CN-Exercise1

April 17, 2016

Complex Networks Exercise 1 - Network Models

Cole MacLean - April 3, 2016

Part 1 - Erdös-Rényi Networks

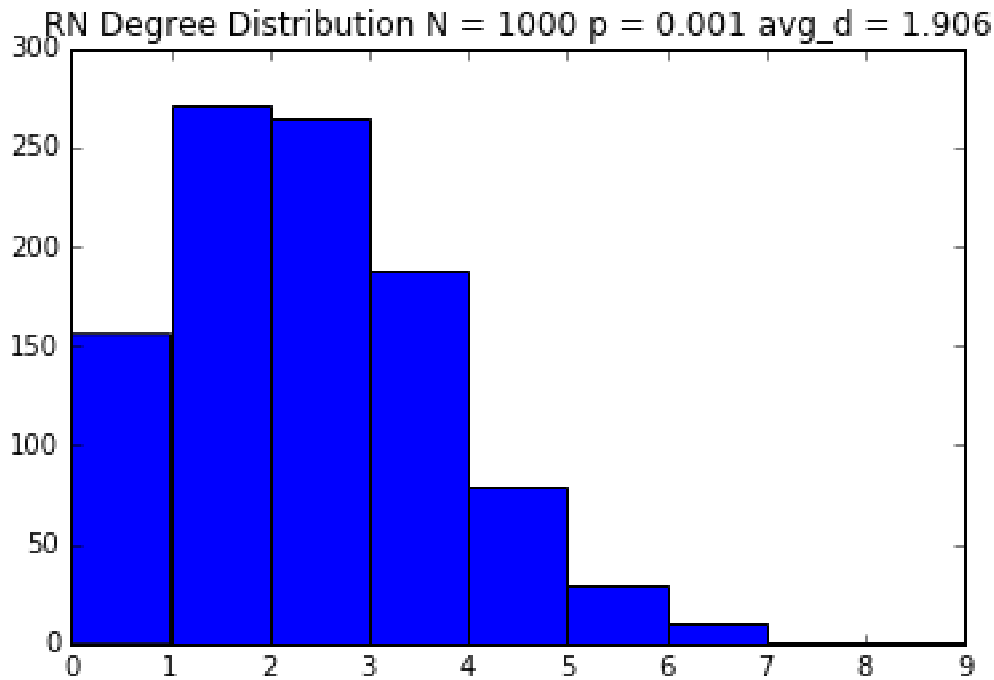
In this section, a function is defined to take as inputs the number of nodes and probably of 2 nodes being linked to generate an ER random graph. Histograms are then plotted for various combinations of node count and probability of linkage to analyse the characteristics of ER random networks.

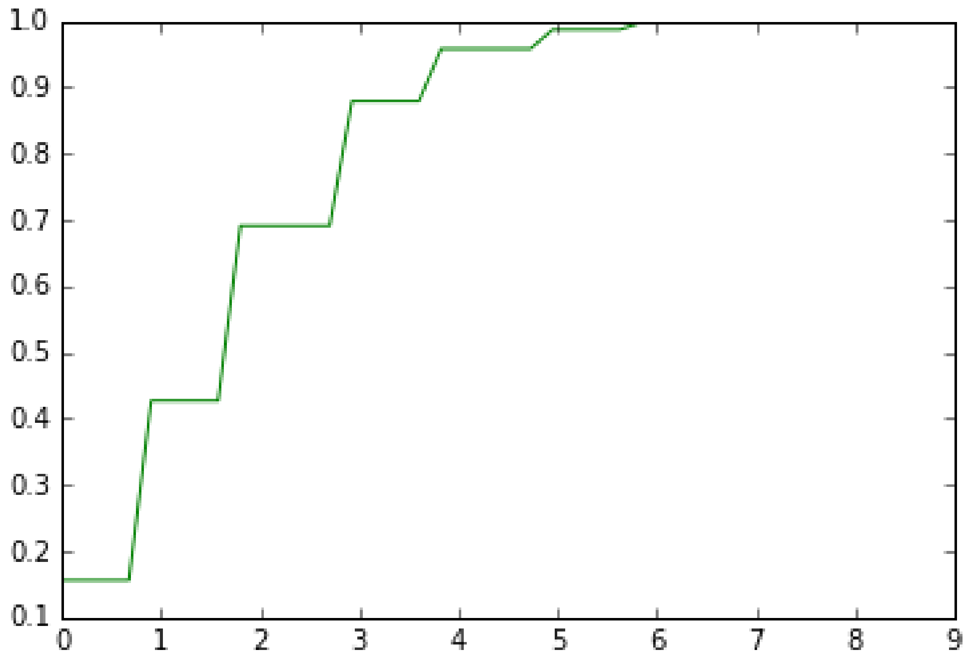
```
In [1]: import random
        import numpy as np
        import matplotlib.pyplot as plt
        import powerlaw as pl
        import warnings
        warnings.filterwarnings('ignore')
        %matplotlib inline

        def rand_ER_network(node_count,link_prob):
            #generator that randomly selects connections for each node,
            #creating a directed network
            network = ({node:[link for link in range(node_count)
                        if random.random() <= link_prob]
                        for node in range(node_count)})
            #search over network to develop non-directed network by adding nodes that contain a link
            #to the current node if not already connected
            for node,links in network.items():
                for link in links:
                    if node not in network[link]:
                        network[link].append(node)
            return network
```

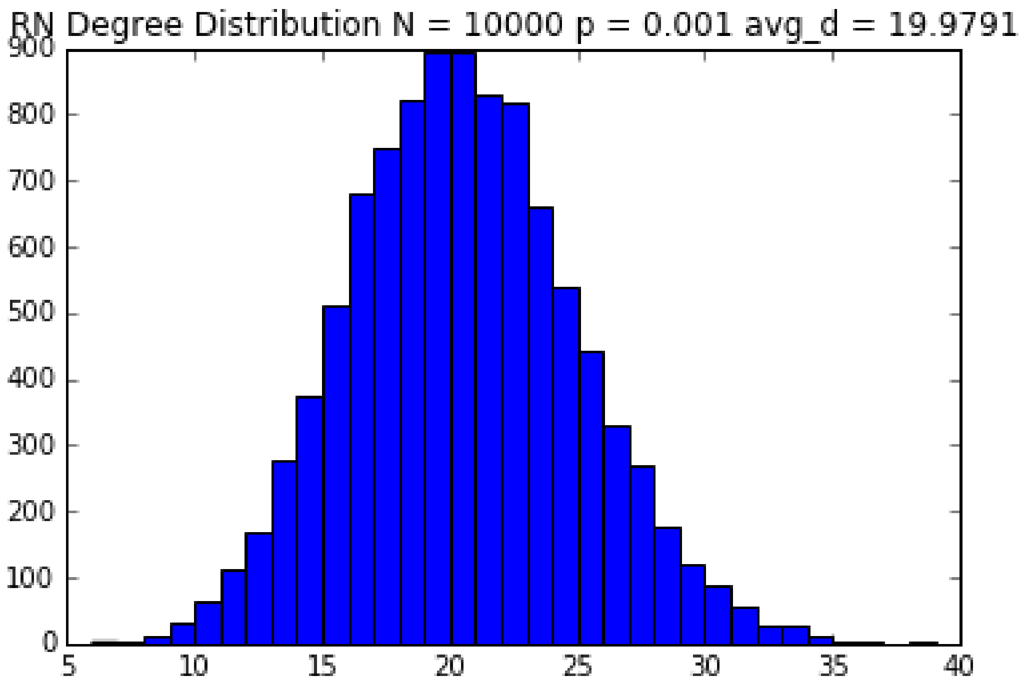
```
In [25]: def plot_distributions(node_count,link_prob):
             network = rand_ER_network(node_count,link_prob)
             degrees = [len(network[i]) for i in network.keys()] #Extract node degrees from network
             plt.hist(degrees, bins=range(min(degrees), max(degrees) + 1, 1))
             plt.title("RN Degree Distribution N = " + str(node_count) + " p = " + str(link_prob)
                       + " avg_d = " + str(np.mean(degrees)))
             plt.show() #plot degree distribution
             values, base = np.histogram(degrees, bins=40)
             cumulative = np.cumsum(values)/node_count
             plt.plot(base[:-1], cumulative, c='green')
             plt.show() #plot cummulative distribution
             return degrees
```

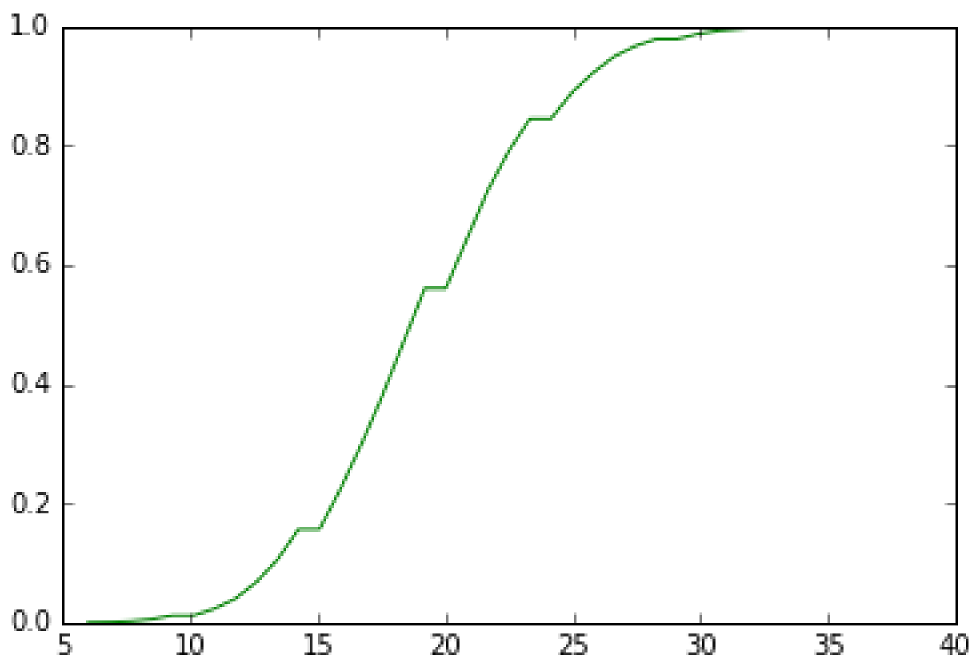
```
In [37]: degrees = plot_distributions(1000,0.001)
```



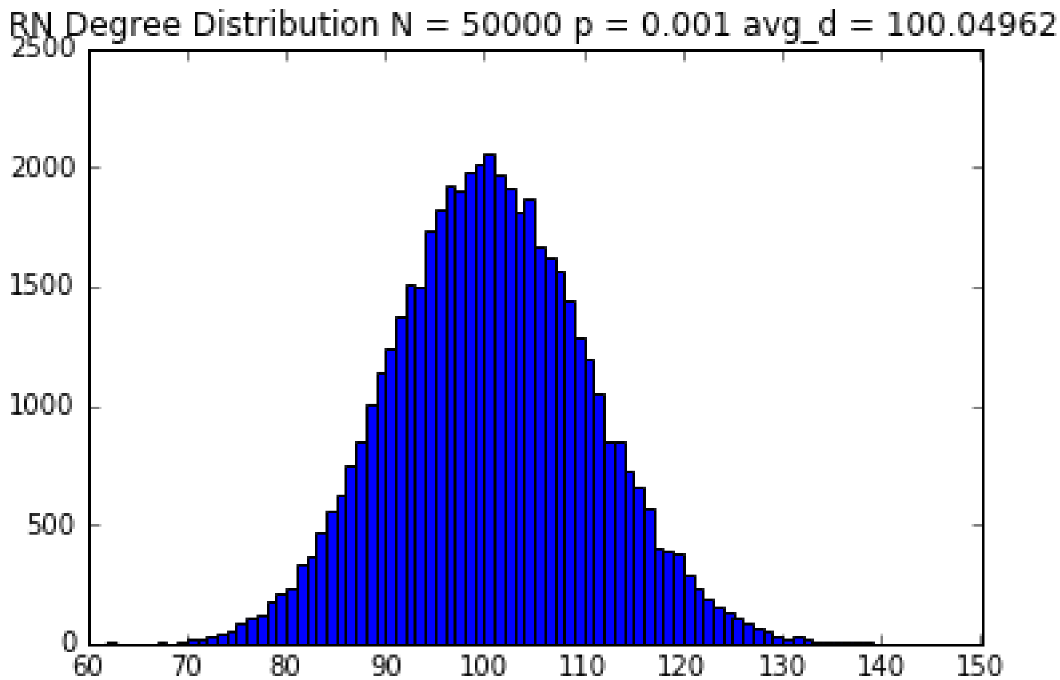


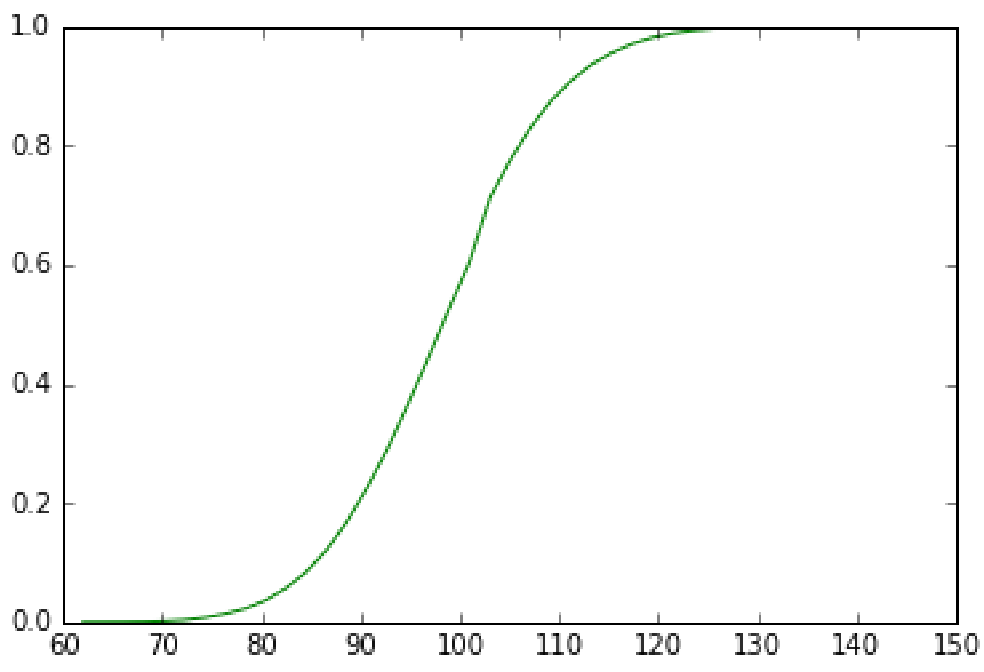

```
In [38]: degrees = plot_distributions(10000,0.001)
```



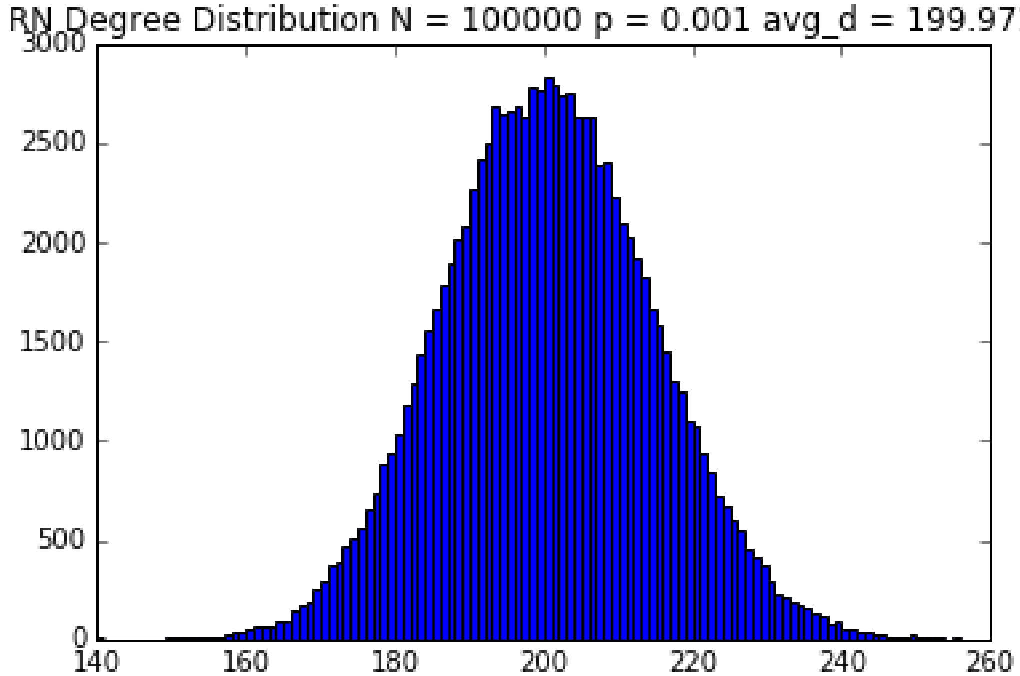


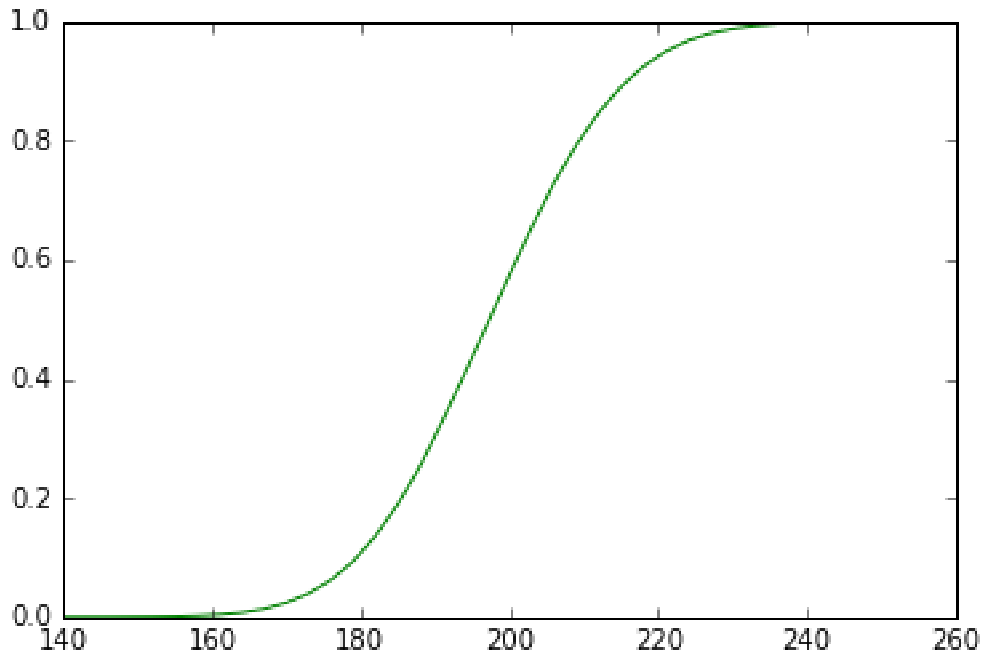

```
In [39]: degrees = plot_distributions(50000,0.001)
```



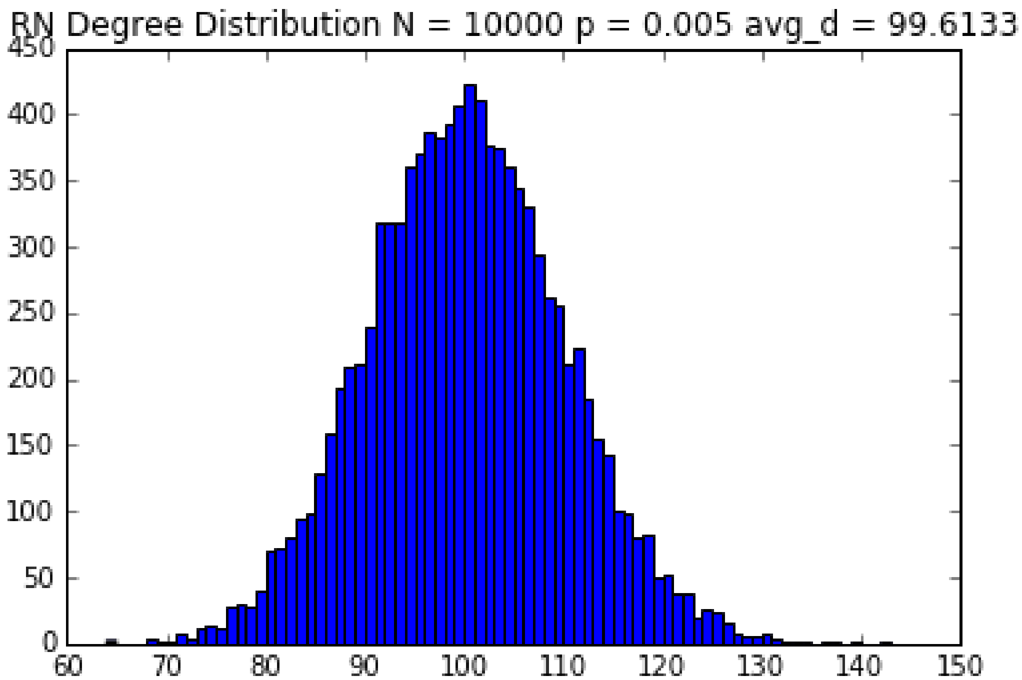


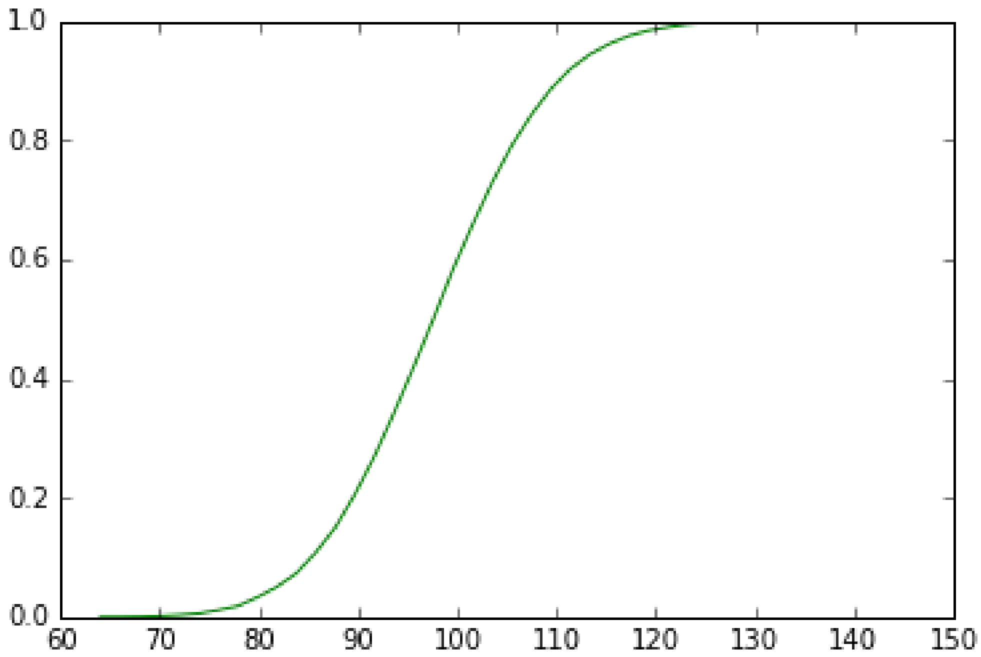
```
In [40]: degrees = plot_distributions(100000,0.001)
```



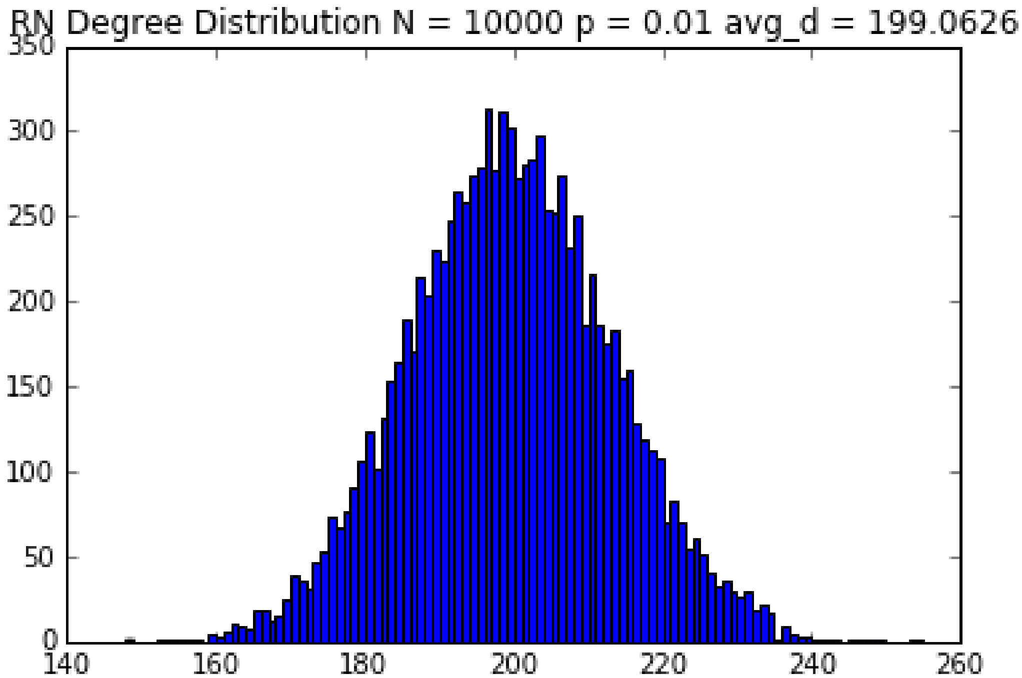


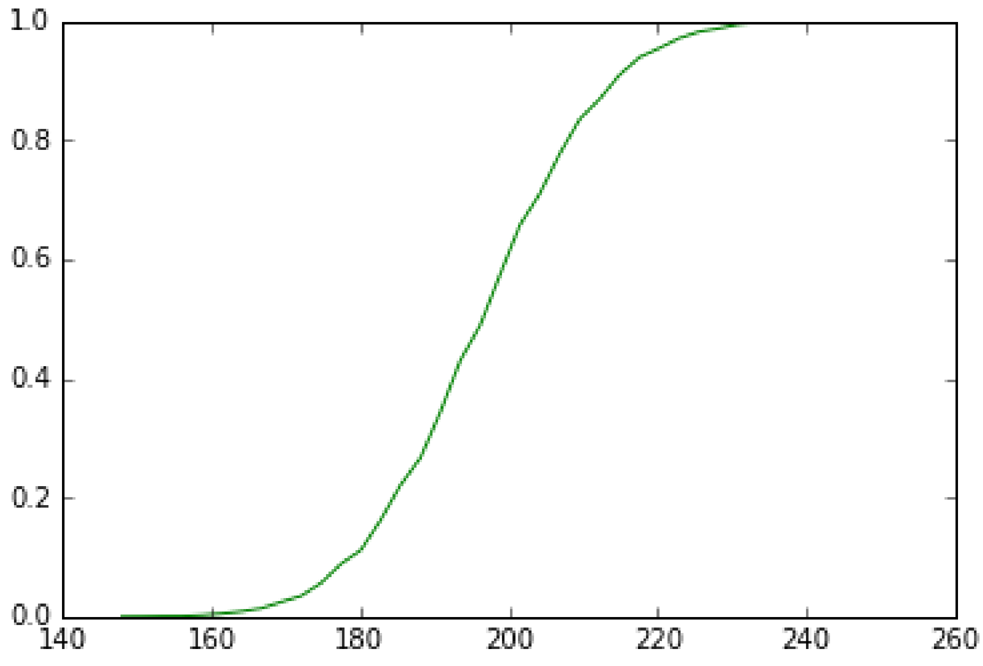

```
In [41]: degrees = plot_distributions(10000,0.005)
```



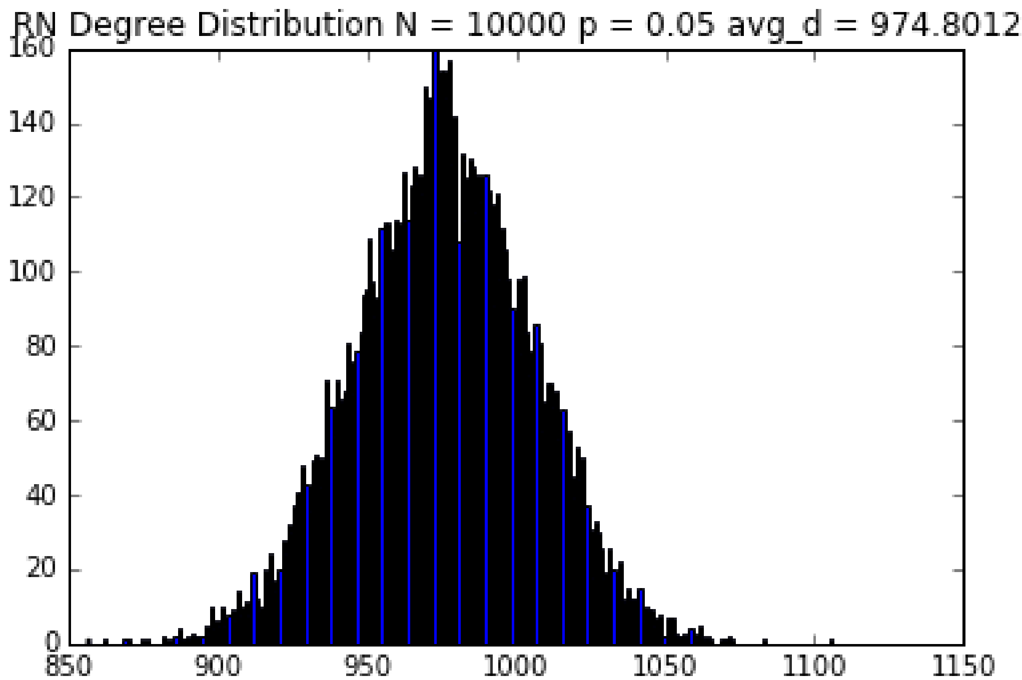


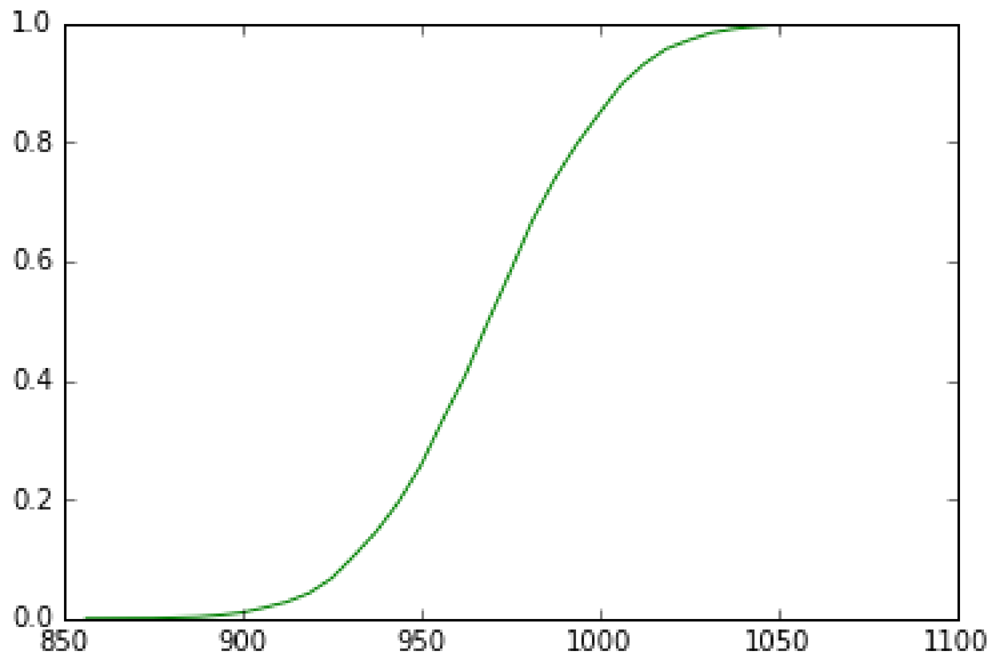

```
In [42]: degrees = plot_distributions(10000,0.01)
```






```
In [43]: degrees = plot_distributions(10000,0.05)
```





From the plots, we can see that the ER random networks tend to normality, and converge to Poisson distributions for larger and larger N.

Part 2 - Barabási–Albert (BA) Preferential Attachment Model

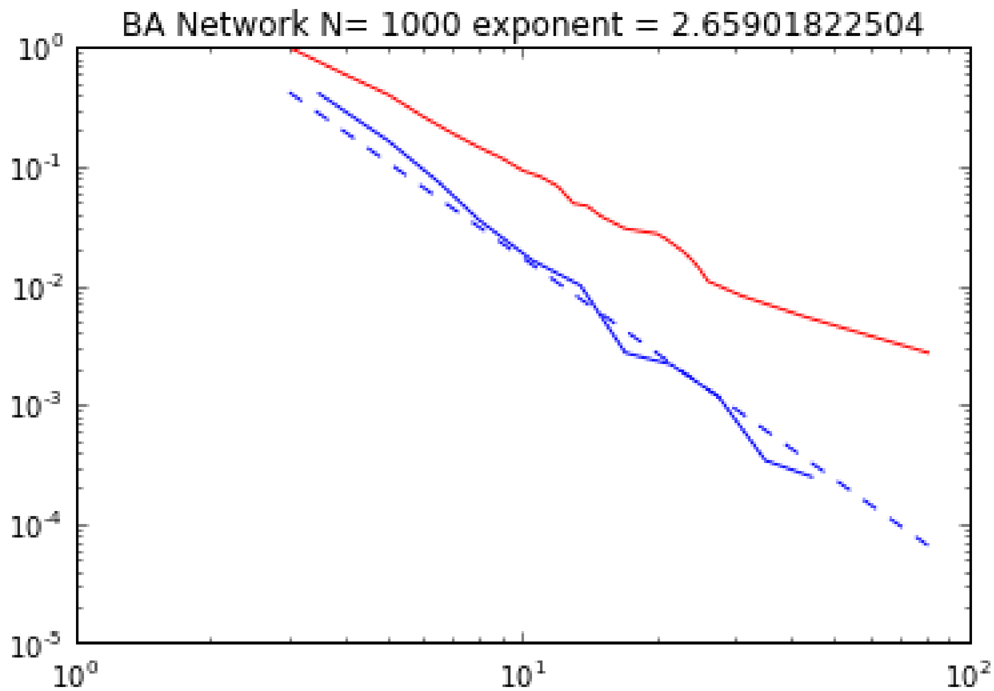
In this section, the Barabási–Albert method of using growth and preferential attachment to model complex networks is analyzed

```
In [2]: def BA_network(start_node_count,start_link_prob,tot_nodes):
            #Random ER network used as initiator of start network in BA model
            #growth iterator, adding new node per step until total nodes reached
            network = rand_ER_network(start_node_count,start_link_prob)
            for new_node in range(start_node_count,tot_nodes):
                network[new_node] = [] #add new node to network
                #calculate the sum total of degrees in the network
                tot_network_degrees = sum([len(network[i]) for i in network.keys()])
                for node,links in network.items():#preferential attachement iterator:
                                                 #check degree of each existing node to calc link prob
                    if tot_network_degrees:#because original network initiated
                                            # with random ER network, there is a none-zero probability
                        link_prob = len(links)/tot_network_degrees#of network with sum total degree of
                                                                  #(ie no links), causing div by 0.
                    else:
                        link_prob = start_link_prob#use original start_link_prob if no links exist in n
                    if random.random() <= link_prob:#connect nodes non-directionally if probability sat
                        network[new_node].append(node)
                        network[node].append(new_node)
            return network
```

```
In [58]: import powerlaw as pl
         def plot_log_distributions(node_count,link_prob,tot_nodes):
             network = BA_network(node_count,link_prob,tot_nodes)
             degrees = [len(network[i])+1 for i in network.keys()] #Extract node degrees from network
             fig = plt.figure()
             ax = fig.add_subplot(111)
             fit = pl.Fit(np.array(degrees),discrete=True, fit_method='Likelihood')
             fit.power_law.plot_pdf( color= 'b',linestyle='--',label='fit ccdf',ax=ax)
             fit.plot_pdf( color= 'b',ax=ax)
             ax.set_title('BA Network N= '+ str(tot_nodes) + ' exponent = ' + str(fit.power_law.alpha))
             fit.plot_ccdf( color= 'r',ax=ax)
             return degrees
```

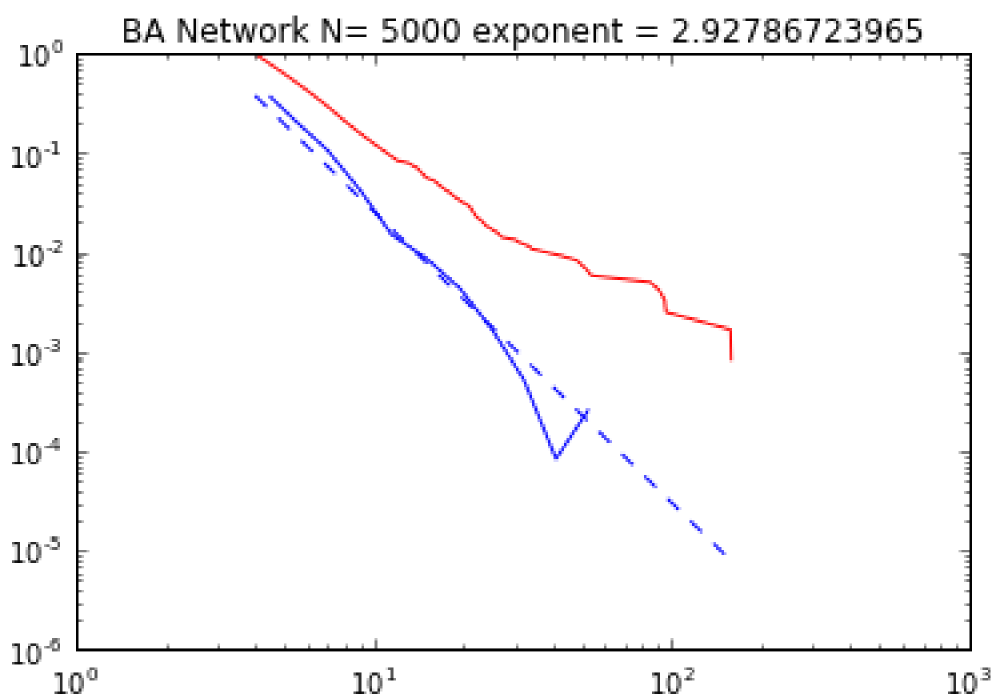
```
In [54]: degrees = plot_log_distributions(2,0.5,1000)
```

```
Calculating best minimal value for power law fit
```



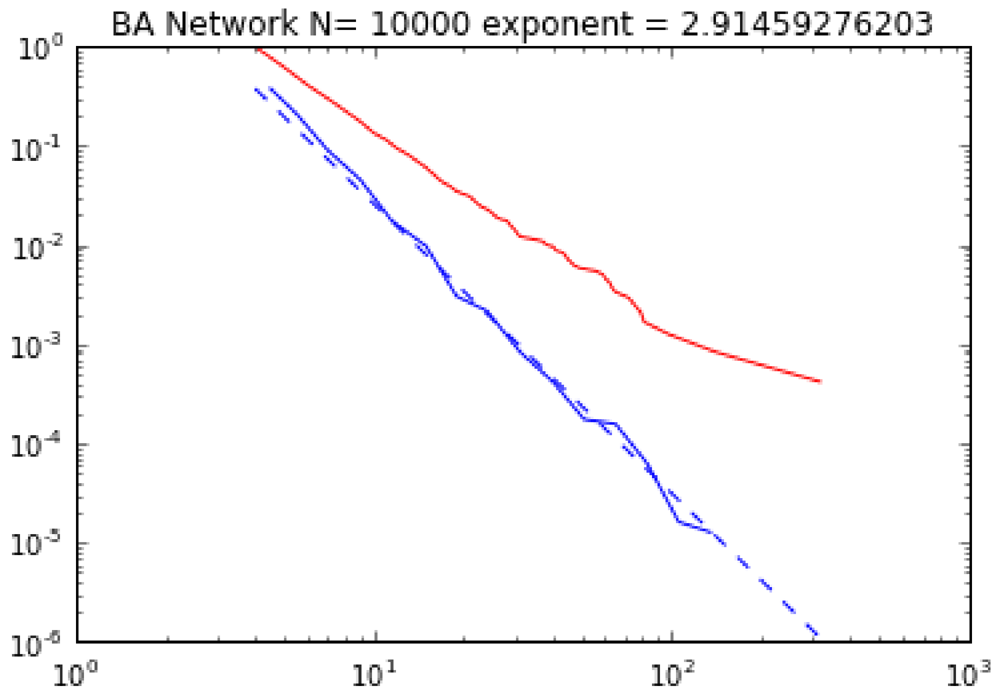
In [55]: degrees = plot_log_distributions(2,0.5,5000)

Calculating best minimal value for power law fit



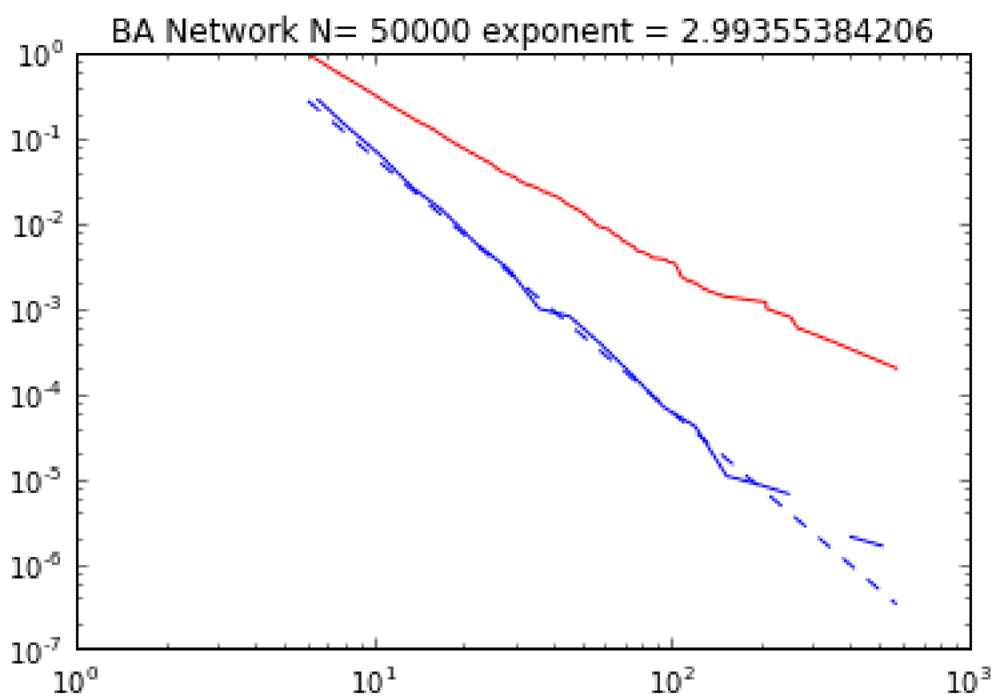
```
In [56]: degrees = plot_log_distributions(2,0.5,10000)
```

```
Calculating best minimal value for power law fit
```




```
In [57]: degrees = plot_log_distributions(2,0.5,50000)
```

```
Calculating best minimal value for power law fit
```



Using pythons powerlaw library to compute the pdf and ccdf of various sized BA networks, we can see that the exponent of the power law distributions converge to 3.0 for large values of N.