Premier contact avec le logiciel R

On lance Rstudio en tapant `rstudio`. En dehors d'une présentation plus intégrée que R, l'avantage de Rstudio est l'accès direct à l'aide et une sauvegarde plus simple des figures.

Le site source de base est `http://lib.stat.cmu.edu/R/CRAN` où l'on trouve également une doc papier en pdf ("Introduction to R"), qui n'a malheureusement pas d'index. **Ne pas hésiter à utiliser la doc html (cliquer sur "packages") et le "search engine"**. Les exemples qui s'y trouvent sont très pédagogiques. L'aide en ligne se fait avec le ? (ex : `?anova`).

Le lecteur est invité à essayer les commandes proposées et à en essayer d'autres de son cru.

Le type d'un objet `x` s'obtient en tapant `typeof(x)`. On trouve en particulier les types suivants :

- DOUBLE, CHARACTER, LOGICAL (TRUE, FALSE, NA=not available (donnée manquante))
  L'assignation se fait par `<-` ; on peut utiliser également le signe `=`. Les vecteurs ont le type commun à chaque élément. Construction (pour analyser le resultat : retaper le nom du vecteur seul p.ex. `x`; l'assignation se fait par `<-` mais on peut aussi utiliser le signe `=`) :

```
x<-(1:6);
x<-seq(3,20,4);
x=c(1,2,3.14);
y=c(x,2.7,x);
print(y);
cat("y=",y,"\n");    # \n est le saut de ligne
y[2]=NA;
y[3]=NA;
l=is.na(y);
y[l]=9999;
y[y<2]=2;
```

  On peut aussi lire un vecteur dans un fichier `y=scan('bidon.dat')`.
  `paste` permet la concaténation de chaînes de caractères :

```
cc=paste(c('X','Y'),1:7,sep='')
```

  On précise l'absence de séparateur. Noter ici aussi le remplissage périodique par défaut. On peut donner des noms aux coefficients

```
x=1:7/3;names(x)<-cc; x['X3']
```

- LIST : `ll=list('3',c(2,3),'tr'); ll[2];` ou mieux `ll=list(num='3',vec=c(2,3),ch='tr'); ll$ch;`. Les objets évolués que l'on verra plus bas ont tous ce type.
- LANGUAGE : type des formules définissant les modèles de régression
- CLOSURE : type des fonctions : `typeof(print)`.

Les classes. Ce sont les structures d'objet propres à R, p. ex. la classe `matrix`. On obtient la classe de `x` avec la commande `data.class(x)`.

Un objet d'une classe possède les **attributs** propres à cette classe, par exemple la dimension `dim` pour une matrice. On les visualise en tapant `attributes(x)`. On obtient sa valeur avec les parenthèses : `dim(mat)`. Les fonctions comme `plot` ont un effet différent selon la classe d'un objet.

- NUMERIC, CHARACTER, LIST : Classes attribuées aux objets élémentaires (réels, vecteurs,...)
- MATRIX : Les matrices se fabriquent à partir d'un vecteur qui énumère les colonnes : on met `x` dans une matrice 5×2 avec `m=matrix(x,5,2)` (noter la répétition si `x` est trop court). La 3-ième ligne de `m` est `m[3,]` (voir l'article " `[` " du help). Le produit se fait par `%*%`, la transposée `t(m)` et l'inverse `solve(m)`. La commande `cbind()` (resp. `rbind()`) permet de concaténer des matrices en colonne (resp en ligne) :

```
m=cbind(c(1:3),c(6:8));
data.class(m);
attributes(m);
dim(m);
m[2,]=3.14;
m[m>3]=4.14;
```

```
m%*%c(1,2);
```

Contrairement à SCILAB ou MALTAB, les vecteurs ne sont pas des matrices, ligne ou colonne. Les vecteurs sont convertis en matrice colonne dans certaines opérations ; ainsi, dans la transposition `y=t(x)` , `y` est une matrice ligne (voir `attributes(y)`) ; `c(y)` est à son tour un vecteur.

- DATA.FRAME : un tableau individus/variables contenant les valeurs (de tout type), et possiblement des noms de variables et des noms d'individus. «NA» indique une donnéée manquante. Certaines données-test sont disponibles avec la commande `data()`, p.ex. observer les sorties des commandes suivantes :

```
data(mtcars);
mtcars;
View(mtcars);
attributes(mtcars);
names(mtcars);
row.names(mtcars);
data.class(mtcars);
```

Noter que `names()` contient les noms des variables. On peut changer les attributs sans peine :

```
names(mtcars)=(1:11)
```

Le `$` permet l'accès aux variables : `mtcars$mpg`. On peut extraire et modifier

```
data(ToothGrowth)
str(ToothGrowth)
t1=ToothGrowth[ToothGrowth$len>27,]
levels(t1$supp)=c('orange','acide')
t2=t1[t1$supp=='orange',]
```

ToothGrowth est un tableau de 60 individus avec deux variables numériques «len» et «dose» et une symbolique «supp» prenant les valeurs VC ou OJ. On extrait les individu pour lesquels $len > 23$ et dans ce nouveau dataframe on change les noms des modalités de *supp* et l'on extrait un nouveau sous-tableau. Noter que `t1$supp=='orange'` est un vecteur de True/False.

Un tableau de données se lit dans un fichier par la commande

```
voit=read.table('voiture.dat')
```

La présence d'une chaîne de caractères en moins sur la première ligne indique que cette ligne constitue des noms de variables et la première colonne des noms d'individus. S'il n'y a pas de nom d'individu mais des noms de variables, faire `read.table('voiture.dat',header=TRUE)`. Voici un exemple de `data.frame` créée à la main, avec deux variables var1 et var2 :

```
datu=data.frame(var1=c(1:11));
row.names(datu)=paste(c('a','b'),1:11,sep='*');
datu$var2=c(9:19);
str(datu)
```

Pour extraire un sous-ensemble d'individus ou de variables, utiliser la commande `subset` (on peut aussi utiliser `[]` comme plus haut).

Pour concaténer, utiliser `rbind` et `cbind`.

- FACTOR. Permet de transformer un vecteur de chaînes de caractères en un facteur, utilisé comme tel dans d'autres routines (regression, plot...)

```
eng=rep(c('engrais1','engrais2','engrais3'),6);
eng=factor(eng);
attributes(eng);
```

→ Effet sur le `plot` : comparer les deux commandes

```
plot(mtcars$gear,mtcars$mpg);
plot(factor(mtcars$gear),mtcars$mpg);
     ou    plot(mpg~factor(gear),data=mtcars);
```

- Conversions : utiliser `as.xxx` (cf la doc en ligne). Par exemple pour faire une data.frame à partir d'une matrice :

```
x=matrix(1:6,2,3);
xd=as.data.frame(x);
names(xd)=c('A','B','C');
xd$C;
```

Soit `y=c(2:7)`. Comparer la dimension requise pour `a` dans les deux commandes `names(y)=a` et `names(data.frame(y))=a`, et le résultat de `y[a[1]]` dans les deux cas.

- LM : linear model. Objet retourné par les fonctions d'estimation de modèle linéaire. Il contient plein d'informations concernant le modèle.

```
md= lm(mpg~cyl+disp,mtcars);
n=dim(mtcars)[1];
plot(md);
attributes(md);
md$residuals;
summary(md);
p=length(md$coefficients);
```

On pourra utiliser ces commandes pour recalculer le $\hat{\sigma}$ et vérifier qu'il correspond bien à la valeur annoncée par `summary` (affichage des coefficients et leur variance d'estimation). `anova` permet de faire analyse de variance et tests.

```
anova(md)
mdi= lm(mpg~cyl*disp,mtcars); anova(md,mdi)
```

- ...et il en existe bien d'autres

Syntaxe. Le `for` et le `if` s'utilisent comme suit, avec des accolades si plusieurs instructions sont demandées :

```
x= c(1,0,0,3,4,0)
for (i in 1:length(x)) if (x[i]==0)  x[i]=99 else x[i]=x[i]+1
for (i in 1:6) {x[i]=x[i]+1;print(i)}
```

Fonctions. L'aide en ligne donne à chaque fois un exemple d'utilisation très simple. Dans l'exemple qui suit `attach(mtcars)` évite d'avoir à taper `$mtcars` à chaque fois ; `detach` fait l'inverse (voir aussi le paragraphe «Pièges» à la fin du document).

Observer l'effet des fonctions `mean()`, `substr()`, `tapply()`.

```
data(mtcars);
attach(mtcars);
mean(mpg);
nm=substr(row.names(mtcars),1,1);
m=tapply(mpg,nm,mean);
```

Autres exemples de fonctions : `max(x);length(x),cov(x,y);sort(x);table(x)...`
Exemple de définition de d'une fonction simple :

```
fac = function(n) {x=0; if (n>0) for (i in 1:n ) x=x+i; x};
```

Une fonction peut avoir plusieurs arguments en sortie, leur donner un nom :

```
fac1 = function(n) {x=0;
if (n>0) for (i in 1:n ) x=x+i; y=n^2/2; return(list(su=x,sq2=y))};
```

avec l'appel

```
a=fac1(10)
print(a$su,a$sq2);
```

Fichiers. On a vu la commande `read.table` pour lire un tableau de données (`write.table` permet d'écrire ; `write` est insuffisant). La commande `scan` existe également, plus rustique, avec davantage de potentialités. Les commandes `save` et `load` permettent de stocker et charger des données sous un format binaire propre à R, plus économique. Voir la documentation en ligne...

Courbes. La fonction de base est `plot()`. Cette fonction s'adapte à la classe (voir le paragraphe sur les classes et l'exemple des rubriques FACTOR et LM). Elle effectue un nouveau tracé en effaçant ce qui précède. Pour compléter une figure (c-à-d sans effacer) utiliser `points()` ou `lines()`. Utiliser `text()` pour placer du texte sur la figure (écrire les noms des individus...).

Lorsqu'on superpose des tracés par cette technique les axes ne sont pas réajustés, de sorte que les nouveaux points hors champ disparaissent. Une solution est de faire un premier plot sans tracer (`type="n"`), pour seulement ajuster les échelles et mettre les titres. Si par exemple on veut tracer à la fois la puissance (hp) et le déplacement (disp) fonction de la consommation (mpg) :

```
data(mtcars);attach(mtcars);
par(cex=1.4);
plot(c(mpg,mpg),c(disp,hp),type="n",ylab="disp (o), hp (#)",xlab="mpg")
points(mpg,disp,pch="o");points(mpg,hp,pch="#")
```

Pour tracer plusieurs courbes d'un coup sur une même figure il est généralement plus simple d'utiliser `matplot()`.

La commande `par(cex=1.4)` permet d'augmenter la taille des caractères d'un facteur 1,4.

Pour faire plusieurs tracés sur une même figure on peut utiliser `par(mfrow=c(m,n))` avec p.ex. $m = n = 2$ si l'on veut 4 figures en format 2×2, ou bien la commande `layout()` (voir la documentation).

Pour sauvegarder la figure dans un fichier, sous Rstudio il suffit d'utiliser le menu, et sous R (ou à l'intérieur d'un programme) faire d'abord `postscript("fich.ps")` (ouverture du fichier ps et début des enregistrements), puis `dev.off()` (pour fermer le fichier) après le tracé. Mettre `postscript("fich.eps")` si l'on veut inclure dans Latex.

Librairies. Dans les librairies (packages), on trouvera les fonctions habituelles d'analyse de variance, de regression, les modèles linéaires généralisés, les séries temporelles et l'analyse des données avec des méthodes de clustering.

Ne pas oublier de taper, p.ex., `library(mva)` si vous voulez utiliser les fonctions d'analyse de données.

Exemple : se brancher sur le "Search Engine", taper "clustering" et faire tourner l'exemple de la fonction `hclust`.

L'adresse `www.ci.tuwien.ac.at/R/src/contrib/PACKAGES.html`. contient des contributions sous forme de librairies supplémentaires. Il s'agit d'estimateurs divers (PLS...), méthodes à la mode,... Par exemple `mvnmle` qui permet d'estimer moyenne et variance d'individus vectoriels gaussiens i.i.d. ayant des coordonnées manquantes. Une librairie chargée apparaît automatiquement dans l'aide en ligne.

Programmes. Le # est le caractère de commentaire.

Simulations en parallèle. Soit une fonction qui renvoie une liste. On peut la réaliser de nombreuses fois en parallèle par la commande `mcapply`. Dans l'exemple qui suit, `n` est le nombre de coeurs de la machine (on gagne un facteur `n` au mieux), et `simu` est la fonction, qui réalise deux gaussiennes.

```
simu=function(i){a=rnorm(2);return(list(l1=a[1],l2=a[2]))}
require(parallel)
nsimu=5
n=detectCores(all.tests = FALSE, logical = TRUE)
a=mclapply(1:nsimu,simu,mc.cores=n)
resu=unlist(a)
```

```
  nn=length(resu)/nsimu
  resu=matrix(resu,nn,nsimu,dimnames=list(names(resu)[1:nn]))
  print(resu)
```

Les dernières lignes sont une remise en forme du résultat. On obtient

```
        [,1]       [,2]       [,3]       [,4]       [,5]
l1 -1.4886671  0.9528496 0.06447107 -0.2356595 -1.7460159
l2 -0.8295282 -1.1801163 0.59778540 -0.1103650 -0.7454331
```

Pièges classiques

►►► Après `attach` ne JAMAIS modifier les variables «attachées» (voir le help). En cas de confusion, on peut détruire toutes les variables (repartir à zéro) avec `rm(list=ls())`.

►►► `1+k:100` est différent de `(1+k):100`

►►► Pour l'optimisation, on peut utiliser `optim` si l'on fournit le gradient de la fonction, sinon, il est plus sage d'utiliser `ucminf` de `library(ucminf)`, ou `nedlermead`.