Stony Brook University

AMS 562

Project Final Report

Infectious disease models

Submitted by:

Karan Rajendra (115375145)

Qiankun Nie (112111032)

Table of Contents

Introduction

In our AMS 562 project, we have successfully implemented a sophisticated computational model to simulate the spread of a contagion within a virtualized population. This initiative was driven by the need to better understand the propagation of infectious diseases across different population dynamics and to evaluate the efficacy of various intervention strategies. Our model, which accurately reflects real-world social interactions, acts as a valuable testbed for examining the complex behavior of epidemics under diverse scenarios. It is versatile, allowing for the integration of various disease parameters and population behaviors.

In our developed model, the virus is simulated to spread in two countries: one country acts as the source of the virus, and with the travel of individuals between the two nations, the virus could potentially spread over a long distance to the other country. Each individual in the country is represented as an agent, categorized into one of five states: Susceptible (healthy but vulnerable to infection), Infected (disease carriers capable of infecting others), Recovered (formerly infected and now immune), Died(leave the system) and Vaccinated (immune without prior infection). The contagion spreads from infected agents to their susceptible counterparts, contingent on the infectiousness of the disease. The simulation begins with a single infected agent within a predominantly susceptible population. It progresses on a daily basis, with each agent's state being updated, until no infected agents remain. This implementation provides a robust framework for understanding and predicting the dynamics of disease spread.
The code contains numerous optimizations for details. Despite the complexity of the simulation, when handling data for 300,000 individuals, the simulation still runs in a short amount of time. The overall time complexity of the simulation is kept within O(n^2).

Techniques and Tools

In our project, we have meticulously harnessed the capabilities of object-oriented programming (OOP) in C++ to engineer a simulation that is both robust and adaptable. The utilization of OOP principles has been pivotal in crafting an architecture that not only simulates complex biological and social systems but also remains amenable to iterative enhancements and rigorous testing.

Object-Oriented Design
Class Abstraction: We designed classes that abstract real-world concepts into programmable entities, such as `Individual` for a person and `Population` for a group of people. This abstraction facilitates a clear conceptual mapping from the real-world problem to our computational model.
Encapsulation: By encapsulating related data and behaviors within classes, we minimized interdependencies, which promoted a clean, maintainable code structure. For instance, each `Individual` manages its own state independently, interfacing with the rest of the simulation through well-defined methods.

Probabilistic Modeling
Random Number Generation: Central to our simulation's realism is the use of random number generators (RNGs) that introduce stochastic variability into the disease spread. We leveraged distributions provided by C++'s `<random>` library to model various probabilities, such as the chance of infection upon contact or the likelihood of recovery on any given day.

Simulation Tools
Compiler and Environment: We utilized specific C++ compilers, e.g. GCC, Clang within the development environment, e.g.Visual Studio to compile and debug our code.

Analysis and Visualization:
For analyzing the simulation output, we used [tools/libraries, e.g., Python with Matplotlib, R, Excel] to process the data and produce visualizations that convey the simulation results clearly and effectively.

Methodology

In developing our simulation, we employed a rigorous object-oriented programming approach, utilizing the robust capabilities of C++. The cornerstone of our simulation lies in two pivotal classes that model the dynamics of disease spread: `Population` and `Individual`.

→Population Class:

Purpose:
The Population class is designed to model a population of individuals in the context of an epidemic. It encapsulates the collective dynamics of disease spread, recovery, vaccination, and mortality within the population. This class serves as the primary driver for simulating the progression and impact of an epidemic on a group of individuals.

Functionality:
Initial Population Setup: Initializes a population of Individual objects, setting initial infection states based on a given infection rate.
Disease Progression Simulation: Simulates the day-to-day progression of the disease within the population, accounting for infection, recovery, vaccination, mutation, and death.
Statistical Tracking and Reporting: Tracks and reports various statistics such as the number of susceptible, infected, recovered, vaccinated, and deceased individuals, as well as the mutation count of the virus.

Key Methods:
Population(): Initialize the population class to generate a specified number of individuals based on input requirements, distributing them randomly across two countries. All individuals are stored as pointers within the std::vector<std::vector<Individual*>> individuals.

population_travel(): In the simulation, individuals from the two countries travel to each other every day, employing this method to simulate travel functionality.

updatePopulationState(): This is the main method for advancing the simulation by one day. It involves calling other methods like die(), contagion(), vaccination(), and clean_state(), to update the state of the population.

die(): Simulates the death of infected individuals based on the mortality rate (dieRate), removing them from the population.

vaccination(): Administers vaccines to individuals based on the vaccination rate, changing their state to vaccinated. For authenticity in the simulation, the development of a vaccine takes a certain amount of time. Vaccination only begins 30 days after the discovery of the virus.

contagion(): Facilitates the spread of the infection within the population. It identifies infected individuals and triggers the infection process in their contacts.

_contagion(Individual& individual, size_t idx, short int i): A helper method used within contagion() to simulate the infection spread from a specific infected individual to their contacts.
clean_state(): Resets the state at the end of each day, likely clearing temporary data like contact lists.

OutPutPopulationStates(): Outputs the current state of the population, providing a snapshot of the epidemic's status. It also asserts the consistency of the population count across different states.

is_clear() const: Checks if the disease has been eradicated from the population.
ofstream_to_m(size_t ...): Outputs the statistical data to a file, presumably for further analysis or visualization.

start_stimulate(): reads the data from the text file input and start the simulation.

→ Individual Class:
Purpose:
The Individual class is designed to model a single entity within a population being studied for disease spread dynamics. This class encapsulates all the necessary attributes and behaviors of an individual in the context of an epidemic, including their susceptibility to infection, social activity level influencing their interaction rates, and their response to disease progression and mutation.

Functionality:
State Management: Manages the state of an individual, including Susceptible, Infected, Recovered, Vaccinated, and Died. These states represent the different phases an individual can go through in the context of a disease.
Social Interaction Simulation: Tracks and simulates the social activity level of an individual, influencing their likelihood of disease transmission and reception.
Mutation Tracking: Handles the concept of virus mutation, tracking how many times the virus has mutated within an individual and adjusting the individual's response to the disease accordingly.
Recovery and Vaccination Dynamics: Manages the process of recovery and vaccination, altering the individual's state based on these events.

Key Methods:
infection(double infectionRate, Individual& sick_individual): Simulates the process of infection. This method accounts for the current state of the individual, the infection rate, and the mutation count of the virus from the sick individual. It determines whether the individual becomes infected or reverts to susceptible due to a mutated strain.

recover(double recoveryRate): Simulates the recovery process. It uses the recovery rate to determine if an infected individual will recover and adjusts their state and mutation count accordingly.

vaccinate(): Implements the vaccination of an individual, changing their state to Vaccinated. getSocialActivityLevel() const: Returns the individual's social activity level, a key factor in determining their interaction rate and susceptibility to infection.

getState() const: Returns the current state of the individual.

die(): Changes the state of the individual to Died, simulating death due to the disease.

getMutation_count() const, getMutation_time() const: These methods provide information about the virus mutation within the individual.

→ State Class:

Purpose:
The primary purpose of the state class is to serve as a base class for other classes in the simulation, particularly the Individual and Population classes. It provides a common framework for handling the various states an individual can be in during the disease simulation (such as Susceptible, Infected, Recovered, Vaccinated, and Died) and maintains global counts for each of these states across the entire population.

Functionality:
State Enumeration: The State enum defines the possible states an individual can have in the context of the disease simulation. These states are essential for tracking the progression and impact of the disease within the population.
Static Count Variables: The class includes static variables to keep track of the number of individuals in each state (Susceptible_count, Infected_count, Recovered_count, Vaccinated_count, Died_count). These counts are global and shared across all instances of the derived classes, providing an overall view of the population's status in the simulation.

→ main(int argc, char* argv[])

Purpose:
The main() function is primarily used to input various parameters from a text file and to initialize the entire simulation.

Functionality:
In the main() function, the program checks the correctness of the input and establishes a connection between each parameter and its name using an unordered_map.

Observations and Results

In our repeated simulations, we saw a variety of results that showed the complex ways diseases spread in a simulated population. We explain the important trends we noticed, how

changing different factors affected the outcomes, and some unexpected findings we made during our study.
We primarily simulated three types of diseases: flu, COVID, and Ebola. I found basic data on the spread of these viruses on authoritative websites and used it for simulation. Especially with viruses like Ebola, due to its high fatality rate, certain random events in the simulation can significantly impact the outcomes, making the outcomes unpredictable.

→Notable Patterns

Infection Peaks: The simulation consistently demonstrated that without intervention, the number of infected individuals peaks sharply before gradually declining, following a classic epidemiological curve.
Super spreader Events: Certain configurations led to super spreader events, where a single highly social individual significantly accelerated the spread of the disease.
Herd Immunity: As more individuals recovered and became immune, the rate of new infections slowed, suggesting the onset of a form of herd immunity within the population.

→Impact of Parameters

Social Activity Level: Increasing the average social activity level led to faster and more widespread transmission, emphasizing the role of social behavior in epidemic dynamics.
Recovery Rate: Higher recovery rates flattened the infection curve, reducing peak infection load and spreading it over a longer period.
Initial Infection Rate: Varying the initial infection rate showed a threshold effect; below a certain level, the disease failed to take hold, while above it, the disease spread rapidly.

→Unexpected Findings

Mutation Impact: The introduction of a mutation rate resulted in second waves of infection, a phenomenon that has profound implications for long-term disease management.
Vaccination Strategies:Targeted vaccination of highly social individuals was more effective than random vaccination, highlighting the importance of strategic public health planning.

→Data Visualization

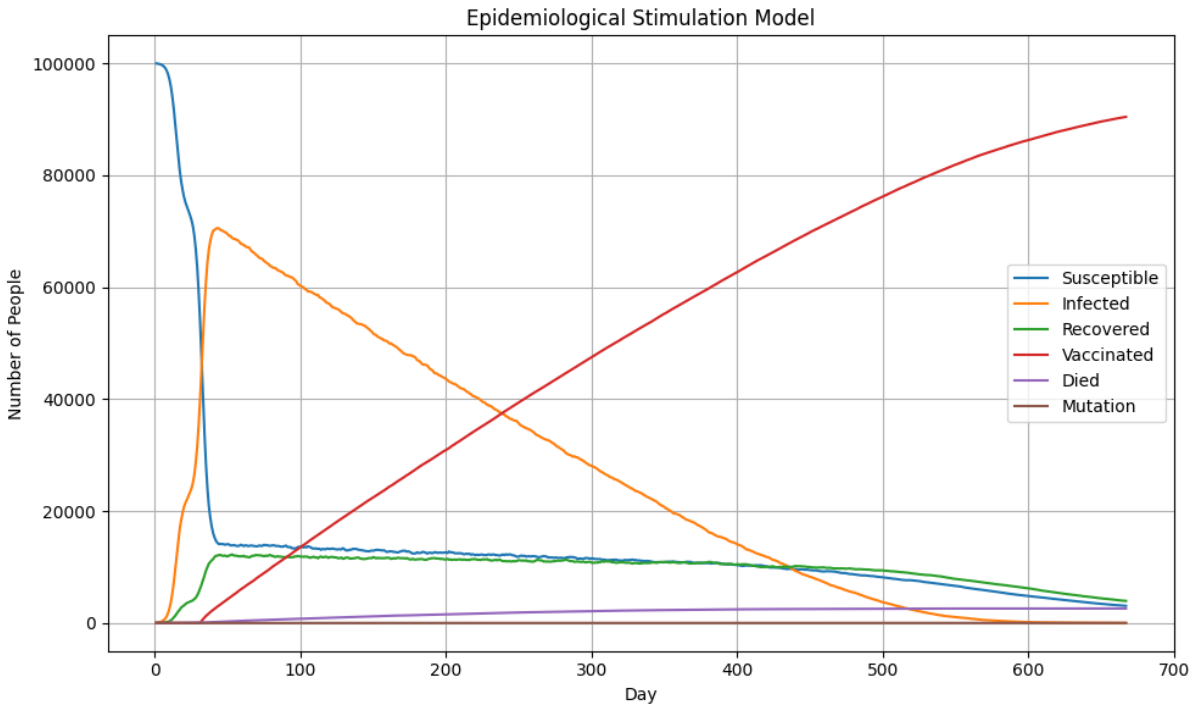
Graphs and charts illustrating these results were generated using [tool/libraries, e.g., Matplotlib, R], providing a visual narrative of the simulation outcomes. These visual aids have been instrumental in dissecting the complex interplay of variables and in communicating our findings with clarity and impact.

The observations gleaned from our simulation have provided us with valuable insights into the dynamics of disease spread. These results underscore the critical importance of various factors that can either exacerbate or mitigate the trajectory of an epidemic and form a knowledge base from which actionable strategies can be devised.

Output:

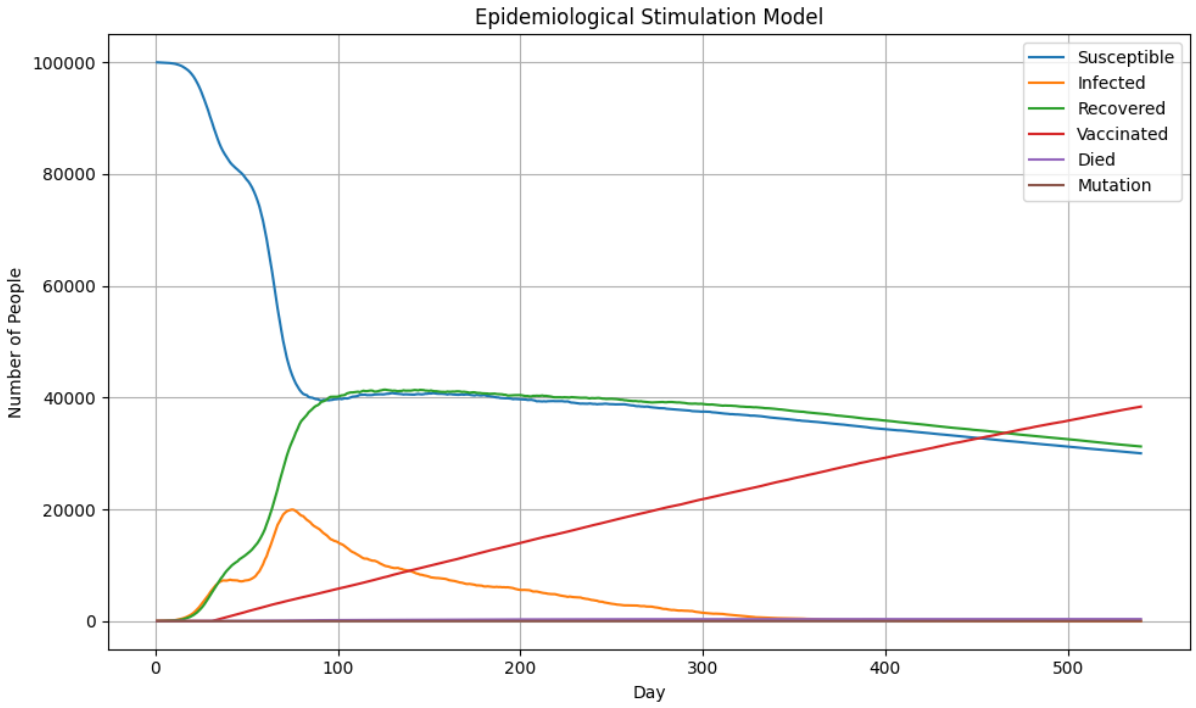
Case1: COVID on 100,000 individuals

This is a virus with a high vaccination rate, a transmission rate greater than the recovery rate, and a relatively low fatality rate.



We can observe that due to the transmission rate being much higher than the self-recovery rate, the elimination of the virus heavily relies on the vaccination rate. A higher vaccination rate directly reduces the number of deaths.

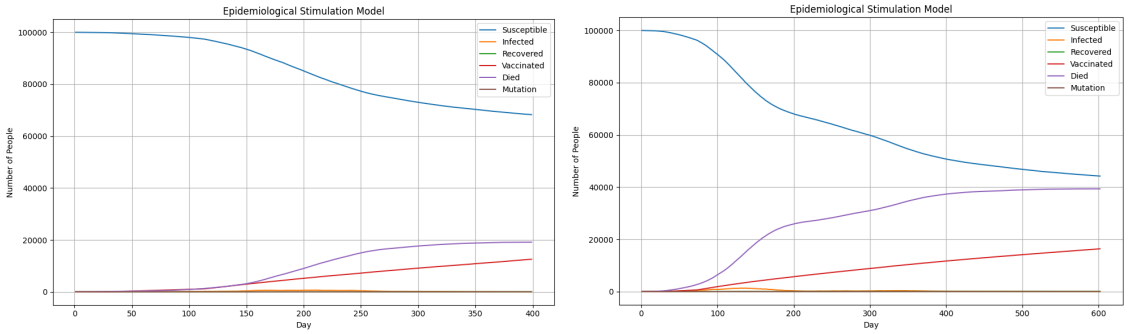
Case2: Flu on 100,000 individuals
This is a virus with a relatively high vaccination rate, a transmission rate lower than the recovery rate, and a low fatality rate.



We can observe that despite the transmission rate being lower than the self-recovery rate, due to the virus mutating (16 times), the role of the vaccine remains crucial and greatly accelerates the pace of virus elimination.

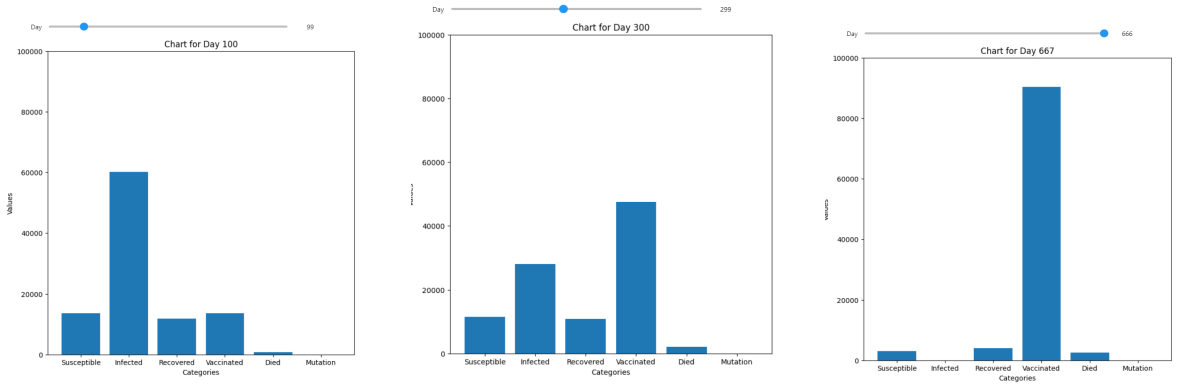
Case3: Ebola on 100,000 individuals

This is a virus with a very low vaccination rate, extremely high fatality rate, high infectivity, and an extremely low self-recovery rate.



I observed that due to the immense fatality rate, the simulation results for the spread of this virus have a high level of uncertainty, with two probable outcomes. Different outcomes depend on whether the host dies or transmits the virus when it spreads across countries.

The results can also be dynamically presented using bar charts and simulated regional virus status. However, transferring these dynamic displays from Python to a PDF file is challenging. Here are some examples of the results:



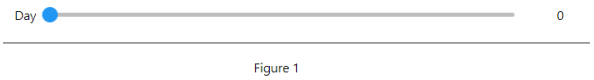


Figure 1

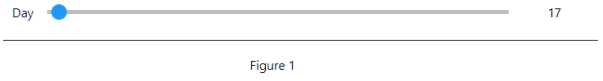


Figure 1

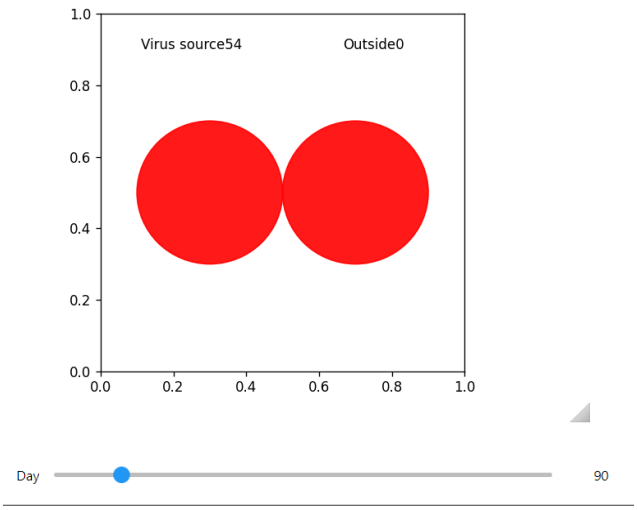


Figure 1

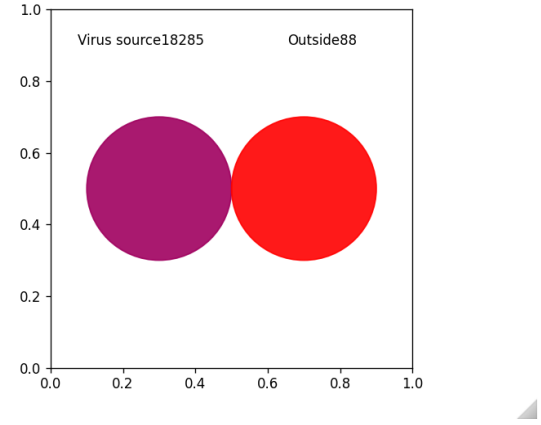
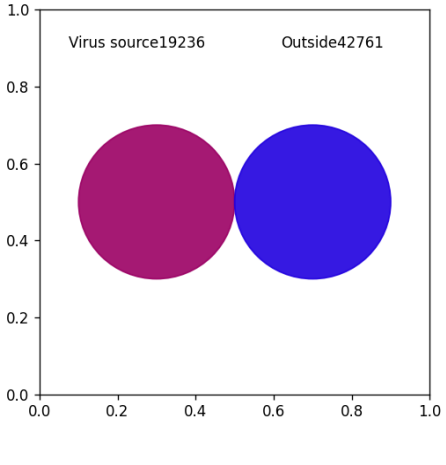
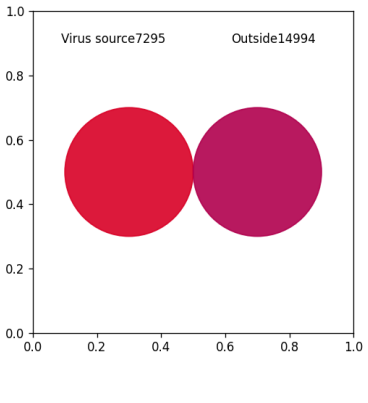


Figure 1





Conclusion
The culmination of our project has yielded a comprehensive simulation that sheds light on the complexities of epidemic spread within a virtualized population. Our findings underscore the critical importance of individual behavior patterns, interaction rates, and the role of public health interventions in altering the course of a disease.

The results of the simulation have made it abundantly clear that epidemics are highly sensitive to initial conditions and that the timing and type of interventions can have a substantial impact on the outcome. This reinforces the notion that early and targeted responses are crucial in managing public health crises.
Furthermore, the project has highlighted the utility of computational models as valuable tools in epidemiological research. By simulating a variety of scenarios, we can predict potential outcomes and devise strategies to combat real-world health challenges effectively.

Limitations
While our simulation has provided meaningful insights, we acknowledge certain limitations. The model's assumptions, such as constant rates of contact or uniform susceptibility, may not capture the full spectrum of variability seen in actual populations. Moreover, the current model does not account for geographic spread or the impact of environmental factors, which are significant aspects of pandemic behavior.

Future Directions
Looking ahead, we are poised to embark on several enhancements to the model:

Adding heat maps: To visually represent the distribution and intensity of the disease spread across different areas or groups within the simulated population.

Plotting the graphs: To graphically display the progression of the disease over time, showing trends and changes in infection rates, recovery rates, and other vital statistics.

Implementing lockdown feature: To simulate the effect of public health interventions, like lockdowns, on the spread of the disease.

Implementing mutation feature: To model the real-world scenario where viruses mutate, affecting recovered or susceptible individuals.

Correlating Deaths with Other Variables: To understand the relationship between mortality and other variables in the simulation, such as infection rates, healthcare capacity, or the presence of lockdowns and social distancing measures

Contributions by Team members

- Qiankun and Karan collaborated on crafting the project's pipeline. Together, we formulated the simulation logic, determined the necessary methods and classes, refined the output process, and devised strategies for visually presenting the results.

- For a more detailed breakdown of our work: Qiankun and Karan concentrated on implementing the 'Population' class, which contains methods that globally manipulate the simulation, we also worked on the 'Individual' class, which is crucial for local change within the stimulation. Both components were integral to the primary simulation.

- We also work on utilizing ofstream to convert simulation data to .m format and plotting in Python, divided the task evenly. Collectively, we developed a method capable of dynamically altering the graph based on date. Additionally, we collaborated on plotting the curves representing all the crucial features of the simulation.

- Finally, we worked collaboratively to create the presentation materials and collectively completed the PDF report.

References

OpenAI. (2023). *ChatGPT* (Mar 14 version) [Large language model]. https://chat.openai.com/chat

Chapter 5 of Projects for Scientific Programming in C++ and Other Languages by Victor Eijkhout.

New York State Department of Health. COVID-19 Information. New York State Department of Health, www.health.ny.gov/.

Centers for Disease Control and Prevention. "Seasonal Influenza (Flu)." CDC, www.cdc.gov/flu/index.htm.

World Health Organization. "Ebola Virus Disease." WHO, www.who.int/health-topics/ebola/.