Matrices en R

Jordi Vanrell

22/10/2020

Matrices en R

Para representar matrices se usa la nomenclatura:

matrix(vector, nrow=n, byrow=valor_logico) para definir una matriz de n filas formada por las entradas del vector.

nrow es el número de filas

byrow igualado a TRUE se construye la matriz por filas.

```
M=matrix(1:12, nrow=4)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
M=matrix(1:12, nrow=4, byrow=TRUE)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
## [4,]   10   11   12
```

```
M=matrix(1:12, nrow=5)
```

```
## Warning in matrix(1:12, nrow = 5): la longitud de los datos [12] no es un
## submúltiplo o múltiplo del número de filas [5] en la matriz
```

```
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8    1
## [4,]    4    9    2
## [5,]    5   10    3
```

```r
matrix(1,nrow=4,ncol=6)
```

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

```r
matrix(0,nrow=3,ncol=5)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
```

```r
vec=c(1,2,3,4,5,6,7,8,9,10,11,12)
matrix(vec,nrow=3,byrow=FALSE)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Construcción de matrices a partir de rbind o cbind

Para añadir filas o columnas a una matriz o crearlas a partir de la definición de filas y columnas.

```r
rbind(M,c(1,2,4),c(-1,-2,-3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8    1
## [4,]    4    9    2
## [5,]    5   10    3
## [6,]    1    2    4
## [7,]   -1   -2   -3
```

Para construir matrices diagonales:

```r
diag(c(1,2,3,4))
```

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

```r
diag(5,ncol=3,nrow=3)
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    0
## [2,]    0    5    0
## [3,]    0    0    5
```

Submatrices

```r
M[2,2]
```

```
## [1] 7
```

```r
M[,3]
```

```
## [1] 11 12  1  2  3
```

```r
M[3,]
```

```
## [1] 3 8 1
```

```r
M[c(2,3,5),1:2]
```

```
##      [,1] [,2]
## [1,]    2    7
## [2,]    3    8
## [3,]    5   10
```

Funcones de las matrices

```r
diag(M)
```

```
## [1] 1 7 1
```

```r
ncol(M)
```

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

```r
nrow(M)
```

```
## [1] 5
```

```r
sum(M)
```

```
## [1] 84
```

```r
dim(M)
```

```
## [1] 5 3
```

```r
mean(M)
```

```
## [1] 5.6
```

```r
colSums(M)
```

```
## [1] 15 40 29
```

```r
rowSums(M)
```

```
## [1] 18 21 12 15 18
```

Para aplicar una función a las filas o columnas de una matriz. MARGIN=1 lo hace por filas, y 2 es por columnas.

```r
apply(M, MARGIN=1,FUN=function(x){sqrt(sum(x^2))})
```

```
## [1] 12.569805 14.035669  8.602325 10.049876 11.575837
```

```r
apply(M, MARGIN=2,FUN=function(x){sqrt(sum(x^2))})
```

```
## [1]  7.416198 18.165902 16.703293
```

Operaciones con matrices

```r
t(M) #Es la traspuesta de la matriz
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    2    3    4    5
## [2,]    6    7    8    9   10
## [3,]   11   12    1    2    3
```

Para multiplicar matrices

```r
M%*%t(M) #Multiplicamos M por su transpuesta
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  158  176   62   80   98
## [2,]  176  197   74   95  116
## [3,]   62   74   74   86   98
## [4,]   80   95   86  101  116
## [5,]   98  116   98  116  134
```

```
M*M #sería el producto escalar
```

```
##      [,1] [,2] [,3]
## [1,]    1   36  121
## [2,]    4   49  144
## [3,]    9   64    1
## [4,]   16   81    4
## [5,]   25  100    9
```

```
M+M #para sumar matrices
```

```
##      [,1] [,2] [,3]
## [1,]    2   12   22
## [2,]    4   14   24
## [3,]    6   16    2
## [4,]    8   18    4
## [5,]   10   20    6
```

Para elevar una matriz a n se necesita el paquete Biodem, pero solo aproxima las potencias, no las calcula.

```
#Sería:
#install.packages(Biodem)
#library(Biodem)
#mtx.exp(matriz,n)
```

Para elevar matrices también puede usarse %% del paquete expm, pero tampoco las calcula de manera exacta, solo las aproxima.

```
A=rbind(c(2,0,2),c(1,2,3),c(0,1,3))
B=rbind(c(3,2,1),c(1,0,0),c(1,1,1))
A
```

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

```
B
```

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

```r
A%*%B
```

```
##      [,1] [,2] [,3]
## [1,]    8    6    4
## [2,]    8    5    4
## [3,]    4    3    3
```

```r
A%*%A
```

```
##      [,1] [,2] [,3]
## [1,]    4    2   10
## [2,]    4    7   17
## [3,]    1    5   12
```

```r
B%*%B%*%B
```

```
##      [,1] [,2] [,3]
## [1,]   47   28   16
## [2,]   12    7    4
## [3,]   20   12    7
```

Para calcular el determinante (de una matriz siempre cuadrada):

```r
det(A)
```

```
## [1] 8
```

Para calcular el rango de una matriz:

```r
qr(A)$rank
```

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

Por cierto: qr(matriz) devuelve los atributos de la matriz:

```r
qr(A)
```

```
## $qr
##            [,1]       [,2]      [,3]
## [1,] -2.2360680 -0.8944272 -3.130495
## [2,]  0.4472136 -2.0493902 -3.025290
## [3,]  0.0000000  0.4879500  1.745743
##
## $rank
## [1] 3
##
## $qraux
## [1] 1.894427 1.872872 1.745743
##
```

```
## $pivot
## [1] 1 2 3
##
## attr(,"class")
## [1] "qr"
```

Para calcular la inversa de una matriz (siempre que sea invertible)

```
solve(A)
```

```
##        [,1]  [,2] [,3]
## [1,]  0.375  0.25 -0.5
## [2,] -0.375  0.75 -0.5
## [3,]  0.125 -0.25  0.5
```

La función solve(matriz,b) también sirve para **resolver sistemas de ecuaciones** añadiendo el argumento b, siendo b el vector de términos independientes.

```
b=c(3,5,7)
solve(A,b)
```

```
## [1] -1.125 -0.875  2.625
```

```
solve(A,c(1,2,3))
```

```
## [1] -0.625 -0.375  1.125
```

Para calcular vectores y valores propios:

```
eigen(A) #devuelve los valores propis (vaps) y los vectores propios (veps)
```

```
## eigen() decomposition
## $values
## [1] 4.511547+0.000000i 1.244226+0.474477i 1.244226-0.474477i
##
## $vectors
##              [,1]                  [,2]                  [,3]
## [1,] 0.4022596+0i  0.7337066+0.0000000i  0.7337066+0.0000000i
## [2,] 0.7635534+0i  0.4042133-0.4371684i  0.4042133+0.4371684i
## [3,] 0.5051469+0i -0.2772580+0.1740634i -0.2772580-0.1740634i
```

```
eigen(A)$values
```

```
## [1] 4.511547+0.000000i 1.244226+0.474477i 1.244226-0.474477i
```

```
eigen(A)$vectors
```

```
##              [,1]                  [,2]                  [,3]
## [1,] 0.4022596+0i  0.7337066+0.0000000i  0.7337066+0.0000000i
## [2,] 0.7635534+0i  0.4042133-0.4371684i  0.4042133+0.4371684i
## [3,] 0.5051469+0i -0.2772580+0.1740634i -0.2772580-0.1740634i
```

```
N=rbind(c(2,6,-8),c(0,6,-3),c(0,2,1))
```

Si Pe es la matriz de vectores propios de N y D la matriz diagonal ciyas entradas son los valores propios de N, entonces se cumple la descomposición canónica:

N=P · D · P^-1

```
eigen(N)$vectors
```

```
##           [,1]       [,2] [,3]
## [1,] 0.2672612 -0.8164966    1
## [2,] 0.8017837  0.4082483    0
## [3,] 0.5345225  0.4082483    0
```

```
D=(eigen(N)$values)*diag(1,nrow=3)
D
```

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

```
solve(eigen(N)$vectors)
```

```
##      [,1]      [,2]      [,3]
## [1,]    0  3.741657 -3.741657
## [2,]    0 -4.898979  7.348469
## [3,]    1 -5.000000  7.000000
```

```
eigen(N)$vectors%*%D%*%solve(eigen(N)$vectors)
```

```
##      [,1] [,2] [,3]
## [1,]    2    6   -8
## [2,]    0    6   -3
## [3,]    0    2    1
```

Si hay algún valor propio mayor que 1 que se repita la función eigen() dará tantos valores de ese valor propio como su multiplicidad algebraica. Como resultado tendremos vectores propios repetidos y la matriz no será diagonalizable.

```
Q=matrix(c(0,1,0,-7,3,-11,16,-3,4),nrow=3,byrow=TRUE)
eigen(Q)
```

```
## eigen() decomposition
## $values
## [1]  5.646370+1.610997i  5.646370-1.610997i -4.292741+0.000000i
##
## $vectors
##                       [,1]                  [,2]          [,3]
## [1,] -0.1519603+0.0433566i -0.1519603-0.0433566i -0.1780206+0i
## [2,] -0.9278713+0.0000000i -0.9278713+0.0000000i  0.7641961+0i
## [3,]  0.3199285+0.1083001i  0.3199285-0.1083001i  0.6199298+0i
```

Los determinantes de matrices con números complejos no pueden resolverse si no se toma la propiedad de que el determinante de una matriz es siempre igual al producto de los valores propios:

```
prod(eigen(Q)$values)
```

```
## [1] -148+0i
```