Introducción a python para computación numérica III

Importamos el módulo **`numpy`** .

```
import numpy as np
```

El método de Gauss para sistemas lineales tridiagonales

Un sistema tridiagonal tiene por matriz de coeficientes una matriz tridiagonal, que es una matriz donde todos los elementos son cero excepto los de la diagonal principal y las adyacentes por encima y por debajo. Por ejemplo, el sistema lineal siguiente

$$
\begin{array}{rrrrrrrcl}
a_{11}x_1 & +a_{12}x_2 & & & & & & = & b_1 \\
a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & & & & & = & b_2 \\
 & a_{32}x_2 & +a_{33}x_3 & +a_{34}x_4 & & & & = & b_3 \\
 & & a_{43}x_3 & +a_{44}x_4 & +a_{45}x_5 & & & = & b_4 \\
 & & & a_{54}x_4 & +a_{55}x_5 & +a_{56}x_6 & & = & b_5 \\
 & & & & a_{65}x_5 & +a_{66}x_6 & +a_{67}x_7 & = & b_6 \\
 & & & & & a_{76}x_6 & +a_{77}x_7 & = & b_7
\end{array}
$$

tiene por matriz de coeficientes y términos independientes

$$
A = \begin{pmatrix}
a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 \\
a_{21} & a_{22} & a_{23} & 0 & 0 & 0 & 0 \\
0 & a_{32} & a_{33} & a_{34} & 0 & 0 & 0 \\
0 & 0 & a_{43} & a_{44} & a_{45} & 0 & 0 \\
0 & 0 & 0 & a_{54} & a_{55} & a_{56} & 0 \\
0 & 0 & 0 & 0 & a_{65} & a_{66} & a_{67} \\
0 & 0 & 0 & 0 & 0 & a_{76} & a_{77}
\end{pmatrix}
\qquad
b = \begin{pmatrix}
b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7
\end{pmatrix}
$$

Por Gauss, resolvemos el sistema en dos pasos:

- Triangularización de la matriz aumentada.
- Sustitución regresiva.

Triangularización

Tenemos que hacer ceros por debajo de la diagonal principal utilizando operaciones por filas. Para el ejemplo que estamos viendo

$$\text{Si } k=1 \quad f \leftarrow \frac{a_{21}}{a_{11}} \text{ entonces} \qquad a_{21} \leftarrow a_{21} - f\,a_{11} = 0 \qquad a_{22} \leftarrow a_{22} - f\,a_{12} \qquad b_2 \leftarrow$$

$$\text{Si } k=2 \quad f \leftarrow \frac{a_{32}}{a_{22}} \text{ entonces} \qquad a_{32} \leftarrow a_{32} - f\,a_{22} = 0 \qquad a_{33} \leftarrow a_{33} - f\,a_{23} \qquad b_3 \leftarrow$$

$$\text{Si } k=3 \quad f \leftarrow \frac{a_{43}}{a_{33}} \text{ entonces} \qquad a_{43} \leftarrow a_{43} - f\,a_{33} = 0 \qquad a_{44} \leftarrow a_{44} - f\,a_{34} \qquad b_4 \leftarrow$$

$$\text{Si } k=4 \quad f \leftarrow \frac{a_{54}}{a_{44}} \text{ entonces} \qquad a_{54} \leftarrow a_{54} - f\,a_{44} = 0 \qquad a_{55} \leftarrow a_{55} - f\,a_{45} \qquad b_5 \leftarrow$$

$$\text{Si } k=5 \quad f \leftarrow \frac{a_{65}}{a_{55}} \text{ entonces} \qquad a_{65} \leftarrow a_{65} - f\,a_{55} = 0 \qquad a_{66} \leftarrow a_{66} - f\,a_{56} \qquad b_6 \leftarrow$$

$$\text{Si } k=6 \quad f \leftarrow \frac{a_{76}}{a_{66}} \text{ entonces} \qquad a_{76} \leftarrow a_{76} - f\,a_{66} = 0 \qquad a_{77} \leftarrow a_{77} - f\,a_{67} \qquad b_7 \leftarrow$$

Ejercicio 1

Dado un sistema tridiagonal, usando un bucle **for**, escribir una función **`At, bt = triangulariza(A,b)`** que devuelva las matrices **`At`** y **`bt`** que son las matrices transformadas de **`A`** y **`b`** tras el proceso de triangularización.

- Usar, para que la impresión de las matrices sea más legible

```
np.set_printoptions(precision = 2)   # solo dos decimales
np.set_printoptions(suppress = True) # no usar notación exponencial
```

- Extraer las dimensiones de **`A`** con `shape`. Por ejemplo **`m, n = A.shape`**. También se puede obtener el número de elementos de un vector con `len`. Por ejemplo **`n = len(b)`**.
- Inicializar la matriz triangularizada **`At = np.copy(A)`** si no queremos modificar la matriz **`A`**. Analogamente con **`b`**.
- Triangularizar el sistema $Ax = b$ donde las matrices **`A`** y **`b`** se generan con el código

```
n = 7

A1 = np.diag(np.ones(n))*3
A2 = np.diag(np.ones(n-1),1)
A = A1 + A2 + A2.T

b = np.arange(n,2*n)*1.
```

- Sin modificar la función, triangularizar también el sistema de matrices **`A`** y **`b`** generadas con el código

```
n = 8

np.random.seed(3)
A1 = np.diag(np.random.rand(n))
A2 = np.diag(np.random.rand(n-1),1)
A = A1 + A2 + A2.T

b = np.random.rand(n)
```

```
%run Ejercicio1.py
```

```
-------------  DATOS  -------------
A
[[3. 1. 0. 0. 0. 0. 0.]
 [1. 3. 1. 0. 0. 0. 0.]
 [0. 1. 3. 1. 0. 0. 0.]
 [0. 0. 1. 3. 1. 0. 0.]
 [0. 0. 0. 1. 3. 1. 0.]
 [0. 0. 0. 0. 1. 3. 1.]
 [0. 0. 0. 0. 0. 1. 3.]]
b
[ 7.  8.  9. 10. 11. 12. 13.]



----  SISTEMA TRIANGULARIZADO ----
At
[[3.   1.   0.   0.   0.   0.   0.  ]
 [0.   2.67 1.   0.   0.   0.   0.  ]
 [0.   0.   2.62 1.   0.   0.   0.  ]
 [0.   0.   0.   2.62 1.   0.   0.  ]
 [0.   0.   0.   0.   2.62 1.   0.  ]
 [0.   0.   0.   0.   0.   2.62 1.  ]
 [0.   0.   0.   0.   0.   0.   2.62]]
bt
[7.   5.67 6.88 7.38 8.18 8.88 9.61]



-------------  DATOS  -------------
A
[[0.55 0.05 0.   0.   0.   0.   0.   0.  ]
 [0.05 0.71 0.44 0.   0.   0.   0.   0.  ]
 [0.   0.44 0.29 0.03 0.   0.   0.   0.  ]
 [0.   0.   0.03 0.51 0.46 0.   0.   0.  ]
 [0.   0.   0.   0.46 0.89 0.65 0.   0.  ]
 [0.   0.   0.   0.   0.65 0.9  0.28 0.  ]
 [0.   0.   0.   0.   0.   0.28 0.13 0.68]
 [0.   0.   0.   0.   0.   0.   0.68 0.21]]
b
[0.59 0.02 0.56 0.26 0.42 0.28 0.69 0.44]



----  SISTEMA TRIANGULARIZADO ----
At
[[ 0.55  0.05  0.    0.    0.    0.    0.    0.  ]
 [ 0.    0.7   0.44  0.    0.    0.    0.    0.  ]
 [ 0.    0.    0.01  0.03  0.    0.    0.    0.  ]
 [ 0.    0.    0.    0.45  0.46  0.    0.    0.  ]
 [ 0.    0.    0.    0.    0.43  0.65  0.    0.  ]
 [ 0.    0.    0.    0.    0.   -0.09  0.28  0.  ]
 [ 0.    0.    0.    0.    0.    0.    1.03  0.68]
 [ 0.    0.    0.    0.    0.    0.    0.   -0.24]]
bt
[ 0.59 -0.03  0.58 -0.92  1.35 -1.76 -5.01  3.74]
```

Sustitución regresiva

Tras la triangularización, tenemos el sistema equivalente $A^*x = b^*$

$$
\begin{array}{rrrrrrrcl}
a^*_{11}x_1 & +a^*_{12}x_2 & & & & & & = & b^*_1 \\
 & a^*_{22}x_2 & +a^*_{23}x_3 & & & & & = & b^*_2 \\
 & & a^*_{33}x_3 & +a^*_{34}x_4 & & & & = & b^*_3 \\
 & & & a^*_{44}x_4 & +a^*_{45}x_5 & & & = & b^*_4 \\
 & & & & a^*_{55}x_5 & +a^*_{56}x_6 & & = & b^*_5 \\
 & & & & & a^*_{66}x_6 & +a^*_{67}x_7 & = & b^*_6 \\
 & & & & & & a^*_{77}x_7 & = & b^*_7
\end{array}
$$

Despejamos las incógnitas de abajo a arriba, empezando por la última

$$\text{Si } k = 7 \qquad x_7 \leftarrow \frac{b^*_7}{a^*_{77}}$$

$$\text{Si } k = 6 \qquad x_6 \leftarrow \frac{b^*_6 - a^*_{67}x_7}{a^*_{66}}$$

$$\text{Si } k = 5 \qquad x_5 \leftarrow \frac{b^*_5 - a^*_{56}x_6}{a^*_{55}}$$

$$\text{Si } k = 4 \qquad x_4 \leftarrow \frac{b^*_4 - a^*_{45}x_5}{a^*_{44}}$$

$$\text{Si } k = 3 \qquad x_3 \leftarrow \frac{b^*_3 - a^*_{34}x_4}{a^*_{33}}$$

$$\text{Si } k = 2 \qquad x_2 \leftarrow \frac{b^*_2 - a^*_{23}x_3}{a^*_{22}}$$

$$\text{Si } k = 1 \qquad x_1 \leftarrow \frac{b^*_1 - a^*_{12}x_2}{a^*_{11}}$$

Ejercicio 2

Usando un bucle **for**, escribir una función `x = sust_reg(At,bt)` que resuelva un sistema triangular como el anterior por sustitución regresiva. Utilizar como entrada las matrices transformadas obtenidas con la función de triangularización.

Nota:

- Se puede usar un bucle con paso negativo. Por ejemplo

```
for k in range(n1,n2,-1):
```

donde `n1` es el valor inicial `n2` el valor siguiente al final y `-1` el paso.

```
%run Ejercicio2.py
```

```
x
[1.86 1.42 1.89 1.91 2.37 1.99 3.67]



x
[ -3.    43.59 -69.63  53.46 -54.66  38.2    5.47 -15.72]
```

Almacenamiento de los coeficientes en una matriz rectangular de tres columnas

Almacenemos ahora los coeficientes en una matriz rectangular. El sistema era

$$\begin{array}{rrrrrrrcl}
a_{11}x_1 & +a_{12}x_2 & & & & & & = & b_1 \\
a_{21}x_1 & +a_{22}x_2 & +a_{23}x_3 & & & & & = & b_2 \\
 & a_{32}x_2 & +a_{33}x_3 & +a_{34}x_4 & & & & = & b_3 \\
 & & a_{43}x_3 & +a_{44}x_4 & +a_{45}x_5 & & & = & b_4 \\
 & & & a_{54}x_4 & +a_{55}x_5 & +a_{56}x_6 & & = & b_5 \\
 & & & & a_{65}x_5 & +a_{66}x_6 & +a_{67}x_7 & = & b_6 \\
 & & & & & a_{76}x_6 & +a_{77}x_7 & = & b_7
\end{array}$$

Si guardamos los coeficientes en una matriz rectangular

$$A_r = \begin{pmatrix}
0 & a_{11} & a_{12} \\
a_{21} & a_{22} & a_{23} \\
a_{32} & a_{33} & a_{34} \\
a_{43} & a_{44} & a_{45} \\
a_{54} & a_{55} & a_{56} \\
a_{65} & a_{66} & a_{67} \\
a_{76} & a_{77} & 0
\end{pmatrix} = \begin{pmatrix}
0 & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33} \\
r_{41} & r_{42} & r_{43} \\
r_{51} & r_{52} & r_{53} \\
r_{61} & r_{62} & r_{63} \\
r_{71} & r_{72} & 0
\end{pmatrix}$$

$$\text{Si } k=1 \quad f \leftarrow \frac{a_{21}}{a_{11}} \text{ entonces} \qquad a_{21} \leftarrow a_{21} - f\,a_{11} = 0 \qquad a_{22} \leftarrow a_{22} - f\,a_{12} \qquad b_2 \leftarrow$$

$$\text{Si } k=2 \quad f \leftarrow \frac{a_{32}}{a_{22}} \text{ entonces} \qquad a_{32} \leftarrow a_{32} - f\,a_{22} = 0 \qquad a_{33} \leftarrow a_{33} - f\,a_{23} \qquad b_3 \leftarrow$$

$$\text{Si } k=3 \quad f \leftarrow \frac{a_{43}}{a_{33}} \text{ entonces} \qquad a_{43} \leftarrow a_{43} - f\,a_{33} = 0 \qquad a_{44} \leftarrow a_{44} - f\,a_{34} \qquad b_4 \leftarrow$$

$$\text{Si } k=4 \quad f \leftarrow \frac{a_{54}}{a_{44}} \text{ entonces} \qquad a_{54} \leftarrow a_{54} - f\,a_{44} = 0 \qquad a_{55} \leftarrow a_{55} - f\,a_{45} \qquad b_5 \leftarrow$$

$$\text{Si } k=5 \quad f \leftarrow \frac{a_{65}}{a_{55}} \text{ entonces} \qquad a_{65} \leftarrow a_{65} - f\,a_{55} = 0 \qquad a_{66} \leftarrow a_{66} - f\,a_{56} \qquad b_6 \leftarrow$$

$$\text{Si } k=6 \quad f \leftarrow \frac{a_{76}}{a_{66}} \text{ entonces} \qquad a_{76} \leftarrow a_{76} - f\,a_{66} = 0 \qquad a_{77} \leftarrow a_{77} - f\,a_{67} \qquad b_7 \leftarrow$$

Ahora, teniendo en cuenta la nueva forma de almacenar los coeficientes, será

$$\text{Si } k=1 \quad f \leftarrow \frac{r_{21}}{r_{12}} \text{ entonces} \qquad r_{21} \leftarrow r_{21} - f\,r_{12} = 0 \qquad r_{22} \leftarrow r_{22} - f\,r_{13} \qquad b_2 \leftarrow b$$

$$\text{Si } k=2 \quad f \leftarrow \frac{r_{31}}{r_{22}} \text{ entonces} \qquad r_{31} \leftarrow r_{31} - f\,r_{22} = 0 \qquad r_{32} \leftarrow r_{32} - f\,r_{23} \qquad b_3 \leftarrow b$$

$$\text{Si } k=3 \quad f \leftarrow \frac{r_{41}}{r_{32}} \text{ entonces} \qquad r_{41} \leftarrow r_{41} - f\,r_{32} = 0 \qquad r_{42} \leftarrow r_{42} - f\,r_{33} \qquad b_4 \leftarrow b$$

$$\text{Si } k=4 \quad f \leftarrow \frac{r_{51}}{r_{42}} \text{ entonces} \qquad r_{51} \leftarrow r_{51} - f\,r_{42} = 0 \qquad r_{52} \leftarrow r_{52} - f\,r_{43} \qquad b_5 \leftarrow b$$

$$\text{Si } k=5 \quad f \leftarrow \frac{r_{61}}{r_{52}} \text{ entonces} \qquad r_{61} \leftarrow r_{61} - f\,r_{52} = 0 \qquad r_{62} \leftarrow r_{62} - f\,r_{53} \qquad b_6 \leftarrow b$$

$$\text{Si } k=6 \quad f \leftarrow \frac{r_{71}}{r_{62}} \text{ entonces} \qquad r_{71} \leftarrow r_{71} - f\,r_{62} = 0 \qquad r_{72} \leftarrow r_{72} - f\,r_{63} \qquad b_7 \leftarrow b$$

Y la matriz triangulada queda

$$\begin{pmatrix} 0 & r^*_{12} & r^*_{13} \\ 0 & r^*_{22} & r^*_{23} \\ 0 & r^*_{32} & r^*_{33} \\ 0 & r^*_{42} & r^*_{43} \\ 0 & r^*_{52} & r^*_{53} \\ 0 & r^*_{62} & r^*_{63} \\ 0 & r^*_{72} & 0 \end{pmatrix}$$

Y ahora la sustitución regresiva será

$$\text{Si } k = 7 \qquad x_7 \leftarrow \frac{b^*_7}{r^*_{72}}$$

$$\text{Si } k = 6 \qquad x_6 \leftarrow \frac{b^*_6 - r^*_{63}x_7}{r^*_{62}}$$

$$\text{Si } k = 5 \qquad x_5 \leftarrow \frac{b^*_5 - r^*_{53}x_6}{r^*_{52}}$$

$$\text{Si } k = 4 \qquad x_4 \leftarrow \frac{b^*_4 - r^*_{43}x_5}{r^*_{42}}$$

$$\text{Si } k = 3 \qquad x_3 \leftarrow \frac{b^*_3 - r^*_{33}x_4}{r^*_{32}}$$

$$\text{Si } k = 2 \qquad x_2 \leftarrow \frac{b^*_2 - r^*_{23}x_3}{r^*_{22}}$$

$$\text{Si } k = 1 \qquad x_1 \leftarrow \frac{b^*_1 - r^*_{13}x_2}{r^*_{12}}$$

Ejercicio 3

En el segundo sistema, la matriz de coeficientes es tridiagonal. Este tipo de matrices dispersas (con muchos ceros) son habituales en problemas de la física (ecuación del calor, ecuación de onda) que se resuelven con diferencias finitas.

Por todo esto, se adaptan algoritmos ya existentes a estos casos particulares. Para ello, por ejemplo, en lugar de almacenar la matriz A del segundo sistema como una matriz cuadrada, podemos almacenar sólo las diagonales en una matriz rectangular A_r de la siguiente forma

```
n = 7

Ar = np.zeros((n,3))
Ar[:,0] = np.concatenate((np.array([0]),np.ones((n-1),)))
Ar[:,1] = np.ones((n),)*3
Ar[:,2] = np.concatenate((np.ones((n-1),),np.array([0])))

b = np.arange(n,2*n)*1.
```

Modificar el código de las funciones de los dos ejercicios anteriores de forma que admitan la matriz A en la forma A_r.

- Sin modificar las funciones, aplicarlas a los datos

```
n = 8

np.random.seed(3)
Ar = np.zeros((n,3))
Ar[:,1] = np.random.rand(n)
Ar[:,0] = np.concatenate((np.array([0]),np.random.rand(n-1)))
Ar[0:n-1,2] = Ar[1:n,0]

b = np.random.rand(n)
```

```
%run Ejercicio3.py
```

```
-------------  DATOS  -------------
Ar
[[0. 3. 1.]
 [1. 3. 1.]
 [1. 3. 1.]
 [1. 3. 1.]
 [1. 3. 1.]
 [1. 3. 1.]
 [1. 3. 0.]]
b
[ 7.  8.  9. 10. 11. 12. 13.]


----  SISTEMA TRIANGULARIZADO ----
At
[[0.   3.   1.  ]
 [0.   2.67 1.  ]
 [0.   2.62 1.  ]
 [0.   2.62 1.  ]
 [0.   2.62 1.  ]
 [0.   2.62 1.  ]
 [0.   2.62 0.  ]]


------------ SOLUCIÓN ------------
x
[1.86 1.42 1.89 1.91 2.37 1.99 3.67]




-------------  DATOS  -------------
Ar
[[0.   0.55 0.05]
 [0.05 0.71 0.44]
 [0.44 0.29 0.03]
 [0.03 0.51 0.46]
 [0.46 0.89 0.65]
 [0.65 0.9  0.28]
 [0.28 0.13 0.68]
 [0.68 0.21 0.  ]]
b
[0.59 0.02 0.56 0.26 0.42 0.28 0.69 0.44]


----  SISTEMA TRIANGULARIZADO ----
At
[[ 0.    0.55  0.05]
 [ 0.    0.7   0.44]
 [ 0.    0.01  0.03]
 [ 0.    0.45  0.46]
 [ 0.    0.43  0.65]
 [ 0.   -0.09  0.28]
 [ 0.    1.03  0.68]
 [ 0.   -0.24  0.  ]]


------------ SOLUCIÓN ------------
x
[ -3.    43.59 -69.63  53.46 -54.66  38.2    5.47 -15.72]
```

Ejercicios propuestos

Producto de matrices

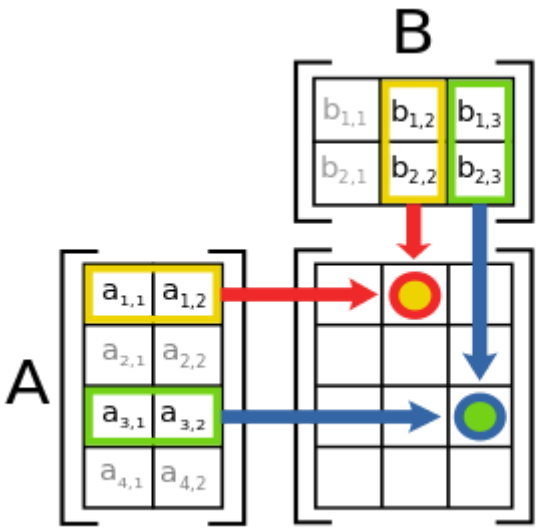
Utilizaremos tres matrices numpy

- A, matriz $m \times n$
- B, matriz $n \times p$
- C, matriz $m \times p$

Como vamos a multiplicar A por B, el número de columnas de A, n, ha de ser igual al número de filas de B. Y la matriz resultante C, ha de tener las mismas filas que A y las mismas columnas que B.

Un elemento de la matriz producto C viene dado por

$$c_{ij} = a_{i1}\,b_{1j} + a_{i2}\,b_{2j} + \cdots + a_{in}\,b_{nj} = \sum_{k=1}^{n} a_{ik}\,b_{kj} \quad \text{para } i = 1, \ldots, m \quad j = 1, \ldots, p \quad (1)$$



(By File:Matrix multiplication diagram.svg (//commons.wikimedia.org/wiki/File:Matrix_multiplication_diagram.svg))

Ejercicio 4

Utilizando tres bucles **for**, uno con el índice **i** de la fila, otro con el índice **j** de la columna y otro índice **k** para la suma acumulada, construir una función **C = multiplicar_3bucles(A,B)** que multiplique las matrices **A** y **B** y devuelva la matriz producto **C** .

- Extraer las dimensiones de **A** y **B** con shape . Por ejemplo **m, n = A.shape**
- Inicializar la matriz **C = np.zeros((m,p))** siendo **m** el número de filas de **A** y **p** el número de columnas de **B**
- Cada elemento de la matriz **C** ha sido inicializado a cero y se construye como una suma acumulada. Es decir, empezamos con $c_{ij} = 0$. Luego

$$\begin{aligned} k = 1 \qquad & c_{ij} \leftarrow c_{ij} + a_{i1} b_{1j} \\ k = 2 \qquad & c_{ij} \leftarrow c_{ij} + a_{i2} b_{2j} \\ & \dots \\ k = n \qquad & c_{ij} \leftarrow c_{ij} + a_{in} b_{nj} \end{aligned}$$

- Utilizar la función para multiplicar las matrices

```
A = np.array([[-3.,2],[-2,0],[-4,4],[4,-4]])
B = np.array([[4.,-3,1],[-2,1,1]])
```

- Sin cambiar la función, utilizarla para multiplicar las matrices

```
A1 = np.array([[-3.,2],[-2,0],[-4,4],[4,-4],[1,1]])
B1 = np.array([[4.,-3,1,1],[-2,1,1,1]])
```

```
%run Ejercicio4.py
```

```
C
 [[-16.  11.  -1.]
 [ -8.   6.  -2.]
 [-24.  16.   0.]
 [ 24. -16.   0.]]

C1
 [[-16.  11.  -1.  -1.]
 [ -8.   6.  -2.  -2.]
 [-24.  16.   0.   0.]
 [ 24. -16.   0.   0.]
 [  2.  -2.   2.   2.]]
```

Ejercicio 5

Podemos obtener el producto de forma más eficaz quitando el bucle interior y sustituyéndolo por una sola línea de código.

- La fila `i` de `A` es `A[i,:]`, que es un vector de n elementos.
- La columna `j` de `B` es `B[:,j]`, que también es un vector de n elementos.
- Los multiplicamos elemento a elemento `A[i,:]*B[:,j]` y obtenemos un nuevo vector de n elementos.
- Sumamos los elementos de este vector, producto de dos vectores, `np.sum(A[i,:]*B[:,j])`

Construir una función `multiplicar_2bucles(A,B)` que multiplique las matrices `A` y `B` y devuelva la matriz producto `C` usando solo dos bucles. Multiplicar otra vez las matrices del **Ejercicio 4**.

```
%run Ejercicio5.py
```

```
C
 [[-16.  11.  -1.]
 [ -8.   6.  -2.]
 [-24.  16.   0.]
 [ 24. -16.   0.]]

C1
 [[-16.  11.  -1.  -1.]
 [ -8.   6.  -2.  -2.]
 [-24.  16.   0.   0.]
 [ 24. -16.   0.   0.]
 [  2.  -2.   2.   2.]]
```

Podemos aumentar el grado de vectorización obteniendo una columna (o fila) de la matriz `C` por línea de código. Así quitaríamos otro bucle.

- `B[:,j]` es un vector de n componentes que contiene la columna `j` de `B`.
- En `D = A*B[:,j]` cada fila de `D` es el producto elemento a elemento de cada fila de `A` con el vector `B[:,j]`. Por lo tanto, `D` tiene el mismo tamaño que `A`.
- `C[:,j] = np.sum(D,axis=1)` es una matriz column donde cada elemento es la suma de los elementos de una fila de `D` y el el producto matricial de la matriz `A` por la matriz columna `B[:,j]`.

```
A = np.array([[-3.,2],[-2,0],[-4,4],[4,-4]])
B = np.array([[4.,-3,1],[-2,1,1]])

m = A.shape[0]
p = B.shape[1]

C = np.zeros((m,p))

for j in range(p):
    C[:,j] = np.sum(A*B[:,j],axis=1)

print(C)
```

```
[[-16.  11.  -1.]
 [ -8.   6.  -2.]
 [-24.  16.   0.]
 [ 24. -16.   0.]]
```

Ejercicio 6

Escribir una función **`C = multiplica_1bucle(A,B)`** . Multiplicar otra vez las matrices del **Ejercicio 4**.

```
%run Ejercicio6.py
```

```
C
 [[-16.  11.  -1.]
 [ -8.   6.  -2.]
 [-24.  16.   0.]
 [ 24. -16.   0.]]

C1
 [[-16.  11.  -1.  -1.]
 [ -8.   6.  -2.  -2.]
 [-24.  16.   0.   0.]
 [ 24. -16.   0.   0.]
 [  2.  -2.   2.   2.]]
```

Ejercicio 7

Comparemos el tiempo de ejecución de las tres funciones de multiplicar matrices y el comando numpy **`np.dot(A,B)`** . Usar las matrices $A_{m\times n}$ y $B_{n\times p}$ con elementos obtenidos alatoriamente. Usar el comando **`time.time()`** visto en la práctica anterior para calcular el tiempo de ejecución. **`np.random.rand(m,n)`** crea una matriz de elementos aleatorios distribuidos uniformemente en $[0,1]$.

```
m = 300; n = 200; p = 400
A = np.random.rand(m,n)
B = np.random.rand(n,p)
```

```
%run Ejercicio7.py
```

```
Tiempo de ejecución con tres bucles
17.99861788749695
Tiempo de ejecución con dos bucles
0.7922441959381104
Tiempo de ejecución con un bucle
0.042597055435180664
Tiempo de ejecución con el comando dot
0.0010800361633300781
```