In [ ]:
```
import numpy as np
from matplotlib import pyplot as plt
import scipy.stats as scistats
from collections import deque, Counter
```

In [ ]:
```
class Network():
    def __init__(self,num_nodes):
        self.adj = {i:set() for i in range(num_nodes)}
        self.num_edge = 0

    def add_edge(self,i,j):
        self.adj[i].add(j)
        self.adj[j].add(i)
        self.num_edge+=1

    def neighbors (self,i):
        return self.adj[i]

    def edge_list(self):
        return [(i,j) for i in self.adj for j in self.adj[i] if i<j]
```

Q1

Implement code to sample from the configuration models where degrees k_i are sampled from a Poisson distribution and Geometric distribution.

For both cases, let n=10,000 and $< k >= 10$. Verify that the degree distribution approximately match the required PMF.

In [ ]:
```
class Poisson_Configuration_Network(Network):

    def __init__(self, num_nodes,mn=10):
        super().__init__(num_nodes)

        k = np.random.poisson(lam=mn,size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i])])
        S = np.random.permutation(S)

        if len(S)%2:
            S = S[:-1]

        S = S.reshape(-1,2)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)
```

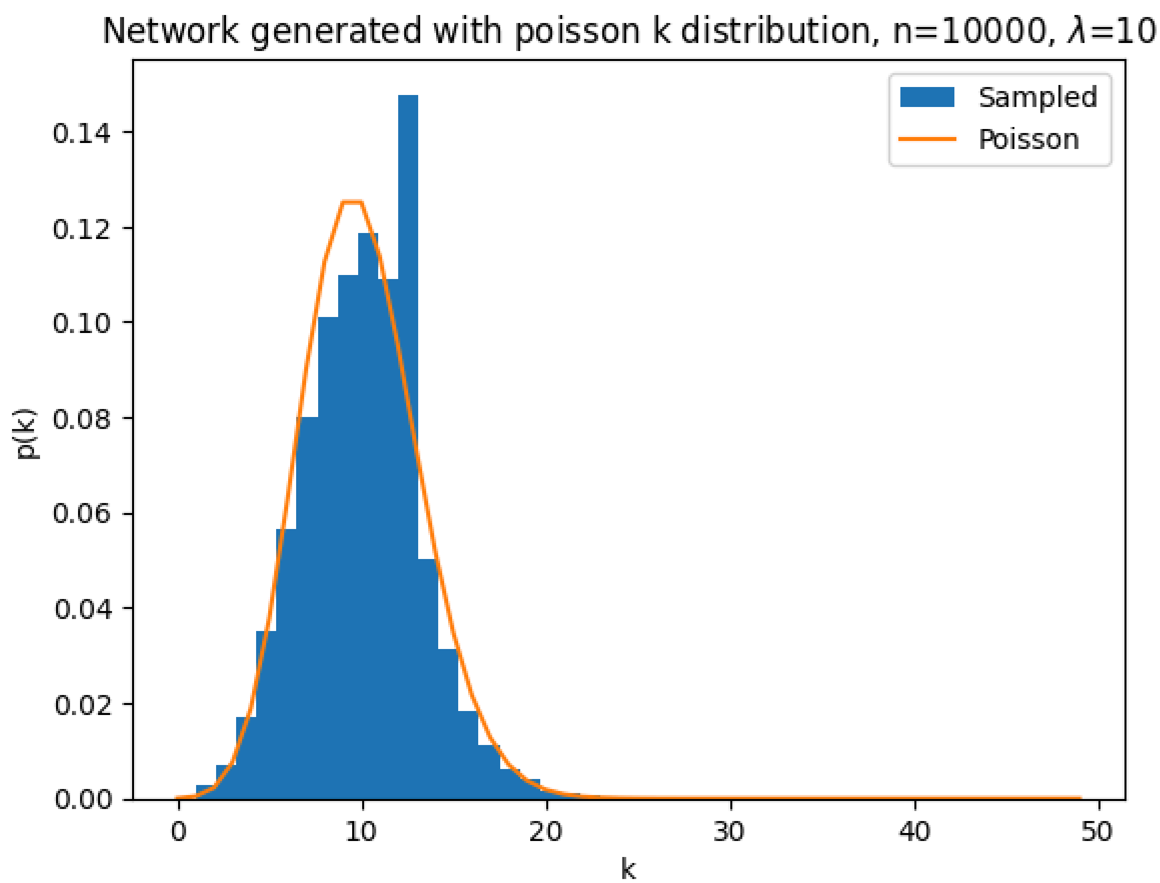
In [ ]:
```
class Geometric_Configuration_Network(Network):
```

```
    def __init__(self, num_nodes, mn=10):
        super().__init__(num_nodes)

        k = np.random.geometric(1/(mn+1), size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i]-1)])
        S = np.random.permutation(S)

        if len(S)%2:
            S = S[:-1]

        S = S.reshape(-1,2)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)
```

In [ ]:
```
n = 10000
mn = 10

poisson_network = Poisson_Configuration_Network(n,mn)
geometric_network = Geometric_Configuration_Network(n,mn)

degree_dist_poisson = []
degree_dist_geometric = []

for i in range(n):

    degree_dist_poisson.append(len(poisson_network.neighbors(i)))
    degree_dist_geometric.append(len(geometric_network.neighbors(i)))
```

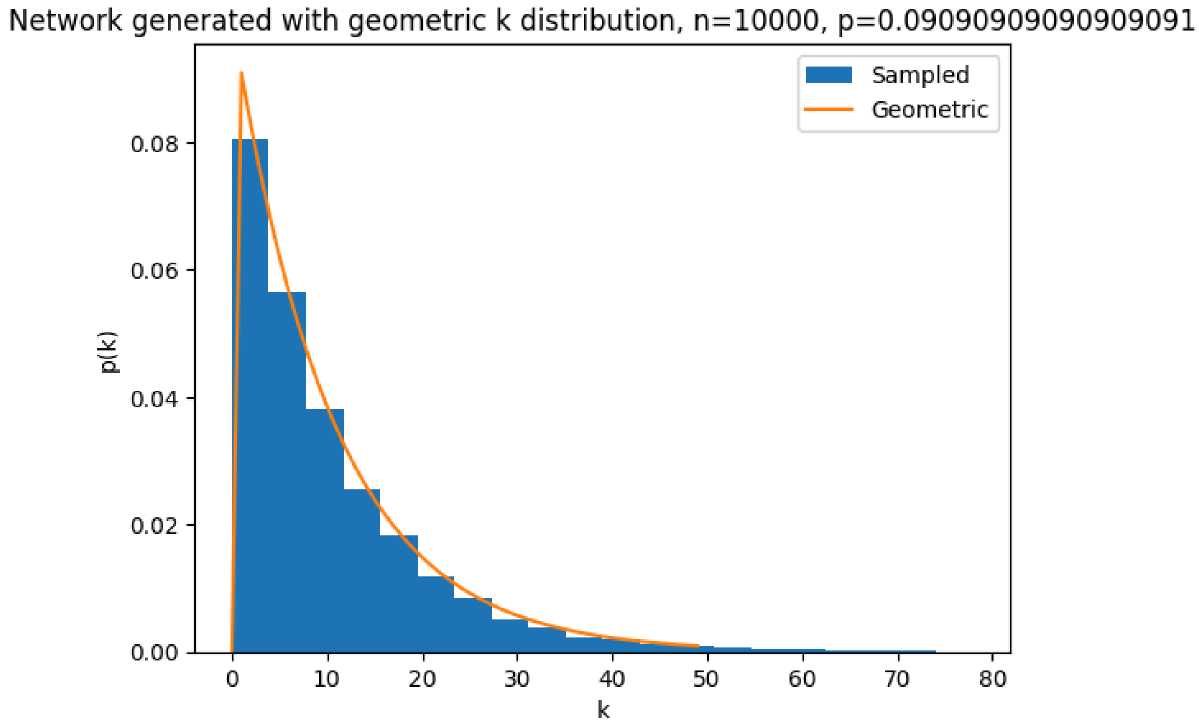
In [ ]:
```
k_range = [k for k in range(50)]

print('Sampled mean = {}'.format(np.mean(degree_dist_poisson)))

plt.hist(degree_dist_poisson,label='Sampled',density=True,bins=20)
plt.plot(k_range,scistats.poisson.pmf(k_range, mu=mn),label='Poisson')
plt.title(r'Network generated with poisson k distribution, n={}, $\lambda$={}'.form
plt.xlabel('k')
plt.ylabel('p(k)')
plt.legend()
plt.show()
```

```
Sampled mean = 10.0138
```



In [ ]:
```
k_range = [k for k in range(50)]

print('Sampled mean = {}'.format(np.mean(degree_dist_geometric)))

plt.hist(degree_dist_geometric,label='Sampled',density=True,bins=20)
plt.plot(k_range,scistats.geom.pmf(k_range,p=1/(mn+1)),label='Geometric')
plt.title('Network generated with geometric k distribution, n={}, p={}'.format(n,1/
plt.xlabel('k')
plt.ylabel('p(k)')
plt.legend()
plt.show()
```

```
Sampled mean = 9.9572
```



Q2

The friendship paradox states that

$$<\kappa>\geq<k>$$

Meaning on average, people have fewer friends than their friends.

Verify the friendship paradox numerically by repeating the following :

1. Choose a node at random
2. Choose a neighbour of that node at random
3. Record both of their degrees

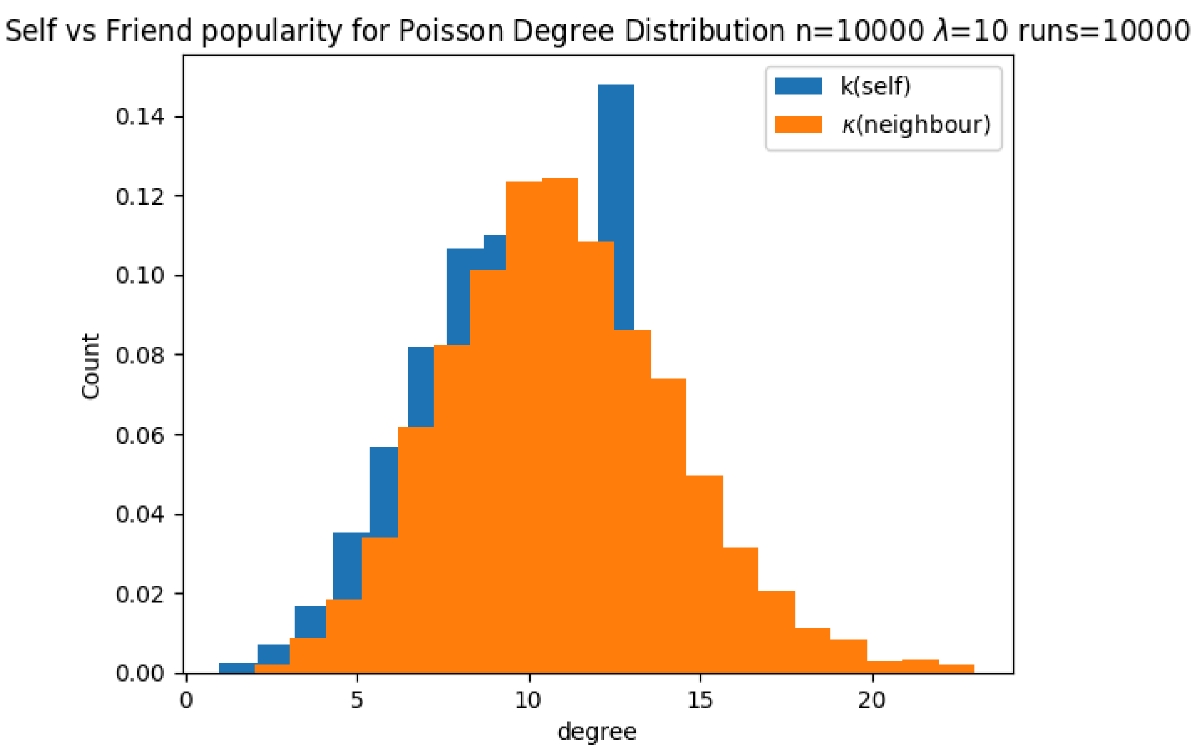
Finally, make a histogram of the degrees. Ignore nodes of degree 0.

In [ ]:
```
samples = 10000
n=10000

random_self = np.random.randint(0,n-1,size=samples)
poisson_self_k_array = []
poisson_friend_k_array = []
geom_self_k_array = []
geom_friend_k_array = []

for i in random_self:

    p_i = i
    g_i = i
```

```
    while not len(poisson_network.neighbors(p_i)):
        poisson_self_k_array.append(0)
        p_i = np.random.randint(0,n-1)

    poisson_friends =  poisson_network.neighbors(p_i)
    poisson_self_k_array.append(len(poisson_friends))
    poisson_friend_k_array.append(len(poisson_network.neighbors(np.random.choice(li

    while not len(geometric_network.neighbors(g_i)):
        geom_self_k_array.append(0)
        g_i = np.random.randint(0,n-1)

    geometric_friends = geometric_network.neighbors(g_i)
    geom_self_k_array.append(len(geometric_friends))
    geom_friend_k_array.append(len(geometric_network.neighbors(np.random.choice(lis
```

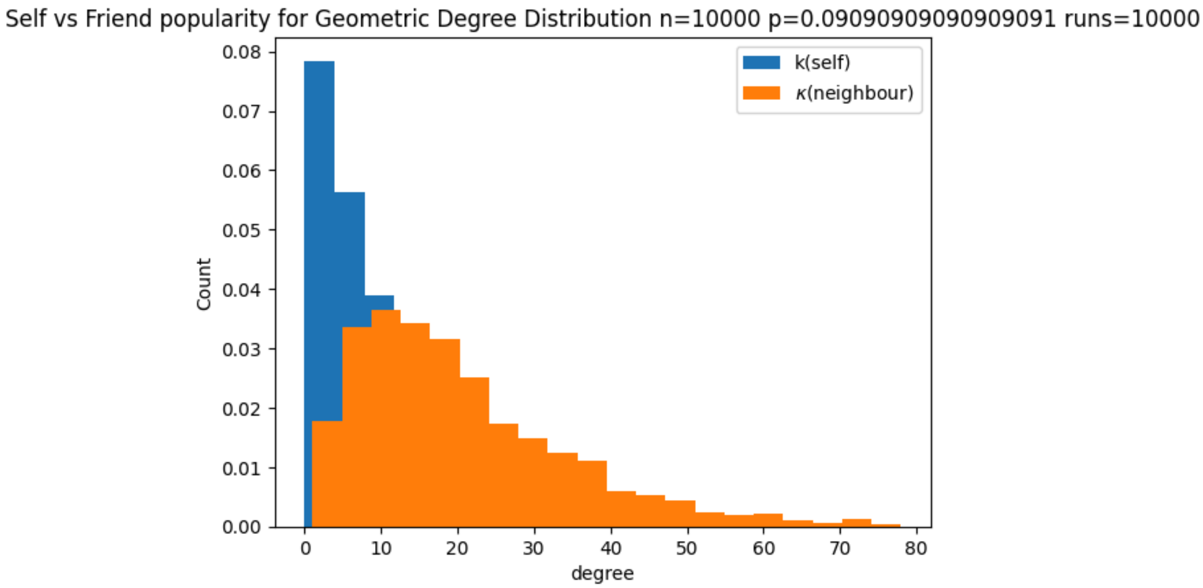
In [ ]:

```
Q4_poisson = poisson_friend_k_array

plt.hist(poisson_self_k_array,density=True,label='k(self)',bins=20)
plt.hist(poisson_friend_k_array,density=True,label=r'$\kappa$(neighbour)',bins=20)
plt.title(r'Self vs Friend popularity for Poisson Degree Distribution n={} $\lambda
plt.xlabel('degree')
plt.ylabel('Count')
plt.legend()
plt.show()

mn_k_poisson = np.mean(poisson_self_k_array)
mn_kappa_poisson = np.mean(poisson_friend_k_array)

print('Mean of k(self) = {}'.format(mn_k_poisson))
print(r'Mean of $\kappa$(neighbour) = {}'.format(mn_kappa_poisson))
```



```
Mean of k(self) = 9.9563
Mean of $\kappa$(neighbour) = 11.0309
```

In [ ]:
```
Q4_geom = geom_friend_k_array

plt.hist(geom_self_k_array,density=True,label='k(self)',bins=20)
plt.hist(geom_friend_k_array,density=True,label=r'$\kappa$(neighbour)',bins=20)
plt.title('Self vs Friend popularity for Geometric Degree Distribution n={} p={} ru
plt.xlabel('degree')
plt.ylabel('Count')
plt.legend()
plt.show()

mn_k_geom = np.mean(geom_self_k_array)
mn_kappa_geom =np.mean(geom_friend_k_array)

print('Mean of k(self) = {}'.format(mn_k_geom))
print(r'Mean of $\kappa$(neighbour) = {}'.format(mn_kappa_geom))
```




```
Mean of k(self) = 10.021833256563795
Mean of $\kappa$(neighbour) = 20.4104
```

Q3

Make a histogram for

$$\Delta_i = \kappa_i - k_i$$

Show that the mean is larger than 0

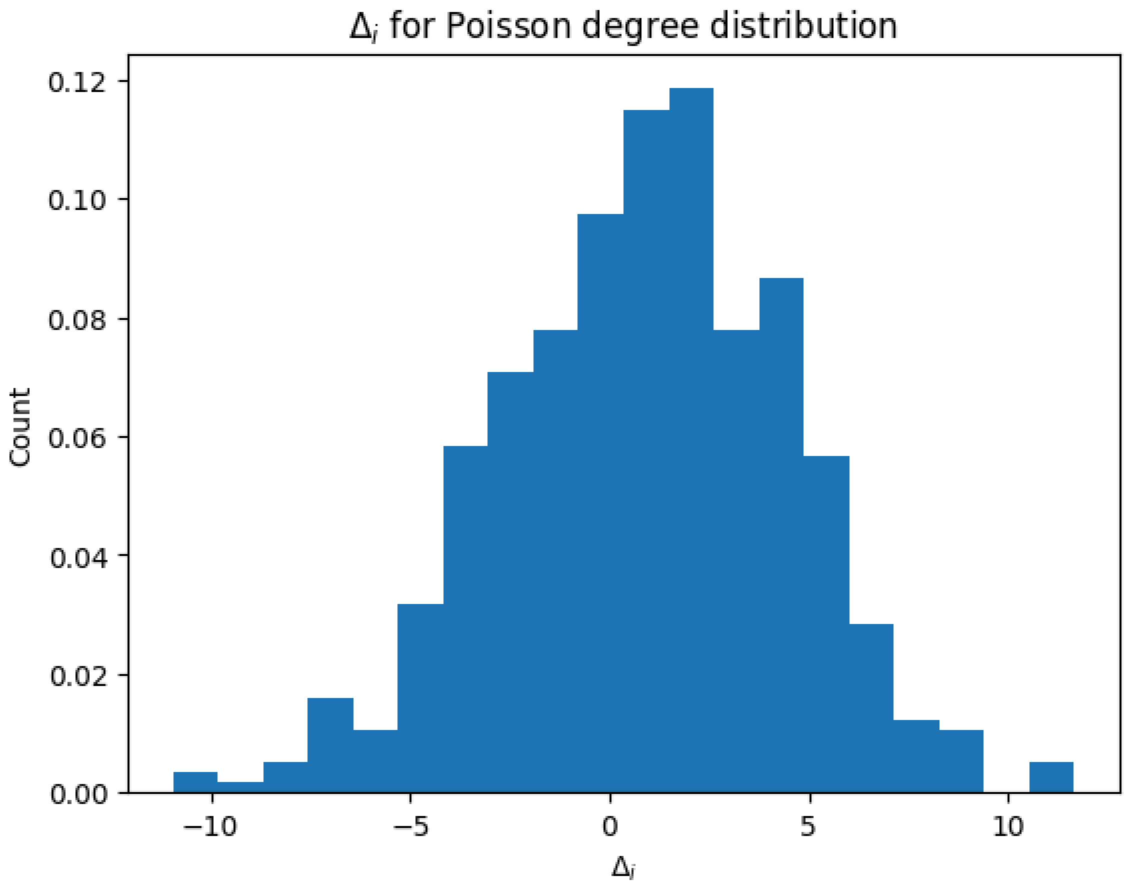
If degree(i) = 0 , the degree difference is 0

In [ ]:
```
samples = 500
n = 10000

random_self = np.random.randint(0,n-1,size=samples)
```

```
poisson_self_k_array = []
poisson_delt_k_avg_array = []
geom_self_k_array = []
geom_delt_k_avg_array = []

for i in random_self:

    p_i = i
    g_i = i

    while len(poisson_network.neighbors(p_i))==0:
        poisson_delt_k_avg_array.append(0)
        p_i = np.random.randint(0,n-1)

    poisson_friends =  poisson_network.neighbors(p_i)
    poisson_self_k_array.append(len(poisson_friends))
    poisson_friend_k_array = []

    for j in poisson_friends:
        poisson_friend_k_array.append(len(poisson_network.neighbors(j)))

    poisson_delt_k_avg_array.append(np.mean(poisson_friend_k_array)-len(poisson_fri


    while len(geometric_network.neighbors(g_i))==0:
        geom_delt_k_avg_array.append(0)
        g_i = np.random.randint(0,n-1)

    geometric_friends = geometric_network.neighbors(g_i)
    geom_self_k_array.append(len(geometric_friends))
    geom_friend_k_array = []

    for j in geometric_friends:
        geom_friend_k_array.append(len(geometric_network.neighbors(j)))

    geom_delt_k_avg_array.append(np.mean(geom_friend_k_array)-len(geometric_friends
```

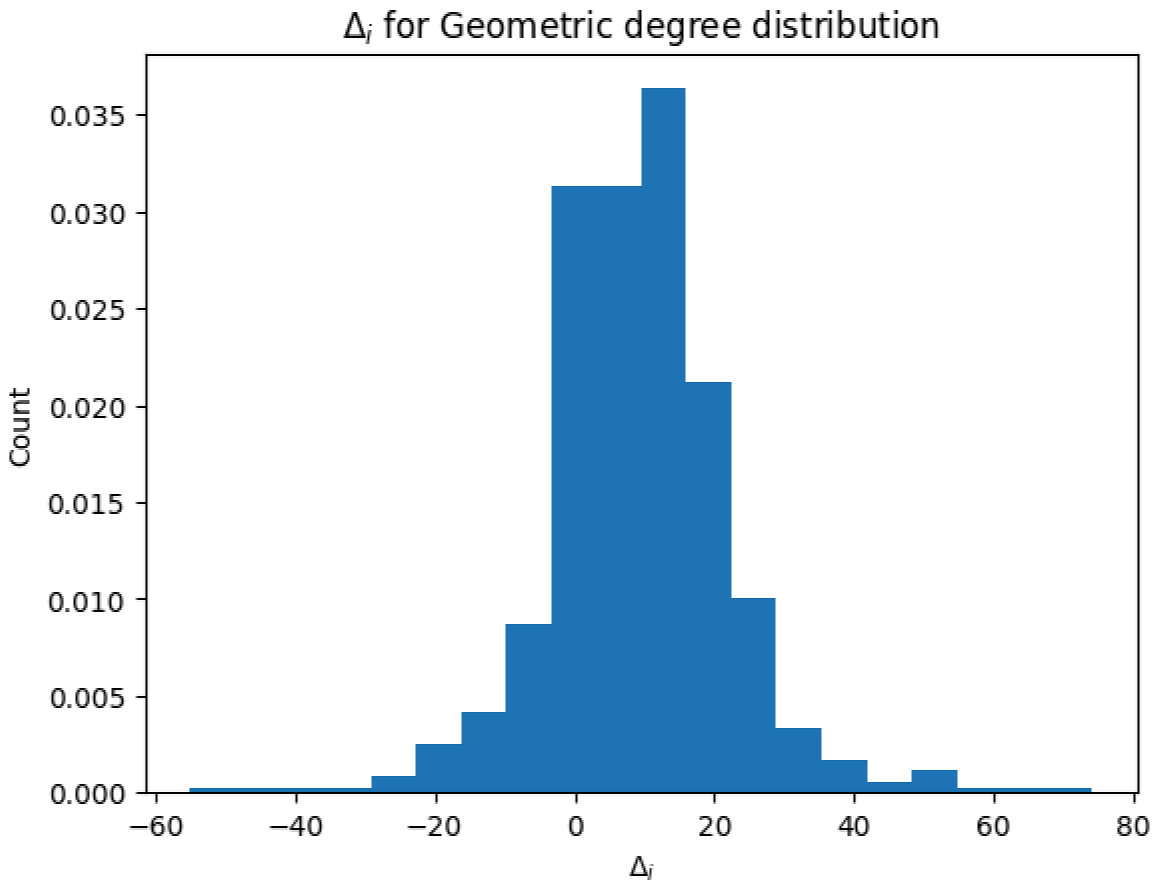
In [ ]:

```
plt.hist(poisson_delt_k_avg_array,density=True,bins=20)
plt.xlabel(r'$\Delta_{i}$')
plt.ylabel('Count')
plt.title(r'$\Delta_{i}$ for Poisson degree distribution')
plt.show()

print(r'Mean $\Delta_i$ = {}'.format(np.mean(poisson_delt_k_avg_array)))
```




```
Mean $\Delta_i$ = 0.8948931634327145
```

In [ ]:
```
plt.hist(geom_delt_k_avg_array,density=True,bins=20)
plt.xlabel(r'$\Delta_{i}$')
plt.ylabel('Count')
plt.title(r'$\Delta_{i}$ for Geometric degree distribution')
plt.show()

print(r'Mean $\Delta_i$ = {}'.format(np.mean(geom_delt_k_avg_array)))
```




```
Mean $\Delta_i$ = 8.95533518856917
```

Q4

Proof of q_k :

Assume that we randomly pick a vertex and assume this vertex has degree bigger than 0. What is the probability that a randomly chosen neighbour of this vertex having degree k? This does not follow p_k since we know for sure that our vertex has degree of at least one, and hence any non-zero p_0 will be at odds with this fact. Note that a network has in total $2m$ half-edges. Such that $2m = \sum(k_i)$. For a neighbour of degree k, there are k half-edges along which we can approach this neighbour, and the total number of half-edges (excluding the edge followed initially to get to the random vertex) is 2m-1. The total number of such neighbours in the network is $n_k = p_k * n$. Therefore

$$q_k = \frac{k * n * p_k}{2m-1} \approx \frac{k * n * p_k}{2m} = \frac{k * p_k}{\sum_{k'} k' p_{k'}}$$

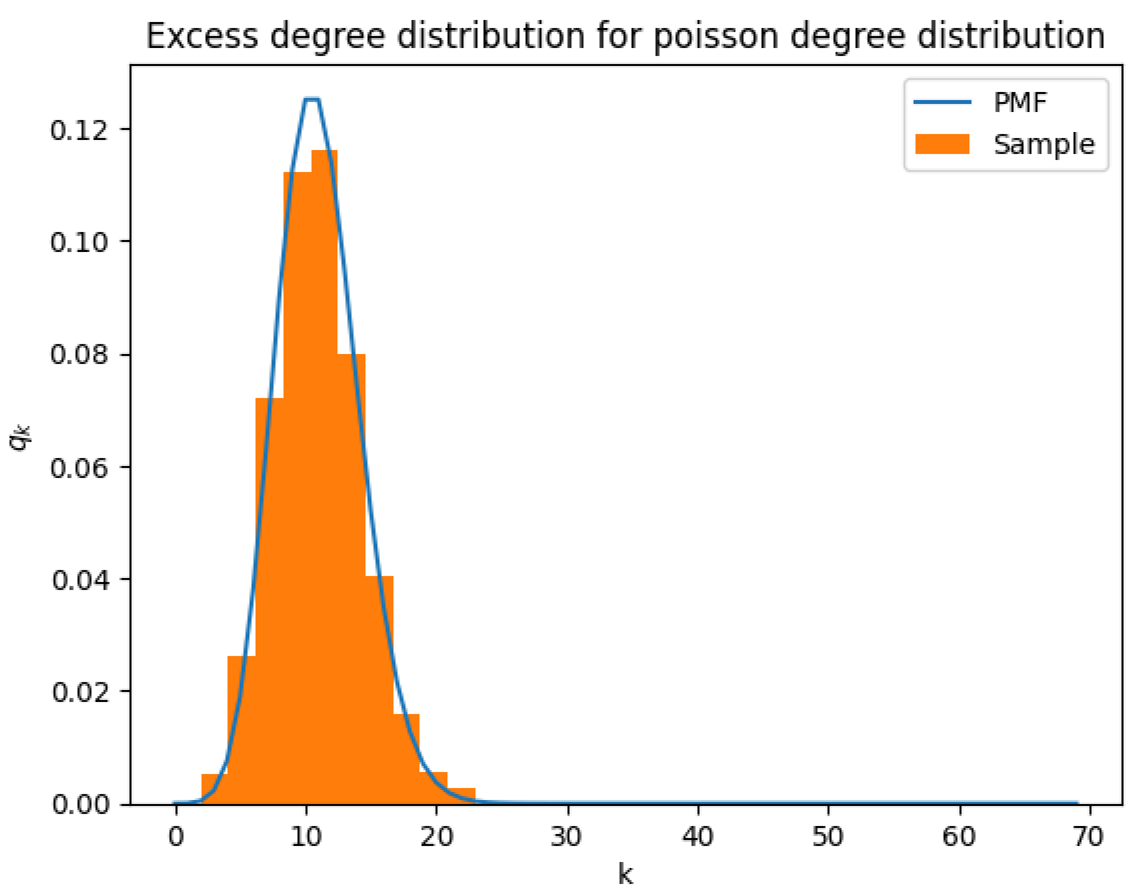
Source :
https://www.ndsu.edu/pubweb/~novozhil/Teaching/767%20Data/47_pdfsam_notes.pdf

https://sites.santafe.edu/~aaronc/courses/5352/fall2013/csci5352_2013_L12.pdf

NOTE : Derive equation of mean degree and mean excess degree in terms of variance of degree distribution

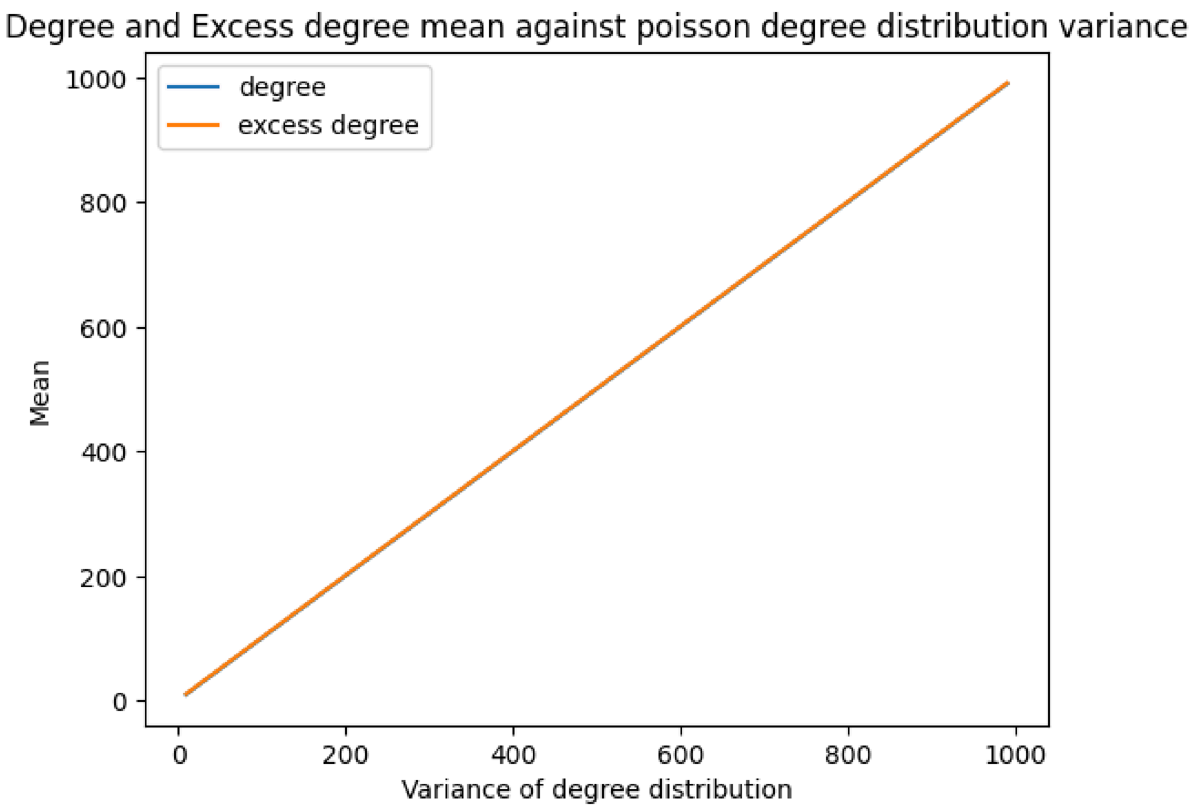
In [ ]:
```
n=10000
mn = 10
k_range = [k for k in range(70)]

mean, var = scistats.poisson.stats(mu = mn, moments='mv')

poisson_excess_pmf = [k*p/mean for k,p in zip(k_range,scistats.poisson.pmf(k_range,
```

In [ ]:
```
plt.plot(k_range,poisson_excess_pmf,label="PMF")
plt.hist(Q4_poisson,density=True,label='Sample')
plt.title('Excess degree distribution for poisson degree distribution')
plt.xlabel('k')
plt.ylabel(r'$q_{k}$')
plt.legend()
plt.show()
```



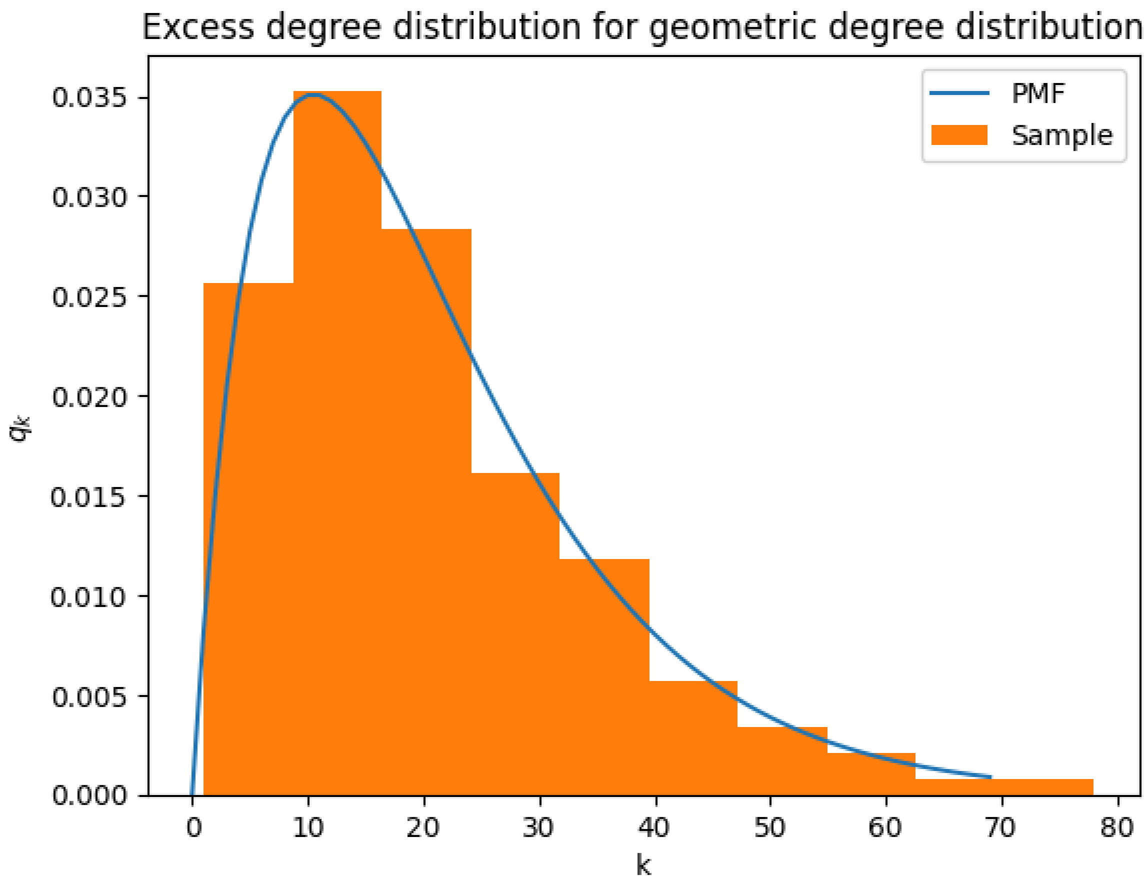
In [ ]:
```
poisson_lambda_range = np.arange(10,1000,10)
p_mean_array=[]
p_excess_mean_array=[]

for lam in poisson_lambda_range:
```

```
    p_mean, var = scistats.poisson.stats(mu=lam, moments='mv')
    p_mean_array.append(p_mean)

    p_excess_mean_array.append(scistats.poisson.expect(lambda k: k**2/p_mean,(lam,)

plt.plot(poisson_lambda_range,p_mean_array,label='degree')
plt.plot(poisson_lambda_range,p_excess_mean_array,label='excess degree')
plt.title('Degree and Excess degree mean against poisson degree distribution varian
plt.xlabel('Variance of degree distribution')
plt.ylabel('Mean')
plt.legend()
plt.show()
```



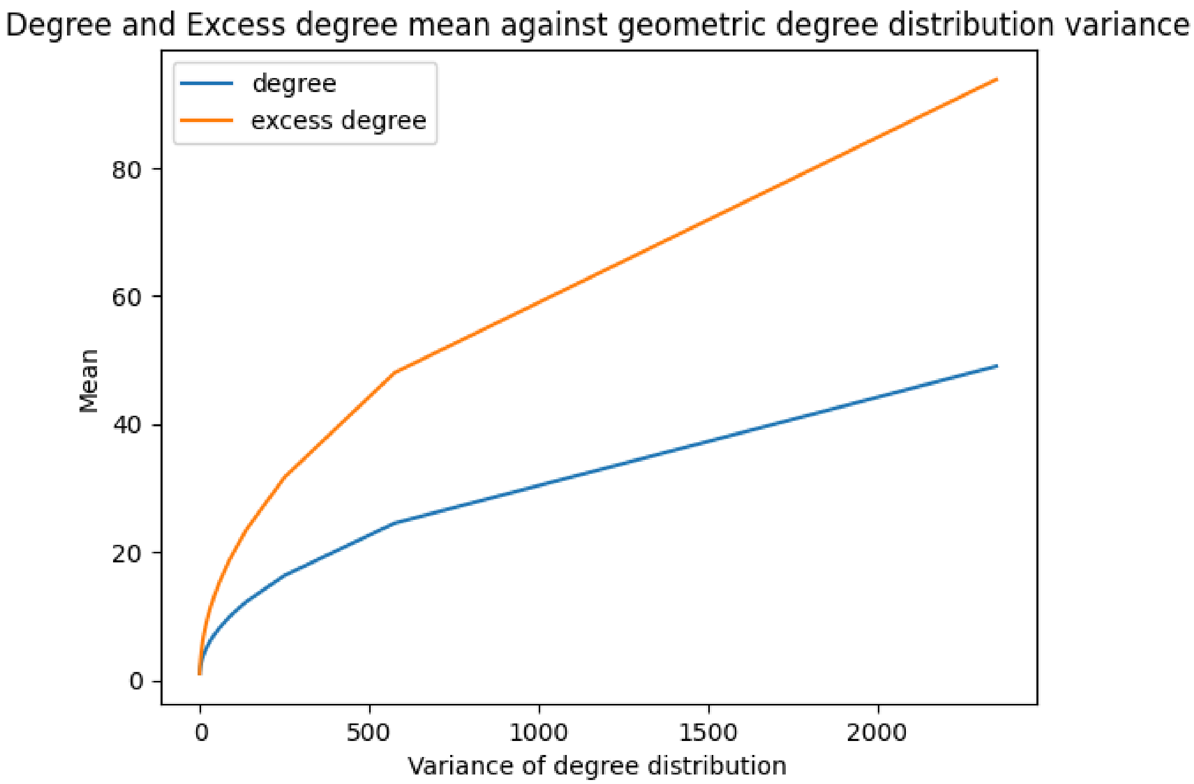
In [ ]:
```
mn = 10
n = 10000
k_range = [k for k in range(70)]

mean, var = scistats.geom.stats(p = 1/(mn+1), moments='mv')

geom_excess_pmf = [k*p/mean for k,p in zip(k_range,scistats.geom.pmf(k_range, p=1/(
```

In [ ]:
```
plt.plot(k_range,geom_excess_pmf,label='PMF')
plt.hist(Q4_geom,density=True,label='Sample')
plt.title('Excess degree distribution for geometric degree distribution')
plt.xlabel('k')
plt.ylabel(r'$q_{k}$')
plt.legend()
plt.show()
```



In [ ]:
```
geom_p_range = np.linspace(0,1,50)
g_mean_array=[]
g_var_array=[]
g_excess_mean_array=[]

for p in geom_p_range:

    g_mean, var = scistats.geom.stats(p=p, moments='mv')
    g_mean_array.append(g_mean)
    g_var_array.append(var)

    g_excess_mean_array.append(scistats.geom.expect(lambda k: k**2/g_mean,(p,)))

plt.plot(g_var_array,g_mean_array,label='degree')
plt.plot(g_var_array,g_excess_mean_array,label='excess degree')
plt.title('Degree and Excess degree mean against geometric degree distribution vari
plt.xlabel('Variance of degree distribution')
plt.ylabel('Mean')
plt.legend()
plt.show()
```

```
c:\Users\Hsin\AppData\Local\Programs\Python\Python39\lib\site-packages\scipy\stats\_
distn_infrastructure.py:3814: RuntimeWarning: expect(): sum did not converge
  warnings.warn('expect(): sum did not converge', RuntimeWarning)
c:\Users\Hsin\AppData\Local\Programs\Python\Python39\lib\site-packages\scipy\stats\_
discrete_distns.py:446: RuntimeWarning: divide by zero encountered in divide
  g1 = (2.0-p) / sqrt(qr)
c:\Users\Hsin\AppData\Local\Programs\Python\Python39\lib\site-packages\scipy\stats\_
discrete_distns.py:447: RuntimeWarning: divide by zero encountered in divide
  g2 = np.polyval([1, -6, 6], p)/(1.0-p)
c:\Users\Hsin\AppData\Local\Programs\Python\Python39\lib\site-packages\scipy\stats\_
discrete_distns.py:438: RuntimeWarning: divide by zero encountered in log1p
  vals = ceil(log1p(-q) / log1p(-p))
c:\Users\Hsin\AppData\Local\Programs\Python\Python39\lib\site-packages\scipy\stats\_
discrete_distns.py:428: RuntimeWarning: divide by zero encountered in log1p
  return -expm1(log1p(-p)*k)
```



Q5

Poisson

$$\sum_k z^k q_k = ze^{\lambda(z-1)}$$

Mean of excess degree = λ Mean of degree = λ

Geometric

$$\sum_k z^k q_k = \frac{zp^2}{(1-(1-p)z)^2}$$

Mean of excess degree = $\frac{(2-p)}{p}$ Mean of degree = $\frac{1-p}{p}$

Showing that mean of excess degree is always > mean of degree

Relate the above to the friendship paradox
https://www.stat.auckland.ac.nz/~fewster/325/notes/ch4.pdf

Q6

Generate Poisson/Geometric graphs with mean degrees between [0,2]. Investigate the size of the component of an arbitrary node. Prove that the component size starts to grow earlier for the geometric degree distribution than for the poisson distribution.

The current expectation is that large components start to present themself when the mean excess degree is >1.

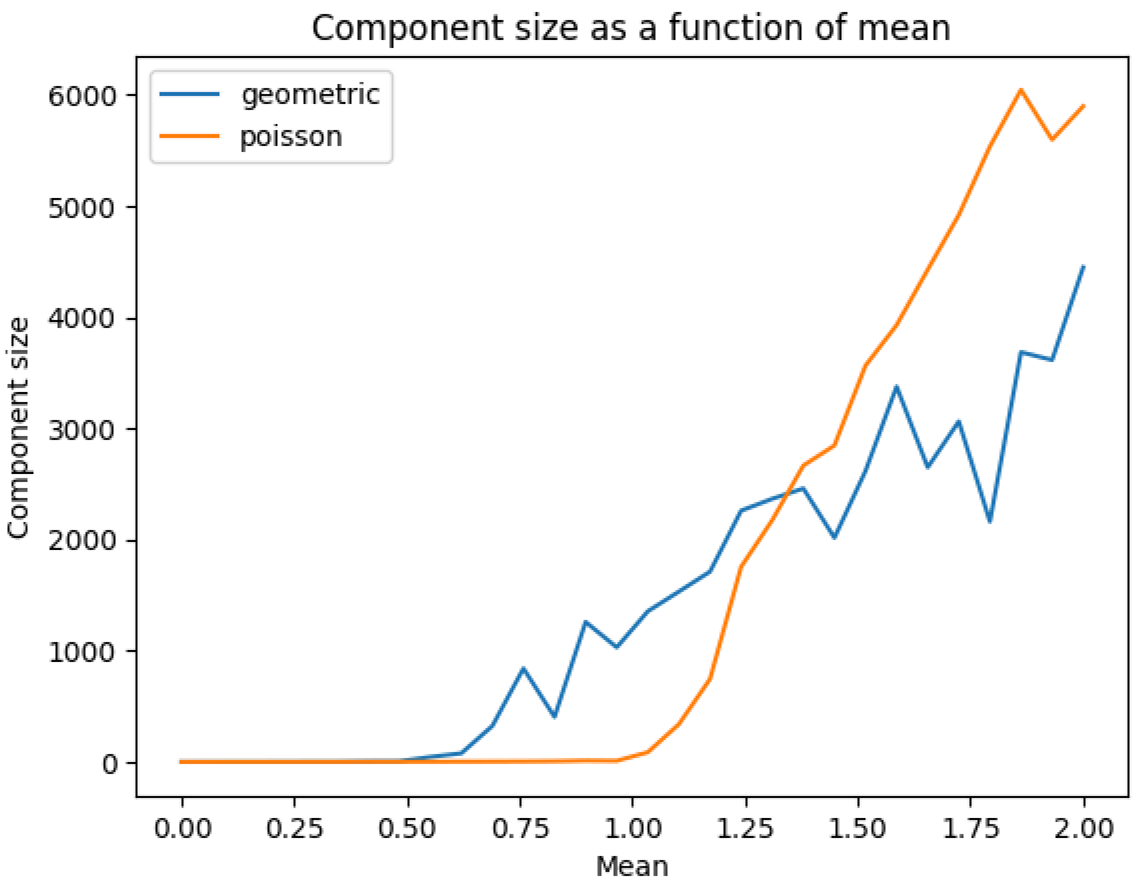
In [ ]:
```
def BFS(node,network: Network) -> int:
    q = deque([node])
    visited = set()
    count = 1

    while q:
        curr = q.popleft()

        for adj in network.neighbors(curr):
            if adj not in visited:
                count+=1
                visited.add(adj)
                q.append(adj)

    return count
```

In [ ]:
```
runs = 50

n=10000
means = np.linspace(0,2,30)
p_comp_size_array = []

for mn in means:
    comp_size = 0
    for _ in range(runs):
        p_network = Poisson_Configuration_Network(n,mn)
        comp_size+=BFS(np.random.randint(0,n),p_network)

    p_comp_size_array.append(comp_size/runs)
```

In [ ]:
```
runs = 50

n = 10000
```

```
g_comp_size_array = []

for mn in means:
    comp_size = 0
    for _ in range(runs):
        g_network = Geometric_Configuration_Network(n,mn)
        comp_size += BFS(np.random.randint(0,n),g_network)

    g_comp_size_array.append(comp_size/runs)
```

In [ ]:
```
plt.plot(means,g_comp_size_array,label='geometric')
plt.plot(means,p_comp_size_array,label='poisson')
plt.xlabel('Mean')
plt.ylabel('Component size')
plt.title('Component size as a function of mean')
plt.legend()
plt.show()
```



Q7

The current assumption is that the degrees of neighbours are uncorrelated (Saying q_k is independent of the other q_k of the same node?). In reality, a node with high degree is more likely to have an excess degree of negative skew ?

Simulation :

Generate a graph. Choose set of nodes. Record excess degree of it's neighbours. Plot function of x = k , y = skew of excess degree

Analytical :

Show that given k, q_k is a function of k?

In [ ]:
```
runs = 5
n = 10000
lam = 10
p_ks = []
p_skews = []
p_ex_means = []
p_means=[]

for _ in range(runs):
    p_network = Poisson_Configuration_Network(n,lam)

    nodes = np.random.randint(0,n,50)

    for node in nodes:

        while (len(p_network.neighbors(node))==0):
            node = np.random.randint(0,n)
            p_means.append(0)

        friend_k_array = []

        for adj in p_network.neighbors(node):
            friend_k_array.append(len(p_network.neighbors(adj)))

        p_ks.append(len(p_network.neighbors(node)))
        p_means.append(len(p_network.neighbors(node)))
        p_skews.append(scistats.skew(friend_k_array))
        p_ex_means.append(np.mean(friend_k_array))
```

In [ ]:
```
runs = 5
n = 10000
mn = 10
g_ks = []
g_skews = []
g_means=[]
g_ex_means = []

for _ in range(runs):
    g_network = Geometric_Configuration_Network(n,mn)

    nodes = np.random.randint(0,n,50)

    for node in nodes:

        while (len(g_network.neighbors(node))==0):
            node = np.random.randint(0,n)
            g_means.append(0)
```

```
        friend_k_array = []

        for adj in g_network.neighbors(node):
            friend_k_array.append(len(g_network.neighbors(adj)))

        g_ks.append(len(g_network.neighbors(node)))
        g_means.append(len(g_network.neighbors(node)))
        g_skews.append(scistats.skew(friend_k_array))
        g_ex_means.append(np.mean(friend_k_array))
```

```
C:\Users\Hsin\AppData\Local\Temp\ipykernel_13796\3817351467.py:27: RuntimeWarning: P
recision loss occurred in moment calculation due to catastrophic cancellation. This
occurs when the data are nearly identical. Results may be unreliable.
  g_skews.append(scistats.skew(friend_k_array))
```

In [ ]:

For a configuration network, the degree distributions are indeed uncorrelated.

We now explore the skew as a function of k. By allowing a 'popular' person to more likely be friends with other 'popular' people.

We also inspect mean of degree and mean of excess degree for such a network each for a Poisson and Geometric Degree distribution.

We form the 'biased' configuration network by first generating the degree distribution as usual. Then split the list of nodes into a 'popular' half and 'loner' half. Then we pair the nodes in each half. Finally, we perform a slight shuffle to the pairing.

In [ ]:
```
def split(arr):
    freq = Counter(arr)
    sorted_nodes = sorted(freq.items(),key= lambda item:item[1])

    loner = np.array([node for node,f in sorted_nodes[:len(sorted_nodes)//2] for _
    popular = np.array([node for node,f in sorted_nodes[len(sorted_nodes)//2:] for

    return popular,loner
```

In [ ]:
```
def shuffle(arr: np.array, bias):
    times = int(bias*arr.shape[0])

    for _ in range(times):
        i,j = np.random.randint(0,arr.shape[0],size=2)

        arr[i][0], arr[j][0] = arr[j][0], arr[i][0]

    return arr
```

In [ ]:
```
class Biased_Poisson_Config_Network(Network):

    def __init__(self, num_nodes,mn,bias=0.3):
        super().__init__(num_nodes)
```

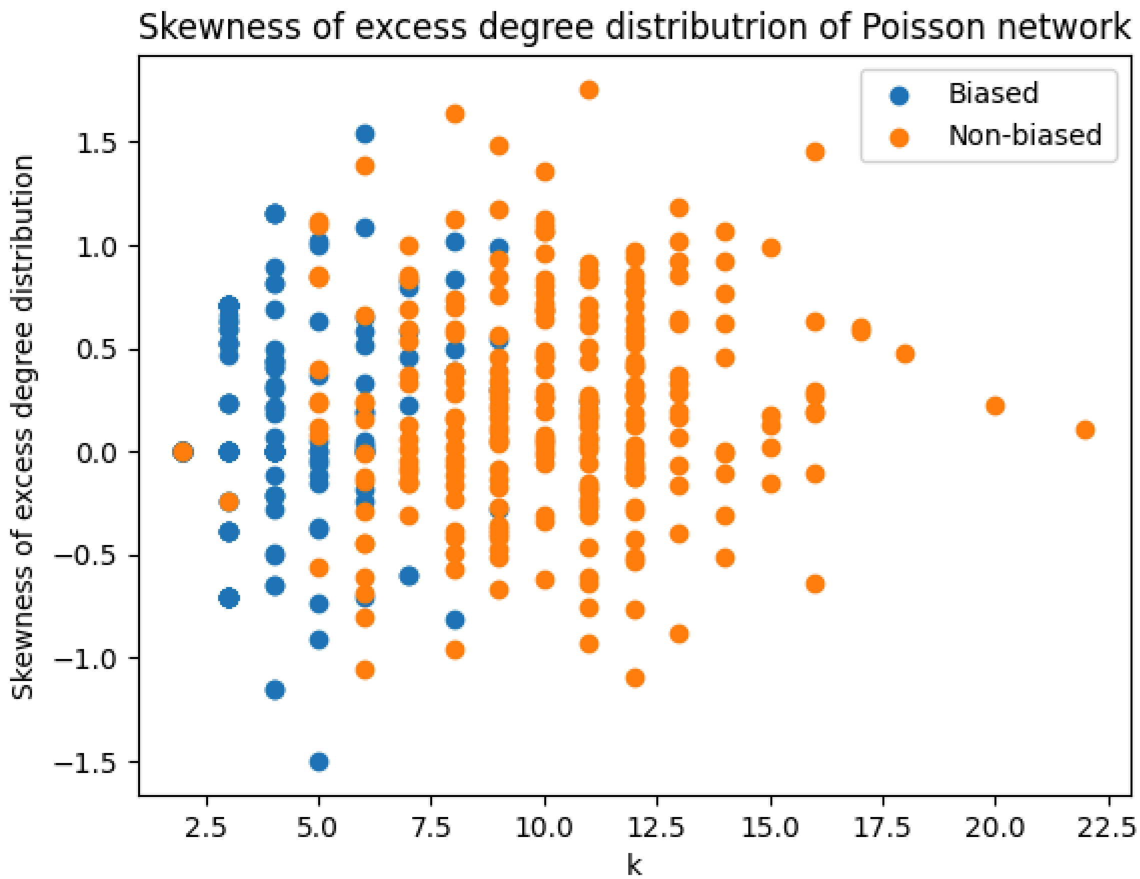
```
        k = np.random.poisson(lam=mn,size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i])])
        S = np.random.permutation(S)

        popular, loner = split(S)

        if len(popular)%2:
            popular = popular[:-1]

        popular = popular.reshape(-1,2)

        if len(loner)%2:
            loner = loner[:-1]

        loner = loner.reshape(-1,2)

        S = np.concatenate((popular,loner),axis=0)

        S = shuffle(S,bias)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)
```

In [ ]:
```
class Biased_Geometric_Config_Network(Network):

    def __init__(self, num_nodes,mn,bias=0.3):
        super().__init__(num_nodes)

        k = np.random.geometric(1/(mn+1), size=num_nodes)

        S = np.array([i for i in range(num_nodes) for _ in range(k[i]-1)])
        S = np.random.permutation(S)

        popular, loner = split(S)

        if len(popular)%2:
            popular = popular[:-1]

        popular = popular.reshape(-1,2)

        if len(loner)%2:
            loner = loner[:-1]

        loner = loner.reshape(-1,2)

        S = np.concatenate((popular,loner),axis=0)

        S = shuffle(S,bias)

        for i,j in S:
            if i!=j:
                self.add_edge(i,j)
```

In [ ]:
```
runs = 5
n = 10000
mn = 10
p_b_ks = []
p_b_skews = []
p_b_means = []
p_b_ex_means = []


for _ in range(runs):
    p_bias_network = Biased_Poisson_Config_Network(n,mn,bias=0.2)

    nodes = np.random.randint(0,n,50)

    for node in nodes:

        while (len(p_bias_network.neighbors(node))==0):
            node = np.random.randint(0,n)
            p_b_means.append(0)

        friend_k_array = []

        for adj in p_bias_network.neighbors(node):
            friend_k_array.append(len(p_bias_network.neighbors(adj)))

        p_b_ks.append(len(p_bias_network.neighbors(node)))
        p_b_means.append(len(p_bias_network.neighbors(node)))
        p_b_skews.append(scistats.skew(friend_k_array))
        p_b_ex_means.append(np.mean(friend_k_array))
```

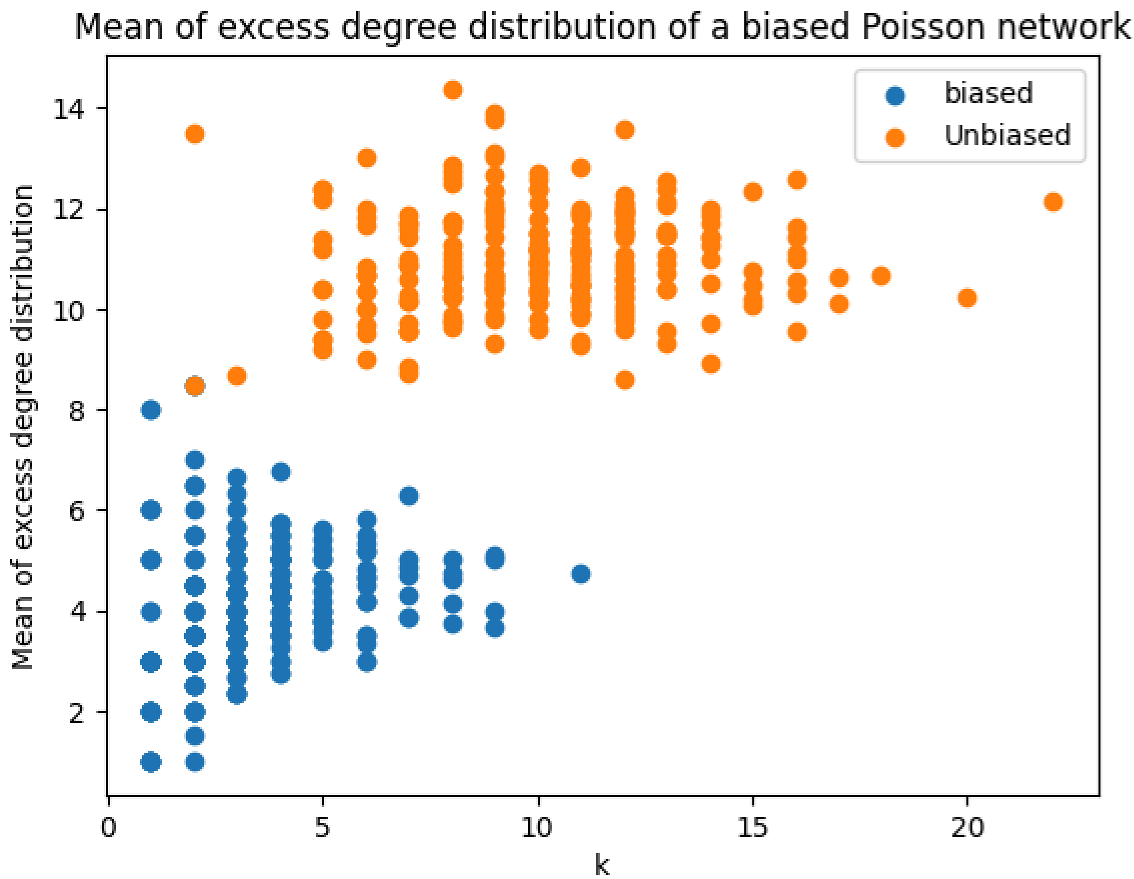
```
C:\Users\Hsin\AppData\Local\Temp\ipykernel_13796\3486855537.py:28: RuntimeWarning: P
recision loss occurred in moment calculation due to catastrophic cancellation. This
occurs when the data are nearly identical. Results may be unreliable.
  p_b_skews.append(scistats.skew(friend_k_array))
```

In [ ]:
```
plt.scatter(p_b_ks,p_b_skews, label='Biased')
plt.scatter(p_ks,p_skews, label='Non-biased')
plt.title('Skewness of excess degree distributrion of Poisson network')
plt.xlabel('k')
plt.ylabel('Skewness of excess degree distribution')
plt.legend()
plt.show()
```



In [ ]:

```
plt.scatter(p_b_ks,p_b_ex_means, label='biased')
plt.scatter(p_ks,p_ex_means, label='Unbiased')
plt.title('Mean of excess degree distribution of a biased Poisson network')
plt.xlabel('k')
plt.ylabel('Mean of excess degree distribution')
plt.legend()
plt.show()
```



In [ ]:
```
print('Mean degree of node (Biased)= {}'.format(np.mean(p_b_means)))
print('Mean degree of node (Unbiased) = {}'.format(np.mean(p_means)))
print('Correlation of k and mean degree dist biased Poisson = {}'.format(scistats.p
print('Correlation of k and mean degree dist unbiased Poisson = {}'.format(scistats
```

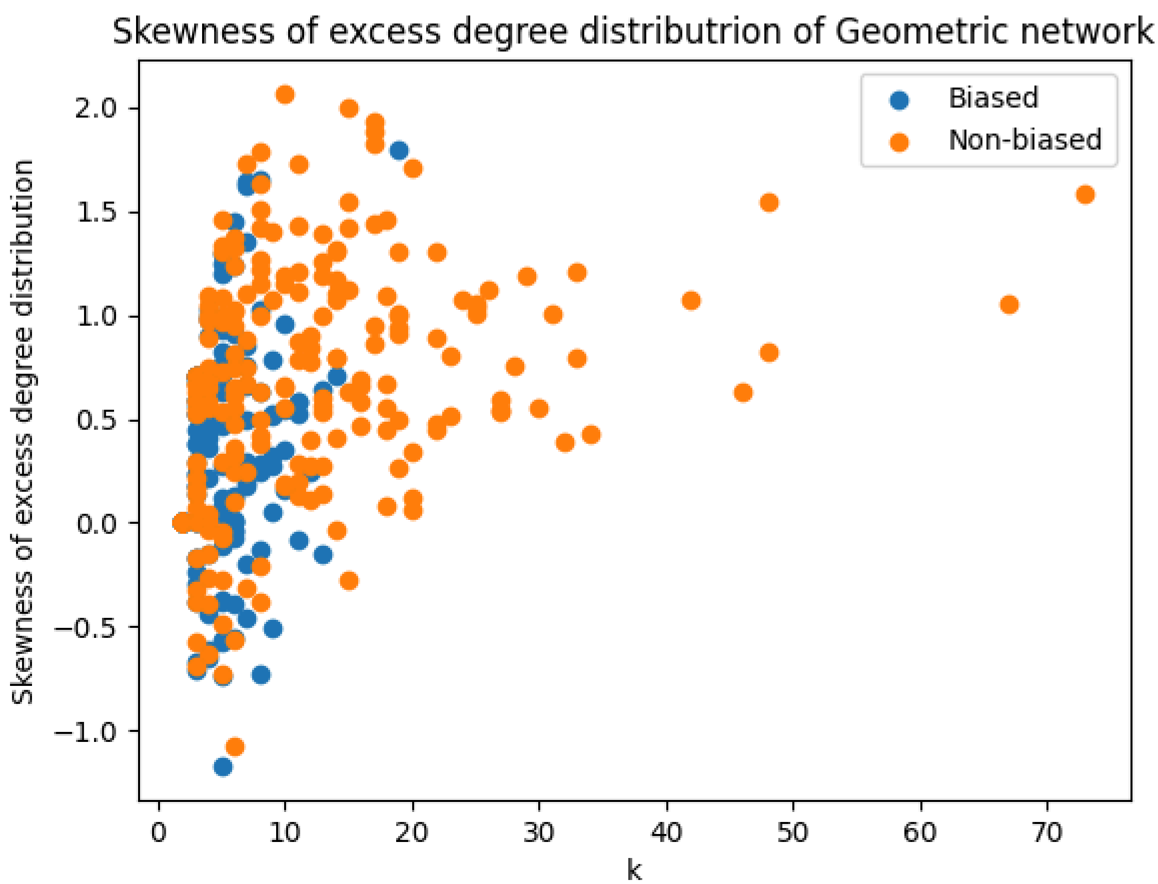
```
Mean degree of node (Biased)= 3.2384615384615385
Mean degree of node (Unbiased) = 10.172
Correlation of k and mean degree dist biased Poisson = 0.21252772351465735
Correlation of k and mean degree dist unbiased Poisson = 0.07604427183609698
```

In [ ]:
```
runs = 5
n = 10000
mn = 10
g_b_ks = []
g_b_skews = []
g_b_means = []
g_b_ex_means = []


for _ in range(runs):
    g_bias_network = Biased_Geometric_Config_Network(n,mn,bias=0.2)

    nodes = np.random.randint(0,n,50)

    for node in nodes:

        while (len(g_bias_network.neighbors(node))==0):
            node = np.random.randint(0,n)
```

```
            g_b_means.append(0)

        friend_k_array = []

        for adj in g_bias_network.neighbors(node):
            friend_k_array.append(len(g_bias_network.neighbors(adj)))

        g_b_ks.append(len(g_bias_network.neighbors(node)))
        g_b_means.append(len(g_bias_network.neighbors(node)))
        g_b_skews.append(scistats.skew(friend_k_array))
        g_b_ex_means.append(np.mean(friend_k_array))
```

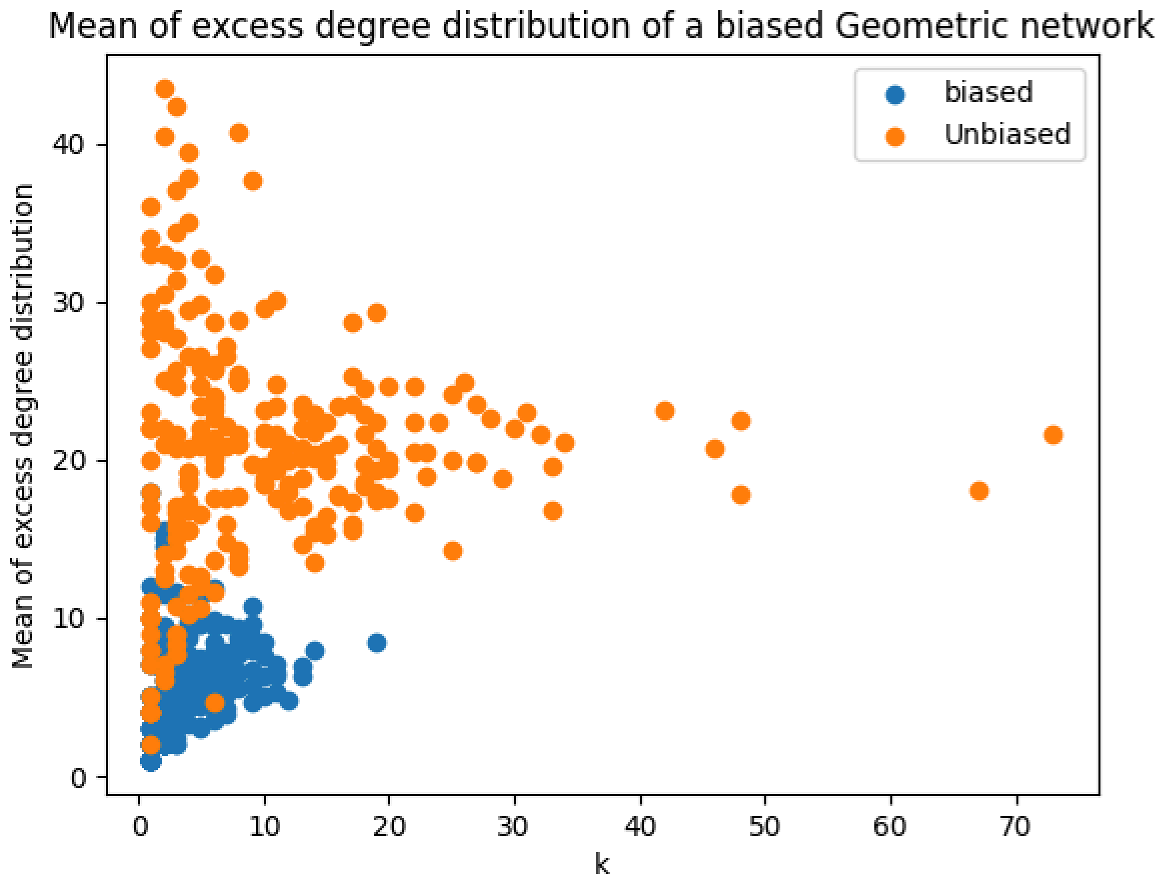
```
C:\Users\Hsin\AppData\Local\Temp\ipykernel_13796\3754050162.py:28: RuntimeWarning: P
recision loss occurred in moment calculation due to catastrophic cancellation. This
occurs when the data are nearly identical. Results may be unreliable.
  g_b_skews.append(scistats.skew(friend_k_array))
```

In [ ]:
```
plt.scatter(g_b_ks,g_b_skews, label='Biased')
plt.scatter(g_ks,g_skews, label='Non-biased')
plt.title('Skewness of excess degree distributrion of Geometric network')
plt.xlabel('k')
plt.ylabel('Skewness of excess degree distribution')
plt.legend()
plt.show()
```



In [ ]:
```
plt.scatter(g_b_ks,g_b_ex_means, label='biased')
plt.scatter(g_ks,g_ex_means, label='Unbiased')
plt.title('Mean of excess degree distribution of a biased Geometric network')
plt.xlabel('k')
```

```
plt.ylabel('Mean of excess degree distribution')
plt.legend()
plt.show()
```



In [ ]:
```
print('Mean degree of node (Biased)= {}'.format(np.mean(g_b_means)))
print('Mean degree of node (Unbiased) = {}'.format(np.mean(g_means)))
print('Correlation of k and mean degree of biased Geom = {}'.format(scistats.pearso
print('Correlation of k and mean degree of unbiased Geom = {}'.format(scistats.pear
```

```
Mean degree of node (Biased)= 3.2202797202797204
Mean degree of node (Unbiased) = 9.309090909090909
Correlation of k and mean degree of biased Geom = 0.3950830236281581
Correlation of k and mean degree of unbiased Geom = 0.03975656428102818
```

In [ ]: