TP2 Correction

Clément LAROCHE

14 janvier 2019

Exercice 1

```
V <- seq(from = 2,to = 20,by = 2)
W <- rep(-1,10)
# produit scalaire
V%*%W
```

```
##      [,1]
## [1,] -110
```

```
# construction de la matrice M
M <- cbind(V,W)
M
```

```
##        V  W
##  [1,]  2 -1
##  [2,]  4 -1
##  [3,]  6 -1
##  [4,]  8 -1
##  [5,] 10 -1
##  [6,] 12 -1
##  [7,] 14 -1
##  [8,] 16 -1
##  [9,] 18 -1
## [10,] 20 -1
```

```
# construction de la matrice H
H <- M
H[1:2,] <- 0
H
```

```
##        V  W
##  [1,]  0  0
##  [2,]  0  0
##  [3,]  6 -1
##  [4,]  8 -1
##  [5,] 10 -1
##  [6,] 12 -1
##  [7,] 14 -1
##  [8,] 16 -1
##  [9,] 18 -1
## [10,] 20 -1
```

```r
# On effectue les calculs demandés
M+2
```

```
##         V W
##  [1,]   4 1
##  [2,]   6 1
##  [3,]   8 1
##  [4,]  10 1
##  [5,]  12 1
##  [6,]  14 1
##  [7,]  16 1
##  [8,]  18 1
##  [9,]  20 1
## [10,]  22 1
```

```r
2*M
```

```
##         V  W
##  [1,]   4 -2
##  [2,]   8 -2
##  [3,]  12 -2
##  [4,]  16 -2
##  [5,]  20 -2
##  [6,]  24 -2
##  [7,]  28 -2
##  [8,]  32 -2
##  [9,]  36 -2
## [10,]  40 -2
```

```r
M*2
```

```
##         V  W
##  [1,]   4 -2
##  [2,]   8 -2
##  [3,]  12 -2
##  [4,]  16 -2
##  [5,]  20 -2
##  [6,]  24 -2
##  [7,]  28 -2
##  [8,]  32 -2
##  [9,]  36 -2
## [10,]  40 -2
```

```r
M/5
```

```
##          V    W
##  [1,] 0.4 -0.2
##  [2,] 0.8 -0.2
##  [3,] 1.2 -0.2
##  [4,] 1.6 -0.2
##  [5,] 2.0 -0.2
##  [6,] 2.4 -0.2
##  [7,] 2.8 -0.2
##  [8,] 3.2 -0.2
##  [9,] 3.6 -0.2
## [10,] 4.0 -0.2
```

```
M+H
```

```
##        V  W
##  [1,]  2 -1
##  [2,]  4 -1
##  [3,] 12 -2
##  [4,] 16 -2
##  [5,] 20 -2
##  [6,] 24 -2
##  [7,] 28 -2
##  [8,] 32 -2
##  [9,] 36 -2
## [10,] 40 -2
```

```
# on supprime les lignes demandées
M <- M[-c(4,5),]
# on inverse l'ordre des lignes de M
M <- M[sort(1:nrow(M),decreasing = TRUE),]
M
```

```
##       V  W
## [1,] 20 -1
## [2,] 18 -1
## [3,] 16 -1
## [4,] 14 -1
## [5,] 12 -1
## [6,]  6 -1
## [7,]  4 -1
## [8,]  2 -1
```

```
# on ordonne la première ligne de M de manière croissante
M[1,] <- sort(M[1,])
M
```

```
##       V  W
## [1,] -1 20
## [2,] 18 -1
## [3,] 16 -1
## [4,] 14 -1
## [5,] 12 -1
## [6,]  6 -1
## [7,]  4 -1
## [8,]  2 -1
```

Exercice 2

```
# construction de M (j'ai envie de construire une
# matrice triangulaire supérieure dont les termes sont
# aléatoires tirés entre 1 et 100)
M <- matrix(data = 0,nrow = 3,ncol = 3)
M[1,] <- ceiling(runif(n = 3,min = 0,max = 1)*100)
# runfi(3,0,1) me donne trois nombres aléatoires
# compris entre 0 et 1. Si je les multiplie par
# 100 j'ai bien trois nombres aléatoires comrpis
# entre 0 et 100. Maintenant je veux des nombres
```

```r
# entiers aléatoires. Je prends la fonction ceiling
# qui me permet d'avoir la partie entière supérieure.
# J'aurais donc des nombres entiers aléatoires qui
# prennent leur valeur dans 1,2,...,100
M[2,2:3] <- ceiling(runif(n = 2,min = 0,max = 1)*100)
M[3,3] <- ceiling(runif(n = 1,min = 0,max = 1)*100)
# je les stocke aux endroits appropriés dans ma
# matrice pour qu'elle soit triangulaire supérieure.

# calcul du déterminant
det(M)
```

```
## [1] 5760
```

```r
# calcul de son inverse
H <- solve(M)
# on effectue les calculs demandés
H%*%M
```

```
##      [,1] [,2]          [,3]
## [1,]    1    0  3.552714e-15
## [2,]    0    1 -4.440892e-16
## [3,]    0    0  1.000000e+00
```

```r
M%*%H
```

```
##      [,1] [,2]          [,3]
## [1,]    1    0  0.000000e+00
## [2,]    0    1 -2.220446e-16
## [3,]    0    0  1.000000e+00
```

```r
# calcul du déterminant de H
det(H)
```

```
## [1] 0.0001736111
```

```r
# on le compare à l'inverse de celui de M
1/(det(H)) == det(M)
```

```
## [1] TRUE
```

```r
# on regarde si ils sont proches
1/(det(H))
```

```
## [1] 5760
```

```r
det(M)
```

```
## [1] 5760
```

```r
# oui ils sont proches mais on voit que les calculs se font de manière approchée
```

Exercice 3

Pour construire la matrice, on considère des matrices M de la forme :

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

Pour la construire avec les valeurs propres demandées, on va utiliser le polynome caractéristique de la matrice. Il faut que 1 et 0 soient des racines du polynôme suivant :

$$(a_{11} - \lambda)(a_{22} - \lambda) - a_{12}a_{21} = \lambda^2 - (a_{11} + a_{22})\lambda + a_{11}a_{22} - a_{12}a_{21}$$

Pour rappel, ce polynôme correspond au déterminant de la matrice :

$$\begin{pmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{pmatrix}$$

Pour que 0 et 1 soient racines de ce polynome, on peut dégager deux conditions suffisantes qui sont les suivantes :

- $a_{11}a_{22} - a_{12}a_{21} = 0$
- $a_{11} + a_{22} = 1$

On remarque que, par exemple la matrice suivante satisfait ces conditions :

$$\begin{pmatrix} 0.5 & 0.25 \\ 1 & 0.5 \end{pmatrix}$$

Nous allons donc l'utiliser.

```
# construction de M
M <- matrix(data = c(0.5,0.25,1,0.5),nrow = 2,ncol = 2,byrow = TRUE)
M
```

```
##      [,1] [,2]
## [1,]  0.5 0.25
## [2,]  1.0 0.50
```

```
# calcul de ses valeurs propres
x <- eigen(M)
x$values
```

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

Notre matrice M est donc un bon candidat. Calculons maintenant la base du noyau de M. La matrice échelonnée réduite de M est :

$$\begin{pmatrix} 0.5 & 0.25 \\ 0 & 0 \end{pmatrix}$$

Quand on cherche à calculer le noyau de cette matrice, on obtient que :

$$\begin{pmatrix} 0.5 & 0.25 \\ 0 & 0 \end{pmatrix} X = 0 \iff -2x_1 = x_2$$

Donc la base du noyau de la matrice échelonnée de M et donc de M est $Vect(\begin{bmatrix} 1 \\ -2 \end{bmatrix})$. Pour ce qui est de la base de l'image de M, on se rend vite compte que c'est l'ensemble $Vect(\begin{bmatrix} 1 \\ 2 \end{bmatrix})$

```
# on crée un vecteur X quelconque de R^2
X <- ceiling(runif(n = 2)*100)
X
```

```
## [1] 28 29
```

```
# on effectue les calculs demandés
M%*%X
```

```
##       [,1]
## [1,] 21.25
## [2,] 42.50
```

```
M%*%M%*%X
```

```
##       [,1]
## [1,] 21.25
## [2,] 42.50
```

```
# remarquez que M%*%M = M
```

On prend $V_1 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ et $V_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. On va montrer que cette famille forme une base de R^2. On montre que c'est une famille de vecteurs libres. Pour se faire, soient λ_1 et λ_2 deux réels tels que :

$$\lambda_1 V_1 + \lambda_2 V_2 = 0 \iff \begin{cases} \lambda_1 + \lambda_2 = 0 \\ -2\lambda_1 + 2\lambda_2 = 0 \end{cases}$$

$$\iff \begin{cases} \lambda_1 = -\lambda_2 \\ \lambda_1 = \lambda_2 \end{cases}$$

La seule solution possible est $\lambda_1 = \lambda_2 = 0$. V_1 et V_2 forment donc une famille libre. Montrons maintenant qu'elle également génératrice.Soit $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$ un vecteur quelconque de $\mathbb{R}^2$. On voudrait écrire y comme une combinaison linéaire de V_1 et V_2. POur cela, on voudrait qu'ils existent deux scalaires λ_1 et λ_2 tels que :

$$y = \lambda_1 V_1 + \lambda_2 V_2$$

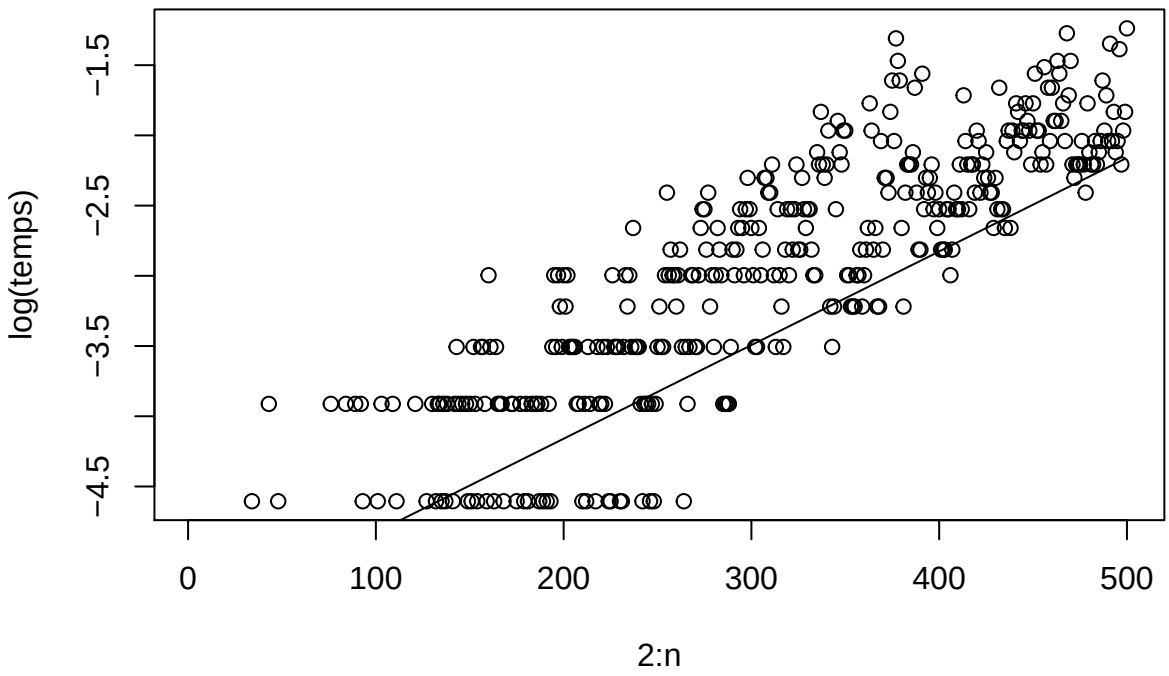
Or si on pose $\lambda_1 = \frac{2y_1 - y_2}{4}$ et $\lambda_2 = \frac{2y_1 + y_2}{4}$, on remarque que l'on retrouve bien les coordonnées de y. V_1 et V_2 forment donc bien une famille génératrice de $\mathbb{R}^2$. C'est donc une base de $\mathbb{R}^2$. Calculons les coordonnées de X dans cette base, X aurait donc pour coordonnées :

```
paste("(",as.character((2*X[1]-X[2])/4),",",as.character((2*X[1]+X[2])/4),")")
```

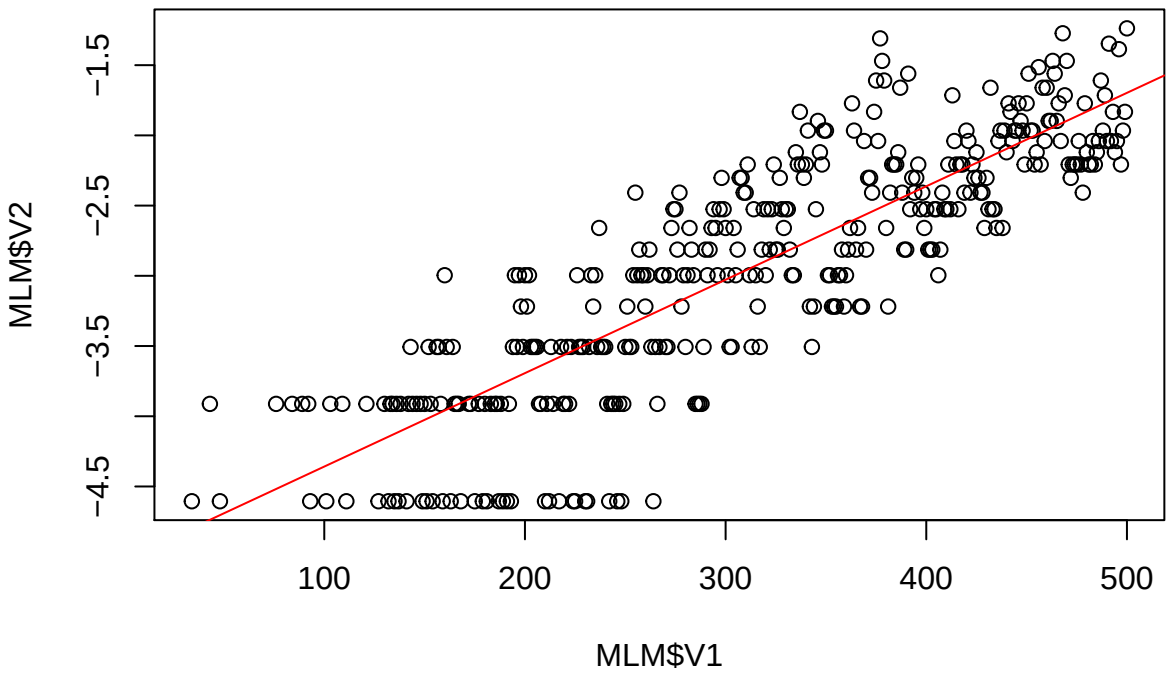
```
## [1] "( 6.75 , 21.25 )"
```

Exercice 4

```
n <- 500
temps <- c()
# création de la matrice
for(i in 2:n)
{
  M = runif(i^2)
  N <- matrix(data = M,nrow = i,ncol = i)
  tic <- proc.time()
  res <- solve(N)
  tac <- proc.time() - tic
  temps <- c(temps,as.numeric(tac[3]))
}
{
plot(2:n,log(temps))
lines((2:n/150-5.5))
}
```

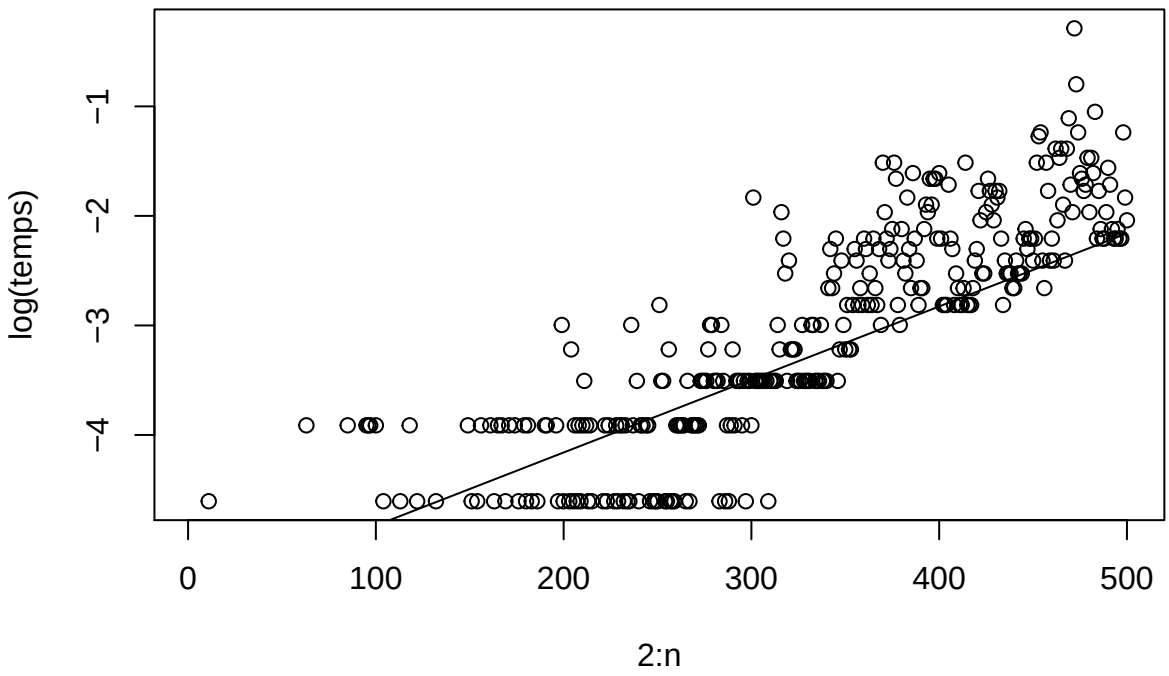



```r
# si je procède par régression linéaire
MLM <- cbind(2:n,log(temps))
MLM <- MLM[-which(MLM[,2] == -Inf),]
MLM <- as.data.frame(MLM)
res <- lm(MLM$V2~MLM$V1)
{
plot(MLM$V1,MLM$V2)
abline(a = res$coefficients[1],b = res$coefficients[2],col="red")
}
```



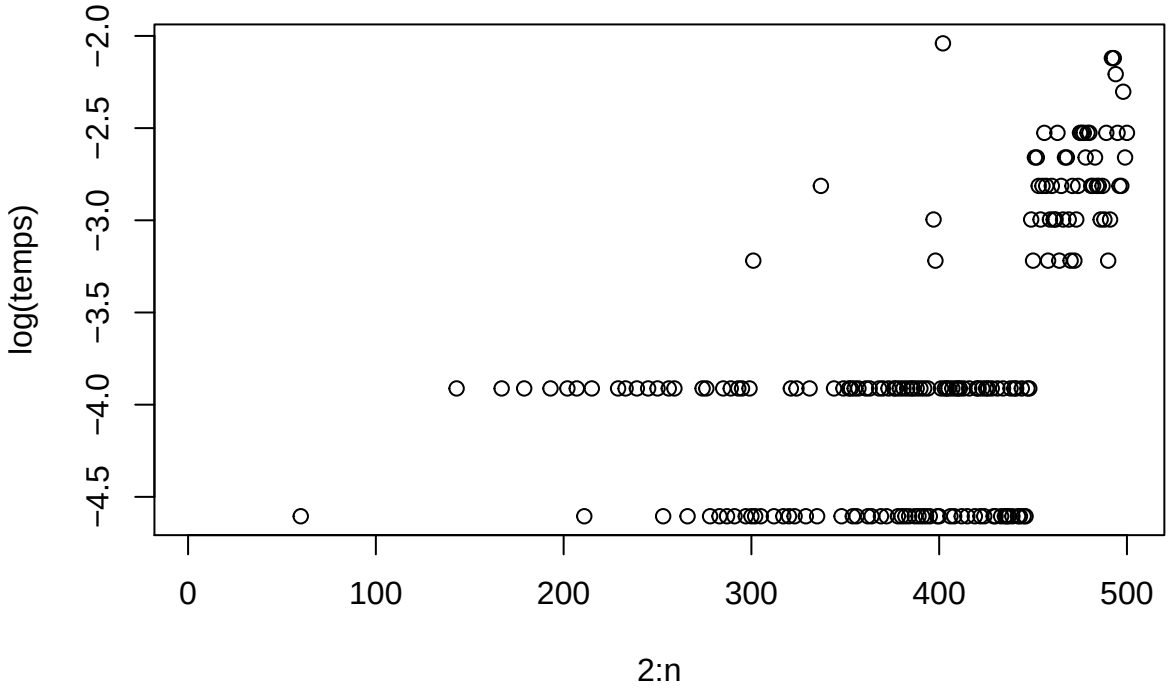
Pour coder des matrices symétriques de manière pratique, on peut remarquer qu'une matrice multipliée par sa transposée donne une matrice symétrique.

```
n <- 500
temps <- c()
for(i in 2:n)
{
  M = runif(i^2)
  N <- matrix(data = M,nrow = i,ncol = i)
  N <- N%*%t(N)
  tic <- proc.time()
  res <- solve(N)
  tac <- proc.time() - tic
  temps <- c(temps,as.numeric(tac[3]))
}
{
plot(2:n,log(temps))
lines((2:n/150-5.5))
}
```

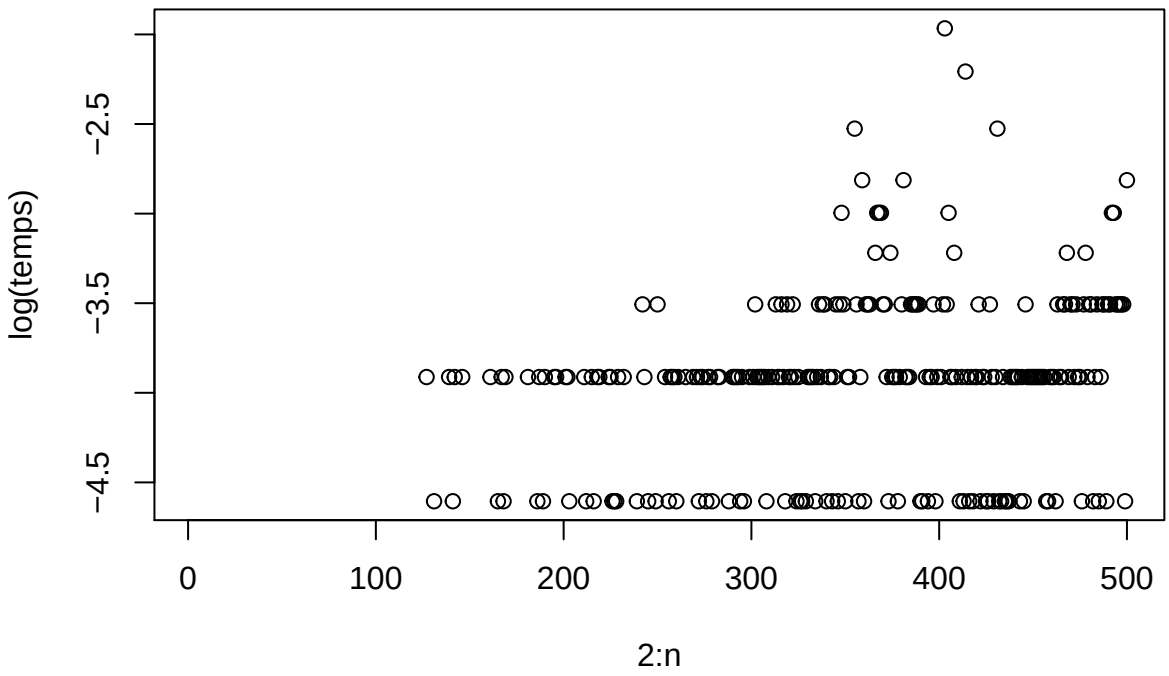


On remarque que les vitesse sont similaires ! Utililsons maintenant la méthode `chol2inv`.

```
n <- 500
temps <- c()
# création de la matrice
for(i in 2:n)
{
  M = runif(i^2)
  N <- matrix(data = M,nrow = i,ncol = i)
  tic <- proc.time()
  res <- chol2inv(N)
  tac <- proc.time() - tic
  temps <- c(temps,as.numeric(tac[3]))
}
plot(2:n,log(temps))
```




```
n <- 500
temps <- c()
for(i in 2:n)
{
  M = runif(i^2)
  N <- matrix(data = M,nrow = i,ncol = i)
  N <- N%*%t(N)
  tic <- proc.time()
  res <- chol2inv(N)
  tac <- proc.time() - tic
  temps <- c(temps,as.numeric(tac[3]))
}
plot(2:n,log(temps))
```



On voit que cette méthode est beaucoup plus rapide que ce soit pour les matrices symétriques ou non symétriques.

Faire de même pour le déterminant (fonction `det`).

Attenzione !!!
Notez que le temps que j'utilise ici est le temps écoulé pour l'algorithme. *Il dépend donc fortement des capacités de la machine avec laquelle j'ai fait les calculs.* Cette machine n'étant pas une machine de guerre, il est possible que vous obteniez des résultats beaucoup plus rapidement que moi.
De plus, notez que j'ai choisi de coder ici ce qui était demandé à l'aide de boucles. Je rappelle que ce n'est pas forcément une manière de coder optimale en `R`. On aurait pu,par exemple, coder une fonction qui calcule la matrice et l'inverse tout en enregistrant le temps de calcul et utiliser un `apply` sur un vecteur de différentes valeurs de n. Je vous enccourage fortement à essayer pour vous entrainer.
Enfin, veuillez remarquez que j'ai obtenu les coefficients des droites à l'oeil nu. Vous trouverez ces coefficients en lisant les formules que j'ai rentré dans les fonctions `lines` apràs chaque `plot`. Si l'on était vraiment méticuleux et consciencieux, on utiliserait une régression linéaire pour obtenir les coefficients précisément ! Pour effectuer une régression linéaire, on utilise la fonction `lm`. Je vous encourage également à regarder la fonction `help` de cette fonction et à l'utiliser pour obtenir des coefficients plus précis.