Food-Bacteria Growth Model

Food-Bacteria Growth is a grid-based simulation that models the interplay between food quantity and bacterium population size within each grid cell. The simulation is governed by a set of parameters, variables, and update rules that define its behavior.

Parameters

- *Growth rate of food*
- *Food capacity for each cell*
- *Probability for food reseeding*
- *Diffusion rate of food*
- *Consumption rate*
- *Growth rate for bacteria*
- *Bacteria diffusion rate*

Variables

- *Grid*: The grid on which the simulation runs.
- *Food*: The amount of food in each grid cell (real number).
- *Bacteria Population*: The size of the bacterium population in each grid cell (real number).

Update Rules

1. *Food Growth*: Increase the amount of food in each cell according to the logistic growth formula, which involves the growth rate, current food amount, and food capacity.
2. *Food Reseeding*: Add 1 unit of food to each grid cell with a defined probability.
3. *Food Diffusion*: Food units migrate from each cell to its 4 neighbors, simulating the spreading of plant-like food. The diffusion rate determines the extent of this migration.
4. *Consumption*: Bacteria consume food based on the consumption rate. In cases of insufficient food, bacteria may die, while well-fed bacteria reproduce.
5. *Starvation*: Bacteria lacking adequate food quantities perish.
6. *Reproduction*: Well-nourished bacteria reproduce at a specified growth rate.
7. *Bacteria Diffusion*: Bacteria move across the grid in a similar manner to food, but with a different diffusion rate.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
import random
matplotlib.rcParams['animation.embed_limit'] = 2**128
```

In [ ]:
```
class Model:
    def __init__(self, size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, consumption_rate, food_capacity=100):
        """
        Initialize the model with the specified parameters and create an empty grid.
```

```
        Args:
        size (int): The size of the grid (grid is size x size).
        food_growth_rate (float): Rate of food growth.
        bacteria_growth_rate (float): Growth rate of bacteria.
        food_diff (float): Food diffusion rate.
        bacteria_diff (float): Bacteria diffusion rate.
        reseed_prob (float): Probability of reseeding food in an empty cell.
        consumption_rate (float): Rate at which bacteria consume food.
        food_capacity (float): Maximum food capacity for each cell.

        """
        self.size = size
        self.food_growth_rate = food_growth_rate
        self.bacteria_growth_rate = bacteria_growth_rate
        self.bacteria_diff = bacteria_diff
        self.food_diff = food_diff
        self.reseed_prob = reseed_prob
        self.consumption_rate = consumption_rate
        self.food_capacity = food_capacity
        self.grid = np.zeros((self.size, self.size, 2))  # Initialize an empty grid for food and bacteria.

    def initialize(self):
        """
        Initialize the grid by placing random food values in some cells.

        This method randomly selects cells and assigns them a random food value.
        It can be seen as the initial seeding of the environment with food.

        """
        row, col = random.randint(0, self.size - 1), random.randint(0, self.size - 1)
        random_value = [random.uniform(0, self.food_capacity), random.uniform(0, self.food_capacity)]
        self.grid[row, col] = random_value

    def food_grow(self):
        """
        Simulate the growth of food in the grid.

        This method models the growth of food in each cell of the grid based on
        the specified food growth rate. The food level increases while respecting
        the maximum food capacity for each cell.

        """
        for x in range(self.size):
            for y in range(self.size):
                food, bacteria = self.grid[x, y]
                growth_rate = self.food_growth_rate
                new_food = food + growth_rate * food * (1 - food / self.food_capacity)
                new_food = min(new_food, self.food_capacity)  # Limit the food value to food_capacity
                self.grid[x, y] = new_food, bacteria

    def diffuse_food(self):
        """
        Simulate the diffusion of food in the grid.

        This method calculates how food diffuses from one cell to neighboring cells
        based on the specified diffusion rate. It updates the current cell's food level
```

```
        and the food levels in neighboring cells.

        """
        for x in range(self.size):
            for y in range(self.size):
                if self.grid[x, y][0] != 0.0:
                    food_diffusion = self.food_diff * self.grid[x, y][0]

                    # Update the food level in the current cell
                    self.grid[x, y][0] -= food_diffusion

                    # Update neighboring cells with a fraction of the diffused food
                    for dx in [-1, 1, 0, 0]:
                        for dy in [0, 0, -1, 1]:
                            new_x = (x + dx) % self.size
                            new_y = (y + dy) % self.size
                            self.grid[new_x, new_y][0] += 0.25 * food_diffusion

    def diffuse_bacteria(self):
        """
        Simulate the diffusion of bacteria in the grid.

        This method calculates how bacteria diffuse from one cell to neighboring cells
        based on the specified diffusion rate. It updates the current cell's bacteria count
        and the bacteria counts in neighboring cells.

        """
        for x in range(self.size):
            for y in range(self.size):
                if self.grid[x, y][1] != 0:
                    bacteria_diffusion = self.bacteria_diff * self.grid[x, y][1]

                    # Update the bacteria count in the current cell
                    self.grid[x, y][1] -= bacteria_diffusion

                    # Update neighboring cells with a fraction of the diffused bacteria
                    for dx in [-1, 1, 0, 0]:
                        for dy in [0, 0, -1, 1]:
                            new_x = (x + dx) % self.size
                            new_y = (y + dy) % self.size
                            self.grid[new_x, new_y][1] += 0.25 * bacteria_diffusion

    def reseed(self):
        """
        Simulate the reseeding of food in the grid.

        This method checks empty cells in the grid and reseeds some of them with food
        based on the specified reseeding probability.

        """
        for x in range(self.size):
            for y in range(self.size):
                if self.grid[x, y][0] == 0:
                    if np.random.random() < self.reseed_prob:
                        self.grid[x, y][0] = 1
                    else:
```

```
                    self.grid[x, y][0] = 0

    def reproduce_bacteria(self, bacteria_present):
        """
        Calculate the reproduction of bacteria in a cell.

        This method calculates the new bacteria count based on the current bacteria count
        and the specified bacteria growth rate.

        Args:
        bacteria_present (float): Current bacteria count in a cell.

        Returns:
        float: New bacteria count after reproduction.

        """
        new_bacteria = bacteria_present * (1 + self.bacteria_growth_rate)
        return new_bacteria

    def consume(self):
        """
        Simulate the consumption of food by bacteria in the grid.

        This method models how bacteria consume food in the grid. It checks the amount of
        food required by bacteria in each cell and updates the food and bacteria counts
        accordingly.

        """
        for x in range(self.size):
            for y in range(self.size):
                bacteria_present = self.grid[x, y][1]
                food_required = self.consumption_rate * bacteria_present
                food_available = self.grid[x, y][0]

                if food_available != 0:
                    if food_required <= food_available:
                        # There's enough food for bacteria, so they consume it
                        self.grid[x, y][0] -= food_required
                        self.grid[x, y][1] = self.reproduce_bacteria(bacteria_present)
                    else:
                        # Bacteria consume as much food as available
                        bacteria_fed = food_available / self.consumption_rate
                        self.grid[x, y][0] -= food_available
                        self.grid[x, y][1] = self.reproduce_bacteria(bacteria_fed)
                else:
                    # No food available, so bacteria count becomes zero
                    self.grid[x, y][1] = 0

    def run(self,steps):
        """
        Run the simulation for a single time step.

        This method combines all the simulation steps for one time step, including
        food growth, reseeding, food diffusion, bacteria consumption, and bacteria diffusion.

        Returns:
```

```
        numpy.ndarray: Updated grid after the simulation step.

        """
        self.initialize()
        for i in range(steps):
            self.food_grow()  # Simulate food growth
            self.reseed()  # Reseed empty cells with food
            self.diffuse_food()  # Simulate food diffusion
            self.consume()  # Simulate bacteria consumption
            self.diffuse_bacteria()  # Simulate bacteria diffusion
            self.grid = np.nan_to_num(self.grid)  # Remove any NaN values from the grid

        return self.grid
```

Visualization

To observe how the system evolves over time, the simulation was run for 500 time steps. The 'create_animation' method animates the results of the simulation so the changes in the system can be visualized.

In [ ]:
```
def create_animation(model, total_frames, steps_per_frame=1, interval=100):
    """
    Create an animation of bacteria and food populations over time.

    Parameters:
    - model: Model object representing the simulation.
    - total_frames: Total number of animation frames.
    - steps_per_frame: Number of simulation steps per frame.
    - interval: Time interval between frames in milliseconds.

    Returns:
    - HTML animation of the simulation.
    """
    model.initialize()  # Initialize the model

    def update(frame):
        for _ in range(steps_per_frame):
            model.food_grow()  # Simulate food growth
            model.reseed()  # Reseed empty cells with food
            model.diffuse_food()  # Simulate food diffusion
            model.consume()  # Simulate bacteria consumption
            model.diffuse_bacteria()  # Simulate bacteria diffusion
            model.grid = np.nan_to_num(model.grid)  # Remove any NaN values

        # Update the bacteria and food plots
        bacteria_plot.set_data(model.grid[:, :, 1])  # Update the bacteria plot data
        food_plot.set_data(model.grid[:, :, 0])  # Update the food plot data
        return [bacteria_plot, food_plot]  # Return updated plots

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))  # Create a figure with two subplots
    ax1.set_title('Bacteria Population')  # Set title for the first subplot
    ax2.set_title('Food Population')  # Set title for the second subplot

    # Create the initial bacteria and food plots
    bacteria_plot = ax1.imshow(model.grid[:, :, 1], cmap='inferno')  # Initial bacteria plot
    food_plot = ax2.imshow(model.grid[:, :, 0], cmap='inferno')  # Initial food plot
    fig.colorbar(bacteria_plot, ax=ax1, orientation='vertical', label='Bacteria')  # Add colorbar to bacteria plot
```

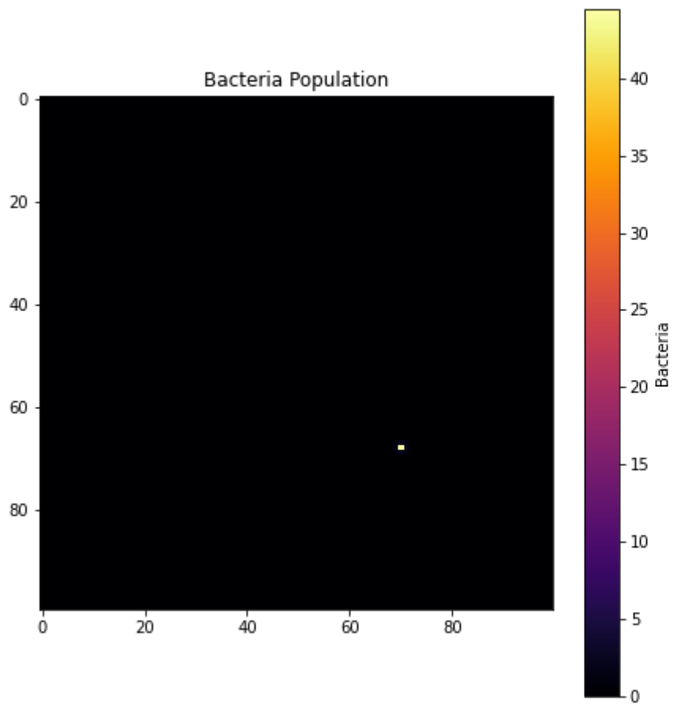
```
    fig.colorbar(food_plot, ax=ax2, orientation='vertical', label='Food')  # Add colorbar to food plot
    fig.set_facecolor("white")  # Set the figure background color

    anim = FuncAnimation(fig, update, frames=total_frames, repeat=False)  # Create the animation

    return HTML(anim.to_jshtml())  # Return the animation as HTML
```

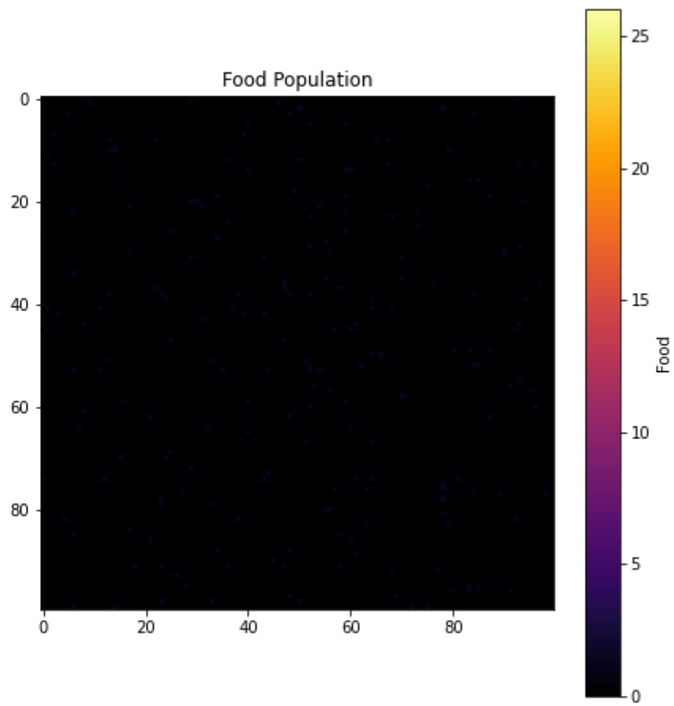
In [ ]:
```
# input parameters

size = 100
food_growth_rate = 0.2
bacteria_growth_rate = 0.3
food_diff = 0.01
bacteria_diff = 0.05
reseed_prob = 0.02
consumption_rate = 0.4
steps = 500
total_frames = 500
```

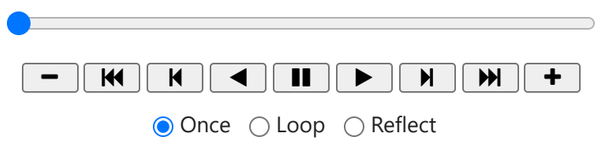
Food and Bacteria: A Harmonic Balance

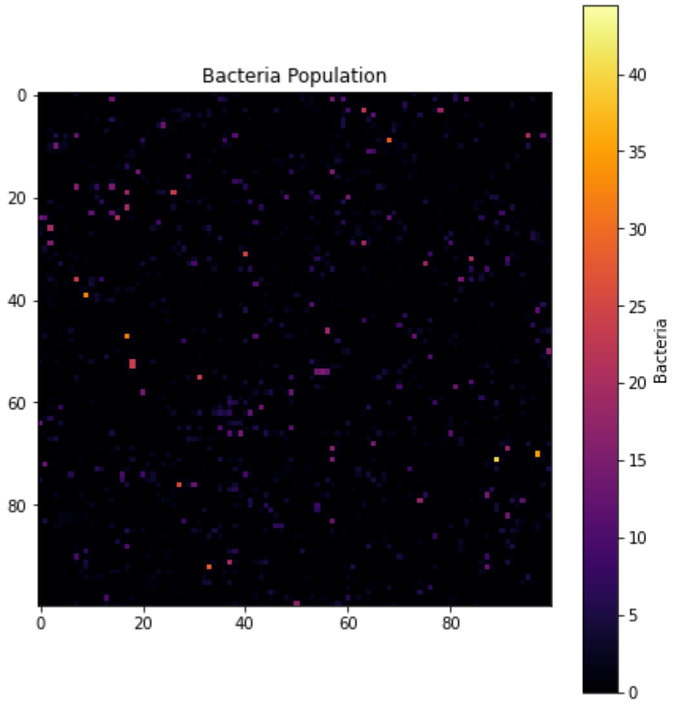
The set paramaters results in the emergence of a beautiful pattern in the grid, a harmonic balance of food and bacteria thriving in a grid. As the simulation progresses, the population of food increases exponentially while the population of bacterium increases at a much slower rate. As the bacterium population grows, the food population decreases exponentially, in turn triggering a decline in the bacterium population. The system reaches equilibrium after approximately 225 steps. Interestingly, at equilibrium, the bacterium and food populations not only oscillate around a target value - they also converge quite close to each other, growing and declining in harmony.

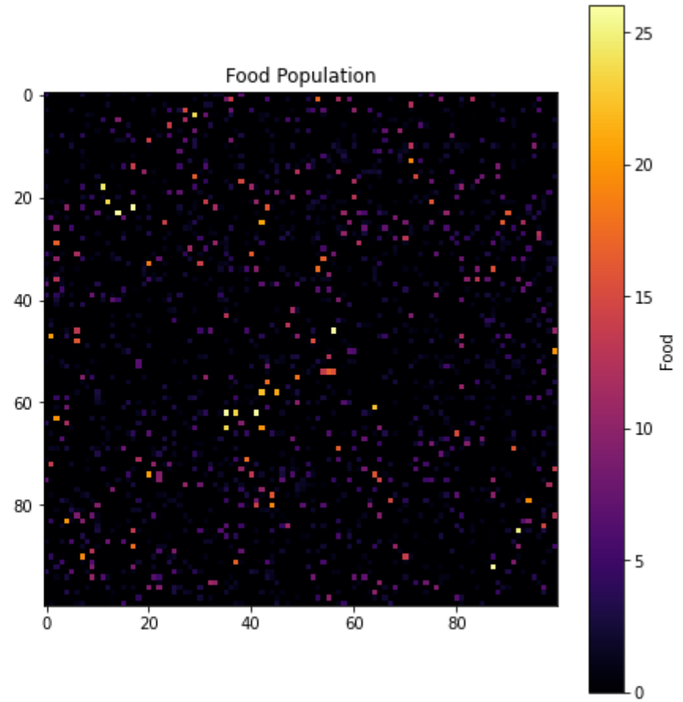
In [ ]:
```
model = Model(
    size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, consumption_rate)

# Create a bacteria animation
animation = create_animation(model, total_frames)

# Display the animation
display(animation)
```


Bacteria Population
0
20
40
60
80
0
20
40
60
80
40
35
30
25
20
15
10
5
0
Bacteria


Food Population
0
20
40
60
80
0
20
40
60
80
25
20
15
10
5
0
Food

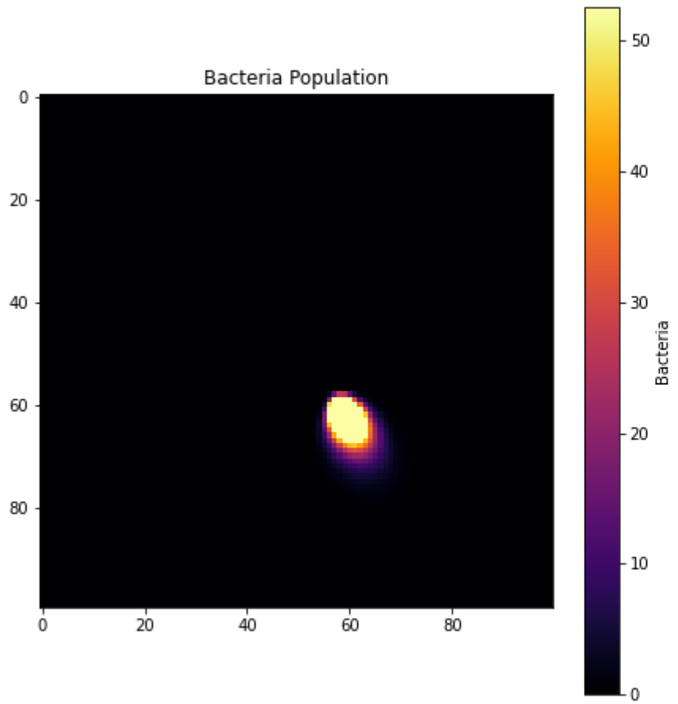

Once
Loop
Reflect

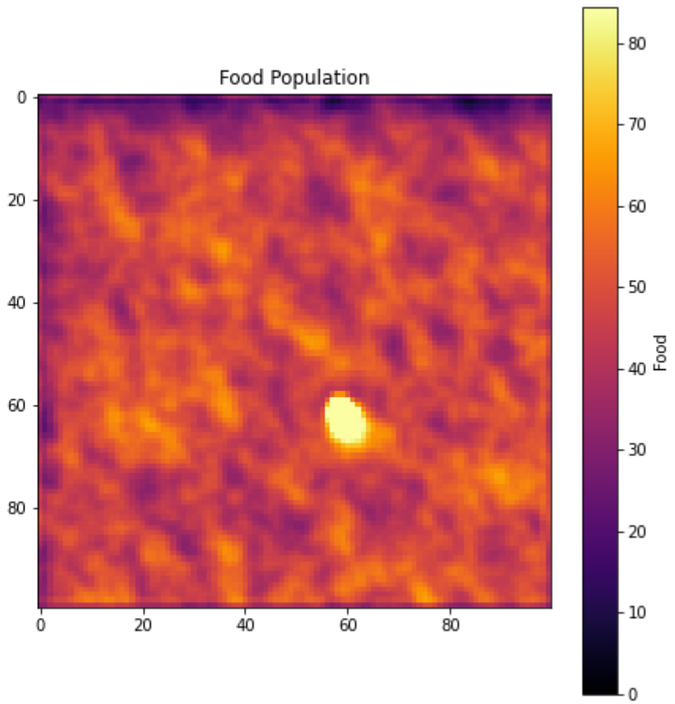


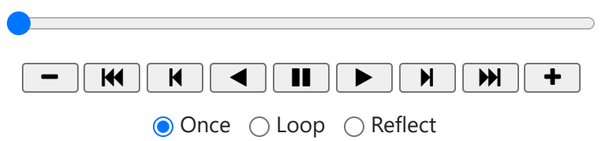


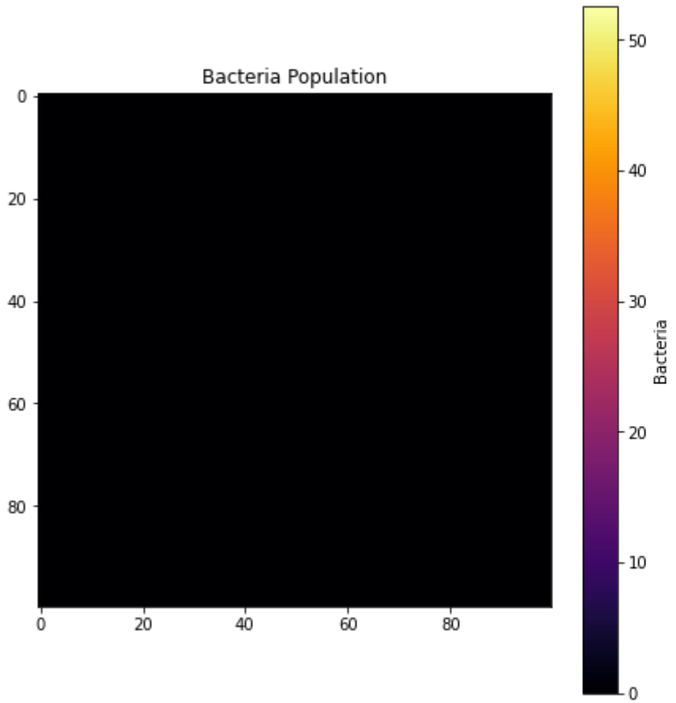
In [ ]:
```
#Test Case 2: Extreme values of food and bacteria growth

model = Model(
    100, 1, 1, 0.5, 0.5, 0.5, 1)

# Create a bacteria animation
animation = create_animation(model, total_frames)

# Display the animation
display(animation)
```

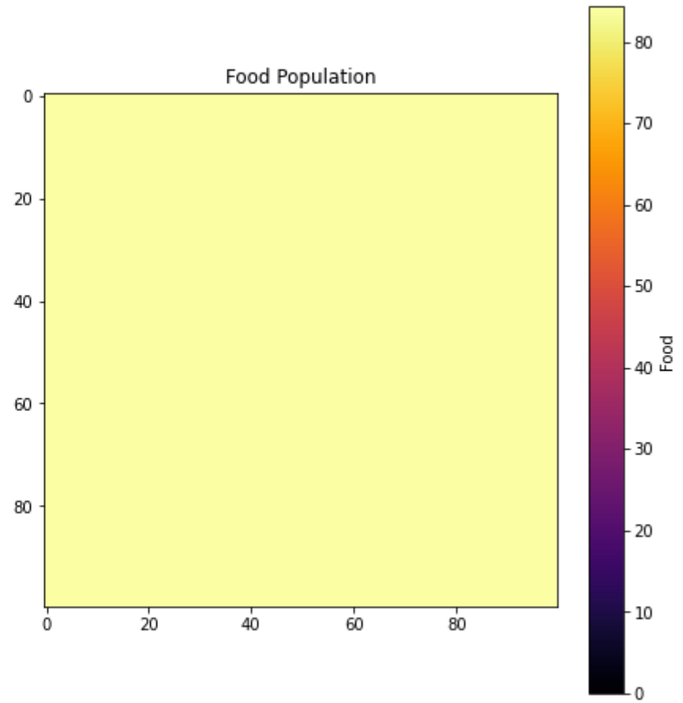
```
<ipython-input-53-02a76fb7e7a4>:52: RuntimeWarning: overflow encountered in double_scalars
  new_food = food + growth_rate * food * (1 - food / self.food_capacity)
<ipython-input-53-02a76fb7e7a4>:71: RuntimeWarning: invalid value encountered in double_scalars
  self.grid[x, y][0] -= food_diffusion
```


Bacteria Population
0
20
40
60
80
0
20
40
60
80
50
40
30
20
10
0
Bacteria


Food Population
0
20
40
60
80
0
20
40
60
80
80
70
60
50
40
30
20
10
0
Food


Once
Loop
Reflect





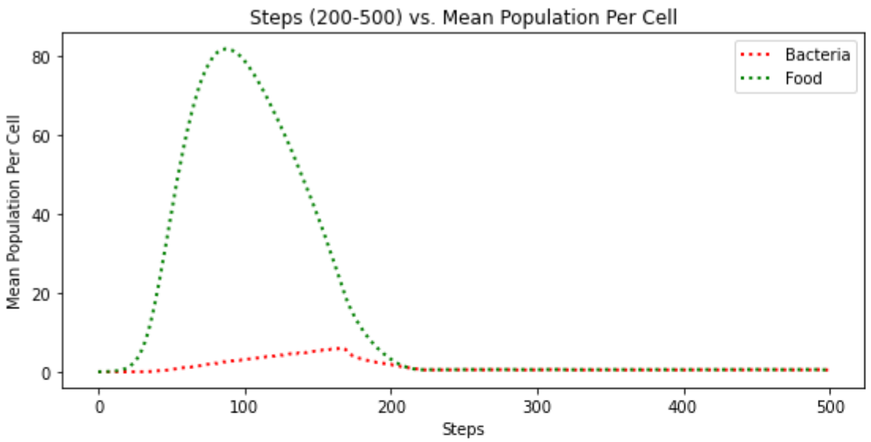
Change in Mean Populations with Increasing Number of Steps

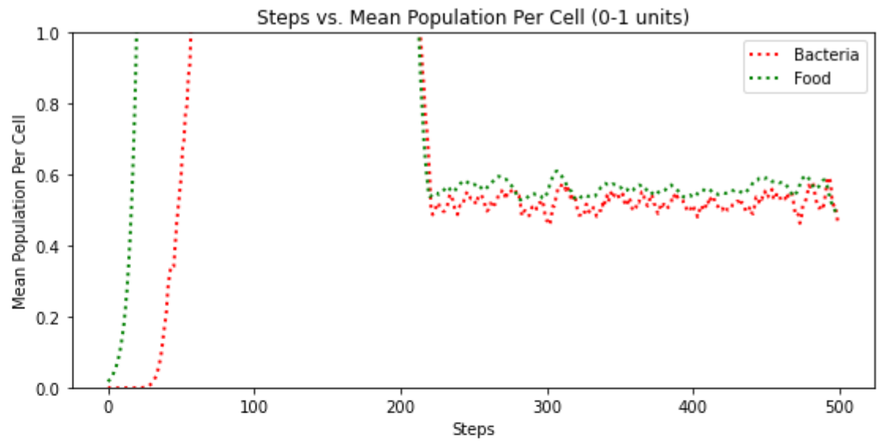
To see how the mean populations of food and bacterium change respectively with an increasing number of steps, the simulation was run for a toal of 500 steps and the mean population was recorded after each step. Both the populations of food and bacteria experience a sharp increase, peak, and decline within 200 steps (albeit at different rates) before stabilizing around an average value not far from each other. From top to bottom and left to righ, the four plots below show an overview of the evolution of population mean, the difference in the peaks of food and bacterium population, the stabilization, and the difference in stable mean populations for food and bacterium.

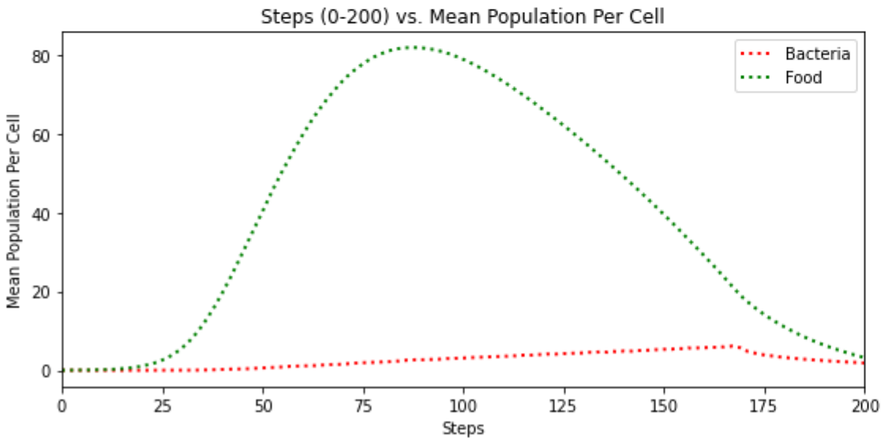
In [ ]:
```python
# Create lists to store results
mean_populations_bacteria_growth = []  # List to store mean population of bacteria
mean_populations_food_growth = []  # List to store mean population of food

def empirical_analysis(size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, num_steps):
    """
    Perform an empirical analysis of a model's behavior over a specified number of time steps.

    Parameters:
    size (int): The size of the model grid.
    food_growth_rate (float): The growth rate of food in the model.
    bacteria_growth_rate (float): The growth rate of bacteria in the model.
    food_diff (float): The diffusion rate of food in the model.
    bacteria_diff (float): The diffusion rate of bacteria in the model.
    reseed_prob (float): The probability of food reseeding.
    num_steps (int): The number of time steps to simulate.

    Returns:
    Tuple (List[float], List[float]): A tuple containing lists of mean population of bacteria and food over time.
    """
```

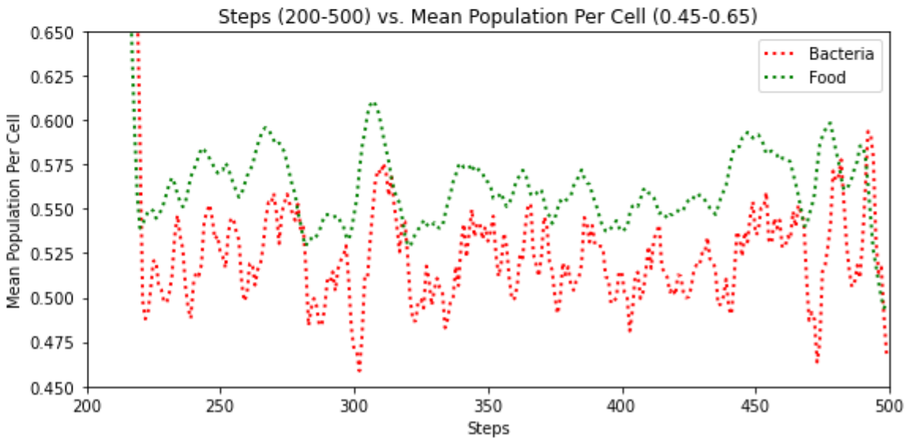
```
    # Initialize the model with the given parameters
    sim = Model(size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, consumption_rate)
    sim.initialize()

    # Lists to store the mean population of bacteria and food over time
    mean_populations_bacteria_growth = []
    mean_populations_food_growth = []

    # Simulate the model over the specified number of time steps
    for i in range(num_steps):
        sim.food_grow()
        sim.reseed()
        sim.diffuse_food()
        sim.consume()
        sim.diffuse_bacteria()
        sim.grid = np.nan_to_num(sim.grid)
        mean_population_bacteria = np.mean(sim.grid[:, :, 1])
        mean_population_food = np.mean(sim.grid[:, :, 0])
        mean_populations_bacteria_growth.append(mean_population_bacteria)
        mean_populations_food_growth.append(mean_population_food)

    return mean_populations_bacteria_growth, mean_populations_food_growth



# Running empirical analysis on step plot
mean_populations_bacteria_growth, mean_populations_food_growth = empirical_analysis(size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, steps)


# Create a figure with subplots
fig, axs = plt.subplots(2, 2, figsize=(16, 8))

# Subplot 1: Mean population per cell over time
axs[0, 0].plot(mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
axs[0, 0].plot(mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)
axs[0, 0].set_title('Steps (0-500) vs. Mean Population Per Cell')
axs[0, 0].set_xlabel('Steps')
axs[0, 0].set_ylabel('Mean Population Per Cell')
axs[0, 0].legend()

# Subplot 2: Focus on y values 0 to 1
axs[0, 1].plot(mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
axs[0, 1].plot(mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)
axs[0, 1].set_title('Steps vs. Mean Population Per Cell (0-1 units)')
axs[0, 1].set_xlabel('Steps')
axs[0, 1].set_ylabel('Mean Population Per Cell')
axs[0, 1].set_ylim(0, 1)
axs[0, 1].legend()

# Subplot 3: Focus on x values 0 to 200
axs[1, 0].plot(mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
axs[1, 0].plot(mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)
axs[1, 0].set_xlim(0,200)
axs[1, 0].set_title('Steps (0-200) vs. Mean Population Per Cell')
axs[1, 0].set_xlabel('Steps')
```

```
axs[1, 0].set_ylabel('Mean Population Per Cell')
axs[1, 0].legend()

# Subplot 4: Focus on y values 0.45 to 0.65 and x values 200 to 500
axs[1, 1].plot(mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
axs[1, 1].plot(mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)
axs[1, 1].set_title('Steps (200-500) vs. Mean Population Per Cell (0.45-0.65)')
axs[1, 1].set_xlabel('Steps')
axs[1, 1].set_ylabel('Mean Population Per Cell')
axs[1,1].set_xlim(200,500)
axs[1,1].set_ylim(0.45,0.65)
axs[1, 1].legend()

# Adjust layout to prevent overlapping
plt.tight_layout()

# Display the subplots
plt.show()
```









Out[ ]:
```
"\nplt.figure(figsize=(16, 8))  # Create a figure with a specified size\n\n# Create line plots for mean populations of bacteria (red) and food (green)\nplt.plot(mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)\nplt.plot(mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)\nplt.ylim(0,1)\n\n# Set plot title, x-axis label, and y-axis label\nplt.title('Steps vs. Mean Population Per Cell')\nplt.xlabel('Steps')\nplt.ylabel('Mean Population Per Cell')\n\n# Add a legend to the plot\nplt.legend()\n\n# Display the plot\nplt.show()\n"
```

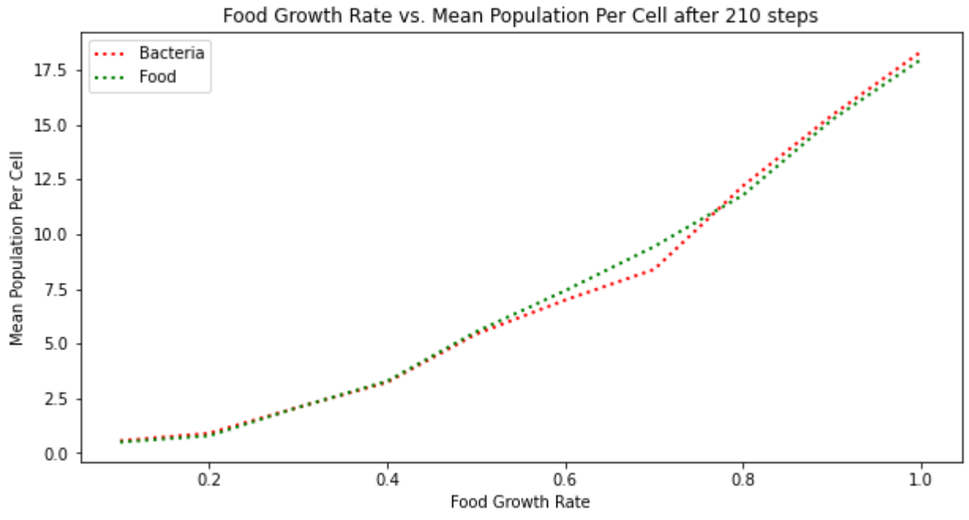
In [ ]:
```
# A function to run the simulation with different parameter values
def run_simulation(size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, num_steps):
    sim = Model(size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, consumption_rate)
    grid = sim.run(num_steps)  # Run the simulation for a number of steps
    # Calculate the mean populations per cell
    mean_population_bacteria = np.mean(grid[:, :, 1])
    mean_population_food = np.mean(grid[:, :, 0])
    return mean_population_bacteria, mean_population_food

# Define parameter ranges to explore
bacteria_growth_rates = np.linspace(0.1, 1.0, 10)  # Vary bacteria growth rate from 0.1 to 1.0
food_growth_rates = np.linspace(0.1, 1.0, 10)  # Vary food growth rate from 0.1 to 1.0
reseed_probs = np.linspace(0.01, 0.1, 10)  # Vary reseed probability from 0.01 to 0.1
```

Food Growth Rate and Mean Populations

For 210 time steps, the food growth rate was varied from 0.1 to 1.0 while other parameters were held constant to evaluate its effect on mean populations of both food and bacterium. The result of the experiment (as visualized in the line plot below) are quite intuitive. The mean populations of both food and bacterium are proportional to the food growth rate. A higher food growth rate results in a larger mean population value for both food and bacterium. Bacterium population goes from exceeding the food population to matching it to deceeding and, finally, to exceeding the food population as the food growth rate increases. The mean populations approximately match each other which is not surprisingly, given how the availability of food is the single most important limiting factor for bacteria growth. Hence, as the food growth rate increases, the mean population of both food and bacterium increase.

In [ ]:
```
steps = 210

num_rates = len(food_growth_rates)
mean_populations_bacteria_growth = np.zeros(num_rates)  # Array to store the mean population of bacteria
mean_populations_food_growth = np.zeros(num_rates)      # Array to store the mean population of food

# Loop through different food growth rates in the 'food_growth_rates' list
for i, fg_rate in enumerate(food_growth_rates):
    # Run a simulation with specified parameters and return the means directly
    mean_pop_bacteria, mean_pop_food = run_simulation(size, fg_rate, bacteria_growth_rate, food_diff, bacteria_diff, reseed_prob, steps)

    # Assign the calculated mean population values to the respective arrays
    mean_populations_bacteria_growth[i] = mean_pop_bacteria
    mean_populations_food_growth[i] = mean_pop_food

# Create a plot with a specified size
plt.figure(figsize=(10, 5))

# Create line plots for mean populations of bacteria (red) and food (green)
plt.plot(food_growth_rates, mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
plt.plot(food_growth_rates, mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)

# Set plot title, x-axis label, and y-axis label
plt.title(f'Food Growth Rate vs. Mean Population Per Cell after {steps} steps')
plt.xlabel('Food Growth Rate')
plt.ylabel('Mean Population Per Cell')
```

```
# Add a legend to the plot to distinguish between bacteria and food
plt.legend()

# Display the plot
plt.show()
```

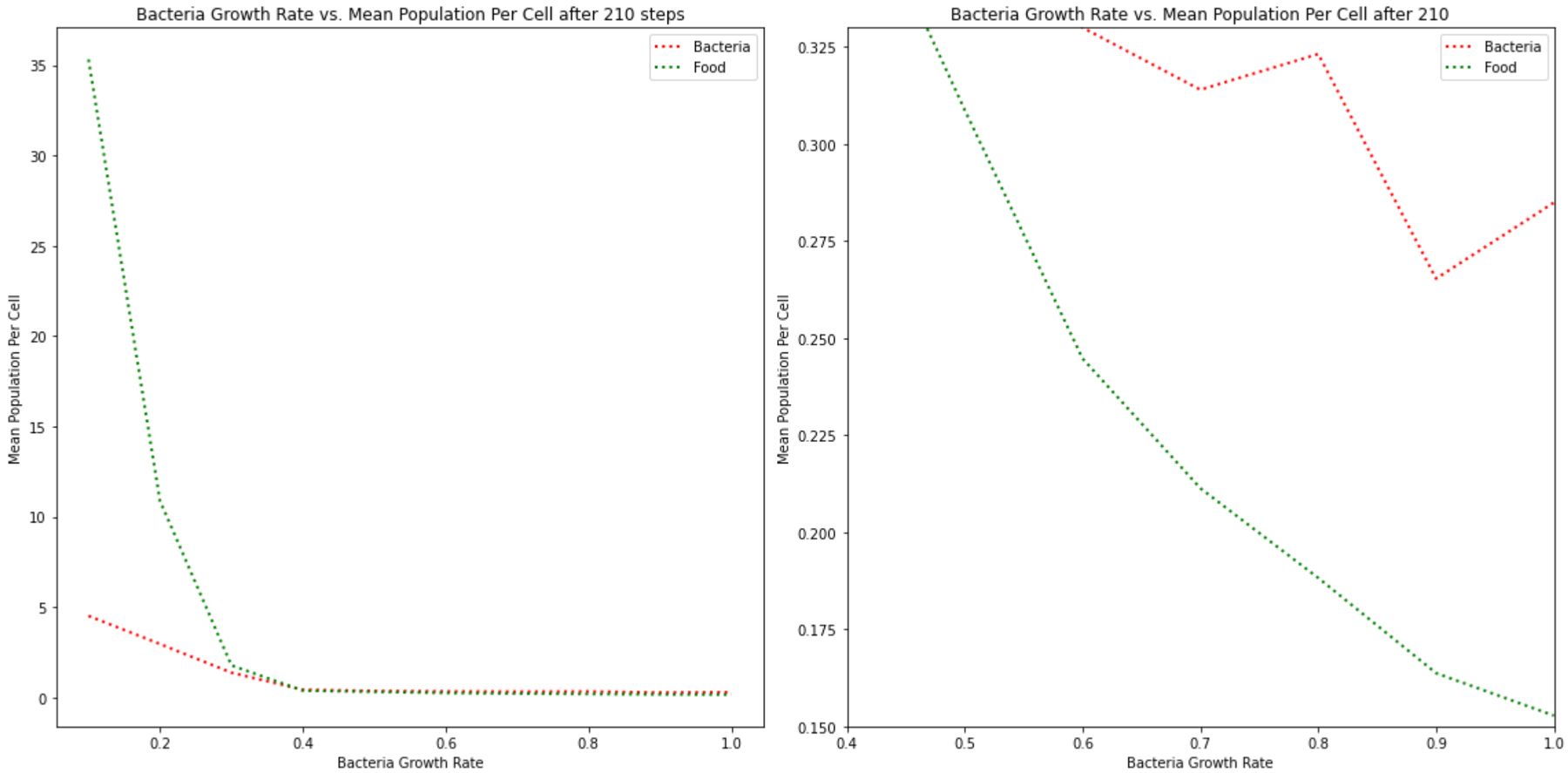


Bacteria Growth Rate and Mean Populations

As the bacteria growth rate increases, the mean food and bacterium populations both decrease. Again, this result is not surprising because an increase in bacterial growth means greater consumption of food, which in turn acts as a limiting factor for further bacterial growth. Zooming in shows that the mean food population decreases at a higher rate than the mean bacterial population.

In [ ]:
```
steps = 210

# Create arrays to store the results of the simulation
mean_populations_bacteria_growth = np.empty_like(bacteria_growth_rates)
mean_populations_food_growth = np.empty_like(bacteria_growth_rates)

# Run simulations for all bacteria growth rates at once
for i, bg_rate in enumerate(bacteria_growth_rates):
    # Run simulations and store the results in corresponding arrays
    mean_populations_bacteria_growth[i], mean_populations_food_growth[i] = run_simulation(size, food_growth_rate, bg_rate, food_diff, bacteria_diff, reseed_prob, steps)


# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(16, 8))

# Subplot 1: Entire View
axs[0].plot(bacteria_growth_rates, mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
axs[0].plot(bacteria_growth_rates, mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)

axs[0].set_title(f'Bacteria Growth Rate vs. Mean Population Per Cell after {steps} steps')
axs[0].set_xlabel('Bacteria Growth Rate')
axs[0].set_ylabel('Mean Population Per Cell')
```

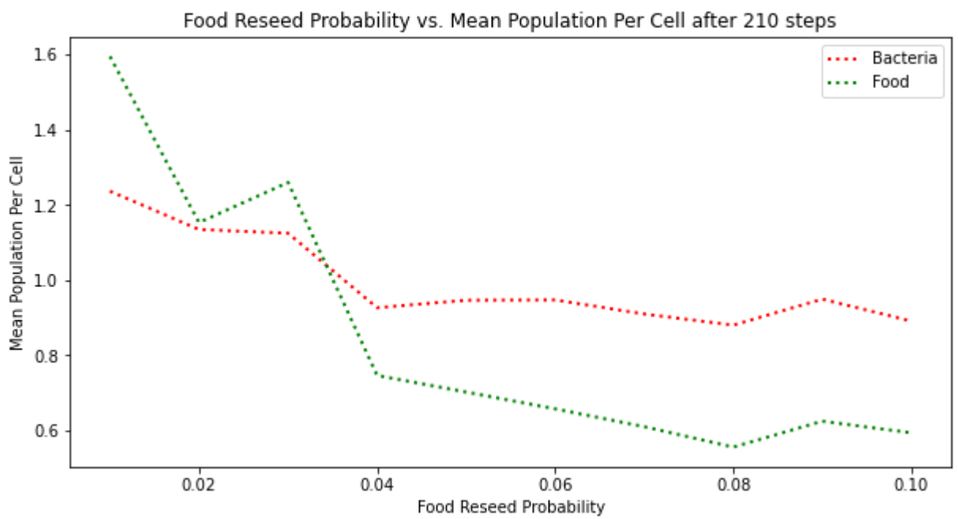
```
axs[0].legend()

# Subplot 2: Zoomed View (0 to 3)
axs[1].plot(bacteria_growth_rates, mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
axs[1].plot(bacteria_growth_rates, mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)

axs[1].set_title(f'Bacteria Growth Rate vs. Mean Population Per Cell after {steps}')
axs[1].set_xlabel('Bacteria Growth Rate')
axs[1].set_ylabel('Mean Population Per Cell')
axs[1].set_ylim(0.15,0.33)  # Set the y-axis limits for the zoomed view
axs[1].set_xlim(0.4,1)
axs[1].legend()

# Adjust layout to prevent overlap
plt.tight_layout()

# Display the plot
plt.show()
```



Reseed Probability and Mean Populations

Reseed probability is one of the factors that determines the availability of food in the grid. It represents the probability of growth upon initialization of an empty grid and of regrowth once the food has been eliminated by bacteria in the cell. Increasing the reseed probability decreases the mean population of both the food and bacterium in the grid recorded after 210 steps. Beyond the reseed probability of 0.03, the mean population of food becomes less than the mean population of bacteria. This result is also not surprising, as the population of bacteria depends on the probability of food sprouting in different parts of the grid. As grid cells activate for the first time or rejuvenate, the population of bacteria through diffusion is sustained, resulting in a higher mean population of bacterium than that of food.

In [ ]:
```
steps = 210

# Create arrays to store the results of the simulation
mean_populations_bacteria_growth = np.empty_like(reseed_probs)
mean_populations_food_growth = np.empty_like(reseed_probs)

# Run simulations for all reseed probabilities at once
for i, rp in enumerate(reseed_probs):
    # Run simulations and store the results in corresponding arrays
    mean_populations_bacteria_growth[i], mean_populations_food_growth[i] = run_simulation(size, food_growth_rate, bacteria_growth_rate, food_diff, bacteria_diff, rp, steps)

# Create a plot with a specified size
plt.figure(figsize=(10, 5))

# Create line plots for mean populations of bacteria (red) and food (green)
plt.plot(reseed_probs, mean_populations_bacteria_growth, linestyle='dotted', color='red', label='Bacteria', lw=2)
plt.plot(reseed_probs, mean_populations_food_growth, linestyle='dotted', color='green', label='Food', lw=2)

# Set plot title, x-axis label, and y-axis label
plt.title(f'Food Reseed Probability vs. Mean Population Per Cell after {steps} steps')
plt.xlabel('Food Reseed Probability')
plt.ylabel('Mean Population Per Cell')

# Add a legend to the plot to distinguish between bacteria and food
plt.legend()

# Display the plot
plt.show()
```



Theoretical Analysis

The bacterium population in a cell depends on the amount of food available for the bacterium, its reproduction, and the diffusion of bacterium in and out of the cell. These three variables can be linked together to find a mathematical relationship between the food growth rate parameter, f_g, and the bacterium population in the cell over time, n_b. This mathematical relationship assumes that the system has reached equilibrium and the food and bacterium populations oscillate around a mean value.

Because only the bacteria that get access to food surive and reproduce, the bacterium population directly depends on the amount of food available. In cases where the food available is less than the food required by all the bacteria in the cell, only a fraction of the bacterium population survives and reproduce. This fraction is determined by the ratio of food available to food required.

The total food available in a cell at $t = t + 1$ is a function of two things:

- The food present at t
- The percentage growth of the food

The total food available is governed by the logistic growth equation, provided as one of the update rules of the system: $n_t(1 + g_f(1 - \frac{n_t}{k_f}))$.

Note, we can disregard the diffusion of food as the diffusion into the cell cancels out the diffusion out of the cell in the state of equilibrium. The total food required to feed the entire bacterium population in the cell is given by the product of consumption rate, c_b, and the bacterium population in the cell, n_b. Dividing the total food available by the total food required, gives us the fraction of bacterium population that survives and reproduces. This value has a lower bound of 0 and upper bound of 1. Subsitituting this value in the bacterium growth equation gives us the population of bacterium at the next time step:

$$n_{b_{t+1}} = \begin{cases} \frac{n_f(1+g_f(1-\frac{n_f}{k_f}))}{c_b \cdot n_b} \cdot b_t \cdot (1 + b_g) & \text{if } 0 \leq \frac{n_f(1+g_f(1-\frac{n_f}{k_f}))}{c_b \cdot n_b} < 1 \\ b_t \cdot (1 + b_g) & \text{if } \frac{n_f(1+g_f(1-\frac{n_f}{k_f}))}{c_b \cdot n_b} \geq 1 \end{cases}$$

AI Use Statement

I used ChatGPT to add inline comments and docstrings to code. I also used it to cross-check if there would be any unexpcted errors in code before running them and if the code produced by me matched the simulation update rules.