Estructura de Datos en R - Matrices

Ramon Ceballos

16/1/2021

MATRICES EN R

Una matriz es una tabla rectangular de parámetros organizada en filas y columnas. Suelen escribirse entre paréntesis.

Vamos a definir como se construye una matriz, como se accede a posiciones y como se hacenalguans cosas básica del algebra lineal.

Más info en el curso de Algebra Lineal que tienes.

Definición de una matriz en R

Para definir una matriz se hace uso de la función **matrix()**.

- `matrix(vector, nrow=n, byrow=valor_lógico)`: para definir una matriz de n filas formada por las entradas del vector.
  - `nrow`: número de filas
  - `byrow`: si se iguala a TRUE, la matriz se construye por filas; si se iguala a FALSE (valor por defecto), se construye por columnas (**IMPORTANTE**)
  - `ncol`: número de columnas (puede usarse en lugar de nrow)
  - R muestra las matrices indicando como $[i,]$ la fila i-ésima y $[,j]$ la columna j-ésima
  - Todas las entradas de una matriz han de ser del mismo tipo de datos

```
#Ordenado por columnas
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
```

```
#Ordenado por filas
M2 = matrix(1:12, nrow=4, byrow = TRUE)
M2
```

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

Si al definir una matriz el número de elementos definidos es diferente al tamaño de la matriz, R crea la matriz pese a todo.

```
MA = 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
```

```
MA
```

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

Si en vez de emplear un vector empleamos un numero o variable para definir un matriz se obtiene lo siguiente.

```
#Es necesario especificar tanto el nº de columnas como de filas
MA = matrix(1,nrow=4,ncol=3)
MA
```

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

Ejercicios propuesto

Ejercicio 1. ¿Cómo definirías una matriz constante? Es decir, ¿cómo definirías una matriz A tal que $\forall\ i = 1, ..., n; j = 1, ..., m,\ a_{i,j} = k$ siendo $k \in \mathbb{R}$? Como R no admite incógnitas, prueba para el caso específico $n = 3,\ m = 5,\ k = 0$.

```
#Hay que realizar una matriz de n filas y m columnas
#En esta matriz cada parámetro se define como k
#Si n=3, m=5 y k=0.
matriz.X <- matrix(0, nrow=3,ncol=5)
matriz.X
```

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

Ejercicio 2. Con el vector vec = (1,2,3,4,5,6,7,8,9,10,11,12) crea la matriz

$$\begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

```r
vec = c(1,2,3,4,5,6,7,8,9,10,11,12)
matriz.X2 = matrix(vec,nrow=3,byrow=FALSE)
matriz.X2
```

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

Construir una matriz en R

Normalmente se emplean las funciones rbind() y cbind() para construir una matriz, en vez de matrix().

- `rbind(vector1, vector2, ...)`: construye la matriz de filas vector1, vector2,...
- `cbind(vector1, vector2, ...)`: construye la matriz de columnas vector1, vector2,...
  - Los vectores han de tener la misma longitud
  - También sirve para añadir columnas (filas) a una matriz o concatenar por columnas (filas) matrices con el mismo número de filas (columnas)
- `diag(vector)`: para construir una matriz diagonal con un vector dado
  - Si aplicamos diag a un número n, produce una matriz identidad de orden n

```r
#Añadir filas a matriz M
rbind(M, c(1,2,3),c(-9,5,3))
```

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

```r
#Construir matriz de cero
rbind( c(1,2,3),c(-9,5,3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]   -9    5    3
```

```r
#Construir matriz diagonal
Matriz_diagonal <- diag(c(1,2,3,4))
Matriz_diagonal
```

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

Submatrices (acceso a matrices)

Se define la i como la fila y la j como la columna.

- `matriz[i,j]`: indica la entrada (i, j) de la matriz, siendo $i, j \in \mathbb{N}$. Si i y j son vectores de índices, estaremos definiendo la submatriz con las filas pertenecientes al vector i y columnas pertenecientes al vector j
- `matriz[i,]`: indica la fila i-ésima de la matriz, siendo $i \in \mathbb{N}$
- `matriz[,j]`: indica la columna j-ésima de la siendo $j \in \mathbb{N}$
  - Si i (j) es un vector de índices, estaremos definiendo la submatriz con las filas (columnas) pertenecientes al vector i (j)

```
#Obtener un elemento de la matriz
M[2,2]
```

```
## [1] 6
```

```
#Obtener una fila de la matriz
M[2,]
```

```
## [1]  2  6 10
```

```
#Obtener una columna de la matriz
M[,2]
```

```
## [1] 5 6 7 8
```

```
#Obtener una submatriz
M[c(2,4),c(1,3)]
```

```
##      [,1] [,2]
## [1,]    2   10
## [2,]    4   12
```

Funciones en R para Matrices

Teniendo la matriz M podemos usar las siguientes funciones:

```
M
```

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

- `diag(matriz)`: para obtener la diagonal de la matriz

```r
diag(M)
```

```
## [1]  1  6 11
```

- `nrow(matriz)`: nos devuelve el número de filas de la matriz

```r
nrow(M)
```

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

- `ncol(matriz)`: nos devuelve el número de columnas de la matriz
- `dim(matriz)`: nos devuelve las dimensiones de la matriz

```r
dim(M)
```

```
## [1] 4 3
```

- `sum(matriz)`: obtenemos la suma de todas las entradas de la matriz
- `prod(matriz)`: obtenemos el producto de todas las entradas de la matriz
- `mean(matriz)`: obtenemos la media aritmética de todas las entradas de la matriz
- `colSums(matriz)`: obtenemos las sumas por columnas de la matriz
- `rowSums(matriz)`: obtenemos las sumas por filas de la matriz
- `colMeans(matriz)`: obtenemos las medias aritméticas por columnas de la matriz
- `rowMeans(matriz)`: obtenemos las medias aritméticas por filas de la matriz

Función apply()

- `apply(matriz, MARGIN=..., FUN=función)`: para aplicar otras funciones a las filas o las columnas de una matriz
  - `MARGIN`: ha de ser 1 si queremos aplicar la función por filas; 2 si queremos aplicarla por columnas; o c(1,2) si la queremos aplicar a cada entrada

```r
#Definimos matriz A
A = matrix(c(1,2,3,4,5,6,7,8,9,10,11,12), ncol = 3)
#Con apply empleamos una funcion en filas y columnas
apply(A, MARGIN = c(1,2), FUN = function (x){x^2})
```

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

```r
#Por filas
apply(A, MARGIN = 1, FUN = function (x){sqrt(sum(x^2))})
```

```
## [1] 10.34408 11.83216 13.37909 14.96663
```

```r
#Por columnas
apply(A, MARGIN = 2, FUN = function (x){sqrt(sum(x^2))})
```

```
## [1]  5.477226 13.190906 21.118712
```

Operaciones en R con matrices

- `t(matriz)`: para obtener la transpuesta de la matriz
- `+`: para sumar matrices
- `*`: para el producto de un escalar por una matriz
- `%*%`: para multiplicar matrices
- `mtx.exp(matriz,n)`: para elevar la matriz a n
  - Del paquete `Biodem`
    - No calcula las potencias exactas, las aproxima
- `%^%`: para elevar matrices
  - Del paquete `expm`
    - No calcula las potencias exactas, las aproxima
- `det(matriz)`: para calcular el determinante de la matriz. La matriz debe de ser cuadrada
- `qr(matriz)$rank`: para calcular el rango de la matriz

```r
M_EJ=rbind(c(1,4,2),c(0,1,3),c(1,8,9))
qr(M_EJ)$rank
```

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

```r
#El rango es 3
```

- `solve(matriz)`: para calcular la inversa de una matriz cuadrada que sea invertible
  - También sirve para resolver sistemas de ecuaciones lineales. Para ello introducimos `solve(matriz,b)`, donde b es el vector de términos independientes

```r
#La inversa de la matriz M_EJ
solve(M_EJ)
```

```
##      [,1] [,2] [,3]
## [1,]  3.0  4.0 -2.0
## [2,] -0.6 -1.4  0.6
## [3,]  0.2  0.8 -0.2
```

```r
#Al multiplicar la inversa de la matriz por la matriz debe dar la matriz identidad
solve(M_EJ)%*%M_EJ
```

```
##      [,1]          [,2]          [,3]
## [1,]    1 -3.552714e-15 -3.552714e-15
## [2,]    0  1.000000e+00  0.000000e+00
## [3,]    0  0.000000e+00  1.000000e+00
```

```r
#Para solucionar los errores de redondeo aplicamos...
round(solve(M_EJ)%*%M_EJ,2)
```

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

```r
#Para resolver sistemas de ecuaciones lineales se indica la igualdad de cada ecuación (fila)
solve(M_EJ,c(1,2,0))
```

```
## [1] 11.0 -3.4  1.8
```

```r
#Por tanto en este caso la x vale 11, la y vale-3.4 y la z vale 1.8
```

Valores y vectores propios

Vector propio y valor propio

- `eigen(matriz)`: para calcular los valores (vaps) y vectores propios (veps)
  - `eigen(matriz)$values`: nos da el vector con los vaps de la matriz en orden decreciente de su valor absoluto y repetidos tantas veces como su multiplicidad algebraica.
  - `eigen(matriz)$vectors`: nos da una matriz cuyas columnas son los veps de la matriz.

```r
#La lista generada con los tres valores propios
#y los tres vectores propios en columnas
eigen(M_EJ)
```

```
## eigen() decomposition
## $values
## [1] 11.5677644 -1.0000000  0.4322356
##
## $vectors
##            [,1]       [,2]        [,3]
## [1,] -0.2744671  0.7427814 -0.98750842
## [2,] -0.2626036 -0.5570860  0.15481805
## [3,] -0.9250444  0.3713907 -0.02930006
```

```r
M = rbind(c(2,6,-8), c(0,6,-3), c(0,2,1))
eigen(M)
```

```
## eigen() decomposition
## $values
## [1] 4 3 2
##
## $vectors
##           [,1]       [,2] [,3]
## [1,] 0.2672612 -0.8164966    1
## [2,] 0.8017837  0.4082483    0
## [3,] 0.5345225  0.4082483    0
```

Si hay algún vap con multiplicidad algebraica mayor que 1 (es decir, que aparece más de una vez), la función `eigen()` da tantos valores de este vap como su multiplicidad algebraica indica. Además, en este caso, R intenta que los veps asociados a cada uno de estos vaps sean linealmente independientes. Por tanto, cuando como resultado obtenemos veps repetidos asociados a un vap de multiplicidad algebraica mayor que 1, es porque para este vap no existen tantos veps linealmente independientes como su multiplicidad algebraica y, por consiguiente, la matriz no es diagonalizable.

```r
#Este es un ejemmplo de matriz no diagonalizable
M = matrix(c(0,1,0,-7,3,-1,16,-3,4), nrow=3, byrow=TRUE)
eigen(M)
```

```
## eigen() decomposition
## $values
## [1] 3 2 2
##
## $vectors
##            [,1]       [,2]       [,3]
## [1,] -0.1301889 -0.1825742 -0.1825742
## [2,] -0.3905667 -0.3651484 -0.3651484
## [3,]  0.9113224  0.9128709  0.9128709
```

```r
#También se podría operar con números complejos Beji
A = matrix(c(3-2i,5+3i,1+2i,2-1i),nrow=2,byrow=T)
A%*%A
```

```
##       [,1]  [,2]
## [1,]  4+1i 34+0i
## [2,] 11+7i  2+9i
```

```r
eigen(A)
```

```
## eigen() decomposition
## $values
## [1] 4.902076+1.101916i 0.097924-4.101916i
##
## $vectors
##                    [,1]                [,2]
## [1,] 0.8483705+0.000000i  0.8519823+0.000000i
## [2,] 0.4695014+0.244614i -0.5216168-0.045189i
```

```r
solve(A,c(3,6+2i))
```

```
## [1] 1.2470588-1.2117647i 0.7882353+0.7529412i
```

```r
#No está definida la función det(matriz) para complejo
#En su lugar para el calculo del determinante se emplea...
prod(eigen(A)$values)
```

```
## [1] 5-20i
```

Ejercicios

1. Observad qué ocurre si, siendo $A = \begin{pmatrix} 2 & 0 & 2 \\ 1 & 2 & 3 \\ 0 & 1 & 3 \end{pmatrix}$ y $B = \begin{pmatrix} 3 & 2 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$, realizamos las operaciones $A * B$, A^2 y B^3

```
A= matrix(c(2,0,2,1,2,3,0,1,3),nrow=3,byrow=TRUE)
A
```

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

```
B= matrix(c(3,2,1,1,0,0,1,1,1),nrow=3,byrow=TRUE)
B
```

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

```
#Multiplicar A * B
A%*%B
```

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

```
#A ^2
library(Biodem)
mtx.exp(A,2)
```

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

```
#B ^3
library(Biodem)
mtx.exp(B,3)
```

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

2. Comprobad, con los datos del ejemplo anterior, que si P es la matriz de vectores propios de M en columna y D la matriz diagonal cuyas entradas son los valores propios de M, entoces se cumple la siguiente igualdad llamada **descomposición canónica**:

$$M = P \cdot D \cdot P^{-1}$$

```r
#Esta es la matriz M que vamos a emplear
M = rbind(c(2,6,-8), c(0,6,-3), c(0,2,1))
M
```

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

```r
eigen(M)
```

```
## eigen() decomposition
## $values
## [1] 4 3 2
##
## $vectors
##           [,1]       [,2] [,3]
## [1,] 0.2672612 -0.8164966    1
## [2,] 0.8017837  0.4082483    0
## [3,] 0.5345225  0.4082483    0
```

```r
#Cálculo del parametro P
P <- eigen(M)$vectors
P
```

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

```r
#Cálculo del parámetro D
D <- diag(eigen(M)$values)
D
```

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

```r
#solve(P) es la inversa de la matriz P
#Empleo round para contrarrestar el error por redondeo
M == round(P %*% D %*% solve(P),2)
```

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