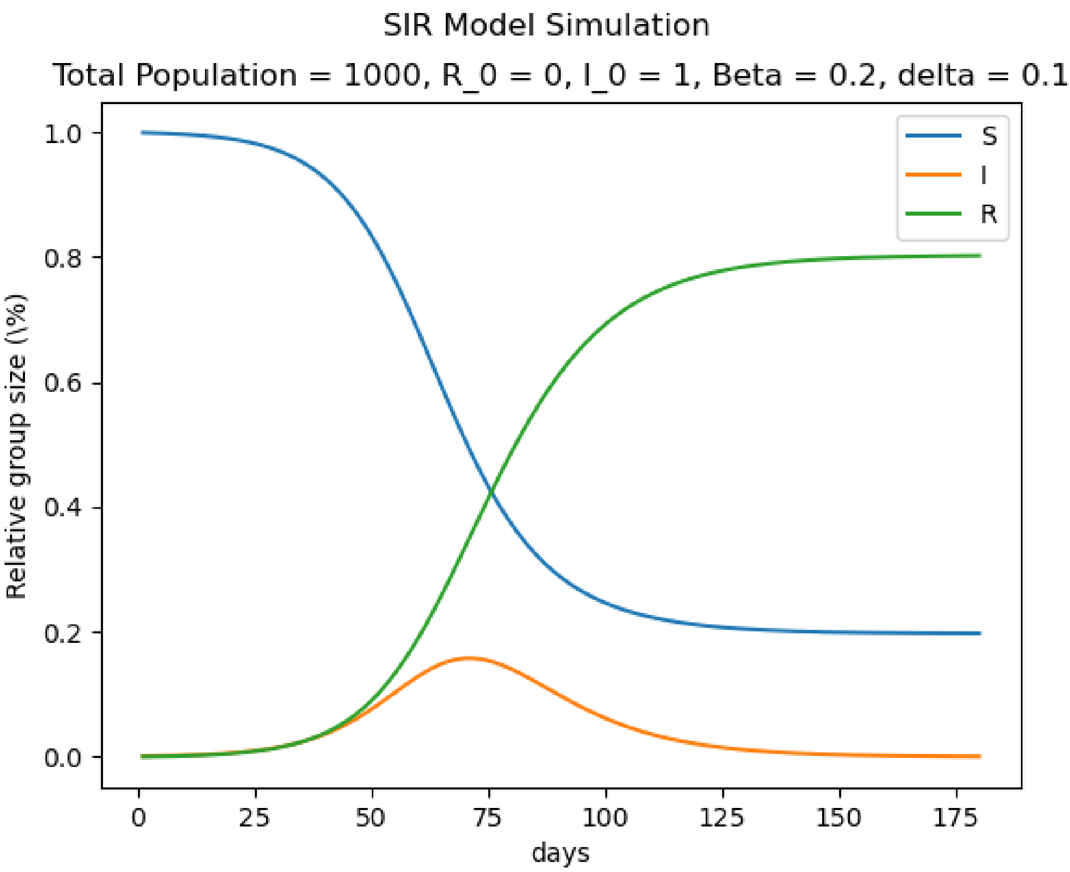
VE444 Networks

Homework 3

FA 2020

Name	Stud. ID
Zhou Zhanpeng	*518021910594*

Question 1



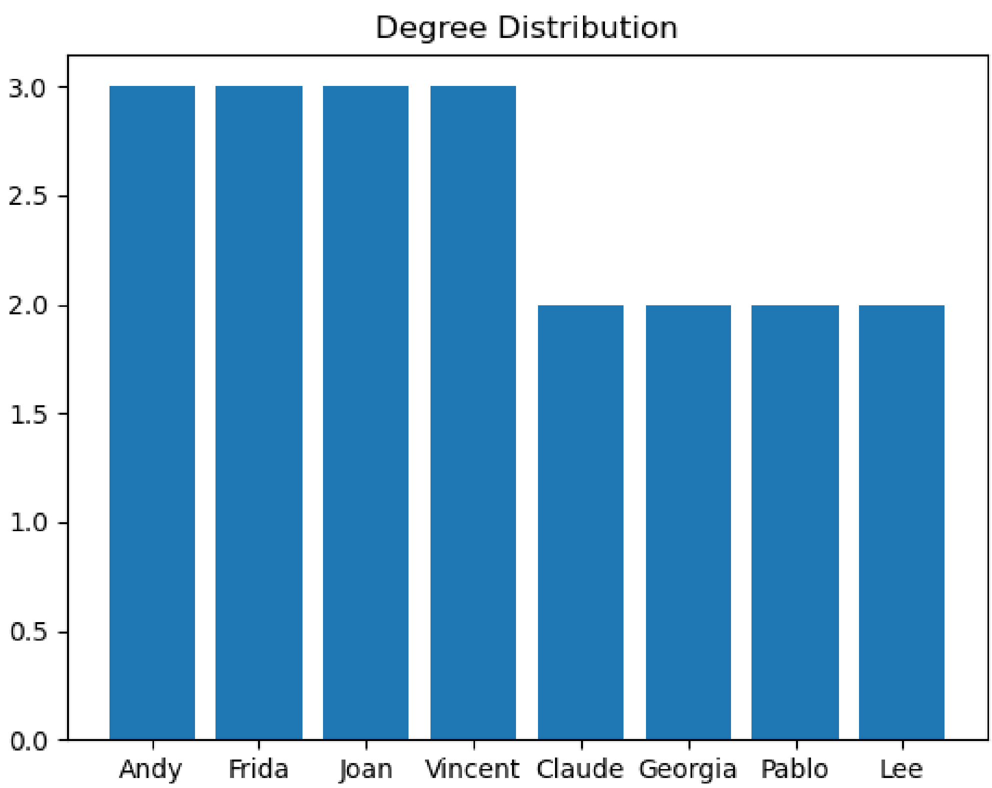
Following is the core part of the code.

```
# init
S, I, R  = [(args.total - args.recovery - args.infected) / args.total], [args.infected / args.total], [args.recovery / args.total]
while round(I[-1] * args.total) > 0:
    # solve the S, I, R for each iteration
    dS = - args.beta * S[-1] * I[-1]
    dR = args.delta * I[-1]
    dI = 0 - dS - dR
    S.append(S[-1] + dS)
    R.append(R[-1] + dR)
    I.append(I[-1] + dI)
```

Simply, we are iterating through every day to update the `S`, `I`, `R` until there is no infected person.

Question 2

1)



Simply, we manually calculate the degree for each node. And there is a quick package in python which can calculate the degree very easily, called `networkx`. Following is the core part of the code.

```
# construct the graph
graph = nx.Graph()
graph.add_edges_from(friends_df.values)
# calculate the degrees for each node.
degrees = [graph.degree(i) for i in graph.nodes()]
```

2)

The set includes

- Anaconda;
- The Shawshank Redemption;
- Forrest Gump.

In this exercise, we simply follow the procedure in lecture slides keeping to find the next node to maximize the current influence until all the nodes are covered.

```
influ_set, current = list(), set()
while current != target and influ_set != list(groups.keys()):
    size, tmp = len(current), None
    for key in groups.keys():
        if key not in influ_set and size < len(current.union(groups[key])):
            size, tmp = len(current.union(groups[key])), key
    if tmp:
        influ_set.append(tmp)
        current = current.union(groups[tmp])
return influ_set
```

Question 3

1)

$$\begin{bmatrix} 0. & 1. & 0. & 0. & 1. & 0. \\ 0. & 0. & 1. & 1. & 0. & 0. \\ 0. & 0. & 0. & 1. & 1. & 1. \\ 1. & 0. & 0. & 0. & 0. & 0. \\ 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. \end{bmatrix}$$

Still, quite simple and no further explaination. Following is the core part of the code.

```
# get edges from reading file
edges_df = pd.read_table(filepath, header = None, sep = " ")
# get the number of nodes
num_of_nodes = np.max(edges_df.values)
# init adjacency matrix
adj_mat = np.zeros((num_of_nodes, num_of_nodes))
for i, j in edges_df.values:
    adj_mat[i - 1, j - 1] = 1
```

2)

$$\begin{array}{cc} 1 & [0.32075476] \\ 2 & [0.17070158] \\ 3 & [0.10638908] \\ 4 & [0.13671355] \\ 5 & [0.20102606] \\ 6 & [0.06441497] \end{array}$$

For this part, we should resolve two potential problems:

1. spider trip.
2. dead end.

For spider trip, we set a random teleport probability to let each node have the chance to jump to all the other nodes in the graph. For dead end, we simply uniformly distribute the probability of jumping to all the other nodes for the dead ends nodes. With these two ideas, we can find the google matrix and simply use power iteration to find the rank scores for each node. Following is the equation we used in iteration.

$$\mathbf{A} = \beta\mathbf{M} + (1-\beta)[\frac{1}{n}]_{N\times N}$$
$$r = \mathbf{A}\cdot r$$

Here is the core part of the code.

```
# number of nodes
num_of_nodes = adj_mat.shape[0]
# create the stochastic matrix=
M_mat = np.transpose(adj_mat)
# find the dead ends
dead_ends = np.where(np.sum(M_mat, axis = 0) == 0)[0]
# solution to dead ends
if dead_ends:
    M_mat[:, dead_ends] = 1
M_mat = M_mat / np.sum(M_mat, axis = 0)
# solution to spider traps
A_mat = beta * M_mat + (1 - beta) * (np.ones(M_mat.shape) / num_of_nodes)
# init the rank_lst
rank_lst = np.ones((num_of_nodes, 1)) / num_of_nodes
# constantly updating rank_lst
for i in range(max_itr):
    new_rank_lst = np.dot(A_mat, rank_lst)
    if np.linalg.norm(rank_lst - new_rank_lst) > epsilon:
        rank_lst = new_rank_lst
    else:
        break
return rank_lst
```