Tableaux et matrices sous R

Ricco Rakotomalala

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Les tableaux (array) et les matrices (matrix) sont des vecteurs, sauf qu'ils sont encapsulés de manière à ce qu'on puisse les manipuler en plusieurs dimensions. Ils possèdent l'attribut **dim**.

Une matrice est un tableau restreint à deux dimensions avec un accès ligne et colonne. Elle dispose d'opérateurs spécifiques (inversion, déterminant, produit matriciel, etc.). Nous utiliserons principalement des matrices dans nos calculs.

Création à partir d'un vecteur, accès aux valeurs, …

CRÉATION ET MANIPULATION

Création à partir d'un vecteur : Utilisation de la commande matrix()

```
v <- c(1.2,2.3,4.1,2.5,1.4,2.7)
m <- matrix(v,nrow=2,ncol=3)
attributes(m)
print(m)
```

```
> attributes(m)
$dim
[1] 2 3

> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
```

- Par défaut, les valeurs sont organisées par colonne ; on peut appliquer un traitement par lignes avec l'option **byrow = TRUE**

- Si nrow et ncol sont trop « courts », la matrice est sous dimensionnée

```
> print(matrix(v,nrow=2,ncol=1))
     [,1]
[1,]  1.2
[2,]  2.3
```

- Si nrow et ncol sont trop « longues », tout dépend des possibilités de réplication

```
> # 8 valeurs
> print(matrix(v,nrow=4,ncol=2))
     [,1] [,2]
[1,]  1.2  1.4
[2,]  2.3  2.7
[3,]  4.1  1.2
[4,]  2.5  2.3
Warning message:
In matrix(v, nrow = 4, ncol = 2) :
  data length [6] is not a sub-multiple or multiple of the number of rows [4]
```

```
v <- c(1.2,2.3,4.1,2.5,1.4,2.7)
m <- matrix(v,nrow=2,ncol=3)
```

- Si nrow et ncol sont trop « longues », tout dépend des possibilités de réplication (suite)

```
> # 12 valeurs
> print(matrix(v,nrow=3,ncol=4))
     [,1] [,2] [,3] [,4]
[1,]  1.2  2.5  1.2  2.5
[2,]  2.3  1.4  2.3  1.4
[3,]  4.1  2.7  4.1  2.7
```

- Connaître les nombres de lignes et de colonnes avec les commandes **nrow()** et **ncol()**

```
> #nombre de lignes
> print(nrow(m))
[1] 2
> #nombre de colonnes
> print(ncol(m))
[1] 3
```

```
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
```

m[1,2] # renvoie 4.1

m[3] # renvoie 4.1 (on peut linéariser l'accès, il décompte par colonne) !!!

m[1,] # renvoie le *vecteur* (1.2, 4.1, 1.4) !!!

m[,1] # renvoie le *vecteur* (1.2, 2.3) !!!

m[,2:3] # renvoie une matrice composée des colonnes 2 et 3 de **m**

On peut affecter les résultats à une nouvelle variable (vecteur, matrice)

Accès aux valeurs : indexation par un vecteur de booléens

```
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
```

```
#indexation par vecteur de booléens
lig.b <- c(T,F)
col.b <- c(T,F,T)
print(m[lig.b,col.b])
```

```
> print(m[lig.b,col.b])
[1] 1.2 1.4
```

```
#ça veut dire qu'on peut aussi
#utiliser des conditions
#ex.
sum.col <- apply(m,2,sum)
print(sum.col)
#et...
m.prim <- m[,sum.col<5]
print(m.prim)
```

Somme de chaque colonne → vecteur à 3 valeurs

```
> print(sum.col)
[1] 3.5 6.6 4.1
```

Extraction de certains colonnes

```
> print(m.prim)
     [,1] [,2]
[1,]  1.2  1.4
[2,]  2.3  2.7
```

Nommage des lignes et des colonnes

```
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
> #nommage
> rownames(m) <- c("pierre","paul")
> colnames(m) <- c("x1","x2","x3")
> print(m)
       x1  x2  x3
pierre 1.2 4.1 1.4
paul   2.3 2.5 2.7
>
> #intérêt ? accès par nom
> print(m["pierre",])
 x1  x2  x3
1.2 4.1 1.4
> print(m[,c("x1","x3")])
       x1  x3
pierre 1.2 1.4
paul   2.3 2.7
```

On peut attribuer des noms aux lignes et aux colonnes avec **rownames()** et **colnames()**

On peut s'appuyer sur les noms pour accéder au contenu de la matrice

On peut adjoindre des colonnes cbind() ou des lignes rbind() à une matrice

```
> v <- c(1.2,2.3,4.1,2.5,1.4,2.7)
> m <- matrix(v,nrow=2,ncol=3)
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
> #extraction de la 3e colonne
> z <- m[,3]
> print(z)
[1] 1.4 2.7
> #adjonction de cette colonne à m
> w <- cbind(m,z)
> print(w)
                  z
[1,] 1.2 4.1 1.4 1.4
[2,] 2.3 2.5 2.7 2.7
> #un autre vecteur de 4 valeurs
> s <- c(1.1,2.0,1.4,0.5)
> u <- rbind(w,s)
> print(u)
                  z
  1.2 4.1 1.4 1.4
  2.3 2.5 2.7 2.7
s 1.1 2.0 1.4 0.5
```

Note : Un warning est envoyé si les tailles ne correspondent pas : colonne ou ligne additionnelle trop grande ou trop petite. Le principe de réplication est utilisé dans ce second cas.

Matrice = vecteur particulier ➔ opérations « elementwise »

```
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
> #création de z
> #multiplication par un scalaire
> z <- 2 * m
> print(z)
     [,1] [,2] [,3]
[1,]  2.4  8.2  2.8
[2,]  4.6  5.0  5.4
>
> #addition élément par élément
> w <- z + m
> print(w)
     [,1] [,2] [,3]
[1,]  3.6 12.3  4.2
[2,]  6.9  7.5  8.1
>
> #multiplication entre 2 matrices
> q <- z * m
> print(q)
      [,1]  [,2]  [,3]
[1,]  2.88 33.62  3.92
[2,] 10.58 12.50 14.58
```

2.4 = 2 * 1.2

8.2 = 2 * 4.1

…

3.6 = 2.4 + 1.2

12.3 = 8.2 + 4.1

…

2.88 = 2.4 * 1.2

33.62 = 8.2 * 4.1

…

On dispose également des opérateurs matriciels usuels : t(…) pour la transposée ; x%*%y pour la multiplication; det(…) pour le déterminant ; solve(…) pour l'inversion et la résolution d'un système d'équations ; eigen(…) pour le calcul des valeurs et vecteurs propres ; svd(…) pour la décomposition en valeurs singulières; etc.

```
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
> #transposée
> tm <- t(m)
> print(tm)
     [,1] [,2]
[1,]  1.2  2.3
[2,]  4.1  2.5
[3,]  1.4  2.7
>
> #produit matriciel
> p <- tm %*% m
> print(p)
      [,1]  [,2]  [,3]
[1,]  6.73 10.67  7.89
[2,] 10.67 23.06 12.49
[3,]  7.89 12.49  9.25
>
> #valeurs et vect. propres
> print(eigen(p))
$values
[1]  3.632417e+01  2.715828e+00 -2.083170e-15

$vectors
           [,1]       [,2]         [,3]
[1,] -0.4078975  0.5027224  0.762161303
[2,] -0.7780304 -0.6282234 -0.002013636
[3,] -0.4777953  0.5938060 -0.647384039
```

Calculs récapitulatifs selon les lignes ou les colonnes

OPÉRATIONS RÉCAPITULATIVES

Calculs par lignes ou par colonnes : utilisation de apply()

```
> print(m)
     [,1] [,2] [,3]
[1,]  1.2  4.1  1.4
[2,]  2.3  2.5  2.7
```

```
#somme globale
sum(m) #renvoie un scalaire 14.2
#somme par ligne (sur la 1ère dimension)
print(apply(m,1,sum)) #renvoie un vecteur (6.7, 7.5)
#somme par colonne (2nde dimension)
print(apply(m,2,sum)) #renvoie un vecteur (3.5, 6.6, 4.1)
```

Lorsque chaque calcul renvoie une valeur, le tout se présente sous forme de vecteur

```
#calcul de profil
profil <- function(x){
  return(x/sum(x))
}

#profil colonne
print(apply(m,2,profil))
```

Lorsque chaque calcul renvoie un vecteur, le tout se présente sous forme de matrice

```
          [,1]      [,2]      [,3]
[1,] 0.3428571 0.6212121 0.3414634
[2,] 0.6571429 0.3787879 0.6585366
```

Conclusion - R est magique

De la documentation à profusion (**n'achetez jamais des livres sur R**)

Site du cours

http://eric.univ-lyon2.fr/~ricco/cours/cours_programmation_R.html

Programmation R

http://www.duclert.org/

Quick-R

http://www.statmethods.net/

POLLS (Kdnuggets)

Data Mining / Analytics Tools Used

(R, 2nd ou 1er depuis 2010)

What languages you used for data mining / data analysis?

http://www.kdnuggets.com/polls/2013/languages-analytics-data-mining-data-science.html

(Août 2013, langage R en 1ère position)

Article New York Times (Janvier 2009)

"**Data Analysts Captivated by R's Power**" - http://www.nytimes.com/2009/01/07/technology/business-computing/07program.html?_r=1