Initiation au logiciel R

Support du cours pour les L3 EURIA

Année 2022-2023

8 septembre 2022

1 Introduction

R est un langage de programmation interactif interprété et orienté objet qui contient une très large collection de méthodes statistiques ainsi que des facilités graphiques importantes. Initié dans les années 90 par Robert Gentleman et Ross Ihaka (Département de Statistique, Université d'Auckland, Nouvelle-Zélande), auxquels sont venus depuis s'ajouter de nombreux chercheurs, le logiciel R constitue aujourd'hui un langage de programmation intégré d'analyse statistique. Le site Internet du "R coredevelopment Team"
http ://www.r-project.org
est la meilleure source d'informations sur le logiciel libre R. Vous pourrez y trouver les différentes distributions du logiciel, de nombreuses bibliothèques de fonctions et des documents d'aide. Des bibliothèques supplémentaires (appelés 'packages' ou 'library') sont aussi disponibles sur le "comprehensive R archive network"(CRAN)
http ://lib.stat.cmu.edu/R/CRAN/
L'apprentissage d'un langage de programmation est basée sur l'exposition à un maximum de codes et surtout sur la pratique. C'est pourquoi ces notes expliquent seulement de manière très synthétique les points les plus importants. En revanche, de nombreux exemples sont donnés dans chaque paragraphe. Le lecteur est appelé à exécuter et comprendre ces exemples. Des exercices sont ensuite donnés à la fin de chaque paragraphe. Ils ont pour objectif de réutiliser les notions vues dans les différents paragraphes et de vérifier qu'elles sont bien maîtrisées.

2 RStudio

RStudio est un environnement de développement intégré pour R. Il forme une interface utilisateur simple, structurée autour d'une barre de menu et de diverses fenêtres. Les menus sont très peu développés. Leur objectif est de fournir un raccourci vers certaines commandes parmi les plus utilisées, mais leur usage n'est pas exclusif, ces mêmes commandes pouvant être pour la plupart exécutées depuis la console.
Les menus File et Session contiennent les outils nécessaires à la gestion de l'espace de travail, tels que la sélection du répertoire par défaut, le chargement de fichiers sources externes, le chargement et la sauvegarde d'historiques de commandes, etc...

- Le menu Edit contient les habituelles commandes de copier-coller et de recherche.
- Les menus Code et Debug sont très utiles pour développer et debugger des programmes.
- Le menu View permet de personnaliser de l'apparence de l'interface, tandis que le menu Plots permet de gérer les fenêtre graphiques.

- Le menu `Session` permet de personnaliser l'apparence de l'interface.
- Le menu `Tools` automatise la gestion et le suivi des librairies (packages) de fonctions, permettant leur installation et leur mise à jour de manière transparente au départ du site CRAN (Comprehensive `R` Archive Network, http ://cran.r-project.org/) ou de toute autre source locale ainsi que l'importation de données.
- Enfin, `Help` permet l'accès à l'aide en ligne et aux manuels de références de `R`.

Parmi les fenêtres qui s'ouvrent quand on lance `RStudio` , on distingue en particulier :

- La **console**, fenêtre principale où sont exécutées les commandes et où apparaissent les résultats.
- L'éditeur de texte dans lequel on tapera les commandes. On peut ensuite exécuter simplement les commandes dans la console en sélectionnant une ligne ou un bloc de commandes, puis en cliquant sur "run" ou en tapant CTRL+Entrée. Les commandes sont alors copiées dans la console et exécutées. On sauvera régulièrement le fichier texte afin de pouvoir retrouver les commandes lors des TD suivants (on pourra créer un fichier par TD ou chapitre)

A ces fenêtres peuvent s'ajouter un certain nombre de fenêtres facultatives, telles que les fenêtres graphiques et les fenêtres d'informations (historique des commandes, objets en mémoire, aide, visualisation de fichier, etc...).

3 Les objets

Les éléments de base du langage `R` sont des objets qui peuvent être des vecteurs, des matrices, des tableaux de données, des séries chronologiques, des fonctions, des graphiques... Les objets `R` se différencient par leur mode, qui décrit leur contenu, et leur classe, qui décrit leur structure.

Les différents modes sont :

- Entier : `integer`
- Logique (ou booléen) TRUE/FALSE : `logical`
- Numérique : `numeric`
- Complexe : `complex`
- Caractère : `character`

Les principales classes d'objets sont :

- Les vecteurs : `vector`
- Les matrices : `matrix`
- Les tableaux : `array`
- Les listes : `list`
- Les tableaux de données : `data.frame`
- Les séries temporelles : `time.series`

On peut avoir des vecteurs, matrices, tableaux, séries temporelles de mode `integer`, `logical`, `numeric`, `complex`, `character`, mais les modes doivent être tous identiques (par exemple, un vecteur ne peut pas contenir à la fois des nombres et des chaînes de caractères), alors que dans une liste ou un tableau de données les modes peuvent être hétérogènes (par exemple, on peut avoir une colonne de nombres donnant l'age des individus et une colonne de caractères donnant le sexe des mêmes individus).

3.1 Les vecteurs

Il s'agit de l'objet de base dans `R`. Un vecteur est une entité unique formée d'une collection ordonnée d'éléments de même mode (c'est à dire `numeric`, `integer` ou `logical`,...). La constitution manuelle d'un vecteur est réalisée grâce à la fonction `c(el1,el2, ...)`.

Les nombres décimaux doivent être encodés avec un point décimal (et pas une virgule), les chaînes de caractères entourées par des guillemets doubles " ", et les valeurs logiques sont codées par les chaînes de caractères `TRUE` et `FALSE` ou leurs abréviations respectives `T` et `F`. Enfin, les données manquantes sont codées par la chaîne de caractères NA.

Méthode de travail. Vous allez maintenant exécuter les séquences suivantes. L'objectif est de comprendre tous les résultats obtenus. Vous pouvez copier-coller les commandes dans une fenêtre script puis les exécuter ensuite (par exemple en cliquant sur le bouton 'run' en haut à droite de la fenêtre script avec le raccourci clavier ctrl+entrée (sous Windows). Le signe # signale les commentaires : tout ce qui est situé sur une ligne après un signe # n'est pas exécuté. Des exercices sont proposés à la fin de chaque paragraphe pour vérifier que les principales notions ont été comprises.

Création de vecteurs avec la fonction `c` *:*

```
c(5,6,1)                    #Création manuelle d'un vecteur
a1=c(5,6,1)                 #"Dans l'objet a1 mettre le vecteur (5,6,1)"
a1                          #Affiche l'objet a1 dans la console
a2=c(4.3,-5)                #on peut remplacer <- par =
a=c(a1,a2)                  #c permet aussi de concaténer des vecteurs
a                           #affichage dans la console
?c                          #Aide sur la fonction c
texte=c("grand","petit")    #Création d'un vecteur de caractères
```

Affichage des résultats. Lorsqu'on veut savoir ce que contient un objet, on peut taper le nom de l'objet dans la console comme dans les exemples ci-dessus. Le contenu de l'objet est alors affiché dans la console. Sous `RStudio` , on peut aussi regarder dans la fenêtre 'environment' (en haut à droite par défaut) qui décrit toutes les variables présentes dans l'environnement de travail. Pour le moment, votre environnement de travail contient quatre objets, nommés `a`, `a1`, `a2` et `texte` qui sont des vecteurs numériques. Pour éviter les erreurs de programmation lorsqu'on travaille avec R, il faut regarder régulièrement ce que contiennent les objets qui sont créés. Dans la suite, vous devez vous demander à chaque ligne de commande si des nouveaux objets ont été créés ou modifiés (c'est le cas lorsque le signe `=` apparaît), et, lorsque c'est le cas, ce que ces objets contiennent.

Gestion de l'environnement de travail. Il est possible d'effacer les objets créés avec la commande `rm()` ou avec le menu de la fenêtre 'environment'. Exécuter la commande `>rm(a2)` et vérifier que la variable a2 a bien disparu de l'environnement de travail.
Lorsque vous quittez R, vous avez la possibilité de sauvegarder votre espace de travail pour retrouver toutes les variables créées lors de la prochaine session. Faites le test : sauvegarder votre script qui apparaît dans l'éditeur de texte dans un répertoire approprié puis quitter R. `RStudio` vous demande si vous voulez sauver votre environnement de travail : répondez oui. Ouvrez à nouveau `RStudio` : vous retrouvez le même environnement de travail, en particulier les objets créés lors de la session R présente sont à nouveau disponibles.

Extraction/modification des composantes d'un vecteur.

```
a[1]                        # [] permet d'accéder aux composantes d'un vecteur
a[0]                        #Attention : l'indice du premier élément est 1 (avec
                            Python c'est 0)
```

```
a[1] =0
```
#vérifier que la première composante du vecteur `a` a été modifié

```
ind = c(2,4)
a[ind]
a[c(2,4)]
d=a[c(1,3,5)]
x=3:14

x[1:5]=1
```
#Que contient le vecteur ind ?
#Extrait les coordonnées associées au vecteur ind
#Appel plus compact
#Le résultat de l'extraction est stocké dans d
#La commande : est utile pour créer certains vecteurs
#Modification de plusieurs composantes d'un vecteur

Opérations sur les vecteurs.

```
2*a

e=3/d
f=a-4
d=d+e
d=d-e
d*e
e/d
sum(d)
length(d)
t(d)
g=c(sqrt(2),log(10))
(1+exp(2))/cos(8)
```
#Multiplication par 2 de chacune des coordonnées du vecteur a et affichage du résultat
#Remarquer que $1/0 = \text{Inf}$
#Enlève 4 aux composantes de a

#Remarquer que Inf-Inf=NaN ("Not a Number")
#Multiplication terme à terme des vecteurs d et e
#Division terme à terme entre les vecteurs e et d
#Calcul de la somme de d
#Affichage de la dimension du vecteur d
#Transposition du vecteur d

#`R` est aussi une calculatrice

Tests.

```
1==2

1==1
1>2
1!= 2
(1==1) & (1>2)
(1==1) | (1>2)
d==5
a>1 & a<=5
ind=(a>1 & a<=5)
is.vector(a)


is.numeric(a)
is.character(a)
mode(a)
a*a
a[1]='test'
mode(a)
a*a
```
#== réalise un test d'égalité et renvoie un booléen TRUE/FALSE

#!= signifie "différent de"
#& #signifie "ET"
#| signifie "OR"
#Test sur les composantes d'un vecteur

#Le résultat du test est stocké dans ind
#Utilisation de la fonction is.vector() renvoyant l'expression logique TRUE si son argument est un vecteur et FALSE sinon

#On ne peut pas multiplier deux vecteurs de modes `character` !

Extraction d'individus. Lorsqu'on manipule des jeux de données en statistique, on est souvent amené à extraire des individus ayant certaines caractéristiques (par exemple tous les cadres de sexe masculin avec un âge compris entre 30 et 35 ans dans une base de données actuarielle). L'utilisation d'une boucle peut sembler naturelle mais n'est pas optimale en temps de calcul. Il est préférable d'utiliser des tests comme dans les exemples ci-dessous. **D'une manière générale, il faut éviter au maximum d'utiliser les boucles quand on travaille avec R.**

En utilisant which.

```
x=-5:5
x<0
ind=which(x<0)                  #which permet d'extraire les indices TRUE dans un
                                vecteur de booléen
ind
x[ind]
x[-ind]                         #- permet d'accéder aux indices complémentaires
```

Sans utiliser which (plus compact mais moins naturel ?).

```
x=-5:5
ind=x>0                         #ind est un vecteur de booléen
x[ind]                          #renvoie les valeurs de x pour lesquelles ind égal
                                TRUE
x[x>=0]                         #commande la plus compacte!
x[x<0]=10                       #Remplace les valeurs négatives par 10
```

Valeurs manquantes. De nombreux jeux de données comportent des valeurs manquantes (par exemple en actuariat lorsqu'il manque une certaine information sur un individu. Ces valeurs manquantes sont codés avec la chaîne de caractère 'NA' dans R. Cela permet de les identifier et d'utiliser des traitements statistiques adaptés.

```
x=1:10
mean(x)
x[10]=NA                        #NA ("Not Available") permet de coder les valeurs
                                manquantes (jeux de données incomplets)
x
mean(x)                         #renvoie NA car une valeur est manquante
mean(x,na.rm = TRUE)            #Moyenne sans les valeurs manquantes
is.na(x)                        #Utile pour identifier les valeurs manquantes
which(is.na(x))
```

Quelques fonctions utiles.

```
a=rep(1:4,2)
rep(c(1.4,3,5),each=2)
seq(0,1, length=11)
seq(1.575, 5.125, by=0.05)
unique(a)
```

```
b=c(3,4,2,1)
sort(b)
order(b)                          #Permutation à effectuer sur les indices pour trier le
                                  vecteur
order(b)[1]                       #Indice du plus petit élément
order(b)[length(b)]               #Indice du plus grand élément
max(b)                            #Plus grande valeur
which.max(b)                      #Indice du plus grand élément
letters[1:10]
```

Les exercices ci-dessous doivent être faits sans utiliser les boucles !

Exercice 3.1 *Soit* $x = (7, 9, 13, 8, 4, 2, 16, 1, 6, 19, 15, 12, 19, 14, 8, 2, 19, 11, 18, 7)$.

1. *Créer le vecteur x.*
2. *Donner des commandes qui permettent d'extraire les éléments suivants du vecteur x.*
   (a) *Le deuxième élément.*
   (b) *Les cinq premiers éléments.*
   (c) *Les éléments strictement supérieurs à 14.*
   (d) *Tous les éléments sauf les éléments en positions 6, 10 et 12.*

Exercice 3.2 *À l'aide des fonctions rep et seq (par exemple on n'écrira pas c(0,6,0,6,0,6) pour la première question), créer les vecteurs suivants :*

1. *0 6 0 6 0 6*
2. *1 4 7 10*
3. *1 2 3 1 2 3 1 2 3 1 2 3*
4. *1 2 2 3 3 3*
5. *1 1 1 2 2 3*
6. *1 5.5 10*
7. *1 1 1 1 2 2 2 2 3 3 3 3*

Exercice 3.3

1. *Créer le vecteur* $a = (2.5, 3, 1, 0, 4, -1)$.
2. *Ordonner le vecteur a avec la fonction* `sort`. *Que fait la fonction* `order` *? Comment peut-on ordonner a en utilisant cette dernière fonction ?*

Exercice 3.4 *Étant donné un vecteur d'observations* $x = (x_1, x_2, ..., x_n)$ *on calcule sa moyenne harmonique selon la formule suivante* $\frac{n}{\frac{1}{x_1}+...+\frac{1}{x_n}}$ *Calculer la moyenne harmonique du vecteur* $x =$ *(7, 13, 3, 8, 12, 12, 20, 11). On pourra commencer pas créer le vecteur* $(\frac{1}{x_1}, ..., \frac{1}{x_n})$ *avant d'en calculer la somme puis la moyenne harmonique.*

Exercice 3.5

1. *Créer un vecteur* $x = (x_1, ..., x_n)$ *de taille n (on pourra commencer avec* $n = 50$*) qui contient des nombres au hasard entre* 0 *et* 1 *à l'aide de la fonction* `runif` *(après le cours de Probabilités on parlera d'échantillon indépendant de la loi uniforme sur l'intervalle* $[0, 1]$*).*
2. *Créer un vecteur y de même taille que x qui est tel que*
   — $y_i = 0$ *si* $x_i < 0.5$
   — $y_i = 1$ *si* $x_i \geq 0.5$

3. *En déduire le nombre puis la proportion des coefficients du vecteur x qui sont supérieurs à 0.5. Recommencer l'expérience en simulant plusieurs fois l'échantillon x et en testant différentes valeurs de n. Les résultats obtenus sont-ils conformes à votre intuition ?*

Exercice 3.6 1. *Créer un vecteur x de taille n (on pourra commencer avec $n = 50$) qui est tel que $x_i = 1/i$ pour $i \in \{1, ..., n\}$.*

2. *En déduire la valeur de $y_n = \sum_{i=1}^{n} \frac{1}{i} - ln(n)$ pour $n = 50$ puis $n = 100$ et $n = 1000$. La suite (y_n) suite converge t'elle lorsque $n \to +\infty$ (on ne demande pas de prouver le résultat !). Si oui, proposer une valeur approchée de la limite (avec une seule ligne de commande !)*

Exercice 3.7 *On admet que $\pi = 4 \sum_{k=0}^{+\infty} \frac{(-1)^k}{2k+1}$. En déduire une méthode permettant de fournir une valeur approchée de π. Tester votre méthode en vérifiant que vous arrivez à retrouver les 6 premières décimales de π (la commande `R pi` permet d'obtenir une valeur approchée de π).*

3.2 Les matrices

Les matrices, comme les vecteurs, sont de mode quelconque, mais elles ne peuvent pas contenir des éléments de nature différente (c'est à dire qu'une matrice ne peut pas contenir des valeurs numériques et du texte par exemple). La syntaxe de base pour créer une matrice est `matrix(vec,nrow=n,ncol=p,byrow=T)` où `vec` est le vecteur contenant les éléments de la matrice, qui seront rangés en colonne (sauf si l'option `byrow=T` est choisie).

Création d'une matrice.

```
a=1:20
x1=matrix(a,nrow=5)          #Création d'une matrice de dimension 5*4 à partir
                             du vecteur a (les valeurs sont rangées par colonne)
x1
x2=matrix(a,nrow=5,byrow=T)  #Création d'une matrice de dimension 5*4 à partir
                             du vecteur a (les valeurs sont rangées par ligne)
```

Quelques opérations sur les matrices.

```
x3=t(x2)                     #Transposition de la matrice x2
b=x1*x2                      #Produit coefficient par coefficient
b=x3%*%x2                    #Produit matriciel
log(x1)
sum(x1)
```

Extraction de sous-matrices.

```
x3=t(x2)                     #Transposition de la matrice x2
b=x1*x2                      #Produit coefficient par coefficient
b=x3%*%x2                    #Produit matriciel
dim(b)                       #Affichage de dimension de la matrice b
b[3,2]                       #Extraction de l'élément [3,2] de la matrice b
b[3,2] =5                    #Change le coefficient [3,2] de la matrice
b[,2]                        #Extraction de la deuxième colonne de b
b[c(3,4),]                   #Extraction des troisième et quatrième lignes de b
```

```
b[-2,]                          #Suppression de la deuxième ligne de b
b[,-c(2,4)]                     #Suppression des deuxième et quatrième colonnes de b
b[,2]                           #Extraction de la deuxième colonne de b
b[1,]>600                       #Test sur la première ligne de la matrice
ind=which(b[1,]>600)
b[,ind]                         #Extraction des colonnes de b telles que la première ligne est supérieure à 600
b[b>100]                        #Extraction des coefficients de b supérieurs à 700, le résultat est un vecteur
```

Concaténation de matrices.

```
rbind(x1,x2)                    #Concaténation verticale des matrices x1 et x2
cbind(x1,x2)                    #Concaténation horizontale des matrices x1 et x4
```

La fonction apply. La fonction apply permet d'appliquer une fonction qui fonctionne sur les vecteurs aux colonnes ou aux lignes d'une matrice. C'est très utile pour éviter de faire des boucles.

```
apply(x1,2,sum)                 #Calcul de la somme des colonnes de x1.
apply(x1,1,sum)                 #Calcul de la somme des lignes de x1
apply(x1,1,max)                 #Ici on calcule le max par ligne
```

Exercice 3.8

1. *Créer les vecteurs $x = (2;2;2;2;2)$ à l'aide de la fonction* `rep` *et $y = (-1;0;1;2;3)$ à l'aide de la fonction* `seq`.
2. *Créer une matrice A de 5 lignes et 2 colonnes dont la première colonne est formée des éléments de x et la deuxième des éléments de y.*
3. *Modifier la deuxième ligne de A en $(2,4)$.*

Exercice 3.9

1. *Créer la matrice*

$$A = \begin{pmatrix} 2 & 23 & 8 \\ 10 & 6 & 90 \\ 4 & 7 & 12 \end{pmatrix}$$

   à l'aide des fonctions `cbind` *puis* `matrix`.
2. *Calculer la moyenne et le produit des lignes et des colonnes de A à l'aide des commandes* `apply`, `sum` *et* `prod`.
3. *Créer une matrice B qui contient les deux premières ligne de A.*
4. *Calculer la somme des deux premières lignes de A puis la somme de la première et de la troisième colonne.*
5. *Que permettent de calculer les commandes* `t(A)`, `det(A)` *et* `diag(A)` *? En déduire la trace de A.*
6. *Que permettent de calculer les commandes* `A^2`, `A*A` *et* `A%*%A` *?*
7. *Que permettent de calculer les commandes* `1/A`, `A^(-1)` *et* `solve(A)` *?*
8. *Résoudre le système ci-dessous :*

$$\begin{aligned} 2x + 23y + 8z &= 5 \\ 10x + 6y + 90z &= 6 \\ 4x + 7y + 12z &= 7 \end{aligned}$$

9. *Extraire les lignes de la matrice A pour lesquelles la somme des coefficients est supérieure à 30 (si possible avec une seule ligne de commande).*

3.3 Les tableaux de données.

Un tableau de données (`data.frame`) est un tableau dont les colonnes peuvent être hétérogènes. Par exemple, un `data.frame` peut contenir des colonnes de caractères et des colonnes de numériques, mais chacune des colonnes contiendra des éléments de même nature. Il s'agit sans doute de la classe d'objet la plus importante puisqu'elle est consacrée spécifiquement au stockage des données. Chaque colonne décrit une variable alors que les lignes correspondent aux individus. On peut alors mettre dans le `data.frame` des variables qualitatives décrites par des caractères (par exemple une colonne qui décrit le sexe M/F d'un individu ou sa catégorie socio-professionnelle) et des variables quantitatives décrites par des nombres (par exemple le montant des sinistres ou l'âge). Nous verrons ultérieurement que de nombreuses méthodes statistiques sont disponibles dans `R` pour cette classe d'objet .
Pour créer un tableau de données, on peut regrouper des variables de même longueur à l'aide de la commande `data.frame(nom1=var1,nom2=var2,...)`. Il est également possible de transformer une matrice en tableau de données en utilisant la commande `as.data.frame`. Enfin, c'est ce type d'objet qui est généralement créé par `R` lorsque l'on importe des données (voir section 9).

Création et manipulation des `data.frame`

```
v1=LETTERS[1:5]
v2=LETTERS[6:10]
v3=1:5                                  #30 réalisations indépendantes d'une loi uniforme
                                        sur [0,1]
v4=6:10
xx=data.frame(Age=v1,Prenom=v2,Taille=v3,Poids=v4)
                                        #Création d'un data.frame avec 4 variables et 5 in-
                                        dividus.
xx
str(xx)                                 #Donne la structure de l'objet
xx$Prenom
xx[,1]                                  #On peut accéder aux variables avec $ ou []
xx$Prenom[3]                            #Permet d'accéder au troisième élément du vecteur
                                        xx$Prenom
xx[3,1]                                 #Donne le même résultat
summary(xx)                             Résumé statistique du jeu de données
xx=data.frame(v1,v2,as.factor(v3),v4)
                                        #Constitution du tableau de données xx avec la va-
                                        riable v3 déclarée comme une variable qualitative (R
                                        voit alors les nombres comme des chaînes de carac-
                                        tères)
str(xx)                                 #Remarquer que v3 est qualitative
summary(xx)                             #Remarquer que le résultat est différent pour les va-
                                        riables qualitatives et quantitatives
ma=matrix(1:15,nrow=3);ma
ma=as.data.frame(ma)                    #Transformation de ma en un objet de type
                                        data.frame
ma
str(ma)
names(ma)=c('VA','VB','VC','VD','VE')  #Donne des noms aux variables
row.names(ma)=c('I1','I2','I3')         #Donne des noms aux individus
ma
```

Manipulation des variables qualitatives

```
Sexe=sample(c('H','F'),100,rep=T)   #Simulation du sexe (H ou F) (variable qualitative)
Taille=rnorm(100,mean=170,sd=10)    #Simulation de la taille des individus selon une loi
                                    normale de moyenne 170 et écart-type 10
Taille[which(Sexe=='H')]=Taille[which(Sexe=='H')]+10
                                    #On rajoute 10 pour la taille des hommes
tapply(Taille,Sexe,mean)            #Calcul de la taille moyenne par sexe
tapply(Taille,Sexe,sd)              #Calcul de l'écart-type par sexe
```

Il existe des jeux de données disponibles dans R (souvent utilisés dans les sujets d'examen !)

```
data()                              # Ouvre une fenêtre texte listant l'ensemble des ta-
                                    bleaux de données disponibles dans R
women                               #Affiche le tableau de données women
? women                             #Descriptif du tableau de données women
names(women)                        #Affiche le nom des variables de women
attributes(women)                   #Donne les caractéristiques du tableau de données
                                    women
women$height
height
height
apply(women,1,sum)                  #La fonction apply peut être utilisée avec des
                                    data.frame
apply(women,2,max)
```

Exercice 3.10 *On considère dans cet exercice le jeu de données "iris" disponible sous* `R`.

1. *Vérifier que c'est un objet de type* `data.frame`.
2. *Que contient ce jeu de données ?*
3. *Taper la commande* `summary(iris)`. *Combien d'individus de l'espèce* `versicolor` *sont dans la base de données ? Quelle est la moyenne de la* `variable Sepal.Width` *? La plus petite valeur ?*
4. *Créer un objet de type* `data.frame`, *nommé* `iris2`, *qui contient seulement les individus de l'espèce "setosa".*
5. *Trier les individus de l'espèce "setosa" en fonction de la longueur du sépale.*
6. *Retrouver les moyennes des variables* `Sepal.Length`, `Sepal.Width`, `Petal.Length` *et* `Petal.Width` *à l'aide des fonctions* `apply` *et* `mean`.
7. *Calculer les moyennes de la variable* `Sepal.Length` *pour chacune des trois espèces à l'aide des fonctions* `tapply` *et* `mean`.

Exercice 3.11 *Le tableau suivant détaille le nombre d'étudiants des universités de cinq académies inscrits en 2006 dans les cinq filières suivantes : Lettres, Sciences, Médecine, Sport et I.U.T.*

	Lettres	*Sciences*	*Médecine*	*Sport*	*I.U.T.*
Bordeaux	*12220*	*6596*	*7223*	*357*	*2239*
Lyon	*15310*	*6999*	*10921*	*395*	*3111*
Paris	*112958*	*40244*	*46146*	*1247*	*7629*
Rennes	*8960*	*6170*	*4661*	*279*	*4013*
Toulouse	*12125*	*8233*	*6653*	*553*	*3178*

1. *Créer un objet z de type* `data.frame` *qui contient ce jeu de données. On renseignera en particulier le nom des colonnes (avec la commande* `names`*) et des lignes (avec la commande* `row.names`*).*
2. *Calculer le nombre total d'étudiants dans chaque académie puis ordonner le tableau de données par ordre croissant de cette variable.*
3. *Calculer le nombre total d'étudiants dans chaque filière puis ordonner le tableau de données par ordre croissant de cette variable.*
4. *Créer un data.frame qui contient seulement les académies pour lesquelles l'effectif en sciences est supérieur à l'effectif en médecine (une seule ligne de commande).*

3.4 Les listes

Une liste est une collection ordonnée d'objets, non nécessairement de même mode. Les éléments des listes peuvent donc être n'importe quel objet défini dans R. On peut par exemple créer une liste qui contient un vecteur numérique et une matrice de caractères ! Cette propriété est notamment utilisée par certaines fonctions pour renvoyer des résultats complexes sous forme d'un seul objet. La constitution d'une liste passe par la fonction `list(nom1=el1,nom2=el2,...)`. On peut accéder à chaque élément de la liste à l'aide de son index entre double crochets [[...]], ou par son nom précédé du signe $.

```
li=list(num=1:5,y="couleur",a=TRUE)#Création d'une liste avec 3 objets
li
li$num
li$a
li[[1]]                          #On peut accéder aux éléments d'une liste avec $ ou
                                 avec [[]]
li[[3]]
a=matrix(c(6,2,0,2,6,0,0,0,36),nrow=3)
eigen(a)                         #Calcul des éléments propres de la matrice
res=eigen(a)                     #Le résultat est donné dans une liste
attributes(res)                  #Donne le nom des éléments de la liste
str(res)                         #
res$values                       #Valeurs propres
res$vectors                      #Vecteurs propres
diag(res$values)
res$vectors%*%diag(res$values)%*%t(res$vectors)
x <- list(a=1:10,beta=exp(-3:3),logic=c(TRUE,FALSE,FALSE,TRUE))
lapply(x, mean)                  #La commande lapply permet d'appliquer une fonc-
                                 tion à chaque élément d'une liste
```

Exercice 3.12 *Créer une liste qui contient votre nom de famille, le prénom de vos parents et votre département de naissance.*

Exercice 3.13 *On reprend la matrice A de l'exercice 3.9.*

1. *Calculer les valeurs propres et les vecteurs propres de A. La matrice est-elle diagonalisable ?*
2. *Déduire de la question précédente le calcul de A^n pour $n = 10$ et $n = 50$ (sans utiliser une boucle !).*

4 Quelques éléments de programmation

Nous traitons succinctement dans cette section des instructions de condition et de répétition. Il s'agit des commandes `if`, `while`, `for`, `repeat`.
Une boucle `repeat` est toujours exécutée au moins une fois. Elle devra comporter un test d'arrêt qui utilisera la commande `break`. La commande `break` peut aussi être utilisée pour sortir d'une boucle `for` ou `while`. La commande `next` permet le passage immédiat à la prochaine itération d'une boucle for, while ou repeat.
Le bloc d'instruction est délimité par des accolades (certains langages utilisent les commandes begin et end). La condition est délimitée par des parenthèses.
Boucle `for`.

```
x=NULL                              #Initialisation
for (i in 1:10){                    #i va varier de 1 à 10
 x[i]=i                             #Instruction exécutée pour i=1 puis i=2...puis i=10
}                                   #Indique la fin de la boucle for
 x
s=0
x=rnorm(10000)
for (i in 1:10000){
 if (x[i]<10){
  s=s+x[i]
} else{
  s=s+2*x[i]
}
}
s
sum(x[x<10])+2*sum(x[x>=10])        #Permet d'éviter la boucle for : plus efficace !
```

Lorsqu'on commence un bloc d'instruction avec le signe {, le prompt est modifié dans la console et le signe + apparaît. Afin de forcer `R` à sortir du bloc d'instruction, on peut appuyer sur la touche `echap`. Faites le test suivant : en utilisant la flèche ↑ du clavier, remontez jusqu'à l'instruction `for (i in 1:10000){` et exécutez là. Observez le prompt : vous pouvez taper n'importe quelle instruction dans la console sans qu'elle soit exécutée. Appuyer sur la touche `echap` pour récupérer la main.

Boucle `while`.

```
x=NULL                              #Initialisation
i=0                                 #Initialisation
while(i<10){                        #Condition d'arrêt : R exécute le bloc d'instruction
                                    tant que i est inférieur à 10
 i=i+1                              #Sans cette instruction la boucle while est sans fin
 x[i]=i
}                                   #Fin du while
x
```

Boucle `repeat`.

```
x=NULL
i=1
```

```
repeat{
 x[i]=i
 if (i>=10){
 break                          #On sort de la boucle repeat lorsque i dépasse 10
 }                              #Fin du if
 i=i+1
 }                              #Fin du repeat
```

Il faut éviter d'utiliser des boucles avec `R` puisqu'elles sont généralement inefficaces (temps de calcul importants). Il est souvent possible de vectoriser les calculs pour éviter les boucles explicites (les opérateurs matriciel de `R` utilisent des boucles C beaucoup plus rapides), ou encore d'utiliser les fonctions `outer`, `apply`, `lapply`, `tapply` et `mapply` pour réaliser les boucles de manière plus efficace.

```
x=matrix(rep(1,10^8),nrow=10^4)     #Création d'une grande matrice
a=Sys.time()                        #Utilisé pour calculer les temps de calcul
S=rep(0,nrow(x))                    #Calcul de la somme des lignes avec des boucles for
                                    imbriquées
for (i in 1:nrow(x)){
 tmp=0
 for (j in 1:ncol(x)){
 tmp=tmp+x[i,j]
 }
 S[i]=tmp
}
b=Sys.time()
S2=apply(x,1,sum)                   #Calcul de la somme des lignes avec apply
c=Sys.time()
S3=x%%rep(1,nrow(x))                #Calcul de la somme des lignes avec un produit ma-
                                    triciel
d=Sys.time()
c(b-a,c-b,d-c)                      #Temps écoulé pour chaque méthode : le produit
                                    matriciel est beaucoup plus rapide!
```

Exercice 4.1 *Reprendre la deuxième question de l'exercice 3.5 en utilisant une boucle `for` et la commande `if`. Recommencer avec un boucle `while` puis une boucle `repeat`.*

Exercice 4.2 *On considère la suite $(U_n)_{n\geq 1}$ définie par $U_1 = 10$ et $U_{n+1} = \sqrt{U_n}$ pour $n \geq 1$.*

1. *Ecrire une boucle `for` qui calcule les n premiers termes de la suite (U_n) (la boucle renverra un vecteur de taille n).*
2. *Ecrire une boucle `while` qui calcule les n premiers termes de la suite (U_n) avec $n \geq 1$ le plus petit entier tel que $|U_{n+1} - U_n| < \epsilon$ et ϵ une valeur fixée. Donner la valeur de n correspondant à $\epsilon = 10^{-4}$.*
3. *Reprendre la question précédente avec une boucle `repeat`.*

Exercice 4.3 *Si l'année A n'est pas divisible par 4, alors elle n'est pas bissextile. Si A est divisible par 4, l'année est bissextile sauf si A est divisible par 100 et pas par 400. Exemples :*

- *1901 n'est pas bissextile car non divisible par 4*
- *2004 est bissextile car divisible par 4 et pas par 100*
- *2100 n'est pas bissextile car divisible par 4, divisible par 100 mais pas par 400*
- *2000 est bissextile car divisible par 4, par 100 et par 400*

Ecrire un programme qui détermine si une année A est bissextile ou non. On pourra utiliser que la commande A%%4 permet de calculer le reste de la division euclidienne de A par 4. On testera le programme sur les exemples ci-dessus

Exercice 4.4 *A l'aide de boucles* `for` *imbriquées, calculer les N premières lignes du triangle de Pascal à l'aide de la formule de récurrence* $\binom{n+1}{k} = \binom{n}{k-1} + \binom{n}{k}$ *pour* $k \in \{1...n\}$ *et stocker le résultat dans une matrice carré de taille N.*

Exercice 4.5 *Un palindrome est un mot pouvant se lire aussi bien de gauche à droite que de droite à gauche. Par exemple le mot RADAR est un palindrome. Ecrire un programme qui détermine si une chaîne de caractères est un palindrome. On pourra utiliser les fonctions suivantes :*

- *substr permet d'extraire une partie d'une chaîne de caractères*
- *nchar donne la taille d'une chaîne de caractères*

Par exemple si vous testez votre code avec le mot RADAR, R devra retourner "RADAR est un palindrome" (on utilisera la fonction paste). Tester votre fonction avec différents mots, par exemple avec "kayak", "fané", "tarte", "été",. . .

5 Construction d'une nouvelle fonction

Le logiciel R dispose de fonctions préprogrammées, appelées "'primitives"'. Nous en avons déjà rencontré plusieurs : `c, mean, sum, log, sqrt`... L'utilisateur a aussi la possibilité de définir ses propres fonctions lorsqu'elles n'existent pas déjà dans `R`. Une fonction est un sous-programme, c'est-à-dire une portion de code qui est exécutée lorsqu'on l'appelle.
De manière générale, la définition d'une nouvelle fonction passe par l'expression suivante :

```
nom_fonction=function(arg1[=expr1],arg2[=expr2],...) { bloc d'instructions }
```

Les accolades signalent à l'interpréteur de commande le début et la fin du code source de la fonction ainsi définie, tandis que les crochets ne font pas partie de l'expression mais indiquent le caractère facultatif des valeurs par défaut des arguments. Il faut ensuite soumettre les fonctions au logiciel, soit en exécutant le bloc de commandes correspondant, soit en utilisant la fonction `source`.
Lors de l'exécution, `R` renvoie par défaut **la dernière expression** évaluée dans la fonction. On peut également utiliser la fonction **return** pour spécifier la sortie.

Un premier exemple.

```
puissance=function(x,n) {
 y=x^n
 return(y)                          #Indique l'objet à retourner
 }                                  #Indique la fin de la fonction
puissance(3,2)                      #R exécute les commandes de la fonction carre avec
                                    x=2 et renvoie l'objet désigné par return
```

Variables locales. A l'intérieur d'une fonction, les variables sont locales : les arguments sont passés à la fonction par valeur et leur portée ainsi que celle de toute assignation classique à l'intérieur d'une fonction est locale. Lors de l'exécution, une copie des arguments est transmise à la fonction et la fonction ne modifiera pas les valeurs des variables disponibles dans l'environnement de travail. Cependant, on peut utiliser les objets de l'environnement de travail dans les fonctions comme dans l'exemple ci-dessous. Attention, c'est souvent une source d'erreur

de travailler ainsi et il est généralement préférable de déclarer en entrée toutes les variables qui sont utilisées dans la fonction !

```
puissance2=function(x) {          #la valeur de n n'est plus passée en entrée
 y=x^n;
 return(y)                        #Indique l'objet à retourner
 }                                #Indique la fin de la fonction
rm(n)                             #Efface l'objet n de l'environnement de travail
y=0
puissance2(5)                     #Renvoie un message d'erreur car n est inconnu
n=3
puissance2(5)                     #La fonction puissance est exécutée avec la valeur
                                  de n disponible dans l'environnement de travail
puissance2(5)
y                                 #La valeur de y n'est pas modifiée (la fonction
                                  puissance a créé une variable y locale)
```

Un exemple plus sophistiqué. Lorsque la fonction doit retourner plusieurs objets, on utilise généralement une liste comme dans l'exemple ci-dessous. On peut aussi donner des valeurs par défaut à certains paramètres d'entrée.

```
loto=function(a=5) {              #a est le nombre de tirage : 5 par défaut
 d=date()                         #crée une chaîne de caractère avec l'heure et la date
 n=sample(x=1:50,size=a)          #Vecteur avec a nombres au hasard entre 1 et 50
 print(paste('Tirage du',d,       #Affichage de la date
':'),quote=FALSE)
 print(n)                         #Affichage des nombres tirés
 li=list(dat=d,alea=n)            #Création de l'objet à retourner
 return(li)                       #Indique l'objet à retourner
 }                                #Indique la fin de la fonction
res=loto()                        #R exécute la fonction avec la valeur par défaut
res$alea
loto(8)
```

Enfin, ajoutons qu'il est fortement recommandé d'ajouter des commentaires au code des fonctions, en les faisant précéder du symbole dièse #. La suite de la ligne est alors ignorée lors de l'interprétation de la fonction et peut donc être complétée par un commentaire libre.

Exercice 5.1 *Ecrire une fonction* `bissextile=function(A)` *qui teste si une année A est bissextile (cf exercice 4.3).*

Exercice 5.2
- *Ecrire une fonction nommée* `facto=function(n)` *qui calcule* $n!$.
- *Calculer les valeurs de* $n!$ *pour* $n \in \{1, ..., 20\}$ *à l'aide des fonctions* `facto` *et* `apply`.
- *Ecire une fonction* `comb=function(n,k)` *qui calcule le coefficient binomial* $\binom{n}{k} = \frac{n!}{k!(n-k)!}$ *(la fonction* `comb` *appellera la fonction* `facto`*)*

Exercice 5.3 *Ecrire une fonction* `puiss_mat=function(A,n)` *qui calcule la puissance nième d'une matrice A en utilisant une boucle for.*

Exercice 5.4 *On considère les deux suites* $(a_n)_{n\in\mathbb{N}}$ *et* $(b_n)_{n\in\mathbb{N}}$ *définies par* $a_0 = 1$ *et* $b_0 = 2$ *et les relations de récurrence*

$$a_{n+1} = \frac{1}{2}(a_n + b_n)$$
$$b_{n+1} = \sqrt{a_n b_n}$$

pour $n \in \mathbb{N}$.

1. *Écrire une fonction* `R` *qui permet de calculer les* N *premiers termes des suites* (a_n) *et* (b_n) *ou* N *désigne un entier naturel arbitraire (on utilisera une boucle* `for`*). En sortie la fonction renverra une liste qui contient les éléments des deux suites (une fonction* `R` *peut renvoyer seulement un objet). En déduire les valeurs de* a_{10} *et* b_{10}*.*
2. *Représenter graphiquement les 20 premiers termes de la suite* a_n *en utilisant* `R` *et reproduire le graphique schématiquement sur votre copie. Pensez-vous que la suite* (a_n) *est monotone et/ou convergente ? (on ne demande pas de démonstration mathématique !). Si la suite est convergente, proposer une valeur numérique approchée pour la limite.*
3. *Représenter sur le même graphique les 20 premiers termes de la suite* b_n *en utilisant* `R` *et reproduire le graphique schématiquement sur votre copie. Pensez-vous que la suite* b_n *est monotone et/ou convergente ? (on ne demande pas de démonstration mathématique !).*

Exercice 5.5 *On rappelle qu'un entier naturel* n *est un nombre premier si et seulement si il admet deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même). Par exemple 4 n'est pas un nombre premier (puisqu'il est divisible par 2) mais 5 est un nombre premier (puisqu'il est divisible seulement par 1 et par 5). Un algorithme simple pour vérifier si un entier* n *est un nombre premier est donc de tester si il est divisible par les nombres entiers compris entre 2 et* $\sqrt{n}$*.*

1. *Créer une fonction avec pour entête* `est.premier=function(n)` *qui renvoie*
   — *Un message d'erreur si* n *n'est pas un entier naturel*
   — *Un booléen de valeur TRUE si* n *est un nombre premier*
   — *Un booléen de valeur FALSE si* n *n'est pas un nombre premier*
   On pourra utiliser la commande `%%` *qui donne le reste de la division euclidienne (par exemple* `5%%3` *renvoie 2)*
2. *En 1640, Fermat a conjecturé que les nombres de la forme*

$$F_n = 2^{2^n} + 1$$

   pour $n \in \mathbb{N}$ *sont des nombres premiers. Vérifier si* F_1*,* F_2*,* F_3*,* F_4 *et* F_5 *sont bien des nombres premiers.*

Exercice 5.6 *Un vecteur* $a = (a_1, ..., a_n)$ *forme une partition de l'intervalle* $[0, 1]$ *si il vérifie les conditions suivantes :*

— $a_1 = 0$ *et* $a_n = 1$
— *la suite est strictement croissante :* $a_1 < a_2 < ... < a_n$*.*

1. *En utilisant une boucle* `for`*, écrire une fonction* `R` *dont l'entête est* `test_partition=function(a)` *qui renvoie un booléen avec*
   — *la valeur* `TRUE` *si le vecteur* `a` *forme une partition de de l'intervalle* $[0, 1]$
   — *la valeur* `FALSE` *si le vecteur* `a` *ne forme pas une partition de l'intervalle* $[0, 1]$.
   Tester votre fonction sur le vecteur $a = (0, \frac{1}{n}, \frac{2}{n}, ..., \frac{n-1}{n}, 1)$ *avec* $n = 100$ *puis sur le vecteur* $a = (0, \frac{2}{3}, \frac{1}{3}, 1)$*.*
2. *Reprendre la question précédente sans utiliser une boucle.*
3. *En utilisant une boucle* `while`*, écrire une fonction* `R` *dont l'entête est* `indice=function(a,x)` *qui renvoie*
   — *l'indice* i *tel que* $a_i \leq x < a_{i+1}$ *si cet indice existe et si* `a` *est une partition de l'intervalle* $[0, 1]$
   — *un message d'avertissement sinon*
   Tester votre fonction avec les vecteurs a *de la question précédente et* $x = 0.8$ *puis* $x = 2$*.*
4. *Reprendre la question précédente sans utiliser une boucle.*

6 Graphiques

La méthode basique pour créer un graphique est d'utiliser la commande `plot(x,y)` avec `x` un vecteur qui contient l'abcisse des points à représenter et `y` un autre vecteur qui contient l'donnée des points à représenter. La commande `plot` permet aussi de fixer les limites des axes, de donner des noms aux axes et à la figure, etc. Une aide détaillée sur les deux options peut être obtenue en tapant la commande `? plot.default`. On peut ensuite rajouter des points ou des lignes avec les commandes `points` ou `lines`.

Utilisation de la fonction `plot` pour créer un nouveau graphique.

```
x=1:10
y=x*x
plot(x,y)
plot(x,y,type='l')                  #La courbe obtenue est constituée de segments.
x=seq(1,10,by=.01)                  #Il faut discrétiser plus finement pour que le graphique obtenu ressemble à une parabole
y=x*x
plot(x,y,type='l')
plot(x,y,type='l',col='red')        Couleurs classiques : red, blue, cyan, yellow, green,...
colors()                            #Liste des couleurs disponibles
plot(x,y,type='l',col='red',xlab='x',ylab='cos(x)',main='Titre')
                                    #N'oubliez pas de donner des noms à vos axes pour les rapports
```

Utilisation des fonctions `points` et `lines` pour ajouter des éléments sur un graphique existant.

```
y2=exp(x)
points(x,y2,col='blue')             #Points permet de rajouter des points sur le graphique
points(x,y2,col='blue',pch='o')
lines(x,y2,col='gray')              #Lines permet de rajouter des lignes sur le graphique
plot(x,y2,type='l',col='green')     #Vérifier qu'on peut passer d'une fenêtre graphique à l'autre
colors()                            #Liste des couleurs disponibles
par(mfrow=c(1,2)                    #Divise la fenêtre graphique en 1*2 sous-fenêtres
plot(x,sin(x),type='l')
plot(x,cos(x),type='l')
```

`Gestion des différentes fenêtres graphiques.` Avec `RStudio`, une nouvelle fenêtre graphique est créée à chaque exécution de la commande `plot` et les anciennes fenêtres sont conservées. Les commandes précédentes ont donc créé plusieurs fenêtres graphiques différentes. Vous pouvez passer d'un graphique à l'autre en utilisant les flèches situées au dessus du graphique. Il faut aussi penser à supprimer les graphiques inutiles pour éviter de surcharger la session `R`. Pour cela, on peut les boutons situés au dessus de la fenêtre graphique. Par exemple, en cliquant sur le balai on efface tous les graphiques existants. On peut aussi utiliser la commande en ligne `dev.off()` .

Utilisation des fonctions existantes dans R pour faire des graphiques spécifiques. Nous verrons par la suite que R contient des fonctions (*hist*, *pie*, *boxplot*, *qqplot*,...) pour faire certains graphiques spécifiques (respectivement histogrammes, camemberts, boîtes à moustache,

'Quantile-Quantile' plot,...). Nous verrons également qu'il existe des packages dédiés à la réalisation de graphiques : par exemple *ggplot2* pour les statistiques descriptives ou *leaflet* pour les cartes.

Exercice 6.1 *1. Tracer la fonction* sin *sur l'intervalle* $[0, 10]$.

2. *Reprendre le graphique précédent en imposant les limites* $[-1.5, 1.5]$ *pour l'axe des ordonnées (option* `ylim`*). On donnera le titre 'Graphique de la fonction sinus' au graphique, le nom 'Angle' à l'axe des abscisses et le nom 'Sinus' à l'axe des ordonnées.*
3. *Rajouter la droite d'équation* $y = \frac{x}{2\pi}$ *sur le graphique précédent à l'aide la fonction* `lines`*. On tracera cette droite en rouge.*
4. *Rajouter les points de coordonnées* $(0, 1.3)$ *et* $(2, -1)$ *sur le graphique précédent à l'aide de la fonction* `points`*. On représentera ces points à l'aide du symbole '*' colorié en bleu.*

Exercice 6.2 *Séparer la fenêtre graphique en 2*2 parties puis tracer*
— *la fonction* cos *en haut à gauche,*
— *la fonction* sin *en haut à droite,*
— *la fonction* exp *en bas à gauche,*
— *la fonction* ln *en bas à droite.*
On donnera des titres aux différents graphiques et des noms aux axes.

Exercice 6.3 *1. Écrire une fonction phi servant à calculer la fonction de densité de probabilité d'une loi normale centrée réduite. La fonction prendra en argument un réel (ou un vecteur) x.*

2. *Ensuite regarder l'aide de R concernant la fonction* dnorm *et comparer les résultats obtenus entre cette fonction et la fonction que vous venez de créer.*
3. *Tracer la fonction phi sur l'intervalle* $[-3, 3]$

7 Quelques fonctions utiles

Quelques fonctions de statistique exploratoire.

```
mean(women$height)                    #Calcul de la moyenne empirique de la variable height
var(women$height)                     #Calcul de la variance empirique (estimateur non biaisé)
sd(women$height)                      #Calcul de l'écart-type de height
median(women$height)                  #Calcul de la médiane empirique
quantile(women$height)                #Calcul des quantiles empiriques
summary(women$height)                 #Résumé de la variable height
summary(women)                        #Résumé de women
hist(women$height,nclass=5)           #Histogramme de weight
boxplot(women$weight)                 #Diagramme en boîte de weight
cor(women$height,women$weight)        #Calcul du coefficient de corrélation linéaire empirique entre weight et height
v1=rnorm(100)                         #Simulation de valeurs selon un loi de Gauss (cf cours de proba!)
hist(v1)
v2=factor(sample(letters[1:4],100,rep=T))
table(v2)                             #Tableau d'effectifs
barplot(table(v2))                    #Diagramme en barre de v2
```

```
pie(table(v2))                    #Diagramme circulaire de v2
boxplot (v1 ~ v2)                 #Diagramme en boîte de v1 pour chaque modalité
                                  de v2
```

Quelques fonctions d'analyse numérique.

```
f=function(x){                    #Définition de la fonction à étudier
 return(sin(x)/x)
}
plot(f,-20,20)
integrate(f,0,5)                  #Calcul approché de l'intégrale
optim(.1,f)                       #Recherche d'un minimum local autour de x = .1
res=optim(.1,f)                   #res est une liste
points(res$par,res$value)         #Représentation graphique
res=optim(10,f)                   #Recherche d'un minimum local autour de x = 10
points(res$par,res$value)         #Représentation graphique
uniroot(f,lower=.1,upper=6)       #Recherche d'une solution de l'équation f(x) = 0
                                  sur l'intervalle x ∈ [.1, 6]
```

Exercice 7.1 *On considère dans cet exercice à nouveau le jeu de données "iris".*

1. *Réaliser un histogramme de la taille des sépales.*
2. *Proposer un graphique adapté permettant de représenter la distribution de la taille des sépales pour les différentes espèces d'iris.*

Exercice 7.2 *On reprend dans cet exercice les données de l'exercice 3.11. Représenter les distributions du nombre d'étudiants par discipline puis par académie à l'aide de diagrammes circulaires.*

Exercice 7.3 *(extrait de l'examen 2009-2010). Soit f la fonction définie par $f(x) = log(x(1-x)) * sin(2\pi x)$ pour $x \in [0, 1]$.*

1. *Tracer cette fonction à l'aide de R. On donnera les commandes utilisées et on reproduira rapidement le graphique obtenu.*
2. *Cette fonction possède t'elle des extrema locaux ? Si oui, utiliser R pour en donner la valeur numérique. On donnera les commandes utilisées.*
3. *Utiliser R pour calculer $\int_0^1 f(x)dx$. On donnera les commandes utilisées.*

8 Les packages R

Un `package` (ou bibliothèque ou extension) est un ensemble de programmes qui complète les fonctionnalités de R. Il existe un très grand nombre de packages qui sont diffusées par un réseau appelé CRAN (Comprehensive R Archive Network).
La liste de toutes les extensions disponibles sur CRAN est disponible ici :
http ://cran.r-project.org/web/packages/.
L'installation d'un package se fait en deux étapes.

— Il faut d'abord aller chercher le package sur internet et l'installer en local sur le disque dur de l'dinateur. Pour cela, on peut utiliser la commande `install.packages` ou l'outil "'Install Packages"' du menu "'Tools"'. On n'a besoin d'effectuer cette opération qu'une seule fois.

— Il faut ensuite charger le package dans la session R. Pour cela, on peut utiliser la commande `library` ou le gestionnaire de package de `RStudio` (onglet dans la fenêtre en bas à droite par défaut). On a besoin d'effectuer cette opération à chaque début de session ou de script.

Le package MASS permet en particulier d'ajuster des lois.

```
library(MASS)                          #Il faut d'abord téléchager le package MASS
x=rnorm(n=1000,mean=3,sd=5)            #Simulation d'un échantillon gaussien
hist(x,freq=FALSE)                     #Histogramme normalisé
tab=seq(min(x),max(x),by=.1)
lines(tab,dnorm(tab,mean=3,sd=5)       #Tracé de la densité
,col='red')
fit=fitdistr(x,"normal")               #Ajustement d'une loi normale (estimation de μ et
                                       σ)
confint(fit)                           #Intervalle de confiance
m=fit$estimate[1]                      #Estimation de μ
s=fit$estimate[2]                      #Estimation de σ
lines(tab,dnorm(tab,mean=m,sd=s)       #Tracé de la densité de la loi ajustée
,col='cyan')
```

Le package ggplot2 permet de faire des graphiques sophistiqués tels que les 'violin plot'.

```
library(ggplot2)                       #Il faut d'abord télécharger le package ggplot2
p=ggplot(iris,aes(Species,Sepal.Length))
p + geom_violin()
```

Le package leaflet permet de faire des cartes.

```
library(leaflet)
lon=-4.5015
lat=48.40095
leaflet() %>%
addTiles() %>%
setView(lng = lon, lat = lat, zoom = 18)%>%
addMarkers(lng=lon, lat=lat,label ='EURIA',labelOptions =
labelOptions(noHide = T, textsize = "15px"))
```

Exercice 8.1 *Après avoir installé le package* `MASS`*...*

1. *Simuler un échantillon de taille $n = 100$ d'une loi Exponentielle de paramètres $\lambda = 2$ avec la fonction* `rexp`*. Faire un histogramme de l'échantillon simulé.*
2. *On suppose maintenant que λ est inconnu. Utiliser la fonction* `fitdistr` *pur estimer λ et superposer la densité correspondante sur l'histogramme de la question précédente.*

Exercice 8.2 *Après avoir installé le package* `leaflet`*, représenter sur une même carte votre lieu de naissance et l'EURIA.*

9 Les entrées et sorties

Dans de nombreux cas, les données que l'on souhaite analyser proviennent de sources externes sous forme de fichiers. Aussi, les objets créés doivent pouvoir être sauvegardés dans des fichiers afin qu'ils soient complètement transportables. C'est ainsi que divers outils d'accès aux fichiers ont été développés sous R. On distingue les accès aux fichiers propriétaires de R, aux fichiers

ASCII, aux fichiers provenant d'autres logiciels d'analyse statistique, aux bases de données relationnelles et aux fichiers binaires.

Formats propriétaire La fonction générique `save()` autorise la sauvegarde de n'importe quelle liste d'objets en mémoire, sous un chemin quelconque, aussi bien en format binaire qu'ASCII. Ces objets peuvent ensuite être rechargés en mémoire grâce à la fonction `load()` qui est le pendant de la première. Par ailleurs, pour des raisons de compatibilité avec le langage S, il existe la fonction `dump()` qui permet d'exporter des objets vers un fichier texte pouvant alors être relu et interprété séquentiellement par la fonction `source()`. Cette dernière fonction est très utilisée en programmation car elle autorise la lecture d'une série d'instructions sauvegardées dans un fichier texte à l'aide d'un éditeur externe.

Fichiers textes ASCII Le format d'échange le plus courant en ce qui concerne les données brutes reste le fichier ASCII. La lecture de tels fichiers est prise en charge par la commande élémentaire `scan()`. Les arguments de cette fonction permettent de décrire précisément la structure du fichier texte. Afin de faciliter la lecture des fichiers de données structurées en colonnes plusieurs commandes spécifiques ont été développées à partir de la fonction `scan()`. Ces fonctions (`read.table()` et ses dérivés) automatisent la lecture des fichiers de données ASCII standards (csv, texte délimité, largeur fixe ...) et stockent leurs résultats dans des data.frame. Bien qu'elles soient plus spécifiques que `scan()`, ces fonctions conservent une grande adaptativité grâce à l'utilisation de nombreux arguments permettant de préciser le format interne du fichier (présence de titre de colonnes, type de séparateur ...). Le menu "Tools>Import data" de la fenêtre `RStudio` ouvre une boîte de dialogue qui permet de spécifier ces différents arguments. L'exportation de tableaux de données sous forme de fichiers ASCII standards peut être réalisée par la fonction `write.table()`.

Logiciels statistiques Lorsque les données ont été sauvegardées sous le format propriétaire d'un logiciel statistique tiers, il est nécessaire de disposer d'outils permettant leur transfert vers le système *R*. La librairie foreign offre ces outils pour une sélection des logiciels statistiques les plus courants, à savoir Minitab, S-Plus, SAS, SPSS et Stata. Par exemple, la fonction read.spss prend en charge les données enregistrées au moyen des commandes save et export de SPSS. Il existe aussi des packages pour importer des données de tableur (.xls par exemple). Une alternative classique est de générer un fichier texte (par exemple .csv) à partir du logiciel puis d'importer ce fichier sous `R` avec la commande `read.table` ou `read.csv`.

Bases de données relationnelles La version de base de l'environnement `R` n'est pas adaptée à la gestion de très grosses quantités de données. En effet, tous les objets sont chargés intégralement en mémoire centrale et plusieurs copies de ces objets peuvent être créées lors de l'exécution des procédures de traitement, ce qui peut entraîner une saturation du système dès que la taille totale des jeux de données dépasse une certaine fraction de l'espace mémoire disponible. De plus, il ne permet pas à plusieurs utilisateurs d'accéder aux mêmes données de manière concurrente, c'est-à-dire en intégrant en temps réel les modifications des uns et des autres. Ce travail de gestion est d'dinaire le domaine de prédilection des systèmes de gestion de bases de données (SGBD), et plus particulièrement des SGBD relationnels. Une série de modules faisant office d'interface entre ces SGBD et l'environnement `R` a été développé. On peut ainsi trouver diverses librairies de fonctions dédiées au pilotage des SGBD.

Fichiers binaires `R` fournit des outils permettant l'accès aux fichiers binaires tels que les fichiers images ou sons, par l'intermédiaire de connexions, objets servant de lien entre le système et les fichiers externes. Une fois créée par la commande générique `open()`, une connexion permet un accès transparent au fichier qu'elle représente. Dans le cas d'une connexion en mode binaire, les fonctions `readBin()` et `writeBin()` se chargent des opérations de lecture et d'écriture dans le

fichier ainsi ouvert.

Exercice 9.1 *1. Télécharger le jeu de données disponible à l'adresse http ://pagesperso.univ-brest.fr/∼ailliot/L3EURIA.html*

2. *Ouvrir les données à l'aide du tableur (Excel ou équivalent) puis importer les données sous* `R`*. On commencera par exporter les données au format .csv à partir du tableur avant de les importer sous* `R` *à l'aide de l'outil* `Import data` *de* `Rstudio` *(disponible dans le menu* `Tools`*).*
3. *Etablir la répartition du personnel par site. Représenter cette répartition avec un graphique adapté.*
4. *Quel est le pourcentage de femmes dans le jeu de données ?*
5. *Calcul le salaire médian, le salaire moyen, la salaire maximal.*
6. *Réaliser un histogramme de la variable salaire.*
7. *Créer un jeu de données qui contient seulement les individus ayant un salaire parmi les 10% les plus élevés. Quel est le pourcentage de femmes dans ce jeu de données ?*

10 Gestion des objets créés

Lors de l'exécution de `R`, les fichiers .RData et .Rhistory sont automatiquement créés dans le répertoire de travail. Aussi, lorsque l'on quitte `R` à l'aide de la commande `q()`, les nouveaux objets créés peuvent être sauvegardés dans le fichier .RData (le choix est donné à l'utilisateur) en mode binaire et la suite des commandes que l'on a tapée est automatiquement sauvegardée dans le fichier .Rhistory en mode texte. Ainsi, lorsque l'on relance `R` dans ce même répertoire, le fichier .RData est automatiquement chargé et l'on retrouve l'intégralité des objets que l'on y a créés. Par ailleurs, on peut visualiser la suite de commandes que l'on a tapée à l'aide de l'instruction `history()` dans `R` ou en éditant simplement le fichier .Rhistory. Pour connaître le répertoire de travail courant au cours d'une session, il suffit de taper `getwd()` et pour en changer, il faut utiliser la commande `setwd()`.
Enfin, la commande `ls()` permet de visualiser la liste des objets créés et la commande `rm()` permet de détruire des objets.

11 Exercices

Exercice 11.1 *(extrait de l'examen 2011-2012)*

1. *Que permettent de calculer les fonctions* `floor` *et* `log10` *?*
2. *En utilisant ces deux fonctions, écrire une fonction* `nbdec=function(n)` *qui calcule le nombre de décimales de l'entier naturel n. Par exemple, si $n = 1295372$, la fonction renverra la valeur 7. On donnera les commandes* `R` *utilisées.*
3. *Ecrire une fonction* `inverseordre=function(n)` *qui renvoie*
   — *Un message d'erreur si n n'est pas un entier naturel*
   — *L'entier naturel obtenu en écrivant les décimales de n dans l'dre inverse si n est un entier naturel. Par exemple, si $n = 1295372$, la fonction renverra la valeur 2735921.*

Exercice 11.2 *(extrait de l'examen 2012-2013)*

1. *Ecrire une fonction qui extrait les composantes positives dans un vecteur. Par exemple, pour le vecteur $(1, -3, 6, -1, 2)$, la fonction rendra le vecteur $(1, 6, 2)$. On écrira deux versions : une avec un boucle* `for` *et une sans boucle.*

2. *Ecrire une fonction qui recherche la position du premier élément négatif dans un vecteur. Par exemple, pour le vecteur* $(1, 3, 6, -1, 2)$*, la fonction rendra la valeur* 4*. On écrira trois versions : une avec un boucle* `for`*, une avec une boucle* `while` *et une sans boucle.*

Exercice 11.3 *(extrait de l'examen 2013-2014)*
Ecrire une fonction qui calcule le nombre de termes strictement croissants dans un vecteur $x = (x_1, ..., x_n)$*, c'est à dire* $card\{i \in \{1, ...n-1\} | x_{i+1} > x_i\}$*. Par exemple, pour* $x = (3, 2, 1, 4)$*, la fonction renverra la valeur* 1*. On écrira une version avec une boucle for et une version sans boucle.*

Exercice 11.4 *(extrait de l'examen 2013-2014)*
Ecrire une fonction qui calcule l'indice du premier terme strictement croissant dans un vecteur $x = (x_1, ..., x_n)$*, c'est à dire* $min\{i \in \{1, ...n-1\} | x_{i+1} > x_i\}$*. La fonction renverra le message 'La suite est décroissante' si* $x_{i+1} \leq x_i$ *pour tout* $i \in \{1, ..., n\}$*. Par exemple, pour* $x = (3, 2, 1, 4)$*, la fonction renverra la valeur* 3*. On écrira une version avec une boucle while et une version sans boucle.*

Exercice 11.5 *(extrait de l'examen 2013-2014)*
Un ami vous propose le jeu suivant. On lance un dé. Si le résultat est 5 ou 6, on gagne 3 euros, si le résultat est 4 on gagne 1 euro et si c'est 3 ou moins on perd 2.5 euros. Avant d'accepter la partie, vous essayez de simuler ce jeu, pour voir si vous avez des chances de vous enrichir. Conclusion ?

Exercice 11.6
1. *Écrire une fonction nommée 'fonc1' qui permet de calculer la fonction* $x \to xln(x)$ *puis trouver le minimum de cette fonction à l'aide de la fonction* `optim`*.*
2. *Écrire une fonction nommée 'fonc2' qui permet de calculer la fonction* $x \to xtan(x) - 1$ *puis trouver les zéros de cette fonction à l'aide de la fonction* `uniroot`*.*

Exercice 11.7 *Ecrire une fonction* `prodmat` *qui calcule le produit matriciel de deux matrices A et B. Si les dimensions des matrices ne sont pas compatibles, la fonction affichera un message d'erreur.*

Exercice 11.8 *Ecrire une fonction* `resol_systeme=function(A,b)` *qui*
1. *Renvoie un message d'erreur si la matrice A n'est pas inversible ou si les dimensions de A et b ne sont pas compatibles.*
2. *Renvoie la solution x du système d'équation linéaire* $Ax = b$ *sinon. On pourra dans un premier temps utiliser la fonction* `solve` *de* `R` *puis on programmera la méthode du pivot de Gauss.*

Exercice 11.9
1. *Simuler* $n = 50$ *réalisations* $(x_1, ..., x_n)$ *indépendantes d'une loi normale de moyenne* 0 *et variance* 1 *(loi* $\mathcal{N}(0, 1)$*) à l'aide de la fonction* `rnorm`*.*
2. *Calculer la moyenne, la variance, l'écart type et la médiane de l'échantillon simulé. Discuter.*
3. *Faire un histogramme de l'échantillon simulé. Ajouter la densité de la loi* $\mathcal{N}(0, 1)$ *sur le graphique (on pourra utiliser la fonction* `pnorm`*). Discuter.*
4. *Simuler une autre échantillon de taille* 50 *de la loi* $\mathcal{N}(0, 1)$*, et recommencer les questions 2. et 3. Discuter.*
5. *Simuler une autre échantillon de taille* 1000 *de la loi* $\mathcal{N}(0, 1)$*, et recommencer les questions 2. et 3. Discuter.*

Exercice 11.10 *Ecrire une fonction* `trapeze` *qui calcule une valeur approchée de* $\int_a^b f(x)dx$*, avec* $f : [a, b] \to \mathbb{R}$ *par la méthode des trapèzes. Comparer les résultats avec ceux obtenus avec la fonction* `integrate` *sur quelques fonctions f simples.*

Exercice 11.11 *(extrait de l'examen 2010-2011)*
On considère la suite $(U_n)_{n\in\mathbb{N}^*}$ définie par $U_1 = 1$ et la relation de récurrence

$$U_{n+1} = cos(U_n)$$

pour $n \geq 1$.

1. *Écrire une fonction `R` qui permet de calculer les N premiers termes de la suite U_n. En déduire les valeurs de U_{10} et $\sum_{i=1}^{100} U_i$. $\sum_{i=1}^{100} U_i^2$.*
2. *Représenter graphiquement les 20 premiers termes de la suite U_n en utilisant `R` et reproduire le graphique schématiquement sur votre copie. La suite U_n est-elle monotone ?*
3. *Représenter graphiquement les 100 premiers termes de la suite U_n en utilisant `R` et reproduire le graphique schématiquement sur votre copie. D'après ce graphique, pensez-vous que la suite U_n est convergente (on ne demande pas de démonstration mathématique !) ? Si oui, proposer une valeur numérique approchée pour la limite de la suite.*
4. *Tracer sur un même graphique la fonction cos en rouge et la droite d'équation $y = x$ en noir en utilisant `R`. D'après ce graphique, combien de solutions réelles a l'équation $cos(x) = x$ (on ne demande pas de démonstration mathématique !) ?*

Exercice 11.12 *(extrait de l'examen 2012-2013)*

1. *Cette question est à réaliser en utilisant `R`. On donnera toutes les commandes `R` utilisées. On considère la suite $(u_n)_{n\geq 1}$ définie par $u_1 = 1$, $u_2 = 2$ et la relation de récurrence*

   $$u_n = \sqrt{u_{n-1}u_{n-2}}$$

   pour $n \geq 3$
   Ecrire une fonction qui calcule les n premiers termes de cette suite puis représenter graphiquement les 30 premiers de la suite (on reproduira schématiquement le graphique sur la copie). Donner la valeur de $\sum_{n=1}^{30} u_n$.
2. *Reprendre la question précédente avec le tableur de votre choix. On décrira précisément les différentes étapes sur la copie.*

Exercice 11.13 *(extrait de l'examen 2017-2018)*

1. Tracer sur un même graphique les fonctions $f(x) = ln(x^2 + 1)$ et $g(x) = cos(x)$ définies sur l'intervalle $[0, 1]$ en utilisant deux couleurs différentes. Reproduire le graphique sur la copie.
2. Donner les valeurs de $x \in [0, 1]$ telles que $f(x) = g(x)$.
3. Donner une valeur approchée de $\int_0^1 f(x)dx$.

Exercice 11.14 *(extrait de l'examen 2017-2018)*

On considère dans cet exercice le jeu de données `iris`.

1. Créer un jeu de données `iris2` qui contient uniquement les lignes de `iris` telles que
   — la variable `Species` prend la valeur `setosa` ou `virginica` ;
   — et la variable `Sepal.Length` prend une valeur inférieure à 6.
   On répondra à cette question de deux manières différentes
   — en utilisant une boucle ;
   — sans utiliser une boucle.
2. Quelle est le nombre de lignes de l'objet `iris2` (le nombre de colonnes doit être identique au nombre de colonnes de `iris`) ? Trier les individus de `iris2` par ordre croissant de la variable `Sepal.Width`.

Exercice 11.15 *(extrait de l'examen 2017-2018)*

On considère la suite de Syracuse définie pour $n \geq 1$ par $a_1 = a$ avec $a \in \mathbb{N}^*$ et

$$\begin{cases} a_{n+1} = a_n/2 \text{ si } a_n \text{ est pair} \\ a_{n+1} = 3a_n + 1 \text{ si } a_n \text{ est impair} \end{cases}$$

On vérifie facilement que si $a = 1$ alors la suite de Syracuse est cyclique et prend les valeurs $(1, 4, 2, 1, 4, 2, 1, ...)$. La conjecture de Syracuse est l'hypothèse selon laquelle la suite de Syracuse associée à n'importe quelle valeur initiale a atteint la valeur 1 à partir d'un certain rang. Après avoir atteint cette valeur, la suite prend alors les valeurs cycliques $(..., 1, 4, 2, 1, 4, 2, 1, ...)$. Par exemple pour $a = 10$, on obtient la suite $(10, 5, 16, 8, 4, 2, 1, 4, 2, 1, ...)$. On propose de tester cette conjecture en utilisant R.

1. Ecrire une fonction R nommée `Syracuse` qui calcule les n premiers de la suite de Syracuse avec n le plus petit entier tel que $a_n = 1$. Par exemple, la commande `Syracuse(10)` renverra les valeurs $10, 5, 16, 8, 4, 2, 1$. On donnera le code de la fonction sur la copie. On écrira cette fonction de telle manière à éviter les calculs inutiles.
2. Quelle est la suite de Syracuse pour $a = 30$?
3. Quel est plus petit entier n tel que $a_n = 1$ pour $a = 1000$?
4. Ecrire un code qui vérifie si l'hypothèse de Syracuse est valide pour $a \in \{2, ..., 10^5\}$.

Exercice 11.16 *(extrait de l'examen 2018-2019)*

On veut trouver le maximum d'un vecteur V, le nombre d'occurrences du maximum ainsi que les positions du maximum. Par exemple, si $V = (1, 4, 3, 2, 4)$ alors
- le maximum de V est 4 ;
- le nombre d'occurrences du maximum est 2 (nombre de fois où 4 apparaît dans le vecteur V) ;
- les positions du maximum sont (2,5).

1. Ecrire une boucle `for` qui détermine le maximum d'un vecteur V ainsi que le nombre d'occurrences et les positions du maximum (dans cette question, on n'utilisera pas la fonction max et les fonctions associées et on utilisera une seule boucle `for`).
2. Reprendre la question précédente sans boucle `for`.
3. Quelle méthode vous semble la plus efficace dans R ? On implémentera cette méthode dans une fonction nommée `maxperso` qui prend en entrée un vecteur V et renvoie en sortie le maximum de V, le nombre d'occurrences ainsi que les positions du maximum.

Exercice 11.17 *(extrait de l'examen 2018-2019)*

On considère dans cet exercice le jeu de données `Seatbelts` disponible dans `R`.

1. Transformer l'objet `Seatbelts` en un objet nommé z de type `data.frame`.
2. Réaliser un histogramme de la variable `DriversKilled`. On reproduira rapidement le graphique sur la copie.
3. Proposer un graphique adapté permettant de représenter la distribution de la variable `DriversKilled` pour les différentes valeurs de la variable `law`. On reproduira rapidement le graphique sur la copie.
4. Créer un jeu de données `z2` qui contient uniquement les lignes de `z` telles que
   - la variable `law` prend la valeur `0` ;
   - et la variable `DriversKilled` prend une valeur inférieure à 100 ou une valeur supérieure à 130.

On répondra à cette question sans utiliser une boucle.

5. Quelle est le nombre de lignes de l'objet `z2` (le nombre de colonnes doit être identique au nombre de colonnes de `z`) ? Quelle est la moyenne des différentes colonnes de l'objet `z2` (on donnera les valeurs numériques sur la copie) ? Trier les individus de `z2` par ordre croissant de la variable `DriversKilled`.