Γραμμική & Συνδυαστική Βελτιστοποίηση

Εργασία #1

Μαρία-Νίκη Ζωγράφου
ΑΜ: 1096060

Περιεχόμενα

Άσκηση 1.

(α) Να παραστήσετε γραφικά την εφικτή περιοχή του προβλήματος καθώς και όλες τις κορυφές της. Περιγράψτε τη μορφή της εφικτής περιοχής. Με γραφικό τρόπο βρείτε τη βέλτιστη κορυφή του προβλήματος, εάν υπάρχει:

Σχεδιασμός ευθειών και κορυφών:

Αρχικά σχεδιάζουμε τις ευθείες που αντιστοιχούν στους περιορισμούς του προβλήματος. Βρίσκουμε τα σημεία τομής των ευθειών χρησιμοποιώντας Επίλυση συστημάτων της βιβλιοθήκης της python Numpy Linalg. Έχουμε ευθείες της μορφής $kx_1 + mx_2 = b$, δηλαδή αν θέσουμε ως διάνυσμα x τα x1,x2 θα έχουμε εξισώσεις με μορφή Ax=b. Επομένως για να βρούμε το σημείο τομής δύο περιορισμών ένα και δύο θα πρέπει να λύσουμε ένα σύστημα της μορφής:

$$\begin{Bmatrix} k_1 * x_1 + m_1 * x_2 = b_1 \\ k_2 * x_1 + m_2 * x_2 = b_2 \end{Bmatrix}$$

Υλοποίηση σε Python:

```
def intersection(A1, b1, A2, b2): # εξισώσεις με μορφή Ax = b
    A = np.array([A1, A2])
    b = np.array([b1, b2])
    try:
        sol = np.linalg.solve(A, b)
        return tuple(sol)
    except np.linalg.LinAlgError:
        return None
```

Από όλα τα σημεία τομής κρατάμε ως κορυφές αυτά που ικανοποιούν όλους τους περιορισμούς του προβλήματος.

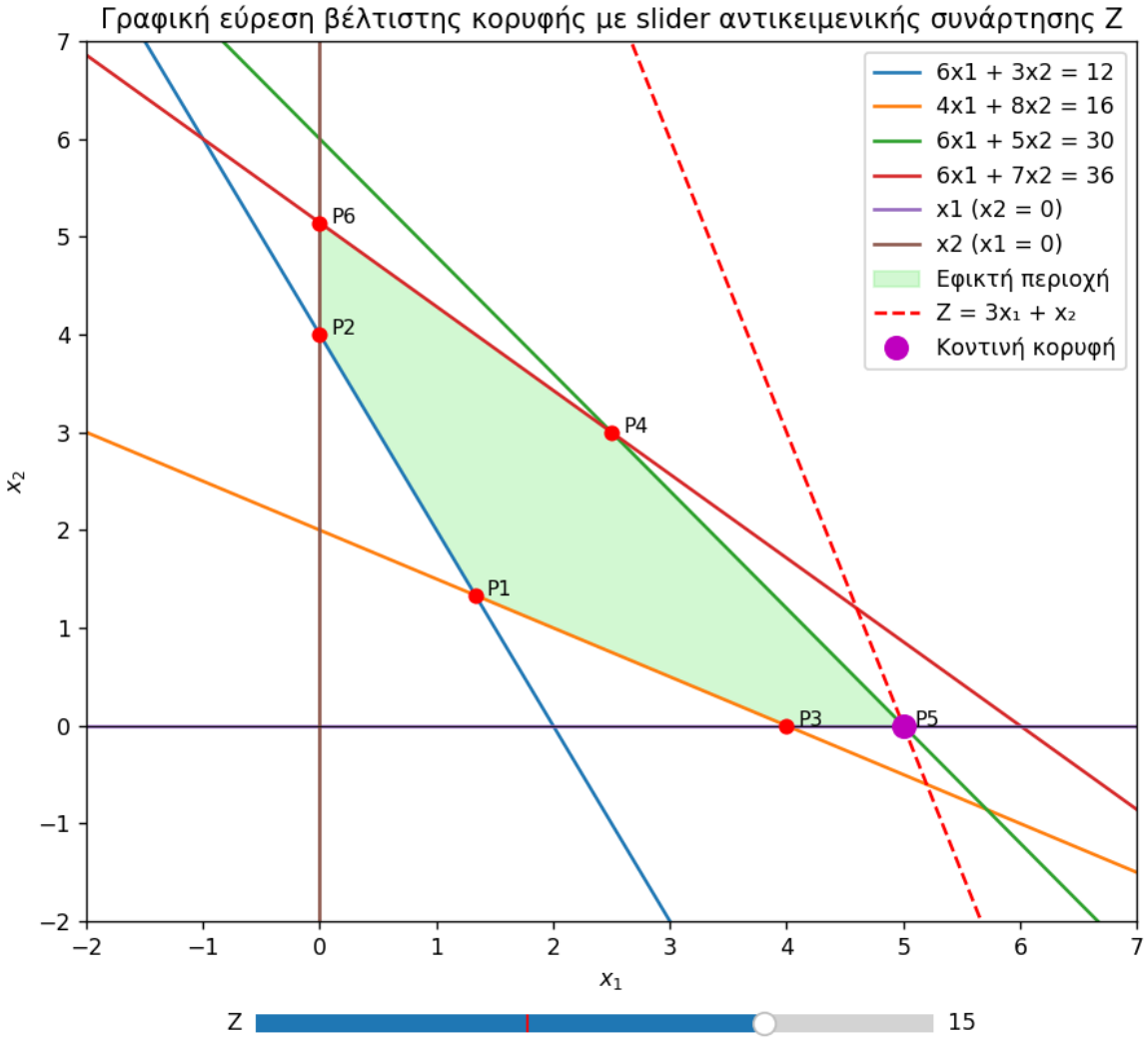
Σχεδιασμός εφικτής περιοχής:

Χρειάζεται να σκιαγραφήσουμε το εσωτερικό του κυρτού περιβλήματος των κορυφών. Γνωρίζουμε ότι όλα τα σημεία βρίσκονται στην επιφάνεια του κυρτού περιβλήματος οπότε δεν χρειάζονται αλγόριθμοι όπως Gift Wrapping (Jarvis March),Graham Scan, QuickHull. Κάνουμε απλώς Υπολογισμό του κέντρου των σημείων, υπολογισμό γωνιών (με arctangent) και ταξινόμηση των σημείων ανά γωνία. Ύστερα συνδέουμε τα σημεία κυκλικά και γεμίζουμε την εσωτερική περιοχή τους.

Εύρεση βέλτιστης κορυφής με γραφικό τρόπο:

Για γραφική αναπαράσταση δημιούργησα έναν slider κινούμενη ευθεία με εξίσωση Z=3x1+x2. Μετακινώντας τον slider Μπορεί να φανεί ότι η μέγιστη τιμή του Z από σημείο που βρίσκεται μέσα στην εφικτή περιοχή είναι Z=15 για την κορυφή P5 (x1=5.0, x2=0.0)

Σημείωση: στο πρόγραμμα, με tolerance 0.1, η κορυφή από την οποία περνάει ο slider τονίζεται.



Κορυφές:

Κορυφή P1: 1.3333333333333333, 1.3333333333333333
Κορυφή P2: 0.0, 4.0
Κορυφή P3: 4.0, 0.0
Κορυφή P4: 2.5, 3.0
Κορυφή P5: 5.0, 0.0
Κορυφή P6: 0.0, 5.142857142857143

(β) Αν η αντικειμενική συνάρτηση του παραπάνω προβλήματος είναι min Z = x1 + c2x2, ποιο είναι το εύρος τιμών που θα μπορούσε να πάρει το c2 έτσι ώστε η βέλτιστη λύση να βρίσκεται στην τομή των ευθειών που ορίζουν οι περιορισμοί Π1 και Π2;

Η τομή των περιορισμών Π1,Π2 συμβαίνει στο σημείο P1 (4/3,4/3). Χρειάζεται να βρούμε για ποια c2 το Z θα είναι ελάχιστο για x1=4/3 kai x2=4/3.

Για κάθε σημείο έχουμε:

$$Z_{P1} = \frac{4}{3} + \frac{4}{3} * c_2, Z_{P2} = 4 * c_2,\ Z_{P3} = 4,\ Z_{P4} = 2.5 + 3 * c_2, Z_{P5} = 5,\ Z_{P6} = \frac{36}{7} * c_2$$

Όπως φαίνεται από το σχήμα αλλά επιβεβαιώνουν και οι εξισώσεις ZP6>ZP2 πάντα και ZP5>ZP3 πάντα. Επομένως μένει να εξετάσουμε τα ZP1,ZP2,ZP3,ZP4.

ZP1 < ZP4 για $7.5 + 9 * c2 > \frac{4}{3} + \frac{4}{3} * c2 \Rightarrow c2 > -\frac{3.5}{5}$

ZP1< ZP2 για $\frac{4}{3} + \frac{4}{3} * c2 < 4 * c2 \Rightarrow c2 > \frac{1}{2}$

ZP1< ZP3 για $4 > \frac{4}{3} + \frac{4}{3} * c2 \Rightarrow c2 < 2$

Επομένως:

$$\boldsymbol{c_2} \in (\frac{\mathbf{1}}{\mathbf{2}}, \mathbf{2})$$

(γ) Αν στο παραπάνω πρόβλημα μεγιστοποίησης η αντικειμενική συνάρτηση ήταν Z = c1x1 + c2x2 βρείτε τις σχετικές τιμές των c1 και c2 έτσι ώστε η βέλτιστη λύση να βρίσκεται στην τομή των ευθειών που ορίζουν οι περιορισμοί Π3 και Π4.

Η τομή των ευθειών των περιορισμών Π3 και Π4 γίνεται στο σημείο P4 (2.5, 3.0)

$$Z_{P1} = \frac{4}{3} * c_1 + \frac{4}{3} * c_2,$$

$$Z_{P2} = 4 * c_2,$$

$$Z_{P3} = 4 * c_1,$$

$$Z_{P4} = 2.5 * c_1 + 3 * c_2,$$

$$Z_{P5} = 5 * c_1,$$

$$Z_{P6} = \frac{36}{7} * c_2$$

Γνωρίζουμε ότι ZP6>ZP2 πάντα και ZP5>ZP3 πάντα, οπότε πρέπει να βρούμε **c1,c2 | ZP4 > ZP1, ZP5, ZP6.**

$$\mathbf{ZP4} > \mathbf{ZP1} \Rightarrow 2.5 * c_1 + 3 * c_2 > \frac{4}{3} * c_1 + \frac{4}{3} * c_2$$

$$5c_2 > -3.5c_1 \Rightarrow \frac{c_2}{c_1} > -\frac{3.5}{5} = -0.7$$

$$\mathbf{ZP4} > \mathbf{ZP5} \Rightarrow 2.5 * c_1 + 3 * c_2 > 5 * c_1$$

$$3c_2 > 2.5c_1 \Rightarrow \frac{c_2}{c_1} > \frac{2.5}{3} = \frac{5}{6}$$

$$\mathbf{ZP4} > \mathbf{ZP6} \Rightarrow 2.5 * c_1 + 3 * c_2 > \frac{36}{7} * c_2$$

$$\frac{-15}{7} c_2 > -2.5c_1 \Rightarrow \frac{c_2}{c_1} < \frac{17.5}{15}$$

Επομένως πρέπει: $\mathbf{\frac{5}{6} < \frac{c_1}{c_2} < \frac{7}{6}}$

Κώδικας

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
# Επανακαθορισμός όλων των περιορισμών ως γραμμές
x = np.linspace(-10, 10, 400)

# Περιορισμοί
def line1(x): return 4 - 2 * x       # 6x1 + 3x2 = 12
def line2(x): return (4 - x) / 2     # 4x1 + 8x2 = 16
def line3(x): return (30 - 6*x) / 5  # 6x1 + 5x2 = 30
def line4(x): return (36 - 6*x) / 7  # 6x1 + 7x2 = 36
def line5(x): return 0 * x            # Άξονας x1


def intersection(A1, b1, A2, b2): # εξισώσεις με μορφή Ax = b ευρεση σημείων τομής
    A = np.array([A1, A2])
    b = np.array([b1, b2])
    try:
        sol = np.linalg.solve(A, b)
        return tuple(sol)
    except np.linalg.LinAlgError:
        return None

# Συντελεστές περιορισμών A και b
#x1: x2=0 άρα A=[0,1] και b=0 για αξονα x1
#x2: x1=0 άρα A=[1,0] και b=0 για αξονα x2
coefsA=[[6,3],[4,8],[6,5],[6,7],[0,1],[1,0]]
coefsB=[12,16,30,36,0,0]
# Υπολογισμός τομών μεταξύ περιορισμών και σχεδιασμός σημείων τομής
intersections = []
for i in range(len(coefsA)):
    for j in range(i+1, len(coefsA)):
        x_val, y_val = intersection(coefsA[i], coefsB[i], coefsA[j], coefsB[j])
        if x_val is not None and y_val is not None:


            # Έλεγχος αν οι τομές είναι στον 1ο τεταρτημόριο
            if x_val >= 0 and y_val >= 0:
                # Έλεγχος αν οι τομές ικανοποιούν όλους τους περιορισμούς
                if (
                    2*x_val + y_val >= 4 and
                    x_val + 2*y_val >= 4 and
                    6*x_val + 5*y_val <= 30 and
                    6*x_val + 7*y_val <= 36
                ):
```

```
                    intersections.append([x_val, y_val])
                    #plt.plot(x_val, y_val, 'ro')
                    #plt.text(x_val + 0.1, y_val, f'P{i+1}{j+1}')

intersections = np.array(intersections) #valid intersection
############ - χρωματισμός εφικτής περιοχής  ######################
################## εύρεση κυρτού περιβλήματος ####################
# Υπολογισμός κέντρου
center = np.mean(intersections, axis=0)
# Υπολογισμός γωνίας κάθε σημείου σε σχέση με το κέντρο
angles = np.arctan2(intersections[:,1] - center[1], intersections[:,0] - center[0])
# Ταξινόμηση των σημείων με βάση τις γωνίες
sorted_indices = np.argsort(angles)
sorted_vertices = intersections[sorted_indices]

# Με γραφικό τρόπο βρείτε  τη βέλτιστη κορυφή του προβλήματος, εάν υπάρχει. #
Υπολογισμός Z
#slider
# Z = 3x1 + x2
# Υπολογισμός Z για κάθε σημείο
def Z(x1, x2): return 3 * x1 + x2
z_values = [Z(x, y) for x, y in intersections]

# Δημιουργία γραφήματος
fig, ax = plt.subplots(figsize=(8, 8))
plt.subplots_adjust(bottom=0.15)

# Σχεδίαση περιορισμών
ax.plot(x, line1(x), label='6x1 + 3x2 = 12')
ax.plot(x, line2(x), label='4x1 + 8x2 = 16')
ax.plot(x, line3(x), label='6x1 + 5x2 = 30')
ax.plot(x, line4(x), label='6x1 + 7x2 = 36')
ax.plot(x, line5(x), label='x1 (x2 = 0)')
ax.plot(0 * x, x, label='x2 (x1 = 0)')
ax.axhline(0, color='black', lw=0.5)

# Σχεδίαση εφικτής περιοχής
ax.fill(sorted_vertices[:, 0], sorted_vertices[:, 1], color='lightgreen', alpha=0.4,
label='Εφικτή περιοχή')

# Σχεδίαση κορυφών
for i, (xv, yv) in enumerate(intersections):
    ax.plot(xv, yv, 'ro')
    ax.text(xv + 0.1, yv, f'P{i+1}', fontsize=9)
    print(f'Κορυφή P{i+1}: {xv}, {yv}')
# Προσθήκη ευθείας Z
initial_z = 8
z_line, = ax.plot(x, initial_z - 3 * x, 'r--', label='Z = 3x₁ + x₂')  #x2=z-3x1
```

```
def currentvertex(z_val):
    z_val = z_slider.val
    tol=0.1
    for x, y in intersections:
        if abs(Z(x, y) - z_val) <= tol:
            #print(f'Κορυφή: {x}, {y}')
            return np.array([x, y])
    return None
highlight, = ax.plot([], [], 'mo', markersize=10, label='Κοντινή κορυφή')

# Slider
ax_slider = plt.axes([0.25, 0.05, 0.5, 0.03])
z_slider = Slider(ax_slider, 'Z', 0, 20, valinit=initial_z, valstep=0.1)

# Slider update function
def update(val):
    z_val = z_slider.val
    z_line.set_ydata(z_val - 3 * x)
    bv = currentvertex(z_val)
    if bv is not None:
        highlight.set_data([bv[0]], [bv[1]])
    else:
        highlight.set_data([], [])
    fig.canvas.draw_idle()


z_slider.on_changed(update)
update(initial_z)

ax.set_xlim(-2, 7)
ax.set_ylim(-2, 7)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('Γραφική εύρεση βέλτιστης κορυφής με slider αντικειμενικής συνάρτησης
Z')
ax.legend()
plt.grid(True)
plt.show()
```

Άσκηση 2.

Η συνολική ακτινοβολία που τελικά απορροφά η κάθε περιοχή του σώματος του ασθενούς είναι το άθροισμα της ακτινοβο λίας που απορροφά από την κάθε ξεχωριστή δέσμη.

Περιοχή	Ποσοστό Ραδιενέργειας που απορροφά η περιοχή		Περιορισμοί στη Συνολική Ραδιενέργεια
	Δέσμη 1	Δέσμη 2	
Υγιής περιοχή	0.4	0.5	Ελάχιστο
Ευαίσθητοι ιστοί	0.3	0.1	≤ 2.7
Περιοχή όγκου	0.5	0.5	$= 6$
Κέντρο μάζας όγκου	0.6	0.4	≥ 6.0

(α) Μοντελοποιήστε το παραπάνω πρόβλημα σχεδιασμού ραδιοθεραπείας με τη βοήθεια του γραμμικού προγραμματισμού.

1. Θέτουμε **x1** και **x2** τις εντάσεις σε kilorad των ακτινών 1 και 2 αντίστοιχα.

2. Ορίζουμε την αντικειμενική συνάρτηση $Z = \min(0.4 * x_1 + 0.5 * x_2)$

Ή αλλιώς $Z = \max -(0.4 * x_1 + 0.5 * x_2)$

3. Καταγράφουμε τους περιορισμούς με ανισώσεις:

$$0.3 * x_1 + 0.1 * x_2 \leq 2.7 \text{ (Π1)}$$

$$0.5 * x_1 + 0.5 * x_2 = 6 \text{ (Π2)}$$

$$0.6 * x_1 + 0.4 * x_2 \geq 6 \text{ (Π3)}$$

$$x_1, x_2 \geq 0$$

Από τον περιορισμό 2 παρατηρούμε ότι $x_2 = 12 - x_1$.

(β) Λύστε το πρόβλημα με γραφικό τρόπο.

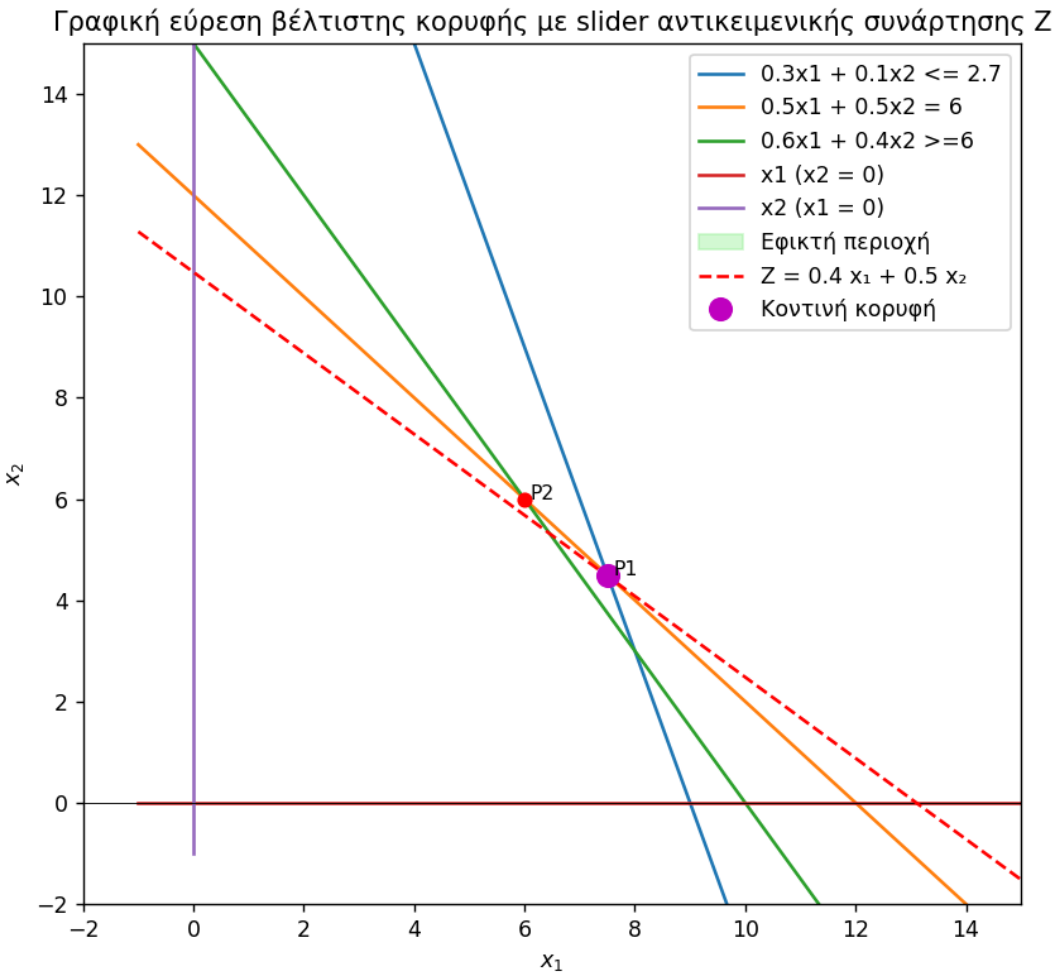
Έχουμε περιορισμό ισότητας, άρα η εφικτή περιοχή είναι μόνο πάνω στην ευθεία του περιορισμού Π2 (πορτοκαλί ευθεία στο σχήμα). Επομένως υπάρχουν μόνο δύο εφικτές κορυφές.

Εφικτές κορυφές στο terminal εκτυπωμένες:

Κορυφή P1: 7.5, 4.5, Z:5.25

Κορυφή P2: 6.0, 6.0, Z:5.4

Ζωγραφίζουμε την ευθεία της αντικειμενικής συνάρτησης Z. Μετακινώντας την από το 0 προς τα πάνω παρατηρούμε ότι τέμνει πρώτα το P1 (**κορυφή x1=7.5** kilorad**, x2=4.5** kilorad). Εκεί έχουμε το μικρότερο **Z=5.25**.



Κώδικας

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
# Επανακαθορισμός όλων των περιορισμών ως γραμμές
x = np.linspace(-1, 20, 400)

# Περιορισμοί
# 0.3x1 + 0.1x2 <= 2.7
# 0.5x1 + 0.5x2 = 6
# 0.6x1 + 0.4x2 >=6
def line1(x): return 27. - 3. * x       # 0.3*x1 + 0.1*x2 <= 2.7
def line2(x): return 12.-x     # 0.5*x1 + 0.5*x2 = 6
def line3(x): return 15.-3.*x/2  # 0.6*x1 + 0.4*x2 >=6

def line5(x): return 0 * x            # Άξονας x1

def intersection(A1, b1, A2, b2): # εξισώσεις με μορφή Ax = b
    A = np.array([A1, A2])
    b = np.array([b1, b2])
```

```
    try:
        sol = np.linalg.solve(A, b)
        return tuple(sol)
    except np.linalg.LinAlgError:
        return None
def isclose(a, b, rel_tol=1e-09, abs_tol=0.0):
    return abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)
# Συντελεστές περιορισμών A και b
#x1: x2=0 άρα A=[0,1] και b=0
#x2: x1=0 άρα A=[1,0] και b=0
coefsA=[[0.3,0.1],[0.5,0.5],[0.6,0.4],[0,1],[1,0]]
coefsB=[2.7, 6, 6, 0, 0]
# Υπολογισμός τομών μεταξύ περιορισμών και σχεδιασμός σημείων τομής
intersections = []
for i in range(len(coefsA)):
    for j in range(i+1, len(coefsA)):
        x_val, y_val = intersection(coefsA[i], coefsB[i], coefsA[j], coefsB[j])

        if x_val is not None and y_val is not None:


            # Έλεγχος αν οι τομές είναι στον 1ο τεταρτημόριο
            if x_val >= 0 and y_val >= 0:
                # Έλεγχος αν οι τομές ικανοποιούν όλους τους περιορισμούς
                if (
                    round(0.3*x_val + 0.1*y_val,3) <= 2.7 and
                    isclose(0.5*x_val + 0.5*y_val, 6.0 )and
                    0.6*x_val + 0.4*y_val >=6):

                    intersections.append([x_val, y_val])
                    #print(f'Intersection {i+1} and {j+1}: {x_val}, {y_val}, Z:{}')
                    #plt.plot(x_val, y_val, 'ro')
                    #plt.text(x_val + 0.1, y_val, f'P{i+1}{j+1}')
# Υπολογισμός Z για κάθε σημείο
def Z(x1, x2): return ((0.4 * x1 + 0.5*x2)*1.)
# Retrieve the second row #row = arr[1, :]
intersections = np.array(intersections) #valid intersection

################## εύρεση κυρτού περιβλήματος ##########################
# Υπολογισμός κέντρου
center = np.mean(intersections, axis=0)
# Υπολογισμός γωνίας κάθε σημείου σε σχέση με το κέντρο
angles = np.arctan2(intersections[:,1] - center[1], intersections[:,0] - center[0])
# Ταξινόμηση των σημείων με βάση τις γωνίες
sorted_indices = np.argsort(angles)
sorted_vertices = intersections[sorted_indices]

# Σχεδίαση περιοχής με fill
```

```
#plt.fill(sorted_vertices[:, 0], sorted_vertices[:, 1], color='lightgreen',
alpha=0.4, label='Εφικτή περιοχή')

# Με γραφικό τρόπο βρείτε  τη βέλτιστη κορυφή του προβλήματος, εάν υπάρχει. #
Υπολογισμός Z
#slider
# Z = 3x1 + x2

z_values = [Z(x, y) for x, y in intersections]

# Δημιουργία γραφήματος
fig, ax = plt.subplots(figsize=(8, 8))
plt.subplots_adjust(bottom=0.15)

# Σχεδίαση περιορισμών
ax.plot(x, line1(x), label='0.3x1 + 0.1x2 <= 2.7')
#
# 0.6x1 + 0.4x2 >=6
ax.plot(x, line2(x), label='0.5x1 + 0.5x2 = 6')
ax.plot(x, line3(x), label='0.6x1 + 0.4x2 >=6')
ax.plot(x, line5(x), label='x1 (x2 = 0)')
ax.plot(0 * x, x, label='x2 (x1 = 0)')
ax.axhline(0, color='black', lw=0.5)

# Σχεδίαση εφικτής περιοχής
ax.fill(sorted_vertices[:, 0], sorted_vertices[:, 1], color='lightgreen', alpha=0.4,
label='Εφικτή περιοχή')

# Σχεδίαση κορυφών
for i, (xv, yv) in enumerate(intersections):
    ax.plot(xv, yv, 'ro')
    ax.text(xv + 0.1, yv, f'P{i+1}', fontsize=9)
    x_val, y_val = intersections[i]
    print(f'Κορυφή P{i+1}: {round(xv,3)}, {round(yv,3)}, Z:{round(Z(x_val, y_val),
3)} ')
# Προσθήκη ευθείας Z
initial_z = 2
z_line, = ax.plot(x, 2.*initial_z - 0.8 * x, 'r--', label='Z = 0.4 x₁ + 0.5
x₂')  #x2=2z-0.8x1

def currentvertex(z_val):
    z_val = z_slider.val
    tol=0.02
    for x, y in intersections:
        if abs(round(Z(x, y),3) - round(z_val,3)) <= tol:
            #print(f'Κορυφή: {x}, {y}')
            return np.array([x, y])
    return None
```

```
highlight, = ax.plot([], [], 'mo', markersize=10, label='Κοντινή κορυφή')

# Slider
ax_slider = plt.axes([0.25, 0.01, 0.65, 0.03], facecolor='lightgoldenrodyellow')
z_slider = Slider(ax_slider, 'Z', -10,  10, valinit=initial_z, valstep=0.0001,
valfmt='%1.2f')

# Slider update function
def update(val):
    z_val = z_slider.val
    z_line.set_ydata(np.round(2. * z_val - 0.8 * x, 3))

    bv = currentvertex(z_val)
    if bv is not None:
        highlight.set_data([bv[0]], [bv[1]])
    else:
        highlight.set_data([], [])
    fig.canvas.draw_idle()


z_slider.on_changed(update)
update(initial_z)

ax.set_xlim(-2, 15)
ax.set_ylim(-2, 15)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('Γραφική εύρεση βέλτιστης κορυφής με slider αντικειμενικής συνάρτησης
Z')
ax.legend()
plt.grid(True)
plt.show()
```

Άσκηση 3.

Στόχος Παραγωγής σε Τόνους	
Τύπος I	12
Τύπος II	8
Τύπος III	9

Τιμή ανά kg	
Καλαμπόκι	0.20
Ασβεστόλιθο	0.12
Σόγια	0.24
Ιχθυάλευρα	0.12

Διαθεσιμότητα Υλικών σε Τόνους	
Καλαμπόκι	9
Ασβεστόλιθο	12
Σόγια	5
Ιχθυάλευρα	6

Πίνακας 1: Περιεκτικότητα σε θρεπτικά συστατικά (ανά kg υλικού)

Πρώτη Ύλη	Βιταμίνες	Πρωτεΐνες	Ασβέστιο	Λίπος
Καλαμπόκι	8	10	6	8
Ασβεστόλιθος	6	5	10	6
Σόγια	10	12	6	6
Ιχθυάλευρο	4	8	6	9

Πίνακας 2: Απαιτήσεις σε θρεπτικά συστατικά (μονάδες θρεπτικής ουσίας ανά kg ζωοτροφής)

	Βιταμίνες		Πρωτεΐνες		Ασβέστιο		Λίπος	
Ζωοτροφή	Min	Max	Min	Max	Min	Max	Min	Max
τύπου I	6	∞	6	∞	7	∞	4	8
τύπου II	6	∞	6	∞	6	∞	4	6
τύπου III	4	6	6	∞	6	∞	4	5

Θα χρειαστεί, για να περιγράψουμε τους περιορισμούς, να γνωρίζουμε την ποσότητα κάθε πρώτης ύλης που θα χρησιμοποιηθεί. Επειδή κάθε τύπος τροφής έχει διαφορετικούς περιορισμούς θα πρέπει να επαναλάβουμε αυτή τη διαδικασία τρεις φορές. Εν τέλει ορίζουμε ως μεταβλητές απόφασης την **ποσότητα** σε **κιλά** που θα χρησιμοποιήσουμε από κάθε πρώτη ύλη για τον κάθε τύπο τροφής.

Μεταβλητή απόφασης: $\boldsymbol{x_{ij}, i \in \{1,2,3\}, j \in \{1,2,3,4\}}$, όπου i ο τύπος τροφής και j το είδος πρώτης ύλης: καλαμπόκι, ασβεστόλιθοι, σόγια, ιχθυάλευρα αντίστοιχα.

Συμβολίζω Βιταμίνες, Πρωτεΐνες, Ασβέστιο, Λίπος με $\boldsymbol{\beta_i, \pi_i, \alpha_i, \lambda_i}$ αντίστοιχα (i ο τύπος τροφής).

Αντικειμενική Συνάρτηση: Για αντικειμενική συνάρτηση θα χρησιμοποιήσω το συνολικό κόστος παραγωγής και των τριών τροφών

$$\boldsymbol{Z = min(\sum_{i=1}^{3} 0.20\, x_{i1} + 0.12 x_{i2} + 0.24 x_{i3} + 0.12 x_{i4})}$$

Περιορισμοί:

Αρχικά ελέγχουμε ότι δεν υπερβαίνουμε την **Διαθεσιμότητα Υλικών σε Τόνους:**

Καλαμπόκι: $\sum_{i=1}^{3} x_{i1} * 10^3 < 9$

Ασβεστόλιθο: $\sum_{i=1}^{3} x_{i2} * 10^3 < 12$

Σόγια: $\sum_{i=1}^{3} x_{i3} * 10^3 < 5$

Ιχθυάλευρο: $\sum_{i=1}^{3} x_{i4} * 10^3 < 6$

Στη συνέχεια περιγράφουμε μαθηματικά τους **Στόχους Παραγωγής σε Τόνους**

Τύπος I: $\sum_{j=1}^{4} x_{1j} * 10^3 = 12$

Τύπος II: $\sum_{j=1}^{4} x_{2j} * 10^3 = 8$

Τύπος III: $\sum_{j=1}^{4} x_{3j} * 10^3 = 9$

Τέλος εξετάζουμε αν τηρείται η **απαιτούμενη περιεκτικότητα σε θρεπτικά συστατικά (Πίνακες 1 και 2):**

Τύπος I:

$$\beta 1 = 8x_{11} + 6x_{12} + 10x_{13} + 4x_{14} \geq 6$$

$$\pi 1 = 10x_{11} + 5x_{12} + 12x_{13} + 8x_{14} \geq 6$$

$$\alpha 1 = 6x_{11} + 10x_{12} + 6x_{13} + 6x_{14} \geq 7$$

$$\lambda 1: 8x_{11} + 6x_{12} + 6x_{13} + 9x_{14} \geq 4$$

$$\kappa\alpha\iota\ 8x_{11} + 6x_{12} + 6x_{13} + 9x_{14} \leq 8$$

Εν συντομία:

$$\begin{bmatrix} 6 \\ 6 \\ 7 \\ 4 \end{bmatrix} \leq \begin{bmatrix} 8 & 6 & 10 & 4 \\ 10 & 5 & 12 & 8 \\ 6 & 10 & 6 & 6 \\ 8 & 6 & 6 & 9 \end{bmatrix} * \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \end{bmatrix} \leq \begin{bmatrix} \infty \\ \infty \\ \infty \\ 8 \end{bmatrix}$$

Τύπος II:

$$\begin{bmatrix} 6 \\ 6 \\ 6 \\ 4 \end{bmatrix} \leq \begin{bmatrix} 8 & 6 & 10 & 4 \\ 10 & 5 & 12 & 8 \\ 6 & 10 & 6 & 6 \\ 8 & 6 & 6 & 9 \end{bmatrix} * \begin{bmatrix} x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} \leq \begin{bmatrix} \infty \\ \infty \\ \infty \\ 6 \end{bmatrix}$$

Τύπος III:

$$\begin{bmatrix} 4 \\ 6 \\ 6 \\ 4 \end{bmatrix} \leq \begin{bmatrix} 8 & 6 & 10 & 4 \\ 10 & 5 & 12 & 8 \\ 6 & 10 & 6 & 6 \\ 8 & 6 & 6 & 9 \end{bmatrix} * \begin{bmatrix} x_{31} \\ x_{32} \\ x_{33} \\ x_{34} \end{bmatrix} \leq \begin{bmatrix} 6 \\ \infty \\ \infty \\ 5 \end{bmatrix}$$

Τέλος: $x_{i,j} \geq 0$

Άσκηση 4.

Εξετάστε ως προς την κυρτότητα τα σύνολα:

(α) $\{(x_1, x_2) \in R^2 \mid x_1^2 + x_2^2 \geq 3\}$

Γνωρίζουμε ότι ένα σύνολο $X \subset R^n$ είναι κυρτό αν και μόνο αν:

$$\forall x, y \in X \text{ και } \forall \lambda \in [0,1]: \lambda x + (1-\lambda) y \in X$$

(δηλ. αν και μόνο αν το ευθύγραμμο τμήμα που ενώνει δύο οποιαδήποτε σημεία του Χ βρίσκεται ολόκληρο μέσα στο Χ.)

Το παραπάνω σύνολο περιγράφει όλα τα σημεία εκτός ενός κύκλου με ακτίνα √3 σε ένα επίπεδο. Μπορούμε να αποδείξουμε την μη κυρτότητα του συνόλου με ***αντιπαράδειγμα***:

Σημείο x: (0, √3) και σημείο y: (√3,0). Τα σημεία μας είναι στον χώρο R^2 επομένως το $\lambda x + (1-\lambda) y$ θα είναι:

$$\begin{Bmatrix} \lambda x_1 + (1-\lambda) y_1 \\ \lambda x_2 + (1-\lambda) y_2 \end{Bmatrix} = \begin{Bmatrix} \lambda\sqrt{3} \\ (1-\lambda)\sqrt{3} \end{Bmatrix} = z$$

Για το νέο σημείο z: $z_1^2 + z_2^2 = 3(2\lambda^2 - 2\lambda + 1)$. Θέλουμε $z_1^2 + z_2^2 \geq 3 \Rightarrow 2\lambda^2 - 2\lambda \geq 0$

Άρα $\lambda(\lambda - 1) \geq 0$. Όμως $\lambda(\lambda - 1) < 0$ για $\lambda \in (0,1)$, επομένως για 0<λ<1 το z δεν ανήκει στο σύνολο Χ, οπότε ***<u>το σύνολο Χ δεν μπορεί να είναι κυρτό.</u>***

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

Έστω σημείο x (x_1, x_2, x_3) και y (y_1, y_2, y_3) και $x, y \in X = \{(x_1, x_2, x_3) \in R^3 \mid x_1 + 2x_2 \leq 1, x_1 - 2x_3 \leq 2\}$

Πρέπει νδο. $z = \lambda x + (1-\lambda) y \in X$ για $\forall \lambda \in [0,1]$

Έχουμε:

$$\begin{Bmatrix} z_1 \\ z_2 \\ z_3 \end{Bmatrix} = \begin{Bmatrix} \lambda x_1 + (1-\lambda) y_1 \\ \lambda x_2 + (1-\lambda) y_2 \\ \lambda x_3 + (1-\lambda) y_3 \end{Bmatrix}$$

Εξετάζουμε αν ανήκει στο Χ:

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

Άρα $z_1 + 2z_2 \leq 1$ τηρείται.

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

Άρα $z_1 - 2z_3 \leq 2$.

Οπότε $\forall x, y \in X \text{ και } \forall \lambda \in [0,1]: \lambda x + (1-\lambda) y \in X$ ***<u>επομένως το σύνολο είναι κυρτό.</u>***

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

Όπως και προηγουμένως:

$$\begin{Bmatrix} z_1 \\ z_2 \\ z_3 \end{Bmatrix} = \begin{Bmatrix} \lambda x_1 + (1-\lambda)y_1 \\ \lambda x_2 + (1-\lambda)y_2 \\ \lambda x_3 + (1-\lambda)y_3 \end{Bmatrix}$$

Πρέπει νδο: $z_2 \geq \mathrm{x}_1^2 \Rightarrow z_2 - \mathrm{z}_1^2 \geq 0 \Rightarrow \lambda x_2 + (1-\lambda)y_2 \geq (\lambda x_1 + (1-\lambda)y_1)^2$

Γνωστό ότι: $x_2 \geq \mathrm{x}_1^2 \; και \; y_2 \geq \mathrm{y}_1^2 \Rightarrow \lambda x_2 + (1-\lambda)y_2 \geq (1-\lambda)\mathrm{y}_1^2 + \lambda \mathrm{x}_1^2$ ($\boldsymbol{Σχέση\ 1}$)

Θέτω $f(\mathrm{x}) = x^2, με\ f''(x) = 2 > 0$ *άρα* f *κυρτή στο* R

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

Για $\lambda_1 = \lambda, \lambda_n = 1-\lambda, x_1 = x_1$και $x_2 = y_1$, έχουμε:

$$(\lambda x_1 + (1-\lambda)y_1)^2 \leq \lambda \mathrm{x}_1^2 + (1-\lambda)\mathrm{y}_1^2 \xrightarrow{\boldsymbol{Σχέση\ 1}} (\lambda x_1 + (1-\lambda)y_1)^2 \leq \lambda x_2 + (1-\lambda)y_2$$

Άρα $z_2 \geq \mathrm{z}_1^2$.

Πρέπει νδο: $\mathrm{z}_1 + 2\mathrm{z}_2 + \mathrm{z}_3 \leq 4$

$$\mathrm{x}_1 + 2\mathrm{x}_2 + \mathrm{x}_3 \leq 4 \; και \; y_1 + 2\mathrm{y}_2 + \mathrm{y}_3 \leq 4 \Rightarrow$$

$$\lambda x_1 + 2\lambda x_2 + \lambda x_3 + (1-\lambda)y_1 + 2(1-\lambda)\mathrm{y}_2 + (1-\lambda)\mathrm{y}_3 \leq 4\lambda + (1-\lambda)4$$

$$\mathrm{z}_1 + 2\mathrm{z}_2 + \mathrm{z}_3 = \lambda x_1 + 2\lambda x_2 + \lambda x_3 + (1-\lambda)y_1 + 2(1-\lambda)\mathrm{y}_2 + (1-\lambda)\mathrm{y}_3 \leq 4$$

Άρα $\mathrm{z}_1 + 2\mathrm{z}_2 + \mathrm{z}_3 \leq 4$ *ισχύει.*

Επομένως το σύνολο είναι κυρτό.

(δ) $\{\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) \in \mathrm{R}^3 \mid \mathrm{x}_3 = |\mathrm{x}_2|, \; \mathrm{x}_1 \leq 3\}$

Όπως και προηγουμένως:

$$\begin{Bmatrix} z_1 \\ z_2 \\ z_3 \end{Bmatrix} = \begin{Bmatrix} \lambda x_1 + (1-\lambda)y_1 \\ \lambda x_2 + (1-\lambda)y_2 \\ \lambda x_3 + (1-\lambda)y_3 \end{Bmatrix}$$

$z_3 = \lambda|x_2| + (1-\lambda)|y_2|$ *και* $z_2 = \lambda x_2 + (1-\lambda)y_2$

$$|z_2| = |\lambda x_2 + (1-\lambda)y_2|$$

Από τριγωνική ανισότητα γνωρίζουμε ότι $|a+\beta| \leq |\alpha| + |\beta|$. Επειδή η ισότητα δεν ισχύει πάντα μπορούμε να βρούμε ***αντιπαράδειγμα:***

Για $x_1 = (0, -1, 1)$ και $y_1 = (0, 1, 1)$

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

Έστω $\lambda = 0.25 \Rightarrow |z_2| = 0.5 \neq z_3$.

Άρα $z \notin \{\mathrm{x}_1, \mathrm{x}_2, \mathrm{x}_3) \in \mathrm{R}^3 \mid \mathrm{x}_3 = |\mathrm{x}_2|, \; \mathrm{x}_1 \leq 3\}$, επομένως ***το σύνολο δεν είναι κυρτό.***

Άσκηση 5.

$$min\ Z\ =\ 8x_1 + 5x_2 + 4x_3$$

$$x_1\ + x_2\ \geq\ 10\ (1)$$

$$x_3\ + x_2\ \geq\ 15\ (2)$$

$$x_1\ + x_3\ \geq\ 12\ (3)$$

$$20x_1\ + 10x_2 + 15x_3 \leq\ 300\ (4)$$

$$x_1\ , x_2, x_3\ \geq\ 0\ \ (5), (6), (7)$$

(α) Θεωρήστε το πολύτοπο των εφικτών λύσεων του παραπάνω προβλήματος γραμμικού προγραμματισμού. Βρείτε όλες τις κορυφές που δημιουργούνται από τομές των υπερεπιπέδων του και ξεχωρίστε ποιες από αυτές είναι κορυφές του πολύτοπου των εφικτών λύσεων. Εντοπίστε, αν υπάρχουν, τις εκφυλισμένες κορυφές.

Καθώς έχουμε 3 μεταβλητές, οι κορυφές προκύπτουν από την τομή 3 υπερεπιπέδων, δηλαδή από συστήματα 3 εξισώσεων. Έχουμε 7 περιορισμούς άρα οι πιθανές κορυφές είναι:

$$\binom{7}{3} = \frac{7!}{3! * 4!} = 35$$

Θα πάρουμε όλες τους συνδυασμούς των υπερεπιπέδων και θα υπολογίσουμε όλες τις τομές. Ύστερα θα ελέγξουμε αν μια κορυφή τηρεί όλους τους περιορισμούς και επομένως είναι εφικτή και ανήκει στο πολύτοπο. Τέλος θα ελέγξουμε αν μια κορυφή είναι εκφυλισμένη. Αν η κορυφή προκύπτει από τομή παραπάνω από 3 περιορισμών είναι εκφυλισμένη.

Αποτελέσματα:

Κορυφές:

	x1	x2	x3	Constraints	Status	Degenerate	Z
0	3.50	6.5	8.500000	(0, 1, 2)	feasible	False	94.500000
1	5.00	5.0	10.000000	(0, 1, 3)	feasible	False	105.000000
2	-0.00	10.0	5.000000	(0, 1, 4)	infeasible	False	70.000000
3	10.00	-0.0	15.000000	(0, 1, 5)	infeasible	False	140.000000
4	-5.00	15.0	-0.000000	(0, 1, 6)	infeasible	False	35.000000
5	-4.00	14.0	16.000000	(0, 2, 3)	infeasible	False	102.000000
6	-0.00	10.0	12.000000	(0, 2, 4)	feasible	False	98.000000
7	10.00	0.0	2.000000	(0, 2, 5)	infeasible	False	88.000000
8	12.00	-2.0	-0.000000	(0, 2, 6)	infeasible	False	86.000000
9	0.00	10.0	13.333333	(0, 3, 4)	feasible	False	103.333333
10	10.00	-0.0	6.666667	(0, 3, 5)	infeasible	False	106.666667
11	20.00	-10.0	-0.000000	(0, 3, 6)	infeasible	False	110.000000
12	NaN	NaN	NaN	(0, 4, 5)	singular	False	NaN
13	-0.00	10.0	-0.000000	(0, 4, 6)	infeasible	False	50.000000
14	10.00	-0.0	-0.000000	(0, 5, 6)	infeasible	False	80.000000
15	6.00	9.0	6.000000	(1, 2, 3)	feasible	False	117.000000
16	-0.00	3.0	12.000000	(1, 2, 4)	infeasible	False	63.000000
17	-3.00	-0.0	15.000000	(1, 2, 5)	infeasible	False	36.000000
18	12.00	15.0	-0.000000	(1, 2, 6)	infeasible	False	171.000000
19	0.00	-15.0	30.000000	(1, 3, 4)	infeasible	False	45.000000
20	3.75	-0.0	15.000000	(1, 3, 5)	infeasible	False	90.000000

21	7.50	15.0	-0.000000	(1, 3, 6)	infeasible	False	135.000000
22	-0.00	-0.0	15.000000	(1, 4, 5)	infeasible	False	60.000000
23	-0.00	15.0	-0.000000	(1, 4, 6)	infeasible	False	75.000000
24	NaN	NaN	NaN	(1, 5, 6)	singular	False	NaN
25	0.00	12.0	12.000000	(2, 3, 4)	feasible	False	108.000000
26	24.00	-0.0	-12.000000	(2, 3, 5)	infeasible	False	144.000000
27	12.00	6.0	-0.000000	(2, 3, 6)	infeasible	False	126.000000
28	-0.00	-0.0	12.000000	(2, 4, 5)	infeasible	False	48.000000
29	NaN	NaN	NaN	(2, 4, 6)	singular	False	NaN
30	12.00	-0.0	-0.000000	(2, 5, 6)	infeasible	False	96.000000
31	0.00	-0.0	20.000000	(3, 4, 5)	infeasible	False	80.000000
32	0.00	30.0	-0.000000	(3, 4, 6)	infeasible	False	150.000000
33	15.00	-0.0	-0.000000	(3, 5, 6)	infeasible	False	120.000000
34	-0.00	-0.0	-0.000000	(4, 5, 6)	infeasible	False	-0.000000

Vertices of the feasible region:

	x1	x2	x3	Z
0	3.5	6.5	8.500000	94.500000
1	5.0	5.0	10.000000	105.000000
2	-0.0	10.0	12.000000	98.000000
3	0.0	10.0	13.333333	103.333333
4	6.0	9.0	6.000000	117.000000
5	0.0	12.0	12.000000	108.000000

Παρατηρούμε ότι:

1. Καμία τομή δεν είναι εκφυλισμένη
2. Από τις 35 κορυφές, 3 δεν είναι υπολογίσιμες δηλαδή για τρεις συνδυασμούς 3 επιπέδων δεν υπάρχει μοναδικό σημείο τομής
3. Από τις 35 οι 6 κορυφές είναι εφικτές

Κώδικας α:

```
'''min Z = 8x1+5x2+4x3
 x1 +x2 ≥ 10
 x2 +x3 ≥ 15
 x1 +x3 ≥ 12
 20x1 +10x2 +15x3 ≤ 300
 x1, x2,x3 ≥ 0
'''
import numpy as np
import itertools
import pandas as pd
```

```
# Ορίζουμε τους περιορισμούς ως εξισώσεις της μορφής Ax = b
# έχουμε 7 περιορισμούς: της μορφής Αχ<=β
# (1) x1 + x2 >= 10     → -x1 - x2 <= -10
# (2) x2 + x3 >= 15     → -x2 - x3 <= -15
# (3) x1 + x3 >= 12     → -x1 - x3 <= -12
# (4) 20x1 + 10x2 + 15x3 <= 300
# (5) x1 >= 0           → -x1 <= 0
# (6) x2 >= 0           → -x2 <= 0
# (7) x3 >= 0           → -x3 <= 0

A = np.array([
    [-1, -1,  0],  # (1)
    [ 0, -1, -1],  # (2)
    [-1,  0, -1],  # (3)
    [20, 10, 15],  # (4)
    [-1,  0,  0],  # (5)
    [ 0, -1,  0],  # (6)
    [ 0,  0, -1]   # (7)
])
b = np.array([-10, -15, -12, 300, 0, 0, 0])

# Δημιουργούμε όλα τα δυνατά συστήματα 3 περιορισμών (συνδυασμοί 3 από 7)
# και αποθηκεύουμε ΚΑΙ τις μη εφικτές ή μη ορισμένες λύσεις

all_combinations = list(itertools.combinations(range(7), 3)) # επιλέγουμε 3 από 7
περιορισμούς (παίρνουμε όλους τους συνδυασμούς τριάδων αριθμών 0 εως 6)

# Λίστες για πλήρη καταγραφή όλων των συνδυασμών
all_vertices = []
all_statuses = []  # 'feasible', 'infeasible', 'singular'
all_Z = []
fes_vertices=[] # Λίστα για εφικτές κορυφές
fes_Z=[] # Λίστα για εφικτές τιμές Z
for indices in all_combinations:
    A_eq = A[list(indices)]
    b_eq = b[list(indices)]
    try:
        sol = np.linalg.solve(A_eq, b_eq) # Λύνουμε το σύστημα των 3 εξισώσεων
        # Ελέγχουμε αν ικανοποιεί όλους τους περιορισμούς (εφικτή κορυφή)
        if np.all(A @ sol <= b + 1e-6):
            status = "feasible"
            fes_vertices.append(sol) # Αποθηκεύουμε την εφικτή κορυφή
            #κάνουμε έλεγχο για το αν είναι εφικτή η λύση A * sol <=b ελέγχει όλους
τους περιορισμούς περασμένους στον πίνακα A
        else:
            status = "infeasible"
        Z_val = 8 * sol[0] + 5 * sol[1] + 4 * sol[2] # Υπολογίζουμε την τιμή του Z
        all_vertices.append(sol)
```

```
        all_statuses.append(status)
        all_Z.append(Z_val)
        if status == "feasible":
             fes_Z.append(Z_val) # Αποθηκεύουμε την εφικτή τιμή Z
    except np.linalg.LinAlgError:
        # Σύστημα μη αντιστρέψιμο (γραμμικά εξαρτημένο)
        all_vertices.append([np.nan, np.nan, np.nan])
        all_statuses.append("singular")
        all_Z.append(np.nan)

# Τώρα βρίσκουμε για κάθε κορυφή πόσοι περιορισμοί είναι ενεργοί
degenerate_flags = []
Z_values = []

for v in all_vertices :
    # Ένας περιορισμός είναι ενεργός αν A[i] @ x == b[i]
    active = np.isclose(A @ v, b, atol=1e-6) # ελέγχουμε αν είναι κοντά στην τιμή b
    # αντίστοιχο του active = np.array([np.isclose(A[i] @ v, b[i], atol=1e-6) for i
in range(len(A))]) η numpy είναι πιο γρήγορη
    num_active = np.sum(active) #   active = διάνυσμα boolean τιμών, άρα num_active
= πλήθος True τιμών
    degenerate_flags.append(num_active > 3) # αν είναι περισσότεροι από 3 τότε είναι
εκφυλισμένο
    # Υπολογισμός Z = 8x1 + 5x2 + 4x3
    Z = 8*v[0] + 5*v[1] + 4*v[2]
    Z_values.append(Z)
# Δημιουργία DataFrame με όλα τα αποτελέσματα
df_all = pd.DataFrame(all_vertices, columns=["x1", "x2", "x3"])
df_all["Constraints"] = all_combinations
df_all["Status"] = all_statuses
df_all["Degenerate"] = degenerate_flags
df_all["Z"] = all_Z
print("\n Κορυφές:")
print(df_all)
print("\n\n")

# Δημιουργούμε πίνακα αποτελεσμάτων
df = pd.DataFrame(fes_vertices, columns=["x1", "x2", "x3"])
df["Z"] = fes_Z
print("Vertices of the feasible region:")
print(df)
```

(β) Προσθέστε μεταβλητές χαλάρωσης στο σύστημα ανισώσεων του παραπάνω προβλήματος και βρείτε όλες τις βασικές (εφικτές και μη-εφικτές) λύσεις για το μη ομογενές σύστημα εξισώσεων που δημιουργείται. Εντοπίστε (αν υπάρχουν) τις εκφυλισμένες βασικές λύσεις.

Προσθέτουμε μεταβλητές χαλάρωσης:

$$\begin{Bmatrix} x_1 + x_2 - s_1 = 10 \\ x_1 + x_2 - s_2 = 15 \\ x_1 + x_3 - s_3 = 12 \\ 20x_1 + 10x_2 + 15x_3 + s_4 = 300 \end{Bmatrix}$$

Με τις μεταβλητές: $x_1, x_2, x_3, s_1, s_2, s_3 \geq 0$

Το πρόβλημα έχει 7 μεταβλητές. Θα λύσουμε όλους τους συνδυασμούς 4 εξισώσεων με 4 βασικές μεταβλητές (το πρόβλημα έχει 7 συνολικά μεταβλητές). Κάθε φορά, επιλέγουμε 4 μεταβλητές ως βασικές (οι άλλες 3 μη βασικές, τις θέτουμε ίσες με 0). Λύνουμε το σύστημα για αυτές τις 4. Αν η λύση υπάρχει, την αποθηκεύουμε. Ελέγχουμε αν είναι εφικτή (όλες οι μεταβλητές ≥ 0). Τέλος ελέγχουμε αν είναι εκφυλισμένη (έχει μη βασική μεταβλητή = 0).

Αποτελέσματα:

Basic Solutions:

	x1	x2	x3	s1	s2	s3	s4	Basic Vars	Feasible	Degenerate
0	6.00	9.0	6.000000	5.00	0.000000	0.000000	0.0	[x1, x2, x3, s1]	True	False
1	-4.00	14.0	16.000000	0.00	15.000000	0.000000	0.0	[x1, x2, x3, s2]	False	False
2	5.00	5.0	10.000000	0.00	0.000000	3.000000	0.0	[x1, x2, x3, s3]	True	False
3	3.50	6.5	8.500000	0.00	0.000000	0.000000	37.5	[x1, x2, x3, s4]	True	False
4	12.00	6.0	0.000000	8.00	-9.000000	0.000000	0.0	[x1, x2, s1, s2]	False	False
5	7.50	15.0	0.000000	12.50	0.000000	-4.500000	0.0	[x1, x2, s1, s3]	False	False
6	12.00	15.0	0.000000	17.00	0.000000	0.000000	-90.0	[x1, x2, s1, s4]	False	False
7	20.00	-10.0	0.000000	0.00	-25.000000	8.000000	0.0	[x1, x2, s2, s3]	False	False
8	12.00	-2.0	0.000000	0.00	-17.000000	0.000000	80.0	[x1, x2, s2, s4]	False	False
9	-5.00	15.0	0.000000	0.00	0.000000	-17.000000	250.0	[x1, x2, s3, s4]	False	False
10	24.00	0.0	-12.000000	14.00	-27.000000	0.000000	0.0	[x1, x3, s1, s2]	False	False
11	3.75	0.0	15.000000	-6.25	0.000000	6.750000	0.0	[x1, x3, s1, s3]	False	False
12	-3.00	0.0	15.000000	-13.00	0.000000	0.000000	135.0	[x1, x3, s1, s4]	False	False
13	10.00	0.0	6.666667	0.00	-8.333333	4.666667	0.0	[x1, x3, s2, s3]	False	False
14	10.00	0.0	2.000000	0.00	-13.000000	0.000000	70.0	[x1, x3, s2, s4]	False	False
15	10.00	0.0	15.000000	0.00	0.000000	13.000000	-125.0	[x1, x3, s3, s4]	False	False
16	15.00	0.0	0.000000	5.00	-15.000000	3.000000	0.0	[x1, s1, s2, s3]	False	False
17	12.00	0.0	0.000000	2.00	-15.000000	0.000000	60.0	[x1, s1, s2, s4]	False	False
18	10.00	0.0	0.000000	0.00	-15.000000	-2.000000	100.0	[x1, s2, s3, s4]	False	False
19	0.00	12.0	12.000000	2.00	9.000000	0.000000	0.0	[x2, x3, s1, s2]	True	False
20	0.00	-15.0	30.000000	-25.00	0.000000	18.000000	0.0	[x2, x3, s1, s3]	False	False
21	0.00	3.0	12.000000	-7.00	0.000000	0.000000	90.0	[x2, x3, s1, s4]	False	False
22	0.00	10.0	13.333333	0.00	8.333333	1.333333	0.0	[x2, x3, s2, s3]	True	False
23	0.00	10.0	12.000000	0.00	7.000000	0.000000	20.0	[x2, x3, s2, s4]	True	False
24	0.00	10.0	5.000000	0.00	0.000000	-7.000000	125.0	[x2, x3, s3, s4]	False	False
25	0.00	30.0	0.000000	20.00	15.000000	-12.000000	0.0	[x2, s1, s2, s3]	False	False
26	0.00	15.0	0.000000	5.00	0.000000	-12.000000	150.0	[x2, s1, s3, s4]	False	False
27	0.00	10.0	0.000000	0.00	-5.000000	-12.000000	200.0	[x2, s2, s3, s4]	False	False
28	0.00	0.0	20.000000	-10.00	5.000000	8.000000	0.0	[x3, s1, s2, s3]	False	False
29	0.00	0.0	12.000000	-10.00	-3.000000	0.000000	120.0	[x3, s1, s2, s4]	False	False
30	0.00	0.0	15.000000	-10.00	0.000000	3.000000	75.0	[x3, s1, s3, s4]	False	False
31	0.00	0.0	0.000000	-10.00	-15.000000	-12.000000	300.0	[s1, s2, s3, s4]	False	False

Total Basic Solutions Found: 32

Degenerate Solutions:
Empty DataFrame
Columns: [x1, x2, x3, s1, s2, s3, s4, Basic Vars, Feasible, Degenerate]
Index: []
Total Degenerate Solutions Found: 0

Κώδικας β:

```
import numpy as np
import pandas as pd
import itertools

# Πλήθος μεταβλητών: x1, x2, x3, s1, s2, s3, s4 = 7
# Πλήθος εξισώσεων: 4

# Πίνακας συντελεστών A για το σύστημα εξισώσεων μετά την προσθήκη χαλαρώσεων
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])
var_names = ['x1', 'x2', 'x3', 's1', 's2', 's3', 's4']

basic_solutions = []

# Δοκιμάζουμε όλους τους συνδυασμούς 4 βασικών μεταβλητών από 7
for basic_vars in itertools.combinations(range(7), 4):
    A_basic = A[:, basic_vars]
    try:
        # Λύνουμε το σύστημα για τις βασικές μεταβλητές
        x_basic = np.linalg.solve(A_basic, b)

        # Φτιάχνουμε το πλήρες διάνυσμα λύσης
        full_x = np.zeros(7)
        for i, var_idx in enumerate(basic_vars):
            full_x[var_idx] = x_basic[i]

        # Ελέγχουμε αν είναι εφικτή λύση (όλες οι μεταβλητές ≥ 0)
        feasible = np.all(full_x >= -1e-6)

        # Εκφυλισμένη: αν κάποια βασική μεταβλητή είναι μηδέν (ή πολύ κοντά)
        degenerate = np.any(np.isclose(x_basic, 0.0, atol=1e-6))

        basic_solutions.append({
            'Solution': full_x,
            'Basic Vars': [var_names[i] for i in basic_vars],
            'Feasible': feasible,
            'Degenerate': degenerate
        })

    except np.linalg.LinAlgError:
        continue  # το σύστημα δεν έχει λύση (γραμμικά εξαρτημένο)
```

```
# Δημιουργία DataFrame για εμφάνιση
df_basic = pd.DataFrame([{
    **{var_names[i]: sol['Solution'][i] for i in range(7)},
    'Basic Vars': sol['Basic Vars'],
    'Feasible': sol['Feasible'],
    'Degenerate': sol['Degenerate']
} for sol in basic_solutions])

if __name__ == "__main__":
    print("Basic Solutions:")
    print(df_basic)
    print("\nTotal Basic Solutions Found:", len(basic_solutions))
    #degenerate solutions
    print("\nDegenerate Solutions:")
    print(df_basic[df_basic['Degenerate']].to_string(index=False))
    print("\nTotal Degenerate Solutions Found:",
len(df_basic[df_basic['Degenerate']]))

# Υπολογισμός της αντικειμενικής συνάρτησης Z = 8x1 + 5x2 + 4x3 για κάθε λύση
df_basic["Z"] = 8 * df_basic["x1"] + 5 * df_basic["x2"] + 4 * df_basic["x3"]

# Εύρεση της καλύτερης (εφικτής) λύσης με το ελάχιστο Z
feasible_df = df_basic[df_basic["Feasible"]]
best_idx = feasible_df["Z"].idxmin()  # idxmin επιστρέφει το index της ελάχιστης
τιμής
best_vertex = feasible_df.loc[best_idx]

# Δημιουργία DataFrame μόνο με τα σημαντικά
bvf = pd.DataFrame([{
    "x1": best_vertex["x1"],
    "x2": best_vertex["x2"],
    "x3": best_vertex["x3"],
    "Z": best_vertex["Z"]
}])
```

(γ) Αντιστοιχίστε τις βασικές λύσεις που βρήκατε στο (β) ερώτημα με τις κορυφές του ερωτήματος (α) και τέλος υποδείξτε τη βέλτιστη λύση και βέλτιστη κορυφή του προβλήματος.

Παρατηρούμε ότι, όπως αναμενόμενο, οι δύο μέθοδοι υπολογίζουν τις ίδιες κορυφές με ίδιες τιμές Z.

Βέλτιστη κορυφή:

min Z= 94.5

x1	x2	x3	Z
3.5	6.5	8.5	94.5

Αντιστοίχιση Κορυφών:

--- Matching All Basic Feasible Solutions to Vertices ---
Basic feasible solution #0 matches vertex #14: [6. 9. 6.]
Basic feasible solution #1 matches vertex #5: [-4. 14. 16.]
Basic feasible solution #2 matches vertex #1: [5. 5. 10.]
Basic feasible solution #3 matches vertex #0: [3.5 6.5 8.5]
Basic feasible solution #4 matches vertex #25: [12. 6. -0.]
Basic feasible solution #5 matches vertex #20: [7.5 15. -0.]
Basic feasible solution #6 matches vertex #17: [12. 15. -0.]
Basic feasible solution #7 matches vertex #11: [20. -10. -0.]
Basic feasible solution #8 matches vertex #8: [12. -2. -0.]
Basic feasible solution #9 matches vertex #4: [-5. 15. -0.]
Basic feasible solution #10 matches vertex #24: [24. -0. -12.]
Basic feasible solution #11 matches vertex #19: [3.75 -0.   15.]
Basic feasible solution #12 matches vertex #16: [-3. -0. 15.]
Basic feasible solution #13 matches vertex #10: [10. -0. 6.66666667]
Basic feasible solution #14 matches vertex #7: [10. 0. 2.]
Basic feasible solution #15 matches vertex #3: [10. -0. 15.]
Basic feasible solution #16 matches vertex #30: [15. -0. -0.]
Basic feasible solution #17 matches vertex #27: [12. -0. -0.]
Basic feasible solution #18 matches vertex #13: [10. -0. -0.]
Basic feasible solution #19 matches vertex #23: [0. 12. 12.]
Basic feasible solution #20 matches vertex #18: [0. -15. 30.]
Basic feasible solution #21 matches vertex #15: [-0. 3. 12.]
Basic feasible solution #22 matches vertex #9: [0. 10. 13.33333333]
Basic feasible solution #23 matches vertex #6: [-0. 10. 12.]
Basic feasible solution #24 matches vertex #2: [-0. 10. 5.]
Basic feasible solution #22 matches vertex #9: [0. 10. 13.33333333]
Basic feasible solution #23 matches vertex #6: [-0. 10. 12.]
Basic feasible solution #24 matches vertex #2: [-0. 10. 5.]
Basic feasible solution #25 matches vertex #29: [0. 30. -0.]
Basic feasible solution #26 matches vertex #22: [-0. 15. -0.]
Basic feasible solution #27 matches vertex #12: [-0. 10. -0.]
Basic feasible solution #22 matches vertex #9: [0. 10. 13.33333333]
Basic feasible solution #23 matches vertex #6: [-0. 10. 12.]
Basic feasible solution #24 matches vertex #2: [-0. 10. 5.]
Basic feasible solution #25 matches vertex #29: [0. 30. -0.]
Basic feasible solution #26 matches vertex #22: [-0. 15. -0.]

```
Basic feasible solution #27 matches vertex #12: [-0. 10. -0.]
Basic feasible solution #28 matches vertex #28: [ 0. -0. 20.]
Basic feasible solution #29 matches vertex #26: [-0. -0. 12.]
Basic feasible solution #30 matches vertex #21: [-0. -0. 15.]
Basic feasible solution #31 matches vertex #31: [-0. -0. -0.]
```

Κώδικας:

```
from ex5 import all_vertices
from ex5b import basic_solutions
import pandas as pd
import numpy as np
print("\n--- Matching All Basic Feasible Solutions to Vertices ---")
#print(basic_solutions)
#print(all_vertices)
#remone Nan values from all_vertices
all_vertices = np.array(all_vertices)
all_vertices = all_vertices[~np.isnan(all_vertices).any(axis=1)]
#remone Nan values from basic_solutions
basic_solutions = np.array([sol['Solution'] for sol in basic_solutions])
basic_solutions = basic_solutions[~np.isnan(basic_solutions).any(axis=1)]
for i, sol in enumerate(basic_solutions):
    for j, vertex in enumerate(all_vertices):
        if np.allclose(sol[:3], vertex, atol=1e-6):

            print(f"Basic feasible solution #{i} matches vertex #{j}: {vertex}")
######### ΒΕΛΤΙΣΤΗ ΛΥΣΗ #########
from ex5b import df_basic
# Υπολογισμός της αντικειμενικής συνάρτησης Z = 8x1 + 5x2 + 4x3 για κάθε λύση
df_basic["Z"] = 8 * df_basic["x1"] + 5 * df_basic["x2"] + 4 * df_basic["x3"]

# Εύρεση της καλύτερης (εφικτής) λύσης με το ελάχιστο Z
feasible_df = df_basic[df_basic["Feasible"]]
best_idx = feasible_df["Z"].idxmin()  # idxmin επιστρέφει το index της ελάχιστης
τιμής
best_vertex = feasible_df.loc[best_idx]

# Δημιουργία DataFrame μόνο με τα σημαντικά
bvf = pd.DataFrame([{
    "x1": best_vertex["x1"],
    "x2": best_vertex["x2"],
    "x3": best_vertex["x3"],
    "Z": best_vertex["Z"]
}])
print("\nBest vertex:\n")
print(bvf.to_string(index=False))
print("\n")
```

Άσκηση 6.

Δίνεται το πρόβλημα γραμμικού προγραμματισμού:

$$\max Z = 2x_1 + x_2 + 6x_3 - 4x_4$$

Όταν:

$$\begin{aligned} x_1 + 2x_2 + 4x_3 - x_4 &\leq 6 \\ 2x_1 + 3x_2 - x_3 + x_4 &\leq 12 \\ x_1 + x_3 + x_4 &\leq 2 \\ x_1, x_2, x_3, x_4 &\geq 0 \end{aligned}$$

(α) Εφαρμόστε τον αλγόριθμο Simplex για να βρείτε τη βέλτιστη λύση του, αν υπάρχει. Σε κάθε επανάληψη του αλγορίθμου περιγράψτε συνοπτικά τα βήματα που ακολουθείτε και τις αποφάσεις που παίρνετε μέχρι το επόμενο βήμα.

Θα εφαρμόσουμε τον αλγόριθμο Simplex (LinearProgramming_2025_eclass.pdf σελ 47) σε python.

Βήμα 1: Εισάγονται μεταβλητές χαλάρωσης και δημιουργείται μία πρώτη βασική εφικτή λύση $x = 0, s = b$.

Έχουμε σύστημα της μορφής $Ax = b$, με $x = [x_1 \quad x_2 \quad x_3 \quad x_4 \quad s_1 \quad s_2 \quad s_3]^T$, όπου s1-s3 μεταβλητές χαλάρωσης. Οι μεταβλητές χαλάρωσης (slack) μπαίνουν ως βασικές μεταβλητές και ο κυρίως μεταβλητές τίθενται 0. Αυτό συνεπάγεται ότι οι slack variables, για να επιλύσουν το σύστημα θα πάρουν τις τιμές του διανύσματος b.

Μορφή tableau:

```
--------------------------------------------------------------------------------
Step 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
--------------------------------------------------------------------------------
Εισερχόμενη μεταβλητή: x3
Εξερχόμενη μεταβλητή: x5
Pivot στοιχείο: 4.00
```

Βήμα 2: Η γραμμή Z (ή -Z) ενημερώνεται. Αν δεν υπάρχουν θετικοί συντελεστές στην Z-γραμμή σημαίνει ότι έχουμε βέλτιστη λύση και ο simplex σταματά (Συνθήκη τερματισμού).

```
last_row = tableau[-1, :-1]
      if np.all(last_row <= 0):
            print("Βρέθηκε βέλτιστη λύση (όλοι οι συντελεστές Z ≤ 0).")
            break
```

Βήμα 3: Επιλέγουμε μεταβλητή x_a με **θετικό** συντελεστή στην αντικειμενική συνάρτηση (στην Z-γραμμή), δηλαδή μεταβλητή που μπορεί να **αυξήσει το κέρδος**.

Επιλέγουμε την μεταβλητή με το μέγιστο **Z value,** καθώς η μεταβολή της θα προκαλέσει το μεγαλύτερο κέρδος ανά μονάδα στην Z. Έτσι βρίσκουμε την "pivot column".

```
entering_candidates = np.where(last_row > 0)[0]
entering = entering_candidates[np.argmax(last_row[entering_candidates])]
```

Βήμα 4: Στη στήλη του B−1N που αντιστοιχεί στην xα βρίσκουμε εκείνον τον δείκτη k για τον οποίο:

$$\frac{\left(\mathbf{B}^{-1}\mathbf{b}\right)_k}{\left(\mathbf{B}^{-1}\mathbf{N}\right)_{k\alpha}} = min\left\{\frac{\left(\mathbf{B}^{-1}\mathbf{b}\right)_j}{\left(\mathbf{B}^{-1}\mathbf{N}\right)_{j\alpha}} \,\middle|\, j \in \{1,\ldots,m\}, \left(\mathbf{B}^{-1}\mathbf{N}\right)_{j\alpha} > 0\right\}$$

΄Εστω β ο δείκτης της στήλης για την οποία IB mkβ = 1. Η μεταβλητή xβ ονομάζεται εξερχόμενη μεταβλητή. (LinearProgramming_2025_eclass.pdf σελ 48)

Χρειάζεται να εφαρμόσουμε δηλαδή το **Κριτήριο ελαχίστου λόγου,** διαιρώντας τις τιμές στην στήλη b με την αντίστοιχη τιμή στη στήλη Pivot column για κάθε σειρά της b.

```
# Step 2: Minimum ratio test
        ratios = []
        for i in range(len(tableau) - 1):
            col_val = tableau[i, entering]
            if col_val > 0:
                ratios.append(tableau[i, -1] / col_val)
            else:
                ratios.append(np.inf)

        leaving_row = np.argmin(ratios)
```

Διαλέγουμε ως pivot row την σειρά με τον μικρότερο λόγο (ratio) και επιλέγουμε την βασική μεταβλητή αυτής της σειράς ως **εξερχόμενη μεταβλητή**.

Σε περίπτωση που το πρόβλημα είναι μη φραγμένο σταματάμε:

```
if ratios[leaving_row] == np.inf: # αν ολα αρνητικά => unbounded
            print(" Το πρόβλημα είναι μη φραγμένο.")
            break
```

Βήμα 5: Η εισερχόμενη γίνεται βασική, η εξερχόμενη φεύγει από τη βάση. Επαναλαμβάνεται η απαλοιφή Gauss.

Θέλουμε το pivot element να γίνει 1. Οπότε διαιρούμε όλες τις τιμές της σειράς με το pivot (στοιχείο στη pivot row και pivot column). Ύστερα, θέλουμε οι υπόλοιπες τιμές στην pivot στήλη να γίνου 0 και το επιτυγχάνουμε με γραμμοπράξεις. Αφαιρούμε από τις υπόλοιπες σειρές συντελεστών πολλαπλάσια της σειράς pivot.

```
# Normalize pivot row
        tableau[leaving_row] /= pivot
        # Eliminate pivot column from other rows
        for i in range(len(tableau)):
            if i != leaving_row:
                tableau[i] -= tableau[i, entering] * tableau[leaving_row]
        basic_vars[leaving_row] = entering
```

Αποτελέσματα:

```
--------------------------------------------------------------------------------
Step 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
--------------------------------------------------------------------------------
Εισερχόμενη μεταβλητή: x3
Εξερχόμενη μεταβλητή: x5
Pivot στοιχείο: 4.00

--------------------------------------------------------------------------------
Step 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
--------------------------------------------------------------------------------
Εισερχόμενη μεταβλητή: x1
Εξερχόμενη μεταβλητή: x7
Pivot στοιχείο: 0.75
```

```
--------------------------------------------------------------------------------
Step 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
--------------------------------------------------------------------------------
Βρέθηκε βέλτιστη λύση (όλοι οι συντελεστές Z ≤ 0).
Βέλτιστη λύση:
x1 = 0.67
x2 = 0.00
x3 = 1.33
x4 = 0.00
s1 = 0.00
s2 = 12.00
s3 = 0.00
Z = 9.33
```

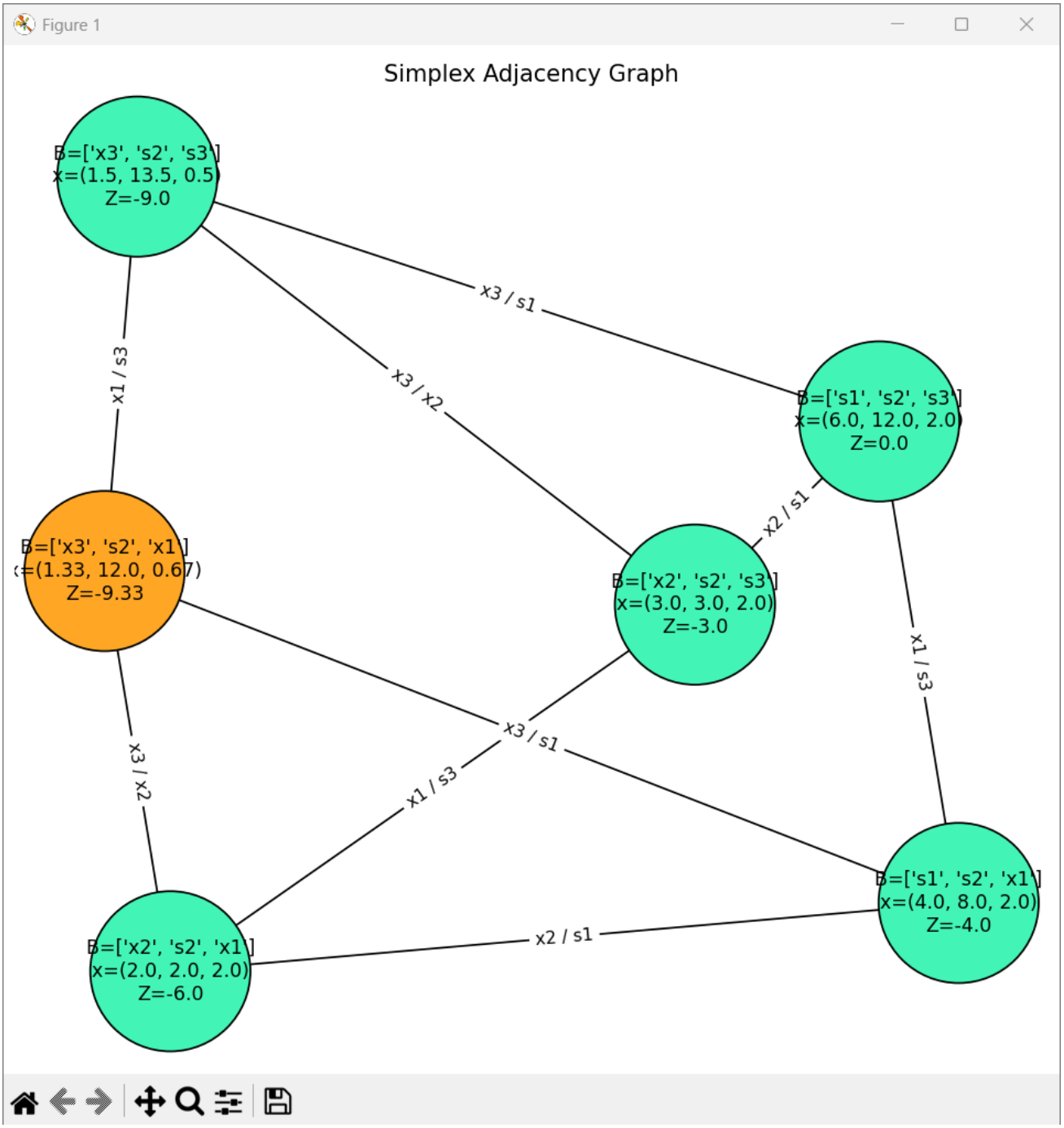
Κώδικας:

```
import numpy as np

# Problem definition
# max Z = 2x1 + x2 + 6x3 - 4x4
# Subject to:
# x1 + 2x2 + 4x3 - x4       ≤ 6
# 2x1 + 3x2 -  x3 + x4      ≤ 12
# x1        +  x3 + x4      ≤ 2

# Add slack variables: s1, s2, s3
A = np.array([
    [1, 2, 4, -1, 1, 0, 0],
    [2, 3, -1, 1, 0, 1, 0],
    [1, 0, 1, 1, 0, 0, 1]
], dtype=float)

b = np.array([6, 12, 2], dtype=float)
c = np.array([2, 1, 6, -4, 0, 0, 0], dtype=float)  # Objective function

'''Εισάγονται μεταβλητές χαλάρωσης και δημιουργείται μία πρώτη βασική
εφικτή λύση: (x = 0,xS = b)
'''
# Create initial tableau
tableau = np.zeros((A.shape[0] + 1, A.shape[1] + 1)) #rows + 1, columns + 1
tableau[:-1, :-1] = A # Coefficients of constraints
tableau[:-1, -1] = b # Right-hand side
tableau[-1, :-1] = c # Coefficients of objective function
tableau[-1, -1] = 0 # Objective function value

basic_vars = [4, 5, 6]  # Slack variables initially in basis

def print_tableau(tab, basic_vars, step_desc):
    print("\n" + "-"*80)
    print(step_desc)
    print("-"*80)
    header = ["x1", "x2", "x3", "x4", "s1", "s2", "s3", "b"]
    print("    | " + " | ".join(f"{h:>6}" for h in header))
    print("="*80)
    print("-Z  | " + " | ".join(f"{v:6.2f}" for v in tab[-1]))
    print("-"*80)
    for i in range(len(tab) - 1):
        var_names = ["x1", "x2", "x3", "x4", "s1", "s2", "s3"]
        row_label = var_names[basic_vars[i]]

        #row_label = f"x{basic_vars[i]+1}"
        print(f"{row_label:>3} | " + " | ".join(f"{v:6.2f}" for v in tab[i]))
```

```
    print("-"*80)
    # print("-Z  | " + " | ".join(f"{v:6.2f}" for v in tab[-1]))
    # print("-"*80)

def simplex(tableau, basic_vars):
    iteration = 0
    while True:
        print_tableau(tableau, basic_vars, f"Step {iteration}")

        last_row = tableau[-1, :-1]
        if np.all(last_row <= 0):
            print("Βρέθηκε βέλτιστη λύση (όλοι οι συντελεστές Z ≤ 0).")
            break


        # Step 1: Entering variable
        entering_candidates = np.where(last_row > 0)[0]
        entering = entering_candidates[np.argmax(last_row[entering_candidates])] #
is index of the entering variable

        print(f"Εισερχόμενη μεταβλητή: x{entering + 1}")

        # Step 2: Minimum ratio test
        ratios = []
        for i in range(len(tableau) - 1):
            col_val = tableau[i, entering]
            if col_val > 0:
                ratios.append(tableau[i, -1] / col_val)
            else:
                ratios.append(np.inf)

        leaving_row = np.argmin(ratios)
        if ratios[leaving_row] == np.inf: # αν ολα αρνητικά => unbounded
            print(" Το πρόβλημα είναι μη φραγμένο.")
            break

        pivot = tableau[leaving_row, entering]
        print(f"Εξερχόμενη μεταβλητή: x{basic_vars[leaving_row] + 1}")
        print(f"Pivot στοιχείο: {pivot:.2f}")

        # Normalize pivot row
        tableau[leaving_row] /= pivot

        # Eliminate pivot column from other rows
        for i in range(len(tableau)):
            if i != leaving_row:
                tableau[i] -= tableau[i, entering] * tableau[leaving_row]

        basic_vars[leaving_row] = entering
```

```
        iteration += 1
    return tableau, basic_vars

# Run the algorithm
# Run the algorithm
final_tableau, final_basic_vars = simplex(tableau.copy(), basic_vars.copy())

# Ονόματα μεταβλητών (με βάση τις στήλες)
var_names = ["x1", "x2", "x3", "x4", "s1", "s2", "s3"]

# Αρχικοποίηση λύσης με μηδενικά
solution = {var: 0.0 for var in var_names}

# Για κάθε βασική μεταβλητή, πάρε την τιμή από τη στήλη b (τελευταία)
for i, var_index in enumerate(final_basic_vars):
    var_name = var_names[var_index]
    solution[var_name] = final_tableau[i, -1]

# Εκτύπωση τιμών μεταβλητών
print("Βέλτιστη λύση:")
for var in var_names:
    print(f"{var} = {solution[var]:.2f}")

# Τιμή της αντικειμενικής συνάρτησης Z
print(f"Z = {-1*final_tableau[-1, -1]:.2f}")
```

(β) Εφαρμόστε όλες τις εναλλακτικές επιλογές που μπορεί να έχετε σε κάθε βήμα επιλογής της εισερχόμενης ή εξερχόμενης μεταβλητής στις επαναλήψεις του αλγορίθμου και δημιουργήστε έναν γράφο (τον Simplex adjacency graph) με τα βήματα (κορυφές) του αλγορίθμου και τις εναλλακτικές διαδρομές που θα μπορούσε να ακολουθήσει μέχρι τη βέλτιστη κορυφή (εφ΄όσον αυτή υπάρχει).



Κώδικας:

```
# Άσκηση 6 (β)

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

# Ορισμός προβλήματος (ίδιο με 6α)
A = np.array([
    [1, 2, 4, -1, 1, 0, 0],
    [2, 3, -1, 1, 0, 1, 0],
    [1, 0, 1, 1, 0, 0, 1]
], dtype=float)

b = np.array([6, 12, 2], dtype=float)
c = np.array([2, 1, 6, -4, 0, 0, 0], dtype=float)

var_names = ["x1", "x2", "x3", "x4", "s1", "s2", "s3"]

# Αρχικοποίηση
initial_tableau = np.zeros((A.shape[0] + 1, A.shape[1] + 1))
initial_tableau[:-1, :-1] = A
initial_tableau[:-1, -1] = b
initial_tableau[-1, :-1] = c
initial_tableau[-1, -1] = 0
initial_basis = [4, 5, 6]

# Κόμβοι: καταστάσεις με (basis, values, Z)
G = nx.DiGraph()
visited = set()


def tableau_key(basis, tableau):
    return tuple(basis), tuple(np.round(tableau[:, -1], 4)), round(tableau[-1, -1],
4)


def explore(tableau, basis):
    # Δημιουργούμε ένα μοναδικό κλειδί για την τρέχουσα κατάσταση (βάση και τιμές b
+ Z)
    key = tableau_key(basis, tableau)
    # Αν το έχουμε ήδη επισκεφθεί, δεν το ξαναεπεξεργαζόμαστε
    if key in visited:
        return
    visited.add(key)
    # ετικέτα για τον κόμβο
    label = f"B={[var_names[i] for i in basis]}\n"
    label += f"x={tuple(round(tableau[i, -1], 2) for i in range(len(basis)))}\n"
    label += f"Z={round(tableau[-1, -1], 2)}"
```

```
    # Προσθέτουμε τον κόμβο στον γράφο G
    # Αν η γραμμή Z έχει όλα τα στοιχεία ≤ 0 → είναι βέλτιστη λύση (χρωματίζεται
πορτοκαλί)
    G.add_node(key, label=label, color="#ffa724" if np.all(tableau[-1, :-1] <= 0)
else "#42f5b6")
    # υποψήφιες εισερχόμενες μεταβλητές = αυτές με θετικό συντελεστή στο Z
    last_row = tableau[-1, :-1]
    entering_vars = [j for j in range(len(last_row)) if last_row[j] > 0] #
Εξετάζουμε τις υποψήφιες εισερχόμενες μεταβλητές
    for entering in entering_vars:
        # κάνουμε το κριτήριο ελαχίστου λόγου
        ratios = []
        for i in range(len(tableau) - 1):
            aij = tableau[i, entering]
            if aij > 0:
                ratios.append((tableau[i, -1] / aij, i))
        if not ratios:# Αν δεν υπάρχει έγκυρη επιλογή, πάμε στο επόμενο entering
            continue

        min_ratio = min(ratios)[0]
         # Επιλέγουμε όλες τις γραμμές που έχουν min_ratio (ισοπαλία => πολλαπλές
πορείες)
        for ratio, leaving_row in ratios:
            if np.isclose(ratio, min_ratio):
                new_tableau = tableau.copy()
                pivot = new_tableau[leaving_row, entering]# κάνουμε κανονικοποίηση
της γραμμής εξερχόμενης μεταβλητής
                new_tableau[leaving_row] /= pivot
                for i in range(len(new_tableau)):
                    if i != leaving_row:
                        new_tableau[i] -= new_tableau[i, entering] *
new_tableau[leaving_row]
                new_basis = basis.copy()
                # Ενημερώνουμε τη βάση: η εισερχόμενη μεταβλητή μπαίνει στη θέση της
εξερχόμενης
                new_basis[leaving_row] = entering
                new_key = tableau_key(new_basis, new_tableau) ## Δημιουργούμε νέο
μοναδικό κλειδί για τον επόμενο κόμβο
                # Προσθέτουμε ακμή στον γράφο από την προηγούμενη κατάσταση στη νέα
                G.add_edge(key, new_key, label=f"{var_names[entering]} /
{var_names[basis[leaving_row]]}")
                explore(new_tableau, new_basis)  # Επαναληπτικά εξερευνούμε τον νέο
κόμβο
explore(initial_tableau.copy(), initial_basis.copy())
# Σχεδίαση γράφου
pos = nx.spring_layout(G, seed=42)
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")

plt.figure(figsize=(14, 8))
nx.draw_networkx_nodes(G, pos, node_color = node_colors, edgecolors = "black",
node_size = 7000)
nx.draw_networkx_edges(G, pos)
nx.draw_networkx_labels(G, pos, labels=labels, font_size = 10)
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size = 9)
plt.title("Simplex Adjacency Graph")
plt.axis("off")
plt.tight_layout()
plt.show()
```