Nociones de Numpy y Matplotlib

Contenidos

- 1 Python
- 2 Módulos y paquetes
- 3 NumPy
  - 3.1 Operaciones básicas con arrays numpy
  - 3.2 Indexado
  - 3.3 Copias
  - 3.4 Funciones elementales
  - 3.5 Definición de funciones
- 4 Matplotlib
  - 4.1 Plot función real de una variable
  - 4.2 Plot de una función real de dos variables
- 5 Ejercicios
- 6 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 π 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], '\nb[0,0]  = ',b[0,0], '\nb[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),'\nb.ndim  =  ', b.ndim, '\nb.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('\na1+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('\n3+a = \n', 3+a)
print('\n3*a = \n', 3*a)
print('\n3/a = \n', 3/a)
print('\na/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('\nB')
print(B)
print('\nA*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('\nAB')
print(C)
```

```
AB
[[3 7]
 [1 4]]
```

Indexado

Es parecido al utilizado en Python. Por ejemplo

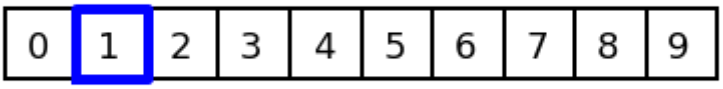
```
a = np.arange(10)
```

```
print('a\n', a)
print('\na[1]\n', a[1])
print('\na[1:8]\n', a[1:8])
print('\na[1:8:2]\n', a[1:8:2])
print('\na[1:]\n', a[1:])
print('\na[:8]\n', a[:8])
print('\na[::2]\n', a[::2])
print('\na[-1]\n', a[-1])
print('\na[:-1]\n', a[:-1])
print('\na[0]\n', a[0])
print('\na[1:]\n', a[1:])
print('\na[::-1]\n', a[::-1])
print('\na[::-2]\n', a[::-2])
print('\na[7:1:-2]\n', a[7:1:-2])
```

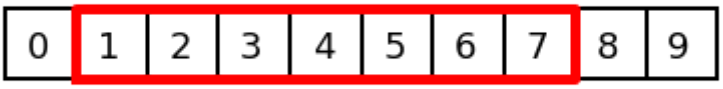
```
a
 [0 1 2 3 4 5 6 7 8 9]

a[1]
 1

a[1:8]
 [1 2 3 4 5 6 7]

a[1:8:2]
 [1 3 5 7]

a[1:]
 [1 2 3 4 5 6 7 8 9]

a[:8]
 [0 1 2 3 4 5 6 7]

a[::2]
 [0 2 4 6 8]

a[-1]
 9

a[:-1]
 [0 1 2 3 4 5 6 7 8]

a[0]
 0

a[1:]
 [1 2 3 4 5 6 7 8 9]

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

a[::-2]
 [9 7 5 3 1]

a[7:1:-2]
 [7 5 3]
```

```
%run slicing1
```

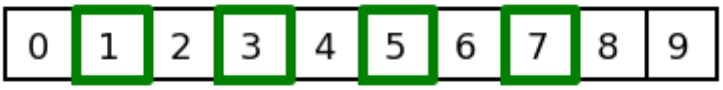
a[1]



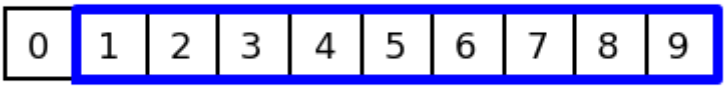
a[1:8]



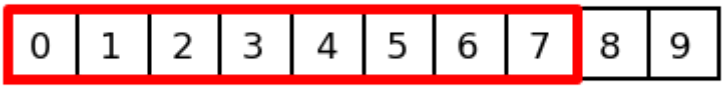
a[1:8:2]



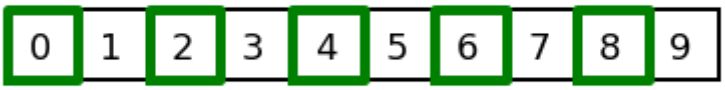
a[1:]



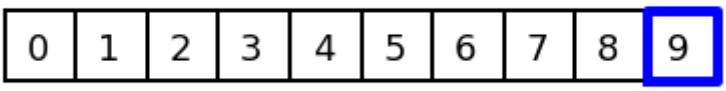
a[:8]

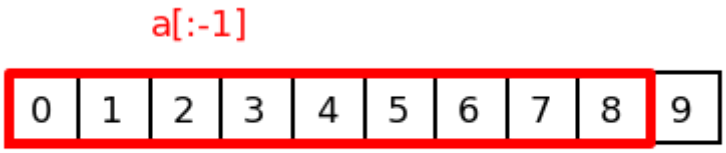


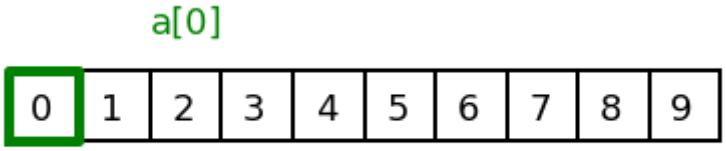
a[::2]

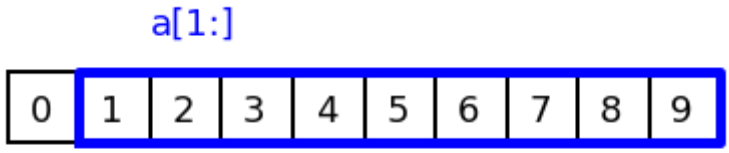


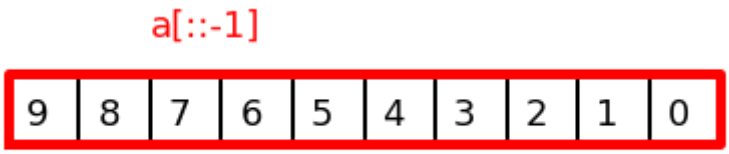
a[-1]

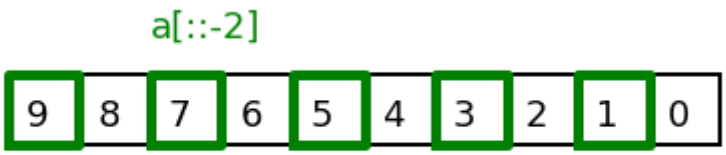


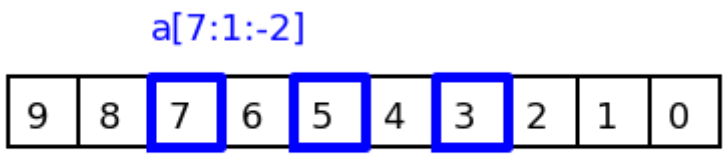













```
b = np.array([0, 1, 5, -1])
print('\na[b]\n',a[b])
```

```
a[b]
 [0 1 5 9]
```

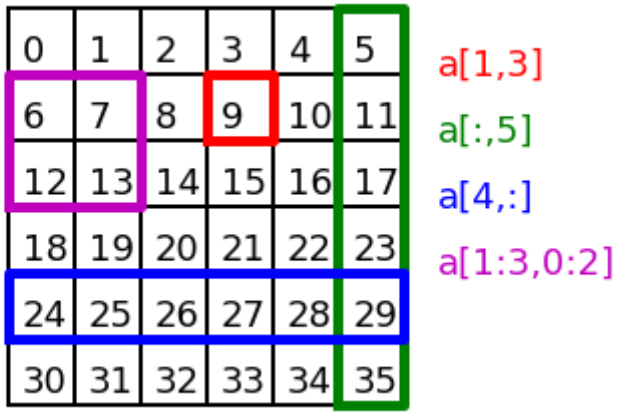
Análogamente para arrays multidimensionales:

```
n = 6
s = n*n
a = np.arange(s)
a = np.reshape(a,(n,n))
```

```
print('a = \n', a)
print('\033[91m \na[1,3] = \n', a[1,3])
print('\033[92m \na[:,5] = \n', a[:,5])
print('\033[94m \na[4,:] = \n', a[4,:])
print('\033[95m \na[1:3, 0:2] = \n', a[1:3, 0:2])
```

```
a =
 [[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]

a[1,3] =
 9

a[:,5] =
 [ 5 11 17 23 29 35]

a[4,:] =
 [24 25 26 27 28 29]

a[1:3, 0:2] =
 [[ 6  7]
 [12 13]]
```

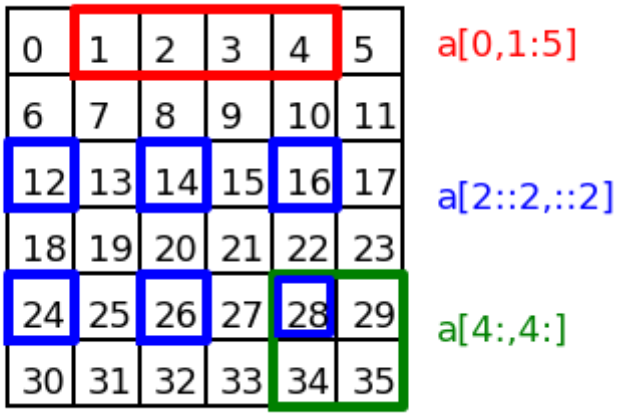
```
%run slicing2
```



```
print('\033[91m \na[0,1:5] = \n', a[0,1:5])
print('\033[92m \na[4:,4:] = \n', a[4:,4:])
print('\033[94m \na[2::2,::2] = \n', a[2::2,::2])
```

```
a[0,1:5] =
 [1 2 3 4]

a[4:,4:] =
 [[28 29]
 [34 35]]

a[2::2,::2] =
 [[12 14 16]
 [24 26 28]]
```

```
%run slicing3
```



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], '\nb[0] = ',b[0])
```

```
a[0] =  0
b[0] =  0
```

```
b[0] = 10
```

```
print('a[0] = ', a[0], '\nb[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], '\nb[0] = ',b[0])
```

```
a[0] =  10
b[0] =  10
```

```
b[0] = 0
```

```
print('a[0] = ', a[0], '\nb[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('\nnp.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
```

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

Matplotlib

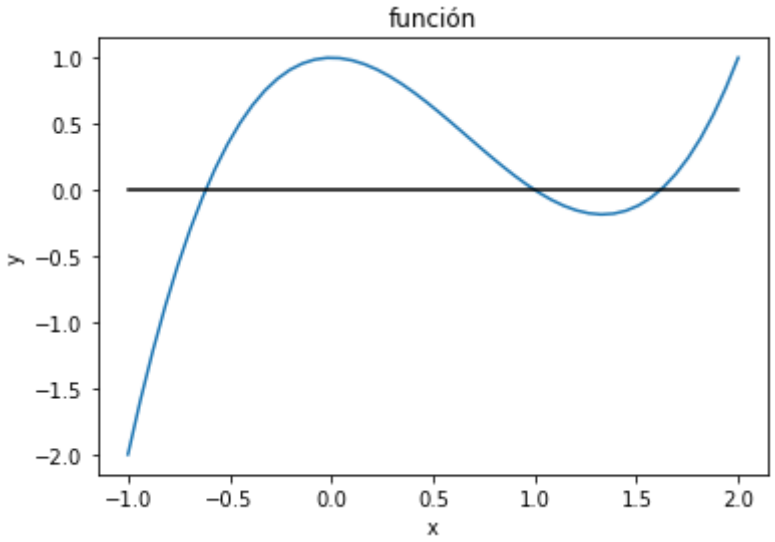
Usamos el paquete Matplotlib para hacer plots.

```
import matplotlib.pyplot as plt
```

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()
```



Plot de una función real de dos variables

Creamos una malla para la coordenada x y otra para la coordenada y. Es decir, por ejemplo, 20 puntos igualmente espaciados entre -1 y 1 para cada coordenada.

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

Creamos ahora dos matrices que contienen la malla bidimensional

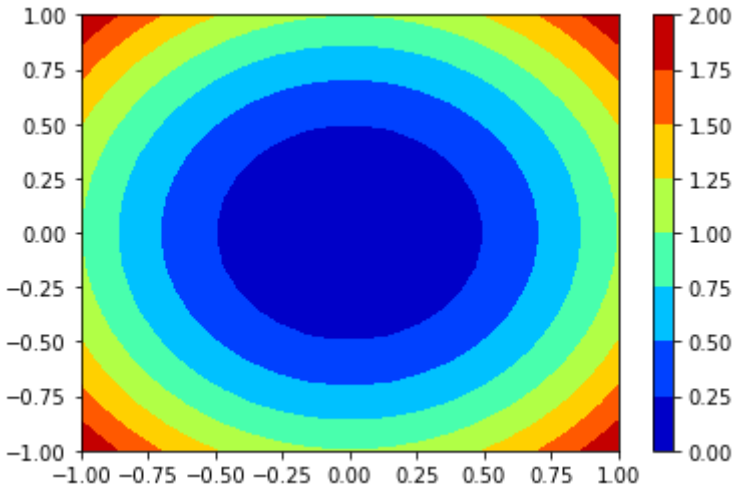
```
X, Y = np.meshgrid(xgrid,ygrid)
```

Definimos la función de dos variables

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

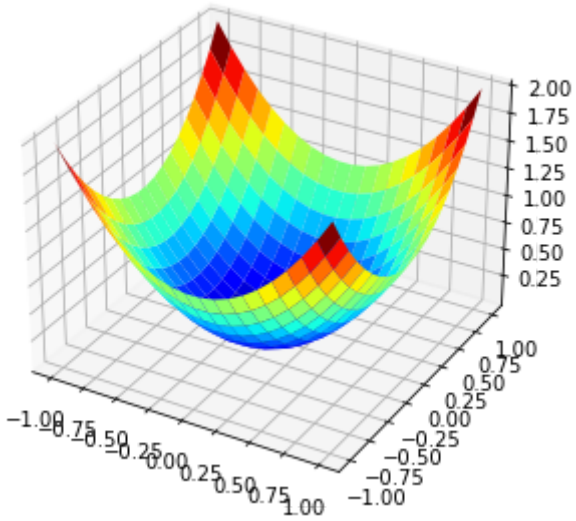
Y representamos la función usando diferentes colores para cada intervalo de valores de la función

```
plt.figure()
plt.contourf(X,Y,g(X,Y), cmap='jet')
plt.colorbar()
plt.show()
```



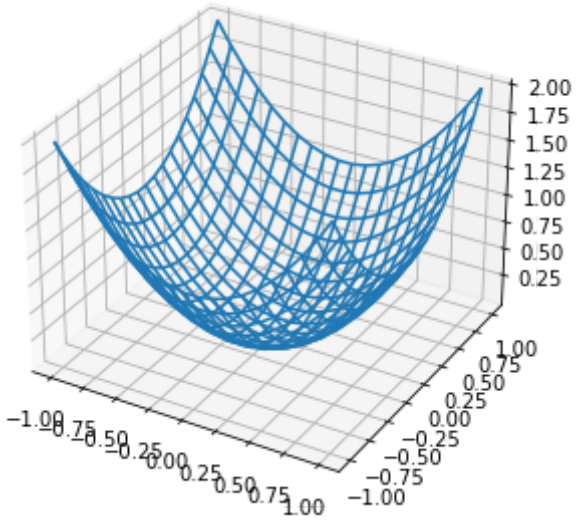
También podemos representarla como una superficie

```
from mpl_toolkits.mplot3d import Axes3D

fig1  = plt.figure(figsize=(10,5))
ax1 = plt.axes(projection ='3d')
ax1.plot_surface(X, Y, g(X,Y), cmap='jet')
plt.show()
```



O como una malla

```
fig2  = plt.figure(figsize=(10,5))
ax2 = plt.axes(projection ='3d')
ax2.plot_wireframe(X, Y, g(X,Y))
plt.show()
```



Ejercicios

Ejercicio 1

Crea e imprime los siguientes vectores y matrices numpy:

$$a = (1,\ 3,\ 7) \qquad b = \begin{pmatrix} 2 & 4 & 3 \\ 0 & 1 & 6 \end{pmatrix} \qquad c = (1,\ 1,\ 1) \qquad d = (0,\ 0,\ 0,\ 0)$$

$$e = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \qquad f = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

```
%run Ejercicio1
```

```
a =
[1 3 7]

b =
[[2 4 3]
 [0 1 6]]

c =
[1. 1. 1.]

d =
[0. 0. 0. 0.]

e =
[[0. 0.]
 [0. 0.]
 [0. 0.]]

f =
[[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:

- $a = (7,\ 9,\ 11,\ 13,\ 15)$
- $b = (10,\ 9,\ 8,\ 7,\ 6)$
- $c = (15,\ 10,\ 5,\ 0)$
- d es un vector que empieza en 0 y acaba en 1 y contiene 11 puntos equiespaciados.
- e es un vector que empieza en -1 y acaba en 1 y sus puntos divididen $[-1, 1]$ en 10 intervalos iguales.
- f es un vector que empieza en 1 y acaba en 2 y sus puntos están separados por un paso 0.1

Usar

```
np.set_printoptions(precision=2,suppress=True)
```

para que el formato no sea exponencial y use dos dígitos.

```
%run Ejercicio2
```

```
a =  [ 7  9 11 13 15]
b =  [10  9  8  7  6]
c =  [15 10  5  0]
d =  [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
e =  [-1.  -0.8 -0.6 -0.4 -0.2 -0.   0.2  0.4  0.6  0.8  1. ]
f =  [1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]


a =  [ 7.  9. 11. 13. 15.]
b =  [10.  9.  8.  7.  6.]
c =  [15. 10.  5.  0.]
d =  [0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
e =  [-1.  -0.8 -0.6 -0.4 -0.2  0.   0.2  0.4  0.6  0.8  1. ]
f =  [1.  1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2. ]
```

Ejercicio 3

- Utilizando `arange` , crea el vector

$$v = (0.0,\ 1.1,\ 2.2,\ 3.3,\ 4.4,\ 5.5,\ 6.6,\ 7.7,\ 8.8,\ 9.9,\ 11.0,\ 12.1)$$

- Utilizando *slicing* obtén, a partir de v, el vector v_i

$$v_i = (12.1,\ 11.0,\ 9.9,\ 8.8,\ 7.7,\ 6.6,\ 5.5,\ 4.4,\ 3.3,\ 2.2,\ 1.1,\ 0.0)$$

- Utilizando *slicing* obtén, a partir de v, los vectores v_1 y v_2

$$\begin{aligned} v_1 &= (0.0,\ 2.2,\ 4.4,\ 6.6,\ 8.8,\ 11.) \\ v_2 &= (1.1,\ 3.3,\ 5.5,\ 7.7,\ 9.9,\ 12.1) \end{aligned}$$

- Utilizando *slicing* obtén, a partir de v, los vectores v_1, v_2 y v_3

$$\begin{aligned} v_1 &= (0.0,\ 3.3,\ 6.6,\ 9.9) \\ v_2 &= (1.1,\ 4.4,\ 7.7,\ 11.0) \\ v_3 &= (2.2,\ 5.5,\ 8.8,\ 12.1) \end{aligned}$$

- Utilizando *slicing* obtén, a partir de v, los vectores v_1, v_2, v_3 y v_4.

$$\begin{aligned} v_1 &= (0.0,\ 4.4,\ 8.8) \\ v_2 &= (1.1,\ 5.5,\ 9.9) \\ v_3 &= (2.2,\ 6.6,\ 11.0) \\ v_4 &= (3.3,\ 7.7,\ 12.1) \end{aligned}$$

```
%run Ejercicio3
```

```
v =  [ 0.   1.1  2.2  3.3  4.4  5.5  6.6  7.7  8.8  9.9 11.  12.1]

vi =  [12.1 11.   9.9  8.8  7.7  6.6  5.5  4.4  3.3  2.2  1.1  0. ]

v1 =  [ 0.   2.2  4.4  6.6  8.8 11. ]
v2 =  [ 1.1  3.3  5.5  7.7  9.9 12.1]

v1 =  [0.  3.3 6.6 9.9]
v2 =  [ 1.1  4.4  7.7 11. ]
v3 =  [ 2.2  5.5  8.8 12.1]

v1 =  [0.  4.4 8.8]
v2 =  [1.1 5.5 9.9]
v3 =  [ 2.2  6.6 11. ]
v4 =  [ 3.3  7.7 12.1]
```

Ejercicio 4

A partir de un vector $a = (1, 2, 3)$ crear un vector $b = (0, 1, 2, 3, 0)$

1. Usando np.append (https://numpy.org/doc/stable/reference/generated/numpy.append.html) dos veces: añadir un cero al final y luego gira el vector añadir el otro cero y volver a girar el vector.
2. Creando un vector $b = (0, 0, 0, 0, 0)$ e insertando el vector a con slicing.
3. Usando np.concatenate (https://numpy.org/doc/stable/reference/generated/numpy.concatenate.html) creando un vector $c = (0)$ previamente.

```
%run Ejercicio4
```

```
1.
b =  [0. 1. 2. 3. 0.]

2.
b =  [0. 1. 2. 3. 0.]

3.
b =  [0. 1. 2. 3. 0.]
```

Ejercicio 5

Se considera la matriz

$$A = \begin{pmatrix} 2 & 1 & 3 & 4 \\ 9 & 8 & 5 & 7 \\ 6 & -1 & -2 & -8 \\ -5 & -7 & -9 & -6 \end{pmatrix}$$

Obtén a partir de A las submatrices:

$$a = \begin{pmatrix} 2 \\ 9 \\ 6 \\ -5 \end{pmatrix} \qquad b = \begin{pmatrix} 6 & -1 & -2 & -8 \end{pmatrix} \qquad c = \begin{pmatrix} 2 & 1 \\ 9 & 8 \end{pmatrix}$$

$$d = \begin{pmatrix} -2 & -8 \\ -9 & -6 \end{pmatrix} \qquad e = \begin{pmatrix} 8 & 5 \\ -1 & -2 \end{pmatrix} \qquad f = \begin{pmatrix} 1 & 3 & 4 \\ 8 & 5 & 7 \\ -1 & -2 & -8 \\ -7 & -9 & -6 \end{pmatrix} \qquad g = \begin{pmatrix} 8 & 5 \\ -1 & -2 \\ -7 & -9 \end{pmatrix}$$

```
%run Ejercicio5
```

```
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 6

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

$f(x) = x\,e^x$ en $x = 2$ $\qquad$ $g(z) = \dfrac{z}{\text{sen}z\cos z}$ en $z = \pi/4$ $\qquad$ $h(x,y) = \dfrac{xy}{x^2+y^2}$

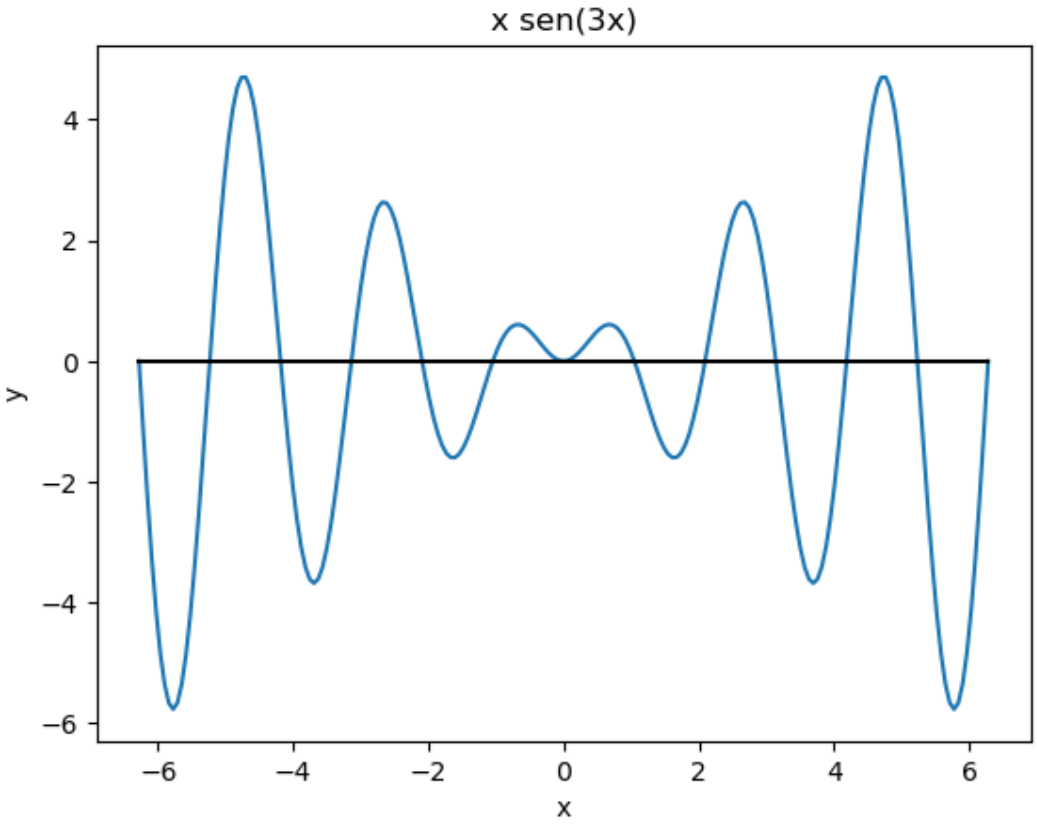
```
%run Ejercicio6
```

```
f(2) =  14.7781121978613

g(pi/4) =  1.5707963267948966

h(2,4) =  0.4
```

Ejercicio 7

Dibuja en el intervalo $[-2\pi, 2\pi]$ la función $f(x) = x\,\text{sen}(3x)$

```
%run Ejercicio7
```



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)