Université Paul Sabatier

2012-2013
X. Gendre

Aide mémoire

Ce document a été rédigé à partir de l'excellent travail d'Aymeric Duclert que l'on peut retrouver à l'adresse `http://www.duclert.org`. On pourra s'y reporter pour plus d'informations et d'exemples

1 Le langage R

1.1 Commandes de base

- ▷ `R` : pour démarrer R en interactif.
- ▷ `R --vanilla` : pour démarrer R en interactif sans charger ni sauver de session.
- ▷ `q()` : pour quitter.
- ▷ `help(solve)` ou `?solve` : avoir de l'aide sur la commande `solve`.
- ▷ `help.search("chi")` ou `??chi` : pour chercher dans l'aide les mots contenant "chi".

1.2 Syntaxe de base

Les commentaires commencent par le caractère '`#`'.
Les différents lignes de code peuvent être séparées par des retours chariots et/ou des caractères '`;`'.
Il est possible de grouper des instructions grâce aux accolades '`{`' et '`}`'.
Les conditions sont testées avec les mots clefs `if` et `else`. Seule une ligne est alors prise en considération. Si il y a plusieurs instructions, il faut utiliser les accolades.
Pour faire des boucles, on peut utiliser `for` et `while`. Même remarque que pour `if` si plusieurs instructions.

```
if (x > y) print("x est grand") # Affiche un message si x > y

if (r-s > 0)
{
  print("r est grand");print("s est petit") # Affiche deux messages
}
else
  print("r est petit")

# Affiche les nombres de 1 à 5
for (i in 1:5) print(i)
```

```
# Idem avec while
i <- 1
while (i < 6)
{
  print(i)
  i <- i+1
}

q() # Quitte R
```

Pour définir une fonction, on utilise `function()` (voir le TP2). Dans R, les fonctions sont des objets comme les autres et peuvent donc être manipulées de la même façon.
Si on souhaite que la fonction retourne une valeur, il faut employer l'instruction `return()`.
Il est possible de passer des paramètres à une fonction en précisant la liste de ceux-ci dans les parenthèses de `function()`. Il est aussi possible de donner une valeur par défaut à un paramètre. Dans ce dernier cas, le paramètre n'est plus obligatoire lors de l'appel de la fonction.

```
f1 <- function()
{
  for (i in 1:5) print(i)
}

f2 <- function(n, m)
{
  for (i in n:m) print(i)
}

f3 <- function(n=10)
{
  for (i in 1:n) print(i)
}

f1() # Affiche les nombres de 1 à 5
f2(4, 9) # Affiche les nombres de 4 à 9
f3() # Affiche les nombres de 1 à 10
f3(7) # Affiche les nombres de 1 à 7
```

1.3 Manipulation des objets

- ▷ `x <- 5` : l'objet `x` est affecté de la valeur `5`.
- ▷ `print(x)` : affiche le contenu de l'objet `x`.
- ▷ `print("x")` : affiche la chaîne de caractères `"x"`.
- ▷ `vt <- NULL` : affecte la valeur nulle (pas d'objet) à `vt`.
- ▷ `ls()` : pour avoir la liste des objets.
- ▷ `rm(x, vt)` : pour supprimer les objets `x` et `vt`.
- ▷ `rm(list=ls())` : pour supprimer tous les objets.

1.4 Vecteurs

Il n'y a pas de nombre isolé sous R, il n'y a que des vecteurs (un nombre isolé est un vecteur de taille 1). Un vecteur est une liste d'éléments simples (numériques, booléens, chaînes de caractères) tous du même type. Les indices des vecteurs commencent à 1 (comme toujours en R).

Avec R, il est important d'avoir en tête le principe de la complétion cyclique : le vecteur le plus petit voit ses éléments recyclés autant de fois que nécessaire :

```
x <- c(2, 3); y <- c(4, 5, 6, 7); x + y
[1]  6  8  8 10
```

▷ `x + y` : addition terme à terme des vecteurs `x` et `y`.
▷ `sum(x)` : somme de tous les éléments du vecteur `x`.
▷ `x * y` : multiplication terme à terme des vecteurs `x` et `y`.
▷ `prod(x)` : produit de tous les éléments du vecteur `x`.
▷ `length(vect)` : nombre d'éléments du vecteur `vect`.
▷ `sum(vect)` : somme des éléments du vecteur `vect`.
▷ `mean(vect)` : moyenne uniforme des éléments du vecteur `vect`.
▷ `is.numeric(vect)` : teste si `vect` est numérique.
▷ `is.character(vect)` : teste si `vect` est de type chaîne de caractères.
▷ `as.numeric(vect)` : convertit un vecteur `vect` de type chaîne de caractères en numérique.
▷ `as.character(vect)` : convertit un vecteur `vect` de type numérique en chaîne de caractères.
▷ `1/vect` : vecteur dont chaque coordonnée est l'inverse de celle de `vect`.
▷ `c(x, y)` : concaténation des 2 vecteurs x et y.
▷ `x[c(1,5)]` : donne un vecteur de taille 2 avec les valeurs d'indices 1 et 5 de `x`.
▷ `x[1:5]` : donne un vecteur avec les 5 premières valeurs de `x`.
▷ `x[5]` : donne la 5ème valeur de `x`.
▷ `x[c(1,2,2,1)]` : donne un vecteur à 4 valeurs (la 1ère, la 2ème, la 1ère et la 2ème valeur de `x`).
▷ `x[-(1:5)]` : donne un vecteur avec toutes les valeurs de `x` sauf les 5 premières.
▷ `x[c(TRUE, FALSE, TRUE)]` : si `x` est un vecteur à 3 éléments, renvoie un vecteur avec le premier et le troisième élément. Si `x` a plus de valeurs que le vecteur booléen indice, les valeurs booléennes sont recyclées selon le principe de la complétion cyclique.
▷ `x[!is.na(x)]` : renvoie un vecteur avec toutes les valeurs déterminées.
▷ `x[x > 5]` : renvoie un vecteur avec toutes les valeurs de `x` supérieures 5.
▷ `x[x > 5] <- 0` : met à 0 toutes les valeurs de `x` supérieures à 5.
▷ `summary(vect)` : pour un vecteur numérique, indique des statistiques simples sur les valeurs (moyenne, médiane, min, max, 1er et 3ème quartiles).
▷ `3 %in% x` : revoie `TRUE` si `3` est dans `x`.

1.5 Séquences et répétitions

▷ `1:5` : donne le vecteur (1,2,3,4,5).
▷ `5:1` : donne le vecteur (5,4,3,2,1).
▷ `seq(5)` : donne le vecteur (1,2,3,4,5).
▷ `seq(1, 5)` : donne le vecteur (1,2,3,4,5).
▷ `seq(1, 2, 0.5)` : donne le vecteur (1.0,1.5,2.0).
▷ `rep(x, 3)` : renvoie un vecteur de taille `3*length(x)` avec 3 fois `x` mis bout à bout.
▷ `rep(x, times=3)` : idem que `rep(x, 3)`.
▷ `rep(x, each=3)` : renvoie un vecteur de taille `3*length(x)` avec 3 fois la première coordonnée de `x`, 3 fois la seconde, ...

1.6 Matrices

L'indexation des éléments d'une matrice est similaire à celui des vecteurs en donnant l'indice de la ligne, puis celui de la colonne. L'accès aux éléments de la matrice suit la même règle.
▷ `mat[1,2]` : cellule de la première ligne, deuxième colonne de la matrice `mat`.

▷ `mat[,1]` : renvoie le premier vecteur colonne de `mat`.
▷ `mat[3,]` : renvoie le troisième vecteur ligne de `mat`.
▷ `matrix(1:6, nrow=2)` : renvoie une matrice à 2 lignes avec les nombres de 1 à 6.
▷ `matrix(1:6, ncol=2, byrow=TRUE)` : renvoie une matrice à 2 colonnes avec les nombres de 1 à 6 en les disposant ligne par ligne.
▷ `matrix(c(3,2,1,5,6,4), nrow=3, dimnames=list(c("A","B","C"), c("a","b")))` : donne des noms aux lignes et aux colonnes.
▷ `dim(mat)` : renvoie les dimensions de la matrice `mat`.
▷ `t(mat)` : la transposée de `mat`.
▷ `diag(mat)` : renvoie un vecteur correspondant à la diagonale de la matrice `mat`.
▷ `diag(vect)` : renvoie une matrice diagonale dont les éléments sont ceux du vecteur `vect`.
▷ `diag(k)` : renvoie une matrice identité carrée de dimension `k`.
▷ `diag(k, n)` : renvoie une matrice diagonale carrée de dimension `n` et dont les éléments diagonaux sont égaux à `k`.
▷ `diag(mat) <- vect` : remplace la diagonale de `mat` par les éléments du vecteur `vect`.
▷ `mat1 + mat2` : addition de matrices.
▷ `mat1 * mat2` : produit de matrices terme à terme.
▷ `mat1 %*% mat2` : produit de matrices au sens mathématique.
▷ `rowSums(mat)` et `colSums(mat)` : renvoient les vecteurs contenant la somme des lignes ou la somme des colonnes.
▷ `rowMeans(mat)` et `colMeans(mat)` : renvoient les vecteurs contenant la moyenne des lignes ou la moyenne des colonnes (moyenne uniforme).
▷ `apply(mat, 1, sum)` : renvoie un vecteur avec la somme des lignes.
▷ `apply(mat, 2, sum)` : renvoie un vecteur avec la somme des colonnes.
▷ `apply(mat, 1, mean, trim = 0.1)` : les arguments après la fonction sont les arguments supplémentaires à passer à celle-ci.
▷ `scale(mat)` : centre et réduit les colonnes de `mat`.
▷ `solve(mat)` : renvoie l'inverse de `mat`.
▷ `eigen(mat)` : revoie une liste avec les valeurs propres et les vecteurs propres d'une matrice carrée.

1.7 Listes

▷ `list(c("a", "b"), 5, c(3, 2))` : création d'une liste.
▷ `list(firstname="jean", lastname="dupond", age=35)` : création d'une liste avec des noms.
▷ `length(lis)` : longueur de la liste `lis`.
▷ `lis[[1]]` : accès au premier élément de la liste.
▷ `lis$firstname` ou `lis[["firstname"]]` : accès à un élément par son nom.

1.8 Data Frames et Facteurs

Un facteur est un type de vecteur qui code pour une propriété qualitative (attribut nominal) et qui est codé en interne par un numéro et non par la chaîne de caractères représentant sa valeur.
▷ `factor(c("rouge", "vert", "rouge", "bleu", "vert"))` : exemple de facteur.
▷ `levels(fac)` : modalités du facteur `fac`.
▷ `nlevels(fac)` : nombre de modalités de `fac`.
Une data frame (jeu de données) est une sorte de matrice où chaque colonne peut avoir son type (contrairement à une matrice dont tous les éléments doivent avoir le même type). On accède aux éléments de la data frame comme pour les matrices mais la façon de faire pour les listes fonctionne aussi. Les colonnes représentent les variables et les lignes les individus.
▷ `data.frame(age=c(15,20), nom=c("pierre","jean"))` : exemple de data frame.

▷ `dtf$nom` : accède à la colonne (variable) `nom` de la data frame `dtf`.
▷ `dtf[2,"age"]` : accède au deuxième élément de la colonne `age`.
▷ `dtf[,"age"]` : récupération de la colonne `age`.
▷ `summary(dtf)` : fait un résumé sur chaque variable (chaque colonne de la data frame).

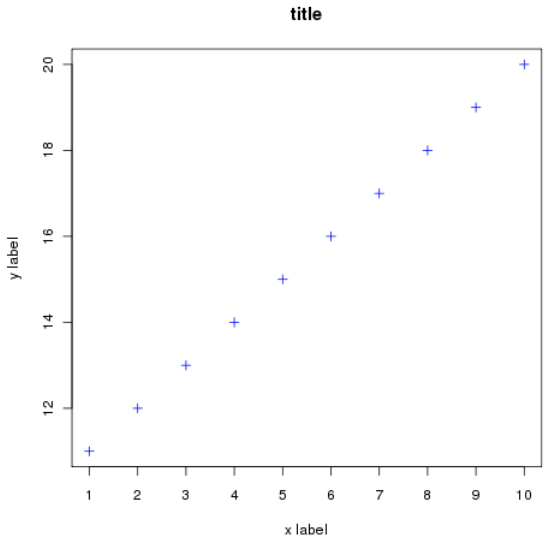
2 Graphiques

2.1 Graphes simples

▷ `plot(y)` : nuage des valeurs d'un vecteur contre leurs indices.
▷ `plot(x,y)` : nuage des valeurs du vecteur `y` contre le vecteur `x`.
▷ `plot(mat)` : trace la deuxième colonne de la matrice `mat` en fonction de la première.
▷ `plot(f)` : trace la fonction `f`.
▷ `x11()` : ouvre une fenêtre X11 pour tracer un graphe.
▷ `pdf("sortie.pdf")` : crée un fichier PDF pour tracer un graphe.
▷ `png("outfile.png", 800, 500)` : crée une image PNG de taille 800 par 500 pixels pour tracer un graphe.
▷ `jpeg("outfile.jpg", 800, 500)` : crée une image JPG de taille 800 par 500 pixels pour tracer un graphe.
▷ `dev.off()` : referme le device courant.
▷ `graphics.off()` : referme tous les devices graphiques.
▷ `dev.list()` : la liste des devices ouverts actuellement.
▷ `dev.cur()` : le numéro du device courant.
▷ `dev.set(2)` : positionne le device courant au device numéro 2.
▷ `dev.off(2)` : ferme le device numéro 2.

2.2 Paramètres des graphes

R offre de nombreuses manières de tracer et de présenter les graphiques. A cette fin, il faut utiliser les paramètres graphiques qui régissent la façon dont le graphe est tracé dans le device courant. Nous donnons ici les options les plus classiques, pour plus d'information, voir `help(par)`.




```
plot(seq(1, 10), seq(11, 20),
     main="title",
     xlab="x label", ylab="y label",
     type="p", col="blue", pch=3, lab=c(10, 5, 0))
```

▷ `type` : `p` pour points, `l` pour lignes, `b` pour points et lignes et `o` pour points et lignes se

chevauchants.

▷ `col` : couleur du tracé (voir `colors()`).

▷ `pch` : caractère du tracé (`1` rond, `2` triangle, `'A'` pour A, ...).

▷ `lab` : donne le nombre de graduations sur l'axe des x et sur l'axe des y.

▷ `lwd` : épaisseur du trait.

▷ `lty` : type de trait (`"blank"`, `"solid"`, `"dashed"`, ...).

▷ `fg` : couleur des axes et des ticks (ou de la grille).

▷ `bty` : allure de la boîte des axes (`o`, `l`, `7`, `c`, `u`, `]`, `n`).

▷ `cex` : taille du texte et des symboles (défaut à 1).

▷ `xlim=c(0,5)` : impose d'avoir l'axe des x qui varie entre 0 et 5 (idem avec `ylim`).

2.3 Rajout d'éléments

▷ `abline(a=1, b=2, col="green")` : ajoute une droite verte d'ordonnée à l'origine 1 et de pente 2.

▷ `abline(lm(y ~ x), col="red")` : rajout de la droite de régression linéaire.

▷ `points` : ajoute un nuage de points.

▷ `segments` : ajoute des segments.

▷ `arrows` : ajoute des flèches.

2.4 Boxplot (diagramme à moustaches)

Permet de représenter une distribution de valeurs sous forme simplifiée avec la médiane (trait épais), une boîte s'étendant du quartile 0.25 au quartile 0.75, et des moustaches qui s'étendent par défaut jusqu'à la valeur distante d'au maximum 1.5 fois la distance inter-quartile.

▷ `boxplot(c(3,5,6,7,4,5))` : exemple avec un simple vecteur.

▷ `boxplot(list(a=c(3,5,6,7,4,5), b=c(7,5,6,7,8,4)))` : avec une liste de vecteurs donnant une boîte par vecteur.

Paramètres de `boxplot` :

▷ `range` : règle la longueur des moustaches (si c'est 0, les moustaches vont du min au max).

▷ `varwidth` : si `TRUE`, largeur des boîtes proportionnelle à racine de `n` où `n` est le nombre de valeurs.

▷ `names` : renommage des catégories de chaque boîte.

▷ `border` : couleur de la bordure des boîtes.

▷ `col` : couleur de l'intérieur des boîtes.

▷ `horizontal` : si `TRUE`, les boîtes sont horizontales plutôt que verticales.

2.5 Histogrammes

▷ `hist(vect)` : trace un histogramme de la distribution des valeurs du vecteur.

▷ `hist(vect, freq=FALSE)` : indique en ordonnées les fréquences au lieu du nombre d'individus.

▷ `hist(vect, breaks=100)` : trace l'histogramme en partageant l'intervalle de variation de `vect` en 100 sous-intervalles.

▷ `his <- hist(vect, plot=FALSE)` : calcule l'histogramme mais sans le tracer. Dans ce cas `his$breaks` donne les limites des intervalles et `his$counts` donne le nombre d'individus sur chaque intervalles.

2.6 Q-Q plot

Les commandes `qqplot` et `qqnorm` permettent de comparer graphiquement des distributions pour savoir si elles sont identiques. Si les points sont alignés, les distributions sont similaires, sinon elles ne le sont pas.

- ▷ `qqplot(x, y)` : compare les distributions des points des vecteurs `x` et `y`.
- ▷ `qqnorm(x)` : compare la distribution des points du vecteur `x` avec une loi normale.
- ▷ `qqline(x)` : trace une ligne entre le 1er et le 3ème quartile (pour `qqnorm`).

2.7 Régression linéaire

La fonction `lm` permet de calculer la régression linéaire d'une variable numérique en fonction de variables explicatives.

- ▷ `lm(y ~ x)` : calcule la régression linéaire de $y = ax + b$.
- ▷ `lm(y ~ u + v)` : calcule la régression linéaire de $y = au + bv + c$.