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

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

Η λύση x του γραμμικού συστήματος $Ax = b$, δίνεται από τη συνάρτηση solve που βρίσκεται στη βιβλιοθήκη linalg της numpy

In [1]:
```
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(np.dot(A,x))
```

```
[[ 2.]
 [ 3.]]
[[ 9.]
 [ 8.]]
```

Προβλήματα με κακή κατάσταση

Όλα τα γραμμικά συστήματα δεν λύνονται "εύκολα". Το γραμμικό σύστημα με

$$A = \begin{pmatrix} 1/3 & 1/3 \\ 1/3 & .3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

έχει λύση

$$x = \begin{pmatrix} -27 \\ 30 \end{pmatrix}$$

In [2]:
```
import numpy as np
A=np.array([[1./3,1./3],[1./3,.33]])
b=np.array([[1],[0]])
# Επίλυση
x=np.linalg.solve(A,b)
print(x)
# Επαλήθευση
print(np.dot(A,x))
```

```
[[-297.]
 [ 300.]]
[[ 1.]
 [ 0.]]
```

Αν όμως τροποιήσουμε τον πίνακα A, μεταβάλλοντας το στοιχείο στη θέση (2,2)

$$A = \begin{pmatrix} 1/3 & 1/3 \\ 1/3 & 1/3 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

το οποίο δεν λύνεται.

Άσκηση 1:

Μεταβάλετε το στοιχείο (2,2) του πίνακα A, προσθέτοντας δεκαδικά ψηφία με το 3. Έτσι ξεκινήστε με το $.33$ και λύστε το γραμμικό σύστημα. Συνεχίστε, προσθέτοντας "3-άρια" ως δεκαδικά ψηφία ώστε να πλησιάσετε το $1/3$. Σε κάποιο σημείο η λύση σας δεν θα είναι υπολογίσιμη.

Δείκτης κατάστασης

Νόρμες διανύσματος - πίνακα

Μπορούμε να μετρήσουμε πόσο "εύκολα" μπορούμε να λύσουμε ένα γραμμικό σύστημα χρησιμοποιώντας την έννοια της νόρμας πίνακα. Με την εντολή norm μπορούμε να υπολογίσουμε τη νόρμα ενός array.

In [3]:
```
print(x) # διάνυσμα x
s=np.linalg.norm(x,1) # νόρμα 1
print(s)
s=np.linalg.norm(x,np.inf) # νόρμα απείρου
print(s)
s=np.linalg.norm(x,2) # νόρμα 2
print(s)
```

```
[[-297.]
 [ 300.]]
597.0
300.0
422.148078285
```

Για 2-διάστατα array δηλαδή πίνακες ορίζεται ανάλογα.

In [4]:
```
print(A) # πίνακας A
s=np.linalg.norm(A,1) # νόρμα 1
print(s)
s=np.linalg.norm(A,np.inf) # νόρμα απείρου
print(s)
s=np.linalg.norm(A,2) # νόρμα 2
print(s)
```

```
[[ 0.33333333  0.33333333]
 [ 0.33333333  0.33      ]]
0.666666666667
0.666666666667
0.665004166641
```

Μιά χρήσιμη εντολή για να ¨τυπώνουμε" έναν επιθυμητό αριθμό δεκαδικών ψηφίων ενός array είναι η εντολή set_printoptions

In [5]:
```
np.set_printoptions(precision=20) # τυπώνουμε εως 8 δεκαδικά ψηφία
print(A) # πίνακας A
s=np.linalg.norm(A,1) # νόρμα 1
print(s)
s=np.linalg.norm(A,np.inf) # νόρμα απείρου
print(s)
s=np.linalg.norm(A,2) # νόρμα 2
print(s)
```

```
[[ 0.33333333333333331483  0.33333333333333331483]
 [ 0.33333333333333331483  0.33000000000000001554]]
0.666666666667
0.666666666667
0.665004166641
```

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 [6]:
```
A=np.array([[3,1],[1,2]])
B=np.linalg.inv(A)
print(B)
print(np.dot(A,B)) # Επαναλήθευση οτι Β είναι ο αντίστροφος
print(np.dot(B,A)) # Επαναλήθευση οτι Β είναι ο αντίστροφος
print(np.linalg.inv(B))
```

```
[[ 0.39999999999999996669 -0.19999999999999998335]
 [-0.19999999999999998335  0.59999999999999999778 ]]
[[ 1.  0.]
 [ 0.  1.]]
[[ 1.  0.]
 [ 0.  1.]]
[[ 3.  1.]
 [ 1.  2.]]
```

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

In [7]:
```
s=np.linalg.norm(A,1)*np.linalg.norm(B,1) # γινόμενο των νορμών
print(s)
print(np.linalg.cond(A,1)) # εντολή cond
```

```
3.2
3.2
```

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

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

Θεωρούμε π.χ, τον πίνακα

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

In [8]:
```
np.set_printoptions(precision=5) #θέτουμε 5 δεκαδικά ψηφια για την εκτύπωση των στοιχείων ενός πίνακα
A=np.array([[100,100],[100,100.0001]]) #πίνακας A
print(np.linalg.cond(A,1)) # δείκτης κατάστασης είναι μεγάλος
B=np.linalg.inv(A)  # υπολογισμός αντιστρόφου
print(B)
print(np.dot(A,B)) # Επαναλήθευση οτι B είναι ο αντίστροφος
np.set_printoptions(precision=16) #θέτουμε 15 δεκαδικά ψηφια
print(np.dot(A,B))    # Επαναλήθευση αν B είναι ο αντίστροφος  (AB=I)
print(np.dot(B,A))    # Επαναλήθευση αν B είναι ο αντίστροφος  (BA=I)
print(np.linalg.inv(B)) # Επαναλήθευση αν ο αντίστροφος του B είναι ο A
```

```
4000003.99987
[[ 10000.01 -10000.  ]
 [-10000.    10000.  ]]
[[ 1.  0.]
 [ 0.  1.]]
[[ 1.  0.]
 [ 0.  1.]]
[[ 1.  0.]
 [ 0.  1.]]
[[ 100.0000000059546181  100.0000000059546323]
 [ 100.0000000059546181  100.0001000059546357]]
```

Άσκηση 2:

Δοκιμάστε να τροποποιήστε το στοιχείο στη θέση (2,2), ώστε ο πίνακας A να είναι πιο "κοντά" ή πιο μακρυά σε έναν μη αντιστρέψιμο και βρείτε το δείκτη κατάστασης, τον αντίστροφο και παρατηρήστε αν όντως είναι ο αντίστροφος.

Ένας άλλος τρόπος να ελέγχουμε αν ένας πίνακας είναι ο επιθυμητός, είναι να ελέγχουμε το σφάλμα που έχει ο υπολογισμένος πίνακας από τον επιθυμητό και στη συνέχεια να υπολογίσουμε μια νόρμα αυτού του σφάλματος. Στο προηγούμενο παράδειγμα ελέγχουμε τη νόρμα της διαφοράς $||AA^{-1} - I||$.

In [9]:
```
A=np.array([[100,100],[100,100.001]]) #πίνακας A
print(np.linalg.cond(A,1)) # δείκτης κατάστασης είναι μεγάλος
B=np.linalg.inv(A)  # υπολογισμός αντιστρόφου
print(B)
C=np.dot(A,B)-np.eye(2) # AB-I
print(C)
print(np.linalg.norm(C,1)) # Επαναλήθευση οτι B είναι ο αντίστροφος
C=np.dot(B,A)-np.eye(2)   # BA-I
print(C)
print(np.linalg.norm(C,1))    # Επαναλήθευση αν B είναι ο αντίστροφο
ς

C=np.linalg.inv(B)-A # B-A
print(C)
print(np.linalg.norm(C,1)) # Επαναλήθευση αν ο αντίστροφος του B είν
αι ο A
```

```
400004.000008
[[ 1000.0099999952252574  -999.9999999952252665]
 [ -999.9999999952251528   999.9999999952251528]]
[[  0.0000000000000000e+00   0.0000000000000000e+00]
 [  1.4551915228366852e-11  -1.4551915228366852e-11]]
1.45519152284e-11
[[  0.0000000000000000e+00   0.0000000000000000e+00]
 [  0.0000000000000000e+00  -1.4551915228366852e-11]]
1.45519152284e-11
[[ -4.8457593493367312e-10  -4.8456172407895792e-10]
 [ -4.8457593493367312e-10  -4.8456172407895792e-10]]
9.69151869867e-10
```

Άσκηση 3:

Φτιάξτε μια συνάρτηση my_cond η οποία να υπολογίζει τον δείκτη κάταστασης ενός πίνακα A. (χρησιμοποιήστε όποια νόρμα θέλετε)

```
def my_cond(A):
    ........ # συμπληρώστε με τις κατάλληλες εντολές
    return s # επιστρέφει την τιμή s που είναι ο δείκτης κατάστασης
```

Γράψτε ένα πρόγραμμα που να κάνει χρήση της συνάρτησης που φτιάξατε και συγκρίνετε τα αποτελέσματα σας με την εντολή cond της numpy

Επίλυση γραμμικών συστημάτων

Η επίλυση γραμμικών συστημάτων με πίνακες με μεγάλο δείκτη κατάστασης οδηγεί σε σφάλματα.
Θεωρούμε το γραμμικό σύστημα $Ax = b$, με

$$A = \begin{pmatrix} 100 & 100 \\ 100 & 100.01 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

όπου η ακριβή λύση είναι

$$x = \begin{pmatrix} -99.99 \\ 100 \end{pmatrix}$$

In [10]:
```
np.set_printoptions(precision=5) #ορίζουμε ακρίβεια εκτύπωσης
A=np.array([[100,100],[100,100.01]])
print(np.linalg.cond(A,1))
b=np.array([[1],[2]])
x=np.linalg.solve(A,b)
print(x) # τυπώνουμε τη λύση
np.set_printoptions(precision=10) #ορίζουμε ακρίβεια εκτύπωσης
print(x) # τυπώνουμε τη λύση με μεγαλύτερη ακρίβεια
```

```
40004.0001
[[-99.99]
 [100.  ]]
[[-99.9899999999]
 [ 99.9999999999]]
```

Συγκρίνουμε τη λύση που βρήκαμε με την ακριβή, ή υπολογίζουμε τη νόρμα της διαφοράς

In [11]:
```
np.set_printoptions(precision=5) #ορίζουμε ακρίβεια εκτύπωσης
y=np.array([[-99.99],[100]])
print(x-y)
print(np.linalg.norm(x-y,1))
```

```
[[  5.11449e-11]
 [ -5.11591e-11]]
1.02303943095e-10
```

Ένας άλλος τρόπος αν δεν γνωρίζουμε την ακριβή λύση είναι να υπολογίσουμε το υπόλοιπο $r = Ax - b$.
Αν έχουμε λύσει το πρόβλημα ακριβώς το υπόλοιπο r θα είναι 0.

In [12]:
```
r=np.dot(A,x)-b # υπόλοιπο Ax-b
print(r)
print(np.linalg.norm(r,1))
```

```
[[  0.00000e+00]
 [ -1.81899e-12]]
1.81898940355e-12
```

Τα σφάλματα στις πράξεις μπορούν να υπάρξουν ακόμα και σε πίνακες με μικρό δείκτη κατάστασης.

In [13]:
```
np.set_printoptions(precision=5)
A=np.array([[1,3,5],[2,5,1],[2,3,8]])
print(np.linalg.cond(A,1))
b=np.array([[10],[8],[3]])
x=np.linalg.solve(A,b)
print(x)
r=np.dot(A,x)-b # υπόλοιπο Ax-b
print(r)
print(np.linalg.norm(r,1))
```

```
30.8
[[-9.28]
 [ 5.16]
 [ 0.76]]
[[  0.00000e+00]
 [  0.00000e+00]
 [ -1.77636e-15]]
1.7763568394e-15
```

Άσκηση 4:

Φτιάξτε μια συνάρτηση my_residual η οποία να υπολογίζει τη νόρμα του υπολοίπου r της λύσης ενός γραμμικού συστήματος με πίνακα A και δεξιό μέλος b. (χρησιμοποιήστε όποια νόρμα θέλετε)

```
def my_residual(A,b):
    ........ # συμπληρώστε με τις κατάλληλες εντολές
    return s # επιστρέφει την τιμή s που είναι η νόρμα του υπολοίπου r
```

Γράψτε ένα πρόγραμμα που να κάνει χρήση της συνάρτησης που φτιάξατε

Πίνακες Hilbert

Οι πίνακες Hilbert είναι τετραγωνικοί πίνακες με στοιχεία $a_{ij} = \frac{1}{i+j-1}$,

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \cdots & \frac{1}{n} \\ \frac{1}{2} & \frac{1}{3} & \cdots & \frac{1}{n+1} \\ & & \ddots & \\ \frac{1}{n} & & & \frac{1}{2n-1} \end{pmatrix}$$

Εκτός από τη βιβλιοθήκη Numpy υπάρχει στην python και η Scipy. Περιέχει ορισμένες χρήσιμες εντολές όπως η δημιουργία ενός πίνακα Hilbert.

In [14]:
```
import scipy.linalg as sp # Η βιβλιοθήκη linalg της Scipy
k=20 #διάσταση πίνακα
A=sp.hilbert(k)
np.set_printoptions(precision=5)
#print(A)
b=np.ones((k,1))
x=np.linalg.solve(A,b)
#print(np.dot(A,x))
r=np.dot(A,x)-b
print(np.linalg.norm(r))
print(np.linalg.cond(A))
```

```
9.99600281194e-08
2.1211455691e+18
```

Άσκηση 5:

Δοκιμάστε να αλλάξτε τη διάσταση του πίνακα Hilbert και παρατηρήστε την αύξηση του δείκτη κατάστασης.

Άσκηση 6:

Υπολογίστε το δείκτη κατάστασης ενός πίνακα Hilbert για διάσταση $k = 5, 10, 20, 30$ σε διάφορες νόρμες και συγκρίνετε το αποτέλεσμα σας με το αποτέλεσμα που δίνει η δική σας ρουτίνα my_cond.

Υπολογισμός αντιστρόφου

Για το υπολογισμό του αντιστρόφου A^{-1} ενός αντιστρέψιμου πίνακα A διαστασης n, μπορούμε να ακολουθήσουμε την εξής διαδικασία. Να θεωρήσουμε το διάνυσμα e_j το οποίο έχει μονάδα μόνο στη θέση j και μηδέν αλλού, και να λύσουμε το γραμμικό σύστημα $Ax_j = e_j$. Η λύση x_j είναι η προσέγγιση της j-στήλης του πίνακα A^{-1}. Αν το επαναλάβουμε αυτό για κάθε $j = 1, \dots, n$, θα δημιουργήσουμε τον πίνακα B που θα προσεγγίζει τον A^{-1}.

Άσκηση 7:

Δημιουργήστε μια συνάρτηση η οποία να κατασκευάζει τον αντίστροφο ενός πίνακα A, και φτιάξτε ένα πρόγραμμα ωστε να ελέγξτε το αποτέλεσμα σας

```
def my_inv(A):
     # συμπληρώστε τις κατάλληλες εντολές.
    return B
```