4101553 Métodos Numéricos aplicados a la Ingenieria Civil

Departamento de Ingeniería Civil
Universidad Nacional de Colombia
Sede Manizales

Docente: Juan Nicolás Ramírez Giraldo (jnramirezg@unal.edu.co (mailto:jnramirezg@unal.edu.co))

"*Cum cogitaveris quot te antecedant, respice quot sequantur*"
Séneca

Repositorio de la asignatura (https://github.com/jnramirezg/metodos_numericos_ingenieria_civil/)

Unidad 1: Sistemas de ecuaciones lineales

Matrices, modificación de matrices y operaciones

Un sistema de ecuaciones algebraicas lineales tiene la forma general:

$a_{11}x_1 + a_{12}x_2 + ... + a_{1n}x_n = b_1$

$a_{21}x_1 + a_{22}x_2 + ... + a_{2n}x_n = b_2$

.

.

.

$a_{n}x_1 + a_{n2}x_2 + ... + a_{nn}x_n = b_n$

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:

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

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 constantes b_i con el vector columna:

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

Que en sintésis 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],
               ]
```

In [2]:
```python
print(A)
```

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

In [3]:
```python
# También se puede usar simplemenhte
A
```

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

In [4]:
```python
# Tipo de dato
type(A)
```

Out[4]:
```
list
```

In [5]:
```python
# 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
```

In [6]:
```python
print('El número de filas de A es:')
print(m)
```

```
El número de filas de A es:
3
```

In [7]:
```python
print('El número de columnas de A es:')
print(n)
```

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

In [8]:
```python
A
```

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

In [9]:
```python
# Llamado de un elemento
a00 = A[0][0]
```

In [10]:
```python
# Impresión de resultados.
print('\nEl elemento de la fila 1 y la columna 1 es:')
print(a00)
```

```

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

In [11]:
```python
# Llamado de un elemento
a22 = A[2][2]
a12 = A[1][2]

# Impresión de resultados.
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 3 y la columna 3 es:
8

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

In [12]:
```
A
```

Out[12]:
```
[[5, 1, 2], [7, 3, 7], [4, 7, 8]]
```

In [13]:
```
# 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.
```

In [14]:
```
# 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 [15]:
```
# 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 [16]:
```
# Mediante el módulo numpy con arrays (arreglos).
# Importar la librería
import numpy as np
```

In [17]:
```
# Definición explícita.
B = np.array([
              [3, 1, 9],
              [2, 4, 1],
                      ])
```

In [18]:
```
# Impresión de resultados
print('\n La matriz B es:')
print(B)
```

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

In [19]:
```
# Definición directa con una variable previamente guardada.
C = [
     [3, 1, 9],
     [2, 4, 1],
              ]

D = np.array(C)
```

In [20]:
```
D
```

Out[20]:
```
array([[3, 1, 9],
       [2, 4, 1]])
```

In [21]:
```
# Tipo de dato
type(B)
```

Out[21]:
```
numpy.ndarray
```

In [22]:
```
# Tamaño de la matriz
dim_B = B.shape
print(dim_B)   # Imprime el tamaño del arreglo B.
```

```
(2, 3)
```

In [23]:
```
# Tipo de dato
type(dim_B)
```

Out[23]:
```
tuple
```

In [24]:
```
# Definiendo número de filas y columnas.
m = dim_B[0]
n = dim_B[1]
```

In [25]:
```
# 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 [26]:
```
print(A.shape)   # Genera error.
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-26-1ce236246647> in <module>
----> 1 print(A.shape)   # Genera error.

AttributeError: 'list' object has no attribute 'shape'
```

Otras formas de definir arreglos:

In [27]:
```
# Arreglo de unos
E = np.ones((4, 5))
```

In [28]:
```
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 [29]:
```
# Arreglo de ceros
F = np.zeros((3, 2))
```

In [30]:
```
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 [31]:
```
# Matriz vacía rellena con los valores residuales de la memoria.
G = np.empty((2, 3))
```

In [32]:
```
print(G)
print("\nCon elementos de tipo:")
print(type(G[0][0]))
```

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

In [33]:
```
# 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.
```

In [34]:
```
print(H)
print("\nCon elementos de tipo:")
print(type(H[0][0]))
```

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

Con elementos de tipo:
<class 'numpy.int32'>
```

In [35]:
```
# Matriz identidad para matrices cuadradas
np.identity(5)
```

Out[35]:
```
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 [36]:
```
# Matriz identidad en un caso más general mxn
np.eye(4,3)
```

Out[36]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 0.]])
```

Llamado de elementos, filas y columnas.

In [37]:
```
B # Previamente fue definida la matriz B.
```

Out[37]:
```
array([[3, 1, 9],
       [2, 4, 1]])
```

In [38]:
```
# 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 [39]:
```
# Alternativamente, se usa así:
b12 = B[1, 2]

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

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

In [40]:

```
B[1, 0]
```

Out[40]: 1

In [41]:

```
# 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 [42]:

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

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-42-6afd15b6aa00> in <module>
----> 1 f2 = B[2] # Se llama la tercera fila. Como no existe: genera error.

IndexError: index 2 is out of bounds for axis 0 with size 2
```

In [43]:

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

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

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

In [44]:

```
# 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 [45]:
```
C[1:]
```

Out[45]:
```
array([[ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [46]:
```
C[:2]
```

Out[46]:
```
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
```

In [47]:
```
C[:, :2]
```

Out[47]:
```
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]])
```

In [48]:
```
C[1:, :2]
```

Out[48]:
```
array([[ 4,  5],
       [ 8,  9],
       [12, 13]])
```

In [49]:
```
C[1:3, 1:3]
```

Out[49]:
```
array([[ 5,  6],
       [ 9, 10]])
```

3. Matrices en sympy

In [50]:
```
# Mediante el módulo sympy con "Array" y con "Matrix".
# Importar la librería
import sympy as sp
```

In [51]:
```
# Definición con 'Array', no se recomienda usar pues es inmutable.
J = sp.Array([
              [ 8,  1,  9],
              [-1,  4, -2],
              [ 2, -4,  1],
                         ])
```

In [52]:
```
# 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[52]:

$$\begin{bmatrix} 8 & 1 & 9 \\ -1 & 4 & -2 \\ 2 & -4 & 1 \end{bmatrix}$$

In [53]:
```
# Tipo de dato
type(J)
```

Out[53]:
```
sympy.tensor.array.dense_ndim_array.ImmutableDenseNDimArray
```

In [54]:
```
# Definición con "Matrix"
K = sp.Matrix([
                [-1,   1,   9],
                [ 4,   4,  -2],
                [ 2,   0,   8],
                             ])
```

In [55]:
```
# Impresión de resultados
print('La matriz K es:')
K
```

```
La matriz K es:
```

Out[55]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 4 & -2 \\ 2 & 0 & 8 \end{bmatrix}$$

In [56]:
```
# Tipo de dato
type(K)
```

Out[56]:
```
sympy.matrices.dense.MutableDenseMatrix
```

In [57]:
```
# 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 [58]:
```
# Tipo de dato
type(dim_K)
```

Out[58]:
```
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)

Las siguientes matrices tienen la forma de 'Matrix', es decir, son mutables:

In [59]:
```
# Matriz de ceros.
sp.zeros(2)
```

Out[59]:

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

In [60]:
```
sp.zeros(3, 2)
```

Out[60]:

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

In [61]:
```
# Matriz de unos.
sp.ones(3)
```

Out[61]:

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

In [62]:
```
# Matriz identidad
sp.eye(3)
```

Out[62]:

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

In [63]:
```
# Matriz diagonal
sp.diag(1, 2, 3)
```

Out[63]:

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

Llamado de elementos, filas y columnas.

In [64]:
```
K # Previamente fue definida la matriz K.
```

Out[64]:

$$\begin{bmatrix}-1 & 1 & 9\\4 & 4 & -2\\2 & 0 & 8\end{bmatrix}$$

```
In [65]: # 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[65]: 9

```
In [66]: # Se puede llamar a cualquiera de los elementos de la matriz en un ordena
         K[0]
```

Out[66]: -1

```
In [67]: K[-1]  # Llamado del último elemento, que es lo mismo que llamar K[8]
```

Out[67]: 8

```
In [68]: # Pero el llamado básico se hace de la misma manera que numpy
         K[1, 1]  # No funciona poner K[1][1], genera error.
```

Out[68]: 4

```
In [69]: # Llamado de filas
         K.row(1)  # LLamado de la segunda fila
```

Out[69]: $\begin{bmatrix} 4 & 4 & -2 \end{bmatrix}$

```
In [70]: # Llamado de columnas
         K.col(2)
```

Out[70]:

$$\begin{bmatrix} 9 \\ -2 \\ 8 \end{bmatrix}$$

Modificación de elementos

Cambiar algún elemento de la matriz

1. En una lista de listas

```
In [71]: A # Previamente fue definida la matriz A.
```

```
Out[71]: [[5, 1, 2], [7, 3, 7], [4, 7, 8]]
```

```
In [72]: # Por ejemplo, el elemento de la fila 3 y la columna 1
         A[2][0]
```

Out[72]: 4

```
In [73]: # Se puede modificar directamente
         A[2][0] = -8
```

In [74]:
```
A   # Matriz A modificada
```
Out[74]:
```
[[5, 1, 2], [7, 3, 7], [-8, 7, 8]]
```

2. En un array de numpy

In [75]:
```
C # Previamente fue definida la matriz C.
```
Out[75]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

In [76]:
```
# Modificando el elemento C[2, 1]
C[2, 1] = -15
```

In [77]:
```
C   # Matriz C modificada
```
Out[77]:
```
array([[  0,   1,   2,   3],
       [  4,   5,   6,   7],
       [  8, -15,  10,  11],
       [ 12,  13,  14,  15]])
```

In [78]:
```
# Así modifico toda la fila 2 simultáneamente
C[2] = 0
```

In [79]:
```
C
```
Out[79]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 0,  0,  0,  0],
       [12, 13, 14, 15]])
```

In [80]:
```
# Por ejemplo, así se modifican todos los elementos.
C[:, :] = 0
```

In [81]:
```
C
```
Out[81]:
```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

3. En una matriz de sympy

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

Out[82]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 4 & -2 \\ 2 & 0 & 8 \end{bmatrix}$$

In [83]:
```python
# Modificando el elemento K[1, 1]
K[1, 1] = 0
```

In [84]:
```python
K  # Matriz K modificada
```

Out[84]:

$$\begin{bmatrix} -1 & 1 & 9 \\ 4 & 0 & -2 \\ 2 & 0 & 8 \end{bmatrix}$$

In [85]:
```python
# Pilas que así se modifica el elemento 2 del recorrido fila a fila.
K[2] = 1
```

In [86]:
```python
K
```

Out[86]:

$$\begin{bmatrix} -1 & 1 & 1 \\ 4 & 0 & -2 \\ 2 & 0 & 8 \end{bmatrix}$$

In [87]:
```python
# Si se quiere modificar la columna 1.
K.col(1)
```

Out[87]:

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

In [88]:
```python
# Se genera error.
K.col(1) = -100
```

```
  File "<ipython-input-88-56f824f47dfd>", line 2
    K.col(1) = -100
    ^
SyntaxError: cannot assign to function call
```

In [89]:
```python
# La forma alternativa de llamar la columna 1.
K[:,1]
```

Out[89]:

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

In [90]:
```python
# Si se trata de modificar
K[:,1] = -100
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-90-dedd104c22b5> in <module>
      1 # Si se trata de modificar
----> 2 K[:,1] = -100

C:\ProgramData\Anaconda3\lib\site-packages\sympy\matrices\dense.py in __setitem__(self, key, value)
    356                  [2, 2, 4, 2]])
    357         """
--> 358         rv = self._setitem(key, value)
    359         if rv is not None:
    360             i, j, value = rv

C:\ProgramData\Anaconda3\lib\site-packages\sympy\matrices\matrices.py in _setitem(self, key, value)
   1177                 self.copyin_list(key, value)
   1178                 return
-> 1179             raise ValueError('unexpected value: %s' % value)
   1180         else:
   1181             if (not is_mat and

ValueError: unexpected value: -100
```

In [ ]:
```python
# Una forma correcta es:
K[:,1] = (-100, -100, -100)
```

In [91]:
```python
K
```

Out[91]:

$$\begin{bmatrix} -1 & 1 & 1 \\ 4 & 0 & -2 \\ 2 & 0 & 8 \end{bmatrix}$$

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 A y B es denotada por: $A + B$

Usando numpy

In [92]:
```
# 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 [93]:
```
M+N   # La suma de las matrices M y N.
```

Out[93]:
```
array([[-1,  7],
       [ 8,  4]])
```

¿Qué pasa si las matrices a sumar son diferente tamaño?

In [94]:
```
B   # Llamando la matriz B, previmante definida
```

Out[94]:
```
array([[3, 1, 9],
       [2, 4, 1]])
```

In [95]:
```python
B+N  # Lo que ocurre si se suman matrices de diferente tamaño.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-95-fd3fd26b354f> in <module>
----> 1 B+N  # Lo que ocurre si se suman matrices de diferente tamaño.

ValueError: operands could not be broadcast together with shapes (2,3) (2,2)
```

Usando sympy

In [96]:
```python
# Se crean algunas variables simbólicas con sympy.
a, b, c, d = sp.symbols("a b c d")
```

In [97]:
```python
# Usando el módulo sympy se crean las matrices O y P.
O = sp.Matrix([
               [ a, c],
               [ b, d],
                     ])

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

In [98]:
```python
O-P  # La resta de las matrices O y P.
```

Out[98]:

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

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 A y B es denotada por: A * B

Usando numpy

In [99]:
```python
# Usando el módulo numpy se crean las matrices Q y R.
Q = np.array([
              [3, -2],
                   ])
```

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

In [100]:
```
np.matmul(Q, R)  # La operación Q*R hace otra cosa completamente diferent
```

Out[100]:
```
array([[11]])
```

In [101]:
```
Q@R  # Otra forma más sencilla de hacer la misma operación.
```

Out[101]:
```
array([[11]])
```

Los arreglos en numpy permiten realizar sobre ellos las operaciones básicas **elemento a elemento**, lo cual no responde a las reglas de las operaciones con matrices.

In [102]:
```
N + 100  # Es equivalente a la suma de una matriz de unos multiplicada po
```

Out[102]:
```
array([[ 97, 100],
       [109, 101]])
```

In [103]:
```
N * 3  # Es equivalente a N+N+N
```

Out[103]:
```
array([[-9,  0],
       [27,  3]])
```

In [104]:
```
N**2  # Matricialmente no es lo mimso que NxN. Porque lo que hace es elev
```

Out[104]:
```
array([[ 9,  0],
       [81,  1]], dtype=int32)
```

In [105]:
```
(N+2)**2/8
```

Out[105]:
```
array([[ 0.125,  0.5  ],
       [15.125,  1.125]])
```

In [106]:
```
1/N  # Cuidado, que esta operacción no es la matriz inversa.
```

```
<ipython-input-106-a51a79dbb120>:1: RuntimeWarning: divide by zero enc
ountered in true_divide
  1/N  # Cuidado, que esta operacción no es la matriz inversa.
```

Out[106]:
```
array([[-0.33333333,         inf],
       [ 0.11111111,  1.        ]])
```

In [107]:
```
# ¿Qué ocurre si se eleva a potencias negativas.
N**-1
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

In [108]:
```
Q*R # De acuerdo con las reglas de las operaciones entre matrices, deberí
```

Out[108]:
```
array([[ 9, -6],
       [-3,  2]])
```

Usando sympy

In [109]:
```
# 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, d],
                       ])

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

In [110]:
```
A
```

Out[110]:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

In [111]:
```
X
```

Out[111]:

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

In [112]:
```
A*X   # En sympy sí se puede hacer esta operación así.
```

Out[112]:

$$\begin{bmatrix} ax + by \\ cx + dy \end{bmatrix}$$

Algunas consideraciones sobre las operaciones:

In [113]:
```
O  # Llamando la matriz previamente definida.
```

Out[113]:

$$\begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

In [114]:
```
# Nótese que aquí sí se respetan las reglas de las operaciones entre matr
A+10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-114-5324d02d0726> in <module>
      1 # Nótese que aquí sí se respetan las reglas de las operaciones entre matrices.
----> 2 A+10

C:\ProgramData\Anaconda3\lib\site-packages\sympy\core\decorators.py in binary_op_wrapper(self, other)
    134                     if f is not None:
    135                         return f(self)
--> 136             return func(self, other)
    137         return binary_op_wrapper
    138     return priority_decorator

C:\ProgramData\Anaconda3\lib\site-packages\sympy\matrices\common.py in __add__(self, other)
   2694             return MatrixArithmetic._eval_add(self, other)
   2695 
-> 2696         raise TypeError('cannot add %s and %s' % (type(self), type(other)))
   2697 
   2698     @call_highest_priority('__rtruediv__')

TypeError: cannot add <class 'sympy.matrices.dense.MutableDenseMatrix'> and <class 'int'>
```

In [115]:
```
# Las matrices se pueden multiplicar por escalares.
O*15
```

Out[115]:

$$\left[\begin{matrix}15a & 15c\\15b & 15d\end{matrix}\right]$$

In [116]:
```
# Esta operación sí representa la multiplicación de O con O.
O**2
```

Out[116]:

$$\left[\begin{matrix}a^{2} + bc & ac + cd\\ab + bd & bc + d^{2}\end{matrix}\right]$$

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 A es denotada por A^{-1}.

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

Usando numpy

In [117]:
```
# Usando el módulo numpy se crea la matriz S.
S = np.array([[ 1, -2],
              [-3, 5]])
```

In [118]:
```
S  # Se imprime S
```

Out[118]:
```
array([[ 1, -2],
       [-3,  5]])
```

In [119]:
```
np.linalg.det(S)  # Se determina el determinante de la matriz S.
```

Out[119]:
```
-1.0000000000000004
```

In [120]:
```
# Como el determinante es diferente de 0, la matriz es inversible.
np.linalg.inv(S)  # Inversa de la matriz S.
```

Out[120]:
```
array([[-5., -2.],
       [-3., -1.]])
```

¿Qué ocurre si se trata de sacar la inversa de una matriz singular?

In [121]:
```
W = np.array([[ 1, -2],
              [-2,  4]])
```

In [122]:
```
np.linalg.det(W)
```

Out[122]:
```
0.0
```

In [123]:
```
np.linalg.inv(T)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-123-479ac258ede6> in <module>
----> 1 np.linalg.inv(T)

NameError: name 'T' is not defined
```

Usando sympy

In [124]:
```
# Usando la matriz A previamente creada con varibles simbólicas.
```

```
A
```

Out[124]:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

In [125]:
```
# Cálculo del determinante.
A.det()
```

Out[125]:

$$ad - bc$$

In [126]:
```
A.inv()  # Inversa de la matriz A. Aquí la utilidad del módulo sympy para
```

Out[126]:

$$\begin{bmatrix} \frac{d}{ad-bc} & -\frac{b}{ad-bc} \\ -\frac{c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix}$$

In [127]:
```
A**-1  # Esta operación sí es la inversa de la matriz, al contrario de lo
```

Out[127]:

$$\begin{bmatrix} \frac{d}{ad-bc} & -\frac{b}{ad-bc} \\ -\frac{c}{ad-bc} & \frac{a}{ad-bc} \end{bmatrix}$$

4. Matriz transpuesta

Se denota a la transpuesta de la matriz A como A^T.

¿Cómo se transpone la matriz A ? (ejercicio de clase con lista de listas)

In [128]:
```
A = [[  0,  1,  2,  3,  4],
     [  5,  6,  7,  8,  9],
     [ 10, 11, 12, 13, 14]]
```

In [129]:
```
# Solución
```

Usando numpy

In [130]:
```
B # Previamente fue definida la matriz B.
```

Out[130]:
```
array([[3, 1, 9],
       [2, 4, 1]])
```

In [131]:
```
B.T   # Matriz transpuesta.
```
Out[131]:
```
array([[3, 2],
       [1, 4],
       [9, 1]])
```

Usando sympy

In [132]:
```
K  # Previamente fue definida la matriz K.
```
Out[132]:

$$\begin{bmatrix} -1 & 1 & 1 \\ 4 & 0 & -2 \\ 2 & 0 & 8 \end{bmatrix}$$

In [133]:
```
K.T   # Matriz transpuesta.
```
Out[133]:

$$\begin{bmatrix} -1 & 4 & 2 \\ 1 & 0 & 0 \\ 1 & -2 & 8 \end{bmatrix}$$

5. Producto punto y producto cruz

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

$$\underline{A} . \underline{B} = (a_1, a_2, \ldots, a_n). (b_1, b_2, \ldots, b_n) = a_1b_1 + a_2b_2 + \ldots + a_nb_n = \sum a_ib_i$$

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: $|\underline{\underline{A}}|$.

Pregunta de clase

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

Pregunta de clase

¿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 $\underline{A} \times \underline{B}$, en donde:

$\underline{A} = (a_x, a_y, a_z)$

$\underline{B} = (b_x, b_y, b_z)$

es decir,

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

In [134]:
```
# 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[134]:
$$\hat{i}\left(a_y b_z - a_z b_y\right) + \hat{j}\left(-a_x b_z + a_z b_x\right) + \hat{k}\left(a_x b_y - a_y b_x\right)$$

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

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

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

Además, es últil la definición:

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

Pregunta de clase

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

Usando numpy

In [135]:
```
# Se crean los vectores u1 y u2.
u1 = np.array([7, -4, -1])
u2 = np.array([3, -5,  2])
```

In [136]:
```
u1  # Se visualiza el vector u1.
```

Out[136]:
```
array([ 7, -4, -1])
```

In [137]:
```
u2  # Se visualiza el vector u2.
```

Out[137]:
```
array([ 3, -5,  2])
```

In [138]:
```
# El producto punto de u1 y u2.
np.dot(u1, u2)
```

Out[138]:
```
39
```

Usando sympy

In [139]:
```
# 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 [140]:
```
v1  # Se visualiza el vector v1.
```

Out[140]:

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

In [141]:
```
v2  # Se visualiza el vector v2.
```

Out[141]:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

In [142]:
```
# El producto punto de v1 y v2.
v1.dot(v2)  # Es lo mismo que v2.dot(v1)
```

Out[142]: $a_1 b_1 + a_2 b_2 + a_3 b_3$

In [143]:
```
# El producto cruz de v1 y v2.
```

```python
v3 = v1.cross(v2)

v3  # Se imprime v3.
```

Out[143]:

$$\begin{bmatrix} a_2b_3 - a_3b_2 \\ -a_1b_3 + a_3b_1 \\ a_1b_2 - a_2b_1 \end{bmatrix}$$

In [144]:
```python
# El producto punto de v1 y v3.
v1.dot(v3)   # Debería dar 0.
```

Out[144]:

$$a_1\left(a_2b_3 - a_3b_2\right) + a_2\left(-a_1b_3 + a_3b_1\right) + a_3\left(a_1b_2 - a_2b_1\right)$$

In [145]:
```python
sp.simplify(v1.dot(v3))   # Se usa el comando "simplify".
```

Out[145]: 0

In [146]:
```python
# El producto punto de v2 y v2.
sp.simplify(v2.dot(v3))
```

Out[146]: 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(__)