Laboratorio 1: Soluciones de los ejercicios

En este documento se recopilan soluciones orientativas a los ejercicios del Laboratorio 1. Se muestran tanto las expresiones matemáticas como ejemplos de código en Python/NumPy.

1. Vectores y Matrices

Ejercicio 1

Enunciado. Definir en Python (usando NumPy) las matrices y vectores

$$A = \begin{pmatrix} -1 & \sqrt{5} & 2 & \pi \\ 0 & 3 & -1 & 4 \\ \pi & \sqrt[3]{5} & 0 & 1 \end{pmatrix}, \quad B = \begin{pmatrix} 3 & \frac{2}{5} & 0 \\ 0 & -1 & 4 \\ \pi & 0 & 1 \end{pmatrix},$$

$$v = \begin{pmatrix} 4 \\ -3 \\ 2 \end{pmatrix}, \quad w = \begin{pmatrix} \sqrt{7} \\ 0 \\ 2 \\ \pi \end{pmatrix}.$$

Solución (código en Python).

```
import numpy as np

A = np.array([[-1, np.sqrt(5), 2, np.pi],
              [ 0, 3,         -1, 4],
              [ np.pi, 5**(1/3), 0, 1]])

B = np.array([[3, 2/5, 0],
              [0, -1, 4],
              [np.pi, 0, 1]])

v = np.array([[4],
              [-3],
              [2]])

w = np.array([[np.sqrt(7)],
              [0],
              [2],
              [np.pi]])
```

Ejercicio 2

Enunciado. Realizar todos los productos matriz–matriz y matriz–vector que las dimensiones admitan.

Solución. Con las dimensiones indicadas,

$$A \in \mathbb{R}^{3\times 4}, \quad B \in \mathbb{R}^{3\times 3}, \quad v \in \mathbb{R}^{3\times 1}, \quad w \in \mathbb{R}^{4\times 1}.$$

Los productos válidos (del tipo matriz–matriz o matriz–vector) son:

$$B\,A, \quad B\,v, \quad A\,w.$$

En Python (usando `@` para el producto matricial):

```
BA = B @ A   # 3x3 por 3x4 -> 3x4
Bv = B @ v   # 3x3 por 3x1 -> 3x1
Aw = A @ w   # 3x4 por 4x1 -> 3x1
```

2. Sucesiones

Ejercicio. Consideramos la sucesión

$$2, 5, 8, 11, 14, 17, 20, 23, 26.$$

Definir esta sucesión de tres formas distintas y guardarlas bajo sendas variables `v1`, `v2` y `v3`.

Solución (código en Python).

```
import numpy as np

# Forma 1: arange con paso 3 (hasta 27 excluido)
v1 = np.arange(2, 27, 3)

# Forma 2: 2 + 3*k, k = 0,...,8
v2 = 2 + 3 * np.arange(9)

# Forma 3: linspace indicando número de puntos
v3 = np.linspace(2, 26, 9)  # 9 puntos entre 2 y 26
```

3. Funciones matemáticas y álgebra lineal

Ejercicio 1

Enunciado. Resolver el sistema, planteándolo de forma matricial:

$$\begin{aligned} 3x + 2y &= 0, \\ 2x - 2z + t &= 1, \\ y + 4z - 3t &= -2, \\ x + 5y - z - 3t &= 4. \end{aligned}$$

Solución. Escribimos el sistema como $A\mathbf{x} = \mathbf{b}$, donde

$$A = \begin{pmatrix} 3 & 2 & 0 & 0 \\ 2 & 0 & -2 & 1 \\ 0 & 1 & 4 & -3 \\ 1 & 5 & -1 & -3 \end{pmatrix}, \quad \mathbf{x} = \begin{pmatrix} x \\ y \\ z \\ t \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ 1 \\ -2 \\ 4 \end{pmatrix}.$$

En Python:

```
import numpy as np

A = np.array([[3, 2,  0,  0],
              [2, 0, -2,  1],
              [0, 1,  4, -3],
              [1, 5, -1, -3]])

b = np.array([0, 1, -2, 4])

x_sol = np.linalg.solve(A, b)
print(x_sol)
```

El resultado es

$$x = -\frac{14}{65}, \quad y = \frac{21}{65}, \quad z = -\frac{64}{65}, \quad t = -\frac{7}{13}.$$

Ejercicio 2

Enunciado. Dados los vectores

$$x = \begin{pmatrix} 1 \\ 2 \\ -3 \\ 5 \end{pmatrix}, \quad y = \begin{pmatrix} 4 \\ -4 \\ 1 \\ 2 \end{pmatrix}, \quad z = \begin{pmatrix} 3 \\ 1 \\ 2 \\ -4 \end{pmatrix},$$

formar una matriz cuyas dos primeras columnas sean todo ceros, sus siguientes tres columnas sean los vectores x, y, z y sus últimas columnas sean todo unos.

Solución. Una posibilidad es tomar dos columnas de unos, de forma que la matriz total tenga $2 + 3 + 2 = 7$ columnas. Simbólicamente:

$$M = \begin{pmatrix} 0 & 0 & 1 & 4 & 3 & 1 & 1 \\ 0 & 0 & 2 & -4 & 1 & 1 & 1 \\ 0 & 0 & -3 & 1 & 2 & 1 & 1 \\ 0 & 0 & 5 & 2 & -4 & 1 & 1 \end{pmatrix}.$$

En Python:

```
import numpy as np

x = np.array([[ 1],
              [ 2],
              [-3],
              [ 5]])

y = np.array([[ 4],
              [-4],
              [ 1],
              [ 2]])

z = np.array([[ 3],
              [ 1],
              [ 2],
              [-4]])

zeros_cols = np.zeros((4, 2))
ones_cols  = np.ones((4, 2))

M = np.hstack([zeros_cols, x, y, z, ones_cols])
```

Ejercicio 3

Enunciado. En la matriz del apartado anterior, acceder a la submatriz formada por las filas 1, 3 y 4 y las columnas 1, 2 y 6 (teniendo en cuenta que en Python los índices empiezan en 0). Calcular:

- el tamaño de esa submatriz (número de filas y columnas),
- la suma de todos sus elementos.

Solución. Con la matriz M anterior, las filas 1, 3 y 4 en numeración matemática corresponden a los índices 0, 2 y 3 de Python; las columnas 1, 2 y 6 corresponden a los índices 0, 1 y 5. Obtenemos la submatriz

$$S = M_{\{1,3,4\},\{1,2,6\}} = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}.$$

Por tanto, el tamaño de S es 3×3 y la suma de sus elementos es

$$0+0+1+0+0+1+0+0+1 = 3.$$

En Python:

```
filas = [0, 2, 3]      # filas 1, 3, 4 (notación matemática)
columnas = [0, 1, 5]   # columnas 1, 2, 6

S = M[np.ix_(filas, columnas)]
print("Submatriz S =\n", S)
print("Tamaño de S:", S.shape)
print("Suma de los elementos de S:", np.sum(S))
```

4. Funciones definidas por el usuario

Ejercicio 1

Enunciado. Crear una función (en un archivo `.py`) cuyos parámetros de entrada sean una matriz invertible `A` y un vector `b`, y que devuelva la solución del sistema $Ax = b$ usando `np.linalg.solve`.

Solución (código en Python).

```
import numpy as np

def resolver_sistema(A, b):
    """Resuelve el sistema lineal A x = b."""
    x = np.linalg.solve(A, b)
    return x

# Ejemplo de uso:
# A = np.array([[1, 2], [3, 4]])
# b = np.array([5, 6])
# sol = resolver_sistema(A, b)
# print(sol)
```

Ejercicio 2

Enunciado. Crear una función cuyos parámetros de entrada sean una matriz, un vector con tantas coordenadas como filas de la matriz y un número no mayor que el número de columnas de la matriz, y que devuelva la matriz sustituyendo la columna indicada por el número por el vector (utilizando indexado y asignación en NumPy).

Solución (código en Python).

```
import numpy as np

def sustituir_columna(A, v, k):
    """
    Sustituye la columna k de la matriz A por el vector v.
    Se asume 0 <= k < número de columnas de A
    y que v tiene el mismo número de filas que A.
    """
    B = A.copy()
    B[:, k] = v
    return B

# Ejemplo de uso:
# A = np.array([[1, 2, 3],
#               [4, 5, 6]])
# v = np.array([10, 20])
# B = sustituir_columna(A, v, 1)  # reemplaza la columna 1
# print(B)
```

Ejercicio 3

Enunciado. Crear una función cuyos parámetros sean una matriz y que devuelva:

- la suma de todos sus elementos,
- la suma de los elementos de la diagonal principal (traza).

Solución (código en Python).

```
import numpy as np

def suma_y_traza(A):
    """
    Devuelve la suma de todos los elementos de A
    y la suma de los elementos de la diagonal principal (traza).
```

```
    """
    suma_total = np.sum(A)
    traza = np.trace(A)
    return suma_total, traza

# Ejemplo de uso:
# A = np.array([[1, 2, 3],
#               [4, 5, 6],
#               [7, 8, 9]])
# s, t = suma_y_traza(A)
# print("Suma total:", s)   # 45
# print("Traza:", t)        # 1 + 5 + 9 = 15
```

Ejercicio 4

Enunciado. Crear una función que reciba como entrada un array de NumPy y devuelva dos valores: la suma de sus elementos y el valor medio (media aritmética). Probarla con un vector de 5 números.

Solución (código en Python).

```
import numpy as np

def suma_y_media(x):
    """
    Devuelve la suma de los elementos de x y su media aritmética.
    """
    suma = np.sum(x)
    media = suma / x.size
    return suma, media

# Ejemplo de uso:
# v = np.array([1, 2, 3, 4, 5])
# s, m = suma_y_media(v)
# print("Suma:", s)    # 15
# print("Media:", m)   # 3.0
```

Ejercicio 5

Enunciado. Crear una función que reciba como entrada un array bidimensional `A` y un número `k` y devuelva la suma de los elementos de la fila `k` y la suma de los elementos de la columna `k`. Probarla con una matriz 3×3 sencilla.

Solución (código en Python).

```
import numpy as np

def suma_fila_columna(A, k):
    """
    Devuelve la suma de la fila k y la suma de la columna k de A.
    Se asume 0 <= k < número de filas/columnas.
    """
    suma_fila = np.sum(A[k, :])
    suma_col  = np.sum(A[:, k])
    return suma_fila, suma_col

# Ejemplo de uso:
# A = np.array([[1, 2, 3],
#               [4, 5, 6],
#               [7, 8, 9]])
# sf, sc = suma_fila_columna(A, 1)  # fila y columna de índice 1
# print("Suma fila 1:", sf)   # 2 + 5 + 8 = 15
# print("Suma columna 1:", sc) # 4 + 5 + 6 = 15
```