Librería numpy

Numpy es un paquete adecuado para cálculos numéricos eficientes y que permite manejar estructuras de datos tales como arrays multidimensionales homogéneos. Ofrece herramientas para integrar código Python con C++ o Fortran y para manejar números aleatorios, álgebra lineal, transformadas de Fourier, etc.

Conceptos básicos, creación de arrays, imprimirlos

Tipo de datos, Homogéneos, dimensiones, tamaño.

In [1]:
```
# Creación de arrays a partir de listas:

import numpy as np

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

print("a = ", a)

print(a.ndim, a.shape, a.dtype)    # dimensión, forma y tipo de los elem
```

```
a =  [[1. 2. 3.]
 [4. 5. 6.]]
2 (2, 3) float64
```

In [2]:
```
# Creación de arrays a partir de listas. Ahora forzamos el tipo de los e

b = np.array([[1, 2], [3, 4]], dtype=complex)
print(b, b.dtype)
```

```
[[1.+0.j 2.+0.j]
 [3.+0.j 4.+0.j]] complex128
```

Arrays grandes, definidos por rangos

In [3]:

```
a = np.arange(12)
print("a = ", a)
b = a.reshape(3, 4)
print("b = ", b)
print(type(a), type(b))

print("núm. de dimensiones (ejes) de a: ", a.ndim)
print("núm. de dimensiones (ejes) de b: ", b.ndim)

print("dimensiones de a: ", a.shape)
print("dimensiones de b: ", b.shape)

print("Tipo de los elementos de a: ", a.dtype.name)
print("Tipo de los elementos de b: ", b.dtype.name)

print("Número total de elementos de a: ", a.size)   # igual al producto
print("Número total de elementos de b: ", b.size)

print(np.arange(10000))                             # Cuando es muy larg
```

```
a =  [ 0  1  2  3  4  5  6  7  8  9 10 11]
b =  [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
<class 'numpy.ndarray'> <class 'numpy.ndarray'>
núm. de dimensiones (ejes) de a:  1
núm. de dimensiones (ejes) de b:  2
dimensiones de a:  (12,)
dimensiones de b:  (3, 4)
Tipo de los elementos de a:  int32
Tipo de los elementos de b:  int32
Número total de elementos de a:  12
Número total de elementos de b:  12
[   0    1    2 ... 9997 9998 9999]
```

In [4]:
```
# Todo ceros:
b = np.zeros((3, 4))
print("b = ", b)

# Todo unos:
c = np.ones((2, 3, 4), dtype=np.int16)   # El tipo también puede especif
print("c = ", c)

# Array vacío:
d = np.empty( (2,3) )                    # Sin valores iniciales, el res
print("d = ", d)

# Con elementos elegidos aleatoriamente:
e = np.random.random((2,3))
print("e = ", e)
```

```
b =  [[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
c =  [[[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]

 [[1 1 1 1]
  [1 1 1 1]
  [1 1 1 1]]]
d =  [[1. 2. 3.]
 [4. 5. 6.]]
e =  [[0.23283508 0.12321309 0.53060682]
 [0.52823324 0.69007926 0.68322812]]
```

In [5]:
```
# Secuencias de valores:
f = np.arange( 10, 50, 5)
print("f = ", f)

# Secuencias, interpolando linealmente
g = np.linspace( 0, 2, 25)
print("g = ", g)

from numpy import pi
h = np.linspace(0, 2*pi, 50)
print(h)
```

```
f =  [10 15 20 25 30 35 40 45]
g =  [0.         0.08333333 0.16666667 0.25       0.33333333 0.41666667
 0.5        0.58333333 0.66666667 0.75       0.83333333 0.91666667
 1.         1.08333333 1.16666667 1.25       1.33333333 1.41666667
 1.5        1.58333333 1.66666667 1.75       1.83333333 1.91666667
 2.        ]
[0.         0.12822827 0.25645654 0.38468481 0.51291309 0.64114136
 0.76936963 0.8975979  1.02582617 1.15405444 1.28228272 1.41051099
 1.53873926 1.66696753 1.7951958  1.92342407 2.05165235 2.17988062
 2.30810889 2.43633716 2.56456543 2.6927937  2.82102197 2.94925025
 3.07747852 3.20570679 3.33393506 3.46216333 3.5903916  3.71861988
 3.84684815 3.97507642 4.10330469 4.23153296 4.35976123 4.48798951
 4.61621778 4.74444605 4.87267432 5.00090259 5.12913086 5.25735913
 5.38558741 5.51381568 5.64204395 5.77027222 5.89850049 6.02672876
 6.15495704 6.28318531]
```

In [6]:
```
np.sin(h)    # se aplica la función seno a todos los puntos
```

Out[6]:
```
array([ 0.00000000e+00,  1.27877162e-01,  2.53654584e-01,  3.75267005e-
01,
        4.90717552e-01,  5.98110530e-01,  6.95682551e-01,  7.81831482e-
01,
        8.55142763e-01,  9.14412623e-01,  9.58667853e-01,  9.87181783e-
01,
        9.99486216e-01,  9.95379113e-01,  9.74927912e-01,  9.38468422e-
01,
        8.86599306e-01,  8.20172255e-01,  7.40277997e-01,  6.48228395e-
01,
        5.45534901e-01,  4.33883739e-01,  3.15108218e-01,  1.91158629e-
01,
        6.40702200e-02, -6.40702200e-02, -1.91158629e-01, -3.15108218e-
01,
       -4.33883739e-01, -5.45534901e-01, -6.48228395e-01, -7.40277997e-
01,
       -8.20172255e-01, -8.86599306e-01, -9.38468422e-01, -9.74927912e-
01,
       -9.95379113e-01, -9.99486216e-01, -9.87181783e-01, -9.58667853e-
01,
       -9.14412623e-01, -8.55142763e-01, -7.81831482e-01, -6.95682551e-
01,
       -5.98110530e-01, -4.90717552e-01, -3.75267005e-01, -2.53654584e-
01,
       -1.27877162e-01, -2.44929360e-16])
```

Operaciones aritméticas básicas

Las operaciones aritméticas actúan con arrays elemento a elemento:

In [7]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print(a+1)
print(a**2)
print(np.sin(a))
print(a<4)

b = np.array([[2, 2, 2], [3, 3, 3]])
print(a+b)
print(a-b)
print(a*b) # OJO: elemento a elemento
```

```
[[1 2 3]
 [4 5 6]]
[[2 3 4]
 [5 6 7]]
[[ 1  4  9]
 [16 25 36]]
[[ 0.84147098  0.90929743  0.14112001]
 [-0.7568025  -0.95892427 -0.2794155 ]]
[[ True  True  True]
 [False False False]]
[[3 4 5]
 [7 8 9]]
[[-1  0  1]
 [ 1  2  3]]
[[ 2  4  6]
 [12 15 18]]
```

In [8]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)

print(a.sum(), a.prod(), a.min(), a.max())
print(a.sum(axis=0), a.sum(axis=1))
print(a.cumsum(axis=1))  #sumas acumuladas por filas
```

```
[[1 2 3]
 [4 5 6]]
21 720 1 6
[5 7 9] [ 6 15]
[[ 1  3  6]
 [ 4  9 15]]
```

Observaciones:

- Las funciones como el seno, coseno o exponencial actúan también elemento a elemento.
- El producto de matrices se realiza con la operación "@", o la función "dot".

Operaciones básicas con vectores

In [9]:
```
# Operaciones matemáticas con vectores:

u = [1, 2, 3]
v = [4, 5, 6]

# Producto interior o escalar:
print("u . v = ", np.inner(u, v))

# Producto vectorial o producto cruzado:
print("u x v = ", np.cross(u, v))

# Producto combinado (cada u_i * cada v_j):
print("u xx v = ", np.outer(u, v))
```

```
u . v =  32
u x v =  [-3  6 -3]
u xx v =  [[ 4  5  6]
 [ 8 10 12]
 [12 15 18]]
```

Operaciones básicas con matrices

In [10]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[10, 20], [30, 40], [50, 60]])
print(a)
print(b)

# Producto de matrices:

print(a@b)
print(a.dot(b))

# Producto "escalar" de matrices:
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
print(np.vdot(a, b))    # 1*5  +  2*6  +  3*7  +  4*8
```

```
[[1 2 3]
 [4 5 6]]
[[10 20]
 [30 40]
 [50 60]]
[[220 280]
 [490 640]]
[[220 280]
 [490 640]]
70
```

Funciones básicas con matrices

In [11]:
```
a = np.array([[1.0, 2.0], [3.0, 4.0]])
print(a)

# Traspuesta de una matriz:
print(a.transpose())

# Inversa de una matriz:
print(np.linalg.inv(a))

# Matriz unidad:
u = np.eye(2)                                # unit 2x2 matrix; "eye" repres
print(u)

j = np.array([[0.0, -1.0], [1.0, 0.0]])

print(j @ j)                                 # producto de matrices

print(np.trace(u))                           # trace = suma de la diagonal p
2.0
```

```
[[1. 2.]
 [3. 4.]]
[[1. 3.]
 [2. 4.]]
[[-2.   1. ]
 [ 1.5 -0.5]]
[[1. 0.]
 [0. 1.]]
[[-1.  0.]
 [ 0. -1.]]
2.0
```

Out[11]:
```
2.0
```

Álgebra lineal básica:

In [12]:
```
# Resolución de ecuaciones lineales:

a = np.array([[1.0, 2.0], [3.0, 4.0]])
print(a)
y = np.array([[5.], [7.]])
print(y)

print(np.linalg.solve(a, y))

# Vectores y valores propios:

print(np.linalg.eig(j))
```

```
[[1. 2.]
 [3. 4.]]
[[5.]
 [7.]]
[[-3.]
 [ 4.]]
(array([0.+1.j, 0.-1.j]), array([[0.70710678+0.j        , 0.70710678-0.
j        ],
       [0.        -0.70710678j, 0.        +0.70710678j]]))
```

Iteración con arrays

Es mejor que lo veas con un ejemplo.

In [13]:
```
a = np.array([[1, 2, 3], [4, 5, 6]])
print(a)
print("líneas:")
for linea in a:
    print(" -> ", linea)
print("elementos:")
for elemento in a.flat:
    print(" -> ", elemento)
```

```
[[1 2 3]
 [4 5 6]]
líneas:
 ->  [1 2 3]
 ->  [4 5 6]
elementos:
 ->  1
 ->  2
 ->  3
 ->  4
 ->  5
 ->  6
```

Estadística básica

Funciones típicas: media, varianza, covarianza, desviación típica.

In [14]:
```
# Media:

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

print(a)
print("Media de una matriz:", np.mean(a))
print("   ... por columnas: ", np.mean(a, axis=0))
print("   ... por filas: ", np.mean(a, axis=1))
print("... mdia ponderada: ", np.average([1, 2, 3, 4], weights=[0.1, 0.3

# Mediana:
print("Mediana de un vector: ", np.median(np.array([4, 2, 5, 3, 7, 6, 1,

# Varianza:
print("Varianza de una matriz: ", np.var(a))

# Desviacón típica
print("Desviación típica: ", np.std(a))

# Matriz de covarianzas:
x = [-2.1, -1,  4.3]
y = [3,  1.1,  0.12]

print("Covarianza de x", np.cov(x))
print("Covarianza de x e y: ", np.cov(x, y))

xy = np.stack((x, y), axis=0) # Combinamos ambos vectores
print(np.cov(xy))
```

```
[[1 2]
 [3 4]]
Media de una matriz: 2.5
   ... por columnas:  [2. 3.]
   ... por filas:  [1.5 3.5]
... mdia ponderada:  3.115942028985507
Mediana de un vector:  4.0
Varianza de una matriz:  1.25
Desviación típica:  1.118033988749895
Covarianza de x 11.709999999999999
Covarianza de x e y:  [[11.71       -4.286     ]
 [-4.286       2.14413333]]
[[11.71       -4.286     ]
 [-4.286       2.14413333]]
```

Histogramas

Se incluye este ejemplo final para dejar buen sabor de boca y para que te itereses por otra librería útil: matplotlib, que permite crear gráficos claros, de una forma sencilla.

In [15]:
```
import numpy as np
import matplotlib.pyplot as plt

# Construimos un vector de 10000 elementos, generados aleatoriamente sig
# esto es, una distribución normal con media = 2 y desviación típica = 0

v = np.random.normal(2, 0.5, 10000)

# Ahora trazamos el histograma normalized histogram with 50 bins

plt.hist(v, bins=50)       # matplotlib version (plot)
plt.show()
```

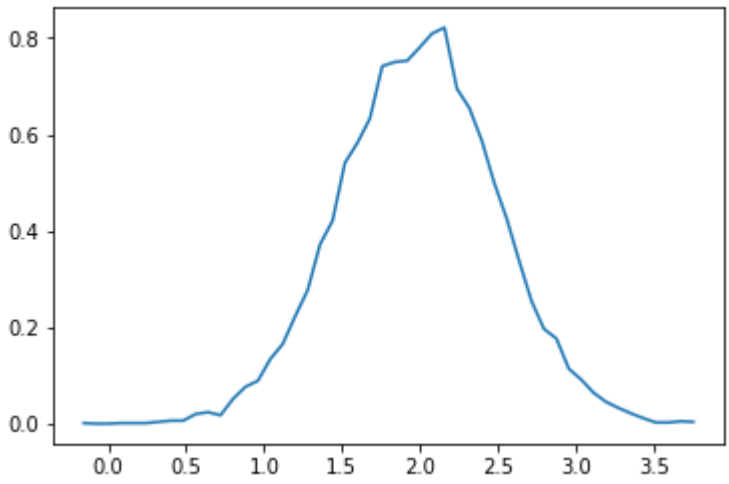
```
<Figure size 640x480 with 1 Axes>
```

In [16]:
```
# Ahora, computamos el histograma con numpy y luego lo trazamos:

(n, bins) = np.histogram(v, bins=50, density=True)  # NumPy version (no

plt.plot(.5*(bins[1:]+bins[:-1]), n)

plt.show()
```



Brevísimo comentario final

La librería numpy contiene otras funciones y operaciones útiles, pero que nos ha parecido mejor dejar para una lectura más avanzada:

- Funciones para ordenar arrays: argmax, argmin, argsort, max, min, ptp, searchsorted, sort
- Funciones de conversión de tipos: ndarray.astype, atleast_1d, atleast_2d, atleast_3d, mat
- Funciones para cortar arrays y combinarlos por bloques: array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack
- Y un largo etcétera. V. "quickstart", en la lista de referencias siguiente.

He aquí algunas referencias recomendables para complementar el material de este sencillo script:

- http://www.numpy.org/

- https://www.w3schools.in/python-data-science/introduction-to-numpy-library/
- https://docs.scipy.org/doc/numpy/user/quickstart.html