

TP 2 : Calcul vectoriel

Merci de lire la fiche de TP en détail. Avant d'appeler le responsable du TP, vérifier **systématiquement** :

- que vous avez lu le message d'erreur (et traduit s'il est partiellement en anglais),
- que vous avez bien lu la fiche en détail,
- que votre code est bien formaté (espace, retour à la ligne, etc.) et qu'il n'y a pas de fautes de frappe,
- et que vous avez cherché dans les pages d'aide de R.

Les lignes qui commencent par `>` dans ce fichier Rmd sont des lignes qui correspondent à *des questions qui vous sont posées*.

1 Quelques éléments sur RMarkdown

1.1 Format de sortie

Ce fichier que vous lisez est un fichier de type `.Rmd` (ou Rmarkdown). On peut l'utiliser pour produire (le verbe choisi par les programmeurs de cette solution est `knit` en anglais, pour tricoter) :

- une page web (HTML)
- un fichier PDF
- un fichier word

Dans ce cas, à chaque fois que l'on demande de tricoter un fichier de ce type, R refait tous les calculs des chunks, en utilisant comme "working directory" ou répertoire de travail celui dans lequel ce fichier est enregistré.

On peut également utiliser un mode "Preview", qui correspond à un Rnotebook. Dans ce cas, R produit une page web, et se contente d'afficher les résultats des chunks qui ont déjà été évalués. L'enregistrement du fichier produit la mise à jour de la page HTML.

Noter que l'on passe d'un mode à l'autre de sortie en cliquant sur le petit triangle noir, à droite de Preview ou Knit dans les boutons de cette fenêtre.

- Essayer les différents modes sur ce fichier. Constater les sorties.

1.2 Fenêtres de RStudio

À l'aide du petit triangle à côté de la roue dans les boutons de cette fenêtre, on peut mettre l'aperçu du mode Preview dans une fenêtre à part de la fenêtre R ou bien dans l'onglet "Viewer" dans la partie inférieure droite de RStudio, en choisissant entre "Preview in Window" ou "Preview in Viewer Pane".

- Passer en mode "Preview in Viewer Pane".

On peut aussi paramétrer pour que les résultats aillent dans la console plutôt que sous le "chunk" (bloc R), toujours à partir du même triangle, en choisissant entre "Chunk Output inline" ou "Chunk Output in Console".

Noter également que vous pouvez détacher la fenêtre de ce fichier `.Rmd` avec le bouton qui contient une petite fenêtre et une flèche.

Je vous conseille, pour respecter la règle de lignes qui ne dépassent pas 80 caractères, d'ajouter la barre verticale à 80 caractères dans cet éditeur de texte. Pour cela, vous devez aller dans les préférences de R et dans la partie “Code”, onglet “Display”, cocher “Show margin” et saisissez 80 dans la case “Margin column”.

Dans l'onglet “Diagnostics” au même endroit, je vous conseille de cocher “Provide R style diagnostics (e.g. whitespace)” afin de vous aider à respecter les règles de styles standard de R.

- Réaliser ces réglages<.

Attention, suivant les versions de RStudio, ces choix de paramétrages peuvent ne pas exister...

1.3 Raccourcis clavier

Lors du TP 1, nous avons vu :

- Ctrl + Entrée pour l'exécution
- Ctrl + Maj + Entrée pour executer tout le chunk.

Deux autres raccourcis sont très utiles quand on manipule des fichiers Rmd :

- Pour enregistrer : Ctrl + S
- Pour insérer un chunk : Alt + Ctrl + I

Comme d'habitude, vous pouvez les retrouver en laissant la souris sur les boutons de cette fenêtre qui réalise cette action, jusqu'à ce qu'une bulle d'aide apparaisse.

1.4 Inclusion de formules mathématiques avec LaTeX

On peut inclure des formules mathématiques avec LaTeX. Par exemple : $x^2 + y^2 = 1$ est l'équation d'un cercle dans le plan euclidien. On peut aussi inclure des formule plus compliquée. Par exemple, la variable aléatoire X suit la loi gaussienne centrée réduite $\mathcal{N}(0, 1)$ si et seulement si, pour tout A borélien de \mathbb{R} ,

$$\mathbb{P}(X \in A) = \int_{x \in A} \frac{1}{\sqrt{2\pi}} \exp(-x^2) dx.$$

2 Vecteurs, data.frame et matrices

2.1 Vecteurs

Les types de base en R sont des vecteurs :

- de nombres réels (**numeric** ou **double**),
- de nombres complexes (**complexe**),
- de nombres entiers (**integer**)
- de chaînes de caractères (**character**),
- de facteurs discrets (**factor** et **ordered**),
- de vrai/faux (**logical**).

Il existe une valeur particulière en R, qui s'appelle “NA” et correspond à “non observé”. (Il ne faut pas confondre NA avec NaN, not a number, qui correspond à un calcul qui n'est pas faisable. Par exemple $\log(-1)$)

Le bloc de code ci-dessous permet de créer 5 vecteurs :

```

set.seed(1234) # ne pas chercher à comprendre cette ligne...
v1 <- round(runif(15, min = -10, max = 20), digits = 2)
v2 <- round(runif(15, min = -10, max = 20), digits = 2) +
  1i * round(runif(15, min = -10, max = 20), digits = 2)
v3 <- c(1L:4L, 19L, -3L, sample(c(-7L, 17L, 65L), 9, replace = TRUE))
v4 <- sample(c("riri", "fifi", "loulou"), 15, replace = TRUE)
v5 <- as.factor(v3)
v6 <- sample(c(TRUE, FALSE), 15, TRUE)

```

Répondre aux questions ci-dessous, en utilisant le cas échéant le chunk ci-dessous :

- Que font les commandes du chunk ci-dessus ?
- Afficher le contenu de chacun de ces vecteurs.
- Quel est le type de chacun des vecteurs ? (On pourra s'aider de la fonction `class` pour déterminer le type de l'objet.)
- Un certain nombre de fonctions de R s'adaptent au type de l'objet passé en paramètre. C'est le cas de la fonction `summary`. Appliquer cette fonction à chacun des vecteurs, et constater comment les sorties diffèrent d'un vecteur à l'autre, suivant le type.
- La fonction `length` renvoie la longueur du vecteur. La tester.

Indexation d'un vecteur

Que renvoie `v[iii]` si `v` est un vecteur lorsque l'on change le contenu ou le type de l'objet `iii` ?

- Si `iii` est un vecteur d'entiers positifs, on obtient le sous-vecteur correspondant aux coordonnées dont les numéros sont dans `iii`.
- Si `iii` est un vecteur d'entiers négatifs, on obtient le sous-vecteur de `v` qui exclut les coordonnées dont les numéros sont dans `-iii`.
- Si `iii` est un vecteur logique de même longueur que `v`, alors on obtient le sous-vecteur de `v` où `v[i]` est présent si et seulement si `iii[i]` est égal à `TRUE`.
- Comprendre ce que font les commandes ci-dessous.

```
v1[c(1, 24, 6, 7)]
```

```
## [1] -6.59    NA   9.21 -9.72
```

```
v2[24]
```

```
## [1] NA
```

```
v4[v6]
```

```
## [1] "fifi"    "fifi"    "loulou" "riri"    "riri"    "fifi"    "loulou" "fifi"
```

```
v1[v1 > 0]
```

```
## [1] 8.67 8.28 8.70 15.83 9.21 9.98 5.43 10.81 6.35 17.70
```

Pour les vecteurs numériques, on peut aussi faire des additions avec `+`, des multiplication termes à termes avec `*`, des divisions termes à termes avec `/`. Les deux opérandes de ces opérations sont censés être de mêmes longueurs. Lorsque ce n'est pas le cas, R parcourt le vecteur le plus court plusieurs fois pour recycler les valeurs des coordonnées (règle de recyclage).

- Comprendre les calculs ci-dessous. Quelle est la règle de recyclage utilisée, et comment ?
- Quelles sont les modifications de types faites automatiquement ?

```
v1 + v2
```

```
## [1] 8.53+ 3.68i 7.26- 2.04i 6.28- 0.86i 4.30+ 5.22i 12.80- 4.57i
## [6] 8.71+12.79i -10.64- 3.96i -8.25- 2.24i 1.18+19.76i 1.99+14.22i
```

```
## [11] 25.13+ 6.60i 12.12+ 9.39i 15.92- 0.65i 32.64+ 8.65i -9.86- 0.11i
v3 * v6

## [1] 0 2 0 4 19 0 65 0 -7 0 17 0 0 17 -7
v3 * (v3 > 0)

## [1] 1 2 3 4 19 0 65 65 0 65 17 17 65 17 0
v3 * (v3 <= 0)

## [1] 0 0 0 0 0 -3 0 0 -7 0 0 0 0 0 -7
v1 + 5000.3

## [1] 4993.71 5008.97 5008.58 5009.00 5016.13 5009.51 4990.58 4997.28 5010.28
## [10] 5005.73 5011.11 5006.65 4998.78 5018.00 4999.07
v3 + c(100, 200, 300)

## [1] 101 202 303 104 219 297 165 265 293 165 217 317 165 217 293
v3 + c(-1000, -2000)

## Warning in v3 + c(-1000, -2000): la taille d'un objet plus long n'est pas
## multiple de la taille d'un objet plus court
## [1] -999 -1998 -997 -1996 -981 -2003 -935 -1935 -1007 -1935 -983 -1983
## [13] -935 -1983 -1007
diff(v3)

## [1] 1 1 1 15 -22 68 0 -72 72 -48 0 48 -48 -24


- Que font les fonctions sum, cumsum, prod, diff, mean, sd et which ? (lire les pages d'aide)
- Comprendre le résultat des commandes ci-dessous.
- Quelles sont les modifications de types faites automatiquement ?


sum(v1)

## [1] 78.88
mean(v1)

## [1] 5.258667
mean(v2)

## [1] 1.948667+4.392i
sd(v1)

## [1] 7.975099
prod(v3)

## [1] -5.878714e+15
prod(v6)

## [1] 0
cumsum(v6)

## [1] 0 1 1 2 3 3 4 4 5 5 6 6 6 7 8
```

```
diff(v3)
```

```
## [1] 1 1 1 15 -22 68 0 -72 72 -48 0 48 -48 -24
```

```
which(v6)
```

```
## [1] 2 4 5 7 9 11 14 15
```

```
which(v1 <= 0)
```

```
## [1] 1 7 8 13 15
```

La règle de recyclage s'applique aussi dans `v[iii]` quand le vecteur d'indices `iii` est de type `logical`, et plus court que la longueur de `v`.

- Que font les deux commandes ci-dessous ?

```
v1[c(TRUE, FALSE)]
```

```
## [1] -6.59 8.28 15.83 -9.72 9.98 10.81 -1.52 -1.23
```

```
v5[c(TRUE, FALSE, FALSE)]
```

```
## [1] 1 4 65 65 65
```

```
## Levels: -7 -3 1 2 3 4 17 19 65
```

On peut modifier la partie indexée, en mettant à gauche de `<-` la partie que l'on veut modifier. Et à droite de `<-`, les valeurs que l'on veut mettre dans ces cases. (À nouveau, la règle de recyclage s'applique s'il n'y pas assez de valeurs à droite...)

- Comprendre les commandes ci-dessous et justifier le contenu de `yyy` et `zzz`.

```
yyy <- v3
```

```
yyy[yyy < 0] <- 0
```

```
yyy
```

```
## [1] 1 2 3 4 19 0 65 65 0 65 17 17 65 17 0
```

```
zzz <- v6
```

```
zzz[v6] <- FALSE
```

```
zzz
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
## [13] FALSE FALSE FALSE
```

On peut aussi créer des vecteurs avec la fonction `rep`.

- Lire la page d'aide de `rep`.
- Comprendre les lignes de code ci-dessous.

```
rep(c(1, 2, 3), each = 4)
```

```
## [1] 1 1 1 1 2 2 2 2 3 3 3 3
```

```
rep(c(1, 2, 3), times = 4)
```

```
## [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

```
rep(c("titi", "toto", 24), times = 2)
```

```
## [1] "titi" "toto" "24" "titi" "toto" "24"
```

```
rep(0, 24)
```

```
## [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Cette fonction est très utilisée pour créer des vecteurs de longueur donnée, et les initialiser à 0.

2.2 Un exemple de `data.frame`

Une table de données est un objet de type `data.frame` en R. Les différentes variables sont rangées dans des colonnes alors que les lignes correspondent aux différentes observations. Les lignes et les colonnes sont numérotées à partir de 1. Elles peuvent avoir des noms.

On peut facilement extraire une sous-table de données ou bien un vecteur colonne depuis une table de données. Ainsi, `NomDeLaTable$NomDeLaColonne` permet d'obtenir la colonne dont le nom est `NomDeLaColonne` de la table dont le nom est `NomDeLaTable`. Cela renvoie un vecteur. (Attention, R est sensible à la différence majuscule/minuscule~!)

On peut aussi utiliser les crochets `[]` pour extraire un nombre, un vecteur ou une partie de la table. La convention est `[ligne, colonne]`. Le mieux est de donner des exemples.

Supposons que l'on dispose d'une table de données (`data.frame`) qui s'appelle `hills` et qui a les propriétés suivantes :

- elle est formée de 35 lignes et 3 colonnes,
- les colonnes numérotées 1, 2 et 3 s'appellent respectivement `dist`, `climb` et `time`.

Les fonctions `nrow` (number of rows), `ncol` (number of columns) et `dim` renvoient le nombre de lignes, de colonnes et la dimension de la matrice. Voici un exemple.

```
library(MASS)
data(hills)
?hills
nrow(hills)
```

```
## [1] 35
```

```
ncol(hills)
```

```
## [1] 3
```

```
dim(hills)
```

```
## [1] 35  3
```

Sur cette table, on a que :

- `hills$climb` renvoie la colonne `climb` de la table `hills` sous forme de vecteur
- `hills[6, 2]` renvoie le nombre dans la ligne 6 et la colonne 2
- `hills[, 2]` renvoie la deuxième colonne (Le numéro de ligne n'est pas spécifié, il conserve donc toutes les lignes. Le résultat est un objet à un seul indice (qui correspond au numéro de ligne), donc un vecteur.)
- `hills[6,]` renvoie la sixième ligne
- `hills[1:4, 2]` renvoie les nombres dans les lignes de 1 à 4 et sur la seconde colonne
- `hills[6, 1:2]` renvoie les nombres sur la ligne 6, dans les colonnes 1 et 2.
- `hills[1:4, 1:2]` renvoie une sous-table ne contenant que les lignes de 1 à 4 et les colonnes de 1 à 2.
- Essayer tous ces exemples sous R.

Les fonctions `is.data.frame`, `is.numeric`, `is.integer`, `is.character`, `is.factor`,... permettent de tester si un objet est un `data.frame`, un vecteur numérique, ou un vecteur d'entiers, ou un vecteur de chaînes de caractères, ou un vecteur de modalités...

- Pour chacun des exemples que vous avez essayé dans le chunk précédent, vérifier quel est le type de l'objet renvoyé avec les fonctions `is.***`.

La table de données `anorexia` du package `MASS` contient des colonnes de types différents. Elle est chargée au début du chunk ci-dessous.

- En utilisant la fonction `class` sur chacune des colonnes, trouver le type de chacune d'entre elles
- Que renvoie la fonction `summary` si on l'applique sur `anorexia` tout entier ?
- Constater que la longueur de chacune des colonnes est identique.

```
data(anorexia)
class(anorexia$Treat)
```

```
## [1] "factor"
```

```
class(anorexia$Prewt)
```

```
## [1] "numeric"
```

```
class(anorexia$Postwt)
```

```
## [1] "numeric"
```

```
# Répondre aux questions ci-dessus dans les lignes qui suivent :
```

- Que font les commandes ci-dessous ?

```
hills[hills$dist <= 5.2, c(2,3)]
```

```
##           climb    time
## Greenmantle    650 16.083
## Scolty         800 29.750
## Dollar        2000 43.050
## Eildon Two     1500 26.933
## Knock Hill     350 78.650
## Black Hill     1000 17.417
## Kildcon Hill   300 15.950
## Meall Ant-Suidhe 1500 27.900
## Cow Hill       900 17.933
## N Berwick Law  600 18.683
## Creag Dubh     2000 26.217
## Largo Law      950 28.567
## Acmony         500 20.950
## Cockleroi      850 28.100
```

```
hills$time[hills$dist <= 5.2]
```

```
## [1] 16.083 29.750 43.050 26.933 78.650 17.417 15.950 27.900 17.933 18.683
## [11] 26.217 28.567 20.950 28.100
```

- Pourquoi la commande `hills[hills$dist <=5.2]` renvoie une erreur ?

```
hills[hills$dist <=5.2]
```

2.3 Matrices

Les matrices sont comme des `data.frame`, mais les colonnes sont toutes de type `numeric` (ou `complex`).

Attention, les symboles `*` et `/` désignent en R les opérations termes à termes, et non des opérations de calcul matriciel.

La multiplication matricielle, quand les dimensions le permettent, s'obtient avec `%*%`. Et l'inversion matricielle, ou la solution d'un système linéaire avec la fonction `solve`. Le déterminant se calcule avec `det`. La diagonalisation (vecteurs propres, valeurs propres) s'obtient avec `eigen`. La transposée d'une matrice avec la fonction `t` ou la fonction `aperm`. Le produit $(X')Y$, où X' désigne la transposée de la matrice X avec la fonction `crossprod`, ainsi que le produit $X'X$.

Les fonctions `nrow` (number of rows), `ncol` (number of columns) et `dim` renvoient le nombre de lignes, de colonnes et la dimension de la matrice.

On peut aussi inverser les matrices avec la fonction `ginv` du package `MASS`. Cette fonction est plus stable que la fonction `solve`, mais plus couteuse en temps de calcul. Lorsque la matrice n'est pas inversible, elle renvoie l'*inverse de Moore-Penrose* de cette matrice, qui est une généralisation de l'inversion de matrice. (Lorsque le package `MASS` n'est pas chargé avec une commande `library`, on peut accéder à cette fonction en utilisant `MASS::ginv`.)

- Lire la page d'aide de la fonction `matrix`.
- Comprendre ce que font les lignes de codes ci-dessous.
- Dans quelle ligne une règle de recyclage est-elle utilisée ?

```
set.seed(6789)
(AA <- matrix(1:6, nrow = 2))

##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6

(BB <- matrix(1:6, ncol = 2))

##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6

(CC <- matrix(1:6, nrow = 2, byrow = TRUE))

##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6

(DD <- matrix(c(11, 22, 33, 44), nrow = 2, ncol = 4))

##      [,1] [,2] [,3] [,4]
## [1,]   11   33   11   33
## [2,]   22   44   22   44

EE <- matrix(runif(4), nrow = 2)
```

Notez que les parenthèses, autour de chaque ligne, permettent d'afficher le contenu de la variable, sinon, l'affectation ne produit aucune sortie texte.

Notez que la fonction `matrix` est aussi très utilisée pour initialiser une matrice à zéro.

- Calculer le résultat des opérations suivantes en R dans le chunk ci-dessous :
 1. la multiplication `AA BB`,
 2. le déterminant et l'inverse de `EE` (par deux méthodes),
 3. les vecteurs propres et les valeurs propres de `EE`.
- Pour les produits $(AA' CC)$ et $(AA' AA)$, proposer deux méthodes, l'une avec `%*%`, et l'autre avec un appel à une seule fonction. (La technique préférable est la seconde.)
- Vérifier que le produit des valeurs propres est égal au déterminant.

- Vérifier que les vecteurs propres sont bien des vecteurs propres pour la valeur propre correspondante.

3 Développement d'un nombre en base b

La fin du sujet du TP 1 vous invitait à préparer cette partie.

On veut créer en R une fonction qui prenne un nombre x entre 0 et 1 en entrée, et renvoie un vecteur v , dont les coordonnées sont le développement en base 10 avec une précision de 8 chiffres après la virgule. Autrement dit,

$$\left| x - \sum_{k=1}^8 10^{-k} v_k \right| < 10^{-8}.$$

Le pseudo-code est dans fiche de TP 1.

3.1 Programmation naive

Dans le chunk ci-dessous, qui définit la fonction `dixRep`, programmer le pseudo-code de la fiche de TP 1.

```
dixRep <- function(x) {
  v <- rep(0, 8)
  # Mettre ici le code, avec une boucle for
  return(v)
}
```

On peut ensuite essayer cette fonction sur différents nombres entre 0 et 1, et vérifier que les résultats sont ceux attendus.

```
dixRep(pi / 10)
```

```
## [1] 0 0 0 0 0 0 0 0
```

```
dixRep(0.12345678)
```

```
## [1] 0 0 0 0 0 0 0 0
```

```
dixRep(0.12345678901)
```

```
## [1] 0 0 0 0 0 0 0 0
```

```
dixRep(0.22222222)
```

```
## [1] 0 0 0 0 0 0 0 0
```

Malheureusement, ce code contient une boucle `for` et n'est donc pas efficace en temps de calcul.

3.2 Programmation vectorielle

On introduit le vecteur \mathbf{p} de dimension 8, dont les coordonnées sont $p_i = 10^i$, pour tout i .

- Si $x = \sum_i v_i 10^{-i}$, que vaut le vecteur $\text{trunc}(x\mathbf{p})/\mathbf{p}$, où $/$ représente la division termes à termes entre vecteurs ?
- Comment peut-on récupérer les valeurs de v_i à partir des coordonnées de ce vecteur ? (Indication : faire des opérations entre les coordonnées de ce vecteur.)

- En déduire une programmation vectorielle, sans boucle `for` du calcul de v , sous forme de pseudo-code.
- Implémenter cette méthode dans le chunk ci-dessous.

```
dixRep2 <- function(x) {
  p <- 10 ^ (1:8)
  # compléter ici
}
```

3.3 Ajout d'options

On souhaite modifier le programme précédent pour obtenir le développement dans la base b , avec une précision de n chiffres. Les paramètres de la fonction seront donc : x , b et n .

- Implémenter l'algorithme dans le chunk ci-dessous.
- Mettre comme valeur par défaut $b = 10$ et $n = 8$.
- Commencer la fonction par des tests qui vérifient :
 1. que x est un vecteur numérique, et stoppe le programme si ce n'est pas le cas,
 2. que x est un vecteur de longueur 1, et émet un warning si ce n'est pas le cas, et qui remplace x par sa première coordonnée,
 3. que x est un nombre entre 0 et 1, et qui arrête la fonction si ce n'est pas le cas.

```
baseRep <- function(...) {
  # remplacer les ... ci-dessus, et compléter ici.
}
```

- Dans le chunk ci-dessous, programmer la fonction inverse, qui calcule x à partir du vecteur v et de la valeur de b . (Pour n , choisir la longueur du vecteur v .)

```
computBaseRep <- function(...) {
  # compléter ici, et changer ... ci-dessus
}
```

- Dans le chunk ci-dessous, vérifier que les deux fonctions sont à peu près justes en faisant différents tests.