torio 1: Introducción a Python, NumPy y Matp

A lo largo de estas sesiones de prácticas vamos a aprender a usar el lenguaje de programación Python junto con las librerías NumPy y Matplotlib. Python es un lenguaje muy utilizado en ingeniería y ciencia de datos, que va más allá de lo que puede hacer una calculadora, y nos permitirá realizar no sólo la mayor parte de los cálculos que estáis viendo en clase, sino que os dará mucha libertad para diseñar vuestros propios métodos y algoritmos para resolver los cálculos matemáticos que os encontraréis a lo largo de la carrera. Usaremos Python (junto con NumPy y Matplotlib) para:

- realizar operaciones aritméticas,
- programar en un lenguaje interpretado,
- trabajar con vectores y matrices de forma eficiente,
- realizar gran variedad de gráficos,
- ...

Introducción a NumPy y Matplotlib

Python se amplía mediante *librerías* (también llamadas *módulos* o *paquetes*). Una librería es un conjunto de funciones y herramientas ya programadas que podemos reutilizar en nuestros propios programas. En este laboratorio utilizaremos principalmente dos:

- **NumPy**: proporciona tipos de datos para vectores y matrices (arrays) y muchas funciones matemáticas optimizadas. Es la base del cálculo científico en Python.
- **Matplotlib**: permite crear gráficos de muchos tipos (curvas, nubes de puntos, barras, etc.) de forma similar a como se hace en programas científicos.

Supondremos que Python ya está instalado en el ordenador. Para instalar las librerías usaremos el gestor de paquetes `pip` desde la línea de comandos (terminal, PowerShell, etc.). Los siguientes comandos instalan NumPy y Matplotlib de forma global (sin usar entornos virtuales):

- En muchos sistemas (Windows, macOS, Linux):

  ```
  pip install numpy matplotlib
  ```

- Si el comando anterior no funciona, se puede probar:

  ```
  python -m pip install numpy matplotlib
  ```

o, en algunas instalaciones de Windows:

```
py -m pip install numpy matplotlib
```

Una vez instaladas, para usarlas en un programa de Python las importaremos al comienzo del archivo:

```
import numpy as np
import matplotlib.pyplot as plt
```

Aquí usamos los alias `np` y `plt` por comodidad (son los más habituales en la documentación). A partir de ahora supondremos que en todos los ejemplos del laboratorio se han hecho estas instrucciones de importación.

Las operaciones se pueden realizar directamente en la línea de comandos de Python (el intérprete interactivo, donde aparecen los símbolos `>>>`) o bien escribiendo un programa en un archivo de texto con extensión `.py`, guardándolo y ejecutándolo desde la terminal con `python nombre_archivo.py`. Si se tiene una idea clara de qué cálculos se van a realizar, es conveniente escribir un programa en un archivo `.py`, guardarlo y luego ejecutarlo. Así, si se ha cometido un fallo o se quiere hacer una mínima modificación, no hace falta empezar todo desde el principio.

Cuando se realizan las distintas operaciones, además, se podrán almacenar los resultados en variables: así, si queremos guardar el valor resultante de multiplicar dos números, en lugar de introducir `8*5` sin más, lo podemos guardar bajo `q = 8*5` y, siempre que no modifiquemos este valor de `q`, podremos hacer uso de él mediante una simple llamada, como si fuese una constante más.

Ejemplo. Introducir en Python (en el intérprete interactivo o en un programa) las siguientes instrucciones:

- `q = 8*5`
- `q**2`

Arrays en Python (NumPy)

En NumPy, el tipo de dato fundamental para trabajar con números es el *array* (*ndarray*). Un array es una colección ordenada de números que puede representar:

- un **vector** (array de una dimensión),
- una **matriz** (array de dos dimensiones),
- o estructuras de más dimensiones (que en este curso no necesitaremos).

Creación básica de arrays. Para crear arrays sencillos podemos usar `np.array` a partir de listas de Python, o bien funciones de NumPy que generan valores automáticamente.

```
import numpy as np

# Vector fila con 4 elementos
v = np.array([1, 2, 3, 4])

# Matriz de 2 filas y 3 columnas
A = np.array([[1, 2, 3],
              [4, 5, 6]])

# Vector de 5 ceros
z = np.zeros(5)

# Matriz 3x3 de unos
B = np.ones((3, 3))
```

Propiedades básicas: forma y tamaño. Cada array tiene una *forma* (`shape`) y un número total de elementos:

```
A.shape   # (2, 3): 2 filas, 3 columnas
A.size    # 6 elementos en total
```

Indexación (acceder a elementos). En Python los índices empiezan en **0**. El primer elemento tiene índice `0`, el segundo `1`, etc.

```
v = np.array([10, 20, 30, 40])

v[0]      # 10 (primer elemento)
v[2]      # 30 (tercer elemento)

A = np.array([[1, 2, 3],
              [4, 5, 6]])

A[0, 0]   # 1 (fila 0, columna 0)
A[1, 2]   # 6 (fila 1, columna 2)
```

También podemos acceder a filas o columnas completas:

```
A[0, :]   # primera fila: [1, 2, 3]
A[:, 1]   # segunda columna: [2, 5]
```

Operaciones básicas con arrays. Si dos arrays tienen la misma forma, las operaciones aritméticas se aplican coordenada a coordenada:

```
u = np.array([1, 2, 3])
v = np.array([4, 5, 6])

u + v     # array([5, 7, 9])
u - v     # array([-3, -3, -3])
2*u       # array([2, 4, 6])
u * v     # producto coordenada a coordenada
```

Para productos matriciales (multiplicar "matriz por vector" o "matriz por matriz") se usa el operador `@`:

```
A = np.array([[1, 2],
              [3, 4]])
x = np.array([5, 6])

A @ x     # producto matriz-vector
A @ A     # producto matriz-matriz
```

Estos conceptos serán la base para todo el trabajo posterior con vectores, matrices y sistemas de ecuaciones.

Primeras operaciones

Las operaciones en Python siguen las reglas usuales de la aritmética, y lo único que hay que tener en cuenta es el código específico para ejecutarlas. A continuación daremos cuenta de los operadores más usuales (si `x` e `y` son números reales, o bien arrays de NumPy de la misma forma):

`x + y`	Suma de los números (o arrays) `x` e `y`.
`x - y`	Resta de los números (o arrays) `x` e `y`.
`x * y`	Producto de los números `x` e `y`. Si son arrays NumPy del mismo tamaño, producto coordenada a coordenada.
`x / y`	Cociente de los números `x` e `y`. Si son arrays del mismo tamaño, cociente coordenada a coordenada.
`x ** y`	Elevar `x` a `y`. Para arrays, potencia coordenada a coordenada.
`x @ y`	Producto matricial de arrays bidimensionales (si las dimensiones lo permiten).
`x.T`	Traspuesta del array bidimensional `x`.

Introduzcamos las siguientes operaciones en el intérprete de Python:

- `5 - 3**2*1/4 + 8`
- `(5 - 3)**((2*1)/(4+8))`

Mostrar resultados en pantalla. Para ver el valor de una variable o de una expresión usamos la función `print`:

```
x = 3.5
print(x)                    # muestra 3.5

y = x**2 + 1
print("El valor de y es:", y)

v = np.array([1, 2, 3])
print("Vector v:", v)
```

Cuando se imprime un array de NumPy, se muestran todos sus elementos. Si queremos cambiar cuántos decimales se muestran, podemos usar, por ejemplo, `np.set_printoptions(precision=4)`, que hará que NumPy muestre sus arrays con cuatro decimales.

También tenemos las constantes usuales: `np.pi` (pi), `np.e`, ...

Vectores y Matrices

Un vector es un conjunto de datos dispuestos uno detrás de otro a los que podemos acceder mediante un índice. Una matriz es un conjunto de datos dispuestos en un tablero a los que podemos acceder mediante dos índices, de tal forma que el primer índice indica la fila donde se encuentra el elemento y el segundo indica la columna.

En Python, usando NumPy, definimos estos elementos de la siguiente forma: los datos se dispondrán entre corchetes `[]`, y las filas de una matriz se indican mediante listas anidadas. Así, tenemos los siguientes ejemplos:

```
M = np.array([[1, 2, -3*np.pi, 0],
              [0, np.sqrt(2), -0.01, 4]])
```

representa la matriz $\begin{pmatrix} 1 & 2 & -3\pi & 0 \\ 0 & \sqrt{2} & -0{,}01 & 4 \end{pmatrix}$

```
v = np.array([1, 2, -3, np.pi, 0, np.sqrt(2), -0.01, 4])
```

representa el vector fila $\begin{pmatrix} 1 & 2 & -3 & \pi & 0 & \sqrt{2} & -0{,}01 & 4 \end{pmatrix}$

Ejercicios.

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

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

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

   Por ejemplo:

```
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]])
```

2. Realizar todos los productos matriz–matriz y matriz–vector que las dimensiones admitan (usando el operador `@` para producto matricial).

Sucesiones

Una sucesión es una colección ordenada, finita o no, de números. Desde cierto punto de vista, no son más que casos particulares de vectores. A continuación veremos algunas formas rápidas de definir una sucesión de números, si sabemos qué patrón siguen. Usaremos funciones de NumPy:

- `np.arange(n, m+1)` creará un array cuyo primer elemento es `n` y donde cada coordenada se calcula sumando una unidad a la coordenada anterior, hasta acabar en `m` (es decir, creará la sucesión $n, n+1, n+2, \ldots, m$). Obviamente, `n` tiene que ser menor que `m`.

- `np.arange(n, m+q, q)` creará un array cuyo primer elemento es `n`, y que va a paso `q` (es decir, creará la sucesión $n, n+q, n+2q, \ldots$ hasta sobrepasar `m`).

- `np.linspace(n, m, r)` creará un array cuyo primer elemento es `n`, su último elemento es `m` y que consta de `r` elementos igualmente espaciados.

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`.

Funciones Matemáticas

Python (con NumPy) incluye una serie de funciones matemáticas y trigonométricas que nos ayudan a simplificar algunos cálculos. A continuación mostramos algunas de ellas (supondremos que `import numpy as np` ya se ha ejecutado).

Función	**Descripción**
`np.sqrt(x)`	Raíz cuadrada de `x`.
`np.abs(x)`	Valor absoluto de `x`.
`np.log(x)`	Logaritmo neperiano de `x`.
`np.log2(x)`	Logaritmo en base 2 de `x`.
`np.log10(x)`	Logaritmo en base 10 de `x`.
`np.exp(x)`	Constante e elevada a `x`.
`np.power(2, x)`	Para cada elemento de `x`, calcula 2^x.
`np.remainder(x, y)` o `x % y`	Resto entre la división de `x` e `y`.
`np.round(x)`	Redondeo de `x` al entero más cercano.
`np.ceil(x)`	Redondeo al entero superior de `x`.
`np.floor(x)`	Redondeo al entero inferior de `x`.
`np.trunc(x)`	Redondeo hacia el entero más cercano a cero.
`np.gcd(x, y)`	Máximo común divisor (elemento a elemento).
`np.lcm(x, y)`	(En versiones recientes) m.c.m. elemento a elemento.
`np.sin(x)`, `np.cos(x)`, `np.tan(x)`	Funciones trigonométricas ordinarias (en radianes).
`np.arcsin(x)`, `np.arccos(x)`, `np.arctan(x)`	Funciones trigonométricas inversas.
`np.sign(x)`	Devuelve 1 para elementos positivos, 0 para 0 y -1 para negativos.
`v[n]`	Elemento con índice `n` del vector `v` (primer índice: 0).
`A[n, m]`	Elemento que se encuentra en la fila `n` y columna `m`.
`v[n:m]`	Elementos desde el índice `n` hasta el `m-1`.
`A[n, :]`	Todos los elementos de la fila `n`.
`A[:, m]`	Todos los elementos de la columna `m`.
`A[np.ix_(filas, columnas)]`	Submatriz cuyas filas y columnas se indican mediante los vectores de índices `filas` y `columnas`.
`x.shape[0]`	Número de filas del array bidimensional `x`.
`x.shape[1]`	Número de columnas del array bidimensional `x`.
`len(v)`	Longitud del vector `v`.
`x.shape`	Tupla con (número de filas, número de columnas).
`np.eye(n, m)`	Matriz $n \times m$ con diagonal 1 y resto 0. Si se invoca sólo con `n`, matriz identidad $n \times n$.

`np.diag(x, k)`	Si `x` es un vector, crea una matriz cuya diagonal son los elementos de `x` empezando en la columna `k`. Si `x` es una matriz, devuelve un vector con la diagonal que empieza en el desplazamiento `k`.
`np.zeros((n, m))`	Crea una matriz $n \times m$ con sus elementos iguales a 0.
`np.ones((n, m))`	Crea una matriz $n \times m$ con sus elementos iguales a 1.
`np.linspace(p, q, n)`	Vector de `n` elementos espaciados uniformemente desde `p` hasta `q`.
`np.linalg.matrix_rank(A)`	Rango de la matriz `A`.
`np.linalg.inv(A)`	Inversa de la matriz cuadrada `A`.
`np.linalg.det(A)`	Determinante de la matriz cuadrada `A`.
`np.trace(A)`	Suma de los elementos de la diagonal principal de `A`.
`np.sum(x)`	Suma de los elementos de `x` (o por columnas con `axis=0`).
`np.prod(x)`	Producto de los elementos de `x`.
`np.max(x)`	Máximo de los elementos de `x`.
`np.min(x)`	Mínimo de los elementos de `x`.
`np.sort(x)`	Ordena de menor a mayor los elementos de `x`.

Ejercicios.

1. Resolver el sistema, planteándolo de forma matricial y usando NumPy.

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

   (Pista: escribir el sistema como $A\mathbf{x} = \mathbf{b}$ y usar `np.linalg.solve(A, b)`.)

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

3. 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 el índice 0 de Python). Calcular:
   - el tamaño de esa submatriz (número de filas y columnas),
   - la suma de todos sus elementos.

Funciones definidas por el usuario

En Python podemos crear nuestras propias funciones, ya sea escribiéndolas directamente en el intérprete interactivo o, preferiblemente, en un archivo `.py`. Es recomendable crearlas en un archivo de texto y guardarlo, para poder reutilizar el código.

En Python la **identación** (tabulador o espacios al principio de la línea) es *fundamental*: indica qué instrucciones pertenecen al cuerpo de la función. Todas las líneas del cuerpo deben estar indentadas de la misma forma. La instrucción `return` indica qué valor (o valores) devuelve la función y, además, marca el final de la ejecución de la función.

Las funciones que creamos en Python deben cumplir con el siguiente formato:

```
def nombre_funcion(argumentos_entrada):
    # cuerpo de la función (obligatorio indentar)
    variable_salida = ...
    return variable_salida
```

En caso de devolver varias variables, éstas pueden devolverse como una tupla separada por comas:

```
def nombre_funcion(argumentos_entrada):
    salida1 = ...
    salida2 = ...
    return salida1, salida2
```

Por ejemplo, crearemos una función que calcula el seno(x) en grados:

```
import numpy as np

def sind(x):
    """SIND(x) Calcula seno(x) en grados."""
    s = np.sin(x * np.pi / 180)
    return s
```

Y ejecutamos, por ejemplo en el intérprete:

```
>>> sind(45)
0.7071067811865475
>>> sind(90)
1.0
```

Para que Python ejecute un archivo es suficiente con que éste tenga extensión `.py` y se encuentre en el directorio desde donde lo ejecutamos, por ejemplo:

```
python mi_programa.py
```

Ejercicios.

1. 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`.

2. 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).

3. Crear una función cuyos parámetros sean una matriz y que devuelva dos valores: la suma de todos sus elementos y la suma de los elementos de la diagonal principal (traza).

4. 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.

5. 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.