Listas y matrices

3

3.1 Listas

Maxima tiene una manera fácil de agrupar objetos, ya sean números, funciones, cadenas de texto, etc. y poder operar con ellos. Una lista se escribe agrupando entre corchetes los objetos que queramos separados por comas. Por ejemplo,

```
(%i1)   [0,1,-3];
(%o1)   [0,1,-3]
```

es una lista de números. También podemos escribir listas de funciones

```
(%i2)   [x,x^2,x^3]
(%o2)   [x,x^2,x^3]
```

o mezclar números, variables y texto

```
(%i3)   [0,1,-3,a,"hola"];
(%o3)   [0,1,-3,a,hola]
```

`first, second,...,tenth`	primera, segunda,...,décima entrada de una lista
`lista[i]`	entrada *i*-ésima de la lista
`last`	último elemento de una lista
`part`	busca un elemento dando su posición en la lista
`reverse`	invertir lista
`sort`	ordenar lista
`flatten`	unifica las sublistas en una lista
`length`	longitud de la lista
`unique`	elementos que sólo aparecen una vez en la lista

Los elementos que forman la lista pueden ser, a su vez, listas (aunque no es exactamente lo mismo, piensa en matrices como "listas de vectores"):

```
(%i4)   lista:[[1,2],1,[3,a,1]]
(%o4)   [[1,2],1,[3,a,1]]
```

Podemos referirnos a una entrada concreta de una lista. De hecho *Maxima* tiene puesto nombre a las diez primeras: `first, second,..., tenth`

```
(%i5)   first(lista);
(%o5)   [1,2]
(%i6)   second(lista);
(%o6)   1
```

last o podemos referirnos directamente al último término.

```
(%i7)   last(lista);
(%o7)   [3,a,1]
```

part Si sabemos la posición que ocupa, podemos refererirnos a un elemento de la lista utilizando `part`. Por ejemplo,

```
(%i8)   part(lista,1)
(%o8)   [1,2]
```

nos da el primer elemento de la lista anterior. Obtenemos el mismo resultado indicando la posición entre corchetes. Por ejemplo,

```
(%i9)   lista[3];
(%o9)   [3,a,1]
```

y también podemos anidar esta operación para obtener elementos de una sublista

```
(%i10)   lista[3][1];
(%o10)   3
```

Con `part` podemos extraer varios elementos de la lista enumerando sus posiciones. Por ejemplo, el primer y el tercer elemento de la lista son

```
(%i11)   part(lista,[1,3]);
(%o11)   [[1,2],[3,a,1]]
```

o el segundo término del tercero que era a su vez una lista:

```
(%i12)   part(lista,3,2);
(%o12)   a
```

El comando `flatten` construye una única lista con todas los elementos, sean estos listas o no. Mejor un ejemplo: **flatten**

```
(%i13)   flatten([[1,2],1,[3,a,1]])
(%o13)   [1,2,1,3,a,1]
```

La lista que hemos obtenido contiene todos los anteriores. Podemos eliminar los repetidos con `unique` **unique**

```
(%i14)   unique(%)
(%o14)   [1,2,3,a]
```

Vectores

En el caso de vectores, listas de números, tenemos algunas posibilidades más. Podemos sumarlos

```
(%i15)   v1:[1,0,-1];v2:[-2,1,3];
(%o15)   [1,0,-1]
(%o16)   [-2,1,3]
(%i17)   v1+v2;
(%o17)   [-1,1,2]
```

o multiplicarlos.

```
(%i18)   v1*v2;
(%o18)   [-2,0,-3]
```

Un momento, ¿cómo los hemos multiplicado? Término a término. Esto no tiene nada que ver con el producto escalar o con el producto vectorial. El producto escalar, por ejemplo, se indica con "."

```
(%i19)   v1.v2;
```

```
(%o19)  -5
```

sort Podemos ordenar los elementos de la lista (del vector en este caso)

```
(%i20)  sort(v1);
(%o20)  [-1,0,1]
```

length o saber cuántos elementos tiene

```
(%i21)  length(v1);
(%o21)  3
```

3.1.1 Construir y operar con listas

`makelist`	genera lista
`apply`	aplicar un operador a una lista
`map`	aplicar una función a una lista
`listp(`*expr*`)`	devuelve `true` si la expresión es una lista

makelist Los ejemplos que hemos visto de listas hasta ahora son mezcla de números y letras de forma bastante aleatoria. En la práctica, muchas de las listas que aparecen están definidas por alguna regla. Por ejemplo, queremos dibujar las funciones sen(x), sen($2x$),..., sen($20x$). Seguro que no tienes ganas de escribir la lista completa. Este es el papel de la orden `makelist`. Para escribir esa lista necesitamos la regla, la fórmula que la define, un parámetro y entre qué dos valores se mueve dicho parámetro:

```
(%i22)  makelist(sin(t*x),t,1,20)
        [sin(x),sin(2 x),sin(3 x),sin(4 x),sin(5 x),sin(6 x),
        sin(7 x),sin(8 x),sin(9 x),sin(10 x),sin(11 x),
(%o22)  sin(12 x),sin(13 x),
        sin(14 x),sin(15 x),
        sin(16 x),sin(17 x),sin(18 x),sin(19 x),sin(20 x)]
```

Las listas también se pueden utilizar como contadores. El caso que suele ser más útil es una lista cuyas entradas sean un rango de enteros. Por ejemplo, los primeros cien naturales empezamos en uno) son

```
(%i23)  makelist(i,i,1,100);
```

```
(%o23)  [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,
23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,
42,43,44,45,46,47,48,49,50,51,52,53,54,55,56,57,58,59,60,
61,62,63,64,65,66,67, 68,69,70,71,72,73,74,75,76,77,78,79,
80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,
99,100]
```

o si sólo queremos los pares:

```
(%i24)  makelist(2*i,i,1,50);
(%o24)  [2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,
42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,
80,82,84,86,88,90,92,94,96,98,100]
```

Ya que tenemos una lista, ¿cómo podemos "jugar" con sus elementos? Por ejemplo, ¿se puede calcular el cuadrado de los 100 primeros naturales? ¿Y su media aritmética o su media geométrica? Las órdenes `map` y `apply` nos ayudan a resolver este problema. La orden `map` permite aplicar una función a cada uno de los elementos de una lista. Por ejemplo, para calcular sen(1), sen(2),..., sen(10), hacemos lo siguiente

map

```
(%i25)  map(sin,makelist(i,i,1,10));
(%o25)  [sin(1),sin(2),sin(3),sin(4),sin(5),sin(6),sin(7),sin(8),
sin(9),sin(10)]
```

o si queremos la expresión decimal

```
(%i26)  %,numer
(%o26)  [0.8414709848079,0.90929742682568,0.14112000805987,
-0.75680249530793,-0.95892427466314,-0.27941549819893,
0.65698659871879,0.98935824662338,0.41211848524176,
-0.54402111088937]
```

La orden `apply`, en cambio, pasa todos los valores de la lista a un operador que, evidentemente, debe saber qué hacer con la lista. Ejemplos típicos son el operador suma o multiplicación. Por ejemplo

apply

```
(%i27)  apply("+",makelist(i,i,1,100));
(%o27)  5050
```

nos da la suma de los primeros 100 naturales.

Ejemplo 3.1. Vamos a calcular la media aritmética y la media geométrica de los 100 primeros naturales. ¿Cuál será mayor? ¿Recuerdas la desigualdad entre ambas medias? La media aritmética es la suma de todos los elementos dividido por la cantidad de elementos que sumemos:

```
(%i28)  apply("+",makelist(i,i,1,100))/100;
(%o28)  101/2
```

La media geométrica es la raíz n-ésima del producto de los n elementos:

```
(%i29)  apply("*",makelist(i,i,1,100))^(1/100);
(%o29)  17^(1/20) 19^(1/20) 23^(1/25) 37^(1/50) 41^(1/50) 43^(1/50) 47^(1/50) 2401^(1/25) 15625^(1/25) 531441^(1/25)
        63871474118205504453 0[30digits]99766363898994157944832^(1/100)
(%i30)  float(%);
(%o30)  37.9926893448343
```

Parece que la media geométrica es menor.

Ejemplo 3.2. ¿Cuál es el módulo del vector $(1, 3, -7, 8, 1)$? Tenemos que calcular la raíz cuadrada de la suma de sus coordenadas al cuadrado:

```
(%i31)  vector:[1,3,-7,8,1];
(%o31)  [1,3,-7,8,1]
(%i32)  sqrt(apply("+",vector^2));
(%o32)  2*sqrt(31)
```

También es posible calcular el módulo como la raíz cuadrada del producto escalar de un vector consigo mismo.

```
(%i33)  sqrt(vector.vector);
(%o33)  2*sqrt(31)
```

A la vista de estos ejemplos, ¿cómo podríamos definir una función que nos devuelva la media aritmética, la media geométrica de una lista o el módulo de un vector?

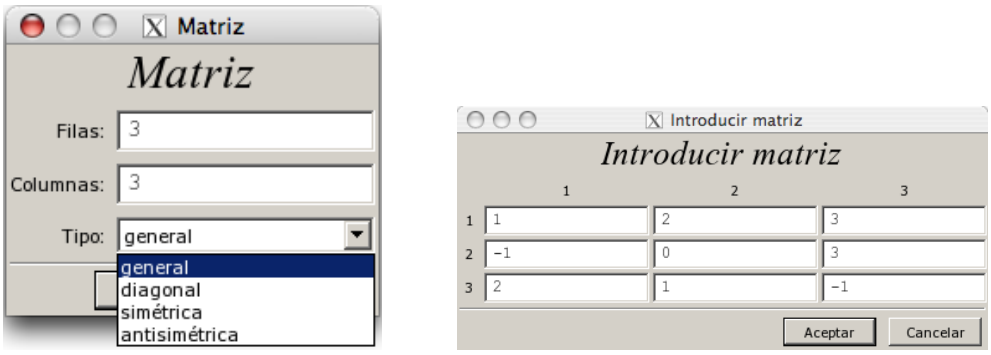
3.2 Matrices

matrix

Las matrices se escriben de forma parecida a las listas y, de hecho, sólo tenemos que agrupar las filas de la matriz escritas como listas bajo la orden `matrix`. Vamos a definir un par de matrices y un par de vectores que van a servir en los ejemplos en lo que sigue.

```
(%i34)  A:matrix([1,2,3],[-1,0,3],[2,1,-1]);
        B:matrix([-1,1,1],[1,0,0],[-3,7,2]);
        a:[1,2,1];
        b:[0,1,-1];
(%o35)  matrix([1,2,3],[-1,0,3],[2,1,-1])
(%o36)  matrix([-1,1,1],[1,0,0],[-3,7,2])
(%o37)  [1,2,1]
(%o37)  [0,-1,1]
```

En *wxMaxima* también podemos escribir una matriz usando el menú **Álgebra→Introducir matriz**. Nos aparece una ventana como las de la Figura 3.1 donde podemos rellenar los valores.



Seleccionar tipo de matriz Introducir valores

Figura 3.1 Introducir matriz

`matrix(`*`fila1,fila2,...`*`)`	matriz
`matrix_size(`*`matriz`*`)`	número de filas y columnas
`matrixp(`*`expresión`*`)`	devuelve `true` si *expresión* es una matriz

Las dimensiones de una matriz se pueden recuperar mediante la orden `matrix_size` que devuelve una lista con el número de filas y columnas.

matrix_size

```
(%i38)  matrix_size(A);
(%o38)  [3,3]
```

Observación 3.3. Aunque muy similares, *Maxima* distingue entre listas y matrices. La orden `matrixp(`*`expresión`*`)` devuelve `true` o `false` dependiendo de si la expresión es o no una matriz. Por ejemplo, los vectores *a* y *b* que hemos definido antes, ¿son o no son matrices?

matrixp

```
(%i39)  matrixp(a);
(%o39)  false
```

Aunque pueda parecer lo contrario, no son matrices, son listas.

```
(%i40)  listp(a);
(%o40)  true
```

Sólo es aceptado como matriz aquello que hallamos definido como matriz mediante la orden `matrix` o alguna de sus variantes. Al menos en *wxMaxima*, hay un pequeño truco para ver si algo es o no una matriz. ¿Cuál es la diferencia entre las dos siguientes salidas?

```
(%i41)  [1,2,3];
(%o41)  [1,2,3]
(%i42)  matrix([1,2,3]);
(%o42)  [1   2   3]
```

wxMaxima respeta algunas de las diferencias usuales entre vectores y matrices: no pone comas separando las entradas de las matrices y, además, dibuja los corchetes un poco más grandes en el caso de matrices.

3.2.1 Operaciones elementales con matrices

La suma y resta de matrices se indica como es usual,

```
(%i43)  A+B;
(%o43)  [ 0   3   4 ]
        [ 0   0   3 ]
        [ -1  8   1 ]
(%i44)  A-B;
(%o44)  [ 2    1    2  ]
        [ -2   0    3  ]
        [ 5    -6   -3 ]
```

en cambio el producto de matrices se indica con un punto, ".", como ya vimos con vectores. El operador $*$ multiplica los elementos de la matriz entrada a entrada.

```
(%i45)  A.B;
(%o45)  [ -8   22   7 ]
        [ -8   20   5 ]
        [ 2    -5   0 ]
```

```
(%i46)  A*B;
(%o46)  [ -1  2  3 ]
        [ -1  0  0 ]
        [ -6  7 -2 ]
```

Con las potencias ocurre algo parecido: "^ ^n" eleva toda la matriz a n, esto es, multiplica la matriz consigo misma n veces,

```
(%i47)  A^^2
(%o47)  [  5  5  6 ]
        [  5  1 -6 ]
        [ -1  3 10 ]
```

y "^n" eleva cada entrada de la matriz a n.

```
(%i48)  A^2
(%o48)  [ 1  4  9 ]
        [ 1  0  9 ]
        [ 4  1  1 ]
```

Para el producto de una matriz por un vector sólo tenemos que tener cuidado con utilizar el punto.

```
(%i49)  A.a;
(%o49)  [ 8 ]
        [ 2 ]
        [ 3 ]
```

y no tenemos que preocuparnos de si el vector es un vector "fila" o "columna"

```
(%i50)  a.A
(%o50)  [ 1  3  8 ]
```

El único caso en que $*$ tiene el resultado esperado es el producto de una matriz o un vector por un escalar.

```
(%i51)  2*A;
(%o51)  [  2  4  6 ]
        [ -2  0  6 ]
        [  4  2 -2 ]
```

3.2.2 Otras operaciones usuales

`rank(`*matriz*`)`	rango de la matriz
`col(`*matriz*`,`*i*`)`	columna *i* de la *matriz*
`row(`*matriz*`,`*j*`)`	fila *j* de la *matriz*
`minor(`*matriz*`,`*i*`,`*j*`)`	menor de la matriz obtenido al eliminar la fila *i* y la columna *j*
`submatrix(`*fila1*`,`*fila2*`,..,`*matriz*`,`*col1*`,..)`	matriz obtenida al eliminar las filas y columnas mencionadas
`triangularize(`*matriz*`)`	forma triangular superior de la matriz
`determinant(`*matriz*`)`	determinante
`invert(`*matriz*`)`	matriz inversa
`transpose(`*matriz*`)`	matriz transpuesta
`nullspace(`*matriz*`)`	núcleo de la matriz

transpose Existen órdenes para la mayoría de las operaciones comunes. Podemos calcular la matriz transpuesta con `transpose`,

```
(%i52)  transpose(A);
(%o52)  [ 1  -1   2 ]
        [ 2   0   1 ]
        [ 3   3  -1 ]
```

determinant calcular el determinante,

```
(%i53)  determinant(A);
(%o53)  4
```

invert o, ya que sabemos que el determinante no es cero, su inversa:

```
(%i54)  invert(A);
(%o54)  [ -3/4   5/4   3/2 ]
        [  5/4  -7/4  -3/2 ]
        [ -1/4   3/4   1/2 ]
```

rank Como $\det(A) \neq 0$, la matriz A tiene rango 3. En general, podemos calcular el rango de una matriz cualquiera $n \times m$ con la orden `rank`

```
(%i55)  m:matrix([1,3,0,-1],[3,-1,0,6],[5,-3,1,1])$
(%i56)  rank(m);
```

```
(%o56)  3
```

El rango es fácil de averiguar si escribimos la matriz en forma triangular superior utilizando el método de Gauss con la orden `triangularize` y le echamos un vistazo a la diagonal: **triangularize**

```
(%i57)  triangularize(m);
                                [ 1   3    0    -1  ]
                                [                   ]
                                [ 0  -10   0     9  ]
(%o57)                          [                   ]
                                [ 0   0   -10   102 ]
```

Cualquiera de estos métodos es más rápido que ir menor a menor buscando alguno que no se anule. Por ejemplo, el menor de la matriz A que se obtiene cuando se eliminan la segunda fila y la primera columna es **minor**

```
(%i58)  minor(A,2,1);
                                [ 2   3 ]
(%o58)                          [       ]
                                [ 1   -1 ]
```

Caso de que no fuera suficiente con eliminar una única fila y columna podemos eliminar tantas filas y columnas como queramos con la orden `submatrix`. Esta orden elimina todas las filas que escribamos antes de una matriz y todas las columnas que escribamos después. Por ejemplo, para eliminar la primera y última columnas junto con la segunda fila de la matriz m escribimos: **submatrix**

```
(%i59)  submatrix(2,m,1,4);
                                [ 3   0 ]
(%o59)                          [       ]
                                [ -3  1 ]
```

En el extremo opuesto, si sólo queremos una fila o una columna de la matriz, podemos usar el comando `col` para extraer una columna **col**

```
(%i60)  col(m,2);
                                [  3 ]
                                [    ]
(%o60)                          [ -1 ]
                                [    ]
                                [ -3 ]
```

y el comando `row` para extraer una fila. El resultado de ambas órdenes es una matriz. **row**

```
(%i61)  row(m,1);
(%o61)  [1   3   0   -1]
(%i62)  matrixp(%);
(%o62)  true
```

Para acabar con esta lista de operaciones, conviene mencionar cómo se calcula el núcleo de una matriz. Ya sabes que el núcleo de una matriz $A = (a_{ij})$ de orden $n \times m$ es el subespacio

$$\ker(A) = \{x;\ A.x = 0\}$$

nullspace y es muy útil, por ejemplo, en la resolución de sistemas lineales de ecuaciones. La orden `nullspace` nos da una base del núcleo de la matriz:

```
(%i63)  nullspace(matrix([1,2,4],[-1,0,2]));
```

$$(\%o63)\quad \mathrm{span}\left(\begin{bmatrix} -4 \\ 6 \\ -2 \end{bmatrix}\right)$$

3.2.3 Más sobre escribir matrices

Si has utilizado el menú **Álgebra→Introducir matriz** para escribir matrices ya has visto que tienes atajos para escribir matrices diagonales, simétricas y antisimétricas.

`diagmatrix(n,x)`	matriz diagonal $n \times n$ con x en la diagonal
`entermatrix(m,n)`	definir matriz $m \times n$
`genmatrix`	genera una matriz mediante una regla
`matrix[i,j]`	elemento de la fila i, columna j de la matriz

entermatrix Existen otras formas de dar una matriz en *Maxima*. La primera de ellas tiene más interés si estás utilizando *Maxima* y no *wxMaxima*. Se trata de la orden `entermatrix`. Por ejemplo, para definir una matriz con dos filas y tres columnas, utilizamos `entermatrix(2,3)` y *Maxima* nos va pidiendo que escribamos entrada a entrada de la matriz:

```
(%i64)  c:entermatrix(2,3);
        Row 1 Column 1: 1;
        Row 1 Column 2: 2;
        Row 1 Column 3: 4;
        Row 2 Column 1: -1;
        Row 2 Column 2: 0;
        Row 2 Column 3: 2;
        Matrix entered.
```

$$(\%o64)\quad \begin{bmatrix} 1 & 2 & 4 \\ -1 & 0 & 2 \end{bmatrix}$$

diagmatrix También es fácil de escribir la matriz diagonal que tiene un mismo valor en todas las entradas de la diagonal: sólo hay que indicar el orden y el elemento que ocupa la diagonal. Por ejemplo, la matriz identidad de orden 4 se puede escribir como sigue.

```
(%i65)  diagmatrix(4,1);
```

```
(%o65)  [1 0 0 0]
        [0 1 0 0]
        [0 0 1 0]
        [0 0 0 1]
```

Por último, también podemos escribir una matriz si sabemos una regla que nos diga cuál es el valor de la entrada (i, j) de la matriz. Por ejemplo, para escribir la matriz que tiene como entrada $a_{ij} = i * j$, escribimos en primer lugar dicha regla[7]

```
(%i66)  a[i,j]:=i*j;
(%o66)  a_ij:=ij
```

y luego utilizamos `genmatrix` para construir la matriz (3×3 en este caso): **genmatrix**

```
(%i67)  genmatrix(a,3,3);
(%o67)  [1 2 3]
        [2 4 6]
        [3 6 9]
```

Observa que hemos utilizado corchetes y no paréntesis para definir la regla a_{ij}. Bueno, que ya hemos definido la matriz `a`...un momento, ¿seguro?

```
(%i68)  matrixp(a);
(%o68)  false
```

¿Pero no acabábamos de definirla? En realidad, no. Lo que hemos hecho es definir la regla que permite construir los elementos de la matriz pero no le hemos puesto nombre:

```
(%i69)  c:genmatrix(a,4,5);
(%o69)  [1 2  3  4  5 ]
        [2 4  6  8  10]
        [3 6  9  12 15]
        [4 8  12 16 20]
```

Podemos utilizar la misma notación para referirnos a los elementos de la matriz. Por ejemplo, al elemento de la fila i y la columna j, nos referimos como $c[i, j]$ (de nuevo, observa que estamos utilizando corchetes):

```
(%i70)  c[2,3];
(%o70)  6
```

[7] Si no has borrado el vector `a` que definimos hace algunas páginas, *Maxima* te dará un error.

3.2.4 Valores propios

`charpoly(`*`matriz`*`,`*`variable`*`)`	polinomio característico
`eigenvalues(`*`matriz`*`)`	valores propios de la matriz
`eigenvectors(`*`matriz`*`)`	valores y vectores propios de la matriz

charpoly

Los valores propios de una matriz cuadrada, A, son las raíces del polinomio característico $\det(A - xI)$, siendo I la matriz identidad. La orden `charpoly` nos da dicho polinomio.

```
(%i71) S:matrix([-11/15,-2/15,-4/3],[-17/15,16/15,-1/3],[-
       8/15,4/15,5/3]);
(%o71)  [ -11/15   -2/15   -4/3 ]
        [ -17/15   16/15   -1/3 ]
        [ -8/15    4/15    5/3  ]
(%i72) charpoly(S,x);
(%o72) ((16/15 - x)*(5/3 - x) + 4/45)*(- x - 11/15) + (2*(- 17*(5/3 - x)/15 - 8/45))/15
       - (4*(8*(16/15 - x)/15 - 68/225))/3
(%i73) expand(%);
(%o73) -x^3+2x^2+x-2
```

Por tanto, sus valores propios son

```
(%i74) solve(%,x);
(%o74) [x=2,x=-1,x=1]
```

eigenvalues

Todo este desarrollo nos lo podemos ahorrar: la orden `eigenvalues` nos da los valores propios junto con su multiplicidad.

```
(%i75) eigenvalues(S);
(%o75) [[2,-1,1],[1,1,1]]
```

En otras palabras, los valores propios son 2, -1 y 1 todos con multiplicidad 1. Aunque no lo vamos a utilizar, también se pueden calcular los correspondientes vectores propios con la orden `eigenvectors`:

eigenvectors

```
(%i76) eigenvectors(S);
(%o76) [[[2,-1,1],[1,1,1]],[1,-1/2,-2],[1,4/7,1/7],[1,7,-2]]
```

La respuesta es, en esto caso, cinco listas. Las dos primeras las hemos visto antes: son los valores propios y sus multiplicidades. Las tres siguientes son los tres vectores propios asociados a dichos valores propios.

3.3 Ejercicios

Ejercicio 3.1. Consideremos los vectores $a = (1, 2, -1)$, $b = (0, 2, 3/4)$, $c = (e, 1, 0)$, y $d = (0, 0, 1)$. Realiza las siguientes operaciones

a) $a + b$,
b) $3c + 2b$,
c) $c.d$, y
d) $b.d + 3a.c$.

Ejercicio 3.2. Consideremos las matrices

$$A = \begin{pmatrix} 1 & -2 & 0 \\ 2 & 5 & 3 \\ -3 & 1 & -4 \end{pmatrix} \qquad B = \begin{pmatrix} 0 & -2 & 6 \\ 12 & 2 & 0 \\ -1 & -1 & 3 \end{pmatrix}$$

$$C = \begin{pmatrix} 1 & 2 & 0 & -5 \\ -4 & -2 & 1 & 0 \\ 3 & 2 & -1 & 3 \\ 5 & 4 & -1 & -5 \end{pmatrix} \qquad D = \begin{pmatrix} -1 & 2 & 3 & 0 \\ 12 & -5 & 0 & 3 \\ -6 & 0 & 0 & 1 \end{pmatrix}$$

a) Calcular $A.B$, $A + B$, $D.C$.
b) Extraer la segunda fila de A, la tercera columna de C y el elemento $(3, 3)$ de D.
c) Calcular $\det(A)$, $\det(B)$ y $\det(C)$. Para las matrices cuyo determinante sea no nulo, calcular su inversa. Calcular sus valores propios.
d) Calcular el rango de las matrices A, B, C, D, $D.C$ y $A + B$.
e) Construye una matriz del orden 3×3, de forma que el elemento (i, j) sea $i * j + j - i$. Calcula el determinante, su inversa si la tiene, y su rango. ¿Cuáles son sus valores propios?

Ejercicio 3.3. Calcula el rango de la matriz

$$A = \begin{pmatrix} 2 & 7 & -4 & 3 & 0 & 1 \\ 0 & 0 & 5 & -4 & 1 & 0 \\ 2 & 1 & 0 & -2 & 1 & 3 \\ 0 & 6 & 1 & 1 & 0 & -2 \end{pmatrix}$$

Ejercicio 3.4. Calcula los valores y vectores propios de las siguientes matrices:

$$A = \begin{pmatrix} 0 & 4 \\ 4 & -4 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & 0 & 4 \\ 0 & 3 & 1 \\ 4 & 1 & -4 \end{pmatrix} \text{ y } C = \begin{pmatrix} 0 & 3 & 9 \\ -4 & 8 & 10 \\ 8 & -4 & -2 \end{pmatrix}$$

Ejercicio 3.5.

a) Genera una lista de 10 números aleatorios entre 5 y 25 y reordénala en orden decreciente.

Ejercicio 3.6. Define `listauno:makelist(i,i,2,21)`, `listados:makelist(i,i,22,31)`. Realiza las siguientes operaciones usando algunos de los comandos antes vistos.

a) Multiplica cada elemento de "listauno" por todos los elementos de "listados". El resultado será una lista con 20 elementos (que a su vez serán listas de 10 elementos), a la que llamarás "productos".
b) Calcula la suma de cada una de las listas que forman la lista "productos" (no te equivoques, comprueba el resultado). Obtendrás una lista con 20 números.
c) Calcula el producto de los elementos de la lista obtenida en el apartado anterior.

Ejercicio 3.7. Genera una lista de 30 elementos cuyos elementos sean listas de dos números que no sean valores exactos.

Ejercicio 3.8.

a) Calcula la suma de los números de la forma $\frac{(-1)^{k+1}}{\sqrt{k}}$ desde $k = 1$ hasta $k = 1000$.
b) Calcula el producto de los números de la forma $\left(1 + \frac{1}{k^2}\right)$ desde $k = 1$ hasta $k = 1000$.