INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Initiation à R

Romain François — Jean-Michel Marin

N° 0332

Mars 2007

Thème COG

Rapport technique

ISSN 0249-0803 ISRN INRIA/RT--0332--FR+ENG

Initiation à R

Romain François * , Jean-Michel Marin†

Thème COG — Systèmes cognitifs
Projets SELECT

Rapport technique n° 0332 — Mars 2007 — 21 pages

Résumé : Ce document a pour objectif de familiariser son lecteur avec le langage et l'environnement de programmation R dans sa version 2.4.1 (décembre 2006). Il ne constitue pas une référence complète mais plutôt un aperçu des capacités de R et un point d'entrée vers d'autres documents plus complets. À plusieurs reprises, le lecteur est invité à exécuter les commandes proposées dans une session R afin de s'habituer à la syntaxe.

Mots-clés : Traitement de données, logiciel libre R

* INRIA Futurs, Projet SELECT, Université Paris-Sud
† INRIA Futurs, Projet SELECT, Université Paris-Sud

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 — Télécopie : +33 1 60 19 66 08

Introduction to R

Abstract: The aim of this paper is to introduce the reader to the R programming language, version 2.4.1 (December 2006). It does not represent a detailed reference but an overview on the potentialities of R.

Key-words: Data analysis, R, Open Source

1 Introduction

R est un langage de programmation interactif interprété et orienté objet contenant une très large collection de méthodes statistiques et des facilités graphiques importantes. R une implémentation libre (au sens GNU/GPL) du langage S (Carlier, 1991; Fayet, 1991; Venables and Ripley, 2002), conçu par les laboratoires Bell puis racheté par la société Insightful en vue d'une commercialisation sous le nom de S-Plus. R est né en 1995 des travaux de Robert Gentleman et Ross Ihaka (Département de Statistique, Université d'Auckland, Nouvelle-Zélande) pour être ensuite adopté par une communauté d'utilisateurs et de développeurs de plus en plus importante. Aujourd'hui, la *R Development Core Team* est composée d'une vingtaine de personnes, principalement des chercheurs, qui contribuent à l'amélioration et la documentation des puissantes fonctionnalités de R. En quelques années, grâce notamment à son modèle de diffusion basé sur la liberté, R est devenu un laboratoire incontournable des nouvelles idées du paysage statistique mondial. L'une des forces de R est son extensibilité, sous forme de packages réalisés par les utilisateurs à des fins bien précises. Le CRAN (Comprehensive R Archive Network, `http://cran.r-project.org/`) recense aujourd'hui plus de 700 de ces packages, couvrant ainsi largement le spectre de la statistique actuelle.

La documentation sur R est très importante, parfois un peu éparpillée, et souvent difficile à appréhender au premier essai. Différents points d'entrée sont recensés sur site internet central de R : `http://www.r-project.org/`, on peut notamment citer R-Development-Core-Team (2006a,c,b). Nous conseillons la lecture de Broutaux (2002), Dalgaard (2002), Venables and Ripley (2002), Verzani (2005) qui constituent d'excellentes références sur R (ou S au sens large). Les ouvrages de Dalgaard (2002) et Verzani (2005) donnent de nombreux exemples de traitement de données à l'aide de R. Concernant l'impressionnante interface graphique qu'est R, le lecteur peut se référer à Maindonald and Braun (2003). Notons que R est depuis peu doté d'un système collaboratif de documentation, permettant à tout un chacun de partager ses ruses avec le reste de la communauté : `http://wiki.r-project.org`, et d'une galerie de graphiques : `http://addictedtor.free.fr/graphiques`.

Le but de cet article est de donner un bref aperçu des capacités de R en terme de programmation, de traitement des données, et de réalisation de graphiques. Comme pour tout langage évolué, la maîtrise de R s'acquiert par la pratique et la réalisation de vraies études et projets.

2 Comment utiliser R

À l'exécution de R, les fichiers `.RData` et `.Rhistory` sont automatiquement créés dans le répertoire courant. Aussi, lorsque l'on quitte R à l'aide de la commande `q()`, les nouveaux objets créés peuvent être automatiquement sauvegardés dans le fichier `.RData` (fichier binaire) ainsi que la suite des commandes saisies dans le fichier `.Rhistory` (fichier texte). Ainsi, lorsque l'on relance R dans le même répertoire, le fichier `.RData` est automatiquement chargé et l'on retrouve l'intégralité des objets que l'on y a précédemment créés. Par ailleurs,

on peut visualiser la suite de commandes que l'on a tapées à l'aide de l'instruction `history()` dans R ou en consultant simplement le fichier `.Rhistory`. Enfin, la commande `ls()` permet de visualiser la liste des objets créés et la commande `rm()` permet de détruire des objets. Il y a plusieurs façons d'utiliser R, selon les habitudes et les préférences de chacun :

- en mode console, en tapant R à l'invite de commande ;
- en tâche de fond (BATCH job) ;
- avec un interface utilisateur graphique (GUI) : Rgui.exe, Sciviews sous Windows, Raqua sous Mac/OS, gnomeGUI ou RKWard sous GNU/Linux, Tcl/Tk, JGR, PMG, R Commander... ;
- via un editeur de texte : Emacs+ESS, Tinn-R, Texmacs, Eclipse, WinEdt... ;
- depuis d'autres langages de programmation : Perl, Python, Java, C... ;
- en mode serveur TCP/IP : Rserve avec clients Java ou C++ ;
- au sein d'un document LaTeX, HTML ou ODF grâce à Sweave ;
- par internet : Rpad, R Zope, R-PHP, RApache ;
- depuis Excel : R-(D) COM...

Le mode console ou l'utilisation de R depuis un autre langage nécessite une bonne connaissance des fonctions R. Le débutant ou l'utilisateur occasionnel de R sera sans aucun doute plus confiant dans l'utilisation de R via une interface graphique. Celles de Windows et MAC/OS enrobent certaines fonctionnalités utiles comme par exemple la gestion des fichiers et des packages. Nous conseillons l'utilisation JGR (Java Gui for R, `http://rosuda.org/JGR/`) qui offre le double avantage d'être multi-plateforme et très avancé. Ce système est doté d'un éditeur de texte spécialisé qui propose au fur et à mesure de la saisie du code les arguments des principales fonctions sous forme de conseils discrets. JGR dispose également d'un système évolué de gestion des packages et des fichiers d'aide de R.

3 Les objets

Les éléments de base de R sont des objets qui peuvent être des données (vecteurs, matrices, séries chronologiques...), des fonctions, des graphiques... Les objets R se différencient par leur mode, qui décrit leur contenu et leur classe. Les objets atomiques sont de mode homogène et les objets récursifs sont de mode hétérogène. Les différents modes sont :

```
null (objet vide), logical, numeric, complex, character
```

Les principales classes d'objets sont :

```
vector, matrix, array, factor, time-series, data.frame, list
```

Les vecteurs, matrices, tableaux, variables catégorielles et séries chronologiques sont de mode homogène. Par contre, les listes ou les tableaux de données peuvent être de mode hétérogènes.

Les vecteurs Il s'agit de l'objet de base dans R. Un vecteur est une entité unique formée d'une collection ordonnée d'éléments de même nature. Les vecteurs peuvent être constitués d'éléments numériques, logiques ou alphanumériques. Pour créer un vecteur, on utilise la fonction `c(el1,el2, ...)`. Les nombres décimaux sont encodés avec un point décimal, les chaînes de caractères entourées par des guillemets doubles `"une chaine"` ou simple `'une autre chaine'`, et les valeurs logiques sont codées par les chaînes de caractères `TRUE` et `FALSE` (il est fortement déconseillé d'utiliser les abréviations `T` et `F`). Enfin, les données manquantes sont codées par la chaîne de caractères `NA`.

À partir de la console, vous allez maintenant taper les séquences du tableau 1 au clavier (excepté le premier caractère `>`, invite de R signalant que la console est en attente d'une instruction. Le tableau 2 présente quelques fonctions utiles.

Les matrices Les matrices, comme les vecteurs, sont de mode quelconque, mais elles ne peuvent pas contenir des éléments de nature différente. La syntaxe de création d'une matrice est la suivante : `matrix(vec,nrow=n,ncol=p,byrow=T)` où `vec` est le vecteur contenant les éléments de la matrice, qui seront rangés en colonne (sauf si l'option `byrow=TRUE` est choisie). Le tableau 3 présente quelques opérations de base sur les matrices. Notons que la fonction `apply` permet d'éviter l'utilisation de boucles R explicites.

Les matrices à plus de deux dimensions Les matrices à plus de deux dimensions sont de mode homogène. Elles sont créées à l'aide de la commande suivante : `array(vec, c(n,p,q...))` où `vec` est le vecteur contenant les éléments de la matrice qui seront rangés en colonne et l'argument `c(n,p,q...)` désigne les dimensions : `n` est le nombre de lignes, `p` le nombre de colonnes, `q` le nombre de matrices. Le tableau 4 illustre ce mécanisme.

Les facteurs Un facteur est un vecteur avec un attribut supplémentaire, les niveaux (`levels`). Cet objet permet de définir une variable catégorielle. Pour créer un facteur, on utilise la fonction `factor`, voir tableau 5. Un vecteur de chaînes de caractère est en général converti automatiquement en facteur lorsque le contexte l'impose. En revanche, ce n'est pas le cas pour un vecteur numérique. Il faut donc être prudent lorsque l'on utilise une variable catégorielle dont les différentes modalités sont numériques. Les variables catégorielles ordinales sont créées à l'aide de la fonction `ordered`.

Les listes Une liste est une collection ordonnée d'objets, non nécessairement de même mode. Les listes sont très utilisées comme sortie de fonctions. Elles permettent en effet de récupérer des résultats complexes sous forme d'un seul objet. Elles sont créées à l'aide de la commande suivante : `list(nom1=el1,nom2=el2,...)` (l'utilisation des noms est facultative). On peut accéder à chaque élément de la liste à l'aide de son index entre double crochets `[[...]]` ou par son nom précédé du signe `$`. Le tableau 6 détaille quelques opérations de base sur les listes.

Les structures de données Les tableaux de données (`data.frame`) constituent une classe particulière de listes consacrées au stockage des données. Les éléments de la liste sont les variables et les composantes de ces éléments sont les individus.

Pour créer un tableau de données, on peut regrouper des variables de même longueur à l'aide de la commande `data.frame(nom1=var1,nom2=var2,...)`. Aussi, il est possible de transformer une matrice en tableau de données en utilisant la commande `as.data.frame(mat)`. Enfin, on peut utiliser un fichier externe avec la fonction `read.table`. Le tableau 7 présente quelques opérations de base sur les data frames.

4 Les distributions usuelles

Le tableau 8 recense les distributions usuelles disponibles dans R. D'autres distributions plus spécifiques se trouvent dans des packages supplémentaires. Pour chacune de ces distributions, on dispose de quatre commandes préfixées par une des lettres `d, p, q, r` et suivies du nom de la distribution :

`dnomdist :` il s'agit de la fonction de densité pour une distribution de probabilité continue et de la fonction de probabilité ($\mathbb{P}(X = k)$) dans le cas d'une distribution discrète ;

`pnomdist :` il s'agit de la fonction de répartition ($\mathbb{P}(X \leq x)$) ;

`qnomdist :` il s'agit de la fonction des quantiles, à savoir la fonction qui renvoie le plus petit entier u tel que $F(u) \geq p$ où F est la fonction de répartition de la distribution considérée ;

`rnomdist :` génère des réalisations aléatoires indépendantes de la distribution `nomdist`.

Le tableau 10 présente quelques fonctions usuelles de statistique exploratoire : moyenne, variance, mediane...

5 Les principales fonctions génériques

Il s'agit de fonctions qui s'appliquent à tous les types d'objets, mais qui exécutent une commande spécifique en fonction de la classe de l'objet concerné. Les trois principales fonctions génériques sont :

`print` qui gère l'affichage écran de différents objets,

`plot` qui réalise des représentations graphiques,

`summary` qui renvoie un résumé sur le contenu d'un objet.

En pratique, les fonctions réellement exécutées sont différentes suivant la classe d'objets. Ainsi, en tapant `print(x)`, on fait appel à la fonction `print.ts` si `x` est une série chronologique, à la fonction `print.glm` si `x` est le résultat de la mise en oeuvre d'un modèle linéaire généralisé, à la fonction `print.aov` si `x` est le résultat de la mise en oeuvre d'une analyse de la variance... Si `x` n'a pas de classe ou si la méthode `print` n'est pas implémentée pour la classe de `x`, c'est la fonction `print.default` qui est utilisée. Ces comportements sont illustrés par le tableau 11. Il est à noter qu'il existe un autre mécanisme d'orientation objet dans R, les classes S4 qui sont à la fois plus puissantes et moins populaires. Elles ne sont pas décrites dans cet article.

6 Construction d'une nouvelle fonction

Pour définir une nouvelle fonction, on utilise l'expression suivante :

```
nom_fonction <- function(arg1,arg2,...)
{
# commentaire
instructions
}
```

Les accolades indiquent le début et la fin de la fonction. Le symbole dièse `#` permet d'ajouter des commentaires au code. Enfin, on peut remplacer `arg1` par `arg1=x` afin d'indiquer que par défaut le premier argument est fixé à `x`.

Les commandes de création des fonctions peuvent être saisies directement dans la console. On travaille alors ligne par ligne sans retour en arrière possible. Aussi, il est plutôt conseillé d'utiliser un éditeur de texte externe. On crée alors un fichier texte contenant la ou les fonctions produites, puis on lance la commande `source("nom_fichier")` pour charger le contenu du fichier dans R (voir section 8).

Lors de l'exécution d'une fonction, R renvoie par défaut le résultat de la dernière expression évaluée dans la fonction. Aussi lors de l'exécution, une copie des arguments est transmise à la fonction, laissant les originaux intacts. La fonction suivante appelée `Newton` met en oeuvre une version de la méthode de Newton pour le calcul de racines carrées :

```
Newton=function(y)
{
x=y/2
while (abs(x*x-y) > 1e-10) x=(x+y/x)/2
x
}
```

7 Quelques éléments de programmation

Nous traitons succinctement dans cette section des instructions de sélections et de répétitions. Il s'agit des commandes `if, while, for` illustrées dans le tableau 12. Les deux dernières commandes de ce tableau montrent que l'utilisation de boucles est très coûteuse en temps de calcul. Il est ainsi indispensable de limiter l'utilisation des boucles en les remplaçant par des outils vectoriels ou matriciels qui font appel à des boucles C considérablement plus rapides.

8 Les entrées et sorties

Lorsque l'on utilise R sur différentes plate-formes, il faut pouvoir stocker les objets créés dans des fichiers. Par ailleurs, il est bien souvent nécessaire de récupérer des données provenant de sources externes sous forme de fichiers.

Les sorties La fonction générique `save()` permet la sauvegarde de n'importe quelle liste d'objets en mémoire, sous un chemin quelconque, aussi bien en format binaire que texte. Pour des raisons de compatibilité avec le langage S, il existe aussi la fonction `dump()` qui permet d'exporter des objets vers un fichier texte. L'exportation de tableaux de données sous forme de fichiers textes standards peut être réalisée à l'aide de la fonction `write.table()`. La fonction `write.csv` permet d'exporter des tableaux de données vers des fichiers au format csv.
Enfin, la fonction `source()` permet quant à elle la lecture d'une série d'instructions sauvegardées dans un fichier texte. Elle est très utilisée en programmation.

Les entrées La commande élémentaire `scan()` permet la lecture des fichiers textes. Plusieurs commandes spécifiques ont été développées à partir de la fonction `scan()`. Par exemple, les fonctions `read.table()` et `read.csv()` automatisent la lecture des fichiers de données textes standards et csv et stockent leurs résultats dans des `data.frame`.

La librairie `foreign` permet le transfert vers R de données créées à l'aide d'autres logiciels statistiques, à savoir Minitab, S-Plus, SAS, SPSS et Stata. Par exemple, la fonction `read.spss` prend en charge les données enregistrées au moyen des commandes `save` et `export` de SPSS. Enfin, les fichiers créés à l'aide de la commande `save` peuvent être rechargés en mémoire grâce à la fonction `load()`.

9 R et grande masse de données

La version de base de R charge intégralement en mémoire vive les objets crées. Aussi, dans de nombreux cas, R effectue plusieurs copies d'une base de données lors de l'exécution de procédures statistiques sur celles-ci. Ceci peut entraîner une saturation du système dès que la taille des jeux de données dépasse une certaine fraction de l'espace mémoire disponible. Ainsi,

il est clair que la version de base de R n'est pas adaptée à la gestion de grandes quantités de données. Il existe une série de librairies R assurant l'interface entre l'environnement R et des Systèmes de Gestion des Bases de Données (SGBD) relationnels. Il est fortement conseillé de les utiliser lorsque l'on souhaite traiter des quantités importantes de données.

10 Graphiques

La réalisation de graphiques est une entreprise courante dans n'importe quelle étude statistique. Aussi, la qualité d'un logiciel de statistique dépend de ses aptitudes graphiques. R est suffisamment flexible et puissant pour permettre la réalisation de graphiques de très bonne qualité dans un nombre important de formats : PNG, BMP, JPG, SVG, PS, PDF... La *R Graph Gallery* `http://addictedtor.free.fr` présente plus d'une centaine de graphiques entièrement réalisés avec R ainsi que le code source correspondant.

Trois systèmes graphiques R est doté de trois systèmes graphiques concurrents parfois complémentaires : les graphiques de base, le système grid et le système lattice.

Les graphiques de base sont les plus utilisés. Il s'agit du système graphique le plus ancien de R, inspiré des graphiques déjà présents dans S. La principale fonction de ce système est la fonction `plot`, qui est une méthode générique[1]. On trouve alors les fonctions `plot.ts` pour représenter les séries temporelles, `plot.hclust` pour représenter les arbres de classification ascendante hiérarchique... Pour connaitre l'ensemble des classes pour lesquelles la méthode `plot` est prévue, il faut utiliser la commande `methods("plot")`. D'autres fonctions de haut niveau sont présentes au sein de ce système graphique : `hist`, `boxplot`, `pairs`, `image`, `contour`. Il existe aussi des fonctions de bas niveaux permettant d'ajouter des éléments à un graphique existant : `points`, `lines`, `axis`, `rug`, `text`, `segments`, `arrows`.

Le package `grid` est un nouveau système graphique, spécifique à R, qui permet un parfait contrôle de tous les aspects du graphique à réaliser. Ce système est basé sur le concept de boîtes (les viewports). Le système `grid` est très prometteur mais est encore assez peu utilisé. `grid` est constitué d'un ensemble de fonctions de bas niveau permettant de rajouter du contenu à la boîte (viewport) courante : `grid.rect`, `grid.lines`...

Le système `lattice` est en réalité basé sur `grid`, ce qui signifie que l'on peut combiner l'utilisation de `grid` et `lattice` afin de produire de nouveaux graphiques plus spécifiques ou plus compliqués. Un des avantages de `lattice` est la facilité avec laquelle ce système réalise des graphiques conditionnels. Par exemple, en utilisant les données `iris`, la commande suivante `xyplot(Sepal.Length  Petal.Length | Species, data=iris)` trace un nuage

[1] Comme nous l'avons vu précédemment, il s'agit donc d'une fonction qui s'adapte au type d'objet qu'elle reçoit.

de points de la variable `Sepal.Length` en ordonnée contre la variable `Petal.Length` en abscisse pour chaque modalité de la variable catégorielle `Species`.

La figure 1 présente un échantillon de graphiques réalisables avec R prélevés sur la *R Graph Gallery*. Le lecteur est incité à consulter le site internet pour d'autres exemples.

Extensions possibles Outre les systèmes graphiques présentés ci-dessus, il existe d'autres possibilités graphiques pour R :

- Le package `rgl`, basé sur OpenGL, permet la réalisation de puissants graphiques 3D que l'utilisateur peut ensuite zoomer, tourner à sa guise.
- Le package `rggobi` permet une communication entre R et GGobi. Les avantages de GGobi sont alors mis à disposition depuis R
- Le package `ggplot` implémente la grammaire des graphiques et propose des fonctionnalités intéressantes.
- Le package `iPlots` permet la réalisation de graphiques interactifs...

Références

Broutaux, Y. (2002). Introduction à l'environnement statistique R. http ://cran.r-project.org/other-docs.html.

Carlier, A. (1991). Une présentation des langages S et SPLUS. *La revue Modulad*, 6 :1–18.

Dalgaard, P. (2002). *Introductory Statistics with R*. Springer-Verlag, New York.

Fayet, G. (1991). Une histoire de S à l'INRA. *La revue Modulad*, 6 :19–26.

Maindonald, J. and Braun, J. (2003). *Data Analysis and Graphics Using R - An Example-Based Approach*. Cambridge University Press.

R-Development-Core-Team (2006a). An Introduction to R (version 2.4.1). http ://cran.r-project.org/manuals.html.

R-Development-Core-Team (2006b). The R Language Definition (version 2.4.1). http ://cran.r-project.org/manuals.html.

R-Development-Core-Team (2006c). The R Reference Index (version 2.4.1). http ://cran.r-project.org/manuals.html.

Venables, W. and Ripley, B. (2002). *Modern Applied Statistics with S*. Springer-Verlag, New York, fourth edition.

Verzani, J. (2005). *Using R for Introductory Statistics*. Chapman and Hall / CRC Press.

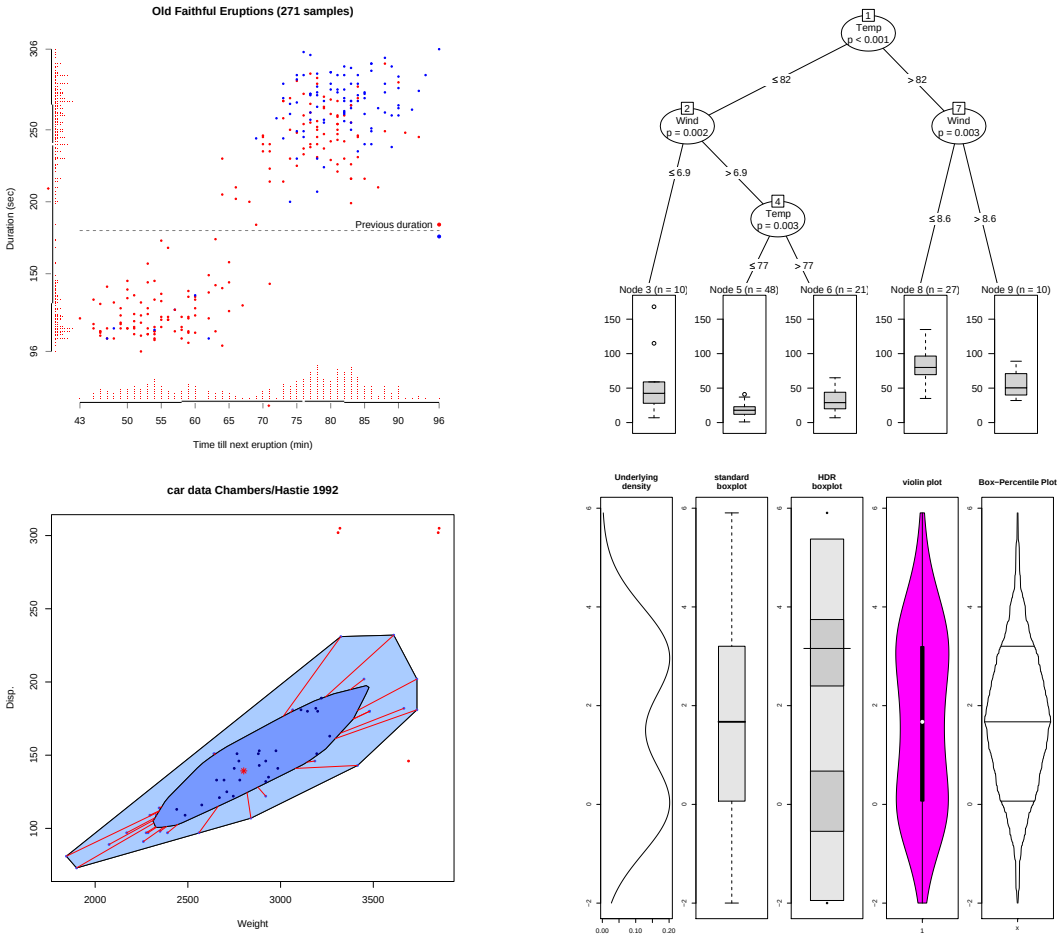

FIG. 1 – Quelques exemples de graphiques réalisés avec R. Ces graphiques sont issus de la *R Graph Gallery* : `http ://addictedtor.free.fr`. Les codes source sont disponibles sur le site web.
Nuage de points avec axes à la E. Tufte. Arbre de régression
Bagplot, généralisation en deux dimensions du boxplot Les cousins du boxplot

`> a <- c(5, 5.6, 1, -5)`	Création de l'objet `a` recevant un vecteur numérique de dimension 4 et de cordonnées 5, 5.6, 1 et -5
`> a`	Affichage du vecteur `a`
`> a[1]`	Affichage de la première coordonnée du vecteur `a`
`> a[-c(1,3)]`	`a` privé de ses coordonnées 1 et 3
`> b <- a[2:4]`	Création du vecteur numérique `b` de dimension 3 et de coordonnées 5.6,1,-5
`> a[2,4]`	Injures
`> d <- a[c(1,3,4)]`	Création du vecteur numérique `d` de dimension 3 et de coordonnées 5,1,-5
`> 2 * a`	Multiplication par deux de chacune des coordonnées du vecteur `a` et affichage du résultat
`> e <- 3 / d`	Création du vecteur numérique `e` de dimension 3 et de coordonnées 3/5, 3, -3/5
`> f <- a - 4`	Création du vecteur `f` de dimension 5 dont les coordonnées sont égales à celles de `a` moins 4
`> d <- d + e`	Remplacement du vecteur `d` par le vecteur résultant de la somme des vecteurs `d` et `e`
`> d <- d - e`	Remplacement du vecteur `d` par le vecteur résultant de la différence entre les vecteurs `d` et `e`
`> d * e`	Multiplication terme à terme des vecteurs `d` et `e`
`> e / d`	Division terme à terme entre les vecteurs `e` et `d`
`> sum(d)`	Calcul de la somme de `d`
`> length(d)`	Affichage de la dimension du vecteur `d`
`> t(d)`	Transposition du vecteur `d`, le résultat est un vecteur ligne
`> t(d) %*% e`	Produit scalaire entre le vecteur ligne `t(b)` et le vecteur colonne `e`
`> g <- c(sqrt(2), log(10))`	Création du vecteur numérique `g` de dimension 2 et de coordonnées $\sqrt{2}, \log(10)$
`> (1 + exp(2)) / cos(8)`	R est aussi une calculatrice
`> bool <- (d==5)`	Création du vecteur booléen `bool` de dimension 3 recevant `TRUE` si `d[i]=5` et `FALSE` sinon
`> text <- c("grand","petit")`	Création du vecteur caractère `text` de dimension 2 de coordonnées `grand`,`petit`
`> is.vector(d)`	Utilisation de la fonction `is.vector()` renvoyant l'expression logique `TRUE` si son argument est un vecteur et `FALSE` sinon

TAB. 1 – Opérations de base sur les vecteurs (le symbole `>` en début de ligne est le prompt R)

```
> d <- c(5, 1, -5)
> rep(1:4, 2)
> rep(c(1.4, 3, 5), each=2)
> ?rep
> seq(0, 1, length=11)
> seq(1.575, 5.125, by=0.05)
> help(seq)
> sort(b)
> sample(c(0, 1), 10, rep=TRUE)
> sample(c(1,2,3), 3, prob=c(0.9,0.05,0.05), rep=TRUE)
> letters[2:5]
> LETTERS[20:26]
> sample(letters[1:9], 5)
> order(d)
> help(order)
> help.start()
> abs(d)
> log(abs(d))
```

TAB. 2 – Quelques fonctions utiles

`> a <- 1:20`	
`> b <- sample(1:10, 10)`	
`> x1 <- matrix(a, nrow=5)`	Création de la matrice numérique `x1` de dimension 5×4 ayant pour première ligne 1,6,11,16
`> x2 <- matrix(a, nrow=5, byrow=TRUE)`	Création de la matrice numérique `x2` de dimension 5×4 ayant pour première ligne 1,2,3,4
`> x3 <- t(x2)`	Transposition de la matrice `x2`
`> x4 <- matrix(b, ncol=2)`	
`> b <- x3%*%x2`	Produit matriciel entre `x2` et `x3` le langage contrôle l'adéquation des dimensions
`> f <- x2 %*% x4`	
`> dim(x1)`	Affichage de dimension de la matrice `x1`
`> dim(x4)`	
`> b[3,2]`	Sélection de l'élément `[3,2]` de la matrice `b`
`> b[,2]`	Sélection de la deuxième colonne de `b`
`> b[c(3,4),]`	Sélection des troisième et quatrième lignes de `b`
`> b[-2,]`	`b` sans sa seconde ligne
`> b[,-c(2,4)]`	`b` sans ses colonnes 2 et 4
`> b[,2]`	Sélection de la deuxième colonne de `b`
`> a <- sample(1:10,30,rep=T)`	
`> a <- matrix(a,nrow=6);a`	
`> a > 5`	Affichage d'une matrice de booléen dont l'élément `[i,j]` est égal à `TRUE` si `a[i,j]>5`
`> a[a < 5] <- 0`	On annule les éléments de `a` inférieurs à 5
`> rbind(x1, x2)`	Concaténation verticale des matrices `x1` et `x2`
`> cbind(x1, x4)`	Concaténation horizontale des matrices `x1` et `x4`
`> apply(x1, 2, sum)`	Calcul de la somme de `x1` par colonne
`> apply(x1, 1, sum)`	Calcul de la somme de `x1` par ligne
`> colSums(x1)`	
`> rowMeans(x2)`	

TAB. 3 – Opérations de base sur les matrices

```
> x <- array(1:50,c(2,5,5))
> x[1,2,2]
> dim(x)
> aperm(x)                    Transposition généralisée de x
```

TAB. 4 – Opérations sur les matrices à plus de deux dimensions

```
> x <- c(1, 2, 2, 1, 0, 2, 1, 0)
> z <- factor(x)
> attributes(z)
> z <- factor(x, labels = c("jamais", "rarement", "souvent"))
> attributes(z)
> ordered(sample(1:4,100,rep=TRUE))
```

TAB. 5 – Les facteurs

```
> li <- list(num = 1:5, y = "couleur", a = TRUE)
> li
> li$num
> li$a
> li[[1]]
> li[[3]]
> a <- matrix(c(6,2,0,2,6,0,0,0,36), nrow=3)
> res <- eigen(a, symmetric = TRUE)                    Diagonalisation
> res$values
> res$vectors
> a
> diag(res$values)
> res$vectors %*% diag(res$values) %*% t(res$vectors)
> str(res)
```

TAB. 6 – Opérations de base sur les listes

`> v1 <- sample(1:12, 30, rep=TRUE)`	Échantillonnage avec remise dans les entiers de 1 à 12
`> v2 <- sample(LETTERS[1:10], 30, rep=TRUE)`	
`> v3 <- runif(30)`	30 réalisations indépendantes d'une loi uniforme sur [0,1] (voir section 4)
`> v4 <- rnorm(30)`	30 réalisations indépendantes d'une loi normale de moyenne 0 et variance 1
`> xx <- data.frame(v1,v2,v3,v4)`	
`> xx$v2`	
`> summary(xx)`	
`> xx <- data.frame(v1,factor(v2),v3,v4)`	Constitution du tableau de données `xx` avec la variable `x2` déclarée comme facteur (variable qualitative)
`> summary(xx)`	
`> ma <-matrix(1:15, nrow = 3)`	
`> plot(ma)`	
`> ma <- as.data.frame(ma)`	
`> is.data.frame(ma)`	
`> plot(ma)`	
`> data()`	Ouvre une fenêtre texte listant l'ensemble des tableaux de données disponibles dans R
`> data(women)`	Charge le tableau de données `women`
`> names(women)`	Affiche le nom des variables de `women`
`> attach(women)`	
`> height`	

TAB. 7 – Opérations de base sur les tableaux de données

Loi	Nom	Paramètres	Valeurs par défaut
Beta	`beta`	`shape1, shape2`	
Binomiale	`binom`	`size, prob`	
Cauchy	`cauchy`	`location, scale`	0, 1
Khi-Deux	`chisq`	`df`	
Exponentielle	`exp`	`1/mean`	1
Fisher	`f`	`df1, df2`	
Gamma	`gamma`	`shape,1/scale`	-, 1
Géométrique	`geom`	`prob`	
Hypergéométrique	`hyper`	`m, n, k`	
Log-Normale	`lnorm`	`mean, sd`	0, 1
Logistique	`logis`	`location, scale`	0, 1
Normale	`norm`	`mean, sd`	0, 1
Poisson	`pois`	`lambda`	
Student	`t`	`df`	
Uniforme	`unif`	`min, max`	0, 1
Weibull	`weibull`	`shape`	

TAB. 8 – Les distributions usuelles

```
> qnorm(0.975) ; dnorm(0) ; pnorm(1.96)
> rnorm(20) ; rnorm(10, mean = 5, sd = 0.5)
> x <- seq(-3, 3, 0.1) ; densite <- dnorm(x)
> plot(x, densite, type = "l")
> runif(3)
> rt(5,10)
```

TAB. 9 – Utilisation des lois de probabilité

`> data(women)`	
`> names(women)`	
`> attach(women)`	
`> mean(height)`	Calcul de la moyenne empirique de la variable quantitative `height`
`> var(height)`	Calcul de la variance empirique de `height` estimateur non biaisé (diviseur $n-1$)
`> sd(height)`	Calcul de l'écart-type de `height`
`> median(height)`	Calcul de la médiane empirique de `height`
`> quantile(height)`	Calcul des quantiles empiriques de `height`
`> summary(weight)`	Résumé de `weight`
`> summary(women)`	Résumé de `women`
`> hist(weight, nclass=15)`	Histogramme de `weight` constitué d'approximativement 15 classes
`> boxplot(weight)`	Diagramme en boîte de `weight`
`> cor(height,weight)`	Calcul du coefficient de corrélation linéaire empirique entre `weight` et `height`
`> v1 <- rnorm(100)`	
`> hist(v1)`	
`> v2 <- factor( sample(letters[1:4],`	
`+ 100, rep = TRUE) )`	
`> table(v2)`	Résumé de la variable qualitative `v2`
`> barplot(table(v2))`	Diagramme en barre de `v2`
`> piechart(table(v2))`	Diagramme en secteur de `v2`
`> boxplot(v1 ~ v2)`	Diagramme en boîte de `v1` pour chaque modalité de `v2`

TAB. 10 – Statistique exploratoire

`> x <- rnorm(20)`	
`> print(x)`	
`> summary(x)`	
`> y <- 3 * x + 5 + rnorm(20, sd = 0.3)`	
`> plot(x, y)`	
`> reslm <- lm(y ~ x)`	Régression linéaire simple entre la variable à expliquer `y` et la variable explicative `x`
`> print(reslm)`	
`> summary(reslm)`	
`> plot(reslm, ask=T)`	
`> class(reslm) <- "sillyness"`	Ré-essayer les 3 commandes précédentes

TAB. 11 – Utilisation de méthodes génériques S3

```
> bool <- TRUE
> i <- 0
> while(bool) {i <- i+1; if (i>10) {bool <- FALSE}}
> i
> s <- 0
> x <- rnorm(100000)
> for (i in 1:100000) { s <- s + x[i] }
> s
> un <- rep(1, 100000)
> t(un) %*% x
> s <- 0
> system.time(for (i in 1:100000) {s <- s + x[i] })
> system.time(t(un) %*% x)
> system.time(sum(x))
```

TAB. 12 – Eléments de programmation

```
> data(iris); x <- iris[,1]; y <- iris[,2];
> g <- iris[,5]; gn <- as.numeric(g)

> palette(c("red","orange", "royalblue"))
> plot(x, y, pch=20+gn, bg=gn)
> rug(x, .02, 1) ; rug(y, .02, 2)
> boxplot(x~g)
> pairs(iris[,1:4], col=gn, pch="+", cex=1.3)
> require(lattice); densityplot(~x+y|g)
> den <- kde2d(x,y)
> filled.contour(pdf, plot.axes=
+ {contour(pdf, add=TRUE); axis(1); axis(2);
+ rug(jitter(x), -.02, 3); rug(jitter(y), -.02, 4);
+ points(x,y, pch="+", col="gray")})
> par(las=1); z <- rnorm(200);
> hist(z, prob=TRUE, border="red", col="lightgray")
> lines(density(z), lty="dotted", col="blue")
> rug(z, -.02, 3)
> curve(dnorm, col="green", lty="dashed", add=TRUE)
> legend("topleft",
+ c("vraie", "estimation par noyaux","histogramme"),
+ lty = c("dashed","dotted","solid"),
+ col = c("green", "blue","red"),
+ pch = c(NA,NA, 22), pt.bg=c(NA,NA,"lightgray"))
```

TAB. 13 – Quelques graphiques élémentaires (le caractère + en lieu et place du prompt > signifie que R attend la suite d'une instruction)

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-0803