INSTITUT DE MATHÉMATIQUES DE TOULOUSE
PLATEFORME DE BIOSTATISTIQUE

Encore besoin d'R ?

SÉBASTIEN DÉJEAN(*), THIBAULT LAURENT(+)

(*) Institut de Mathématiques de Toulouse, Sebastien.Dejean@math.univ-toulouse.fr
(+) Toulouse School of Economics, Thibault.Laurent@univ-tlse1.fr

21 juin 2016
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

Chapitre 1

Introduction

1.1 Principe du document

Ce document fait suite à celui destiné à une initiation à R *Pour se donner un peu d'R* disponible à l'adresse `math.univ-toulouse.fr/~sdejean`. Il en reprend la structure avec des commandes à saisir, des commentaires pour attirer l'attention sur des points particuliers et quelques questions/réponses pour tester la compréhension des points vus précédemment. Pour certains points particuliers, nécessitant par exemple un environnement logiciel particulier, les faits ne sont que mentionnés et il n'y a pas toujours de mise en œuvre pratique.

1.2 Pré-requis

Même si la plupart des points abordés dans ce document ne sont pas très compliqués, ils relèvent d'une utilisation avancée de R et ne s'adressent donc pas au débutant en R. Avant d'utiliser ce document, le lecteur doit notamment savoir :

- se servir de l'aide en ligne de R ;
- manipuler les objets de base de R : vecteur, matrice, liste, *data.frame* ;
- programmer une fonction élémentaire.

Chapitre 2

Manipulation de données particulières

2.1 Chaînes de caractères

```
R > phrase = "je manipule les caracteres"
R > nchar(phrase)
R > substr(phrase,start=1,stop=8)
R > strsplit(phrase,"p")
R > strsplit(phrase,"les")
R > mots = strsplit(phrase," ")
R > mots = unlist(mots)
R > lettres = strsplit(phrase,NULL)
R > length(lettres[[1]])
R > toupper(phrase)
R > tolower("AAA")
R > res = grep("le",mots)
R > mots[res]
R > grep("des",mots)
R > agrep("des",mots)
R > sub("je","il",phrase)
```

- R dispose de fonctions adaptées à la manipulation de chaînes de caractères
- Attention à la gestion des espaces
- Noter l'utilisation de `unlist`
- L'utilisation d'expressions régulières est possible en R (voir `?regexp`).

Ordonnancement

On s'intéresse ici plus particulièrement aux règles d'ordonnancement utilisées pour la langue franÃğaise. Selon la langue utilisée par la machine, il existe donc des règles particulières pour ordonnancer les chaÃőnes de caractères, qui diffèrent d'une langue à l'autre. Pour vérifier la langue utilisée par la machine, on peut utiliser la commande

```
R > Sys.setlocale(category = "LC_CTYPE", locale = "")
```

Considérons la citation suivante stockée dans l'objet `citation`.

```
R > citation <- "Il est important que les étudiants portent
```

```
+ un regard neuf et irrévérencieux sur leurs études;
+ il ne doivent pas vénérer le savoir mais le remettre
+ en question (chapitre : 1 - paragraphe : 2 - ligne :
+ 10 - page : 185. Jacob Chanowski)."
```

On peut récupérer chaque "mot" (entités séparées par des espaces") par la commande :

```
R > mots <- unlist(strsplit(citation, " "))
```

Le tri des éléments (uniques) du vecteur `mots` nous montre les règles d'ordonnancement appliquées par R.

```
R > sort(unique(mots))
```

- les caractères spéciaux `-, :, ;, (`, etc. sont prioritaires sur les chiffres et les lettres.
- les chiffres sont prioritaires sur les lettres.
- les mots sont ordonnancés comme dans un dictionnaire franÃğais.
- quand il y a des lettres avec des accents, on ordonnance comme s'il n'y avait pas d'accents.
- les lettres majuscules sont insérées dans l'ordre alphabétique et ne sont pas prioritaires par rapport aux lettres minuscules.
- les chiffres ne sont pas regardés comme des chiffres mais comme une chaÃőne de caractères. Autrement dit "10" doit être vu comme un mot avec 2 caractères consécutifs : "1", puis "0". "2" doit être vu comme 1 mot avec un seul caractère. Pour comparer ces 2 mots, on compare les caractères entre eux les uns après les autres. Dans un premier temps, on regarde le 1er caractère de chaque mot : "1" est plus petit que "2". Aussi, quelque soit le nombre de caractère qu'on va ajouter après "1" ce mot sera plus petit que "2". Par exemple "15552525" sera plus petit que "2".

Questions

1. À partir du jeu de données `USArrests`, extraire les lignes dont le nom contient la chaîne de caractères "New".
2. Comparer les résultats obtenus avec les instructions suivantes :

```
R > USArrests[grep("C",rownames(USArrests)),]
```

```
R > USArrests[grep("^C",rownames(USArrests)),]
```

Réponses

1.
```
R > USArrests[grep("New",rownames(USArrests)),]
```
2. Dans le premier cas, tous les noms de lignes contenant la lettre "C" sont renvoyés ; dans le second cas, seuls ceux commençant par "C" sont renvoyés. Consulter la fiche d'aide sur les expressions régulières pour en savoir (beaucoup !) plus (`?regexp`).

2.2 Facteurs

```
R > genre <- sample(c("Ctrl","Trait"),
+ size=10,replace=TRUE)
R > genre
R > genre.fact = factor(genre)
R > genre.fact
R > genre[1] = "Autre"; genre
R > genre.fact[1] = "Autre"
R > genre.fact
R > levels(genre.fact)
R > levels(genre)
R > vec <- sample(1:4, size=20, rep=T)
R > f.vec <- factor(vec, levels=1:3,
+ labels = c("Rien","Peu","Beaucoup")
R > object.size(genre)
R > object.size(genre.fact)
R > mesures = rnorm(100)
R > codage = cut(mesures, breaks=-4:4)
R > codage
R > table(codage)
R > require("classInt")
R > codage2 = cut(mesures,
+ classIntervals(mesures,n=5,
+ style="kmeans")$brks)
```

- Même s'ils peuvent a priori ressembler à des chaînes de caractères, les *factor* ont un comportement différent.
- Les facteurs ont des modalités pré-définies. L'affectation d'une valeur différente de ces modalités pré-définies provoque un message d'avis et une valeur manquante dans le vecteur.
- Le stockage d'un *factor* est moins volumineux qu'un objet *character* "équivalent" car les *levels* ne sont stockés qu'une seule fois.
- La fonction `cut()` qui permet le recodage d'une variable quantitative en classes génère un objet de type *factor*. Dans le même contexte, la fonction `classIntervals()` du package *classInt* propose différentes méthodes de discrétisation d'une variable quantitative.

Remarque : les fonctions classiques d'importation des jeux de données codent les variables qualitatives sous forme de *factor*.

Questions

1. Facteur ordonné
   - Créer un vecteur de chaîne de caractères `doses` comprenant 25 éléments "Faible", "Moyenne" ou "Forte".
   - Convertir cet élément en `f.dose` de type *factor* ordonné (`?factor`).
   - Vérifier que les niveaux du facteur sont effectivement ordonnés
2. Codage et comptage : donner un équivalent de l'enchaînement des fonctions `cut()` et `table()` utilisées précédemment.

Réponses

1. Facteur ordonné

```
R > doses = sample(c("faible", "moyenne", "forte"),
+ size=25,replace=TRUE)
```

```
R > f.dose = factor(doses,levels=c("faible","moyenne",
+ "forte"),ordered=TRUE); f.dose
R > f.dose[1]<f.dose[2]
```

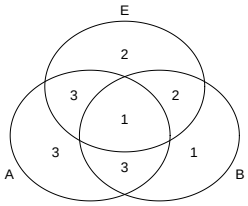
2. Codage et comptage Les 2 opérations consistent à calculer les informations permettant de représenter un histogramme. On peut donc les réaliser avec la fonction `hist()`.

```
R > res.hist = hist(mesures, breaks=-4:4)
```

2.3 Opérations ensemblistes

```
R > A=1:10
R > B=c(3:6,12,15,18)
R > union(A,B)
R > intersect(A,B)
R > setdiff(A,B)
R > setdiff(B,A)
R > is.element(2,A)
R > is.element(2,B)
R > is.element(A,B)
R > is.element(B,A)
R > let=letters[1:10]
R > union(A,let)
```

- Noter la différence entre les résultats des 2 utilisations de `setdiff`.
- Ces opérations nécessitent que les éléments passés en paramètres soient du même type. Voir le résultat de l'union de `A` et `let`.



Questions

1. Donner une notation équivalente à `is.element(2,y)`.
2. L'objet `letters` est un vecteur de longueur 26 qui contient les lettres de l'alphabet. Tester l'appartenance des lettres "k" et "m" à l'alphabet ; que renvoie la syntaxe "réciproque" (appartenance de l'alphabet à l'ensemble `c("k","m")`) ? Comment obtenir le rang des lettres "k" et "m" dans l'alphabet ?
3. En plus des 2 vecteurs A et B définis précédemment, considérons le vecteur E [1] contenant les valeurs 18,1,9,14,12, 6,2 et 19. Construire E et donner l'enchaînement des commandes qui permettent de retrouver les valeurs disposées sur le diagramme de Venn ci-dessus (obtenu avec la fonction `venn()` du package `gplots`).

Réponses

1. La rubrique `See Also` de l'aide de la fonction `is.element()` propose `%in%`. Donc une notation équivalente est `2 %in% y`.
2. Réponses en utilisant la fonction `%in%` :

```
R > c("k","m")%in%letters
```

1. nous évitons la lettre C qui existe déjà dans R, voir `?C`

`R > letters%in%c("k","m")` renvoie un vecteur VRAI/FAUX marquant l'appartenance pour chaque letter de l'alphabet à l'ensemble `c("k","m")`.

`R > which(letters%in%c("k","m"))` pour connaître la position des lettres en question dans l'alphabet.

3. Voici une possibilité (parmi d'autres).
   - Les intersections des ensembles pris 2 à 2

     `R > iAE=intersect(A,E)`

     `R > iBE=intersect(B,E)`

     `R > iAB=intersect(A,B)`
   - Les ensembles "au centre"

     `R > setdiff(iAE,iBE)`

     `R > intersect(iAE,iBE)`

     `R > setdiff(iBE,iAE)`
   - Les ensembles "en bas"

     `R > setdiff(setdiff(A,iAB),iAC)`

     `R > setdiff(iAB,intersect(iAC,iBC))`

     `R > setdiff(setdiff(B,iBC),iAB)`
   - L'ensemble "en haut"

     `R > setdiff(setdiff(C,iAC),iBC)`

2.4 Manipulation de bases de données

2.4.1 Jointure et agrégation

La fonction `merge()` permet de réaliser la jointure entre deux tables. Il est essentiel de préciser la clé de référence des deux tables avec l'option `by=` si les deux tables ont un nom de clé identique ou alors `by.x=` et `by.y=` si la clé de référence porte un nom différent selon la table. Par exemple, on a une table *patient* qui contient des informations sur les patients et une table *visite* qui contient des informations sur les visites. La clé de référence est le nom du patient qui ne porte pas le même nom dans les deux tables.

```
R > patient<-data.frame(
+ nom.famille=c("René","Jean","Ginette","Joseph"),
+ ddn=c("02/02/1925","03/03/1952","01/10/1992","02/02/1920"),
+ sexe=c("m","m","f","m"))
R > visite<-data.frame(
+ nom=c("René","Jean","René","Simone","Ginette"),
+ ddv=c("01/01/2013","10/12/2013","05/01/2014","04/12/2013",
+ "05/10/2012"))
```

Dans la première commande, le merge se fait sur les variables qui sont présentes dans les deux tables

```
R > merge(visite,patient,by.x="nom",by.y="nom.famille")
```

Si on souhaite garder toute l'information contenue dans le fichier visite, on ajoute l'option `all.x=TRUE`.

```
R > merge(visite,patient,by.x="nom",by.y="nom.famille",
+ all.x=TRUE)
```

Enfin, si on souhaite conserver l'information dans les deux tables, on utilise `all.x=TRUE` et `all.y=TRUE`

```
R > visite.patient<- merge(visite, patient, by.x="nom",
+ by.y="nom.famille", all.x=TRUE, all.y=TRUE)
```

Maintenant, on souhaite connaître l'âge des patients lors de leurs visites en fonction du sexe. D'abord, on calcule l'âge du patient (on verra plus de détails sur le format des dates dans le paragraphe suivant) :

```
R > visite.patient$age <-
+ round(as.numeric(as.Date(visite.patient$ddv,format="%d/%m/%Y")
+ - as.Date(visite.patient$ddn,format="%d/%m/%Y"))/365,0)
```

Pour faire l'agrégation, on utilise la fonction `aggregate()`. La variable qui permet d'aggréger doit être mise sous forme d'une liste dans l'option `by=`. L'option FUN= renseigne s'il s'agit de faire une somme, une moyenne, etc. Dans notre cas :

```
R > aggregate(visite.patient$age, by=list(S=visite.patient$sexe),
+ FUN=mean, na.rm=TRUE)
```

Remarque : la fonction `aggregate()` fait appel à la fonction `tapply()`

Questions

1. À partir du jeu de données iris, construire l'objet iris1 qui contient en fonction des espèces, la taille moyenne de la variable "Petal.Length".
2. Construire l'objet iris2 qui contient en fonction des espèces, la taille totale de la variable "Petal.Width".
3. Faire le merge des jeux de données `iris1` et `iris2`.

Réponses

1. iris1<-aggregate(iris[,"Petal.Length"], by=list(espece=iris$Species),FUN=mean)
2. iris2<-aggregate(iris[,"Petal.Width"], by=list(espece=iris$Species),FUN=sum)
3. merge(iris1,iris2,by="espece")

2.4.2 Package `dplyr`

Le package `dplyr` fournit un ensemble de fonctions permettant de manipuler des données stockées dans un `data.frame` de façon plus intuitive qu'avec les fonctions de base de R. Il propose des fonctions au nom assez explicite :

- `filter` permet de sélectionner un sous-ensemble de lignes
  `R > filter(iris, Species=="setosa"` ce qui est probablement plus explicite que
  `R > iris[iris$Species=="setosa",]` qui fournit cependant le même résultat.

– `arrange` ré-ordonne les lignes d'un `data.frame` selon une variable passée en paramètre (plusieurs variables permettant de gérer les ex-aequo)

```
R > arrange(iris, Sepal.Length)
R > arrange(iris, desc(Sepal.Length))
R > arrange(iris, Sepal.Length, Sepal.width)
```

La première commande ci-dessus étant équivalente à :

```
R > iris[order(iris$Sepal.Length),]
```

– `select` extrait certaines colonnes d'intérêt toujours selon une syntaxe très simple

```
R > select(iris, Sepal.Length, Species)
```

et plus lisible que

```
R > iris[,c("Sepal.Length","Species")]
```

– `mutate` ajoute des colonnes calculées à partir de colonnes existantes

```
R > iris.ratio <- mutate(iris,
+ Sepal.ratio = Sepal.Length/Sepal.Width,
+ Petal.ratio = Petal.Length/Petal.Width)
```

est équivalent à

```
R > iris.ratio <- data.frame(iris,
+ Sepal.ratio = iris$Sepal.Length/iris$Sepal.Width,
+ Petal.ratio = iris$Petal.Length/iris$Petal.Width)
```

– `sample_n` et `sample_frac` permettent de sélectionner aléatoirement un sous-ensemble de lignes soit à partir d'un nombre de lignes soit à partir d'une proportion.

```
R > sample_n(iris, 25)
R > sample_frac(iris, 0.1)
```

2.4.3 Gestion de données volumineuses

Avec `read.table()`

Une première astuce concernant la gestion de données volumineuses consiste à mieux gérer l'importation des données. Regardons ce que cela donne avec un fichier contenant 100000 lignes et 2 colonnes.

– Commençons par générer des données.

```
R > Donnees.a.importer <- data.frame(chiffre = 1:100000,
+ lettre=paste0("caract", 1:100000))
```

– Exportons ces données dans un fichier avec la fonction `write.table()`

```
R > write.table(Donnees.a.importer, "fichier.txt",
+ row.names=F)
```

On dispose ainsi d'un fichier appelé `fichier.txt` dans l'espace de travail.

– Importons ces données avec la fonction `read.table()` en mesurant le temps pris pour accomplir cette action.

```
R > system.time(
+ import1 <- read.table("fichier.txt", header=T))
utilisateur     syst\`eme      \'ecoul\'e
      0.912         0.004       0.926
```

– Exécutons la même opération en précisant la classe de chaque colonne.

```
R > system.time(
+ import2 <- read.table("fichier.txt", header=T,
+ colClasses=c("integer","character")))
utilisateur     syst\`eme      \'ecoul\'e
      0.152         0.000         0.160
```

Le gain de temps dans le second cas vient du fait que R n'a pas besoin de parcourir l'ensemble du fichier pour savoir sous quel type il va stocker les données.

Avant d'importer un fichier (de cette faÃğon ou d'une autre), l'utilisation de la fonction `readLines()` permet d'identifier le type des éléments à importer ainsi que les délimiteurs de champs.

```
R > readLines("fichier.txt", n = 10)
```

Quelques packages dédiés

Voici 3 packages, parmi d'autres, susceptibles d'aider l'utilisateur à manipuler des fichiers de données volumineux. Le package `Matrix` est plus particulièrement dédié à la manipulation de matrices creuses (*sparse*) c'est-à-dire dont la majorité des éléments sont nuls.

data.table : Extension of Data.frame

```
Fast aggregation of large data (e.g. 100GB in RAM),
fast ordered joins, fast add/modify/delete of columns
by group using no copies at all, list columns and a
fast file reader (fread). Offers a natural and flexible
syntax, for faster development...
```

ff : memory-efficient storage of large data on disk and fast access functions

```
The ff package provides data structures that are
stored on disk but behave (almost) as if they were
in RAM by transparently mapping only a section
(pagesize) in main memory - the effective virtual
memory consumption per ff object.
```

Matrix : Sparse and Dense Matrix Classes and Methods

```
Classes and methods for dense and sparse matrices
and operations on them using 'LAPACK' and 'SuiteSparse'.
```

2.4.4 Bases de données

R dispose de plusieurs packages permettant d'intéragir avec des systèmes de gestion de base de données : `RODBC`, `RMySQL`, `RPostgresSQL`, `RSQLite` ainsi qu'avec des bases de données orientées document comme MongoDB et couchDB via les packages `mongolite` et `couchDB`.

2.5 Dates et unités de temps

Il existe plusieurs packages pour manipuler des données temporelles ; voir la *Task View* **Time Series Analysis**, http://cran.r-project.org/web/views/TimeSeries.html). Nous nous contentons ici de présenter quelques manipulations élémentaires. Une référence sur le sujet : G. Grothendieck and T. Petzoldt (2004), R Help Desk : Date and Time Classes in R *R News 4(1)*, 29-32, http://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf. Le format POSIXlt ou POSIXct est plus précis que le format date car il mesure à la fois la date et l'heure. Ce format est notamment utilisé danes les séries temporelles d'indices boursiers.

```
R > date()
R > Sys.Date();Sys.time()
R > ?format.Date
R > weekdays(.leap.seconds)
R > months(.leap.seconds)
R > quarters(.leap.seconds)
R > as.Date(32768,
+ origin="1900-01-01")
R > (z = Sys.time())
R > format(z,"%a %d %b %Y %X %Z")
R > system.time(for(i in 1:100)
+ var(runif(100000)))
```

- voir également le package `chron`
- l'élément `.leap.seconds` contient les dates auxquelles une seconde intercalaire (http://fr.wikipedia.org/wiki/Seconde_intercalaire) a été ajouté au *Temps universel coordonné, UTC*.
- il existe une classe pour la manipulation de séries temporelles (voir plus loin).

N.B. La fonction `system.time()` renvoie plusieurs informations concernant le temps de calcul d'une commande :

- Utilisateur : il s'agit du temps mis par l'ordinateur pour exécuter directement le code donné par l'utilisateur
- Systême : il s'agit du temps utilisé pas directement par le calcul, mais par le système (ex : gestion des entrées/sorties, écriture sur le disque, etc.) lié au code qui doit être exécuté.
- écoulé : il s'agit du temps Utilisateur + Systême. C'est en général ce dernier qui est utilisé pour comparer des temps de calcul.

Séries temporelles

Nous faisons ici un aparte pour évoquer la manipulation de séries temporelles.

```
R > set.seed(493)
R > x=2+round(rnorm(15),3)
R > x.ts=ts(x,st=c(2005,2),freq=12)
R > cbind(x.ts, lag(x.ts,-1))
R > plot(x.ts)
R > require("zoo")
R > date.x = c("02/27/92", "03/27/92",
+ "01/14/92", "02/28/92", "02/01/92")
R > date.x=strptime(date.x,"%m/%d/%y")
R > x=zoo(cbind(rnorm(4), rnorm(4)+5),
+ date.x)
R > plot(x,col=c("blue","red"))
```

- `set.seed()` permet de fixer la graine du générateur aléatoire. On peut ainsi re-générer plus tard les mêmes valeurs sans les stocker.
- `?zoo ; ?strptime`...
- observer le changement de classe de l'objet `date.x` avant et après conversion par la fonction `strptime()`.

Questions

1. Quel jour (lundi, mardi ... ?) sera le 1er janvier de l'année 2015 ?
2. Combien de jours nous séparent du 31 décembre de l'année en cours ?

Réponses

1. `R > weekdays(as.Date("2015-01-01"))`
2. En 2 étapes pour faire les choses lisiblement :

```
R > aujourdhui = Sys.Date()
R > as.Date("2013-12-31")-aujourdhui
```

2.6 Répertoires et fichiers

```
R > dir();getwd()
R > file.info(dir())
R > R.home()
R > fic=dir(file.path(R.home(),
+ "bin"),full=T)
R > file.access(fic, 1)
R > fic[file.access(fic, 0) == 0]
R > fi = dir(file.path(R.home()))
R > fi[file.access(fic, 0) == 0]
R > fi[file.access(fic, 1) == 0]
R > fi[file.access(fic, 2) == 0]
R > fi[file.access(fic, 4) == 0]
R > dir.create("Sorties")
R > dir.exists("Sorties")
```

- Noter bien la différence entre `dir()` et `ls()`.
- Essayer quelques options de la fonction `dir()`.
- L'utilisation de la plupart de ces fonctions est intéressante lors de l'écriture de scripts "propres" qui stockeront les résultats dans des fichiers bien rangés dans des répertoires bien nommés.
- Noter les différents résultats de la fonction `file.access()`.

Questions

1. Écrire une fonction qui, à partir d'un nombre entier n passé en paramètre, effectue un tirage aléatoire de n valeurs selon une loi normale $\mathcal{N}(0,1)$ puis renvoie les valeurs générées dans un fichier d'un répertoire `Num` et trace une boxplot de ces données qui sera stockée dans un fichier jpg du répertoire `Graph`.

Réponses

1. Voici un exemple de fonction répondant à la question posée :

```
R > MaFonction = function(n){
+ out = rnorm(n)
+ dir.create("Num")
+ dir.create("Graph")
+ jpeg("Graph/box.jpg")
+ boxplot(out)
+ dev.off()
+ sink("Num/valeurs.txt")
+ print(out)
+ sink() }
```

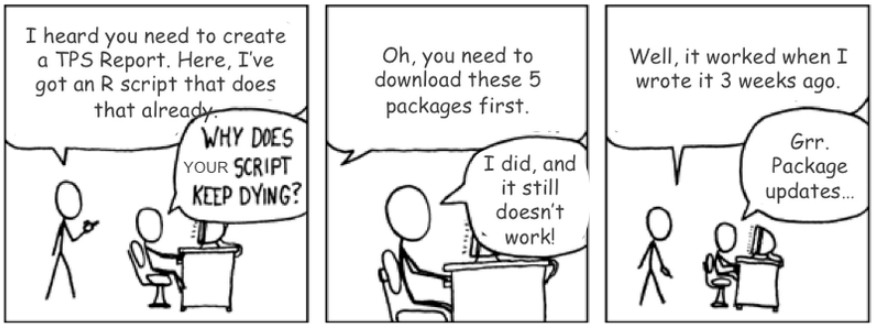
`R > MaFonction(100)` pour utiliser la fonction

2.7 Autour de l'espace de travail

```
R > search()
R > library(foreign)
R > search()
R > ls(pos=2)
R > library(pls)
R > search()
R > find("loadings")
R > ?loadings
R > ?stats::loadings
R > airquality
R > airquality$Ozone
R > attach(airquality)
R > search()
R > Ozone
R > Ozone = 1.2*Ozone
R > ls(); find("Ozone")
R > detach()
```

- La fonction `search()` donne la liste des *packages* (mais pas seulement) attachés à l'espace de travail.
- L'option `pos` de la fonction `ls()` permet de lister le contenu d'un package particulier en donnant sa position dans la liste renvoyée par `search()`.
- Certains packages peuvent avoir des fonctions portant le même nom ; d'où un message d'avertissement indiquant qu'un objet est masqué dans un package situé plus loin dans la liste `search()`.
- La fonction `attach()` permet d'attacher un *data.frame* ou un espace de travail sauvegardé ailleurs en `.RData`. Ce dernier cas peut être utile pour attacher un fichier `MesFonctions.RData` dans lequel on a sauvegardé des fonctions testées, approuvées et souvent utilisées.

Remarque : Il est parfois utile de connaître certains détails de l'environnement de travail notamment pour comprendre pourquoi *ça ne marche pas ! ! !* Les fonctions `sessionInfo()` et `R.Version()` peuvent permettre d'identifier certaines incompatibilités entre notre version de R et un package particulier que l'on vient d'installer.



Questions

1. Quel est l'inconvénient illustré par les manipulations de la variable *Ozone* du jeu de données `airquality` ?
2. À quoi sert l'opérateur `::` ?

Réponses

1. En attachant le jeu de données `airquality`, on peut atteindre ses variables sans nommer le jeu de données, mais il faut être conscient du fait qu'en voulant modifier une variable (ici par la commande `Ozone = 1.2*Ozone`), on crée une variable `Ozone` dans l'espace de travail courant `".GlobalEnv"` sans toucher au jeu de données. Il existe ainsi 2 objets `Ozone` comme le confirme la commande `find("Ozone")`.
2. `R > help("::")` cet opérateur permet d'accéder à un objet masqué par un autre du même nom dans un autre package.

Chapitre 3

Programmation

3.1 Structures de contrôle

3.1.1 `switch`

Les structures de contrôle classiques (`for, while, repeat, if else`) sont bien entendu disponibles dans R (vori le document *Pour se donner un peu d'R*. Nous nous intéressons ici à la notion d'aiguillage (`switch`) moins couramment utilisée bien que très pratique. L'exemple ci-dessous est extrait de l'aide en ligne de la fonction `switch()`.

```
R > centre <- function(x, type) {
+ switch(type,
+ mean = mean(x),
+ median = median(x),
+ trimmed = mean(x, trim = .1))
+ }
```

La fonction `centre()` effectuera un traitement différent selon la valeur passée au paramètre `type`.

```
R > donnees <- rnorm(10)
R > centre(donnees, "mean")
R > centre(donnees, "median")
R > centre(donnees, "trimmed")
```

Une chaÃőne de caractères non prévues ne renverra aucun résultat (et aucun message d'erreur).

```
R > centre(donnees, "moyenne")
```

L'aiguillage peut également se faire par un entier indiquant le numéro de l'alternative à choisir.

```
R > centre(donnees, 1);centre(donnees, 2);centre(donnees,3)
```

Dans cet exemple, une valeur au-delà de 3 produira le même effet qu'une chaÃőne de caractère non reconnue.

```
R > centre(donnees, 4)
```

On peut également définir une alternative par défaut.

```
R > centre <- function(x, type) {
```

```
+ switch(type,
+ mean = mean(x),
+ median = median(x),
+ trimmed = mean(x, trim = .1),
+ message("Mauvais choix"))
+ }
```

R > `centre(donnees, "moyenne");centre(donnees, 4)` renvoient le message `Mauvais choix`.

3.1.2 `stopifnot`

la fonction `stopifnot()` permet de tester simultanément plusieurs conditions. Elle est particulièrement utile à l'intérieur d'une fonction pour, par exemple, vérifier la conformité des valeurs passées en paramètres.

R > `stopifnot(1<2, length(1:2)==2, pi<2, cos(pi)>3)`

Si plusieurs conditions ne sont pas respectées, c'est la première non respectée qui est renvoyée comme source de l'erreur.

R > `stopifnot(1<2, length(1:2)==2, pi<2, cos(pi)>3)`

Questions

1. Modifier la fonction `centre()` définie précédemment pour vérifier que le paramètre `x` est numérique et que `type` est une chaÃőne de caractères (on supprimera la possibilité de passer un entier pour choisir le type).

Réponses

1. Il faut ajouter la ligne

   R > `stopifnot(is.numeric(x), is.character(type))` Les appels suivants à la fonction `centre()` renvoient une erreur en précisant quelle condition n'est pas respectée.

   R > `centre("toto", "mean"); centre(1:10, 3)`

   ```
   Erreur : is.numeric(x) n'est pas TRUE
   Erreur : is.character(type) n'est pas TRUE
   ```

3.2 Éviter les boucles

```
R > tab <- array(1:24,dim=2:4)
R > res <- apply(tab,1,sum)
R > res <- apply(tab,1:2,sum)
R > res <- apply(tab,3,function(x)
+ runif(max(x)))
R > tapply(mesures,codage,mean)
R > lapply(res,mean)
R > sapply(res,mean)
R > lapply(res,quantile)
R > sapply(res,quantile)
R > vec = replicate(500,
+ mean(rnorm(10)))
R > boxplot(vec)
R > rep(x=1:4,times=4:1)
R > mapply(rep,x=1:4,times=4:1)
R > x <- matrix(runif(10e6),nc=5)
R > system.time(apply(x,2,mean))
R > system.time(colMeans(x))
```

- Noter que le résultat renvoyé par `apply()` peut avoir différentes formes
- Les fonctions `rowSums()`, `colSums()`, `rowMeans()` et `colMeans()` sont plus rapides que les commandes équivalentes utilisant `apply()`.
- De même, la fonction `rowsum()` sera plus rapide qu'une alternative utilisant `tapply()`.
- La fonction `tapply()` est illustrée en utilisant les objets créés en 2.2. Voir aussi la fonction `by()`.
- Quelle est la différence entre `lapply()` et `sapply()` ?
- Pour se convaincre de l'intérêt d'éviter les boucles, voir l'exercice ci-dessous.
- En évitant les boucles, on gagne en temps de calcul (plus ou moins selon la taille du problème) et (raisonnablement) en lisibilité.

Exercice

- Définir les deux fonctions f1 et f2 :

```
f1<-function(n,p)
{
vec<-matrix(0,n,1)
for(i in 1:n)
{vec[i]<-mean(rnorm(p))}
 return(vec)
}
```

```
f2<-function(n,p)
{
vec<-replicate(n,mean(rnorm(p)))
return(vec)
}
```

- Comparer les temps de calcul des 2 fonctions

```
R > system.time(f1(50000,500))
R > system.time(f2(50000,500))
```

3.3 Fonctions

Lorsqu'un utilisateur définit une fonction, il peut permettre à sa fonction d'utiliser tous les options d'une autre fonction sans les lister une à une. Prenons l'exemple de la fonction `plotlm()` définie ci-dessous.

```
plotlm = function(x, y, np=TRUE,...)
```

```
{
plot(y~x,...)
abline(lm(y~x),col="blue")
 if(np)
 {
  np.reg=loess(y~x)
  x.seq=seq(min(x),max(x),length.out=25)
  lines(x.seq, predict(np.reg,x.seq), col="red")
 }
}
```

L'utilisation de la syntaxe "`...`" permet de préciser que la fonction `plotlm()` pourra, si besoin, faire appel à n'importe quelle option de la fonction `plot()`.

Pour illustrer la chose, exécuter les instructions suivantes.

```
R > x = rnorm(100); y = runif(100)
R > plotlm(x,y)
R > plotlm(x,y,np=FALSE)
R > plotlm(x,y, pch=16, col="pink",
+ xlab="variable explicative", ylab="variable à expliquer")
```

Questions

1. Que fait la fonction `plotlm()` ?
2. Écrire une fonction `simul()` qui prend en argument d'entrée un nombre `n`, un nombre `B` et `...` qui correspondra aux options utilisées dans la fonction `hist()`. Cette fonction devra mettre en œuvre l'algorithme ci-dessous :
   - Initialisation : res=0 et extremes=NULL
   - Début boucle : on répète B fois l'opération suivante :
     - simulation d'un vecteur x de taille n selon une N(0,1)
     - est-ce qu'il existe au moins un élément de x supérieur en valeur absolue à 1.96. Si oui, on ajoute à chaque 1 unité à res et on stocke dans extremes les valeurs concernées.
   - Fin boucle
   - On comptabilise sur les B boucles le pourcentages de boucles où le phénomène s'est produit.
   - On représente l'histogramme des valeurs extrêmes.

Réponses

1. La fonction `plotlm()` basée sur la fonction `plot()` permet de représenter un nuage de points (y en fonction de x), de tracer une droite de régression (fonction `abline()`) et d'ajouter, selon le souhait de l'utilisateur, un autre modèle de régression non paramétrique (fonctions `loess()` et `lines()`).
2. Voici une fonction répondant à la question posée :

```
simul <-function(n,B,...)
 {
```

```
# initialisation : le ph\'enom\'ene s'est produit 0 fois
 res=0
 extremes=NULL
  for (b in 1:B)
   {
# simulation d'un vecteur x de taille n
   x<-rnorm(n)
     if (any(abs(x)>1.96))
     {
# on ajoute 1 \'a chaque fois que la condition est v\'erifi\'ee
     res<-res+1
     extremes<-c(extremes,x[abs(x)>1.96])
 }
   }
   print(paste("Le ph\'enom\'ene s'est produit",
   res/B*100,"% de fois"))
  hist(extremes,...)
 }  # fin de la fonction
```

Exemple d'utilisation :

```
R > simul(50,100,nclass=20,
+ xlab="représentation des valeurs extrêmes")
```

3.4 Le mode non-interactif (*BATCH*)

Créer le fichier `MonScript.R` contenant les lignes suivantes :

```
2+2
jpeg("mon_graphique.jpg")
plot(rnorm(10))
dev.off()
3+3
```

```
R > ?BATCH
```

À partir d'un invite de commandes, on demande l'exécution du script à l'aide de la commande :

```
C:\> "C:\Program Files\R\R-2.12.2\bin\R" CMD BATCH MonScript.R
```

depuis le répertoire courant où le fichier `MonScript.R` se trouve, ou par la commande

```
C:\> : R CMD BATCH D:\Formation\R\script.R
```

depuis le répertoire d'installation de R (que l'on peut connaître grâce à la fonction `R.home()`.

3.5 Débugger une fonction

R dispose d'outils élémentaires de *débugage*. Pour illustrer leur fonctionnement, définissons et utilisons la fonction `f3()` appelant les fonctions `f1()` et `f2()` définies dans la section 3.2.

```
f3 = function(n,p)
{
n2 = as.character(n)
length(which(f1(n,p)>f2(n2,p)))
return(p3)
}
```

```
R > f3(5,10)
R > traceback()
R > debug(f3)
R > f3(5,10)
R > undebug(f3)
```

- à quoi peut servir cette fonction ?
- pourquoi va-t-elle poser problème ?
- le message d'erreur renvoyé par R est-il explicite ?
- `traceback()` renvoie les différentes instructions qui ont été exécutées jusqu'à l'erreur (`?traceback`)
- la fonction `debug(f3)` permet de suivre pas à pas l'exécution de la commande suivante. Ne pas oublier de "fermer" le débugage avec `undebug()`

Pour illustrer différemment l'utilisation de la fonction `debug()`, définissons la fonction

```
f4 = function(x)
{x=x+1
 x=x^3
 x=log(x)
 return(x)
}
```

et utilisons la

```
R > f4(5)
R > f4(-5)
R > debug(f4)
R > f4(-5)
R > undebug(f4)
```

En exécutant pas à pas la fonction `f4()`, on peut demander à chaque étape la valeur que prend la variable `x` en cours d'exécution ; il faut pour cela saisir la variable `x` devant le prompt du débugger (`Browse[2]>`). Noter que l'utilisation de `traceback()` dans ce cas n'apporte rien vu que le problème signalé par R est un avertissement (*Warning*) et pas un message d'erreur.

3.6 Écriture de script

Que pensez-vous de cette ligne de commande ? Est-elle raisonnablement lisible ?

```
R > plot(hclust(dist(USArrests,"manhattan"),"ward")
+ ,hang=-1,m="Mon dendrogramme",s="",xl="Etats")
```

Comment rendre les mêmes commandes plus lisibles dans un script ? Voyons plusieurs étapes.

1. Nommer les arguments de chaque fonction

```
R > plot(hclust(dist(USArrests,method="manhattan"),
+ method="ward")
+ ,hang=-1,main="Mon dendrogramme",sub="",xlabel="Etats")
```

2. Décomposer la ligne de commande

```
R > mon.resultat.dist = dist(USArrests,method="manhattan")
R > mon.resultat.hclust = hclust(mon.resultat.dist,
+ method="ward")
R > plot(mon.resultat.hclust,hang=-1,
+ main="Mon dendrogramme",sub="",xlabel="Etats")
```

3. Isoler la déclaration de certains paramètres

```
R > choix.methode.dist = "manhattan"
R > choix.methode.hclust = "ward"
R > titre = "Mon dendrogramme"
R > sous.titre = ""
R > label.axe.x = "Etats"
R > mon.resultat.dist = dist(USArrests,
+ method=choix.methode.dist)
R > mon.resultat.hclust = hclust(mon.resultat.dist,
+ method=choix.methode.hclust)
R > plot(mon.resultat.hclust,hang=-1,
+ main=titre,sub=sous.titre,xlabel=label.axe.x)
```

   Ces lignes peuvent être enregistrées dans un fichier `MonScriptClassif.R` qui peut être lu par R grâce à la fonction `source()`.

```
R > source("MonScriptClassif.R")
```

3.7 Création de package

La partie la plus délicate, en tout cas celle qui demande probablement le plus de temps, est la rédaction des fiches d'aide. Pour le "reste", à partir du moment où les fonctions existent et fonctionnent, la fonction `package.skeleton()` fera la plus grande partie du travail de mise en forme des différents fichiers et répertoires nécessaires.

```
R > help(package.skeleton)
R > help(prompt)
```

Avant d'être soumis au CRAN, un package doit être vérifié. Cela se fait en utilisant la commande `R CMD check MonPackage` dans une console. Sous Windows, cela nécessite l'installation de plusieurs programmes réunis dans un fichier exécutable à l'adresse

http://cran.r-project.org/bin/windows/Rtools/. Cet exécutable installe l'environement Linux CYGWIN ainsi que plusieurs compilateurs (C, C++, Fortran...). Pour plus d'informations, se référer au document *Writing R extensions* disponible par exemple à l'adresse http://cran.r-project.org/doc/manuals/r-release/R-exts.pdf.

3.8 Pour aller plus loin

3.8.1 C dans l'R

Nous présentons ici de façon succincte les différentes étapes permettant d'utiliser un sous-programme écrit en C à partir d'une fonction R. Pour des explications plus détaillées (et plus techniques !), on peut se référer au document *Writing R extensions*. Le recours à une fonction C peut permettre des gains en temps de calcul considérables.

1. Fichier convolve.c

   ```
   void convolve(double *a, int *na, double *b, int *nb, double *ab){
     int i, j, nab = *na + *nb - 1;
     for(i = 0; i < nab; i++) ab[i] = 0.0;
     for(i = 0; i < *na; i++);
      for(j = 0; j < *nb; j++) ab[i + j] += a[i] * b[j];}
   ```

   Bien vérifier que le copier-coller a bien fonctionné ; certains caractères spéciaux pouvant poser problème.

2. Création d'une librairie dynamique (Unix, `.o` et `.so`, Windows, `.dll`) : `R CMD SHLIB convolve.c`

3. Création d'une fonction R qui fait appel à la librairie (ce n'est pas obligatoire mais cela facilite la lecture)

   ```
   conv.avec.C = function(a, b)
     .C("convolve", as.double(a), as.integer(length(a)), as.double(b),
        as.integer(length(b)), ab = double(length(a) + length(b) - 1))$ab
   ```

4. Chargement de la librairie dynamique dans R :

   ```
   R > dyn.load("convolve.so") # sous Unix/Linux
   ```

   ```
   R > dyn.load("convolve.dll") # sous Windows
   ```

5. Utilisation :

   ```
   R > res.C = conv.avec.C(1:10,seq(0,1,l=10))
   ```

Définissons également une fonction équivalente n'utilisant pas de C

```
conv.que.R<-function(a,b)
{na=length(a)
 nb=length(b)
 nab=na+nb-1
 ab<-matrix(0,1,nab)
```

```
 for (i in 1:na)
 {for( j in 1:nb)
  {ab[i+j-1]=ab[i+j-1] + a[i]*b[j]
  }
 }
return(ab)
}
```

Son utilisation est classique :

```
R > res.R = conv.que.R(1:10,seq(0,1,l=10))
```

La comparaison montre que les résultats sont les mêmes (aux précisions des 2 systèmes près)

```
R > plot(res.C, res.R);res.C-res.R;all(res.C==res.R)
```

En revanche les temps de calcul penchent nettement en faveur de la version appelant du code C.

```
R > system.time(
+ conv.avec.C(1:500,seq(0,500,l=500)))
R > system.time(
+ conv.que.R(1:500,seq(0,500,l=500)))
```

Le codage en C (ou en un autre langage compilé) d'une partie d'un code R est une alternative très intéressante lorsque l'utilisation de boucles est inévitable.

3.8.2 R parallèle

Un coup d'œil à la *Task View* **High-Performance and Parallel Computing with R** à l'adresse `http://cran.r-project.org/web/views/HighPerformanceComputing.html` vous montrera l'étendue des développements liés au calcul intensif sous R. Nous nous contentons de présenter ici une utilisation relativement simple d'un package permettant de paralléliser certaines parties d'un calcul.

Reprenons la fonction `conv.que.R()` précédemment définie et exécutons la deux fois (en mesurant le temps écoulé).

```
R > system.time(
+ for(i in 1:2)
+ conv.que.R(sample(1:500),sample(seq(0,500,l=500))))
```

Selon la machine dont on dispose, on va solliciter la participation de plusieurs processeurs pour effectuer les calculs. Sous Windows, pour connaître le nombre de processeurs, faire par exemple un clic droit dans la barre des tâches, à côté de l'horloge, et ouvrir le "gestionnaire des tâches". Aller ensuite dans l'onglet Performance. Le nombre de processeurs correspond au nombre de cadres ouverts dans la partie supérieure. Dans la suite, supposons que l'on souhaite utiliser 2 processeurs.

```
R > library(snowfall)
R > nb.cpus=2
```

La fonction `conv.que.R` nécessite quelques petits ajustements pour devenir `conv.paral()` définie ainsi :

```
conv.paral<-function(x,a,b)
{
 a=sample(a); b=sample(b)
 na=length(a)
 nb=length(b)
 nab=na+nb-1
 ab<-matrix(0,1,nab)
 for (i in 1:na)
 {for( j in 1:nb)
  {ab[i+j-1]=ab[i+j-1] + a[i]*b[j]
  }
 }
return(ab)
}
```

- Initialisation des paramètres de la parallélisation
  `R > sfInit(parallel=TRUE, cpus=nb.cpus)`
- exécution en parallèle avec mesure du temps écoulé
  ```
  R > system.time(result <-
  + sfClusterApplyLB(1:2,conv.paral,
  + a=1:500,b=seq(0,500,l=500)))
  ```
- fin de l'exécution parallèle
  `R > sfStop()`

3.8.3 Classes

Sans rentrer dans les subtilités liées aux classes dans R, nous évoquons ici quelques facilités d'utilisation du langage liées à l'existence de ces classes.

Par exemple, effectuons une Analyse en Composantes Principales avec la fonction `prcomp()`.

`R > res.acp = prcomp(USArrests)`

L'objet créé `res.acp` est un objet de classe `prcomp`. Pour s'en convaincre :

`R > class(res.acp)` et pour avoir un aperçu de ce que contient cet objet, on peut utiliser la fonction `str()`

`R > str(res.acp)`

On peut ensuite constater que l'utilisation de la fonction `plot()` sur l'objet en question renverra un graphique particulier, un éboulis des valeurs propres, adapté à l'interprétation des résultats d'une ACP.

`R > plot(res.acp)`

Ce comportement particulier de la fonction `plot()` est liée à l'existence d'une fonction `plot.prcomp()` qui est automatiquement appelée à la place de la fonction de base `plot()` lorsque l'objet passé en paramètre est de classe `prcomp`. Voir également les différents graphiques obtenus par

`R > plot(Sepal.Length ~ Petal.Length, data=iris)`

`R > plot(Sepal.Length ~ Species, data=iris)`

C'est également le cas pour la fonction `summary()` qui donnera pour un objet de la classe `prcomp`, les variances et parts de variance expliquées pour chaque composante principale.

```
R > summary(res.acp)
```

Classes S3 / S4

Souvent, les classes d'objet sont au format "list" et pour appeler des éléments de cet objet, on utilise le symbole $, par exemple :

```
R > res.acp$center
```

Lorsque les classes d'objet sont récentes, il s'agit de classes d'objet de la famille S4 (celles évoquées précédemment sont S3). Pour ces objets, on utilise généralement le symbole @ pour récupérer un élément. Voici un exemple d'objet de classe S4 :

```
R > require("sp")
```

```
R > data("meuse")
```
Pour le moment, l'objet `meuse` est un data.frame.

```
R > class(meuse)
```

```
R > coordinates(meuse) <- ~ x+y
```
On signale ici que `meuse` est un objet *spatial* et que ses coordonnées correspondent aux variables x et y.

```
R > class(meuse);str(meuse)
```

```
R > meuse$data
```
renvoie `NULL` car la classe `sp` est de type S4

```
R > meuse@data
```
renvoie effectivement les données du composant `data` de l'objet `meuse`.

Un utilisateur peut être tenté de construire ses propres classes d'objet si :

- il travaille sur des objets constitués de plusieurs éléments, qui sont toujours les mêmes quelles que soient les expériences réalisées ;
- il souhaite appliquer systématiquement les mêmes opérations sur ces objets (résumé, représentation, etc.)

Par exemple, un objet spatial est toujours constitué de :

- coordonnées spatiales des unités statistiques,
- référentiel géographique,
- données associées à chaque unité spatiale,
- ...

Et on souhaite faire sur ces données spatiales :

- une représentation cartographique,
- de l'agrégation de données,
- des changements de repères,
- ...

Chapitre 4

Graphique

4.1 Disposition de graphiques

```
R > x = runif(10); y = runif(10)
R > mat=matrix(c(2,0,1,3),2,2,byrow=TRUE)
R > nf = layout(mat, c(3,1), c(1,3), TRUE)
R > layout.show(nf)
R > plot(x,y,pch=16,col=rainbow(10),cex=2)
R > plot(x,rep(1,10),pch=16,col=rainbow(10),cex=2)
R > plot(rep(1,10),y,pch=16,col=rainbow(10),cex=2)
```

Questions

1. représenter le nuage de points de y en fonction de x, avec les boxplots associés (en bas horizontalement pour x et à gauche verticalement pour y)

   ```
   R > x=rnorm(500)
   R > y=rf(500,5,5)
   ```

2. Consulter l'aide en ligne de la fonction `layout()`. Expliquer les différentes étapes de l'exemple *Create a scatterplot with marginal histograms*.

Réponses

1. ```
   R > mat=matrix(c(2,1,0,3),2,2,byrow=TRUE)
   R > nf = layout(mat, c(1,3), c(3,1), TRUE)
   R > layout.show(nf)
   R > plot(x,y,pch=16)
   R > boxplot(y)
   R > boxplot(x,horizontal=T)
   ```

2. Le point particulier consiste à voir que la fonction `hist()` ne permet pas de représenter un histogramme horizontalement. `hist()` est donc utilisée sans produire de graphique (option `plot=FALSE`) et les sorties numériques de `hist()` sont passées à la fonction `barplot()` qui dispose de l'option `horiz`.
   ```

Recadrer un graphique

Plusieurs options de la fonction `par()` permettent de recadrer un graphique ; en voici une illustration.

Exemple 1 d'une représentation de 2 séries temporelles sans mettre d'option sur les marges

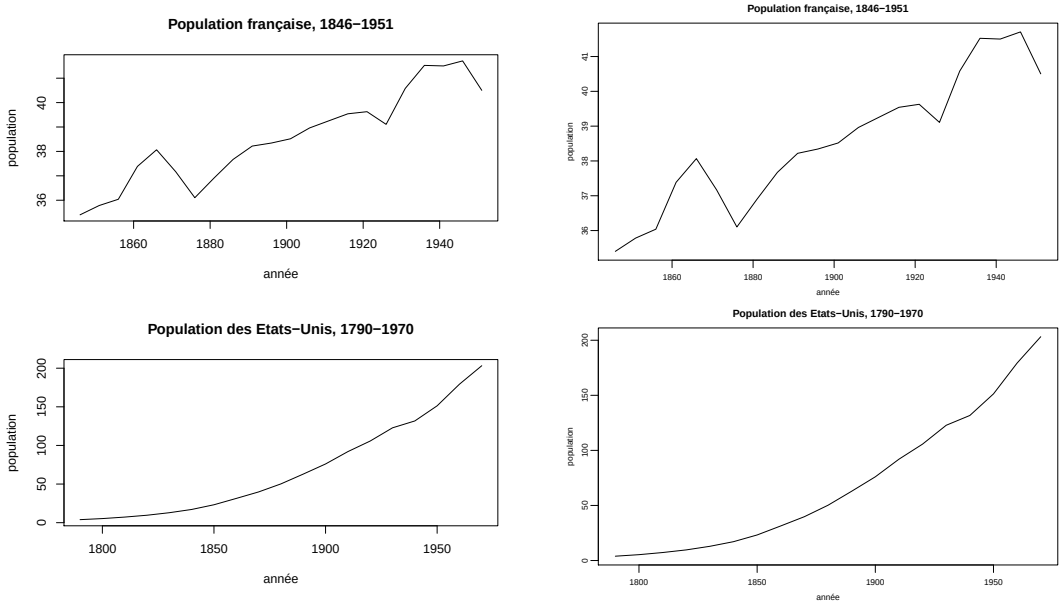
```
R > require("caschrono")
R > data("popfr")
R > op=par(mfrow=c(2,1))
R > plot.ts(popfr,xlab="année", ylab="population",
+ main ="Population française, 1846-1951")
R > plot.ts(uspop,xlab="année", ylab="population",
+ main ="Population des Etats-Unis, 1790-1970")
R > par(op)
```

Le résultat n'est pas satisfaisant dans la mesure où les titres prennent beaucoup de place et les graphiques semblent écrasés...

Pour remédier à cela, on peut utiliser les options `mar`, `mgp` et `oma` de la fonction `par()`. Voir l'aide de la fonction pour voir sur quelles parties de la fenêtre ces paramètres agissent exactement.

```
R > op=par(mfrow=c(2,1),mar=c(2.5,2.5,2,2),mgp=c(1.5,.5,0),
+ oma=c(0,0,0,0),cex.main=.8,cex.lab=.7,cex.axis=.7)
R > plot.ts(popfr,xlab="année", ylab="population",
+ main ="Population française, 1846-1951")
R > plot.ts(uspop,xlab="année", ylab="population",
+ main ="Population des Etats-Unis, 1790-1970")
R > par(op)
```

Voici les graphiques obtenus dans les 2 cas.



4.2 Axes et marges d'un graphique

```
R > x = runif(50); y = runif(50)
R > plot(x,y)
R > plot(x,y,axes=FALSE)
R > ?axis
R > axis(1)
R > axis(1,at=c(0.2,0.5,0.8),padj=1,
+ label=c("bas","milieu","haut"))
R > axis(2,lty=2,col=2)
R > mtext(c("un","deux","trois",
+ "quatre","cinq"),side=2,
+ at=seq(0.2,1,by=0.2))
R > plot(x,y)
R > rug(x)
R > rug(y,side=4)
R > par(mfrow=c(2,2))
R > plot(x,y, las=0, main="las=0",
+ sub="Parallèle aux axes")
R > plot(x,y, las=1, main="las=1",]
+ sub="Horizontal")
R > plot(x,y, las=2, main="las=2",
+ sub="Perpendiculaire aux axes")
R > plot(x,y, las=3, main="las=3",
+ sub="Vertical")
```

- noter que les axes peuvent être construits a posteriori.
- la fonction `axis()` ajoute au graphique existant et donc à l'axe existant. Il faut donc reconstruire le graphique (avec `plot()`) si on souhaite modifier un axe.
- la fonction `rug()` n'utilise pas vraiment les marges.
- `mtext()` permet d'écrire dans les marges d'un graphique.
- la position des étiquettes sur les axes peut Ãłtre modifiée par l'option graphique `las`. Cela est très pratique pour améliorer la lisibilité notamment avec des noms relativement longs.
- plus généralement, consulter l'aide de la fonction `par()` permet de découvrir régulièrement de nouveaux trucs.

Questions

On souhaite représenter deux séries de données v1 et v2 définies par :

```
R > v1 = 1:10
R > v2 = 100*sample(v1)
```

1. Quel est le problème ?
2. Utiliser l'option `new` de la fonction `par()` pour représenter les 2 vecteurs ; v1 étant repéré sur un axe des ordonnées à gauche et v2 sur un axe vertical à droite.

Réponses

1. Le problème vient du fait que les 2 séries de données contiennent des valeurs n'ayant pas le même ordre de grandeur. Il est difficile dans ce cas de trouver une échelle pertinente pour représenter simultanément les 2 séries.
2. Voici un enchaînement de commandes permettant de répondre à la question posée :
   `R > plot(v1,pch=16,cex=2)` représentation du vecteur `v1`

`R > par(new=TRUE)` permet au graphique suivant de se superposer à l'existant

`R > plot(v2,pch=16,col="red",axes=FALSE,ylab="",cex=2)` l'option importante ici est `axes=FALSE` qui permet de ne pas tracer d'axes pour ce nouveau graphique

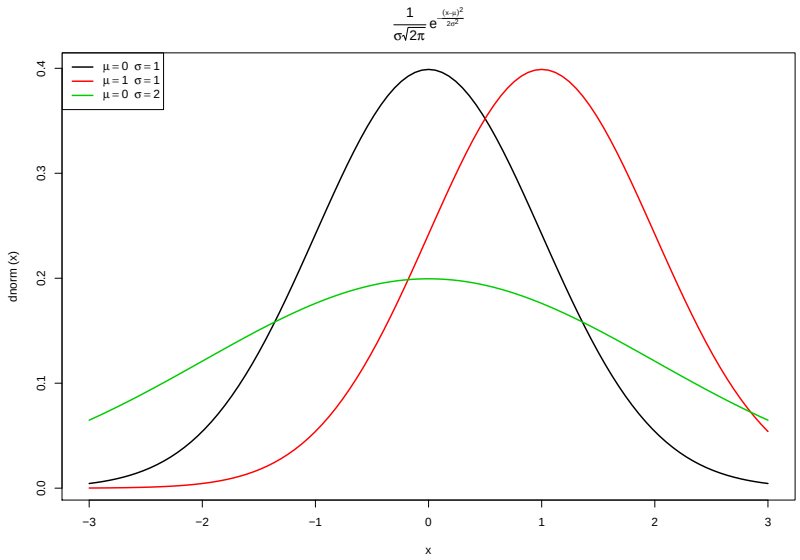
`R > axis(4,col="red",ylab="v2")` on ajoute maintenant l'axe pour `v2` sur la droite du graphique (1er argument = 4)

`R > mtext("v2",4,col="red")` on ajoute le label de l'axe des ordonnées à droite grâce à `mtext()`.

4.3 Insertion de formules mathématiques

Les formules mathématiques peuvent être introduites sur un graphique de la même façon qu'un texte standard : dans le titre, les labels des axes, la légende... Il faut pour cela respecter une syntaxe particulière.

```
R > ?expression;?plotmath
R > demo(plotmath)
R > plot(dnorm,-3,3)
R > x=seq(-3,3,l=100)
R > lines(x,dnorm(x,1,1),col=2)
R > lines(x,dnorm(x,0,2),col=3)
R > title(expression(
+ over(1,sigma*sqrt(2*pi))* e^{-over((x-mu)^2,2*sigma^2)}))
R > expr1 = expression(mu==0~~sigma==1)
R > expr2 = expression(mu==1~~sigma==1)
R > expr3 = expression(mu==0~~sigma==2)
R > legend("topleft",c(expr1,expr2,expr3),lty=1,col=1:3)
```

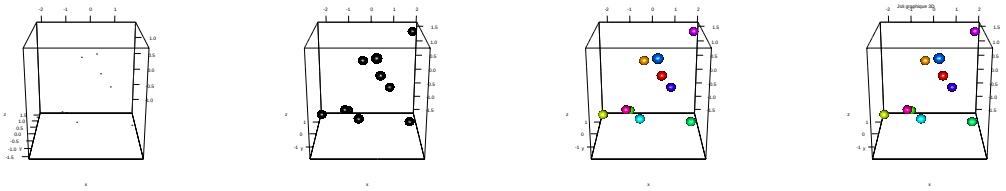


4.4 Quelques packages

Dans cette section, quelques packages proposant des graphiques particuliers sont présentés. Il pourrait y en avoir beaucoup d'autres ; voir par exemple la *Task View* **Graphics** à l'adresse http://cran.r-project.org/web/views/Graphics.html. Pour chacun de ces packages, quelques lignes de commande issues généralement de l'aide en ligne donnent un aperçu des possibilités offertes. Pour aller plus loin, comme d'habitude, la rubrique *See Also* permettra de construire petit à petit sa propore boîte à outils.

4.4.1 Graphiques 3D avec `rgl`

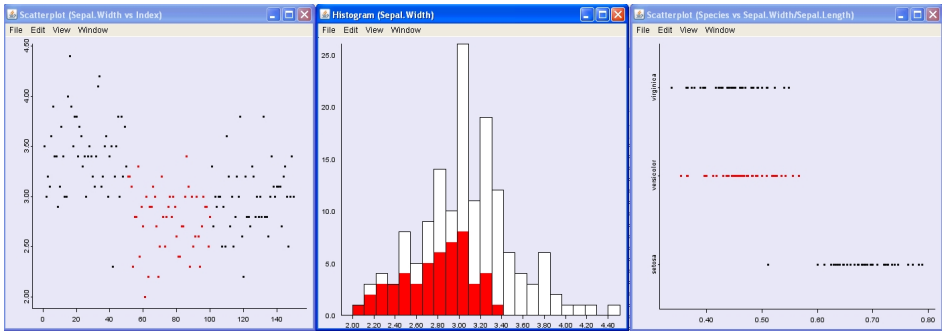
Le package `rgl` permet de représenter des données en 3 dimensions dans un environnement interactif dans lequel l'utilisateur peut zoomer et effectuer des rotations à l'aide de la souris.



```
R > library(rgl)
R > example(rgl)
R > x=rnorm(50);y=rnorm(50);z=rnorm(50)
R > plot3d(x,y,z)
R > rgl.postscript("rgl1.pdf",fmt="pdf")
R > plot3d(x,y,z,type="s")
R > plot3d(x,y,z,type="s",col=rainbow(10))
R > title3d("Joli graphique 3D",font=2,cex=1.5)
```

4.4.2 Graphiques interactifs avec `iplots`

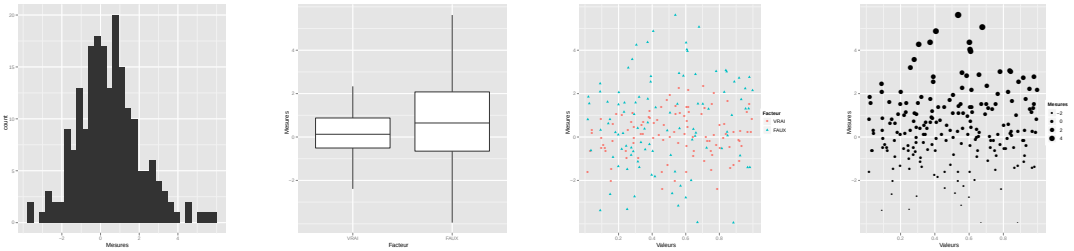
Le package `iplots` permet de représenter des graphiques interactifs dans des fenêtres *Java*. La sélection d'un ensemble de points sur un graphique implique automatiquement la mise en évidence de ces mêmes individus sur les autres graphiques.



```
R > attach(iris)
R > library(iplots)
R > ?iplot
R > data(iris)
R > iplot(Sepal.Width/Sepal.Length,Species)
R > ihist(Sepal.Width)
R > iplot(Sepal.Width)
```

4.4.3 Graphiques "élégants" avec `ggplot2`

Le package `ggplot2` permet de produire des graphiques plus "élégants" que les graphiques classiques. Il nécessite un investissement relativement important car il propose plus globalement une véritable grammaire pour la construction de représentations graphiques à partir de données. Les sources d'information concernant ce package sont le site http://ggplot2.org/ et l'ouvrage *ggplot2 : Elegant Graphics for Data Analysis (Use R !)* (Springer) par Hadley Wickham.

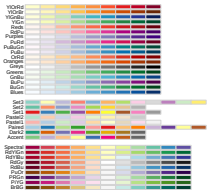
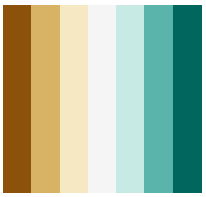
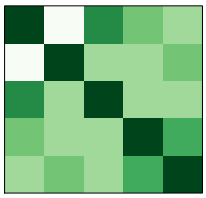
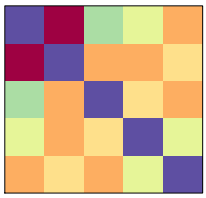


```
R > library(ggplot2)
```
Préparation d'un petit data.frame
```
R > mes = c(rnorm(100),rnorm(100,1,2))
R > mes2 = runif(200)
R > fac = gl(2,100,labels=c("VRAI","FAUX"))
R > Mon.df = data.frame(Mesures=mes,Valeurs=mes2,Facteur=fac)
```
Représentation "rapide" à l'aide de la fonction `qplot()` (**q***uick plot*)
```
R > qplot(Mesures, data=Mon.df,geom="histogram")
R > MonGraphe = ggplot(Mon.df,aes(x=Facteur,y=Mesures))
R > MonGraphe2 = ggplot(Mon.df,aes(x=Valeurs,y=Mesures))
```
Aucun graphique n'est représenté pour le moment.
```
R > MonGraphe+geom_boxplot()
R > MonGraphe2+geom_point(aes(shape=Facteur,colour=Facteur))
R > MonGraphe2+geom_point(aes(size=Mesures))
```

Remarque : le package *lattice* est également très utilisé pour constuire des graphiques complexes avec une programmation simplifiée (tout comme `ggplot2`) (voir http://docs.ggplot2.org/current/translate_qplot_lattice.html).

4.4.4 Palettes de couleur avec `RColorBrewer`

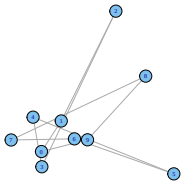
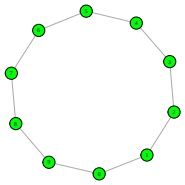
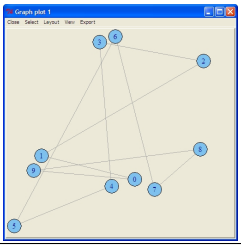
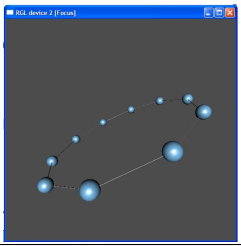
Le package `RColorBrewer` propose des palettes de couleur qui peuvent être très utile pour utiliser la couleur comme traduisant l'évolution d'une variable.

```
R > library("RColorBrewer")
R > display.brewer.all()
R > display.brewer.pal(7,"BrBG")
R > cormat = cor(matrix(rnorm(50),nc=5))
R > image(t(cormat[5:1,]),axes=FALSE);box()
R > MaPalette = brewer.pal(9,"Greens")
```
pour ceux qui aiment le vert
```
R > image(t(cormat[5:1,]),col=MaPalette,axes=FALSE);box()
R > MaPalette = brewer.pal(10,"Spectral")
R > image(t(cormat[5:1,]),col=MaPalette,axes=FALSE);box()
```

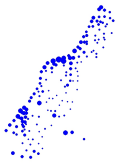
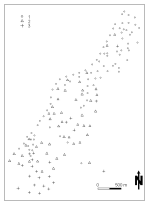
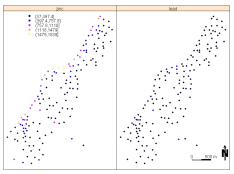
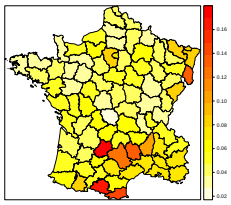
4.4.5 Représentation de réseaux avec `igraph`

Le package `igraph` propose des fonctions pour l'analyse de réseaux. Nous nous focalisons ici sur les représentations graphiques de ce type de données. Pour en savoir plus, consulter le site consacré à `igraph` à l'adresse `http://igraph.sourceforge.net/`.

```
R > library(igraph)
R > ?igraph;?plot.igraph;?igraph.plotting
R > g = graph.ring(10)
R > plot(g)
R > plot(g, layout=layout.kamada.kawai,vertex.color="green")
R > tkplot(g)
R > coords=layout.fruchterman.reingold(g, dim=3)
R > rglplot(g, layout=coords)
```

4.4.6 Données géo-localisées

```
R > require("sp")
R > data("meuse")
R > coordinates(meuse) <-  ~  x+y
R > spplot(meuse,"zinc")
R > require("GeoXp")
R > ginimap(meuse,"zinc")
```

- Consulter l'aide sur les données `meuse`
- le package `GeoXp` est dédiée à l'analyse exploratoire de données géo-localisées. Ref : T. Laurent, A. Ruiz-Gazen, C. Thomas-Agnan (2012), GeoXp : An R Package for Exploratory Spatial Data Analysis, Journal of Statistical Software, 47(2).

4.5 Exercice : plusieurs représentations autour d'une loi normale centrée réduite

Pour conclure cette section, nous proposons un ensemble de représentations autour d'une loi normale centrée réduite. à partir d'un vecteur de longueur 500 issu du tirage aléatoire selon une loi normale centrée réduite, représenter sur un même graphique :

1. un histogramme des données ;
2. un "tapis" sur l'axes des abscisses matérialisant les valeurs observées ;
3. un courbe représentant la densité empirique des données (voir la fonction, `density()`) ;
4. la "vraie" densité d'une loi normale centrée réduite ;
5. une légende.

Le tout avec des couleurs différentes pour en rendre la lecture agréable.

Génération des données :

```
R > x<-rnorm(500)
```

Voici une façon de représenter les différents éléments demandés :

1. l'histogramme

```
R > hist(x, freq=FALSE, nclass=15,
+ main="Représentations autour d'une N(0,1)",
+ col="lightblue", xlim=c(-3,3),ylim=c(0,0.4))
```

2. le "tapis"

```
R > rug(x)
```

3. la densité empirique

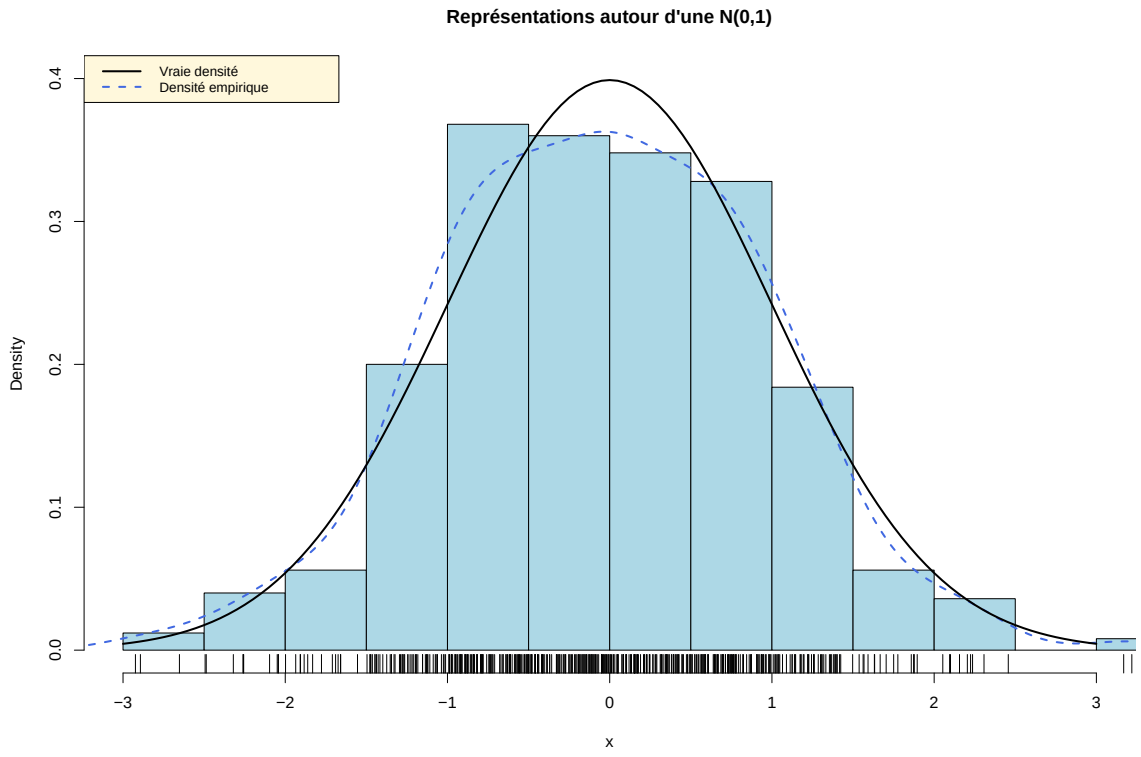
```
R > lines(density(x), col="royalblue", lty=2,lwd=2)
```

4. la "vraie" densité

```
R > lines(seq(-3,3,0.05),dnorm(seq(-3,3,0.05)),
+ lty=1, col="black",lwd=2)
```

5. la légende

```
R > legend("topleft",
+ legend=c("Vraie densité","Densité empirique"),
+ col=c("black","royalblue"),lty=1:2,lwd=2,
+ cex=0.8, bg="cornsilk")
```



Chapitre 5

Mathématique

5.1 Calcul matriciel

```
R > A <- matrix(1:12, nc=3)
R > B <- matrix(1:6, nc=2)
R > t(A); A*B; A%*%B
R > solve(A)
R > solve(A[1:3,1:3])
R > A <- matrix(runif(4), nc=2)
R > diag(A); solve(A)
R > eigen(t(A)%*%A))
R > svd(A); qr(A)
```

- La fonction `t` transpose une matrice.
- Le produit matriciel se fait avec l'opérateur `%*%`.
- L'inversion de matrice est possible avec la fonction `solve`.
- Plusieurs décompositions de matrices sont disponibles : éléments propres (`eigen`), valeurs singulières (`svd`), QR (`qr`), Choleski (`chol`)...

5.2 Optimisation

```
R > f <- function(x) sin(x/2)
R > optim(par=0, fn=f)
R > sol <- optimise(f, c(-5,5))
R > plot(f, -5, 5)
R > points(sol$min, sol$obj)
R > solb <- nlminb(0,f,
+ lower=-2, upper=2)
R > abline(v=c(-2,2))
R > points(solb$par, solb$obj)
```

- Les fonctions d'optimisation sont adaptées à des cas plus ou moins spécifiques.
- Par exemple, `optim` est générique, `optimize` adaptée aux problèmes 1D, `constrOptim` et `nlminb` pour l'optimisation sous contrainte...
- Plusieurs algorithmes sont généralement disponibles pour chaque fonction via des paramètres optionnels. Consulter l'aide en ligne pour voir lesquelles sont mises en œuvre.

5.3 Dérivation - Intégration

```
R > f <- expression(3*x^2 + 5*y^3)
R > dx <- deriv(f,"x")
R > dy <- deriv(f,"y")
R > dxy <- deriv(f,c("x","y"))
R > x <- 1:10
R > y <- 10:1
R > eval(dxy)
R > integrate(dnorm, -1.96, 1.96)
R > pnorm(1.96)-pnorm(-1.96)
```

- Les fonctions `D` et `deriv` permettent de faire du calcul symbolique de dérivées.
- Elles opèrent sur des objets `expression`.
- Le package `Deriv` simplifie certaines opérations de dérivation.
- La fonction `integrate` permet le calcul d'intégrales de fonctions 1D sur des intervalles donnés.

Questions

Considérons la fonction f et deux vecteurs définis par

```
R > f <- function(x) 3*x+10*cos(x)
R > tps <- seq(0,10, l=100)
R > bruit <- 5*rnorm(100)
```

1. Pour chaque valeur du vecteur `tps`, simuler des mesures d'après la fonction `f` bruitées par le vecteur `bruit`.
2. Mettre en oeuvre un lissage spline des données avec la fonction `smooth.spline`.
3. Représenter graphiquement les données ainsi que le lissage obtenu.
4. Calculer la dérivée de la fonction lissée avec la fonction `predict.smooth.spline`
5. Représenter graphiquement les valeurs de la dérivée et constater que la fonction lissée est bien croissante (respectivement décroissante) quand la dérivée est positive (respectivement négative).

Réponses

1. Simulation des mesures
   ```
   R > mesure <- f(tps) + bruit
   ```
2. Lissage spline
   ```
   R > res.sm <- smooth.spline(tps, mesure)
   ```
3. Représentation graphique
   ```
   R > par(mfrow=c(2,1)
   R > plot(tps, mesure);lines(res.sm, col=2)
   ```
4. Calcul de la dérivée
   ```
   R > der.sm <- predict(res.sm, tps, deriv = 1)
   ```
5. Représentation graphique
   ```
   R > plot(tps, der.sm$y, col=3,type="l", ylab="Dérivée")
   R > abline(h=0)
   ```

Chapitre 6

Modélisation statistique

Dans ce chapitre, nous passons en revue quelques méthodes statistiques pouvant entrer dans un cadre générique de modélisation c'est à dire où une variable est à expliquer par une plusieurs autres variables. Pour chacune de ces méthodes, nous allons voir que la syntaxe R reste quasiment la même et est basée sur l'expression de formules.

```
R > help(formula)
```

> ...
>
> **Details**
>
> The models fit by, e.g., the lm and glm functions are specified in a compact symbolic form. The ~operator is basic in the formation of such models. An expression of the form y ~model is interpreted as a specification that the response y is modelled by a linear predictor specified symbolically by model. Such a model consists of a series of terms separated by + operators. The terms themselves consist of variable and factor names separated by : operators. Such a term is interpreted as the interaction of all the variables and factors appearing in the term.
>
> In addition to + and :, a number of other operators are useful in model formulae. The * operator denotes factor crossing : a*b interpreted as a+b+a :b. The ^ operator indicates crossing to the specified degree. For example (a+b+c)^2 is identical to (a+b+c)*(a+b+c) which in turn expands to a formula containing the main effects for a, b and c together with their second-order interactions. The %in% operator indicates that the terms on its left are nested within those on the right. For example a + b %in% a expands to the formula a + a :b. The - operator removes the specified terms, so that (a+b+c)^2 - a :b is identical to a + b + c + b :c + a :c. It can also used to remove the intercept term : when fitting a linear model y x - 1 specifies a line through the origin. A model with no intercept can be also specified as y x + 0 or y 0 + x.
>
> ...

6.1 Régression linéaire multiple

```
R > data("swiss")
```

```
R > fit = lm(Fertility ~ Education+Infant.Mortality,swiss)
```

Que contient l'objet `fit` créé ?

```
R > str(fit)
```

Pour obtenir un résumé du modèle, à savoir tests de signifficativité du modèle et des coefficients, estimation des coefficients, indice d'ajustement du modèle (R2 et R2 ajusté), etc.

```
R > summary(fit)
```

- Les coefficients :
  ```
  R > coefficients(anov)
  ```
- Les valeurs ajustées
  ```
  R > predict(fit)
  ```
- les résidus :
  ```
  R > residuals(fit)
  ```
- analyse des résidus (moyenne constante, normalité, homoscédasticité, points influents)
  ```
  R > plot(fit)
  ```
- Critére de comparaison :
  ```
  R > AIC(fit)
  ```

Indépendamment de cet exemple, une petite astuce pour écrire un modèle sans saisir une à une les variables :

```
R > xnam <- paste("x", 1:25, sep="")
R > fmla <- as.formula(paste("y ~ ", paste(xnam, collapse= "+")))
```

6.2 Analyse de variance à un facteur

```
R > data("InsectSprays")
```

Pour afficher la significativité du modèle

```
R > anov=aov(sqrt(count) ~spray, data = InsectSprays)
```

`R > anov` renvoie un résumé très succint avec la somme des carrés intra-classes, puis inter-classes.

Pour obtenir le test de significativité du modèle :

```
R > summary(anov)
```

Les fonctions déjà vues pour la régression linéaire sont utilisables :

```
R > coefficients(anov);predict(anov);residuals(anov)
R > plot(anov);str(anov)
```

6.3 Régression logistique

```
R > require("mlbench")
R > data("BreastCancer")
R > res.glm=glm(Class ~Normal.nucleoli,data=BreastCancer[,-1],
+ family=binomial)
```

– Qualité du modèle

```
R > summary(res.glm)
```

– courbe de ROC

```
R > libary(ROCR)
R > pred = prediction( fitted(res.glm), BreastCancer$Class)
R > perf = performance( pred, "tpr", "fpr" )
R > plot(perf, colorize = TRUE, lwd=3)
```

– Table des bons et mal classés

```
R > table(BreastCancer$Class,ifelse(fitted(res.glm)>0.5,
+ "hat.malin","hat.benin"))
```

– Les odds ratio :

```
R > lreg.coeffs <- coef(summary(res.glm))*0.01
R > odds <- data.frame(signif(cbind(exp(lreg.coeffs[, 1]),
+ exp(lreg.coeffs[, 1] - 1.96 * lreg.coeffs[, 2]),
+ exp(lreg.coeffs[, 1] + 1.96 * lreg.coeffs[, 2])),3))
+ names(odds) <- c("odds", "l.95", "u.95")
R > odds
```

Pour sélectionner les variables d'un modèle, on peut par exemple choisir comme modèle final celui minimisant le critère AIC :

```
R > fit = glm(Fertility ~. ,swiss, family=gaussian)
R > step(fit)
```

6.4 Les arbres de régression (*Classification And Regression Trees, CART*)

Le principe de cette méthode ainsi que l'interprétation de ces résultats sont relativement facile à comprendre à la vue des sorties graphiques proposées.

```
R > require("rpart")
R > fit.rpart <- rpart(Class ~. ,BreastCancer[,-1], method = "class")
R > plot(fit.rpart, uniform = TRUE, branch = 0.5, margin = 0.1)
R > text(fit.rpart, all = FALSE, use.n = TRUE)
```

Une méthode complémentaire aux arbres de régression est la méthode des forêts aléatoires (voir package `randomForest`).

6.5 Modélisation de type Modèles Additifs généralisés

Pour plus de détails sur le package `mgcv` consulter la page de Simon Wood à l'adresse people.bath.ac.uk/sw283/mgcv. Le code ci-dessous en est, en partie, extrait.

```
R > library("mgcv");library("gamair")
R > data("brain")
```

Représentation des données brutes

```
R > par(mfrow=c(1,1))
```
```
R > brain <- brain[brain$medFPQ>5e-3,]
```
pour exclure 2 pixels défectueux
```
R > X<-sort(unique(brain$X));Y<-sort(unique(brain$Y))
R > z <- matrix(NA,49,43)
R > mx<-min(brain$X)-1;my<-min(brain$Y)-1
R > for (i in 1:length(brain$medFPQ))
+ z[brain$Y[i]-my,brain$X[i]-mx] <- brain$medFPQ[i]
R > image(Y,X,log(z),zlim=c(-4,4),main="medFPQ brain image")
```
Utilisons un modèle où les variables explicatives s'expriment comme une fonction non paramétrique...
```
R > m0 <- gam(medFPQ s(X,Y),data=brain)
R > gam.check(m0)
```
... puis un modèle plus complexe
```
R > m2 <- gam(medFPQ s(X,Y,k=100),data=brain,
+ family=Gamma(link=log),method="REML")
R > gam.check(m2)
R > plot(m2)
R > vis.gam(m2,view=c("Y","X"),too.far=.03,plot.type="contour")
```
Il existe beaucoup d'autres méthodes pouvant entrer dans ce cadre "Modélisation" ; voir par exemple les Support Vecteur Machines (SVM, package `e1071`, les réseaux de neurones (package `nnet`), les forêts aléatoires (package `randomForest`)...

6.6 Données de panel

Le package *plm* contient un certain nombre de fonctions pour effectuer des régressions ainsi que des tests sur des données de panel. Par exemple, pour faire un modèle à effets fixes individuels :
```
R > require("plm")
R > data("Produc")
R > res = plm(log(gsp)   log(pcap)+log(pc) + unemp,
+ data=Produc,index=c("state","year"))
```

Chapitre 7

Divers

7.1 LaTeX

Pour les utilisateurs de LaTeX, la mise en forme de sorties R est possible directement par l'utilisation, par exemple, du package `xtable`.

```
R > toLatex(sessionInfo())
R > library(xtable)
R > x=rnorm(10);y=x+rnorm(10,0,0.5)
R > df.xy=data.frame(x,y)
R > xtable(df.xy)
R > ModLin = lm(y x)
R > xtable(ModLin)
R > ?xtable
```

7.2 Sweave

La fonction Sweave de R convertit un fichier au format .rnw, dont la propriété est de contenir à la fois du code LaTeXet du code R, en un fichier .tex, où les lignes de codes ainsi que les sorties de console R, ainsi que les graphiques, ont été inclus comme étant du TeX. L'avantage est de produire un seul document (codes + commentaires) et du coup, s'il y a des changements de code à faire, ceci se fait uniquement dans le fichier .rnw.

Remarque : pour effectuer la compilation latex2pdf, il faut que le package Sweave.sty soit dans le même répertoire que le fichier .tex.

Exemple de fichier Sweave :

```
R > example = system.file("Sweave","Sweave-test-1-Rnw",
+ package="utils")
R > Sweave(example,syntax = "SweaveSyntaxNoweb")
```

Pour extraire uniquement le code R d'un fichier .rnw, on utilise la fonction Stangle qui créé ensuite le fichier example.R

```
R > Stangle(example)
```

Remarque : lorsqu'un compilateur LaTeXest installé, RStudio permet de créer le document pdf en un seule clic.

7.3 odfWeave

On part d'un fichier .odt dans lequel on inclut du code R entre les balises `<< >> =` et se terminant par `@`. On compile ce document sous R avec la fonction `odfWeave()` du package *odfWeave* qui crée un nouveau document avec le code R exécuté.

```
R > require("odfWeave")
R > demoFile = system.file("examples","simple.odt",
+ package="odfWeave")
R > odfWeave(demofile)
```

7.4 Knitr

Il s'agit d'un package (pré-installé avec RStudio) qui permet d'écrire des rapports dans des formats html, markdown... incluant les sorties de codes R. Son usage est trivial avec Rstudio car ce dernier propose des fichiers pré-remplis. Enfin, il est possible d'utiliser les fichiers .Rnw pour créer un document .pdf dans un style un peu différent que celui utilisé par Sweave. Les possibilités de *knitr* semblent énormes (graphiques interactifs, graphiques avec des formules LaTeX...).

```
R > require("knitr")
R > Sweave2knitr(example,output="Sweave-test-knitr.Rnw")
R > knit2pdf("Sweave-test-knitr.Rnw")
```

7.5 Make et Makefile

Lorsqu'il y a plusieurs documents de travail au format Sweave (par exemple si vous écrivez des livres avec plusieurs chapitres), il est préférable de créer un fichier `makefile` qui contiendra les instructions pour compiler séparément les fichiers, puis les réunit dans un fichier source.... Une fois le fichier `makefile` créé, vous pouvez le compiler dans une console en tapant l'instruction :

```
> make
```

Voir le fichier makefile joint

Pour plus d'informations : www.stat.auckland.ac.nz/ stat782/downloads/make-tutorial.pdf

7.6 Rforge

La Rforge r-forge.r-project.org/ permet le partage de fichiers avec des collaborateurs ainsi que le suivi des versions des fichiers. Cela peut se révéler très pratique voir indispensable pour le développement de packages par plusieurs personnes.

Bibliographie

Voici quelques compléments aux références déjà listées dans le document *Pour se donner un peu d'R*

Sur le web :

- *Using color in R*, Earl F. Glynn - research.stowers-institute.org/efg/Report/UsingColorInR.pdf
- *Mise en forme de la fenêtre "graphique" : agencements complexes de figures*, Caroline Simonis - rug.mnhn.fr/semin-r/PDF/semin-R_graphiquescomplexes_CSimonis_080408.pdf
- *Programmer en R*, Gilles Hunault - www.info.univ-angers.fr/g̃h/wstat/Programmation_R
- gallery.r-enthusiasts.com
- *R Graphics Gallery* - research.stowers-institute.org/efg/R
- *R plot and graph gallery* - www.sr.bham.ac.uk/ ajrs/R/r-gallery.html

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