# Homework 3

Vincenzo Marciano' (282004)

January 15, 2022

**Abstract**

In the third and final Homework of Network Dynamics and Learning, we are supposed to analyze a SIR evolution for a particular pandemic in Sweden in 2009, the Influenza H1N1: the key idea here is to exploit the Epidemics Dynamics of the given network.

The exercises are made in collaboration with Alessandro Pecora (290369) and Andrea Ferretti (289677).

## Exercise 1

In this first task, we have to build a graph $\mathcal{G}$ with 500 nodes in which every node is directly connected to the $k = 4$ nodes whose index is closest to their own modulo $n$. Also, the system is called $SIR$ as the following index suggests:

- **S** Susceptible;
- **I** Infected;
- **R** Recovered.

### 1.1

This exercise let us dealing with a SIR environment. For this reason the main ingredients are:

- an undirected graph $\mathcal{G}$, which features were described before;
- set of parameters $\beta = 0.3$ (probability that the infection is spread from an infected individual linked to a susceptible one) and $\rho = 0.7$ (the probability that an infected individual will recover during one time step);
- controlling equation of the studied epidemic:

$$\mathcal{P}(X_i(t+1) = I | X_i(t) = S, \sum_{j \in \mathcal{V}} W_{ij} \delta^{I}_{X_j(t)} = m) = 1 - (1-\beta)^m$$

$$\mathcal{P}(X_i(t+1) = R | X_i(t) = I) = \rho$$

Another tool that will be useful is a set of builded function that help us in the various calculations, as it's declared in the Listing 1. In particular:

- *createInitialState* provides the starting point from which the simulation begins, i.e. the first state with $n$ infected;
- *getSIRcount* simply gets the total number of S, I and R individuals respectively;
- *countNeighbours* returns the number of a given node's neighbours set in the specified state generated by *createInitialState*

```python
def createInitialState(graphSize, NInitialInfected, init_node_list = None):
    state = np.zeros(graphSize)
    if init_node_list is not None:
        for i in range(graphSize):
            if i in init_node_list:
                state[i] = 1
    else:
        startingINodes = np.random.randint(0, graphSize, size=NInitialInfected)
        for i in range(graphSize):
            if i in startingINodes:
                state[i] = 1
    return state

def getSIRCount(state):
    count = collections.Counter(state)
    countI = count[1]
    countR = count[2]
    countS = count[0]
    return [countS,countI,countR]

def countNeighbours(G, node, state):
    count = 0
    for i in G.neighbors(node):
        if state[i] == 1:
            count = count + 1
    return count
```

Listing 1: Methods used to experiment the various simulations

Last but not least we build a function, namely *processNextInteractions*, in charge of controlling the process by returning the new state vector, number of Infected and number of Recovered. More details are displayed in the following Listing 2:

```python
def processNextInteractions(G, state, beta, rho):

    nextState=list(state)
    newInfWeekly = 0
    newRecoveredWeekly = 0
    individuals = list(G.nodes)
    np.random.shuffle(individuals)
    for node in individuals:
        if state[node] == 1: #Check if node is currently infected and can eventually recover
            if np.random.rand() < rho: # recover the node (COME SI SETTA rho??)
                nextState[node] = 2
                newRecoveredWeekly += 1
        else:
            if state[node] == 0:
                m = countNeighbours(G, node, state)
                if np.random.rand() < 1-(1-beta)**m:
                    nextState[node] = 1
                    newInfWeekly += 1
    return nextState, newInfWeekly, newRecoveredWeekly
```

Listing 2: Methods used to understand the process in infecting or to be infected by looking at the other nodes state.

Running the simulation for 100 times, the obtained outcomes are illustrated in Figure 1.

## 1.2

The aim of this task is to have a randomly generated graph with average degree close to $k$. Furthermore, one must pay attention to some specific conditions:

- using *preferential attachment* rule, i.e. the new linking node to graph $\mathcal{G}_{t-1}$ at time $t$ will have a degree $c = k/2$;
- if $k$ is odd it could create some problem in generating the the random graph since in this case $c = k/2$ is not an integer. However, this can be solved by alternating between adding the *floor* and *ceiling* functions of $c$.

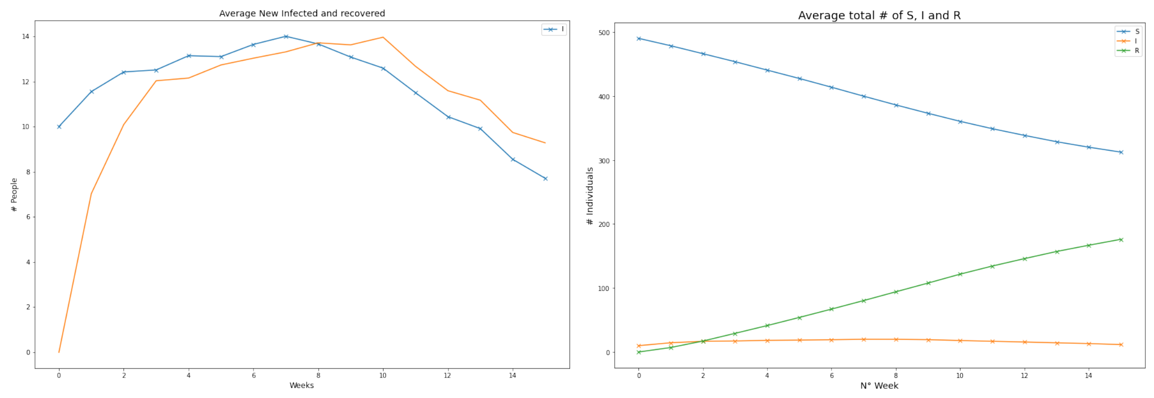

Figure 1: On the left: average number of people infected each week. On the right: total average number of individuals that are S, I or R at each week

To generate the graph, we start from a complete graph with $k + 1$ nodes. At every iteration we create a node with degree $wt = c = k/2$. The whole process relies on a handmade function called *generateRanGraphK* which arguments are the number of nodes $GSize$ of the graph we want to create (in our case $\geq 900$) and the average degree $k \in \mathcal{Z}^+$. The aforementioned function can be retrieved in Listing 3.

```python
def generateRanGraphK(GSize, k):
    RandomG = nx.complete_graph(k + 1)
    for nt in range(len(RandomG.nodes), GSize):
        probabilities = []
        RandomG.add_node(nt)
        c = (k + (nt % 2)) // 2
        totalDegree = np.array(RandomG.degree()).sum(axis=0)[1]

        for node in RandomG.nodes():
            probabilities.append(RandomG.degree[node]/totalDegree)
        selNodes = np.random.choice(RandomG.nodes(), p=probabilities, size=int(c), replace=False)
        for cn in selNodes:
            RandomG.add_edge(nt, cn)
    return RandomG
```

Listing 3: Function that generates a Random Graph

In order to achieve consistent results with the passed $k$, we use a toy example of $GSize = 950$ and $k = 7$ and we compute the mean on the weight matrix obtaining **7.0** as final average degree, confirming its functionalities.

## Exercise 2

In this problem we use the methods created before in order to initialize a preferential attachment random graph $G = (V, E)$, with $|V| = 500$ nodes. The average degree should be $k = 6$, $\beta = 0.3$ and $\rho = 0.7$. We simulate an epidemic for 15 weeks, with an initial configuation of 10 infected nodes selected randomly from the node set $V$.

The following Figure 2 shows the average number of newly infected individuals each week and the average total number of susceptible, infected, and recovered individuals at each week.

## Exercise 3

This problem is the same as before but with the addiction of vaccination: during each week, some parts of the population will receive vaccination. **Once a person is vaccinated it cannot be infected. Furthermore, the vaccination is assumed to take effect immediately once given.**

The total fraction of population that has received vaccination by each week is according to:

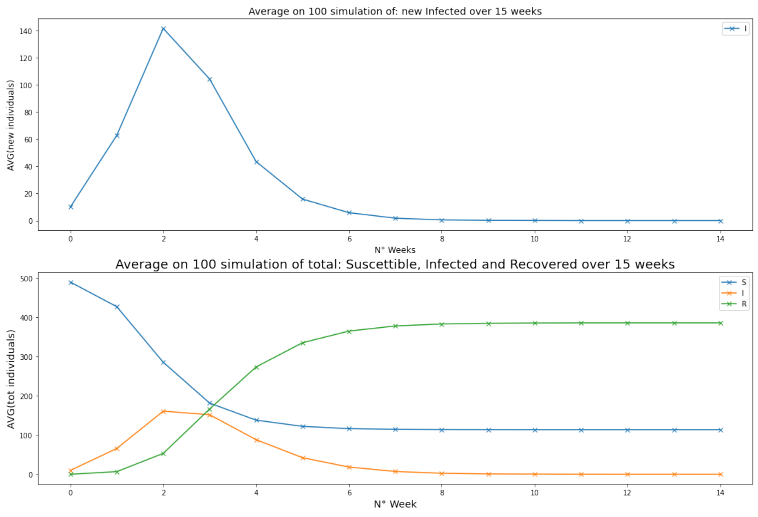

Figure 2: Average number of newly infected and average total number at each week.

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

We will divide this problem in two cases:

1. Simulation with vaccine recovered;
2. Simulation without vaccine recovered.

## 3.1 Simulation with vaccine recovered

In the first simulation we decide to vaccinate the recovered individuals even if they have generated antibodies.

The following figure shows the average number of newly infected and newly vaccinated individuals each week and the average total number of susceptible, infected, recovered and vaccinated individuals at each week.

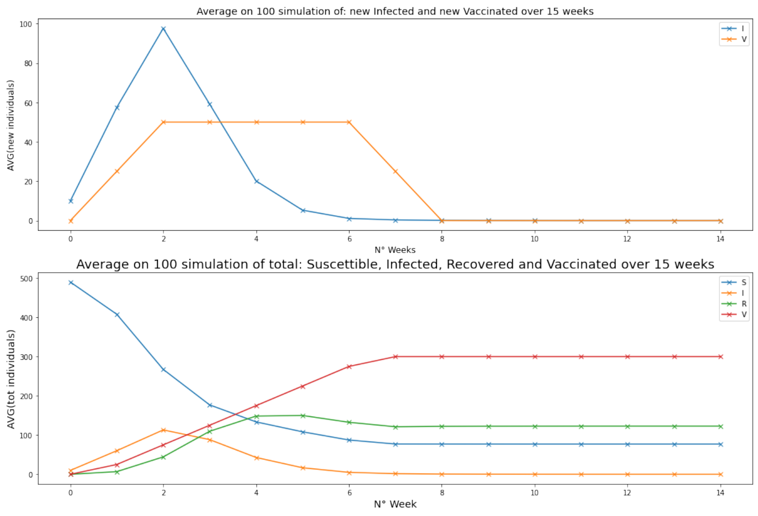

Figure 3: Average number of newly infected and vaccinated, and average total number at each week with recovered individuals vaccinated.

We can see that with respect to the case without vaccination, the pandemic spreads less.

Moreover the number of recovered is lower with vaccination, this can be interpretated as less pressure on the sanitary system.

### 3.2 Simulation without vaccine recovered

In the second simulation we decide to **not** vaccinate the recovered individuals even if they have generated antibodies.

The following figure shows the average number of newly infected and newly vaccinated individuals each week and the average total number of susceptible, infected, recovered and vaccinated individuals at each week.

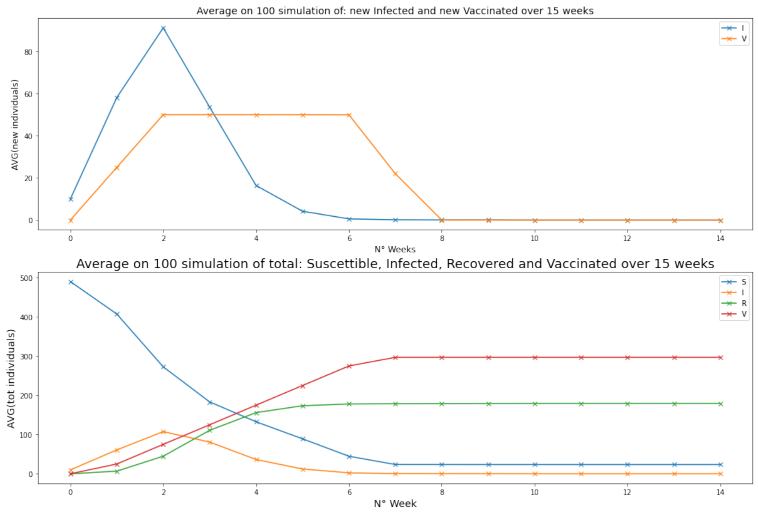

Figure 4: Average number of newly infected and vaccinated, and average total number at each week without recovered individuals vaccinated.

A consideration can be done on this:
If we decide to not vaccinate the recovered individuals, we can see that the new recovered go to zero fastly in the fifth week we have some recovered individual less.

This analisys can be used to make decision in order to prioritizing the vaccination campaign.

Since in our model recovered individuals cant become re-infected prioritizing the vaccination of not recovered individuals can reduce the number of recovered.

## Exercise 4

In this problem, we are interested in estimating the social structure of the Swedish population and the disease-spread parameters during the H1N1 pandemic: During the fall of 2009 about 1.5 million people out of a population of 9 million were infected with H1N1, and about 60% of the population received vaccination.

We will simulate the pandemic between week 42, 2009 and week 5, 2010. During these weeks, the fraction of population that had received vaccination was:

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

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

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

By means of an algorithm that does a gradient-based search, we estimate the average degree $k$ and the disease-spread parameters $\beta$ and $\rho$ for the pandemic.

The best parameters found are:

- $k = \mathbf{12}$
- $\beta = \mathbf{0.09\overline{9}}$
- $\rho = \mathbf{0.4.}$

, ,
With the following Figure 5, we show (1) the average number of newly infected individuals each week according to the model compared to the true value of newly infected individuals each week and (2) the total number of susceptible, infected, recovered and vaccinated individuals at each week according to the model.

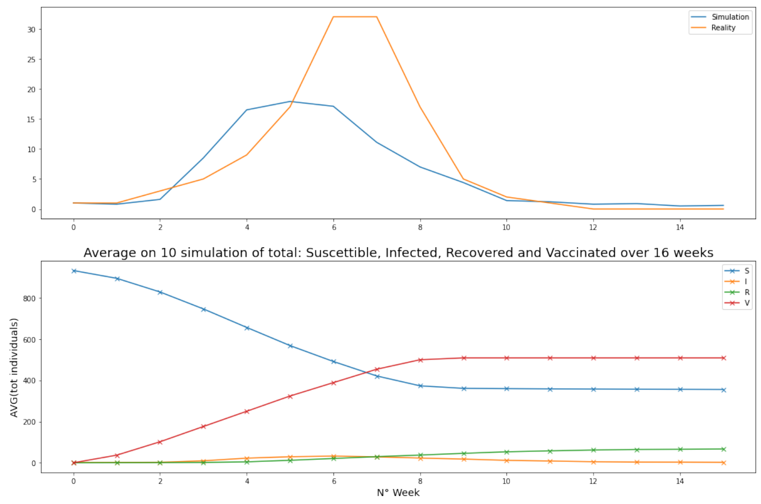

Figure 5: Average number of newly infected and vaccinated according to the model compared to the true value, and average total number at each week without recovered individuals vaccinated.

## Exercise 5

### 5.1 - Erdos-Rényi

The last task is to perform an improvement in representing the pandemic network with a better random graph. For our choice, the **Erdos-Rényi** representation comes in handy.

The key idea behind this type of representation is to find a graph $\mathcal{G}$ such that its chromatic number is $\geq k$, for every cycle of length $\geq g$ and every $k$. Moreover, to model the system another parameter is needed: the probability that an edge is rewired is a parameter, $p$, that controls how complete the graph is. With p=0, the graph is empty; with p=1 it is complete. From here is imperative to state out some important properties to run a correct algorithm:

- $diam(\mathcal{G}) \leq A \log n$;
- distribution of degree of node i

$$\mathcal{P}(w_i = k) = \binom{n-1}{k} p^k (1-p)^{n-1-k}$$

- number of isolated nodes:

$$N_0 = |i : w_i = 0| = \sum \mathbf{1}_{w_i = 0 i \in \mathcal{V}}$$

From this setting, we made a further assumption: as it's stated before, the probability in obtainingthe i-th weight equal to the average degree $k$ is modelled by a binomial function, and for this reason we fixed $p$ as a function of $k$. Hence, we obtain a learnable $p$ that stochastically updates itself by following the mean value of a binomial function, i.e.:

$$p = \frac{k}{N-1}$$

From here each link will be generated with the aforementioned probability $p$. A good overview is then expressed in the following snippet of code:

```python
#avg(deg)=p*(n-1)
def generate_ER(N, k):
  p = k/(N-1)
  # add links between couple of different nodes with probability p
  GER = nx.erdos_renyi_graph(N, p, directed=False)
  return GER
```

Listing 4: New function to use for probability $p$ updating

With this simple yet efficient hypothesis, the process shows a final **RMSE** of **6.85**, achieving both a lower error and a new set of best parameters as it follows:

- $k = \mathbf{11}$
- $\beta = \mathbf{0.1\overline{9}}$
- $\rho = \mathbf{0.5}$

The final result can be seen also in the following Figure

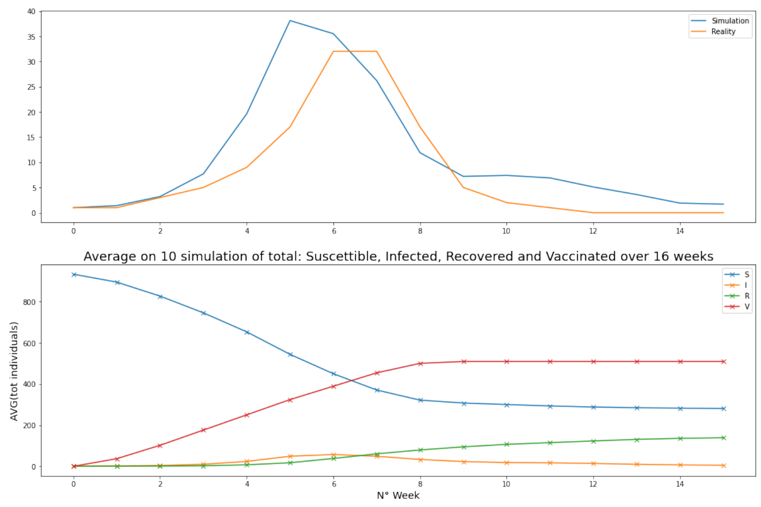

Figure 6: Simulation of the system by exploiting the proposed model.

## 5.2 - Small World

Another conducted experiment was based on **Small-World** representation. Watts and Strogatz exploited two well-known kinds of graph:

- random graph, where nodes are connected at random.
- regular graph, every node has the same number of neighbors

They consider two properties of these graphs, *clustering* and *small diameter*:

- **Clustering**, in a social network for example, is a set of people who are all friends with each other. Watts and Strogatz defined a clustering coefficient that quantifies the likelihood that two nodes that are connected to the same node are also connected to each other. This first element was actually the first idea that leads us to conceive a possible application in a Epidemic-based background.

- **Small Diameter** is a measure of the average distance between two nodes, which corresponds to the degrees of separation in a social network.

After this premises, the final results were quite unsatifying: with a sub-optimal **RMSE** of **10.16** this model is the worst in term of performance w.r.t. the other two. A first hypothesis linked to this behaviour could be related to the unproper situation for which a Small-World application was exploited: since in a pandemic environment the contageous vector spreads quite quickly this does not create the right "small diameter" condition in which a geographic proximity is required (see as an example the Covid-19 diffusion).