Trabalho 4 de Econofísica

May 26, 2021

Edelson Luis Pinheiro Sezerotto Júnior, 288739

0.1 Introdução

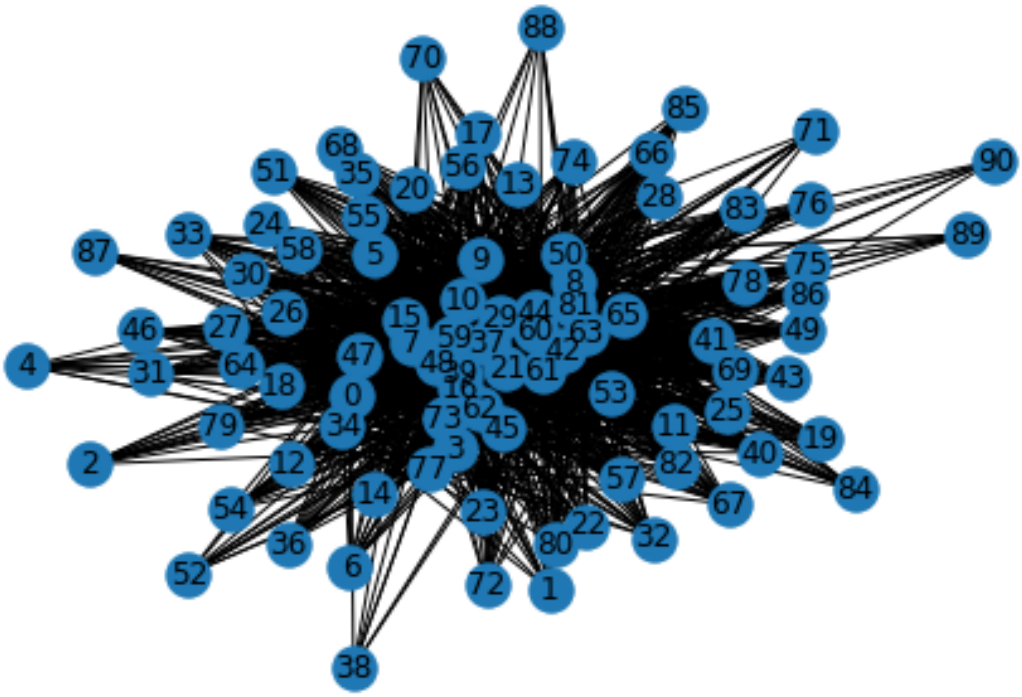
Neste trabalho vamos calcular várias medidas importantes para um grafo G obtido da referência [1]. Um grafo é uma estrutura matemática na qual temos um conjunto de agentes (representados por pontos chamados vértices) relacionados entre si (essas relações são representadas por linhas ligando os pontos, chamadas de arestas). Podemos visualizar a estrutura de G usando o código abaixo:

[1]:
```
import pandas as pd
import numpy as np
import networkx as nx

#gerando o dataframe
df = pd.read_csv('grafo.csv',delimiter=' ', header=None) #data frame da serie␣
↪temporal
df = df[[0,1]]
df.columns = ['i', 'j']
df_copy = df.copy()
qv = len(df.stack().to_frame()[0].value_counts()) #qtd de vertices

#manipulando os dados
l = df.values.tolist() #lista do data frame
laux = [] #lista auxiliar
for i in l:
    if ([i[0],i[1]] not in laux) == True: laux.append([i[0],i[1]])
    if ([i[1],i[0]] not in laux) == True: laux.append([i[1],i[0]])
df = pd.DataFrame(laux,columns=['i','j'])

#desenhando a rede
rede = nx.Graph()
rede.add_edges_from(laux)
nx.draw(rede, with_labels=True)
```



Existem diversos sistemas (biológicos, sociais, físicos etc.) que podem ser modelados por esse tipo de estrutura, como por exemplo, as amizades em uma rede social: as pessoas nesse caso são vértices, e se duas pessoas são amigas, elas podem ser associadas por uma aresta. Há muitas medidas que podemos tirar de um grafo para entendermos melhor as relações que ele modela. O grafo aqui em estudo se refere a redes cerebrais, mas as medidas que vamos calcular podem ser interpretadas do ponto de vista de qualquer outro sistema. Vamos agora explicar os conceitos teóricos associados a essas medidas, e depois implementá-las na prática.

0.2 Conceitos teóricos

0.2.1 Coeficiente de Clusterização Global

Quando temos três vértices ligados entre si, dizemos que temos um triângulo. Por exemplo, um triângulo aparece se tivermos o vértice 1 ligado ao 2, que por sua vez está ligado ao 3, que está ligado ao 1 - ou seja, temos o conjunto de ligações $(1,2)$, $(1,3)$ e $(2,3)$. Se tivermos uma sequência de duas arestas adjacentes, mas sem formarem um triângulo, dizemos que temos um triplet - um exemplo de triplet seria o conjunto de conexões $(1,2)$ e $(1,3)$, sem a conexão $(2,3)$. Podemos representar a quantidade total de triângulos por Δ e a de triplets por T. Assim, o Coeficiente de Clusterização Global C_g é definido como

$$C_g = \frac{3\Delta}{T} \tag{1}$$

ou seja, ele é uma medida da tendência de dois vértices se unirem, uma vez que eles estejam ligados a um vértice em comum (no contexto de redes sociais, seria uma medida da tendência de duas

pessoas se tornarem amigas uma vez que elas tenham um amigo em comum). O cálculo de C_g é bastante direto: contamos a quantidade de triângulos e de triplets e aplicamos a equação (1).

0.2.2 Coeficiente de Clusterização Local

O Coeficiente de Clusterização Local $C_l(v)$ para um certo vértice v é uma medida da tendência dos viznhos de v (isto é, os vértices conectados a ele) se unirem entre si. Ele é calculado pela expressão

$$C_l(v) = \frac{L(v)}{L_p(v)} \tag{2}$$

onde $L(v)$ é a quantidade de ligações existentes entre os vzinhos de v e $L_p(v)$ é a quantidade total de possíveis ligações que poderiam existir entre eles. $L(v)$ é obtido numericamente, contando-se cada ligação individual; já $L_p(v)$ pode ser determinado analiticamente a partir do grau k do vértice v, que representa sua quantidade de vizinhos. Isso pode ser feito da seguinte maneira: vamos supor que $v = 1$ e que para esse vértice temos as ligações $(1, 2)$, $(1, 3)$, $(1, 4)$, $(1, 5)$, $(1, 6)$ e $(1, 7)$ - temos, portanto, $k = 6$. Agora, para o vértice 2 poderíamos ter as ligações $(2, 3)$, $(2, 4)$, $(2, 5)$, $(2, 6)$ e $(2, 7)$, nos dando $k - 1 = 5$ ligações; para o 3 poderíamos ter $(3, 4)$, $(3, 5)$, $(3, 6)$ e $(3, 7)$ (repare que a dupla $(3, 2)$ não foi contada porque estaríamos repetindo a ligação $(2, 3)$), nos dando $k - 2 = 4$ ligações; e assim por diante, até chegarmos ao vértice 6, para o qual teríamos uma única ligação $(6, 7)$. A quantidade total de ligações é a soma das quantidades para cada vértice, o que nos dá

$$L_p(1) = 1 + 2 + \cdots + k - 1$$

Generalizando o procedimento descrito no parágrafo anterior, obtemos:

$$L_p(v) = \sum_{i=1}^{k-1} i = \frac{(1 + k - 1)(k - 1)}{2} = \frac{(k - 1)k}{2} \tag{3}$$

onde utilizamos a conhecida equação da soma dos termos de uma progressão aritmética para obter a equação (3).

0.2.3 Centralidade

A centralidade $C(v)$ de um certo vértice é definida como

$$C(v) = \frac{k(v)}{N - 1} \tag{5}$$

onde N é a quantidade total de vértices do grafo. Ou seja, é uma medida da fração de vizinhos que o vértice v tem em relação à quantidade total de vizinhos que ele poderia ter - quanto maior for $C(v)$, mais o vértice v está ligado aos demais vértices. Para uma rede pouco robusta, esperamos ter poucos vértices com uma alta centralidade e a maioria com centralidades baixas, de forma que toda a estrutura fica conectada por esses poucos vértices centrais; e vice-versa para uma rede mais robusta, de forma que mesmo tirando um vértice com alta centralidade, a rede se mantém porque temos muitos vértices ligados entre si.

0.2.4 Assortatividade

A assortatividade A do grafo G é uma medida da tendência de que vértices de graus semelhantes se conectem entre si. Usando a referência [4], ela pode ser calculada pela expressão

$$A = \frac{1}{\sigma_1 \sigma_2} \sum_{k_1,k_2} k_1 k_2 [P(k_1, k_2) - P(k_1)P(k_2)] \tag{6}$$

onde $P(k_1, k_2)$ representa quantas vezes ocorre uma aresta em que um dos vértices tem grau k_1, dividido pelo número de arestas; $P(k_1)$ representa a quantidade de arestas associadas a vértices de grau k_1, dividido pelo número de arestas (e analogamente para $P(k_2)$); σ_1 é o desvio padrão da distribuição de valores k_1, e σ_2 é para os valores k_2. A pertence ao intervalo [-1,1], de forma que se $A > 0$, significa que os vértices tendem a se conectarem com outros com valores próximos de k, e se $A < 0$ significa que os vértices tendem a se conectarem com vizinhos de graus diferente (vértices de alto grau se conectam com os de pequeno grau e vice-versa).

0.2.5 Menor Distância Média

A Menor Distância Média S (em inglês, Average Shortest Distance) é o valor médio das menores distâncias entre quaisquer dois vértices. De acordo com a referência [3], ela pode ser calculada pela expressão

$$S = \frac{\sum_{i \neq j} d(v_i, v_j)}{N(N-1)} \tag{7}$$

onde $d(v_i, v_j)$ é a menor distância entre os vértices 1 e 2 (note que $N(N-1)$ é a quantidade total de distâncias consideradas). Para obtermos o conjunto de menores distâncias a partir de um certo vértice i foi usado o algoritmo apresentado na referência [2], que funciona da seguinte maneira: selecionamos um vértice de interesse v_i e verificamos quais são seus vizinhos. Para ilustrar, vamos considerar que $v_i = v_1$ e seus vizinhos são dados pelo conjunto $n_1 = \{v_2, v_3, v_4, v_5\}$. Assim, a menor distância de v_1 até qualquer vértice pertencente a n_1 vale 1. Agora, analisamos cada vizinho de v_1, começando por v_2 - digamos que $n_2 = \{v_5, v_6, v_7\}$. Sabemos então que para chegar de v_1 até qualquer vizinho de v_2, precisamos de dois passos ($d = 2$); porém, para o elemento v_5, poderíamos chegar usando apenas um passo. Nesse caso, consideramos que a menor distância para v_6 e v_7 vale 2, mas para v_5 continua valendo 1. E assim vamos repetindo esse procedimento para todos os vizinhos de v_1, e depois vamos para os vizinhos dos vizinhos, até mapear o grafo inteiro. Com isso, concluímos a exposição dos conceitos teóricos - vejamos agora como implementar as medidas descritas.

0.3 Implementação e resultados

0.3.1 Calculando C_g

Inicialmente, obtemos T (a quantidade de triplets):

[2]:
```
#obtendo a qtd de triplets
lv = df['i'].value_counts().index[:qv].tolist() #lista contendo os vertices
lv2 = lv.copy()
lk = [] #lista dos graus
```

```
for i in range(len(lv)):
    lk.append(len(df.loc[df['i'] == lv[i]])) #data frame das arestas do vertice
 ↪i
    df.drop(index=df[df['i'] == lv[i]].index, inplace=True)

lplev = [] #lista das qtds de possiveis ligacos entre vizinhos
for k in lk:
    plev = (k-1)*k #possiveis ligacoes entre vizinhos
    lplev.append(int(plev/2))
qtp = sum(lplev) #qtd de triplets
```

Agora obtemos a quantidade de triângulos:

[3]:
```
qtg = 0 #qtd de triangulos
df = pd.DataFrame(laux,columns=['i','j'])
ldf = [] #lista dos dataframes dos vertices
for i in lv:
    dfi = df.loc[df['i'] == i]
    ldf.append([i,dfi['j']])
    df.drop(index=df[df['i'] == i].index, inplace=True)

for dv in laux:
    for item in range(len(ldf)):
        if ldf[item][0] == dv[1]:
            lvi = ldf[item][1].to_frame()['j'].values #viznhos atuais
            break

    for i in lvi:
        if ([dv[0],i] in laux) == True: qtg += 1

qtg /= 6
```

Notamos que com o método usado nesse código, cada triângulo é contado 6 vezes. Isso acontece pelo seguinte motivo: consideremos o triângulo $\{(1,2),(2,3),(3,1)\}$. Esse conjunto de valores - partindo do vértice 1, indo para o 2, depois para o 3, e depois voltando para o 1 - vai ser contado novamente usando o percurso contrário, ou seja, através do conjunto $\{(1,3),(3,2),(2,1)\}$, e a contagem será feita novamente duas vezes partindo do vértice 2 e mais duas vezes partindo do vértice 3. Esse algoritmo não é otimizado, mas está de acordo com a proposta do trabalho de entender a maneira como os cálculos são feitos. Podemos verificar se os cálculos foram feitos corretamente usando a biblioteca NetworkX:

[5]:
```
number_of_triangles = sum(nx.triangles(rede).values())/3 #numero real de
 ↪triangulos
teste = (qtg/number_of_triangles) #se teste == 1 a qtd de triangulos esta certa
print(teste)
```

```
1.0
```

A variável "teste" armazena a razão entre o Δ calculado pelo código desenvolvido no trabalho e aquele calculado pela biblioteca. Obtemos uma razão valendo 1, o que indica que os cálculos estão corretos. Finalmente, obtemos C_g com o trecho abaixo:

[6]:
```
Cg = 3*qtg/qtp #coeficiente de clusterizacao global
print(Cg)
```

```
0.49526425913207367
```

0.3.2 Calculando $\overline{C_l(v)}$, $\overline{C(v)}$ e A

Vamos obter os valores médios dos coeficientes de cliusterização locais $C_l(v)$ e das centralidades $C_(v)$, e também calcular a assortatividade A. Primeiro, criamos um data frame para cada vértice:

[7]:
```
lCl = [] #lista de coeficientes de clusterizacao locais
df = pd.DataFrame(laux,columns=['i','j'])

dfaux = pd.DataFrame(lv2,columns=['lv2'])
dfaux['k'] = lk
dfaux['plev'] = lplev
dfaux = dfaux.sort_values('lv2')

lv = dfaux['lv2'].tolist() #lista de vertices

del ldf
ldf = []
for i in lv: ldf.append([i,df.loc[df['i'] == i]['j']])
llev = [] #lista de ligacoes entre vizinhos
```

Agora contamos as ligações entre vizinhos, para todos os vértices:

[8]:
```
for v in lv:
    for item in range(len(ldf)):
        if ldf[item][0] == v:
            lvi = ldf[item][1].to_frame()['j'].values #viznhos atuais
            break

    lev = 0 #ligacoes entre vizinhos
    vpe = [v] #vizinhos para eliminar

    for v2 in lvi:
        for item in range(len(ldf)):
            if ldf[item][0] == v2:
                lvj = ldf[item][1].to_frame()['j'].values #viznhos atuais
                break

        for item in vpe: lvj = lvj[lvj != item]
```

```
        for item in lvj:
            if (item in lvi): lev += 1

        vpe.append(v2)

    llev.append(lev)
```

E daí obtemos $C_l(v)$ para cada vértice, a partir do qual obtemos $\overline{C_l(v)}$:

[9]:
```
dfaux['lev'] = llev
dfaux['Cl'] = dfaux['lev']/dfaux['plev']
Clm = np.mean(dfaux['Cl']) #coeficiente de clusterizacao local medio
print(Clm)
```

```
0.742395429494937
```

A centralidade média e a assortatividade são calculadas com o trecho abaixo:

[11]:
```
#obtendo as centralidades
dfaux['Cent'] = dfaux['k']/(len(lv2)-1) #centralidades
Centm = np.mean(dfaux['Cent']) #centralidade media

#obtendo a assortatividade
A = nx.degree_assortativity_coefficient(rede) #assortatividade

print(Centm)
print(A)
```

```
0.3421245421245422
-0.5534348420390326
```

O valor negativo para A nos indica que os vértices tendem se conectar a outros que tenham graus diferentes deles próprios. Por fim, vamos implementar o cálculo de S.

0.3.3 Calculando S

A função a seguir toma como entrada um vértice de interesse, e nos retorna a soma das menores distâncias desse vértice em relação a todos os outros:

[15]:
```
def path_length(v): #v eh o numero do vertice
    dfpl = pd.DataFrame(lv,columns=['vertice']) #data frame dos path lengths
    dfpl['distancia'] = np.inf
    dfpl.loc[dfpl.vertice == v, 'distancia'] = 0 #distancia de um vertice ate␣
 ↪ele mesmo
    d = 0 #distancia atual
    cda = 1 #contador de distancias atualizadas a cada iteracao

    while cda > 0:
        #obtendo os vertices de distancia 1
```

```
        cda = 0 #contador de distancias atualizadas
        lj = np.array(dfpl.loc[dfpl['distancia'] == d]['vertice'])
        d += 1

        for j in lj:
            for item in range(len(ldf)):
                if ldf[item][0] == j:
                    lvi = ldf[item][1].to_frame()['j'].values #vizinhos atuais
                    break

            for i in lvi:
                if np.float(dfpl.loc[dfpl['vertice'] == i]['distancia']) > d:
                    dfpl.loc[dfpl.vertice == i, 'distancia'] = d
                    cda += 1

    dfpl.loc[dfpl.distancia == np.inf, 'distancia'] = 0
    sd = np.sum(dfpl['distancia']) #soma das distancias
    return sd
```

Agora chamamos a função acima para todos os vértices; somamos todas as distâncias e dividimos pela quantidade de termos para obter a Menor Distância Média:

[17]:
```
sd = 0
for v in lv: sd += path_length(v)
aspl = sd/(len(lv)*(len(lv)-1)) #average shortest path length
print(aspl)
```

```
1.6581196581196582
```

Podemos ainda verificar a razão entre o A calculado com o código desenvolvido nesse trabalho e aquele calculado pela biblioteca NetworkX:

[20]:
```
aspl_real = nx.average_shortest_path_length(rede)
teste_aspl = aspl/aspl_real
print(teste_aspl)
```

```
1.0
```

A razão vale 1, indicando que o cálculo foi feito corretamente.

0.4 Referências

[1] http://networkrepository.com/bn-macaque-rhesus-cerebral-cortex-1.php

[2] https://www.youtube.com/watch?v=aU_qPL6VN6o

[3] http://qasimpasta.info/data/uploads/sina-2015/calculating-average-path-length.pdf

[4] http://folivetti.github.io/courses/ComunicacaoRedes/PDF/AULA%2009.pdf