ALGEBRA LINEAL

El lenguaje Wolfram representa matrices y vectores mediante listas. Todo lo que no sea una lista, Wolfram Language lo considera un escalar.

Un vector en Wolfram Language consiste en una lista de escalares. Una matriz consiste en una lista de vectores que representan cada una de sus filas. Para ser una matriz válida, todas las filas deben tener la misma longitud, de modo que sus elementos formen un arreglo rectangular.

Funciones en la estructura de vectores y matrices

VectorQ [expr]
Da Verdadero si **expr** tiene la forma de un vector y Falso en caso contrario.

MatrixQ [expr]
Da Verdadero si expr tiene la forma de una matriz y Falso en caso contrario.

Dimensions [expr]
Da una lista de las dimensiones de un vector o matriz.

Ejemplos:

In[ ]:=
```
Clear["Global`*"]
```
borra

La lista {a, b, c} tiene la forma de un vector:

In[ ]:=
```
VectorQ[{a, b, c}]
```
¿vector?

Out[ ]=
```
True
```

Todo lo que no sea manifiestamente una lista se trata como un escalar, por lo que aplicar **VectorQ** da como resultado Falso:

In[ ]:=
```
VectorQ[x + y]
```
¿vector?

Out[ ]=
```
False
```

Escribimos una matriz de 2 x 3:

In[ ]:=
```
Dimensions[{{a, b, c}, {ap, bp, cp}}]
```
dimensiones

Out[ ]=
```
{2, 3}
```

Para un vector, **Dimensions** proporciona una lista con un único elemento igual al resultado de su longitud:

In[•]:=
```
Dimensions[{a, b, c}]
```
dimensiones

Out[•]=
```
{3}
```

Este objeto no cuenta como una matriz porque sus filas tienen longitudes diferentes:

In[•]:=
```
MatrixQ[{{a, b, c}, {ap, bp}}]
```
¿matriz?

Out[•]=
```
False
```

Operaciones con escalares, vectores y matrices

La mayoría de las funciones matemáticas en Wolfram Language están configuradas para aplicarse por separado a cada elemento de una lista. Esto aplica especialmente a todas las funciones que tienen el atributo Listable.

Una consecuencia es que la mayoría de las funciones matemáticas se aplican elemento por elemento a matrices y vectores.

Ejemplos:

El registro se aplica por separado a cada elemento del vector:

In[•]:=
```
Log[{a, b, c}]
```
logaritmo

Out[•]=
```
{Log[a], Log[b], Log[c]}
```

Lo mismo ocurre con una matriz o, en realidad, con cualquier lista anidada:

In[•]:=
```
Log[{{a, b}, {c, d}}]
```
logaritmo

Out[•]=
```
{{Log[a], Log[b]}, {Log[c], Log[d]}}
```

La función de diferenciación D también se aplica por separado a cada elemento de una lista:

In[•]:=
```
D[{x, x^2, x^3}, x]
```
deriva

Out[•]=
$$\{1, 2x, 3x^2\}$$

La suma de dos vectores se realiza elemento por elemento:

In[•]:=
```
{a, b} + {ap, bp}
```

Out[•]=
```
{a + ap, b + bp}
```

Si intenta sumar dos vectores con diferentes longitudes, obtendrá un error:

In[•]:=
```
{a, b, c} + {ap, bp}
```

Thread : Objects of unequal length in {a, b, c } + {ap, bp } cannot be combined.

Out[•]=
```
{ap, bp} + {a, b, c}
```

El resultado de agregar el escalar 1 a un vector es que se suma el escalar a cada elemento del vector:

In[•]:=
```
1 + {a, b}
```

Out[•]=
```
{1 + a, 1 + b}
```

Cualquier objeto que no sea manifiestamente una lista se trata como un escalar. Aquí, **c** se trata como un escalar y se suma por separado a cada elemento del vector:

In[•]:=
```
{a, b} + c
```

Out[•]=
```
{a + c, b + c}
```

Es importante tener en cuenta que **Wolfram Language** trata un objeto como un vector en una operación específica solo si el objeto es explícitamente una lista al momento de realizar la operación. Si el objeto no es explícitamente una lista, **Wolfram Language** siempre lo trata como un escalar. Esto significa que se pueden obtener diferentes resultados según se asigne un objeto específico como lista antes o después de realizar una operación específica.

El objeto **p** se trata como un escalar y se agrega por separado a cada elemento del vector:

In[•]:=
```
{a, b} + p
```

Out[•]=
```
{a + p, b + p}
```

Esto es lo que sucede si ahora reemplaza **p** por la lista {c, d}. Ahora el tratamiento es vectorial:

In[•]:=
```
% /. p → {c, d}
```

Out[•]=
```
{{a + c, a + d}, {b + c, b + d}}
```

Habríamos obtenido un resultado diferente si hubieras reemplazado **p** por {c, d}. Ahora el tratamiento es escalar:

In[•]:=
```
{a, b} + {c, d}
```

Out[•]=
```
{a + c, b + d}
```

Multiplicando vectores y matrices

c v, c m, ...
Se multiplica cada elemento por un escalar.

u· v, v· m, m· v, m1· m2, ...
Multiplicación de vectores y matrices.

Cross [u, v]
Producto vectorial de dos vectores.

Outer [Times, t, u]

Otro producto.

KroneckerProduct [m1, m2, ...]

Producto Kronecker

Ejemplos:

```
In[•]:= Clear["Global`*"]
```
borra

Esto multiplica cada elemento del vector por el escalar k:

```
In[•]:= k {a, b, c}
Out[•]=
{a k, b k, c k}
```

El operador "**dot**" da el producto escalar de dos vectores:

```
In[•]:= {a, b, c}.{ap, bp, cp}
Out[•]=
a ap + b bp + c cp
```

Podemos usar **dot** para multiplicar una matriz por un vector:

```
In[•]:= {{a, b}, {c, d}}.{x, y}
Out[•]=
{a x + b y, c x + d y}
```

El punto también es la notación para la multiplicación de matrices en el lenguaje Wolfram:

```
In[•]:= {{a, b}, {c, d}}.{{1, 2}, {3, 4}}
Out[•]=
{{a + 3 b, 2 a + 4 b}, {c + 3 d, 2 c + 4 d}}
```

El punto también es la notación para la multiplicación de matrices en el lenguaje Wolfram:

Es importante tener en cuenta que se puede usar "punto" para la multiplicación de vectores por matrices tanto por la izquierda como por la derecha. El lenguaje Wolfram no distingue entre vectores de "fila" y de "columna". Punto realiza cualquier operación posible. (En términos formales, a.b contrae el último índice del tensor a con el primer índice de b).

Definimos una matriz **m** y un vector **v**:

```
In[•]:= m = {{a, b}, {c, d}};
        v = {x, y};
```

```
In[•]:= MatrixForm[m]
```
forma de matriz

```
Out[•]//MatrixForm=
(a  b)
(c  d)
```

```
In[•]:= MatrixForm[v]
```
forma de matriz

```
Out[•]//MatrixForm=
(x)
(y)
```

Si queremos multiplicar **v** por **m** por la izquierda. En este caso, el objeto **v** se considera un vector columna:

```
In[•]:= m.v
Out[•]=
{a x + b y, c x + d y}
```

```
In[•]:= MatrixForm[m.v]
        forma de matriz
Out[•]//MatrixForm=
(a x + b y)
(c x + d y)
```

También puedes usar el punto para multiplicar **v** por **m** por la derecha. Ahora **v** se trata como un vector fila:

```
In[•]:= v.m
Out[•]=
{a x + c y, b x + d y}
```

```
In[•]:= MatrixForm[v.m]
        forma de matriz
Out[•]//MatrixForm=
(a x + c y)
(b x + d y)
```

Puedes multiplicar **m** por **v** en ambos lados para obtener un escalar:

```
In[•]:= v.m.v
Out[•]=
x (a x + c y) + y (b x + d y)
```

Para algunos propósitos, puede que necesite representar vectores y matrices simbólicamente sin especificar sus elementos. Puede usar Dot para representar la multiplicación de estos objetos simbólicos.

El punto actúa aquí efectivamente como una forma no conmutativa de multiplicación:

```
In[•]:= a.b.a
Out[•]=
a.b.a
```

Es, sin embargo, asociativo:

```
In[•]:= (a.b).(a.b)
Out[•]=
a.b.a.b
```

Los productos escalares de sumas no se expanden automáticamente:

```
In[•]:= (a + b).c.(d + e)
Out[•]=
(a + b).c.(d + e)
```

Puedes aplicar la ley distributiva en este caso usando la función **Distribute**, como se explica en “Operaciones Estructurales”:

```
In[•]:= Distribute[%]
        distribuye
Out[•]=
        a.c.d + a.c.e + b.c.d + b.c.e
```

El operador “punto” proporciona “productos internos” de vectores, matrices, etc. En cálculos más avanzados, también podría necesitar construir productos externos o de Kronecker de vectores y matrices. Puede usar la función general “Outer” o “KroneckerProduct” para ello.

El producto exterior de dos vectores es una matriz:

```
In[•]:= Outer[Times, {a, b}, {c, d}]
        produ··· multiplicación
Out[•]=
        {{a c, a d}, {b c, b d}}
```

El producto externo de una matriz y un vector es un tensor de rango tres:

```
In[•]:= Outer[Times, {{1, 2}, {3, 4}}, {x, y, z}]
        produ··· multiplicación
Out[•]=
        {{{x, y, z}, {2 x, 2 y, 2 z}}, {{3 x, 3 y, 3 z}, {4 x, 4 y, 4 z}}}
```

El producto de Kronecker de una matriz y un vector es una matriz:

```
In[•]:= KroneckerProduct[{{1, 2}, {3, 4}}, {x, y, z}]
        producto Kronecker
Out[•]=
        {{x, y, z, 2 x, 2 y, 2 z}, {3 x, 3 y, 3 z, 4 x, 4 y, 4 z}}
```

El producto de Kronecker de un par de matrices 2 por 2 es una matriz 4 por 4:

```
In[•]:= KroneckerProduct[{{1, 2}, {3, 4}}, {{a, b}, {c, d}}]
        producto Kronecker
Out[•]=
        {{a, b, 2 a, 2 b}, {c, d, 2 c, 2 d}, {3 a, 3 b, 4 a, 4 b}, {3 c, 3 d, 4 c, 4 d}}
```