# Compte Rendu TD 1

## Propriétés du document

| | |
|---|---|
| Classification | Interne |
| Version | Version 1.0 |
| Auteurs<br><br>Encadrant | Meryem EL KADIRI meryem.el_kadiri@telecom-sudparis.eu<br>Duc-Vinh TRAN duc-vinh.tran@telecom-sudparis.eu<br><br>Cedric GOUY-PAILLER cedric.gouy-pallier@cea.fr |
| Pages | 17 |

## Historique des modifications

| Date | Version | Objet de la mise à jour | Auteur(s) | Action |
|---|---|---|---|---|
| 13/12/15 | 0.1 | Création du document | Meryem EL KADIRI ; Duc-Vinh TRAN | Création |

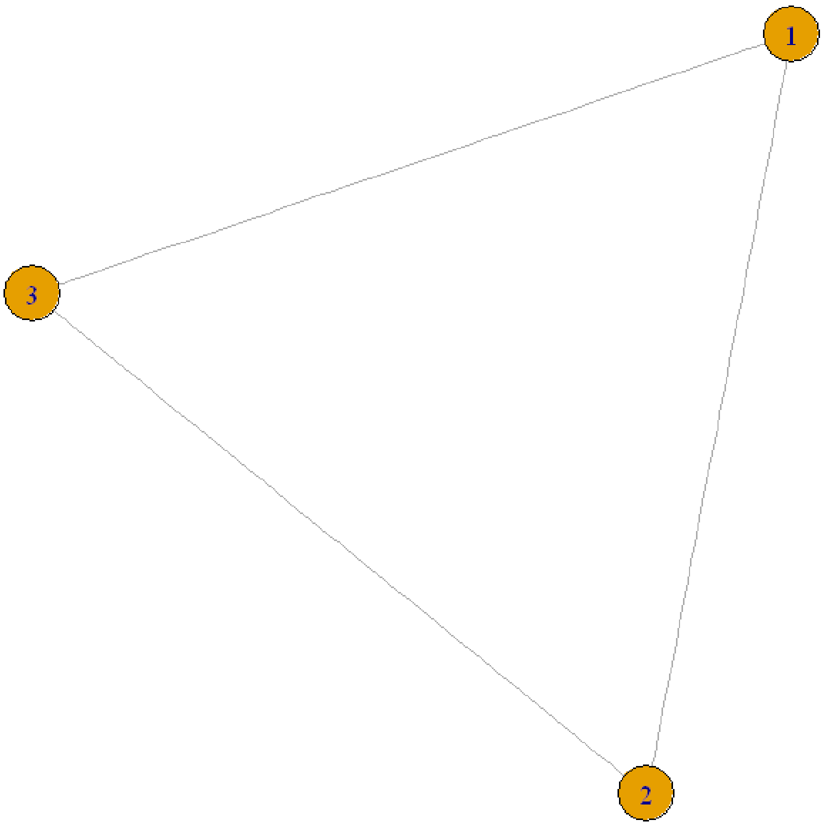
# I- GRAPHES ALEATOIRES

**a- On souhaite générer un graphe non-dirigé de 20000 nœuds, pour cela nous allons suivre la procédure suivante :**

- Nous allons commencer avec 3 nœuds :

```
g <- make_ring(3)
plot(g)
```

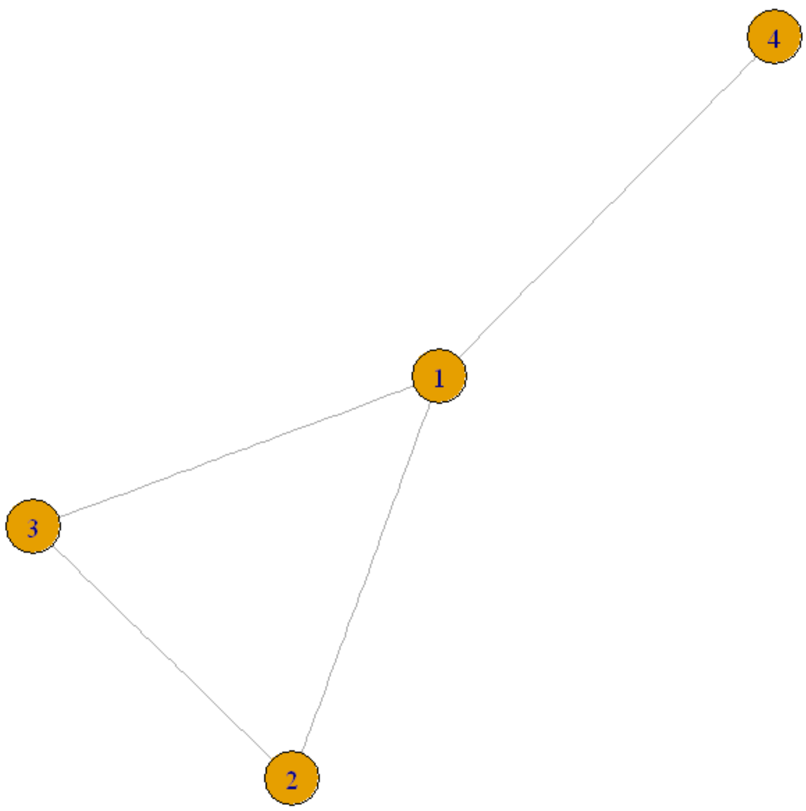
On obtient par la suite le graphe suivant :



- Ensuite, pour chaque nouveau nœud ajouté, nous allons sélectionner au hasard (uniformément), un nœud existant o :

```
n=3
o <- sample(n,1)
g <- add_vertices(g,1)
p<-0.1
if(runif(1)<=p){
    cat("Tirage 1 : succes\n")
    g<-add_edges(g,c(n+1,o))
} else {
    cat("Tirage 1 : echec\n")
    candidates<-neighbors(g,o)
    #le réseau des plus proches voisins peut il être vide?
    selected <- sample(candidates,1)
    g<-add_edges(g,c(n+1,selected))
}

plot(g)
```

Ce code nous permet de générer le graphe suivant :

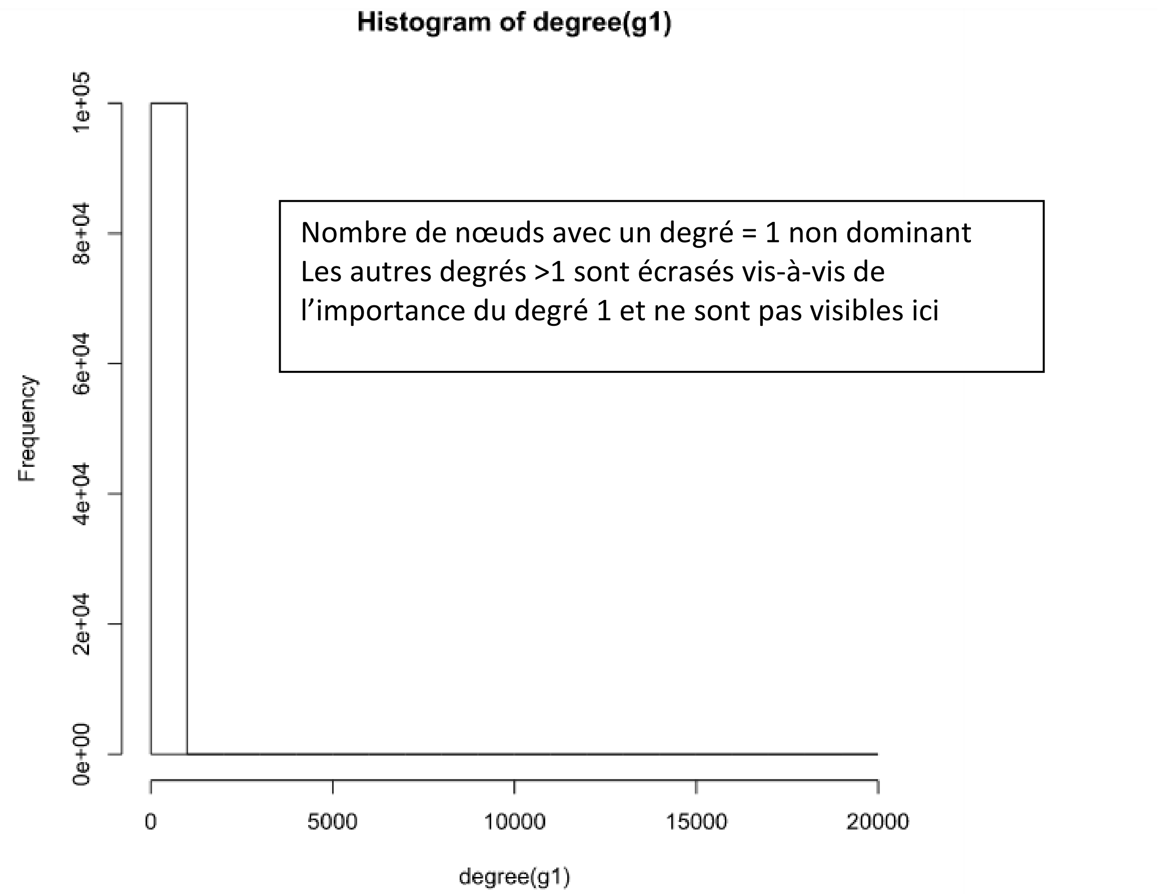


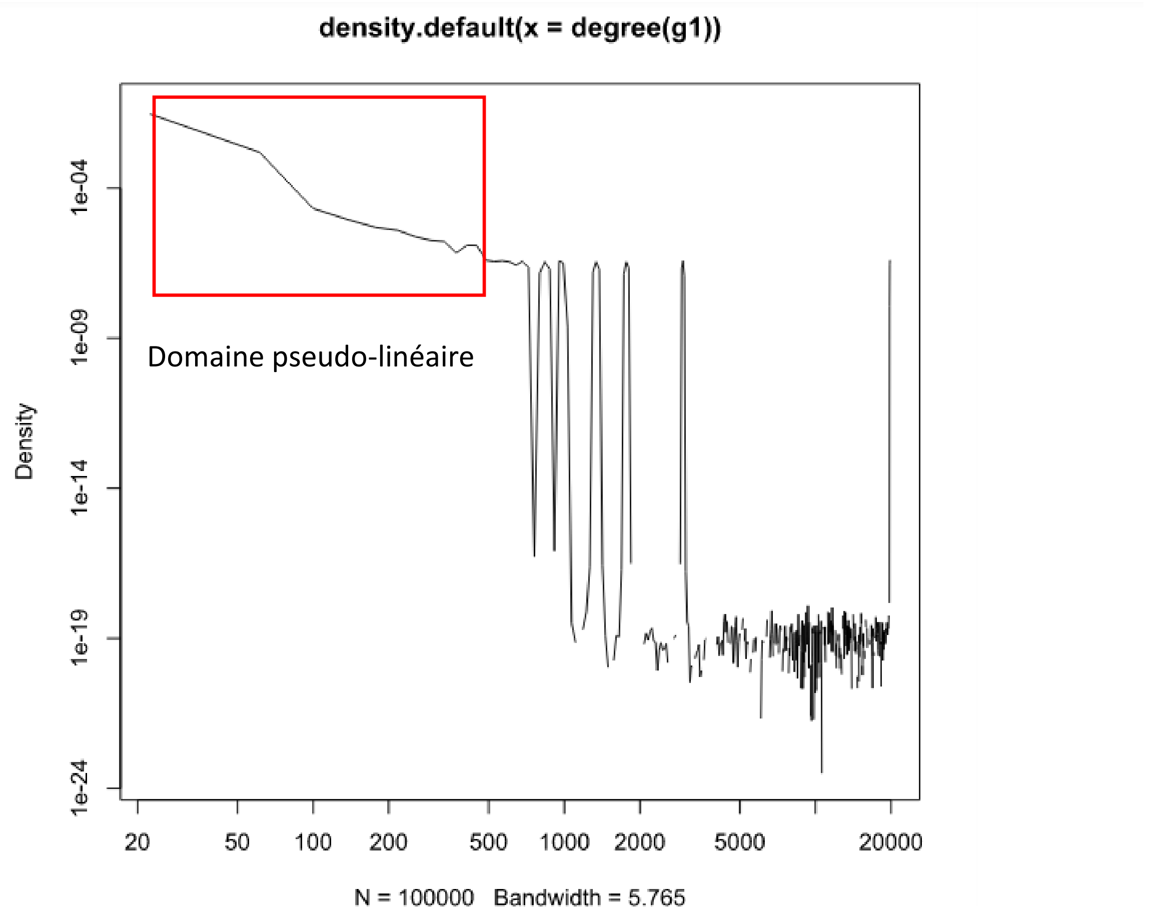
- Nous allons maintenant, ajouter avec la probabilité p l'arête (n,o), sinon sélectionner uniformément un nœud o' parmi les voisins de o, et ajouter l'arête (n,o'). Et nous allons étudier les distributions des degrés des nœuds du réseau résultant pour plusieurs valeurs de p ; on obtient alors les résultats suivants :

## Pour p= 0.1 :

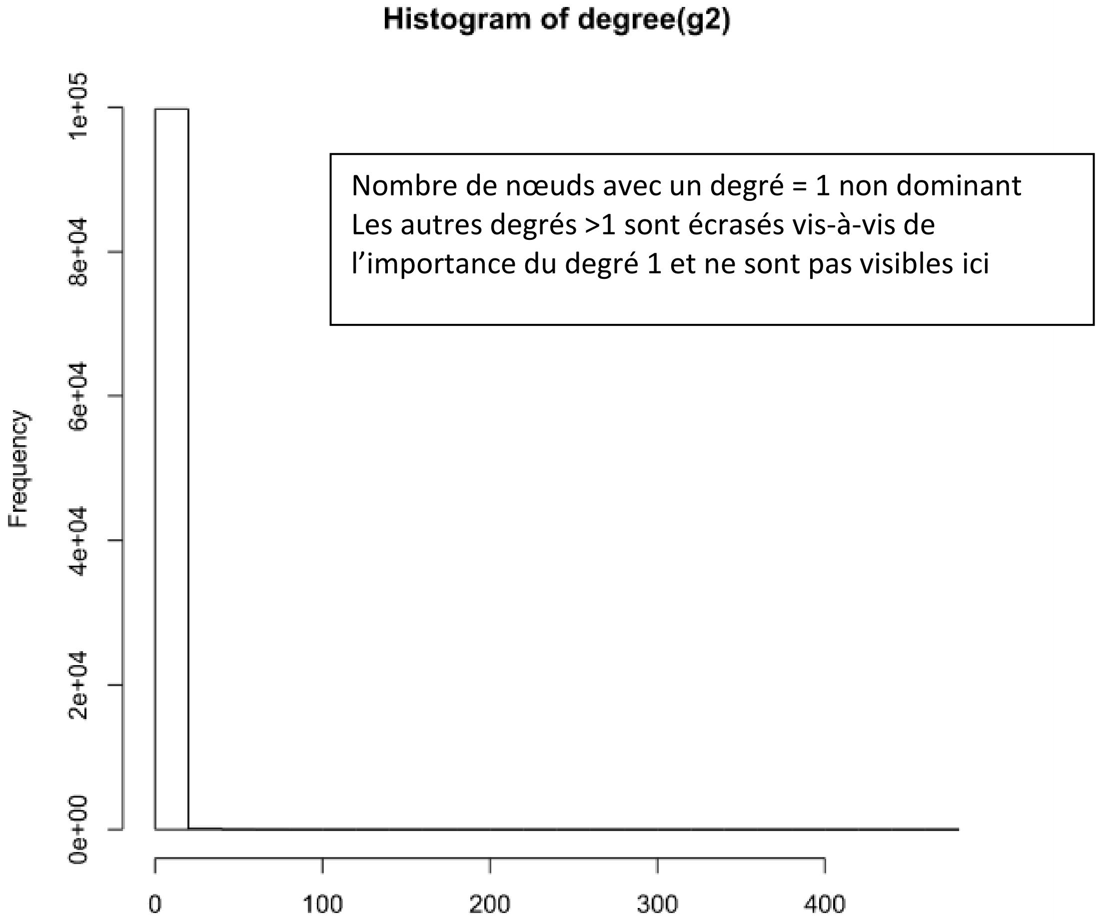
```
library(igraph)
N <- 100000
p <- 0.1
## adjlist est l'ensemble des listes de voisins
adjlist <- vector(mode="list",length=N)
adjlist[[1]] <- c(2,3)
adjlist[[2]] <- c(1,3)
adjlist[[3]] <- c(1,2)
tirages <- runif(N)   # on fait tous les tirages d'un coup

for (i in 4:N) {
  selected <- sample(i-1,1)
  if (tirages[i]<p) {
    # l'arête (n,o) est ajoutée, il y a deux voisinages à modifier
    adjlist[[i]] <- selected
    adjlist[[selected]] <- c(adjlist[[selected]],i)
  } else {
    candidates <- adjlist[[selected]]
    selected <- sample(candidates,1)
    # l'arête (n,o') est ajoutée, il y a deux voisinages à modifier
    adjlist[[i]] <- selected
    adjlist[[selected]] <- c(adjlist[[selected]],i)
  }
}

g1 <- graph_from_adj_list(adjlist,mode="all")
# et s'arrête là.
degrees <- degree(g1)

plot(hist(degree(g1)))
plot(density(degree(g1)),log="xy")
```

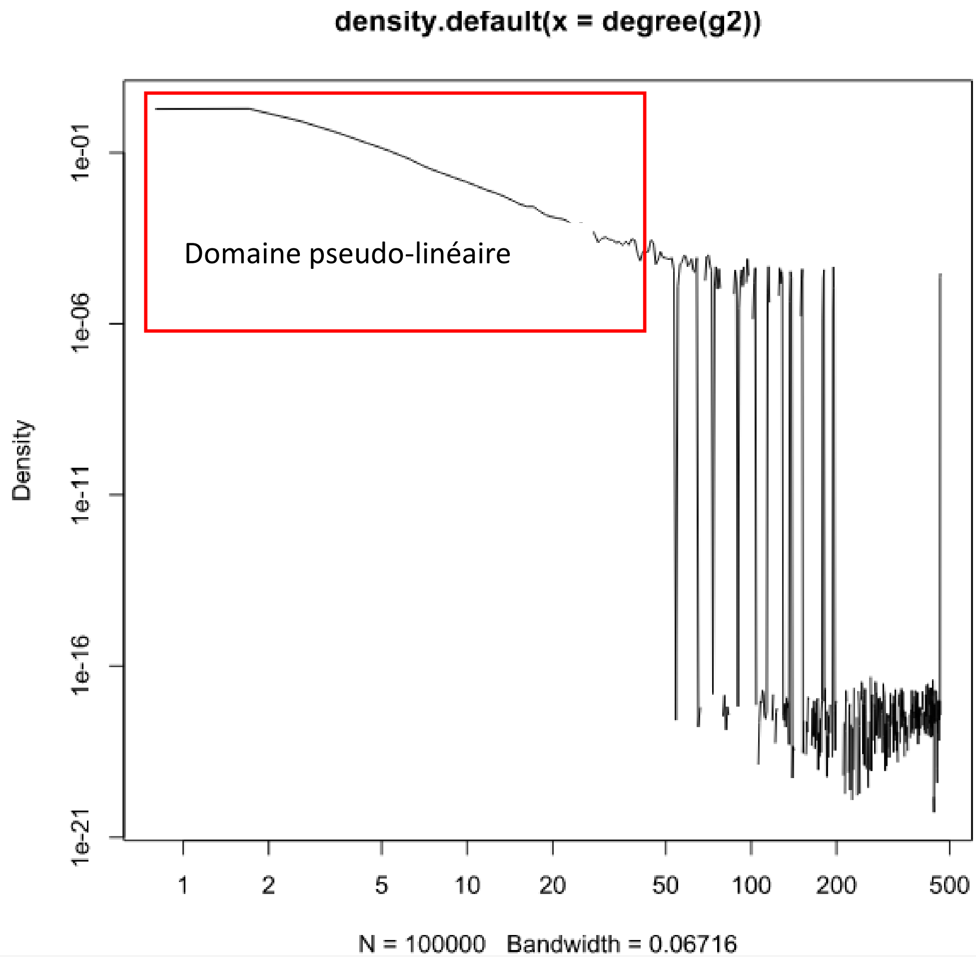
On obtient les graphes suivants :



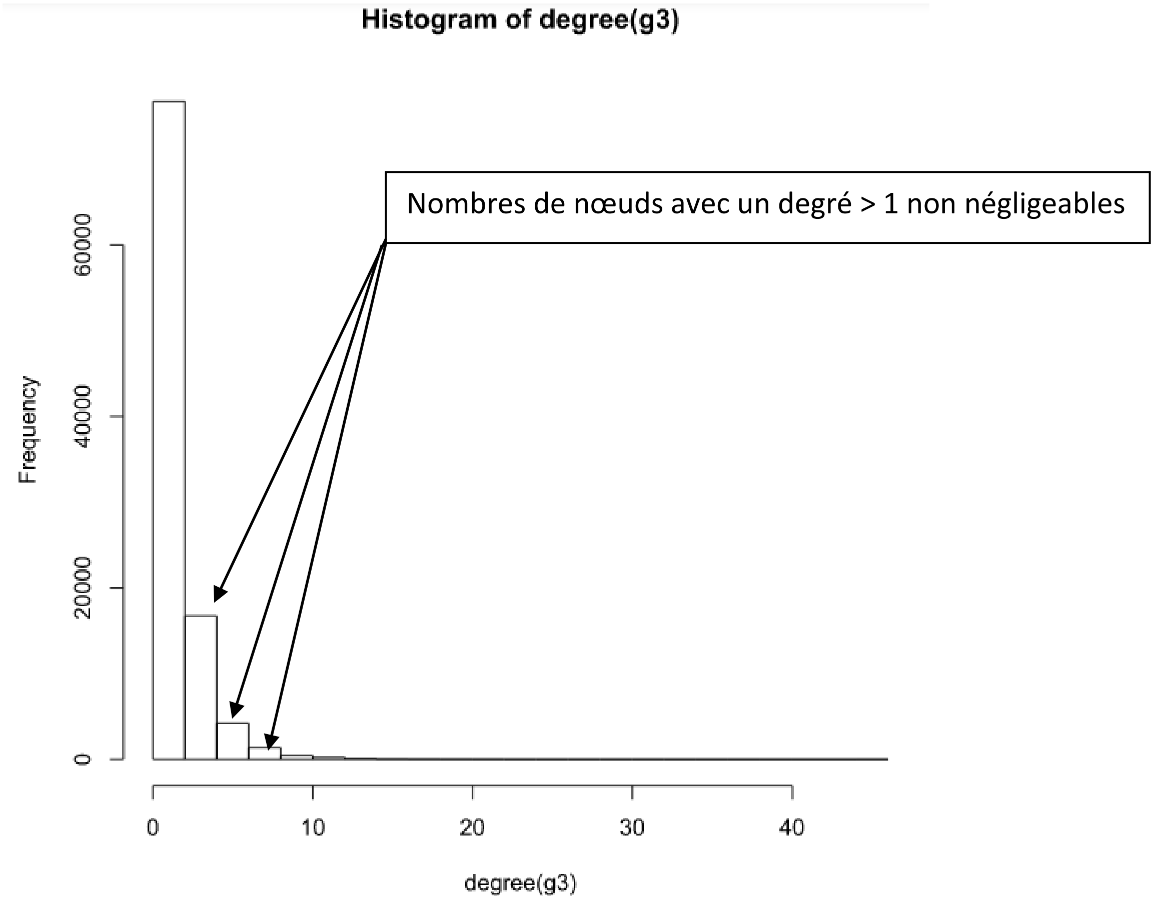



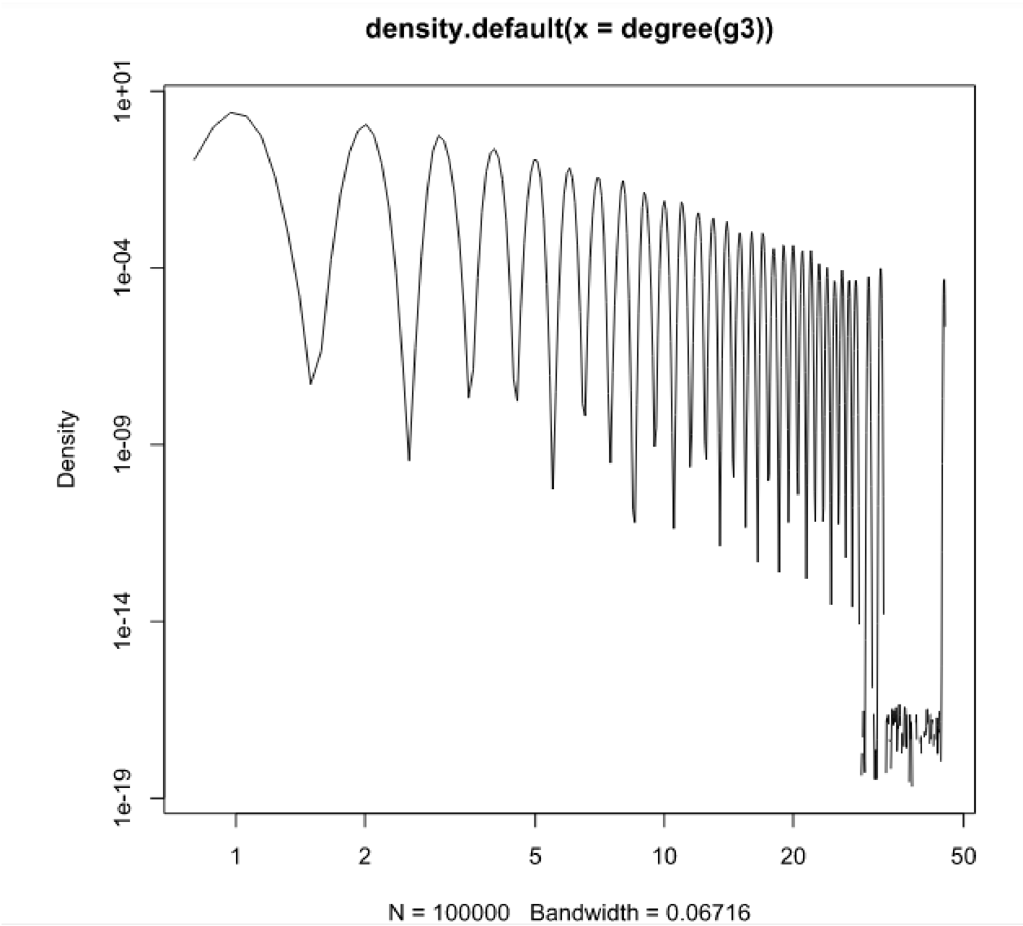
## Pour p= 0.2 :


Histogram of degree(g2)
Frequency
0e+00
2e+04
4e+04
6e+04
8e+04
1e+05
0
100
200
300
400
Nombre de nœuds avec un degré = 1 non dominant
Les autres degrés >1 sont écrasés vis-à-vis de l'importance du degré 1 et ne sont pas visibles ici


density.default(x = degree(g2))
Density
1e-01
1e-06
1e-11
1e-16
1e-21
1
2
5
10
20
50
100
200
500
N = 100000 Bandwidth = 0.06716
Domaine pseudo-linéaire

**Pour p=0.8 :**


Histogram of degree(g3)
Nombres de nœuds avec un degré > 1 non négligeables
Frequency
0
20000
40000
60000
0
10
20
30
40
degree(g3)

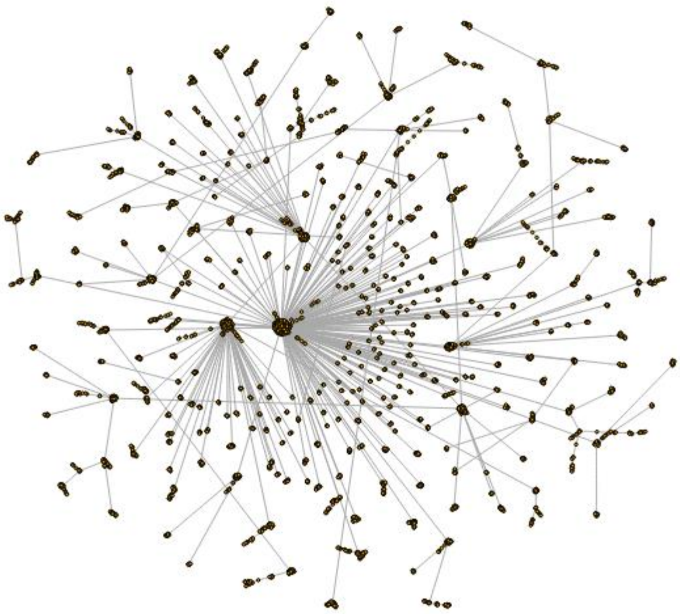

density.default(x = degree(g3))
Density
1e+01
1e-04
1e-09
1e-14
1e-19
1
2
5
10
20
50
N = 100000 Bandwidth = 0.06716

**CONCLUSION :**

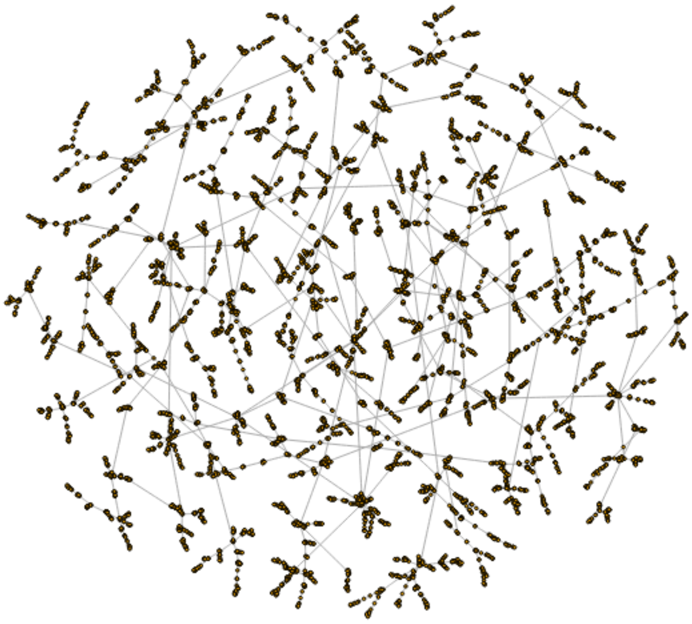
Nous remarquons que plus la probabilité p est faible, le nombre de degré égale à 1 est si important qu'il occulte les autres nombres de degrés supérieurs.

Néanmoins lorsque p tend vers 1, le nombre de degrés supérieurs à 1 (c'est-à-dire le nombre de nœuds possédant plus d'une arête) devient non négligeable.

(N=10000, p=0.1)



(N=10000, P=0.8)



Cette illustration montre bien que plus la probabilité p est faible, plus l'existence de quelques 'hubs' concentrant nombre élevé de degrés. Néanmoins ce faible nombre de hubs est écrasé par le grand nombre de nœuds ayant un degré égale à un (cf. histogramme).

Lorsque la probabilité p tend vers 1, nous constatons que les nœuds se répartissent les arêtes de meilleur façon : disparition des hubs qui concentraient un grand nombre n'arêtes. C'est pourquoi nous observons, selon l'histogramme, que les degrés supérieures à un sont également présent en nombres conséquents.

### b- Power_law :

On applique la fonction suivante : `fit_power_law(degree(g))` pour les trois valeurs de p et on obtient les résultats récapitulés dans le tableau ci-dessous :

| Probabilité | P=0.1 | P=0.2 | P=0.8 |
|---|---|---|---|
| **Continuous** | FALSE | FALSE | FALSE |
| **Alpha** | 2.82465522102459 | 2.75968461729794 | 5.28963198932339 |
| **xmin** | 2 | 2 | 11 |
| **logLik** | -36392.3129261455 | -47963.4946130432 | -990.156876545762 |
| **KS.stat** | 0.00109348335208272 | 0.00612531645870662 | 0.0195495063525525 |
| **KS.p** | 0.0053463770150231 | 0.187447108023777 | 0.994411903175673 |

**CONCLUSION :**

L'application de la fonction power_law sur notre réseau nous permet de savoir si la distribution de notre réseau s'adapte bien à une loi de puissance :

KS.stat : représente le résultat du test statistique de Kolmogorov-Smirnov, et on remarque que plus la probabilité augmente, plus Ks.Stat augmente, et plus la loi du réseau s'éloigne de la distribution en « power law »

KS.p : il s'agit de la p-value du test de Kolmogorov-Smirnov. On remarque alors que plus la probabilité p augmente, plus le teste rejette l'hypothèse que les données ont été tirées d'une distribution de loi de puissance.

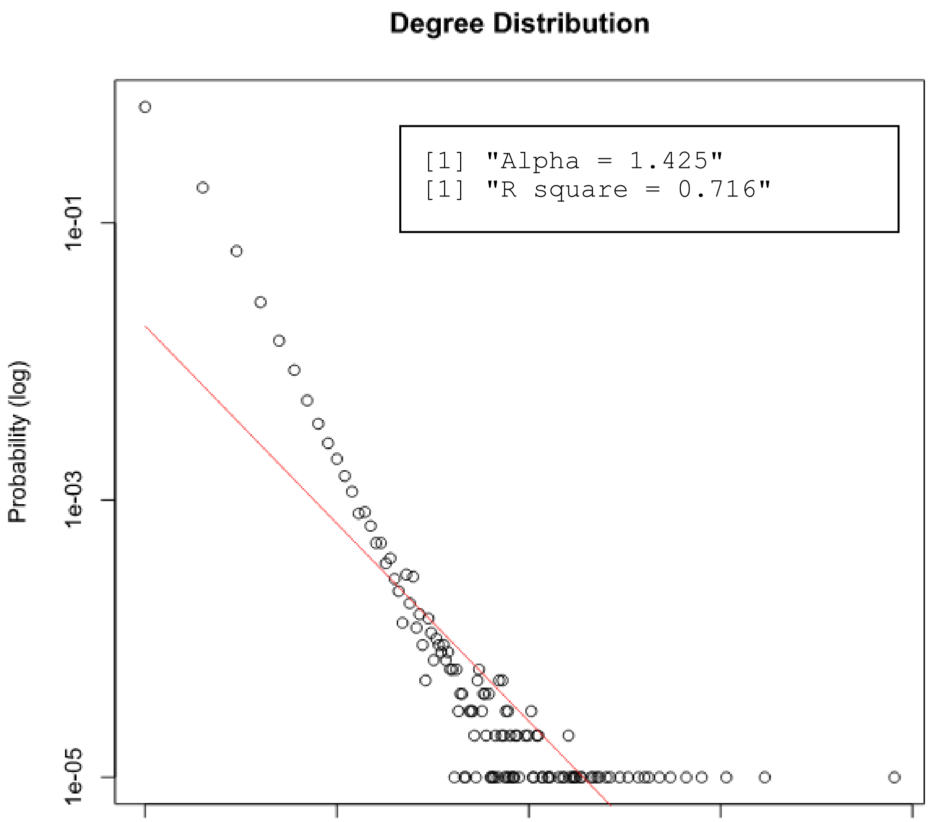
**REMARQUE :**

Pour p tendant vers 1, le test rejette presque totalement cette hypothèse. Ce qui s'illustre correctement avec le graphe de densité des degrés en échelle log qui présente un domaine pseudo-linéaire pour des probabilités faibles chose qui est totalement rejeté avec une probabilité tendant vers 1.

On s'intéresse par la suite au coefficient alpha que l'on cherchera à déterminer par régression linéaire.

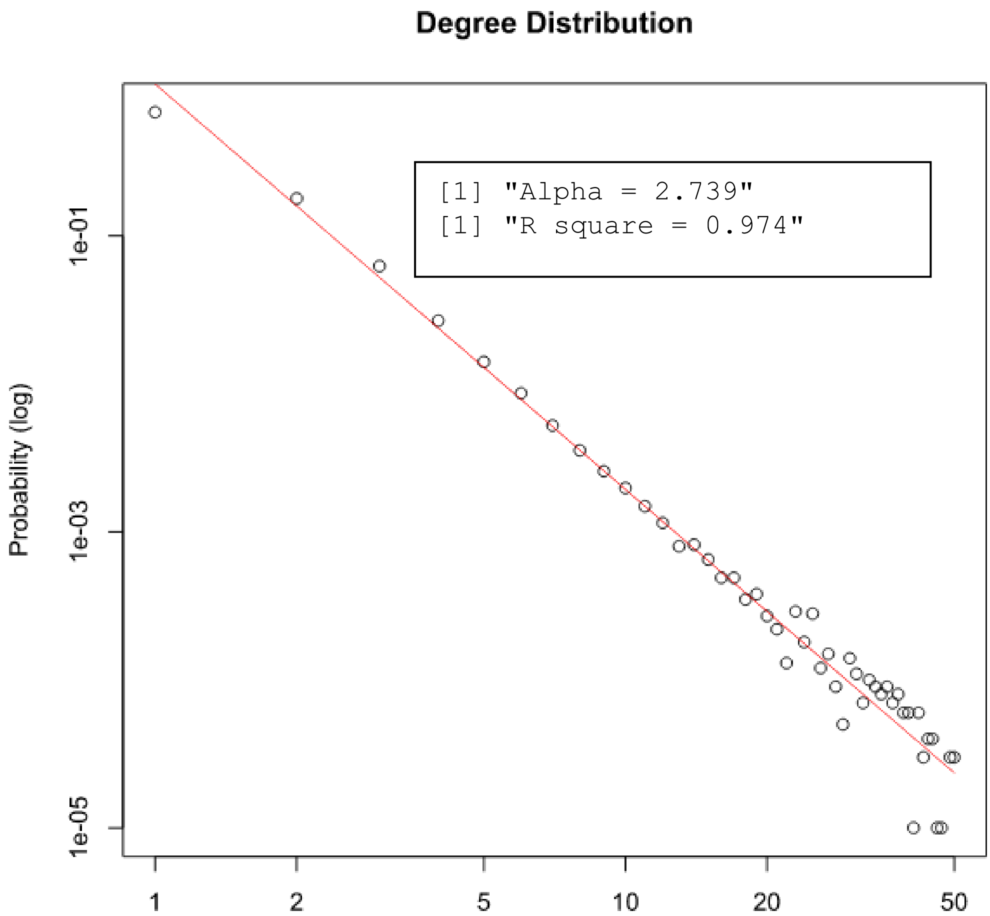
**c- En utilisant la fonction fit_power_law() du paquet igraph, nous allons estimer l'exposant du graphe résultant de p=0.2**

```
# plot and the power law distribution
fit_power_law = function(graph) {
    # calculate degree
    d = degree(graph, mode = "all")
    dd = degree.distribution(graph, mode = "all", cumulative = FALSE)
    degree = 1:max(d)
    probability = dd[-1]
    # delete blank values
    nonzero.position = which(probability != 0)
    probability = probability[nonzero.position]
    degree = degree[nonzero.position]
    reg = lm(log(probability) ~ log(degree))
    cozf = coef(reg)
    power.law.fit = function(x) exp(cozf[[1]] + cozf[[2]] * log(x))
    alpha = -cozf[[2]]
    R.square = summary(reg)$r.squared
    print(paste("Alpha =", round(alpha, 3)))
    print(paste("R square =", round(R.square, 3)))
    # plot
    plot(probability ~ degree, log = "xy", xlab = "Degree (log)", ylab =
"Probability (log)",
        col = 1, main = "Degree Distribution")
    curve(power.law.fit, col = "red", add = T, n = length(d))
}

fit_power_law(g2)
```



On change le domaine d'étude pour ne prendre que les valeurs dans le domaine linéaire.

```
# plot and fit the power law distribution
fit_power_law = function(graph) {
    # calculate degree
    d = degree(graph, mode = "all")
    dd = degree.distribution(graph, mode = "all", cumulative = FALSE)
    degree = 1:max(d)
    probability = dd[-1]
    # delete blank values
    nonzero.position = which(probability != 0 & degree<=50)
    probability = probability[nonzero.position]
    degree = degree[nonzero.position]
    reg = lm(log(probability) ~ log(degree))
    cozf = coef(reg)
    power.law.fit = function(x) exp(cozf[[1]] + cozf[[2]] * log(x))
    alpha = -cozf[[2]]
    R.square = summary(reg)$r.squared
    print(paste("Alpha =", round(alpha, 3)))
    print(paste("R square =", round(R.square, 3)))
    # plot
    plot(probability ~ degree, log = "xy", xlab = "Degree (log)", ylab
= "Probability (log)",
        col = 1, main = "Degree Distribution")
    curve(power.law.fit, col = "red", add = T, n = length(d))
}

fit_power_law(g2)
```

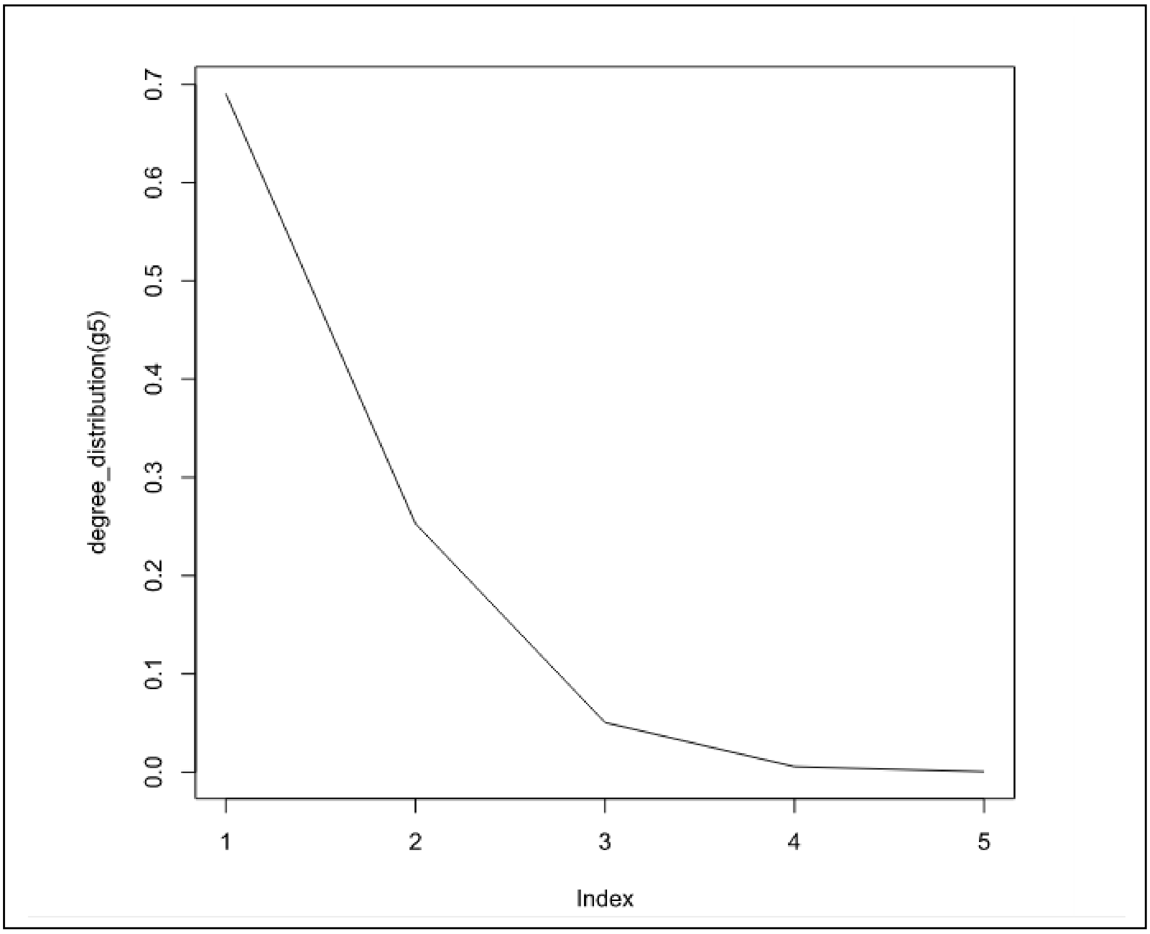


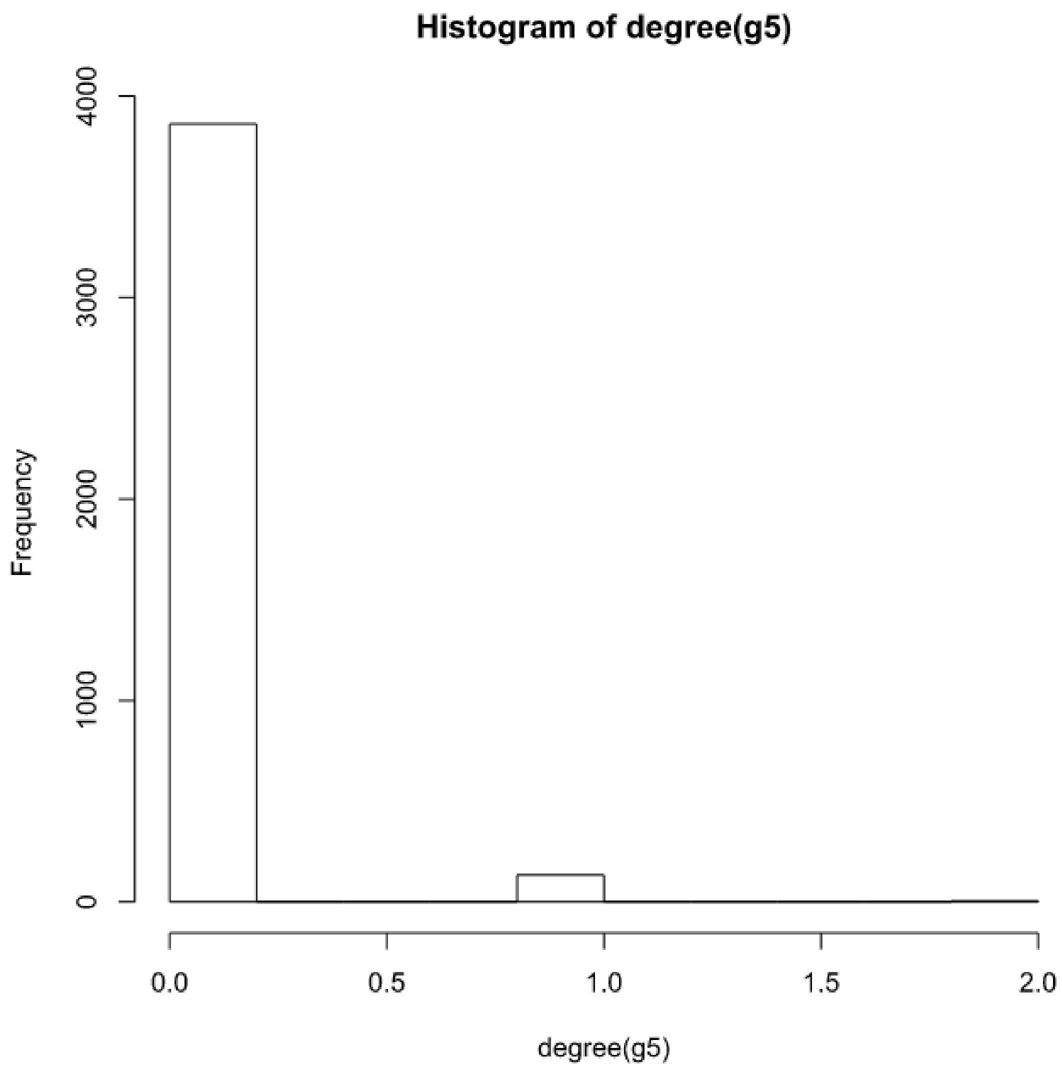
**CONCLUSION :**

On constate bien que l'on retrouve une valeur alpha similaire à celle trouvée grâce à *power_law* via la régression linéaire sur la distribution des degrés : 2.759 via power law et 2.739 via regression linéaire.
Il faut néanmoins pour cela restreindre le domaine d'études au domaine de pseudo-linéarité, autrement on s'écarte de la valeur théorique.

### d- On utilise désormais la fonction `erdos.renyi.game` pour générer un graphe

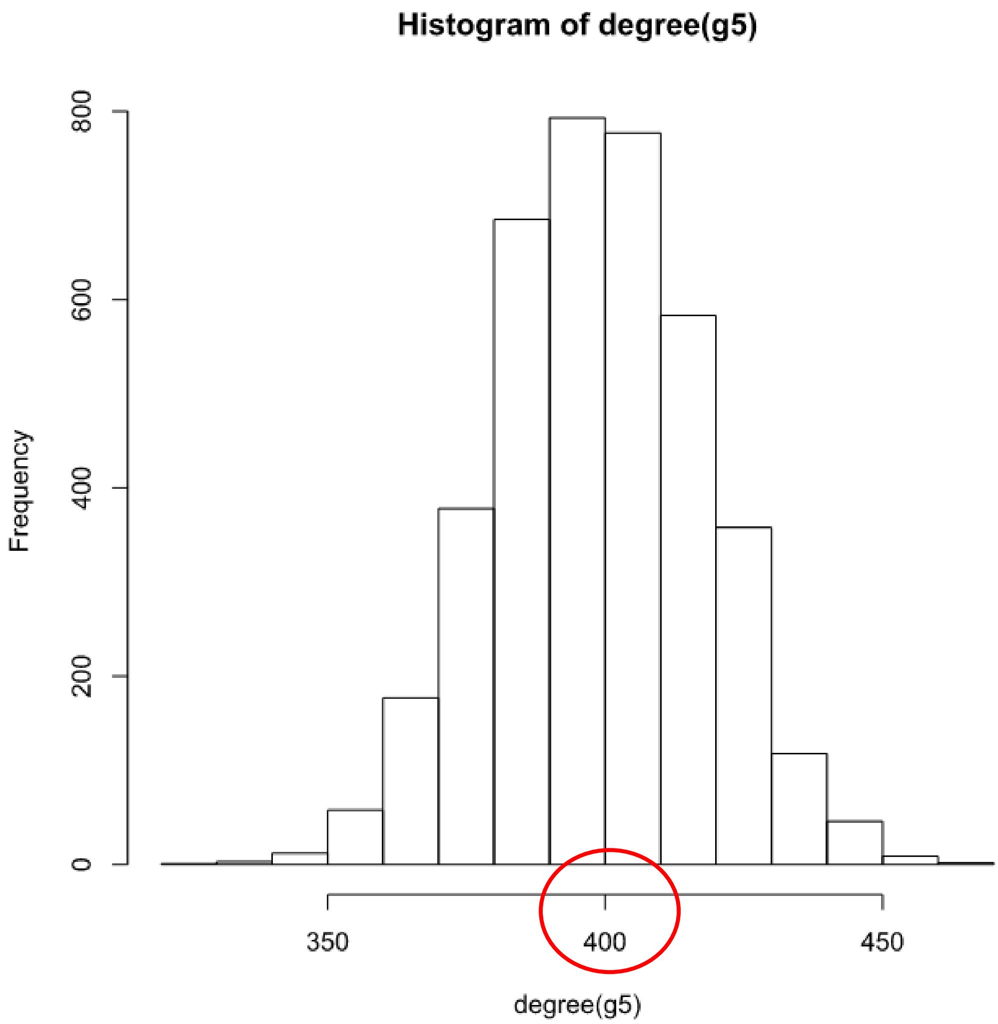
```
g5<-erdos.renyi.game(4000,0.0001,type="gnp")
plot(degree_distribution(g5),type='l')
```





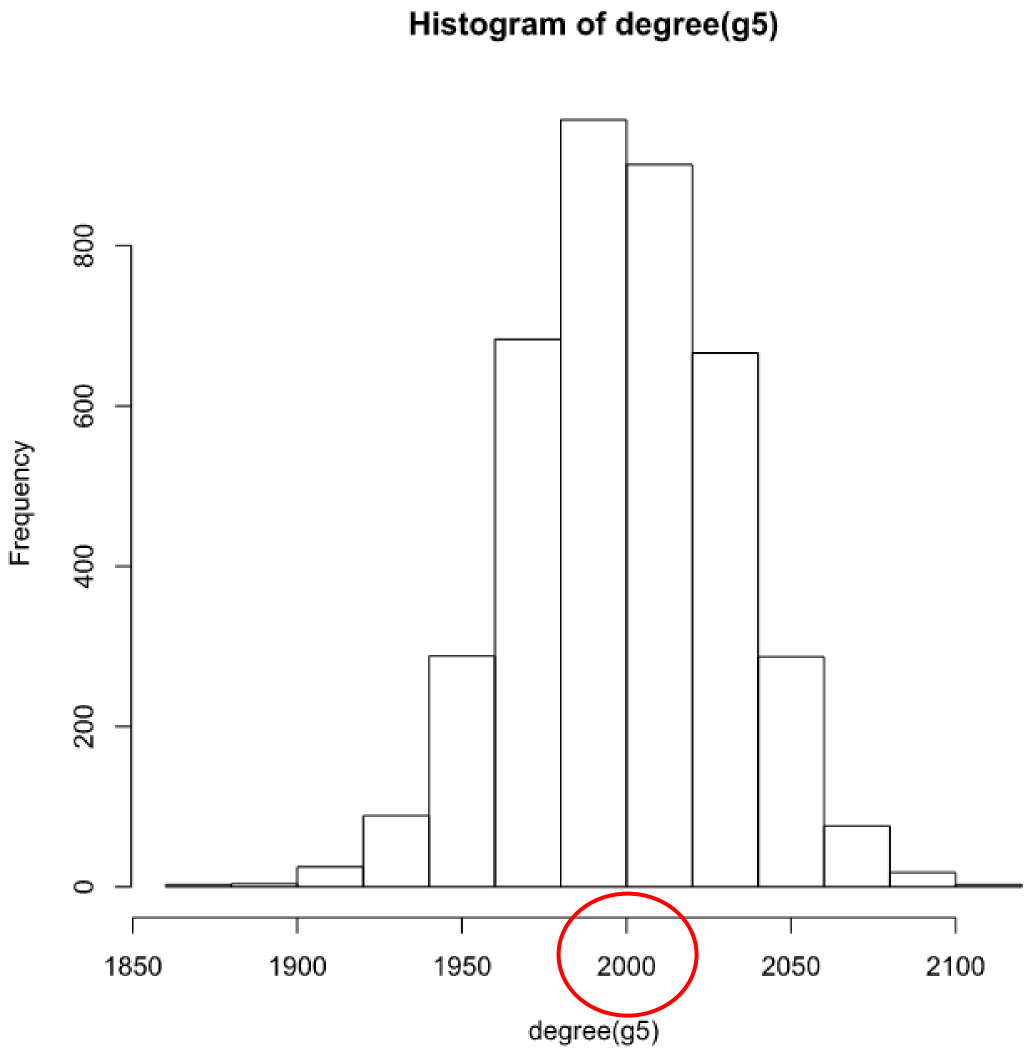

Histogramme des degrés pour p=0.0001

On obtient un graphe vide avec la quasi-totalité des nœuds à degré nul.




Histogramme des degrés pour p=0.1

On obtient un histogramme en Bell-curve, avec pour sommet de cloche 400 or
400 = Nombres de Nœuds * p
Degrés élevés pour un très grand nombre de nœuds.




**Histogram of degree(g5)**

Histogramme des degrés pour p=0.5

On obtient un histogramme en Bell-curve, avec pour sommet de cloche 2000 or

2000 = Nombres de Nœuds * p

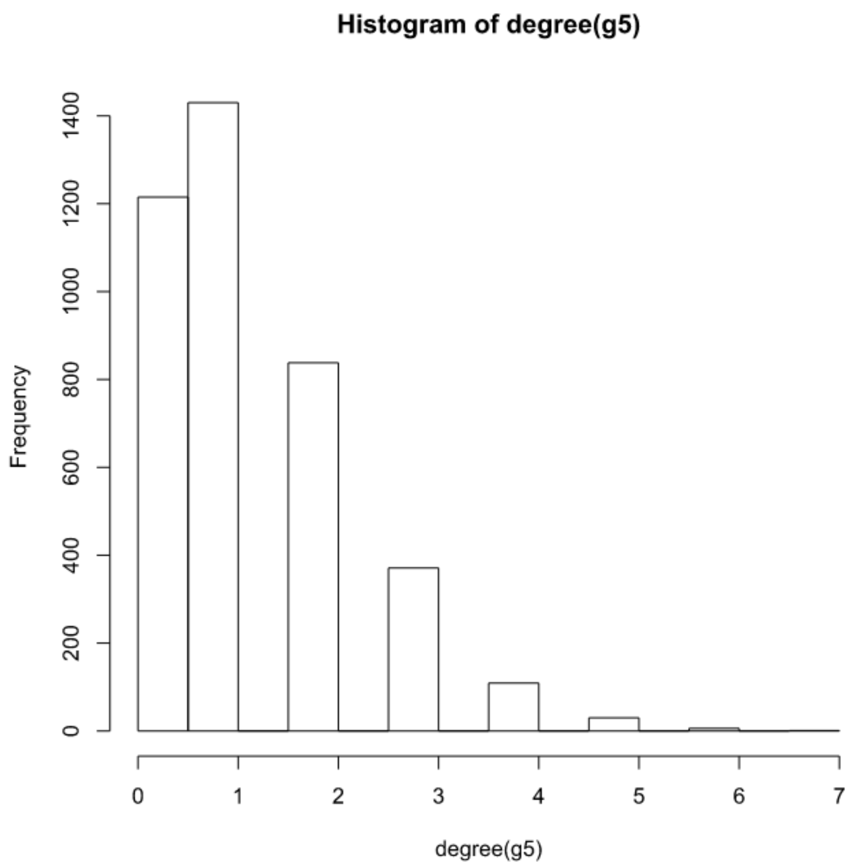
Degrés élevés pour un très grand nombre de nœuds. Plus la probabilité p se rapproche de 1, plus on se rapproche d'un graphe complet où tous les nœuds sont de degrés N (= nombre de nœuds = degrés) puisque tous les nœuds sont reliés à chacun d'entre eux.

## CONCLUSION:

On remarque qu'en utilisant la fonction erdos.renyi.game on a généré une distribution de degrés des réseaux qui n'est plus power law.

Mais si

- p est petit (< 1/N, pour N le nombre de nœuds), alors de nombreux nœuds ne seront pas connectés entre eux, on se rapproche d'un graphe vide.
- p de l'ordre de 1/N, on obtient des composantes géantes du types :




- p est de l'ordre de log(n)/n on a graphe qui ne présente plus de nœuds isolés.
- p>>log(n)/n on obtient une distribution en Bell curve, et pour p tendant vers 1 on se rapproche d'un graphe complet.