Υπολογισμοί με Python

Εφαρμοσμένη Άλγεβρα

2023-2024 (ver. 2024.03.1501)

Βιβλιοθήκες Python για μήτρες και διανύσματα

Υπάρχουν πολλές βιβλιοθήκες για την αναπαράσταση και χειρισμό μητρών και διανυσμάτων.
Παρακάτω παρουσιάζονται, μέσα από παραδείγματα, ορισμένες μέθοδοι των βιβλιοθηκών **SymPy** (**Symbolic Python**) και **NumPy** (**Numerical Python**) που αφορούν μήτρες και διανύσματα.

Η βιβλιοθήκη Sympy

Για χρησιμοποιήσουμε την βιβλιοθήκη `sympy` δίνουμε την εντολή

```
import sympy as sp
```

οπότε όλες οι διαθέσιμες μέθοδοι της `sympy` μπορούν να προσπελαστούν γράφοντας

```
sp.Onoma_Methodou(Orismata)
```

ή, εναλλακτικά μπορούμε να προσπελάσουμε απευθείας (χωρίς το πρόθεμα `sp.`) όλες τις μεθόδους δίνοντας την εντολή

```
from sympy import *
```

Συμβολικοί υπολογισμοί

Η διαφορά της sympy από άλλες βιβλιοθήκες υπολογισμών είναι ότι χειρίζεται τους αριθμούς ως σύμβολα. Έτσι, οι αριθμητικές μέθοδοι της sympy επιστρέφουν αλγεβρικές/συμβολικές παραστάσεις.

```
import sympy as sp
x = sp.sqrt(5**2 + 7)
print(x) #sqrt 32
```

Output:

```
4*sqrt(2)
```

Στην περίπτωση που χρειαζόμαστε την αριθμητική τιμή μιας παράστασης μπορούμε να χρησιμοποιήσουμε την μέθοδο `evalf()` ή την συνάρτηση `N`

```
import sympy as sp
x = sp.sqrt(5**2 + 7)
print(x.evalf(15))
print(sp.N(x,20))
```

Output:

```
5.65685424949238
5.6568542494923801952
```

Συμβολικοί υπολογισμοί

Επίσης, ένα άλλο σημαντικό πλεονέκτημα τις βιβλιοθήκης `sympy` είναι ότι μπορεί να χειρίζεται **σύμβολα** που αναπαριστούν μεταβλητές. Μπορεί για παράδειγμα καταλάβει ότι η έκφραση $2x + 5x - 3x + 1$ είναι ισοδύναμη με την έκφραση $4x + 1$ ανεξάρτητα από την τιμή της μεταβλητής x.
Προκειμένου να ορίσουμε σύμβολα χρησιμοποιούμε την κλάση `symbols`

```
1 import sympy as sp
2 x = sp.symbols('x')
3 print(2*x + 5*x - 3*x + 1)
```

Output:

```
1 4*x + 1
```

Στην περίπτωση αυτή η μεταβλητή `x` αναπαριστά το σύμβολο x.

Συμβολικοί υπολογισμοί

Μπορούμε να ορίσουμε περισσότερα σύμβολα ταυτόχρονα:

```
S = [x,y,z,a] = sp.symbols('x y z a')
print(S)
```

Output:

```
(x, y, z, a)
```

Επίσης, υπάρχει η δυνατότητα να ορίσουμε πολλά σύμβολα χρησιμοποιώντας την συντομογραφία :

```
S = sp.symbols('x:10')
print(S)
```

Output:

```
(x0, x1, x2, x3, x4, x5, x6, x7, x8, x9)
```

Έτσι η μεταβλητή $x4$ μπορεί να προσπελαστεί ως το στοιχείο $S[4]$ της λίστας S.

Συμβολικοί υπολογισμοί

Η βιβλιοθήκη sympy χειριζεται αλγεβρικά όλα σύμβολα που έχουμε ορίσει.

```
x, y = sp.symbols('x y')
f = x**2 - y**2
g = x - y
print(sp.simplify(f/g))
```

Output:

```
x + y
```

Προσοχή! Η βιβλιοθήκη sympy υποθέτει (πάντα) ότι οι συμβολικές εκφράσεις αναπαριστούν αριθμούς διαφορετικούς από το 0. (Εδώ στο παράδειγμα η βιβλιοθήκη υποθέτει ότι $x - y \neq 0$.)

Συμβολικοί υπολογισμοί

Ένα δεύτερο παράδειγμα:

```
x,y,z = sp.symbols('x y z')
D = -x**2*y + x**2*z + x*y**2 - x*z**2 - y**2*z + y*z**2
print("D:",D)
print("D:",sp.factor(D))
```

Output:

```
D: -x**2*y + x**2*z + x*y**2 - x*z**2 - y**2*z + y*z**2
D: -(x - y)*(x - z)*(y - z)
```

Συμβολικοί υπολογισμοί

Ένα τρίτο παράδειγμα:

```
x = sp.symbols('x')
f = ((1 + 2*x)**10 - 1)*((1 + x)**15 - 1)
print(sp.expand(f))
```

Output:

```
1024*x**25 + 20480*x**24 + 195840*x**23 + 1191680*x**22 + 5180800*x
    **21 + 17127936*x**20 + 44749600*x**19 + 94789280*x**18 +
    165702420*x**17 + 242080320*x**16 + 298199264*x**15 +
    311603520*x**14 + 277258800*x**13 + 210441280*x**12 +
    136241760*x**11 + 75089792*x**10 + 35090040*x**9 + 13809600*x
    **8 + 4528160*x**7 + 1216320*x**6 + 260400*x**5 + 42400*x**4 +
    4800*x**3 + 300*x**2
```

Μήτρες

Για την sympy μια μήτρα είναι μια λίστα από γραμμές. Για να ορίσουμε την μήτρα $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ γράφουμε

```
A = sp.Matrix([[1,2,3],[4,5,6]])
```

Μπορούμε να εκτυπώσουμε στοιχισμένα τα στοιχεία της μήτρας A χρησιμοποιώντας την μέθοδο sp.StrPrinter()

Παράδειγμα

```
import sympy as sp

A = sp.Matrix([[1,2,3],[4,5,6]])
print("A:",A) #default printing
print(A.table(sp.StrPrinter())) #pretty printing
```

Output:

```
A: Matrix([[1, 2, 3], [4, 5, 6]])
[1, 2, 3]
[4, 5, 6]
```

Μήτρες

Για να αλλάξουμε ή να προσπελάσουμε το στοιχείο της A που βρίσκεται στην γραμμή i και στήλη j γράφουμε

```
x = A[i,j]
```

Όπως συνήθως οι γραμμές και οι στήλες έχουν δείκτες που αρχίζουν από 0. Έτσι, η εντολή

```
print(A[1,2])
```

για την μήτρα $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ θα εκτυπώσει

```
6
```

Οι διαστάσεις της A περιέχονται στην μεταβλητή `A.shape`. Ειδικότερα ο αριθμός των γραμμών και των στηλών είναι αποθηκευμένος στις μεταβλητές `A.rows` και `A.cols` αντίστοιχα.

```
mrows, ncols = A.shape
m = A.rows
n = A.cols
```

Μήτρες

Μπορούμε να κατασκευάσουμε μια (άδεια) μήτρα με διαστάσεις $m \times n$ χρησιμοποιώντας τους constructors `zeros(m,n)` ή `ones(m,n)` που κατασκευάζουν μια μήτρα διαστάσεων $m \times n$ που έχει παντού μηδέν και παντού άσσους αντίστοιχα.

```
B = sp.zeros(2,3)
C = sp.ones(2,3)
```

Παράδειγμα

```
import sympy as sp
B = sp.zeros(2,3)
C = sp.ones(2,4)
print(B.table(sp.StrPrinter()))
print(C.table(sp.StrPrinter()))
```

Output:

```
[0, 0, 0]
[0, 0, 0]
[1, 1, 1, 1]
[1, 1, 1, 1]
```

Μήτρες

Μπορούμε να κατασκευάζουμε μήτρες στήλες $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ χρησιμοποιώντας τον constructor

```
V = sp.Matrix([1,2,3])
```

ή, ισοδύναμα την εντολή

```
V = sp.Matrix([[1],[2],[3]])
```

Στην περίπτωση που χρειαζόμασταν την μήτρα γραμμή $\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ η σύνταξη θα ήταν

```
V = sp.Matrix([[1,2,3]])
```

Μπορούμε να κατασκευάζουμε τυχαίες μήτρες (δηλαδή μήτρες με στοιχεία τυχαίους (ακέραιους) αριθμούς) χρησιμοποιώντας την εντολή `randMatrix(rows, cols, minvalue, maxvalue)`

Παράδειγμα

```
import sympy as sp
R = sp.randMatrix(7,7,0,10)
print("A random nonnegative", R.rows, "x", R.cols, "matrix:")
print(R.table(sp.StrPrinter()))
```

Output:

```
A random nonnegative 7 x 7 matrix:
[3,  4, 0, 8, 10, 1,  7]
[2,  0, 8, 0,  4, 7,  4]
[8,  2, 4, 0,  7, 8,  1]
[6, 10, 0, 9,  6, 6,  7]
[2,  7, 6, 3,  5, 9, 10]
[8, 10, 5, 3,  8, 0,  1]
[4,  3, 6, 3,  8, 5,  6]
```

Πράξεις μητρών

Η πρόσθεση μητρών και ο πολλαπλασιασμός μήτρας επί πραγματικό αριθμό συμβολίζονται με τις εντολές

```
D = A + B
E = 2*A
```

Η ανάστροφη της μήτρας A δίνεται από τον τελεστή `A.T`

```
F = A.T
```

Το γινόμενο δύο μητρών, όταν ορίζεται, συμβολίζεται με τον τελεστή `@`

```
G = A@B
```

Η k-οστή δύναμη μιας τετραγωνικής μήτρας A υπολογίζεται με τον τελεστή `A**k`:

```
H = A**3
```

Αντίστοιχα, η αντίστροφη της μήτρας A (αν υπάρχει) υπολογίζεται με την τον τελεστή `A**(-1)`.

```
P = A**(-1)
```

Παράδειγμα

```
import sympy as sp

k = sp.symbols('k')
A = sp.Matrix([[5,-6],[1,0]])
B = sp.Matrix(([[1,1],[2,3]]))
print("A:", A, "B:", B)
print("A.T:",A.T) #transpose matrix
print("A+2*B:",A+2*B) #addition and multiplication by scalar
print("A*B:",A@B) #matrix product
print("A**5:",A**5) #matrix power
print("A**k:") #symbolic matrix power
print((A**k).table(sp.StrPrinter()))
print("A**(-1):",A**(-1)) #inverse matrix
```

Output:

```
A: Matrix([[5, -6], [1, 0]]) B; Matrix([[1, 1], [2, 3]])
A.T: Matrix([[5, 1], [-6, 0]])
A+2*B: Matrix([[7, -4], [5, 6]])
A*B: Matrix([[-7, -13], [1, 1]])
A**5: Matrix([[665, -1266], [211, -390]])
A**k:
[-2*2**k + 3*3**k, 6*2**k - 6*3**k]
[    -2**k + 3**k, 3*2**k - 2*3**k]
A**(-1): Matrix([[0, 1], [-1/6, 5/6]])
```

Πράξεις μητρών

Μπορούμε να επαυξήσουμε μια μήτρα A με διαστάσεις $m \times n$ προσθέτοντας στο τέλος της (μετά την τελευταία στήλη) μια δεύτερη μήτρα B με διαστάσεις $m \times k$ δημιουργώντας μια επαυξημένη μήτρα με διαστάσεις $m \times (n + t)$

$$H = [A|B]$$

χρησιμοποιώντας την εντολή

```
H = A.row_join(B)
```

Αντίστοιχα, μπορούμε να επαυξήσουμε μια μήτρα με διαστάσεις $m \times n$ προσθέτοντας στο τέλος της (μετά την τελευταία γραμμή) μια δεύτερη μήτρα B με διαστάσεις $k \times n$ δημιουργώντας μια επαυξημένη μήτρα με διαστάσεις $(m + k) \times n$

$$J = \begin{bmatrix} A \\ B \end{bmatrix} = [A^T|B^T]$$

χρησιμοποιώντας την εντολή

```
J = A.col_join(B)
```

Πράξεις μητρών

```
import sympy as sp
a = sp.symbols('a')
A = sp.Matrix([[1,2,3,a],[1,-1,5,1],[2,3,1,-1]])
B = sp.Matrix([[1],[2],[3]])
C = sp.Matrix([[4,5,6,7],[8,8,9,9]])
H = A.row_join(B)    #H = [A | B]
J = A.col_join(C)    #J = [A  = [A.T | C.T].T
                     #     C]
print("H:")  #contains the joined rows of A and B
print(H.table(sp.StrPrinter()))
print("J:")  #contains the joined cols of A and B
print(J.table(sp.StrPrinter()))
```

Output:

```
H:
[1,  2, 3,  a, 1]
[1, -1, 5,  1, 2]
[2,  3, 1, -1, 3]
J:
[1,  2, 3,  a]
[1, -1, 5,  1]
[2,  3, 1, -1]
[4,  5, 6,  7]
[8,  8, 9,  9]
```

Πράξεις μητρών

Επίσης, μπορούμε να διαγράψουμε την γραμμή i ή την στήλη j μιας μήτρας A χρησιμοποιώντας αντίστοιχα τις εντολές

```
A.row_del(i)
A.col_del(j)
```

Μπορούμε να αντιγράψουμε την γραμμή i ή την στήλη j μιας μήτρας A χρησιμοποιώντας αντίστοιχα τις εντολές

```
R = A.row(i)
C = A.col(j)
```

Για να δημιουργήσουμε ένα αντίγραφο μιας μήτρας A χρησιμοποιούμε την εντολή

```
B = A.copy()
```

Πράξεις μητρών

Στην περίπτωση όπου θέλουμε να αντικαταστήσουμε τις συμβολικές τιμές μιας μήτρας A από συγκεκριμένες τιμές ή με άλλες παραστάσεις μπορούμε να χρησιμοποιήσουμε την εντολή subs.

Παράδειγμα

```
a = sp.symbols('a')
A = sp.Matrix([[1,2,3,a],[1,-1,5,1],[2,3,1,-1]])
R = A.subs(a,7)
print("A:",A)
print("R:",R)
```

Output:

```
A: Matrix([[1, 2, 3, a], [1, -1, 5, 1], [2, 3, 1, -1]])
R: Matrix([[1, 2, 3, 7], [1, -1, 5, 1], [2, 3, 1, -1]])
```

Παρατηρήστε ότι η αντικατάσταση $a = 7$ δεν άλλαξε την μήτρα A

Ο αλγόριθμος του Gauss

Ο αλγόριθμος του Gauss μετασχηματίζει μια μήτρα A σε μια R-ισοδύναμη υποβαθμισμένη (ανηγμένη) κλιμακωτή μήτρα B με την βοήθεια πράξεων γραμμών.
Η βιβλιοθήκη `sympy` διαθέτει την μέθοδο `rref` (`reduced row echelon form`) η οποία υπολογίζει την ισοδύναμη μήτρα (ακόμα και στην περίπτωση όπου περιέχει σύμβολα αντί για αριθμούς) καθώς και τα ηγούμενα (κύρια) στοιχεία (pivots) κάθε γραμμής:

Παράδειγμα

```
import sympy as sp
a = sp.symbols('a')
A = sp.Matrix([[1,2,3,a,1],[1,-1,5,1,2],[2,3,1,-1,3]])
ReducedA, pivots = A.rref()
print("Reduced A:\n", ReducedA.table(sp.StrPrinter()))
print("pivots:", pivots)
```

Output:

```
Reduced A:
[1, 0, 0, -16*a/17 - 6/17, 37/17]
[0, 1, 0,   9*a/17 - 3/17, -7/17]
[0, 0, 1,   5*a/17 + 4/17, -2/17]
pivots: (0, 1, 2)
```

Ορίζουσα μήτρας

Στην περίπτωση μιας τετραγωνικής μήτρας A μπορούμε να υπολογίσουμε την ορίζουσα της A χρησιμοποιώντας την εντολή

```
A.det()
```

Παράδειγμα

```
import sympy as sp
x, y, z = sp.symbols('x, y, z')
A = sp.Matrix([[1,x,x**2],[1,y,y**2],[1,z,z**2]]) #Vandermonde
    matrix
D = A.det()
print("D:",D)
print("D:",sp.factor(D))
```

Output:

```
D: -x**2*y + x**2*z + x*y**2 - x*z**2 - y**2*z + y*z**2
D: -(x - y)*(x - z)*(y - z)
```

Rank μήτρας

Μπορούμε να υπολογίσουμε το rank μιας μήτρας A χρησιμοποιώντας την εντολή

```
A.rank()
```

Παράδειγμα

```
import sympy as sp
A = sp.Matrix([[1,2,3,4],[2,5,7,9],[3,7,10,13]])
r = A.rank()
print("Rank of matrix:",A,"is",r)
```

Output:

```
Rank of matrix: Matrix([[1, 2, 3, 4], [2, 5, 7, 9], [3, 7, 10,
    13]]) is 2
```

Rank μήτρας

Εναλλακτικά, μπορούμε να υπολογίσουμε το rank μιας μήτρας A εκτελώντας τον αλγόριθμο του Gauss με την εντολή `rref` και μετρώντας το πλήθος των pivots που επιστρέφει η εντολή.

Παράδειγμα

```
import sympy as sp
A = sp.Matrix([[1,2,3,4],[2,5,7,9],[3,7,10,13]])
ReducedA, pivots = A.rref()
print("Rank of matrix:",A,"is",len(pivots))
```

Output:

```
Rank of matrix: Matrix([[1, 2, 3, 4], [2, 5, 7, 9], [3, 7, 10,
    13]]) is 2
```

Ή, πιο συνοπτικά

```
import sympy as sp
A = sp.Matrix([[1,2,3,4],[2,5,7,9],[3,7,10,13]])
print("Rank of matrix:",A,"is",len(A.rref()[1]))
```

Γραμμικά συστήματα

Η βιβλιοθήκη sympy διαθέτει μεθόδους για την επίλυση γραμμικών συστημάτων με παραμέτρους ή/και συστημάτων με άπειρες λύσεις.
Μια τέτοια μέθοδος είναι η linsolve:
Αν A, B είναι οι μήτρες των συντελεστών και των σταθερών του συστήματος και $[x, y, z, w]$ είναι το διάνυσμα των μεταβλητών, τότε η εντολή

```
X = sp.linsolve((A,B), [x, y, z, w])
```

επιστρέφει το διάνυσμα X των λύσεων του συστήματος $AX = B$.

Παράδειγμα

```
import sympy as sp
S = [x,y,z,w,a,b] = sp.symbols('x y z w a b')
A = sp.Matrix([[1,2,4,3],[2,0,4,2],[3,2,8,5]])
#print("A:",A.table(sp.StrPrinter()))
B = sp.Matrix([1,2,3])
#print("B:",B.table(sp.StrPrinter()))

X = sp.linsolve((A,B), [x, y, z, w])
print("General case:", X)
```

Output:

```
General case: FiniteSet((-w - 2*z + 1, -w - z, z, w))
```

Γραμμικά συστήματα

Υπάρχουν όμως περιπτώσεις όπου η `linsolve` δεν πετυχαίνει να βρει όλες τις λύσεις για όλες τις περιπτώσεις, όπως στο παράδειγμα:

```
import sympy as sp
x,y,a = sp.symbols('x y a')
A = sp.Matrix([[1,2],[2,4]])
B = sp.Matrix([1,a])
X = sp.linsolve((A,B), [x, y])
print("General case:", X) #Fails to find solutions!
a = 2
B = sp.Matrix([1,a])
X = sp.linsolve((A,B), [x, y])
print("Case a =", a, X)
```

Output:

```
General case: EmptySet
Case a = 2 FiniteSet((1 - 2*y, y))
```

Η εξήγηση είναι ότι η επαυξημένη μήτρα $E = [A|B] = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & a \end{bmatrix}$ είναι R-ισοδύναμη με την μήτρα $\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & a-2 \end{bmatrix}$ όπου η sympy θεωρεί $a - 2 \neq 0$, οπότε προκύπτει η αδύνατη εξίσωση $0x + 0y = a - 2$.

Γραμμικά συστήματα

Μια άλλη παρόμοια μέθοδος είναι η `solve_linear_system`:
Αν A, B είναι οι μήτρες των συντελεστών και των σταθερών του συστήματος, $[x, y, z]$ είναι το διάνυσμα των μεταβλητών και $E = [A|B]$ η επαυξημένη μήτρα, τότε η μέθοδος

```
X = sp.solve_linear_system(A.row_join(B), x, y, z)
```

επιστρέφει το διάνυσμα X των λύσεων του συστήματος $AX = B$.

Γραμμικά συστήματα

Όπως και προηγουμένως, η μέθοδος `solve_linear_system` δεν πετυχαίνει πάντα να βρει όλες τις λύσεις:

```
import sympy as sp
S = [x,y,z,a,b] = sp.symbols('x y z a b')
A = sp.Matrix([[1,1,0],[2,0,1],[b,1,1]])
B = sp.Matrix([1,a,4])
E = A.row_join(B) #Augmented matrix
X = sp.solve_linear_system(A.row_join(B), x, y, z)
print("General case:", X)
b = 3
A = sp.Matrix([[1,1,0],[2,0,1],[b,1,1]])
X = sp.solve_linear_system(A.row_join(B), x, y, z)
print("Case b =",b,X) #Fail to find solutions!
a = 3
B = sp.Matrix([1,a,4])
X = sp.solve_linear_system(A.row_join(B), x, y, z)
print("Case b =",b,", a =", a,X)
```

Output:

```
General case: {x: (-a + 3)/(b - 3), y: (a + b - 6)/(b - 3), z: (a*(
    b - 1) - 6)/(b - 3)}
Case b = 3 None
Case b = 3 , a = 3 {y: z/2 - 1/2, x: -z/2 + 3/2}
```

Γραμμικά συστήματα - Παραδείγματα

Άσκηση 1

Να λυθεί και να διερευνηθεί το σύστημα

$$\begin{cases} x + y = 1 \\ 2x + z = a \\ bx + y + z = 4 \end{cases}$$

όπου x, y, z *είναι οι άγνωστοι και* a, $b \in \mathbb{R}$ *είναι παράμετροι.*

Γραμμικά συστήματα - Παραδείγματα

Λύση Έχουμε ένα 3×3 σύστημα.

Θεωρούμε τις μήτρες A, B των συντελεστών και των σταθερών του συστήματος

$$A = \begin{bmatrix} 1 & 1 & 0 \\ 2 & 0 & 1 \\ b & 1 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \\ a \\ 4 \end{bmatrix}$$

```
import sympy as sp
S = [x,y,z,a,b] = sp.symbols('x y z a b')
A = sp.Matrix([[1,1,0],[2,0,1],[b,1,1]])
B = sp.Matrix([1,a,4])
D = A.det()
print("D:",D)
```

Output:

```
D: b - 3
```

Η ορίζουσα D των συντελεστών του συστήματος ισούται με

$$D \overset{C_2 -> C_2 - C_1}{=} \begin{vmatrix} 1 & 0 & 0 \\ 2 & -2 & 1 \\ b & 1-b & 1 \end{vmatrix} = 1 \cdot \begin{vmatrix} 2 & -2 \\ b & 1-b \end{vmatrix} = b-3$$

Γραμμικά συστήματα - Παραδείγματα

Διακρίνουμε δύο περιπτώσεις:

- Αν $D \neq 0$, δηλαδή $b \neq 3$, τότε το σύστημα έχει μοναδική λύση

$$x = \frac{D_x}{D} = \frac{\begin{vmatrix} 1 & 1 & 0 \\ a & 0 & 1 \\ 4 & 1 & 1 \end{vmatrix}}{b-3} = \frac{3-a}{b-3}$$

$$y = \frac{D_y}{D} = \frac{\begin{vmatrix} 1 & 1 & 0 \\ 2 & a & 1 \\ b & 4 & 1 \end{vmatrix}}{b-3} = \frac{a+b-6}{b-3}$$

$$z = \frac{D_z}{D} = \frac{\begin{vmatrix} 1 & 1 & 0 \\ 2 & a & 1 \\ b & 4 & 1 \end{vmatrix}}{b-3} = \frac{ab-a-6}{b-3}$$

```
X = [] #solution vector
#Cramer: x=Dx/D, y=Dy/D, z=Dz/D.
C = A.copy()
#In every loop (except the first) we
#replace exactly two columns of C
#For every column i of C
for i in range(0,len(B)):
    #For every row j of C
    for j in range(0,len(B)):
        #Set the column i of C equal to B
        C[j,i]=B[j]
        #Correct the previous column of C
        if i>0:
            C[j,i-1]=A[j,i-1]
    print("A"+str(S[i])+":",C,
          "D"+str(S[i])+":",C.det())
    X.append(C.det()/D)
print("Solutions:",X)
```

Output:

```
Ax: Matrix([[1, 1, 0], [a, 0, 1], [4, 1,
    1]]) Dx: -a + 3
Ay: Matrix([[1, 1, 0], [2, a, 1], [b, 4,
    1]]) Dy: a + b - 6
Az: Matrix([[1, 1, 1], [2, 0, a], [b, 1,
    4]]) Dz: a*b - a - 6
Solutions: [(-a + 3)/(b - 3), (a + b - 6)/(
    b - 3), (a*b - a - 6)/(b - 3)]
```

Γραμμικά συστήματα - Παραδείγματα

- Αν $D = 0$, δηλαδή $b = 3$, τότε το σύστημα γίνεται

$$\begin{cases} x + y = 1 \\ 2x + z = a \\ 3x + y + z = 4 \end{cases}$$

Θεωρούμε την επαυξημένη μήτρα E του συστήματος

$$E = [A|B] = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 2 & 0 & 1 & a \\ 3 & 1 & 1 & 4 \end{bmatrix}$$

και χρησιμοποιούμε τον αλγόριθμο του Gauss

$$E \overset{\substack{R_2 \to R_2 - 2R_1 \\ R_3 \to R_3 - 3R_1}}{\sim} \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & -2 & 1 & a-2 \\ 0 & -2 & 1 & 1 \end{bmatrix}$$

$$\overset{R_3 \to R_3 - R_2}{\sim} \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & -2 & 1 & a-2 \\ 0 & 0 & 0 & 3-a \end{bmatrix}$$

```
b = 3
print("Case b =",b)
A = sp.Matrix([[1,1,0],[2,0,1],[b,1,1]])
print("A:",A,"B:",B)
E = A.row_join(B)
print("E:",E)
(Ereduced, pivots) = E.rref()
print("Reduced E:",Ereduced)
print("Pivot elements:",pivots)
```

Γραμμικά συστήματα - Παραδείγματα

⋆ Αν $a \neq 3$, τότε το σύστημα είναι αδύνατο, διότι η τελευταία γραμμή αντιστοιχεί στην εξίσωση

$$0x + 0y + 0z = 3 - a$$

Output:

```
Case b = 3
A: Matrix([[1, 1, 0], [2, 0, 1], [3, 1,
    1]]) B: Matrix([[1], [a], [4]])
E: Matrix([[1, 1, 0, 1], [2, 0, 1, a], [3,
    1, 1, 4]])
Reduced E: Matrix([[1, 0, (-2*a + 6)/(-4*a
    + 12), 0], [0, 1, (2*a - 6)/(-4*a +
    12), 0], [0, 0, 0, 1]])
Pivot elements: (0, 1, 3)
```

Γραμμικά συστήματα - Παραδείγματα

⋆ Αν $a = 3$, τότε προκύπτει το απλοποιημένο σύστημα

$$E = \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & -2 & 1 & -1 \end{bmatrix}$$

δηλαδή

$$\begin{cases} x + y = 1 \\ -2y + z = -1 \end{cases}$$

Άρα, το σύστημα έχει άπειρες λύσεις (x, y, z) όπου

$x = 1-y,\ z = -1+2y,\ y \in \mathbb{R}$

```
a = 3
print("Case b =",b, ", a =", a)
A = sp.Matrix([[1,1,0],[2,0,1],[b,1,1]])
B = B = sp.Matrix([1,a,4])
print("A:",A,"B:",B)
E = A.row_join(B)
print("E:",E)
(Ereduced,pivots) = E.rref()
print("Reduced E:",Ereduced)
print("Pivot elements:",pivots)
```

Output:

```
Case b = 3 , a = 3
A: Matrix([[1, 1, 0], [2, 0, 1], [3, 1,
    1]]) B: Matrix([[1], [3], [4]])
E: Matrix([[1, 1, 0, 1], [2, 0, 1, 3], [3,
    1, 1, 4]])
Reduced E: Matrix([[1, 0, 1/2, 3/2], [0, 1,
    -1/2, -1/2], [0, 0, 0, 0]])
Pivot elements: (0, 1)
```

Γραμμικά συστήματα - Παραδείγματα

Άσκηση 2

Να λυθεί και να διερευνηθεί το σύστημα

$$\begin{cases} x + ay + 2z = 1 \\ x + (2a - 1)y + 3z = 1 \\ x + ay + (a - 3)z = 2a - 1 \end{cases}$$

όπου x, y, z *είναι οι άγνωστοι και* $a \in \mathbb{R}$ *είναι παράμετρος.*

Γραμμικά συστήματα - Παραδείγματα

Λύση

Έχουμε ένα 3×3 σύστημα. Θεωρούμε τις μήτρες A, B των συντελεστών και των σταθερών του συστήματος

$$A = \begin{bmatrix} 1 & a & 2 \\ 1 & 2a-1 & 3 \\ 1 & a & a-3 \end{bmatrix} \quad B = \begin{bmatrix} 1 \\ 1 \\ 2a-1 \end{bmatrix}$$

Εδώ θα χρησιμοποιήσουμε την μέθοδο `linsolve` για να βρούμε τις λύσεις του συστήματος.

```
1 import sympy as sp
2 S = [x,y,z,a] = sp.symbols('x y z a')
3 A = sp.Matrix([[1,a,2],[1,2*a-1,3],[1,a,a
      -3]])
4 #print("A:",A)
5 B = sp.Matrix([1,1,2*a-1])
6 #print("B:",B)
7 X = sp.linsolve((A,B),[x,y,z])
```

Output:

```
1 Solutions: {(-(a + 1)/(a - 5), -2/(a - 5),
      2*(a - 1)/(a - 5))}
```

Γραμμικά συστήματα - Παραδείγματα

Η λύση που μας έδωσε η μέθοδος `linsolve` ορίζεται όταν $a \neq 5$, επομένως με βάση αυτό συμπεραίνουμε ότι όταν $a \neq 5$ τότε το σύστημα έχει μοναδική λύση

$$x = \frac{a+1}{5-a},\ y = \frac{2}{5-a},\ z = \frac{2(a-1)}{a-5}$$

Απομένει να εξετάσουμε τι συμβαίνει όταν $a = 5$. Τότε το σύστημα παύει να είναι παραμετρικό.

```
a = 5
A = sp.Matrix([[1,a,2],[1,2*a-1,3],[1,a,a-3]])
#print("A:",A)
B = sp.Matrix([1,1,2*a-1])
#print("B:",B)
X = sp.linsolve((A,B),[x,y,z])
print("Case a =", a, "Solutions:", X)
```

Output:

```
Case a = 5 Solutions: EmptySet()
```

Άρα, όταν $a = 5$ το σύστημα είναι αδύνατο. Τελικά συμπεραίνουμε ότι

όταν $a \neq 5$, το σύστημα (είναι παραμετρικό) και έχει μοναδική λύση.

όταν $a = 5$, το σύστημα είναι αδύνατο.

Έχουμε κατανοήσει πλήρως αυτό το σύστημα?

Γραμμικά συστήματα - Παραδείγματα

Η απάντηση είναι όχι! Αν θέσουμε $a = 1$ τότε έχουμε άπειρες λύσεις!

```
a = 1
A = sp.Matrix([[1,a,2],[1,2*a-1,3],[1,a,a-3]])
B = sp.Matrix([1,1,2*a-1])
X = sp.linsolve((A,B),[x,y,z])
print("Case a =", a, "Solutions:", X)
```

Output:

```
Case a = 1 Solutions: {(-y + 1, y, 0)}
```

Ο λόγος που συμβαίνει αυτό ότι η ορίζουσα D της μήτρας των συντελεστών του συστήματος έχει δύο ρίζες: Το 5 και 1.

$$D = \begin{vmatrix} 1 & a & 2 \\ 1 & 2a-1 & 3 \\ 1 & a & a-3 \end{vmatrix} = (a-5)(a-1)$$

Γραμμικά συστήματα - Παραδείγματα

Η ρίζα 1 όμως ““έτυχε”” να είναι και ρίζα όλων των οριζουσών D_x, D_y, D_z

$$D_x = -(a-1)(a+1)$$

$$D_y = -2(a-1)$$

$$D_z = 2(a-1)^2$$

οπότε απλοποιήθηκε και δεν εμφανίστηκε στον παρονομαστή των απαντήσεων.

```
import sympy as sp
S = [x,y,z,a] = sp.symbols('x y z a')
A = sp.Matrix([[1,a,2],[1,2*a-1,3],[1,a,a
    -3]])
B = sp.Matrix([1,1,2*a-1])

D = sp.factor(A.det())
print("D:",D)
roots = sp.solve(D,a)
print("Roots of D:",roots)

C = A.copy()
#In every loop (except the first) we
#replace exactly two columns of C
#For every column i of C
for i in range(0,len(B)):
    #For every row j of C
    for j in range(0,len(B)):
        #Set the column i of C equal to B
        C[j,i]=B[j]
        #Correct the previous column of C
        if i>0:
            C[j,i-1]=A[j,i-1]
    print("A"+str(S[i])+":",C,
          "D"+str(S[i])+":",sp.factor(C.det
     ()))

X = sp.linsolve((A,B),[x,y,z])
print("Solutions:", X)
```

Γραμμικά συστήματα - Παραδείγματα

Επομένως, δεν μας έδωσε προειδοποίηση ότι το σύστημα μπορεί να συμπεριφέρεται διαφορετικά για κάποιο άλλο $a \in \mathbb{R}$.

Το **δίδαγμα** αυτών των δύο παραδειγμάτων είναι να μην εμπιστευόμαστε τυφλά τις έτοιμες μεθόδους αλλά να επιβεβαιώνουμε τις απαντήσεις μας μέσα από την θεωρία που γνωρίζουμε.

Output:

```
D: (a - 5)*(a - 1)
Roots of D: [1, 5]
Ax: Matrix([[1, a, 2], [1, 2*a - 1, 3], [2*
    a - 1, a, a - 3]]) Dx: -(a - 1)*(a +
    1)
Ay: Matrix([[1, 1, 2], [1, 1, 3], [1, 2*a -
    1, a - 3]]) Dy: -2*(a - 1)
Az: Matrix([[1, a, 1], [1, 2*a - 1, 1], [1,
    a, 2*a - 1]]) Dz: 2*(a - 1)**2
Solutions: {(-(a + 1)/(a - 5), -2/(a - 5),
    2*(a - 1)/(a - 5))}
```

Η βιβλιοθήκη NumPy

Για χρησιμοποιήσουμε την βιβλιοθήκη numpy δίνουμε την εντολή

```
import numpy as np
```

οπότε όλες οι διαθέσιμες μέθοδοι της numpy μπορούν να προσπελαστούν γράφοντας

```
np.Onoma_Methodou(Orismata)
```

ή, εναλλακτικά μπορούμε να προσπελάσουμε απευθείας (χωρίς το πρόθεμα np.) όλες τις μεθόδους δίνονται την εντολή

```
from numpy import *
```

Μήτρες

Για την numpy μια μήτρα είναι μια λίστα από γραμμές. Για να ορίσουμε την μήτρα $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ γράφουμε

```
A = np.array([[1,2,3],[4,5,6]])
```

Για να αλλάξουμε ή να προσπελάσουμε το στοιχείο της A που βρίσκεται στην γραμμή i και στήλη j γράφουμε

```
x = A[i,j]
```

Όπως συνήθως οι γραμμές και οι στήλες έχουν δείκτες που αρχίζουν από 0. Οι διαστάσεις της A περιέχονται στην μεταβλητή `A.shape`. Ειδικότερα ο αριθμός των γραμμών και των στηλών είναι αποθηκευμένος στις μεταβλητές `A.rows` και `A.cols` αντίστοιχα.

```
mrows, ncols = A.shape
m = A.rows
n = A.cols
```

Μήτρες

Μπορούμε να κατασκευάσουμε μια (άδεια) μήτρα με διαστάσεις $m \times n$ χρησιμοποιώντας τους constructors `zero(m,n)` ή `ones(m,n)` που κατασκευάζουν μια μήτρα διαστάσεων $m \times n$ που έχει παντού μηδέν και παντού άσσους αντίστοιχα.

```
B = sp.zeros((2,3))
C = sp.ones((2,3))
```

Μπορούμε να αναπαριστούμε τα διανύσματα ως μήτρες στήλες χρησιμοποιώντας την σύνταξη

```
V = np.transpose(np.array([[1,2,4]]))
```

Πράξεις μητρών

Η πρόσθεση μητρών και ο πολλαπλασιασμός μήτρας επί πραγματικό αριθμό συμβολίζονται με τις εντολές

```
D = A + B
E = 2*A
```

Η ανάστροφη της μήτρας A δίνεται από τον τελεστή `A.T` ή χρησιμοποιώντας την μέθοδο `transpose(A)`

```
F = np.transpose(A)
```

Το γινόμενο δύο μητρών, όταν ορίζεται, συμβολίζεται με τον τελεστή `@`

```
G = A@B
```

Για να δημιουργήσουμε ένα αντίγραφο μιας μήτρας A χρησιμοποιούμε την εντολή

```
B = A.copy()
```

Στην περίπτωση μιας τετραγωνικής μήτρας A μπορούμε να υπολογίσουμε την ορίζουσα της A χρησιμοποιώντας την εντολή

```
np.linalg.det(A)
```

Πρόσθεση και βαθμωτός πολλαπλασιασμός διανυσμάτων

```
import numpy as np

x = np.array([1,-3,5])
y = np.array([7,3,2])

print("x =", x)
print("y =", y)
print("x + y =", x + y)
print("x - y =", x - y)
print("2*x =", 2*x)
print("3*y =", 3*y)
print("2*x + 3*y =", 3*x + 5*y)
```

Output:

```
x = [ 1 -3  5]
y = [7 3 2]
x + y = [8 0 7]
x - y = [-6 -6  3]
2*x = [ 2 -6 10]
3*y = [21  9  6]
2*x + 3*y = [38  6 25]
```

Εσωτερικό γινόμενο

Το εσωτερικό γινόμενο δύο διανυσμάτων **v** και **u** υπολογίζεται με την εντολή

```
v.dot(u)
```

ή με την μέθοδο `np.dot(v, u)`.

```
import numpy as np
x = np.array([1,-3,5])
y = np.array([7,3,2])
z = np.array([8,6,2])
print("x =", x)
print("y =", y)
print("z =", z)
print("x*y =",np.dot(x,y))
print("x*z =",x.dot(z))
```

Output:

```
x = [ 1 -3  5]
y = [7 3 2]
z = [8 6 2]
x*y = 8
x*z = 0
```

Μήκος, απόσταση, γωνία διανυσμάτων

Το μήκος $\|v\|$ του διανύσματος v υπολογίζεται χρησιμοποιώντας την μεθόδους `norm()` της `numpy.linalg`.

```
import numpy as np

x = np.array([1,-3,5])
w = np.array([2,4,4])

print("x =", x)
print("w =", w)
print("|x|=",np.linalg.norm(x))
print("|w|=",np.sqrt(np.dot(w,w)))
```

Output:

```
x = [ 1 -3  5]
w = [2 4 4]
|x|= 5.916079783099616
|w|= 6.0
```

Μήκος, απόσταση, γωνία διανυσμάτων

```
import numpy as np

x = np.array([7,-3,5])
y = np.array([1,4,9])

print("x =", x)
print("y =", y)
print("d(x,y) =",np.linalg.norm(x-y))
print("d(x,y) =",np.sqrt(np.dot(x-y,x-y)))
```

Output:

```
x = [ 7 -3  5]
y = [1 4 9]
d(x,y) = 10.04987562112089
d(x,y) = 10.04987562112089
```

Μήκος, απόσταση, γωνία διανυσμάτων

```
import numpy as np
x, y = np.array([1,1,4]), np.array([2,7,7])
z, w = np.array([1,-1,2]), np.array([2,6,2])
print("x =", x)
print("y =", y)
print("z =", z)
print("w =", w)
costheta1 = np.dot(x,y)/(np.sqrt(np.dot(x,x)*np.dot(y,y)))
theta1 = np.arccos(costheta1)
print("cos(theta1)=", costheta1, "theta1=", theta1)
costheta2 = np.dot(z,w)/(np.sqrt(np.dot(z,z)*np.dot(w,w)))
theta2 = np.arccos(costheta2)
print("cos(theta2)=", costheta2, "theta2=", theta2)
```

Output:

```
x = [1 1 4]
y = [2 7 7]
z = [ 1 -1  2]
w = [2 6 2]
cos(theta1)= 0.8635060518172727 theta1= 0.528615703143876 5
cos(theta2)= 0.0 theta2= 1.5707963267948966
```

Γραμμική ανεξαρτησία

```
import numpy as np
b = np.array([10,-5,5])
x1, x2, x3 = np.array([3,-1,2]), np.array([1,0,4]), np.array
    ([-1,1,1])
A = np.array([[3,1,-1],[-1,0,1],[2,4,1]])
print("b =", b)
print("x1 =", x1)
print("x2 =", x2)
print("x3 =", x3)
print("A =\n",A)
[l1,l2,l3] = np.linalg.solve(A, b)
print("l1 l2 l3 =", l1,l2,l3)
print("l1* x1 + l2*x2 + l3*x3 =", l1*x1 + l2*x2 + l3*x3)
```

Output:

```
b = [10 -5  5]
x1 = [ 3 -1  2]
x2 = [1 0 4]
x3 = [-1  1  1]
A =
 [[ 3  1 -1]
 [-1  0  1]
 [ 2  4  1]]
l1 l2 l3 = 1.9999999999999993 1.0000000000000004 -3.000000000000001
l1* x1 + l2*x2 + l3*x3 = [10. -5.  5.]
```

Γραμμική ανεξαρτησία

```
import numpy as np
b = np.array([0,0,0])
A = np.transpose(np.array([[2,-1,0],[-5,2,2],[-1,0,2]]))
print("b =", b)
print("A =\n",A)
try:
    [l1,l2,l3] = np.linalg.solve(A, b)
    print("l1 l2 l3 =", l1,l2,l3)
    print("The system A x = b has unique solution, hence the
    vectors are linearly independent")
except:
    print("The system A x = b has infinite solutions, hence the
    vectors are not linearly independent")
```

Output:

```
b = [0 0 0]
A =
 [[ 2 -5 -1]
 [-1  2  0]
 [ 0  2  2]]
The system A x = b has infinite solutions, hence the vectors are
    not linearly independent
```

Γραμμική ανεξαρτησία

```
import numpy as np
b = np.array([0,0,0])
A = np.transpose(np.array([[2,-1,0],[1,3,1],[-1,0,2]]))
print("b =", b)
print("A =\n",A)
try:
    [l1,l2,l3] = np.linalg.solve(A, b)
    print("l1 l2 l3 =", l1,l2,l3)
    print("The system A x = b has unique solution, hence the
    vectors are linearly independent")
except:
    print("The system A x = b has infinite solutions, hence the
    vectors are not linearly independent")
```

Output:

```
b = [0 0 0]
A =
 [[ 2  1 -1]
 [-1  3  0]
 [ 0  1  2]]
l1 l2 l3 = 0.0 0.0 0.0
The system A x = b has unique solution, hence the vectors are
    linearly independent
```

Μέθοδος ελαχίστων τετραγώνων

```
import numpy as np

b = np.array([5,3,12])
A = np.transpose(np.array([[1,1,2],[1,-1,1]]))
print("b =", b)
print("A =\n",A)

#First method
x, residuals, rank, s = np.linalg.lstsq(A, b, rcond=None)
print("Least squares solution of system A x = b =", x)
print("Projection = ", x[0]*np.transpose(A)[0] + x[1]*np.transpose(
    A)[1])
print("Error", np.sqrt(residuals[0]))

#Second method
At = np.transpose(A)
newx = np.linalg.solve(At@A,At@b)
print("Solution of system At*A*x = At*b = ", newx)
```

Μέθοδος ελαχίστων τετραγώνων

Output:

```
b = [ 5  3 12]
A =
 [[ 1  1]
 [ 1 -1]
 [ 2  1]]
Least squares solution of system A x = b = [4.85714286 1.42857143]
Projection =  [ 6.28571429  3.42857143 11.14285714]
Error 1.6035674514745457
Solution of system At*A*x = At*b =  [4.85714286 1.42857143]
```