Handout 4

Laura Basalo Tur, Camila Pérez Arévalo, Josep Roman Cardell

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, r, s, beta, gamma){

  R = r + gamma * x
  X = x + beta * s * (A %*% x) - gamma * x
  S = I - X - R

  return(list(S, R, X))
}
```

Se ejecuta el algoritmo SIR con un grafo aleatorio de 1000 vértices y los nodos infectados como los 10 nodos más centrales según "*eigen_centrality()*". El algoritmo se ejecuta 150 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 = 150

RX = matrix(0, nrow = n, ncol = iter)
RX[,1] = x0

RR = matrix(0, nrow = n, ncol = iter)
RR[,1] = R
```

```
RS = matrix(0, nrow = n, ncol = iter)
RS[,1] = I - x0 - R

#SIR model, 150 iterations
for (i in 2:(iter)){
  result = SIR_Model(RX[,i-1], RR[,i-1], RS[,i-1], beta, gamma)
  #Prob. infected
  RX[,i] = result[[3]]
  #Prob. recovered
  RR[,i] = result[[2]]
  #Prob. sus
  RS[,i] = result[[1]]
}
```

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

RR_Random = matrix(0, nrow = n, ncol = iter)
RR_Random[,1] = R

RS_Random = matrix(0, nrow = n, ncol = iter)
RS_Random[,1] = I - x0 - R

#SIR model, 150 iterations
for (i in 2:(iter)){
  result_Random = SIR_Model(RX_Random[,i-1], RR_Random[,i-1], RS_Random[,i-1], beta, gamma)
  #Prob. infected
  RX_Random[,i] = result_Random[[3]]
  #Prob. recovered
  RR_Random[,i] = result_Random[[2]]
  #Prob. sus
  RS_Random[,i] = result_Random[[1]]
}
```

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	0.234162	0.2971629	0.3616584	0.4256001	0.4842934	0.5316024	0.5617739
0	0.0	0.010	0.025929	0.0472894	0.0747400	0.1091027	0.1508122	0.1995232	0.2538595
1	0.9	0.810	0.729000	0.6561000	0.5904900	0.5314410	0.4782969	0.4304672	0.3874205
0	0.0	0.000	0.001000	0.0034903	0.0080326	0.0154234	0.0267333	0.0432912	0.0666156
1	0.9	0.810	0.729000	0.6561000	0.5904900	0.5314410	0.4782969	0.4304672	0.3874205

Table 2: Source infection: random nodes

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0.9	0.81	0.729	0.6561000	0.5904900	0.5314410	0.4782969	0.4304672	0.3874205
0	0.0	0.00	0.000	0.0000000	0.0000000	0.0000020	0.0000124	0.0000449	0.0001232
0	0.0	0.00	0.000	0.0001000	0.0004390	0.0011795	0.0025115	0.0046643	0.0079219
0	0.0	0.00	0.000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000002
0	0.0	0.00	0.002	0.0069413	0.0152368	0.0270038	0.0421433	0.0603482	0.0810899

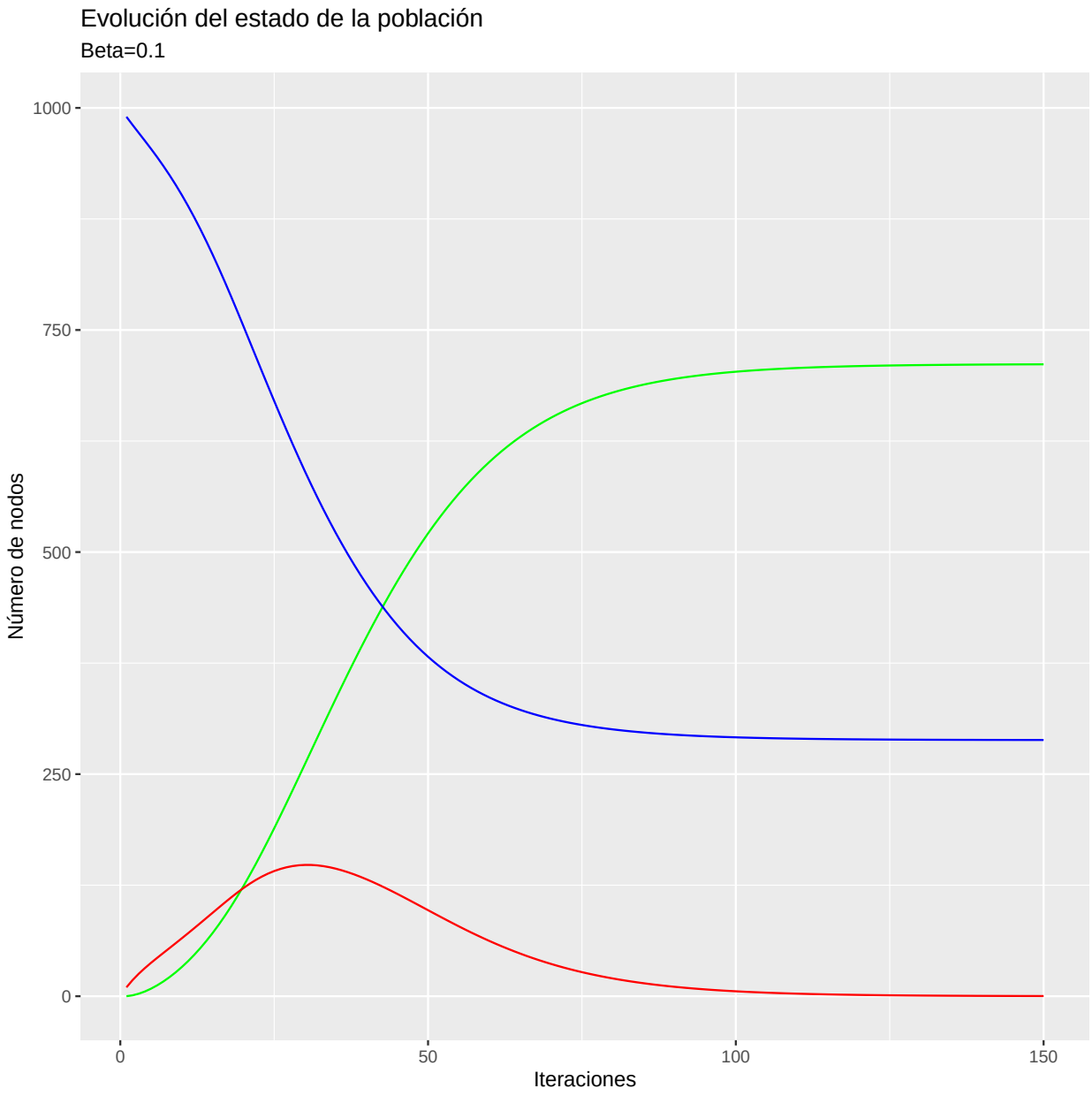
Centrándonos en la recuperación de los nodos, vemos que tienen la misma velocidad de recuperación los nodos de ambos experimentos. Esto se debe a que la recuperación de un nodo depende de su propia infección y es independiente de la de sus vecinos. A partir de los nodos inicialmente infectados podemos ver como si nivel de infección disminuye a la misma velocidad en los dos grafos.

Por otro lado, si nos fijamos en los nodos que no estaban contagiados inicialmente, i.e. $V1 = 0$, 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 de infección tras la primera iteración:

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

Se puede apreciar claramente como 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 connectividad de los nodos infectados inicialmente. Ya que los más centrales tienen un mayor alcance hacia otros nodos, la infección se propaga rápidamente.

```
case1_RR = colSums(RR)
case1_RX = colSums(RX)
case1_RS = colSums(RS)

df = as.data.frame(t(rbind(case1_RR, case1_RX, case1_RS)))

ggplot(df, aes(x = c(1:iter))) +
  geom_line(aes(y = case1_RR), color = "green") +
  geom_line(aes(y = case1_RX), color = "red") +
  geom_line(aes(y = case1_RS), color = "blue") +
  ggtitle("Evolución del estado de la población", subtitle="Beta=0.1") +
  xlab("Iteraciones") + ylab("Número de nodos")
```



A continuación realizamos el mismo experimento cambiando el valor de beta a $\beta = 0.5$

Comparamos los resultados de ambas ejecuciones:

Table 3: Source infection: highly central nodes

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	0.25	0.39375	0.6264727	0.8153284	0.8104367	0.6933373	0.6636000	0.5400733	0.5760713
0	0.00	0.06250	0.1485352	0.3056771	0.5145603	0.7082398	0.8012582	0.7525254	0.6667008
1	0.90	0.81000	0.7290000	0.6561000	0.5904900	0.5314410	0.4782969	0.4304672	0.3874205
0	0.00	0.00000	0.0156250	0.0506161	0.1346472	0.2939424	0.5344253	0.7802994	0.8441266
1	0.90	0.81000	0.7290000	0.6561000	0.5904900	0.5314410	0.4782969	0.4304672	0.3874205

Se observa que al aumentar el beta, la infección en la segunda iteración de los nodos que se han infectado pasa de 0.1 a 0.25, por lo que la propagación es más rápida. Al hacer uso de la métrica explicada anteriormente

Table 4: Source infection: random nodes

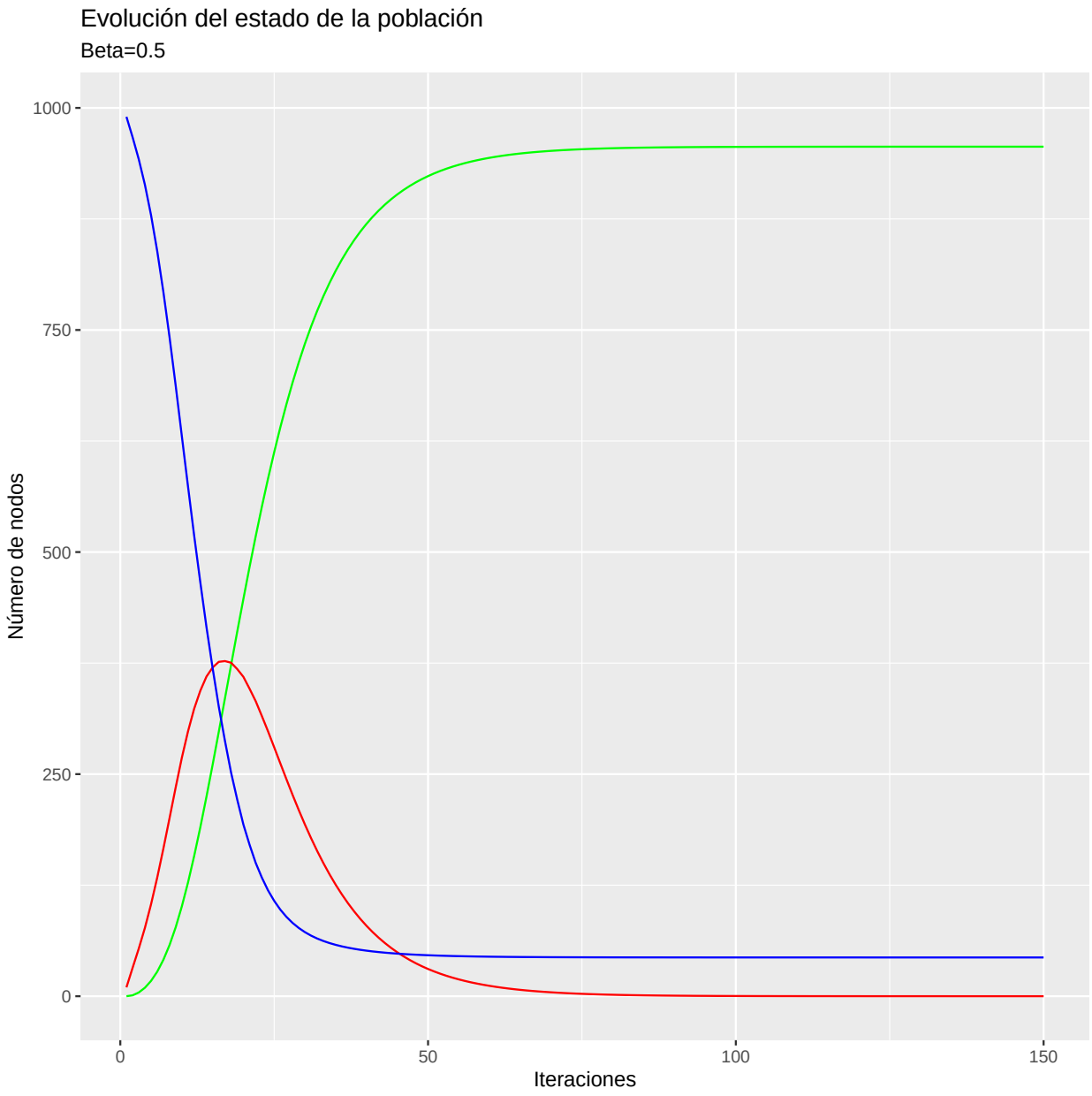
V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
1	0.9	0.81	0.72900	0.6561000	0.5904900	0.5314410	0.4782969	0.4304672	0.3874205
0	0.0	0.00	0.00000	0.0000000	0.0000000	0.0004883	0.0028771	0.0104169	0.0292666
0	0.0	0.00	0.00000	0.0039063	0.0161202	0.0447816	0.1027863	0.2052992	0.3552762
0	0.0	0.00	0.00000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000305	0.0002425
0	0.0	0.00	0.03125	0.0970917	0.1963746	0.3102634	0.4070127	0.4671385	0.4943239

obtenemos:

- Infección con nodos centrales: 23
- Infección con nodos aleatorios: 3.5

Se observa claramente que en la segunda iteración la puntuación de la infección, en relación a los valores obtenidos cuando beta era 0.1, ha incrementado notablemente en cada situación inicial respectivamente.

```
case1_RR = colSums(RR)
case1_RX = colSums(RX)
case1_RS = colSums(RS)

df = as.data.frame(t(rbind(case1_RR, case1_RX, case1_RS)))

ggplot(df, aes(x = c(1:iter))) +
  geom_line(aes(y = case1_RR), color = "green") +
  geom_line(aes(y = case1_RX), color = "red") +
  geom_line(aes(y = case1_RS), color = "blue") +
  ggtitle("Evolución del estado de la población", subtitle="Beta=0.5") +
  xlab("Iteraciones") + ylab("Número de nodos")
```

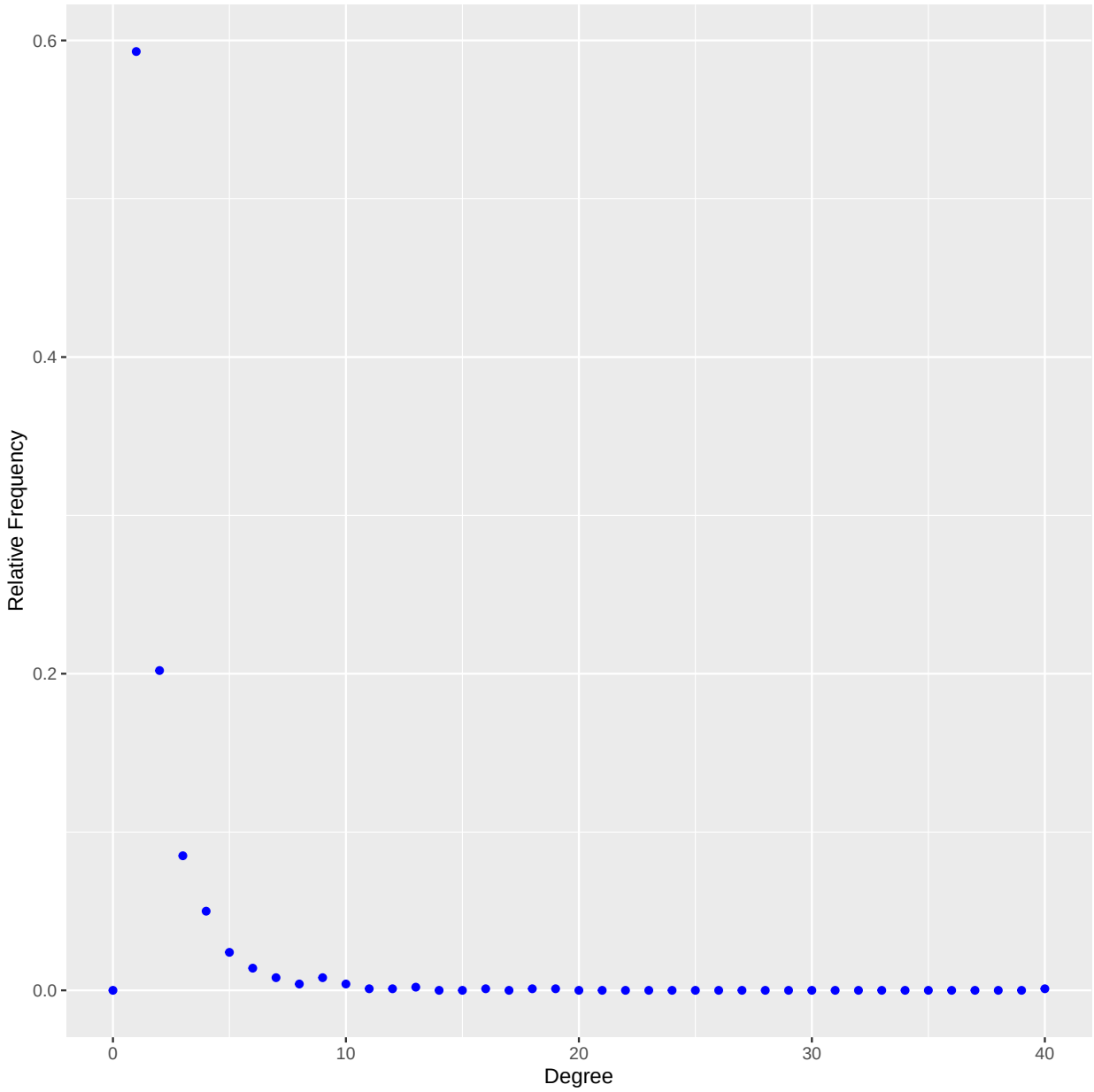


2) Consider the random graph generated in the previous exercice.

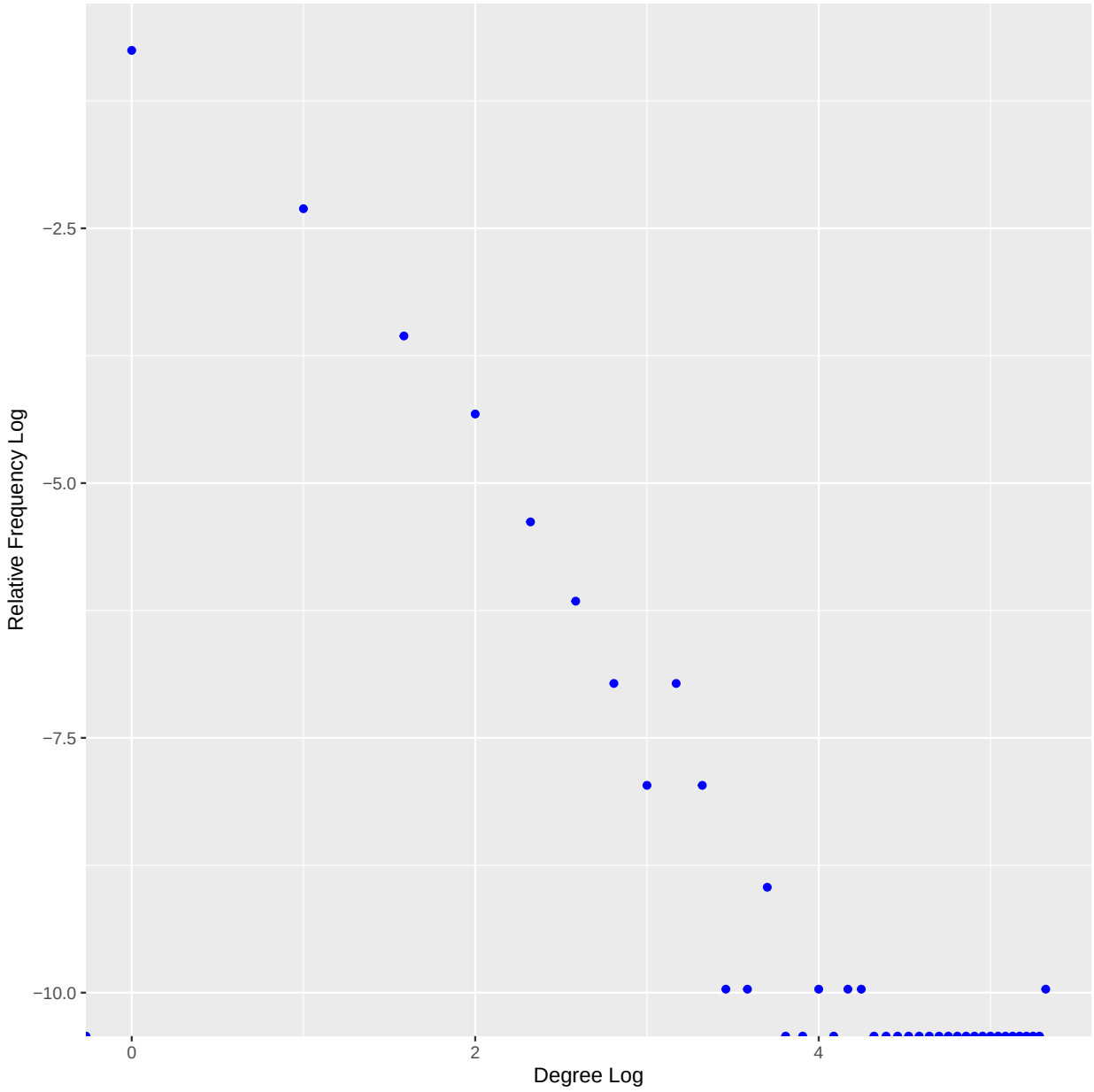
a) Plot its degrees distribution in linear and in log-log scale. Which is more helpful to understand this distribution?

```
#Calculate degree distribution
d_dist = degree_distribution(g)
df_dist = data.frame(0:40, d_dist)
colnames(df_dist) = c("Degree", "Relative Frequency")

#Calculate log
df_dist$`Degree Log` = log(df_dist$Degree, 2)
df_dist$`Relative Frequency Log` = log(df_dist$`Relative Frequency`, 2)
```

```
#Lineal
ggplot(df_dist) +
  geom_point(aes(x = Degree, y =`Relative Frequency`), color = "blue")
```




```
#Log-Log
ggplot(df_dist) +
  geom_point(aes(x = `Degree Log`, y =`Relative Frequency Log`), color = "blue")
```



Al representar ambos gráficos observamos que, en este caso, al tener un grafo de 1000 vértices, la representación lineal es fácil e intuitiva para comprender que la mayoría de nodos tienen grado 1, 2 y 3. Los nodos restantes, una minoría, están dispersos en un rango de grado más amplio, que no sobrepasa de 40 grados.

En el caso de tener que representar un grafo más grande, cuyo rango de grados podría ser mucho más amplio, sería más adecuado utilizar una representación log-log.

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

La distribución de grados de nuestro grafo, como se ha podido observar en los gráficos anteriores, sigue la relación matemática *Power Law*, donde a la derecha de la representación encontramos un amplio rango de grados altos con frecuencias bajas y en la parte izquierda, se encuentra un gran conjunto de nodos de grados menores.

Si solamente se tuviese en cuenta la distribución de nodos con grado mayor a 5, 10 o 100, no se seguiría la

relación *Power Law*, ya que no habría una gran cantidad de nodos con el mismo grado en la parte izquierda de la representación. En nuestro ejemplo, la mayoría se encuentran con grado 1, 2 y 3.

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(g)  #Obtenemos la distribución de grado
xx = which(D>0)[-(1:10)]  #Eliminamos los 10 primeros valores
lyy = log(D[xx]) #Calculamos el logaritmo neperiano de la distribución completa del grafo
lxx = log(xx) #Calculamos el logaritmo neperiano de todo, menos los 10 primeros grados
LMI = lm(lyy~lxx)$coefficients  #Coeficientes

#Representación gráfica
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)
```

La recta que mejor aproxima la distribución es:

log(P_{d}(k)) = 12 + 12k

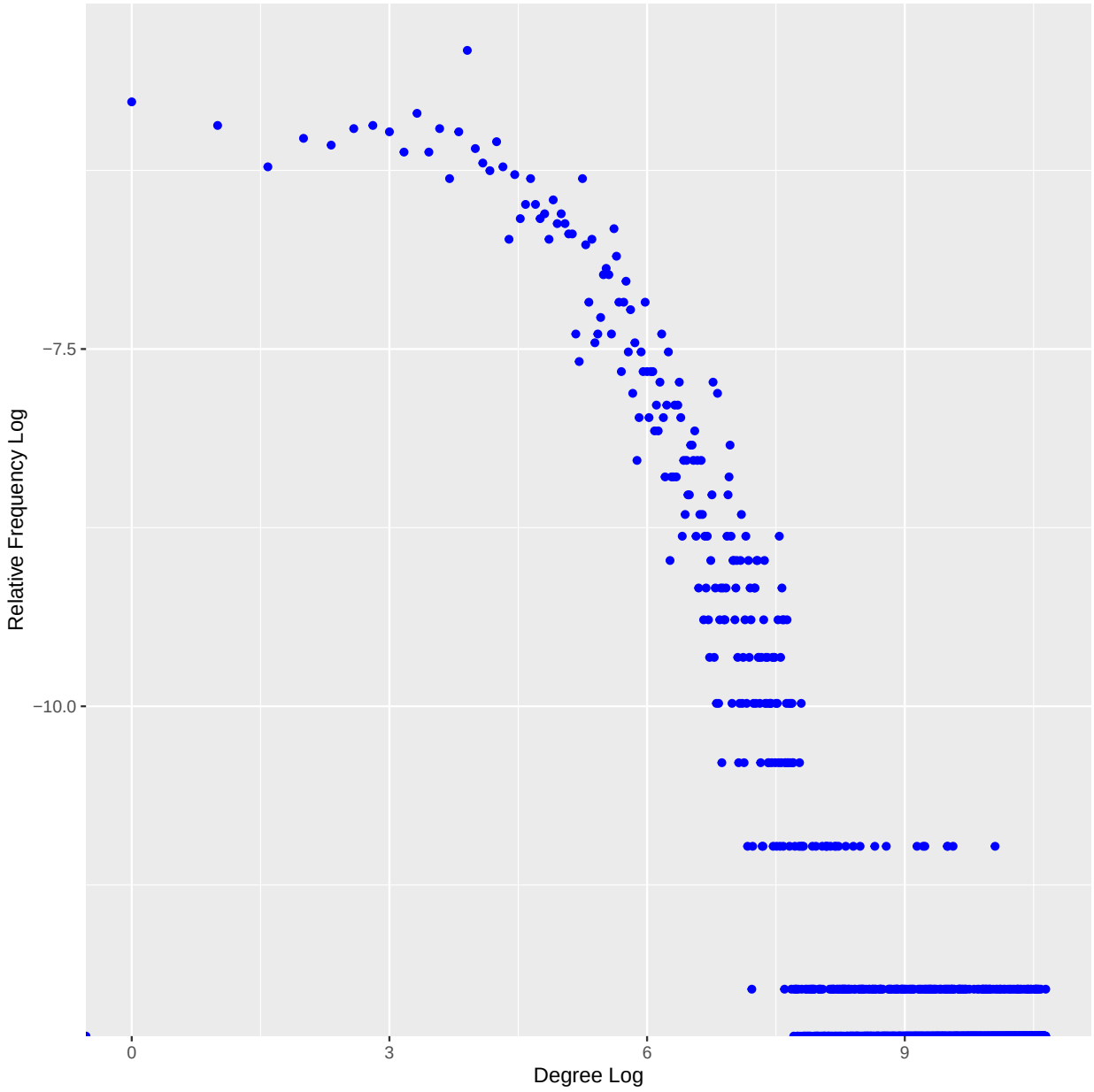
d) What is the exponent of the Power Law for the degree probabilities?

```
pl = fit_power_law(df_dist$`Relative Frequency`)
```

El exponente de *Power Law* de nuestro grafo es 1.4801424.

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?**

```
#Cargamos los datos del fichero .txt en un dataframe
dataframe = read.table("data/facebook_sample_anon.txt",
                       header = FALSE,
                       col.names = c("nodeA", "nodeB"),
                       sep = " ")

#Generamos el grafo no dirigido desde los datos del dataframe
undirected_graph = graph_from_data_frame(dataframe, directed=F)

g2 = sample_pa(length(V(undirected_graph)), out.seq = degree(undirected_graph, mode="out"))

d_dist = degree_distribution(g2)
df_dist = data.frame(0:1597, d_dist)
colnames(df_dist) = c("Degree", "Relative Frequency")

#Calculate log
df_dist$`Degree Log` = log(df_dist$Degree, 2)
df_dist$`Relative Frequency Log` = log(df_dist$`Relative Frequency`, 2)

#Log-Log
ggplot(df_dist) +
  geom_point(aes(x = `Degree Log`, y =`Relative Frequency Log`), color = "blue")
```




```
#Alpha
pl = fit_power_law(df_dist$`Relative Frequency`)
```

El exponente de *Power Law* de nuestro grafo es 1.8099073.