SME0130 - Redes Complexas

Modelos de redes: Grafos aleatórios e small world

Professor: Francisco Aparecido Rodrigues, francisco@icmc.usp.br (mailto:francisco@icmc.usp.br).
Estudante: Bruno F. Bessa (num. 5881890), bruno.fernandes.oliveira@usp.br (mailto:bruno.fernandes.oliveira@usp.br)
Universidade de São Paulo, São Carlos, Brasil.

In [1]:

```
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
import math
```

In [33]:

```
def erdos_renyi(N: int,
                p: float,
                plot: bool = True,
                file_name: str = None) -> nx.classes.graph.Graph:

    """
    Define as conexões (i,j) = (j,i) para todos os pares de pontos com base em um
    evento medida aleatória para probabilidade p, recebida como parâmetro na constru
    """

    G = nx.gnp_random_graph(N, p, seed=None, directed=False)

    # Para calcularmos medidas de distância precisaremos remover nós não conectados
    # No trecho abaixo mantemos somente o maior componente conctado da rede.
    G = G.to_undirected()
    G.remove_edges_from(nx.selfloop_edges(G))
    Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
    G = G.subgraph(Gcc[0])
    G = nx.convert_node_labels_to_integers(G, first_label=0)

    # Opção de visualização da rede gerada (não utilizar para séries grandes de expe
    if plot:
        pos = nx.spring_layout(G)
        fig_net = nx.draw(G, pos, node_color='w', node_size=1, with_labels=False)
        plt.suptitle("Erdos-Renyi Network (N={}, p={})".format(N, p), fontsize=15)
        plt.show(fig_net)
    if file_name != None:
        pos = nx.spring_layout(G)
        fig_net = nx.draw(G, pos, node_color='w', node_size=1, with_labels=False)
        plt.suptitle("Erdos-Renyi Network (N={}, p={})".format(N, p), fontsize=15)
        plt.savefig("images/"+file_name)
        plt.close(fig_net)

    return G


def watts_strogatz(N: int,
                avg_deg: float,
                p: float,
                plot: bool = True,
                file_name: str = None) -> nx.classes.graph.Graph:

    """
    """

    k = int(avg_deg)
    G = nx.watts_strogatz_graph(N, k, p, seed=None)

    # Para calcularmos medidas de distância precisaremos remover nós não conectados
    # No trecho abaixo mantemos somente o maior componente conctado da rede.
    G = G.to_undirected()
    G.remove_edges_from(nx.selfloop_edges(G))
    Gcc = sorted(nx.connected_components(G), key=len, reverse=True)
    G = G.subgraph(Gcc[0])
    G = nx.convert_node_labels_to_integers(G, first_label=0)

    # Opção de visualização da rede gerada (não utilizar para séries grandes de expe
    if plot:
        pos = nx.circular_layout(G)
```

```
        fig_net = nx.draw(G, pos, node_color='w', node_size=1, with_labels=False)
        plt.suptitle("Watts-Strogatz Network (N={}, p={})".format(N, p), fontsize=15
        plt.show(fig_net)
    if file_name != None:
        pos = nx.circular_layout(G)
        fig_net = nx.draw(G, pos, node_color='w', node_size=1, with_labels=False)
        plt.suptitle("Watts-Strogatz Network (N={}, p={})".format(N, p), fontsize=15
        plt.savefig("images/"+file_name)
        plt.close(fig_net)

    return G
```

In [39]:

```
# Definições de medidas para as redes

def avg_shortest_path(G: nx.classes.graph.Graph) -> float:
    """
    Percorre todos os nodos do grafo e para cada um deles verifica o menor caminho a
    Retorna a média desses valoeres.
    Disclaimer: this function uses shortest_path_length build in function from Netwo
    """
    dict_shortest_paths = nx.shortest_path_length(G)
    node_path_avg = []
    for node, paths in dict_shortest_paths:
        node_path_avg.append(sum(paths.values())/len(G.nodes()))

    return sum(node_path_avg)/len(node_path_avg)


def degree_distribution(G: nx.classes.graph.Graph) -> list:

    """
    Retorna a lista de valores de grau (k) para todos os nós da rede.
    """

    dict_degree = dict(G.degree())
    list_k = []
    for node, k_value in dict_degree.items():
        list_k.append(k_value)

    return list_k

def momment_of_degree_distribution2(G,m):
    """
    Moment of order m
    """

    M = 0
    N = len(G)
    for i in G.nodes:
        M = M + G.degree(i)**m
    M = M/N
    return M

def clustering_coef_distribution(G: nx.classes.graph.Graph) ->list:

    """
    Retorna a lista de valores de cluster coefficient (cc) para todos os nós da rede

    """

    list_cc_nodes = []
    for node in G.nodes():
        list_cc_nodes.append(nx.clustering(G, node))

    return list_cc_nodes

def spl_distribution(G: nx.classes.graph.Graph) ->list:
    """
    Retorna a lista de valores de shortest path length (spl) para todos os nós da re

    """
```

```
    N = len(G)
    if nx.is_connected(G) == True:
        distance_matrix = np.zeros(shape=(N,N))
        diameter = nx.diameter(G)
        slp_values = []
        for i in np.arange(0,N):
            for j in np.arange(i+1, N):
                if(i != j):
                    aux = nx.shortest_path(G,i,j)
                    dij = len(aux)-1
                    distance_matrix[i][j] = dij
                    distance_matrix[j][i] = dij
                    slp_values.append(dij)
        return slp_values
    else:
        pass

def shannon_entropy(G: nx.classes.graph.Graph) ->float:

    """
    Calcula a entropia de Shannon para um grafo G recebido como parâmetro.
    """
    list_k = degree_distribution(G)
    min_k = np.min(list_k)
    max_k = np.max(list_k)

    k_values= np.arange(0,max_k+1)
    k_prob = np.zeros(max_k+1)
    for k in list_k:
        k_prob[k] = k_prob[k] + 1
    k_prob = k_prob/sum(k_prob)

    H = 0
    for p in k_prob:
        if(p > 0):
            H = H - p*math.log(p, 2)
    return H

def normalized_shannon_entropy(G):
    k,Pk = degree_distribution(G)
    H = 0
    for p in Pk:
        if(p > 0):
            H = H - p*math.log(p, 2)
    return H/math.log(len(G),2)


def complexity_coefficient(G):
    return momment_of_degree_distribution(G, 2)/momment_of_degree_distribution(G,1)


def distribution_plot(list_values: list,
                      plot_title: str = "Histograma de densidade",
                      var_name: str = "Variável",
                      file_name: str = None) -> None:
    """
    Produz histgrama de uma medida recebida na forma de lista.
    """

    avg_value = np.mean(list_values)
```

```
    var_value = np.var(list_values)

    fig, ax = plt.subplots()
    n, bins, patches = ax.hist(list_values, density=True)
    ax.set_xlabel(var_name)
    ax.set_ylabel("Densidade de probabilidade")
    ax.set_title("{} de {}: média={:.2f}, var={:.2f}".format(plot_title,
                                                            var_name,
                                                            avg_value,
                                                            var_value),
                                                            fontsize=15)
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)

def correlation_plot(x: list,
                     y: list,
                     x_label: str = "x",
                     y_label: str = "y",
                     file_name: str = None) -> None:
    """
    Produz gráfico de dispersão de duas variáveis x e y recebidas na forma de listas
    Calcula correlação de Pearson e Spearman para x e y.
    """

    pearson_corr = np.corrcoef(x, y)[0,1]
    spearman_corr, spearman_pval = scipy.stats.spearmanr(x, y)

    fig, ax = plt.subplots()
    ax.scatter(x, y)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title("Dispersão de {} e {}: Pearson: {:.2f}, Spearman: {:.2f} (p-val: {:
```

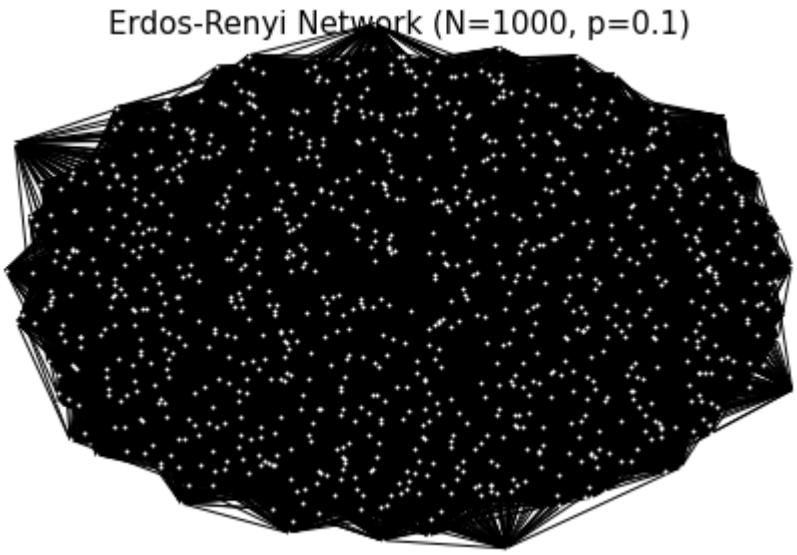
```
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)

def simple_plot2d(x: list,
                  y: list,
                  x_label: str = "x",
                  y_label: str = "y",
                  file_name: str = None) -> None:
    """
    Produz gráfico simples com associação entre suas variáveis x e y recebidas na fo
    """
    fig, ax = plt.subplots()
    ax.plot(x, y)
    ax.set_xlabel(x_label)
    ax.set_ylabel(y_label)
    ax.set_title("Dispersão de {} e {}".format(x_label, y_label, fontsize=15))
    plt.show(True)
    if file_name != None:
        fig.savefig("images/"+file_name)
```

Questions

1 - Gere um grafo aleatório com N=1000 e p = 0.1. Qual o valor do grau médio, segundo momento do grau e coeficiente de aglormeração médio (average clustering coefficient)?

In [7]:

```
G = erdos_renyi(N=1000, p=0.1)
```



In [18]:

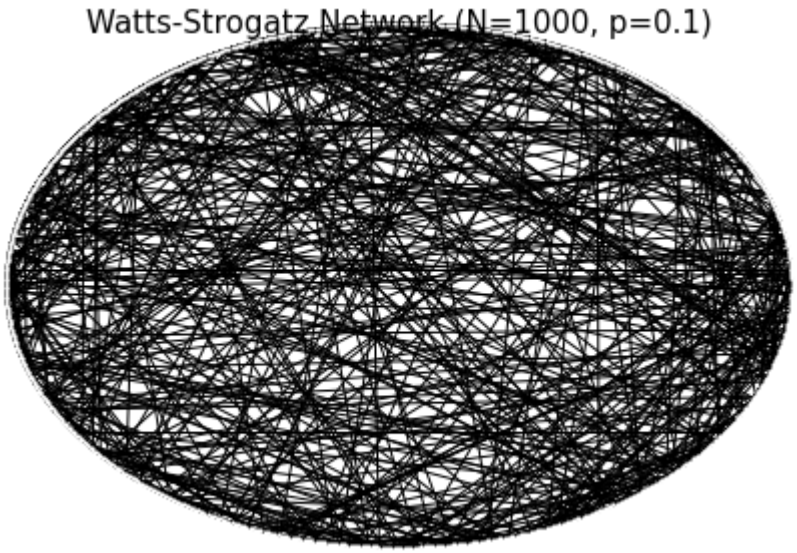
```
avg_deg = np.mean(degree_distribution(G)[0])
sec_moment_deg = momment_of_degree_distribution2(G, 2)
avg_clust_coef = np.mean(clustering_coef_distribution(G)[0])

print("Grau médio: {:.2f}".format(avg_deg))
print("Segundo momento do grau: {:.2f}".format(sec_moment_deg))
print("Cluster coefficient médio: {:.2f}".format(avg_clust_coef))
```

```
Grau médio: 99.00
Segundo momento do grau: 10125.39
Cluster coefficient médio: 0.09
```

02 - Gere um small-world com N=1000, grau médio igual 10 e p = 0.1. Qual o valor do grau médio, segundo momento do grau e coeficiente de aglormeração médio (average clustering coefficient)?

In [34]:

```
G = watts_strogatz(N=1000, avg_deg=10, p=0.1)
```



In [35]:

```
avg_deg = np.mean(degree_distribution(G)[0])
sec_moment_deg = momment_of_degree_distribution2(G, 2)
avg_clust_coef = np.mean(clustering_coef_distribution(G)[0])

print("Grau médio: {:.2f}".format(avg_deg))
print("Segundo momento do grau: {:.2f}".format(sec_moment_deg))
print("Cluster coefficient médio: {:.2f}".format(avg_clust_coef))
```

```
Grau médio: 10.00
Segundo momento do grau: 100.85
Cluster coefficient médio: 0.58
```

3 - Considere uma rede aleatória (Erdos-Renyi) com N=1000 vértices. Qual o valor da entropia de Shannon do grau para $\langle k \rangle$ =5, $\langle k \rangle$ =10, $\langle k \rangle$ =50.

Usaremos a seguinte propriedade de redes aleatórias:

$$p = \frac{k}{(N-1)}$$

In [40]:

```
G_3_1 = erdos_renyi(N=1000, p=(5/999), plot=False)
G_3_2 = erdos_renyi(N=1000, p=(10/999), plot=False)
G_3_3 = erdos_renyi(N=1000, p=(50/999), plot=False)

S_3_1 = shannon_entropy(G_3_1)
S_3_2 = shannon_entropy(G_3_2)
S_3_3 = shannon_entropy(G_3_3)

print("Entropia de Shannon para <k>=5: {:.2f}".format(S_3_1))
print("Entropia de Shannon para <k>=5: {:.2f}".format(S_3_2))
print("Entropia de Shannon para <k>=5: {:.2f}".format(S_3_3))
```

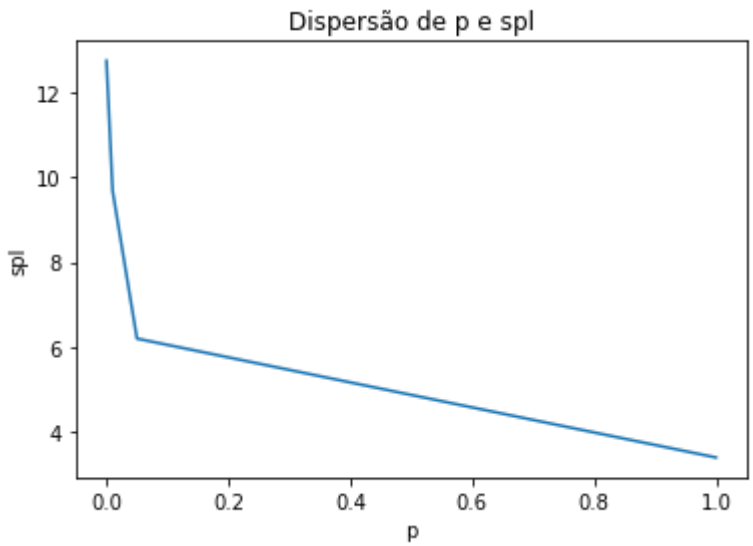
```
Entropia de Shannon para <k>=5: 3.11
Entropia de Shannon para <k>=5: 3.61
Entropia de Shannon para <k>=5: 4.81
```

4 - Para o modelo small-world, calcule o valor da menor distância média (average shortest path) para p=0; p=0.01; p=0.05 e p=0.1. Considere grau médio igual a 4 e N = 100.

In [47]:

```
list_p = [0, 0.01, 0.05, 1]
list_spl = []

for p in list_p:
    _list_spl_sample = []
    for sample in range(30):
        _G = watts_strogatz(N=100, avg_deg=4, p=p, plot=False)
        _list_spl_sample.append(avg_shortest_path(_G))
    list_spl.append(np.mean(_list_spl_sample))

for i in range(len(list_p)):
    print("N=100, p={:.2f}, spl={:.2f}".format(list_p[i], list_spl[i]))

simple_plot2d(list_p, list_spl, "p", "spl")
```

```
N=100, p=0.00, spl=12.75
N=100, p=0.01, spl=9.69
N=100, p=0.05, spl=6.20
N=100, p=1.00, spl=3.40
```

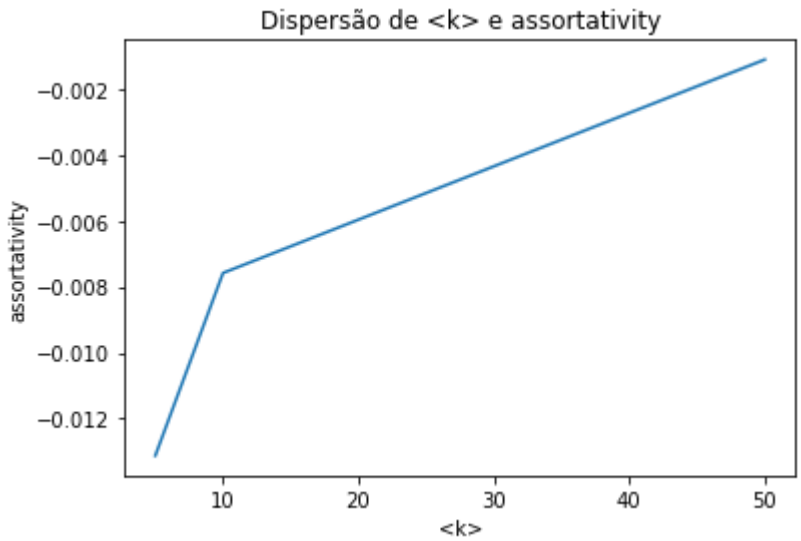


5 - Considere o modelo de Erdos-Renyi. Gere redes com grau médio igual a 5, 10 e 50 e N=1000. Qual o valor da assortatividade?

In [50]:

```
list_k = [5, 10, 50]
list_assortativity = []

for k in list_k:
    _G = erdos_renyi(N=1000, p=(k/999), plot=False)
    list_assortativity.append(nx.degree_assortativity_coefficient(_G))

for i in range(len(list_k)):
    print("N=1000, k={:.1f}, spl={:.1f}".format(list_k[i], list_assortativity[i]))

simple_plot2d(list_k, list_assortativity, "<k>", "assortativity")
```

```
N=1000, k=5.0, spl=-0.0
N=1000, k=10.0, spl=-0.0
N=1000, k=50.0, spl=-0.0
```



6 - Considere o modelo small-world. Gere redes com grau médio 10 e N=1000. Qual o valor da assortatividade para p=0.01; 0.05 e 0.2?

In [56]:

```
list_p = [0.01, 0.05, 0.2]
list_assortativity = []

for p in list_p:
    _G = watts_strogatz(N=1000, avg_deg=10, p=p, plot=False)
    list_assortativity.append(nx.degree_assortativity_coefficient(_G))

for i in range(len(list_k)):
    print("N=1000, kp{:.1f}, spl={:.1f}".format(list_p[i], list_assortativity[i]))

simple_plot2d(list_k, list_assortativity, "<k>", "assortativity")
```

```
N=1000, kp0.0, spl=-0.0
N=1000, kp0.1, spl=-0.0
N=1000, kp0.2, spl=-0.0
```