Travaux pratiques en langage R : opérations matricielles de base

Pierre Legendre
Département de sciences biologiques
Université de Montréal

Janvier 2005
Janvier 2006
Mai 2007

1. Créer des vecteurs et des matrices

Il y a plusieurs façons de créer des vecteurs en langage R.

1. Fonction 'scan' (lire des données dans la console R pour former un vecteur)

```
vec1=scan()
1: 1
2: 5
3: 35
4:
Read 3 items

vec1
[1]  1  5 35

vec2=scan()
1: 1
2: 2
3: 4
4:
Read 3 items
```

2. Fonction 'c' (combiner)

```
vec1 = c(1, 5, 35)
vec1
[1]  1  5 35

# La commande 'c' permet de combiner des vecteurs préexistants

vec12 = c(vec1, vec2)
vec12
[1]  1  5 35  1  2  4
```

3. Créer un vecteur formé de 6 valeurs, puis tenter de transformer celui-ci en matrice

```
vec3 = c(6, 3, 0, 7, -5, 1)
mat3 = matrix(vec3, 3, 2)
# Ou sur une seule ligne:   mat3 = matrix(c(6, 3, 0, 7, -5, 1), 3, 2)
mat3
     [,1] [,2]
[1,]    6    7
[2,]    3   -5
[3,]    0    1

mat1 = matrix(vec3, 3, 2, byrow=TRUE)
mat1
     [,1] [,2]
[1,]    6    3
[2,]    0    7
[3,]   -5    1
```

Pourquoi ces deux matrices sont-elles différentes?
Quelle convention suit le langage R, par défaut, lorsqu'il crée une matrice à partir d'un vecteur?

On peut également inscrire les données dans un fichier et demander à R de lire ce fichier par la commande 'read.table'. Voir le document « Introduction aux fonctions du langage R ».

4. Générer des vecteurs de nombres aléatoires

Fonction 'rnorm' pour des nombres aléatoires tires d'une distribution normale

```
aleaNorm1=rnorm(10)  # ou  aleaNorm1=rnorm(10,mean=0,sd=1)
aleaNorm1
 [1]  0.5537578  0.3176950  1.0835924  0.1095523  0.7744556 -1.0011711  1.0175194
 [8]  0.8675578  1.5805411  0.5824715

aleaNorm2=rnorm(10,mean=50,sd=10)
aleaNorm2
 [1] 49.51681 42.66049 54.33008 47.16372 70.80451 48.25744 55.69606 49.06176 63.45640
[10] 47.96367
```

Fonction 'runif' pour des nombres aléatoires tires d'une distribution uniforme

```
aleaUnif1=runif(5)  # ou  aleaUnif1=runif(5,min=0,max=1)
aleaUnif1
[1] 0.7684457 0.8364512 0.3568679 0.9119424 0.9935300

aleaUnif2=runif(5,min=10,max=20)
aleaUnif2
[1] 15.75800 13.37846 17.76781 13.36102 17.49161
```

Fonction 'rlnorm' pour la distribution lognormale (distribution dont le log donne une distribution normale). On peut préciser la moyenne 'meanlog' et l'écart type 'sdlog' de la distribution normale correspondante.

```
aleaLNorm1=rlnorm(5)  # ou aleaLNorm1=rlnorm(5,meanlog=0,sdlog=1)
aleaLNorm1
[1] 0.2129886 1.6512231 3.2237191 5.8959147 1.2279456
aleaLNorm2=rlnorm(5,meanlog=2,sdlog=5)
aleaLNorm2
[1]  0.75931953  0.02862776  1.12951044  0.51971353 48.50212899
```

5. Générer des sequences régulières de nombres

```
res = seq(1, 5, 0.5)
res
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

6. Répéter des séquences de nombres

```
res = rep(c(1, 2, 3), 4)
res
 [1] 1 2 3 1 2 3 1 2 3 1 2 3
```

7. Les fonctions 'fix(nom)' et 'edit(nom)' permettent de modifier une ou des valeurs dans une matrice. L'objet peut être de type 'data.frame', 'matrix' ou 'vector'.

'fix' modifie directement les valeurs de la matrice :

```
fix(mat1)
```

'edit' conserve les changements dans une nouvelle matrice dont on aura fourni le nom.
Exemple :

```
mat1a = edit(mat1)
```

2. Opérations matricielles de base

1. Somme de deux vecteurs ou de deux matrices: opérateur '+'

```
vec1
[1]  1  5 35
vec2
[1] 1 2 4
vec1+vec2
[1]  2  7 39

mat1
     [,1] [,2]
[1,]    6    3
[2,]    0    7
[3,]   -5    1
mat2              # Saisissez la matrice mat2 à l'aide de la fonction matrix(), page 2
     [,1] [,2]
[1,]    3    8
[2,]    2   -1
[3,]    6   -4
mat1+mat2
     [,1] [,2]
[1,]    9   11
[2,]    2    6
[3,]    1   -3
```

Il faut s'assurer que deux matrices ont le même nombre de lignes et de colonnes avant de calculer leur somme.
Que produirait la commande mat1+t(mat2) ?

2. Produit scalaire de deux vecteurs ou de deux matrices: opérateur %*%

```
# Exemple du manuel, p. 72
vec1
[1]  1  5 35
vec2
[1] 1 2 4
vec1 %*% vec2
     [,1]
[1,]  151
```

Lors du calcul d'un produit de deux vecteurs, R considère que le premier vecteur est horizontal et le second vertical.

```
# On peut spécifier l'orientation des vecteurs en les transformant en matrices.
Mvec1=matrix(vec1, 1, 3)
Mvec2=matrix(vec2, 3, 1)
Mvec1
     [,1] [,2] [,3]
[1,]    1    5   35
Mvec2
     [,1]
[1,]    1
[2,]    2
[3,]    4
Mvec1 %*% Mvec2
     [,1]
[1,]  151
# Que produirait la commande   Mvec2 %*% Mvec1 ?
```

Il faut s'assurer que deux matrices sont *conformes* avant de calculer leur produit, c'est-à-dire que le nombre de colonnes de la matrice de gauche est identique au nombre de lignes de la matrice de droite.

```
# La fonction « dim » permet de connaître les dimensions d'une matrice :
dim(Mvec1)
[1] 1 3
dim(Mvec2)
[1] 3 1

mat1 %*% mat2
# Quel message obtenez-vous ? Comment pouvez-vous remédier à cette situation ?

mat1 %*% t(mat2)
     [,1] [,2] [,3]
[1,]   42    9   24
[2,]   56   -7  -28
[3,]   -7  -11  -34
# L'opérateur 't' a transposé la matrice 'mat2'
```

3. Produit Hadamard : produit élément par élément de deux vecteurs ou matrices

```
vec1 = c(1, 5, 35)
vec2 = c(1, 2, 4)
vec1 * vec2

mat1 = matrix(c(6,0,-5,3,7,1), 3, 2)
mat2 = matrix(c(3,2,6,8,-1,-4), 3, 2)
mat1 * mat2
```

Autres commandes matricielles utiles
Pour details, voir les fichier d'aide. Exemple: ?as.matrix

Conversion d'un objet 'data.frame' ou 'vector' au type 'matrix': fonction 'as.matrix'
Centrer ou centrer-réduire les valeurs des colonnes d'une matrice: 'scale'
Calcul d'une matrice de covariances: fonction 'cov'
Calcul d'une matrice de corrélations: fonction 'cor'
Transposition d'une matrice: opérateur 't'
Calcul du déterminant: fonction 'det'
Inversion d'une matrice: fonction 'solve', ou 'ginv' (*generalized inverse*, bibliothèque MASS)
Calcul des valeurs propres et des vecteurs propres: fonction 'eigen'
Décomposition en valeurs singulières: fonction 'svd'

4. Calcul d'un déterminant (manuel p. 78-80)

```
mat3x3 = matrix(c(1,2,3,4,5,6,7,8,10), 3, 3, byrow=TRUE)
mat3x3
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8   10
det(mat3x3)
[1] –3
```

5. Calcul de valeurs propres et de vecteurs propres (manuel p. 92-97)

```
mat2x2
     [,1] [,2]
[1,]    2    2
[2,]    2    5
res = eigen(mat2x2)

# La fonction 'summary' nous apprend que le fichier 'res' comporte deux composantes:
'res$values' et 'res$vectors' :
summary(res)
        Length Class  Mode
values  2      -none- numeric
vectors 4      -none- numeric

res$values
[1] 6 1

res$vectors
          [,1]       [,2]
[1,] 0.4472136  0.8944272
[2,] 0.8944272 -0.4472136
```

6. Mise à une puissance par produit matriciel (produit scalaire)

```
B = matrix(c(2,3,3,5), 2, 2)
B
[,1] [,2]
[1,]    2    3
[2,]    3    5
B %*% B
```

```
     [,1] [,2]
[1,]   13   21
[2,]   21   34

# Attention: l'opérateur exposant '^' produit une matrice dont chaque élément est mis à la puissance spécifiée et non la mise à puissance de la matrice
B^2
     [,1] [,2]
[1,]    4    9
[2,]    9   25
```

7. Mise à une puissance *via* les valeurs et les vecteurs propres (manuel p. 100, eq. 2.29)

```
eig.B = eigen(B)

eig.B$values
[1] 6.8541020 0.1458980
eig.B$vectors
          [,1]       [,2]
[1,] 0.5257311  0.8506508
[2,] 0.8506508 -0.5257311

# B^2 = U %*% (matrice diagonale des valeurs propres au carré) %*% inverse de U
B.exp2 = eig.B$vectors %*% diag(eig.B$values^2) %*% solve(eig.B$vectors)
B.exp2
     [,1] [,2]
[1,]   13   21
[2,]   21   34

# Trouver l'exposant 3.1416 de la matrice B
BB = eig.B$vectors %*% diag(eig.B$values^3.1416) %*% solve(eig.B$vectors)
BB
         [,1]     [,2]
[1,] 116.8843 189.1189
[2,] 189.1189 306.0031

# N.B. Ce calcul est impossible s'il y a des valeurs propres négatives et l'exposant est fractionnaire.  En effet, l'exposant fractionnaire d'un nombre négatif n'est pas défini.
```

8. Appliquer une fonction aux lignes ou aux colonnes à un tableau: 'apply'

```
?apply

# Créer une matrice de nombres aléatoires à distribution normale
mat = matrix(rnorm(15, 5, 1),5,3)
# Calculer la somme des valeurs de 'mat' ligne par ligne
# Le paramètre "1" indique d'appliquer la fonction "sum" aux lignes de "mat"
row.sums = apply(mat, 1, sum)
row.sums

# Calculer la somme des valeurs de 'mat' colonne par colonne
# Le paramètre "2" indique d'appliquer la fonction "sum" aux colonnes de "mat"
col.sums = apply(mat, 2, sum)
col.sums
# Que produira la commande suivante?   sum(mat)
```

```
# Calculer la moyenne des variables de ‘mat’ colonne par colonne
col.means = apply(mat, 2, mean)
col.means
# Que produira la commande suivante?   mean(mat)

# Calculer l’écart type des variables de ‘mat’ colonne par colonne
col.sd = apply(mat, 2, sd)
col.sd
# Que produira la commande suivante?   sd(mat)

# Calculer la variance des variables de ‘mat’ colonne par colonne
col.var = apply(mat, 2, var)
col.var
# Que produira la commande suivante?  var(mat)

# Centrer les variables de ‘mat’ colonne par colonne
mat.centred = scale(mat, center=TRUE, scale=FALSE)              # ou encore
mat.centred = apply(mat, 2, scale, center=TRUE, scale=FALSE)
# Vérifier que la somme des valeurs de chaque colonne est maintenant zéro
apply(mat.centred, 2, sum)
# Comment faut-il comprendre ces résultats?

# Centrer et réduire les variables de ‘mat’ colonne par colonne,
# i.e., soustraire la moyenne et diviser par l’écart type de chaque colonne
mat.stan = scale(mat, center=TRUE, scale=TRUE)                  # ou encore
mat.stan = apply(mat, 2, scale, center=TRUE, scale=TRUE)
# Vérifier la somme et la variance de chaque colonne
apply(mat.stan, 2, sum)
apply(mat.stan, 2, var)
```

9. Une fonction utile: 'which'

```
# Trouver la position des valeurs possédant certaines propriétés dans un vecteur ou une matrice
?which

# Où se trouvent les valeurs positives dans un vecteur de nombres aléatoires?
vec = rnorm(10)
vec.positif = which(vec > 0)
vec
vec.positif

# Où se trouvent les valeurs négatives dans un tableau de nombres aléatoires?
mat = matrix(rnorm(15), 5, 3)
mat. negatif = which(mat < 0)
mat
mat.negatif
# Dans quel ordre les valeurs sont-elles lues dans une matrice?
```

10. Une fonction utile: 'sweep'

```
# Produit une matrice résultant d'une opération, impliquant une statistique, appliquée
# à chaque case d'une matrice fournie en entrée
?sweep

# Créer un tableau de valeurs ressemblant à des abondances d'espèces (10 lignes et 5 colonnes)
# La proportion de zéros dans 'tableau' dépend de la valeur de l'écart type (1.5) de 'rnorm'[1]
tableau = matrix(round(exp(rnorm(50, 0, 1.5))), 10, 5)

# Diviser chaque valeur par la somme de sa ligne pour créer un tableau d'abondances relatives
row.sums = apply(tableau, 1, sum)
tableau.profiles = sweep(tableau, 1, row.sums, "/")
tableau.profiles
# Vérifier que la somme de chaque ligne du tableau produit est bien 1
apply(tableau.profiles, 1, sum)

# Diviser chaque valeur par la valeur maximum qui se trouve dans le tableau
tableau.max = max(tableau)
tableau.norm = sweep(tableau, 2, tableau.max, "/")
tableau.norm
# Vérifier quelle est la valeur maximum dans le tableau transformé
max(tableau.norm)
```

[1] Formule utilisée dans le program SimSSD. La formule est décrite à la p. 443 de l'article suivant :

Legendre, P., D. Borcard and P. R. Peres-Neto. 2005. Analyzing beta diversity: partitioning the spatial variation of community composition data. *Ecological Monographs* 75: 435-450.