Herramientas de programación en Python

2. Tipos de datos estructurados

2.3 homogéneos dinámicos: array de Numpy

Pedro Gomis

pedro.gomis@upc.edu

NumPy

Tipos de datos **array**

- Los arreglos de números o tablas se pueden trabajar mucho más eficientemente con los objetos ***array*** *del paquete (package) de NumPy*.*
- Los ***arrays*** de *NumPy* son datos estructurados ***homogéneos***, similares a las listas (éstas son ***heterogéneas***), pero enfocados al **cálculo numérico usando vectores y matrices**.
- Los ***arrays*** suelen contener números enteros, reales y complejos.
- Los números **complejos** son otro tipo de **dato numérico** en Python:

```
>>> c = 2+3j   # se pueden escribir con espacios: 2 + 3j
>>> type(c)
<class 'complex'>
```

- Para crear un ***array*** se importa la librería de cualquiera de las formas que hemos indicado previamente

```
>>> import numpy as np
>>> a = np.array([1, 2, 3])
>>> a
array([1, 2, 3])
```

* http://www.numpy.org/ Disponible en Spyder de Winpython, http://winpython.sourceforge.net/, anaconda, https://anaconda.org/anaconda/numpy y en http://scipy.org/

NumPy

Vectores y matrices con NumPy

Vectores

- Los **vectores** se utilizan en matemáticas y procesamiento de datos y señales para agrupar valores, como:
  - las tres coordenadas de un punto en el espacio (x, y, z), o (x_1, x_2, x_3).
  - Las soluciones de un conjunto de ***n*** ecuaciones pueden agruparse en el vector (x_1, x_2, x_3, …, x_n).
  - Los datos numéricos de una señal muestreada en el tiempo se pueden agrupar en un vector.

Ejemplo: Suma de vectores con *arrays*

```
import numpy as np
v1 = np.array([1, 2, 3.2])
v2 = np.array([5, 5, 5])
v3 = v1 + v2
print(v3)
```

```
[ 6.   7.   8.2]        # observar que los números son todos reales
```

NumPy

Vectores y matrices con NumPy

Vectores

- La inicialización de los vectores se hace para prefijar su tamaño. Es una buena opción crear un vector de ceros o de unos, que luego se le añaden los elementos.

Ejemplo: Suma de vectores con *arrays* leyendo los valores

```
import numpy as np
v1 = np.zeros(3) # se crea un vector de 3 ceros (reales)
print("Introduce vector 1:")
for i in range(len(v1)):
    v1[i]=int(input("Elemento "+str(i)+": "))
v2 = np.zeros(3)
print("Introduce vector 2:")
for i in range(len(v1)):
    v2[i]=int(input("Elemento "+str(i)+": "))
v3 = v1 + v2
print(v3)
```

- Los objetos *array* se indexan y recortan de igual forma que las listas.
- También incluyen métodos asociados para hallar varias funciones matemáticas sobre sus elementos, como la **suma, la media, desviación estándar, el máximo**, etc.

NumPy

Vectores y matrices con NumPy

Matrices

- Las matrices que se presentaron con listas anidadas pueden también operarse con *arrays*. Veamos varios ejemplos:

Ejemplo: Suma de matrices con *arrays*

```
>>> m1 = np.array([[1, 1],[1, 1]])
>>> m1
array([[1, 1],
       [1, 1]])
>>> m2 = np.array([[1, 0],[0, 1]])
>>> m2
array([[1, 0],
       [0, 1]])
>>> m3 = m1 + m2
>>> m3
array([[2, 1],
       [1, 2]])
```

NumPy

Vectores y matrices con NumPy

Operaciones con vectores en lugar de iteraciones

- En los *arrays* de Numpy, se pueden realizar directamente operaciones con funciones en lugar de recorrerlas con una composición iterativa.
- Se puede **vectorizar** la operación.
- Algunas funciones internas de Python, como max(), min() y sum(), también se pueden aplicar sobre una secuencia tipo tupla, lista o array.
- Supongamos que queremos aplicar una función matemática a un vector. En lugar de utilizar un bucle para recorrer el vector, podemos hacer

```
import numpy as np
def f(x):
    return 4*2**x
a = np.array(range(1,11))
y = f(a)
print('El vector',a)
print('Aplicada la función f(x) = 4*2**x se obtiene')
print(y)
```

```
El vector [ 1  2  3  4  5  6  7  8  9 10]
Aplicada la función f(x) = 4*2**x se obtiene
[   8   16   32   64  128  256  512 1024 2048 4096]
```

NumPy

Vectores y matrices con NumPy

Operaciones con vectores en lugar de iteraciones

- La secuencia de la dicotomía de Zenon, representada anteriormente $\sum 2^{-i}$, puede calcularse con una función definida y calcular la **sumatoria** con la función interna `sum()` sobre un array:

```
import numpy as np
def f(x):
    return 1/2**x
N = int(input('Entra el número de términos: ' ))
a = np.array(range(1,N+1))
y = f(a)
print('El vector',y)
print('La suma es', sum(y))
```

```
Entra el número de términos: 7
El vector [0.5    0.25   0.125   0.0625    0.03125   0.015625    0.0078125]
La suma es 0.9921875
```

NumPy

Vectores y matrices con NumPy

Operaciones con vectores en lugar de iteraciones

- El cálculo de sumatorias realizado con esquemas de recorrido con iteraciones definidas, se puede sustituir y realizar con más eficacia con la función interna **sum**() en una secuencia numérica (tupla, lista o array). También se puede hallar su mayor o menor valor:

```
>>> Lista = [1, 2, 3, 4]
>>> sum(Lista)                  # devuelve valor 10
>>> max(Lista)                  # devuelve valor 4
>>> min(Lista)                  # devuelve valor 1
>>> T = (1, 2, 3, 4, 2, 4, 0)
>>> max(T)                      # devuelve valor 4
>>> min(T)                      # devuelve valor 0
>>> sum(T)                      # devuelve valor 16
>>> media = sum(T)/len(T)       # Cálculo del valor medio
>>> media
2.2857142857142856
```

Vectores y matrices con NumPy

NumPy

Operaciones con matrices en NumPy

- Pre asignación de ceros para trabajar con matrices

```
import numpy as np
# Matriz de 4x3 ceros
m1 = np.zeros([4, 3])
# Matriz de 2x3 unos
m2 = np.ones([2,3])
```

- Leer, recorriendo, una matriz

```
FILAS = 4
COLS = 3
m = np.zeros([FILAS, COLS])
# Leer del teclado los valores de los elementos de la matriz
for i in range(FILAS):
    for j in range(COLS):
            m[i][j] = float(input('m[' +str(i)+'][' +str(j)+']: '))
```

- Dimensiones de una matriz

```
m.shape
>>> (4,3)
m.shape[0]
>>> 4            # Así sabemos el número de filas
m.shape[1]
>>> 3            # o el número de columnas
```

NumPy

Vectores y matrices con NumPy

Operaciones con matrices en NumPy

- Producto y división *punto-a-punto* de matrices con NumPy

```
m1 = np.array([[1,1],[2,2]])
>>> m1
array([[1, 1],
       [2, 2]])
>>> m2 = np.array([[3,3],[4,4]])
>>> m2
array([[3, 3],
       [4, 4]])
>>> m1*m2
array([[3, 3],
       [8, 8]])
>>> m1/m2
array([[ 0.33333333,  0.33333333],
       [ 0.5       ,  0.5       ]])
>>> m2/m1
array([[ 3.,  3.],
       [ 2.,  2.]])
```

NumPy

Vectores y matrices con NumPy

Operaciones con matrices en NumPy

- Producto de matrices con NumPy

```
m1 = np.array([[1,1],[2,2]])
>>> m1
array([[1, 1],
       [2, 2]])
>>> m2 = np.array([[1,2],[-3,5]])
>>> m2
array([[ 1,  2],
       [-3,  5]])
>>> np.dot(m1,m2)
array([[-2,  7],
       [-4, 14]])
```