ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ
ΜΗΧΑΝΙΚΩΝ ΚΑΙ
ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΓΡΑΜΜΙΚΗ ΚΑΙ ΣΥΝΔΥΑΣΤΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

ΕΡΓΑΣΙΑ 1η

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ - ΑΜ 1092581

ΥΠΕΥΘΥΝΗ ΚΑΘΗΓΗΤΡΙΑ

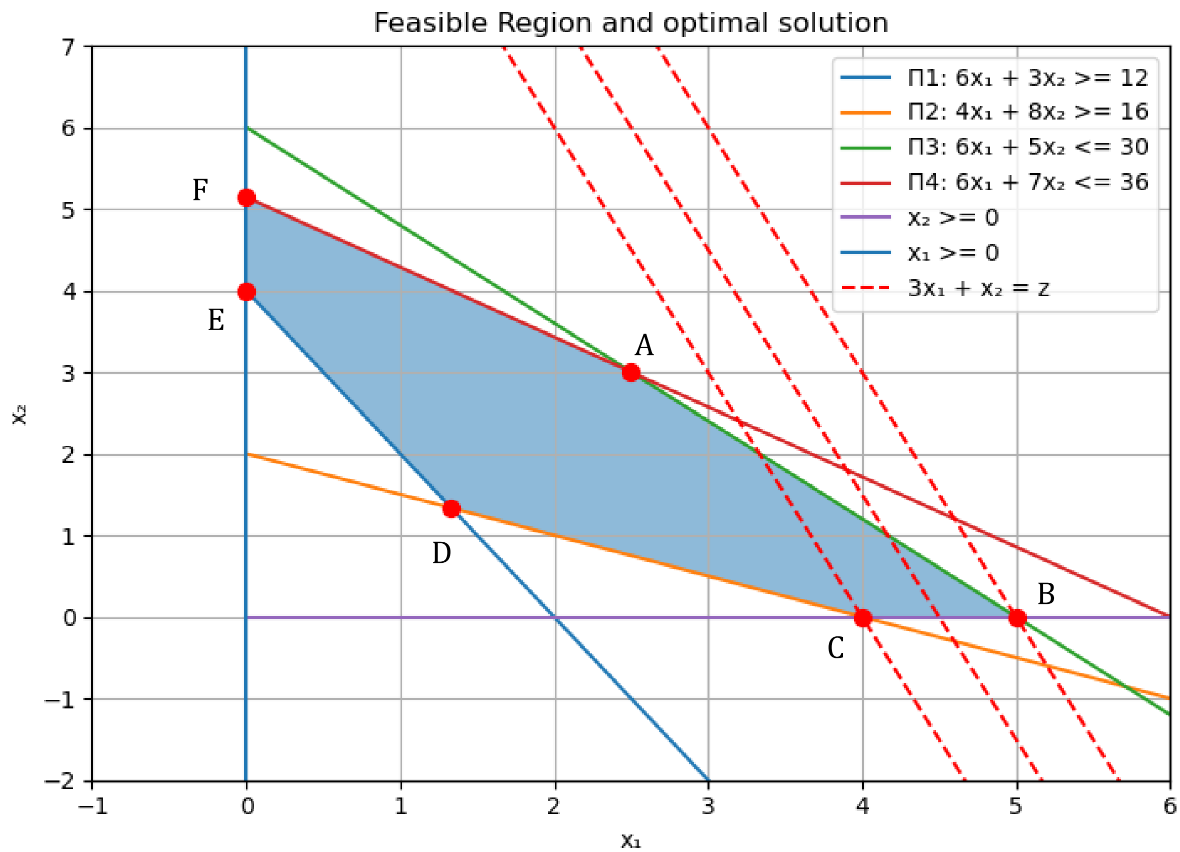
ΣΟΦΙΑ ΔΑΣΚΑΛΑΚΗ

ΡΙΟ

11 ΑΠΡΙΛΙΟΥ 2025

Άσκηση 1

(α) Η εφικτή περιοχή λύσεων αποτελείται από το κοινό τόπο των ημιχώρων που προκύπτει από τους 4 περιορισμούς και τους δύο περιορισμούς προσήμου. Παρακάτω φαίνεται το γραμμοσκιασμένο πολύτοπο F με κορυφές A, B, C, D, E, F.



Κορυφές: A(2.5,3) B(5,0) C(4,0) D(1.33,1.33) E(0,4) F(0,5.14)

Τα σημεία τομής των ευθειών που ορίζονται από τους περιορισμούς, βρίσκονται με την `np.linalg.solve(A,b)` η οποία λύνει το γραμμικό σύστημα εξισώσεων A**x**=b

Υπολογίζω την τιμή της αντικειμενικής συνάρτησης σε κάθε κορυφή της εφικτής περιοχής και η βέλτιστη είναι αυτή που την μεγιστοποιεί δηλαδή η B(5,0) με z*=15

Οι διακεκομμένες ευθείες παριστάνουν τις ισουψείς καμπύλες της αντικειμενικής συνάρτησης για z = 12, 13.5, 15. Το υπερεπίπεδο στήριξης $3x_1 + x_2 = 15$ του πολύτοπου F, εκφράζει την βέλτιστη λύση της αντικειμενικής συνάρτησης.

(β) Η κορυφή που ορίζεται από την τομή των περιορισμών Π1, Π2 είναι η $D(\frac{4}{3},\frac{4}{3})$. Η κλίση της ευθείας του Π1 είναι -2 και η κλίση της ευθείας του Π2 είναι $-\frac{1}{2}$. Η αντικειμενική συνάρτηση περνά από το σημείο D και έχει εξίσωση $x_2 = \min z - \frac{x_1}{c_2}$, άρα η κλίση της είναι μεταξύ -2 και $-\frac{1}{2}$.

$$-2 < -\frac{1}{c_2} <-\frac{1}{2} \ \Rightarrow \frac{1}{2} < c_2 <-2$$

(γ) Η κορυφή που ορίζεται από την τομή των περιορισμών Π3, Π4 είναι η A(2.5,3). Η κλίση της ευθείας του Π3 είναι $-\frac{6}{5}$ και η κλίση της ευθείας του Π4 είναι $-\frac{6}{7}$. Η αντικειμενική συνάρτηση περνά από το σημείο A και έχει εξίσωση $x_2 = max\, z - \frac{c_1 x_1}{c_2}$, άρα η κλίση της είναι μεταξύ $-\frac{6}{5}$ και $-\frac{6}{7}$.

$$-\frac{6}{5} < -\frac{c_1}{c_2} <-\frac{6}{7} \ \Rightarrow \frac{6}{7} < \frac{c_1}{c_2} <\frac{6}{5}$$

Κώδικας

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

x1=np.linspace(0,6,700) # 700 points between 0 and 6, x1>=0
#constraints
c0=0*x1 # x2>=0
c1=(12- 6*x1) / 3  # Π1: 6x1 + 3x2 >= 12
c2=(16-4*x1)/8  # Π2: 4x1 + 8x2 >= 16
c3= (30-6*x1)/5  # Π3: 6x1 + 5x2 >= 30
c4= (36-6*x1)/7  # Π4: 6x1 + 7x2 >= 36

# vertices of the feasible area
# Intersection of Π3 and Π4
A = np.linalg.solve([[6, 5], [6, 7]], [30, 36])

# Intersection of Π3 and x2=0
B = np.linalg.solve([[6, 5], [0, 1]], [30, 0])

# Intersection of Π2 and x2=0
C = np.linalg.solve([[4, 8], [0, 1]], [16, 0])

# Intersection of Π1 and Π2
D = np.linalg.solve([[6, 3], [4, 8]], [12, 16])

# Intersection of Π1 and x1=0
E = np.linalg.solve([[6, 3], [1, 0]], [12, 0])

# Intersection of Π4 and x1=0
F = np.linalg.solve([[6, 7], [1, 0]], [36, 0])

vertices = np.array([A, B, C, D, E, F])
vertices=np.round(vertices,3) #3 decimals
print(vertices)

z_values = 3*vertices[:, 0] + vertices[:, 1] #objective function in each vertice

optimal_index = np.argmax(z_values) #index of the max z_value
max_z=np.max(z_values)
```

```python
##contour lines of the objective function for z=15,12,13.5
z0 = max_z - 3*x1
z1 = 12- 3*x1
z2 = 13.5-3*x1
print(max_z)

optimal_vertex = vertices[optimal_index]
print(optimal_vertex)

# Plot the constraints
plt.plot(x1, c1, label='Π1: 6x₁ + 3x₂ >= 12')
plt.plot(x1, c2, label='Π2: 4x₁ + 8x₂ >= 16')
plt.plot(x1, c3, label='Π3: 6x₁ + 5x₂ <= 30')
plt.plot(x1, c4, label='Π4: 6x₁ + 7x₂ <= 36')
plt.plot(x1, c0, label='x₂ >= 0')
plt.axvline(x=0, label='x₁ >= 0')

plt.plot(x1, z0, 'r--', label='3x₁ + x₂ = z')
plt.plot(x1, z1, 'r--')
plt.plot(x1, z2, 'r--')

plt.grid(True)
plt.xlabel('x₁')
plt.ylabel('x₂')
plt.ylim(-2,7)
plt.xlim(-1,6)
plt.title('Feasible Region and optimal solution')

upper_bound= np.minimum(c3,c4) # in each x2 take the minimum of <= constraints
lower_bound= np.maximum(np.maximum(c1,c2),c0) # in each x2 take the maximum of >= constraints
# plot area between constraints
plt.fill_between(x1,lower_bound,upper_bound,where=(upper_bound>lower_bound) & (x1>0),alpha=0.5)
#plot vertices
plt.scatter(vertices[:,0],vertices[:,1],color='red',s=50,zorder=5)

#display labels
plt.legend()
# display the plot
plt.show()
```

Άσκηση 2

(α) Μοντελοποίηση του προβλήματος σχεδιασμού ραδιοθεραπείας

Μεταβλητές απόφασης:

x_1 : η ένταση της δέσμης ακτινοβολίας 1

x_2 : η ένταση της δέσμης ακτινοβολίας 2

Η αντικειμενική συνάρτηση εκφράζει το άθροισμα της συνολικής ακτινοβολίας των δύο δεσμών που απορροφά η υγιής περιοχή.

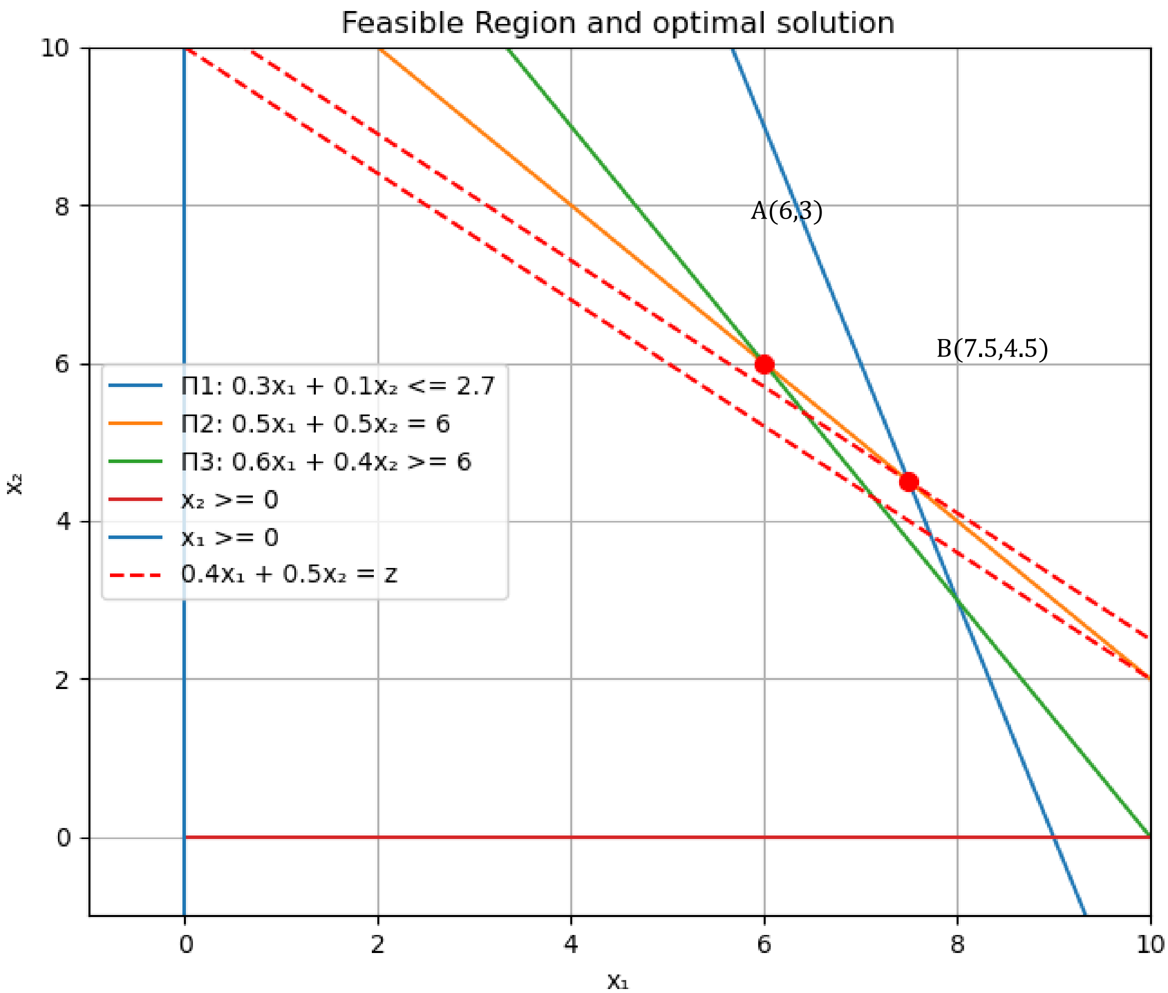
Ο στόχος είναι η ελαχιστοποίηση της αντικειμενικής συνάρτησης, ώστε να ικανοποιούνται όλοι οι περιορισμοί: $\min\ (0.4x_1 + 0.5x_2)$

$0.3x_1 + 0.1x_2 \leq 2.7$
$0.5x_1 + 0.5x_2 = 6$
$0.6x_1 + 0.4x_2 \geq 6$

$x_1, x_2 \geq 0$

(b) Γραφική επίλυση

Η εφικτή περιοχή του προβλήματος είναι ο κοινός τόπος των ημιχώρων που ορίζονται από τους 3 περιορισμούς και τους 2 περιορισμούς προσήμου. Λόγω της ισότητας το ευθύγραμμο τμήμα AB είναι η εφικτή περιοχή λύσεων.



Υπολογίζω την τιμή της αντικειμενικής συνάρτησης σε κάθε κορυφή της εφικτής περιοχής και η βέλτιστη είναι αυτή που την ελαχιστοποιεί δηλαδή η B(7.5,4.5) με z*=5.25.

Οι διακεκομμένες ευθείες παριστάνουν τις ισουψείς καμπύλες της αντικειμενικής συνάρτησης για z = 5, 5.25. Η ευθεία με z=5.25 τέμνει πρώτα το ακραίο σημείο B της εφικτής περιοχής μετακινούμενη προς τα πάνω, οπότε είναι η βέλτιστη κορυφή.

Τελικά η κάθε δέσμη έχει ένταση 7.5 και 4.5 kilorad αντίστοιχα ώστε να ελαχιστοποιείται η ακτινοβολία που απορροφά η υγιής περιοχή (5.25 kilorad) και να ικανοποιούνται οι περιορισμοί για την ακτινοβολία που απορροφάται και από τις άλλες περιοχές του σώματος.

Κώδικας

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

x1=np.linspace(0,10,700) # 700 points between 0 and 10, x1>=0

#constraints
c0=0*x1 # x2>=0
c1=(2.7- 0.3*x1) / 0.1  # Π1: 0.3x1 + 0.1x2 <= 2.7
c2=(6-0.5*x1)/0.5  # Π1: 0.3x1 + 0.1x2 = 6
c3= (6-0.6*x1)/0.4  # Π3: 0.6x1 + 0.4x2 >= 6

# vertices of the feasible area
# Intersection of Π1 and Π2
A = np.linalg.solve([[0.3, 0.1], [0.5, 0.5]], [2.7, 6])


# Intersection of Π2 and Π3
B = np.linalg.solve([[0.5, 0.5], [0.6, 0.4]], [6, 6])

vertices = np.array([A, B])
print(vertices)

z_values = 0.4*vertices[:, 0] + 0.5*vertices[:, 1]
optimal_index = np.argmin(z_values) #index of the minimum z_value
min_z=np.min(z_values)
## contour lines for z=5.25,5
z0=(min_z -0.4*x1)/0.5
z1=(5-0.4*x1)/0.5
print(min_z)
optimal_vertex = vertices[optimal_index]
print(optimal_vertex)
```

```
# Plot the constraints
plt.plot(x1, c1, label='Π1: 0.3x₁ + 0.1x₂ <= 2.7')
plt.plot(x1, c2, label='Π2: 0.5x₁ + 0.5x₂ = 6')
plt.plot(x1, c3, label='Π3: 0.6x₁ + 0.4x₂ >= 6')
plt.plot(x1, c0, label='x₂ >= 0')
plt.axvline(x=0, label='x₁ >= 0')
plt.plot(x1, z0, 'r--', label='0.4x₁ + 0.5x₂ = z')
plt.plot(x1, z1, 'r--')
plt.grid(True)
plt.xlabel('x₁')
plt.ylabel('x₂')
plt.xlim(-1,10)
plt.ylim(-1,10)
plt.title('Feasible Region and optimal solution')
plt.scatter(vertices[:,0],vertices[:,1],color='red',s=50,zorder=5)
#display labels
plt.legend()
# display the plot
plt.show()
```

Άσκηση 3

(α) Μοντελοποίηση του προβλήματος παραγωγής

Μεταβλητές απόφασης:

x_{ij} : ποσότητα της πρώτης ύλης i για τον τύπο ζωοτροφής j σε kg

Η αντικειμενική συνάρτηση εκφράζει το ολικό κόστος για τη συνολική ποσότητα όλων των πρώτων υλών που θα χρησιμοποιηθούν για την παραγωγή και των τριών τύπων ζωοτροφής.

Ο στόχος είναι η ελαχιστοποίηση του κόστους παραγωγής:

$$\min \{ 0.2(x_{11} + x_{12} + x_{13}) + 0.12(x_{21} + x_{22}+x_{23}) + 0.24(x_{31} + x_{32}+x_{33}) + 0.12(x_{41} + x_{42}+x_{43}) \}$$

Ο κάθε όρος αφορά το κόστος της ποσότητας της πρώτης ύλης i που θα χρησιμοποιηθεί και για τους τρεις τύπους ζωοτροφής.

Περιορισμοί Ι: Να μην υπερβούμε τη διαθεσιμότητα της κάθε πρώτης ύλης σε τόνους

$(x_{11} + x_{12} + x_{13})\, 10^3 \leq 9$	καλαμπόκι
$(x_{21} + x_{22} + x_{23})\, 10^3 \leq 12$	ασβεστόλιθος
$(x_{31} + x_{32} + x_{33})\, 10^3 \leq 5$	σόγια
$(x_{41} + x_{42} + x_{43})\, 10^3 \leq 6$	ιχθυάλευρο

Περιορισμοί 2: Να παραχθεί ακριβής ποσότητα της κάθε ζωοτροφής σε τόνους

$(x_{11} + x_{21} + x_{31} + x_{41})\ 10^3 = 12$ τύπος Ι
$(x_{12} + x_{22} + x_{32} + x_{42})\ 10^3 = 8$ τύπος ΙΙ
$(x_{13} + x_{23} + x_{33} + x_{43})\ 10^3 = 9$ τύπος ΙΙΙ

Περιορισμοί ΙΙΙ: Να καλύπτεται η απαιτούμενη ποσότητα σε θρεπτικά συστατικά για τον κάθε τύπο ζωοτροφής.

Τύπος Ι (12 τόνους):

$8x_{11} + 6x_{21} + 10x_{31} + 4x_{41} \geq 72\ 10^3$ βιταμίνες
$10x_{11} + 5x_{21} + 12x_{31} + 8x_{41} \geq 72\ 10^3$ πρωτείνες
$6x_{11} + 10x_{21} + 6x_{31} + 6x_{41} \geq 84\ 10^3$ ασβέστιο
$8x_{11} + 6x_{21} + 6x_{31} + 9x_{41} \leq 96\ 10^3$ λίπος
$8x_{11} + 6x_{21} + 6x_{31} + 9x_{41} \geq 48\ 10^3$ λίπος

Τύπος ΙΙ (8 τόνους):

$8x_{12} + 6x_{22} + 10x_{32} + 4x_{42} \geq 48\ 10^3$ βιταμίνες
$10x_{12} + 5x_{22} + 12x_{32} + 8x_{42} \geq 48\ 10^3$ πρωτείνες
$6x_{12} + 10x_{22} + 6x_{32} + 6x_{42} \geq 48\ 10^3$ ασβέστιο
$8x_{12} + 6x_{22} + 6x_{32} + 9x_{42} \leq 48\ 10^3$ λίπος
$8x_{12} + 6x_{22} + 6x_{32} + 9x_{42} \geq 32\ 10^3$ λίπος

Τύπος ΙΙΙ (9 τόνους):

$8x_{13} + 6x_{23} + 10x_{33} + 4x_{43} \leq 54\ 10^3$ βιταμίνες
$8x_{13} + 6x_{23} + 10x_{33} + 4x_{43} \geq 36\ 10^3$ βιταμίνες
$10x_{13} + 5x_{23} + 12x_{33} + 8x_{43} \geq 54\ 10^3$ πρωτείνες
$6x_{13} + 10x_{23} + 6x_{33} + 6x_{43} \geq 54\ 10^3$ ασβέστιο
$8x_{13} + 6x_{23} + 6x_{33} + 9x_{43} \leq 45\ 10^3$ λίπος
$8x_{13} + 6x_{23} + 6x_{33} + 9x_{43} \geq 36\ 10^3$ λίπος

Περιορισμοί προσήμου : $x_{ij} \geq 0$

Άσκηση 4

(a) Το σύνολο Χ περιγράφει το εξωτερικό ενός κύκλου με ακτίνα $\sqrt{3}$. Τα σημεία Α(-2,0) και Β(2,0) βρίσκονται στο σύνολο Χ και το ευθύγραμμο τμήμα που ορίζουν είναι το σύνολο των σημείων $x \in R^2$, $x = \lambda A + (1-\lambda)B$ με λ ∈ [0,1]. Για λ=0.5 το σημείο Γ(0,0) του ΑΒ δεν ανήκει στο σύνολο Χ. Άρα το σύνολο δεν είναι κυρτό.

(β) $X = \{(x_1, x_2, x_3) \in R^3 \mid x_1+2x_2 \leq 1, x_1-2x_3 \leq 2\}$

Έστω σημείο x (x_1, x_2, x_3) και y (y_1, y_2, y_3) και $x, y \in X$

Πρέπει νδο ένα σημείο $z\ (z_1, z_2, z_3) = \lambda x+(1-\lambda)y \in X\ για\ \forall\lambda \in[0,1]$ δηλαδή $z_1+2z_2\leq 1$ και $z_1-2z_3\leq 2$.

$z_1+2z_2=\lambda x_1+(1-\lambda)y_1+2\lambda x_2+2(1-\lambda)y_2= \lambda(x_1+2x_2)+(1-\lambda)(y_1+2y_2)\leq \lambda+(1-\lambda) \leq 1$

$z_1-2z_3=\lambda x_1+(1-\lambda)y_1- 2\lambda x_3-2(1-\lambda)y_3=\lambda(x_1-2x_3)+(1-\lambda)(y_1-2y_3)\leq 2(\lambda+1-\lambda) \leq 2$

Άρα το σύνολο X είναι κυρτό.

(γ) $X = \{(x_1, x_2, x_3) \in R^3 \mid x_2 \geq x_1^2, x_1+ 2x_2 +x_3 \leq 4\}$

Έστω σημείο x (x_1, x_2, x_3) και y (y_1, y_2, y_3) και $x, y \in X$

Πρέπει νδο ένα σημείο $z\ (z_1, z_2, z_3) = \lambda x+(1-\lambda)y \in X\ για\ \forall\lambda \in[0,1]$ δηλαδή $z_2 \geq z_1^2$και $z_1+ 2z_2 + z_3 \leq 4$

$z_2 = \lambda x_2 + (1-\lambda)y_2 \geq \lambda x_1^2 + (1-\lambda)y_1^2$

*** $f(x)=x^2\ \ με\ f''(x)=2>0\ \ άρα\ f\ κυρτή\ στο\ R$

Ανισότητα Jensen: $f(\lambda_1x_1+\cdots+\lambda_nx_n) \leq \lambda_1 f(x_1)+\cdots+\lambda_n f(x_n)$ $για\ κάθε\ κυρτή\ συνάρτηση\ f$.

Για $\lambda_1=\lambda$, $\lambda_2=1-\lambda$, $x_1=x_1$και $x_2=y_1$, έχουμε: $(\lambda_1x_1 + (1-\lambda)\ y_1)^2 \leq \lambda x_1^2 + (1-\lambda)y_1^2$

Άρα $z_2 = \lambda x_2 + (1-\lambda)y_2 \geq \lambda x_1^2 + (1-\lambda)y_1^2 \geq (\lambda_1x_1 + (1-\lambda)\ y_1)^2 \geq z_1^2$

$z_1+ 2\,z_2 + z_3 = \lambda x_1 + (1-\lambda)y_1 + 2\lambda x_2 + 2(1-\lambda)y_2 + \lambda x_3 + (1-\lambda)y_3$

$= \lambda(x_1+ 2x_2 +x_3) + (1-\lambda)\ (y_1+ 2\,y_2 + y_3) \leq 4\lambda +(1-\lambda)4 \leq 4$

Άρα το σύνολο X είναι κυρτό.

(δ) $X = \{(x_1, x_2, x_3) \in R^3 \mid x_3 = |x_2|, x_1 \leq 3\}$

Έστω σημείο x (x_1, x_2, x_3) και y (y_1, y_2, y_3) και $x, y \in X$

Πρέπει νδο ένα σημείο $z\ (z_1, z_2, z_3) = \lambda x+(1-\lambda)y \in X\ για\ \forall\lambda \in[0,1]$ δηλαδή $z_3 = |z_2|$ και $z_1 \leq 3$

Για $x = (1,-1,1) \in X$ και $y = (1,1,1) \in X$

$|z_2|=|-\lambda+(1-\lambda)1|=|1-2\lambda|$ και $z_3=\lambda+(1-\lambda)=1$

Έστω $\lambda=0.5 \Rightarrow |z_2|=0 \neq z_3$. Άρα το σύνολο δεν είναι κυρτό.

Άσκηση 5

(α)

$min\ Z = 8x_1+5x_2+4x_3$
$x_1 + x_2 \geq 10$ (0)
$x_3 + x_2 \geq 15$ (1)
$x_1 + x_3 \geq 12$ (2)
$20x_1 + 10x_2 + 15x_3 \leq 300$ (3)
$x_1, x_2, x_3 \geq 0$ (4), (5), (6)

Μια κορυφή στο R^3 ορίζεται από την τομή τριών υπερεπιπέδων. Εδώ έχουμε 7 υπερεπίπεδα που ορίζονται από τους 7 περιορισμούς, οπότε $\binom{7}{3} = \frac{7!}{3!4!} = 35$ πιθανές κορυφές.

Παρακάτω φαίνονται τα υπερεπίπεδα που δεν έχουν μοναδικό σημείο τομής (ιδιάζον πίνακας συντελεστών)

```
Hyperplanes(Constraint Indices) that don't form a vertex
(0, 4, 5)
(1, 5, 6)
(2, 4, 6)
```

Παρακάτω φαίνονται όλες οι κορυφές, καθώς και από ποια υπερεπίπεδα σχηματίζονται

```
All Vertices:
Vertex [x1, x2, x3]        Hyperplanes (Constraint Indices)
0      [3.5 6.5 8.5]       (0, 1, 2)
1      [ 5.  5. 10.]       (0, 1, 3)
2      [-0. 10.  5.]       (0, 1, 4)
3      [10. -0. 15.]       (0, 1, 5)
4      [-5. 15. -0.]       (0, 1, 6)
5      [-4. 14. 16.]       (0, 2, 3)
6      [-0. 10. 12.]       (0, 2, 4)
7      [10.  0.  2.]       (0, 2, 5)
8      [12. -2. -0.]       (0, 2, 6)
9      [-0.   10.   13.33] (0, 3, 4)
10     [10.   -0.    6.67] (0, 3, 5)
11     [ 20. -10.  -0.]    (0, 3, 6)
12     [-0. 10. -0.]       (0, 4, 5)
13     [10. -0. -0.]       (0, 4, 6)
14     [6. 9. 6.]          (0, 5, 6)
15     [-0.  3. 12.]       (1, 2, 3)
16     [-3. -0. 15.]       (1, 2, 4)
17     [12. 15. -0.]       (1, 2, 5)
18     [  0. -15.  30.]    (1, 2, 6)
19     [ 3.75 -0.   15.  ] (1, 3, 4)
20     [ 7.5 15.  -0. ]    (1, 3, 5)
21     [-0. -0. 15.]       (1, 3, 6)
22     [-0. 15. -0.]       (1, 4, 5)
23     [ 0. 12. 12.]       (1, 4, 6)
24     [ 24.  -0. -12.]    (1, 5, 6)
25     [12.  6. -0.]       (2, 3, 4)
26     [-0. -0. 12.]       (2, 3, 5)
27     [12. -0. -0.]       (2, 3, 6)
28     [ 0. -0. 20.]       (2, 4, 5)
29     [ 0. 30. -0.]       (2, 4, 6)
30     [15. -0. -0.]       (2, 5, 6)
31     [-0. -0. -0.]       (3, 4, 5)
```

Μία κορυφή ανήκει στο πολύτοπο των εφικτών λύσεων αν ικανοποιεί όλους τους περιορισμούς. Παρακάτω φαίνονται οι (6) εφικτές κορυφές και τα υπερεπεπίπεδα που τις σχηματίζουν.

```
Feasible Vertices:
Vertex [x1, x2, x3]         Hyperplanes (Constraint Indices)
0      [3.5 6.5 8.5]        (0, 1, 2)
1      [ 5.  5. 10.]        (0, 1, 3)
2      [-0. 10. 12.]        (0, 1, 4)
3      [-0.   10.   13.33]  (0, 1, 5)
4      [6. 9. 6.]           (0, 1, 6)
5      [ 0. 12. 12.]        (0, 2, 3)
```

Μία κορυφή είναι εκφυλισμένη αν σχηματίζεται με περισσότερα από 3 υπερεπίπεδα. Εδώ δεν υπάρχουν εκφυλισμένες κορυφές.

Κώδικας

```python
import numpy as np
from itertools import combinations
from scipy.optimize import linprog

#7 constraints in standard form (A x <= b)
A = np.array([
    [-1, -1,  0],  # x1 + x2 >= 10 -> -x1 -x2 <= -10
    [ 0, -1, -1],  # x2 + x3 >= 15 -> -x2 -x3 <= -15
    [-1,  0, -1],  # x1 + x3 >= 12 -> -x1 -x3 <= -12
    [20, 10, 15],  # 20x1 + 10x2 + 15x3 <= 300
    [-1,  0,  0],  # x1 >= 0 -> -x1 <= 0
    [ 0, -1,  0],  # x2 >= 0 -> -x2 <= 0
    [ 0,  0, -1]   # x3 >= 0 -> -x3 <= 0
])
b = np.array([-10, -15, -12, 300, 0, 0, 0])

# a vertex is formed by the intersection of three constraints
vertices = []
hyperplanes=[]
no_intersecting_hyperplanes=[]
feasible_vertices=[]
for indices in combinations(range(len(A)), 3): # all combinations of three constraints
    A_sub = A[list(indices)]
    b_sub = b[list(indices)]
    hyperplanes.append(indices)
    try:
        x = np.linalg.solve(A_sub, b_sub) # solve 3x3, x is the intersection
        vertices.append(x)
    except np.linalg.LinAlgError:
        no_intersecting_hyperplanes.append(indices)

vertices=np.round(vertices,decimals=2) #only 2 decimals in [x1, x2, x3]

#if a vertex is formed twice by different combination of hyperplanes added in degenerate_vertices
unique_vertices, counts = np.unique(vertices, axis=0, return_counts=True)
degenerate_vertices=unique_vertices[counts>1]
print(degenerate_vertices)
```

```python
print("Hyperplanes(Constraint Indices) that don't form a vertex")
for i in no_intersecting_hyperplanes: print(i)

print("\nAll Vertices:")
print(f"{'Vertex':<6} {'[x1, x2, x3]':<20} {'Hyperplanes (Constraint Indices)'}")

for i, (v, h) in enumerate(zip(vertices, hyperplanes)):
    print(f"{i:<6} {str(v):<20} {h}")
    if np.all(A @ v <= b ):  # vertex that satisfies all the constraints is feasible
        feasible_vertices.append(v)

print("\nFeasible Vertices:")
print(f"{'Vertex':<6} {'[x1, x2, x3]':<20} {'Hyperplanes (Constraint Indices)'}")
for i, (v, h) in enumerate(zip(feasible_vertices, hyperplanes)):
    print(f"{i:<6} {str(v):<20} {h}")
```

(β) Εισάγονται 4 μεταβλητές χαλάρωσης s_1, s_2, s_3, s_4, όσοι και οι περιορισμοί του προβλήματος. Τώρα έχουμε 7 μεταβλητές, οπότε μπορούν να σχηματιστούν $\binom{7}{4} = \frac{7!}{3!4!} = 35$ βασικοί πίνακες παίρνοντας διαφορετικό αριθμό στηλών του $[A\ I_4]$ κάθε φορά.

$min\ Z = 8x_1+5x_2+4x_3$
$x_1 + x_2 - s_1 = 10$ (0)
$x_3 + x_2 - s_2 = 15$ (1)
$x_1 + x_3 - s_3 = 12$ (2)
$20x_1 + 10x_2 + 15x_3 + s_4 = 300$ (3)
$x_1, x_2, x_3 \geq 0$ (4), (5), (6)
$s_1, s_2, s_3 \geq 0$

Επιλέγουμε 4 μεταβλητές ως βασικές x_B και οι υπόλοιπες είναι μη βασικές $x_N = 0$.

Βρίσκουμε την βασική λύση $x_B = B^{-1}b$. Ελέγχουμε αν είναι εφικτή ($x_B \geq 0$) και αν είναι εκφυλισμένη (δηλαδή να έχει βασική μεταβλητή που ισούται με το 0).

Παρακάτω φαίνονται όλες οι βασικές λύσεις (32). Τρεις βασικοί πίνακες από τους 35 δεν αντιστρέφονται, οπότε δεν δίνουν μοναδική λύση.

```
Solution Basic variables
0        x1=6.0    x2=9.0    x3=6.0    s1=5.0
1        x1=-4.0   x2=14.0   x3=16.0   s2=15.0
2        x1=5.0    x2=5.0    x3=10.0   s3=3.0
3        x1=3.5    x2=6.5    x3=8.5    s4=37.5
4        x1=12.0   x2=6.0    s1=8.0    s2=-9.0
5        x1=7.5    x2=15.0   s1=12.5   s3=-4.5
6        x1=12.0   x2=15.0   s1=17.0   s4=-90.0
7        x1=20.0   x2=-10.0  s2=-25.0  s3=8.0
8        x1=12.0   x2=-2.0   s2=-17.0  s4=80.0
9        x1=-5.0   x2=15.0   s3=-17.0  s4=250.0
10       x1=24.0   x3=-12.0  s1=14.0   s2=-27.0
11       x1=3.75   x3=15.0   s1=-6.25  s3=6.75
12       x1=-3.0   x3=15.0   s1=-13.0  s4=135.0
13       x1=10.0   x3=6.67   s2=-8.33  s3=4.67
14       x1=10.0   x3=2.0    s2=-13.0  s4=70.0
15       x1=10.0   x3=15.0   s3=13.0   s4=-125.0
16       x1=15.0   s1=5.0    s2=-15.0  s3=3.0
17       x1=12.0   s1=2.0    s2=-15.0  s4=60.0
18       x1=10.0   s2=-15.0  s3=-2.0   s4=100.0
19       x2=12.0   x3=12.0   s1=2.0    s2=9.0
20       x2=-15.0  x3=30.0   s1=-25.0  s3=18.0
21       x2=3.0    x3=12.0   s1=-7.0   s4=90.0
22       x2=10.0   x3=13.33  s2=8.33   s3=1.33
23       x2=10.0   x3=12.0   s2=7.0    s4=20.0
24       x2=10.0   x3=5.0    s3=-7.0   s4=125.0
25       x2=30.0   s1=20.0   s2=15.0   s3=-12.0
26       x2=15.0   s1=5.0    s3=-12.0  s4=150.0
27       x2=10.0   s2=-5.0   s3=-12.0  s4=200.0
28       x3=20.0   s1=-10.0  s2=5.0    s3=8.0
29       x3=12.0   s1=-10.0  s2=-3.0   s4=120.0
30       x3=15.0   s1=-10.0  s3=3.0    s4=75.0
31       s1=-10.0  s2=-15.0  s3=-12.0  s4=300.0
```

Παρακάτω φαίνονται οι (6) βασικές εφικτές ($x_B \geq 0$) λύσεις του συστήματος $B\boldsymbol{x} = b$. Εδώ δεν υπάρχουν εκφυλισμένες λύσεις.

```
Feasible Solutions
Solution Basic variables x_basic>=0 (x_N = 0)
0        x1=6.0    x2=9.0    x3=6.0    s1=5.0
2        x1=5.0    x2=5.0    x3=10.0   s3=3.0
3        x1=3.5    x2=6.5    x3=8.5    s4=37.5
19       x2=12.0   x3=12.0   s1=2.0    s2=9.0
22       x2=10.0   x3=13.33  s2=8.33   s3=1.33
23       x2=10.0   x3=12.0   s2=7.0    s4=20.0
```

Κώδικας

```python
import numpy as np
from itertools import combinations

#the constraints with slack variables [A I]
A = np.array([
    [ 1,  1,  0,  -1,  0,  0,  0],  # x1 + x2 - s1 = 10
    [ 0,  1,  1,  0,  -1,  0,  0],  # x2 + x3 - s2 = 15
    [ 1,  0,  1,  0,  0,  -1,  0],  # x1 + x3 - s3 = 12
    [20, 10, 15,  0,  0,  0,  1],  # 20x1 + 10x2 + 15x3 + s4 = 300

])

b = np.array([10, 15, 12, 300])
solutions = []
feasible_solutions=[]
num_vars=7
degenerate_solutions=[]
for indices in combinations(range(7), 4): #combinations of 4 columns (out of 7 columns) for basic matrix

    B = A[:, list(indices)]  # select columns corresponding to the basic variables
    b_sub = b
    try:
        # Solve the system  B* x_basic = b
        x_basic = np.linalg.solve(B, b_sub)

        x_full = np.zeros(num_vars)
        x_full[list(indices)] = x_basic #[x_basic, x_N]

        var_names = ['x1', 'x2', 'x3', 's1', 's2', 's3', 's4']
        solution = {var_names[i]: round(x_full[i], 2) for i in range(num_vars)} # dictionary {var_name1:value, var_name2:value, var_name3:value..}
        #{var_name1:value, var_name2:value, var_name3:value.., 'basic_vars':[var_name1,var_name2,...]}
        solution['basic_vars'] = [var_names[i] for i in indices]
        solutions.append(solution)

    except np.linalg.LinAlgError: ## linear dependent columns, B not invertible
        continue
```

```python
print("Solution Basic variables x_basic (x_N = 0)")
for i, sol in enumerate(solutions):
    basic_vars = sol['basic_vars']
    basic_values = [f"{var}={sol[var]:<6}" for var in basic_vars]
    print(f"{i:<8}  {' '.join(basic_values)}")
    if all(val >= 0 for var, val in sol.items() if var != 'basic_vars'): ## feasible solution if x_basic >=0
        feasible_solutions.append(sol)

    if any(val == 0 for val in x_basic): ## degenerate solution basic_variable=0
        degenerate_solutions.append(sol)

print("Feasible Solutions")
print("Solution Basic variables x_basic>=0 (x_N = 0)")
for i, sol in enumerate(feasible_solutions):
    basic_pairs = [f"{var}={sol[var]:<6}" for var in sol['basic_vars']]
    print(f"{i:<8}  {' '.join(basic_pairs)}")

print(degenerate_solutions)
```

(γ) Κάθε βασική εφικτή λύση του συστήματος αντιστοιχεί σε μία ακριβώς κορυφή του πολύτοπου που αναπαριστά την εφικτή περιοχή και αντιστρόφως κάθε κορυφή αντιστοιχεί σε τουλάχιστον μία βασική εφικτή λύση (αν η κορυφή είναι μη εκφυλισμένη σε ακρίβως μία)

```
Basic solution #0 matches vertex #14: [6. 9. 6.]
Basic solution #1 matches vertex #5: [-4. 14. 16.]
Basic solution #2 matches vertex #1: [ 5.  5. 10.]
Basic solution #3 matches vertex #0: [3.5 6.5 8.5]
Basic solution #4 matches vertex #25: [12.  6. -0.]
Basic solution #5 matches vertex #20: [ 7.5 15.  -0. ]
Basic solution #6 matches vertex #17: [12. 15. -0.]
Basic solution #7 matches vertex #11: [ 20. -10.  -0.]
Basic solution #8 matches vertex #8: [12. -2. -0.]
Basic solution #9 matches vertex #4: [-5. 15. -0.]
Basic solution #10 matches vertex #24: [ 24.  -0. -12.]
Basic solution #11 matches vertex #19: [ 3.75 -0.   15.  ]
Basic solution #12 matches vertex #16: [-3. -0. 15.]
Basic solution #13 matches vertex #10: [10.   -0.    6.67]
Basic solution #14 matches vertex #7: [10.  0.  2.]
Basic solution #15 matches vertex #3: [10. -0. 15.]
Basic solution #16 matches vertex #30: [15. -0. -0.]
Basic solution #17 matches vertex #27: [12. -0. -0.]
Basic solution #18 matches vertex #13: [10. -0. -0.]
Basic solution #19 matches vertex #23: [ 0. 12. 12.]
Basic solution #20 matches vertex #18: [  0. -15.  30.]
Basic solution #21 matches vertex #15: [-0.  3. 12.]
Basic solution #20 matches vertex #18: [  0. -15.  30.]
Basic solution #20 matches vertex #18: [  0. -15.  30.]
Basic solution #21 matches vertex #15: [-0.  3. 12.]
Basic solution #22 matches vertex #9: [-0.   10.   13.33]
Basic solution #23 matches vertex #6: [-0. 10. 12.]
Basic solution #24 matches vertex #2: [-0. 10.  5.]
Basic solution #25 matches vertex #29: [ 0. 30. -0.]
Basic solution #26 matches vertex #22: [-0. 15. -0.]
Basic solution #27 matches vertex #12: [-0. 10. -0.]
Basic solution #28 matches vertex #28: [ 0. -0. 20.]
Basic solution #29 matches vertex #26: [-0. -0. 12.]
Basic solution #30 matches vertex #21: [-0. -0. 15.]
Basic solution #31 matches vertex #31: [-0. -0. -0.]
```

Υπολογίζω την τιμή της αντικειμενικής συνάρτησης σε κάθε κορυφή, βέλτιστη είναι αυτή που την ελαχιστοποιεί. Δηλαδή η κορυφή #0: $[x_1, x_2, x_3] = [3.5, 6.5, 8.5]$ που αντιστοιχεί στην βασική λύση #3: $[x_1, x_2, x_3, s_4] = [3.5, 6.5, 8.5, 37.5]$, όπου η αντικειμενική συνάρτηση έχει την ελάχιστη τιμή της $z^* = 94.5$

```
Optimal z_value 94.5
Optimal vertex [3.5 6.5 8.5]
```

Κώδικας

```python
from ex5a import vertices,feasible_vertices
from ex5b import solutions, feasible_solutions
import numpy as np

for i, sol in enumerate(solutions):
    #extract x1, x2, x3 from the solution dictionary
    x1 = sol['x1']
    x2 = sol['x2']
    x3 = sol['x3']
    current_solution = np.array([x1, x2, x3])
    for j, vertex in enumerate(vertices):
        if (current_solution==vertex).all():
            print(f"Basic solution #{i} matches vertex #{j}: {vertex}")
feasible_vertices=np.array(feasible_vertices)

z_values = 8*feasible_vertices[:, 0] + 5*feasible_vertices[:, 1] + 4*feasible_vertices[:, 2]#objective function in each vertex
optimal_index = np.argmin(z_values) #index of the max z_value
min_z=np.min(z_values)

print(f"Optimal z_value {min_z}")
optimal_vertex = vertices[optimal_index]
print(f"Optimal vertex {optimal_vertex}")
```

Άσκηση 6

(α)

$\max Z = 2x_1+x_2+6x_3 - 4x_4$
$x_1 + 2x_2 + 4x_3 - x_4 \leq 6$
$2x_1 + 3x_2 - x_3 + x_4 \leq 12$
$x_1 + x_3 + x_4 \leq 2$
$x_1, x_2, x_3, x_4 \geq 0$

Εισάγονται τόσες μεταβλητές χαλάρωσης όσοι και οι περιορισμοί s_1, s_2, s_3 και δημιουργείται ο πρώτος βασικός πίνακας $B=I_3$, όποτε έχουμε την πρώτη βασική λύση για την εκκίνηση του αλγορίθμου. Δηλαδή $\mathbf{x}=0$, $\mathbf{x_s}=b$ και αυτή η λύση ικανοποιεί όλους τους περιορισμούς οπότε είναι εφικτή και μπορεί να ξεκινήσει ο αλγόριθμος.

Παρακάτω φαίνεται το αρχικό tableau με βασικές μεταβλητές τις slack variables και Z=0

```
Iteration 0:
        x1      x2      x3      x4      s1      s2      s3      b
-Z      2.00    1.00    6.00    -4.00   0.00    0.00    0.00    0.00
s1      1.00    2.00    4.00    -1.00   1.00    0.00    0.00    6.00
s2      2.00    3.00    -1.00   1.00    0.00    1.00    0.00    12.00
s3      1.00    0.00    1.00    1.00    0.00    0.00    1.00    2.00
```

Επιλέγουμε ως εισερχόμενη μεταβλητή στην βάση αυτή με τον μεγαλύτερο θετικό αντικειμενικό συντελεστή (6) δηλαδή την x_3. Αν δεν υπαρχουν θετικοί αντικειμενικοί συντελεστές έχουμε φτάσει σε βέλτιστη λύση και ο αλγόριθμος σταματάει

Επιλέγουμε ως εξερχόμενη μεταβλητή με βάση το κριτήριο ελαχίστου λόγου την s_1

$\min\{\frac{6}{4}, -, \frac{2}{1}\} = \frac{6}{4}$. Αν δεν υπάρχουν θετικες τιμές στην στήλη της εισερχόμενης μεταβλητής το πρόβλημα είναι μη φραγμένο, άρα ο αλγόριθμος σταματάει.

Εδώ το οδηγό στοιχείο είναι το [leaving_row,entering_col] = 4.00 και εφαρμόζουμε απαλοιφή Gauss, ώστε να γίνει 1 και όλα τα υπόλοιπα στοιχεία της εισερχόμενης στήλης να μηδενιστούν.

Παρακάτω φαίνεται το tableau που προκύπτει μετά το πρώτο βήμα του αλγορίθμου

```
Iteration 1:
        x1      x2      x3      x4      s1      s2      s3       b
-Z      0.50    -2.00   0.00    -2.50   -1.50   0.00    0.00    -9.00
x3      0.25    0.50    1.00    -0.25   0.25    0.00    0.00    1.50
s2      2.25    3.50    0.00    0.75    0.25    1.00    0.00    13.50
s3      0.75    -0.50   0.00    1.25    -0.25   0.00    1.00    0.50
```

$Z = 9$, $x_B = [x_3, s_2, s_3] = [\ 1.50, 13.50, 0.50]$, $[x_1, x_2, x_3] = [0,0,1.50]$

Εισερχόμενη μεταβλητή x_1

Εξερχόμενη μεταβλητή s_3

Με κάθε επανάληψη του αλγορίθμου μετακινούμαστε σε μία γειτονική κορυφή της εφικτής περιοχής. Συνεχίζονται οι επαναλήψεις μέχρι να συμβεί κάποιο κριτήριο σταματήματος.

```
Iteration 2:
        x1      x2      x3      x4      s1      s2      s3       b
-Z      0.00    -1.67   0.00    -3.33   -1.33   0.00    -0.67   -9.33
x3      0.00    0.67    1.00    -0.67   0.33    0.00    -0.33   1.33
s2      0.00    5.00    0.00    -3.00   1.00    1.00    -3.00   12.00
x1      1.00    -0.67   0.00    1.67    -0.33   0.00    1.33    0.67
```

$Z^* = 9.33$, $x_B = [x_3, s_2, x_1] = [\ 1.33, 12, 0.67]$, $[x_1, x_2, x_3] = [0.67,0,1.33]$ βέλτιστη κορυφή

Δεν υπάρχουν θετικοί αντικειμενικοί συντελεστές άρα η z δεν μπορεί να βελτιωθεί περισσότερο και ο αλγόριθμος σταματάει.

Κώδικας

```python
import numpy as np

# Problem setup
c = np.array([2, 1, 6, -4])  # Objective coefficients (maximize)
A = np.array([
    [1, 2, 4, -1],   # x1 + 2x2 + 4x3 − x4 ≤ 6
    [2, 3, -1, 1],    # 2x1 + 3x2 − x3 + x4 ≤ 12
    [1, 0, 1, 1]      # x1 + x3 + x4 ≤ 2
])
b = np.array([6, 12, 2])

num_constraints, num_vars = A.shape
# tableau structure rows= 3 constraints and the objective,col= 4 decision var + 3 slack vars + b
tableau = np.zeros((num_constraints + 1, num_vars + num_constraints + 1))

# Initialize tableau
tableau[:-1, :num_vars] = A # all rows except from the objective and cols of decision vars
tableau[:-1, num_vars:num_vars+num_constraints] = np.eye(num_constraints)  # identity matrix (cols: slack vars)
tableau[:-1, -1] = b # last column b
tableau[-1, :num_vars] = c  # last row objective funct (c coefficients)

# variable labels [x1, x2, x3, x4, s1, s2, s3 ,b]
var_labels = ['x1', 'x2', 'x3', 'x4', 's1', 's2', 's3',' b']
basic_vars=[4,5,6] # initial basis slack variables

def print_tableau(tableau, iteration, basic_vars):
    print(f"\nIteration {iteration}:")
    #print header
    print("\t" + "\t".join(var_labels))

    #print z_row
    z_row = tableau[-1, :]
    print(f"-Z\t" + "\t".join(f"{val:.2f}" for val in z_row))

    #print basis rows
    for row in range(num_constraints):
        basis_col = basic_vars[row]
        print(f"{var_labels[basis_col]}\t" + "\t".join(f"{val:.2f}" for val in tableau[row]))

```

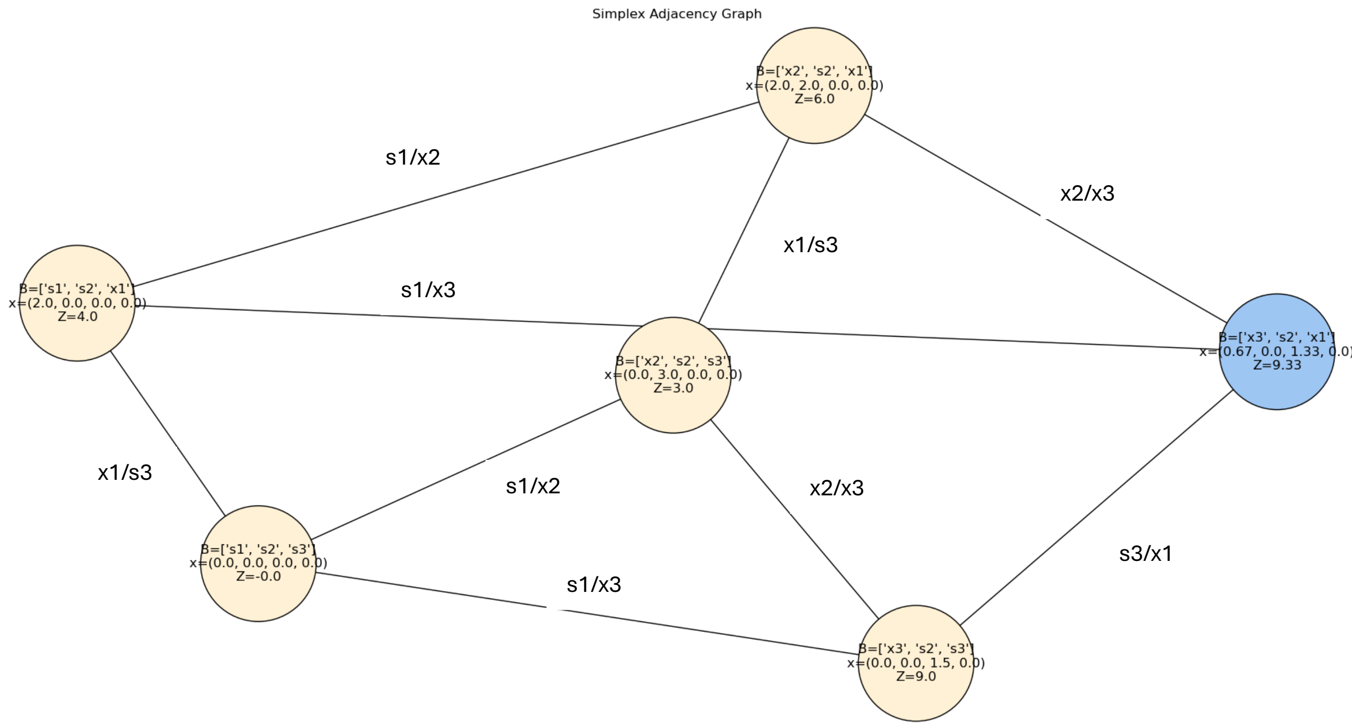
```python
def simplex(tableau):
    iteration = 0
    print_tableau(tableau, iteration, basic_vars)
    while True:
        iteration += 1
        # check for optimality if there are not any non positive c coefficients stop
        if np.all(tableau[-1, :-1] <= 0): #last row
            break

        # select entering variable most positive in objective row
        entering_col = np.argmax(tableau[-1, :-1])

        # check for unbounded problem
        # pivot  column must have at least one value>0 for the minimum ratio test
        if np.all(tableau[:-1, entering_col] <= 0):
            raise Exception("Problem is unbounded")

        # select leaving variable (minimum ratio test)
        ratios = []
        for i in range(num_constraints):
            if tableau[i, entering_col] > 0: #positive coefficients in the entering column
                # b_i / entering column coefficient i
                ratios.append(tableau[i, -1] / tableau[i, entering_col])
            else:
                ratios.append(np.inf) #else ignore ratio
        leaving_row = np.argmin(ratios)
        # pivot element
        pivot = tableau[leaving_row, entering_col]
        basic_vars[leaving_row] = entering_col
        tableau[leaving_row, :] /= pivot # 1 in pivot element
        for i in range(num_constraints + 1):
            if i != leaving_row:
                #zeros in all the rows of entering column except pivot
                tableau[i, :] -= tableau[i, entering_col] * tableau[leaving_row, :]
        # print current tableau
        print_tableau(tableau, iteration, basic_vars)
    return tableau
# run simplex algorithm
simplex(tableau)
```

(b) Στα βήματα της Simplex η επιλογή της εισερχόμενης μεταβλητής είναι αυθαίρετη αρκεί ο αντικειμενικός υσντελεστής να είναι θετικός. Επίσης η επιλογή της εξερχόμενης μεταβλητής σε περίπτωση ισοβαθμίας στο κριτήριο του ελάχιστο λόγου είναι αυθαίρετη. ΟΙ διαφορετικές επιλογές οδηγούν σε εναλλακτικές διαδρομές προς την βέλτιστη λύση. Αυτές φαίνονται στο Adjacency Graph.



Κώδικας

```python
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
# Problem setup
c = np.array([2, 1, 6, -4])  # Objective coefficients (maximize)
A = np.rray([
    [1, 2, 4, -1],   # x1 + 2x2 + 4x3 − x4 ≤ 6
    [2, 3, -1, 1],    # 2x1 + 3x2 − x3 + x4 ≤ 12
    [1, 0, 1, 1]      # x1 + x3 + x4 ≤ 2
])
b = np.array([6, 12, 2])

num_constraints, num_vars = A.shape
# tableau structure rows= 3 constraints and the objective,col= 4 decision var + 3 slack vars + b
tableau = np.zeros((num_constraints + 1, num_vars + num_constraints + 1))

# Initialize tableau
tableau[:-1, :num_vars] = A # all rows except from the objective and cols of decision vars
tableau[:-1, num_vars:num_vars+num_constraints] = np.eye(num_constraints)  # identity matrix (cols: slack vars)
tableau[:-1, -1] = b # last column b
tableau[-1, :num_vars] = c  # last row objective funct (c coefficients)

# variable labels [x1, x2, x3, x4, s1, s2, s3 ,b]
var_labels = ['x1', 'x2', 'x3', 'x4', 's1', 's2', 's3',' b']
basic_vars=[4,5,6] # initial basis slack variables


G = nx.DiGraph() #empty directed graph
visited = set()  #visited basis


#key for each state (basic_vars, b_values, z)
def tableau_key(basis, tableau):
    return tuple(basis), tuple(np.round(tableau[:, -1],4)), round(tableau[-1, -1],4)
```

```python
def dfs(tableau, basis):
    key = tableau_key(basis, tableau)

    if key in visited:
        return
    visited.add(key)

    # info in node (basis,decision_vars,z)
    node = f"B={[var_labels[i] for i in basis]}\n"
    decision_vars=[0.0]*4
    for row, idx in enumerate(basis):
        if idx < 4:
            decision_vars[idx] = round(tableau[row, -1], 2) #if it is in basis its value is in b column

    node  += f"x={tuple(decision_vars)}\n"  # Shows (x₁, x₂, x₃, x₄)
    node += f"Z={round(-tableau[-1, -1], 2)}"

    ##add node to graph
    # if c coefficients <=0 stop optimal solution
    G.add_node(key, label=node, color="#9EC6F3" if np.all(tableau[-1, :-1] <= 0) else "#FFF1D5")
    # c coefficients
    z_row = tableau[-1, :-1]
    entering_vars = [j for j in range(len(z_row)) if z_row[j] > 0] # possible entering vars
    for entering_col in entering_vars:
        # minimum ratio test
        #compute ratios for each possible entering variable
        ratios = []
        for i in range(num_constraints):
            aij = tableau[i, entering_col] #value in column of the entering var
            if aij > 0:
                ratios.append((tableau[i, -1] / aij, i)) # b_i/ aij
        if not ratios: #ratios for the next entering var
            continue

        min_ratio = min(ratios)[0]
```

```python
        for ratio, leaving_row in ratios:
            if np.isclose(ratio,min_ratio): #for floating errors
                new_tableau = tableau.copy()
                pivot = new_tableau[leaving_row, entering_col]
                new_tableau[leaving_row] /= pivot #1 in pivot element

                for i in range(num_constraints + 1):
                    if i != leaving_row:
                        #zeros in all the rows of entering column except pivot
                        new_tableau[i, :] -= new_tableau[i, entering_col] * new_tableau[leaving_row, :]

                new_basis = basis.copy()

                new_basis[leaving_row] = entering_col #variables interchange in basis
                new_key = tableau_key(new_basis, new_tableau)

                G.add_edge(key, new_key)
                dfs(new_tableau, new_basis)

#starts dfs algorithm with initial state
dfs(tableau, basic_vars)
# Σχεδίαση γράφου
pos = nx.spring_layout(G)
node_colors = [G.nodes[n]["color"] for n in G.nodes]
labels = nx.get_node_attributes(G, "label")
edge_labels = nx.get_edge_attributes(G, "label")
nx.draw_networkx_nodes(G, pos, node_color = node_colors, edgecolors = "black", node_size = 11000)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos, labels=labels, font_size = 12)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size = 5)
plt.title("Simplex Adjacency Graph")
plt.axis("off")
plt.tight_layout()
plt.show()
```