12 - Numpy

November 16, 2015

Figure 1: BY-SA

Authors: Sonia Estrad´e

Jos´e M. G´omez

Ricardo Graciani

Franc, Guell

Manuel L´opez

Xavier Luri

Josep Sabater

1 Llibreria numpy

Numpy ´es una llibreria de python per a c`alcul cient´ıfic que cont´e una gran varietat d’eines per a la creaci´o, manipulaci´o i c`alculs de vectors i matrius. L’avantatge respecte l’´us de llistes normals de python per a fer aquest tipus de c`alculs ´es que les eines de numpy s´on m´es senzilles, completes i molt m´es r`apides.

Podeu veure una introducci´o a numpy aqu´ı: What is numpy? i diversos exemples d’´us aqu´ı: Numpy examples

Nota: per a usar la llibreria *numpy* cal fer la importaci´o:

import numpy \*

1.1 ndarray

L’objecte b`asic de numpy ´es ndarray. Aquest tipus d’objecte ´es una extensi´o de les llistes normals de python amb adaptacions pensades per al c`alcul num`eric. En aquest sentit ja inclou conceptes com a *dimensions* i *eixos*.

*•* ndarray.ndim d´ona la dimensi´o de la llista

*•* ndarray.shape d´ona la “forma” de la llista (n1 x n2 x n3 x . . . )

*•* ndarray.size n´umero d’elements de la llista

Exemple:

In [3]: import numpy as np

*# Generem un ndarray a partir de la funci´o de numpy arange(),*

*# similar a range() per`o que genera ndarrays*

*# reshape() converteix la llista en una matriu 3 x 5*

1

un\_array = np.arange(15).reshape(3, 5)

print(un\_array)

*# Comprovem la dimensi´o*

print("Dimensi´o: ",un\_array.ndim)

*# Comprovem la forma*

print("Forma: ",un\_array.shape)

*# Comprovem el numero total d’elements*

print("N´umero d’elements:", un\_array.size)

[[ 0 1 2 3 4]

[ 5 6 7 8 9]

[10 11 12 13 14]]

Dimensi´o: 2

Forma: (3, 5)

N´umero d’elements: 15

1.2 Creaci´o 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 m´es simple de crear un objecte ndarray ´es a partir d’una llista normal, utilitzant la funci´o array(). Aquesta funci´o convertir`a una llista en un objecte ndarray de les mateixes dimensions.

In [4]: import numpy as np

*# Creem una llista 1D amb nou elements i la convertim en un ndarray 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))

*# Exemple d’operaci´o amb ndarray. L’analitzarem m´es tard*

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]))

2

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.int32’>

6.9 <class ’numpy.float64’>

Cal tenir en compte que els objectes ndarray s´on homogenis, tots els seus elements s´on del mateix tipus b`asic. Si ´es necessari es pot especificar aquest tipus quan es crea l’objecte amb la funci´o 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 funci´o zeros()

Aquesta funci´o crea un ndarray 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 funci´o ones()

De forma similar la funci´o ones() crea un ndarray 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.]]

3

1.2.4 Usant la funci´o eye()

Aquesta funci´o retorna una matriu identitat (usualment s’escriu *I*, que es pronuncia ‘eye’ en angl`es): 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 funci´o arange() permet crear seq¨u`encies num`eriques regulars (de forma similar a range() per`o m´es general). La seva sintaxi ´es

arange(min,max,pas) \*m`axim no incl`os\*

i el resultat de la funci´o ´es 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 ´es 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 funci´o arange() t´e el problema de que, donada la precisi´o finita dels c`alculs amb *float*, a vegades no es pot predir exactament el n´umero d’elements que generar`a. Quan aix`o sigui important es pot usar la funci´o linspace() que permet especificar exactament el n´umero 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]

4

1.2.6 A partir d’una funci´o

Es pot crear un objecte ndarray usant una funci´o. La funci´o ha de rebre com a par`ametres els ´ındexs d’una component i retornar el valor de la component que correspongui a aquests ´ındexs.

In [5]: import numpy as np

*# Definim la funci´o*

def funcio(i,j):

*"""Reb com a par`ametres els dos ´ındex i1,i2 i assigna com*

*a valor de la component la s´uma dels dos"""*

return i+j

*# Creem una matriu de floats a partir de la funci´o*

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 C`opia de ndarray

Important: tingueu en compte que, al contrari que amb els tipus b`asics de dades i de forma similar als conjunts de dades, quan es fa un assignament a una nova variable d’un ndarray no es copia, sin´o que les dues variables corresponen al mateix objecte:

In [6]: 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 tamb´e 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)

5

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 c`opia d’un array per assignar-lo a una nova variable s’ha d’usar la funci´o copy():

In [7]: 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 tamb´e 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 es pot canviar amb la funci´o reshape()

In [8]: 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*

6

array\_2D = np.reshape(llista\_1D,(3,3))

print("Array 2D")

print(array\_2D)

print()

*# Fem el mateix per`o 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 Acc´es als elements

Els elements d’un objecte ndarray es poden accedir com els de les llistes normals de python, donant els ´ındexs corresponents:

In [9]: 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

Per`o en el cas de ndarray tamb´e es pot indicar un element donant els dos ´ındexs en forma de llista: 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])

7

[[ 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

Tamb´e com en el cas de les llistes es poden seleccionar subconjunts indicant un rang d’´ındexs i el pas: In [10]: 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 Inserci´o 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 dimensi´o m´es gran que 1. En aquest cas possiblement caldr`a especificar el par`ametre *axis* per qu`e el resultats sigui el desitjat.

In [11]: import numpy as np

*# En una dimensi´o*

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))

8

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 num`erics o amb altres arrays, senzillament usant els operadors habituals +,-,\*,/,\*\*,% Noteu que realitzar les operacions d’aquesta manera ´es molt m´es r`apid que fer-ho usant bucles *for*.

1.7.1 Operacions amb valors num`erics

In [13]: import numpy as np

A = np.linspace(1.,9.,9).reshape( (3,3) )

print("Array A:")

print(A)

*# Suma i resta d’un valor a tots els elements*

C = A+2

print("Array C=A+2:")

print(C)

D = A-2

print("Array D=A-2:")

print(D)

*# Producte i divisi´o de tots els elements per un valor*

9

E = A\*2

print("Array E=A\*2:")

print(E)

F = A/2

print("Array F=A/2:")

print(F)

*# Exponenciaci´o*

G = A\*\*2

print("Array G=A\*\*2:")

print(G)

*# M`odul*

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]]

10

1.7.2 Operacions element a element

In [14]: 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 (element a element)* C = A+B

print("Array C=A+B:")

print(C)

D = A-B

print("Array D=A-B:")

print(D)

*# Producte i divisi´o (element a element)*

E = A\*B

print("Array E=A\*B:")

print(E)

F = A/B

print("Array F=A/B:")

print(F)

*# Exponenciaci´o (element a element)*

G = A\*\*B

print("Array G=A\*\*B:")

print(G)

*# Operacions booleanes (element a element)* 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.]

11

[ 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 unit`aries (que s’apliquen a un ´unic objecte ndarray):

*•* sum() suma de totes les components (o de les d’un eix)

*•* min() retorna la component amb el m´ınim valor

*•* max() retorna la component amb el m`axim valor

In [15]: 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))

*# M`axim*

print("M`axim global: ",A.max())

print("M`axim de cada columna: ",A.max(axis=0))

print("M`axim de cada fila: ",A.max(axis=1))

*# M´ınim*

print("M´ınim global: ",A.min())

print("M´ınim de cada columna: ",A.min(axis=0))

print("M´ınim 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.]

M`axim global: 9.0

M`axim de cada columna: [ 7. 8. 9.]

M`axim de cada fila: [ 3. 6. 9.]

M´ınim global: 1.0

12

M´ınim de cada columna: [ 1. 2. 3.]

M´ınim de cada fila: [ 1. 4. 7.]

1.7.4 Funcions

Numpy tamb´e ext´en les funcions habituals de manera que es poden aplicar directament als ndarray. En altres paraules, es pot aplicar una funci´o a cadascuna de les components de ndarray amb una sola crida.

In [22]: import numpy as np

A = np.linspace(1,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+0.0001)) *# Sumem 0.0001 per evitar que el zero doni errors*

*# sqrt()*

print("sqrt(A): ")

print(np.sqrt(A))

Array A:

[[ 1. 1.26769908 1.53539816]

[ 1.80309725 2.07079633 2.33849541]

[ 2.60619449 2.87389357 3.14159265]]

sin(A):

[[ 8.41470985e-01 9.54416610e-01 9.99373550e-01]

[ 9.73139260e-01 8.77582562e-01 7.19510518e-01]

[ 5.10183526e-01 2.64513174e-01 1.22464680e-16]]

exp(A):

[[ 2.71828183 3.55266875 4.6431739 ]

[ 6.06841375 7.93113638 10.36562879]

[ 13.54739788 17.70582306 23.14069263]]

log(A):

[[ 9.99950003e-05 2.37282391e-01 4.28854865e-01]

[ 5.89561337e-01 7.27981522e-01 8.49550496e-01]

[ 9.57929477e-01 1.05570255e+00 1.14476172e+00]]

sqrt(A):

[[ 1. 1.12592144 1.23911184]

[ 1.34279457 1.43902617 1.52921398]

[ 1.61437124 1.6952562 1.77245385]]

Noteu que en aquest cas ´es important distingir la funci´o sinus del m`odul math de la funci´o amb el mateix nom del m`odul numpy, d’aqu´ı la import`ancia del prefix i la recomanaci´o d’evitar fer “wild imports”. Noteu tamb´e que amb les funcionalitats d’operacions de numpy el calcul de valors de funcions i la seva representaci´o ´es molt simple. Nom´es cal definir un ndarray amb els valors de x i calcular ndarray amb els valors corresponents de la funci´o, com podem veure en l’exemple seg¨uent:

13

In [25]: %pylab inline

import numpy as np

import matplotlib.pyplot as plt

*# Calculem valors de x entre -2\*pi i 2\*pi*

x = np.linspace(0., np.pi, 25, endpoint=True)

*# Calculem funcions d’aquests valors*

lineal = 2\*x+3

sinusoidal = np.sin(x)

exponencial = np.exp(x)

print ("x = ", x)

print ("lineal = ", lineal)

*# Els representem (usem mathplotlib, que explicarem en les sessions seg¨uents)* plt.plot(x, lineal, color = ’red’)

plt.plot(x, sinusoidal, color = ’blue’)

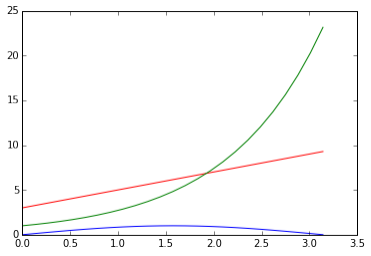
plt.plot(x, exponencial, color = ’green’)

plt.show()

Populating the interactive namespace from numpy and matplotlib

x = [ 0. 0.13089969 0.26179939 0.39269908 0.52359878 0.65449847 0.78539816 0.91629786 1.04719755 1.17809725 1.30899694 1.43989663 1.57079633 1.70169602 1.83259571 1.96349541 2.0943951 2.2252948 2.35619449 2.48709418 2.61799388 2.74889357 2.87979327 3.01069296 3.14159265]

lineal = [ 3. 3.26179939 3.52359878 3.78539816 4.04719755 4.30899694 4.57079633 4.83259571 5.0943951 5.35619449 5.61799388 5.87979327 6.14159265 6.40339204 6.66519143 6.92699082 7.1887902 7.45058959 7.71238898 7.97418837 8.23598776 8.49778714 8.75958653 9.02138592 9.28318531]

14

1.8 Acumulaci´o i separaci´o d’arrays

Numpy tamb´e ofereix funcions per ajuntar dos ndarray, verticalment o horitzontalment: In [26]: import numpy as np

A = np.linspace(1.,9.,9).reshape( (3,3) )

B = np.linspace(10.,18.,9).reshape( (3,3) )

C = np.linspace(19.,27.,9).reshape( (3,3) )

print ("Array A:")

print (A)

print ("Array B:")

print (B)

print ("Array C:")

print (C)

*# Els ajuntem horitzontalment*

print ("hstack((A,B))")

print (np.hstack( (A,B) )) *# Noteu que passem com a argument una tupla* print ("hstack((A,B,C))")

print (np.hstack( (A,B,C) ))

*# Els ajuntem verticalment*

print ("vstack((A,B))")

print (np.vstack( (A,B) ))

print ("vstack((A,B,C))")

print (np.vstack( (A,B,C) ))

Array A:

[[ 1. 2. 3.]

[ 4. 5. 6.]

[ 7. 8. 9.]]

Array B:

[[ 10. 11. 12.]

[ 13. 14. 15.]

[ 16. 17. 18.]]

Array C:

[[ 19. 20. 21.]

[ 22. 23. 24.]

[ 25. 26. 27.]]

hstack((A,B))

[[ 1. 2. 3. 10. 11. 12.]

[ 4. 5. 6. 13. 14. 15.]

[ 7. 8. 9. 16. 17. 18.]]

hstack((A,B,C))

[[ 1. 2. 3. 10. 11. 12. 19. 20. 21.]

[ 4. 5. 6. 13. 14. 15. 22. 23. 24.]

[ 7. 8. 9. 16. 17. 18. 25. 26. 27.]]

vstack((A,B))

[[ 1. 2. 3.]

[ 4. 5. 6.]

[ 7. 8. 9.]

15

[ 10. 11. 12.]

[ 13. 14. 15.]

[ 16. 17. 18.]]

vstack((A,B,C))

[[ 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.]]

Nota: tamb´e es pot separar un array en diverses parts amb les funcions vsplit() i hsplit().

1.9 Bucles for

Ja hem vist en la secci´o “4.3 - for” que els ndarray es poden usar en els bucles *for*. Quan s´on d’una dimensi´o s’itera sobre els elements i quan s´on multidimensionals s’itera sobre els elements del primer eix, que s´on ndarrays:

In [27]: import numpy as np

print ("Array 1D")

array\_1D = np.arange(3)

for element in array\_1D:

print (element)

print ("-------")

print ("Array 2D")

array\_2D = np.arange(9).reshape(3,3)

for element in array\_2D:

print (element)

print ("-------")

print ("Array 3D")

array\_3D = np.arange(27).reshape(3,3,3)

for element in array\_3D:

print (element)

print ("-------")

Array 1D

0

-------

1

-------

2

-------

Array 2D

[0 1 2]

-------

[3 4 5]

-------

16

[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]]

-------

Si es vol iterar sobre tots els elements independentment de les dimensions de l’ndarray es pot usar la funci´o flat per a “desplegar” l’ndarray en el bucle for:

In [28]: import numpy as np

array\_2D = np.arange(9).reshape(3,3)

print (array\_2D)

print

for element in array\_2D.flat:

print (element)

[[0 1 2]

[3 4 5]

[6 7 8]]

0

1

2

3

4

5

6

7

8

1.10 Algebra lineal `

Numpy implementa les operacions d’algebra lineal, ´es a dir, les operacions amb vectors i matrius. Les eines d’`algebra lineal es troben en el m`odul numpy.linalg:

from numpy.linalg import \*

1.10.1 Eines d’`algebra lineal

Trasposici´o Es pot fer amb el m`etode transpose() dels ndarrays:

In [29]: import numpy as np

*# Generem un array 2D*

17

A = np.linspace(1.,9.,9).reshape(3,3)

print ("Array A")

print (A)

*# El transposem*

print ("Array A transposat")

print (A.transpose())

Array A

[[ 1. 2. 3.]

[ 4. 5. 6.]

[ 7. 8. 9.]]

Array A transposat

[[ 1. 4. 7.]

[ 2. 5. 8.]

[ 3. 6. 9.]]

1.10.2 Determinants i matriu inversa

Els determinants es calculen amb la funci´o linalg.det() i la matriu inversa amb la funci´o linalg.inv() In [30]: import numpy as np

*# Generem un array 2D*

A = np.array( [[1,1,2], [2,1,1], [1,1,1]] )

print ("Array A")

print (A)

*# Calculem el determinant*

print ("Determinant de A: ", np.linalg.det(A))

*# L’invertim*

print ("Inversa de A")

print (np.linalg.inv(A))

Array A

[[1 1 2]

[2 1 1]

[1 1 1]]

Determinant de A: 1.0

Inversa de A

[[ 0. 1. -1.]

[-1. -1. 3.]

[ 1. -0. -1.]]

1.10.3 Rang i tra¸ca d’una matriu

El rang d’una matriu es pot calcular usant la funci´o numpy.linalg.matrix rank(A) La tra¸ca d’una matriu es calcula amb la funci´o trace():

In [31]: import numpy as np

*# Generem un array 2D*

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

print ("Array A")

print (A)

18

*# Calculem el rang*

print("Rang de A: ", np.linalg.matrix\_rank(A)) *# Calculem la tra¸ca*

print ("Tra¸ca de A: ", np.trace(A))

Array A

[[1 1 2]

[2 4 1]

[1 1 1]]

Rang de A: 3

Tra¸ca de A: 6

1.10.4 Producte de matrius

El producte de matrius s’implementa amb la funci´o dot(): In [32]: import numpy as np

*# Generem un array 2D*

A = np.array( [[1,1,2], [2,1,1], [1,1,1]] ) print ("Array A")

print (A)

*# El multipliquem per ell mateix*

print ("A\*A")

print (np.dot(A,A))

*# Multipliquem la matriu per un vector*

v = array( [3,1,3] )

print ("A\*v")

print (np.dot(A,v))

Array A

[[1 1 2]

[2 1 1]

[1 1 1]]

A\*A

[[5 4 5]

[5 4 6]

[4 3 4]]

A\*v

[10 10 7]

1.10.5 Soluci´o de sistemes d’equacions

Un sistema d’equacions

*a*11*x*1 + *a*12*x*2 + *a*13*x*3 = *b*1

*a*21*x*1 + *a*22*x*2 + *a*23*x*3 = *b*2

*a*31*x*1 + *a*32*x*2 + *a*33*x*3 = *b*3

es pot representar en forma matricial

 

*a*11 *a*12 *a*13 *a*21 *a*22 *a*23 *a*31 *a*32 *a*33

 

 

*x*1 *x*2 *x*3



 =

 

*b*1 *b*2 *b*3





19

o de forma compacta

*A~x* = *~b*

Donada la matriu *A* del sistema i el vector de termes independents *~b* es pot obtenir la soluci´o *~x* amb la funci´o solve():

In [33]: import numpy as np

*# Matriu del sistema*

A = np.array( [[1,1,2], [2,1,1], [1,1,1]] )

print (A)

*# Vector de termes independents*

b = np.array( [1,1,1] )

print ("Vector de termes independents")

print (b)

*# Calculem la soluci´o*

x = np.linalg.solve(A,b)

print ("Soluci´o")

print (x)

[[1 1 2]

[2 1 1]

[1 1 1]]

Vector de termes independents

[1 1 1]

Soluci´o

[ 0. 1. -0.]

1.11 Classe matrix

En les operacions d’`algebra lineal que hem descrit fins ara hem usat objectes de tipus ndarray. Tanmateix, si hem de treballar amb matrius “classiques”, ´es a dir 2D i regulars, podem usar una classe de *numpy* dissenyada espec´ıficament per aquest cas, la classe matrix. Podeu trobar una descripci´o completa de la classe aqu´ı.

Aquesta classe es pot usar com un ndarray normal en les operacions anteriors

In [ ]:

20