**Nociones de Numpy y Matplotlib**

**Contenidos**

* [1  Python](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Python)
* [2  Módulos y paquetes](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#M%C3%B3dulos-y-paquetes)
* [3  NumPy](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#NumPy)
  + [3.1  Operaciones básicas con arrays numpy](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Operaciones-b%C3%A1sicas-con-arrays-numpy)
  + [3.2  Indexado](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Indexado)
  + [3.3  Copias](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Copias)
  + [3.4  Funciones elementales](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Funciones-elementales)
  + [3.5  Definición de funciones](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Definici%C3%B3n-de-funciones)
* [4  Matplotlib](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Matplotlib)
  + [4.1  Plot función real de una variable](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Plot-funci%C3%B3n-real-de-una-variable)
  + [4.2  Plot de una función real de dos variables](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Plot-de-una-funci%C3%B3n-real-de-dos-variables)
* [5  Ejercicios](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Ejercicios)
* [6  Referencias](https://www.unioviedo.es/compnum/laboratorios_py/new/00_Nociones.html#Referencias)

**Python**

En este curso, trabajaremos con Python de tres formas distintas:

* **spyder**: es un entorno interactivo para escribir código Python que facilita la edición y el debugging de una forma sencilla. Lo utilizaremos para escribir el **código de los ejercicios**.
* **consola**: es la interación básica con Python.
* **jupyter notebook**: es un entorno interactivo en el que se combinan ejecución de código, texto enriquecido, fórmulas matemáticas, gráficos, videos, animaciones, etc. Con él han sido realizados estos guiones de prácticas.

Ejecutar el código (asumiendo que los paths son correctos):

* **spyder**: Puede abrirse o bien desde un lanzador de Spyder en el escritorio o desde el menú de Inicio. En cualquier caso, siempre se puede abrir spyder desde la línea de comandos tecleando **spyder**.
* **consola**: Simplemente abrir la consola y ejecutar **python**. Así entraríamos en python en línea de comandos. También se puede usar la consola para ejecutar un programa python tecleando **python programa\_python.py**.
* **jupyter notebook**: abrir la consola e ir a la carpeta donde hemos guardado el código (o queremos guardar el código) y ejecutar **jupyter notebook**. Se abrirá una ventana nueva del navegador por defecto, donde podemos leer los ficheros existentes con sufijo ipynb o crear ficheros nuevos.

Para estas prácticas usaremos [la distribución Anaconda](https://www.anaconda.com/download) con Python 3.

**Módulos y paquetes**

Python es un lenguaje muy versatil que puede realizar tareas muy distintas. En este curso nos centraremos en **algoritmos numéricos** que se pueden implementar con Python.

Python es modular. Un módulo es un programa que contiene, entre otras cosas, las definiciones de las variables y las funciones. A menudo, los módulos se agrupan en paquetes para una determinada aplicación. Por ejemplo, utilizaremos mayormente los paquetes:

* NumPy (numerical python).
* Matplotlib (python plotting).
* SciPy (scientific python).

Un programa típico python empieza importando módulos de los paquetes. Por ejemplo

**import** **numpy** **as** **np**

Aquí hemos importado todos los módulos de numpy y nos referiremos a él con el prefijo np:

print(np.pi)

3.141592653589793

Hay variaciones de la orden import. Por ejemplo, si sólo queremos cargar la definición de 𝜋π usaremos

**from** **numpy** **import** pi **as** PI

print(PI)

3.141592653589793

**NumPy**

El objeto principal de NumPy es el array multidimensional homogéneo. Es una tabla de elementos (habitualmente números), todos del mismo tipo, indexados por una tupla de enteros positivos. En NumPy las dimensiones se llaman ejes. El número de ejes es el rango.

Hay muchas formas de crear un array NumPy. Por ejemplo,

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

b = np.array([(1.5, 2, 3), (4, 5, 6)])

c = np.zeros((3, 4))

d = np.ones((2, 3))

e = np.arange(1, 10, 2)

f = np.arange(1., 10, 2)

g = np.linspace(1, 9, 5)

print('a**\n**',a,'**\n**')

print('b**\n**',b,'**\n**')

print('c**\n**',c,'**\n**')

print('d**\n**',d,'**\n**')

print('e**\n**',e,'**\n**')

print('f**\n**',f,'**\n**')

print('g**\n**',g)

a

[1 2 3 4]

b

[[1.5 2. 3. ]

[4. 5. 6. ]]

c

[[0. 0. 0. 0.]

[0. 0. 0. 0.]

[0. 0. 0. 0.]]

d

[[1. 1. 1.]

[1. 1. 1.]]

e

[1 3 5 7 9]

f

[1. 3. 5. 7. 9.]

g

[1. 3. 5. 7. 9.]

Los índices comienzan en cero. El índice -1 da el último elemento del array.

En arange, los valores se generan en el intervalo [1,10). Es decir, 1 está incluído pero 10 no lo está. El último índice es el paso. Es similar a range pero mientras este genera una lista, arange genera un array numpy.

print('a[0] = ', a[0], '**\n**b[0,0] = ',b[0,0], '**\n**b[0][0] = ',b[0][0])

print('e[-1] = ',e[-1])

a[0] = 1

b[0,0] = 1.5

b[0][0] = 1.5

e[-1] = 9

**len**: número de elementos de un array de dimensión uno (vector) of número de filas en un array de dimensión dos (matriz).

**ndim**: número de dimensiones del array (1 para vector, 2 para matriz).

**shape**: número de elementos en cada dimensión (número de filas y columnas para una matriz).

print('len(a) = ', len(a),'**\n**b.ndim = ', b.ndim, '**\n**b.shape = ', b.shape)

len(a) = 4

b.ndim = 2

b.shape = (2, 3)

**Operaciones básicas con arrays numpy**

Los operadores aritméticos se aplican en los arrays elemento a elemento. ¡Las listas Python funcionan de forma diferente!

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

b = np.array([(1.5, 2, 3, 5)])

a1 = [1, 2, 3, 4]

b1 = [1.5, 2, 3, 5]

print('a+b array numpy')

print(a+b)

print('**\n**a1+b1 lista')

print(a1+b1)

a+b array numpy

[[2.5 4. 6. 9. ]]

a1+b1 lista

[1, 2, 3, 4, 1.5, 2, 3, 5]

Por otra parte

print('a = **\n**', a)

print('**\n**3+a = **\n**', 3+a)

print('**\n**3\*a = **\n**', 3\*a)

print('**\n**3/a = **\n**', 3/a)

print('**\n**a/2 = **\n**', a/2)

a =

[1 2 3 4]

3+a =

[4 5 6 7]

3\*a =

[ 3 6 9 12]

3/a =

[3. 1.5 1. 0.75]

a/2 =

[0.5 1. 1.5 2. ]

A diferencia de otros lenguajes, el operador \* se aplica elemento a elemento en los arrays NumPy.

A = np.array( [[1, 1], [0, 1]] )

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

print('A')

print(A)

print('**\n**B')

print(B)

print('**\n**A\*B')

print(A\*B)

A

[[1 1]

[0 1]]

B

[[2 3]

[1 4]]

A\*B

[[2 3]

[0 4]]

Si queremos multiplicar las matrices utilizamos la función **np.dot**

C = np.dot(A,B)

print('**\n**AB')

print(C)

AB

[[3 7]

[1 4]]

**Indexado**

Es parecido al utilizado en Python. Por ejemplo

a = np.arange(10)

b = np.array([0, 1, 5, -1])

print('a**\n**', a)

print('**\n**a[1:8]**\n**', a[1:8])

print('**\n**a[1:8:2]**\n**', a[1:8:2])

print('**\n**a[-1]**\n**', a[-1])

print('**\n**a[:-1]**\n**', a[:-1])

print('**\n**a[3]**\n**', a[3])

print('**\n**a[3:]**\n**', a[3:])

print('**\n**a[b]**\n**',a[b])

a

[0 1 2 3 4 5 6 7 8 9]

a[1:8]

[1 2 3 4 5 6 7]

a[1:8:2]

[1 3 5 7]

a[-1]

9

a[:-1]

[0 1 2 3 4 5 6 7 8]

a[3]

3

a[3:]

[3 4 5 6 7 8 9]

a[b]

[0 1 5 9]

Análogamente para arrays multidimensionales:

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

print('a = **\n**', a)

print('**\n**a[0,2] = **\n**', a[0,2])

print('**\n**a[:,1] = **\n**', a[:,1])

print('**\n**a[0,:] = **\n**', a[0,:])

print('**\n**a[:1, :2] = **\n**', a[:1, :2])

a =

[[ 1 2 4]

[ 3 -1 2]]

a[0,2] =

4

a[:,1] =

[ 2 -1]

a[0,:] =

[1 2 4]

a[:1, :2] =

[[1 2]]

**Copias**

Cuando operamos y manipulamos arrays, algunas veces, sus datos se copian en un nuevo array, mientras que otras no. Esto crea confusión en los principiantes. Veamos tres ejemplos:

**Sin copia**: Una asignación sencilla no crea copia de los arrays o sus datos.

a = np.arange(12)

b = a

print('a[0] = ', a[0], '**\n**b[0] = ',b[0])

a[0] = 0

b[0] = 0

b[0] = 10

print('a[0] = ', a[0], '**\n**b[0] = ',b[0])

a[0] = 10

b[0] = 10

**Con copia**: el método copy realiza una copia completa del array y de sus datos.

b = a.copy()

print('a[0] = ', a[0], '**\n**b[0] = ',b[0])

a[0] = 10

b[0] = 10

b[0] = 0

print('a[0] = ', a[0], '**\n**b[0] = ',b[0])

a[0] = 10

b[0] = 0

**Funciones elementales**

NumPy contiene las funciones matemáticas elementales con sus nombres habituales. Por ejemplo

print(np.sin(PI/2))

print(np.exp(-1))

print(np.arctan(np.inf))

print(np.sqrt(4))

1.0

0.36787944117144233

1.5707963267948966

2.0

Si se aplican a un array numpy, el resultado es un array numpy.

a = np.linspace(2,4,5)

print('a =**\n**', a)

print('**\n**np.sqrt(a) =**\n**', np.sqrt(a))

a =

[2. 2.5 3. 3.5 4. ]

np.sqrt(a) =

[1.41421356 1.58113883 1.73205081 1.87082869 2. ]

**Definición de funciones**

Podemos definir funciones de dos formas:

* Usando una función **lambda**.
* Usando **def**.

f1 = **lambda** x: x\*\*3

f2 = **lambda** x,y: x+y

print('f1(2) = ', f1(2))

print('f2(1,1) = ', f2(1,1))

f1(2) = 8

f2(1,1) = 2

**def** f3(x):

**if** x > 2:

**return** 0

**else**:

**return** 1

print('f3(-1) = ', f3(-1))

print('f3(3) = ',f3(3))

f3(-1) = 1

f3(3) = 0

Pero si definimos la función de esta manera no funciona con arrays numpy:

a = np.arange(4)

print(a)

print('f1(a) = ', f1(a))

print(f3(a))

[0 1 2 3]

f1(a) = [ 0 1 8 27]

---------------------------------------------------------------------------

ValueError Traceback (most recent call last)

<ipython-input-32-b4897f62ca61> in <module>

**2** print(a)

**3** print('f1(a) = ', f1(a))

----> 4 print(f3(a))

<ipython-input-30-87c0f4b82b1f> in f3(x)

**1** def f3(x):

----> 2 if x > 2:

**3** return 0

**4** else:

**5** return 1

ValueError: The truth value of an array with more than one element is ambiguous. Use a.any() or a.all()

Para más información, ver [Numpy Quickstart tutorial](https://docs.scipy.org/doc/numpy/user/quickstart.html)

**Matplotlib**

Usamos el paquete Matplotlib para hacer plots. La segunda línea del código siguiente se usa sólo en Notebooks pero no en línea de comando o en Spyder.

**import** **matplotlib.pyplot** **as** **plt**

%**matplotlib** inline

**Plot función real de una variable**

x = np.linspace(-1,2) *# malla*

f = **lambda** x : x\*\*3 - 2\*x\*\*2 + 1 *# función*

OX = 0\*x *# eje OX*

plt.figure()

plt.plot(x,f(x)) *# dibujar la función*

plt.plot(x,OX,'k-') *# dibujar el eje X*

plt.xlabel('x')

plt.ylabel('y')

plt.title('función')

plt.show()

Imagen que contiene Gráfico de líneas

Descripción generada automáticamente

**Plot de una función real de dos variables**

Creamos una malla para la coordenada 𝑥x y otra para la coordenada 𝑦y. Es decir, por ejemplo, 20 puntos igualmente espaciados entre −1−1 y 11 para cada coordenada.

xgrid = np.linspace(-1,1,20)

ygrid = np.linspace(-1,1,20)

Creamos ahora dos matrices que contienen la malla bidimensional

X, Y = np.meshgrid(xgrid,ygrid)

Definimos la función de dos variables

g = **lambda** x,y: x\*\*2 + y\*\*2

Y representamos la función usando diferentes colores para cada intervalo de valores de la función

plt.figure()

plt.contourf(X,Y,g(X,Y), cmap='jet')

plt.colorbar()

plt.show()

Imagen que contiene Gráfico

Descripción generada automáticamente

También podemos representarla como una superficie

**from** **mpl\_toolkits.mplot3d** **import** Axes3D

fig1 = plt.figure(figsize=(10,5))

ax1 = fig1.gca(projection='3d')

ax1.plot\_surface(X, Y, g(X,Y), cmap='jet')

plt.show()

Gráfico, Gráfico de superficie

Descripción generada automáticamente

O como una malla

fig2 = plt.figure(figsize=(10,5))

ax2 = fig2.gca(projection='3d')

ax2.plot\_wireframe(X, Y, g(X,Y))

plt.show()

Gráfico, Gráfico de superficie

Descripción generada automáticamente

**Ejercicios**

**Ejercicio 1**

Crea e imprime los siguientes vectores y matrices numpy:

𝑎=(1,3,7)𝑏=(204136)a=(1,3,7)b=(243016)

𝑐=⎛⎝⎜⎜000000⎞⎠⎟⎟𝑑=⎛⎝⎜⎜111111111111⎞⎠⎟⎟c=(000000)d=(111111111111)

%**run** Ejercicio1

a =

[1 3 7]

b =

[[2 4 3]

[0 1 6]]

c =

[[0. 0.]

[0. 0.]

[0. 0.]]

d =

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

**Ejercicio 2**

Utilizando primero la orden arange y luego linspace crea los vectores:

* 𝑎=(7,9,11,13,15)a=(7,9,11,13,15)
* 𝑏=(10,9,8,7,6)b=(10,9,8,7,6)
* 𝑐=(15,10,5,0)c=(15,10,5,0)

%**run** Ejercicio2

a = [ 7 9 11 13 15]

b = [10 9 8 7 6]

c = [15 10 5 0]

a = [ 7. 9. 11. 13. 15.]

b = [10. 9. 8. 7. 6.]

c = [15. 10. 5. 0.]

**Ejercicio 3**

Se considera la matriz

𝐴=⎛⎝⎜⎜⎜⎜296−518−1−735−2−947−8−6⎞⎠⎟⎟⎟⎟A=(213498576−1−2−8−5−7−9−6)

Obtén a partir de 𝐴A las submatrices:

𝑎=⎛⎝⎜⎜⎜⎜296−5⎞⎠⎟⎟⎟⎟𝑏=(6−1−2−8)𝑐=(2918)a=(296−5)b=(6−1−2−8)c=(2198)

𝑑=(−2−9−8−6)𝑒=(8−15−2)𝑓=⎛⎝⎜⎜⎜⎜18−1−735−2−947−8−6⎞⎠⎟⎟⎟⎟𝑔=⎛⎝⎜⎜8−1−75−2−9⎞⎠⎟⎟d=(−2−8−9−6)e=(85−1−2)f=(134857−1−2−8−7−9−6)g=(85−1−2−7−9)

%**run** Ejercicio3

A =

[[ 2 1 3 4]

[ 9 8 5 7]

[ 6 -1 -2 -8]

[-5 -7 -9 -6]]

a =

[ 2 9 6 -5]

b =

[ 6 -1 -2 -8]

c =

[[2 1]

[9 8]]

d =

[[-2 -8]

[-9 -6]]

e =

[[ 8 5]

[-1 -2]]

f =

[[ 1 3 4]

[ 8 5 7]

[-1 -2 -8]

[-7 -9 -6]]

g =

[[ 8 5]

[-1 -2]

[-7 -9]]

**Ejercicio 4**

Escribe las funciones y calcula sus valores en los puntos que se indican:

* 𝑓(𝑥)=𝑥𝑒𝑥,𝑓(2)f(x)=xex,f(2).
* 𝑔(𝑧)=𝑧sen𝑧cos𝑧,𝑔(𝜋/4)g(z)=zsenzcos⁡z,g(π/4).
* ℎ(𝑥,𝑦)=𝑥𝑦𝑥2+𝑦2,ℎ(2,4)h(x,y)=xyx2+y2,h(2,4).

%**run** Ejercicio4

f(2) = 14.7781121978613

g(pi/4) = 1.5707963267948966

h(2,4) = 0.4

**Ejercicio 5**

Dibuja en el intervalo [−2𝜋,2𝜋][−2π,2π] la función 𝑓(𝑥)=𝑥sen(3𝑥)f(x)=xsen(3x)

%**run** Ejercicio5

Gráfico, Gráfico de líneas

Descripción generada automáticamente

**Referencias**

* [Style Guide for Python Code](https://www.python.org/dev/peps/pep-0008)
* [NumPy Quickstart tutorial](https://docs.scipy.org/doc/numpy/user/quickstart.html)
* [Download Anaconda Distribution](https://www.anaconda.com/download)