

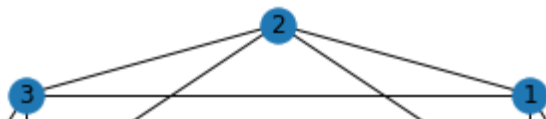
# Network Dynamics and Graph - HW3 - Politecnico di Torino -

## Hafez Ghaemi - S289963

```
1 import numpy as np
2 import networkx as nx
3 import matplotlib.pyplot as plt
4
5 import networkx.generators.random_graphs as rg
6 import math
7
8 import collections
9 from networkx.generators.random_graphs import watts_strogatz_graph as ws
10 import itertools
11
```

### Preliminary parts

```
1 G = nx.Graph()
2 k = 4
3 n = 8
4 for i in range(n-2):
5     G.add_edge(i, i+1)
6     G.add_edge(i, i+2)
7 G.add_edge(n-2, n-1)
8 G.add_edge(n-2, 0)
9 G.add_edge(n-1, 0)
10 G.add_edge(n-1, 1)
11
12 nx.draw_circular(G, with_labels = True)
13 plt.savefig("k_reg.eps",format='eps')
```



```

1 def sim_SIR_epidemic(G, n_weeks = 5, initial_states = 'random', random_sick = 3, beta = 0.
2     '''
3     G: networkx.classes.graph.Graph, graph
4     n_weeks: int, number of simulation weeks
5     initial_states: list, initial state of agents or default 'random' initialize sick peop
6     random_sick: int, how many sick agents if initial state is randomly generated, default
7     beta: float, the probability that the infection is spread from an infected individual
8     rho: float, the probability that an infected individual will recover, default 0.7
9     until_end: boolean, indicates whether the simulation continues until the end or finish
10    '''
11
12    # Number of people
13    n_agents = len(G)
14
15    A_ = [0,1,2] # 0-S, 1-I, 2-R
16
17    states = [0 for _ in range(n_agents)] # 0-S, 1-I, 2-R
18
19    if initial_states == 'random':
20        choices = np.random.choice(G.nodes(),size = random_sick, replace= False)
21        for i in choices:
22            states[i] = 1
23        initial_states = states.copy()
24
25    old_states = initial_states.copy()
26 #    print('Initial states of agents: {}'.format(initial_states))
27
28    W = nx.convert_matrix.to_numpy_matrix(G)
29
30    weeks = []
31    weeks.append(initial_states.copy())
32
33    newly_infected_list = [1]
34    for week in range(n_weeks):
35        if not until_end:
36            if not 1 in states:
37                #print('No one is sick! \nGREAT SUCCESS!')
38                break
39
40 #        print("States: {}, \nWeek: {}".format(states, week+1))
41        newly_infected = 0
42        for agent in range(n_agents):
43            # if S:
44            if old_states[agent] == 0:
45                ngh_states = np.squeeze(np.asarray(W[agent]))*old_states # get neighbors'
46                m = collections.Counter(ngh_states)[1] # count sick neighbors
47

```

```

47
48         if np.random.rand() < (1-(1-beta)**m):
49             states[agent] = 1 # get sick
50             newly_infected += 1
51         # if I:
52         elif old_states[agent] == 1:
53             if np.random.rand() < rho:
54                 states[agent] = 2 # get recovered
55
56         weeks.append(states.copy())
57         newly_infected_list.append(newly_infected)
58
59         old_states = states.copy()
60 #         print(old_states)
61     return weeks, newly_infected_list
62     # weeks = weeks[:5]

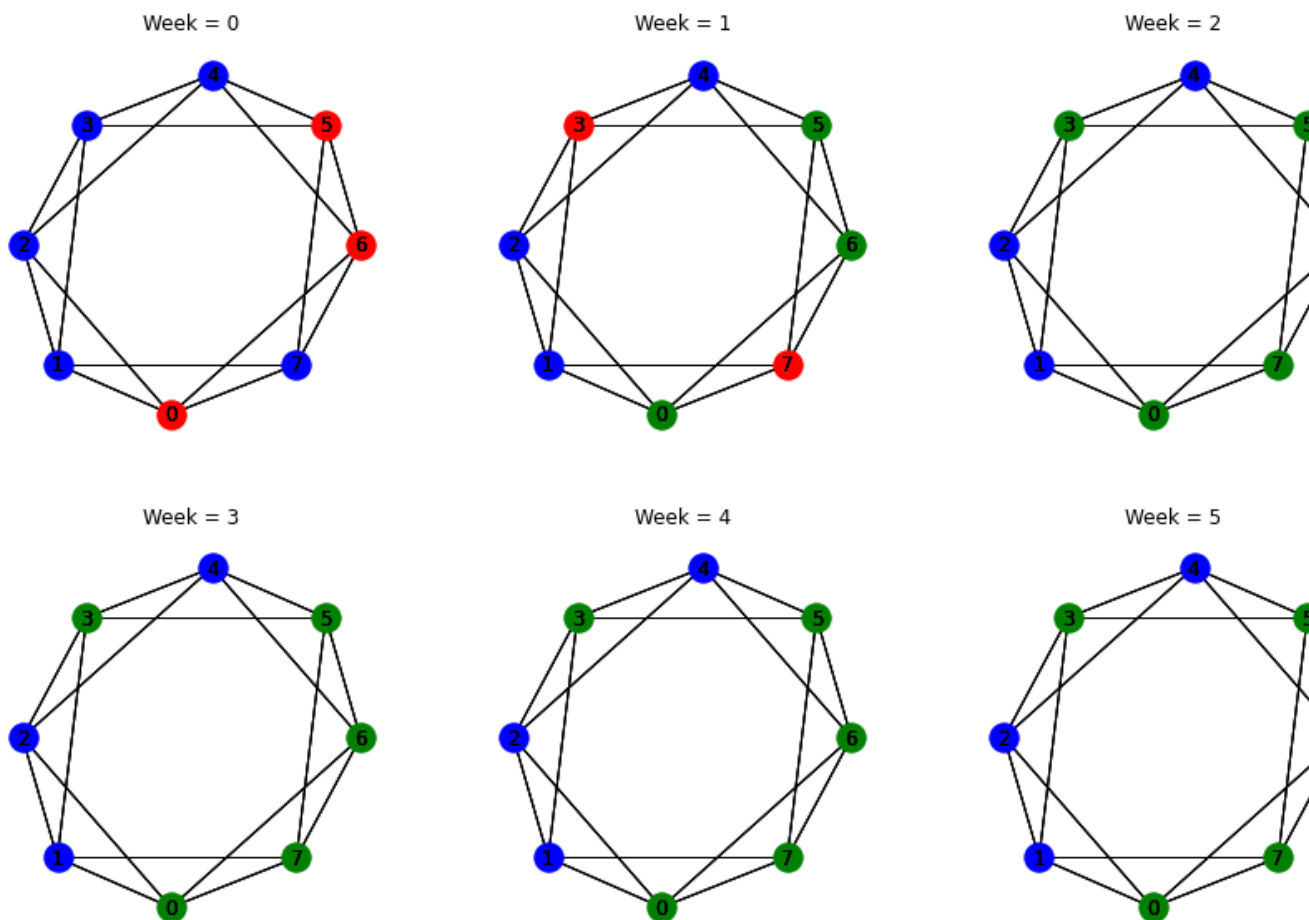
1 def get_sir_count(weeks):
2     '''
3     weeks: list, list of states through weeks
4     '''
5     susceptibles = [collections.Counter(np.asarray(w))[0] for w in weeks]
6     infected = [collections.Counter(np.asarray(w))[1] for w in weeks]
7     recovered = [collections.Counter(np.asarray(w))[2] for w in weeks]
8     #print('susceptibles: {}, \ninfected: {}, \nrecovered: {}'.format(susceptibles, infect
9     return susceptibles, infected, recovered

1 weeks, newly_infected_list = sim_SIR_epidemic(G, n_weeks=5, initial_states='random', rando
2 # Plot the infection spread
3 pos = nx.spectral_layout(G)
4 fig = plt.figure(figsize=(15,10))
5 for t in range(0,len(weeks)):
6     plt.subplot(2,3,t+1)
7     week = weeks[t]
8
9     x = np.array(week.copy())
10    nx.draw_spectral(G,
11        with_labels=True,
12        nodelist=np.argwhere(x==0).T[0].tolist(),
13        node_color = 'b')
14    nx.draw_spectral(G,
15        with_labels=True,
16        nodelist=np.argwhere(x==1).T[0].tolist(),
17        node_color = 'r')
18    nx.draw_spectral(G,
19        with_labels=True,
20        nodelist=np.argwhere(x==2).T[0].tolist(),
21        node_color = 'g')
22    plt.title('Week = {}'.format(t))

```

```
23 susceptibles, infected, recovered = get_sir_count(weeks)
```

```
24
```



## ▼ Problem 1.1

### ▼ Random Initial Condition

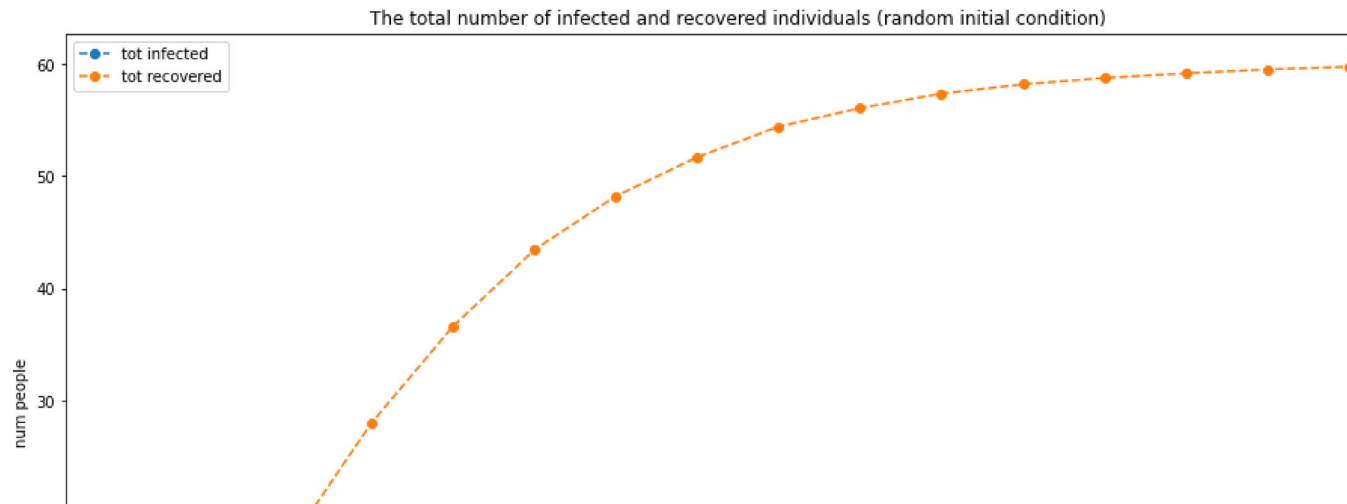
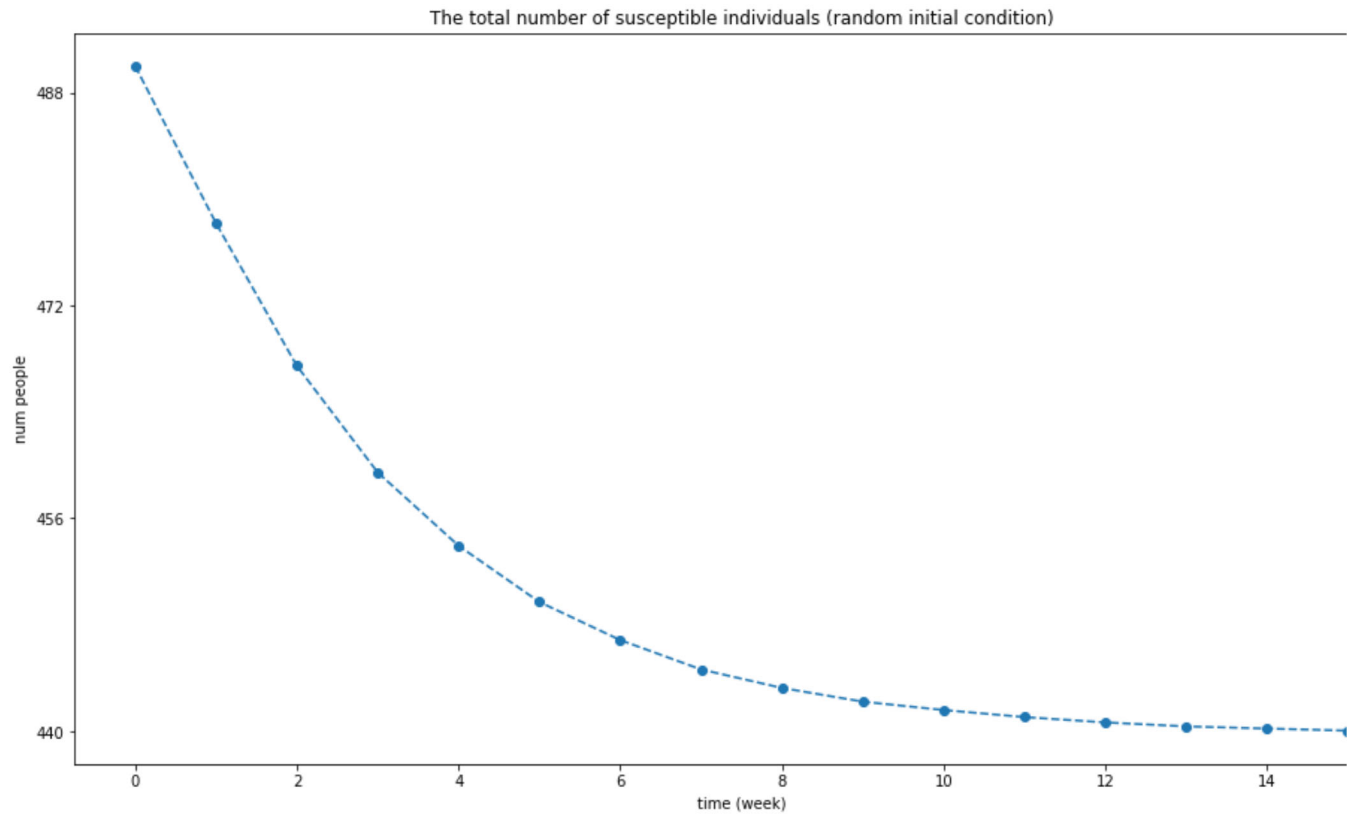
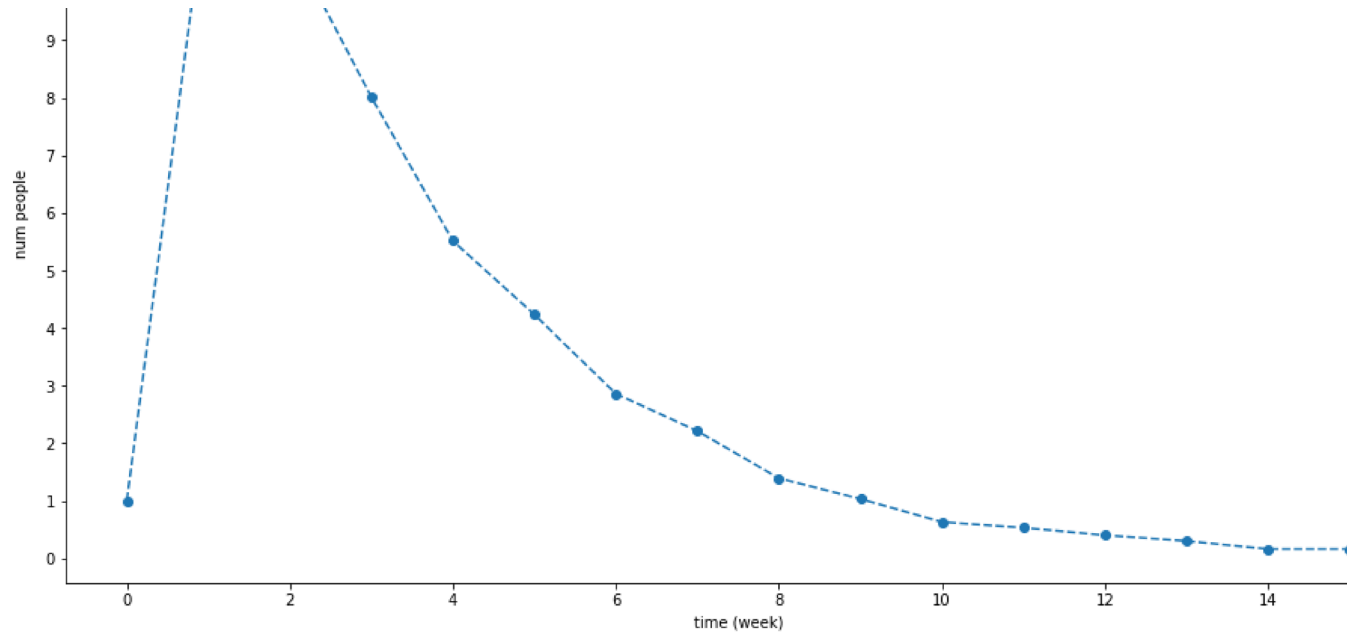
```
1 G = nx.Graph()
2 k = 4
3 n = 500
4 for i in range(n-2):
5     G.add_edge(i, i+1)
6     G.add_edge(i, i+2)
7 G.add_edge(n-2, n-1)
8 G.add_edge(n-2, 0)
9 G.add_edge(n-1, 0)
10 G.add_edge(n-1, 1)
```

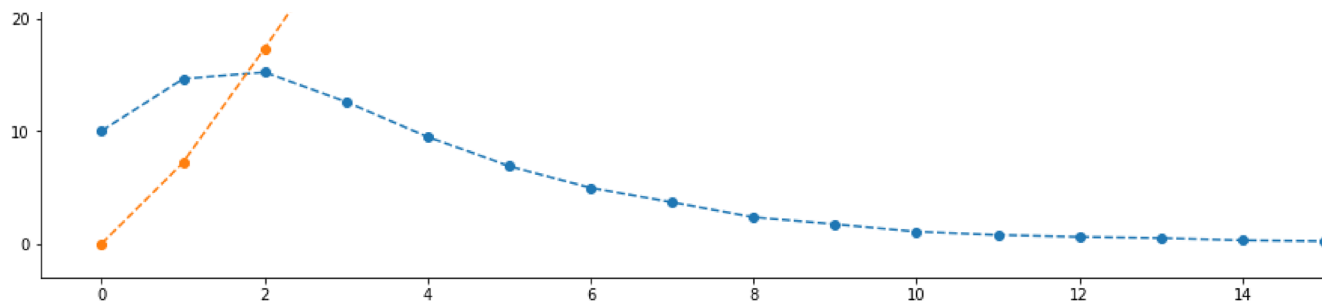
```

11
12
13 N = 100
14 sus_list = []
15 inf_list = []
16 rec_list = []
17 new_inf_list = []
18 for n in range(N):
19     # print("Simulation #{0}".format(n+1))
20     weeks, newly_infected_list = sim_SIR_epidemic(G, n_weeks=15, initial_states='random',
21     susceptibles, infected, recovered = get_sir_count(weeks)
22     # print('newly infected: {0}\n'.format(newly_infected_list))
23     sus_list.append(susceptibles)
24     inf_list.append(infected)
25     rec_list.append(recovered)
26     new_inf_list.append(newly_infected_list)
27 new_inf_list = np.array(new_inf_list)
28

1 avg_sus = np.mean(np.array(sus_list), axis=0)
2 avg_inf = np.mean(np.array(inf_list), axis=0)
3 avg_rec = np.mean(np.array(rec_list), axis=0)
4 avg_new_inf = np.mean(new_inf_list, axis=0)
5 fig, ax= plt.subplots(figsize=(16,9))
6 ax.plot(avg_new_inf, label='newly infected',linestyle='--', marker='o')
7 ax.set(xlabel='time (week)', ylabel='num people')
8 yint = range(int(min(avg_new_inf)), math.ceil(max(avg_new_inf))+1)
9 ax.set_yticks(yint)
10 ax.set_title("The number of newly infected individuals (random initial condition)")
11 plt.savefig("new_inf.eps",format='eps')
12 fig, ax= plt.subplots(figsize=(16,9))
13 ax.plot(avg_sus, label='tot avg sus',linestyle='--', marker='o')
14 ax.set(xlabel='time (week)', ylabel='num people')
15 yint = range(int(min(avg_sus)), math.ceil(max(avg_sus))+1, len(avg_sus))
16 ax.set_yticks(yint)
17 ax.set_title("The total number of susceptible individuals (random initial condition)")
18 plt.savefig("tot_sus.eps",format='eps')
19 fig, ax= plt.subplots(figsize=(16,9))
20 ax.plot(avg_inf, label='tot infected',linestyle='--', marker='o')
21 ax.plot(avg_rec, label='tot recovered',linestyle='--', marker='o')
22 ax.set(xlabel='time (week)', ylabel='num people')
23 ax.set_title("The total number of infected and recovered individuals (random initial condi
24 ax.legend(loc='best')
25 plt.savefig("tot_rec_inf.eps",format='eps')

```





## ▼ Realistic Initial Condition

```

1 G = nx.Graph()
2 k = 4
3 n = 500
4 for i in range(n-2):
5     G.add_edge(i, i+1)
6     G.add_edge(i, i+2)
7 G.add_edge(n-2, n-1)
8 G.add_edge(n-2, 0)
9 G.add_edge(n-1, 0)
10 G.add_edge(n-1, 1)
11
12 sus_list = []
13 inf_list = []
14 rec_list = []

```

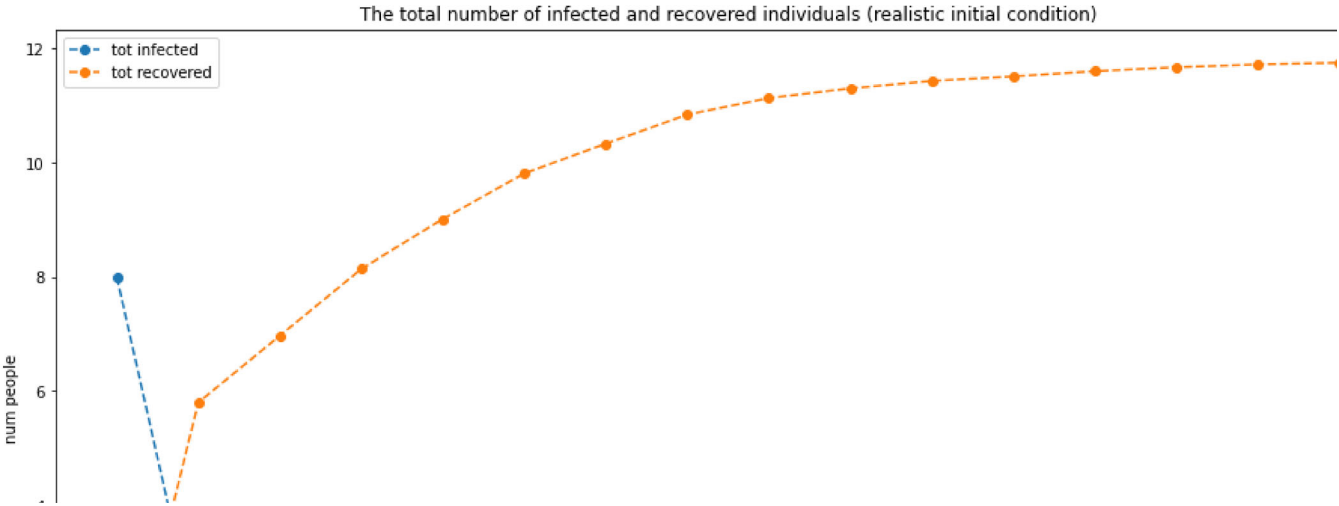
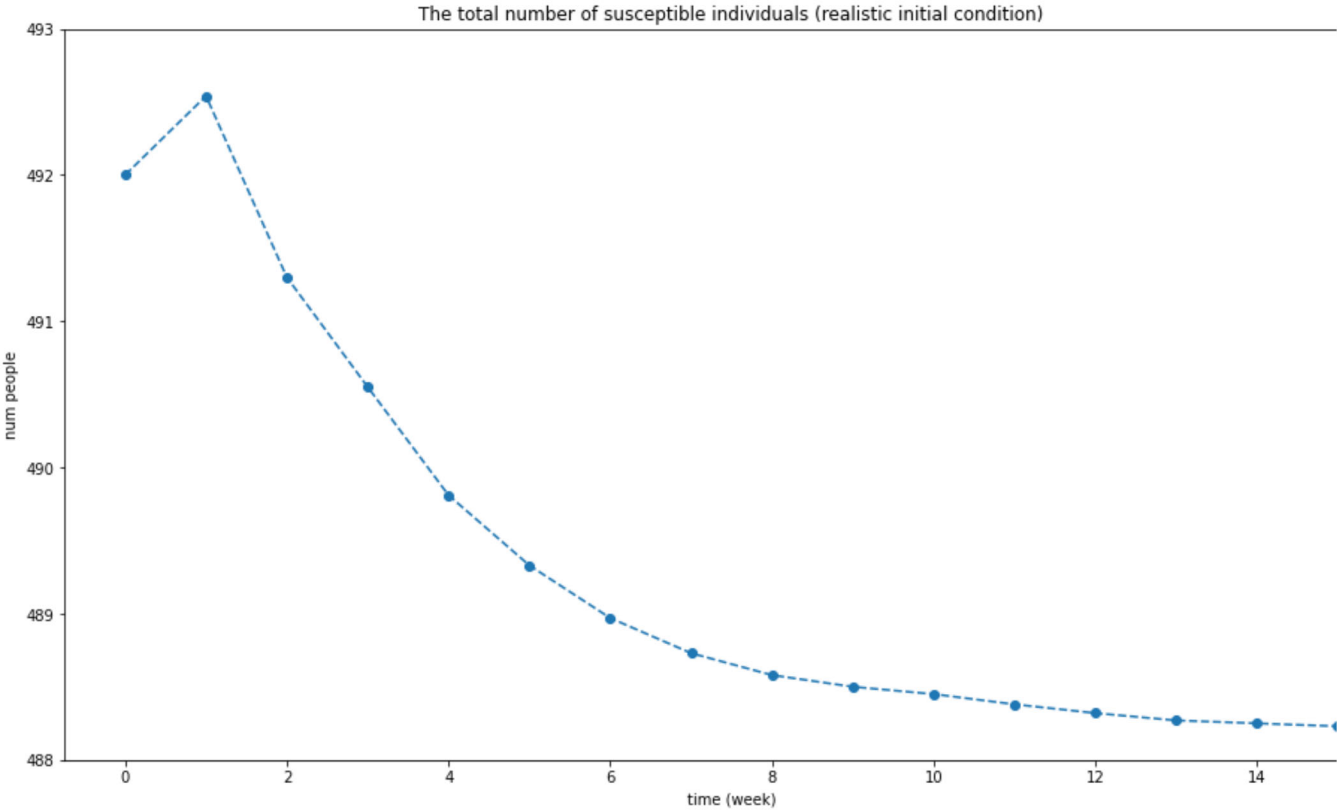
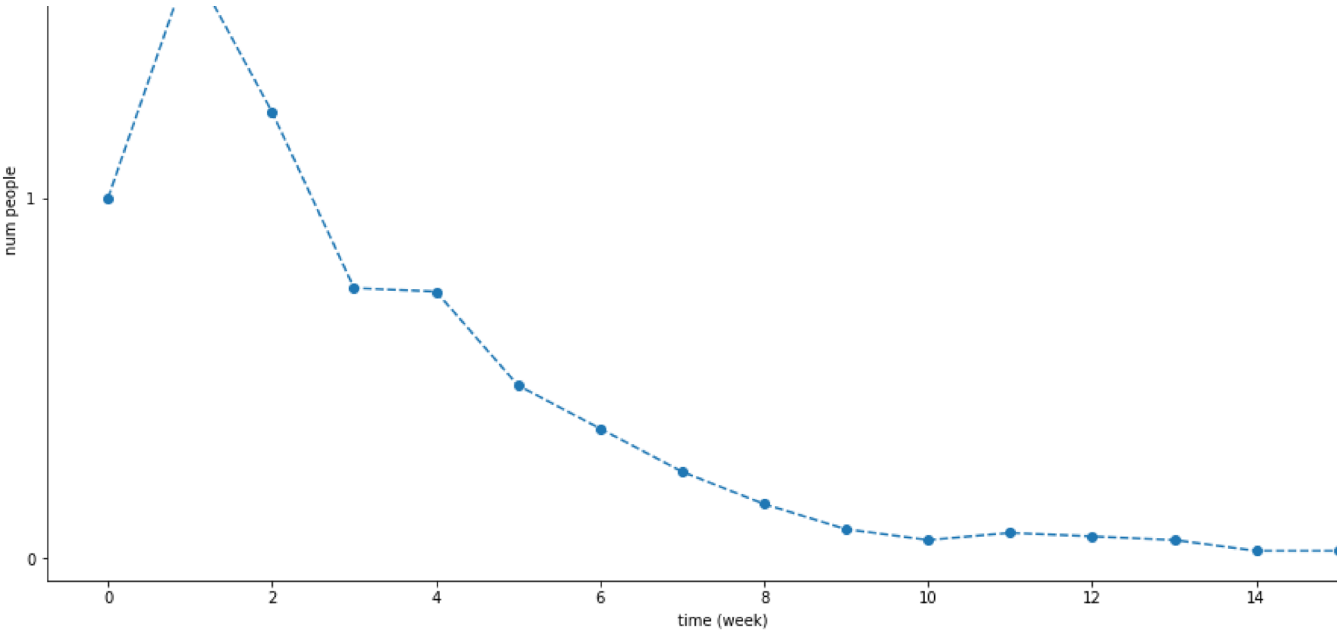
```

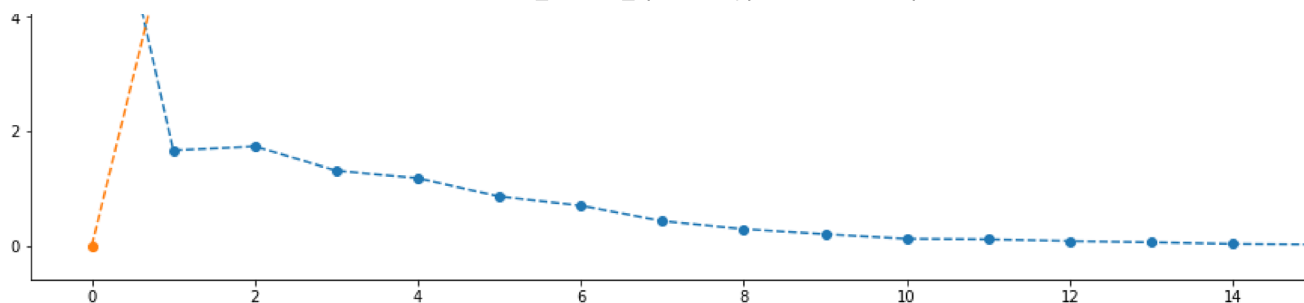
15 new_inf_list = []
16
17 init_inf = [0]*n
18 for i in range(8):
19     init_inf[i] = 1
20
21 N = 100
22
23 for i in range(N):
24     # print("Simulation #{0}".format(i+1))
25     weeks, newly_infected_list = sim_SIR_epidemic(G, n_weeks=15, initial_states=init_inf,
26     susceptibles, infected, recovered = get_sir_count(weeks)
27     # print('newly infected: {0}\n'.format(newly_infected_list))
28     sus_list.append(susceptibles)
29     inf_list.append(infected)
30     rec_list.append(recovered)
31     new_inf_list.append(newly_infected_list)
32 new_inf_list = np.array(new_inf_list)

1 avg_sus = np.mean(np.array(sus_list), axis=0)
2 avg_inf = np.mean(np.array(inf_list), axis=0)
3 avg_rec = np.mean(np.array(rec_list), axis=0)
4 avg_new_inf = np.mean(new_inf_list, axis=0)
5 fig, ax= plt.subplots(figsize=(16,9))
6 ax.plot(avg_new_inf, label='newly infected',linestyle='--', marker='o')
7 ax.set(xlabel='time (week)', ylabel='num people')
8 yint = range(int(min(avg_new_inf)), math.ceil(max(avg_new_inf))+1)
9 ax.set_yticks(yint)
10 ax.set_title("The number of newly infected individuals (realistic initial condition)")
11 plt.savefig("new_inf_real.eps",format='eps')
12 fig, ax= plt.subplots(figsize=(16,9))
13 ax.plot(avg_sus, label='tot avg sus',linestyle='--', marker='o')
14 ax.set(xlabel='time (week)', ylabel='num people')
15 yint = range(int(min(avg_sus)), math.ceil(max(avg_sus))+1)
16 ax.set_yticks(yint)
17 ax.set_title("The total number of susceptible individuals (realistic initial condition)")
18 plt.savefig("tot_sus_real.eps",format='eps')
19 fig, ax= plt.subplots(figsize=(16,9))
20 ax.plot(avg_inf, label='tot infected',linestyle='--', marker='o')
21 ax.plot(avg_rec, label='tot recovered',linestyle='--', marker='o')
22 ax.set(xlabel='time (week)', ylabel='num people')
23 ax.set_title("The total number of infected and recovered individuals (realistic initial co
24 ax.legend(loc='best')
25 plt.savefig("tot_rec_inf_real.eps",format='eps')
26

```







## ▼ Problem 1.2

```

1 def generate_graph(k, n_nodes, verbose = False):
2     '''
3     k: int, average degree
4     n_nodes: int, number of nodes
5     verbose: boolean, for printing information
6     '''
7     t = 1
8     G = nx.random_graphs.complete_graph(k+1) # generate an initial complete graph with k+1
9     # nx.draw_circular(G)
10    while 1:
11        degrees = [deg for (node, deg) in G.degree()] # degree of nodes
12        probab = [d/sum(degrees) for d in degrees]
13        sel_nodes = np.random.choice(list(G.nodes), size=int((k+(-1)**(t*k))/2), p=probab, r

```

```

14      G.add_edges_from([(G.number_of_nodes(), n) for n in sel_nodes]) # add edge from n+
15 #      G.add_edge(len(G.nodes()), sel_nodes)
16      avg_deg = 2*G.number_of_edges() / float(G.number_of_nodes())
17 #      print("Time:", t, "\nAvg degree:", avg_deg)
18
19      t += 1
20      if G.number_of_nodes() == n_nodes:
21          break
22 #      nx.draw_circular(G)
23      if verbose:
24          print('Graph with {} nodes and {} degree is created using preferential attachment.
25      return G
26

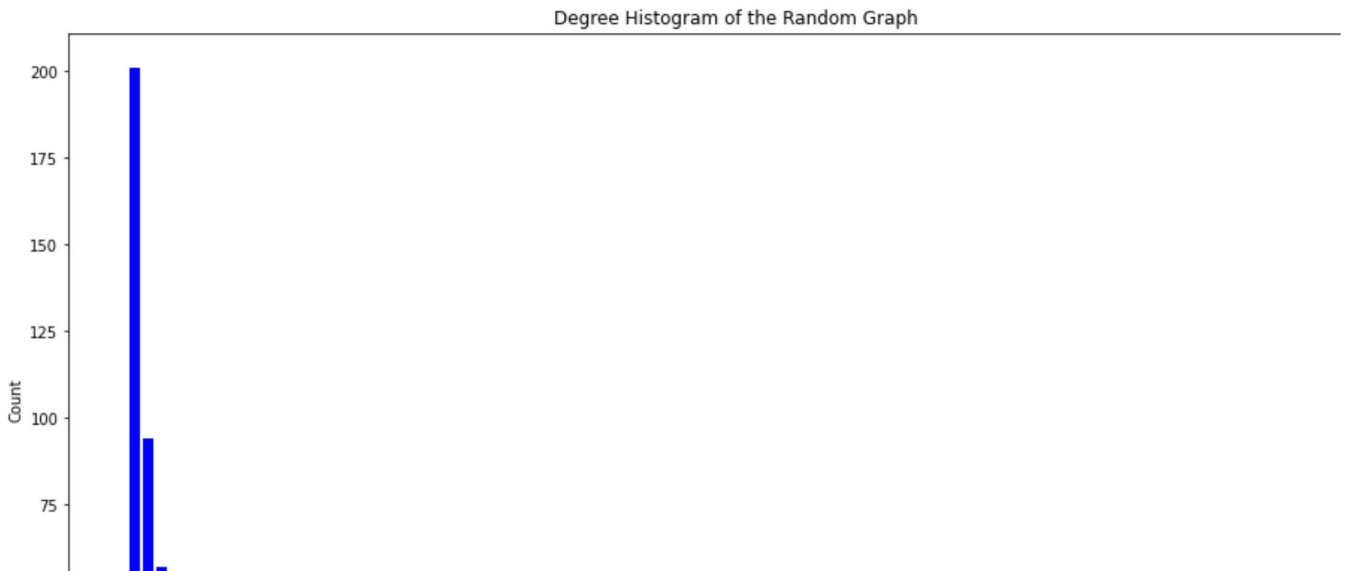
```

```

1 k = 6
2 N = 500
3
4 G = generate_graph(k,N, verbose=False)
5 degree_sequence = sorted([d for n, d in G.degree()], reverse=True) # degree sequence
6 # print "Degree sequence", degree_sequence
7 degreeCount = collections.Counter(degree_sequence)
8 deg, cnt = zip(*degreeCount.items())
9
10 fig, ax = plt.subplots(figsize=(16,9))
11 ax.bar(deg, cnt, width=0.80, color='b')
12
13 ax.set_title("Degree Histogram of the Random Graph")
14 ax.set_ylabel("Count")
15 ax.set_xlabel("Degree")
16 ax.set_xticks([d for d in deg])
17 ax.set_xticklabels(deg)
18 degs = list(dict(G.degree()).values())
19 print("The average degree of nodes:", np.mean(np.array(degs)))
20 plt.savefig("deg_hist.eps",format='eps')
21

```

The average degree of nodes: 6.0



## ▼ Problem 2



## ▼ Random Initial Condition

```

1 sus_list = []
2 inf_list = []
3 rec_list = []
4 new_inf_list = []
5
6 k = 6
7 beta = 0.3
8 rho = 0.7
9 n=500
10 G2 = generate_graph(k,n, verbose=False)
11
12 N = 100
13 for i in range(N):
14     #print("Simulation #{0}".format(i+1))
15     weeks, newly_infected_list = sim_SIR_epidemic(G2, n_weeks=15, initial_states='random',
16     susceptibles, infected, recovered = get_sir_count(weeks)
17     #print('newly infected: {0}\n'.format(newly_infected_list))
18     sus_list.append(susceptibles)
19     inf_list.append(infected)
20     rec_list.append(recovered)
21     new_inf_list.append(newly_infected_list)

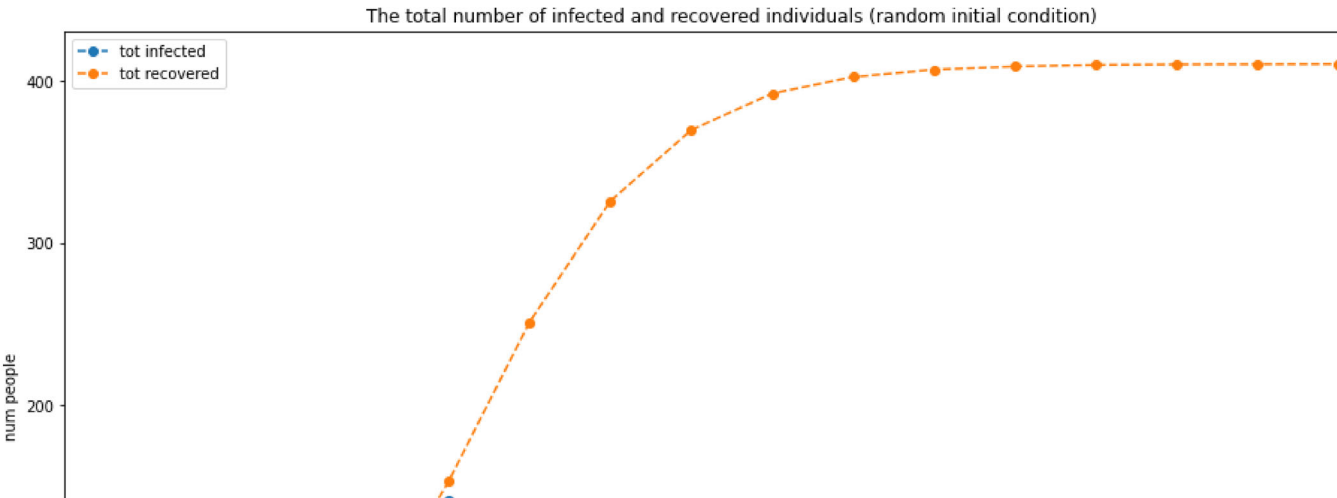
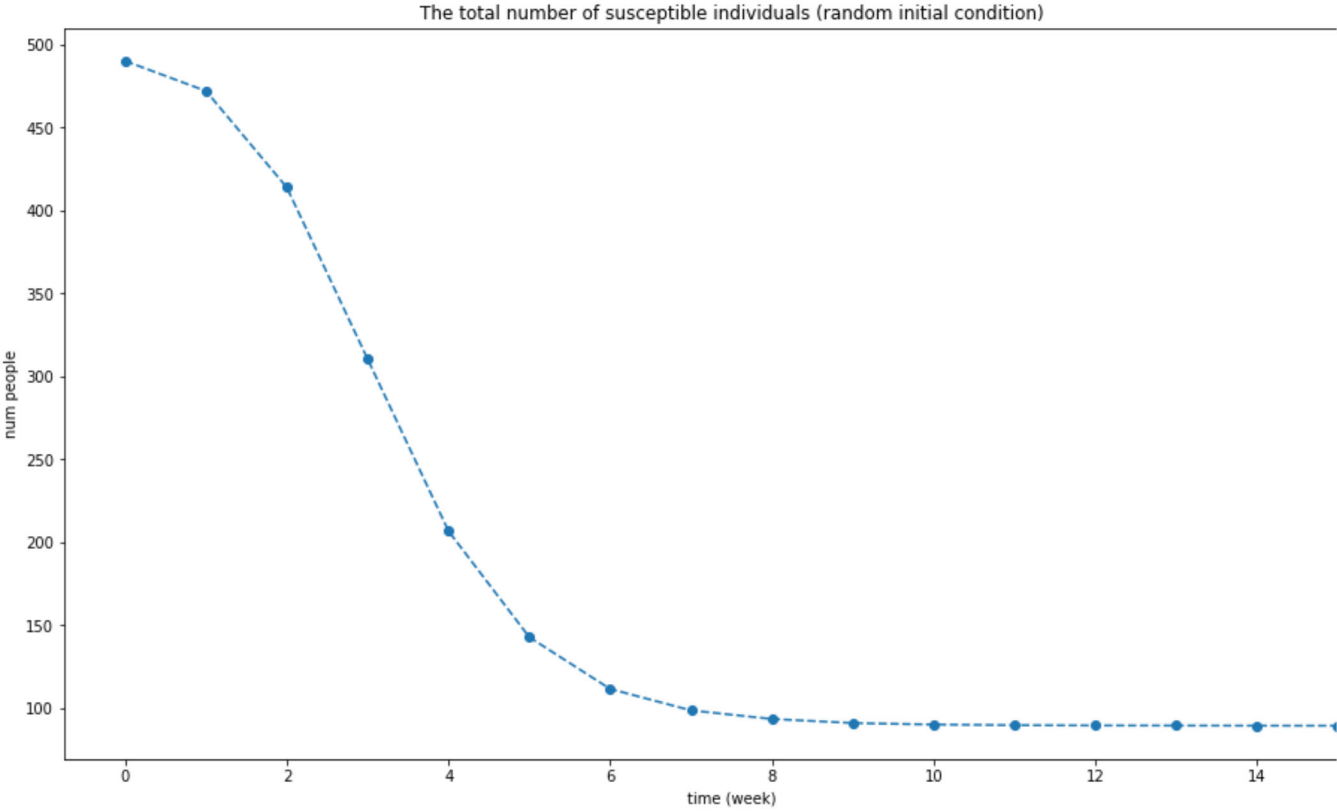
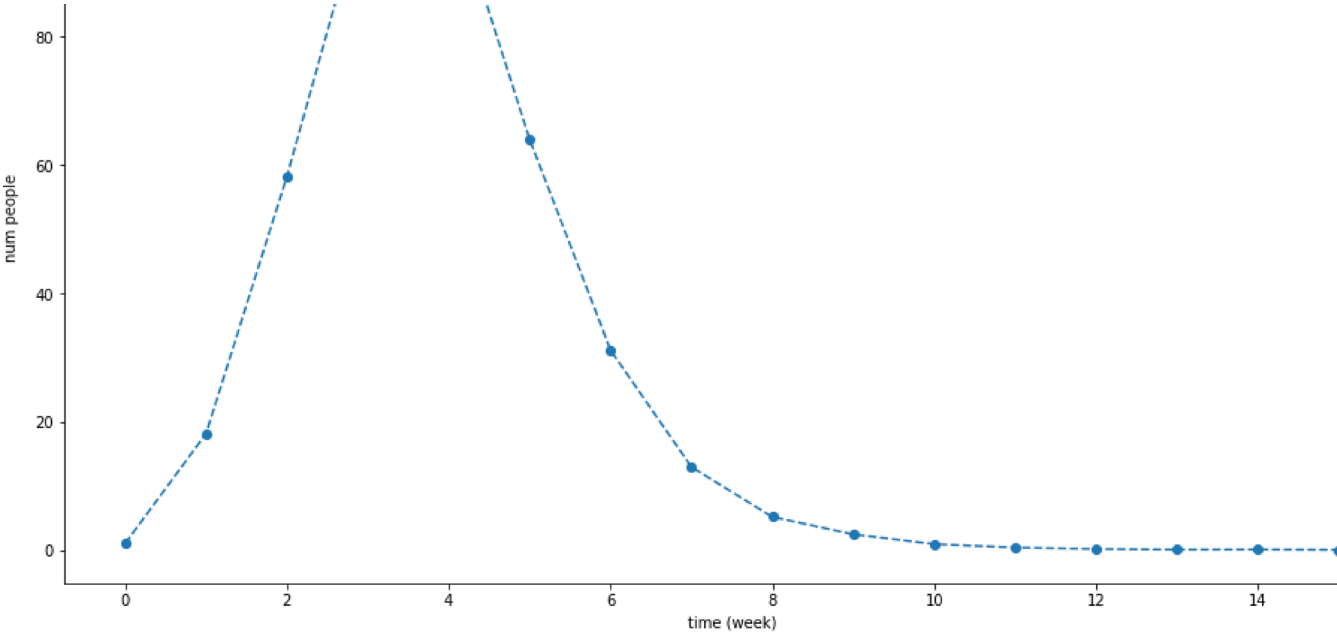
```

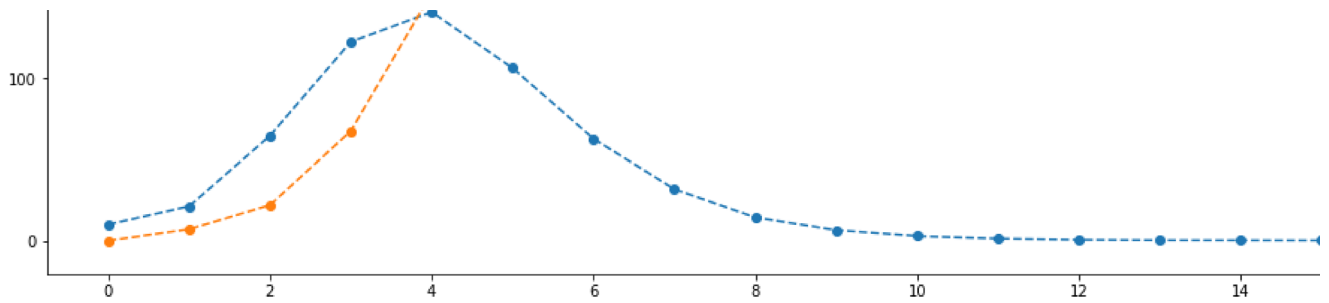
```

1 avg_sus = np.mean(np.array(sus_list), axis=0)
2 avg_inf = np.mean(np.array(inf_list), axis=0)

```

```
3 avg_rec = np.mean(np.array(rec_list), axis=0)
4 avg_new_inf = np.mean(new_inf_list, axis=0)
5 fig, ax= plt.subplots(figsize=(16,9))
6 ax.plot(avg_new_inf, label='newly infected',linestyle='--', marker='o')
7 ax.set(xlabel='time (week)', ylabel='num people')
8 yint = range(int(min(avg_new_inf)), math.ceil(max(avg_new_inf))+1)
9 #ax.set_yticks(yint)
10 ax.set_title("The number of newly infected individuals (random initial condition)")
11 plt.savefig("new_inf_p2.eps",format='eps')
12 fig, ax= plt.subplots(figsize=(16,9))
13 ax.plot(avg_sus, label='tot avg sus',linestyle='--', marker='o')
14 ax.set(xlabel='time (week)', ylabel='num people')
15 yint = range(int(min(avg_sus)), math.ceil(max(avg_sus))+1)
16 #ax.set_yticks(yint)
17 ax.set_title("The total number of susceptible individuals (random initial condition)")
18 plt.savefig("tot_sus_p2.eps",format='eps')
19 fig, ax= plt.subplots(figsize=(16,9))
20 ax.plot(avg_inf, label='tot infected',linestyle='--', marker='o')
21 ax.plot(avg_rec, label='tot recovered',linestyle='--', marker='o')
22 ax.set(xlabel='time (week)', ylabel='num people')
23 ax.set_title("The total number of infected and recovered individuals (random initial condi
24 ax.legend(loc='best')
25 plt.savefig("tot_rec_inf_p2.eps",format='eps')
26
```





## ▼ Realistic Initial Condition

```

1 sus_list = []
2 inf_list = []
3 rec_list = []
4 new_inf_list = []
5
6 k = 6
7 beta = 0.3
8 rho = 0.7
9 n=500
10 G2 = generate_graph(k,n, verbose=False)
11
12 init_state = [0]*n
13
14 inf_init = [0]+list(dict(G2.adjacency())[0].keys())[:7]
15 for i in inf_init:
16     init_state[i] = 1
17
18 N = 100
19 for i in range(N):
20     #print("Simulation #{0}".format(i+1))
21     weeks, newly_infected_list = sim_SIR_epidemic(G2, n_weeks=15, initial_states=init_state)
22     susceptibles, infected, recovered = get_sir_count(weeks)
23     #print('newly infected: {0}\n'.format(newly_infected_list))
24     sus_list.append(susceptibles)
25     inf_list.append(newly_infected_list)
26     rec_list.append(recovered)
27     new_inf_list.append(newly_infected_list)

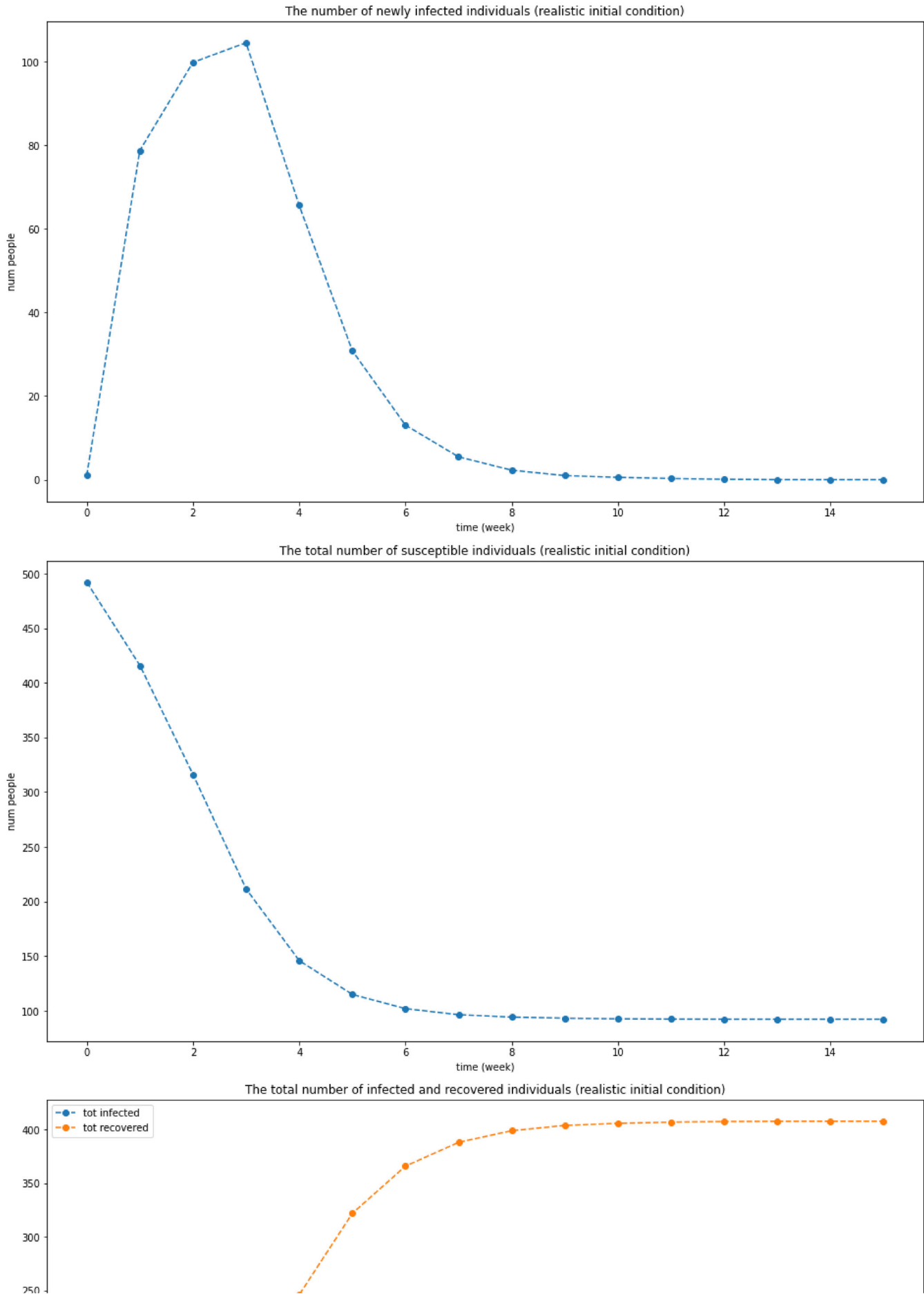
1 avg_sus = np.mean(np.array(sus_list), axis=0)
2 avg_inf = np.mean(np.array(inf_list), axis=0)
3 avg_rec = np.mean(np.array(rec_list), axis=0)
4 avg_new_inf = np.mean(new_inf_list, axis=0)
5 fig, ax= plt.subplots(figsize=(16,9))
6 ax.plot(avg_new_inf, label='newly infected',linestyle='--', marker='o')

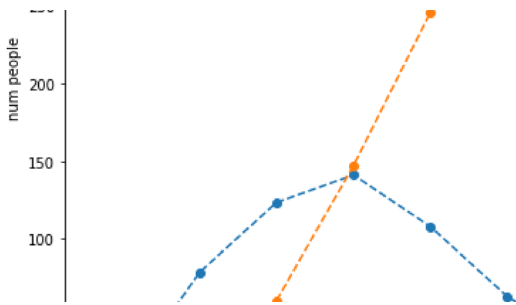
```

```
7 ax.set(xlabel='time (week)', ylabel='num people')
8 yint = range(int(min(avg_new_inf)), math.ceil(max(avg_new_inf))+1)
9 #ax.set_yticks(yint)
10 ax.set_title("The number of newly infected individuals (realistic initial condition)")
11 plt.savefig("new_inf_p2_real.eps",format='eps')
12 fig, ax= plt.subplots(figsize=(16,9))
13 ax.plot(avg_sus, label='tot avg sus',linestyle='--', marker='o')
14 ax.set(xlabel='time (week)', ylabel='num people')
15 yint = range(int(min(avg_sus)), math.ceil(max(avg_sus))+1)
16 #ax.set_yticks(yint)
17 ax.set_title("The total number of susceptible individuals (realistic initial condition)")
18 plt.savefig("tot_sus_p2_real.eps",format='eps')
19 fig, ax= plt.subplots(figsize=(16,9))
20 ax.plot(avg_inf, label='tot infected',linestyle='--', marker='o')
21 ax.plot(avg_rec, label='tot recovered',linestyle='--', marker='o')
22 ax.set(xlabel='time (week)', ylabel='num people')
23 ax.set_title("The total number of infected and recovered individuals (realistic initial co
24 ax.legend(loc='best')
25 plt.savefig("tot_rec_inf_p2_real.eps",format='eps')
26
```



The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will





### ▼ Problem 3

```

1 def sim_SIRV_epidemic(G, initial_infections, vacc, n_weeks = 5, initial_states = 'random',
2     '''
3     G: networkx.classes.graph.Graph, graph
4     initial_infections: int, number of initially infected nodes
5     vacc: list, the fraction of the total population vaccinated during each week
6     n_weeks: int, number of simulation weeks
7     initial_states: list, initial state of agents or default 'random' initialize sick peop
8     random_sick: int, how many sick agents if initial state is randomly generated, default
9     beta: float, the probability that the infection is spread from an infected individual
10    rho: float, the probability that an infected individual will recover, default 0.7
11    vacc: list, vaccinated people percentage
12    until_end: boolean, indicates whether the simulation continues until the end or fi
13    '''
14
15    # Number of people
16    n_agents = len(G)
17
18    A_ = [0,1,2,3] # 0-S, 1-I, 2-R, 3-V
19
20    states = [0 for _ in range(n_agents)] # 0-S, 1-I, 2-R, 3-V
21
22    if initial_states == 'random':
23        choices = np.random.choice(G.nodes(),size = random_sick, replace= False)
24        for i in choices:
25            states[i] = 1
26        initial_states = states.copy()
27
28    old_states = initial_states.copy()
29
30 #    print('Initial states of agents: {}'.format(initial_states))
31
32    W = nx.convert_matrix.to_numpy_matrix(G)
33
34    weeks = []
35    weeks.append(initial_states.copy())
36
37
38    newly_infected_list = [initial_infections]

```

```

39
40 unv_people = list(G.nodes) # unvaccinated people
41 newly_vaccinated_list = [0]
42 vacc2 = [vacc[i+1] - vacc[i] for i in range(len(vacc)-1)]
43
44 for week in range(n_weeks):
45     if not until_end:
46         if not 1 in states:
47             #print('No one is sick! \nGREAT SUCCESS!')
48             break
49
50     n_tobe_vac = int(n_agents*vacc2[week]/100) # number of people to be vaccinated in
51     tobe_vac_list = np.random.choice(unv_people, size = n_tobe_vac, replace = False) #
52     newly_vaccinated_list.append(n_tobe_vac)
53     for agent in tobe_vac_list:
54         old_states[agent] = 3
55         states[agent] = 3
56         unv_people.remove(agent) # remove agent from unvaccinated people's list
57 #     print("States: {}, \nWeek: {}".format(states, week+1))
58
59     newly_infected = 0
60     for agent in range(n_agents):
61         # if S:
62         if old_states[agent] == 0:
63             ngh_states = np.squeeze(np.asarray(W[agent]))*old_states # get neighbors'
64             m = collections.Counter(ngh_states)[1] # count sick neighbors
65
66             if np.random.rand() < (1-(1-beta)**m):
67                 states[agent] = 1 # get sick
68                 newly_infected += 1
69         # if I:
70         elif old_states[agent] == 1:
71             if np.random.rand() < rho:
72                 states[agent] = 2 # get recovered
73
74     weeks.append(states.copy())
75     newly_infected_list.append(newly_infected)
76
77     old_states = states.copy()
78 #     print(old_states)
79     return weeks, newly_infected_list, newly_vaccinated_list
80     # weeks = weeks[:5]

1 def get_sirv_count(weeks):
2     '''
3     weeks: list, list of states through weeks
4     '''
5     susceptibles = [collections.Counter(np.asarray(w))[0] for w in weeks]
6     infected = [collections.Counter(np.asarray(w))[1] for w in weeks]
7     recovered = [collections.Counter(np.asarray(w))[2] for w in weeks]
8     vaccinated = [collections.Counter(np.asarray(w))[3] for w in weeks]

```

```

9     #print('susceptibles: {}, \ninfected: {}, \nrecovered: {}, \nvaccinated: {}'.format(su
10     return susceptibles, infected, recovered, vaccinated

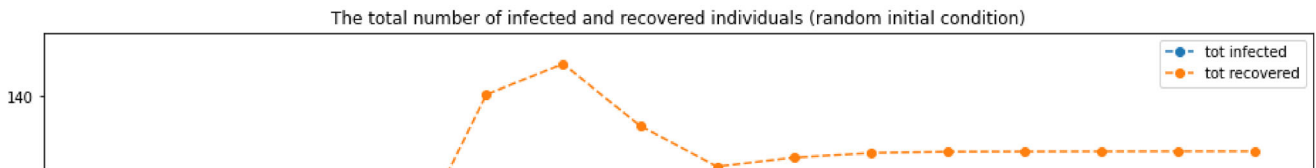
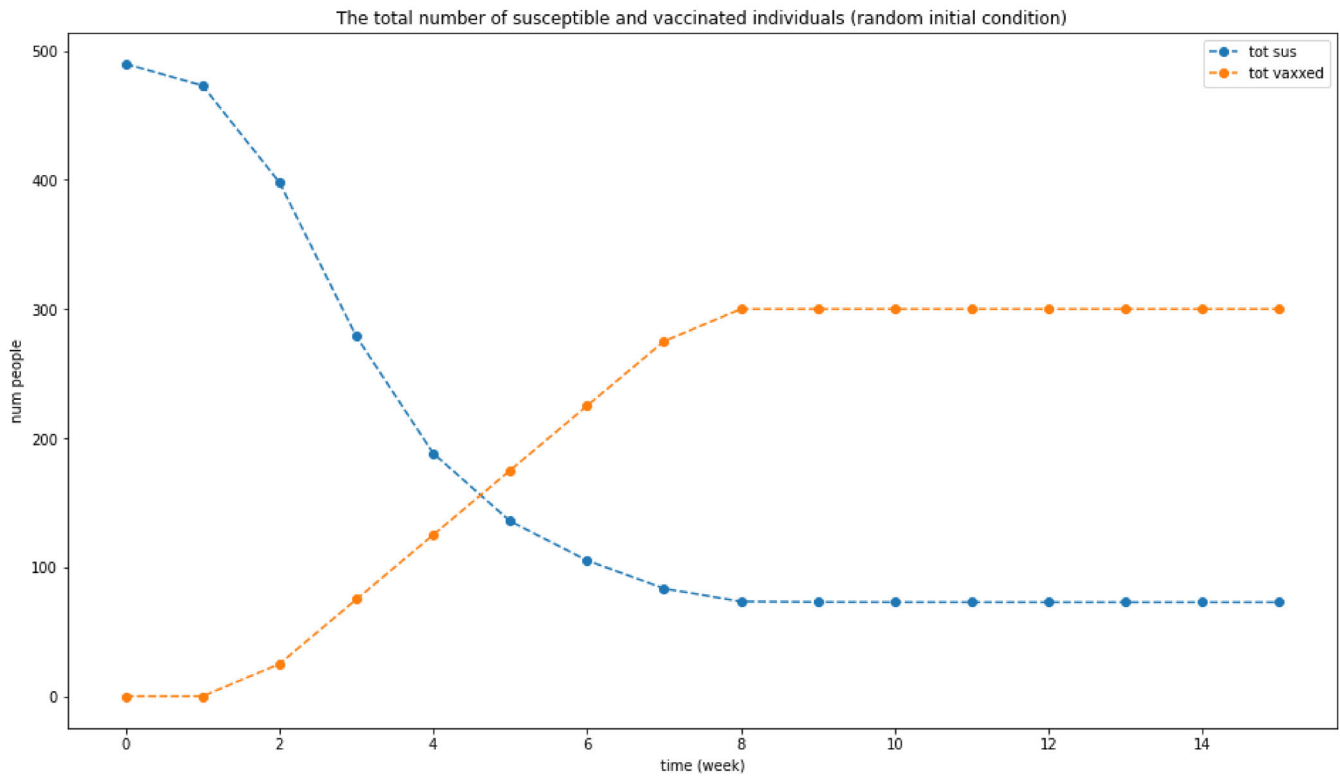
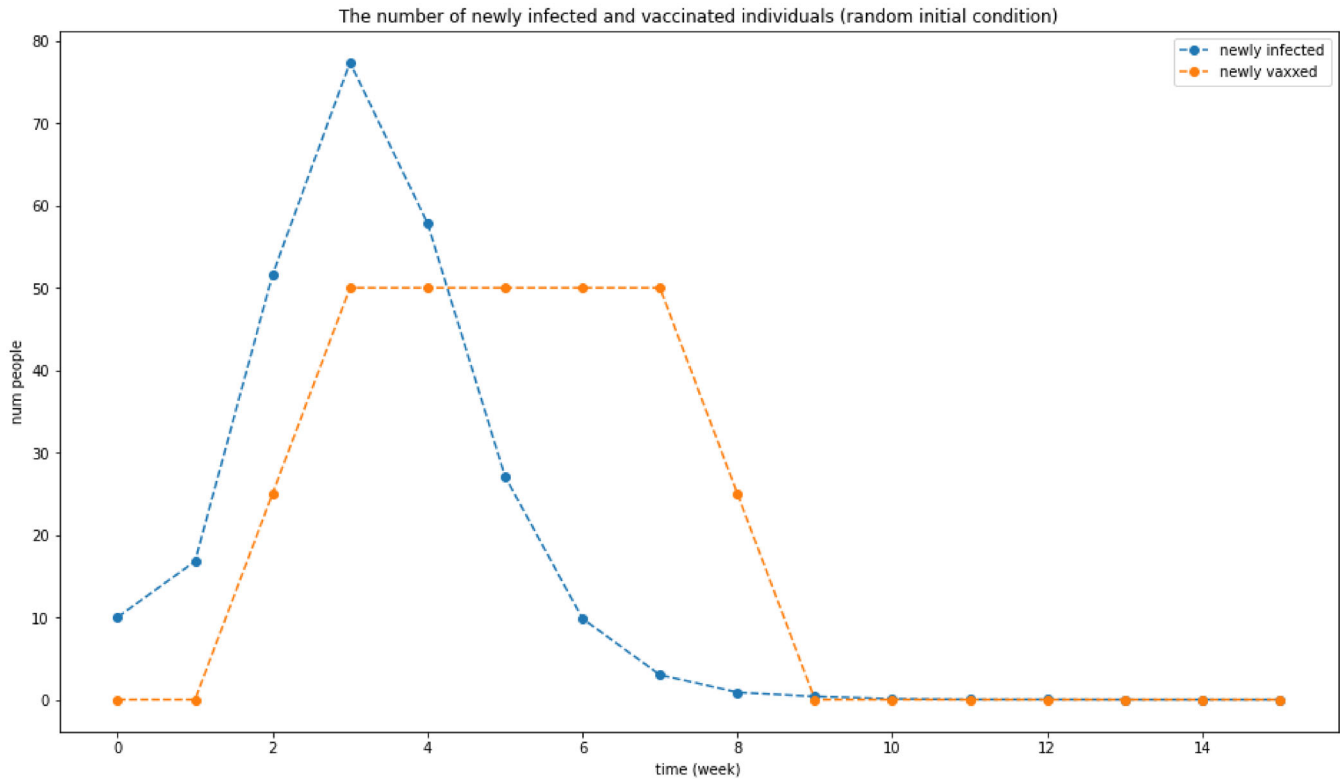
1 sus_list = []
2 inf_list = []
3 rec_list = []
4 vac_list = []
5 new_inf_list = []
6 new_vac_list = []
7 k = 6
8 n = 500
9 beta = 0.3
10 rho = 0.7
11 n_weeks = 15
12
13 vacc_t = [0, 5, 15, 25, 35, 45, 55, 60, 60, 60, 60, 60, 60, 60]
14 vacc = [0]
15 vacc.extend(vacc_t)
16
17 G3 = generate_graph(k,n, verbose=False)
18 N = 100
19 for i in range(N):
20     #print("Simulation #{}".format(i+1))
21     weeks, newly_infected_list, newly_vaccinated_list = sim_SIRV_epidemic(G3, 10, vacc, n_
22     susceptibles, infected, recovered, vaccinated = get_sirv_count(weeks)
23     #print('newly infected: {}'.format(newly_infected_list))
24     #print('newly vaxed: {}'.format(newly_vaccinated_list))
25
26     sus_list.append(susceptibles)
27     inf_list.append(infected)
28     rec_list.append(recovered)
29     vac_list.append(vaccinated)
30
31     new_inf_list.append(newly_infected_list)
32     new_vac_list.append(newly_vaccinated_list)

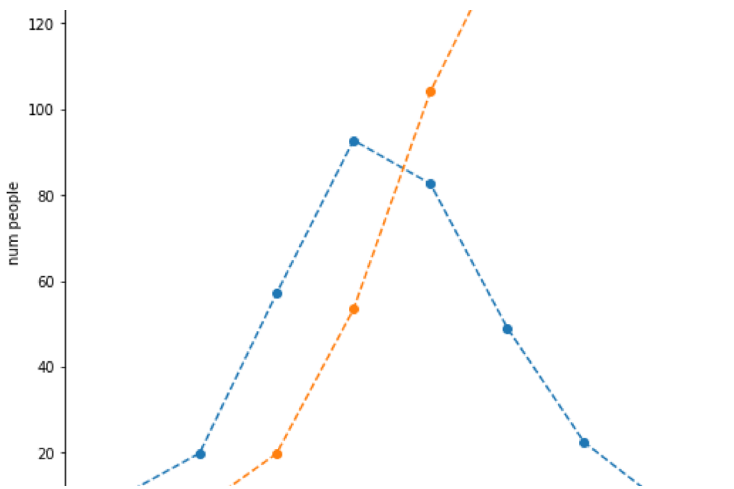
1 avg_new_vac = np.mean(np.array(new_vac_list), axis=0)
2 avg_vac = np.mean(np.array(vac_list), axis=0)
3 avg_sus = np.mean(np.array(sus_list), axis=0)
4 avg_inf = np.mean(np.array(inf_list), axis=0)
5 avg_rec = np.mean(np.array(rec_list), axis=0)
6 avg_new_inf = np.mean(new_inf_list, axis=0)
7 fig, ax= plt.subplots(figsize=(16,9))
8 ax.plot(avg_new_inf, label='newly infected',linestyle='--', marker='o')
9 ax.plot(avg_new_vac, label='newly vaxxed',linestyle='--', marker='o')
10
11 ax.set(xlabel='time (week)', ylabel='num people')
12 yint = range(int(min(avg_new_inf)), math.ceil(max(avg_new_inf))+1)
13 #ax.set_yticks(yint)

```

```
14 ax.legend(loc='best')
15
16 ax.set_title("The number of newly infected and vaccinated individuals (random initial cond
17 plt.savefig("new_inf_p3.eps",format='eps')
18 fig, ax= plt.subplots(figsize=(16,9))
19 ax.plot(avg_sus, label='tot sus',linestyle='--', marker='o')
20 ax.plot(avg_vac, label='tot vaxxed',linestyle='--', marker='o')
21 ax.legend(loc='best')
22 ax.set(xlabel='time (week)', ylabel='num people')
23 yint = range(int(min(avg_sus)), math.ceil(max(avg_sus))+1)
24 #ax.set_yticks(yint)
25 ax.set_title("The total number of susceptible and vaccinated individuals (random initial c
26 plt.savefig("tot_sus_p3.eps",format='eps')
27 fig, ax= plt.subplots(figsize=(16,9))
28 ax.plot(range(n_weeks+1),avg_inf, label='tot infected', linestyle='--', marker='o')
29 ax.plot(range(n_weeks+1),avg_rec, label='tot recovered', linestyle='--', marker='o')
30 ax.set(xlabel='time (week)', ylabel='num people')
31 ax.set_title("The total number of infected and recovered individuals (random initial condi
32 ax.legend(loc='best')
33 plt.savefig("tot_rec_inf_p3.eps",format='eps')
34
```

The PostScript backend does not support transparency; partially transparent artists will:  
The PostScript backend does not support transparency; partially transparent artists will:  
The PostScript backend does not support transparency; partially transparent artists will:  
The PostScript backend does not support transparency; partially transparent artists will:  
The PostScript backend does not support transparency; partially transparent artists will:  
The PostScript backend does not support transparency; partially transparent artists will:





## ▼ Problem 4

```

1 vacc = [5, 9, 16, 24, 32, 40, 47, 54, 59, 60, 60, 60, 60, 60, 60]
2
3 # we assume that during the week 0 nobody is vaccinated and
4 # add 0'th week in order to calculate percentage of newly vaccinated people during week 1
5 I0 = [1, 1, 3, 5, 9, 17, 32, 32, 17, 5, 2, 1, 0, 0, 0]
6 n_agents = 934
7
8 k0 = 10
9 beta0 = 0.3
10 rho0 = 0.6
11
12 dk = 1
13 dbeta = 0.1
14 drho = 0.1

```

```

1 def RMSE(I, I0):
2     return np.sqrt(np.mean((np.array(I)-np.array(I0))**2))

```

```

1 def find_pand_params(n_agents = 934, k0 = 10, beta0 = 0.3, rho0 = 0.6,
2                       dk_ = 1, dbeta_ = 0.1, drho_ = 0.1, p=0.5, graph_type = 'pref_attach'
3                       RMSE_list = []
4                       N = 10 # number of simulations
5
6                       k_opt = k0
7                       beta_opt = beta0
8                       rho_opt = rho0
9
10                      dk = dk_
11                      dbeta = dbeta_
12                      drho = drho_
13                      min_RMSE = np.inf
14

```

```
15     opt_comb = [k_opt, beta_opt, rho_opt]
16     optimization_option = kwargs.get('opt_option', None)
17
18
19     print("Initial hyperparameter combination: {}".format(opt_comb))
20
21     iteration = 1
22     while True:
23         print("Iteration {}".format(iteration))
24
25         #print("Current best combination: {}".format(opt_comb))
26         k = [k_opt, k_opt-dk, k_opt+dk]
27         beta = [beta_opt, np.round(beta_opt-dbeta,2), np.round(beta_opt+dbeta,2)]
28         rho = [rho_opt, np.round(rho_opt-drho, 2), np.round(rho_opt+drho, 2)]
29
30         combs = []
31         for i in range(len(k)):
32             for j in range(len(beta)):
33                 for l in range(len(rho)):
34                     combs.append([k[i], beta[j], rho[l]])
35
36         for comb in combs:
37             k = comb[0]
38             beta = comb[1]
39             rho = comb[2]
40
41
42
43
44         if graph_type == 'pref_attach':
45             G4 = generate_graph(k, n_agents)
46         else:
47             raise NotImplementedError("The graph model is not valid!")
48
49
50         new_inf_list = []
51         for i in range(N):
52             _, newly_infected_list, _ = sim_SIRV_epidemic(G4, 1, vacc,
53                                                         n_weeks=15,
54                                                         initial_states='random',
55                                                         random_sick=1,
56                                                         beta=beta,
57                                                         rho=rho)
58             #print('newly infected: {}'.format(newly_infected_list))
59
60             new_inf_list.append(newly_infected_list.copy())
61
62         I_pred = np.mean(np.array(new_inf_list), axis=0)
63         #print(I_pred)
64         RMSE_c = RMSE(I_pred, I0)
65         if RMSE_c < min_RMSE:
```



```

66         best_comb = comb
67         min_RMSE = RMSE_c
68         print('The Current Comb.: {}, RMSE: {}'.format(comb, RMSE_c))
69         RMSE_list.append(min_RMSE)
70         print('\nThe Current Best Comb.: {}, RMSE: {}'.format(best_comb, min_RMSE))
71
72
73     #     print('RMSEs:', RMSE_list)
74
75     if (best_comb == opt_comb):
76         print('Convergence reached!')
77         break
78     opt_comb = best_comb.copy()
79     if optimization_option == None:
80         k_opt = best_comb[0]
81         beta_opt = best_comb[1]
82         rho_opt = best_comb[2]
83     elif optimization_option == 'halving':
84         if beta_opt == best_comb[1]:
85             dbeta /= 2
86         if rho_opt == best_comb[2]:
87             drho /= 2
88         k_opt = best_comb[0]
89         beta_opt = best_comb[1]
90         rho_opt = best_comb[2]
91     iteration += 1
92     return RMSE_list, opt_comb
93
94

```

```
1 RMSE_list, opt_comb = find_pand_params(opt_option='halving')
```

```

Initial hyperparameter combination: [10, 0.3, 0.6]
Iteration 1
The Current Comb.: [10, 0.3, 0.6], RMSE: 37.26508083984255
The Current Comb.: [10, 0.3, 0.5], RMSE: 32.59538886100302
The Current Comb.: [10, 0.3, 0.7], RMSE: 35.213278745382404
The Current Comb.: [10, 0.2, 0.6], RMSE: 18.21098569545317
The Current Comb.: [10, 0.2, 0.5], RMSE: 9.066593902894295
The Current Comb.: [10, 0.2, 0.7], RMSE: 7.731995214690708
The Current Comb.: [10, 0.4, 0.6], RMSE: 70.04001534837067
The Current Comb.: [10, 0.4, 0.5], RMSE: 65.84742781612658
The Current Comb.: [10, 0.4, 0.7], RMSE: 57.61515100214526
The Current Comb.: [9, 0.3, 0.6], RMSE: 34.561702070355274
The Current Comb.: [9, 0.3, 0.5], RMSE: 43.049767711336145
The Current Comb.: [9, 0.3, 0.7], RMSE: 33.78029196143811
The Current Comb.: [9, 0.2, 0.6], RMSE: 11.842138742642733
The Current Comb.: [9, 0.2, 0.5], RMSE: 4.74453633140268
The Current Comb.: [9, 0.2, 0.7], RMSE: 9.10078980089091
The Current Comb.: [9, 0.4, 0.6], RMSE: 40.759615429981665
The Current Comb.: [9, 0.4, 0.5], RMSE: 60.97862842832725
The Current Comb.: [9, 0.4, 0.7], RMSE: 46.63494532000654

```

```

The Current Comb.: [11, 0.3, 0.6], RMSE: 37.85368211944513
The Current Comb.: [11, 0.3, 0.5], RMSE: 59.38258898869264
The Current Comb.: [11, 0.3, 0.7], RMSE: 49.6093615157462
The Current Comb.: [11, 0.2, 0.6], RMSE: 13.64388599336714
The Current Comb.: [11, 0.2, 0.5], RMSE: 21.714554450874648
The Current Comb.: [11, 0.2, 0.7], RMSE: 15.683689776324957
The Current Comb.: [11, 0.4, 0.6], RMSE: 65.86649850265309
The Current Comb.: [11, 0.4, 0.5], RMSE: 56.3484915503512
The Current Comb.: [11, 0.4, 0.7], RMSE: 67.35662086981502

```

```

The Current Best Comb.: [9, 0.2, 0.5], RMSE: 4.74453633140268
Iteration 2

```

```

The Current Comb.: [9, 0.2, 0.5], RMSE: 6.748657273858259
The Current Comb.: [9, 0.2, 0.4], RMSE: 18.843632876916278
The Current Comb.: [9, 0.2, 0.6], RMSE: 8.252386018600923
The Current Comb.: [9, 0.1, 0.5], RMSE: 11.68647722797593
The Current Comb.: [9, 0.1, 0.4], RMSE: 11.295519023046262
The Current Comb.: [9, 0.1, 0.6], RMSE: 11.524484804102958
The Current Comb.: [9, 0.3, 0.5], RMSE: 41.775875215248334
The Current Comb.: [9, 0.3, 0.4], RMSE: 41.72259579652254
The Current Comb.: [9, 0.3, 0.6], RMSE: 30.444118890189614
The Current Comb.: [8, 0.2, 0.5], RMSE: 11.848945100725212
The Current Comb.: [8, 0.2, 0.4], RMSE: 7.212445840351247
The Current Comb.: [8, 0.2, 0.6], RMSE: 8.097877808413756
The Current Comb.: [8, 0.1, 0.5], RMSE: 10.779668362245658
The Current Comb.: [8, 0.1, 0.4], RMSE: 10.685124004895778
The Current Comb.: [8, 0.1, 0.6], RMSE: 11.889780275513925
The Current Comb.: [8, 0.3, 0.5], RMSE: 12.061819929015687
The Current Comb.: [8, 0.3, 0.4], RMSE: 29.994676611025497
The Current Comb.: [8, 0.3, 0.6], RMSE: 27.53270782178898
The Current Comb.: [10, 0.2, 0.5], RMSE: 6.362045661577728
The Current Comb.: [10, 0.2, 0.4], RMSE: 24.98893505133822
The Current Comb.: [10, 0.2, 0.6], RMSE: 6.287139651701719
The Current Comb.: [10, 0.1, 0.5], RMSE: 11.892907550300725
The Current Comb.: [10, 0.1, 0.4], RMSE: 6.76248844730991
The Current Comb.: [10, 0.1, 0.6], RMSE: 10.840779492268995
The Current Comb.: [10, 0.3, 0.5], RMSE: 33.34647657549445
The Current Comb.: [10, 0.3, 0.4], RMSE: 44.82474205168391

```

```

1 k = opt_comb[0]
2 beta = opt_comb[1]
3 rho = opt_comb[2]
4
5 G4 = generate_graph(k, n_agents, verbose=False)
6 N = 100
7 sus_list = []
8 inf_list = []
9 rec_list = []
10 vac_list = []
11 new_inf_list = []
12 new_vac_list = []
13 for n in range(N):
14     #print("Simulation #{0}".format(n+1))
15     weeks, newly_infected_list, newly_vaccinated_list = sim_SIRV_epidemic(G4, 1, vacc,

```

```

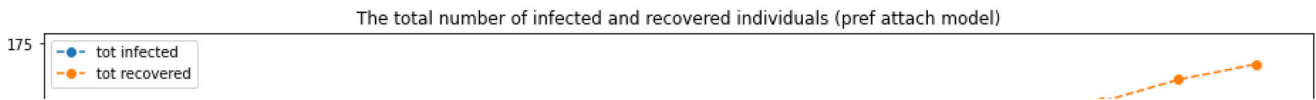
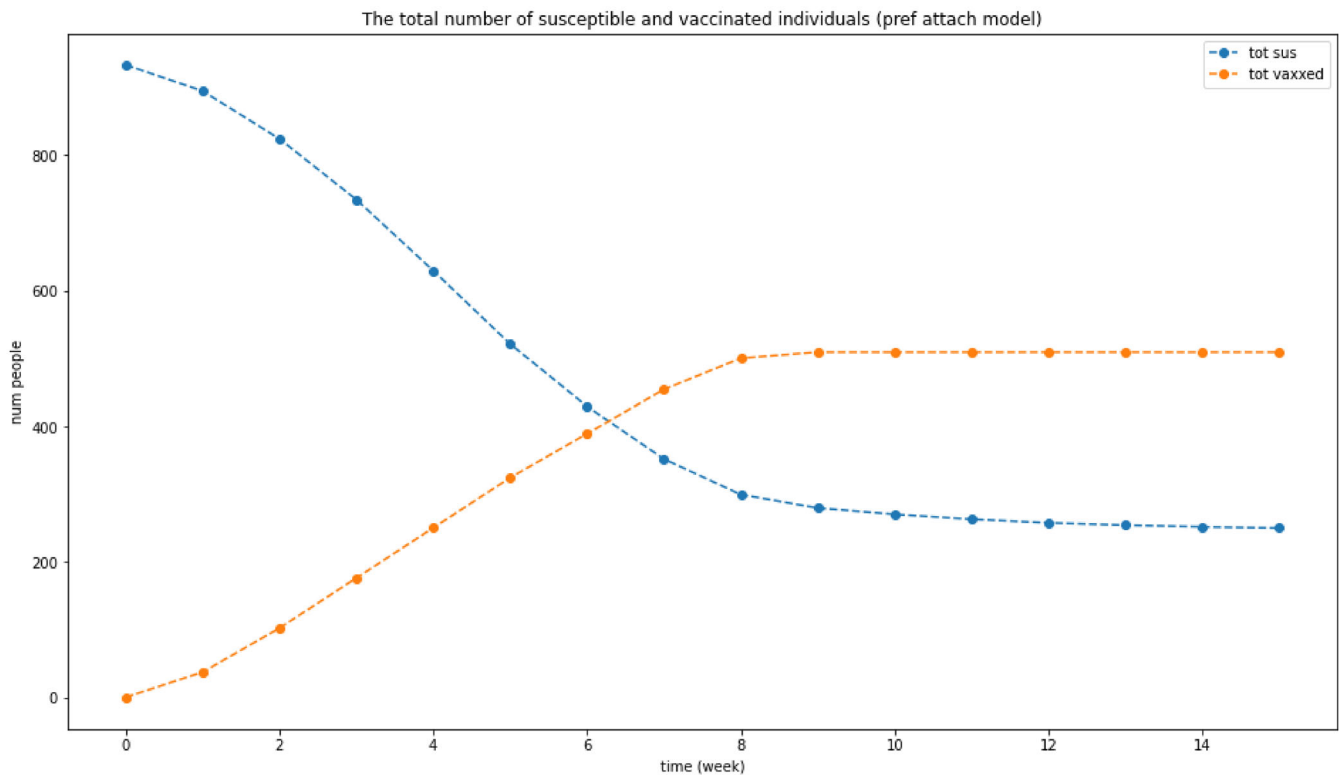
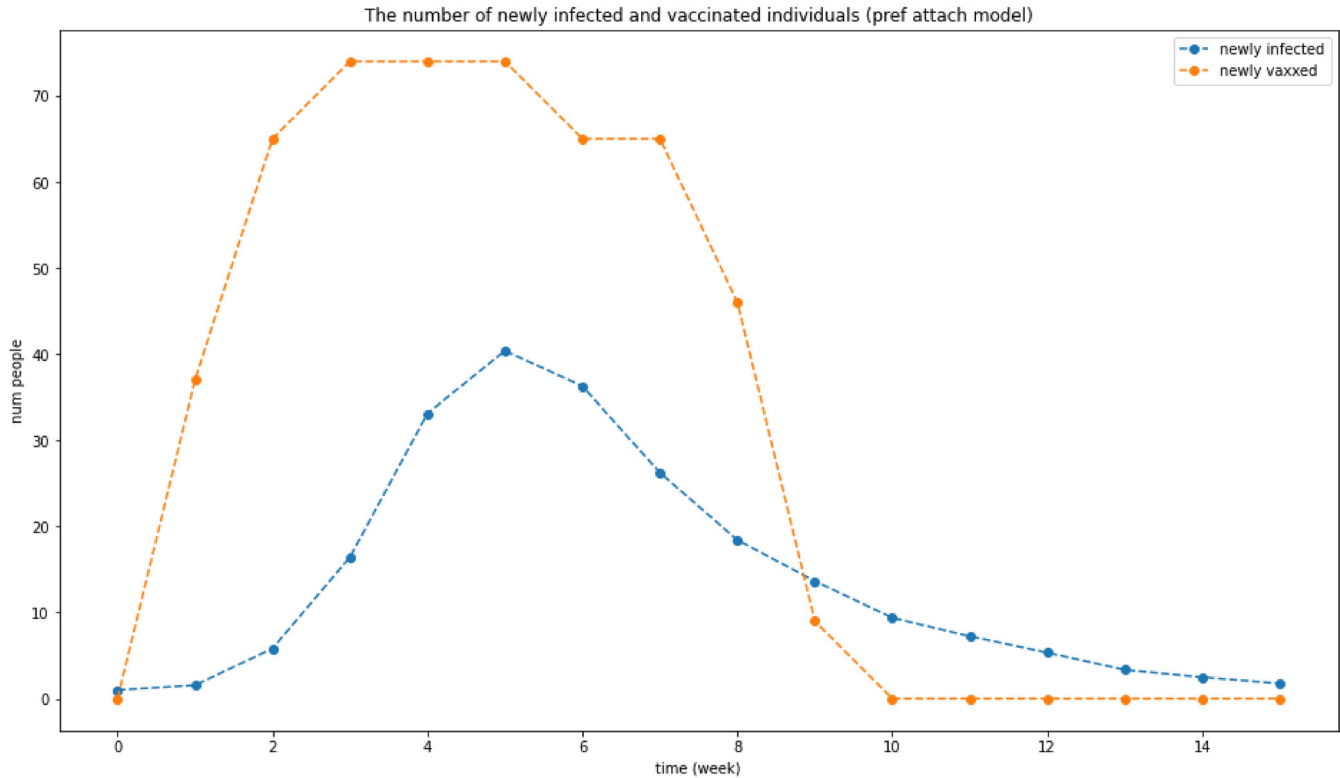
16                                     n_weeks=15,
17                                     initial_states='
18                                     random_sick=1,
19                                     beta=beta,
20                                     rho=rho)
21     susceptibles, infected, recovered, vaccinated = get_sirv_count(weeks)
22     #print('newly infected: {}'.format(newly_infected_list))
23 #     print('newly vaxed: {}'.format(newly_vaccinated_list))
24
25     sus_list.append(susceptibles)
26     inf_list.append(infected)
27     rec_list.append(recovered)
28     vac_list.append(vaccinated)
29
30     new_inf_list.append(newly_infected_list)
31     new_vac_list.append(newly_vaccinated_list)

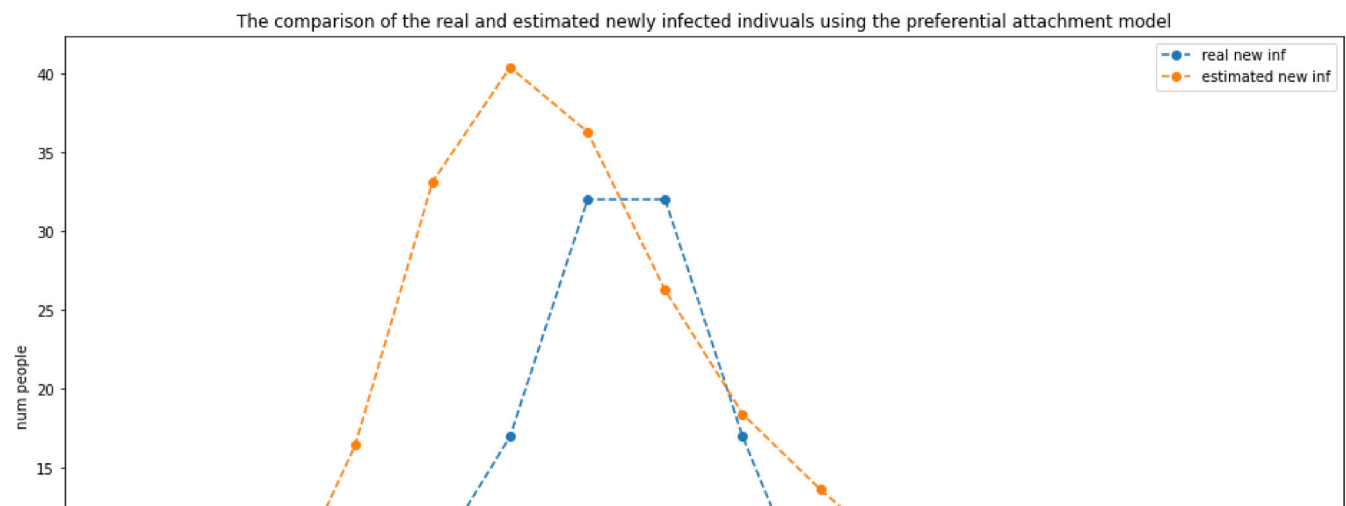
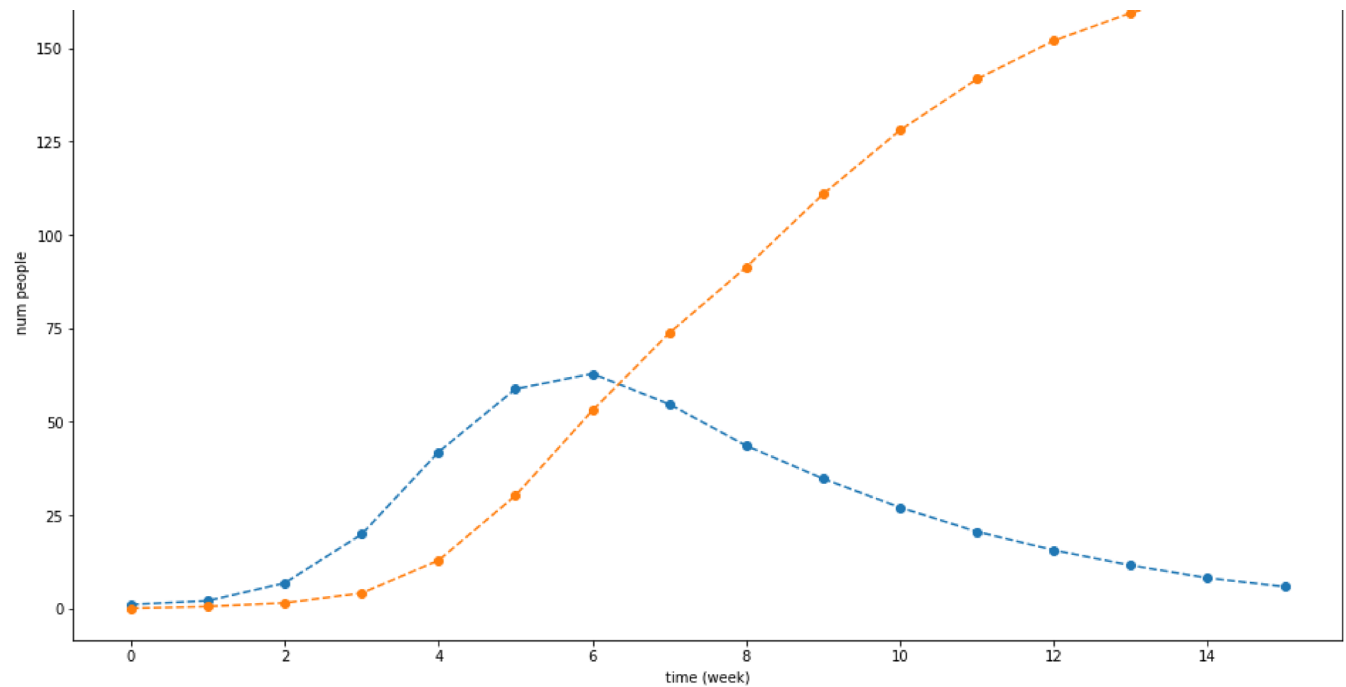
1 avg_new_vac = np.mean(np.array(new_vac_list), axis=0)
2 avg_vac = np.mean(np.array(vac_list), axis=0)
3 avg_sus = np.mean(np.array(sus_list), axis=0)
4 avg_inf = np.mean(np.array(inf_list), axis=0)
5 avg_rec = np.mean(np.array(rec_list), axis=0)
6 avg_new_inf = np.mean(new_inf_list, axis=0)
7 fig, ax= plt.subplots(figsize=(16,9))
8 ax.plot(avg_new_inf, label='newly infected',linestyle='--', marker='o')
9 ax.plot(avg_new_vac, label='newly vaxxed',linestyle='--', marker='o')
10 ax.legend(loc='best')
11
12 ax.set(xlabel='time (week)', ylabel='num people')
13 yint = range(int(min(avg_new_inf)), math.ceil(max(avg_new_inf))+1)
14 #ax.set_yticks(yint)
15 ax.set_title("The number of newly infected and vaccinated individuals (pref attach model)")
16 plt.savefig("new_inf_p4.eps",format='eps')
17
18 fig, ax= plt.subplots(figsize=(16,9))
19 ax.plot(avg_sus, label='tot sus',linestyle='--', marker='o')
20 ax.plot(avg_vac, label='tot vaxxed',linestyle='--', marker='o')
21 ax.legend(loc='best')
22 ax.set(xlabel='time (week)', ylabel='num people')
23 yint = range(int(min(avg_sus)), math.ceil(max(avg_sus))+1)
24 #ax.set_yticks(yint)
25 ax.legend(loc='best')
26 ax.set_title("The total number of susceptible and vaccinated individuals (pref attach mode
27 plt.savefig("tot_sus_p4.eps",format='eps')
28
29 fig, ax= plt.subplots(figsize=(16,9))
30 ax.plot(avg_inf, label='tot infected',linestyle='--', marker='o')
31 ax.plot(avg_rec, label='tot recovered',linestyle='--', marker='o')
32 ax.set(xlabel='time (week)', ylabel='num people')
33 ax.set_title("The total number of infected and recovered individuals (pref attach model)")
34 ax.legend(loc='best')

```

```
35 plt.savefig("tot_rec_inf_p4.eps",format='eps')
36
37 fig, ax= plt.subplots(figsize=(16,9))
38 ax.plot(I0, label='real new inf', linestyle='--', marker='o')
39 ax.plot(avg_new_inf, label='estimated new inf', linestyle='--', marker='o')
40 ax.set(xlabel='time (week)', ylabel='num people')
41 ax.set_title("The comparison of the real and estimated newly infected individuals using the")
42 ax.legend(loc='best')
43 plt.savefig("real_data_comp_p4.eps",format='eps')
```

The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will  
The PostScript backend does not support transparency; partially transparent artists will





## ▼ Problem 5

```

1 def watts_strogats_model_sim(n_agents = 934, k0 = 10, beta0 = 0.3, rho0 = 0.6,
2                               dk_ = 1, dbeta_ = 0.1, drho_ = 0.1, restart_iters=2, **kwargs):
3     RMSE_list = []
4     N = 10 # number of simulations
5
6     k_opt = k0
7     beta_opt = beta0
8     rho_opt = rho0
9
10    dk = dk_
11    dbeta = dbeta_
12    drho = drho_
13    min_RMSE = np.inf
14
15    opt_comb = [k_opt, beta_opt, rho_opt]

```