Analysis of Epidemic Dynamics

Ali Mokhtari Jazi

July 14, 2024

Note: The questions are in blue color and the answers are in black color. The code, as well as the animations and all dependencies, are available in the following GitHub repository: [https://github.com/AliMokhtariJazi/Epidemic.git]

Part A

Consider the following epidemic scenario: For this particular disease, the uninfected (susceptible S) population can become infected at a rate proportional to the fraction of infected individuals times susceptibles within the population. With this particular disease, recently infected individuals (I) are ten times more contagious during the first year than chronic individuals (C). Infected individuals are treated at a rate of five percent of the infected population per year, and half of them recover with immunity (R) and the other half recover without immunity. Additionally, the mortality rate of infected individuals is twice as high as the recovered or susceptible individuals. This epidemic scenario can be modeled by the following set of differential equations:

$$\frac{dS}{dt} = \Pi + \frac{\tau(C+I)}{2} - (\lambda + \mu)S, \tag{1}$$

$$\frac{dI}{dt} = \lambda S - (\tau + 1 + 2\mu)I, \tag{2}$$

$$\frac{dC}{dt} = I - (\tau + 2\mu)C, \tag{3}$$

$$\frac{dR}{dt} = \frac{\tau(C+I)}{2} - \mu R. \tag{4}$$

where the birthrate Π matches the number of deaths in order to keep a constant population; $\lambda = \beta(10I + C)/N$, with N the total population. For parameters and initial conditions, assume the average life expectancy for uninfected individuals is 50 years, the starting susceptible population is 1000 individuals, there are 10 recent infected individuals and 100 longer-term infected individuals, with no initial recovered individuals.

1. The β parameter simulates the rate of transmission. Solve the system for $\beta = 1$. Will this epidemic become endemic or die out?

Answer: The total population $N(t)$, defined as follows:

$$N(t) = S(t) + I(t) + C(t) + R(t), \tag{5}$$

is conserved if

$$\frac{dN(t)}{dt} = 0. \tag{6}$$

In our problem, the total population is not conserved, which can be demonstrated through the differential

equations governing the dynamics of the population. Summing these equations, we get:

$$\begin{aligned}\frac{dN(t)}{dt} &= \frac{dS(t)}{dt} + \frac{dI(t)}{dt} + \frac{dC(t)}{dt} + \frac{dR(t)}{dt} \\ &= \Pi - \mu(S + 2I + 2C + R), \\ &= \Pi - \mu N - \mu(I + C). \end{aligned} \tag{7}$$

Since the birth rate Π needs to match the number of deaths to keep a constant population, we have $\Pi = \mu N$. This indicates that (7) is nonzero, and therefore the population is not conserved. Hence, we need to consider the total population as a dynamic factor as well.

Figure 1 the evolution of total, susceptible, infectious, chronic, and recovered cases over a period of 70 years, for $\beta = 1$. The results indicate that the epidemic becomes endemic, as the number of infected individuals stabilizes at a nonzero value over time.

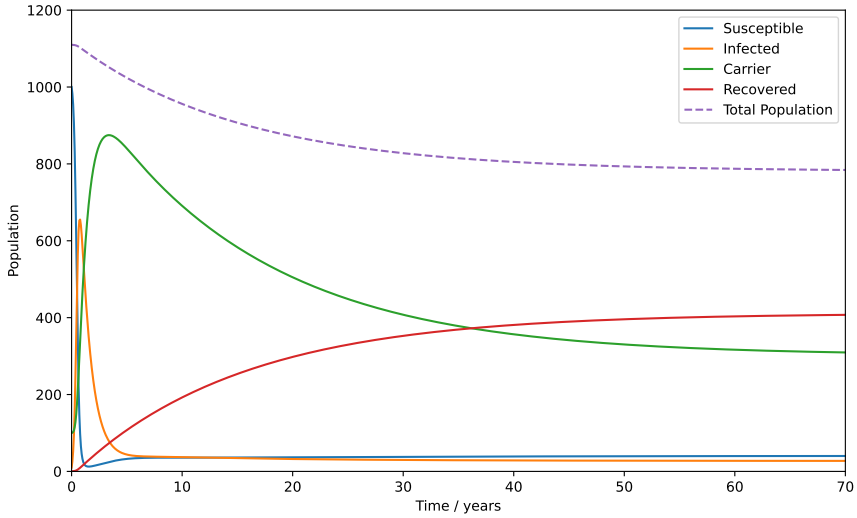

Figure 1: The evolution of the total, susceptible, infectious, chronic, and recovered cases over a period of 70 years for $\beta = 1$. The epidemic becomes endemic as the number of infected individuals stabilizes at a nonzero value over time.

2. Simulate a change in transmission rate. What value of this parameter will result in a long-term prevalence rate of 35%? What about 50%? What is the critical β value at which the epidemic will die out?

Answer: Figure 2 shows the prevalence versus β. It can be seen that the prevalence never reaches 0.5 as it plateaus around 0.42. Also, one can observe that the epidemic dies out only if $\beta \to 0$. Here is a summary of the results:

- A long-term prevalence rate of 35% is achieved with $\beta = 0.24$.
- A long-term prevalence rate of 50% is not achievable with any tested β.
- The critical β value at which the epidemic dies out is $\beta = 0.05$ below which the epidemic dies and above the epidemic doesn't die.

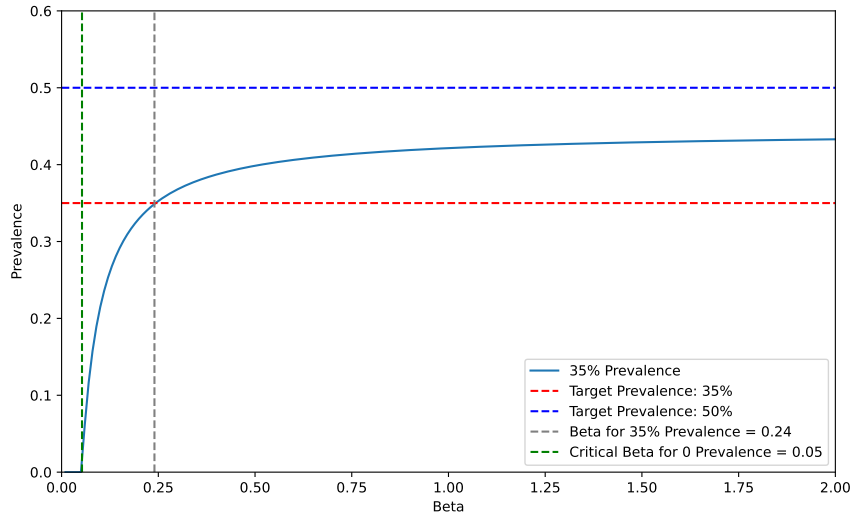

Figure 2: Prevalence vs. β over 70 years. The prevalence stabilizes around 0.42 and never reaches 0.5. The epidemic dies out as β approaches 0.

3. In the scenario that results in a prevalence of 35%, plot the number of cumulative deaths for the first ten years, and the rate of new infections.
Answer: In the scenario where the prevalence stabilizes at 35%, we analyze the cumulative deaths and the rate of new infections over the first ten years. The cumulative deaths represent the total number of individuals who have died due to the epidemic, i.e., $N0 - N(t)$, while the rate of new infections, $\frac{dI}{dt}$ shows how quickly the disease is spreading over time.

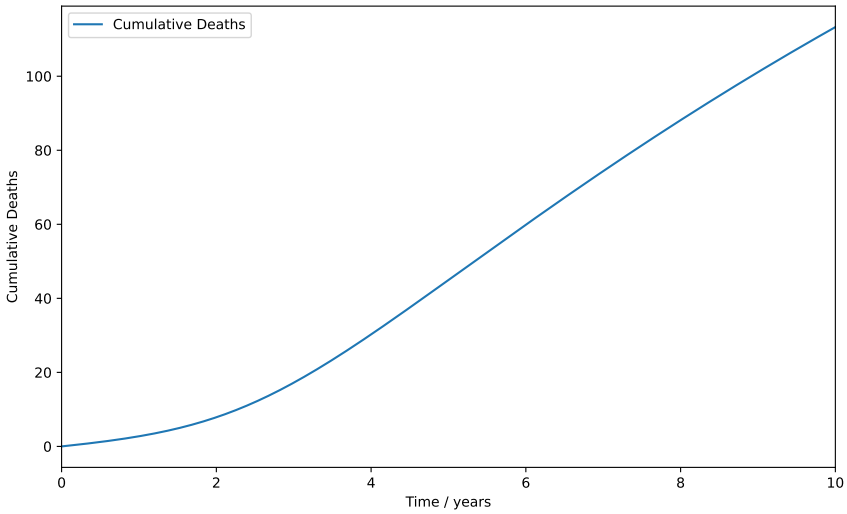

Figure 3: Cumulative deaths over the first ten years with a prevalence rate stabilizing at 35%.

As shown in Figure 3, the cumulative deaths increase over the ten-year period, indicating the ongoing impact of the epidemic on the population.

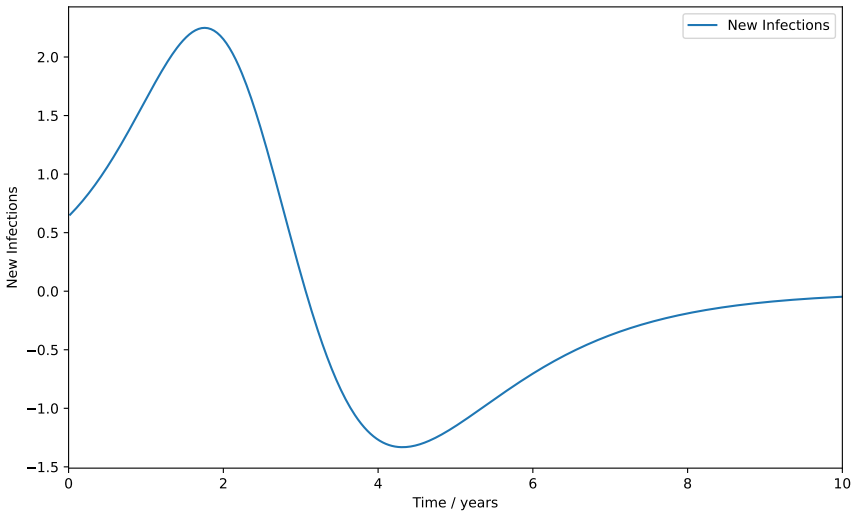

Figure 4: Rate of new infections over the first ten years with a prevalence rate stabilizing at 35%.

Figure 4 shows the rate of new infections over the same period.

4. How would you use eigenvalues to determine whether a specific set of parameters results in an endemic or disease-free scenario?

Answer: While the disease-free equilibrium (DFE) implies no disease spread and appears stable, the eigenvalues of the Jacobian matrix evaluated at the DFE provide critical insights into the local stability of this state. Analyzing the eigenvalues is important to determine whether the DFE is robust to small perturbations.

If all eigenvalues have negative real parts, the DFE is locally stable, meaning small perturbations will decay over time, and the system will return to the DFE, indicating that the disease will die out. Conversely, if any eigenvalue has a positive real part, the DFE is unstable, meaning small perturbations will grow, potentially leading to an outbreak.

Thus, the eigenvalue analysis is essential for understanding the system's behavior near the DFE, predicting the outcome of small infections, and designing effective control measures to ensure that the DFE remains stable and the disease does not resurge.

First, we derive the Jacobian matrix of the system in differential form:

$$J = \begin{bmatrix} \frac{\partial S}{\partial S} & \frac{\partial S}{\partial I} & \frac{\partial S}{\partial C} & \frac{\partial S}{\partial R} \\ \frac{\partial I}{\partial S} & \frac{\partial I}{\partial I} & \frac{\partial I}{\partial C} & \frac{\partial I}{\partial R} \\ \frac{\partial C}{\partial S} & \frac{\partial C}{\partial I} & \frac{\partial C}{\partial C} & \frac{\partial C}{\partial R} \\ \frac{\partial R}{\partial S} & \frac{\partial R}{\partial I} & \frac{\partial R}{\partial C} & \frac{\partial R}{\partial R} \end{bmatrix}$$

By calculating the Jacobian and applying the disease-free equilibrium conditions, i.e., $S = 1000$, $I = 0$, $C = 0$, and $R = 0$, we get:

$$J = \begin{bmatrix} -\mu & \frac{\tau}{2} - 10\beta & \frac{\tau}{2} - \beta & 0 \\ 0 & 10\beta - \frac{\tau}{2} - 2\mu - 1 & \beta & 0 \\ 0 & 1 & -\tau - 2\mu & 0 \\ 0 & \frac{\tau}{2} & \frac{\tau}{2} & -\mu \end{bmatrix}$$

Substituting $\tau = 0.05$ and $\mu = \frac{1}{50}$, we get:

$$J = \begin{bmatrix} -\frac{1}{50} & 0.025 - 10\beta & 0.025 - \beta & 0 \\ 0 & -1.065 + 10\beta & \beta & 0 \\ 0 & 1 & -0.09 & 0 \\ 0 & 0.025 & 0.025 & -\frac{1}{50} \end{bmatrix}$$

The eigenvalues of this matrix are:

$$\lambda_1 = -\frac{1}{50},$$
$$\lambda_2 = -\frac{1}{50},$$
$$\lambda_3 = \frac{1}{2}\left(-1.155 + 10\beta - \sqrt{0.950625 - 15.5\beta + 100\beta^2}\right),$$
$$\lambda_4 = \frac{1}{2}\left(-1.155 + 10\beta + \sqrt{0.950625 - 15.5\beta + 100\beta^2}\right)$$

Among these eigenvalues, all except the last one are negative. The last eigenvalue is negative for $\beta < 0.0504$. Therefore, we can conclude that for $\beta > 0.0504$, the system becomes unstable, and the epidemic persists. This is in agreement with what we obtained for the previous part.

Part B

Generate a random network of 1000 nodes, where the probability that any two nodes are connected is 5% (also known as an Erdos-Renyi network). The nodes can be either susceptible or infected. At each time step, an infected node has a 10% probability of infecting each of its neighbors (independent for each neighbor), and a 5% probability of curing the disease and becoming again susceptible (assume no immunity).

1. Assign a small number of nodes to be infected at t_0 and run a long-term simulation. How does the prevalence (fraction of infected individuals) evolve over time?

Answer: Figure 5 shows how quickly the prevalence number grows to about 95% of the total population. The reason the number stabilizes around 95% is that after a few iterations, almost the entire population gets infected, and about 5% of those recover. The high final prevalence is primarily due to the 5% connection rate of the nodes, which on average makes each node connected to 49 other nodes. With a 10 percent infection rate, at least 5 of the neighboring nodes become infectious, and the 0.05 percent recovery rate without immunity is insufficient to end the epidemic.

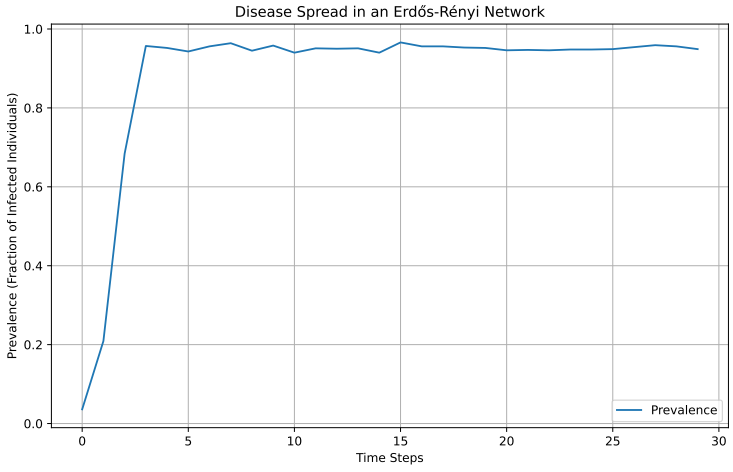

Figure 5: Prevalence of infected individuals over time. Parameters: number of nodes = 1000, edge probability = 0.05, initial infected = 10, infection probability = 0.1, recovery probability = 0.05.

Figure 6 provides a visual representation of the epidemic spread over 6 different time steps, demonstrating the rapid increase in the number of infected individuals. In these snapshots, red nodes represent infectious individuals, blue nodes represent susceptible individuals, and yellow edges indicate the transmission of infection during each iteration. Due to the high density of edges in the 1000-node system, not all edges are clearly visible in the snapshots, but they are more discernible in the animation. Detailed animations showing the spread of the epidemic over a longer period are available in the GitHub repository referenced at the beginning of this document.

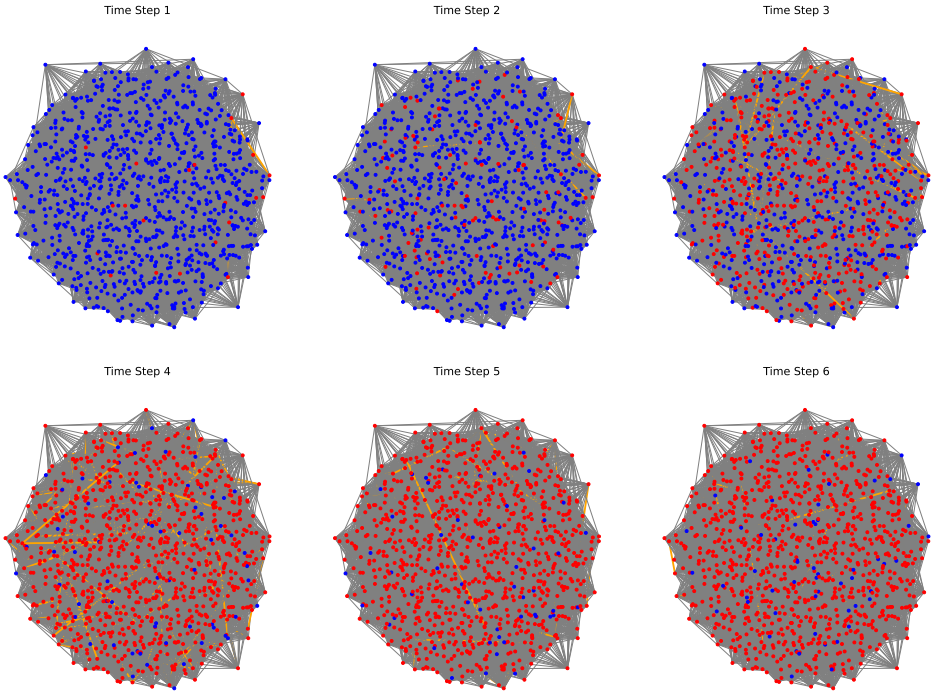

Figure 6: Snapshots of the epidemic simulation over 9 time steps. Parameters: number of nodes = 1000, edge probability = 0.05, initial infected = 10, infection probability = 0.1, recovery probability = 0.05.

2. Is it possible to change the treatment rate so that the prevalence becomes 10%?

Answer: No, it never gets to 10 percent. Even if we set the recovery rate to 1, the prevalence fluctuates around 50%. Figure 7 shows the prevalence with respect to the treatment rate. The high prevalence, despite a perfect recovery rate, indicates the strong impact of network connections and infection probability on the epidemic's persistence.

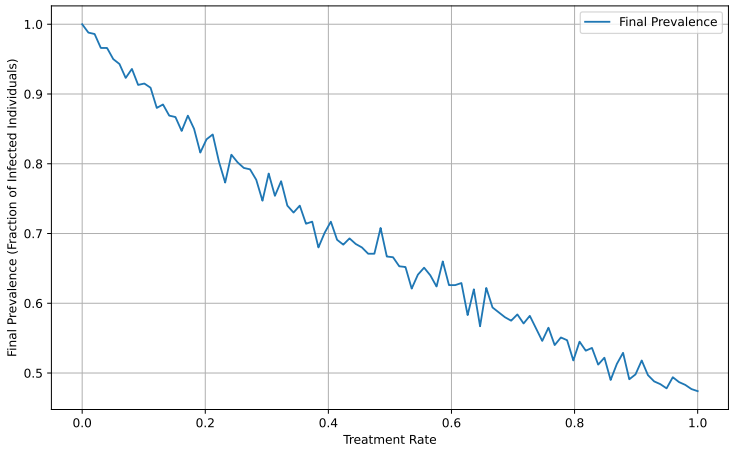

Figure 7: Prevalence of infected individuals with respect to the treatment rate. Parameters: number of nodes = 1000, edge probability = 0.05, initial infected = 5, infection probability = 0.1, number of time steps = 30.

Figure 8 presents a visual representation of the given problem for 6 different time steps, demonstrating the prevalence when the treatment rate is 1. Animations showing the spread of the epidemic over a longer period are available in the GitHub repository referenced at the beginning of this document.

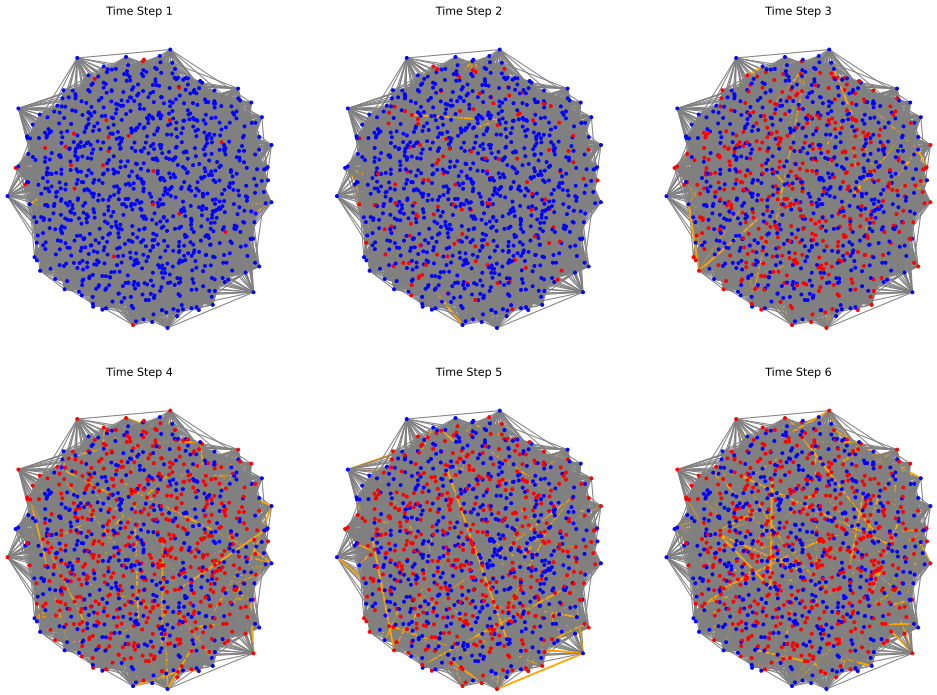

Figure 8: Snapshots of the infection simulation at 6 different time steps with a treatment rate of 1. Parameters: number of nodes = 1000, edge probability = 0.05, initial infected = 5, infection probability = 0.1, number of time steps = 30.

3. What is the critical prevalence that will eliminate the epidemic?

Answer: The critical prevalence should be zero for the epidemic to be eliminated. Even starting with one initially infected case, the epidemic cannot be sustained. This is due to the initial choices of the edge probability and infection probability. We consider two scenarios: in the first, we keep the recovery rate constant and vary the edge probability and infection probability; in the second, we keep the infection probability constant and vary the recovery probability and edge probability. The results of each scenario are demonstrated as 3D plots in Figs. 9 and 10, respectively. It can be observed that there is a phase transition from zero prevalence to finite prevalence for particular choices of the initial parameters.

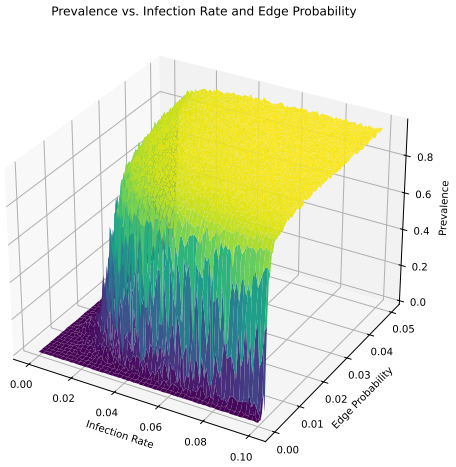

Figure 9: Prevalence vs. Infection Rate and Edge Probability. Parameters: number of nodes = 1000, recovery rate = 0.05, initial infected = 5, number of time steps = 30.

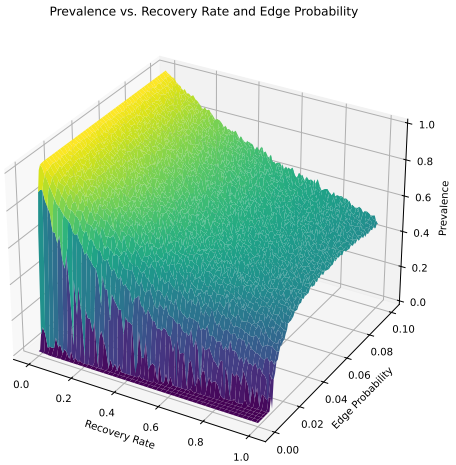

Figure 10: Prevalence vs. Recovery Rate and Edge Probability. Parameters: number of nodes = 1000, infection rate = 0.10, initial infected = 5, number of time steps = 30.

A Python Code for part A

```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

# Parameters
tau = 0.05  # Treatment rate
mu = 1 / 50  # Natural death rate
N0 = 1110  # Initial total population
Pi = mu * N0  # Birthrate matches the initial number of deaths to keep constant initial population

# Initial conditions
S0 = 1000
I0 = 10
C0 = 100
R0 = 0
y0 = [S0, I0, C0, R0]

# Differential equations for the original model
def deriv(y, t, beta, tau, mu, Pi):
    S, I, C, R = y
    N = S + I + C + R  # Total population
    lambda_ = beta * (10 * I + C) / N
    dSdt = Pi + tau * (C + I) / 2 - (lambda_ + mu) * S
    dIdt = lambda_ * S - (tau + 1 + 2 * mu) * I
    dCdt = I - (tau + 2 * mu) * C
    dRdt = tau * (C + I) / 2 - mu * R
    return [dSdt, dIdt, dCdt, dRdt]

# Function for Question 1: Solve the system for beta = 1
def solve_system_for_beta_1():
    beta = 1
    t = np.linspace(0, 2000, 100000)
    solution = odeint(deriv, y0, t, args=(beta, tau, mu, Pi))
    S, I, C, R = solution.T
    N = S + I + C + R
    plt.figure(figsize=(10,6))
    plt.plot(t, S, label='Susceptible')
    plt.plot(t, I, label='Infected')
    plt.plot(t, C, label='Carrier')
    plt.plot(t, R, label='Recovered')
    plt.plot(t, N, label='Total Population', linestyle='--')
    plt.xlabel('Time / years')
    plt.ylabel('Population')
    plt.legend()
    plt.xlim(0, 70)
    plt.ylim(0, 1200)
    plt.savefig('epidemic_model.pdf')
    plt.show()

    final_prevalence = (I[-1] + C[-1]) / N[-1]
    if final_prevalence > 0:
        print(f"The epidemic becomes endemic with a final prevalence of {final_prevalence*100:.2f}%.")
    else:
        print("The epidemic dies out.")

# Function to Find Beta for a Target Prevalence
def find_beta(target_prevalence):
    t = np.linspace(0, 2000, 100000)
    betas = np.linspace(0.01, 2, 200)
    prevalence = []
    for beta in betas:
        solution = odeint(deriv, y0, t, args=(beta, tau, mu, Pi))
        S, I, C, R = solution.T
```

```
        N = S + I + C + R
        final_prevalence = (I[-1] + C[-1]) / N[-1]
        prevalence.append(final_prevalence)
    closest_beta_index = np.argmin(np.abs(np.array(prevalence) - target_prevalence))
    closest_beta = betas[closest_beta_index]
    closest_prevalence = prevalence[closest_beta_index]
    if np.abs(closest_prevalence - target_prevalence) > 0.01:  # Allow a small tolerance
        return None, prevalence  # Indicate no suitable beta found
    return closest_beta, prevalence

# Function for Question 2: Simulate a change in transmission rate
def simulate_change_in_transmission_rate():
    beta_35, prevalence_35 = find_beta(0.35)
    beta_50, prevalence_50 = find_beta(0.5)

    if beta_35 is not None:
        print(f"Beta for 35% prevalence: {beta_35}")
    else:
        print("No suitable beta found for 35% prevalence within the tested range.")

    if beta_50 is not None:
        print(f"Beta for 50% prevalence: {beta_50}")
    else:
        print("No suitable beta found for 50% prevalence within the tested range.")

    def critical_beta_for_zero_prevalence():
        t = np.linspace(0, 2000, 100000)
        betas = np.linspace(0.0, 0.5, 200)

        for beta in betas:
            solution = odeint(deriv, y0, t, args=(beta, tau, mu, Pi))
            S, I, C, R = solution.T
            N = S + I + C + R
            final_prevalence = (I[-1] + C[-1]) / N[-1]
            if final_prevalence > 0.0001:
                break
        return beta
    critical_beta = critical_beta_for_zero_prevalence()
    print(f"The epidemic dies out with a beta of {critical_beta:.2f}.")
    plt.figure(figsize=(10,6))
    betas = np.linspace(0.01, 2, 200)
    plt.plot(betas, prevalence_35, label='35% Prevalence')
    plt.axhline(y=0.35, color='r', linestyle='--', label='Target Prevalence: 35%')
    plt.axhline(y=0.5, color='b', linestyle='--', label='Target Prevalence: 50%')
    plt.axvline(x=beta_35, color='gray', linestyle='--', label=f'Beta for 35% Prevalence = {
        beta_35:.2f}')
    plt.axvline(x=critical_beta, color='green', linestyle='--', label=f'Critical Beta for 0
        Prevalence = {critical_beta:.2f}')
    plt.xlabel('Beta')
    plt.ylabel('Prevalence')
    plt.xlim(0, 2)
    plt.ylim(0, 0.6)
    plt.legend(loc='lower right')
    plt.savefig('prevalence_vs_beta.pdf')
    plt.show()

# Function for Question 3: Plot cumulative deaths and rate of new infections
def plot_cumulative_deaths_and_new_infections():
    beta_35, _ = find_beta(0.35)
    if beta_35 is None:
        print("No suitable beta found for 35% prevalence within the tested range.")
        return
    t = np.linspace(0, 2000, 100000)
    solution = odeint(deriv, y0, t, args=(beta_35, tau, mu, Pi))
    S, I, C, R = solution.T
    N = S + I + C + R

    cumulative_deaths = N0 - N
```

```
    new_infections = np.diff(I)

    plt.figure(figsize=(10,6))
    plt.plot(t, cumulative_deaths, label='Cumulative Deaths')
    plt.xlabel('Time / years')
    plt.ylabel('Cumulative Deaths')
    plt.legend()
    plt.xlim(0, 10)
    plt.ylim(0, 140)
    plt.savefig('cumulative_deaths.pdf')
    plt.show()

    plt.figure(figsize=(10,6))
    plt.plot(t[1:], new_infections, label='New Infections')
    plt.xlabel('Time / years')
    plt.ylabel('New Infections')
    plt.legend()
    plt.xlim(0, 10)
    plt.savefig('new_infections.pdf')
    plt.show()

# Function for Question 4: Use eigenvalues to determine endemic or disease-free scenario
def determine_endemic_or_disease_free():
    beta = 0.07
    # Jacobian matrix at DFE
    J = np.array([
        [-mu, tau/2 - 10 * beta, tau/2 - beta, 0],
        [0, 10 * beta - tau/2 - 2 * mu - 1, beta, 0],
        [0, 1, -tau - 2 * mu, 0],
        [0, tau/2, tau/2, -mu]
    ])


    eigenvalues = np.linalg.eigvals(J)
    print(f"Eigenvalues of the Jacobian at DFE: {eigenvalues}")

    if np.all(np.real(eigenvalues) < 0):
        print(f"The disease-free equilibrium for beta = {beta}, is stable (disease-free
            scenario).")
    else:
        print(f"The disease-free equilibrium for beta = {beta}, is unstable (endemic
            scenario).")

# Main function to call all the parts
def main():
    solve_system_for_beta_1()
    simulate_change_in_transmission_rate()
    plot_cumulative_deaths_and_new_infections()
    determine_endemic_or_disease_free()

if __name__ == "__main__":
    main()
```

B Python code for part B

```
    import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation

# Parameters
num_nodes = 1000
edge_prob = 0.05
```

```
initial_infected = 5
infection_prob = 0.1
recovery_prob = 0.05

num_steps = 30

# Generate Erdos-Renyi network
G = nx.erdos_renyi_graph(num_nodes, edge_prob)

# Initialize states: 0 for susceptible, 1 for infected
states = np.zeros(num_nodes, dtype=int)
initial_infected_nodes = np.random.choice(num_nodes, initial_infected, replace=False)
states[initial_infected_nodes] = 1

def simulate_infection(G, states, infection_prob, recovery_prob, num_steps):
    """
    Simulates the infection spread in the network.

    Parameters:
    - G: NetworkX graph
    - states: Initial states of the nodes
    - infection_prob: Probability of infection spreading
    - recovery_prob: Probability of recovery
    - num_steps: Number of time steps for the simulation

    Returns:
    - history: List of states at each time step
    - edge_history: List of edges that transmitted the infection at each time step
    - prevalence: List of prevalence values at each time step
    """
    history = []
    edge_history = []
    prevalence = []

    for _ in range(num_steps):
        new_states = states.copy()
        edges_infected = []

        for node in range(len(states)):
            if states[node] == 1:  # Infected node
                for neighbor in G.neighbors(node):
                    if states[neighbor] == 0 and np.random.rand() < infection_prob:
                        new_states[neighbor] = 1
                        edges_infected.append((node, neighbor))
                if np.random.rand() < recovery_prob:
                    new_states[node] = 0  # Node recovers and becomes susceptible again

        states = new_states
        history.append(states.copy())
        edge_history.append(edges_infected)
        prevalence.append(np.mean(states))

    return history, edge_history, prevalence

def plot_prevalence():
    """
    Plots the prevalence of infected individuals over time.
    """
    _, _, prevalence = simulate_infection(G, states, infection_prob, recovery_prob,
        num_steps)

    plt.figure(figsize=(10, 6))
    plt.plot(prevalence, label='Prevalence')
    plt.xlabel('Time Steps')
    plt.ylabel('Prevalence (Fraction of Infected Individuals)')
    plt.title('Disease Spread in an Erdos-Renyi Network')
    plt.legend()
    plt.grid(True)
```

```
76      plt.savefig('infection_simulation_prevalence.pdf')
77      plt.show()
78
79  def create_animation():
80      """
81      Creates and saves an animation of the infection spread.
82      """
83      history, edge_history, prevalence = simulate_infection(G, states, infection_prob,
            recovery_prob, num_steps)
84
85      def update(num, history, edge_history, prevalence, graph, pos, ax1, ax2):
86          ax1.clear()
87          current_states = history[num]
88          edges_infected = edge_history[num]
89
90          colors = ['blue' if state == 0 else 'red' for state in current_states]
91          edge_colors = ['gray' for _ in range(len(graph.edges()))]
92          edge_widths = [0.5 for _ in range(len(graph.edges()))]
93
94          for edge in edges_infected:
95              try:
96                  index = list(graph.edges()).index(edge)
97                  edge_colors[index] = 'orange'
98                  edge_widths[index] = 2
99              except ValueError:
100                 continue
101
102         nx.draw(graph, pos, node_color=colors, edge_color=edge_colors, width=edge_widths,
                with_labels=False, node_size=10, ax=ax1)
103         ax1.set_title(f'Time Step {num + 1}')
104
105         ax2.clear()
106         ax2.plot(prevalence[:num + 1], color='blue')
107         ax2.set_xlim(0, num_steps)
108         ax2.set_ylim(0, 1)
109         ax2.set_xlabel('Time Steps')
110         ax2.set_ylabel('Prevalence')
111         ax2.set_title('Prevalence Over Time')
112         ax2.grid(True)
113
114     pos = nx.spring_layout(G)
115
116     fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))
117     ani = animation.FuncAnimation(fig, update, frames=num_steps, fargs=(history,
            edge_history, prevalence, G, pos, ax1, ax2), interval=1000, repeat=False)
118
119     ani.save(f'infection_simulation_with num_nodes={num_nodes} infection_rate={
            infection_prob} treatment_rate={recovery_prob} initial_infected={initial_infected}.
            mp4', writer='ffmpeg')
120     # plt.show()
121
122 def create_snapshots():
123     """
124     Creates and saves snapshots of the infection spread for the first 6 time steps.
125     """
126     history, edge_history, prevalence = simulate_infection(G, states, infection_prob,
            recovery_prob, num_steps)
127     pos = nx.spring_layout(G)
128     fig, axes = plt.subplots(2, 3, figsize=(15, 11))
129     axes = axes.flatten()
130
131     for i in range(6):
132         ax = axes[i]
133         current_states = history[i]
134         edges_infected = edge_history[i]
135
136         colors = ['blue' if state == 0 else 'red' for state in current_states]
137         edge_colors = ['gray' for _ in range(len(G.edges()))]
```

```
        edge_widths = [0.5 for _ in range(len(G.edges()))]

        for edge in edges_infected:
            try:
                index = list(G.edges()).index(edge)
                edge_colors[index] = 'orange'
                edge_widths[index] = 2
            except ValueError:
                continue

        nx.draw(G, pos, node_color=colors, edge_color=edge_colors, width=edge_widths,
            with_labels=False, node_size=10, ax=ax)
        ax.set_title(f'Time Step {i + 1}')

    plt.tight_layout()
    plt.savefig(f'infection_simulation_snapshots num_nodes={num_nodes} infection_rate={
        infection_prob} treatment_rate={recovery_prob} initial_infected={initial_infected}.
        pdf')
    plt.show()

def plot_prevalence_vs_recovery():
    """
    Plots how the final prevalence changes with the recovery rate from 0 to 1.
    """
    recovery_probs = np.linspace(0, 1, 100)
    final_prevalence = []
    for recovery_prob in recovery_probs:
        states = np.zeros(num_nodes, dtype=int)
        initial_infected_nodes = np.random.choice(num_nodes, initial_infected, replace=False
            )
        states[initial_infected_nodes] = 1
        _, _, prevalence = simulate_infection(G, states, infection_prob, recovery_prob,
            num_steps)
        final_prevalence.append(prevalence[-1])  # Append the last prevalence value
        print(f"Recovery Rate: {recovery_prob:.2f}", f"Final Prevalence: {prevalence[-1]:.2f
            }")

    plt.figure(figsize=(10, 5))
    plt.plot(recovery_probs, final_prevalence, label='Final Prevalence')
    plt.xlabel('Recovery Rate')
    plt.ylabel('Final Prevalence (Fraction of Infected Individuals)')
    plt.title('Final Prevalence vs. Recovery Rate')
    plt.legend()
    plt.grid(True)
    plt.savefig('prevalence_vs_recovery_rate.pdf')
    plt.show()



def plot_3d_prevalence(dynamic_param='recovery_rate'):
    """
    Creates a 3D plot where z is the prevalence, x is the dynamic parameter (infection rate
        or recovery rate),
    and y is the edge probability. Infection rate ranges between 0 and 0.2, and edge
        probability ranges between 0 and 0.1,
    both with 100 slices.

    Parameters:
    - dynamic_param: The parameter to vary on the x-axis ('infection_rate' or 'recovery_rate
        ')
    """
    # Parameters
    num_nodes = 1000
    initial_infected = 5
    num_steps = 30

    # Function for simulating infection spread
    def simulate_infection(G, states, infection_prob, recovery_prob, num_steps):
```

```
        """
        Simulates the infection spread in the network.

        Parameters:
        - G: NetworkX graph
        - states: Initial states of the nodes
        - infection_prob: Probability of infection spreading
        - recovery_prob: Probability of recovery
        - num_steps: Number of time steps for the simulation

        Returns:
        - final prevalence after num_steps
        """
        prevalence = []
        for _ in range(num_steps):
            new_states = states.copy()
            for node in range(len(states)):
                if states[node] == 1:  # Infected node
                    for neighbor in G.neighbors(node):
                        if states[neighbor] == 0 and np.random.rand() < infection_prob:
                            new_states[neighbor] = 1
                    if np.random.rand() < recovery_prob:
                        new_states[node] = 0  # Node recovers and becomes susceptible again
            states = new_states
            prevalence.append(np.mean(states))
        return prevalence[-1]  # Return final prevalence

    # Grid of dynamic parameter (infection rate or recovery rate) and edge probabilities
    dynamic_param_values = np.linspace(0, 0.2 if dynamic_param == 'infection_rate' else 1,
        100)
    edge_probs = np.linspace(0, 0.1, 100)
    X, Y = np.meshgrid(dynamic_param_values, edge_probs)
    Z = np.zeros_like(X)

    # Calculate prevalence for each combination of dynamic parameter and edge probability
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            dynamic_value = X[i, j]
            edge_prob = Y[i, j]
            G = nx.erdos_renyi_graph(num_nodes, edge_prob)
            states = np.zeros(num_nodes, dtype=int)
            initial_infected_nodes = np.random.choice(num_nodes, initial_infected, replace=
                False)
            states[initial_infected_nodes] = 1
            if dynamic_param == 'infection_rate':
                Z[i, j] = simulate_infection(G, states, dynamic_value, 0.05, num_steps)
            else:
                Z[i, j] = simulate_infection(G, states, 0.1, dynamic_value, num_steps)

    # Plotting the 3D surface
    fig = plt.figure(figsize=(12, 8))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis')

    ax.set_xlabel('Infection Rate' if dynamic_param == 'infection_rate' else 'Recovery Rate'
        )
    ax.set_ylabel('Edge Probability')
    ax.set_zlabel('Prevalence')
    ax.set_title('Prevalence vs. ' + ('Infection Rate and Edge Probability' if dynamic_param
         == 'infection_rate' else 'Recovery Rate and Edge Probability'))
    filename = f'num_nodes={num_nodes} initial_infected={initial_infected} '
    filename += f'{"infection_rate" if dynamic_param == "recovery_rate" else "recovery_rate
        "}='
    filename += f'{recovery_prob if dynamic_param == "infection_rate" else infection_prob}.
        pdf'
    plt.savefig(filename)

    plt.show()
```

```
259
260
261
262 def main():
263     # Uncomment the function you want to run
264     # plot_prevalence()
265     create_animation()
266     # create_snapshots()
267     # plot_prevalence_vs_recovery()
268     # plot_3d_prevalence()
269 if __name__ == "__main__":
270     main()
```