Álgebra Lineal en Maple

José Luis Torres Rodríguez*

Abril 2011

Uno de los temas mejor soportados por Maple es el de Álgebra Lineal. Este sistema proporciona dos opciones para realizar cálculos propios de esta materia: el paquete **linalg** y el paquete **LinearAlgebra**. Ambos permiten realizar diversos cálculos sobre matrices y vectores; tales como operaciones aritméticas, calculo de inversas, transpuestas, determinantes, wronskianos, manipulacion de columnas y renglones; también permiten determinar polinomios caracteristicos, matrices caracteristicas, valores propios y vectores propios, así como diagonalizacion de matrices, entre otras cosas. Sin embargo, aunque ambos paquetes cuentan con soporte para este tipo de operaciones, en esta versión de Maple se han mejorado notablemente las capacidades y eficiencia de los algoritmos en base a los cuales está construido **LinearAlgebra** (el cual fue concebido como un sustituto de **linalg**), por lo que nos enfocaremos principalmente a éste a lo largo del presente capítulo. No obstante, mencionaremos las características y diferencias principales que existen entre ambos, de tal forma que el usuario pueda elegir cual de los dos paquetes usar.

1. Características de linalg y de LinearAlgebra

A continuación revisaremos las principales características de los paquetes antes mencionados. Posteriormente mostraremos las capacidades de **LinearAlgebra** para manejo de matrices y vectores.

1.1. El paquete linalg

Este paquete está formado por un conjunto de procedimientos, entre los cuales se incluyen alrededor de cien comandos que permiten realizar una gran diversidad de operaciones de Álgebra Lineal. Las características de este paquete son las siguientes:

- **Los arreglos son la estructura de datos básica**.

  Los vectores y matrices definidos y operados por las funciones de este paquete son manejados en forma de **arreglos**. Estos arreglos deben ser creados a partir de la función **array**, y aunque el paquete proporciona funciones para creación de vectores y matrices, éstas son casos particulares de la función **array**.

- **No proporciona funciones para creación de algunas matrices especiales**.

  linalg no contiene funciones predefinidas para creación, por ejemplo, de matrices cero o matrices identidad; éstas deben ser creadas en forma de arreglos o por medio de las funciones **vector** y ***matrix***. Una alternativa es crearlas a partir de secuencias o algún procedimiento alterno para generación de listas y despues convertirlas en vectores o matrices.

- **Evaluación de matrices y vectores**.

  Al ser creados en forma de arreglos, cuando una matriz o vector es asignado a una variable, al colocar este nombre en una linea de comandos, Maple no puede evaluarlo y desplegar su valor, en lugar de esto

*Coordinación de Cómputo, Facultad de Ciencias, UNAM

se despliega el nombre mismo. Para poder desplegar los elementos de una estructura de este tipo se debe usar la función ***eval*** o bien **print**.

- **Álgebra de matrices**.

  Al realizar cálculos algebraicos, Maple no puede evaluar arreglos por medio de los operadores aritméticos normales; es decir, no se puede evaluar una expresión de la forma **A**+**B**. Este tipo de operaciones deben realizarse por medio de la función **evalm**. En particular la multiplicación de matrices requiere el uso de esta función, así como del operador ***&****, el cual indica un producto no conmutativo.

- **Álgebra Lineal abstracta**.

  Es posible realizar cálculos de Álgebra Lineal abstracta, a través del uso de operadores inertes y la función **evalm**.

- **Manejo limitado de matrices grandes con elementos numéricos**.

  Las funciones del paquete ***linalg*** permiten manipular matrices con elementos tanto numéricos como simbólicos; sin embargo, su capacidad para operar matrices muy grandes con elementos numéricos es limitada.

1.2. El paquete LinearAlgebra

Este paquete contiene un conjunto de funciones de Álgebra Lineal, las cuales proporcionan las mismas funcionalidades del paquete ***linalg*** (y aún más). Entre sus características más notables se encuentran:

- **Estructuras de datos básicas**

  Las estructuras de datos usadas como base por las funciones de este paquete son **"Vector"** y **"Matrix"**. Estas estructuras son creadas a partir de los comandos **Vector** y **Matrix**; o bien, por medio de la notación abreviada ***"<a, b, c>"***. Internamente, este tipo de datos están basados en una estructura de datos conocida como ***"rtable"***. Por esta razón, las listas, los arreglos, las matrices y vectores de **linalg** y los arreglos basados en tablas no pueden combinarse con datos creados a partir de **rtables**, sin ser convertidos previamente.

- **Comandos para tipos especiales de matrices y vectores**.

  Este paquete contiene comandos para la creación de tipos especiales de matrices y vectores; tales como identidades, cero y constantes.

- **Álgebra de matrices mejorada**.

  Las matrices y vectores generados por las funciones de este paquete pueden ser operados usando los operadores aritméticos habituales; es decir, las expresiones del tipo **A**+**B**, **A.B**, **A-B** son evaluadas directamente sin necesidad de usar una función especial para calcularlas. Nótese que el producto es expresado usando el operador punto: **"."**.

- **Soporte más eficiente de matrices numéricas grandes**.

  El paquete está implementado en base a algorítmos numéricos más eficientes, lo que proporciona mejor soporte para manejo de matrices grandes de datos numéricos

2. Elección entre linalg y LinearAlgebra

Los puntos más importantes a tener en cuenta al elegir entre estos dos paquetes son:

- El paquete **linalg** es útil para realizar cálculos de álgebra abstracta.

- Comparado con **linalg**, el paquete **LinearAlgebra** incluye varias funciones para creación de matrices especiales, permite realizar álgebra de matrices de manera más fácil y es más poderoso y eficiente al realizar cálculos, especialmente cuando se manipulan matrices numéricas grandes.

3. Conversión de datos entre linalg y LinearAlgebra

Como se mencionó anteriormente, estos paquetes usan estructuras de datos diferentes como base para la creación de matrices y vectores, las cuales no pueden ser combinadas para operarlas. Sin embargo, los datos creados con las funciones de uno de ellos pueden ser convertidos para ser usados por funciones del otro paquete. Esto puede hacerse de la siguiente forma:

- Los vectores creados con el paquete **linalg** deben ser convertidos mediante la función:

  convert(..., Vector),

  para ser usados por **LinearAlgebra**.

- Las matrices creadas con **linalg**, deben convertirse usando la instrucción:

  convert(..., Matrix).

- Los vectores creados con **LinearAlgebra** deben ser convertidos con:

  convert(..., vector)

  para poder ser usados por **linalg**.

- Las matrices creadas con **LinearAlgebra** deben convertise mediante:

  convert(..., matrix)

Adicionalmente, una matrix o un vector creados con ***linalg*** puede ser usado como argumento en las funciones **Matrix** y **Vector** de **LinearAlgebra**, respectivamente; con lo cual contamos con otro mecanismo para convertir datos de **linalg** a **LinearAlgebra**.

En las siguientes secciones haremos uso de las funciones contenidas en el paquete **LinearAlgebra**, por lo cual es conveniente cargar éste antes de continuar.

```
> with(LinearAlgebra):
```

4. Vectores

A continuación haremos una revisión de las principales instrucciones proporcionadas por Maple, útiles en la creación, manipulación y operación de vectores.

4.1. Creación de un vector

Un vector puede ser creado por medio de la función **Vector**. Esta instrucción puede ser usada de varias formas para la creación de vectores, una de esas formas es:

Vector([v1, v2, v3 ,..., vn])

Donde ***"v1, v2, v3, ... , vn"*** son las entradas del vector; la dimensión de éste es calculada automáticamente con el número de entradas proporcionadas, aunque también puede especificarse como primer argumento, en cualquiera de las siguientes formas:

Vector(1..n, [v1, v2, v3, ..., vn])

Vector(n, [v1, v2, v3, ..., vn])

Una características interesante de esta función es que nos da la posibilidad de especificar si deseamos crear un vector columna o un vector renglón. La sintaxis es:

Vector[row](...)

Vector[column](...);

Si no se especifica el tipo de vector a crear, esto es equivalente a la forma: **Vector[column](...)**.

Por ejemplo, a continuación crearemos los vectores **v1** y **v2**, y calcularemos su norma por medio de la función **VectorNorm** de **LinearAlgebra**.

```
> v1 := Vector[row]([9, 3, 2.4]);
```

$$v1 := [9,\ 3,\ 2{,}4]$$

```
> v2 := Vector[column](4, [3.5, 0.4, 7.2, Pi/2]);
```

$$v2 := \begin{bmatrix} 3{,}5 \\ 0{,}4 \\ 7{,}2 \\ \frac{\pi}{2} \end{bmatrix}$$

```
> VectorNorm(v1, Euclidean);
```

$$9{,}78570385818005484$$

```
> VectorNorm(v2, Euclidean);
```

$$\sqrt{64{,}25 + \frac{\pi^2}{4}}$$

Otra forma en la que podemos crear este tipo de elementos es por medio de la notación abreviada:

<**v1, v2, v3, ..., vn**>. Para crear vectores columna.

<**v1** | **v2** | **v3** | **...** | **vn**>. Para crear vectores renglón.

Donde **v1, v2, v3, ..., vn** son las entradas del vector. Por ejemplo:

```
> v3 := <Pi/4, alpha, 3, sqrt(Pi)>;
```

$$v3 := \begin{bmatrix} \frac{\pi}{4} \\ \alpha \\ 3 \\ \sqrt{\pi} \end{bmatrix}$$

```
> v4 := <4.5 | 2 | 8.4 | 3>;
```

$$v4 := [4{,}5,\ 2,\ 8{,}4,\ 3]$$

Este tipo de estructuras son evaluadas automáticamente al colocar su nombre en la linea de comandos, por ejemplo, para desplegar los elementos de los vectores **v1** y **v4** simplemente los referenciamos:

```
> v1;
```

$$[9,\ 3,\ 2{,}4]$$

```
> v4;
```

$$[4{,}5,\ 2,\ 8{,}4,\ 3]$$

Adicionalmente, al crear un vector con la función **Vector**, podemos incluir la opción **'ro'**, la cual indica a Maple que estamos creando un vector de solo lectura, es decir, ninguna de las entradas de este vector podran ser modificadas.

4.2. Creación de un vector a partir de arreglos y listas

Maple nos permite obtener vectores a partir de listas y arreglos previamente definidos, convirtiendo éstos por medio de la función **convert**. Esta función nos devuelve de manera predeterminada vectores columna, sin embargo, podemos especificar el tipo de vector que deseamos obtener de la siguiente manera:

convert(lista, Vector[column])

convert(lista, Vector[row])

La primera forma es equivalente a **convert(lista, Vector)**. Por ejemplo, crearemos los siguientes vectores y calcularemos su producto por medio de la función **DotProduct** y la norma de ambos por medio de la función **VectorNorm** (ambos pertenecen a **LinearAlgebra**).

```
> lista := [4, 8, 9, 3];
```

$$lista := [\,4,\ 8,\ 9,\ 3\,]$$

```
> ar := array(3..6, [3, 12, 4.6, 8]);
```

$$ar := \text{array}(3..6,\ [$$
$$(3) = 3$$
$$(4) = 12$$
$$(5) = 4{,}6$$
$$(6) = 8$$
$$])$$

```
> v5 := convert(lista,Vector);
```

$$v5 := \begin{bmatrix} 4 \\ 8 \\ 9 \\ 3 \end{bmatrix}$$

```
> v6 := convert(ar, Vector[row]);
```

$$v6 := [\,3,\ 12,\ 4{,}6,\ 8\,]$$

```
> DotProduct(v5, v6);
```

$$173{,}400000000000006$$

```
> VectorNorm(v5);
```

$$9$$

```
> VectorNorm(v6);
```

$$12.$$

Nótese que los índices asignados a los arreglos no deben iniciar forzosamente en uno para poder convertirlos en vectores.

Otra forma en la que podemos crear un vector a partir de una lista o un arreglo es incluyendo éstos como argumento en la función **Vector**, como se muestra a continuación:

```
> Vector(4, ar);
```

$$[\,3,\ 12,\ 4{,}6,\ 8\,]$$

```
> Vector(4, lista);
```

$$\begin{bmatrix} 4 \\ 8 \\ 9 \\ 3 \end{bmatrix}$$

En este caso, de manera predeterminada los arreglos son convertidos a vectores renglón, mientras que las listas son convertidas a vectores columna.

4.3. Acceso a los elementos de un vector

Los elementos de un vector pueden ser invocados y modificados de la misma forma que en el caso de las listas, es decir haciendo referencia a sus elementos por medio de índices, los cuales siempre inician en uno. Para ejemplificar esto usaremos el vector **v2** creado anteriormente:

```
> v2;
```

$$\begin{bmatrix} 3{,}5 \\ 0{,}4 \\ 7{,}2 \\ \frac{\pi}{2} \end{bmatrix}$$

```
> v2[1]; v2[3];
```

$$3{,}5$$
$$7{,}2$$

```
> v2[2] := diff(x^2, x);
```

$$v2_2 := 2\,x$$

```
> v2;
```

$$\begin{bmatrix} 3{,}5 \\ 2\,x \\ 7{,}2 \\ \frac{\pi}{2} \end{bmatrix}$$

4.4. Vectores creados sin elementos

Podemos crear vectores sin especificar sus elementos y asignar éstos posteriormente, especificando unicamente la dimensión. En este caso el vector es creado con ceros en todas sus entradas. Por ejemplo:

```
> v7 := Vector(3);
```

$$v7 := \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> v7[1] := 8; v7[2] := 4; v7[3] := 6;
```

$$v7_1 := 8$$
$$v7_2 := 4$$
$$v7_3 := 6$$

```
> v7;
```

$$\begin{bmatrix} 8 \\ 4 \\ 6 \end{bmatrix}$$

4.5. Vectores aleatorios

Este tipo de vectores pueden ser creados con la función **RandomVector(n)**, donde **n** es la dimensión del vector a crear. Esta función nos devuelve un vector columna, aunque opcionalmente también puede especificarse el tipo de vector de la siguiente forma:

RandomVector[row](n)

RandomVector[column](n)

Por ejemplo:

```
> RandomVector(5);
```

$$\begin{bmatrix} 30 \\ 62 \\ -79 \\ -71 \\ 28 \end{bmatrix}$$

```
> RandomVector[row](8);
```

$$[16, -34, -62, -90, -21, -56, -8, -50]$$

4.6. Vectores generados por funciones

Otra manera de crear vectores es a través del uso de la función **Vector**, de la siguiente manera:

Vector(d, f)

Donde **d** es la dimensión del vector y **f** es una función de una variable que se aplicará a los índices del vector para generar los elementos correspondientes. Por ejemplo:

```
> f := x -> x^2 + 4*x;
```

$$f := x \to x^2 + 4\,x$$

```
> d := Vector(4, f);
```

$$d := \begin{bmatrix} 5 \\ 12 \\ 21 \\ 32 \end{bmatrix}$$

```
> e := Vector[row](5, f);
```

$$e := [5, 12, 21, 32, 45]$$

Esta última expresión es equivalente a la siguiente:

```
> e := Vector[row](5, [f(1), f(2), f(3), f(4), f(5)]);
```

$$e := [5, 12, 21, 32, 45]$$

4.7. Aplicación de funciones a los elementos de un vector

Una vez definido un vector, podemos usar **map** para aplicar una función dada a cada uno de sus elementos. Por ejemplo:

```
> v9 := Vector([4, Pi, exp(1.1), 4*sin(Pi/4), Pi^2]);
```

$$v9 := \begin{bmatrix} 4 \\ \pi \\ 3{,}004166024 \\ 2\sqrt{2} \\ \pi^2 \end{bmatrix}$$

```
> map(cos, v9);
```

$$\begin{bmatrix} \cos(4) \\ -1 \\ -0{,}9905718132 \\ \cos(2\sqrt{2}) \\ \cos(\pi^2) \end{bmatrix}$$

```
> v10 := Vector[row](5, n -> n^2 + x^n);
```

$$v10 := [\,1 + x,\ 4 + x^2,\ 9 + x^3,\ 16 + x^4,\ 25 + x^5\,]$$

```
> map(diff, v10, x);
```

$$[\,1,\ 2\,x,\ 3\,x^2,\ 4\,x^3,\ 5\,x^4\,]$$

El paquete **LinearAlgebra** contiene otras dos funciones similares a **map**, éstas son: **Map** y **Map2**, las cuales proporcionan algunas características más para realizar mapeos de funciones sobre vectores. Consúltense sus páginas de ayuda.

Las funciones que proporciona Maple para manipulación de vectores nos permiten realizar, entre otras, las operaciones que se describen en las siguientes secciones. Muchas de estas operaciones están soportadas a través de funciones del paquete **LinearAlgebra**; por lo que es conveniente asegurarse de que éste se ha cargado previamente.

4.8. Operaciones aritméticas con vectores

Este tipo de operaciones pueden realizarse directamente en la linea de comandos, por medio de los operadores aritméticos usuales. Veamos los siguientes ejemplos:

```
> v := <-3 | 1 | 7>;
```

$$v := [\,-3,\ 1,\ 7\,]$$

```
> w := <9 | -3 | -21>;
```

$$w := [\,9,\ -3,\ -21\,]$$

Calcularemos las operaciones: **v** + **w**, **4*w**, ***w*4*** y **7*v** + **w**.

```
> v + w;
```

$$[\,6,\ -2,\ -14\,]$$

```
> w - v;
```

$$[\,12,\ -4,\ -28\,]$$

```
> 4*w;
```

$$[\,36,\ -12,\ -84\,]$$

```
> w*4;
```

$$[\,36,\ -12,\ -84\,]$$

```
> 7*v + w;
```

$$[-12,\ 4,\ 28]$$

Como puede observarse, el operador **'*'** nos permite calcular el producto de un vector por un escalar. Otro operador aplicable en este caso es **'.'**, el cual nos devuelve el producto punto de los vectores:

```
> v.w;
```

$$-177$$

Estas operaciones también pueden calcularse por medio de las funciones:

Add(v1, v2)

ScalarMultiply(v1, c)

DotProduct(v1, v2)

Las cuales nos devuelven la suma de dos vectores, el producto de un vector por un escalar y el producto punto de dos vectores, respectivamente. **v1** y **v2** representan vectores, mientras que **c** representa un escalar.

A continuación comprobaremos si los vectores **w** y **v** (definidos arriba) son paralelos, para verificar esto debemos encontrar un valor **t** tal que **v**=**t***__w__. Esto podemos hacerlo de la siguiente forma.

Primero definamos el siguiente vector:

```
> m := ScalarMultiply(w, t);
```

$$m := [9\,t,\ -3\,t,\ -21\,t]$$

Necesitamos encontrar ***t*** tal que ***v***=***m***, para ello debemos resolver el siguiente sistema de ecuaciones:

```
> sis := {v[1] = m[1], v[2] = m[2], v[3] = m[3]};
```

$$sis := \{-3 = 9\,t,\ 1 = -3\,t,\ 7 = -21\,t\}$$

```
> sol := solve(sis, t);
```

$$sol := \{t = \frac{-1}{3}\}$$

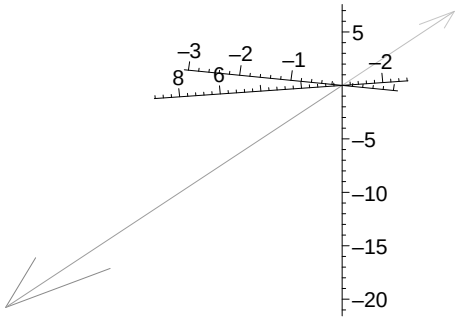
Verifiquemos multiplicando **w** por el valor obtenido y comprobemos si el resultado es igual a **v**:

```
> rhs(sol[1])*w;
```

$$[-3,\ 1,\ 7]$$

Por lo tanto los vectores son paralelos. Grafiquemos **v** y **w**.

```
> grafv := plots[arrow](v, shape=arrow):
> grafw := plots[arrow](w, shape=arrow):
> plots[display](grafv, grafw, axes=normal, orientation=[50, 85]);
```



En la gráfica puede observarse que los vectores son paralelos.

A continuacion, calcularemos la ecuación de la recta que pasa por los puntos **f** y **g** y desplegaremos su gráfica.

```
> f := <2 | 5 | 8>;
```

$$f := [2, 5, 8]$$

```
> g := <3 | -4 | 2>;
```

$$g := [3, -4, 2]$$

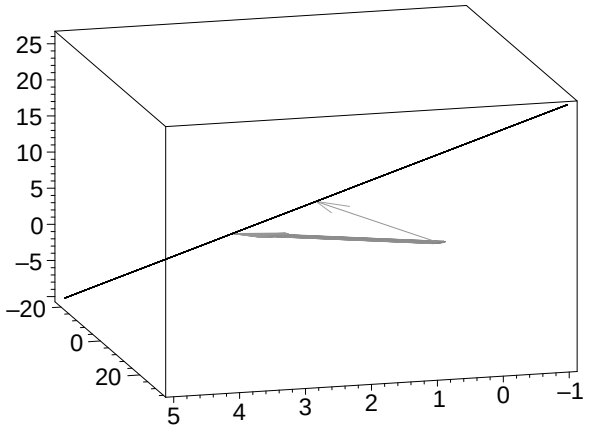
La ecuación de la recta está dada por $\mathbf{x} = \mathbf{f} + \mathbf{t(g - f)}$. Ésta debe ser convertida a lista para poder colocarla como argumento de **plot3d**.

```
> recta := convert(evalm(f + t*(g - f)), list);
```

$$recta := [t + 2, -9\,t + 5, -6\,t + 8]$$

Generamos las gráficas de los vectores y de la recta, y las desplegamos.

```
> grf := plots[arrow](f, shape=arrow):
> grg := plots[arrow](g):
> grrec := plot3d(recta, t=-3..3,y=-3..3, axes=boxed):
> plots[display]({grf, grg, grrec}, orientation=[75, 70]);
```



A continuación, calculamos y graficamos el plano que generan los puntos **P**, **Q** y **R**.

```
> P := <3 | 6 | 7>;
```

$$P := [3, 6, 7]$$

```
> Q := <-4 | 7 | 9>;
```

$$Q := [-4, 7, 9]$$

```
> R := <5 | -9 | 15>;
```

$$R := [5, -9, 15]$$

Primero calcularemos el vector que parte del origen y tienen la misma dirección que el vector que va de **P** a **Q**, lo mismo para el vector que va de **P** a **R**.

```
> u := Q - P; # vector que va de P a Q
```

$$u := [-7,\ 1,\ 2]$$

```
> v := R - P; # vector que va de P a R
```

$$v := [2,\ -15,\ 8]$$

La ecuación del plano está dada por **x** = **P** + **t*u** + **s*v**.
A continuación generaremos las gráficas de **P**, **Q** y **R**, así como la del plano.

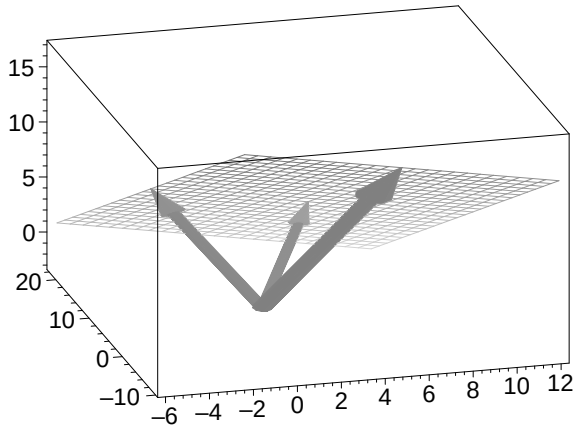
```
> grP := plots[arrow](P):
> grQ := plots[arrow](Q):
> grR := plots[arrow](R):
> plano := convert(evalm(P + t*u + s*v), list);
```

$$plano := [-7\,t + 2\,s + 3,\ t - 15\,s + 6,\ 2\,t + 8\,s + 7]$$

```
> grplano := plot3d(plano, t=-1..1, s=-1..1):
```

En el siguiente despliegue mostraremos todas estas gráficas, con lo cual podemos darnos cuenta de que, efectivamente, el plano es generado por los vectores **P**, **Q** y **R**.

```
> plots[display]({grP, grQ, grR, grplano}, style=wireframe,
> axes=boxed, orientation=[-105, 60]);
```



Veamos otro ejemplo. Consideremos la siguiente definición:
Sea **V** el conjunto de parejas ordenadas de números reales. Para $(x_1,\ y_1)$ y $(x_2,\ y_2) \in \mathbf{V}$, y **c** un número real, definimos:

$$(x_1,\ y_1) + (x_2,\ y_2) = (x_1 + y_1,\ x_2 + y_2)$$

$$c\,(x_1,\ y_1) = \begin{cases} (0,0) & \text{si } c = 0 \\ (c\,x_1,\ \frac{y_1}{2}) & \text{si } c \neq 0 \end{cases}$$

¿Es **V** un espacio vectorial bajo estas operaciones?
Para que ***V*** sea un espacio vectorial, dos de las condiciones que deben cumplirse son, para P, Q elementos de **V** y c, d números reales:

$$c\,(P+Q) = c\,P + c\,Q$$

$$(c+d)\,P = c\,P + d\,P$$

Definamos los siguientes vectores y constantes:

```
> P := <4 | 12>;
```

$$P := [\,4,\,12\,]$$

```
> Q := <3 | 8>;
```

$$Q := [\,3,\,8\,]$$

```
> c := 4;
```

$$c := 4$$

```
> d := 3;
```

$$d := 3$$

A continuación creamos la función producto, de acuerdo a la definición del producto por escalar de **V**:

```
> producto := proc(c::numeric, V::Vector)
> if (c = 0)
> then
> <0 | 0>;
> else
> <c*V[1] | V[2]/c>;
> fi;
> end:
```

Ahora calculamos $c\,(P+Q)$, $c\,P + c\,Q$, y los comparamos:

```
> producto(c, P + Q);
```

$$[\,28,\,5\,]$$

```
> producto(c, P) + producto(c, Q);
```

$$[\,28,\,5\,]$$

Esta relación si se cumple. Verifiquemos ahora si: $(c+d)\,P = c\,P + d\,P$.

```
> producto(c + d, P);
```

$$\left[\,28,\,\frac{12}{7}\,\right]$$

```
> producto(c, P) + producto(d,P);
```

$$[\,28,\,7\,]$$

Podemos ver en este caso que la relación no se cumple; por lo tanto, **V** no es un espacio vectorial.

Consideremos ahora el mismo conjunto **V** de arriba, pero ahora definimos las siguientes operaciones, para (x_1, y_1) y $(x_2, y_2) \in \mathbf{V}$ y c un número real:

$$(x_1,\,y_1) + (x_2,\,y_2) = (x_1 + 2\,y_1,\,x_2 + 3\,y_2)$$

$$c\,(x_1,\,y_1) = (c\,x_1,\,c\,y_1)$$

¿Es V un espacio vectorial bajo estas operaciones?

Esta comprobación podemos hacerla aprovechando la capacidad de Maple para manipular datos simbólicos. Utilizaremos la función **evalb**, la cual nos permite verificar una expresión booleana.

Dos de las condiciones necesarias para que ***V*** sea un espacio vectorial, dados **P**, **Q**, ***R*** en **V**, son:

$$P + Q = Q + P$$

$$(P + Q) + R = P + (Q + R)$$

Verifiquemos si éstas se cumplen, para **P**, **Q** y **R** elementos generales de **V**.

```
> P := <x1 | y1>;
```

$$P := [\, x1,\ y1 \,]$$

```
> Q := <x2 | y2>;
```

$$Q := [\, x2,\ y2 \,]$$

```
> R := <x3 | y3>;
```

$$R := [\, x3,\ y3 \,]$$

```
> suma := proc(V::Vector, W::Vector)
> <V[1] + 2*W[1] | V[2] + 3*W[2]>;
> end;
```

$$suma := \mathbf{proc}(V{::}\mathit{Vector},\ \mathrm{W}{::}\mathit{Vector})\ \langle V_1 + 2 * \mathrm{W}_1 \,|\, V_2 + 3 * \mathrm{W}_2 \rangle\ \mathbf{end\ proc}$$

```
> evalb(suma(P, Q) = suma(Q, P));
```

$$\mathit{false}$$

```
> evalb(suma(suma(P, Q), R) = suma(P, suma(Q, R)));
```

$$\mathit{false}$$

De lo anterior podemos ver que, en general, las dos relaciones no se cumplen, por lo tanto $\boldsymbol{V}$ no es espacio vectorial bajo estas operaciones. Comprobemos esto mismo con valores particulares de P, Q y R.

```
> P := <2 | 4>;
```

$$P := [\, 2,\ 4 \,]$$

```
> Q := <3 | 5>;
```

$$Q := [\, 3,\ 5 \,]$$

```
> R := <4 | 7>;
```

$$R := [\, 4,\ 7 \,]$$

```
> suma(P, Q) = suma(Q, P);
```

$$[\, 8,\ 19 \,] = [\, 7,\ 17 \,]$$

```
> suma(suma(P, Q), R) = suma(P, suma(Q, R));
```

$$[\, 16,\ 40 \,] = [\, 24,\ 82 \,]$$

4.9. Funciones de LinearAlgebra para manipulación de vectores

El paquete **LinearAlgebra** proporciona varias funciones útiles en el manejo de vectores, entre ellas se encuentran las que se describen en las siguientes secciones.

4.9.1. Ángulo entre dos vectores

La función **VectorAngle** nos permite calcular el ángulo entre dos vectores. Por ejemplo:

```
> v1 := <6 | 3>;
```

$$v1 := [\, 6,\ 3 \,]$$

```
> v2 := <-3 | 6>;
```

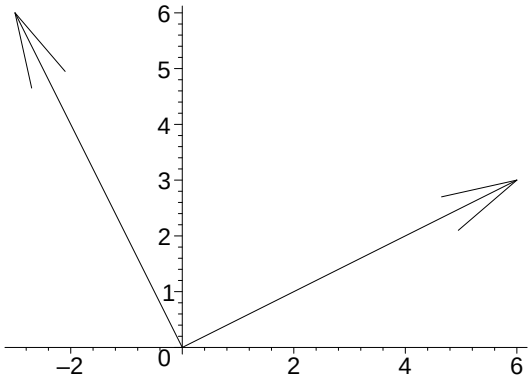
$$v2 := [\, -3,\ 6 \,]$$

```
> VectorAngle(v1, v2);
```

$$\frac{\pi}{2}$$

El resultado anterior se debe a que los vectores son perpendiculares. Verifiquemos con una gráfica:

```
> grafv1 := plots[arrow](v1, shape=arrow):
> grafv2 := plots[arrow](v2, shape=arrow):
> plots[display]({grafv1, grafv2}, scaling=constrained);
```



4.9.2. Igualdad de dos vectores

Dados dos vectores, la función **Equal** nos permite determinar si sus respectivos componentes son iguales (también puede ser usada con matrices), su sintaxis es:

Equal(V, W)

Donde **V** y **W** son vectores. Esta función devuelve verdadero si ambos argumentos son del mismo tipo (vectores o matrices), tienen la misma dimensión y sus componentes son iguales. En particular, si sus argumentos son vectores, éstos deben tener la misma orientación. Por ejemplo:

```
> a := <1 | 3 | 7>;
```

$$a := [1, 3, 7]$$

```
> b := <1 | 3 | 7>;
```

$$b := [1, 3, 7]$$

```
> c := <4 | 5 | 2>;
```

$$c := [4, 5, 2]$$

```
> Equal(a, b);
```

true

```
> Equal(b,c);
```

false

Esta función proporciona algunas otras opciones útiles en la comparación de vectores. Consúltese su página de ayuda.

4.9.3. Producto punto

Otra de las funciones presentes en este paquete es **DotProduct**, con la cual podemos calcular el producto punto de dos vectores. Por ejemplo, calcularemos el producto punto de los siguientes vectores para verificar si son perpendiculares.

```
> v1 := <6 | 3>;
```

$$v1 := [6, 3]$$

```
> v2 := <-3 | 6>;
```

$$v2 := [-3, 6]$$

```
> DotProduct(v1, v2);
```

$$0$$

El resultado que obtuvimos es cero, por lo tanto los vectores son perpendiculares.

Recuérdese que este producto también puede calcularse por medio del operador **'.'**; es decir, la instrucción anterior es equivalente a la operación **v1.v2**.

4.9.4. Norma de un vector

También podemos calcular la norma de un vector, utilizando la función **Norm**, o por medio de **VectorNorm**, ambas contenidas en **LinearAlgebra**. La sintaxis es:

Norm(V, Euclidean);

VectorNorm(V, Euclidean);

Por ejemplo:

```
> A := <-7 | 5>;
```

$$A := [-7, 5]$$

```
> Norm(A, Euclidean);
```

$$\sqrt{74}$$

Verifiquemos este resultado calculando manualmente la norma de **A**.

```
> sqrt(A[1]^2 + A[2]^2);
```

$$\sqrt{74}$$

4.9.5. Producto cruz

Otra operación que podemos aplicar sobre vectores está dada por la función **CrossProduct**, la cual nos permite calcular el producto cruz de dos vectores de dimensión tres. Por ejemplo, encontraremos un vector unitario perpendicular a los vectores **W, Z**.

```
> W := <2 | 5 | -4>;
```

$$\mathrm{W} := [2, 5, -4]$$

```
> Z := <-4 | 6 | 8>;
```

$$Z := [-4, 6, 8]$$

Calculamos el producto cruz:

```
> P := CrossProduct(W, Z);
```

$$P := [64, 0, 32]$$

A continuación calculamos la norma de **P** y el producto **P * 1/norma(P)**, el resultado es un vector **R**, unitario y perpendicular a **W** y **Z**.

```
> normaP := Norm(P, Euclidean);
```

$$normaP := 32\sqrt{5}$$

```
> R := 1/normaP * P;
```

$$R := \left[\frac{2\sqrt{5}}{5},\ 0,\ \frac{\sqrt{5}}{5}\right]$$

Para verificar que el resultado es un vector unitario y perpendicular calcularemos los productos **W.R** y **Z.R**, y la norma de **R**.

```
> W.R;
```

$$0$$

```
> Z.R;
```

$$0$$

```
> Norm(R, Euclidean);
```

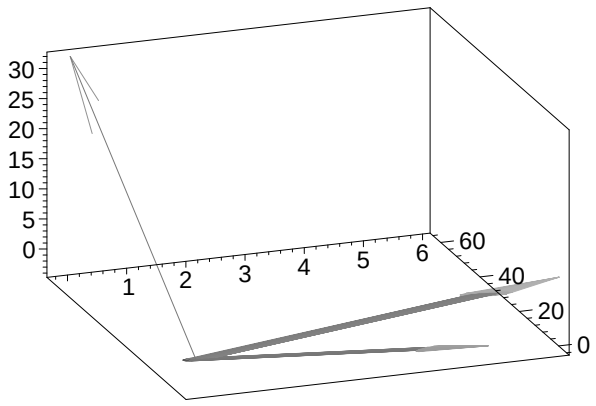
$$1$$

A continuación generaremos las gráficas de los vectores **W**, **Z** y **P**.

```
> grafW := plots[arrow](W):
> grafZ := plots[arrow](Z):
> grafP := plots[arrow](P, shape=arrow):
```

Finalmente mostraremos estas gráficas en un mismo despliegue para comprobar que el vector **P** (y por consiguiente también **R**) es perpendicular a **W** y **Z**.

```
> plots[display]({grafW, grafZ, grafP}, axes=boxed, orientation=[20, 120]);
```



4.9.6. Dimensión de un vector

La función **Dimension** nos permite calcular la dimensión de un vector (también puede usarse para calcular la dimensión de una matriz).

```
> g := x -> 2*x + 4*x^2;
```

$$g := x \rightarrow 2\,x + 4\,x^2$$

```
> w := Vector(5, g);
```

$$w := \begin{bmatrix} 6 \\ 20 \\ 42 \\ 72 \\ 110 \end{bmatrix}$$

```
> Dimension(w);
```

5

Este paquete proporciona algunas otras funciones para manipulación de estructuras creadas con **Vector** (o con la notación abreviada "<...>"), además de algunos comandos para creación de vectores especiales. Entre ellos se encuentran:

- **UnitVector(i, d)**. Construye un **Vector** de dimensión **d**, cuya entrada i-esima es igual a uno y las demas son cero.
- **VectorAdd(V, W)**. Calcula la suma de los vectores **V** y **W**. Esta función puede ser usada de la misma forma que **Add**; aunque tambien puede ser usada en la forma: **VectorAdd(V, W, c, d)**, para calcular la combinación lineal **c*V + d*W**.
- **VectorScalarMultiply(V, c)**. Calcula el producto del vector **V** por el escalar **c**.
- **ZeroVector(d)**. Construye un **Vector** de dimensión **d**, con todos sus elementos iguales a cero.

Antes de continuar ejecútese la siguiente instrucción para eliminar cualquier resultado anterior.

```
> restart;
```

Asegúrese de cargar nuevamente el paquete **LinearAlgebra**, con la instrucción **with(LinearAlgebra)**; ya que las funciones de éste serán usadas en las siguientes secciones.

5. Matrices

Maple proporciona un conjunto de funciones útiles en el manejo de matrices. En las siguientes secciones haremos una revisión de las capacidades de este sistema, para el manejo y operación de este tipo de datos.

5.1. Creación de matrices

Este tipo de dato puede ser creado por medio de la función **Matrix**, usando una de las siguientes expresiones.

Matrix([[a11, a12, ..., a1n], [a21, a22, ..., a2n], ... [am1, am2, ..., amn]])

Matrix(m, n, [[a11, a12, ..., a1n], [a21, a22, ..., a2n], ... [am1, am2, ..., amn]])

Matrix(1..m, 1..n, [[a11, a12, ..., a1n], [a21, a22, ..., a2n], ... [am1, am2, ..., amn]])

En estas expresiones, **aij** representa el elemento del renglón i-esimo y la columna j-esima de la matriz. En el primer caso, al no incluirse las dimensiones, éstas son calculadas por el número de elementos proporcionados. Las siguientes dos formas muestran maneras diferentes de especificar las dimensiones.

Al igual que en el caso de los vectores, también podemos usar la siguiente notación abreviada para la creación de este tipo de estructuras:

<<a11, a12, ..., a1n> | <a21, a22, ..., a2n> | ... | <am1, am2, ..., amn>>

para la creación de una matriz por columnas, o bien:

<<a11 | a12 | ... | a1n>, <a21 | a22 | ... | a2n>, ... , <am1 | am2 | ... | amn>>

para construirla por renglones. Por ejemplo:

```
> m := Matrix(1..3, 1..3, [[1, 2, 3], [5, 2, 1], [3, 6, 2]]);
```

$$m := \begin{bmatrix} 1 & 2 & 3 \\ 5 & 2 & 1 \\ 3 & 6 & 2 \end{bmatrix}$$

```
> n := Matrix([[2, 4, 6], [3, 8, 2]]);
```

$$n := \begin{bmatrix} 2 & 4 & 6 \\ 3 & 8 & 2 \end{bmatrix}$$

```
> q := «2 | 4 | 5>, <6 | 7 | 2»;
```

$$q := \begin{bmatrix} 2 & 4 & 5 \\ 6 & 7 & 2 \end{bmatrix}$$

```
> r := «4, 6, 2, 1> | <3, 5, 8, 3> | <9, 2, 4, Pi»;
```

$$r := \begin{bmatrix} 4 & 3 & 9 \\ 6 & 5 & 2 \\ 2 & 8 & 4 \\ 1 & 3 & \pi \end{bmatrix}$$

Los elementos pueden o no ser especificados durante la creación de la matriz; en caso de que no sean proporcionados, ésta es creada con ceros. Por ejemplo:

```
> s := Matrix(1..3,1..2);
```

$$s := \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

```
> t := Matrix(2,2);
```

$$t := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

Los elementos pueden ser asignados posteriormente. Éstos pueden ser accesados por medio de sus índices, de la misma forma que en las listas. Por ejemplo:

```
> s[1, 1] := 4: s[1, 2] := 8: s[2, 1] := 3: s[2, 2] := 9:
> s[3, 1] := 4: s[3, 2] := 6:

> s;
```

$$\begin{bmatrix} 4 & 8 \\ 3 & 9 \\ 4 & 6 \end{bmatrix}$$

Nótese (en la última instrucción) que para desplegar los elementos de una matriz es suficiente con invocar ésta a través de su nombre, Maple automáticamente la evalua y la muestra.

5.2. Matrices generadas por funciones

La función **Matrix** nos permite dar como argumento una función de dos variables, la cual se usará para generar los elementos de la matriz, aplicandola sobre los índices de los mismos elementos.

Por ejemplo:

```
> f := (i, j) -> x^i + sin(y^j);
```

$$f := (i,\, j) \to x^i + \sin(y^j)$$

```
> m := Matrix(3, 3, f);
```

$$m := \begin{bmatrix} x+\sin(y) & x+\sin(y^2) & x+\sin(y^3) \\ x^2+\sin(y) & x^2+\sin(y^2) & x^2+\sin(y^3) \\ x^3+\sin(y) & x^3+\sin(y^2) & x^3+\sin(y^3) \end{bmatrix}$$

Existen otras opciones proporcionadas por la función **Matrix** para la creación de este tipo de estructuras. Por ejemplo, si incluimos la opción **readonly=true**, la matriz será creada como de solo lectura, en cuyo caso no podemos modificar ninguna de sus entradas.

Algunas otras opciones serán tratadas en las siguientes secciones. Consúltese la página de ayuda de esta función para obtener información detallada de las distintas formas que proporciona para crear matrices.

5.3. Mapeo de funciones sobre matrices

El comando **Map** de **LinearAlgebra** puede ser usado para aplicar una función a cada uno de los elementos de una matriz.

Veamos un ejemplo:

```
> r;
```

$$\begin{bmatrix} 4 & 3 & 9 \\ 6 & 5 & 2 \\ 2 & 8 & 4 \\ 1 & 3 & \pi \end{bmatrix}$$

```
> t := Map(x -> x^2 + 4*x - 5*a, r);
```

$$t := \begin{bmatrix} 32-5\,a & 21-5\,a & 117-5\,a \\ 60-5\,a & 45-5\,a & 12-5\,a \\ 12-5\,a & 96-5\,a & 32-5\,a \\ 5-5\,a & 21-5\,a & \pi^2+4\,\pi-5\,a \end{bmatrix}$$

```
> Map(int, t, a);
```

$$\begin{bmatrix} 32\,a-\frac{5}{2}\,a^2 & 21\,a-\frac{5}{2}\,a^2 & 117\,a-\frac{5}{2}\,a^2 \\ 60\,a-\frac{5}{2}\,a^2 & 45\,a-\frac{5}{2}\,a^2 & 12\,a-\frac{5}{2}\,a^2 \\ 12\,a-\frac{5}{2}\,a^2 & 96\,a-\frac{5}{2}\,a^2 & 32\,a-\frac{5}{2}\,a^2 \\ 5\,a-\frac{5}{2}\,a^2 & 21\,a-\frac{5}{2}\,a^2 & \pi^2\,a+4\,\pi\,a-\frac{5}{2}\,a^2 \end{bmatrix}$$

5.4. Creación de matrices especiales

La función **Matrix** soporta varias opciones que podemos usar para crear matrices especiales; también el paquete **LinearAlgebra** proporciona varios comandos útiles para este fin. Veamos algunos ejemplos:

- Matrices identidad. Pueden ser creadas con:

  Matrix(n, shape=identity)

  o bien:

  Matrix(1..n, shape=identity)

Donde **n** determina la dimensión de la matriz. Las matrices identidad también pueden ser creadas por medio de la función **IdentityMatrix(n)** del paquete **LinearAlgebra**, como se vera a continuación:

```
> Matrix(4, shape=identity);
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
> IdentityMatrix(3);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Matrices de constantes Pueden ser creadas de la siguiente forma:

  Matrix(n, m, c)

  Donde **n**, **m** indican las dimensiones de la matriz y **c** es una constante. También pueden crearse por medio de la función:

  ConstantMatrix(c, n, m)

  Por ejemplo:

```
> Matrix(2, 3, 5);
```

$$\begin{bmatrix} 5 & 5 & 5 \\ 5 & 5 & 5 \end{bmatrix}$$

```
> ConstantMatrix(Pi/2, 2, 4);
```

$$\begin{bmatrix} \frac{\pi}{2} & \frac{\pi}{2} & \frac{\pi}{2} & \frac{\pi}{2} \\ \frac{\pi}{2} & \frac{\pi}{2} & \frac{\pi}{2} & \frac{\pi}{2} \end{bmatrix}$$

- Matrices diagonales. Son creadas mediante:

  Matrix(n, <a1, a2,..., an>, shape=diagonal)

  Matrix(1..n, <a1, a2,..., an>, shape=diagonal)

  Donde **n** determina la dimensión de la matriz y la lista incluye los elementos de la diagonal; si estos no son proporcionados, la matriz es creada con ceros. También se puede usar la función **DiagonalMatrix** para la creación de estas matrices. Por ejemplo:

```
> Matrix(3, <2, 8, 6>, shape=diagonal);
```

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

```
> DiagonalMatrix([9, 4, 5]);
```

$$\begin{bmatrix} 9 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 5 \end{bmatrix}$$

Al hacer uso de esta función no es necesario indicar la dimensión, es suficiente con proporcionar los elementos de la diagonal.

En este tipo de matrices podemos modificar posteriormente tanto los elementos de la diagonal, como los elementos fuera ésta. Por ejemplo:

```
> A := DiagonalMatrix([3, 8, 2]);
```

$$A := \begin{bmatrix} 3 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

```
> A[1, 1] := 5;
```

$$A_{1,1} := 5$$

```
> A[2, 1] := 4;
```

$$A_{2,1} := 4$$

```
> A;
```

$$\begin{bmatrix} 5 & 0 & 0 \\ 4 & 8 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

- Matrices simetricas. Pueden crearse con:

  Matrix(n, m, shape=symmetric)

  De la misma forma que en el caso anterior se pueden inicializar sus elementos durante la creación. Por ejemplo:

```
> B := Matrix(3, 3, shape=symmetric);
```

$$B := \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

```
> B[1, 2] := 4; B[1, 3] := 6;
```

$$B_{1,2} := 4$$

$$B_{1,3} := 6$$

```
> B;
```

$$\begin{bmatrix} 0 & 4 & 6 \\ 4 & 0 & 0 \\ 6 & 0 & 0 \end{bmatrix}$$

  Nótese que al modificar los elementos de esta matriz, Maple automáticamente mantiene la simetría.

- Matrices con elementos cero. Este tipo de matrices pueden ser creadas con:

  ZeroMatrix(m, n)

  Donde **m**, **n** determinan las dimensiones de la matriz. Éstas también pueden ser creadas como matrices constantes, como se vio anteriormente. Por ejemplo:

```
> Matrix(3, 4, 0);
```

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> ZeroMatrix(2, 3);
```

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

- Matrices aleatorias. Éstas pueden ser creadas por medio de la función:

  RandomMatrix(m, n)

  Por ejemplo:

```
> RandomMatrix(2, 3);
```

$$\begin{bmatrix} -50 & 62 & -71 \\ 30 & -79 & 28 \end{bmatrix}$$

- Matrices triangulares. Pueden crearse con las siguientes instrucciones:

  Matrix(n, m, [...], shape=triangular[upper])

  Matrix(n, m, [...], shape=triangular[lower])

  Donde **n**, **m** determinan las dimensiones de la matriz (no tiene que ser cuadrada), mientras que la lista determina los elementos con los cuales se inicializara ésta. Si la matriz es cuadrada, es suficiente con colocar una de las dimensiones. Por ejemplo:

```
> Matrix(3, [[2], [4, 5], [2, 8, 9]], shape=triangular[lower]);
```

$$\begin{bmatrix} 2 & 0 & 0 \\ 4 & 5 & 0 \\ 2 & 8 & 9 \end{bmatrix}$$

La función **Matrix** proporciona otras opciones para crear matrices especiales. Consúltese su página de ayuda. Adicionalmente, el paquete ***linalg*** proporciona un conjunto de funciones para creación de matrices especiales, las cuales pueden ser convertidas por medio de **convert**. Entre éstas se encuentran:

- Matriz bloque. La función **blockmatrix** de ***linalg*** nos permite crear una matriz bloque a partir de matrices previamente definidas. Su sintaxis es:

  blockmatrix(m, n, L)

  Donde **m** y **n** indican la cantidad de bloques a usar y **L** es una lista de **m*n** elementos, cada uno de los cuales es una matriz. Por ejemplo:

```
> m1 := matrix([[1, 2, 4], [8, 7, 3], [5, 2, 9]]);
```

$$m1 := \begin{bmatrix} 1 & 2 & 4 \\ 8 & 7 & 3 \\ 5 & 2 & 9 \end{bmatrix}$$

```
> m2 := matrix([[2, 4, 8], [4, 2, 6], [7, 3, 5]]);
```

$$m2 := \begin{bmatrix} 2 & 4 & 8 \\ 4 & 2 & 6 \\ 7 & 3 & 5 \end{bmatrix}$$

```
> m3 := linalg[blockmatrix](2,2,[m1,m2,m1,m2]);
```

$$m3 := \begin{bmatrix} 1 & 2 & 4 & 2 & 4 & 8 \\ 8 & 7 & 3 & 4 & 2 & 6 \\ 5 & 2 & 9 & 7 & 3 & 5 \\ 1 & 2 & 4 & 2 & 4 & 8 \\ 8 & 7 & 3 & 4 & 2 & 6 \\ 5 & 2 & 9 & 7 & 3 & 5 \end{bmatrix}$$

- Matriz bloque diagonal Este tipo de matrices pueden ser creadas por medio de la función **diag**, su sintaxis es:

  diag(B1, B2, ... Bn)

  Donde **"B1, B2, ... , Bn"** son matrices cuadradas o valores constantes. Por ejemplo:

```
> B1 := matrix([[x^2, sin(x), 4], [5, 8, 2], [3, 6, 7]]);
```

$$B1 := \begin{bmatrix} x^2 & \sin(x) & 4 \\ 5 & 8 & 2 \\ 3 & 6 & 7 \end{bmatrix}$$

```
> B2 := matrix([[1, 8, 3], [-3, 2, 6], [4, 5, 2]]);
```

$$B2 := \begin{bmatrix} 1 & 8 & 3 \\ -3 & 2 & 6 \\ 4 & 5 & 2 \end{bmatrix}$$

```
> linalg[diag](B1, Pi/2, Pi^2, B2);
```

$$\begin{bmatrix} x^2 & \sin(x) & 4 & 0 & 0 & 0 & 0 & 0 \\ 5 & 8 & 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 6 & 7 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{\pi}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \pi^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 8 & 3 \\ 0 & 0 & 0 & 0 & 0 & -3 & 2 & 6 \\ 0 & 0 & 0 & 0 & 0 & 4 & 5 & 2 \end{bmatrix}$$

El paquete **LinearAlgebra** contiene algunas funciones más para creación de matrices especiales, entre las cuales se encuentran: **BezoutMatrix**, **HilbertMatrix**, **SylvesterMatrix**, **VandermondeMatrix** y **ToeplitzMatrix**. Además, también contiene la función **BandMatrix**, la cual permite crear una matriz banda, de la siguiente forma:

```
> lista := [[w, w], [x, x, x], [y, y, y], [z, z]];
```

$$lista := [[w, w], [x, x, x], [y, y, y], [z, z]]$$

```
> LinearAlgebra[BandMatrix](lista);
```

$$\begin{bmatrix} y & z & 0 \\ x & y & z \\ w & x & y \\ 0 & w & x \end{bmatrix}$$

Otra función proporcionada por **LinearAlgebra** es **HankelMatrix**. Esta función permite crear una matriz **M** a partir de los elementos de una lista **L**, de tal forma que el elemento **M[i, j] = L[i + j - 1]**. Vease su página de ayuda para más información.

Consúltese la página de ayuda de **LinearAlgebra** y de **linalg** para una lista completa de estas funciones.

5.5. Operaciones aritméticas con matrices

En el caso de las matrices creadas a partir de **Matrix**, éstas pueden ser operadas directamente con los operadores aritméticos **'+'** y **'-'**, para suma y resta; **'*'** para producto por un escalar, y **'.'** para producto de matrices; además de **'^'** y **'**'** para multiplicar una matriz por si misma. Estos operadores también pueden ser usados en operaciones que involucren vectores, siempre que estos hayan sido creados con la función **Vector** o a partir de las funciones de **LinearAlgebra**. Por ejemplo:

```
> A := Matrix([[2, 5, 8], [8, 2, 6], [4, 2, 6]]);
```

$$A := \begin{bmatrix} 2 & 5 & 8 \\ 8 & 2 & 6 \\ 4 & 2 & 6 \end{bmatrix}$$

```
> B := Matrix([[x^2, y, 3], [4, 2, 7], [5, 3, 8]]);
```

$$B := \begin{bmatrix} x^2 & y & 3 \\ 4 & 2 & 7 \\ 5 & 3 & 8 \end{bmatrix}$$

```
> A + B;
```

$$\begin{bmatrix} 2+x^2 & 5+y & 11 \\ 12 & 4 & 13 \\ 9 & 5 & 14 \end{bmatrix}$$

```
> A.B;
```

$$\begin{bmatrix} 2\,x^2+60 & 2\,y+34 & 105 \\ 8\,x^2+38 & 8\,y+22 & 86 \\ 4\,x^2+38 & 4\,y+22 & 74 \end{bmatrix}$$

```
> A + B - 4*A.B;
```

$$\begin{bmatrix} -238-7\,x^2 & -131-7\,y & -409 \\ -140-32\,x^2 & -84-32\,y & -331 \\ -143-16\,x^2 & -83-16\,y & -282 \end{bmatrix}$$

Las siguientes expresiones pueden ser usadas para multiplicar una matriz por si misma:

```
> A^3;
```

$$\begin{bmatrix} 816 & 640 & 1388 \\ 1008 & 616 & 1456 \\ 704 & 472 & 1080 \end{bmatrix}$$

```
> B**2;
```

$$\begin{bmatrix} x^4+4\,y+15 & x^2\,y+2\,y+9 & 3\,x^2+7\,y+24 \\ 4\,x^2+43 & 4\,y+25 & 82 \\ 5\,x^2+52 & 5\,y+30 & 100 \end{bmatrix}$$

Estos operadores también nos permiten obtener la inversa de una matriz, por ejemplo:

```
> Ainv := A^(-1);
```

$$Ainv := \begin{bmatrix} 0 & \frac{1}{4} & \frac{-1}{4} \\ \frac{3}{7} & \frac{5}{14} & \frac{-13}{14} \\ \frac{-1}{7} & \frac{-2}{7} & \frac{9}{14} \end{bmatrix}$$

```
> Binv := B**(-1);
```

$$Binv := \begin{bmatrix} -\frac{5}{3y+6-5x^2} & -\frac{8y-9}{3y+6-5x^2} & \frac{7y-6}{3y+6-5x^2} \\ \frac{3}{3y+6-5x^2} & \frac{8x^2-15}{3y+6-5x^2} & -\frac{7x^2-12}{3y+6-5x^2} \\ \frac{2}{3y+6-5x^2} & \frac{-3x^2+5y}{3y+6-5x^2} & -\frac{2(-x^2+2y)}{3y+6-5x^2} \end{bmatrix}$$

Podemos comprobar que realmente son las inversas de **A** y **B**.

```
> A.Ainv;
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> Binv.B;
```

$$\begin{bmatrix} -\frac{5x^2}{3y+6-5x^2} - \frac{4(8y-9)}{3y+6-5x^2} + \frac{5(7y-6)}{3y+6-5x^2}, \\ -\frac{5y}{3y+6-5x^2} - \frac{2(8y-9)}{3y+6-5x^2} + \frac{3(7y-6)}{3y+6-5x^2}, \\ -\frac{15}{3y+6-5x^2} - \frac{7(8y-9)}{3y+6-5x^2} + \frac{8(7y-6)}{3y+6-5x^2} \end{bmatrix}$$

$$\begin{bmatrix} \frac{3x^2}{3y+6-5x^2} + \frac{4(8x^2-15)}{3y+6-5x^2} - \frac{5(7x^2-12)}{3y+6-5x^2}, \\ \frac{3y}{3y+6-5x^2} + \frac{2(8x^2-15)}{3y+6-5x^2} - \frac{3(7x^2-12)}{3y+6-5x^2}, \\ \frac{9}{3y+6-5x^2} + \frac{7(8x^2-15)}{3y+6-5x^2} - \frac{8(7x^2-12)}{3y+6-5x^2} \end{bmatrix}$$

$$\begin{bmatrix} \frac{2x^2}{3y+6-5x^2} + \frac{4(-3x^2+5y)}{3y+6-5x^2} - \frac{10(-x^2+2y)}{3y+6-5x^2}, \\ \frac{2y}{3y+6-5x^2} + \frac{2(-3x^2+5y)}{3y+6-5x^2} - \frac{6(-x^2+2y)}{3y+6-5x^2}, \\ \frac{6}{3y+6-5x^2} + \frac{7(-3x^2+5y)}{3y+6-5x^2} - \frac{16(-x^2+2y)}{3y+6-5x^2} \end{bmatrix}$$

```
> Map(simplify,%);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Adicionalmente, el paquete **LinearAlgebra** contiene las siguientes funciones que pueden ser usadas para realizar operaciones aritméticas con este tipo de matrices:

- **Add(A, B)**

  Add(A, B, c, d)

  Esta función calcula la suma de las matrices **A** y **B**. La segunda forma nos permite calcular la combinación: **c*A** + **d*B**. Estas operaciones también pueden calcularse con las funciones:

  MatrixAdd(A, B)

MattrixAdd(A, B, c, d)

Por ejemplo:

```
> MatrixAdd(A, B);
```

$$\begin{bmatrix} 2+x^2 & 5+y & 11 \\ 12 & 4 & 13 \\ 9 & 5 & 14 \end{bmatrix}$$

- **Multiply(A, B)**

  Nos permite calcular el producto de las matrices **A** y **B**; también puede ser usada para calcular un producto de matrices con vectores. Esta misma operación puede ser calculada también con la función:

  MatrixMatrixMultiply(A, B)

  Por ejemplo:

```
> MatrixMatrixMultiply(A, B);
```

$$\begin{bmatrix} 2\,x^2+60 & 2\,y+34 & 105 \\ 8\,x^2+38 & 8\,y+22 & 86 \\ 4\,x^2+38 & 4\,y+22 & 74 \end{bmatrix}$$

- **ScalarMultiply(A, c)**

  Esta función calcula el producto de la matriz **A** con el escalar **c**. Tambien puede calcularse por medio de la función:

  MatrixScalarMultiply(A, c)

- **MatrixVectorMultiply(A, v)**

  Calcula el producto de la matriz **A** con el vector **v**.

5.6. Funciones de LinearAlgebra para operaciones con matrices

El paquete **LinearAlgebra** proporciona funciones para realizar diversos cálculos con matrices; entre otras podemos mencionar las que se decriben a continuación.

5.6.1. Dimensiones de una matriz

Las funciones **RowDimension** y **ColumnDimension** de **LinearAlgebra** nos permiten determinar el número de renglones y columnas, respectivamente, de una matriz. Por otro lado, la función **Dimension**, nos puede devolver ambas dimensiones en forma de secuencia. Por ejemplo:

```
> A := RandomMatrix(7, 5);
```

$$A := \begin{bmatrix} -98 & 59 & 97 & -65 & 16 \\ -41 & -69 & -82 & 5 & -34 \\ 24 & 23 & -66 & 66 & -62 \\ 65 & 25 & 55 & -36 & -90 \\ 1 & 5 & 68 & -41 & -21 \\ -42 & -75 & 26 & 20 & -56 \\ -82 & 38 & 13 & -7 & -8 \end{bmatrix}$$

```
> RowDimension(A);
```

```
> ColumnDimension(A);
```

$$5$$

```
> Dimension(A);
```

$$7, 5$$

5.6.2. Traza de una matriz

Ésta puede ser calculada mediante la función: **Trace(M)**, donde **M** debe ser una matriz cuadrada. Por ejemplo:

```
> m := Matrix([[1, 2, 3], [4, Pi, 5], [Pi^2, 8, exp(1.1)]]);
```

$$m := \begin{bmatrix} 1 & 2 & 3 \\ 4 & \pi & 5 \\ \pi^2 & 8 & 3{,}004166024 \end{bmatrix}$$

```
> Trace(m);
```

$$4{,}004166024 + \pi$$

Podemos aprovechar la capacidad que tiene Maple de manipular expresiones simbólicas para verificar el siguiente resultado:

Dadas dos matrices **A**, **B** de n x n, y **a**, **b** escalares, se cumple que:

traza(a*A + b*B) = a*traza(A) + b*traza(B)

Verifiquemos esto para matrices simbólicas de 4 x 4.

```
> A := Matrix(4, 4, [[A11, A12, A13, A14], [A21, A22, A23, A24],
> [A31, A32, A33, A34], [A41, A42, A43, A44]]);
```

$$A := \begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \\ A41 & A42 & A43 & A44 \end{bmatrix}$$

```
> B := Matrix(4, 4, [[B11, B12, B13, B14], [B21, B22, B23, B24],
> [B31, B32, B33, B34], [B41, B42, B43, B44]]);
```

$$B := \begin{bmatrix} B11 & B12 & B13 & B14 \\ B21 & B22 & B23 & B24 \\ B31 & B32 & B33 & B34 \\ B41 & B42 & B43 & B44 \end{bmatrix}$$

```
> r1 := Trace(a*A + b*B);
```

$$r1 := b\,B11 + a\,A11 + b\,B22 + a\,A22 + b\,B33 + a\,A33 + b\,B44 + a\,A44$$

```
> r2 := expand(a*Trace(A) + b*Trace(B));
```

$$r2 := b\,B11 + a\,A11 + b\,B22 + a\,A22 + b\,B33 + a\,A33 + b\,B44 + a\,A44$$

Una vez realizados los cálculos, podemos comprobar que la igualdad se cumple usando la función **evalb**.

```
> evalb(r1 = r2);
```

$$true$$

5.6.3. Transpuesta de una matriz

La transpuesta de una matriz puede calcularse con la función **Transpose**. Por ejemplo:

```
> n := Matrix([[4, Pi, 5, x^2], [Pi^2, 8, exp(1.1), x^3]]);
```

$$n := \begin{bmatrix} 4 & \pi & 5 & x^2 \\ \pi^2 & 8 & 3{,}004166024 & x^3 \end{bmatrix}$$

```
> Transpose(n);
```

$$\begin{bmatrix} 4 & \pi^2 \\ \pi & 8 \\ 5 & 3{,}004166024 \\ x^2 & x^3 \end{bmatrix}$$

Consideremos las siguientes proposiciones:

transpuesta(a*A + b*B) = a*transpuesta(A) + b*transpuesta(B)

transpuesta(transpuesta(A)) = A

Donde **A**, **B** son matrices de m x n, y **a**, **b** son escalares. Verifiquemos esto para matrices simbólicas de dimensión 3 x 4.

```
> A := Matrix(3, 4, [[A11, A12, A13, A14], [A21, A22, A23, A24],
> [A31, A32, A33, A34]]);
```

$$A := \begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \end{bmatrix}$$

```
> B := Matrix(3, 4, [[B11, B12, B13, B14], [B21, B22, B23, B24],
> [B31, B32, B33, B34]]);
```

$$B := \begin{bmatrix} B11 & B12 & B13 & B14 \\ B21 & B22 & B23 & B24 \\ B31 & B32 & B33 & B34 \end{bmatrix}$$

Utilizaremos la función **equal** de **linalg** para determinar la igualdad de las matrices resultantes.

```
> r1 := Transpose(c*A + d*B);
```

$$r1 := \begin{bmatrix} d\,B11 + c\,A11 & d\,B21 + c\,A21 & d\,B31 + c\,A31 \\ d\,B12 + c\,A12 & d\,B22 + c\,A22 & d\,B32 + c\,A32 \\ d\,B13 + c\,A13 & d\,B23 + c\,A23 & d\,B33 + c\,A33 \\ d\,B14 + c\,A14 & d\,B24 + c\,A24 & d\,B34 + c\,A34 \end{bmatrix}$$

```
> r2 := c*Transpose(A) + d*Transpose(B);
```

$$r2 := c \begin{bmatrix} A11 & A21 & A31 \\ A12 & A22 & A32 \\ A13 & A23 & A33 \\ A14 & A24 & A34 \end{bmatrix} + d \begin{bmatrix} B11 & B21 & B31 \\ B12 & B22 & B32 \\ B13 & B23 & B33 \\ B14 & B24 & B34 \end{bmatrix}$$

```
> Equal(r1, r2);
```

true

Por lo tanto, para este tipo de matrices la proposición es verdadera. Verifiquemos ahora la segunda.

```
> Att := Transpose(Transpose(A));
```

$$Att := \begin{bmatrix} A11 & A12 & A13 & A14 \\ A21 & A22 & A23 & A24 \\ A31 & A32 & A33 & A34 \end{bmatrix}$$

```
> Equal(Att, A);
```

$$true$$

Por lo tanto hemos verificado que, para este tipo de matrices, se cumple la segunda proposición.

5.6.4. Cálculo de bases para espacios vectoriales y combinaciones lineales

El paquete **LinearAlgebra** contiene la función **Basis(V)**, la cual nos permite encontrar una base para un espacio vectorial. El argumento **V** puede ser un vector, un conjunto de vectores o una lista de vectores. Por ejemplo:

```
> vecs := [ <1 | 0 | 0>, <1 | 1 | 0>, <1 | 1 | 1>, <1 | 2 | 3>];
```

$$vecs := [[1, 0, 0], [1, 1, 0], [1, 1, 1], [1, 2, 3]]$$

```
> Basis(vecs);
```

$$[[1, 0, 0], [1, 1, 0], [1, 1, 1]]$$

Nótese que **Basis** nos devuelve únicamente los tres vectores linealmente independientes. Veamos el siguiente caso:

```
> v1 := <2 | -4 | 1>;
```

$$v1 := [2, -4, 1]$$

```
> v2 := <0 | 3 | -1>;
```

$$v2 := [0, 3, -1]$$

```
> v3 := <6 | 0 | -1>;
```

$$v3 := [6, 0, -1]$$

```
> lb := [v1, v2, v3];
```

$$lb := [[2, -4, 1], [0, 3, -1], [6, 0, -1]]$$

```
> Basis(lb);
```

$$[[2, -4, 1], [0, 3, -1]]$$

Esta función solo nos devuelve dos de los vectores, lo cual indica que el tercero es combinación lineal de los otros dos, y por lo tanto no pueden formar una base. Verifiquemos esto:

Debemos encontrar dos escalares **c** y **d** tales que **v3** = **c*v1** + **d*v2**.

```
> ec := v3 = VectorAdd(v1, v2, c, d);
```

$$ec := [6, 0, -1] = [2\,c, -4\,c + 3\,d, c - d]$$

Para encontrar **c** y **d** debemos resolver el siguiente sistema de ecuaciones:

```
> res := solve({2*c = 6, -4*c + 3*d = 0, c - d = -1});
```

$$res := \{c = 3, d = 4\}$$

Comprobemos el resultado obtenido:

```
> assign(res[1], res[2]);
> v3 = c*v1 + d*v2;
```

$$[6, 0, -1] = [6, 0, -1]$$

Por lo tanto **v3** es combinación lineal de **v1** y **v2**, y como consecuencia **lb** no puede ser una base. Consideremos ahora el siguiente ejemplo:

```
> v4 := <1 | 0 | -1>;
```

$$v4 := [1, 0, -1]$$

```
> v5 := <2 | 5 | 1>;
```

$$v5 := [2, 5, 1]$$

```
> v6 := <0 | -4 | 3>;
```

$$v6 := [0, -4, 3]$$

```
> lb2 := [v4, v5, v6];
```

$$lb2 := [[1, 0, -1], [2, 5, 1], [0, -4, 3]]$$

```
> Basis(lb2);
```

$$[[1, 0, -1], [2, 5, 1], [0, -4, 3]]$$

En este caso, **Basis** nos devuelve los mismos tres vectores, lo cual indica que generan el espacio de vectores de tres dimensiones y que son linealmente independientes. Verifiquemos la independencia lineal.

Sean:

```
> ec1 := v4 = VectorAdd(v5, v6, g, h);
```

$$ec1 := [1, 0, -1] = [2\,g, 5\,g - 4\,h, g + 3\,h]$$

```
> ec2 := v5 = VectorAdd(v4, v6, j, k);
```

$$ec2 := [2, 5, 1] = [j, -4\,k, -j + 3\,k]$$

```
> ec3 := v6 = VectorAdd(v4, v5, a, b);
```

$$ec3 := [0, -4, 3] = [a + 2\,b, 5\,b, -a + b]$$

Resolvemos el siguiente sistema para verificar si **v4** es combinación lineal de **v5** y **v6**.

```
> solve({2*g = 1, 5*g - 4*h = 0, g + 3*h = -1});
```

No obtenemos ningun resultado, lo cual indica que el sistema no tiene solución, y por lo tanto **v4** no es combinación lineal de **v5** y **v6**.

De la misma forma podemos comprobar para los vectores **v5** y **v6**.

```
> solve({j = 2, -4*k = 5, -j + 3*k = 1});
```

```
> solve({a + 2*b = 0, 5*b = -4, -a + b = 3});
```

Ahora, verifiquemos si **v4**, **v5** y **v6** generan cualquier vector de tres dimensiones.

```
> vx := <x | y | z>;
```

$$vx := [x, y, z]$$

Debemos encontrar escalares **a**, **b**, **h**, tales que: **vx** = **a*v4** + **b*v5** + **h*v6**.

```
> rel := vx = VectorScalarMultiply(v4, a) + VectorScalarMultiply(v5, b)
> + VectorScalarMultiply(v6, h);
```

$$rel := [x, y, z] = [a + 2\,b, 5\,b - 4\,h, -a + b + 3\,h]$$

```
> solve({a + 2*b = x, 5*b - 4*h = y, -a + b + 3*h = z}, {a, b, h});
```

$$\{a = \frac{19\,x}{27} - \frac{8\,z}{27} - \frac{2\,y}{9}, b = \frac{4\,x}{27} + \frac{4\,z}{27} + \frac{y}{9}, h = \frac{5\,x}{27} + \frac{5\,z}{27} - \frac{y}{9}\}$$

Este resultado comprueba que cualquier vector de tres dimensiones puede ser escrito como combinación lineal de **v4**, **v5** y **v6**; es decir, generan cualquier vector de tres dimensiones. Por lo tanto, hemos comprobado que **lb2** es una base.

Finalmente, calcularemos una base para el espacio formado por las columnas y los renglones de la siguiente matriz. Esto puede obtenerse por medio de las funciones **RowSpace** y **ColumnSpace**.

```
> m := Matrix([[2, 3, 5], [1, 0, 1], [0, 1, 1]]);
```

$$m := \begin{bmatrix} 2 & 3 & 5 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

```
> RowSpace(m);
```

$$[[1, 0, 1], [0, 1, 1]]$$

```
> ColumnSpace(m);
```

$$\left[\begin{bmatrix} 1 \\ 0 \\ 1 \\ \frac{1}{3} \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \frac{-2}{3} \end{bmatrix} \right]$$

Otras dos funciones útiles para este tipo de operaciones son las siguientes:

- **SumBasis**. Calcula una base para la suma directa de espacios vectoriales.
- **IntersectionBasis**. Calcula una base para la intersección de espacios vectoriales.

5.6.5. Bases para el espacio nulo y rango de una transformación lineal

La función **NullSpace(A)** nos permite calcular una base para el espacio nulo (kernel) de una transformación lineal definida por la matriz **A**.

Consideremos la transformacion **T**, de los polinómios de grado tres con coeficientes reales sobre los polinómios de grado dos con coeficientes reales, dada por:

$$T : P^3(R) \longrightarrow P^2(R),\ T(p) = p'$$

Donde **p** es un polinomio de grado tres con coeficientes reales. Esta trasformación está representada por la siguiente matriz:

```
> A := Matrix([[0,1,0,0],[0,0,2,0],[0,0,0,3]]);
```

$$A := \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

Para comprobar esto consideremos el siguiente polinómio:

```
> p := a + b*x + c*x^2 + d*x^3;
```

$$p := a + b\,x + 3\,x^2 + 4\,x^3$$

Definimos la transformación:

```
> T := f -> diff(f, x);
```

$$T := f \to \tfrac{d}{dx} f$$

Calculamos **T(p)**:

```
> T(p);
```

$$b + 6\,x + 12\,x^2$$

El polinomio **p** esta representado por el vector:

```
> v := <x, y, z, w>;
```

$$v := \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

Ahora aplicamos la transformación multiplicando por la matriz:

```
> vt := Multiply(A, v);
```

$$vt := \begin{bmatrix} y \\ 2\,z \\ 3\,w \end{bmatrix}$$

Por otro lado, una base para $P^2(R)$ es:

```
> bas := <1 | x | x^2>;
```

$$bas := [\,1,\, x,\, x^2\,]$$

Multiplicamos el vector **vt**, al que se le aplicó la transformación por medio de la matriz, por esta base.

```
> Multiply(bas, vt);
```

$$y + 2\,z\,x + 3\,w\,x^2$$

De este resultado podemos ver que **A** representa la transformación **T**. Ahora, verifiquemos si **T** es lineal. Debemos comprobar que

T(k*p + q) = k*T(p) + T(q)

Para **p**, **q** polinomios de grado tres y **n** un escalar.

Sean:

```
> p := a1 + b1*x + c1*x^2 + d1*x^3;
```

$$p := a1 + b1\,x + c1\,x^2 + d1\,x^3$$

```
> q := a2 + b2*x + c2*x^2 + d2*x^3;
```

$$q := a2 + b2\,x + c2\,x^2 + d2\,x^3$$

```
> T(k*p + q);
```

$$k\,(b1 + 2\,c1\,x + 3\,d1\,x^2) + b2 + 2\,c2\,x + 3\,d2\,x^2$$

```
> k*T(p) + T(q);
```

$$k\,(b1 + 2\,c1\,x + 3\,d1\,x^2) + b2 + 2\,c2\,x + 3\,d2\,x^2$$

Por lo tanto, **T** es una transformación lineal. Ahora, calcularemos una base para el espacio nulo de **T**, así como la dimensión de este espacio.

```
> NullSpace(A);
```

$$\left\{ \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \right\}$$

Esta base es obvia ya que la derivada de un polinomio es cero solo si éste es una constante. Por otro lado, es obvio que la dimensión de este espacio nulo es 1. Además, si **B** es una base para **P3(R)**.

```
> B := 1 + x + x^2 + x^3;
```

$$B := 1 + x + x^2 + x^3$$

Entonces **T(B)** es una base para el rango de **T**.

```
> T(B);
```

$$1 + 2\,x + 3\,x^2$$

Como podemos ver, la dimensión de esta base es **'3'**, es decir:

dim(P3(R)) = dim(kernel(T)) + dim(rango(T)) = 1 + 3 = 4

Este paquete contiene la función **Rank**, la cual nos permite calcular el rango de una matriz, haciendo eliminación Gaussiana sobre los renglones de ésta. Podemos usar dicha función para determinar la dimensión del rango de **T**.

```
> Rank(A);
```

3

5.6.6. Inversa de una matriz

La inversa de una matriz A de n x n, esta definida como una matriz $A^{(-1)}$, de n x n, tal que:

$$A\,A^{(-1)} = A^{(-1)}\,A = I$$

Una matriz que tiene inversa se dice que es invertible.

El paquete **LinearAlgebra** nos proporciona la función **MatrixInverse**, la cual calcula la inversa de una matriz. Por ejemplo:

```
> A := Matrix([ [1, 4, 5], [2, 8, 3], [3, 6, 9]]);
```

$$A := \begin{bmatrix} 1 & 4 & 5 \\ 2 & 8 & 3 \\ 3 & 6 & 9 \end{bmatrix}$$

```
> Ainv := MatrixInverse(A);
```

$$Ainv := \begin{bmatrix} \frac{-9}{7} & \frac{1}{7} & \frac{2}{3} \\ \frac{3}{14} & \frac{1}{7} & \frac{-1}{6} \\ \frac{2}{7} & \frac{-1}{7} & 0 \end{bmatrix}$$

Verifiquemos este resultado:

```
> A.Ainv;
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> Ainv.A;
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consideremos la siguiente matriz y verifiquemos que tiene inversa:

```
> B := Matrix([[2, 9, 3], [1, 8, 4], [3, 6, 8]]);
```

$$B := \begin{bmatrix} 2 & 9 & 3 \\ 1 & 8 & 4 \\ 3 & 6 & 8 \end{bmatrix}$$

```
> Binv := MatrixInverse(B);
```

$$Binv := \begin{bmatrix} \frac{20}{31} & \frac{-27}{31} & \frac{6}{31} \\ \frac{2}{31} & \frac{7}{62} & \frac{-5}{62} \\ \frac{-9}{31} & \frac{15}{62} & \frac{7}{62} \end{bmatrix}$$

A continuación, para las matrices particulares **A** y **B**, verificaremos que: **A*B** es invertible. Además, verificaremos que se cumple la siguiente relación:

$$(A\,B)^{(-1)} = B^{(-1)}\,A^{(-1)}$$

```
> MatrixInverse(A.B);
```

$$\begin{bmatrix} \frac{-417}{434} & \frac{-13}{217} & \frac{107}{186} \\ \frac{-71}{868} & \frac{8}{217} & \frac{3}{124} \\ \frac{397}{868} & \frac{-5}{217} & \frac{-29}{124} \end{bmatrix}$$

Dado que la inversa de **A*B** existe, entonces **A*B** es invertible.

Verifiquemos ahora la relación.

```
> ABinv := MatrixInverse(A.B);
```

$$ABinv := \begin{bmatrix} \frac{-417}{434} & \frac{-13}{217} & \frac{107}{186} \\ \frac{-71}{868} & \frac{8}{217} & \frac{3}{124} \\ \frac{397}{868} & \frac{-5}{217} & \frac{-29}{124} \end{bmatrix}$$

```
> BinvAinv := MatrixInverse(B).MatrixInverse(A);
```

$$BinvAinv := \begin{bmatrix} \frac{-417}{434} & \frac{-13}{217} & \frac{107}{186} \\ \frac{-71}{868} & \frac{8}{217} & \frac{3}{124} \\ \frac{397}{868} & \frac{-5}{217} & \frac{-29}{124} \end{bmatrix}$$

Comprobaremos si **ABinv** = **BinvAinv**:

```
> Equal(ABinv, BinvAinv);
```

true

Por lo tanto la relación se cumple (al menos para este caso particular).

5.6.7. Extracción de renglones, columnas y submatrices

Este paquete también contiene funciones útiles para extraer columnas, renglones y submatrices de una matriz dada. Consideremos la siguiente matriz:

```
> A := RandomMatrix(6, 6);
```

$$A := \begin{bmatrix} -19 & -70 & -45 & -12 & -66 & 61 \\ 51 & -44 & 72 & 98 & -11 & -70 \\ -35 & -48 & 22 & 56 & 35 & 22 \\ -52 & -52 & -34 & -54 & 61 & -81 \\ 71 & 37 & 69 & -88 & 96 & -99 \\ 89 & 58 & 55 & -90 & 53 & -2 \end{bmatrix}$$

Para extraer un renglón o un conjunto de renglones usamos la función:

Row(A, i)

Row(A, i..j)

En el primer caso se extrae el i-esimo renglón; mientras que la segunda forma extrae los renglones i-esimo al j-esimo. Si se desea extraer renglones no consecutivos, estos deben incluirse en forma de una lista, por ejemplo: **Row(A, [2, 4, 7..11])**. Veamos el siguiente caso.

```
> Row(A, 4);
```

$$[-52, -52, -34, -54, 61, -81]$$

```
> Row(A, [1, 3..5]);
```

$$[-19, -70, -45, -12, -66, 61], [-35, -48, 22, 56, 35, 22], [-52, -52, -34, -54, 61, -81], [71, 37, 69, -88, 96, -99]$$

De la misma forma, podemos usar las siguientes funciones para extraer columnas:

Column(A, i)

Column(A, i..j)

La primera forma extrae la i-esima columna, mientras que la segunda extrae de la i-esima a la j-esima. Por ejemplo:

```
> Column(A, 2);
```

$$\begin{bmatrix} -70 \\ -44 \\ -48 \\ -52 \\ 37 \\ 58 \end{bmatrix}$$

```
> Column(A, [1, 3..5]);
```

$$\begin{bmatrix} -19 \\ 51 \\ -35 \\ -52 \\ 71 \\ 89 \end{bmatrix}, \begin{bmatrix} -45 \\ 72 \\ 22 \\ -34 \\ 69 \\ 55 \end{bmatrix}, \begin{bmatrix} -12 \\ 98 \\ 56 \\ -54 \\ -88 \\ -90 \end{bmatrix}, \begin{bmatrix} -66 \\ -11 \\ 35 \\ 61 \\ 96 \\ 53 \end{bmatrix}$$

Otra de las funciones útiles para este fin es:

SubMatrix(A, r, c)

Esta función nos permite extraer la submatriz de $\boldsymbol{A}$ determinada por los renglones **r** y las columnas **c**. Estos parametros pueden estar dados por un número, un rango o una lista de números y rangos (de la misma forma que en **Row** y **Column**). Por ejemplo:

```
> SubMatrix(A, [1..3, 5], 1..4);
```

$$\begin{bmatrix} -19 & -70 & -45 & -12 \\ 51 & -44 & 72 & 98 \\ -35 & -48 & 22 & 56 \\ 71 & 37 & 69 & -88 \end{bmatrix}$$

Otra manera de generar una submatriz es eliminando columnas y/o renglones de la matriz original. Para este tipo de operaciones se pueden usar las funciones:

DeleteRow(A, r)

DeleteColumn(A, c)

Donde **r** y **c** pueden estar dadas en la misma forma que en el caso de **SubMatrix**. Estas funciones generan una matriz a partir de **A**, en la cual se han eliminado los renglones o columnas indicadas por el segundo argumento. Por ejemplo:

```
> DeleteRow(A, 2..4);
```

$$\begin{bmatrix} -19 & -70 & -45 & -12 & -66 & 61 \\ 71 & 37 & 69 & -88 & 96 & -99 \\ 89 & 58 & 55 & -90 & 53 & -2 \end{bmatrix}$$

```
> DeleteColumn(A, 1..3);
```

$$\begin{bmatrix} -12 & -66 & 61 \\ 98 & -11 & -70 \\ 56 & 35 & 22 \\ -54 & 61 & -81 \\ -88 & 96 & -99 \\ -90 & 53 & -2 \end{bmatrix}$$

```
> DeleteColumn(A, 3);
```

$$\begin{bmatrix} -19 & -70 & -12 & -66 & 61 \\ 51 & -44 & 98 & -11 & -70 \\ -35 & -48 & 56 & 35 & 22 \\ -52 & -52 & -54 & 61 & -81 \\ 71 & 37 & -88 & 96 & -99 \\ 89 & 58 & -90 & 53 & -2 \end{bmatrix}$$

5.6.8. Unión de matrices y matrices aumentadas

La función **Matrix**, en las formas:

Matrix([A, B, ...])

Matrix(n, m, [A, B, ...])

nos permite unir horizontalmente dos o más matrices. Por ejemplo:

```
> A := RandomMatrix(3, 3);
```

$$A := \begin{bmatrix} 78 & 32 & 14 \\ 72 & -23 & 42 \\ 50 & -99 & -68 \end{bmatrix}$$

```
> B := RandomMatrix(3, 4);
```

$$B := \begin{bmatrix} 22 & -84 & 84 & -52 \\ -53 & 42 & -79 & 9 \\ 45 & 33 & -30 & -43 \end{bmatrix}$$

```
> C := RandomMatrix(3, 3);
```

$$C := \begin{bmatrix} 72 & 42 & 77 \\ -78 & -95 & 79 \\ -13 & 29 & 47 \end{bmatrix}$$

```
> Matrix([A, B, C]);
```

$$\begin{bmatrix} 78 & 32 & 14 & 22 & -84 & 84 & -52 & 72 & 42 & 77 \\ 72 & -23 & 42 & -53 & 42 & -79 & 9 & -78 & -95 & 79 \\ 50 & -99 & -68 & 45 & 33 & -30 & -43 & -13 & 29 & 47 \end{bmatrix}$$

El paquete **linalg** contiene la función **stackmatrix**, la cual une las matrices pero verticalmente. Este paquete también proporciona la función **extend**, la cual puede extender una matriz en **m** renglones y **n** columnas.

5.6.9. Operaciones elementales con matrices

LinearAlgebra también proporciona funciones que nos permiten realizar operaciones elementales sobre matrices. Entre ellas tenemos las que se muestran a continuación.

Utilizaremos la siguiente matriz para llevar a cabo los ejemplos:

```
> A := Matrix([[1, 2, 3], [2, -1, 1], [4, 6, 3]]);
```

$$A := \begin{bmatrix} 1 & 2 & 3 \\ 2 & -1 & 1 \\ 4 & 6 & 3 \end{bmatrix}$$

- Intercambio de dos renglones (o columnas) cualesquiera de **A**. Esto puede hacerse por medio de las funciones:

  RowOperation(A, [r1, r2])

  ColumnOperation(A, [c1, c2])

  Éstas generan una matriz basada en **A**, en la cual se han intercambiado los renglones o columnas indicadas en la lista.

  ```
  > RowOperation(A, [1, 3]);
  ```

  $$\begin{bmatrix} 4 & 6 & 3 \\ 2 & -1 & 1 \\ 1 & 2 & 3 \end{bmatrix}$$

- Multiplicación de cualquier renglón (o columna) de **A** por una expresión algebraica. Este puede llevarse a cabo por medio de las funciones:

  RowOperation(A, r, exp)

  ColumnOperation(A, c, exp)

  Éstas funciones generan una matriz a partir de **A**, en la cual se ha multiplicado la columna o renglón indicados por **r** o **c** (según sea el caso) por la expresión dada como tercer argumento.

  ```
  > RowOperation(A, 2, 3*x);
  ```

  $$\begin{bmatrix} 1 & 2 & 3 \\ 6\,x & -3\,x & 3\,x \\ 4 & 6 & 3 \end{bmatrix}$$

```
> ColumnOperation(A, 1, Pi/2);
```

$$\begin{bmatrix} \frac{\pi}{2} & 2 & 3 \\ \pi & -1 & 1 \\ 2\,\pi & 6 & 3 \end{bmatrix}$$

- Suma de un múltiplo de un renglón o columna a otro renglón o columna. Esta operación puede realizarse por medio de las funciones:

  RowOperation(A, [r1, r2], expr)

  ColumnOperation(A, [c1, c2], expr)

  La primera función devuelve una matriz basada en **A**, en la cual el renglón **r1** ha sido reemplazado por la expresión:

  Row(A, r1) + expr*Row(A, r2)

  De la misma forma, la segunda función devuelve una matriz basada en **A**, en la que la columna **c1** ha sido reemplazada por:

  Column(A, c1) + expr*Column(A, c2)

  Por ejemplo, la siguiente instrucción genera una matriz a partir de **A**, en la que el renglón 1 es reemplazado por el renglón 3 multiplicado por 1/8 y sumado al renglón 1.

```
> RowOperation(A, [1, 3], 1/8);
```

$$\begin{bmatrix} \frac{3}{2} & \frac{11}{4} & \frac{27}{8} \\ 2 & -1 & 1 \\ 4 & 6 & 3 \end{bmatrix}$$

5.6.10. Solución de sistemas de ecuaciones lineales

Consideremos el siguiente sistema de m ecuaciones lineales con n incognitas:

$$\begin{cases} a_{11}\,x_1 + a_{12}\,x_2 + a_{13}\,x_3 + \cdots + a_{1n}\,x_n = b_1 \\ a_{21}\,x_1 + a_{22}\,x_2 + a_{23}\,x_3 + \cdots + a_{2n}\,x_n = b_2 \\ \quad\vdots \qquad\qquad\qquad \vdots \qquad\qquad\qquad \vdots \\ a_{m1}\,x_1 + a_{m2}\,x_2 + a_{m3}\,x_3 + \cdots + a_{mn}\,x_n = b_m \end{cases}$$

Donde a_{ij} y b_i son escalares, para $1 \leq i \leq m$ y $1 \leq j \leq n$.
La matriz de m x n:

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix}$$

se le llama *"matriz de coeficientes del sistema"*.
Por otro lado, consideremos los siguientes vectores:

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

El sistema de ecuaciones de arriba lo podemos escribir como:

$$AX = B$$

Una forma de resolver este sistema es por medio de la inversa de la matriz B. En caso de que esta matriz exista, entonces la solución del sistema es única y está dada por:

$$X = A^{(-1)}\, B$$

Por ejemplo, consideremos el siguiente sistema de ecuaciones.

```
> ec1 := x1 + 2*x2 - x3 = 5;
```

$$ec1 := x1 + 2\,x2 - x3 = 5$$

```
> ec2 := x1 + x2 + x3 = 1;
```

$$ec2 := x1 + x2 + x3 = 1$$

```
> ec3 := 2*x1 - 2*x2 + x3 = 4;
```

$$ec3 := 2\,x1 - 2\,x2 + x3 = 4$$

Para crear la matriz de coeficientes podemos usar la función **GenerateMatrix** de **LinearAlgebra**. Su sintaxis es:

GenerateMatrix([eq1, eq2, ..., eqn], [x1, x2, ..., xn])

Donde **eqn** es de la forma:

a*x1 + b*x2 + ... + n*xn

En esta expresión, **"x1, x2, ..., xn"** son las variables de las cuales dependen las ecuaciones. Ésta función nos devuelve una solución de la forma:

M, B

Donde **M** es la matrix de coeficientes del sistema, mientras que **B** es el vector con los términos independientes. Si se incluye la opción **augmented=true**, el resultado es una matriz aumentada, cuya última columna está formada por los términos independientes.

Por medio de esta función podemos crear la matriz de coeficientes y al mismo tiempo el vector con los términos independientes, de la siguiente manera:

```
> (A, B) := GenerateMatrix([ec1, ec2, ec3], [x1, x2, x3]);
```

$$A,\, B := \begin{bmatrix} 1 & 2 & -1 \\ 1 & 1 & 1 \\ 2 & -2 & 1 \end{bmatrix}, \begin{bmatrix} 5 \\ 1 \\ 4 \end{bmatrix}$$

Ahora creamos el vector **X**:

```
> X := Vector([x1, x2, x3]);
```

$$X := \begin{bmatrix} x1 \\ x2 \\ x3 \end{bmatrix}$$

Primero verificamos si **A** tiene inversa:

```
> Ainv := MatrixInverse(A);
```

$$Ainv := \begin{bmatrix} \frac{1}{3} & 0 & \frac{1}{3} \\ \frac{1}{9} & \frac{1}{3} & \frac{-2}{9} \\ \frac{-4}{9} & \frac{2}{3} & \frac{-1}{9} \end{bmatrix}$$

Entonces, el sistema tiene una solución única, la cual esta dada por

```
> sol := Ainv.B;
```

$$sol := \begin{bmatrix} 3 \\ 0 \\ -2 \end{bmatrix}$$

Para confirmar que esta es la solución debemos verificar que se cumple la relación **A.sol = B**.

```
> A.sol = B;
```

$$\begin{bmatrix} 5 \\ 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 4 \end{bmatrix}$$

Otra función útil para estos casos es **geneqs**, su sintaxis es de la siguiente forma:

GenerateEquations(A, [x1, x2, ... , xn])

GenerateEquations(A, [x1, x2, ... , xn], B)

Donde ***A*** es una matriz de coeficientes, **x1, x2, ... xn** son las variables de las cuales depende el sistema y ***B*** es el vector con los términos independientes. Esta función genera las ecuaciones correspondientes al sistema representado por la matriz de coeficientes.

Otra forma en la que podemos obtener esta solución es por medio de la función **LinearSolve**, su sintaxis es:

LinearSolve(A)

LinearSolve(A, B)

Donde **A** es una matriz de coeficientes y **B** es una matriz o un vector columna (con los términos independientes). Por ejemplo, resolvamos por este método el siguiente sistema:

```
> ec4 := x1 + 2*x2 - 3*x3 = -2;
```

$$ec4 := x1 + 2\,x2 - 3\,x3 = -2$$

```
> ec5 := x1 + x2 + x3 = -2;
```

$$ec5 := x1 + x2 + x3 = -2$$

```
> ec6 := x1 + x2 - x3 = 0;
```

$$ec6 := x1 + x2 - x3 = 0$$

```
> (A, B) := GenerateMatrix([ec4, ec5, ec6], [x1, x2, x3]);
```

$$A, B := \begin{bmatrix} 1 & 2 & -3 \\ 1 & 1 & 1 \\ 1 & 1 & -1 \end{bmatrix}, \begin{bmatrix} -2 \\ -2 \\ 0 \end{bmatrix}$$

La solución está dada por.

```
> LinearSolve(A, B);
```

$$\begin{bmatrix} 3 \\ -4 \\ -1 \end{bmatrix}$$

5.6.11. Matrices escalonadas

Otro método para resolver sistemas de ecuaciones lineales es obteniendo una matriz escalonada a partir de la matriz de coeficientes. **LinearAlgebra** proporciona varias funciones para escalonar matrices por distintos métodos. Por ejemplo, resolvamos el siguiente sistema:

```
> ec7 := 2*x1 + x2 - 3*x3 = 3;
```

$$ec7 := 2\,x1 + x2 - 3\,x3 = 3$$

```
> ec8 := x1 + 3*x2 + x3 = 2;
```

$$ec8 := x1 + 3\,x2 + x3 = 2$$

```
> ec9 := 3*x1 + x2 - x3 = 1;
```

$$ec9 := 3\,x1 + x2 - x3 = 1$$

La función **GaussianElimination** nos permite obtener una matriz escalonada por el método de Eliminación Gaussiana, para la matriz de coeficientes del sistema. Una vez obtenida esta matriz escalonada, la cual llamaremos **A**, podemos obtener las soluciones del sistema $\mathbf{A}\ \mathbf{X} = \mathbf{B}$, por medio de la función:

BackwardSubstitute(A, B)

donde **A** es la matriz de coeficientes y **B** es el vector con los términos independientes. Esta función también puede ser invocada en la forma:

BackwardSubstitute(A)

donde **A** es la matriz aumentada del sistema.

Aplicaremos estas funciones para resolver el sistema planteado.

GaussianElimination nos permite obtener una forma escalonada para la matriz de coeficientes del sistema, así como para la matriz aumentada. En este caso usaremos la matriz aumentada, la cual podemos generar con **GenerateMatrix**.

```
> A := GenerateMatrix([ec7, ec8, ec9], [x1, x2, x3], augmented=true);
```

$$A := \begin{bmatrix} 2 & 1 & -3 & 3 \\ 1 & 3 & 1 & 2 \\ 3 & 1 & -1 & 1 \end{bmatrix}$$

Aplicamos el método de Eliminación Gaussiana para obtener la matriz escalonada:

```
> E := GaussianElimination(A);
```

$$E := \begin{bmatrix} 2 & 1 & -3 & 3 \\ 0 & \frac{5}{2} & \frac{5}{2} & \frac{1}{2} \\ 0 & 0 & 4 & \frac{-17}{5} \end{bmatrix}$$

Y despues usamos **BackwardSubstitute** para resolver $\mathbf{A}\ \mathbf{X} = \mathbf{B}$.

```
> BackwardSubstitute(E);
```

$$\begin{bmatrix} \frac{-3}{10} \\ \frac{21}{20} \\ \frac{-17}{20} \end{bmatrix}$$

Por lo tanto la solución es: $x1 = -\frac{3}{10}$, $x2 = \frac{21}{20}$, $x3 = -\frac{17}{20}$.

Esta función también puede obtener una solución para $\mathbf{A}\ \mathbf{X} = \mathbf{B}$, cuando **B** es una matriz. En este caso la solución es una matriz que cumple la relación.

Otra forma de obtener la matriz escalonada es por medio de la función:

ReducedRowEchelonForm(A)

Ésta aplica el método de Gauss-Jordan para escalonar la matriz. Consúltese su página de ayuda.

Otras funciones proporcionadas por este paquete incluyen:

ForwardSubstitute(A, B)

ForwardSubstitute(A)

la cual resuelve el sistema **A X** = **B**, donde **A** debe ser una matriz triangular inferior. La segunda forma calcula también la solución, pero en este caso **A** debe ser la matriz aumentada del sistema.

Otra función que se puede usar para resolver un sistema de ecuaciones es:

LeastSquares(A, B)

Donde **A** es una matriz (las entradas de ésta también pueden ser dadas en forma de una lista o un conjunto), mientras que **B** es una matriz, un vector columna o un conjunto de variables. Esta función devuelve el vector que mejor aproxima la solución de la ecuación **A X** = **B**, esta solución aproximada es calculada por medio del método de mínimos cuadrados. Vease su página de ayuda.

5.6.12. Determinante de una matriz y matrices adjuntas

El determinante de una matriz puede ser calculado por medio de la función **Determinant(A)**, donde **A** es una matriz cuadrada. Por ejemplo:

```
> A := Matrix([[1, 3, 5], [4, 3, 2], [5, 2, 4]]);
```

$$A := \begin{bmatrix} 1 & 3 & 5 \\ 4 & 3 & 2 \\ 5 & 2 & 4 \end{bmatrix}$$

```
> Determinant(A);
```

$$-45$$

A continuación calcularemos el área del paralelogramo generado por los siguientes vectores:

```
> v1 := Vector[row]([3, 4]);
```

$$v1 := [3, 4]$$

```
> v2 := Vector[row]([-4, 5]);
```

$$v2 := [-4, 5]$$

```
> M := Matrix([[v1], [v2]]);
```

$$M := \begin{bmatrix} 3 & 4 \\ -4 & 5 \end{bmatrix}$$

Esta área está dada por: $\left| \det\left(\begin{bmatrix} x1 & y1 \\ x2 & y2 \end{bmatrix} \right) \right|$

donde *x1*, *y1* son las entradas de *v1*, mientras que *x2*, *y2* son las de *v2*.

```
> area := abs(Determinant(A));
```

$$area := 45$$

Grafiquemos los vectores y el paralelogramo. Primero generamos las gráficas de los vectores:

```
> grafv1 := plots[arrow](v1, shape=arrow):
> grafv2 := plots[arrow](v2, shape=arrow):
```

A continuación generamos la gráfica del paralelogramo:

```
> suma := v1 + v2;
```

$$suma := [-1, 9]$$

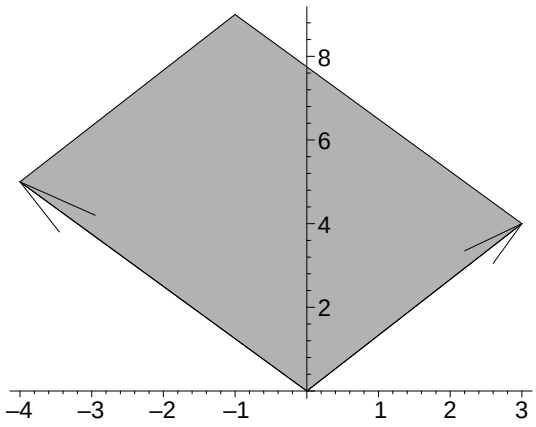
```
> puntos := [[0,0], [v1[1], v1[2]], [suma[1], suma[2]], [v2[1], v2[2]]];
```

$$puntos := [[0, 0], [3, 4], [-1, 9], [-4, 5]]$$

```
> paralel := plots[polygonplot](puntos, color=cyan):
```

Ahora desplegamos las tres gráficas juntas:

```
> plots[display]({grafv1, grafv2, paralel});
```



Por otro lado, la adjunta de una matriz puede ser calculada por medio de la función:

Adjoint(A)

Donde **A** es una matriz cuadrada. Esta función devuelve una matriz tal que el producto de **A** por la adjunta es igual al producto del determinante de **A** por la matriz identidad. Veamos un ejemplo:

```
> A := Matrix([[2, 4, 3], [4, 2, 5], [3, 2, 5]]);
```

$$A := \begin{bmatrix} 2 & 4 & 3 \\ 4 & 2 & 5 \\ 3 & 2 & 5 \end{bmatrix}$$

```
> Aadj := Adjoint(A);
```

$$Aadj := \begin{bmatrix} 0 & -14 & 14 \\ -5 & 1 & 2 \\ 2 & 8 & -12 \end{bmatrix}$$

Verificaremos que esta matriz cumple la propiedad antes mencionada:

```
> AxAadj := A.Aadj;
```

$$AxAadj := \begin{bmatrix} -14 & 0 & 0 \\ 0 & -14 & 0 \\ 0 & 0 & -14 \end{bmatrix}$$

```
> Adet := Determinant(A);
```

$$Adet := -14$$

```
> Id := IdentityMatrix(3);
```

$$Id := \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> Adet*Id;
```

$$\begin{bmatrix} -14 & 0 & 0 \\ 0 & -14 & 0 \\ 0 & 0 & -14 \end{bmatrix}$$

Por lo tanto la propiedad se cumple.

5.6.13. Valores propios y vectores propios

Los valores propios de una matriz pueden ser calculados por medio de la función:

Eigenvalues(A)

Donde **A** es una matriz cuadrada. Si la matriz **A** contiene valores numéricos, esta función usa un método numérico para determinar los valores propios. Si la matriz contiene expresiones simbólicas, los valores propios son calculados resolviendo el polinomio característico:

Determinant(lambda I - A) = 0

Donde **I** es una matriz identidad de la misma dimension que **A**. Este polinomio caracteristico puede ser calculado con la función **CharacteristicPolynomial**. Su sintaxis es:

CharacteristicPolynomial(A, lambda)

Por ejemplo:

```
> A := Matrix([[1, 4, 3], [1, 5, 2], [3, 6, 4]]);
```

$$A := \begin{bmatrix} 1 & 4 & 3 \\ 1 & 5 & 2 \\ 3 & 6 & 4 \end{bmatrix}$$

```
> CharacteristicPolynomial(A, lambda);
```

$$\lambda^3 - 10\,\lambda^2 + 4\,\lambda + 11$$

```
> evalf(Eigenvalues(A));
```

$$\begin{bmatrix} 9{,}453812782 - 0{,}2\,10^{-9}\,I \\ -0{,}839620113 - 0{,}7660254040\,10^{-9}\,I \\ 1{,}385807331 + 0{,}9660254040\,10^{-9}\,I \end{bmatrix}$$

Por otro lado, los vectores propios de una matriz pueden ser calculados por medio de la función:

Eigenvectors(A)

Donde **A** es una matriz cuadrada. Esta función devuelve una secuencia de expresiones cuyo primer elemento es un vector con los eigenvalores de **A**, mientras que el segundo elemento es una matriz cuyas columnas son los eigenvectores de **A**; la i-esima columna de esta matriz es un eigenvector asociado con el i-esimo eigenvalor del vector devuelto. Por ejemplo, calculemos los vectores propios de la matriz definida anteriormente:

```
> evalf(Eigenvectors(A));
```

$$\begin{bmatrix} 9{,}453812782 - 0{,}2\,10^{-9}\,I \\ -0{,}839620113 - 0{,}7660254040\,10^{-9}\,I \\ 1{,}385807331 + 0{,}9660254040\,10^{-9}\,I \end{bmatrix},$$

$$[0{,}634780374 + 0{,}2151270927\,10^{-9}\,I\,,\ -1{,}411731932 - 0{,}8010516764\,10^{-9}\,I\,,$$

$$0{,}5269515574 + 0{,}6707357725\,10^{-9}\,I]$$

$$[0{,}591578611 - 0{,}575635464\,10^{-10}\,I\,,\ -0{,}1007373862 + 0{,}2761882708\,10^{-9}\,I\,,$$

$$-0{,}6991745564 - 0{,}1710303189\,10^{-9}\,I]$$

$$[1.\,,\ 1.\,,\ 1.]$$

Al invocar esta función se puede incluir la opción **output='values'**, **'vectors'** o **'list'**, dependiendo si se desea obtener los eigenvalores, eigenvectores o ambos, respectivamente.

5.6.14. Otras funciones de LinearAlgebra

En las secciones anteriores solo se han presentado algunas de las funciones contenidas en **LinearAlgebra**, existen otras que también pueden ser usadas en operaciones de matrices y vectores. Entre ellas podemos mencionar las siguientes:

- **IsDefinite**. Determina si una matriz es definida positiva, definida negativa, semidefinida positiva o semidefinida negativa.
- **ConditionNumber**. Calcula el número de condición de una matriz **A**, dado por:
  norm(A)*norm(inverse(A)).
- **IsOrthogonal**. Determina si una matriz es ortogonal.
- **IsUnitary**. Determina si una matriz es unitaria.
- **JordanForm**. Calcula la forma de Jordan de una matriz.
- **MinimalPolynomial**. Calcula el polinomio de menor grado que "anula.[a] una matriz.
- **MatrixNorm**. Calcula la norma de una matriz.
- **GramSchmidt**. Calcula una lista o conjunto de vectores ortogonales, a partir de una lista o conjunto de vectores linealmente independientes, usando el método de Gram-Schmidt.
- **HermiteForm**. Calcula la forma normal de Hermite de una matriz de n x n de polinomios univariados, sobre el campo de los racionales.
- **SingularValues**. Calcula los valores singulares de una matriz.
- **SmithForm**. Calcula la forma normal de Smith de una matriz.

Consúltese la página de ayuda de estas funciones para obtener más información.