Bacteria Growth Model

Minerva University

CS166 - Modeling and Analysis of Complex Systems

March 2, 2025

1 Model Description

This paper focuses on the simulation and analysis of the bacteria growth model in closed 2 dimensional environment. In the environment, we track the dynamics between the amount of food and bacteria, which are influenced by factors including bacterial reproduction rate, and food growth rate, bacterial and food diffusion rate, bacterial consumption rate, food reseeding probability, and food reseed rate. We will run simulations on the model using cellular automata (CA), analyze the empirical results, and come up with mathematical relationship between variables.

1.1 Initializing the bacteria growth model

The simulation is run on a 100 x 100 grid, each grid consists of a tuple, recording the food count and the bacteria count. And the CA employs a Von Neumann neighborhood.

The initialization consists of three parameters — bacteria probability (bacteria_prob), food probability (food_prob), and the amount of bacteria for initialization (bacteria_amount). The bacteria population is initiated by choosing which grids of the population is to be populated using the bacteria probability. Afterwards, we put a fixed bacteria amount (bacteria_amount), which is set to 20, in those grids.

The food distribution is initiated by using food probability (food_prob) to decide the grids that contains food. Then, we randomly generate the amount of food between 0 and the maximum food amount (100 units) in them. There is no caps on the bacteria population. It is nonetheless constraint by the maximum food amount.

There are two reasons to make this arrangement. First, I wanted to observe the bacteria's growth, so I let them start from a small number. Second, I wanted to create different growth dynamics in the beginning of the simulation, thus I let food amount vary.

Name	Variable	Description
Food probability	food_prob	The probability that a grid contains certain amount of food when initialized.
Bacteria probability	bacteria_prob	The probability that a grid contains certain bacteria when initialized.
Bacteria amount	bacteria_amount	The fixed amount of bacteria in a grid if present.
Grid size	grid_size	The side length of the square grid.

Table 1: Parameter descriptions for initializing the bacteria growth CA simulation.

The simulation code for the bacterial growth model initialization shows the methods used to initialize the CA. The parameters are all predetermined.

```python
def initialize_ca(self, food_prob=0.3, bacteria_prob=0.1, bacteria_amount=15):
    """
    Initialize the grid with random food and bacteria.
    """
    # Initialize food randomly by generating a temporary grid for deciding where to put food
    # Initiate with randomly chosen food amount between 0 and 100
    food_mask = np.random.random((self.grid_size, self.grid_size)) < food_prob
    self.food = np.where(food_mask, (self.food_max * np.random.random((self.grid_size,
    self.grid_size))), 0)

    # Initialize bacteria randomly by generating a temporary grid for deciding where to
    put bacteria
```

```python
    # Initiate with 15 bacteria for the grids with bacteria
    bacteria_mask = np.random.random((self.grid_size, self.grid_size)) < bacteria_prob
    self.bacteria = np.where(bacteria_mask, bacteria_amount, 0)

    # Record initial state
    self.food_history = [np.mean(self.food)]
    self.bacteria_history = [np.mean(self.bacteria)]

    if self.testing_mode == True:
      np.set_printoptions(threshold=20) # Ensures all elements are printed
      print("Initialized CA")
      print("------- Food distribution --------")
      print(self.food)
      print("------- Bacteria population --------")
      print(self.bacteria)
```

1.2 Updating the bacteria growth model

In every step of updates, the bacteria will first consume the amount of food available in its grid based on its consumption rate (bacteria_consumption_rate). Then, the bacteria reproduces based on its current amount by its reproduction rate (bacteria_growth_rate). The remaining food then grow by its growth rate (food_growth_rate) with a probability to reseed (reseed_prob). The food growth follows logistic growth formula, meaning that for the amount of food in a grid a time t (f_t). The food growth follows the formula:

$$f_{t+1} = f_t(1 + g_f(1 + \frac{f_t}{k_f})),$$

where g_f is the food growth rate and k_f is the maximum food amount (100 units).
Then, the food reseeds with a predetermined probability of 0.01. Finally, the food and bacteria in a grid both diffuse to their's Von-Neumann neighbors by their respective diffusion rates (food_diffusion_rate, bacteria_diffusion_rate), and each of the neighbors gets $\frac{1}{4}$ of the amount that diffused.

Name	Variable	Description
Bacteria reproduction rate	bacteria_growth_rate	A fixed reproduction rate of bacteria population in each step.
Food growth rate	food_growth_rate	A logistic growth rate of food in each step.
Bacteria consumption rate	bacteria_consumption_rate	The amount of food the bacteria consume in proportion to the bacterial population each step.
Food diffusion rate	food_diffusion_rate	The proportion of food in a grid that equally diffuses to it's Von-Neumann neighborhood neighbors each step.
Bacteria diffusion rate	bacteria_diffusion_rate	The proportion of bacteria in a grid that equally diffuses to it's Von-Neumann neighborhood neighbors each step.
Food reseeding probabiltiy	reseed_prob	A predetermined probability that the food amount in a grid + 1

Table 2: Parameter descriptions for updating the bacteria growth CA.

The code for simulation also clearly demonstrated the sequence of interaction between food and bacteria, and their own population dynamics.

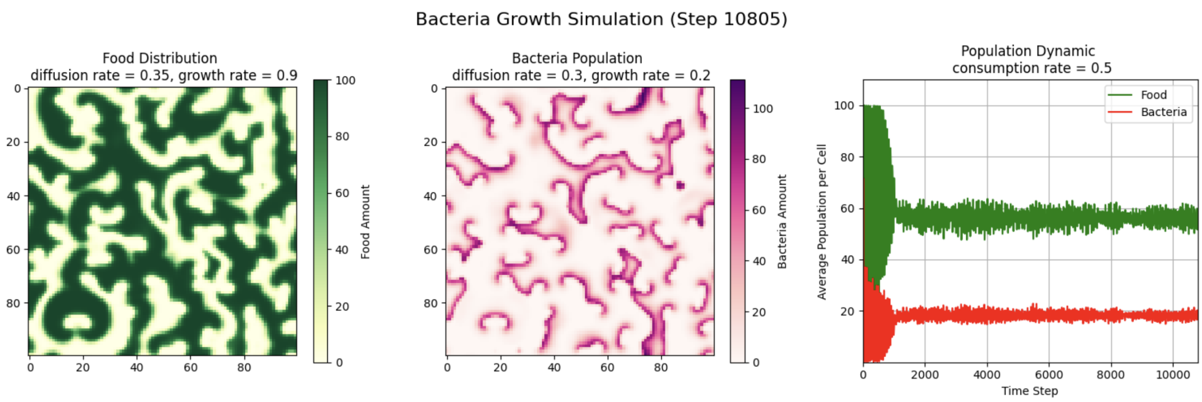
```python
def update(self):
  """
  Perform one step of the simulation.
  """

  # Bacteria consume food, and die out if they don't have enough food
  self.consumption_and_startvation()
```

```python

    # The remaining bacteria reproduce
    self.bacteria_reproduction()

    # The remaining food grows
    self.food_growth()

    # Food reseeds with a probability
    self.food_reseeding()

    # Food diffuses
    self.food_diffusion()

    # Bacteria diffuses
    self.bacteria_diffusion()

    # Record history
    self.food_history.append(np.mean(self.food))
    self.bacteria_history.append(np.mean(self.bacteria))

    if self.testing_mode == True:
      print("One step done!")
```

2 Empirical analysis

2.1 Simulation

The CA simulation of the bacteria growth model were run for 12,000 steps each. There are generally two types of result — the ones that converges, and the ones that do not.



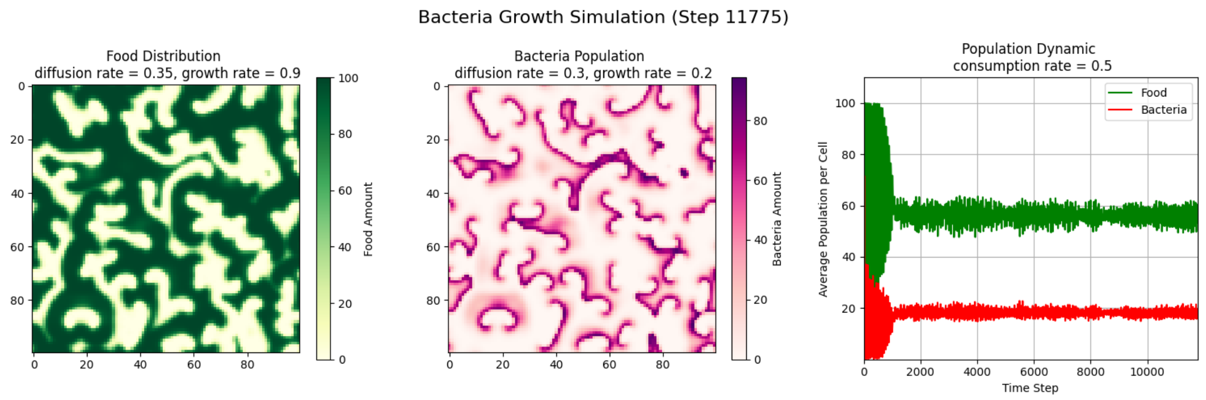


Figure 1: A simulation result that converges. Captures of step 10805 and 11775

When the population of bacteria and food converges, the animation of the grid dynamics shows that the density of bacteria and food both stays relatively constant (See figure 1). The oscillation remains relatively small and the pattern of the CA looks more scattered.

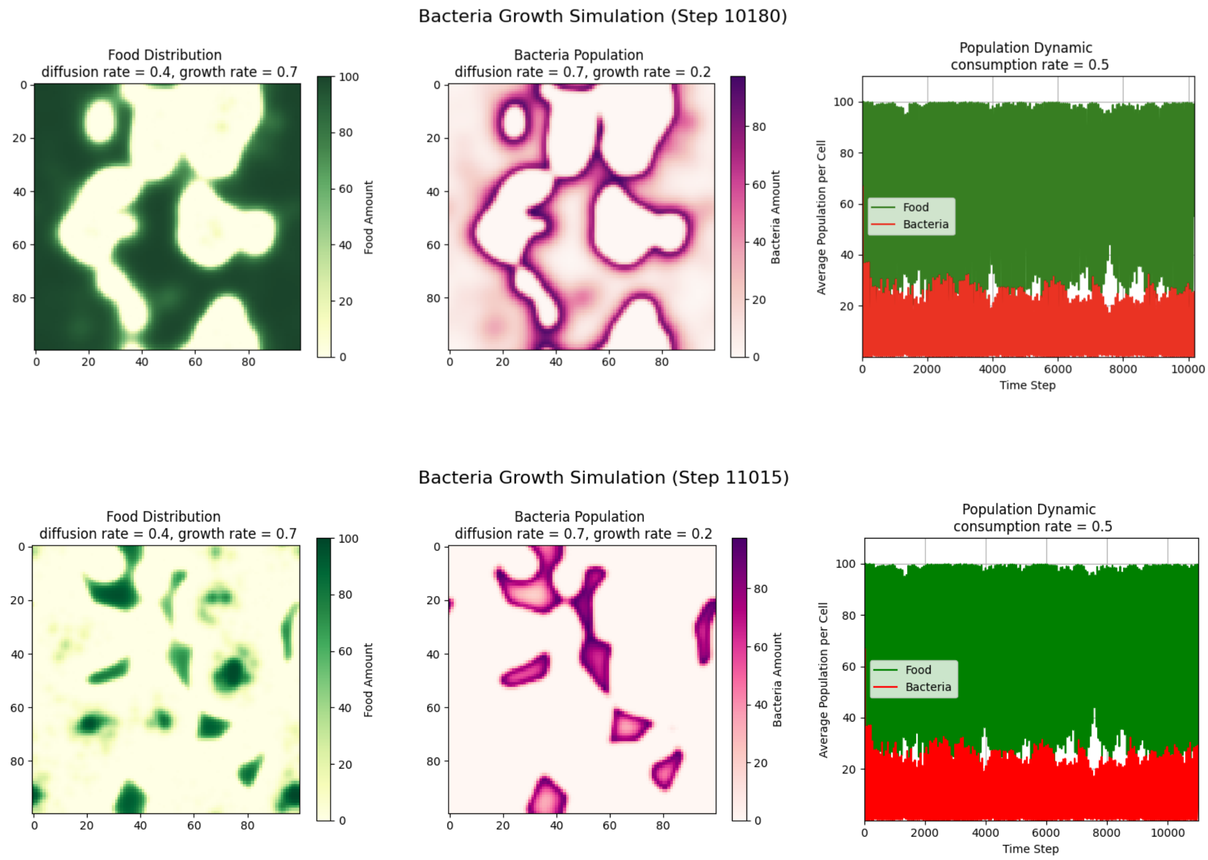


Figure 2: A simulation result does not converge. Captures of step 10180 and 11015

When the population of bacteria and food does not converge, the animation of the grid dynamics shows that the density of bacteria and food oscillates largely (See figure 2). From the animation we can observe that the food keeps blooming and dying out.
In both cases, we can see that bacteria are usually the most dense at the borders of food clusters. Then, the food regrows from the areas that are empty (no bacteria or food), then bacteria returns.

2.2 Relationship between variables

After simulation, I plotted the relationship between the five variables with the outputs — the food and bacteria population. All of the 5 parameters are set to 0.5 in the default mode.

Name	Variable	Value
Bacteria reproduction rate	bacteria_growth_rate	0.5
Food growth rate	food_growth_rate	0.5
Bacteria consumption rate	bacteria_consumption_rate	0.5
Food diffusion rate	food_diffusion_rate	0.5
Bacteria diffusion rate	bacteria_diffusion_rate	0.5

Table 3: Default parameters used for simulation.

I ran the simulation 100 times and each simulation went for 5000 steps (I wanted to run more steps but it ended up taking hours to run). The output of the last 500 steps were recorded and averaged to show the data after the system converges. The results are shown below.

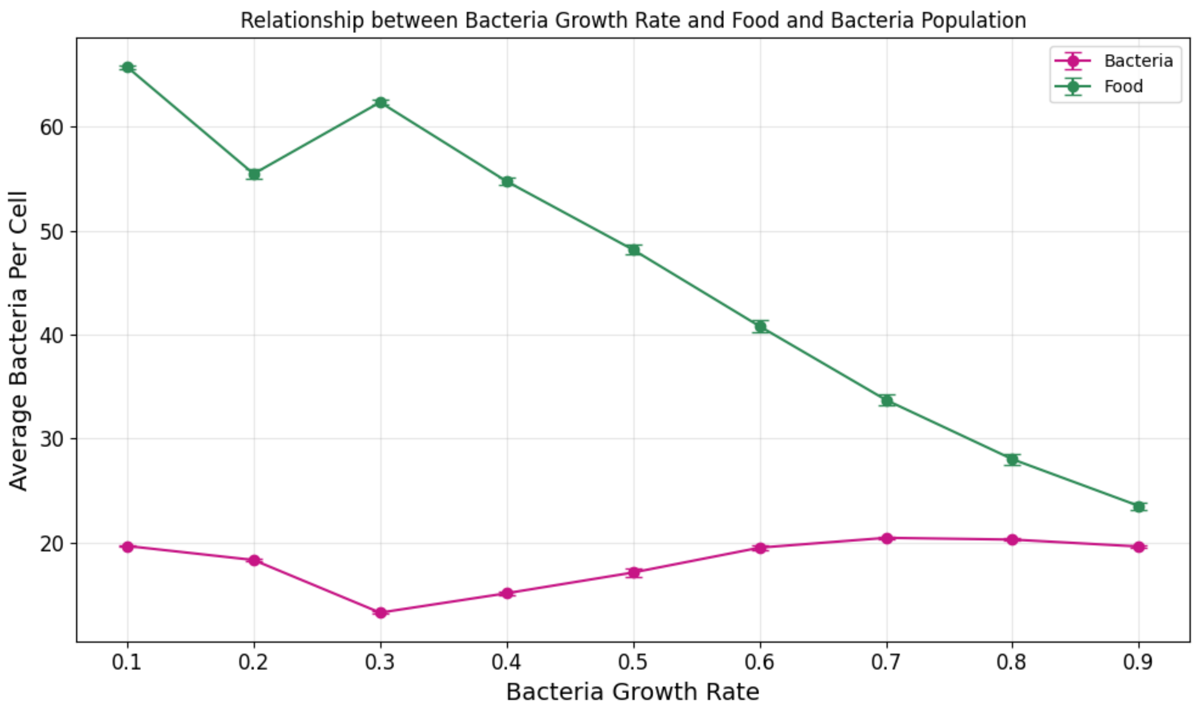


Figure 3: Relationship between bacteria growth rate and food and bacteria population

Figure 3 shows the bacteria growth rate does not actually influence the bacteria population that much — when bacteria growth rate is 0.1 the average bacteria per cell is around 20, which is the same as when it is 0.9. However, it influenced the average food amount per cell. I assume that the bacteria population is still constraint by settings like its diffusion rate and maximum food amount. However, when they have a higher growth rate, the bacteria population can grow too much and each out most of the food rapidly.

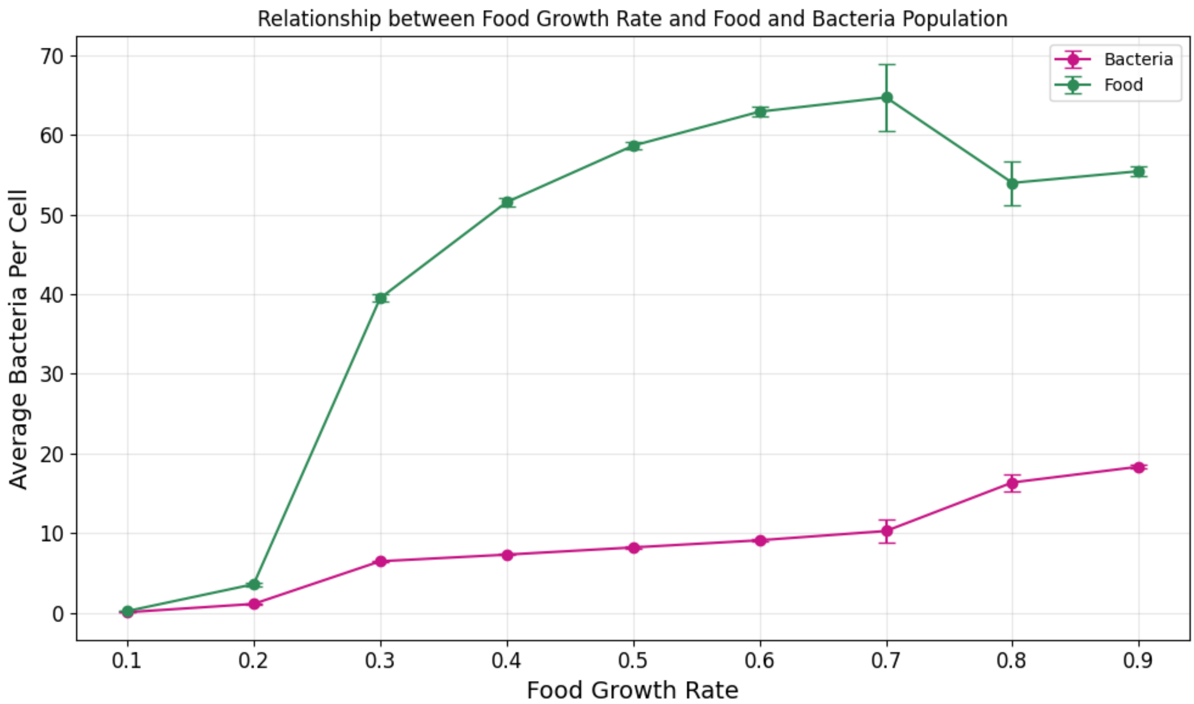


Figure 4: Relationship between food growth rate and food and bacteria population

Figure 4 shows the food growth rate influences the food population largely. The higher the food growth rate is, the higher the food population until it reaches around 0.7. More abundant food also highers the bacteria population, and perhaps this causes higher consumption rate that eventually lowers the food population after food growth rate reaches 0.7.

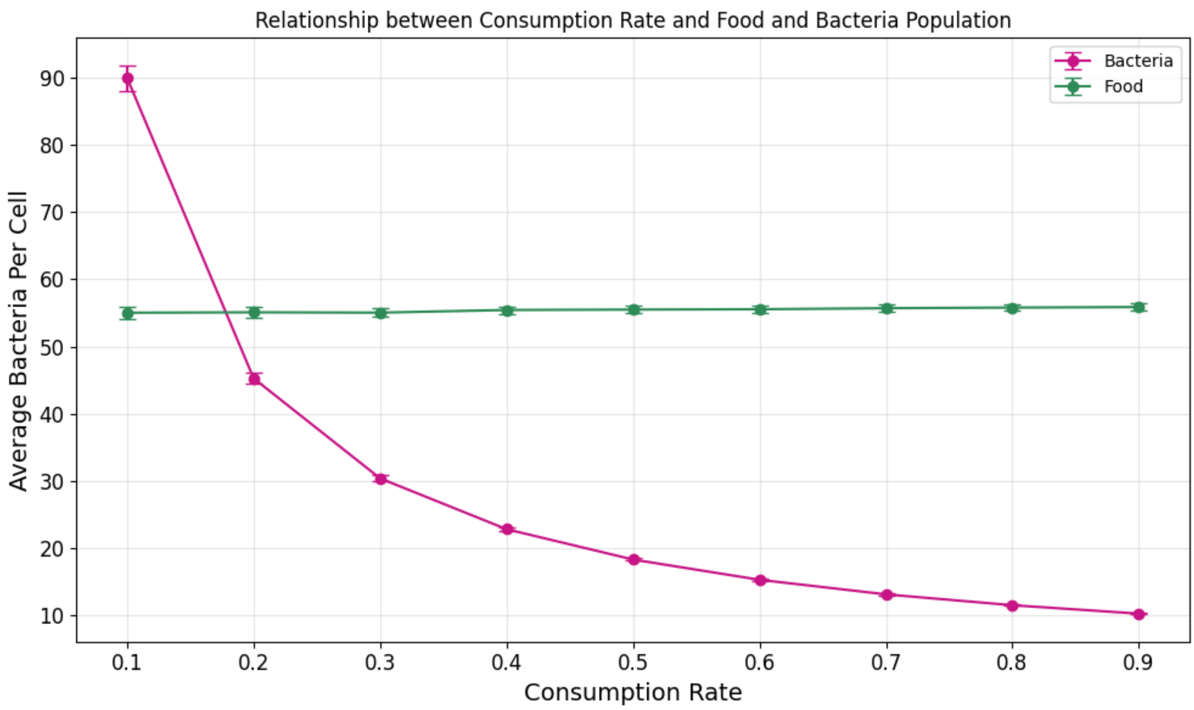


Figure 5: Relationship between consumption rate and food and bacteria population

Figure 5 shows that the bacteria growth rate almost does not influence the food population at all. However, given roughly the same amount of food available all time, the bacteria population decreases exponentially (judge by the shape) as the consumption rate grows.

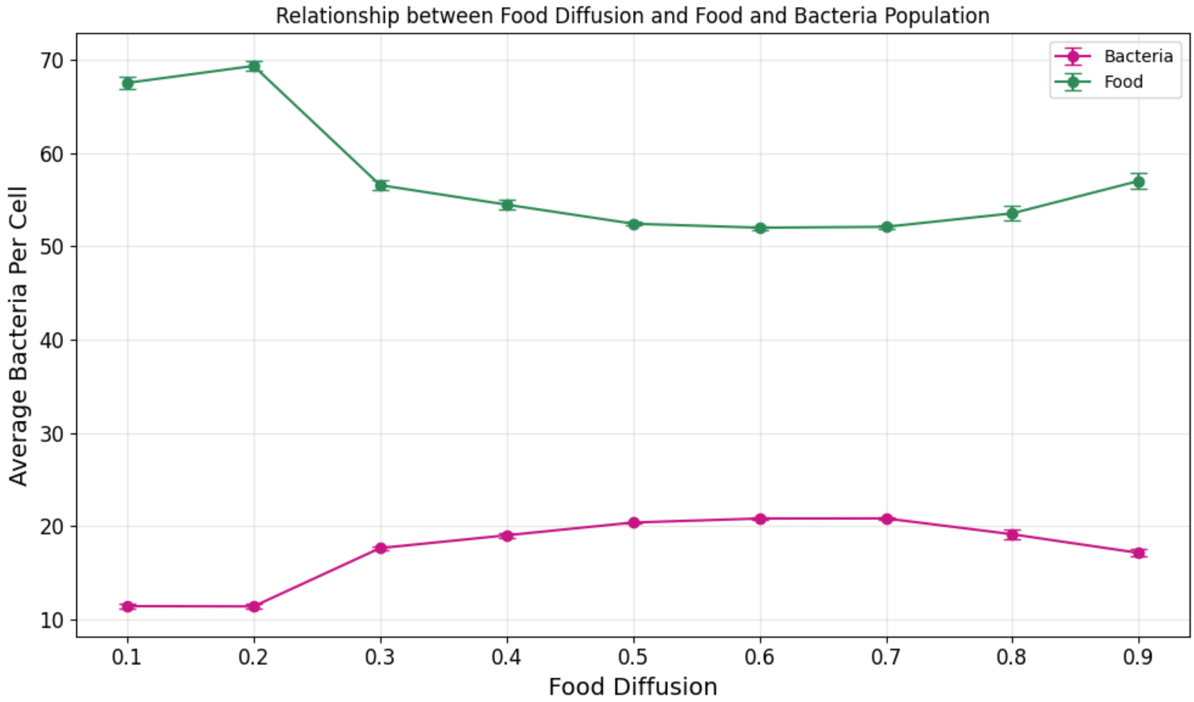


Figure 6: Relationship between food diffusion rate and food and bacteria population

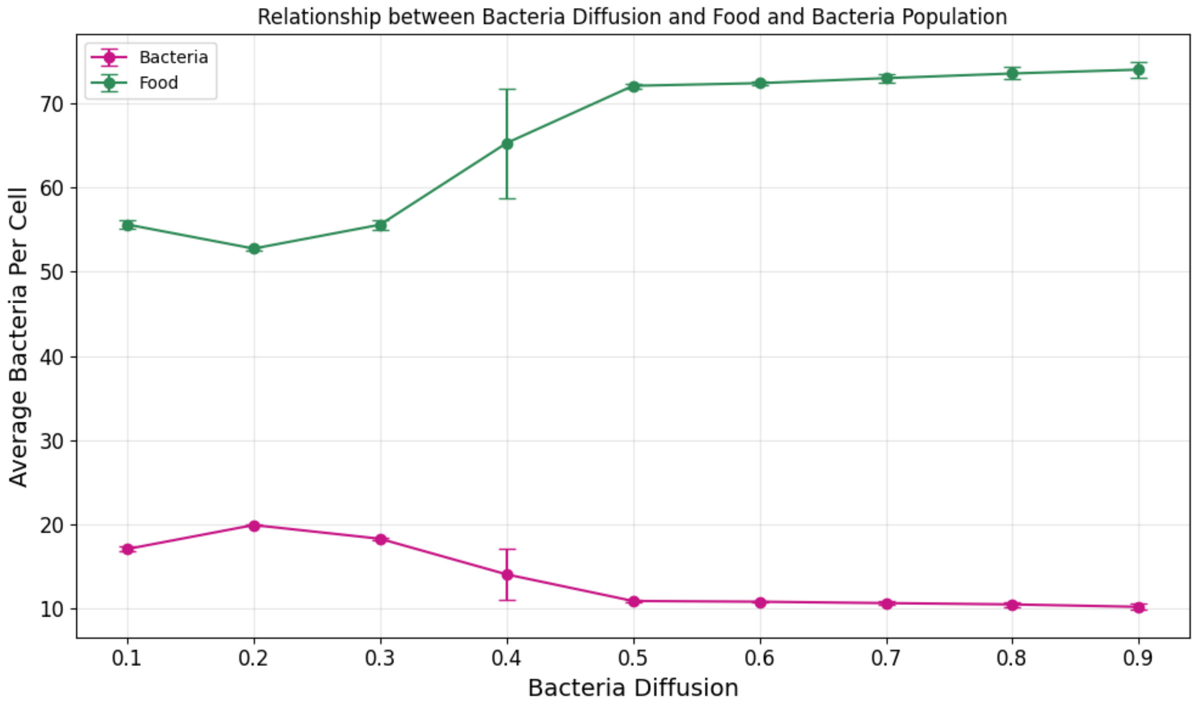


Figure 7: Relationship between bacteria diffusion rate and food and bacteria population

Figure 6 and 7 show that the diffusion rates might influence the two output populations roughly equally, as the bacteria and food population lines on both graphs seem symmetrical to each other.

3 Theoretical analysis

After exploring the relationship between input parameters and output populations, I first fit linear regression between every pair of them. The result is shown below:

	bacteria_correlation	bacteria_slope	bacteria_intercept	bacteria_r_squared	food_correlation	food_slope	food_intercept	food_r_squared
bacteria_growth_rate	0.445085	4.080122	16.108139	0.198100	-0.971953	-53.765603	72.728525	0.944693
food_growth_rate	0.966541	21.384267	-2.118268	0.934202	0.803780	72.345791	7.227771	0.646062
consumption_rate	-0.824998	-76.945612	67.043009	0.680622	0.971766	1.155175	54.876798	0.944329
food_diffusion	0.667625	9.023062	13.023957	0.445724	-0.694638	-16.801923	65.610579	0.482522
bacteria_diffusion	-0.878447	-12.356714	19.811900	0.771669	0.907580	29.598801	51.184762	0.823701

Figure 8: Linear regression between every pair of input and output

The R^2 are generally not very high, as we can see from figures above that the average population's reaction to different input parameters are very dynamic. However, the food growth rate to bacteria population has an R^2 score of 0.93 and a 96% correlation. The bacteria growth rate to food population has an R^2 of 0.94 and a -97% correlation. We can write the relationship mathematically as:

$$AverageBacteriaPopulation = FoodGrowthRate \times 21.38 - 2.11$$

This shows a positively correlated relationship with a slope of 21.38.

$$AverageFoodPopulation = BacteriaGrowthRate \times -53.76 + 72.72$$

This shows a negatively correlation relationship with a slope of -53.76, the output values are still all positive as $FoodGrowthRate$ will not exceed 1.
Lastly, given the shape of the line in Figure 5, I fit an exponential model to the relationship between bacteria consumption rate and the average food amount per cell (see Figure 8).

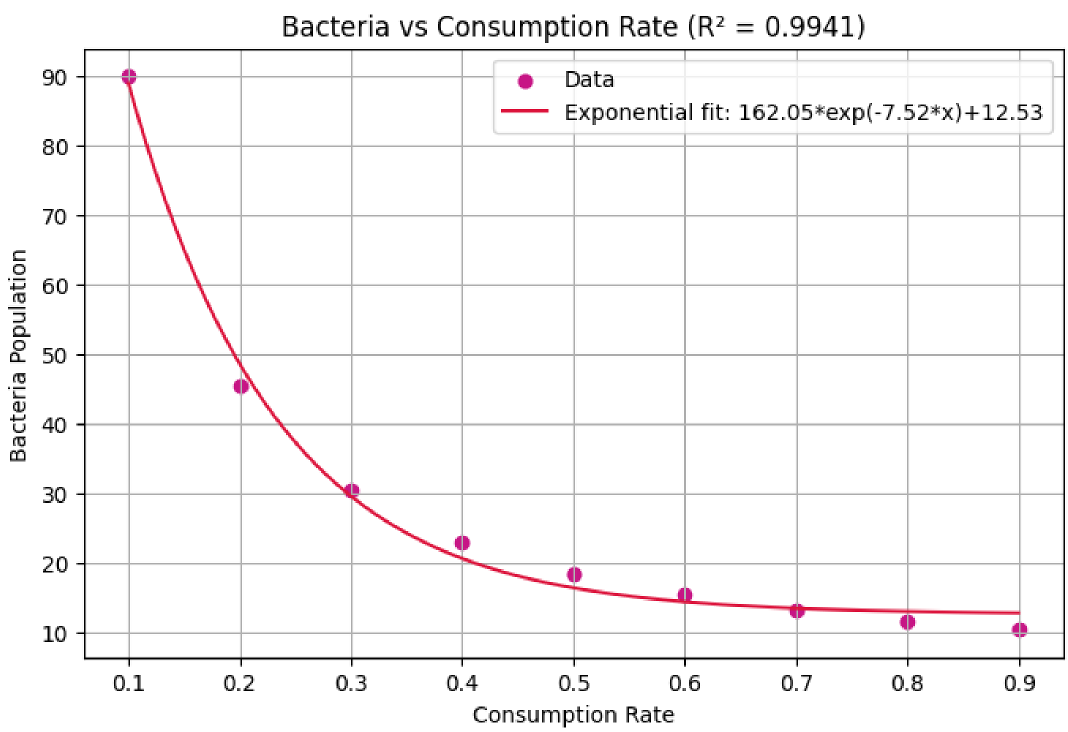


Figure 9: Exponential model between the average bacteria population and the bacteria

The R^2 score is 0.99, meaning that this exponential model can account for 99% of the change in output given the input. The mathematical relation is shown below:

$$AverageBacteriaPopulation = 162.05e^{\text{-7.52*BacteriaConsumptionRate}} + 12.53$$

4 Reference

- CS166 Session 6 - [3.2] Simulating cellular automata -for code that implements and animates CA

AI Statement

I used Grammarly to help correct my grammar mistakes. I used Claude to help debug and correct my code in Python. No other AI tools were used.

Collaboration Statement

I did not discuss, cross-check, or collaborate with anyone on this assignment.

5 Appendix

5.1 Appendix A - Main code for the bacteria growth model

```
class BacteriaGrowthSimulator:
    """
    A cellular automaton simulation of bacteria growth and food dynamics.
    """

    def __init__(self, food_growth_rate, food_diffusion_rate, bacteria_consumption_rate, bacteria_growth_rate, bacteria_diffusion_rate,
                 reseed_prob=0.01, grid_size=100, food_max = 100, testing_mode = False):
        """
        Initialize the simulation with given parameters.

        Parameters:
            grid_size (int): Size of the square grid
            food_growth_rate (float): Growth rate of food
            food_max (float): Food capacity (maximum food per cell)
            food_diffusion_rate (float): Food diffusion rate
            bacteria_consumption (float): Bacteria consumption rate
            bacteria_diffusion_rate (float): Bacteria diffusion rate
            reseed_prob (float): Probability of food reseeding
            food_prob (float): Probability of food in each cell
            bacteria_prob (float): Probability of bacteria in each cell
            bacteria_amount (float): Initial amount of bacteria if present
        """
        self.grid_size = grid_size
        self.food = np.zeros((grid_size, grid_size))
        self.bacteria = np.zeros((grid_size, grid_size))

        # Parameters
```

```python
        self.food_growth_rate = food_growth_rate
        self.food_max = food_max
        self.food_diffusion_rate = food_diffusion_rate
        self.bacteria_consumption_rate = bacteria_consumption_rate
        self.bacteria_growth_rate = bacteria_growth_rate
        self.bacteria_diffusion_rate = bacteria_diffusion_rate
        self.reseed_prob = reseed_prob

        # For tracking system dynamics
        self.food_history = []
        self.bacteria_history = []

        # Figure for visualization
        self.figure = None
        self.axes = None

        # Whether it's for testing
        self.testing_mode = testing_mode

    def initialize_ca(self, food_prob=0.3, bacteria_prob=0.1, bacteria_amount=20):
        """
        Initialize the grid with random food and bacteria.
        """
        # Initialize food randomly by generating a temporary grid for deciding where to put food
        # Initiate with randomly chosen food amount between 0 and 100
        food_mask = np.random.random((self.grid_size, self.grid_size)) < food_prob
        self.food = np.where(food_mask, (self.food_max * np.random.random((self.grid_size, self.grid_size))), 0)

        # Initialize bacteria randomly by generating a temporary grid for deciding where to put bacteria
        # Initiate with 15 bacteria for the grids with bacteria
        bacteria_mask = np.random.random((self.grid_size, self.grid_size)) < bacteria_prob
        self.bacteria = np.where(bacteria_mask, bacteria_amount, 0)

        # Record initial state
        self.food_history = [np.mean(self.food)]
        self.bacteria_history = [np.mean(self.bacteria)]

        if self.testing_mode == True:
            np.set_printoptions(threshold=20)  # Ensures all elements are printed
            print("Initialized CA")
            print("------- Food distribution --------")
            print(self.food)
            print("------- Bacteria population --------")
```

```
        print(self.bacteria)


    def food_growth(self):
        """
        Increase food according to logistic growth formula.
        """
        self.food = self.food * (1 + self.food_growth_rate * (1 - self.food / self.food_max))
        if self.testing_mode == True:
            print("Food grew!")
            print("------- Food distribution --------")
            print(self.food)

    def bacteria_reproduction(self):
        """
        Increase bacteria according to a fixed growth rate.
        """
        self.bacteria = self.bacteria * (1 + self.bacteria_growth_rate)
        if self.testing_mode == True:
            print("Bacteria reproduced!")
            print("------- Bacteria population --------")
            print(self.bacteria)

    def food_reseeding(self):
        """
        Add 1 unit of food to each cell with probability reseed_prob.
        """
        # Decide where to reseed food by generating a temporary grid
        reseed_mask = np.random.random((self.grid_size, self.grid_size)) <
        self.reseed_prob
        self.food = np.where(reseed_mask, self.food + 1, self.food)

        # Ensure food doesn't exceed capacity - over food_max values will be turned to
        food_max
        self.food = np.clip(self.food, 0, self.food_max)
        if self.testing_mode == True:
            print("Food reseed!")
            print("------- Food distribution --------")
            print(self.food)

    def diffusion(self, current_config, diffusion_rate):
        """
        Apply diffusion to food or bacteria.

        Parameters:
            current_config (numpy.ndarray): Grids (100 x 100) to apply diffusion to
```

```
        diffusion_rate (float): Rate of diffusion

    Returns:
        numpy.ndarray: Updated grid after diffusion
    """
    # Amount that diffuses out of each cell
    diffusion_amount = current_config * diffusion_rate

    # Amount that stays in each cell
    next_config = current_config * (1 - diffusion_rate)

    # Amount that diffuses to each of the 4 neighbors (von Neumann neighborhood)
    diff_per_neighbor = diffusion_amount / 4

    # Use roll to shift the grid and add diffusion to neighbors
    next_config += np.roll(diff_per_neighbor, 1, axis=0)  # from cell above
    next_config += np.roll(diff_per_neighbor, -1, axis=0)  # from cell below
    next_config += np.roll(diff_per_neighbor, 1, axis=1)  # from cell left
    next_config += np.roll(diff_per_neighbor, -1, axis=1)  # from cell right

    return next_config

def food_diffusion(self):
    """
    Apply diffusion to the food grid.
    """
    self.food = self.diffusion(self.food, self.food_diffusion_rate)
    if self.testing_mode == True:
        print("Food diffusion!")
        print("------- Food distribution --------")
        print(self.food)

def bacteria_diffusion(self):
    """
    Apply diffusion to the bacteria grid.
    """
    self.bacteria = self.diffusion(self.bacteria, self.bacteria_diffusion_rate)
    if self.testing_mode == True:
        print("Bacteria diffusion!")
        print("------- Bacteria population --------")
        print(self.bacteria)

def consumption_and_startvation(self):
    """
    Handle food consumption, bacteria starvation, and reproduction.
    """
```

```
        # Calculate how much food bacteria need
        food_needed = self.bacteria * self.bacteria_consumption_rate

        # Calculate how much food is actually available (minimum of needed and available)
        food_consumed = np.minimum(food_needed, self.food)

        # Update food after consumption
        self.food -= food_consumed

        if self.testing_mode == True:
            print("Bacteria consume food! yum!")
            print("------- Food distribution --------")
            print(self.food)

        # Calculate how many bacteria survive (those that got food)
        die_out_population = np.ceil((food_needed - food_consumed) /
        self.bacteria_consumption_rate).astype(int)
        self.food = np.clip(self.food, 0, self.food_max)

        # Make sure there is no negative bacteria
        assert np.all(self.bacteria >= 0), "Error: Negative bacteria count detected!"

        # Update bacteria population after starvation
        self.bacteria -= die_out_population
        self.bacteria = np.clip(self.bacteria, 0, None)

        if self.testing_mode == True:
            print("Some bacteria died!")
            print("------- Bacteria population --------")
            print(self.bacteria)

    def update(self):
        """
        Perform one step of the simulation.
        """

        # Bacteria consume food, and die out if they don't have enough food
        self.consumption_and_startvation()

        # The remaining bacteria reproduce
        self.bacteria_reproduction()

        # The remaining food grows
        self.food_growth()

        # Food reseeds with a probability
```

```python
        self.food_reseeding()

        # Food diffuses
        self.food_diffusion()

        # Bacteria diffuses
        self.bacteria_diffusion()

        # Record history
        self.food_history.append(np.mean(self.food))
        self.bacteria_history.append(np.mean(self.bacteria))

        if self.testing_mode == True:
            print("One step done!")

    def run_simulation(self, steps):
        """
        Run the simulation for a specified number of steps.

        Parameters:
            steps (int): Number of steps to run

        Returns:
            tuple: (food_history, bacteria_history) - Lists of average food and bacteria over
            time
        """
        for _ in range(steps):
            self.update()

        return self.food_history, self.bacteria_history

    def setup_visualization(self):
        """Setup the figure and axes for visualization."""
        self.figure, self.axes = plt.subplots(1, 3, figsize=(15, 5))

        # Title for the whole figure
        self.figure.suptitle('Bacteria Growth Simulation', fontsize=16)

        # Setup individual plots
        self.axes[0].set_title(f'Food Distribution \n diffusion rate =
        {self.food_diffusion_rate}, growth rate = {self.food_growth_rate}')
        self.food_plot = self.axes[0].imshow(self.food, cmap='YlGn', vmin=0,
        vmax=self.food_max)
        self.figure.colorbar(self.food_plot, ax=self.axes[0], label='Food Amount')

```

```python
        self.axes[1].set_title(f'Bacteria Population \n diffusion rate = {self.bacteria_diffusion_rate}, growth rate = {self.bacteria_growth_rate}')
        self.bacteria_plot = self.axes[1].imshow(self.bacteria, cmap='RdPu', vmin=0)
        self.figure.colorbar(self.bacteria_plot, ax=self.axes[1], label='Bacteria Amount')

        self.axes[2].set_title(f'Population Dynamic \n consumption rate = {self.bacteria_consumption_rate}')
        self.axes[2].set_xlabel('Time Step')
        self.axes[2].set_ylabel('Average Population per Cell')
        self.axes[2].grid(True)

        # Line plots for average food and bacteria over time
        self.food_line, = self.axes[2].plot([], [], 'g-', label='Food')
        self.bacteria_line, = self.axes[2].plot([], [], 'r-', label='Bacteria')
        self.axes[2].legend()

        plt.tight_layout()

    def observe(self):
        """Update the visualization with current state."""
        # Update the grid plots
        self.food_plot.set_data(self.food)
        self.bacteria_plot.set_data(self.bacteria)

        # Set appropriate range for bacteria plot
        self.bacteria_plot.set_clim(0, np.max(self.bacteria))

        # Update the line plots
        x = range(len(self.food_history))
        self.food_line.set_data(x, self.food_history)
        self.bacteria_line.set_data(x, self.bacteria_history)

        # Adjust y-axis limits if needed
        if len(self.food_history) > 1:
            max_val = max(max(self.food_history), max(self.bacteria_history)) * 1.1
            min_val = min(min(self.food_history), min(self.bacteria_history)) * 0.9
            self.axes[2].set_ylim(min_val, max_val)
            self.axes[2].set_xlim(0, len(self.food_history))

        # Add step count to title
        self.figure.suptitle(f'Bacteria Growth Simulation (Step {len(self.food_history)-1})', fontsize=16)

```

5.2 Appendix B - Plotting and Animating the model

```python

def make_animation(simulation, total_frames, steps_per_frame=1, interval=50):
    """
    Create an animation of the simulation.

    Parameters:
        simulation (BacteriaGrowthSimulator): The simulator to animate
        total_frames (int): Total number of frames in the animation
        steps_per_frame (int): Simulation steps per animation frame
        interval (int): Time between frames in milliseconds

    Returns:
        HTML: Animation that can be displayed in Jupyter notebook
    """
    # Setup visualization
    simulation.setup_visualization()

    # Animation update function
    def update(frame):
        for _ in range(steps_per_frame):
            simulation.update()
        progress_bar.update(1)
        return simulation.observe()

    # Initialize progress bar
    progress_bar = tqdm(total=total_frames)

    # Create animation
    animation = FuncAnimation(
        simulation.figure, update, frames=total_frames,
        interval=interval, blit=True
    )

    # Convert to HTML for display
    html_animation = HTML(animation.to_jshtml())

    return html_animation

def plot_population_dynamics(simulator):
    """
    Plot the food and bacteria population dynamics over time.

    Parameters:
        simulator (BacteriaGrowthSimulator): The simulator after running simulation
```

```python
    """
    plt.figure(figsize=(10, 6))
    plt.plot(simulator.food_history, 'g-', label='Food')
    plt.plot(simulator.bacteria_history, 'r-', label='Bacteria')
    plt.xlabel('Time Step')
    plt.ylabel('Average Population per Cell')
    plt.title('Population Dynamics')
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

```

5.3 Appendix C - Simulation for empirical analysis

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
from matplotlib.ticker import MaxNLocator
from tqdm import tqdm

# Function to run the simulation with varying parameters
def run_simulation(parameter_name, parameter_values, trials = 100):
    """
    Run the bacteria growth simulation with varying parameter values
    and collect data on the converged population sizes.

    Parameters:
    - parameter_name: Name of the parameter being varied
    - parameter_values: List of values for the parameter
    - trials: Number of simulation runs per parameter value for statistical analysis
    - grid_size: Size of the simulation grid
    - steps: Number of simulation steps

    Returns:
    - DataFrame with parameter values and corresponding population statistics
    """

    results = []

    for value in tqdm(parameter_values, desc=f"Testing {parameter_name}"):
        bacteria_final = []
        food_final = []

        for trial in range(trials):
```

```

        if parameter_name == "bacteria_growth_rate":
            # Create a simulator with default parameters
            simulator = BacteriaGrowthSimulator(
                food_growth_rate=0.9,  # Food growth rate
                food_diffusion_rate=0.35,  # Food diffusion rate
                bacteria_consumption_rate=0.5,  # Bacteria consumption rate
                bacteria_growth_rate=value,  # Bacteria growth rate
                bacteria_diffusion_rate=0.3,  # Bacteria diffusion rate
                grid_size = 100
            )

            # Initialize with random food and bacteria
            simulator.initialize_ca(food_prob=0.5, bacteria_prob=0.5,
            bacteria_amount=20)
            food_history, bacteria_history = simulator.run_simulation(steps=5000)

        elif parameter_name == "food_growth_rate":
            # Create a simulator with default parameters
            simulator = BacteriaGrowthSimulator(
                food_growth_rate=value,  # Food growth rate
                food_diffusion_rate=0.35,  # Food diffusion rate
                bacteria_consumption_rate=0.5,  # Bacteria consumption rate
                bacteria_growth_rate=0.2,  # Bacteria growth rate
                bacteria_diffusion_rate=0.3,  # Bacteria diffusion rate
                grid_size = 100
            )

            # Initialize with random food and bacteria
            simulator.initialize_ca(food_prob=0.5, bacteria_prob=0.5,
            bacteria_amount=20)
            food_history, bacteria_history = simulator.run_simulation(steps=5000)

        elif parameter_name == "consumption_rate":
            # Create a simulator with default parameters
            simulator = BacteriaGrowthSimulator(
                food_growth_rate=0.9,  # Food growth rate
                food_diffusion_rate=0.35,  # Food diffusion rate
                bacteria_consumption_rate=value,  # Bacteria consumption rate
                bacteria_growth_rate=0.2,  # Bacteria growth rate
                bacteria_diffusion_rate=0.3,  # Bacteria diffusion rate
                grid_size = 100
            )

            # Initialize with random food and bacteria
```

```
        simulator.initialize_ca(food_prob=0.5, bacteria_prob=0.5,
        bacteria_amount=20)
        food_history, bacteria_history = simulator.run_simulation(steps=5000)

    elif parameter_name == "food_diffusion":
        # Create a simulator with default parameters
        simulator = BacteriaGrowthSimulator(
            food_growth_rate=0.9, # Food growth rate
            food_diffusion_rate=value, # Food diffusion rate
            bacteria_consumption_rate=0.5, # Bacteria consumption rate
            bacteria_growth_rate=0.2, # Bacteria growth rate
            bacteria_diffusion_rate=0.3, # Bacteria diffusion rate
            grid_size = 100
        )

        # Initialize with random food and bacteria
        simulator.initialize_ca(food_prob=0.5, bacteria_prob=0.5,
        bacteria_amount=20)
        food_history, bacteria_history = simulator.run_simulation(steps=5000)

    elif parameter_name == "bacteria_diffusion":
        # Create a simulator with default parameters
        simulator = BacteriaGrowthSimulator(
            food_growth_rate=0.9, # Food growth rate
            food_diffusion_rate=0.35, # Food diffusion rate
            bacteria_consumption_rate=0.5, # Bacteria consumption rate
            bacteria_growth_rate=0.2, # Bacteria growth rate
            bacteria_diffusion_rate=value, # Bacteria diffusion rate
            grid_size = 100
        )

        # Initialize with random food and bacteria
        simulator.initialize_ca(food_prob=0.5, bacteria_prob=0.5,
        bacteria_amount=20)
        food_history, bacteria_history = simulator.run_simulation(steps=5000)

    else:
        # Default pattern
        # Create a simulator with default parameters
        simulator = BacteriaGrowthSimulator(
            food_growth_rate=0.9, # Food growth rate
            food_diffusion_rate=0.35, # Food diffusion rate
            bacteria_consumption_rate=0.5, # Bacteria consumption rate
            bacteria_growth_rate=0.2, # Bacteria growth rate
            bacteria_diffusion_rate=0.3, # Bacteria diffusion rate
            grid_size = 100
```

```
                )

                # Initialize with random food and bacteria
                simulator.initialize_ca(food_prob=0.5, bacteria_prob=0.5,
                bacteria_amount=20)
                food_history, bacteria_history = simulator.run_simulation(steps=5000)

            # Ensure values are positive
            food_history = food_history[4500:]
            bacteria_history = bacteria_history[4500:]

            food_final.append(np.mean(food_history))
            bacteria_final.append(np.mean(bacteria_history))

        # Calculate statistics for this parameter value
        bacteria_mean = np.mean(bacteria_final)
        bacteria_std = np.std(bacteria_final)
        food_mean = np.mean(food_final)
        food_std = np.std(food_final)

        results.append({
            parameter_name: value,
            'bacteria_mean': bacteria_mean,
            'bacteria_std': bacteria_std,
            'food_mean': food_mean,
            'food_std': food_std,
            'total_result': (bacteria_final, food_final)
        })

    return pd.DataFrame(results)

# Define the parameters to investigate
parameters = {
    'bacteria_growth_rate': np.linspace(0.1, 0.9, 9),
    'food_growth_rate': np.linspace(0.1, 0.9, 9),
    'consumption_rate': np.linspace(0.1, 0.9, 9),
    'food_diffusion': np.linspace(0.1, 0.9, 9),
    'bacteria_diffusion': np.linspace(0.1, 0.9, 9),
}

# Run simulations and collect results
all_results = {}
for param_name, param_values in parameters.items():
    all_results[param_name] = run_simulation(param_name, param_values)
```

5.4 Appendix D - Using the empirical analysis result for theoretical analysis

```python
# Function to analyze the relationship between parameters and population sizes
def analyze_relationships(results):
    """
    Analyze and quantify the relationships between parameters and population sizes.
    """
    analysis_results = {}

    for param_name, df in results.items():
        # Calculate correlation
        bacteria_corr = np.corrcoef(df[param_name], df['bacteria_mean'])[0, 1]
        food_corr = np.corrcoef(df[param_name], df['food_mean'])[0, 1]

        # Fit linear regression
        bacteria_slope, bacteria_intercept, bacteria_r, _, _ =
        stats.linregress(df[param_name], df['bacteria_mean'])
        food_slope, food_intercept, food_r, _, _ = stats.linregress(df[param_name],
        df['food_mean'])

        analysis_results[param_name] = {
            'bacteria_correlation': bacteria_corr,
            'bacteria_slope': bacteria_slope,
            'bacteria_intercept': bacteria_intercept,
            'bacteria_r_squared': bacteria_r**2,
            'food_correlation': food_corr,
            'food_slope': food_slope,
            'food_intercept': food_intercept,
            'food_r_squared': food_r**2
        }

    return pd.DataFrame.from_dict(analysis_results, orient='index')

# Analyze the relationships
relationship_analysis = analyze_relationships(all_results)
relationship_analysis.head(5)

```

5.5 Appendix E - Fitting an exponential line to the model

```python
from scipy.optimize import curve_fit

def fit_exponential_to_consumption_rate(results):
    # Get consumption rate data
    df = results['consumption_rate']
    x = df['consumption_rate'].values
```

```python
    y = df['bacteria_mean'].values

    # Define exponential function
    def exp_func(x, a, b, c):
        return a * np.exp(b * x) + c

    # Fit the model
    params, _ = curve_fit(exp_func, x, y, p0=[50.0, -5.0, 5.0], maxfev=10000)
    a, b, c = params

    # Calculate R-squared
    y_pred = exp_func(x, a, b, c)
    ss_res = np.sum((y - y_pred)**2)
    ss_tot = np.sum((y - np.mean(y))**2)
    r_squared = 1 - (ss_res / ss_tot)

    # Create a simple plot
    plt.figure(figsize=(8, 5))
    plt.scatter(x, y, color='mediumvioletred', label='Data')

    # Plot the fitted curve
    x_curve = np.linspace(min(x), max(x), 100)
    y_curve = exp_func(x_curve, a, b, c)
    plt.plot(x_curve, y_curve, 'crimson', label=f'Exponential fit:
    {a:.2f}*exp({b:.2f}*x)+{c:.2f}')

    plt.xlabel('Consumption Rate')
    plt.ylabel('Bacteria Population')
    plt.title(f'Bacteria vs Consumption Rate (R² = {r_squared:.4f})')
    plt.legend()
    plt.grid(True)

    # Return just the key results
    return {
        'r_squared': r_squared,
        'a': a,
        'b': b,
        'c': c,
        'equation': f'{a:.4f} * exp({b:.4f} * x) + {c:.4f}'
    }
results = fit_exponential_to_consumption_rate(all_results)
print(results)
```