Name: Rakesh Gopal Kavodkar
UnityID: rgopalk

NOTE: The code for this project is done in Python. Please note that for the ease of execution the source code is split on problem basis. Each problem has a python file associated with it. The name of the corresponding files are mentioned below in the problem

1. ***The code for this problem is in the code directory under the name problem_1.py. The program expects command line arguments. Please read README.txt. The Eigen calculation takes a LOT of time.***

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

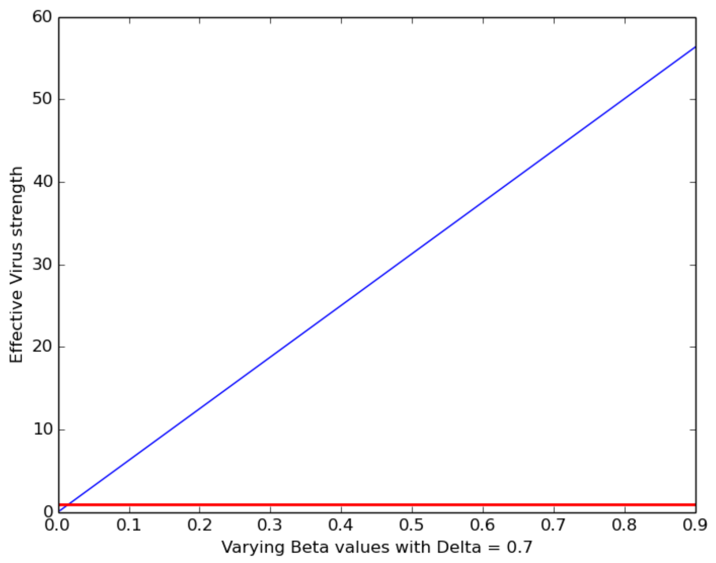
```
Highest Eigen Value:  [ 43.85469576]
With parameter values beta = 0.2 and delta = 0.7
Effective Virus Strength: [ 12.52991307]
```

The Effective Virus Strength is **12.5299**. Hence, we can say that it will result in a 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?**

```
Beta value for Threshold = 1 is [ 0.0159618]
```

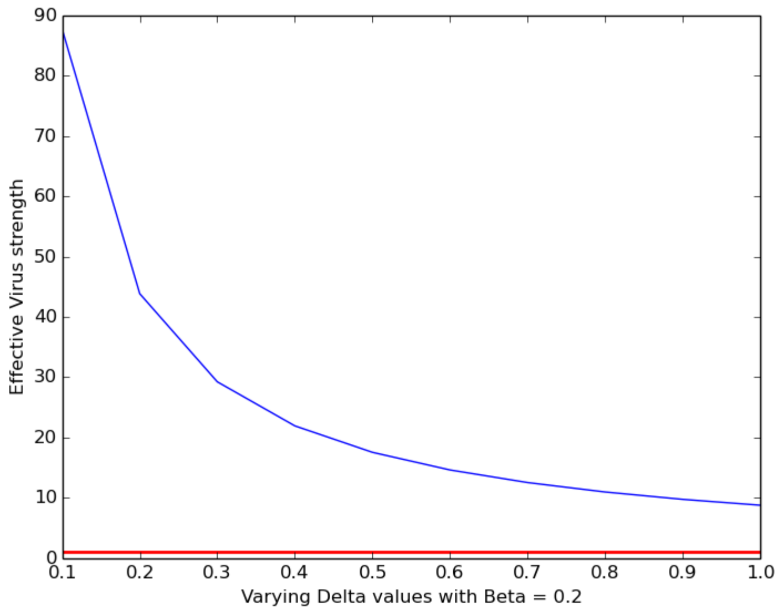
With the given parameters and Eigen Value = 43.85469576, the minimum transmission probability will be **0.0159618**



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?**

```
Delta value for Threshold = 1 is [ 8.77093915]
```

With the given transmission probability and Eigen Value = 43.85469576, we get the maximum healing probability as **8.77093915.** But since it is a probability value, it cannot be more than 1, meaning that for the given beta value, no value of delta can contain the virus.



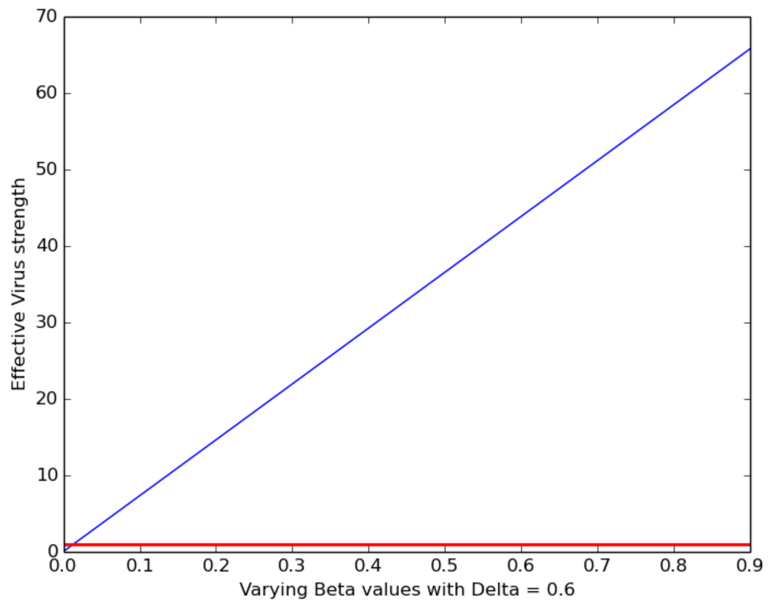
d. **Repeat (1), (1a), (1b) and (1c) with β = β2, and δ =δ2**

```
With parameter values beta = 0.01 and delta = 0.6
Effective Virus Strength: [ 0.7309116]
```

The Effective Virus Strength is 0.7309116. Hence, we can say that the virus can be contained

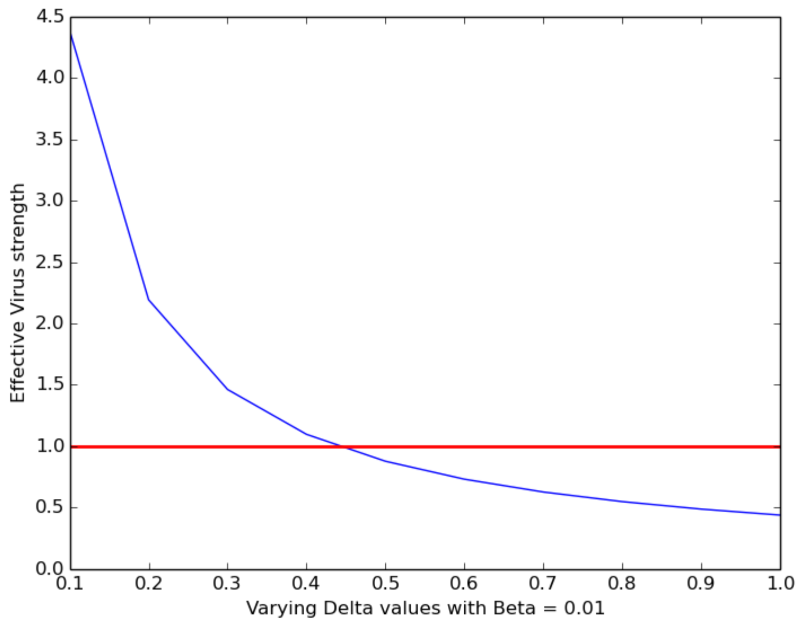
```
Beta value for Threshold = 1 is [ 0.01368155]
```

With the given parameters and Eigen Value = 43.85469576, the minimum transmission probability will be **0.01368155**



```
Delta value for Threshold = 1 is [ 0.43854696]
```

With the given transmission probability and Eigen Value = 43.85469576, we get the maximum healing probability as **0.43854696**



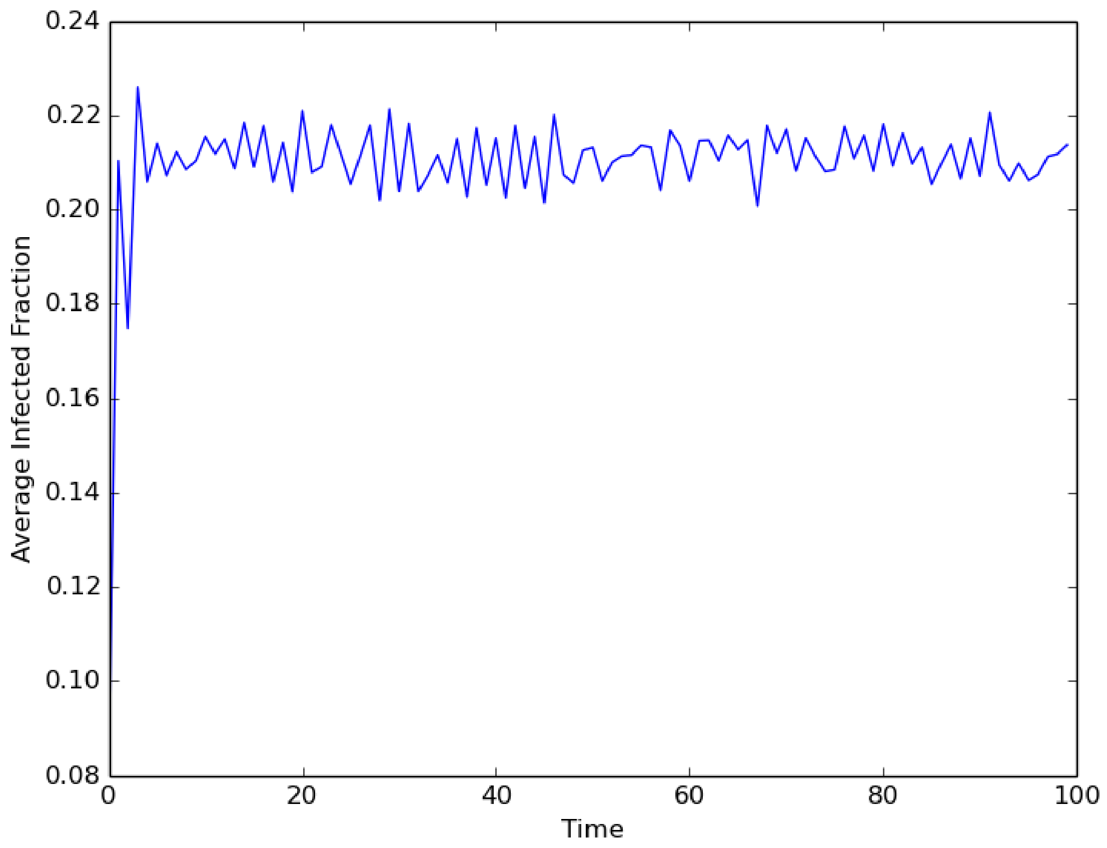
2. ***The code for this problem is in the code directory under the name problem_2.py. The program expects command line arguments. Please read README.txt***

a. **Run the simulation program 10 times for the static contact network provided with β = β1, δ = δ1, c=n/10 and t=100**

Please run the program to see the simulation. The entire simulation for 10 rounds might take 5-10 minutes depending on the processing speed.

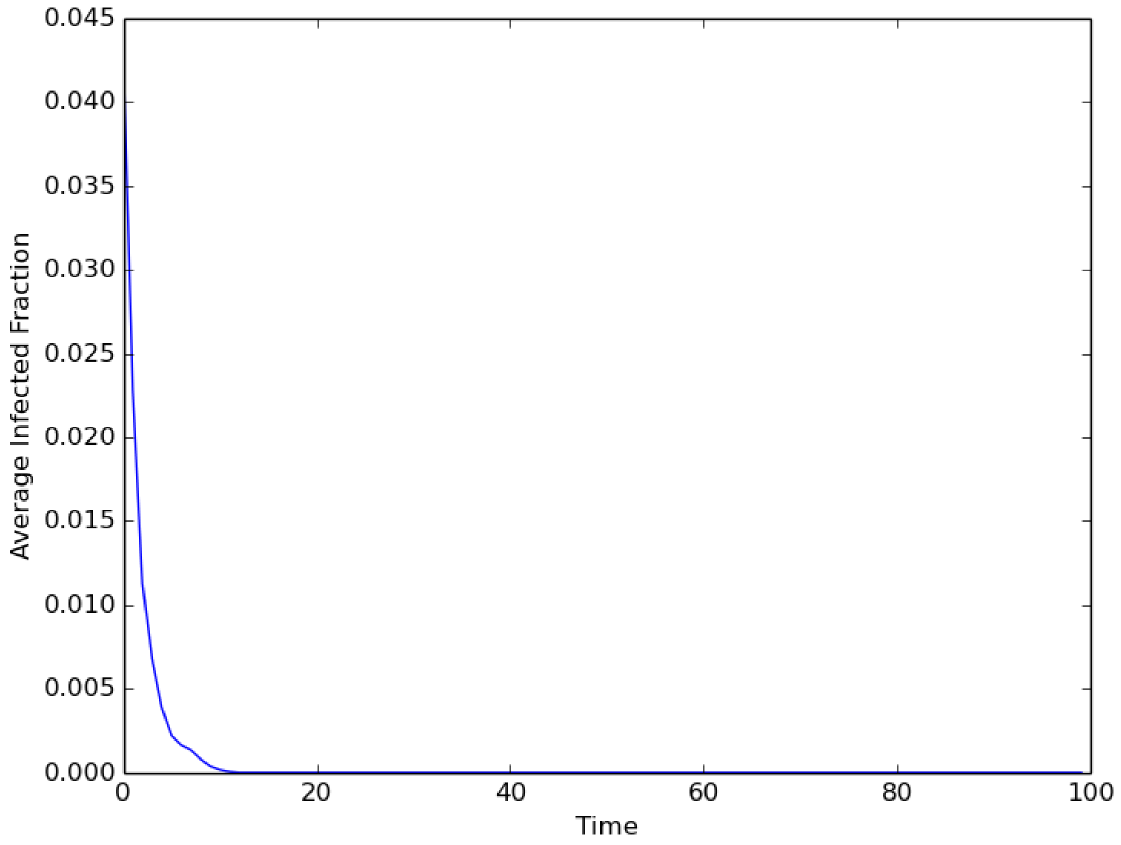
b. **Plot the average(over the 10 simulations) fraction of infected nodes at each time step. Did the infection spread across the network, or did it die quickly? Do the results of the simulation agree with your conclusions in (1a)?**

We see that the virus did not die. It spread across the network. It is in agreement with the conclusion in 1a, ie, it will result in an epidemic.



c. **Repeat (2a) and (2b) with δ=δ2 and β = β2**

We see that the virus died quickly. Again, this is in agreement from our conclusion in 1d, ie, it can be contained.



3. ***The code for this problem is in the code directory under the name problem_3.py and problem_3_effective_strength.py. The program expects command line arguments. Please read README.txt. The policies are enumerated as follows A->1, B->2, C->3, D->4.* Note that running either of the aforementioned files for simulation will take a lot of time.**

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**

As per the lecture, an optimal solution would be to choose *k* nodes that result in the maximum *Eigen Drop* in the Largest Eigen Value. The time complexity of this algorithm is extremely high since calculating the Eigen Drop for every subset of size k is a very costly operation

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

Policy A: The logic is pretty straight forward. Select any *k* number of nodes for immunization randomly. This will not, in most cases, contain the virus unless the value of k is very high. However, since this is randomly chosen, there is always a ***chance*** that the best k nodes are chosen.

Policy B: The logic here is to select *k* nodes with the highest degree in the given network. The rationale behind this is that if we immunize the highly connected nodes, then the connectivity (direct or indirect) between the rest of the nodes will be lesser than before (ie, the average degree will come down). With this, we hope that the virus propagation is slowed down

Policy C: There is subtle difference in this policy and in B. Instead of removing the highest degree nodes at once, we do something similar in iterations. In the first iteration, we remove the highest degree node, and recalculate the degrees of the rest of the nodes. In the next iteration, the corresponding highest degree node is removed, and degrees are recalculated. And, so on. By this, we get lower average degree as compared to the previous one. However, it is to be noted that this is computationally much costlier than the previous policy.

Policy D: This is a subdued version of the NetShield algorithm. We assume that removing the nodes corresponding to the positions of largest values in the eigenvector that corresponds to the largest eigenvalue, will give us a decent (and large) eigen drop. This will reduce the value of the Effective Virus Strength, thereby reducing the spread. Note that although this method might give a good result, it is computationally very costly.

c. **Write a pseudocode for this heuristic immunization policy. What is the time complexity?**
Please find the psuedo-code for the policies below

Policy A:

```
# Get K random nodes for vaccines
nodes_for_vaccines = random.sample(range(0, no_of_vertices), no_of_vaccines)

# Remove the edges of the nodes in the nodes_for_vaccines list
removes_edges(adjacency_matrix, nodes_for_vaccines)
```

Policy B:

```
# Calculate the degree of each node
for i in range(no_of_vertices):
      node_degrees[i] = adjacency_matrix[i].count(1)

# Counter for no_of_vaccines
count = no_of_vaccines
```

```
# In this loop, get K nodes with highest degrees
while count > 0:
       count -= 1
       # Get the highest degree node in the current iteration
       highest_degree_node = node_degrees.index(max(node_degrees))

       # Append the highest degree node to the vaccination list
       nodes_for_vaccines.append(highest_degree_node)

       # Set that node's degree to -1 since it is already considered
       # and shouldn't interfere in the successive iterations
       node_degrees[highest_degree_node] = -1
# Remove the edges of the nodes in the nodes_for_vaccines list
removes_edges(adjacency_matrix, nodes_for_vaccines)
```

Policy C:

```
# Counter for no_of_vaccines
count = no_of_vaccines
# In this loop, get K nodes with highest degrees
while count > 0:
       count -= 1
       # Calculate the degree of each node in each iteration
       for i in range(no_of_vertices):
       node_degrees[i] = adjacency_matrix[i].count(1)

       # Get the highest degree node in the current iteration
       highest_degree_node = node_degrees.index(max(node_degrees))

       # Append the highest degree node to the vaccination list
       nodes_for_vaccines.append(highest_degree_node)

       # Remove the edges of the corresponding node
       removes_edges(adjacency_matrix, [highest_degree_node])

# Restore the adjacency matrix to its original content
adjacency_matrix = original_adjacency_matrix

# Remove the edges of the nodes from the original adjacency matrix
removes_edges(adjacency_matrix, nodes_for_vaccines)
```

Policy D:

```
# Get the highest_eigen_value and its corresponding eigen vector
highest_eigen_value,eigen_vector=linalg.eigh(adjacency_matrix,
       eigvals=(no_of_vertices-1, no_of_vertices-1))
```

```
vector = []

# Convert the values in the eigen vector to their absolute values
for i in range(len(eigen_vector)):
       eigen_vector[i] = math.fabs(eigen_vector[i])
print "Eigen Vector", eigen_vector

for i in range(len(eigen_vector)):
       vector.append(eigen_vector[i][0])

# Counter for no_of_vaccines
count = no_of_vaccines

# Get the indices of the K largest values in the eigen vectors
# These indices correspond to the indices of the nodes to immunize
while count > 0:
       count -= 1
       # Get the highest degree node in the current iteration
       highest_degree_node = eigen_vector.index(max(eigen_vector))

       # Append the highest degree node to the vaccination list
       nodes_for_vaccines.append(highest_degree_node)

       # Set that node's degree to -1 since it is already considered
       # and shouldn't interfere in the successive iterations
       eigen_vector[highest_degree_node] = 0

# Remove the edges of the nodes in the nodes_for_vaccines list
removes_edges(adjacency_matrix, nodes_for_vaccines)
```

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

For policy A, we see that the Virus Strength is way more than threshold. The virus cannot be contained

```
Policy A
No. of vaccines: 200
Highest Eigen Value:  [ 43.2483577]
With parameter values beta = 0.2 delta = 0.7
Effective Virus Strength: [ 12.35667363]
```

For policy B, we see that the Virus Strength is almost equal to 1. Hence, the virus should be contained.

```
Policy B
No. of vaccines: 200
Highest Eigen Value:  [ 3.78096418]
With parameter values beta = 0.2 delta = 0.7
Effective Virus Strength: [ 1.08027548]
```

For policy C, we see that the Virus Strength is almost equal to 1. Hence, the virus should be contained.

```
Policy C
No. of vaccines: 200
Highest Eigen Value:  [ 3.79582189]
With parameter values beta = 0.2 delta = 0.7
Effective Virus Strength: [ 1.08452054]
```

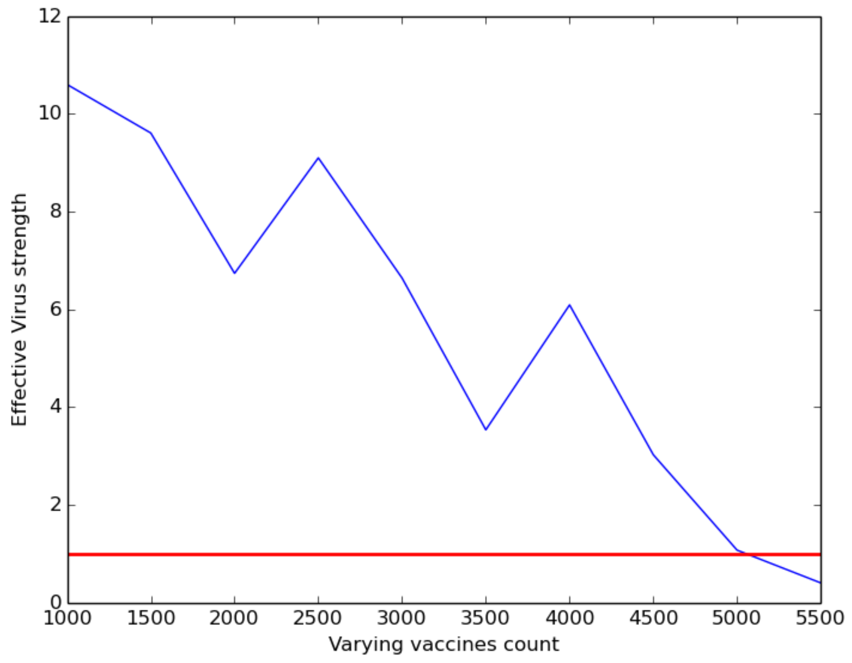
For policy D, we see that the Virus Strength is more than the threshold value of 1. Hence, the virus cannot be contained.

```
Policy D
No. of vaccines: 200
Highest Eigen Value:  [ 10.74684759]
With parameter values beta = 0.2 delta = 0.7
Effective Virus Strength: [ 3.07052788]
```

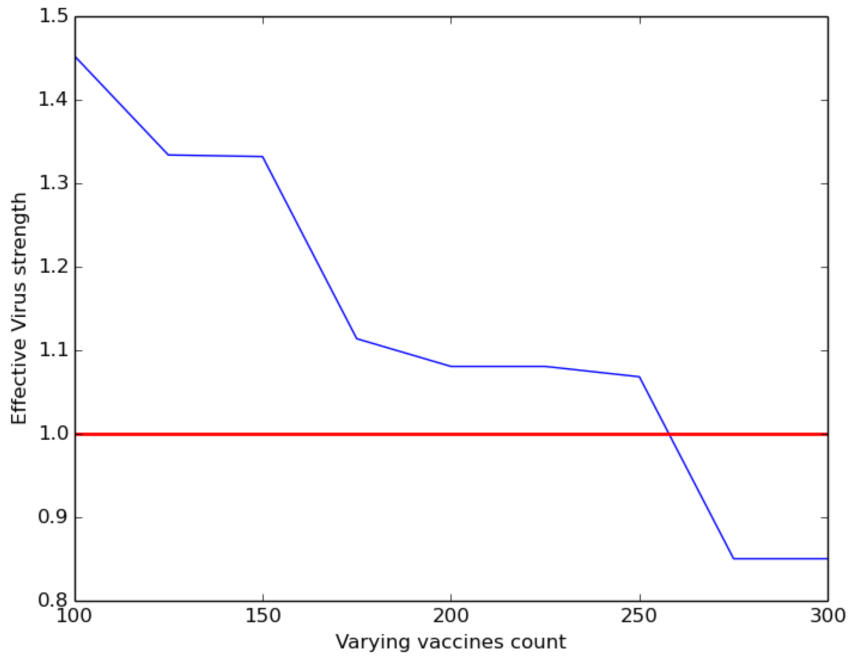
e. **Keeping β and δ fixed, analyse how the value of K affects the effective strength of the virus on the immunized contact network (suggestion: plot your results)> Estimate the minimum number of vaccines necessary to prevent a network-wide epidemic**

NOTE: Logs containing the iteration wise values of Virus Strengths and vaccine count are in the directory *Simulation Logs*

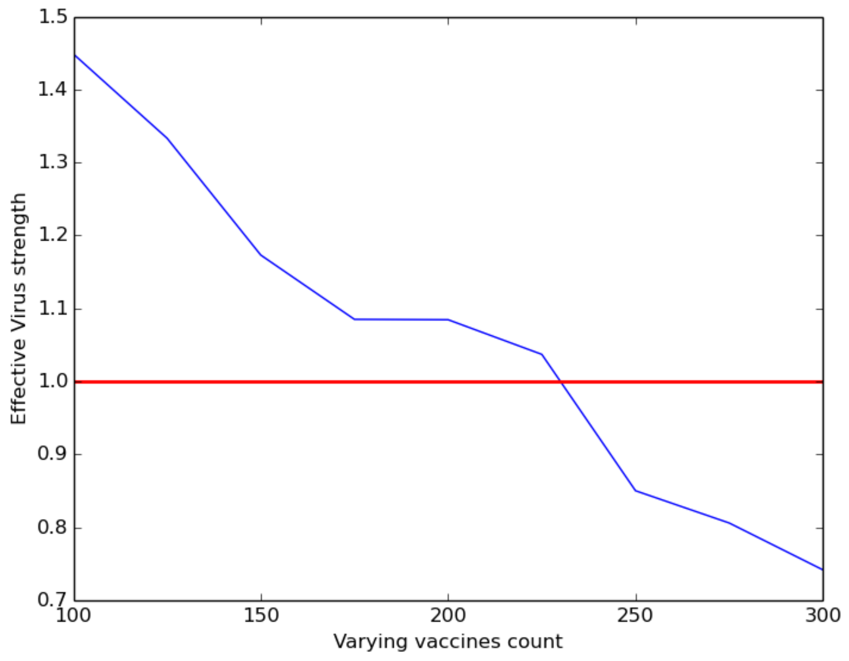
Policy A: The minimum number of vaccines necessary using this policy to avoid epidemic will be in the range of 5000-5200 as seen in the below plot. Note that the number may vary since the nodes are randomly chosen



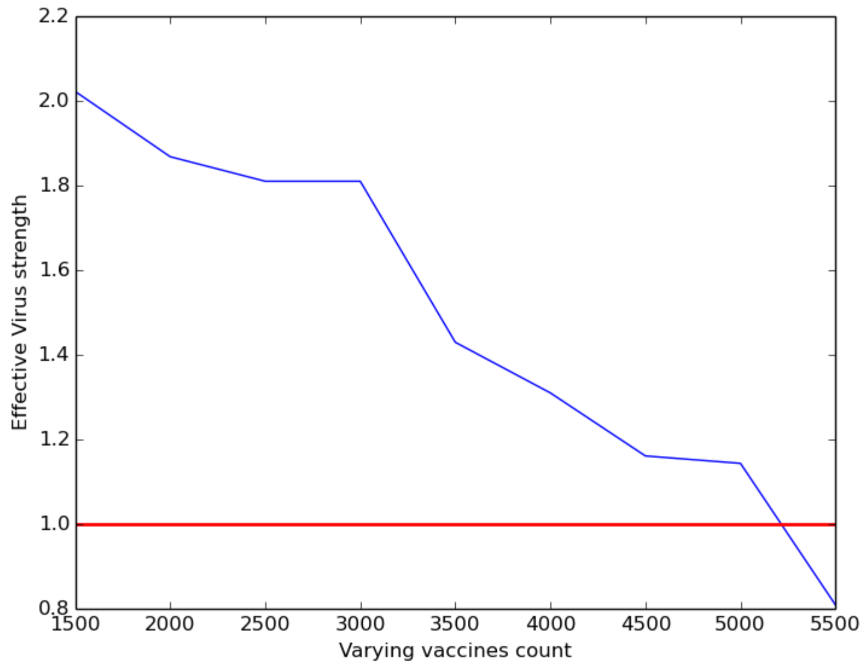
Policy B: The minimum number of vaccines required to contain the virus propagation is in the range of 250-275 as seen in the below plot



Policy C: The minimum number of vaccines required to contain the virus propagation is in the range 225-240 as seen in the below plot



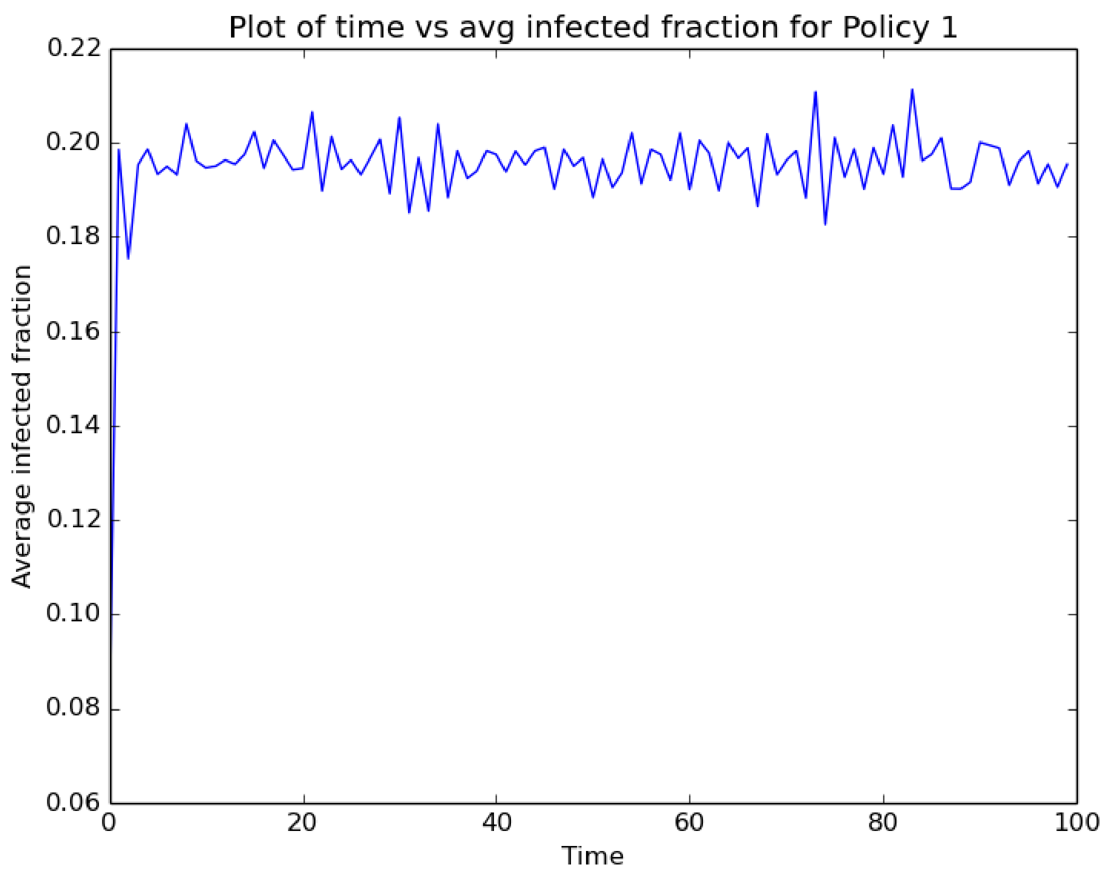
Policy D: The minimum number of vaccines required to contain the virus propagation is in the range 5200-5300 as seen in the below plot



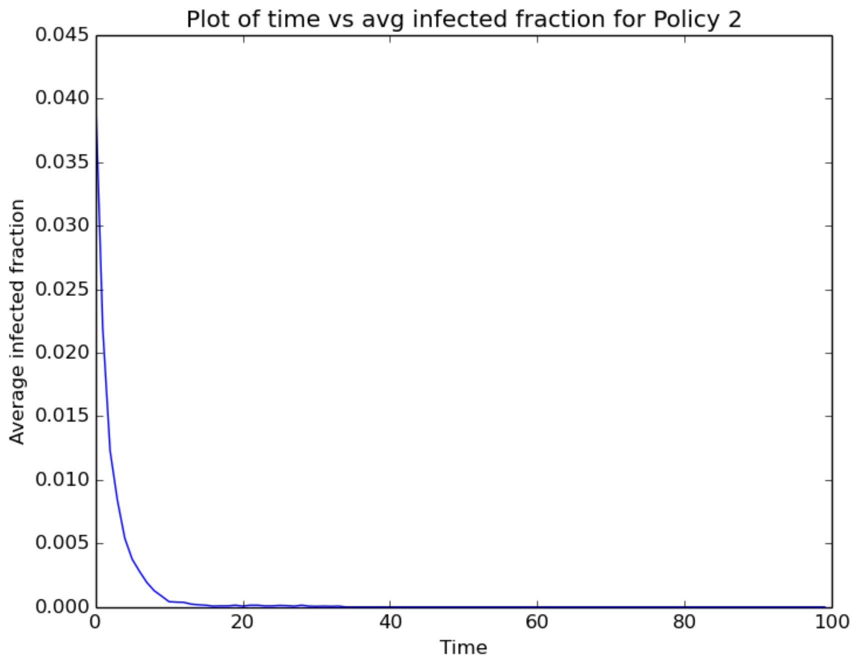
NOTE: The above result is against the fact that Policy D is better suiting for immunization as compared to the other policies. We see that this policy performs worse than B and C, requiring more than 5000 vaccines

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)?

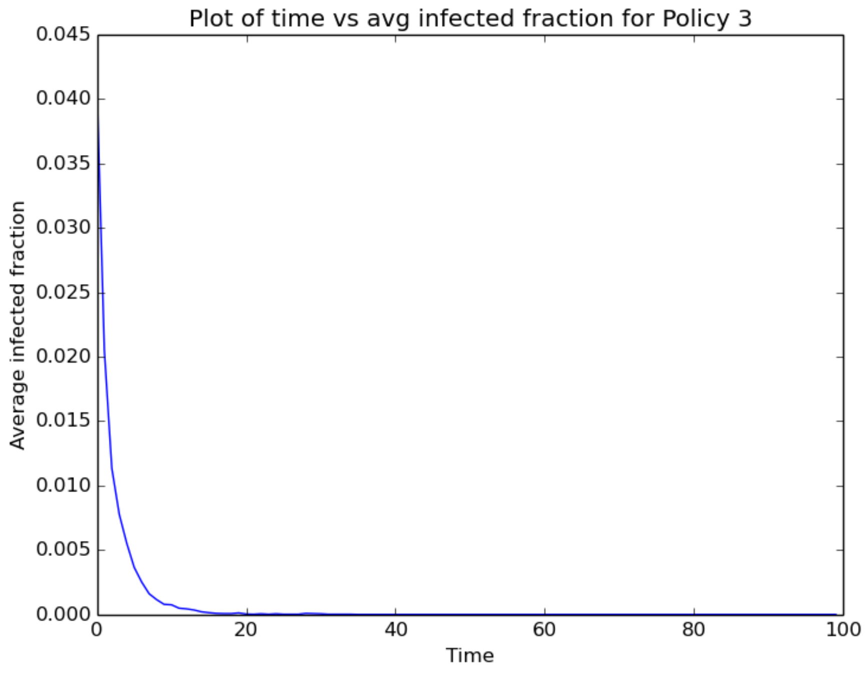
Policy A: From the plot below, we can see that the virus was not contained. This is in agreement with our conclusion in 3d



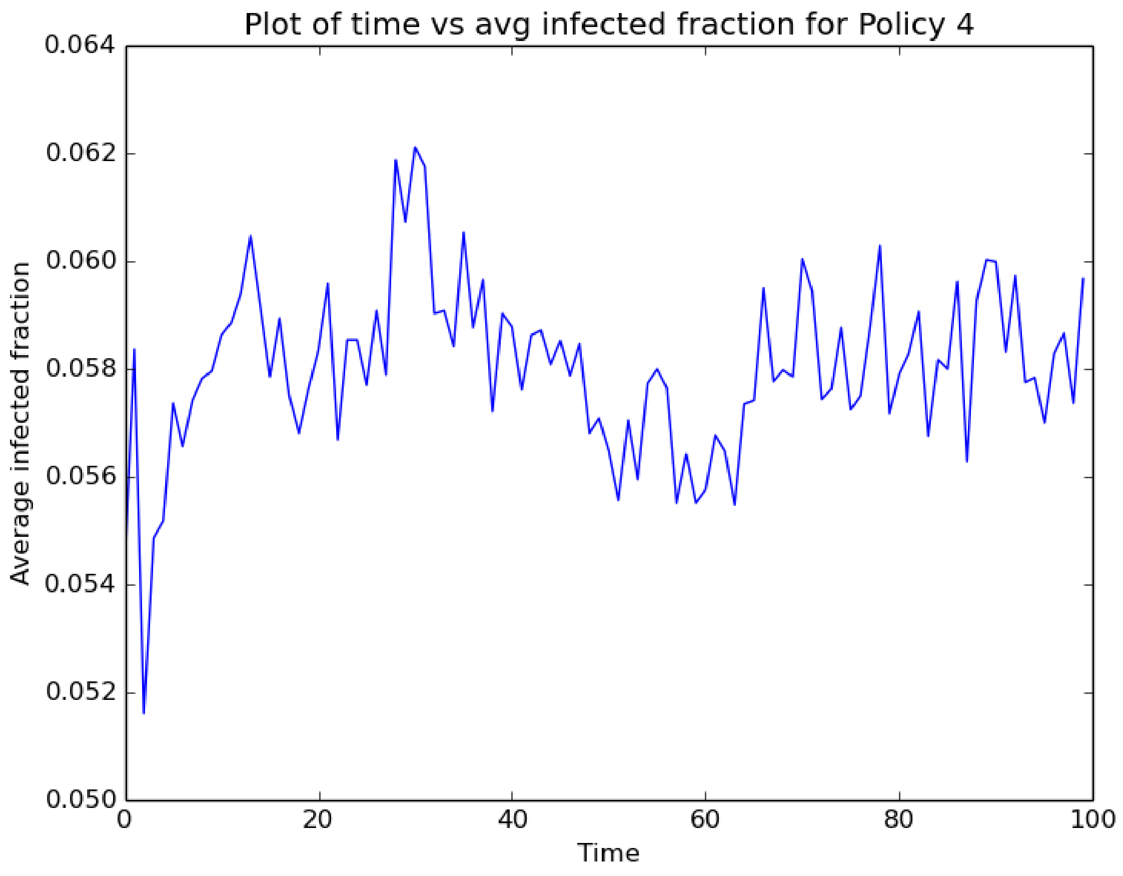
Policy B: From the plot below, we can see that the virus was contained. This is in agreement with our conclusion in 3d



Policy C: From the plot below, we can see that the virus was contained. This is in agreement with our conclusion in 3d



Policy D: From the plot below, we can see that the virus was not contained. This is in agreement with our conclusion in 3d


Plot of time vs avg infected fraction for Policy 4
Average infected fraction
Time