Introducción a la **Computación Científica** con **Python**
Clase 3: **NumPy**

Diego Passarella

Víctor Viana

EDUCACION PERMANENTE
Universidad de la República

SEDE TACUAREMBÓ
CENUR NORESTE
UNIVERSIDAD DE LA REPÚBLICA URUGUAY

Contenido

- Librería NumPy
- Arrays
- Sistemas lineales y su resolución
- Ejercicios

Librería NumPy

Vamos a utilizar como principal fuente de información, la página del proyecto:

https://numpy.org/

Es la librería estandar universal para trabajar con datos numéricos en Python.

Contiene el objeto **ndarray**, el cual es una estructura de datos en forma de arreglo (array) y posee métodos incorporados en él.

Que es Numpy

- NUMPY es el paquete fundamental para el trabajo de computación científica con PYTHON. Contiene
  - Tipos de datos
  - Funciones
  - Clases
  - Módulos

  que posibilitan la creación y manejo de arrays n-dimensionales.

Librería NumPy

Instalación: https://scipy.org/install.html

Es conveniente instalar todo el paquete de librerías de Python científico.

En Linux/Ubuntu:

```
sudo apt-get install python-numpy python-scipy python-matplotlib python-pandas python-sympy python-nose
```

En Windows:

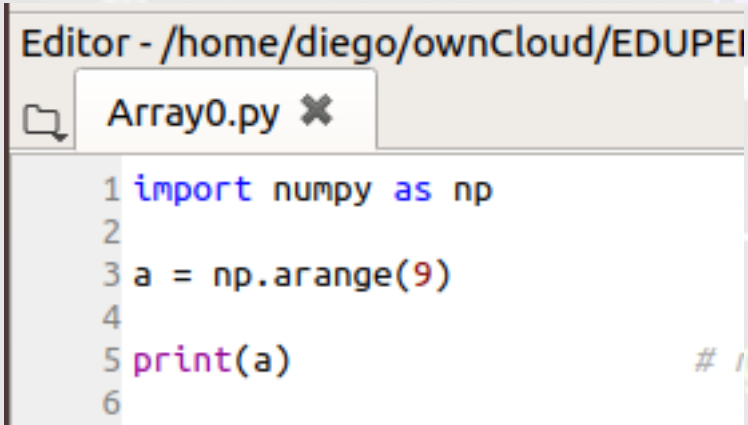
https://www.lfd.uci.edu/~gohlke/pythonlibs/

Librería NumPy

La librería debe ser importada en el encabezado de cada script.

```
import numpy as np
```

Se resume el nombre **numpy** como **np**, para estandarizar los códigos y cualquiera pueda emplearlos (buena práctica)



Librería NumPy

Asegurarse que está instalada la última versión de la librería.

```
import numpy as np
```

```
print(np.__version__)
```

Si no lo está, ejecutar desde una terminal (en Linux/Ubuntu)

```
pip3 install --upgrade numpy
```

Arrays

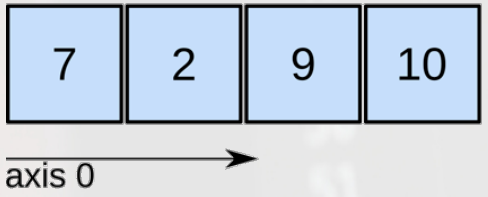
Un **array** (arreglo) es la estructura central de datos de la librería NumPy.

Es una grilla de valores, que contiene información acerca de esos datos, cómo se localiza un elemento y cómo se interpreta. Todos los elementos dentro del array son del mismo tipo.

El rango (**rank**) de un array, es la cantidad de dimensiones que posee. La forma (**shape**) de un array es un conjunto de números enteros que nos indican el tamaño a lo largo de cada dimensión.

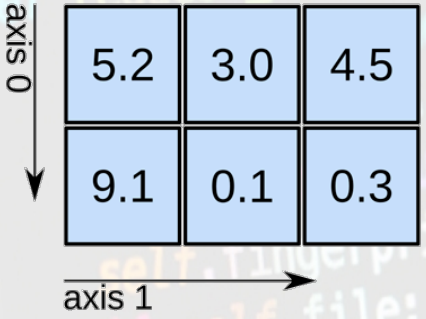
Arrays

1D array



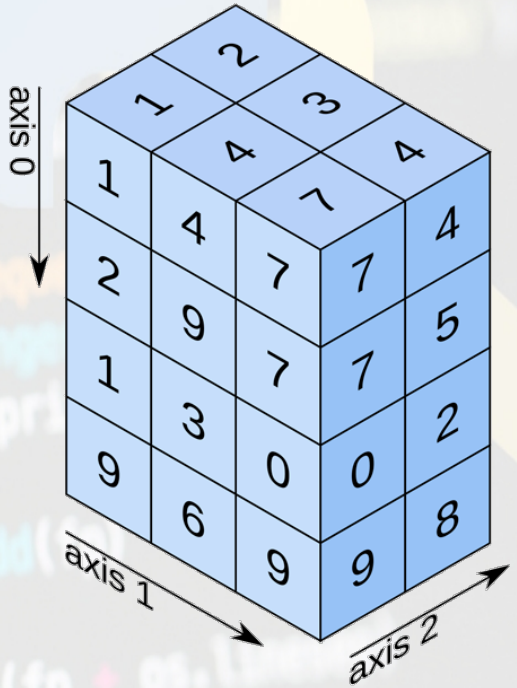
shape: (4,)

2D array



shape: (2, 3)

3D array



shape: (4, 3, 2)

Arrays

Un par de tutoriales en los que nos vamos a basar:

https://numpy.org/doc/stable/user/absolute_beginners.html

https://numpy.org/doc/stable/user/quickstart.html

Arrays

Crear un array:

Por definición de elementos

```
a = np.array([4, 2, 1, 4, 7, 6])
```

Conociendo el array

```
dim_a = a.ndim              # cantidad de "ejes" del array

forma_a = a.shape           # "tamaño" de cada eje

tamano_a = a.size           # cantidad total de elementos

tipo_a = a.dtype            # tipo de elemento del array
```

Arrays

Práctica

Arrays

Distintos comandos para crear un array:

```
np.array()   # crea un array por definición de elementos

np.zeros()   # crea un array lleno de 0's (se define el tamaño
y la forma

np.ones()    # idem anterior, pero lleno de 1's

np.empty()   # idem anterior, pero lleno de cualquier valor
que esté en memoria
```

Arrays

Distintos comandos para crear un array:

```
np.arange([start,] stop[, step][, dtype=])

# crea un array de elementos igualmente espaciados que inician
en "start", terminan en "stop", con un espaciado de "step".
El único argumento obligatorio es "stop".

np.linspace(start,stop,[num = 50])

# crea un array de "num" elementos igualmente espaciados en un
intervalo ("start","stop")
```

Arrays

Práctica

Arrays

Un par de métodos útiles

```
np.sort(array)     # Ordena de forma creciente los elementos del array
```

```
np.concatenate((array1,array2))    # concatena (une) los arrays 1 y 2. Hay que tener cuidado con la forma de los arrays que se pasan como argumento
```

Práctica

Arrays

Arrays con más de un eje (matrices y tensores)

Se puede cambiar la forma de un vector (array de un solo eje)

```
Arr2 = Arr1.reshape((dim_eje0,dim_eje1))
```

El array Arr2 tiene los elementos de Arr1, distribuidos a lo largo de un eje 0 de tamaño dim_eje0 y un eje 1 de tamaño dim_eje1.

Notar que se comienzan a llenar los ejes de mayor índice.

Arrays

Arrays con más de un eje (matrices y tensores)

Se puede cambiar la forma de un vector (array de un solo eje)

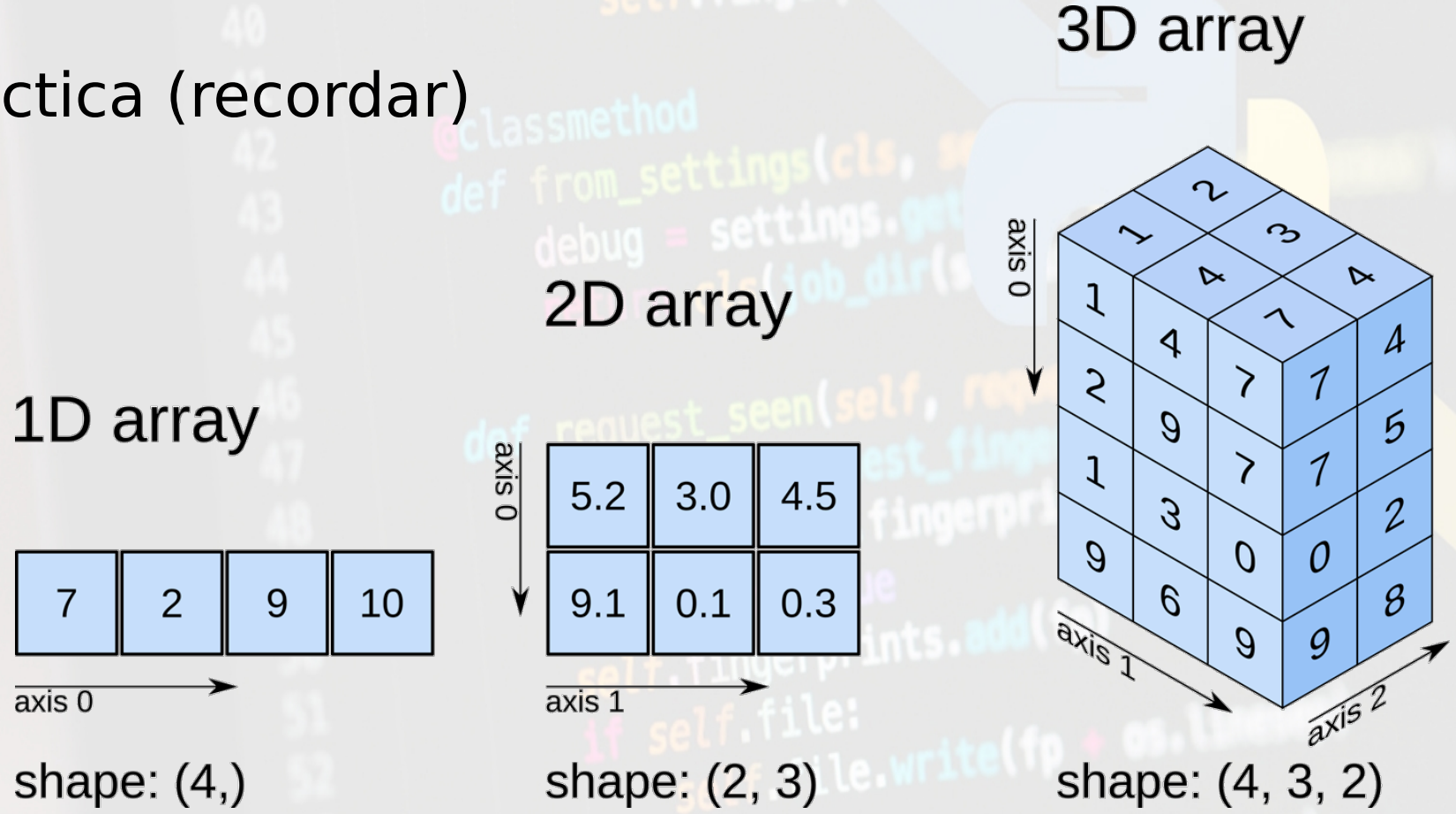
```
Arr2 = Arr1.reshape((dim_eje0,dim_eje1))
```

El array Arr2 tiene los elementos de Arr1, distribuidos a lo largo de un eje 0 de tamaño dim_eje0 y un eje 1 de tamaño dim_eje1.

Notar que se comienzan a llenar los ejes de mayor índice.

Arrays

Práctica (recordar)



Arrays

Y para acelerar un poco el trámite... algunos métodos para explorar por su propia cuenta:

```
a3 = np.vstack((a1,a2))          # "apilado# vertical de los arrays a1 y a2

a4 = np.hstack((a1,a2))          # "apilado" horizontal de arrays

b = np.hsplit(a,[cantidad=x o columnas=(x1,x2)])

# partición de un array en x partes iguales o a partir de sus x1 y x2 columnas.
```

Arrays

Extracción de pedazos de array

```
Arr2 = Arr1[in1:in2]    # Arr2 es un array que contiene los valores de Arr1 entre los índices in1 e in2
```

Para extraer pedazos de arrays de más de un eje, es necesario usar el método `np.hsplit`

Arrays

Operaciones con arrays:

Son elemento a elemento para arrays de iguales dimensiones o para una fila/columna completa

También existen las operaciones "array x escalar", donde las operaciones ocurren elemento a elemento.

Algunos métodos prácticos de tener a mano:

```
max_a = a.max()    min_a = a.min()    suma_a = a.sum()
```

Práctica

Arrays

Una cosa más y nos vamos al intervalo:

Generación de números aleatorios

```
rng = np.random.default_rng(0)
```

```
ArrRnd = rng.random((dim0,dim1,...,dimN))
```

Se pueden generar arrays de distintas dimensiones conteniendo números aleatorios en el intervalo (0,1)

Sistemas Lineales

Podemos definir a los sistemas lineales como un conjunto de ecuaciones (lineales) que comparten incógnitas:

$$4x+3y-5z=2$$
$$-2x+y+4z=3$$
$$-7x+2z=-5$$

El problema radica en encontrar la tupla (x,y,z) que verifique el sistema planteado

Sistemas Lineales

Una forma práctica de presentar los sistemas lineales es a través de su forma matricial

$$\begin{aligned} 4x+3y-5z&=2\\ -2x+y+4z&=3\\ -7x+2z&=-5 \end{aligned} \qquad \begin{pmatrix} 4 & 3 & -5\\ -2 & 1 & 4\\ -7 & 0 & 2 \end{pmatrix}\begin{pmatrix} x\\ y\\ z \end{pmatrix}=\begin{pmatrix} 2\\ 3\\ -5 \end{pmatrix}$$

Las matrices son arreglos bidimensionales, cuyos elementos se organizan en filas (horizontales) y columnas (verticales). $M \in \mathbb{R}^{m\times n}$

Sistemas Lineales

Recordamos algunas operaciones con matrices:

- Suma: M1 = M2 + M3 es una operación elemento a elemento si las matrices poseen la misma dimensión. Es conmutativa.
- Multiplicación: C = AB existe solamente si la cantidad de columnas de A es igual a la cantidad de filas de B. No es conmutativa.

Sistemas Lineales

Multiplicación de matrices (C = AB): $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times r} \Rightarrow C \in \mathbb{R}^{m \times r}$

$$c_{ij} = \sum_{k=1}^{n} a_{in} b_{nj}, 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant r$$

Un vector puede considerarse como una matriz de una sola fila o columna.

El producto de una matriz por un vector, nos devuelve un vector.

Sistemas Lineales

La condición necesaria para que un sistema lineal tenga solución única es que posea tantas ecuaciones como incógnitas (misma cantidad de filas, m, que de columnas, n).

Si hay menos ecuaciones que incóginas (m<n) se dice que el sistema está indeterminado, mientras que si (m>n) se dice que está sobredeterminado y es necesario agregar condiciones extra.

La condición suficiente es que el determinante de la matriz sea no nulo (la matriz posee inversa).

Sistemas Lineales

Práctica

Sistemas Lineales

Unos primeros intentos de solución de un sistema lineal.

No existe la operación "división por una matriz"

$A\,x = b \Rightarrow x = b/A$ ¡NO!

Si conozco la matriz inversa de A, puedo multiplicarla por izquierda en ambos miembros

$A^{-1}\,A\,x = A^{-1}\,b \Rightarrow I\,x = A^{-1}\,b$

```
InvA = np.linalg.inv(A)          # invierto la matriz A

x = np.linalg.matmul(InvA,b)     # la multiplico a izq por b
```

Sistemas Lineales

Un método de resolución un poco más conveniente de estos sistemas, el método linalg.solve basado en la rutina dgesv de LAPACK.

```
x = np.linalg.solve(A, b)
```

Es un método directo para resolver sistemas de ecuaciones lineales basado en la descomposición LU.

Práctica

Sistemas Lineales

Algunos conceptos que conviene conocer y profundizar por su cuenta:

Número de condición de una matriz

```
CondA = np.linalg.cond(A)
```

Debe ser bajo o los resultados pueden variar mucho entre sí ante pequeñas perturbaciones (muy importante para quienes generen resultados sistemas lineales a partir de resultados experimentales)

Sistemas Lineales

Práctica: Resolver los siguientes sistemas

$$A_1 = \begin{pmatrix} 400 & -201 \\ -800 & 401 \end{pmatrix} \quad A_2 = \begin{pmatrix} 401 & -201 \\ -800 & 401 \end{pmatrix} \quad b = \begin{pmatrix} 200 \\ -200 \end{pmatrix}$$

$$A_1 x_1 = b \qquad A_2 x_2 = b$$

Sistemas Lineales

Para cerrar, algunos conceptos que conviene conocer:

- Métodos directos para resolver SEL: basados en la descomposición de la matriz A en el producto de dos o más matrices (A = L*U, por ejemplo), con matrices de descomposición triangulares o diagonales. Es un método "exacto"
- Métodos iterativos: Se descompone la matriz A como suma de dos matrices (A = N-M), con N una matriz fácilmente inversible para generar un punto fijo. Se alcanza la solución con una tolerancia prefijada.

Arrays

```
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a.shape
(2, 3)
>>> a.ndim
2
>>> a.size
6
>>> a.itemsize
8
```

numpy.array(secuencia, tipo_dato)

```
>>> import numpy
>>> numpy.array([1, '5', 4.3, 1+3j])  # ''casting'' implicito
array(['1', '5', '4.3', '(1+3j)'], dtype='|S6')
>>> tupla = (3, 5, 7.7)
>>> a2 = numpy.array(tupla)
>>> a2
array([ 3. ,  5. ,  7.7])
>>> a3 = numpy.array([])
>>> a3
array([], dtype=float64)
>>> a4 = numpy.array(['linea1', 'linea2', 33], dtype='|S3')
array(['lin', 'lin', '33'],
      dtype='|S3')
>>>  # ''casting'' explicito
```

```
numpy.arange([start], stop[, step], dtype=None)
```

Equivalente a la función "`range(start, stop, step)`" de `PYTHON`.

A tener en cuenta...

1. El "`step`" puede ser decimal (novedad!!).
2. El extremo final del intervalo **no** se incluye.

```
>>> numpy.arange(5, 6, 0.1)
array([ 5. ,  5.1,  5.2,  5.3,  5.4,  5.5,
5.6,  5.7,  5.8,  5.9])
```

```
numpy.linspace(start, stop, num=50,
    endpoint=True, retstep=False)
```

Devuelve un array en el que se ha dividido el intervalo [start, stop] (endpoint=True, por defecto) en "num" fragmentos.

```
>>> numpy.linspace(5, 6, 5)
array([ 5.  ,  5.25,  5.5 ,  5.75,  6.  ])
>>> numpy.linspace(5, 6, 5, False, True)
(array([ 5. ,  5.2,  5.4,  5.6,  5.8]), 0.2)
```

Arrays n-dimensionales

```
>>>a = numpy.array([[1, 2, 3, 4],[5, 6, 7, 8], [9, 10, 11, 12]], \
dtype=numpy.int)
>>> a.shape
(3, 4)
>>> a
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Lectura de datos desde un archivo

```
line 1 -> objID,RAJ2000,e_RAJ2000,DEJ2000,e_DEJ2000,upmag,e_upmag,gpmag,e_gpmag,rpmag,e_r
line 2 -> 1237657610717364296,138.692294,0.002,46.253899,0.002,18.049,0.015,16.904,0.033,
...
```

```
numpy.loadtxt(fname, dtype=numpy.float, comments="#",
    delimiter=None, converters=None, skiprows=0,
    usecols=None, unpack=False, ndmin=0)
```

```
>>> array = numpy.loadtxt('datos.csv', \
delimiter=',', skiprows=1)
>>> array.dtype
dtype('float64')
>>> array.size
189
>>> array.shape
(9, 21)
```