Seminario de Modelos y Métodos Cuantitativos
Tarea 2-3

Universidad Técnica Federico Santa María - Campus San Joaquín

10 de diciembre de 2015

Profesor Andrés Moreira

Juan Pablo Escalona G.
juan.escalonag@alumnos.usm.cl
201073515-k

Alonso Lepe M.
alonso.lepe@alumnos.usm.cl
201173593-5

1. Pregunta 1

Se tiene un grafo $G(n = 400, p = 1)$ esto implica que hay 79800 aristas que conectan a los 400 nodos entre todos. i.e. $\frac{399 \cdot 400}{2} = 79800$

Para que una componente conexa sea gigante debe tener un $k = np > 1$, en este caso $k = 400$, como se necesita un $k <= 1$ para que deje de ser gigante, entonces p debería ser a lo mas $\frac{1}{400} = 0{,}0025$

Si se elimina una arista en el segundo 00:00:01 entonces aún quedan 79799 aristas, es decir, una probabilidad de $p = \frac{79799}{79800} = 0{,}9999$. Pero se sabe que se necesita una probabilidad de 0.0025, por lo que la componente dejará de ser conexa cuando solo hayan $\frac{E}{79800} = 0{,}0025 \implies E \sim 200$ aristas.

Luego si cada segundo se elimina 1 arista y es preciso eliminar 79600 aristas entonces se tendría que esperar hasta las 22:06:40 (10 de la noche, 6 minutos y 40 segundos)

Sobre este análisis se realizó el código 1 del anexo en el que se crea una red ER de 400 nodos y probabilidad 1. Luego se comenzó a eliminar nodos hasta lograr un grado promedio ¡1. Se ejecutó el código en un computador con un 2.5 GHz Intel Core i5 y 8 GB ram, tardó 8 minutos 29 segundos en eliminar aristas hasta lograr un grado promedio de 1. Finalmente se eliminaron 79600 aristas por lo que empíricamente también terminó a las 22:06:40

2. Pregunta 2

La Matriz laplaciana de un grafo se puede obtener a partir de la matriz de adyacencia siguiendo el siguiente formato:

$$l_{i,j} = \begin{cases} k_i & \text{si } i = j \\ -1 & \text{si } i \neq j \\ 0 & \text{e.t.o.c.} \end{cases} \tag{1}$$

que es lo mismo que decir $l = K - A$ donde K es la matriz diagonal de los grados de cada nodo y A es la matriz de adyacencia. Luego para la matriz de adyacencia dada se tiene la siguiente matriz laplaciana:

$$l = \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 \\ -1 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & -1 & -1 & 2 \end{bmatrix} \tag{2}$$

l valor de Fiedler es el segundo valor propio mas pequeño, en este caso corresponde a 0.438 ubicado en la tercera columna como se muestra en las ecuaciones (3) y (4).

$$V = \begin{bmatrix} 0{,}185 & -0{,}408 & \mathbf{0.465} & 0{,}378 & -0{,}756 & 0{,}040 \\ 0{,}185 & -0{,}408 & \mathbf{0.465} & -0{,}734 & 0{,}463 & 0{,}238 \\ -0{,}657 & -0{,}408 & \mathbf{0.261} & 0{,}357 & 0{,}292 & -0{,}278 \\ 0{,}657 & -0{,}408 & \mathbf{-0.261} & 0{,}357 & 0{,}292 & -0{,}278 \\ -0{,}185 & -0{,}408 & \mathbf{-0.465} & -0{,}189 & -0{,}148 & -0{,}472 \\ -0{,}185 & -0{,}408 & \mathbf{-0.465} & -0{,}168 & -0{,}145 & 0{,}751 \end{bmatrix} \tag{3}$$

$$E = \begin{bmatrix} 4{,}562 & 0{,}000 & \mathbf{0.438} & 3{,}000 & 3{,}000 & 3{,}000 \end{bmatrix} \tag{4}$$

Es importante ordenarlos para obtener el valor de Fiedler, pues este es el segundo menor valor propio.

$$\text{Valor de Fiedler: } 0{,}4384 \tag{5}$$

$$\text{Vector de Fiedler: } \begin{bmatrix} 0{,}465 & 0{,}465 & 0{,}261 & -0{,}261 & -0{,}465 & -0{,}465 \end{bmatrix} \tag{6}$$

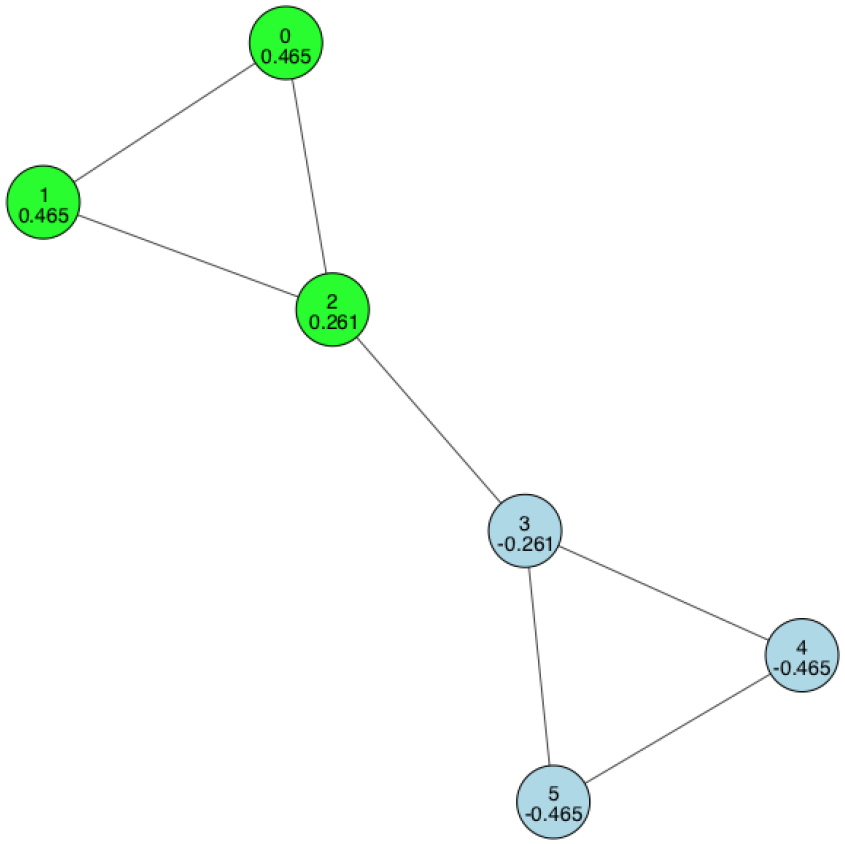

Figura 1: Partición de la red en dos comunidades mostrando sus respectivos valores del vector propio.

Se decidió cortar en el 0, pues los valores se separan uniformemente en torno al este. Se observan claramente dos comunidades en esta red las que interactúan únicamente por el vértice 2-3.

3. Pregunta 3

Dos nodos de un grafo forman un *2-componente* si existen al menos dos caminos independientes entre ellos. Un camino es una secuencia finita de aristas que permiten ir de un nodo inicial a un nodo final dentro de un grafo.

Un *2-componente* o *biconnected component* tienen la particularidad que si se elimina una arista del grafo, aún sería posible llegar de un nodo a otro, pues originalmente existían dos posibles caminos.

Por otro lado un *2-core* es un conjunto de nodos donde cada uno está conectado a los demás con al menos 2 aristas. Un *2-core* solo considera los nodos que estén conectados con una arista dentro del conjunto, si algún nodo tiene una arista a un nodo fuera del conjunto, dicha arista no se cuenta.

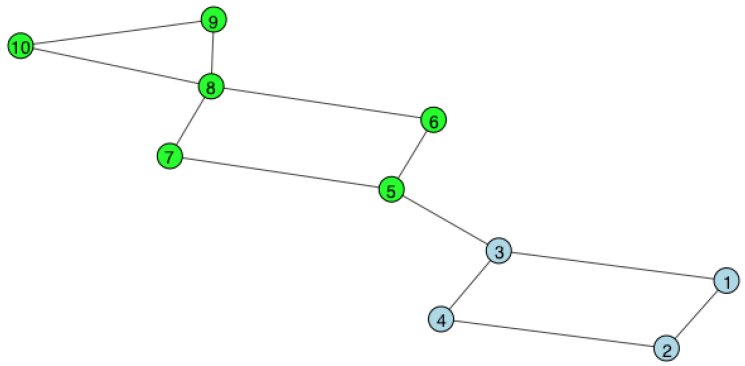

Figura 2: Grafo 2-core con dos 2-componentes.

En la figura 2 se aprecia un grafo con un solo conjunto *2-core* con la totalidad de los nodos, pues todos los nodos del grafo están conectados a los demás con al menos 2 aristas.

Por otra parte existen dos *2-componentes* destacados por los colores. En este caso el *2-componente* $[1, 2, 3, 4]$ y el componente $[5, 6, 7, 8, 9, 10]$. No puede existir un 2-componente entre el nodo 1 y el nodo 8, pues si o si hay que pasar por la arista 3-5 por lo que sería el mismo camino. Si se quita dicha arista no hay otra forma de llegar de 1 a 8.

4. Pregunta 4

4.1. Gráfico de la red

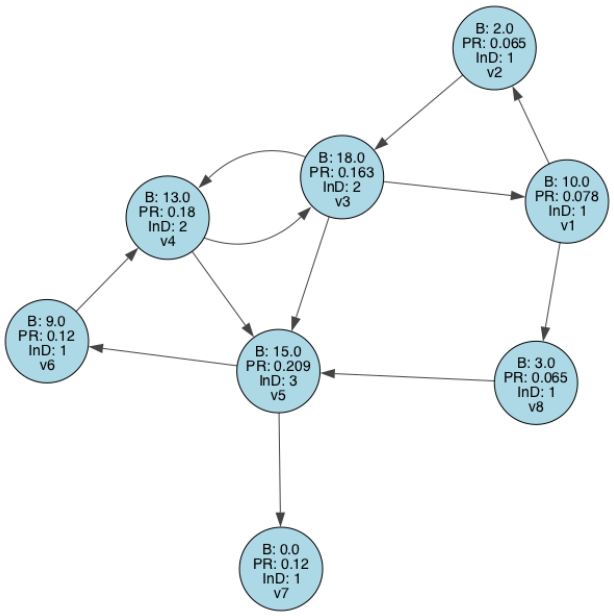

Figura 3: Red chica dirigida. B: Betweenness, PR: PageRank, InD: Grado de entrada.

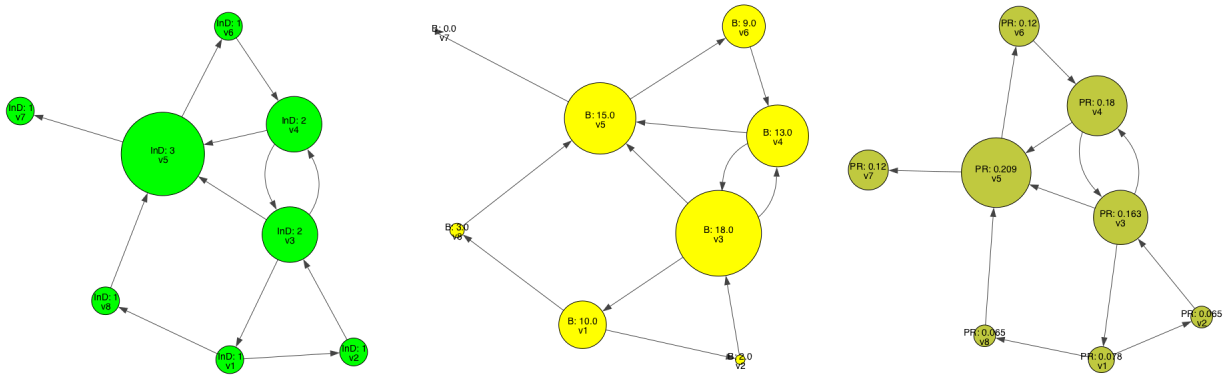

Figura 4: Red chica según el tamaño porcentual para cada indice.

4.2. Ranking de los índices

Grado Entrada			Betweenness			PageRank		
Posición	Nodo	Valor	Posición	Nodo	Valor	Posición	Nodo	Valor
1	v5	3	1	v3	18	1	v5	0.209
2	v4	2	2	v5	15	2	v4	0.180
2	v3	2	3	v4	13	3	v3	0.163
3	v1	1	4	v1	10	4	v6	0.120
3	v2	1	5	v6	9	4	v7	0.120
3	v6	1	6	v8	3	5	v1	0.078
3	v7	1	7	v2	2	6	v2	0.065
3	v8	1	8	v7	0	6	v8	0.065

Cuadro 1: Ranking de cada índice para la red chica.

Lo más evidente que se observa del Cuadro 1 asi como también en la Figura 4 es que los nodos v3, v4 y v5 están siempre en los primeros 3 puestos sin excepción, i.e. tienden a ser los nodos más relevantes de la red independiente del índice que se utilice para rankearlos.

Un aspecto interesante es la importancia del nodo v7, en el caso del grado de entrada el nodo v7 tiene solo 1 nodo de entrada por lo que su importancia es pequeña, en PageRank por otro lado tiene un nodo de entrada, pero esta arista proviene del nodo más importante de la red por lo que v7 adquiere una mayor importancia que los otros nodos que solo tienen grado de entrada 1.

Curiosamente el betweenness del nodo v7 es el más pequeño de todos en dicho ranking, pues no existen caminos que pasen por el. Es una importante diferencia entre Betweenness y PageRank. PageRank considera la importancia de un nodo según la importancia del nodo desde donde se llega, más aún si el enlace es mutuo, esto por la frecuencia con que se visitará dicho nodo al azar. Betweenness al contrario solo considera la cantidad de caminos que utilizan dicho nodo.

El más simple de todos es el grado de entrada, se observa que es muy similar al resultado obtenido en PageRank, pero PageRank le da más importancia a aquellos que llegan de un nodo importante, no solo de la cantidad de referencias que tenga.

El nodo v1 es muy similar en grado de entrada y PageRank, pero betweenness le asigna una gran importancia, siendo que tan solo tiene 1 grade de entrada, esto se podría explicar por la cantidad de caminos que permite construir ya que tiene dos aristas de salida, parece ser que tendría un gran flujo, menor que su predecesor el nodo v3.

5. Pregunta 5

Para esta pregunta se programaron los scripts 5, 6, 7 y 8. En cada caso se carga una red diferente, gnutella, delfines y Erdös-Renyi. Luego para cada red se eliminan los nodos al azar, en orden decreciente de grado y en orden decreciente de betweenness. Luego se calcula el porcentaje de nodos que fue necesario eliminar para que el tamaño de la componente gigante se disminuyera a la mitad.

Red	Azar	Grado Decreciente	Betweenness Decreciente
Gnutella	33.208 %	3.019 %	3.145 %
Delfines	38.710 %	24.194 %	12.903 %
Erdös-Renyi Gnutella	30.566 %	9.128 %	5.786 %
Erdös-Renyi Delfines	46.774 %	32.258 %	27.419 %

Cuadro 2: Porcentaje de nodos eliminados para cada red y para cada heurística de eliminación. Los valores al azar y las redes Erdös-Renyi son el promedio de 25 ejecuciones cada uno.

La red GNUTELLA está compuesta de 795 nodos y 852 aristas mientras que la red DELFINES está compuesta por 62 nodos y 159 aristas. Para ambas redes se realizó una ER con la misma cantidad de nodos y aristas para comparar.

El resultado general mas destacable es que eliminar nodos al azar no es un buen método de reducción de componentes gigantes, pues la cantidad de nodos necesarios a eliminar variaba entre un 30 % y un 47 %, valores muy elevados, pues se está eliminando casi la mitad de la red para deshacerse de la componente gigante.

Por el contrario el método que presenta los mejores resultados tiene ser el de eliminación por betweenness decreciente. Para redes grandes como GNUTELLA basta eliminar solo el 3.145 %, un valor similar fue necesario para la ER correspondiente.

El método del grado decreciente se comporta bien en redes grandes al igual que el método de betweennes. Ambos métodos caen en redes mas pequeñas como la de DELFINES, en donde se tiene que eliminar 24.193 % de los nodos con el método del grado y 12.903 % en el caso del betweenness. Estos valores altos se replican en la ER correspondiente a delfines, en donde para todos los métodos se tienen valores superiores a los 27 %.

6. Pregunta 6

7. Pregunta 7

Ahora se trabaja con la red definida "correos.gdf" la cual muestra los intercambios de correo electrónico al interior de una universidad catalana. Se obtuvieron los siguientes resultados.

Correos			Correos Recableada		
k-core	nodos	vértices	k-core	nodos	vértices
1-core	1133	5451	1-core	1133	5451
2-core	1133	5451	2-core	1133	5451
3-core	978	5296	3-core	979	5297
4-core	848	5040	4-core	853	5047
5-core	748	4746	5-core	747	4733
6-core	665	4422	6-core	668	4422
7-core	534	3796	7-core	580	3986
8-core	434	3224	8-core	486	3435
9-core	349	2681	9-core	383	2729
10-core	238	1837	10-core	197	1313
11-core	121	845			
12-core	12	66			

Cuadro 3: Porcentaje de nodos eliminados para cada red y para cada heurística de eliminación. Los valores al azar y las redes Erdös-Renyi son el promedio de 25 ejecuciones cada uno.

Para la red original se obtuvo una Modularidad de 0.507 y una Asortatividad de 0.078 mientras que para la red recableada se obtuvo una Modularidad de 0.282 y una Asortatividad de -0.009

Lo primero que resalta es que los 1 y 2 core son el mismo, pues tienen igual número de nodos y aristas. Luego a medida que aumenta el k-core los componentes son cada vez mas pequeños como es de esperar. Pareciera haber una tendencia lineal en el número de nodos y algo mas lenta en el numero de aristas pero que cae abruptamente al llegar al 12-core, las aristas decaen de 845 a 66, mientras que en componentes mas grandes las aristas bajaban en el orden de 10^2.

La modularidad de la red original fue de 0,507 lo que indica una fuerte tendencia a la existencia de comunidades en la red. En la red recableada se obtuvo un valor de modularidad de 0.282 lo que es menor a 0.3, indicador de que no existen comunidades en la red.

Con respecto a la Asortatividad en la red original se obtuvo un valor de 0.078 lo cual indica que no existe una preferencia de los nodos para unirse con nodos de grado similar, i.e. los que mas intercambian emails no forman comunidad entre ellos, sino que se comunican todos con todos pero no en la misma cantidad.

Para observar posibles comunidades se utilizó gephi sobre la red original con el algoritmo Chinese Whispers. Se encontraron una gran comunidad, una comunidad mediana y varias otras pequeñas de hasta 30 nodos.

Figura 5: Visualizacion de comunidades en la red original con el algoritmo chinese whispers

La figura 5 apoya el resultado de la modularidad obtenida en la red original que es probable que existan comunidades.

8. Pregunta 8

Para graficar las curvas se realizaron 10 iteraciones de lo pedido en el enunciado, luego se calculó el promedio de las modularidades de cada iteración (para que la curva fuese más aleatoria). Haciendo esto se construyen gráficos que representan 4 grafos ER distintos:

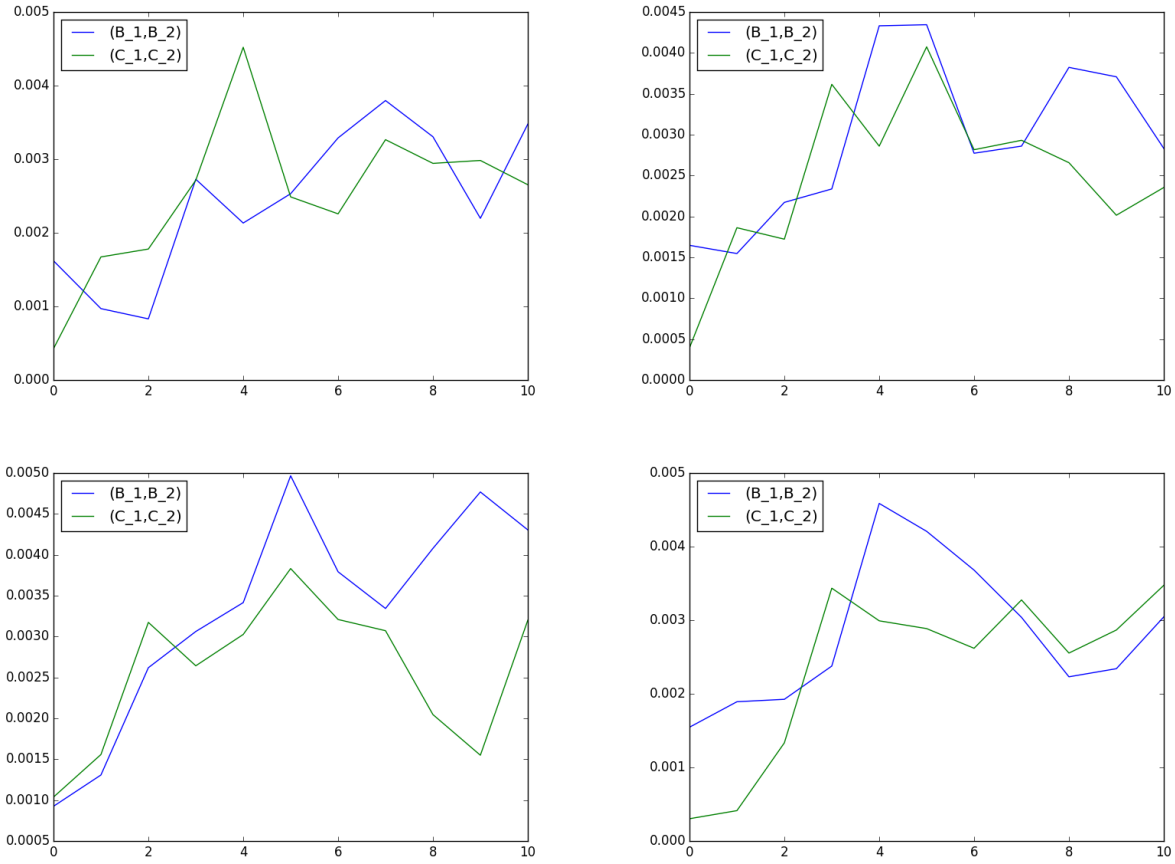

Figura 6: Gráficos del promedio de las modularidades de cuatro grafos ER dirigidos distintos (modificando las aristas)

Observando los gráficos obtenidos se aprecia que la tendencia es tener mayor modularidad en las iteraciones intermedias. Lógicamente en la iteración cero (en donde el grafo es puramente ER) la modularidad es casi nula, pues es una de las características de este tipo de grafo. Una vez que se empieza a modificar las aristas de las particiones se observa que la modularidad comienza a subir, por lo que la aparición de clusters debiese ser inminente. Por otro lado, al trabajar con promedios de modularidades se tiene una mejor aproximación del comportamiento del modelo ER al modificarlo de la manera pedida.

9. Pregunta 9

(a)

Para este experimento se utilizó la red "pescado.gml" y se pretende hallar reciprocidad corregida de la red. Para esto se procedió a encontrar la siguiente expresión:

$$\rho = \frac{\rho_1 - \bar{a}}{1 - \bar{a}} \tag{7}$$

en donde

$$\rho_1 = \frac{1}{m}\sum_{i\neq j} a_{ij}a_{ji} \tag{8}$$

$$\bar{a} = \frac{1}{n(n-1)}\sum_{i\neq j} a_{ij} \tag{9}$$

los valores encontrados fueron

- $\rho_1 = 0{,}4672$
- $\bar{a} = 0{,}1211$

luego $\rho = 0{,}3938 > 0$ lo que implica que la red presenta reciprocidad, la reciprocidad mide la tendencia de pares de nodos para formar conexiones mutuas entre ellos. En este caso se tiene una tendencia $\rho = 0{,}3938$ a formar conexiones mutuas entre pares de nodos, lo cual tiene sentido para el comercio e intercambio de pescado entre naciones.

(b)

Ahora se procede a realizar un histograma para el peso total que entra a cada nodo y el peso total que sale de un nodo. Para esto basta con calcular la suma de las filas o columnas de la matriz de adyacencia con pesos. Esto luego se plotea como histograma, los resultados se adjuntan en la Figura 7.

Inmediatamente se reconoce en ambas distribuciones una ley de potencia, hay unos pocos que tienen mucho, y hay muchos que tienen muy poco. Para hallar las ecuaciones correspondientes a cada distribución se utilizo excel para realizar la regresión potencial correspondiente dado los valores entregados por la función de histograma en matplotlib.

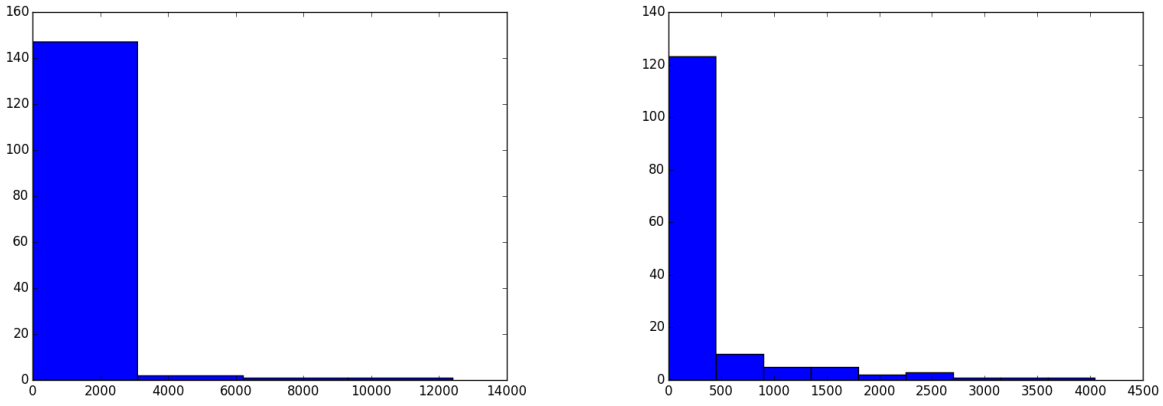

Figura 7: Histogramas para S^{in} y S^{out} respectiamente.

Las expresiones encontradas fueron:

$$f^{in}(x) = 79{,}673 \cdot x^{-3{,}722} \qquad R^2 = 0{,}86766 \tag{10}$$

$$f^{out}(x) = 73{,}115 \cdot x^{-2{,}07} \qquad R^2 = 0{,}93403 \tag{11}$$

(c)

En este experimento se intenta hallar alguna relación entre la suma de los pesos de los arcos que entran y salen de un nodo. Para esto se realiza el gráfico 8

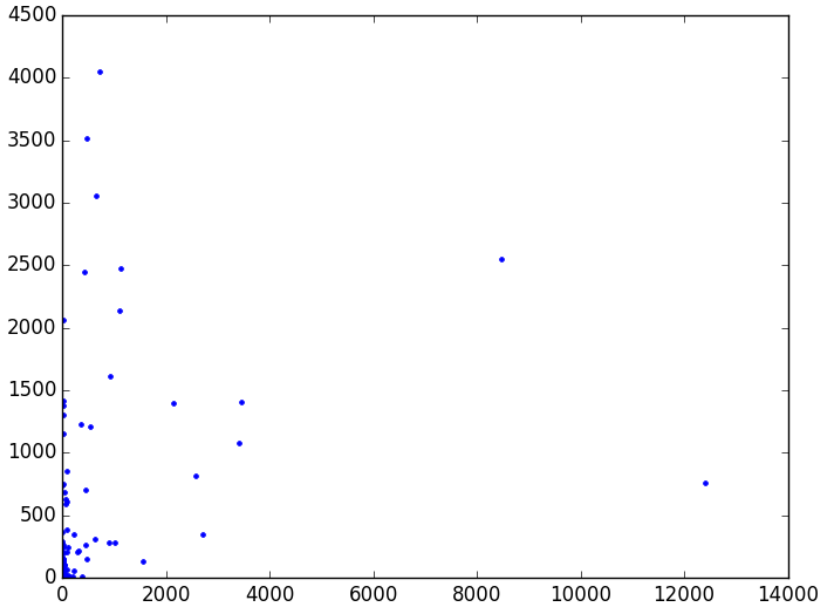

Figura 8: S^{in} vs S^{out}

En este gráfico no se observa ninguna relación entre las variables, i.e. no hay relación entre la cantidad de pescado importado y exportado en un país.

(d)

Ahora se trabaja con el grafo no dirigido en que las aristas son la suma de los pesos del grafo dirigido anterior.

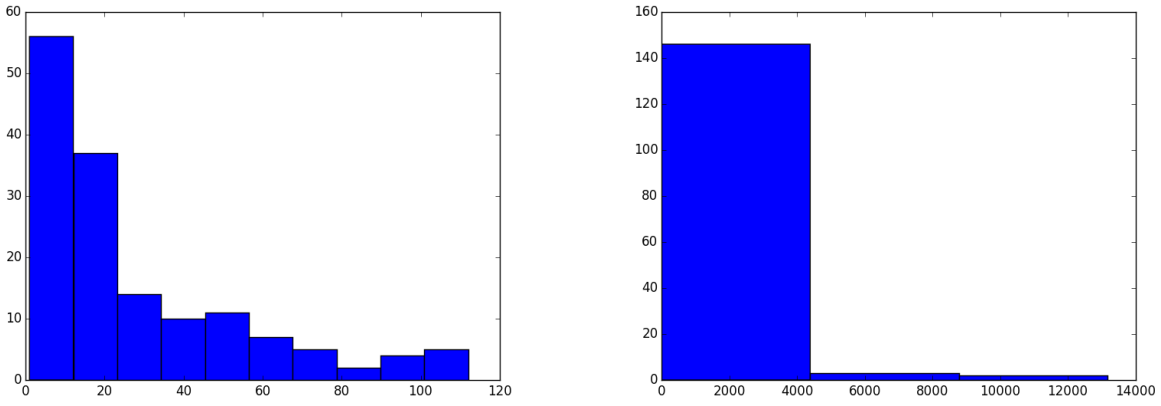

Figura 9: Histogramas para P(k), P(s)

En este caso se hallan nuevamente leyes de potencia. Las expresiones halladas fueron:

$$f^k(x) = 65{,}941 \cdot x^{-1{,}302} \qquad R^2 = 0{,}88592 \tag{12}$$

$$f^s(x) = 109{,}97 \cdot x^{-4{,}088} \qquad R^2 = 0{,}9194 \tag{13}$$

(e)

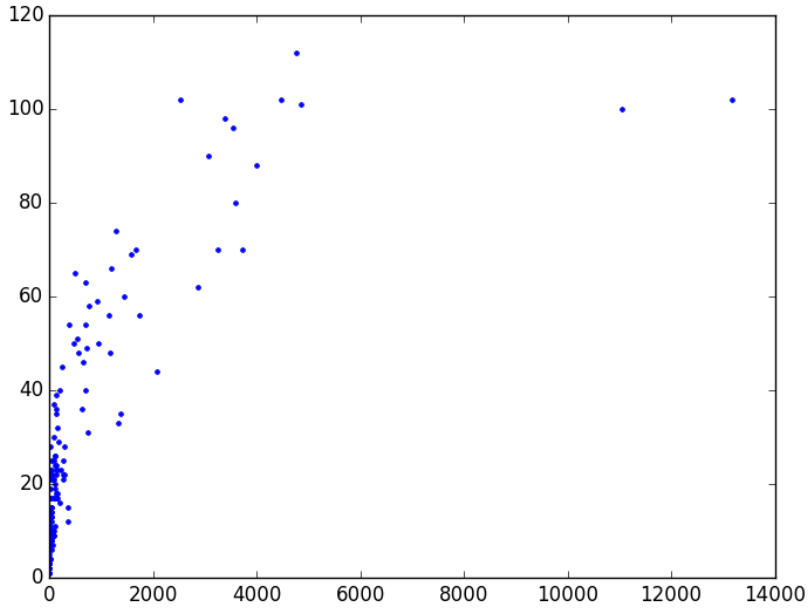

Figura 10: Grado vs fuerza de los nodos.

En este caso se observa una ley de potencia positiva, de este gráfico se obtuvo la siguiente expresión:

$$f(x) = 2{,}5736x^{0{,}4313} \qquad R^2 = 0{,}87075 \tag{14}$$

el R^2 indica la presencia de una buena relación entre las variables. Es decir, la fuerza de un nodo guarda relación con el grado de este cuando se trata de un grafo no dirigido con peso y sigue una ley de potencia.

(f)

Por ultimo se comparan los coeficientes de clustering de la red con y sin peso, para la red sin peso se obtuvo un valor de 0.784 y para la red con peso se obtuvo 0.890. Se observa un mayor numero en la red con peso, lo que indica una mayor tendencia a formar clusters en la red con peso que en la red sin peso.

El coeficiente de clustering indica cuan probable es que dos nodos vecinos estén conectados. Es la razón de los triángulos y tripletas conectadas en el grafo.

En este caso particular se observa una mayor predispocición a tripletas en la red con peso.

10. Pregunta 10

- $t_0 = 58732$
- $t_1 = 396548$
- $t_2 = 451711$
- $t_3 = 4003085$
- $m_+ = 592551$
- $m = 711782$

11. (c)

Si reemplazamos los valores anteriores se aprecia que los valores no coinciden. Si se ve desde el punto de vista en que la cantidad de aristas m_+ deber ser exactamente la suma de las mismas existentes en los triángulos t_1, t_2 y t_3, los números deberían coincidir pues t_0 por hipótesis no posee aristas del tipo m_+. Sin embargo es posible que no coincidan ya que algunas aristas m_+ pueden pertenecer a mas de una relación triangular.

12. (d)

En primer lugar se debe calcular la probabilidad de que una arista sea del tipo m_+:

$$p = \frac{m_+}{m} = 0,8325 \tag{15}$$

Luego, la probabilidad de que el triángulo sea:

- Tipo 0: $p(0) = 1 - p = 0,1675$
- Tipo 1: $p(1) = 0,8325 \cdot 0,1675 \cdot 0,1675 = 0,0233$
- Tipo 2: $p(2) = 0,8325 \cdot 0,8325 \cdot 0,1675 = 0,1161$
- Tipo 3: $P(3) = 0,8325 \cdot 0,8325 \cdot 0,8325 = 0,577$

13. (e)

Al observar las probabilidades se puede apreciar que si tomamos una arista al azar lo más probable es que encontremos una del tipo 3, luego del tipo 0, tipo 2 y por último del tipo 1. Ahora, si se observa las cantidades de aristas de cada tipo, se puede afirmar que las probabilidades respaldan estos datos, pues mientras más aristas hayan de un tipo más probable es "escoger" una de ellas. Por lo tanto, si tomamos una arista al azar lo más probable es que termine siendo una en que sea una relación en que una de las personas confía en las otras dos.

14. Referencias

Referencias

[1] Michael Schnegg Douglas R. White and Lilyan A. Brudner, 1999.

[2] Diego Garlaschelli and Maria I. Loffredo. Patterns of link reciprocity in directed networks. *Phys. Rev. Lett.*, 93:268701, Dec 2004.

[3] Tamás Nepusz, 2014.

[4] Cheng-Jun Wang, 2013.

15. Apéndice

Listing 1: p1.py

```
1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 import igraph as G
5 import random
```

```
6
7  g = G.GraphBase.Erdos_Renyi(400, 1)
8
9  deleted = 0
10 while G.mean(g.degree()) > 1:
11     deleted += 1
12     g.delete_edges(random.randint(0, g.ecount() - 1))
13
14 print "Removed", deleted, "edges"
```

Listing 2: p2.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import igraph as G
5  import numpy as np
6
7  # Build the graph
8  p2 = G.Graph([(0,1),(0,2),(1,2),(2,3),(3,4),(3,5),(4,5)])
9
10 laplacian_matrix = p2.laplacian()
11 w, v = np.linalg.eig(p2.laplacian()) # Eigen values (w) and eigen vectors (v)
12 fiedler_value = sorted(w)[1] # Fiedler value is the second smallest eigenvalues
13 fiedler_vector = v[:, np.where(w == fiedler_value)[0]].T[0] # grab the n-th column of the eigenvectors,
       where n is the index of the Fiedler Value
14
15 print "Matriz␣Laplaciana:"
16 print np.array(laplacian_matrix)
17
18 print "Valores␣Propios", [round(x, 3) for x in sorted(w)]
19 print "Valor␣de␣Fiedler:", round(fiedler_value,4)
20 print "Vector␣de␣Fiedler:", [round(i, 3) for i in fiedler_vector]
21
22 # Plot the graph
23 p2.vs["label"] = [str(j) + "\n" + str(round(i, 3)) for j, i in enumerate(fiedler_vector)]
24 p2.vs["color"] = ["lightblue" if x <= 0 else "green" for x in fiedler_vector]
25 G.plot(p2, layout=p2.layout("kk"), vertex_size=50, bbox=(600,600), margin=40)
```

Listing 3: p3.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import igraph as G
5
6  p1 = G.Graph([(0,1),(0,2),(1,3),(2,3),(2,4),(4,5),(4,6),(5,7),(6,7),(7,8),(7,9),(8,9)])
7  colors = ["lightblue", "green"]
8  p1.vs["color"] = [colors[i] for i in [0, 0, 0, 0, 1, 1, 1, 1, 1, 1]]
9  p1.vs["label"] = range(1,11)
10 G.plot(p1, layout=p1.layout("kk"), bbox=(600,300))
```

Listing 4: p4.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import igraph as G
5  import numpy as np
6
7  # Build the graph
8  p2 = G.Graph.Read_GML("../archivos/redchica.gml")
9  G.summary(p2)
10 # Plot the graph
11
```

```
12 betweenness = p2.betweenness()
13 pageranks = [round(i, 3) for i in p2.pagerank()]
14 indegree = p2.degree(mode="in")
15 names = p2.vs["label"]
16 p2.vs["label"] = ["B:␣" + str(betweenness[i]) + "\nPR:␣" + str(pageranks[i]) + "\nInD:␣" + str(indegree[
       i]) + "\n" + names[i] for i in range(8)]
17 p2.es["width"] = 1
18 p2.vs["color"] = "lightblue"
19 p2.vs["size"] = 80
20 print sorted([(i, j) for i, j in enumerate(indegree)], key=lambda x: x[1], reverse=True)
21 print sorted([(i, j) for i, j in enumerate(betweenness)], key=lambda x: x[1], reverse=True)
22 print sorted([(i, j) for i, j in enumerate(pageranks)], key=lambda x: x[1], reverse=True)
23 G.plot(p2, "../img/p4-all.png", margin=50)
24
25 # plot indegree
26 p2.vs["label"] = ["InD:␣" + str(indegree[i]) + "\n" + names[i] for i in range(8)]
27 p2.vs['size'] = [500.0*i/sum(indegree) for i in indegree]
28 p2.vs["color"] = "green"
29 G.plot(p2, "../img/p4-indegree.png", margin=50)
30
31 # plot betweenness
32 p2.vs["label"] = ["B:␣" + str(betweenness[i]) + "\n" + names[i] for i in range(8)]
33 p2.vs['size'] = [500.0*i/sum(betweenness) for i in betweenness]
34 p2.vs["color"] = "yellow"
35 # G.plot(p2, "../img/p4-betweenness.png", margin=50)
36
37 # plot PageRank
38 p2.vs["label"] = ["PR:␣" + str(pageranks[i]) + "\n" + names[i] for i in range(8)]
39 p2.vs['size'] = [500.0*i/sum(pageranks) for i in pageranks]
40 p2.vs["color"] = "#C0C93F"
41 # G.plot(p2, "../img/p4-pageranks.png", margin=50)
```

Listing 5: p5-1-gnutella.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import igraph as G
5  import random
6
7  def pickRandom(g):
8      return random.randint(0, g.vcount() - 1)
9
10 def pickDegree(g):
11     b = g.degree()
12     return b.index(max(b))
13
14 def pickBetweenness(g):
15     b = g.betweenness()
16     return b.index(max(b))
17
18 g_safe = G.Graph.Read_GML("../archivos/gnutella.gml")
19 total = g_safe.vcount()
20 half = g_safe.components().giant().vcount()/2.0
21
22 for method in [pickRandom, pickDegree, pickBetweenness]:
23     g = g_safe.copy()
24     deleted = 0
25     while g.components().giant().vcount() > half:
26         deleted += 1
27         g.delete_vertices(method(g))
28
29     print method.__name__ + ":␣", round(deleted*100.0/total, 3)
```

Listing 6: p5-2-delfines.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import igraph as G
import random

def pickRandom(g):
    return random.randint(0, g.vcount() - 1)

def pickDegree(g):
    b = g.degree()
    return b.index(max(b))

def pickBetweenness(g):
    b = g.betweenness()
    return b.index(max(b))

g_safe = G.Graph.Read_GML("../archivos/delfines.gml")
total = g_safe.vcount()
half = g_safe.components().giant().vcount()/2.0

for method in [pickRandom, pickDegree, pickBetweenness]:
    g = g_safe.copy()
    deleted = 0
    while g.components().giant().vcount() > half:
        deleted += 1
        g.delete_vertices(method(g))

    print method.__name__ + ":␣", round(deleted*100.0/total, 3)
```

Listing 7: p5-3-erdos-renyi-gnutella.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import igraph as G
import random

def pickRandom(g):
    return random.randint(0, g.vcount() - 1)

def pickDegree(g):
    b = g.degree()
    return b.index(max(b))

def pickBetweenness(g):
    b = g.betweenness()
    return b.index(max(b))

g_safe = G.Graph.Erdos_Renyi(n=795, m=852)
total = g_safe.vcount()
half = g_safe.components().giant().vcount()/2.0

for method in [pickRandom, pickDegree, pickBetweenness]:
    g = g_safe.copy()
    deleted = 0
    while g.components().giant().vcount() > half:
        deleted += 1
        g.delete_vertices(method(g))

    print method.__name__ + ":␣", round(deleted*100.0/total, 3)
```

Listing 8: p5-3-erdos-renyi-delfines.py

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
3
4  import igraph as G
5  import random
6
7  def pickRandom(g):
8      return random.randint(0, g.vcount() - 1)
9
10 def pickDegree(g):
11     b = g.degree()
12     return b.index(max(b))
13
14 def pickBetweenness(g):
15     b = g.betweenness()
16     return b.index(max(b))
17
18 g_safe = G.Graph.Erdos_Renyi(n=62, m=159)
19 total = g_safe.vcount()
20 half = g_safe.components().giant().vcount()/2.0
21
22 for method in [pickRandom, pickDegree, pickBetweenness]:
23     g = g_safe.copy()
24     deleted = 0
25     while g.components().giant().vcount() > half:
26         deleted += 1
27         g.delete_vertices(method(g))
28
29     print method.__name__ + ":␣", round(deleted*100.0/total, 3)
```

Listing 9: p7.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import igraph as G
5
6  def p7(g):
7      # bien pythonesco, tomar los k_cores generados por igraph, filtrar los positivos y armar
8      # la lista de tuplas con (n, m)
9      k_cores = [(x.vcount(), x.ecount()) for x in filter(lambda x: x.vcount() > 0, g.k_core())]
10
11     for i, (n,m) in enumerate(k_cores):
12         print str(i+1) + "-core\t" + "n:␣" + str(n) + "\tm:␣", m
13
14     print "Modularidad", g.community_fastgreedy().as_clustering().modularity
15     print "Asortatividad", g.assortativity_degree()
16
17 g = G.Graph.Read_GML("../archivos/correos.gml")
18 p7(g)
19
20 # Reconectar la red
21 print "\nReconectando␣la␣red␣al␣azar,␣por␣favor␣espere␣un␣momento..."
22 g.rewire(g.ecount()*2)
23 p7(g)
```

Listing 10: p8.py

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import igraph as G
5  import numpy as np
6  import random as R
7  import matplotlib.pyplot as plt
8
9  def probability(choice):
10     p=R.uniform(0,1)
```

```
    pp=1-p
    if(choice==1):
        #return(p,1-p)
        return(p,pp)
    if(choice==2):
        #return(1-p,p)
        return(pp,p)

#Repeticion del experimento
for j in range(10):
    #Grafo Erdos_Renyi
    g=G.Graph.Erdos_Renyi(80,0.2,directed=False)
    #Ahora dirigido
    g.to_directed(False)


    B_1=range(0,40)
    B_2=range(40,80)
    C_1=range(0,20) + range(40,60)
    C_2= range(20,40) + range(60,80)

    #Redirigir particiones
    edges=g.get_edgelist()
    for edge in edges:
        #(B_1,B_2)
        x=edge[0]
        y=edge[1]
        if( (x in B_1) and (y in B_2)):
            weights=probability(1)
            choice=np.random.choice(edge,1,p=weights)
            choice=int(choice[0])
            if(y==choice):
                g.delete_edges([(x,y)])
                g.add_edges([(y,x)])

        #(B_2,B_1)
        if( (y in B_1) and (x in B_2)):
            weights=probability(2)
            choice=np.random.choice(edge,1,p=weights)
            choice=int(choice[0])
            if(y==choice):
                g.delete_edges([(x,y)])
                g.add_edges([(y,x)])

    #Para p =(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1)
    p =(0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1)

    Mx_1=[[0,0,0,0,0,0,0,0,0,0,0]]
    Mx_2=[[0,0,0,0,0,0,0,0,0,0,0]]


    Curve_1=[]
    Curve_2=[]

    #Curve_1
    for i in p:
        edges=g.get_edgelist()
        for edge in edges:
            x=edge[0]
            y=edge[1]
            #(B_1,B_2)
            if((x in B_1) and (y in B_2)):
                weights=(i,1-i)
                choice=np.random.choice(edge,1,p=weights)
                choice=int(choice[0])
                if(y==choice):
                    g.delete_edges([(x,y)])
                    g.add_edges([(y,x)])
```

```
            #(B_2,B_1)
            if((y in B_1) and (x in B_2)):
                weights=(1-i,i)
                choice=np.random.choice(edge,1,p=weights)
                choice=int(choice[0])
                if(y==choice):
                    g.delete_edges([(x,y)])
                    g.add_edges([(y,x)])
        vertexDendrogram=g.community_edge_betweenness()
        Curve_1 += [float(vertexDendrogram.as_clustering().modularity)]
    Mx_1=np.r_[Mx_1,[Curve_1]]

    #Curve_2
    for i in p:
        edges=g.get_edgelist()
        for edge in edges:
            x=edge[0]
            y=edge[1]
            #(C_1,C_2)
            if((x in C_1) and (y in C_2)):
                weights=(i,1-i)
                choice=np.random.choice(edge,1,p=weights)
                choice=int(choice[0])
                if(y==choice):
                    g.delete_edges([(x,y)])
                    g.add_edges([(y,x)])
            #(C_2,C_1)
            if((y in C_1) and (x in C_2)):
                weights=(1-i,i)
                choice=np.random.choice(edge,1,p=weights)
                choice=int(choice[0])
                if(y==choice):
                    g.delete_edges([(x,y)])
                    g.add_edges([(y,x)])
        vertexDendrogram=g.community_edge_betweenness()
        Curve_2 += [float(vertexDendrogram.as_clustering().modularity)]
    Mx_2=np.r_[Mx_2,[Curve_2]]



Mx_1=Mx_1[1:]
Mx_2=Mx_2[1:]
#Promedio
Mx_1_avg=Mx_1.sum(axis=0)/11.
Mx_2_avg=Mx_2.sum(axis=0)/11.


plt.plot(Mx_1_avg)
plt.plot(Mx_2_avg)
plt.legend(['(B_1,B_2)', '(C_1,C_2)'], loc='upper left')
plt.savefig('../img/p8-3.png')
```

Listing 11: p9.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import igraph as G
import numpy as np
import matplotlib.pyplot as plt

# Build the graph
g = G.read("../archivos/pescado.net",format="pajek")

m = g.ecount() # Arcos
n = g.vcount() # Nodos
a = g.get_adjacency()._get_data()
```

```
14
15 rho_ = 0
16 a_ = 0
17 for i in range(len(a)):
18     for j in range(len(a)):
19         if i == j: continue
20         rho_ += a[i][j]*a[j][i]
21         a_ += a[i][j]
22
23 a_ = a_ / float(n*(n-1))
24 rho_ = rho_ / float(m)
25
26 print "Rho␣=", round( ( rho_ - a_ )/ ( 1 - a_ ) , 4)
27
28 # B
29 s = np.matrix(g.get_adjacency(attribute="weight")._get_data())
30 s_in = np.squeeze(np.asarray(s.sum(axis=0, dtype=float)))
31 s_out = np.squeeze(np.asarray(s.sum(axis=1, dtype=float).T))
32
33 plt.hist(s_in, bins=4)
34 plt.savefig('../img/p9-s-in.png')
35 plt.close()
36
37 plt.hist(s_out, bins=9)
38 plt.savefig('../img/p9-s-out.png')
39 plt.close()
40
41 plt.plot(s_in, s_out, '.')
42 plt.savefig('../img/p9-s-s.png')
43 plt.close()
44
45 # D
46 g_undirected = g.as_undirected(mode="collapse", combine_edges=dict(weight=sum))
47
48 plt.hist(g_undirected.degree())
49 plt.savefig('../img/p9-k.png')
50 plt.close()
51
52 s_data = np.matrix(g_undirected.get_adjacency(attribute="weight")._get_data())
53 s = np.squeeze(np.asarray(s_data.sum(axis=0, dtype=float)))
54 plt.hist(s, bins=3)
55 plt.savefig('../img/p9-s.png')
56 plt.close()
57
58 # E
59 plt.plot(s, g_undirected.degree(), '.')
60 plt.savefig('../img/p9-k-vs-s.png')
61 plt.close()
62
63 # F
64 sin_peso = G.mean(filter(lambda x: x == x, g_undirected.transitivity_local_undirected()))
65 con_peso = G.mean(filter(lambda x: x == x, g_undirected.transitivity_local_undirected(weights="weight"))
       )
66
67 print "sin␣peso:", sin_peso
68 print "con␣peso:", con_peso
```