Tema 3

Aplicaciones del Maxima en Álgebra

En esta sección aprenderemos a utilizar MAXIMA para operar con vectores y matrices, veremos algunos ejemplos con *matrices de rotaciones* y cálculo de *autovalores* y *autovectores*, y finalmente nos centraremos en el tema de la aplicación de cambios de base sobre vectores y matrices, lo cual es una operación muy habitual en álgebra especialmente cuando se estudia el tema de diagonalización de matrices.

3.1. Operaciones elementales con vectores y matrices

Además de expresiones y funciones escalares, con MAXIMA también podemos manipular fácilmente expresiones que contengan vectores y matrices. Para ello emplearemos *listas* con varios elementos. Para asignar al parámetro a una lista de 6 elementos (p. ej. A, B, C, D, E, F) la sintaxis es la siguiente

```
(%i1) a : [A, B, C, D, E, F];
```

(%o1) $[A, B, C, D, E, F]$

Para extraer elementos de la lista la notación es "a[n]", siendo n el valor del índice cuyo elemento nos interesa (p. ej. `a[1]` nos devuelve `A`). Si luego aplicamos cualquier función u operador sobre a, la función se aplicará sobre todos los elementos, devolviendo una lista con los resultados (por ejemplo calcule a^2, a^{-1} o $\sin a$). Al programar en MAXIMA muchas veces querremos aplicar la misma función a una lista de valores, por ese motivo es útil que las listas en MAXIMA funcionen de este modo. De todas formas conviene tener cuidado, ya que aplicar una función *elemento a elemento* sobre una lista no es una operación que tenga significado como operación vectorial. Por ejemplo, si definimos la lista b como

```
(%i2) b : [q, w, e, r, t, y]$
```

y calculamos `a*b`, obtenemos una lista cuyos elementos son `a[i]*b[i]` con i entre 1 y 6. En este caso, si lo que queremos es calcular el producto escalar de las listas a y b consideradas como vectores, el comando que debemos usar es "`.`":

```
(%i3) a.b;
```

(%o3) $yF + tE + rD + eC + wB + qA$

La definición de matrices es muy simple en MAXIMA, mediante el comando de construcción `matrix([elementos fila 1], [elementos fila 2], ...)`. Por ejemplo, definimos M como la siguiente la matriz 3×3

```
(%i1) M : matrix([3, -1, 0], [2, -3, 1], [4, 4, -2]);
```

```
(%o1)
```
$$\begin{bmatrix} 3 & -1 & 0 \\ 2 & -3 & 1 \\ 4 & 4 & -2 \end{bmatrix}$$

MAXIMA almacena las entradas M_{ij} de la matriz en la forma `M[i, j]`, p. ej. podemos comprobar que `M[2, 3] = 1`.

Por medio de `matrix` podemos definir cualquier matriz a partir de sus *filas*, si la información que tenemos sobre una matriz está dada *por columnas*, lo más sencillo es introducir esta matriz por filas y posteriormente calcular la traspuesta por medio de `transpose`. Por ejemplo, vamos a definir la matriz `MT` $= M^T$

```
(%i2) MT : transpose( matrix([3, -1, 0], [2, -3, 1], [4, 4, -2]) );
```
```
(%o2)
```
$$\begin{bmatrix} 3 & 2 & 4 \\ -1 & -3 & 4 \\ 0 & 1 & -2 \end{bmatrix}$$

El mismo operador que hemos usado antes para calcular el producto escalar (".") es el que se usa para el producto de matrices y para el producto de matrices por vectores (que es un caso particular del producto de matrices). Por ejemplo, para calcular el resultado de aplicar M sobre el vector (x, y, z) hacemos

```
(%i3) M . [x, y, z];
```
```
(%io3)
```
$$\begin{bmatrix} 3\,x - y \\ z - 3\,y + 2\,x \\ -2\,z + 4\,y + 4\,x \end{bmatrix}$$

Por supuesto, para poder aplicar el operador de producto matricial "." es necesario que las matrices o vectores sobre los que actúa tengan las dimensiones adecuadas, de lo contrario nos dará un error.

El MAXIMA ofrece otras formas de introducir matrices. Por ejemplo, también es posible definir la matriz M de forma interactiva con el comando `entermatrix` de MAXIMA, a medida que nos vaya pidiendo los datos de entrada, una vez que queda definida la dimensión de M,

```
(%i4) M : entermatrix(3,3);
```

Para facilitar la entrada de datos, MAXIMA nos pregunta sobre el tipo de matriz que queremos definir; por ejemplo, si nuestra matriz es simétrica, MAXIMA sólo nos preguntará por los elementos por encima y sobre la diagonal, ya que los restantes quedarán fijados por simetría. El enunciado exacto nos pide introducir un número para caracterizar la matriz, 1 si es diagonal, 2 si es simétrica, 3 si es antisimétrica, y 4 si es general, como nuestro ejemplo. Seguidamente, incluimos los datos celda a celda. Existe un tercer método que es útil si los elementos de la matriz siguen una forma funcional de su posición por filas y columnas, esto es, si conocemos una función $f(i, j)$ tal que asigne a cada entrada M_{ij} de la matriz su valor correspondiente. Para poder construir matrices por este método empleamos un tipo especial de función llamado *function array*, que se define igual que las funciones normales pero con los argumentos entre corchetes, en lugar de paréntesis, y posteriormente empleamos el comando `genmatrix`. Por ejemplo, la matriz de Hilbert de dimensión n está dada por los valores

$$H_{ij} = (i + j - 1)^{-1},$$

con i y j entre 1 y la dimensión de la matriz, n, por tanto definimos

```
(%i1) componentesH[i,j] := (i + j - 1)^(-1)$
```

Construyamos ahora la matriz de Hilbert 4×4, utilizamos el comando `genmatrix(` elementos, número de filas, número de columnas`)`. Con este comando MAXIMA aplica

la función `elementos[i, j]` sobre todos los puntos de la matriz de las dimensiones asignadas. En nuestro caso,

```
(%i2) H: genmatrix(componentesH, 4, 4);
```

$$(\%o2)\quad \begin{bmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \end{bmatrix}$$

Una de las características principales de las matrices de Hilbert es que aunque sus elementos son de orden unidad su determinante es sorprendentemente pequeño. Utilizamos el comando `determinant` y confirmamos esta característica en nuestro caso, con seis cifras significativas

```
(%i3) fpprec:6$ determinant(H), bfloat;
(%o3) 1.65344b-7
```

Recordamos que los determinantes sólo están definidos para matrices cuadradas. Otras construcciones que MAXIMA admite de forma directa, son la matriz identidad `ident(`dimensión`)`, la matriz nula `zeromatrix(`número de filas, número de columnas`)`, y cualquier matriz diagonal con todos los elementos iguales `diagmatrix(`dimensión, elemento`)`. Además tiene un comando específico para verificar si una lista de elementos construida es una matriz, `matrixp(`lista`)`, con dos posibles outputs, `true` o `false`.

Retomamos ahora la matriz M definida numéricamente al comienzo de esta sección. Con las matrices se pueden realizar numerosas operaciones en MAXIMA, por ejemplo con el comando `addrow(`matriz, fila nueva`)` podemos añadirle una fila nueva

```
(%i2) M1: addrow(M, [0, 2, 0] );
```

$$(\%o2)\quad \begin{bmatrix} 3 & -1 & 0 \\ 2 & -3 & 1 \\ 4 & 4 & -2 \\ 0 & 2 & 0 \end{bmatrix}$$

y con el comando `addcol(`matriz, columna nueva`)` podemos añadirle una columna nueva

```
(%i2) M2: addcol(M, [-1, 2, 2] );
```

$$(\%o2)\quad \begin{bmatrix} 3 & -1 & 0 & -1 \\ 2 & -3 & 1 & 2 \\ 4 & 4 & -2 & 2 \end{bmatrix}$$

Podemos multiplicar matrices recordando que la regla de multiplicación matricial de dos matrices A y B determina que el número de columnas de A debe ser igual al número de filas de B, esto es $A(n \times k) \cdot B(k \times m) =$ matriz$(n \times m)$. Esto confirma que siempre se pueden multiplicar matrices cuadradas de la misma dimensión. En el caso anterior, entonces, podemos realizar la multiplicación de `M1` por `M2` y también `M1` por `M`, o la potencia $M \cdot M \equiv M^2$, mientras que la multiplicación `M2` por `M` no está definida, y en ese caso el MAXIMA nos daría el mensaje de error:

```
MULTIPLYMATRICES: attempt to multiply nonconformable matrices.
 -- an error. To debug this try: debugmode(true);
```

Para la potencia de matrices aparte del operador de multiplicación matricial (p. ej. $M^2 =$ `M.M`), puede usarse el operador de *potencia matricial*, dado por: "`^^`". Es decir, el operador de potencia matricial es el operador de potencia escrito de manera explícita "dos veces" (recordar que para obtener este símbolo dos veces en la mayoría de los sistemas operativos será preciso pulsar el símbolo [^] del teclado 4 veces, o 2 veces, seguida cada una de un espacio). Por ejemlo veamos M^2:

```
(%i5) M^^2;
```

```
(%o5)
```
$$\begin{bmatrix} 7 & 0 & -1 \\ 4 & 11 & -5 \\ 12 & -24 & 8 \end{bmatrix}$$

Si aplicamos el operador de potencia normal `^` sobre una matriz lo que obtenemos es la matriz original con todos sus elementos elevados a la potencia indicada *elemento a elemento*:

```
(%i6) M^2;
```
```
(%o6)
```
$$\begin{bmatrix} 9 & 1 & 0 \\ 4 & 9 & 1 \\ 16 & 16 & 4 \end{bmatrix}$$

Aunque esta operación puede resultar práctica en algunas ocasiones (para elevar a una potencia cada elemento de una "lista de listas"), está claro que esta operación es distinta a la operación de *elevar una matriz a una potencia*.

Una de las operaciones más habituales con matrices es calcular la *matriz inversa*. En MAXIMA esto se puede hacer o bien elevando la matriz a la potencia -1 o bien por medio del comando `invert`:

```
(%i7) invert(M);
```
```
(%o7)
```
$$\begin{bmatrix} -1 & 1 & \frac{1}{2} \\ -4 & 3 & \frac{3}{2} \\ -10 & 8 & \frac{7}{2} \end{bmatrix}$$

y se puede comprobar que `M^^(-1);` produce el mismo resultado.

Un ejercicio que puede hacerse con las operaciones definidas hasta ahora es comprobar el teorema de los determinantes

$$\det(A \cdot B) = \det(A)\det(B), \qquad \det(A^n) = \det(A)^n.$$

3.1.1. Matrices Ortogonales, Rotaciones

La localización de un punto en el espacio euclídeo tridimensional suele darse en función del vector de posición en coordenadas cartesianas respecto al origen de coordenadas $\boldsymbol{r} = (x, y, z)$. Las operaciones habituales con vectores resultan extremadamente útiles cuando se interpretan de manera geométrica como operaciones sobre vectores de posición de puntos en un espacio. Por ejemplo, si desplazamos el vector $\boldsymbol{r}$ anterior en la dirección de otro vector $\boldsymbol{d} = (d_1, d_2, d_3)$, la posición final de $\boldsymbol{r}$ tras este desplazamiento estará dada por la suma de vectores $\boldsymbol{r} + \boldsymbol{d}$. La operación de *rotación*, de un cierto ángulo respecto a un ciero eje, aplicada sobre un cierto vector de posición, también puede describirse por medio de una operación matemática sencilla. Ya que la rotación es una operación lineal, la transformación $\boldsymbol{r} \to \boldsymbol{r}'$ resultado de aplicar una rotación puede escribirse en forma matricial: $\boldsymbol{r}' = A \cdot \boldsymbol{r}$. Está claro que al aplicar una rotación sobre un vector la longitud del vector $\boldsymbol{r}$ no cambia, lo que exige que la matriz A (con la que describimos esa rotación) sea *ortogonal*, es decir, la matriz transpuesta debe ser igual a la matriz inversa ($A^T = A^{-1}$).

El ejemplo más sencillo de matriz de rotación en 3 dimensiones es la matriz de rotación de ángulo ϕ alrededor del eje z

$$A = \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Aplicada sobre el vector $\boldsymbol{r}$ esta matriz genera la transformación $\boldsymbol{r}' = A \cdot \boldsymbol{r}$, dada por:

$$\begin{aligned} x' &= x\cos\phi + y\sin\phi \\ y' &= -x\sin\phi + y\cos\phi \\ z' &= z \end{aligned}$$

que es el resultado de rotar el vector $\boldsymbol{r}$ un ángulo ϕ respecto al eje z. Por ejemplo, si aplicamos esta matriz sobre el vector unitario $\boldsymbol{i} = (1,0,0)$ obtenemos[1] el vector $(\cos\phi, -\sin\phi, 0)$ si lo aplicamos sobre el vector unitario $\boldsymbol{j} = (0,1,0)$ obtenemos $(\sin\phi, \cos\phi, 0)$ y si lo aplicamos sobre $\boldsymbol{k} = (0,0,1)$ vemos que este vector permanece invariante, como cabía esperar.

La matriz de rotación A tiene varias propiedades interesantes. En primer lugar podemos comprobar que es ortogonal (su matriz traspuesta es también su inversa), una vez comprobado esto es inmediato darse cuenta de que la matriz inversa de A es igual a la matriz A cambiando ϕ por $-\phi$. Evidentemente, la operación inversa a aplicar una rotación de ángulo ϕ es aplicar una rotación de ángulo $-\phi$. Por último es inmediato observar que la matriz A se reduce a la matriz identidad cuando el ángulo de rotación ϕ es nulo, lógicamente. Estas propiedades son comunes a todas las matrices de rotación.

Al operar con matrices de rotación hay un detalle importante que conviene aclarar. Dada una matriz de rotación se pueden hacer dos cosas diferentes: Por un lado podemos aplicar dicha matriz de rotación *sobre los vectores de la base*, generando de esta manera un *cambio de base*, o podemos aplicar esta matriz de rotación sobre los vectores del espacio, sin cambiar la base que teníamos. Esto último es lo que hemos hecho más arriba al aplicar la matriz A sobre el vector $\boldsymbol{r}$.

- En el primer caso, si generamos un cambio de base los vectores del espacio no cambian, pero sus componentes en la nueva base son diferentes a las componentes que tenían respecto de la base antigua. Para ver cuáles serían las componentes de los vectores respecto de la base nueva situémonos en el punto de vista de la base. Al aplicar el cambio de base esta gira, a medida que la base gira un cierto ángulo α (respecto a un cierto eje) desde el punto de referencia de la base veremos que todos los vectores del espacio rotan un ángulo $-\alpha$ repecto del mismo eje. Por tanto, para calcular las componentes de estos vectores en la nueva base lo que tenemos que hacer es aplicar sobre dichos vectores la matriz de rotación de ángulo $-\alpha$ respecto al eje de rotación dado, es decir, la matriz inversa (o, equivalentemente, la transpuesta) de la matriz que hemos empleado para generar el cambio de base.

- En la segunda posibilidad todo es mucho más sencillo. La base del espacio no cambia, y sencillamente aplicamos la matriz de rotación sobre los vectores del espacio, obteniendo al hacerlo las correspondientes coordenadas de dichos vectores, rotados un ángulo α respecto al eje de rotación que sea, y referidos a la misma base que teníamos al principio.

Aunque hemos ilustrado el tema de las matrices de rotación con un caso en el espacio tridimensional habitual, la discusión anterior es válida independientemente de la dimensionalidad del espacio en que estemos operando, 2D, 3D, . . . , nD.

[1] Maxima es suficientemente listo como para entender que, si se aplica la matriz por la izquierda al vector, es porque éste se considera un vector columna a efectos de la operación de multiplicación; en sentido estricto, esta operación sería incorrecta, pero Maxima traspone automáticamente el vector.

Pasamos ahora a comprobar las propiedades de la transformación A por medio de MAXIMA. En primer lugar dado que la matriz de rotación depende del ángulo de rotación ϕ, y éste puede tomar valores arbitrarios, vamos a definir la matriz $A(\phi)$ como una función de ϕ

```
(%i1) A(phi) := matrix(
            [cos(phi), sin(phi), 0],
            [-sin(phi), cos(phi), 0],
            [0, 0, 1] );
```

$$(\%o1)\quad A(\phi) := \begin{pmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Comprobamos ahora que esta matriz es ortogonal. Para ello, debemos evaluar su inversa y su transpuesta, que en MAXIMA corresponden a los comandos `invert` y `transpose`, con el resultado esperado de ortogonalidad

```
(%i3) invert(A(phi)); transpose(A(phi));
```

El cálculo de la matriz inversa requiere del cálculo del determinante de la matriz, que en este caso es $\det(A) = \sin^2(\phi) + \cos^2(\phi)$. MAXIMA no simplifica este valor a la unidad a no ser que se lo pidamos explícitamente, p. ej. con el comando `trigsimp`. También podemos comprobar que la longitud de un vector arbitrario se mantiene invariante bajo esta rotación. Aplicando la matriz de rotación sobre un vector (x, y, z) obtenemos el vector primado `vectorP`

```
(%i5) vectorP : A . [x, y, z];
```

$$(\%o5)\quad \begin{bmatrix} \sin\phi\, y + \cos\phi\, x \\ \cos\phi\, y - \sin\phi\, x \\ z \end{bmatrix}$$

y por medio de `trigsimp(vectorP . vectorP);` podemos comprobar que el módulo al cuadrado de $\boldsymbol{r}'$ coincide con el de $\boldsymbol{r}$.

3.1.1.1. Ángulos de Euler

En general dado un sistema de referencia arbitrario en 3 dimensiones la orientación de cualquier otro sistema de referencia, con origen en el mismo punto, se puede obtener a partir de la del primero aplicando 3 rotaciones consecutivas, con ángulos de rotación α, β y γ, conocidos como los *ángulos de Euler*, de la siguiente forma: En primer lugar debemos girar un ángulo α en torno al eje z, posteriormente un ángulo β en torno al nuevo eje y (dado por la posición del eje y tras la primera rotación), y finalmente un ángulo γ en torno al nuevo eje z (dado por la posición del eje z tras la segunda rotación). Matemáticamente la matriz de rotación que describe esta operación será el producto de estas tres rotaciones

$$R(\alpha, \beta, \gamma) = A_z(\gamma) A_y(\beta) A_z(\alpha)$$

tomadas en el sentido descrito anteriormente. Las matrices de la rotación según el eje z son las mismas que en el caso anterior, y la matriz de rotación según el eje y es

$$A_y = \begin{pmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{pmatrix}$$

Para determinar la expresión general de la matriz de rotación en MAXIMA, definimos A_y en la forma

```
(%i8) Ay(phi) := matrix(
           [cos(phi), 0, -sin(phi)],
           [0, 1, 0],
           [sin(phi), 0, cos(phi)])$
```

para posteriormente evaluar el producto de rotaciones

```
(%i9) R(alpha, beta, gamma) = A(gamma) . Ay(beta) . A(alpha);
```

(%o9) $R(\alpha,\beta,\gamma) =$

$$\begin{pmatrix} \cos\alpha\cos\beta\cos\gamma-\sin\alpha\sin\gamma & \cos\alpha\sin\gamma+\sin\alpha\cos\beta\cos\gamma & -\sin\beta\cos\gamma \\ -\cos\alpha\cos\beta\sin\gamma-\sin\alpha\cos\gamma & \cos\alpha\cos\gamma-\sin\alpha\cos\beta\sin\gamma & \sin\beta\sin\gamma \\ \cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta \end{pmatrix}$$

En la anterior línea sólo hemos indicado la forma final de la matriz $R(\alpha,\beta,\gamma)$, realizando el producto matricial $A_z(\gamma)A_y(\beta)A_z(\alpha)$, pero no hemos introducido una definición. Con la definición de una función por medio de `:=`

```
(%i10) R(alpha, beta, gamma) := A(gamma) . Ay(beta) . A(alpha)$;
```

MAXIMA responde indicando que $R(\alpha,\beta,\gamma)$ está dado por el producto de las tres matrices, que queda sin evaluar hasta que realicemos una llamada a la función R. Como punto final de este apartado, queremos verificar algunas propiedades de la matriz de rotación basada en los ángulos de Euler, en primer lugar es inmediato confirmar de nuevo que esta matriz es ortogonal y que mantiene invariante la norma del vector $\boldsymbol{r}$. Aparte de esto es interesante verificar las dos propiedades siguientes:

a. La matriz es invariante bajo la transformación $\alpha \to \alpha+\pi$, $\beta \to -\beta$, $\gamma \to \gamma-\pi$. Para verificar esto evaluamos

```
(%i11) subst(alpha +%pi, alpha, R(alpha, beta, gamma))$

       subst(-beta, beta,%)$

       RP: subst(gamma -%pi, gamma,%)$

(%i12) R(alpha, beta, gamma) - RP;
```

(%o12) $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

b. La inversa de $R(\alpha,\beta,\gamma)$ se obtiene deshaciendo las rotaciones efectuadas, es decir rotando según los ángulos contrarios en el orden adecuado:

$$R^{-1}(\alpha,\beta,\gamma) = R(-\gamma,-\beta,-\alpha)$$

Para verificar esto evaluamos

```
(%i13) R(-gamma, -beta, -alpha) - trigsimp(invert(R(alpha, beta, gamma)));
```

(%o13) $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

3.1.2. Autovalores y Autovectores

El cálculo de autovalores y autovectores es una de las tareas más frecuentes en la actividad de un físico, sea cual sea el campo en el que trabaje. Todos los paquetes de cálculo simbólico incorporan herramientas para este tipo de operaciones, cuya eficacia está limitada siempre por la dimensión de la matriz con la que trabajemos, con matrices más o menos pequeñas esto siempre es muy fácil de hacer, pero con matrices muy grandes (p. ej. $10^6 \times 10^6$) esto se convierte en algo realmente difícil.

Tomemos como un primer ejemplo el cálculo de autovalores y autovectores de una matriz simétrica. Según la teoría de matrices todos sus autovalores deben ser reales y los autovectores mutuamente perpendiculares. Definimos la matriz bajo estudio

```
(%i1) A : matrix([0, 1, 0], [1, 0, 0], [0, 0, 0])$
```

Los autovalores son las raíces de la ecuación característica

$$\det(A - \lambda I) = 0$$

En MAXIMA, la resolución de ecuaciones polinómicas se efectúa mediante el comando `solve(`ecuación, incógnita`)`. Definimos primero la matriz extendida y su determinante

```
(%i2) matrizD : A - lambda * ident(3); D : determinant(matrizD);
```

$$\text{(\%o2)} \quad \begin{pmatrix} -\lambda & 1 & 0 \\ 1 & -\lambda & 0 \\ 0 & 0 & -\lambda \end{pmatrix}$$

$$\text{(\%o3)} \quad \lambda - \lambda^3$$

El polinomio en λ que hemos obtenido (en este caso $p(\lambda) = \lambda - \lambda^3$) se denomina polinomio característico de la matriz A, y la ecuación característica $p(\lambda) = 0$ determina los autovalores. Lo resolvemos con Maxima

```
(%i4) solve(D = 0, lambda);
(%o4) [lambda = -1, lambda = 1,lambda = 0]
```

Comprobamos que todos son reales, y al ser distintos, decimos que no existe degeneración[2]. Para calcular los autovectores recordamos que para cada autovalor λ, el autovector correspondiente debe satisfacer

$$(A - \lambda I) \cdot \boldsymbol{v}_\lambda = 0$$

es decir, $A \cdot \boldsymbol{v}_\lambda = \lambda \boldsymbol{v}_\lambda$. Aplicamos la matriz A sobre un vector arbitrario (x, y, z)

```
(%i5) condicion: matrizD . [x, y, z]$
```

e imponemos que se cumpla para cada uno de los 3 autovalores que hemos encontrado:

```
(%i6) condicion1 : subst(-1, lambda, condicion)$
(%i7) condicion2 : subst(+1, lambda, condicion)$
(%i8) condicion3 : subst( 0, lambda, condicion)$
```

Resolviendo cada una de estas condiciones obtenemos los autovectores correspondientes a cada uno de los autovalores

```
(%i9) solve([condicion1[1, 1] = 0, condicion1[2, 1] = 0, condicion1[3, 1]
= 0], [x, y, z]);
```

[2] Decimos que existe degeneración de los autovalores cuando podemos tener dos autovectores linealmente independientes correspondientes al mismo autovalor. En Física habitualmente asociamos cada autovector con un estado físico diferente, por lo que un autovalor es degenerado cuando no sabemos diferenciar entre dos estados basándonos sólo en el valor numérico de ese autovalor repetido.

```
 solve: dependent equations eliminated: (1)
(%o9) [[x=%r1,y=-%r1,z=0]]
```

Este resultado nos indica que el autovector del autovalor $\lambda = -1$ es de la forma $\boldsymbol{v} = c\,(1,-1,0)$, siendo c un factor arbitrario. Tomando $c = 1/\sqrt{2}$ obtenemos el autovector normalizado

$$\boldsymbol{v}_{-1} = (1/\sqrt{2}, -1/\sqrt{2}, 0)$$

En este caso, de las tres incógnitas que aparecen en el sistema de ecuaciones que hemos enviado al MAXIMA sólo se ha podido determinar de manera unívoca una de ellas ($z = 0$), las otras dos quedan determinadas en función de un parámetro al que el MAXIMA asigna el nombre `%r1`. Es decir, el subespacio propio del autovalor $\lambda = -1$ es la recta con vector director $\boldsymbol{v}_{-1} = (1/\sqrt{2}, -1/\sqrt{2}, 0)$.

Para el siguiente autovalor encontramos

```
(%i10) solve([condicion2[1, 1] = 0, condicion2[2, 1] = 0, condicion2[3,
1] = 0], [x, y, z]);
solve: dependent equations eliminated: (1)
(%o10) [[x=%r2,y=%r2,z=0]]
```

de donde deducimos que el autovector normalizado correspondiente al autovalor $+1$ es

$$\boldsymbol{v}_{+1} = (1/\sqrt{2}, 1/\sqrt{2}, 0)$$

por tanto el subespacio para $\lambda = 1$ es la recta con vector director $\boldsymbol{v}_{+1} = (1/\sqrt{2}, 1/\sqrt{2}, 0)$.

Finalmente para el tercer autovalor encontramos

```
(%i11) solve([condicion3[1, 1] = 0, condicion3[2, 1] = 0, condicion3[3,
1] = 0], [x, y, z]);
solve: dependent equations eliminated: (3)
(%o11) [[x=0,y=0,z=%r3]]
```

de modo que

$$\boldsymbol{v}_0 = (0, 0, 1)$$

Es decir, el subespacio propio de $\lambda = 0$ es la recta con vector director $\boldsymbol{v}_0 = (0, 0, 1)$, y observamos que estos tres subespacios propios son mutuamente ortogonales, tal y como esperábamos de una matriz real y simétrica con autovalores distintos.

Hasta ahora hemos hecho esto paso a paso, para ver cómo funciona el comando de resolver sistemas de ecuaciones algebraicas `solve`. El cálculo de autovalores y autovectores es tan habitual que todos los sistemas de álgebra computacional incluyen comandos específicos para estas tareas. El comando del MAXIMA que proporciona los autovalores de una matriz es `eigenvalues`, y para los autovectores, `eigenvectors`.

El comando `eigenvalues(A)`

```
(%i12) eigenvalues(A);
(%o12) [[-1, 1, 0], [1, 1, 1]]
```

da un listado de los autovalores (primera lista del output retornado por `eigenvalues`), y sus correspondientes multiplicidades (segunda lista del output). En este caso son todos reales y simples. Para los autovectores el comando `eigenvectors(A)`

```
(%i13) eigenvectors(A);
(%o13) [[[-1, 1, 0], [1, 1, 1]], [[[-1, 1, 0]], [[1, 1, 0]], [[0, 0, 1]]]]
```

nos vuelve a dar en primer lugar los autovalores y sus multiplicidades y en segundo lugar los tres autovectores correspondientes (no normalizados).

Una de las aplicaciones más importantes del cálculo de autovalores y autovectores es la diagonalización de matrices. En el caso anterior, dado que los tres autovectores

son linealmente independientes, la matriz A admite una representación diagonal en la forma

$$A' = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Con ayuda de los autovectores es fácil determinar qué matriz P define este cambio de base, en la forma

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

La matriz P está dada por

$$P = (\boldsymbol{v}_{-1}, \boldsymbol{v}_1, \boldsymbol{v}_0)$$

es decir, está formada por los autovectores como columnas (siendo irrelevante la normalización de los autovectores, tal y como puede comprobarse fácilmente). Con el MAXIMA, el cálculo quedaría como sigue. En primer lugar extraemos del cálculo de autovectores, los tres vectores fila independientes

```
(%i14) listado : eigenvectors(A)$
         vectorf1 : listado[2][1][1]$
         vectorf2 : listado[2][2][1]$
         vectorf3 : listado[2][3][1]$
```

ya que, por ejemplo, la localización del vector $\boldsymbol{v}_{-1}$ en la variable listado es la siguiente: se sitúa en el segundo grupo de datos, donde se listan los autovectores, dentro de ese grupo, se encuentra en el primer grupo (ya que $\lambda = -1$ es el primer autovalor) y está en primer lugar (ya que sólo hay un vector en ese subespacio). Lo mismo sucede con los restantes vectores (ya que en este caso todas las multiplicidades eran 1). Hecho esto, podemos definir la matriz P como una matriz de una sola columna, igual al vector columna $\boldsymbol{v}_{-1}$ y posteriormente añadir los otros dos autovectores como dos columnas adicionales

```
(%i15) matrizP : transpose(vectorf1);
```

$$\text{(\%o15)} \quad \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

```
(%i16) matrizP : addcol(matrizP, transpose(vectorf2))$
       matrizP : addcol(matrizP, transpose(vectorf3));
```

$$\text{(\%o16)} \quad \begin{bmatrix} 1 & 1 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

y confirmamos la transformación de semejanza a la forma diagonal

```
(%i15) invert(matrizP) . A . matrizP;
```

$$\text{(\%o15)} \quad \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Otro cálculo interesante es verificar que el espacio generado por los tres autovectores linealmente independientes se corresponde con todo el espacio euclídeo de tres dimensiones; para ello basta demostrar, y se deja como ejercicio, que la matriz I definida como la expansión matricial en los tres autovectores normalizados

$$I = \boldsymbol{v}_{-1}\boldsymbol{v}_{-1}^T + \boldsymbol{v}_1\boldsymbol{v}_1^T + \boldsymbol{v}_0\boldsymbol{v}_0^T$$

es la matriz identidad 3×3. Por tanto, para cualquier vector $\boldsymbol{r}$ del espacio euclídeo, tenemos

$$\boldsymbol{r} = I \cdot \boldsymbol{r} = \boldsymbol{v}_{-1}\boldsymbol{v}_{-1}^T \cdot \boldsymbol{r} + \boldsymbol{v}_1\boldsymbol{v}_1^T \cdot \boldsymbol{r} + \boldsymbol{v}_0\boldsymbol{v}_0^T \cdot \boldsymbol{r}$$

que podemos escribir como el desarrollo del vector $\boldsymbol{r}$ en la base formada por los tres autovectores normalizados

$$\boldsymbol{r} = x_{-1}\boldsymbol{v}_{-1} + x_1\boldsymbol{v}_1 + x_0\boldsymbol{v}_0$$

donde $x_i = \boldsymbol{v}_i^T \cdot \boldsymbol{r}$ es el producto escalar, en notación matricial.

Con estos ejemplos hemos visto todo lo necesario para empezar a manejar el MAXIMA por medio del *front end* WXMAXIMA. La mejor forma de aprender cualquier lenguaje de programación es usándolo para resolver problemas concretos, de modo que en lo que sigue indicaremos algunos problemas especialmente representativos que pueden resolverse con el MAXIMA. Aparte del material indicado en la bibliografía básica de la asignatura, como ayuda para realizar estos ejercicios dispone del menú de ayuda del WXMAXIMA y del comando `describe`, que le mostrará en pantalla un resumen de la sintaxis y funcionamiento de cualquiera de los comandos del MAXIMA.

3.2. Cambios de Base

3.2.1. Aplicación de cambios de base sobre vectores

En un espacio vectorial un vector queda descrito por sus componentes respecto de una determinada base del espacio vectorial, p. ej. el vector $\boldsymbol{v} = (1, 2, 3)$ es el que resulta de desplazarse una unidad en la dirección x, 2 en y y 3 en z. En el ejemplo anterior hemos empleado la *base canónica*, que es la formada por los vectores

$$\left.\begin{array}{l} \boldsymbol{e}_1 = (1, 0) \\ \boldsymbol{e}_2 = (0, 1) \end{array}\right\} \text{en 2 dimensiones}$$

$$\left.\begin{array}{l} \boldsymbol{e}_1 = (1, 0, 0) \\ \boldsymbol{e}_2 = (0, 1, 0) \\ \boldsymbol{e}_3 = (0, 0, 1) \end{array}\right\} \text{en 3 dimensiones}$$

$$\left.\begin{array}{l} \boldsymbol{e}_1 = (1, 0, 0, \ldots, 0) \\ \boldsymbol{e}_2 = (0, 1, 0, \ldots, 0) \\ \ldots \quad \ldots \quad \ldots \quad \ldots \\ \boldsymbol{e}_n = (0, 0, \ldots, 0, 1) \end{array}\right\} \text{en } n \text{ dimensiones}$$

Aparte de la base canónica cualquier conjunto de n vectores *linealmente independientes* forma una base válida del espacio. Aparece entonces la cuestión:

- ⋆ **dadas las componentes de un vector (que existe independientemente de la base) respecto de una base cualquiera ¿cómo se calculan las componentes de este vector respecto de otra base distinta?**

Como el objetivo de esta asignatura es sólamente aprender a programar, damos a continuación las instrucciones detalladas sobre cómo se hacen cambios de base para el caso de dimensión n arbitraria, sin entrar en demostraciones. De todas formas, para resolver los ejercicios recomendamos comenzar con cosas más sencillas, resolviendo primero casos con $n = 2$, o $n = 3$ y, cuando eso ya esté superado, generalizándolos a dimensión arbitraria.

1. Cualquier vector $\boldsymbol{v}$ se expresa en términos de la base de partida como

$$\boldsymbol{v} = v_1\boldsymbol{e}_1 + v_2\boldsymbol{e}_2 + \cdots + v_n\boldsymbol{e}_n = \sum_{i=1}^{n} v_i\boldsymbol{e}_i = v_i\boldsymbol{e}_i$$

donde los números v_i ($i = 1, \ldots, n$) son las coordenadas de $\boldsymbol{v}$ respecto de la base formada por los $\boldsymbol{e}_i$. En cálculo con matrices y vectores se usan constantemente expresiones como la anterior, en la que se hace una suma respecto de un índice (p. ej. i en la ecuación de arriba), para simplificar estas expresiones se suele usar el llamado:

⋆ **Convenio de suma de Einstein:**
Cuando en una expresión aparece un índice repetido la expresión representa la suma respecto de ese índice para todos los valores posibles del índice, p.ej.:

- Desarrollo en componentes de un vector

$$\boldsymbol{v} = \sum_{i=1}^{n} v_i\boldsymbol{e}_i = v_i\boldsymbol{e}_i$$

- Producto escalar

$$\boldsymbol{u}\cdot\boldsymbol{v} = \sum_{i=1}^{n} u_i v_i = u_i v_i$$

- Producto de dos matrices

$$(A\cdot B)_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj} = A_{ik}B_{kj}$$

- Producto de muchas matrices

$$(A\cdot B\cdot C\cdot D)_{ij} = \sum_{\alpha=1}^{n}\sum_{\beta=1}^{n}\sum_{\gamma=1}^{n} A_{i\alpha}B_{\alpha\beta}C_{\beta\gamma}D_{\gamma j} = A_{i\alpha}B_{\alpha\beta}C_{\beta\gamma}D_{\gamma j}$$

- Traza de una matriz

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{ii} = A_{ii}$$

- El índice que aparece repetido se llama *índice mudo* (los demás índices que aparezcan en la expresión se llaman *índices libres*). Como la expresión con un índice mudo realmente representa la suma para todos los valores posibles del índice mudo, está claro que el resultado será el mismo independientemente de la letra con que designemos al índice mudo (y que no debe coincidir con la de los índices libres). ej. $\boldsymbol{u}\cdot\boldsymbol{v} = u_i v_i = u_j v_j = u_\alpha v_\alpha = \ldots$.
- El convenio de suma de Einstein se usa con muchísima frecuencia para escribir expresiones de este tipo de una manera rápida, sin sumatorios. En lo sucesivo suponemos que se aplica este convenio a menos que se diga lo contrario.

2. La base "nueva" está formada por los vectores $\boldsymbol{t}_1$, $\boldsymbol{t}_2$, $\boldsymbol{t}_3$, $\ldots$, $\boldsymbol{t}_n$, cuya expresión en términos de la base antigua suponemos conocida.

3. La matriz del cambio de base (C) es la matriz cuadrada formada por los *vectores columna* $\boldsymbol{t}_i$ (con $i = 1, \ldots, n$) expresados en términos de la base antigua

$$C = \begin{pmatrix} \boldsymbol{t}_1 & \boldsymbol{t}_2 & \boldsymbol{t}_3 & \ldots & \boldsymbol{t}_n \end{pmatrix}$$

   Es muy fácil comprobar que cada uno de los vectores $\boldsymbol{t}_i$ se obtiene al aplicar la matriz del cambio sobre la base de partida en la forma

$$\boldsymbol{t}_j = \boldsymbol{e}_i C_{ij}, \qquad i, j = 1, \ldots, n$$

   por eso se define de esta forma la matriz del cambio C.

4. Si los vectores $\boldsymbol{t}_i$ son linealmente independientes entonces C es invertible, de donde deducimos

$$\boldsymbol{e}_j = \boldsymbol{t}_i (C^{-1})_{ij}, \qquad i, j = 1, \ldots, n$$

   donde C^{-1} es la inversa de la matriz C (si los $\boldsymbol{t}_i$ no son linealmente independientes entonces no forman una base).

5. Sustituyendo esta última relación en

$$\boldsymbol{v} = v_i \boldsymbol{e}_i = \hat{v}_i \boldsymbol{t}_i$$

   encontramos finalmente las relaciones

$$\boxed{v_i = C_{ij} \hat{v}_j} \qquad \boxed{\hat{v}_i = (C^{-1})_{ij} v_j}$$

   que indican cómo se relacionan las coordenadas de un vector expresadas en dos bases distintas.

3.2.2. Aplicación de cambios de base sobre aplicaciones lineales

De forma similar a como sucedía con los vectores, las aplicaciones lineales definidas sobre un espacio vectorial también se describen por medio de sus componentes respecto de una base del espacio. En el caso de las aplicaciones lineales estas componentes forman una matriz, cuyas componentes son las coordenandas de los vectores que resultan de aplicar la aplicación lineal sobre cada uno de los vectores de la base del espacio vectorial.

⋆ **¿Cómo se calculan las componentes de una matriz respecto de la nueva base?**

A partir de las relaciones anteriores es muy sencillo deducir la regla de transformación de las aplicaciones lineales.

1. Supongamos una aplicación lineal A, tal que aplicada sobre el vector $\boldsymbol{u}$ nos da el vector $\boldsymbol{v}$

$$A\boldsymbol{u} = \boldsymbol{v}$$

2. Si escribimos esto en componentes respecto de la base de partida tendremos

$$A_{ij}u_j = v_i$$

donde A_{ij} es la matriz de la aplicación lineal A en la base de partida. Suponemos que la matriz A_{ij} es un dato que conocemos. Queremos calcular la forma de esta matriz en la base nueva.

3. La matriz de A en la base nueva (vamos a denotarla por $\hat{A}_{ij}$) cumplirá una expresión análoga a la anterior pero con los vectores $\boldsymbol{u}$ y $\boldsymbol{v}$ referidos a la base nueva, es decir

$$\hat{A}_{ij}\hat{u}_j = \hat{v}_i$$

4. Sustituyendo en esta ecuación la regla de transformación de los vectores deducimos directamente

$$\boxed{A_{ij} = C_{ik}\hat{A}_{kl}(C^{-1})_{lj}} \qquad \boxed{\hat{A}_{ij} = (C^{-1})_{ik}A_{kl}C_{lj}}$$

Estas reglas indican cómo se transforman las componentes de una matriz al aplicar un cambio de base. Esta es toda la teoría que tenemos que aplicar para hacer los ejercicios del tema 3.

3.2.3. Funciones del Maxima que debemos aplicar

Las siguientes funciones del WXMAXIMA son algunas de las que tendremos que emplear para hacer los ejercicios de este tema:

- Asignación del valor `B` a la variable `A` `A : B`
- Definición de una función `A` que depende de `x` `A(x) := ...`
- Para crear una matriz `matrix`
- Para invertir matrices `invert`
- Producto de matrices `.`
- Cálculo de los autovectores de una matriz `eigenvectors`
- **Muy importante:** Siempre que estemos trabajando con WXMAXIMA, cuando se definen funciones más o menos extensas es una buena idea emplear el comando `block`, que nos permite definir variables *locales* dentro de esa función. Por ejemplo, si calculamos una función que calcule integrales como

$$y = \int_a^b f(x)\,dx$$

una buena forma de hacerlo es por medio de la función `block`, definiendo la variable x como una variable local.

Hacer las cosas de esta manera tiene muchísimas ventajas, en particular evita muchos posibles errores, mejora el uso de la memoria y hace que los programas sean más claros.

Para ver cómo funcionan estos comandos se puede consultar la guía del WXMAXIMA.

3.3. Problemas

3.3.1. Problemas propuestos

1. Escriba una función que aplique cambios de base sobre vectores.
   - input: coordenadas del vector en la base estándar, vectores que forman la nueva base.
   - output: coordenadas del vector en la nueva base.
2. Escriba un programa que aplique cambios de base sobre matrices.
   - input: coordenadas de la matriz correspondiente a una aplicación lineal en la base estándar, vectores que forman la nueva base.
   - output: coordenadas de la matriz correspondiente a la aplicación lineal en la nueva base.
3. Invierta las anteriores relaciones para escribir una función que genere las componentes de vectores y matrices respecto de la base estándar a partir de sus componentes respecto a una base arbitraria.
   - input: coordenadas del vector o de la matriz respecto de una base arbitraria, vectores que forman la base arbitraria respecto de la base estándar.
   - output: coordenadas del vector o de la matriz en la base estándar.

3.3.2. Problemas resueltos

1. Dada una aplicación lineal calcule sus autovalores y autovectores. Aplique las funciones que ha definido en el apartado anterior para calcular la matriz correspondiente a esa aplicación lineal en la base formada por sus autovectores. ¿Se encuentra el resultado que se esperaba?
2. A partir de la función para cambios de base realice un programa que calcule el producto escalar de dos vectores expresados por medio de sus componentes respecto de una base cualquiera.

Soluciones

Realmente más que *las* soluciones lo que vamos a exponer en estas notas son *unas* posibles soluciones. En programación suele haber muchas formas posibles de hacer la misma cosa y dependiendo de qué es lo que se quiera optimizar resultará más conveniente una u otra. Lo más habitual es que los programas de cálculo estén optimizados para *velocidad*, para que den la respuesta empleando el mínimo tiempo de cálculo posible, o para *memoria*, para que puedan ejecutarse empleando el mínimo espacio de memoria posible. De todas formas hay otras posibilidades, por ejemplo también es habitual optimizar un programa para velocidad pero teniendo en cuenta no sólo el tiempo de cálculo, sino también el tiempo que nos lleva escribir y depurar el código, lo cual nos llevará a escribir programas que optimizan la *sencillez* del código.

Este último es un poco el espíritu de los programas que incluimos a continuación, que desde luego no están optimizados ni para velocidad ni para uso de memoria.

Para aprender a programar no hay nada mejor que resolver ejercicios concretos y *copiar* (es decir, tomar como punto de partida) ejemplos ya hechos. A continuación incluimos sin más el código para los ejercicios del tema 3.

Nota: El procesador de textos empleado para escribir este documento es LaTeX, para incluir el código en MAXIMA que figura a continuación se ha empleado el paquete `listings`, la instrucción en LaTeX para incluir el contenido del archivo `UD-1_listings/c-02/cambio_base_vector.mc` es:

```
\lstinputlisting[language=Maxima]{UD-1_listings/c-02/cambio_base_vector.mc}
```

Ejercicio 1

Dada una aplicación lineal calcule sus autovalores y autovectores. Aplique las funciones que ha definido en el apartado anterior para calcular la matriz correspondiente a esa aplicación lineal en la base formada por sus autovectores. ¿Se encuentra el resultado que se esperaba?

- En primer lugar vamos a definir una función (`cambiobasevector`) que aplica cambios de base sobre vectores. A continuación incluimos el contenido del archivo `cambio_base_vector.mc` que contiene dicha función :

```
/* FUNCION "cambiobasevector".
   Aplica cambios de base sobre vectores, dimensiones arbitrarias.
   Input:
     1: "basenueva" = lista de vectores que forman la nueva base
     2: "vector" = vector sobre el que vamos a aplicar el cambio de
        base
   Output:
     componentes del vector en la nueva base */

cambiobasevector(basenueva, vector) := block( [A],

  /* construimos la matriz del cambio de base y la guardamos en la
     variable local A */

  A : matrix(basenueva[1]),

  for i : 2 thru length(basenueva) step 1 do A : addrow(A, basenueva[
     i]),

  A : transpose(A),

  /* calculamos la inversa de la matriz del cambio de base */
```

```
  A : invert(A),

  /* aplicamos esta matriz sobre el vector */

  A . vector

)$

/* fin */
```

Como puede verse, en este listado hemos omitido la instrucción `return` al final del conjunto de instrucciones que forman el contenido del comando `block`. Cuando en una función definida por medio de `block` se omite la instrucción `return`, la función retorna el output de la última instrucción contenida en el `block`. En general es una buena idea incluir la instrucción `return` al final, pero en funciones sencillas como esta puede omitirse.

Para ver el funcionamiento de esta función se puede evaluar la instrucción `cambiobasevector([[0, 1, 0], [1, 0, 0],[0, 0, 1]], [x, y, z]);`, que devuelve las coordenadas de un vector genérico (x, y, z) en términos de la base $\{\boldsymbol{j}, \boldsymbol{i}, \boldsymbol{k}\}$, es decir: (y, x, z).

En este ejemplo también vemos una característica interesante de los programas de *alto nivel* como el Maxima: como puede verse la función que hemos definido funciona para un número de dimensiones arbitrario y en ninguna parte hemos tenido que decirle explícitamente cuál es la dimensión del espacio vectorial en el que estamos trabajado, sino que el propio programa *deduce* este valor del número de elementos que forman la base del espacio vectorial, que es una de las variables que pasamos como *input*.

Los programas de cálculo de *alto nivel* son muy fáciles de usar porque manejan objetos matemáticos similares a los conceptos que manejamos nosotros, por ejemplo, este programa admite como input una lista de vectores base y un vector, independientemente de las dimensiones que tengan. Por supuesto, aunque la función admite como input cualquier lista de vectores que le pasemos, el programa sólo funcionará bien cuando los datos que le demos sean coherentes, es decir, si la lista de vectores base tiene n vectores todos ellos deberán tener n componentes y deberán ser linealmente independientes, y el vector al que le aplicamos el cambio también deberá ser de n componentes, de lo contrario el programa dará un error.

Los programas de bajo nivel (como el C) son mucho más potentes, pero son algo más complicados de usar ya que no manejan este tipo de objetos, sino que manejan objetos más próximos a los que maneja el procesador (en última instancia direciones de memoria que tienen asignados valores binarios), de todas formas ése es el tema de la segunda parte de la asignatura.

Respecto al tema de la optimización es importante darse cuenta de que esta función construye e invierte la matriz del cambio de base **cada vez** que se ejecuta. Invertir una matriz es computacionalmente costoso si la dimensión es alta, por tanto esta función está bien si queremos aplicar el cambio sólo a un vector, pero si lo que queremos se aplicar el cambio de base sobre muchos vectores, en lugar

de aplicar muchas veces esta función lo que habría que hacer es modificarla de tal forma que acepte como segundo argumento no a un único vector, sino a toda la colección de vectores a los que queremos aplicar el cambio de base.

- A continuación definimos la función `cambiobasematriz`, que aplica cambios de base sobre matrices. Listado del archivo `cambio_base_matriz.mc` que contiene dicha función:

```
/* FUNCION "cambiobasematriz"
   Aplica cambios de base sobre aplicaciones lineales
   Input:
     1: "basenueva" = lista de vectores que forman la nueva base
     2: "M" = matriz sobre la que vamos a aplicar el cambio de base
   Output:
     componentes de la matriz M en la nueva base */

cambiobasematriz(basenueva, M) := block( [A, B],

  /* construimos la matriz del cambio de base y la guardamos en la
     variable local A */

  A : matrix(basenueva[1]),

  for i : 2 thru length(basenueva) step 1 do A : addrow(A, basenueva[
     i]),

  A : transpose(A),

  /* guardamos la inversa de la matriz del cambio de base en la
     variable local B */

  B : invert(A),

  /* aplicamos el cambio de base sobre M */

  B . (M . A)

)$

/* fin */
```

  Igual que antes, esta función construye la matriz del cambio y la invierte cada vez que se ejecuta, de modo que no es práctico para emplearlo sobre una colección de matrices.

- Una vez hecho esto es muy fácil hacer un programa que *deshaga* el cambio de base. Listado del archivo `cambio_base_invertir.mc` que contiene dicha las funciones `cambiobasevectorm1` y `cambiobasematrizm1`:

```
/* FUNCION "cambiobasevectorm1"
   Deshace cambios de base sobre vectores
```

```
    Input:
      1: "basenueva" = lista de vectores que forman la nueva base
      2: "VN" = coordenadas del vector en la base nueva
    Output:
    coordenadas del vector en la base vieja */

cambiobasevectorm1(basenueva, vector) := block( [A],

  /* construimos la matriz del cambio de base y la guardamos en la
     variable local A */

  A : matrix(basenueva[1]),

  for i : 2 thru length(basenueva) step 1 do A : addrow(A, basenueva[
     i]),

  A : transpose(A),

  /* aplicamos esta matriz sobre el vector */

  A . vector

)$

/* FUNCION "cambiobasematrizm1"
    Deshace cambios de base sobre matrices
    Input:
      1: "basenueva" = lista de vectores que forman la nueva base
      2: "MN" = coordenadas de la matriz en la base nueva
    Output:
      coordenadas de la matriz en la base vieja */

cambiobasematrizm1(basenueva, MN) := block( [A, B],

  /* construimos la matriz del cambio de base y la guardamos en la
     variable local A */

  A : matrix(basenueva[1]),

  for i : 2 thru length(basenueva) step 1 do A : addrow(A, basenueva[
     i]),

  A : transpose(A),

  /* guardamos la inversa de la matriz del cambio de base en la
     variable local B */

  B : invert(A),
```

```
  /* aplicamos la inversa del cambio de base sobre MN */

  A . (MN . B)

)$

/* fin */
```

- Para terminar el ejercicio 1 sólo nos queda escribir un programa que extraiga la lista de autovectores de una matriz. La función `eigenvectors` del maxima proporciona como segundo argumento del output una lista cuyos elementos son la lista de los autovectores correspondientes a cada uno de los autovalores de la matriz. Lo que necesitamos es sencillamente una lista de autovectores, y una posible forma de construirla a partir del output generado por `eigenvectors` es la siguiente función `eigenvectorlist`:

```
/* FUNCION "eigenvectorlist"
   Proporciona la lista de autovectores de una matriz
   Input:
     "M" = matriz
   Output:
     "EL" lista de autovectores */

eigenvectorlist(M) := block( [A, dimM, numV],

  /* asignamos a la variable local dimM la dimensionalidad del
     espacio vectorial */

  dimM : length(M),

  /* calculamos los autovectores y los guardamos en la variable
     local A */

  A : eigenvectors(M)[2],

  /* construimos una lista plana con las coordenadas de los
     autovectores */

  A : flatten(A),

  /* Para que la matriz sea diagonalizable es necesario que haya
     tantos autovectores
      como indica la dimensionalidad del espacio, pero eso no siempre
          ocurre.
      Guardamos en la variable local numV el cardinal del conjuto de
         autovectores. */

  numV : length(A) / dimM,
```

```
  /* asignamos estas coordenadas a una lista de numV vectores de
     dimM componentes */

  makelist(

    makelist(

      A[ dimM * i + j ]

    , j, 1, dimM)

  , i, 0, numV - 1)

)$

/* fin */
```

- Una vez que hemos programado estas funciones, para hacer el ejercicio 1 basta con cargarlas desde una sesión de wxMaxima y llamarlas en el orden adecuado:

```
kill(all);
path : ".../Fisica-Computacional-1/Maxima/Problemas/";
batchload(concat(path,"Algebra/cambio_base_vector.mc"));
batchload(concat(path,"Algebra/cambio_base_matriz.mc"));
batchload(concat(path,"Algebra/cambio_base_invertir.mc"));
batchload(concat(path,"Algebra/eigenvector_list.mc"));
```

  Ahora introduzca una matriz A a modo de ejemplo, para calcular la forma de A en la base formada por sus autovectores basta con hacer:

```
bn : eigenvectorlist(A);
AN : expand(cambiobasematriz(bn,A));
```

⋆ **¿Se obtiene el resultado esperado?**

Dependiendo de si la matriz A que introduzcamos arriba es diagonalizable o no esta función nos dará, o bien una matriz diagonal cuyos elementos no nulos son los autovalores de A, o bien un error. En la introducción al curso comentábamos que en computación lo más importante es saber qué es lo que estamos haciendo, por tanto, antes de ejecutar las instrucciones de arriba deberíamos saber si lo que le estamos pidiendo al MAXIMA es posible o no. En la asignatura de álgebra se estudian los criterios que permiten saber (muy fácilmente) si una matriz se puede diagonalizar o no, para las matrices no diagonalizables lo que se puede calcular es la *forma conónica de Jordan*, algo que también puede programarse en MAXIMA.

Ejercicio 2

A partir de la función para cambios de base realice un programa que calcule el producto escalar de dos vectores expresados por medio de sus componentes respecto de una base cualquiera.

Realmente la forma mas seria de calcular productos escalares en cualquier base es calculando previamente el *tensor métrico*, dado por los productos de los vectores base de la base nueva ($g_{ij} = \boldsymbol{t}_i \cdot \boldsymbol{t}_j$), lo que nos llevaría a calcular el producto escalar en términos de la base nueva como $\boldsymbol{u} \cdot \boldsymbol{v} = g_{ij}\hat{u}_i\hat{v}_j$ (donde se aplica el convenio de suma de Einstein para los índices i y j).

De todas formas, una manera inmediata de hacer este ejercicio a partir de las funciones que se pedía programar en este tema es usar la función que hemos definido antes para *deshacer* cambios de base, que nos proporciona las componentes de los vectores en la base estándar, y posteriormente aplicar la fórmula habitual para el producto escalar:

```
productoescalarBA(base1,V1,base2,V2):=cambiobasevectorm1(base1,V1).
   cambiobasevectorm1(base2,V2)$
```