Handout 4

1) Generate an undirected random graph using the model of "preferential attachment" (sample_pa( )) of 1000 nodes. With $\beta = 0.1$, $\gamma = 0.1$, generate a SIR pandemic (iterative method). The initial infected nodes should be the 10 highest using the eigen_centrality( ). Compare the results to when the initial nodes are at random. Reduce or increase β and compare.

Se define el algoritmo del modelo SIR como se muestra a continuación:

```
SIR_Model = function(x, beta, gamma){

  S = I - x - R
  R = R + gamma * x

  return (x + beta * S * (A %*% x) - gamma * x)
}
```

Se ejecuta el algoritmo SIR con un grafo aleatorio de 1000 vértices y los nodos infectados como los más 10 nodos más centrales según "*eigen_centrality()*". El algoritmo se ejecuta 25 iteraciones en total:

```
#Generate random graph and obtain initial infected nodes
n = 1000
set.seed(1234)
g = sample_pa(n, power=1, directed=FALSE)
eigen_cent = order(eigen_centrality(g)$vector, decreasing = TRUE)
infected_n = head(eigen_cent, 10)

#Iteration 1 vector
x0 = rep(0, n)
x0[infected_n] = 1

#Initial values
beta = 0.1
gamma = 0.1
A = as.matrix(as_adjacency_matrix(g))
I = rep(1, n)
R = rep(0, n)
iter = 25
RX = matrix(0, nrow = n, ncol = iter)
RX[,1] = x0

#SIR model, 25 iterations
for (i in 2:(iter)){
  RX[,i] = SIR_Model(RX[,i-1], beta, gamma)
}
```

A continuación se realiza el experimento donde los 10 nodos infectados iniciales son escogidos aleatoriamente:

```
set.seed(1234)
infected_n2 = sample(V(g), 10)
#Iteration 1 vector
x0_Random = rep(0, n)
x0_Random[infected_n2] = 1

RX_Random = matrix(0, nrow = n, ncol = iter)
RX_Random[,1] = x0_Random

#SIR model, 25 iterations
for (i in 2:(iter)){
  RX_Random[,i] = SIR_Model(RX_Random[,i-1], beta, gamma)
}
```

Comparamos los resultados de ambas ejecuciones:

Table 1: Source infection: highly central nodes

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	0.1	0.171	0.2371316	0.3087218	0.3887139	0.4753357	0.5637220	0.6474010	0.7201057
0	0.0	0.010	0.0259290	0.0476033	0.0762345	0.1132938	0.1598836	0.2163207	0.2818660
1	0.9	0.834	0.8084842	0.8102809	0.8245098	0.8414182	0.8566032	0.8688352	0.8783151
0	0.0	0.000	0.0010000	0.0034903	0.0080644	0.0156037	0.0273278	0.0447888	0.0698186
1	0.9	0.922	0.9291174	0.9366897	0.9411850	0.9453942	0.9480598	0.9506727	0.9522210

Table 2: Source infection: random nodes

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0.9	0.812	0.7372296	0.6752849	0.6250252	0.5850530	0.5539522	0.5303923	0.5131632
0	0.0	0.000	0.0000000	0.0000000	0.0000000	0.0000020	0.0000124	0.0000449	0.0001234
0	0.0	0.000	0.0000000	0.0001000	0.0004390	0.0011807	0.0025189	0.0046913	0.0079976
0	0.0	0.000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000002
0	0.0	0.000	0.0020000	0.0069413	0.0152677	0.0271806	0.0427347	0.0618519	0.0843016
0	0.0	0.000	0.0000000	0.0001000	0.0004390	0.0012139	0.0027187	0.0053920	0.0098658

Centrándonos en los nodos inicialmente infectados, se puede observar en la primera tabla, que si estos son nodos altamente conexos o centrales, su nivel de recuperación no es muy bueno ya que tienen contacto con un gran número de los otros nodos del grafo. Es decir, al estar influenciados por más vecinos, su recuperación es más lenta con el paso de las iteraciones. En cambio, si nos fijamos en los nodos inicialmente infectados de la segunda tabla, los cuales son cogidos aleatoriamente, vemos que su recuperación es continua con el paso de las iteraciones y su nivel de infección va disminuyendo en cada una de ellas. Esto se debe a que, al ser nodos escogidos al azar, no tiene por qué ser nodos centrales o con una gran cantidad de vecinos, por lo que no se ven tan influenciados por estos y es más fácil que se recupere.

Por otro lado, si nos fijamos en los nodos que no estaban contagiados inicialmente (V1), podemos observar que en la segunda iteración de la tabla 1 hay un mayor número de infectados que en la tabla 2. Para medir los nuevos infectados, se ha sumado todas las infecciones de la segunda iteración en ambos casos y se le restan 9 puntos ya que es la parte aportada por los nodos infectados inicialmente (10x0,9), obteniendo los siguientes resultados:

- Infección con nodos centrales: 9.2
- Infección con nodos aleatorios: 1.4

Se puede apreciar claramente como en el primer caso la infección se ha propagado a un mayor número de nodos en el grafo. Esto es debido, como en el caso anterior, a la centralidad y conectividad de los nodos infectados inicialmente. Ya que los más centrales tienen un mayor alcance hacia otros nodos, la infección se propaga rápidamente.

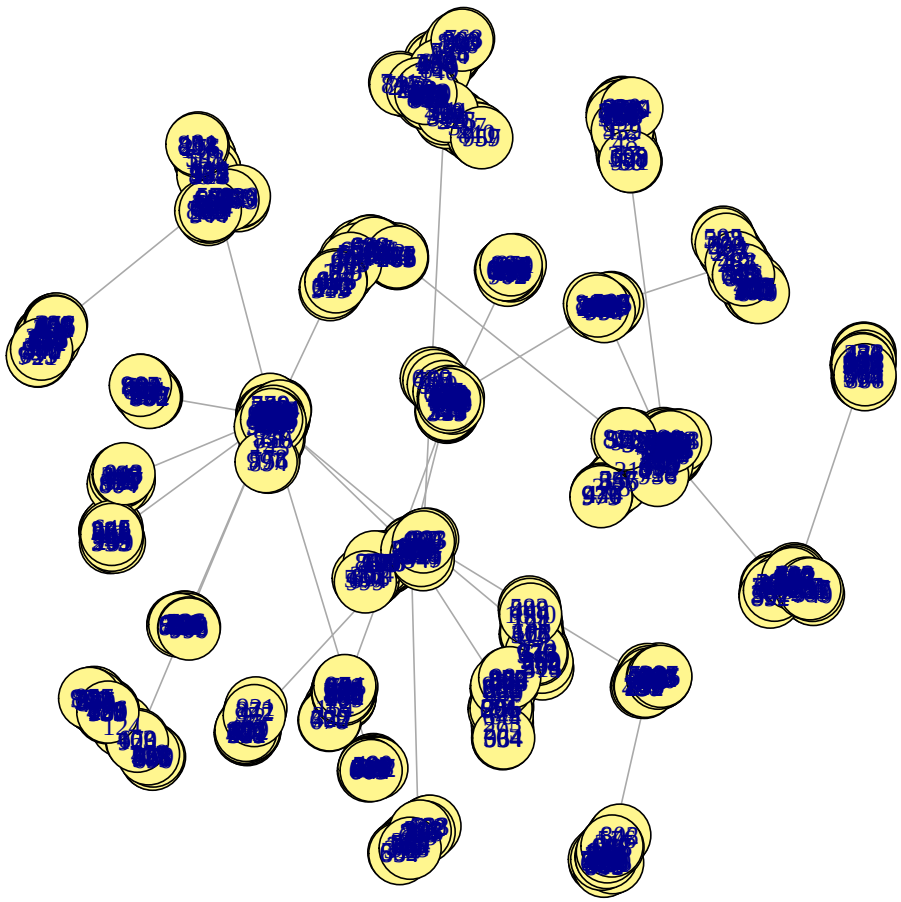
Table 3: Source infection: highly central nodes

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	0.5	0.675	1.498000	-0.6044331	12.0482356	-5.132608e+02	2.692770e+07	1.966810e+18	2.746664e+41
0	0.0	0.250	0.478125	1.2126031	0.7452544	3.376368e+00	6.015333e+02	-8.085495e+09	7.951317e+27
1	0.9	1.230	-0.286225	0.3665221	81.9699805	1.046870e+05	-1.460808e+11	-2.793014e+23	-1.021018e+48
0	0.0	0.000	0.125000	0.3216797	1.0823336	7.900054e-01	2.274234e+00	-3.899100e+02	-1.580350e+12
1	0.9	1.990	-13.130775	247.4103101	-2608.3347564	2.791273e+06	-3.823953e+12	-7.311227e+24	-2.672703e+49

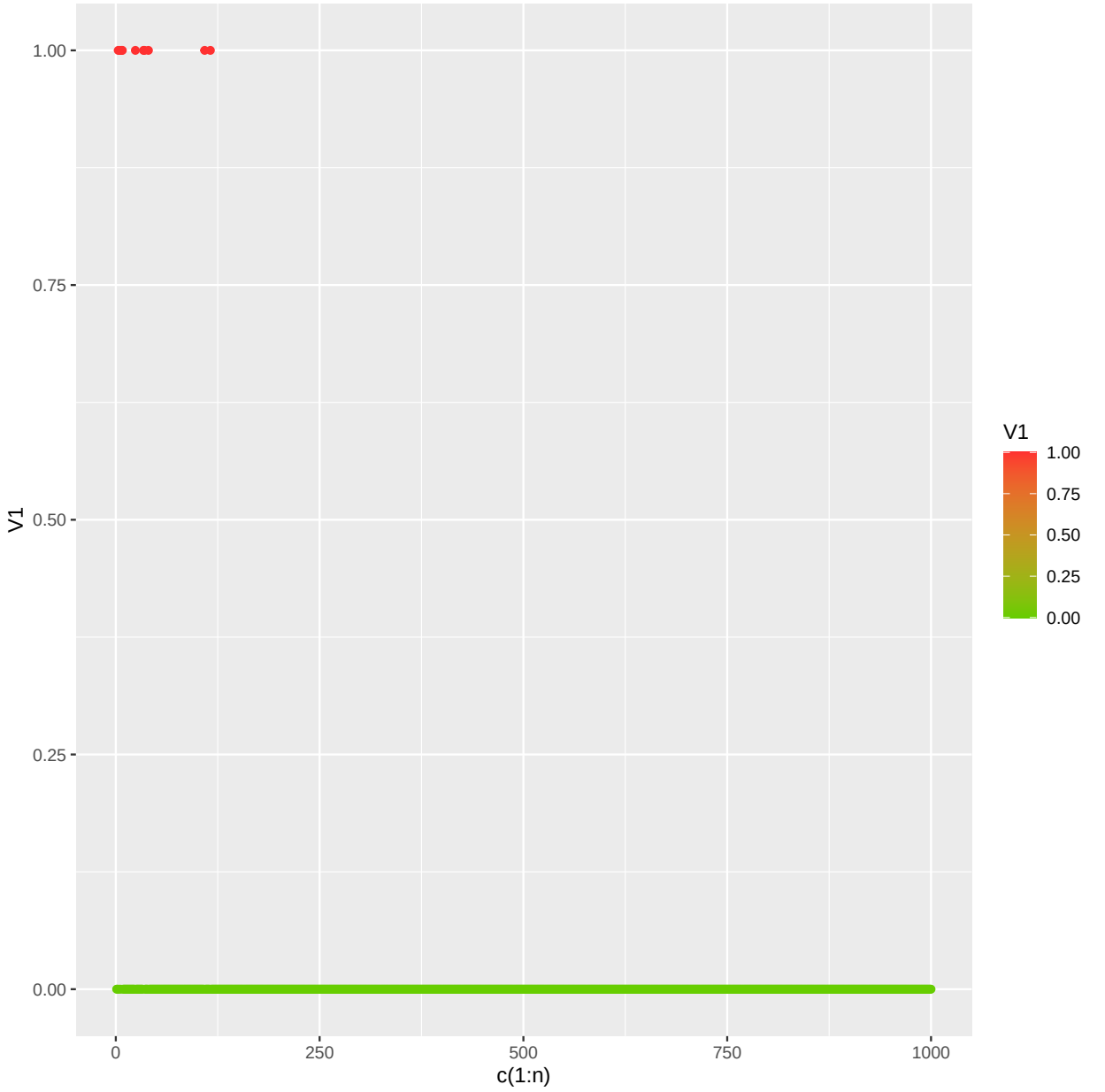
Table 4: Source infection: random nodes

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0.9	0.86	0.8685	0.8986110	0.8922314	0.9086322	0.8823147	0.9572027	0.8766277
0	0.0	0.00	0.0000	0.0000000	0.0000000	0.0312500	0.1372557	0.4044325	0.7259427
0	0.0	0.00	0.0000	0.0625000	0.2070374	0.5920147	0.9051645	0.9560766	0.9887104
0	0.0	0.00	0.0000	0.0000000	0.0000000	0.0000000	0.0000000	0.0078125	0.0520218
0	0.0	0.00	0.2500	0.5203125	0.7364803	0.7537268	0.9117334	0.7685677	2.1604761
0	0.0	0.00	0.0000	0.0625000	0.2070374	0.7296005	0.9565319	0.7226149	-1345.2022881

```
#Graph chart
V(g)$prob = as.matrix(RX)
V(g)[V(g)$prob < 0.5]$color = "chartreuse3"
V(g)[V(g)$prob < 0.5]$color = "khaki1"
V(g)[V(g)$prob > 0.5]$color = "firebrick1"
plot(g)
```



```
#Point chart
df = as.data.frame(as.matrix(RX))
ggplot(df, aes(x = c(1:n), y = V1)) +
  geom_point(aes(color = V1)) +
  scale_colour_gradient(low="chartreuse3", high = "firebrick1")
```



2) Consider the random graph generated in the previous exercice.

a) Plot its degrees distribution in linear and in log-log scale. Which is more helpful to understand this distribution?

b) Does the degree distribution follows a Power Law? And if we consider only the nodes with degree above 5? (or 10? or 100?)

c) Find the best line that approximates the degree distribution after degree 10 (or 5?) using linear regression (lm()) on the log-log plane. Don't worry, it is almost all done in the following code. Explain in detail each line of the following code:

```
D=degree_distribution(GRAPH)
xx=which(D>0)[-(1:10)]  # remove the first 10 prob values
lyy=log(D[xx])
```

```
lxx=log(xx)
LMI=lm(lyy~lxx)$coefficients  # line coefficients
plot(D,pch=20,cex=0.7,xlab="Degree",ylab="Frequencies",main="degrees",log="xy")
points(exp(lxx),exp(LMI[1]+LMI[2]*lxx),col="red",type="l",lwd=2)
```

d) What is the exponent of the Power Law for the degree probabilities?

3) Use the routine sample_pa( ) to generate a rich-get-richer (preferential attachment) graph with similar degree distribution of the *directed* facebook graph of the file **facebook_sample_anon.txt**. Use the code similar to: sample_pa(n.GRAPH, out.seq=degree(GRAPH,mode="out")) Plot the degree distribution of the generated graph (log-log). What is the exponent of the power law of the generated graph for the in-degrees?