Σύντομη επανάληψη εντολών Numpy για πίνακες

Θεωρούμε ότι έχουμε εισάγει τη βιβλιοθήκη Numpy με την εντολή `import numpy as np`

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

Διάσταση array

Τα arrays στη Numpy εκτός από μια διάσταση μπορούν να έχουν 2 διαστάσεις. Π.χ. το `a=np.array([1,2,3,4])` είναι ένα μονοδιάστατο array, ενώ το `a=np.array([[1,2],[3,4]])` είναι διδιάστατο (έχει τη μορφή πίνακα).

Στην αριθμητική λύση γραμμικών συστημάτων, ως παραδοχή για να μην δημιουργηθούν κάποια προβλήματα στα αποτελέσματα, θα θεωρήσουμε εκτός από τους πίνακες και τα διανύσματα διδιάστατα, δηλαδή πίνακες με μία στήλη. Π.χ. το διάνυσμα

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

θα είναι το `x=np.array([[1],[2],[3]])`

Με την εντολή `shape` ελέγχουμε τη διάσταση ενός array της Numpy, και με το size το μέγεθος.

In [2]:
```
x=np.array([1,2,3])
print(x)
print(x.shape)
print(x.size)
print('-----')
x=np.array([[1],[2],[3]])
print(x)
print(x.shape)
print(x.size)
print('-------')
x=np.array([[1,2],[2,3],[3,4]])
print(x)
print(x.shape)
print(x.size)
```

```
[1 2 3]
(3,)
3
-----
[[1]
 [2]
 [3]]
(3, 1)
3
-------
[[1 2]
 [2 3]
 [3 4]]
(3, 2)
6
```

In [3]:
```
# Μονοδιάστατο array
x=np.array([1,2,3,4,5,6])
print(x.shape)
```

```
(6,)
```

In [4]:
```
# Διδιάστατο array
x=np.array([[1],[2],[3],[4],[5],[6]])
print(x.shape)
```

```
(6, 1)
```

Πράξεις με array

Πολλαπλασιασμός και Πρόσθεση Πινάκων

Η πρόσθεση γίνεται με τον συνήθη τρόπο.

In [5]:
```
A=np.array([[1,2],[3,4]])
B=np.array([[1,1],[1,1]])
C=A+B
print('C=',A+B)
```

```
C= [[2 3]
 [4 5]]
```

Ο πολλαπλασιασμός γίνεται με τη συνάρτηση `dot` και όχι με το `*`

In [6]:
```
# Ένας τρόπος
C1=np.dot(A,B)
print('C1=',C1)
print('--------')
# Δεύτερος τρόπος
C2=A.dot(B)
print('C2=',C2)
print('--------')
#Προσοχή!!! Δεν γίνεται ο πολλαπλασιασμός πινάκων με το *
C3=A*B
print('C3=',C3)
```

```
C1= [[3 3]
 [7 7]]
--------
C2= [[3 3]
 [7 7]]
--------
C3= [[1 2]
 [3 4]]
```

Δεν πρέπει να μπερδεύεται η έννοια του διανύσματος στον $\mathbb{R}^n$ με αυτή του `array` στη Numpy. Τη δομή `array` μπορούμε να τη χρησιμοποιήσουμε για να αναπαραστήσουμε τα διανύσματα του $\mathbb{R}^n$ και τους πίνακες $\mathbb{R}^{(n\times n)}$. Στη γραμμική άλγεβρα για να μπορέσουν να γίνουν διάφορες πράξεις πρέπει τα δούμε τα διανύσματα $\mathbb{R}^n$ ως πίνακες $\mathbb{R}^{(n\times 1)}$.

Μια βασική ιδιότητα των πινάκων στη γραμμική άλγεβρα είναι η έννοια του ανάστροφου A^T.

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix}, \quad A^T=\begin{bmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{bmatrix}$$

Η αντίστοιχη εντολή στη Numpy είναι η `transpose`.

In [7]:
```
A=np.array([[1,2],[3,4]])
# Ενας τρόπος
B=A.transpose()
print(B)
print('--------')

# Ενας άλλος τρόπος
C=A.T
print(C)
```

```
[[1 3]
 [2 4]]
--------
[[1 3]
 [2 4]]
```

Στα διανύσματα στη γραμμική άλγεβρα θα ισχύει το ανάλογο για τον ανάστροφο. Έτσι

$$x=\begin{bmatrix}1\\ 2\\ 3\end{bmatrix}, \quad x^T=[1,2,3]$$

Στην Numpy αν δεν ορίσουμε τα διανύσματα ως διδιάστατα arrays δεν έχουμε αυτή την ιδιότητα που ισχύει στα διανύσματα της γραμμικής άλγεβρας.

In [8]:
```
# x μονοδιάστατο array
x=np.array([1,2,3])
print(x)
print('----')
y=x.T
print(y)
print('----')

# x διδιάστατο array
x=np.array([[1],[2],[3]])
print(x)
print('----')
y=x.T
print(y)
```

```
[1 2 3]
----
[1 2 3]
----
[[1]
 [2]
 [3]]
----
[[1 2 3]]
```

Μια βασική παρατήρηση είναι οτι η πράξη Ax, ενός πίνακα $A \in \mathbb{R}^{(n \times n)}$ με ένα διάνυσμα σε μορφή πίνακα στήλη, $x \in \mathbb{R}^{(n \times 1)}$, δίνει ένα διάνυσμα του $\mathbb{R}^{(n \times 1)}$.

Το ανάλογο συμβαίνει και στη Numpy. Αν x είναι μονοδιάστατο array τότε `np.dot(A,x)` είναι μονοδιάστατο array. Ενώ αν x είναι διδιάστατο array τότε `np.dot(A,x)` είναι διδιάστατο array.

In [9]:
```
A=np.array([[1,2],[3,4]])
x=np.array([1,2])

c=np.dot(A,x)
print(c)
print('-------')

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

c1=np.dot(A,x)
print(c1)
```

```
[ 5 11]
-------
[[ 5]
 [11]]
```

Προσοχή

Όταν δημιουργούμε ένα array της Numpy, αυτό δέχεται μόνο ενός είδους ορίσματα, π.χ. είτε integer, float, boolean κ.α. Έτσι αν `x=np.array([1,2,3])` ,τότε

```
x[1]=9
```

αλλάζει το x, σε

```
[1 9 3]
```

Όμως αν δώσουμε

```
x[1]=9.5
```

τότε το x, γίνεται

```
[1 9 3]
```

Για να μπορέσουμε να κάνουμε το x[1] float. Πρέπει το x να δέχεται στοιχεία float. Όταν το δημιουργούμε να είναι κάποιο τουλαχιστον στοιχείο float.

```
x=np.array([1.,2,3])
x[1]=9.5
```

τότε το x, γίνεται

```
[1.  9.5 3. ]
```

Επίλυση γραμμικών συστημάτων με την Numpy

Η βιβλιοθήκη Numpy έχει έτοιμες συναρτήσεις για την επίλυση ενός γραμμικού συστήματος της μορφής $Ax = b$.

Ας υποθέσουμε ότι A και b είναι ο πίνακας και το διάνυσμα που δίνονται από

$$A = \begin{pmatrix} 3 & 1 \\ 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 9 \\ 8 \end{pmatrix}$$

Για την αριθμητική επίλυση ενός γραμμικού συστήματος $Ax = b$, με τη βιβλιοθήκη Numpy μπορούμε να χρησιμοποιήσουμε τη μέθοδο `solve` της βιλιοθήκης numpy.linalg

In [10]:
```
import numpy as np
A=np.array([[3,1],[1,2]])
b=np.array([9,8])
# Επίλυση
x=np.linalg.solve(A,b)
print(x)
# Επαλήθευση
print(b-np.dot(A,x))
```

```
[2. 3.]
[0. 0.]
```

Ένα άλλο παράδειγμα, όπου δημιουργουνται "μικρά" σφάλματα

In [11]:
```
A=np.array([[10,-7,0],[-3.,2.,6],[5.,-1.,5]])
b=np.array([1.,2,3])
x=np.linalg.solve(A,b)
print('x=',x)
#Επαλήθευση αποτελέσματος
print('Ax-b=',np.dot(A,x)-b)
```

```
x= [0.25806452 0.22580645 0.38709677]
Ax-b= [ 2.22044605e-16 -4.44089210e-16  0.00000000e+00]
```

Νόρμες διανυσμάτων και πινάκων

Με την εντολή `norm` της `numpy.linalg` μπορούμε να υπολογίσουμε την ευκλείδεια νόρμα $\|\cdot\|_2$ ενός διανύσματος του $\mathbb{R}^n$ (π.χ. ενός μονοδιάστατου `array` της Numpy). Παρατηρήστε ότι υπολογίζει σωστά το αποτέλεσμα είτε θεωρήσουμε ένα μονοδιάστατο στοιχείο `array` της `numpy` , είτε μια `list` . Επίσης το αποτέλεσμα είναι σωστό ακόμα και σε 2-διάστατα `array` της Numpy, αλλά με μια στήλη.

In [12]:
```
import numpy as np
x=[1,1,1,1]  #λίστα
print('x=',x)
c=np.linalg.norm(x)
print('Ευκλείδεια νόρμα x=',c)
x=np.array([1,1,1,1]) #μονοδιάστατο array της numpy
print('x=',x)
c=np.linalg.norm(x)
print('Ευκλείδεια νόρμα x=',c)
x=np.array([[1],[1],[1],[1]]) # πίνακας στήλη της numpy
print('x=',x)
c=np.linalg.norm(x)
print('Ευκλείδεια νόρμα x=',c)
```

```
x= [1, 1, 1, 1]
Ευκλείδεια νόρμα x= 2.0
x= [1 1 1 1]
Ευκλείδεια νόρμα x= 2.0
x= [[1]
 [1]
 [1]
 [1]]
Ευκλείδεια νόρμα x= 2.0
```

Αν θελήσουμε να υπολογίσουμε την 1-νορμα, $\|\cdot\|_1$ ή τη νόρμα μεγίστου $\|\cdot\|_\infty$, η εντολή norm παίρνει την αντίστοιχη παράμετρο.

In [13]:
```
x=np.array([1,2,3,4]) #μονοδιάστατο array της numpy
print('x=',x)
c=np.linalg.norm(x,np.inf)
print('Νόρμα μεγίστου x=',c)
c=np.linalg.norm(x,1)
print('Νόρμα-1 x=',c)
c=np.linalg.norm(x,2)
print('Νόρμα-2 ή Ευκλείδεια νόρμα x=',c)
```

```
x= [1 2 3 4]
Νόρμα μεγίστου x= 4.0
Νόρμα-1 x= 10.0
Νόρμα-2 ή Ευκλείδεια νόρμα x= 5.477225575051661
```

Ανάλογα ισχύουν και για πίνακες, όπου ορίζουμε την αντίστοιχη φυσική νόρμα πινάκων.

In [14]:
```
A=np.array([[1,2,3],[4,5,6],[9,8,7]])
print('A=',A)
#Νορμα μεγίστου είναι το μέγιστο άθροισμα των απολυτών τιμών κάθε γραμμής
print('Νόρμα μεγίστου του A:',np.linalg.norm(A,np.inf))
#Νορμα 1 είναι το μέγιστο άθροισμα των απολυτών τιμών κάθε στήλης
print('Νόρμα 1 του A:',np.linalg.norm(A,1))
#Νορμα 2 είναι η τετραγωνική ρίζα της μέγιστης κατά απόλυτο τιμή, ιδιοτιμή
print('Νόρμα 2 του A:',np.linalg.norm(A,2))
```

```
A= [[1 2 3]
 [4 5 6]
 [9 8 7]]
Νόρμα μεγίστου του A: 24.0
Νόρμα 1 του A: 16.0
Νόρμα 2 του A: 16.703950917459903
```

Προσοχή

Σε αντίθεση με τα 1-διάστατα `array`, η εντολή `norm` για 2-διάστατα `array`, (δηλ. πίνακες) χωρίς την παράμετρο 2, δεν υπολογίζει τη νόρμα-2, αλλά μια άλλη νόρμα η οποία ονομάζετε νόρμα Frobenious και ορίζεται ώς η τετραγωνική ρίζα του αρθροίσματος των τετραγώνων όλων των στοιχείων ενός πίνακα.

$$\|\mathrm{A}\|_{Fro} = \sqrt{\sum_{i,j=1}^{n} a_{ij}^2}$$

In [15]:
```
c=np.linalg.norm(A,2)
print('Νόρμα 2 του A:',c)
c=np.linalg.norm(A)
print('Νόρμα Frobenious του A:',c)
```

```
Νόρμα 2 του A: 16.703950917459903
Νόρμα Frobenious του A: 16.881943016134134
```

Δείκτης κατάστασης ενός πίνακα

Για ένα αντιστρέψιμο πίνακα A, ορίζουμε ως δείκτη κατάστασης $\kappa(A)$, ως προς μια νόρμα πινάκων $\|\cdot\|$,

$$\kappa(A) = \|A\| \, \|A^{-1}\|$$

Είναι προφανές ότι για να υπολογίσουμε το $\kappa(A)$ χρειάζεται να γνωρίζουμε τον A^{-1}. Συνήθως ο A^{-1} είναι δύσκολο να βρεθεί με το "χέρι", ιδιαίτερα σε πίνακες με μεγάλη διάσταση. Στην numpy υπάρχει η εντολή `inv` για την εύρεση του αντιστρόφου, η οποία όμως λόγω σφαλμάτων πράξεων δίνει μια προσέγγιση του A^{-1}.

In [16]:
```
A=np.array([[3.,1.],[1.,2.]])
B=np.linalg.inv(A)
print('A=',A)
print('inv(A)=B=',B)
print('Για να επαληθεύσουμε ότι AB=BA=I  και ότι inv(B)=A')
print('AB=',np.dot(A,B)) # Επαναλήθευση οτι B είναι ο αντίστροφος
print('BA=',np.dot(B,A)) # Επαναλήθευση οτι B είναι ο αντίστροφος
print('inv(B)=',np.linalg.inv(B)) # Επαναλήθευση οτι ο αντίστροφος του B ε
```

```
A= [[3. 1.]
 [1. 2.]]
inv(A)=B= [[ 0.4 -0.2]
 [-0.2  0.6]]
Για να επαληθεύσουμε ότι AB=BA=I  και ότι inv(B)=A
AB= [[1. 0.]
 [0. 1.]]
BA= [[1. 0.]
 [0. 1.]]
inv(B)= [[3. 1.]
 [1. 2.]]
```

Για τον υπολογισμό του δείκτη κατάστασης μπορούμε να χρησιμοποιήσουμε και την εντολή `cond` .

In [17]:
```
s=np.linalg.norm(A,1)*np.linalg.norm(B,1) # Χρήση του ορισμου με το γινόμε
print(s)
s=np.linalg.cond(A,1) # Χρήση της εντολής cond
print(s)
```

```
3.1999999999999997
3.1999999999999997
```

Αν ο δείκτης κατάστασης είναι μεγάλος

Αν ο πίνακας είναι "κοντά" σε έναν μη αντιστρέψιμο, δηλαδή ο δείκτης κατάστασης $\kappa(A)$ είναι μεγάλος, τότε με την εντολή `inv` δημιουργούνται σφάλματα. Λόγω αυτών των σφαλμάτων

$$A^{-1}A \neq AA^{-1}$$

και

$$(A^{-1})^{-1} \neq A$$

Ας θεωρήσουμε π.χ, τον πίνακα

$$A = \begin{pmatrix} 100 & 100 \\ 100 & 100.01 \end{pmatrix}$$

ο οποίος είναι "κοντά" στο να μην είναι αντιστρέψιμος. Θα ελέγξουμε στη συνέχεια αν η εντολή `inv` υπολογίζει τον αντίστροφο του A. (Πολλές φορές αυτό το σφάλμα δεν είναι ευδιάκριτο και χρειάζεται να τυπώσουμε αρκετά ψηφιά.)

In [18]:
```
A=np.array([[100,100],[100,100.01]]) #πίνακας A
print('κ(A)=',np.linalg.cond(A,1)) # δείκτης κατάστασης είναι μεγάλος
B=np.linalg.inv(A)  # υπολογισμός αντιστρόφου
print('Αντίστροφος του A=',B)
print('Επαλήθευση--------')
print('AB=',np.dot(A,B)) # Επαναλήθευση οτι B είναι ο αντίστροφος
#np.set_printoptions(precision=15) #θέτουμε 15 δεκαδικά ψηφια
#print(np.dot(A,B))    # Επαναλήθευση αν B είναι ο αντίστροφος  (AB=I)
print('BA=',np.dot(B,A))    # Επαναλήθευση αν B είναι ο αντίστροφος  (BA=I
print('Αντίστροφος του B=',np.linalg.inv(B)) # Επαναλήθευση αν ο αντίστροφ
```

```
κ(A)= 40004.00009997953
Αντίστροφος του A= [[ 100.01 -100.  ]
 [-100.    100.  ]]
Επαλήθευση--------
AB= [[1.0000000e+00 0.0000000e+00]
 [1.8189894e-12 1.0000000e+00]]
BA= [[1. 0.]
 [0. 1.]]
Αντίστροφος του B= [[100.   100.  ]
 [100.   100.01]]
```

Η εικόνα για το πόσο διαφέρουν δυο πίνακες ή διανύσματα γίνεται καλύτερη αν χρησιμοποιήσουμε μια νόρμα.

In [19]:
```
c=np.linalg.norm(np.dot(A,B)-np.eye(2),np.inf)
print('Η νόρμα-άπειρο του AB-I:',c)
c=np.linalg.norm(np.dot(B,A)-np.eye(2),np.inf)
print('Η νόρμα-άπειρο του BA-I:',c)
```

```
Η νόρμα-άπειρο του AB-I: 3.637978807091713e-12
Η νόρμα-άπειρο του BA-I: 1.8189894035458565e-12
```