CS166 Final Project - COVID-19 Simulations

Viet Hoang Tran Duong

Spring 2020

Table of content:

Part 1: Introduction

COVID-19 is a contagious disease that can trigger an exponential infection in the population. The pandemics have infected almost 3 million people and killed over 200 thousand lives (as of April 24th). Different countries proposed different approaches to counter the epidemics, with three formal proposals: lockdown, mass testing, and second-order tracing. The details of each approach will be described in the following sections. This analysis will attempt to compare which approach yields the most effective result for the population. The goal is not to predict how many people will get infected but more to examine what policy should be implemented. Hence, even though some non-primary parameters might not be representative, the final comparisons of different policies should still hold.

Part 2: Models & Parameters

This model consists of 2 components: Social Dynamic and Covid-19. The Social Dynamic is to create a realistic social network and input this network directly to see the effects of pandemics and policies. The social structure is inspired by Assignment 3 (SocialDynamicSimulation). The Covid-19 is the new model with flexible parameters to mimic the behaviors of the pandemics and government approaches.

2.1. Social Dynamic

The reason I chose Social dynamics is that it represents more accurately the social settings in the current world. My SocialDynamicSimulation starts with the Barabasi-Albert (BA) model for a random scale-free network with a preferential attachment mechanism. The reason I started with the BA model is the preferential attachment, which is similar to real-world scenarios. The power-law degree distributions represent the asymmetry in the network.

After initializing, the SocialDynamicSimulation mimics human interactions: discussing random topics and their opinions get closer, and their weight of connections changed. In my model, there are three different topics to discuss and their influences are also different (e.g., politics are generally harder to change opinions than of sports). There are new connections created with 0.05 probability and the new weight is 1-sigmoid(opinion difference) formula. Each node in the original model has a unique parameter (persuasiveness, open-mindedness, etc.) to represent individual differences. Finally, the model parameters are set to create critical conditions for the social system. The details of this model are similar to my previous paper (Tran, 2020). The details are in appendix A. The focus of this proposal is the Covid-19 simulation.

2.2. Covid-19

The pandemics model runs on the network created by SocialDynamicSimulation above. The Covid19 model has many parameters, but this analysis will focus on three main parameters: social distancing, second-order track, and mass testing.

2.2.1. Default interactions

Step 1: We create a dynamic social network of 2000 nodes. This network has a scale-free initialization and has many subclusters for topic preferences. (results from Assignment 3 (Tran, 2020)). *model* is the network created from the SocialDynamicSimulation
Parameter: *self.graph = model .graph*
Step 2: We randomize picked 2 nodes to be infected. The reason is that for regions (differ from Wuhan), viruses cannot appear by themselves. The virus came from external sources (through travel). I am assuming one person got infected and spread to at least one more person he shared the flight with. Randomly chosen two people means that these people do not need a connection to be able to spread as they are on a plane together.
Parameter: *num_new = 2*

Step 3: Then I let people interact with each other. The people will interact with 20% of the closest people in their network. This model representation is better than random interaction within its network, which is not realistic. People usually interact with those they are closed with, not randomly. The closest is defined as those with the strongest weights. I sorted the nodes by its connection strength. I chose the top 20% of the sorted neighbors to interact with the selected node.
Parameter: *neighbor_rate = 0.2*

Step 4: To add randomness to the interactions, unless the person is in quarantine, then they will have 0.1 chance of meeting random people in their network.
Parameter: *random_interaction = 0.1*

Step 5: If a person interacts with an infected person, they will have a 0.4 chance of being infected.
Parameter: *infect_rate = 0.4*

Step 6: When being infected, each person is assigned a unique immune system. The immune system is what determines how long until they show symptoms. This parameter represents the trickiness of Covid-19 is that people can be infected and not known until they actually show symptoms. The immune system of each person is sampled from a normal distribution.
Parameter:

- The mean of the distribution: *days_symp = 7*
- The standard deviation of the distribution: *days_symp_std = 2*
- *node[immune] ~ Normal(days_synp, days_symp_std)*

The immune system is like a credit system. They will get subtracted day by day, and when it reaches 0, the disease shows symptoms. Then these people go to the hospital.

Step 7: However, this virus shows symptoms of common flu. Hence, there are chances the symptoms are not severe enough to test for coronavirus. This leads to the hospital admission rate, that even if you start having symptoms, you have a certain probability of being into the hospital (as known as being detected for coronavirus). However, the person goes back to the hospital every day after developing symptoms, which will increase the chance of being detected.
Parameter: *hospital_admit = 0.6.*

After *n* days after the symptoms start, the person will have $1 - 0.4^n$ chance of being admitted. This mechanism guarantees that the more days the symptoms show, the more severe it gets to make this person more likely to be admitted.
Step 8: If the person is admitted (detected for coronavirus), they will have to cut all connections to its network. Also, 0.8 of their network will be notified and put into the self-quarantine mode.
Parameter: *quarantine_noti = 0.8*

Step 9: If a person is in quarantine, they can only interact with the two closest people, and no random interaction is allowed. With all careful precautions, it will have a 0.05 chance of infecting the two closest people they live with.
Parameter: *quarantine_infect_rate = 0.05.*

2.2.2. Policymakers parameters:
3 components: social distancing, second-order track, and mass testing

2.2.2.a. Social distancing
Social distancing (True/False): If social distancing is True, then the social distancing policies are implemented.
Parameter: *social_distancing = True*

Social distancing implies the population will only interact with 1 or 2 closest people to them. The closest people are determined by the strongest weights connected to the node.
To add more realistic components, I added disobedience of people and lag in policy time:

- For disobedience, I set 0.1 chance of people interacting with another one person in their network (in addition to 1-2 closest people above).
  Parameter: *random_interaction = 0.1*

- For lag in policy time: the implication is that no country can or would do social distancing before a case is detected. Also, policymakers cannot act that fast. Hence, the social distancing policy is only implemented after the dates of the first case detected plus the time needed to set the policy. In this model, I set the policy of social distancing (if True) to be up after2 days of the first detected case.
  Parameter: *setting_social_distacing = 2*

2.2.2.b. Second-order track
Second-order track (True/False): This approach is usually not phrased as a policy, but I believe it is important to consider. For coronavirus, it is important to trace back to who they interacted with as they have a high chance of being affected. The people they interacted with are F1. The second-order track means we also consider the people that interacted with F1 (called F2). By default, the government will trace back to F1 and put them into quarantine with 0.8 of success rate (as we account for disobedience and unexpected, unknown interactions). The details about quarantine are discussed above.

If the second-order track is True, the policy implies tracing back to the F2 and put them into quarantine with 0.4 of success rate. The quarantine behavior is similar to what described above.
Parameter: *quarantine_noti_2 = 0.4*

Similar to the above, the policy cannot be implemented before or immediately after any case is detected. There must be a time lag for policymakers. I am generous and let this time lag to be one day (as this policy is internal and easier to set up than of social distancing).
Parameter: *setting_second_track = 1*

2.2.2.c. Mass testing.

Mass testing (True/False): this approach is represented in Korea and Germany. The idea is to mass test people to check if they are infected.
Note: mass testing is different from having test kits to test people if they have symptoms. Those tests are already implied in the hospital procedure. Mass testing means setting up booths on the street to test all people passing through the booths (even those no symptoms appear yet).
Parameter: *mass_testing = True*

The government will mass test randomly 5% of the population that can go outside. The value is fixed because the government is setting booths of testing across the countries (and those traveling through these booths are tested). Hence, only a certain of the population go through these booths and get tested. For further research, we can use simulation to optimize people who got tested by identifying places to set up testing booths.
Parameter: *mass_testing_rate = 0.05*

Also, setting up testing booths on a society scale is not simple. It takes days to set up. In this model, we set it to 6 days after the first case is detected.
Parameters: *setting_mass_test = 6*

There are also false positive and false negative rates for mass testing, but I set these values small to temporarily simplify the model.
Parameter: *false_potive = 0.01, false_negative = 0.01*

2.3. Model evaluation

Overall, this model decently represents the social dynamics in the pandemics time, especially with the policymakers interventions. There is randomness across the parameters to demonstrate disobedience nature of human and non-deterministic results of the world. Furthermore, this model accounts for time lag within policies application and more appropriate network interaction (with the closest people). Also, the model accounts for individual differences like the immune system (time to show coronavirus symptoms). Finally, this model has many states that each person can be in, including infected, quarantined, and hospitalized (detected). These are all real-world characteristics. However, as this is only a simulation, it relies on many assumptions and consists of many limitations. We will discuss this aspect in the latter part.

Part 3: Assumptions & Limitations

The primary assumption is that all non-primary parameters have to be valid (all settings except the boolean values for mass testing, social distancing, or second-order tracking). We can indeed vary the parameters if we want to, but it will require significant time and computing power to finish running. Furthermore, the model involves randomness across parameters; if any changes happen, we should run at least 30 times for the randomness to balance out. All the parameters that are non-primary (all except the three boolean values) are based on different news articles and other models' parameters. There are no parameters that can perfectly represent what is happening in real life, but I believe these values are valid on average. Different countries will have different parameters (e.g., people do not interact up to 20% of their closest; some countries have better immune systems, etc.). Policymakers can alter the parameters based on their countries' data to achieve better predictions. However, within the scope of this model, we do not alter the non-primary parameters.

Another drawback of this model has too many parameters. The flexibility of having many parameters is higher, but to get meaningful insights about the interactions between parameters are very computationally heavy. For example, for three different boolean parameters (mass testing, second-order track, and social distancing), I need eight models to run. Not to mention that these models depend on the initial state and have many randomnesses. Hence, we have to run at least 30 iterations to balance out the random component to have a more holistic result. These eight models with 30 iterations take 5 hours to run on Google Colab GPU successfully. If the users want to explore more, especially on the aggressiveness of the infection rate or other interventions, they should open multiple computers to run separate models altogether.

Another limitation has too much randomness in the model. As randomness appears across models, the final results accumulate the randomness, leading to a very big confidence interval. Randomness is a good aspect to represent uncertainty. For future trials, I suggest running for more time to achieve more certainty. Also, models are a simplification of the real world. Some events cannot be accounted for in these models, like random gathering for religious purposes in Korea, or mass travel from Italy to other countries before its lockdown. These external factors are contextual, dependent, and unpredictable. This model can best account for potential personal differences and policy effectiveness. Components like essential services workers' interactions with the normal population should also be evaluated. Within the scope of this project, these parameters are sufficient. For future exploration, I recommend varying more parameters and having specific special interactions like hospitals or supermarkets.

One final note is that this model is not to predict what percentage of the population will be infected. The primary goal of this model is to compare what policy is best to counter the contagiousness of coronavirus. Hence, even though the validity of parameters is still a concern, its impact on the conclusion to compare policies is acceptable.

Part 4: Results analysis

4.1. Model comparisons

	Social distancing	2nd track	Mass testing	Mean	Median	2.5th percentile	97.5% percentile	Ranking
Group 0	False	False	False	0.492567	0.503	0.393825	0.564325	8
Group 1	False	False	True	0.489517	0.49325	0.368363	0.575625	7
Group 2	False	True	False	0.227217	0.2145	0.099375	0.371837	4
Group 3	True	False	False	0.265417	0.2365	0.088525	0.454487	6
Group 4	False	True	True	0.193567	0.209	0.0142375	0.335762	3
Group 5	True	False	True	0.263967	0.25075	0.051475	0.529237	5
Group 6	True	True	False	0.152633	0.14325	0.0320875	0.325475	2
Group 7	True	True	True	0.147633	0.12375	0.018925	0.337075	1

Figure 1: The table represents the percentage of the population infected by the coronavirus and the ranking of effectiveness of the intervention.

	Group 0	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7
Group 0	----	0.814840724847	0.000000000000	0.000000000000	0.000000000000	0.000000000018	0.000000000000	0.000000000000
Group 1	0.814840724847	----	0.000000000000	0.000000000000	0.000000000000	0.000000000031	0.000000000000	0.000000000000
Group 2	0.000000000000	0.000000000000	----	0.151215585594	0.155539991571	0.217353408761	0.000997943925	0.001463588011
Group 3	0.000000000000	0.000000000000	0.151215585594	----	0.016609770345	0.966330277406	0.000139532732	0.000184213742
Group 4	0.000000000000	0.000000000000	0.155539991571	0.016609770345	----	0.032036630620	0.105977017199	0.093569477989
Group 5	0.000000000018	0.000000000031	0.217353408761	0.966330277406	0.032036630620	----	0.000608512899	0.000671195340
Group 6	0.000000000000	0.000000000000	0.000997943925	0.000139532732	0.105977017199	0.000608512899	----	0.844233049232
Group 7	0.000000000000	0.000000000000	0.001463588011	0.000184213742	0.093569477989	0.000671195340	0.844233049232	----

Figure 2: The table represents the t-test significance between groups.

There are four larger categorizations: no intervention, one intervention, two interventions, and three (all) interventions.

The no-intervention group led to most people getting infected because no active interventions were performed.

Among the one intervention groups (groups 1, 2, 3), the most effective is 2nd order tracking, then the social distancing, and then mass testing.

Interestingly, mass testing (group 1) gives slightly better results than doing nothing (group 0), but there is no significant difference between the groups. **Our first conclusion is that mass testing should not be the single policy to counter coronavirus.** This result is reasonable because mass testing can only be implemented later than other measurements, and the random selection of mass testing cannot exceed the spread of the virus. However, it still contributes some value as it detects some existing cases earlier than waiting for symptoms.

Another finding is that, for a single intervention, second-order tracking is more effective than social distancing. This result is surprising because all newspapers are talking about the lockdown as the best policy, and little to mention the tracking of F2 patients. The difference is not significant, but large enough to raise awareness on the considerations for quarantining F2. The mean difference is 0.07 less infected rate for 2nd-order tracking, with a t-test significance of 0.15. These values are decently high to represent effectiveness. This result is reasonable because of the nature of the virus:

it spreads tremendously fast at the beginning. Hence, if we lockdown, the existing cases will still infect, at least, the people in their household and randomly some of the neighborhood. However, if we can track who is infected and trace to the F2, it creates a feedback loop of tracing viruses. If any of the F1 or F2 is infected, we can further trace to F3 and F4 (of the original case), which can help us find all infected cases in the community faster. **Our 2nd conclusion is if resources allowed, track the F2 of the infected patients. It is shown to be most effective among one-intervention groups.** Some countries are implementing 2nd-order tracking and have tremendously good results. One proud example is Vietnam. We are neighboring China, but we only have 270 cases even when our first case was far back on January 23rd. With this policy, we almost declared virus-free in late February (but then some UK foreigners came, denied quarantining, and started traveling, which brings us back to square 0). However, afterward, we make the policies more strict, denying foreigners coming in, and we have no additional corona cases for at least eight days so far. We will be declaring corona free after next week.

For the two-intervention groups, interestingly, the combination of mass testing and social distancing is still worse than 2nd-order tracking alone, even though not as significant as 2nd-order tracking compared to one-intervention groups. This approach further signifies the effectiveness of F2 tracking.

Within the two-intervention groups, as expected, the best is the combination of the two most effective approaches: social distancing and 2nd-order tracking. **Our next conclusion is, with resources for two interventions, social distancing and 2nd-order tracking are the best combinations.**

What is interesting is the comparison between group 4 (mass-testing + 2nd-order tracking) and group 5 (mass-testing + social distancing). Group 4 has better effectiveness than group 5 (0.07 less infected in mean and 0.032 t-test significance). 0.032 for a significance level is very low (and nearly significant). This result is the story of South Korea. Many consider South Korea's amazing job flattening the curve (they have flattened the curve much faster than any country (Rebecca, 2020)) is on their mass testing strategy. However, people are ignoring the fact that South Korea has one of the most advanced CCTV and phone tracking systems in the world (Hamilton, 2020). With 2nd-order tracking and mass testing, South Korea manages to flatten the curve without lockdown (in March). I believe the key to South Korea's success is not the mass testing strategy, but its ability to effectively track people on a large scale. **Our next conclusion is the explanation for the case of Korea, which efficiently flatten the curve quickly by F2 tracking and mass-testing.**

What is further interesting is the effectiveness of mass-testing combined with other methods. For group 3 (social distancing) and group 5 (social distancing + mass-testing), there is no significance between the two (0.0015 mean difference, and 0.966 t-test significance). However, for group 2 (and tracking) and group 4 (2nd track + mass testing), the mean difference is 0.03 and 0.15 significance value (which is not statistically significant but in the high significance tiers). Mass-testing helps F2 tracking more than helping social distancing. It might be that for social distancing is set, the infected cases already lead to local neighborhood infection, and not spreading around randomly anymore

(due to social distancing). Hence, mass testing only helps to faster detect the cases but not helping to stop the spread (because social distancing stops the spread already). **Our next conclusion is that mass testing should be applied when the lockdown is lifted, and combined with F2 tracking to maximize its benefits. Mass testing does not really help if social distancing is in place.**

Finally, as predicted, having all interventions yield better results than any other method. It has similar results to group 6 (social distancing + F2 tracking) because mass testing is not that helpful when social distancing is in place. It has a high difference compared to group 4 (mass testing + F2 tracking), but not statistically significant (0.09). This significance level is enough to say it is better than group 4. The all-interventions group is statistically more effective than any remaining groups. **Another conclusion is that, if resources are allowed, applying all interventions is the best strategy. However, social distancing and F2 tracking can yield similar results (if testing resources are limited).**

One cautious component is that our model is based on a 2000-agents network. Hence, the result might not be entirely representative of the whole country with a complex network structure. Similarly, F2 tracking on a small scale (2000 nodes) is more accessible than applying on a country level if the existing infrastructure does not support it. South Korea is an exception as its infrastructure of CCTV and phone networks have already matured. Vietnam is another case study as we track the cases from the very beginning and try to control it as much as we can. That is why our results are still on a small scale. If mass infection happens in Vietnam, F2 tracking might not be useful as we do not have the infrastructure to support those implementations on a large scale.

4.2. Results Uncertainty

Results analysis: As I mentioned in the limitations, the fact of having many randomnesses within the model leads to a high level of uncertainty, even when I ran for 30 iterations for each policy. Also, the result is highly dependent on who has the virus at the beginning. If a person with a high number of connections is patient 0, they can lead to mass infection. In contrast, if a person with a low number of connections can lead to less severe results. As there is so much uncertainty, the confidence interval is substantial, especially for those applying more than one policy. To counter this uncertainty and better systematically compare the policies, I use t-tests as described above to evaluate the effectiveness of policies. To reduce uncertainty, we can run for 100+ iterations to test the results. 100+ is just a suggestion if computation power allows. Here, we ran for 30 iterations and the interval was still very large.

Below is the histogram of the final results of the infected population level. The majority of them look symmetry. The groups with social distancing especially look right-skewed, which is a good indication for effective interventions. These histograms indicate that social distancing is a proper intervention in general. F2 tracking policy's results are also promising.

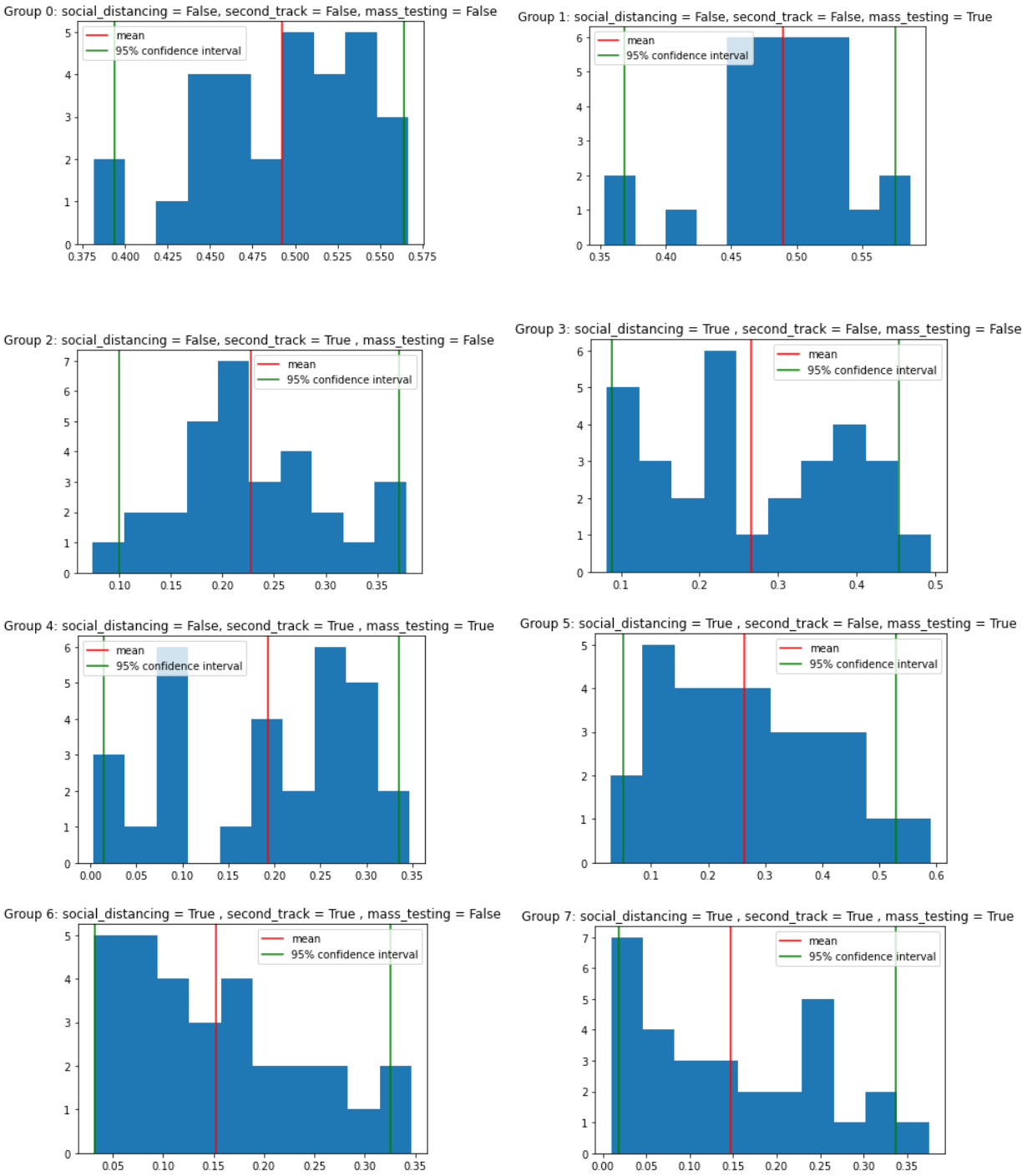


Figure 3: The histogram of the final infection in the population by 8 policies.

The graph represents how the infected rate progress across models are also listed below:

Figure 4: Group 0: social_distancing = False, second_track = False, mass_testing = False

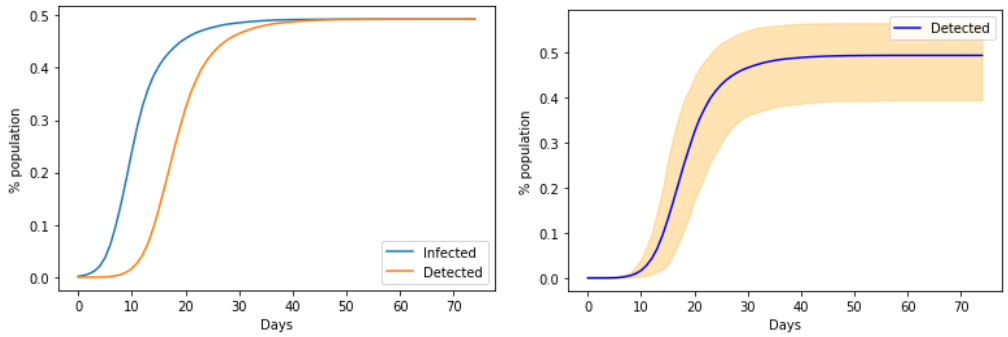


Figure 5: Group 1: social_distancing = False, second_track = False, mass_testing = True

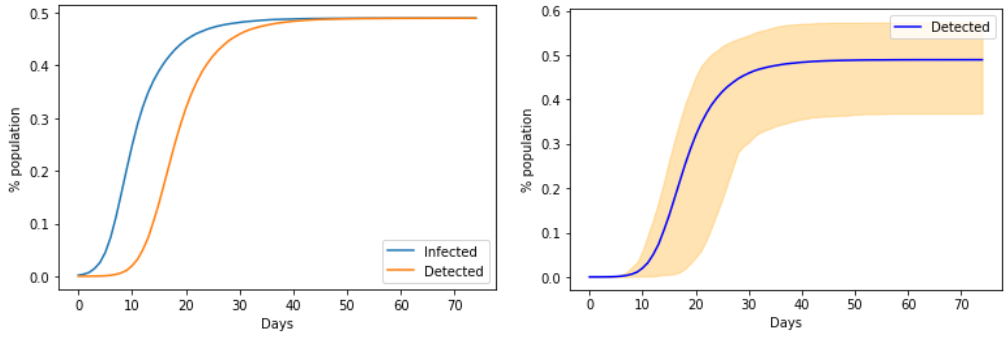


Figure 6: Group 2: social_distancing = False, second_track = True , mass_testing = False

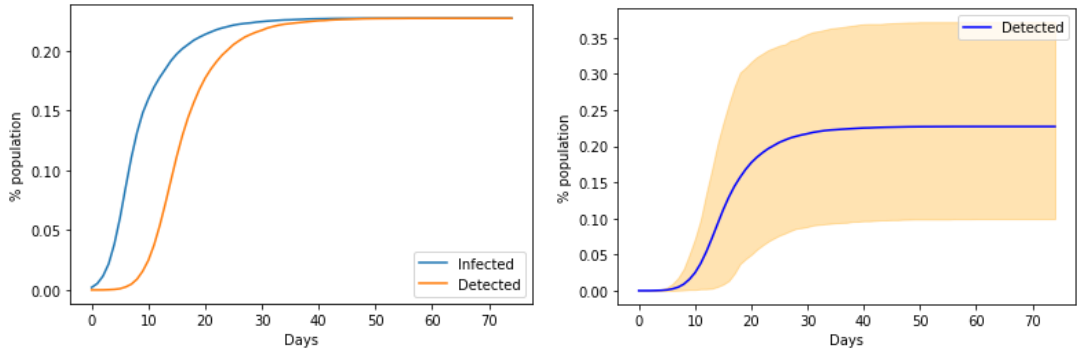


Figure 7: Group 3: social_distancing = True , second_track = False, mass_testing = False

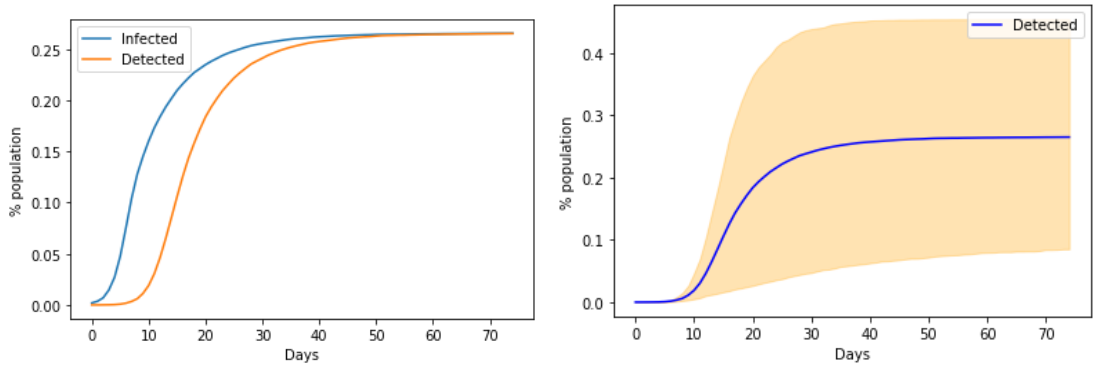


Figure 8: Group 4: social_distancing = False, second_track = True , mass_testing = True

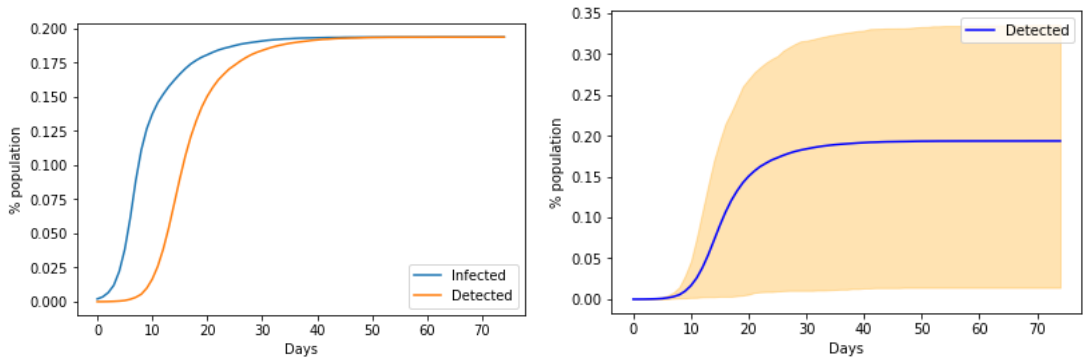


Figure 9: Group 5: social_distancing = True , second_track = False, mass_testing = True

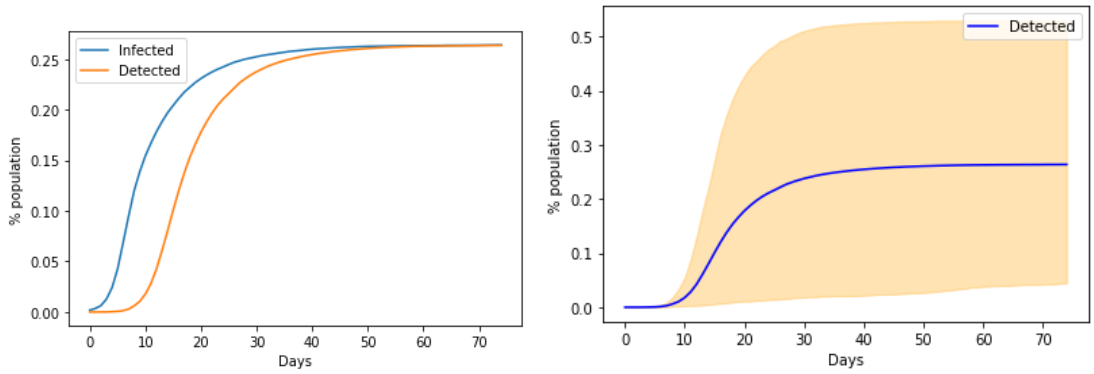


Figure 10: Group 6: social_distancing = True , second_track = True , mass_testing = False

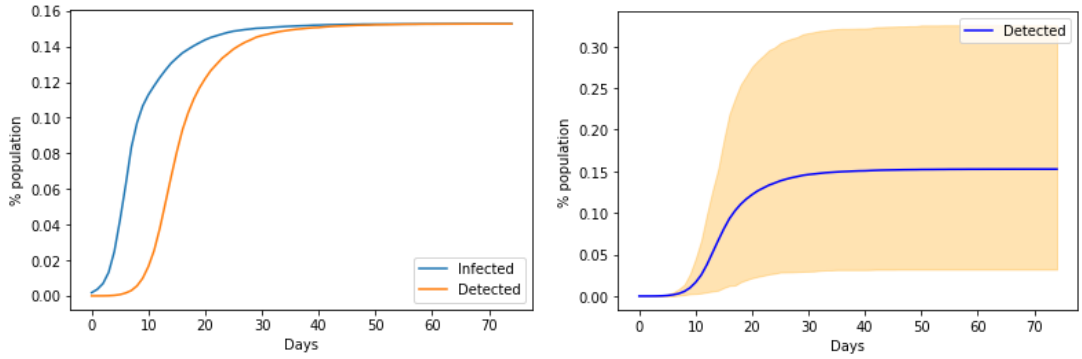
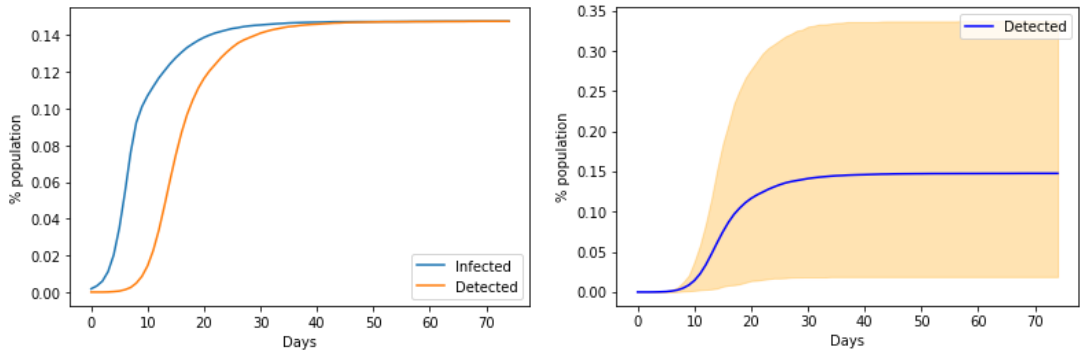


Figure 11: Group 7: social_distancing = True , second_track = True , mass_testing = True



Part 5: Conclusion

The simulations found interesting results and conclusions regarding policies to counter coronavirus: The findings are listed across the results part and are synthesized here:

- Mass testing should not be the single policy to counter coronavirus.
- If resources are allowed, track the F2 of the infected patients. It is shown to be most effective among one-intervention groups. Some countries are implementing
- With resources for two interventions, social distancing and 2nd-order tracking are the best combinations.
- The explanation for the case of Korea, which efficiently flatten the curve quickly by F2 tracking and mass-testing.
- Mass testing should be applied when the lockdown is lifted, and combined with F2 tracking to maximize its benefits. Mass testing does not really help if social distancing is in place.
- If resources are allowed, applying all interventions is the best strategy. However, social distancing and F2 tracking can yield similar results (if testing kits resources are limited).

The model is based upon many parameter assumptions and limitations; however, the suggestions above are valid through analyzing case studies and simulations' mechanisms. More studies should be conducted to extend the findings from these simulations and to apply on large scale networks effectively. For the scope of this project, I conclude the findings, suggestions here with its assumptions and limitations, hoping to inform policymakers how to counter coronavirus.

Part 6: Instruction to run the codes

Folder contains 4 files: 2 .ipynb notebooks and 2 .npy files

- File 1: [Hoang Tran] Covid19 Simulations.ipynb
- File 2: Visualize results of simulations.ipynb
- File 3: CS166 FP tracks hospitals.npy
- File 4: CS166 FP tracks.npy

The simulations are in File 1. You can check this code for some sample trials.
(Note: I suggest not running the whole notebook as it contains multiple trials session, which will take 5 hours to run)

To save time for code reproduction, I saved important values in the 2 .npy files (File 3 and 4). To visualize these data, run File 2. One small note is to change cell [2] under the "Reading the values file" section to the appropriate local directory on your computer.

Part 7: HCs applications:

#systemdynamics: I analyze the effectiveness of intervention policies by mimicking the dynamic of the social systems and humans' behaviors of each agent (human). I develop a simple model where agents can interact and infect viruses, with the consideration for quarantine and social distancing, and consider how it changes the dynamic of the systems.

#interventionalstudy + #controlgroups: I propose 3 interventions (policies) and conduct simulations on the network to evaluate the effectiveness. This project is a good use of #interventionalstudy because I consider how each intervention changes the way people interact and influence the social dynamics. Furthermore, I also consider how the interventions interact with each other (e.g., social distancing reduces the impact of mass testing) and proposes applicable suggestions for policymakers. Also, the use of the #controlgroups is the baseline behaviors when no interventions are applied (group 0). By comparing each intervention with the control groups using t-test as the statistical tool, I conclude which intervention works best to propose effective interventions.

Part 8: Appendix and references

Hamilton, I. A. (2020, April 14). Compulsory selfies and contact-tracing: Authorities everywhere are using smartphones to track the coronavirus, and it's part of a massive increase in global surveillance. Retrieved from https://www.businessinsider.com/countries-tracking-citizens-phones-coronavirus-2020-3

Rebecca, K. K. (2020, March 19). Which Country Has Flattened the Curve for the Coronavirus? Retrieved from https://www.nytimes.com/interactive/2020/03/19/world/coronavirus-flatten-the-curve-countries.html

Tran. (2020), CS166 Social Dynamic Simulation. Retrieved from https://drive.google.com/open?id=1NBV7wXc_JJEzc2uEbgnbmG6DNUDPUqDn

```
#import key packages

from matplotlib import pyplot as plt
import networkx as nx
import random
import numpy as np
random.seed(2021)
import scipy.stats as sts
from scipy.stats import norm
from copy import deepcopy
from tabulate import tabulate
```

COVID-19 Class

```
# The number of days we are simulating for
days = 75

# new class for simulation
class Covid19:
    '''
    Simulate social dynamics by strengthening opinions and connection weights
    based on random interactions between nodes.
    '''

    def __init__(self, network_size = 2000, infect_rate = 0.4, random_interaction = 0.1, \
                 days_symp = 7, days_symp_std = 2, hospital_admit = 0.6, quarantine_noti = 0.8,\
                 quarantine_noti_2 = 0.4, neighbor_rate = 0.2, num_new = 2, second_track = False, \
                 social_distancing = False, quarantine_infect_rate = 0.05, mass_testing = False, \
                 mass_testing_rate = 0.05, false_positive = 0.01, false_negative = 0.01, first_day = np.inf,\
                 setting_mass_test = 6, setting_social_distancing = 2, setting_second_track = 1):
        '''
        Inputs:

                network_size: number of nodes
                infect_rate: the chance of infecting others while interacting
                random_interaction: the chance of a person having random interactions
                days_symp: mean of number of days until a person show symptoms
                days_symp_std: standard deviation of number of days until a person show symptoms
```

```
            days_symp_std: standard deviation of number of days until a person show symptoms
            hospital_admit: the probability of being admitted to the hospital with symptoms
            quarantine_noti: % of F1 got quarantine
            quarantine_noti_2: % of F2 got quarantine
            neighbor_rate: how many people a node interact with
            num_new: initial cases
            second_track: boolean: if F2 tracking is implemented
            quarantine_infect_rate: change of infecting others while being in quarantine
            social_distancing: boolean: if social distancing is implemented
            false_positive: testing false positive rate
            false_negative: testing false negative rate
            mass_testing: boolean: is mass testing is implemented
            mass_testing_rate: % of population got tested randomly
            first_day: track the first day a case is detected
            setting_mass_test: time to set up mass testing
            setting_social_distancing: time to set up social distancing
            setting_second_track: time to set up F2 tracking

        '''
        self.network_size = network_size
        self.infect_rate = infect_rate
        self.random_interaction = random_interaction
        self.days_symp = days_symp
        self.days_symp_std = days_symp_std
        self.hospital_admit = hospital_admit
        self.quarantine_noti = quarantine_noti
        self.quarantine_noti_2 = quarantine_noti_2
        # how many friends u contact with in your neighbor
        self.neighbor_rate = neighbor_rate
        self.num_new = num_new
        self.second_track = second_track
        self.quarantine_infect_rate = quarantine_infect_rate
        self.social_distancing = social_distancing
        self.false_positive = false_positive
        self.false_negative = false_negative
        self.mass_testing = mass_testing
        self.mass_testing_rate = mass_testing_rate
        self.first_day = first_day
        self.setting_mass_test = setting_mass_test
        self.setting_social_distancing = setting_social_distancing
        self.setting_second_track = setting_second_track

```

```
66
67      def initialize(self, sim):
68          # copy the network from the SocialDynamicSimulation
69          self.graph = deepcopy(sim.graph)
70          for node in self.graph.nodes:
71            # infected or not
72            self.graph.nodes[node]['status'] = 0
73            # the value of personal immune system
74            self.graph.nodes[node]['immune'] = round(np.random.normal(self.days_symp, self.days_symp_std))
75            # =1 if detected with corona virus
76            self.graph.nodes[node]['hospital'] = 0
77            # =1 if quarantine
78            self.graph.nodes[node]["quarantine"] = 0
79
80          # getting random nodes to start the pandemics
81          affected = random.sample(list(self.graph.nodes), self.num_new)
82
83          # having a set to track who is infected
84          self.list_affected = set(affected)
85          # a set to track who is hospitalized
86          self.list_hospital = set()
87          # a set of all nodes
88          self.all_nodes = set([n for n in self.graph.nodes])
89
90          # start the pandemic
91          for affect in affected:
92            self.graph.nodes[affect]['status'] = 1
93          self.layout = nx.spring_layout(self.graph)  # Initial visual layout
94          self.step = 0
95
96
97      def update(self):
98        # if mass testing and the set up time is done:
99        # self.first_day + self.setting_mass_test is the time to prepare mass testing
100       if self.mass_testing == True and self.step > self.first_day + self.setting_mass_test:
101         # only test those not hospitalized
102         remain = np.setdiff1d(self.list_hospital, self.all_nodes)
103
104         # get random number of population to be tested
105         test = random.sample(list(remain), int(self.mass_testing_rate*len(remain)))
106
```

```
        # analyze results of tests
        for tested in test:
          # infected -> hospitalized
          if self.graph.nodes[tested]["infected"] == 1:
            self.graph.nodes[tested]["hospital"] = 1
            # false negative rate
            if random.random() < false_negative:
              self.graph.nodes[tested]["hospital"] = 0
              pass
            # if hospitalized then quarantine
            self.graph.nodes[tested]['quarantine'] = 1

          # if not infected
          else:
            # if false positive -> hospitalized
            if random.random() < false_positive:
              self.graph.nodes[tested]["hospital"] = 1
              self.graph.nodes[tested]['quarantine'] = 1

      # new is the set for newly infected case
      new = set()
      for node in self.list_affected:
        # immune credit decreases everyday: count down till symptoms
        self.graph.nodes[node]['immune'] -= 1

        # symptoms show and start going to hospital
        if (self.graph.nodes[node]['immune'] <= 0 and random.random() < self.hospital_admit) or self.graph.nodes[node]['hospital']
          # admitted
          self.graph.nodes[node]['hospital'] = 1

          # check if this is first case -> assign the first day a new value
          if len(self.list_hospital) == 0:
            self.first_day = self.step

          # list of hospitalized
          self.list_hospital.add(node)
          self.graph.nodes[node]['quarantine'] = 1

          # cut all connections and the F1 has "quarantine_noti = .8" chance to be quarantined
          neighbor_list = [n for n in self.graph.neighbors(node)]
          for neighbor in neighbor_list:
```

```
      for neighbor in neighbor_list:
        # cut connections
        self.graph.remove_edge(node, neighbor)

        # F1 has "quarantine_noti = .8" chance to be quarantined
        if random.random() < self.quarantine_noti:
          self.graph.nodes[neighbor]["quarantine"] = 1

          # if F2 track and the set up time is done:
          # self.first_day + self.setting_second_track is the time to prepare F2 track
          if self.second_track == True and self.step > self.first_day + self.setting_second_track :
            neighbor_of_neighbor = [n for n in self.graph.neighbors(neighbor)]
            # F2 has "quarantine_noti = .4" chance to be quarantined
            for neighbor_2 in neighbor_of_neighbor:
              if random.random() < self.quarantine_noti_2:
                self.graph.nodes[neighbor_2]["quarantine"] = 1
    # if hospitalized -> no more interactions
    pass


  # get the weight for the edge --> only contact with the top 10% highly weighted + and sometime random people
  neighbor_list = [n for n in self.graph.neighbors(node)]
  neighbor_weight = []
  for neighbor in neighbor_list:
    neighbor_weight.append(self.graph.get_edge_data(node, neighbor)["weight"])

  # sort the weight
  sorted_neighbor_list = [x for _,x in sorted(zip(neighbor_weight, neighbor_list))]

  # if quarantine, then only take top 2 closest
  if self.graph.nodes[node]["quarantine"] == 1 and sorted_neighbor_list != []:
    currents = sorted_neighbor_list[-2:]
    for current in currents:
      # infect rate if in quarantine (=0.05)
      if random.random() < self.quarantine_infect_rate and self.graph.nodes[current]['status'] != 1:
        self.graph.nodes[current]['status'] = 1
        new.add(current)


  else:
    # if social distancing and the set up time is done:
    # self.first_day + self.setting_social_distancing is the time to prepare social distancing
```

```
        # self.first_day + self.setting_social_distancing is the time to prepare social distancing
        if self.social_distancing == True and self.step > self.first_day + self.setting_social_distancing:
          # only interact with the closest people
          people = random.randint(1, 2)
          currents = sorted_neighbor_list[-people:]

        # if not social distancing
        else:
          # interact with "neighbor_rate = 0.2" of its connections
          currents = sorted_neighbor_list[int(np.floor((1-self.neighbor_rate)*len(neighbor_list))):]

        # random interactions
        if random.random() < self.random_interaction and sorted_neighbor_list != []:
          currents.append(random.sample(sorted_neighbor_list, 1)[0])

        # for those interacted with the infected people, having a infect_rate = 0.4 chance of being infected
        for current in currents:
          # being infected: having a infect_rate = 0.4 chance of being infected
          if random.random() < self.infect_rate and self.graph.nodes[current]['status'] != 1:
            self.graph.nodes[current]['status'] = 1
            new.add(current)

    # update the infected list
    self.list_affected = self.list_affected.union(new)
    self.step += 1

  def observe(self):
    '''
    Draw the state of the network with which topic each node is most prefer
    '''
    self.layout = nx.spring_layout(self.graph, pos = self.layout, iterations=5)
    plt.figure(figsize=(10,10))
    plt.clf()
    # all the nodes and get the infected status
    nx.draw(
        self.graph, pos=self.layout, with_labels=True,
        node_color = [self.graph.nodes[i]["status"] for i in self.graph.nodes],
        edge_color = [self.graph.edges[i, j]['weight'] for i, j in self.graph.edges],
        edge_cmap=plt.cm.binary, edge_vmin=0, edge_vmax=1,
        alpha=0.7, vmin=0, vmax=1)
    plt.title('Step: ' + str(self.step))
    plt.show()
```

```
      plt.show()

    # number of infected cases
    def calculate(self):
      return np.mean([self.graph.nodes[i]["status"] for i in self.graph.nodes])

    # number of detected cases
    def calculate_hospital(self):
      return np.mean([self.graph.nodes[i]["hospital"] for i in self.graph.nodes])
```

Social Dynamic Class

```
class SocialDynamicsSimulation:
    '''
    Simulate social dynamics by strengthening opinions and connection weights
    based on random interactions between nodes.
    '''

    def __init__(self, network_size=2000, alpha=0.05, beta=0.3, gamma=4, alpha_std = 0.01, beta_std=0.05, gamma_std=1, num_opinion
        '''
        Inputs:

            network_size (int) The number of nodes in the random Watts-Strogatz
              small-world network. Default: 50.

            alpha (float) The rate at which nodes adjust their opinions to
              match neighboring nodes' opinions during interactions.
              Default: 0.05. This has a standard deviation: alpha_std. Default: 0.01

            beta (float) The rate at which edge weights are changed in
              response to differing opinions. Default: 0.3.
               This has a standard deviation: abeta_std. Default: 0.05

            gamma (float) The pickiness of nodes. Nodes with opinions differing
              by more than 1/gamma will result in an edge weight decreasing.
              Default: 4.
               This has a standard deviation: alpha_std. Default: 1

            num_opinion: number of opinions (integer)
```

```
        '''
        self.network_size = network_size
        self.alpha = alpha
        self.beta = beta
        self.gamma = gamma
        self.alpha_std = alpha_std
        self.beta_std = beta_std
        self.gamma_std = gamma_std
        self.num_opinion = num_opinion

    def initialize(self):
        '''
        Initialize the simulation with a random graph, with random 0 or 1
        opinions assigned to all nodes and initial edge weights of 0.5.
        '''
        self.graph = nx.barabasi_albert_graph(self.network_size, 5)
        for edge in self.graph.edges:
            self.graph.edges[edge]['weight'] = 0.5
            # generate random values from a normal distribution
            self.graph.edges[edge]['beta'] = norm.rvs(self.beta, self.beta_std, size = 1)[0]
            self.graph.edges[edge]['gamma'] = norm.rvs(self.gamma, self.gamma_std, size = 1)[0]
        for node in self.graph.nodes:
            # an array of opinion values
            self.graph.nodes[node]['opinion'] = [random.randint(0, 1) for _ in range(self.num_opinion)]
            # generate random values from a normal distribution
            self.graph.nodes[node]['alpha'] = norm.rvs(self.alpha, self.alpha_std, size = self.num_opinion)
        self.layout = nx.spring_layout(self.graph)  # Initial visual layout
        self.step = 0

    def observe_fav_topic(self):
        '''
        Draw the state of the network with which topic each node is most prefer
        '''
        self.layout = nx.spring_layout(self.graph, pos = self.layout, iterations=5)
        plt.clf()
        # all the nodes and get the max opinion
        master_list = np.array([list(self.graph.nodes[i]['opinion']).index(max(self.graph.nodes[i]['opinion'])) for i in self.graph
        nx.draw(
            self.graph, pos=self.layout, with_labels=True,
            node_color = master_list,
            edge_color=[self.graph.edges[i, j]['weight'] for i, j in self.graph.edges]
```

```
            edge_color=[self.graph.edges[i, j]['weight'] for i, j in self.graph.edges],
            edge_cmap=plt.cm.binary, edge_vmin=0, edge_vmax=1,
            alpha=0.7, vmin=0, vmax=self.num_opinion - 1)
        plt.title('Favorite topic discussion. Step: ' + str(self.step))

        # print the percentage preference
        for i in range(self.num_opinion):
          print("% likes topic", i, ":", np.mean(master_list == i)*100)
        plt.show()

    def observe_topic(self, topic):
        '''
        Draw the state of the network for each topic
        '''
        self.layout = nx.spring_layout(self.graph, pos = self.layout, iterations=5)
        plt.clf()
        nx.draw(
            self.graph, pos=self.layout, with_labels=True,
            # get the topic value for each node
            node_color=[self.graph.nodes[i]['opinion'][topic] for i in self.graph.nodes],
            edge_color=[self.graph.edges[i, j]['weight'] for i, j in self.graph.edges],
            edge_cmap=plt.cm.binary, edge_vmin=0, edge_vmax=1,
            alpha=0.7, vmin=0, vmax=1)
        plt.title('Topic ' + str(topic) +  '. Step: ' + str(self.step))
        plt.show()

    def update(self):
        if random.uniform(0, 1) < 0.05:
            # Create a new edge with 1-sigmoid(opinion difference in selected topic)
            nodes = list(self.graph.nodes)
            while True:
                new_edge = random.sample(nodes, 2)
                if new_edge not in self.graph.edges:
                    break
            opinions = [self.graph.nodes[n]['opinion'] for n in new_edge]

            # choose random topic
            topic = random.randint(0, self.num_opinion - 1)
            new_weight = abs(np.array(opinions[0]) - np.array(opinions[1]))[topic]
            new_weight = 1 - (1/(1 + np.exp(-new_weight)))
            self.graph.add_edge(new_edge[0], new_edge[1], weight = new_weight,
```

```
                                beta = norm.rvs(self.beta, self.beta_std, size = 1)[0],
                                gamma = norm.rvs(self.gamma, self.gamma_std, size = 1)[0])

        else:
            # Select a random edge and update node opinions and edge weight
            edge = random.choice(list(self.graph.edges))
            weight = self.graph.edges[edge]['weight']
            opinions = [self.graph.nodes[n]['opinion'] for n in edge]
            # select topic
            topic = random.randint(0, self.num_opinion - 1)
            # conservativeness of the topic
            topic_pref = 10**(-topic)
            for i in [0, 1]:
                # update the node with the topic_preference value
                self.graph.nodes[edge[i]]['opinion'][topic] = (
                    opinions[i][topic] + topic_pref * self.graph.nodes[edge[i]]['alpha'][topic] * weight * (opinions[1-i][topic] -

            self.graph.edges[edge]['weight'] = (
                weight +
                self.graph.edges[edge]['beta'] * weight * (1-weight) *
                (1 - self.graph.edges[edge]['gamma'] * abs(opinions[0][topic] - opinions[1][topic])))
            # Remove very weak connections
            if self.graph.edges[edge]['weight'] < 0.05:
                self.graph.remove_edge(*edge)
        self.step += 1
```

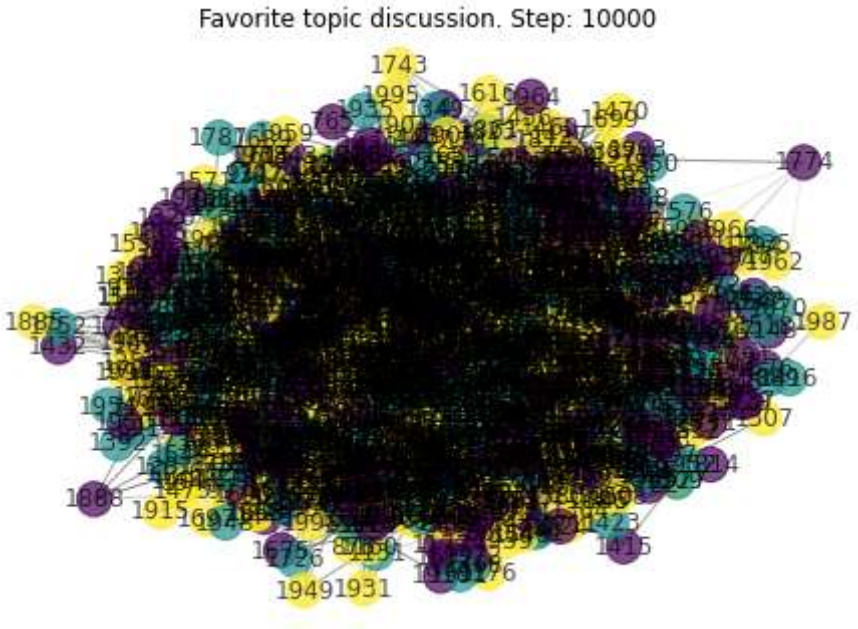
Test the model on single simulation

```
# function to run the Covid simulations for 75 days and calculate the infect and detected cases
def run_sim(sim, model, days = days, plot = False):
    sim.initialize(model)
    if plot:
      sim.observe()
    track = []
    track_hospital = []
    for k in range(days):
      sim.update()
      track.append(sim.calculate())
      track_hospital.append(sim.calculate_hospital())
    if plot:
```

```
    if plot:
      sim.observe()
    return track, track_hospital
```

```
# simulate: initialize the network for the covid19
model = SocialDynamicsSimulation()
model.initialize()

for i in range(10000):
  model.update()
model.observe_fav_topic()
```

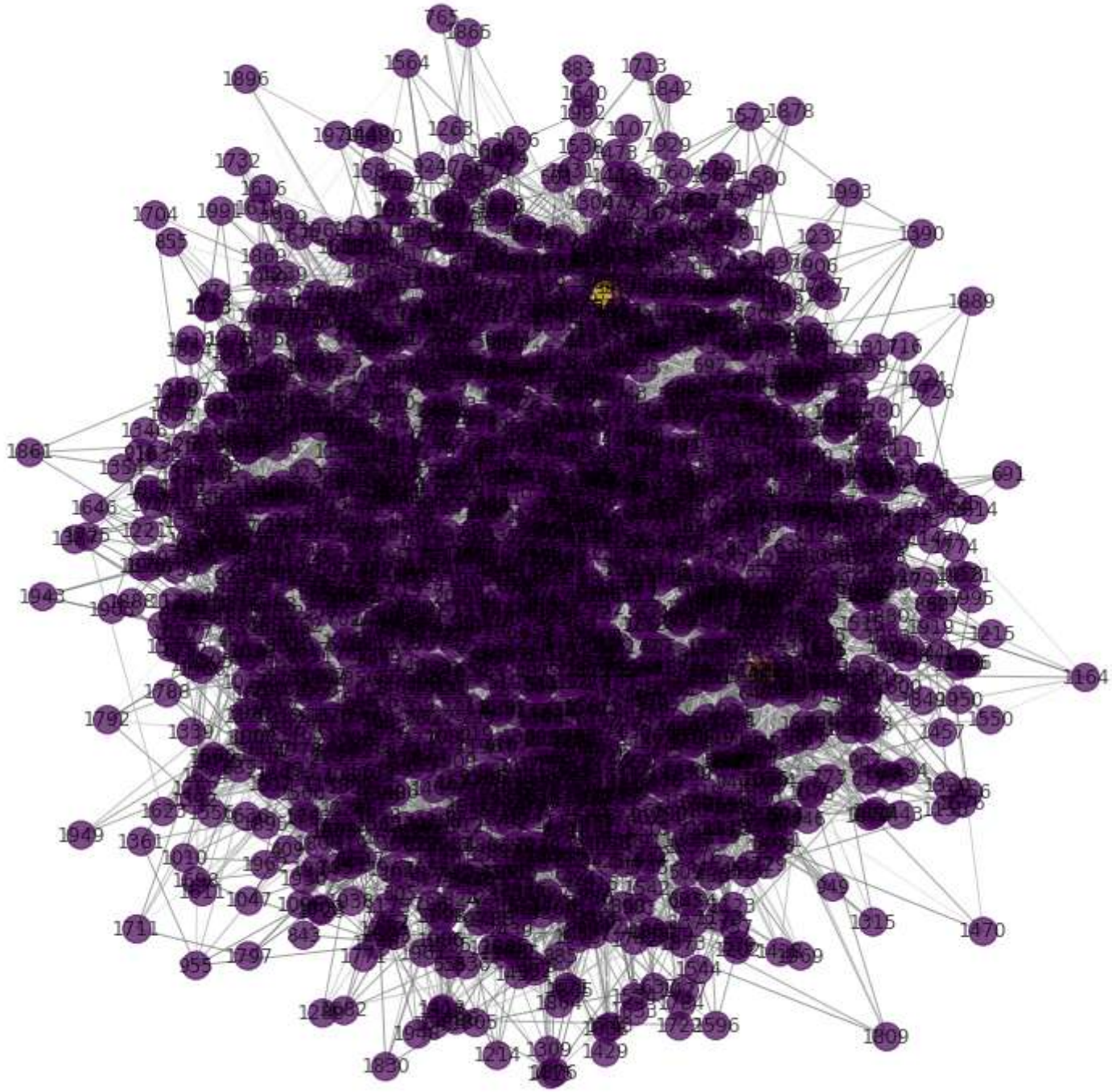
```
% likes topic 0 : 31.45
% likes topic 1 : 29.15
% likes topic 2 : 39.4
```



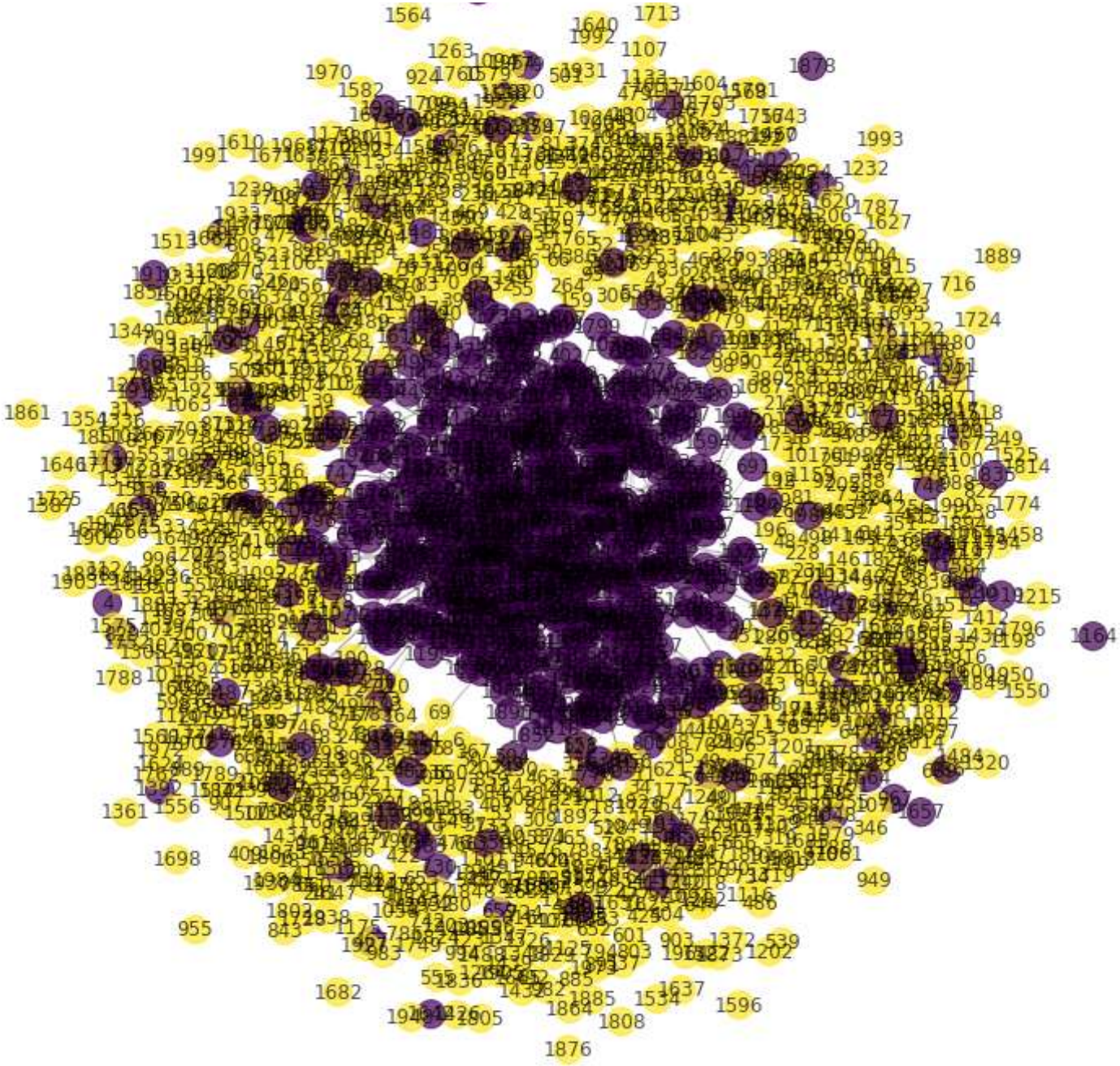
Group 0: social_distancing = False, second_track = False, mass_testing = False

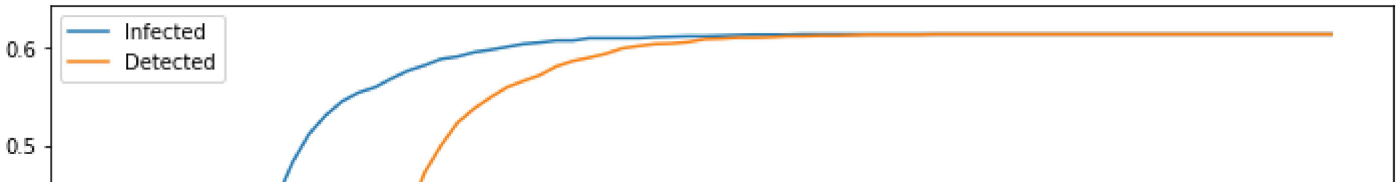
```
# simulate the covid 19 spread and track its progress
print("social_distancing = False, second_track = False, mass_testing = False")
sim = Covid19(social_distancing = False, second_track = False, mass_testing = False)
track, track_hospital = run_sim(sim, model, plot = True)

```

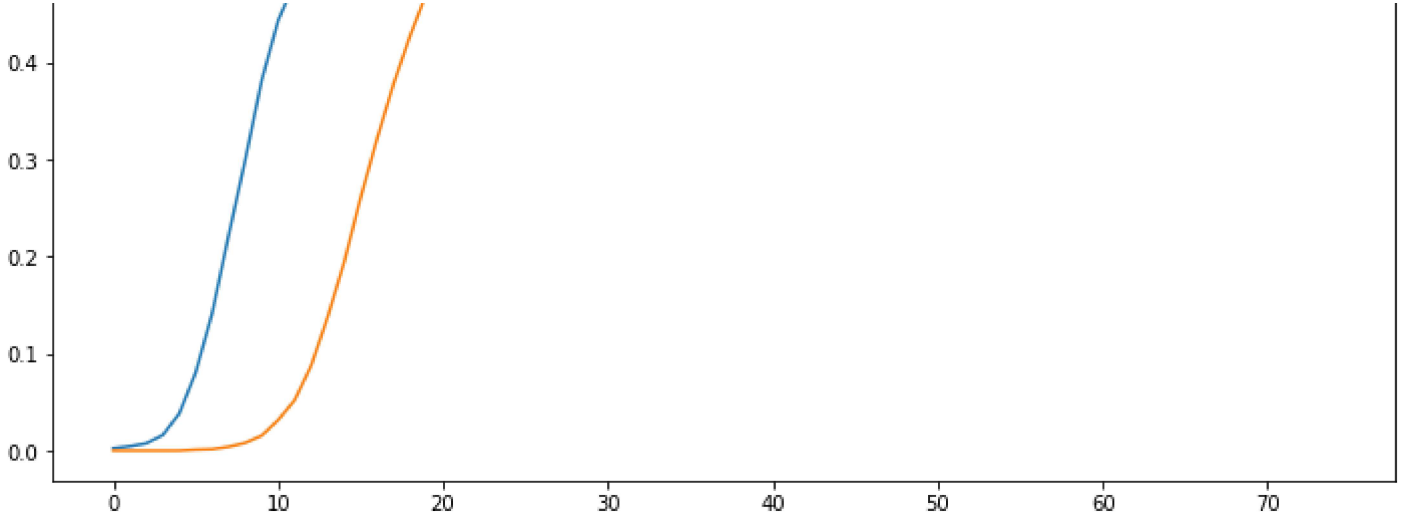
```
plt.figure(figsize = (12,6))
plt.plot(range(days), track, label = "Infected")
plt.plot(range(days), track_hospital, label = "Detected")
plt.legend()
plt.show()
```



Step: 75




Infected
Detected
0.6
0.5

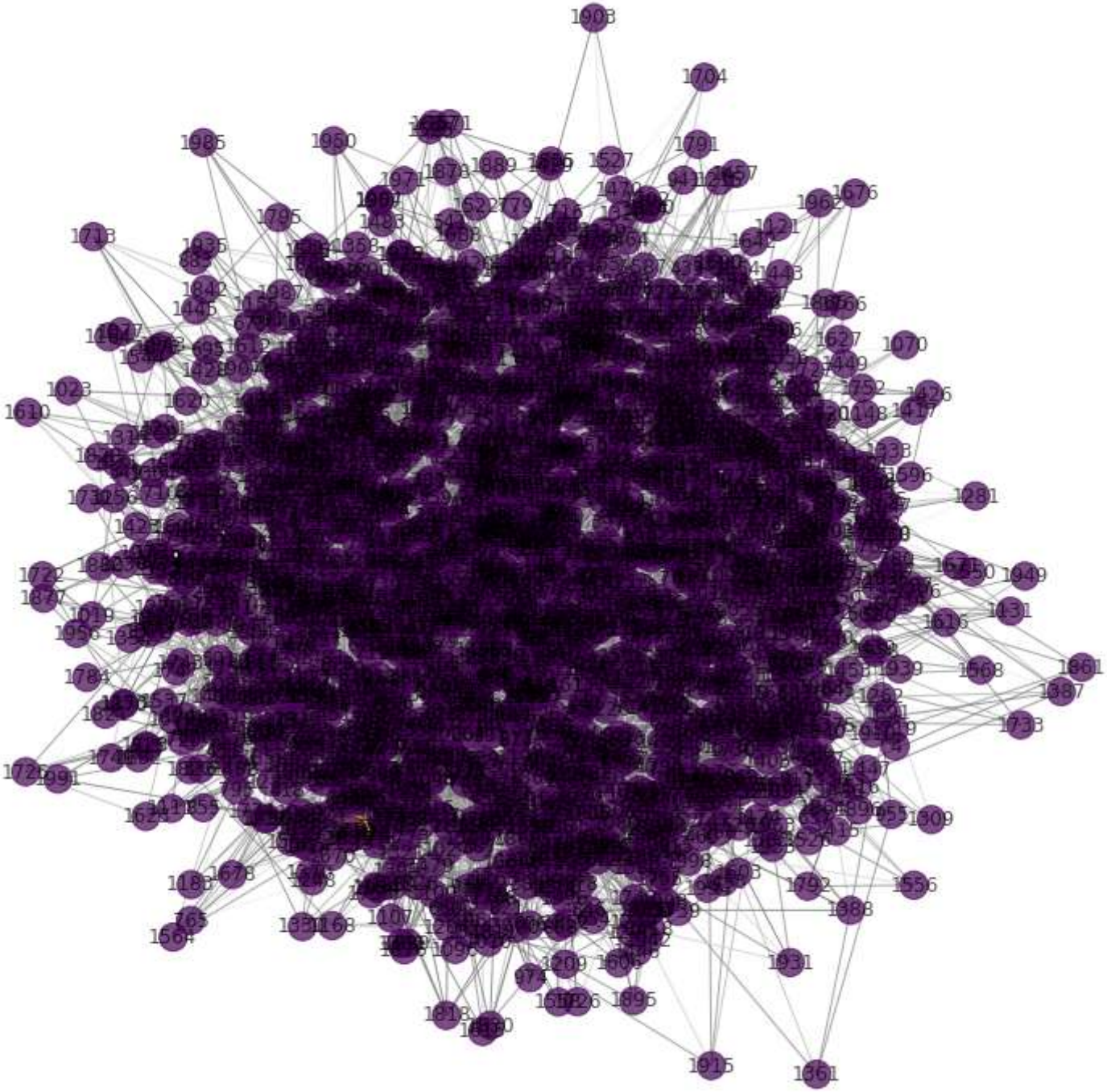


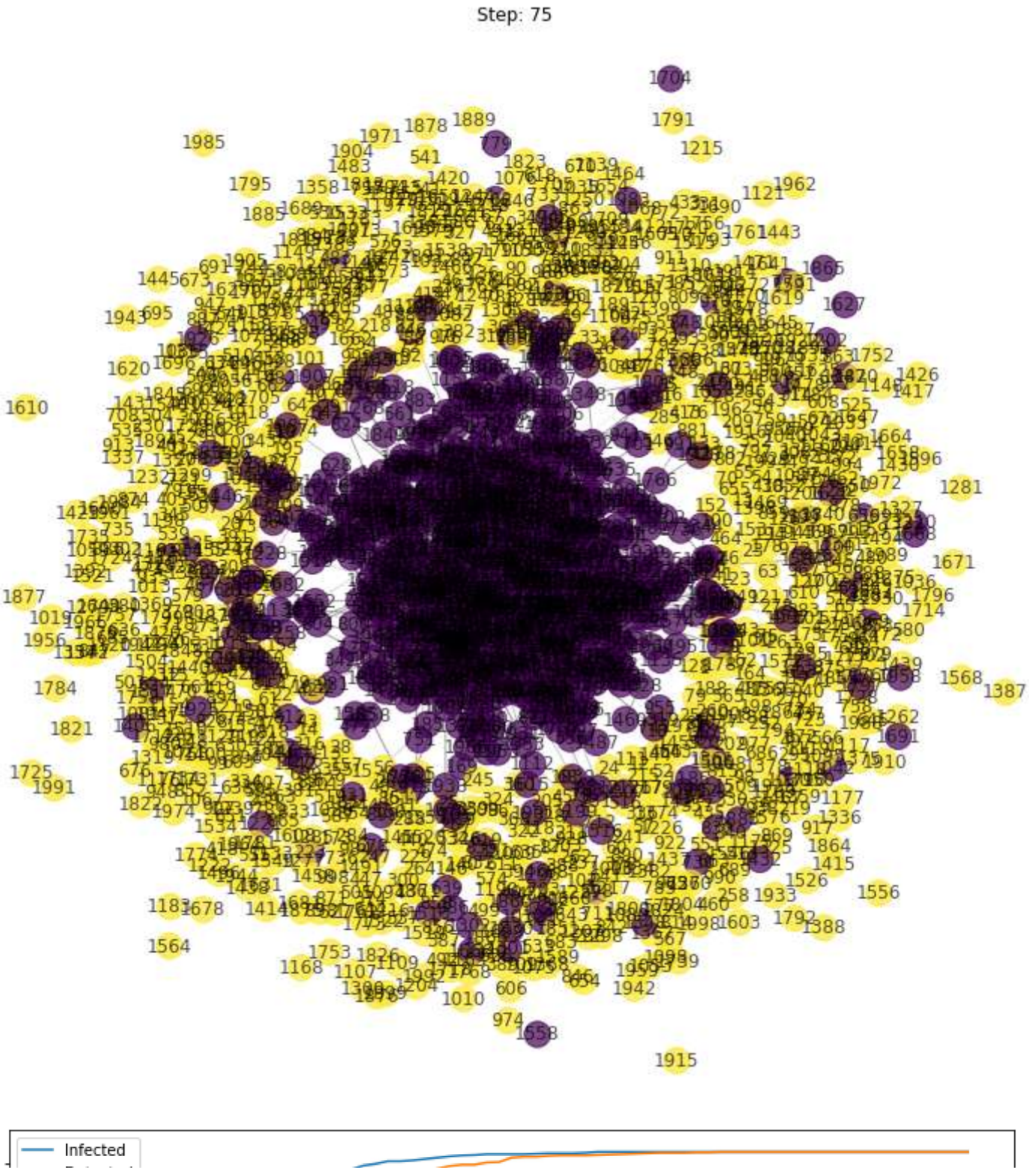
Group 1: social_distancing = False, second_track = False, mass_testing = True

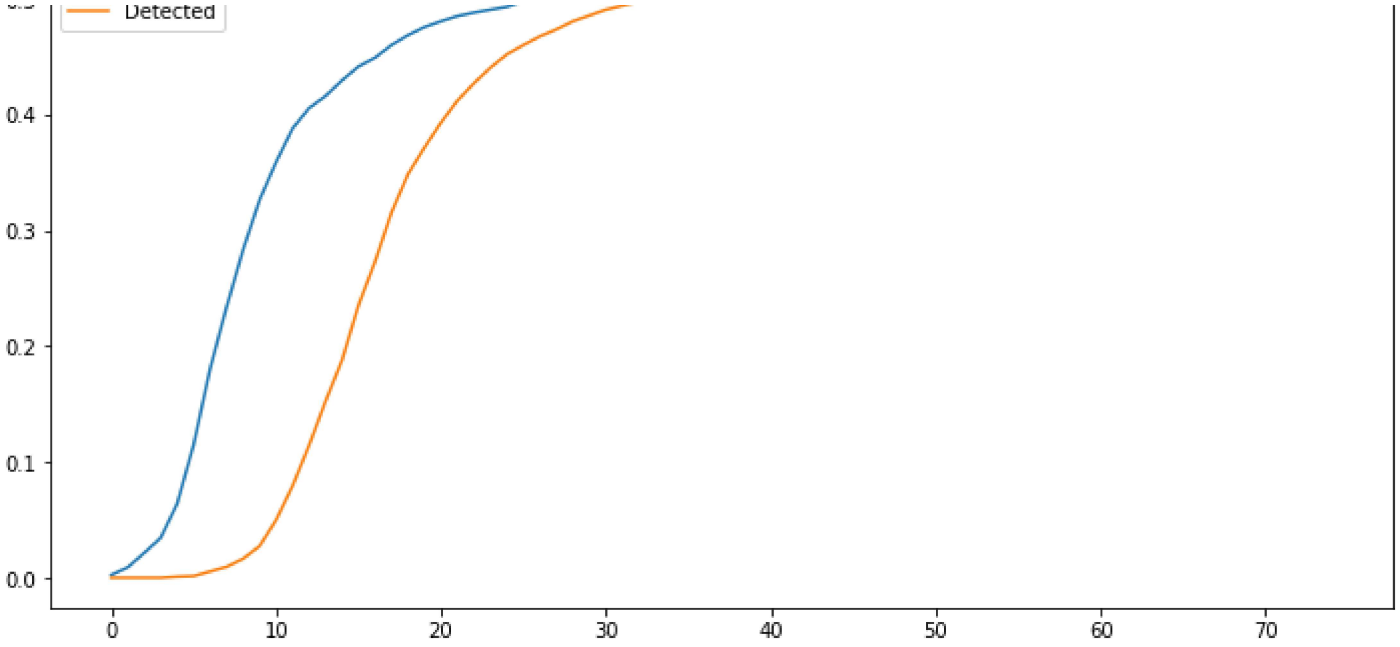
```
# simulate the covid 19 spread and track its progress
print("social_distancing = False, second_track = False, mass_testing = True")
sim = Covid19(social_distancing = False, second_track = False, mass_testing = True)
track, track_hospital = run_sim(sim, model, plot = True)

plt.figure(figsize = (12,6))
plt.plot(range(days), track, label = "Infected")
plt.plot(range(days), track_hospital, label = "Detected")
plt.legend()
plt.show()
```

social_distancing = False, second_track = False, mass_testing = True

Step: 0

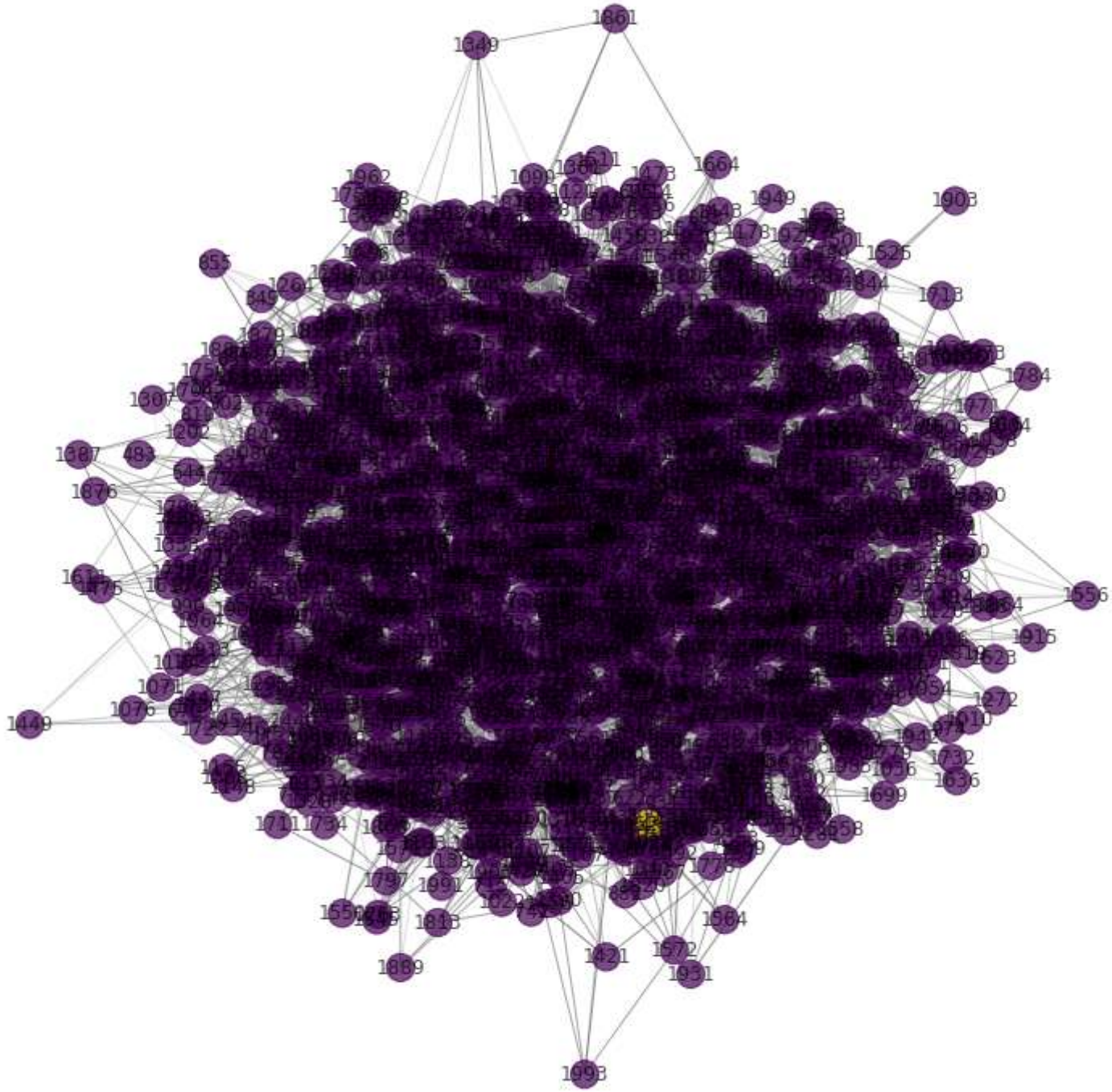


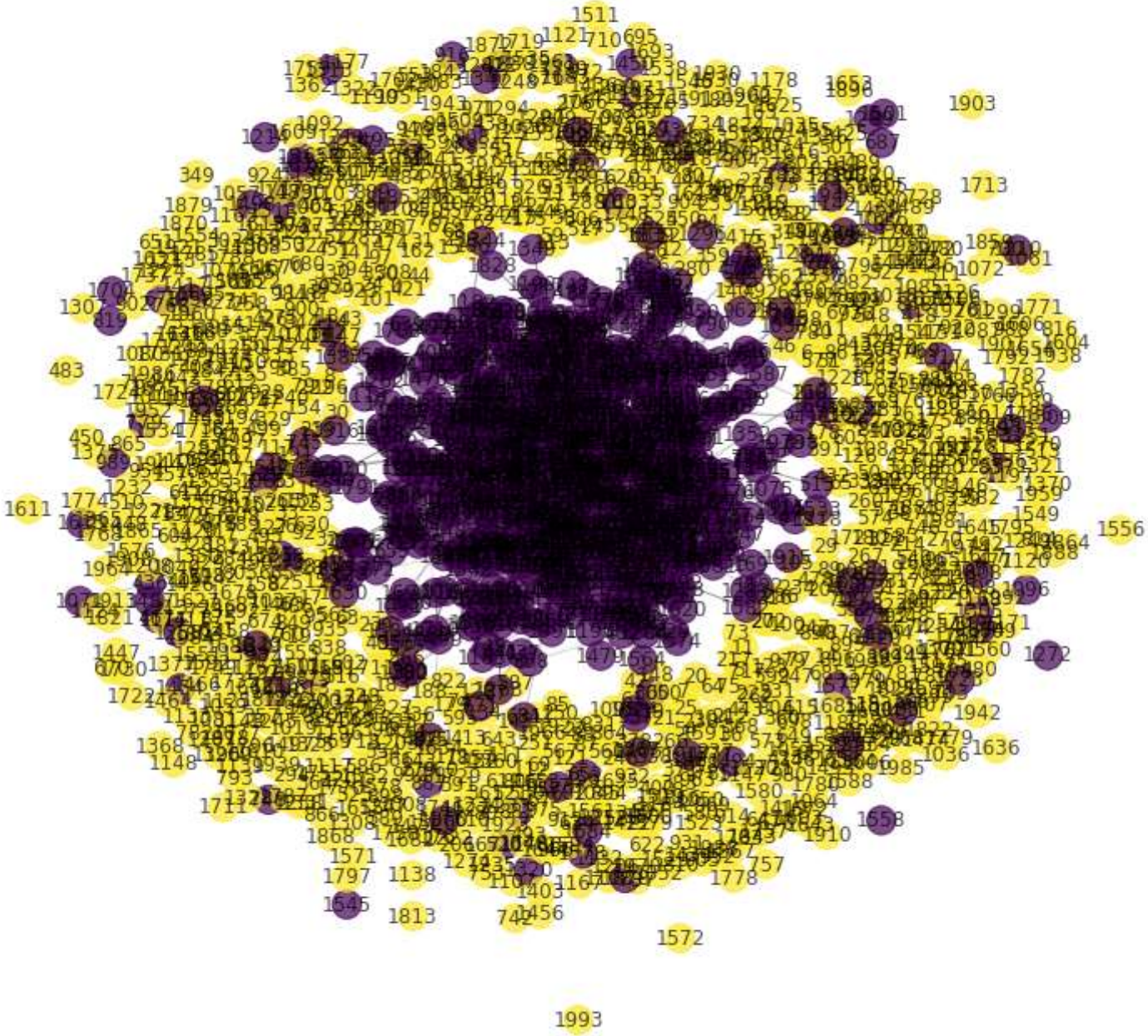

Step: 75
Infected

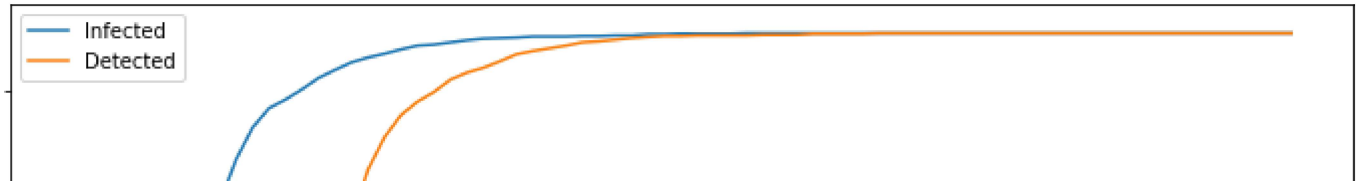


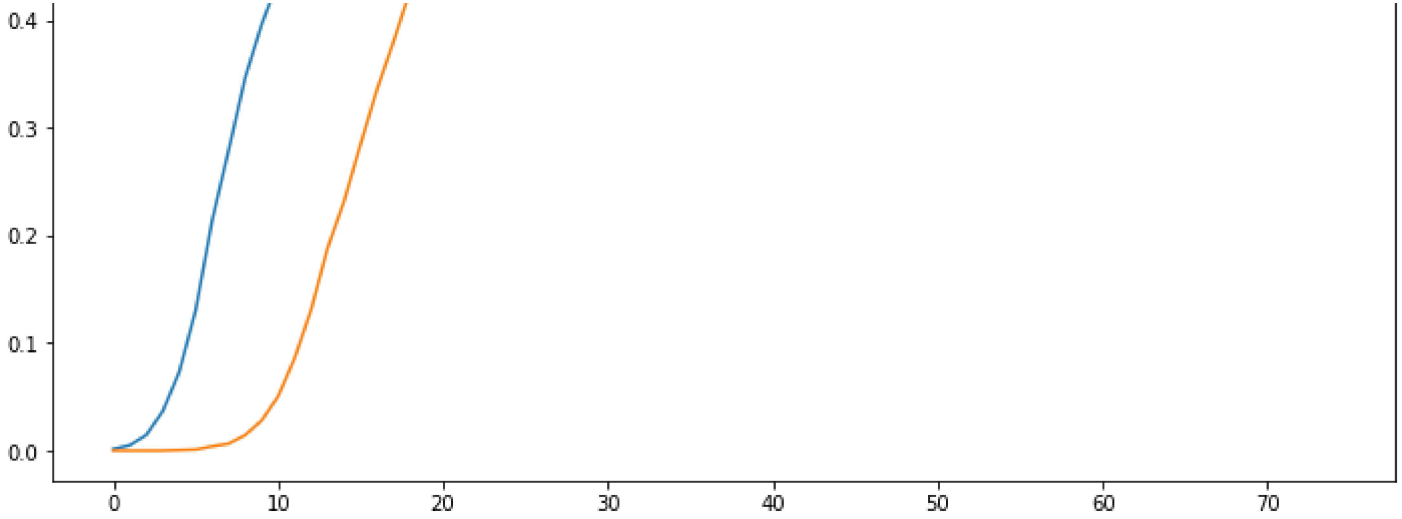
Group 2: social_distancing = False, second_track = True, mass_testing = False

```
# simulate the covid 19 spread and track its progress
print("social_distancing = False, second_track = True, mass_testing = False")
sim = Covid19(social_distancing = False, second_track = False, mass_testing = False)
track, track_hospital = run_sim(sim, model, plot = True)

plt.figure(figsize = (12,6))
plt.plot(range(days), track, label = "Infected")
plt.plot(range(days), track_hospital, label = "Detected")
plt.legend()
plt.show()
```


1861
1349
1664
1962
1949
1903
1626
1713
1784
1556
1916
1272
1732
1636
1699
1658
1964
1672
1421
1931
1889
1993
Step: 75
1861



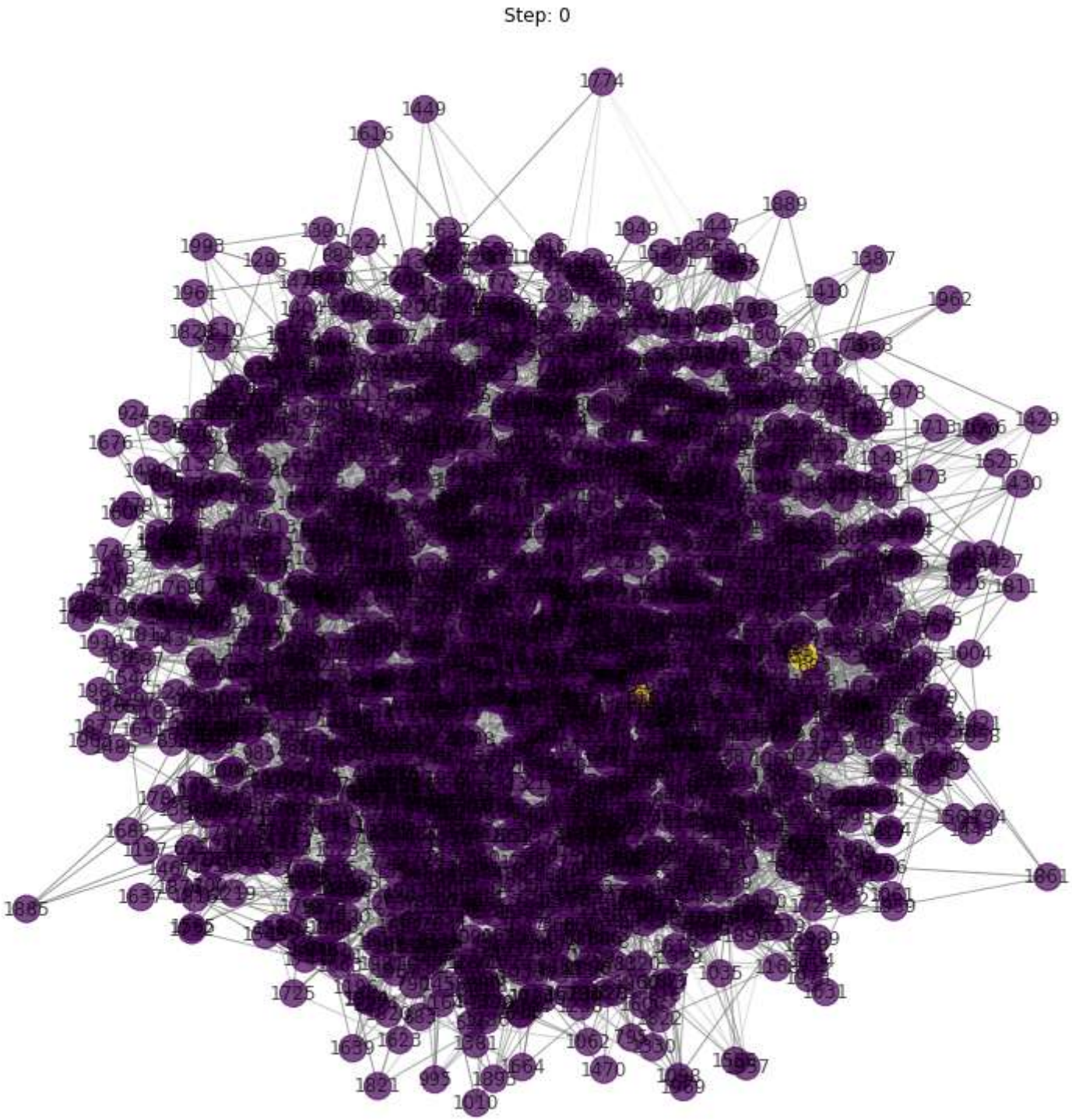

Infected
Detected
0.5

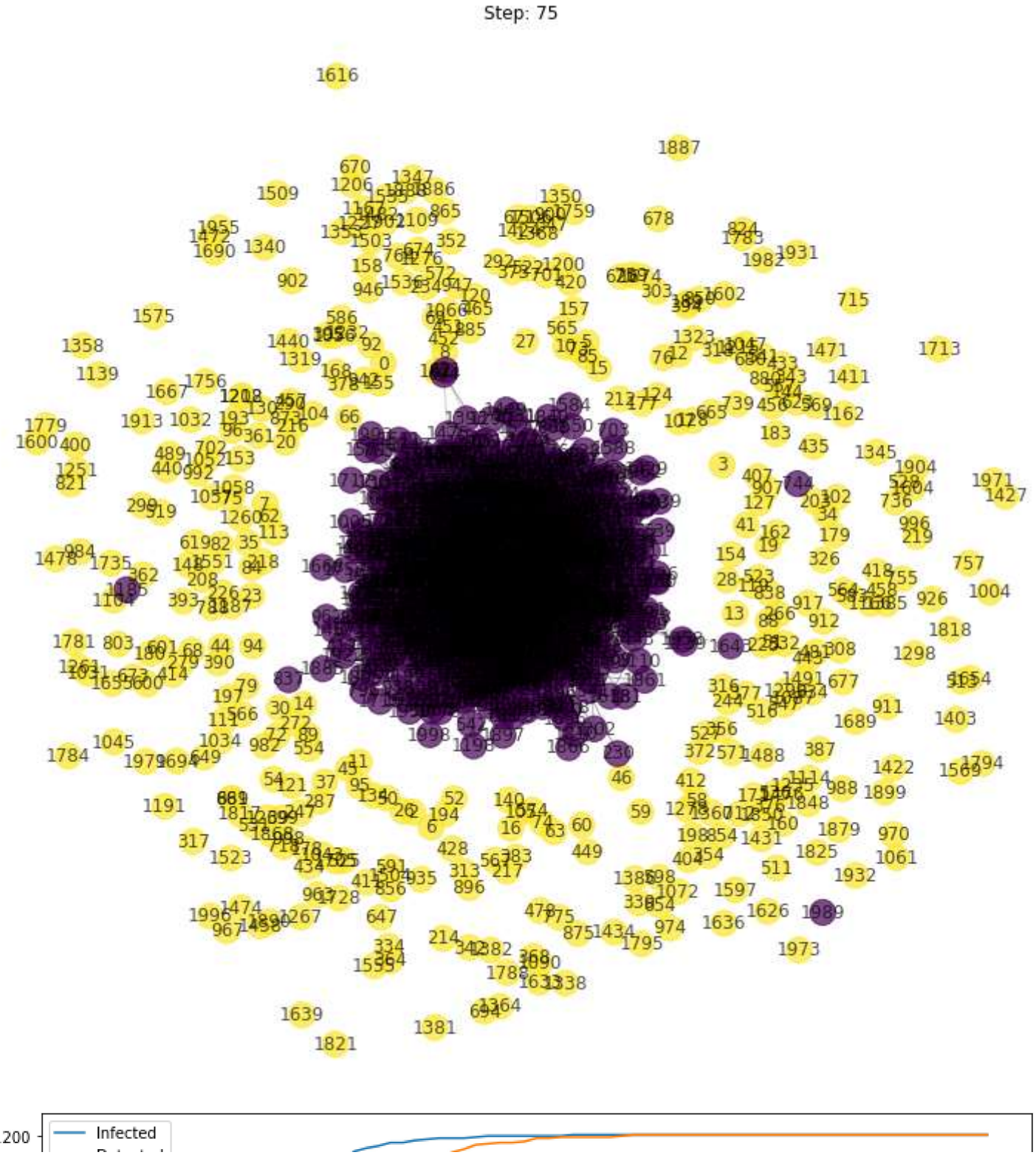


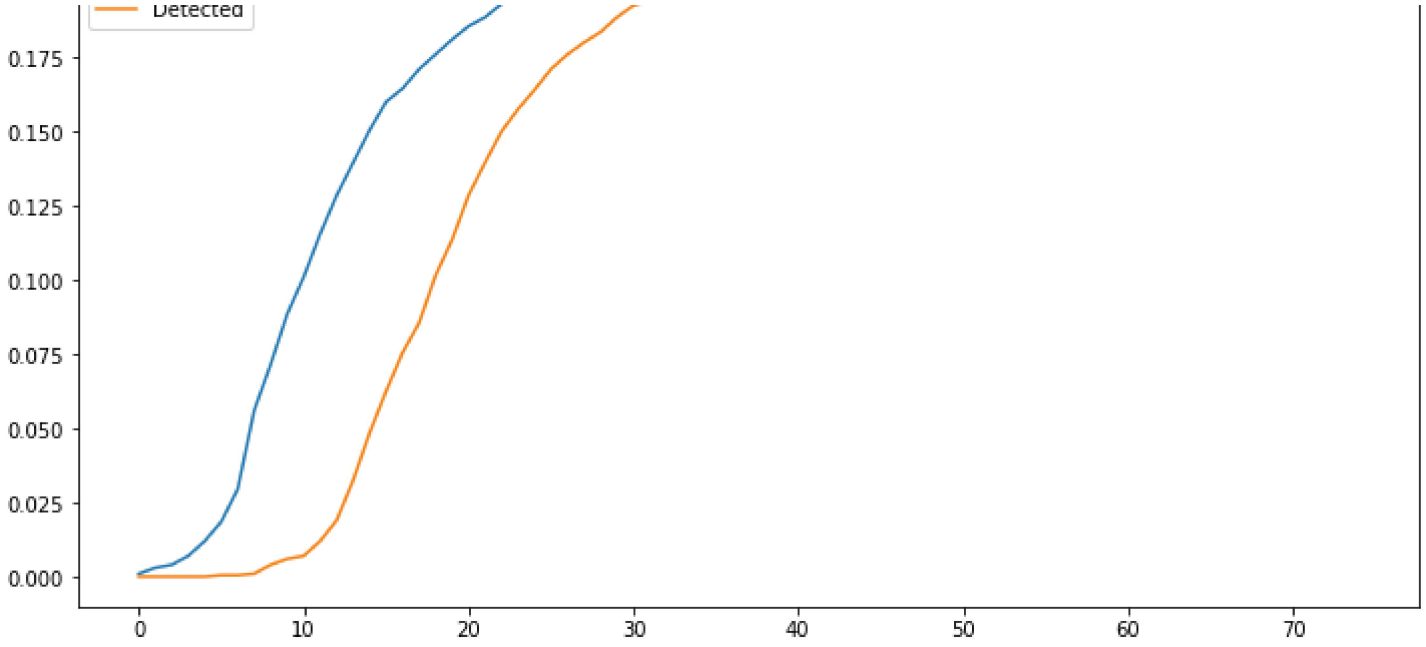
Group 3: social_distancing = True, second_track = False, mass_testing = False

```
# simulate the covid 19 spread and track its progress
print("social_distancing = True, second_track = False, mass_testing = False")
sim = Covid19(social_distancing = True, second_track = False, mass_testing = False)
track, track_hospital = run_sim(sim, model, plot = True)

plt.figure(figsize = (12,6))
plt.plot(range(days), track, label = "Infected")
plt.plot(range(days), track_hospital, label = "Detected")
plt.legend()
plt.show()
```

social_distancing = True, second_track = False, mass_testing = False




Step: 75
Infected
0.200



Test model on multiple simulations

```
# functions to run the model for 8 policies
def run_output(trial = 10, days = days):
  print("Total " + str(trial) + " trials")
  # keep track of the results
  tracks = []
  tracks_hospital = []

  # simulating
  for i in range(trial):
    print("Trial " + str(i) + " is running")

    # initilizing the network model
    model = SocialDynamicsSimulation()
    model.initialize()
    for i in range(30000):
      model.update()

    '''
    Group 0: "social_distancing = False, second_track = False, mass_testing = False",
```

```
    Group 1: "social_distancing = False, second_track = False, mass_testing = True ",
    Group 2: "social_distancing = False, second_track = True , mass_testing = False",
    Group 3: "social_distancing = True , second_track = False, mass_testing = False",
    Group 4: "social_distancing = False, second_track = True , mass_testing = True ",
    Group 5: "social_distancing = True , second_track = False, mass_testing = True ",
    Group 6: "social_distancing = True , second_track = True , mass_testing = False",
    Group 7: "social_distancing = True , second_track = True , mass_testing = True ".

    '''

    # covid 19 group 0:
    sim1 = Covid19(social_distancing=False, second_track=False, mass_testing = False)
    track_1, track_hospital_1 = run_sim(sim1, model, days)

    # covid 19 group 1:
    sim2 = Covid19(social_distancing=False, second_track=False, mass_testing = True)
    track_2, track_hospital_2 = run_sim(sim2, model, days)

    # covid 19 group 2:
    sim3 = Covid19(social_distancing=False, second_track=True, mass_testing = False)
    track_3, track_hospital_3 = run_sim(sim3, model, days)

    # covid 19 group 3:
    sim4 = Covid19(social_distancing=True, second_track=False, mass_testing = False)
    track_4, track_hospital_4 = run_sim(sim4, model, days)

    # covid 19 group 4:
    sim5 = Covid19(social_distancing=False, second_track=True, mass_testing = True)
    track_5, track_hospital_5 = run_sim(sim5, model, days)

    # covid 19 group 5:
    sim6 = Covid19(social_distancing=True, second_track=False, mass_testing = True)
    track_6, track_hospital_6 = run_sim(sim6, model, days)

    # covid 19 group 6:
    sim7 = Covid19(social_distancing=True, second_track=True, mass_testing = False)
    track_7, track_hospital_7 = run_sim(sim7, model, days)

    # covid 19 group 7:
    sim8 = Covid19(social_distancing=True, second_track=True, mass_testing = True)
    track_8, track_hospital_8 = run_sim(sim8, model, days)
```

```
61
62     # get all the values of different policies
63     tracks.append([track_1, track_2, track_3, track_4,
64                    track_5, track_6, track_7, track_8])
65
66     tracks_hospital.append([track_hospital_1, track_hospital_2, track_hospital_3, track_hospital_4,
67                             track_hospital_5, track_hospital_6, track_hospital_7, track_hospital_8])
68   return tracks, tracks_hospital
```

```
1 full_status = ["social_distancing = False, second_track = False, mass_testing = False",
2                "social_distancing = False, second_track = False, mass_testing = True ",
3                "social_distancing = False, second_track = True , mass_testing = False",
4                "social_distancing = True , second_track = False, mass_testing = False",
5                "social_distancing = False, second_track = True , mass_testing = True ",
6                "social_distancing = True , second_track = False, mass_testing = True ",
7                "social_distancing = True , second_track = True , mass_testing = False",
8                "social_distancing = True , second_track = True , mass_testing = True "]
```

```
1 tracks, tracks_hospital = run_output(trial = 30)
```

Save the weight

Save the values to run afterward and not having to wait 5 hours.

```
1 np.save("CS166 FP tracks.npy", tracks)
2 np.save("CS166 FP tracks hospitals.npy", tracks_hospital)
```

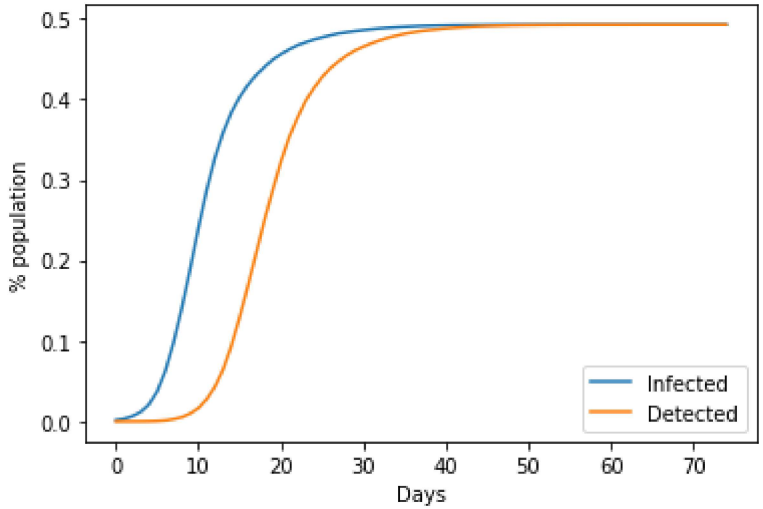
Visulize results

```
1 import os
2 os.chdir("/content/drive/My Drive/Minerva/CS166/CS166 Covid19")
3
4 full_tracks = np.load("CS166 FP tracks.npy")
5 full_tracks_hospital = np.load("CS166 FP tracks hospitals.npy")
```

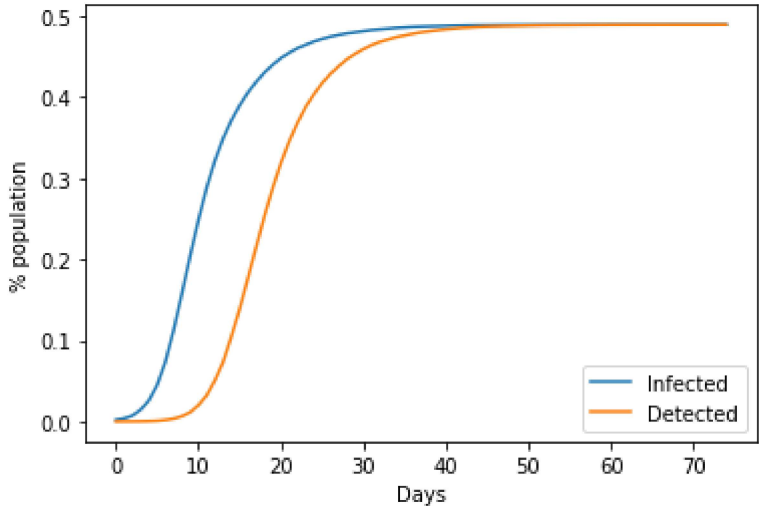
Visulize the pandemic progress: infected vs detected

```
for i in range(8):
  print(full_status[i])
  #plt.figure(figsize = (6,6))
  plt.plot(range(days), np.mean(full_tracks[:,i], axis = 0), label = "Infected")
  plt.plot(range(days), np.mean(full_tracks_hospital[:,i], axis = 0), label = "Detected")
  plt.xlabel("Days")
  plt.ylabel("% population")
  plt.legend()
  plt.show()
```

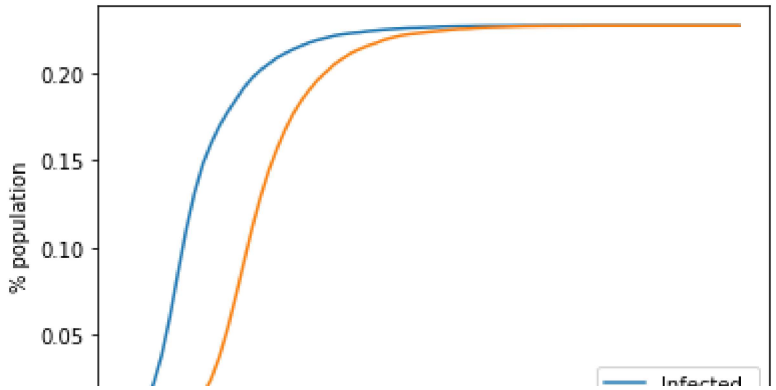
social_distancing = False, second_track = False, mass_testing = False

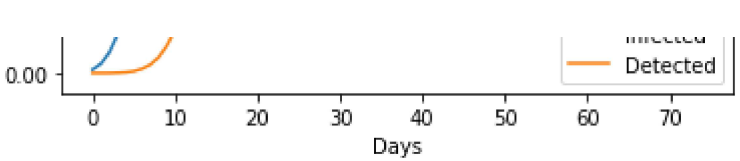


social_distancing = False, second_track = False, mass_testing = True

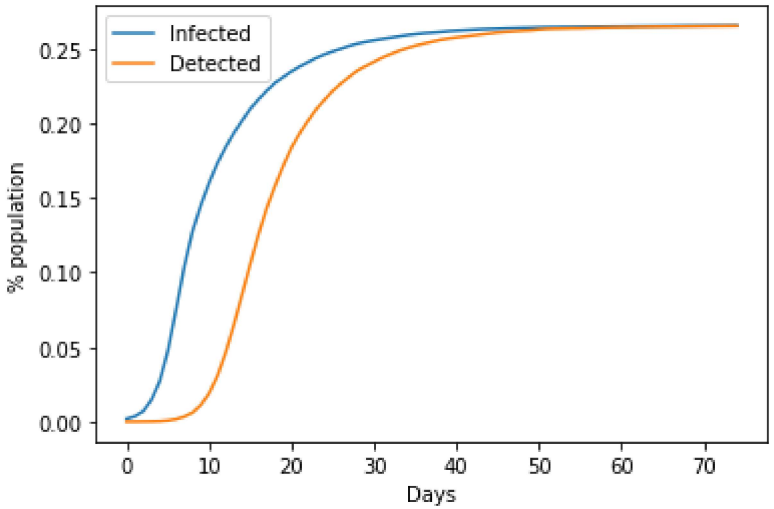


social_distancing = False, second_track = True , mass_testing = False

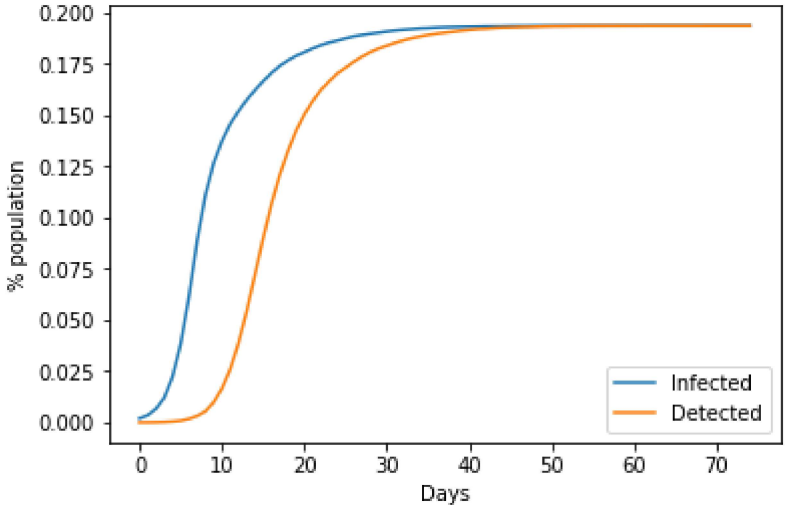




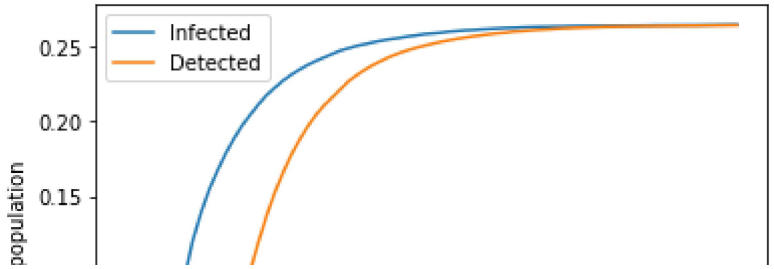
social_distancing = True , second_track = False, mass_testing = False

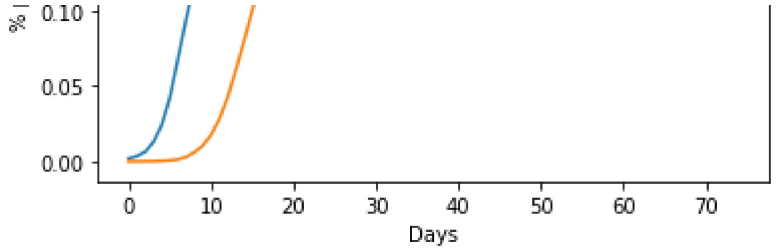


social_distancing = False, second_track = True , mass_testing = True

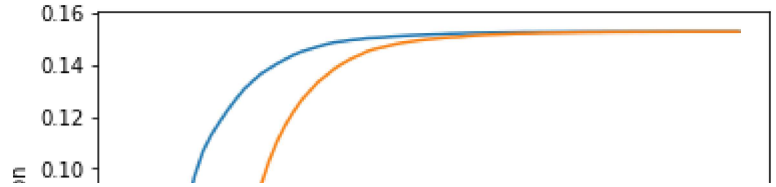


social_distancing = True , second_track = False, mass_testing = True



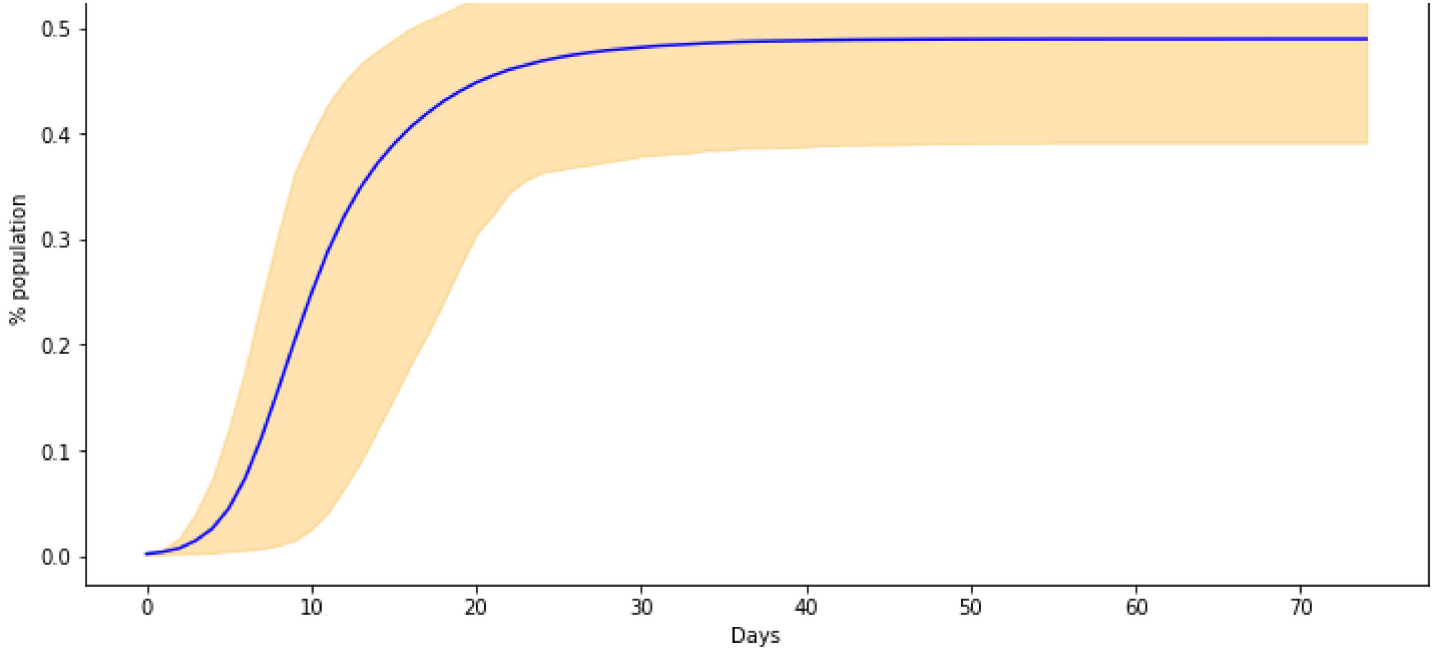



```
social_distancing = True , second_track = True , mass_testing = False
```

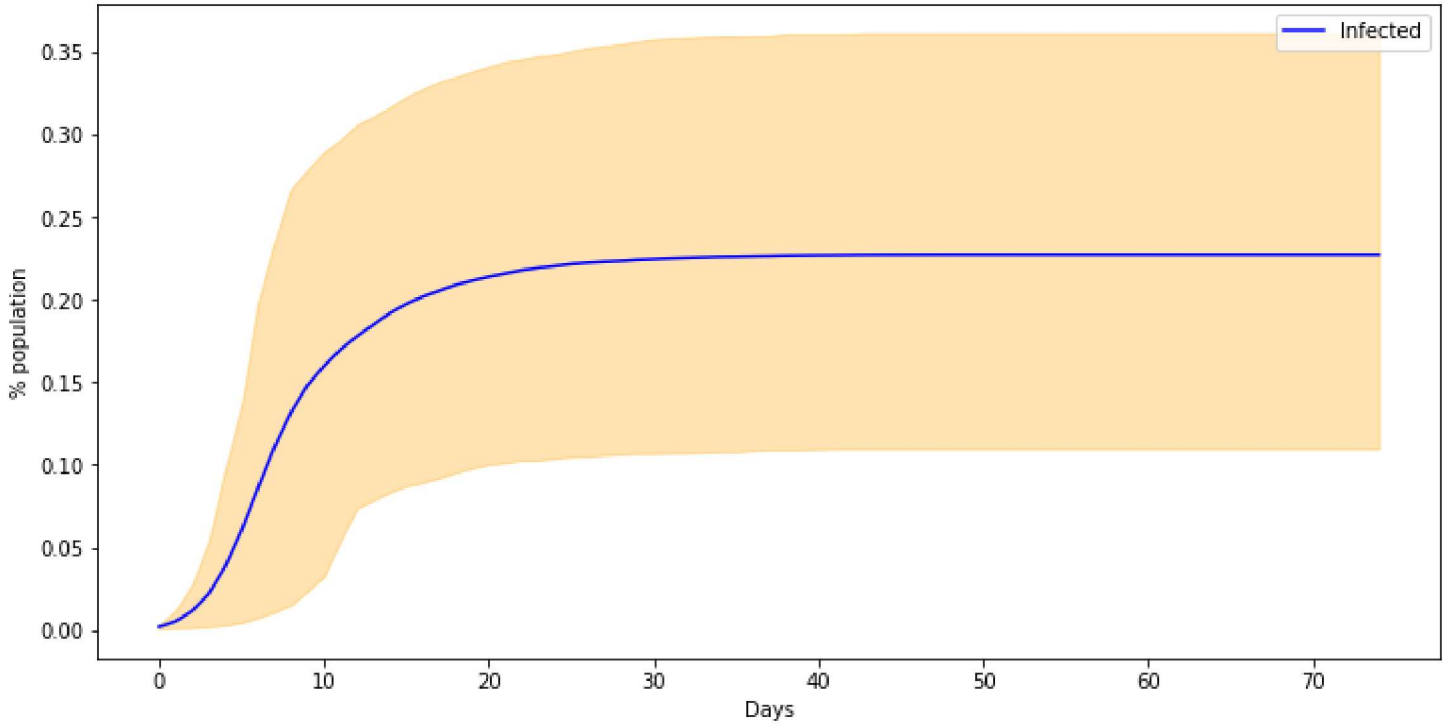


Visulize the pandemic progress: infected and its uncertainty

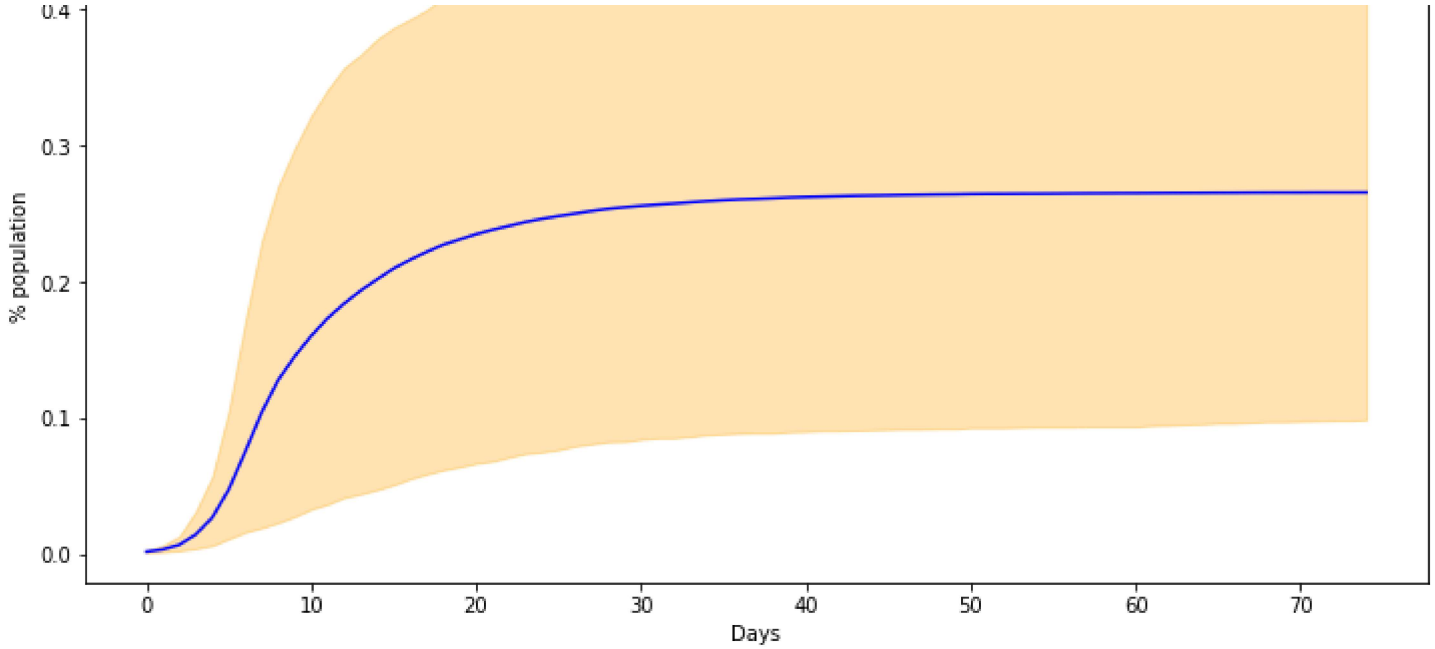
```
for i in range(8):
  print(full_status[i])
  plt.figure(figsize = (12,6))
  plt.plot(range(days), np.mean(full_tracks[:,i], axis = 0), label = "Infected", color = 'blue')
  x1 = np.quantile(full_tracks[:,i], 0.95, axis = 0)
  x2 = np.quantile(full_tracks[:,i], 0.05, axis = 0)
  plt.fill_between(list(range(days)), x1, x2,
                   facecolor="orange", # The fill color
                   color='orange',       # The outline color
                   alpha=0.3)
  plt.xlabel("Days")
  plt.ylabel("% population")
  plt.legend()
  plt.show()
```



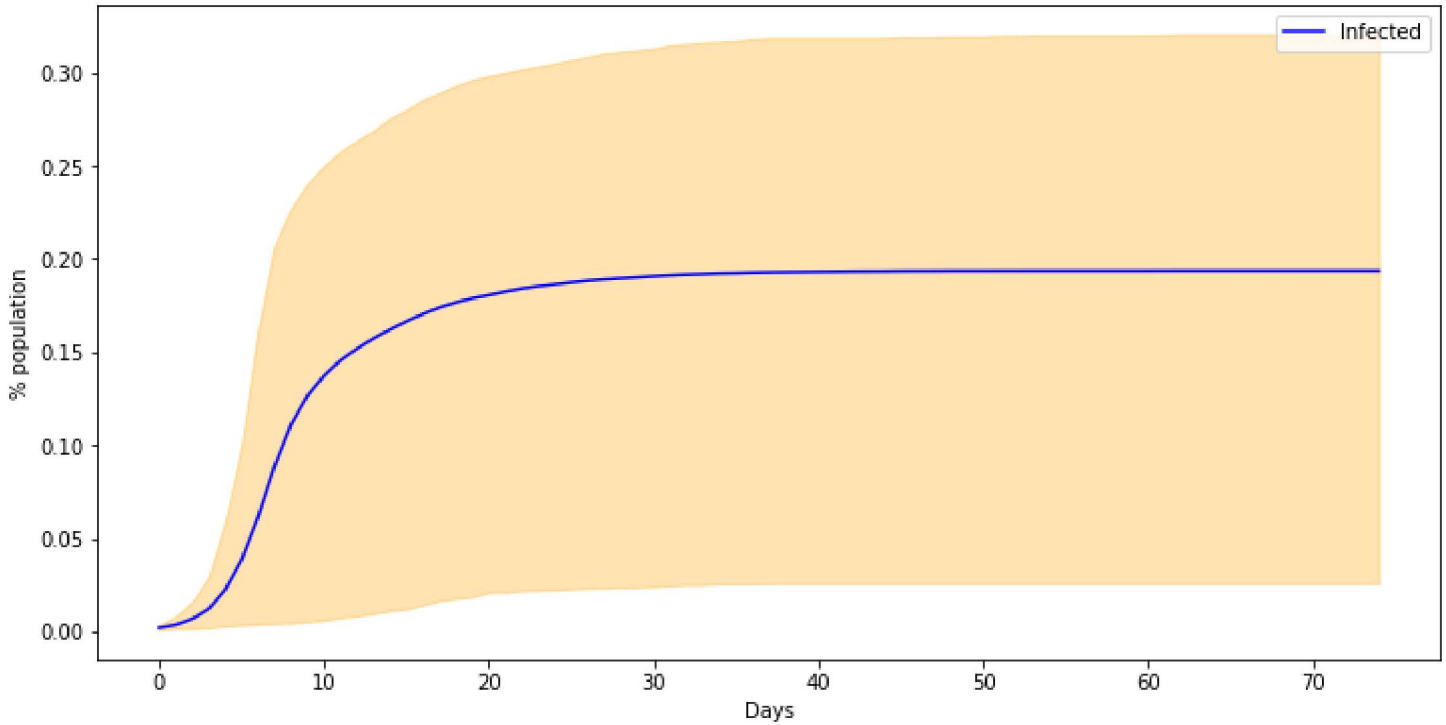
social_distancing = False, second_track = True , mass_testing = False



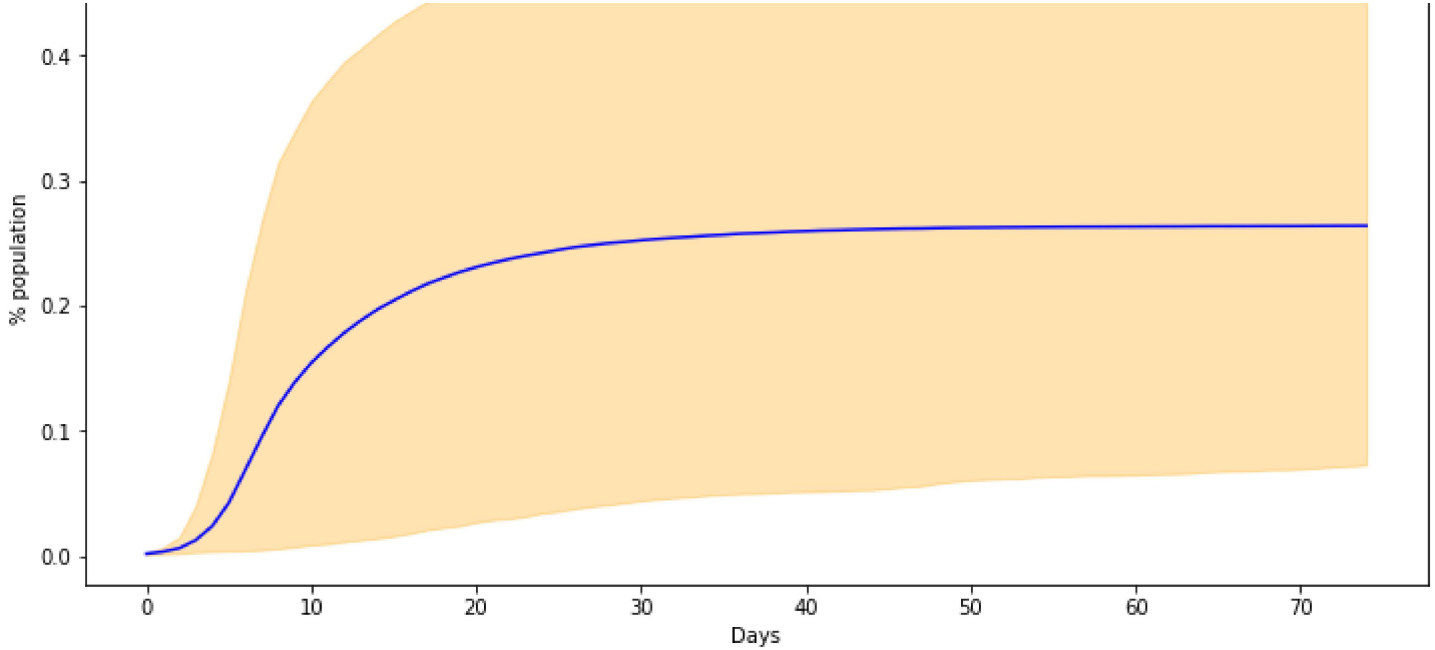
social_distancing = True , second_track = False, mass_testing = False



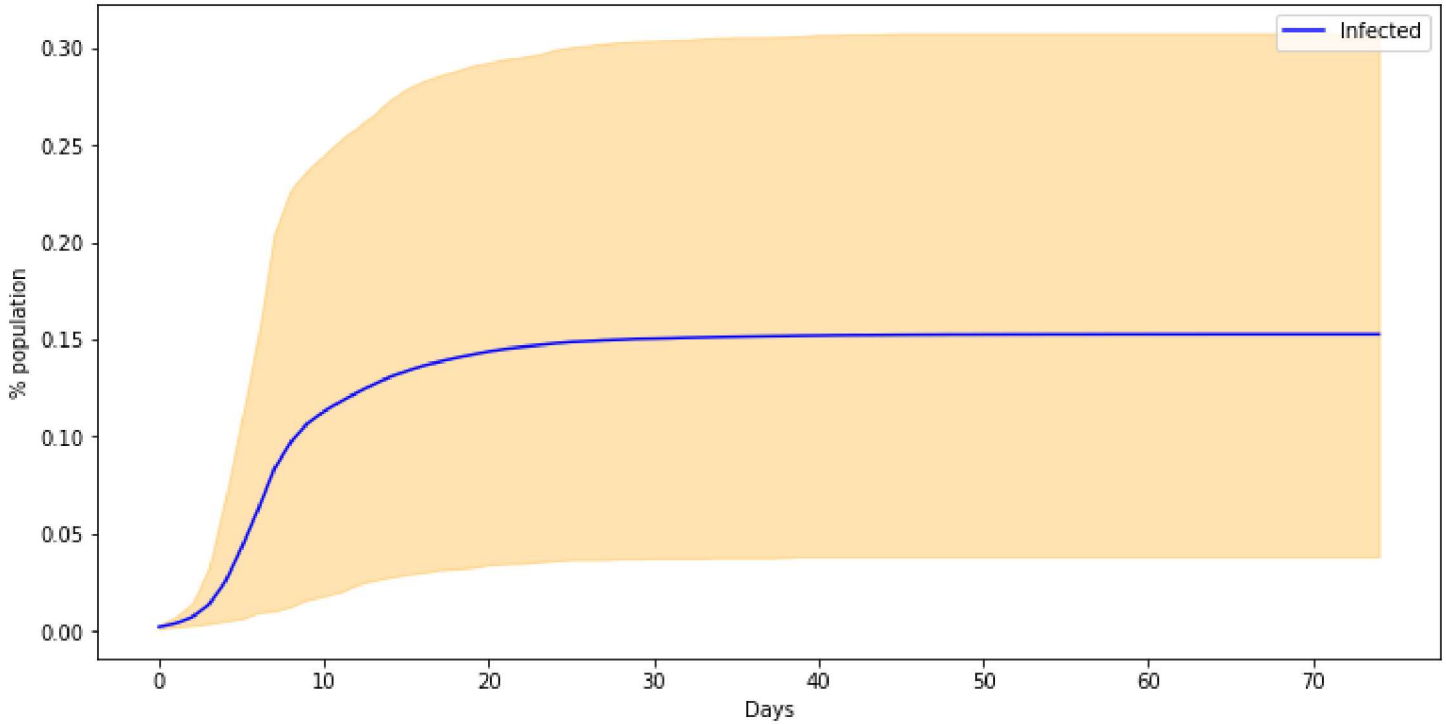
social_distancing = False, second_track = True , mass_testing = True



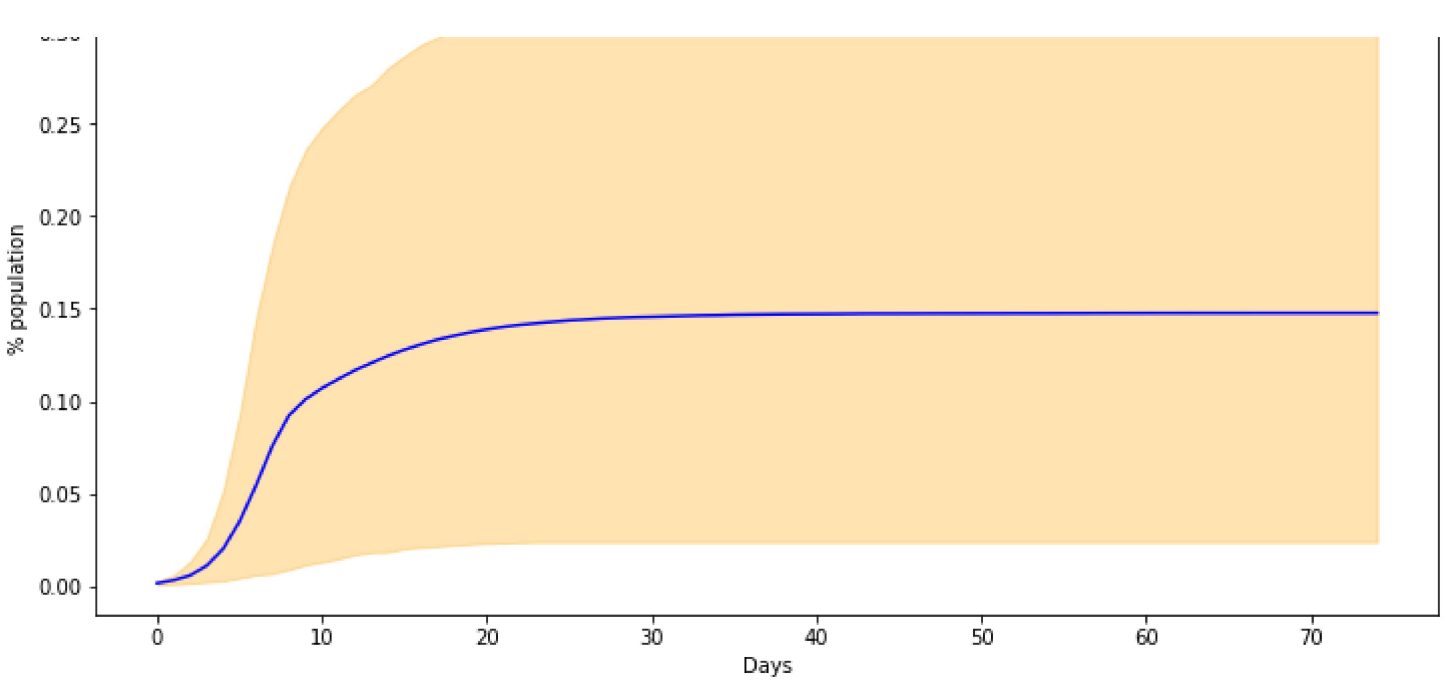
social_distancing = True , second_track = False, mass_testing = True



social_distancing = True , second_track = True , mass_testing = False



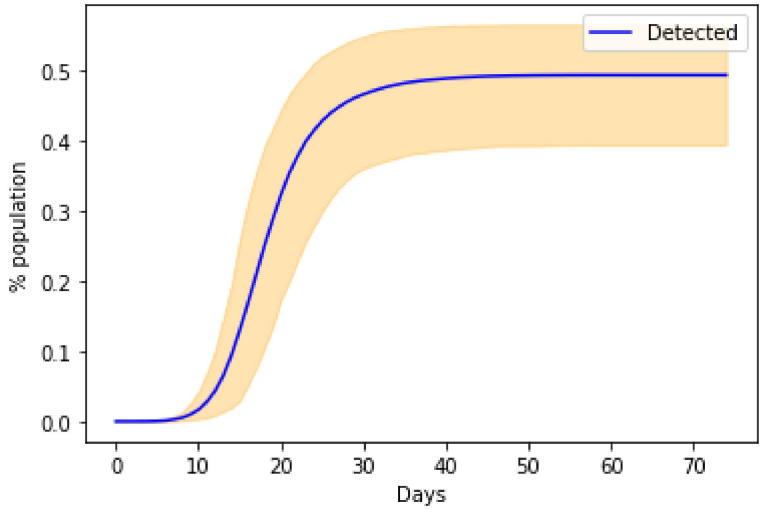
social_distancing = True , second_track = True , mass_testing = True


0.25
0.20
0.15
0.10
0.05
0.00
% population
0
10
20
30
40
50
60
70
Days

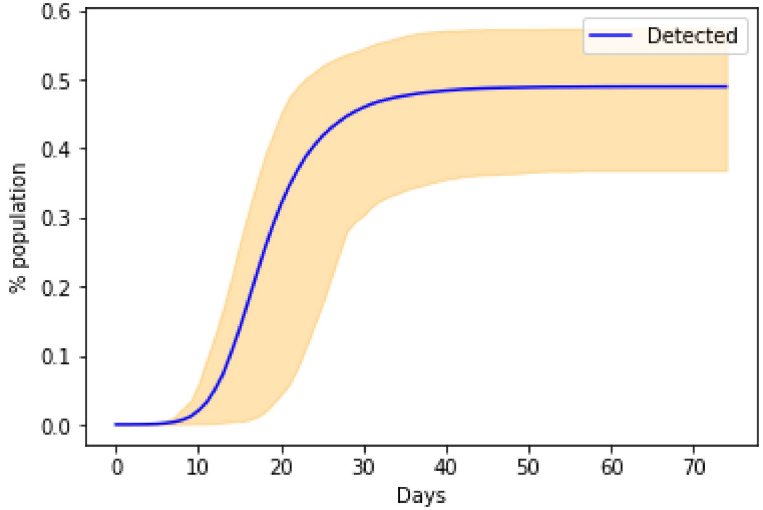
Visulize the pandemic progress: detected and its uncertainty

```
for i in range(8):
  print(full_status[i])
  #plt.figure(figsize = (12,6))
  plt.plot(range(days), np.mean(full_tracks_hospital[:,i], axis = 0), label = "Detected", color = 'blue')
  x1 = np.quantile(full_tracks_hospital[:,i], 0.975, axis = 0)
  x2 = np.quantile(full_tracks_hospital[:,i], 0.025, axis = 0)
  plt.fill_between(list(range(days)), x1, x2,
                   facecolor="orange", # The fill color
                   color='orange',       # The outline color
                   alpha=0.3)
  plt.xlabel("Days")
  plt.ylabel("% population")
  plt.legend()
  plt.show()
```

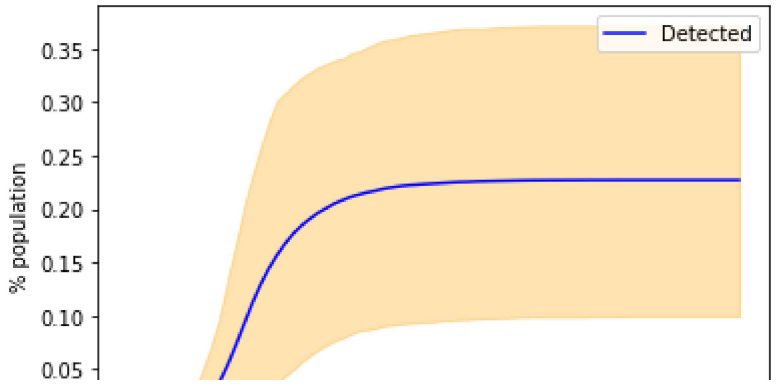
social_distancing = False, second_track = False, mass_testing = False

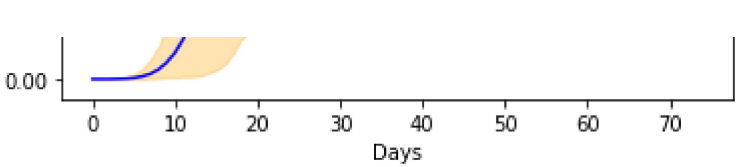


social_distancing = False, second_track = False, mass_testing = True

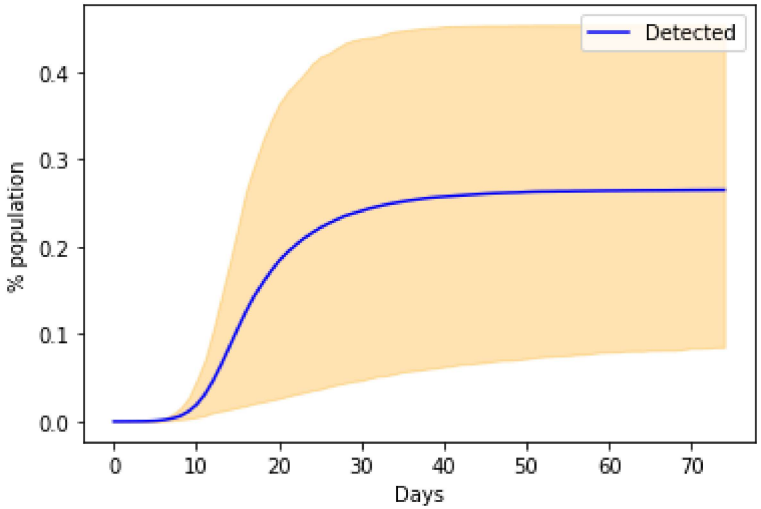


social_distancing = False, second_track = True , mass_testing = False

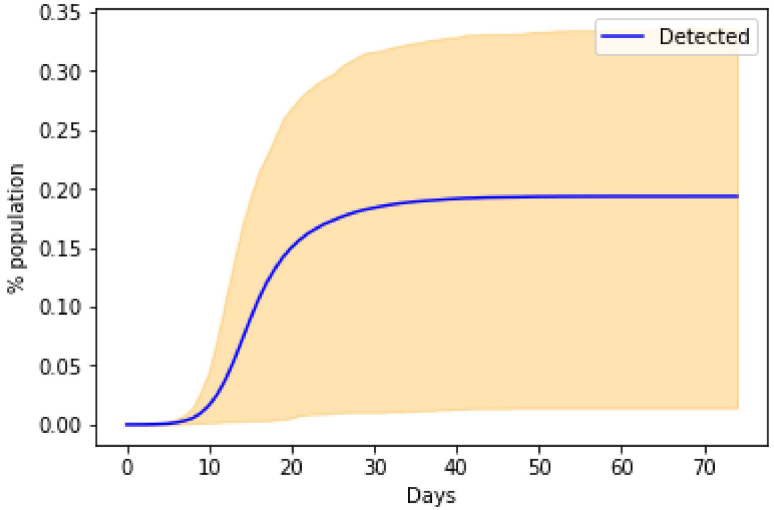




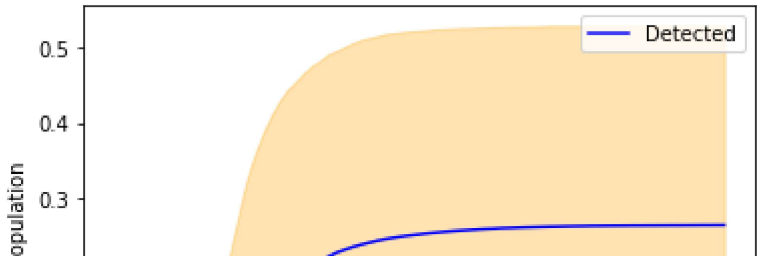
social_distancing = True , second_track = False, mass_testing = False

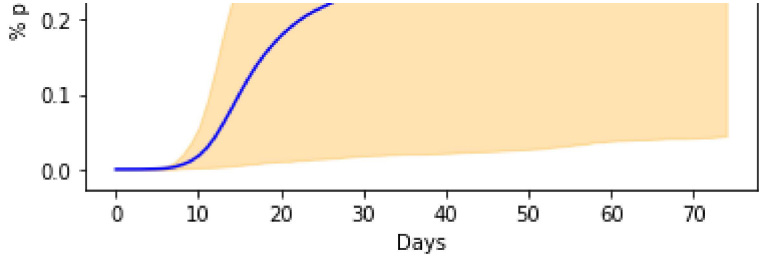


social_distancing = False, second_track = True , mass_testing = True

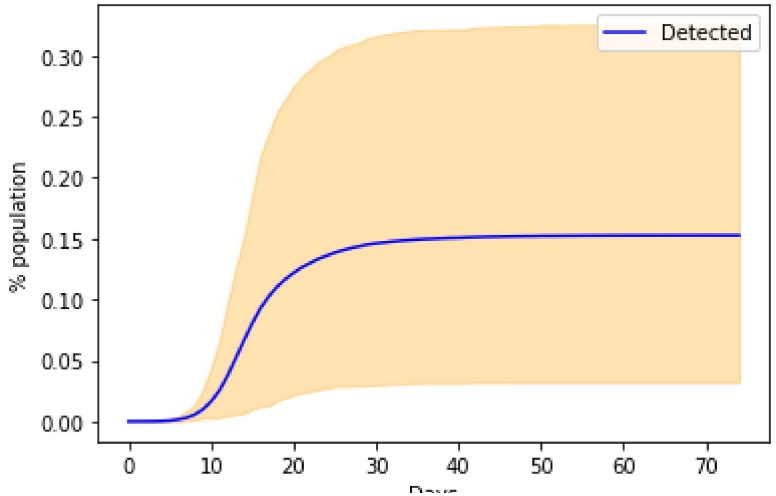


social_distancing = True , second_track = False, mass_testing = True






```
social_distancing = True , second_track = True , mass_testing = False
```

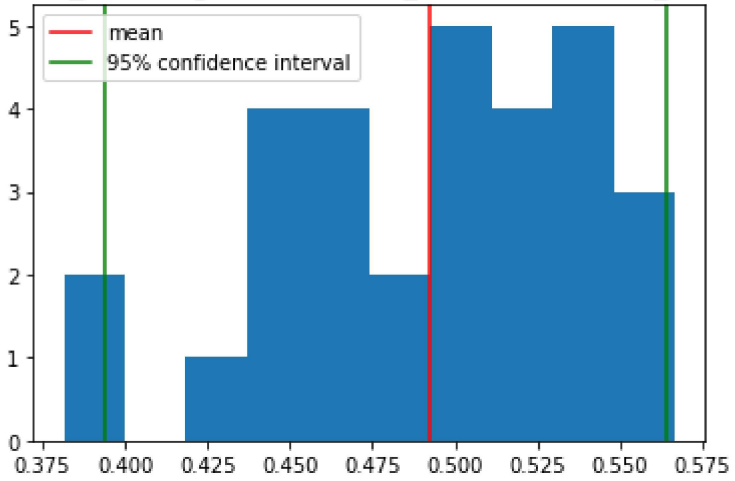


Visulize the histogram of final population infection level in different models

```
for i in range(8):
  results = [full_tracks[:,i][t][-1] for t in range(30)]
  mean = np.mean(results)
  median, i975, i025 = np.quantile(results, [0.5, 0.975, 0.025])
  print("Mean:", mean)
  print("95% confidence interval = [{}, {}]".format(i025, i975))
  plt.hist(results)
  plt.axvline(mean, color = "red", label = "mean")
  plt.axvline(i975, color = "green", label = "95% confidence interval")
  plt.axvline(i025, color = "green")
  plt.legend()
  plt.title("Group " + str(i) + ": " + full_status[i])
  plt.show()
  print("")


```

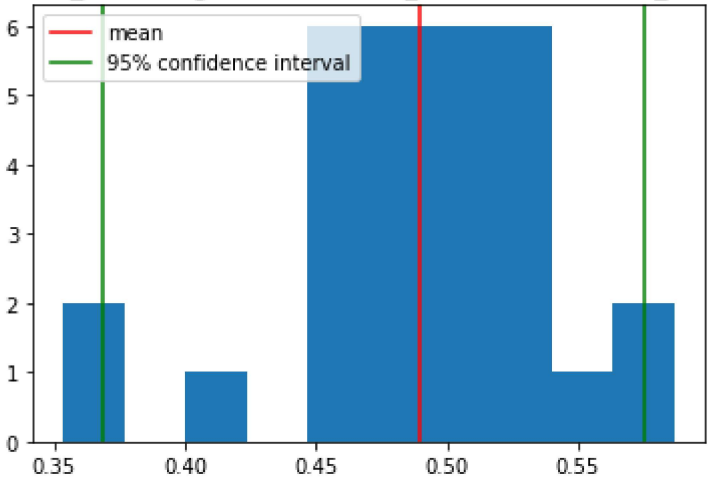
```
Mean: 0.4925666666666667
95% confidence interval = [0.39382500000000004, 0.564325]
```

Group 0: social_distancing = False, second_track = False, mass_testing = False



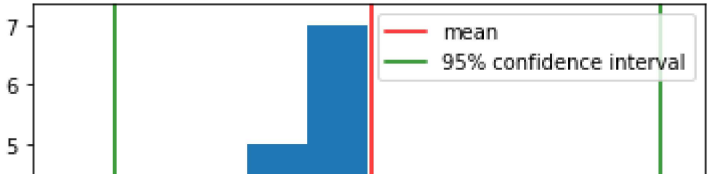

```
Mean: 0.4895166666666667
95% confidence interval = [0.36836250000000004, 0.5756249999999999]
```

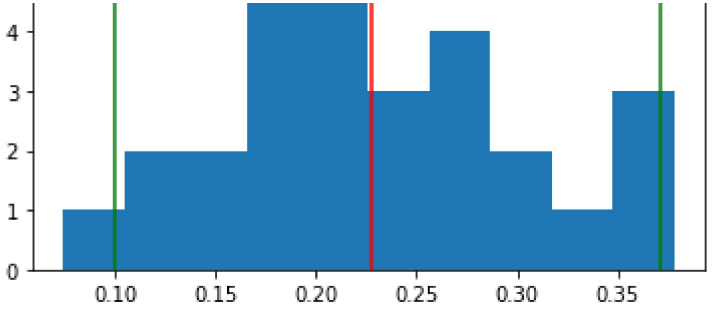
Group 1: social_distancing = False, second_track = False, mass_testing = True




```
Mean: 0.22721666666666668
95% confidence interval = [0.099375, 0.37183749999999993]
```

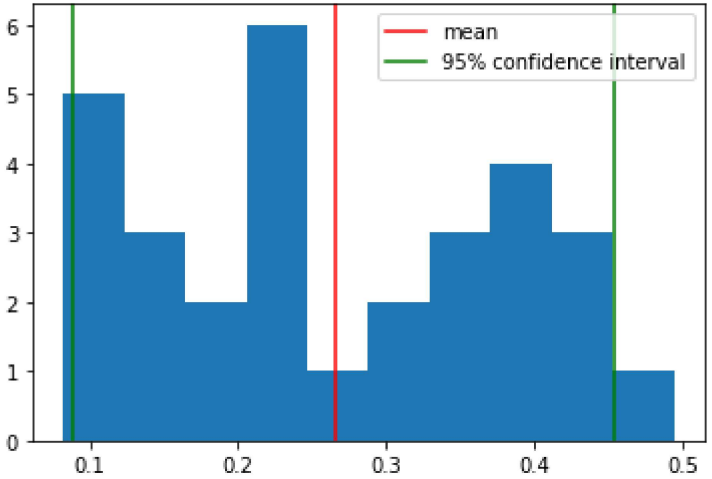
Group 2: social_distancing = False, second_track = True , mass_testing = False





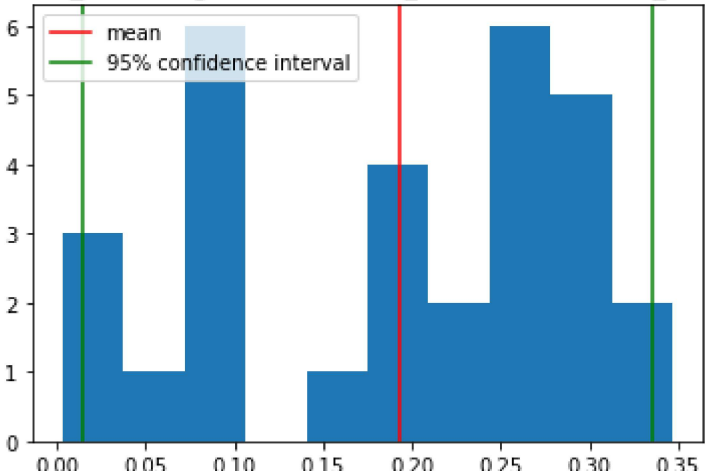
Mean: 0.26541666666666663
95% confidence interval = [0.08852499999999999, 0.45448749999999993]

Group 3: social_distancing = True , second_track = False, mass_testing = False



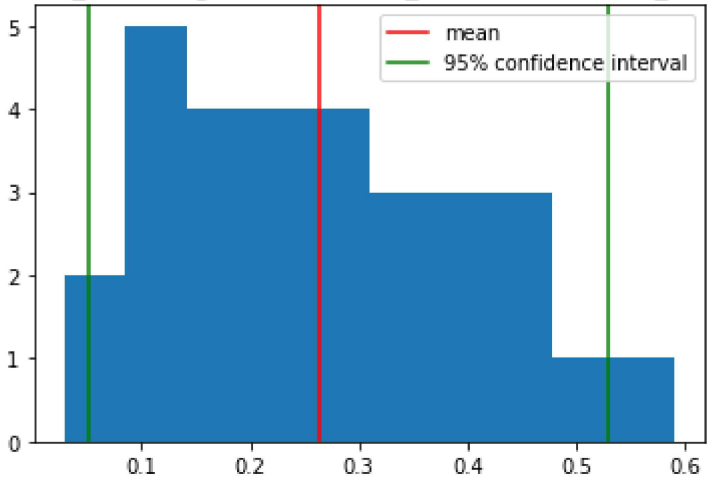
Mean: 0.1935666666666667
95% confidence interval = [0.0142375, 0.33576249999999996]

Group 4: social_distancing = False, second_track = True , mass_testing = True



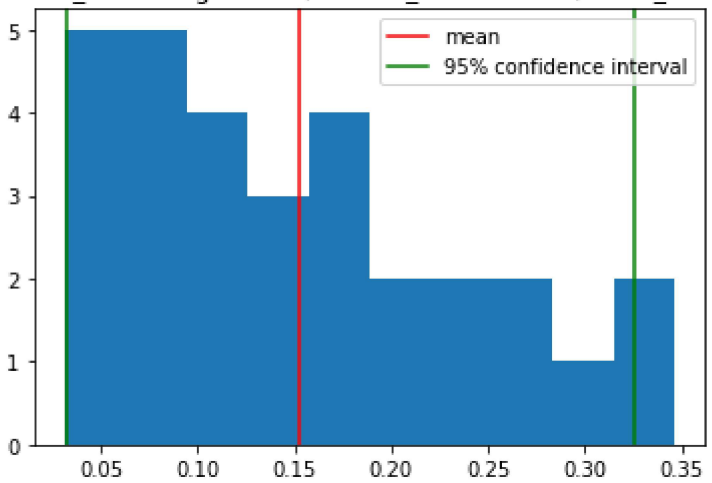
```
Mean: 0.2639666666666666
95% confidence interval = [0.051475, 0.5292374999999999]
```

Group 5: social_distancing = True , second_track = False, mass_testing = True




```
Mean: 0.15263333333333332
95% confidence interval = [0.032087500000000005, 0.32547499999999996]
```

Group 6: social_distancing = True , second_track = True , mass_testing = False




```
Mean: 0.14763333333333334
95% confidence interval = [0.018925, 0.33707499999999996]
```

Results & Significance analysis

```
1 # final values of different models
```

```
final_results = np.array([[full_tracks[:,i][t][-1] for t in range(30)] for i in range(8)])
```

Reports: Mean, Median, and confidence interval

```
report = [["Group " + str(i)] for i in range(8)]
total_mean = sorted([np.mean(final_results[i]) for i in range(8)])
for i in range(8):
  report[i].append(full_status[i][20:25])
  report[i].append(full_status[i][42:47])
  report[i].append(full_status[i][64:])
  report[i].append(np.mean(final_results[i]))
  report[i].append(np.median(final_results[i]))
  report[i].append(np.quantile(final_results[i], 0.025))
  report[i].append(np.quantile(final_results[i], 0.975))
  report[i].append(total_mean.index(np.mean(final_results[i])) + 1)
```

```
print(tabulate(report, headers=["Social distancing", "2nd track", "Mass testing", "Mean", "Median", "2.5th percentile", "97.5% perc
```

	Social distancing	2nd track	Mass testing	Mean	Median	2.5th percentile	97.5% percentile
Group 0	False	False	False	0.492567	0.503	0.393825	0.564325
Group 1	False	False	True	0.489517	0.49325	0.368363	0.575625
Group 2	False	True	False	0.227217	0.2145	0.099375	0.371837
Group 3	True	False	False	0.265417	0.2365	0.088525	0.454487
Group 4	False	True	True	0.193567	0.209	0.0142375	0.335762
Group 5	True	False	True	0.263967	0.25075	0.051475	0.529237
Group 6	True	True	False	0.152633	0.14325	0.0320875	0.325475
Group 7	True	True	True	0.147633	0.12375	0.018925	0.337075

Reports: T-tests significance

```
ttest = [["Group " + str(i)] for i in range(8)]

for i in range(8):
```

```
for i in range(8):
  for j in range(8):
    if i == j:
      ttest[i].append("----")
    else:
      ttest[i].append(format(sts.ttest_ind(final_results[i], final_results[j])[1], '.12f'))
```

```
print(tabulate(ttest, headers=["Group " + str(i) for i in range(8)], tablefmt='orgtbl'))
```

```
|         | Group 0        | Group 1        | Group 2        | Group 3        | Group 4        | Group 5        | Group 6
|---------+----------------+----------------+----------------+----------------+----------------+----------------+----------------
| Group 0 | ----           | 0.814840724847 | 0.000000000000 | 0.000000000000 | 0.000000000000 | 0.000000000018 | 0.000000000000
| Group 1 | 0.814840724847 | ----           | 0.000000000000 | 0.000000000000 | 0.000000000000 | 0.000000000031 | 0.000000000000
| Group 2 | 0.000000000000 | 0.000000000000 | ----           | 0.151215585594 | 0.155539991571 | 0.217353408761 | 0.000997943925
| Group 3 | 0.000000000000 | 0.000000000000 | 0.151215585594 | ----           | 0.016609770345 | 0.966330277406 | 0.000139532732
| Group 4 | 0.000000000000 | 0.000000000000 | 0.155539991571 | 0.016609770345 | ----           | 0.032036630620 | 0.105977017199
| Group 5 | 0.000000000018 | 0.000000000031 | 0.217353408761 | 0.966330277406 | 0.032036630620 | ----           | 0.000608512899
| Group 6 | 0.000000000000 | 0.000000000000 | 0.000997943925 | 0.000139532732 | 0.105977017199 | 0.000608512899 | ----
| Group 7 | 0.000000000000 | 0.000000000000 | 0.001463588011 | 0.000184213742 | 0.093569477989 | 0.000671195340 | 0.844233049232
```