TISD – FICHE 1

Prise en main de R

Adrien Hardy, adrien.hardy@math.univ-lille1.fr

Avant-propos

R est un language et un environnement gratuit permettant (entres autres) de faire de la statistique des données sur toutes les plateformes. Si vous voulez l'installer sur votre ordinateur, il suffit de se rendre sur le site internet du CRAN (Comprehensive R Archive Network). Les commandes sont entrées une à une dans une console puis exécutées. Pour simplifier l'utilisation, on utilisera l'interface R Studio (gratuit, open source, multiplateforme).

Aide

Pour obtenir l'aide détaillée d'une commande on tape `?commande` ou `help(commande)`. Pour des exemple d'utilisation de la commande, on a `example(commande)`. Si on ne connaît pas le nom exact de la commande, on peut tenter `??truc` : l'aide affichera alors toutes les commandes qui contiennent "truc" dans leur descriptif. De façon générale, la syntaxe d'une commande est :

```
commande(variable1, variable2, ... , option1 = a, option2 = b, ... )
```

Attention : Si la liste des variables et des options est vide, on écrit `commande()` sans oublier les parenthèses, sinon R retourne les détails de la commande sans l'exécuter.

Répertoire de travail (working directory)

Pour savoir dans quel répertoire R importe et exporte les données par défaut, on utilise la commande `getwd` ("get working directory"). Si on veut changer ce répertoire en un dossier D, on utilise `setwd("/chemin/D")`.

Créer un répertoire de travail où vous enregistrerez vos scripts R et vos données.

Ecrire des commentaires (non compilés)

Sur une même ligne, tout ce qui suit le symbole # ne sera pas pris en compte par R.

1 Structures des données

1.1 Vecteurs

1. (Concaténation) Saisir le vecteur `a <- c(10,5,3,6,21)`. Taper `a`. Puis `a[2]` et `a[c(1,3)]`.

2. (Suite) Générer le vecteur `b <- seq(from=1, to=10, by=2)` et demander `b`. Si on veut générer la suite des entiers naturels consécutifs de m à n, on peut aussi utiliser la commande `m:n`. Pour vérifier qu'on a bien créé un vecteur, on utilise la commande `is.vector`.

3. (Opérations élémentaires) Taper `3*a+b+1`. Qu'obtient-on ? Même question pour `cos(a)`, `exp(a)`, `sum(a)`, `cumsum(a)`, `mean(a)`.

1.2 Matrices

1. (Matrices diagonales) À l'aide de la commande `diag`, générer une matrice diagonale de dimension 3 dont les éléments diagonaux sont 1, 5 et 9 .

2. (Créer une matrice) Taper `M <- matrix(c(1:6), nrow=3)`. Essayer aussi `matrix(c(1:6), ncol=3)` et `matrix(c(1:6), nrow=3, byrow=TRUE)` et comparer. Créer la matrice

$$N = \begin{pmatrix} 1 & 11 & 0 \\ 5 & 0 & 2 \end{pmatrix} .$$

3. (Produit matriciel) Qu'est-ce que représente `M%*%N` ? Comparer `U*V` et `U%*%V` avec

$$U = \begin{pmatrix} 1 & 0 \\ 0 & 2 \end{pmatrix} , \qquad V = \begin{pmatrix} 1 & 3 \\ 2 & 4 \end{pmatrix} .$$

4. (Systèmes linéaires) On considère l'équation $Ax = b$ où

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 3 & 1 \end{pmatrix} , \qquad b = \begin{pmatrix} 1 \\ 2 \\ 1 \end{pmatrix} .$$

La matrice A est-elle inversible ? Résoudre $Ax = b$ en utilisant la commande `x <- solve(A,b)`. Vérifier le résultat en tapant `A%*%x`. Définir `A_inv <- solve(A)` et vérifier le résultat en tapant `A%*%A_inv` et `A_inv%*%A`.

5. (Décomposition spectrale) Avec la commande `eigen`, calculer les valeurs propres et vecteurs propres de A. Reconstruire A à partir de ses vecteurs et valeurs propres ; utiliser l'aide de la commande !

1.3 Tableaux

1. Saisir le tableau `x <- array(data=c(15,3,12,2,1), dim=c(1,5))`. Comparer au vecteur `y <- c(15,3,12,2,1)` en demandant `nrow(x)`, `ncol(x)`, `dim(x)` et de même pour `y`.

2. Saisir le tableau `x <- array(c(1,0,0,0,1,3,0,2,1), dim=c(3,3))` et comparer x à la matrice A de la question 1.2.4. Conclusion ? Taper `is.matrix(x)`.

3. Saisir le tableau `x <- array(1:8, c(2,2,2))`. Donner une représentation graphique de ce tableau (sur une feuille de papier).

1.4 Listes et data frames

À la différence d'un tableau, une liste peut contenir des objets de différents types, comme "numeric" (`2.1`), "character" (`"Soleil"`), "logical" (`TRUE`), etc

1. Taper

`L <- list(name="Fred", wife="Mary", no.children=3, child.ages=c(9,7,4)).`

Demander `L`, puis `L[[1]]`, `L$name`. Demander `L[[4]]`. Qu'obtient-on ? Demander l'âge du troisième enfant.

Un "data frame" (tableau de données) est une liste dont les vecteurs ont même longueur. Très utilisé pour les tableaux de données, des tableaux où chaque colonne et chaque ligne porte un nom.

2. Jeter un oeil sur `?iris`, puis taper `iris`, `data(iris)`, `class(iris)`, `colnames(iris)` et enfin `iris$Sepal.Length`.

3. (Impoter/Exporter) Exporter le data.frame dans un fichier nommé Iris.txt avec la commande `write.table(iris,"Iris.txt")` et vérifier que le fichier a bien été créé. Si on souhaite l'enregistrer dans un dossier D différent du répertoire de travail, il faut écrire à la place `write.table(iris,"/chemin/D/Iris.txt")`. Importer à nouveau dans R le fichier avec `x <- read.table("Iris.txt")` et demander `x`.

2 Graphiques : la commande "plot"

La commande `plot`, de syntaxe générale `plot(x,y,options)`, trace `y` en fonction de `x`. Voici quelques options utiles :

- `type = "p"` (points), `"l"` (lignes), `"s"` (fonction en escalier).
- `main = "titre"` (pour donner un titre au graphique)
- `xlab = "x label"` (nom de l'axe des abscisses)
- `ylab = "y label"` (nom de l'axe des ordonnées)
- `col = "black"`, ou `red`, `blue`, ... (couleur du graphique)
- `pch = 1`, ou 2, 3... (style des points)
- `lty= 1` ou 2, 3... (style des lignes)
- `xlim = c(a,b)` (restreint la variable `x` à $[a, b]$)
- `ylim = c(a,b)` (restreint la variable `y` à $[a, b]$)

1. Créer un vecteur $x = (0, 0.1, 0.2, \ldots, 2\pi)$ et taper

`plot(x, sin(x), type="l", col="blue", ylim=c(-1,1)).`

Jouer un peu avec les options.

2. Pour superposer un deuxième graphique au premier, on utilise `points` pour dessiner des points, ou `lines` pour tracer des lignes. Taper

`lines(x, cos(x), col="red").`

Que se passe-t-il si on utilise la commande `plot` à la place de `lines` ?

3 Simuler des variables aléatoires

R connaît la plupart des lois de probabilité usuelles, après avoir chargé le package "stats" via la commande `library(stats)`. De façon générale, on peut :

- simuler une variable aléatoire de loi "`loi`" grâce à la commande `rloi`
- donner sa fonction densité $f(\cdot)$ si la variable est continue, ou la fonction $\mathbb{P}(X = \cdot)$ si la variable X est discrète : `dloi`
- donner sa fonction de répartition $F(u) = \mathbb{P}(X \leq x)$: `ploi`
- donner sa fonction quantile $F^{-1}(\alpha)$: `qloi`.

On pourra par exemple remplacer "`loi`" par `norm`, `exp`, `unif`, `binom`, `pois`, pour utiliser une loi normale, exponentielle, uniforme, binomiale, ou de Poisson respectivement. La liste des lois est accessible en tapant `?Distributions`. Quant à l'utilisation des paramètres de chaque loi, on consultera l'aide.

1. Simuler un échantillon `E` de taille 10000 d'une variables de loi normale $\mathcal{N}(0, 1)$. Utiliser la commande `hist` pour tracer un histogramme de cet échantillon, et jouer avec l'option `breaks`, puis rajouter l'option `freq = FALSE`. Commenter. Calculer `mean(E)`. Pouvait-on avoir une idée de ce résultat sans le calculer ? Même chose avec une loi normale $\mathcal{N}(3, 25)$.

2. On considère une variable X de loi binomiale $\mathcal{B}(10, 0.7)$. Quelle est la probabilité que $X = 3$? que $X \leq 8$? et que $X \geq 4$?

3. Soit X une variable de loi exponentielle de paramètre 2. Tracer l'histogramme d'un échantillon de taille 5000 de cette variable. Donner u tel que $\mathbb{P}(X \leq u) = 0.05$.

4. Simuler deux échantillons `E1` et `E2` de taille 1000 d'une variable de loi normale $\mathcal{N}(0, 1)$, puis représenter graphiquement `E1` en fonction de `E2`. Sur le même graphique, tracer en rouge les cercles centrés en zéro de rayons 1 et 2. Enregistrer l'imagine obtenue dans votre dossier personel.

Bonus : Si vous avez terminé le TP, calculer la loi des rayons $\sqrt{E_1^2 + E_2^2}$. Vérifier votre résultat théorique en le comparant à l'histogramme des observations.

4 Boucles

1. Structure d'une "boucle for" : `for (conditions) {lignes de code}`.

Exemple : `for (i in 1:10) {print(i)}`

2. Structure d'une "boucle if" : `if (condition) {lignes de code} else {lignes de code}`
NB : Il n'est pas nécessaire d'avoir un "else" dans la structure.

Exemple :

```
A <- matrix( ,8,8)
for (i in 1:8){
  for (j in 1:8){
    if(i==j) {A[i,j] <- 0} else {A[i,j] <- i+j}
  }
}
```

Donner la formule des coefficients A_{ij} de la matrice A obtenue.

3. Structure d'une "boucle while" : `while (condition) {lignes de code}`

Exemple :

```
seuil <- 1000; n <- 0; s <- 0
while (s<= seuil){n <- n+1; s <- s+n}
c(n,s)
```

Que représente `c(n,s)` ?

5 Fonctions

Il est souvent pratique de créer ses propres fonctions. La syntaxe générale est la suivante :

```
ma_fonction <- function(variable1, variable2, ...) {
... lignes de code ...
return(résultat)
}
```

On peut ainsi écrire une fonction $S($`seuil`$)$ pour systématiser l'exemple de la boucle "while" :

```
S <- function(seuil){
  n <- 0; s <- 0
  while (s<= seuil){
    n <- n+1; s <- s+n
  }
  return(c(n,s))
}
```

Quel est le plus grand entier naturel n tel que $1 + 2 + \cdots + n \leq 987$?

1. Créer une fonction renvoyant la somme de deux nombres.

2. Créer une fonction renvoyant le $n^{\text{ième}}$ terme de la suite de Fibonacci (u_n) définie par la récurrence $u_{n+1} := u_n + u_{n-1}$ pour $n \geq 1$ avec $u_0 := 1$ et $u_1 := 1$.