

Introduction à R

Magalie Fromont

R est un système d'analyse statistique et graphique créé par R. Ihaka et R. Gentleman. C'est à la fois un logiciel et un langage proche du langage S développé par R. Becker, J. Chambers et A. Wilks des Bell laboratories. Il est diffusé selon le principe de licence GNU et accessible : www.r-project.org.

Pour plus d'informations, on renvoie aux références suivantes :

Ihaka R. and Gentleman R. (1996). R : a language for data analysis and graphics. *Journal of Computational and Graphical Statistics* 5 : 299-314.

Becker R., Chambers J. and Wilks A. (1988). *The New S Language : A Programming Environment for Data Analysis and Graphics*. Chapman and Hall, New York.

Chambers J. and Hastie T. (1992). *Statistical Models in S*. Chapman and Hall, New York.

Venables W. and Ripley B. *Modern Applied Statistics with S-PLUS*.

De nombreux tutoriels sont accessibles en ligne, comme par exemple :

Venables W. and Smith D. *An introduction to R*. Distribué avec R.

Paradis E. *R pour les débutants*.

L'objet de cette introduction est de proposer un tour d'horizon de quelques fonctionnalités basiques de R sous UNIX et Windows.

1 Prise en main

1.1 Les commandes de base

R	Appeller R. > s'affiche, indiquant que R est prêt à fonctionner.
q()	Quitter R. <code>Save workspace image? [y/n/c]</code> s'affiche → Sauvegarder ou non l'espace de travail.
INTERRUPT (e.g. C-c ou esc)	Arrêter la commande en cours.

Pour connaître le répertoire de travail, on utilise la commande `getwd()`. Pour modifier ce répertoire, on utilise `setwd("/home/étudiant/R")`. Pour afficher la liste des fichiers du répertoire courant, on utilise `list.files()`.

Caractères spéciaux : ";" permet de séparer des commandes sur une même ligne, "#" de mettre des commentaires, "\n" permet d'aller à la ligne dans une chaîne de caractères, "\t" produit une tabulation.

Les flèches au clavier permettent de rappeler des commandes.

1.2 À propos de l'aide

<code>help.start()</code>	Ouvrir l'aide dans netscape. Visiter les Packages est souvent d'une très grande utilité.
<code>help("lm")</code>	Afficher les détails de l'aide sur la fonction "lm".
<code>help.search("lm")</code>	Rechercher la fonction "lm" dans les packages R.

1.3 À propos des packages

<code>library()</code>	Visualiser tous les packages disponibles.
<code>library(MASS)</code>	Charger le package MASS en mémoire.

Pour installer un nouveau package, par exemple `gee`, on téléchargera dans un premier temps le fichier `gee_4.13-6.tar.gz` disponible sur le site web du CRAN. On tapera ensuite à partir du système (et non de R) la commande `R INSTALL gee_4.13-6.tar.gz`. Il est utile de vérifier régulièrement les versions des packages installés en comparaison à celles disponibles sur CRAN à l'aide de la commande `update.packages()`.

1.4 Les entrées/sorties

<code>source("mescommandes.txt")</code>	Exécuter les commandes contenues dans le fichier "mescommandes.txt"
<code>sink("messorties.txt")</code>	Rediriger les sorties vers le fichier "messorties.txt".
<code>sink()</code>	Arrêter la redirection.

2 Les objets dans R

2.1 Les commandes de base

<code>objects()</code> ou <code>ls()</code>	Afficher la liste des objets en mémoire.
<code>ls(pat="a")</code>	Afficher la liste des objets contenant le caractère "a".
<code>ls(pat="^a")</code>	Afficher la liste des objets commençant par "a".
<code>ls.str()</code>	Afficher la liste des objets en mémoire avec des détails sur ces objets.
<code>rm(x)</code> , <code>rm(x,y)</code>	Effacer les objets x ou x et y de la mémoire.
<code>rm(list=ls())</code> , <code>rm(list=ls(pat="^a"))</code>	Effacer tous les objets ou certains objets de la mémoire.

Pour visualiser un objet x, il suffit de l'appeler à l'aide de `x` ou `print(x)`.

2.2 Classes et attributs

R manipule plusieurs classes d'objets : les vecteurs, les facteurs, les arrays, les matrices, les data frames, les ts, les listes. Pour connaître la classe d'un objet x, on utilise la commande `class(x)`. On peut savoir si un objet x est d'une certaine classe à l'aide des commandes `is.vector(x)`, `is.factor(x)`, `is.matrix(x)`, `is.data.frame(x)`, `is.ts(x)`... (la réponse est `TRUE` ou `FALSE`).

Chaque objet est caractérisé par son nom, son contenu, mais aussi par des attributs, dont le mode (`nul` : `NULL`; numérique : `1`, `2.333`, `pi`, `Inf`, `-Inf`, `2.1e23`; caractère : `'blabla'`, `"blabla"`; complexe : `2+0i`, `2i`; logique : `TRUE`, `FALSE`) et la longueur. Pour connaître ces deux attributs, on utilise les commandes `mode(x)` et `length(x)`. Pour connaître les autres attributs `attributes(x)`.

2.3 Les valeurs manquantes

Elles sont notées `NA` (not available) ou `NaN` (not a number). On peut savoir si un objet est composé de valeurs manquantes à l'aide de `is.na(x)` ou `is.nan(x)`. Ces commandes renvoient un objet de mode logique de même longueur que x.

`na.fail(x)` retourne un message d'erreur si l'objet x contient au moins une valeur manquante.

3 Créer et manipuler des objets

Un objet peut être créé avec l'opérateur "assigner" qui s'écrit avec une flèche "`<-`" ou "`->`", ou tout simplement avec un signe "`=`". Si l'objet existe déjà, sa valeur précédente est effacée.

Attention, R distingue, pour les noms des objets, les minuscules des majuscules.

```
x<-15
15->x
x=15
x="blabla"
```

On peut convertir la classe des objets à l'aide de commandes comme `as.matrix`, `as.data.frame`, `as.ts`..., leur mode à l'aide de `as.numeric`, `as.logical`, `as.character`, `as.complex`, leurs attributs à l'aide de `attr(x, "dim")<-`.

3.1 Les nombres et les vecteurs

Un nombre est considéré dans R comme un vecteur de taille 1 de mode numérique ou complexe.

3.1.1 Créer des vecteurs

<code>numeric(n)</code>	Créer un vecteur de taille <code>n</code> de mode numérique ne contenant que des 0.
<code>logical(n)</code>	Créer un vecteur de taille <code>n</code> de mode logique ne contenant que des <code>FALSE</code> .
<code>character(n)</code>	Créer un vecteur de taille <code>n</code> de mode caractère ne contenant que des <code>" "</code> .
<code>x<-c(1,2,3,4,5,6,7,8,9,10)</code>	Créer un vecteur contenant les valeurs 1,2,3,4,5,6,7,8,9,10.
<code>x<-1 :10</code>	Créer un vecteur contenant une séquence régulière de nombres entiers de 1 à 10.
<code>x<-seq(1,5,by=0.5)</code>	Créer un vecteur contenant une séquence de nombres réels dont le premier est 1, le dernier 5, l'incrément utilisé dans la progression de la séquence 0.5.
<code>x<-seq(1,5,length=100)</code>	Créer un vecteur contenant une séquence régulière de taille 100 de nombres réels dont le premier est 1, le dernier 5.
<code>x<-rep(1,100)</code>	Créer un vecteur contenant 100 fois le nombre 1.
<code>x<-rep(1 :4,25)</code>	Créer un vecteur contenant 25 fois la suite 1,2,3,4.
<code>x<-sequence(c(10,5))</code>	Créer un vecteur contenant une suite de séquences de nombres entiers qui chacune se termine par les nombres 10 et 5.
<code>x<-c("a","b","c")</code>	Créer un vecteur contenant les caractères a, b et c.
<code>x<-rep("a",100)</code>	Créer un vecteur contenant 100 fois le caractère a.
<code>noms<-paste("A",1 :5,sep="")</code>	Créer un vecteur contenant les caractères A1, A2, A3, A4, A5.
<code>cit<-"Elle dit : \" blablabla\""</code>	Créer une chaîne de caractères contenant des guillemets.
<code>cat(cit)</code>	Afficher une chaîne de caractères contenant des guillemets.
<code>x<-c(y,z)</code>	Créer un vecteur qui concatène les vecteurs y et z.

On peut générer des vecteurs contenant des données aléatoires à l'aide de la commande `rfun(n,p1, p2,...)`, où `fun` indique la loi de probabilité, `n` le nombre de données générées, `p1, p2,...` les paramètres de la loi de probabilité. Exemples : `rnorm(n,0,1)`, `rexp(n,1)`, `rpois(n,2)`, `rchisq(n,100)`, `rt(n,100)`, `rf(n,100,50)`, `rbinom(n,100,0.5)`, `runif(n,0,1)`...

Remarque : On peut remplacer dans ces fonctions la lettre `r` par `d`, `p` ou `q` pour obtenir la densité de probabilité, la fonction de répartition ou la valeur des quantiles.

3.1.2 Manipuler des vecteurs

• Les différents opérateurs

Les opérateurs arithmétiques classiques sont donnés par `+` (addition), `-` (soustraction), `*` (multiplication), `/` (division), `^` (puissance), `%%` (congruence), `%/%` (division entière).

Les opérateurs de comparaison sont donnés par `<` (inférieur à), `>` (supérieur à), `<=` (inférieur ou égal à), `>=` (supérieur ou égal à), `==` (égal à), `!=` (différent de).

Ces opérateurs agissent sur les vecteurs élément par élément. Pour effectuer une comparaison globale de deux objets, on peut utiliser `identical`.

Les opérateurs logiques sont donnés par `!x` (non logique), `x&y` (et logique opérant sur tous les éléments de x et y), `x&&y` (et logique opérant sur le premier élément de x et y), `x|y` (ou logique opérant sur tous les éléments de x et y), `x||y` (ou logique opérant sur le premier élément), `xor(x,y)` (ou exclusif).

On code l'inégalité $0 < x < 1$ par `x > 0 & x < 1`.

- Extraire une partie d'un vecteur

La sélection d'une partie d'un vecteur `x` s'opère à l'aide de la commande `x[vecteur]`, où `vecteur` est soit un vecteur d'entiers positifs, soit un vecteur d'entiers négatifs, soit un vecteur logique.

<code>x[i]</code> , <code>x[c(i,j)]</code>	Extraire la <i>i</i> ème valeur ou la <i>i</i> ème et la <i>j</i> ème valeur du vecteur <code>x</code> .
<code>x[-i]</code>	Supprimer l'élément <code>x[i]</code> dans <code>x</code> .
<code>x[-c(i,j)]</code>	Supprimer les éléments <code>x[i]</code> et <code>x[j]</code> dans <code>x</code> .
<code>x[x >= 1]</code>	Extraire les valeurs supérieures ou égales à 1 du vecteur <code>x</code> .
<code>x[x %% 2 == 0]</code>	Extraire les valeurs paires du vecteur <code>x</code> .
<code>x[x >= 1] <- 1</code>	Remplacer dans le vecteur <code>x</code> toutes les valeurs supérieures ou égales à 1 par 1.
<code>x[!is.na(x)]</code>	Extraire les valeurs non manquantes de <code>x</code> . Idem à <code>na.omit(x)</code> .
<code>x[is.na(x)] <- 0</code>	Remplacer les valeurs manquantes de <code>x</code> par 0.

- Les fonctions mathématiques

Les fonctions mathématiques de base sont toutes disponibles dans R (`sqrt`, `abs`, `log`, `exp`, `log10`, `sin`, `cos`, `tan`, `asin`, ...).

<code>sum(x)</code>	Calculer la somme des éléments de <code>x</code> .
<code>prod(x)</code>	Calculer le produit des éléments de <code>x</code> .
<code>diff(x)</code>	Calculer la différence entre les éléments consécutifs de <code>x</code> .
<code>max(x)</code>	Donner le maximum des éléments de <code>x</code> .
<code>min(x)</code>	Donner le minimum des éléments de <code>x</code> .
<code>which.max(x)</code>	Donner l'indice du maximum des éléments de <code>x</code> .
<code>which.min(x)</code>	Donner l'indice du minimum des éléments de <code>x</code> .
<code>range(x)</code>	Donner le minimum et le maximum des éléments de <code>x</code> .
<code>length(x)</code>	Donner le nombre d'éléments dans <code>x</code> .
<code>mean(x)</code>	Calculer la moyenne des éléments de <code>x</code> .
<code>median(x)</code>	Calculer la médiane des éléments de <code>x</code> .
<code>var(x)</code> , <code>cov(x)</code>	Calculer la variance empirique des éléments de <code>x</code> .
<code>var(x,y)</code> , <code>cov(x,y)</code>	Calculer la covariance entre <code>x</code> et <code>y</code> .
<code>corr(x,y)</code>	Calculer la corrélation linéaire entre <code>x</code> et <code>y</code> .
<code>sd(x)</code>	Calculer l'écart type empirique de <code>x</code> .
<code>quantile(x)</code>	Calculer le quantile empirique de <code>x</code> .
<code>round(x,n)</code>	Arrondir les éléments de <code>x</code> à <code>n</code> chiffres après la virgule.
<code>rev(x)</code>	Inverser l'ordre des éléments de <code>x</code> .
<code>sort(x)</code>	Ordonner les éléments de <code>x</code> .
<code>rev(sort(x))</code>	Ordonner les éléments de <code>x</code> dans l'ordre descendant.
<code>rank(x)</code>	Donner les rangs des éléments de <code>x</code> .
<code>order(x)</code>	Donner les coordonnées du plus petit élément de <code>x</code> , du deuxième plus petit élément de <code>x</code> , etc.
<code>pmin(x,y)</code> , <code>pmax(x,y)</code>	Donner un vecteur dont le <i>i</i> ème élément est le minimum ou le maximum entre <code>x[i]</code> et <code>y[i]</code> .
<code>cumsum(x)</code> , <code>cumprod(x)</code>	Donner un vecteur dont le <i>i</i> ème élément est la somme ou le produit des éléments <code>x[1]</code> à <code>x[i]</code> .
<code>cummin(x)</code> , <code>cummax(x)</code>	Donner un vecteur dont le <i>i</i> ème élément est le minimum ou le maximum des éléments <code>x[1]</code> à <code>x[i]</code> .
<code>match(x,y)</code>	Donner un vecteur de même longueur que <code>x</code> contenant les éléments de <code>x</code> qui sont dans <code>y</code> (<code>NA</code> sinon).
<code>which(x==2)</code>	Donner les indices des éléments de <code>x</code> égaux à 2.

<code>choose(n,k)</code>	Calculer les combinaisons de paramètres n et k.
<code>unique(x)</code>	Supprimer les éléments dupliqués de x.
<code>table(x)</code>	Créer un tableau des effectifs des différentes valeurs de x.
<code>sample(x,k)</code>	Ré-échantillonner aléatoirement et sans remise k éléments dans x.
<code>sample(x,k,REPLACE=TRUE)</code>	Ré-échantillonner aléatoirement et avec remise k éléments dans x.

- Manipuler des vecteurs logiques

<code>all(x)</code>	Tester si tous les éléments de x sont TRUE.
<code>any(x)</code>	Tester si au moins un des éléments de x est TRUE.

On peut utiliser les opérateurs arithmétiques, les FALSE étant transformés en 0, les TRUE en 1.

3.2 Les facteurs

Un facteur est constitué des valeurs de la variable catégorique correspondante et des différents niveaux possibles de cette variable (même ceux qui ne sont pas représentés dans les données).

3.2.1 Créer des facteurs

<code>factor(c(1,2,3))</code> , <code>factor(1 :3)</code>	Créer un facteur de valeurs 1,2,3 et de niveaux 1,2,3.
<code>factor(1 :3,levels=1 :5)</code>	Créer un facteur de valeurs 1,2,3 et de niveaux 1,2,3,4,5.
<code>factor(1 :3,labels=c("A","B","C"))</code> , <code>factor(c("A","B","C"))</code>	Créer un facteur de valeurs A,B,C, et de niveaux A,B,C.
<code>factor(1 :4,exclude=3)</code>	Créer un facteur de valeurs 1,2,NA,4 et de niveaux 1,2,4.
<code>gl(2,3)</code>	Créer un facteur de valeurs 1,1,1,2,2,2 et de niveaux 1,2.
<code>gl(2,3,length=30)</code>	Créer un facteur de valeurs 1,1,1,2,2,2, répétées 5 fois (longueur totale=30) de niveaux 1,2.
<code>gl(2,3, label=c("A","B"))</code>	Créer un facteur de valeurs A,A,A,B,B,B de niveaux A,B.

3.2.2 Manipuler des facteurs

<code>ordered(x)</code>	Transformer x en facteur ordonné.
<code>level(x)</code>	Extraire les niveaux du facteur x.

3.2.3 Utiliser des facteurs

Supposons que l'on dispose pour une population donnée contenant des hommes et des femmes (le sexe est donné par un facteur `sexe` contenant les valeurs M,F de niveaux M,F) de la taille de chaque individu (la taille est donnée par un vecteur numérique `taille` correspondant au facteur défini précédemment). On peut appliquer une fonction, par exemple la moyenne ou une fonction `mafonction` programmée, aux éléments de `taille` en fonction du sexe des individus à l'aide de la commande `tapply(taille,sexe,mean)` ou `tapply(taille,sexe,mafonction)`. Si l'on dispose d'un second facteur, par exemple `age`, on peut utiliser `taggregate(taille,list(sexe,age),mafonction)` qui appliquera `mafonction` aux éléments de `taille` en fonction des niveaux de `sexe` et de `age`.

3.3 Les matrices

Une matrice correspond à un vecteur auquel on ajoute un argument `dim` définissant le nombre de lignes et de colonnes. Ses colonnes sont donc de même classe.

Si x est une matrice, `dim(x)` retourne la dimension de la matrice x, `nrow(x)` le nombre de lignes, `ncol(x)` le nombre de colonnes. Si x est un vecteur, `dim(x)` retourne NULL.

3.3.1 Créer des matrices

<code>matrix(1,nr=2,nc=2)</code>	Créer une matrice 2×2 ne contenant que des 1.
<code>matrix(1 :6,2,3)</code>	Créer une matrice 2×3 remplie colonne par colonne par les valeurs du vecteur 1 :6.
<code>matrix (1 :6,2,3,byrow=TRUE)</code>	Créer une matrice 2×3 remplie ligne par ligne par les valeurs du vecteur 1 :6.
<code>x=1 :6 ;dim(x)=c(2,3)</code>	Idem à <code>matrix(1 :6,2,3)</code> .
<code>diag(3)</code>	Créer la matrice identité 3×3 .
<code>diag(c(1,2,3))</code>	Créer une matrice diagonale d'éléments diagonaux 1,2,3.
<code>diag(c(1,2,3),nr=3,nc=5)</code>	Créer une matrice 3×5 dont la première sous-matrice 3×3 est <code>diag(c(1,2,3))</code> .
<code>rbind(x,y), cbind(x,y)</code>	Juxtaposer les matrices x et y (côte à côte ou l'une au dessous de l'autre).

3.3.2 Manipuler des matrices

- Extraire une partie d'une matrice

<code>x[i,j]</code>	Extraire la valeur de la ième ligne et jème colonne de x.
<code>x[,j]</code>	Extraire la colonne j de x.
<code>x[i,]</code>	Extraire la ligne i de x.
<code>x[-c(1,5),]</code>	Supprimer les lignes 1 et 5 de x.

On peut, comme pour les vecteurs, extraire des éléments d'une matrice x à l'aide de vecteurs logiques.

- Les opérateurs et fonctions mathématiques

Les opérateurs disponibles pour les vecteurs sont également utilisables pour des matrices de même dimension, tout comme la plupart des fonctions mathématiques. Ils agissent élément par élément.

On peut appliquer ces opérateurs ou fonctions mathématiques à certains éléments d'une matrice seulement à l'aide de la commande `apply()` :

<code>apply(x,1,mean)</code>	Calculer la moyenne des éléments de x ligne par ligne.
<code>apply(x,2,mafonction)</code>	Appliquer <code>mafonction</code> aux éléments de x colonne par colonne.
<code>apply(x,c(1,2),mafonction)</code>	Appliquer <code>mafonction</code> aux éléments de x ligne par ligne et colonne par colonne.

Pour les matrices numériques, on dispose en plus des opérateurs de calcul matriciel classiques et de fonctions propres au calcul matriciel.

<code>x%*%y</code>	Calculer le produit matriciel de x par y.
<code>t(x)</code>	Donner la transposée d'une matrice x.
<code>diag(x)</code>	Extraire la diagonale de la matrice x.
<code>diag(x)<-0</code>	Remplacer les éléments de la diagonale de x par 0.
<code>solve(x)</code>	Calculer l'inverse de la matrice x.
<code>solve(A,b)</code>	Résoudre l'équation $Ax=b$.
<code>e<-eigen(x)</code>	Calculer les valeurs et vecteurs propres d'une matrice symétrique x.
<code>e\$val</code>	Retourner les valeurs propres de x.
<code>e\$vec</code>	Retourner les vecteurs propres de x.
<code>svd(x)</code>	Donner la décomposition en valeurs singulières de x.
<code>prod(svd(x)\$d)</code>	Calculer la valeur absolue du déterminant d'une matrice carrée x.

3.4 Les arrays

Les arrays sont des tableaux à k dimensions, généralisant les matrices qui sont en fait des arrays avec $k = 2$. Beaucoup des commandes disponibles pour les matrices se généralisent naturellement aux arrays.

3.5 Les data frames

Un `data.frame` est un tableau de données composé de un ou plusieurs objets de même longueur mais pouvant être de modes différents.

Un `data.frame` possède, comme les matrices, un attribut `dim`.

3.5.1 Créer des data frames

On verra plus loin que la lecture de données dans un fichier à l'aide de la commande `read.table()` crée automatiquement un `data.frame`.

La commande `data.frame(x,y,...)` permet de créer un `data.frame` contenant les objets `x,y...`. Les objets entrant dans la commande doivent être de même longueur, ou si un de ces objets est plus court, il est alors recyclé un nombre entier de fois suffisant.

On peut concaténer deux `data frames` à l'aide de `c(dataframe1,dataframe2)`.

3.5.2 Manipuler des data frames

<code>x=factor(c("M","F","F","M","F","F"))</code>	
<code>y=c(1.75,1.68,1.72,1.67,1.59,1.65)</code>	
<code>données=data.frame(y,x)</code>	Créer un <code>data.frame</code> contenant le vecteur <code>y</code> et le facteur correspondant <code>x</code> .
<code>summary(données)</code>	Afficher un résumé de <code>données</code> .
<code>données=data.frame(taille=y,sexe=x)</code>	Changer les noms de <code>x</code> et <code>y</code> .
<code>names(données)</code>	Afficher les noms des objets contenus dans <code>données</code> .
<code>données=data.frame(taille=y,sexe=x, row.names=paste("Individu",1:6,sep=""))</code>	Donner des noms aux colonnes et aux lignes.
<code>données\$taille, données[["taille"]]</code>	Extraire l'objet <code>taille</code> du <code>data.frame</code> <code>données</code> .
<code>données[[1]]</code>	Extraire le premier objet contenu dans <code>données</code> .
<code>attach(données)</code>	Rajouter <code>données</code> dans la liste des répertoires de travail en position 2. Enregistrer dans ce répertoire les objets <code>taille</code> et <code>sexe</code> du <code>data.frame</code> <code>données</code> .
<code>search()</code>	Vérifier la position de <code>données</code> .
<code>taille</code>	Retourner les valeurs de <code>données\$taille</code> .
<code>find(taille)</code>	Trouver le répertoire dans lequel se trouve l'objet <code>taille</code> .
<code>ls(pos=2)</code>	Afficher la liste des objets du deuxième répertoire, ici <code>données</code> .
<code>taille[sexe=="M"]</code>	Extraire les éléments de <code>taille</code> correspondant aux individus de sexe masculin.
<code>mafonction(taille[sexe=="M"])</code>	Appliquer <code>mafonction</code> aux éléments de <code>taille</code> correspondant aux individus de sexe masculin.
<code>detach(données)</code>	Détacher le répertoire <code>données</code> .
<code>lapply(données,mafonction)</code>	Appliquer <code>mafonction</code> à chacun des objets contenus dans <code>données</code> .
<code>apply(données,1,mafonction)</code>	Appliquer <code>mafonction</code> aux différentes lignes de <code>données</code> .

3.6 Les listes

Les listes sont des extensions des `data frames`. La plupart des commandes disponibles pour les `data frames` se généralisent donc naturellement aux listes.

3.7 Les ts

On ne détaillera pas ici les commandes propres aux séries temporelles. On précisera juste la commande permettant de créer de telles séries temporelles : `ts(data, start=, end=, frequency=, deltat=, ts.eps=, class, names)`.

4 Lire et enregistrer des données dans un fichier

4.1 Lire des données

R peut lire des données stockées dans des fichiers de différents formats à l'aide des fonctions `read.table`, `scan`, `read.fwf`.

Si l'on dispose par exemple d'un fichier nommé `données.dat` en format ASCII dans un répertoire `/home/étudiant/R/`, la commande `données=read.table("/home/étudiant/R/données.dat")` crée un `data.frame` nommé `données`. Les objets contenus dans `données` sont nommés par défaut `V1`, `V2`, etc. Plusieurs options peuvent être utilisées dont voici les valeurs par défaut :

```
read.table(file, header=FALSE, sep=" ", quote="\"", dec=".", row.names, col.names, as.is=FALSE,
na.strings="NA", colClasses=NA, nrow=-1, skip=0, check.names=TRUE, fill=!blank.lines.skip,
strip.white=FALSE, blank.lines.skip=TRUE, comment.char="#").
```

La fonction `scan` est plus flexible que `read.table` dans le sens où elle permet de spécifier le mode des variables.

La fonction `read.fwf` sert à lire des fichiers où les données sont dans un format de largeur fixée. Les options sont les mêmes que pour `read.table` sauf `widths` qui spécifie la largeur des champs.

4.2 Enregistrer des données

La fonction `write.table` écrit dans un fichier un objet, typiquement un `data.frame`. Les options par défaut sont les suivantes :

```
write.table(x, file, append=FALSE, quote=TRUE, sep=" ", eol="\n", na="NA", dec=".", row.names=TRUE,
col.names=TRUE, qmethod=c("escape", "double")).
```

Pour enregistrer des objets de n'importe quel type dans un fichier, on utilise la commande `save`, par exemple `save(x, y, z, file="messorties.dat")` sauvegarde les objets `x`, `y` et `z` dans le fichier `messorties.dat`. On pourra les charger en mémoire ultérieurement avec `load("messorties.dat")`.

4.3 Exercice

On cherche à étudier les données de Hoesmer et Lemeshow (1989) stockées dans la library MASS de R sous le nom de `birthwt`. Le but de cet exercice n'est pas de mener cette étude, mais d'apprendre à manipuler les données fournies. On verra l'étude complète dans un TP ultérieur.

1. Importer les données. De quelle classe sont-elles ? Quelles sont les classes des objets contenus dans `birthwt` ?
2. Transformer `low`, `race`, `smoke` en facteurs dans les données d'origine `birthwt`.
3. Transformer `pt1` en un facteur prenant la valeur 0 si `pt1=0`, 1 sinon. Faire la même chose pour `ht` et `ui`. Transformer `ftv` en un facteur de niveaux "0", "1" et "2+".
4. Éliminer toutes les données correspondant aux mères âgées de moins de 20 ans et de plus de 41 ans (compris).
5. Calculer la moyenne des poids de naissance pour chacun des différents facteurs (sauf `low`).
6. Calculer la moyenne des poids de naissance pour les classes de femmes âgées de 21 à 25 ans, de 26 à 30 ans, de 31 à 35 ans, de 36 à 40 ans.
7. Calculer la moyenne des poids de naissance pour 4 classes de femmes de même cardinal (à un près), rangées par âge croissant.
8. Retransformer le facteur `smoke` en entier. Que remarque-t-on ?

5 Les graphiques avec R

5.1 Les principales commandes graphiques

Les fenêtres graphiques sont nommées X11 sous UNIX et `windows` sous Windows. Dans tous les cas, on peut ouvrir une fenêtre graphique avec la commande `x11()`.

Un fichier graphique est ouvert à l'aide de `postscript("mesgraphiques.eps")` ou `pdf("mesgraphiques.pdf")`. Pour afficher la liste des dispositifs graphiques ouverts, on utilise `dev.list()`.

Les graphiques seront tracés dans le dispositif graphique actif. Pour connaître ce dispositif, on utilise `dev.cur()`. Pour le changer, on utilise `dev.set(i)` qui rend le *i*ème dispositif actif. La fonction `dev.off()` ferme le dispositif actif, `dev.off(i)` ferme le *i*ème dispositif.

On peut partitionner la fenêtre graphique active à l'aide des commandes `split.screen(x)` (*x* vecteur) ou `layout(m,widths=w,heights=h)` (*m* matrice, *w* vecteur, *h* vecteur).

Pour visualiser une fenêtre graphique partitionnée avec `layout`, on utilise `layout.show(n)` où *n* est le nombre de sous-fenêtres.

Les principales commandes graphiques sont données dans le tableau suivant.

<code>plot(x)</code>	Tracer le graphe des valeurs de <i>x</i> ordonnées sur l'axe des abscisses.
<code>plot(x,y)</code>	Tracer le graphe de <i>y</i> en fonction de <i>x</i> .
<code>sunflowerplot(x,y)</code>	Idem mais les points superposés sont dessinés sous forme de fleurs dont le nombre de pétales correspond au nombre de points.
<code>pie(x)</code>	Tracer un graphe en camembert.
<code>boxplot(x)</code>	Tracer le graphe en "boîtes et moustaches" de <i>x</i> .
<code>stripplot(x)</code>	Tracer le graphe des valeurs de <i>x</i> sur une ligne.
<code>interaction.plot(f1,f2,x,fun=mean)</code>	Tracer le graphe des moyennes de <i>x</i> en fonction des valeurs des facteurs <i>f1</i> (sur l'axe des abscisses) et <i>f2</i> (plusieurs graphes).
<code>coplot(x~y z)</code>	Tracer le graphe bivarié de <i>x</i> et <i>y</i> pour chaque valeur de <i>z</i> .
<code>matplot(x,y)</code>	Tracer le graphe bivarié de la 1ère colonne de <i>x</i> contre la 1ère colonne de <i>y</i> , la 2ème colonne de <i>x</i> contre la 2ème colonne de <i>y</i> ...
<code>pairs(x)</code>	Si <i>x</i> est une matrice ou un <code>data.frame</code> , tracer tous les graphes bivariés entre les colonnes de <i>x</i> .
<code>plot.ts(x), ts.plot(x)</code>	Tracer le graphe d'une série temporelle <i>x</i> en fonction du temps.
<code>hist(x,freq=T)</code>	Tracer un histogramme des fréquences de <i>x</i> .
<code>barplot(x)</code>	Tracer un histogramme des valeurs de <i>x</i> .
<code>qqnorm(x)</code>	Tracer les quantiles de <i>x</i> en fonction de ceux attendus d'une loi normale.
<code>qqplot(x,y)</code>	Tracer les quantiles de <i>y</i> en fonction de ceux de <i>x</i> .
<code>contour(x,y,z)</code>	Tracer des courbes de niveau.
<code>filled.contour(x,y,z)</code>	Idem mais les aires entre les contours sont colorées.
<code>image(x,y,z)</code>	Idem mais en couleur.
<code>persp(x,y,z)</code>	Idem mais en 3D.
<code>symbols(x,y,...)</code>	Dessiner aux coordonnées données par <i>x</i> et <i>y</i> des symboles (étoiles, cercles, boxplots...).

Pour ces fonctions, les options principales sont les suivantes :

<code>axes=TRUE</code> ou <code>FALSE</code>	Si <code>TRUE</code> , les axes et le cadre sont tracés.
<code>type="n", "p", "l", "b", "h", "s"...</code>	Précise le type de graphe dessiné. <code>type="n"</code> supprime le graphe.
<code>col="blue", col.axis, col.main...</code>	Précise la couleur du graphe, des axes, du titre...
<code>bg="yellow"</code>	Précise la couleur du fond.

<code>xlim=c(0,10),ylim=c(0,20)</code>	Précise les limites des axes.
<code>xlab="abscisse",ylab="ordonnée"</code>	Précise les annotations des axes.
<code>main="titre"</code>	Précise le titre du graphe.
<code>sub="sstitre"</code>	Précise le sous-titre du graphe.
<code>bty="n","o"...</code>	Contrôle comment le cadre est tracé. <code>bty="n"</code> supprime le cadre.
<code>cex=1.5,cex.axis,cex.main...</code>	Contrôle la taille des caractères.
<code>font=1,font.axis...</code>	Précise la police du texte.
<code>las="0"</code>	Contrôle comment sont orientées les annotations des axes.
<code>lty="1"</code>	Contrôle le type de lignes tracées.
<code>lwd=1.5</code>	Contrôle la largeur des lignes.
<code>pch="+","o"...</code>	Contrôle le type de symbole utilisé pour le tracé des points.
<code>ps=1.5</code>	Contrôle la taille en points du texte et des symboles.
<code>tck,tcl</code>	Précise la longueur des graduations sur les axes.
<code>mfc=c(3,2),mfrow=c(3,2)</code>	Partitionne le graphe en 3 lignes et 2 colonnes. Les figures sont remplies colonnes par colonnes ou lignes par lignes.

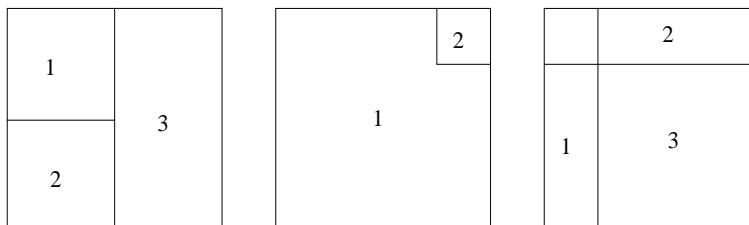
Les paramètres ci-dessus peuvent être précisés de manière permanente à l'aide de la commande `par()`.

Le tableau suivant donne un aperçu des fonctions graphiques secondaires (ayant une action sur un graphe existant).

<code>points(x,y)</code>	Ajouter le graphe des points y en fonction de x .
<code>lines(x,y)</code>	Ajouter le graphe de y en fonction de x en trait continu.
<code>text(x,y,labels)</code>	Ajouter le texte spécifié par <code>labels</code> aux coordonnées (x,y) .
<code>text(x,y,expression())</code>	Ajouter l'expression mathématique spécifiée par <code>expression()</code> aux coordonnées (x,y) .
<code>text(x,y,as.expression(substitute()))</code>	Ajouter une expression mathématique incluant la valeur d'une variable numérique aux coordonnées (x,y) .
<code>mtext(texte,side=3,...)</code>	Ajouter du texte dans la marge spécifiée dans <code>side</code> .
<code>segments(x0,y0,x1,y1)</code>	Tracer des segments joignant les points (x_0,y_0) aux points (x_1,y_1) .
<code>arrows(x0,y0,x1,y1,angle=30,code=1,2 ou 3)</code>	Tracer des flèches dont la pointe forme un angle de 30 avec l'axe aux points (x_1,y_1) , (x_0,y_0) ou aux deux respectivement.
<code>abline(a,b)</code>	Ajouter une ligne d'ordonnée à l'origine a , de pente b .
<code>abline(h=y)</code>	Ajouter une ligne horizontale sur l'ordonnée y .
<code>abline(v=x)</code>	Ajouter une ligne verticale sur l'abscisse x .
<code>abline(lm.obj)</code>	Ajouter la droite de régression donnée par <code>lm.obj</code> .
<code>rect(x1,y1,x2,y2)</code>	Tracer un rectangle délimité à gauche par x_1 , à droite par x_2 , en bas par y_1 , en haut par y_2 .
<code>polygon(x,y)</code>	Tracer un polygone reliant les points de coordonnées (x,y) .
<code>legend(x,y)</code>	Ajouter la légende au point de coordonnées (x,y) .
<code>title(main,sub)</code>	Ajouter un titre, un sous-titre.
<code>axis(side,vect)</code>	Ajouter un axe ou des graduations sur les axes.
<code>rug(x)</code>	Dessiner les données x sur l'axe des abscisses sous forme de traits verticaux.
<code>locator(n,type="n",...)</code>	Retourner les coordonnées (x,y) ou tracer des symboles aux points de coordonnées (x,y) après que l'utilisateur ait cliqué n fois sur le graphe.
<code>text(locator(1), "Point M")</code>	Utiliser la souris pour placer du texte.
<code>identify(x,y,labels)</code>	Mettre en évidence certains points du graphe à l'aide du bouton gauche de la souris, leur accoler du texte.

5.2 Exercices

1. Partitionner la fenêtre graphique active en trois, deux, trois parties de la forme :



2. Que donne la fonction `plot(x)` lorsque `x` est un vecteur numérique ? Une matrice ? Un facteur ? Un `data.frame` ?

3. Tracer un graphe en camembert, un graphe en "boîtes et moustaches", un histogramme, un graphe en barres à l'aide des commandes `pie`, `boxplot`, `hist`, `barplot` et commenter.

4. Les graphes tracés dans cet exercice seront enregistrés dans un fichier graphique.

Partitionner la fenêtre graphique active en deux. Dans les deux parties, tracer le graphe de la fonction $f(x) = \sin(2x) + 2e^{-0.4x} + 1$ sur $[0, 4\pi]$ représentant le signal (théorique) reçu par un radar en fonction du temps avec les options suivantes :

- Le graphe tracé en noir en ligne continue.
- L'axe des abscisses gradué de 0 à 5 de 1 en 1 puis de 2 en 2.
- L'axe des ordonnées gradué de 0 à 5 de 1 en 1.

Dans la première partie :

a) Ajouter l'annotation de l'axe des abscisses, le titre "Signal radar déterministe théorique" dans une font différente, et l'expression mathématique :

$$y = \sin(2x) + \frac{2}{e^{0.4x}} + 1,$$

près de la courbe (à l'aide de la fonction `locator`).

- b) Tracer en pointillés rouges les graphes des fonctions $g(x) = 2 \exp(-0.4x) + 2$ et $h(x) = 2 \exp(-0.4x)$.
- c) Repérer le minimum global de la fonction, le maximum global. Annoter ces points "Min global", "Max global".
- d) Tracer en rouge la tangente à la courbe représentative de f en un point de son choix. Préciser son équation sur le graphe (en rouge).

Dans la deuxième partie :

- a) Ajouter l'annotation de l'axe des abscisses, le titre "Signal radar observé" dans la même font que celle du titre de la première partie.
- b) Tracer un nuage de points aléatoires (y_1, \dots, y_{10}) représentés par des carrés et dont la loi est donnée par $y_i = \sin(2x_i) + 2e^{-0.4x_i} + 1 + \varepsilon_i$, où les x_i sont des points de $[0, 4\pi]$ régulièrement espacés, les ε_i des observations d'une variable aléatoire de loi $\mathcal{N}(0, 0.5)$.
- c) Repérer graphiquement les points les plus éloignés de la courbe à l'aide de la fonction `identify`.

6 La programmation avec R

6.1 Boucles et structures de contrôle

<code>for (i in vecteur) {listecommandes}</code>	Boucle sur les éléments de <code>vecteur</code> . Exécute les commandes contenues dans <code>listecommandes</code> pour chaque élément de <code>vecteur</code> .
<code>if (condition) {listecommandes}</code> <code>else {listealternatives}</code>	Structure conditionnelle (<code>condition</code> est un vecteur logique). Exécute les commandes de <code>listecommandes</code> si <code>condition</code> est <code>TRUE</code> , <code>listealternatives</code> sinon.
<code>while (condition) {listecommandes}</code>	Exécute les commandes de <code>listecommandes</code> tant que <code>condition</code> est <code>TRUE</code> .
<code>repeat {listecommandes}</code>	Boucle infinie couplée avec la commande <code>break</code> qui permet de sortir de la boucle.

À noter que l'utilisation de la vectorisation permet souvent d'éviter l'utilisation de boucles.

6.2 Les fonctions

Lorsque le programme écrit est assez long, il vaut mieux le découper à l'aide de fonctions. Avant d'exécuter le programme, on peut ainsi tester chaque fonction séparément.

Pour créer une fonction avec les arguments `x` et `y` en entrée et renvoyant les résultats `résultat1`, `résultat2` nommés `nom1`, `nom2`, on utilise la commande suivante :

```
mafonction<-function(x,y){ listecommandes return(nom1=résultat1,nom2=résultat2) }
```

Pour exécuter la fonction `mafonction` pour `x=3` et `y=1`, il suffit ensuite d'appeler `mafonction(3,1)`.

On peut également définir de nouveaux opérateurs binaires : `"%!"<-function(x,y){listecommandes}`. On utilise ensuite cet opérateur à l'aide de la commande `x%!y`.

6.3 Quelques commandes utiles

<code>print(x)</code>	Afficher l'objet <code>x</code> .
<code>assign(x,pos=1)</code>	Assigner l'objet <code>x</code> dans le répertoire en position 1.
<code>expression=parse(texte)</code>	Transforme le texte en expression (non évaluée).
<code>eval(expression)</code>	Évaluer <code>expression</code> .
<code>deparse(expression)</code>	Transformer l'expression en texte.
<code>substitute(expression)</code>	Substituer dans une expression les valeurs des variables.
<code>browser()</code>	Stopper l'exécution.

6.4 Exercices

- Reprendre les questions 5, 6 et 7 de l'exercice 4.3 sans utiliser les fonctions mathématiques pré-programmées.
- On veut recréer une fonction qui trace un histogramme. Les arguments d'entrée sont le vecteur des données `x` et un vecteur numérique `coupes`. En sortie, la fonction renvoie `NULL`. `x` est de longueur `n`.
 - Si le vecteur `coupes` est de longueur 1, alors il indique le nombre de coupures à l'intérieur de l'intervalle $[\min(x), \max(x)]$. Ces coupures seront régulièrement réparties. Si le vecteur `coupes` est de longueur supérieure à 1, les coordonnées de ce vecteur sont les points de coupure à l'intérieur de $[\min(x), \max(x)]$. Les coordonnées de `coupes` qui ne sont pas à l'intérieur de $[\min(x), \max(x)]$ devront être éliminées.
 - Pour chaque sous-intervalle i de $[\min(x), \max(x)]$, on calculera l'effectif n_i des éléments de `x` appartenant à cet intervalle i puis l'amplitude de l'intervalle l_i . On en déduira la fréquence des éléments de `x` $f_i = n_i/n$ ainsi que la fréquence corrigée $f_i^c = f_i/l_i$.
 - On tracera enfin l'histogramme correspondant avec la commande `barplot`.

7 Analyse statistique avec R : généralités

7.1 Formules et modèles

Une formule est un objet qui décrit un modèle. L'opérateur \sim sépare la variable à expliquer (vecteur ou matrice) des variables explicatives.

On utilise ces formules pour l'étude des modèles linéaires (fonctions `lm`, `aov`, `manova` ...), linéaires généralisés (fonction `glm`), non-linéaires (fonction `nlme`), les classifications (fonctions `lda`, `qda`, `rpart` ...) ...

7.1.1 Formules de base

<code>y~.</code>	Désigne le modèle de régression linéaire complet.
<code>y~1</code>	Désigne le modèle de régression linéaire constant.
<code>y~x1+x2</code>	Désigne le modèle de régression linéaire avec comme variables explicatives la constante, <code>x1</code> et <code>x2</code> .
<code>y~-1+x1+x2</code> ou <code>y~0+x1+x2</code>	Désigne le modèle avec <code>x1</code> et <code>x2</code> sans la constante.

7.1.2 Formules avec interactions

<code>y~x1+x2+x1 :x2</code> ou <code>y~x1*x2</code>	Désigne le modèle de régression linéaire avec la constante, <code>x1</code> , <code>x2</code> et <code>x1x2</code> .
<code>y~(x1+x2+x3)^2</code>	Désigne le modèle avec la constante et les interactions entre <code>x1</code> , <code>x2</code> et <code>x3</code> jusqu'à l'ordre 2.
<code>y~x1+x2%in%x1</code>	Dénote les hiérarchies. Idem à <code>y~x1+x1 :x2</code> .
<code>y~(x1+x2+x3)^2-x1 :x3</code>	Enlève dans le modèle <code>y~(x1+x2+x3)^2</code> le terme <code>x1x3</code> .

7.1.3 Formules avec transformations

<code>y~poly(x1,degree=2)</code>	Désigne le modèle avec des termes polynomiaux en <code>x1</code> jusqu'au degré 2.
<code>y~ log(x1)</code>	Désigne le modèle avec la constante et la variable <code>log(x1)</code> .
<code>y~x1>1</code>	Désigne un modèle avec la variable prenant la valeur 1 si <code>x1>1</code> , 0 sinon.
<code>I(x1^2)</code>	Permet de suspendre l'interprétation de \wedge comme opérateur formule et de revenir à sa signification arithmétique.

7.2 Facteurs et contrastes

Les modèles d'analyse de la variance (même les plus simples) sont surparamétrés. Pour régler ce problème, on impose une contrainte (ou un contraste) sur les facteurs mis en jeu.

On considère un facteur `fact` à I modalités et on désigne par α_i le coefficient du modèle associé au i ème niveau du facteur `fact`.

<code>contrasts(fact)</code>	Voir la valeur du contraste du facteur <code>fact</code> .
<code>contrast(fact)<-contr.sum(levels(fact))</code>	Affecter un contraste de type "somme" $\sum_{i=1}^I \alpha_i = 0$ au facteur <code>fact</code> .
<code>contrast(fact)<-contr.treatment(levels(fact))</code>	Affecter un contraste de type "traitement-témoin" $\alpha_1 = 0$.
<code>contrasts(fact)<-contr.poly(levels(fact))</code>	Affecter un contraste polynomial.

On peut changer la valeur du contraste de facteurs seulement ponctuellement en option dans une commande d'analyse de modèle :

```
objet=lm(y~fact1+fact2,contrasts=list(fact1="contr.sum",fact2="contr.poly"),data=donnees).
```

7.3 Modélisations

Pour plus de détails sur les fonctions `lm`, `aov`, `glm` ..., on renvoie à l'aide ou aux références bibliographiques données au début de cette introduction.

7.4 Exercice

1. Importer les données `tuchan.csv`. Les lire avec `read.table` puis `scan`. Que remarque-t-on ?

Ces données correspondent à l'analyse de la production de certains vignobles. On note pour la suite P , R et D les variables correspondant respectivement au poids net, au rendement estimé et au degré.

2. Étudier la régression linéaire de D en fonction de P et R ; en fonction de P , R et PR ; en fonction de P , P^2 , P^3 , R , PR ; en fonction de $\ln R$, $(P + R)$ et P^2 .

3. Étudier la régression de D en fonction de P , de la Commune et de l'interaction P -Commune. Que remarque-t-on ?
Changer la contrainte imposée sur le facteur Commune.