# 1η Εργασία

## ‘Γραμμική και Συνδυαστική Βελτιστοποίηση’ 2021-2022


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

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


Σημείωση: Για την επεξεργασία των προβλημάτων αξιοποιήθηκε η γλώσσα Python σε περιβάλλον VScode. Η αναπαράσταση υλοποιήθηκε μέσω του πακέτου matplotlib.

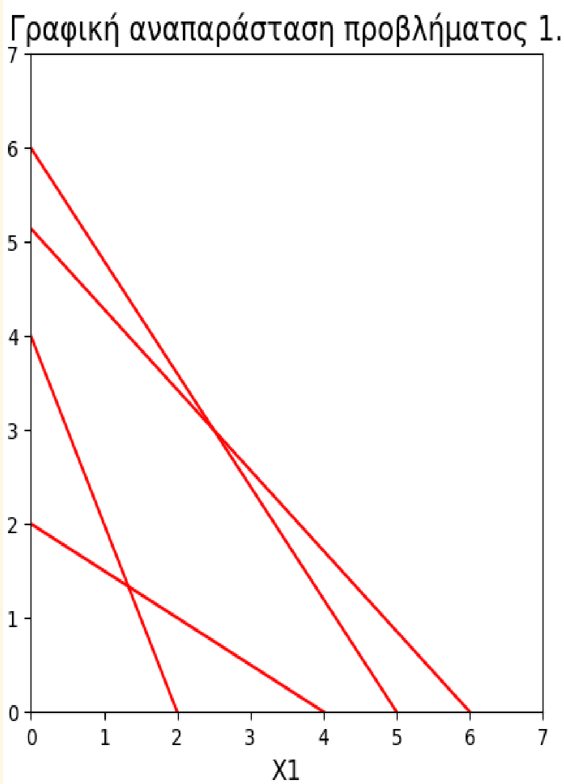
Οι κώδικες μπορούν να αντληθούν και απο το repository στο παρακάτω link: https://github.com/Skorpinakos/1st-sub-linprog

# Άσκηση 1.

**A)** Αρχικά εισάγουμε τους συντελεστές του συστήματος και δημιουργούμε τις εξισώσεις. Ύστερα αποτυπώνουμε τις εξισώσεις γραφικά :

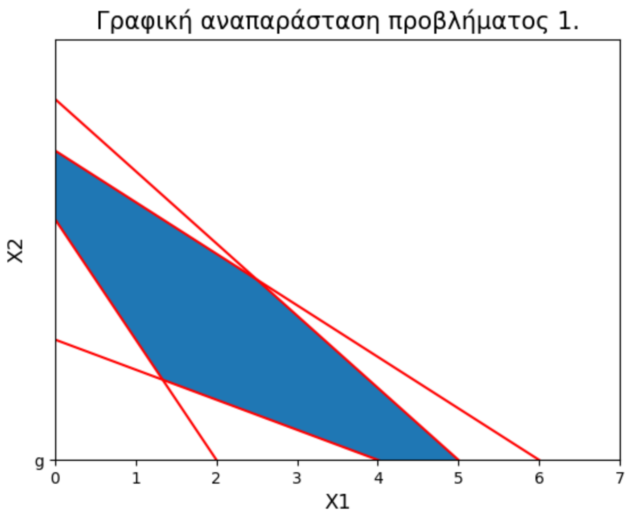
ask1.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt

#x->X1 y->X2

#set simulation limits
x_axis_range=7
y_axis_range=7
x=np.arange(x_axis_range)

#set system
system_order=4
x1_coef=[6,4,6,6]
x2_coef=[3,8,5,7]
b=[12,16,30,36]
y=[]

#create line equations
for i in range(0,4):
    y.append((  (b[i]/x2_coef[i])-(x1_coef[i]/x2_coef[i])*x    ))



#plot metadata
plt.title("Γραφική αναπαράσταση προβλήματος 1.",fontsize=15)
plt.xlabel("X1",fontsize=13)
plt.ylabel("X2",fontsize=13)
plt.ylim(0,y_axis_range)
plt.xlim(0,x_axis_range)

#plot lines
plt.plot(x,y[0],'r')
plt.plot(x,y[1],'r')
plt.plot(x,y[2],'r')
plt.plot(x,y[3],'r')

```



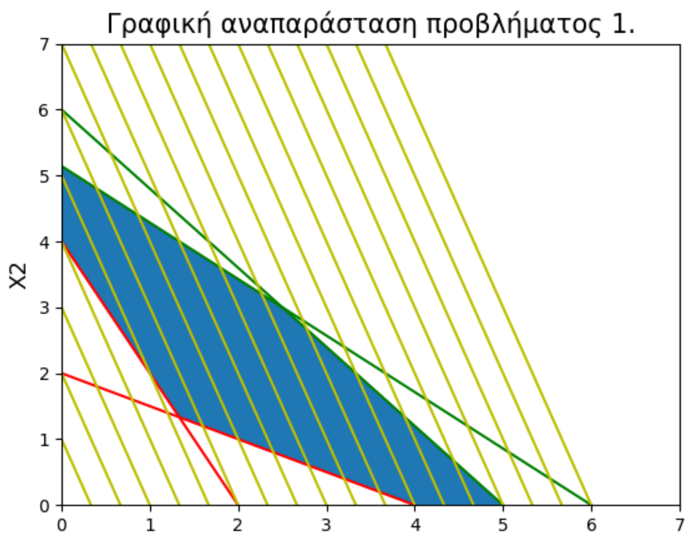

Ο κώδικας υπάρχει και επισυναπτόμενος ή στο αντίστοιχο repository μου ως ‘ask1.py’.

Έπιτα παρατηρούμε τα σημεία τομής των ευθειών μεταξύ τους (καθώς και με τους άξονες) και καταγράφουμε τις κορυφές του εφικτού πολύτοπου (λαμβάνοντας υπόψην την φορά των ανισώσεων) . Αξιοποιώντας τους αντίστοιχους πίνακες με τις X,Y συντεταγμένες των κορυφών σχεδιάζουμε τον εφικτό υποχώρο:




```
#nodes
points_x=[4,4/3,0,0,5/2,5,4]
points_y=[0,4/3,4,36/7,3,0,0]

#fill polygon inside nodes
plt.fill_between(points_x, points_y,)
```

Τέλος σχεδιάζουμε την οικογένεια υπερεπιπέδων της αντικειμενικής συνάρτησης **Z = 3*X1 + X2** :



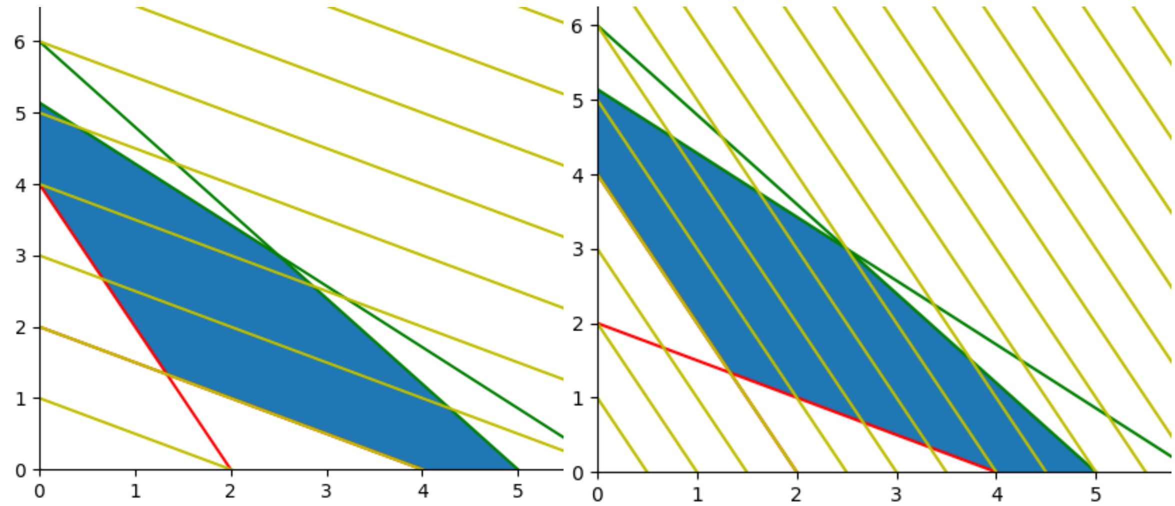

```
#plot value lines

for c in range(0,15):
    plt.plot(x,c-3*x,'y')
```

Μπορούμε πλεόν με γραφικό τρόπο να παρατηρήσουμε ότι η επιθυμητή κορυφή για την μεγιστοποίηση της αντικειμενικής συνάρτησης είναι η **(2.5 , 3)** .

**Β)** Η τομή των περιορισμών Π1,Π2 είναι η κορυφή **(4/3,4/3)** . Μεταβάλλοντας την παράμετρο C2 στην νέα εξίσωση της αντικειμενικής συνάρτησης ($Z = X1 + C2 \cdot X2$) αλλάζουμε την κλίση της οικογένειας ευθειών που αναπαριστούν την Α.Σ. Οι οριακές τιμές της κλίσης των ευθειών της Α.Σ. για να παραμένει ελάχιστο σημείο η κορυφή (4/3 , 4/3) είναι ίσες με αυτές των ευθειών που αντιπροσωπεύουν τους περιορισμούς Π1,Π2. Αν η Α.Σ. περιστραφεί δεξιόστροφα (αλγεβρική ελλάτωση της κλίσης) περισσότερο από τον Π2 η νέα βέλτιστη κορυφή θα είναι η (4,0) ενώ αν περιστραφεί αριστερόστροφα (αλγεβρική άυξηση της κλίσης) περισσότερο από τον Π1 η νέα ελάχιστη θέση θα βρίσκεται στην (0,4).

Συνεπώς θέλουμε $-6/3 < \alpha < -4/8 \Rightarrow -6/3 < -1/C2 < -4/8 \Rightarrow$ $\mathbf{0.5 < C2 < 2}$ .

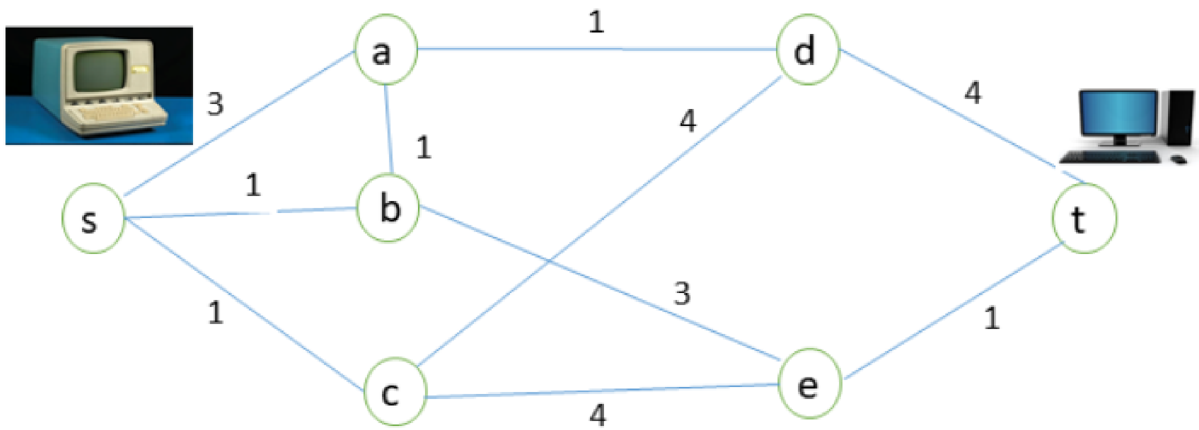
Γραφική αναπαράσταση ακραίων περιπτώσεων :




**Γ)** Αξιοποιώντας την ίδια λογική με το προηγούμενο ερώτημα επιθυμούμε τιμές για την κλίση της Α.Σ. μεταξύ των κλίσεων των δύο περιορισμών που δημιουργούν την θεμιτή κορυφή **(2.5 , 3)** .

Άρα $-6/5 < \alpha < -6/7 \Rightarrow -6/5 < -C1/C2 < -6/7 \Rightarrow$ $\mathbf{1.2 < C2/C1 < 0.857}$ .

# Άσκηση 2.




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 υποκειμενικές επιλογές 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 πλέον 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
```

Σχετικός κώδικας για την μετατροπή:

Υπάρχει και επισυναπτόμενος ή στο αντίστοιχο repository μου ως ‘ask2.py’.

```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)

```

(Κοιτώντας το ρολόι μου συνειδητοποιώ πως η χρήση python για τους υπολογισμούς της μετατροπής και την παρουσίαση του αποτελέσματος δεν ήταν αρκετά έξυπνη ιδέα μιας και με το χέρι θα χρειαζόμουν λιγότερο χρόνο, παραταύτα πιστεύω πως ήταν χρήσιμη απόπειρα μιας και άν είχαμε περισσότερους κόμβους η χρήση python θα ήταν πιο αποδωτική. Επίσης αν οι πληροφορίες του γράφου είχαν δωθεί σε άλλη μορφη (excel , csv κτλ.) θα υπήρχε η δυνατότητα δυναμικής εισαγωγής των δεδομένων.)

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.

Ο παρακάτω κώδικας μου έχει τις απαραίτητες εξηγήσεις σε σχόλια για το πως υλοποιώ όσα ανέφερα νωρίτερα στην python. Υπάρχει και επισυναπτόμενος ή στο αντίστοιχο repository μου ως ‘ask2.py’.

```python
36  #Here the real deal starts
37  #maximize using scipy
38
39  #A.S. matrix
40  c=np.array([0,0,0,0,0,0,0,0,-1,-1]) #normaly linprog optimizes for minimalization so we reverse
41  #the signs of C (we also change signs in the results) to maximize
42
43  #Inequality matrix coef
44  A_ub = np.zeros((10, 10), int) #creates empty 10x10 matrix #our inequality A matrix is basically a
45  #singular (I) 10x10 matrix so we fill the diagonal of the empty matrix with ones
46  np.fill_diagonal(A_ub, 1)  #numpy 1 d array of constant coefficients b for inequalities
47
48  #Inequality matrix const
49  flow_limits_new=[]
50  for i in flow_limits:
51      flow_limits_new.append(2*i) #we double the original limits array since our transformed problem inequalities are x=<2*old_limit
52  b_ub=np.array(flow_limits_new) #numpy 1 d array of constant coefficients b for inequalities
53
54  #equality matrix coef and const
55  names=names_1 #changing names because i copied the following code from my previous function
56  limits=flow_limits #changing names because i copied the following code from my previous function
57  constant_coef=[] #changing names because i copied the following code from my previous function
58  non_const_coef=[] #changing names because i copied the following code from my previous function
59  for eq in node_equations:
60      const=eq[-1]
61      temp_equation=[0,0,0,0,0,0,0,0,0,0]
62      for factor in eq:
63          if factor==0:
64              continue
65          if factor[0]=='-':
66              const=const-limits[names.index(factor[1:])]
67              temp_equation[names.index(factor[1:])]=-1
68          else:
69              const=const+limits[names.index(factor[1:])]
70              temp_equation[names.index(factor[1:])]=+1
71
72      constant_coef.append(const)
73      non_const_coef.append(temp_equation)
74      #using the node equation matrix from earlier i iterate each node
75      # equation and i append the respective coefficients to the 1d array of b constant coefficients and the 2d array of A coefficients
76
77
78  A_eq=np.array(non_const_coef)      #making the numpy array based on my 2d list
79  b_eq=np.array(constant_coef)       #making the numpy array based on my 1d list
80
81  #bounds
82  bounds=(0,None) #default #basically setting the range for our problem variables
83
84  #using all the matrixes we made we can now use the linprog optimizer (documentation here:
85  #  https://docs.scipy.org/doc/scipy/reference/optimize.linprog-simplex.html )
86  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
87  #optimizer (the default optimizer is the simplex method so i do not specify it)
88
89  #printing a visual seperator
90  print('\n\n\n')
91
92  print('optimal solution for transformed problem:',-(result.fun)) #changing signs since this is a maximazation proble, see line 39
93  for i,flow in enumerate(names_1): #printing all new variables with their optimal values
94      print(flow+'_new : ',result.x[i])
95
96  #printing a visual seperator
97  print('\n\n\n')
98  #we need to adjust our result to the original problem
99
100 #our A.S. was dt+et, after the transformation it becomes dt_new+et_new+4+1=> A.S._new=A.S._old+5
101 print('optimal solution for original problem:',-(result.fun)-5)
102 #for the rest of the variables we will just subtract the respective limit since thats the
103 #transformation we applied earlier (Xi_new=Xi_old+LIMITi) to get the problem in canonical form
104 for i,flow in enumerate(names_1): #printing all new variables with their optimal values
105     print(flow+'_old : ',result.x[i]-limits[i])
106
107 #printing a visual seperator
108 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.

Αρχικά ορίζουμε τις συμβάσεις με τις οποίες θα μοντελοποιήσουμε το πρόβλημα.

1) Επιπλέον των X1,X2,X3...X12 ορίζω και X0=0* για ευκολία.
2) Θεωρώ πως καμία παραγγελία δεν μένει ανεκπλήρωτη**.
3) Θέτω S0=0. Το S12 θεωρώ ότι είναι ελεύθερο και όχι απαραίτητα 0.

Η αντικειμενική μας συνάρτηση αποτελείται από τα έξοδα για την αποθήκευση και τη μεταβολή παραγωγικής ισχύος. Τα έξοδα μεταβολής μπορούν να περιγραφούν ώς το άθροισμα των απολύτων της διαφοράς της παραγωγής μεταξύ των μηνών επί 50, άρα από τον τύπο στην εικόνα 3.1, ενώ τα έξοδα αποθήκευσης από την παραγωγή του μήνα συν το αθροιστικό υπόλοιπο από την έως τώρα λειτουργεία μείον την ζήτηση του μήνα όπως φαίνεται στην εικόνα 3.2. Στην εικόνα 3.3 φαίνεται ολόκληρη η αντικειμενική συνάρτηση. Οι περιορισμοί προκύπτουν από τον τύπο στην εικόνα 3.4 ο οποίος αναπαριστά την σύμβαση 2.

*Εικόνα 3.1*

$$z_1 = \sum_{i=1}^{12} \left( |x_i - x_{i-1}| \right) \cdot 50$$

*Εικόνα 3.2*

$$z_2 = \sum_{i=1}^{12} \left( x_i - d_i + \sum_{n=1}^{i-1} (x_n - d_n) \right) \cdot 20$$

*Εικόνα 3.3*

$$z_{total} = \sum_{i=1}^{12} \left( 50 \cdot |x_i - x_{i-1}| + 20 \left[ x_i - d_i + \sum_{n=1}^{i-1} (x_n - d_n) \right] \right)$$

*Εικόνα 3.4*

$$\forall i \geq 1: x_i - d_i + \sum_{n=1}^{i-1} (x_n - d_n) \geq 0$$

*Η περιγραφή του προβλήματος δεν ξεκαθαρίζει τί παραγωγή έχουμε κατά την εκκίνηση οπότε θεωρώ 0, άλλες λογικές συμβάσεις θα ήταν να θεωρήσουμε X0=X1 (μηδενικό κόστος μεταβολής εκκίνησης) ή X0=avg(X) (μια πιο ρεαλιστική προσσέγγιση). Η μοντελοποίηση δεν διαφέρει ανάλογα της σύμβασης αυτής.

**Χωρίς αυτόν τον περιορισμό η βέλτιστη λύση θα ήταν X=[0,0,...0]*** για την ελαχιστοποίηση του κόστους. Αν είχαμε συντελεστή της τιμής/Kg θα μπορούσαμε να θεωρήσουμε συνάρτηση κέρδους και ο περιορισμός αυτός δεν θα ήταν απαραίτητος.

***Εκτός αν η ζήτηση ήταν σταθερή όποτε θα υπήρχαν άπειρες λύσεις.

Εμφανίζεται όμως ένα σοβαρό **πρόβλημα**, η **αντικειμενική συνάρτηση περιέχει έναν μή γραμμικό όρο**. Το Z1 εμπερειέχει τη συνάρτηση του **απολύτου**. Το πρόβλημα δηλαδή λαμβάνει τη **μορφή ελαχιστοποίησης Α.Σ. που περιγράφεται ως |a(x)|+b(y)**. Για το χειρισμό αυτής της ιδιαιτερότητας εισάγουμε μια **νέα μεταβλητή 'R'** και προσθέτουμε τους εξής περιορισμούς : **R =< a(x) και R >= -a(x)** (ή καλύτερα -R=<a(x) ) . Αυτοί οι περιορισμοί ισοδυναμούν με R=<|a(x)| άρα μπορούμε να αντικαταστήσουμε το απόλυτο στην Α.Σ. με τη νέα μεταβλητή χαλάρωσης R.

Στην περίπτωσή μας όμως ο όρος Z1 δέν είναι μία απλή συνάρτηση εντός απολύτου αλλά ένα άθροισμα απολύτων οπότε θα πρέπει να εισάγουμε **12 νέες μεταβλητές R1,R2,R3…R12** όπου **$R_i$=< $X_i$-$X_{(i-1)}$ και -$R_i$=< $X_i$-$X_{(i-1)}$ {με 13>i>0}** και ο όρος Z1 να γίνει όπως στην εικόνα 3.5 .

*Εικόνα 3.5*

$$Z_1 = 50 \cdot \sum_{i=1}^{12} (R_i)$$

Καταλήγουμε λοιπόν σε ένα πρόβλημα ελαχιστοποίησης με αντικειμενική συνάρτηση την $Z_{total}$ = Z1+Z2 (βλέπε εικόνες 3.5 και 3.2) η οποία είναι πλέον γραμμική, και δύο σετ περιορισμών, ένα σετ 12 περιορισμών που προκύπτουν από την τήρηση όλων των παραγγελιών (βλέπε εικόνα 3.4) και ένα ακόμα σετ 24 περιορισμών που προκύπτουν από τις 12 διπλές ανισότητες **$R_i$=< $X_i$-$X_{(i-1)}$ και -$R_i$=< $X_i$-$X_{(i-1)}$ {με 13>i>0}.**

Αν και ιδιαίτερα βολική αυτή η αντικατάσταση εισάγει μεταβλητές που δεν είναι φραγμένες στο θετικό χώρο, οπότε το πρόβλημα θα επιλυθεί είτε με τη χρήση 2 σειριακών μεθόδων simplex είτε αξιοποιώντας τη μεθοδολογία του μεγάλου Μ.

# Άσκηση 4.

4.1

Αντιπαράδειγμα:

Τα σημεία P1(4,4) και P2(4,-4) :

Έστω λ=0.5,

λ*P1+(1-λ)*P2=(2,2)+(2,-2)= (4,0) για το οποίο δεν ισχύει η σχέση x1=<(x2)^2

**Άρα μη κυρτό**

4.2

Αντιπαράδειγμα:

Τα σημεία P1(0,4) και P2(0,-4) :

λ*P1+(1-λ)*P2=(0,2)+(0,-2)= (0,0) για το οποίο δεν ισχύει η σχέση 3*(x1)^2+4*(x2)^2>=5

**Άρα μη κυρτό**

# Άσκηση 5.

Το πρόβλημα:

$$\max\ 4x_1 + 2x_2 + x_3 \quad \text{Α.Σ.}$$

$$\begin{aligned} x_1 &\le 5 & 1 \\ 4x_1 + x_2 &\le 25 & 2 \\ 8x_1 + 4x_2 + x_3 &\le 125 & 3 \\ x_1, x_2, x_3 &\ge 0 & 4 \end{aligned}$$

**A)**

Εισάγουμε 3 νέες μεταβλητές χαλάρωσης X4,X5,X6 για να μετατρέψουμε ανισώσεις τις σε εξισώσεις. Προκύπτουν οι ακόλουθοι δύο πίνακες numpy για τους συντελεστές των μεταβλητών και για τους στεθρούς όρους:

```
import numpy as np
A= np.array([1,0,0,1,0,0],
            [4,1,0,0,1,0]
            [8,4,1,0,0,1])
B= np.array([5,25,125])
```

Υπολογίζουμε όλους τους συνδιασμούς των 6 μεταβλητών ανα 3 και αφαιρούμε τις αντίστοιχες στήλες από τον αρχικό πίνακα 'A'. Λύνουμε τα επιμέρους 20 συστήματα καί βρίσκουμε πόσες βάσεις υπάρχουν.

Ο παρακάτω κώδικάς μου python υπολογίζει τη λύση του συστήματος για κάθε δυνατή μηδενική τριάδα:

ask5.py > solve_bases

```python
from msilib.schema import Error
import numpy as np
from itertools import combinations
A= np.array(([1,0,0,1,0,0],
            [4,1,0,0,1,0],
            [8,4,1,0,0,1]))
b= np.array([5,25,125])


def find_bases(x):
    bases=[]
    for set in combinations([0,1,2,3,4,5],3):
        x_del = np.delete(x,set[0],axis=1)
        x_del = np.delete(x_del,set[1]-1,axis=1)
        x_del = np.delete(x_del,set[2]-2,axis=1)
        bases.append([x_del,set])
    return bases

def print_bases(A):
    for base in find_bases(A):
        print(base[1])
        print(base[0])

def solve_bases(A,b):
    solutions=[]
    for base in find_bases(A):
        try:
            sol = np.linalg.solve(base[0], b)
        except Exception as e:
            sol='singular matrix'
        solutions.append([base[1],sol])
    return solutions

solved=solve_bases(A,b)

for result in solved:
    set=result[0]
    solution=result[1]
    if solution != 'singular matrix':
        for i,x in enumerate(set):
            print('x{}={}'.format(x+1,solution[i]))
    else:
        print('singular matrix error')
        for i,x in enumerate(set):
            print('x{}={}'.format(x+1,'N/A'))

    print('_________________________________________')
```

Βρίσκουμε λοιπόν τις βάσεις οι οποίες είναι τελικά 14 καθώς σε 6 από τους συνδιασμούς η λύση δεν ορίζεται:

```
x1=5.0
x2=25.0
x3=125.0
______________________
singular matrix error
x1=N/A
x2=N/A
x4=N/A
______________________
singular matrix error
x1=N/A
x2=N/A
x5=N/A
______________________
x1=125.0
x2=5.0
x6=25.0
______________________
singular matrix error
x1=N/A
x3=N/A
x4=N/A
______________________
x1=25.0
x3=5.0
x5=25.0
______________________
x1=31.25
x3=5.0
x6=-6.25
______________________
singular matrix error
x1=N/A
x4=N/A
x5=N/A
______________________
singular matrix error
x1=N/A
x4=N/A
x6=N/A
______________________
x1=25.0
x5=25.0
x6=5.0
```

```
x2=5.0
x3=5.0
x4=85.0
______________________
x2=6.25
x3=-1.25
x5=75.0
______________________
x2=15.625
x3=-10.625
x6=-37.5
______________________
singular matrix error
x2=N/A
x4=N/A
x5=N/A
______________________
x2=5.0
x4=85.0
x6=5.0
______________________
x2=6.25
x5=75.0
x6=-1.25
______________________
x3=5.0
x4=5.0
x5=65.0
______________________
x3=5.0
x4=21.25
x6=-16.25
______________________
x3=-3.125
x5=37.5
x6=8.125
______________________
x4=5.0
x5=5.0
x6=65.0
```

Δεν εμφανίζονται εκφυλλισμένες λύσεις.

**B)** Για τις κορυφές δημιουργώ το 3Χ6 σύστημα τεμνόμενων υπερεπιπέδων και αναζητώ λύσεις των ανα 3 εμπλεκόμενων εξισώσεων. Ύστερα εξετάζω αν οι λύσεις αυτές υπακούουν σε όλους τους περιορισμούς και τυπώνω True/False ανάλογα:

```
#find peaks
def feasibility_check(result):
    set=result[0]
    solution=result[1]
    full_form=[0,0,0,0,0,0]
    for i in range(len(solution)):
        full_form[set[i]]=solution[i]

    #the following should be made dynamically but for the situation
    #of this problem manual is ok

    #1st check
    for i in full_form:
        if i<0:
            return False
    #2nd check
    if full_form[0]>5:
        return False
    #3d check
    if 4*full_form[0]+full_form[1]>25:
        return False
    #4th check
    if 8*full_form[0]+4*full_form[1]+full_form[2]>125:
        return False
    return True

def find_hypersurfaces(A,b):
    hs=[]
    b=b.reshape(6,1)
    A=np.hstack([A,b])
    for set in combinations([0,1,2,3,4,5],3):
        x_del = np.delete(A,set[0],axis=0)
        x_del = np.delete(x_del,set[1]-1,axis=0)
        x_del = np.delete(x_del,set[2]-2,axis=0)
        hs.append([x_del[:,3],x_del[:,[0,1,2]],set]) #[b,A,set]
    return hs
def solve_hs(A,b):
    solutions=[]
    for comb in find_hypersurfaces(A,b):
        try:
            sol = np.linalg.solve(comb[1], comb[0])
        except Exception as e:
            sol='singular matrix'
        solutions.append([comb[2],sol])
    return solutions

print('############################################################################')

```

```python
print('############################################################################')

set=(3,4,5)

x_del = np.delete(A,set[0],axis=1)
x_del = np.delete(x_del,set[1]-1,axis=1)
A2 = np.delete(x_del,set[2]-2,axis=1)
#print(A2)

newrow = [1, 0, 0]
A2 = np.vstack([A2, newrow])
newrow = [0, 1, 0]
A2 = np.vstack([A2, newrow])
newrow = [0, 0, 1]
A2 = np.vstack([A2, newrow])
#print(A2)
b2= np.array([5,25,125,0,0,0])
solved=solve_hs(A2,b2)
for result in solved:
    set=result[0]
    solution=result[1]
    if solution != 'singular matrix':
        for i,x in enumerate(set):
            print('x{}={}'.format(x+1,solution[i]))
        print(feasibility_check(result))

    else:
        print('singular matrix error')
        for i,x in enumerate(set):
            print('x{}={}'.format(x+1,'N/A'))

    print('_________________________________________')
```

Και τα αποτελέσματα :

```
x1=0.0
x2=0.0
x3=0.0
True
_________________________
x1=15.625
x2=0.0
x3=0.0
False
_________________________
x1=0.0
x2=31.25
x3=0.0
False
_________________________
x1=0.0
x2=0.0
x3=125.0
True
_________________________
x1=6.25
x2=0.0
x3=0.0
False
_________________________
x1=0.0
x2=25.0
x3=0.0
True
_________________________
singular matrix error
x1=N/A
x2=N/A
x3=N/A
_________________________
x1=-3.125
x2=37.5
x3=0.0
False
_________________________
x1=6.25
x2=-0.0
x3=75.0
False
_________________________
x1=0.0
x2=25.0
x3=25.0
True
```

```
_________________________
x1=5.0
x2=0.0
x3=0.0
True
_________________________
singular matrix error
x1=N/A
x2=N/A
x3=N/A
_________________________
singular matrix error
x1=N/A
x2=N/A
x3=N/A
_________________________
x1=5.0
x2=21.25
x3=0.0
True
_________________________
x1=5.0
x2=0.0
x3=85.0
True
_________________________
singular matrix error
x1=N/A
x2=N/A
x3=N/A
_________________________
x1=5.0
x2=5.0
x3=0.0
True
_________________________
singular matrix error
x1=N/A
x2=N/A
x3=N/A
_________________________
singular matrix error
x1=N/A
x2=N/A
x3=N/A
_________________________
x1=5.0
x2=5.0
x3=65.0
True
```

Βλέπουμε ότι από τους **20 τριπλούς συνδιασμούς υπερεπιπέδων έχουμε 14 υπαρκτές κορυφές εκ των οποίων 9 είναι και εφικτές.**

**Γ)** Εφόσον δεν υπάρχουν εκφυλλισμένες κορυφές κάθε βάση αντιστοιχεί σε μία κορυφη και το αντίστροφο.

Για κάποιο λόγο η αντιστοίχηση ήταν εφικτή μόνο μεταξύ 9 κορυφών-βάσεων. Και δεν προχώρησα σε διόρθωση του (σοβαρού) σφάλματος.

```
relations=[]
for i in solved:
    if i[1]=='singular matrix':
        continue
    full=[0,0,0,0,0,0]
    for b,k in enumerate(i[0]):
        full[k]=i[1][b]
    for j in solved2:
        if j[1]=='singular matrix':
            continue
        full2=[0,0,0,0,0,0]

        for a,k in enumerate(j[0]):
            full2[k]=j[1][a]
        print(full)
        print(full2)
        print('==============================')
        if full[0:3]==full2[0:3]:
            relations.append([i[0],full2[0:3]])

print(len(relations))

for i in relations:
    print(i[0])
    print(i[1])
    print('____________________________________
```

```
9
(1, 3, 5)
[0, 5.0, 0.0]
____________________
(1, 3, 5)
[0, 5.0, 0]
____________________
(1, 3, 5)
[0, 5.0, 0]
____________________
(2, 3, 4)
[0, 0, 5.0]
____________________
(2, 3, 5)
[0, 0, 5.0]
____________________
(3, 4, 5)
[0.0, 0.0, 0.0]
____________________
(3, 4, 5)
[0.0, 0.0, 0]
____________________
(3, 4, 5)
[0.0, 0, 0]
____________________
(3, 4, 5)
[0, 0, 0]
____________________
```