Introduction aux Statistiques et à l'utilisation du logiciel R

Christophe Lalanne* Christophe Pallier†

L'objet de cette séance de travaux pratiques est d'exposer les bases nécessaires de la théorie des statistiques dans un but clairement applicatif. Certaines notions mathématiques indispensables sont présentées dans un souci de clarté, mais ne sont nullement un pré-requis à une bonne compréhension de l'analyse de données et des statistiques. L'idée est d'aborder les outils statistiques indispensables à l'analyse de données au travers de la génération contrôlée de données (simulation). Cela permet d'une part de bien mettre en évidence les concepts clés d'échantillon et de population (ou distribution théorique), de variable aléatoire et de statistique ; d'autre part, la simulation peut être, par définition, répétée, ce qui permet de voir que l'on peut obtenir des résultats différents par le simple fait du hasard (l'erreur d'échantillonnage en expérimentation réelle). L'interprétation souvent avancée qu'il n'y a pas d'effet d'une condition expérimentale sur la performance doit absolument être écartée au profit d'une interprétation des effets en termes de taille des effets (notion d'importance de l'effet observé) et en termes probabilistes dans une optique inférentielle de généralisation (notion d'existence de l'effet parent).

1 R : un environnement et un langage pour les statistiques

1.1 Présentation de R

Les exemples sont développés avec le logiciel R disponible gratuitement sur le site CRAN, à l'adresse suivante : cran.r-project.org. Ce logiciel est comparable à `Matlab` à certains égards, mais se révèle beaucoup plus puissant dans le domaine des traitements statistiques. Concernant l'utilisation de R on trouve de nombreux documents de référence et d'aide en ligne sur le web. Les documents suivants peuvent ainsi se révéler très utiles :

– Notes on the use of R for psychology experiments and questionnaires

*christophe.lalanne@gmx.net
†www.pallier.org

– Using R for psychological research : A very simple guide to a very elegant package

Enfin, plusieurs sites contiennent des exemples d'utilisation de R pour l'analyse de données et la modélisation statistique, par exemple :

– www.lsp.ups-tlse.fr/Besse/enseignement.html
– pbil.univ-lyon1.fr/R
– www.pallier.org/ressources/

La plupart des librairies de base fournies avec R lors de son installation (`Windows` ou `Linux`) permettent d'effectuer la grande majorité des analyses auxquelles on est confrontés en sciences humaines. Certaines librairies additionnelles permettent cependant d'enrichir la « boîte à outils » du statisticien. Parmi celles-ci, citons :

– `car`, pour certaines fonctions utiles en régression ;
– `effects`, pour analyser les effets moyens sur des modèles linéaires (e.g. ANOVA) ;
– `lmtest`, pour les comparaisons de modèles linéaires ;
– `multcomp`, pour les comparaisons multiples ;
– `psy`, pour certaines fonctions utiles en psychométrie ;
– `ade4`, pour l'analyse exploratoire des données multidimensionnelles ;
– `xtable`, pour exporter les tables de résultats statistiques (e.g. un tableau d'ANOVA) au format LaTeX ;
– `nlme`, pour les modèles linéaires mixtes.

Bien que les exemples traités soient abordés sous un environnement `Linux`, ils sont naturellement tout à fait transposables aux plateformes `Windows` et `MacOS`.

1.2 L'environnement R

L'environnement de travail sous R est loin des interfaces traditionnelles, de type `Statistica` ou `SPSS`. On a généralement affaire à une ligne de commande (Fig. 1), ou un environnement un peu plus évolué comme sous `Windows` avec SciViews, ou `MacOS` avec le gui Aqua[1] pour R.

On peut interagir directement avec R en ligne de commande, ou bien éditer un fichier texte (avec l'extension `.R`) sous son éditeur préféré. Emacs possède un mode particulier, le mode ESS (ess.r-project.org), qui permet la coloration syntaxique et l'interaction avec un shell R.

Il existe également l'interface `RCommander` (Fig. 2), développée par J. Fox. L'usage d'une telle interface peut être intéressant dans un premier temps, pour se familiariser avec quelques commandes (graphiques en particulier), mais devient vite à l'usage contre-productif dans la mesure où l'on perd l'avantage de la souplesse du langage.

[1] A noter qu'il existe un « semblant » de ce type d'interface pour `Linux`, nommé JGR.

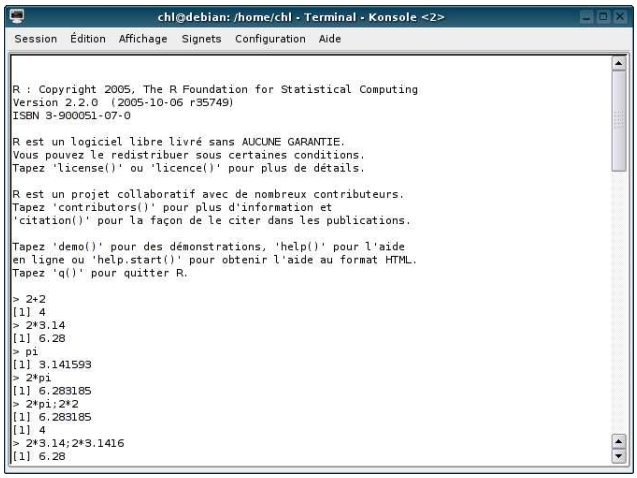

FIG. 1: R sur une console `Linux`

Avant toute chose, il est indispensable de savoir trouver de l'aide. Les commandes `help()` et `apropos()` sont là pour ça :

```
> help(t.test)
```

Suivant le système, cette commande ouvre une page de man (`Linux`) ou une fenêtre externe d'aide (`Windows/Mac`), qui renseigne sur la fonction indiquée, ici `t.test()`. La commande `?t.test` produit le même résultat. Si l'on ne connaît pas le nom exact de la fonction, on peut utiliser la fonction `apropos()` en lui indiquant le mot-clé voulu. Si on cherche la fonction permettant de réaliser le test de Wilcoxon pour deux échantillons indépendants, on peut taper :

```
> apropos('wilcox')
[1] "dwilcox"          "pairwise.wilcox.test" "pwilcox"
[4] "qwilcox"          "rwilcox"              "wilcox.test"
```

Si maintenant, on ne connaît pas le nom de la fonction, on peut utiliser `help.search()`, en lui donnant le mot-clé correspondant, par exemple

```
> help.search('wilcoxon')
```

Pour installer des modules ("packages") supplémentaires, on peut soit télécharger une archive au format `tar.gz` sur le site CRAN – il suffit alors de taper la commande `R CMD INSTALL package.tar.gz` en étant `root` – soit taper directement `install.packages('package')` sous R, comme on l'a vu pour le "package" `Rcmdr`. Par défaut, R ne charge pas au démarrage toutes les librairies installées sur le système ; pour utiliser des fonctions contenues dans une librairie particulière, par exemple `contr.Dunnett()`, il faut charger la librairie au préalable à l'aide de la commande `library(multcomp)`.

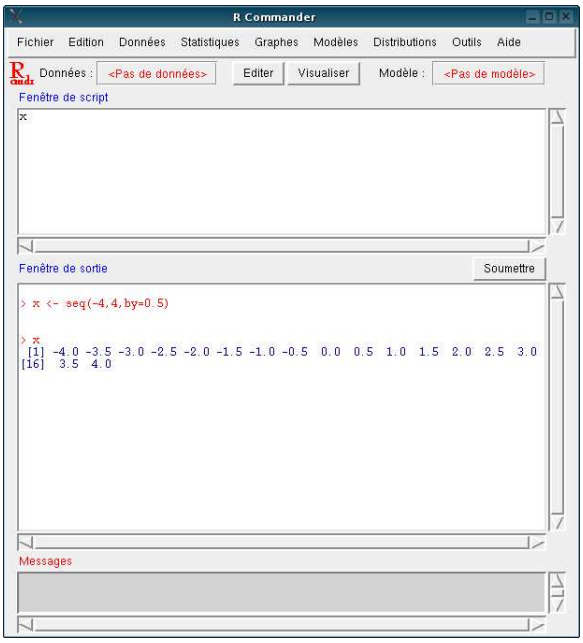

FIG. 2: L'interface RCommander.

Les données manipulées sous R sont stockées dans un espace de travail. Le répertoire de travail actuel peut être visualisé en tapant `getwd()`, tandis que l'accès à des répertoires se fait grâce à la commande `setwd()`. On peut à n'importe quel moment de la session visualiser le contenu de cet espace de travail à l'aide de la commande `ls()`, et supprimer une variable à l'aide de la commande `rm()`, en lui passant comme argument le nom de la variable. Pour supprimer l'ensemble des varaibles contenues dans l'espace de travail, on utilise la commande :

```
> rm(list=ls())
```

Pour quitter l'environnement, il suffit de taper `q()`. Comme on vient de le dire, le travail sous R s'effectue via une session, que l'on peut à tout moment sauver à l'aide de la commande `save.image()`. Au redémarrage, on remarquera la ligne

```
[Previously saved workspace restored]
```

En tapant `ls()`, on constatera que nos variables sont toujours présentes dans l'espace de travail (celui-ci a en fait été sauvé dans un fichier `.RData`, dans le répertoire de travail). La commande `history()` permet de lister les dernières commandes. Enfin, on peut sauver toutes les commandes ayant servi à analyser des données dans un fichier script avec l'extension `.R`, et taper `source('script.R',echo=T)`, depuis l'invite de commande, pour exécuter l'analyse. On peut également lancer ce script depuis une console, sans lancer

R, en tapant `R BATCH script.R` ; les résultats seront alors écrits dans le fichier `script.Rout`. La commande `sink()` permet également de sauver dans un fichier les résultats d'une analyse ; les commandes ne sont pas écrites dans le fichier :

```
> sink('monanalyse.txt',split=T)
> a=1:10
> mean(a)
[1] 5.5
> summary(a)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
> sink()
```

1.3 Types de base

R est un langage en ligne de commande (interprété), comme `Matlab` ou `Octave`, et peut manipuler de nombreux types de données à l'aide de commandes prédéfinies. On va s'intéresser aux principaux types d'objets, à savoir :

- les vecteurs
- les matrices
- les facteurs
- les listes d'objets

et voir quels sont les moyens de créer, manipuler et traiter de tels objets.

Notons que, comme dans tout langage, certains mots-clés sont *réservés* et ne peuvent être utilisés comme noms de variable ou de fonction :

```
FALSE  Inf  NA  NaN  NULL  TRUE
break  else  for  function  if  in  next  repeat  while
```

1.3.1 Les vecteurs

Le vecteur est le type de base de R. Un nombre est simplement un vecteur à un seul élément. Notons dans un premier temps que R peut être utilisé comme un simple calculateur :

```
> 2+2
[1] 4
> 2*3.14;2*3.1416
[1] 6.28
[1] 6.2832
> 2*pi
[1] 6.283185
```

Certaines constantes (e.g. `pi`) et fonctions (`exp`, `cos`, etc.) sont connues de R et peuvent être appelées directement. On peut également assigner une valeur à une variable à l'aide de l'opérateur `<-` (ou `=`) :

```
> x <- 10
```

Par défaut, R n'affiche pas le résultat de l'affectation, sauf si on place l'expression entre parenthèses :

```
> (x <- 10)
[1] 10
```

La variable peut ensuite être utilisée comme dans n'importe quel langage interprété :

```
> x+2
[1] 12
> y <- x+4
> y
[1] 14
```

Bien que nous n'ayons utilisé que des variables de type numérique, il existe d'autres modes de représentation des données (ou classes) dans un vecteur : `character`, `integer`, `logical`, `complex`, `list`. Enfin, les types `NULL` et `NA` jouent un rôle particulier puisqu'ils désignent respectivement l'absence de valeur (i.e. un ensemble vide) et le codage par défaut d'une valeur manquante.

Différentes fonctions permettent de générer facilement des vecteurs : `c`, `seq`, `rep`. Par exemple, on peut créer le vecteur `X = [1 2 3 4 5]` de différentes façons :

```
> X <- c(1,2,3,4,5)
> X
[1] 1 2 3 4 5
> rm(X)
> X <- seq(1:5)
> X
[1] 1 2 3 4 5
```

La fonction `seq()` est utile pour générer des suites d'entiers, et possède plusieurs arguments pour spécifier la séquence recherchée (voir `?seq`). On peut dupliquer un vecteur à l'aide de la commande `rep()` :

```
> Z <- rep(X, 2)
> Z
 [1] 1 2 3 4 5 1 2 3 4 5
```

La fonction `rev()` permet quant à elle de renverser l'ordre de la séquence (`1 2 3` devient `3 2 1`).

Les opérations arithmétiques vues précédemment pour les scalaires s'appliquent de la même manière sur des vecteurs. Par exemple, on peut additionner deux vecteurs `a` et `b`, et effectuer les produits scalaires ou membre à membre classiques :

```
> a <- 1:5
> b <- 5:9
> a*2
[1]  2  4  6  8 10
> a+b
[1]  6  8 10 12 14
> a*b
[1]  5 12 21 32 45

> a%*%b
     [,1]
[1,]  115
> a[5] == b[1]
[1] TRUE
```

De nombreuses fonctions permettent également de classer les éléments (ou les indices) d'un vecteur, de les sommer, etc.

```
> a <- c(1,3,2,7,4)
> a
[1] 1 3 2 7 4
> order(a)
[1] 1 3 2 5 4
> sort(a)
[1] 1 2 3 4 7
> sum(a)
[1] 17
> length(a)
[1] 5
```

Enfin, une particularité de R est que les éléments d'un vecteur peuvent avoir des noms. La fonction `names()` permet en effet d'associer une étiquette à chacun des élements d'un vecteur :

```
> x <- 1:5
> names(x) <- c("a","b","c","d","e")
> x
a b c d e
1 2 3 4 5
> v=c(1,2,3,4)
> names(v)=c('alpha','beta','gamma','delta')
> v['beta']
```

Cela s'avère très utile pour créer des dictionnaires. Par exemple, un vecteur `freq` donnant la fréquence d'usage des mots peut avoir les mots comme étiquette ; il suffit alors de taper `freq['aller']` pour obtenir la fréquence du mot `aller`,

```
> mots=c('aller','vaquer')
> freq=c(45,3)
```

```
> freq
> freq[mots=='aller']
> names(freq)=mots
> freq
> freq['aller']
```

1.3.2 Les matrices

Pour les matrices, on peut soit générer un vecteur de taille `n`, et le réarranger pour créer une matrice de taille `l*m`, soit directement créer une matrice à l'aide de la fonction `matrix()`, en spécifiant en arguments le nombre de lignes, par exemple :

```
> x <- seq(1:5)
> y <- x*2
> cbind(x,y)
     x  y
[1,] 1  2
[2,] 2  4
[3,] 3  6
[4,] 4  8
[5,] 5 10
> xy <- rbind(x,y)
  [,1] [,2] [,3] [,4] [,5]
x    1    2    3    4    5
y    2    4    6    8   10
> matrix(1:20, nrow=5, byrow=T)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
```

Les fonctions `cbind` et `rbind` permettent de manipuler des vecteurs de manière à former une matrice par concaténation sur les colonnes ou sur les lignes. Pour accéder aux éléments d'une matrice, c'est un peu différent de `Matlab` : il faut spécifier les indices entre crochets, et la virgule sert de délimitateur entre indices de ligne et de colonne :

```
> xy[,1]
x y
1 2
> xy[1,]
[1] 1 2 3 4 5
> xy[1,2]
[1] 2
```

Le produit matriciel s'effectue de la même manière que pour les vecteurs. Par exemple, si on a un tableau de contingence (les effectifs ventilés sur les

modalités des variables, ou, en d'autres termes, les fréquences d'association entre les modalités de deux variables qualitatives), on peut le coder sous forme disjonctive, c'est-à-dire associer à chacune des variables un tableau d'indicatrices[2] : le tableau de contingence n'est autre que le produit matriciel des deux tableaux d'indicatrices.

```
> x1 <- matrix(c(1,0,0,0,1,0,0,1,0,1,0,0,0,0,1,0,0,1,
               0,1,0,0,0,1),nrow=8,byrow=T)
> x2 <- matrix(c(1,0,1,0,0,1,1,0,0,1,0,1,1,0,1,0),
               nrow=8, byrow=T)
> x1; x2
> t(x1)%*%x2
     [,1] [,2]
[1,]    2    0
[2,]    2    1
[3,]    1    2
```

La fonction `array()` permet quant à elle de générer des matrices multidimensionnelles :

```
> a=array(1:20,dim=c(4,5,2))
```

Sous R, ce n'est pas le type de données le plus intéressant lorsque l'on travaille dans l'optique de modélisation ou de tests statistiques, et on va voir un type particulier de structure de données beaucoup plus utile.

1.3.3 Les facteurs

Il existe un type particulier de vecteur correspondant à la notion de facteur (ou de variable) en analyse de données et qui permet de créer une suite de valeurs que peut prendre le facteur considéré (les niveaux du facteur ou les modalités de la variable). On utilise pour cela les fonctions `factor()` et `gl()`. La première s'utilise pour convertir explicitement un vecteur (numérique ou de caractères) en un facteur, tandis que la seconde permet de générer directement un vecteur-facteur, en spécifiant l'agencement des niveaux. Voici deux façons de créer un même facteur :

```
> v <- c("A","B","A","B")
> f <- factor(v)
> f
[1] A B A B
Levels: A B
> fbis <- gl(2,1,4,labels=c("A","B"))
[1] A B A B
Levels: A B
> x <- c(1,3,2,4)
> x[f=="A"]
[1] 1 2
```

[2] prenant la valeur 1 lorsque la modalité est observée, 0 sinon

Les dernières commandes permettent de récupérer les valeurs de `x` pour lesquelles le niveau `A` est présent. De même, la fonction `subset()` se révèlera très utile lorsque l'on souhaitera accéder à une partie d'un vecteur (par exemple, les valeurs correspondant à un certain niveau d'un facteur ou au croisement entre les niveaux de plusieurs facteurs). Pour le même exemple, on pourrait taper directement

```
> subset(x,f=="A")
[1] 1 2
```

Pour voir les niveaux (numériques) d'un facteur, on peut utiliser `unclass()`. La fonction `gl()` se révèle très utile lorsqu'il s'agit de créer un grand vecteur (avec de nombreuses répliques), ou lorsque l'on veut contrôler précisemment l'agencement des niveaux du facteur. Notons que ces fonctions sont essentiellement utilisables pour des plans équilibrés en termes d'effectifs et de répliques. Ces deux fonctions permettent de générer des variables qualitatives nominales, tandis que la fonction `ordered()` permet de créer une variable ordinale, c'est-à-dire dont les modalités sont ordonnées. Enfin, on peut discrétiser une variable continue à l'aide de la fonction `cut()`.

1.3.4 Les listes d'objets

Un objet de type `list()` permet de regrouper dans une même structure de données des variables possédant un mode de représentation différent (e.g. `numeric` et `character`), et éventuellement de taille différente :

```
> b=list(alpha=1:3,beta=c('a','b','c','d'))
> b
$alpha
[1] 1 2 3

$beta
[1] "a" "b" "c" "d"
```

Ce type ne sera pas vraiment utilisé par la suite, mais il faut savoir qu'il est utile pour renvoyer des valeurs multiples lorsque l'on crée ses propres fonctions, et c'est ce type de données que R utilise lorsqu'il renvoie le résultat d'un test statistique, par exemple. On peut ainsi facilement obtenir la p-valeur d'un test t, sans afficher le résultat complet de l'analyse :

```
> x <- rnorm(10)+0.5
> res <- t.test(x)
> res$p.value
```

Le type `data.frame` est sans doute l'un des objets les plus importants dans R lorsque l'on souhaire effectuer des analyses statistiques (test t, ANOVA, régression, etc.). Il permet en fait de regrouper, ou concaténer, au sein d'un même objet : (i) le vecteur de la ou les variable(s) mesurée(s) (dépendante(s)),

(ii) les vecteurs de classification (facteurs), (iii) d'autres données appariées aux précédentes (e.g. nom ou n° du sujet). Les éléments du data.frame seront accessible à l'aide de l'opérateur `$` accolé au nom du data.frame. La seule contrainte est que les vecteurs doivent être de même taille. L'autre avantage de ce type d'objet est que la plupart des fonctions statistiques de R (`plot()`, `aov()`, etc.) sont à même de traiter ce type d'objet de manière spécifique.

Si l'on dispose des données brutes d'une expérience, par exemple la liste des temps de réaction par sujet et par condition, dans un seul vecteur, on peut créer un `data.frame` assez facilement. Pour illustrer cela, on va créer un vecteur de taille 100, 2 facteurs (A et B) avec 2 niveaux (a1/a2, b1/b2) et un vecteur de 5 sujets : chaque sujet aura passé à 5 reprises les 2×2 conditions de l'expérience, et on aura récolté 100 temps de réaction individuels (exprimés en ms).

```
> tr <- runif(100)*1000
> suj <- gl(5,20,100,labels=c(1:5))
> A <- gl(2,10,100,labels=c("a1","a2"))
> B <- gl(2,5,100,labels=c("b1","b2"))
> df <- data.frame(tr=round(tr,2),suj,A,B)
> df
        tr suj  A  B
1   140.79   1 a1 b1
2   773.64   1 a1 b1
3   771.82   1 a1 b1
4   668.62   1 a1 b1
5   393.96   1 a1 b1
6    95.01   1 a1 b2
7   765.79   1 a1 b2
8   919.87   1 a1 b2
9   211.76   1 a1 b2
...
```

Une fonction très utile associée à ce type de structure de données est la fonction `attach()` qui permet d'appeler directement les variables contenues dans le data.frame, sans utiliser l'opérateur `$`. Dans l'exemple précédent, pour avoir accès au vecteur `tr`, il faudrait taper `df$tr`. Mais si l'on fait `attach(df)` au préalable, alors il suffira de taper `tr` pour accéder aux élements du vecteur. La fonction `detach()` s'utilise pour libérer cette modification du chemin d'accès aux variables de R. Une autre solution est d'utiliser l'expression `with(df)`, ce qui évite les éventuels masquages multiples de nom de variable dans l'espace de travail. Le type `data.frame` se révèle également très utile lors de l'utilisation de fonctions de test statistique (`aov()` par exemple), puisque celui-ci peut être passé en paramètre de la fonction, et les vecteurs qu'il contient peut alors être indiqués par leur nom. Pour l'exemple précédent, on pourrait faire :

```
> aov(tr~A*B+Error(suj/(A*B)),data=df)
```

1.4 Structures de contrôle et programmation

Il est possible de recourir aux structures de contrôle habituelles :

- `for (initialisation) [instruction]`
- `if (instruction) [instruction]`
- `while (instruction) do [instruction]`

Elles sont utiles lorsqu'il s'agit de répéter une série d'opérations identiques sur des objets différents.

La possibilité de créer ses propres fonctions étend les fonctionnalités de base du langage R. La syntaxe est relativement simple, et s'apparente beaucoup à celle de `Matlab`. Par exemple, il n'existe pas de fonction standard permettant de calculer un écart-type non-corrigé[3]. On peut créer alors sa propre fonction :

```
ety <- function(x) {
  ss <- 0
  n <- length(x)
  for (i in 1:n) {
    ss <- ss + (x[i]-mean(x))^2
  }
  ety <- sqrt(ss/n)
  return(ety)
}
```

En `Matlab`, la même fonction s'écrirait ainsi :

```
function out = ety(x)
ss = 0;
n = length(x);
for i=1:n
  ss = ss + (x(i)-mean(x))^2;
end
out = sqrt(ss/n);
```

Mais on peut profiter du mode de représentation vectoriel des données et ainsi éviter le procédé d'itérations :

```
ety2 <- function(x) sqrt(sum((x-mean(x))^2)/length(x))
```

ou

```
ety2 <- function(data) {
  stopifnot(length(x)>0)
  b <- data-mean(data)
  return(as.vector(sqrt(b%*%b/length(b))))
}
```

[3] A noter que c'est valable pour d'autres logiciels (e.g. `Excel`, `Statistica`, `SPSS`) qui, par défaut, calculent toujours un écart-type de population, c'est-à-dire avec un dénominateur à $n-1$.

R ne possède pas non plus de fonction permettant de calculer l'erreur-type ($s/\sqrt{(n)}$). On peut en définir une de la manière suivante :

```
se <- function (x) { sd(x)/sqrt(length(x)) }
```

De même, on peut calculer une moyenne arithmétique après suppression des valeurs atypiques, i.e. supérieures à 2 écarts-type de la moyenne[4] :

```
clmean <- function (x) {
  m <- mean(x)
  d <- sqrt(var(x))
  threshold <- 2
  mean(x[(x-m)/d<threshold])
}
```

On pourra comparer les résultats des deux moyennes sur l'estimation de la tendance centrale d'un échantillon aléatoire normalement distribué de taille 100 :

```
  > a<-c(rnorm(100),5)
  > mean(a)
  > clmean(a)
```

Enfin, une autre mesure de tendance centrale communément utilisée est la médiane, qui est la valeur qui partage l'effectif en deux sous-effectifs égaux. Bien entendu, il existe une fonction `median` sous R. Pour se familiariser avec les structure de contrôle (branchement conditionnel de type `if...else`), on peut essayer de la reprogrammer soi-même, par exemple :

```
med<-function(x) {
  odd.even<-length(x)%%2
  if (odd.even == 0)  (sort(x)[length(x)/2]+sort(x)[1+ length(x)/2])/2
  else  sort(x)[ceiling(length(x)/2)]
}
```

1.5 Création et importation de données

On a déjà vu comment créer des vecteurs à l'aide de la commande `c()`. La fonction `scan()` permet de créer de la même manière un vecteur, soit en lisant une série de valeurs contenues dans un fichier, soit en saisissant ses éléments l'un après l'autre dans la console (il faut alors utiliser la syntaxe `valeurs <- scan('')`).

Il y a plusieurs méthodes pour importer des données contenues dans un fichier texte. La fonction `read.table()` est la plus utilisée. Elle permet de

[4] On notera que cette notion d'atypicalité en termes d'écarts-type n'est justifiée que dans le cas d'une distribution quasi-normale (i.e. sans asymétrie notable) ; le cas échéant, il vaut mieux considérer comme indicateur d'extrêmalité la distance à la médiane ($1.5\times(Q_3-Q_1)$), comme le fait par défaut la fonction `boxplot()`.

lire un fichier de données comportant éventuellement un en-tête (il faut alors spécifier l'option `header=T`) et des données présentées sous forme tabulaire. Les données peuvent ainsi être contenues dans une variable qui est ensuite transformée en `data.frame`, comme dans l'exemple suivant :

```
> a <- read.table('file1.dat', header=T)
> a.df <- data.frame(a$Score,as.factor(a$Group))
> names(a.df) <- c("Score","Group")
> summary(a.df)
> attach(a.df)
> summary(Score[Group==1])
> summary(Score[Group==2])
> tapply(Score,Group,sd)
```

Les options d'importation (type de délimiteur, codage des valeurs manquantes, etc.) sont nombreuses, comme on peut le voir dans l'aide en ligne en tapant `?read.table`. La librarie `foreign` fournit également des fonctions d'importation de données au format `Statistica`, `SPSS`, `SAS`, ou bien contenues dans des bases de données `ODBC` ou `SQL` (voir `library(help='foreign')`).

2 Distributions et lois de probabilités

2.1 Lois de probabilités

L'un des intérêts de R est que l'on peut générer très facilement des séquences de nombres suivant une distribution de probabilité (discrète ou continue) précise, entre autres : binomiale, χ^2,exponentielle, F de Fisher-Snedecor, log-normale, logistique, normale, poisson, t de Student, uniforme, Weibull. R peut non seulement générer de tels échantillons, mais il fournit également les tables de probabilités associées. On peut ainsi lire les quantiles de la loi normale (e.g. `qnorm(.95)`), ou bien obtenir les probabilités de dépasser une certaine valeur pour une loi de Student (e.g. `1-pt(1.81,df=10)`). De manière générale, pour chaque distribution on retrouve quatre fonctions préfixées par les lettres `d`, `p`, `q` et `r`[5], suivie du nom abrégé de la loi (e.g. `norm` pour la loi normale) et donnant respectivement :

- (d) les valeurs de la fonction de densité de la loi,
- (p) la probabilité (i.e. la surface sous la courbe) associée à cette fonction de densité,c'est-à-dire les valeurs de la fonction de répartition, pour un intervalle donné (e.g. $]-\infty, q_0]$),
- (q) la valeur correspondant à une probabilité donnée (c'est l'opération inverse au cas précédent[6]),
- (r) la génération d'un échantillon aléatoire de taille quelconque issu de cette loi.

[5]Il s'agit ici de distributions univariées, mais on dispose également des fonctions `dmvnorm`, `pmvnorm` et `rmvnorm` pour des distributions multivariées.

[6]On peut le vérifier en comparant les valeurs `qnorm(0.975)` et `pnorm(1.96)`.

Cela permet d'une part de faire des études de simulation, d'autre part de générer rapidement des vecteurs de données issues d'une certaine loi – comme la loi normale, très utilisée en sciences humaines et sociales – pour aborder certains concepts clés des statistiques.

C'est ce deuxième aspect que nous allons privilégier par la suite, en essayant d'introduire les notions de base de la statistique descriptive et inférentielle. En effet, pour cette dernière, il ne faut pas oublier que le concept clé est la notion de *distribution d'échantillonnage* (cf. [4], chap. 5), et que « faire » de l'inférence consiste à se prononcer sur la généralisation possible à la population parente d'un effet observé sur un échantillon. Effectuer un test d'hypothèse ne consiste en rien d'autre que regarder la position d'une statistique de test (e.g. une différence de moyenne pour le t de Student, un rapport de variance pour le F de Fisher-Snedecor) par rapport à une position de référence (typiquement celle correspondant à une probabilité de 5%), à partir de laquelle on considère qu'obtenir un tel échantillon relève du hasard (d'échantillonnage).

2.1.1 Loi uniforme

La fonction `runif()` permet de générer des séries de nombres aléatoires, tirés d'une loi uniforme ; on a donc un générateur de nombres pseudo-aléatoires « classique » :

```
> runif(5)
[1] 0.5948152 0.2722563 0.7051001 0.3123071 0.3180319
```

On pourrait ainsi créer un vecteur de nombres au hasard prenant comme valeur 0 ou 1, en écrivant par exemple :

```
> x <- runif(100)
> for (i in 1:length(x)) {
  if (x[i] <= 0.5) x[i]=0
  else x[i]=1
}
```

ou encore, en profitant de la souplesse de la syntaxe R

```
> x[x<=0.5] <- 0
> x[x>0.5] <- 1
```

On peut ensuite vérifier la proportion de 1 en tapant `length(x[x==1])/length(x)`. Mais en fait, on peut se passer de cet artifice puisque l'on peut générer directement des séquences de nombres tirés d'une loi binomiale.

2.1.2 Loi binomiale

La loi binomiale est dérivée de l'épreuve de Bernoulli qui modélise la probabilité p de succès pour un événement possédant deux résultats possibles, et généralise cette loi de l'alternative à k événements ($P(k) = C_n^k p^k (1-p)^{n-k}$).

L'exemple classique d'utilisation de ce type de loi est le jeu du pile ou face. Imaginons que vous disposiez d'une pièce et que vous vous demandiez si elle est baisée ou non. Vous prévoyez de la lancer 10 fois. A partir de quelle proportion relative d'essais face/pile considérerez-vous que la pièce est truquée ? Si la pièce n'est pas truquée, le nombre de « pile » suit une loi binomiale :

```
> plot(dbinom(0:10,rep(10,11),prob=1/2),type='h')
> hist(rbinom(100,10,.5))
> hist(rbinom(1000,10,.5))
> hist(rbinom(10000,10,.5))
```

2.1.3 Loi normale

La loi normale est fondamentale en statistique inférentielle. La fonction `rnorm` génère des nombres aléatoires distribués selon une loi normale. En augmentant le nombre d'échantillons générés (de 10 à 10000), on constate que la distribution des valeurs obtenues se rapproche de plus en plus d'une distribution normale continue, comme on peut le voir dans la figure 3 :

```
> s1=rnorm(10,mean=2)
> summary(s1)
> s2=rnorm(100,mean=2)
> summary(s2)
> s3=rnorm(10000,mean=2)
> summary(s3)
> par(mfrow=c(3,3))
> stripchart(s1,method='jitter',vert=T,pch=16)
> stripchart(s2,method='jitter',vert=T,pch=16)
> stripchart(s3,method='jitter',vert=T,pch='.')
> hist(s1)
> hist(s2)
> hist(s3)
> plot(density(s1))
> x=seq(-5,5,by=.01)
> lines(x,dnorm(x,mean=2),col=2)
> plot(density(s2))
> lines(x,dnorm(x,mean=2),col=2)
> plot(density(s3))
> lines(x,dnorm(x,mean=2),col=2)
```

2.1.4 Loi du χ^2

La loi du χ^2 est la loi que suit une somme de V.A. $\sim \mathcal{N}(\mu;\sigma)\,(i.i.d)$. Cette loi est importante car c'est la loi que suit une variance et c'est par conséquent sur elle que repose la distribution du F de Fisher-Snedecor (utilisé en ANOVA). En effet, la statistique de test F_{obs} n'est rien d'autre que le rapport de deux variances $\sim \chi^2(dl)$ avec leurs ddl respectifs.

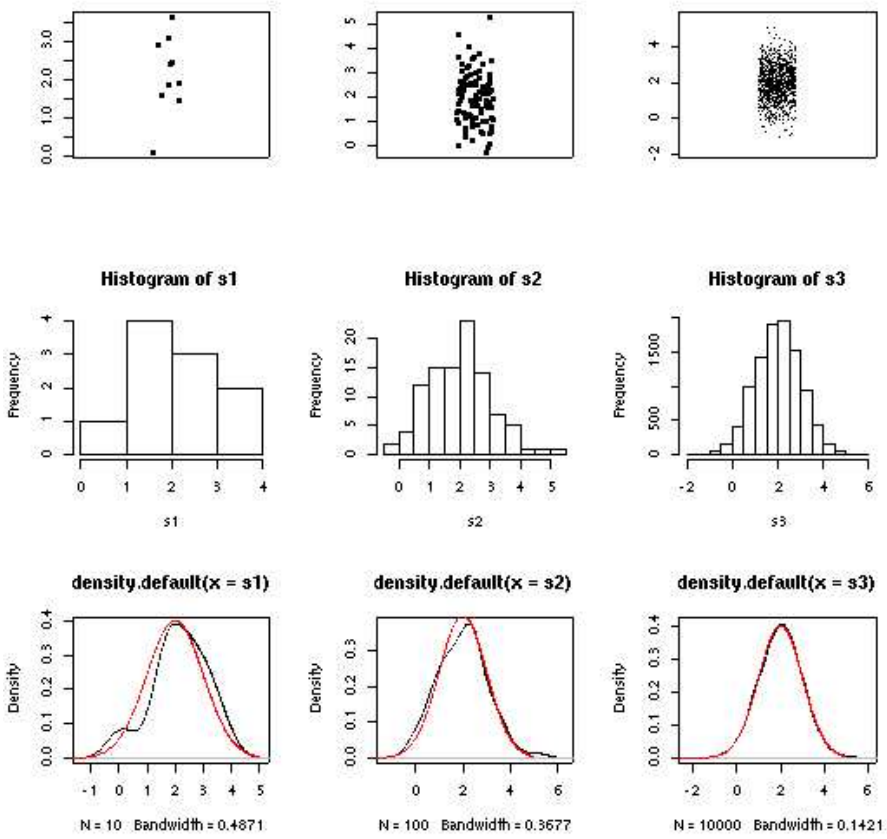

FIG. 3: Simulation de lois normales.

```
> for (i in 1:10) {
    plot(dchisq(1:100,i),ylim=c(0,0.4),xlim=c(0,50),type='l',col=i,lty=i)
    par(new=T)
}
> legend(40,0.35,legend=c(1:10),lty=c(1:10))
```

2.1.5 Loi de Student

La loi de Student à 9 ddl peut être visualisée à l'aide de la commande :

```
> x <- seq(-4,4,by=0.05)
> plot(x, dt(x,df=9), pch=20, xlab="x")
```

On notera que les queues de distribution de cette loi sont plus épaisses que celle de la loi normale, ce qui est mieux adapté aux petits échantillons dans lesquels les valeurs extrêmes sont plus fréquentes. Lorsque le nombre de degrés de liberté est très élevé, celle-ci peut être approximée par une loi normale (Fig. 5) :

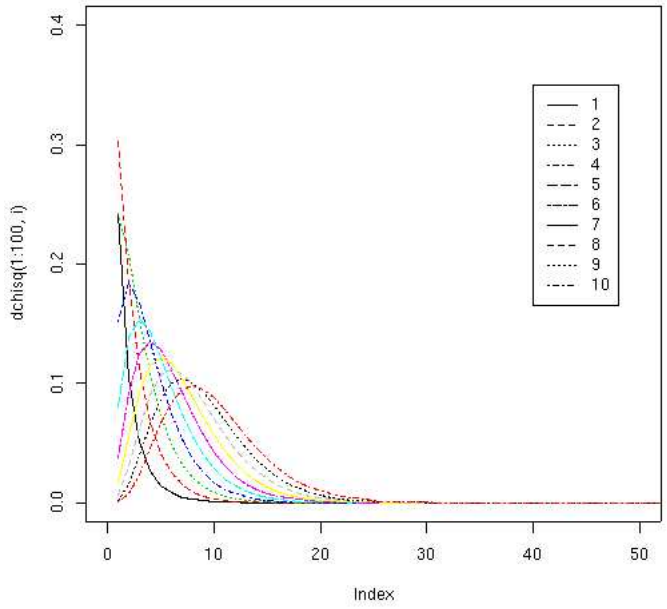

FIG. 4: Loi du χ^2 pour différents ddl.

```
> plot(x, dt(x,df=100), pch=20, xlab="x")
> curve(dnorm(x,mean=0), type="l", col="red", add=T)
```

On peut vérifier de la même manière que la loi de Student à 1 ddl est une loi de Cauchy (utiliser `dcauchy()`).

2.1.6 Remarques

Il ne faut pas voir les différentes lois de probabilités comme des « cas particuliers » sans lien de parenté entre eux. En fait, la majorité des lois discrètes et continues sont dérivées de quelques familles de lois. Par exemple, c'est à partir de la loi de Bernoulli que l'on peut dériver toutes les lois discrètes (loi de Pascal, loi de Poisson, etc.).

D'autre part, s'il existe un grand nombre de fonctions disponibles sous R, on peut avoir besoin de générer des distributions « non-standard ». Par exemple, on peut vouloir travailler avec une loi double-gaussienne. Rappelons qu'une telle loi est construite par le mélange de deux fonctions gaussiennes de même moyenne mais de variances différentes. Sa fonction de densité s'exprime sous la forme : $f(x;\mu,\lambda) = \alpha \times f(x;\mu,\lambda) + (1-\alpha)f(x;\mu,\frac{\lambda}{k})$, avec $0 < \alpha < 1$ et k une constante. Il suffit d'utiliser ce qui a été vu lors de la programmation de scripts et de fonction et d'écrire soi-même notre fonction, par exemple :

```
rdnorm <- function(n, mu=0, sd1=1, sd2=2, alpha=0.01)
  {
```

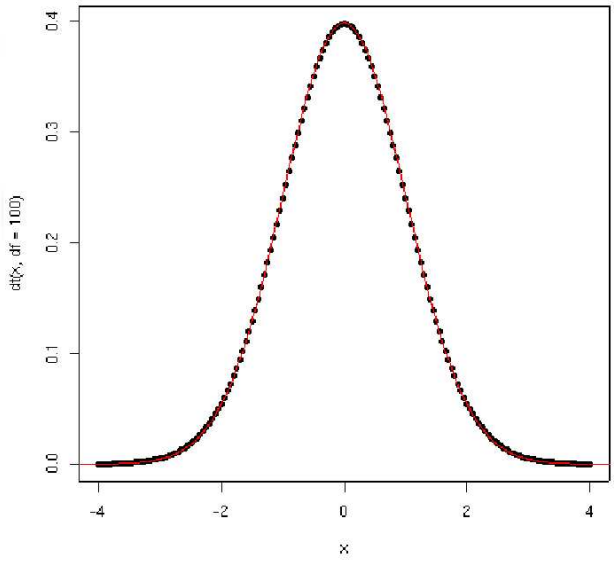

FIG. 5: Loi de Student T(100) et loi normale (en rouge).

```
  gs1 <- rnorm(n, sd=sd1)
  im1 <- trunc(alpha*n)
  gs2 <- rnorm(im1, sd=sd2)
  ind <- sample(n, im1)
  gs1[ind] <- gs2
  gs1
}
```

On peut ensuite tester que la fonction donne le résultat recherché (Fig. 6) :

```
> hist(rdnorm(100, sd1=1, sd2=10, alpha=0.3), breaks=50)
```

On peut générer beaucoup plus rapidement un échantillon issu d'un mélange de deux lois $\mathcal{N}(0;1)$, avec probabilité 0.95, et $\mathcal{N}(0;9)$, avec probabilité 0.05, à l'aide de la commande suivante :

```
> y <- rnorm(100,0,(1+2*rbinom(100,1,0.05)))
> hist(y)
```

Ce genre de procédure peut se révéler utile lorsque l'on souhaite générer des distributions asymétriques (pour étudier l'influence de l'asymétrie sur les estimateurs ou les tests statistiques, par exemple).

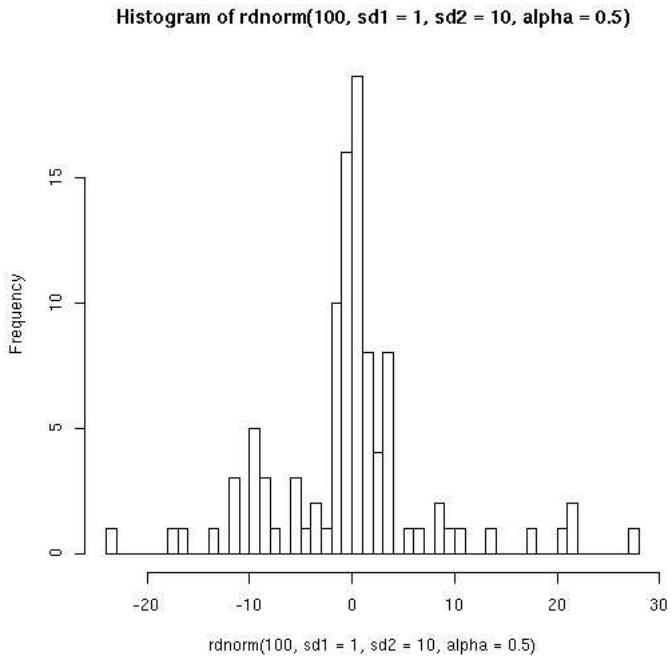

FIG. 6: Loi double-gaussienne $\mathcal{N}(0; 1, 10)$.

2.2 Positionnement d'une statistique

Si l'on prend comme exemple la distibution théorique de la taille des individus (de sexe masculin, français, et dans la tranche d'âge 20-35 ans), celle-ci suit une loi normale de moyenne 170 et d'écart-type 10. On peut donc non seulement situer un individu, ou un groupe d'individus, dans cette distribution, mais également évaluer la probabilité qu'un individu choisi au hasard parmi la population entière mesure moins de 185 cm, ou plus de 198 cm, ou ait une taille comprise entre 174 et 186 cm.

Lorsque l'on ne dispose pas des tables de lois normales $\mathcal{N}(\mu; \sigma^2)$ (il y en a une infinité puisqu'il y a 2 paramètres libres), on utilise la loi normale centrée-réduite $\mathcal{N}(0; 1^2)$ (encore appelée loi Z), dont la table est disponible dans n'importe quel manuel ou sur le web. Mais, l'avantage de R est de fournir directement les tables des lois normales, par l'intermédiaire de la commande `pnorm`, qui prend en arguments la valeur repère, la moyenne et l'écart-type théoriques. Ainsi, la probabilité qu'un individu choisi au hasard parmi la population entière mesure moins de 185 cm est de 0.933 ($P(X < 185)$), la probabilité qu'un individu mesure plus de 198 cm est de 0.003 ($1 - P(X < 198)$, et la probabilité que sa taille soit comprise entre 174 et 186 est 0.290 ($P(X < 186) - P(X < 174)$).

```
> taille=seq(130,210,by=1)
> plot(taille,dnorm(taille,mean=170,sd=10),type='b',col="red")
```

```
> pnorm(185,mean=170,sd=10)
> abline(v=185,col=4)
> text(185,.012,paste("P(X<185)=",signif(p,3)),col=4,pos=2,cex=.6)
> p=pnorm(198,mean=170,sd=10)
> abline(v=198,col=4)
> text(198,.002,paste("P(X>198)=",round(1-p,3)),col=4,pos=4,cex=.6)
```

Comme on le voit, la probabilité qu'un individu choisi aléatoirement dans une population de moyenne 170 ± 10 mesure plus de 198 cm est très faible. C'est en fait sur la base de ce calcul de probabilités que repose le test de typicalité, ou test Z : un groupe d'individus (i.e. un échantillon) sera déclaré atypique ou non représentatif de la population parente dont il est issu, lorsqu'il a une position au moins aussi extrême qu'une certaine position de référence, correspondant en général à la probabilité 0.05. C'est également la base des tests inférentiels usuels (test t, test F) : on calcule une statistique de test (une différence de moyennes empiriques pondérée par l'erreur standard associée à cette différence, dans le test t pour des échantillons indépendants) dont on compare la position par rapport à une position de référence[7].

2.3 Distributions conjointes

Si l'on reprend l'exemple précédent des tailles de la population française masculine (20-25 ans), on a une distribution similaire (i.e. suivant une loi normale de moyenne 70 et d'écart-type 7) pour les poids. On peut bien évidemment se poser les mêmes questions que précédemment, mais on peut également s'intéresser à la relation entre ces deux variables quantitatives. En représentant le poids en fonction de la taille, on peut évaluer la liaison linéaire entre ces deux variables à l'aide du coefficient de corrélation de Bravais-Pearson.

Pour illustrer cela, nous allons utiliser les données issues d'une population d'enfants de sexe masculin âgés de 11 à 16 ans.

```
> taille<-scan('')
1: 172 155 160 142 157 142 148 180 167 165
11:
Read 10 items
> poids<-scan('')
1: 50.5 38.1 57.3 39.3 46.1 37.1 45.9 66.3 60 50.5
11:
Read 10 items
> plot(poids~taille)
> r<-lm(poids~taille)             # modèle linéaire (x,y)
> summary(r)                      # diagnostic de la régression
```

[7] C'est l'approche classique de Fisher, mais on peut également évaluer directement la probabilité associée à cette statistique (approche de Neyman-Pearson).

```
> abline(r)                          # tracé de la droite de régression
> -55.1963626 + 175 * 0.6568411      # "prédiction" pour taille=175 cm
> predict(r,list(taille=c(175))
```

Ensuite, à partir de la connaissance de cette liaison linéaire, on peut se demander quelle serait le poids théorique (non observé) d'un individu dont on ne connaît que la taille : c'est le domaine de la régression linéaire. L'affichage des paramètres de la droite de régression donne la relation poids = 0.657 x taille - 55.196. Ainsi, on peut *prédire* que le poids d'un enfant mesurant 175 cm sera de 59.8 kg.

2.4 Estimation de densité

De nombreux logiciels de statistiques (e.g. `Statistica`, `SPSS`) proposent d'emblée lors de la création d'un histogramme la distribution normale approchée correspondante aux données unidimensionnelles. C'est ce que permettent de produire les quelques lignes de code qui suivent :

```
hist_norm <- function(x, xlab=deparse(substitute(x)),...) {
  h <- hist(x, plot=F, ...)
  s <- sd(x)
  m <- mean(x)
  ylim <- range(0,h$density,dnorm(0,sd=s))
  hist(x,freq=F,ylim=ylim,xlab=xlab, ...)
  curve(dnorm(x,m,s),add=T)
}
```

On peut également estimer directement la densité à l'aide de la fonction `density()`. C'est une méthode d'estimation non-paramétrique qui repose sur l'utilisation d'une fonction noyau pour estimer la fonction de répartition empirique. Le principe de base est simple : la fonction de répartition empirique n'étant pas dérivable (c'est une fonction en escalier), on essaye de l'approcher à l'aide d'une somme de fonctions noyaux (on pourrait le faire plus simplement avec des fonctions gaussiennes), de type $x = \mathbf{1}_{[-2;2]}x^2$. On a vu un exemple de son utilisation dans les paragraphes précédents (§ 2.1.3). Il existe également la fonction `fitdistr()` (`MASS`) qui fournit une estimation de la distribution de probabilité ([3]) :

```
> x <- rnorm(100,mean=5,sd=2)
> library(MASS)
> fitdistr(x,"normal")
      mean         sd
   5.2234888   2.1577120
  (0.2157712) (0.1525733)
```

2.5 Résumés numérique et graphique

L'objet de l'analyse descriptive est avant tout de décrire les données et de résumer (ou réduire) celles-ci à l'aide d'indicateurs caractéristiques.

Typiquement, le choix de ces indicateurs va dépendre de la nature de la variable considérée : il est clair qu'une moyenne portant sur des données qualitatives nominales (e.g. couleur des yeux : bleus, verts ou marrons) n'a pas de sens, pas plus qu'un écart-type associé à une moyenne n'a de sens si les données s'écartent trop fortement d'une distribution normale. On considère en général deux grandes mesures pour une distribution observée : l'une porte sur la tendance centrale (ou position) et l'autre sur la dispersion relative des données par rapport à cette mesure de tendance centrale.

On a vu de nombreux outils donnant des indications sur les principaux indicateurs descriptifs de position et de dispersion, comme par exemple `mean()`, `sd()` ou `median()`. La fonction `summary()` est sans doute la fonction la plus utile, car elle donne un résumé numérique des données passées en argument, en fonction du type de données (vecteur, facteur, etc.).

```
> x <- rnorm(100)
> summary(x)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-1.96700 -0.73770 -0.06421 -0.02392  0.70850  2.65300
```

Comme on peut le constater, il n'y a pas d'indication concernant la dispersion, excepté l'intervalle inter-quartile que l'on peut recalculer $(q_3 - q_1)$. On pourrait préférer une nouvelle fonction de résumé numérique incorporant plus d'information, comme :

```
describe <- function(x,digits=2) {
  valid <- function(x) {return(sum(!is.na(x)))}
  len=dim(x)[2]
  stats=array(dim=c(len,3))

  for (i in 1:len) {
    stats[i,1]=skew(x[,i],na.rm=TRUE)
    stats[i,2]=kurtosis(x[,i],na.rm=TRUE)
    stats[i,3]=valid(x[,i])
    stats[i,4]=median(x[,i],na.rm=TRUE)
  }
}
```

qui utilise les deux fonctions suivantes :

```
skew <- function (x, na.rm = FALSE) {
  if (na.rm)    x <- x[!is.na(x)]
  sum((x - mean(x))^3)/(length(x) * sd(x)^3)
}

kurtosis <- function(x, na.rm = FALSE) {
  if (na.rm)    x <- x[!is.na(x)]
  sum((x - mean(x))^4)/(length(x) * sd(x)^4)
}
```

Cette fois-ci, en appelant la fonction `describe()` au lieu de `summary`, pour des données numériques bien entendu, on aura un résumé un peu plus exhaustif.

Si le résumé numérique des données est une étape souvent utile pour analyser les caractéristiques générales d'un échantillon (mais cela ne doit pas faire oublier l'examen des données individuelles !), les représentations graphiques sont également très bénéfiques à la « lecture » des données. R permet de générer de nombreux graphiques (cf. Fig. 7), et ce document ne saurait prétendre à une liste exhaustive de toutes ses fonctionnalités graphiques. Les principales fonctions utilisées, que l'on a déjà vues en partie, sont sans doute :

```
plot()  hist()  boxplot()  curve()  interaction.plot()  mat.plot()
stripchart()  abline()  axis()
```

mais l'on ne saurait que trop recommander la lecture de la documentation en ligne et des différents documents accessibles sur le site du CRAN. Notonms que l'on peut « découper » la fenêtre de sortie graphique à l'aide de la commande `par()`, en spécifiant par exemple un découpage sur 2 lignes et 3 colonnes :

```
> par(mfrow=c(2,3))
```

Les graphiques seront affichés successivement dans chacune des 3 colonnes des 2 lignes. La commande `frame()` (voir l'aide en ligne) peut également être utilisée.

On peut sauvegarder les graphiques produits par R à l'aide de la commande `postscript()` ou `png()`, selon le format d'image désiré. Il ne faut pas oublier de libérer le flux grâce à la commande `dev.off()`. Par exemple, les commandes suivantes

```
> x <- runif(100)
> y <- runif(100)
> postscript('graphe_xy.ps')
> plot(x,y)
> dev.off()
```

génèrent un nuage de points sauvés dans le fichier `graphe_xy.ps` dans le répertoire de travail. De nombreuses options pour le format de l'image sont disponibles (tapez `?postscript` par exemple).

3 Estimation

3.1 Estimation ponctuelle

Le principe de l'estimation ponctuelle est d'estimer un paramètre d'une distribution (la moyenne d'une population par exemple)à l'aide d'une statistique appropriée (une moyenne empirique, ou d'échantillon). Bien que l'on

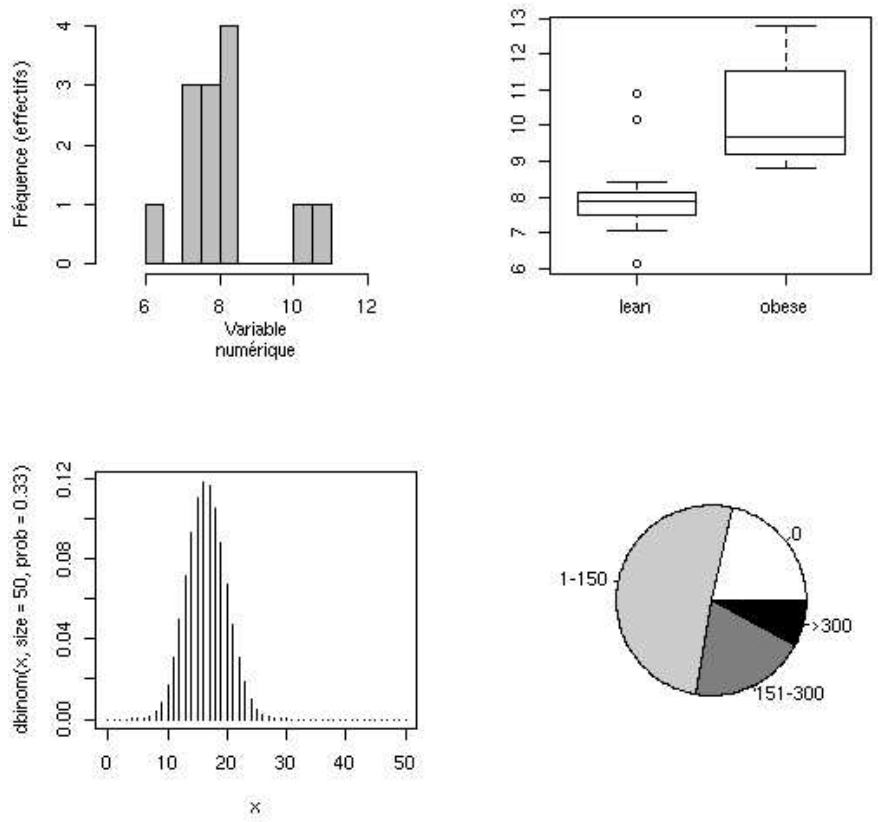

FIG. 7: Exemples de représentations graphiques sous R.

retienne le plus souvent la moyenne ou la médiane comme estimateur de tendance centrale, on présente dans les paragraphes qui suivent la notion d'estimateur, qui généralise l'usage des deux statistiques mentionnées.

3.1.1 Choix d'un estimateur

Il y a deux grandes classes d'estimateurs (de la tendance centrale par exemple) : les estimateurs classiques et les estimateurs robustes. La moyenne est un estimateur classique de la tendance centrale d'une distribution de variables aléatoires normalement distribuées. On peut montrer que c'est un estimateur sans biais et de variance minimale. Ce n'est pas le cas de tous les estimateurs, mais généralement on cherche toujours à optimiser l'un de ses deux critères (biais et variance).

Dans le cadre de l'estimation robuste, on cherche à construire des estimateurs pour de petits échantillons, qui minimisent cette erreur et/ou variabilité dans l'estimation. Pour illustrer leur intérêt, on peut comparer la

précision de différents estimateurs pour un même échantillon. Par exemple, on peut générer 300 n-échantillons $\sim \mathcal{N}(0;1)$ ($n = 100$), et calculer pour chacun de ces échantillons différents estimateurs de localisation : on pourra prendre la moyenne, la médiane, la médiane généralisée, le biweight, la tri-moyenne, la moyenne 0.10-censurée, et l'estimateur de Huber.

Exemple de solution : Pour générer les n-échantillons, et leur moyenne, on peut utiliser les commandes suivantes :

```
> gauss <- sapply(rep(100,300), rnorm)
> gauss.mean <- sapply(gauss, mean)
```

La moyenne alpha-censurée s'obtient simplement avec la fonction `mean` et l'option `trim=alpha` :

```
gauss.alpha-mean <- sapply(gauss, mean(trim=0.10))
```

Les estimateurs robustes peuvent être obtenus à l'aide des fonctions suivantes :

```
bmed <- function(x) # médiane généralisée
    {
      nx <- length(x)
      alpha <- 0.5-ifelse(nx>12, 2.5/nx, 1.5/nx)
      mean(x, trim=alpha)
    }

tri.mean <- function(x) # tri-moyenne
    {
      med <- median(x)
      uplow <- quantile(x, c(0.25, 0.75))
      t <- 0.25*(uplow[1]+2*med+uplow[2])
      t
    }

biweight <- function(x, cut=6, eps=0.001)  # biweight
    {
      tt <- median(x)
      nx <- length(x)
      uplow <- quantile(x, c(0.25, 0.75))
      cs <- 0.5*cut*(uplow[2]-uplow[1])
      repeat {
        u <- (x-tt)/cs
        les <- (1:length(x))[abs(u) > 1]
        w <- (1-u^2)^2
        w[les] <- rep(0, length(les))
        tp <- tt
        tt <- weighted.mean(x, w)
        if (abs(tt-tp) > eps)
```

```
        break
    }
    tp
}
```

L'estimateur de Hubert est défini dans R à l'aide de la fonction `huber()` de la librairie `MASS`.

3.1.2 Grands échantillons

Il est évident que plus l'échantillon sur lequel on travaille est grand, plus il a de chances d'être représentatif de la population parente dont il est tiré (cf Fig. 3). Si la distribution de cet échantillon ne s'écarte pas trop d'une distribution normale, la variance de cet échantillon sera elle-même plus petite (par rapport à un échantillon de petite taille), et la moyenne empirique sera un indicateur relativement fidèle de la moyenne de population. A ce titre, un résultat important de la théorie statistique est le fait que plus la taille de l'échantillon est grande, plus la moyenne empirique approche la « vraie » moyenne (loi faible des grands nombres).

Loi (faible) des grands nombres. Simuler plusieurs échantillons issus d'une loi normale $\mathcal{N}(100, 25)$, de tailles $n = 1 \dots 1000$. Calculer pour chacun sa moyenne. Représenter graphiquement l'évolution de la moyenne empirique en fonction de n. Qu'observe-t-on ?

Exemple de solution :

```
> n <- 1000
> x <- matrix(data = NA, nrow = n, ncol = n, byrow = T)
> kmean <- vector(mode="numeric",length=n)
> for (i in 1:n) {
    x[i,1:i] <- rnorm(i,mean=100,sd=25)
    kmean[i] <- mean(x[i,],na.rm=T)
  }
> plot(kmean,ylab="moyenne empirique",
       xlab="taille d'échantillon")
> abline(h=100, col="blue")
> text(x=800,y=105,expression(mu==100), col="blue")
```

ce qui permet de produire la figure 8. On peut également augmenter la taille de l'échantillon, e.g. $n = 5000$, mais le temps de calcul devient plus conséquent (en raison de l'utilisation de matrices de taille importante).

On constate clairement que la moyenne empirique converge vers la « vraie » moyenne, i.e. la moyenne de population. Plus la taille de l'échantillon est importante, plus l'écart entre la moyenne empirique et la moyenne théorique est réduit. On peut d'ailleurs visualiser l'évolution de cet écart à la moyenne théorique à l'aide des commandes suivantes (Fig. 9) :

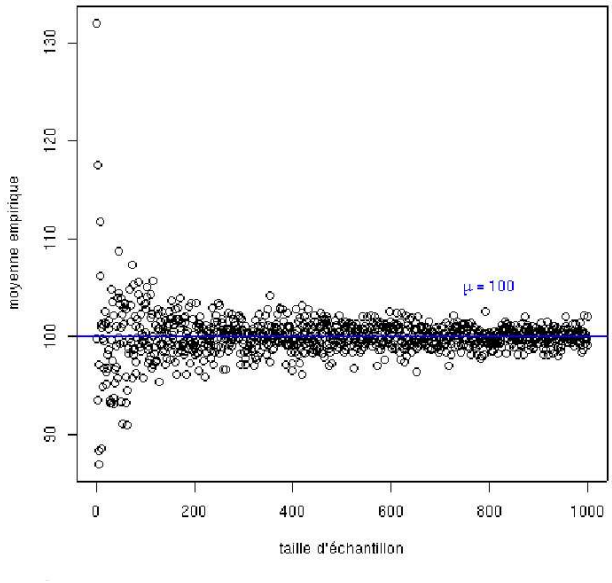

FIG. 8: Application de la loi (faible) des grands nombres.

```
> emean <- vector(mode="numeric",length=n)
> for (i in 1:n) {
    x[i,1:i] <- rnorm(i,mean=100,sd=25)
    emean[i] <- kmean[i]-mean(kmean)
  }
> plot(emean,type="l",ylab=expression(abs(bar(xn)-bar(X))),
       xlab="taille d'échantillon")
> elim <- mean(max(emean[500:1000]))
> abline(h=elim,col="blue",lty=2)
> abline(h=-elim,col="blue",lty=2)
```

3.1.3 Petits échantillons

Comme on a dit que la moyenne empirique est un estimateur non biaisé de la moyenne de population, elle peut également être utilisée dans le cas des petits échantillons ($n < 20$), avec les précautions d'usage, c'est-à-dire lorsque la distribution ne présente pas d'asymétrie notable. On peut également rechercher les « meilleurs » estimateurs de la tendance centrale d'une distribution empirique, i.e. générée à partir d'un échantillon observé (cf. 3.1.1). Néanmoins, un cas fréquent est la présence de valeurs extrêmes, ou atypiques, ou d'asymétrie dans la distribution, et qui justifie le recours à une autre statistique d'échantillon que la moyenne, par exemple la médiane. Des

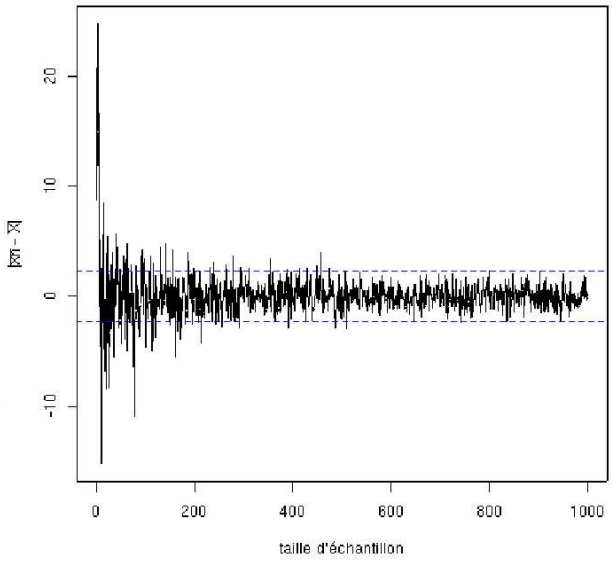

FIG. 9: caption

tests statistiques sont également disponibles pour ce type d'estimateur dans le cadre de la comparaison de deux échantillons. Une mesure de la précision d'un tel estimateur est ce que l'on appelle l'erreur standard.

3.2 Estimation par intervalle de confiance

Dans le cas de deux échantillons, l'intervalle de confiance de la moyenne peut être obtenu à l'aide de la fonction `t.test` :

```
a<-10+rnorm(10,sd=10)
t.test(a,conf.level=.01)
```

ou de la fonction `confint()` : dans ce cas, il faut passer en argument un modèle (e.g. `lm()` ou `glm()`) plutôt qu'une série de valeurs.

4 Théorie des tests

L'idée est ici de présenter la démarche générale de l'analyse statistique de différentes structures de données. L'approche ne se veut ni exhaustive (d'autres analyses sont possibles sur les mêmes jeux de données), ni détaillée sur le plan théorique. L'accent est mis sur la comparaison de moyennes (test t et ANOVA), et la régression est abordée au travers d'exemples simples.

Un rapide survol des analyses factorielles pour données qualitatives est également fourni.

4.1 Comparaison de deux distributions

On peut comparer une distribution empirique, observée sur un échantillon, à une distribution de référence à l'aide d'un test d'ajustement. C'est le cas, par exemple, lorsque l'on souhaite s'assurer de la « normalité » de données observées. Les plus fréquents sont les tests du χ^2 et le test de Kolmogorov-Smirnov (ce dernier peut être utilisé avec un ou deux échantillon(s)). Ils sont disponibles sous R grâce aux fonctions `chisq.test` et `ks.test()`. La fonction `qqplot()` est également largement utilisée pour comparer la distribution des résidus (en analyse de variance ou en régression, par exemple) à une distribution théorique normale.

```
> x <- rnorm(100)
> ks.test(x,"pnorm")
> y <- rnorm(100)+0.5*runif(100)
> ks.test(y,''pnorm'')
> ks.test(x,y)
> par(mfrow=c(2,1))
> qqnorm(x,sub="x")
> qqline(x)
> qqnorm(y,sub="y")
> qqline(y)
```

4.2 Comparaison de variances

La comparaison des variances de deux ou plusiers groupes d'observations est souvent nécessaire avant la mise en oeuvre d'un test t ou d'une analyse de variance, lorsque l'on s'intérese aux différences significatives entre des moyennes. En effet, cela n'a pas de sens de comparer des moyennes empiriques associées à une grande hétérogénéité des variances (hétéroscédasticité). Plusieurs tests sont disponibles, dont le test du rapport des variances `var.test` (test de Hartley) mais celui-ci est peu recommandé puisqu'il a tendance à ne rejeter l'hypothèse nulle que dans des cas « extrêmes »[8]. D'autre part, il est limité à la comparaison de deux échantillons. Il vaut mieux utiliser le test de Levene (`levene.test` (`car`)) ou le test de Bartlett (`bartlett.test()`) dans le cas où on 2 ou plusieurs échantillon(s).

```
> x1 <- rnorm(100,mean=0,sd=1)
> x2 <- rnorm(100,mean=0,sd=2.5)
> var.test(x1,x2)
> group <- gl(2,100,200)
```

[8] Par exemple, pour deux groupes de 20 sujets, la valeur de la statistique de test doit être supérieure à 2.12 au seuil .05.

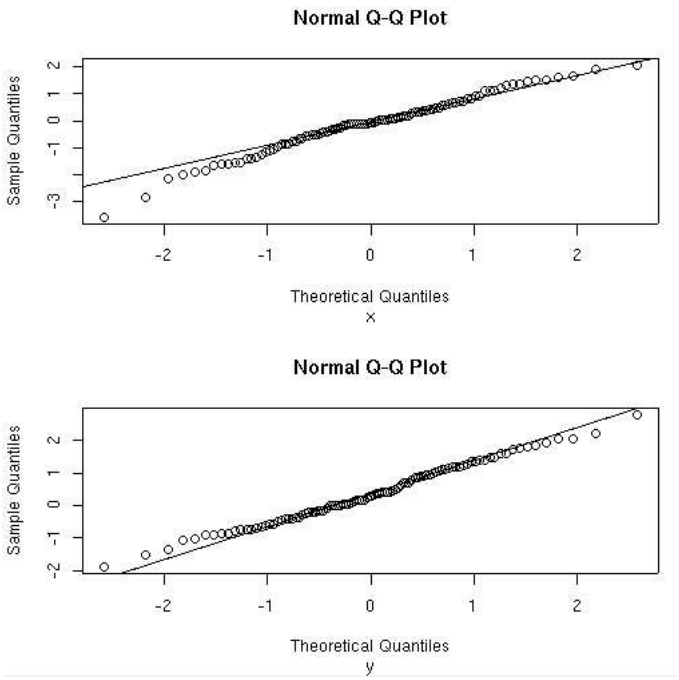

FIG. 10: Ex. 4.1

```
> x <- c(x1,x2)
> bartlett.test(x,group)
```

4.3 Comparaison de deux proportions

On peut comparer deux proportions à l'aide de la fonction `prop.test()`. Par exemple, si on a une promotion de 3310 candidats, dont 3270 hommes et 40 femmes, et que l'on observe une sélection de 4 femmes et 196 hommes à l'issue d'une épreuve de sélection, on peut se demander si la discrimination positive apparente en faveur des femmes (10 %, contre 6 % de sélection chez les hommes) est statistiquement significative([2], p. 84) :

```
> prop.test(c(4,196),c(40,3270))
```

4.4 Test t de Student

4.4.1 Deux échantillons indépendants

On a déjà vu à de nombreuses reprises comment générer des échantillons aléatoires tirés d'une distribution connue (e.g. binomiale, uniforme, normale). Pour aborder le concept de base de la comparaison de moyennes à l'aide du test t de Student, nous allons faire de même :

- générer deux échantillons aléatoires ;
- comparer leurs moyennes à l'aide d'un test t ;

– se prononcer sur la généralisation des différences observées à la population.

Allons-y :

```
> x <- rnorm(20)
> y <- rnorm(20)+5
> boxplot(x,y)
> t.test(x,y,var.equal=T)
```

On peut également tracer une boîte à moustache avec l'option `notch=T` : la forme des boîtes est alors trapézoïdale, et si les extrêmités de ces boîtes ne se touchent pas, il y a une forte probabilité que les médianes diffèrent. C'est utile lorsque l'on ne veut pas tester directement les moyennes en cas de valeurs atypiques ou de suspicion de non-normalité. Si les variances ne sont pas homogènes entre les groupes, alors on ne spécifie pas l'option `var.equal=T`, et R effectue un test t en ajustant les degrés de liberté : il s'agit du test modifié de Welch. Notons que celui-ci est aussi puissant que le test t classique (mais il ne donne génralement pas de ddl « interprétables »).

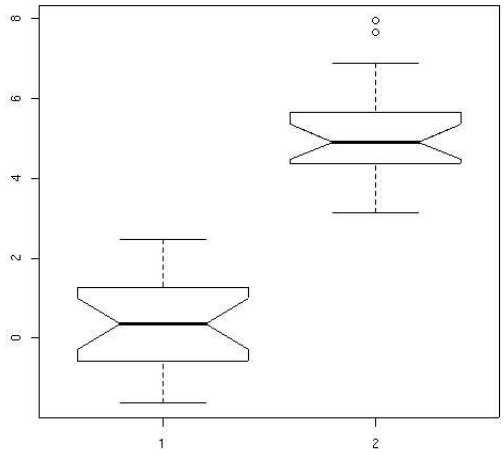

FIG. 11: Ex. 4.4.1 : `boxplot()` avec l'option `notch=T`.

On peut également effectuer un test non-paramétrique, de type test de Wilcoxon, accessible sous R grâce à la commande `wilcox.test()`. On peut d'ailleurs comparer ce que donnent les tests paramétrique et non-paramétrique sur le même jeu de données :

```
> a <- rnorm(10)
> b <- rnorm(10,mean=1)
> t.test(a,b)
> wilcox.test(a,b)
```

Une autre possibilité pour indiquer les échantillons de test aux fonctions `t.test()` ou `wilcox.test()` est d'utiliser une formule :

```
> c <- c(a,b)
> x <- gl(2,10,20)
> t.test(c~x)
> wilcox.test(c~x)
```

4.4.2 Deux échantillons indépendants avec mesures répétées

On se place dans le même schéma de groupes indépendants, mais avec cette fois-ci plusieurs mesures recueillies pour chacun des sujets.

```
> nsuj <- 40
> nmeasures <- 10
> effectsize <- 0.5
> subject <- gl(nsuj,nmeasures)
> group <- gl(2,nsuj*nmeasures/2)
> scores <- rnorm(nsuj*nmeasures)+rep(3*rnorm(nsuj),rep(nmeasures,nsuj))+
                ((0:(nmeasures*nsuj-1))%/%(nsuj*nmeasures/2))*effectsize
> data  <- data.frame(subject,group,scores)
> par(mfcol=c(1,2))
> stripchart(scores~subject,vertical=T,
        col=tapply(as.numeric(group),subject,max))
> plot.design(data)
> summary(aov(scores~group+Error(subject)))
```

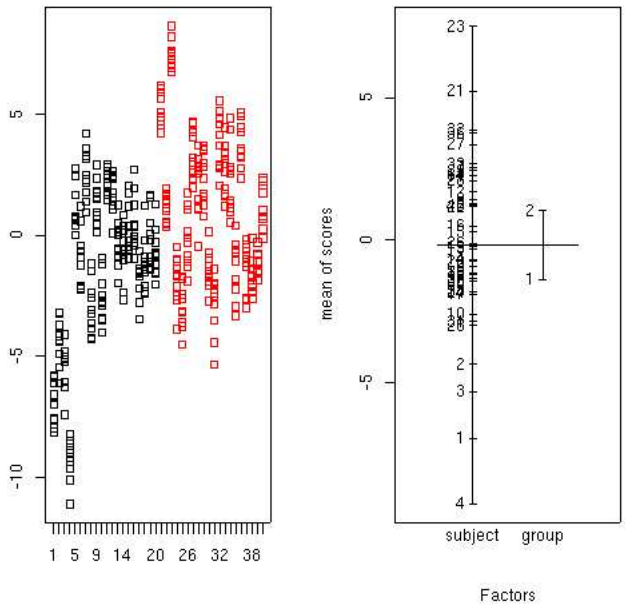

FIG. 12: Ex. 4.4.2.

Une approche plus moderne de la question consiste à considérer le facteur sujet comme un facteur aléatoire, et on a alors un modèle à effets mixtes (le facteur de groupe étant considéré comme un facteur à effets fixes) :

```
> library(nlme)
> l <- lme(scores~group,random=~1|subject)
> summary(l)
```

La sortie produite par la commande `lme()` diffère sensiblement de celle obtenue avec `aov()`, puisque ce sont des coefficients qui sont fournis et non une valeur F_{obs} (cf. 4.5.1, pour l'interprétation de ces coefficients dans le cadre d'un modèle linéaire général).

4.4.3 Deux échantillons appariés

Lorsque les mêmes sujets sont soumis à deux conditions expérimentales différentes, on parle d'échantillons appariés. Le principe général du test est le même que précédemment, mais il faut indiquer qu'il y a appariement des données en passant l'argument `paired=T` à la fonction `t.test()` :

```
> a <- rnorm(10)
> b <- rnorm(10)+2*runif(10)
> plot(a,b,pch=19)
> abline(coef=c(0,1))
> boxplot(a,b)
> t.test(a,b,paired=T)
```

Pour le test non-paramétrique (Wilcoxon-Mann-Whitney), il suffit également de passer l'argument `paired=T`. La procédure repose alors sur un classement par rang des différences signées. Enfin, notons que le test t pour échantillons appariés est strictement équivalent à un test t classique sur les différences des mesures que l'on compare à la valeur théorique 0.

```
> c <- b-a
> t.test(c,mu=0)
```

On pourrait aussi utiliser l'analyse de variance pour traiter ce type de données :

```
> nsuj <- 20
> subject <- gl(nsuj,2)
> cond <- gl(2,1,2*nsuj)
> effectsize <- 2
> scores <- (as.numeric(cond)-1)*effectsize+rnorm(2*nsuj)
> data <- data.frame(subject,cond,scores)
> par(mfcol=c(1,2))
> plot.design(data)
> interaction.plot(cond,subject,scores)
> t.test(scores~cond,paired=T)
> summary(aov(scores~cond+Error(subject/cond)))
```

On pourra vérifier que dans le cas de deux échantillons, la valeur du F_{obs} obtenue dans le modèle d'ANOVA est le carré de la statistique t_{obs}.

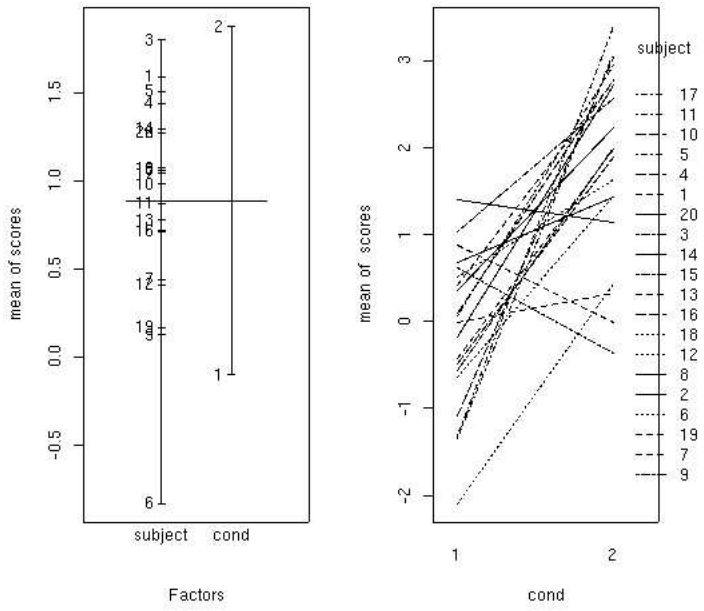

FIG. 13: Ex. 4.4.3.

4.4.4 Deux échantillons appariés avec mesures répétées

La situation est identique à la précédente, mais nous avons cette fois plusieurs observations pour chaque individu dans chacune des deux conditions expérimentales.

```
> nsuj <- 20
> nmeasures <- 10
> subject <- gl(nsuj,2*nmeasures)
> trial <- gl(nmeasures,2,2*nsuj*nmeasures)
> cond <- gl(2,1,2*nsuj*nmeasures)
> effectsize <- 1
> scores <- rnorm(2*nsuj*nmeasures)+(as.numeric(cond)-1)*effectsize
> data <- data.frame(subject,cond,scores)
> par(mfcol=c(1,2))
> plot.design(data)
> interaction.plot(cond,subject,scores)
> summary(aov(scores~cond+Error(subject/cond)))
> data2 <- aggregate(scores,list(subject=subject,cond=cond),mean)
> t.test(x~cond,data=data2,paired=T)
> library(lattice)
> bwplot(scores~cond|subject)
```

Encore une fois, on peut se ramener au cas classique du test t pour échantillons appariés en moyennant les observations individuelles par condition, à l'aide de la fonction `aggregate()`.

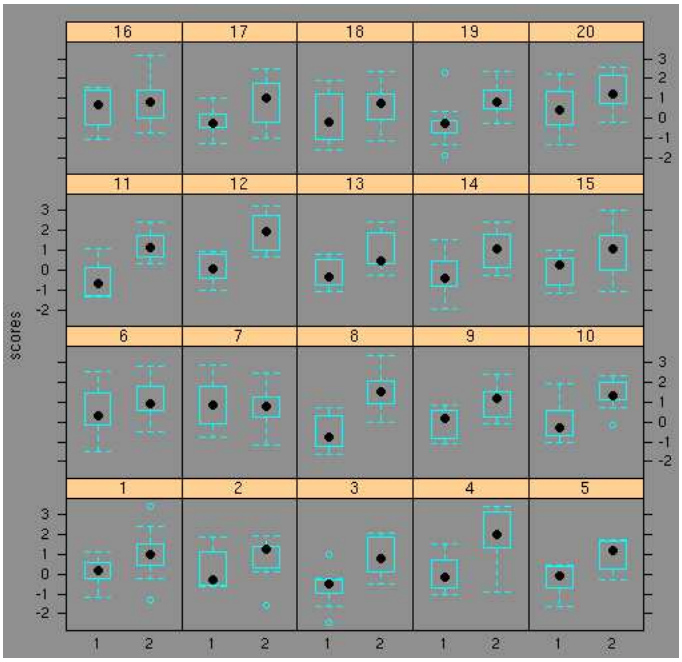

FIG. 14: Ex. 4.4.4.

4.5 Généralisation à k groupes : l'Analyse de Variance

4.5.1 ANOVA sur un facteur

Lorsque l'on est en présence d'un ensemble de k observations indépendantes (un seul facteur inter-sujets), on peut comparer leurs moyennes respectives à l'aide de la fonction `aov()` (ou selon un modèle linéaire général, avec les fonctions `lm` ou `lme`).

```
> ns <- c(15,28,10,20,35)
> n <- length(ns)
> group <- factor(rep(1:n,ns),labels=paste("g",1:n,sep=""))
> data <- rnorm(length(group),mean = 100 + (as.numeric(group)-2)^2)
> par(mfrow=c(2,2))
> plot(data~group)
> stripchart(data~group,method="jitter",vertical=T)
> plot.design(data~group)
> l <- aov(data~group)
> summary(l)
> pairwise.t.test(data,group)
> hsd <- TukeyHSD(l, "group", ordered = FALSE)
> print(hsd)
> plot(hsd)
```

Notons que R affiche directement un graphique en boïte à moustaches lors de l'appel à `plot()` en raison du facteur de classification. Le tableau d'ANOVA,

donné par la fonction `summary()` ou `anova()`[9] appliqué au modèle spécifié, peut être exporté directement au format LaTeX grâce à la fonction `xtable` (`xtable`). Les comparaisons multiples, ici grâce à la fonction `pairwise.t.test()`, repose par défaut sur la méthode de Holm mais on peut utiliser la correction de Bonferroni (correction des p-valeurs en fonction du nombre de comparaison pour que α reste fixé à 0.05) en utilisant l'option `p.adj="bonf"`. Le package `multcomp` fournit un ensemble de procédure élaborées pour les comparaisons multiples, mais on peut également utiliser le test post-hoc de Tukey standard.

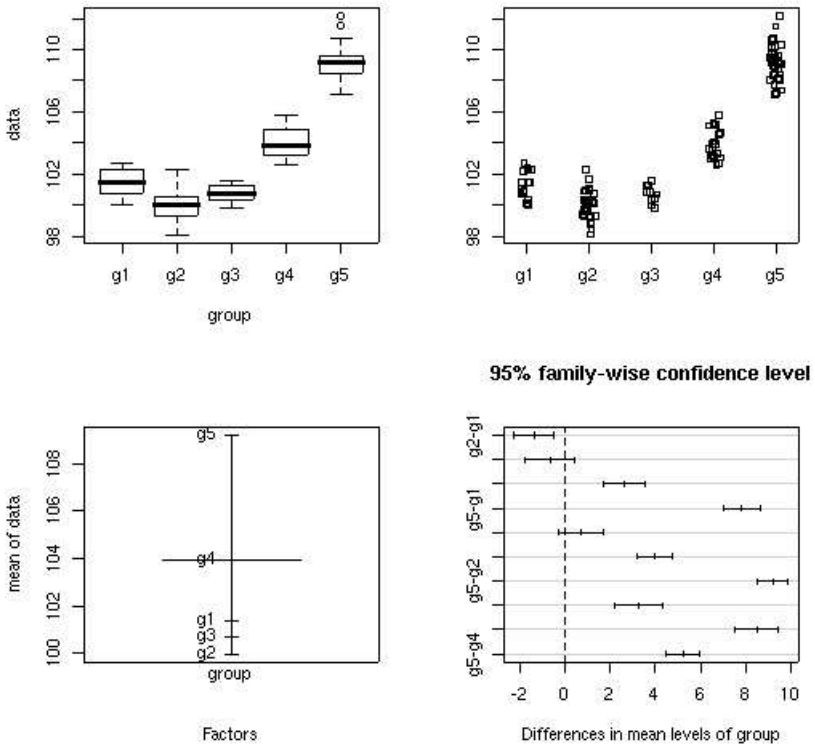

FIG. 15: Ex. 4.5.1.

On pourrait également utiliser la méthode des contrastes pour les comparaisons (planifiées ou non, dans ce cas).

Enfin, on est souvent intéresser par la taille des effets en fonction des niveaux du facteur, ce que ne nous donne pas la sortie produite par `aov()`. La fonction `model.tables()` donne un résumé de ces effets, en indiquant la différence des moyennes de groupe avec la moyenne générale, mais on peut

[9] Cette fonction ne marchera pas pour des modèles complexes, comme dans l'ANOVA.

tester directement l'effet des niveaux sur la variable mesurée. Pour cela, il faut utiliser la fonction `summary.lm()` :

```
> summary.lm(l)
```

qui produit un tableau de coefficients semblables à ceux que l'on obtiendrait à l'issue d'une régression. On obtiendrait la même chose en tapant `summary(lm(data~group))`, c'est-à-dire en effectuant un modèle de régression sur la variable catégorielle. L'interprétation de ces coefficients n'est pas instantanée, mais demeure relativement simple : le facteur de groupe possédant 5 niveaux, on a 5 variables explicatives catégorielles, dont les niveaux sont codés 0 sauf pour le niveau associé à la variable expliquée (ici, `data`) qui est codé 1. Sand rentrer dans les détails, le premier coefficient (noté `intercept`) représente la moyenne du premier groupe et le deuxième coefficient la différence entre la moyenne du deuxième groupe et celle du premier groupe (on peut le vérifier avec `mean(data[group=='g2']) - mean(data[group=='g1'])`). En résumé, quelque soit le nombre k de niveaux du facteur, le modèle général est

$$y \sim a_0 + \sum_{i=1}^{k} a_i x_i$$

où a_0 représente la moyenne du premier niveau, et les a_i des différences de moyennes entre les autres niveaux et le premier niveau[10].

4.5.2 ANOVA sur deux facteurs inter

Avec deux facteurs inter-sujets, le principe d'analyse est le même, mais on étudie également l'interaction entre les deux facteurs.

```
> x1 <- rnorm(25)+0.5*runif(25)
> x2 <- rnorm(25)+0.25*runif(25)
> x3 <- rnorm(25)+1.5*runif(25)
> x4 <- rnorm(25)+2.5*runif(25)
> x <- c(x1,x2,x3,x4)
> a <- gl(2,50,100)
> b <- gl(2,25,100)
> levels(a) <- c("a1","a2")
> levels(b) <- c("b1","b2")
> par(mfcol=c(1,3))
> plot(x~factor(a:b))
> gmean <- tapply(x,list(a,b),mean)
> barplot(gmean,beside=T,ylim=c(-0.5,1.5),legend.text=rownames(gmean),
          ylab="Score",col=c("grey50","grey80"))
> interaction.plot(a,b,x)
> l<-aov(x~a*b)
```

[10] Sous R, c'est le premier niveau dans l'*ordre alphabétique* qui est considéré comme ordonnée à l'origine.

```
> anova(l)
> effects <- model.tables(l,se=T)
> xx <- 2
> yy <- mean(x)
> lwb <- yy-effects$se[3]
> upb <- yy+effects$se[3]
> points(xx,yy,pch=16)
> arrows(xx,lwb,xx,upb,length=.05,angle=90,code=3)
```

Ce modèle saturé peut être simplifié, en omettant le terme d'interaction (si celle-ci n'est pas significative, ou si elle n'est pas pertinente dans le cadre des recherches) ; dans ce cas, il faut spécifier la formule `x~a+b`. En fait, la notation `x~a*b` est un raccourci pour l'expression du modèle linéaire incluant les effets des deux facteurs (effets principaux) et l'effet d'interaction, et l'on peut très bien le spécifier sous la forme `x~a+b+a:b`.

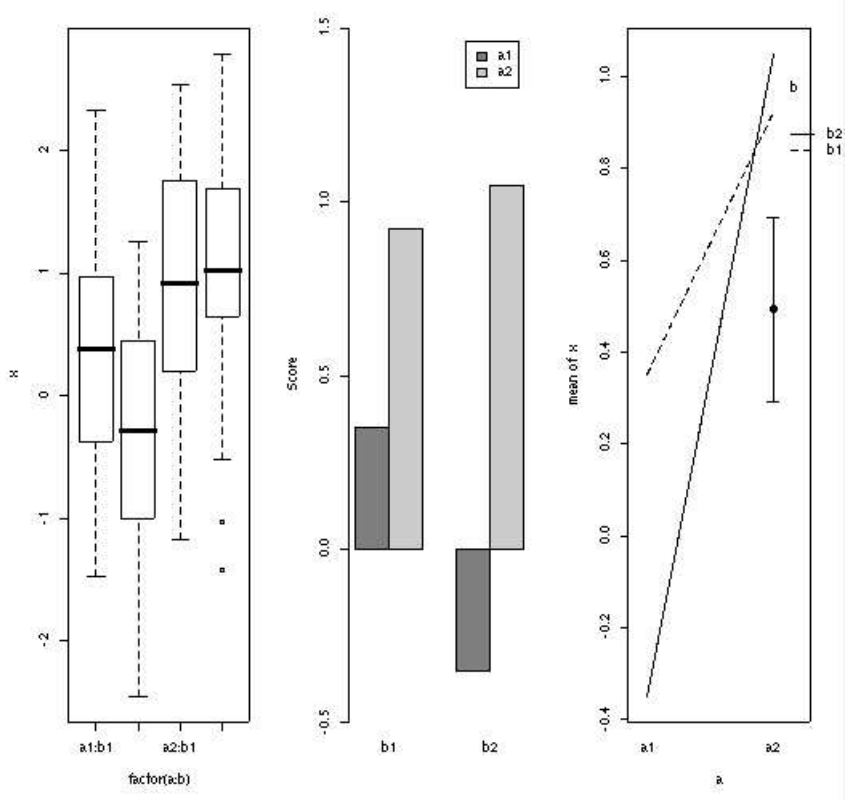

FIG. 16: EX. 4.5.2.

D'autres analyses sont évidemment possibles :

– il faudrait par exemple vérifier quelles sont les paires de moyennes qui diffèrent signifcativement si l'on était en présence de facteurs à plus de deux niveaux ;

– on pourrait utiliser la commande `summary.lm()` pour vérifier quels sont les coefficients du modèle linéaire qui sont statistiquement significatifs, et ne conserver que ceux-là dans un nouveau modèle d'ANOVA, ou on pourrait réduire le nombre de niveaux des facteurs en fonction des différences de moyenne significatives ; le nouveau modèle postulé peut être comparé au précédent à l'aide de la commande `anova(model1,model2)`, etc.

4.5.3 ANOVA factorielle avec mesures répétées

Dans le cas des protocoles avec mesures répétées (un même sujet est étudié dans différentes conditions expérimentales), les choses se compliquent sensiblement en raison de la structure de covariance des données. En fait, on doit prendre en compte dans l'erreur résiduelle du fait qu'il y a des répétitions sur le même individu, et donc une certaine structure de covariance. En plus de l'homogénéité des variances, on doit supposer l'homogénéité des covariances : c'est ce qu'on appelle la contrainte de sphéricité. Dans le modèle classique d'ANOVA, avec la fonction `aov()`, on suppose implicitement une matrice de covariance homogène. Si ces conditions ne sont pas vérifiées, on peut utiliser des facteurs de correction dans les tests, typiquement la correction de Greenhouse-Geisser[11]. Une autre alternative est d'utiliser le modèle linéaire généralisé et de spécifier la structure de covariance. Par ailleurs, la fonction `aov()` ne fonctionne correctement qu'avec des plans équilibrés. S'il y a des valeurs manquantes dans les données[12], il vaut mieux utiliser la fonction `lme()`. La fonction `lme()` permet de spécifier le type d'erreurs intra (corrélées ou inhomogènes), via l'argument `weight`.

Un facteur intra. Il s'agit d'une extension du cas des échantillons appariés avec mesures répétées : au lieu d'avoir 2 échantillons, on en a autant qu'il y a de niveaux pour le facteur considéré. Suivant le type de protocole expérimental, ces niveaux peuvent être relatifs à des conditions différentes (on parle de pseudo-réplication spatiale) ou à un facteur temporel (on parle de pseudo-réplication temporelle) : ce serait typiquement le cas si on étudiait un effet d'apprentissage sur la répétition de la même condition à différentes périodes de temps. Évidemment, on peut traiter ce dernier cas avec une régression, ou des modèles de séries chronologiques, mais un modèle d'ANOVA ou l'utilisation du modèle linéaire généralisé (avec spécification de la structure de covariance, e.g. processus stationnaire d'ordre 2 ou processus auto-régressifs d'ordre 1) peuvent également être envisagés.

Voici l'exemple de traitement classique par ANOVA avec un facteur intra :

[11] voir § 6.10.2 de "Notes on the use of R for psychology experiments and questionnaires"
[12] c'est une condition suffisante (mais non nécessaire)

```
> nsuj <- 10
> nmeasures <- 10
> subject <- gl(nsuj,2*nmeasures)
> trial <- gl(nmeasures,2,2*nsuj*nmeasures)
> cond <- gl(5,1,2*nsuj*nmeasures)
> effectsize <- 1
> scores <- rnorm(2*nsuj*nmeasures)+(as.numeric(cond)-1)*effectsize
> data <- data.frame(subject,cond,scores)
> tapply(scores,cond,summary)
> par(mfcol=c(1,3))
> boxplot(scores~cond)
> plot.design(data)
> interaction.plot(cond,subject,scores,col=c(1:nsuj))
> summary(aov(scores~cond+Error(subject/cond)))
```

Deux facteurs intra. Ici, on suppose qu'il n'y a qu'une seule observation par sujet et par croisement de niveaux. Il y a plusieurs modèles possibles pour l'étude de ce type de protocole. En voici un (Modèle type I, effets fixes) :

```
> subject <- gl(10,4,40)
> cond1 <- gl(2,1,40)
> cond2 <- gl(2,2,40)
> table(cond1,cond2)
> x <- rnorm(40)+1.5*rbinom(40,1,0.5)*rnorm(40)
> plot(x~factor(cond1:cond2))
> interaction.plot(cond1,cond2,x)
> interaction.plot(cond1,subject,x)
> interaction.plot(cond2,subject,x)
> summary(aov(x~cond1*cond2+Error(subject/(cond1*cond2))))
```

et en voici un autre (Modèle type III), en utilisant la procédure `lme()` pour les modèles mixtes :

```
> summary(lme(x~cond1*cond2,random=~1|subject))
```

4.5.4 ANOVA hiérarchique

L'exemple suivant est tiré de [6]. Lorsqu'on est en présence de facteurs emboîtés les uns dans les autres, le modèle d'ANOVA correspondant est dit hiérarchique. Par exemple, on souhaite comparer deux méthodes d'enseignement (facteur A). Pour cela, on forme 6 groupes de 4 élèves, dont 3 reçoivent un enseignement suivant la méthode a1, et 3 suivant la méthode a2. Les enseignements sont dispensés par 6 professeurs différents (facteur B). On dit dans ce cas que l'effet de A a pour source adjointe B, et l'effet de B a pour source adjointe S.

La fonction `aov()` ne permet pas de spécifier plusieurs sources adjointes, on va donc l'appeller deux fois avec chacun des termes pertinents pour l'analyse :

```
> x <- scan('')
1: 9 7 10 8 10 14 8 12 8 3 3 7
13: 12 14 10 14 11 13 9 9 14 10 9 14
25:
> a <- gl(2,12,24)
> b <- gl(6,4)
> summary(aov(x~a+Error(b)))
> summary(aov(x~a/b))
```

Le premier résultat porte sur l'effet de A, tandis que le second indique l'effet de B emboïté dans A (`a:b`).

4.6 Mesures d'association et de dépendance

4.6.1 Analyse de tableaux de contingence

Un tableau croisant pour deux modalités les effectifs observés peut être analysé à l'aide d'un test d'écart à l'indépendance tel que le test du χ^2. Par exemple,

```
> a <- scan('')
1: 38 11
4: 14 51
7:
Read 6 items
> chisq.test(matrix(a,2,2,byrow=T))
```

Par défaut, la correction de continuité de Yates est appliquée. Pour avoir le résultat sans correction, il faut passer l'option `correct=F`.

Lorsque une ou plusieurs des fréquences théoriques attendues sont inférieures à 5, il faut employer le test exact de Fisher, que l'on obtient avec la fonction `fisher.test()`, qui s'utilise comme la procédure `chisq.test()`.

Dans le cas de grands tableaux de contingence, on peut utiliser le modèle linéaire généralisé, ou un modèle log-linéaire. L'exemple suivant est tiré de [5] (pp. 69-70) :

```
> Fr <- c(51,22,70,12,52,33,20,11,63,30,115,43,19,28,47,32,
      70,33,47,25,46,61,32,38)
> y <- expand.grid(A=c("oui","non"),B=c("oui","non"),C=c("M","F"),
       D=c("<2",">=2,<=5",">5"))
> donnees <- data.frame(y,Fr)
> attach(donnees)
> tableau <- table(A,B,C,D)
> tableau[cbind(A,B,C,D)] <- Fr
> resultat <- loglin(tableau,margin=list(c(1,2),c(1,3),c(1,4),
                c(2,3),c(2,4),c(3,4)),par=T)
> deviance <- resultat$lrt
> pval <- 1-pchisq(deviance,resultat$df)
> startmod <- glm(Fr~A+B+C+D,family="poisson",data=donnees)
```

```
> optmod <- step(startmod,scope=list(upper=~.^3,lower=~.),trace=F)
> optmod$anova
> optmodel <- glm(Fr~.^2+B:C:D,family="poisson",data=donnees,x=T)
> summary(optmodel)$coef
> Frfit <- optmodel$fitted.values
> respearson <- (Fr-Frfit)/sqrt(Frfit)
> d <- diag(Frfit)
> x <- optmodel$x
> lev <- diag(x%*%solve(t(x)%*%d%*%x)%*%t(x)%*%d)
> resadj <- (Fr-Frfit)/sqrt(Frfit*(1-lev))
```

Les fonctions `mosaicplot()` et `assocplot` sont utiles pour visualiser les effectifs ventilés sur les différentes modalités des variables catégorielles (Fig. 17).

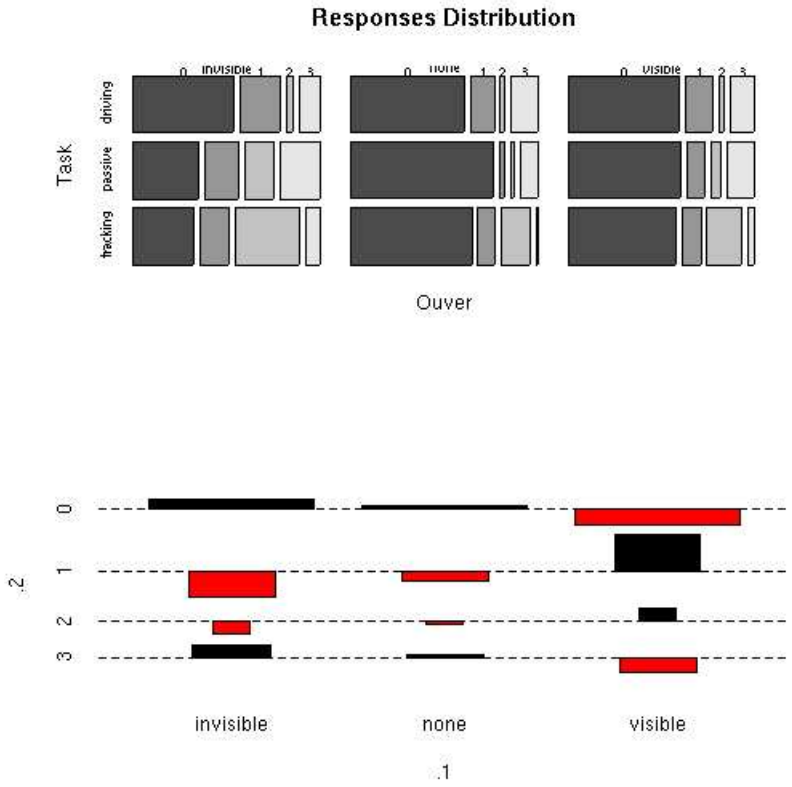

FIG. 17: Ex. 4.6.1.

4.6.2 Régression linéaire

Comme nous l'avons vu dans le cas des distributions conjointes (§ 2.3), la démarche générale pour effectuer de la régression linéaire est la suivante :

```
> x <- rnorm(100)
> y <- 2*x+1.5*rnorm(100)
```

```
> plot(y~x)
> corrcoeff <- cor(x,y)
> r <- lm(y~x)
> plot(r)
> residus <- residuals(r)
> anova(r)
> pred.df <- data.frame(x)
> pp <- predict(r,int="p",newdata=pred.df)
> pc <- predict(r,int="c",newdata=pred.df)
> layout(matrix(c(1,2,3,3),2,2))
> plot(y~x,main=paste("y~x, r = ", round(corrcoeff,3)))
> qqnorm(residus)
> qqline(residus,col='red',new=F)
> plot(y~x,ylim=range(y,pp))
> pred.x<-pred.df$x
> matlines(pred.x,pc,lty=c(1,2,2),col="red")
> matlines(pred.x,pp,lty=c(1,3,3),col="blue")
> abline(r,col='red')
```

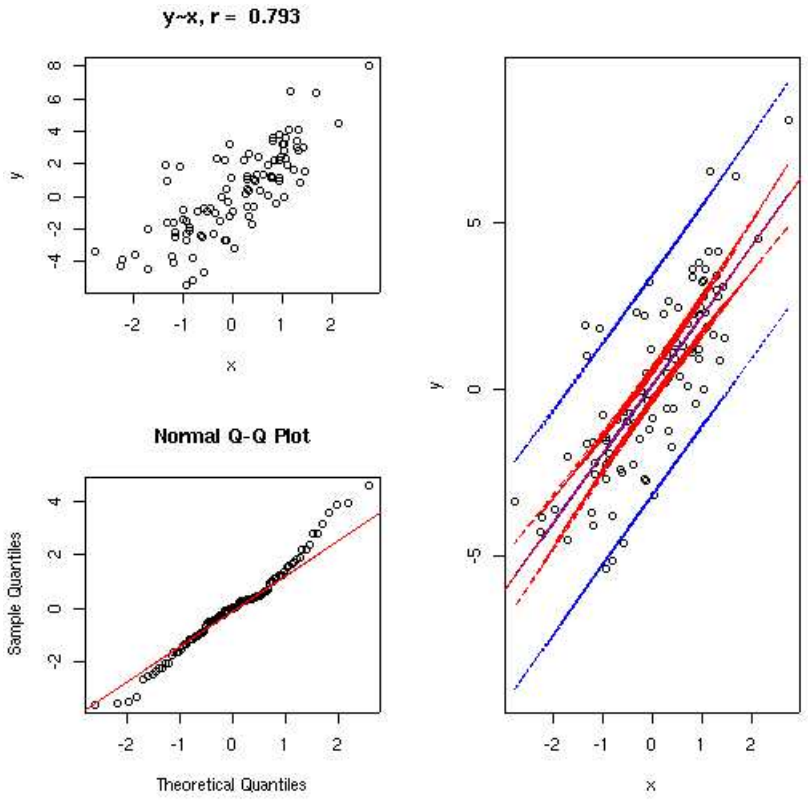

FIG. 18: Ex. 4.6.2 (1).

Avec plusieurs variables explicatives continues, on peut également faire le même type de procédure :

```
> a <- rnorm(100)
> b <- 2*a+rnorm(100)
> c <- 5*a+rnorm(100)
> pairs(cbind(a,b,c))
> summary(lm(c~a*b))
```

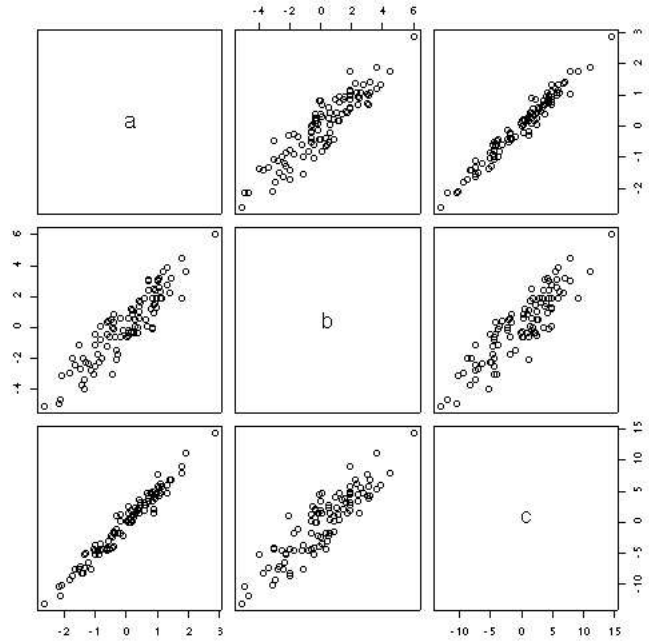

FIG. 19: Ex. 4.6.2 (2).

Références

1. Dalgaard, P. (2002). *Introductory Statistics with R*. Springer-Verlag.
2. Crawley, M.J. (2005). *Statistics – An Introduction using R*. Wiley.
3. Venables, W.N. & Ripley, B.D. (2002). *Modern Applied Statistics with S*. Springer-Verlag.
4. Howell, D.C. (1998). *Méthodes Statistiques en Sciences Humaines*. De Boeck Université.
5. Droesbeke, J.-J., Lejeune, M. & Saporta, G. (2005). *Modèles Statistiques pour Données Qualitatives*. Technip.
6. http ://www.pallier.org/

Table des matières

1 R : un environnement et un langage pour les statistiques 1
- 1.1 Présentation de R . . . 1
- 1.2 L'environnement R . . . 2
- 1.3 Types de base . . . 5
  - 1.3.1 Les vecteurs . . . 5
  - 1.3.2 Les matrices . . . 8
  - 1.3.3 Les facteurs . . . 9
  - 1.3.4 Les listes d'objets . . . 10
- 1.4 Structures de contrôle et programmation . . . 12
- 1.5 Création et importation de données . . . 13

2 Distributions et lois de probabilités 14
- 2.1 Lois de probabilités . . . 14
  - 2.1.1 Loi uniforme . . . 15
  - 2.1.2 Loi binomiale . . . 15
  - 2.1.3 Loi normale . . . 16
  - 2.1.4 Loi du χ^2 . . . 16
  - 2.1.5 Loi de Student . . . 17
  - 2.1.6 Remarques . . . 18
- 2.2 Positionnement d'une statistique . . . 20
- 2.3 Distributions conjointes . . . 21
- 2.4 Estimation de densité . . . 22
- 2.5 Résumés numérique et graphique . . . 22

3 Estimation 24
- 3.1 Estimation ponctuelle . . . 24
  - 3.1.1 Choix d'un estimateur . . . 25
  - 3.1.2 Grands échantillons . . . 27
  - 3.1.3 Petits échantillons . . . 28
- 3.2 Estimation par intervalle de confiance . . . 29

4 Théorie des tests 29
- 4.1 Comparaison de deux distributions . . . 30
- 4.2 Comparaison de variances . . . 30
- 4.3 Comparaison de deux proportions . . . 31
- 4.4 Test t de Student . . . 31
  - 4.4.1 Deux échantillons indépendants . . . 31
  - 4.4.2 Deux échantillons indépendants avec mesures répétées 33
  - 4.4.3 Deux échantillons appariés . . . 34
  - 4.4.4 Deux échantillons appariés avec mesures répétées . . . 35
- 4.5 Généralisation à k groupes : l'Analyse de Variance . . . 36
  - 4.5.1 ANOVA sur un facteur . . . 36

4.5.2 ANOVA sur deux facteurs inter 38
4.5.3 ANOVA factorielle avec mesures répétées 40
4.5.4 ANOVA hiérarchique 41
4.6 Mesures d'association et de dépendance 42
4.6.1 Analyse de tableaux de contingence 42
4.6.2 Régression linéaire 43

Table des figures

1 R sur une console Linux . . . 3
2 L'interface RCommander. . . . 4
3 Simulation de lois normales. . . . 17
4 Loi du χ^2 pour différents ddl. . . . 18
5 Loi de Student T(100) et loi normale (en rouge). . . . 19
6 Loi double-gaussienne $\mathcal{N}(0; 1, 10)$. . . . 20
7 Exemples de représentations graphiques sous R. . . . 25
8 Application de la loi (faible) des grands nombres. . . . 28
9 caption . . . 29
10 Ex. 4.1 . . . 31
11 Ex. 4.4.1 : `boxplot()` avec l'option `notch=T`. . . . 32
12 Ex. 4.4.2. . . . 33
13 Ex. 4.4.3. . . . 35
14 Ex. 4.4.4. . . . 36
15 Ex. 4.5.1. . . . 37
16 Ex. 4.5.2. . . . 39
17 Ex. 4.6.1. . . . 43
18 Ex. 4.6.2 (1). . . . 44
19 Ex. 4.6.2 (2). . . . 45