Humza Salman mhs180007

In [19]:
```
import networkx as nx
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
plt.ioff()
import sys
sys.path.append('../d3networkx/')
import d3networkx as d3nx
from d3graph import D3Graph, D3DiGraph
import asyncio
import random
import randomnet
```

The `randomnet` import statement provides functions to build local attachment and small world random networks. This small world network is slightly different from the version that is implemented in NetowrkX.

Section 15.1.0: Small World

Small World Networks

This function generates a small world network, where `n` nodes are connected to `q` neighboring nodes "around the circle" and with probability `p` to all other nodes. Even though the `randomnet.py` file contains a very similar function, this version has some extra code to lay out the network in an intuitive way (with the nodes in a circle).

In [2]:
```
async def small_world(n,q,p,G=None,d3=None,x0=300,y0=300):
    '''
    q must be even
    '''
    if d3:
        d3.set_interactive(False)
    if G is None:
        G = D3Graph()
    for i in range(n):
        G.add_node(i)
        if d3:
            x = 200*cos(2*pi*i/n) + x0
            y = 200*sin(2*pi*i/n) + y0
            d3.position_node(i,x,y)
    # add the regular edges
    for u in range(n):
        for v in range(u+1,int(u+1+q/2)):
            v = v % n
            G.add_edge(u,v)
    print(nx.diameter(G))
    if d3:
        d3.update()
        await asyncio.sleep(3)
```

```
        d3.set_interactive(True)

    # add the random edges
    for u in range(n):
        for v in range(u+1,n):
            if not G.has_edge(u,v):
                if random.random() <= p:
                    G.add_edge(u,v)
    print(nx.diameter(G))
    return G
```

In [3]:

```
d3 = await d3nx.create_d3nx_visualizer()
```

```
websocket server started...visualizer connected...
```

Now with the visualizer running, we will visualize a small world network

In [4]:

```
G = D3Graph()
d3.set_graph(G)
G = await small_world(20,4,0.1,G,d3)
```

```
5
networkx connected...4
```

In [5]:

```
print('HIGHLIGHTED QUESTION -  write down your (visual) observations about the diamete
print('With local connections we have a higher value for the diameter as compared to w
```

```
HIGHLIGHTED QUESTION -  write down your (visual) observations about the diameter of t
he network with only the local connections and then with the random connections added
in. In the first case, without random connections, why is the diameter so high?
With local connections we have a higher value for the diameter as compared to with ra
ndom connection where we have a lower value for the diameter. Without random connecti
ons the diameter is higher since our longest shortest path between any two nodes is
n/q nodes long since we add edges between nodes u and v with v being v % n, whereas t
he random graph will always have a path of length <= n/q.
```

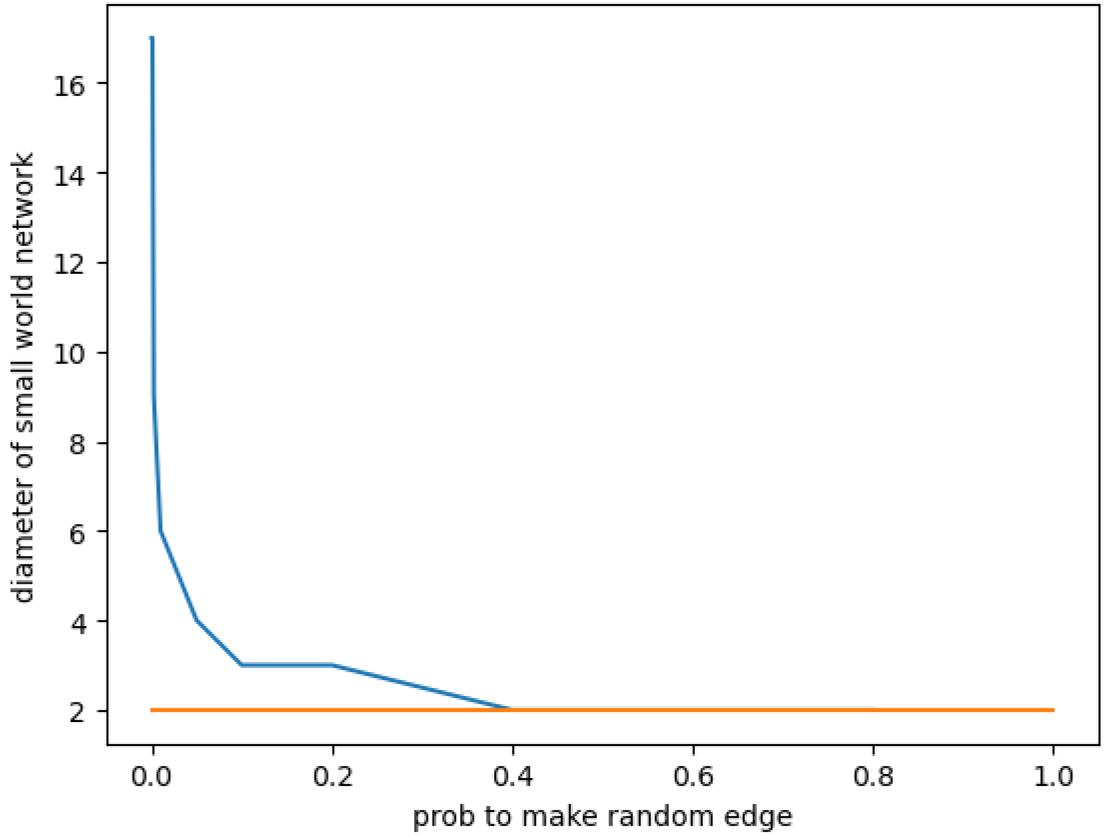
In [ ]:

Now let's plot the convergence of the small world effect.

In [6]:

```
n = 100
P = [0,0.0001,0.001,0.0025,0.005,0.01,0.05,0.1,0.2,0.4,0.6,0.8]
d = []
for p in P:
    G = randomnet.small_world_graph(n,6,p)
    d.append(nx.diameter(G))
    # calculate the diameter and store it for plotting below

print('HIGHLIGHTED QUESTION - plot the diameter versus the p values.')
## Plot the Convergence
plt.figure()
plt.plot(P, d) # change x and y to your "x" and "y" values
plt.plot([0.0001,1],[(log10(n)),(log10(n))])
plt.xlabel('prob to make random edge')
plt.ylabel('diameter of small world network')
plt.show()
```

```
HIGHLIGHTED QUESTION - plot the diameter versus the p values.
```



In [162…

```
print('HIGHLIGHTED QUESTION - Make a brief comment on the how and why these random cor
print('Since the small world effect claims the diameter should scale proportionally tc
```

```
HIGHLIGHTED QUESTION - Make a brief comment on the how and why these random connectio
ns (qualitatively) lead to the small world effect.
Since the small world effect claims the diameter should scale proportionally to log
(n), we see that the diameter of small word network converges to the orange line whic
h represents log(n). So, it is telling us that the diameter is decreasing as we incre
ase our probability to make an edge between any two nodes which causes a shorter aver
age path between any two nodes which causes the small world effect.
```

Sections 8.1-8.4.1: Power Law Networks

Fitting Power Law

The following helper functions provide easy access to the degree sequence and the degree and cumulative degree distributions.

In [8]:

```
def degree_sequence(G):
    return [d for n, d in G.degree()]

def degree_distribution(G,normalize=True):
    deg_sequence = degree_sequence(G)
    max_degree = max(deg_sequence)
    ddist = zeros((max_degree+1,))
    for d in deg_sequence:
        ddist[d] += 1
    if normalize:
```

```
        ddist = ddist/float(G.number_of_nodes())
    return ddist

def cumulative_degree_distribution(G):
    ddist = degree_distribution(G)
    cdist = [ ddist[k:].sum()  for k in range(len(ddist)) ]
    return cdist
```

The following function, which you must complete, plots the degree distribution and calculates the power law coefficient, α.

In [201…

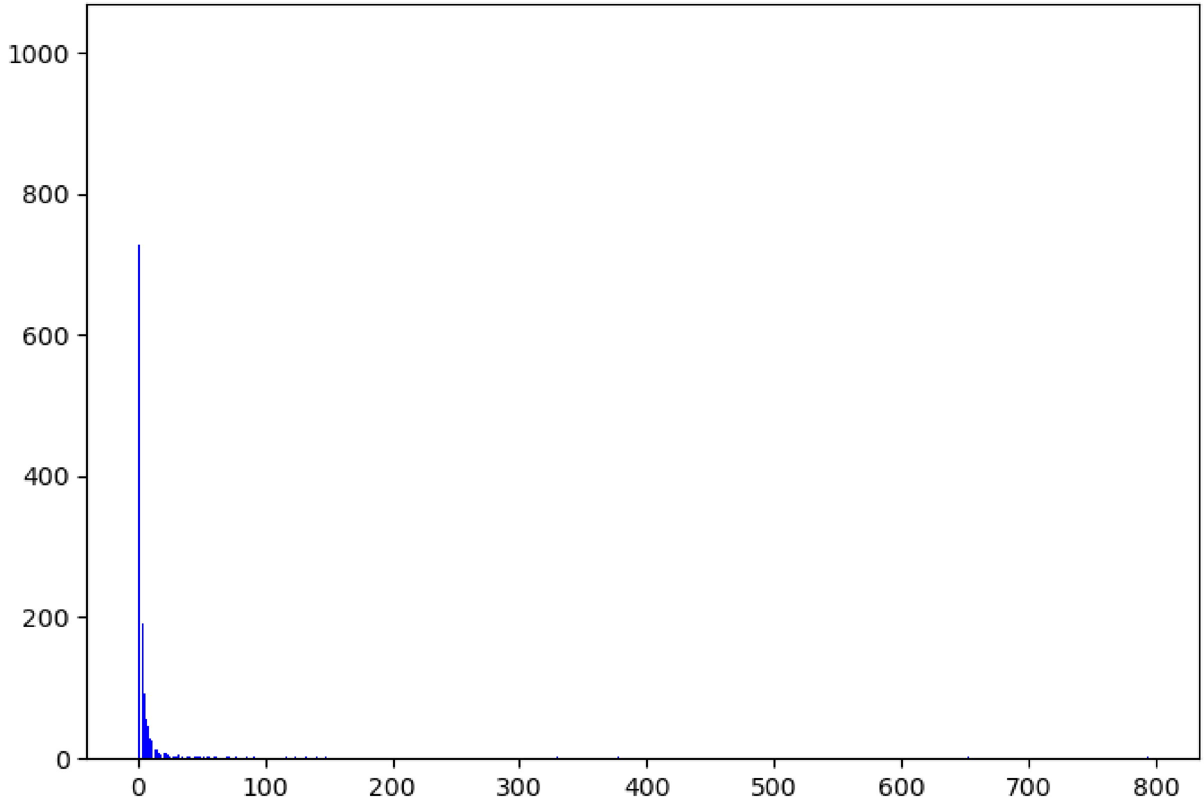
```
def calc_powerlaw(G,kmin=None):
    ddist = degree_distribution(G,normalize=False)
    cdist = cumulative_degree_distribution(G)
    k = arange(len(ddist))

    N = 0
    k_vec = []
    for d in degree_sequence(G):
        if d >= kmin:
            k_vec.append(d)
            N += 1

    # N = cdist[kmin] * G.number_of_nodes()
    # print(N)

    # sum (ki / kmin - 0.5) where ki >= kmin
    inside = 0
    for ki in k_vec:
        if ki >= kmin:
            inside += np.log(ki / (kmin - 0.5))

    alpha = 1 + (N * (1/inside)) # calculate using Newman (8.6)!
    sigma = float(alpha - 1) / float(np.sqrt(N)) # calculate using Newman (8.7)!
    print('HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ fo
    print( '%1.2f +/- %1.2f' % (alpha,sigma) )


    print('HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the
    plt.figure(figsize=(8,12))
    plt.subplot(211)
    plt.bar(k,ddist, width=0.8, bottom=0, color='b') # replace xvalues and barheights

    plt.subplot(212)
    plt.loglog(k,cdist) # replace xvalues and yvalues!
    plt.grid(True)
```

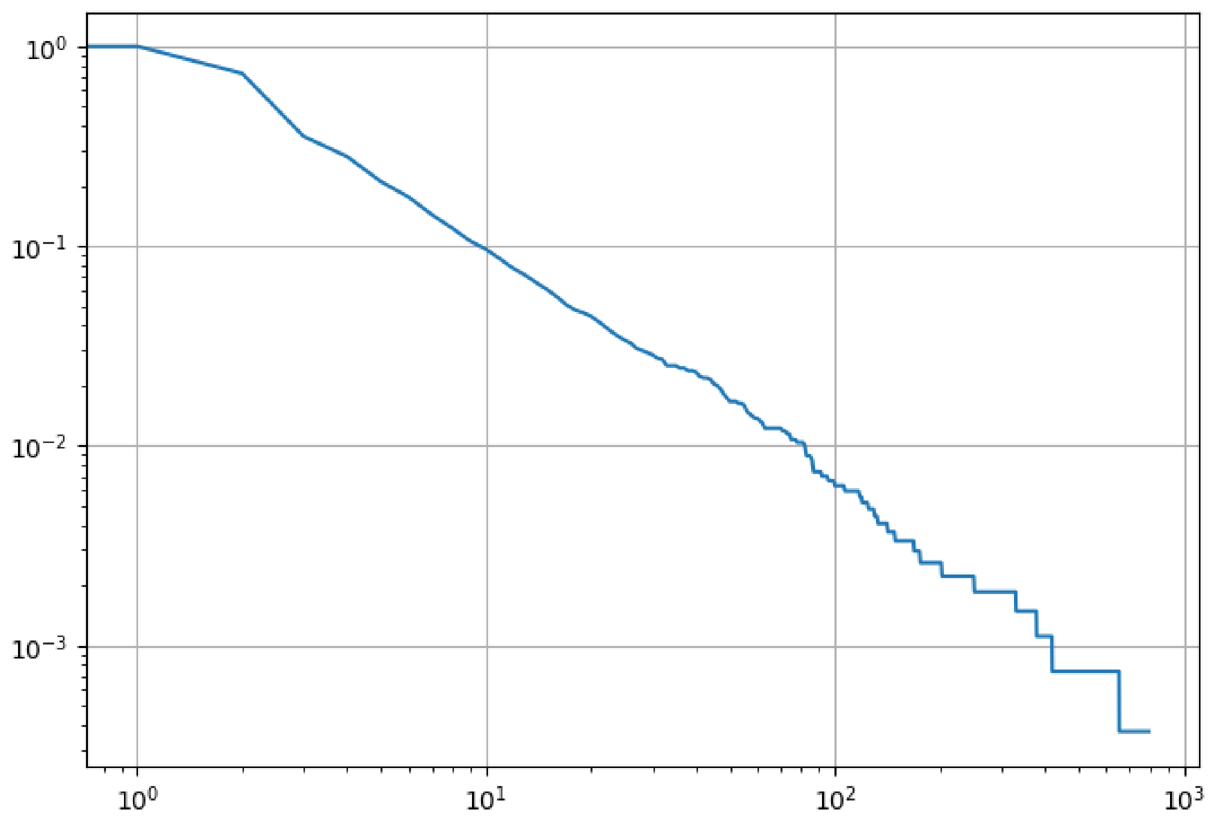
In [78]:

```
G = nx.read_weighted_edgelist('japanese.edgelist',create_using=nx.DiGraph)
print('Japanese Network')
calc_powerlaw(G,10) # select kmin!
plt.show()
G = nx.read_weighted_edgelist('ca-HepTh.edgelist',create_using=nx.Graph)
print('ca-HepTh Network')
calc_powerlaw(G,45) # select kmin!
plt.show()
G = nx.read_weighted_edgelist('soc-Epinions1.edgelist',create_using=nx.DiGraph)
print('soc-Epinions1 Network')
```

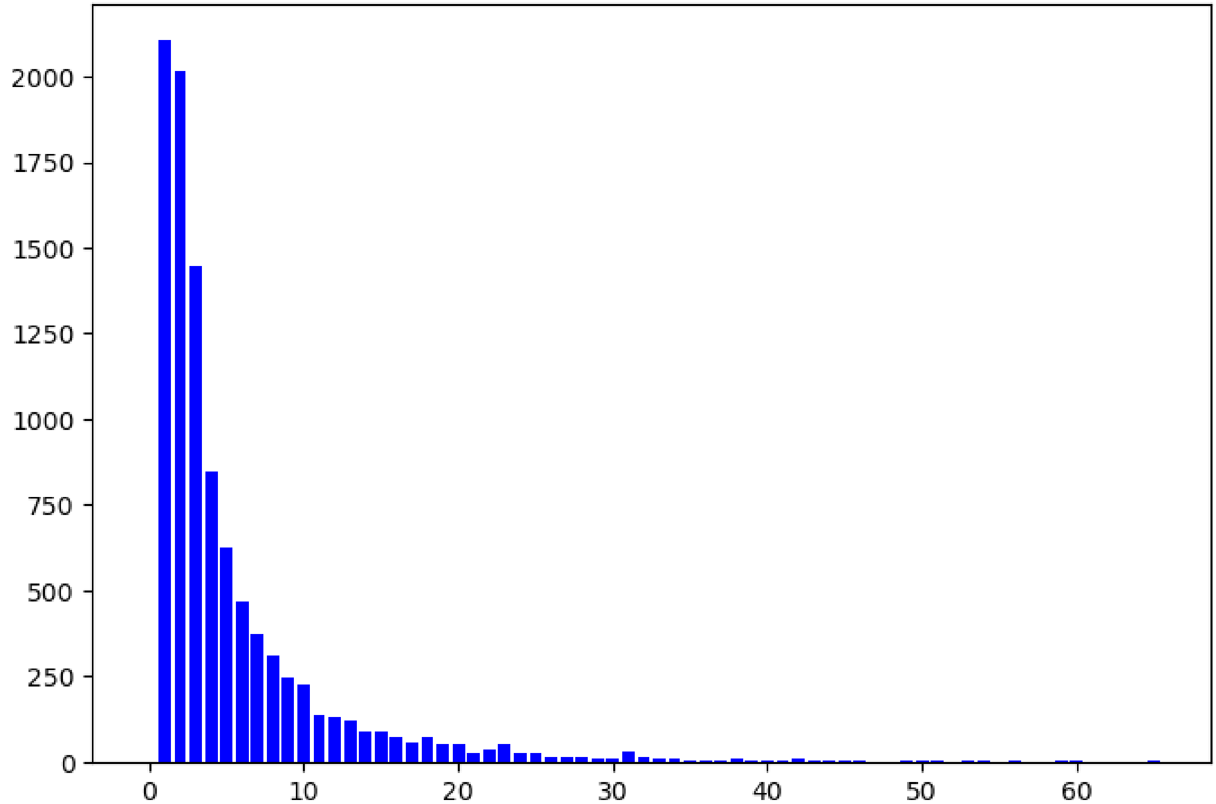
```
calc_powerlaw(G,10) # select kmin!
plt.show()
```

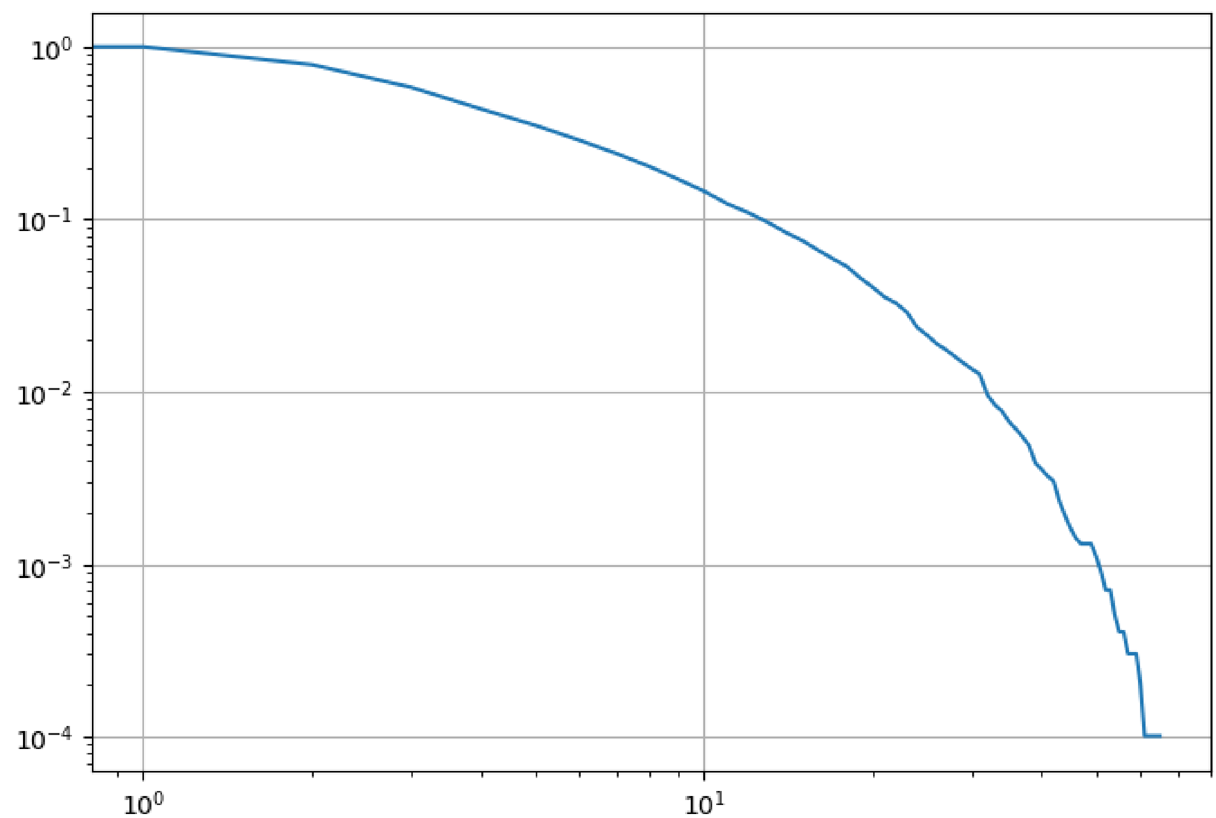
```
Japanese Network
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of the networks.
2.11 +/- 0.07
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree distribution, which will be a bar plot, and one of the cumulative degree distribution, which should
```


1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800

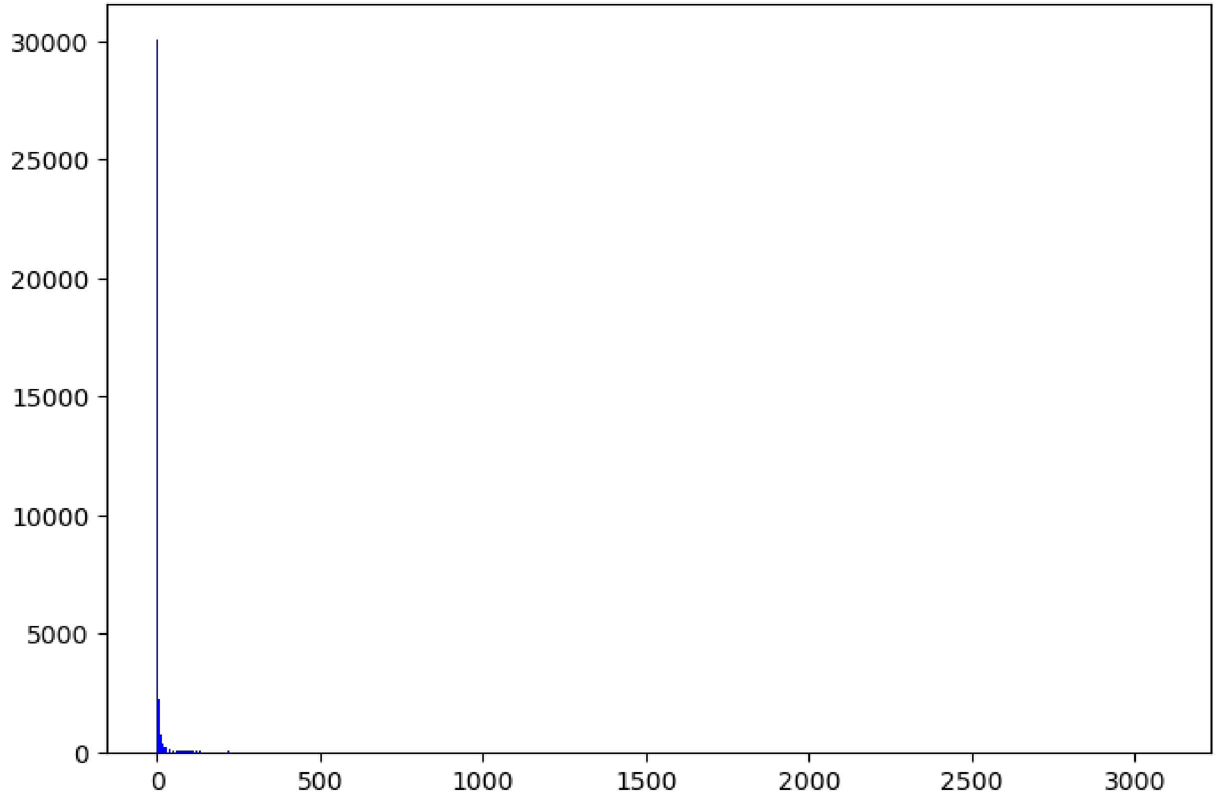

10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3

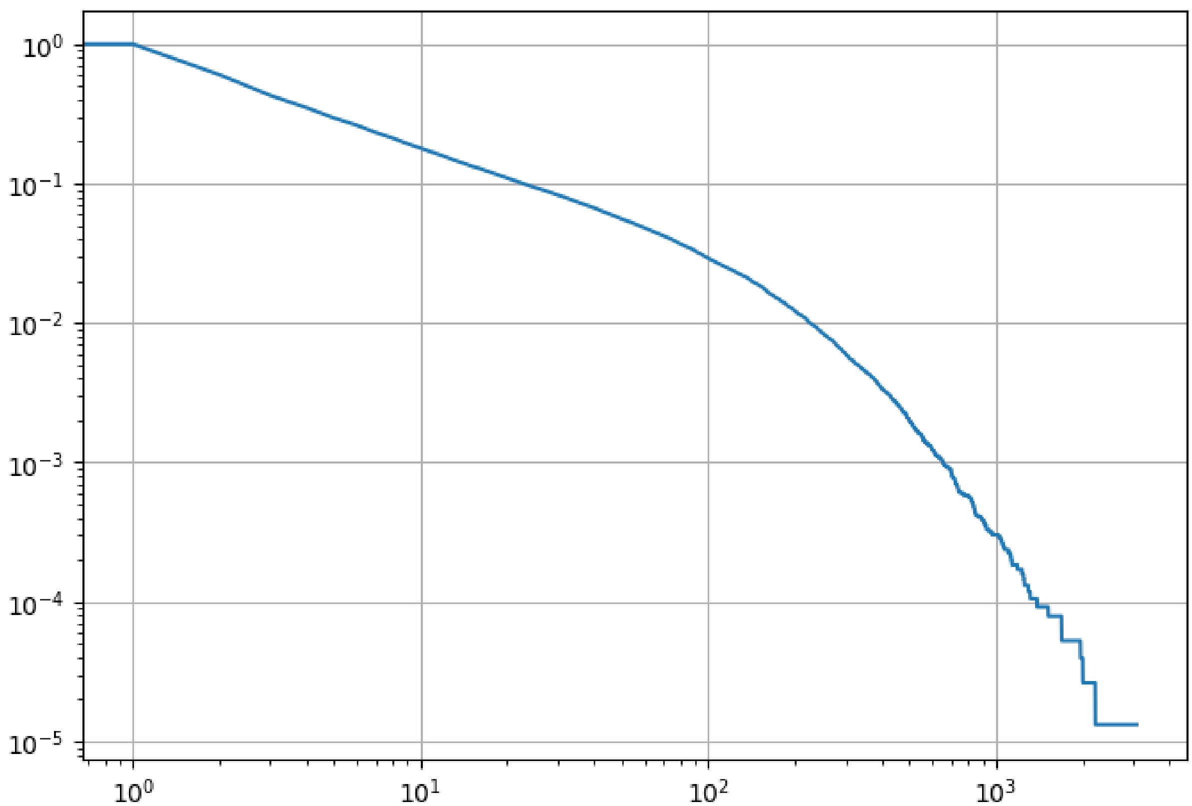
```
ca-HepTh Network
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
7.43 +/- 1.61
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


2000
1750
1500
1250
1000
750
500
250
0
0
10
20
30
40
50
60


10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^0
10^1

```
soc-Epinions1 Network
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
1.79 +/- 0.01
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```



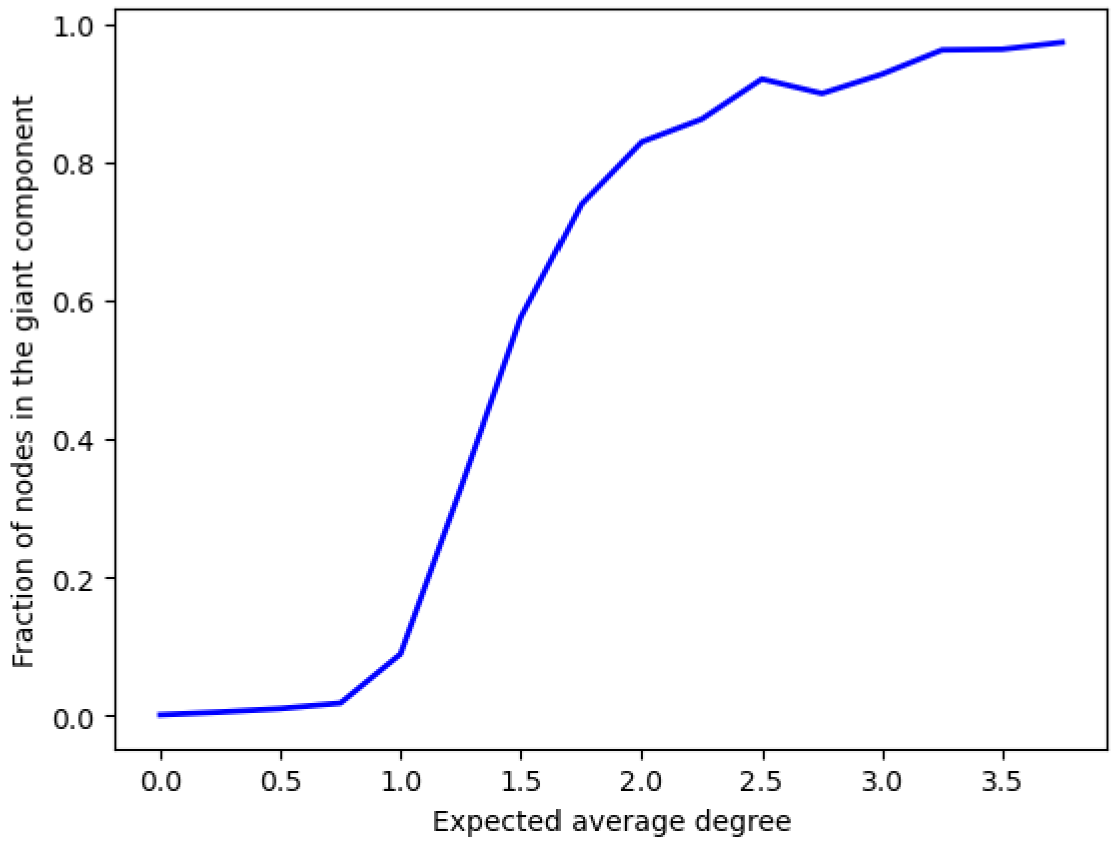


Giant Component

In [150…

```
n = 1000
c = np.arange(0, 4, 0.25)
```

```
S = []
for ci in c:
    p = float(ci) / (float(n - 1))
    ERG = nx.erdos_renyi_graph(n, p)
    giant_component = list(max(nx.connected_components(ERG), key=len))
    S.append(len(giant_component) / float(n))

plt.plot(c, S, 'b', linewidth=2)
plt.xlabel("Expected average degree")
plt.ylabel("Fraction of nodes in the giant component")
plt.show()
```



In [115…

```
print("HIGHLIGHTED QUESTION - Once you have your plot, you'll notice that the line is
print('This is due to the random nature or ER graphs. Edges between nodes are formed w
```

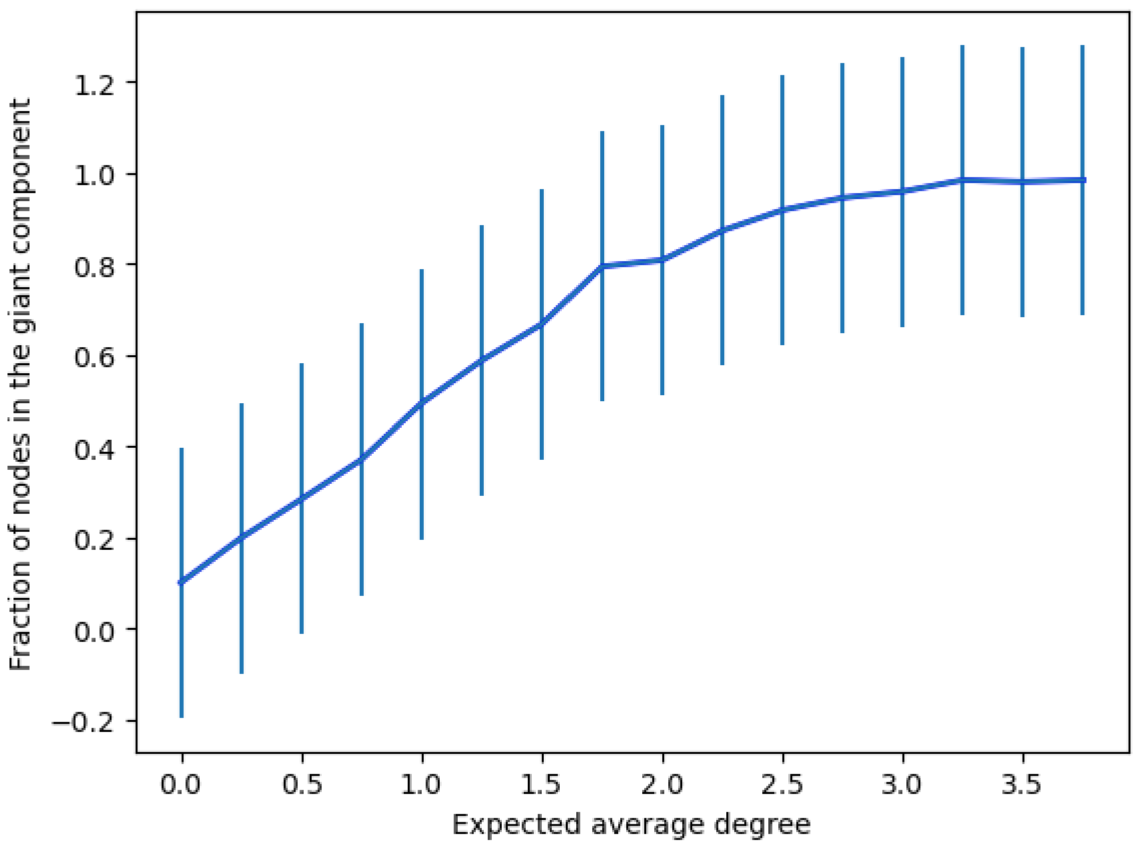
```
HIGHLIGHTED QUESTION - Once you have your plot, you'll notice that the line is pretty
rough and in some cases it may go up and down, even though the theoretical results pr
edicts a curve for the expected giant component size that is monotonic with increasin
g average degree. Why are you seeing this non-monotonic behavior?
This is due to the random nature or ER graphs. Edges between nodes are formed with a
probability p = c / (N - 1), and assuming we keep N = 1000, then as the expected aver
age degree c rises we find that there are more edges being formed and therefore more
nodes in the giant component. However as the network grows there is still a likelihoo
d of forming components that are not connected to the giant component. If these compo
nents are large in size (but still smaller than the giant component), then the fluctu
ation will be greatly noticeable by a dip in the graph. This can cause the nonmonoton
ic behavior that we see in the graph. As we increase the size of N then the graph wil
l be closer to the theoretical result.
```

In [151…

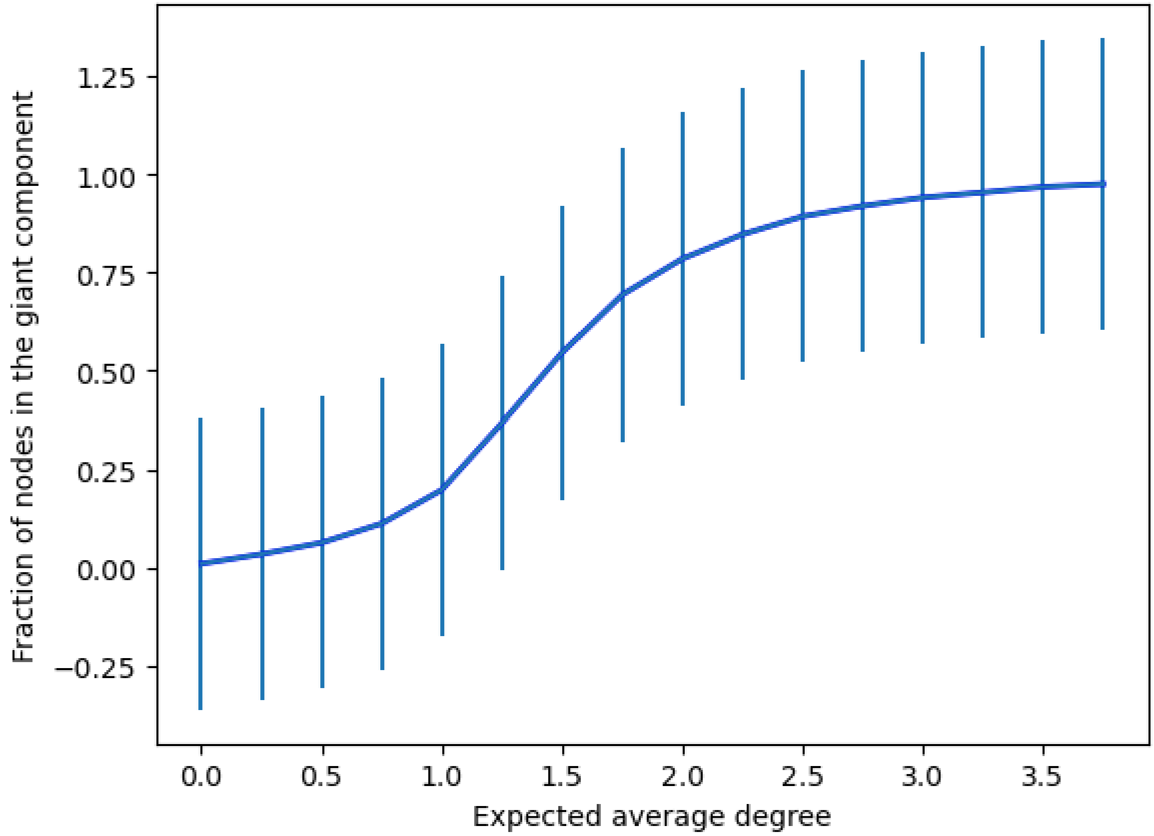
```
def plot_giant_component(n, num_graphs):
    c = np.arange(0, 4, 0.25)
    S_mean = []
```

```
    num_graphs = 100
    p = 0.5

    for ci in c:
        S = []
        for i in range(num_graphs):
            p = float(ci) / (float(n - 1))
            ERG = nx.erdos_renyi_graph(n, p)
            giant_component = list(max(nx.connected_components(ERG), key=len))
            S.append(len(giant_component) / float(n))

        S_mean.append(np.mean(S))
    S_std = np.std(S_mean)
    plt.errorbar(c, S_mean, yerr=S_std)
    plt.plot(c, S_mean, 'b', linewidth=2)
    plt.xlabel("Expected average degree")
    plt.ylabel("Fraction of nodes in the giant component")
    plt.show()
print('HIGHLIGHTED QUESTION -  Make three plots, one for n=10, one for n=100, and one
print('n = 10')
plot_giant_component(n=10, num_graphs=100)
print('n = 100')
plot_giant_component(n=100, num_graphs=100)
# print('n = 1000')
# plot_giant_component(n=1000, num_graphs=100)
```

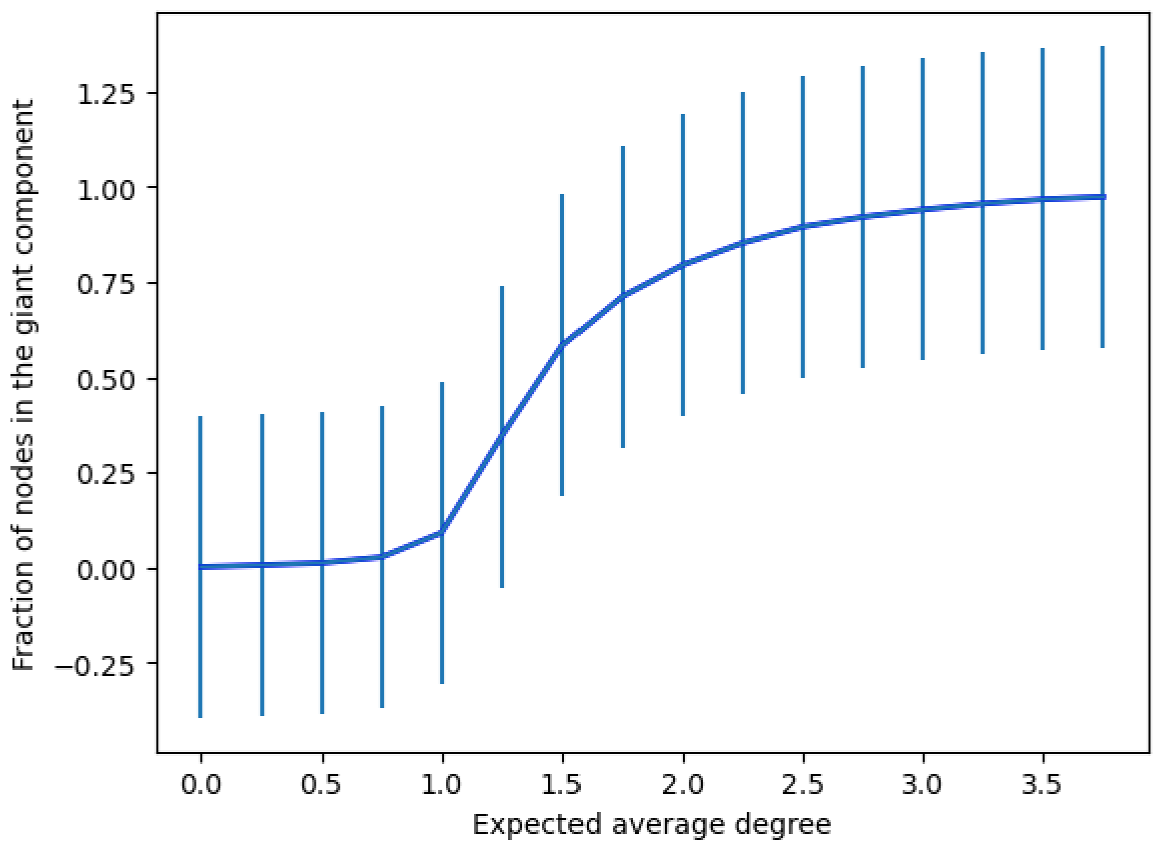
```
HIGHLIGHTED QUESTION -  Make three plots, one for n=10, one for n=100, and one for n=
1000.
n = 10
```




```
n = 100
```



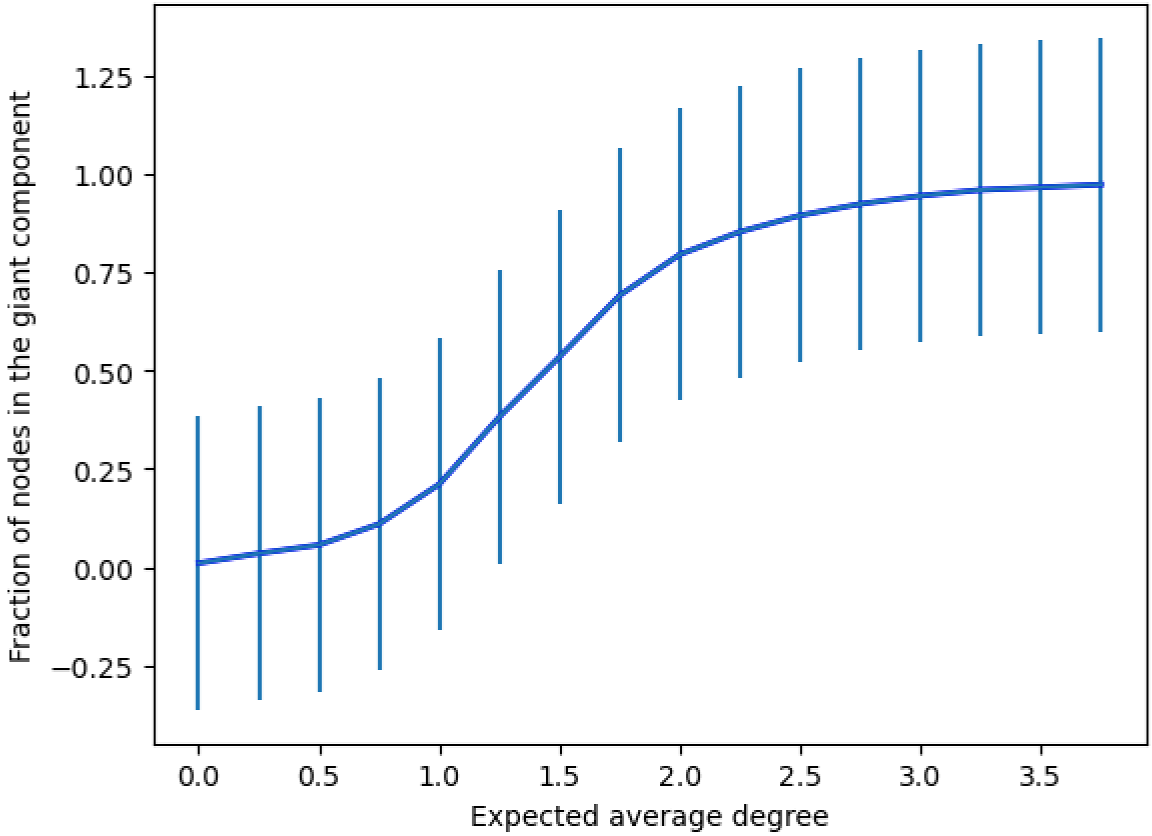

```
n = 1000
```



In [161…

```
print('n = 100 but averaging over 1 graph')
plot_giant_component(n=100, num_graphs=1)
```

```
n = 100 but averaging over 1 graph
```



In [138…

```
print('HIGHLIGHTED QUESTION - Observe the differences in the plots and explain how thi
print('As n increases we see that the fluctuations in the giant component of the ER gr
```

```
HIGHLIGHTED QUESTION - Observe the differences in the plots and explain how this supp
orts the notion that the statistics of the model converge to their expected values as
n gets larger.
As n increases we see that the fluctuations in the giant component of the ER graph de
crease significantly. For n = 10 and averaging over 100 graph iterations the graph ex
hibits some non-monotonic behavior and the curve is not smooth. For n=1000 and averag
ing over 100 graph iterations the graph is closer to the monotonic behavior that we e
xpect and also exhibits a smoother curve. This tells us that as the graph size increa
ses, the statistics of the ER model give us a better approximation of a random graph
for a given average degree.
```

Degree Distributions of Random Network Models

In [164…

```
N = 5000
```

In [220…

```
print('HIGHLIGHTED QUESTION - Use this same code to look at the degree distributions a
```

```
HIGHLIGHTED QUESTION - Use this same code to look at the degree distributions and cum
ulative degree distributions of these networks. Do this for small, intermediate, and
large values of the various parameters that determine these networks
```

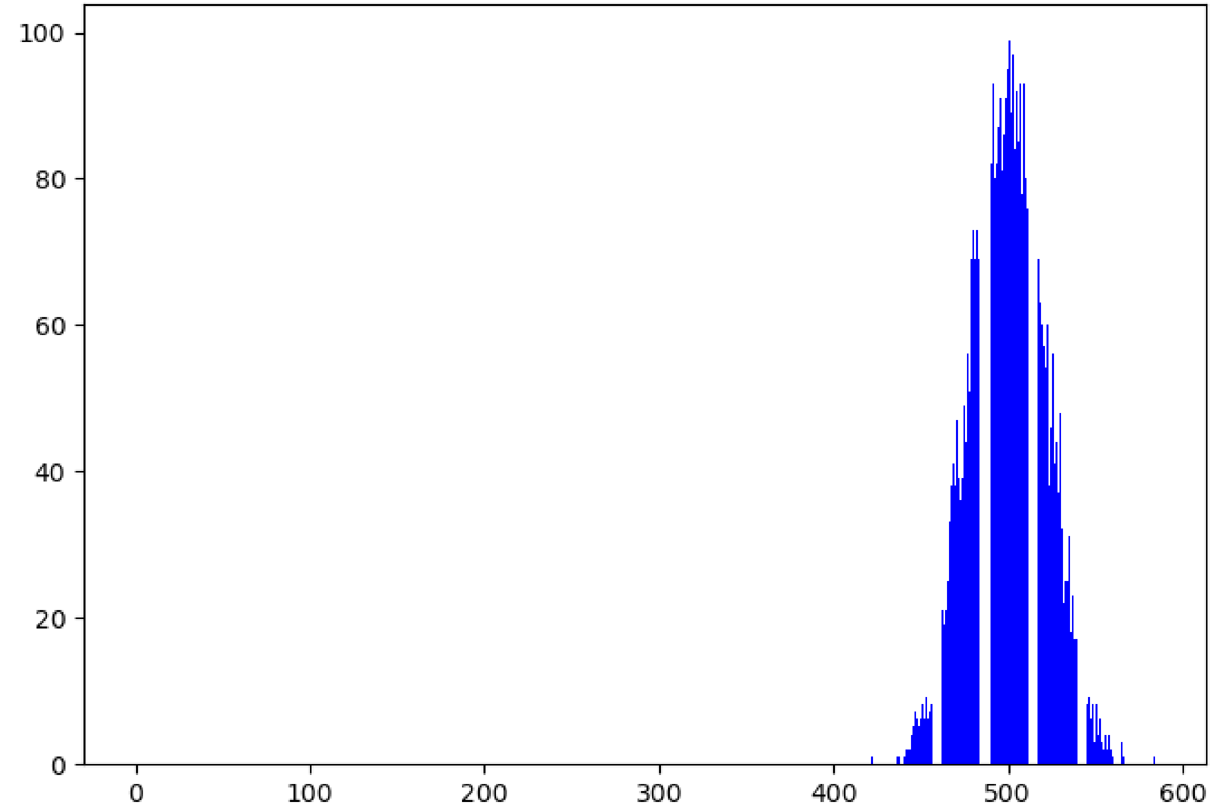
Erdos-Renyi

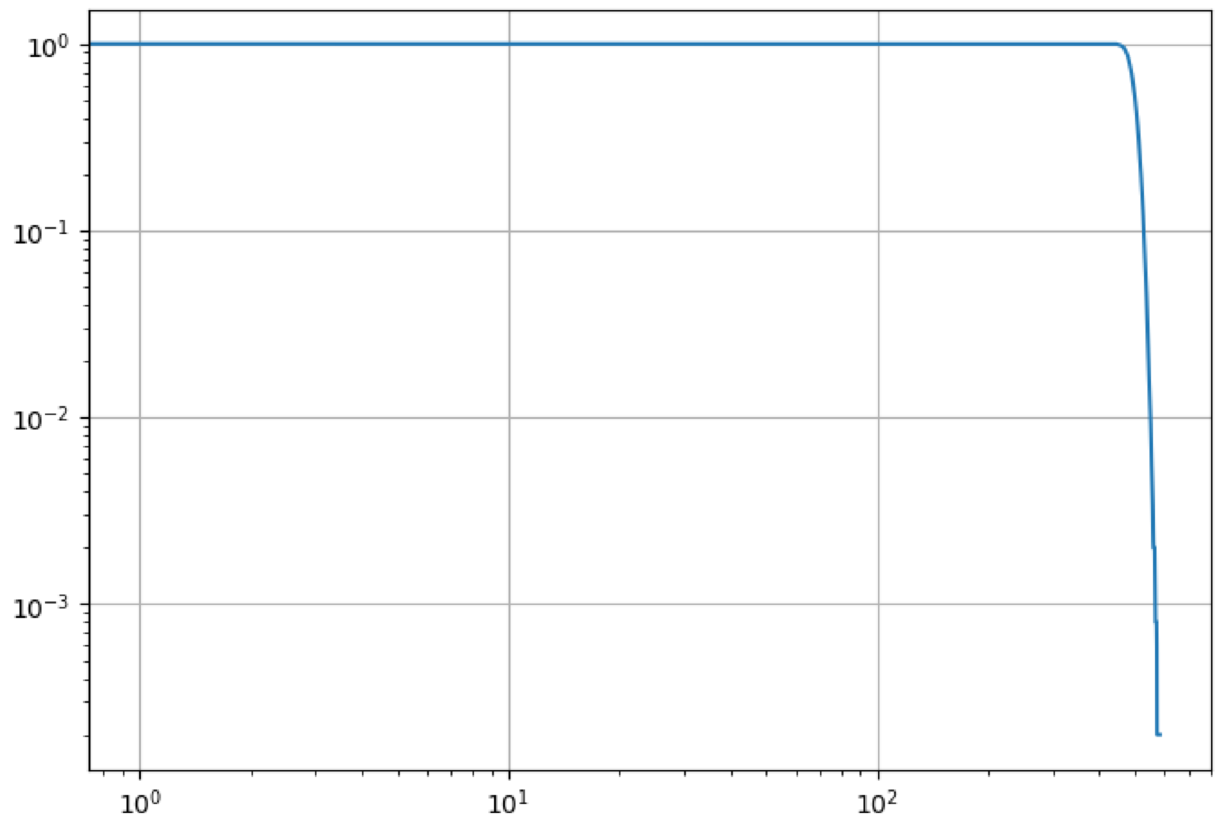
In [170…

```
p = [0.1, 0.2, 0.4]
kmin = [500, 1000, 2000]
for pi, kmini in zip(p, kmin):
    print(f'ERG with N = {N} and p = {pi} and kmin = {kmini}')
```

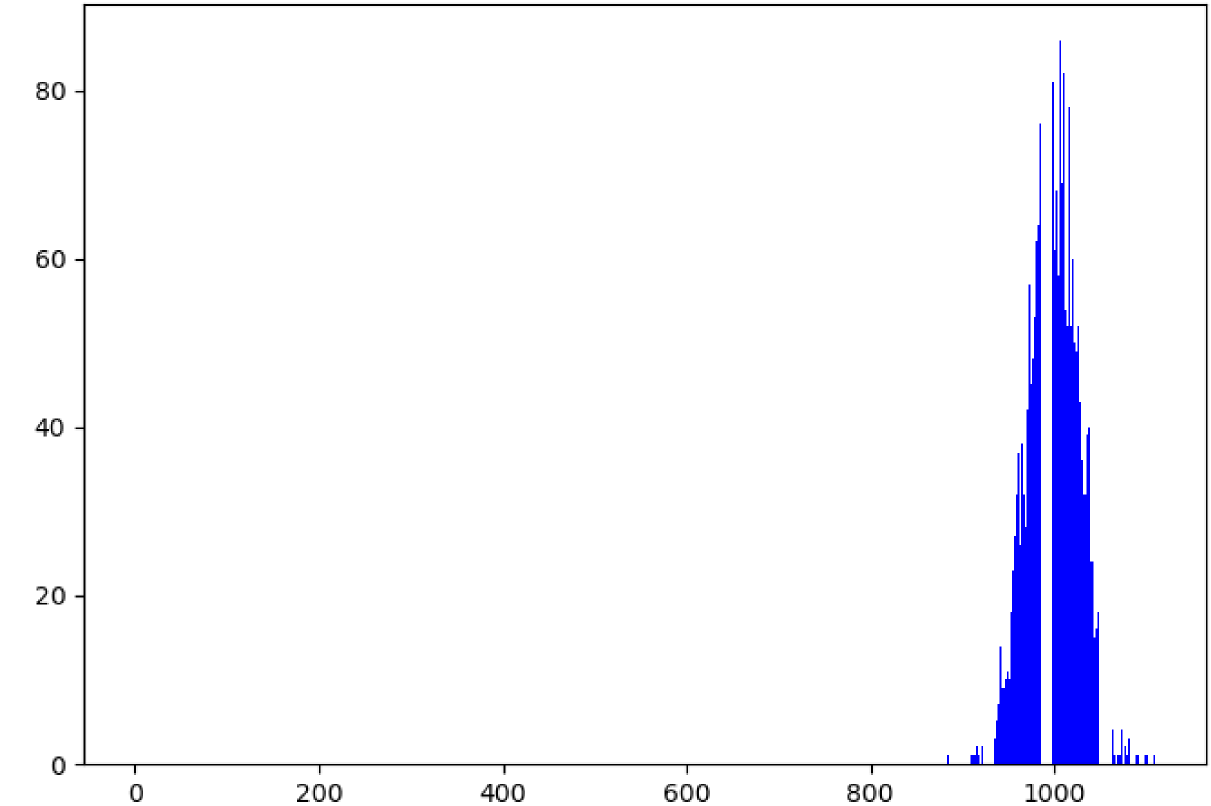
```
    ERG = nx.erdos_renyi_graph(N, pi)
    calc_powerlaw(ERG, kmini)
    plt.show()
```

```
ERG with N = 5000 and p = 0.1 and kmin = 500
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of the networks.
30.57 +/- 0.59
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree distribution, which will be a bar plot, and one of the cumulative degree distribution, which should
```

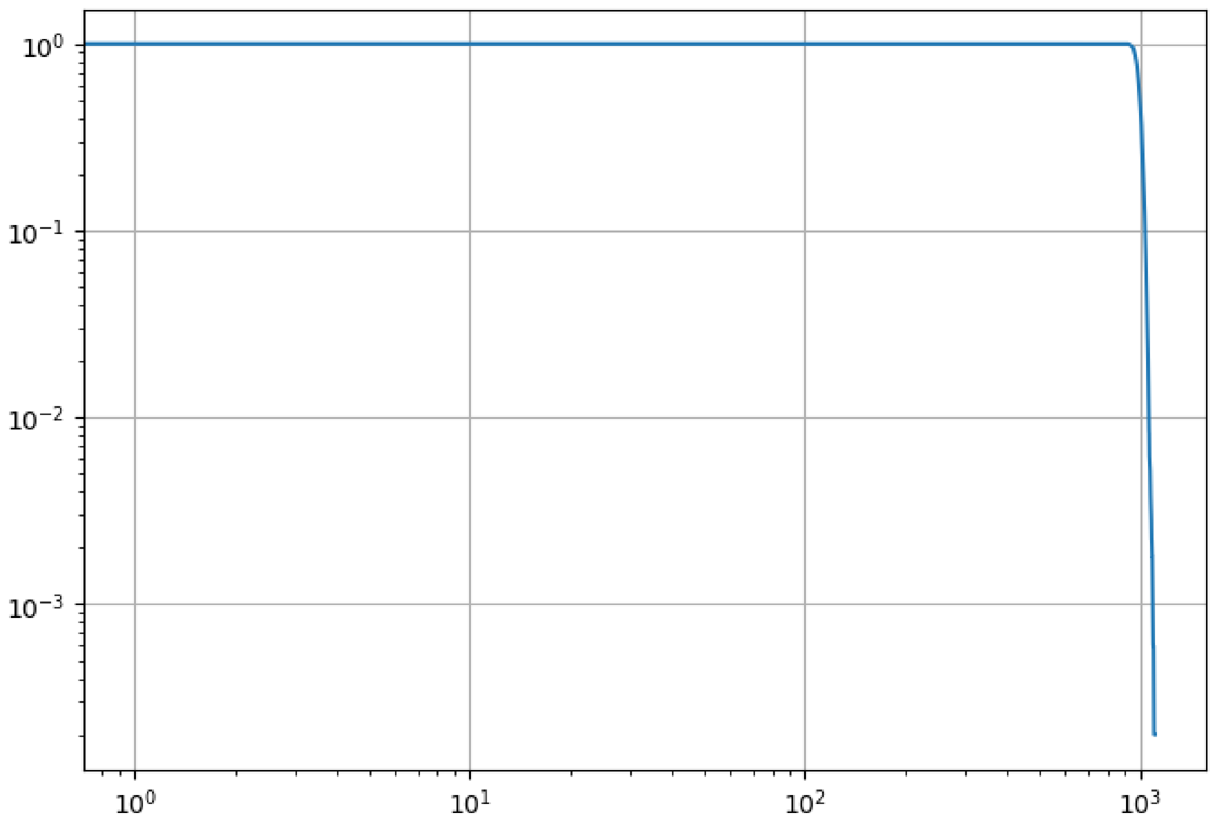

100
80
60
40
20
0
0
100
200
300
400
500
600


10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

```
ERG with N = 5000 and p = 0.2 and kmin = 1000
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
45.60 +/- 0.89
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```

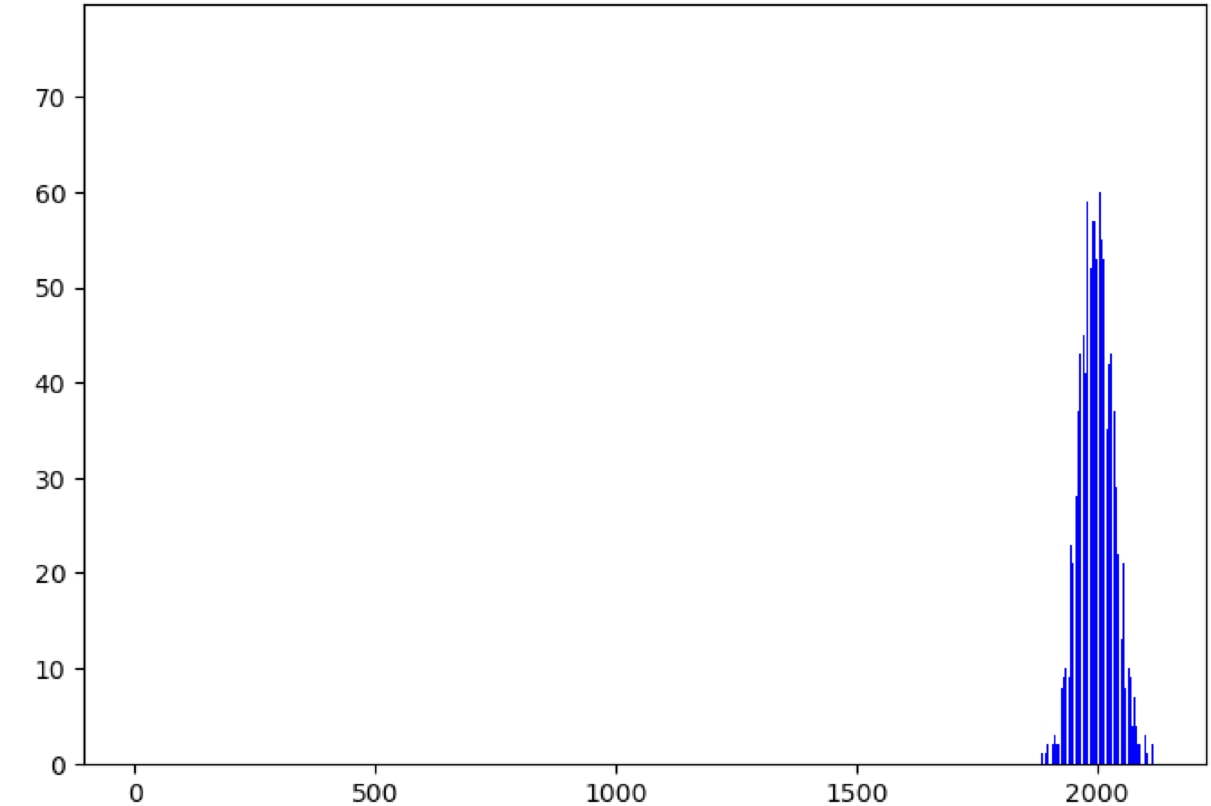


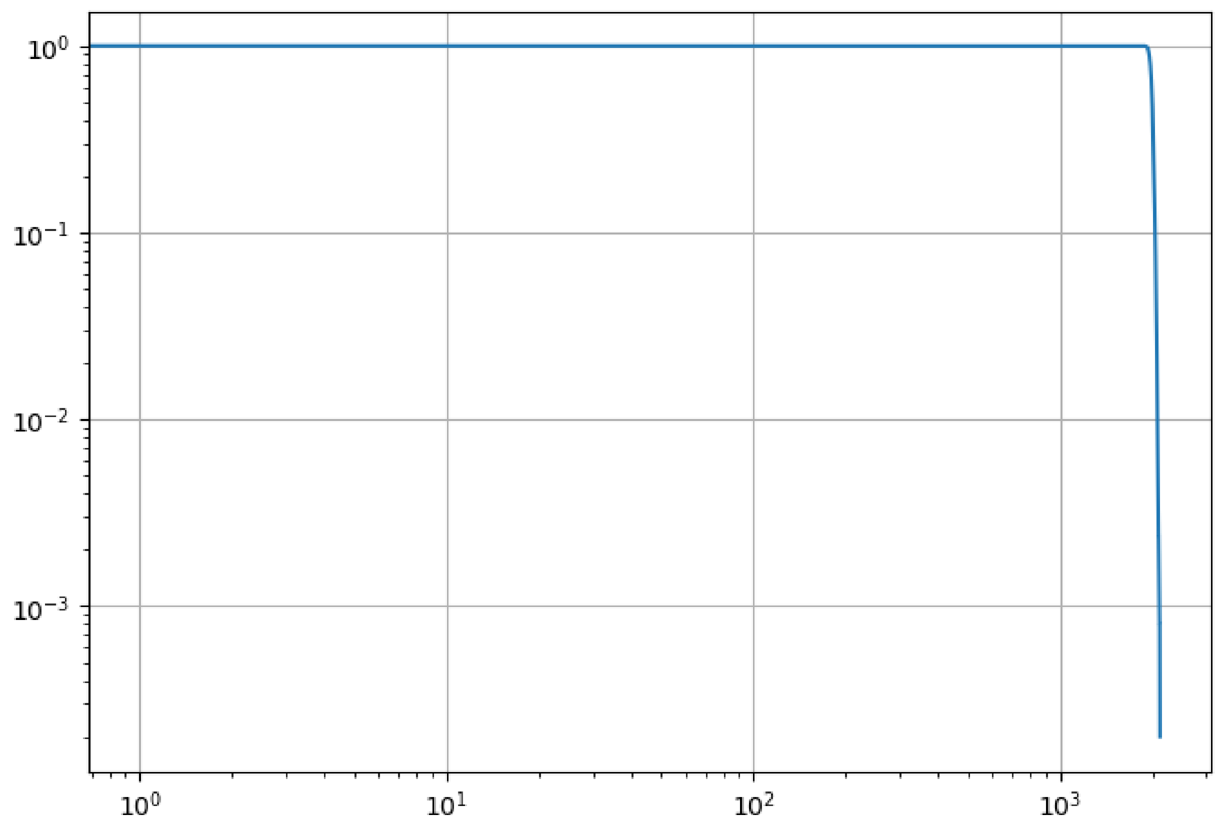
0
20
40
60
80
0
200
400
600
800
1000



10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3

```
ERG with N = 5000 and p = 0.4 and kmin = 2000
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of the networks.
75.66 +/- 1.52
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree distribution, which will be a bar plot, and one of the cumulative degree distribution, which should
```


70
60
50
40
30
20
10
0
0
500
1000
1500
2000

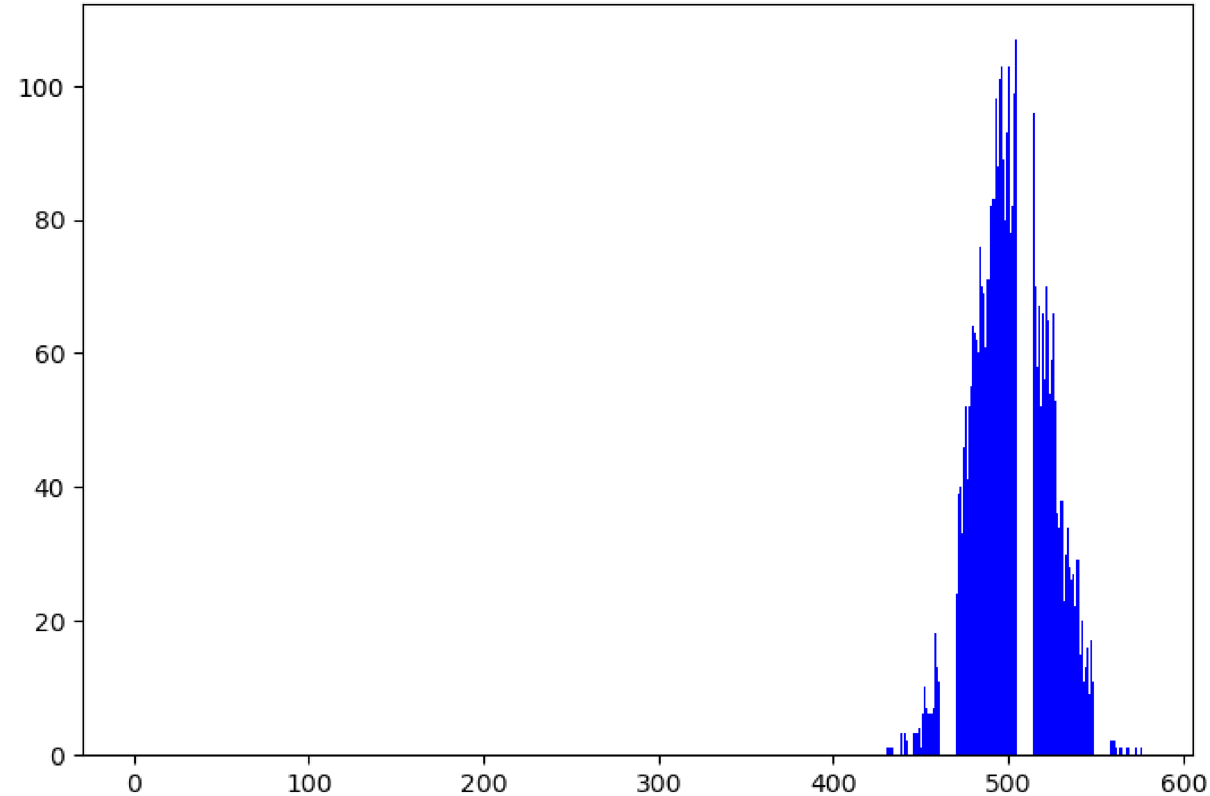

10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3

Small-World

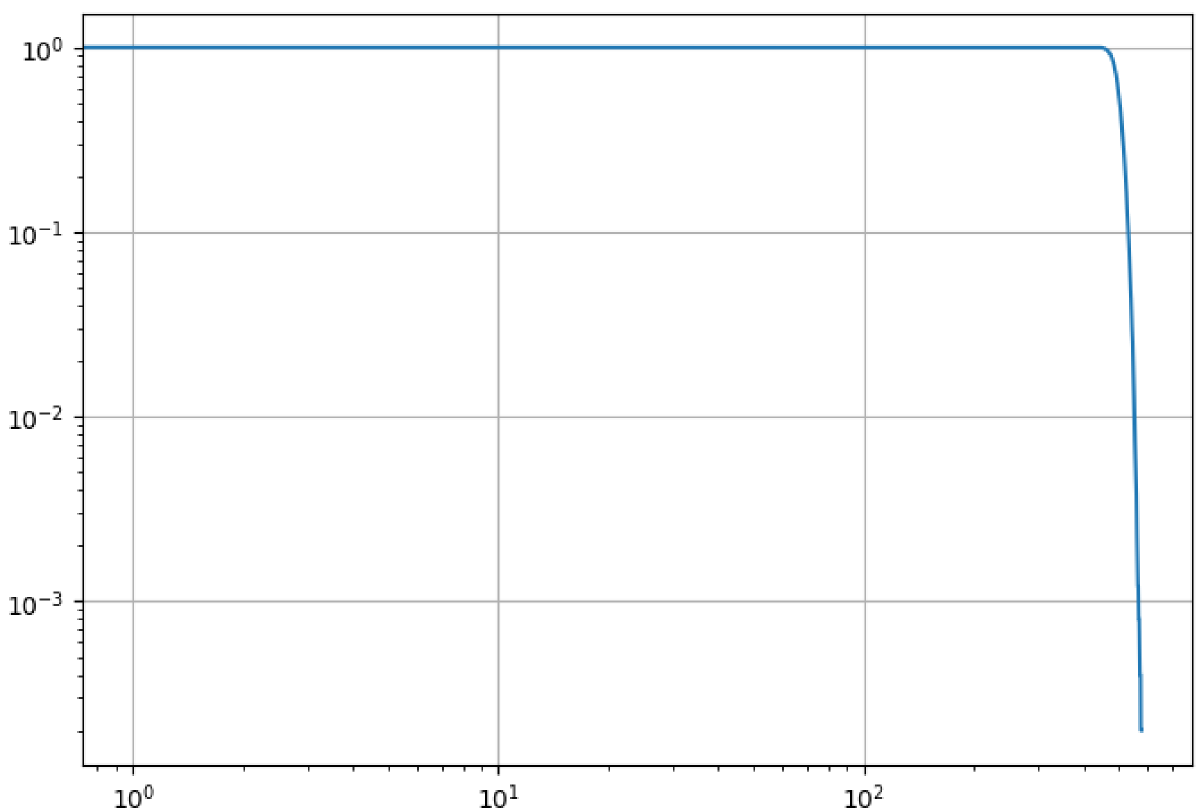
In [173…

```
p = [0.1, 0.2, 0.4]
q = [2, 4, 10]
kmin = [500, 1000, 2000]
for pi, qi, kmini in zip(p, q, kmin):
    print(f'SW with N = {N} and p = {pi} | q = {qi} | kmin = {kmini}')
    SW = randomnet.small_world_graph(N,qi,pi)
    calc_powerlaw(SW, kmini)
    plt.show()
```

```
SW with N = 5000 and p = 0.1 | q = 2 | kmin = 500
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
29.16 +/- 0.54
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```

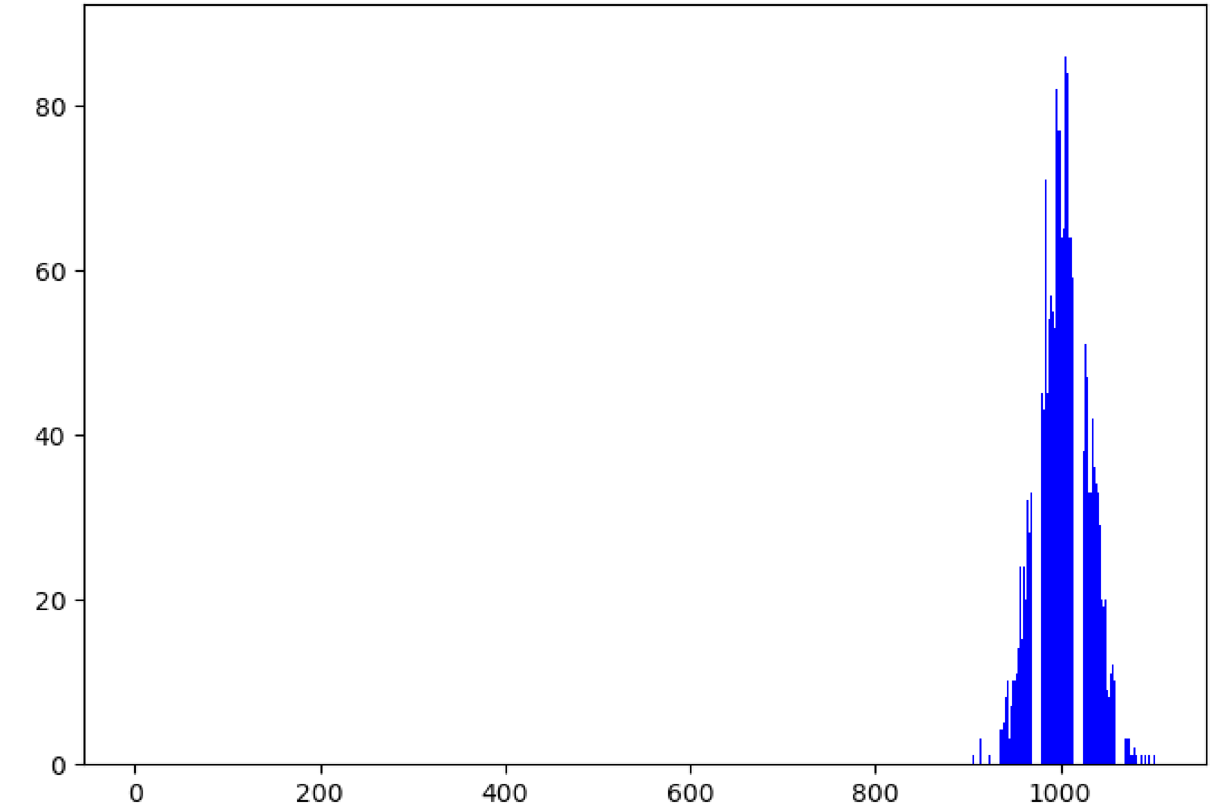


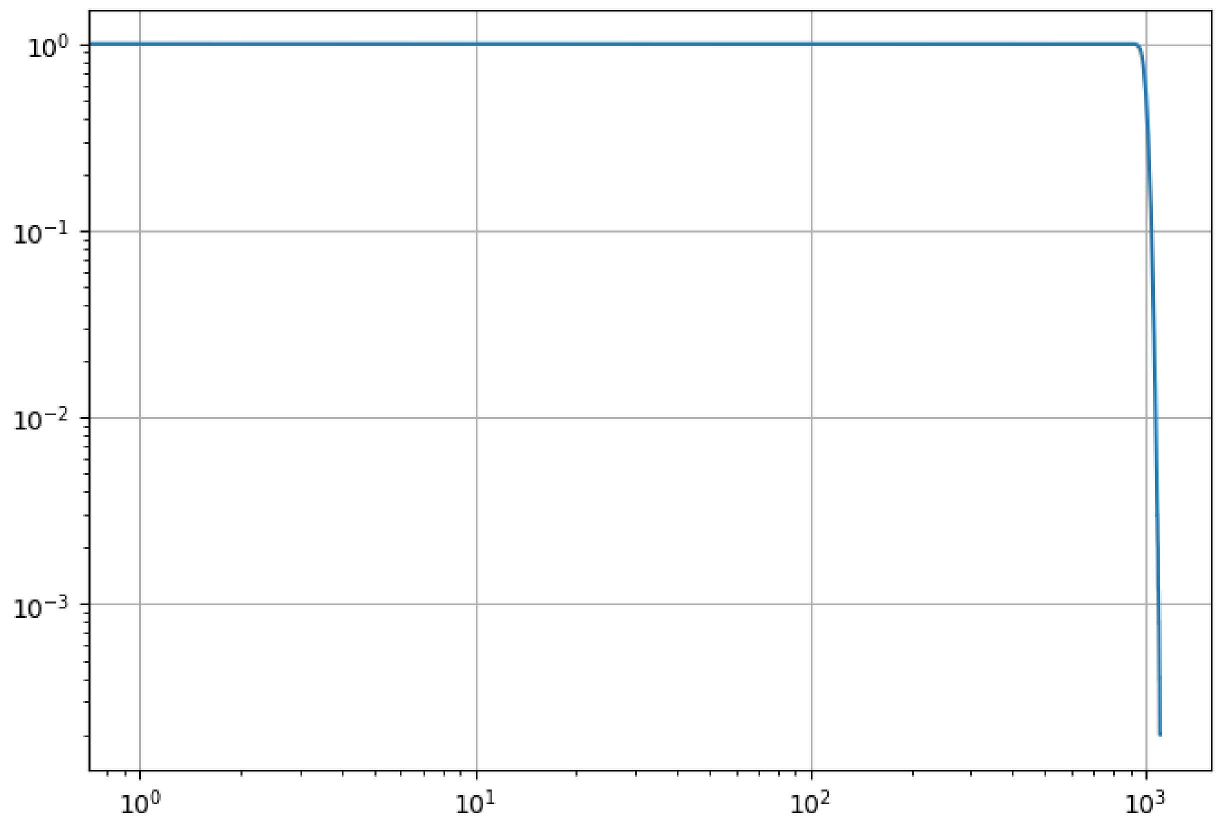
100
80
60
40
20
0
0
100
200
300
400
500
600



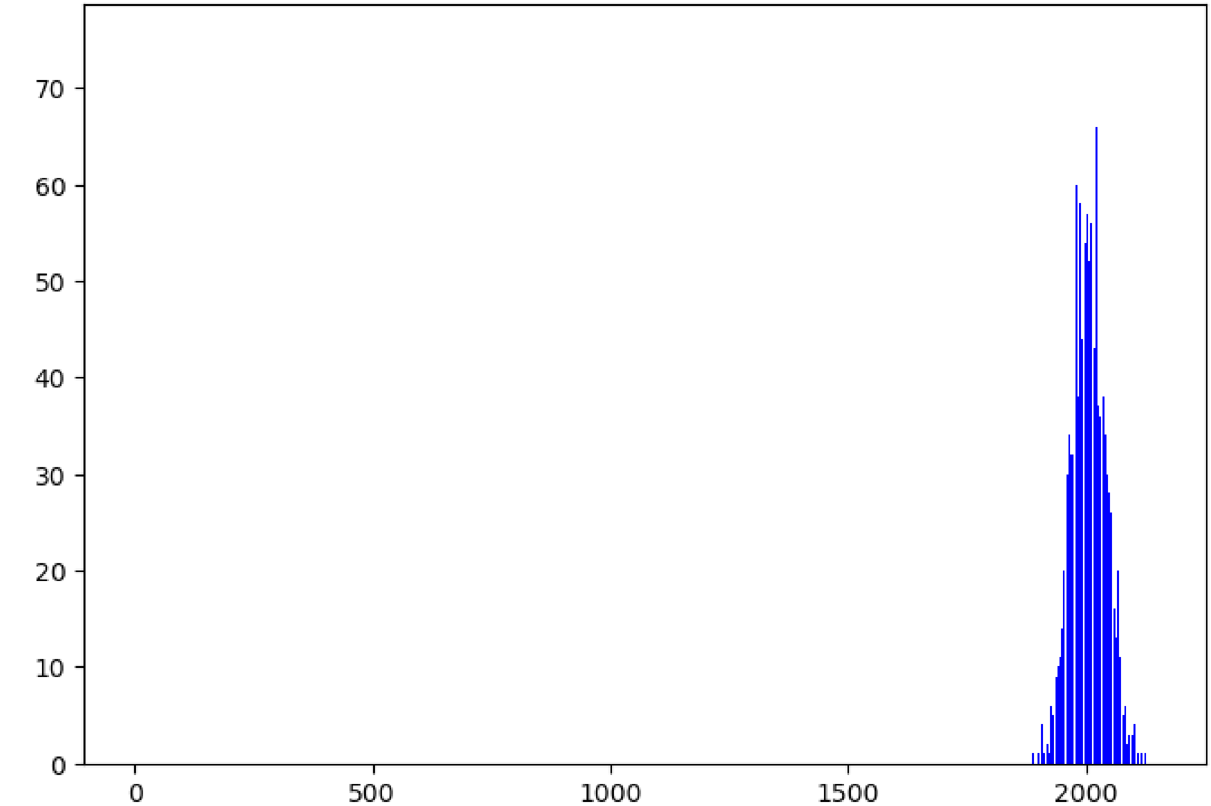
10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

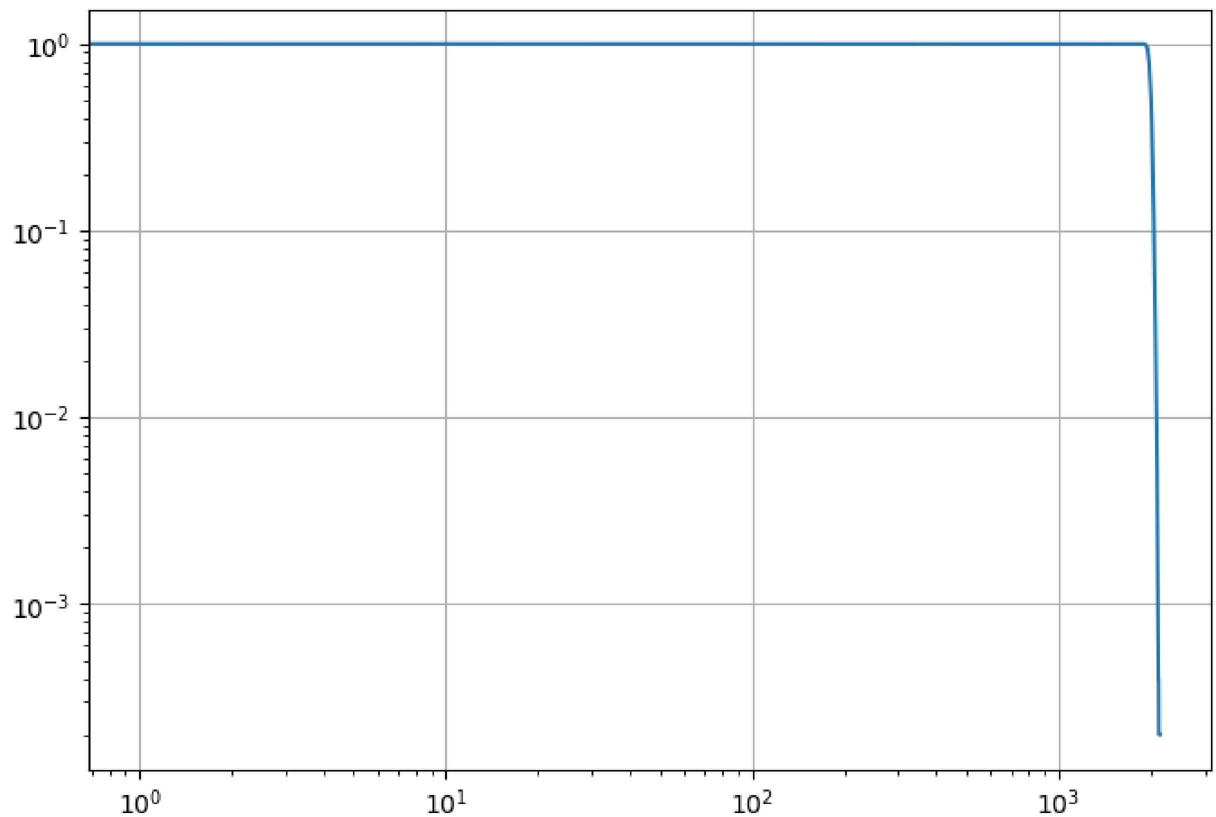
```
SW with N = 5000 and p = 0.2 | q = 4 | kmin = 1000
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
44.11 +/- 0.82
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


80
60
40
20
0
0
200
400
600
800
1000


10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3

```
SW with N = 5000 and p = 0.4 | q = 10 | kmin = 2000
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
69.62 +/- 1.29
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```



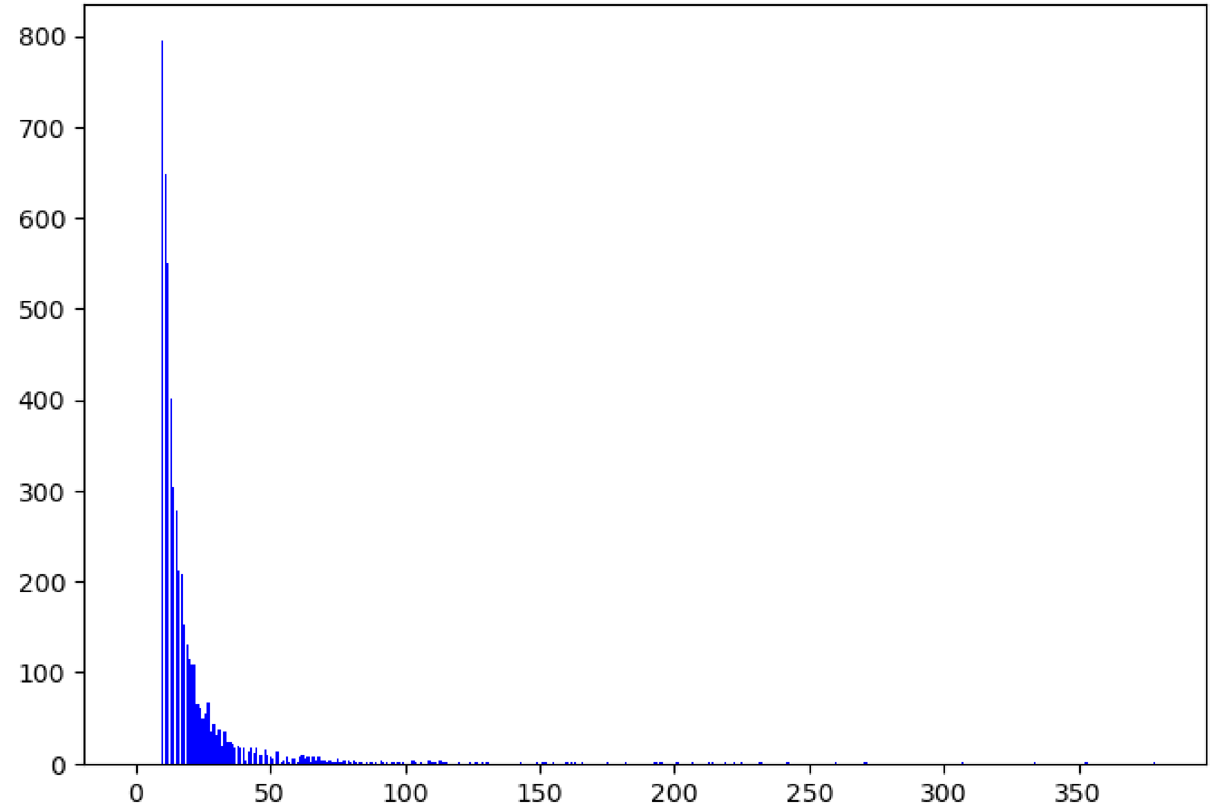


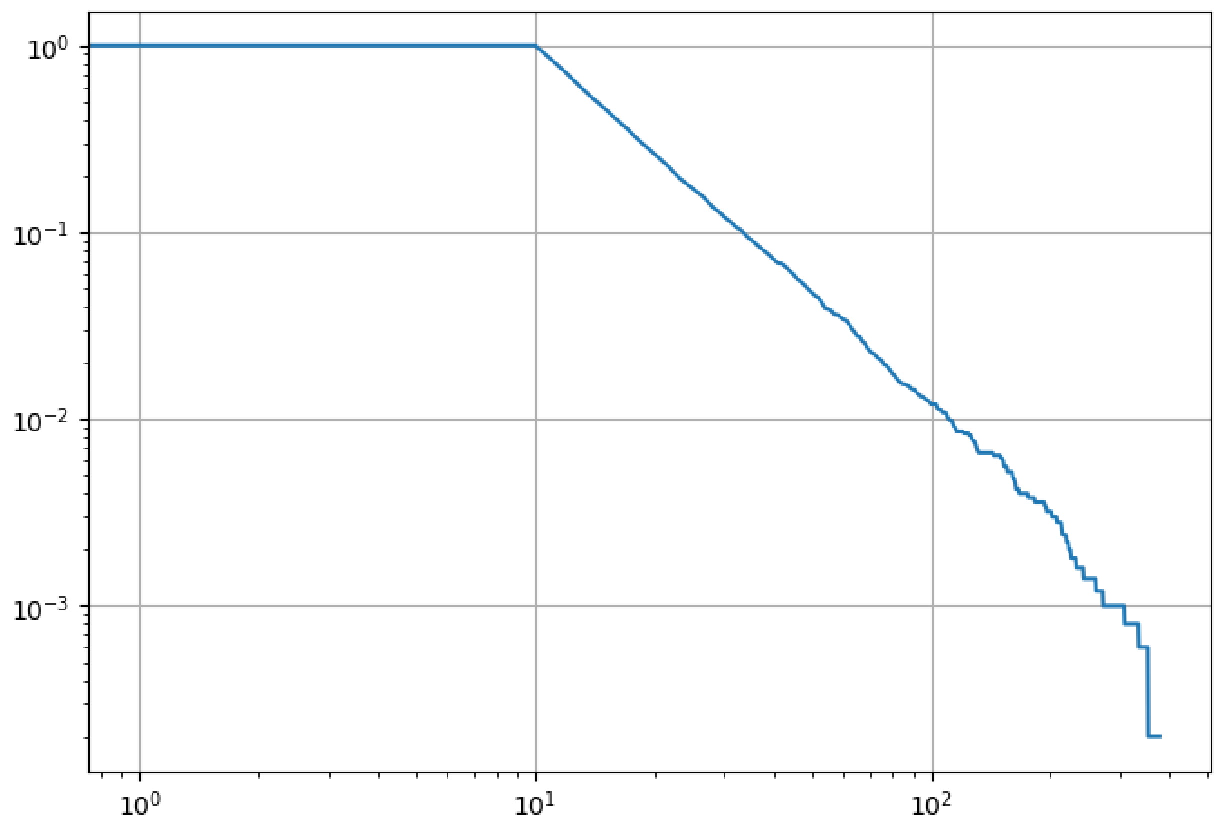
Barabasi-Albert

In [208…

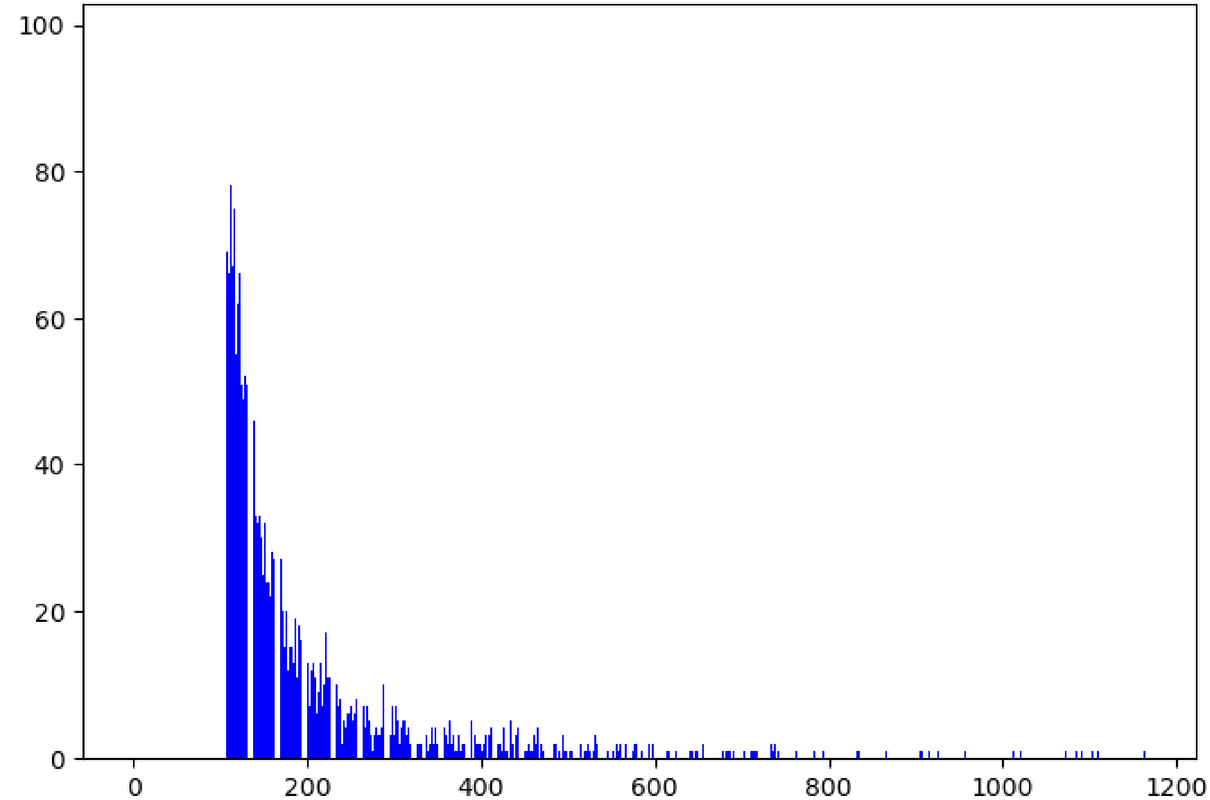
```
m = [10, 100, 1000]
kmin = [40, 200, 1000]
for mi, kmini in zip(m, kmin):
    print(f'BA with N = {N} | k = {kmini} | m = {mi}')
    BAG = nx.barabasi_albert_graph(N,mi)
    calc_powerlaw(BAG, kmini)
    plt.show()
```

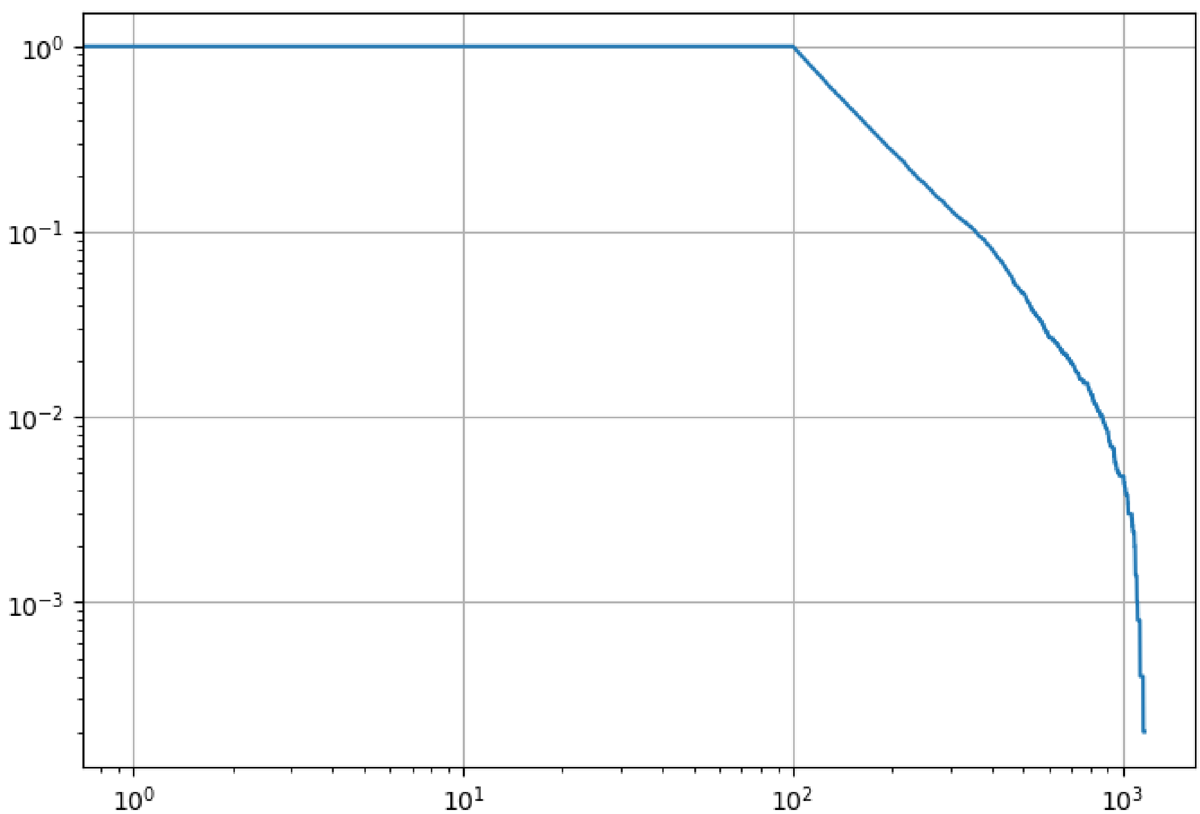
```
BA with N = 5000 | k = 40 | m = 10
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.95 +/- 0.10
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


800
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300
350

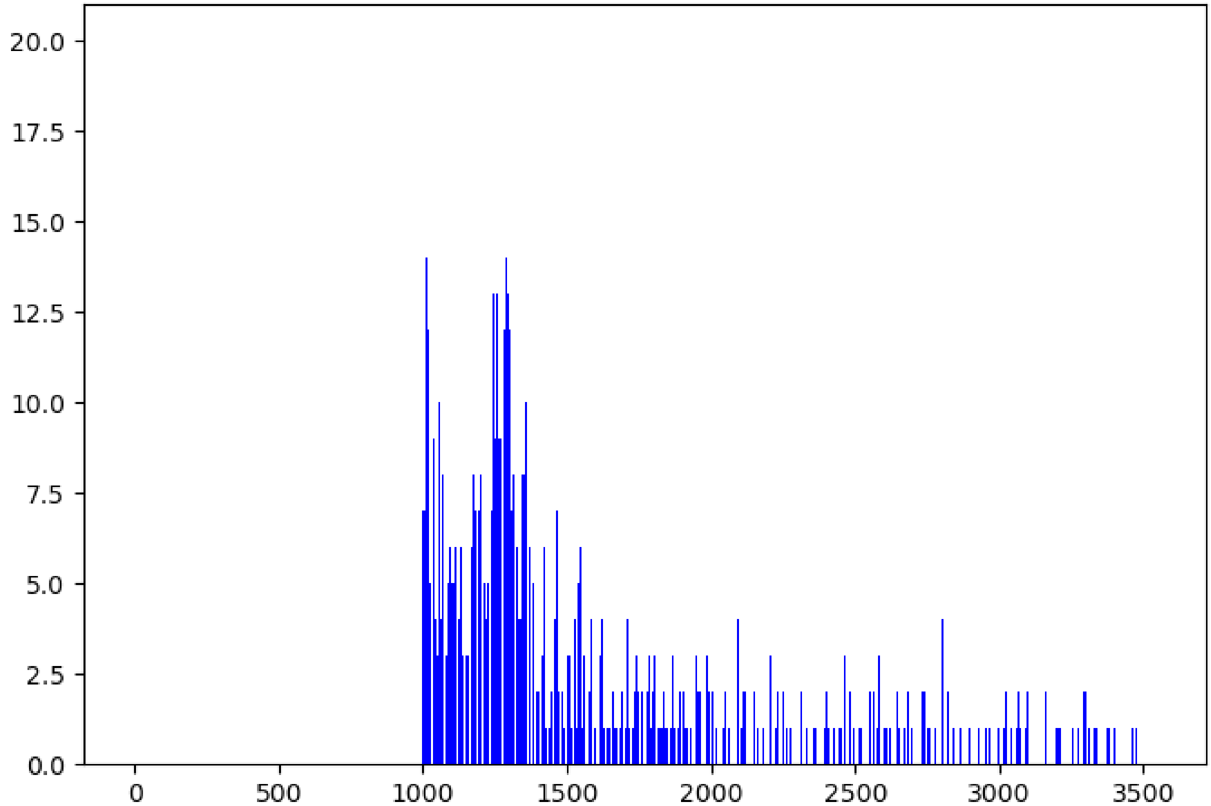

10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

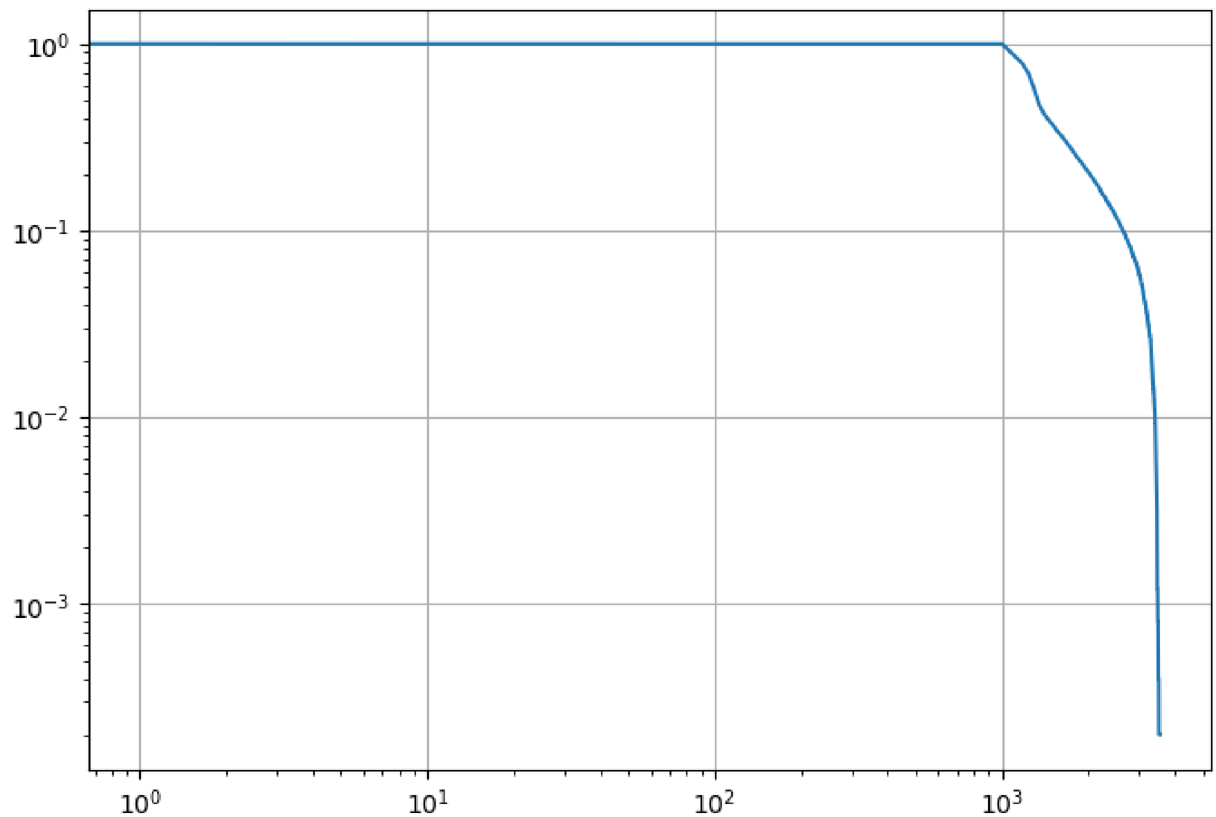
```
BA with N = 5000 | k = 200 | m = 100
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.98 +/- 0.05
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


0
20
40
60
80
100
0
200
400
600
800
1000
1200


10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3

```
BA with N = 5000 | k = 1000 | m = 1000
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
3.43 +/- 0.03
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





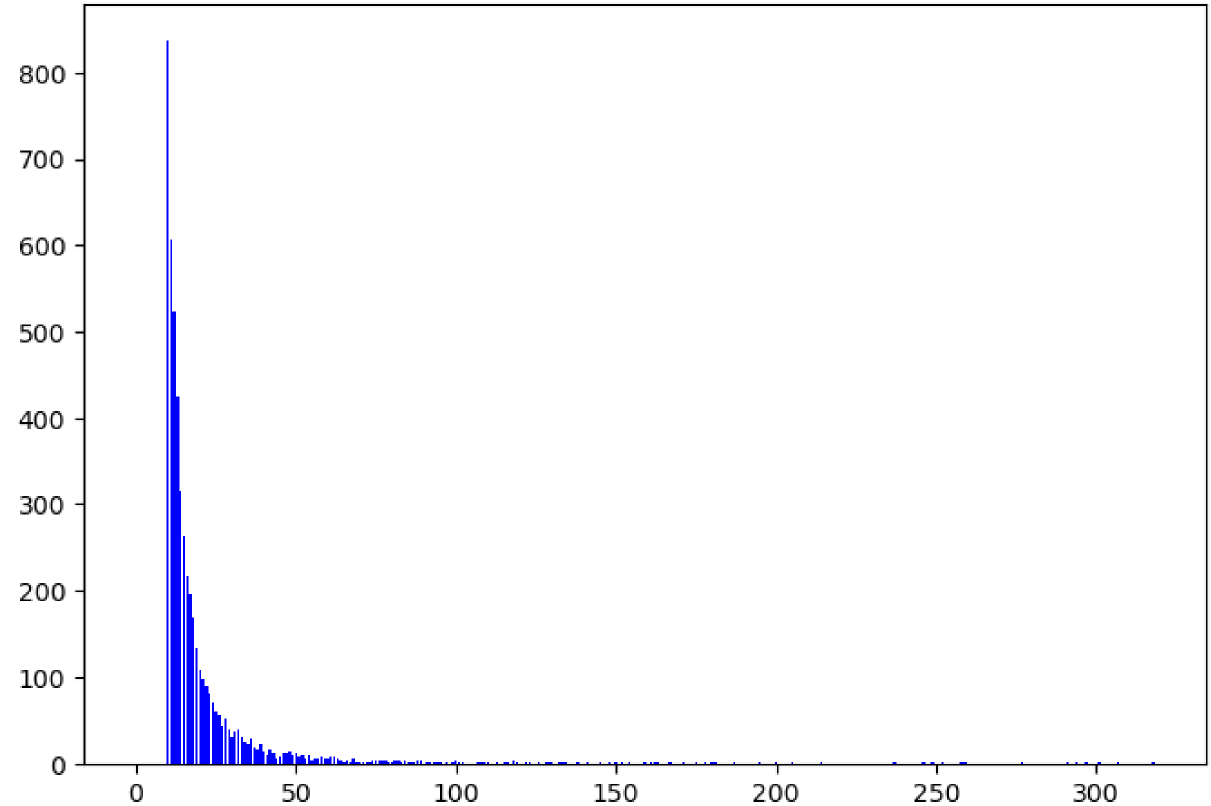
Local Attachment

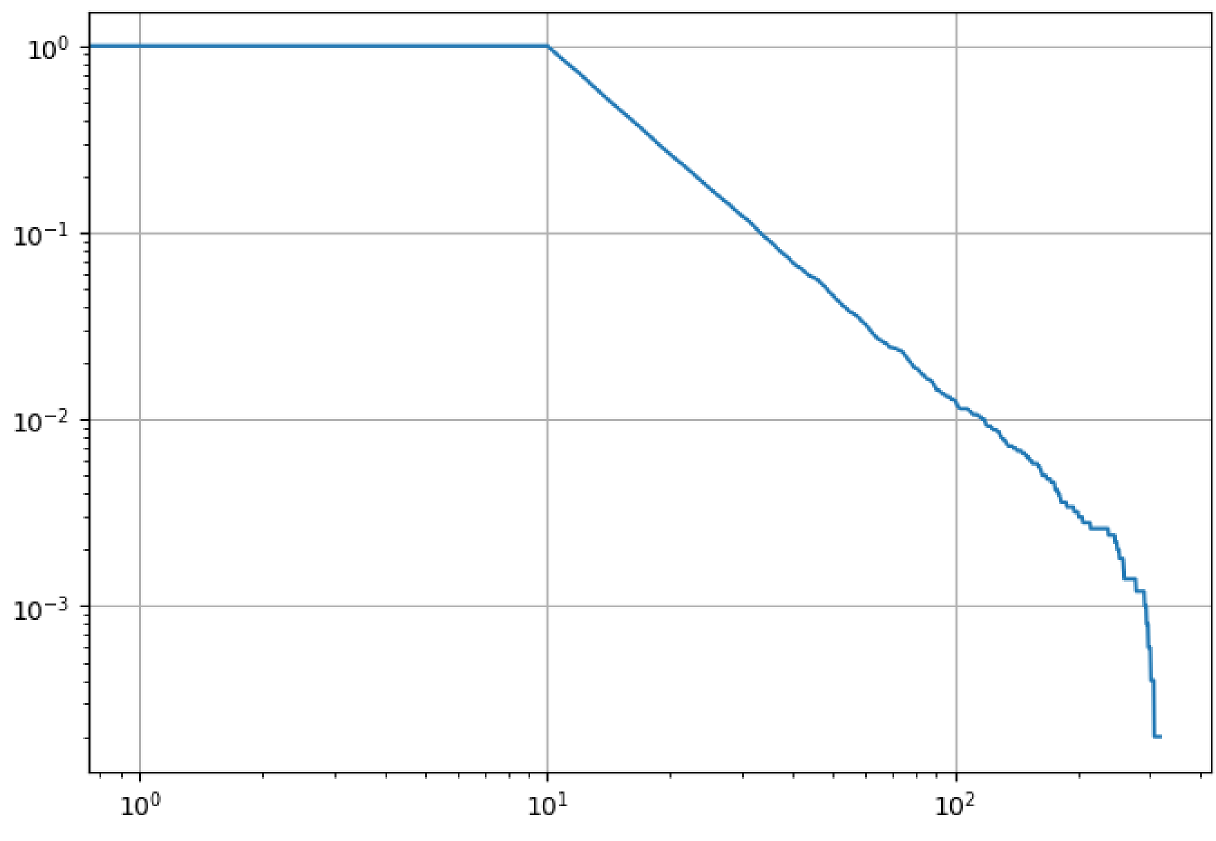
In [13]:
```
G = randomnet.local_attachment_graph(N,m,r)
```

In [215…

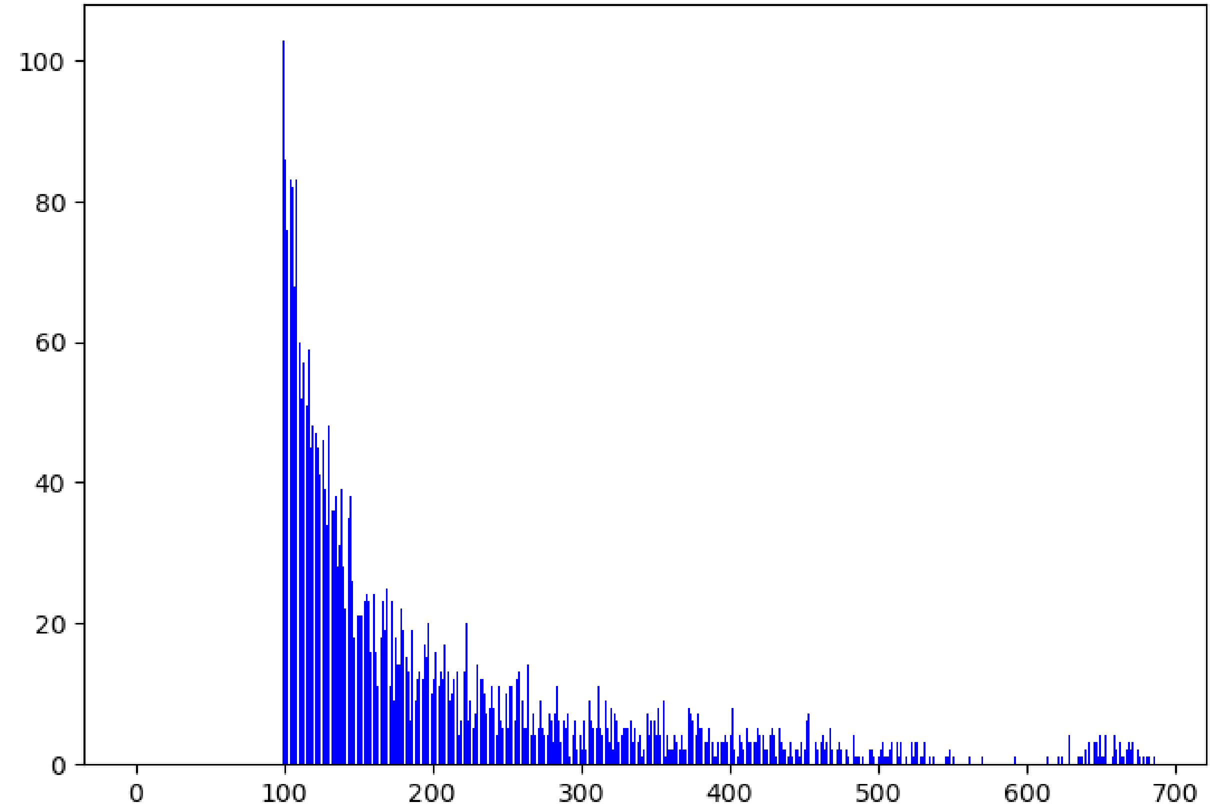
```
m = [10, 100, 1000]
r = [5, 50, 500]
kmin = [15, 120, 1000]
for mi, kmini, ri in zip(m, kmin, r):
    print(f'LAG with N = {N} | k = {kmini} | m = {mi} | r = {ri}')
    LAG = randomnet.local_attachment_graph(N,mi,ri)
    calc_powerlaw(LAG, kmini)
    plt.show()
```

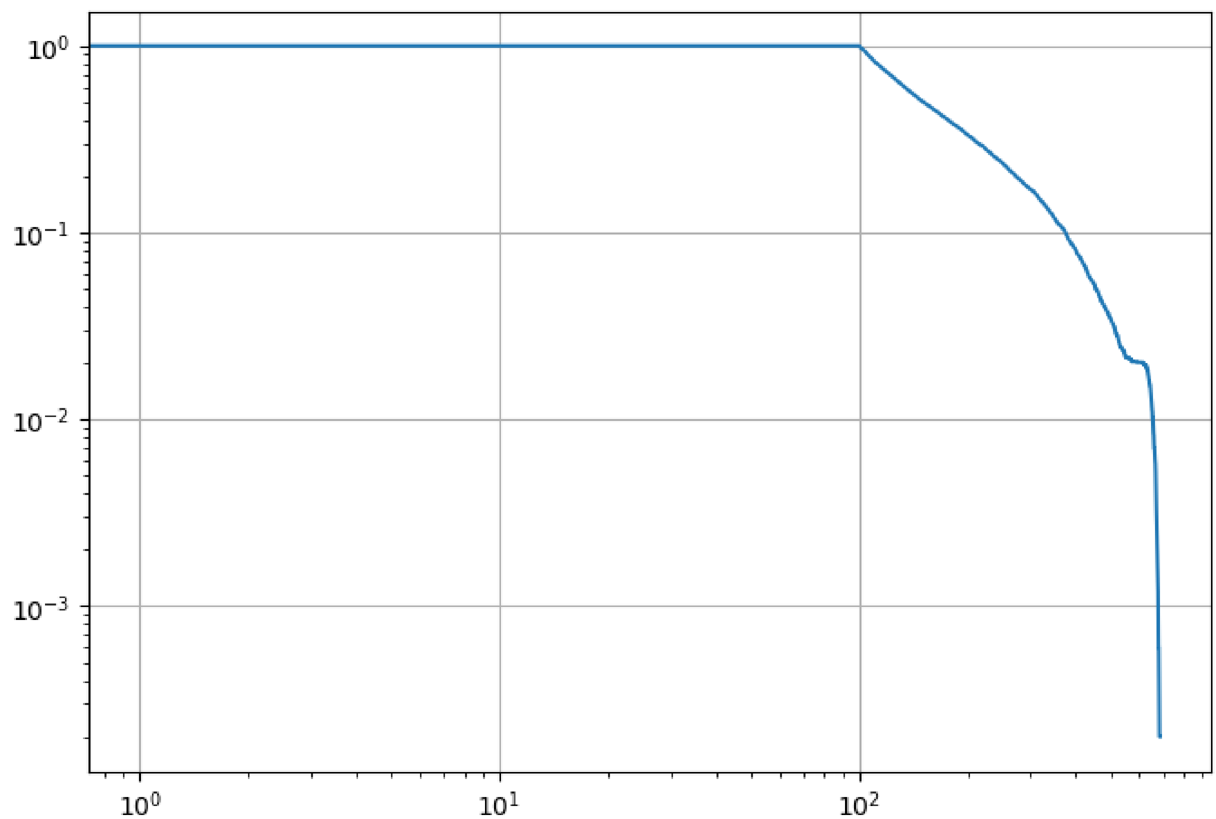
```
LAG with N = 5000 | k = 15 | m = 10 | r = 5
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.87 +/- 0.04
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


800
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300

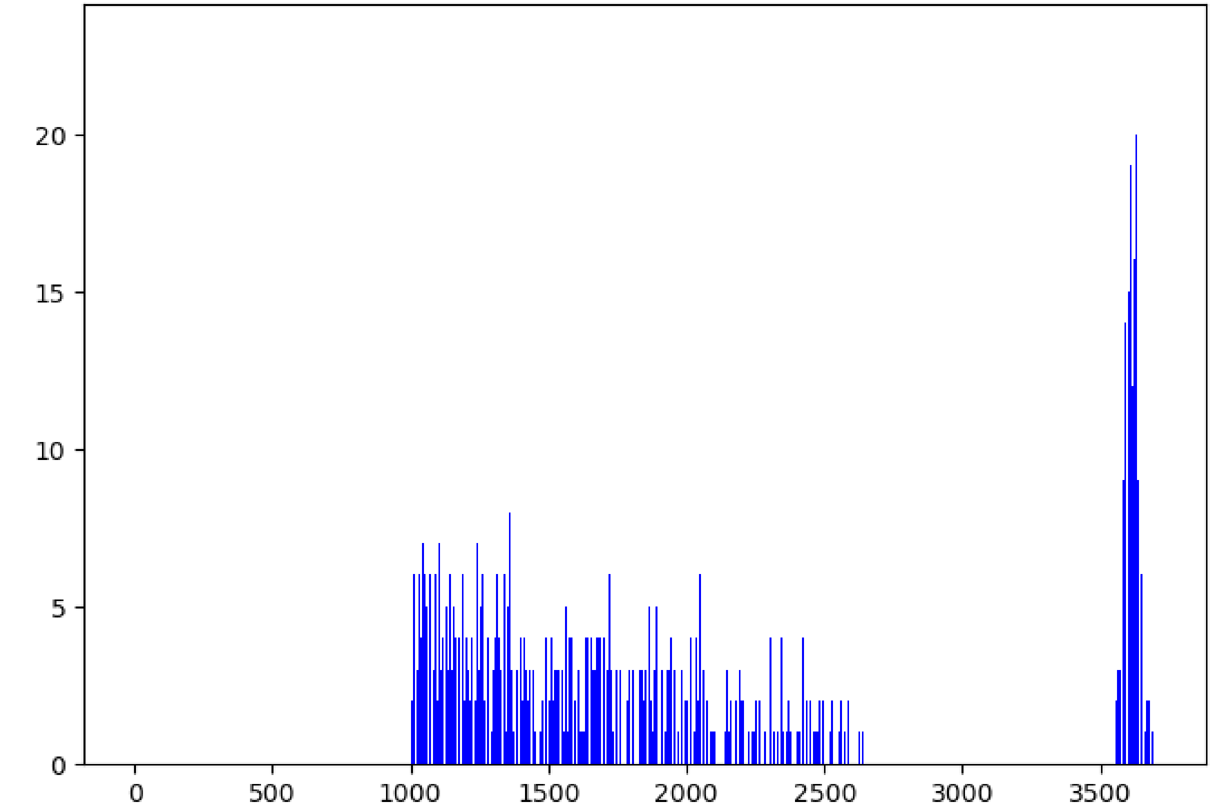

10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

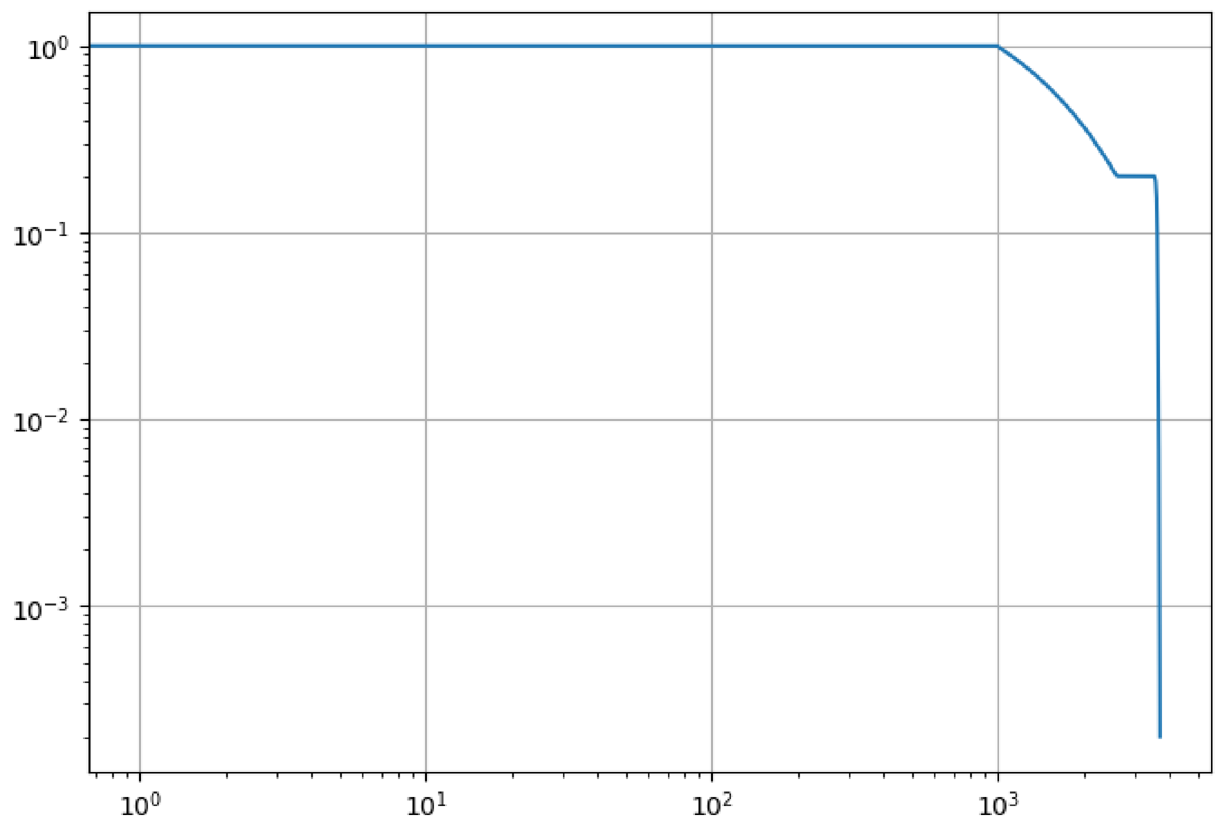
```
LAG with N = 5000 | k = 120 | m = 100 | r = 50
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.76 +/- 0.03
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


0
20
40
60
80
100
0
100
200
300
400
500
600
700


10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

```
LAG with N = 5000 | k = 1000 | m = 1000 | r = 500
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.66 +/- 0.02
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





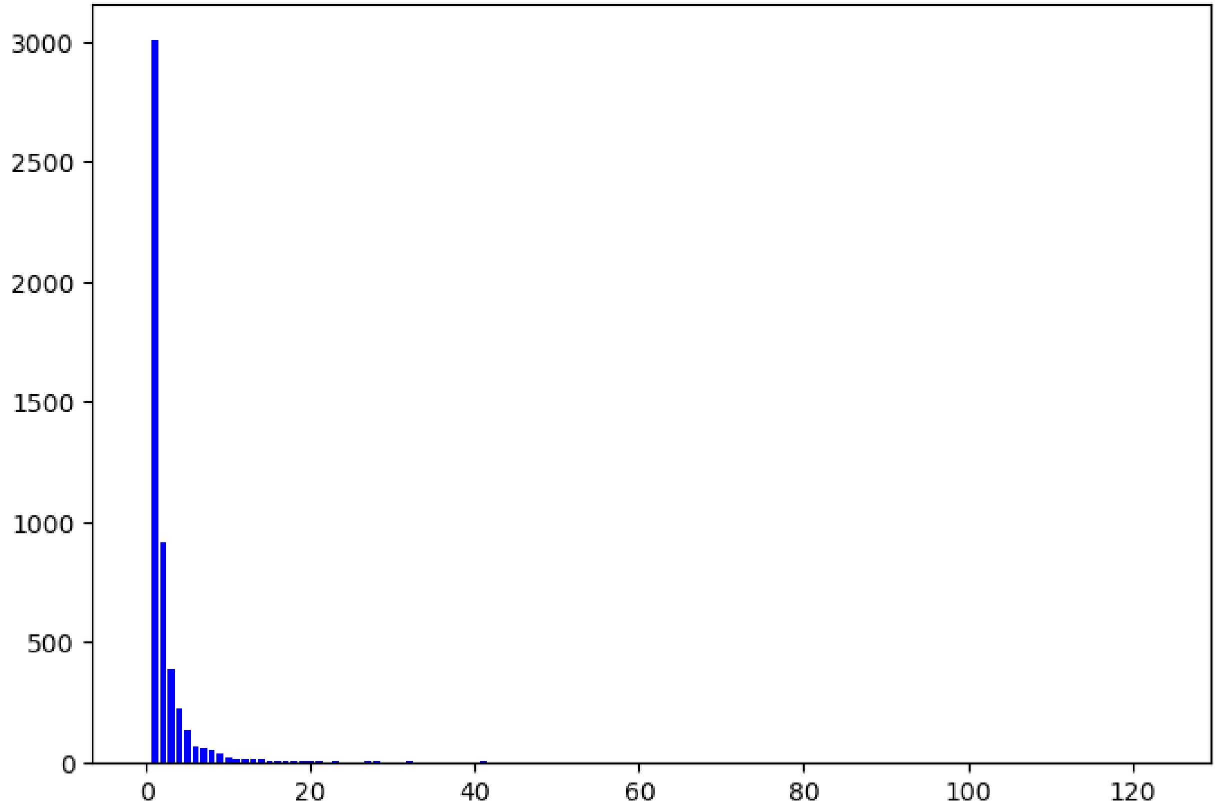
Duplication Divergence

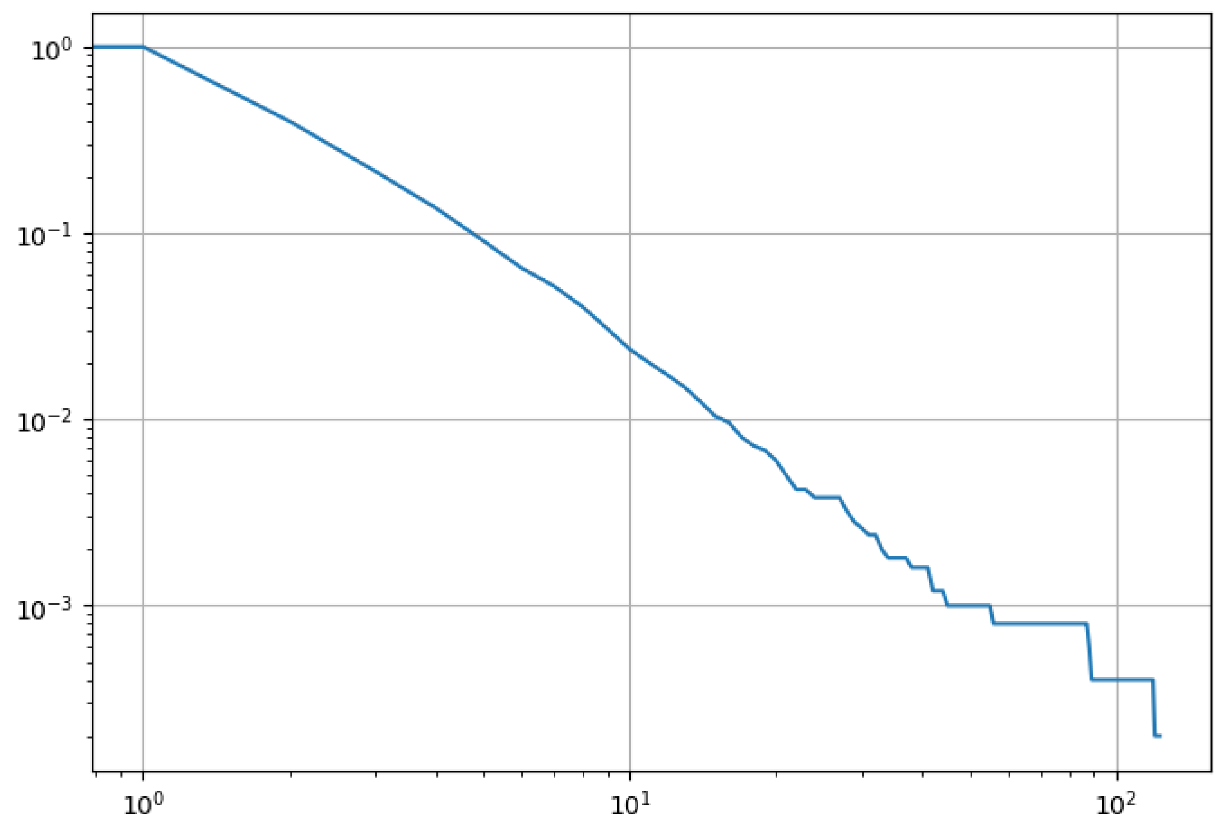
In [14]:
```
G = nx.duplication_divergence_graph(N,s)
```

In [218…

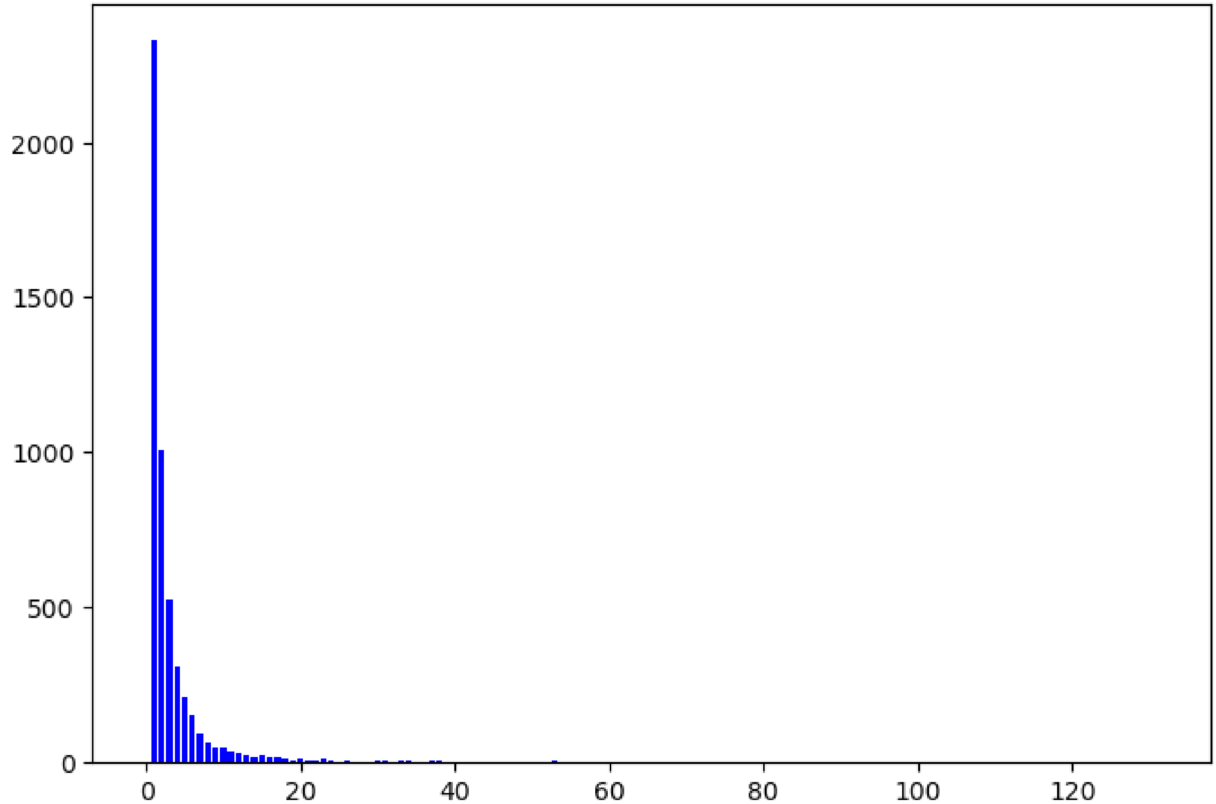
```
p = [0.05, 0.2, 0.4]
kmin = [5, 10, 50]
for pi, kmini in zip(p, kmin):
    print(f'BA with N = {N} | p = {pi} | k = {kmini}')
    DDG = nx.duplication_divergence_graph(N, pi)
    calc_powerlaw(DDG, kmini)
    plt.show()
```

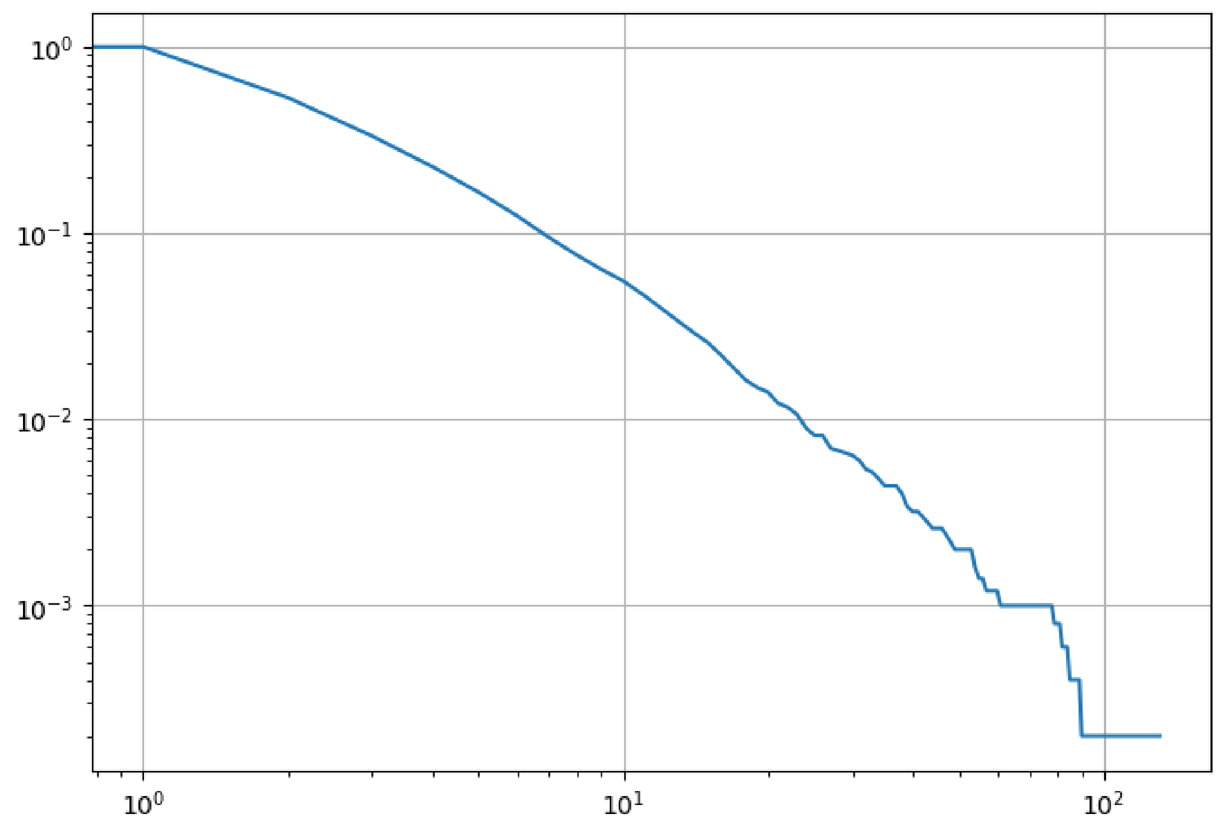
```
BA with N = 5000 | p = 0.05 | k = 5
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.74 +/- 0.08
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


3000
2500
2000
1500
1000
500
0
0
20
40
60
80
100
120

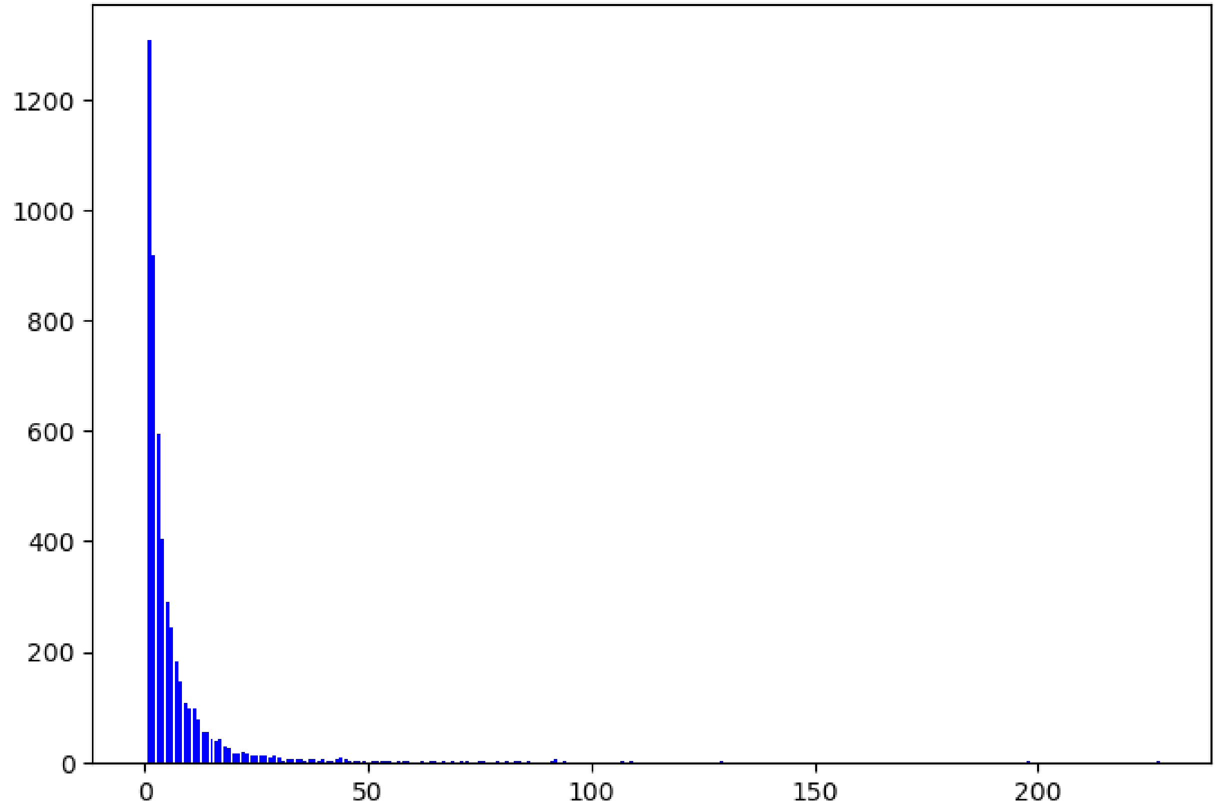

10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

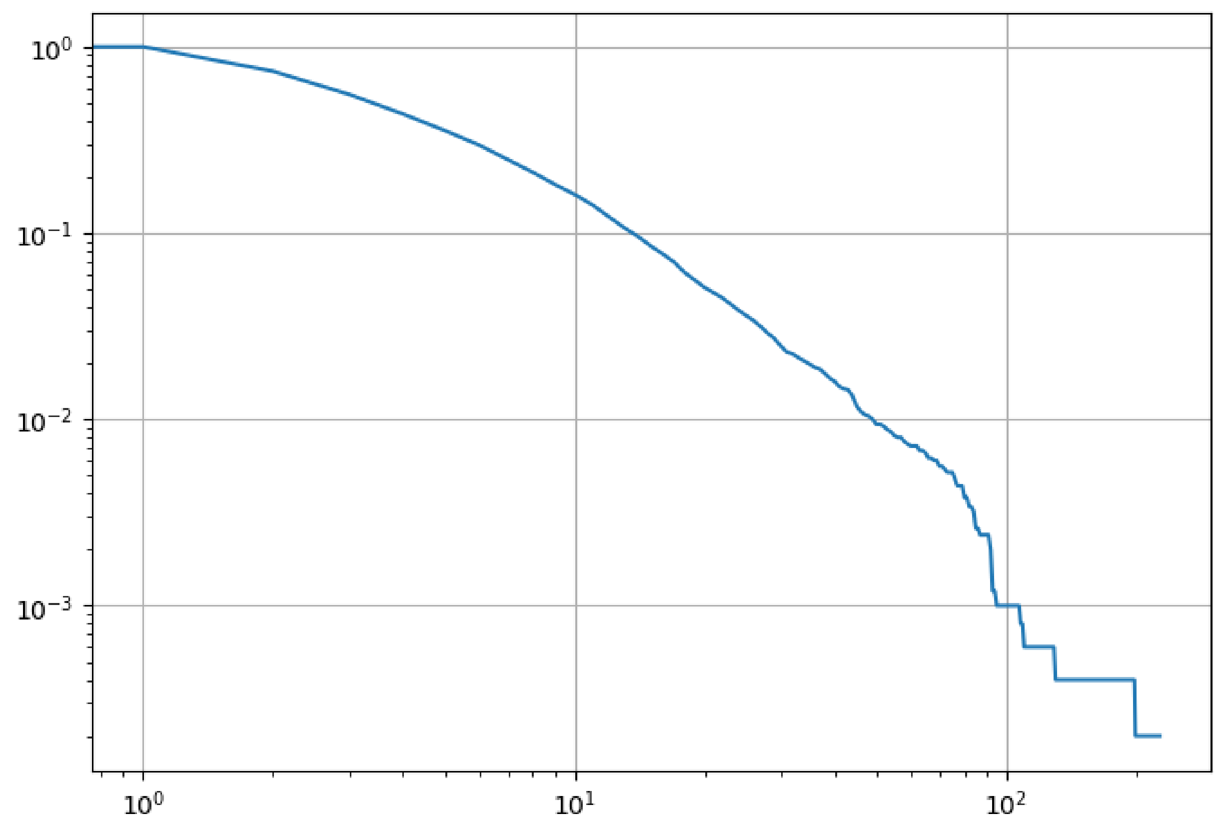
```
BA with N = 5000 | p = 0.2 | k = 10
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.93 +/- 0.12
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```


2000
1500
1000
500
0
0
20
40
60
80
100
120


10^0
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2

```
BA with N = 5000 | p = 0.4 | k = 50
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
3.29 +/- 0.33
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





In [219…

```
print('HIGHLIGHTED QUESTION -  What are their power-law exponents?')
print('The following models exhibit scale-free distributions:')
print('These models\' power law exponents vary given their parameters, but here is an
print('Barabasi_Albert_graph with an alpha value of about 3')
```

```
print('Local_Attachment_graph with an alpha value of about 2.6 to 2.8')
print('Divergence_graph with an alpha value of about 3')
```

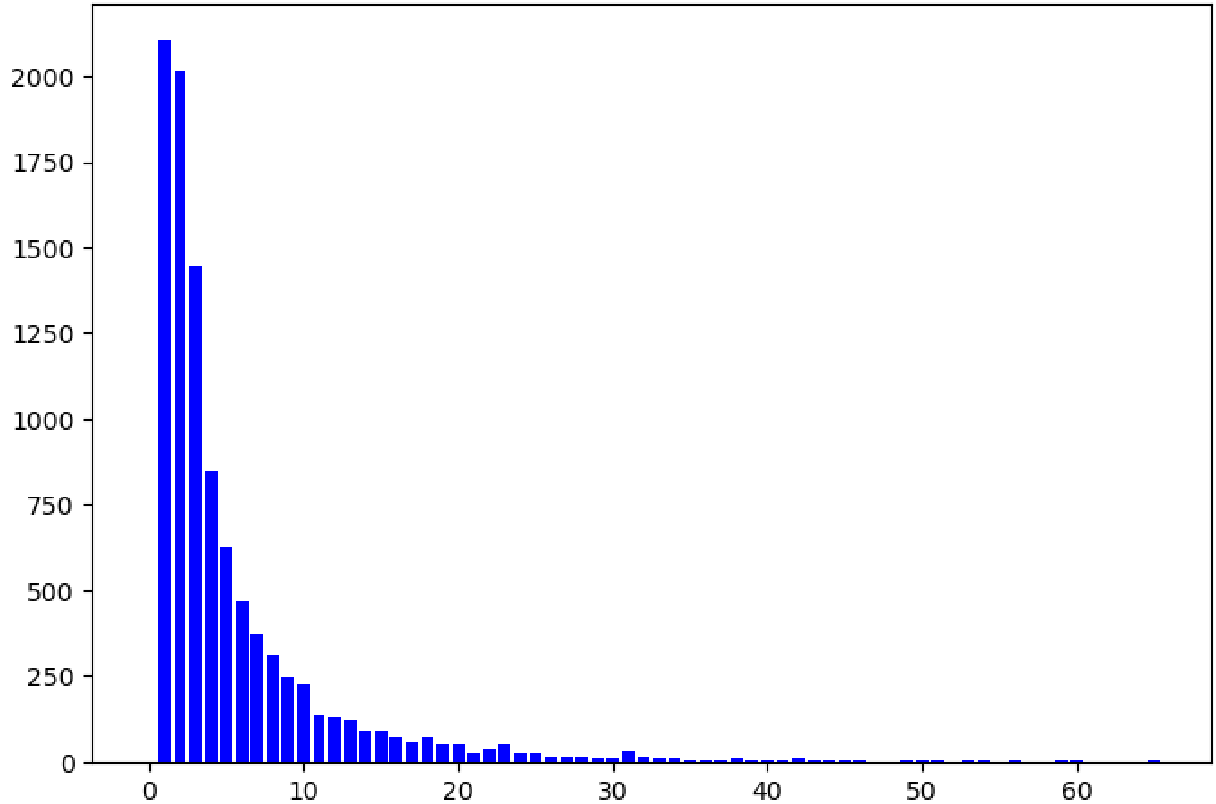
```
HIGHLIGHTED QUESTION -  What are their power-law exponents?
The following models exhibit scale-free distributions:
These models' power law exponents vary given their parameters, but here is an approxi
mate value
Barabasi_Albert_graph with an alpha value of about 3
Local_Attachment_graph with an alpha value of about 2.6 to 2.8
Divergence_graph with an alpha value of about 3
```

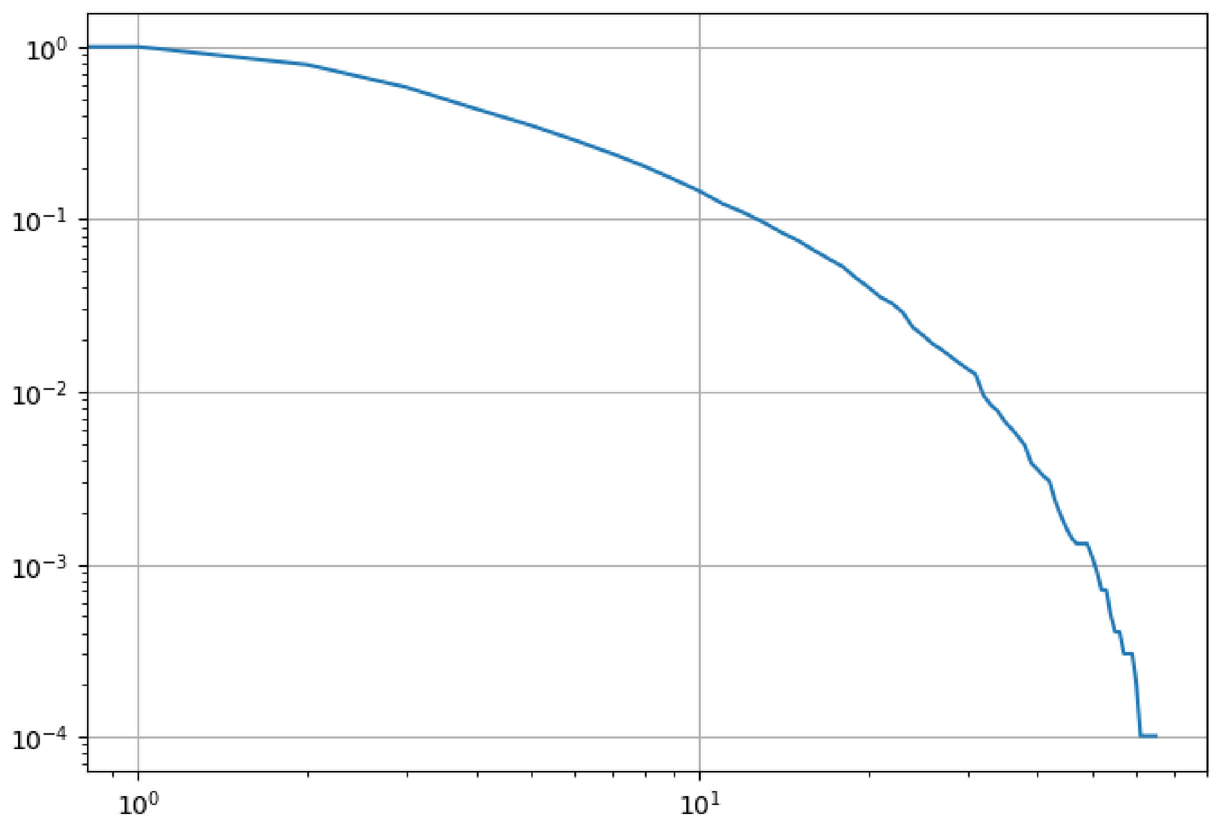
Fitting Random Models

In [238…

```
CA_G = nx.read_weighted_edgelist('ca-HepTh.edgelist')
n = CA_G.number_of_nodes()
m = CA_G.number_of_edges()
ds = degree_sequence(CA_G)
cc = nx.average_clustering(CA_G)
print(f'Number of Nodes: {n}')
print(f'Number of Edges: {m}')
print(f'Average Degree: {np.sum(ds)/n}')
print(f'Clustering Coefficient: {cc}')
calc_powerlaw(CA_G, 45)
plt.show()
print('We will be comparing our ER, LA, and DD networks to the statistics just printed
```

```
Number of Nodes: 9877
Number of Edges: 25998
Average Degree: 5.264351523742027
Clustering Coefficient: 0.4714390529669332
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
7.43 +/- 1.61
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





We will be comparing our ER, LA, and DD networks to the statistics just printed

In [235…

```
ds = degree_sequence(CA_G)
print(np.sum(ds)/n)
```

```
5.264351523742027
```

In [239…

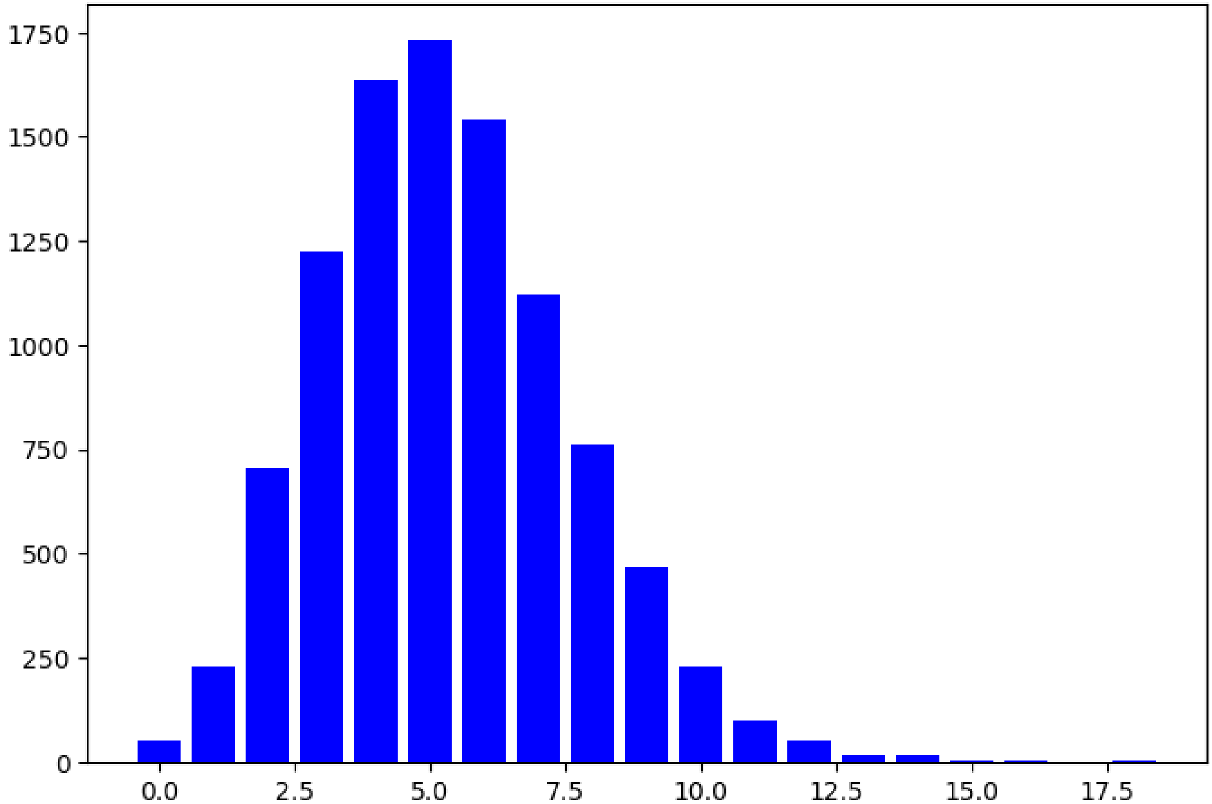
```
CA_ERG = nx.erdos_renyi_graph(n, m/(n*(n-1)/2))
print('Erdos Renyi ca-Hep tuning')
print(f'Number of Nodes: {CA_ERG.number_of_nodes()}')
print(f'Number of Edges: {CA_ERG.number_of_edges()}')
print(f'Average Degree: {np.sum(degree_sequence(CA_ERG))/n}')
print(f'Clustering Coefficient: {nx.average_clustering(CA_ERG)}')
```

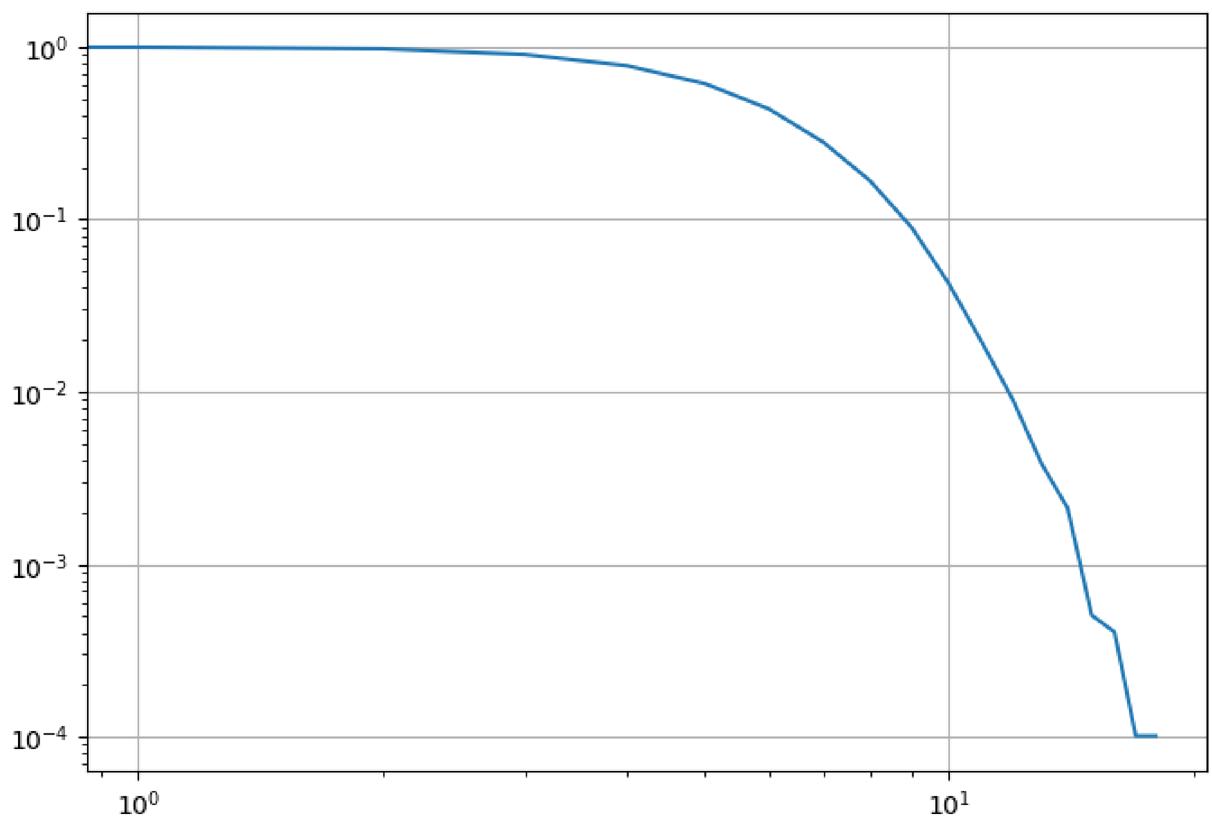
```
Erdos Renyi ca-Hep tuning
Number of Nodes: 9877
Number of Edges: 26185
Average Degree: 5.302217272451149
Clustering Coefficient: 0.0003998993368235037
```

In [240…

```
calc_powerlaw(CA_ERG, 10)
plt.show()
```

```
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
8.98 +/- 0.39
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





In [279…

```
CA_LA = randomnet.local_attachment_graph(n,3,1)
print('Local Attachment ca-Hep tuning')
print(f'Number of Nodes: {CA_LA.number_of_nodes()}')
print(f'Number of Edges: {CA_LA.number_of_edges()}')
```

```
print(f'Average Degree: {np.sum(degree_sequence(CA_LA))/n}')
print(f'Clustering Coefficient: {nx.average_clustering(CA_LA)}')
```

```
Local Attachment ca-Hep tuning
Number of Nodes: 9877
Number of Edges: 29631
Average Degree: 6.0
Clustering Coefficient: 0.4110777725194516
```

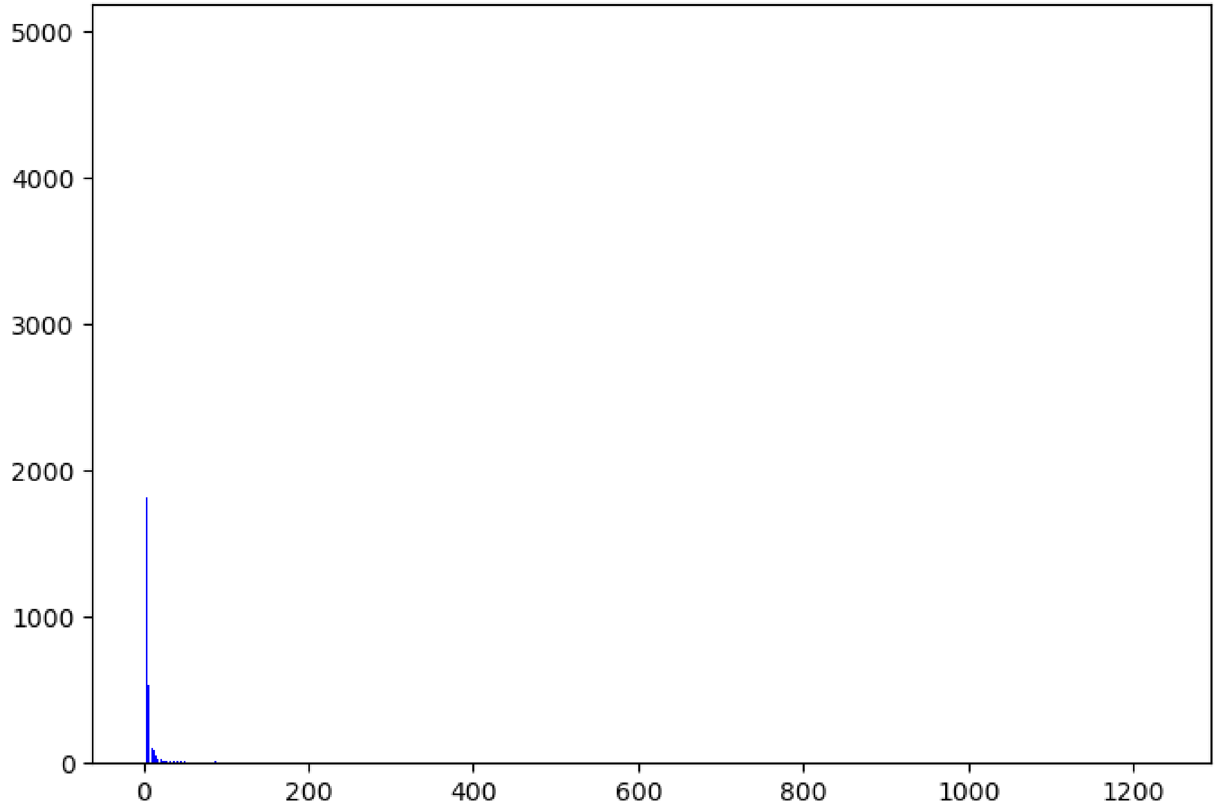
In [262…

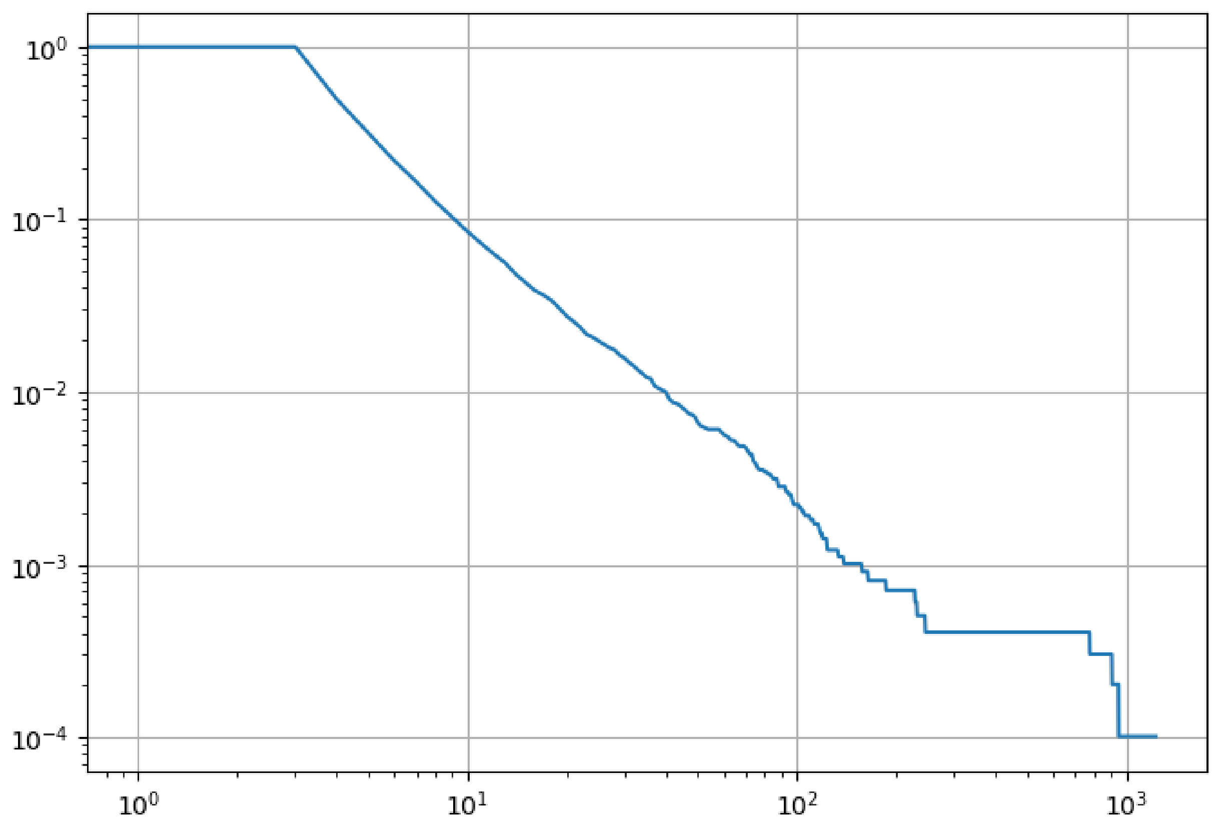
```
calc_powerlaw(CA_LA, 10)
plt.show()
```

HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty, σ for each of the networks.

2.53 +/- 0.05

HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree distribution, which will be a bar plot, and one of the cumulative degree distribution, which should





In [265…

```
CA_DD = nx.duplication_divergence_graph(n, 0.35)
print('Duplication Divergence ca-Hep tuning')
print(f'Number of Nodes: {CA_DD.number_of_nodes()}')
print(f'Number of Edges: {CA_DD.number_of_edges()}')
```

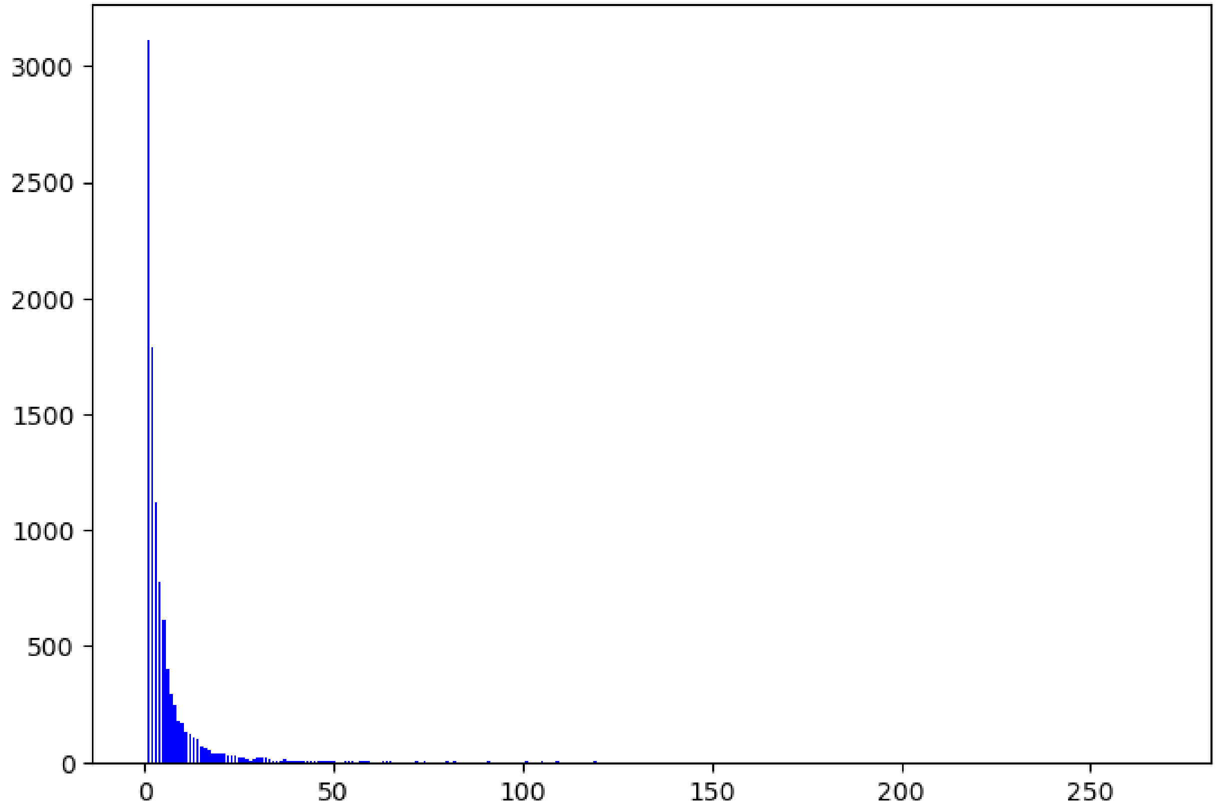
```python
print(f'Average Degree: {np.sum(degree_sequence(CA_DD))/n}')
print(f'Clustering Coefficient: {nx.average_clustering(CA_DD)}')
```

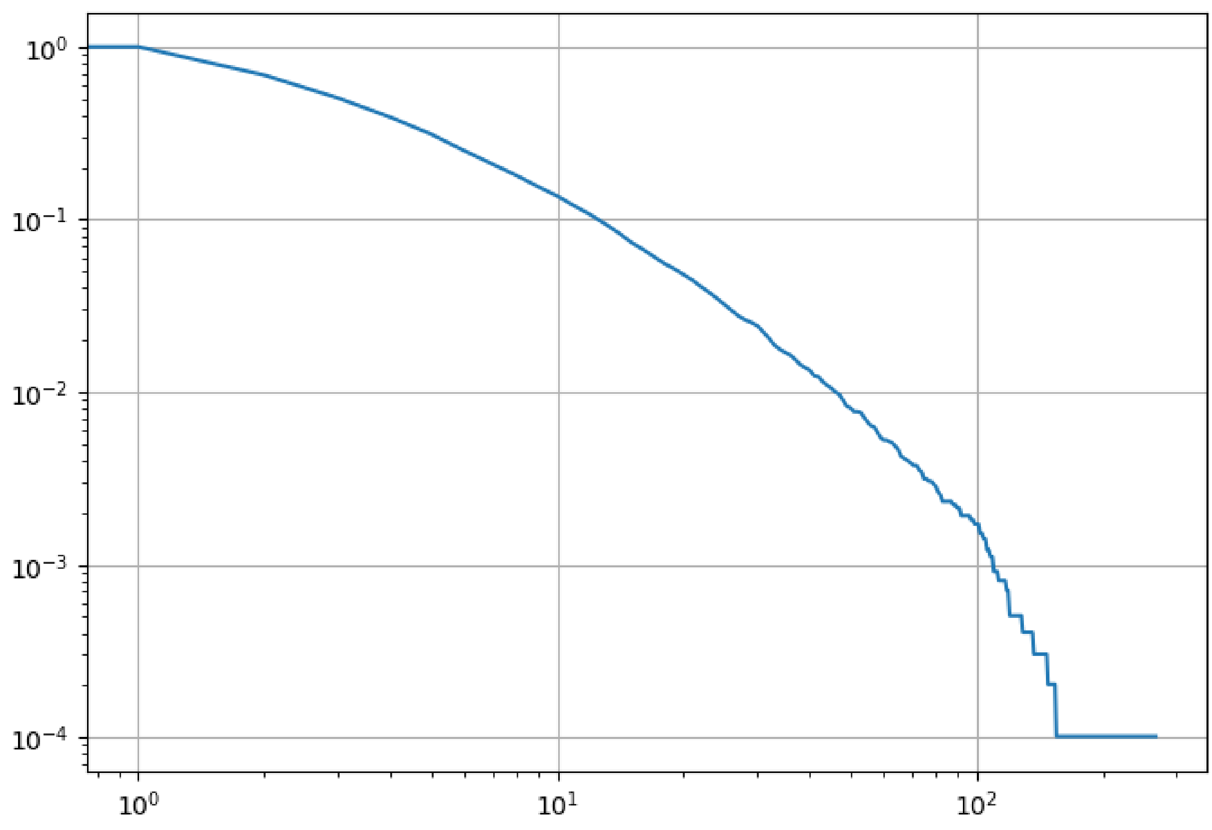
```
Duplication Divergence ca-Hep tuning
Number of Nodes: 9877
Number of Edges: 26696
Average Degree: 5.405689986838109
Clustering Coefficient: 0.0
```

In [267…

```python
calc_powerlaw(CA_DD, 10)
plt.show()
```

```
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.57 +/- 0.04
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





In [288…

```
CA_CM = nx.configuration_model(ds, create_using=nx.Graph())
print('Configuration Model ca-Hep tuning')
print(f'Number of Nodes: {CA_CM.number_of_nodes()}')
print(f'Number of Edges: {CA_CM.number_of_edges()}')
```

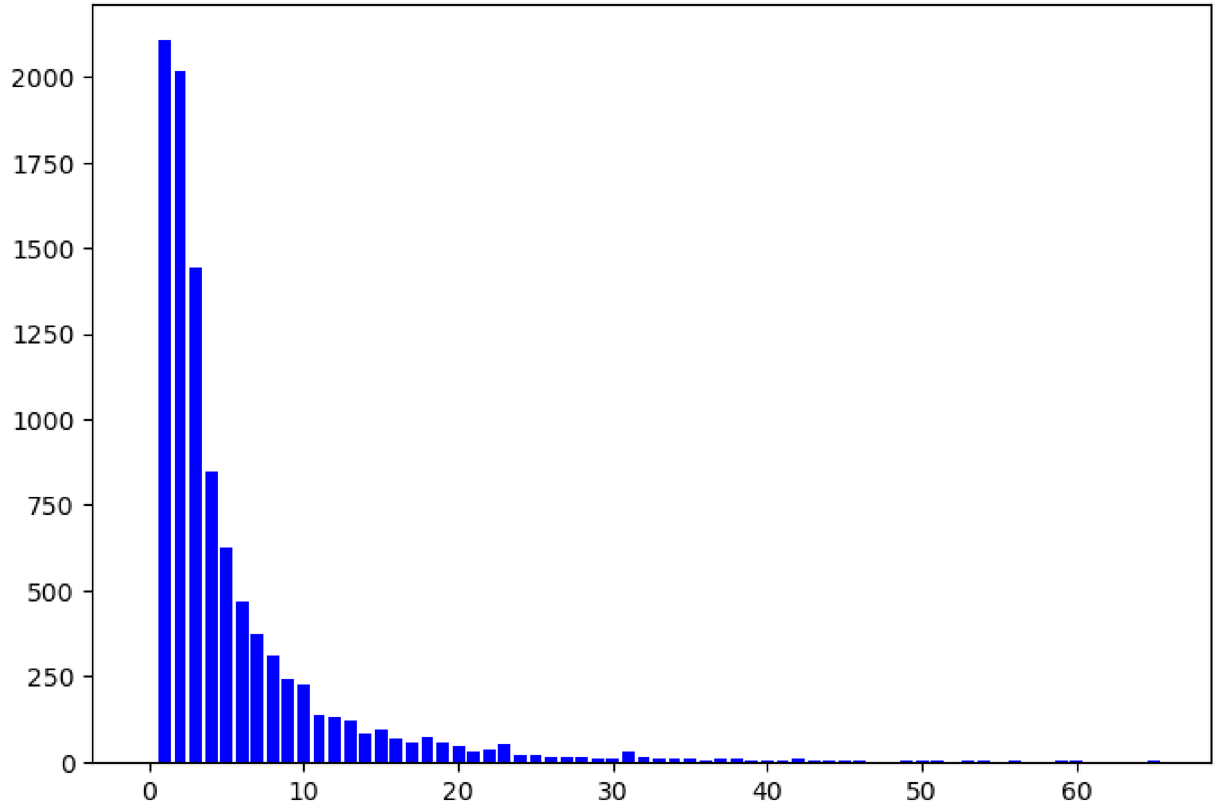
```
print(f'Average Degree: {np.sum(degree_sequence(CA_CM))/n}')
print(f'Clustering Coefficient: {nx.average_clustering(CA_CM)}')
```

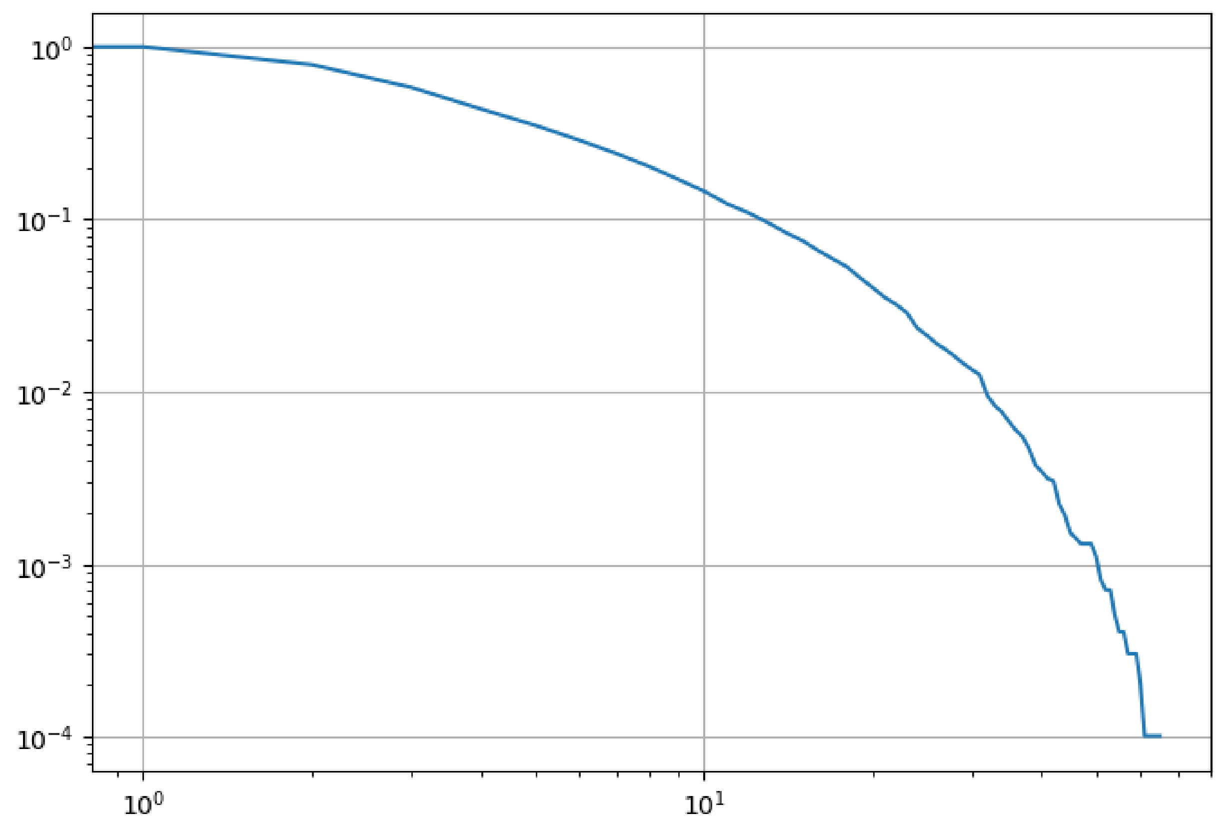
```
Configuration Model ca-Hep tuning
Number of Nodes: 9877
Number of Edges: 25973
Average Degree: 5.259289257871823
Clustering Coefficient: 0.0014703018933473328
```

In [289…

```
calc_powerlaw(CA_CM, 10)
plt.show()
```

```
HIGHLIGHTED QUESTION - calculate α and the corresponding uncertainty,  σ for each of
the networks.
2.94 +/- 0.05
HIGHLIGHTED QUESTION - Use these functions to create two plots - one of the degree di
stribution, which will be a bar plot, and one of the cumulative degree distribution,
which should
```





Sections 13.2-13.4 Configuration Model

In [310…
```
G = nx.read_weighted_edgelist('texas_road_sample.edgelist')
```

In [312…
```
print('HIGHLIGHTED QUESTION - compute the average degree of the network and compare it
print('Texas Road Sample')
avg_degree = np.sum(degree_sequence(G)) / G.number_of_nodes()
print(f'Average Degree of Network: {avg_degree}')
average_degree_neighbor = list(dict(nx.average_neighbor_degree(G)).values())
# print(average_degree_neighbor)
mean = sum(average_degree_neighbor) / len(average_degree_neighbor)
print(f'Mean of Average Degrees of Neighbors: {mean}')
```

```
HIGHLIGHTED QUESTION - compute the average degree of the network and compare it to th
e average degree of neighbor nodes.
Texas Road Sample
Average Degree of Network: 2.498556304138595
Mean of Average Degrees of Neighbors: 2.973788899582932
```

In [313…
```
G = nx.read_weighted_edgelist('international_airports.edgelist')
print('HIGHLIGHTED QUESTION - compute the average degree of the network and compare it
print('International Airports Sample Sample')
avg_degree = np.sum(degree_sequence(G)) / G.number_of_nodes()
print(f'Average Degree of Network: {avg_degree}')
average_degree_neighbor = list(dict(nx.average_neighbor_degree(G)).values())
# print(average_degree_neighbor)
mean = sum(average_degree_neighbor) / len(average_degree_neighbor)
print(f'Mean of Average Degrees of Neighbors: {mean}')
```

```
HIGHLIGHTED QUESTION - compute the average degree of the network and compare it to th
e average degree of neighbor nodes.
International Airports Sample Sample
Average Degree of Network: 10.668254508336169
Mean of Average Degrees of Neighbors: 45.069840918826
```

In [314…
```
print('HIGHLIGHTED QUESTION - Explain why we see this effect much more strongly in one
print('If we think about the texas road sample, we realize that this is similar a squa
```

```
HIGHLIGHTED QUESTION - Explain why we see this effect much more strongly in one netwo
rk than the other.
If we think about the texas road sample, we realize that this is similar a square gri
d network, so at most any node is connected to 4 other nodes (going forward, taking a
left, a right, or a u-turn). So, it makes sense that the average degree of the networ
k would be similar to the average mean degree of its neighbors. It is because of the
design of roads can only have them connect to so many intersections at a given time.
However, International Airports are not constrained by petty 2 dimensional grids. Rat
her, there can be multiple hubs an airplane can fly to. So, we might see that there a
re some small international airports that do not connect to many other places interna
tional airports. However, large hubs such as DFW, LaGuardia, or JFK are centrally loc
ated and have access and power to fly to many other airpots, hence they are also conn
ected to these smaller airports. So, when these smaller airports are connected to the
big-degree airports then the mean average degree of neighbors will be greater than th
e average degree of the network. It is because these big-degree networks are connecte
d to a significant portion of other small-degree networks.
```

In [ ]: