12_Numpy_Introduccio

November 24, 2014

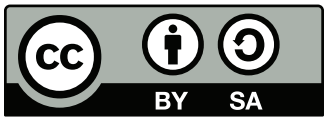

Figure 1: BY-SA

Authors : *Sonia Estradé*
José M. Gómez
Ricardo Graciani
Manuel López
Xavier Luri
Josep Sabater

1 Llibreria numpy

Numpy és una llibreria de python per a càlcul científic que conté una gran varietat d'eines per a la **creació, manipulació i càlculs de vectors i matrius**. L'avantatge respecte l'ús de llistes normals de python per a fer aquest tipus de càlculs és que les eines de numpy són més senzilles, completes i **molt més ràpides**.

Podeu veure una introducció a numpy aquí: What is numpy? i diversos exemples d'ús aquí: Numpy examples

Nota: per a usar la llibreria *numpy* cal fer la importació:

```
import numpy  *
```

1.1 ndarray

L'objecte bàsic de numpy és `ndarray`. Aquest tipus d'objecte és una extensió de les llistes normals de python amb adaptacions pensades per al càlcul numèric. En aquest sentit ja inclou conceptes com a *dimensions* i *eixos*.

- `ndarray.ndim` dóna la dimensió de la llista
- `ndarray.shape` dóna la "forma" de la llista (n1 x n2 x n3 x ...)
- `ndarray.size` número de components de la llista

Exemple:

```
In [1]: import numpy as np

        # Generem un ndarray a partir de una llista arange()
        # reshape() converteix la llista en una matriu 3 x 5
        un_array = np.arange(15).reshape(3, 5)
```

```
        print(un_array)

        # Comprovem la dimensió
        print("Dimensió: ",un_array.ndim)

        # Comprovem la forma
        print("Forma: ",un_array.shape)

        # Comprovem el numero total d'elements
        print("Número de components:", un_array.size)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
Dimensió:  2
Forma:  (3, 5)
Número de components: 15
```

1.2 Creació de ndarrays

Numpy ofereix diverses formes per crear objectes `ndarray` (ja hem vist en l'exemple anterior una forma simple de crear-ne un).

1.2.1 A partir d'una llista

La manera més simple de crear un objecte `ndarray` és a partir d'una llista normal, utilitzant la funció `array()`. Aquesta funció convertirà una llista en un objecte `ndarray` de les mateixes dimensions.

```
In [2]: import numpy as np

        # Creem una llista 1D amb nou elements i la convertim en una matriu 1D
        llista_1D = range(9)
        print(llista_1D)
        print(type(llista_1D))
        array_1D = np.array(llista_1D)
        print(array_1D)
        print("Forma: ", array_1D.shape)
        print("Mida: ", len(array_1D.shape))
        print(type(array_1D))

        # Fem el mateix partint d'una llista 2D
        llista_2D = [ [1,2,3], [4,5,6], [7,8,9] ]
        array_2D = np.array(llista_2D)
        print(llista_2D)
        print(array_2D)
        print("Forma: ", array_2D.shape)
        print("Mida: ", len(array_2D))
        print(type(array_2D))

        array_vell= array_2D
        array_2D=array_2D+1.9
        print(array_vell[1][1], type(array_vell[1][1]))
        print(array_2D[1][1], type(array_2D[1][1]))
```

```
range(0, 9)
<class 'range'>
```

```
[0 1 2 3 4 5 6 7 8]
Forma:  (9,)
Mida:  1
<class 'numpy.ndarray'>
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Forma:  (3, 3)
Mida:  3
<class 'numpy.ndarray'>
5 <class 'numpy.int64'>
6.9 <class 'numpy.float64'>
```

Cal tenir en compte que els objectes `ndarray` són homogenis, totes les seves components són del mateix tipus bàsic. Si és necessari es pot especificar aquest tipus quan es crea l'objecte amb la funció `array()`.

```
In [3]: import numpy as np

        un_array = np.array( [ [1,2], [3,4] ], dtype=complex )
        print(un_array) # Noteu que tots els enters s'han convertit en complexes
```

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

1.2.2 Usant la funció `zeros()`

Aquesta funció crea un array de les mides donades amb totes les components a zero:

```
In [4]: import numpy as np

        array_Z = np.zeros( (5,5) )
        print(array_Z)
```

```
[[ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.]]
```

1.2.3 Usant la funció `ones()`

De forma similar la funció `ones()` crea un array de la mida donada ple de uns.

```
In [5]: import numpy as np

        array_U = np.ones( (5,5) )
        print(array_U)
```

```
[[ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]
 [ 1.  1.  1.  1.  1.]]
```

1.2.4 Usant la funció `eye()`

Aquesta funció retorna una matriu identitat (usualment s'escriu I, que es pronuncia 'eye' en anglès):

```
In [6]: import numpy as np

        # Generem la identitat 3x3
        I = np.eye(3)

        print("Matriu identitat 3x3")
        print(I)
```

```
Matriu identitat 3x3
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

1.2.5 Usant les funcions `arange()` o `linspace()`

La funció `arange()` permet crear seqüències numèriques regulars (de forma similar a `range()` però més general). La seva sintaxi és

```
arange(min,max,pas)   *màxim no inclòs*
```

i el resultat de la funció és un objecte `ndarray()`:

```
In [7]: import numpy as np

        llista = np.arange(0.,4.5,0.5)
        print(llista)
        print(type(llista))
```

```
[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4. ]
<class 'numpy.ndarray'>
```

Com que el resultat és un objecte `ndarray` podem usar directament `reshape` per modificar la seva forma:

```
In [8]: import numpy as np

        un_array = np.arange(0.,4.5,0.5).reshape( (3,3) )
        print(un_array)
```

```
[[ 0.   0.5  1. ]
 [ 1.5  2.   2.5]
 [ 3.   3.5  4. ]]
```

La funció `ndarray()` té el problema de que, donada la precisió finita dels càlculs amb *float* a vegades no es pot predir exactament el número d'elements que generarà. Quan això sigui important es pot usar la funció `linspace()` que permet especificar exactament el número d'elements:

```
linspace(min, max, num_elements)
```

```
In [9]: import numpy as np
        import math

        vector = np.linspace(0.,math.pi,10)
        print(vector)
```

```
[ 0.          0.34906585  0.6981317   1.04719755  1.3962634   1.74532925
  2.0943951   2.44346095  2.7925268   3.14159265]
```

1.2.6 A partir d'una funció

Es pot crear un objecte `ndarray` usant una funció. La funció ha de rebre com a paràmetres els índexs d'una component i retornar el valor de la component que correspongui a aquests índexs.

```
In [10]: import numpy as np

         # Definim la funció
         def funcio(i,j):
             """Reb com a paràmetres els dos índex i1,i2 i assigna com
                a valor de la component la súma dels dos"""
             return i+j

         # Creem una matriu entera a partir de la funció
         A = np.fromfunction(funcio,(5,5),dtype=float)
         print("Matriu i1+i2:")
         print(A)

Matriu i1+i2:
[[ 0.  1.  2.  3.  4.]
 [ 1.  2.  3.  4.  5.]
 [ 2.  3.  4.  5.  6.]
 [ 3.  4.  5.  6.  7.]
 [ 4.  5.  6.  7.  8.]]
```

1.3 Còpia de `ndarray`

Important: tingueu en compte que, al contrari que amb els tipus bàsics de dades quan, es fa un assignament a una nova variable d'un `ndarray` no es copia, sinó que les dues variables corresponen al mateix objecte:

```
In [11]: import numpy as np

         A = np.arange(10)
         print("Array A")
         print(A)

         B = A
         print("Array B")
         print(B)

         # Mofiquem A
         A[0]=10
         print("Array A modificat")
         print(A)

         # Comprovem que B també s'ha modificat
         print("Array B modificat?")
         print(B)

         # Tanmateix, si reassigneu A aleshores B no es modifica!!
         A = np.arange(2)
         print("A reassignat")
         print(A)
         print("B com queda?")
         print(B)
```

```
Array A
[0 1 2 3 4 5 6 7 8 9]
Array B
[0 1 2 3 4 5 6 7 8 9]
Array A modificat
[10  1  2  3  4  5  6  7  8  9]
Array B modificat?
[10  1  2  3  4  5  6  7  8  9]
A reassignat
[0 1]
B com queda?
[10  1  2  3  4  5  6  7  8  9]
```

Si es vol realment fer una còpia d'un array per assignar-lo a una nova variable s'ha d'usar la funció `copy()`:

```
In [12]: import numpy as np

         A = np.arange(10)
         print("Array A")
         print(A)

         B = A.copy()
         print("Array B")
         print(B)

         # Mofiquem A
         A[0]=10
         print("Array A modificat")
         print(A)

         # Comprovem que B també s'ha modificat
         print("Array B modificat?")
         print(B)
```

```
Array A
[0 1 2 3 4 5 6 7 8 9]
Array B
[0 1 2 3 4 5 6 7 8 9]
Array A modificat
[10  1  2  3  4  5  6  7  8  9]
Array B modificat?
[0 1 2 3 4 5 6 7 8 9]
```

1.4 Canvi de forma d'un objecte `ndarray`

La "forma" d'un objecte `ndarray` o d'una llista es pot canviar amb la funció `reshape()`

```
In [13]: import numpy as np

         # Creem una matriu 1D
         llista_1D = range(9)
         array_1D = np.array(llista_1D)

         # Convertim la matriu_1D de 9 elements en una matriu 3x3
```

```
        array_2D = np.reshape(llista_1D,(3,3))
        print("Array 2D")
        print(array_2D)
        print()

        # Fem el mateix però creant una matriu 3D a partir d'una llista
        llista_1D = range(27)
        array_3D = np.reshape(llista_1D,(3,3,3))
        print("Array 3D")
        print(array_3D)
```

```
Array 2D
[[0 1 2]
 [3 4 5]
 [6 7 8]]

Array 3D
[[[ 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]]]
```

1.5 Accés als elements

Els elements d'un objecte **ndarray** es poden accedir com els de les llistes normals de python, donant els índexs corresponents:

```
In [14]: import numpy as np

         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print(un_array)
         print("Element [0][0]: ",un_array[0][0])
         print("Element [2][2]: ",un_array[2][2])
```

```
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
Element [0][0]:  1.0
Element [2][2]:  9.0
```

Però en el cas de **ndarray** també es pot indicar un element indicant els dos índexs en forma de tupla:

```
In [15]: import numpy as np

         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print(un_array)
         print("Element [0][0]: ",un_array[0][0])
         print("Element [0,0]: ",un_array[0,0])
         print("Element [2][2]: ",un_array[2][2])
         print("Element [2,2]: ",un_array[2,2])
```

```
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
Element [0][0]:  1.0
Element [0,0]:  1.0
Element [2][2]:  9.0
Element [2,2]:  9.0
```

També com en el cas de les llistes es poden seleccionar subconjunts indicant un rang d'índexs i el pas:

```
In [16]: import numpy as np

         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print("Array")
         print(un_array)
         print("Elements [0:2,0]: ")
         print(un_array[0:2,0])
         print("Elements [0:3:2,0]: ")
         print(un_array[0:3:2,0])
         print("Elements [0:2,0:2]: ")
         print(un_array[0:2,0:2])
         print("Elements [0:3,1:2]: ")
         print(un_array[0:3,1:2])
```

```
Array
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
Elements [0:2,0]:
[ 1.  4.]
Elements [0:3:2,0]:
[ 1.  7.]
Elements [0:2,0:2]:
[[ 1.  2.]
 [ 4.  5.]]
Elements [0:3,1:2]:
[[ 2.]
 [ 5.]
 [ 8.]]
```

1.6 Inserció i esborrat d'elements

Per a afegir o esborrar elements cal usar les funcions `insert()`, `append()` i `delete()`. Noteu que aquestes operacions **no modifiquen l'array original i retornen un nou array amb els canvis**.

Cal anar amb compte amb arrays de dimensió més gran que 1. En aquest cas possiblement caldrà especificar el paràmetre *axis* per què el resultats sigui el desitjat.

```
In [17]: import numpy as np

         # En una dimensió
         un_array = np.linspace(1.,5.,5)
         print("Array original 1D: ", un_array)

         print("append():", np.append(un_array,un_array))
         print("insert(pos 1):", np.insert(un_array,1,0))
```

```
print("delete(pos 1):", np.delete(un_array,1))

# En dos dimensions
un_array = np.linspace(1.,9.,9).reshape( (3,3) )
print("Array original 2D: ", un_array)

print("append() 2D:", np.append(un_array,un_array))
print("append(axis=0) 2D:")
print(np.append(un_array,un_array,axis=0))
print("append(axis=1) 2D:")
print(np.append(un_array,un_array,axis=1))
```

```
Array original 1D:  [ 1.  2.  3.  4.  5.]
append(): [ 1.  2.  3.  4.  5.  1.  2.  3.  4.  5.]
insert(pos 1): [ 1.  0.  2.  3.  4.  5.]
delete(pos 1): [ 1.  3.  4.  5.]
Array original 2D:  [[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
append() 2D: [ 1.  2.  3.  4.  5.  6.  7.  8.  9.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
append(axis=0) 2D:
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]
 [ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
append(axis=1) 2D:
[[ 1.  2.  3.  1.  2.  3.]
 [ 4.  5.  6.  4.  5.  6.]
 [ 7.  8.  9.  7.  8.  9.]]
```

1.7 Operacions amb ndarray

Un dels grans avantatges dels objecte **ndarray** es que es poden realitzar directament operacions amb valors numèrics o amb altres arrays, senzillament usant els operadors habituals +,-,*,/,**,%

Noteu que realitzar les operacions d'aquesta manera és molt més ràpid que fer-ho usant bucles *for*.

1.7.1 Operacions amb valors numèrics

```
In [18]: import numpy as np

A = np.linspace(1.,9.,9).reshape( (3,3) )
print("Array A:")
print(A)

# Suma i resta per un valor
C = A+2
print("Array C=A+2:")
print(C)
D = A-2
print("Array D=A-2:")
print(D)

# Producte i divisió
```

```
E = A*2
print("Array E=A*2:")
print(E)
F = A/2
print("Array F=A/2:")
print(F)

# Exponenciació
G = A**2
print("Array G=A**2:")
print(G)

# Mòdul
H = A%2
print("Array H=A%2:")
print(H)

# Operadors booleans
I = A>5
print("Array I=A>5:")
print(I)
```

```
Array A:
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
Array C=A+2:
[[  3.   4.   5.]
 [  6.   7.   8.]
 [  9.  10.  11.]]
Array D=A-2:
[[-1.  0.  1.]
 [ 2.  3.  4.]
 [ 5.  6.  7.]]
Array E=A*2:
[[  2.   4.   6.]
 [  8.  10.  12.]
 [ 14.  16.  18.]]
Array F=A/2:
[[ 0.5  1.   1.5]
 [ 2.   2.5  3. ]
 [ 3.5  4.   4.5]]
Array G=A**2:
[[  1.   4.   9.]
 [ 16.  25.  36.]
 [ 49.  64.  81.]]
Array H=A%2:
[[ 1.  0.  1.]
 [ 0.  1.  0.]
 [ 1.  0.  1.]]
Array I=A>5:
[[False False False]
 [False False  True]
 [ True  True  True]]
```

1.7.2 Operacions component a component

In [19]:
```
import numpy as np

A = np.linspace(1.,9.,9).reshape( (3,3) )
B = np.linspace(10.,18.,9).reshape( (3,3) )
print("Array A:")
print(A)
print("Array B:")
print(B)

# Suma i resta de matrius
C = A+B
print("Array C=A+B:")
print(C)
D = A-B
print("Array D=A-B:")
print(D)

# Producte i divisió (component a component)
E = A*B
print("Array E=A*B:")
print(E)
F = A/B
print("Array F=A/B:")
print(F)

# Exponenciació
G = A**B
print("Array G=A**B:")
print(G)

# Operacions booleanes
H = A>B
print("Array H=A>B:")
print(H)
```

```
Array A:
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
Array B:
[[ 10.  11.  12.]
 [ 13.  14.  15.]
 [ 16.  17.  18.]]
Array C=A+B:
[[ 11.  13.  15.]
 [ 17.  19.  21.]
 [ 23.  25.  27.]]
Array D=A-B:
[[-9. -9. -9.]
 [-9. -9. -9.]
 [-9. -9. -9.]]
Array E=A*B:
[[  10.   22.   36.]
```

```
 [  52.   70.   90.]
 [ 112.  136.  162.]]
Array F=A/B:
[[ 0.1         0.18181818  0.25      ]
 [ 0.30769231  0.35714286  0.4       ]
 [ 0.4375      0.47058824  0.5       ]]
Array G=A**B:
[[  1.00000000e+00   2.04800000e+03   5.31441000e+05]
 [  6.71088640e+07   6.10351562e+09   4.70184985e+11]
 [  3.32329306e+13   2.25179981e+15   1.50094635e+17]]
Array H=A>B:
[[False False False]
 [False False False]
 [False False False]]
```

1.7.3 Operacions unitaries

Numpy inclou algunes operacions unitàries (que s'apliquen a un únic objecte `ndarray`:

- `sum()` suma de totes les components (o de les d'un eix)
- `min()` retorna la component amb el mínim valor
- `max()` retorna la component amb el màxim valor

```
In [20]: import numpy as np

         A = np.linspace(1.,9.,9).reshape( (3,3) )
         print("Array A: ")
         print(A)

         # Suma de totes les components
         print("Suma global: ", A.sum())
         print("Suma columnes: ", A.sum(axis=0))
         print("Suma files: ", A.sum(axis=1))

         # Màxim
         print("Màxim global: ",A.max())
         print("Màxim de cada columna: ",A.max(axis=0))
         print("Màxim de cada fila: ",A.max(axis=1))

         # Mínim
         print("Mínim global: ",A.min())
         print("Mínim de cada columna: ",A.min(axis=0))
         print("Mínim de cada fila: ",A.min(axis=1))
```

```
Array A:
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
Suma global:  45.0
Suma columnes:  [ 12.  15.  18.]
Suma files:  [  6.  15.  24.]
Màxim global:  9.0
Màxim de cada columna:  [ 7.  8.  9.]
Màxim de cada fila:  [ 3.  6.  9.]
Mínim global:  1.0
```

```
Mínim de cada columna:  [ 1.  2.  3.]
Mínim de cada fila:  [ 1.  4.  7.]
```

1.7.4 Funcions

Numpy també extén les funcions habituals de manera que es poden aplicar directamet als `ndarray`. En altres paraules, es pot aplicar una funció a cadascuna de les components de `ndarray` amb una sola crida.

```
In [21]: import numpy as np

         A = np.linspace(0,np.pi,9).reshape( (3,3) )
         print("Array A: ")
         print(A)

         # sin()
         print("sin(A): ")
         print(np.sin(A))

         # exp()
         print("exp(A): ")
         print(np.exp(A))

         # log()
         print("log(A): ")
         print(np.log(A))

         # sqrt()
         print("sqrt(A): ")
         print(np.sqrt(A))
```

```
Array A:
[[ 0.          0.39269908  0.78539816]
 [ 1.17809725  1.57079633  1.96349541]
 [ 2.35619449  2.74889357  3.14159265]]
sin(A):
[[  0.00000000e+00   3.82683432e-01   7.07106781e-01]
 [  9.23879533e-01   1.00000000e+00   9.23879533e-01]
 [  7.07106781e-01   3.82683432e-01   1.22464680e-16]]
exp(A):
[[  1.           1.48097267   2.19328005]
 [  3.24818781   4.81047738   7.12418553]
 [ 10.55072407  15.62533401  23.14069263]]
log(A):
[[       -inf -0.93471166 -0.24156448]
 [ 0.16390063  0.45158271  0.67472626]
 [ 0.85704781  1.01119849  1.14472989]]
sqrt(A):
[[ 0.          0.62665707  0.88622693]
 [ 1.08540188  1.25331414  1.4012478 ]
 [ 1.53499006  1.65797876  1.77245385]]

-c:17: RuntimeWarning: divide by zero encountered in log
```