Assignment 2 by Ali Dhanani ULB 000470296

Q1.1) pseudo-code _ ErdosRenye

Here in this function we would pass on the number of nodes we want to create and the edges it should have, as per the question it should have average degree of 4.

Here I am using package networkx just for the creation of an empty graph and the process of adding nodes. In the question as already we are given the average degree as 4. I have used a formula by which with the number of average degree and the vertex I would find the number of edges that I require to run the value.

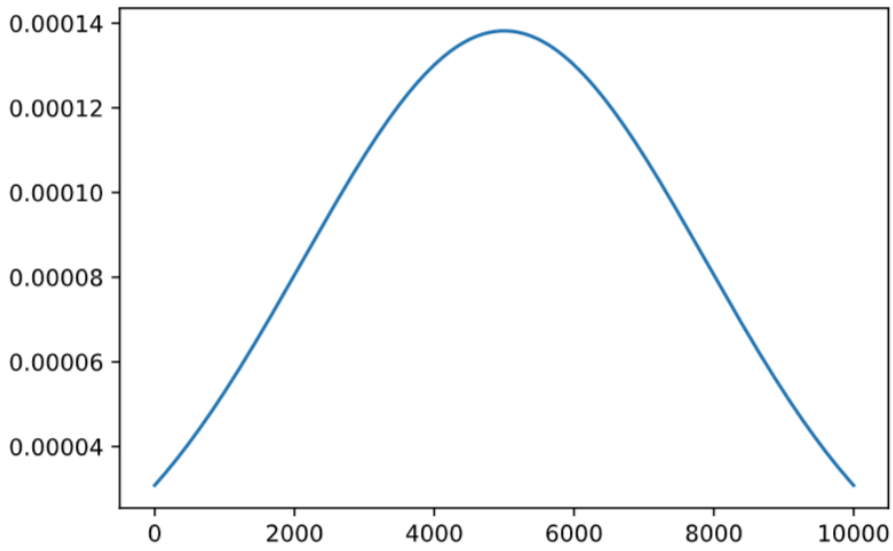
The formula I used was AverageDegree = 2(Edges)/Vertex. Here I given values were AverageDegree = 4 and Vertex = 10000 by which the number of edges were 20000

```
erdos_renyi(Nodes N):
        g = networkx.empty_graph(n)  // Here I have created all the 10000 nodes
         networkx.set_node_attributes(g, 0, "state")
        networkx.set_node_attributes(g, 0, "payoff")
         ne = 0
         for i in range(n):
            for j in range(2000): // run for 2000 times
                    if g.has_edge(i, j) == False: // As we know that 2 nodes can only have one connection between them
                        g.add_edge(i, j, attr_dict = {'added': ne})
                        ne += 1
        return g
```

For this we would pass the value like erdos_renyi(10000)

Q1.2): Here we need to find the mean and standard deviation of the graph, I have used the method of np.mean() and np.std() the method from numpy package.

mean = 4999.5 standard deviation = 2886.751331514372



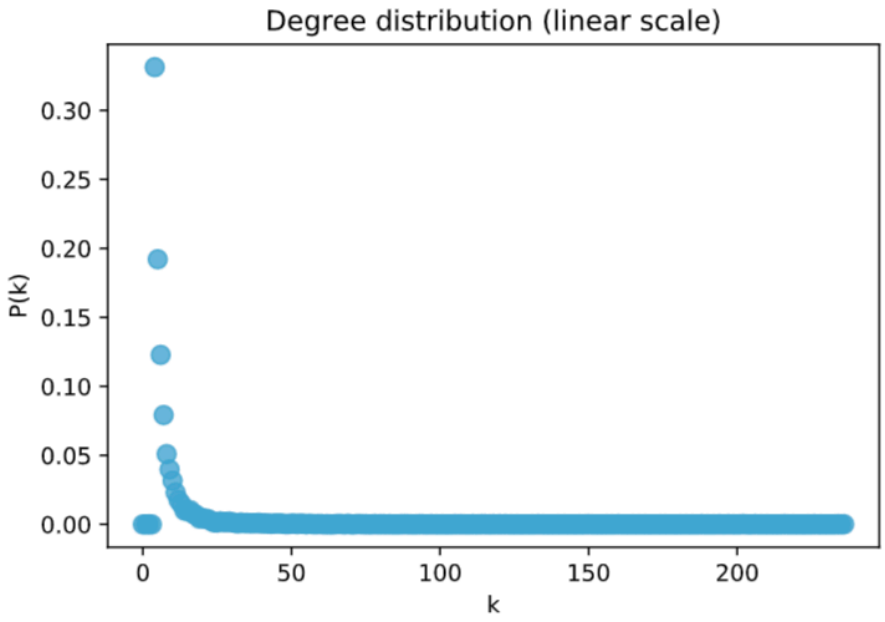
Q1.3) pseudo-code _ barabasi_albert

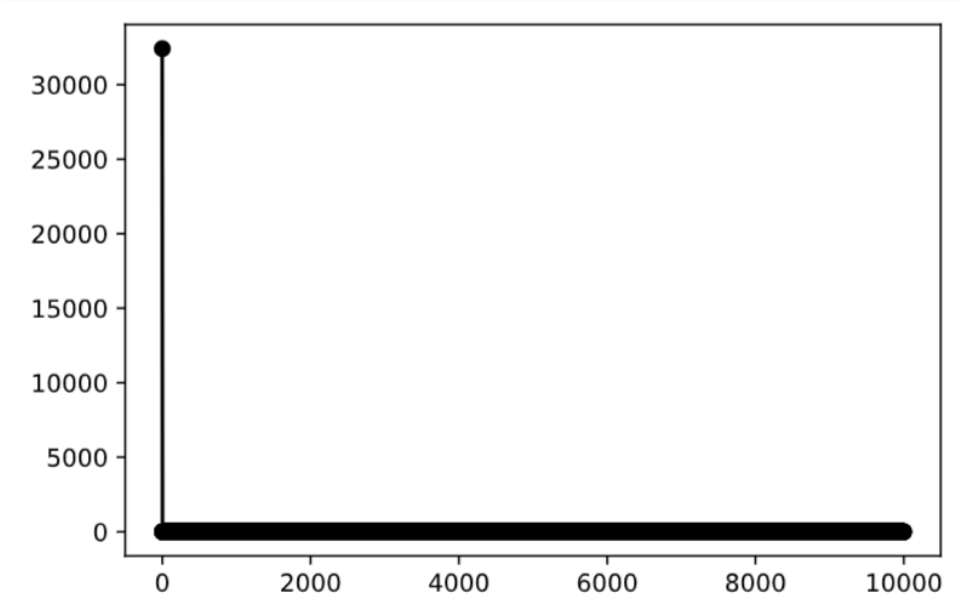
In this function we would be passing the number of nodes, and I have used the package networkx for the creation of the empty graph and also I have used the package for adding nodes to the graph. Here is idea is of adding the nodes in the way that using the probability formula and then choosing the 4 highest value of the array and then connecting the graph with them. The formula that I have used is the “grap.degree[j]/total”, the number of degree at that particular point divide by the total number of edges in that particular point, at that time I would be getting an array of float values that if I add together the sum would be 1. From that array of float values I would chose the 4 maximum values of float and Identify that nodes and connect that to the newly arrived node.

```
def barabasi_albert(N):
    grap = networkx.empty_graph(4)  // initially making 4 empty nodes
    ne = 0
    ArrayMax = []  // Array that would fill the probability
    ArrayOfDegree = [] // Array that would fill the number of nodes
      For the node 4 we already know that it would be connected 0, 1, 2, 3
    grap.add_edge(4, 0, attr_dict = {'added': ne})
    grap.add_edge(4, 1, attr_dict = {'added': ne})
    grap.add_edge(4, 2, attr_dict = {'added': ne})
    grap.add_edge(4, 3, attr_dict = {'added': ne})
    for i in range(5,N): // we are not going to run the for loop from 5 to 10000
        total = 0 // for calculating the total number of edges
        for j in range(i-1):
```

```
            total += grap.degree[j]
        for j in range(i-1):
            Pi = grap.degree[j]/total
            ArrayMax.append(Pi)
            ArrayOfDegree.append(j)
          value1 = np.random.choice(ArrayOfDegree, 4,  replace=False, p=ArrayMax) // Using
NP formula to get 4 max value of probability from the array
        for j in value1:    // Adding that 4 valus to the node
            grap.add_edge(i, j, attr_dict = {'added': j})
        ArrayMax.clear()
        ArrayOfDegree.clear()
    networkx.set_node_attributes(grap, 0, "state")
    networkx.set_node_attributes(grap, 0, "payoff")
    return grap // return the new Array
```

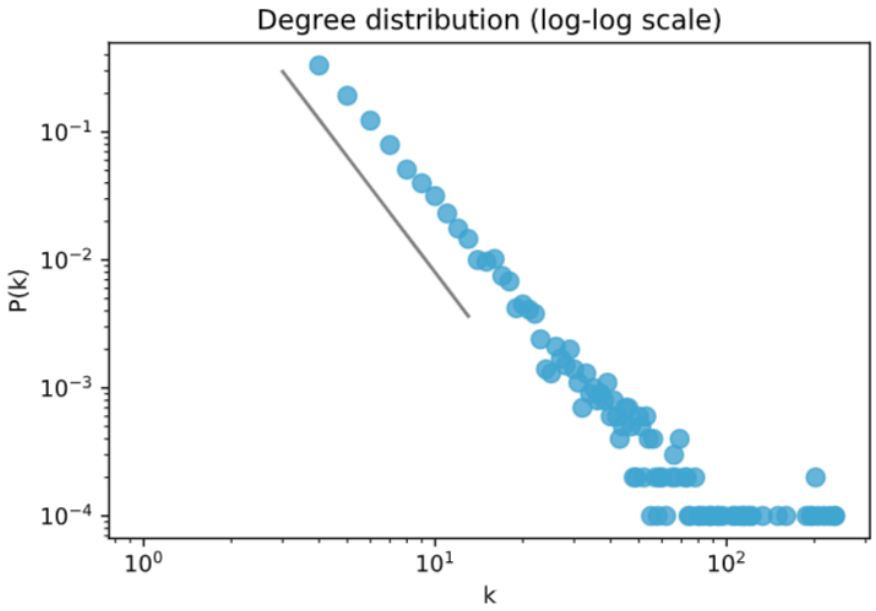
Q1.4) Here we had to show the degree distribution of the provided graph in a linear scale and then we had to plot an exponential distribution on the provided graph.





I have used here the function of K_distrib() for the creation of the linear scale degree distribution and I have used stats.expon for the creation of exponential distribution. Seeing here that both the graph of degree distribution and exponential distribution. We can see that after just the 4th node the value is constant to 0. When the nodes weren't connected the value of degree as well as the exponential was high but after a 4 node connection to every new node the value decreased to 0 and became constant as new node adding is an on going thing.

Q1.5)

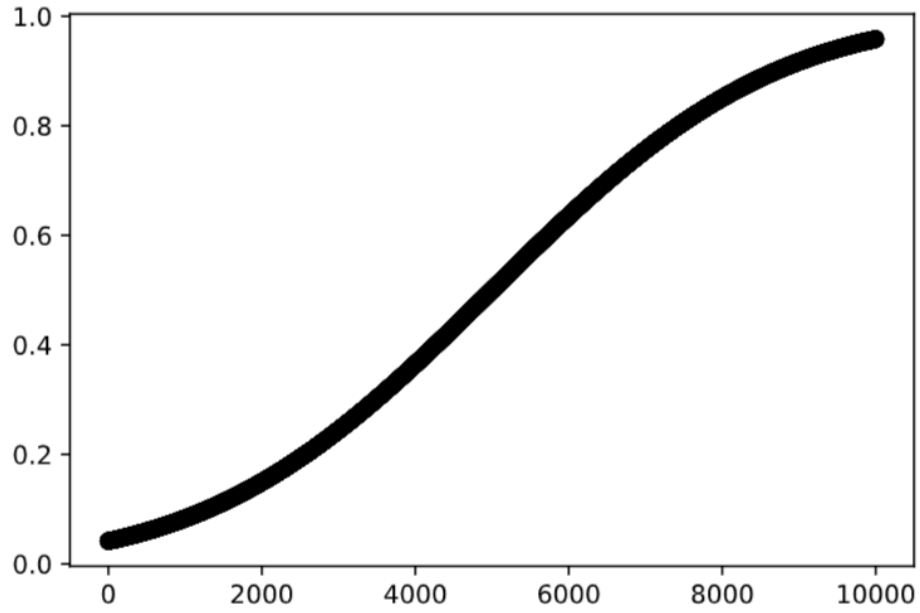


For the log log scale, I have used a function "k_distrib(graph=b,colour='#40a6d1', scale='log',alpha=.8, expct_lo=3, expct_hi=14, expct_const=8)", b stating the Array containing the value, for the log creation we have stated the min, max value stating the x and y axis. Expct_cont are expected for the line's length

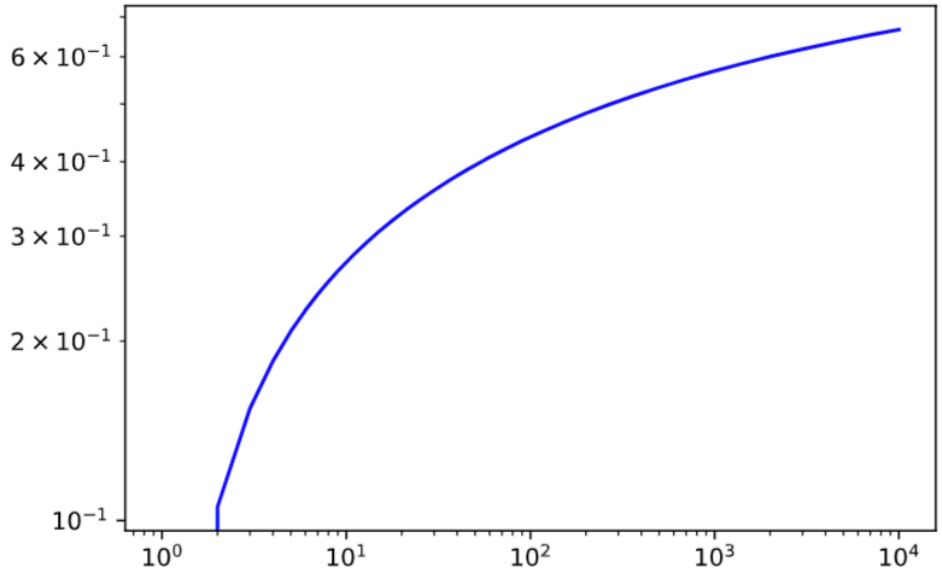
and fit intercept. Least square fit is used to find the best fit for a set of data points. A square line is drawn across the dotted line making a perfect fit of the scale. It would be fairly unusual to find psower law data for which the variance is actually constant across x values. However, let's take that constant variance as given and proceed with the analysis as a least squares problem.

Also it gives a bad estimation when fit to a power law, in general least square fit doesn't give you a good probability distribution.

Q1.6) this is a Complementary Cumulative Distribution using least square fit. We can see the increase in the nodes have increased the Complementary Cumulative Distribution.



Q1.7) Here I have fitted Complementary Cumulative Distribution the the maximum likelihood Method Found in the Power law package. Same as with the likelihood method we see the increase in Complementary Cumulative Distribution with an increase in nodes but here we can see a curve representation of the Complementary Cumulative Distribution.

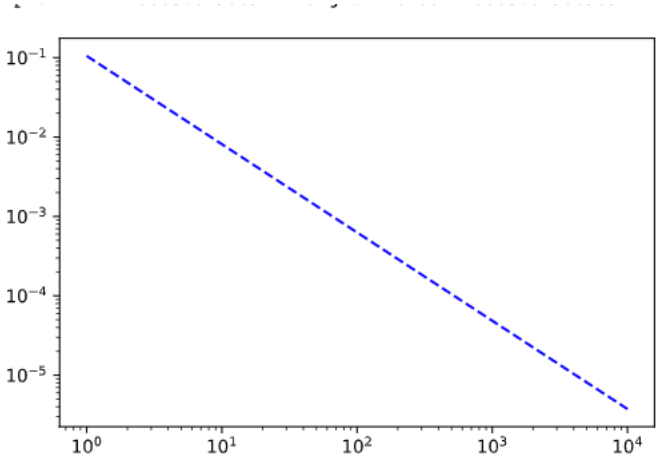


Q1.8) Here I would be using the package of powerlaw

Import powerlaw

fit = powerlaw.Fit(np.array(g)+1,xmin=1,discrete=True)

fit.power_law.plot_ccdf(color= 'b',linestyle='-',label='fit ccdf')



This is the pdf representation of the with maximum likelihood method.

Q1.9) R is Loglikelihood ratio of the two distributions' fit to the data. If Loglikelihood ratio of the two distributions' fit to the data. If 0, the second distribution is preferred. P is the Significance of the R value.

For the R, I am getting very high that the comparison has stated it to be infinite and for P I am getting value of nan value.

Q1.10)

$$p_i = \frac{K_i}{\sum_j K_j}$$

$$Mean(u) = \int_{x_{min}}^{\infty} x(p_i)\,dx$$

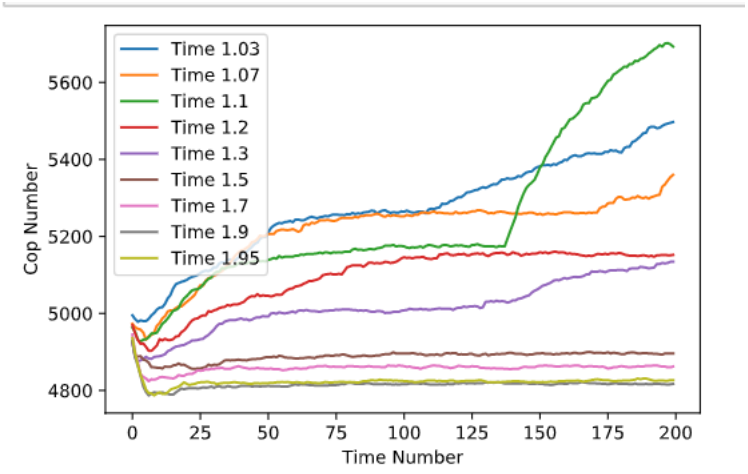
$$Std(u) = \int_{x_{min}}^{\infty} (x - Mean(u))^2 (p_i)\,dx$$

Mean of 2.7 = 7.5

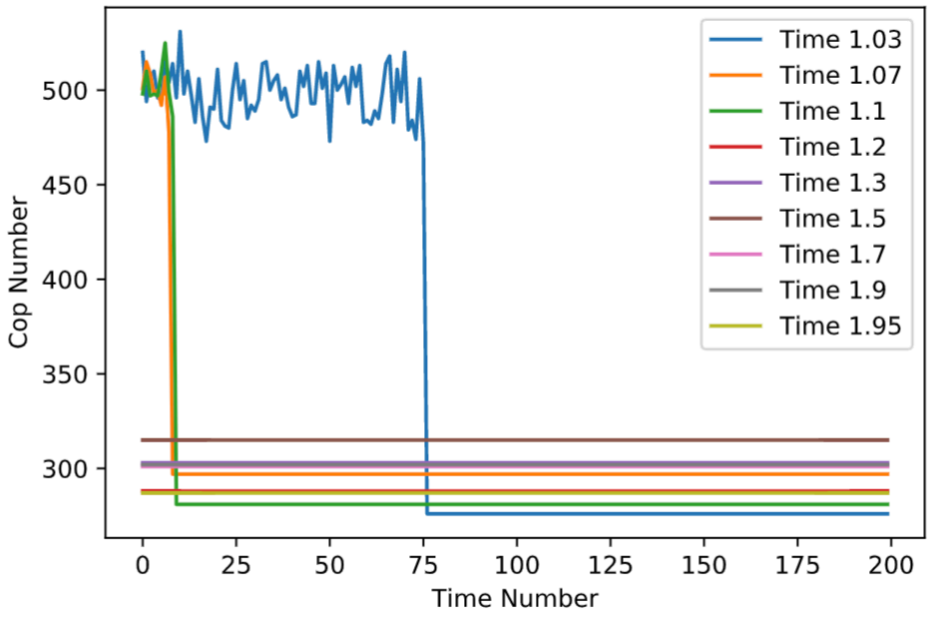
Std of 2.7 = 4.60977

Q2)1) With this formula actually we are comparing the payoffs that the current node has with its neighbor, and according to this we can say that when a node plays with its neighbor we would be comparing the result with their payoffs that our current action lets suppose is 0 and the other one has the action let's say 1 and when I play with my neighbor with my payoffs value and his payoff value I can see that he is better than me so when I take the probability of how my action would improve if I change my action to the action of my neighbor with respect to his number of node degree, and if the probability is like 0.5 I know at that point that it would be better for me to change my action to the action of my neighbor.

Q2)2)

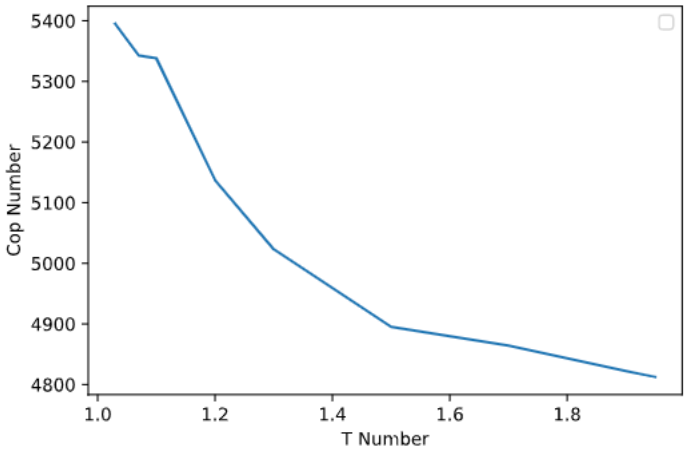


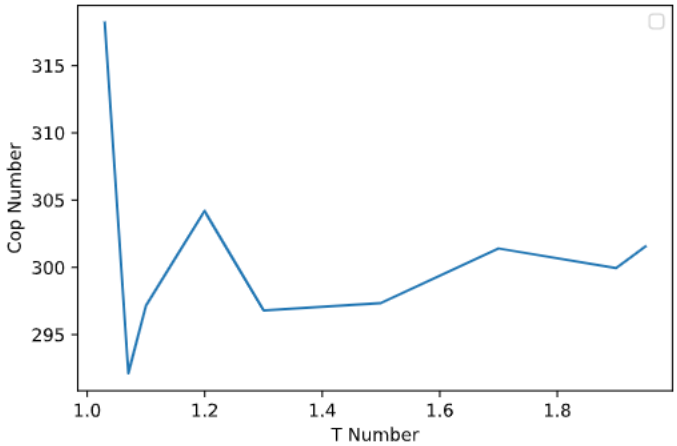
This is Barabasi_albert, here we can see with in increase in t number the coperation. Decrease from 4950. It takes 200 rounds to be stationary I have done this network for 10000 nodes.



This network is for Erdos-Renye, I have done this network for 1000 nodes, with. 200 rounds.

Q2)3) This is for barabasi with 10000 nodes.





This is for erdos_renyis for 1000 nodes

Q2)4) According to the graphs we can say that the cooperation level is better if we use the erdos_renyi then if use the barabasi_albert. We can say with an increase in the number of T, there is a slight decrease in the graph. All effects start from 4950 of the Cooperate level.