[INFO-F409] Learning Dynamics
Second assignment

BUI QUANG PHUONG Quang Linh
Université libre de Bruxelles - ULB ID : 000427796
MA1 Computer Sciences

December 2018

Contents

1 Part I: Complex networks

1.1 1 - Erdos-Renye network

1.1.1 Q1 statement

Generate Erdos-Renye network (Random networks) [1,2]. Generate the network from scratch and present/describe the part of the code you used to generate the network in your document. For a size of the network of N=10000, calculate a K so that each node has in average degree 4.

Pseudo-code

Algorithm 1 Generation of Erdos-Renye network

```
procedure ErdosRenye(n, K)
    Initialize the network edges list
    for each 'future' edges ≡ 0 → K do:
        Pick two random nodes in the network
        Define newEdge variable taking the 2 picked nodes
    if newEdge not already exists then:
        Add newEdge in the list of edges
    return list of network's edges
```

Algorithm description

The Erdos-Renye network is a random network generated. Before all, we have to initialize a list of edges which will be returned and will form the random network. This edges list is composed by some tuples of 2 elements (x, y) representing the edges where x, y are the two nodes that the edge is connecting. Now, we have to keep creating edges while the number of edges K passed in parameters isn't reached. The random part is managed by the step of picking nodes by pair (x, y) in the network which is totally random, all nodes can be picked at this moment. Of course, before accepting the edge between the two nodes chosen, a verification that this edge doesn't already exists, which means that the edge (x, y) **and** (y, x) isn't already created, is done.

K for average degree of 4

To calculate the value of K for an average degree of 4, first we know that an edge generates a link between two nodes and then increases the degree of two nodes. Moreover, we have 10000 nodes, thus we can calculate the value of K for an average degree of 4 using the following formula : $\frac{2K}{10000} = 4$ that gives $K = 20000$.

1.1.2 Q2 statement

Plot the degree distribution of the generated network. Calculate the mean and standard deviation and plot the normal distribution with these same parameters

Degree distribution

The degree distribution of the network generated by the Erdos-Renye algorithm with $K = 20000$ is given by the Figure 1. Obviously, the most present degree is 4 followed by 3 and 5 because we previously set a value of K for which the network has an average degree of 4. Thereby, it follows a normal distribution centralized in 4.

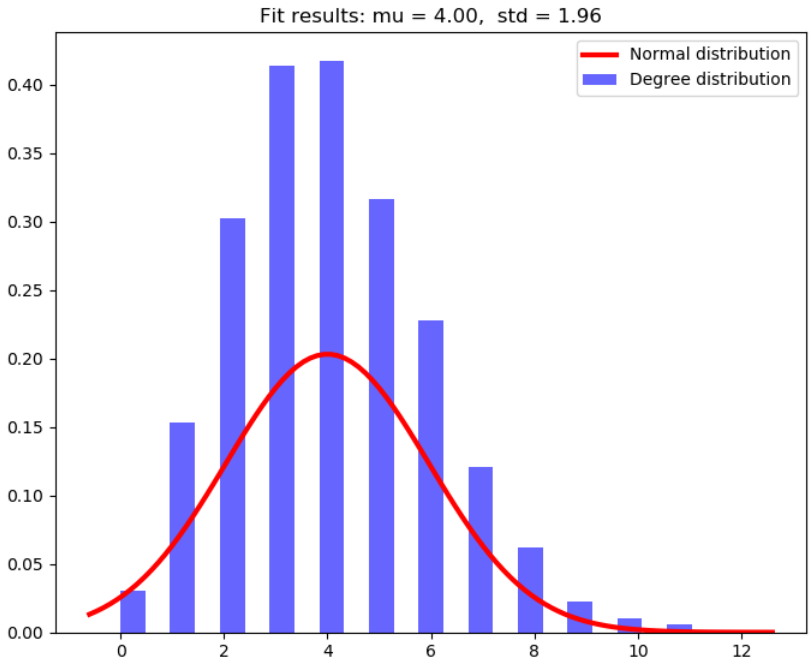

Figure 1 – Distribution for Erdos-Renye generated network

Mean and standard deviation

The value of the mean in the distribution of Figure 1 is

$$\mu = 3.999$$

and for the standard deviation is

$$\sigma = 1.964$$

.

1.2 2 - Barabasi-Albert network

1.2.1 Q3 statement

Generate a Barabasi-Albert network (Scale Free network) [1,3]. Generate the network from scratch and present/describe the part of the code you used to generate the network in your document.

Pseudo-code

Algorithm 2 Generation of Barabasi-Albert network

```
1: procedure BarabasiAlbert(Graph, N)
2:     Initialize a fully connected 4 nodes network
3:     for each 'future' node until N ≡ 4 → N do:
4:         Add a node x to the network
5:         for each current network's nodes do:
6:             Compute the probability of the added node x being linked to those nodes
7:             Append it into a list probasLinking
8:         Pick 4 nodes 'randomly' (with their associated probabilities) in the network's nodes
9:         for the 4 picked nodes do:
10:            Link the added node x to them
11:    Reinitialize probasLinking and increment by 8 (4 new edges = 8 nodes with their degree
       increasing by 1) the total degree number for the computation of the next probabilities
```

Algorithm description

The Barabasi-Albert algorithm generates a scale-free network. The only requirement of the *Barabasi-Albert* algorithm is an initialization of a fully connected 4 nodes network. Now that the graph is initialized, we can start to add nodes one by one until the number N of nodes desired to our network. The number of nodes N is passed in parameter. At every add, the node x has to be linked to **strictly 4** random nodes (that explains why an initialization of a 4-nodes graph is required). Every node i of the network has a computed probability of being linked to the added node. This computation is done following the formula

$$P_i = \frac{k_i}{\sum k_j}$$

where k_i is the degree of the existing node i and $\sum k_j$ is the total degree of the network (sums over all the nodes in the network). The nodes with higher degree have then obviously more chance to be linked to the added node x. The pick phase is then not totally random. Once the 4 nodes are chosen, we simply add an edge between the 4 picked nodes and the added node x. Finally, reinitialize the probabilities and increment the total degree number by 8 for the next probabilities computation. Repeat it until N nodes are added.

1.2.2 Q4 statement

Plot the degree distribution of the generated network using a linear scale on both axes. Plot in the same figure an exponential distribution which looks similar and reports on the parameters of that distribution.

Degree distribution and exponential distribution

The degree distribution of the network generated by the Barabasi-Albert algorithm with $N = 10000$ is given by the Figure 2. Note that it exists some nodes with degree more than 100, but actually not visible in the Figure 2 because the counter for those degrees are too small. Thus, a zoom of this interval is presented in the Figure 3. Moreover, the exponential distribution which looks similar is plotted in the figures in red. Concerning the parameters of that distribution, an exponential function is then used to fit the degree distribution. This exponential function follows the probability density function formula of this kind of distribution :

$$\lambda e^{\lambda x}$$

that gives :

$$a * exp(-b * x) + c$$

where x is the degree distribution and a, b, c arbitrary values.

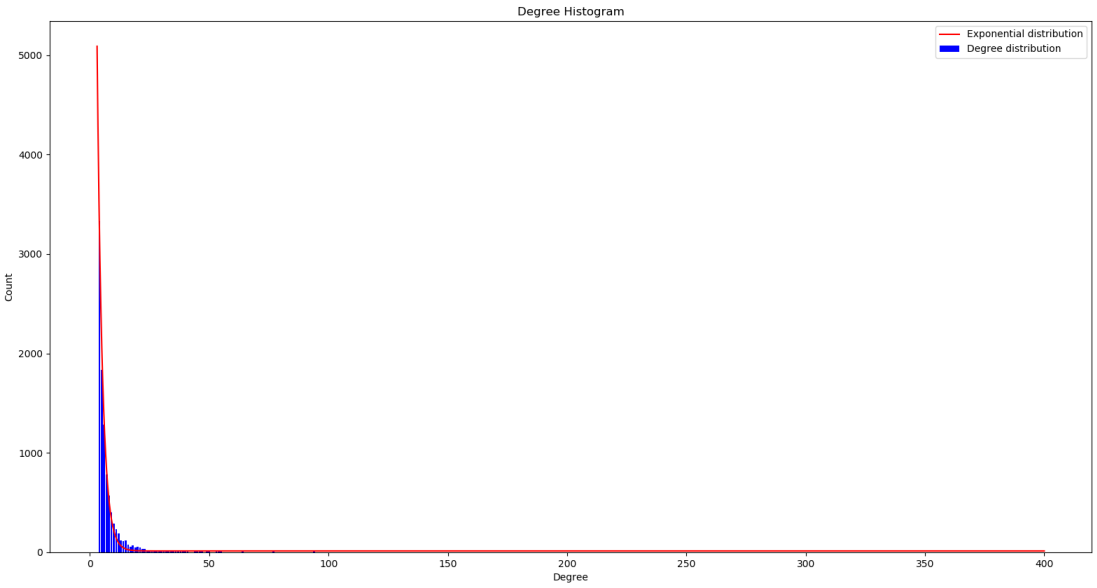

Figure 2 – Distribution for Barabasi-Albert generated network

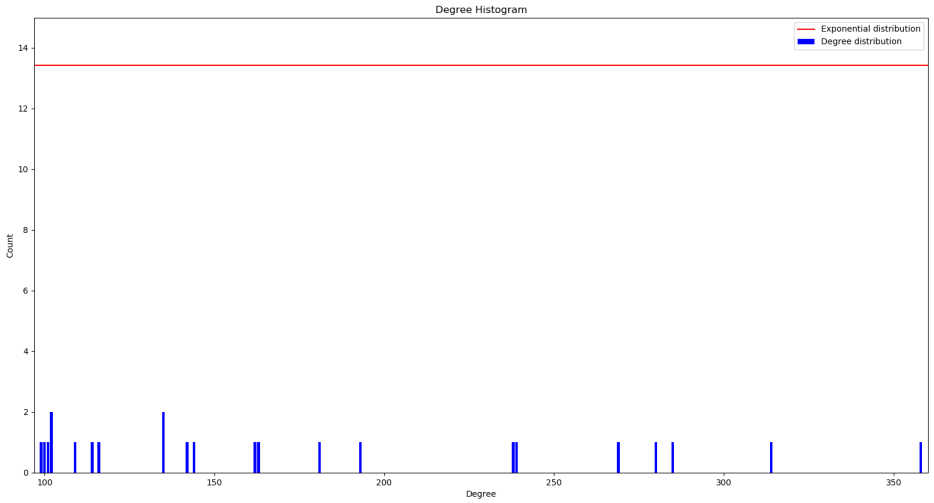

Figure 3 – Zoom of the interval 100-400 in the BA generated network distribution

1.2.3 Q5 statement

Plot the same distribution on log-log scale. Fit the distribution using Least Square fit. You can use existing functions for fitting and plot the fit next to the data. What are the parameters of the fit? How does it fit? Why? Write a paragraph about why we should not use Least Square fit to fit power laws.

Distribution on log-log scale

The same degree distribution of the network generated by the Barabasi-Albert presented in Figure 2 in log-log scale is shown in the Figure 4.

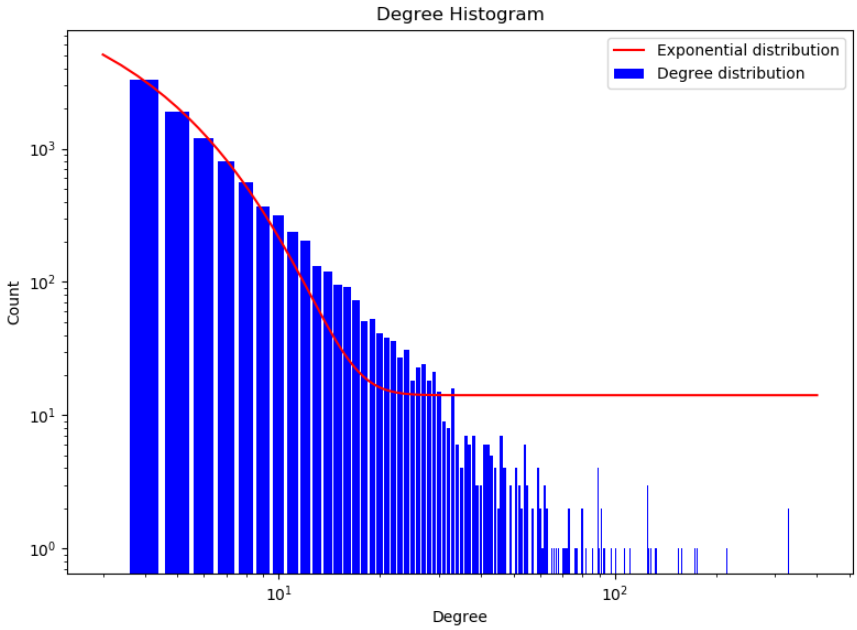

Figure 4 – Distribution for Barabasi-Albert generated network in log-log scale

Fitting using Least Square Fit

The fitting on the log-log scale distribution using Least Square Fit is presented in Figure 5. The parameters used are simply a fit function and an error function. The fit function is creating the fit line following a linear model to approximate the data set (x, y) where x is the network's degree list and y the corresponding counters of degrees x (i.e. degree x_i is counted y_i times in the network). The least square fit function is trying to minimize the sum of the squares of the residuals (difference between real data and fitted data). The fit function is then of the form :

$$y = a \cdot x^b$$

The error function is calculating the "error bar" (for this fitting, we arbitrary chose an error rate of 10%) for each point of the distribution.

The fit is not going well, most of the error bars (or even the points) of the distribution points are missing the fit line. Indeed, we should not use Least Square fit to fit power laws because of the Gaussian errors. In other words, the calculation of the noise is working only for Gaussian noise. Here, when using log-log scale, we can note that the noise isn't Gaussian anymore. Therefore, the calculation becomes false. That's why in the Figure 5, we see that where the "error bars" are missing the least square fit line, it's at the level of the noise which is not Gaussian anymore, then the calculation of the error is not valid.

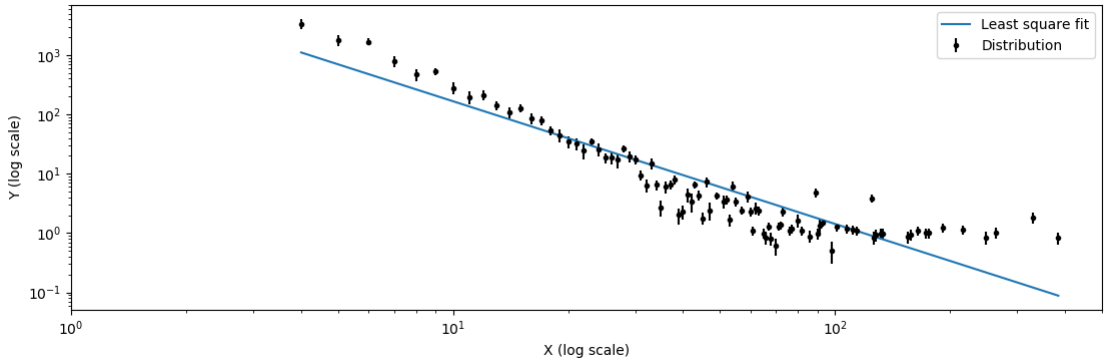

Figure 5 – Least Square Fit on log-log scale BA distribution

1.2.4 Q6 statement

Plot cumulative distribution and fit it with Least Square Fit, report the obtained parameters and plot of the fitted function.

Plotting result and parameters

The cumulative distribution of the Barabasi-Albert network is given by the Figure 6. The distribution is plotted in lin-lin scale.

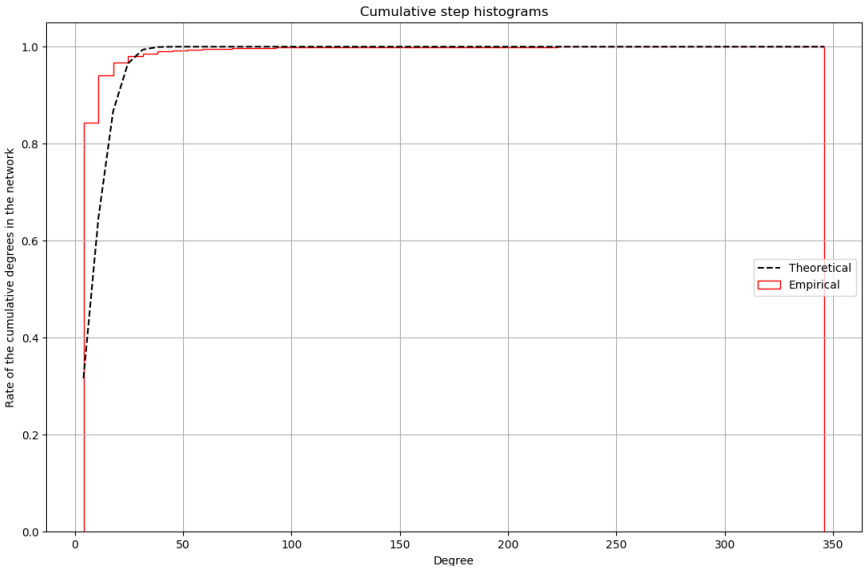

Figure 6 – Cumulative distribution of BA network

Fitting with Least square fit

After fitting the cumulative distribution with Least square fit, we obtain the fit curve for lin-lin scale represented in Figure 7. The Figure 8 represents the same fit in a log-log scale. The parameters are the same as the Least square fit on the distribution on log-log scale excepting that the y dataset (corresponding to the counters of degrees) is now the percentage rate of the degree (e.g if $y_4 = 1000$ in a network of 10000 nodes, y_4 is now equal to 0.1).

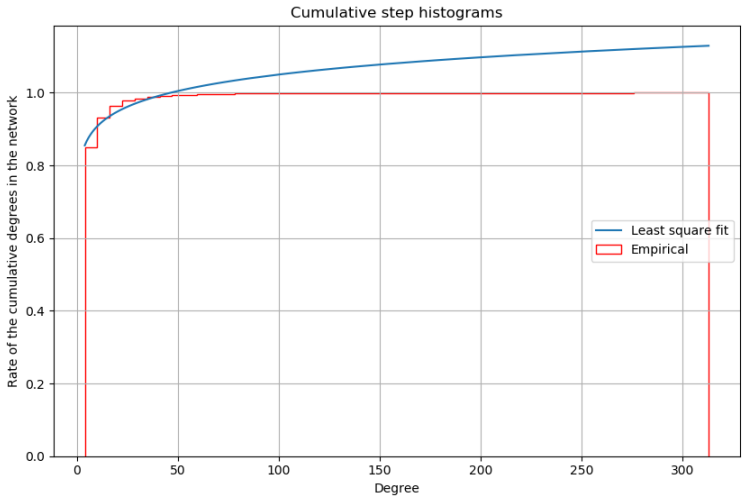

Figure 7 – Least square fit for cumulative distribution in lin-lin scale

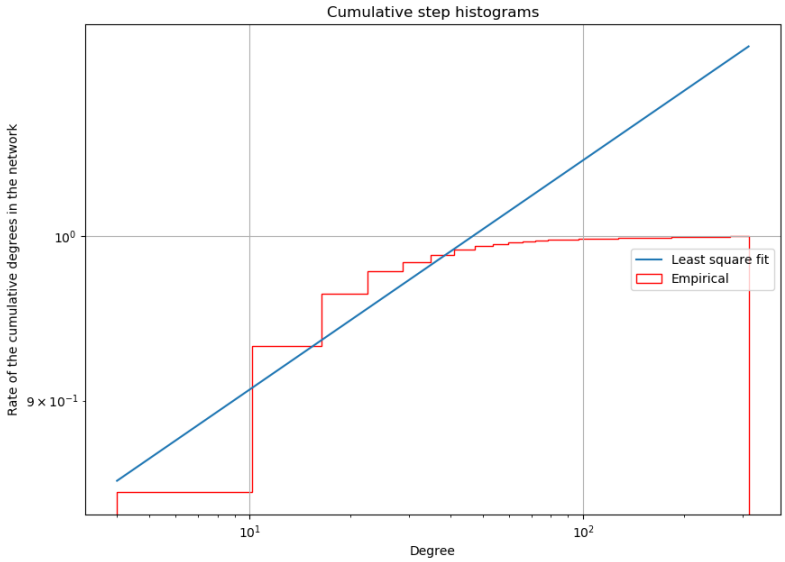

Figure 8 – Least square fit for cumulative distribution in lin-lin scale

1.2.5 Q7 statement

Now fit your distribution using maximum likelihood method. You can use any of the packages which has the method developed.

Plot result

The outcome obtained by fitting the distribution using maximum likelihood method is given in the Figure 9. The dotted line is the fitted distribution while the filled line is the empirical distribution.

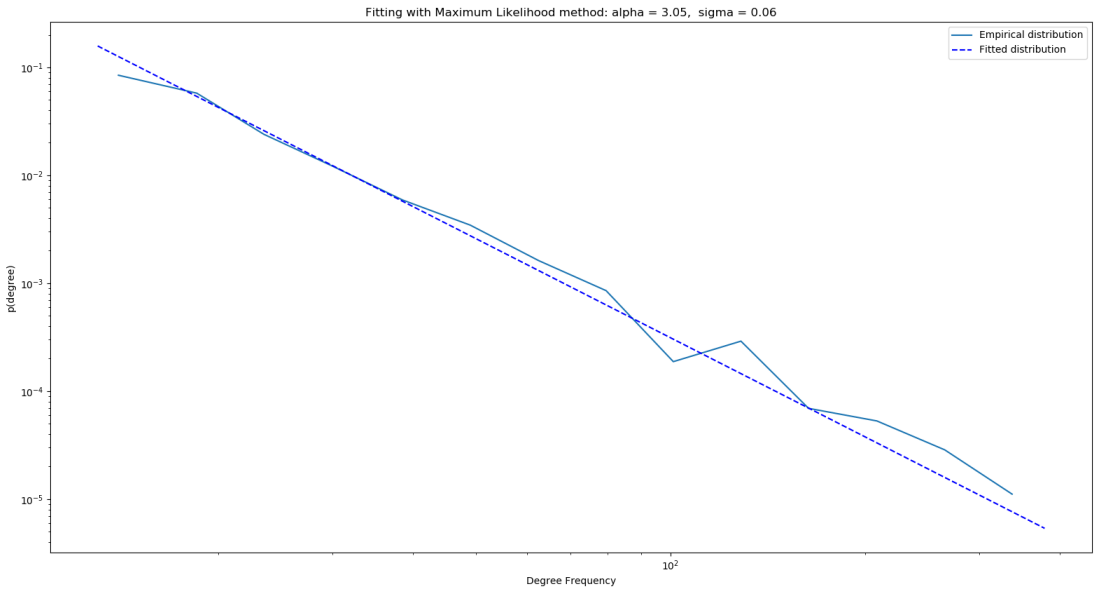

Figure 9 – Fitting the distribution with maximum likelihood method

1.2.6 Q8 statement

Report the parameters of the fit and plot them next to distribution.

Parameters report

To fit with the maximum likelihood method, we have used the `powerlaw` library which allow us to simply instantiate an object `Fit` to fit the dataset (the distribution) with the maximum likelihood method. By doing `powerlaw.Fit(distribution)`, the Fit object creates a collection of Distribution objects (corresponds to a maximum likelihood fit to a specific distribution) fitted to that dataset. The parameters of the fit and their value on the distribution of our scale-free network [1] are :

- α : the scaling parameter (the larger the scale parameter, the more spread out the distribution), $\alpha = 3.050967010354474$
- σ : its standard error , $\sigma = 0.061532086991902775$

[1] Those values are plotted in the Figure 9 in the title.

1.2.7 Q9 statement

Compare the power law fit with the exponential fit (using the same package). Report the log likelihood ratio R and the p-value. What do these numbers mean?

Comparison between the power law fit and the exponential fit

To compare the fits of two candidate distributions, here the power law fit and the exponential fit, we simply use the method

```
distribution_compare('power_law', 'exponential', normalized_ratio=True)
```

on the `Fit` object previously created by the distribution. The 2 first parameters are the two fits of candidate distributions to compare while the third one `normalized_ratio=True` option normalizes R (which will be explained in the following sentences) by its standard deviation, $\frac{R}{(\sigma\sqrt{n})}$

This method is returning two values :

- R corresponds to the loglikelihood ratio between the two candidate distributions. In other words, R will be positive if the data is more likely in the first distribution, and negative if the data is more likely in the second distribution.
- p corresponds to the $p-value$ which determines the reliability (i.e. the significance) of the value of R. It quantifies the plausibility of the hypothesis (the fitted distribution). Thus, the $p-value$ gives an idea of how is the fit going, if it approaches or not to the empirical distribution.

Now let's interpret the values that we obtain for R and p when comparing the fit with exponential and power law distribution to identify which one fits the best. We obtain :

$$R = 6.537876030701002 \text{ and } p = 6.239861089048283\ e-11$$

Concerning the value of R, we obtain a positive value which means that the distribution is quite more likely a power law distribution. However, to interpret the p value, if p is large (≥ 1), then the difference between the empirical data and the model can be attributed to statistical fluctuations alone; if it is small, the model is not a plausible fit to the data. In our case, the value of p is (really!) small, which means that the model is clearly not plausible and reliable.

In conclusion, the distribution of our scale-free network seems more like a power law distribution but the model created by fitting it is not plausible and then not significant to represent the distribution.

1.2.8 Q10 statement

What is the mathematical formula for scale free distribution you generated? Calculate the mean and the standard deviation of function? What would be the mean and standard deviation if the exponent would be 2.5?

2 Part II: Game Theory on Networks

2.1 Question 1 - Prisoner's dilemma on networks

Statement

Run a simulation of agents playing Prisoner's Dilemma on the generated networks. Explain why would we set up the probability P_{ij} like this? Why does it make sense to update your actions like that?

2.1.1 Description of the problem

In this part of the assignment, we are trying to play the prisoner's dilemma game on the previous networks generated by the Erdos-Renye and Barabasi-Albert algorithms. As a reminder, the prisoner's dilemma is a game in which two players each have two options : "Cooperate" or "Defect" whose payoffs depends on the simultaneous choice made by the other. In this section, we will simulate each node of the network as an agent playing the game with every of its neighbours. The payoff of a node is calculated by the sum of all the payoffs gained for each of its neighbours. To summarize the procedure of the game on our networks :

1. First round: each node randomly pick "Cooperate" or "Defect"
2. Rest of rounds: every node i
   (a) picks randomly one of its neighbour j
   (b) adapts the strategy chosen related to the action of j by copying the action of j with a certain probability
   $$P_{ij} = \frac{W_j - W_i}{k_{max} D_{max}}$$
   where W_i, W_j are respectively the total payoff of the current node i and of the picked neighbour j, k_{max} the maximum degree between nodes i and j, $D_{max} = max(T, R) - min(S, P)$ corresponding to the gap between the best possible payoff and the worst possible payoff.

In our case, the matrix showing the payoff for each combination of actions between node 1 and 2 is given in the Figure 10.

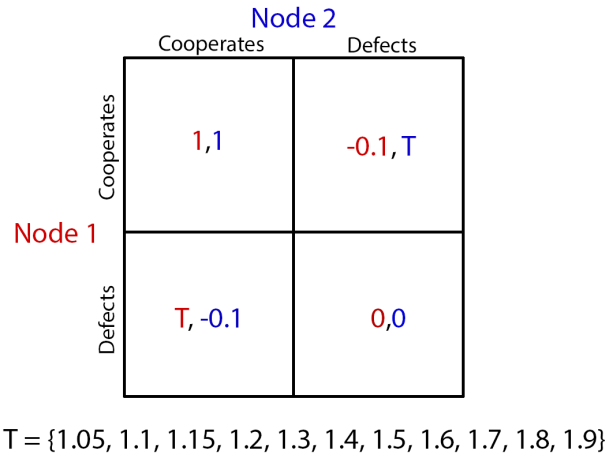

Figure 10 – Payoffs matrix for the prisoner's dilemma game

2.1.2 Adapting the strategy

The question is now why does it make sense for a node to adapt its strategy by following the probability P_{ij} ? Let's analyse the formula step by step :

- In the numerator we find $W_j - W_i$, corresponding to the difference between the payoff of the neighbour j and the payoff of the node i itself. That means that more the payoff of W_j is high, more the numerator will be high, then more the probability will have a huge value. It makes obviously a sense. More the payoff of the neighbour j is high, more the node i will try to copy the strategy of j to get a higher payoff.
- Concerning the denominator, we find the product between k_{max} and D_{max} which are then inversely proportional to the probability P_{ij}. First, k_{max} is the highest degree between i and j, it means that if j has a higher degree, he will have more chance to have a higher payoff because of summing more payoffs **even if he sums bad strategies payoffs**. Thereby, the fact that a payoff is higher is not necessarily due to a good strategy choice but can be due by summing a high number of payoff. In this case, more huge is the degree of j, less adapting the strategy of i is a good choice. Dividing the numerator by k_{max} is then necessary to decreases the chance to copy a bad strategy and then creating a good balance with the numerator. Moreover, concerning D_{max}, more the gap between the best the best strategy payoff and the worst one is big, more the payoff of the bad strategy is big. Thus, it is more critical to adapt the worst strategy. This D_{max} is then useful for a question of safety which means that the node i prefers to not gain more instead of losing a lot. Indeed, more D_{max} is high, more the bad strategy has a huge impact, less the node i will risk to change to maybe adopts a worse strategy.

2.2 Question 2 - Stationary state of cooperation level

Statement

Plot the cooperation level over time for all the values of T and for both networks. Establish after how many rounds the system reaches stationary state (when the level of cooperation does not change too much).

2.2.1 Plot result

The following plotting outcomes are got by simulating the prisoner's dilemma game on 1000 rounds. We arbitrary chose to pick a huge number of rounds to have more accuracy in the result. The Figure 11 presents the plot of the simulation on the network generated by Erdos-Renye algorithm while the Figure 12 presents the simulation on the network generated by Barabasi-Albert algorithm.

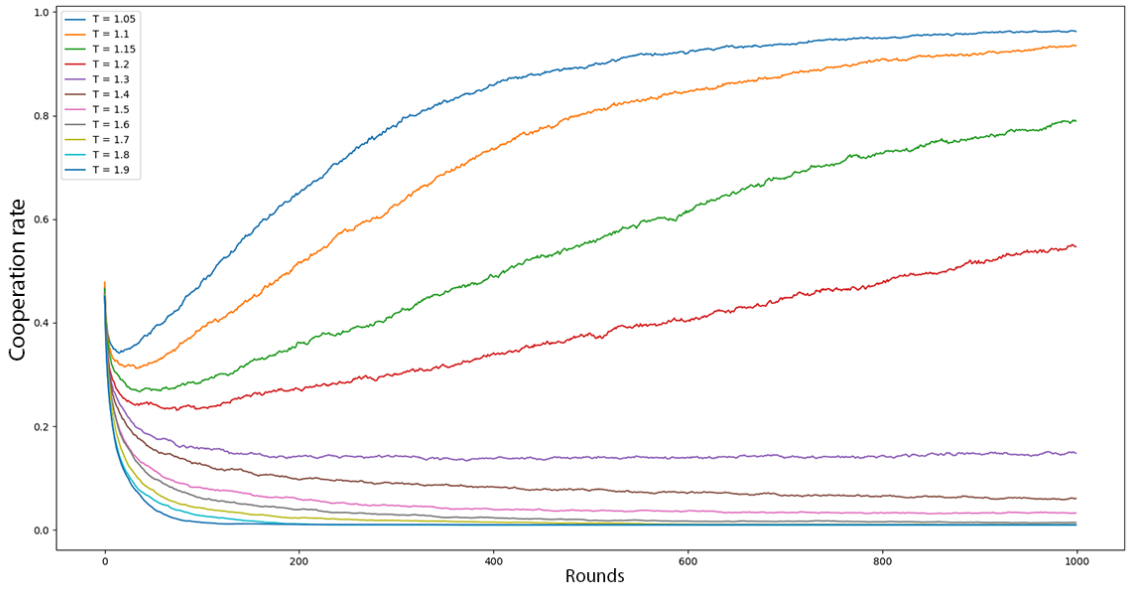

Figure 11 – Cooperation rate for prisoner's dilemma game on **Erdos-Renye** network for 1000 rounds

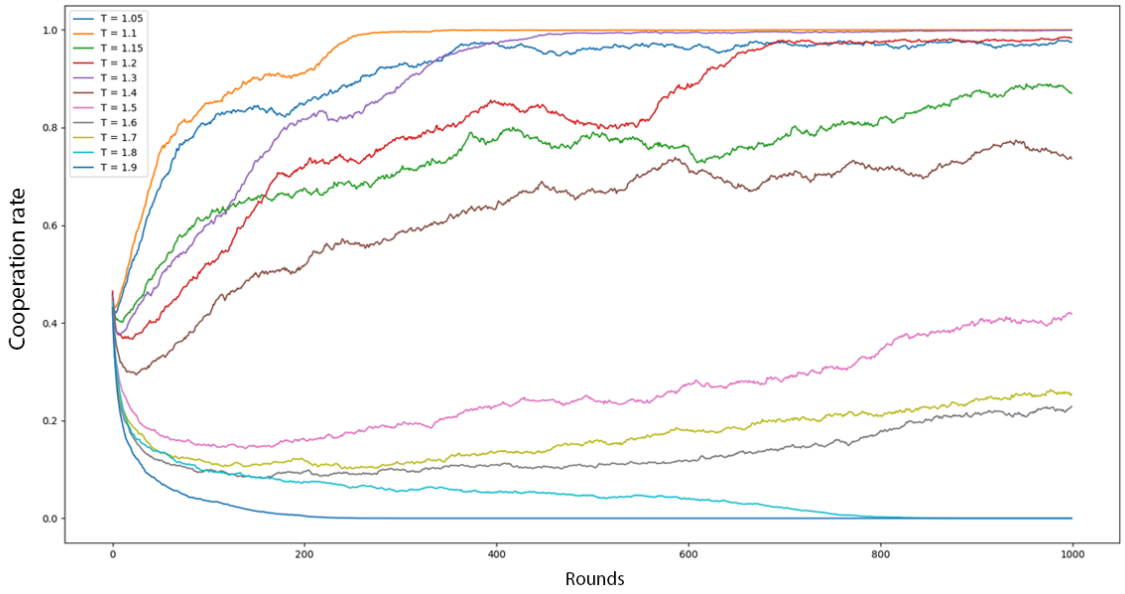

Figure 12 – Cooperation rate for prisoner's dilemma game on **Barabasi-Albert** network for 1000 rounds

2.2.2 Stationary state

For Erdos-Renye : Generally, in the case of a graph generated randomly, we observe that more the value of T is small, more quickly the cooperation rate will converge to 1. We can distinct 2 "sides" of T value. For all the $T \leq 1.2$ (i.e. the top side), the cooperation rate will increase and converges to 1 while from a value of $T \geq 1.3$ (i.e. the bottom side), the cooperation rate will converge or stay steady in a value near to 0. For this generated graph, the stationary state depends of the value of T. For the bottom side (i.e. $T \geq 1.3$), the stationary state seems to be near 100-150 rounds. On the other side, for $T \leq 1.2$, the stationary state approaches more near 450-500 rounds.

For Barabasi-Albert : Concerning the scale-free network, the cooperation rate is not really steady, we observe a lot of variations so that we can note the cooperation rate decreasing just after being increased and vice versa. The difference that could make this variation is the fact that in a scale-free network, it can exists a node with a high degree (i.e. more than 200-300 for a 10000 nodes network). As a node adapts his strategy according to his neighbour, in such a case, all the nodes connected to the 'high degree' node are potentially changing then creating a variation in the curve. Nevertheless, the general behaviour of the cooperation level over time is quite the same than for random networks. For a value of $T \leq 1.4$, the curves seems to converge to one while for a $T \geq 1.5$, the curves are more converging to 0. Thus, as the random graph, the stationary state depends of the value of T. For the bottom side (i.e. $T \geq 1.5$), the stationary state seems to be near 200-300 rounds. On the other side, for $T \leq 1.4$, the stationary state appears more near 800 rounds.

2.3 Question 3 - Average of the stationary cooperation

Statement

Average the stationary cooperation level over 100 simulations (for each T and both networks). Plot the dependence of the stationary cooperation level on the value of T for both networks.

2.3.1 Plot result

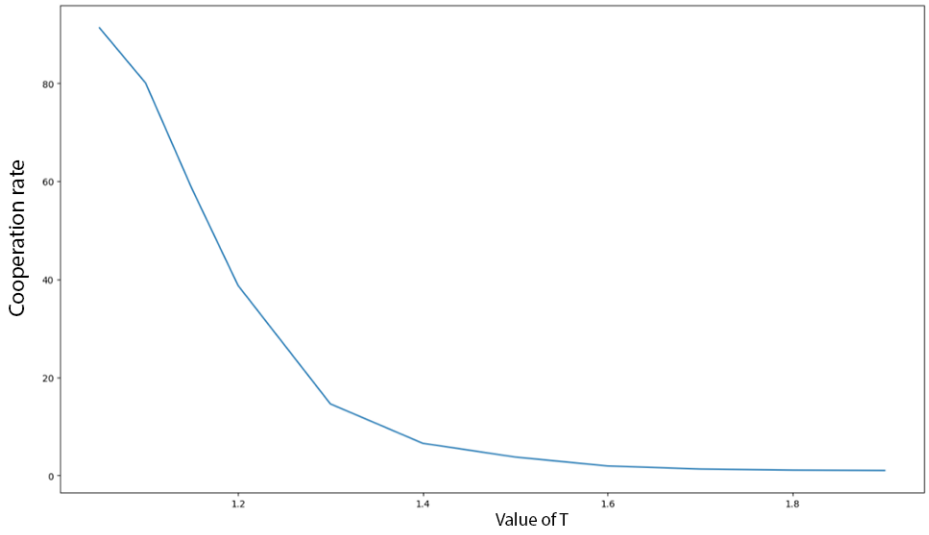

Figure 13 – Average of stationary cooperation level on 100 simulations for **Erdos-Renyi** network

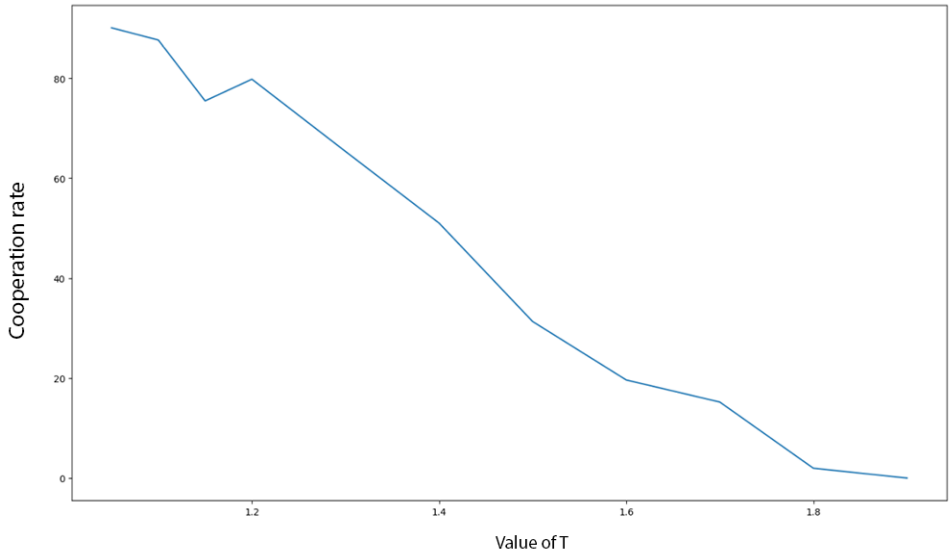

Figure 14 – Average of stationary cooperation level on 100 simulations for **Barabasi-Albert** network

2.3.2 Description

We remark that the general look of the obtained graph is a decreasing function proportionally to the value of T but seems to be more steady in the case of a random network than on a scale-free network. An hypothesis of explanation of this difference is given in the previous section (2.2.2).

2.4 Question 4 - Difference between the final cooperation levels

Statement

Comment the differences of the final cooperation levels.

2.4.1 Comments

What we can conclude about the difference of the final cooperation levels considering different values of T is that the general behaviour of the curve is decreasing more the value of T is low as just said before. This feature is valid whether for the random generated graph (Erdos-Renye) or for the scale-free network (Barabasi-Albert). Thus, the final[2] cooperation level of a large value of T ($\geq$ 1.4-1.5) will be a value near to 1 while the final cooperation level of a small value of T ($\leq$ 1.4-1.3) will be a value near to zero. However, we can observe that a scale-free network promotes the cooperation. Indeed, the final value of almost all value of T is bigger for the scale-free network than the random network for the same values of T.

[2]Here we call "final", the value obtained at the end of 1000 rounds, a large value arbitrary chosen previously.