protein-network

February 1, 2021

```
[1]: import networkx as nx
     import numpy as np
     import matplotlib.pyplot as plt
     import collections
```

1 Protein Network Data Analysis

- Undirected network
- N=2,018 proteins as nodes
- L=2,930 binding interactions as links.
- Average degree =2.90

```
[146]: G = nx.read_edgelist('data/protein.edgelist.txt', create_using=nx.Graph(),␣
        ↪delimiter ='\t', nodetype=int, encoding = 'utf-8')
```

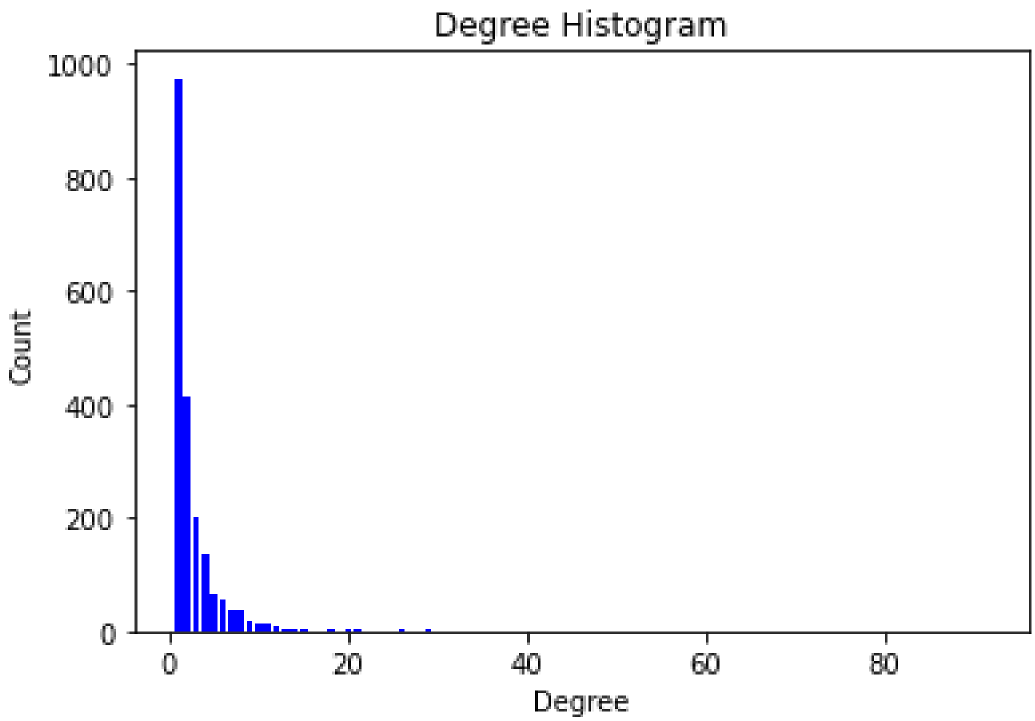
```
[147]: degrees = [G.degree[node] for node in G]
       N = len(G)
       L = G.size()

       kmin = np.min(degrees)
       kmax = np.max(degrees)
       kavg = np.mean(degrees)

       print("N=", N)
       print("L=", L)

       print("Average degree=", kavg)
       print("Min degree=", kmin)
       print("Max degree=", kmax)
```

```
N= 2018
L= 2930
Average degree= 2.9038652130822595
Min degree= 1
Max degree= 91
```

=> avg degree close to the min degree, max degree much higher than min/mean (skewed degree distribution) => first indication the network could be a power-low (or some heavy-tailed distribution)

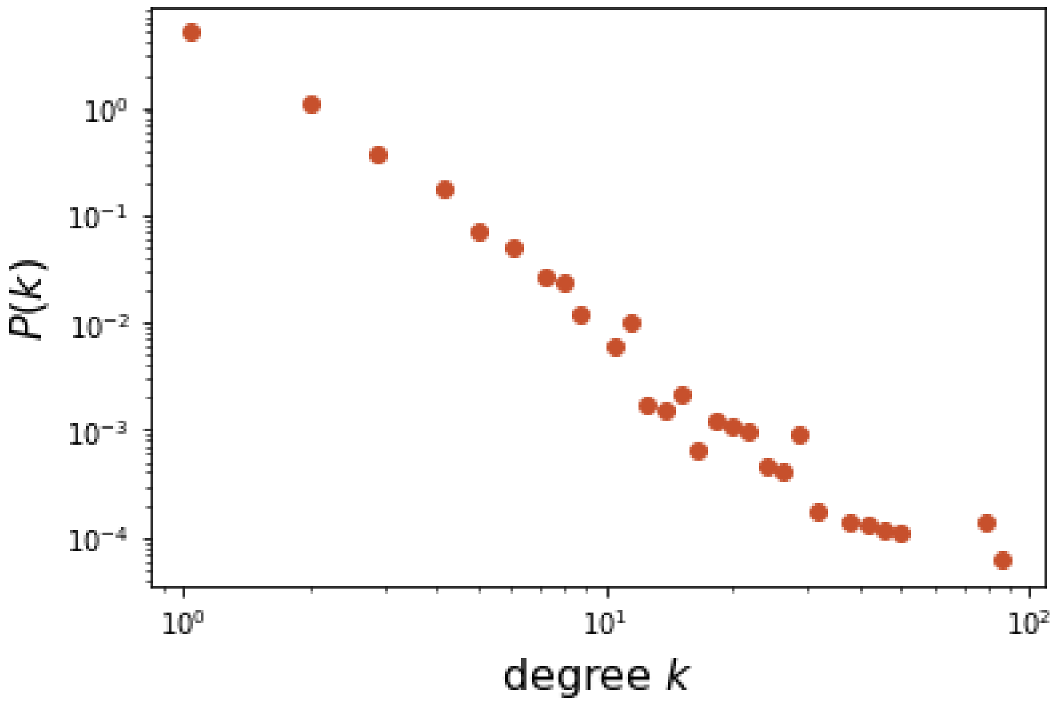
1.1 Degree distribution

[10]:
```
degree_sequence = sorted([d for n, d in G.degree()], reverse=True)  # degree␣
↪sequence
degreeCount = collections.Counter(degree_sequence)
deg, cnt = zip(*degreeCount.items())

plt.bar(deg, cnt, width=0.80, color="b")
plt.title("Degree Histogram")
plt.ylabel("Count")
plt.xlabel("Degree")
plt.show()
```



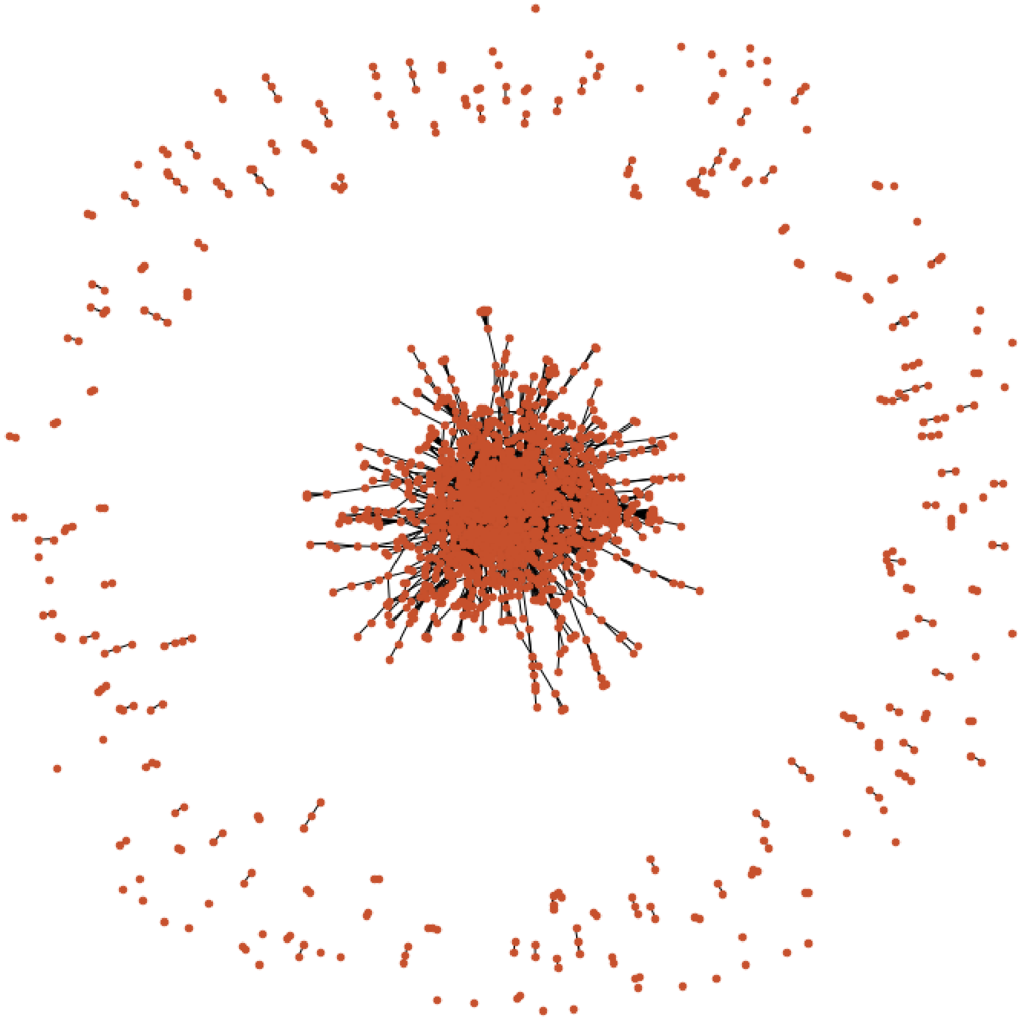
[20]:
```
bin_edges = np.logspace(np.log10(kmin), np.log10(kmax), num=50)
density, _ = np.histogram(degrees, bins=bin_edges, density=True)

fig = plt.figure(figsize=(6,4))
log_be = np.log10(bin_edges)
x = 10**((log_be[1:] + log_be[:-1])/2)

plt.loglog(x, density, marker='o', linestyle='none', color='#c7502c')
plt.xlabel(r"degree $k$", fontsize=16)
plt.ylabel(r"$P(k)$", fontsize=16)
```

```
plt.show()
```



1.2 Visualization

[21]:
```
fig=plt.figure(figsize=(10,10))
nx.draw_spring(G, node_size=20, node_color="#c7502c")
```



1.3 Connected components

[16]:
```
nx.is_connected(G)
```

[16]:
```
False
```

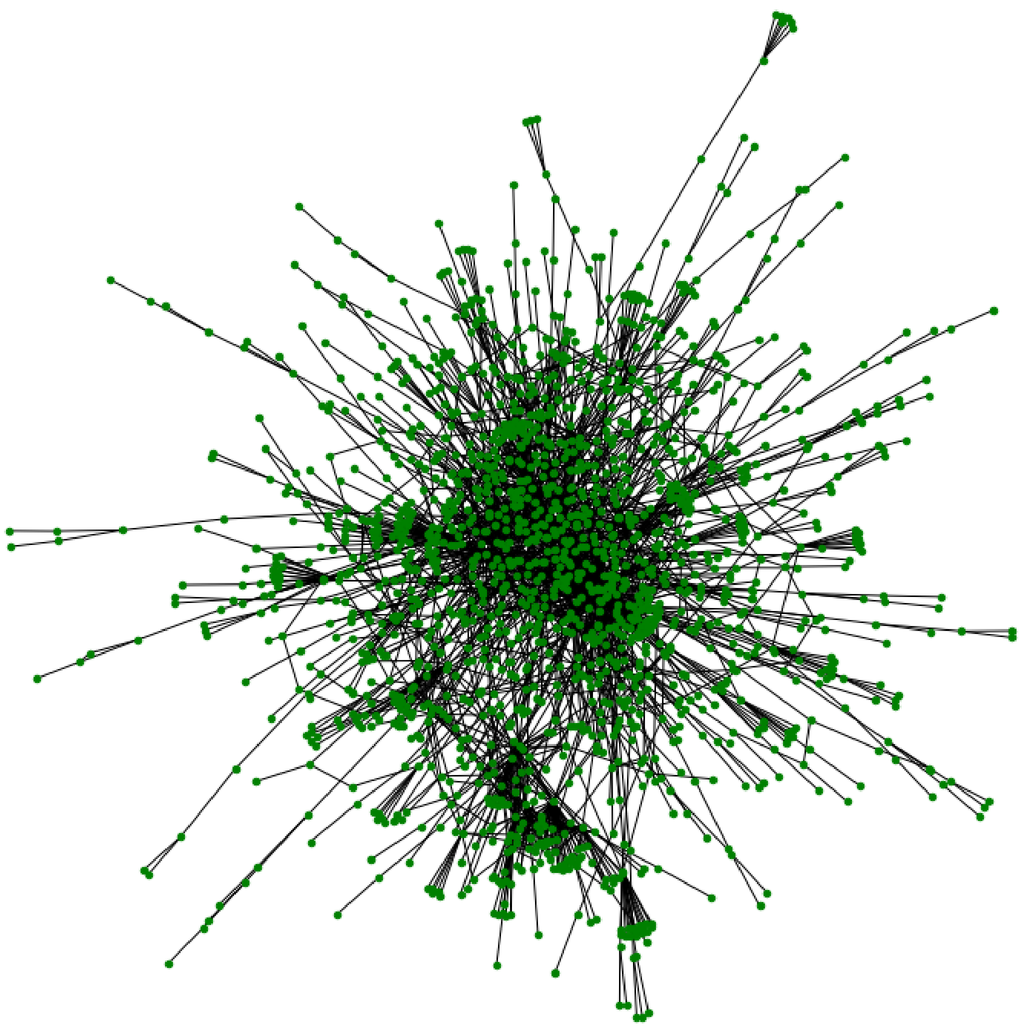
[28]:
```
sorted_components = sorted(nx.connected_components(G), key=len, reverse=True)
print("There are ", nx.number_connected_components(G), ", the largest one␣
↪(giant component) has ", len(sorted_components[0]), "nodes.")
```

```
There are  185 , the largest one (giant component) has  1647 nodes.
```

[30]:
```
G0 = G.subgraph(sorted_components[0])
print(nx.info(G0))
```

```
Name:
Type: Graph
Number of nodes: 1647
Number of edges: 2682
Average degree:   3.2568
```

[32]:
```
fig=plt.figure(figsize=(10,10))
nx.draw_spring(G0, node_size=20, node_color="green")
```



1.4 Paths and Distances

[34]:
```
# diameter can not be determined for G since it is not connected, but can be␣
↪done for G0
diameter = nx.diameter(G0) #largest shortest distance (between the two most␣
↪distant nodes)
print("Diameter of giant component =", diameter)
```

```
Diameter of giant component = 14
```

[37]:
```
# same: graph is not connected, so can do this for G0 only
avg_dist = nx.average_shortest_path_length(G0)
print("Average distance in giant component = ", avg_dist) # compared to N =␣
↪1647, the distance is small
```

```
Average distance in giant component =  5.611747416599716
```

[157]:
```
N0 = len(G0)
print(N0)
```

```
1647
```

[168]:
```
D = np.zeros(shape=(N,N)) # D is the matrix of distances
vl = []
for node1 in G0.nodes():
    for node2 in G0.nodes():
        if (node1 != node2) and D[node1][node2] == 0:
            aux = nx.shortest_path(G0, node1, node2)
            dij = len(aux)
            D[node1][node2] = dij
            D[node2][node1] = dij
            vl.append(dij)
```

[179]:
```
len(vl)
```

[179]: 1355481

[181]:
```
N0*(N0-1)//2
```

[181]: 1355481

[182]:
```
d = {}
for elem in vl:
    if elem in d:
        d[elem] += 1
    else:
        d[elem] = 1
```

[183]:
```
d
```

[183]: {2: 2518, 8: 200419, 7: 317798, 6: 330082, 5: 226215, 4: 88834, 9: 101109, 10: 43935, 11: 16183, 3: 23013, 12: 4228, 13: 966, 14: 171, 15: 10}

[186]:
```
# the above can also be obtained as follows:
distCount = collections.Counter(vl)
dist, cntDist = zip(*distCount.items())
print(dist)
print(cntDist)
```

```
(2, 8, 7, 6, 5, 4, 9, 10, 11, 3, 12, 13, 14, 15)
(2518, 200419, 317798, 330082, 226215, 88834, 101109, 43935, 16183, 23013, 4228,
966, 171, 10)
```

[195]:
```
distances = sorted(d.keys())
pdistances = [d[i]/N0 for i in distances]
```

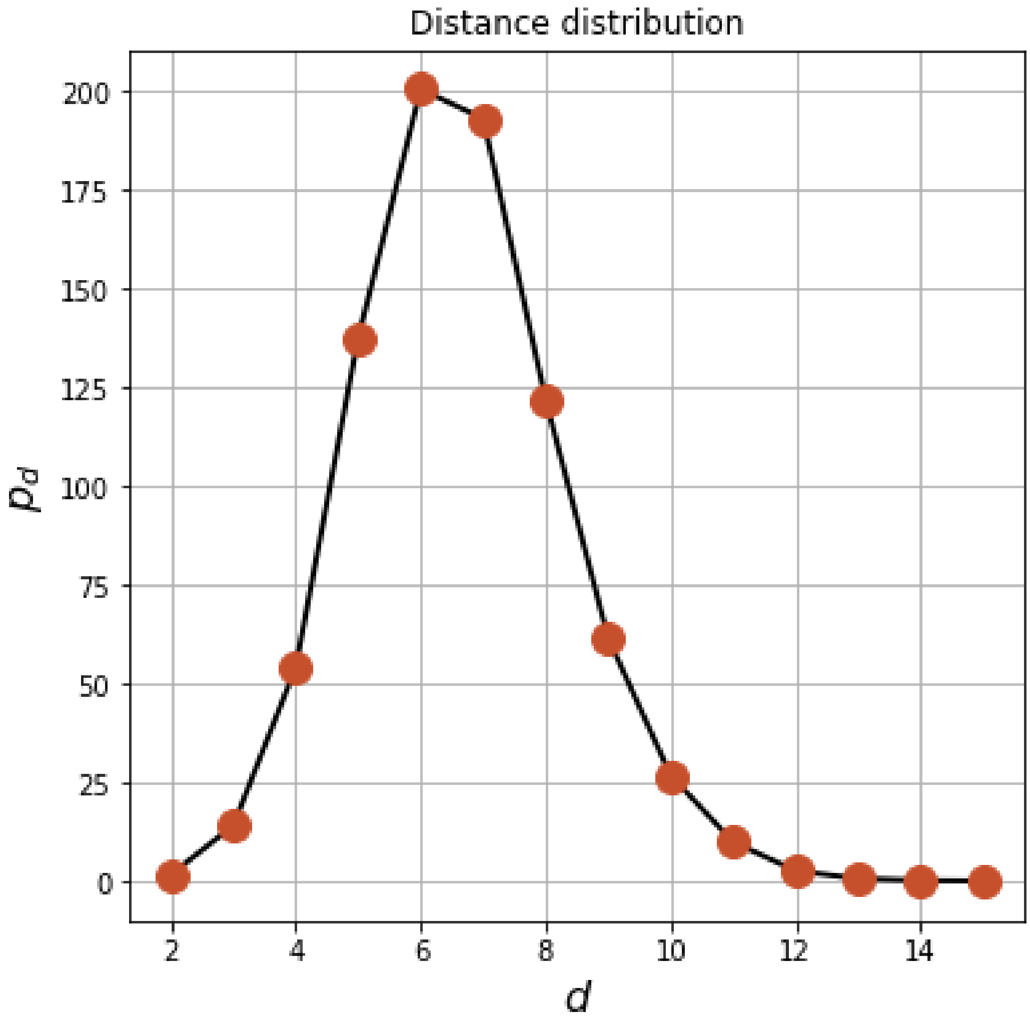
[192]:
```
distances
```

[192]: [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[196]:
```
pdistances
```

[196]: [1.5288403157255617, 13.972677595628415, 53.936854887674556, 137.34972677595627, 200.4140862173649, 192.95567698846386, 121.68731026108075, 61.38979963570127, 26.67577413479053, 9.82574377656345, 2.5670916818457803, 0.5865209471766849, 0.10382513661202186, 0.006071645415907711]

[240]:
```
plt.figure(figsize=(6,6))
plt.plot(distances, pdistances, linestyle="solid", linewidth=2, color="black")
plt.plot(distances, pdistances, "o", color='#c7502c', markersize=12)
#plt.axvline(x=avg_dist, color="black", linestyle="dashed")
plt.xlabel(r"$d$", fontsize=16)
plt.ylabel(r"$p_d$", fontsize=16)
plt.title("Distance distribution")
plt.grid(True)
plt.show()
```



1.5 Clustering Coefficient

[40]:
```
density = nx.density(G) #  no of edges / maximum possible edges
print("Edge density in G = ", density)

density0 = nx.density(G0) #  no of edges / maximum possible edges
print("Edge density in G0 = ", density0)
```

```
Edge density in G =  0.0014396951973635397
Edge density in G0 =  0.0019786334150017596
```

[148]:
```
nx.average_clustering(G)
```

[148]: 0.046194001297365166

[144]:
```
avg_cc = nx.average_clustering(G)
print("Avg clustering coefficient:", avg_cc) # is this high? compare it to␣
↪density

avg_cc0 = nx.average_clustering(G0)
print("Avg clustering coefficient:", avg_cc0) # is this high? compare it to␣
↪density
```

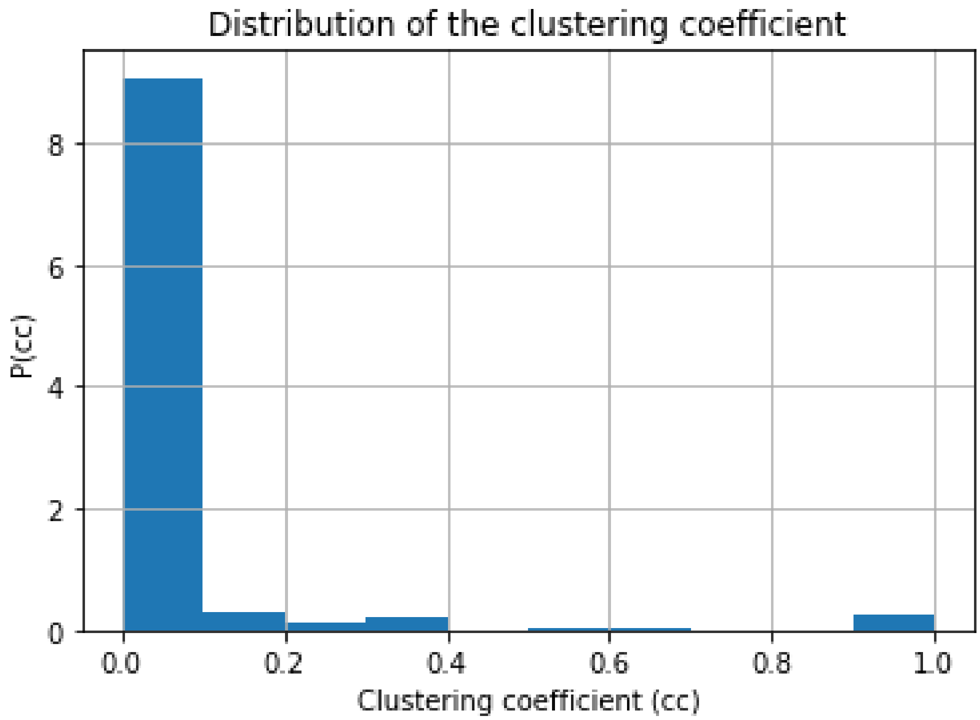
```
Avg clustering coefficient: 0.046194001297365166
Avg clustering coefficient: 0.05659957171711166
```

[42]:
```
cc = nx.clustering(G)
```

The dependence of the average local clustering coefficient on the node's degree, k. The C(k) function is obtained by averaging over the local clustering coefficient of all nodes with the same degree k.

[124]:
```
vcc = []
for n in G.nodes():
    vcc.append(nx.clustering(G, n))

#print(vcc)
vcc = np.array(vcc) # vcc can also be obtained from cc.values()
```

[130]:
```
plt.hist(cc.values(), bins=10, density=True)
plt.grid(True)
plt.title("Distribution of the clustering coefficient")
plt.xlabel("Clustering coefficient (cc)")
plt.ylabel("P(cc)")
plt.show()
```



[133]:
```
ck = []
ks = []
for node_degree in degreeCount.keys():
    ks.append(node_degree) # this will contain the degrees of the node
    #print("Nodes with degree ", node_degree, " are: ")
    nodes = [n for n, d in G.degree() if d == node_degree]
    #print(nodes)
    local_cc = 0
    for n in nodes:
        local_cc += nx.clustering(G, n)
    ck.append(local_cc/len(nodes)) # this is the average clustering coefficient␣
 ↪for all nodes that have degree node_degree
print(ck)
print(len(ck))
print(ks)
print(len(ks))
```

```
[0.008426966292134831, 0.005696202531645570, 0.0030864197530864196,
0.0024489795918367346, 0.007399577167019027, 0.0011614401858304297,
0.006006006006006006, 0.009195402298850575, 0.05291005291005291,
0.012108262108262111, 0.010869565217391304, 0.018115942028985508,
0.007905138339920948, 0.021052631578947368, 0.026928432191590083,
0.0196078431372549, 0.018586601307189542, 0.007352941176470588, 0.0,
0.05653235653235653, 0.010323010323010324, 0.02459207459207459,
```

```
0.026868686868686868, 0.05858585858585859, 0.08088624338624338,
0.06516290726817042, 0.07448979591836734, 0.05306122448979592,
0.07321428571428575, 0.06368159203980099, 0.10049019607843135,
0.09195402298850577, 0.10628019323671498, 0.0]
34
[91, 82, 81, 52, 46, 42, 37, 32, 30, 29, 26, 24, 23, 22, 21, 20, 18, 17, 16, 15,
14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
34
```

[153]:

```python
plt.loglog(ks, ck, 'bo', color='#c7502c')
plt.title("Clustering coefficient according to degree")
plt.ylabel("C(k)")
plt.xlabel("k")
plt.show()
```