

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

Networkx

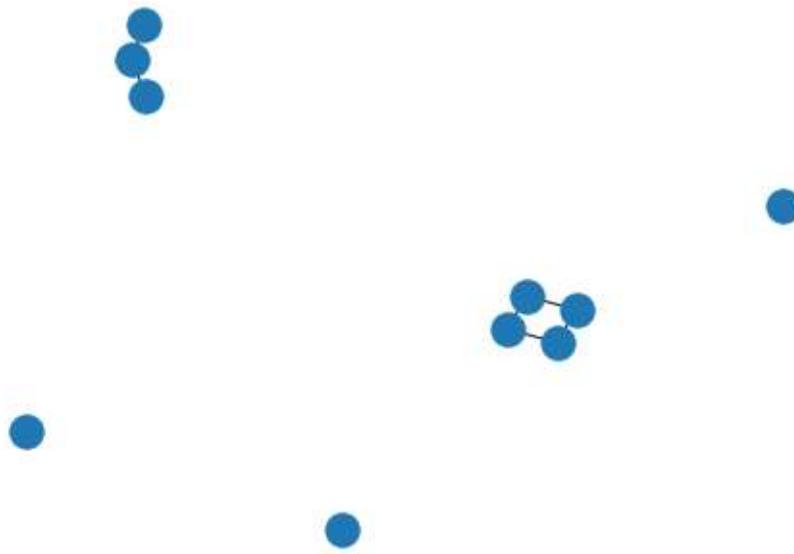
Networkx is a Python library people commonly use when dealing with graphs. Please follow the instructions to install the package: <https://networkx.org/documentation/stable/install.html>. A tutorial for quick-start is also available on their webpage: <https://networkx.org/documentation/stable/tutorial.html>.

$$G(n, p)$$

This notebook aims to empirically investigate the properties of Erdős-Rényi graphs, specifically the $G(n, p)$ model, and compare them with real-world networks. Networkx provides a convenient way to generate random graphs using built-in functions for various commonly used graph models. To create an Erdős-Rényi graph, we can use the function `nx.erdos_renyi_graph(n, p)`, where `n` is the number of nodes and `p` is the probability of an edge between two nodes. The function returns an `nx.Graph` object containing the nodes and edges of the generated graph.

```
In [ ]: # Example: generate a graph from G(10, 0.1)
G_test = nx.erdos_renyi_graph(10, 0.1)
# Check how many nodes and edges it has.
print(G_test.number_of_nodes(), G_test.number_of_edges())
# Plot the graph
nx.draw(G_test)
```

10 6



Statistics of $G(n, p)$ (20 pts)

In the upcoming code cell, you will be requested to generate graphs from $G(n, p)$ model with n set to 100, using different values of p ranging from 0.001 to 0.081, with a small step size such as 0.005. For each value of p , you should generate 10 graph samples from the model and report the average of the following graph statistics with error bars for std. We also provide a brief explanation on how to compute these statistics using networkx functions:

- Number of edges: To obtain the number of edges in a given nx.graph object G, you can utilize the function `G.number_of_edges()` .
- Number of triangles: Using `nx.triangles(G)` , a dictionary of (node id, number of triangles participated) key-value pairs is returned. To calculate the total number of triangles in G, simply sum all the values in the dictionary and divide the result by 3 (since each triangle is counted three times in the dictionary).
- Number of isolated nodes: If a node has degree 0, then it is an isolated node. The function `G.degree` is a map-like object consisting of (node id, node degree) pairs. To count the number of isolated nodes, iterate through `G.degree` and count the number of 0s in the values.
- Number of connected components: In graph theory, a connected component is a set of vertices in a graph that are linked to each other by paths. Using `nx.connected_components(G)`, a node list generator is returned, which yields one component at a time. To get the total number of connected components, you can use `len(list(nx.connected_components(G)))` .

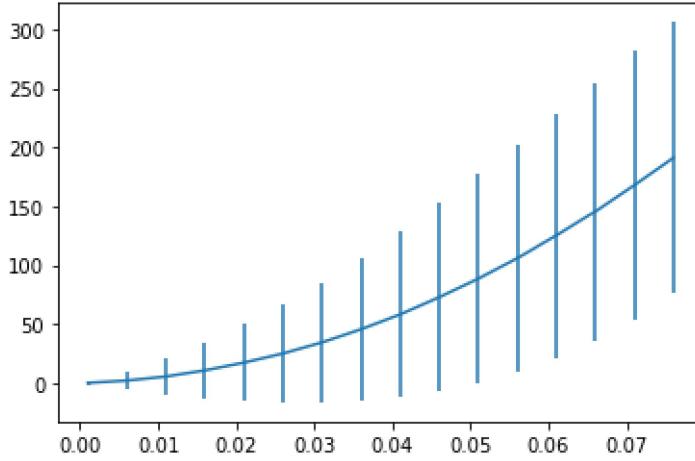
For each graph statistic, generate a plot where the x-axis represents the values of p , and the y-axis represents the average of the statistic.

```
In [ ]: # Write your code below
edge = np.zeros((16, 10))
triangle = np.zeros((16, 10))
isolated = np.zeros((16, 10))
cc = np.zeros((16, 10))
avg_edge = np.zeros(16)
avg_triangle = np.zeros(16)
avg_isolated = np.zeros(16)
avg_cc = np.zeros(16)
std_edge = np.zeros(16)
std_triangle = np.zeros(16)
std_isolated = np.zeros(16)
std_cc = np.zeros(16)
prob = np.arange(0.001, 0.081, 0.005)
for i, p in enumerate(prob):
    for j in range(10):
        G_test = nx.erdos_renyi_graph(100, p)
        edge[i][j] = nx.number_of_edges(G_test)
        triangle[i][j] = sum(nx.triangles(G_test).values()) / 3
        isolated[i][j] = sum(1 for degree in G_test.degree() if degree[1] == 0)
        cc[i][j] = len(list(nx.connected_components(G_test)))
    avg_edge[i] = np.mean(edge)
    std_edge[i] = np.std(edge)
    avg_triangle[i] = np.mean(triangle)
    std_triangle[i] = np.std(triangle)
    avg_isolated[i] = np.mean(isolated)
    std_isolated[i] = np.std(isolated)
    avg_cc[i] = np.mean(cc)
    std_cc[i] = np.std(cc)
```

```
In [ ]: # Generate plot with error bars for number of edges.
plt.errorbar(x=prob, y=avg_edge, yerr = std_edge)

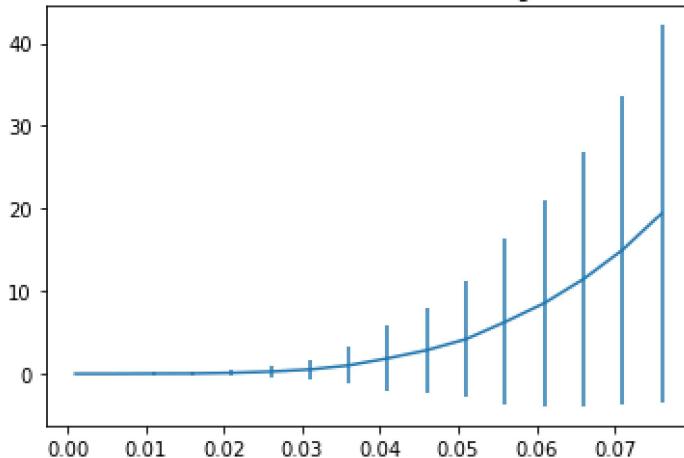
plt.title('Error bars for number of edges')
plt.show()
```

Error bars for number of edges



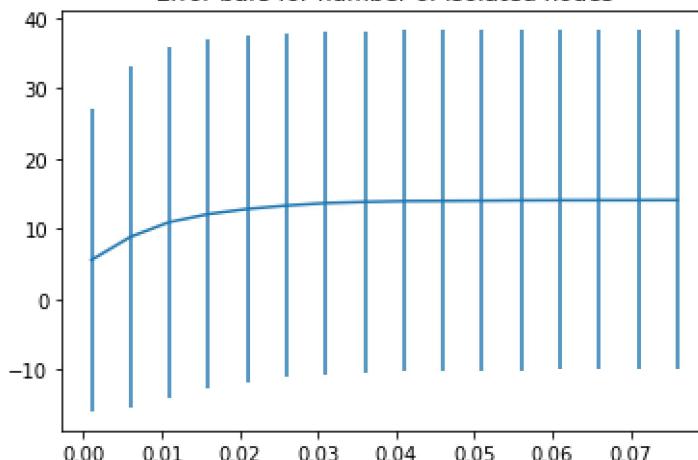
```
In [ ]: # Generate plot with error bars for number of triangles.  
plt.errorbar(x=prob, y=avg_triangle, yerr = std_triangle)  
  
plt.title('Error bars for number of triangles')  
plt.show()
```

Error bars for number of triangles



```
In [ ]: # Generate plot with error bars for number of isolated nodes.  
plt.errorbar(x=prob, y=avg_isolated, yerr = std_isolated)  
  
plt.title('Error bars for number of isolated nodes')  
plt.show()
```

Error bars for number of isolated nodes

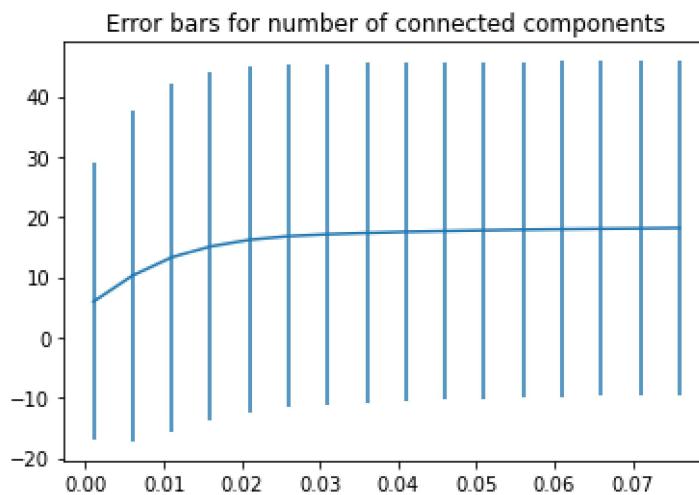


```
In [ ]: # Generate plot with error bars for number of connected components.  
plt.errorbar(x=prob, y=avg_cc, yerr = std_cc)
```

```

plt.title('Error bars for number of connected components')
plt.show()

```



Real world network

Graph statistics (10pts)

In the same folder we provide a real-world social network dataset saved as edgelist format named "fb-pages-food.edges". Each line in this file has the format of "node1,node2" that represents an edge connecting node1 and node2. An edgelist file can be loaded as nx.Graph directly using nx.read_edgelist() like the following.

```
In [ ]: G = nx.read_edgelist("fb-pages-food.edges", delimiter=',')
```

How many nodes and edges are there in this graph? Also report the number of triangles, number of isolated nodes, and number of connected components in this graph.

```
In [ ]: # Report the graph statistics.
nodes = nx.number_of_nodes(G)
edge = nx.number_of_edges(G)
triangle = sum(nx.triangles(G).values()) / 3
isolated = sum(1 for degree in G.degree() if degree[1] == 0)
cc = len(list(nx.connected_components(G)))
```

```

print("nodes ", nodes)
print("edges ", edge)
print("triangles ", triangle)
print("isolated nodes ", isolated)
print("connected components ", cc)

```

```

nodes . 620
edges . 2102
triangles . 2935.0
isolated nodes . 0
connected components . 1

```

In the next cell, plot a histogram of the node degrees in this graph with bins=15.

```
In [ ]: d = G.degree()
l = np.array([])
for (a,b) in d:
    l = np.append(l, b)

# Plot a histogram of the node degrees with 15 bins
plt.hist(l, bins=15)
```

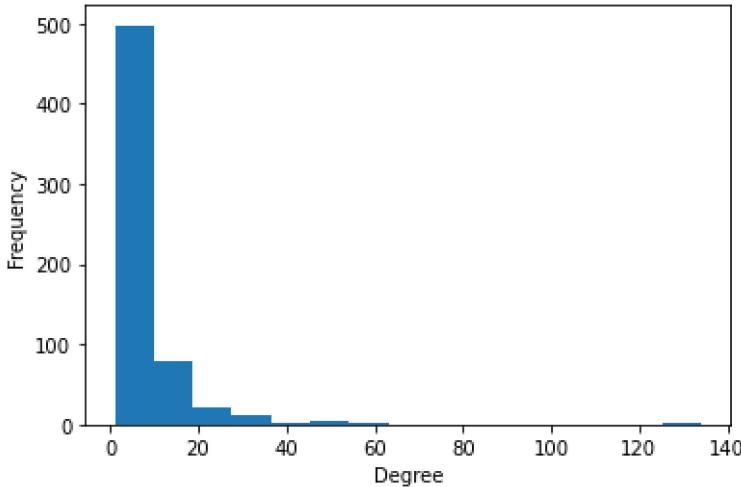
```

# Set the x-axis label
plt.xlabel('Degree')

# Set the y-axis label
plt.ylabel('Frequency')

# Show the plot
plt.show()

```



Try to fit the data with G(n,p) (10pts)

If we want to fit the graph with $G(n,p)$ model, i.e., find a $G(n,p)$ model whose expected number of edges equals to number of edge in this graph, How should we set p to get m edges in expectation?

```
In [ ]: # calculate and print the value of p.
p = 2 * edge / (nodes*(nodes-1))
print (p)
```

0.0109541925061233

Now use the p value you calculated and sample 10 graphs from $G(n,p)$. Report the same graph statistics in average.

```
In [ ]: n = nodes
nod = np.array([])
edg = np.array([])
tri = np.array([])
iso = np.array([])
c = np.array([])
s = nx.erdos_renyi_graph(n, p)
for i in range(10):
    G_sample= nx.erdos_renyi_graph(n, p)
    nod = np.append (nod, nx.number_of_nodes(G_sample))
    edg = np.append(edg,nx.number_of_edges(G_sample))
    tri = np.append(tri,sum(nx.triangles(G_sample).values())/3)
    iso = np.append (iso,sum(1 for degree in G_sample.degree() if degree[1] == 0))
    c = np.append (c,len(list(nx.connected_components(G_sample))))
mean_nodes = np.mean(nod)
mean_edges = np.mean(edg)
mean_tri = np.mean(tri)
mean_iso = np.mean(iso)
mean_cc = np.mean(c)
print("nodes ",mean_nodes)
print("edges ",mean_edges)
print("triangles ",mean_tri)
print("isolated nodes ",mean_iso)
print("connected components ",mean_cc)
```

```

nodes : 620.0
edges : 2106.2
triangles : 48.5
isolated nodes : 0.6
connected components : 1.6

```

Plot the histogram of node degrees for one random graph you generated with bins=15, what's the difference between the node degree sequences of the real graph and the random graph?

```

In [ ]: deg = s.degree()
li = np.array([])
for (a, b) in deg:
    li = np.append(li, b)

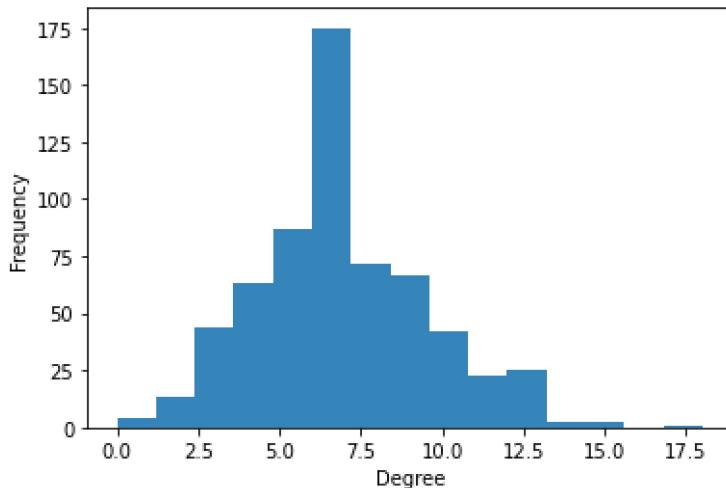
# Plot a histogram of the node degrees with 15 bins
plt.hist(li, bins=15, alpha=0.9, label='random')

# Set the x-axis label
plt.xlabel('Degree')

# Set the y-axis label
plt.ylabel('Frequency')

# Show the plot
plt.show()

```



A random graph model that fits the degree sequence (10pts)

Given a sequence of expected degrees $(d_1, d_2 \dots d_n)$ of length n , we generate a graph with n nodes, and assigns an edge between node u and node v with probability

$$p_{uv} = \frac{d_u d_v}{\sum_k d_k}$$

.

This model is known as the Chung-Lu model and is implemented in the networkx library. To generate a graph from this model, simply use the function

```
nx.expected_degree_graph(node_degree_list, selfloops=False).
```

To compare the generated graph with a real social network, pass the degree sequence of the real network into this model and generate 10 samples. Then plot the degree histogram of the generated graphs and compare it with the original graph. Additionally, report the same graph statistics for the

generated graphs and compare them with the original graph. What observations can we make from this comparison?

```
In [ ]: node_degree_list = np.array([])
for (a, b) in G.degree():
    node_degree_list=np.append (node_degree_list, b)
node_degree_list=list(node_degree_list)
xnode = np.array([])
xedge = np.array([])
xtri = np.array([])
xiso = np.array([])
xc = np.array([])
deg = np.zeros ((10, nodes))

for i in range(10):
    sample = nx.expected_degree_graph(node_degree_list)
    xnode = np.append (xnode, nx.number_of_nodes(sample))
    xedge = np.append (xedge, nx.number_of_edges(sample))
    xtri = np.append (xtri, sum(nx.triangles(sample).values()) / 3)
    xiso = np.append (xiso, sum(1 for degree in G_sample.degree() if degree[1] == 0))
    xc = np.append (xc, len(list(nx.connected_components(sample))))
    for j in range(nodes):
        deg [i][j] = sample.degree[j]
xdeg=deg.mean(axis=0)
mu_nodes = np.mean(xnode)
mu_edges = np.mean(xedge)
mu_tri = np.mean(xtri)
mu_isol = np.mean(xiso)
mu_cc = np.mean(xc)
print("Chung-Lu Statistics")
print("nodes ", mu_nodes)
print("edges ", mu_edges)
print("triangles ", mu_tri)
print("isolated nodes ", mu_isol)
print("connected components ", mu_cc)
print("G's statistics")
print("nodes ", nodes)
print("edges ", edge)
print("triangles ", triangle)
print("isolated nodes ", isolated)
print("connected components ", cc)
```

```
Chung-Lu Statistics
nodes 620.0
edges 2096.2
triangles 1000.9
isolated nodes 1.0
connected components 62.5
G's statistics
nodes 620
edges 2102
triangles 2935.0
isolated nodes 0
connected components 1
```

```
In [ ]: plt.hist(l, bins='auto', alpha =0.5, label='real')
plt.hist(xdeg,bins='auto',alpha =0.5, label='Chung-Lu')
plt.legend(loc='upper right')
# Set the x-axis label
plt.xlabel('Degree')

# Set the y-axis label
plt.ylabel('Frequency')

# Show the plot
plt.show()
```

