Unidad 1: Sistemas de ecuaciones lineales

Un sistema de ecuaciones algebraicas lineales tiene la forma general:

$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$
$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$
.
.
.
$a_nx_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n$

siendo a coeficientes constantes, b las constantes, n el número de ecuaciones y x las incógnitas.

Sistemáticamente se usa la notación matricial para facilitar su resolución, de la siguiente manera:

$$\underline{\underline{A}} = \begin{bmatrix} a_{11} & a_{12} & ... & a_{1n} \\ a_{21} & a_{22} & ... & a_{2n} \\ \vdots & \vdots & ... & \vdots \\ a_{n1} & a_{n2} & ... & a_{nn} \end{bmatrix}$$

donde cada elemento a_{ij} representa un elemento de la matriz (los coeficientes constantes).

Nota: De aquí en adelante se usará la siguiente nomenclatura.

$\underline{\underline{M}}$: matrices
$\underline{V}$: vectores filas y vectores columnas
E : constantes y escalares

Las incognitas x_j se representan en el vector columna:

$$\underline{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

Y las constante b_i con el vector columna:

$$\underline{B} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Que en síntesis es: $\underline{\underline{A}}\ \underline{X} = \underline{B}$

Por lo tanto, el objetivo es despejar el vector $\underline{X}$

Formas de definir matrices

1. Lista de listas
2. Arreglos en numpy
3. Matrices en sympy

1. Lista de listas

In [1]:
```
# Mediante una lista de listas
A = [
     [5, 1, 2],
     [7, 3, 7],
     [4, 7, 8],
               ]

print(A)
```

```
[[5, 1, 2], [7, 3, 7], [4, 7, 8]]
```

In [2]:
```
# Tipo de dato
type(A)
```

Out[2]:
```
list
```

In [3]:
```
# Tamaño de la matriz
m = len(A)     # Número de filas de la matriz, o tamaño de la lista A.
n = len(A[0]) # Número de columnas de la matriz, o tamaño de la primera l

print('El número de filas es:')
print(m)
print('El número de columnas es:')
print(n)
```

```
El número de filas es:
3
El número de columnas es:
3
```

In [4]:
```
# Llamado de un elemento
a00 = A[0][0]
a22 = A[2][2]
a12 = A[1][2]

# Impresión de resultados.
print('\nEl elemento de la fila 1 y la columna 1 es:')
print(a00)
print('\nEl elemento de la fila 3 y la columna 3 es:')
print(a22)
print('\nEl elemento de la fila 2 y la columna 3 es::')
print(a12)
```

```
El elemento de la fila 1 y la columna 1 es:
5

El elemento de la fila 3 y la columna 3 es:
8

El elemento de la fila 2 y la columna 3 es::
7
```

In [5]:
```python
# Llamado de filas
f0 = A[0] # Se llama la primera fila. En Matlab es con 1, acá es con 0.
f2 = A[2] # Se llama la tercera fila.

# Impresión de resultados.
print('\nLa primera fila es:')  # El comando \n es usado dejar un espacio
print(f0)
print('\nLa tercera fila es:')
print(f2)
```

```
La primera fila es:
[5, 1, 2]

La tercera fila es:
[4, 7, 8]
```

In [6]:
```python
# Llamado de columnas
c0 = []
for i in range(len(A)):
    c0 += [A[i][0]]

print('\nLa primera columna es:')
print(c0)
```

```
La primera columna es:
[5, 7, 4]
```

2. Arreglo en numpy

In [7]:
```python
# Mediante el módulo numpy con arrays (arreglos)

# Importar la librería
import numpy as np

# Definición explícita
B = np.array([
              [3, 1, 9],
              [2, 4, 1],
                       ])

# Definición directa con una variable previamente guardada
C = [
```

```python
     [3, 1, 9],
     [2, 4, 1],
            ]

D = np.array(C)
```

In [8]:
```python
# Impresión de resultados
print('\n La matriz B es:')
print(B)

print('\n La matriz D es:')
print(D)
```

```
 La matriz B es:
[[3 1 9]
 [2 4 1]]

 La matriz D es:
[[3 1 9]
 [2 4 1]]
```

In [9]:
```python
# Tipo de dato
type(B)
```

Out[9]:
```
numpy.ndarray
```

In [10]:
```python
# Tamaño de la matriz
dim_B = B.shape
print(dim_B)   # Imprime el tamaño del arreglo B.
```

```
(2, 3)
```

In [11]:
```python
# Tipo de dato
type(dim_B)
```

Out[11]:
```
tuple
```

In [12]:
```python
# Definiendo número de filas y columnas.
m = dim_B[0]
n = dim_B[1]
```

In [13]:
```python
# Impresión de resultados
print('El número de filas es:')
print(m)
print('El número de columnas es:')
print(n)
```

```
El número de filas es:
2
El número de columnas es:
3
```

In [14]:
```python
#print(A.shape)   # Genera error.
```

Otras formas de definir arreglos:

In [15]:
```
# Arreglo de unos
E = np.ones((4, 5))

print(E)
print("\nCon elementos de tipo:")
print(type(E[0][0]))
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]

Con elementos de tipo:
<class 'numpy.float64'>
```

In [16]:
```
# Arreglo de ceros
F = np.zeros((3, 2))

print(F)
print("\nCon elementos de tipo:")
print(type(F[0][0]))
```

```
[[0. 0.]
 [0. 0.]
 [0. 0.]]

Con elementos de tipo:
<class 'numpy.float64'>
```

In [17]:
```
# Matriz vacía rellena con los valores residuales de la memoria.
G = np.empty((2, 3))

print(G)
print("\nCon elementos de tipo:")
print(type(G[0][0]))
```

```
[[0. 0. 0.]
 [0. 0. 0.]]

Con elementos de tipo:
<class 'numpy.float64'>
```

In [18]:
```
# De forma opcional se puede definir el tipo de dato que tendrá el arregl
H = np.zeros((3, 3), dtype=int)   # dytpe tiene muchísimas posibilidades.

print(H)
print("\nCon elementos de tipo:")
print(type(H[0][0]))
```

```
[[0 0 0]
 [0 0 0]
 [0 0 0]]
```

In [19]:
```python
# Matriz identidad para matrices cuadradas
np.identity(5)
```

Out[19]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [20]:
```python
# Matriz identidad en un caso más general mxn
np.eye(4,3)
```

Out[20]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 0.]])
```

Llamado de elementos, filas y columnas.

In [21]:
```python
print(B) # Previamente fue definida la matriz B.
```

```
[[3 1 9]
 [2 4 1]]
```

In [22]:
```python
# Llamado de un elemento
b12 = B[1][2] # En la numeración natural sería el elemento de la fila 2 y

print(f'El elemento b12 es: {b12}')  # Nótese el uso de la f y de {}.
```

```
El elemento b12 es: 1
```

In [23]:
```python
# Alternativamente, se usa así:
b12 = B[1, 2]

print(f'El elemento b12 es: {b12}')
```

```
El elemento b12 es: 1
```

In [24]:
```python
# Llamado de una fila
f1 = B[0] # Se llama la segunda fila.

print(f'La segunda fila es: {f1}')
```

```
La segunda fila es: [3 1 9]
```

In [25]:
```python
#f2 = B[2] # Se llama la tercera fila. Como no existe: genera error.
```

In [26]:
```python
# Llamado de columnas
c1 = B[:, 1]
```

```
print(f'La segunda columna es: {c1}')
```

```
La segunda columna es: [1 4]
```

In [27]:
```
# Se define una matriz C, nuevamente con
C = np.array([
              [ 0,  1,  2,  3],
              [ 4,  5,  6,  7],
              [ 8,  9, 10, 11],
              [12, 13, 14, 15],  # Nótese que esta ',' del final no gener
                             ])

print(f'La matriz C es: \n{C}')
```

```
La matriz C es:
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]
```

Llamado de más de una fila o de una columna:

In [28]:
```
C[1:]
```

Out[28]:
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [29]:
```
C[:2]
```

Out[29]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [30]:
```
C[:, :2]
```

Out[30]:
```
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]])
```

In [31]:
```
C[1:, :2]
```

Out[31]:
```
array([[ 4,  5],
       [ 8,  9],
       [12, 13]])
```

In [32]:
```
C[1:3, 1:3]
```

Out[32]:
```
array([[ 5,  6],
       [ 9, 10]])
```

3. Matrices en sympy

In [33]:
```python
# Mediante el módulo sympy con "Array" y con "Matrix".

# Importar la librería
import sympy as sp

# Definición con 'Array', no se recomienda usar pues es inmutable.
J = sp.Array([
              [ 8,   1,   9],
              [-1,   4,  -2],
              [ 2,  -4,   1],
                            ])
```

In [34]:
```python
# Impresión de resultados
print('La matriz J es:')
J  # Acá no se pone el comando 'print' para que genere un 'Out', y en sym
```

```
La matriz J es:
```

Out[34]:
$$\begin{bmatrix} 8 & 1 & 9 \\ -1 & 4 & -2 \\ 2 & -4 & 1 \end{bmatrix}$$

In [35]:
```python
# Tipo de dato
type(J)
```

Out[35]:
```
sympy.tensor.array.dense_ndim_array.ImmutableDenseNDimArray
```

In [36]:
```python
# Definición con "Matrix"
K = sp.Matrix([
               [-1,   1,   9],
               [ 4,   4,  -2],
               [ 2,   0,  -1],
                             ])
```

In [37]:
```python
# Impresión de resultados
print('La matriz K es:')
K
```

```
La matriz K es:
```

Out[37]:
$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 4 & -2 \\ 2 & 0 & -1 \end{bmatrix}$$

In [38]:
```python
# Tipo de dato
type(K)
```

Out[38]:
```
sympy.matrices.dense.MutableDenseMatrix
```

In [39]:
```python
# Tamaño de la matriz
dim_K = K.shape   # Nótese que es el mismo comando en numpy y en sympy
print(dim_K)      # Imprime el tamaño del arreglo K
```

```
(3, 3)
```

In [40]:
```python
# Tipo de dato
type(dim_K)
```

Out[40]:
```
tuple
```

Formas específicas de arreglos:

Se recomienda revisar en detalle la documentación de sympy en: [https://docs.sympy.org/latest/tutorial/matrices.html (https://docs.sympy.org/latest/tutorial/matrices.html)](https://docs.sympy.org/latest/tutorial/matrices.html)

In [41]:
```python
# Matriz de ceros.
sp.zeros(2)
```

Out[41]:
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

In [42]:
```python
sp.zeros(3, 2)
```

Out[42]:
$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$$

In [43]:
```python
# Matriz de unos.
sp.ones(3)
```

Out[43]:
$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

In [44]:
```python
# Matriz identidad
sp.eye(3)
```

Out[44]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In [45]:
```python
# Matriz diagonal
sp.diag(1, 2, 3)
```

Out[45]:
$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$

Llamado de elementos, filas y columnas.

In [46]:
```
K # Previamente fue definida la matriz K.
```
Out[46]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 4 & -2 \\ 2 & 0 & -1 \end{bmatrix}$$

In [47]:
```
# Teniendo en cuenta la longitud de matriz, es decir, la cantidad de elem
len(K)   # Nótese que aquí sí se puede usar el comando "len()" a diferenci
```
Out[47]: 9

In [48]:
```
# Se puede llamar a cualquiera de los elementos de la matriz en un ordena
K[0]
```
Out[48]: -1

In [49]:
```
K[-1]   # Llamado del último elemento, que es lo mismo que llamar K[8]
```
Out[49]: -1

In [50]:
```
# Pero el llamado básico se hace de la misma manera que numpy
K[1, 1]   # No funciona poner K[1][1], genera error.
```
Out[50]: 4

In [51]:
```
# Llamado de filas
K.row(1)   # LLamado de la segunda fila
```
Out[51]: $\begin{bmatrix} 4 & 4 & -2 \end{bmatrix}$

In [52]:
```
# Llamado de columnas
K.col(2)
```
Out[52]:

$$\begin{bmatrix} 9 \\ -2 \\ -1 \end{bmatrix}$$

Modificación de elementos

Cambiar algún elemento de la matriz

1. En una lista de listas

In [53]:
```
A # Previamente fue definida la matriz A.
```
Out[53]: [[5, 1, 2], [7, 3, 7], [4, 7, 8]]

In [54]:
```
# Por ejemplo, el elemento de la fila 3 y la columna 1
A[2][0]
```

Out[54]:
```
4
```

In [55]:
```
# Se puede modificar directamente
A[2][0] = -8
```

In [56]:
```
A   # Matriz A modificada
```

Out[56]:
```
[[5, 1, 2], [7, 3, 7], [-8, 7, 8]]
```

2. En un array de numpy

In [57]:
```
C # Previamente fue definida la matriz C.
```

Out[57]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [58]:
```
# Modificando el elemento C[2, 1]
C[2, 1] = -15
```

In [59]:
```
C   # Matriz C modificada
```

Out[59]:
```
array([[  0,   1,   2,   3],
       [  4,   5,   6,   7],
       [  8, -15,  10,  11],
       [ 12,  13,  14,  15]])
```

3. En una matriz de sympy

In [60]:
```
K # Previamente fue definida la matriz K.
```

Out[60]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 4 & -2 \\ 2 & 0 & -1 \end{bmatrix}$$

In [61]:
```
# Modificando el elemento K[1, 1]
K[1, 1] = 0
```

In [62]:
```
K   # Matriz K modificada
```

Out[62]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 0 & -2 \\ 2 & 0 & -1 \end{bmatrix}$$

Operaciones con matrices y vectores

1. Suma
2. Multiplicación
3. Inversión
4. Matriz transpuesta
5. Producto punto y producto cruz

Las operaciones básicas presentan algunas dificultades en una lista de lista, por lo que, claramente es mejor realizar las operaciones con los módulos numpy y sympy.

1. Suma

La suma de matrices solo es posible en matrices del mismo tamaño mxn.

La suma de las matrices $\underline{\underline{A}}$ y $\underline{\underline{B}}$ es denotada por: $\underline{\underline{A}} + \underline{\underline{B}}$

Usando numpy

In [63]:
```
# Usando el módulo numpy se crean las matrices M y N.
M = np.array([
              [ 2, 7],
              [-1, 3],
                     ])

N = np.array([
              [-3, 0],
              [ 9, 1],
                     ])
```

In [64]:
```
M+N  # La suma de las matrices M y N.
```

Out[64]:
```
array([[-1,  7],
       [ 8,  4]])
```

Usando sympy

In [65]:
```
# Usando el módulo sympy se crean las matrices O y P.

# Se crean algunas variables simbólicas
a, b, c, d = sp.symbols("a b c d")

O = sp.Matrix([
               [ a, c],
               [ b, d],
                      ])

P = sp.Matrix([
               [-a, b],
```

```
                 [ c, d],
                        ])
```

In [66]:
```
Q-R  # [illegible]
```

Out[66]:

$$\begin{bmatrix} 2a & -b+c \\ b-c & 0 \end{bmatrix}$$

2. Multiplicación

El producto de una matriz A de tamaño axb con una matriz B de tamaño cxd solo es posible si $b = c$ y, en donde la matriz resultante tendrá un tamaño axd.

La multiplicación de las matrices $\underline{\underline{A}}$ y $\underline{\underline{B}}$ es denotada por: $\underline{\underline{A}}$ x $\underline{\underline{B}}$

Usando numpy

In [67]:
```
# Usando el módulo numpy se crean las matrices Q y R.
Q = np.array([
                [3, -2],
                       ])

R = np.array([
                [ 3],
                [-1],
                    ])
```

In [68]:
```
np.matmul(Q, R)   # La operación Q*R hace otra cosa completamente diferent
```

Out[68]:
```
array([[11]])
```

In [69]:
```
Q@R   # Otra forma más sencilla de hacer la misma operación.
```

Out[69]:
```
array([[11]])
```

Usando sympy

In [70]:
```
# Usando el módulo sympy se crea la matriz A y el vector X.

# Se crean algunas variables simbólicas adicionales.
x, y = sp.symbols("x y")

A = sp.Matrix([
                 [ a, b],
                 [ c, c],
                        ])

X = sp.Matrix([
                 [x],
                 [y],
                    ])
```

In [71]:
```
A*X   # En sympy sí se puede hacer esta operación así.
```
Out[71]:
$$\begin{bmatrix} ax+by \\ cx+cy \end{bmatrix}$$

3. Inversión de matrices

Solo es posible invertir matrices cuadradas no singulares, es decir, que su determinante sea diferente de 0.

La inversa de la matriz $\underline{\underline{A}}$ es denotada por $\underline{\underline{A}}^{-1}$.

El determinante de la matriz a se puede denotar como $det(\underline{\underline{A}})$ o como $|\underline{\underline{A}}|$.

Usando numpy

In [72]:
```
# Usando el módulo numpy se crea la matriz S.
S = np.array([[ 1, -2],
              [-3, 5]])

S   # Se imprime S
```
Out[72]:
```
array([[ 1, -2],
       [-3,  5]])
```

In [73]:
```
np.linalg.det(S)   # Se determina el determinante de la matriz S.
```
Out[73]:
```
-1.0000000000000004
```

In [74]:
```
# Como el determinante es diferente de 0, la matriz es inversible.
np.linalg.inv(S)   # Inversa de la matriz S.
```
Out[74]:
```
array([[-5., -2.],
       [-3., -1.]])
```

Usando sympy

In [75]:
```
# Usando la matriz A previamente creada con varibles simbólicas.
A
```
Out[75]:
$$\begin{bmatrix} a & b \\ c & c \end{bmatrix}$$

In [76]:
```
# Cálculo del determinante.
A.det()
```
Out[76]:
$$ac - bc$$

In [77]:
```
A.inv()   # Inversa de la matriz A. Aquí la utilidad del módulo sympy para
```

Out[77]:

$$\begin{bmatrix} \frac{1}{a-b} & -\frac{b}{ac-bc} \\ -\frac{c}{ac-bc} & \frac{a}{ac-bc} \end{bmatrix}$$

4. Matriz transpuesta

Se denota a la transpuesta de la matriz $\underline{\underline{A}}$ como $\underline{\underline{A}}^T$.

¿Cómo se transpone la matriz $\underline{\underline{A}}$? (ejercicio de clase con lista de listas)

In [78]:
```
A = [[  0,   1,   2,   3,   4],
     [  5,   6,   7,   8,   9],
     [ 10,  11,  12,  13,  14]]
```

In [79]:
```
# Solución
```

Usando numpy

In [80]:
```
B # Previamente fue definida la matriz B.
```

Out[80]:
```
array([[3, 1, 9],
       [2, 4, 1]])
```

In [81]:
```
B.T   # Matriz transpuesta.
```

Out[81]:
```
array([[3, 2],
       [1, 4],
       [9, 1]])
```

Usando sympy

In [82]:
```
K # Previamente fue definida la matriz K.
```

Out[82]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 0 & -2 \\ 2 & 0 & -1 \end{bmatrix}$$

In [83]:
```
K.T   # Matriz transpuesta.
```

Out[83]:

$$\begin{bmatrix} -1 & 4 & 2 \\ 1 & 0 & 0 \\ 9 & -2 & -1 \end{bmatrix}$$

5. Producto punto y producto cruz

El producto punto o producto escalar de dos vectores está definido como:

$$\underline{A}.\,\underline{B} = (a_1, a_2, \ldots, a_n).\,(b_1, b_2, \ldots, b_n) = a_1 b_1 + a_2 b_2 + \ldots + a_n b_n = \sum a_i b_i$$

Alternativamente, se puede definir como:

$\underline{A}.\,\underline{B} = ||\underline{A}||.\,||\underline{B}||cos(\theta)$, donde θ es el ángulo formado entre los vectores.

Nota: De aquí en adelante un vector entre dos líneas a lado y lado representará la magnitud o norma: $||\underline{A}||$.
Una matriz entre dos líneas representa un determinante: $|A|$

Pregunta

¿Qué significa que el $\underline{A}.\,\underline{B} = 0$?

Pregunta

¿Qué significa que el $\underline{A}.\,\underline{B} = ||\underline{A}||.\,||\underline{B}||$?

El producto cruz o producto vectorial de dos vectores está definido en $\mathbb{R}^3$ como $\underline{A}$x$\underline{B}$, en donde:

$\underline{A} = (a_x, a_y, a_z)$
$\underline{B} = (b_x, b_y, b_z)$

es decir,

$$\underline{A} \text{ x } \underline{B} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

In [84]:
```python
# Usando sympy se obtiene

ax, ay, az, bx, by, bz = sp.symbols("a_x a_y a_z b_x b_y b_z")
ig, jg, kg = sp.symbols("\hat{i} \hat{j} \hat{k}")

AxB = sp.Matrix([
                 [ig, jg, kg],
                 [ax, ay, az],
                 [bx, by, bz]
                           ]).det()

sp.collect(AxB, [ig, jg, kg])
```

Out[84]: $\hat{i}\left(a_y b_z - a_z b_y\right) + \hat{j}\left(-a_x b_z + a_z b_x\right) + \hat{k}\left(a_x b_y - a_y b_x\right)$

De esta manera, $\underline{A}$ x $\underline{B} = \underline{C}$, donde:

$\underline{C} = (c_x, c_y, c_z),$

$c_x = a_y . b_z - a_z . b_y$
$c_y = -a_x . b_z + a_z . b_x$

Además, es últil la definición:

$||\underline{A}$ x $\underline{B}||$ = $||\underline{A}||$ $||\underline{B}||$ $|sin(\theta)|$

Pregunta

¿Qué significa que $\underline{A}$ x $\underline{B}$ = $\underline{0}$

Usando numpy

In [85]:
```
# Se crean los vectores u1 y u2.
u1 = np.array([7, -4, -1])
u2 = np.array([3, -5,  2])
```

In [86]:
```
u1  # Se visualiza el vector u1.
```

Out[86]:
```
array([ 7, -4, -1])
```

In [87]:
```
u2  # Se visualiza el vector u2.
```

Out[87]:
```
array([ 3, -5,  2])
```

In [88]:
```
# El producto punto de u1 y u2.
np.dot(u1, u2)
```

Out[88]:
```
39
```

Usando sympy

In [89]:
```
# Se crean algunas variables simbólicas
a1, a2, a3, b1, b2, b3 = sp.symbols("a_1 a_2 a_3 b_1 b_2 b_3")

# Se crean los vectores v1 y v2 con las variables simbólicas disponibles.
v1 = sp.Matrix([a1, a2, a3])
v2 = sp.Matrix([b1, b2, b3])
```

In [90]:
```
v1  # Se visualiza el vector v1.
```

Out[90]:

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

In [91]:
```python
v2  # Se visualiza el vector v2.
```
Out[91]:
$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

In [92]:
```python
# El producto punto de v1 y v2.
v1.dot(v2)
```
Out[92]: $a_1 b_1 + a_2 b_2 + a_3 b_3$

In [93]:
```python
# El producto cruz de v1 y v2.
v3 = v1.cross(v2)

v3  # Se imprime v3.
```
Out[93]:
$$\begin{bmatrix} a_2 b_3 - a_3 b_2 \\ -a_1 b_3 + a_3 b_1 \\ a_1 b_2 - a_2 b_1 \end{bmatrix}$$

In [94]:
```python
# El producto punto de v1 y v3.
v1.dot(v3)   # Debería dar 0.
```
Out[94]: $a_1 (a_2 b_3 - a_3 b_2) + a_2 (-a_1 b_3 + a_3 b_1) + a_3 (a_1 b_2 - a_2 b_1)$

In [95]:
```python
sp.simplify(v1.dot(v3))   # Se usa el comando "simplify".
```
Out[95]: 0

In [96]:
```python
# El producto punto de v2 y v2.
sp.simplify(v2.dot(v3))
```
Out[96]: 0

Resumen de conocimientos

- Formas de definir matrices
- Modificación de elementos
- Operaciones con matrices y vectores

Resumen de comandos usados

- print()
- type()
- len()
- for __ in __ :
- np.array()
- __ .shape
- np.ones()

- np.zeros()
- np.empty()
- dtype
- np.identitity()
- np.eye()
- sp.Array()
- sp.Matrix()
- sp.zeros()
- sp.ones()
- sp.eye()
- sp.diag()
- __ , __ = sp.symbols('__ __ ')
- np.linalg.det()
- np.linalg.inv()
- __ .det()
- __ .inv()
- __ .T
- sp.collect()
- np.dot(__ , __)
- __ .dot(__)
- __ .cross(__)