



INSTITUT
de MATHÉMATIQUES
de TOULOUSE

INSTITUT DE MATHÉMATIQUES DE TOULOUSE
PLATEFORME DE BIOSTATISTIQUE



Pour se donner un peu d'

SÉBASTIEN DÉJEAN

19 avril 2012

Mises à jour et compléments :

math.univ-toulouse.fr/~sdejean

Institut de Mathématiques de Toulouse UMR 5219 — Université de Toulouse et CNRS
Plateforme de Biostatistique — Génopôle Toulouse Midi-Pyrénées



Table des matières

1	Introduction	3
1.1	R??????	3
1.2	Environnement de travail	4
1.2.1	R sous Windows	4
1.2.2	R sous Unix	5
1.3	Ligne de commande	5
1.4	Aide en ligne	5
1.5	Utilisation de packages	6
1.6	Principe du document	6
2	Initiation	8
2.1	Structures de données	8
2.1.1	Scalaire	8
2.1.2	Vecteur - vector	9
2.1.3	Matrice - matrix	10
2.1.4	Tableau - array	11
2.1.5	list	12
2.1.6	data.frame	12
2.2	Entrée / Sortie	13
2.2.1	Importation d'un jeu de données	13
2.2.2	Exportation d'objets R	14
2.2.3	Liens avec un tableur	14
3	Fonctions graphiques	16
3.1	Données qualitatives - Effectifs	16
3.2	Données quantitatives	17
3.3	Graphiques 2D	18
3.4	Vers la 3D	19
3.5	Exportation de graphiques	21
4	Programmation	22
4.1	Structure de contrôle	22
4.2	Fonctions	23

5	Un peu de statistique	25
5.1	Simulation	25
5.2	Tests statistiques	26
5.3	Statistique descriptive uni- et bi-dimensionnelle	27
5.4	Régression	28
5.5	Statistique descriptive multidimensionnelle	29
5.5.1	Analyse en composantes principales	29
5.5.2	Classification hiérarchique	30

Chapitre 1

Introduction

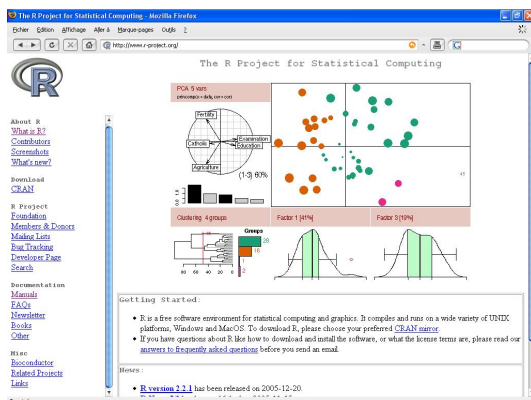
1.1 R??????

Extrait de la page d'accueil "The Comprehensive R Archive Network" :

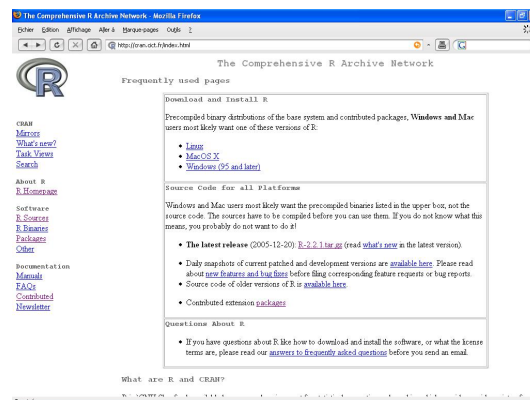
R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques : linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage (www.r-project.org/) for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you (cran.cict.fr) to minimize network load.

Dans les deux sites évoqués ci-dessus, on trouvera toutes les ressources nécessaires à l'utilisateur de R, débutant ou expérimenté : fichiers d'installation, mises à jour, packages, FAQ, newsletter, documentation...



www.r-project.org



cran.cict.fr

1.2 Environnement de travail

1.2.1 R sous Windows

Les captures d'écran présentées dans ce document proviennent de la version 2.2.1 (décembre 2005).

Une des premières choses à faire consiste à créer un environnement de travail afin de pouvoir récupérer le travail effectué lors d'une session ultérieure. Dans le menu **Fichier**, sélectionner **Sauver l'environnement de travail...** (Fig. 1.1) et placer le fichier suffixé par `.RData` dans le répertoire de votre choix.

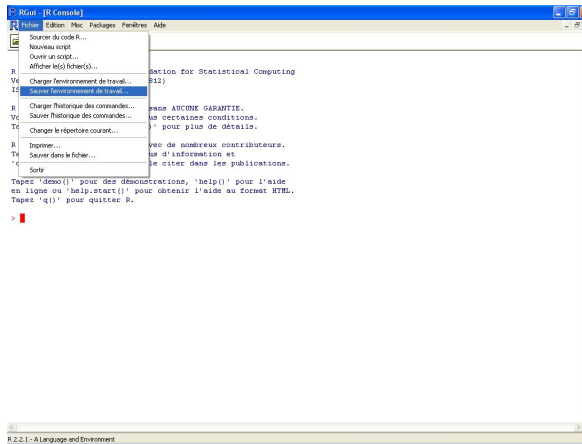


FIGURE 1.1 – Sauvegarde de l'environnement de travail.

La sauvegarde est à renouveler chaque fois que cela semble nécessaire (après la création de nouvelles variables, avant le lancement d'un gros calcul...) et surtout avant de quitter la session de R. *Attention*, à la sortie par **Fichier > Sortir**, répondre oui au message *Sauver une image de la session ?* (Fig. 1.2), sauvegarde l'environnement de travail dans un fichier `.RData` qui se trouve dans le répertoire courant (par défaut le répertoire d'installation de R (`C:/Program Files/R/R-2.2.1`)).

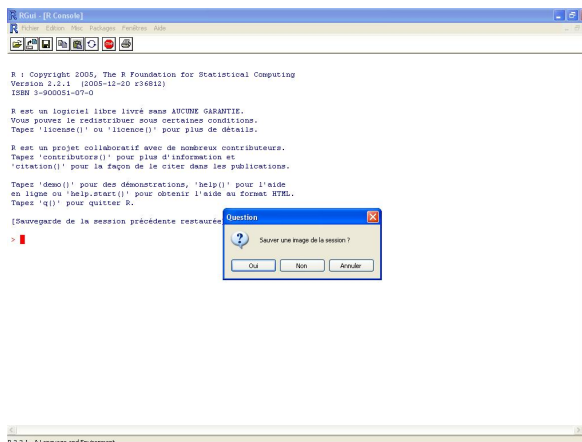


FIGURE 1.2 – Sortie de R.

1.2.2 R sous Unix

À l'appel de commande Unix, il suffit de saisir la commande `R` pour entrer dans R. Toutes les objets que vous créez dans R sont stockés dans un fichier `.RData` qui se crée automatiquement dans le répertoire courant. Ce fichier peut être sauvegardé à tout moment par la fonction `save.image()`. À la sortie de R, un message propose d'effectuer une dernière fois la sauvegarde du fichier `.RData`.

1.3 Ligne de commande

Quel que soit le système d'exploitation, l'utilisateur finit par se retrouver face à un *prompt* qui attend des commandes. On parle de logiciel fonctionnant en « ligne de commande » par opposition à un logiciel « clic-bouton » où la maîtrise de la souris suffit à faire plein de choses (et de bêtises...).

Une fonction est toujours appelée avec ses paramètres entre parenthèses. En l'absence de paramètres, l'utilisateur doit quand même saisir les parenthèses (vides), faute de quoi, c'est le code de la fonction qui est listé.

Le caractère `#` permet d'insérer un commentaire. C'est très utile voire indispensable lorsque l'on écrit des fonctions. Toute ligne précédée du caractère `#` n'est pas interprétée.

1.4 Aide en ligne

L'aide en ligne est accessible de la même façon par un appel à la fonction `help`. Par exemple, l'aide de la fonction `plot` est accessible par :

```
R> help(plot)
```

De manière générale, l'aide en ligne est composée des rubriques suivantes :

- Description
- Usage
- Arguments
- Details
- Value
- Note
- Authors
- Reference(s)
- See also
- Examples

Le nom de ces rubriques est suffisamment explicite pour désigner ce qu'elle contient. Pour un utilisateur non expérimenté, les deux dernières rubriques sont certainement les plus utiles : `See also` permet d'apprendre le nom de nouvelles fonctions en relation avec la recherche initiale et `Examples` permet de manipuler immédiatement, par copier-coller, toute nouvelle fonction.

La fonction `help.search()` permet de rechercher dans l'aide en ligne les fonctions en rapport avec un terme passé en paramètres. Pour tout savoir sur les fonctions en relation avec le tracé de graphiques, on pourra saisir :

```
R> help.search("plot")
```

Et pour comprendre comment fonctionne `help.search()`, on pourra bien entendu entrer :

```
R> help(help.search) et même
```

```
R> help(help) pour savoir comment utiliser la fonction help() !
```

1.5 Utilisation de packages

Les *packages* sont des ensembles de fonctions (et parfois de données) proposés par la communauté des utilisateurs de R et permettant d'accomplir des tâches spécifiques. En avril 2012, le *Comprehensive R Archive Network* en recense plus de 3700. L'utilisation d'une fonction incluse dans un package nécessite 2 étapes préalables (réalisables via le menu Packages de R dans l'environnement Windows) :

- l'installation : utiliser la fonction `install.packages()` pour récupérer les sources du package et les installer. Cette opération ne doit être accomplie qu'une seule fois.
- le chargement : utiliser la fonction `library()` pour charger le package et l'utiliser durant une session. Cette opération est à renouveler à chaque lancement de R.

Il est recommandé de procéder régulièrement à une mise à jour des packages (menu Packages ou fonction `update.packages()`).

La liste des packages installés (mais pas forcément chargés) est obtenue par la commande `library()`. La commande `search()` permet de vérifier qu'un package a bien été chargé. Pour lister le contenu d'un package, on peut utiliser la fonction `ls()` en précisant le numéro du package dans la liste renvoyée par `search()`. Exemple avec le package `foreign`.

- Installation du package :

```
R> install.packages("foreign")
```

- Vérification :

```
R> library()
```

- Chargement du package

```
R> library(foreign)
```

- Vérification

```
R> search()
```

 En général, le package se retrouve en 2ème position (juste après l'environnement de travail courant).

- Liste le contenu du package

```
R> ls(pos=2)
```

1.6 Principe du document

Le document est organisé de la façon suivante :

- une succession de commandes à saisir est donnée. Chaque commande est mise en évidence par une graphie particulière

```
R> ls()
```

La suite d'une commande sur la ligne suivante est identifiée par

```
+ xlab="Axes des abscisses"
```

En effet, lorsque R identifie une commande incomplète (parenthèse ouverte et pas fermée par exemple), le symbole "+" apparaît automatiquement sur la ligne suivante, il n'est pas à saisir.

Deux commandes indépendantes peuvent être tapées sur la même ligne séparées par un ";". Elles seront traitées séquentiellement.

- Quelques remarques viennent parfois mettre en évidence certains points particuliers de syntaxe ou autre.
- Des questions permettent de tester l'acquisition des fonctions étudiées.
- Une réponse possible à chaque question est fournie à la suite.

TABLE 1.1 – Principe du document

Commandes	Remarques
R >
R >
R >

Questions

...

...

...

Réponses

...

...

...

Chapitre 2

Initiation

Le chapitre d'initiation consiste essentiellement à manipuler différents types d'objets élémentaires existants dans R. Le problème d'importation/exportation de données est également traité.

2.1 Structures de données

2.1.1 Scalaire

```
R> 2+2
R> exp(10)
R> a = log(2)
R> b <- cos(10)
R> a+b
R> a
R> b
R> 2==3
R> b = 2<3
R> ls()
R> rm(a)
R> a
R> a="toto"
```

- Identifier les différents types de données : entier, réel, logique, chaîne de caractères.
- le signe `<-` est équivalent à `=`.
- l'affectation est également possible avec le signe `=>`, le nom de la variable étant alors placée après.
- La fonction `ls()` liste les variables de l'environnement de travail.
- La fonction `rm()` permet d'effacer une ou plusieurs variables.

2.1.2 Vecteur - vector

```
R> d = c(2,3,5,8,4,6);d
R> is.vector(d)
R> c(2,5,"toto")
R> 1:10
R> seq(from=1,to=20,by=2)
R> seq(1,20,by=5)
R> seq(1,20,length=5)
R> rep(5,times=10)
R> rep(c(1,2),3)
R> rep(c(1,2),each=3)
R> e = rep(1,10)
R> d[2];d[2:3];d[-3]
R> d[-1:2];d[-(1:2)] #*
R> d[3]=NA;d;summary(d)
R> is.na(d);help(NA)
R> any(is.na(d));all(is.na(d))
R> f = c(a=12,b=26,c=32,d=41);f
R> names(f);f["a"]
R> names(f)=c("a1","a2","a3","a4")
R> f>30;f[f>30] #**
R> which(f>30)
R> f[2] = 22;f+100;f+d
R> cos(f);length(f);sort(d)
```

- Les commandes illustrent comment créer un vecteur : soit en donnant les valeurs une à une (`c()`), soit par une séquence (`seq()`) soit par répétition (`rep()`).
- Les éléments d'un vecteur sont du même type (numérique, caractère...)
- L'extraction des éléments d'un vecteur se fait par l'opérateur `[]`.
- La première commande de la ligne `*` renvoie une erreur.
- On peut également nommer les éléments d'un vecteur.
- La notation `NA` (*Not Available*) permet de signaler des données manquantes.
- L'utilisation des fonctions `any()` et `all()` s'avère très utile pour synthétiser des commandes plus complexes.
- Noter la différence entre les 2 commandes de la ligne `**`.
- Un problème avec `f+d`?

Questions

1. Quels sont les différents paramétrages de la fonction `seq` ?
2. de la fonction `rep` ?
3. À quoi sert la fonction `unique` ? Illustrer son fonctionnement sur un exemple.

Réponses

1. `seq` construit un vecteur par une séquence de points équi-répartis entre `from` et `to` soit, par pas de `by`, soit, en précisant le nombre de points (`length`).
2. `rep` crée un vecteur par répétition d'une scalaire ou d'un vecteur. Dans ce dernier cas, le vecteur est répété séquentiellement (`times`) ou valeur par valeur (`each`).
3. `unique` supprime les répliqués dans un vecteur

```
R> unique(rep(c(1,2),each=3)) renvoie le vecteur 1-2.
```

2.1.3 Matrice - matrix

```
R> A = matrix(1:15, ncol=5); A; t(A)
R> B = matrix(1:15, nc=5, byrow=T)
R> B2=B; B2[1,1]="toto"; B2
R> cbind(A,B); rbind(A,B)
R> A[1,3]; A[,2]; B[2,]
R> A[1:3,2:4]
R> g = seq(0,1,length=20)
R> C = matrix(g, nrow=4)
R> C[C[,1]>0.1,] # *
R> D = matrix(runif(16), nc=4)
R> D>0.5
R> D[D[,1]>0.5,2] # **
R> A+B; A*B
R> cos(A); cos(A[1:2,1:2])
R> solve(A); solve(A[1:2,1:2])
R> A %*% B
R> A[1:2,1:2] %*% B[1:2,1:3]
R> apply(A, 2, sum) # ***
R> apply(D, 1, max)
```

Construction d'une matrice; voir section 5.1 pour la fonction `runif()`.

Comme pour les vecteurs, les éléments d'une matrice sont du même type.

Extraction des parties d'une matrice.

Noter qu'un argument peut-être nommer intégralement ou par un raccourci non ambigu : `ncol` devient `nc`.

Transposition d'une matrice (fonction `t()`).

Opérations terme à terme (*).

Produit matriciel (%*%).

Inversion d'une matrice (fonction `solve()`).

Fonction **apply!!!** C'est l'idée même du logiciel qui apparaît dans cette fonction. C'est plus rapide que l'utilisation de boucles (voir section 4.1).

Questions

1. Que font `rbind` et `cbind` ?
2. Décortiquer la ligne marquée * et décrire ce qu'elle fait.
3. Même chose avec **.
4. Que renvoie la ligne *** ?

Réponses

1. `rbind` et `cbind` collent deux vecteurs ou matrices respectivement en ligne ou en colonne.
2. `R> C[C[,1]>0.1,]` peut se décomposer ainsi :
`R> C[,1]` extrait la première colonne de la matrice C
`R> C[,1]>0.1` renvoie un vecteur logique de longueur le nombre de lignes de C contenant TRUE si la valeur est supérieure à 0.1 et FALSE sinon.
`R> C[C[,1]>0.1,]` extrait de la matrice C les lignes où les éléments sur la première colonne sont supérieurs à 0.1 et toutes les colonnes (rien après la virgule).
3. la ligne ** extrait de la colonne 2 de la matrice D, les lignes où l'élément sur la première colonne est supérieur à 0.5.
4. la ligne *** renvoie un vecteur de longueur 5 (le nombre de colonnes de A) dont chaque élément est la somme des éléments d'une colonne de A.

2.1.4 Tableau - array

```
R> array(c(1:8, rep(1,8),
+ seq(0,1,len=8)), dim = c(2,4,3))
R> E = .Last.value
R> E[, , 1]
R> dim(E); length(E)
R> nrow(E); ncol(E)
R> E+10
R> H=array(1:12, c(2,3,2))
R> apply(H, 1, mean)
R> apply(H, 2, mean)
R> apply(H, 3, mean)
```

Un array généralise une matrice. On peut le voir comme un tableau de matrices. Il peut avoir plus de 3 dimensions.

Une représentation de l'array H

	1	3	5	7	9	11
1	1	3	5	7	9	11
2	2	4	6	8	10	12

Éviter de nommer un objet F ou T (confusion avec TRUE ou FALSE).

Questions

1. Expliquer les résultats des 3 appels à la fonction `apply()`.
2. Créer un array à 4 dimensions et calculer la somme des éléments dans toutes les dimensions.

Réponses

1. Dans le premier cas, on calcule la moyenne de tous les éléments ligne par ligne. Les éléments de la ligne 1 sont tous les éléments de la tranche supérieure horizontale du parallélépipède H (1,3,5,7,9,11); de la tranche inférieure pour la ligne 2 (2,4,6,8,10,12). Dans le deuxième cas, le calcul est effectué colonne par colonne, le vecteur résultat est donc de longueur 3; il contient la moyenne des éléments des 3 tranches verticales (gauche - [1,2,7,8], centre - [3,4,9,10] et droite - [5,6,11,12]). Dans le troisième cas, le calcul de moyenne est fait sur les 2 tranches verticales "avant" (1,2,3,4,5,6) et "arrière" (7,8,9,10,11,12).
2. `R> H2=array(1 :24, c(2,3,2,2))` crée un array à 4 dimensions équivalent dans cet exemple à 2 tableaux H tels que représentés ci-dessus.
3. `R> apply(H, 1, sum)` [1] 144 156
4. `R> apply(H, 2, sum)` [1] 84 100 116
5. `R> apply(H, 3, sum)` [1] 114 186
6. `R> apply(H, 4, sum)` [1] 78 222

2.1.5 list

```
R> x = list("toto", 1:8); x
R> x[[1]]; x[[1]]+1; x[[2]]+10 #*
R> y = list(matrice=D, vecteur=f,
+ texte="toto", scalaire=8)
R> names(y); y[[1]]
R> y$matrice; y$vec
R> y[c("texte", "scal")] #**
R> y[c("texte", "scalaire")]
R> length(y)
R> length(y$vecteur)
R> cos(y$scal)+y[[2]][1]
```

Un objet de type `list` peut contenir plusieurs types d'objets (vecteur, matrice...).

Utile pour renvoyer plusieurs types d'informations comme résultats d'une fonction.

Accès aux composants d'un objet par son numéro entre double-crochets `[[]]` ou par l'opérateur `$` suivi du nom du composant (s'il en a un) ou d'un raccourci non ambigu.

Questions

1. Quel est le problème avec la ligne `*` ?
2. Et avec `**` ?

Réponses

1. C'est la 2ème commande de la ligne qui renvoie une erreur. On cherche à ajouter 1 à un élément qui n'est pas numérique.
2. Aucun composant de la liste ne s'appelle `scal` ("lettre à lettre"). Le raccourci non ambigu ne vaut qu'après l'opérateur `$`.

2.1.6 data.frame

```
R> taille = runif(20, 150, 180)
R> masse = runif(20, 50, 90)
R> sexe = sample(c("M", "F"), 20, rep=T)
R> coul=c("b", "n", "v", "m")
R> yeux = sample(coul, 20, rep=T)
R> table(sexe); table(yeux)
R> table(sexe, yeux)
R> H = data.frame(taille, masse,
+ sexe, yeux)
R> H; summary(H); head(H); tail(H)
R> H[1,]; H$taille; H$sexe
R> is.data.frame(H)
R> is.matrix(H)
R> MH = as.matrix(H)
R> summary(MH)
```

Structure spéciale pour la manipulation de jeux de données du type individus (en ligne) \times variables (en colonne).

La fonction `sample()` effectue un tirage aléatoire dans un ensemble donné (avec remplacement si `rep=T`).

Noter les résultats de la fonction `table()`.

Les variables peuvent être de types différents (numérique, caractère, logique...).

Noter les analogies entre `data.frame`, `list` et `matrix`.

Questions

1. Tester la fonction `summary` sur d'autres types d'objets.
2. Quel est l'effet de la conversion "forcée" du data.frame en matrice opérée par la fonction `as.matrix()` ?
3. Extraire la masse des individus dont la taille est supérieure à 160.
4. Extraire la masse et le sexe de ces mêmes individus.
5. Extraire la taille des individus de sexe masculin dont la masse est inférieure à 70. C'est possible en une seule ligne (voir l'opérateur `&`, `help("&")`).

Réponses

1. Fonction `summary`

```
R> vec=c(2,5,3,6,5,4,1,8);summary(vec)
```

```
R> mat=matrix(1:20,nc=4,nr=5)
```

```
R> summary(mat);summary(c(mat))
```

```
R> ...
```
2. La conversion en matrice implique que tous les éléments sont désormais du type `character`. La fonction `summary()` ne calcule plus des indicateurs numériques pour les colonnes `taille` et `masse`.
3.

```
R> H[H$taille>160,2]
```
4.

```
R> H[H$taille>160,2:3]
```
5.

```
R> H[H$masse<70 & H$sexe=="M",1]
```

2.2 Entrée / Sortie

2.2.1 Importation d'un jeu de données

Utiliser un éditeur de texte pour saisir les fichiers ci-dessous :

5,2.5,3.8	5 2.5 3.8	X1 ;X2 ;X3	5 ;2,5 ;3,8
8,3.2,3.4	8 3.2 3.4	5 ;2.5 ;3.8	8 ;3,2 ;3,4
12,4.6,5	12 4.6 5	8 ;3.2 ;3.4	12 ;4,6 ;5
		12 ;4.6 ;5	
fic1.csv	fic2.txt	fic3.txt	fic4.txt

```
R> fic1=read.table("fic1.csv",
+ sep=", ")
R> fic1
R> fic1b=read.csv("fic1.csv")
R> fic1b
R> fic1b=read.csv("fic1.csv",
+ header=FALSE)
R> fic1b
```

- La fonction `read.table()` permet de lire le contenu d'un fichier pour créer un objet R.
- Les arguments `header`, `sep` et `dec` permettent de prendre en compte les 3 facteurs : entête, séparateur des colonnes et séparateur décimal.
- Les fonctions `read.csv()` et `read.csv2()` sont analogues à `read.table()` mais avec un paramétrage par défaut différent.

Questions

1. Importer les fichiers `fic2.txt`, `fic3.txt` et `fic4.txt`.

Réponses

1. Importation des fichiers :

```
R> fic2 = read.table("fic2.txt")
R> fic3 = read.table("fic3.txt",header=T,sep=";")
R> fic4 = read.table("fic4.txt",sep=";",dec=",")
```

2.2.2 Exportation d'objets R

```
R> A=seq(1,10,l=50)
R> write.table(A,"A.txt")
R> sink("A2.txt")
R> A;summary(A)
R> sink()
R> summary(A)
```

- La fonction `write.table()` (analogue à `read.table()`) permet de renvoyer des données dans un fichier.
- La fonction `sink()` redirige le résultat des commandes vers un fichier au lieu de l'afficher à l'écran. Ne pas oublier de fermer le fichier en rappelant `sink()` sans argument.

2.2.3 Liens avec un tableur

Principe de base : tous les logiciels susceptibles de traiter des données peuvent importer et exporter des données dans des fichiers au format texte c'est-à-dire lisible dans un éditeur de texte quelconque (Bloc-notes, Emacs, Asedit...)

Les manipulations décrites ici ont pour objectif d'assurer les transferts de données entre R et un logiciel tableur de type Excel. Ce choix résulte du fait que c'est le principal cas de figure auquel nous sommes régulièrement confrontés. Cependant, la procédure s'applique à tout autre logiciel moyennant quelques légères adaptations.

Transférer des données de Excel (ou d'un tableur équivalent) dans R

- Dans Excel (ou dans un logiciel pouvant ouvrir les fichiers xls) :
 1. Ouvrir le fichier de données,
 2. Enregistrer le fichier au format csv (Comma Separated Value) en le spécifiant dans la liste déroulante (Type de fichier).

Le format csv est lisible dans un éditeur de texte quelconque, sa spécificité réside dans les virgules qui séparent les valeurs de colonnes différentes. Il est préférable au format txt qui propose des séparateurs tabulation qu'il est impossible de distinguer à l'oeil d'un espace alors que le codage est différent. Dans une version française d'Excel, le séparateur des colonnes est un point-virgule, la virgule étant le séparateur décimal.
 3. Quitter Excel.
- Dans R : L'importation du fichier de données s'effectue avec la fonction `read.table()`. L'aide en ligne (`help(read.table)`) vous fournira tous les détails sur l'utilisation de cette fonction. On peut en général assurer l'importation avec les 4 arguments : `file`, `header`, `sep`, `dec` :
 - * `file` : nom du fichier qui contient les données avec le chemin pour y accéder,
 - * `header` (TRUE ou FALSE) : permet de préciser si la première ligne contient le titre des colonnes
 - * `sep` : permet de préciser le caractère qui délimite les colonnes,
 - * `dec` : permet de préciser le séparateur décimalN.B. Le chemin pour accéder au fichier est indiqué par des / et non par des \.

Retour dans Excel de données traitées avec R

La fonction `write.table` permet d'envoyer un jeu de données R dans un fichier texte. Exemple : On souhaite exporter le jeu de données `donnees_traitees` dans un fichier afin de poursuivre son analyse avec Excel. Pour cela, on précise que le séparateur décimal est la virgule et le séparateur le point-virgule.

```
R> write.table(donnees_traitees,  
+ "data-trait.csv", sep=";", dec=", ")
```

Depuis Excel, on ouvre le fichier `data-trait.csv` en précisant le type de fichier csv, sinon toutes les colonnes seront concaténées dans la première colonne du tableau Excel.

Il existe des packages R permettant d'assurer plus directement ces transferts mais la procédure décrite ici est en général recommandée. Pour plus de détails, on pourra se référer au document **R Data Import/Export** disponible sur le CRAN.

Chapitre 3

Fonctions graphiques

`R> help.search("plot")` vous donnera un aperçu des multiples fonctions liées à la représentation graphique dans R.

3.1 Données qualitatives - Effectifs

```
R> vec=c(12,10,7,13,26,16,4,12)
R> pie(vec);pie(vec,clockwise=T)
R> names(vec)=LETTERS[1:8]
R> barplot(vec)
R> par(mfrow=c(1,2))
R> pie(vec);barplot(vec)
R> par(mfrow=c(1,1))
R> windows()
R> barplot(vec,col=1:8)
R> dotchart(vec)
R> par(bg="lightgrey")
R> dotchart(vec,pch=16,col=1:8)
R> par(bg="white")
R> colors()
```

- Les fonctions `pie()` et `barplot()`, comme les autres fonctions graphiques, disposent d'un très grand nombre de paramètres permettant de modifier l'aspect du graphique obtenu.
- Noter l'utilisation de la commande `par(mfrow=c(1,2))` qui découpe la fenêtre graphique en 1 ligne et 2 colonnes et inclue les graphiques à venir ligne par ligne.
- La commande `windows()` ouvre une autre fenêtre graphique.

Questions

1. Quelle est la différence entre `par(mfrow=c(2,2))` et `par(mfcol=c(2,2))` ?
2. Consulter l'aide en ligne pour tester certains arguments optionnels des fonctions `pie()` et `barplot()`.
3. Commenter les commandes suivantes (extraites de l'aide en ligne de `pie()`):

```
R> n=200
```

```
R> pie(rep(1,n), labels="", col=rainbow(n), border=NA)
```



Réponses

1. Les 2 commandes découpent la fenêtre graphique en 4 cellules. La différence entre les 2 commandes est illustrée par le tableau ci-dessous (`par(mfrow=c(2,2))` - `par(mfcol=c(2,2))`):

1 - 1	2 - 3
3 - 2	4 - 4

Avec `par(mfrow=c(2,2))`, les 4 graphiques à venir sont intégrés ligne par ligne ; avec `par(mfcol=c(2,2))`, ils le sont colonne par colonne.

2. Par exemple, pour modifier les couleurs du camembert :

```
R> pie(vect, col=rgb(0,0,seq(0,1,l=8)))
```

```
R> pie(vect, col=rainbow(8))
```

```
R> ...
```

3. Commentaire de la ligne de commande :

- `R> n=200` # La valeur 200 est affectée à n.
- `R> rep(1,n)` # Crée le vecteur de taille n ne contenant que des 1.
- `R> pie(rep(1,n))` # Produit un camembert pour n variables d'effectifs tous égaux à 1. Les secteurs sont donc de même angle.
- `R> rainbow(n)` # Renvoie un vecteur de longueur n contenant un dégradé des couleurs de l'arc-en-ciel.
- `R> border=NA` # Évite de tracer les bords des secteurs.
- `R> labels=""` # Aucune étiquette n'identifie les secteurs.

3.2 Données quantitatives

```
R> x=rnorm(50)
R> boxplot(x)
R> hist(x)
R> stripchart(x)
```

- La génération de nombres aléatoires est traitée dans la section 5.1.
- La section 5.3 détaille certaines de ces fonctions graphiques.

Questions

1. Représenter dans la même fenêtre graphique, le "stripchart", le diagramme-boîte horizontal et l'histogramme correspondant l'un sous l'autre.

Réponses

1. Représentation des 3 graphiques

```
R> par(mfrow=c(3,1))
```

```
R> stripchart(x)
```

```
R> boxplot(x, horizontal=T)
```

```
R> hist(x)
```

3.3 Graphiques 2D

```
R> x=seq(-10, 10, l=50)
R> plot(x, sin(x))
R> plot(x, sin(x), type="l")
R> abline(v=0, col="blue", lwd=5, lty=3)
R> abline(h=sin(0.7), col=3)
R> text(-5, -0.5, "texte", font=3)
R> par(mfrow=c(1, 2))
R> plot(x, sin(x), type="l", col=1,
+ main="sinus")
R> plot(x, cos(x), type="b", col=3,
+ xlab="Abscisses")
R> par(mfrow=c(1, 1))
R> plot(x, cos(x), type="l")
R> points(0, 1, pch="o", cex=3,
+ col="blue")
R> lines(c(-5, 5), c(0, 0), lty=2, col=2)
R> locator(1); text(locator(2),
+ c("clic", "clac"), font=c(2, 3))
R> A=cbind(seq(0, 1, l=20), rnorm(20),
+ runif(20))
R> matplot(A, type="b")
```

Certaines fonctions créent un nouveau graphique, d'autres ajoutent des éléments à un graphique existant.

La fonction `locator()` attend de l'utilisateur qu'il clique sur la fenêtre graphique.

Les arguments `main`, `xlab`, `ylab`, `sub...` permettent de placer les légendes des axes et du graphique.

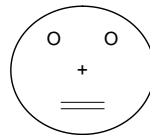
La fonction `matplot()` trace les colonnes d'une matrice.

Les options graphiques sont listées dans la rubrique "Graphical Parameters" de

```
R> help(par)
```

Questions

1. Saurez-vous tracer la « tête à Toto » ?



Réponses

1. Voici un exemple pour obtenir la « tête à Toto » :

```
R> plot(0, 0, xlim=c(-15, 15), ylim=c(-15, 15),
+ type="n", axes=FALSE, xlab="", ylab="")
R> points(0, 0, pch="+", cex=4) # le nez
```

```
R> points(c(-4, 4), c(5, 5), pch="0", cex=4) # les yeux
R> lines(c(-3, 3), c(-5, -5), lwd=3) # la lèvre supérieure
R> lines(c(-3, 3), c(-6, -6), lwd=3) # la lèvre inférieure
R> lines(10*sin(0:360*pi/180),
+ 10*cos(0:360*pi/180), lwd=5) # le contour du visage
En option, le chapeau :
R> lines(c(-12, 12), c(10, 10), lwd=3)
R> rect(-6, 10, 6, 14, border=1, lwd=3, col=1)
```

3.4 Vers la 3D

```
R> M=matrix(1:100, nc=10)
R> image(M)
R> x = seq(-10, 10, length= 30); y=x
R> f = fonction(x, y)
+ {r=sqrt(x^2+y^2); 10 * sin(r)/r}
R> z = outer(x, y, f)
R> z[is.na(z)] = 1
R> persp(x, y, z)
R> persp(x, y, z, theta=30, phi=30,
+ expand = 0.5, col="lightblue")
R> image(x, y, z)
R> contour(x, y, z)
R> filled.contour(x, y, z)
R> image(x, y, z)
R> contour(x, y, z, add=T)
R> install.packages("rgl")
R> library(rgl)
R> x=rnorm(100); y=rnorm(100)
R> z=rnorm(100)
R> plot3d(x, y, z)
```

Représentation sous forme d'image (`image()`), de nappe (`persp()`) ou de contour (`contour()`).

Exemple de la fameuse fonction `sinc()` (sinus cardinal), extrait de l'aide en ligne de la fonction `persp()`.

Pour mieux comprendre ce que fait la fonction `outer()`, on testera :

```
R> x=y=1:5
```

```
R> z=outer(x, y, "+")
```

Le package `rgl` permet de produire des représentations graphiques 3D interactives (zoom, rotation)

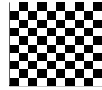
D'autres packages permettent d'améliorer les sorties graphiques standards (voir par exemple `ggplot2`).

Questions

1. Représenter le drapeau français (ou tout autre drapeau de votre choix, à vos risques et périls).



2. Représenter un damier 10×10 (c'est possible en une seule ligne).



Réponses

1. Drapeaux :

– France :

```
R> image(t(matrix(c(rep(-1,30), rep(0,30), rep(1,30)),
+ nr=10,nc=9)), col=c("blue", "white", "red"), axes=F); box()
```

– Italie :

```
R> image(t(matrix(c(rep(-1,30), rep(0,30), rep(1,30)),
+ nr=10,nc=9)), col=c("green", "white", "red"), axes=F); box()
```

– Japon :

```
R> plot(0,0,xlim=c(-10,10),ylim=c(-10,10),type="n",
+ axes=FALSE,xlab="",ylab="")
```

```
R> polygon(c(-8,8,8,-8),c(-5,-5,5,5),col="white",border=1)
```

```
R> lines(3*sin(0:360*pi/180),3*cos(0:360*pi/180),
+ lwd=5,col="red")
```

```
R> polygon(3*sin(0:360*pi/180),3*cos(0:360*pi/180),
+ lwd=5,col="red",border=0)
```

– Canada (!!!) :

```
R> X=c(-0.25,-0.2,-2.5,-2,-4,-3.5,-3.75,-2.25,-2,-1,-1.5,
+ -0.75,0,0.75,1.5,1,2,2.25,3.75,3.5,4,2,2.5,0.2,0.25)
```

```
R> Y=c(-5,-2.5,-3,-2,0,0.25,1.75,1.5,2,1,4,3.5,5,3.5,4,1,
+ 2,1.5,1.75,0.25,0,-2,-3,-2.5,-5)
```

```
R> plot(0,0,xlim=c(-10,10),ylim=c(-10,10),type="n",
+ axes=FALSE,xlab="",ylab="")
```

```
R> polygon(c(-8,8,8,-8),c(-6,-6,6,6),col="white")
```

```
R> polygon(c(-8,-5,-5,-8),c(-6,-6,6,6),col="red")
```

```
R> polygon(c(5,8,8,5),c(-6,-6,6,6),col="red")
```

```
R> polygon(X,Y,col="red",border=0)
```

– ...

2. Damier (en une seule ligne, sans ";") :

```
R> image(matrix(rep(c(rep(c(1,0),5),rep(c(0,1),5)),5),
+ nc=10),col=c("white","black"),axes=F)
```

Juste pour empêcher les pions de tomber :

```
R> box()
```

En détaillant, les différentes étapes, cela peut ressembler à :

```
R> v1 = rep(c(1,0),5) #1010101010
```

```
R> v2 = rep(c(0,1),5) #0101010101
```

```

R> v3 = c(v1, v2) #10101010100101010101
R> v4 = rep(v3, 5) #10101010100101010101011010101010010101010110101010100101010110
101010100101010101101010101001010101
      1  0  1  0  1  0  1  0  1  0
      0  1  0  1  0  1  0  1  0  1
      1  0  1  0  1  0  1  0  1  0
      0  1  0  1  0  1  0  1  0  1
R> mat = matrix(v4, nc=10)
      1  0  1  0  1  0  1  0  1  0
      0  1  0  1  0  1  0  1  0  1
      1  0  1  0  1  0  1  0  1  0
      0  1  0  1  0  1  0  1  0  1
      1  0  1  0  1  0  1  0  1  0
      0  1  0  1  0  1  0  1  0  1
      1  0  1  0  1  0  1  0  1  0
      0  1  0  1  0  1  0  1  0  1
R> image(mat, col=c("white", "black"))

```

3.5 Exportation de graphiques

Dans l'environnement Windows, une première solution consiste à copier le graphique (menu Fichier > Copier vers le presse-papier). Le graphique ainsi placé dans le presse-papier peut ensuite être collé dans le logiciel de son choix.

Pour sauvegarder le graphique dans un fichier, on utilisera dans le menu Fichier la rubrique Sauver sous. Plusieurs formats sont disponibles : vectoriel (métafichier), post-script, pdf, png, bmp, jpeg.

Une autre façon de procéder, et c'est la seule possible en environnement Unix, consiste à utiliser les fonctions associées à la sauvegarde de fichiers graphiques : `bmp()`, `jpeg()`, `png()`, `pdf()`, `postscript()`. La procédure à suivre est la suivante :

1. Créer un fichier graphique vers lequel la sortie graphique est redirigée

```
R> jpeg("fichier.jpg")
```
2. Tracer le graphique : **le graphique n'apparaît pas à l'écran.**

```
R> plot(1:100)
```

```
R> text(20, 80, "abcdef")
```
3. Fermer le fichier. **Ne surtout pas oublier cette étape!** La sortie graphique revient alors à l'écran pour le prochain tracé.

```
R> dev.off()
```

Cette procédure est également utile en environnement Windows lorsque l'on souhaite sauvegarder un graphique tracé lors du déroulement d'une fonction.

Chapitre 4

Programmation

4.1 Structure de contrôle

```
R> for (i in 1:10) print(i)
R> y=z=0;
R> for (i in 1:10) {
+ x=runif(1)
+ if (x>0.5) y=y+1
+ else z=z+1 }
R> y;z
R> for (i in c(2,4,5,8)) print(i)
R> x = rnorm(100)
R> y = ifelse(x>0, 1, -1)
R> i=0
R> while (i<10){
+ print(i)
+ i=i+1}
```

— La répétition d'instructions se fait par `for` ou `while`.

— On peut très souvent s'abstenir d'avoir recours à des boucles en "*pensant*" vectoriel (ou matriciel).

— La condition s'exprime par `if ... else` ou en condensé par `ifelse`.

Questions

1. Que pensez-vous de :
`for (i in 1:length(b)) a[i]=cos(b[i])`
2. Comment obtenir l'équivalent de `y` et `z` dans la deuxième boucle `for` sans passer par une boucle ?
3. Dans l'enchaînement de commandes ci-dessous, supprimer d'abord la boucle `for` sur `j` puis les 2 boucles.

```
R> M=matrix(1:20,nr=5,nc=4)
```

```
R> res=rep(0,5)
```

```
R> for (i in 1:5){
```

```
+ tmp=0
```

```
+ for (j in 1:4) {tmp = tmp + M[i,j]}
+ res[i]=tmp}
```

Réponses

1. Cette boucle est inutile. Il suffit de saisir
`R> a=cos(b)` . L'élément de base de R est la matrice dont le vecteur est un cas particulier.
2. Une solution consiste à sommer les éléments TRUE d'un vecteur logique
`R> x=runif(10);y=sum(x>0.5);z=10-y`
3. Suppression de boucles
 - Boucle for sur j :
`R> for (i in 1:5) res[i]=sum(M[i,])`
 - Les 2 boucles :
`R> res=apply(M,1,sum)`

4.2 Fonctions

```
R> MaFonction=function(x){x+2}
R> ls()
R> MaFonction
R> MaFonction(3);x = MaFonction(4)
R> MaFonction = fix(MaFonction)
R> Fonction2=function(a,b=0){a+b}
R> Fonction2(2,3);Fonction2(5)
R> Fonction3=function(a,b=a){a+b}
R> Fonction3(2,3);Fonction3(5)
R> Calcule=function(r){
+ p=2*pi*r;s=pi*r*r;
+ list(rayon=r,perimetre=p,
+ surface=s)}
R> resultat=Calcule(3)
R> resultat$ray
R> 2*pi*resultat$r==resultat$perim
```

- La fonction `function()` permet de créer une fonction.
- La fonction `fix()` permet d'éditer le corps de la fonction.
- On peut donner une valeur par défaut à un paramètre d'une fonction. Cela signifie que si l'utilisateur ne renseigne pas un paramètre, une valeur par défaut est prévue et ne génère donc pas d'erreur. Noter la différence entre `Fonction2` et `Fonction3`.
- Noter l'utilisation d'un objet de type `list` pour renvoyer les résultats de la fonction `Calcule()`.

Questions

1. le recours à un objet de type `list` est-il indispensable pour la fonction `Calcule()` ?
2. Écrire une fonction qui calcule le périmètre et la surface d'un rectangle à partir des longueurs l_1 et l_2 des deux côtés. La fonction renvoie également la longueur et la largeur du rectangle.
3. Écrire une fonction qui calcule les n premiers termes de la suite de Fibonacci ($u_1 = 0, u_2 = 1, \forall n > 2, u_n = u_{n-1} + u_{n-2}$)

- Utiliser cette fonction pour calculer le rapport entre 2 termes consécutifs. Représenter ce rapport en fonction du nombre de termes pour $n = 20$. Que constatez-vous ? Avez-vous lu *Da Vinci Code* ?
- Écrire une fonction qui supprime les lignes d'un data.frame ou d'une matrice présentant au moins une valeur manquante.

Réponses

- Les 3 éléments à renvoyer étant de type numérique, un vecteur peut suffire.
- Fonction `rectangle()` (la fonction `rect()` existe déjà) :

```
R> rectangle=function(l1,l2) {
+ p=(l1+l2)*2
+ s=l1*l2
+ list(largeur=min(l1,l2),longueur=max(l1,l2),
+ perimetre=p,surface=s)}
```

Utilisation de la fonction :

```
R> rectangle(4,6);res=rectangle(8,7)
```

- Fonction `fibonacci()` pour calculer les n premiers termes de la suite de Fibonacci :

```
R> fibo=function(n) {
+ res=rep(0,n);res[1]=0;res[2]=1
+ for (i in 3:n) res[i]=res[i-1]+res[i-2]
+ res}
```

- Calcul du rapport de 2 termes consécutifs

```
R> res=fibo(20)
R> ratio=res[2:20]/res[1:19]
R> plot(1:19,ratio,type="b")
```

Le rapport tend vers le nombre d'or $\frac{1+\sqrt{5}}{2} \approx 1.618034$.

- Une façon, parmi beaucoup d'autres, de répondre à la question :
 - Création d'une fonction intermédiaire `ligne.NA()` qui repère s'il y a au moins une valeur manquante sur un vecteur passé en paramètre

```
R> ligne.NA=function(vec){any(is.na(vec))}
```

- Puis la fonction qui va filtrer les lignes avec au moins une valeur manquante :

```
R> filtre.NA=function(mat) {
+ tmp = apply(mat,1,ligne.NA)
+ mat[!tmp,]}
```

- Application sur une matrice de test

```
R> matrice.test = matrix(1:40,nc=5)
R> matrice.test[2,5]=NA
R> matrice.test[4,2]=NA
R> matrice.test[7,1]=NA
R> matrice.test[7,5]=NA
R> filtre.NA(matrice.test)
```

Chapitre 5

Un peu de statistique

Rappel :

*R is 'GNU S', a freely available language and environment for **statistical** computing and graphics which provides a wide variety of **statistical** and graphical techniques : ...*

5.1 Simulation

```
R> help.search("Distribution")
R> help(rnorm)
R> rnorm(10);pnorm(0)
R> qnorm(0.5);dnorm(0)
R> plot(dnorm, -3, 3,
+ col="blue", lwd=3)
R> y=seq(qnorm(1-0.025), 3, l=100)
R> polygon(c(y, rev(y)),
+ c(dnorm(y), rep(0, 100)), col=3)
R> text(2.2, 0.015, "0.025", cex=0.9,
+ font=2)
```

La plupart des distributions courantes sont programmées dans R. Noter les différentes fonctions liées à chaque distribution. Par exemple, pour la loi normale : `dnorm()`, `pnorm()`, `qnorm()`, `rnorm()` donnent respectivement la densité, la fonction de répartition, la fonction quantile et un générateur aléatoire. Observer au passage une utilisation un peu particulière de la fonction `plot()`.

Questions

1. Tracer les fonctions densités et les fonctions de répartition de quelques distributions de probabilités usuelles (exponentielle, log-normale, Cauchy...)

Réponses

1. Quelques densités et fonctions de répartition associées (sur un même graphique) :

```
R> par(mfcol=c(2, 4))
R> plot(dnorm, -5, 5)
R> plot(pnorm, -5, 5)
```

```
R> plot(dexp, 0, 5)
R> plot(pexp, 0, 5)
R> plot(dlnorm, 0, 5)
R> plot(plnorm, 0, 5)
R> plot(dcauchy, 0, 5)
R> plot(pcauchy, 0, 5)
```

5.2 Tests statistiques

```
R> help.search("test")
R> x=rnorm(100)
R> y=rnorm(100,mean=1)
R> t.test(x,y)
R> MonResultat = t.test(x,y)
R> MonResultat
R> names(MonResultat)
R> MonResultat$p.value
R> var.test(x,y)
R> t.test(x,y,var.equal=T)
R> cor.test(x,y)
R> ks.test(x,y)
R> ks.test(x,"pnorm")
R> ks.test(y,"pnorm")
R> ks.test(y,"pnorm",1)
```

- La plupart des tests statistiques courants (et bien d'autres) sont programmés dans R.
- Test de Student pour la comparaison de moyennes.
- Test de Fisher pour la comparaison de variances.
- Test de nullité du coefficient de corrélation.
- Test de Kolomogorov-Smirnov
- ...
- Le résultat d'un test statistique peut être sauvegardé dans une variable pour accéder à différents éléments (statistique de test, p-value, intervalle de confiance...)

Questions

1. Appliquer le test de Shapiro-Wilk à la place du test de Kolmogorov-Smirnov.
2. Tester la nullité du coefficient de corrélation de Spearman entre x et y.

Réponses

1. Mise en œuvre du test de Shapiro-Wilk

`R> help.search("shapiro")` nous indique que la fonction qui permet de mettre en œuvre le test est la fonction `shapiro.test()`.

`R> help(shapiro.test)` pour voir comment utiliser la fonction

`R> shapiro.test(x)` pour tester la normalité de la variable x.

2. `R> cor.test(x,y,method="spearman")`

5.3 Statistique descriptive uni- et bi-dimensionnelle

```
R> x=runif(100)
R> y=runif(100)
R> mean(x);var(x);sd(x)
R> min(x);max(x)
R> quantile(x);median(x)
R> quantile(x,0.5)
R> quantile(x,0.9)
R> boxplot(x,plot=FALSE)
R> cov(x,y);cor(x,y)
R> cor(x,y,method="spearman")
R> summary(x)
R> stem(x);stem(y)
R> boxplot(x);hist(x)
R> x[25]=2
R> res=boxplot(x);res
R> hist(x)
R> x[25]=runif(1)
R> hist(x,density=10)
R> hist(x,plot=FALSE)
R> hist(x,nclass=5)
```

On revient ici en partie sur les graphiques pour données quantitatives évoquées dans la section 3.2.

Les fonctions `boxplot()` et `hist()` peuvent ne pas produire un graphique (option `plot=FALSE`).

La fonction `stem` produit une diagramme *stem-and-leaf* (tige et feuille) qui donne un aperçu de la répartition des données de façon plus « rustique » qu'un histogramme

La fonction `hist()` propose des options pour modifier l'apparence de l'histogramme.

Questions

1. Calculer quelques indicateurs statistiques pour un échantillon de longueur 1000 tiré aléatoirement selon une loi normale de moyenne 10 et de variance 25.

2. Même chose avec le vecteur `x3` défini par :

```
R> x3=c(rnorm(500,5,1),rnorm(500,10,1))
```

 Quelle est la particularité de ce vecteur ?

Réponses

1. Tirage selon une loi normale de moyenne et de variance 25 (donc d'écart-type 5, valeur à passer en paramètre de `rnorm()`)

```
R> x2 = rnorm(1000,mean=10,sd=5)
```

```
R> summary(x2);boxplot(x2);hist(x2)
```

2. Vecteur `x3`

```
R> summary(x3);boxplot(x3);hist(x3)
```

La distribution du vecteur `x3` fait clairement apparaître une bi-modalité. Elle est visible ici sur l'histogramme.

5.4 Régression

```
R> search()
R> ls(pos=7) #(*)
R> help(cars)
R> res1 = lm(dist ~ speed,
+ data=cars)
R> res1
R> plot(cars)
R> abline(res1)
R> names(res1)
R> summary(res1)
R> anova(res1)
R> res2=lowess(cars$speed,
+ cars$dist, f=0.5)
R> lines(res2, col="blue", lty=2)
```

Vérifier la présence de "package:datasets" dans la liste des bibliothèques disponibles (fonction `search()`).

Dans la ligne (*), le chiffre 7 correspond au numéro de l'extension "package:datasets" dans la liste fournie par `search`. À vérifier.

Vérifier la présence du jeu de données `cars`. Une description des données est fournie par la fonction `help()`.

La fonction `lowess()` (*LOcally WEighted Scatterplot Smooth*) produit un lissage des données par une fonction polynomiale par morceaux.

Questions

1. Représenter, en plus de la droite de régression, les points correspondants aux valeurs de distance ajustées par le modèle de régression linéaire.
2. Relier par des segments de droite les valeurs de distance ajustées et réelles pour chaque valeur de vitesse.
3. Reprendre la fonction `lowess()` en modifiant la valeur du paramètre `f`. Représenter systématiquement la courbe lissée correspondante sur le graphique des données initiales.
4. Ajouter une légende à ce graphique (fonction `legend`).

Questions

1. Représentations des valeurs ajustées :

```
R> plot(cars); abline(res1)
```

```
R> points(cars[,1], res1$fitted, pch="+", col="blue")
```

2. Segments de droite entre valeurs ajustées et observées :

```
R> for (i in 1:length(res1$fitted))
```

```
+ lines(c(cars[i,1], cars[i,1]),
```

```
+ c(cars[i,2], res1$fitted[i]), col="red", lty=2)
```

3. Quelques courbes `lowess` ajoutées au graphiques des données :

```
R> plot(cars)
```

```
R> lines(lowess(cars$speed, cars$dist, f=0.1), col=2)
```

```
R> lines(lowess(cars$speed, cars$dist, f=0.3), col=3)
R> lines(lowess(cars$speed, cars$dist, f=0.5), col=4)
R> lines(lowess(cars$speed, cars$dist, f=0.7), col=5)
R> lines(lowess(cars$speed, cars$dist, f=0.9), col=6)
```

4. Légende positionnée par un clic de la souris :

```
R> legend(locator(1),
+ c("f=0.1", "f=0.3", "f=0.5", "f=0.7", "f=0.9"),
+ col=2:6, lty=1, text.col=2:6)
```

5.5 Statistique descriptive multidimensionnelle

5.5.1 Analyse en composantes principales

```
R> df.body=read.table("body.csv",
+ sep=";", dec=".", header=T,
+ row.names=1)
R> dim(df.body)
R> dimnames(df.body)
R> acp.body = prcomp(df.body)
R> names(acp.body)
R> summary(acp.body)
R> plot(acp.body)
R> biplot(acp.body)
R> plot(acp.body$x)
R> plot(acp.body$x, type="n")
R> text(acp.body$x,
+ dimnames(df.body)[[1]])
```

Le fichier `body.csv` est accessible ici : math...fr/~sdejean/R

Noter la redéfinition des fonctions `summary()` et `plot()`.

La fonction `biplot()` peut être utilisée indépendamment du résultat de `prcomp()`.

La fonction `princomp()` existe aussi pour effectuer une ACP (les résultats sont équivalents mais les calculs permettant de les obtenir diffèrent avec ceux de `prcomp()`).

Questions

1. Représenter seulement les variables sur les 2 premières composantes principales

Réponses

1. `R> plot(acp.body$rotation, type="n", xlim=c(-1, 1), ylim=c(-1, 1))` ne représente rien avec l'option `type="n"`
2. `R> text(acp.body$rotation, colnames(df.body))` ajoute le nom des variables
3. `R> abline(v=0, h=0, lty=2)` complète le graphique avec les axes du repère

5.5.2 Classification hiérarchique

```
R> dist.body=dist(df.body)
R> cor.dist=function(x){
+ as.dist(1-cor(t(x)))}
R> dist.c.body=cor.dist(df.body)
R> hc.body=hclust(dist.body,
+ method="ward")
R> plot(hc.body)
R> plot(hc.body, hang=-1)
R> plot(hclust(dist.c.body,
+ method="ave"), hang=-1)
R> plot(hc.body, hang=-1)
R> rhc.body=rect.hclust(hc.body,
+ k=3)
```

- On récupère certains objets créés précédemment.
- Deux notions de distances : inter-individus et inter-groupes.
- Représentation (par `plot()`) d'une arbre de classification ou dendrogramme.
- Le résultat de `rect.hclust()` n'est pas seulement graphique. Voir le contenu de `rhc.body`. Voir également la fonction `cutree()` au fonctionnement similaire.

Questions

1. Mettre en œuvre la classification hiérarchique sur les données `eurodist`.
2. Représenter les dendrogrammes pour différents critères d'agglomération.

Réponses


1. Classification hiérarchique :

```
R> hc.eurodist.ward=hclust(eurodist, method="ward")
R> hc.eurodist.single=hclust(eurodist, method="single")
R> hc.eurodist.complete=hclust(eurodist, method="complete")
R> hc.eurodist.ave=hclust(eurodist, method="ave")
```

2. Représentation des dendrogrammes :

```
R> par(mfrow=c(1,4))
R> plot(hc.eurodist.ward, hang=-1, main="Ward")
R> plot(hc.eurodist.single, hang=-1, main="Single")
R> plot(hc.eurodist.complete, hang=-1, main="Complete")
R> plot(hc.eurodist.ave, hang=-1, main="Average")
```

Bibliographie

Comme cela est évoqué en début de document, toutes les ressources souhaitables sur  sont disponibles sur les 2 sites :

- www.r-project.org
- cran.cict.fr

Parmi la documentation, signalons les documents en français :

- *R pour les débutants* - Emmanuel Paradis, version française de *R for Beginners* (81 pages).
- *Introduction au système R* - Yves Brostaux, (22 pages avec jeux de données).

ainsi que le document

- Writing R extensions

utile à tout utilisateur désireux de développer en R (interface avec des programmes C, création de package...).

Sur le web, le site de Vincent Zoonekynd

- *Statistiques avec R* - zoonek2.free.fr/UNIX/48_R_2004/all.html

est très complet sur les méthodes statistiques élémentaires accessibles sur la base de R (sans package additionnel) ainsi que sur les solutions graphiques.

Certains ouvrages sont également très utiles pour se perfectionner sur des points particuliers :

- *Le logiciel R : Maîtriser le langage, Effectuer des analyses statistiques* P. Lafaye de Micheaux, R. Drouilhet, B. Liquet, 2010, Springer
- les ouvrages de la série Use R !, Springer