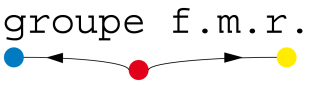
Analyser les réseaux avec R (packages statnet, igraph et tnet)

Laurent Beauguitte

Analyser les réseaux avec R (*packages* statnet, igraph et tnet) version 1

Laurent Beauguitte
(UMR IDEES)
beauguittelaurent<at>hotmail.com

Avril 2012



Introduction

Ceci n'est pas un manuel d'initiation à R, de nombreux tutoriaux (en anglais et en français) sont disponibles sur le net [1]. En ce qui concerne l'analyse des réseaux (techniques, sociaux et/ou complexes), je me permets de renvoyer aux précédentes synthèses du groupe fmr [2], ainsi qu'aux principaux manuels existants [5][4].

Les scripts à taper dans R et les résultats obtenus sont en `typewriter`. Tous les scripts sont utilisables directement (une fois les *packages* installés évidemment). La lecture de Barnier 2011 [1] est conseillée pour compléter les indications relatives à `statnet`.

Si vous repérez des erreurs ou des approximations, n'hésitez surtout pas à m'en faire part [3]. Enfin, comme l'ont sans doute déjà deviné les utilisateurs de R, ce livret ne prétend pas épuiser les possibilités du logiciel...

Deux fonctions indispensables

Personne, jamais, n'a maîtrisé, ne maîtrise ou ne maîtrisera l'ensemble du logiciel R. Aussi, les deux fonctions suivantes sont indispensables à tout utilisateur.

1. Voir par exemple l'excellent tutoriel de Julien Barnier disponible sur Quanti/sciences sociales, *R pour les sociologues*, http ://quanti.hypotheses.org/217/.
2. http ://halshs.archives-ouvertes.fr/FMR/
3. Merci à Pierre Beauguitte pour sa relecture attentive.

Pour demander de l'aide sur une fonction : `help(x)` où `x` est le nom de la fonction. `help(x)` peut être remplacé par `?x` ou encore `??x`.

Pour connaître toutes les fonctions et les jeux de données d'un *package* : `library(help=x)` où `x` est le nom du *package*.

Enfin, pour mettre à jour un *package* : dans la console, Packages > Mettre à jour les packages.

Sous Windows, l'installation d'un *package* entraîne automatiquement l'installation des *packages* associés. Sous Mac, il est nécessaire de cocher la case « Installer les dépendances » lors de l'installation du *package*. Il est enfin possible, quel que soit le système d'exploitation, d'utiliser la commande `install.packages("nom du package", dependencies = TRUE)`.

La mise en forme des données

La forme la plus commune est la liste de liens, valués ou non, que ce soit pour les graphes simples ou bipartis. Comme d'habitude avec R, évitez autant que possible les formats .xls (ou .xlsx) qui favorisent messages d'erreur et détériorations de données. Enregistrez plutôt vos données tableur en format .txt ou .csv. Tous les *packages* présentés ici sont également capables de lire les formats de graphe les plus courants (Ucinet, Pajek).

L'importation des données est l'étape la plus pénible à franchir. Une fois que le *package* utilisé reconnaît vos données comme un réseau, la suite des opérations est extrêmement aisée. Ne vous découragez pas, toute personne débutant avec un nouveau *package* dans R passe quelques heures à parcourir les forums... mais le jeu en vaut la chandelle.

Si les codes des sommets sont numériques (ex. code INSEE), il est nécessaire de les transformer en caractères sinon `statnet` et `igraph` considèrent que le dernier code numérique correspond au nombre de sommets. Avant la transformation en graphe, il conviendra donc de sélectionner les colonnes contenant les codes et de leur appliquer la fonction `as.character()`.

1 Analyse de graphe avec statnet

Attention, évitez de lancer en même temps `statnet` et `igraph` : risque de conflit entre certaines fonctions (ex `gplot`).

1.1 Import et transformation en graphe

Soit un fichier "g1.csv" contenant la matrice suivante :

```
CODE;A;B;C;D
A;0;1;1;0
B;1;0;0;0
C;0;1;0;0
```

```
D;0;1;0;0
```

Pensez à insérer un saut de ligne à la fin de la dernière ligne. Le script suivant permet de les importer dans `statnet`, à condition évidemment de préciser où se trouve le fichier en question.

```
library(statnet)
setwd(emplacement du fichier)
g<-read.table(file="g1.csv", header=TRUE,
              row.names=1, sep=";")
g<-as.network(as.matrix(g))
g
```

`header=TRUE` signifie que les colonnes ont un nom ; `row.names=1` que la première colonne contient les noms des lignes. La commande `as.matrix` permet de transformer le tableau en matrice, la commande `as.network` de la transformer en objet matrice analysable avec `statnet`.

Soit un fichier "g2.csv" contenant :

```
A;B
A;C
B;A
C;B
D;B
```

Pensez à insérer un saut de ligne à la fin de la dernière ligne. Pour l'importer et le transformer :

```
library(statnet)
g<-read.table(file="g2.csv", header=FALSE, sep=";")
g<-as.network(g)
```

Après toute importation, taper `g` dans la console permet de vérifier que l'importation s'est bien déroulée et donne des indications de base sur la matrice (nombre d'acteurs et de liens, liens orientés ou non, présence de boucles, de liens multiples...). Si les données sont valuées, il est recommandé de seuiller sur le tableau de départ, avant la transformation en graphe, `statnet` étant conçu pour les graphes booléens.

1.2 Mesures, cliques et visualisation

Le script suivant liste une grande partie des mesures disponibles mais ne prétend nullement à l'exhaustivité. Il est possible de le copier et de le coller directement dans R après avoir créé un nouveau script.

```
#initialisation
rm(list=ls())

#chargement du package
library(statnet)

#création d'un graphe aléatoire de 20 sommets, densité 0.25

g <- rgraph(20, tprob = 0.25)

#densité - résultat prévisible ici...

gden(g)

#centralités

din<-degree(g, cmode="indegree")
dout<-degree(g, cmode="outdegree")
table(din)
table(dout)

#pour un graphe non orienté

table(degree(g))

#afficher les résultats triés

x<-sort(degree(g, cmode="freeman"))
dotchart(x, main="Degree")

#proximité
closeness(g)

#intermédiarité
betweenness(g)

#vecteurs propres
evcent(g)

#réciprocité et transitivité

grecip(g)
gtrans(g)
```

```
#dyades et triades

dyad.census(g)
triad.census(g)

#distances géodésiques et voisins d'ordre 1, 2...

geodist(g)
g1<-neighborhood(g,1)
plot.sociomatrix(g1)
g2<-neighborhood(g,2)
plot.sociomatrix(g2)

#points d'articulation

cutpoints(g)

#groupes

kcores(g)

#blockmodels
#k = nombre de classes désirées

eq<-equiv.clust(g)

#matrices bloquées

bm3<-blockmodel(g, eq, k=3)
bm3$block.model
bm4<-blockmodel(g, eq, k=4)
bm4$block.model

#extraction d'un égo network

eg<-ego.extract(g1, ego=7)
gplot(eg)

#visualisation

#création d'un graphe de 20 sommets
#partition en 3 blocs
#taille des sommets = in-degree
#couleurs des sommets = blockmodels
```

```
#effacer objets créés précédemment
rm(list=ls())

g <- rgraph(20, tprob = 0.25)
eq<-equiv.clust(g)
bm <- blockmodel(g, eq, k=3)
groupes <- bm$block.membership[order(bm$order.vector)]

#couleur sommet = blockmodels
#taille = in-degree

gplot(g, displayisolates=FALSE,
      vertex.cex=degree(g, cmode="indegree")/4,
  vertex.col=groupes,
  mode="fruchtermanreingold")

#visualisation 3D

gplot3d(g)

#pour connaître les options

?gplot
```

Dix-sept options différentes sont possibles pour le positionnement des acteurs (mds, circle, kamadakawai ...), pour connaître les détails de chacune, consulter l'aide de la fonction `gplot.layout`. Il est possible d'obtenir des liens courbes avec l'option `gplot(g, usecurve=TRUE)` (le résultat est parfois peu satisfaisant). Pour pouvoir repositionner les nœuds et augmenter la lisibilité du graphe, inclure dans la parenthèse gplot l'option `interactive=T`. La commande `plot.sociomatrix(g)` permet de représenter la matrice d'adjacence du graphe. Enfin, il est possible d'obtenir des graphes en 3D (avec possibilité de les faire pivoter autour d'un acteur, de zoomer et dézoomer à l'intérieur avec la fonction `gplot3d(g)`.

Les deux exemples ci-dessous donnent une idée des résultats possibles.

```
library(statnet)

gplot3d(rgws(1,5,3,1,0.2))
```

1.3 Modèles statistiques

Les modèles statistiques permettant de comparer, de caractériser ou d'émettre des hypothèses sur des graphes sont nombreux, complexes et en continuelle évolution. Il est conseillé une fois encore de commencer par le Faust et Wasserman[5], puis, une fois les bases maîtrisées, de se plonger dans le Scott, Carrington et Wasserman[3].

L'outil le plus basique pour étudier un graphe issu de travaux empiriques consiste à mesurer un indicateur sur ce graphe, puis à générer quelques centaines de graphes aléatoires de même ordre et de même taille, et enfin de comparer les résultats [4].

Avec R, il conviendra de calculer un indicateur, par exemple le niveau de transitivité dans un graphe orienté (est-ce que chaque fois que x est lié y et à z, y est lié z ?) à l'aide la fonction `gtrans(g)`. Le jeu de données utilisé est `flo` dont l'indice de transitivité est de 0,19 (dans 19% des cas, quand une famille est liée à deux autres, ces deux dernières sont également liées entre elles). Puis les trois lignes suivantes permettent de générer 1000 graphes aléatoires possédant le même nombre d'acteurs (16) et la même densité (0,167), et de représenter la distribution de l'indicateur.

```
rm(list=ls())
library(statnet)
data(flo)
gtrans(flo)
gg<-rgraph(16, m=1000, tprob=gden(flo))
tr<-apply(gg,1,gtrans)
hist(tr, main="Transitivité",ylab="Fréquence")
```

Si vous testez le script, vous constaterez que le niveau de transitivité du réseau `flo` est tout à fait banal et moyen.

L'outil le plus utilisé actuellement pour l'étude statistique des réseaux est sans doute le modèle ERGM (*Exponential Random Graph Model*), modèle dérivé du modèle p^* [5]. Le package `statnet` est tout spécialement conçu pour mener à bien ce type de modélisation.

En deux mots, le principe est de mesurer les propriétés (essentiellement dyadiques et triadiques) du graphe et de vérifier si elles s'écartent significativement des propriétés d'un graphe aléatoire de même ordre et de même taille. Les résultats se lisent comme ceux d'une régression logistique. La densité du graphe observé est systématiquement incluse dans le modèle comme variable de contrôle. Les variables explicatives les plus souvent incorporées

4. Cet exemple est emprunté au tutoriel de Julien Barnier, *op. cit.*

5. Une synthèse fmr est en cours de préparation sur le sujet, elle devrait être disponible en avril 2012.

dans les modèles sont la popularité (*in-degree*), l'activité (*out-degree*), la réciprocité des relations et la transitivité. Il est possible d'inclure des données attributaires relatives aux acteurs et/ou aux liens dans le modèle.

2 Analyse de graphe avec igraph

Attention, évitez de lancer en même temps `igraph` et `statnet` : risque de conflit pour certaines fonctions.

Une fois origines et destinations importées, il convient de les transformer en matrice (`as.matrix`) puis d'utiliser la fonction `graph.edgelist` afin de les transformer en graphe. Il convient de préciser si le graphe est orienté, si les boucles sont autorisées etc.

Soit un fichier "g.csv" contenant deux colonnes, l'une origine, l'autre destination.

```
rm(list=ls())
library(igraph)

g<-read.csv("g.csv", sep=";", header=TRUE)
g<-as.matrix(g)
g<-graph.edgelist(g, directed=TRUE)

#contrôle

class(g)
```

2.1 Mesures, cliques

La liste des mesures disponibles est légèrement plus complète que pour `statnet` : `igraph` permet ainsi le calcul du *clustering coefficient* ou de l'intermédiarité des liens (et non des seuls sommets comme `statnet`). Si le point fort de `statnet` est la modélisation statistique, celui d'`igraph` est sa capacité à générer les modèles de graphes devenus classiques (aléatoires, *small-world* et *scale-free*).

Le script suivant peut être copié et exécuté dans R. Sans prétendre à l'exhaustivité, il donne un aperçu des possibilités du *package*.

```
#initialisation
rm(list=ls())

#chargement du package
library(igraph)
```

```
#création d'un graphe aléatoire orienté

g <- erdos.renyi.game(20, 1/20, directed=TRUE)

#densité et diamètre

graph.density(g)
diameter(g)

#centralités

#degré
#choix entre "in", "out" et "total"

table(degree(g, mode=c("in")))

#intermédiarité des sommets et des liens

betweenness(g)
edge.betweenness(g)
closeness(g)
evcent(g)

#distance et plus court chemin

average.path.length(g)
shortest.paths(g)

#réciprocité

reciprocity(g, ignore.loops=TRUE)

#transitivité locale ou globale

transitivity(g, type=c("local"), vids=NULL)
transitivity(g)

#dyades et triades

dyad.census(g)
triad.census(g)

#points d'articulation
```

```
articulation.points(g)

#recherche de cliques
#à éviter sur les graphes de grande taille

cliques(g)
largest.cliques(g)
maximal.cliques(g)
clique.number(g)

#k-cores

graph.coreness(g)

#suppression boucles et isolés

g1 <- delete.vertices(g, V(g)[degree(g)==0])
g2 <- simplify(g1, remove.loops = TRUE)

#composantes "weak" ou "strong"

is.connected(g, mode=c("strong"))
clusters(g, mode=c("strong"))

#décompose en composantes connexes

compco <- decompose.graph(g, mode = c("weak"),
max.comps = NA, min.vertices = 3)
plot(compco[[1]])
plot(compco[[2]])

#arbre couvrant minimum

plot(minimum.spanning.tree(g))

#recherche des voisins d'ordre n

neighborhood.size(g, 1, nodes=V(g), mode=c("all", "out", "in"))
neighborhood.size(g, 2, nodes=V(g), mode=c("all", "out", "in"))

#3 modes de visualisation

plot(g)   #non interactif
tkplot(g) #interactif
```

```
rglplot(g)  #3D non interactif

#choix des algorithmes

help(package="igraph", "layout")

#graphe circulaire

g <- graph.ring(10)
g$layout <- layout.circle
plot(g)

#graphe scale-free

g <- barabasi.game(100)
plot(g, layout=layout.fruchterman.reingold, vertex.size=4,
     vertex.label.dist=0.5, vertex.color="red",
     edge.arrow.size=0.5)

#graphe small-world

g <-watts.strogatz.game(1, 100, 5, 0.05)
plot(g, layout=layout.kamada.kawai, vertex.size=4,
     vertex.label.dist=0.5, vertex.color="green",
     edge.arrow.size=0.5)

#graphe aléatoire et une couleur par composante

g <- erdos.renyi.game(60, 1/20)
comps <- clusters(g)$membership
colbar <- rainbow(max(comps)+1)
V(g)$color <- colbar[comps+1]
plot(g, layout=layout.fruchterman.reingold,
            vertex.size=5, vertex.label=NA)

#visualisation des communautés

g <- graph.full(5) %du% graph.full(5) %du% graph.full(5)
g <- add.edges(g, c(0,5, 0,10, 5,10))
com <- spinglass.community(g, spins=5)
V(g)$color <- com$membership+1
g <- set.graph.attribute(g, "layout", layout.kamada.kawai(g))
rglplot(g, vertex.label.dist=1.5)
```

```
#visualisation d'arbres

igraph.par("plot.layout", layout.reingold.tilford)
plot(graph.tree(20, 2))
plot(graph.tree(50, 3), vertex.size=3, vertex.label=NA)
tkplot(graph.tree(50, 2, mode="undirected"),
       vertex.size=10, vertex.color="green")
```

3 Analyse de graphe avec tnet

3.1 Graphe valué

Le format demandé par `tnet` est des plus classiques : une liste de liens, éventuellement valués[6], soit au maximum trois colonnes origine - destination - poids. La fonction `as.tnet()` ne crée pas un objet différent (il s'agit d'un `data.frame`standard) mais permet de vérifier que les données sont correctement structurées. Attention, les codes sont nécessairement numériques, croissants et commençant par 1...[7] Une partie des mesures proposées reste exploratoire et s'appuie essentiellement sur des articles très récents de l'auteur du *package*.

En deux mots, le principe est de pondérer la prise en compte du poids des liens en faisant varier un paramètre α. Si celui-ci est égal à zéro, le poids des liens n'est pas du tout pris en compte, plus il s'approche de 1, plus le poids est pris en compte. En fonction de l'étendue de la valeur des liens, il conviendra donc de prendre un α plus ou moins grand.

```
rm(list=ls())
library(igraph)
library(tnet)

sample <- rbind(
c(1,2,4),
c(1,3,2),
c(2,1,4),
c(2,3,4),
c(2,4,1),
c(2,5,2),
c(3,1,2),
c(3,2,4),
```

6. Si le graphe de départ n'est pas valué, il est préférable d'utiliser l'un des *packages* vus précédemment.

7. C'est ce que divers tests semblent indiquer mais je ne certifie pas ce dernier point.

```
c(4,2,1),
c(5,2,2),
c(5,6,1),
c(6,5,1))

#"transformation" en objet tnet
#supprime boucle par défaut

g <- as.tnet(g)

#degré pondéré
#out degree

degree_w(net=g, measure=c("degree","output","alpha"), alpha=0.5)

#in-degree

degree_w(net=g, measure=c("degree","output","alpha"),
alpha=0.5, type="in")

#closeness - booléen

closeness_w(net=g, alpha=0)

#centralité de proximité valuée, alpha = 1

closeness_w(net=g, alpha=1)

#centralité de proximité valuée, alpha= 0.5

closeness_w(net=g, alpha=0.5)

#intermédiarité booléenne

betweenness_w(g, alpha=0)

#intermédiarité valuée

betweenness_w(g, alpha=1)

#intermédiarité valuée

betweenness_w(g, alpha=0.5)
```

```
#clustering coefficient global
#bi: booléen; am: moyenne arithmétique; gm: moyenne géométrique
#ma: maximum method; mi: minimum method

clustering_w(g, measure=c("bi", "am", "gm", "ma", "mi"))

#clustering coefficient local
#seulement pour non orienté

clustering_local_w(g, measure=c("bi", "am", "gm", "ma", "mi"))

#rich-club coefficient basé sur le degré
#parameter and link-reshuffled random networks
#sans intérêt ici étant donnée l'ordre du graphe

out <- weighted_richclub_w(g, rich="k", reshuffle="links",
                           NR=1000, seed=1)

#visualisation

plot(out[,c("x","y")], type="b", log="x", xlim=c(1, max(out[,"x"]+1)),
     ylim=c(0,max(out[,"y"])+0.5), xaxs = "i", yaxs = "i",
     ylab="Weighted Rich-club Effect", xlab="Prominence
     (degree greater than)")
lines(out[,"x"], rep(1, nrow(out)))

#distance moyenne

bg <- dichotomise_w(g)

#distance moyenne dans le graphe booléen

mean(distance_w(bg),na.rm=T)

#distance moyenne dans le graphe valué

mean(distance_w(g),na.rm=T)
```

3.2 Graphe biparti valué

La structure des données est la même que pour un graphe simple valué : origine, destination et poids. Les sommets des deux sous-ensembles doivent être numérotés de 1 à n. Les principales fonctions proposées sont les suivantes : `distance_tm`, `clustering_tm` et `clustering_local_tm`.

```
g2m<- rbind(
c(1,1,3),
c(2,1,2),
c(1,2,2),
c(3,2,1),
c(3,3,2),
c(4,1,2),
c(4,3,1))

g2 <- as.tnet(g2m, type="weighted two-mode tnet")

distance_tm(g2)
clustering_tm(g2)
clustering_local_tm(g2)
```

Rappelons que transformer un graphe biparti en graphe de coappartenance sur les lignes ou les colonnes ne nécessite aucun *package* particulier dans la mesure où il s'agit d'opérations matricielles élémentaires (multiplication de la matrice par sa transposée ou de la transposée par la matrice de départ, voir [2]).

Conclusion

R étant libre et gratuit, la communauté des utilisateurs étant extrêmement active, bien d'autres développements sont à prévoir. D'autres modules existent pour analyser les réseaux (voir le portail CRAN) et la liste qui suit ne prétend pas être exhaustive :

- *package* `blockmodeling` : serait sans doute très utile, la documentation disponible à ce jour est malheureusement proche de l'indigence ;
- *package* `bipartite` pour l'analyse des graphes... bipartis [8] ;
- *package* `Rsiena` consacré à la modélisation des dynamiques d'un graphe, développé par R.M. Ripley et T.A.B. Snijders. Page personnelle très riche (articles, jeux de données, exemples), documentation complète (le manuel officiel compte 82 pages) et ardue (les longs scripts supposent de solides connaissances concernant le modèle `Siena`).

La situation aujourd'hui est la même que pour les logiciels presse-boutons : l'outil polyvalent, multi-tâches, offrant une large palette d'indicateurs et des possibilités graphiques soignées... n'existe pas. Chacun des trois modules présentés ici a ses avantages et ses inconvénients et, en fonction de ses besoins, il conviendra d'utiliser tantôt l'un, tantôt l'autre.

8. Non encore testé par l'auteur de ces lignes.

Le temps d'apprentissage peut sembler long. Mais un programme correctement écrit pourra servir x fois sur x matrices différentes.

Ressources internet

Le site du projet `statnet` [9] permet de s'abonner à une liste de diffusion très utile pour mener les modèles ERGM, fournit les liens vers un numéro spécial du *Journal of Statistical Software* paru en 2008 [10] ainsi que des articles et des tutoriels. Des ressources équivalentes sont disponibles sur le site du projet `igraph` [11].

Les pages personnelles de Tore Opshal [12] méritent le détour, que l'on souhaite ou non utiliser `tnet`. On y trouve notamment tous ses articles consacrés aux mesures adaptées aux réseaux valués et/ou bipartis. Consulter les pages personnelles de Snijders [13] est à peu près indispensable pour pouvoir utiliser le module Rsiena.

Un outil très pratique en ligne : le site du docteur en biologie Daizaburo (Dai) Shizuka. Une page permet ainsi de savoir si tel ou tel indicateur est disponible dans `statnet` et/ou `igraph` et la commande correspondante est fournie [14]. L'ensemble des pages R Resources est consacré à ces deux *packages* [15].

Références

[1] J. Barnier : Analyse de réseau avec R. 2011. http ://alea.fr.eu.org/git/doc_reseaux_r.git/blob_plain/HEAD :/networks.pdf.

[2] L. Beauguitte et P. Beauguitte : Opérations matricielles et analyse de graphe. *Groupe fmr*, 2011. http ://halshs.archives-ouvertes.fr/FMR/fr/.

[3] P.J. Carrington, J. Scott et S. Wasserman : *Models and methods in social network analysis*. Cambridge University Press, 2005.

[4] E. Lazega : *Réseaux sociaux et structures relationnelles*. Que sais-je ? Presses Universitaires de France, 11 édition, 2007.

9. http ://www.statnetproject.org/
10. Tous les articles sont accessibles gratuitement à l'adresse http ://www.jstatsoft.org/v24/.
11. http ://igraph.sourceforge.net/
12. http ://toreopsahl.com/
13. http ://www.stats.ox.ac.uk/s̃nijders/
14. http ://sites.google.com/site/daishizuka/toolkits/node-level-calculations
15. http ://sites.google.com/site/daishizuka/toolkits

[5] S. Wasserman et K. Faust : *Social Network Analysis. Methods and Applications*. Structural analysis in the social sciences. Cambridge University Press, 1994.

Table des matières