2η Εργασία

‘Γραμμική και Συνδυαστική Βελτιστοποίηση’ 2021-2022

Ιωάννης Τσάμπρας / 1066584 / (κατ. Υπολογιστές) / 4ο έτος

Επικοινωνία : ioannistsampras@computer.org / up1066584@upnet.gr / gtsambras@gmail.com

1η άσκηση

A) Αρχικά συντάσουμε τον απαραίτητο κώδικα για τον επιλυτή (αρχείο **ask1.py**)

```
ergasia2 > ask1.py > ...
import numpy as np
from scipy import optimize



#Here the real deal starts
#maximize using scipy

#A.S. matrix
c=np.array([-3,-11,-9,1,29]) #normaly linprog optimizes for minimalization so we reverse the signs of C (we also ch

#Inequality matrix coef
A_ub = np.array([[0,1,1,1,-2],[-1,+1,-1,-2,-1],[1,1,1,0,-3]])




b_ub=np.array([4,0,1]) #numpy 1 d array of constant coefficients b for inequalities



#bounds
bounds=[(None,None),(0,None),(0,None),(0,None),(0,None)] #default #basically setting the range for our problem vari

#using all the matrixes we made we can now use the linprog optimizer (documentation here: https://docs.scipy.org/dc
result=optimize.linprog(c,A_ub=A_ub,b_ub=b_ub,bounds=bounds) #using the linprog optimizer (the default optimizer is

print('optimal solution :',-(result.fun))
print(result.x)
```

(Έχω μετατρέψει τον 2ο περιορισμό σε κανονική μορφή πολλαπλασιάζοντας με -1)

Και παίρουμε τα εξής αποτελέσματα για την τιμή της Α.Σ. και των μεταβλητών απόφασης:

Α.Σ.: 27.999999998918383

[-3.00000000e+00 5.00000000e-01 3.50000000e+00 3.44755137e-11 1.79574565e-11]

Πιό ανθρώπινα: **Α.Σ. 28 , X = [-3, 0.5, 3.5, 0, 0]**

Οι βασικές μεταβλητές έχουν τιμές όπως αυτές ορίζονται στον πίνακα X των αποτελεσμάτων και για τις μη βασικές αρκεί αν πάρουμε τους 3 περιορισμούς , να εισάγουμε μεταβλητές χαλάρωσης και έχοντας τις τιμές των X να λύσουμε ως προς τις χαλάρωσης:

$$x_2+x_3+x_4-2x_5+z_1=4 \overset{X[\,]}{\Rightarrow} 0,5+3,5+0-0+z_1=4 \Rightarrow z_1=0$$

$$-x_1+x_2-x_3-2x_4-x_5+z_2=0 \overset{X[\,]}{\Rightarrow} +3+0,5-3,5-0-0+z_2=0 \Rightarrow z_2=0$$

$$x_1+x_2+x_3-3x_5+z_3=1 \Rightarrow -3+0,5+3,5+0+z_3=1 \Rightarrow z_3=0$$

Οι ισότητες ισχύουν με τις **μεταβλητές χαλάρωσης να είναι μηδενικές** οπότε οι **μόνες βασικές** μεταβλητές είναι οι 3 πρώτες μεταβλητές απόφασης **(χ1,χ2,χ3).**

Ο βασικός πίνακας προκύπτει:

X1	X2	X3
0	1	1
-1	1	-1
1	1	1

Δεσμευτικοί είναι οι περιορισμοί οι οποίοι κατά την χρήση των τιμών της βέλτιστης λύσης ικανοποιείται η ισότητα. Αντίστοιχα, μη δεσμευτικοί είναι αυτοί για τους οποίους ικανοποιείται η ανισότητα.

Άρα οι μόνοι **μη δεσμευτικοί περιορισμοί είναι τα x1εR , x2,x3>=0**

Μπορούμε να πούμε πως **η βέλτιστη κορυφή** ορίζεται από την **τομή των υπερεπιπέδων** των **δεσμευτικών περιορισμών**.

Β) Επιλέγω την X2 και την X5.

Παραμετροποιώ τον κωδικα μου ώστε να βρίσκω την κατάσταση εξόδου του αλγορύθμου και την τιμή της αντικειμενικής συνάρτησης καθώς εφαρμόζω αλλαγές στους συντελεστές. Για την X2 αρχικά ελέγχω μια μεγάλη περιοχή (ξεκίνησα απο -1000 έως 1000) και βλέπω πως η κρίσιμες περιοχές είναι κοντά στο 2 και το -2. Ύστερα με μικρότερη περιοχή και μεγαλύτερη ακρίβεια βρίσκω πώς καθώς μεταβάλλεται ο συντελεστής **προς τα επάνω** η τιμή της αντικειμενικής συνάρτησης επίσης αυξάνει μέχρι να τον αυξήσουμε κατα 2 καθώς παραπάνω από αυτό το πρόβλημα **σταματά να είναι φραγμένο**, ενώ εφαρμόζοντας **αρνητική μεταβολή** η τιμή της Α.Σ. μικραίνει έως ότου αλλάζει κορυφή με **νέες τιμές για τις μεταβλητές απόφασης** [-3.00000000e+00 2.25509114e-01 3.77449089e+00 5.07261500e-11 8.78418672e-12] ή πιό ανθρώπινα: **η x2 αλλάζει από 0.5 σε 0.443 , η x3 αλλάζει σε 3.774 από 3.5** οπότε βρίσκουμε περιθώριοα αντοχής στο **συντελεστή της X2 κατά αλλαγή [-2,2] (κατά απόλυτο τιμή [9,13])**

Ο σχετικός κώδικας (αρχείο **ask1_b_1.py**):

```python
print('_________________________________________________________________')
m2=ask1([2,0])[0]
message2=''
#print(ask1([2,0])[0])
for i in range(0,4000):
    m,of,message=ask1([-20+i/100,0])
    #print(m,m2)
    if message=='The algorithm terminated successfully and determined that the problem is unbounded.':
        print(round(-20+i/100,3),'unbound')
        break
    flag=True
    for j in range(0,len(m)):
        if round(m2[j],2)!=round(m[j],2):
            flag=False
            break

    if message2==message:
        if flag==False:
            print(round(-20+i/100,3),m,of,message)
    m2=m
    message2=message
print('_________________________________________________________________')
exit()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\ioannis\Desktop\VS CODE\grammiki sundiastikh> & C:/Users/ioannis/AppData/Local/Programs/Python/Python310/pytho
_________________________________________________________________
-2.0 [-3.00000000e+00  2.25509114e-01  3.77449089e+00  5.07261500e-11
  8.78418672e-12] 26.99999999835413 Optimization terminated successfully.
-1.99 [-3.00000000e+00  4.99999859e-01  3.50000014e+00  1.09291008e-11
  4.39070208e-11] 27.004999995985592 Optimization terminated successfully.
2.0 [-7.05841860e-01  5.08831628e+00  3.50000000e+00  2.05437705e-11
  2.29415814e+00] 28.99999999962823 Optimization terminated successfully.
2.01 unbound

_________________________________________________________________
PS C:\Users\ioannis\Desktop\VS CODE\grammiki sundiastikh>
```

Παρόμοια διαδικασία ακολουθώ και για αλλαγές στον συντελεστή της X5 (αρχείο κώδικα **ask1_b_2.py**) και βρίσκω πώς **το πρόβλημα γίνεται μη φραγμένο** αν ο συντελεστής της X5 **μεταβληθεί κατα +4 .**

Δ) Θέτουμε χ6-χ7=χ1 με χ6,χ7>=0 ώστε να έρθει το πρόβλημα σε κανονική μορφή και βρίσκουμε το δυϊκό του.

Και το δυϊκό πρόβλημα είναι:

Αρχικό

$x_2 + x_3 + x_4 - 2x_5 \le 4$ $\quad \max(11x_2 + 9x_3 - x_4 - 29x_5 + 3x_6 - 3x_7)$

$x_2 - x_3 - 2x_4 - x_5 - x_6 + x_7 \le 0$

$x_2 + x_3 - 3x_5 + x_6 - x_7 \le 1$

$x \ge 0$

Δυϊκό $\quad \min(4y_1 + 1y_3)$

$1y_1 \quad 1y_2 \quad 1y_3 \quad \ge 11$

$1y_1 \quad -1y_2 \quad 1y_3 \quad \ge 9$

$1y_1 \quad -2y_2 \quad 0y_3 \quad \ge -1$

$-2y_1 \quad -1y_2 \quad -3y_3 \quad \ge -29$

$0y_1 \quad -1y_2 \quad 1y_3 \quad \ge 3$

$0y_1 \quad 1y_2 \quad -1y_3 \quad \ge -3 \qquad y \ge 0$

Το λύνουμε με την python(**αρχείο ask1_d.py**):

ask1_d.py > ...

```
import numpy as np
from scipy import optimize


#Here the real deal starts
#maximize using scipy

#A.S. matrix
c=np.array([4,0,1])

#Inequality matrix coef
A_ub = np.array([[1,1,1],[1,-1,1],[1,-2,0],[-2,-1,-3],[0,-1,1],[0,1,-1]])*-1 #change sign to invert inequalities to normal form




b_ub=np.array([11,9,-1,-29,3,-3])*-1 #change sign to invert inequalities to normal form



#bounds
bounds=[(0,None),(0,None),(0,None)] #default #basically setting the range for our problem variables

#using all the matrixes we made we can now use the linprog optimizer (documentation here: https://docs.scipy.org/doc/scipy/refere
result=optimize.linprog(c,A_ub=A_ub,b_ub=b_ub,bounds=bounds) #using the linprog optimizer (the default optimizer is the simplex m

print('optimal solution :',(result.fun))
print(result.x)
print(result.slack)
```

```
PS C:\Users\ioannis\Desktop\VS CODE\grammiki sundiastikh> & C:/Users/ioannis/App
optimal solution : 27.999999999289283
[6. 1. 4.]
[-1.53214330e-10 -2.46043186e-10  5.00000000e+00  4.00000000e+00
 -7.56794627e-11  7.56794627e-11]
PS C:\Users\ioannis\Desktop\VS CODE\grammiki sundiastikh>
```

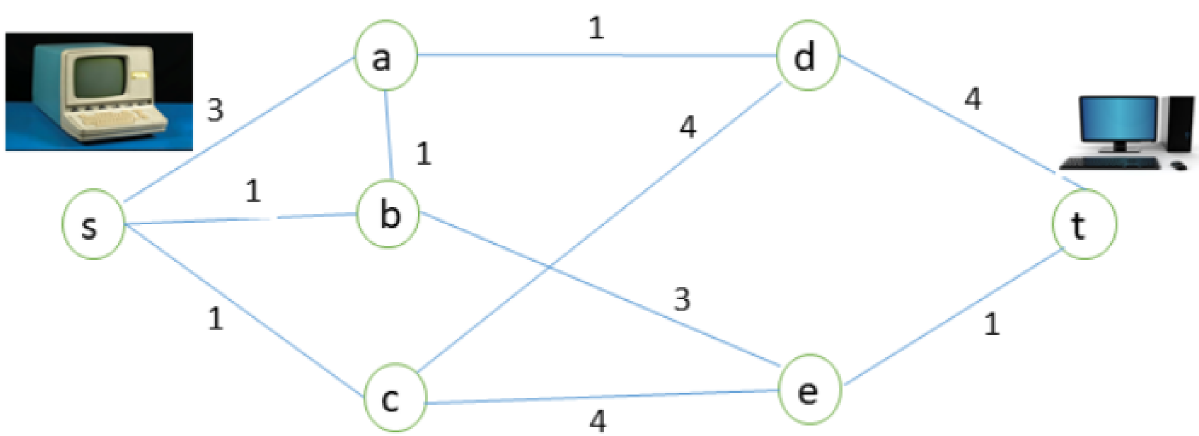
Άρα y1=6,y2=1,y3=4,y4=0,y5=0,y6=5,y7=4,y8=0,y9=0

Προφανώς το γινόμενο των μεταβλητών χαλάρωσης Χ με των μεταβλητών απόφασης Υ είναι 0 αφού όλες οι μεταβλητές χαλάρωσης Χ είναι 0.

Για το γινόμενο των μεταβλητών χαλάρωσης Υ με των μεταβλητών απόφασης Χ βρίσκουμε επίσης 0.

Άσκηση 2.

Κώδικας στο αρχείο (**ask2.py**)



Σημείωση: Στην λύση μου θα θεωρήσω πως η **θετική φορά των ροών** πάνω στις ακμές είναι αυτή που ακολουθεί **αλφαβητική προτεραιότητα εκτός από τις ακραίες ακμές** (αυτές που συνδέονται με τους κόμβους S και T) όπου **θετική φορά είναι από S και προς T**. Οι συμβάσεις αυτές έχουν σκοπό την καλύτερη κατανόηση του προβλήματος για εμένα και αποτελούν <u>υποκειμενικές επιλογές</u> που δεν επηρεάζουν την επίλυση του προβλήματος.

Αρχικά παρατηρούμε πως ο γράφος αποτελείται απο **10 δικατευθυντήριες ακμές, 5 μη αποθηκευτικούς κόμβους και 2 κόμβους με αποθήκευση**. Η **αντικειμενική συνάρτηση** είναι το **άθροισμα των ροών στις ακμές DT και ET** με θετικό πρόσημο (εναλλακτικά η συνάρτηση **θα μπορούσε να είναι το άθροισμα των ροών στις ακμές SA,SB,SC** καθώς το δίκτυο δεν έχει δυνατότητα αποθήκευσης η ροή της εξόδου είναι ίση με τη ροή της εισόδου, αυτή η συμμετρία θα επαληθευτεί και μέσω των σχέσεων που θα παραχθούν παρακάτω) και επιθυμούμε την **μεγιστοποίηση αυτού του αθροίσματος**.

Από τους **περιορισμούς της εκφώνησης** προκύπτουν οι εξής **προτάσεις**:

Α) Για κάθε μη αποθηκευτικό κόμβο προκύπτει η σχέση πως το αλγεβρικό άθροισμα των εισερχόμενων ροών ισούτε με το 0.

Β) Για κάθε ακμή προκύπτει η σχέση πως η αντίστοιχη ροή είναι κατ’ απόλυτο τιμή μικρότερη απο το όριο μεταφοράς της ακμής.

Από αυτή την περιγραφή θα προκύψουν 5 ισότητες και 10 διπλές ανισότητες για τις μεταβλητές μας. Για να φέρουμε το πρόβλημα στην κλασσική μορφή γραμμικού προγραμματισμού πρώτα θα αντικαταστήσουμε τις μεταβλητές μας έτσι ώστε οι νέες μεταβλητές να είναι όλες θετικές.

Άρα για κάθε διπλή ανισότητα προσθέτουμε και στα τρία μέλη σταθερά έτσι ώστε να μετατραπεί σε δύο ανισότητες της μορφης 0<=Xnew και Xnew<=2*Flow_limit με Xnew=Xold+Flow_limit .

Με λίγα λόγια κάνουμε αλλαγή μεταβλητών σε Xnew=Xold+Flow_limit.

Πραγματοποιούμε τις αντίστοιχες αλλαγές και στις 5 εξισώσεις οι οποίες πλεόν θα έχουν και μη μηδενικό σταθερό μέλος. Αντίστοιχες αλλαγές πρέπει να γίνουν και στην Α.Σ.

Το νέο πρόβλημα αποτελείται από εξισώσεις και ανισώσεις και είναι πλέον σε κανονική μορφή προβλήματος γραμμικού προγραμματισμού μεγιστοποίησης.

Αρχικό πρόβλημα:

Πρώτη μορφή:

$-3 \le sa \le 3$	$sa - ab - ad = 0$	Α.Σ.: $dt + et \,/\, sa + sb + sc$
$-1 \le ad \le 1$	$sb + ab - be = 0$	
$-1 \le ab \le 1$	$sc - cd - ce = 0$	
$-1 \le sb \le 1$	$ad + cd - dt = 0$	
$-3 \le be \le 3$	$be + ce - et = 0$	
$-1 \le sc \le 1$		
$-4 \le cd \le 4$		
$-4 \le ce \le 4$		
$-4 \le dt \le 4$		
$-1 \le et \le 1$		

Μετασχηματισμένο πρόβλημα:

```
sa_new,ad_new,ab_new,sb_new,be_new,sc_new,cd_new,ce_new,dt_new,et_new,>=0
sa_new=<6
ad_new=<2
ab_new=<2
sb_new=<2
be_new=<6
sc_new=<2
cd_new=<8
ce_new=<8
dt_new=<8
et_new=<2
+sa_new-ab_new-ad_new=1
+sb_new+ab_new-be_new=-1
+sc_new-cd_new-ce_new=-7
+ad_new+cd_new-dt_new=1
+be_new+ce_new-et_new=6
```

```python

#import information into code
names_1=['sa','ad','ab','sb','be','sc','cd','ce','dt','et']
flow_limits=[3,1,1,1,3,1,4,4,4,1]
node_equations=[['+sa','-ab','-ad',0],['+sb','+ab','-be',0],['+sc','-cd','-ce',0],['+ad','+cd','-dt',0],['+be','+ce','-et',0]]

def print_2nd_form(names,limits,nodes):
    #print positivity
    for the_name in names:
        print(the_name+'_new,',end='')
    print('>=0')

    #print new limits
    for the_name in names:
        print(the_name+'_new=<'+str(2*limits[names.index(the_name)]))

    #print new node equations
    for eq in nodes:
        const=eq[-1]
        for factor in eq:
            if factor==0:
                continue
            if factor[0]=='-':
                const=const-limits[names.index(factor[1:])]
            else:
                const=const+limits[names.index(factor[1:])]
            print(factor+'_new',end='')
        print('={}'.format(const))



print_2nd_form(names_1,flow_limits,node_equations)

```

Η νέα Α.Σ. είναι η ίδια (απλά όπου et και dt θα έχουμε et_new και dt_new) καθώς απλά προσθέσαμε μία σταθερά. Στο τέλος θα υπολογίσουμε την τιμή για την Α.Σ. του αρχικού προβλήματος. Άρα η νέα Α.Σ. είναι Z=DT+ET+4+1.

Έχοντας πλέον εξισώσεις σε κανονική μορφή μπορούμε να εφαρμόσουμε μέθοδο simplex.

```python
#Here the real deal starts
#maximize using scipy

#A.S. matrix
c=np.array([0,0,0,0,0,0,0,0,-1,-1]) #normaly linprog optimizes for minimalization so we reverse
#the signs of C (we also change signs in the results) to maximize

#Inequality matrix coef
A_ub = np.zeros((10, 10), int) #creates empty 10x10 matrix #our inequality A matrix is basically a
#singular (I) 10x10 matrix so we fill the diagonal of the empty matrix with ones
np.fill_diagonal(A_ub, 1)  #numpy 1 d array of constant coefficients b for inequalities

#Inequality matrix const
flow_limits_new=[]
for i in flow_limits:
    flow_limits_new.append(2*i) #we double the original limits array since our transformed problem inequalities are x=<2*old_limit
b_ub=np.array(flow_limits_new) #numpy 1 d array of constant coefficients b for inequalities

#equality matrix coef and const
names=names_1 #changing names because i copied the following code from my previous function
limits=flow_limits #changing names because i copied the following code from my previous function
constant_coef=[] #changing names because i copied the following code from my previous function
non_const_coef=[] #changing names because i copied the following code from my previous function
for eq in node_equations:
    const=eq[-1]
    temp_equation=[0,0,0,0,0,0,0,0,0,0]
    for factor in eq:
        if factor==0:
            continue
        if factor[0]=='-':
            const=const-limits[names.index(factor[1:])]
            temp_equation[names.index(factor[1:])]=-1
        else:
            const=const+limits[names.index(factor[1:])]
            temp_equation[names.index(factor[1:])]=+1

    constant_coef.append(const)
    non_const_coef.append(temp_equation)
    #using the node equation matrix from earlier i iterate each node
    # equation and i append the respective coefficients to the 1d array of b constant coefficients and the 2d array of A coefficients


A_eq=np.array(non_const_coef)      #making the numpy array based on my 2d list
b_eq=np.array(constant_coef)       #making the numpy array based on my 1d list

#bounds
bounds=(0,None) #default #basically setting the range for our problem variables

#using all the matrixes we made we can now use the linprog optimizer (documentation here:
#  https://docs.scipy.org/doc/scipy/reference/optimize.linprog-simplex.html )
result=optimize.linprog(c,A_ub=A_ub,b_ub=b_ub,A_eq=A_eq,b_eq=b_eq,bounds=bounds) #using the linprog
#optimizer (the default optimizer is the simplex method so i do not specify it)

#printing a visual seperator
print('\n\n\n')

print('optimal solution for transformed problem:',-(result.fun)) #changing signs since this is a maximazation proble, see line 39
for i,flow in enumerate(names_1): #printing all new variables with their optimal values
    print(flow+'_new : ',result.x[i])

#printing a visual seperator
print('\n\n\n')
#we need to adjust our result to the original problem

#our A.S. was dt+et, after the transformation it becomes dt_new+et_new+4+1=> A.S._new=A.S._old+5
print('optimal solution for original problem:',-(result.fun)-5)
#for the rest of the variables we will just subtract the respective limit since thats the
#transformation we applied earlier (Xi_new=Xi_old+LIMITi) to get the problem in canonical form
for i,flow in enumerate(names_1): #printing all new variables with their optimal values
    print(flow+'_old : ',result.x[i]-limits[i])

#printing a visual seperator
print('\n\n\n')
```

Και τα αποτελέσματα :

```
optimal solution for transformed problem: 8.999999947681017
sa_new :  4.999999975020198
ad_new :  1.999999984018387
ab_new :  1.9999999969530036
sb_new :  1.9999999895251037
be_new :  4.9999999805269235
sc_new :  1.9999999861113007
cd_new :  6.425698282526834
ce_new :  2.574301685730888
dt_new :  7.425698266545222
et_new :  1.5743016811357946



optimal solution for original problem: 3.999999947681017
sa_old :  1.9999999750201978
ad_old :  0.9999999840183871
ab_old :  0.9999999969530036
sb_old :  0.9999999895251037
be_old :  1.9999999805269235
sc_old :  0.9999999861113007
cd_old :  2.4256982825268336
ce_old :  -1.4256983142691122
dt_old :  3.425698266545222
et_old :  0.5743016811357946
```

Βλέπουμε τις ροές και για το μετασχηματισμένο πρόβλημα (τα new) καθώς και για το αρχικό (τα old) που μας ενδιαφέρει. **Τελικά η βέλτιστη ταχύτητα μεταφοράς που επιτυγχάνει το δίκτυο είναι σχεδόν 4Mbit/s**.

Παρακάτω παραθέτω documentation για το πακέτο επίλυσης που αξιοποίησα:

Parameters: **c** : ***1-D array***

The coefficients of the linear objective function to be minimized.

A_ub : ***2-D array, optional***

The inequality constraint matrix. Each row of `A_ub` specifies the coefficients of a linear inequality constraint on `x`.

b_ub : ***1-D array, optional***

The inequality constraint vector. Each element represents an upper bound on the corresponding value of `A_ub @ x`.

A_eq : ***2-D array, optional***

The equality constraint matrix. Each row of `A_eq` specifies the coefficients of a linear equality constraint on `x`.

b_eq : ***1-D array, optional***

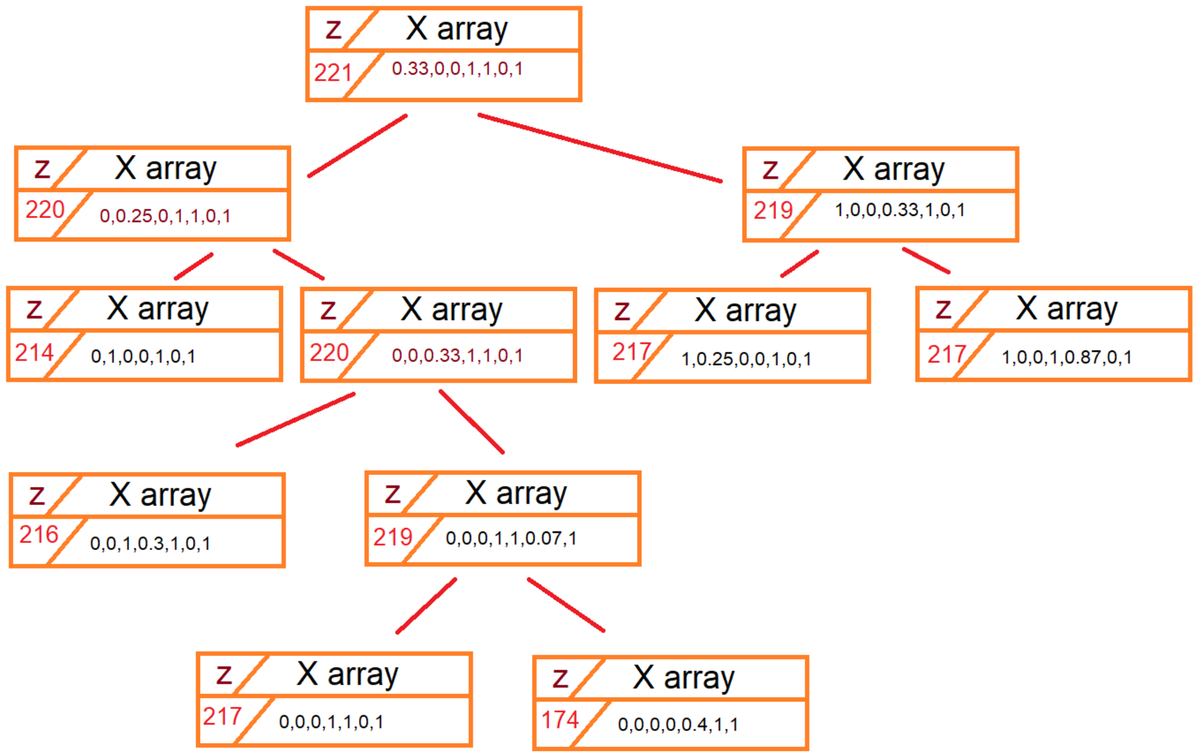
The equality constraint vector. Each element of `A_eq @ x` must equal the corresponding element of `b_eq`.

bounds : ***sequence, optional***

A sequence of `(min, max)` pairs for each element in `x`, defining the minimum and maximum values of that decision variable. Use `None` to indicate that there is no bound. By default, bounds are `(0, None)` (all decision variables are non-negative). If a single tuple `(min, max)` is provided, then `min` and `max` will serve as bounds for all decision variables.

https://docs.scipy.org/doc/scipy/reference/optimize.linprog-simplex.html

Άσκηση 3. (αρχείο **ask3_b.py**)



ergasia2 > ask3_b.py > ...

```python
from pulp import LpProblem , LpMaximize , LpStatus , LpVariable

model = LpProblem ( name='ex' , sense=LpMaximize )

x1 = LpVariable( name='x1'  , lowBound=0, upBound=1)
x2 = LpVariable( name='x2 ' , lowBound=0, upBound=1)
x3 = LpVariable( name='x3' , lowBound=0, upBound=1)
x4 = LpVariable( name='x4' , lowBound=0, upBound=1)
x5 = LpVariable( name='x5' , lowBound=0, upBound=1)
x6 = LpVariable( name='x6' , lowBound=0, upBound=1)
x7 = LpVariable( name='x7' , lowBound=0, upBound=1)
model += ( 3 * x1 + 4*x2 + 3*x3 + 3*x4 + 15* x5 + 13* x6 + 16* x7 <=35 )
model += ( x1 == 0 )
model += ( x2 == 0 )
model += ( x3 == 0 )
model += ( x6 == 0 )
model += 12 * x1 + 12 * x2 + 9*x3 + 15* x4 + 90* x5 + 26*x6 + 112* x7
status = model.solve()
print( f'status: {model.status} , {LpStatus[ model.status()]}' )
print( f'Z = {model.objective.value() }' )
for var in model.variables( ) :
    print( f' { var.name} = { var.value ( ) }' )


print( f' s t a t u s : {model.status } , {LpStatus[model.status] }' )
print( f' = {model.objective.value( ) } ')
```

Άσκηση 4.

Θα εκμεταλλευτώ τις ιδιότητες της σχέσης του δυϊκού προβλήματος με το αρχικό. Αρχικά κατασκευάζω το δυϊκό:

$$Z = 6y_1 + 4y_2 + 2y_3 + y_4$$

$$\begin{aligned}
&y_1 \geq 1\\
&y_1 + y_2 + y_3 + y_4 \geq 2\\
&y_3 \geq 1\\
&-y_1 - y_2 - y_4 \geq -3\\
&y_2 \geq 1\\
&2y_1 - 2y_2 + y_3 - y_4 \geq 1\\
&-2y_1 - 2y_2 - y_3 - y_4 \geq -1\\
&y_4 \geq 0
\end{aligned}$$

Αν η λύση $X=(7,0,2.5,0,3,0,0.5)$ είναι βέλτιστη τότε οι μεταβλητές χαλάρωσης των περιορισμών 1,3,5,7 θα είναι 0. Άρα θα ισχύουν οι ισότητες $y_1=1$, $y_3=1$, $y_2=1$ και $-2y_1+y_2 2-y_3+y_4=1 \Rightarrow y_4=0$.

Εντοπίζουμε πως για τη λύση που μας δώθηκε $Z_x=12$ αλλά και $Z_y=12$ άρα η λύση είναι βέλτιστη.

Άσκηση 5.

A)

Έστω ο πίνακας X που περιέχει τις μεταβλητές απόφασης για το ποιές επενδύσεις θα επιλέξουμε. Από τη περιγραφή του ποροβλήματος έχουμε πώς:

X3+X4 =< 1,
X5+X6 =< 1 ,
sum(X) =< 4 ,
sum(X) >= 2,
sum(.C * .X) =< Q,
Xj =< 1,
Xj >= 0.

Η συνθήκη ‘αναγκαιότητας’ ύπάρξης ενός από τα X3 ή X4 για την δέσμευση X5 ή X6 είναι κάπως πιό σύνθετη και όχι απαραίτητα προφανής. Μετά από σχετική αναζήτηση στη βιβλιογραφεία μοντελοποιώ αυτόν τον περιορισμό ως **X3+X4>=X5+X6 ή X3+X4-X5-X6>=0**. Αρκετά ενδιαφέρουσα τροποποίηση και με ύστερη γνώση κάπως αυτονόητη.

Η συνάρτηση που σκοπέυουμε να βελτιστοποιήσουμε είναι η
max Z = sum(.P*.X)

B)

Δίνω τυχαίες τιμές στα κόστη, το κεφάλαιο και τα κέρδη:

```
import random as rand
rand.seed(1066584)#setting random number generator to produce same values between runs
#print(rand.randint(0,10000))
C=[]
P=[]
X=[]
Q=20000
max_cost=5000
max_profit=15000
for i in range(0,10):
    C.append(rand.randint(0,max_cost))
    P.append(rand.randint(0,max_profit))
```

Καί έχουμε λύση:

```
COSTS:  [1317, 2038, 521, 2248, 4186, 371, 4888, 870, 191, 2265]
PROFITS:  [9866, 3613, 3341, 126, 1826, 3036, 10457, 7455, 12028, 9813]
INVESTMENTS DONE:
 x0 = 1.0
 x1 = 0.0
 x2 = 0.0
 x3 = 0.0
 x4 = 0.0
 x5 = 0.0
 x6 = 1.0
 x7 = 0.0
 x8 = 1.0
 x9 = 1.0
```

Κώδικας (**ask5.py**)

```
ergasia2 > ask5.py > ...
from pulp import LpProblem , LpMaximize , LpStatus , LpVariable
import random as rand
rand.seed(1066584)#setting random number generator to produce same values between runs
#print(rand.randint(0,10000))
C=[]
P=[]
X=[]
Q=20000
max_cost=5000
max_profit=15000
for i in range(0,10):
    C.append(rand.randint(0,max_cost))
    P.append(rand.randint(0,max_profit))


model = LpProblem ( name='ask5' , sense=LpMaximize )

for i in range(0,10):
    X.append(LpVariable( name='x{}'.format(i)  , lowBound=0, upBound=1))




model += ( X[2]+X[3] <= 1) #restrictions
model += ( X[4]+X[5] <= 1) #restrictions
model += ( X[0]+X[1]+X[2]+X[3]+X[4]+X[5]+X[6]+X[7]+X[8]+X[9] <= 4) #restrictions
model += ( X[0]+X[1]+X[2]+X[3]+X[4]+X[5]+X[6]+X[7]+X[8]+X[9] >= 2) #restrictions
model += ( X[0]*C[0]+X[1]*C[1]+X[2]*C[2]+X[3]*C[3]+X[4]*C[4]+X[5]*C[5]+X[6]*C[6]+X[7]*C[7]+X[8]*C[8]+X[9]*C[9] <= Q) #restrictions
model += ( X[2]+X[3]-X[4]-X[5] >= 0) #restrictions




model += X[0]*P[0]+X[1]*P[1]+X[2]*P[2]+X[3]*P[3]+X[4]*P[4]+X[5]*P[5]+X[6]*P[6]+X[7]*P[7]+X[8]*P[8]+X[9]*P[9]


status = model.solve()
print('COSTS: ',C)
print('PROFITS: ',P)
print('INVESTMENTS DONE:')
for var in model.variables( ) :
    print( f' { var.name} = { var.value ( ) }' )

```