# Matrices

Adrian Vitys

11/1/2022

## Como se definen las matrices

```
# Los coloca de arriba a abajo
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
```

```
# Coloca de izquierda a derecha
M = matrix(1:12, nrow = 4, byrow = T)
M
```

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

```
# Matriz de 4 filas y 6 columnas
M = matrix(1, nrow = 4, ncol = 6)
M
```

```
##      [,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
```

### Con el vector vec = (1,2,3,4,5,6,7,8,9,10,11,12) crea una matriz de 3x4 de arriba abajo

```
vec = c(1,2,3,4,5,6,7,8,9,10,11,12)

mat = matrix(vec, nrow = 3, ncol = 4)
mat
```

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

### Se puede construir matrices con las funciones rbind (row bind) y cbind (column bind) de la siguiente forma

```
vector1 = c(1,2,3)
vector2 = c(2,3,5)
vector3 = c(3,76,24)
vector4 = c(4,0,10)
vector5 = c(5,7,8)
matriz = rbind(vector1, vector2, vector3, vector4, vector5)
matriz
```

```
##         [,1] [,2] [,3]
## vector1    1    2    3
## vector2    2    3    5
## vector3    3   76   24
## vector4    4    0   10
## vector5    5    7    8
```

### Se puede construir una matriz diagonal (diagonal con números y lo demás 0) con diag

```
diagonal = diag(c(1,2,3,4))
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
```

### De la siguiente forma podemos acceder a valores concretos de una matriz

```
# Obtener la fila 2 columna 3
matriz[2,3]
```

```
## vector2
##       5
```

```
# Obtener la fila 2
matriz[2,]
```

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

```r
# Obtener la columna 3
matriz[,3]
```

```
## vector1 vector2 vector3 vector4 vector5
##       3       5      24      10       8
```

```r
# Obtener las filas 2,3 y 5 y la columna 1,2,3
matriz[c(2,3,5), 1:3]
```

```
##         [,1] [,2] [,3]
## vector2    2    3    5
## vector3    3   76   24
## vector5    5    7    8
```

# Funciones de una matriz

```r
# Obtener la diagonal de la matriz
diag(matriz)
```

```
## [1]  1  3 24
```

```r
# Obtener el número de filas de la matriz
nrow(matriz)
```

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

```r
# Obtener el número de columnas de la matriz
ncol(matriz)
```

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

```r
# Obtener las dimensiones de la matriz
dim(matriz)
```

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

```r
# Obtener la suma de todas las entradas de la matriz
sum(matriz)
```

```
## [1] 153
```

```r
# Obtener el producto de todas las entradas de la matriz
prod(matriz)
```

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

```r
# Obtener la media aritmética de todas las entradas de la matriz
mean(matriz)
```

```
## [1] 10.2
```

```r
# Obtener las sumas por columnas de la matriz
colSums(matriz)
```

```
## [1] 15 88 50
```

```r
# Obtener las sumas por filas de la matriz
rowSums(matriz)
```

```
## vector1 vector2 vector3 vector4 vector5
##       6      10     103      14      20
```

```r
# Obtener la media aritmética por columnas de la matriz
colMeans(matriz)
```

```
## [1]  3.0 17.6 10.0
```

```r
# Obtener la media aritmética por filas de la matriz
rowMeans(matriz)
```

```
##   vector1   vector2   vector3   vector4   vector5
##  2.000000  3.333333 34.333333  4.666667  6.666667
```

### Aplicar otras funciones a una matriz

```r
# La raiz cuadrada de las sumas al cuadrado de las filas, Margin = 1 es filas y 2 es columnas
apply(matriz, MARGIN = 1, FUN = function(x){sqrt(sum(x^2))})
```

```
##   vector1   vector2   vector3   vector4   vector5
##  3.741657  6.164414 79.755878 10.770330 11.747340
```

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

```
## [1]  7.416198 76.406806 27.820855
```

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

```
##         [,1] [,2] [,3]
## vector1    1    2    3
## vector2    2    3    5
## vector3    3   76   24
## vector4    4    0   10
## vector5    5    7    8
```

### Otras funciones

```r
# Obtener la traspuesta de una matriz
t(matriz)
```

```
##      vector1 vector2 vector3 vector4 vector5
## [1,]       1       2       3       4       5
## [2,]       2       3      76       0       7
## [3,]       3       5      24      10       8
```

```r
# Multiplicar matrices (columnas de una matriz * filas de la otra matriz por lo que solo se podrían mul
matriz%*%t(matriz)
```

```
##         vector1 vector2 vector3 vector4 vector5
## vector1      14      23     227      34      43
## vector2      23      38     354      58      71
## vector3     227     354    6361     252     739
## vector4      34      58     252     116     100
## vector5      43      71     739     100     138
```

```r
# Cuando pones * realmente no se multiplica, se hace un producto tensorial
matriz * matriz
```

```
##         [,1] [,2] [,3]
## vector1    1    4    9
## vector2    4    9   25
## vector3    9 5776  576
## vector4   16    0  100
## vector5   25   49   64
```

```r
# Calcular la potencia de una matriz de forma aproximada
#install.packages("Biodem", dep = TRUE)

vecA = c(2,0,2,1,2,3,0,1,3)
matriz = matrix(vecA, nrow = 3, byrow = T)

library(Biodem)
mtx.exp(matriz, 2)
```

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

```r
# Calcular la potencia de una matriz de forma aproximada
#install.packages("expm", dep = TRUE)

library(expm)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'expm'
```

```
## The following object is masked from 'package:Matrix':
##
##     expm
```

```r
mtx.exp(matriz, 2)
```

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

## Dadas las matrices A(2, 0, 2 | 1, 2, 3 | 0, 1, 3) y B(3, 2, 1 | 1, 0, 0 | 1, 1, 1)

```r
vecA = c(2,0,2,1,2,3,0,1,3)
vecB = c(3,2,1,1,0,0,1,1,1)
A = matrix(vecA, nrow = 3, byrow = T)
B = matrix(vecB, nrow = 3, byrow = T)
A
```

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

```r
B
```

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

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

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

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

```
##      [,1] [,2] [,3]
## [1,]    8    5   15
## [2,]    2    0    2
## [3,]    3    3    8
```

```r
# A^2
mtx.exp(A, 2)
```

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

```r
# B^3
mtx.exp(B, 3)
```

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

## Operaciones con una matriz

```r
# Calcular el determinante de una matriz.
det(A)
```

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

```r
# Calcular el rango de una matriz.
qr(A)$rank
```

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

```r
# Calcular la inversa de una matriz cuadrada que es 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
```

```r
# Obtener la matriz identidad
solve(A)%*%A
```

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

### Resolver sistemas de ecuaciones lineales

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

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

### Valores y Vectores propios

```r
# calcular los valores(vaps) y vectores(veps) propios
eigen(A)
```

```
## 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
```

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

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

```r
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
```

## Comprobar la descomposición canónica de M:

### Descomposición canónica: M = P * D * P invertida

#### P = Matriz de vectores propios de M en columna

#### D = Matriz diagonal cuyas entradas son los valores propios de M

```r
# M viene dada por:
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
# Obtener la matriz de vectores propios de M en columna:
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
# Obtener la matriz diagonal cuyas entradas son los valores propios de M
D = diag(eigen(M)$values)
D
```

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

```r
# Obtener la matriz invertida de M
Pinv = solve(P)
Pinv
```

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

```r
# Calcular la descomposicion canonica
canonical = P%*%D%*%Pinv

# Comprobar si M y su descomposicion canonica es identica
all.equal(M,canonical)
```

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