In the name of God

درس : علوم شبکه

استاد : دکتر خوانساری

دانشجو : توحید حقیقی سیس (۸۳۰۵۹۸۰۲۱)

پیاده سازی سوال ۱ فصل ۵ :

With the help of a computer, generate a network with $N = 10^4$ nodes using the Barabási-Albert model with $m = 4$. Use as initial condition a fully connected network with $m = 4$ nodes.

(a) Measure the degree distribution at intermediate steps, namely when the network has 10^2, 10^3 and 10^4 nodes.

(b) Compare the distributions at these intermediate steps by plotting them together and fitting each to a power-law with degree exponent γ. Do the distributions "converge"?

(c) Plot together the cumulative degree distributions at intermediate steps.

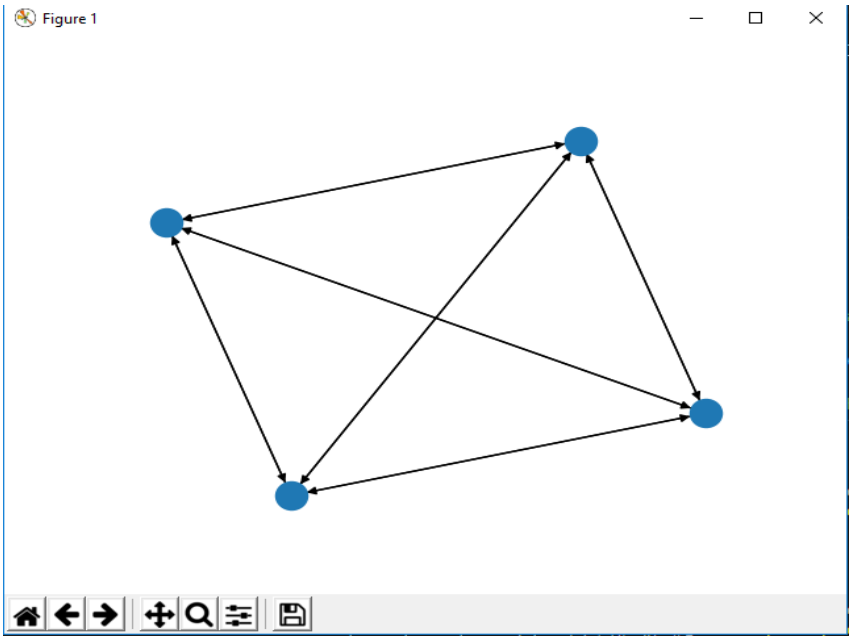
(d) Measure the average clustering coefficient in function of N.

(e) Following Figure 5.6a, measure the degree dynamics of one of the initial nodes and of the nodes added to the network at time $t = 100$, $t = 1{,}000$ and $t = 5{,}000$.

برای حل این قسمت ابتدا یک گراف ابتدایی تولید میکنیم

که گراف کامل با ۴ نود است .

برای این قسمت از کتابخانه های Networkx , matplotlib.pyplot

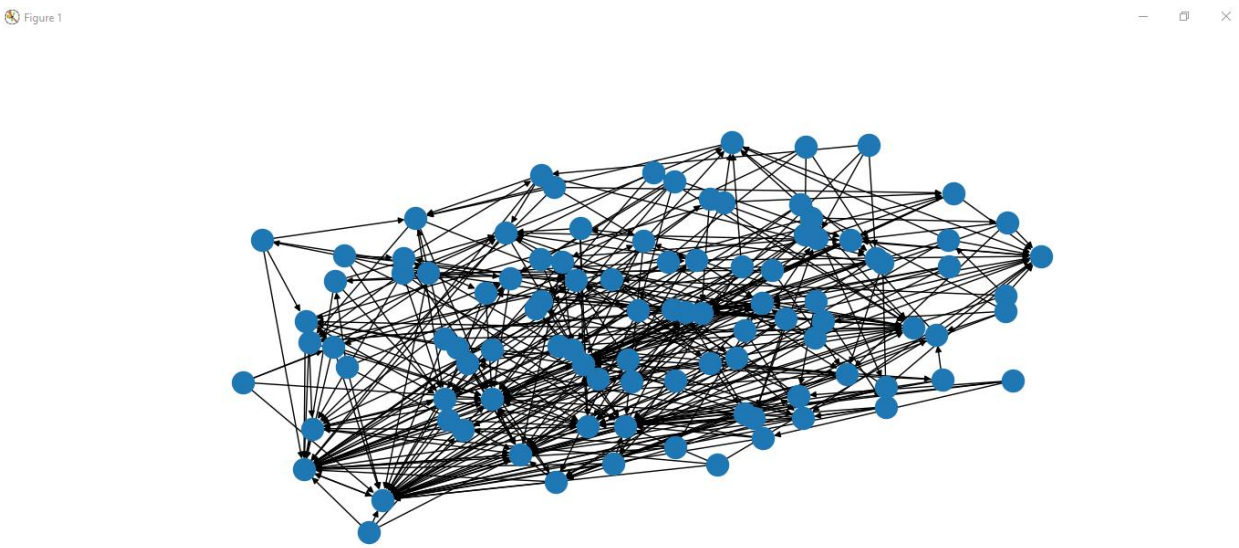


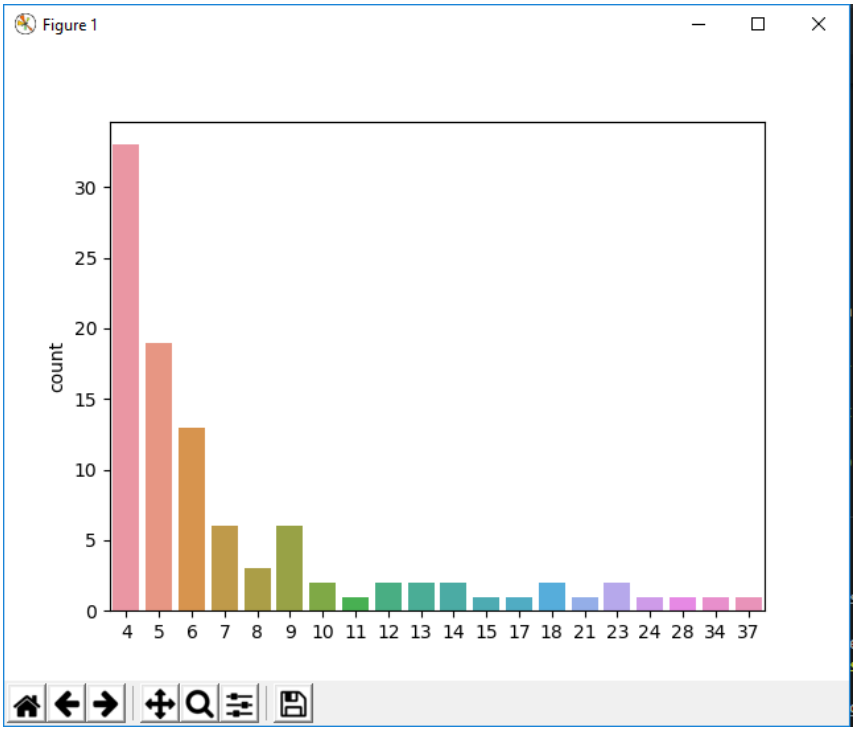
که کد ان به صورت زیر است :

```
    def make_initial_Complete_graph(self):
        ig = nx.empty_graph(0, create_using=nx.DiGraph())
        ig.add_nodes_from([0, 1, 2, 3])
        ig.add_edges_from([(0,1), (0,2), (0,3),
                  (1,0), (1,2), (1,3),
                  (2,1), (2,0), (2,3),
                  (3,1), (3,2), (3,0),])
        return ig
```

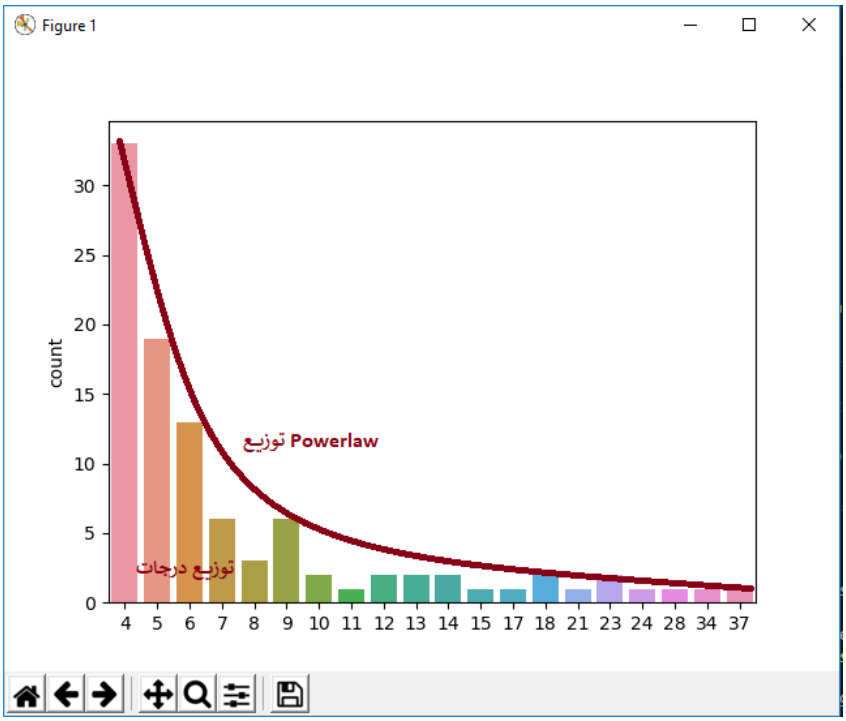
در روی سوال گفته که هر نود جدید به ۴ نود دیگر به صورت رندوم وصل شود

```
    def barabasi_albert(self,initial_graph, n, m, seed):
        random.seed(seed)
        #گراف اولیه
        g = initial_graph
        #تا جایی که کل نود ها اضافه شود
        while(len(g)<n):
            #در اوردن لیست درجات هر نود گراف
            node_list, degree_list = zip(*list(g.degree))
            #احتمال تک تک درجات
            probability_list = [x for x in degree_list]
            #مجموع درجات لیست
            s = sum(probability_list)
            #احتمال هر لیست درجات
            probability_list = [x/s for x in probability_list]
            #  تا نود توسط احتمال ٤انتخاب
            selected_nodes = np.random.choice(np.array(node_list),4 ,replace=Fals
e ,p=probability_list)
            # لیست درجات نود
            selected_nodes = list(selected_nodes)
            #افزودن نود جدید
            g.add_node(len(g))
            # افزودن هر یال نود
            for i in range(m):
                g.add_edge(len(g)-1, selected_nodes[i])
        return g
```





با توجه به شکل بالا میبینیم که توزیع powerlaws دارد یعنی تعداد درجات کم زیاد و درجات بالا کم است و همه نود ها در یک نود اصلی به هم وصل شده اند



برای محاسبه Cluster Coeffitiont از روش زیر استفاده میکنیم و خروجی به صورت زیر است

```
[31, 28, 32, 45, 15, 25, 12, 16, 16, 18, 15, 10, 14, 8, 9, 6, 21, 15, 10, 8, 9, 7, 8, 5, 8, 8, 8, 7, 11, 8, 7, 8, 7, 9, 4, 7, 6, 5, 6, 5,
11, 8, 8, 5, 4, 5, 4, 8, 4, 5, 7, 7, 4, 5, 4, 4, 6, 5, 6, 5, 4, 6, 4, 4, 5, 5, 5, 7, 5, 5, 4, 6, 5, 5, 4, 4, 4, 4, 4, 4, 4, 4, 5, 5, 4,
 6, 4, 4, 5, 4, 4, 5, 4, 4, 4, 4, 4, 4, 4]
0.11983695967469579
```

```
def Cluster_Coeffistiont(self,graph):
    return nx.algorithms.cluster.average_clustering(graph)
```

مقدار این تابع در شکل بالا اورده شده است

پیاده سازی سوال ۲ فصل ۵ :

Consider a variation of the Barabási-Albert model, where at each time step a new node arrives and connects with a directed link to a node chosen with probability

$$\Pi(k_i^{in}) = \frac{k_i^{in} + A}{\sum_j (k_j^{in} + A)} .$$

Here k_i^{in} indicates the in-degree of node i and A is the same constant for all nodes. Each new node has m directed links.

(a) Calculate, using the rate equation approach, the in- and out-degree distribution of the resulting network.

(b) By using the properties of the Gamma and Beta functions, can you find a power-law scaling for the in-degree distribution?

(c) For $A = 0$ the scaling exponent of the in-degree distribution is different from $\gamma = 3$, the exponent of the Barabási-Albert model. Why?

در این سوال مانند سوال قبل عمل میکنیم با این تفاوت که احتمال نود ها متفاوت است و به نود های با احتمال بالا بیشتر ممکن است وصل شود

بر اساس درجات ورودی نود ها عمل میکند هر چه درجه ورودی نود ها بیشتر شود احتمال برخورد بیشتر است

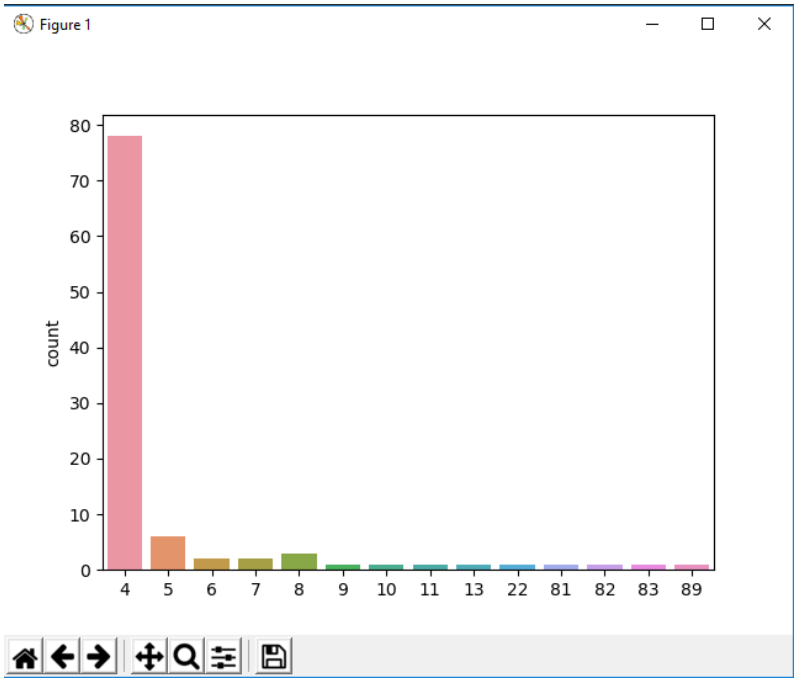
کد این قسمت به صورت زیر است:

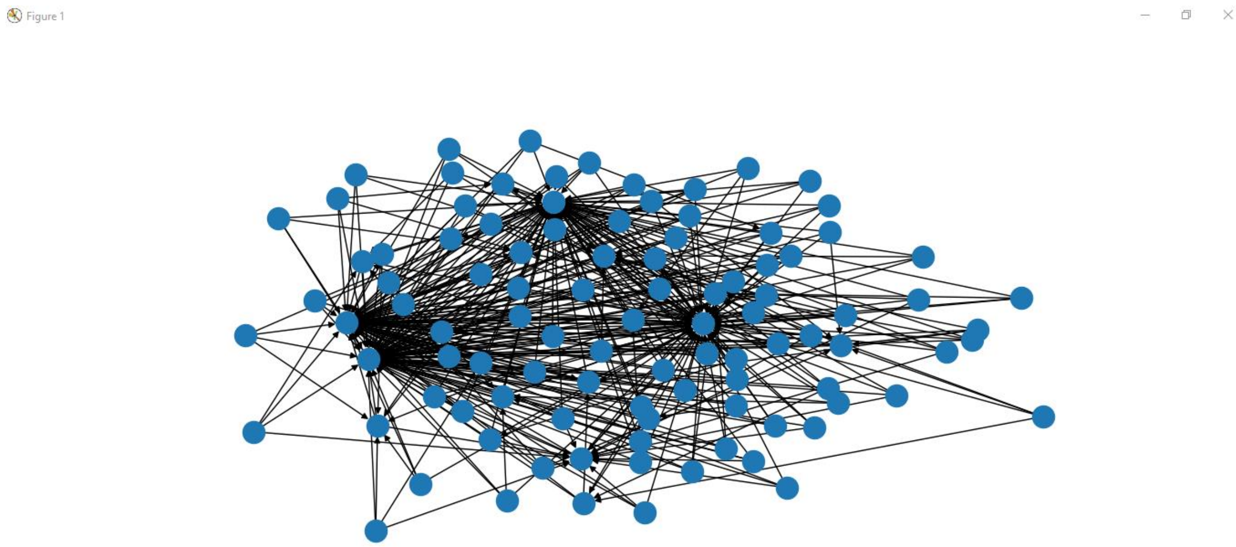
```
barbashi=Barbashi_Albert_Model(1234)
ig=barbashi.make_initial_Complete_graph()

barbashi.show_plot(ig)

ig_empty=barbashi.make_Empty_graph()
Directed_graph=barbashi.directed_barabasi_albert(ig,100,4,0.1,barbashi.seed)
l=barbashi.Get_Degrees(Directed_graph)
print(l)

barbashi.Destribution_Show_Graph(l)
barbashi.show_plot(Directed_graph)

print(barbashi.Cluster_Coeffistiont(Directed_graph))
```

کد Directed Barbashi Albert Model به صورت زیر است :

```
    def directed_barabasi_albert(self,initial_graph, n, m, A, seed):
        random.seed(seed)
        # گراف اولیه
        ig = initial_graph
        # به تعداد کل نود های داده شده اجرا میشود
        while(len(ig)<n):
            # لیست درجات را در می آورد
            node_list, degree_list = zip(*list(ig.in_degree))
            # احتمال درجات را محاسبه میکند با فرمول روی سوال
            probability_list = [x+A for x in degree_list]
            s = sum(probability_list)
            probability_list = [x/s for x in probability_list]
            selected_nodes = np.random.choice(np.array(node_list),4 ,replace=Fals
e ,p=probability_list)
            selected_nodes = list(selected_nodes)
            ig.add_node(len(ig))
            for i in range(m):
                ig.add_edge(len(ig)-1, selected_nodes[i])
        return ig
```

خروجی های این قسمت به صورت زیر است :

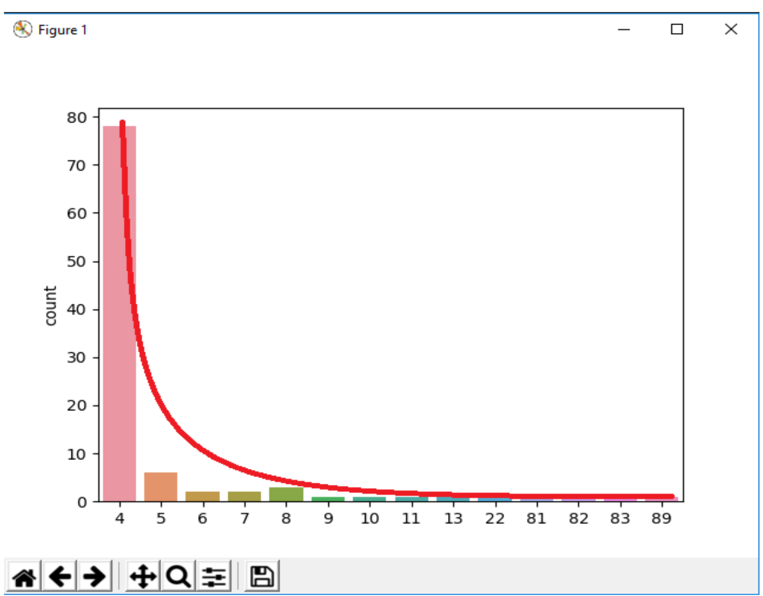




این قسمت نیز به صورت زیر است : Cluster Coeffitiont

```
[82, 83, 81, 89, 4, 11, 5, 7, 4, 22, 5, 9, 4, 4, 4, 8, 4, 5, 8, 6, 4, 4, 4, 5, 13, 4, 5, 8, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 5, 10, 4, 7, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 6, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4]
0.6807520894619947
PS C:\Users\tohid\Desktop\ردکد یراس اوخ لمف 5>
```

و مقایسه ان با PowerLaw به صورت زیر خواهد بود :



پیاده سازی سوال ۳ فصل ۵ :

5.3. Copying Model

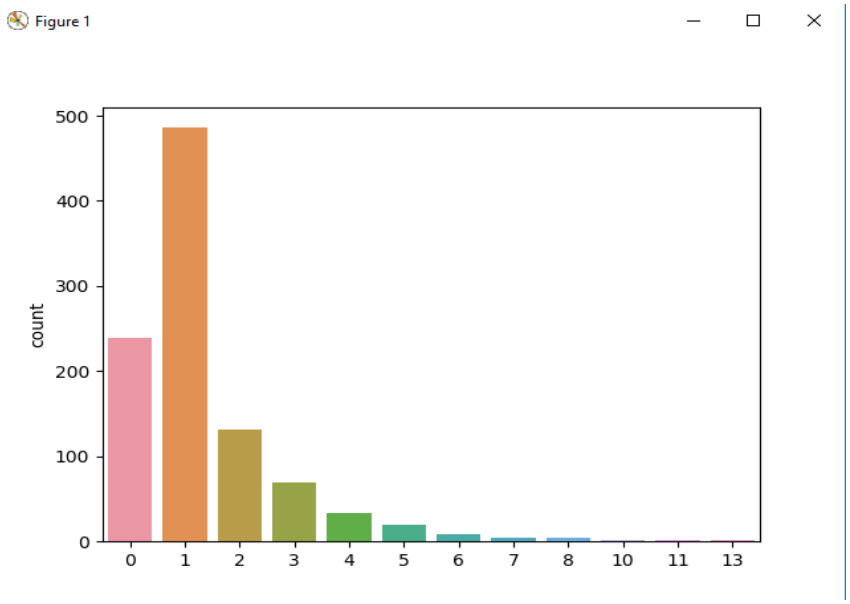
Use the rate equation approach to show that the directed copying model leads to a scale-free network with incoming degree exponent $\gamma_{in} = \frac{2-p}{1-p}$.

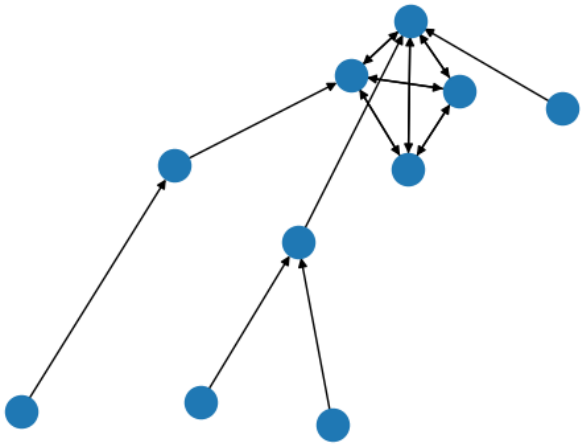
در این قسمت مفهوم Copying Model را پیاده میکنیم

به این صورت که نود جدید که وارد شد یه تاس میاندازیم اگر از احتمال مد نظر ما کمتر بود به آن نود وصل میکنیم وگر نه به نود های همسایه ان وصل میکنیم

کد این قسمت به صورت زیر است :

```
def copying_model(self,initial_graph, n, p, seed):
    random.seed(seed)
    # ایجاد نود اولیه
    g = initial_graph
    while(len(g)<n):
        # لیست نود ها
        nodes = list(g.nodes)
        # انتخاب تصادفی نود ها
        selected_node = random.choice(nodes)
        # افزودن نود جدید
        g.add_node(len(g))
        # انتخاب عدد رندوم
        rn = random.random()
        # اگر عدد رندوم از احتمال مد نظر ما کوچکتر بود به ان نود وصل میشود
        if(rn<p):
            g.add_edge(len(g)-1, selected_node)
        else:
            # وصل شدن به همسایه ها
            selected_nodes = [y for x,y in list(g.edges) if y==selected_node]
            for i in range(len(selected_nodes)):
                g.add_edge(len(g)-1, selected_nodes[i])
    return g
```





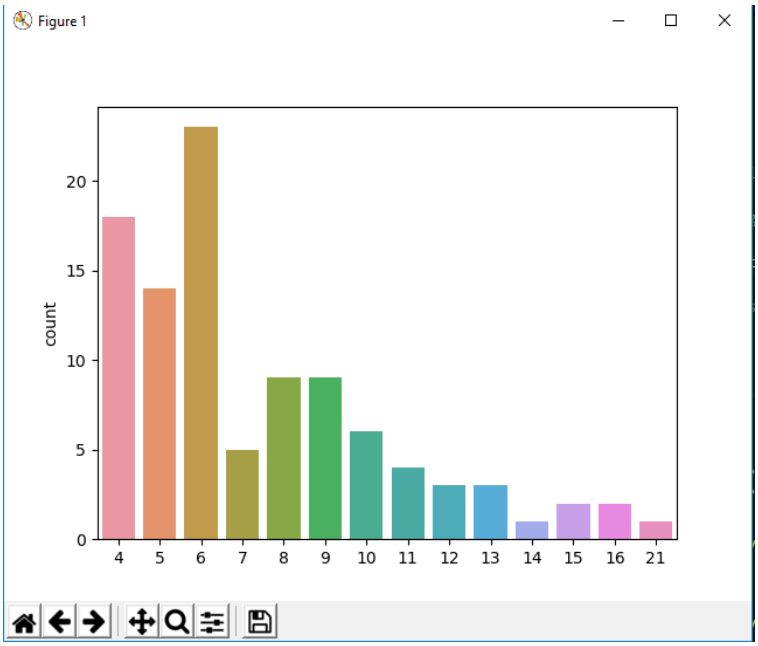
پیاده سازی سوال ۴ فصل ۵ :

Derive the degree distribution (5.18) of Model A, when a network grows by new nodes connecting randomly to m previously existing nodes. With the help of a computer, generate a network of 10^4 nodes using Model A. Measure the degree distribution and check that it is consistent with the prediction (5.18).

در این قسمت Proffestional Attachment وجود ندارد و فقط Growth وجود دارد

کد این قسمت به صورت زیر است :

```
def growth_Without_PrefesionalAttachment(self,initial_graph, n, m, seed):
    random.seed(seed)
    g = initial_graph
    while(len(g)<n):
        nodes = list(g.nodes)
        selected_nodes = [random.choice(nodes) for x in range(m)]
        g.add_node(len(g))
        for i in range(m):
            g.add_edge(len(g)-1, selected_nodes[i])
    return g
```



همان طور که از پراکندگی مشخص است دارد کم کمک از حالت PowerLaw خارج میشود و Random میشود