Γραμμική και Συνδυαστική Βελτιστοποίηση

Εργασία 1

Ονοματεπώνυμο: Γεώργιος Παπουτσάς

ΑΜ: 1083738

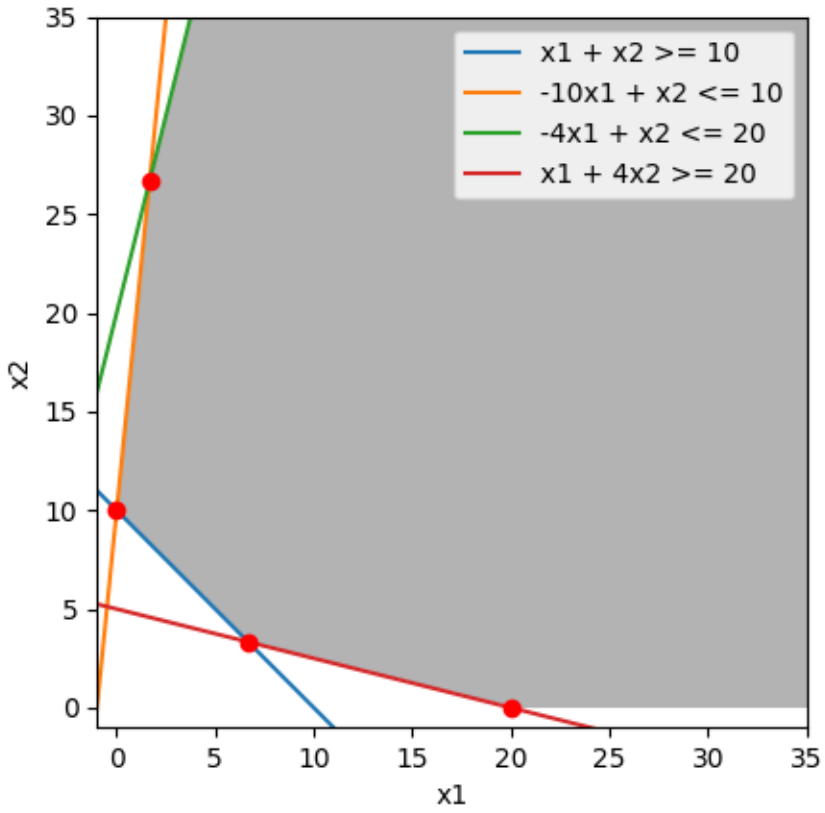
Έτος: 4ο

Περιεχόμενα

Άσκηση 1:

Ερώτημα (α):

Σύμφωνα με τους περιορισμούς που δίνονται σχεδιάζουμε όλες τις ευθείες που τους αντιστοιχούν και έτσι λαμβάνουμε την εξής εφικτή περιοχή:



Εικόνα 1: Εφικτή περιοχή του προβλήματος

Παρατίθεται παρακάτω ο κώδικας για τον σχεδιασμό της εφικτής περιοχής:

```
# Define the constraints
d = np.linspace(0, 35, 300)
x1, x2 = np.meshgrid(d, d)

x = np.linspace(-100, 100, 3000)

constraint1 = x1 + x2 >= 10
constraint2 = -10 * x1 + x2 <= 10
constraint3 = -4 * x1 + x2 <= 20
constraint4 = x1 + 4 * x2 >= 20
constraint5 = x1 >= 0
constraint6 = x2 >= 0
feasible_region = constraint1 & constraint2 & constraint3 & constraint4 & constraint5 & constraint6

# Feasible region
plt.imshow( (feasible_region).astype(int),
            extent=(x1.min(), x1.max(), x2.min(), x2.max()), origin="lower", cmap="Greys", alpha = 0.3)
```

Παρατηρούμε ότι η εφικτή περιοχή δεν είναι φραγμένη και επεκτείνεται στο άπειρο.

Οι κορυφές του προβλήματος είναι:

```
Κορυφές:
 [[-0.         10.        ]
 [ 1.66666667 26.66666667]
 [ 6.66666667  3.33333333]
 [20.          0.        ]]
```

Οι κορυφές βρέθηκαν παρατηρώντας ποιες ευθείες τέμνονται και λύνοντας το αντίστοιχο 2x2 σύστημα εκτός από την τελευταία κορυφή που φαίνεται από το σχήμα αμέσως. Συγκεκριμένα:

```
## Intersections
# constraint1 & constraint2
a1 = np.array([[1, 1], [-10, 1]])
b1 = np.array([10, 10])
intersection1 = np.linalg.solve(a1, b1)

# constraint2 & constraint3
a2 = np.array([[-10, 1], [-4, 1]])
b2 = np.array([10, 20])
intersection2 = np.linalg.solve(a2, b2)

# constraint1 & constraint4
a3 = np.array([[1, 1], [1, 4]])
b3 = np.array([10, 20])
intersection3 = np.linalg.solve(a3, b3)

# constraint4 & constraint5
intersection4 = [20, 0]

intersections = np.array([intersection1, intersection2, intersection3, intersection4])
```

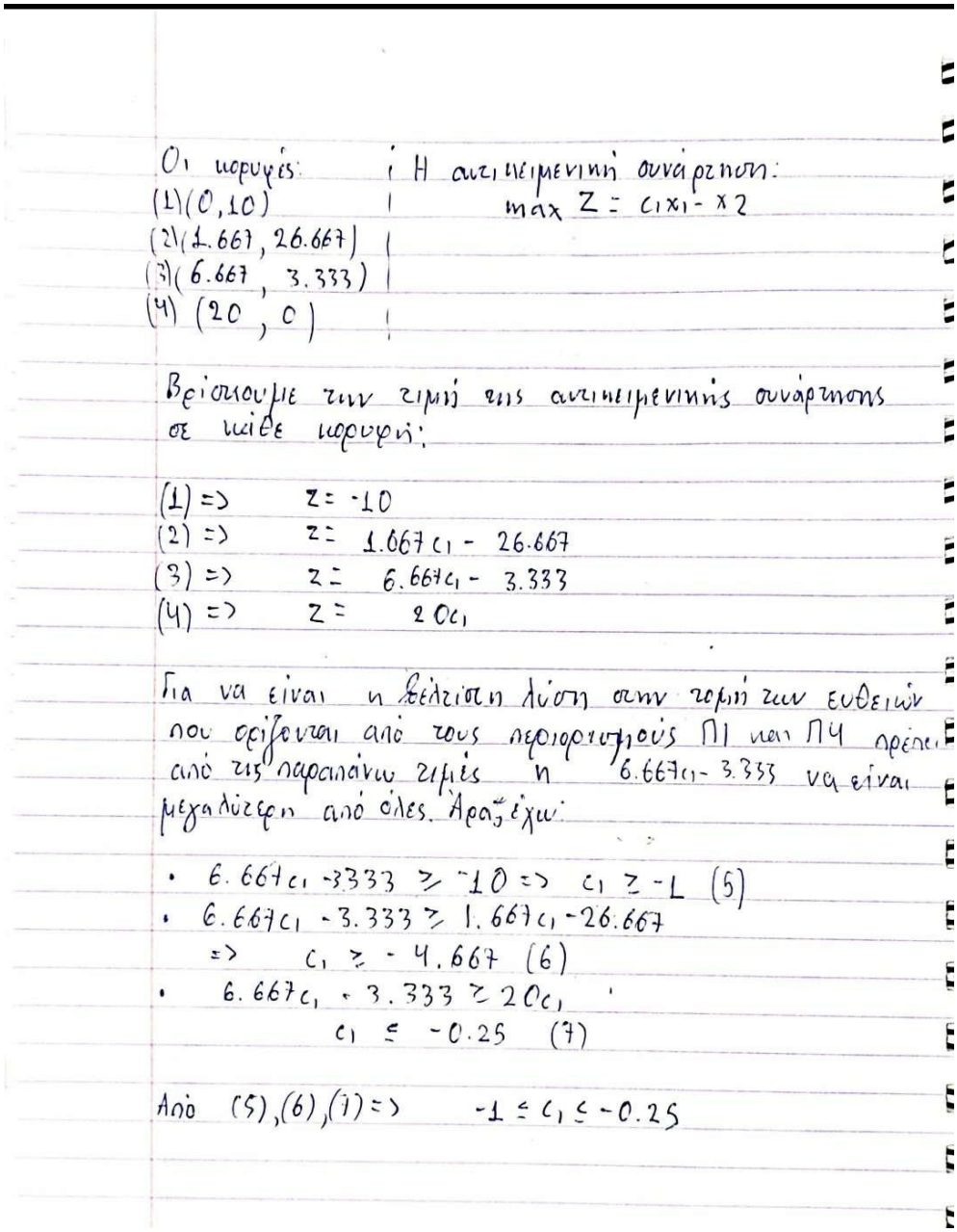
Η κορυφή που αντιστοιχεί στην μικρότερη τιμή της αντικειμενικής συνάρτησης είναι η (1.667, 26.667) με τιμή -23.333 και βρέθηκε με την εξής συνάρτηση:

```python
def find_min_or_max(fz, intersections:np.array, type:str)->tuple:
    """
    Finds the minimum or maximum value of a function at the given intersections.

    Args:
        fz1: The function to be optimized.
        intersections: The intersections of the constraints.
        type: The type of the optimization. It can be either "min" or "max".
    Returns:
        A tuple containing the minimum or maximum value of the function and the point where it occurs.
    """

    assert type in ["min", "max"], "The type must be either 'min' or 'max'."
    if type == "max":

        index = 0
        maxima = -np.inf
        for i, intersection in enumerate(intersections):
            z = fz(intersection[0], intersection[1])
            if z > maxima:
                maxima = z
                index = i
        print(f"Κορυφές:\n {intersections}")
        return maxima, intersections[index]
    elif type == "min":

        index = 0
        minima = np.inf
        for i, intersection in enumerate(intersections):
            z = fz(intersection[0], intersection[1])
            if z < minima:
                minima = z
                index = i
        print(f"Κορυφές:\n {intersections}")
        return minima, intersections[index]
```

Ερώτημα (β):

Η κορυφή που αντιστοιχεί στην μικρότερη τιμή της αντικειμενικής συνάρτησης είναι η (0, 10) με τιμή -10 και βρέθηκε με παρόμοιο τρόπο.

Ερώτημα (γ):



Οι κορυφές:
(1) (0, 10)
(2) (1.667, 26.667)
(3) (6.667, 3.333)
(4) (20, 0)

Η αντικειμενική συνάρτηση:
max Z = $c_1 x_1 - x_2$

Βρίσκουμε την τιμή της αντικειμενικής συνάρτησης σε κάθε κορυφή:

(1) => Z = -10
(2) => Z = $1.667 c_1 - 26.667$
(3) => Z = $6.667 c_1 - 3.333$
(4) => Z = $20 c_1$

Για να είναι η βέλτιστη λύση στην τομή των ευθειών που ορίζονται από τους περιορισμούς Π1 και Π4 πρέπει από τις παραπάνω τιμές η $6.667 c_1 - 3.333$ να είναι μεγαλύτερη από όλες. Άρα έχω:

- $6.667 c_1 - 3333 \geq -10 \Rightarrow c_1 \geq -1$ (5)
- $6.667 c_1 - 3.333 \geq 1.667 c_1 - 26.667$
  $\Rightarrow c_1 \geq -4.667$ (6)
- $6.667 c_1 - 3.333 \geq 20 c_1$
  $c_1 \leq -0.25$ (7)

Από (5), (6), (7) => $-1 \leq c_1 \leq -0.25$

Άσκηση 2:

Ερώτημα (α):

Ως μεταβλητές απόφασης x_1, x_2 ορίζουμε τα κέρδη παραγωγής του χυμού μήλου και πορτοκαλιού αντίστοιχα.

Τα έσοδα $E\sigma = 5(x_1+x_2)$ αφού η εταιρεία πουλάει τον φρουτοχυμό 5χ.μ. ανεξαρτήτως αναλογίας.

Τα έξοδα παραγωγής $E\xi = 2x_1+x_2$ αφού το κόστος για 1 λίτρο πορτοκαλιού και μήλου είναι 2 χ.μ. και 1 χ.μ. αντίστοιχα.

Άρα η αντικειμενική συνάρτηση κέρδους $K = E\sigma - E\xi$:

$$\max Z = 3x_1 + 4x_2$$

Σύμφωνα με την εκφώνηση λαμβάνουμε τους εξής περιορισμούς:

(Π1) $x_1 + x_2 \leq 500$
(Π2) $x_1 + x_2 \geq 400$
(Π3) $x_1 - 3x_2 \leq 0$
(Π4) $0.6x_1 - 0.4x_2 \geq 0$
(Π5) $x_2 \leq 250$
$x_1, x_2 \geq 0$

Αναλυτικότερα:

Ο (Π1) προκύπτει από το από την εβδομαδιαία δυναμικότητα της εταιρείας για το προϊόν που δεν μπορεί να ξεπερνά τα 500 λίτρα. Ο (Π2) προκύπτει από την ζήτηση αγοράς του προϊόντος που είναι τουλάχιστον 400 λίτρα κάθε εβδομάδα. Ο (Π3) προκύπτει επειδή ο φρουτοχυμός περιέχει το πολύ 3 μέρη πορτοκαλιού και τουλάχιστον 1 μέρος μήλου δηλαδή

$$x_1 \leq 3x_2$$

Ο (Π4) προκύπτει αφού η εταιρεία θα παράξει φρουτοχυμό με περιεκτικότητα 40% πορτοκαλιού δηλαδή

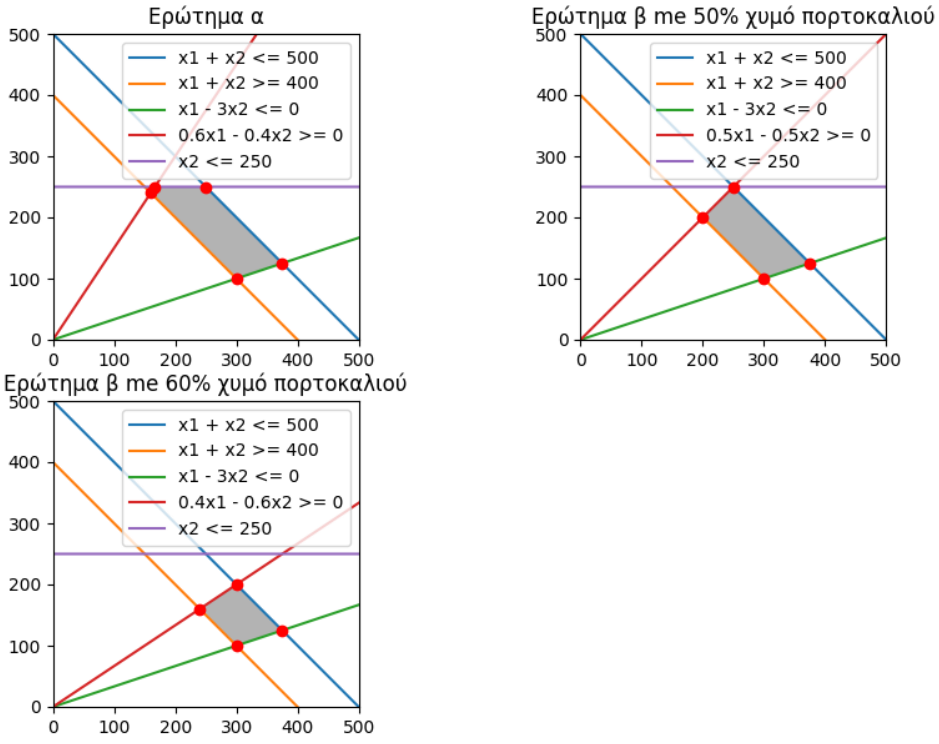
$$x_1 \geq 0.4(x_1 + x_2)$$

την επόμενη εβδομάδα. Τέλος επειδή η εταιρεία μπορεί να διαθέσει μέχρι 250 λίτρα χυμό μήλου έχουμε και τον (Π5).

Η εφικτή περιοχή του προβλήματος με τους παραπάνω περιορισμούς φαίνεται στην Εικόνα 2. Η εύρεση των κορυφών και της βέλτιστης λύσης έγινε με τον ίδιο κώδικα που αξιοποιήθηκε στην Άσκηση 1 και λαμβάνουμε:

```
--------------------Ερώτημα α--------------------
Κορυφές:
 [[375.         125.        ]
 [250.         250.        ]
 [300.         100.        ]
 [160.         240.        ]
 [166.66666667 250.        ]]
The maximum value of the function is: 1750.0 at point [250. 250.]
```

Αντικαθιστώντας τη βέλτιστη λύση στους περιορισμούς, παρατηρούμε ότι οι (Π1), (Π5) ικανοποιούνται οριακά αφού ισχύει η ισότητα και οι υπόλοιποι περιορισμοί ικανοποιούνται χαλαρά.



Εικόνα 2: Εφικτές περιοχές για την Άσκηση 2

Ερώτημα (β):

Οι εφικτές περιοχές για κάθε περίπτωση περιεκτικότητας χυμού πορτοκαλιού φαίνονται στην Εικόνα 2 και οι κορυφές και οι βέλτιστες λύσεις είναι οι εξής:

```
----------------------Ερώτημα β με 50% χυμό πορτοκαλιού----------------------
Κορυφές:
 [[375. 125.]
 [250. 250.]
 [300. 100.]
 [200. 200.]]
The maximum value of the function is: 1750.0 at point [250. 250.]
----------------------Ερώτημα β με 60% χυμό πορτοκαλιού----------------------
Κορυφές:
 [[375. 125.]
 [300. 100.]
 [300. 200.]
 [240. 160.]]
The maximum value of the function is: 1700.0 at point [300. 200.]
```

Παρατηρούμε ότι στην 1η περίπτωση με 50% χυμό πορτοκαλιού το η βέλτιστη λύση δεν αλλάζει ενώ με 60% η βέλτιστη λύση μειώνεται, αρνητικό φαινόμενο διότι επιθυμούμε μεγιστοποίηση τους κέρδους.

Άσκηση 3:

Ως μεταβλητές απόφασης $x_i, i = 1, \dots, 6$ διαλέγουμε τις βδομάδες που μειώνονται για την ολοκλήρωση κάθε εργασίας. Σύμφωνα με τον γράφο η μέγιστη διάρκεια όλων των εργασιών είναι 10+5+8=23 εβδομάδες. Άρα για να μειωθεί ο συνολικός χρόνος διεκπεραίωσης σε 19 εβδομάδες πρέπει:

$$\sum_{i=1}^{6} x_i = 23 - 19 = 4 \ (\Pi 1)$$

Επίσης η μέγιστη μείωση κάθε εργασίας είναι:

$$x_1 \leq 2 \ (\Pi 2)$$

$$x_2 \leq 5 \ (\Pi 3)$$

$$x_3 = 0 \ (\Pi 4)$$

$$x_4 \leq 2 \ (\Pi 5)$$

$$x_5 \leq 2 \ (\Pi 6)$$

$$x_6 \leq 3 \ (\Pi 7)$$

Τέλος, η αντικειμενική συνάρτηση ελαχιστοποίησης του κόστους είναι:

$$min\ Z = 6x_1 + 10x_2 + 8x_4 + 8x_5 + 3x_6$$

Άσκηση 4:

(Π1):

Πρώτη απόδειξη:

Για κάθε $x, y \in X$ δηλαδή $x, y \in X_1 \cap X_2$

$\Longrightarrow x, y \in X_1$ και $x, y \in X_2$

$\Longrightarrow \lambda x + 1(1-\lambda)y \in X_1$ και $\lambda x + (1-\lambda)y \in X_2$

$\Longrightarrow \lambda x + 1(1-\lambda)y \in X_1 \cap X_2$

Άρα η τομή X των δύο κυρτών συνόλωνX_1, X_2 είναι και αυτό κυρτό σύνολο.

Δεύτερη απόδειξη:

Έστω X_1 ο κλειστός μοναδιαίος κύκλος στο $\mathbb{R}^2$, δηλαδή $X_1 = \{(x,y)\epsilon\mathbb{R}^2 : x^2 + y^2 \leq 1\}$ και έστω X_2 το κλειστό μοναδιαίο τετράγωνο στο $\mathbb{R}^2$, δηλαδή $X_2 = \{(x,y)\epsilon\mathbb{R}^2 : |x| \leq 1, |y| \leq 1\}$

Το X_1, X_2 είναι κυρτά σύνολα. Όμως, η ένωση τους, $X_1 \cup X_2$ περιέχει σημεία εξωτερικά του τετραγώνου και εσωτερικά του κύκλου όπως το σημείο $(\sqrt{2}, 0)$. Συνεπώς, η ένωση δεν είναι κυρτό σύνολο επειδή περιέχει ένα ευθύγραμμο τμήμα που συνδέει δύο σημεία

$\left(\sqrt{2}\,, 0\right) \in X_1 \cup X_2$ και

$\left(-\sqrt{2}\,, 0\right) \in X_1 \cup X_2$ αλλά όλο το ευθύγραμμο τμήμα δεν ανήκει στο $X_1 \cup X_2$

(Π2):

Για να δείξουμε ότι το M $=\{(x,y)\epsilon\mathbb{R}^2{}_{++} : xy \geq k, k \in \mathbb{R}\}$ είναι κυρτό πρέπει να δείξουμε ότι για οποιαδήποτε δύο σημεία $p_1 = (x_1, y_1)$ και $p_2 = (x_2, y_2)$ του M, το ευθύγραμμο τμήμα που τα ενώνει είναι ολόκληρο μέσα στο M.

Η εξίσωση του ευθύγραμμου τμήματος που περνά από τα p_1, p_2:

$p = \lambda p_1 + (1-\lambda)p_2, \qquad \lambda\epsilon[0,1]$

Αντικαθιστώντας τις συντεταγμένες των επιμέρους σημείων:

$p = (\,\lambda x_1 + (1-\lambda)x_2, \lambda y_1 + (1-\lambda)y_2\,)$

Τώρα πρέπει να δείξουμε ότι $xy \geq k, k \in \mathbb{R}$ για οποιοδήποτε σημείο στο ευθύγραμμο τμήμα p. Άρα πρέπει:

$(\lambda x_1 + (1-\lambda)x_2) \cdot (\lambda y_1 + (1-\lambda)y_2) \geq k$

$\Rightarrow \lambda^2 x_1 y_1 + (\lambda - \lambda^2)x_1 y_2 + (\lambda - \lambda^2)x_2 y_1 + (1-\lambda)^2 x_2 y_2 \geq k \quad (1)$

Έχω:

- $\lambda^2 \epsilon [0,1]$
- $(\lambda - \lambda^2)\ \epsilon [-1,1]$
- $(1 - \lambda^2)\ \epsilon [0,1]$
- $\lambda^2 x_1 y_1 \geq \lambda^2 k$
- $(1-\lambda)^2 x_2 y_2 \geq (1-\lambda)^2 k$
- $\frac{x_1 y_2}{x_2 y_1} + \frac{x_2 y_1}{x_1 y_2} \geq 2 \Rightarrow (x_1 y_2)^2 + (x_2 y_1)^2 \geq 2 x_1 y_2 x_2 y_1 \Rightarrow (x_1 y_2 + x_2 y_1)^2 \geq 4 x_1 y_2 x_2 y_1 \Rightarrow (x_1 y_2 + x_2 y_1) \geq 2k$

Άρα ισχύει η (1) και το Μ είναι κυρτό σύνολο.

Άσκηση 5:

Ερώτημα (α):

Έχουμε τέσσερις μεταβλητές απόφασης και 7 περιορισμούς. Δηλαδή είμαστε στο $\mathbb{R}^4$ άρα μία κορυφή είναι τομή 4 υπερεπιπέδων. Έχουμε 7 υπερεπίπεδα άρα $\binom{7}{4} = 35$ πιθανές κορυφές.

Παίρνοντας όλους τους δυνατούς συνδυασμούς, αυτοί που βγάζουν λύση είναι 30 και οι εξής:

Number of vertices of polytope: 30

hyperplanes	x1	x2	x3	x4
(0, 1, 2, 3)	0.0	3.0	-0.6	1.4
(0, 1, 2, 4)	1.5	0.0	0.3	0.8
(0, 1, 2, 5)	1.0	1.0	0.0	1.0
(0, 1, 2, 6)	3.5	-4.0	1.5	0.0
(0, 1, 3, 4)	0.0	-0.0	-0.0	2.0
(0, 1, 3, 5)	0.0	-0.0	0.0	2.0
(0, 1, 3, 6)	0.0	10.0	-2.0	0.0
(0, 1, 4, 5)	0.0	-0.0	-0.0	2.0
(0, 1, 4, 6)	2.5	-0.0	0.5	0.0
(0, 1, 5, 6)	2.0	2.0	0.0	0.0
(0, 2, 3, 5)	0.0	3.0	0.0	0.5
(0, 2, 3, 6)	0.0	3.0	0.33	0.0
(0, 2, 4, 5)	1.5	0.0	0.0	1.25
(0, 2, 4, 6)	1.5	0.0	0.83	0.0
(0, 2, 5, 6)	-1.0	5.0	0.0	0.0
(0, 3, 4, 5)	0.0	-0.0	-0.0	2.0
(0, 3, 4, 6)	0.0	-0.0	1.33	0.0
(0, 3, 5, 6)	0.0	4.0	0.0	0.0
(0, 4, 5, 6)	4.0	0.0	0.0	0.0
(1, 2, 3, 5)	0.0	3.0	0.0	2.0
(1, 2, 3, 6)	0.0	3.0	-2.0	0.0
(1, 2, 4, 5)	1.5	0.0	-0.0	0.5
(1, 2, 4, 6)	1.5	0.0	-0.5	0.0
(1, 2, 5, 6)	2.0	-1.0	0.0	0.0
(1, 3, 4, 5)	0.0	0.0	0.0	2.0
(1, 3, 4, 6)	0.0	0.0	-2.0	0.0
(1, 4, 5, 6)	2.0	0.0	0.0	0.0
(2, 3, 5, 6)	0.0	3.0	0.0	0.0
(2, 4, 5, 6)	1.5	0.0	0.0	0.0
(3, 4, 5, 6)	0.0	0.0	0.0	0.0

Ο κώδικας για αυτό το κομμάτι:

```python
# Solving all possible combinations of the constraints
combs = nCr_combs(A, len(c))
indices = np.linspace(0, len(b)-1, len(b), dtype=int)
indices = nCr_combs(indices, len(c))
vertices = []
for i in range(len(combs)):
    augmented = np.array(combs[i])
    a = augmented[:, :-1]
    b_ = augmented[:, -1]
    try:
        vertices.append((np.round(np.linalg.solve(a, b_), 2), indices[i]))
    except np.linalg.LinAlgError:
        continue
```

Οι κορυφές που ανήκουν στην εφικτή περιοχή, δηλαδή τηρούν τους περιορισμούς του προβλήματος μας, είναι 15 και οι εξής:

```
Number of vertices in feasible region: 15
hyperplanes             x1      x2      x3      x4
(0, 1, 2, 4)            1.5     0.0     0.3     0.8
(0, 1, 2, 5)            1.0     1.0     0.0     1.0
(0, 1, 3, 4)            0.0     -0.0    -0.0    2.0
(0, 1, 3, 5)            0.0     -0.0    0.0     2.0
(0, 1, 4, 5)            0.0     -0.0    -0.0    2.0
(0, 2, 3, 5)            0.0     3.0     0.0     0.5
(0, 2, 3, 6)            0.0     3.0     0.33    0.0
(0, 2, 4, 6)            1.5     0.0     0.83    0.0
(0, 3, 4, 5)            0.0     -0.0    -0.0    2.0
(0, 3, 4, 6)            0.0     -0.0    1.33    0.0
(1, 2, 4, 5)            1.5     0.0     -0.0    0.5
(1, 3, 4, 5)            0.0     0.0     0.0     2.0
(2, 3, 5, 6)            0.0     3.0     0.0     0.0
(2, 4, 5, 6)            1.5     0.0     0.0     0.0
(3, 4, 5, 6)            0.0     0.0     0.0     0.0
```

Και από αυτές οι εκφυλισμένες είναι:

```
Number of degenerate vertices: 5
hyperplanes             x1      x2      x3      x4
(0, 1, 3, 4)            0.0     -0.0    -0.0    2.0
(0, 1, 3, 5)            0.0     -0.0    0.0     2.0
(0, 1, 4, 5)            0.0     -0.0    -0.0    2.0
(0, 3, 4, 5)            0.0     -0.0    -0.0    2.0
(1, 3, 4, 5)            0.0     0.0     0.0     2.0
```

Κώδικας:

```python
## Finding the feasible solutions and the hyperplanes they belong to
feasible_vertices = []
for i in range(len(vertices)):
    if  c1(*vertices[i][0]) \
    and c2(*vertices[i][0]) \
    and c3(*vertices[i][0]) \
    and c4(*vertices[i][0]) \
    and c5(*vertices[i][0]) \
    and c6(*vertices[i][0]) \
    and c7(*vertices[i][0]):
        feasible_vertices.append(vertices[i])
    else:
        continue

## Finding the degenerate vertices and the hyperplanes they belong to
vert_only = np.array([element[0] for element in feasible_vertices])
degenerate_vertices = np.array([x for x in vert_only if np.count_nonzero(np.all(vert_only == x, axis=1)) > 1])
```

Ερώτημα (β):

Οι βασικές λύσεις (εφικτές και μη εφικτές) είναι 30 και οι εξής:

```
Number of general solutions: 30
basic variables             general solution
(0, 1, 2)                   [ 3.5 -4.   1.5]
(0, 1, 3)                   [1. 1. 1.]
(0, 1, 4)                   [ 2. -1.  3.]
(0, 1, 5)                   [-1.  5.  3.]
(0, 1, 6)                   [ 2.  2. -3.]
(0, 2, 3)                   [1.5 0.3 0.8]
(0, 2, 4)                   [ 1.5 -0.5  4. ]
(0, 2, 5)                   [1.5  0.83 1.33]
(0, 2, 6)                   [ 2.5  0.5 -2. ]
(0, 3, 4)                   [1.5 0.5 1.5]
(0, 3, 5)                   [ 1.5   1.25 -0.75]
(0, 3, 6)                   [0. 2. 3.]
(0, 4, 5)                   [1.5 2.5 0.5]
(0, 4, 6)                   [ 2.  2. -1.]
(0, 5, 6)                   [ 4. -2. -5.]
(1, 2, 3)                   [ 3.  -0.6  1.4]
(1, 2, 4)                   [ 3. -2.  7.]
(1, 2, 5)                   [3.   0.33 2.33]
(1, 2, 6)                   [10. -2. -7.]
(1, 3, 4)                   [ 3.  2. -3.]
(1, 3, 5)                   [3.  0.5 1.5]
(1, 3, 6)                   [0. 2. 3.]
(1, 4, 5)                   [3. 1. 2.]
(1, 4, 6)                   [ 4.  2. -1.]
(1, 5, 6)                   [-0.  2.  3.]
(2, 3, 4)                   [-2. 10.  3.]
(2, 3, 5)                   [1.33 3.33 3.  ]
(2, 3, 6)                   [2. 0. 3.]
(2, 4, 5)                   [2. 0. 3.]
(2, 4, 6)                   [4. 2. 3.]
```

Υπολογίζονται, βρίσκοντας όλους τους πιθανούς πίνακες B που αντιστρέφονται και έπειτα:

$$x_B = B^{-1} \cdot b$$

Κώδικας:

```
        continue

## Finding all the possible general solutions and their basic variables
general_solutions = []
for i in range(len(Bs_inv)):
    try:
        x_b = Bs_inv[i][0] @ b
        general_solutions.append( (x_b, basic_variables[i]) )

    except:
        continue
```

Από αυτές οι εφικτές και εκφυλισμένες είναι 14 και 4 αντίστοιχα και οι εξής:

```
Number of feasible solutions: 14
basic variables             feasible solution
(0, 1, 3)                   [1. 1. 1.]
(0, 2, 3)                   [1.5 0.3 0.8]
(0, 2, 5)                   [1.5  0.83 1.33]
(0, 3, 4)                   [1.5 0.5 1.5]
(0, 3, 6)                   [0. 2. 3.]
(0, 4, 5)                   [1.5 2.5 0.5]
(1, 2, 5)                   [3.   0.33 2.33]
(1, 3, 5)                   [3.  0.5 1.5]
(1, 3, 6)                   [0. 2. 3.]
(1, 4, 5)                   [3. 1. 2.]
(2, 3, 5)                   [1.33 3.33 3.  ]
(2, 3, 6)                   [2. 0. 3.]
(2, 4, 5)                   [2. 0. 3.]
(2, 4, 6)                   [4. 2. 3.]
----------------------------------------
Number of degenerate solutions: 4
basic variables             degenerate solution
(0, 3, 6)                   [0. 2. 3.]
(1, 3, 6)                   [0. 2. 3.]
(2, 3, 6)                   [2. 0. 3.]
(2, 4, 5)                   [2. 0. 3.]
```

Κώδικας:

```
## Finding which of the solutions are feasible and degenerate
feasible_solutions = []
degenerate_solutions = []
for i in range(len(general_solutions)):
    if contains_negative(general_solutions[i][0]) == False:
        feasible_solutions.append( general_solutions[i] )
    if contains_zero(general_solutions[i][0]) == True:
        degenerate_solutions.append( general_solutions[i] )
```

Ερώτημα (γ):

Η οι τιμές της αντικειμενικής συνάρτησης σε κάθε εφικτή λύση και η βέλτιστη (μέγιστη) λύση είναι οι εξής:

```
basic variables             value of the objective function
(0, 1, 3)                   2.00
(0, 2, 3)                   -1.80
(0, 2, 5)                   -6.33
(0, 3, 4)                   -1.50
(0, 3, 6)                   6.00
(0, 4, 5)                   -3.00
(1, 2, 5)                   1.67
(1, 3, 5)                   4.50
(1, 3, 6)                   6.00
(1, 4, 5)                   3.00
(2, 3, 5)                   4.67
(2, 3, 6)                   -8.00
(2, 4, 5)                   -8.00
(2, 4, 6)                   -16.00
----------------------------------------
Optimal solution: 6.00
```

Υπολογίζονται ως εξής:

$$z = c_B \cdot x_B$$

Εδώ x_b είναι μόνο οι βασικές εφικτές λύσεις.

Κώδικας:

```
## Finding the optimal solution
# Calculating the value of the objective function for every feasible solution
z = []
for i in range(len(feasible_solutions)):
    # Selecting the coefficients of the basic variables
    c_b  = [ c_s[ feasible_solutions[i][1][0] ], c_s[ feasible_solutions[i][1][1] ], c_s[ feasible_solutions[i][1][2] ] ]
    z.append( ( c_b @ feasible_solutions[i][0], feasible_solutions[i][1]) )



# Finding the optimal solution
values_only = [i[0] for i in z]
maxima = -np.inf
for value in values_only:
    if value > maxima:
        maxima = value
```

Άσκηση 6:

Ερώτημα (α):

Οι πίνακες του αλγόριθμου Simplex για το πρόβλημα της εκφώνησης είναι οι εξής:

```
Iteration 1:
starting basic variables: x5 x6

                |      x1           x2           x3           x4           x5           x6     |
Basis   c_b     |      5.0          4.0          -1.0         3.0          0.0          0.0    |      b
-------------------------------------------------------------------------------------------------------
x5      0.0     |      3.0          2.0          -3.0         1.0          1.0          0.0    |      24.0
x6      0.0     |      3.0          3.0          1.0          3.0          0.0          1.0    |      36.0
-------------------------------------------------------------------------------------------------------
z               |      0.0          0.0          0.0          0.0          0.0          0.0    |      0.0
c-z             |      5.0          4.0          -1.0         3.0          0.0          0.0    |


Iteration 2:


                |      x1           x2           x3           x4           x5           x6     |
Basis   c_b     |      5.0          4.0          -1.0         3.0          0.0          0.0    |      b
-------------------------------------------------------------------------------------------------------
x1      5.0     |      1.0          0.67         -1.0         0.33         0.33         0.0    |      8.0
x6      0.0     |      0.0          1.0          4.0          2.0          -1.0         1.0    |      12.0
-------------------------------------------------------------------------------------------------------
z               |      5.0          3.33         -5.0         1.67         1.67         0.0    |      40.0
c-z             |      0.0          0.67         4.0          1.33         -1.67        0.0    |


Iteration 3:


                |      x1           x2           x3           x4           x5           x6     |
Basis   c_b     |      5.0          4.0          -1.0         3.0          0.0          0.0    |      b
-------------------------------------------------------------------------------------------------------
x1      5.0     |      1.0          0.92         0.0          0.83         0.08         0.25   |      11.0
x3      -1.0    |      0.0          0.25         1.0          0.5          -0.25        0.25   |      3.0
-------------------------------------------------------------------------------------------------------
z               |      5.0          4.33         -1.0         3.67         0.67         1.0    |      52.0
c-z             |      0.0          -0.33        0.0          -0.67        -0.67        -1.0   |

Optimal solution: 52.0
x1 = 11.0
x2 = 0
x3 = 3.0
x4 = 0
x5 = 0
x6 = 0
```

Ο τρόπος παρουσίασης του πίνακα ακολουθεί την λογική του συγκεκριμένου βίντεο https://www.youtube.com/watch?v=9YKLXFqCy6E&t=476s

Για την μορφή του πίνακα, η πρώτη γραμμή κάτω από τις μεταβλητές απόφασης είναι οι συντελεστές της αντικειμενικής συνάρτησης μαζί με τις μεταβλητές χαλάρωσης. Αυτή η γραμμή δεν αλλάζει ποτέ. Οι βασικές μεταβλητές που χρησιμοποιούνται σε κάθε επανάληψη φαίνονται στα αριστερά του πίνακα μαζί με τους συντελεστές τους. Στο κέντρο του πίνακα βάζουμε τους συντελεστές των περιορισμών όλων των μεταβλητών. Στο κεντρικό κάτω κομμάτι του πίνακα στη γραμμή z έχουμε την βασική λύση. Η γραμμή c-z περιέχει τους αντικειμενικούς συντελεστές. Το δεξί κομμάτι του πίνακα έχει το διάνυσμα των σταθερών όρων των περιορισμών και την τιμή της αντικειμενικής συνάρτησης στην τρέχουσα βασική λύση.

Όσον αφορά τα βήματα του αλγορίθμου, Στην πρώτη επανάληψη(αρχικοποίηση, επανάληψη 0), αρχικοποιούμε το πρόβλημα εισάγοντας μεταβλητές χαλάρωσης και αρχίζουμε με μία πρώτη βασική εφικτή λύση, όπως $x = 0, x_s = b$.

Ο κώδικας:

```
# Adding slack variables
c_B = np.append(c, np.zeros(len(A)))
A_s = np.column_stack((A, np.eye(len(A))))

z = np.zeros(len(c_B))
c_z = np.ones(len(c_B))
iters = 0
basic_variables = [i for i in range(len(c), len(c_B))]
```

```
# Initial Simplex Tableau
if iters == 0:
    print(f"\nInitial Simplex Tableau:")
    print(f"starting basic variables: ", end="")
    for var in range(len(basic_variables)):
        print(f"x{basic_variables[var]+1}", end=" ")
    temp1 = [ c_B[basic_variables[i]] for i in range(len(basic_variables))]
    for j in range(len(z)):
        temp2 = A_s[:, j]
        z_j = temp1 @ temp2
        z[j] = z_j
    c_z = c_B - z
    z_value = temp1 @ b
    present_tableau(temp1, A_s, b, z, c_z, z_value, basic_variables, c_B)
```

Σε κάθε επόμενη επανάληψη βρίσκουμε από την c-z το μέγιστο στοιχείο ή γενικά κάποιο θετικό και στην στήλη που αντιστοιχεί την αποκαλούμε οδηγό στήλη και σε αυτήν ανήκει η καινούρια βασική μεταβλητή μας.

```
pivot_column = np.where(c_z == np.max(c_z))[0][0]
```

Έπειτα από τον κεντρικό πίνακα για να βρούμε την οδηγό γραμμή εφαρμόζουμε το κριτήριο ελαχίστου λόγου που βρίσκει την εξερχόμενη βασική μεταβλητή. Η οδηγός γραμμή είναι εκείνη στην οποία ανήκει ο μικρότερος θετικός λόγος. Αν οι λόγοι είναι ίσοι τότε επιλέγουμε αυθαίρετα γραμμή. Επίσης στην ειδική περίπτωση που ο λόγος βγει 0 και προκύψει αρνητικό οδηγό στοιχείο τότε δεν το λαμβάνουμε υπόψη.

```
# Κριτήριο ελαχίστου λόγου
ratios = []
for i in range(len(b)):
    try:
        if A_s[i][pivot_column] < 0:
            ratio = np.inf
            ratios.append(ratio)
            continue
        ratio = b[i] / A_s[i][pivot_column]
        if ratio < 0:
            ratio = np.inf
            ratios.append(ratio)
        else:
            ratios.append(ratio)
    except:
        ratio = np.inf
        ratios.append(ratio)
```

Στη συνέχεια κάνουμε απαλοιφή Gauss(οδήγηση) και υπολογίζουμε τις γραμμές z, c-z πάλι και την τιμή της αντικειμενικής συνάρτησης στην συγκεκριμένη εφικτή λύση.

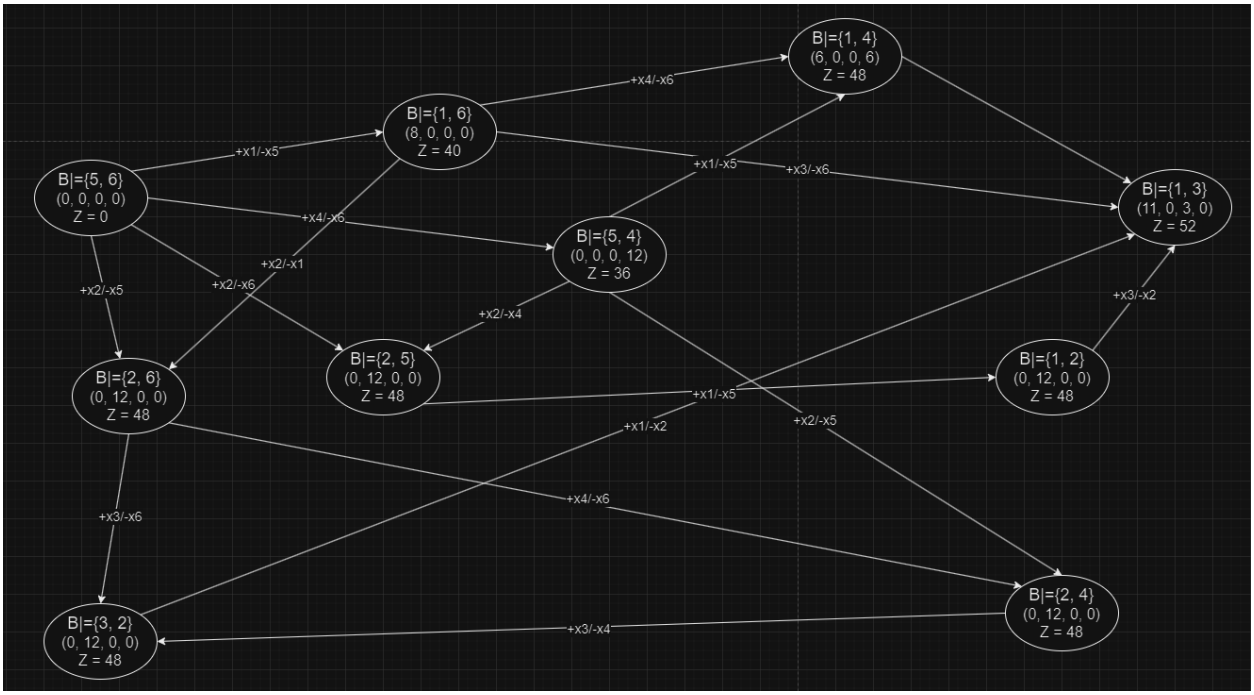
```
for i in range(len(A_s)):
    if i != pivot_row:
        el = A_s[i][pivot_column]
        A_s[i] = A_s[i] - ( el  * A_s[pivot_row] )

        b[i] = b[i] - (  el  * b[pivot_row] )

temp1 = [ c_B[basic_variables[i]] for i in range(len(basic_variables))]
for j in range(len(z)):
    temp2 = A_s[:, j]
    z_j = temp1 @ temp2
    z[j] = z_j

c_z = c_B - z
z_value = temp1 @ b
present_tableau(temp1, A_s, b, z, c_z, z_value, basic_variables, c_B)
```

Όταν η γραμμή c-z περιέχει μόνο μη θετικά στοιχεία τότε βρισκόμαστε στην βέλτιστη λύση και σταματάμε τον αλγόριθμο.

```
while contains_positive(c_z):
```

Ερώτημα (β):

Αλλάζοντας σε κάθε δυνατό βήμα την εισερχόμενη μεταβλητή παίρνουμε τον εξής γράφο:



Εικόνα 3: Simplex Adjacency Graph

Κώδικας για δοκιμές διάφορων διαδρομών:

```
if iters == 1:
    # pivot_column = np.where(c_z == np.max(c_z))[0][0]
    pivot_column = pos_c_z[2]
    print(f"pivot_column: {pivot_column}")
elif iters == 2:
    # pivot_column = np.where(c_z == np.max(c_z))[0][0]
    pivot_column = pos_c_z[1]
    print(f"pivot_column: {pivot_column}")
elif iters == 4:
    # pivot_column = np.where(c_z == np.max(c_z))[0][0]
    pivot_column = pos_c_z[1]
    print(f"pivot_column: {pivot_column}")
else:
    pivot_column = np.where(c_z == np.max(c_z))[0][0]
    print(f"pivot_column: {pivot_column}")
```