POLITECNICO DI TORINO

NETWORK DYNAMICS AND LEARNING

2020 / 2021

ALBERTO MARIA FALLETTA

S277971

HOMEWORK III

Problem 1

1.1 Epidemic on a known graph

The aim of this first part of the assignment, whose code can be found in Appendix I, is to simulate an epidemic on a symmetric k-regular undirected graph, which is a graph with node set $\mathcal{V} = \{1, \dots, n\}$ where every node is directly connected to the k closest modulo n index nodes.

The disease propagation model for the epidemic is a discrete-time simplified version of the SIR epidemic model for which at any time, nodes are in state $\mathcal{X}_i(t) \in \{S, I, R\}$, where S is susceptible, I is infected and R is recovered and for which two parameters are defined:

- $\beta \in [0, 1]$ probability that a susceptible node is infected by an "I state" neighbor, during a time step, this means that, if a susceptible node i has m infected neighbors, the probability that the individual i does not get infected during a time step is $(1 - \beta)^m$.
- $\varrho \in [0, 1]$ probability that an infected individual will recover during a time step.

The epidemic is therefore driven by the following transition probabilities:

$$\mathbb{P}[X_i(t+1) = I \mid X_i(t) = S, \sum_{j \in \mathcal{V}} W_{ij} \cdot \delta^I_{X_j(t)} = m] = 1 - (1 - \beta)^m$$

$$\mathbb{P}(X_i(t+1) = R \mid X_i(t) = I) = \varrho$$

The epidemic is simulated on a symmetric k-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ over 15 weeks of time, averaging results over 100 simulations, with parameters:

- $|\mathcal{V}| = 500$ nodes
- $k = 4$
- $\beta = 0.3$
- $\rho = 0.7$
- 10 random initial infected nodes

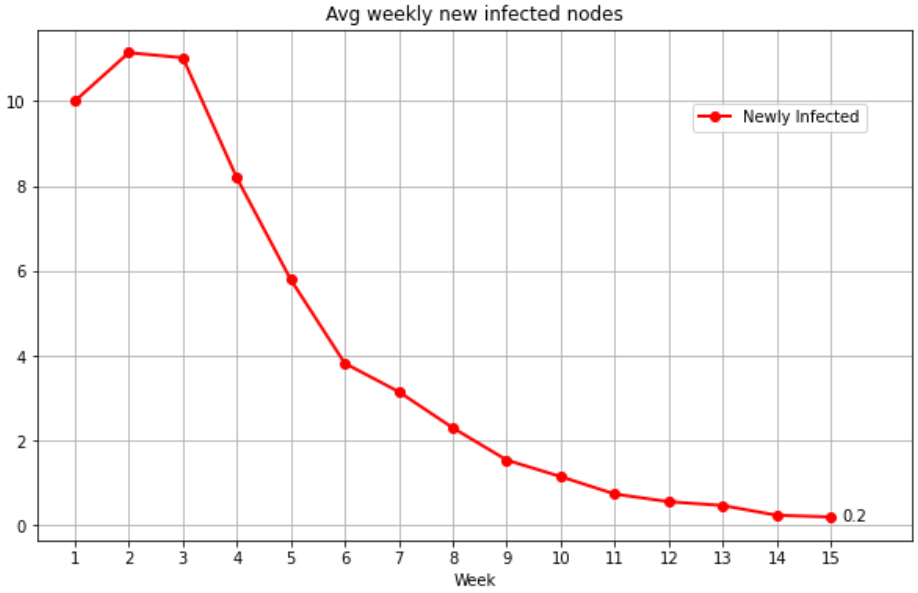


Figure 1: Average number of newly infected individuals each week

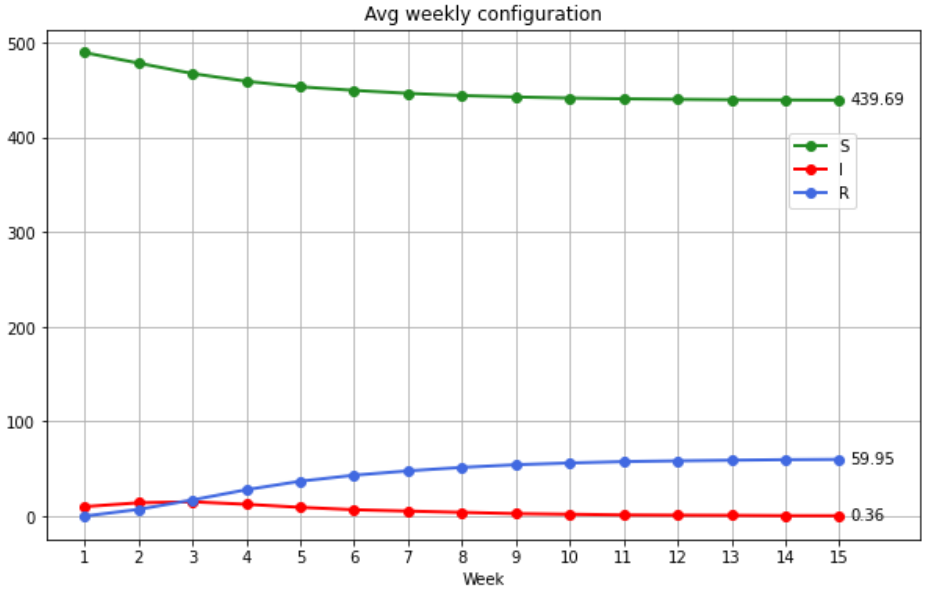


Figure 2: Average total number of susceptible, infected, and recovered individuals at each week

As it is possible to see from the previous two plots showing the average number of newly infected individuals and the average total number of susceptible, infected, and recovered individuals at each week, the epidemic modelled with previous parameters reaches on average 60 nodes before extinguishing.

Notes about the code:

Differently from the implementations of the next exercises, in order to build the k-regular graph for this task, the method starting from the construction of the adjacency matrix W has been preferred. A sparse matrix has been used to store node configuration for each week but more in general the sparse format will be used extensively in the following exercises also to store the adjacency matrix W.

1.2 Random graph generation

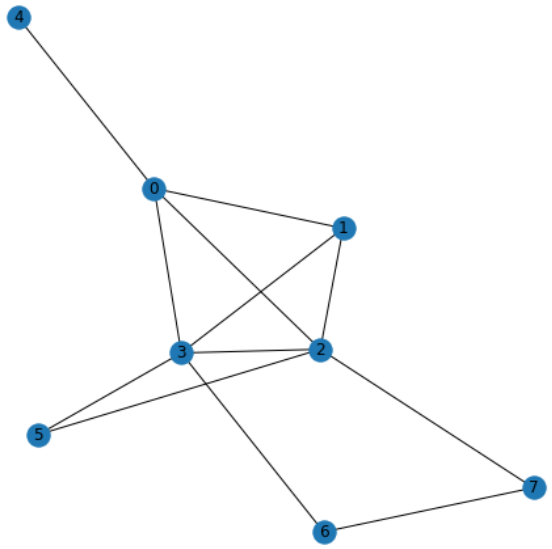
The aim of this part of the assignment, whose code can be retrieved in Appendix II, is to generate a random graph according to the preferential attachment model with average degree close to a parameter k.

To do this, the process starts with an initial complete graph $\mathcal{G}_1$ provided by NetworkX function complete_graph(n_nodes), with $n\_nodes = k + 1$, then, at every time $t \geq 2$ a new graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ is created by adding a new node to $\mathcal{G}_{t-1}$ and connecting it to some of the existing nodes $\mathcal{V}_{t-1}$ of $\mathcal{G}_{t-1}$ chosen proportionally to the degree of the nodes.

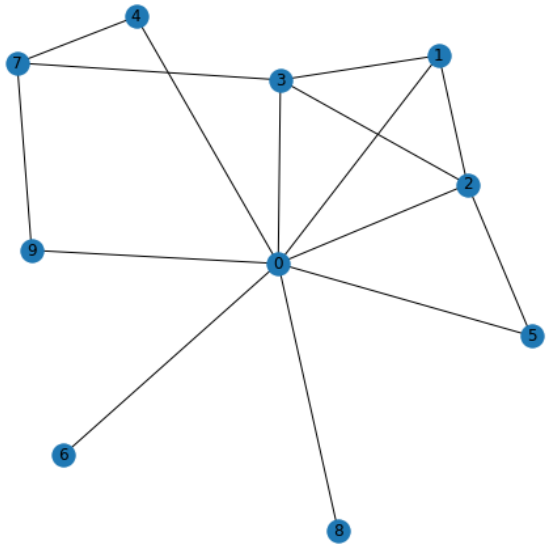
The rule by which a new node add links to the nodes of $\mathcal{G}_{t-1}$ is preferential attachment, that means at every time step $t \geq 2$, every new node will have a degree $c = k/2$. Hence, it should add c undirected links to the existing graph $\mathcal{G}_{t-1}$.

Some care has been taken to avoid multiple links to the same node and this has been performed by generating the sample of neighbors without replacement.

A further trick is been used in case k is an odd number (since in this case $c = k/2$ would not be an integer) to still achieve an average degree of k when many nodes are added, and this is alternating between adding $\lfloor k/2 \rfloor$ and $\lceil k/2 \rceil$ links when adding a new node to the graph.



NUMBER OF NODES: 8
NODES DEGREES: [4, 3, 5, 5, 1, 2, 2, 2]
AVG DEGREE: 3.0



NUMBER OF NODES: 10
NODES DEGREES: [8, 3, 4, 4, 2, 2, 1, 3, 1, 2]
AVG DEGREE: 3.0

The two images above show a possible result of the implemented algorithm when $total\_nodes = 8, k = 3$ and $total\_nodes = 10, k = 3$, as we can see in both cases the average degree is satisfied.

Notes about the code:

Even though only the matrix W is needed in the codes (specifically when looking for infected neighbors of given nodes) the proposed algorithm generates and returns a NetworkX graph in order to later obtain W in sparse format by using NetworkX function $W = \text{nx.adjacency\_matrix(G)}$.

Problem 2

Pandemic without vaccination

In this part, whose code can be found in Appendix III, the function implemented in the previous exercise will be used to build a graph on which to simulate an epidemic with discrete-time SIR propagation model.

The epidemic is simulated on a symmetric k-regular graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ over 15 weeks of time, averaging results over 100 simulations, with parameters:

- $|\mathcal{V}| = 500$ nodes
- $k = 6$
- $\beta = 0.3$
- $\rho = 0.7$
- 10 random initial infected nodes

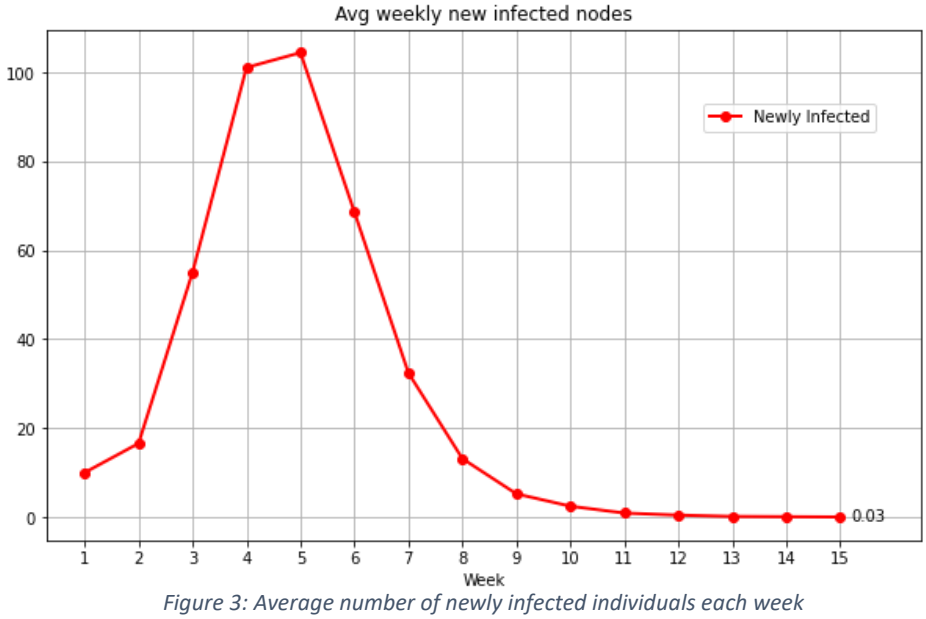


Figure 3: Average number of newly infected individuals each week

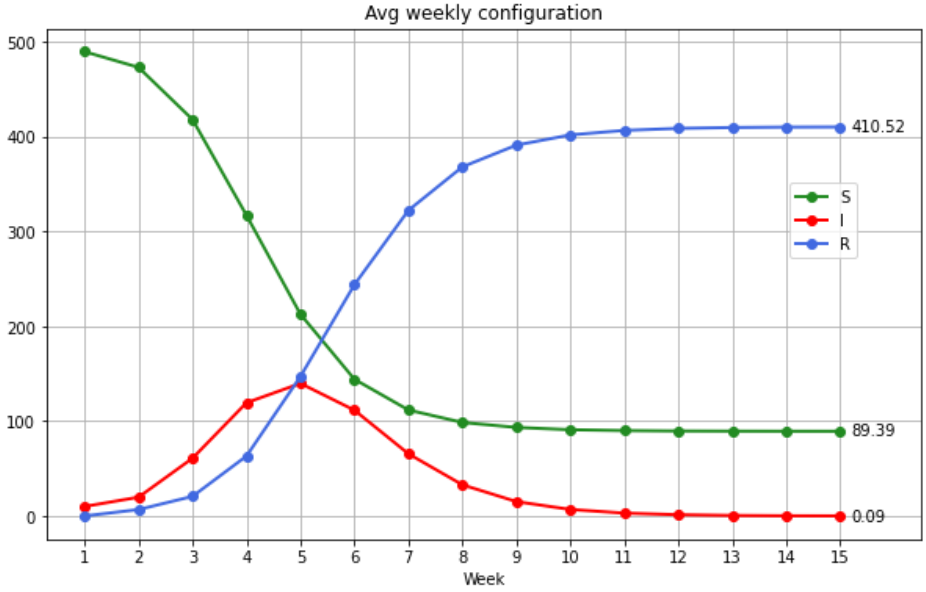


Figure 4: Average total number of susceptible, infected, and recovered individuals at each week

As it is possible to see from the previous two plots showing the average number of newly infected individuals and the average total number of susceptible, infected, and recovered individuals at each week, the configuration and evolution of the epidemic is completely different from what experienced in the first task of the assignment. While in fact, the newly infected individual graph peaked at around 11 in the first case, now we can see that only building the graph with preferential attachment model and increasing k to 6 lead to a very different scenario with the newly infected graph peaking above the value of 100 and the overall epidemic interesting as many as 410 over 500 nodes on average against the 60 over 500 of the first case.

Notes about the code:

The code implemented in this exercise is very similar to the one proposed for the first task of the assignment with the only difference being that, since the graph is generated by the previous exercise's algorithm, the adjacency matrix W is obtained by using NetworkX function $W = \mathrm{nx.adjacency\_matrix(G)}$ and is therefore in a sparse format. This is to be taken into account when looking for neighbors of nodes.

Problem 3

Pandemic with vaccination

In this third part of the assignment, whose code can be retrieved in Appendix IV, the simulation is performed taking actions to slow down the epidemic by means of vaccination. During each week, some parts of the population, eligible to receive the vaccine (that is nodes not yet vaccinated), will receive it and will therefore, be impossible to get infected and infect other nodes. Vaccination is assumed to take effect immediately once given, so it can therefore be considered also as a cure for the case of "I state" nodes.

Once again, the disease propagation is simulated over 15 weeks, but this time vaccine distribution is taken into consideration and this is done according to:

$$\mathrm{Vacc(t)} = [0, 5, 15, 25, 35, 45, 55, 60, 60, 60, 60, 60, 60, 60]$$

Where $\mathrm{Vacc(t)}$ represents the total fraction of population that has received vaccination by each week and it should be interpreted as: 55% of the population has received vaccination by week 7, and 5% received vaccination during week 7.

To simulate the actual vaccination, at the beginning of each week, the correct number of individuals to vaccinate according to $\mathrm{Vacc(t)}$ is found. The individuals to vaccinate are then selected uniformly at random from the population that has not yet received vaccination, and this means that an infected individual might receive vaccination as well. The reason is that some people are not able to tell whether they are infected or just have a common cold. If an infected individual becomes vaccinated it is assumed that she will not be able to infect another individual. In other words, we assume that regardless of the state of an individual prior to the vaccination, once vaccinated the individual will not be able to become infected nor infect any other individuals.

The code implemented in this exercise is very similar to the one proposed for the previous one (i.e., preferential attachment random graph, $|\mathcal{V}| = 500$ nodes, $k = 6$, $\beta = 0.3$, $\rho = 0.7$, 10 random initial infected nodes, 100 iterations) with the only difference being that the epidemic has been simulated with vaccination for 15 weeks, using the vaccination scheme Vacc(t) above.

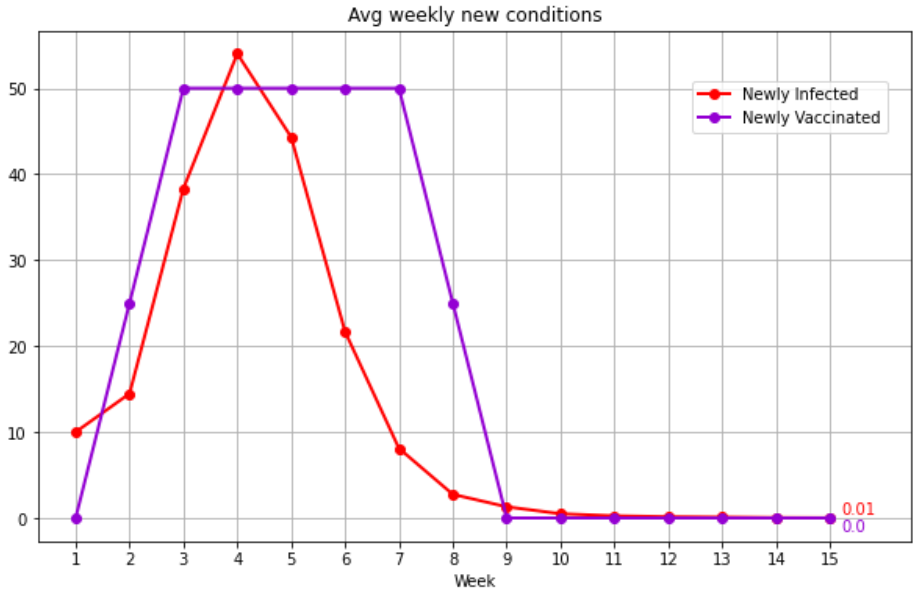


Figure 3: Average number of newly infected and newly vaccinated individuals each week

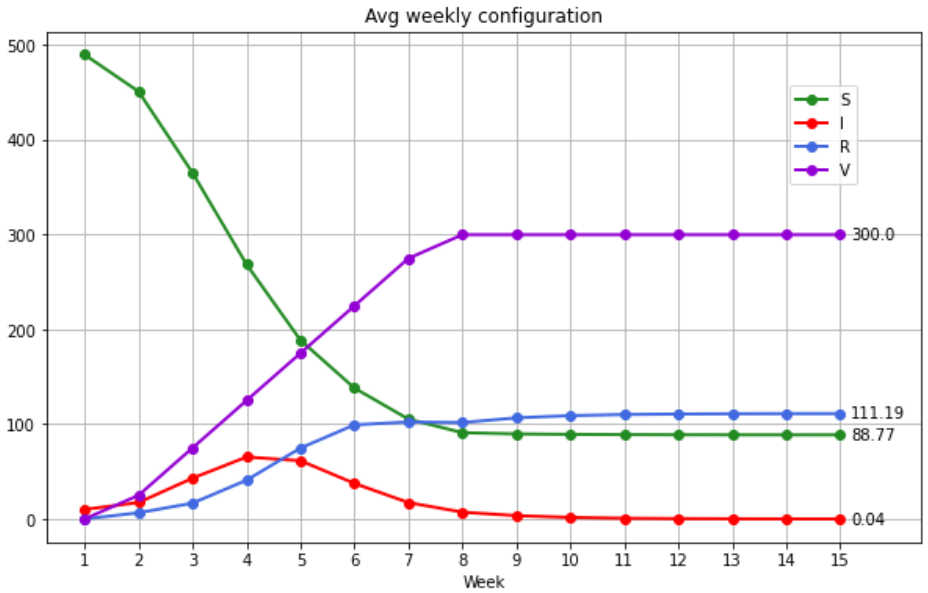


Figure 4: Average total number of susceptible, infected, recovered and vaccinated individuals at each week

As it is possible to see from the previous two plots showing the average number of newly infected individuals and the average total number of susceptible, infected, and recovered individuals at each week, the newly infected individual graph peak slightly above 50 (against the value of 100 in the simulation without vaccination) and the overall epidemic interests around 111 over 500 nodes (against the previous 410 over 500).

Problem 4

The 2009 H1N1 pandemic in Sweden

In this final part, whose code can be found in Appendix V, an estimation of the social structure of the Swedish population and the disease-spread parameters during the H1N1 pandemic in 2009 is performed. During the fall of 2009, in fact, about 1.5 million people out of a total population of 9 million were infected with H1N1 virus, and about 60% of the population received vaccination.

The simulation will reflect the real epidemic between week 42, 2009 and week 5, 2010. During these weeks, the fraction of population that received the vaccine was:

$$\mathrm{Vacc(t)} = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60, 60]$$

In order not to spend too much time running simulations, the population of Sweden will be scaled down by a factor of 10^4. This means that the population during the simulation will be $n = |\mathcal{V}| = 934$. For this scaled version, the number of newly infected individuals each week in the period between week 42, 2009 and week 5, 2010 is:

$$I_0(\mathrm{t}) = [1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0, 0]$$

To estimate the disease-spread parameters and social structure of the Swedish population (i.e., k, β, ρ) a gradient-based algorithm will be implemented, leading to the set of parameters that best matches the real pandemic.

Algorithm:

Starting with some initial values of $k0$, $\beta 0$, $\rho 0$, along with some Δk, $\Delta\beta$, $\Delta\rho$, for each set of parameters (k, β, ρ) in the parameter-space $k \in \{\mathrm{k0} - \Delta\mathrm{k},\ \mathrm{k0},\ \mathrm{k0} + \Delta\mathrm{k}\}$, $\beta \in \{\beta 0 - \Delta\beta,\ \beta 0,\ \beta 0 + \beta\mathrm{k}\}$, and $\varrho \in \{\varrho 0 - \Delta\varrho,\ \varrho 0,\ \varrho 0 + \Delta\varrho\}$, the code has to:

- Generate a random graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ using the preferential attachment model developed in previous sections with average degree k, and $|\mathcal{V}| = 934$.
- Simulate the pandemic for 15 weeks on $\mathcal{G}$ using the vaccination scheme described above, averaging the number of newly infected individuals each week (i.e., $I(t)$) over 10 iterations.
- Compute the root-mean-square error between the result of the simulation $I(t)$ and the one of the real epidemic $I_0(\mathrm{t})$
- Update $k0$, $\beta 0$, $\rho 0$ to the set of parameters yielding the lowest RMSE. If for one iteration there is no RMSE improvement the deltas are divided by two. If two consecutive iterations go by without improvement the algorithm stops.

Notes about the code:

The implemented gradient-based algorithm has been run with 1 initial random infected node from two different starting points ($k0 = 10$, $\beta 0 = 0.3$, $\rho 0 = 0.6$, $\Delta k = 1$, $\Delta\beta = 0.1$, $\Delta\rho = 0.1$ and $k0 = 6$, $\beta 0 = 0.3$, $\rho 0 = 0.6$, $\Delta k = 1$, $\Delta\beta = 0.1$, $\Delta\rho = 0.1$) leading to the minimum RMSE score found with parameters $k0 = 5$, $\beta 0 = 0.4$, $\rho 0 = 0.7$. It is possible to retrieve the execution log in Appendix VI.

The code also implements a list to store all the tried combinations in order not to compute twice the algorithm on a parameters combination already unsuccessfully tried.

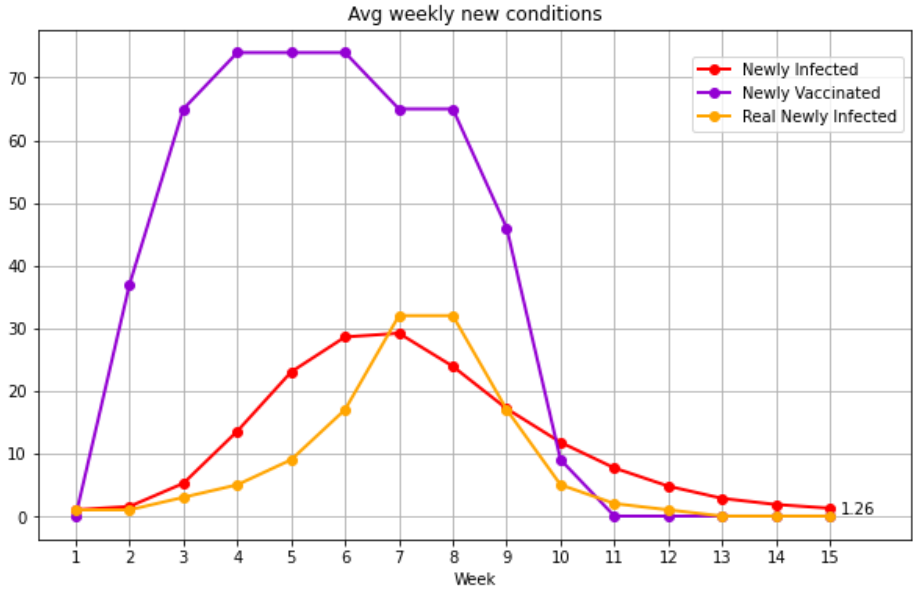


Figure 7: Average number of newly infected individuals each week according to the model (with best parameters) compared to the true value of newly infected individuals each week.

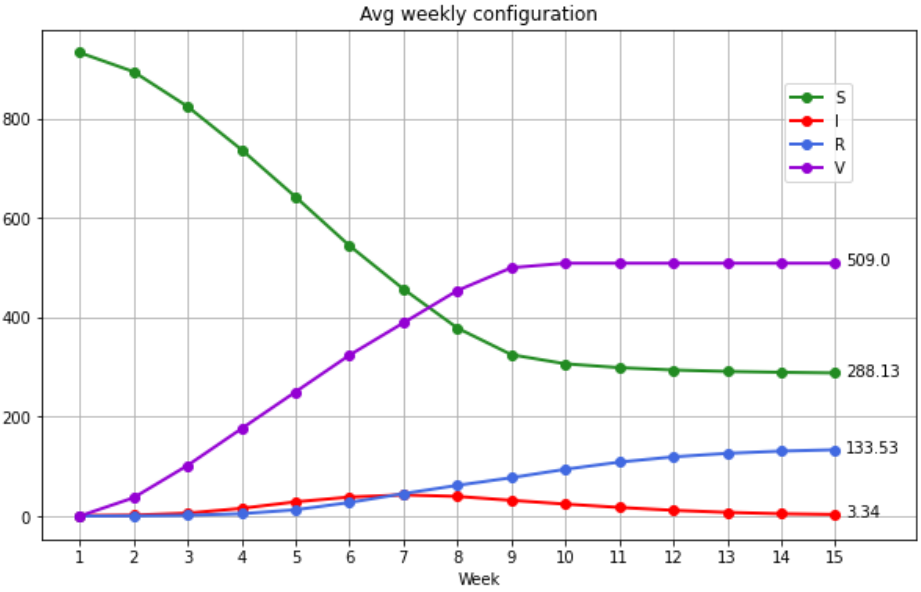


Figure 8: Total number of susceptible, infected, recovered and vaccinated individuals at each week according to the model.

As it is possible to see from the previous two plots showing the average number of newly infected individuals from the simulation against the actual number of newly infected individuals, and the average total number of susceptible, infected, and recovered individuals at each week, the simulation result is rather close to the actual Swedish epidemic evolution with the only difference being the first more spread over time.

Problem 5

Challenge

The estimation of the social structure of the Swedish population and the disease-spread parameters during the H1N1 pandemic in 2009 has been also tried on a different random graph generated according to Erdos-Renyi model, consisting in a graph with number of nodes equal $|\mathcal{V}| = 934$ connected by an undirected edge one to another with probability p, chosen so to have expected degree equal to k. The code can be found in Appendix VII.

Once again the previous gradient-based algorithm has been run with 1 initial random infected node from two different starting points ($k0 = 10$, $\beta 0 = 0.3$, $\rho 0 = 0.6$, $\Delta k = 1$, $\Delta\beta = 0.1$, $\Delta\rho = 0.1$ and $k0 = 6$, $\beta 0 = 0.3$, $\rho 0 = 0.6$, $\Delta k = 1$, $\Delta\beta = 0.1$, $\Delta\rho = 0.1$) leading to the minimum RMSE score found with parameters $k0 = 6$, $\beta 0 = 0.2$, $\rho 0 = 0.6$. It is possible to retrieve the execution log in Appendix VIII.

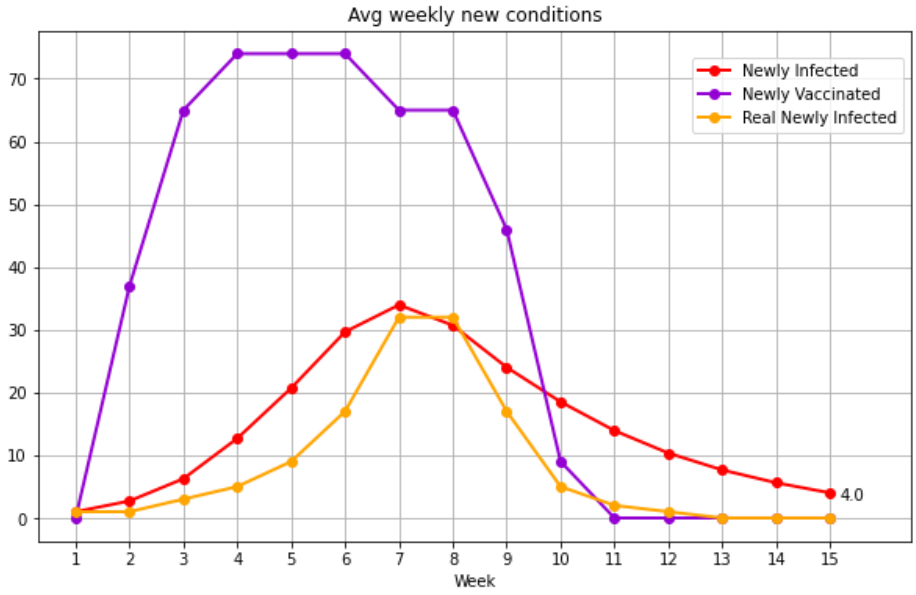


Figure 9: Average number of newly infected individuals each week according to the model (with best parameters) compared to the true value of newly infected individuals each week.

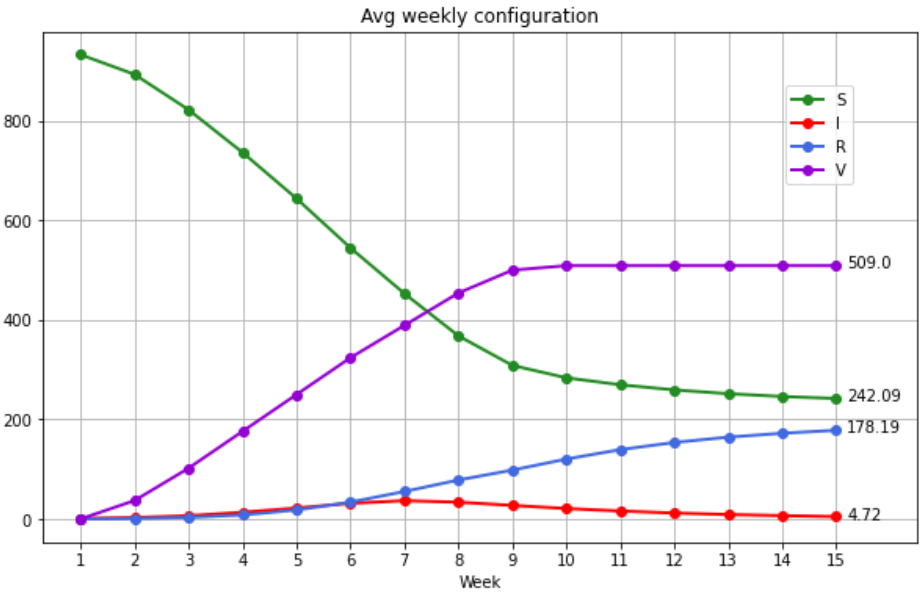


Figure 10: Total number of susceptible, infected, recovered and vaccinated individuals at each week.

Appendix

I)

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from sklearn.model_selection import ParameterGrid

# PARAMETERS
n = 500
k = 4
beta = 0.3  # Infection probability
rho = 0.7  # Recovery probability
n_weeks = 15
n_initial_infected = 10
n_iter = 100

new_infected_over_time = np.zeros(n_weeks, dtype=int)
susceptible_over_time = np.zeros(n_weeks, dtype=int)
infected_over_time = np.zeros(n_weeks, dtype=int)
recovered_over_time = np.zeros(n_weeks, dtype=int)

# GRAPH CONSTRUCTION
neighs = [int(i-(k/2)) for i in range(0, k+1) if i!=k/2]
W = np.zeros((n, n), dtype=int)

for neigh in neighs:
  W = W + np.diag(np.ones((n-abs(neigh)), dtype=int), neigh) +
np.diag(np.ones((abs(neigh)), dtype=int), int(n*(abs(neigh)/neigh))-neigh)

if n < 32:
  G = nx.from_numpy_matrix(W)
  plt.figure(1,figsize=(n,n))
  nx.draw_spectral(G, with_labels=True)

# SIMULATION
for iter in range(0,n_iter):

  # Selecting n_initial_infected
  x0 = np.zeros(n, dtype=int)  # initial configuration
  initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
  for infected in initial_infected:
    x0[infected] = 1
  states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

  susceptible_over_time[0] += n - n_initial_infected
  infected_over_time[0] += n_initial_infected
  recovered_over_time[0] += 0
  new_infected_over_time[0] += n_initial_infected

  for week in range(1,n_weeks):
    last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
    current_week_config = np.zeros(n, dtype=int)
    new_infected_cnt = 0

    for node in range(0, n):

      if last_week_config[node] == 0:  # count infected neighbors, compute infection prob
        m = np.count_nonzero(np.multiply(W[node], last_week_config) == 1.0)
```

```
        current_week_config[node] = np.random.choice([0, 1], p=[(1-beta)**m, 1-((1-
beta)**m)])
        if current_week_config[node] == 1:
          new_infected_cnt += 1

      elif last_week_config[node] == 1:  # compute recovery probability
        current_week_config[node] = np.random.choice([1, 2], p=[1-rho, rho])

      else:  # stays recovered
        current_week_config[node] = 2

    states = sp.sparse.vstack([states,current_week_config])

    # Counts for visualizations
    susceptible_over_time[week] += np.count_nonzero(current_week_config == 0)
    infected_over_time[week] += np.count_nonzero(current_week_config == 1)
    recovered_over_time[week] += np.count_nonzero(current_week_config == 2)
    new_infected_over_time[week] += new_infected_cnt

avg_susceptible_over_time = susceptible_over_time / n_iter
avg_infected_over_time = infected_over_time / n_iter
avg_recovered_over_time = recovered_over_time / n_iter
avg_new_infected_over_time = new_infected_over_time / n_iter

time = [i for i in range(1,n_weeks+1)]
fig_1, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_new_infected_over_time, c='red', marker='o', linestyle='-',
linewidth=2, label='Newly Infected')
ax.legend(loc=(0.75, 0.8))
ax.set_xlabel('Week')
ax.set_title('Avg weekly new infected nodes')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_new_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-0.1, str(y), color='k')
plt.show()

fig_2, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_susceptible_over_time, c='forestgreen', marker='o', linestyle='-',
linewidth=2, label='S')
ax.plot(time, avg_infected_over_time, c='red', marker='o', linestyle='-', linewidth=2,
label='I')
ax.plot(time, avg_recovered_over_time, c='royalblue', marker='o', linestyle='-',
linewidth=2, label='R')
ax.legend(loc=(0.85, 0.65))
ax.set_xlabel('Week')
ax.set_title('Avg weekly configuration')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_susceptible_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_recovered_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
plt.show()

if n < 32:
```

```
  # Plot the infection spread
  pos = nx.spectral_layout(G)
  fig = plt.figure(figsize=(20,20))
  for t in range(0,n_weeks):
      plt.subplot(3,5,t+1)
      x = np.array(states.tocsr()[t].toarray()[0], dtype=int)
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==0).T[0].tolist(),
          node_color = 'forestgreen')
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==1).T[0].tolist(),
          node_color = 'red')
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==2).T[0].tolist(),
          node_color = 'royalblue')
      plt.title('Week = {0}'.format(t+1))
```

II)

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from sklearn.model_selection import ParameterGrid


def Generate_PA_Graph(total_nodes, avg_degree, a):

  GPA = nx.complete_graph(avg_degree + 1)  # starting complete graph
  residual = 0

  while len(GPA) < total_nodes:
    node = len(GPA)

    if avg_degree % 2 == 0:
      node_degree = avg_degree // 2
    else:
      node_degree = avg_degree // 2 + residual
       # to alternate adding one more and one less if k odd
      residual = (avg_degree % 2 + residual) % 2

    deg_PA = np.array([d for n, d in GPA.degree()]) + a
    prob = deg_PA/sum(deg_PA)  # normalization to obtain a probability distribution
    neighs = np.random.choice(np.arange(len(GPA)), p=prob, size=node_degree,
replace=False)

    GPA.add_node(node)  # Node insertion
    for neigh in neighs:
      GPA.add_edge(node,neigh)  # Edges creation

  return GPA

G = Generate_PA_Graph(8, 3, 0)
W = nx.adjacency_matrix(G)
# print(W.todense())
plt.figure(1,figsize=(6,6))
nx.draw(G, with_labels=True)
degrees = [d for n, d in G.degree()]
```

```
print("NUMBER OF NODES:", len(G))
print("NODES DEGREES:", degrees)
print("AVG DEGREE:", sum(degrees)/len(degrees))
```

III)

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from sklearn.model_selection import ParameterGrid

# PARAMETERS
n = 500
k = 6
a = 0
beta = 0.3  # Infection probability
rho = 0.7  # Recovery probability
n_weeks = 15
n_initial_infected = 10
n_iter = 100

new_infected_over_time = np.zeros(n_weeks, dtype=int)
susceptible_over_time = np.zeros(n_weeks, dtype=int)
infected_over_time = np.zeros(n_weeks, dtype=int)
recovered_over_time = np.zeros(n_weeks, dtype=int)

# GRAPH CONSTRUCTION
G = Generate_PA_Graph(n, k, a)
W = nx.adjacency_matrix(G)

if n < 32:
  plt.figure(1,figsize=(6,6))
  nx.draw(G, with_labels=True)

# SIMULATION
for iter in range(0,n_iter):

  # Selecting n_initial_infected
  x0 = np.zeros(n, dtype=int)  # initial configuration
  initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
  for infected in initial_infected:
    x0[infected] = 1
  states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

  susceptible_over_time[0] += n - n_initial_infected
  infected_over_time[0] += n_initial_infected
  recovered_over_time[0] += 0
  new_infected_over_time[0] += n_initial_infected

  for week in range(1,n_weeks):
    last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
    current_week_config = np.zeros(n, dtype=int)
    new_infected_cnt = 0

    for node in range(0, n):

      if last_week_config[node] == 0:  # count infected neighbours and compute infection
probability
```

```
        m = np.count_nonzero(np.multiply(W.tocsr()[node].toarray()[0], last_week_config)
== 1.0)
        current_week_config[node] = np.random.choice([0, 1], p=[(1-beta)**m, 1-((1-
beta)**m)])
        if current_week_config[node] == 1:
          new_infected_cnt += 1

      elif last_week_config[node] == 1:  # compute recovery probability
        current_week_config[node] = np.random.choice([1, 2], p=[1-rho, rho])

      else:  # stays recovered
        current_week_config[node] = 2

    states = sp.sparse.vstack([states,current_week_config])

    # Counts for visualizations
    susceptible_over_time[week] += np.count_nonzero(current_week_config == 0)
    infected_over_time[week] += np.count_nonzero(current_week_config == 1)
    recovered_over_time[week] += np.count_nonzero(current_week_config == 2)
    new_infected_over_time[week] += new_infected_cnt

avg_susceptible_over_time = susceptible_over_time / n_iter
avg_infected_over_time = infected_over_time / n_iter
avg_recovered_over_time = recovered_over_time / n_iter
avg_new_infected_over_time = new_infected_over_time / n_iter

time = [i for i in range(1,n_weeks+1)]
fig_1, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_new_infected_over_time, c='red', marker='o', linestyle='-',
linewidth=2, label='Newly Infected')
ax.legend(loc=(0.75, 0.8))
ax.set_xlabel('Week')
ax.set_title('Avg weekly new infected nodes')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_new_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-1, str(y), color='k')
plt.show()

fig_2, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_susceptible_over_time, c='forestgreen', marker='o', linestyle='-',
linewidth=2, label='S')
ax.plot(time, avg_infected_over_time, c='red', marker='o', linestyle='-', linewidth=2,
label='I')
ax.plot(time, avg_recovered_over_time, c='royalblue', marker='o', linestyle='-',
linewidth=2, label='R')
ax.legend(loc=(0.85, 0.55))
ax.set_xlabel('Week')
ax.set_title('Avg weekly configuration')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_susceptible_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_recovered_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
plt.show()
```

```
if n < 32:
  # Plot the infection spread
  pos = nx.spectral_layout(G)
  fig = plt.figure(figsize=(20,20))
  for t in range(0,n_weeks):
      plt.subplot(3,5,t+1)
      x = np.array(states.tocsr()[t].toarray()[0], dtype=int)
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==0).T[0].tolist(),
          node_color = 'forestgreen')
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==1).T[0].tolist(),
          node_color = 'red')
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==2).T[0].tolist(),
          node_color = 'royalblue')
      plt.title('Week = {0}'.format(t+1))
```

IV)

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from sklearn.model_selection import ParameterGrid

# PARAMETERS
n = 500
k = 6
a = 0
beta = 0.3  # Infection probability
rho = 0.7  # Recovery probability
n_weeks = 15
n_initial_infected = 10
vaccination = [0, 5, 15, 25, 35, 45, 55, 60, 60, 60, 60, 60, 60, 60, 60]
n_iter = 100

new_infected_over_time = np.zeros(n_weeks, dtype=int)
susceptible_over_time = np.zeros(n_weeks, dtype=int)
infected_over_time = np.zeros(n_weeks, dtype=int)
recovered_over_time = np.zeros(n_weeks, dtype=int)
vaccinated_over_time = np.zeros(n_weeks, dtype=int)
new_vaccinated_over_time = np.zeros(n_weeks, dtype=int)

# GRAPH CONSTRUCTION
G = Generate_PA_Graph(n, k, a)
W = nx.adjacency_matrix(G)

if n < 32:
  plt.figure(1,figsize=(6,6))
  nx.draw(G, with_labels=True)

# SIMULATION
for iter in range(0,n_iter):

  # Selecting n_initial_infected
  x0 = np.zeros(n, dtype=int)  # initial configuration
```

```
  initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
  for infected in initial_infected:
    x0[infected] = 1
  states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

  susceptible_over_time[0] += n - n_initial_infected
  infected_over_time[0] += n_initial_infected
  recovered_over_time[0] += 0
  new_infected_over_time[0] += n_initial_infected
  vaccinated_over_time[0] += 0

  for week in range(1,n_weeks):
    last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
    current_week_config = np.zeros(n, dtype=int)
    new_infected_cnt = 0

    # Vaccination
    n_to_be_vaccinated = int(((vaccination[week] - vaccination[week-1]) / 100) * n)
    nodes_not_yet_vaccinated = np.where(last_week_config != 3)[0]
    nodes_to_be_vaccinated = np.random.choice(nodes_not_yet_vaccinated,
size=n_to_be_vaccinated, replace=False)
    for node in nodes_to_be_vaccinated:
      last_week_config[node] = 3

    for node in range(0, n):

      if last_week_config[node] == 0:  # count infected neighbours and compute infection
probability
        m = np.count_nonzero(np.multiply(W.tocsr()[node].toarray()[0], last_week_config)
== 1.0)
        current_week_config[node] = np.random.choice([0, 1], p=[(1-beta)**m, 1-((1-
beta)**m)])
        if current_week_config[node] == 1:  # Count newly infected
          new_infected_cnt += 1

      elif last_week_config[node] == 1:  # compute recovery probability
        current_week_config[node] = np.random.choice([1, 2], p=[1-rho, rho])

      else:  # stays recovered or vaccinated
        current_week_config[node] = last_week_config[node]

    states = sp.sparse.vstack([states,current_week_config])

    # Counts for visualizations
    susceptible_over_time[week] += np.count_nonzero(current_week_config == 0)
    infected_over_time[week] += np.count_nonzero(current_week_config == 1)
    recovered_over_time[week] += np.count_nonzero(current_week_config == 2)
    vaccinated_over_time[week] += np.count_nonzero(current_week_config == 3)
    new_infected_over_time[week] += new_infected_cnt
    new_vaccinated_over_time[week] += n_to_be_vaccinated

avg_susceptible_over_time = susceptible_over_time / n_iter
avg_infected_over_time = infected_over_time / n_iter
avg_recovered_over_time = recovered_over_time / n_iter
avg_vaccinated_over_time = vaccinated_over_time / n_iter
avg_new_infected_over_time = new_infected_over_time / n_iter
avg_new_vaccinated_over_time = new_vaccinated_over_time / n_iter

time = [i for i in range(1,n_weeks+1)]
fig_1, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_new_infected_over_time, c='red', marker='o', linestyle='-',
linewidth=2, label='Newly Infected')
```

```
ax.plot(time, avg_new_vaccinated_over_time, c='darkviolet', marker='o', linestyle='-',
linewidth=2, label='Newly Vaccinated')
ax.legend(loc=(0.75, 0.8))
ax.set_xlabel('Week')
ax.set_title('Avg weekly new conditions')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_new_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y+0.5, str(y), color='red')
for x, y in enumerate(avg_new_vaccinated_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-1.5, str(y), color='darkviolet')
plt.show()

fig_2, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_susceptible_over_time, c='forestgreen', marker='o', linestyle='-',
linewidth=2, label='S')
ax.plot(time, avg_infected_over_time, c='red', marker='o', linestyle='-', linewidth=2,
label='I')
ax.plot(time, avg_recovered_over_time, c='royalblue', marker='o', linestyle='-',
linewidth=2, label='R')
ax.plot(time, avg_vaccinated_over_time, c='darkviolet', marker='o', linestyle='-',
linewidth=2, label='V')
ax.legend(loc=(0.85, 0.70))
ax.set_xlabel('Week')
ax.set_title('Avg weekly configuration')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_susceptible_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_recovered_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_vaccinated_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
plt.show()

if n < 32:
  # Plot the infection spread
  pos = nx.spectral_layout(G)
  fig = plt.figure(figsize=(20,20))
  for t in range(0,n_weeks):
      plt.subplot(3,5,t+1)
      x = np.array(states.tocsr()[t].toarray()[0], dtype=int)
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==0).T[0].tolist(),
          node_color = 'forestgreen')
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==1).T[0].tolist(),
          node_color = 'red')
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==2).T[0].tolist(),
          node_color = 'royalblue')
```

```
      nx.draw_spectral(G,
          with_labels=True,
          nodelist=np.argwhere(x==3).T[0].tolist(),
          node_color = 'darkviolet')
      plt.title('Week = {0}'.format(t+1))
```

V)

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from sklearn.model_selection import ParameterGrid

def rmse(real, simulated):
  return np.sqrt(np.mean(np.power(np.subtract(simulated, real), 2)))

def mse(real, simulated):
  return np.mean(np.power(np.subtract(simulated, real), 2))

n = 934
a = 0
n_weeks = 15
n_initial_infected = 1
n_iter = 10

vaccination = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60]
real_new_infected_over_time = np.array([1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0])

flag = 0
best_error = float('inf')
best_config = {}
best_error_hystory = [100000]
index = 0
tried_configs = []  # Avoid computing twice

while True:

  if index == 0:
    k_0 = 6
    beta_0 = 0.3
    rho_0 = 0.6
    D_k = 1
    D_beta = 0.1
    D_rho = 0.1
  else:
    k_0 = best_config['k']
    beta_0 = best_config['beta']
    rho_0 = best_config['rho']

  print('\n', f"Now running config: K = {k_0}, beta_0 = {beta_0}, rho_0 = {rho_0}, D_k =
{D_k}, D_beta = {D_beta}, D_rho = {D_rho}" '\n')

  parameters = {
                "k": [k_0 - D_k, k_0, k_0 + D_k],
                "beta": [round(beta_0 - D_beta, 3), beta_0, beta_0 + D_beta],
                "rho": [round(rho_0 - D_rho, 3), rho_0, rho_0 + D_rho],
                "D_k": [D_k],
                "D_beta": [D_beta],
                "D_rho": [D_rho]
                }
```

```
  for config in ParameterGrid(parameters):

    if config['k'] < 0:
      config['k'] = 0
    if config['beta'] < 0:
      continue
    if config['rho'] < 0:
      config['rho'] = 0

    if config in tried_configs:
      print(f"Simulation Parameters: {config} - Already Computed")
      continue
    else:
      tried_configs.append(config)

    # GRAPH CONSTRUCTION
    G = Generate_PA_Graph(n, config['k'], a)
    W = nx.adjacency_matrix(G)

    simulated_new_infected_over_time = np.zeros(n_weeks, dtype=int)

    # SIMULATION
    for iter in range(0,n_iter):

      # Selecting n_initial_infected
      x0 = np.zeros(n, dtype=int)  # initial configuration
      initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
      for infected in initial_infected:
        x0[infected] = 1
      states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

      simulated_new_infected_over_time[0] += n_initial_infected

      for week in range(1,n_weeks):
        last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
        current_week_config = np.zeros(n, dtype=int)
        new_infected_cnt = 0

        # Vaccination
        n_to_be_vaccinated = int(((vaccination[week] - vaccination[week-1]) / 100) * n)
        nodes_not_yet_vaccinated = np.where(last_week_config != 3)[0]
        nodes_to_be_vaccinated = np.random.choice(nodes_not_yet_vaccinated,
size=n_to_be_vaccinated, replace=False)
        for node in nodes_to_be_vaccinated:
          last_week_config[node] = 3

        for node in range(0, n):

          if last_week_config[node] == 0:  # count infected neighbours and compute
infection probability
            m = np.count_nonzero(np.multiply(W.tocsr()[node].toarray()[0],
last_week_config) == 1.0)
            current_week_config[node] = np.random.choice([0, 1], p=[(1-
config['beta'])**m, 1-((1-config['beta'])**m)])
            if current_week_config[node] == 1:  # Count newly infected
              new_infected_cnt += 1

          elif last_week_config[node] == 1:  # compute recovery probability
            current_week_config[node] = np.random.choice([1, 2], p=[1-config['rho'],
config['rho']])

          else:  # stays recovered or vaccinated
            current_week_config[node] = last_week_config[node]
```

```
        states = sp.sparse.vstack([states,current_week_config])

        simulated_new_infected_over_time[week] += new_infected_cnt

    avg_simulated_new_infected_over_time = simulated_new_infected_over_time / n_iter
    error = rmse(real_new_infected_over_time, avg_simulated_new_infected_over_time)

    if error < best_error:
      best_error = error
      best_config = config

    print(f"Simulation Parameters: {config} - RMSE: {round(error, 3)}")

  if best_error >= best_error_hystory[index]:
    if flag == 0:
      print('\n', f"No improvement. Decreasing Deltas", '\n')
      flag = 1
      D_beta = D_beta / 2
      D_rho = D_rho / 2
    else:
      break  # breaks only if there is no improvement for two times in a row
  else:
    best_error_hystory.append(best_error)
    index += 1
    flag = 0
    print('\n', f"Best Config: {best_config} - Error: {best_error}")

"""
Good configurations:
k=6, beta=0.3, rho=0.6
k=11, beta=0.15, rho=0.6
k=5, beta=0.4, rho=0.7
"""

# PARAMETERS
n = 934
k = 6
a = 0
beta = 0.3  # Infection probability
rho = 0.6  # Recovery probability

n_weeks = 15
n_initial_infected = 1
vaccination = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60]
real_new_infected_over_time = np.array([1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0])
n_iter = 100

new_infected_over_time = np.zeros(n_weeks, dtype=int)
susceptible_over_time = np.zeros(n_weeks, dtype=int)
infected_over_time = np.zeros(n_weeks, dtype=int)
recovered_over_time = np.zeros(n_weeks, dtype=int)
vaccinated_over_time = np.zeros(n_weeks, dtype=int)
new_vaccinated_over_time = np.zeros(n_weeks, dtype=int)

# GRAPH CONSTRUCTION
G = Generate_PA_Graph(n, k, a)
W = nx.adjacency_matrix(G)

# SIMULATION
for iter in range(0,n_iter):

  # Selecting n_initial_infected
  x0 = np.zeros(n, dtype=int)  # initial configuration
```

```
  initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
  for infected in initial_infected:
    x0[infected] = 1
  states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

  susceptible_over_time[0] += n - n_initial_infected
  infected_over_time[0] += n_initial_infected
  recovered_over_time[0] += 0
  new_infected_over_time[0] += n_initial_infected
  vaccinated_over_time[0] += 0

  for week in range(1,n_weeks):
    last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
    current_week_config = np.zeros(n, dtype=int)
    new_infected_cnt = 0

    # Vaccination
    n_to_be_vaccinated = int(((vaccination[week] - vaccination[week-1]) / 100) * n)
    nodes_not_yet_vaccinated = np.where(last_week_config != 3)[0]
    nodes_to_be_vaccinated = np.random.choice(nodes_not_yet_vaccinated,
size=n_to_be_vaccinated, replace=False)
    for node in nodes_to_be_vaccinated:
      last_week_config[node] = 3

    for node in range(0, n):

      if last_week_config[node] == 0:  # count infected neighbours and compute infection
probability
        m = np.count_nonzero(np.multiply(W.tocsr()[node].toarray()[0], last_week_config)
== 1.0)
        current_week_config[node] = np.random.choice([0, 1], p=[(1-beta)**m, 1-((1-
beta)**m)])
        if current_week_config[node] == 1:  # Count newly infected
          new_infected_cnt += 1

      elif last_week_config[node] == 1:  # compute recovery probability
        current_week_config[node] = np.random.choice([1, 2], p=[1-rho, rho])

      else:  # stays recovered or vaccinated
        current_week_config[node] = last_week_config[node]

    states = sp.sparse.vstack([states,current_week_config])

    # Counts for visualizations
    susceptible_over_time[week] += np.count_nonzero(current_week_config == 0)
    infected_over_time[week] += np.count_nonzero(current_week_config == 1)
    recovered_over_time[week] += np.count_nonzero(current_week_config == 2)
    vaccinated_over_time[week] += np.count_nonzero(current_week_config == 3)
    new_infected_over_time[week] += new_infected_cnt
    new_vaccinated_over_time[week] += n_to_be_vaccinated

avg_susceptible_over_time = susceptible_over_time / n_iter
avg_infected_over_time = infected_over_time / n_iter
avg_recovered_over_time = recovered_over_time / n_iter
avg_vaccinated_over_time = vaccinated_over_time / n_iter
avg_new_infected_over_time = new_infected_over_time / n_iter
avg_new_vaccinated_over_time = new_vaccinated_over_time / n_iter

time = [i for i in range(1,n_weeks+1)]
fig_1, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_new_infected_over_time, c='red', marker='o', linestyle='-',
linewidth=2, label='Newly Infected')
```

```
ax.plot(time, avg_new_vaccinated_over_time, c='darkviolet', marker='o', linestyle='-',
linewidth=2, label='Newly Vaccinated')
ax.plot(time, real_new_infected_over_time, c='orange', marker='o', linestyle='-',
linewidth=2, label='Real Newly Infected')
ax.legend(loc=(0.75, 0.8))
ax.set_xlabel('Week')
ax.set_title('Avg weekly new conditions')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_new_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-1, str(y), color='k')
plt.show()

fig_2, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_susceptible_over_time, c='forestgreen', marker='o', linestyle='-',
linewidth=2, label='S')
ax.plot(time, avg_infected_over_time, c='red', marker='o', linestyle='-', linewidth=2,
label='I')
ax.plot(time, avg_recovered_over_time, c='royalblue', marker='o', linestyle='-',
linewidth=2, label='R')
ax.plot(time, avg_vaccinated_over_time, c='darkviolet', marker='o', linestyle='-',
linewidth=2, label='V')
ax.legend(loc=(0.85, 0.70))
ax.set_xlabel('Week')
ax.set_title('Avg weekly configuration')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_susceptible_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_recovered_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_vaccinated_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
plt.show()
```

VI)

```
Now running config: K = 6, beta_0 = 0.3, rho_0 = 0.6, D_k = 1, D_beta = 0.1, D_rho = 0.1

Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.5} - RMSE: 11.223
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.6} - RMSE: 8.114
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.7} - RMSE: 10.868
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.5} - RMSE: 6.477
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.6} - RMSE: 12.172
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.7} - RMSE: 11.391
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.5} - RMSE: 6.343
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.6} - RMSE: 8.266
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.7} - RMSE: 9.227
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.5} - RMSE: 6.341
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.6} - RMSE: 6.678
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.7} - RMSE: 9.05
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.5} - RMSE: 8.258
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.6} - RMSE: 6.649
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.7} - RMSE: 12.556
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.5} - RMSE: 21.156
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.6} - RMSE: 10.976
```

```
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.7} - RMSE: 20.162
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.5} - RMSE: 8.936
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.6} - RMSE: 22.76
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.7} - RMSE: 5.242
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.5} - RMSE: 17.174
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.6} - RMSE: 18.347
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.7} - RMSE: 22.481
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 7, 'rho': 0.5} - RMSE: 43.095
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 7, 'rho': 0.6} - RMSE: 34.31
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 7, 'rho': 0.7} - RMSE: 36.583

 Best Config: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.7} - Error: 5.242136968832463

 Now running config: K = 5, beta_0 = 0.4, rho_0 = 0.7, D_k = 1, D_beta = 0.1, D_rho = 0.1

Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 4, 'rho': 0.6} - RMSE: 10.921
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 4, 'rho': 0.7} - RMSE: 11.362
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 4, 'rho': 0.7999999999999999} -
RMSE: 8.352
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.7999999999999999} -
RMSE: 8.733
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.7999999999999999} -
RMSE: 9.891
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 4, 'rho': 0.6} - RMSE: 9.385
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 4, 'rho': 0.7} - RMSE: 8.031
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 4, 'rho': 0.7999999999999999} -
RMSE: 10.353
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.7999999999999999} -
RMSE: 13.703
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.7999999999999999} -
RMSE: 14.038
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 4, 'rho': 0.6} - RMSE: 17.498
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 4, 'rho': 0.7} - RMSE: 18.133
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 4, 'rho': 0.7999999999999999} -
RMSE: 25.881
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 5, 'rho': 0.6} - RMSE: 14.745
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 5, 'rho': 0.7} - RMSE: 17.958
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 5, 'rho': 0.7999999999999999} -
RMSE: 25.828
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 6, 'rho': 0.6} - RMSE: 43.175
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 6, 'rho': 0.7} - RMSE: 35.817
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.5, 'k': 6, 'rho': 0.7999999999999999} -
RMSE: 36.275

 No improvement. Decreasing Deltas


 Now running config: K = 5, beta_0 = 0.4, rho_0 = 0.7, D_k = 1, D_beta = 0.05, D_rho = 0.05

Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 4, 'rho': 0.65} - RMSE: 9.105
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 4, 'rho': 0.7} - RMSE: 6.535
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 4, 'rho': 0.75} - RMSE: 8.245
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 5, 'rho': 0.65} - RMSE: 9.158
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 5, 'rho': 0.7} - RMSE: 5.917
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 5, 'rho': 0.75} - RMSE: 8.111
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 6, 'rho': 0.65} - RMSE: 11.837
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 6, 'rho': 0.7} - RMSE: 15.432
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.35, 'k': 6, 'rho': 0.75} - RMSE: 14.279
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 4, 'rho': 0.65} - RMSE: 13.655
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 4, 'rho': 0.7} - RMSE: 8.539
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 4, 'rho': 0.75} - RMSE: 10.119
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 5, 'rho': 0.65} - RMSE: 15.814
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 5, 'rho': 0.7} - RMSE: 18.133
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 5, 'rho': 0.75} - RMSE: 10.063
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 6, 'rho': 0.65} - RMSE: 36.525
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 6, 'rho': 0.7} - RMSE: 31.368
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.4, 'k': 6, 'rho': 0.75} - RMSE: 12.465
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 4, 'rho': 0.65} - RMSE: 12.344
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 4, 'rho': 0.7} - RMSE: 8.724
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 4, 'rho': 0.75} - RMSE: 13.414
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 5, 'rho': 0.65} - RMSE: 15.567
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 5, 'rho': 0.7} - RMSE: 13.443
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 5, 'rho': 0.75} - RMSE: 20.866
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 6, 'rho': 0.65} - RMSE: 37.505
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 6, 'rho': 0.7} - RMSE: 40.324
```

```
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.45, 'k': 6, 'rho': 0.75} - RMSE: 19.351
```

VII)

```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sp
from sklearn.model_selection import ParameterGrid

def Generate_ER_Graph(total_nodes, avg_degree):

  G = nx.Graph()
  p = avg_degree / (total_nodes-1)

  for node_1 in range(0, total_nodes):
    for node_2 in range(0, total_nodes):
      if node_1 != node_2 and (np.random.choice([0, 1], p=[1-p,p])) == 1:
        G.add_edge(node_1,node_2)  # Edges creation

  G.add_nodes_from([i for i in range(0, total_nodes)])

  return G

  def rmse(real, simulated):
    return np.sqrt(np.mean(np.power(np.subtract(simulated, real), 2)))

n = 934
n_weeks = 15
n_initial_infected = 1
n_iter = 10

vaccination = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60]
real_new_infected_over_time = np.array([1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0])

flag = 0
best_error = float('inf')
best_config = {}
best_error_hystory = [100000]
index = 0
tried_configs = []

while True:

  if index == 0:
    k_0 = 6
    beta_0 = 0.3
    rho_0 = 0.6
    D_k = 1
    D_beta = 0.1
    D_rho = 0.1
  else:
    k_0 = best_config['k']
    beta_0 = best_config['beta']
    rho_0 = best_config['rho']

  print('\n', f"Now running config: K = {k_0}, beta_0 = {beta_0}, rho_0 = {rho_0}, D_k =
{D_k}, D_beta = {D_beta}, D_rho = {D_rho}" '\n')

  parameters = {
                "k": [k_0 - D_k, k_0, k_0 + D_k],
                "beta": [round(beta_0 - D_beta, 3), beta_0, round(beta_0 + D_beta, 3)],
```

```
                "rho": [round(rho_0 - D_rho, 3), rho_0, round(rho_0 + D_rho, 3)],
                "D_k": [D_k],
                "D_beta": [D_beta],
                "D_rho": [D_rho]
                }

  for config in ParameterGrid(parameters):

    if config['k'] < 0:
      config['k'] = 0
    if config['beta'] < 0:
      continue
    if config['rho'] < 0:
      config['rho'] = 0

    if config in tried_configs:
      print(f"Simulation Parameters: {config} - Already Computed")
      continue
    else:
      tried_configs.append(config)

    # GRAPH CONSTRUCTION
    G = Generate_ER_Graph(n, config['k'])
    W = nx.adjacency_matrix(G)

    simulated_new_infected_over_time = np.zeros(n_weeks, dtype=int)

    # SIMULATION
    for iter in range(0,n_iter):

      # Selecting n_initial_infected
      x0 = np.zeros(n, dtype=int)  # initial configuration
      initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
      for infected in initial_infected:
        x0[infected] = 1
      states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

      simulated_new_infected_over_time[0] += n_initial_infected

      for week in range(1,n_weeks):
        last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
        current_week_config = np.zeros(n, dtype=int)
        new_infected_cnt = 0

        # Vaccination
        n_to_be_vaccinated = int(((vaccination[week] - vaccination[week-1]) / 100) * n)
        nodes_not_yet_vaccinated = np.where(last_week_config != 3)[0]
        nodes_to_be_vaccinated = np.random.choice(nodes_not_yet_vaccinated,
size=n_to_be_vaccinated, replace=False)
        for node in nodes_to_be_vaccinated:
          last_week_config[node] = 3

        for node in range(0, n):

          if last_week_config[node] == 0:  # count infected neighbours and compute
infection probability
            m = np.count_nonzero(np.multiply(W.tocsr()[node].toarray()[0],
last_week_config) == 1.0)
            current_week_config[node] = np.random.choice([0, 1], p=[(1-
config['beta'])**m, 1-((1-config['beta'])**m)])
            if current_week_config[node] == 1:  # Count newly infected
              new_infected_cnt += 1
```

```
          elif last_week_config[node] == 1:  # compute recovery probability
            current_week_config[node] = np.random.choice([1, 2], p=[1-config['rho'],
config['rho']])

          else:  # stays recovered or vaccinated
            current_week_config[node] = last_week_config[node]

        states = sp.sparse.vstack([states,current_week_config])

        simulated_new_infected_over_time[week] += new_infected_cnt

    avg_simulated_new_infected_over_time = simulated_new_infected_over_time / n_iter
    error = rmse(real_new_infected_over_time, avg_simulated_new_infected_over_time)

    if error < best_error:
      best_error = error
      best_config = config

    print(f"Simulation Parameters: {config} - RMSE: {round(error, 3)}")

  if best_error >= best_error_hystory[index]:
    if flag == 0:
      print('\n', f"No improvement. Decreasing Deltas", '\n')
      flag = 1
      D_beta = D_beta / 2
      D_rho = D_rho / 2
    else:
      break  # breaks only if there is no improvement for two times in a row
  else:
    best_error_hystory.append(best_error)
    index += 1
    flag = 0
    print('\n', f"Best Config: {best_config} - Error: {best_error}")

"""
Good configurations:
k=13, beta=0.1, rho=0.9
k=6, beta=0.2, rho=0.6
"""

# PARAMETERS
n = 934
k = 13
beta = 0.1  # Infection probability
rho = 0.9  # Recovery probability

n_weeks = 15
n_initial_infected = 1
vaccination = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60]
real_new_infected_over_time = np.array([1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0])
n_iter = 100

new_infected_over_time = np.zeros(n_weeks, dtype=int)
susceptible_over_time = np.zeros(n_weeks, dtype=int)
infected_over_time = np.zeros(n_weeks, dtype=int)
recovered_over_time = np.zeros(n_weeks, dtype=int)
vaccinated_over_time = np.zeros(n_weeks, dtype=int)
new_vaccinated_over_time = np.zeros(n_weeks, dtype=int)

# GRAPH CONSTRUCTION
G = Generate_ER_Graph(n, k)
W = nx.adjacency_matrix(G)

# SIMULATION
```

```
for iter in range(0,n_iter):

  # Selecting n_initial_infected
  x0 = np.zeros(n, dtype=int)  # initial configuration
  initial_infected = np.random.choice([i for i in range(0,n)], n_initial_infected,
replace=False)
  for infected in initial_infected:
    x0[infected] = 1
  states = sp.sparse.lil_matrix(x0)  # insertion in configuration matrix

  susceptible_over_time[0] += n - n_initial_infected
  infected_over_time[0] += n_initial_infected
  recovered_over_time[0] += 0
  new_infected_over_time[0] += n_initial_infected
  vaccinated_over_time[0] += 0

  for week in range(1,n_weeks):
    last_week_config = np.array(states.tocsr()[week-1].toarray()[0], dtype=int)
    current_week_config = np.zeros(n, dtype=int)
    new_infected_cnt = 0

    # Vaccination
    n_to_be_vaccinated = int(((vaccination[week] - vaccination[week-1]) / 100) * n)
    nodes_not_yet_vaccinated = np.where(last_week_config != 3)[0]
    nodes_to_be_vaccinated = np.random.choice(nodes_not_yet_vaccinated,
size=n_to_be_vaccinated, replace=False)
    for node in nodes_to_be_vaccinated:
      last_week_config[node] = 3

    for node in range(0, n):

      if last_week_config[node] == 0:  # count infected neighbours and compute infection
probability
        m = np.count_nonzero(np.multiply(W.tocsr()[node].toarray()[0], last_week_config)
== 1.0)
        current_week_config[node] = np.random.choice([0, 1], p=[(1-beta)**m, 1-((1-
beta)**m)])
        if current_week_config[node] == 1:  # Count newly infected
          new_infected_cnt += 1

      elif last_week_config[node] == 1:  # compute recovery probability
        current_week_config[node] = np.random.choice([1, 2], p=[1-rho, rho])

      else:  # stays recovered or vaccinated
        current_week_config[node] = last_week_config[node]

    states = sp.sparse.vstack([states,current_week_config])

    # Counts for visualizations
    susceptible_over_time[week] += np.count_nonzero(current_week_config == 0)
    infected_over_time[week] += np.count_nonzero(current_week_config == 1)
    recovered_over_time[week] += np.count_nonzero(current_week_config == 2)
    vaccinated_over_time[week] += np.count_nonzero(current_week_config == 3)
    new_infected_over_time[week] += new_infected_cnt
    new_vaccinated_over_time[week] += n_to_be_vaccinated

avg_susceptible_over_time = susceptible_over_time / n_iter
avg_infected_over_time = infected_over_time / n_iter
avg_recovered_over_time = recovered_over_time / n_iter
avg_vaccinated_over_time = vaccinated_over_time / n_iter
avg_new_infected_over_time = new_infected_over_time / n_iter
avg_new_vaccinated_over_time = new_vaccinated_over_time / n_iter

time = [i for i in range(1,n_weeks+1)]
```

```
fig_1, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_new_infected_over_time, c='red', marker='o', linestyle='-',
linewidth=2, label='Newly Infected')
ax.plot(time, avg_new_vaccinated_over_time, c='darkviolet', marker='o', linestyle='-',
linewidth=2, label='Newly Vaccinated')
ax.plot(time, real_new_infected_over_time, c='orange', marker='o', linestyle='-',
linewidth=2, label='Real Newly Infected')
ax.legend(loc=(0.75, 0.8))
ax.set_xlabel('Week')
ax.set_title('Avg weekly new conditions')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_new_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-1, str(y), color='k')
plt.show()

fig_2, ax = plt.subplots(figsize=(10, 6))
ax.plot(time, avg_susceptible_over_time, c='forestgreen', marker='o', linestyle='-',
linewidth=2, label='S')
ax.plot(time, avg_infected_over_time, c='red', marker='o', linestyle='-', linewidth=2,
label='I')
ax.plot(time, avg_recovered_over_time, c='royalblue', marker='o', linestyle='-',
linewidth=2, label='R')
ax.plot(time, avg_vaccinated_over_time, c='darkviolet', marker='o', linestyle='-',
linewidth=2, label='V')
ax.legend(loc=(0.85, 0.70))
ax.set_xlabel('Week')
ax.set_title('Avg weekly configuration')
ax.grid('on')
plt.xticks(ticks=[i for i in range(1, n_weeks+1)])
ax.set_xlim(left=None, right=16.5)
for x, y in enumerate(avg_susceptible_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_infected_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_recovered_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
for x, y in enumerate(avg_vaccinated_over_time):
  if x == n_weeks - 1:
    ax.text(x+1.2, y-5, str(y), color='k')
plt.show()
```

VIII)

```
Now running config: K = 6, beta_0 = 0.3, rho_0 = 0.6, D_k = 1, D_beta = 0.1, D_rho = 0.1

Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.5} - RMSE: 8.514
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.6} - RMSE: 8.475
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.7} - RMSE: 7.769
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.5} - RMSE: 17.0
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.6} - RMSE: 6.599
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.7} - RMSE: 9.65
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.5} - RMSE: 24.513
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.6} - RMSE: 17.999
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.7} - RMSE: 19.144
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.5} - RMSE: 28.738
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.6} - RMSE: 20.639
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.7} - RMSE: 17.059
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.5} - RMSE: 41.97
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.6} - RMSE: 23.2
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.7} - RMSE: 42.298
```

```
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.5} - RMSE: 52.753
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.6} - RMSE: 53.079
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.7} - RMSE: 49.054
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.5} - RMSE: 37.306
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.6} - RMSE: 47.783
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 5, 'rho': 0.7} - RMSE: 35.603
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.5} - RMSE: 65.963
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.6} - RMSE: 67.403
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 6, 'rho': 0.7} - RMSE: 63.971
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 7, 'rho': 0.5} - RMSE: 74.011
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 7, 'rho': 0.6} - RMSE: 69.512
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.4, 'k': 7, 'rho': 0.7} - RMSE: 68.0

 Best Config: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.6} - Error: 6.598989821682305

 Now running config: K = 6, beta_0 = 0.2, rho_0 = 0.6, D_k = 1, D_beta = 0.1, D_rho = 0.1

Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 5, 'rho': 0.5} - RMSE: 13.144
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 5, 'rho': 0.6} - RMSE: 13.284
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 5, 'rho': 0.7} - RMSE: 13.33
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 6, 'rho': 0.5} - RMSE: 12.502
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 6, 'rho': 0.6} - RMSE: 13.048
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 6, 'rho': 0.7} - RMSE: 13.019
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 7, 'rho': 0.5} - RMSE: 11.541
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 7, 'rho': 0.6} - RMSE: 11.494
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.1, 'k': 7, 'rho': 0.7} - RMSE: 11.831
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.5} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 5, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.5} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 6, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.5} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.2, 'k': 7, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.5} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 5, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.5} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 6, 'rho': 0.7} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.5} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.6} - Already Computed
Simulation Parameters: {'D_beta': 0.1, 'D_k': 1, 'D_rho': 0.1, 'beta': 0.3, 'k': 7, 'rho': 0.7} - Already Computed

 No improvement. Decreasing Deltas


 Now running config: K = 6, beta_0 = 0.2, rho_0 = 0.6, D_k = 1, D_beta = 0.05, D_rho = 0.05

Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 5, 'rho': 0.55} - RMSE: 11.684
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 5, 'rho': 0.6} - RMSE: 10.71
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 5, 'rho': 0.65} - RMSE: 12.446
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 6, 'rho': 0.55} - RMSE: 9.118
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 6, 'rho': 0.6} - RMSE: 9.919
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 6, 'rho': 0.65} - RMSE: 10.254
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 7, 'rho': 0.55} - RMSE: 11.478
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 7, 'rho': 0.6} - RMSE: 11.26
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.15, 'k': 7, 'rho': 0.65} - RMSE: 7.981
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 5, 'rho': 0.55} - RMSE: 9.16
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 5, 'rho': 0.6} - RMSE: 8.436
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 5, 'rho': 0.65} - RMSE: 9.963
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 6, 'rho': 0.55} - RMSE: 15.292
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 6, 'rho': 0.6} - RMSE: 9.226
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 6, 'rho': 0.65} - RMSE: 10.35
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 7, 'rho': 0.55} - RMSE: 14.627
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 7, 'rho': 0.6} - RMSE: 19.826
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.2, 'k': 7, 'rho': 0.65} - RMSE: 22.149
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 5, 'rho': 0.55} - RMSE: 15.925
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 5, 'rho': 0.6} - RMSE: 15.254
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 5, 'rho': 0.65} - RMSE: 10.14
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 6, 'rho': 0.55} - RMSE: 36.565
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 6, 'rho': 0.6} - RMSE: 22.838
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 6, 'rho': 0.65} - RMSE: 28.071
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 7, 'rho': 0.55} - RMSE: 44.274
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 7, 'rho': 0.6} - RMSE: 39.025
Simulation Parameters: {'D_beta': 0.05, 'D_k': 1, 'D_rho': 0.05, 'beta': 0.25, 'k': 7, 'rho': 0.65} - RMSE: 36.651
```