# Network Science Anlytics

# Assignment 1

AURIAU Vincent

If need the whole code can be found here:

https://github.com/VincentAuriau/NGSA/blob/master/Assignment%201/Assignment_1.ipynb

## I. Graph Theory and Graph Properties

***Question 1***

Let A be the adjacency matrix of an undirected graph (unweighted, with no self-loops) and 1 be the column vector whose elements are all 1.

(a)

Let k be the vector whose elements are the degrees $k_i$ of the nodes.

$$\boxed{k = A * 1}$$

(b)

Let m be the number of edges in the graph.

$$\boxed{m = \frac{1}{2} * \sum_i k_i = \frac{1}{2} * k^T * 1}$$

(c)

Let N be the matrix whose element $N_{i,j}$ is equal to the number of common neighbours of nodes I and j.

$$\boxed{N = A^t * A = A^2}$$

***Question 2***

Let h be the number of edges in the graph. All of them connect a node from the group (1) to a node from the group (2).

In the group (1), the mean degree is $c_1 = \frac{h}{n_1}$

In the group (2), the mean degree is $c_2 = \frac{h}{n_2}$

Therefore, we say that $n_1 * c_1 = n_2 * c_2$ and

$$\boxed{c_2 = \frac{n_1}{n_2} * c_1}$$

***Question 3***

(a)

$A^3$ counts the number of paths of length 3 between two nodes. We want to count the number $\Delta\mathrm{G}$ of triangles in the graph. Each element of the diagonal of $A^3$ counts twice the number of triangles the node is involved in. Indeed it counts the paths of length 3 that start and finish with this node.

If we add the diagonal elements of the matrix we also count each triangle three times with each of the nodes of this triangle.

Thus, $\boxed{\Delta\mathrm{G} = \frac{Trace(A^3)}{6}}$

(b)

A is a symmetric matrix and can be diagonalised as a symmetric matrix. We can write:

$$A = U * \Sigma * U^{-1}$$

With $\Sigma$ a diagonal matrix whose elements are $\lambda_i$, the eigenvalues of the matrix A.

The trace is a linear form and $A^3 = U * \Sigma^3 * U^{-1}$

Thus: $\mathrm{trace(A^3)} = \mathrm{trace(U} * \Sigma^3 * \mathrm{U^{-1})} = \sum_{i=0}^{n} \lambda_i^3$

And $\boxed{\Delta\mathrm{G} = \frac{1}{6} * \sum \lambda_i^3}$

(c)

As we have seen it, $\Delta_i = \frac{1}{2} * a_{i,i}^3$ with $a_{i,i}^3$ the ith element of the diagonal of the $A^3$matrix. Indeed it represents twice (because there are two directions) the number of length 3 paths that begin and end on the ith node.

Using $A^3 = U * \Sigma^3 * U^T$, we can say that:

$\boxed{\Delta_i = \frac{1}{2} * \sum_{j=0}^{n} \lambda_j^3 * u_{i,j}^2}$

with $u_{i,j}$the elements of the matrix U.

II. Graph Models

***Question 4***

We consider the random graph $G_{n,p}$ with average degree *c.*

(a)

We place ourselves in the limit of a large n.

Let's call $p = \frac{c}{n}$ the probability that two nodes are connected to each other.

Let's choose a node n1. This node has a probability p to be connected to the node n2 different than n1. The node n2 has also a probability p to be connected to the node n3 different than n1 and n2. Finally n3 has a probability p to be connected to n1. It means that there is a probability $p^3$ that n1, n2 and n3 create a triangle. We are still placed at the node n1. We have (n-1) possibilities to choose n2 and once it's chosen, we have (n-2) possibilities to choose n3.

Thus, n1 will create $\frac{1}{2} * \ (n-1) * (n-2) * p^3$ triangles ($\frac{1}{2}$ is there because we this method we count twice each triangle: n1-n2-n3 and n1-n3-n2). We have n nodes, and each triangle is counted for its three nodes.
Finally the number of triangles is $\frac{n*(n-1)*(n-2)*p^3}{2*3}$.

When n is big, we have $n(n-1)(n-2)p^3 = \frac{n(n-1)(n-2)c^3}{n^3} \ \approx c^3$

$$\boxed{\Delta G = \frac{c^3}{6}}$$

(b)

Using the same method, each node has a probability $(n-1)(n-2)p^2$to be at the beginning (or at the end) of a triplet. We have n nodes but we count each triplet twice (at the first and last node). The same way, when n is big, $\frac{n-1}{n} \approx 1$.

And we find the number of triplets: $\frac{n(n-1)(n-2)p^2}{2}$. Finally:

$$\boxed{\Delta N = \frac{nc^2}{2}}$$

(c)

We now define the clustering coefficient

## III. Centrality Criteria

***Question 5***

(a)

$$x_i = \sum_{j=0}^{n} \alpha^j * |\{k : d(i,k) = j\}|$$

Where $|.|$ corresponds to the cardinal (number of elements) of the ensemble.

We can also write it:

$$\boxed{x_i = 1 + \sum_{j \in V, j \neq i} \alpha^{d(i,j)}}$$

(b)

Idea for the algorithm:

-Initalisation of the central criterion to 0

-We iterate over the variable n:

We compute the $A^n$matrix. For all $1 \leq i \leq n \; and \; 1 \leq j \leq n$, we check:

If on the ith line a one appears on the jth columns and the element on the ith line, jth column was a zero for the $A^{n-1}$ matrix, it means that the shortest path from the node i to the node j has a length of n. We add $\alpha^n$ to the central criterion value.

-We stop when we have found all the path length.

Thus we can compute the centrality criterion.

If p is the number or nodes of the graph, calculating the cost of $A^n$ has a cost of $p^2$operations using $A^{n-1}$. In the worst case, we have to calculate through $A^{p-1}$ (if the graph is just a line of nodes and we compute the geodesic distance for one of the two nodes at the extremities.) If we want to compute the geodesic distance for the p nodes, we will have to iterate this p times.

Thus our algorithm will have a complexity of $o(p^4)$.

***Question 6***

Let’s place ourselves at the point A. the distance to any point in the graph is the distance from this point to B plus 1 if this point belongs to the same subgraph as B and minus one if it belongs to the same subgraph as A.

For the nodes in the same subgraph as A, we have:

$$\sum_j d_{A,j} = \sum_j (d_{B,j} - 1) = \sum_j d_{B,j} - n_A$$

We have a similar relation for the nodes of B:

$$\sum_j d_{A,j} = \sum_j (d_{B,j} + 1) = \sum_j d_{B,j} + n_B$$

Obviously, all the nodes are either in the same subgraph as A or in the same subgraph as B, thus we can write for all the nodes:

$$\sum_j d_{A,j} = \sum_j d_{B,j} + n_B - \ n_A$$

$$\frac{1}{n} * (\sum_j d_{A,j} + n_A) = \frac{1}{n} * (\sum_j d_{B,j} + n_B)$$

$$\boxed{\frac{1}{C_A} + \frac{n_A}{n} = \frac{1}{C_B} + \frac{n_B}{n}}$$

Analysing a Real Network

For the next questions, we will use Python and the NetwokX library.

***Question 7***

(a)

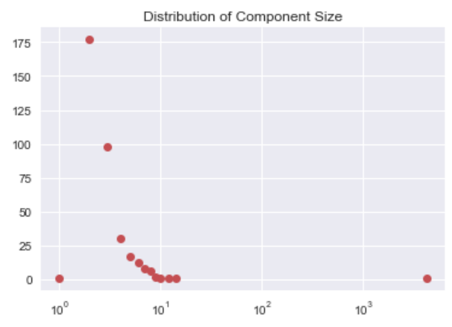
(1) After creating the graph using the text file given, we compute the number of nodes and edges:

```
print('Number of Nodes in the Graph:', len(dG))
print('Number of Edges in the Graph:', dG.number_of_edges())
```

```
Number of Nodes in the Graph: 5242
Number of Edges in the Graph: 28980
```

(2) We also compute the number of connected components and the distribution of their sizes. We display this distribution using a logarithm scale on the x axis so that the size of the GCC can appear on the graph.

```
Number of Connected Components: 355
Distribution of components size: {1: 1, 2: 177, 3: 98, 4: 30, 5: 17, 6: 12, 7: 8, 8: 6, 9: 2, 10: 1, 12: 1, 14: 1, 4158: 1}
```



(3) We extract the GCC and compute a few of its features:

```
print('Number of Nodes in the GCC:', len(GCC))
print('Number of Edges in the GCC:', GCC.number_of_edges())

print('Fraction of Nodes in the GCC:', len(GCC)/len(G))
print('Fraction of Edges in the GCC:', GCC.number_of_edges()/G.number_of_edges())
```

```
Number of Nodes in the GCC: 4158
Number of Edges in the GCC: 13428
Fraction of Nodes in the GCC: 0.7932086989698588
Fraction of Edges in the GCC: 0.9263245033112583
```

The GCC represents 80% of the nodes and 92% of the edges of the graph. Most of the graph is contained in this GCC. It means that most of the authors are linked to each other.

(b)

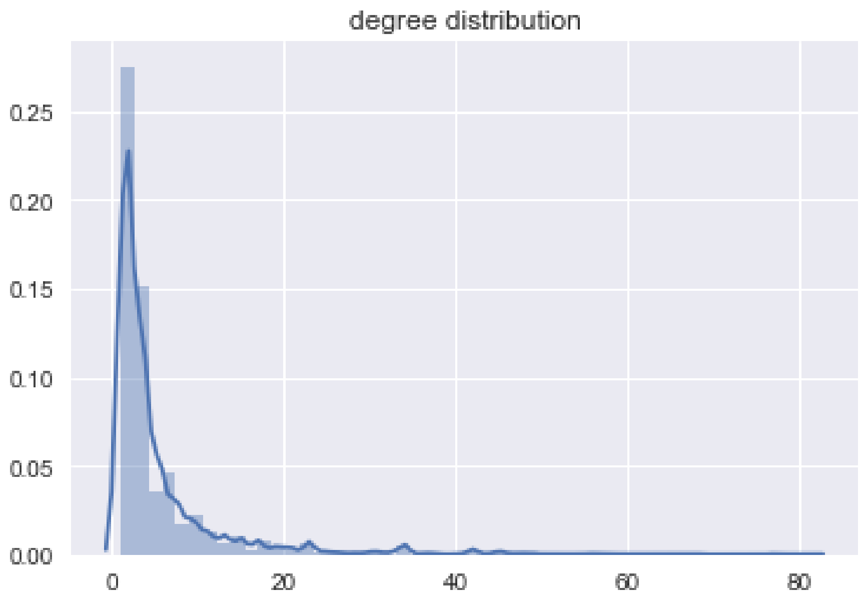
We compute a few features about the degree of the nodes in the graph :

```
print('Maximum Degree', degree_max)
print('Minimum Degree', degree_min)
print('Average Degree', degree_avg)
print('Median Degree', degree_median)
```

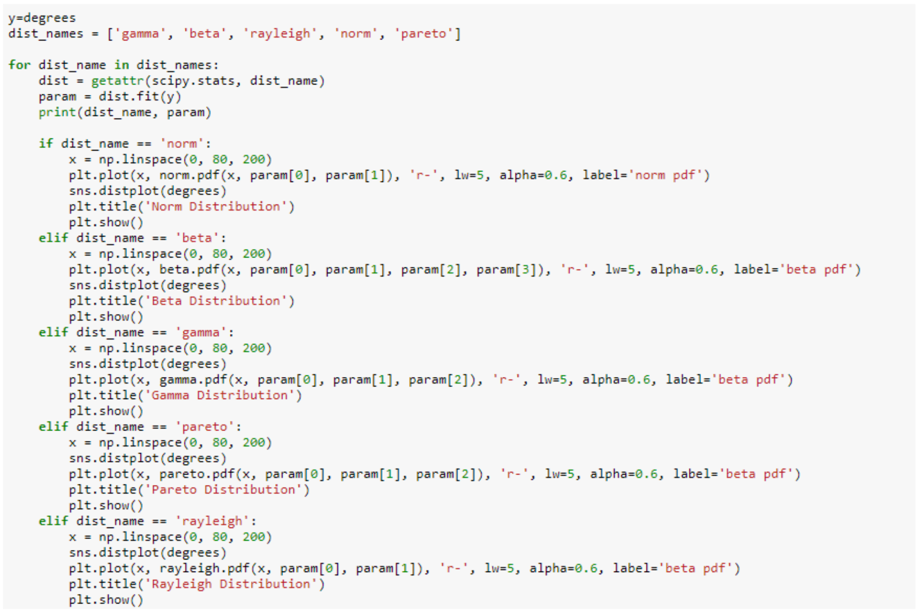
```
Maximum Degree 81
Minimum Degree 1
Average Degree 5.530713468141931
Median Degree 3.0
```

We observe that most have a small degree (50% of them equal or below 3) but some nodes have a really high degree (40 times the average degree).

We display the distribution of the degrees of the nodes :



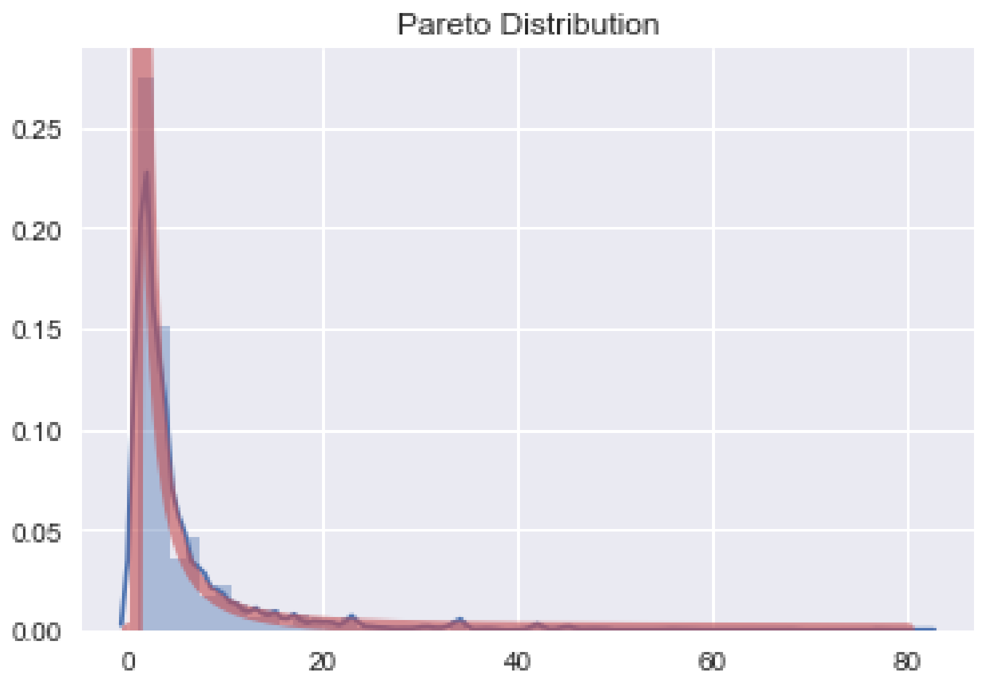
We try to fit several models to see which one it corresponds to. We tried to fit a Beta model, a Gamma model, a Normal model, a Rayleigh model, but the best one is the Pareto model.



```
y=degrees
dist_names = ['gamma', 'beta', 'rayleigh', 'norm', 'pareto']

for dist_name in dist_names:
    dist = getattr(scipy.stats, dist_name)
    param = dist.fit(y)
    print(dist_name, param)

    if dist_name == 'norm':
        x = np.linspace(0, 80, 200)
        plt.plot(x, norm.pdf(x, param[0], param[1]), 'r-', lw=5, alpha=0.6, label='norm pdf')
        sns.distplot(degrees)
        plt.title('Norm Distribution')
        plt.show()
    elif dist_name == 'beta':
        x = np.linspace(0, 80, 200)
        plt.plot(x, beta.pdf(x, param[0], param[1], param[2], param[3]), 'r-', lw=5, alpha=0.6, label='beta pdf')
        sns.distplot(degrees)
        plt.title('Beta Distribution')
        plt.show()
    elif dist_name == 'gamma':
        x = np.linspace(0, 80, 200)
        sns.distplot(degrees)
        plt.plot(x, gamma.pdf(x, param[0], param[1], param[2]), 'r-', lw=5, alpha=0.6, label='beta pdf')
        plt.title('Gamma Distribution')
        plt.show()
    elif dist_name == 'pareto':
        x = np.linspace(0, 80, 200)
        sns.distplot(degrees)
        plt.plot(x, pareto.pdf(x, param[0], param[1], param[2]), 'r-', lw=5, alpha=0.6, label='beta pdf')
        plt.title('Pareto Distribution')
        plt.show()
    elif dist_name == 'rayleigh':
        x = np.linspace(0, 80, 200)
        sns.distplot(degrees)
        plt.plot(x, rayleigh.pdf(x, param[0], param[1]), 'r-', lw=5, alpha=0.6, label='beta pdf')
        plt.title('Rayleigh Distribution')
        plt.show()
```

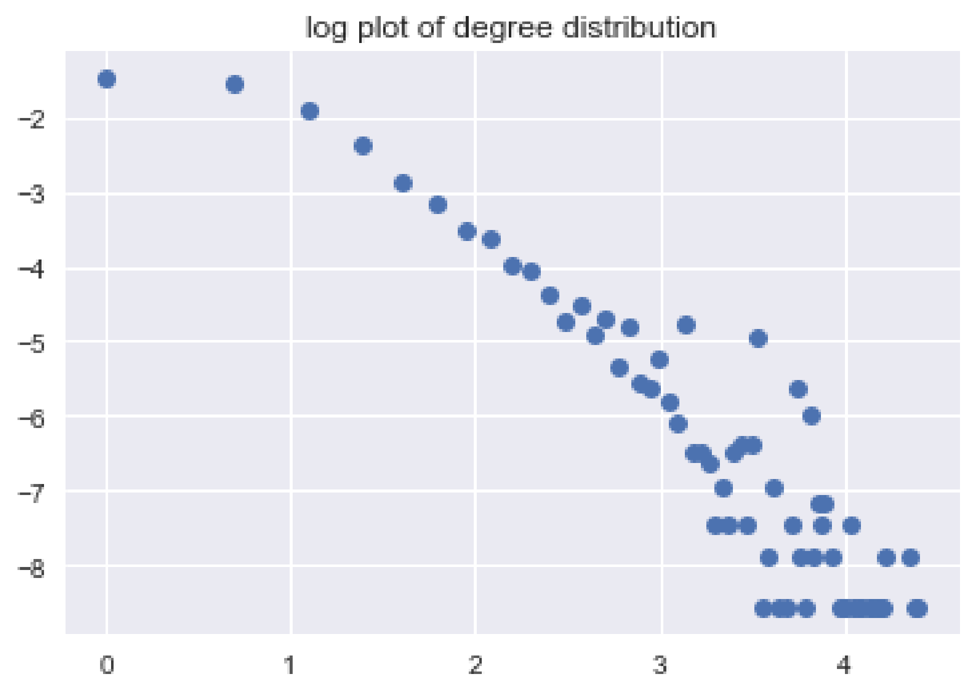


The Pareto model with the parameters:

(Shape=0.9981359104274129, location=-0.4078395532384388, scale=1.407839552290595) is the one that fits the best the nodes degree distribution.

We can also, as in the class try to fit a power-law: $p(k) = c * k^{-\alpha}$.

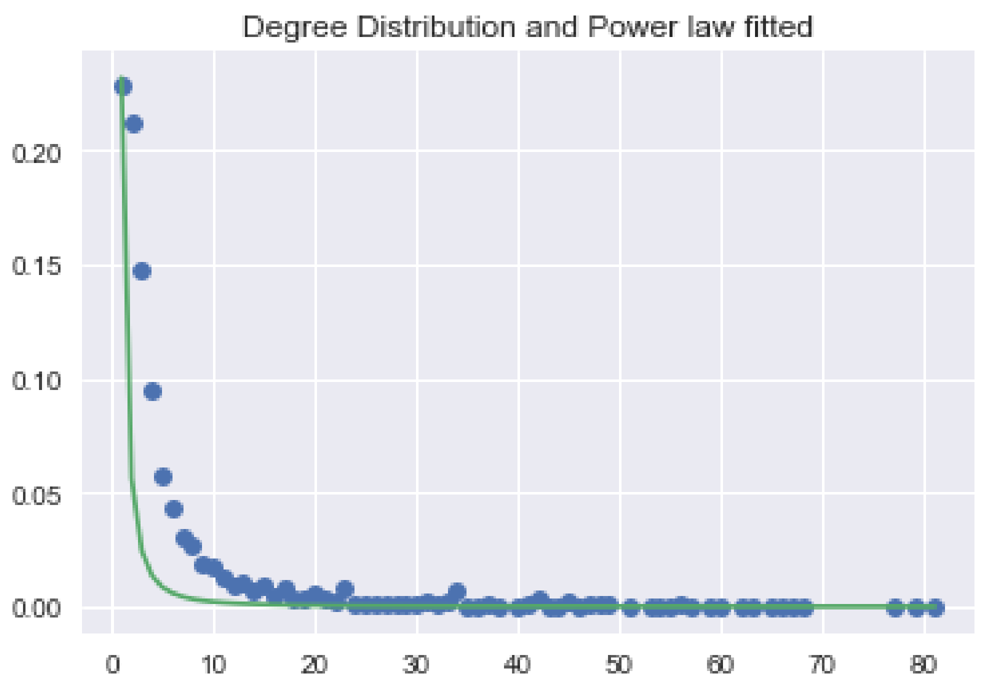
First if we plot the log distribution of the degree distribution:



We can fit a line and this will give us the parameters of the power law. Fitting a degree one polynomial function gives us :

$$c = 0.23 \; and \; \alpha = -2.0$$

We can now try to compare the power law with the degree distribution:



It’s pretty good, but the pareto la looked better. Looks logic since it is almost the same law with one more parameter.

```
# Let's work with a distibution p(k) ~ ck^-a with p(k) the fraction of nodes with degree k

degrees_dict = {}
for i in range(len(degrees)):
    try:
        degrees_dict[degrees[i]] += 1
    except:
        degrees_dict[degrees[i]] = 1

for key in degrees_dict.keys():
    degrees_dict[key] = degrees_dict[key] / len(degrees)

def log_pk(x):
    global degrees_dict
    return math.log(degrees_dict[x])

x_axis = sorted(list(degrees_dict.keys()))
x_axis_log = []
y_axis_log = []
y_axis_normal = []
for element in x_axis:
    x_axis_log += [math.log(element)]
    y_axis_log += [math.log(degrees_dict[element])]
    y_axis_normal += [degrees_dict[element]]

plt.plot(x_axis_log, y_axis_log, 'o')
plt.title('log plot of degree distribution')
plt.show()
p = np.polyfit(x_axis_log, y_axis_log, deg=1)
print(p)

def pk(x):
    return p[1] * (x**(p[0]))

y_axis = []
for element in x_axis:
    y_axis += [pk(element)]

plt.plot(x_axis, y_axis_normal, 'o')
plt.plot(x_axis, y_axis)
plt.title('Degree Distribution and Power law fitted')
plt.show()
```

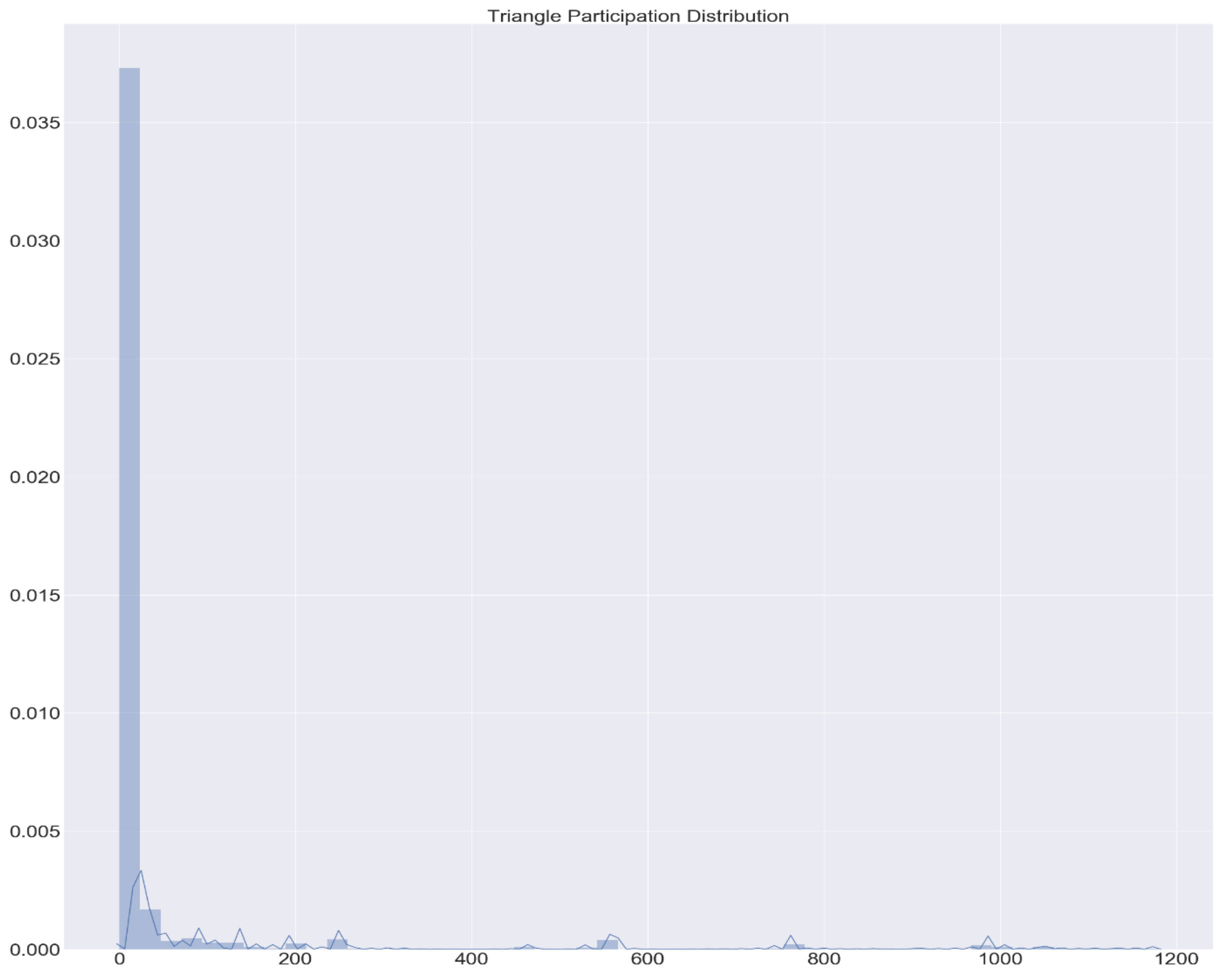
(c)

For the following questions, we will focus on the GCC. We compute the number of triangles:

```
print('Number of triangles in the GCC:', number_triangles)
```

```
Number of triangles in the GCC: 47779.0
```

Let’s visualize the triangle participation distribution.



Most nodes participate only in a few triangles. It means that there are a lot of small clusters of a few persons. Most authors collaborates mostly with the few persons in their group.

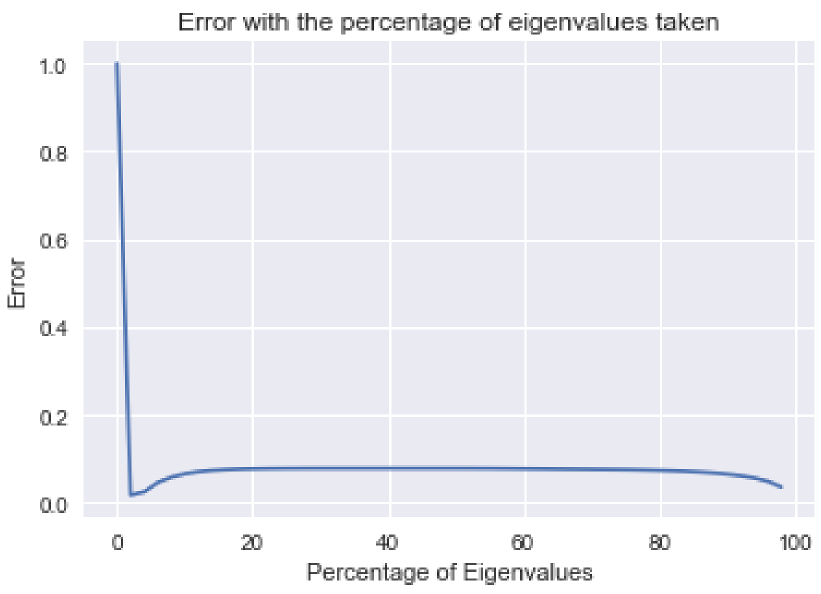
(d)

(1) Calculation the spectrum of the adjancy matrix can be a computational bottleneck. Indeed the best performing algorithm have a computation cost of $O(n^3)$ with n the number of nodes (the adjency matrix will be a $n\ x\ n$ sized matrix. If we have too many nodes it will quickly take a lot of time.

According to the Triangle Particiaption Distribution, some nodes do not participate in any triangle. Thus, the eigenvalues representethe number of triangles its nodes participates in. If we do not take the smallest eigenvalues, we will not take small values (0 , 1, 2, …).

Taking the highest eigen values will give a really good approximation. In my case I did not have any issue computing those values and thus made the calculus with all of them.

(2) We compute and visualize the error with the next graph.



We can see that with 20% of the values we have a really good estimation of the number of triangles. Thus we can take the top 1000 eigenvalues.

## *Question 8*

We generate an Erdös-Rényi graph, with n=1000 nodes and p=0.009.

(a)

The mean degree of the graph will be $n * p = 9$.

Indeed, each of the n nodes has a probability p to be linked to the n-1 other nodes. The average number of links will be $n * p$.

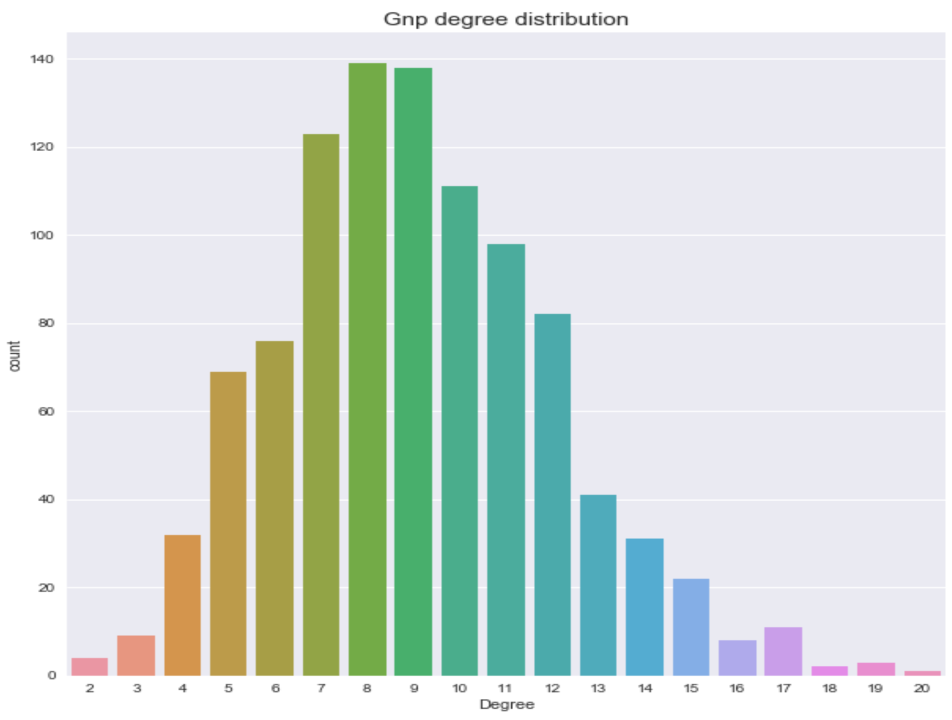
(b)

The graph is connected. Indeed $p > \frac{(1+\epsilon)*\ln(n)}{n}$ which makes the graph almost surely connected. Basically the probability that a node is not connected is $(1-p)^n$ which is very low.

(c)

We compute the mean degree of the graph. We find 8.93. It's close to what we predicted. Because it's a matter of probability, we are not exactly at 9 which is logical.

We display the degree distribution of $G_{n,p}$.



We compute the degree distribution. We can see that most nodes have a degree between 7 and 12. The minimal degree is 2 (not even one, indicating that the probability to have 0 is very low), and the maximum 20.

After several tries, I observed that sometimes we have one node with a degree of 1 and that the maximum degree is most of the time between 20 and 25 but for only one or two nodes.

***Question 9***

(a)

The graph will be connected. Indeed in the initial graph, we have 0.99 has an edge probability. This quiet big. During the propagation this probability will create a lot of edges. The probability of a node not to be linked to any of the other nodes whereas there will be such high probabilities will be really low.
We can also think that all these nodes connected through this high probability will constitutes the GCC.

(i) The Kronecker graph will be connected.
(ii) The graph will also have a giant connected component.

This is confirmed computing the properties of the grap.

(b)

We will use three properties to describe both graph:

-The degree distribution

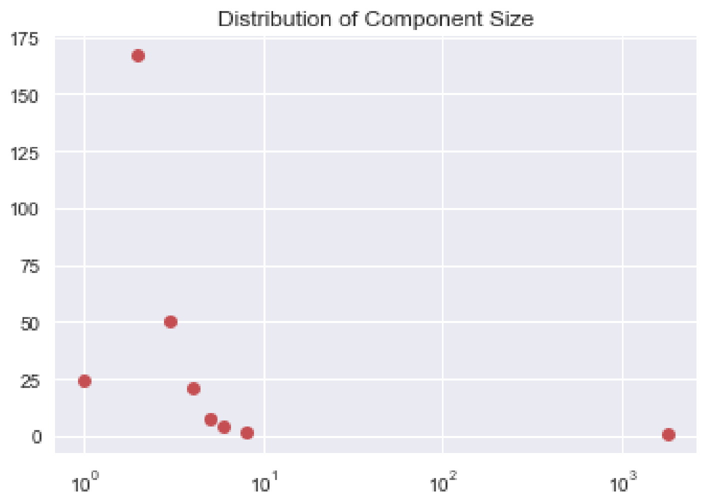
-The triangle distribution

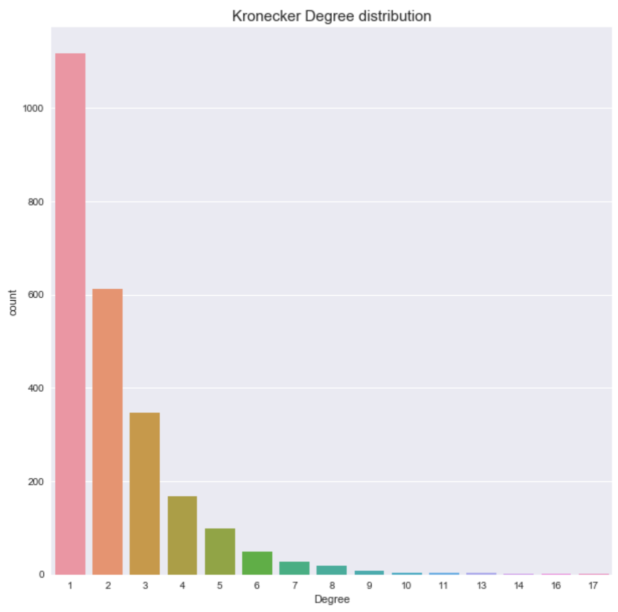
-The size of the Giant Connected Component

We already have these information for the CA-GrQC, we can compute them for the Kronecker graph.
For the Kronecker Graph, we find a GCC that represents 34% of the nodes and 15% of the edges, we found really different numbers for the CA-GrQC Graph.

However the degree distribution is different: the nodes have smaller degrees. The median is only 2 and the average is 2.2 (compared to 3 and 5.5).

We find a Component Size Distribution that looks like the one for the CA-GrQC Graph even though thevalues are different.





We can suppose the the CA-GrQC graph had too many edges to be well represented by a Kronecker Graph.

```
K = nx.Graph()

for i in range(len(An)):
    for j in range(i, len(An)):
        rn = random.uniform(0, 1)
        if rn <= An[i][j]:
            K.add_edge(i, j)

K_degrees = K.degree()
K_degrees = np.array(list(K_degrees.values()))
K_degree_avg = np.average(K_degrees)

print('Average Degree of the K network:', K_degree_avg)

sns.set(color_codes=True)

plt.figure(figsize=(10, 10))
sns.countplot(K_degrees)
plt.title('Kronecker Degree distribution', fontsize=15)
plt.xlabel('Degree')
plt.show()
```

```
Average Degree of the K network: 2.2035002035002034
```

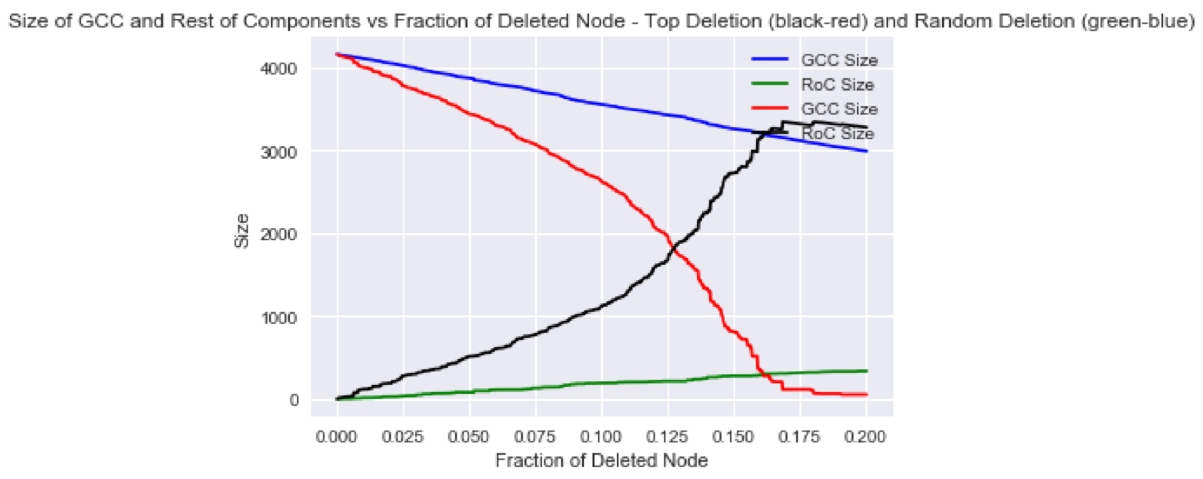
***Question 10***

We adopt two strategies:

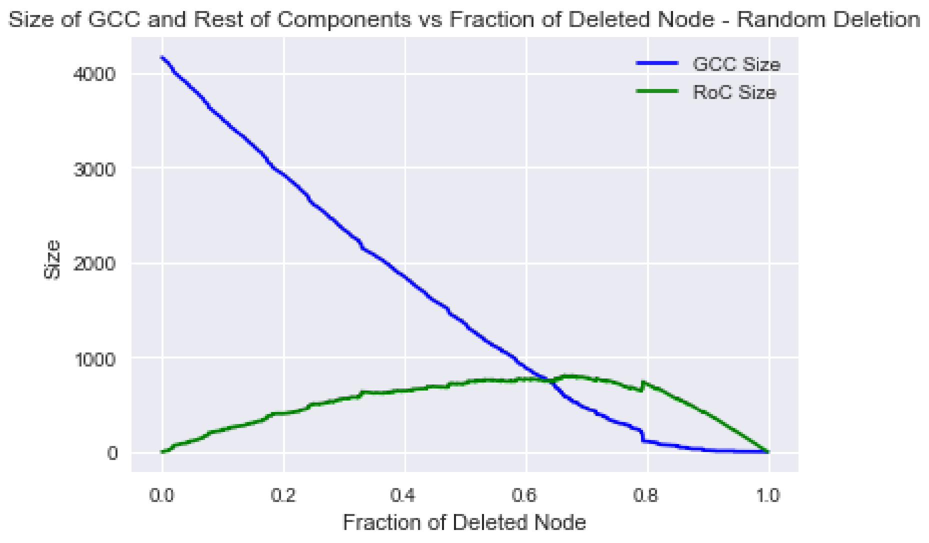
-The Random Strategy with the deletion of a random node

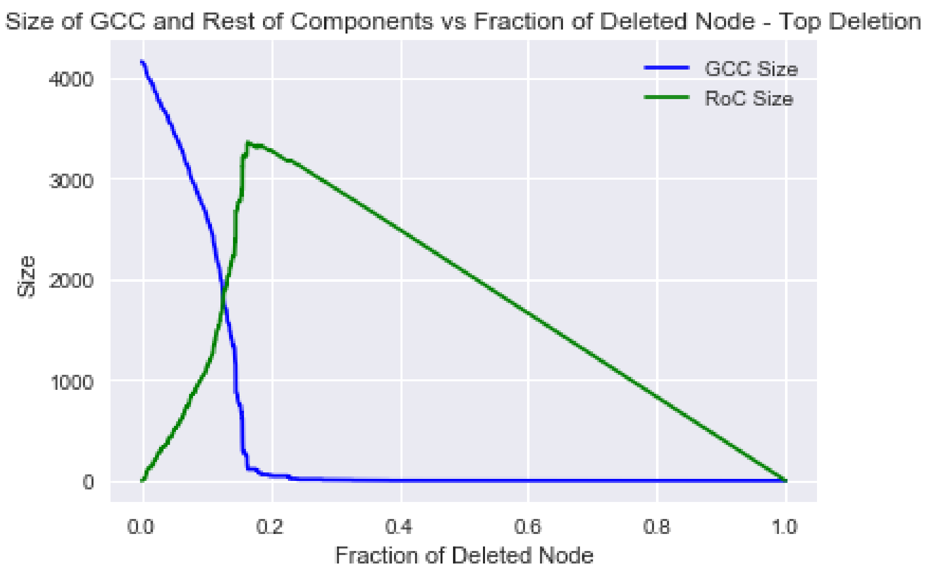
-The Top Strategy with the deletion of a random node among the top ten of the nodes with the highest degree

We plot the size of the GCC and of the rest of the components versus the fraction of deleted nodes.



We can also display the same graphs up to a fraction 1 of deleted nodes in order to better visualize what's happening.





The size of the Rest of the Components (RoC size) is 0 at the beginning because the starting network is the GCC and its GCC is itself.

We can observe that with the strategy of deleting one of the nodes with the highest degree we can very quickly decrease the size of the GCC. In the context of a targeted attack, with just a few attacks to the nodes with the highest degree, the graph will be split into many small clusters and the exchanges between will be slowed down, we can think. If the attacks do not target these important nodes, the GCC is affected, but it is way slower.

Code for the targeted deletion:

```
size_GCC_top = [len(GCC)]
size_rest_components_top = [total_number_nodes-len(GCC)]
fraction_deleted_node_top = [0]

# for i in range(1, total_number_nodes+1):
for i in range(1, 833):
    try:
        nodes_number = len(list(G_del))
        G_degrees = G_del.degree()
        node_degrees, nodes = zip(*sorted(zip(list(G_degrees.values()), list(G_degrees.keys()))))
        nodes_to_del = nodes[len(nodes)-min(10, len(nodes)):len(nodes)]
        indice_to_del = np.random.randint(1, min(len(nodes), 10))
        node_to_del = nodes_to_del[indice_to_del]

        G_del.remove_node(node_to_del)
        Connected_Components_Subgraphs = sorted(nx.connected_component_subgraphs(G_del), key=len, reverse=True)
        GCC = Connected_Components_Subgraphs[0]
        size_GCC_top += [len(GCC)]
        size_rest_components_top += [nodes_number -1 - len(GCC)]
        fraction_deleted_node_top += [i/total_number_nodes]
    except:
        pass
```