COMPLEX NETWORKS

LAB02: Structural descriptors of complex networks

For this practical work, NetworkX Python library has been used, to create the empty graph, to add the chosen number of nodes to it, to add the edges and to plot the entire network.

Erdös-Rényi (ER) network: G(N,p)

Figure 1 shows different networks created for different combinations of nodes and probabilities such that the average degree is 3, 5 and 10.

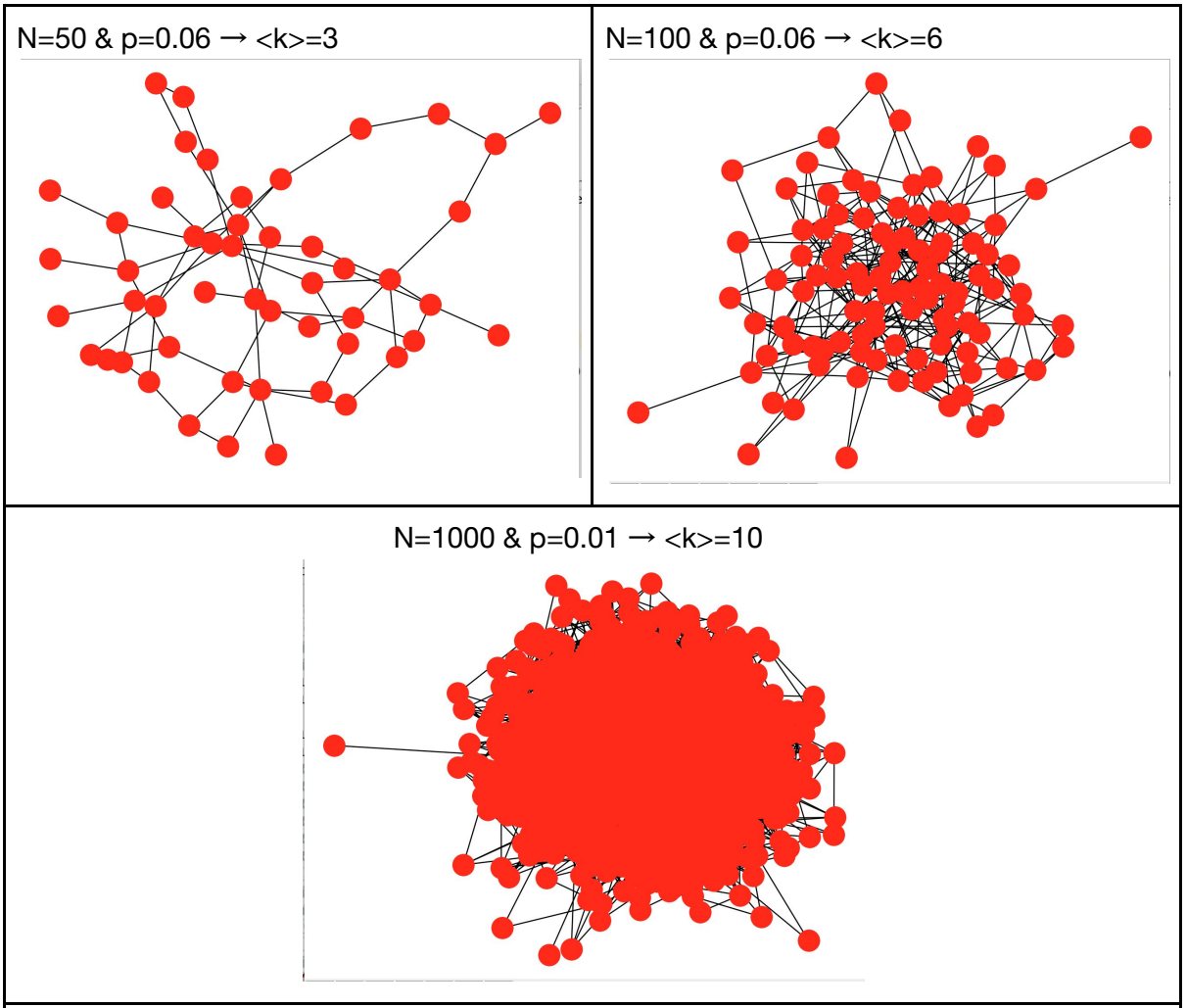

Figure 1: Networks with <k>=3, <k>=6 and <k>=10

Degree distributions

As the computation of the factorial for higher numbers is a limitation, Figures 2 and 3 show the **experimental** (blue) and **theoretical** (red) **linear** degree distributions for 1000 and 10000 nodes and lower probability values (0.1 and 0.001).

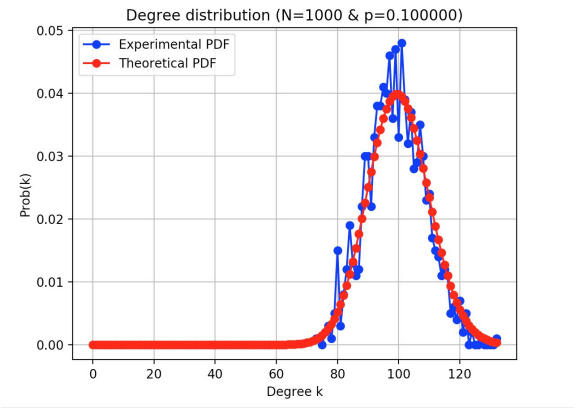

Figure 2: Linear PDF of ER Network with 1000 nodes and probability 0.1

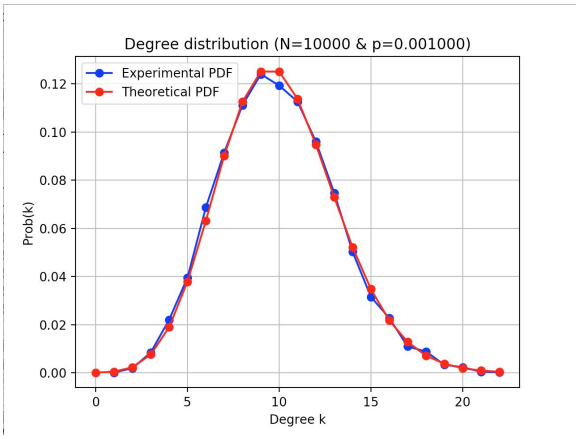

Figure 3: Linear PDF of ER Network with 10000 nodes and probability 0.001

ER Algorithm

The **algorithm** to create the ER Network is the following:

```
def ERnet(N,p):
    # Create empty graph
    G = nx.Graph()
    # Add n nodes to it. No edges yet.
    # add_nodes_from(nodes) --> nodes: iterable container
    G.add_nodes_from(range(N))

    # Add edges to the graph randomly:
    # For all possible pairs of nodes
    # 1. Take a pair of nodes
    # 2. get a random number between 0 and 1
    # 3. if r <= p --> add edge, else ignore
    list_edges = []
    for i in range(N):
        for j in range(i+1,N):  # we don't need a->a edges or a->b and b->a
            r = random.random()
            if r <= p:
                G.add_edge(i,j)
                edge = (i,j)
                list_edges.append(edge) # keep created edges
    print('list of edges: {}'.format(list_edges))

    # Plot the Network
    if (N<=50):
        nx.draw(G)
        plt.show()
    print('nodes: {}, probability: {}'.format(N,p))

    return G
```

Barabási & Albert (BA) preferential attachment model

Figure 4 shows different networks created for 100 nodes, with initial number of nodes = 5, and different numbers of edges that each new node forms with the existing nodes, where the initial network of 5 nodes is connected as a path.

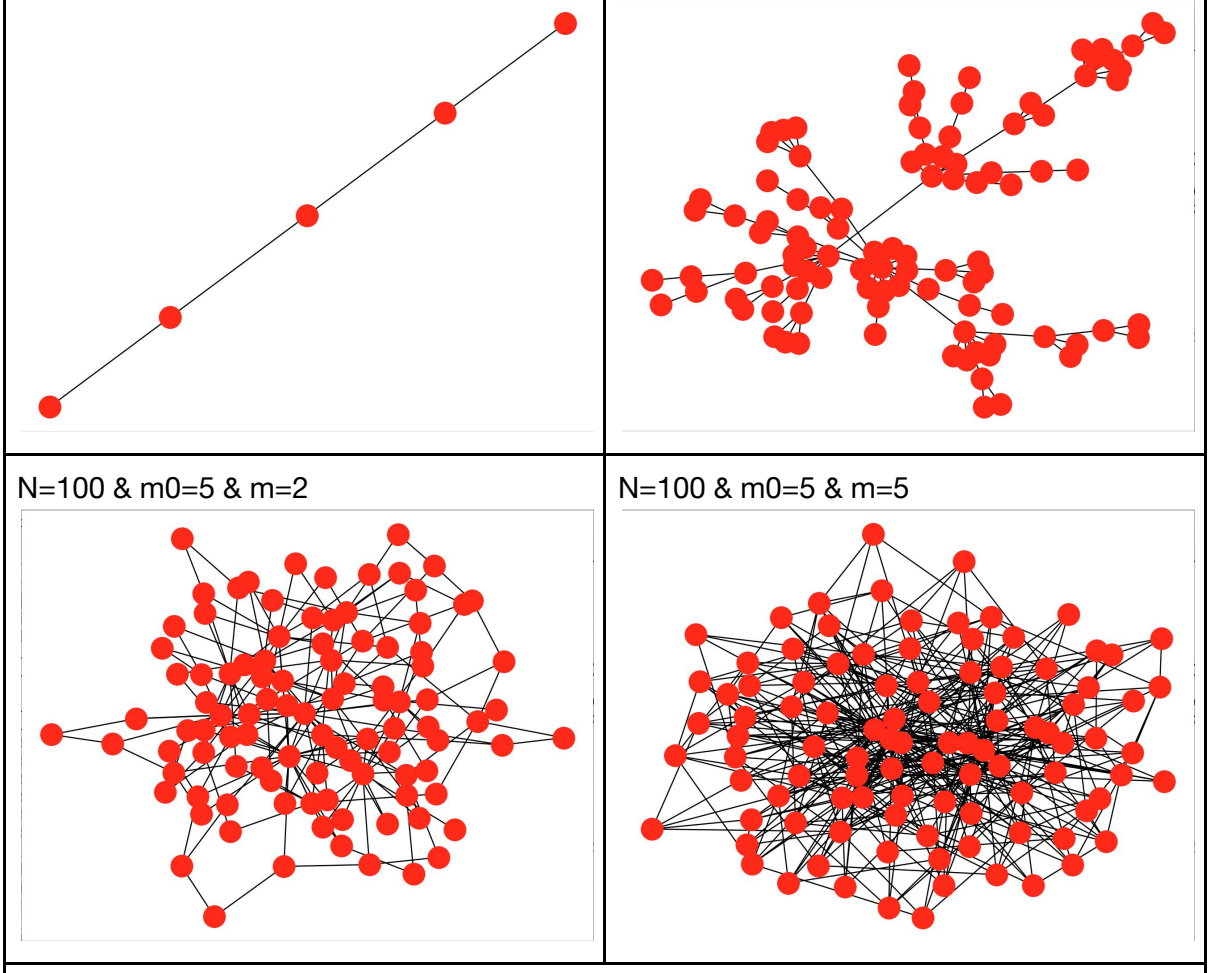

Figure 4: Initial network and nodes added with m=1, 2 and 5 for N=100 and m0=5

Degree distributions

To plot the degree distributions, we assume that the minimum degree is m, the number of edges each new node forms with the existing nodes. However, the first m0 initial nodes have degree 1 and 2 as we created it as a path, and we are not representing them.

Figures 5 and 6 show the **experimental** (blue) and **theoretical** (red) **linear** degree distributions for 1000 nodes (given the initial path of 50 nodes and 20 edges for each new node) and 10000 nodes (given the initial path of 750 nodes and 120 edges for each new node).

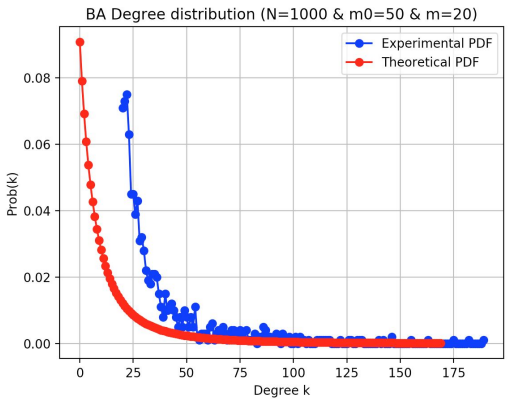

Figure 5: Linear PDF of BA Network with 1000 nodes

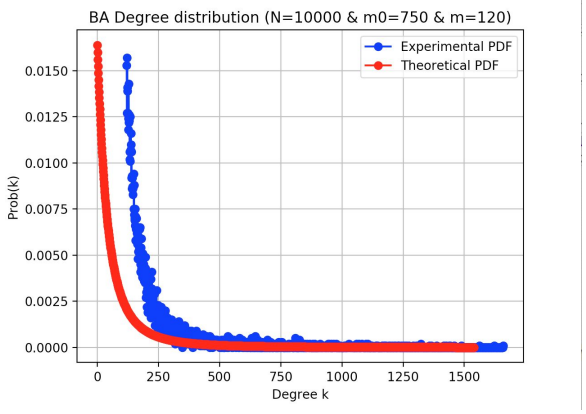

Figure 6: Linear PDF of BA Network with 10000 nodes

Estimation of probabilities

Figures 7 and 8 show **log-log** degree distribution for the estimation of the probability of k, for 1000 and 10000 nodes.

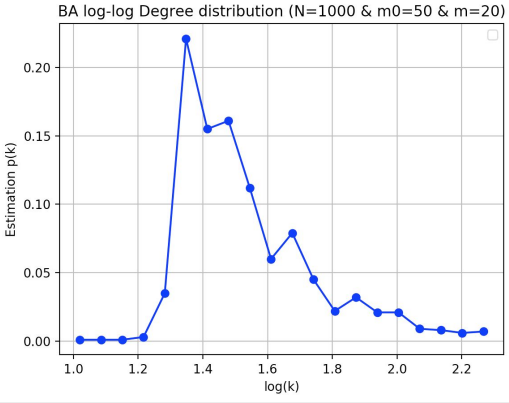

Figure 7: log-log PDF of BA Network with 1000 nodes

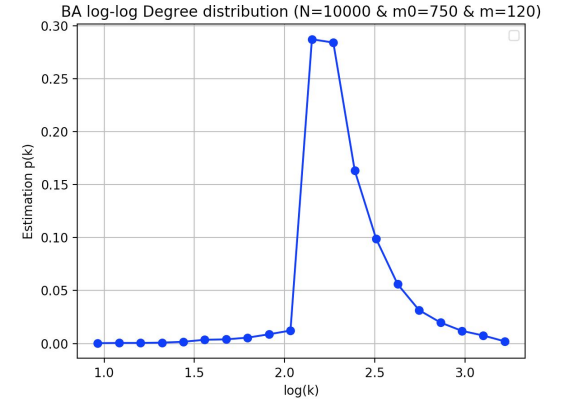

Figure 8: log-log PDF of BA Network with 10000 nodes

BA Algorithm

The **algorithm** to create the BA Network is the following:

```
def BAnet(N,m0,m):
    # Create a path graph with the initial number of nodes
    G = nx.path_graph(m0)
    nx.draw(G)
    plt.show()
    temp = nx.degree(G)
    print('degrees initial nodes: {}'.format(temp))
    k = [i[1] for i in temp]
    print('degrees: {}'.format(k))


    for i in range(m0,N):   # add new nodes (N-m0) to the network
        # 1. add the node to the network
        G.add_node(i)
        # 2. add m edges to this node (one by one)
        # 2.1. Preprocessing
        # dictionary of degrees
        temp = nx.degree(G)
        deg = [i[1] for i in temp]
        #print('deg: {}'.format(deg))

        # dictionary of probabilities (more degree more probability)
        # p = k / sum(k)
        node_probs = {}
        for n in G.nodes():
            #print(n)
            node_probs[n] = deg[n]/sum(deg)

        # order of nodes -- list cumulative probabilities
        node_probs_cum = []
        acc = 0
        for k,v in node_probs.items():
            temp = [k,acc+v]
            node_probs_cum.append(temp)
            acc = acc + v

        # 3. while the number of edges added is != m
        nodes_used = []
        num_added = 0
        edges_used = []
        new_edges = []

        while (num_added < m):
            # choose a random number btw 0 and 1
            r = random.random()
            # ens quedem amb el primer node que té cumulative >= random number
            cum = 0
            k = 0
            while(not(node_probs_cum[k][1] >= r)):
                cum = node_probs_cum[k][1]
                k = k+1
            node = node_probs_cum[k][0]
            # if the node is not used yet and the edge is not in the list of edges,
            # we add it to the list of nodes used
            if (node_probs_cum not in nodes_used) and
              (node_probs not in edges_used):
                nodes_used.append(node)
                G.add_edge(i,node)              # create new edge
                num_added = num_added +1
```

```
                edges_used.append((i,node))
                new_edges.append((i,node))

        print('new_edges (it #{}): {}'.format(i,new_edges))

    print('node_probs: {}'.format(node_probs))
    print('node_probs_cum: {}'.format(node_probs_cum))

# Plot the Network
if N<=50:
    nx.draw(G)
    plt.show()

return G
```