Untitled1

April 17, 2017

```
In [1]: import networkx as nx
        import matplotlib.pyplot as plt
        plt.style.use('classic')
        %matplotlib inline
        import seaborn as sb
        import pandas as pd
```

0.1 Reading Some Graphs

```
In [2]: graph1 = nx.read_edgelist('./CA-HepTh.txt', comments="#", delimiter="\t", create_using=n
        graph2 = nx.read_edgelist('./CA-GrQc.txt', comments='#', delimiter='\t', create_using=nx
```

0.2 Node Degree

The node degree corresponds to the amount of neighbors that is linked to a node. It varies according to the two following properties: * Undirected Graph: Is the amount of edges that connect a node k_i to its neighbors. * Directed Graph: Is the amount of incoming and outgoing edges of a Node. Thus, the degree of a node k_i is $k_i^{out} + k_i^{in}$.

To retrieve the degree of each node is quite simple using networkx, the only thing necessary is to call the builtin function `networkx.degree()` in the library. The function receives as argument a graph and as a result returns a dictionary containing the degree of each node.

```
In [3]: degreeGraph1 = pd.DataFrame(pd.Series(nx.degree(graph1)), columns=["Degree"])
        degreeGraph2 = pd.DataFrame(pd.Series(nx.degree(graph2)), columns=["Degree"])
```

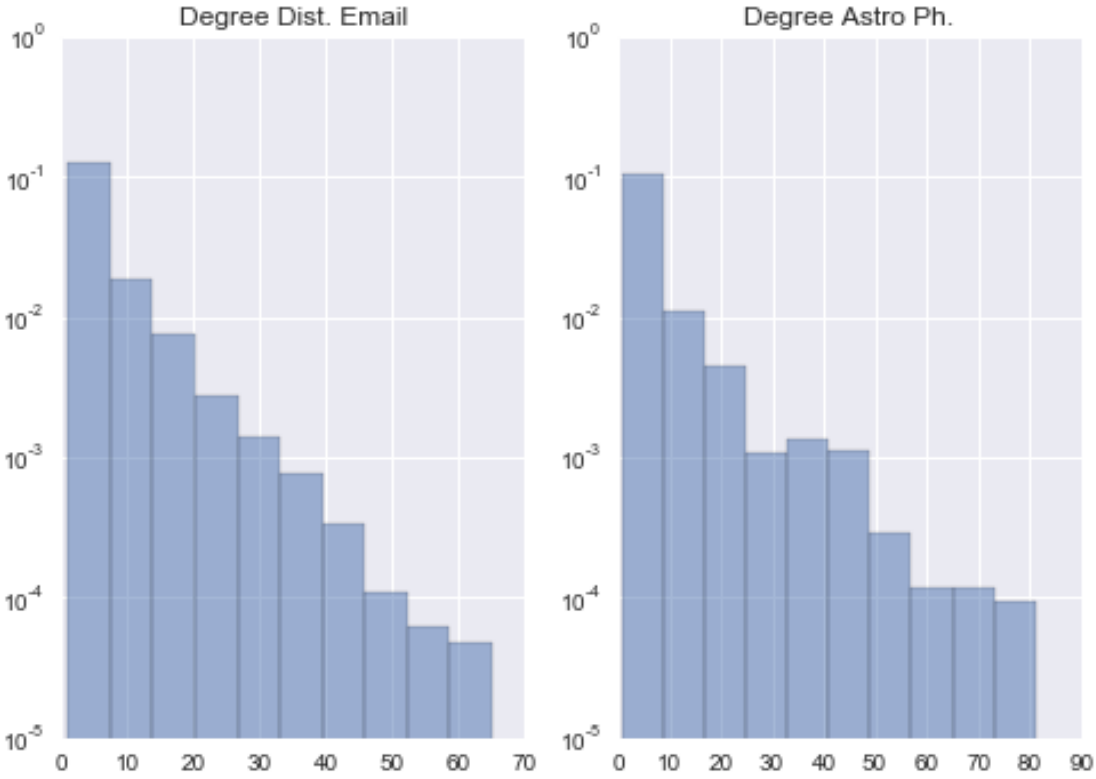
The degree distribution of a node denotes the number of k-degree nodes in a network. The easiest way of showing the distribution is plotting the histogram for the degrees.

As we can see bellow, the node distribution for the Graph1 shows that the higher percentage of nodes have a degree in the range $[0, 10]$. The degree seems to uniformly drops until the degree in the range $[59, 65]$ of nodes with degree in the range $[1250, 1400]$.

In contrast to the First Graph, the distribution for the Astro Ph. graph shows that the higher percentage of nodes have a degree in the range $[0, 50]$.

```
In [6]: fig, ax = plt.subplots(nrows=1, ncols=2)
        ax[0].hist(degreeGraph1['Degree'], alpha=0.5, normed=True, stacked=True, log=True)
        ax[0].set_title('Degree Dist. Email')
        ax[1].hist(degreeGraph2['Degree'], alpha=0.5, normed=True, stacked=True, log=True)
        ax[1].set_title('Degree Astro Ph.')
```

Out[6]: <matplotlib.text.Text at 0x7f664fb1ce48>



0.3 Clustering Coefficients

The clustering coefficient captures the degree to which the neighbors of a given node link to each other. In other words, for unweighted graphs, the degree of a node u is the fraction of possible triangles through that node that exist. In `networkx`, the following equations is used:

$c_u = \frac{2T(u)}{deg(u)(deg(u)-1)}$

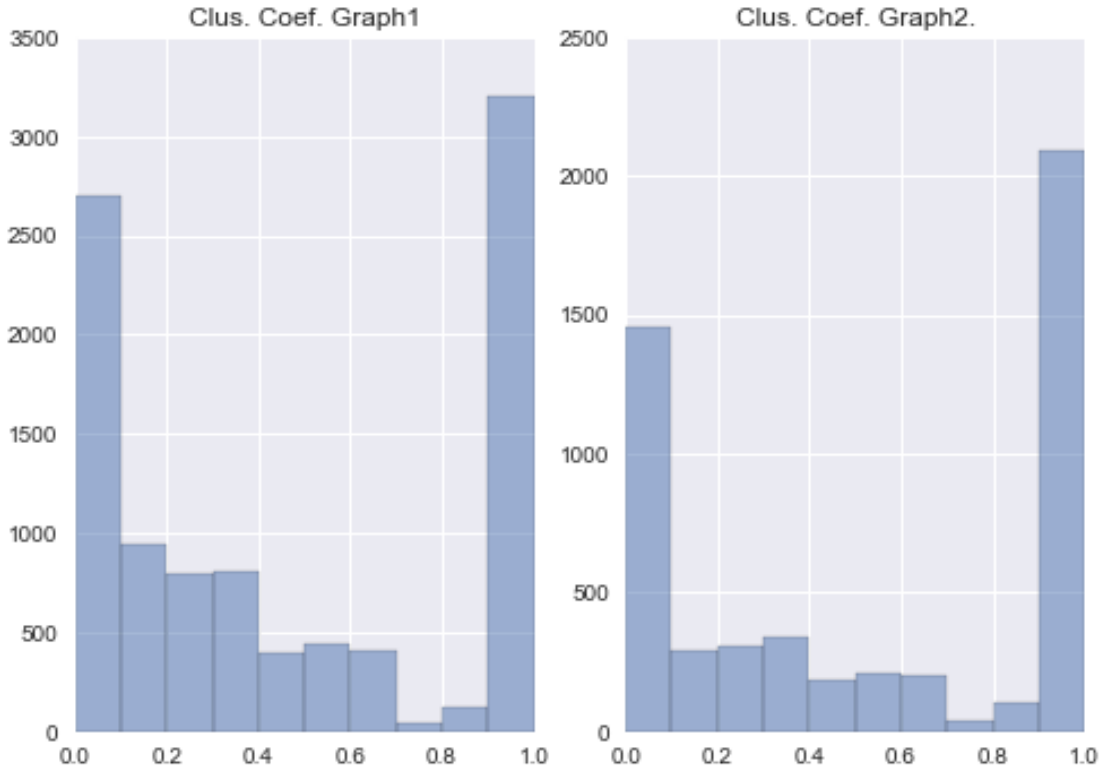
where $T(u)$ is the number of triangles through

```
In [7]: clusteringGraph1 = nx.clustering(graph1)
        clusteringGraph2 = nx.clustering(graph2)
```

```
In [8]: clusteringGraph1 = pd.DataFrame(pd.Series(clusteringGraph1),
                                       columns=["Clust.Coeff."])
        clusteringGraph2 = pd.DataFrame(pd.Series(clusteringGraph2),
                                       columns=["Clust.Coeff."])
```

```
In [13]: fig, ax = plt.subplots(nrows=1, ncols=2)
         ax[0].hist(clusteringGraph1['Clust.Coeff.'], normed=False, alpha=0.5, log=False)
         ax[0].set_title('Clus. Coef. Graph1')
         ax[1].hist(clusteringGraph2['Clust.Coeff.'], normed=False, alpha=0.5, log=False)
         ax[1].set_title('Clus. Coef. Graph2.')
```

Out[13]: <matplotlib.text.Text at 0x7f664f1e2128>



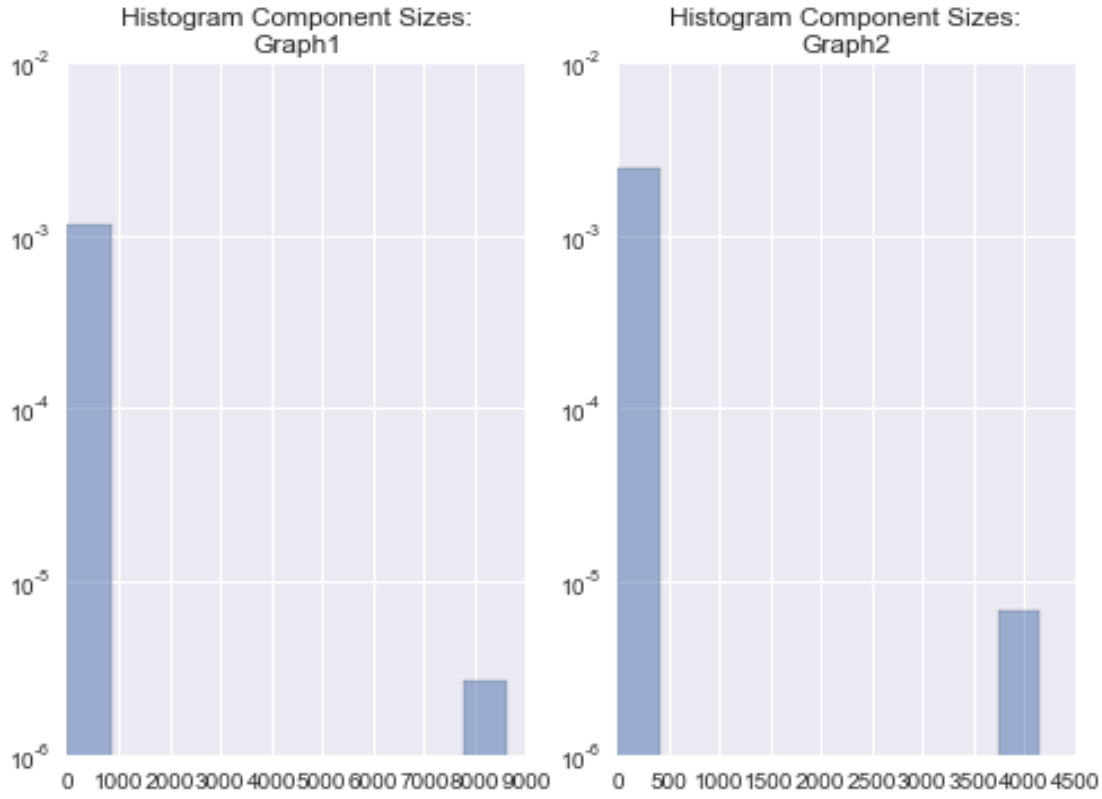
0.4 Number of Connected Components and Their Sizes

```
In [15]: number_components_graph1 = nx.number_connected_components(graph1)
         number_components_graph2 = nx.number_connected_components(graph2)
         print("Number Components Email Enron:", number_components_graph1)
         print("Number Components Astro Ph.:", number_components_graph2)
```

```
Number Components Email Enron: 429
Number Components Astro Ph.: 355
```

```
In [17]: con_comp_graph1 = list(nx.connected_component_subgraphs(graph1))
         con_comp_graph2 = list(nx.connected_component_subgraphs(graph2))
         scon_comp_graph1 = [nx.number_of_nodes(con_comp_graph1[x])  for x in range(0, len(con_c
         scon_comp_graph2 = [nx.number_of_nodes(con_comp_graph2[x])  for x in range(0, len(con_c
         fig, ax = plt.subplots(nrows=1, ncols=2)
         ax[0].hist(scon_comp_graph1, normed=True, stacked=True, log=True,  alpha=0.5)
         ax[0].set_title('Histogram Component Sizes:\nGraph1')
         ax[1].hist(scon_comp_graph2, normed=True, stacked=True, log=True,  alpha=0.5)
         ax[1].set_title('Histogram Component Sizes:\nGraph2')
```

Out[17]: <matplotlib.text.Text at 0x7f664df3d6d8>

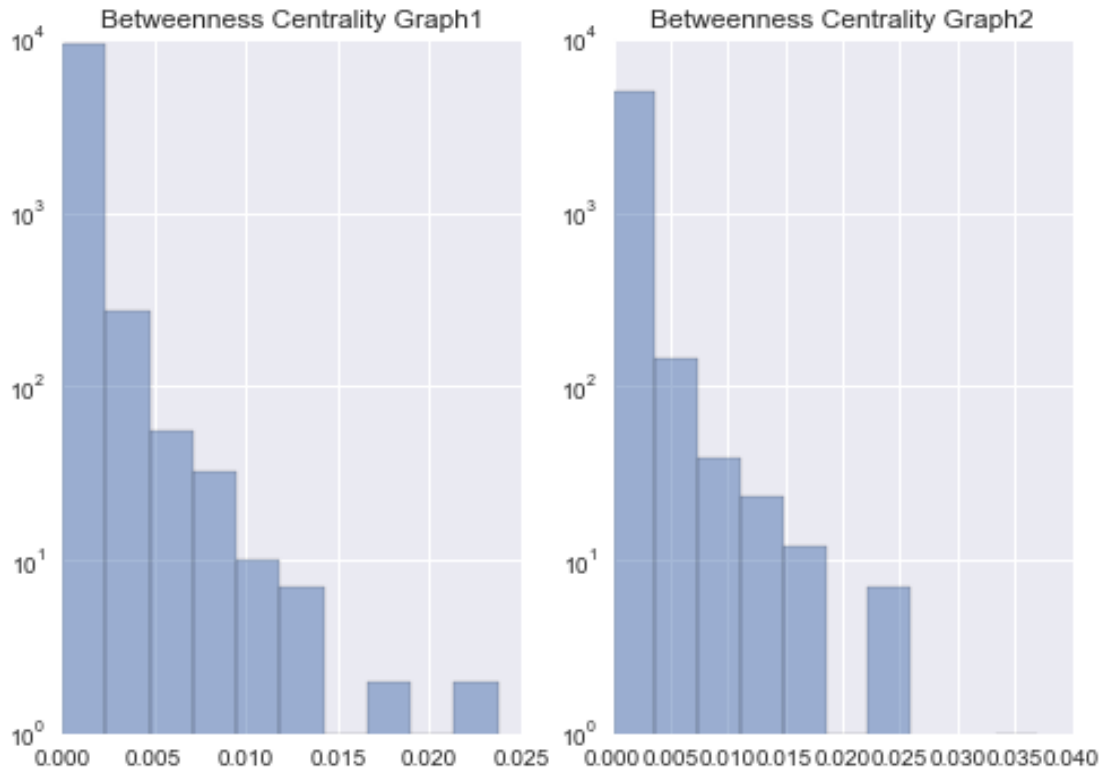


0.5 Betweenness Centrality

In [18]:
```
betweenness_centrality_graph1 = nx.betweenness_centrality(graph1)
betweenness_centrality_graph2 = nx.betweenness_centrality(graph2)
betweennessGraph1 = pd.DataFrame(pd.Series(betweenness_centrality_graph1),
                                 columns=["Bet_Centrality"])
betweennessGraph2 = pd.DataFrame(pd.Series(betweenness_centrality_graph2),
                                 columns=["Bet_Centrality"])
```

In [29]:
```
fig, ax = plt.subplots(nrows=1, ncols=2)
ax[0].hist(betweennessGraph1['Bet_Centrality'], normed=False, alpha=0.5, log=True)
ax[0].set_title('Betweenness Centrality Graph1')
ax[1].hist(betweennessGraph2['Bet_Centrality'], normed=False, alpha=0.5, log=True)
ax[1].set_title('Betweenness Centrality Graph2')
```

Out[29]: <matplotlib.text.Text at 0x7f664e1ec080>

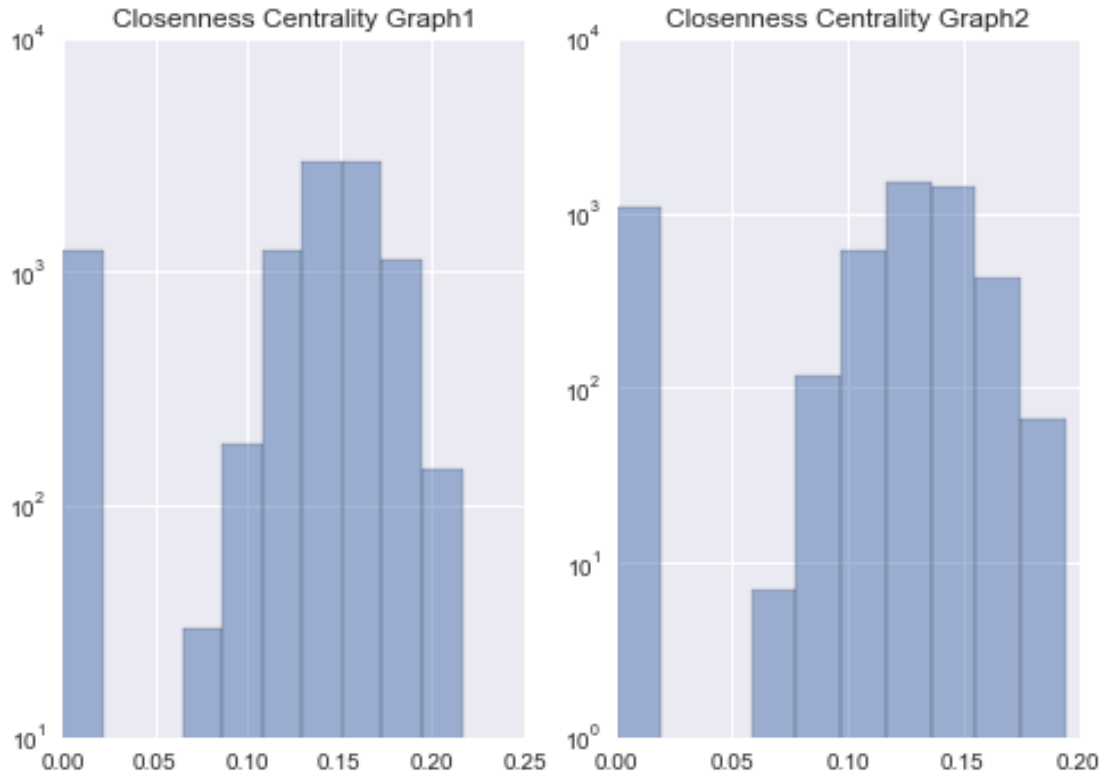


0.6 Closenness Centrality

```
In [20]: closenness_centrality_graph1 = nx.closeness_centrality(graph1)
         closenness_centrality_graph2 = nx.closeness_centrality(graph2)
         closennessGraph1 = pd.DataFrame(pd.Series(closenness_centrality_graph1), columns=["Clos
         closennessGraph2 = pd.DataFrame(pd.Series(closenness_centrality_graph2), columns=["Clos
```

```
In [38]: fig, ax = plt.subplots(nrows=1, ncols=2)
         ax[0].hist(closennessGraph1['Clos_Centrality'], normed=False, alpha=0.5, log=True)
         ax[0].set_title('Closenness Centrality Graph1')
         ax[1].hist(closennessGraph2['Clos_Centrality'], normed=False, alpha=0.5, log=True)
         ax[1].set_title('Closenness Centrality Graph2')
```

```
Out[38]: <matplotlib.text.Text at 0x7f664c89beb8>
```



0.7 Diameter

```
In [61]: diameter_graph1 = list()
         for subgraph in con_comp_graph1:
             diameter_graph1.append(nx.diameter(subgraph))
         diameter_graph2 = list()
         for subgraph in con_comp_graph2:
             diameter_graph2.append(nx.diameter(subgraph))
```

```
In [65]: print("Maximum diameter for Graph1: ", len(diameter_graph1))
         print("Maximum diameter for Graph2: ", len(diameter_graph2))
```

```
Maximum diameter for Graph1:  429
Maximum diameter for Graph2:  355
```

```
In [112]: pd_correlation = pd.DataFrame({'Degree G1':degreeGraph1.sort_values(by="Degree", ascen
                                         'Degree G2':degreeGraph2.sort_values(by="Degree", ascen
                                         'Betw. G1':betweennessGraph1.sort_values(by='Bet_Centra
                                         'Betw. G2':betweennessGraph2.sort_values(by='Bet_Centra
                                         'Clos. G1':closennessGraph1.sort_values(by='Clos_Centra
                                         'Clos. G2':closennessGraph2.sort_values(by='Clos_Centra
                                        })
```

In [116]:
```
pd_correlation.corr(method='pearson')
```

Out[116]:

	Betw. G1	Betw. G2	Clos. G1	Clos. G2	Degree G1	Degree G2
Betw. G1	1.000000	0.993556	0.896832	0.888738	0.955736	0.964774
Betw. G2	0.993556	1.000000	0.896370	0.889669	0.956569	0.967485
Clos. G1	0.896832	0.896370	1.000000	0.998184	0.983691	0.968010
Clos. G2	0.888738	0.889669	0.998184	1.000000	0.981173	0.966477
Degree G1	0.955736	0.956569	0.983691	0.981173	1.000000	0.994484
Degree G2	0.964774	0.967485	0.968010	0.966477	0.994484	1.000000

In [117]:
```
pd_correlation.corr(method='kendall')
```

Out[117]:

	Betw. G1	Betw. G2	Clos. G1	Clos. G2	Degree G1	Degree G2
Betw. G1	1.000000	0.998953	0.999868	0.999532	0.970251	0.969565
Betw. G2	0.998953	1.000000	0.998861	0.998569	0.971103	0.969950
Clos. G1	0.999868	0.998861	1.000000	0.999466	0.970355	0.969674
Clos. G2	0.999532	0.998569	0.999466	1.000000	0.970647	0.969990
Degree G1	0.970251	0.971103	0.970355	0.970647	1.000000	0.984547
Degree G2	0.969565	0.969950	0.969674	0.969990	0.984547	1.000000

In [118]:
```
pd_correlation.corr(method='spearman')
```

Out[118]:

	Betw. G1	Betw. G2	Clos. G1	Clos. G2	Degree G1	Degree G2
Betw. G1	1.000000	0.999971	1.000000	0.999995	0.997647	0.997213
Betw. G2	0.999971	1.000000	0.999971	0.999967	0.997637	0.997093
Clos. G1	1.000000	0.999971	1.000000	0.999995	0.997645	0.997213
Clos. G2	0.999995	0.999967	0.999995	1.000000	0.997644	0.997214
Degree G1	0.997647	0.997637	0.997645	0.997644	1.000000	0.997168
Degree G2	0.997213	0.997093	0.997213	0.997214	0.997168	1.000000

In [ ]: