Lección 4

Matrices

Una *matriz de orden* $n \times m$ es una tabla rectangular de números (o, en algunas situaciones específicas, algún otro tipo de datos: valores lógicos, etiquetas...) formada por n filas y m columnas. Una matriz es *cuadrada* cuando tiene el mismo número de filas que de columnas, es decir, cuando $n = m$; en este caso, decimos que la matriz es de *orden* n. En matemáticas, es costumbre escribir las matrices rodeadas por paréntesis para marcar con claridad sus límites. Por ejemplo,

$$A = \begin{pmatrix} 2 & 1 & -3 \\ -4 & 3 & \sqrt{3} \end{pmatrix}$$

es una matriz 2×3.

4.1. Construcción de matrices

Disponemos de dos maneras básicas de definir una matriz en `R`. En primer lugar, la instrucción

`matrix(`*vector*`, nrow=`n`, byrow=`*valor lógico*`)`

define una matriz de n filas (*rows*) formada por las entradas del *vector*. Si se entra `byrow=TRUE`, la matriz se construye por filas, mientras que con `byrow=FALSE` se construye por columnas; este último es el valor por defecto, por lo que no hace falta especificarlo. En vez de emplear `nrow`, se puede indicar el número de columnas con `ncol`. Veamos algunos ejemplos:

```
> matrix(1:6, nrow=2)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> matrix(1:6, nrow=3)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> matrix(1:6, nrow=2, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> matrix(1:6, nrow=3, byrow=TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Observad cómo muestra `R` las matrices: indica las filas con `[i,]`, donde i es el índice de la fila, y las columnas con `[,j]`, donde j es el índice de la columna.

Para construir una matriz de n filas o n columnas, es necesario que la longitud del vector al que se aplica `matrix` sea múltiplo de n. Si no es así, `R` rellena la última fila o columna con

entradas del principio del vector y emite un mensaje de advertencia.

```
> matrix(1:6, nrow=4) #6 no es múltiplo de 4
     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3    1
[4,]    4    2
Warning message:
In matrix(1:6, nrow = 4) :
  data length [6] is not a sub-multiple or multiple of the number
  of rows [4]
```

En particular, se puede definir una matriz constante aplicando la función `matrix` a un número. En este caso, se han de usar los parámetros `nrow` y `ncol` para especificar el orden de la matriz.

```
> matrix(1, nrow=2, ncol=3)   #Matriz constante 1 de orden 2x3
     [,1] [,2] [,3]
[1,]    1    1    1
[2,]    1    1    1
```

Otra posible manera de definir matrices es combinando filas o columnas. La instrucción

`rbind`(*vector1, vector2, ...*)

construye la matriz de filas *vector1, vector2, ...* (que han de tener la misma longitud) en este orden. Si en lugar de `rbind` se usa `cbind`, se obtiene la matriz cuyas columnas son los vectores a los que se aplica.

```
> rbind(c(1,0,2),c(2,3,6),c(1,2,0))
     [,1] [,2] [,3]
[1,]    1    0    2
[2,]    2    3    6
[3,]    1    2    0
> cbind(c(1,0,2),c(2,3,6),c(1,2,0))
     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    0    3    2
[3,]    2    6    0
```

Con las funciones `cbind` o `rbind` también podemos añadir columnas, o filas, a una matriz; en concreto, si A es una matriz de orden $n \times m$ y v es un vector de longitud n, la instrucción `cbind(A, v)` define la matriz de orden $n \times (m+1)$ que tiene como primeras m columnas las de A y como columna $m+1$ el vector v; de manera similar, `cbind(v, A)` define la matriz de orden $n \times (m+1)$ que tiene como primera columna el vector v y después las columnas de A. Con `cbind` también podemos concatenar por columnas dos matrices con el mismo número de filas.

La instrucción `rbind` es similar a `cbind`, pero actúa por filas en vez de por columnas: permite añadir filas arriba o abajo de una matriz ya existente, y, en general, concatenar por filas dos matrices con el mismo número de columnas.

```
> A=matrix(c(1,2,3,4), nrow=2)
```

```
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> cbind(A,c(5,6))
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> cbind(c(5,6),A)
     [,1] [,2] [,3]
[1,]    5    1    3
[2,]    6    2    4
> cbind(c(7,8),c(5,6),A)
     [,1] [,2] [,3] [,4]
[1,]    7    5    1    3
[2,]    8    6    2    4
> cbind(A,A)
     [,1] [,2] [,3] [,4]
[1,]    1    3    1    3
[2,]    2    4    2    4
> rbind(A,c(10,12))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]   10   12
> rbind(A,A)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    1    3
[4,]    2    4
```

Como pasaba con los vectores, todas las entradas de una matriz en `R` han de ser del mismo tipo de datos, y si una matriz contiene datos de diferentes tipos, automáticamente los convierte a un tipo que pueda ser común a todos ellos.

4.2. Acceso a entradas y submatrices

La *entrada* (i, j) de una matriz es el elemento situado en su fila i y su columna j. Por ejemplo, las entradas $(1, 2)$ y $(2, 1)$ de la matriz

$$A = \begin{pmatrix} 2 & 1 & -3 \\ -4 & 3 & \sqrt{3} \end{pmatrix}$$

son, respectivamente, 1 y -4.

El acceso a las entradas de una matriz se realiza como en los vectores, sólo que ahora en las matrices podemos especificar la fila y la columna:

- `M[i, j]` indica la entrada (i, j) de la matriz `M`.
- `M[i, ]` indica la fila i-ésima de `M`.

- `M[ ,j]` indica la columna j-ésima de `M`.

En los dos últimos casos, el resultado es un vector. Si queremos que el resultado sea una matriz de una sola fila o de una sola columna, respectivamente, tenemos que añadir el parámetro `drop=FALSE` dentro de los corchetes.

```
> M=matrix(c(1,3,5,2,3,6,2,9,8,4,2,5), nrow=3, byrow=TRUE)
> M
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    2
[2,]    3    6    2    9
[3,]    8    4    2    5
> M[2,4]  #Entrada (2,4)
[1] 9
> M[3,1]  #Entrada (3,1)
[1] 8
> M[1, ]  #Fila 1
[1] 1 3 5 2
> M[1, , drop=FALSE] #ATENCIÓN: fijaos en las dos comas
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    2
> M[ ,3]  #Columna 3
[1] 5 2 2
> M[ ,3, drop=FALSE]
     [,1]
[1,]    5
[2,]    2
[3,]    2
```

Estas construcciones sirven también para definir submatrices, y no sólo entradas, filas o columnas. Naturalmente, para indicar más de una fila o más de una columna tenemos que usar vectores de índices.

```
> M[c(1,2),c(1,3)] #Submatriz de filas 1, 2 y columnas 1, 3
     [,1] [,2]
[1,]    1    5
[2,]    3    2
> M[c(1,3), ] #Submatriz de filas 1, 3 y todas las columnas
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    2
[2,]    8    4    2    5
> M[ ,c(2,3,4)] #Submatriz de columnas 2, 3, 4 y todas las filas
     [,1] [,2] [,3]
[1,]    3    5    2
[2,]    6    2    9
[3,]    4    2    5
```

La diagonal principal de una matriz cuadrada (la que va de la esquina superior izquierda a la esquina inferior derecha) se obtiene con la función `diag`. Si la matriz no es cuadrada, `diag` produce el vector de entradas $(1,1),(2,2),\ldots$ hasta que se para en la última fila o la última columna.

```
> A=matrix(1:9, nrow=3, byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> diag(A)
[1] 1 5 9
> B=matrix(1:10, nrow=2, byrow=TRUE)
> B
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
> diag(B)
[1] 1 7
```

4.3. Algunas funciones para matrices

Las *dimensiones* de una matriz, es decir, sus números de filas y de columnas, se obtienen con las funciones `nrow` y `ncol`, respectivamente. Si queremos un vector formado por las dos dimensiones, podemos emplear la función `dim`.

```
> X=matrix(c(1,2,4,3,5,1,4,6,7,1,6,4), byrow=TRUE, nrow=2)
> X
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    2    4    3    5    1
[2,]    4    6    7    1    6    4
> nrow(X)
[1] 2
> ncol(X)
[1] 6
> dim(X)
[1] 2 6
```

La mayoría de las funciones numéricas para vectores se pueden aplicar a matrices. Por ejemplo, podemos usar las funciones `sum`, `prod` o `mean` para obtener la suma, el producto o la media, respectivamente, de todas las entradas de una matriz.

```
> A=matrix(c(1,2,1,3,-1,3), nrow=2, byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    1    2    1
[2,]    3   -1    3
> sum(A)
[1] 9
> mean(A)
[1] 1.5
```

En estadística a veces es necesario calcular la suma o la media por filas o por columnas de una matriz. Esto se puede llevar a cabo con las instrucciones siguientes:

- `colSums` produce un vector con las sumas de las columnas.
- `rowSums` produce un vector con las sumas de las filas.
- `colMeans` produce un vector con las medias de las columnas.
- `rowMeans` produce un vector con las medias de las filas.

```
> A=rbind(c(1,2,3,2),c(2,5,3,1),c(4,1,2,4))
> A
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    2
[2,]    2    5    3    1
[3,]    4    1    2    4
> colSums(A)  #Sumas de columnas
[1] 7 8 8 7
> rowSums(A)  #Sumas de filas
[1] 8 11 11
> colMeans(A)  #Medias de columnas
[1] 2.333333 2.666667 2.666667 2.333333
> rowMeans(A)  #Medias de filas
[1] 2.00 2.75 2.75
```

Si queremos aplicar otras funciones a las filas o las columnas de una matriz, podemos emplear la función `apply`. Su estructura básica es

`apply(A, MARGIN=...,` `FUN=`*función*`)`,

donde `A` es una matriz, la *función* es la que queremos aplicar, y el valor de `MARGIN` ha de ser 1 si la queremos aplicar por filas, 2 si la queremos aplicar por columnas, o `c(1, 2)` si la queremos aplicar entrada a entrada; como pasaba con los vectores, en muchas ocasiones podemos aplicar una función a todas las entrada de una matriz entrando la matriz en su argumento, pero a veces es necesaria usar `apply` con `MARGIN=c(1,2)`.

Por ejemplo, para calcular la *norma euclídea* de las filas de la matriz A anterior (la raíz cuadrada de la suma de los cuadrados de sus entradas), para ordenar cada una de sus columnas, y para calcular la matriz de entradas las raíces cuadradas de sus entradas, haríamos lo siguiente:

```
> f=function(x){sqrt(sum(x^2))}
> apply(A, MARGIN=1, FUN=f)  #Normas euclídeas de filas
[1] 4.242641 6.244998 6.082763
> A_ord=apply(A, MARGIN=2, FUN=sort)  #Matriz con cada columna de A
    ordenada
> A_ord
     [,1] [,2] [,3] [,4]
[1,]    1    1    2    1
[2,]    2    2    3    2
[3,]    4    5    3    4
> sqrt(A)   #Matriz de raíces cuadradas
         [,1]     [,2]     [,3]     [,4]
```

```
[1,] 1.000000 1.414214 1.732051 1.414214
[2,] 1.414214 2.236068 1.732051 1.000000
[3,] 2.000000 1.000000 1.414214 2.000000
> apply(A, MARGIN=c(1,2), FUN=sqrt)  #La anterior, usando apply
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.414214 1.732051 1.414214
[2,] 1.414214 2.236068 1.732051 1.000000
[3,] 2.000000 1.000000 1.414214 2.000000
```

4.4. Cálculo matricial

Las operaciones algebraicas usuales con matrices numéricas se indican de la manera siguiente:[1]

- La traspuesta se obtiene con la función `t`.
- La suma de matrices se indica con el símbolo usual `+`.
- El producto de un escalar por una matriz se indica con el símbolo usual `*`.
- El producto de matrices se indica con `%*%`.

¡Atención! Si «multiplicáis» dos matrices con el símbolo `*`, no obtenéis el producto de las dos matrices, sino la matriz que tiene en cada entrada (i, j) el producto de las entradas (i, j) de cada una de las dos matrices. Esto a veces es útil, pero no es el producto de matrices. De manera similar, si M es una matriz y entráis `M^n`, el resultado no es la potencia n-ésima de M, sino la matriz que tiene en cada entrada la potencia n-ésima de la entrada correspondiente de M. De nuevo, esto a veces es útil, pero muy pocas veces coincide con la potencia n-ésima de M.

```
> A=matrix(c(1,2,1,3), nrow=2, byrow=TRUE)
> A
     [,1] [,2]
[1,]    1    2
[2,]    1    3
> B=matrix(c(-2,4,3,1,0,2), nrow=3, byrow=TRUE)
> B
     [,1] [,2]
[1,]   -2    4
[2,]    3    1
[3,]    0    2
> C=matrix(c(1,0,1,2,1,0), nrow=2, byrow=TRUE)
> C
     [,1] [,2] [,3]
[1,]    1    0    1
[2,]    2    1    0
> t(B) #Traspuesta
     [,1] [,2] [,3]
```

[1] Si necesitáis repasar los conceptos básicos sobre operaciones de matrices que aparecen en esta sección, podéis consultar la entrada sobre matrices en la *Wikipedia*, `http://es.wikipedia.org/wiki/Matriz_(matemáticas)`.

```
[1,]   -2    3    0
[2,]    4    1    2
> t(B)+C #Suma
     [,1] [,2] [,3]
[1,]   -1    3    1
[2,]    6    2    2
> 5*A #Producto por escalar
     [,1] [,2]
[1,]    5   10
[2,]    5   15
> C%*%B #Producto
     [,1] [,2]
[1,]   -2    6
[2,]   -1    9
> (C%*%B)%*%A #Producto
     [,1] [,2]
[1,]    4   14
[2,]    8   25
> A^2 #Esto no es elevar al cuadrado
     [,1] [,2]
[1,]    1    4
[2,]    1    9
> A%*%A  #Esto sí
     [,1] [,2]
[1,]    3    8
[2,]    4   11
```

La versión básica de `R` no lleva ninguna función para calcular potencias de matrices, y hay que cargar algún paquete adecuado para disponer de ella.[2] Por ejemplo, el paquete `Biodem` dispone de la función `mtx.exp`, y el paquete `expm` dispone de la operación `%^%`. No obstante, hay que tener presente que estas funciones no calculan las potencias de manera exacta, sino que emplean algoritmos de cálculo numérico para aproximarlas a cambio de calcularlas rápido, y por lo tanto no siempre dan el resultado exacto.

```
> A=matrix(c(1,2,1,3), nrow=2, byrow=TRUE)
> #Instalamos y cargamos el paquete Biodem
...
> mtx.exp(A, 20) #A^20
             [,1]         [,2]
[1,] 58063278153 158631825968
[2,] 79315912984 216695104121
> #Instalamos y cargamos el paquete expm
...
> A%^%20 #A^20
             [,1]         [,2]
[1,] 58063278153 158631825968
[2,] 79315912984 216695104121
```

El determinante de una matriz cuadrada se calcula con la función `det`.

[2] Explicamos el manejo de paquetes en la Sección 0.5.

```
> Y=rbind(c(1,3,2),c(2,3,5),c(-1,3,2))
> Y
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    3    5
[3,]   -1    3    2
> det(Y)
[1] -18
```

Para ganar en rapidez, `R` calcula los determinantes usando un método numérico que a veces produce efectos no deseados como el siguiente:

```
> A=matrix(c(3,10,30,100), nrow=2)
> A
     [,1] [,2]
[1,]    3   30
[2,]   10  100
> det(A)
[1] 3.552714e-14
```

Pero, de hecho,

$$\begin{vmatrix} 3 & 30 \\ 10 & 100 \end{vmatrix} = 3 \cdot 100 - 30 \cdot 10 = 0.$$

Por lo tanto, este determinante $3.552714 \cdot 10^{-14}$ es en realidad 0.

El rango de una matriz A se puede calcular mediante la instrucción `qr(A)$rank`.

```
> X=matrix(c(0,1,0,-7,3,-1,16,-3,4), nrow=3, byrow=TRUE)
> X
     [,1] [,2] [,3]
[1,]    0    1    0
[2,]   -7    3   -1
[3,]   16   -3    4
> det(X)
[1] 12
> qr(X)$rank
[1] 3
> Y=rbind(rep(0,3),rep(1,3))
> Y
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    1    1    1
> qr(Y)$rank
[1] 1
```

Podemos calcular la inversa de una matriz invertible con la instrucción `solve`. Por ejemplo, para calcular la inversa A^{-1} de la matriz

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 0 & 2 & -1 \\ 2 & 1 & 2 \end{pmatrix}$$

podemos entrar lo siguiente:

```
> A=matrix(c(1,3,4,0,2,-1,2,1,2), nrow=3, byrow=TRUE)
> solve(A)
           [,1]       [,2]       [,3]
[1,] -0.2941176  0.1176471  0.64705882
[2,]  0.1176471  0.3529412 -0.05882353
[3,]  0.2352941 -0.2941176 -0.11764706
```

Obtenemos

$$A^{-1} = \begin{pmatrix} -0.2941176 & 0.1176471 & 0.64705882 \\ 0.1176471 & 0.3529412 & -0.05882353 \\ 0.2352941 & -0.2941176 & -0.11764706 \end{pmatrix}.$$

Comprobemos si esta matriz es realmente la inversa de A:

```
> A%*%solve(A)
             [,1] [,2]          [,3]
[1,] 1.000000e+00    0  0.000000e+00
[2,] 5.551115e-17    1 -2.775558e-17
[3,] 0.000000e+00    0  1.000000e+00
> solve(A)%*%A
     [,1]          [,2]          [,3]
[1,]    1 0.000000e+00 0.000000e+00
[2,]    0 1.000000e+00 1.110223e-16
[3,]    0 2.775558e-17 1.000000e+00
```

Los productos `A%*%solve(A)` y `solve(A)%*%A` no han dado exactamente la matriz identidad, como deberían, pero la diferencia está en la decimosexta cifra decimal. Recordad que `R` no trabaja con precisión infinita, a veces los errores de redondeo son inevitables.

La función `solve` también sirve para resolver sistemas de ecuaciones lineales

$$\left.\begin{array}{c} a_{1,1}x_1 + \cdots + a_{1,n}x_n = b_1 \\ a_{2,1}x_1 + \cdots + a_{2,n}x_n = b_2 \\ \vdots \\ a_{n,1}x_1 + \cdots + a_{n,n}x_n = b_n \end{array}\right\}$$

cuya *matriz del sistema*

$$A = \begin{pmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} \end{pmatrix}$$

sea cuadrada e invertible. Para ello, se usaría la instrucción

`solve(A, b)`,

donde `A` es la matriz A del sistema y `b` es el vector de términos independientes

$$b = (b_1, \ldots, b_n).$$

Por ejemplo, para resolver el sistema

$$\left.\begin{array}{r} x + 6y - 3z = 7 \\ 2x - y + z = 2 \\ x + y - z = 3 \end{array}\right\},$$

que escrito en forma matricial es

$$\begin{pmatrix} 1 & 6 & -3 \\ 2 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 7 \\ 2 \\ 3 \end{pmatrix},$$

podemos entrar

```
> A=matrix(c(1,6,-3,2,-1,1,1,1,-1), nrow=3, byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    1    6   -3
[2,]    2   -1    1
[3,]    1    1   -1
> b=c(7,2,3)
> solve(A, b)
[1] 1.6666667 0.4444444 -0.8888889
```

y obtenemos que la solución del sistema (redondeada a 7 cifras decimales) es

$$x = 1.6666667,\ y = 0.4444444,\ z = -0.8888889.$$

4.5. Valores y vectores propios

La función básica para calcular valores y vectores propios[3] es `eigen` (en inglés, los valores y vectores propios se llaman *eigenvalues* y *eigenvectors*, respectivamente). Supongamos, por ejemplo, que queremos calcular los valores propios de la matriz

$$A = \begin{pmatrix} 2 & 6 & -8 \\ 0 & 6 & -3 \\ 0 & 2 & 1 \end{pmatrix}.$$

```
> A=matrix(c(2,6,-8,0,6,-3,0,2,1), nrow=3, byrow=TRUE)
> A
     [,1] [,2] [,3]
[1,]    2    6   -8
[2,]    0    6   -3
[3,]    0    2    1
> eigen(A)
$values
[1] 4 3 2

$vectors
          [,1]       [,2] [,3]
[1,] 0.2672612 -0.8164966    1
[2,] 0.8017837  0.4082483    0
```

[3] Si necesitáis repasar las definiciones de vector y valor propio de una matriz y de descomposición canónica de una matriz diagonalizable, y por qué son importantes, podéis consultar las entradas correspondientes de la *Wikipedia*: `http://es.wikipedia.org/wiki/Vector_propio_y_valor_propio` y `http://es.wikipedia.org/wiki/Matriz_diagonalizable`.

```
[3,] 0.5345225  0.4082483    0
```

El resultado de `eigen` es una `list` con dos objetos: `values` y `vectors`.

```
> str(eigen(A))
List of 2
 $ values : num [1:3] 4 3 2
 $ vectors: num [1:3, 1:3] 0.267 0.802 0.535 -0.816 0.408 ...
> eigen(A)$values
[1] 4 3 2
> eigen(A)$vectors
          [,1]       [,2] [,3]
[1,] 0.2672612 -0.8164966    1
[2,] 0.8017837  0.4082483    0
[3,] 0.5345225  0.4082483    0
```

El objeto `values` es un vector con los valores propios, y el objeto `vectors` es una matriz cuyas columnas son vectores propios: la primera columna es un vector propio del primer valor propio del vector `values`, la segunda lo es del segundo, y así sucesivamente. De este modo, del resultado anterior deducimos que los valores propios de A son 4, 3 y 2, y que

$$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0.2672612 \\ 0.8017837 \\ 0.5345225 \end{pmatrix}$$

son vectores propios de A de valores propios 2 y 4, respectivamente (o, para ser precisos, el segundo vector es un vector propio de valor propio 4 redondeado a 7 cifras decimales).

Es importante tener presentes algunas propiedades de la función `eigen`:

- Da los valores propios en orden decreciente de su valor absoluto (o de su módulo, si hay valores propios complejos) y repetidos tantas veces como su multiplicidad.
- Si hay algún valor propio con multiplicidad mayor que 1, da tantos vectores de este valor propio como su multiplicidad. Además, en este caso procura que estos vectores propios sean linealmente independientes. Por lo tanto, cuando da vectores propios repetidos de algún valor propio es porque para este valor propio no existen tantos vectores propios linealmente independientes como su multiplicidad y, por consiguiente, la matriz no es diagonalizable.

Del resultado de `eigen(A)` se puede obtener una descomposición canónica

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

de una matriz diagonalizable A: basta tomar como D la matriz diagonal que tiene como diagonal principal el vector `eigen(A)$values` y como P la matriz `eigen(A)$vectors`.

Para construir una matriz diagonal cuya diagonal principal sea un *vector* dado, podemos usar la instrucción `diag(`*vector*`)`. Si aplicamos `diag` a un número n, produce la matriz identidad de orden n.

```
> diag(c(2,5,-1))
     [,1] [,2] [,3]
[1,]    2    0    0
[2,]    0    5    0
[3,]    0    0   -1
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
```

La función `diag` ya había salido en la Sección 4.2: si se aplica a una matriz, se obtiene el vector formado por sus entradas $(1,1),(2,2),\ldots$, pero si se aplica a un vector, produce una matriz diagonal.

```
> B=matrix(1:10, nrow=2, byrow=TRUE)
> B
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
> diag(B)
[1] 1 7
> diag(diag(B))
     [,1] [,2]
[1,]    1    0
[2,]    0    7
```

Veamos un ejemplo de uso de `eigen` para calcular una descomposición canónica. Como hemos visto, la matriz

$$A = \begin{pmatrix} 2 & 6 & -8 \\ 0 & 6 & -3 \\ 0 & 2 & 1 \end{pmatrix}$$

es de orden 3 y tiene sus tres valores propios diferentes. Por lo tanto, es diagonalizable y las matrices de una posible descomposición canónica serían las siguientes:

```
> D=diag(eigen(A)$values) #Matriz diagonal D de valores propios
> D
     [,1] [,2] [,3]
[1,]    4    0    0
[2,]    0    3    0
[3,]    0    0    2
> P=eigen(A)$vectors #Matriz P de vectores propios
> P
          [,1]       [,2] [,3]
[1,] 0.2672612 -0.8164966    1
[2,] 0.8017837  0.4082483    0
[3,] 0.5345225  0.4082483    0
```

Comprobemos que, efectivamente, $A = P \cdot D \cdot P^{-1}$:

```
> P%*%D%*%solve(P)
     [,1] [,2] [,3]
[1,]    2    6   -8
[2,]    0    6   -3
[3,]    0    2    1
> A
     [,1] [,2] [,3]
[1,]    2    6   -8
[2,]    0    6   -3
[3,]    0    2    1
```

Por consiguiente, una descomposición canónica de A es (salvo redondeos)

$$A = \begin{pmatrix} 0.2672612 & -0.8164966 & 1 \\ 0.8017837 & 0.4082483 & 0 \\ 0.5345225 & 0.4082483 & 0 \end{pmatrix} \cdot \begin{pmatrix} 4 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{pmatrix} \cdot \begin{pmatrix} 0.2672612 & -0.8164966 & 1 \\ 0.8017837 & 0.4082483 & 0 \\ 0.5345225 & 0.4082483 & 0 \end{pmatrix}^{-1}.$$

Veamos otro ejemplo. Queremos decidir si la matriz

$$B = \begin{pmatrix} 0 & 1 & 0 \\ -7 & 3 & -1 \\ 16 & -3 & 4 \end{pmatrix}$$

es diagonalizable y, en caso afirmativo, obtener una descomposición canónica.

```
> B=matrix(c(0,1,0,-7,3,-1,16,-3,4), nrow=3, byrow=TRUE)
> eigen(B)
$values
[1] 3 2 2

$vectors
           [,1]       [,2]       [,3]
[1,] -0.1301889 -0.1525742 -0.1525742
[2,] -0.3905667 -0.3651484 -0.3651484
[3,]  0.9113224  0.9128709  0.9128709
```

Da dos veces el mismo vector propio de valor propio 2. Esto significa que B no tiene dos vectores propios linealmente independientes de este valor propio, y, por lo tanto, no es diagonalizable.

4.6. Matrices complejas

La mayoría de las instrucciones explicadas en esta lección para operar con matrices numéricas sirven sin ningún cambio para operar con matrices de entradas números complejos. Por ejemplo, para elevar al cuadrado la matriz

$$\begin{pmatrix} 3-2i & 5+3 \\ 1+2i & 2-i \end{pmatrix},$$

podemos entrar:

```
> A=matrix(c(3-2i,5+3i,1+2i,2-1i), nrow=2, byrow=TRUE)
> A
     [,1] [,2]
[1,] 3-2i 5+3i
[2,] 1+2i 2-1i
> A%*%A
      [,1]   [,2]
[1,]  4+1i 34+0i
[2,] 11+7i  2+9i
```

Para calcular sus valores y vectores propios, podemos entrar:

```
> eigen(A)
$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
```

Y para resolver el sistema de ecuaciones

$$\left.\begin{aligned}(3-2i)x+(5+3i)y &= 2-i\\ (1+2i)x+(2-i)y &= 3\end{aligned}\right\}$$

podemos entrar:

```
> A=matrix(c(3-2i,5+3i,1+2i,2-1i), nrow=2, byrow=TRUE)
> b=c(2-1i,3)
> solve(A, b)
[1] 0.4705882-0.7176471i 0.4823529+0.1294118i
```

La excepción más importante son los determinantes.

```
> det(A)
Error in determinant.matrix(x, logarithm = TRUE, ...) :
  'determinant' not currently defined for complex matrices
```

Pero resulta que el determinante de una matriz es igual al producto de sus valores propios, contados con su multiplicidad. Por lo tanto, para calcular el determinante de una matriz compleja A podemos usar `prod(eigen(A)$values)`.

```
> A=matrix(c(3-2i, 5+3i, 1+2i, 2-1i), nrow=2, byrow=TRUE)
> prod(eigen(A)$values)
[1] 5-20i
```

4.7. Guía rápida

- `matrix` sirve para construir una matriz a partir de un vector. Algunos parámetros importantes:

  - `byrow`: un parámetro lógico para indicar si la matriz se construye por filas (igualado a `TRUE`) o por columnas (valor por defecto).
  - `nrow`: el número de filas.
  - `ncol`: el número de columnas.

- `cbind` concatena vectores y matrices por columnas.
- `rbind` concatena vectores y matrices por filas.
- *matriz*`[...,...]` se usa para especificar un elemento, una fila, una columna o una submatriz de la *matriz*. Si extraemos una fila o una columna con el parámetro `drop=FALSE`, el resultado es una matriz y no un vector.
- `diag` tiene dos funciones: aplicada a un vector, construye una matriz diagonal, y aplicada a una matriz, extrae su diagonal principal.
- `nrow` da el número de filas de una matriz.
- `ncol` da el número de columnas de una matriz.
- `dim` da un vector con las dimensiones de una matriz.
- `sum`, `prod` y `mean` calculan, respectivamente, la suma, el producto y la media de las entradas de una matriz.
- `colSums` y `rowSums` suman, respectivamente, las columnas y las filas de una matriz.
- `colMeans` y `rowMeans` calculan, respectivamente, las medias de las columnas y de las filas de una matriz.
- `apply(`*matriz*`, MARGIN=..., FUN=`*función*`)` aplica la *función* a las filas (`MARGIN=1`), a las columnas (`MARGIN=2`) o a todas las entradas (`MARGIN=c(1, 2)`) de la *matriz*.
- Símbolos de operaciones con matrices:

Función	Suma	Producto por escalar	Producto	Potencia (paquete `expm`)
Símbolo	`+`	`*`	`%*%`	`%^%`

- `t` calcula la traspuesta de una matriz.
- `det` calcula el determinante de una matriz.
- `qr(`*matriz*`)$rank` calcula el rango de la *matriz*.
- `solve`, aplicada a una matriz invertible A, calcula su inversa A^{-1}, y aplicada a una matriz invertible A y un vector b, calcula $A^{-1} \cdot b$.
- `eigen` calcula los valores y vectores propios de una matriz. El resultado es una `list` con dos componentes:
  - `values`: un vector con los valores propios.
  - `vectors`: una matriz cuyas columnas son vectores propios de los correspondientes valores propios.

4.8. Ejercicio

Sean x e y dos cantidades de las cuales efectuamos una serie de k observaciones conjuntas, de forma que obtenemos una secuencia de pares de valores

$$(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k).$$

Como veíamos en la Lección 2, si queremos encontrar la recta $y = ax + b$ que aproxime mejor estas observaciones, una posibilidad es calcular los valores $a, b \in \mathbb{R}$ tales que

$$\sum_{i=1}^{k} (ax_i + b - y_i)^2$$

sea mínimo. De este modo encontraríamos la *recta de regresión por mínimos cuadrados*. Resulta (no lo demostraremos aquí) que los coeficientes a y b de esta recta de regresión se obtienen por medio de la fórmula

$$\begin{pmatrix} b \\ a \end{pmatrix} = (D_2 \cdot D_2^t)^{-1} \cdot D_2 \cdot w,$$

donde

$$w = \begin{pmatrix} y_1 \\ \vdots \\ y_k \end{pmatrix}, \quad D_2 = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_k \end{pmatrix}.$$

(a) Calculad de este modo los valores de a y b cuando las observaciones son las de la Tabla 4.1 (es la Tabla 2.1 de la Lección 2), y comprobad que obtenéis el mismo resultado que obteníamos en su momento con la función `lm`.

x	1	3	5	7	9	11	13
y	75	92	108	121	130	142	155

Tabla 4.1

De manera similar, si queremos obtener una función cuadrática $y = ax^2 + bx + c$ que aproxime los pares $(x_i, y_i)_{y=1,\ldots,k}$, podemos buscar los coeficientes a, b, c que minimicen el valor de

$$\sum_{i=1}^{k} (ax_i^2 + bx_i + c - y_i)^2.$$

Estos coeficientes se obtienen de manera similar, por medio de la fórmula

$$\begin{pmatrix} c \\ b \\ a \end{pmatrix} = (D_3 \cdot D_3^t)^{-1} \cdot D_3 \cdot w,$$

donde w es como antes y ahora

$$D_3 = \begin{pmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_k \\ x_1^2 & x_2^2 & \ldots & x_k^2 \end{pmatrix}.$$

(b) Calculad los valores de a, b, c para los pares (x, y) de la Tabla 4.1. Con `R`, podríamos calcular estos coeficientes definiendo un nuevo vector z con los cuadrados de x, y aplicando entonces la función

```
lm(y~x+z)
```

e interpretando que la z representa x^2. Comprobad que, efectivamente, dan lo mismo.