Licence de Mathématiques – Université de Lomé

MTH229
Introduction au logiciel 

Issa Cherif GERALDO

Janvier 2023

TABLE DES MATIÈRES

CHAPITRE 0

NOTE INTRODUCTIVE

0.1 BREF DESCRIPTIF DU COURS

Ce cours présente le logiciel statistique R et son utilisation pour mettre en œuvre les méthodes et techniques usuelles en statistique descriptive et en calcul de probabilités. Il peut être utile à toute personne ayant suivi les cours de statistique descriptive et de calcul de probabilités (niveau première année de Licence) et souhaitant appliquer ces cours à l'aide d'un logiciel statistique.

0.2 OBJECTIF GÉNÉRAL

L'objectif général de ce cours est d'appliquer les méthodes usuelles de la statistique descriptive et du calcul de probabilités avec le logiciel statistique R.

A la fin de ce cours, l'apprenant devra être capable de :

- saisir des données dans le logiciel R (chapitres 1 et 2)
- Appliquer les méthodes de la statistique descriptive à l'aide du logiciel R (chapitre 3)
- Programmer en R (chapitre 4)
- faire des représentations graphiques (chapitre 5)
- Implémenter quelques scénarios probabilistes classiques (calcul de probabilités pour les lois usuelles, lancer de dés, simulations simples, etc. . .) (chapitre 6).

0.3 STRUCTURE DU DOCUMENT

Le présent document est subdivisé en six (06) chapitres contenant chacun des syntaxes et des exemples détaillés. Chaque chapitre finit par des exercices non corrigés.

Il est vivement conseillé au lecteur d'installer le logiciel R et de tester les syntaxes et les exemples. Le lecteur est aussi exhorté à consulter la documentation de R pour une compréhension plus détaillée de certaines commandes.

CHAPITRE 1

GÉNÉRALITÉS SUR LE LOGICIEL R

Sommaire

1.1 PRÉSENTATION

Le logiciel R est un logiciel d'analyse statistique et graphique créé par deux statisticiens Ross Ihaka (statisticien néo-zélandais) et Robert Gentleman (statisticien canadien) à l'université d'Auckland (Nouvelle-Zélande) dans le cadre d'un projet de recherche débuté en 1993.

Il est un clone gratuit du logiciel payant S-plus, dérivé du langage S (Logiciel payant) développé dans les années 1970 par une équipe de chercheurs menée par un statisticien du nom de John M. Chambers. Depuis 1997, son développement et sa distribution sont assurés par plusieurs statisticiens rassemblés dans le **R Development Core Team**.

Le logiciel R présente plusieurs avantages : c'est un logiciel **libre** et **gratuit**, très complet, en essor permanent, disponible pour divers environnements : Linux, Windows, Macintosh. C'est un logiciel utilisé en entreprise, dans le monde académique et de la recherche, au sein d'organismes publics ou d'ONG ou encore par les analystes (data miners, data scientists)

spécialistes de la fouille de données.

Le site officiel du logiciel R est :

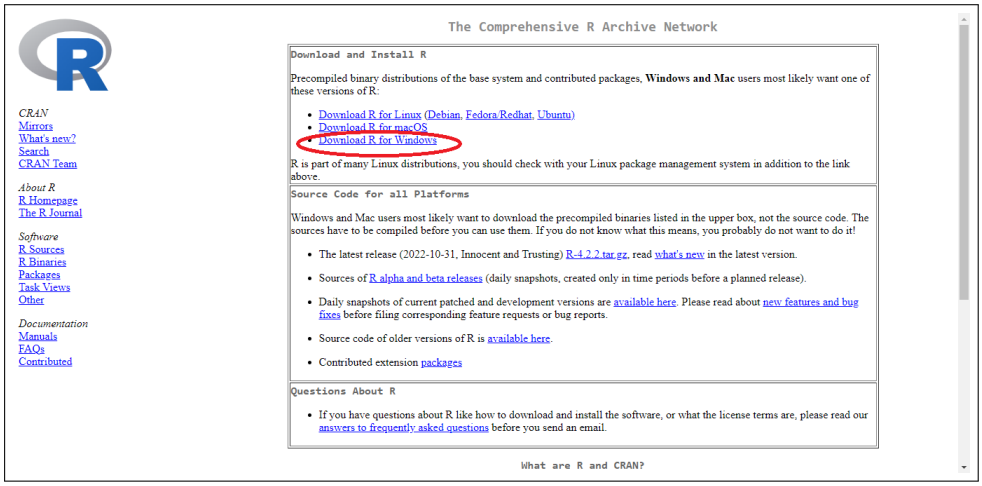
https://www.r-project.org/

1.2 INSTALLATION

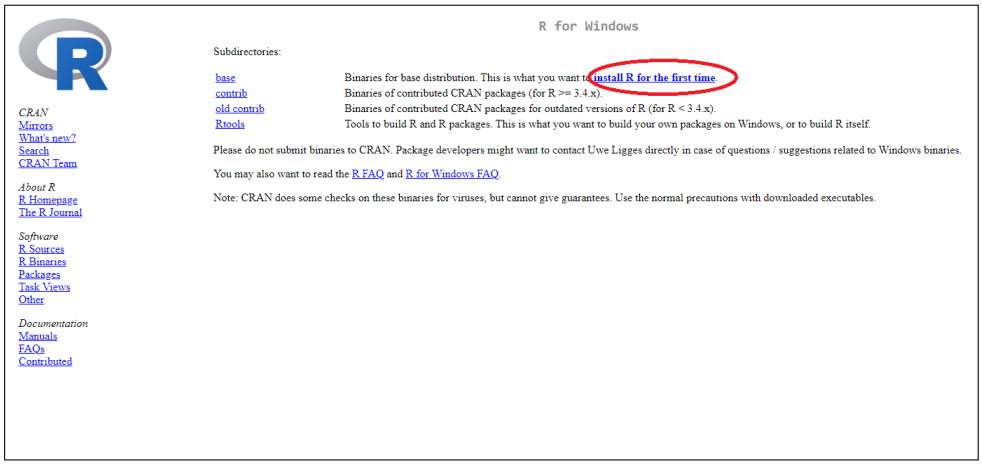
1.2.1 Installation sous Windows

Pour télécharger le fichier d'installation, il faut :

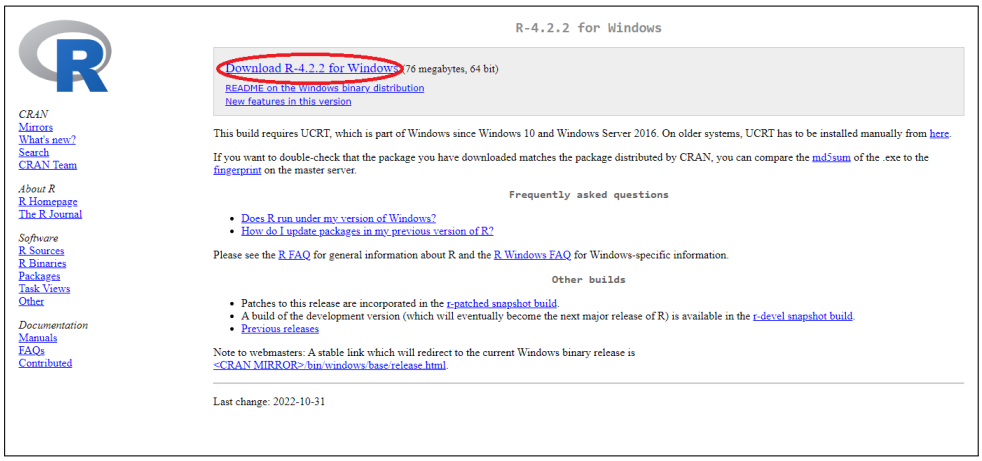
- se rendre sur le site du CRAN (Comprehensive R Archive Network) sur lequel se trouvent les ressources pour le logiciel R : https://cran.r-project.org/)
- cliquer sur le lien **Download R for Windows** :



- cliquer sur **install R for the first time** :



- cliquer sur **Download R-x.x.x for Windows**.



Après téléchargement du fichier d'installation, il faut l'exécuter par double-clic. **Durant l'installation, il ne faut pas oublier de cocher l'option de mettre un raccourci sur le bureau**. Une fois l'installation terminée, le logiciel `R` est accessible (comme tout autre logiciel) dans le menu *Démarrer* ou dans *Tous les programmes* ou encore sur le bureau. Sur le bureau, il est repérable par son icône :

1.2.2 Installation sous Linux

L'installation peut se faire via le *gestionnaire des paquets* en installant les paquets **`r-base-dev`** et **`r-base`**.

1.3 Lancement et fermeture de R

1.3.1 Lancement de R sous Windows

Pour lancer `R` **sous Windows**, il suffit de double-cliquer sur l'icône

de RGUI (R Graphical User Interface). Il s'ouvre alors une interface graphique contenant une fenêtre de la forme de la figure 1.1.

```
>
```

Le symbole > appelé *prompt* indique que `R` est prêt à travailler et attend une commande.

Remarque 1.1 Lorsque le symbole + apparaît au lieu de >, cela signifie que la commande en cours de saisie est incomplète (par exemple, il manque une parenthèse ou une accolade) et que `R` attend qu'elle soit complétée. On peut alors soit compléter la commande soit appuyer sur la touche *Esc* pour annuler la commande et revenir à >.

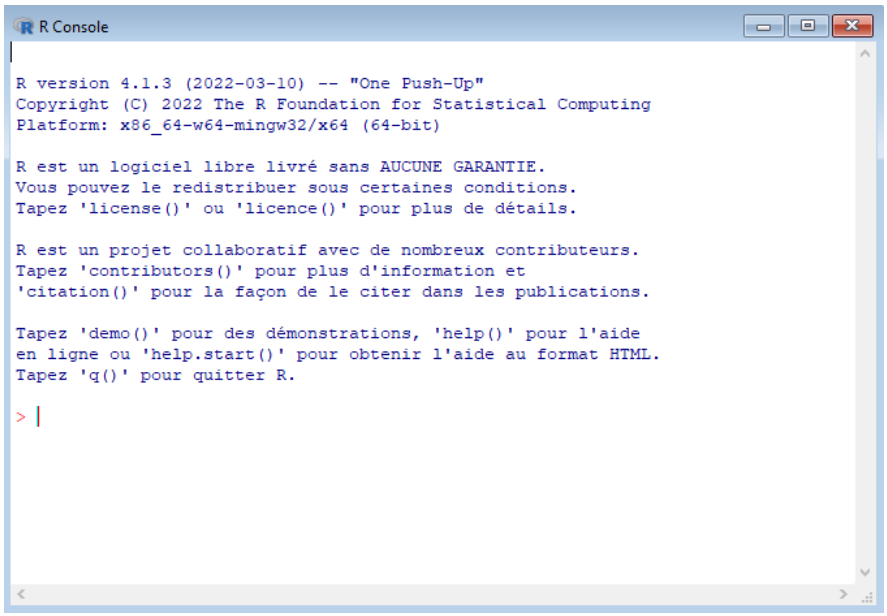


FIGURE 1.1 – Ecran de démarrage de R. Le symbole >, appelé prompt, signifie que R attend une commande.

Remarque 1.2 On peut vider la console à tout moment en appuyant simultanément sur les touches `Ctrl` et `L`.

1.3.2 Lancement de R sous Linux

Sous Linux, l'on peut lancer R depuis un terminal Linux en tapant simplement la commande R dans le terminal (voir Figure).

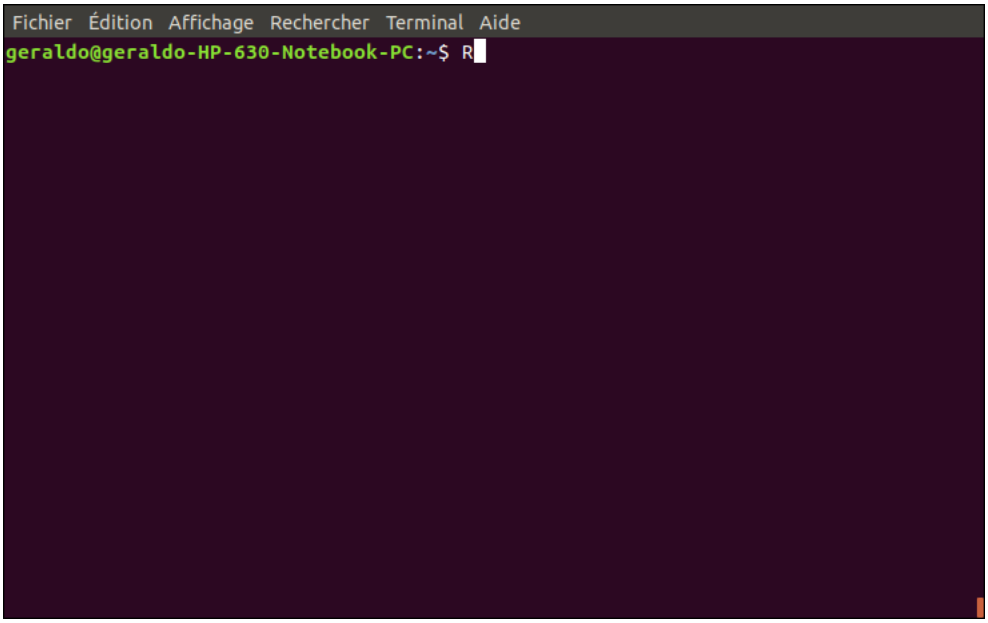


FIGURE 1.2 – Terminal Linux avant lancement de R

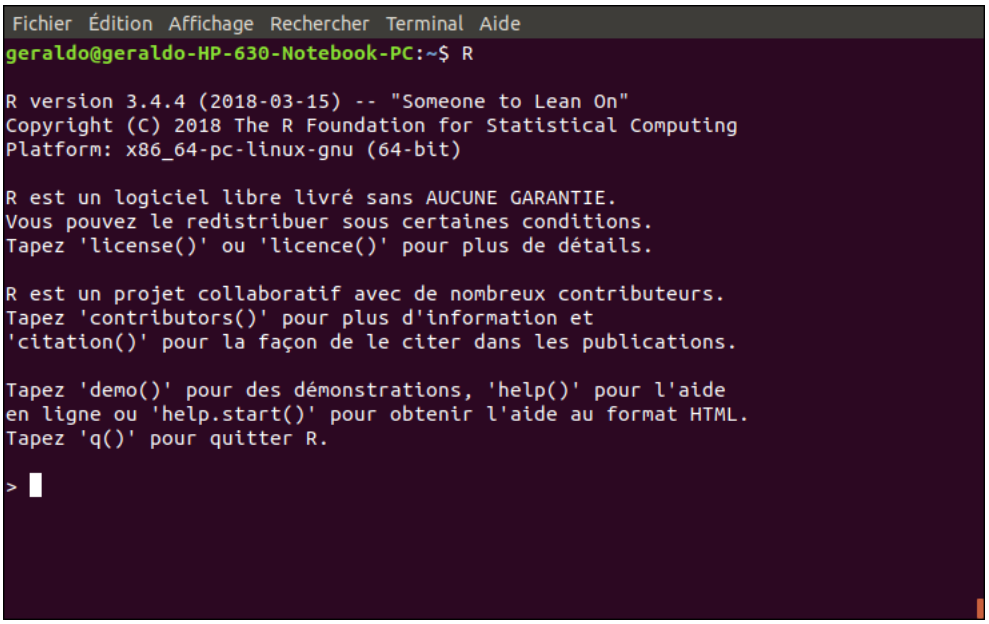


FIGURE 1.3 – Terminal Linux après lancement de R

1.3.3 Fermeture de R

```
quit()
q()
```

Commandes équivalentes à saisir dans le terminal pour fermer R. Il ne faut pas oublier les parenthèses.

Quand on exécute l'une de ces deux commandes, R demande si l'on souhaite sauvegarder la session (c'est-à-dire enregistrer toutes les variables que l'on a créées dans un fichier sur le disque dur). Il faut toujours répondre non.

```
QUIT()
Quit()
```

Ces commandes ne fonctionnent pas car R est sensible à la casse c'est-à-dire R distingue les majusules des minuscules.

1.4 COMMANDES ÉLÉMENTAIRES

1.4.1 Création de variables et affichage

```
x=2
x <- 2
2 -> x
x = 1 + 1
```

Commandes équivalentes permettant de créer (s'il n'existe pas encore) un objet x et d'affecter la valeur 2 à x.

`employe.salaire2 = 100`	Les noms de variables peuvent contenir des lettres, des chiffres, les caractères "_" et "."

Remarque 1.3 Même si les noms de variables sont très flexibles dans `R`, il y a quelques règles importantes à respecter :

1) Les noms de variables ne peuvent pas commencer par un chiffre ou un caractère spécial.
2) Comme dit plus haut, `R` est sensible à la casse donc les noms des variables sont aussi sensibles à la casse (les variables `x` et `X` sont deux variables différentes).
3) Quelques noms (comme `c`, `q`, `t`, `C`, `D`, `F`, `I`, `pi`, `T`, etc...) sont déjà utilisés par `R` pour désigner des objets internes et doivent être évités.

`x` `print(x)`	Affiche la valeur de l'objet `x`.
`print("Bonjour")`	Affichage de texte.

1.4.2 Séparation de commandes

Pour écrire plusieurs commandes, on peut :

- soit les aligner sur une même ligne en les séparant obligatoirement par un point-virgule
- soit passer à une nouvelle ligne pour chaque commande (dans ce cas le point-virgule n'est pas obligatoire à la fin d'une commande).

`x=2; y=3; z=-1`	Commandes sur la même ligne séparées par un ";"

`x=2`
`y=3`
`z=-1`
	Commandes séparées par un retour à la ligne.

1.4.3 Espace de travail

`objects()` `ls()`	Deux commandes équivalentes permettant d'obtenir la liste des objets (variables et fonctions) présents en mémoire.
`rm(a,b)`	Supprime les objets `a` et `b` de la mémoire.
`rm(list=ls())`	Supprime tous les objets en mémoire.
`history()`	Historique des commandes entrées dans le terminal dans la session courante.

`getwd()`	Afficher le répertoire courant (le répertoire dans lequel `R` écrit ou lit les fichiers par défaut).
`setwd("C:/Users/Toto/")`	Changer le répertoire courant Remarquer l'utilisation de "/" et non de "\".

1.4.4 Documentation de `R`

`help(solve)` `?solve`	Deux commandes équivalentes pour afficher l'aide HTML sur la fonction `solve`.
`help.search("bin")` `??bin`	Affiche la liste de toutes les pages de la documentation de `R` contenant le mot "bin".

1.4.5 Commentaires

`x=10 # Création de x`	Le symbole `#` est utilisé pour faire les commentaires **sur une seule ligne**. Sur une ligne donnée, `R` n'exécute pas le texte ou les commandes rencontrées après le symbole `#`.

1.5 Méthodes d'exécution des commandes

1.5.1 Première méthode

On peut entrer les commandes les unes après les autres dans la console de `R` et appuyer sur la touche **Enter** ↩ pour les exécuter.

1.5.2 Deuxième méthode

On peut aussi créer un fichier de commandes (fichier d'extension `.R`) en utilisant l'éditeur de `R` (ou un autre éditeur). Sous Windows, l'éditeur de `R` est disponible en cliquant sur `Fichier` puis sur `Nouveau script`. Après enregistrement sur le disque dur, on peut l'exécuter dans la console à l'aide de la commande **`source`** comme suit :

`source("somme.R")`	Exécute les commandes contenues dans le fichier `somme.R`

Par défaut, `R` cherchera le fichier `somme.R` dans le répertoire courant (résultat de la commande `getwd()`). Si le fichier à exécuter se trouve dans un autre répertoire, il faudra donc au

FIGURE 1.4 – Exemple d'exécution de commandes dans la console de R.

préalable utiliser la commande `setwd()` pour indiquer à R le répertoire dans lequel se trouve ledit fichier.

1.6 BIBLIOTHÈQUES OU PACKAGES

Une bibliothèque (ou package) est un ensemble de fonctions et de données permettant de faire des tâches bien précises. Par exemple la bibliothèque `stats` contient un ensemble de fonctionnalités permettant de faire des statistiques. Il existe deux types de bibliothèques :

- les bibliothèques de base qui sont déjà présentes dans l'installation de base de R : seules certaines d'entre elles sont chargées automatiquement au démarrage de R (exemple des bibliothèques `base` et `stats`).
- d'autres bibliothèques écrites par d'autres utilisateurs de R disponibles sur le site du CRAN (Comprehensive R Archive Network) : https://cran.r-project.org/web/packages/index.html.

A la date de la dernière mise à jour de ce cours (janvier 2023), le site du CRAN compte un peu plus de 19 000 packages.

`library()`	Liste toutes les bibliothèques disponibles localement sur la machine où R est exécuté.
`library(stats)`	Chargement d'une bibliothèque. Cette commande est obligatoire pour utiliser les fonctionnalités (fonctions, données) d'une bibliothèque (ici stats).
`help(package=stats)`	Documentation du package `stats`.
`search()`	Liste des bibliothèques chargées.

`install.package(nom)`	Télécharge la dernière version de la bibliothèque `nom` et l'installe.

A l'exécution de la commande d'installation d'un package, `R` peut demander de spécifier un miroir. Les miroirs sont des serveurs situés à plusieurs endroits dans le monde et disposant des mêmes fichiers que ceux téléchargeables sur le site du CRAN.

Remarque 1.4 Dans ce cours, nous n'aurons besoin de télécharger aucun package. La version basique de `R` contient tout ce nous avons besoin.

CHAPITRE 2

CLASSES D'OBJETS ET FONCTIONS USUELLES

Sommaire

2.1 INTRODUCTION

Tous les objets `R` (variables, fonctions) ont une classe et les manipulations possibles sur un objet donné dépendent de sa classe. Les classes d'objets usuelles sont :

`numeric`, `matrix`, `character`, `factor`, `list`, `data.frame`, `logical`.

`x=2; class(x)`	Affichage de la classe de l'objet `x`, ici `"numeric"`.

2.2 VECTEURS NUMÉRIQUES (CLASSE `numeric`)

2.2.1 Valeurs particulières

`NA`	Valeur manquante (Not Available).
`Inf ; -Inf`	Respectivement $+\infty$ et $-\infty$.
`pi`	Variable interne contenant la valeur de π.
`1/0 ; -1/0`	Respectivement `Inf` et `-Inf`.
`0/0 ; Inf-Inf`	`NaN` (*Not a Number*).

2.2.2 Nombres complexes

`sqrt(-1)`	Affiche `NaN`.
`sqrt(-1+0i)`	Affiche `0+1i`.

2.2.3 Création

Il existe plusieurs méthodes pour créer un vecteur numérique. Dans la pratique, il revient à l'utilisateur de choisir la méthode la mieux adaptée à ce qu'il souhaite faire.

`x = c(1,3,4,-5)`	Création du vecteur $(1,3,4,-5)$ affecté à `x`. La lettre `c` peut être comprise comme *collect* ou *combine*.
`print(x)`	Affichage de `x`
`1:10`	Vecteur $(1,2,\ldots,9,10)$.
`seq(from=1,to=10,by=2)`	Création d'un vecteur dont les composantes forment une suite arithmétique de premier terme 1 (`from=1`), dont la raison (encore appelée incrément ou pas) vaut 2 (`by=2`). L'argument `to=10` indique que la composante maximale ne doit pas dépasser 10. Ici le résultat est un vecteur à 5 composantes : 1, 3, 5, 7 et 9.

`x=c(1,2);` `c(x,x,x); rep(x,times=3)`	Commandes équivalentes permettant de créer un nouveau vecteur formé en mettant `x` bout à bout 3 fois. Le résultat est $(1,2,1,2)$.
`rep(x,each=3)`	Création d'un vecteur dans lequel chaque composante de `x` est répétée 3 fois successivement. Le résultat est $(1,1,1,2,2,2)$.
`rep(x,times=c(2,3))`	Création d'un vecteur contenant 2 fois la première composante et 3 fois la seconde. Le résultat est $(1,1,2,2,2)$.

2.2.4 Opérations et fonctions

2.2.4.1 Opérations et fonctions usuelles

Soient `x` et `y` deux vecteurs créés à l'aide du code ci-après :

```
x=c(1,-5,7,0)
y=c(2,3,-4,1)
```

`z=x+y`
`z=x*y`
`z=x/y`
`z=2*x`
	Addition, multiplication et division composante par composante pour deux vecteurs de même nombre de composantes.

`2*x`	Multiplication de toutes les composantes de `x` par 2.
`x+2`	Ajoute 2 à toutes les composantes de `x`.

`abs(x); x^2; x^(1/3); sqrt(x);`
`exp(x); log(x); sin(x); cos(x);`
`tan(x); sinh(x); cosh(x);`
`tanh(x); ...`
	Ces fonctions usuelles s'appliquent à toutes les composantes de `x` c'est-à-dire elles retournent un vecteur de même taille que `x` dont les composantes sont les images respectives de celles de `x`.

`length(x)`	Nombre de composantes de `x`.
`max(x) ; min(x)`	Maximum et minimum des composantes de `x`.
`range(x)`	Vecteur composé du minimum et du maximum des composantes de `x`.
`sum(x) ; mean(x)`	Somme des composantes, moyenne.
`round(x); round(x,2)`	Arrondi à l'entier le plus proche ; arrondi à la seconde décimale.

`ceiling(x)`	Plus petit entier supérieur.
`floor(x)`	Fonction partie entière (plus grand entier inférieur).
`trunc(x)`	Troncature des décimales.
`factorial(x)`	Calcule les factorielles des éléments du vecteur `x`.
`choose(5,3)`	Valeur de C_5^3.
`rev(x)`	Renverse le vecteur `x`. Résultat : $(0, 7, -5, 1)$.
`sort(x)`	Tri par ordre croissant.
`sort(x,decreasing=TRUE)`	Tri par ordre décroissant.
`order(x)`	Renvoie l'ordre dans lequel il faut sélectionner les composantes de `x` pour avoir le tri par ordre croissant. Ici, on obtient 2, 4, 1, 3, ce qui signifie que pour trier par ordre croissant, il faut prendre le 2ème élément de `x` (c'est-à-dire -5), puis le 4ème (0), puis le 1er (1) et enfin le 3ème (7).
`order(x,decreasing=TRUE)`	Renvoie l'ordre dans lequel il faut sélectionner les composantes de `x` pour avoir le tri par ordre décroissant.
`which.min(x)`	Position du minimum. Retourne 2 (le deuxième élément de `x` est le minimum).
`which.max(x)`	Position du maximum. Retourne 3 (le troisième élément de `x` est le maximum).

2.2.4.2 Nécessité de l'utilisation de parenthèses

`1:4-1 ; 1:(4-1)`	Respectivement $(0, 1, 2, 3)$ et $(1, 2, 3)$. Dans le premier cas, `R` exécute la commande `1:4` pour former le vecteur $(1, 2, 3, 4)$ puis retranche 1 à toutes les composantes. Le second cas est équivalent à `1:3`.
`2^2^3 ; (2^2)^3`	Respectivement $2^8 = 256$ et $4^3 = 64$.

2.2.5 Composantes

`x[1]`	La 1ère composante de `x`.
`x[1:4]`	Les quatre premières composantes de `x`.
`x[c(2,3)]`	Les 2ème et 3ème composantes de `x`.
`x[-c(1,3)]`	Toutes les composantes de `x` sauf la première et la troisième.
`head(x,2)`	Extraction des 2 premiers éléments.
`tail(x,2)`	Extraction des 2 derniers éléments.
`x[2]=-3`	La 2ème composante de `x` est remplacée par 2.

2.2.6 Opérations entières

`x%%2`	Renvoie pour chaque élément du vecteur numérique `x`, le reste de la division euclidienne dudit élément par 2.
`x%/%2`	Renvoie pour chaque élément du vecteur numérique `x`, le quotient de la division euclidienne dudit élément par 2.

2.3 MATRICES (CLASSE `matrix`)

2.3.1 Création et accès aux composantes

`A=matrix(0,nrow=2,ncol=3)` `A=matrix(0,2,3)`	Commandes équivalentes permettant de créer une matrice A à 2 lignes et 3 colonnes composée de 0.
`A=matrix(1:6,nrow=2,ncol=3)`	Matrice à 2 lignes et 3 colonnes dont les composantes sont les nombres de 1 à 6. Par défaut, la matrice est remplie colonne par colonne i.e. $$A = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}.$$

`A=matrix(c(-1,3,4,-2,0,7), nrow=2, ncol=3, byrow=TRUE)`	L'argument `byrow=TRUE` permet de spécifier que le remplissage doit se faire suivant les lignes. $$A = \begin{pmatrix} -1 & 3 & 4 \\ -2 & 0 & 7 \end{pmatrix}.$$
`nrow(A) ; ncol(A)`	Nombres de lignes et de colonnes respectivement.
`dim(A)`	Un vecteur de deux composantes qui sont respectivement le nombre de lignes et le nombre de colonnes de `A`.
`A[1,3]`	Elément de la 1ere ligne et de la 3e colonne.
`A[,3]`	Toute la 3ème colonne.
`A[1,]`	Toute la 1ère ligne.
`A[1:2,2:3]`	Sous-matrice composée des deux 1ères lignes et des 2ème et 3ème colonnes.

2.3.2 Opérations et fonctions matricielles

`A+B ; A*B ; A^2`	Opérations termes à termes : `A` et `B` doivent être de même dimension.
`A%*%B`	Produit matriciel (nécessite que le nombre de colonnes de A soit égal au nombre de lignes de B).
`t(A)`	Transposée.
`diag(v)`	Si `v` est un vecteur, cette commande donne une matrice diagonale avec les éléments de `v` sur la diagonale.
`diag(A)`	Si `A` est une matrice, cette commande permet d'extraire la diagonale de `A` sous forme d'un vecteur.
`diag(k)`	Si `k` est un nombre entier naturel, cette commande permet d'obtenir la matrice identité d'ordre `k`.
`det(A)`	Déterminant d'une matrice carrée A.

`solve(A)`	Inverse de A.
`solve(A,B)`	Résolution de l'équation $AX = B$ où A est une matrice carrée d'ordre n inversible et B est un vecteur de taille n ou une matrice à n lignes.
`eigen(A)$values` `eigen(A)$vectors`	Valeurs propres de A. Vecteurs propres de A.
`apply(A,dim,f)`	Applique la fonction `f` à chaque ligne de A si `dim=1` et chaque colonne si `dim=2`.
`apply(A,1,mean); rowMeans(A)`	Commandes équivalentes pour calculer la moyenne des éléments de A par ligne.
`apply(A,2,mean); colMeans(A)`	Commandes équivalentes pour calculer la moyenne des éléments de A par colonne.
`apply(A,1,sum); rowSums(A)`	Commandes équivalentes pour calculer la somme des éléments de A par ligne.
`apply(A,2,sum); colSums(A)`	Commandes équivalentes pour calculer la somme des éléments de A par colonne.
`addmargins(A)`	Rajoute à la matrice `A` des marges contenant respectivement les sommes par lignes et par colonnes.
`cbind(A,B,C)`	Met les matrices A, B et C bout à bout horizontalement.
`rbind(A,B,C)`	Met les matrices A, B et C bout à bout verticalement.
`outer(x,y,"*"); outer(x,y,"+")`	Produit cartésien des vecteurs numériques `x` et `y` et application des opérateurs * et +.

2.4 Vecteurs de chaînes de caractères (classe `character`) et facteurs (classe `factor`)

Ces deux classes permettent de saisir des données qualitatives. La principale différence est que les facteurs (classe factor) sont beaucoup plus adaptés aux variables qualitatives ayant des modalités alors que les chaînes de caractères simples (classe character) sont souvent utilisées pour des données qualitatives n'ayant pas a priori un caractère statistique (par exemple, le nom et les prénoms).

2.4.1 Vecteurs de chaînes de caractères (classe `character`)

2.4.1.1 Création et manipulation

`y = c("Jean","Louise")`	Création d'un vecteur `y` vecteur composé de deux chaînes de caractères. Ne pas oublier l'emploi des "".
`paste("bon","jour")`	La commande `paste` permet de coller (concaténer) autant de chaînes de caractères que l'on veut. Ici, le résultat est la chaîne de caractères "bon jour". Par défaut, un espace sépare les chaînes concaténées.
`paste("bon","jour",sep="")`	Résultat : "bonjour". Ici, l'on a précisé le séparateur (`sep`) qui est une chaîne de caractères vide.
`paste(1:3,c("A","B","C"),sep="")`	On peut aussi concaténer des vecteurs. Ici le résultat est le vecteur de chaînes de caractères : `"1A" "2B" "3C"`
`as.character(10)`	Le nombre 10 devient la chaîne de caractères "10".
`nchar("Bonjour")`	Nombre de caractères de la chaîne : ici 7.
`toupper("bon"); tolower("BON")`	Conversion en majuscules / minuscules.
`substr("ABCDE",2,4)`	Extraction des caractères de la 2ème à la 4ème position. Le résultat est `"BCD"`.
`chartr("12345", "ABCDE", x)`	Remplace respectivement 1, 2, 3, 4 et 5 par A, B, C, D et E dans le vecteur de chaînes de caractères `x`
`unique(x)`	Retourne un objet de même classe que `x` en enlevant les doublons.
`help(character)`	Ouverture de la documentation de `R` sur les chaînes de caractères.

2.4.1.2 Vecteurs de chaînes de caractères internes à `R`

Le logiciel `R` contient des vecteurs de chaînes de caractères prédéfinis sous forme de constantes internes. Parmi eux, on peut citer :

`LETTERS`	$\longrightarrow$	Les 26 lettres de l'alphabet en majuscules.
`letters`	$\longrightarrow$	Les 26 lettres de l'alphabet en minuscules.
`month.name`	$\longrightarrow$	Les 12 mois de l'année en anglais .
`month.abb`	$\longrightarrow$	Les abréviations de 3 lettres de `month.name`.

2.4.1.3 Nommage des colonnes d'un vecteur

Soit `x` un vecteur numérique. La commande `names(x)` permet d'attribuer des noms aux colonnes de `x`.

Exemple 2.1 Lors de l'année scolaire 2020-2021, l'effectif des élèves dans le second cycle au Togo est réparti par région comme suit (source : https://togo.dataforall.org/) :

DAGL	Maritime	Plateaux	Centrale	Kara	Savanes
58 560	34 386	34 385	17 070	24 703	18 048

DAGL = District Autonome du Grand Lomé

Console R

```
> x=c(58560,34386,34385,17070,24703,18048)
> print(x)
[1] 58560 34386 34385 17070 24703 18048
> names(x)=c("DAGL","Maritime","Plateaux","Centrale","Kara","Savanes")
> print(x)
    DAGL Maritime Plateaux Centrale     Kara  Savanes
   58560    34386    34385    17070    24703    18048
```

On voit bien la différence à l'affichage.

On peut aussi utiliser les noms des colonnes pour sélectionner les éléments du vecteur comme suit :

`x[2] ; x["Maritime"]`	Commandes équivalentes.

2.4.1.4 Nommage des lignes et des colonnes d'une matrice

Les commandes `rownames` et `colnames` permettent d'attribuer des noms aux lignes et aux colonnes d'une matrice.

Exemple 2.2

Console R

```
> notes=matrix(c(12,15,7,13,11,20),2,3,byrow=TRUE)
> print(notes)
     [,1] [,2] [,3]
[1,]   12   15    7
[2,]   13   11   20
> rownames(notes)=c("Kodjo","Afi")
> colnames(notes)=c("MTH","PHY","SVT")
> print(notes)
      MTH PHY SVT
Kodjo  12  15   7
Afi    13  11  20
```

Ici aussi, on voit bien la différence à l'affichage.

On peut aussi utiliser les noms des lignes et des colonnes pour sélectionner les éléments d'une matrice comme suit :

`notes[1,2] ;` `notes["Kodjo","PHY"]`	Commandes équivalentes permettant d'extraire la note de Kodjo en PHY.
`notes[,1] ; notes[,"MTH"]`	Commandes équivalentes permettant d'extraire toutes les notes de MTH (première colonne).
`notes[2,] ; notes["Afi",]`	Commandes équivalentes permettant d'extraire les notes de Afi (deuxième ligne).

2.4.2 Facteurs (classe `factor`)

Cette classe d'objets permet de saisir des données concernant une variable statistique qualitative ayant plusieurs modalités. Cette structure va au-delà du simple vecteur de caractères puisqu'elle est conçue pour gérer de façon spécifique les données qualitatives (ou catégorielles).

Pour saisir des données d'une variable statistique qualitative ayant plusieurs modalités, on utilise la fonction **`factor`** comme dans l'exemple ci-après.

Exemple 2.3 Considérons les mentions obtenues par 10 élèves admis au BAC2. On peut saisir les données et les afficher avec le code suivant :

Console R

```
> mention = factor(c("B","TB","P","P","AB","P","B","P","B","TB"))
> print(mention)
 [1] B TB P P AB P B P B TB
Levels: AB B P TB
```

La deuxième ligne de l'affichage donne les modalités (levels).

`levels(mention)`	Liste des modalités.
`summary(mention)` `table(mention)`	Deux commandes équivalentes permettant d'obtenir le tableau de contingence du vecteur **`mention`** c'est-à-dire un tableau donnant les modalités et le nombre de fois que chacune apparaît.

Console R

```
> print(table(mention))
mention
AB  B  P TB
 1  3  4  2
```

2.5 Listes (classe list)

Une liste est une collection ordonnée d'objets qui peuvent être de différentes classes. C'est donc un outil très pratique pour disposer, sous la forme d'une seule variable de type liste, de plusieurs variables de différents types. Les listes présentent en particulier de l'intérêt lors de la création et l'utilisation des objets de type fonctions que nous verrons plus tard.

`etud=list(nom="Kodjo", age=21, genre="M")`	Création d'une liste à trois composantes de différentes classes : les composantes `nom` et `genre` sont de classe `character` alors que age est de classe `numeric`.
`names(etud)`	Noms des composantes de la liste `etud`.
`etud$age ; etud[[2]] ; etud[["age"]]`	Commandes équivalentes pour obtenir la seconde composante age.

2.6 Tableaux de données (classe data.frame)

Les tableaux de données (ou structures de données hétérogènes) sont des tableaux dont les lignes sont des listes de mêmes composantes et dont les colonnes peuvent être de différents types (numérique, logique, chaînes de caractères). Ils se manipulent de façon très semblable aux matrices. Ce type de structure sert à stocker des informations (en colonnes) relatives aux individus (en lignes).

2.6.1 Création par lignes de commandes

Elle se fait à l'aide de la fonction `data.frame`.

Exemple 2.4 Considérons le tableau suivant :

Nom	Age	Genre
Afi	18	F
Koffi	24	M
Kodjo	22	M
Adjo	20	F

Il s'agit d'un tableau de données de quatre (4) lignes (individus) et trois (3) colonnes (variables) avec des chaînes de caractères en 1ère colonne, du numérique dans la seconde et des facteurs dans la dernière colonne.

- Pour saisir les données, on peut utiliser le code

```
x = data.frame(nom=c("Afi","Koffi","Kodjo","Adjo"), age=c(18,24,22,20),
               genre=factor(c("F","M","M","F")))
```

ou le code

```
nom = c("Afi","Koffi","Kodjo","Adjo")
age = c(18,24,22,20)
```

```
genre = factor(c("F","M","M","F"))
x = data.frame(nom,age,genre)
```

- Pour l'affichage, on utilise le code

```
print(x)
```

qui donne comme résultat :

```
    nom age genre
1   Afi  18     F
2 Koffi  24     M
3 Kodjo  22     M
4  Adjo  20     F
```

2.6.2 Jeux de données internes

Ce sont des jeux de données installées avec R, accessibles à chaque lancement de R et qui n'ont pas besoin d'être créés par l'utilisateur.

`data()`	Commande permettant de lister tous les jeux de données internes disponibles.
`data(iris)`	Chargement du jeu de données interne nommé `iris`. La variable `iris` est maintenant un objet de l'espace de travail.

2.6.3 Importation de tableaux de données

On peut aussi importer des tableaux de données dans R depuis des fichiers d'extensions telles que `.txt`, `.csv`, `.xls` et `.xlsx`.

Un fichier de données doit être organisé de préférence, de la façon suivante :
- La première ligne peut contenir ou non les noms des composantes ou variables.
- Chaque ligne suivante correspond à un nouvel individu décrit par les variables précédentes. Le début de la ligne peut contenir ou non l'identifiant de cet individu.
- Les colonnes sont séparées par un espace ou par un autre séparateur.

Exemple 2.5 Voici un exemple de contenu de fichier à importer pour lequel les colonnes sont séparées par un espace :

`example.txt`

```
nom age genre
Afi 18 F
Koffi 24 M
Kodjo 22 M
Adjo 20 F
```

Voici les mêmes données au format CSV avec séparateur point-virgule :

example.csv

```
nom;age;genre
Afi;18;F
Koffi;24;M
Kodjo;22;M
Adjo;20;F
```

Suivant l'extension du fichier à importer, l'on pourra utiliser l'une des fonctions suivantes :

Fichiers `*.txt`	$\longrightarrow$	Fonction `read.table`
fichiers `csv`	$\longrightarrow$	Fonctions `read.csv` ou `read.csv2`.
fichiers `xls` ou `xlsx`	$\longrightarrow$	Fonctions `read.xlsx` et `read.xlsx2` de la librairie `xlxs` (à télécharger, installer et charger).

L'importation de données n'est pas abordée en détails dans ce cours. Le lecteur intéressé pourra consulter la documentation ou les références bibliographiques du cours pour plus de détails sur les fonctions sus-citées.

2.6.4 Manipulation

`View(x)`	Affichage de `x` dans le tableur interne de `R`.
`x$age ; x[,2] ; x[["age"]]; x[[2]]`	Commandes équivalentes permettant de récupérer la colonne `age` de `x`.
`names(x)`	Noms des composantes (colonnes) de `x`.
`tapply(x$age, x$genre, mean)`	Calcul de l'âge moyen par genre.
`transform(x,age.new=age+2)`	Création d'une copie de `x` et ajout d'une nouvelle colonne `age.new` obtenue en ajoutant 2 à `age`.

2.7 VECTEURS LOGIQUES (CLASSE `logical`)

Un vecteur logique est un vecteur dont chaque composante prend l'une des valeurs logiques ou valeurs de vérité **TRUE** (Vrai) et **FALSE** (Faux).

2.7.1 Création par saisie

Exemple 2.6 On interroge dix étudiants d'un cours de Statistique sur une question de type Vrai (V) ou Faux (F). Les réponses obtenues sont les suivantes :

F V F V F F F V V V

On peut saisir les données et les afficher :

Console R

```
> x=c(FALSE,TRUE,FALSE,TRUE,FALSE,FALSE,FALSE,TRUE,TRUE,TRUE)
> print(x)
 [1] FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

`as.numeric(FALSE) ; as.numeric(TRUE)`	Respectivement 0 et 1.
`sum(x)`	Lorsqu'on applique une fonction numérique à un vecteur logique, les `FALSE` et les `TRUE` deviennent respectivement 0 et 1. Ainsi, cette commande permet de compter le nombre de fois que `TRUE` apparaît dans `x`.
`table(x)`	Tableau de contingence de `x`.

2.7.2 Opérateurs de comparaison

`x=c(4,-2,5,0,7)` `print(x>3)`	Affichage d'un vecteur de même longueur que `x` avec des `TRUE` correspondant aux composantes de `x` strictement plus grandes que 3 et des `FALSE` ailleurs.
`x>=3 ; x<3 ; x<=3 ; x==3 ; x!=3`	Autres exemples de comparaison : inégalités larges, égalité, différence.

Console R

```
> x=c(4,-2,5,0,7)
> print(x>3)
[1]  TRUE FALSE  TRUE FALSE  TRUE
```

2.7.3 Opérateurs (connecteurs) logiques

Soient `y1` et `y2` deux vecteurs logiques ayant le même nombre d'éléments.

`y1 & y2`	Conjonction (« et » logique).
`y1 | y2`	Disjonction (« ou »).
`!y1`	Négation.
`xor(y1,y2)`	« ou » exclusif.

2.7.4 Fonctions logiques usuelles

`is.na(x)`	Retourne un vecteur ou une matrice de même taille que `x` indiquant pour chaque composante de `x` s'il s'agit d'une valeur manquante.
`is.finite(x)`	Retourne un vecteur ou une matrice de même taille que `x` indiquant pour chaque composante de `x` si elle est finie ou non (infinie).
`is.vector(x), is.matrix(x), is.data.frame(x)`	L'objet `x` est-il un vecteur ? une matrice ? un tableau de données ? Ces fonctions retournent une seule valeur logique.
`60%in%x; is.element(60,x)`	Commandes équivalentes pour tester si 60 est une composante de `x`.
`any(x>2)`	Retourne `TRUE` si au moins un élément de `x` est strictement supérieur à 2 et `FALSE` sinon.
`all(x>2)`	Retourne `TRUE` si tous les éléments de `x` sont strictement supérieurs à 2 et `FALSE` sinon.
`exists("y")`	Retourne `TRUE` si la variable `y` existe et `FALSE` sinon.

2.7.5 Sélection d'éléments dans un vecteur

`x=c(3,-1,0,10,12); which(x>=10)`	Positions des éléments de `x` supérieurs ou égaux à 10. Retourne un vecteur de deux éléments 4 et 5.
`x[x>2]` `subset(x,x>2)`	Commandes équivalentes permettant de récupérer les composantes de `x` strictement supérieures à 2.
`x[x>2]=1`	Remplace toutes les composantes de `x` strictement supérieures à 2 par 1.

2.7.6 Sélection d'éléments dans un tableau (matrice ou tableau de données)

Sans perte de généralités, soit `A` la matrice définie comme suit :

Console R

```
> A=matrix(-4:4,nrow=3,ncol=3)
> print(A)
     [,1] [,2] [,3]
[1,]   -4   -1    2
[2,]   -3    0    3
[3,]   -2    1    4
```

On sait que la sélection des éléments de `A` peut se faire sous l'une des trois formes `A[idx1,idx2]`, `A[idx1,]` et `A[,idx2]`, où `idx1` et `idx2` sont des vecteurs contenant respectivement les numéros des lignes et des colonnes à sélectionner. On peut remplacer ces vecteurs de numéros de lignes et de colonnes par des vecteurs logiques définis à partir des lignes et des colonnes. Dans ce cas, seules les lignes et/ou colonnes correspondant à la valeur `TRUE` seront sélectionnées.

`which(A[,2]>=0)`	Numéros des lignes pour lesquelles l'élément de la colonne 2 est supérieur ou égal à 0.
`A[A[,2]==0,]` `subset(A,A[,2]==0)`	Commandes équivalentes permettant de récupérer les lignes pour lesquelles la deuxième colonne est nulle.
`A[,A[1,]>0]`	Commande permettant de récupérer les colonnes pour lesquelles la première ligne est strictement positive.
`A[A>2]=5`	Remplace toutes les composantes de `A` strictement supérieures à 2 par 5.

2.8 EXERCICES

Exercice 2.1

1) Créer le vecteur `v1a` de composantes -1, 2.5, 3, -4, 6, 15, -4 et le vecteur `v1b` formé des entiers de 1 à 5. Créer, sans ressaisir les composantes, le vecteur `v1` composé des 5 premières composantes de `v1a` et de toutes celles de `v1b`.
2) Créer le vecteur `v2` composé de la suite des nombres allant de 0 à 2π par pas de $\pi/2$. Calculer leurs cosinus respectifs dans un vecteur `v2.cos`.
3) Créer le vecteur `v3` contenant tous les multiples de 2 compris entre 2 et 50. Combien y en a-t-il ?
4) Créer le vecteur `v4` contenant tous les multiples de 3 compris entre 2 et 50. Combien y en a-t-il ?
5) En utilisant les fonctions `intersect` et `setdiff`, créer le vecteur `v5` des multiples communs à 2 et 3 et le vecteur `v6` des multiples de 3 qui ne sont pas multiples de 2.

Exercice 2.2

1) Construire, en une seule commande, la matrice M suivante :

$$M = \begin{pmatrix} 2 & 4 & 1 & 2 \\ 3 & 4 & 5 & 6 \\ 7 & 8 & 9 & 10 \\ 11 & 12 & 35 & 7 \end{pmatrix}$$

2) Afficher :
   (a) l'élément situé à l'intersection de la première ligne et de la troisième colonne ;
   (b) la première ligne de M ;
   (c) la troisième colonne de M ;
   (d) la sous-matrice obtenue après avoir enlevé la première ligne et la première colonne de M ;
   (e) la somme des éléments de M par lignes ;
   (f) la sous-matrice obtenue en ne gardant que les lignes dont la somme est supérieure ou égale à 25.

Exercice 2.3 Ecrire et exécuter le code permettant de créer sans saisir les composantes,

1) la matrice `A` d'ordre 5×5 avec des 1 partout sauf des 0 sur la diagonale ;
2) la matrice `B` d'ordre 5×5 avec les nombres 1 à 49 de deux en deux et répartis de façon croissante colonne par colonne.
3) La matrice `B2` issue de `B` où les multiples de 3 sont remplacés par la valeur 0.

Exercice 2.4 Le jeu de données `precip` est pré-enregistré dans le logiciel `R` et regroupe des données de précipitations dans différentes villes américaines. Ecrire et exécuter le code `R` permettant de (d') :

1) importer le jeu de données `precip`;
2) afficher la classe de `precip`;
3) afficher la liste des villes concernées;
4) déterminer le nombre de villes concernées;
5) afficher le niveau de précipitation des villes suivantes : Philadelphia, Columbia, Baltimore.

Exercice 2.5 On a relevé les informations sur les appartements disponibles auprès d'un agent immobilier et obtenu les données suivantes :

loyer	cuisine	nb.chb	quartier
27000	TRUE	1	AGOE
35000	TRUE	2	TOTSI
40000	TRUE	2	KEGUE
10000	FALSE	1	BE
15000	FALSE	1	TOTSI
30000	TRUE	2	AGOE
12000	FALSE	1	TOTSI

où **loyer** indique le loyer mensuel en francs CFA, **cuisine** indique la présence ou non d'une cuisine, **nb.chb** indique le nombre de chambres et **quartier** indique le quartier où se trouve l'appartement. Ecrire et exécuter le code `R` permettant de (d') :

1) saisir les données dans un tableau de données nommé **M**;
2) déterminer le nombre d'appartements concernés;
3) calculer le loyer moyen;
4) calculer le loyer moyen par quartier;
5) déterminer le nombre de quartiers concernés;
6) déterminer le nombre d'appartements avec cuisine;
7) afficher les données de l'appartement le plus cher;
8) afficher les données de tous les appartements avec cuisine situés à Totsi;
9) ajouter à M, sans ressaisir M :
   - une colonne **caution** contenant la valeur 6 et correspondant au nombre de mois de loyer servant de caution,
   - une colonne **avance** contenant les valeurs 3, 4, 6, 2, 2, 4 et 2 correspondant au nombre de mois de loyer à payer en avance
   - et une colonne **loyer2** correspondant au loyer augmenté des charges additionnelles (par exemple, frais d'usage de fosse sceptique) s'élevant à 500 francs CFA pour tous les appartements.
10) afficher le tableau M en triant les appartements du plus cher au moins cher.

Exercice 2.6 Répondre aux questions ci-après en utilisant des commandes `R`.

1) Importer dans votre session `R` le jeu de données `WorldPhones`.

2) Afficher son contenu.
3) Quelle est sa classe ?
4) Calculer le nombre total de téléphones pour chaque année (vecteur `nb.tel.an`).
5) Quel est le continent qui a le plus / moins de numéros attribués en 1961 ?
6) En 1961, dans combien de continents y a-t-il plus de : 20 millions de téléphones ?
7) En quelle année, le nombre total de téléphones a-t-il dépassé 120 millions pour la première fois ?
8) Pour chaque année, calculer la proportion qu'occupe le nombre de téléphones de l'Afrique.

Exercice 2.7

1) Charger le jeu de données `airquality`. Que représente ce jeu de données ? Quelle est sa classe ? Quelles sont ses variables (colonnes) ? Combien a-t-il de lignes ?
2) Donner le nombre de valeurs manquantes (indiquées `NA`) pour chaque variable.
3) Créer, à partir de `airquality`, un tableau de données `x` dont les températures seront converties en degrés Celsius. On rappelle la formule de conversion de Fahrenheit en Celsius : $°C = (°F - 32)/1.8$.
4) Calculer, en degrés Celsius, les températures moyennes, minimales et maximales pour chacun des mois de mai, juin, juillet, août et septembre.
5) Calculer la concentration moyenne d'ozone (faire attention aux données manquantes !).
6) Calculer la concentration moyenne d'ozone par mois.

CHAPITRE 3

STATISTIQUE DESCRIPTIVE

Sommaire

3.1 SÉRIES STATISTIQUES À UNE VARIABLE

3.1.1 Variable quantitative

3.1.1.1 Indicateurs numériques

Soit `x` un vecteur numérique à n composantes dont les composantes sont ici notées $x_1, \ldots, x_n$.

`min(x); max(x)`	Minimum et maximum des composantes de `x`.
`sum(x)`	Somme des composantes de `x`
`mean(x)`	Moyenne des composantes de `x`

Dans `R`, il existe une fonction `var`. Mais cette fonction appliquée à un vecteur `x` calcule plutôt la **variance corrigée** des composantes de `x` dont la formule est :

$$S_n'^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2, \tag{3.1}$$

où $\overline{x}_n$ désigne la moyenne des composantes de `x`. Pour calculer la variance usuelle dont la formule est

$$S_n^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x}_n)^2, \tag{3.2}$$

on utilise la propriété

$$S_n^2 = \left(\frac{1}{n}\sum_{i=1}^{n} x_i^2\right) - (\overline{x}_n)^2 = \text{moyenne des carrés} - (\text{moyenne})^2 \tag{3.3}$$

comme suit :

`mean(x^2)-(mean(x))^2`	Variance des composantes de `x`.
`weighted.mean(c(0,1,2),c(3,2,5))`	Moyenne pondérée de 0, 1 et 2 avec poids respectifs 3, 2 et 5. Elle correspond à $\frac{3\times 0+2\times 1+5\times 2}{3+2+5}=1.2$
`median(x)`	Médiane des valeurs de `x`.
`quantile(x,0.25)`	Quantile d'ordre 0.25 des composantes de `x` (encore appelé premier quartile ou quartile Q_1).
`quantile(x,probs=c(0.25,0.75))`	Les quantiles d'ordre 0.25 et 0.75 respectivement (encore appelés quartiles).
`summary(x)`	Résumé des indicateurs numériques (moyenne, quartiles, minimum, etc.) sur le vecteur `x`. Cette fonction peut aussi être utilisée pour d'autres classes.
`table(x)`	Tableau de contingence de `x`. Ce tableau donne les différentes modalités et les effectifs associés à chaque modalité.
`prop.table(table(x))` `table(x)/length(x)`	Commandes équivalentes permettant d'obtenir le tableau des fréquences des composantes de `x`.

Exemple 3.1 On a relevé le nombre de frères et sœurs de 20 étudiants lors d'un cours de Statistique et l'on a obtenu les données suivantes :

2	4	3	2	2	1	1	3	2	4	4	0	3	3	3	4	4	0	1	3

1) On peut saisir les données à l'aide du code suivant :

```
x=c(2,4,3,2,2,1,1,3,2,4,4,0,3,3,3,4,4,0,1,3)
```

2) On peut afficher le tableau de contingence à l'aide du code suivant :

```
table(x)
```

dont le résultat est

```
x
0 1 2 3 4
2 3 4 6 5
```

La première ligne donne les modalités et la deuxième donne les effectifs associés à chaque modalité.

3) On peut afficher le tableau des fréquences à l'aide du code suivant :

```
prop.table(table(x))
```

dont le résultat est

```
x
   0    1    2    3    4
0.10 0.15 0.20 0.30 0.25
```

La première ligne donne les modalités et la deuxième donne les fréquences associées à chaque modalité.

3.1.1.2 Diagramme en barres

Soit `x` un vecteur numérique contenant les données.

`barplot(table(x))`	Affiche le diagramme en barres associé à `x`.
`barplot(table(x), xlab="...", ylab="...", col="...", border=NA)`	Affiche le diagramme en barres associé à `x`. L'argument **xlab** permet d'écrire du texte en dessous de l'axe des abscisses et l'argument **ylab** permet d'écrire du texte à gauche de l'axe des ordonnées. L'argument **col** permet de préciser la couleur. L'argument **border** permet de préciser la couleur de la bordure des barres (ici l'option **border=NA** supprime la bordure des barres).

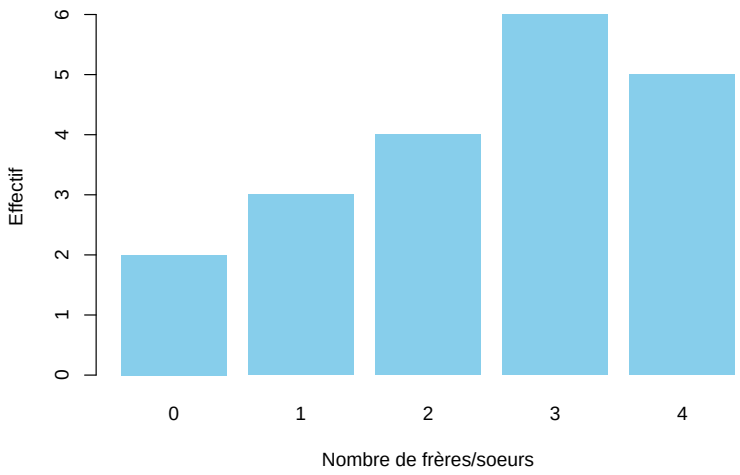
Remarque 3.1 Dans `R`, il existe 657 couleurs prédéfinies. Leur liste est disponible grâce à la commande

```
colors()
```

Parmi les noms de couleurs classiques, on note : `"blue"`, `"red"`, `"green"`, `"yellow"`, `"gray"`, `"lightgray"`, `"yellowgreen"`, `"lightblue"`, `"skyblue"`, `"lightgreen"`, `"tomato"`.

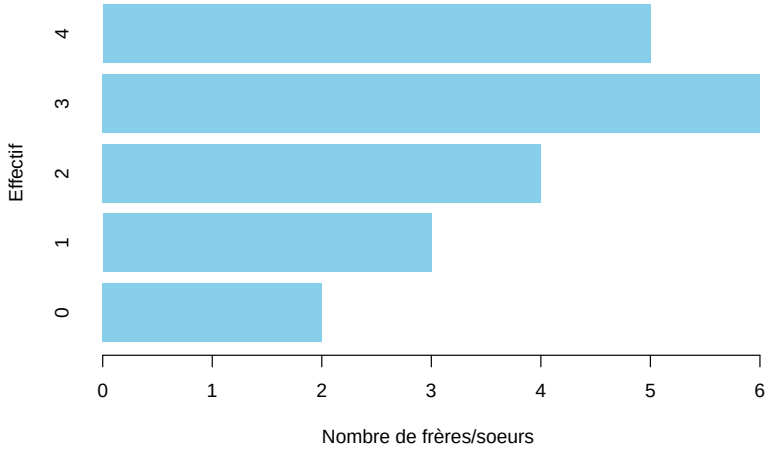
Exemple 3.1 (suite).

```
barplot(table(x), xlab="Nombre de frères/soeurs", ylab="Effectif",
        col="skyblue", border=NA)
```



On peut avoir des barres horizontales en ajoutant l'option **horiz=TRUE** comme suit :

```
barplot(table(x), xlab="Nombre de frères/soeurs", ylab="Effectif",
        col="skyblue", border=NA, horiz=TRUE)
```



3.1.1.3 Diagramme en bâtons

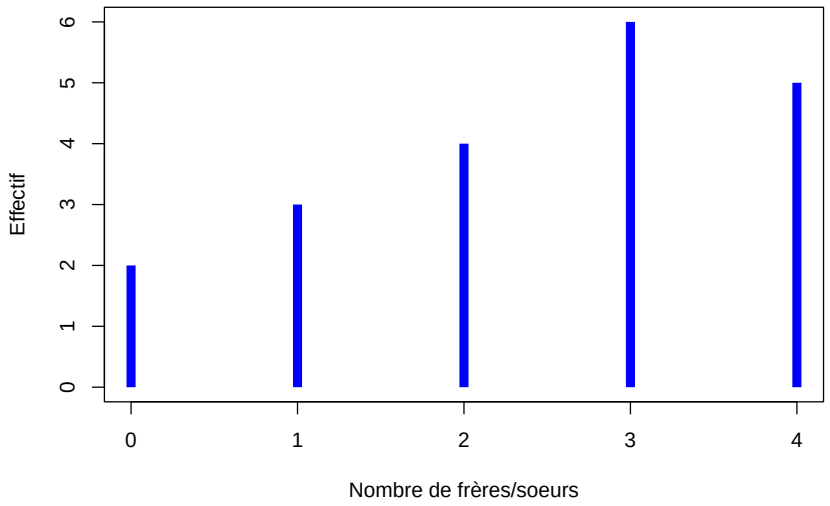
Soit x un vecteur numérique contenant les données.

```
plot(table(x),lwd=5,lend=2)
```

Construction du diagramme en bâtons associé à `x`. L'option `lwd` (line width) est un entier naturel spécifiant la largeur de chaque trait. L'option `lend` (line end) permet de préciser la forme des extrémités (0 = extrémité arrondie ou circulaire, 1 = extrémité plate ou carrée).

Exemple 3.1 (suite).

```
plot(table(x), lwd=7, lend=1, xlab="Nombre de frères/soeurs",
     ylab="Effectif", col="blue")
```




```
plot(x,y,type="h")
```

L'option `type="h"` permet aussi de construire un diagramme en bâtons dont `x` contient les modalités et `y` contient les effectifs.

3.1.1.4 Histogramme

L'histogramme permet de représenter une variable statistique continue groupée (définie par intervalles).

Exemple 3.2 On a noté l'âge (arrondi à l'année près) de 48 salariés d'une entreprise. La série statistique brute (données obtenues pendant l'enquête) est donnée ci-dessous :

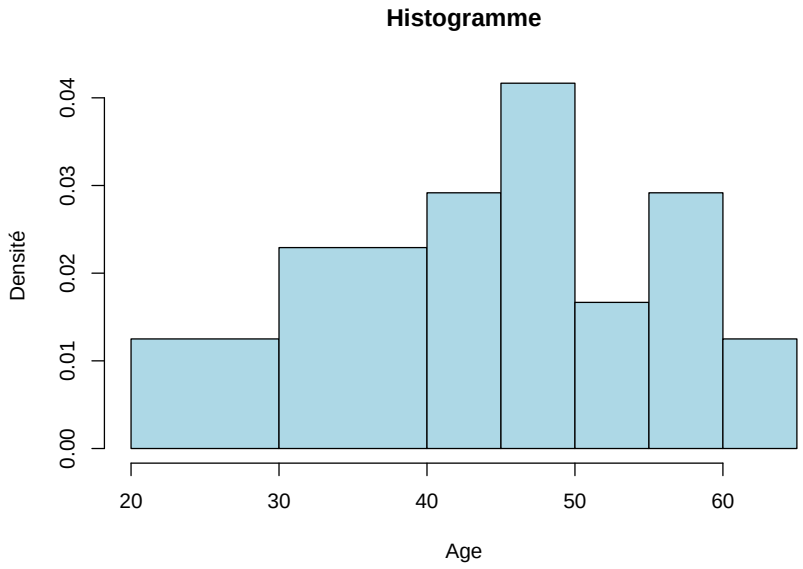
43	29	57	45	50	29	37	59	46	31	46	24
33	38	49	31	62	60	52	38	43	26	41	52
60	49	52	41	38	26	37	59	57	41	29	33
33	43	46	57	46	33	46	49	57	57	46	43

On peut saisir les données à l'aide de la commande suivante :

```
age = c(43, 29, 57, 45, 50, 29, 37, 59, 46, 31, 46, 24,
        33, 38, 49, 31, 62, 60, 52, 38, 43, 26, 41, 52,
        60, 49, 52, 41, 38, 26, 37, 59, 57, 41, 29, 33,
        33, 43, 46, 57, 46, 33, 46, 49, 57, 57, 46, 43)
```

L'**histogramme en fréquences** (ou en densité) du vecteur `x` en considérant les classes d'âges $[20, 30[$, $[30, 40[$, $[40, 45[$, $[45, 50[$, $[50, 55[$, $[55, 60[$ et $[60, 65]$ peut être obtenu à l'aide de la fonction **hist** comme suit :

```
h = hist(age, breaks = c(20, 30, 40, 45, 50, 55, 60, 65), right = FALSE,
         freq = FALSE, col="lightblue", xlab="Age", ylab="Densité",
         main="Histogramme")
```



- Par défaut, les classes sont semi-fermées à droite et la première est fermée (`right = TRUE`). L'argument `right = FALSE` permet de spécifier que les classes soient semi-fermées à gauche et que la dernière classe soit fermée.
- L'option `freq = FALSE` est obligatoire pour obtenir un histogramme au sens classique du terme c'est-à-dire un histogramme des fréquences.
- L'argument `col` précise la couleur, les arguments `xlab` et `ylab` précisent les textes à écrire respectivement en dessous de l'axe des abscisses et à gauche de l'axe des ordonnées et l'argument `main="Histogramme"` permet de préciser le titre du graphique.

`h$counts`	Les effectifs des classes.
`prop.table(h$counts)`	Les fréquences des classes.
`h$mids`	Les centres des classes.

`h=hist(age,freq=FALSE,breaks=5)`	Histogramme avec cinq (5) classes de même amplitude.

3.1.2 Variables qualitatives

Soit `x` un vecteur de classe **factor** ou **character** ou **logical** considéré comme un vecteur d'observations d'une variable qualitative.

3.1.2.1 Tableau de contingence et fréquences

`table(x)`	Tableau de contingence pour les observations contenues dans le vecteur `x`.
`freq=prop.table(table(x))` `freq=proportions(table(x))`	Commandes équivalentes permettant d'obtenir le tableau des fréquences.

3.1.2.2 Diagramme circulaire

`pie(freq)`	Affiche un camembert (diagramme circulaire) dont les fréquences sont contenues dans le vecteur `freq`.

Exemple 3.3 On a relevé la région de provenance de 15 étudiants participant à un cours de Statistique et obtenu les données suivantes :

KAR	SAV	SAV	KAR	CEN
CEN	CEN	KAR	SAV	KAR
KAR	MAR	MAR	KAR	SAV

où les régions sont codées comme suit : CEN = Centrale, KAR = Kara, MAR = Martime, SAV = Savanes.

- On peut saisir les données à l'aide du code suivant :

```
region = factor(c("KARA","SAVANES","SAVANES","KARA","CENTRALE",
                  "CENTRALE","CENTRALE","KARA","SAVANES","KARA",
                  "KARA","MARITIME","MARITIME","KARA","SAVANES"))
```

- On peut afficher le tableau de contingence à l'aide du code

```
table(region)
```

dont le résultat est :

```
region
CENTRALE     KARA MARITIME  SAVANES
       3        6        2        4
```

- On peut afficher le tableau des fréquences à l'aide du code

```
freq=prop.table(table(region))
print(freq)
```

dont le résultat est :

```
region
 CENTRALE      KARA  MARITIME   SAVANES
0.2000000 0.4000000 0.1333333 0.2666667
```

- On peut arrondir le tableau des fréquences à deux chiffres après la virgule à l'aide du code

```
freq=round(prop.table(table(region)),2)
print(freq)
```

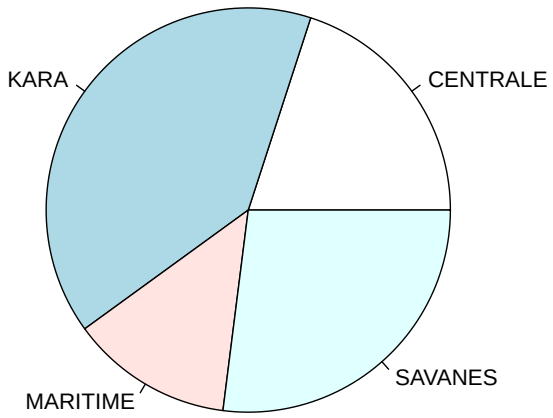
dont le résultat est :

```
region
CENTRALE     KARA MARITIME  SAVANES
    0.20     0.40     0.13     0.27
```

- On peut afficher le diagramme circulaire à l'aide du code

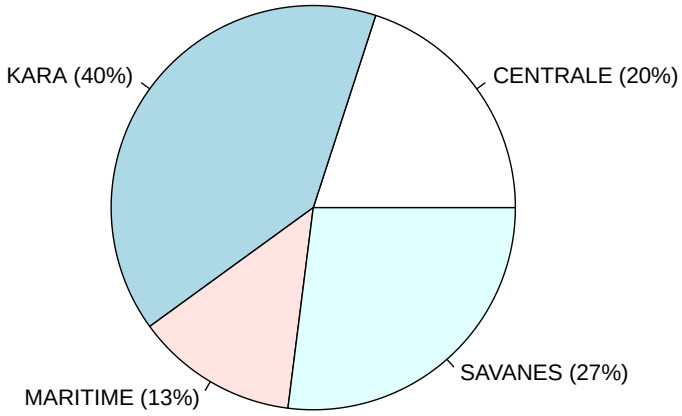
```
pie(freq)
```

dont le résultat est :



On peut afficher les fréquences entre parenthèses sur le diagramme en utilisant l'argument **labels** qui permet de donner le texte à afficher sur le diagramme et la commande **paste** qui permet de coller des chaînes de caractères.

```
pie(freq,labels=paste(names(freq)," (",100*freq,"%)",sep=""))
```



3.2 Séries statistiques à deux variables quantitatives

Soient x et y deux vecteurs de n composantes contenant des données numériques.

3.2.1 Nuage de points

`plot(x,y)`	Représentation graphique du nuage de points des variables x et y.

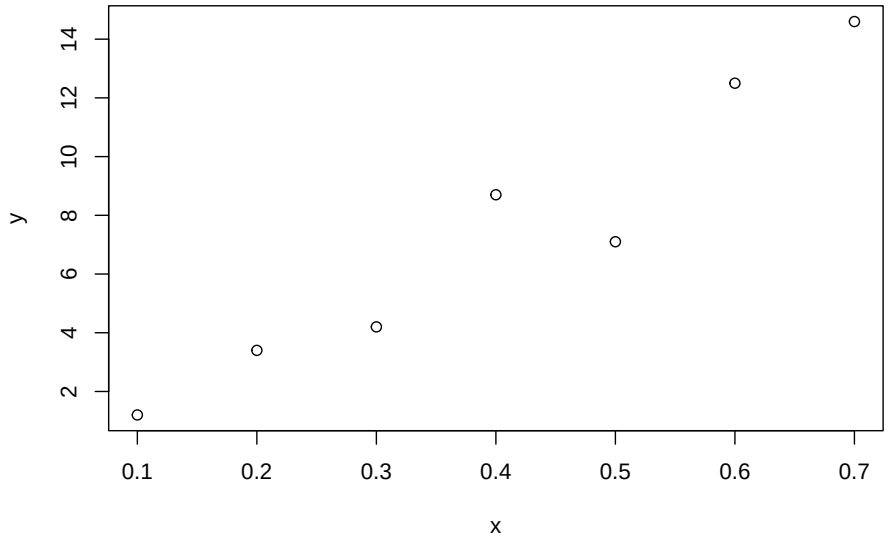
Exemple 3.4 Considérons deux variables statistiques x et y dont les observations sont données par le tableau suivant :

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7
y	1.2	3.4	4.2	8.7	7.1	12.5	14.6

On peut saisir les données et représenter le nuage de points à l'aide du code suivant :

```
x = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7)
y = c(1.2, 3.4, 4.2, 8.7, 7.1, 12.5, 14.6)
plot(x,y)
```

On obtient comme résultat :



On peut changer le symbole utilisé pour représenter chaque point du nuage à l'aide du paramètre `pch` (plotting character) qui prend comme valeur un entier compris entre 0 et 25. Le tableau 3.1 donne sur la première ligne les différentes valeurs du paramètre `pch` et, sur la deuxième ligne, les symboles associés à ces différentes valeurs.

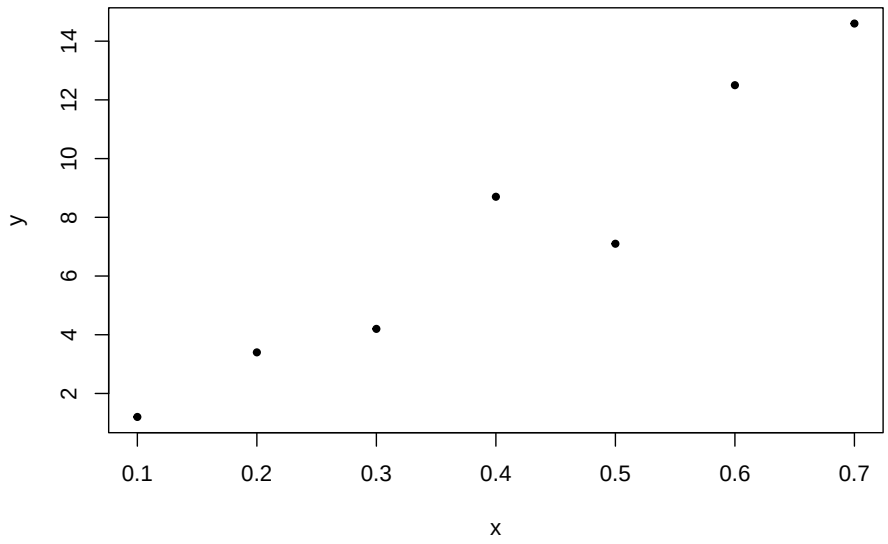
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
□	○	△	+	×	◇	▽	⊠	✳	⊕	⊕	✡	⊞	⊠	⧄	■	●	▲	◆	●	•	○	□	◇	△	▽

TABLE 3.1 – Valeurs du paramètre `pch` et symboles correspondants

Ainsi, le code

```
plot(x,y,pch=20)
```

donne comme résultat :



3.2.2 Coefficient de corrélation linéaire

`cor(x,y)`	Coefficient de corrélation linéaire entre `x` et `y`.

3.2.3 Ajustement linéaire

L'ajustement linéaire par la méthode des moindres carrés ordinaires (MCO) se fait grâce à la fonction `lm` (signifiant linear model) comme suit :

`reg=lm(y~x)`	Ajustement linéaire de la forme $y = ax + b$. L'estimation des coefficients a et b est faite par la méthode MCO.

L'objet **`reg`** contient alors les résultats de l'ajustement linéaire. Pour afficher les valeurs des coefficients a et b, on peut utiliser le code suivant :

```
print(reg$coefficients)
```

qui donne comme résultat :

```
(Intercept)           x
  -1.371429   21.892857
```

L'objet **`reg$coefficients`** est un vecteur numérique à deux composantes :

- la première désignée par le terme **Intercept** représente la constante c'est-à-dire le coefficient b ;

- la seconde composante est le coefficient qui multiplie x c'est-à-dire le coefficient a.

On a donc

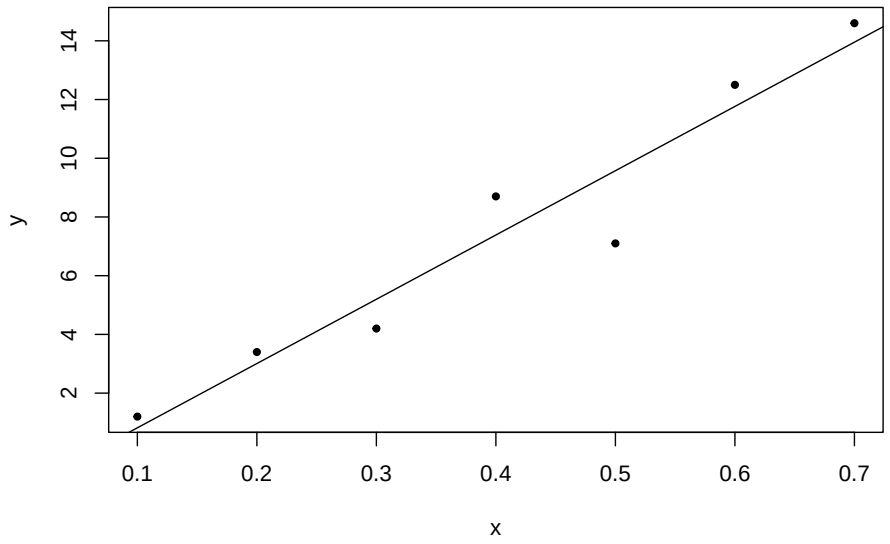
$$a = 21.892857 \qquad \text{et} \qquad b = -1.371429.$$

```
a=reg$coefficients[2]
b=reg$coefficients[1]
```

Récupération des coefficients du modèle.

On peut tracer la droite de régression sur le nuage de points en utilisant la fonction **abline** comme suit :

```
abline(reg$coefficients)
```



3.3 EXERCICES

Exercice 3.1 On a relevé le nombre d'enfants de 24 couples. Les données relevées sont les suivantes :

2	2	0	2	4	3	2	1	2	1	0	0
0	2	2	1	3	0	1	2	1	0	2	2

1) Saisir les données dans un vecteur nommé `x`.
2) Faire le tableau de contingence et calculer les fréquences.
3) Représenter les données par un diagramme en bâtons.
4) Calculer la moyenne, la variance et l'écart-type du nombre d'enfants.

Exercice 3.2 En 2019, la population togolaise est estimée par région comme suit (source : https://togo.dataforall.org/) :

DAGL	Maritime	Plateaux	Centrale	Kara	Savanes
1 953 418	1 226 928	1 666 625	750 733	935 684	995 206

DAGL = District Autonome du Grand Lomé

1) Saisir les données dans un vecteur numérique nommé `pop` dont les colonnes seront nommées telles qu'à l'affichage, l'on ait le résultat suivant :

```
   DAGL Maritime Plateaux Centrale     Kara  Savanes
1953418  1226928  1666625   750733   935684   995206
```

2) Calculer l'effectif total estimé de la population togolaise en 2019.
3) Calculer les fréquences par région (arrondies à deux chiffres après la virgule) puis les représenter à l'aide d'un diagramme circulaire sur lequel figureront les noms des régions.

Exercice 3.3 L'observation du prix de la tomate en fonction des quantités vendues sur un marché a donné les résultats suivants :

Quantités x (kg)	10	20	35	50	70	90	110	130
Prix y du kg (kFCFA)	5	3.75	2.75	2.25	1.75	1.25	0.8	0.5

Ainsi, une quantité de 35 kg de tomates est vendue au prix de 2750 FCFA le kg.

1) Représenter graphiquement le nuage de points.
2) On cherche un ajustement non linéaire de la forme

$$y = f(x) = a \ln x + b.$$

Déterminer les coefficients a et b par la méthode MCO (on pourra poser $z = \ln x$ et se ramener à un ajustement linéaire $y = az + b$).
3) Prévoir le prix du kg pour un achat de 140 kg.
4) En utilisant de la fonction `curve`, ajouter une représentation de la fonction f sur le nuage de points.

CHAPITRE 4

PROGRAMMATION DANS R

Sommaire

4.1 STRUCTURES DE CONTRÔLE

4.1.1 Exécution conditionnelle avec `if`

Il existe deux façons d'utiliser l'exécution conditionnelle avec **if**. On peut utiliser la syntaxe

Console R

```
if(condition)
{
    # commandes à exécuter si condition est vraie
}
```

ou la synthaxe

Console R

```
if(condition)
{
    # commandes à exécuter si condition est vraie
} else {
    # commandes à exécuter sinon
}
```

où `condition` est une variable de classe logique ne contenant qu'une seule valeur (Si c'est un vecteur logique de longueur supérieure ou égal à 2, seul le premier élément est utilisé). La différence entre les deux syntaxes est que la seconde précise le traitement à réaliser lorsque condition prend la valeur `FALSE`.

Remarque 4.1 Les accolades permettent de délimiter une commande ou un ensemble de commandes.

Remarque 4.2 Lorsqu'elle est utilisée, la commande `else` doit être placée **obligatoirement** sur la même ligne que l'accolade fermante de `if`.

Exemple 4.1 Le code suivant

```
x=3
if(x%%2==0)
{
    print("x est pair")
}
```

est correct mais n'affiche aucun résultat. En effet, l'opération `x%%2` donne le reste de la division euclidienne de `x` par 2. La condition `x%%2==0` permet de tester si ce reste est nul. Ici, le reste de la division euclidienne de 3 par 2 est égal à 1 et comme 1 est différent de 0, la condition `x%%2==0` vaut `FALSE` et donc la commande entre accolades n'est pas exécutée.

Exemple 4.2 Le code suivant

```
x=3
if(x%%2==0)
{
    print("x est pair")
} else {
    print("x est impair")
}
```

affiche le résultat suivant :

```
[1] "x est impair"
```

4.1.2 Boucle `for`

Elle permet d'exécuter un bloc de commandes pour des valeurs fixées d'une variable appelée **compteur**. Sa syntaxe est la suivante :

Console R

```
for(compteur in vecteur)
{
    # commandes à exécuter
}
```

où `compteur` représente le nom de la variable compteur et `vecteur` est un vecteur contenant les différentes valeurs du compteur.

Exemple 4.3 Le code suivant

```
for(i in 1:5)
{
    print(i)
}
```

donne comme résultat :

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

4.1.3 Boucle `while`

Elle permet d'exécuter un bloc de commandes tant qu'une condition est vraie. Sa syntaxe est la suivante :

Console R

```
while(condition)
{
    # commandes à exécuter
}
```

Exemple 4.4 Le code suivant

```
i=1
while(i<=5)
{
    print(i)
    i=i+1
}
```

donne comme résultat

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Il faut s'assurer que parmi les commandes à exécuter, il y a en ait une qui permette de mettre à jour la condition sinon l'on risque d'avoir une boucle infinie (une boucle qui ne s'arrête jamais). Dans l'exemple 4.4, l'on peut remarquer que la commande `i=i+1` permet de mettre à jour la valeur de la variable `i` afin d'éviter la boucle infinie. Si l'on ne l'avait pas mise, la variable resterait bloquée à 1 et comme 1 est inférieur ou égal à 5, la boucle va afficher des 1 de façon infinie (sans s'arrêter).

4.2 Création de nouvelles fonctions

Une fonction est un objet permettant de stocker un ensemble de commandes dans l'espace de travail. Il existe de nombreuses fonctions prédéfinies dans `R` que nous avons déjà vues dans les chapitres précédents (`c`, `data.frame`, `matrix`, `table`, `factor`, `cos`, `sin`, `mean`, etc...). Il est également possible pour un utilisateur de créer de nouvelles fonctions.

4.2.1 Création

```
somme = function(a,b)
{
    return(a+b)
}
```

Création d'une fonction `somme` permettant de calculer la somme des deux nombres `a` et `b` donnés en arguments.

4.2.2 Appel de la fonction

```
somme(2,5)
somme(a=2,b=5)
somme(b=5,a=2)
```

Appels équivalents de la fonction `somme` avec `a=2` et `b=5`. Le résultat est 7.

4.2.3 Affichage du code et des arguments d'une fonction

```
print(somme)
```

Affiche le code de la fonction `somme`.

```
args(somme)
```

Affiche les noms des arguments de la fonction `somme`.

4.2.4 Sorties multiples

```
som.prod = function(a,b)
{
    s=a+b
    p=a*b
    return(list(s=a+b, p=a*b))
}
```

Pour sortir plusieurs résultats en même temps d'une fonction, il faut créer une liste avec ces résultats. Ici, la fonction donne une liste de première composante `s` et de deuxième composante `p`.

4.2.5 Arguments par défaut

```
somme = function(a,b=0){
return(a+b)}
```

Par défaut `b` vaut 0. Si `b` n'est pas spécifié, il sera mis à 0.

```
somme(1,0) ; somme(1)
```

Commandes équivalentes.

4.2.6 Gestion des erreurs sur les arguments

Dans l'écriture d'une fonction, il peut être nécessaire de vérifier si les arguments vérifient certaines conditions (par exemple, la positivité). Dans ce cas, on peut utiliser la commande **stop** qui permet d'afficher un message d'erreur et d'arrêter l'exécution du code de la fonction.

Exemple 4.5 Considérons le code suivant :

```
f = function(x)
{
    if(x<=0){stop("L'argument ne doit pas être négatif.")}
    return(1/sqrt(x))
}
```

Exécutons ce code pour des valeurs positives et négatives de l'argument. On a par exemple :

Console R

```
> f(4)
[1] 0.5
> f(7)
[1] 0.3779645
> f(0)
Error in f(0) : L'argument ne doit pas être négatif.
> f(-3)
Error in f(-3) : L'argument ne doit pas être négatif.
```

4.3 EXERCICES

Exercice 4.1 Créer une matrice carrée A d'ordre 7 telle que

$$\begin{cases} a_{ii} = 2 & \text{si} \quad i = 1, \ldots, 7 \\ a_{i,i+1} = 1 & \text{si} \quad i = 1, \ldots, 6 \\ a_{i,i-1} = -1 & \text{si} \quad i = 2, \ldots, 7 \\ a_{ij} = 0 & \text{dans tous les autres cas.} \end{cases}$$

Exercice 4.2 Programmer la fonction factorielle sous forme d'une fonction `R` récursive (c'est à dire qui s'appelle elle-même). Cette fonction sera appelée **fact** et prendra comme seul argument un entier naturel **n**. Elle retournera un message d'erreur lorsque l'argument n'est pas un entier naturel.

Exercice 4.3 Ecrire une fonction **matrix.power** qui prend en arguments une matrice carrée A et un entier naturel n et retourne la matrice A^n (on prendra $A^0 = I$). Un message d'erreur sera affiché si A n'est pas une matrice carrée ou si n n'est pas un entier naturel.

Exercice 4.4 On s'intéresse à la suite de terme général $S_n = 1 + 2^2 + 3^2 + \cdots + n^2$.

1) Ecrire deux fonctions `f1` et `f2` qui prennent en argument un entier naturel n et calculent S_n de deux façons différentes : l'une avec boucle et l'autre sans aucune boucle.
2) Que font les fonctions suivantes : `Sys.time` ? `difftime` ?
3) Proposer une méthode pour mesurer le temps d'exécution de la fonction `f1`.
4) Pour $n \in \{10,\ 10^2,\ 10^3,\ 10^4,\ 10^5,\ 10^6,\ 10^7\}$, mesurer les temps de calcul (en secondes) de S_n par les deux méthodes. Les résultats seront stockés sous la forme d'un tableau à trois colonnes correspondant respectivement aux valeurs de n, aux temps d'exécution de `f1` et `f2`. Que peut-on conclure ?

CHAPITRE 5

FONCTIONNALITÉS GRAPHIQUES

Sommaire

5.1 GÉNÉRALITÉS

Les fonctionnalités graphiques sont une composante extrêment importante du système R. Il existe trois (03) types de commandes graphiques :

- Les fonctions **haut niveau** qui permettent de créer un nouveau graphique.
- Les fonctions **bas niveau** qui permettent d'ajouter des informations sur un graphique existant.
- Les fonctions **interactives** (non vues dans ce cours) qui permettent d'interagir sur le graphique afin d'ajouter ou d'en extraire des informations à l'aide de la souris.

5.2 GESTION DE LA FENÊTRE GRAPHIQUE

5.2.1 Création et fermeture

`x11() ; X11() ; windows(); dev.new()`	Commandes équivalentes pour ouvrir une nouvelle fenêtre graphique vierge.
`dev.list()`	Vecteur contenant les numéros des fenêtres graphiques ouvertes.
`dev.set(1)`	La fenêtre graphique numéro 1 est activée.

`dev.off(1)`	Ferme la fenêtre graphique numéro 1.
`dev.off()`	Ferme la fenêtre graphique active.
`graphics.off()`	Ferme toutes les fenêtres graphiques.

5.2.2 Fractionnement d'une fenêtre grapique

La figure 5.1 donne un exemple de fenêtre graphique fractionnée.

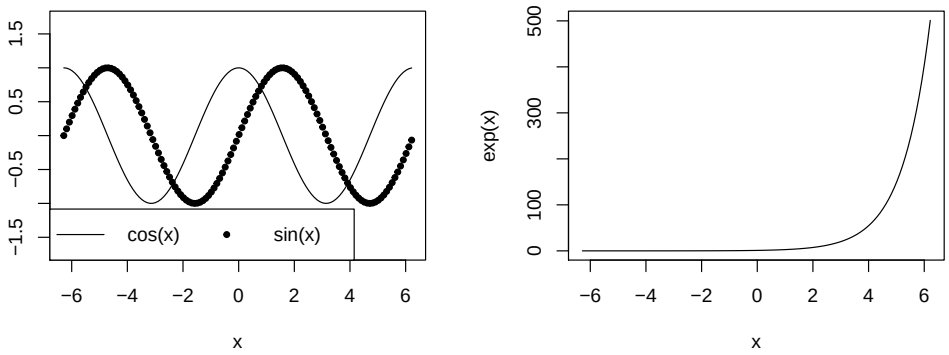


FIGURE 5.1 – Exemple de fenêtre graphique fractionnée

`split.screen(c(2,3))`	Fractionnement de la fenêtre graphique active en 6 graphiques (2 lignes et 3 colonnes) numérotées par lignes.
`screen(5)`	Active le 5ème graphique de la fenêtre active (donc celui en 2ème ligne et 2ème colonne).

5.2.3 Sauvegarde d'une fenêtre graphique dans un fichier

`dev.size("px")`	Donne les dimensions en pixels de la fenêtre graphique active.
`dev.print(pdf, "essai.pdf", width=6, height=4)`	Sauvegarde la fenêtre graphique active dans le fichier pdf "essai.pdf" avec les dimensions en pouces (1 pouce = 2.54cm). Le paramètre `pdf` peut être remplacé par `postscript` (fichier "essai.ps").

`dev.print(png, "essai.png", width=700, height=600)`	Sauvegarde la fenêtre graphique active dans le fichier png "essai.png" avec les paramètres spécifiés. Il faut préciser les dimensions (l'unité est le pixel). Le paramètre `png` peut être remplacé par `jpeg` (fichier "essai.jpg").

5.3 FONCTIONS HAUT NIVEAU

Les fonctions haut niveau créent un nouveau graphique dans la fenêtre graphique active, éventuellement avec des caractéristiques complémentaires (axes, labels, titres, etc.). Si un graphique existait déjà dans la fenêtre, alors il est effacé.

5.3.1 Commande générique : la commande `plot`

`plot(x)`	`x` vecteur numérique : cette commande produit le nuage de points (i, x_i) avec les composantes de `x` en ordonnées et les index des composantes de `x` en abscisses.
`plot(x,y)`	`x` et `y` sont des vecteurs numériques de même longueur : cette commande produit le nuage de points (x_i, y_i) avec les composantes de `x` en abscisses et celles de `y` en ordonnées.
`plot(M)`	`M` est une matrice à deux colonnes : cette commande produit un nuage de points avec les composantes de la 1ère colonne en abscisses et celles de la 2ème colonne en ordonnées.
`x=seq(-2,5,0.1)` `plot(x,exp(x))`	Représentation graphique de la fonction exponentielle à l'aide d'un nuage de points sur l'intervalle $[-2, 5]$.

Amélioration. Par défaut, la commande `plot` représente un nuage de points où chaque point est représenté par un cercle. Il est possible d'améliorer l'aspect du graphique en spécifiant la valeur de certains paramètres.

`plot(x,y,type="p",pch="+")`	Nuage de points avec le symbole "+".
`plot(x,y,type="l")`	Points reliés par des lignes.
`plot(x,y,type="o")`	Nuage de points représentés par des cercles et reliés par des lignes.

`plot(x,y,type="n")`	N'affiche rien mais prépare le graphique (création, axes) : utile pour les fonctions bas niveau.
`plot(x,y, axes=FALSE)`	Graphique sans axes.
`plot(x,y, main="Titre", xlab="X", ylab="Y")`	Titre du graphique, noms des axes.
`plot(x,y,col="blue")`	Gestion de la couleur.
`plot(x,y, xlim=c(-1,3), ylim=c(0,2))`	Les paramètres `xlim` et `ylim` permettent de délimiter l'axe des abscisses et l'axe des ordonnées.
`plot(x,y, lwd=2)`	Le paramètre `lwd` (line width) permet de gérer l'épaisseur des traits.
`help("par")`	Consulter l'aide pour plus de détails sur les paramètres graphiques.
`colors()`	Affiche la liste des couleurs disponibles.

5.3.2 Autres fonctions

Il existe d'autres fonctions haut niveau dont nous présenterons certaines dans les chapitres suivants concernant spécifiquement la statistique et la probabilité.

5.4 FONCTIONS BAS NIVEAU

Les fonctions bas niveau ajoutent des informations (points, lignes, légendes, titres) à un graphique déjà créé par une fonction haut niveau sans effacer ledit graphique. Il faut donc avoir déjà utilisé une fonction haut niveau auparavant.

5.4.1 Ajout de points, de lignes ou de symboles

`points(x,y,pch="+")`	Commande similaire à `plot(x,y, type="p", pch="+")` mais sans effacer le graphique.
`lines(x,y)`	Commande similaire à `plot(x,y, type="l")` mais sans effacer le graphique.
`lines(x,y,col="blue",lty="dotted")`	Spécification de la couleur (bleue) et du type de ligne (pointillés).
`abline(b=2,a=-3)`	Rajoute la droite d'équation $y = a + bx$ sur la fenêtre graphique active.

`segments(x0, y0, x1, y1)`	Trace une ligne du point `(x0,y0)` au point `(x1,y1)`.

5.4.2 Ajout de texte

`text(x,y,noms)`	En chaque position (x_i, y_i), affiche la composante correspondante du vecteur de chaînes de caractères `noms`.
`title("Titre du graphique")`	Titre du graphique.
`legend("topright", legend=c("courbe 1","courbe 2"), col=c("blue","red"), lty=c(1,2))`	Légende placée en haut à droite avec les deux libellés courbe 1 et courbe 2, le 1er associé à une ligne continue bleue, le 2nd à une ligne pointillée rouge.
`help(plotmath)`	Afficher l'aide sur l'ajout de symboles mathématiques (racine carrée, exposants, etc...) sur un graphique.

5.5 EXERCICES

Exercice 5.1 Représenter chacune des fonctions suivantes sur l'intervalle mentionné :

$$f : x \longmapsto e^x, \qquad I = [-3, 4]$$

$$g : x \longmapsto \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}, \qquad I = [-4, 4]$$

$$h : x \longmapsto \log(x), \qquad I =]0, 10]$$

Pour chaque graphique, veiller à renseigner les noms des axes ainsi que le titre.

Exercice 5.2 Représenter les fonctions sinus et cosinus sur $[-2\pi, 2\pi]$ superposées sur le même graphique à la manière de la figure 5.2.

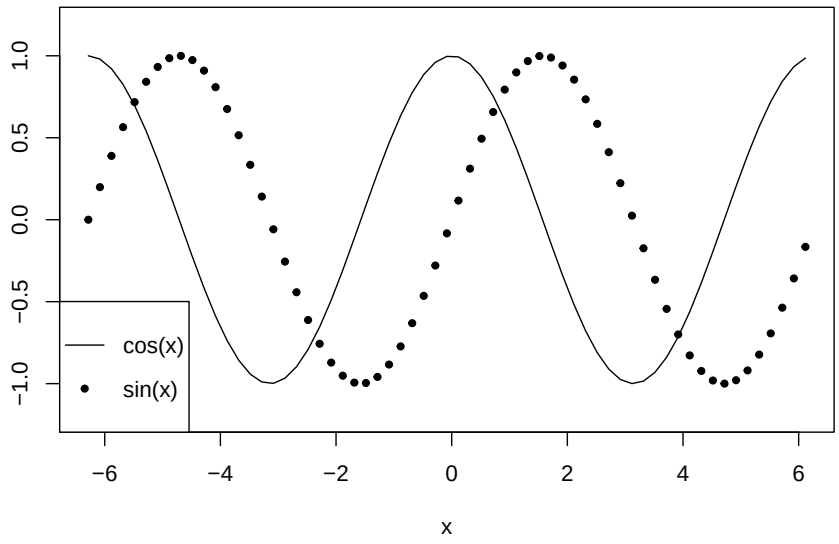


FIGURE 5.2 – Représentation des fonctions sinus et cosinus sur $[-2\pi, 2\pi]$.

Exercice 5.3 Tracer les courbes paramétrées suivantes pour $t \in [-10, 10]$:

1) $x(t) = \cos^2(t), \quad y(t) = \cos^3(t)\sin(t)$.

2) $x(t) = \dfrac{t}{1+t^4}, \quad y(t) = \dfrac{t^3}{1+t^4}$.

3) $x(t) = \sin^3(t), \quad y(t) = \cos(t) - cos^4(t)$.

4) $x(t) = [250 + a\sin(bt)]\cos(t), \quad y(t) = [250 + a\sin(bt)]\sin(t), \quad a = 220$ et $b = 6$.

Exercice 5.4 Représenter sur un même graphique les fonctions $x \mapsto x^2$ et $x \mapsto \sqrt{x}$ sur les intervalles $[0, 3]$ et $[0, 9]$ respectivement. Ajouter au graphique obtenu, la première bissectrice en pointillés pour apprécier la symétrie des deux courbes par rapport à cette dernière. Le résultat obtenu sera semblable à la figure 5.3.

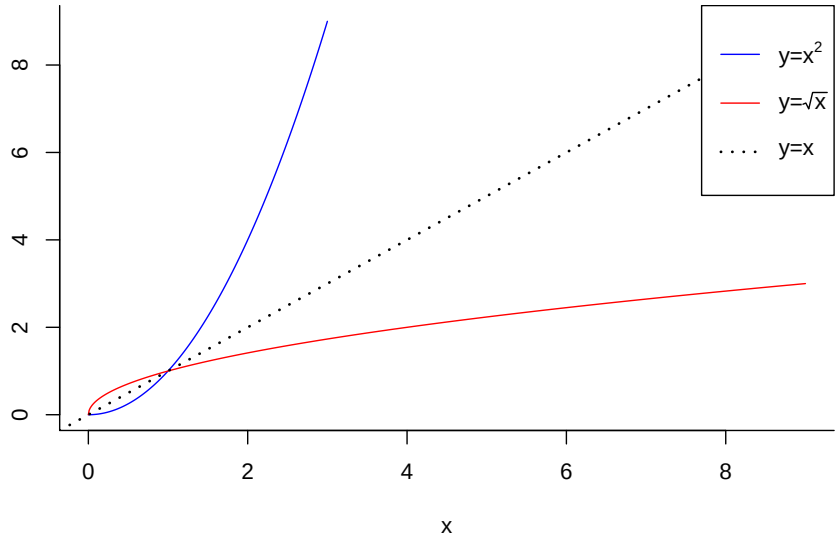


FIGURE 5.3

CHAPITRE 6

CALCUL DE PROBABILITÉS ET SIMULATIONS

Sommaire

6.1 LOIS DE PROBABILITÉS USUELLES

Toutes les lois de probabilités usuelles sont disponibles dans R. Chaque loi est identifiée par une abréviation :

TABLE 6.1 – Liste des lois de probabilités usuelles

Loi	Abréviation
Beta	`beta`
Binomiale (y compris Bernoulli)	`binom`
Binomiale négative	`nbinom`
Cauchy	`cauchy`
Exponentielle	`exp`
Fisher-Snedecor	`f`
Gamma	`gamma`
Géométrique	`geom`
Hypergéométrique	`hyper`
Khi-deux	`chisq`
Log-normale	`lnorm`
Logistique	`logis`
Normale	`norm`
Poisson	`pois`
Student	`t`
Uniforme	`unif`
Weibull	`weibull`

Pour chaque loi d'abréviation `xxx`, quatre (04) fonctions sont disponibles, identifiées par un préfixe :

`dxxx`	Fonction densité de la loi `xxx`.

`pxxx`	Fonction de répartition de la loi `xxx`.
`qxxx`	Fonction quantile de la loi `xxx`.
`rxxx`	Génération de nombres aléatoires suivant la loi `xxx`.

Le lecteur est invité à consulter l'aide pour plus de détails sur les paramètres des différentes lois. Nous donnons ci-dessous un exemple de loi discrète et un exemple de loi continue.

`dbinom(3,20,1/2)`	Probabilité individuelle de la loi binomiale $\mathcal{B}(20, \frac{1}{2})$ évaluée en 3 c'est-à-dire $\mathbb{P}(X = 3)$ lorsque $X \rightsquigarrow \mathcal{B}(20, \frac{1}{2})$.
`pbinom(3,20,1/2)`	Fonction de répartition de la loi $\mathcal{B}(20, \frac{1}{2})$ évaluée en 3 c'est-à-dire $\mathbb{P}(X \leqslant 3)$.
`qbinom(0.9,20,1/2)`	Quantile d'ordre 0.9 de $\mathcal{B}(20, \frac{1}{2})$.
`rbinom(10,20,1/2)`	Simulation d'un vecteur de 10 réalisations indépendantes de $\mathcal{B}(20, \frac{1}{2})$.
`dnorm(1,0,2)`	Densité de la loi normale $\mathcal{N}(0,2)$ évaluée en 1.
`pnorm(1,0,2)`	Fonction de répartition de la loi normale $\mathcal{N}(0,2)$ évaluée en 1.
`qnorm(0.9,0,2)`	Quantile d'ordre 0.9 de $\mathcal{N}(0,2)$.
`rnorm(10,0,2)`	Simulation d'un vecteur de 10 réalisations indépendantes de $\mathcal{N}(0,2)$.

6.2 TIRAGES AVEC OU SANS REMISE

`sample(1:10,100,replace=TRUE)`	Tirage avec remise de 100 éléments parmi les nombres $1, \ldots, 10$. L'option `replace=TRUE` précise que le tirage se fait avec remise. Par défaut, si rien n'est précisé, `replace=FALSE`.
`sample(1:10); sample(10)`	Commandes équivalentes désignant une permutation des entiers naturels 1, 2, ..., 10.

`sample(c("A","B","C"),size=5, replace=TRUE,prob=c(0.2,0.5,0.3))`	Tirage avec remise de 5 lettres parmi les lettres A, B et C. Si l'on ne veut pas un tirage équiprobable, on précise les probabilités de tirage dans le vecteur `prob`.

6.3 Réplication d'une commande

`replicate(n,expr)`	Exécute `n` fois le code `expr` et stocke les résultats sous forme de vecteur (si `expr` renvoie un nombre) et sous forme de matrice (si `expr` renvoie un vecteur).

La fonction `replicate` permet d'éviter l'usage de boucles. Par exemple, le code suivant simule 5 fois la génération de 1 000 valeurs suivant la loi normale centrée réduite et calcule la moyenne à chaque fois.

```
replicate(5,mean(rnorm(1000)))
```

```
[1] -0.01514964 -0.04786793 -0.06151126 -0.07324169  0.01370721
```

6.4 EXERCICES

Exercice 6.1 Soit X une variable aléatoire de loi binomiale de paramètres $n = 50$ et $p = 1/3$. Calculer les probabilités suivantes :

$$\mathbb{P}(X = 1) \quad ; \quad \mathbb{P}(X \leqslant 5) \quad ; \quad \mathbb{P}(X \geqslant 15) \quad ; \quad \mathbb{P}(X \notin \{15, 3, 4, 10\}) \quad ; \quad \mathbb{P}(X \in 2\mathbb{N}).$$

Exercice 6.2 Soit X une variable aléatoire de loi normale standard $\mathcal{N}(0, 1)$.

1) Calculer les probabilités suivantes : $\mathbb{P}(X \leqslant 1)$, $\mathbb{P}(X \geqslant 2.6)$ et $\mathbb{P}(0.5 < X \leqslant 1)$.
2) Calculer les quantiles d'ordre 0.95 et 0.975.
3) Représenter graphiquement la densité et la fonction de répartition de la loi de X.

Exercice 6.3 Soit X une variable suivant la loi normale $\mathcal{N}(3, 2)$. Calculer les probabilités suivantes : $\mathbb{P}(X < 4)$, $\mathbb{P}(X < -1)$, $\mathbb{P}(X > 1)$, $\mathbb{P}(-3 < X < 4)$.

Exercice 6.4

1) Créer un vecteur x de taille $n = 1000$ et dont les éléments suivent la loi normale centrée réduite.
2) Représenter l'histogramme des éléments de ce vecteur et superposer sur ce graphique la fonction de densité de la loi normale centrée réduite. Que remarquez-vous ?

Exercice 6.5

1) En vous aidant de la fonction `sample` et du vecteur `piece=c("F","P")` (dont les composantes représentent FACE et PILE), écrire une fonction `lancer.piece(n)` qui simule n lancers d'une pièce de monnaie équilibrée.
2) Simuler $n = 5\,000$ lancers de ladite pièce. Calculer la proportion de `F` (vous devriez trouver une valeur proche de 0.5).

Exercice 6.6 Un ami vous propose le jeu suivant. Vous lancez deux (2) fois un dé supposé parfaitement équilibré dont les faces sont numérotées de 1 à 6. A chaque lancer, si le résultat est 5 ou 6, vous gagnez 3 € ; si le résultat est 4, vous gagnez 1 € et si c'est 3 ou moins, vous perdez 2.5 €.

1) Ecrire une fonction `jeu` permettant de simuler le jeu.
2) Simuler ce jeu 1 000 fois. Résumer les résultats à l'aide d'un tableau de contingence et représenter ce tableau à l'aide d'un diagramme en bâtons.
3) Ce jeu serait-il avantageux pour vous ? Justifier la réponse.

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] Bertrand, F. and Maumy-Bertrand, M. (2018). *Initiation à la Statistique avec R*. Dunod, Paris, 3ème edition.

[2] Biernacki, C. (2010). Logiciel R - Tutorial avec applications statistiques. Notes de cours, Université Lille 1.

[3] Broc, G. (2016). *Stats faciles avec R*. De Boeck Superieur.

[4] Lafaye de Micheaux, P., Drouilhet, R., and Liquet, B. (2014). *Le logiciel R : Maîtriser le langage - Effectuer des analyses statistiques*. Springer, 2ème edition.

[5] Paradis, E. (2005). R pour les débutants. URL http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf.

[6] Venables, W. N., Smith, D. M., and the R Core Team (2014). *An Introduction to R*. Notes on R : A Programming Environment for Data Analysis and Graphics Version 3.0.3 (2014-03-06).