University of Amsterdam

Masters Thesis

Mathematically Modeling the Interactions Between Phages, Bacteria, and the Environment

Examiner:
Dr. Jaap Kaandorp
Author:
Victor Piaskowski
Supervisor:
Dr. Matti Gralka
Assessor:
Dr. Yuval Mulla

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computational Science

in the

Computational Science Lab
Informatics Institute

May 2025

Declaration of Authorship

I, Victor Piaskowski, declare that this thesis, entitled 'Mathematically Modeling the Interactions Between Phages, Bacteria, and the Environment' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at the University of Amsterdam.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date: May 26, 2025

"All models are wrong, but some are useful"

George E. P. Box

UNIVERSITY OF AMSTERDAM

Abstract

Faculty of Science
Informatics Institute

Master of Science in Computational Science

Mathematically Modeling the Interactions Between Phages, Bacteria, and the Environment

by Victor Piaskowski

Include your abstract here Abstracts must include sufficient information for reviewers to judge the nature and significance of the topic, the adequacy of the investigative strategy, the nature of the results, and the conclusions. The abstract should summarize the substantive results of the work and not merely list topics to be discussed. Length 200-400 words.

Acknowledgements

I would like to thank my parents for eternally loving me and for financially supporting me through my Bachelor and Master studies, for without them, I wouldn't know where my life would be right now. Thank you to Dr. Matti Gralka for the weekly meetings and teaching me everything about phages and bacteria. Every meeting was always insightful, productive, and informative. I will forever be amazed at how he can remember which paper talks about which topic, and how he always had a paper for every topic. Thank you to Sofia Blaszczyk for finding this opening and suggesting that I email Dr. Gralka for an introductory meeting, and for acting as my rubber duck programming buddy, and watching my cringe screen recordings that I sent her at 2am showcasing various demos of my code. If I hadn't followed Dr. Rik Kaasschieter's and Dr. Martijn Anthonissen's courses "Introduction Computational Sciences" and "Numerical Linear Algebra" in my Bachelors, I would not have been interested in Computational Sciences and would not have found the MSc Computational Sciences program, as Computational Sciences fits my interests and skill sets better than any other program I could have taken. For they have forever altered my career trajectory. Thank you to Sarah Flickinger for showing me the research that she has been doing in the lab. She allowed me to really connect my research and models to real life, reminding me that what I am doing has real life use cases than just a purely theoretical or programming challenge. And finally, thank you to all of my friends for keeping me sane and helping me through both of my programs.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

ABM	**A**gent **B**ased **M**odelling
ARD	**A**rms **R**ace **D**ynamic
BVP	**B**oundary **V**alue **P**roblem
CBASS	**C**yclic oligonucleotide-**B**ased **A**ntiphage **S**ignalling **S**ystems
CRISPR	**C**lustered **R**egularly **I**nterspaced **S**hort **P**alindromic **R**epeats
DDE	**D**elay **D**ifferential **E**quation
DNA	**D**eoxyribo**N**ucleic **A**cid
FSD	**F**luctuating **S**election **D**ynamics
GUI	**G**raphical **U**ser **I**nterface
OD	**O**ptical **D**ensity
ODE	**O**rdinary **D**ifferential **E**quation
PDE	**P**artial **D**ifferential **E**quation
RNA	**R**ibo**N**ucleic **A**cid
SIE	**S**uper**I**nfection **E**xclusion
SNP	**S**ingle **N**ucleotide **P**olymorphism
TAB	**T**ail **A**ssembly **B**locker
UvA	**U**niversitiet **v**an **A**msterdam

A table of Symbols and Parameter Values can be viewed in Appendix A: Equation Parameters and Appendix E: Parameter Values Used.

Chapter 1

Methods

1.1 Project Overview

To help complete this Master thesis, I created various tools that would help create the final model outputs. The project is divided into four parts. The first part is the tool that a user can use to design and create the network of agent interactions. The second part is the simulation framework that handles the data and runs the ODE solving method. The third part is a dashboard that runs in the browser. The dashboard allows for a user to interact with the model, for example by changing parameter and environment values, and run some basic simulations and receive different plots as output. The final part allows the user to download the simulation data to create their own custom graphs and analyses.

A flowchart showing the user-system interactions can be seen in Appendix C: Flowchart of User and System Interactions.

1.2 The Golden Model

The default model, the "Golden model", sourced from Geng et al. [4], describes the interactions between Resources, Uninfected bacteria, Infected bacteria, and Phages. All plots and simulations will use this model by default, unless stated otherwise.

1.2.1 The Golden Model

where R is resources, U is uninfected bacteria, $I_{1,...,M}$ is the infected stage of the bacteria, and P is the phage population.

$$\frac{dR}{dt} = -e \cdot g(R, v, K) \cdot (U + \sum_{i=1}^{M} I_M) + w^i - w^o \cdot R \tag{1.1}$$

$$\frac{dU}{dt} = g(R, v, K) \cdot U - r \cdot U \cdot P - w^o \cdot U \tag{1.2}$$

$$\frac{dI_1}{dt} = r \cdot U \cdot P - \frac{M}{\tau} \cdot I_1 - w^o \cdot I_1 \tag{1.3}$$

$$\frac{dI_k}{dt} = \frac{M}{\tau}(I_{k-1} - I_k) - w^o \cdot I_k \text{ for } k = 2, \ldots, M \tag{1.4}$$

$$\frac{dP}{dt} = \beta \cdot \frac{M}{\tau} \cdot I_M - r \cdot (U + \sum_{i=1}^{M} I_M) \cdot P - w^o \cdot P \tag{1.5}$$

$$g(R, v, K) = \frac{v \cdot R}{R + K} \tag{1.6}$$

Equation 1.6: The golden model sourced from Geng et al. [4]. The text in red has been added to the model, adding (the wash-in) fresh resources (ω^i) and the removal (wash-out) of agents (ω^o). The washin is not dependent on the current resource population, as it is a constant rate being added. By default these values are 0. A summary of the parameters can be found at Table 3.1.

The model describes three biological processes, cell consumption of resources and growing, phage/cell encounters and infection, and cell lysis. The cell growth process is described by $g(R, v, K)$, the instantaneous growth rate dependent on the Monod equation, where v is the maximal growth rate and K is the Monod constant. The consumption rate of a resource by a bacteria is e.

Once infected by a phage, the bacteria goes from U to I_1. The bacteria goes through M stages of infection $I_1, \ldots, I_M$ before lysing, where the bacteria goes from state I_k to state I_{k+1} with equal transition rate $\frac{M}{\tau}$. The probability of a successful infection of a cell is r. After a bacteria lyses after stage I_M, β phages are released, the burst size of the phage.

However this model is specifically designed for a $1 \times 1 \times 1$ model. In order to adapt this model to fit an $p \times b \times r$ model, the model needs to be slightly adapted. There are other changes that can be made to the model, for example by adding a washin rate ω^i, where resources are constantly being introduced, and a washout rate ω^o where all agents are washed out at a constant rate. These changes are highlighted in Equation (1.6) in red.

1.2.2 The Adapted Golden Model

Equation (1.12) accounts for the interactions of multiple agents. Each agent is a sum of all interactions. When resource R_1 is being consumed by bacteria B_1 and B_2, R_1 is reduced by the sum of the resources used by B_1 and B_2. Likewise, if bacteria B_1 is being

infected by phage P_1 and P_2, then the uninfected B_1 population is reduced by the sum of infections from P_1 and P_2.

$$\frac{dR_r}{dt} = -\sum_{b\in B} e_{b,r} \cdot g(R_r, v_{b,r}, K_{b,r}) \cdot (U_b + \sum_{i=1}^{M} I_{i_b}) + w_r^i - w^o \cdot R_r \tag{1.7}$$

$$\frac{dU_b}{dt} = U_b \cdot \sum_{r\in R} g(R_r, v_{b,r}, K_{b,r}) - U_b \cdot \sum_{p\in P} r_{p,b} \cdot P_p - w^o \cdot U_b \tag{1.8}$$

$$\frac{dI_{b_1}}{dt} = U_b \cdot \sum_{p\in P} r_{p,b} \cdot P_p - \frac{M}{\tau_b} \cdot I_{b_1} - w^o \cdot I_{b_1} \tag{1.9}$$

$$\frac{dI_{b_k}}{dt} = \frac{M}{\tau_b}(I_{b_{k-1}} - I_{b_k}) - w^o \cdot I_{b_k} \text{ for } k = 2, \ldots, M \tag{1.10}$$

$$\frac{dP_p}{dt} = \sum_{b\in B} \beta_{p,b} \cdot \frac{M}{\tau_b} \cdot I_{b_M} - r_{p,b} \cdot (U_b + \sum_{i=1}^{M} I_{b_i}) \cdot P_p - w^o \cdot P_p \tag{1.11}$$

$$g(R_r, v_{b,r}, K_{b,r}) = \frac{v_{b,r} \cdot R_r}{R_r + K_{b,r}} \tag{1.12}$$

.

Equation 1.12: Probability of phage infection $r_{p,b}$ is not to be confused with R_r, short for Resource r. The interactions are a sum of all interactions due to all interactions taking place at the same time.

1.2.3 Network Creation Tool

The network creation tool is the first step that a user needs to use, and it revolves around using a GUI tool built to create and edit the graph representation of the agent interaction. Numerous interactions occur between agents in a microbial environment. However, not every agent can and will interact with one another. Based on which agents interact with one another, a network topography representing the agent interactions and dynamics can be created.

Every node represents a unique agent, and each agent has their own intrinsic properties. The user can intuitively define agents, their interactions, environmental and model settings using the GUI tool. This tool allows users to quickly and intuitively define agents and their attributes, agent interactions and their attributes, environmental data, and model settings. An edge links two agents together if there is an arbitrary interaction occurring between the agents, with the properties exhibited in the interaction dependent on the interacting agents. Self interactions are allowed in the network. There is an environment node that is used to store global environmental data, such as the temperature and pH of the system. The settings node holds information such as simulation length, max time step, and the type of ODE solver to use. The tool provides functionalities for

adding, editing, and visualizing nodes and edges, as well as importing and exporting the network structure.

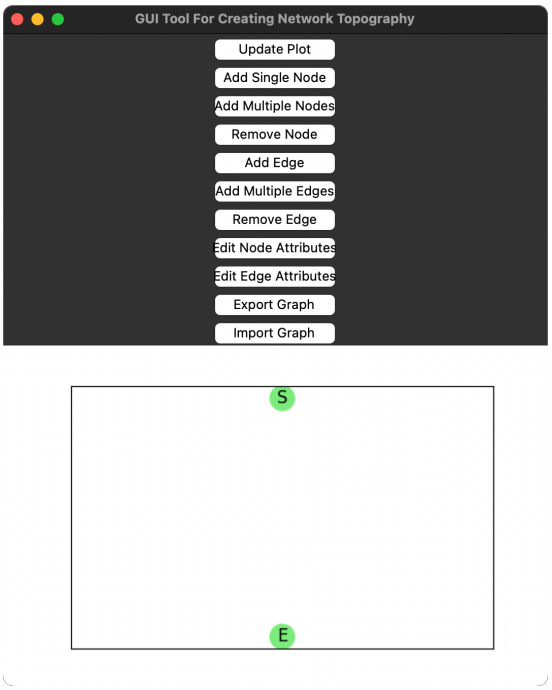
Once the user is happy with the graph shape, they can export the network representation for use in Simulation Framework, Visualization Dashboard, and Custom Visualizations and Analyses. The most important part is that the user defines the shape and the attributes of the network, as that can't be edited in part 2 onwards. It is possible go back to the network creation tool and upload the graph to the tool to be able to edit the network representation.
The user can edit the values of the attributes in Visualization Dashboard, so the parameter values do not have to be perfect. As such, the user does not need to keep on using the GUI tool to edit parameter values.

Figure 1.1a shows the layout of the GUI tool. Figure 1.1b shows an example network that can be created. There are numerous buttons that can be used to edit the graph, for example adding or removing nodes and edges. By default, an environment node holding parameters such as pH and temperature is added. A settings node is added as well, holding settings data to be used for the solver, like the type of solver or simulation length. Manually adding nodes and edges can get tedious and repetitive for large graphs, so the user can add multiple nodes and edges at the same time. Nothing can interact with the environment and setting node, as they are used to hold data about the environment and network solver. The user can alter the default attribute name and value by importing the GUI tool class and overriding the method implementation implementing the default names and values. The user can self-decide which parameter values to give, and if and how the parameter values are randomized.

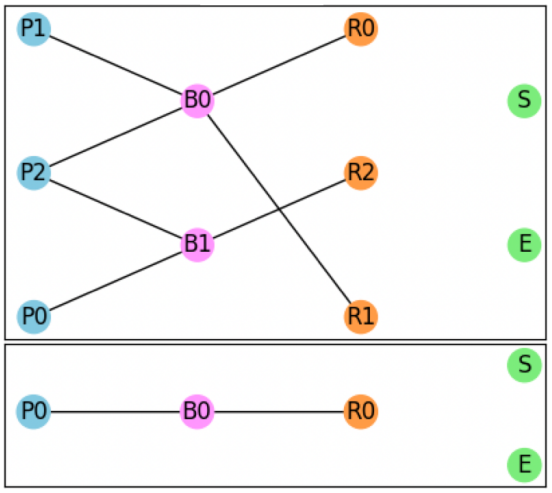
1.2.4 Simulation Framework

The user provides an ODE model and the network topography as input to the framework. The simulation framework deals with handling the input, output, collecting and storing of the simulation input and output. The framework uses SciPy's [5] *solve_ivp()* numerical solver [6] to simulate the provided ODE equations and calculate the population levels through time. The user receives two outputs from the framework. The first output is an array of time values that the solver used to calculate the population count. The second output is an array containing the population count at each time step for every agent.

In order to facilitate more complex model behavior, "hidden" agents can be added to the simulation, where the hidden agents represent states that a "visible" agent can be in. Such an example would be the distinction between uninfected and infected bacteria. Each bacteria agent b_i would contain two hidden sub-agents, called sub-agent states, an



(a) The GUI tool as seen on the startup of the program.



(b) An arbitrary $3 \times 2 \times 3$ and $1 \times 1 \times 1$ network with each node representing a phage, bacteria, or resource, with arbitrary interactions occurring between them. Although not used here, edges between the same agent types and self loops are allowed. This network topography along with a $1 \times 1 \times 1$ network will be used in the Methods and Experiments and results sections.

uninfected sub-agent state $b_{i_{\text{uninfected}}}$ and infected sub-agent state $b_{i_{\text{infected}}}$. It is possible to create further sub-agents, by having a bacteria go through M stages of infection. So each infected agent has sub-agent $b_{i_{\text{infected}_k}}$, where $1 \leq k \leq M$.

In the network model you would explicitly create a $3 \times 2 \times 3$ network, but when passing the network to the simulation framework, you would tell the framework to model the subagents. The user's ODE model has to correctly model each (hidden) agent and correctly handle the changes in states.

Even though the user might submit a $3 \times 2 \times 3$ model, if the user follows the uninfected and infected classification, where each bacterium goes through 4 stages, the ODE model and framework will explicitly be modelling 3 phages, 10 bacteria (2 uninfected $+ 2{\cdot}4 = 10$ infected bacteria), and 3 resources.

The user might also be interested in modelling a resource reservoir in a chemostat, where resources go from an external reservoir not accessible by the bacteria to the virtual chemostat ready to be consumed by the bacteria. 3 hidden resource agents would be added to the system, where the resources would model the transfer of resources from the reservoir to the simulation environment. The provided ODE model will have to correctly model and transfer the resources from $R_{i_{\text{reservoir}}}$ to $R_{j_{\text{chemostat}}}$.

1.2.5 Visualization Dashboard

The third part involves analyzing and visualizing the simulation results on an interactive Dash Plotly [7] dashboard. The user can use a dashboard built using Plotly Dash to interact with the solver and network. The user can interact with the solver and network by changing parameter, environment, and setting values on the fly. This allows the user to quickly change parameter values and test different situations. The dashboard includes various starter plots that allow the user to test the model. As output, the dashboard will show interactive plots so that the user can analyze the system.

The dashboard allows the user to interact with the network, the model, and some prebuilt visualizations, and is built into three logical sections. The first section allows for the user to edit the network parameters and setting values on the fly to quickly iterate through different conditions and to fine-tune parameter selection without having to rebuild the network using the GUI tool. The second section allows for the user to see how the population count evolves over time for a given initial condition and parameter values, allowing to quickly test the network input. The final section allows for the user to run more advanced analyses on the network, for example, by changing multiple parameter values and visualizing the output.

1.2.5.1 Editing Network and Parameter Values

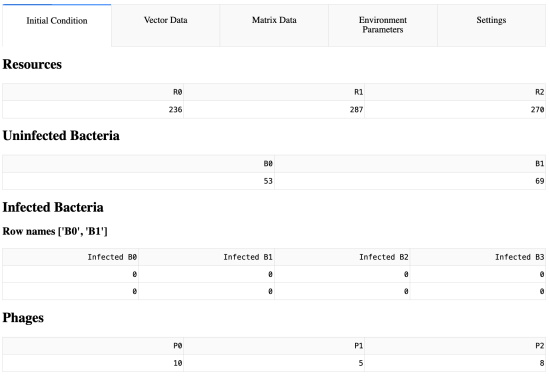
The editing network and parameter value contain five separate sections.

Initial Condition The initial condition settings panel (Figure 1.2a) allows for the user to edit the initial starting values of the agents. Each agent type has a table containing the initial population count. Extra hidden agents can be included. When a bacteria has been infected, the bacteria goes through multiple stages before lysing. Each bacteria agent starts out as uninfected, and once infected, the bacteria goes through 4 stages of infection before lysing as seen in Figure 1.2a.

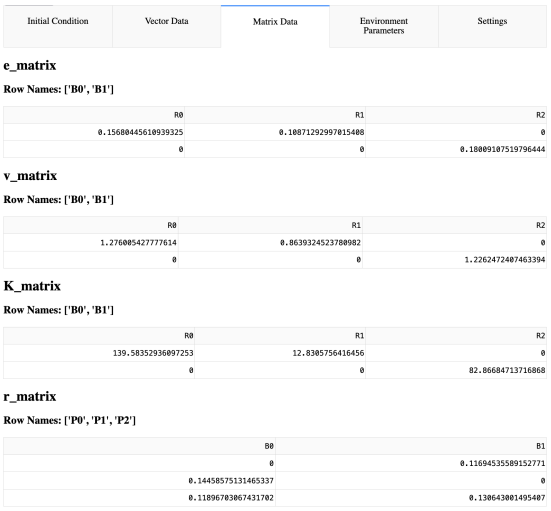
Vector Data Data that can be represented as a vector, for example the data attributed to an agent type have their own section, Figure 1.2c.

Matrix Data Data that is stored as a matrix, the data stored on edges between agents, is stored in the matrix tab (Figure 1.2b).

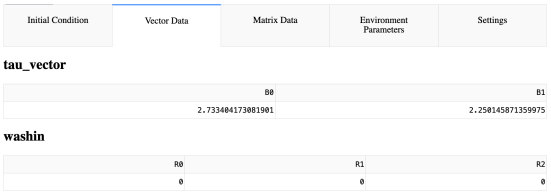
Environment and settings The environment data and settings data also have their own tab, Figure 1.2d and Figure 1.2e respectively.



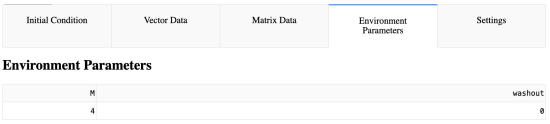
(a) The tab where the user can edit the initial conditions of the agents.



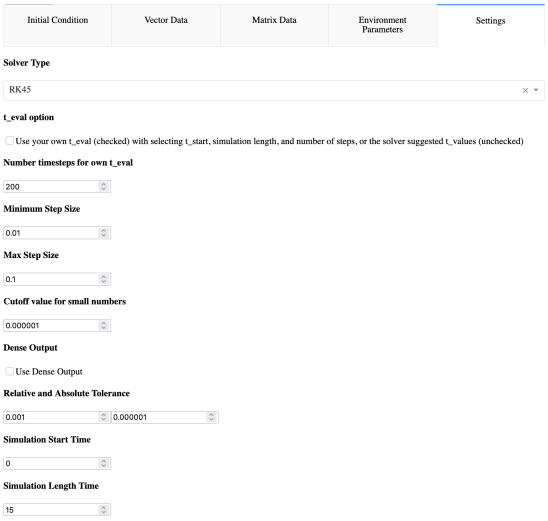
(b) The tab where the user can edit the matrix attribute values.



(c) The tab where the user can edit the vector attribute values.



(d) The tab where the user can edit the environment values.



(e) The tab where a user can edit the settings of the solver and simulation.

Figure 1.2: The tabs where the user can edit the various parameter values and control the simulation parameters

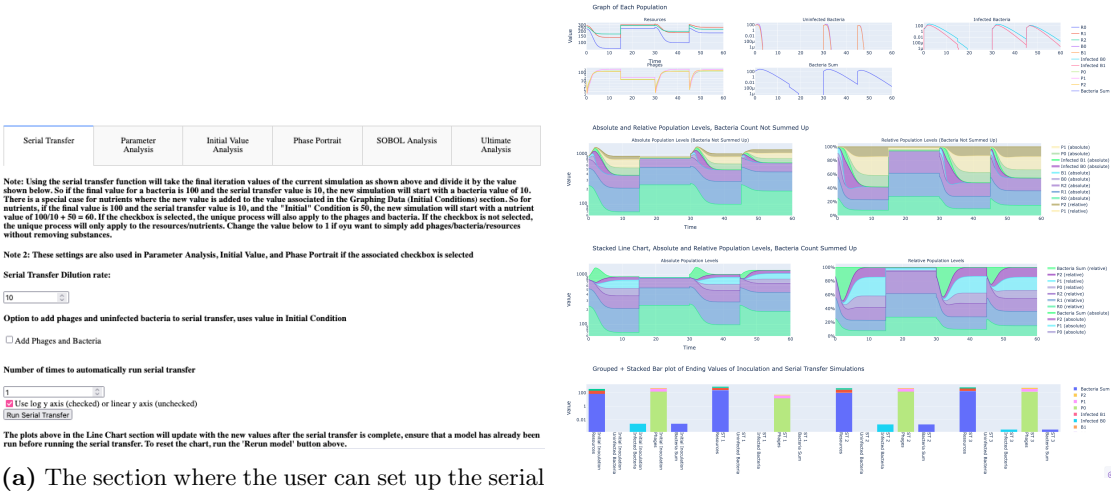
1.2.5.2 Visualization and Analysis

In the analysis section, the user can run different analysis methods to gain a greater understanding of the model. For simplicity, the visualizations only support a $1 \times 1 \times 1$ model, in order to make the analysis easier for the user, and to make it easier to analyze the visualization as the aim of the tool is to gain a deeper understanding of the interactions in a reduced complexity environment. These advanced visualizations were created with the mind of understanding a simple network. There are five different analysis and visualization methods, and one system where the user can run a large simulation on the whole network and receive an output file containing the raw simulation file data. The raw data is stored as a *parquet* file, a tabular-like data format, which when combined with Dask [8], allows for querying of the data similarly to Pandas. Parquet with Dask offers superior performance and data storage solutions that Pandas can't offer. Once queried, the user can create their own graphs and plots as they have access to the parameter values used and the raw simulation data.

Serial Transfer Serial transfer is a method employed by bacteriologist where after a set amount of time, the bacteriologist pipettes a specified amount of media (for example 10ml of liquid) containing bacteria and resources, possibly with phages, and transfers the old media into a solution containing new media. At this stage, the bacteriologist can introduce new agents, or re-introduce agents if the agent population or concentration has died out. However, usually only resources are added during the transfer process. An example would be an experiment starts with 50ml of solution. The experiment runs for 24 hours before 5ml is removed. Researchers can run various tests, such as using optical density measurements to assess bacterial density in the solution or employing a mass spectrometer to determine the concentration of the resources. The 5ml is then re-added to a new solution of 45ml containing fresh resources. The effect that this has is it creates a sort of artificial stable point. As the bacteria grow, they consume the resources found in the solution. However eventually the resources run out, and the bacteria die out due to a lack of resources. By introducing new resources at set time intervals, the bacteria can regrow and exhibit a semi-stationary behavior.

The implementation of serial transfer is slightly different. A user can select a number which will divide the population count of the agents by that number (Figure 1.3a). Then the program takes the initial condition values defined for the resources the initial condition in Section 1.2.5.1 and adds those values to the resources respectively. By selecting a checkbox, the values as defined in the initial condition box for phages and bacteria in Section 1.2.5.1 can optionally be added as well. As an example, if at the end of a simulation, there are 120 resources, 5000 bacteria, and 1000 phages remaining

and the chosen serial transfer value is 15, then the resource, bacteria, and phage values would be decreased to 8, 333.33, and 66.66 respectively. Then, if the initial condition for the resources, bacteria, and phages in Section 1.2.5.1 are 500, 80, and 10 respectively, and the checkbox is unchecked, the new population count will be 508, 333.33 and 66.66 respectively. If the checkbox is checked, the new population count will be 508, 413.33, and 76.66 respectively. These new values would be used as the new starting initial condition for a new simulation, and the run results will be appended to the previous run. As output, new graphs are created showing the runs appended to one another, with an example output shown in Figure 1.3b.



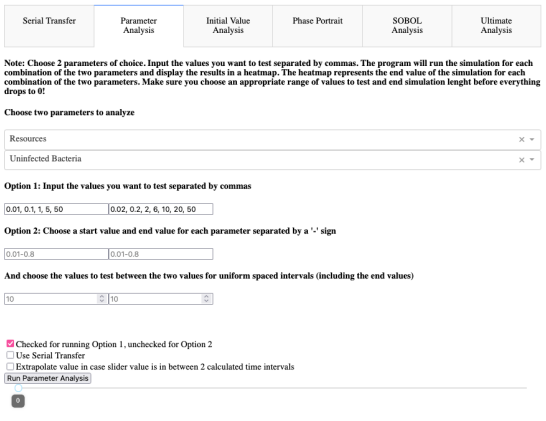
(a) The section where the user can set up the serial transfer.

(b) The output plots of serial transfer.

Figure 1.3: Serial Transfer

Parameter Analysis The parameter analysis settings tab as shown in Figure 1.4a allows the user to choose two parameters and individually run the model with the varying input values. The values that can be tested and changed include all initial condition values, vector and matrix data, and environmental data. As input, the user can select 2 parameters of choice. After the parameter name selection, the user can manually choose which parameter values they want to test or test a range of values equally spaced by selecting the number of values to test. Finally, the user can optionally run a serial transfer, where the serial transfer uses the settings found on the Serial Transfer tab.

Figure 1.4b shows the heatmap that the user can expect, one heatmap for each agent type. Each heatmap cell represents the input of 2 unique parameter values, and shows the population count for that parameter run at the time shown in the slider. As the user slides the slider, the value inside the cell updates to correspond with the selected time. Note that the heatmap color range resets for each heatmap, so similar colors across heatmaps and across time will not correspond to the same values.



(a) The options available for the parameter analysis.

(b) The output that the user can expect

Figure 1.4: Parameter Analysis

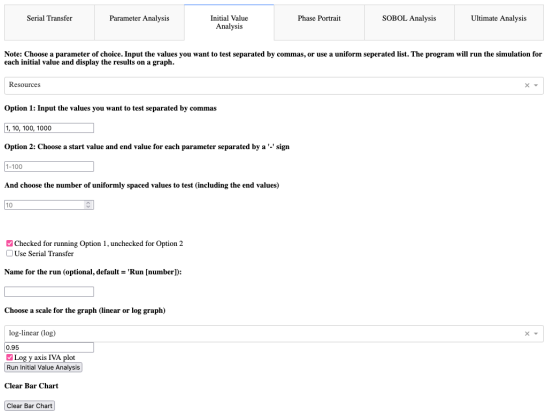
Initial Value Analysis The initial value analysis settings tab as shown in Figure 1.5a allows the user to choose a single parameter and vary the value of that parameter, visualizing how a change in parameter value affects the population count of the agents.

Figure 1.5b shows the plots that the user receives. For each agent type, there are three plots made. The left plot shows the population count through time, one line for each parameter value submitted. The middle plot takes each run and calculates the "percentage from the max value" (default value of 0.95 $\rightarrow$ 95%) reached of the peak. This value is considered the time of peak, and is used to fix some issues that can arise where the population plateaus or only keeps on rising. The initial value is plotted on the x-axis, with the time at which the max value is reached on the y-axis. Using the plotted data, a linear or log fit can be created.

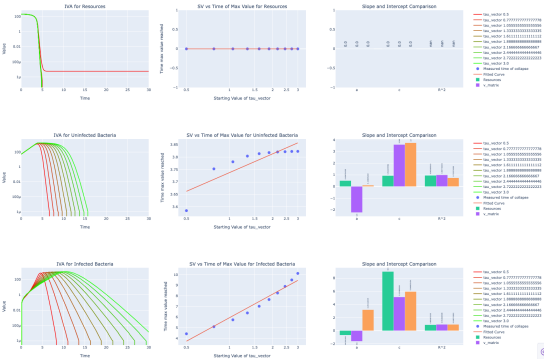
The R^2 value, or coefficient of determination, is calculated as $R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$ where y_i is the observed values, $\hat{y}_i$ is the predicted values, and $\bar{y}$ is the mean of the observed values.

Using this data can be useful for understanding how a change in parameter value affects the time at which the population count reaches a maximum. The slope, intercept and R^2 value is stored and saved in the third plot, a bar chart, with an editable name. For every re-run of the initial value analysis, the slope, intercept and R^2 value is stored in the bar chart, allowing comparison of the slope-intercept data across different parameters.

Phase Portrait The phase portrait plot allows for the user to analyze how an agent population evolves with respect to the other agent population through time. Phase portraits indicate how one population increases while the other decreases, and vice versa. Steady states can be identified and classified as either stable, unstable, or as



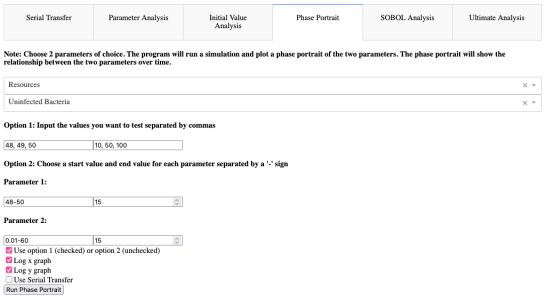
(a) The settings for the initial value analysis tab.



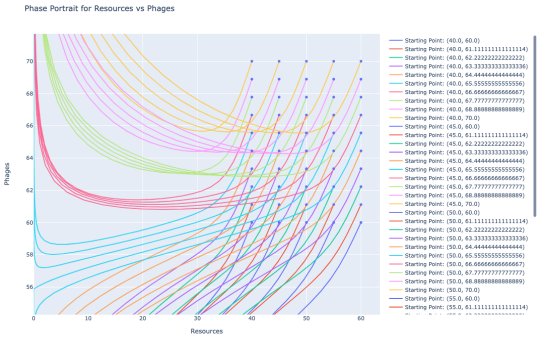
(b) An example initial value analysis output.

Figure 1.5: Initial value analysis

saddle points. By comparing different starting points, it is possible to see if the system is chaotic or not. The setup for the phase portrait can be seen in Figure 1.6a, and a sample output can be seen in Figure 1.6b.



(a) The user can select two starting values for the initial condition, but they can't choose vector, matrix, or environment settings due to the plot showing the development of agent populations against other agent populations. As typical, the user can select their own values or auto-generate values between two values, as well as use a serial transfer option. There is also an option to take the logarithm of the x and/or y-axis.



(b) An example run of a phase portrait.

Figure 1.6: Phase Portrait

SOBOL Sensitivity Analysis SOBOL analysis, a variance-based sensitivity analysis, is a method that allows a user to quantify how important an input parameter has on a measured aspect of the output by changing the parameter values of the model and measuring the change in model output. SOBOL quantifies how much variance in the output can be attributed to a specific parameter and can measure the effect of global/total (ST), first ($S1$), and second order sensitivity ($S2$). Global, also called total sensitivity, is the summation of all higher order interactions. First order Si, or local sensitivity, is the measurement of the effect that parameter i has on the variance of the

output. Second order is the measurement of parameter i interacting with parameter j, nad how the interaction attributes to the output variance. Etc for third order and higher. When $ST_i >> S1_i$ then parameter i depends on interactions with other parameters, while when $ST_i \approx S1_i$, then i doesn't interact much with and depend on other parameters. It should be stated that $ST_i \geq S1_i$.

When a model is viewed as a black-box model, the model can be seen as a function $Y = f(X)$, where X is an input vector of d elements, and Y is a univariate model output. X is assumed to be independently and uniformly distributed within a hypercube $X_i \in [0, 1]$ for $i = 1, \ldots d$. The first order sensitivity measures the output variance of the main affect of parameter X_i. Measuring the effect of varying X_i averaged over other input parameters, and standardized to provide a fractional contribution to the overall output variance. The first order sensitivity is described as

$$S1_i = \frac{V_i}{Var(Y)}$$

where $V_i = Var_{X_i}(\mathbb{E}_{X_{\sim i}}[Y|X_i])$ and where $X_{\sim i}$ represents all the parameters that are not X_i. All parameters are summarized in Table 3.2

The second order index measures the impact of input X_i interacting with X_j. For many inputs, this becomes unwieldy to analyze. The global sensitivity is used to analyze the global sensitivity without evaluating $2^d - 1$ indices, and measures the contribution to the output variance of X_i, including all variance due to X_i's interaction with other variables.

$$S1_i = \frac{\mathbb{E}_{X_{\sim i}}[Var_{X_i}(Y|X_{\sim i}))}{Var(Y)} = 1 - \frac{Var_{X_i}(\mathbb{E}_{X_i}[Y|X_{\sim i}])}{Var(Y)}$$

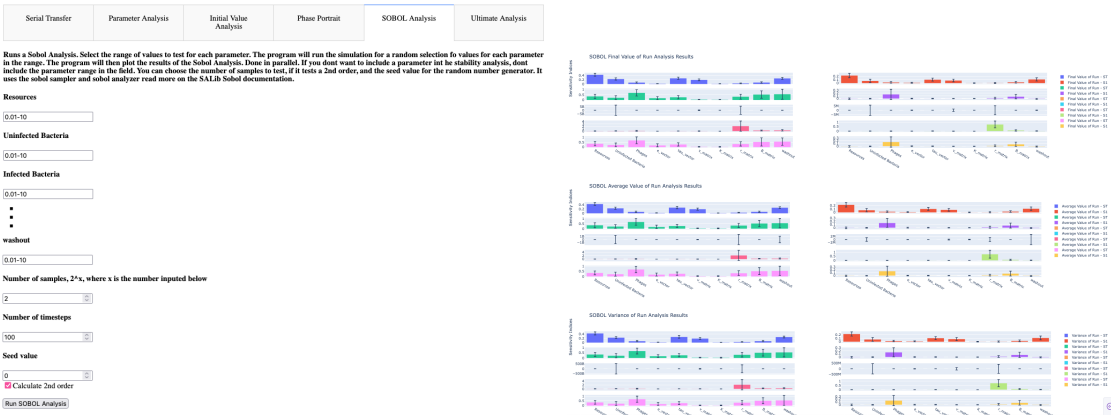
SOBOL can analyze various univariate outputs. This could be either the average value of an agent population, the variance in population count, the time at the peak of an agent count, the final population value, etc.
SOBOL accepts a list of parameter names and a list of range of values to sample from, which the user can input in the SOBOL settings tab, Figure 1.7a. If no values are added, the parameter is not included in the simulation and the default value is instead used. The user then needs to select the number of samples to run, using the formula 2^x, where x is the number they input, and 2^x is the number of samples that SOBOL will create and run. The larger x is, the more accurate the SOBOL analysis results will be, but the more simulations would need to be run.
Otherwise, if 2nd order is not chosen, the model is run $N(D+2)$ times. If the user wants to analyze the second order interactions, then the model will run the system $N(2D+2)$ times with the randomly sampled input values, where N is a multiple of 2, and D is the

number of parameters being tested. Due to the randomness of the sampling method, the user can, but does not need to, submit a seed value.

Three SOBOL analyses are included by default in the dashboard, as shown in Figure 1.7b. An analysis of the final value of the simulation, the average population count, and the variance in population count. The global and first sensitivity are shown next to one another, and each sub-row within a plot represents each agent type. The proportion of the global and local sensitivity can be seen for each agent type and each parameter.

It can be argued that the final, average, and variance value of the run is not a useful statistic to measure and run a SOBOL analysis on. One might give the reasoning that the population value at time t depends on the previous time step $t-1$. Thus the average and variance of the value is not completely random and is semi dependent on the previous value. Another argument is that the simple golden model doesn't exhibit complex behavior unlike the output exhibited in **??**.

Making a dashboard that can be used for different inputs is hard to make. Predicting the type of plots that a user might be interested in, and the type of behavior the user wants to analyze is impossible to predict. Therefore, three simple and easy to understand default SOBOL analysis methods are provided that aims to capture the simple dynamics of the system. Upon completion of a SOBOL analysis, the original simulation data is stored to the disk as a *.pickle* file so that the user can use the data and run their own SOBOL analyses.



(a) The SOBOL settings tab. **(b)** The output that can be expected from SOBOL.

Figure 1.7: SOBOL variance analysis

Ultimate Analysis The Ultimate Analysis section does not produce any visualizations or analysis, but instead allows for the user to define which initial conditions and parameter values they want to run a simulation on. The solver will iterate over every

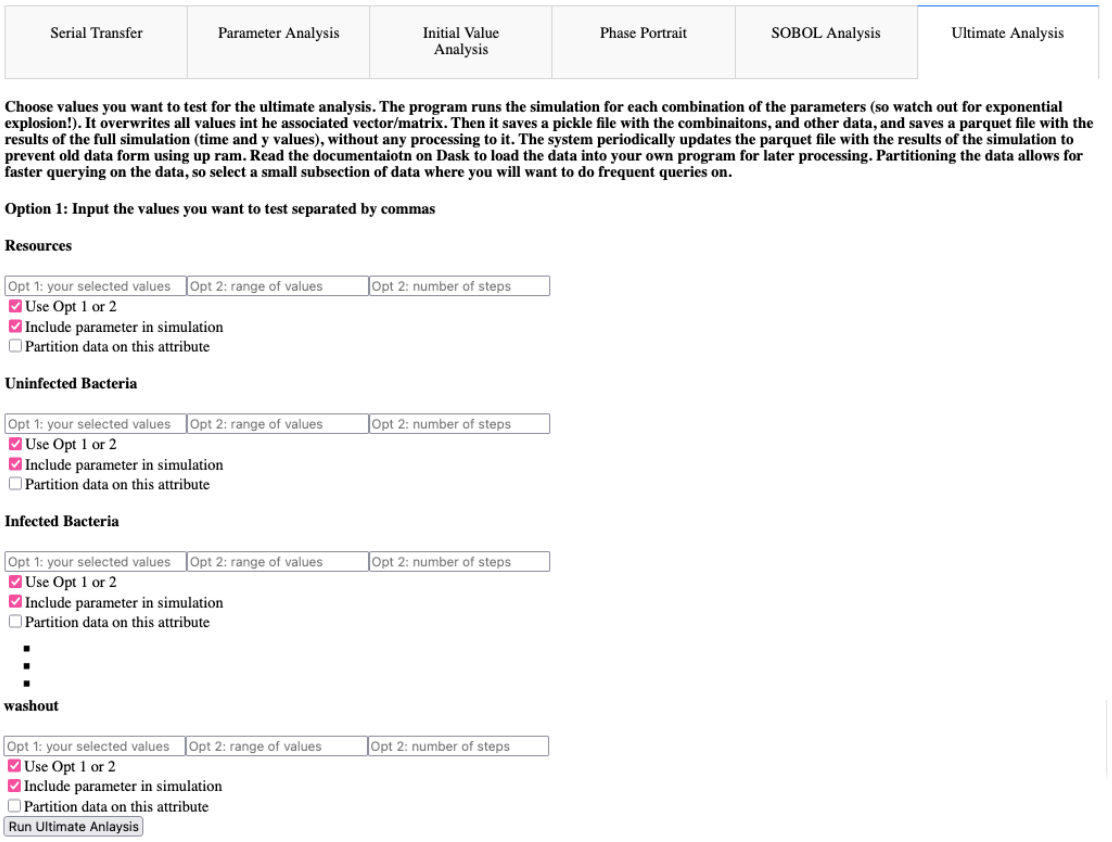


Figure 1.8: The ultimate analysis setup tab.

single parameter input possibility and save the results in a *.parquet* file. Similarly settings in the other sections, the user can specify a start and end value, along with the number of values to generate evenly spaced within that range, including both the start and end values (Figure 1.8).
Using Dask and the saved *.parquet* file, the user can query for specific runs, for example runs where a parameter value was greater than 0.05, and use the simulation data to create their own plots.

1.2.6 Custom Visualizations and Analyses

The final part, an optional step, allows the user to define a number of parameters they want to simulate and download the simulation results. The user can use this data to create their own custom visualizations without having to rerun the simulations, especially if there are many simulations. The data can be further processed and visualized as the user wishes.

Depending on the provided model, different behavior might appear. As the dashboard can not create a graph for every situation, or be easily adapted to analyze every situation, Ultimate Analysis can be used to run and download the simulation data to the disk

to later create your own custom visualizations. For example the agents in a model can exhibit cyclic behavior. A custom visualization that the could be created for this cyclic model would be to perform a Fourier transformation on the curve to obtain the predominant frequencies. A change in parameter values would change the frequencies of the curve, so it would be easy to quantify how a change in parameter value affects the frequency output. If dampening occurs, then a change in amplitude can also be measured, and compared to the change in frequency, allowing the user to identify if the frequency-dampening relationship is correlated or not.

1.3 Software Used and Packages

The program was created exclusively in Python [9], and makes extensive usages of various packages, ranging from the standard scientific packages such as NumPy [10] and SciPy to more niche packages such as pickle and SALib [11, 12].

The graphical tool uses Tkinter acting as the front end, handling the user inputs, while NetworkX [13] stores the graph and contains the attribute data. The GUI tool also uses Matplotlib [14]to create the figure of the graph to display to the user in the GUI tool.

The simulation framework, the backend of the modelling, makes extensive usage of SciPy's *solve_ivp()* to create the ODE data. It also makes light usage of NetworkX to load the graph, as it initially takes a graph as an input, and light usage of NumPy to setup the parameters at startup.

The visualization part makes heavily usage of Dash and Plotly. Dash acts as the server and is used for displaying the HTML aspect of the frontend and dealing with any input and output. Upon choosing parameter values and clicking on "submit", Dash registers the activity and calls the function registered to the button, sending data such as parameter values and options like "log x-axis" form the frontend to the backend server. In the backend, the various inputs are handled, like changing the input string "0.05, 0.1, 0.15, 0.2" into an iterable list [0.05, 0.1, 0.15, 0.2] that the simulation framework can iterate over to vary the parameter value.

If there are many simulations to run through, in the case of Ultimate Analysis, an intermediate call to a parallel computing library Joblib is called, where Joblib parallelizes the for-loop to compute the simulations in parallel.

Ultimate analysis uses Pandas to write the data to a *.parquet* file. Pandas parquet offers efficient data compression, efficient memory usage and when combined with Dask,

efficient querying functionalities in a Dataframe format that many data scientists would be familiar with.

SOBOL uses the SALib library to sample and analyze the parameter input. Both ultimate analysis and SOBOL save a *.pickle* file containing a dictionary with the parameter values tested, setting values, and other important information regarding the simulation.

Initial Value Analysis uses SciPy's *curve_fit()* function to curve fit the points in the middle plot (Figure 1.5b).

Other packages that are used include collections, copy, warnings, itertools, os, datetime, json, gc, and time.

Chapter 2

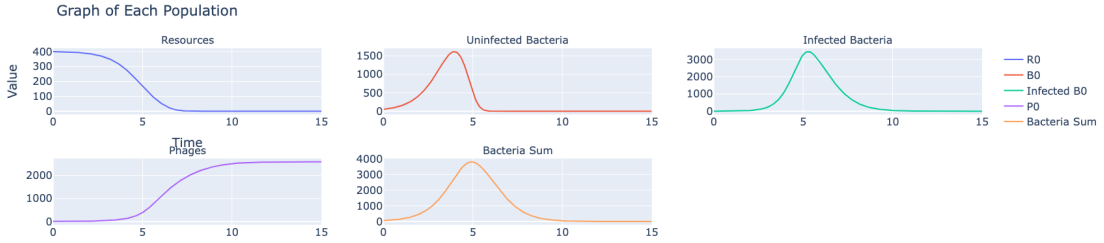
Experiments and results

2.1 A Good Curve

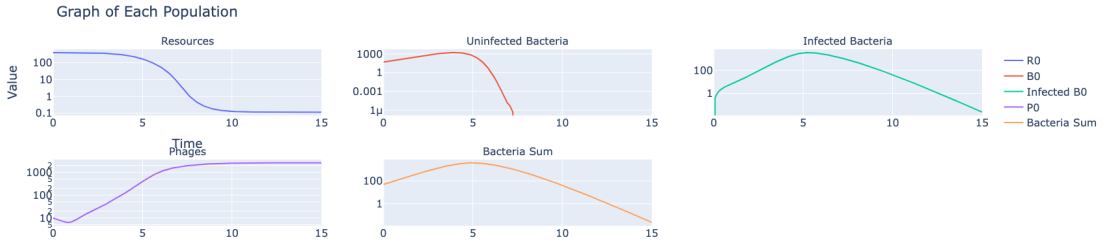
Assuming a very simple model, with no washin or washout rate, a good bacteria growth curve looks like a mountain, with a clear rise, peak, and fall in population levels. For a given initial condition, the bacteria start to consume resources and replicate leading to exponential growth. The phages start to infect the bacteria and eventually the bacteria start to die, releasing new phages. The new phages infect more bacteria, putting pressure on the bacteria growth. Eventually, more bacteria are being infected than being created, causing the decline in bacteria population. Figure 2.1a shows an example of a good curve. Figure 2.1b is the same plot but with a logarithmic y-axis.

As the bacteria population grow, the resource consumption speeds up until there are trace amount of resources left at $t = 8$. The uninfected and infected bacteria exhibit exponential growth, peaking at 1617 at $t = 3.99$ and 3463 at $t = 5.27$ respectively. The delay in the uninfected to infected bacteria's peak is due to the infection stages and latent period of the phage infection. The bacteria sum do not have as stark of a peak in comparison to the uninfected and infected bacteria, due to the graph measuring all bacteria populations, but the peak of 3805 at $t = 4.89$ is still clear. The phages saw a significant increase in population count at around $t = 4$, coinciding with the peak in uninfected bacteria. At this point in time, the infection rate is larger than the bacteria replication rate, so the bacteria are starting to die out even though there are still sufficient resources remaining. At around $t = 4$ is when the the resource consumption rate inflects. The rate at which the resources are being consumed starts to slow down, showing a decreasing sigmoid shape. The total bacteria population reached a peak of 3805 at $t = 4.89$, a 76.1x increase in population count from the initial 50 starting

uninfected bacteria. The phage population reached a peak of 2584 phages at $t = 15$, a 258.4x increase in population count.



(a) Linear y-axis for a "good" plot.



(b) Logarithmic y-axis for a "good" plot.

Figure 2.1: The log plot allows to see behavior happening at values near 0. The parameters used for this plot can be found in Table 7.1.

2.2 SOBOL Sensitivity Analysis Results

It is important to understand how a change in parameter value affects the change in output of a model. Models will have parameters that are more important and have a larger effect on the model output than other parameters.

Figure 8.1a shows the impact that the parameter had on the final value of the population at $t = 15$. THe average value and variance of population value were intentionally left out of the analysis, despite being a part of the dashboard because the SOBOL sensitivity values are almost identical to the final sensitivity values. There were some very minor differences from bar to bar across plots, but the difference was imperceptible. Since the plots all look similar, only an analysis on the final value, Figure 2.2a will be done.

The parameters that were tested include all the parameters listed in the extended golden model, except for Uninfected Bacteria and M. Uninfected Bacteria was left out as it doesn't make sense to already add infected bacteria to the system M, the number of stages that the infection goes through, can not be tested as M hardcodes the number of infection stages that the bacteria has to go through. The hardcoding is done before the simulation framework starts. As such, it is not possible to change M.

2.2.1 Final Value Analysis

2.2.1.1 Resources

The ω^i/washin rate had the largest influence on the final, average and variance value. Without a washin rate, the resources will most likely have been consumed by the time the simulation ended at $t = 15$. The final values for Resources, Uninfected, Infected, and Phages would often be something similar to (0, 0, 0, 10000) at $t = 15$, where all the resources were consumed and the bacteria died out due to the phages, leaving only the phages remaining. The final value of the resources would often be 0, no matter what parameter values were used, with $\omega^i, \omega^o = 0$. With the addition of the washin, new resources were constantly being re-added. Once the bacteria died out, the resources could accumulate, with the accumulation dependent on the rate of the washin rate, hence why the washin rate has such a large impact on the final, average, and variance of population value for the resources. The final value would then be dependent on when the bacteria died out, allowing the resources to accumulate, and the $\omega^i - \omega^0 \cdot R$ rate. Resources were less dependent on higher order interactions, unlike the uninfected, infected, phages, and total bacteria sum.

2.2.1.2 Uninfected

The uninfected bacteria population sensitivities depend on many higher order interactions between the parameters as $ST_i >> S1_i$. The uninfected are highly dependent on β/B_matrix and initial phage population, as the initial phage population will determine how many bacteria become infected, and how quickly the phages can proliferate through the bacteria population. Surprisingly, r/r_matrix did not have as big of an influence on the uninfected as β did, even though the infection rate is dependent on r. The larger or smaller r is, the faster or slower the infection rate is. If r is really small, the infection rate would take forever, potentially allowing the bacteria to keep a stable population. r is equally as important at explaining the final value as τ/tau_vector, washin, e/e_matrix, and washout of sensitivity around 0.25.

2.2.1.3 Infected

Since $ST_i >> S1_i$ for the infected bacteria, where $S1_i \approx 0$ for nearly all of the parameters, the infected bacteria heavily depend on many interactions happening at the same time. This makes intuitive sense after looking at The Golden Model. The infected (and uninfected) bacteria directly interact with R, U, P, v, K, r, τ, and ω^o, (M is not

included as it was left out of the analysis). So due to the high coupling of parameters, the infected (and also the uninfected) have large global sensitivities compared to the local sensitivity.

However SOBOL had some difficulties assigning a good sensitivity score to each parameter for the ST and $S1$ tests as noticed by the slightly larger error bars in the infected than the uninfected or resources. This is most likely due to the infected bacteria going through multiple stages of infection, causing a delay and uncertain behavior in the final value, despite the ODE model being deterministic.

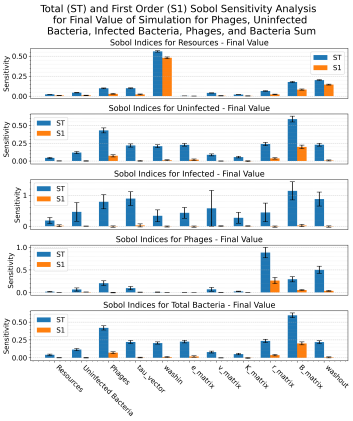
2.2.1.4 Phages

The most important factor for the final phage value is r, followed by β, ω^o and P. The r value allows the phages to infect the uninfected. When r is decreased, the final phage population is counterintuitively higher than when r is larger. The behavior is counterintuitive because one would expect that a higher infection rate would lead to more infections and thus more phages. With a higher r value, more phages are being removed from the phage population and infecting the bacteria. It can be seen as a way of "more phages are needed to infect a bacterium", therefore getting less phages out as a result as more phages are needed to infect a single bacteria.

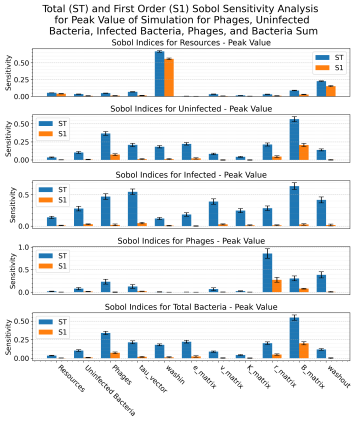
Washout has a noticeable influence on the phage population, as not the phage population is being reduced at a rate proportional to the washout rate. The larger the washout rate, the larger the drawdown of phages. When the infected all die out, the phage population wont grow anymore. Given the phage population at that point in time, the phage removal rate is proportional to the washout rate.

2.2.1.5 Total Bacteria

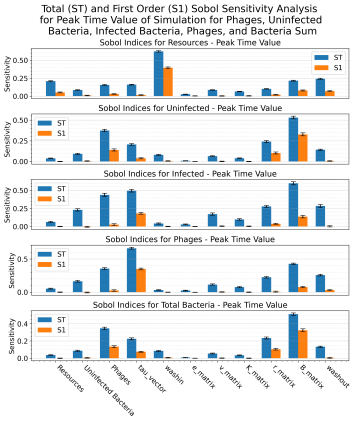
Total bacteria is the sum of both uninfected and infected bacteria, so it makes sense for total bacteria to have similar values to uninfected and infected bacteria. Apparently the uninfected bacteria have a stronger influence on the output variance than the infected bacteria. The total bacteria sensitivities resemble the sensitivities of the uninfected bacteria more than the infected bacteria. It would have been expected for the total bacteria to resemble more of an average between the uninfected and infected.



(a) The ST and $S1$ sensitivity for the final Resource, Uninfected, Infected, Phage, and Total Bacteria population.



(b) The ST and $S1$ sensitivity for the peak value of the Resource, Uninfected, Infected, Phage, and Total Bacteria population.



(c) The ST and $S1$ sensitivity for the peak time value for the Resource, Uninfected, Infected, Phage, and Total Bacteria population.

Figure 2.2: SOBOL analyses for the average, peak, and peak time. The data was saved from the dashboard and plotted using Matplotlib. The average and variance analysis results were left out for nearly identical results to the final value. The values used for this SOBOL test can be found in Table 7.2. The same data used in Figure 2.2b was used for Figure 2.2b and Figure 2.2c.

2.2.2 Custom SOBOL Analysis - Peak Value and Peak Time

Due to the similarity of the final, average, and variance value, a custom SOBOL analysis that isn't included in the dashboard might result in a different SOBOL analysis result. the peak value and the time of the peak of the population is measured and analyzed. The peak is defined as the point where the population reaches 95% of its absolute maximum value. The time of the peak is measured at the point in time that the population reaches 95% of the maximum value. This removes unintended side effects of the simulation. For populations that are only increasing in value, this prevents the measured peak from bunching up at the end of the simulation, skewing the data. As the peak is defined at 95% of the absolute maximum value, populations that have a faster increase on population count at the end will have a time value closer towards the end of the simulation. For populations that reach a plateau, the 95% rule will push the peak time towards the beginning of the simulation, while still "respecting" the absolute final value since $95\% \approx 100\%$. The 95% rule does fail when there is cyclical behavior, as the resource washin can influence the final value of the resources.

Chapter 3

Appendix A: Equation Parameters

Parameters used in the various equations.

3.1 Simple/Advanced Golden Model Parameters

Variable	Name	Description
P/P_p	Phages agent	Phage population for phage p
U/U_b	Uninfected Bacteria agent	Uninfected population for bacteria b
I_i/I_{b_i}	Infected Bacteria agent	Infected population for bacteria b at stage $1, \ldots, i, \ldots, M$
B/B_b	Bacteria agent	Total bacteria population for bacteria b, assuming $B_b = U_b + \sum_{i=1}^{M} I_{b_i}$
R/R_r	Resource agent	Resource r concentration
$e/e_{b,r}$	Consumption rate	Rate at which resources are consumed
$\beta/\beta_{p,b}$	Burst size	Lytic burst size for phage p and bacteria b
$r/r_{p,b}$	Successful phage/cell encounter	Probability of a successful bacteria b infection from phage p
τ/τ_b	Latent period	Time it takes bacteria b to go through one infection stage
$v/v_{b,r}$	Maximal growth rate	Growth rate of bacteria b from resource r
$K/K_{b,r}$	Monod Constant	Monod constant for bacteria b's resource consumption rate dependent on resource r concentration
ω^i/ω^i_r	wash-in rate	Rate of resources r being added
ω^o/ω^o	wash-out rate	Rate of agents being removed, acts on all agents equally
M	Number of infection stages	Number of infection stages that a bacteria goes through, all bacteria agents have the same value for M
t	time	time value

Table 3.1: Golden model parameters with variables, names, and descriptions. Subscripts on parameters indicate relationships; for example, $e_{b,r}$ is nonzero if there is an edge connecting bacteria b to resource r in the network, zero otherwise.

3.2 SOBOL Parameters

Variable	Name	Description
Y	Univariate parameter output	univariate model output, such as mean μ or variance σ
X	Input vector	Vector of size d, input vector to f
i	Parameter input	Parameter i of input
X_i	Parameter value	Value of vector X at position $i = 1, \ldots, d$, the value of parameter i
d	Input size	Size of input vector X
$X_{\sim i}$	Parameter input	All values of X that are not X_i
f	Function f	Arbitrary black-box function describing model
N	Samples	Number of samples, power of 2, 2^x
D	Parameter input size	Number of parameters inputted into SOBOL, $d = \|X\|$
ST_i	Global sensitivity	Contribution of parameter X_i to output variance of Y due to interactions with other variables
$S1_i$	First order sensitivity	Contribution of X_i to output variance of Y

Table 3.2: SOBOL parameter symbols, name, and description.

3.3 Linear Regression Parameters

Variable	Name	Description
a	Slope	Slope of the linear regression line
c	Intercept	y-intercept of linear regression line
R^2	Regression Coefficient	Coefficient of determination of linear regression fit, quality of regression
x_i	Data point	Data point on the x-axis
y_i	Actual Value	Actual value of data for a given x_i
$\hat{y}_i$	Predicted Value	Value predicted of equation for a given x_i
$\bar{y}$	Average Value	Average y value
n	Number of samples	Number of samples being tested

Table 3.3: Variable symbol, name, and description used for the linear regression.

Chapter 4

Appendix B: Industrial and Real Life Applications of Phages

Due to the nature of killing bacteria, there are numerous applications where a researcher or an organization might be interested in controlling bacterial populations.

A Food Safety Specialist might be interested in introducing a solution containing a high concentration of phages during food production to prevent the spread and growth of *Salmonella* or *E. coli* in the pet food. Alternatively, the Food Safety Specialist might want to promote beneficial bacteria like *Streptococcus thermophilus* used in the production of Emmental cheese, which heat would kill when the milk undergoes the pasteurization process.

A doctor might be interested in providing swallowable pills, more commonly known as phage cocktails, to a patient with a bacterial infection. There is evidence that phage-resistant bacteria are more susceptible to antibiotics, so the doctor might prescribe both medicines to effectively deal with the infection.

An Environmental Protection Officer might be interested to see how they can use phages to stop the spread of *Cyanobacteria* blooms in waterways, more commonly known as blue-green algae, a photosynthetic microscopic organism that is technically a type of bacteria. This would keep waterways safe for boating and swimming activity, aquatic life, and water consumption in farms, factories, and homes.

When there are a few known bacterial strains, a targeted concoction of phages can be used to control the bacterial population growth in any setting, either be it food, healthcare, or environmental. Phages offer properties of microbial control that other methods do not, making them an ideal candidate for some applications.

4.1 Controlling Foodborne Bacteria

Foodborne diseases are one of the primary ways for bacteria to spread to humans and animals. Some bacteria use the food as a vector to infect hosts, while some bacteria will deposit toxins on the food that is then ingested. If consumed in large enough quantities, or further produced in the host, the toxins can be fatal to the host.

Methods exist to control bacterial growth, for example by storing food below 5°C or above 60°C. Bacteria need moisture to grow, so starches like rice will have minimal bacterial growth. Bacteria prefer to live in slightly acidic to neutral pH environments, so having an environment that is extremely acidic like vinegar will prevent bacterial growth. The use of chemical antibacterial agents such as bleach is not desirable due to leaving chemicals on the food, which can be fatal if ingested. Physical agents like heat or radiation can kill bacteria, but at the cost of altering the food quality [15].

For example, *Streptococcus thermophilus* is one of three different bacteria strains used to create Emmental cheese. However, Emmental cheese does not use pasteurized milk, increasing the risk of *E. coli*. Emmental cheese producers can add phages that target *E. coli* to the milk during the production stage, while not affecting the bacteria used to produce the cheese.

4.1.1 Current Applications

Phage cocktails like SalmoFresh^{TM} have been proven to safely reduce *Salmonella* contamination in pet food and raw pet food ingredients [1], as well as in romaine lettuce and bean sprouts [2]. Pet food contains meat and vegetables, where vegetables grown in or on the ground are at risk of *Salmonella* due to contact with soil, manure, compost, and other agricultural runoff from neighboring farms [16]. Figure 4.1 and Figure 4.2 show how application of phages have reduced the count of *Salmonella* in ingredients used in pet food as well as romaine lettuce and bean sprouts. In Figure 4.1, each food group noticed at least a 68% reduction in CFU/g compared to the control when the 9×10^6 phage treatment was applied. There was at least an 80% reduction in CFU/g across all food groups when treated with a 9×10^6 or stronger phage solution. In Figure 4.2, the lettuce and bean sprouts noticed a reduction of at least 0.6 log CFU/mL in *Salmonella* count across all temperature ranges. The smallest reduction in bacteria count in lettuce was noticed at 1 hour at 2°C with an absolute reduction in 62.0% between the control and treatment, while the largest reduction in bacteria of 90.0% was found at 72 hours at 2°C. For the bean sprouts, the lowest reduction in phages was found at 1 hour at 2°C with a reduction of 78.1%, and the largest reduction was 90.0% at 25°C after 48

hours. Although these values are still high above food safe, the ability to reduce the *Salmonella* population by at least 62% and up to 90% at different temperatures and incubation periods is impressive and can prolong shelf life, especially for foods that do not have long shelf lives before spoiling due to bacteria. As such, phages can be shown to control the spread of *Salmonella* in food sources and extend the potential shelf life of certain foods.

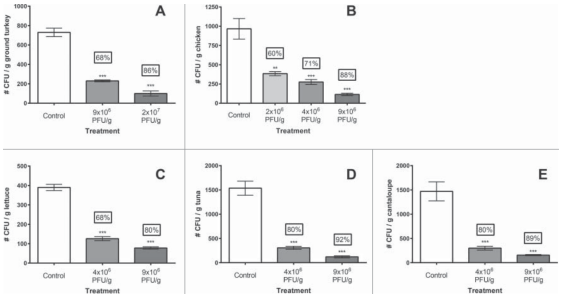


Figure 4.1: SalmoLyse® reduces Salmonella contamination on various food surfaces: Mean and standard error bars shown. Statistical analyses were carried out for each food group independently. Asterisks denote significant reduction from corresponding controls based on one-way ANOVA with Tukey's post-hoc tests for multiple corrections: ** denotes $p < 0.01$, while *** denotes $p < 0.001$ compared to the corresponding controls. There was significant reduction in Salmonella on all food surfaces with the addition of SalmoLyse® compared to the controls; the mean percent reductions from the control are noted in the boxes above treatment bars. CFU/g D colony forming units per gram. Each letter denotes a food group that was tested with SalmoLyse® and compared to a control: A= chicken; B= lettuce; C= tuna; D= cantaloupe; E= ground turkey. Plot sourced from Soffer et al. [1].

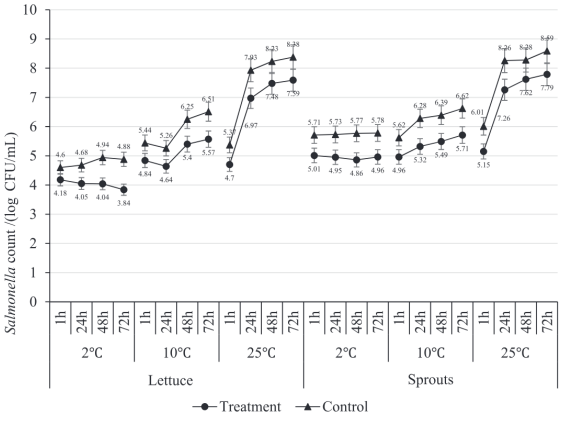


Figure 4.2: *Salmonella* count in a mixture of 5 *Salmonella* strains spot-inoculated (CFU/g) onto a) lettuce and b) sprouts after spraying with a mixture of bacteriophage (SalmoFresh™) relative to positive controls at 2, 10 and 25C and stored for 1, 24, 48 and 72 h. Plot sourced from Zhang et al. [2]

4.2 Phage Therapy and Antibiotics

Antibiotics are a common way to treat bacterial infections. However, antibiotics are not selective in the bacteria they kill, killing both harmful and beneficial bacteria. This can lead to the development of antibiotic-resistant bacteria, which makes it harder to combat that bacteria in the future. It has also been shown that antibiotics have a negative effect on the gut microbiome and brain development in mice. Phages are an alternative to antibiotics, as they are selective in the bacteria they kill and do not interact with cells or other important biological functions. The rise in antibiotic resistant bacteria can be attributed to the overuse and over-prescription of antibiotics and incorrect usage of antibiotics (for example prematurely stopping) [17]. These actions provide an evolutionary pressure on bacteria to mutate and gain resistance to the antibiotics. The phage therapy can contain any number of different phages that can target specific bacterial infections such as *Streptococcus pneumoniae* with minimal risk of side effects.

4.2.1 Current Applications: Bacterial Infection Control

One active area of research is the use of phages to control bacterial infections. Due to the specificity of phages, they can be used to target specific bacteria strains without affecting other beneficial bacteria. When sick with a bacterial infection, patients swallow antibiotic pills to help the body fight the infection. Antibiotics work by either interrupting intercellular processes like the synthesis of RNA [18], by disrupting the structural integrity of the cell wall [19], or by inhibiting protein synthesis [20].

However, antibiotics are not strain specific and indiscriminately kill gut and other bacteria. Common side effects of antibiotics, although usually not serious, include diarrhea, nausea, and headaches. It has also been shown that the effects of early-stage penicillin exposure in mice has found to have a long-lasting effect on the gut microbiome, frontal cortex gene expression, and amygdala gene expression [21]. Penicillin increases cytokine expression (small proteins used in cell signaling) in the frontal cortex of the brain, modifies the blood-brain barrier integrity, and alters behavior. The mice exhibited an increase in aggression and anxiety-like behavior [22]. Phages can be used as an alternative to antibiotics without the side effects and without affecting the gut biome.

With an increase in antibiotic usage, there has been an increase in antibiotic-resistant bacteria. The World Health Organization has stated that antibiotic resistance threatens the modern medicine and the sustainability of an effective, global public health response to the enduring threat from infectious diseases. Common infections, that previously

would have been easy to treat, are harder to treat, and can increase the risk of disease spread, severe illness, and death [23].

One area of research is exploring how bacteria can exchange traits such as phage resistance and antibiotic resistance. Some bacteria are multi-drug resistant, and don't react with the medicine anymore.

Laure and Ahn [24] showed evidence that *Salmonella Typhimurium* is more susceptible to ampicillin in the presence of phages, and phage-resistance can lead to reduced virulence and decreased antibiotic resistance.

Zhao et al. [25] showed that there exists an antagonist coevolution between the bacteria and phages, where the dynamics changed from an arms race dynamic (ARD) to a fluctuating selection dynamics (FSD). Due to phage selection and bacterial competition pressure, when the bacteria gained phage resistance, it lost antibiotic resistance. A genome analysis revealed mutations in the btuB gene of *Salmonella anatum*, with q higher mutation frequency in the ARD stage. A knockout experiment confirmed that the btuB gene is a receptor for the JNwz02 phage and resulted in reduced bacterial competitiveness. Further analysis detected multiple single nucleotide polymorphism (SNP) mutations in the phage-resistant strains. The SNPs potentially affected the membrane components, partially weakening the cell defense against antibiotics. These findings help advance our understanding of phage-host-antibiotics interactions and the impact of adaptations to antibiotic resistance. The research shows how phages can be used to reintroduce antibiotic susceptibility to previous insusceptible bacteria, preventing costly and lengthy research in new antibiotics [25].

Phage research is facing challenges due to bacterial strains evolving resistance to phages. Understanding the interplay between antibiotics and phages is essential for shaping future research [25].

4.3 Environmental Protection

Algae blooms, also called red tides, is the rapid spread of bacterial or algae organisms. Blooms are a growing environmental concern impacting water quality, aquatic ecosystems, and human health. These rapid increases in algae populations, often fueled by excess resources like nitrogen and phosphorus, can occur in freshwater, coastal, and marine environment.

Cyanobacteria blooms have major effects on the aquatic environment as well as human health. Cyanobacteria release nitrogen and phosphorous, which the bacteria use to grow

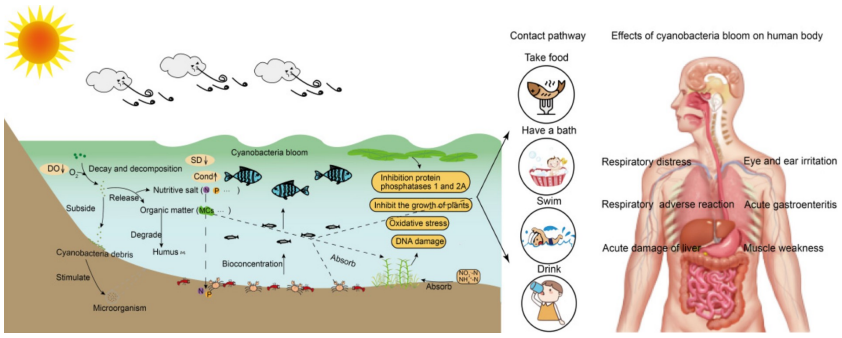


Figure 4.3: Cyanobacteria degradation cycle, main hazards of cyanobacteria bloom to water bodies, aquatic organisms, and the human body. (DO: dissolved oxygen; SD: water transparency; Cond: conductivity; N: nitrogen; P: phosphorus; MCs: microcystins). [3]

with oxygen, outpacing other aquatic growth, and killing aquatic marine life. Bacterial toxins can make their way into the food and water consumed by humans, causing muscle fatigue, respiratory issues, liver damage, and gastrointestinal issues [3]. Figure 4.3 shows the process of how cyanobacteria degrade and are absorbed into the environment, eventually making their way into the human body via various contact points.

4.3.1 Current Applications

There is interest in using phages to control cyanobacteria blooms. Phages can offer better and safer options than chemical options when trying to control bacterial blooms. Chemical options are indiscriminate, killing cyanobacteria, while also killing other beneficial bacteria and aquatic life, and can eventually seep into groundwater. Although not used to control bacteria blooms, some chemicals like PFAS, also called "Forever Chemicals", can last a long time in the environment and don't degrade and keep on negatively affecting the environment. Due to the specificity of phages, only the cyanobacteria will be targeted, and will not affect the surrounding environment.

Tucker and Pollard found that an isolated phage cocktail collected from Lake Baroon in Australia could decrease the abundance of *M. aeruginosa* by 95% within 6 days in a lab setting, before recovering within 3 weeks time [26].
There is evidence that phage-resistant bacteria can influence the population dynamics of other bacteria. It has been shown that the plankton level has been experimentally affected by the frequency of the phage-resistant *Nodularia* marine bacteria. Populations with high phage resistance ($> 50\%$) dominate the plankton communities despite a high phage count and eventually out compete other bacteria due to their slower loss in population count. Contrastingly, populations of bacteria with low phage resistance (between 0% and 5%) were lysed to extinction, releasing resources like nitrogen. This allows for other bacterial strains to absorb the resources and dominate the bacterial

community. Phages and the lysis of bacterial strains can have a dramatic effect on community dynamics and composition of other agents like phages, bacteria, and resources [27]. Phages have the potential to be used as a highly specific strategy for the control of cyanobacterial blooms, with minimal effects to the environment, and offer control of bacterial blooms, with limited impact to the environment. Usage should be relatively safe, novel, efficient, and sensitive.

However, there are issues with using phages to control bacterial blooms. Bacterial blooms can cover vast areas, or be in areas that would be hard to reach like marshlands, applying phages to combat the bloom might be infeasible. If the method of choice was to spray a solution of water containing phages, the solution needs to be shipped to the site and loaded onto special boats to spray the solution into the water, or the trucks need to drive along the shore and spray the solution into the water.

The phage density in the solution will have to be relatively high to quickly combat the bloom. These problems provide major logistical problems with creating the phages in a lab or factory, transporting the phages, and finally the administration of the phages to the waterways. Phages can only diffuse through the water, and can't actively swim, so they are dependent on the rate of diffusion and water currents. This will be difficult in marshlands, where the bacteria can "hide" in the grass and crevices created by aquatic life. If the bloom is in a high current area, like in a river or a bay, the water can wash the phages away.

Scientists have not yet fully understood the phage infection mechanism, and research into the artificial engineering of phages is limited, making it challenging to conduct studies in this area [28, 29].
Algae can produce toxins that threaten wildlife, contaminate drinking water, and disrupt local economies dependent on fishing and tourism. In the state of Florida, between the years 1995 and 2000, the restaurant and hotel industry lost an estimated $6.5 million to algae blooms. This accounts for about 25% of the average total monthly sales revenue in the region from June through October, the months that are most commonly affected by red tide[30]. During a red bloom event, hospital diagnoses in the county of Sarasota for pneumonia, gastrointestinal, and respiratory illness increased by 19%, 40% and 54% respectively [31, 32], with a respiratory illness visit costing between $0.5 and $4 million [33].

Chapter 5

Appendix C: Flowchart of User and System Interactions

Figure 5.1 shows how the user can interact with the system, the input and outputs for subsystems, and the systems working with one another. To read the flow chart, start from the top to the bottom. First the user creates a network using the GUI Network Creation Tool. After the graph is finished, the user provides an implementation of the network as an ODE model, using Python. Once finished, the user provides the network file and ODE model to the ODE solver. The solver uses information from the network file to determine the number of agents to create, parameter details (including names, values, and dimensions), and setting values. Then the user interacts with the Visualization Dashboard Tool, for example by clicking on buttons to run simulations, changing parameter values, (un)selecting checkboxes, and zooming in and out of plots, and hovering over plots to show data. Once a user has selected the parameter values, the parameter values are sent to the solver. The solver calculates the time and population values using the provided graph and ODE model and sends the data back to the Visualization Dashboard Tool, which then outputs the visualizations. If the user has run an ultimate analysis, then the user can query the saved data to make their own custom visualizations.

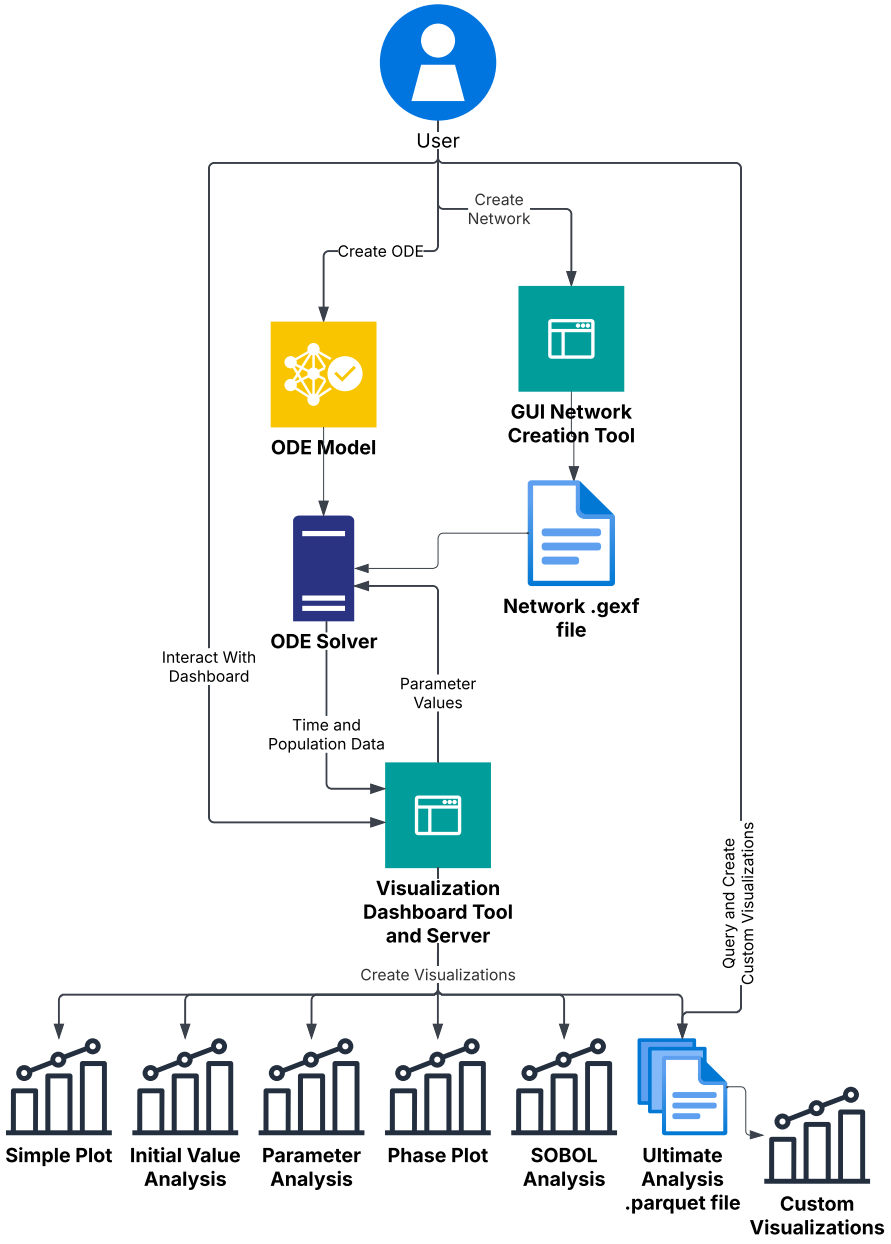


Figure 5.1: The flowchart of user and system interactions. Read from top to bottom.

Chapter 6

Appendix D: ODE Model Implementation

The code listed here is the implementation of Section 1.2.2.

```
import numpy as np
from Classes.Analysis import Analysis
from Classes.GraphMakerGUI import GraphMakerGUI
from Classes.Visualizer import Visualizer

# use base class Analysis to create a new class System
class System(Analysis):
    def __init__(self, graph_location):
        """The ODE representation of the Golden Model from 'Using
    population dynamics to count bacteriophages and their lysogens' from
    Yuncong Geng, Thu Vu Phuc Nguyen, Ehsan Homaee, and Ido Golding.
        Uses Classes.Analysis as a base class.
        Args:
            graph_location (str): location of the graph file
        """
        super().__init__(graph_location)

    def odesystem(self, t, Y, *params):
        """The sytem of ODEs that represent the Golden Model.
        Args:
            t (float): The time value that the solver is currently at.
            Y (np.array): The initial (for t=0)/current (for t>0)
    population values of the system. A 1D array of length equal to the
    number of variables in the system.
        """
        def g(R, v, K):
```

```
23          """Calculate the growth rate of the bacteria. The growth rate
     is a function of the concentration of the resource, the growth rate
    of the bacteria, and the carrying capacity of the bacteria.
24          Args:
25              R (float): Resource concentration
26              v (float): max rate of growth
27              K (float): carrying capacity
28          Returns:
29              float: The growth rate of the bacteria. v*R/(R+K)
30          """
31          return (R * v) / (R + K)
32      # Unpack the parameters. Need to get the graph network, the list
    of phage/bacteria/resource node names, value of M, the vectors (tau
    and washin), and the matrices (e, v, K, r, B), and the environment
    settings
33      graph_object, phage_nodes, bacteria_nodes, resource_nodes, M,
    tau_vector, washin_vector, e_matrix, v_matrix, K_matrix, r_matrix,
    B_matrix, environment = params
34      graph = graph_object.graph
35      Y = self.check_cutoff(Y) # check to see if any values are really
    small. If yes, set to 0
36      R, U, I, P = self.unflatten_initial_matrix(Y, [len(resource_nodes
    ), len(bacteria_nodes), (len(bacteria_nodes), M), len(phage_nodes)]) #
     Turn Y into the shape of the system.
37      new_R = np.zeros_like(R) # create fresh copies of the arrays to
    be updated.
38      new_U = np.zeros_like(U)
39      new_I = np.zeros_like(I)
40      new_P = np.zeros_like(P)
41
42      #update N vector
43      for resource in resource_nodes: # loop over the resource names
44          r_index = resource_nodes.index(resource) # get the index of
    the phage
45          washin = washin_vector[r_index]
46          sum = 0
47          for bacteria in bacteria_nodes: # loop over the bacteria
    names
48              b_index = bacteria_nodes.index(bacteria)
49              if graph.has_edge(bacteria, resource): # important to
    check if the edge between bacteria_b and resource_r exists.
50                  e = e_matrix[b_index, r_index] # get the associated
    values for this interaction
51                  v = v_matrix[b_index, r_index]
52                  K = K_matrix[b_index, r_index]
53                  U_b = U[b_index] # uninfected
54                  I_b = np.sum(I[b_index]) # sum of all infected
    bacteria b agents
```

```
55                  sum += e * g(R[r_index], v, K) * (U_b + I_b)
56          new_R[r_index] = -sum - R[r_index] * environment['washout'] +
     washin # calculate the new value of the resource.
57
58      # update U vector
59      for uninfected_bacteria in bacteria_nodes:
60          u_index = bacteria_nodes.index(uninfected_bacteria)
61          p_sum = 0
62          g_sum = 0
63          for resource in resource_nodes: # loop over the resource
    names
64              r_index = resource_nodes.index(resource) # get index of
    the resource
65              if graph.has_edge(uninfected_bacteria, resource):
66                  g_sum += g(R[r_index], v_matrix[u_index, r_index],
    K_matrix[u_index, r_index])
67          for phage in phage_nodes: # loop over the phage names
68              p_index = phage_nodes.index(phage) # get index of the
    phage
69              if graph.has_edge(phage, uninfected_bacteria): # check if
     the edge between phage_p and bacteria_b exists.
70                  p_sum += r_matrix[p_index, u_index] * P[p_index]
71          # update the uninfected bacteria
72          new_U[u_index] = U[u_index] * g_sum - U[u_index] * p_sum - U[
    u_index] * environment['washout']
73
74      # update I vector
75      for infected_bacteria in bacteria_nodes:
76          i_index = bacteria_nodes.index(infected_bacteria)
77          for k_index in range(0, M):
78              if k_index == 0: # if we are at the first stage of
    infection
79                  p_sum = 0
80                  for phage in phage_nodes: # loop over the phage names
81                      p_index = phage_nodes.index(phage) # get index of
     the phage
82                      if graph.has_edge(phage, infected_bacteria): #
    check if the edge between phage_p and bacteria_b exists.
83                          p_sum += r_matrix[p_index, i_index] * P[
    p_index]
84                  M_tau = 0 if tau_vector[i_index] == 0 else M /
    tau_vector[i_index]
85                  new_I[i_index, 0] = U[i_index] * p_sum - M_tau * I[
    i_index, 0] - environment['washout'] * U[i_index]
86              else: # if we are at the other stages of infection
87                  if(tau_vector[i_index] == 0): # prevent divide by 0
    error
88                      M_tau = 0
```

```
                    else:
                        m_tau = M / tau_vector[i_index] # get the value
    of M_tau
                    right = I[i_index, k_index - 1] - I[i_index, k_index]
                    new_I[i_index, k_index] = m_tau * right - environment
    ['washout'] * new_I[i_index, k_index]

        # update P vector
        for phage in phage_nodes: # loop over the phage names
            p_index = phage_nodes.index(phage) # get index of the phage
            left_sum = 0 # initialize sums
            right_sum = 0
            for infected_bacteria in bacteria_nodes: # loop over the
    bacteria names
                i_index = bacteria_nodes.index(infected_bacteria) # get
    index of the bacteria
                if graph.has_edge(phage, infected_bacteria): # check if
    the edge between phage_p and bacteria_b exists.
                    if (tau_vector[i_index] == 0): # prevent divide by 0
    error
                        M_tau = 0
                    else:
                        M_tau = M / tau_vector[i_index] # get the value
    of M_tau
                    left_sum += B_matrix[p_index, i_index] * M_tau * I[
    i_index, -1]
                    right_sum += r_matrix[p_index, i_index] * (U[i_index]
     + np.sum(I[i_index]))
            # update the phage value
            new_P[p_index] = left_sum - right_sum * P[p_index] -
    environment['washout'] * P[p_index]
        # flatten the new updated initial conditions, undoes the
    flattening done by unflatten_initial_matrix().
        flattened_y1 = self.flatten_lists_and_matrices(new_R, new_U,
    new_I, new_P)
        return flattened_y1

# graph = GraphMakerGUI(seed=0) # create a new object using the GUI tool.
system = System('simple_graph.gexf') # load the graph from the file.

phage_nodes = system.get_nodes_of_type('P') # get the phage nodes
bacteria_nodes = system.get_nodes_of_type('B') # get the bacteria nodes
resource_nodes = system.get_nodes_of_type('R') # get the resource nodes
environment_nodes = system.get_nodes_of_type('E') # get the environment
    nodes

# get the 'Initial_Condition' attribute values from the nodes. Saves as
    vector.
```

```
R0 = system.initialize_new_parameter_from_node("Initial_Concentration",
    resource_nodes)
U0 = system.initialize_new_parameter_from_node("Initial_Population",
    bacteria_nodes)
I0 = system.initialize_new_matrix(len(U0), system.M)
P0 = system.initialize_new_parameter_from_node("Initial_Population",
    phage_nodes)
# get the 'tau' and 'washin' values from the bacteria and resource nodes.
     Saves as vector
tau_vector = system.initialize_new_parameter_from_node('tau',
    bacteria_nodes)
washin = system.initialize_new_parameter_from_node('washin',
    resource_nodes)

# get the 'e', 'v', 'K', 'r', and 'B' values from the edges between the
    listed nodes. Saves as matrix.
e_matrix = system.initialize_new_parameter_from_edges('e', bacteria_nodes
    , resource_nodes)
v_matrix = system.initialize_new_parameter_from_edges('v', bacteria_nodes
    , resource_nodes)
K_matrix = system.initialize_new_parameter_from_edges('K', bacteria_nodes
    , resource_nodes)
r_matrix = system.initialize_new_parameter_from_edges('r', phage_nodes,
    bacteria_nodes)
B_matrix = system.initialize_new_parameter_from_edges('Burst_Size',
    phage_nodes, bacteria_nodes)

visualizer = Visualizer(system) # start the visualizer system.

# add the initial conditions to the visualizer, with a name, the initial
    conditions, and the node names.
visualizer.add_graph_data("Resources", R0, resource_nodes)
# create an uninfected 'hidden' agent
visualizer.add_graph_data("Uninfected Bacteria", U0, bacteria_nodes)
# create infected 'hidden' agent. provide row names, as well as column
    names.
visualizer.add_graph_data("Infected Bacteria", I0, row_names=
    bacteria_nodes, column_names=[f"Infected B{i}" for i in range(int(4))
    ], add_rows=4)
visualizer.add_graph_data("Phages", P0, phage_nodes)

# add the vector parameters to the visualizer, with a name, the parameter
     values, and the node names.
visualizer.add_non_graph_data_vector("tau_vector", tau_vector,
    bacteria_nodes)
visualizer.add_non_graph_data_vector("washin", washin, resource_nodes)

```

```
152 # add matrix parameters to the visualizer, with a name, the parameter
        values, and the node names.
153 visualizer.add_non_graph_data_matrix("e_matrix", e_matrix, bacteria_nodes
        , resource_nodes)
154 visualizer.add_non_graph_data_matrix("v_matrix", v_matrix, bacteria_nodes
        , resource_nodes)
155 visualizer.add_non_graph_data_matrix("K_matrix", K_matrix, bacteria_nodes
        , resource_nodes)
156 visualizer.add_non_graph_data_matrix("r_matrix", r_matrix, phage_nodes,
        bacteria_nodes)
157 visualizer.add_non_graph_data_matrix("B_matrix", B_matrix, phage_nodes,
        bacteria_nodes)
158
159 # optionally add other parameters to the visualizer, that will be passed
        to the ODE system.
160 visualizer.add_other_parameters(phage_nodes, bacteria_nodes,
        resource_nodes, int(system.M))
161
162 visualizer.run() # run the visualizer/dashboard
```

Chapter 7

Appendix E: Parameter Values Used

7.1 A Good Curve

Initial Condition **Resources**		**Uninfected Bacteria**		**Infected Bacteria**	**Phages**
400		50		(0, 0, 0, 0)	10
Vector Data $\boldsymbol{\tau}$	$\boldsymbol{\omega^i}$				
2.14	0				
Matrix Data $\boldsymbol{e}$	$\boldsymbol{v}$	$\boldsymbol{K}$	$\boldsymbol{r}$	$\boldsymbol{\beta}$	
0.03	1.2	10	0.01	20	
Environment Data $\boldsymbol{M}$		$\boldsymbol{\omega^o}$			
4		0			

Table 7.1: The parameter values used for Figure 2.1.

7.2 SOBOL Analysis

Initial Condition **Resources**	**Uninfected Bacteria**	**Phages**
100-500	1-100	1-50

Vector Data $\boldsymbol{\tau}$	$\boldsymbol{\omega^i}$
0.5-3.5	0-100

Matrix Data $\boldsymbol{e}$	$\boldsymbol{v}$	$\boldsymbol{K}$	$\boldsymbol{r}$	$\boldsymbol{\beta}$
0.05-0.25	0.8-1.9	10-250	0.001-0.2	0-100

Environment Data $\boldsymbol{\omega^o}$
0-0.1

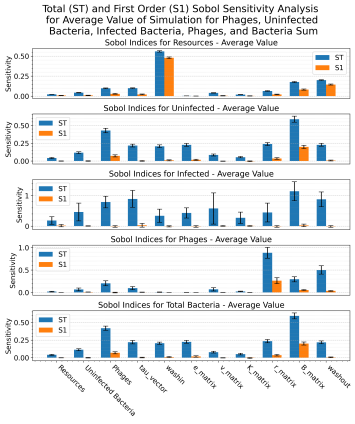
Other Data **Seed Value**	**2nd Order**	**Number Samples**	**Simulations Run**	**Simulation Lenght**
0	False	15	$2^{15}(11+2) = 425984$	15

Table 7.2: The parameter values used for the SOBOL sensitivity analysis in Figure 2.2.

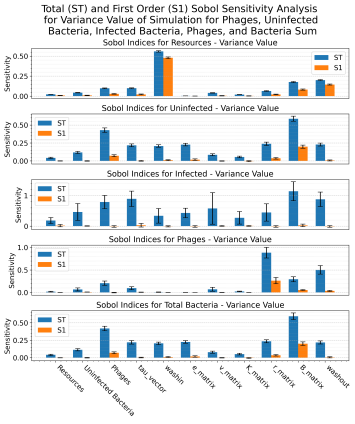
Chapter 8

Appendix F: Extra Plots and Figured

8.1 Extra SOBOL Analyses



(a) The ST and $S1$ sensitivity for the average Resource, Uninfected, Infected, Phage, and Total Bacteria population.



(b) The ST and $S1$ sensitivity for the variance of the Resource, Uninfected, Infected, Phage, and Total Bacteria population.

Figure 8.1: SOBOL analyses for the average, peak, and peak time. The data was saved from the dashboard and plotted using Matplotlib. The values used for this SOBOL test can be found in Table 7.2. The same data used in **??** was used for Figure 8.1a and Figure 8.1b.

Bibliography

[1] Nitzan Soffer, Tamar Abuladze, Joelle Woolston, Manrong Li, Leigh Farris Hanna, Serena Heyse, Duane Charbonneau, and Alexander Sulakvelidze. Bacteriophages safely reduce Salmonella contamination in pet food and raw pet food ingredients. *Bacteriophage*, 6(3):e1220347, July 2016. ISSN null. doi: 10.1080/21597081.2016.1220347. URL `https://doi.org/10.1080/21597081.2016.1220347`.

[2] Xuan Zhang, Yan Dong Niu, Yuchen Nan, Kim Stanford, Rick Holley, Tim McAllister, and Claudia Narváez-Bravo. SalmoFresh™ effectiveness in controlling Salmonella on romaine lettuce, mung bean sprouts and seeds. *International Journal of Food Microbiology*, 305:108250, September 2019. ISSN 0168-1605. doi: 10.1016/j.ijfoodmicro.2019.108250. URL `https://www.sciencedirect.com/science/article/pii/S0168160519301709`.

[3] Weizhen Zhang, Jing Liu, Yunxing Xiao, Yumiao Zhang, Yangjinzhi Yu, Zheng Zheng, Yafeng Liu, and Qi Li. The Impact of Cyanobacteria Blooms on the Aquatic Environment and Human Health. *Toxins*, 14(10):658, September 2022. ISSN 2072-6651. doi: 10.3390/toxins14100658. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9611879/`.

[4] Yuncong Geng, Thu Vu Phuc Nguyen, Ehsan Homaee, and Ido Golding. Using bacterial population dynamics to count phages and their lysogens. *Nature Communications*, 15(1):7814, September 2024. ISSN 2041-1723. doi: 10.1038/s41467-024-51913-6. URL `https://www.nature.com/articles/s41467-024-51913-6`.

[5] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and

Paul van Mulbregt. SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, March 2020. ISSN 1548-7105. doi: 10.1038/s41592-019-0686-2.

[6] J. R. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, March 1980. ISSN 0377-0427. doi: 10.1016/0771-050X(80)90013-3.

[7] Dash Documentation & User Guide | Plotly. https://dash.plotly.com/.

[8] Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL `http://dask.pydata.org`.

[9] Python Software Foundation. Python programming language, version 3.11.6, 2024. URL `https://www.python.org`. Accessed: 2025-05-19.

[10] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

[11] Takuya Iwanaga, William Usher, and Jonathan Herman. Toward SALib 2.0: Advancing the accessibility and interpretability of global sensitivity analyses. *Socio-Environmental Systems Modelling*, 4:18155–18155, May 2022. ISSN 2663-3027. doi: 10.18174/sesmo.18155.

[12] Jon Herman and Will Usher. SALib: An open-source Python library for Sensitivity Analysis. *Journal of Open Source Software*, 2(9):97, January 2017. ISSN 2475-9066. doi: 10.21105/joss.00097.

[13] Aric Hagberg, Pieter J. Swart, and Daniel A. Schult. Exploring network structure, dynamics, and function using NetworkX. Technical Report LA-UR-08-05495; LA-UR-08-5495, Los Alamos National Laboratory (LANL), Los Alamos, NM (United States), January 2008.

[14] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[15] Lars Fieseler and Steven Hagens. Food Safety. In David R. Harper, Stephen T. Abedon, Benjamin H. Burrowes, and Malcolm L. McConville, editors, *Bacteriophages: Biology, Technology, Therapy*, pages 857–890. Springer International Publishing, Cham, 2021. ISBN 978-3-319-41986-2. doi: 10.1007/978-3-319-41986-2_29. URL https://doi.org/10.1007/978-3-319-41986-2_29.

[16] Beata Kowalska. Fresh vegetables and fruit as a source of Salmonella bacteria. *Annals of agricultural and environmental medicine: AAEM*, 30(1):9–14, March 2023. ISSN 1898-2263. doi: 10.26444/aaem/156765.

[17] Stephen Odonkor and Kennedy Addo. Bacteria Resistance to Antibiotics: Recent Trends and Challenges. *International Journal of Biological & Medical Research*, pages 1204–1210, January 2011.

[18] Heinz G. Floss and Tin-Wein Yu. RifamycinMode of Action, Resistance, and Biosynthesis. *Chemical Reviews*, 105(2):621–632, February 2005. ISSN 0009-2665. doi: 10.1021/cr030112j. URL https://doi.org/10.1021/cr030112j.

[19] A. Tomasz. The Mechanism of the Irreversible Antimicrobial Effects of Penicillins: How the Beta-Lactam Antibiotics Kill and Lyse Bacteria. *Annual Review of Microbiology*, 33(Volume 33, 1979):113–137, October 1979. ISSN 0066-4227, 1545-3251. doi: 10.1146/annurev.mi.33.100179.000553. URL https://www.annualreviews.org/content/journals/10.1146/annurev.mi.33.100179.000553.

[20] Sergei B. Vakulenko and Shahriar Mobashery. Versatility of Aminoglycosides and Prospects for Their Future. *Clinical Microbiology Reviews*, 16(3):430–450, July 2003. doi: 10.1128/cmr.16.3.430-450.2003. URL https://journals.asm.org/doi/10.1128/cmr.16.3.430-450.2003.

[21] Angelina Volkova, Kelly Ruggles, Anjelique Schulfer, Zhan Gao, Stephen D. Ginsberg, and Martin J. Blaser. Effects of early-life penicillin exposure on the gut microbiome and frontal cortex and amygdala gene expression. *iScience*, 24(7): 102797, July 2021. ISSN 2589-0042. doi: 10.1016/j.isci.2021.102797. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8324854/.

[22] Sophie Leclercq, Firoz M. Mian, Andrew M. Stanisz, Laure B. Bindels, Emmanuel Cambier, Hila Ben-Amram, Omry Koren, Paul Forsythe, and John Bienenstock. Low-dose penicillin in early life induces long-term changes in murine gut microbiota, brain cytokines and behavior. *Nature Communications*, 8:15062, April 2017. ISSN 2041-1723. doi: 10.1038/ncomms15062. URL https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5382287/.

[23] Global action plan on antimicrobial resistance. URL https://www.who.int/publications/i/item/9789241509763.

[24] Nana Nguefang Laure and Juhee Ahn. Phage resistance-mediated trade-offs with antibiotic resistance in Salmonella Typhimurium. *Microbial Pathogenesis*, 171: 105732, October 2022. ISSN 08824010. doi: 10.1016/j.micpath.2022.105732. URL https://linkinghub.elsevier.com/retrieve/pii/S088240102200345X.

[25] Yuanyang Zhao, Mei Shu, Ling Zhang, Chan Zhong, Ningbo Liao, and Guoping Wu. Phage-driven coevolution reveals trade-off between antibiotic and phage resistance in Salmonella anatum. *ISME Communications*, 4(1):ycae039, January 2024. ISSN 2730-6151. doi: 10.1093/ismeco/ycae039. URL https://doi.org/10.1093/ismeco/ycae039.

[26] Stephen Tucker and Peter Pollard. Identification of Cyanophage Ma-LBP and Infection of the Cyanobacterium Microcystis aeruginosa from an Australian Subtropical Lake by the Virus. *Applied and Environmental Microbiology*, 71(2):629–635, February 2005. doi: 10.1128/AEM.71.2.629-635.2005. URL https://journals.asm.org/doi/10.1128/aem.71.2.629-635.2005.

[27] Sebastián Coloma, Ursula Gaedke, Kaarina Sivonen, and Teppo Hiltunen. Frequency of virus-resistant hosts determines experimental community dynamics. *Ecology*, 100(1):e02554, 2019. ISSN 1939-9170. doi: 10.1002/ecy.2554. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/ecy.2554.

[28] Christopher R. Grasso, Kaytee L. Pokrzywinski, Christopher Waechter, Taylor Rycroft, Yanyan Zhang, Alyssa Aligata, Michael Kramer, and Anisha Lamsal. A Review of Cyanophage–Host Relationships: Highlighting Cyanophages as a Potential Cyanobacteria Control Strategy. *Toxins*, 14(6):385, June 2022. ISSN 2072-6651. doi: 10.3390/toxins14060385. URL https://www.mdpi.com/2072-6651/14/6/385.

[29] Katelyn M. McKindles, Makayla A. Manes, Jonathan R. DeMarco, Andrew McClure, R. Michael McKay, Timothy W. Davis, and George S. Bullerjahn. Dissolved Microcystin Release Coincident with Lysis of a Bloom Dominated by Microcystis spp. in Western Lake Erie Attributed to a Novel Cyanophage. *Applied and Environmental Microbiology*, 86(22):e01397–20, October 2020. doi: 10.1128/AEM.01397-20.

[30] (PDF) Economic Impacts of Red Tide Events on Restaurant Sales. URL https://www.researchgate.net/publication/23515658_Economic_Impacts_of_Red_Tide_Events_on_Restaurant_Sales.

[31] Yung Sung Cheng, Yue Zhou, Clinton M. Irvin, Richard H. Pierce, Jerome Naar, Lorraine C. Backer, Lora E. Fleming, Barbara Kirkpatrick, and Dan G. Baden. Characterization of Marine Aerosol for Assessment of Human Exposure to Brevetoxins. *Environmental Health Perspectives*, 113(5):638–643, May 2005. ISSN 0091-6765. doi: 10.1289/ehp.7496. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1257561/`.

[32] Barbara Kirkpatrick, Judy A Bean, Lora E Fleming, Gary Kirkpatrick, Lynne Grief, Kate Nierenberg, Andrew Reich, Sharon Watkins, and Jerome Naar. Gastrointestinal Emergency Room Admissions and Florida Red Tide Blooms. *Harmful algae*, 9(1):82–86, January 2010. ISSN 1568-9883. doi: 10.1016/j.hal.2009.08.005. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2786186/`.

[33] Porter Hoagland, Di Jin, Lara Y. Polansky, Barbara Kirkpatrick, Gary Kirkpatrick, Lora E. Fleming, Andrew Reich, Sharon M. Watkins, Steven G. Ullmann, and Lorraine C. Backer. The costs of respiratory illnesses arising from Florida gulf coast Karenia brevis blooms. *Environmental Health Perspectives*, 117(8):1239–1243, August 2009. ISSN 1552-9924. doi: 10.1289/ehp.0900645.