Option 1: Virus Propagation on Static Networks

1. For the SIS (susceptible, infected, susceptible) Virus Propagation Model (VPM), with transmission probability $\beta = \beta_1$, and healing probability $\delta = \delta_1$, calculate the effective strength (s) of the virus on the static contact network provided (*static.network*). See supplementary material provided for details on the SIS VPM and on how to calculate the effective strength of a virus. Answer the following questions:

a. Will the infection spread across the network (i.e., result on an epidemic), or will it die quickly?

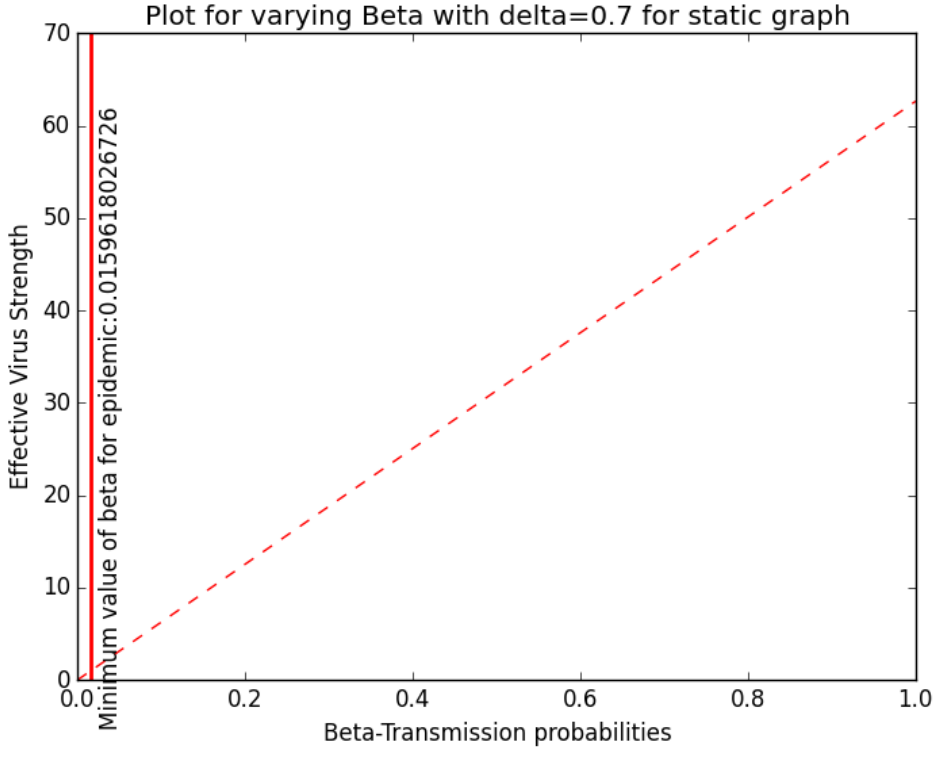
Graph Type	static.network	largest Eigen Value	Effective Virus Strength
β	0.2		
δ	0.7	43.85469576	12.52991307

Since effective virus Strength > 1 infection will spread across the network and result in an epidemic.

b. Keeping δ fixed, analyze how the value of β affects the effective strength of the virus (*suggestion*: plot your results). What is the minimum transmission probability (β) that results in a network-wide epidemic?

Ans)

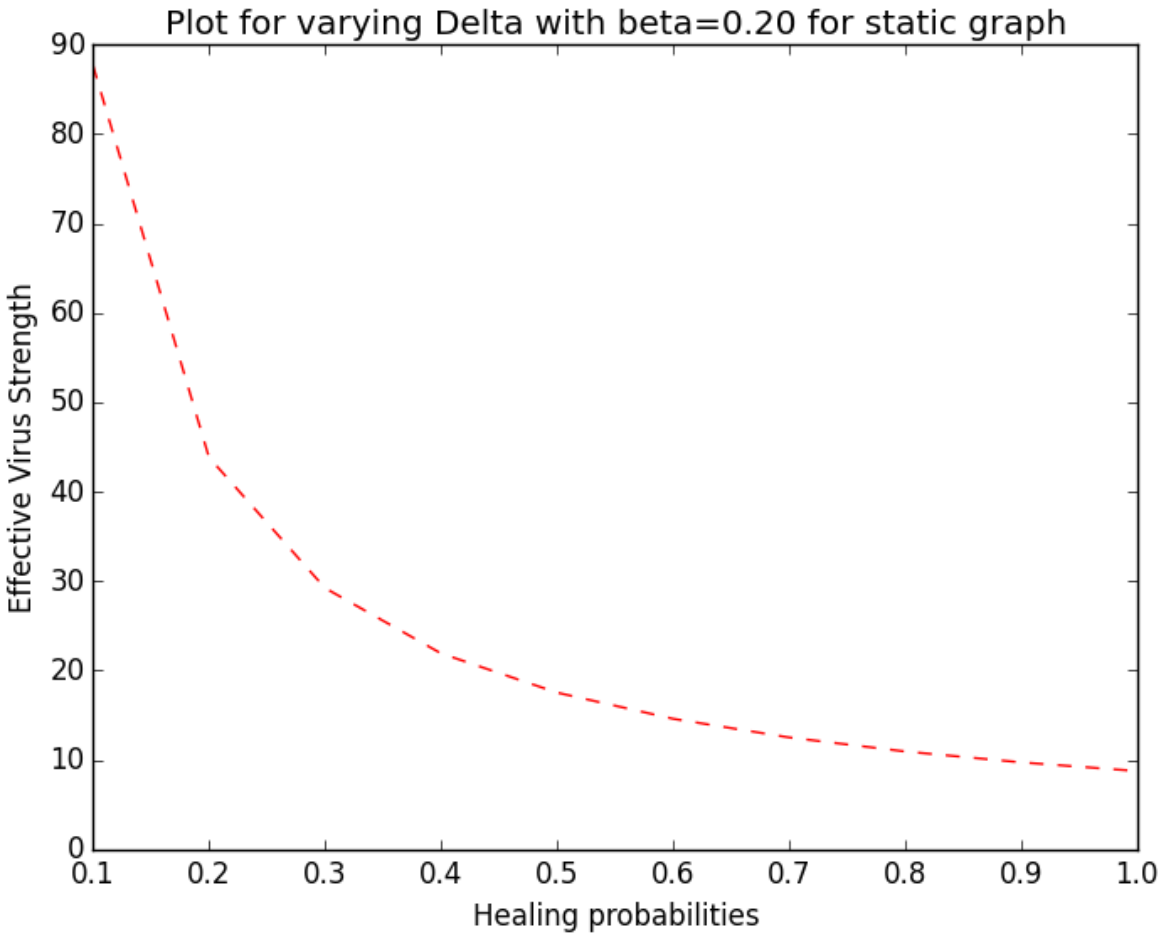
Minimum value of beta for network to be epidemic is : 0.0159618026726



c. Keeping β fixed, analyze how the value of δ affects the effective strength of the virus (*suggestion*: plot your results). What is the maximum healing probability (δ) that results in a network-wide epidemic?

Ans)

the maximum healing probability (δ) that results in a network-wide epidemic is 8.77093915215

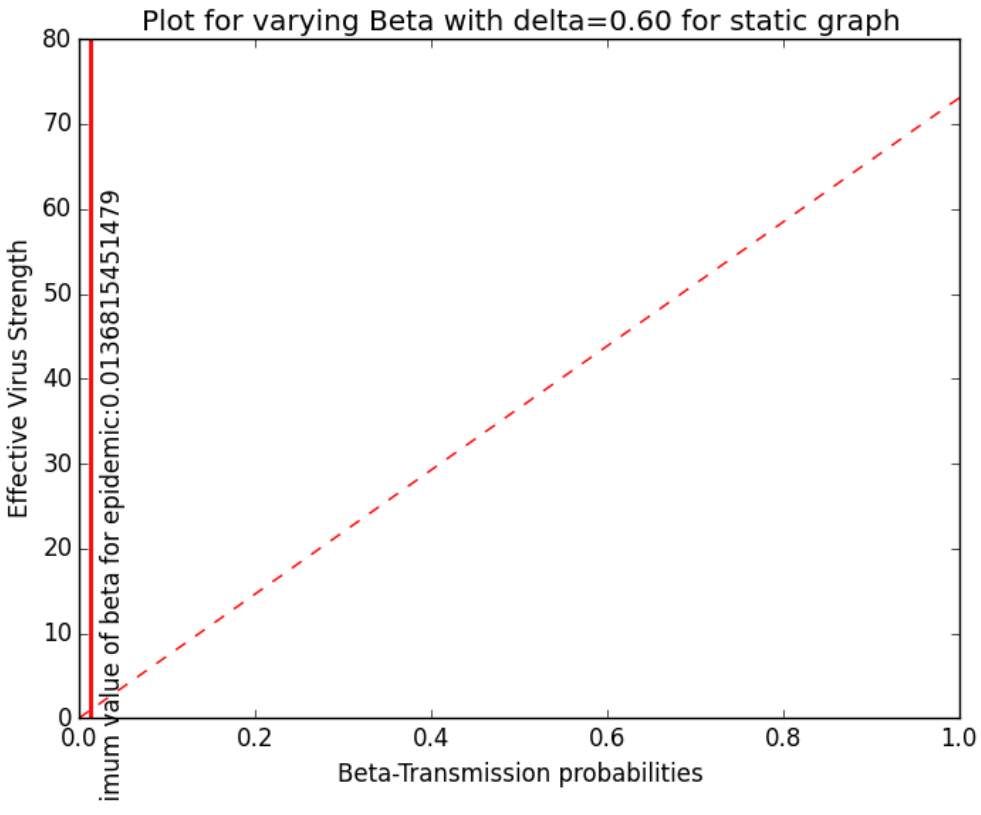


d. Repeat (1), (1a), (1b) and (1c) with $\beta = \beta_2$, and $\delta = \delta_2$.

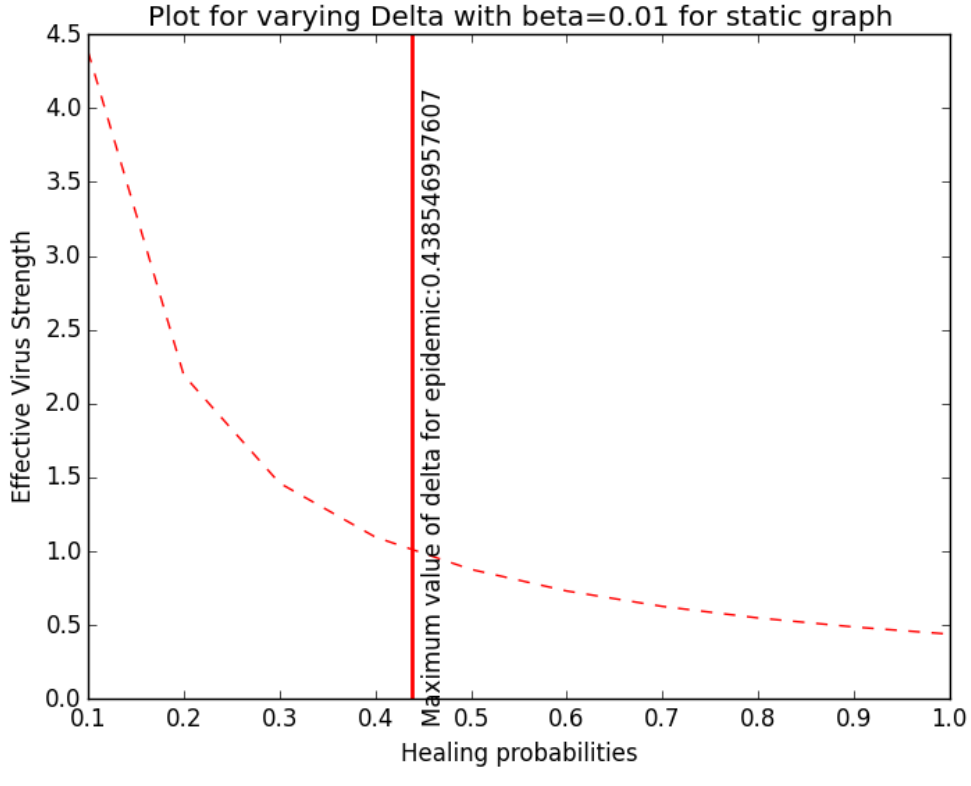
Graph Type	static.network	largest Eigen Value	Effective Virus Sterngth
β	0.01		
δ	0.6	43.85469576	0.730911596

Since effective virus strength is <1 , there is no network epidemic

Minimum value of beta for network to be epidemic is:0.0136815451479



Maximum value of delta for network to be epidemic is:0.438546957607

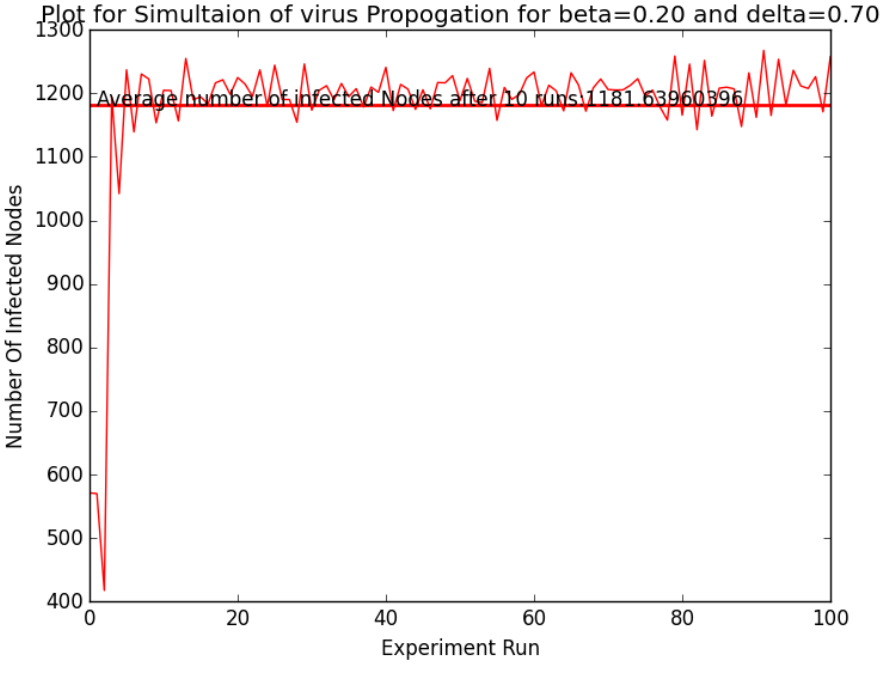


2. Write a program that simulates the propagation of a virus with the SIS VPM, given a static contact network, a transmission probability (β), a healing probability (δ), a number of initially infected nodes (c), and a number of time steps to run the simulation (t). The initially infected nodes should be chosen from a random uniform probability distribution. At each time step, every susceptible (i.e., non-infected) node has a β probability of being infected by neighboring infected nodes, and every infected node has a δ probability of healing and becoming susceptible again. Your program should also calculate the fraction of infected nodes at each time step.

a. Run the simulation program 10 times for the static contact network provided (*static.network*), with $\beta = \beta_1$, $\delta = \delta_1$, $c = n/10$ (n is the number of nodes in the network), and $t = 100$.
Ans)
Refer to 2a.py file for code in static folder.

Initial Infected Nodes: 571
Mean number of Infected nodes after 10 runs: 1181.63

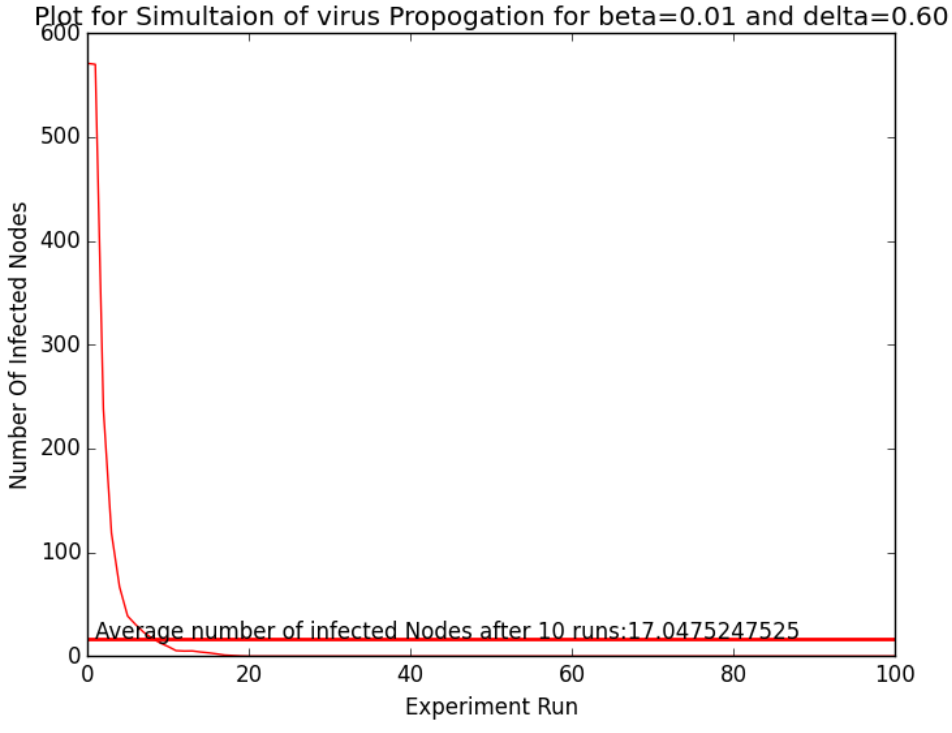
b. Plot the average (over the 10 simulations) fraction of infected nodes at each time step. Did the infection spread across the network, or did it die quickly? Do the results of the simulation agree with your conclusions in (1a)?
Ans)
As seen from the plot the average number of infected nodes after 10 runs is 1181. Since we initially start with 571 nodes it is clear that state is epidemic. This is in tune with the conclusions of 1a.



c. Repeat (2a) and (2b) with $\beta = \beta_2$, and $\delta = \delta_2$.
Ans)
Refer to 2b.py file for code in static folder.

Initial Infected Nodes:571
Mean Infected nodes:17.04

As we see from the plot the number of infected nodes is down to 1 at the end of the simulation. Hence the situation is not epidemic as asserted from the conclusions from question 1.



3. Write a program that implements an immunization policy to prevent the virus from spreading across the network. Given a number of available vaccines (k) and a contact network, your program should select k nodes to immunize. The immunized nodes (and their incident edges) are then removed from the contact network.
a. What do you think would be the optimal immunization policy? What would be its time complexity? Would it be reasonable to implement this policy? Justify.
For your program, use the following heuristic immunization policies:
- Policy A: Select k random nodes for immunization.
- Policy B: Select the k nodes with highest degree for immunization.
- Policy C: Select the node with the highest degree for immunization. Remove this node (and its incident edges) from the contact network. Repeat until all vaccines are administered.
- Policy D: Find the eigenvector corresponding to the largest eigenvalue of the contact network's adjacency matrix. Find the k largest (absolute) values in the eigenvector. Select the k nodes at the corresponding positions in the eigenvector.

a. What do you think would be the optimal immunization policy? What would be its time complexity? Would it be reasonable to implement this policy? Justify.
It make sense to remove nodes from the infected graph such way that the probability of disease spreading is reduced. The optimal immunization policy would be to find subset k nodes with the largest eigen drop among all possible $\binom{n}{k}$ possible combinations of nodes. Time complexity is combinatorial in the order of $\binom{n}{k}$ .This is not reasonable to implement since it is combinatorial in nature and would take a lot of time in computing. NetShield algorithm tries to approximate this approach which is the policy D. This in comparision has reasonable time complexity.

Policy A: Select k random nodes for immunization.

b. What do you think is the intuition behind this heuristic?

- Intuition is that when we randomly select nodes to immunize there is always probability that we select the nodes to immunize such a way that it may contain/stop the epidemic.We hope that these nodes are part of best possible subset resulting in eigen drop.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_random_nodes(Graph G, k):
    Random_Nodes = select random k nodes of graph G to immunize
    Delete Random_Nodes from graph G
    return G
```

```
def immunize_random_nodes(self, k = None):
        if k is None:
            k = self.k
        if self.vertices < k:
            print("Cannot immunize")
        random_node_ids = random.sample(range(self.vertices), k)
        self.graph.delete_vertices(random_node_ids)
        self.vertices = len(self.graph.vs)
        self.calculateLargestEigenValue()
```

This algorithm takes O(1) to select k random nodes and then O(K) steps to delete K nodes from the graph.

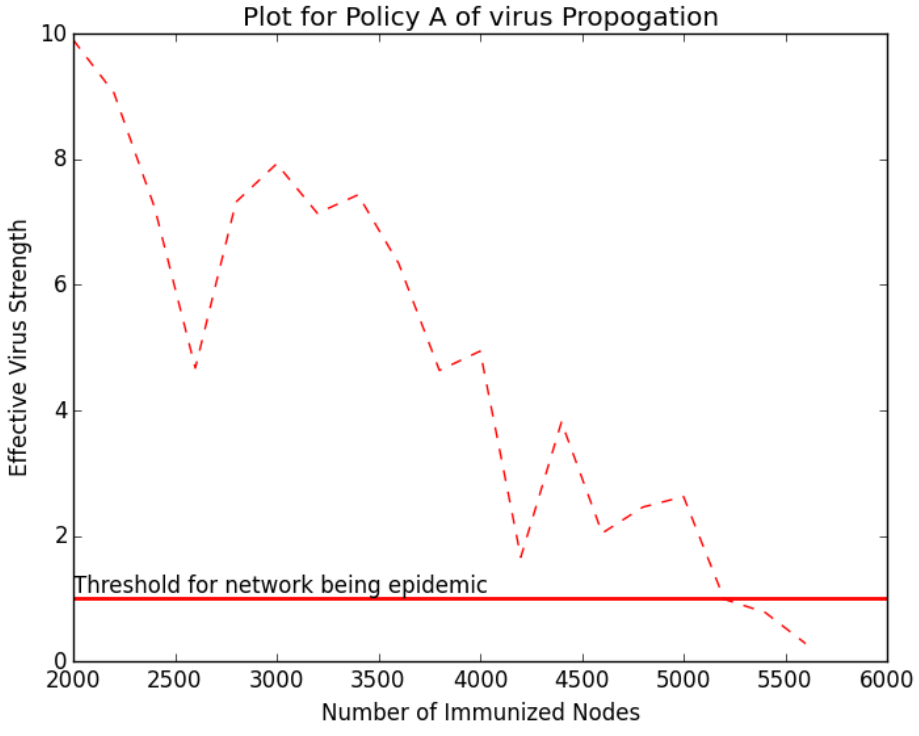
d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?

Refer to policy_A_d.py in static folder the project.

```
Effective Strength of the virus before immunization is 12.5299130745
Effective Strength of the virus is after immunization 12.2681434187
```

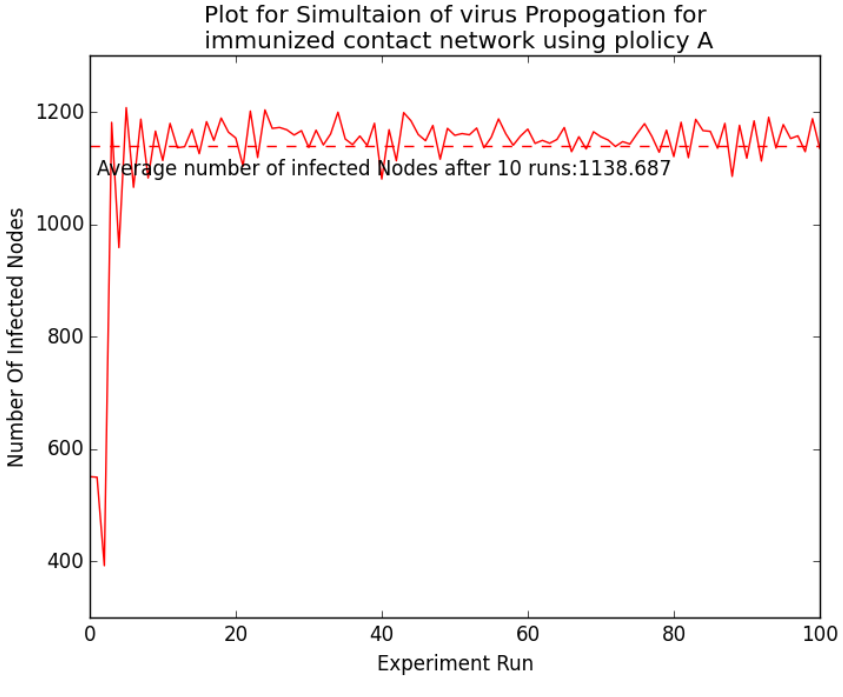
Since the effective virus strength is > 1 , the network is still epidemic.

e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.



Value of number of vaccines for containing the network is around 5250 to 5400

f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?



As we in plot above the contact network is epidemic which is in sync with what we found out in (d) since there are close to 1138 infected nodes on an average in each step.

Policy B: Select the k nodes with highest degree for immunization

b. What do you think is the intuition behind this heuristic?

- Intuition is that when we chose vertices with the k highest degree, it is more like that a bunch of dense subgraphs are immunized. By this we are trying to reduce average degree of the graph. This may in turn make network less epidemic.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_highest_degree_nodes (Graph G, k):
    Highest_Degree_Nodes = select k highest degree nodes from G
    Delete Highest_Degree_Nodes from graph G
    return G
```

```
def immunize_highest_degree_nodes(self, k = None):
        if k is None:
            k = self.k
        node_degrees = [self.graph.degree(i) for i in range(self.vertices)]
        nodes_ordered_ascending = np.argsort(node_degrees)
        top_k_nodes = nodes_ordered_ascending[-k:]
        self.graph.delete_vertices(top_k_nodes)
```

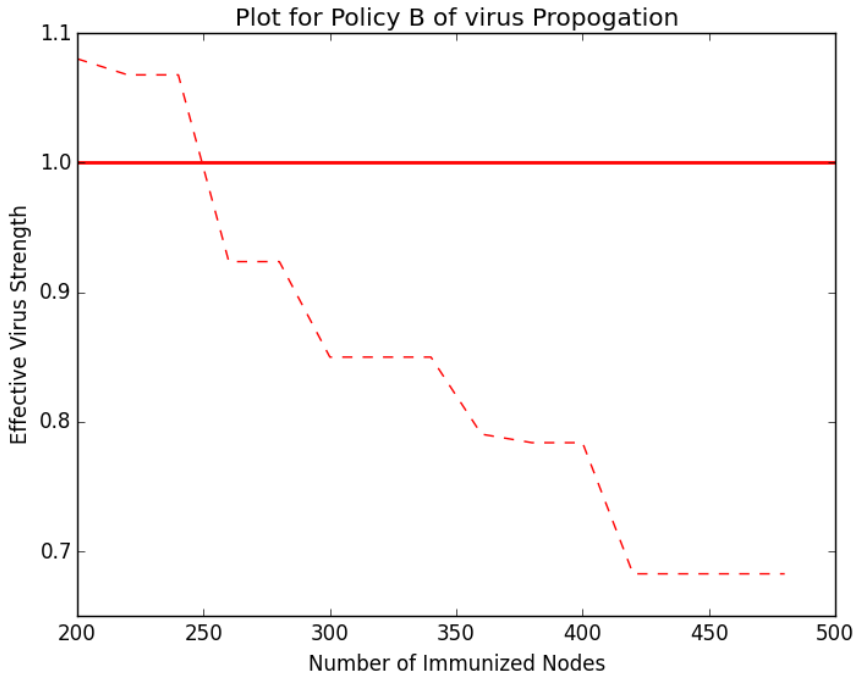
This algorithm takes O(m) steps to find degree of each node and (mlogm) sort the nodes and finally k Steps to delete nodes.

d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?
Refer to policy_B_d.py

```
Effective Strength of the virus before immunization is 12.5299130745
Effective Strength of the virus is after immunization 1.08027548024
```

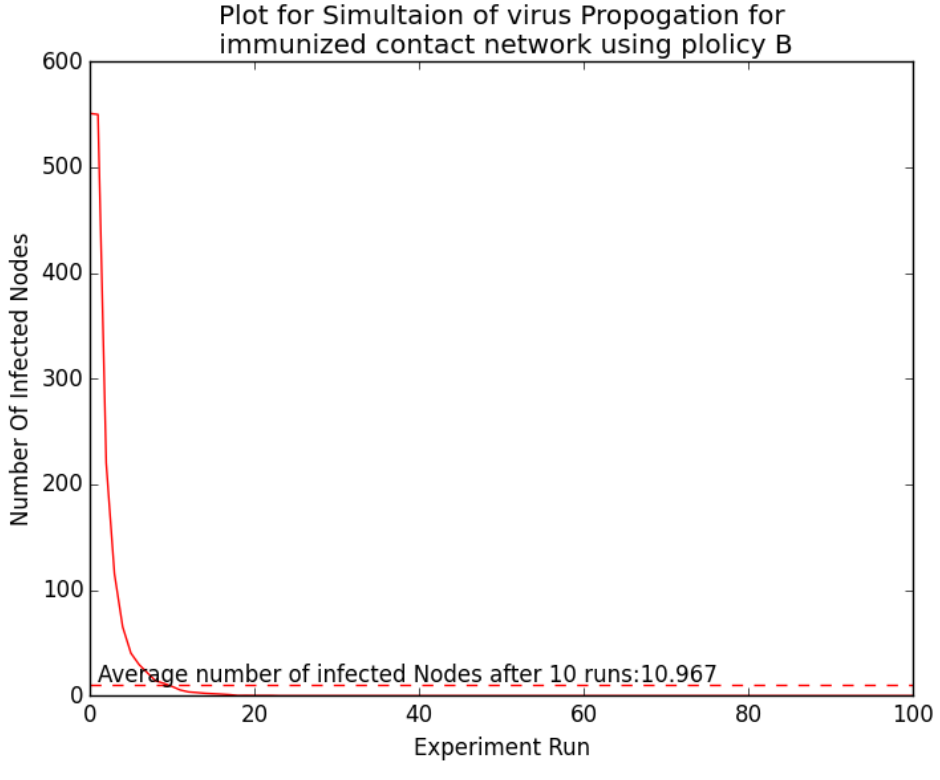
Since the effective virus strength is almost equal to 1 , the network can be contained

e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.



As we see in the plot number of vaccines required are close to 250 – 260.

f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?



Since the contact network can be contained as we see only 10 nodes on an average infected, the results agree with findings in (d)

Policy C: Select the node with the highest degree for immunization. Remove this node (and its incident edges) from the contact network. Repeat until all vaccines are administered

b. What do you think is the intuition behind this heuristic?

- This follows an iterative approach where at each step after finding the maximum degree node, algorithm removes the node from the graph. In the next step it finds the next highest degree node and removes it. This algorithm is more intuitive since we are making sure that we do not localize the maximum degree to a subset of nodes. This algorithm we are trying to remove the clusters in graph hoping that effect of virus can be contained Also we have seen that lot of times the approaching of taking highest degree like policy B does not give good results.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_highest_degree_nodes_step_by_step (Graph G, k):
  for i 1 to k
      Highest_Degree_Node = select highest degree nodes from G
      G : G – {highest_degree_node}
  return G
```

```
def immunize_highest_degree_nodes_step_by_step(self, k = None):
        if k is None:
            k = self.k
        for i in range(k):
            current_len = len(self.graph.vs)
            node_degrees = [self.graph.degree(count) for count in
range(current_len)]
            nodes_ordered_ascending = np.argsort(node_degrees)
            node_to_remove = nodes_ordered_ascending[-1:]
            self.graph.delete_vertices(node_to_remove)
```

In each iteration this algorithm takes O(m) steps to find degree of each node and (mlogm) sort the nodes and O(1) to delete the node. This is repeated k times. Hence total time complexity is O(kmlogm).This algorithm is computationally more expesive than B.

d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?

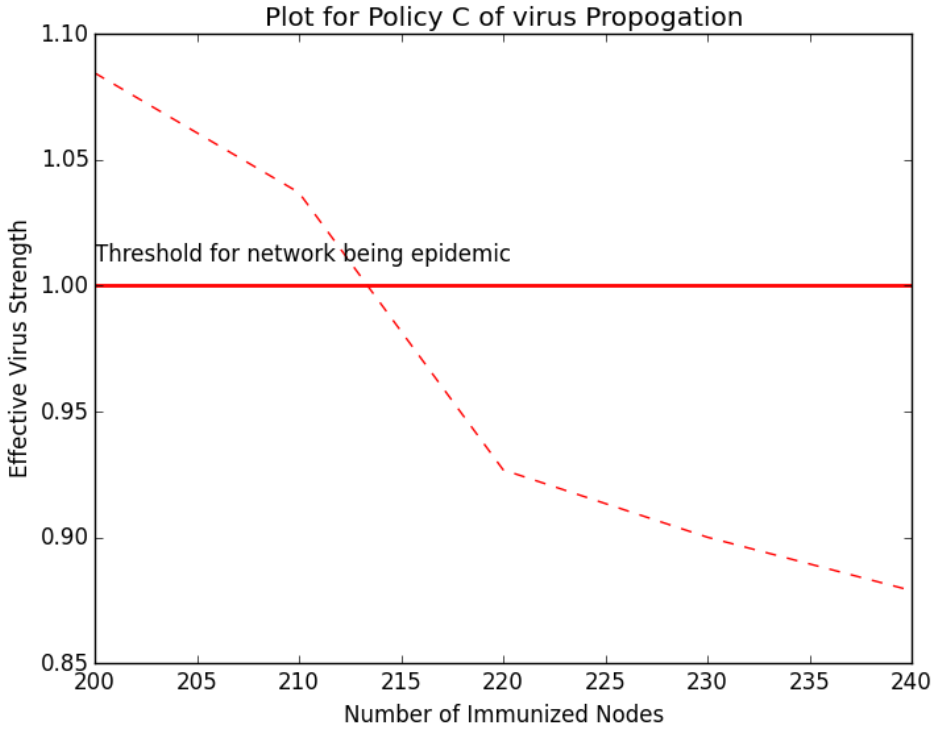
Refer to policy_C_d.py in static folder of the project.

```
Effective Strength of the virus before immunization is 12.5299130745
Effective Strength of the virus is after immunization 1.08452053952
```

Since the effective virus strength is approximately 1 , the network can be contained.

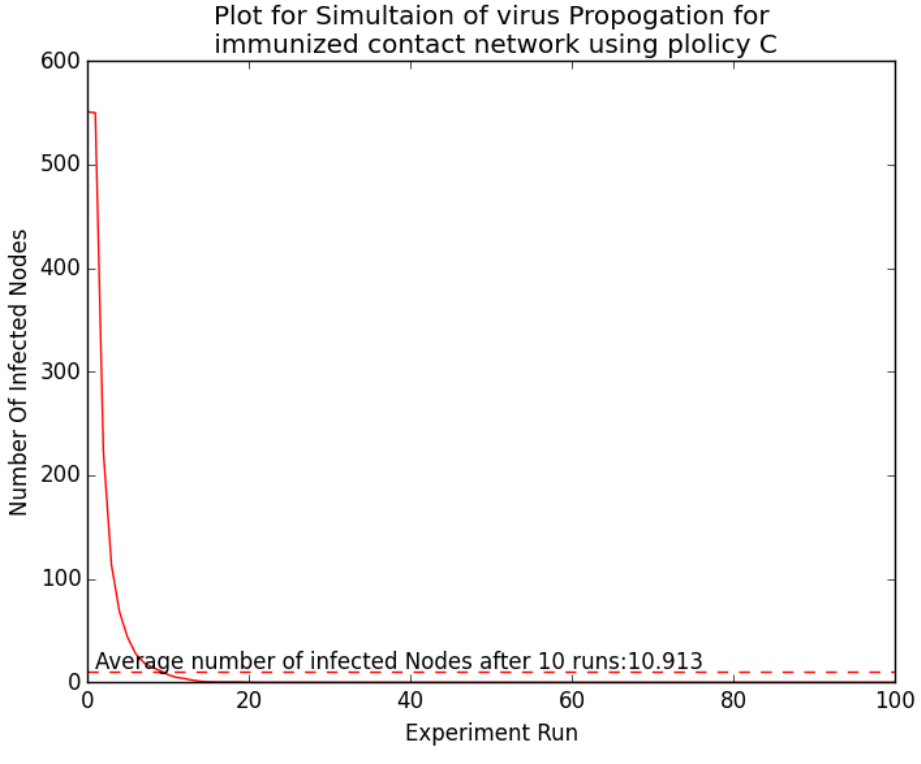
e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.

As we see in the plot below number of vaccines required are close to 213 – 215.



f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?

Since the contact network can be contained as we see only 10 nodes on an average infected, the results agree with findings in (d)



Policy D: Find the eigenvector corresponding to the largest eigenvalue of the contact network's adjacency matrix. Find the k largest (absolute) values in the eigenvector. Select the k nodes at the corresponding positions in the eigenvector

b. What do you think is the intuition behind this heuristic?

- Idea is to remove the node which has highest impact on network.
- This approximates the optimal immunization policy to find subset k nodes with the largest eigen drop among all possible $\binom{n}{k}$ possible combinations of nodes.
- NetShield Algorithm by Tong et al uses shield value to immunize the network. This method computes the largest eigen value in the eigen vector and removes the nodes corresponding from the network to immunize.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_k_nodes_from_eigen_vector (Graph G, k):
  eigen_value = Find the largest eigen value for graph G
  eigen_vector = Find the corresponding eigen vector for largest eigen value
  k_large_nodes = Find k largest largest corresponding to largest values in eigen_vector
  Delete k_large_nodes from the graph G
return G
```

```
def immunize_k_nodes_from_eigen_vector(self, k = None):
      if k is None:
            k = self.k
      eigen_value,eigen_vectors = linalg.eigh
                        (self.graph.get_adjacency().data,eigvals=(self.vertices - 1,
      self.vertices - 1))
      prinicipal_eigen_vector = eigen_vectors[0:]
      prinicipal_eigen_vector = prinicipal_eigen_vector.tolist();
      vector_value = [prinicipal_eigen_vector[i][0] for i in range(self.vertices)]
      vector_value = list(vector_value)
      nodes_ordered_ascending = list(np.argsort(vector_value))
      nodes_to_delete = nodes_ordered_ascending[-k:]
      self.graph.delete_vertices(nodes_to_delete)
```

This algorithm is dominated by the calculation of eigen vector which involves matrix multiplication. Hence it has a time complexity of $O(m^3)$.

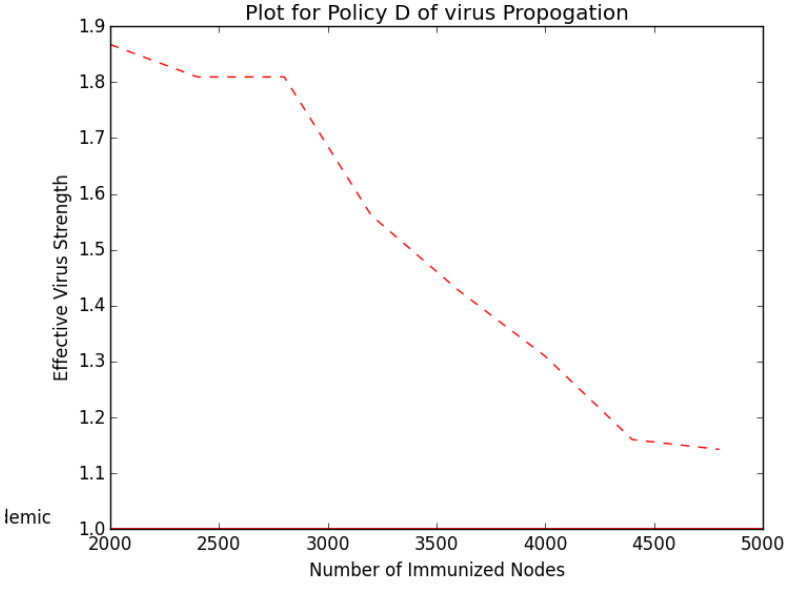
d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?

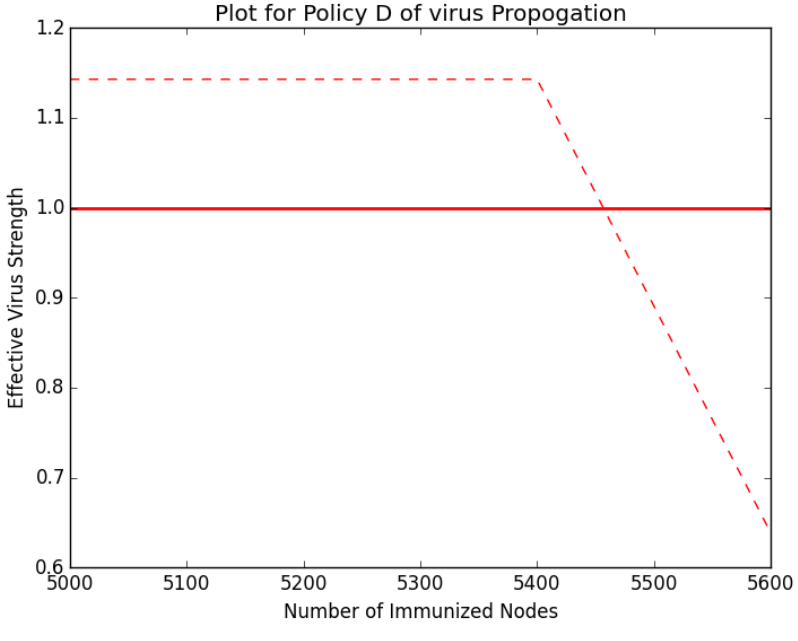
Refer to policy_D_d.py

```
Effective Strength of the virus before immunization is 12.5299130745
Effective Strength of the virus is after immunization 3.0705278823
```

Since the effective virus strength is > 1 , the network is still epidemic.

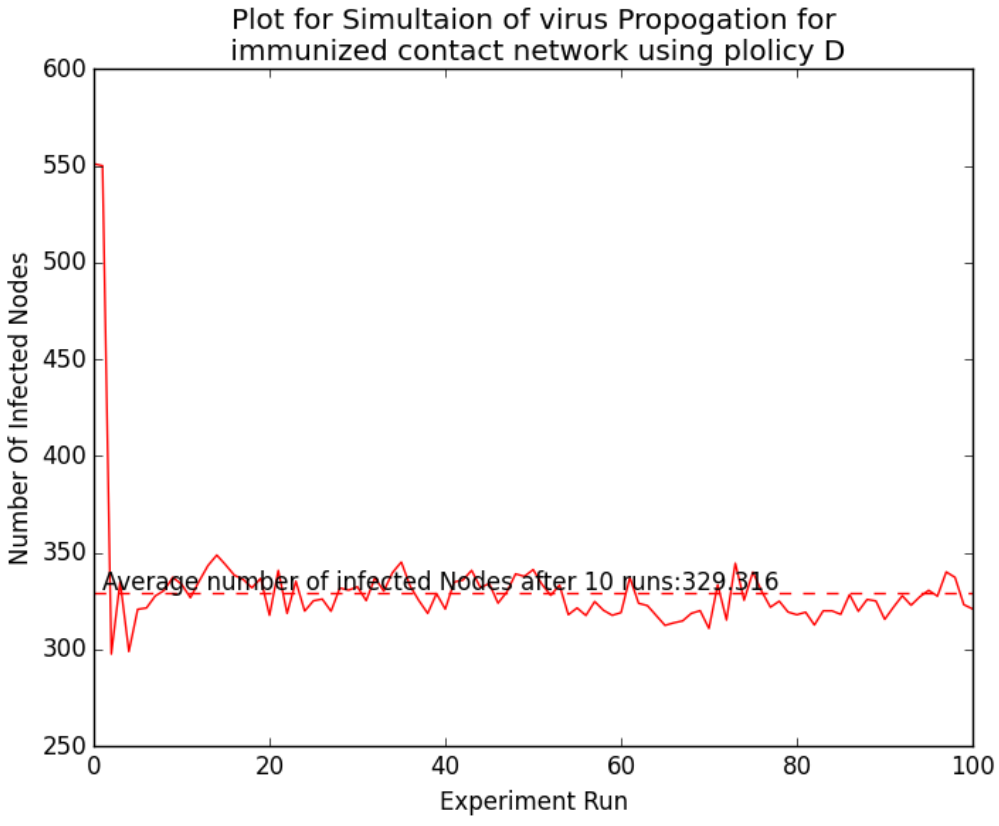
e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.





Minimum number of vaccines required is close to 5480 as seen from the plot above.

f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?



As we see from the plot the network doesn't seem to be strongly epidemic. At the same time it is not completely contained. Ideally I would expect policy D to outperform other policies. But for some reason policy D is giving poor results as compared to policy C or B. Possible reasons might be inaccuracy of the library to calcualte 'eigen vectors corresponding to maximum eigen value' limited by the machine(Policy_D_e.py,Policy_D_f.py).It might be a good idea to use other library or programming language to debug this further and understand the behavior.

Option 2: Virus Propagation on Time-Varying Networks

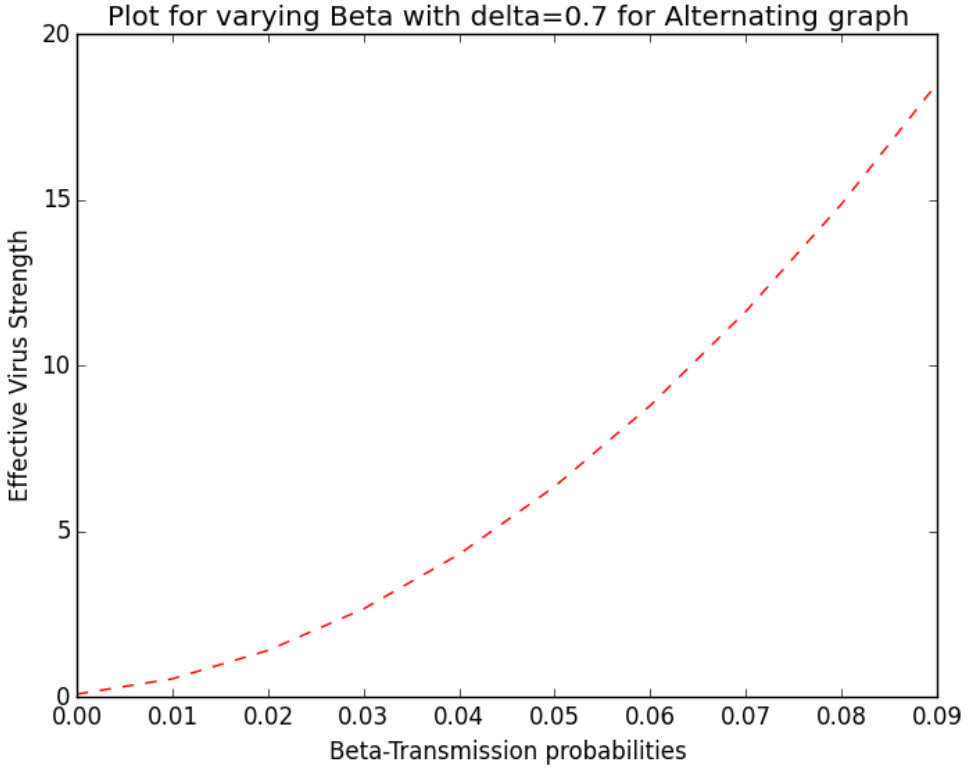
4. Repeat problem (1) using the set of 2 alternating contact networks provided (*alternating1.network* and *alternating2.network*). Here, instead of calculating the effective strength of the virus, you should calculate the largest eigenvalue of the system-matrix (see supplementary materials for details on the system-matrix). Answer questions (1), (1a), (1b), (1c), and (1d).

a. Will the infection spread across the network (i.e., result on an epidemic), or will it die quickly?

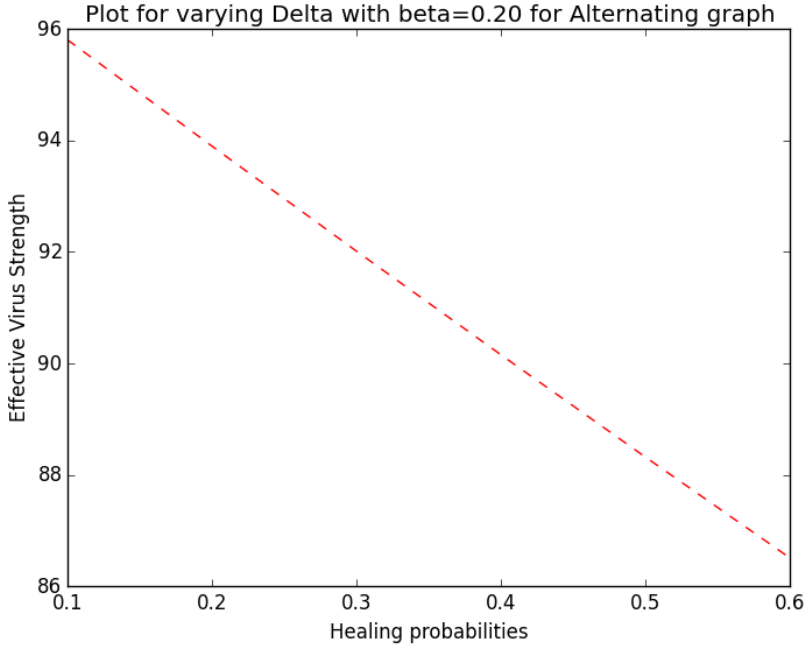
Graph Type	static.network	Effective Virus Strength
β	0.2	
δ	0.7	84.7304008089

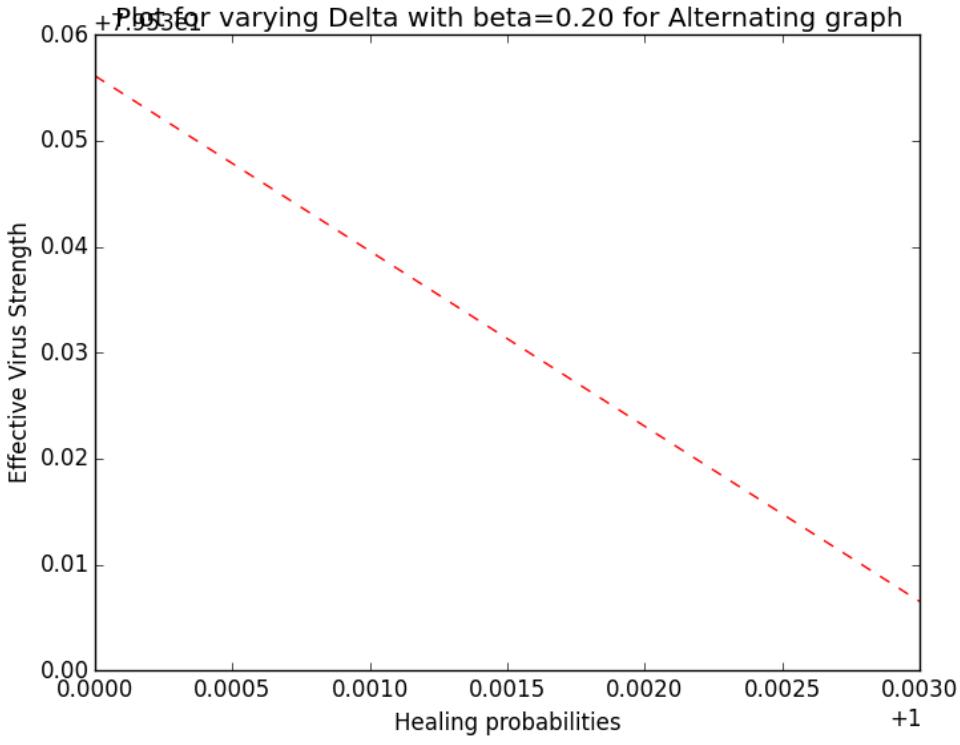
Since effective virus Strength > 1 infection will spread across the network and result in an epidemic.

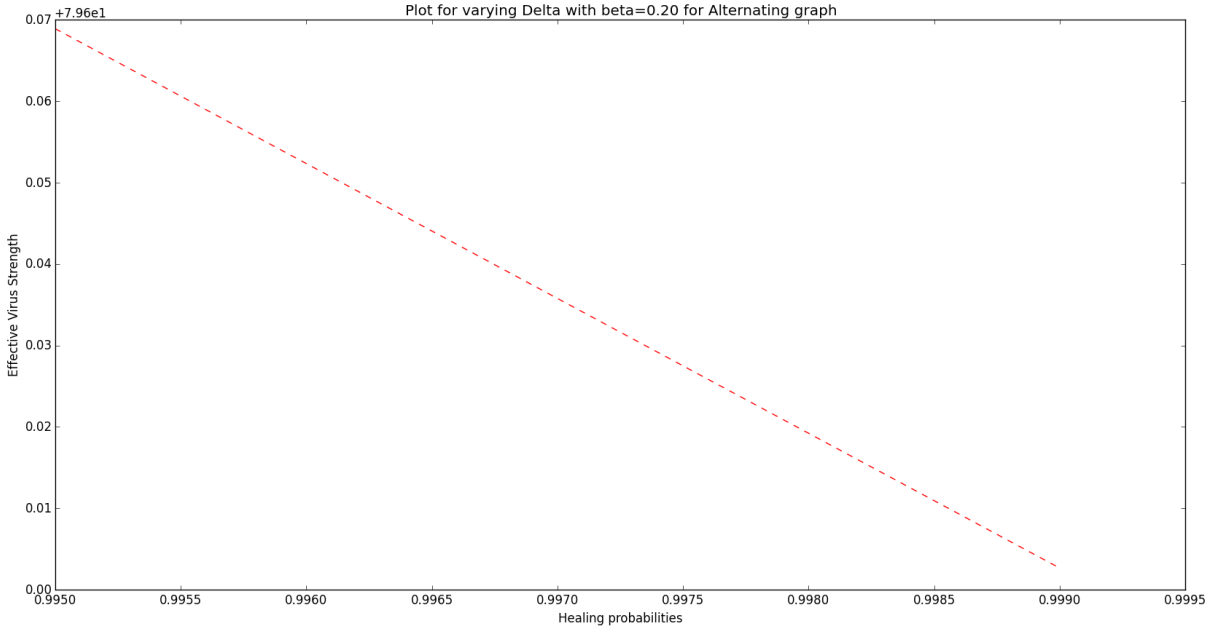
b. Keeping δ fixed, analyze how the value of β affects the effective strength of the virus (*suggestion*: plot your results). What is the minimum transmission probability (β) that results in a network-wide epidemic?
Ans)
Minimum value of beta for epidemic is between 0 and 0.01

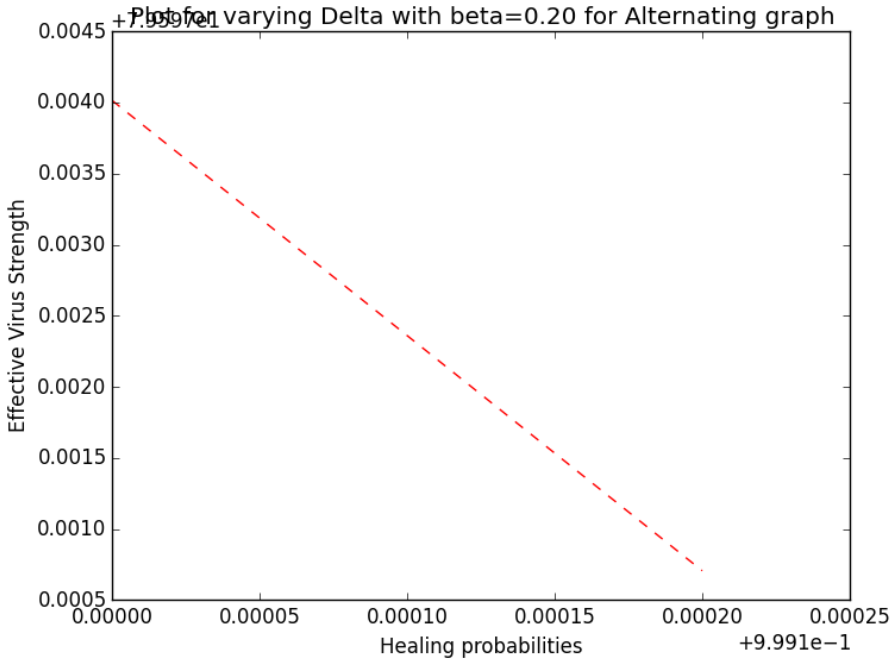


c. Keeping β fixed, analyze how the value of δ affects the effective strength of the virus (*suggestion*: plot your results). What is the maximum healing probability (δ) that results in a network-wide epidemic?
-Maximum Value of delta for which network is not epidemic is around 0.9991 to 0.9998. I have plotted multiple graphs to arrive at this conclusion.




Plot for varying Delta with beta=0.20 for Alternating graph
+7.953e1
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Effective Virus Strength
0.0000
0.0005
0.0010
0.0015
0.0020
0.0025
0.0030
Healing probabilities
+1

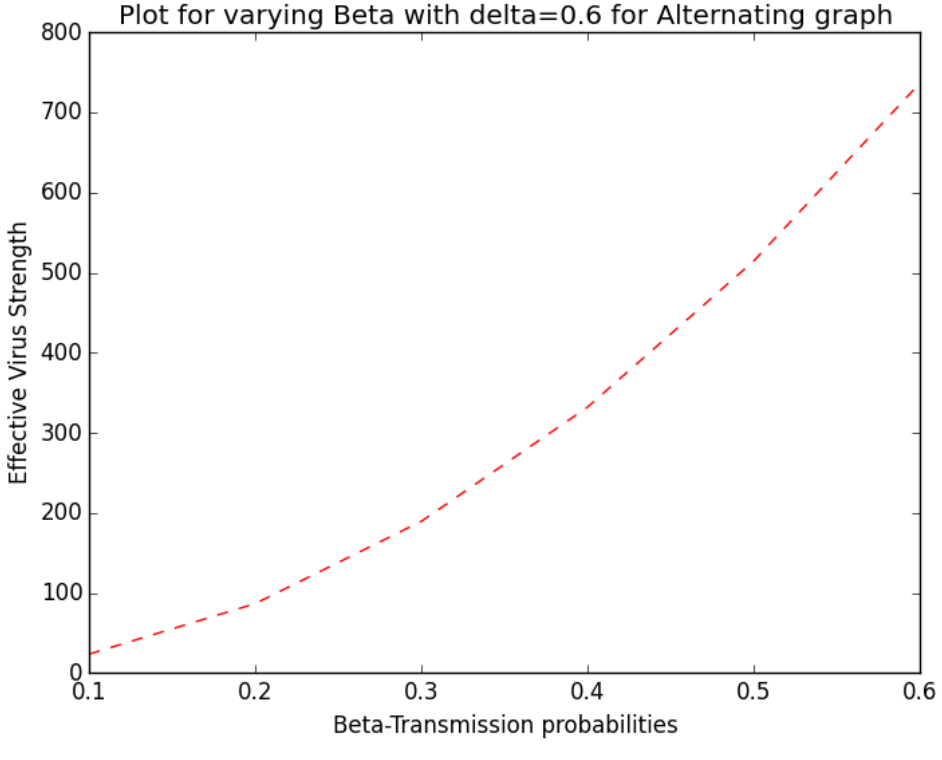

Plot for varying Delta with beta=0.20 for Alternating graph
+7.96e1
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
Effective Virus Strength
0.9950
0.9955
0.9960
0.9965
0.9970
0.9975
0.9980
0.9985
0.9990
0.9995
Healing probabilities



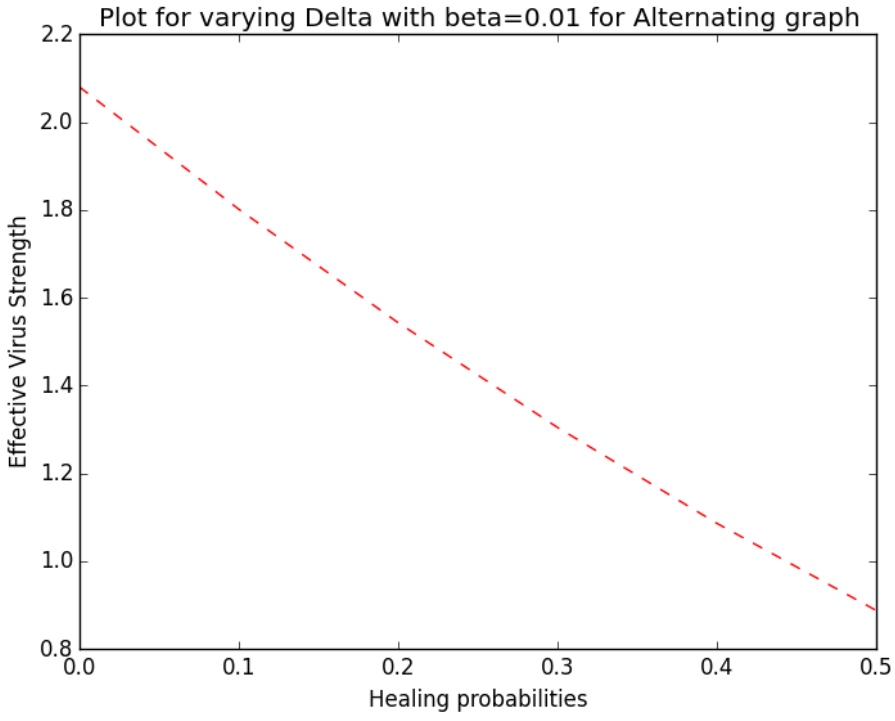
d. Repeat (1), (1a), (1b) and (1c) with $\beta = \beta_2$, and $\delta = \delta_2$.

Graph Type	static.network	Effective Virus Sterngth
β	0.01	
δ	0.6	0.70984037317

Since effective virus strength is <1 , there is no network epidemic



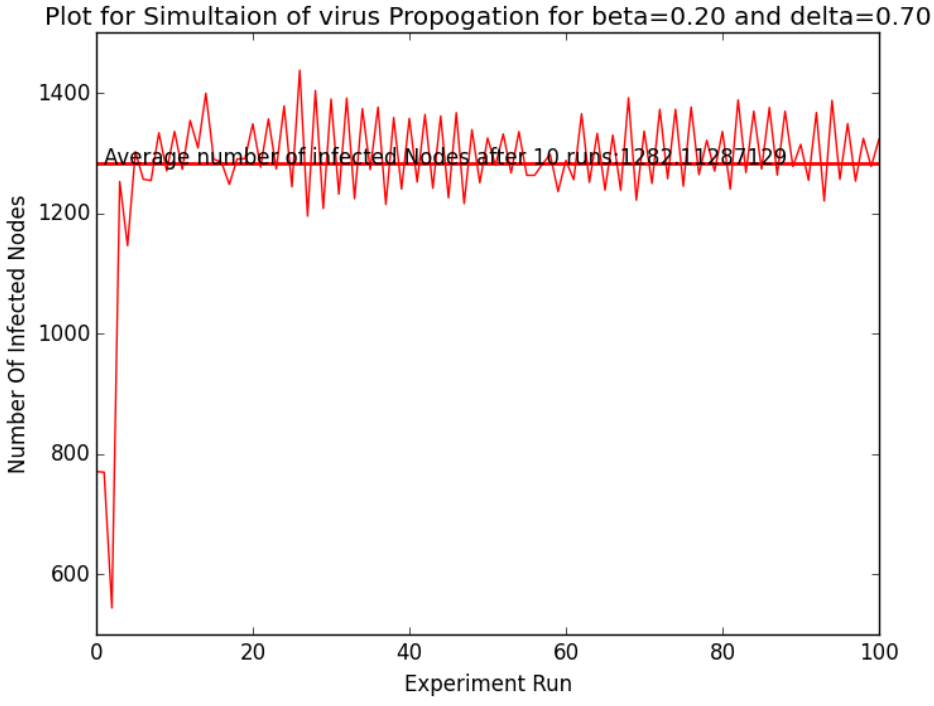
The minimum value of beta required for vaccination is close to or less than 0.1 as seen from the plot above.



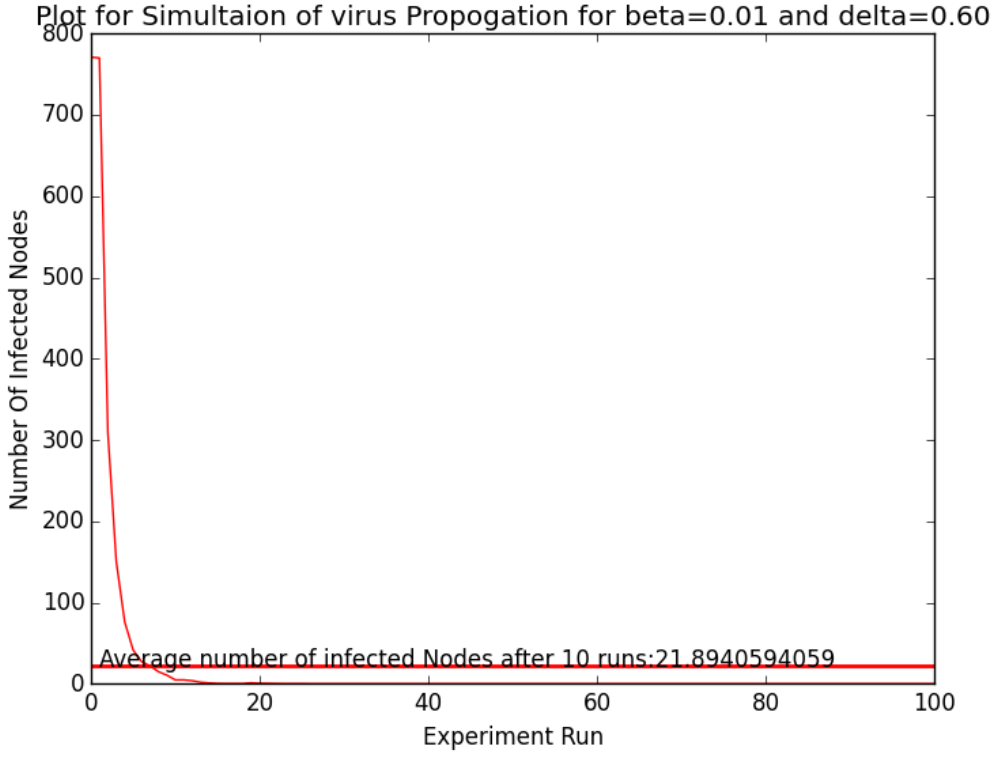
The maximum value of delta required for network to become epidemic is in range 0.4 to 0.45 as seen from the plot above.

5. Extend your virus propagation simulation program from problem (2) to allow time-varying graphs. Your modified program will be given a set of *T* alternating contact networks, a transmission probability (β), a healing probability (δ), a number of initially infected nodes (*c*), and a number of time steps to run the simulation (*t*). Your program should alternate between contact networks at each time step. That is, at a given time step t_i, for $i \in [0, t)$, your program should simulate the propagation of the virus on alternating contact network $(t_i \bmod T) + 1$.
For the modified virus propagation simulation program, and the set of 2 alternating contact networks provided (*alternating1.network* and *alternating2.network*), answer questions (2a), (2b), and (2c).
a)
Run the script 4_2a.py in /Alternate Graph/42a.py of project folder

```
__author__ = 'pejakalabhargava'
import matplotlib.pyplot as plot
import numpy as np
import AlternateSimulatePropogation as simulation
if __name__ == '__main__':
    sp = simulation.AlternateSimulatePropogation(fileName1 =
"input/alternating1.network", fileName2 =
"input/alternating2.network",beta=0.01,delta=0.60,t=100)
    infectedNodeNumber =  sp.simulateInfectionMultipleTimes(run_times=10);
    infectedNodeNumber.insert(0,sp.c1);
    runs = [i for i in range(0,101)]
    plot.plot(runs,infectedNodeNumber,color="r")
    plot.ylabel("Number Of Infected Nodes")
    plot.xlabel("Experiment Run")
    plot.title("Plot for Simultaion of virus Propogation for beta=0.01 and delta=0.60")
    mean_infected = np.mean(infectedNodeNumber)
    plot.axhline(y=mean_infected, linewidth=2, color="r")
    plot.text(1, mean_infected+0.1 ,'Average number of infected Nodes after 10 runs:'
+str(mean_infected))
    plot.show()
```



As seen from the plot above the network is epidemic at the end of simulation which is in accordance with the finding from calculating the effective virus strength.



As seen from the plot above the network is not epidemic and is contained at the end of simulation which is in accordance with the finding from calculating the effective virus strength.

6. Extend your immunization policy program from problem (3) to allow time-varying graphs. Your modified program will be given a set of *T* alternating contact networks and a number of available vaccines (*k*). Answer question (3a).

For your program, use the following heuristic immunization policies:

- Policy A: Select *k* random nodes for immunization.
- Policy B: Select the *k* nodes with highest average degree across all alternating contact networks for immunization.
- Policy C: Select the node with the highest degree out of all the alternating contact networks for immunization. Remove this node (and its incident edges) from all the alternating contact networks. Repeat until all vaccines are administered.
- Policy D: Find the eigenvector corresponding to the largest eigenvalue of the system-matrix (see supplementary materials for details on the system-matrix). Find the *k* largest (absolute) values in the eigenvector. Select the *k* nodes at the corresponding positions in the eigenvector.

For each heuristic immunization policy (A, B, C, and D) and for the set of 2 alternating contact networks provided (*alternating1.network* and *alternating2.network*), answer questions (3b), (3c), (3d), and (3e).

a. What do you think would be the optimal immunization policy? What would be its time complexity? Would it be reasonable to implement this policy? Justify.
It make sense to remove nodes from the infected graph such way that the probability of disease spreading is reduced. The optimal immunization policy would be to find subset k nodes with the largest eigen drop among all possible $\binom{n}{k}$ possible combinations of nodes. Time complexity is combinatorial in the order of $\binom{n}{k}$ .This is not reasonable to implement since it is combinatorial in nature and would take a lot of time in computing. NetShield algorithm tries to approximate this approach which is the policy D. This in comparision has reasonable time complexity.

Policy A: Select k random nodes for immunization.

b. What do you think is the intuition behind this heuristic?

- Intuition is that when we randomly select nodes to immunize there is always probability that we select the nodes to immunize such a way that it may contain/stop the epidemic. We hope that these nodes are part of best possible subset resulting in eigen drop.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_random_nodes(Graph G1,Graph G2, k):
    random_Nodes = select random k nodes of graph G1/G2to immunize
    Delete random_Nodes from graph G1 and graph G2
    return G1,G2
```

```
def immunize_random_nodes(self, k = None):

  if k is None:
            k = self.k
        if self.vertices1 < k:
            print("Cannot immunize")
        random_node_ids = random.sample(range(self.vertices1), k)
        #delete the nodes from graph1 and graph2
        self.graph1.delete_vertices(random_node_ids)
        self.vertices1 = np.size(self.graph1.vs())
        self.graph2.delete_vertices(random_node_ids)
        self.vertices2 = np.size(self.graph2.vs())
```

This algorithm takes O(1) to select k random nodes and then O(K) steps to delete K nodes from G1 and G2.

d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?

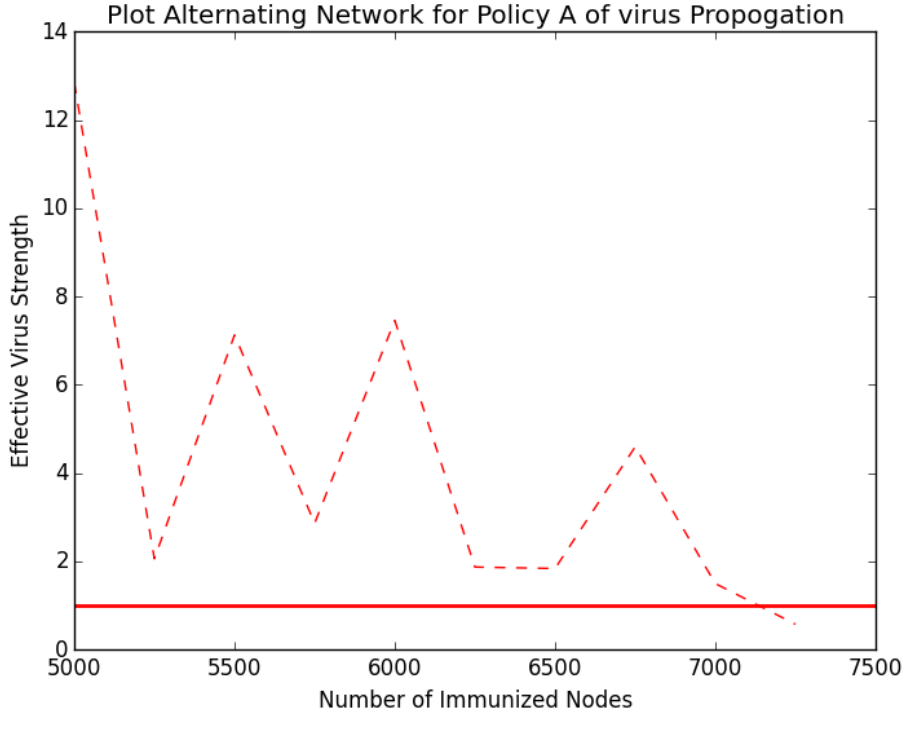
Refer to policy_A_d.py in Alternate Graph folder of the project

```
Effective Strength of the virus before immunization is 84.7304008089
Effective Strength of the virus is after immunization 81.0326000242
```

Since the effective virus strength is > 1 , the network is still epidemic.

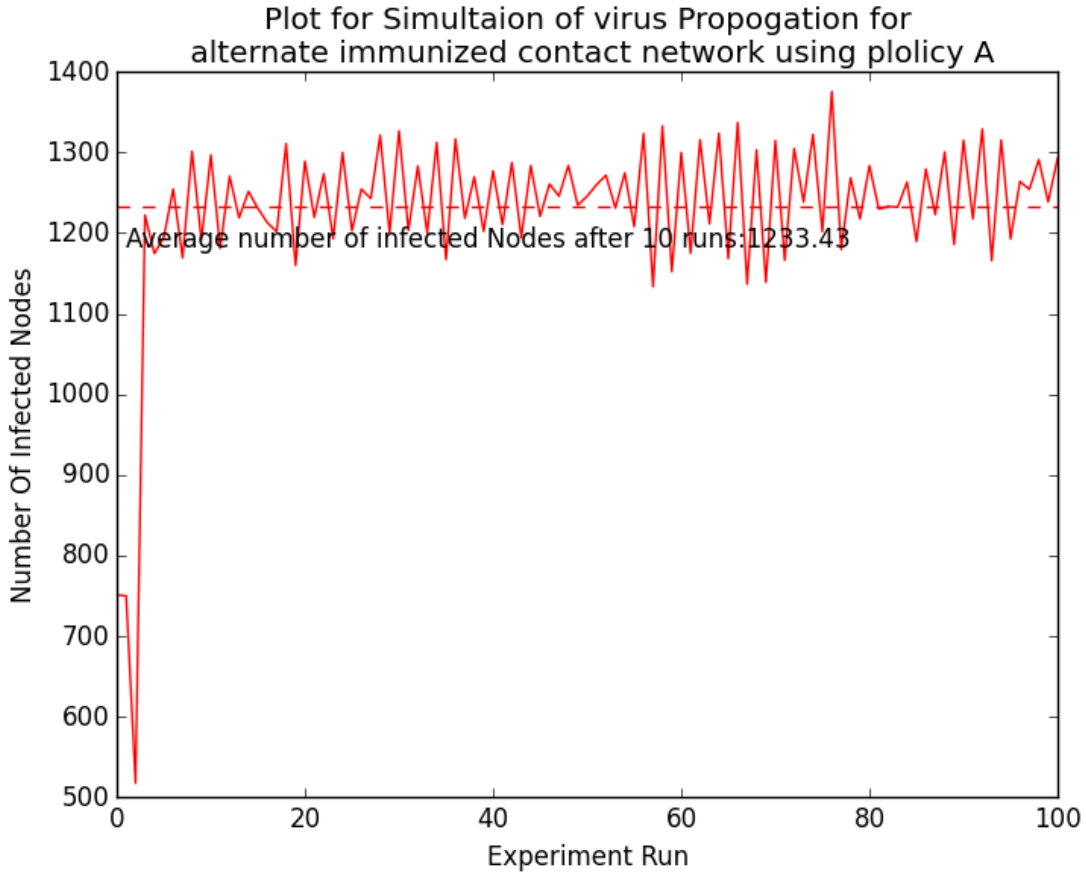
e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.

Value of number of vaccines for containing the network is around 7300 approximately as seen from the plot below.



f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?

As we in plot below the contact network is epidemic which is in sync with what we found out in (d) since there are close to 1233 infected nodes on an average in each step.



Policy B: Select the k nodes with highest degree for immunization

b. What do you think is the intuition behind this heuristic?

- Intuition is that when we chose vertices with the k highest degree, it is more like that a bunch of dense subgraphs are immunized. By this we are trying to reduce average degree of the graph. This may in turn make network less epidemic.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_highest_degree_nodes (Graph G1,Graph G2, k):
    Degree_Nodes1 = select degree of each node from G1
    Degree_Nodes1 = select degree of each node from G2
    Average_degree = Average of Degree_Nodes1 and Degree _Nodes2 in order
    Highest_Degree_Nodes = top k nodes from Average_degree
    Delete Highest_Degree_Nodes from graph G1 and G2
    return G1,G2
```

```
def immunize_highest_degree_nodes(self, k = None):
  if k is None:
      k = self.k
  node_degrees1 = [self.graph1.degree(i) for i in range(self.vertices1)]
  node_degrees2 = [self.graph2.degree(i) for i in range(self.vertices2)]
  avg_degree = (np.array(node_degrees1)+ np.array(node_degrees2))/2
  #sort it according to the degree and maintain the index of the value which is
  #nothign but the node id
  nodes_ordered_ascending = np.argsort(avg_degree)
  #Selects the k nodes with highest degree for immunization
  top_k_nodes = nodes_ordered_ascending[-k:]
  self.graph1.delete_vertices(top_k_nodes)
  self.graph2.delete_vertices(top_k_nodes)
  self.vertices1 = np.size(self.graph1.vs())
  self.vertices2 = np.size(self.graph2.vs())
```

This algorithm takes O(m) steps to find degree of each node for G1 and G2. O(m) to calculate average defree. I takes (mlogm) sort the nodes and finally K Steps to delete nodes from G1 and G2.

d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?

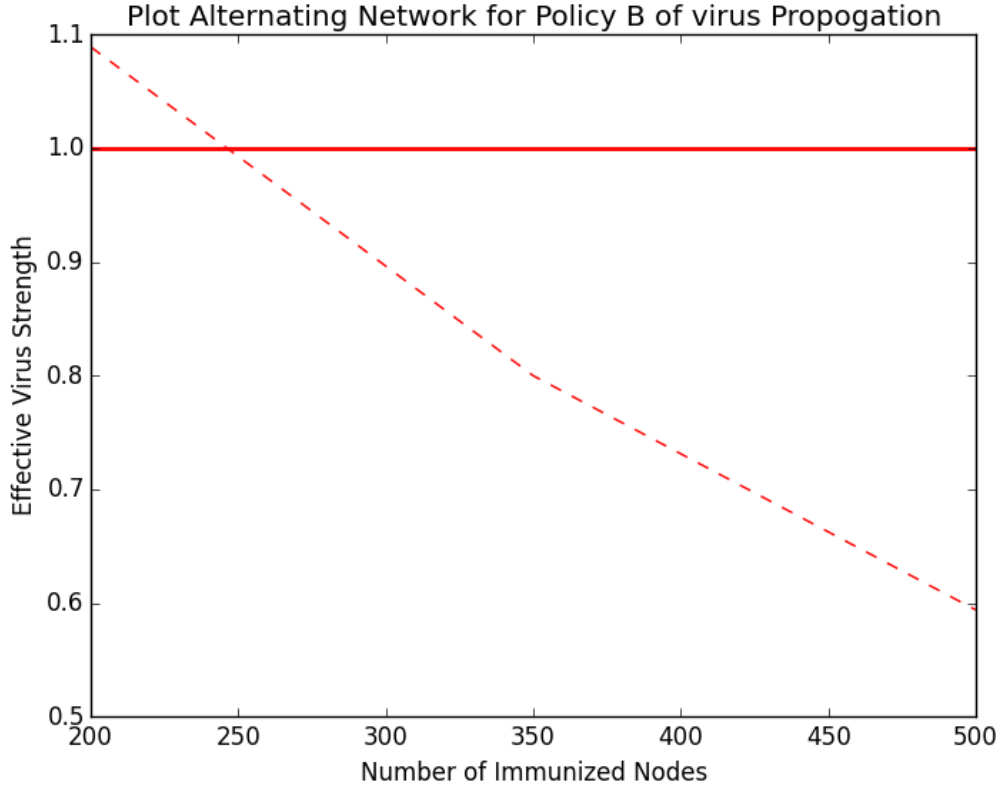
Run Policy_B_d.py from Alternate Graph folder of project

```
Effective Strength of the virus before immunization is 84.7304008089
Effective Strength of the virus is after immunization 1.08914879355
```

Since the effective virus strength is almost equal to 1 , the network can be contained

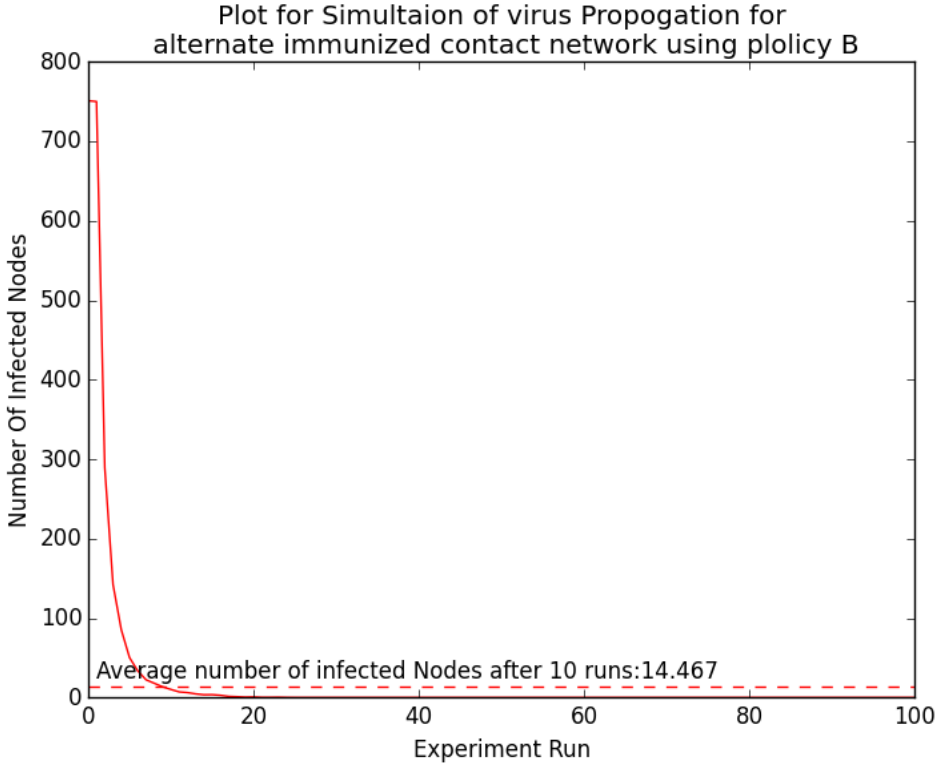
e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.

As we see in the plot number of vaccines required are close to 250 – 260.



f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?

Since the contact network can be contained as we see only 14 nodes on an average infected at the end of simulation, the results agree with findings in (d)



Policy C: Select the node with the highest degree for immunization. Remove this node (and its incident edges) from the contact network. Repeat until all vaccines are administered

b. What do you think is the intuition behind this heuristic?

- This follows an iterative approach where at each step after finding the maximum degree node, algorithm removes the node from the graph. In the next step it finds the next highest degree node and removes it. This algorithm is more intuitive since we are making sure that we do not localize the maximum degree to a subset of nodes. This algorithm we are trying to remove the clusters in graph hoping that effect of virus can be contained Also we have seen that lot of times the approaching of taking highest degree like policy B does not give good results.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_highest_degree_nodes_step_by_step (Graph G1,GraphG2, k):
  for i 1 to k
      immunize_highest_degree_nodes(G1,G2)
  return G
```

```
def immunize_highest_degree_nodes_step_by_step(self, k = None):
  if k is None:
      k = self.k
  for i in range(k):
   #call immunize_highest_degree_nodes with node count as 1
   self.immunize_highest_degree_nodes(1
```

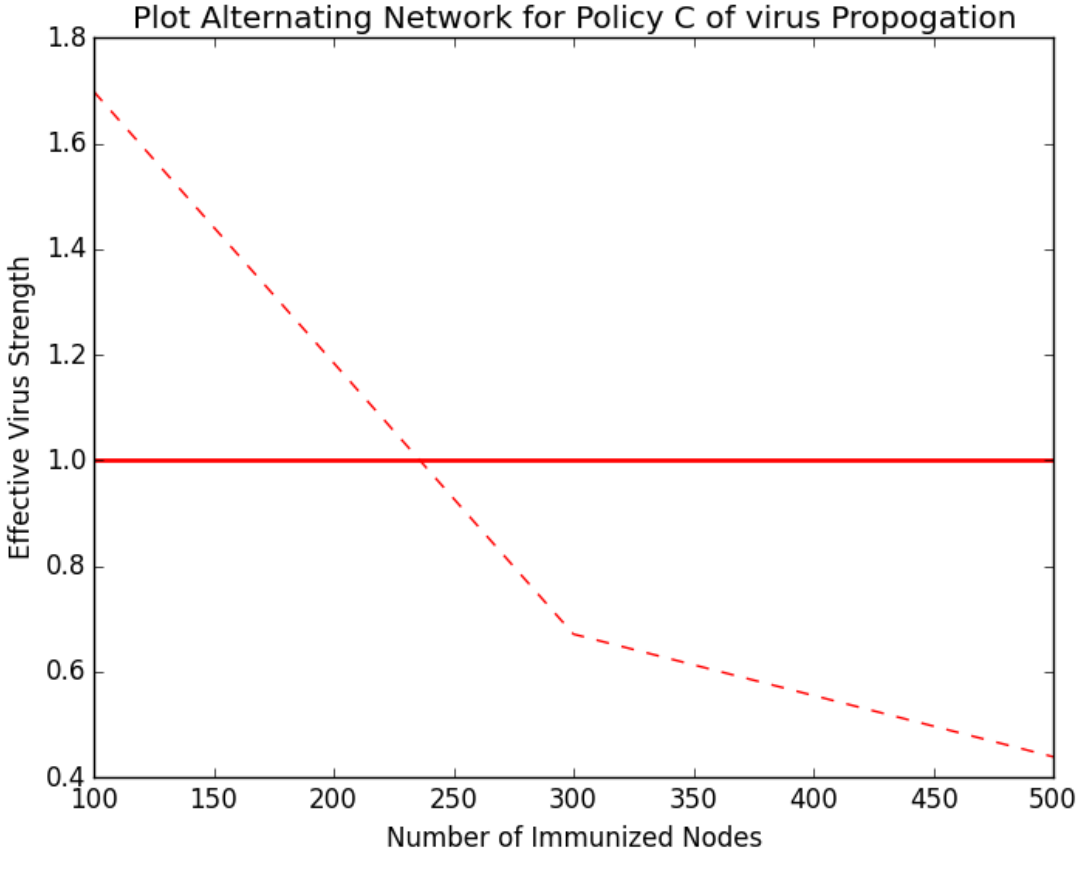
To find highest degree node and delte the vertes takes O(mlogm) as seen in policy B. Hence total time complexity is O(kmlogm).

d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?
Run Policy_C_d.py from Alternate Graph folder of project

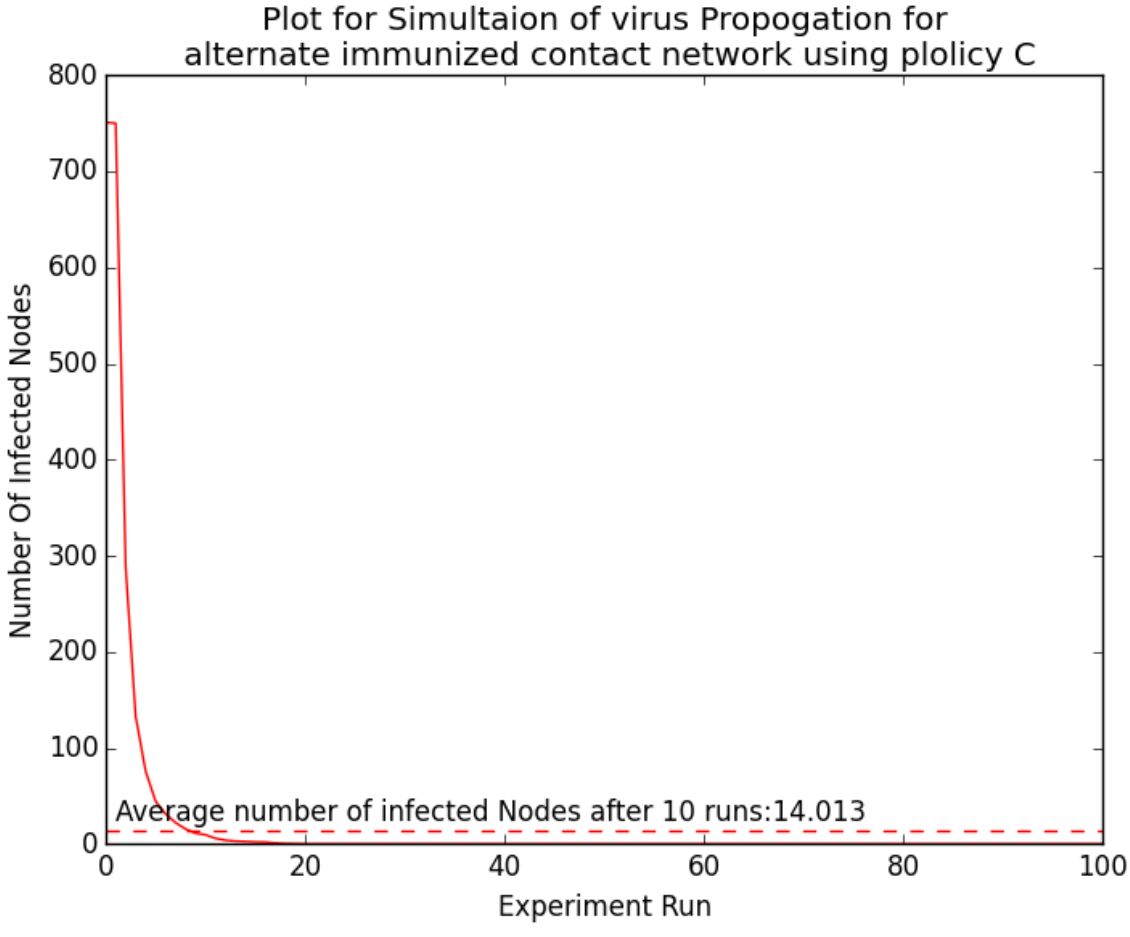
```
Effective Strength of the virus before immunization is 84.7304008089
Effective Strength of the virus is after immunization 0.975718574006
```

Since the effective virus strength is less than 1 , the network can be contained.

e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.
As we see in the plot below number of vaccines required are close to 240 – 250.



f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?
Since the contact network can be contained as we see only 14 nodes on an average infected, the results agree with findings in (d)



Policy D: Find the eigenvector corresponding to the largest eigenvalue of the contact network's adjacency matrix. Find the k largest (absolute) values in the eigenvector. Select the k nodes at the corresponding positions in the eigenvector

b. What do you think is the intuition behind this heuristic?

- Idea is to remove the node which has highest impact on network.
- This approximates the optimal immunization policy to find subset k nodes with the largest eigen drop among all possible $\binom{n}{k}$ possible combinations of nodes.
- NetShield Algorithm by Tong et al uses shield value to immunize the network. This method computes the largest eigen value in the eigen vector and removes the nodes corresponding from the network to immunize.

c. Write a pseudocode for this heuristic immunization policy. What is its time complexity?

```
immunize_k_nodes_from_eigen_vector (Graph G1,Graph G2, k):
  system_matrix = calculate system_matrix for G1 and G2
  eigen_value = Find the largest eigen value for system_matrix
  eigen_vector = Find the corresponding eigen vector for largest eigen value of system_matrix
  k_large_nodes = Find k largest largest corresponding to largest values in eigen_vector
  Delete k_large_nodes from the graph G1,graph G2
  return G1 and G2
```

```
    def immunize_k_nodes_from_eigen_vector(self, k = None, beta = None, delta = None):
        if k is None:
            k = self.k
        if beta is None:
            beta = self.beta
        if delta is None:
            delta = self.delta
        #get the system matrix fro graph G1 and G2
        system_matrix = self.calculate_system_matrix(beta,delta)
        #get the maximum eigen value and vector for the system matrix
        eigen_value,eigen_vectors = linalg.eigh(system_matrix,eigvals=(self.vertices1 - 1,
self.vertices1 - 1))
        prinicipal_eigen_vector = eigen_vectors[0:]
        prinicipal_eigen_vector = prinicipal_eigen_vector.tolist();
        vector_value = [prinicipal_eigen_vector[i][0] for i in range(self.vertices1)]
        vector_value = list(vector_value)
        #convert vector entries to absoulte value
        x = np.array(vector_value)
        x = np.absolute(x)
        vector_value = x.tolist()
        nodes_ordered_ascending = list(np.argsort(vector_value))
        nodes_to_delete = nodes_ordered_ascending[-k:]
        #delete the k nodes from both graphs
        self.graph1.delete_vertices(nodes_to_delete)
        self.vertices1 = np.size(self.graph1.vs())
        self.graph2.delete_vertices(nodes_to_delete)
        self.vertices2 = np.size(self.graph2.vs()))
```

This algorithm is dominated by matrix multiplication used find the system matrix and also the eigen vector for the system matrix. Hence it ahs a run time of $O(m^3)$

d. Given k = k1, β = β1, and δ = δ1, calculate the effective strength (s) of the virus on the immunized contact network (i.e., contact network without immunized nodes). Did the immunization policy prevented a network-wide epidemic?

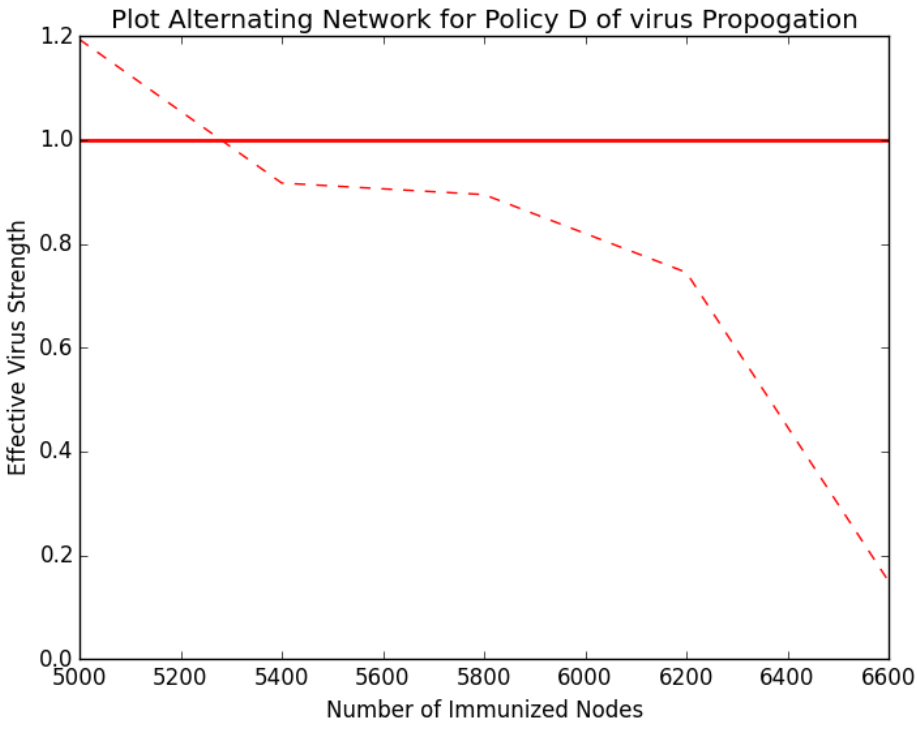
Run the cpde Policy_D_d.py from Alternate Graph folder of the proejct

```
Effective Strength of the virus before immunization is 12.5299130745
Effective Strength of the virus is after immunization 6.96108849748
```

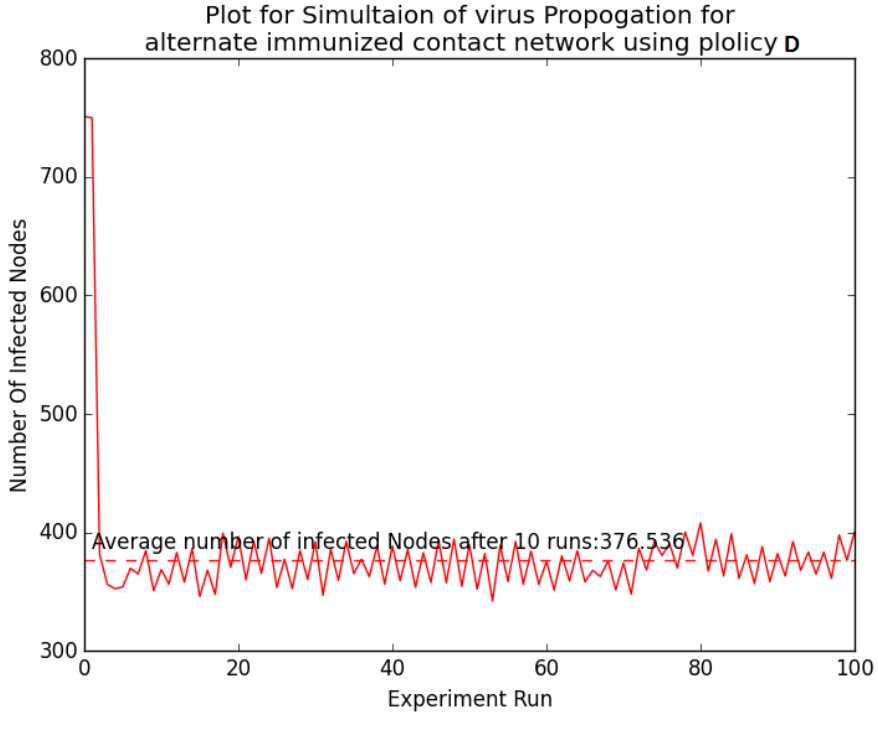
Since the effective virus strength is > 1 , the network is still epidemic.

e. Keeping β and δ fixed, analyze how the value of k affects the effective strength of the virus on the immunized contact network (suggestion: plot your results). Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic.

Minimum number of vaccines required is close to 5300 to immunize the alternate network as seen from the plot below.



f. Given k = k1, β = β1, δ = δ1, c = n/10, and t = 100, run the simulation from problem (2) for the immunized contact network 10 times. Plot the average fraction of infected nodes at each time step. Do the results of the simulation agree with your conclusions in (3d)?
As we see from the plot the network doesn't seem to be strongly epidemic. At the same time it is not completely contained. There are on an average 376 nodes still infected at the end of simulation.



Ideally I would expect policy D to outperform other policies. But for some reason policy D is giving poor results as compared to policy C or B. Possible reasons might be inaccuracy of the library to calcualte 'eigen vectors corresponding to maximum eigen value' limited by the machine(Policy_D_e.py,Policy_D_f.py).It might be a good idea to use other library or programming language to debug this further and see whether this is expected behavior.
