as␣1.␣')
857             p_temp = 1
858         self.set_p(p_temp)
859         cmd = input('Longitudinal␣social␣network?␣[
                y/n]␣')
860         if cmd == 'y':
861             print('Default␣rule:␣independent␣update
                    .␣')
862             cmd_update_rule = input('Change?␣[y/n]␣
                    ')
863             if cmd_update_rule == 'y':
864                 self.contact_nwk.update_rule = 'XBS
                        '
865                 pUpd_temp = float(input('p␣>>>␣'))
866                 pUpd = super().set_correct_epi_para
                        (pUpd_temp, self.contact_nwk.
                        PUpdate)
867                 self.set_pupdate(pUpd)
868             else:
869                 self.contact_nwk.update_rule = '
                        random'
870                 l0_temp = float(input('l0␣>>>␣'))
871                 l0 = super().set_correct_epi_para(
                        l0, self.contact_nwk.l0)
```

```
872                 self.set_p(10)
873         self.set_network()
874         self.raise_flag()
875         print('E-R graph settings done.')
876 
877 
878 '''
879 52: Preferential attachment.
880 '''
881 class Mode52(Mode):
882     def __init__(self, people, contact_nwk=None):
883         super().__init__(people,52)
884         # Initially set partner living in the same
                region.
885         self.contact_nwk = contact_nwk
886         self.m = 1  # Pairing probability
887 
888     def set_network(self):
889         '''
890         Setup the network and nwk_graph
891         '''
892         self.contact_nwk.nwk_graph = nx.generators.
                random_graphs.barabasi_albert_graph(len(
                self.people), self.m)
893 
894         # Relabel nodes to People objects
895         mapping = {node: self.people[node] for node
                 in self.contact_nwk.nwk_graph}
896         self.contact_nwk.nwk_graph = nx.
                relabel_nodes(self.contact_nwk.nwk_graph,
```

Code

```
            mapping)
897
898         # Add edge list to contact_nwk.network
899         self.contact_nwk.network = [e for e in self
                .contact_nwk.nwk_graph.edges]
900
901     def set_pupdate(self, p):
902         '''
903         Set probability to update network
904         '''
905         if p > 1:
906             self.contact_nwk.PUpdate = 1
907         elif p < 0:
908             self.contact_nwk.PUpdate = 0
909         else:
910             self.contact_nwk.PUpdate = p
911
912     def set_l0(self, l0):
913         '''
914         Set probability to debond
915         '''
916         if l0 > 1:
917             self.contact_nwk.l0 = 1
918         elif l0 < 0:
919             self.contact_nwk.l0 = 0
920         else:
921             self.contact_nwk.l0 = l0
922
923     def set_l1(self, l1):
924         '''
```

```
925         Set probability to rebond
926         '''
927         if l0 > 1:
928             self.contact_nwk.l1 = 1
929         elif l0 < 0:
930             self.contact_nwk.l1 = 0
931         else:
932             self.contact_nwk.l1 = l1
933 
934     def set_m(self, m):
935         if m < 1:
936             self.m = 1
937         else:
938             self.m = m
939 
940     def __call__(self):
941         '''
942         Setting mode 52 into model. If other
               network modes have set, they are dropped
               by `main.py`.
943         '''
944         print('-------------------------')
945         print('You␣are␣creating␣mode␣52.␣')
946         print('-------------------------\n')
947         print('Please␣set␣infection␣parameter␣below
               .␣')
948         try:
949             m_temp = int(input('m␣>>>␣'))
950         except ValueError:
951             print('Invalid␣data␣type␣for␣m,␣set␣m␣
```

Code

```
                as␣1.␣')
952             m_temp = 1
953         self.set_m(m_temp)
954
955         cmd = input('Longitudinal␣social␣network?␣[
                y/n]␣')
956         if cmd == 'y':
957             print('Default␣rule:␣Xulvi-Brunet␣
                    Sokolov.␣')
958             cmd_update_rule = input('Change?␣[y/n]␣
                    ')
959             if cmd_update_rule == 'y':
960                 self.contact_nwk.update_rule = '
                        random'
961                 l0_temp = float(input('l0␣>>>␣'))
962                 l0Upd = super().
                        set_correct_epi_para(l0_temp,
                        self.contact_nwk.l0)
963                 self.set_l0(l0Upd)
964                 l1_temp = float(input('l1␣>>>␣'))
965                 l1Upd = super().
                        set_correct_epi_para(l1_temp,
                        self.contact_nwk.l1)
966                 self.set_l1(l1Upd)
967             else:
968                 self.contact_nwk.update_rule = 'XBS
                        '
969                 pUpd_temp = float(input('p␣>>>␣'))
970                 pUpd = super().set_correct_epi_para
                        (pUpd_temp, self.contact_nwk.
```

```
                    PUpdate)
971                 self.set_pupdate(pUpd)
972         self.set_network()
973         self.raise_flag()
974         print('Preferential␣attachment␣graph␣
               settings␣done.')
975
976 '''
977 53: Small world network
978 '''
979 class Mode53(Mode):
980     def __init__(self, people, contact_nwk=None):
981         super().__init__(people,53)
982         # Initially set partner living in the same
               region.
983         self.contact_nwk = contact_nwk
984         self.k = 1  # k neighbours are joined
985         self.p = 1  # Rewiring probability
986
987     def set_network(self):
988         '''
989         Setup the network and nwk_graph
990         '''
991         self.contact_nwk.nwk_graph = nxgenerators.
               random_graphs.watts_strogatz_graph(len(
               self.people), self.k, self.p)
992
993         # Relabel nodes to People objects
994         mapping = {node: self.people[node] for node
                in self.contact_nwk.nwk_graph}
```

Code

```
995         self.contact_nwk.nwk_graph = nx.
               relabel_nodes(self.contact_nwk.nwk_graph,
                mapping)
996
997         # Add edge list to contact_nwk.network
998         self.contact_nwk.network = [e for e in self
               .contact_nwk.nwk_graph.edges]
999
1000    def set_k(self, m):
1001        if m < 1:
1002            self.k = 1
1003        else:
1004            self.k = m
1005
1006    def set_p(self, p):
1007        if p > 1:
1008            self.p = 1
1009        elif p < 0:
1010            self.p = 0
1011        else:
1012            self.p = p
1013
1014    def set_pupdate(self, p):
1015        '''
1016        Set probability to update network
1017        '''
1018        if p > 1:
1019            self.contact_nwk.PUpdate = 1
1020        elif p < 0:
1021            self.contact_nwk.PUpdate = 0
```

```
1022            else:
1023                self.contact_nwk.PUpdate = p
1024
1025        def set_l0(self, l0):
1026            '''
1027            Set probability to debond
1028            '''
1029            if l0 > 1:
1030                self.contact_nwk.l0 = 1
1031            elif l0 < 0:
1032                self.contact_nwk.l0 = 0
1033            else:
1034                self.contact_nwk.l0 = l0
1035
1036        def set_l1(self, l1):
1037            '''
1038            Set probability to rebond
1039            '''
1040            if l0 > 1:
1041                self.contact_nwk.l1 = 1
1042            elif l0 < 0:
1043                self.contact_nwk.l1 = 0
1044            else:
1045                self.contact_nwk.l1 = l1
1046
1047        def __call__(self):
1048            '''
1049            Setting mode 53 into model. If other
                network modes have set, they are dropped
                by `main.py`.
```

Code

```
        '''
        print('-------------------------')
        print('You␣are␣creating␣mode␣53.␣')
        print('-------------------------\n')
        print('Please␣set␣infection␣parameter␣below
           .␣')
        try:
            k_temp = int(input('k␣>>>␣'))
        except ValueError:
            print('Invalid␣data␣type␣for␣m,␣set␣m␣
               as␣1.␣')
            k_temp = 1
        self.set_k(k_temp)
        try:
            p_temp = int(input('p␣>>>␣'))
        except ValueError:
            print('Invalid␣data␣type␣for␣p,␣set␣m␣
               as␣1.␣')
            p_temp = 1
        self.set_p(p_temp)
        '''
        Set update rule
        '''
        cmd = input('Longitudinal␣social␣network?␣[
           y/n]␣')
        if cmd == 'y':
            print('Default␣rule:␣independent␣update
               .␣')
            cmd_update_rule = input('Change?␣[y/n]␣
               ')
```

```
            if cmd_update_rule == 'y':
                self.contact_nwk.update_rule = 'XBS
                    '
                pUpd_temp = float(input('p␣>>>␣'))
                pUpd = super().set_correct_epi_para
                    (pUpd_temp, self.contact_nwk.
                    PUpdate)
                self.set_pupdate(pUpd)
            else:
                self.contact_nwk.update_rule = '
                    random'
                l0_temp = float(input('l0␣>>>␣'))
                l0Upd = super().
                    set_correct_epi_para(l0_temp,
                    self.contact_nwk.l0)
                self.set_l0(l0Upd)
                l1_temp = float(input('l1␣>>>␣'))
                l1Upd = super().
                    set_correct_epi_para(l1_temp,
                    self.contact_nwk.l1)
                self.set_l1(l1Upd)
        self.set_network()
        self.raise_flag()
        print('Watts-Strogatz␣graph␣settings␣done.'
            )

'''
54: Lattice network
'''
class Mode54(Mode):
```

Code

```
1095    def __init__(self, people, contact_nwk=None):
1096        super().__init__(people,54)
1097        # Initially set partner living in the same
               region.
1098        self.contact_nwk = contact_nwk
1099        self.m = 1  # Nunber of rows
1100        self.n = len(self.people)//self.m  # Nunber
               of columns
1101
1102    def set_network(self):
1103        '''
1104        Setup the network and nwk_graph
1105        '''
1106        self.contact_nwk.nwk_graph = nx.generators.
               lattice.grid_2d_graph(len(self.people),
               self.m, self.n)
1107
1108        # Relabel nodes to People objects
1109        mapping = {node: self.people[node] for node
               in self.contact_nwk.nwk_graph}
1110        self.contact_nwk.nwk_graph = nx.
               relabel_nodes(self.contact_nwk.nwk_graph,
               mapping)
1111
1112        # Add edge list to contact_nwk.network
1113        self.contact_nwk.network = [e for e in self
               .contact_nwk.nwk_graph.edges]
1114
1115    def set_dim(self, m):
1116        if m < 1:
```

```
1117            self.m = 1
1118        else:
1119            self.m = m
1120        self.n = len(self.people)//self.m
1121
1122    def set_pupdate(self, p):
1123        '''
1124        Set probability to update network
1125        '''
1126        if p > 1:
1127            self.contact_nwk.PUpdate = 1
1128        elif p < 0:
1129            self.contact_nwk.PUpdate = 0
1130        else:
1131            self.contact_nwk.PUpdate = p
1132
1133    def set_l0(self, l0):
1134        '''
1135        Set probability to debond
1136        '''
1137        if l0 > 1:
1138            self.contact_nwk.l0 = 1
1139        elif l0 < 0:
1140            self.contact_nwk.l0 = 0
1141        else:
1142            self.contact_nwk.l0 = l0
1143
1144    def set_l1(self, l1):
1145        '''
1146        Set probability to rebond
```

Code

```
        '''
        if l0 > 1:
            self.contact_nwk.l1 = 1
        elif l0 < 0:
            self.contact_nwk.l1 = 0
        else:
            self.contact_nwk.l1 = l1

    def __call__(self):
        '''
        Setting mode 52 into model. If other
           network modes have set, they are dropped
           by `main.py`.
        '''
        print('-------------------------')
        print('You␣are␣creating␣mode␣54.␣')
        print('-------------------------\n')
        print('Please␣set␣infection␣parameter␣below
           .␣')
        try:
            m_temp = int(input('m␣>>>␣'))
        except ValueError:
            print('Invalid␣data␣type␣for␣m,␣set␣m␣
               as␣1.␣')
            m_temp = 1
        self.set_dim(m_temp)
        '''
        Set update rule
        '''
        cmd = input('Longitudinal␣social␣network?␣[
```

```
            y/n]␣')
1173        if cmd == 'y':
1174            print('Default␣rule:␣independent␣update
                .␣')
1175            cmd_update_rule = input('Change?␣[y/n]␣
                ')
1176            if cmd_update_rule == 'y':
1177                self.contact_nwk.update_rule = 'XBS
                    '
1178                pUpd_temp = float(input('p␣>>>␣'))
1179                pUpd = super().set_correct_epi_para
                    (pUpd_temp, self.contact_nwk.
                    PUpdate)
1180                self.set_pupdate(pUpd)
1181            else:
1182                self.contact_nwk.update_rule = '
                    random'
1183                l0_temp = float(input('l0␣>>>␣'))
1184                l0Upd = super().
                    set_correct_epi_para(l0_temp,
                    self.contact_nwk.l0)
1185                self.set_l0(l0Upd)
1186                l1_temp = float(input('l1␣>>>␣'))
1187                l1Upd = super().
                    set_correct_epi_para(l1_temp,
                    self.contact_nwk.l1)
1188                self.set_l1(l1Upd)
1189        self.set_network()
1190        self.raise_flag()
1191        print('Preferential␣attachment␣graph␣
```

Code

```
            settings␣done.')
```

Listing 6.8: write.py

```
1  import csv
2  import networkx as nx
3  import datetime
4
5  def WriteStates(obs, filename):
6      '''
7          Write everyone's infected state into a .csv
               file.
8      '''
9      filename = str(filename)+'.csv'
10     with open(filename, 'a', newline='', encoding='
          utf8') as f:
11         writer = csv.writer(f)
12         writer.writerow([obs.S, obs.I, obs.V, obs.R
              ])
13
14 def WriteCompartmentHistory (obs, filename):
15     '''
16         Write everyone's compartment state into a .
              csv file.
17     '''
18     filename = str(filename)+'-compartment.csv'
19     for i in range(len(obs.people)):
20         with open(filename, 'a', newline='',
              encoding='utf8') as f:
21             writer = csv.writer(f)
22             writer.writerow(obs.people[i].
                  compartment_history)
```

```
23
24 def WriteOpinion(obs, filename):
25     '''
26         Write everyone's opinion and infected state
               into a .csv file.
27     '''
28     filename = str(filename)+'-opinion.csv'
29     with open(filename, 'a', newline='', encoding='
          utf8') as f:
30         writer = csv.writer(f)
31         for i in range(len(obs.people)):
32             writer.writerow([obs.people[i].group_no
                  , obs.people[i].id, obs.people[i].
                  opinion])
33
34 def WriteOpinionPersonality(obs, filename):
35     '''
36         Write everyone's opinion into a .csv file.
             Their personality are flagged as well.
37
38         Coulmns
39         - Group number of the agent
40         - Agent name
41         - Agent's personality
42             - 0 means normal
43             - 1 means inflexible (pro-vaccine)
44             - 2 means inflexible (against)
45             - 3 means balancer
46         - Agent's opinion at time step
47     '''
```

Code

```
48      filename = str(filename)+'-opinion.csv'
49      with open(filename, 'a', newline='', encoding='
           utf8') as f:
50          writer = csv.writer(f)
51          for i in range(len(obs.people)):
52              writer.writerow([obs.people[i].group_no
                   , obs.people[i].id, obs.people[i].
                   opinion, obs.people[i].personality])
53
54  def WriteNetwork(graph_obj, filename):
55      export_graph = graph_obj
56      mapping = {}
57      for node in graph_obj.nodes:
58          mapping[node] = node.id
59      export_graph = nx.relabel_nodes(export_graph,
           mapping)
60      nx.write_graphml(export_graph, filename+'.
           graphml')
61
62  def WriteNetworkAvgDegree(graph_obj, filename):
63      '''
64      Argument
65      --------
66      graph_obj: Graph
67          The graph to be calculated.
68      filename: str
69          Output filename.
70      '''
71      filename = filename+'-nwk-deg.csv'
72      with open(filename, 'a', newline='') as f:
```

```
73          writer=csv.writer(f)
74          writer.writerow([2 * graph_obj.
                number_of_edges()/graph_obj.
                number_of_nodes()])
75
76  def WriteNetworkAvgDegree_I(graph_obj, filename):
77      '''
78      Argument
79      --------
80      graph_obj: Graph
81          The graph to be calculated.
82      filename: str
83          Output filename.
84      '''
85      filename = filename+'-nwk-deg_I.csv'
86      deg_I = {}
87      for node in graph_obj.nodes():
88          if node.suceptible == 1:
89              deg_I[node] = graph_obj.degree[node]
90      content = [v for v in deg_I.values()]
91      try: content.append(sum(content)/len(content))
92      except ZeroDivisionError: content.append(0)
93      with open(filename, 'a', newline='') as f:
94          writer=csv.writer(f)
95          writer.writerow(content)
96
97  def WriteNetworkAvgDegree_S(graph_obj, filename):
98      '''
99      Argument
100     --------
```

Code

```
    graph_obj: Graph
        The graph to be calculated.
    filename: str
        Output filename.

    Note
    ----
    This include the average degree of people whom
       vaccinated.
    '''
    filename = filename+'-nwk-deg_S.csv'
    deg_S = {}
    for node in graph_obj.nodes():
        if node.suceptible == 0 and node.removed ==
            0:
            deg_S[node] = graph_obj.degree[node]
    content = [v for v in deg_S.values()]
    try: content.append(sum(content)/len(content))
    except ZeroDivisionError: content.append(0)
    with open(filename, 'a', newline='') as f:
        writer=csv.writer(f)
        writer.writerow(content)

def WriteNetworkAssortativity(graph_obj, filename):
    '''
    Argument
    --------
    graph_obj: Graph
        The graph to be calculated.
    filename: str
```

```
129         Output filename.
130     '''
131     filename = filename+'-nwk-assort.csv'
132     with open(filename, 'a', newline='') as f:
133         writer=csv.writer(f)
134         writer.writerow([nx.
              degree_assortativity_coefficient(
              graph_obj)])
135
136 def WriteNetworkData(obs):
137     '''
138     Save basic network information.
139
140     Parameters
141     ----------
142     obs: Simulation
143         Accepts Simulation object
144     filename: str
145         File name for export
146     '''
147     text = []
148
149     text.append('============================\n\n'
          )
150     text.append('Condom␣usage\n\n')
151     text.append('============================\n\n'
          )
152     text.append('#␣Basic␣data\n\n')
153     text.append('Number␣of␣agents␣(N):␣{}\n\n'.
          format(len(obs.N)))
```

Code

```
154
155 def WriteTestingHistory(obs, filename):
156     filename = str(filename)+'-testing.csv'
157     for i in range(len(obs.people)):
158         with open(filename, 'a', newline='',
               encoding='utf8') as f:
159             writer = csv.writer(f)
160             writer.writerow(obs.people[i].
                   test_history)
161
162 def WriteSummary(obs, filename):
163     '''
164     Write simulation summary.
165
166     Parameters
167     ----------
168     obs: Simulation
169         Accepts Simulation object
170     filename: str
171         File name for export
172     '''
173     with open('{}-summary.txt'.format(filename), 'w
          ') as f:
174         contents = ['␣
              ========================================
              ␣\n\n','␣','␣Agent␣Based␣Modelling:␣COVID
              -19␣SEIP␣Model␣\n\n','␣','␣
              ========================================
              ␣\n']
175         contents.append('\n\nThis␣simulation␣was␣
```

```
            performed␣on␣{}.\n\n'.format(datetime.
            datetime.now().strftime('%H:%M:%S,␣%d/␣%m
            /␣%Y␣(%z)')))
176         contents.append('Simulation␣name:␣{}\n\n'.
            format(filename))
177         contents.append('#␣Summary\n')
178         contents.append('N:␣{}␣people\n'.format(len
            (obs.N)))
179         contents.append('T:␣{}␣days\n'.format(obs.T
            ))
180         contents.append('\n##␣Epidemiology\n')
181         contents.append('Alpha:␣{}\n'.format(obs.
            alpha))
182         if any(i in obs.modes for i in [1, 7, 8]):
183             contents.append('Beta:␣*\n')
184         else:
185             contents.append('Beta:␣{}\n'.format(obs
                .beta))
186         contents.append('Gamma:␣{}\n'.format(obs.
            gamma))
187         if any(i in obs.modes for i in [7, 8]):
188             contents.append('Delta:␣*\n')
189         else:
190             contents.append('Delta:␣{}\n'.format(
                obs.delta))
191         contents.append('Phi:␣{}\n'.format(obs.phi)
            )
192         contents.append('Tau:␣{}\n'.format(obs.
            test_rate))
193         contents.append('Immune␣time:␣{}␣days␣\n'.
```

Code

```
            format(obs.immune_time))
194         contents.append('Test␣rate:␣{}␣\n'.format(
               obs.test_rate))
195         if any(i in obs.modes for i in [7, 8]):
196             contents.append('\n#␣Demographics␣\n')
197             if 7 in obs.modes:
198                 age_backets = ['0␣-␣9', '10␣-␣19',
                       '20␣-␣29', '30␣-␣39', '40␣-␣49',
                       '50␣-␣59', '60␣-␣69', '70␣-␣79',
                       '80␣-␣89', '90␣-␣99']
199                 contents.append('\n##␣Age␣factor\n'
                       )
200                 contents.append('\nAge␣group␣␣Beta␣
                       ␣␣Delta␣')
201                 contents.append('\n---------␣␣----␣
                       ␣␣-----␣\n')
202                 for i in range(len(age_backets)):
203                     contents.append('{}␣␣␣␣{}␣␣{}\n
                           '.format(age_backets[i], obs.
                           modes[7].beta_age[i], obs.
                           modes[7].delta_age[i]))
204             if 8 in obs.modes:
205                 gender = ['Male', 'Female']
206                 contents.append('\n##␣Gender␣factor
                       \n')
207                 contents.append('\nGender␣␣Beta␣␣␣
                       Delta␣')
208                 contents.append('\n------␣␣----␣␣␣
                       -----␣\n')
209                 for i in range(len(gender)):
```

```
210                     contents.append('{}␣␣␣{}␣{}\n'.
                            format(gender[i], obs.modes
                            [8].beta_gender[i], obs.modes
                            [8].delta_gender[i]))
211             if 1 in obs.modes:
212                 contents.append('\n#␣City␣and␣Rural␣
                        Compartment␣\n')
213                 contents.append('Proportion␣living␣in␣
                        city,␣{}␣\n'.format(obs.modes[1].
                        weight[0]))
214                 contents.append('Proportion␣living␣in␣
                        rural␣area,␣{}␣\n'.format(obs.modes
                        [1].weight[1]))
215                 contents.append('\n##␣Transmission␣
                        parameter␣\n')
216                 contents.append('City:␣{}␣\n'.format(
                        obs.modes[1].betas[0]))
217                 contents.append('Rural:␣{}␣\n'.format(
                        obs.modes[1].betas[1]))
218             if any(i in obs.modes for i in [4, 21]):
219                 contents.append('\n#␣Game␣Theoretical␣
                        Option␣\n')
220                 if 4 in obs.modes:
221                     contents.append('\n##␣Bounded␣
                            Rationality␣\n')
222                     contents.append('Alpha:␣{}\n'.
                            format(obs.alpha))
223                     contents.append('Rationality␣
                            parameter:␣\n')
224                     contents.append('Append␣mode:␣Fixed
```

Code

```
                    \n')
225                 contents.append('N: {} people\n'.
                        format(len(obs.N)))
226                 contents.append('Value: {} \n'.
                        format(obs.people[0].lambda_BR))
227                 contents.append('P(V): {} \n'.
                        format(obs.modes[4].P_Alpha[0]))
228             if any(i in obs.modes for i in [51, 52, 53,
                    54]):
229                 contents.append('\n# Network Topology \
                        n')
230                 if 51 in obs.modes:
231                     nwk_type = "Erdos-Renyi"
232                 elif 52 in obs.modes:
233                     nwk_type = "Barabasi-Albert"
234                 elif 53 in obs.modes:
235                     nwk_type = "Watts-Strogatz"
236                 elif 54 in obs.modes:
237                     nwk_type = "Lattice"
238                 contents.append('Type: {}\n\n'.format(
                        nwk_type))
239                 contents.append('\n## Basic Network
                        Quantities \n')
240                 contents.append('Nodes: {}\n'.format(
                        obs.contact_nwk.nwk_graph.
                        number_of_nodes()))
241                 contents.append('Edges: {}\n'.format(
                        obs.contact_nwk.nwk_graph.
                        number_of_edges()))
242                 contents.append('Avg degree: {}\n'.
```

```
                format(2 * obs.contact_nwk.nwk_graph.
                number_of_edges()/obs.contact_nwk.
                nwk_graph.number_of_nodes()))
243             contents.append('Assortativity: {}\n'.
                format(nx.
                degree_assortativity_coefficient(obs.
                contact_nwk.nwk_graph)))
244             if obs.contact_nwk.update_rule != None:
245                 contents.append('\n## Longitudinal
                    network \n')
246                 if obs.contact_nwk.update_rule != '
                    random':
247                     contents.append('Type:
                        Independent\n')
248                     contents.append('Probability
                        to bond (l0): {}\n'.format(
                        obs.contact_nwk.l0))
249                     contents.append('Probability
                        to debond (l1): {}\n'.format
                        (obs.contact_nwk.l1))
250                 elif obs.contact_nwk.update_rule !=
                    'XBS':
251                     contents.append('Type: XBS\n')
252                     contents.append('Rewire
                        probability: {}\n'.format(obs
                        .contact_nwk.PUpdate))
253                     contents.append('Rewire type:
                        {}\n'.format(obs.contact_nwk.
                        assort))
254                 contents.append('Average degree per
```

Code

```
                    time step stored in "{}-nwk-deg.
                    csv"\n'.format(obs.filename))
255                 contents.append('Assortativity
                    information per time step stored
                    in "{}-nwk-assort.csv"\n'.format(
                    obs.filename))
256         if any(i in obs.modes for i in [1, 7, 8]):
257             contents.append('\n## Longitudinal
                    Network \n')
258             contents.append('# Notes\n')
259             contents.append('* Epidemic parameter
                    controlled by optional modes. Consult
                    the relevant modes for more
                    information. \n')
260         f.writelines(contents)
```

1 Code of Data Analysis

Listing 6.9: Run script to generate different longitudinal social network.

```
1 import os
2 import time
3
4 '''
5 Warning:
6
7 If you see one (or more of your code has no results
    , like just flattened). Rerun that again and good
     luck.
8 '''
9
```

```
10
11 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
      0.000005␣-m␣--52␣*m=3␣*p=0.1␣-f␣20201127_run01␣
      run')
12 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
      0.000005␣-m␣--52␣*m=3␣*p=0.1␣*a=0␣-f␣20201127
      _run02␣run')
13 print('Rest␣for␣10␣seconds...␣')
14 time.sleep(10)
15 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
      0.000005␣-m␣--52␣*m=3␣*p=0.9␣-f␣20201127_run03␣
      run')
16 print('Rest␣for␣10␣seconds...␣')
17 time.sleep(10)
18 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
      0.000005␣-m␣--52␣*m=3␣*p=0.9␣*a=0␣-f␣20201127
      _run04␣run')
19
20 print('Rest␣for␣10␣seconds...␣')
21 time.sleep(10)
22 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
      0.000005␣-m␣--52␣*m=20␣*p=0.1␣-f␣20201127_run05␣
      run')
23 print('Rest␣for␣10␣seconds...␣')
24 time.sleep(10)
25 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
      0.000005␣-m␣--52␣*m=20␣*p=0.1␣*a=0␣-f␣20201127
      _run06␣run')
26
27 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
```

Code

```
    0.000005␣-m␣--52␣*m=20␣*p=0.9␣-f␣20201127_run07␣
    run')
28 os.system('py␣main.py␣100000␣500␣0␣0.14␣0.05␣0␣
    0.000005␣-m␣--52␣*m=20␣*p=0.9␣*a=0␣-f␣20201127
    _run08␣run')
```

Listing 6.10: Run script to generate different immune period.

```
1  import os
2  import time
3
4
5  os.system('py␣main.py␣100000␣300␣0␣0.14␣0.05␣0␣
    0.000005␣-immune_time␣0␣-f␣20201127_run09_0␣run')
6  # print('Rest for 10 seconds... ')
7  # time.sleep(10)
8  # os.system('py main.py 100000 300 0 0.14 0.05 0
    0.000005 -immune_time 60 -f 20201127_run09_60 run
    ')
9  # print('Rest for 10 seconds... ')
10 # time.sleep(10)
11 # os.system('py main.py 100000 300 0 0.14 0.05 0
    0.000005 -immune_time 180 -f 20201127_run09_180
    run')
12 # print('Rest for 10 seconds... ')
13 # time.sleep(10)
14 # os.system('py main.py 100000 300 0 0.14 0.05 0
    0.000005 -immune_time 210 -f 20201127_run09_210
    run')
```

Listing 6.11: Run script to generate different inflexibles and balancers mix.

```
1  import numpy as np
```

```
2  import pandas as pd
3
4  import glob
5
6  import matplotlib.pyplot as plt
7  from matplotlib import cm
8
9  N = 100000
10
11 data_files = glob.glob("data/20201118/*.csv")
12 # print(data_files)
13 all_df = {}
14 mix = {}
15
16 for filename in data_files:
17     function_name = filename.split('-')
18     if len(function_name) == 2:
19         function_name = function_name[-1][:-4]
20     else:
21         function_name = ''
22     if function_name == '':
23         df_filename = filename[14:-4]
24         parsed = df_filename.split('_')
25         df_name = [float(parsed[3].split('p')[1].
               split('a')[0]), *[float(x) for x in
               parsed[3].split('p')[1].split('a')[1].
               split('b')]]
26         print(f'Reading␣{filename}...␣')
27         all_df[str(df_name)] = pd.read_csv(filename
               , names=['S', 'I', 'V', 'R'])
```

Code

```
        mix[str(df_name)] = df_name

I = np.zeros((21,41))
V = np.zeros((21,41))
for k, df in all_df.items():
    coor = np.array(mix[k])/2.5
    # print(coor[0], coor[2])
    last_record = df.tail(1).values.tolist()[0]
    # print(last_record)
    # print('Before:',I[int(coor[0]), int(coor[2])])
    I[int(coor[0]), int(coor[2])] = last_record[1]/N
    V[int(coor[0]), int(coor[2])] = last_record[2]/N
    # print('After:',I[int(coor[0]), int(coor[2])])
I = np.transpose(I)
V = np.transpose(V)

plt.matshow(I, cmap='gray_r')
plt.ylabel('Proportion of balancers')
plt.xlabel('Proportion of inflexibles')
plt.title('Effect on long term infection from personality')
plt.colorbar()
plt.show()

plt.matshow(V, cmap='gray_r')
plt.ylabel('Proportion of balancers')
plt.xlabel('Proportion of inflexibles')
```

```
54 plt.title('Effect␣on␣long␣term␣vaccine␣adoption␣
      from␣personality')
55 plt.colorbar()
56 plt.show()
```