Prise en main du logiciel R

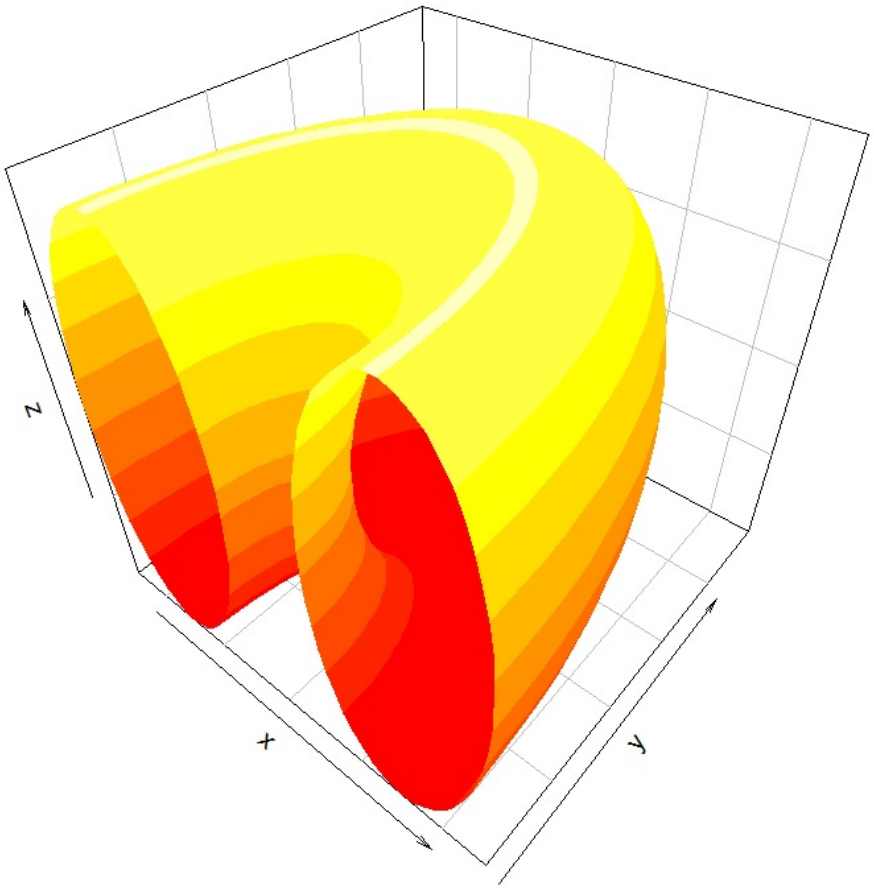
Christophe Chesneau

Université de Caen

Prise en main du logiciel 

Christophe Chesneau

http://www.math.unicaen.fr/~chesneau/



Caen, le 16 Octobre 2016

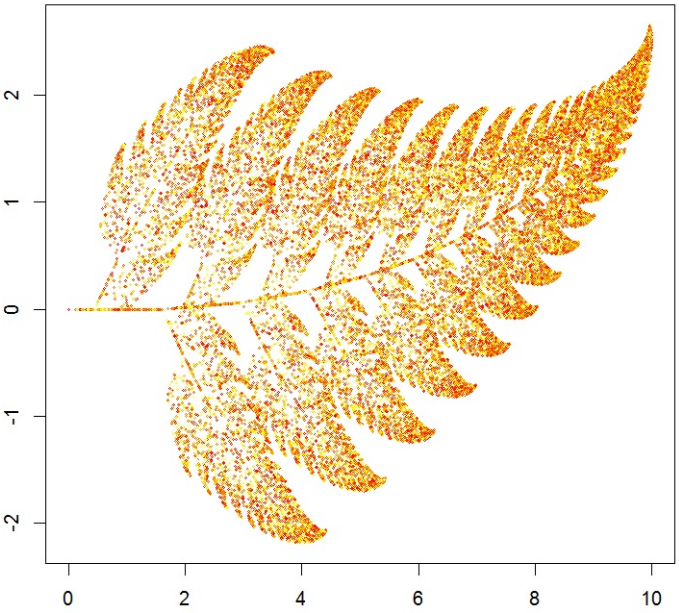
Table des matières

1 Présentation

Pourquoi s'intéresser au logiciel R ?

Parce que :

- Il permet de faire des analyses statistiques avancées.
- Il permet de faire de beaux graphiques :



- Il commence à être un logiciel de référence dans le monde du travail.
- Il est simple d'utilisation.
- Il est gratuit.

L'objectif de ce document est de présenter les objets élémentaires du logiciel R (scalaires, vecteurs, matrices, listes...) et quelques opérations associées. Le format "exemple - explication" est adopté.

Pour tout commentaire, merci de me contacter :

`christophe.chesneau@gmail.com`

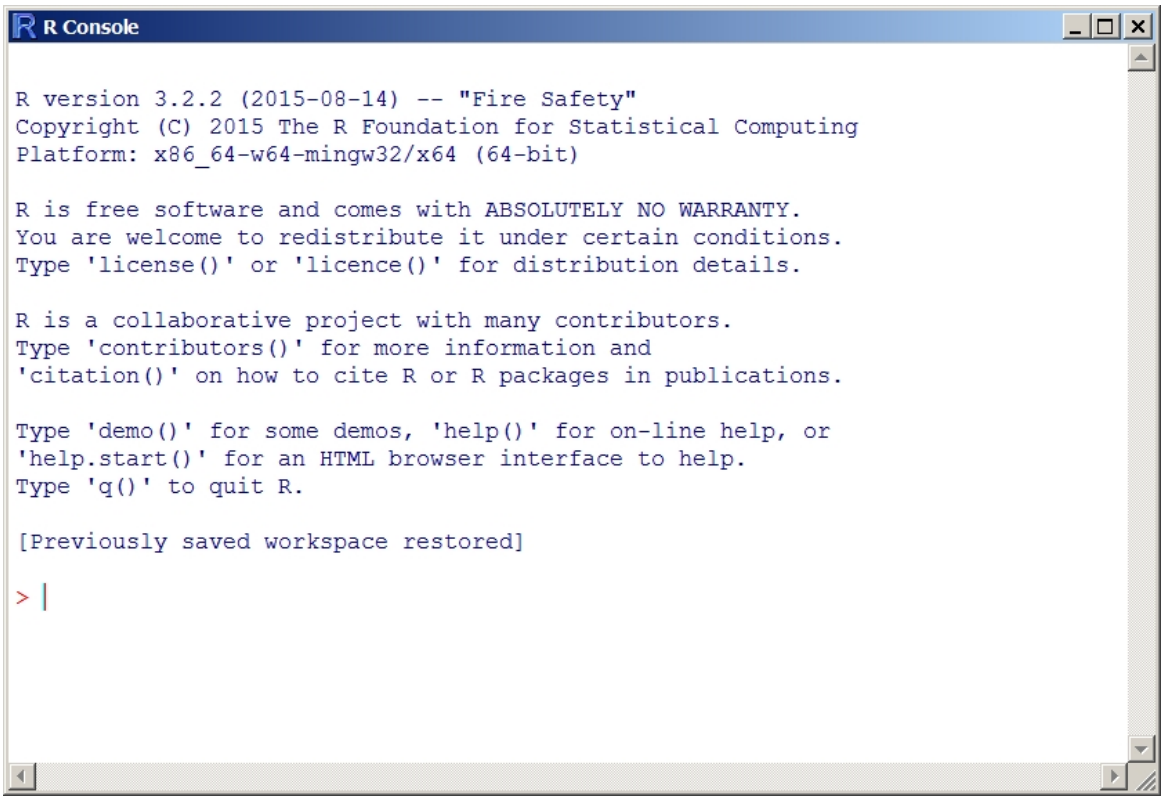
Bonne lecture !

2 Fenêtre "R Console"

Installer la dernière version du logiciel R en fonction de votre système d'exploitation (Windows, Linux, Mac...) :

`https://cran.r-project.org/`

La version utilisée dans ce document est installée sous Windows. Ensuite, double-cliquer sur le logo R. Une fenêtre "R Console" s'ouvre alors :

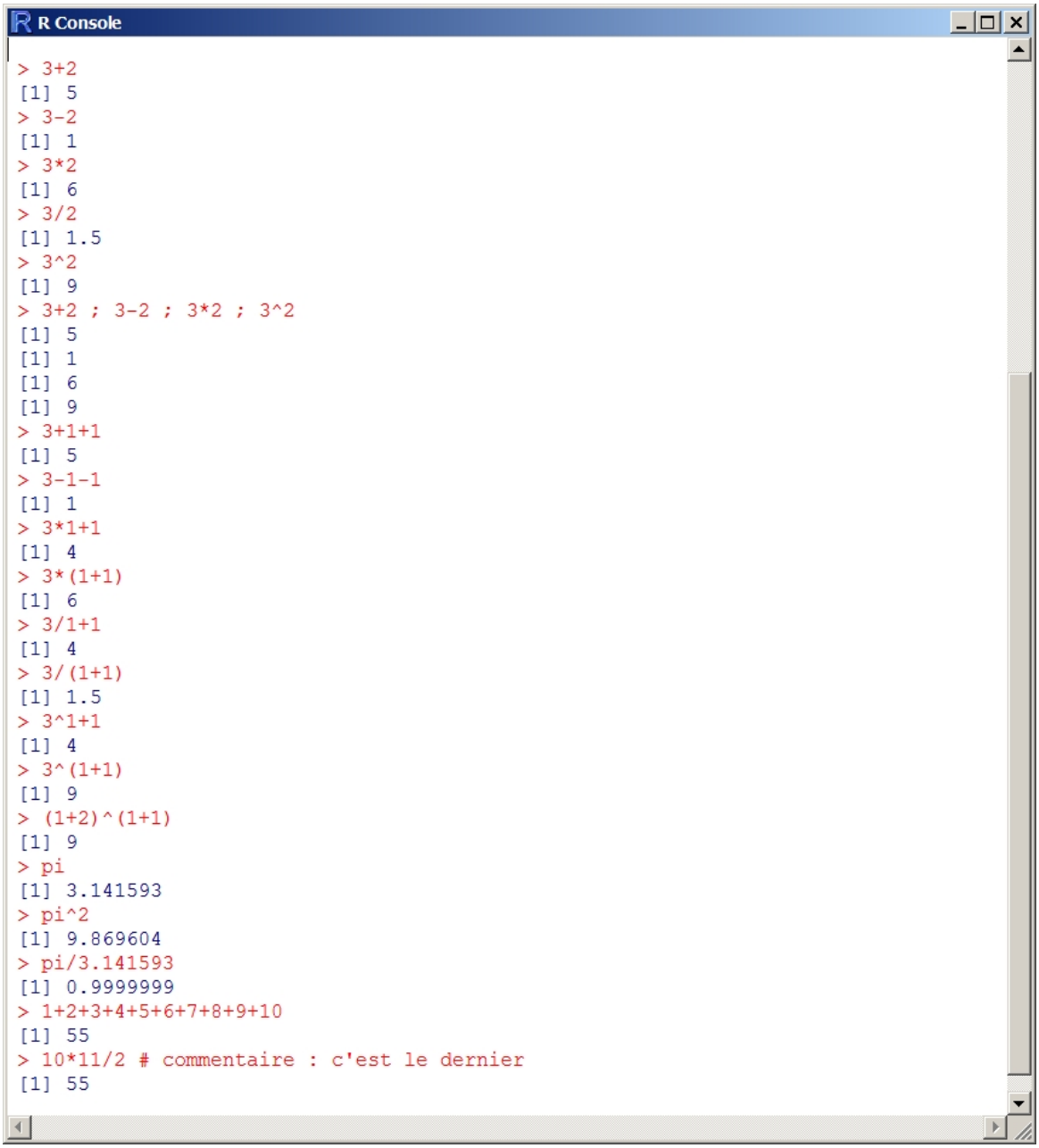


On écrit les commandes désirées après le >, lesquelles s'affiche en rouge, on les exécute en appuyant sur la touche Entrée. Les sorties s'affichent en bleu.

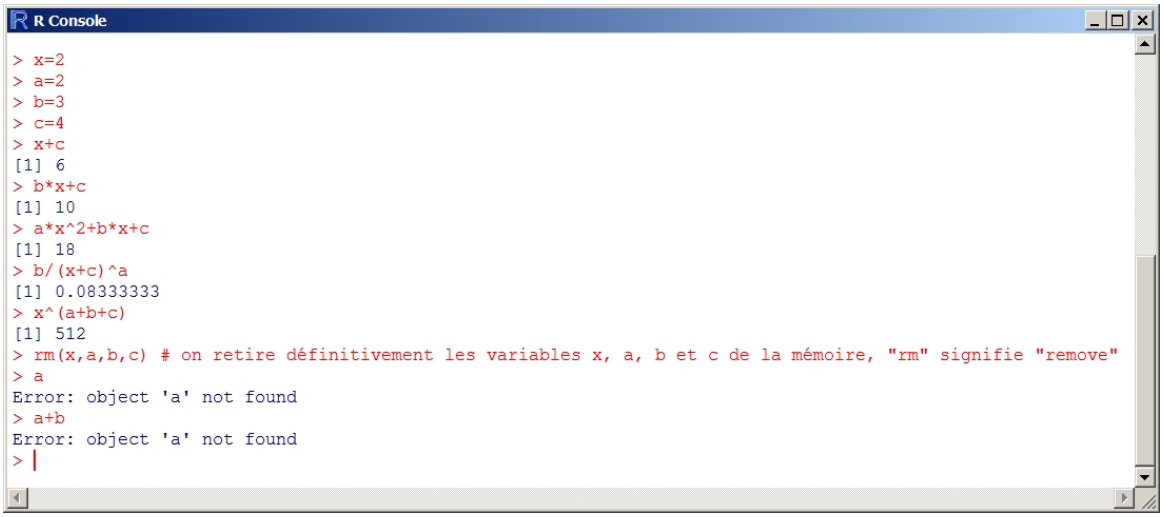
Pour fermer sèchement la fenêtre "R Console", écrire : `q("no")` (puis appuyer sur Entrée).

3 Opérations de base (%traîner)

On peut utiliser R pour faire des opérations algébriques élémentaires, comme on le fait avec la calculette. Recopier pas à pas les commandes R (en rouge) présentées dans la fenêtre "R Console" ci-dessous (*si le symbole* + *apparaît après une éxécution prématurée, appuyer sur la touche* `Échap`*).*



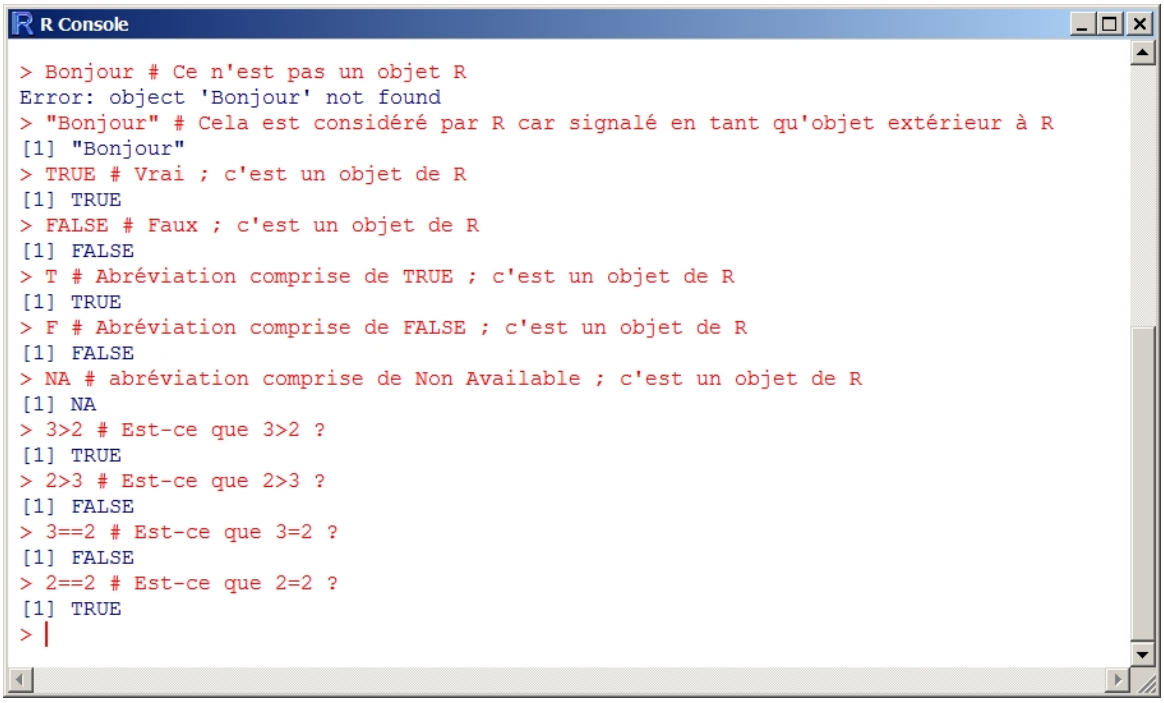
On peut travailler directement avec des variables (scalaires) que l'on met en mémoire dès le début. Recopier pas à pas les commandes R présentées dans la fenêtre "R Console" ci-dessous.



Pour compléter cette première approche, précisons que :

- tout ce qui n'est pas un objet de R doit être mis entre guillemets,
- on peut faire des tests logiques.

Recopier pas à pas les commandes R présentées dans la fenêtre "R Console" ci-dessous.



4 Création de vecteurs

R : un langage vectoriel

Un vecteur dans R est un objet fourre tout ; c'est une collection de données et non un vecteur au sens mathématique du terme. Le langage R est centré autour du fait qu'un vecteur est un ensemble ordonné de mesures plutôt qu'une position géométrique ou un état physique. Mêmes si R est en mesure de faire des opérations vectorielles mathématiques classiques, celles-ci sont secondaires dans la conception de la langue. À partir de tels objets, d'autres objets plus sophistiqués sont construits (matrice, liste, data frame ...). Il est donc crucial de bien maitriser cette notion de "R vecteur" pour bien commencer.

Dorénavant, il est conseillé de recopier une à une, et dans l'ordre, les commandes R à venir dans la fenêtre "R Console". La plupart des objets définis seront utilisés pour en définir d'autres.

Création de vecteurs numériques

On considère les commandes :

```
vec1 = c(2.8, 2.4, 2.1, 3.6, 2.8)

vec1
```

Cela renvoie : `[1] 2.8 2.4 2.1 3.6 2.8`

On a ainsi créé le vecteur ligne `vec1` en concaténant les éléments : `2.8`, `2.4`, `2.1`, `3.6` et `2.8`.

Le `[1]` signifie que `vec1` est défini sur la 1-ère ligne.

Création de vecteurs chaînes de caractères

On fait :

```
vec2 = c("rouge", "vert", "vert", "vert", "jaune")

vec2
```

Cela renvoie : `[1] "rouge" "vert" "vert" "vert" "jaune"`

Création de vecteurs logiques

On considère les commandes :

```
vec3 = c(TRUE, TRUE, FALSE, FALSE, FALSE)

vec3
```

Cela renvoie : `[1] TRUE TRUE FALSE FALSE FALSE`

Création par répétition

On fait :

```
rep(4, 3)
```

Cela renvoie : `[1] 4 4 4`

Ainsi, on a répété le chiffre `4` trois fois.

On considère les commandes :

```
vec4 = rep(vec1, 2)

vec4
```

Cela renvoie : `[1] 2.8 2.4 2.1 3.6 2.8 2.8 2.4 2.1 3.6 2.8`

On a alors créé un vecteur `vec4` en concaténant le vecteur `vec1` avec lui-même.

On fait :

```
vec5 = rep(vec1, c(2, 1, 3, 3, 2))

vec5
```

Cela renvoie : `[1] 2.8 2.8 2.4 2.1 2.1 2.1 3.6 3.6 3.6 2.8 2.8`

On a ainsi créé un vecteur `vec5` en répétant certains éléments de `vec1`. Ainsi, le i-ème élément de `vec1` est répété autant de fois que l'indique le i-ème élément du second vecteur.

Création par construction de suites

On considère les commandes :

```
vec6 = 1:10

vec6
```

Cela renvoie : `[1] 1 2 3 4 5 6 7 8 9 10`

Ainsi, on a créé un vecteur dont les éléments sont les 10 premiers entiers positifs.

On fait :

```
vec7 = seq(from = 3, to = 5, by = 0.2)

vec7
```

Cela renvoie : `[1] 3.0 3.2 3.4 3.6 3.8 4.0 4.2 4.4 4.6 4.8 5.0`

On a alors créé un vecteur `vect7` dont les éléments sont les valeurs $3 + 0.2 \times i$ pour tous les entiers i tels que $3 + 0.2 \times i \leq 5$: 3.0 3.2 `3.4`, `3.6`, `3.8`, `4.0`, `4.2`, `4.4`, `4.6`, `4.8` et `5.0`.

On aurait aussi pu construire `vec7` en faisant :

```
vec7 = seq(from = 3, length = 11, by = 0.2)
```

Création élément par élément

On considère les commandes :

```
vec8 = numeric()

vec8[1] = 41.8

vec8[2] = -0.3

vec8[3] = 92

vec8
```

Cela renvoie : `[1] 41.8 -0.3 92`

Ainsi, on a créé un vecteur `vec8` en introduisant les 3 éléments : `41.8`, `-0.3` et `92`.

On peut faire la même chose avec un vecteur de chaînes de caractères :

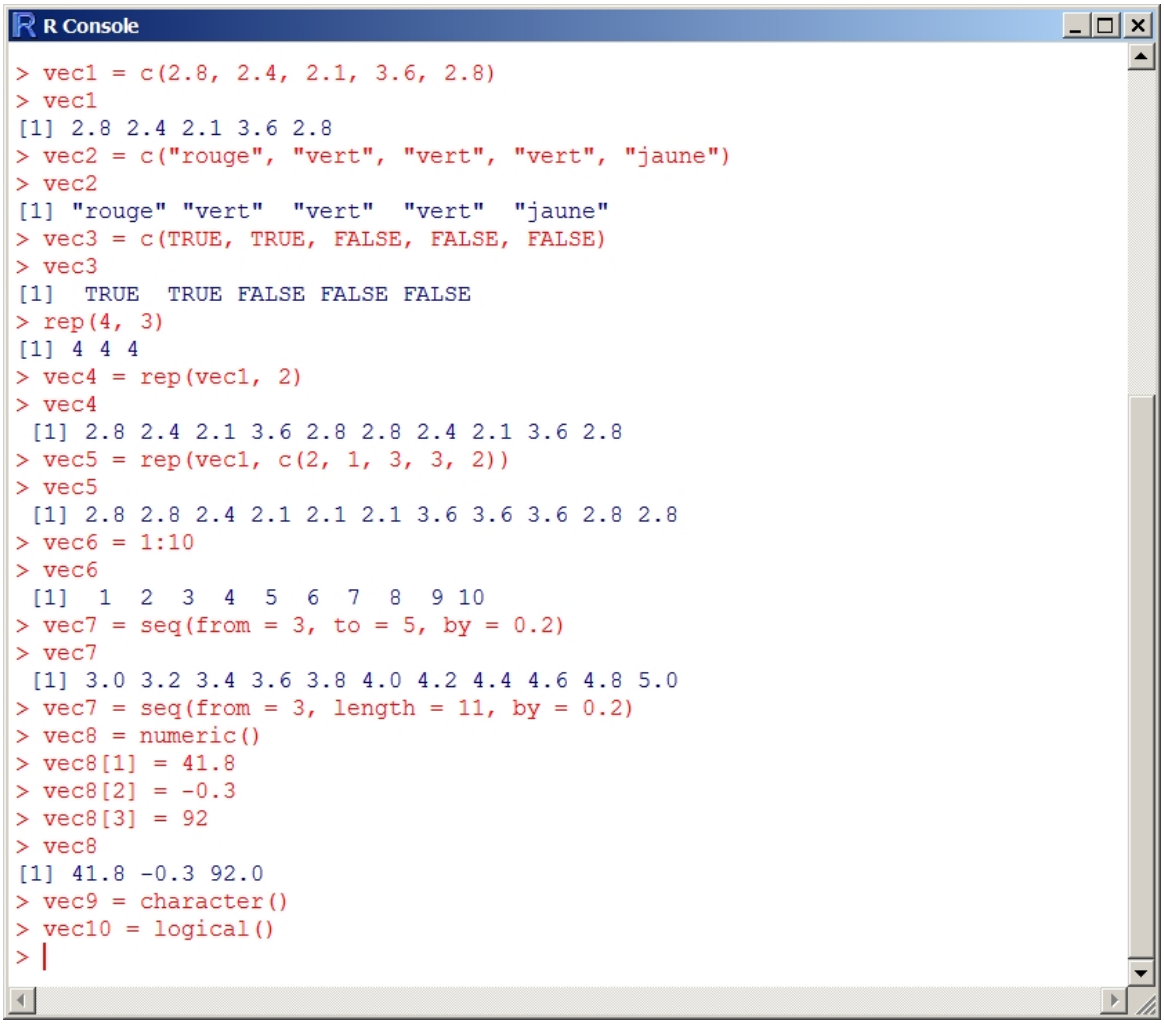
```
vec9 = character()
```

On peut faire de même avec un vecteur logique :

```
vec10 = logical()
```

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous (l'indentation a été respectée uniquement pour un meilleur rendu visuel).



5 Création de matrices

Méthode générique

On considère les commandes :

```
mat1 = matrix(vec4, ncol = 5)

mat1
```

Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.8  2.1  2.8  2.4  3.6
[2,]  2.4  3.6  2.8  2.1  2.8
```

On a ainsi transformé le vecteur ligne `vec4` en une matrice à 5 colonnes (et, par conséquent, 2 lignes). Par défaut, les éléments de `vec4` sont rentrés par colonnes.

Si on veut rentrer les éléments de `vec4` par lignes, on fait :

```
mat2 = matrix(vec4, ncol = 5, byrow = T)

mat2
```

Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.8  2.4  2.1  3.6  2.8
[2,]  2.8  2.4  2.1  3.6  2.8
```

Autre méthode

On considère les commandes :

```
mat3 = cbind(vec1, 3:7)

mat3
```

Cela renvoie :

```
     [,1] [,2]
[1,]  2.8    3
[2,]  2.4    4
[3,]  2.1    5
[4,]  3.6    6
[5,]  2.8    7
```

Ainsi, la commande `cbind` concatène des colonnes de vecteurs de même longueur. On peut faire de même en concaténant des lignes avec la commande `rbind`.

On fait :

```
mat4 = diag(c(2, 1, 5))
mat4
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    1    0
[3,]    0    0    5
```

On a alors créé une matrice diagonale `mat4` d'éléments diagonaux : `2`, `1` et `5`.

Extraire une colonne ou une ligne

On considère les commandes :

```
mat1[, 3]
```

Cela renvoie : `[1] 2.8 2.8`

On obtient alors un vecteur ligne dont les éléments sont ceux de la 3-ème colonne de `mat1`.

On fait :

```
mat1[2, ]
```

Cela renvoie : `[1] 2.4 3.6 2.8 2.1 2.8`

Ainsi, on obtient un vecteur ligne dont les éléments sont ceux de la 2-ème ligne de `mat1`.

Extraire une sous-matrice

On considère les commandes :

```
mat1[, c(2, 4, 5)]
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]  2.1  2.4  3.6
[2,]  3.6  2.1  2.8
```

On obtient ainsi une matrice construite avec les 2-ème, 4-ème et 5-ème colonnes de `mat1`.

On aurait aussi pu construire cette matrice en faisant :

```
mat1[, c(FALSE, TRUE, FALSE, TRUE, TRUE)]
```

On considère les commandes :

```
mat3[c(1, 4), ]
```

Cela renvoie :

```
     [,1] [,2]
[1,]  2.8    3
[2,]  3.6    6
```

Ainsi, cette matrice est construite avec les 1-ère et 4-ème lignes de `mat3`.

Calculer les dimensions d'une matrice

On considère les commandes :

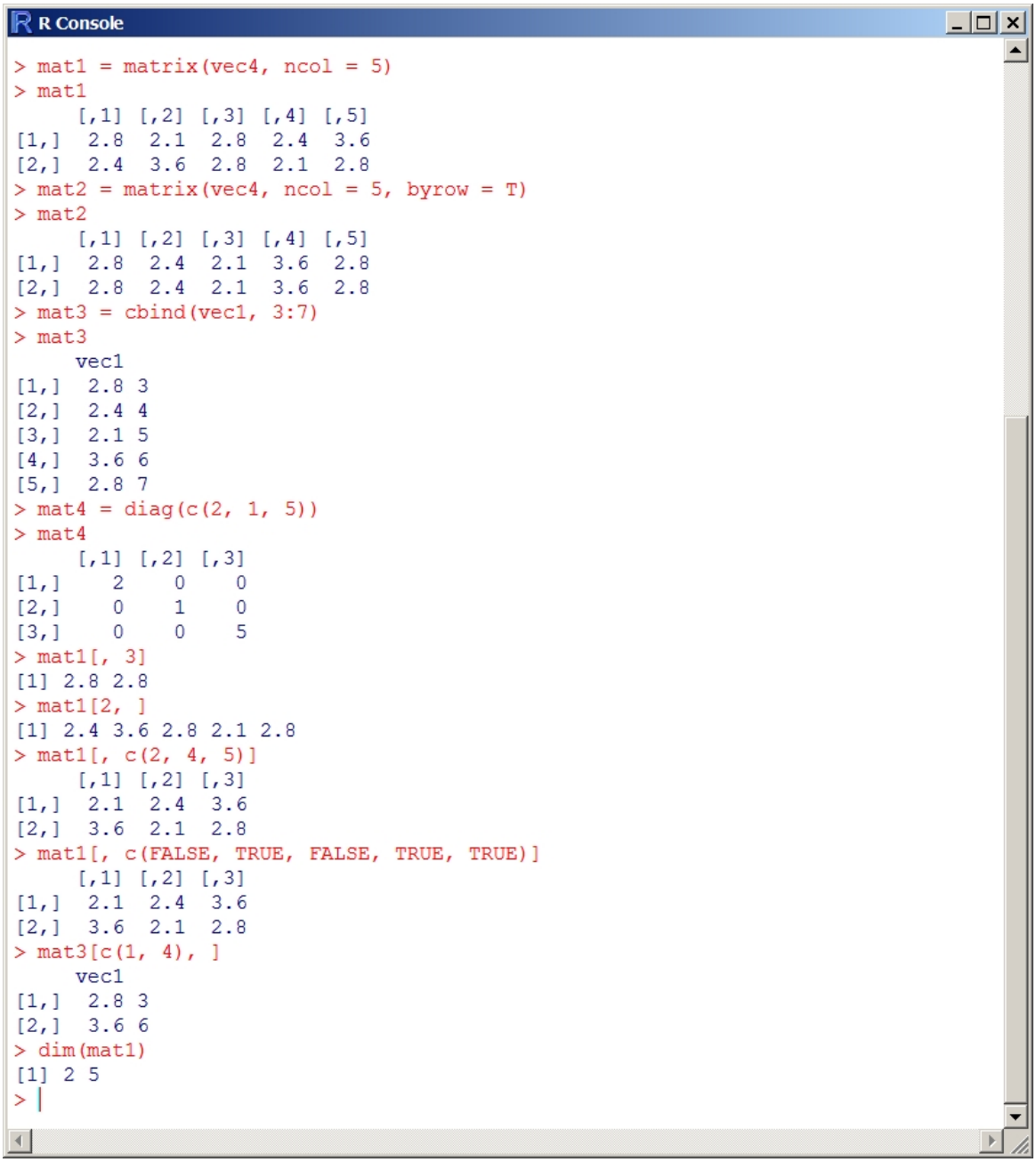
```
dim(mat1)
```

Cela renvoie : `[1] 2 5`

Ainsi, sans surprise, la matrice `mat1` a 2 lignes et 5 colonnes.

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



6 Création de listes

Méthode directe

On considère les commandes :

```
list1 = list(vec1, c("rouge", "bleu"), mat1)

list1
```

Cela renvoie :

```
[[1]]

[1] 2.8 2.4 2.1 3.6 2.8


[[2]]

[1] "rouge" "bleu"


[[3]]

     [,1] [,2] [,3] [,4] [,5]

[1,]  2.8  2.1  2.8  2.4  3.6

[2,]  2.4  3.6  2.8  2.1  2.8
```

Ainsi, la liste `list1` est composée de plusieurs objets : le vecteur numérique `vec1`, le vecteur chaînes de caractères `c("rouge", "bleu")` et la matrice `mat1`.

Création élément par élément

On considère les commandes :

```
list2 = list()

list2[[1]] = vec1

list2[[2]] = c("rouge", "bleu", "vert")

list2[[3]] = mat2

list2
```

Cela renvoie :

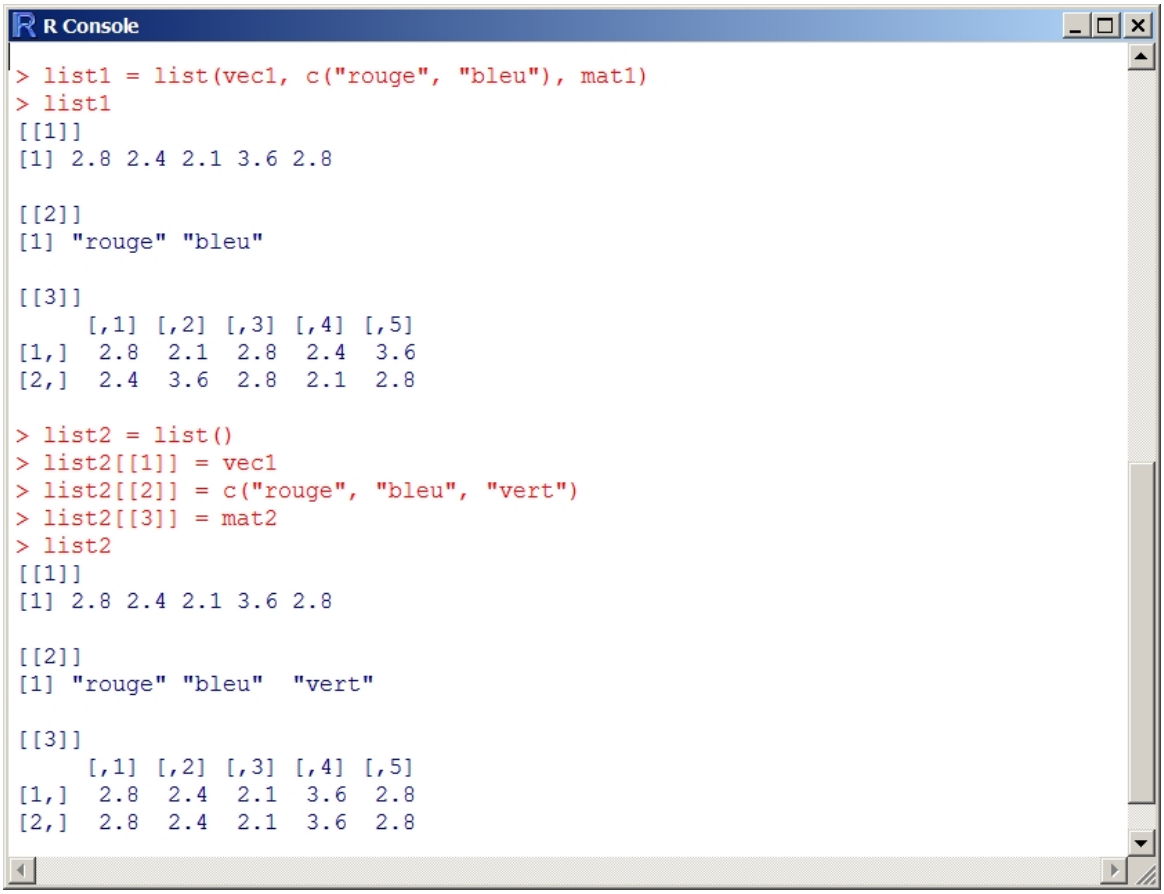
```
[[1]]
[1] 2.8 2.4 2.1 3.6 2.8


[[2]]
[1] "rouge" "bleu"  "vert"


[[3]]
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.8  2.4  2.1  3.6  2.8
[2,]  2.8  2.4  2.1  3.6  2.8
```

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



7 Opérations sur les vecteurs

Opérations élémentaires

Copier. On considère les commandes :

```
vec11 = vec1

vec13 = vec12 = vec11
```

On a ainsi copié `vect1` dans les vecteurs `vect11`, `vect12` et `vect13`.

Concaténer. On fait :

```
c(vec1, c(3.9, 2.7))
```

Cela renvoie : `[1] 2.8 2.4 2.1 3.6 2.8 3.9 2.7`

On a donc rajouté les éléments : `3.9` et `2.7`, à la fin de `vect1`.

On considère les commandes :

```
c("blanc", vec2)
```

Cela renvoie : `[1] "blanc" "rouge" "vert" "vert" "vert" "jaune"`

Ainsi, on a rajouté l'élément `"blanc"` au début de `vect2`.

Extraire. On considère les commandes :

```
vec1[2]
```

Cela renvoie : `2.4`

On a alors extrait le 2-ème élément de `vect1`.

On fait :

```
vec1[c(2, 4)]
```

Cela renvoie : `2.4 3.6`

On a ainsi extrait les 2-ème et 4-ème éléments de `vect1`.

On considère les commandes :

```
vec1[2:4]
```

Cela renvoie : `2.4 2.1 3.6`

On a alors extrait les 2-ème, 3-ème et 4-ème éléments de `vect1`.

On considère les commandes :

```
vec1[vec3]
```

Cela renvoie : `2.8 2.4`

On a ainsi extrait le 1-er et 2-ème éléments de `vect1` dont les indices correspondent aux éléments `T` de `vec3`.

Raccourcir. On fait :

```
vec11[-2]
```

Cela renvoie : `2.8 2.1 3.6 2.8`

On obtient ainsi le vecteur `vect1` privé de son 2-ème élément.

On considère les commandes :

```
vec12[-c(2, 4)]
```

Cela renvoie : `2.8 2.1 2.8`

On obtient ainsi `vect12` privé de ses 2-ème et 4-ème éléments.

On considère les commandes :

```
vec13[-(2:4)]
```

Cela renvoie : `2.8 2.8`

Ainsi, on a `vect13` privé de ses 2-ème, 3-ème et 4-ème éléments.

Remplacer. On fait :

```
vec5[3:5] = c(1034, 238, -99)

vec5
```

Cela renvoie : `[1] 2.8 2.8 1034.0 238.0 -99.0 2.1 3.6 3.6 3.6 2.8 2.8`

Ainsi, on a remplacé les 3-ème, 4-ème et 5-ème éléments de `vect5` par les éléments : `1034`, `238` et `-99`.

Attribuer des labels. On considère les commandes :

```
names(vec1) = c("julie", "paul", "solveigh", "valentin", "elsa")

vec1
```

Cela renvoie :

```
[1] julie paul solveigh valentin elsa

      2.8  2.4    2.1      3.6   2.8
```

On peut alors extraire le 2-ème élément de `vec1` en faisant :

```
vec1["paul"]
```

On peut enlever les labels attribués en faisant :

```
names(vec1) = c()
```

Connaître la nature. On considère les commandes :

```
mode(vec5)
```

Cela renvoie : `[1] "numeric"`

Une autre solution est :

```
is.numeric(vec5)
```

Cela renvoie : `[1] T`

Changer la nature. On considère les commandes :

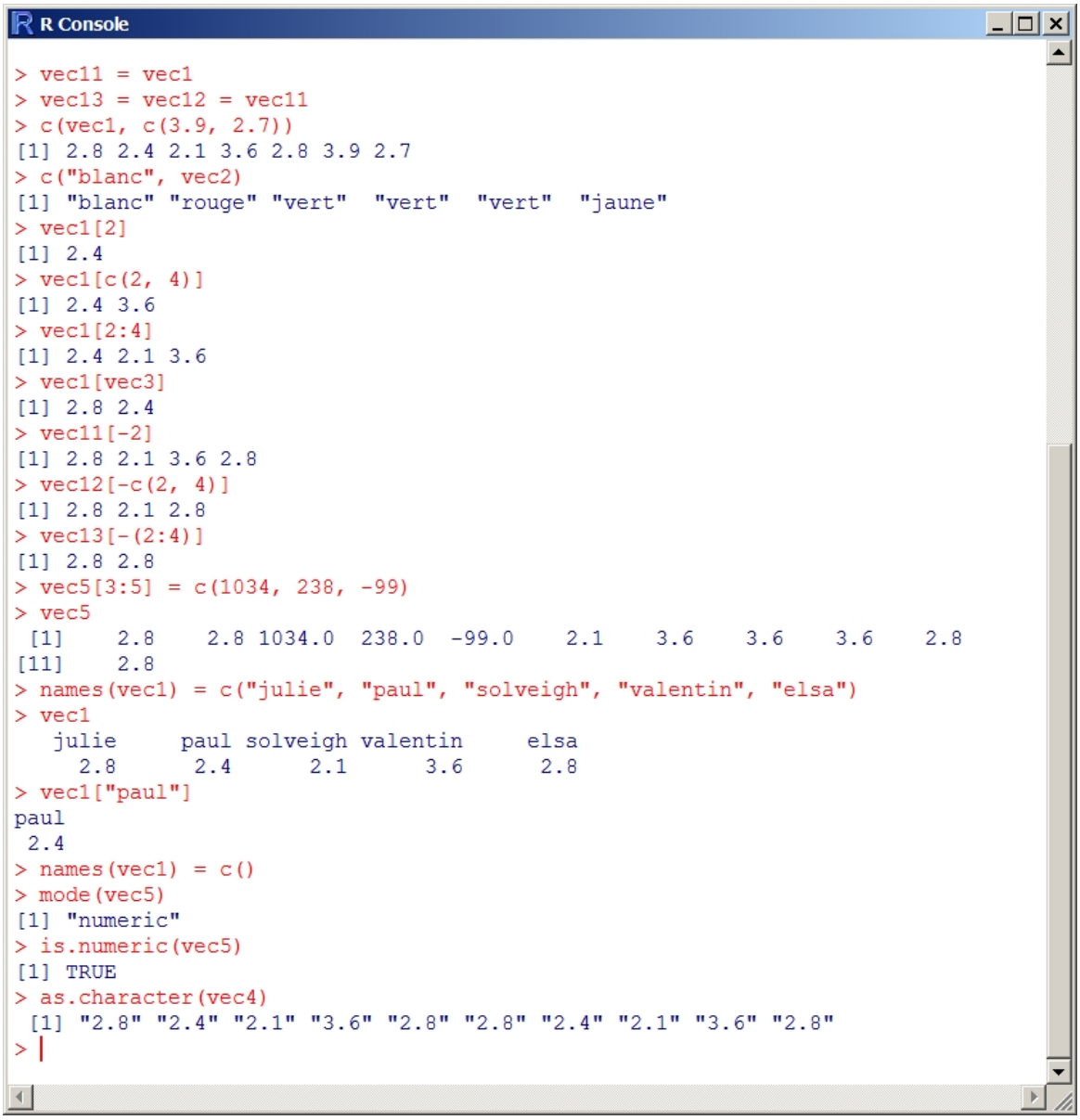
```
as.character(vec4)
```

Cela renvoie : `[1] "2.8" "2.4" "2.1" "3.6" "2.8" " 2.8" "2.4" "2.1" "3.6" "2.8"`

On a alors changé un vecteur numérique en vecteur de chaînes de caractères.

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



Opérations arithmétiques sur des vecteurs numériques

Opérations élémentaires. On considère les commandes :

```
vec1 + 4
```

Cela renvoie : `[1] 6.8 6.4 6.1 7.6 6.8`

On obtient ainsi un vecteur dont le i-ème éléments est la valeur du i-ème élément de `vec1` plus 4.

On fait :

```
3 * vec1 + 1:5
```

Cela renvoie : `[1] 9.4 9.2 9.3 14.8 13.4`

On considère les commandes :

```
2 * vec1 - c(1, 2)
```

Cela renvoie : `[1] 4.6 2.8 3.2 5.2 4.6`

Ainsi, quand deux vecteurs n'ont pas la même longueur, le plus court est répliqué jusqu'à ce qu'il ait atteint la longueur du plus grand.

Réarrangement des éléments

On considère les commandes :

```
rev(vec1)
```

Cela renvoie : `[1] 2.8 3.6 2.1 2.4 2.8`

On obtient ainsi un vecteur dont l'ordre des éléments de `vec1` est renversé.

On fait :

```
sort(vec1)
```

Cela renvoie : `[1] 2.1 2.4 2.8 2.8 3.6`

Ainsi, on a trié les valeurs des éléments de `vec1` par ordre croissant.

On considère les commandes :

```
order(vec1)
```

Cela renvoie : `[1] 3 2 1 5 4`

On donne alors les indices des éléments de `vec1` par ordre de valeurs croissant.

Application d'une fonction : vecteur longueur $k \longrightarrow$ vecteur longueur k

Pour les fonctions : `abs`, `log`, `log10`, `sqrt`, `exp`, `sin`, `cos`, `tan`, `acos`, `atan`, `asin`, `cosh`, `sinh`, `tanh`, `gamma`, `lgamma` et `round` (et bien d'autres), l'application à un vecteur donne un vecteur de même longueur, dont la valeur du i-ème élément est le résultat de la fonction pour la valeur du i-ème élément du vecteur initial. On fait :

```
sin(vec1)
```

Cela renvoie : `[1] 0.3349882 0.6754632 0.8632094 -0.4425204 0.3349882`

On fait :

```
log(vec1) + 3
```

Cela renvoie : `[1] 4.029619 3.875469 3.741937 4.280934 4.029619`

On considère les commandes :

```
round(c(9.238, -1.34222))
```

Cela renvoie : `[1] 9 -1`

On obtient alors les valeurs arrondies des éléments du vecteur : `c(9.238, -1.34222)`.

Application d'une fonction : vecteur longueur $k \longrightarrow$ une valeur numérique

Pour les fonctions : `length`, `sum`, `cumsum`, `prod`, `cumprod`, `min`, `max`, `which.min`, `which.max`, `mean`, `var` et `sd` (et bien d'autres), l'application à un vecteur donne une valeur numérique.

On considère les commandes :

```
length(vec5)
```

Cela renvoie : `[1] 11`

Ainsi, le vecteur `vec5` est de taille 11 (il a 11 éléments).

On fait :

```
which.min(vec8)
```

Cela renvoie : `[1] 2`

Ainsi, la plus petite valeur de `vec8` se situe au 2-ème élèment (c'est `-0.3`).

On fait :

```
mean(vec1)
```

Cela renvoie : `[1] 2.74`

On obtient alors la moyenne des valeurs des éléments de `vec1`.

On considère les commandes :

```
sd(vec1)
```

Cela renvoie : `[1] 0.5639149`

On obtient alors l'écart-type corrigé des valeurs des éléments de `vec1` de formule générale :

$$\text{“} \quad s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2} \quad \text{”}.$$

On peut vérifier cette formule avec les commandes :

```
sqrt((1 / (length(vec1) - 1)) * sum((vec1 - mean(vec1))^2))
```

Cela renvoie : `[1] 0.5639149`

On fait :

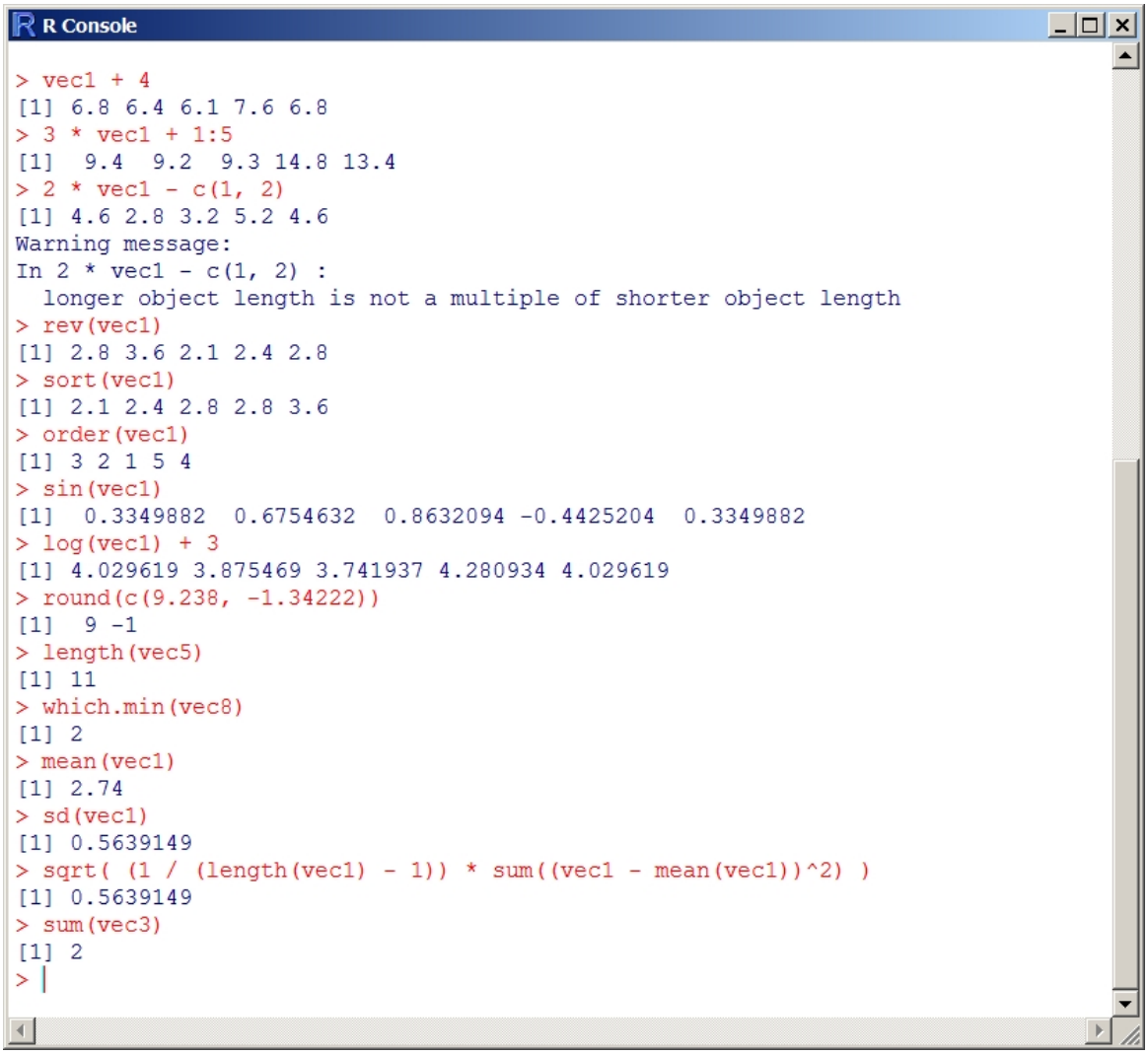
```
sum(vec3)
```

Cela renvoie : `[1] 2`

Ainsi, les éléments du vecteur logique `vec3` ont été codés numériquement avec `0` pour `F` et `1` pour `T`, et on a fait la somme de ces valeurs binaires.

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



Opérations sur les vecteurs chaînes de caractères

Concaténer deux chaînes. On fait :

```
paste("jules", "jim")
```

Cela renvoie : `[1] "jules jim"`

On considère les commandes :

```
paste("jules", "jim", sep = "")
```

Cela renvoie : `[1] "julesjim"`

Ainsi, le séparateur par défaut est un espace.

Concaténer deux (ou plus) vecteurs de chaînes de caractères. On considère les commandes :

```
paste("X", 1:4, sep = "")
```

Cela renvoie : `[1] "X1" "X2" "X3" "X4"`

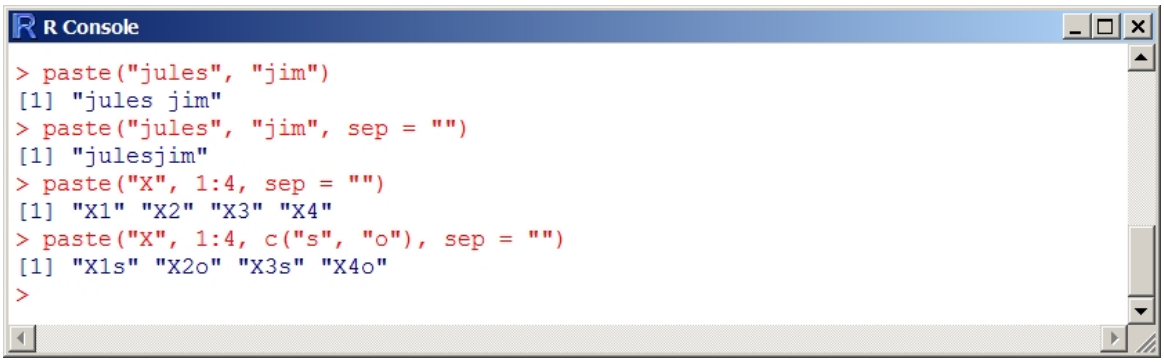
On fait :

```
paste("X", 1:4, c("s", "o"), sep = "")
```

Cela renvoie : `[1] "X1s" "X2o" "X3s" "X4o"`

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



Opération sur les vecteurs logiques

Opérateurs de comparaison. On utilise les opérateurs de comparaison suivants :

- égal `==`,
- strictement plus petit `<`,
- strictement plus grand `>`,
- plus petit ou égal `<=`,
- plus grand ou égal `>=`,
- différent `!=` .

On considère les commandes :

```
a = c(1, 4, -2, 5)
b = c(2, 4, -3, 6)
a < b
```

Cela renvoie : `[1] TRUE FALSE FALSE TRUE`

Ainsi, quand deux vecteurs a et b ont même longueur, le résultat de la comparaison est un vecteur logique de même longueur que a et b et dont le i-ème élément est le résultat de la comparaison des i-ème éléments de a et b.

On fait :

```
a = c(1, 4, -2, 5, 6, 7, -1)
b = c(1, 4, -3, 5, 6, 7, -2)
a == b
```

Cela renvoie : `[1] TRUE TRUE FALSE TRUE TRUE TRUE FALSE`

On considère les commandes :

```
vec1 > 2.7
```

Cela renvoie : `[1] TRUE FALSE FALSE TRUE TRUE`

En réalité, le logiciel R a effectué : `vec1 > c(2.7, 2.7, 2.7, 2.7, 2.7)`.

On considère les commandes :

```
a = c(3, 2, 8, 3, -3)

b = c(1, 2)

a <= b
```

Cela renvoie : `[1] FALSE TRUE FALSE FALSE TRUE`

En réalité, le logiciel R a effectué : `a <= c(1, 2, 1, 2, 1)`.

Ainsi, quand un vecteur a est de plus grande longueur qu'un vecteur b, lorsqu'on les compare, b est répliqué en un vecteur b^* jusqu'à ce qu'il atteigne la longueur de a. Le résultat de la comparaison est un vecteur logique de même longueur que a et dont le i-ème élément est le résultat de la comparaison des i-ème éléments de a et b^*.

Ceci permet de construire un vecteur où seuls les éléments vérifient la relation de comparaison :

```
vec1[vec1 > 2.7]
```

Cela renvoie : `[1] 2.8 3.6 2.8`

On obtient ainsi un vecteur ayant pour éléments ceux du vecteur `vect1` dont les valeurs sont strictement supérieures à `2.7`.

On fait :

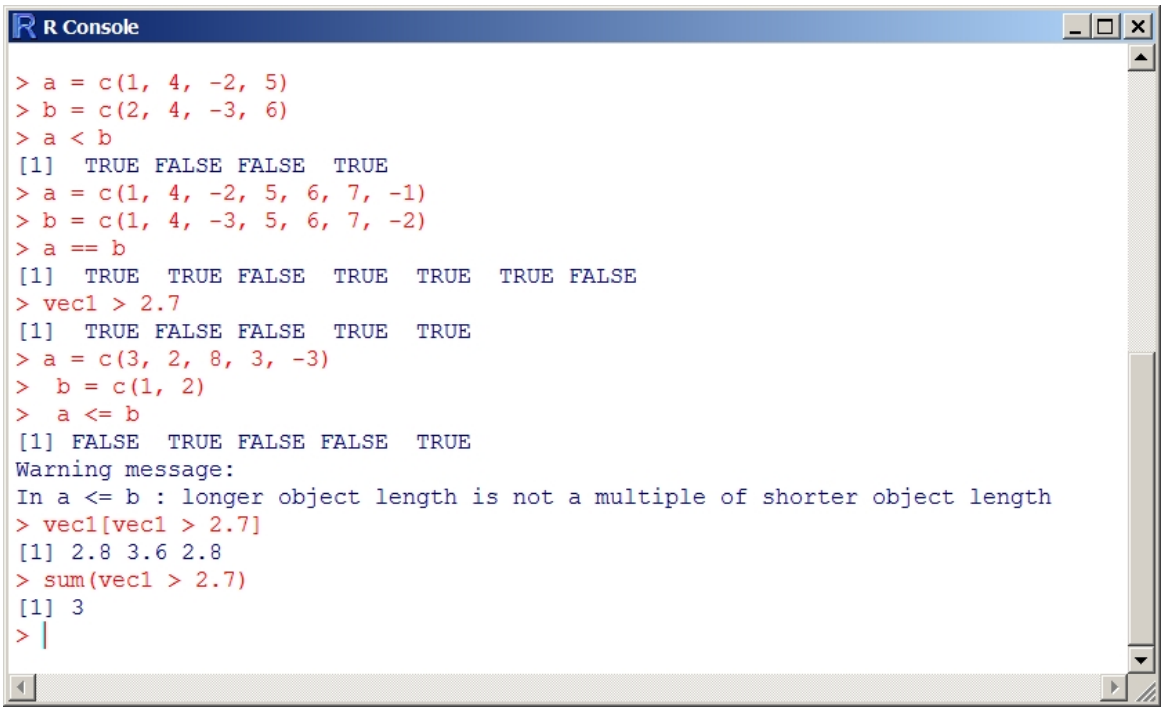
```
sum(vec1 > 2.7)
```

Cela renvoie : `[1] 3`

On a ainsi compté le nombre d'éléments de `vec1` dont les valeurs sont strictement supérieures à `2.7`.

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



Opérateurs appliqués à des vecteurs logiques. On utilise les opérateurs appliqués à des vecteurs logiques suivants :

- ou `|`,
- et `&`,
- négation `!`
- `any` appliqué à un vecteur logique renvoie `TRUE` si au moins un des éléments du vecteur vérifie la relation de comparaison,
- `all` appliqué à un vecteur logique renvoie `TRUE` si tous les éléments du vecteur vérifient la relation de comparaison.

On considère les commandes :

```
(vec1 > 2.7 | vec1 < 2)
```

Cela renvoie : `[1] TRUE FALSE FALSE TRUE TRUE`

On considère les commandes :

```
(vec1 > 2.7 & vec1 < 2)
```

Cela renvoie : `[1] FALSE FALSE FALSE FALSE FALSE`

On fait :

```
any(c(3, 2, 8, 3, -3) <= c(1, 2))
```

Cela renvoie : `[1] TRUE`

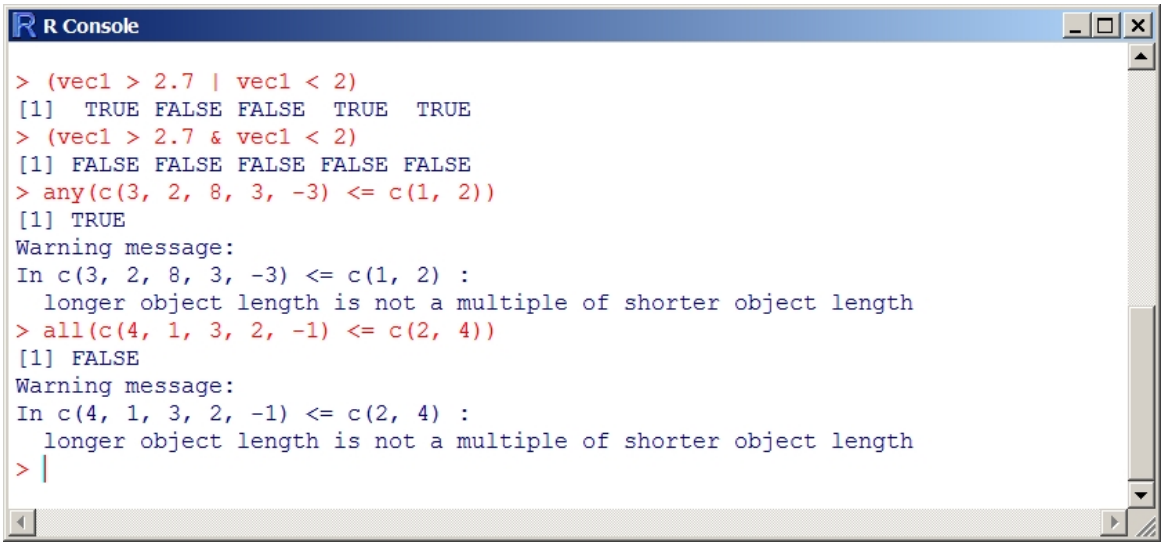
On fait :

```
all(c(4, 1, 3, 2, -1) <= c(2, 4))
```

Cela renvoie : `[1] FALSE`

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



8 Opérations sur les listes

Attribuer des labels aux éléments d'une liste. On fait :

```
names(list1) = c("poids", "couleur", "matrice")
```

Ou alors, plus directement :

```
list2 = list(poids = vec1, couleur = c("rouge", "bleu"), matrice = mat1)
```

Extraire un élément d'une liste. On considère les commandes :

```
list1$couleur
```

Cela renvoie : `[1] "rouge" "bleu"`

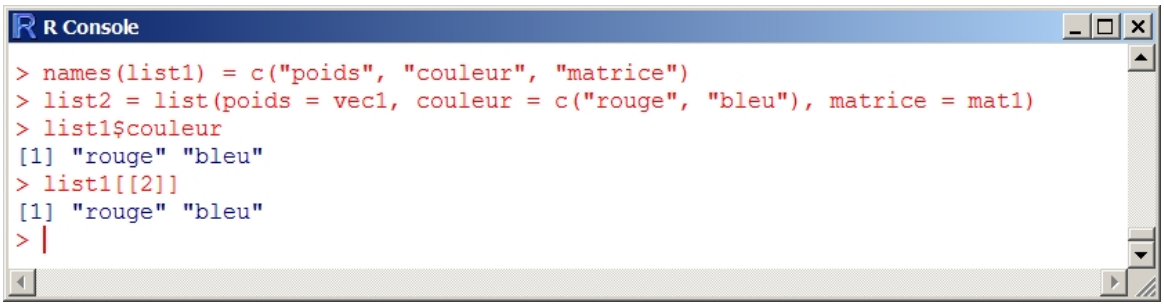
On considère les commandes :

```
list1[[2]]
```

Cela renvoie : `[1] "rouge" "bleu"`

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.



9 Opérations sur les matrices

Attribuer des labels aux lignes et colonnes. On fait :

```
dimnames(mat1) = list(c("jules", "jim"), paste("X", 1:5, sep = ""))

mat1
```

Cela renvoie :

```
      X1  X2  X3  X4  X5
jules 2.8 2.1 2.8 2.4 3.6
jim   2.4 3.6 2.8 2.1 2.8
```

On considère les commandes :

```
mat1["jim", "X3"]
```

Cela renvoie : `[1] 2.8`.

On peut enlever les labels attribués en faisant :

```
dimnames(mat1) = c()
```

Opérations arithmétiques. On fait :

```
mat1 + 5
```

Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.8  2.1  2.8  2.4  3.6
[2,]  2.4  3.6  2.8  2.1  2.8
```

On fait :

```
2 * mat1 - 3
```

Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  2.6  1.2  2.6  1.8  4.2
[2,]  1.8  4.2  2.6  1.2  2.6
```

On considère les commandes :

```
mat1 + mat2
```

Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5]
[1,]  5.6  4.5  4.9  6.0  6.4
[2,]  5.2  6.0  4.9  5.7  5.6
```

Opérations spécifiques

Produit de matrices. On considère les commandes :

```
A = matrix(1:9, ncol = 3)

A %*% mat4
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]    2    4   35
[2,]    4    5   40
[3,]    6    6   45
```

On fait :

```
B = matrix(c(1, 3, 4, 5, 2, 7, 8, 9, 6), ncol = 3)

A %*% B
```

Cela renvoie :

```
[,1] [,2] [,3]
[1,]   41   62   86
[2,]   49   76  109
[3,]   57   90  132
```

Transposée d'une matrice. On fait :

```
t(mat1)
```

Cela renvoie :

```
     [,1] [,2]
[1,]  2.8  2.4
[2,]  2.1  3.6
[3,]  2.8  2.8
[4,]  2.4  2.1
[5,]  3.6  2.8
```

On considère les commandes :

```
t(A)
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Déterminant d'une matrice. On fait :

```
det(mat4)
```

Cela renvoie : `[1] 10`

On fait :

```
det(B)
```

Cela renvoie : `[1] 143`

Inverse d'une matrice. On fait :

```
solve(mat4)
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]  0.5    0  0.0
[2,]  0.0    1  0.0
[3,]  0.0    0  0.2
```

On remarque alors que la matrice obtenue multipliée par `mat4` donne bien la matrice identité ; `solve(mat4)` est l'inverse de `mat4`.

On fait :

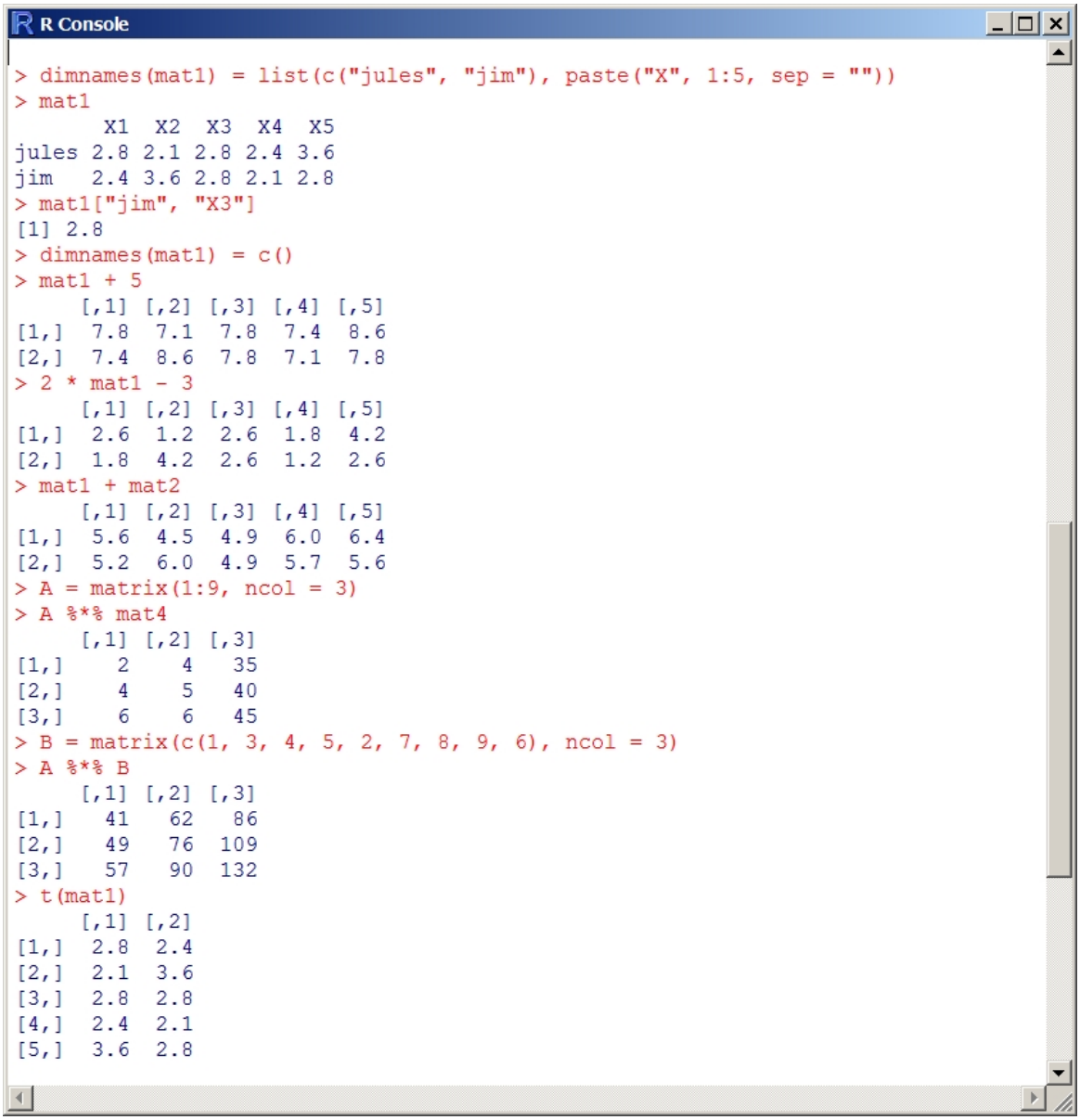
```
solve(B)
```

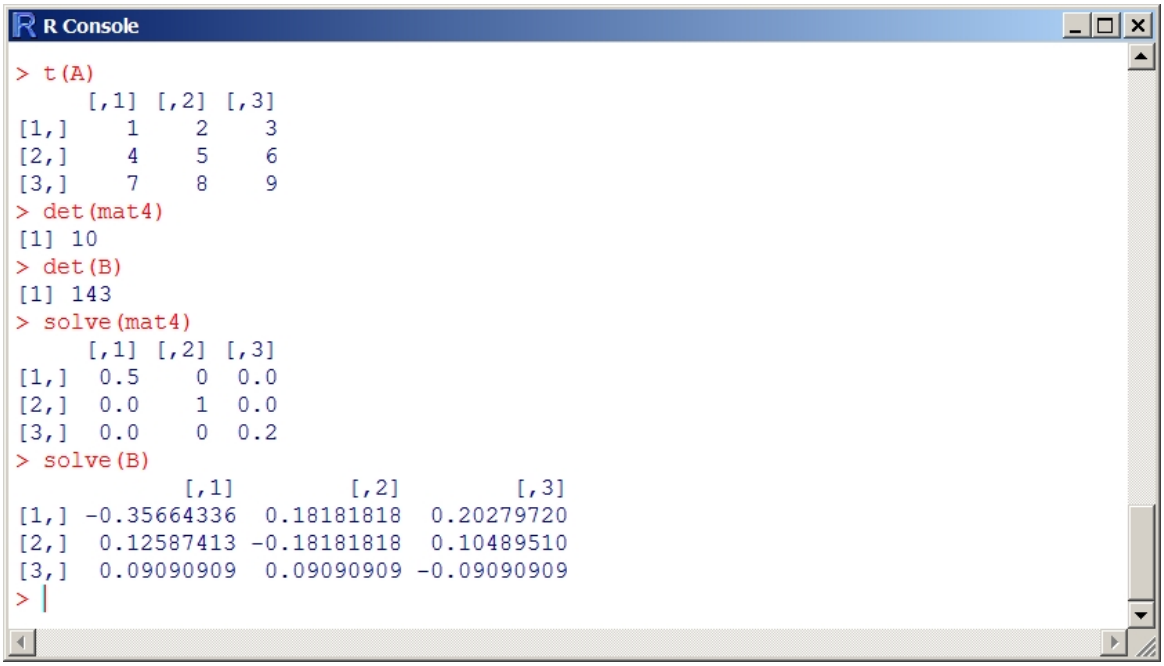
Cela renvoie :

```
            [,1]        [,2]        [,3]
[1,] -0.35664336  0.18181818  0.20279720
[2,]  0.12587413 -0.18181818  0.10489510
[3,]  0.09090909  0.09090909 -0.09090909
```

Bilan

Les commandes précédentes et les sorties associées sont présentées ci-dessous.




R Console
> t(A)
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
> det(mat4)
[1] 10
> det(B)
[1] 143
> solve(mat4)
[,1] [,2] [,3]
[1,] 0.5 0 0.0
[2,] 0.0 1 0.0
[3,] 0.0 0 0.2
> solve(B)
[,1] [,2] [,3]
[1,] -0.35664336 0.18181818 0.20279720
[2,] 0.12587413 -0.18181818 0.10489510
[3,] 0.09090909 0.09090909 -0.09090909
>

10 Pour aller plus loin

10.1 Commandes `Help` et `Help.search`

Dans la fenêtre "R Console", pour avoir les détails d'un l'objet R ainsi que plusieurs exemples d'utilisation, utiliser la commande `help` sous la forme `help("objet")` (`help("matrix")`, `help("sin")`, `help("solve")`...). Vous pouvez alors approfondir des objets R inconnus que vous rencontrerez au fil de vos lectures. Essayer, par exemple :

`help("which.min")`, `help("which.max")`, `help("cumprod")`, `help("function")`, `help("data.frame")`, `help("summary")`, `help("table")`.

Si le nom d'un objet R vous échappe, il existe un moteur de recherche qui permet de le retrouver. Pour cela, dans la fenêtre "R Console", utiliser la commande `help.search` sous la forme `help.search("objet")` (`help.search("diagonale")`, `help.search("transpose")`...). Vous aurez alors une liste d'objet R, dont peut-être celui qui vous intéresse.

10.2 Installer un package

Il est possible que l'objet R qui vous intéresse se trouve dans un "package" qu'il faut installer.

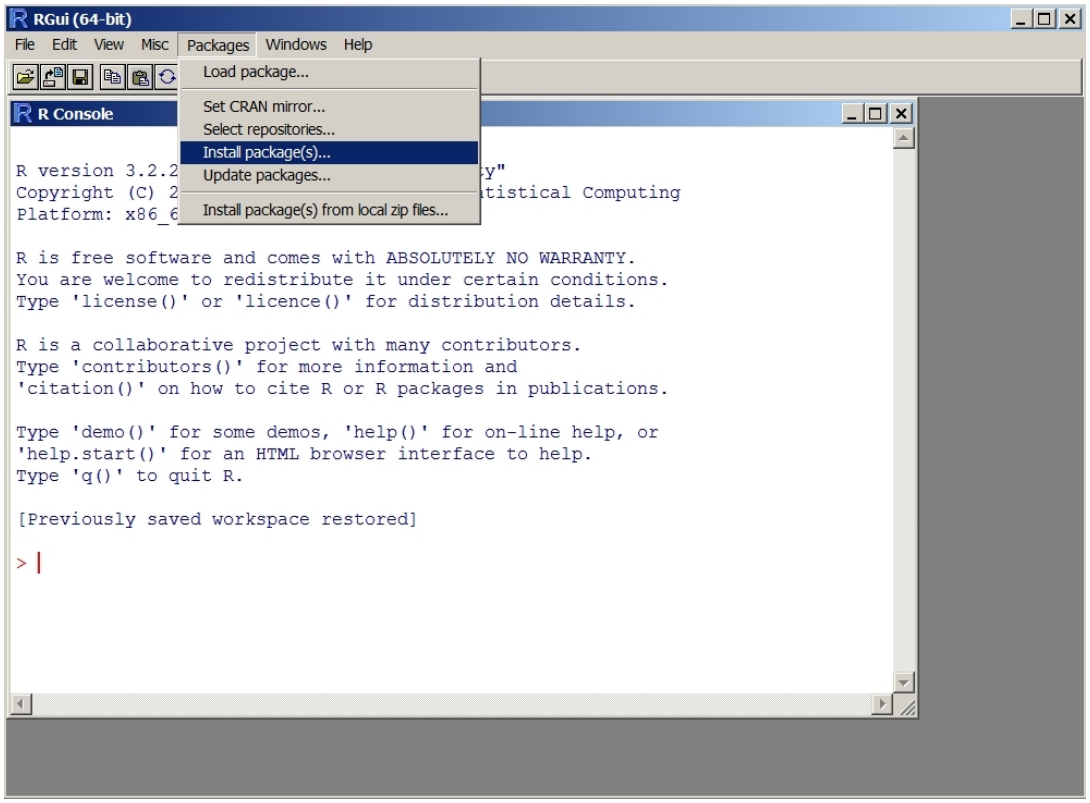
- Lancer le logiciel R, cliquer sur l'option Packages,
- Cliquer sur l'option Install package(s),
- Une fenêtre HTTPS CRAN mirror s'affiche,
- Double-cliquer sur l'option France (Lyon 1)[https],
- Une fenêtre Packages s'affiche,
- Double-cliquer sur le package de votre choix, lançant ainsi son installation.

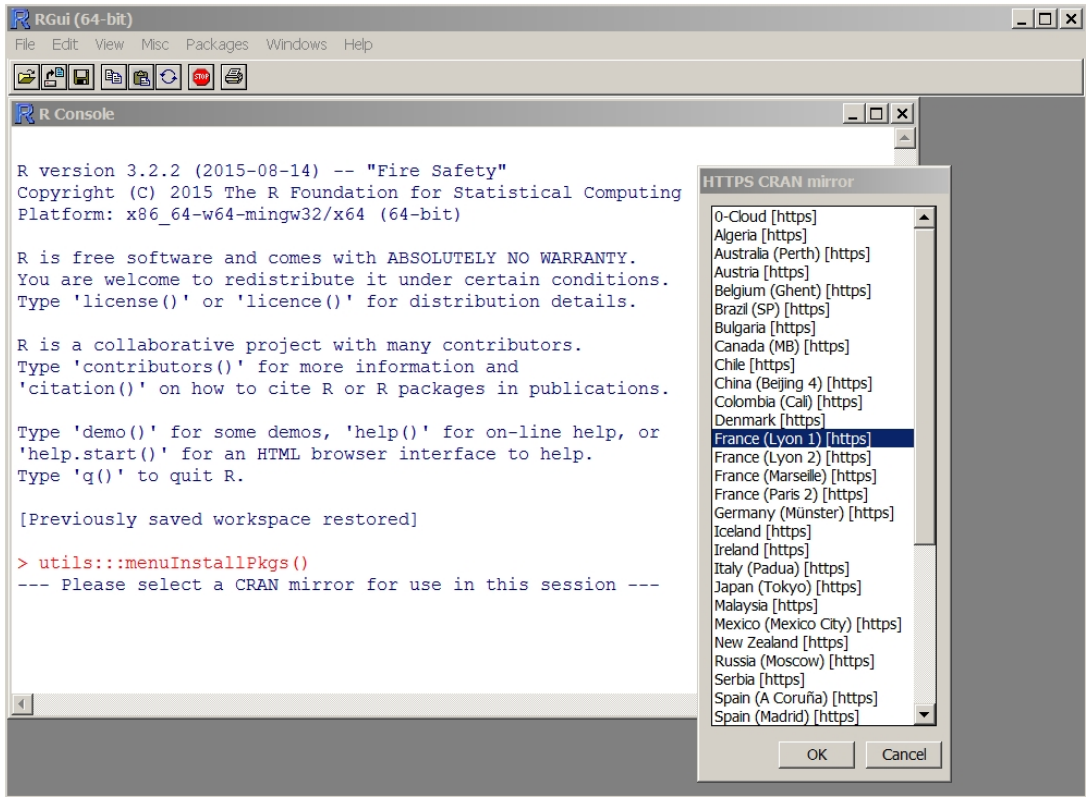
Une fois l'installation terminée, pour charger le package, dans le fenêtre "R Console", faire :

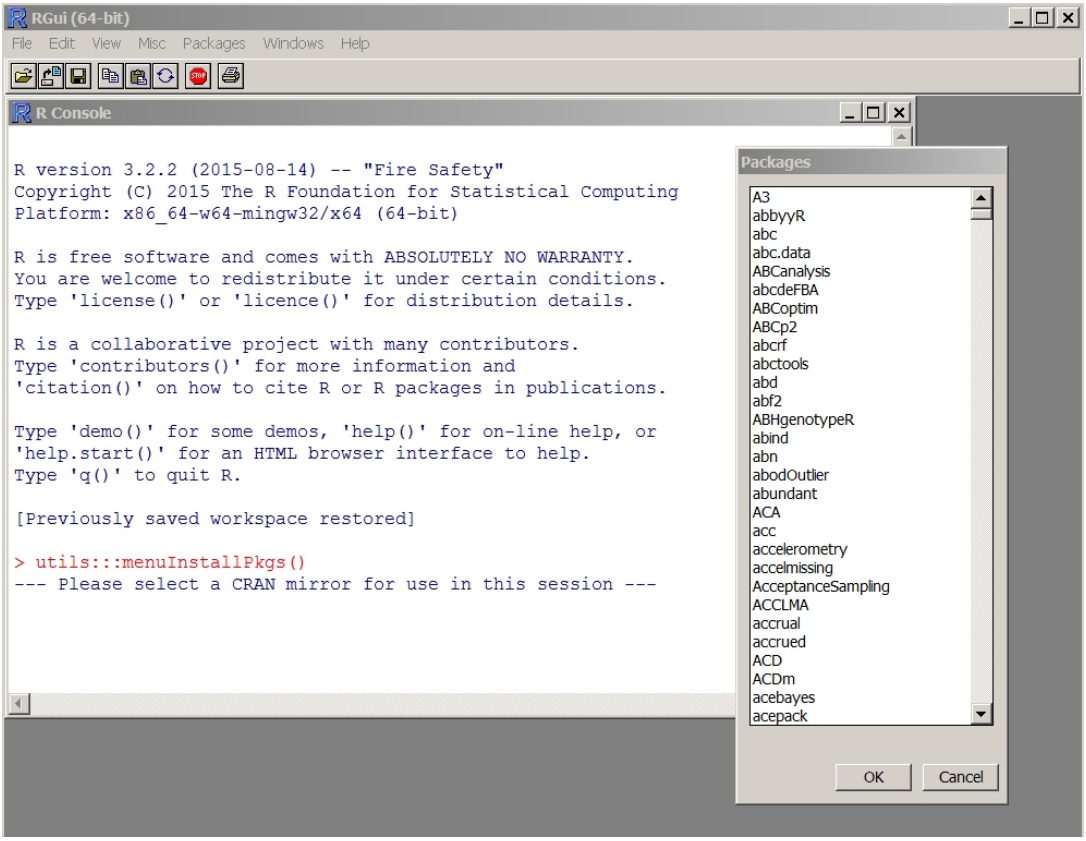
```
library(nomdupackage)
```

Vous pouvez alors appeler votre objet R avec les commandes adequates.

La procédure précédente est présentée ci-dessous.







(Puis double-cliquer sur le package de votre choix, attendre qu'il s'installe et faire `library(nomdupackage)` dans la fenêtre "R Console").

10.3 R-studio

Il est important de se familiariser avec R-studio qui est aujourd'hui l'environnement de développement R le plus utilisé :

```
https://www.rstudio.com/
```

11 Exercices

Exercice 1. On définit trois vecteurs x, y et z par les commandes R suivantes :

```
x = c(1, 3, 5, 7, 9)
y = c(2, 3, 5, 7, 11, 13)
z = c(9, 3, 2, 5, 9, 2, 3, 9, 1)
```

Reproduire et comprendre les résultats des commandes R suivantes :

```
x + 2
y * 3
length(x)
x + y
sum(x > 5)
sum(x[x > 5])
sum(x > 5 | x < 3)
```

```
y[3]
y[-3]
y[x]
(y > 7)
y[y > 7]
sort(z)
sort(z, dec = TRUE)
```

```
rev(z)
order(z)
unique(z)
duplicated(z)
table(z)
rep(z, 3)
```

Exercice 2. Créer les vecteurs suivants :

1. $x0 = (51, 64, 71, 82, 98, 107)$,
2. $x1$ constitué de 5 fois la suite de nombres $(5, 6, 3)$,
3. $x2$ constitué de un 1, deux 2, trois 3, ..., dix 10,
4. $x3$ constitué de 7 noms de votre choix.

Exercice 3. Créer deux vecteurs de dimensions quelconques. Créer un vecteur en insérant le second vecteur entre les 2-ème et 3-ème éléments du premier vecteur.

Exercice 4. Créer un vecteur dont les valeurs des éléments sont $e^x \sin(x)$, avec $x = 2, 2.1, 2.2, \ldots, 7.9, 8$.

Exercice 5. On définit un vecteur x par les commandes R suivantes :

```
x = c (4.12, 1.84, 4.28, 4.23, 1.74, 2.06, 3.37, 3.83, 5.15, 3.76, 3.23, 4.87, 5.96, 2.29, 4.58)
```

1. Créer un vecteur égal à x sans ses 4 premiers éléments.
2. Créer un vecteur égal à x sans ses 1-er et 15-ème éléments.
3. Créer un vecteur contenant les éléments de x dont les valeurs sont strictement supérieures à 2.57 et strictement inférieures à 3.48.
4. Créer un vecteur contenant les éléments de x dont les valeurs sont strictement supérieures à 4.07 ou strictement inférieures à 1.48.
5. Déterminer la coordonnée de la plus petite valeur des éléments de x.

Exercice 6. On considère deux vecteurs numériques à n éléments : $x = (x_1, \ldots, x_n)$ et $y = (y_1, \ldots, y_n)$, que l'on crée dans R. On les note `x` et `y`. Expliciter les vecteurs que l'on crées en faisant :

```
a = y[-1] - x[-length(x)]
b = cos(y[-length(y)]) / sin(x[-1])
```

Exercice 7.

1. Créer les vecteurs suivants :
   (a) $y0$ constitué de la suite des entiers de 0 à 10 par pas de 2,
   (b) $y1$ constitué de tous les entiers pairs entre 1 et 18,
   (c) $y2$ constitué de 20 fois de suite la valeur 4,
   (d) $y3$ constitué de 20 nombres entre 0 et 10.
2. Extraire de $y3$:
   (a) le 3-ème élément,
   (b) tous les éléments sauf le 3-ème.
3. Commenter les commandes suivantes :

```
matrix(y3, nrow = 2)
matrix(y3, byrow = TRUE)
```

4. Construire une matrice A comportant quatre lignes et trois colonnes remplies par lignes successives avec les éléments du vecteur `1:12`.
5. Construire une matrice B comportant quatre lignes et trois colonnes remplies par colonnes successives avec les éléments du vecteur `1:12`.
6. Extraire l'élément situé en 2-ème ligne et 3-ème colonne de A.
7. Extraire la 1-ère colonne de A, puis la 2-ème ligne de A.
8. Construire une matrice C constituée des 1-ère et 4-ème ligne de A.

Exercice 8. Construire une matrice comportant 9 lignes et 9 colonnes avec des 0 sur la diagonale et des 1 partout ailleurs (on pourra utiliser la commande `diag`).

Exercice 9. On définit deux vecteurs x et y par les commandes R suivantes :

```
x = 1:6
y = 5:10
```

1. Remplacer les éléments de $x + y$ dont les valeurs sont supérieures à 11 par 1.
2. Calculer le produit scalaire de x et y.
3. On définit la matrice M par les commandes R suivantes :

   ```
   M = matrix(1:36, nrow = 6)
   ```

   Calculer Mx, xM, M^t et MM^t.

Exercice 10. Proposer des commandes R renvoyant la matrice :

```
       John Lilly Stef Bob Anna Marik Boris
Poids    95    68   85  72   55    86   115
Taille  189   169  179 167  171   178   179
```

Exercice 11. On considère les matrices :

$$A = \begin{pmatrix} -2 & 1 & -3 & -2 \\ 1 & 2 & 1 & -1 \\ -2 & 1 & 1 & -1 \\ -1 & -3 & 1 & 1 \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & -1 & 3 & -4 \\ 2 & -2 & 4 & -5 \\ -2 & 1 & 3 & -1 \\ -1 & -3 & 1 & -1 \end{pmatrix}.$$

1. Montrer que A et B sont inversibles, puis calculer leurs inverses.
2. Vérifier que $\det(A^t) = \det(A)$, $\det(A^{-1}) = 1/\det(A)$ et $\det(AB) = \det(A)\det(B)$.
3. Vérifier que $(A^{-1})^t = (A^t)^{-1}$, $(AB)^t = B^t A^t$ et $(AB)^{-1} = B^{-1}A^{-1}$.

Exercice 12. On considère la matrice :

$$A = \begin{pmatrix} 1 & 1 & 3 \\ 5 & 2 & 6 \\ -2 & -1 & -3 \end{pmatrix}.$$

1. Montrer que A est nilpotente, *i.e.* il existe un entier n tel que A^n est la matrice nulle.
2. Remplacer la 3-ème ligne de A par la somme des deux premières.

Exercice 13. On considère la fonction $f : \mathbb{R} \to \mathbb{R}$ définie par

$$f(x) = ax^2 + bx + c,$$

où a, b et c sont trois réels inconnus tels que :

$$f(0.5) = 7, \qquad f(1) = 4, \qquad f(1.5) = 5.$$

1. Créer dans R la matrice X telle que les informations dont on dispose sur f se traduisent sous la forme matricielle : $X\beta = r$, avec $\beta = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ et $r = \begin{pmatrix} 7 \\ 4 \\ 5 \end{pmatrix}$.

2. Montrer que X est inversible et calculer X^{-1}.

3. Déterminer a, b et c.

Exercice 14. On considère la matrice B décrite par les commandes suivantes :

```
A = matrix(0, nrow = 5, ncol = 5)
B = abs(col(A) - row(A)) + 1
```

1. Montrer que B est inversible et calculer B^{-1}.

2. On considère le système linéaire à 5 réels inconnus : a, b, c, d et e, défini par :

$$\begin{cases} a + 2b + 3c + 4d + 5e = 1 \\ 2a + b + 2c + 3d + 4e = 2 \\ 3a + 2b + c + 2d + 3e = 2 \\ 4a + 3b + 2c + d + 2e = 3 \\ 5a + 4b + 3c + 2d + e = 2 \end{cases}$$

   Résoudre ce système en utilisant la matrice B.

12 Solutions

Solution 1.

`x + 2` renvoie : `[1] 3 5 7 9 11`

`y * 3` renvoie : `[1] 6 9 15 21 33 39`

`length(x)` renvoie : `[1] 5`

On obtient ainsi le nombre d'éléments dans `x`.

`x + y` renvoie : `[1] 3 6 10 14 20 14`

`sum(x > 5)` renvoie : `[1] 2`

Ainsi, on a le nombre d'éléments de `x` dont les valeurs sont > 5 (soient `7` et `9`).

`sum(x[x > 5])` renvoie : `[1] 16`

On a ainsi la somme des valeurs des éléments de `x` supérieurs à 5 (soit `7 + 9`),

`sum(x > 5 | x < 3)` renvoie : `[1] 3`

C'est le nombre d'éléments de `x` dont les valeurs sont > 5 et < 3 (soient `1`, `7` et `9`).

`y[3]` renvoie : `[1] 5`

On a alors la valeur du 3-ème élément de `y`.

`y[-3]` renvoie : `[1] 2 3 7 11 13`

Ainsi, on a le vecteur `y` privé du 3-ème élément.

`y[x]` renvoie : `[1] 2 5 11 NA NA`

On obtient alors un vecteur composé du 1-er élément de `y`, donc `2`, puis du 3-ème élément de `y`, donc `5`, puis du 5-ème élément de `y`, donc `11`, puis du 7-ème élément de `y`, donc rien, d'où le `NA` = Non Available, et enfin, du 9-ème élément de `y`, donc rien, d'où le `NA`.

`(y > 7)` renvoie : `[1] FALSE FALSE FALSE FALSE TRUE TRUE`

On a ainsi un vecteur logique dont les éléments sont `TRUE` si la valeur de l'élément de `y` associé est > 7, et `FALSE` sinon.

`y[y > 7]` renvoie : `[1] 11 13`

On a alors un vecteur contenant les éléments de `y` dont les valeurs sont > 7.

`sort(z)` renvoie : `[1] 1 2 2 3 3 5 9 9 9`

On obtient ainsi un vecteur dont les éléments sont ceux de `z` ordonnés par ordre croissant de leurs valeurs.

`sort(z, dec = TRUE)` renvoie : `[1] 9 9 9 5 3 3 2 2 1`

On obtient ainsi un vecteur dont les éléments sont ceux de `z` ordonnés par ordre décroissant de leurs valeurs.

`rev(z)` renvoie : `[1] 1 9 3 2 9 5 2 3 9`

On a un vecteur qui réarrange les éléments du vecteur `z` en sens inverse.

`order(z)` renvoie : `[1] 9 3 6 2 7 4 1 5 8`

On obtient ainsi le vecteur des rangs de classement des valeurs des éléments de `z` par ordre croissant (la plus petite valeur est en 9-ème position, la 2-ème plus petite est en 3-ème position...).

`unique(z)` renvoie : `[1] 9 3 2 5 1`

On a ainsi un vecteur dont les éléments sont ceux de `z` privés des doublons.

`duplicated(z)` renvoie : `[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE FALSE`

On a ainsi un vecteur logique dont les éléments sont `TRUE` si la valeur de l'élément `z` commence à être répété (de gauche à droite),

`table(z)` renvoie : `[1] 1 2 2 1 3` Ainsi, on obtient un vecteur dont les valeurs des éléments précisent le nombre de fois qu'apparaissent les chiffres : 1, 2, 3, 5 et 9.

`rep(z, 3)` renvoie : `[1] 9 3 2 5 9 2 3 9 1 9 3 2 5 9 2 3 9 1 9 3 2 5 9 2 3 9 1`

On a ainsi un vecteur qui concatène 3 fois le vecteur `z`.

Solution 2.

1. On fait : `x0 = c(51, 64, 71, 82, 98, 107)`
2. On fait : `x1 = rep(c(5, 6, 3), 4)`
3. On fait : `x2 = rep(1:10, 1:10)`
4. On fait : `c("John", "Lilly", "Stef", "Bob", "Anna", "Marik", "Boris")`

Solution 3. On fait :

```
x = 1:10
```

```
y = rep(0, 3)
z = c(x[1:2], y, x[3:length(x)])
```

Solution 4. On fait :

```
x = seq(2, 8, by = 0.1)
exp(x) * sin(x)
```

Solution 5.

1. On fait : `x1 = x[-(1:4)]`
2. On fait : `x2 = x[-c(1, 15)]`
3. On fait : `x3 = x[(x > 2.57) & (x < 3.48)]`
4. On fait : `x4 = x[(x > 4.07) | (x < 1.48)]`
5. On fait : `which.min(x)`

   Cela renvoie : `[1] 5`

   La plus petite valeur se situe au 5-ème élément de x.

Solution 6. On calcule les vecteurs :

$$a = (y_2 - x_1, y_3 - x_2, \ldots, y_n - x_{n-1}), \qquad b = \left(\frac{\cos(y_1)}{\sin(x_2)}, \frac{\cos(y_2)}{\sin(x_3)}, \ldots, \frac{\cos(y_n)}{\sin(x_{n-1})}\right).$$

Solution 7.

1. (a) On fait : `y0 = seq(0, 10, 2)`

   (b) On fait : `y1 = seq(2, 18, 2)`

   (c) On fait : `y2 = rep(4, 20)`

   (d) On fait : `y3 = seq(0, 10, length.out = 20)`
2. (a) On fait : `y3[3]`

   (b) On fait : `y3[-3]`
3. Les commandes `matrix(y3, nrow = 2)` transforment le vecteur `y3` en une matrice à 2 lignes et 10 colonnes.

Les commandes `matrix(y3, byrow = TRUE)` transforment le vecteur `y3` en un vecteur colonne.

4. On fait : `A = matrix(1:12, nrow = 4, ncol = 3, byrow = TRUE)`
5. On fait : `B = matrix(1:12, nrow = 4, ncol = 3, byrow = FALSE)`
6. On fait : `A[2, 3]`
7. On fait : `A[ ,1] ; A[2, ]`
8. On fait : `C = A[c(1, 4), ]`

Solution 8. On fait : `M = matrix(1, 9, 9) - diag(9)`

Solution 9.

1. On fait :

   ```
   z = x + y
   z[z > 11] = 1
   z
   ```

   Cela renvoie : `[1] 6 8 10 1 1 1`

2. On fait : `x %*% y`

   Cela renvoie :

   ```
        [,1]
   [1,]  175
   ```

3. ◦ On fait : `M %*% x`

   Cela renvoie :

   ```
        [,1]
   [1,]  441
   [2,]  462
   [3,]  483
   [4,]  504
   [5,]  525
   [6,]  546
   ```

- On fait : `x %*% M`

  Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]   91  217  343  469  595  721
```

- On fait : `t(M)`

  Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    3    4    5    6
[2,]    7    8    9   10   11   12
[3,]   13   14   15   16   17   18
[4,]   19   20   21   22   23   24
[5,]   25   26   27   28   29   30
[6,]   31   32   33   34   35   36
```

- On fait : `M %*% t(M)`

  Cela renvoie :

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 2166 2262 2358 2454 2550 2646
[2,] 2262 2364 2466 2568 2670 2772
[3,] 2358 2466 2574 2682 2790 2898
[4,] 2454 2568 2682 2796 2910 3024
[5,] 2550 2670 2790 2910 3030 3150
[6,] 2646 2772 2898 3024 3150 3276
```

Solution 10. On fait :

```
M = matrix(c(95, 189, 68, 169, 85, 179, 72, 167, 55, 171, 86, 178, 115, 179), ncol = 7)
dimnames(M) = list(c("Poids", "Taille"), c("John", "Lilly", "Stef", "Bob",
"Anna", "Marik", "Boris"))
M
```

Solution 11.

1. ◦ On fait :

```
A = matrix(c(-2, 1, -2, -1, 1, 2, 1, -3, -3, 1, 1, 1, -2, -1, -1, 1), ncol = 4)
det(A)
```

Cela renvoie : `[1] 25`

Comme $\det(A) = 25 \neq 0$, A est inversible. On obtient son inverse en faisant :

```
solve(A)
```

Cela renvoie :

```
      [,1]  [,2]  [,3] [,4]
[1,]  0.08  0.48 -0.44  0.2
[2,] -0.24 -0.44  0.32 -0.6
[3,] -0.12  0.28  0.16  0.2
[4,] -0.52 -1.12  0.36 -0.8
```

◦ `B = matrix(c(2, 2, -2, -1, -1, -2, 1, -3, 3, 4, 3, 1, -4, -5, -1, -1), ncol = 4)`

```
det(B)
```

Cela renvoie : `[1] 4`

Comme $\det(B) = 4 \neq 0$, B est inversible. On obtient son inverse en faisant :

```
solve(B)
```

Cela renvoie :

```
       [,1]  [,2]  [,3] [,4]
[1,]  -7.50  6.50 -0.50 -2.0
[2,]   3.25 -2.75  0.25  0.5
[3,] -10.25  8.75 -0.25 -2.5
[4,] -12.50 10.50 -0.50 -3.0
```

2. ◦ On fait :

```
det(t(A)) ; det(A)
```

Cela renvoie :

```
[1] 25
[1] 25
```

D'où l'égalité : $\det(A^t) = \det(A)$.

- On fait :

```
det(solve(A)) ; 1 / det(t(A))
```

Cela renvoie :

```
[1] 0.04
```

```
[1] 0.04
```

D'où l'égalité : $\det(A^{-1}) = 1/\det(A)$.

- On fait :

```
det(A %*% B) ; det(A) * det(B)
```

Cela renvoie :

```
[1] 100
```

```
[1] 100
```

D'où l'égalité : $\det(AB) = \det(A)\det(B)$.

3.

- On fait :

```
t(solve(A)) ; solve(t(A))
```

Cela renvoie :

```
      [,1]  [,2]  [,3]  [,4]
[1,]  0.08 -0.24 -0.12 -0.52
[2,]  0.48 -0.44  0.28 -1.12
[3,] -0.44  0.32  0.16  0.36
[4,]  0.20 -0.60  0.20 -0.80
      [,1]  [,2]  [,3]  [,4]
[1,]  0.08 -0.24 -0.12 -0.52
[2,]  0.48 -0.44  0.28 -1.12
[3,] -0.44  0.32  0.16  0.36
[4,]  0.20 -0.60  0.20 -0.80
```

D'où l'égalité : $(A^{-1})^t = (A^t)^{-1}$.

- On fait :

```
t(A %*% B) ; t(B) %*% t(A)
```

Cela renvoie :

```
     [,1] [,2] [,3] [,4]
[1,]    6    5   -3  -11
[2,]    3   -1    4    5
[3,]  -13   13    0  -11
[4,]    8  -14    3   17
     [,1] [,2] [,3] [,4]
[1,]    6    5   -3  -11
[2,]    3   -1    4    5
[3,]  -13   13    0  -11
[4,]    8  -14    3   17
```

D'où l'égalité : $(AB)^t = B^t A^t$.

- On fait :

```
solve(A %*% B) ; solve(B) %*% solve(A)
```

Cela renvoie :

```
      [,1]  [,2]  [,3]  [,4]
[1,] -1.06 -4.36  4.58 -3.90
[2,]  0.63  2.28 -2.09  1.95
[3,] -1.59 -6.04  6.37 -5.35
[4,] -1.90 -7.40  7.70 -6.50
      [,1]  [,2]  [,3]  [,4]
[1,] -1.06 -4.36  4.58 -3.90
[2,]  0.63  2.28 -2.09  1.95
[3,] -1.59 -6.04  6.37 -5.35
[4,] -1.90 -7.40  7.70 -6.50
```

D'où l'égalité : $(AB)^{-1} = B^{-1} A^{-1}$.

Solution 12.

1. On fait :

```
A %*% A %*% A
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

Donc A est nilpotente ; A^3 est la matrice nulle.

2. On fait :

```
A[3, ] = A[1, ] + A[2, ]
A
```

Cela renvoie :

```
     [,1] [,2] [,3]
[1,]    1    1    3
[2,]    5    2    6
[3,]    6    3    9
```

Solution 13.

1. On a

$$\begin{pmatrix} f(0.5) \\ f(1) \\ f(1.5) \end{pmatrix} = \begin{pmatrix} 0.5^2 a + 0.5b + c \\ 1^2 \times a + 1 \times b + c \\ 1.5^2 a + 1.5b + c \end{pmatrix} = \begin{pmatrix} 0.5^2 & 0.5 & 1 \\ 1^2 & 1 & 1 \\ 1.5^2 & 1.5 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} = X\beta$$

avec $X = \begin{pmatrix} 0.5^2 & 0.5 & 1 \\ 1^2 & 1 & 1 \\ 1.5^2 & 1.5 & 1 \end{pmatrix}$.

Ainsi, la matrice X recherchée est donnée par :

```
X = matrix(c(0.5^2, 0.5, 1, 1^2, 1, 1, 1.5^2, 1.5, 1), ncol = 3, byrow = TRUE)
```

2. On fait :

   ```
   det(X)
   ```

   Cela renvoie : `[1] -0.25`.

   Comme $\det(X) \neq 0$, X est inversible.

   L'inverse de X est donnée par :

   ```
   solve(X)
   ```

   Cela renvoie :

   ```
        [,1] [,2] [,3]
   [1,]    2   -4    2
   [2,]   -5    8   -3
   [3,]    3   -3    1
   ```

3. On a $\beta = X^{-1}r$. On fait :

   ```
   solve(X) %*% c(7, 4, 5)
   ```

   Cela renvoie :

   ```
        [,1]
   [1,]    8
   [2,]  -18
   [3,]   14
   ```

   Ainsi, on a $a = 8$, $b = -18$ et $c = 14$.

Solution 14.

1. On fait : `A = matrix(0, nrow = 5, ncol = 5)`

   ```
   B = abs(col(A) - row(A)) + 1
   B
   ```

Cela renvoie :

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

On fait :

```
det(B)
```

Cela renvoie : `[1] 48`.

Comme $\det(B) \neq 0$, B est inversible.

L'inverse de B est donnée par :

```
solve(B)
```

Cela renvoie :

```
              [,1]          [,2]          [,3]          [,4]          [,5]
[1,] -4.166667e-01  5.000000e-01 -8.881784e-17  3.256654e-16  8.333333e-02
[2,]  5.000000e-01 -1.000000e+00  5.000000e-01 -7.401487e-16  1.850372e-16
[3,] -1.110223e-16  5.000000e-01 -1.000000e+00  5.000000e-01  0.000000e+00
[4,]  2.775558e-17 -1.110223e-16  5.000000e-01 -1.000000e+00  5.000000e-01
[5,]  8.333333e-02  2.465190e-32 -1.110223e-16  5.000000e-01 -4.166667e-01
```

2. On considère les commandes :

```
solve(B) %*% c(1, 2, 2, 3, 2)
```

Cela renvoie :

```
      [,1]
[1,]  0.75
[2,] -0.50
[3,]  0.50
[4,] -1.00
[5,]  0.75
```

Ainsi, on a $a = 0.75$, $b = -0.50$, $c = 0.50$, $d = -1.00$ et $e = 0.75$.