



**INSTITUT
de la
communication**



Algèbre Linéaire et Analyse de Données

Licence 2 MIASHS (2021-2022)

Guillaume Metzler


Institut de Communication (ICOM)

Université de Lyon, Université Lumière Lyon 2

Laboratoire ERIC UR 3083, Lyon, France

guillaume.metzler@univ-lyon2.fr

Résumé

Ces premiers travaux Pratiques ont pour objectif de vous faire prendre en main le logiciel  via des manipulations simples mobilisant quelques outils présentés en Algèbre Linéaire pour résoudre des problèmes élémentaires mais aussi à vous faire manipuler les objets courants de l'Analyse de Données.


1 Introduction


Le logiciel , disponible en téléchargement à l'adresse suivante :


<http://www.r-project.org/>

est un langage de programmation spécifiquement dédiée aux statisticiens mais également à toutes personnes souhaitant pratiquer l'Apprentissage Machine ou l'Analyse de Données.

Il est également utilisé pour la richesse de ses options graphiques.


Son fonctionnement repose essentiellement sur l'utilisation de *packages* ou de *librairies* comme c'est le cas pour Python. Ces *packages* sont créés par la communauté et peuvent être mis à disposition pour l'ensemble de la communauté d'utilisateurs de .

Bien que  soit utilisé dans un cadre universitaire, il est aussi employé dans le professionnel par certaines entreprises qui utilisent grandement les statistiques dans le cadre de leur activité.

Dans un cadre plus orienté *Machine Learning* qui n'est pas éloigné de l'*Analyse de Données*,  se retrouve parfois mis de côté afin de privilégier l'utilisation de *Python* dont l'usage est croissant parmi les informaticiens.


On va cependant privilégier l'usage de  dans le cadre de ce cours, pour sa simplicité d'utilisation et surtout pour son usage encore prépondérant par des non informaticiens, *i.e.* sociologues, géographes, journalistes, biologistes, ... à des fins statistiques.

Il est possible d'utiliser  de deux façons différentes :



- en l'installant directement sur votre machine,
- directement en ligne via Studio Cloud.

Il est disponible sur Mac/Windows/Linux

1.1 et Studio

Le logiciel  est un logiciel gratuit et OpenSource. Vous pouvez l'installer gratuitement à l'adresse suivante :

<http://www.r-project.org/>

Une fois que le logiciel est installé, vous pouvez maintenant l'utiliser. Cette première version est cependant assez "sobre" d'un point de vue esthétique et on préférera souvent installer un deuxième logiciel qui viendra se greffer sur  dont l'usage sera plus agréable et qui fournira une interface plus riche. Pour cela, vous pouvez installer Studio à l'adresse suivante :

<https://rstudio.com/products/rstudio/download/#download>

1.2 RStudio Cloud

Dans le cas où vous ne souhaitez pas installer RStudio sur votre ordinateur ou que vous ne l'avez pas en votre possession, il est toujours possible d'accéder aux fonctionnalités de R directement en ligne. Vous pouvez faire cela dans le cadre de ce TP si vous souhaitez ensuite pouvoir avoir accès à votre fichier chez vous.

Une version gratuite de l'utilisation de cette version en ligne peut se faire à l'adresse suivante :

<https://rstudio.cloud/plans/free>

Vous devrez cependant vous créer un compte pour cela et veillez à bien choisir la création d'un compte **gratuit**.

2 Prise en main

Remarquez que votre interface Rse compose de quatre parties dont vous découvrirez tous les usages au fur et à mesure que vous vous familiariserez avec le logiciel

1. **Supérieure gauche** : c'est l'emplacement de votre script (*i.e.* fichier code) ou vous pourrez **écrire** mais aussi **commenter** votre code et l'enregistrer à toute fin utile.
Ce code peut directement être exécuté via la commande *run* se trouvant en haut à droite.
2. **Inférieure gauche** : il s'agit de la *console* dans laquelle vous pourrez écrire du code directement et l'exécuter en appuyant sur *Entrée*.
3. **Supérieure droite** : vous y trouverez toutes les informations relatives à votre environnement de travail, notamment les objets que vous avez créés ainsi que leurs valeurs. C'est également ici que vous aurez la possibilité d'importer des jeux de données.
4. **Inférieure droite** : vous pourrez y trouver l'interface graphique de R, *i.e.* l'endroit où sont générés les graphes. Vous pourrez également, dans cette partie là, accéder à l'interface d'aide des fonctions de R ou encore installer les packages dont vous aurez besoin dans le cadre de votre travail.

2.1 Préliminaires

1. Ouvrez un nouveau fichier en allant sur *File* puis *New File*, fichier dans lequel vous écrirez votre code pour cette première séance.

2. Enregistrez votre fichier sous le nom *PriseEnMain* dans un répertoire de votre ordinateur (ou espace). Cela vous permettra de retrouver et de manipuler votre code à n'importe quel moment.
3. Entrer la commande suivante dans la console et l'exécuter

```
# Création d'une variable x  
x = 1
```

Que remarquez-vous ?

Rien ne s'affiche dans la console. La commande est une commande d'**affectation**, elle attribue la valeur 1 à une variable globale x . En revanche, elle n'affiche pas sa valeur.

Exécuter ensuite la commande

```
x  
## [1] 1
```

La variable x ainsi créée est une variable dite globale à laquelle on a affecté la valeur 1. On peut maintenant réutiliser cette variable aux différentes étapes de notre code.

4. Déterminer la valeur de $(x + 3)^5 - 5x^4 + 2 \times x + 2$.

```
(x+3)^5 - 5*x^4 + 2*x + 2  
## [1] 1023
```

On peut aussi créer des objets plus complexes qui ne contiennent pas qu'une seule valeur.

5. Dans votre script, entrer les commandes suivantes et les exécuter


```
# Création d'une variable  
s = seq(1,10)  
r = rep(0,10)
```

Que font ces différentes commandes ?

La variable s contient tous les entiers de 1 à 10. La fonction *seq* désigne le terme *sequence* qui signifie *suite* et génère la suite des valeurs de 1 à 10 avec un pas de 1.

La variable r est un vecteur de taille 10 ne comprenant que des 0. La fonction *rep* signifie *repeat*.

2.2 Vecteurs et Matrices

On va maintenant se concentrer sur les objets comme les vecteurs et les matrices afin de voir comment effectuer les opérations sous .

1. On peut créer un vecteur à l'aide de commande suivante

```
# Création d'un vecteur v1  
v1 = c(1,2,3)
```

où c désigne la concaténation des éléments 1, 2 et 3. Créer les vecteurs suivants, on les appellera v_2 et v_3

$$\mathbf{v}_2 = \begin{pmatrix} 2 \\ 8 \\ -1 \end{pmatrix} \quad \text{et} \quad \mathbf{v}_3 = \begin{pmatrix} 3 \\ -1 \\ 5 \end{pmatrix}.$$

```
v2 = c(2,8,-1)  
v3 = c(2,-1,5)
```

2. Que font les opérations suivantes ?

```
v1+v2  
## [1] 3 10 2  
  
v1*v2  
## [1] 2 16 -3
```

On effectue la somme et le produit des deux vecteurs. Les opérations effectuées se font "terme à terme" ou encore "composante par composante".

3. Ecrire la somme des vecteurs \mathbf{v}_2 et \mathbf{v}_3 .

```
v2+v3  
## [1] 4 7 4
```

4. Déterminer la valeur de $3\mathbf{v}_3$.

```
3*v3  
## [1] 6 -3 15
```

5. On peut aussi extraire la composante d'un vecteur à l'aide de la commande suivante

```
#Extraction de la première composante du vecteur v1
v1[1]

## [1] 1
```

Calculer la valeur du vecteur $\mathbf{v}_1 - 3\mathbf{v}_2 + 2\mathbf{v}_3$ et en extraire la deuxième composante

```
v = v1 - 3*v2 + 2*v3
v[2]

## [1] -24
```

6. Il est possible de concaténer des vecteurs à l'aide de la commande

```
# Concaténation des vecteurs v1 et v2
u = c(v1,v2)
u

## [1] 1 2 3 2 8 -1
```

et d'accéder à un sous ensemble de ses composantes, par exemple 2 et 5, par

```
# Extraction des première et cinquième coordonnées
u[c(1,5)]

## [1] 1 8
```

Concaténer les vecteurs \mathbf{v}_1 , \mathbf{v}_2 et \mathbf{v}_3 dans un objet \mathbf{w} et extraire les coordonnées paires de ce vecteur

```
# Création du vecteur
w = c(v1,v2,v3)
# Extraction des coordonnées paires
w[c(2,4,6,8)]

## [1] 2 2 -1 -1

# ou encore
w[seq(2,length(w),2)]

## [1] 2 2 -1 -1
```

7. Créer des les vecteurs \mathbf{u}_1 , \mathbf{u}_2 et \mathbf{u}_3 définis par

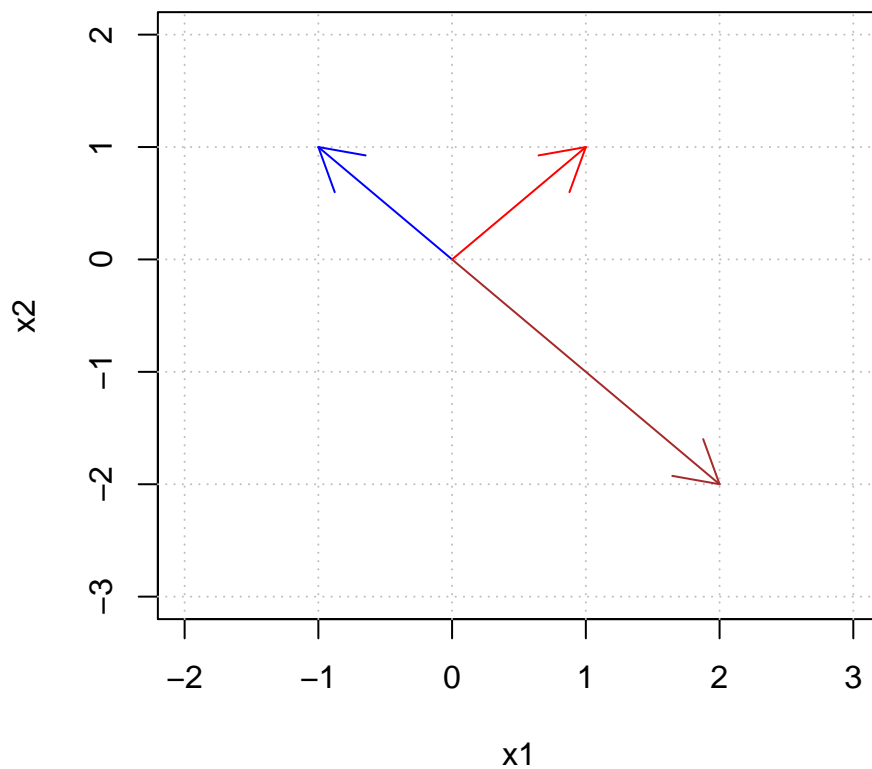
$$\mathbf{u}_1 = \begin{pmatrix} -1 \\ 1 \end{pmatrix}, \quad \mathbf{u}_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \text{et} \quad \mathbf{u}_3 = \begin{pmatrix} 2 \\ -2 \end{pmatrix}$$

```
# Création de vecteurs dans R2
u1=c(-1,1)
u2=c(1,1)
u3=c(2,-2)
```

Exécuter le code suivant et le commenter. Que peut-on dire des vecteurs \mathbf{u}_1 , \mathbf{u}_2 et \mathbf{u}_3

```
# Création d'un plan et ajout d'une grille
plot(NA, xlim=c(-2,3), ylim=c(-3,2), xlab="x1", ylab="x2")
grid(col = "gray", lty = "dotted", lwd = 1)

# Représentation des vecteurs dans le plan
arrows(0,0,u1[1],u1[2],col="blue")
arrows(0,0,u2[1],u2[2],col="red")
arrows(0,0,u3[1],u3[2],col="brown")
```



La fonction `plot` permet d'initier une fenêtre graphique. Ici, cette fenêtre sera

vide de "points" en raison du paramètre **NA**. On précise ensuite les limites graphiques de chaque axe avec les paramètres *xlim* et *ylim*. On nomme chaque avec les paramètres *xlab* et *ylab*.

La fonction *grid* permet de générer une grille sur notre fenêtre graphique, ici de couleur grise (paramètre *col*), avec des lignes en pointillés (paramètre *lty* pour "line type") et enfin l'épaisseur du trait avec le paramètre *lwd* pour "line width".

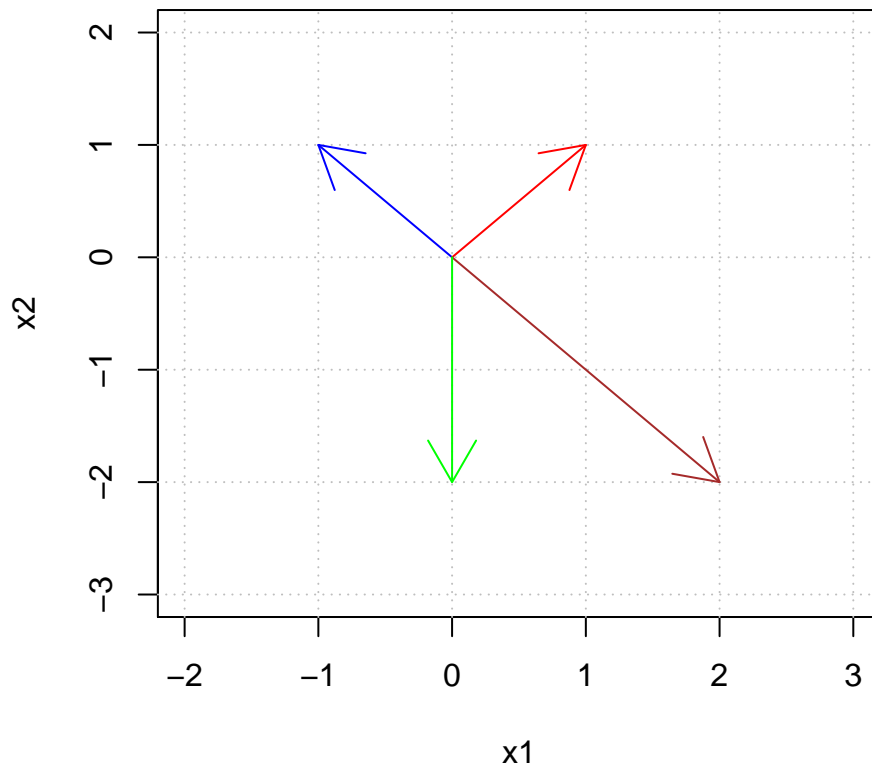
Les vecteurs sont ensuite représentés par des flèches (*arrows*) dont on précise les coordonnées de la base de la flèche et les coordonnées de l'extrémité de la flèche ainsi que la couleur.

8. Représenter graphiquement le vecteur $\mathbf{u} = \mathbf{u}_1 - \mathbf{u}_2 + \mathbf{u}_3$ dans le plan précédent et le colorant en vert.

```
u = u1 - u2 + u3

# Création d'un plan et ajout d'une grille
plot(NA, xlim=c(-2,3), ylim=c(-3,2), xlab="x1", ylab="x2")
grid(col = "gray", lty = "dotted", lwd = 1)

# Représentation des vecteurs dans le plan
arrows(0,0,u1[1],u1[2],col="blue")
arrows(0,0,u2[1],u2[2],col="red")
arrows(0,0,u3[1],u3[2],col="brown")
arrows(0,0,u[1],u[2],col="green")
```

9. Exécuter les commandes suivantes et comparer les résultats pour la création de matrices

```
V1 = matrix(1:9, nrow = 3)
V2 = matrix(1:9, nrow = 3, byrow = TRUE)
```

V1 est une matrice comprenant les valeurs de 1 à 9 avec une complétion de la matrice par colonnes alors que dans V2, la complétion se fait par ligne (option *byrow = TRUE*).

Quid si l'on exécute la commande suivante ?

```
matrix(1:9, nrow = 2)

## Warning in matrix(1:9, nrow = 2): data length [9] is not a sub-multiple
or multiple of the number of rows [2]
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  1   3   5   7   9
## [2,]  2   4   6   8   1
```

Ici on remarque un manque de cohérence entre le nombre de valeurs que l'on souhaite mettre dans une matrice et sa dimension.

10. Exécuter les commandes suivantes et comparer les résultats pour la création de matrices

```
V3 = rbind(v1,v2,v3)
V4 = cbind(v1,v2,v3)
```

V3

```
##      [,1] [,2] [,3]
## v1      1      2      3
## v2      2      8     -1
## v3      2     -1      5
```

V4

```
##      v1 v2 v3
## [1,]  1  2  2
## [2,]  2  8 -1
## [3,]  3 -1  5
```

La fonction *rbind* permet de concaténer des vecteurs en **ligne**, "r pour row".
La fonction *cbind* permet de concaténer des vecteurs en *colonne*, "c pour column".

11. Exécuter et commenter les opérations suivantes sur les matrices

V1+V3

```
##      [,1] [,2] [,3]
## v1      2      6     10
## v2      4     13      7
## v3      5      5     14
```

V1-V3

```
##      [,1] [,2] [,3]
## v1      0      2      4
## v2      0     -3      9
## v3      1      7      4
```

V1*V3

```
##      [,1] [,2] [,3]
## v1      1      8     21
## v2      4     40     -8
## v3      6     -6     45
```

```

V1**V3

##      [,1] [,2] [,3]
## [1,]   23   27   34
## [2,]   28   36   41
## [3,]   33   45   48

3*V1

##      [,1] [,2] [,3]
## [1,]    3   12   21
## [2,]    6   15   24
## [3,]    9   18   27

V1 + v1

##      [,1] [,2] [,3]
## [1,]    2    5    8
## [2,]    4    7   10
## [3,]    6    9   12

```

Les trois premières opérations sont respectivement la somme, la différence et le produit de deux matrices (au sens produit de Hadamard \neq différent du produit matriciel connu). Toutes ces opérations se font composante par composante.

La quatrième opération est le produit matriciel standard.

La cinquième opération consiste en la multiplication de toutes les entrées de la matrice par la constante 3.

La dernière opération ajoute les valeurs de v_1 à V_1 , colonne par colonne.

12. Sachant que la transposée et le déterminant d'une matrice A sont données par les commandes suivantes

```

t(A)
det(A)

```

Calculer le déterminant de la matrice V_1 précédente ainsi que le produit $Z = 4V_1^T V_3$.

```

det(V1)

## [1] 0

Z = 4*t(V1)**V3

```

13. Enfin identifier les fonctions des lignes de code suivantes

```

V1
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

V1[1,3]

## [1] 7

V1[1,3] = 100
V1[1,3]

## [1] 100

V1
##      [,1] [,2] [,3]
## [1,]    1    4 100
## [2,]    2    5    8
## [3,]    3    6    9

V2
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9

V2[,2]

## [1] 2 5 8

V2[3,]

## [1] 7 8 9

V2[1,] = c(-3,-2,-1)
V2
##      [,1] [,2] [,3]
## [1,]   -3   -2   -1
## [2,]    4    5    6
## [3,]    7    8    9

```

Il suffit d'observer les résultats des sorties des différentes commandes pour conclure.

14. On considère l'application linéaire $\theta : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ définie par

$$\theta(\mathbf{x}) = \left(\frac{\sqrt{2}}{2}(x_1 - x_2), \frac{\sqrt{2}}{2}(x_1 + x_2) \right).$$

- (a) Donner la matrice O de cette application linéaire et la définir dans .

On rappelle que la matrice d'une application linéaire est donnée par l'image des vecteurs de base par cette application.

```
O = sqrt(2)/2*matrix(c(1,1,-1,1),nrow=2)
O
##           [,1]      [,2]
## [1,] 0.7071068 -0.7071068
## [2,] 0.7071068  0.7071068
```

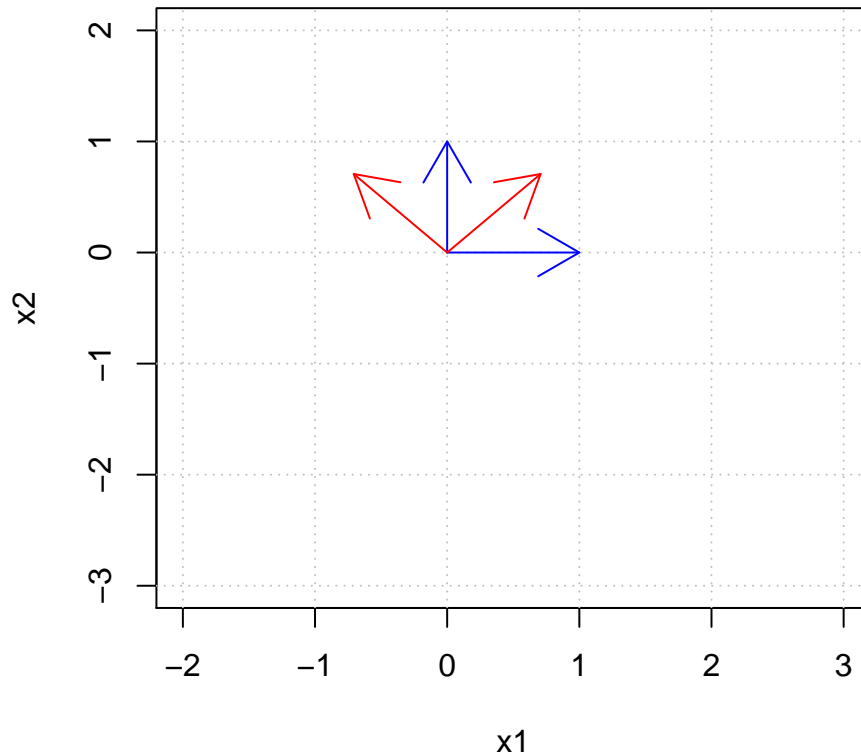
- (b) Calculer les images des vecteurs de base $\mathbf{f}_1 = \theta(\mathbf{e}_1)$ et $\mathbf{f}_2 = \theta(\mathbf{e}_2)$ par l'application θ et représenter ces 4 vecteurs dans un plan.

```
# Vecteurs de la base canonique de R2
e1 = c(1,0)
e2 = c(0,1)

# Image des vecteurs de base
f1 = O%%e1
f2 = O%%e2

# Création d'un plan et ajout d'une grille
plot(NA, xlim=c(-2,3), ylim=c(-3,2), xlab="x1", ylab="x2")
grid(col = "gray", lty = "dotted", lwd = 1)

# Représentation des vecteurs dans le plan
arrows(0,0,e1[1],e1[2],col="blue")
arrows(0,0,e2[1],e2[2],col="blue")
arrows(0,0,f1[1],f1[2],col="red")
arrows(0,0,f2[1],f2[2],col="red")
```



(c) Que pouvez vous-dire sur l'application θ ?

L'application θ est une rotation d'angle $\pi/4$ (ou 45° degrés)

15. On considère enfin les matrices P et D définies par

$$P = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & -1/3 \\ -1 & 1 & 1/2 \end{pmatrix} \quad \text{et} \quad D = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

(a) Définir ces deux matrices sous .

```
P = rbind(c(1,1,0),c(1,0,-1/3),c(-1,1,1/2))
D = diag(c(1,2,2))
```

(b) La matrice P est-elle inversible ?

```
det(P)
## [1] 0.1666667
```

La matrice P est donc inversible.

- (c) Calculer le produit $A = PDP^{-1}$.

```
A = P%*%D%*%solve(P)
A
##           [,1] [,2] [,3]
## [1,] -4.440892e-16  3    2
## [2,] -2.000000e+00  5    2
## [3,]  2.000000e+00 -3    0
```

- (d) Que peut-on dire des vecteurs colonnes de P et des éléments de D vis à vis de la matrice A ?

La matrice D contient les valeurs propres de la matrice A et les colonnes de la matrice P sont formées par les vecteurs propres associées aux différentes valeurs propres de la matrice A .

2.3 Systèmes Linéaires

On se propose ici de voir comment on peut résoudre certains systèmes linéaires directement à partir de \mathbb{R} . Plus précisément, les systèmes linéaires de Cramer. Pour cela, on considère le système d'équations défini par

$$\begin{cases} x - 2y + 2z = 3 \\ 3x - 2z = -7 \\ -x + y + z = 6 \end{cases}$$

On rappelle que l'on peut également écrire ce système sous la forme matricielle suivante

$$A\mathbf{x} = \mathbf{b}$$

1. Définir la matrice A et le vecteur b associés à ce système et créer les objets correspondants sur \mathbb{R} .

```
A = rbind(c(1,-2,2),c(3,0,-2),c(-1,1,1))
b = c(3,-7,6)
```

2. Montrer que ce système est un système de Cramer.

Il suffit de vérifier que le déterminant de A est non nul.

```
det(A)
## [1] 10
```

Nous avons bien un système de Cramer.

3. On se propose maintenant de résoudre ce système linéaire. Nous pouvons le faire directement à l'aide de la commande suivante

```
solve(A,b)
## [1] 0.2 2.4 3.8
```

Résoudre le système.

4. Retrouver les solutions de ce système à l'aide des formules de Cramer.

On applique nos formules de cramer pour déterminer les solutions du système.

```
# Calcul de x
x = det(cbind(b,A[,2:3]))/det(A)
x
## [1] 0.2

# Calcul de y
y = det(cbind(A[,1],b,A[,3]))/det(A)
y
## [1] 2.4

# Calcul de z
z = det(cbind(A[,1:2],b))/det(A)
z
## [1] 3.8
```

5. Sachant que la solution d'un système de Cramer $A\mathbf{x} = \mathbf{b}$ est donnée par $\mathbf{x} = A^{-1}\mathbf{b}$ et que la matrice inverse de A est donnée par

```
solve(A)
```

Déterminer une troisième façon de trouver les solutions de ce système avec .


```
solve(A)%*%b
##      [,1]
## [1,]  0.2
## [2,]  2.4
## [3,]  3.8
```

2.4 Diagonalisation

On va maintenant regarder comment on peut aisément déterminer les valeurs et vecteurs propres d'une matrice sous [R](#).

On considère la matrice B donnée par

$$B = \begin{pmatrix} 2 & 0 & 4 \\ 3 & -4 & 12 \\ 1 & -2 & 5 \end{pmatrix}$$

1. Définir la matrice B dans [R](#).

```
B = matrix(c(2,0,4,3,-4,12,1,-2,5),nrow = 3, byrow = TRUE)
```

2. Vérifiez que les vecteurs suivants sont des vecteurs propres de B

$$\mathbf{w}_1 = \begin{pmatrix} -4 \\ 3 \\ 2 \end{pmatrix}, \quad \mathbf{w}_2 = \begin{pmatrix} -4 \\ 0 \\ 1 \end{pmatrix} \quad \text{et} \quad \mathbf{w}_3 = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}$$

```
# Définition des vecteurs
w1 = c(-4,3,2)
w2 = c(-4,0,1)
w3 = c(2,1,0)

# Vérification vecteurs propres
B%*%w1 # valeur propre nulle

##      [,1]
## [1,]    0
## [2,]    0
## [3,]    0

B%*%w2 # valeur propre égale à 1

##      [,1]
## [1,]   -4
## [2,]    0
## [3,]    1
```

```
B%%w3 # valeur propre égale à 2
```

```
##      [,1]
## [1,]    4
## [2,]    2
## [3,]    0
```

3. Exécuter les lignes de codes suivantes et commenter

```
# Réduction d'un endomorphisme
eig = eigen(B)
eig$values

## [1] 2.000000e+00 1.000000e+00 5.093148e-15

eig$vectors

##      [,1]      [,2]      [,3]
## [1,] 8.944272e-01 -9.701425e-01 0.7427814
## [2,] 4.472136e-01 3.539979e-15 -0.5570860
## [3,] -1.119535e-16 2.425356e-01 -0.3713907
```

On retrouve pas les même vecteurs propres que précédemment ! Bien que l'on ait les mêmes valeurs propres.

4. Calculer la norme des vecteurs propres de B . Que constatez-vous ?

```
vecp = eig$vectors
apply(vecp^2,2,sum)

## [1] 1 1 1
```

On remarque que les vecteurs propres données par la fonction *eigen* sont des vecteurs propres de norme égale à 1.

2.5 Décomposition en valeurs singulières

On considère la matrice T suivante

$$T = \begin{pmatrix} 1 & 2 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 \\ 0 & 1 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{pmatrix}$$

1. Créer la matrice T

```
T = rbind(c(1,2,0,0,0),
          c(1,2,1,0,0),
          c(0,1,1,2,0),
          c(0,0,0,0,3))
```

2. Exécuter et comprendre les commandes suivantes

```
n = nrow(T)
n

## [1] 4

d = ncol(T)
d

## [1] 4

dim(T)

## [1] 4 5
```

Les variables n et d contiennent respectivement le nombre de lignes et de colonnes de la matrice T . La fonction `dim` renvoie la dimension de l'objet en argument.

3. Exécuter et comprendre les commandes suivantes

```
svdT = svd(T)
svdT

## $d
## [1] 3.5343154 3.0000000 2.0244549 0.6404662
##
## $u
##          [,1] [,2]          [,3]          [,4]
## [1,] -0.5716775  0  0.4170476 -0.7065806
## [2,] -0.6634348  0  0.2717257  0.6971509
## [3,] -0.4827412  0 -0.8673156 -0.1213445
## [4,]  0.0000000  1  0.0000000  0.0000000
##
## $v
##          [,1] [,2]          [,3]          [,4]
## [1,] -0.3494630  0  0.3402266 -0.01472313
## [2,] -0.8355128  0  0.2520338 -0.21890894
## [3,] -0.3242993  0 -0.2941977  0.89904264
## [4,] -0.2731738  0 -0.8568387 -0.37892536
```

```
## [5,] 0.0000000 1 0.0000000 0.0000000

sigma = svdT$d
U = svdT$u
V = svdT$v
```

La fonction *svd* effectue la décomposition en valeurs singulières de la matrice T , elle donne donc les valeurs propres de cette matrice ainsi que les vecteurs propres associés.

L'objet *svdT\$u* contient les vecteurs singuliers de la matrice TT^T et L'objet *svdT\$v* contient les vecteurs singuliers de la matrice T^TT .

4. En faisant le lien avec votre cours, dire ce que ce représente les objets T_1 et T_2 définis ci-dessous

```
T1=sigma[1]*U[,1]%*%t(V[,1])
T1

##           [,1]      [,2]      [,3]      [,4] [,5]
## [1,] 0.7060859 1.688144 0.6552430 0.5519446 0
## [2,] 0.8194165 1.959100 0.7604131 0.6405347 0
## [3,] 0.5962396 1.425518 0.5533064 0.4660782 0
## [4,] 0.0000000 0.000000 0.0000000 0.0000000 0

T2=T1+sigma[2]*U[,2]%*%t(V[,2])
T2

##           [,1]      [,2]      [,3]      [,4] [,5]
## [1,] 0.7060859 1.688144 0.6552430 0.5519446 0
## [2,] 0.8194165 1.959100 0.7604131 0.6405347 0
## [3,] 0.5962396 1.425518 0.5533064 0.4660782 0
## [4,] 0.0000000 0.000000 0.0000000 0.0000000 3
```

L'objet T_1 représente la meilleure approximation par une matrice de rang 1 de la matrice T .

De la même façon, l'objet T_2 représente la meilleure approximation par une matrice de rang 2 de la matrice T .

5. Donner les meilleures approximations (au sens de la distance de Frobenius) de rang 3 et de rang 4 de la matrice T .

```
T3=T2+sigma[3]*U[,3]%*%t(V[,3])
T3
```

```
##           [,1]      [,2]      [,3]      [,4] [,5]
## [1,]  0.993337180  1.900935  0.4068537 -0.1714793    0
## [2,]  1.006573902  2.097743  0.5985760  0.1691908    0
## [3,] -0.001144238  0.982987  1.0698709  1.9705510    0
## [4,]  0.000000000  0.000000  0.0000000  0.0000000    3

T4=T3+sigma[4]*U[,4]%*%t(V[,4])
T4

##           [,1] [,2]      [,3]      [,4] [,5]
## [1,]  1.000000e+00      2  4.440892e-16  5.551115e-16    0
## [2,]  1.000000e+00      2  1.000000e+00 -1.665335e-16    0
## [3,]  4.638217e-16      1  1.000000e+00  2.000000e+00    0
## [4,]  0.000000e+00      0  0.000000e+00  0.000000e+00    3
```

Que remarquez vous en particulier ce qui concerne la meilleure approximation de rang 4.

La matrice T_4 est exactement égale à la matrice T .

6. Stocker dans la variable Z , la matrice issue du calcul TT^T .

```
Z = T%*%t(T)
```

Puis stocker dans la variable *eigZ* le résultat de la décomposition spectrale de cette matrice.

```
# Décomposition en valeurs propres
eigZ = eigen(Z)
vecpz = eigZ$vectors
valpz = eigZ$values
```

Etudier les éléments propres de cette matrice.

```
# Vecteurs propres
vecpz

##           [,1] [,2]      [,3]      [,4]
## [1,] -0.5716775      0 -0.4170476  0.7065806
## [2,] -0.6634348      0 -0.2717257 -0.6971509
## [3,] -0.4827412      0  0.8673156  0.1213445
## [4,]  0.0000000      1  0.0000000  0.0000000

# Valeurs propres
valpz
```

```
## [1] 12.491386  9.000000  4.098417  0.410197
```

```
sqrt(valpz)
```

```
## [1] 3.5343154 3.0000000 2.0244549 0.6404662
```

Faire le lien avec la décomposition en valeurs singulières de la matrice T .

7. Faire de même avec la matrice $T^T T$ que vous stockerez dans une matrice Y .

```
Y = t(T)%*%T
```

On peut ensuite étudier les valeurs et propres et vecteurs propres de cette matrice.

```
# Décomposition en valeurs propres
```

```
eigY = eigen(Y)
```

```
vecpy = eigY$vectors
```

```
valpy = eigY$values
```

Etudier les éléments propres de cette matrice.

```
# Vecteurs propres
```

```
vecpy
```

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

```
## [1,] -0.3494630  0  0.3402266  0.01472313  8.728716e-01
```

```
## [2,] -0.8355128  0  0.2520338  0.21890894 -4.364358e-01
```

```
## [3,] -0.3242993  0 -0.2941977 -0.89904264 -5.689893e-16
```

```
## [4,] -0.2731738  0 -0.8568387  0.37892536  2.182179e-01
```

```
## [5,]  0.0000000  1  0.0000000  0.00000000  0.000000e+00
```

```
# Valeurs propres
```

```
valpy
```

```
## [1] 1.249139e+01 9.000000e+00 4.098417e+00 4.101970e-01 1.433776e-15
```

```
sqrt(valpy)
```

```
## [1] 3.534315e+00 3.000000e+00 2.024455e+00 6.404662e-01 3.786524e-08
```


8. Faire le lien entre les éléments propres de X et Y et la décomposition en valeurs singulières de T .

Faire le lien entre décompositions en valeurs propres et décomposition en valeurs singulières.

	Q1	Q2	Q3	Q4	Q5
Individu 1	3	3	3	5	5
Individu 2	2	3	1	5	4
Individu 3	2	3	3	4	5
Individu 4	1	1	1	1	1
Individu 5	5	5	4	3	3
Individu 6	4	5	5	2	3
Individu 7	5	5	5	3	3
Individu 8	1	1	1	1	1

TABLE 1 – Résultats du questionnaire sur un ensemble de 8 individus.

2.6 Un peu de statistiques et de géométrie

Cette dernière section se propose de faire quelques manipulations statistiques sous  afin de préparer le prochain TP.

Pour cela, on considère le jeu de données présenté en Table 1. C'est le jeu de données présenté dans l'introduction de ce cours. Pour rappel, pour obtenir ces données, nous avons demandé aux étudiants qui suivent un cours s'ils en sont satisfaits, à travers 5 critères sur lesquels ils doivent se positionner sur une échelle de 1 à 5, 1 correspondant à "très insatisfait" et 5 à "très satisfait". Voici ces 5 critères :

1. Clarté du cours écrit
2. Fluidité de la lecture du cours écrit
3. Facilité à comprendre les exemples du cours
4. Clarté des vidéos
5. Satisfaction vis à vis de l'enseignant dans la vidéo

1. Créer la matrice correspondante à ce jeu de données sous .

```
data <- rbind(c(3,3,3,5,5),
              c(2,3,1,5,4),
              c(2,3,3,4,5),
              c(1,1,1,1,1),
              c(5,5,4,3,3),
              c(4,5,5,2,3),
              c(5,5,5,3,3),
              c(1,1,1,1,1))
```

2. La fonction *mean* permet de calculer la moyenne des éléments d'un vecteur, (*i.e.* d'un échantillon. Calculer la moyenne obtenue à chaque question.


```
# Calcul de la moyenne sur une colonne
m1 = mean(data[,1])
m2 = mean(data[,2])
m3 = mean(data[,3])
m4 = mean(data[,4])
m5 = mean(data[,5])

# On peut aussi faire cela en une ligne
m = apply(data,2,mean)
```

3. La fonction `var` permet de calculer la variance des éléments d'un vecteur, (*i.e.* d'un échantillon). Calculer la variance de l'échantillon associée à la variable.

```
# Calcul de la variance sur une colonne
v1 = var(data[,1])
v2 = var(data[,2])
v3 = var(data[,3])
v4 = var(data[,4])
v5 = var(data[,5])

# On peut aussi faire cela en une ligne
variance = apply(data,2,var)
# puis écart-type
std = sqrt(apply(data,2,var))
```

Remarque :  calcule la variance **débiaisée**. On fera attention dans la suite, notamment lors de la séance sur l'ACP à biaiser cette estimation comme cela est d'usage pour l'ACP.

4. A l'aide de la fonction `var` et à l'aide de la définition de la variance. Que constatez vous ?

Illustrons la remarque précédente avec le code ci-dessous sur notre première variable :

```
# Variance biaisée
mean(data[,1]^2) - mean(data[,1])^2

## [1] 2.359375

# Variance débiaisée
n = nrow(data)
sum((data[,1]-mean(data[,1]))^2)/(n-1)

## [1] 2.696429
```



```
# Fonction "var" de R
var(data[,1])

## [1] 2.696429
```

5. Déterminer la matrice de variance-covariance de votre jeu de données (c'est une matrice de taille 5×5) à l'aide de la fonction *cov*.

```
cov_data = cov(data)
cov_data

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 2.6964286 2.6071429 2.5535714 0.5714286 0.7321429
## [2,] 2.6071429 2.7857143 2.6071429 0.8571429 1.1071429
## [3,] 2.5535714 2.6071429 2.9821429 0.2857143 0.8750000
## [4,] 0.5714286 0.8571429 0.2857143 2.5714286 2.2857143
## [5,] 0.7321429 1.1071429 0.8750000 2.2857143 2.4107143
```

Attention!  calcule à nouveau la covariance **débiaisée**.

Recalculer cette matrice là à l'aide des opérations précédentes.

```
# Centrage des données
data_cen = t(t(data)-m)
data_cen

##           [,1] [,2] [,3] [,4] [,5]
## [1,] 0.125 -0.25 0.125 2 1.875
## [2,] -0.875 -0.25 -1.875 2 0.875
## [3,] -0.875 -0.25 0.125 1 1.875
## [4,] -1.875 -2.25 -1.875 -2 -2.125
## [5,] 2.125 1.75 1.125 0 -0.125
## [6,] 1.125 1.75 2.125 -1 -0.125
## [7,] 2.125 1.75 2.125 0 -0.125
## [8,] -1.875 -2.25 -1.875 -2 -2.125

# Matrice des covariances
(1/(n-1))*t(data_cen)%*%data_cen

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 2.6964286 2.6071429 2.5535714 0.5714286 0.7321429
## [2,] 2.6071429 2.7857143 2.6071429 0.8571429 1.1071429
## [3,] 2.5535714 2.6071429 2.9821429 0.2857143 0.8750000
## [4,] 0.5714286 0.8571429 0.2857143 2.5714286 2.2857143
## [5,] 0.7321429 1.1071429 0.8750000 2.2857143 2.4107143
```

6. Déterminer la matrice de corrélation de votre jeu de données (c'est une matrice de taille 5×5) à l'aide de la fonction *cor*.

```
# Calcul de la matrice des corrélations linéaires
cor_data = cor(data)
cor_data

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.9512662 0.9005116 0.2170103 0.2871630
## [2,] 0.9512662 1.0000000 0.9045495 0.3202563 0.4272308
## [3,] 0.9005116 0.9045495 1.0000000 0.1031764 0.3263405
## [4,] 0.2170103 0.3202563 0.1031764 1.0000000 0.9180405
## [5,] 0.2871630 0.4272308 0.3263405 0.9180405 1.0000000
```

Recalculer cette matrice là à l'aide des opérations précédentes.

```
# Réduction des données
data_scaled = t(t(data_cen)/std)
data_scaled

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 0.07612294 -0.1497862 0.07238454 1.2472191 1.20761473
## [2,] -0.53286058 -0.1497862 -1.08576803 1.2472191 0.56355354
## [3,] -0.53286058 -0.1497862 0.07238454 0.6236096 1.20761473
## [4,] -1.14184410 -1.3480756 -1.08576803 -1.2472191 -1.36863003
## [5,] 1.29408998 1.0485032 0.65146082 0.0000000 -0.08050765
## [6,] 0.68510646 1.0485032 1.23053710 -0.6236096 -0.08050765
## [7,] 1.29408998 1.0485032 1.23053710 0.0000000 -0.08050765
## [8,] -1.14184410 -1.3480756 -1.08576803 -1.2472191 -1.36863003

# Calcul de la matrice des corrélations linéaires
(1/(n-1))*t(data_scaled)%*%data_scaled

##           [,1]      [,2]      [,3]      [,4]      [,5]
## [1,] 1.0000000 0.9512662 0.9005116 0.2170103 0.2871630
## [2,] 0.9512662 1.0000000 0.9045495 0.3202563 0.4272308
## [3,] 0.9005116 0.9045495 1.0000000 0.1031764 0.3263405
## [4,] 0.2170103 0.3202563 0.1031764 1.0000000 0.9180405
## [5,] 0.2871630 0.4272308 0.3263405 0.9180405 1.0000000
```

7. Calculer la distance entre les individus 1 et 2 puis entre les individus 1 et 6.

On le fera ici en prenant la distance euclidienne entre les individus concernés.

```
# Distance entre les individus 1 et 2
dist_12 = sqrt( sum(data[1,]-data[2,])^2)
dist_12
```

```
## [1] 4

# Distance entre les individus 1 et 6
dist_16 = sqrt( sum(data[1,]-data[6,])^2)
dist_16

## [1] 0
```