```
# Ceci est un script R permettant de reproduire et de visualiser les analyses
présentées dans le Chapitre 2 du QSJ n° 3399, Réseaux sociaux et structures
relationnelles
# PUF, 2014, 3e ed. Il a été produit par Maud Gellée, Alexis Gomes-Matias et
Emmanuel Lazega en mai 2019.
# Son objectif est de faciliter la prise en main de R en analyse de réseaux
sociaux, niveau débutant.
# Les données utilisées par les exemples : le réseau de conseil (advice.dat,
n=71, inclu dans lawfirm data disponible sur elazega.fr/datasets).
# Avant d'utiliser ce script, il est recommandé de suivre une formation
initiale minimale et plus générale à R.
# Un Guide d'utilisation (installation, exécution, etc.) accompagne ce script.

# Table des matières du script:
# Guide d'utilisation
# Installation
# Préparation des inputs
# Centralités
# Cliques


# ======================================== GUIDE D'UTILISATION
========================================

# Le lancement du script doit être réalisé en plusieures étapes et il est
nécessaire d'ordonner ses répertoires afin de retrouver facilement
# les différentes composantes du script.
# Nous vous conseillons d'avoir trois répertoires :
# - Un premier qui contiendra les scripts
# - Un deuxieme qui stockera les données
# - Un dernier qui récupérera les sorties (outputs)
# Une fois ces trois répertoires créés, il vous suffit de lancer R qui vous
servira de console afin d'exécuter et d'interagir avec les scripts.
# Tout d'abord, il est nécessaire de lancer le script « installation.r », en
glissant le fichier dans la console, qui va installer toutes
# les bibliothèques (ensemble de codes qui permettent l'ajout de nouvelles
fonctions) nécessaires au bon fonctionnement du script de traitement des
données du QSJ.
# Si l'utilitaire ouvre une boite de dialogue vous demandant si vous souhaitez
utiliser une bibliothèque personnelle, cliquez sur « oui » deux fois puis il
vous est
# demandé de choisir un lien miroir de téléchargement, veuillez choisir
l'endroit le plus proche de vous (Paris II par exemple) afin d'avoir le temps
d'installation
# le plus court possible.
# En cas d'erreur, n'hésitez pas à relancer R, cela supprime toutes les
bibliothèques chargées et peut s'avérer utile en cas de conflit entre celles-
ci.
```

```
# Une fois les bibliothèques installées, il suffit d'exécuter le script du QSJ qui vous demandera de saisir un dossier dans lequel il stockera les outputs.
# Vous sélectionnerez alors le dossier précédemment créé où vous verrez apparaître les sorties du programme.
# Ensuite, une nouvelle fenêtre s'ouvrira afin de vous demander les données que vous souhaitez utiliser, vous pourrez alors choisir les données
# que vous avez dans le répertoire prévu à cet effet (advice_avocats, amitie_avocats, cowork_avocats, ou autre) et le script se lancera automatiquement après ceci.
# Il vous demandera plus tard d'entrer de nouvelles données : il s'agit d'une table de densité.
# De temps en temps, sur la console R apparaîtra un message vous demandant d'écrire dans ladite fenêtre (choisir le nombre de blocs de CONCOR par exemple).
# Il vous suffit d'écrire un chiffre entier (2,3 …). Pour la table de densité, veuillez utiliser la notation anglaise des nombres décimaux (1.2 et non 1,2) puis appuyer sur entrée.

# Guide des sorties (outputs) :
# A la fin de l'exécution du script, si tout s'est bien déroulé, il y aura 5 fichiers de sortie :
# - Output_dataFrame_centralite
# - Output_script_QSJ
# - Output_matriceImageDensite
# - Output_CONCOR

# Output_dataFrame_centralite
# Ce fichier excel (.csv) est une retranscription du tableau rassemblant, pour chaque acteur, un ensemble de mesures sur leur position dans le réseau social.
# Ces données sont les demi-degrés intérieurs et extérieurs normalisés et non normalisés de chacun, ainsi que leur centralité d'intermédiarité.

# Output_script_QSJ
# Ce pdf est un condensé des différents graphes affichés par le script du QSJ, il est composé de 4 pages.
# Le premier graphe est le réseau précédemment sélectionné dans lequel la taille des nœuds évolue selon le nombre de kcores dans lequel un sommet se trouve;
# le second est le dendrogramme des cliques du réseau, la troisième est le blockmodel du réseau,
# le quatrième représente le réseau entier en scindant les individus par appartenance à un bloc et enfin le dernier représente les dépendances entre chacun des blocs
# du blockmodelling.
# Output_matriceImageDensite
# Ce fichier excel (.csv) est une matrice d'adjacence qui, pour une table de densité, représente les liens entre plusieurs blocs d'individus
# en fonction d'un seuil précisé.
#Output_CONCOR
```

```r
# Ce fichier excel (.csv) est une matrice carrée représentant pour chaque
acteur (ici avocat) les coefficients de corrélation associés.


#
================================================================================
========================

# Tout d'abord, veillez à lancer le script d'installation (installation.R) des
différents packages avant de lancer ce script

# Si vous voulez acquérir les bases en R, suivez ce tutoriel :
# https://openclassrooms.com/fr/courses/1393696-effectuez-vos-etudes-
statistiques-avec-r

# pour executer ce script : source("chemin du script") (ex :
source("C:\\Users\\nomUtilisateur\\R\\script.r")) ou glissez l'icone du script
sur la console

# initialisation du workspace
library(tcltk)
setwd(tk_choose.dir(caption="Choisissez le repertoire de travail (l'endroit ou
les outputs seront stockes)"))


#======================================== PREPARATION DES INPUTS
====================================

# on lit le fichier qui sera sélectionné
fichier <- tk_choose.files(caption = "Choisissez la matrice d'adjacence a
analyser")
# et on importe les données dans la matrice d'adjacence, importée en tant que
liste (is.list(advice) = TRUE)
# il existe une fonction similaire, read.csv(file = nom du fichier), mais le
séparateur par défaut est la virgule et non le point-virgule
# si vous voulez lire un fichier dont vous voulez spécifier le séparateur,
utilisez read.table()
# (voir la doc pour plus de details :
https://www.rdocumentation.org/packages/utils/versions/3.6.0/topics/read.table
)
advice <- read.csv2(fichier, header=TRUE, row.names=1)

# extraction du nom du fichier d'où proviennent les données selectionnées
# afin de signifier dans le nom du fichier d'output quel données d'input ont
été utilisées
noms <- strsplit(fichier, "[\\]")
fic <- noms[[1]][length(noms[[1]])]
fic <- strsplit(fic, "[/]")
fic <- fic[[1]][length(fic[[1]])]
```

```r
fic <- strsplit(fic, "[.]")
fic <- fic[[1]][1]

#=======================================
CENTRALITES=============================================

# on charge la bibliothèque statnet
library(statnet)
# attention, statnet et igraph peuvent rentrer en conflit, la dernière
bibliothèque chargée est la prioritaire,
# si vous voulez utiliser une bibliothèque précédente, il faut décharger la
bibliothèque :
# detach(package: nom du package)

# ou bien, si vous souhaitez utiliser une fonction particulière d'une
bibliothèque, mais que cette dernière peut entrer en conflit avec une autre
(voir exemple ligne 62) :
# nomDeLaBibliothèque::nomDeLaMethode(...)

#
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++

# récupération des demi-degrés intérieurs (indegree) et extérieurs (outdegree)
inDegree <- sna::degree(advice, cmode="indegree")
outDegree <- sna::degree(advice, cmode="outdegree")

# arrondi à deux chiffres après la virgule
inDegree <- round(inDegree, 2)
outDegree <- round(outDegree, 2)

# calcul des demi-degrés normalisés
# (on divise par le nombre de sommets du graphe -1 et on multiplie par 100, cf
"Que sais je ?")
inNormDegree <- (inDegree/(length(advice) - 1)) * 100
outNormDegree <- (outDegree/(length(advice) - 1)) * 100

# arrondi à deux chiffres après la virgule
inNormDegree <- round(inNormDegree, 2)
outNormDegree <- round(outNormDegree, 2)

# La centralité de proximité :
# la méthode closeness(graph g) existe, cependant elle ne fonctionne pas si un
ou plusieurs sommets du graphe n'émettent rien (uniquement des 0 dans la
colonne)
# il semble lorsque l'on calcule les distances géodésiques que certains
sommets, comme le 6, sont considérés comme isolés
# car il ne "choisit" personne comme conseiller même s'il reçoit des choix
```

```r
# la centralité d'intermédiarité :
betweenness <- sna::betweenness(advice)

# la centralité d'intermédiarité normalisée :
# selon le "Que sais je ?" il faut diviser la centralité d'intermédiarité par
:
# (nombre de sommet du graphe -1)(nombre de sommet du graphe -2)/2
# cependant les résultats obtenue avec cette formule ne correspondent pas aux
résultats du tableau page 46

# la centralité de Bonacich :
# il existe plusieurs méthode s'apparentant à cette centralité :
# infocent(graph g)
# (igraph) alpha_centrality(graph g)
# (igraph) power_centrality(graph g)
# (igraph) eigen_centrality(graph g)
# bonpow(graph g)
# cependant, aucune ne donne les même résultats que le "Que sais je ?" (basé
sur Ucinet)

# on crée un dataFrame (tableau de données) contenant toutes les valeurs
obtenues afin d'avoir un affichage propre
# les variables de gauche (in_degree, out_degre, ...) sont le nom des colonnes
# et celles de droite, les données à stocker
dataFrame <- data.frame(
in_degree = inDegree,
out_degree = outDegree,
Normalized_in_degree = inNormDegree,
Normalized_out_degree = outNormDegree,
betweenness = betweenness)

# écriture du fichier csv stockant les données du data frame
# rappel : fic est le nom du fichier d'input contenant la matrice d'adjacence
write.csv2(dataFrame, file = paste("output_dataFrame_Centralite_",fic,".csv",
sep =""))


# pour limiter les conflits, on détache statnet
detach(package:statnet, unload=TRUE)

#======================================== ETUDE DES CLIQUES
====================================

# chargement de la bibliothèque igraph
library(igraph)
# chargement de la bibliothèque statnet
library(statnet)
# ou, si l'on veut se concentrer sur les arcs entrant ou sortant
library(RBGL)
```

```r
#
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++
# definition des fonctions nécessaires aux calculs

# remplissage de la matrice de co-appartenance
rempMat <- function(len,cli){ #len est le nombre d'acteurs et cli la liste des
cliques du réseau renvoyée par max_cliques
  retMat <- matrix(0, nrow = len, ncol=len) #Initialisation d'une matrice
carrée de taille len
  for(i in 1:len){ #On va remplir la matrice pour chaque sommet
    for(j in 1:length(cli)){ #Pour chacun, on parcourt la liste des cliques
      if((paste("X",i, sep="") %in% names(cli[[j]]))[1]){ #Si X1, X2, X3...
est contenu dans la liste
        retMat[i,i] <- retMat[i,i] + 1 #On incrémente la diagonale
        for(k in i+1:len){ #On va compléter la matrice pour tous les autres
sommets appartenants à la clique actuelle
           if((paste("X",k, sep="") %in% names(cli[[j]]))[1]){
             retMat[i,k] <- retMat[i,k]+1 #On remplit les cases de la matrice
à la ligne i
             retMat[k,i] <- retMat[k,i]+1 #On fait pareil pour la colonne car
la matrice est symétrique
           }
        }
      }
    }
  }
  return(retMat) #retour de la matrice remplie
}

# source de la fonction : Alexander Montgomery/Stackoverflow
nCliques <- function(g,n){
  g <- as.undirected(g)
  E(g)$weight <- 1 #just in case g has weights - does not modify original
graph
  ncliques <- kCliques(ugraph(igraph.to.graphNEL(g))) #get cliques
  n.cand <- ncliques[[n]] #n-clique candidates to be an n-clan
  n.clan <- list() #initializes a list to store the n-clans
  n.clan.i <- 1 #initializes a list pointer
  for (n.cand.i in 1:length(n.cand)){ #loop over all of the candidates
    g.n.cand <- induced_subgraph(g,n.cand[[n.cand.i]]) #get the subgraph
    if (diameter(g.n.cand)<=n){ #check diameter of the subgraph
      n.clan[[n.clan.i]] <- n.cand[[n.cand.i]] #add n-clan to the list
      n.clan.i <- n.clan.i+1 #increment list pointer
    }
  }
  return(n.clan) #return the entire list
}
```

```r
#
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++

# pour stocker un reseau igraph utilise marjoritairement des graphes, on
convertit donc advice en graph
# à l'aide de la fonction graph_from_adjacency_matrix(matrix m)
# la fonction as.matrix(object o) permet de convertir une liste (advice) en
matrice
graph <- graph_from_adjacency_matrix(as.matrix(advice))

# cliques (tous les membres d'une clique sont liés directement à tous les
autres membres) :
# cliques <- cliques(graph, min = NULL, max = NULL) # fonctionne mais ne
correspond pas à UCINET

# n-cliques (tous les membres d'une clique sont liés à tous les autres mais
pas necessairement directement,
# ils ont droit à des chemins de taille n) :
cliques <- max_cliques(graph, min = NULL, max = NULL) # ignore la direction du
graphe et renvoie les même output que UCINET

largestCliques <- largest_cliques(graph) # donne les cliques les plus grandes(
=contenant le plus de sommets)

# nous n'avons rien trouvé qui permettait de faire les k-plex de manière
simple et efficace, nous ne les avons donc pas faits

# k-cores (il doit y avoir minimum k liens entre tous sommets de la clique et
les autres) :
kCores <- kcores(advice)

# modification de la taille des noeuds en fonction du nombre de kcores
auxquels ils appartiennent :
V(graph)$size <- (kCores/4)^2

# méthodes permettant de spécifier si on veut les kcores des chemins entrants
ou des chemins sortants :
# kCoresIn <- kCores(graph, "in")
# kCoresOut <- kCores(graph, "out")

appMatrice <- rempMat(length(advice), cliques) # appMatrice est la matrice de
co-appartenances correspondant à advice

write.csv2(appMatrice, file =
paste("output_scriptQJS_matriceCoAppartenance.csv", sep =""))
```

```r
d <- dist(appMatrice) # calcul de la distance euclidienne entre tous les sommets
# (pour utiliser d'autres méthodes de calculs de distance, il faut ajouter un paramètre, plus d'informations sur:
# https://www.rdocumentation.org/packages/stats/versions/3.5.3/topics/dist)

# modification du placement des labels des noeuds
V(graph)$label.cex = 0.8

# création d'un layout s'adaptant relativement bien aux graphes de grande taille, permettant de rendre le graphe plus lisible
l <- layout_with_graphopt(graph)

# affichage sur la console du graphe où les noeuds n'ont pas la même taille en fonction du nombre de kcores auxquels ils appartiennent
plot(graph, main = "Variation des tailles des noeuds en fonction du nombre de kcores dans lequel un sommet se trouve", layout = l, edge.arrow.size = 0.3, edge.label.color="black",vertex.label.color="black", edge.color="black", vertex.color="#F4FEFE")

# arguments du plot si jamais on veut simplement voir le graphe dans sa globalité (noeuds plus petits mais plus espacés)
#vertex.size=5, edge.arrow.size=0.3

# création du dendrogramme
hc <- hclust(d)

pdf("output_script_QSJ.pdf", width=20, height=20) # adapter la width et la height des pages en fonction de la taille des données

# insertion dans le pdf du graphe où les noeuds n'ont pas la même taille en fonction du nombre de kcores auxquels ils appartiennent
plot(graph, main = "Variation des tailles des noeuds en fonction du nombre de kcores dans lequel un sommet se trouve",
edge.label.color="black",vertex.label.color="black",edge.arrow.size = 0.3, edge.color="black", vertex.color="#F4FEFE")

# on prépare l'export du graphe pour gephi (extension .gexf)
library("rgexf")
V(graph)$id <- paste("X", 1:length(advice[,1]), sep="")
nodes_df <- data.frame(ID = c(1:vcount(graph)), NAME = V(graph)$id)
edges_df <- data.frame(igraph::get.edges(graph, c(1:ecount(graph))))
nodes_size <- V(graph)$size

colors <- rep("#F4FEFE", length(as.matrix(advice)[,1]))
nodes_col <- colors
nodes_col_df <- as.data.frame(t(col2rgb(nodes_col, alpha = FALSE)))
nodes_col_df <- cbind(nodes_col_df, alpha = rep(1, times = nrow(nodes_col_df)))
```

```r
nodes_att_viz <- list(color = nodes_col_df, size = nodes_size)

# ecriture du graphe dans un fichier .gexf
write.gexf(nodes = nodes_df, nodesVizAtt = nodes_att_viz, edges = edges_df,
defaultedgetype = "directed", output =
"kcores_taille_du_noeud_correspond_au_nombre_de_kcores_dans_lequel_lacteur_se_
trouve.gexf")

# insertion du dendrogramme, hang = -1 permet d'aligner tous les acteurs sur
une meme ligne dans le graphe
plot(hc, hang = -1, main="Dendrogramme des cliques")


# pour limiter les conflits, on détache les bibliothèques utilisées dans cette
partie
detach(package:rgexf, unload=TRUE)
detach(package:igraph, unload=TRUE)
detach(package:statnet, unload=TRUE)
detach(package:RBGL, unload=TRUE)

#======================================== BLOCKMODELLING (CONCOR)
========================================

# limite propre à la bibliothèque concoR: attention, la partie suivante ne
peut pas fonctionner avec des matrices d'adjacence
# dont l'une des colonne est uniformément remplie (une colonne de 0 ou une
colonne de 1)
# il faut donc que tous les sommets soient choisis au moins une fois

# nous avons suivi le tutoriel suivant (attention le site peut ne pas
fonctionner sur tous les navigateurs) :
# https://rpubs.com/pjmurphy/325575

# chargement de la bibliothèque statnet
library(statnet)
# chargement de la bibliothèque igraph
library(igraph)
# chargement de la bibliothèque concoR
library(concoR)
# chargement de la bibliothèque RColorBrewer
library(RColorBrewer)

#
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++
# definition des fonctions nécessaires aux calculs
```

```r
# fonction permettant de créer une matrice d'adjacence à partir des
blocksmodels
# block -> blks$block
# dens -> blk_mod[5]
adjMat <- function(block, dens, seuil){
    retMat <- matrix(0, nrow = max(block), ncol=max(block)) # initialisation
d'une matrice carrée de la taille du nombre de blockmodels
    dens <- as.matrix(dens[[1]]) # selection de la table de densité
    for(i in 1:length(dens[1,])){ # parcours de la table
        for(j in 1:length(dens[,1])){
            if(dens[i,j] >= seuil){ # si la densité est supérieur au seuil
donné
                retMat[i,j]<- 1 # il y a un lien entre les deux blocks
            }
        }
    }
    return(retMat)
}

#
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++

# conversion en matrice
matrice <- as.matrix(advice)

# conversion en graphe
g <- graph_from_adjacency_matrix(matrice)

# conversion en réseau
net<-network(matrice, directed=TRUE, matrix.type="adjacency")


# recupération de la matrice d'adjacence du graphe
mat <- as.matrix(get.adjacency(g))

# calcul des coefficients de correlations
m0 <- cor(mat)
# remplace tous les NA par des 0
m0[is.na(m0)] <- 0

# arrondit des coefficients au centième
round(m0, 2)

# création d'un fichier .csv contenant les coefficients de corrélations
write.csv2(m0, file =
paste("output_CONCOR_matrice_finale_coefficients_correlation_",fic,".csv", sep
=""))
```

```r
print("Veuillez ecrire un nombre de blocs a creer : ")
# on lit la densité choisie par l'utilisateur (si vous entrez 2, 4 blocks
seront créés)
# attention, si le nombre choisit est trop grand, il devient compliqué de
différencier les couleurs des noeud pour le prochain graphe
nb <- scan(nmax=1, what=double())

# création d'un vecteur d'appartenance necessaire à la fonction blockmodel
blks <- concor_hca(list(mat), p = nb)

# création du graphe
blk_mod <- blockmodel(net, blks$block)

# création d'un attribut blocks pour chaque sommet, permettant de savoir à
quels blocks ils appartiennent
# cet attribut permettra de modifier la couleurs des sommets en fonction de
leurs appartenance
V(g)$blocks <- blks$block

# création d'un palette de couleurs contenant le même nombre de couleurs qu'il
y a de blocks
colors <- rainbow(length(unique(blks$block)))

# création d'un tableau contenant la couleur d'un noeud en fonction de son
block model
node_colors <- rep("", length(blks$block))
# remplissage du tableau de couleurs
for(i in 1:length(blks$block)){ # on parcourt tous les noeud
    p <- blks$block[i] # on regarde à quel block il appartient
    indice <- match(p, unique(blks$block)) # on cherche l'indice de la couleur
correspondant au block
    node_colors[i] <- colors[indice] # on donne au sommet i la couleur
correspondante
}

# création d'un vecteur contenant des nombres
bloques <- vector("numeric", max(blks$block))

# on remplit le vecteur, il contiendra à le nombre de sommet en fonction du
block
for(i in 1:max(blks$block)){
    bloques[i] <- sum(blks$block == i)
}
expt <- blks[1]

blocksList <- vector()
for(i in 1:length(rownames(blks))){
  if(is.na(blocksList[blks[[1]][i]])){
    blocksList[blks[[1]][i]]<- blks[[2]][i]
```

```
  }else{
  blocksList[blks[[1]][i]]<- paste(blocksList[blks[[1]] [i]], blks[[2]][i],
sep=",")
  }
}

blocksList <- as.data.frame(blocksList)
write.csv2(blocksList,
file="output_scriptQSJ_composition_de_chacun_des_blocks.csv")

write.csv2(expt, file =
"output_scriptQSJ_variable_block_appartenance_de_chaque_individu.csv")

# création de la matrice d'adjacence
matrice <- adjMat(blks$block, blk_mod[5], 0.302)

# on crée le graphe associé
gBlock <- graph_from_adjacency_matrix(matrice)

# on change la taille des nouveaux sommets (les blocks) en fonction du nombre
de sommet qu'ils contiennent
V(gBlock)$size <- bloques*2

# modification pour positionner la matrice
par(mfrow=c(2,2),oma=c(0,0,4,0))
# insertion dans le pdf de la matrice CONCOR
plot(blk_mod, main = "", xlab ="", ylab ="")
# modification pour positionner le titre
par(mfrow=c(1,1),oma=c(0,0,0,0))
# titre de la matrice CONCOR
mtext("Matrice CONCOR",line=2,font=2,cex=1.2)

# remarque : les deux manipulations par() sont necessaires uniquement pour la
matrice CONCOR
# si on le le fait pas, le titre deviens illisible

# insertion des sommets avec des couleurs différentes en fonction de leur
block
plot.igraph(g, vertex.color = node_colors, main = "Graphes des sommets et de
leurs appartenances aux blocks (chaque couleur correspond a un block)",
vertex.size = 12, edge.arrow.size = 0.5)

# export du graphe gephi
library("rgexf")
V(g)$id <- paste("X", 1:length(advice[,1]), sep="")
nodes_df <- data.frame(ID = c(1:vcount(g)), NAME = V(g)$id)
edges_df <- data.frame(igraph::get.edges(g, c(1:ecount(g))))

nodes_col <- node_colors
```

```
nodes_col_df <- as.data.frame(t(col2rgb(nodes_col, alpha = FALSE)))
nodes_col_df <- cbind(nodes_col_df, alpha = rep(1, times =
nrow(nodes_col_df)))
nodes_att_viz <- list(color = nodes_col_df)

# export du graphe dans un fichier .gexf
write.gexf(nodes = nodes_df, nodesVizAtt = nodes_att_viz, edges = edges_df,
defaultedgetype = "directed", output =
"tous_les_noeuds_appartenant_au_meme_block_ont_la_meme_couleur.gexf")

# insertion du graphe des blocks model
plot(gBlock, main = "Graphe du block model")
dev.off()


# export du graphe gephi

V(gBlock)$id <- paste("Block ", 1:length(matrice[,1]), sep="")
nodes_df <- data.frame(ID = c(1:vcount(gBlock)), NAME = V(gBlock)$id)
edges_df <- data.frame(igraph::get.edges(gBlock, c(1:ecount(gBlock))))

nodes_col <- colors
nodes_col_df <- as.data.frame(t(col2rgb(nodes_col, alpha = FALSE)))
nodes_col_df <- cbind(nodes_col_df, alpha = rep(1, times =
nrow(nodes_col_df)))
nodes_att_viz <- list(color = nodes_col_df)

# export du graphe dans un fichier .gexf
write.gexf(nodes = nodes_df, nodesVizAtt = nodes_att_viz, edges = edges_df,
defaultedgetype = "directed", output =
"pattern_des_relations_entre_blocks.gexf")

# pour limiter les conflits, on détache les bibliothèques utilisées dans cette
partie
detach(package:statnet, unload=TRUE)
detach(package:rgexf, unload=TRUE)
detach(package:igraph, unload=TRUE)
detach(package:concoR, unload=TRUE)

#======================================== VISUALISATION DE LA TABLE DE
DENSITE ========================================

# chargement de igraph
library(igraph)

#
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++
```

```
# on récupère les données enregistrées dans un tableur excel (qui sera choisi
par l'intérmédiaire d'une boîte de dialogue)
# il faut choisir la table de densité correspondante à la matrice choisie plus
haut
fichier <- tk_choose.files(caption = "Choisissez la table de densite a
analyser")
advice <- read.csv2(fichier, header=TRUE, row.names=1)

# extraction du nom du fichier d'où proviennent les données selectionnées
# afin de specifier dans le nom du fichier d'output quelles données d'input
ont été utilisées

noms <- strsplit(fichier, "[\\]")
fic <- noms[[1]][length(noms[[1]])]
fic <- strsplit(fic, "[/]")
fic <- fic[[1]][length(fic[[1]])]
fic <- strsplit(fic, "[.]")
fic <- fic[[1]][1]
# on convertit la liste des données obtenues en matrice d'incidence (qui met
en relation deux classes d'objets)
data_matrix <- as.matrix(advice)

print("Veuillez ecrire un seuil pour la visualisation de votre table de
densite, p.ex. la densite generale du reseau")
# on lit le seuil choisi par l'utilisateur
seuil <- scan(nmax=1, what=double())

# Création de la matrice d'adjacence de la taille de la table de densité
sélectionnée, remplie de zeros
imgMat <- matrix(0, nrow=length(data_matrix[1,])-1,
ncol=length(data_matrix[,1])-1)

# parcours de la table de densité
for(i in 1:(length(data_matrix[1,])-1)){
    for(j in 1:(length(data_matrix[,1])-1)){
        # Si la donnée en i,j est supérieure au seuil, alors on met un 1 dans
la matrice d'adjacence
        if(data_matrix[i,j] >= seuil){
            imgMat[i,j]<- 1
        }
    }
}

# écriture du fichier csv stockant la matrice image obtenue pour permettre son
utilisation dans un autre script
# rappel : fic est le nom du fichier d'input contenant la table de densité
write.csv2(imgMat, file =
paste("output_Matrice_Image_De_La_Table_De_Densite_",fic,".csv", sep =""))
```

```r
# on detache igraph pour limiter les conflits avec un un autre futur script
detach(package:igraph, unload=TRUE)


# ======================================== FIN
=====================================
```