ΓΡΑΜΜΙΚΗ ΚΑΙ ΣΥΝΔΥΑΣΤΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

ΕΡΓΑΣΙΑ 1

Ονοματεπώνυμο: ΓΙΑΝΝΑΚΑΚΗΣ ΑΝΑΣΤΑΣΙΟΣ,

Τμήμα: ΗΜΤΥ

Κατεύθυνση: ΥΠΟΛΟΓΙΣΤΩΝ

Έτος: 4ο

Α.Μ: 1072905

Άσκηση 1.

Α) Έστω πρόβλημα γραμμικού προγραμματισμού που έχει ως περιορισμούς τις παρακάτω ανισώσεις:

(Π1) $2x1 + x2 \geq 4$

(Π2) $x1 + 2x2 \geq 5$

(Π3) $x1 - 2x2 \leq 1$

$x1, x2 \geq 0$

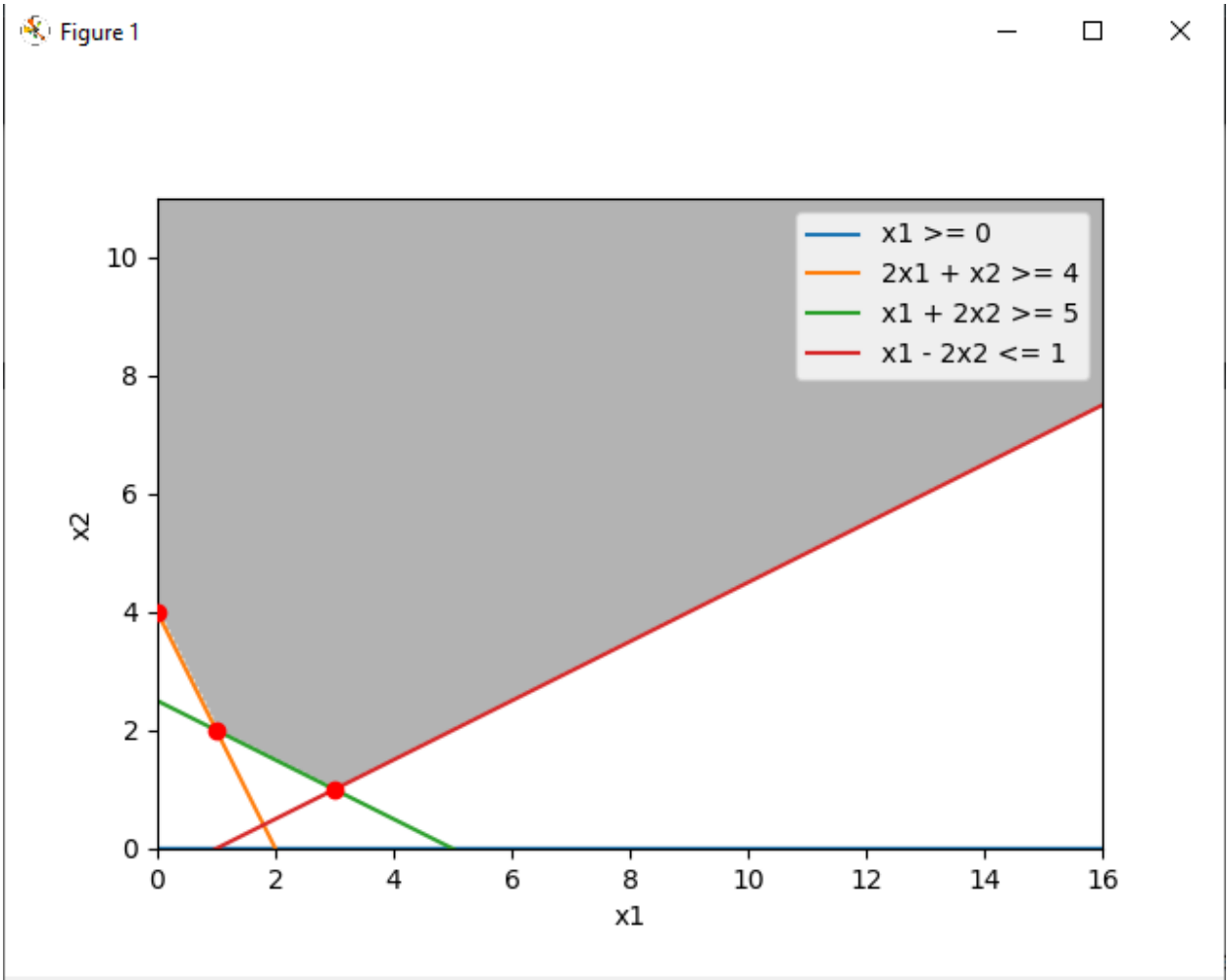
Εδώ ορίζουμε τις εξισώσεις που είναι και οι περιορισμοί μας και σχεδιάζουμε τις ευθείες που ορίζουν. Παράλληλα βρίσκουμε όλες τις πιθανές κορυφές της εφικτής περιοχής λύνοντας με ένα for loop κάθε φορά ζευγάρια εξισώσεων και στο τέλος τις ελέγχουμε περνώντας τις ξανά από τις ανισώσεις για να δούμε αν τις ικανοποιούν.

Παρακάτω ο κώδικας:

```
# displays a 2D image and greys out the feasible area
plt.imshow( ((y>=0) & (2*y>=x1-1) & (2*y>=5-x1) & (y>=4-2*x1)).astype(int) ,
            extent=(x1.min(),x1.max(),y.min(),y.max()),origin="lower", cmap="Greys", alpha = 0.3)


# plot the lines defining the constraints
x1 = np.linspace(0, 16, 2000)

# y >= 0
y1 = x1*0

# 2x1 + y ≥ 4
y2 = 4-2*x1

# x1 + 2y ≥ 5
y3 = (5-x1)/2

# x1 - 2y ≤ 1
y4 = (x1-1)/2


# Make plot
plt.plot(x1, 0*np.ones_like(y1), label="x1 >= 0")
plt.plot(x1, y2, label="2x1 + x2 >= 4")
plt.plot(x1, y3, label="x1 + 2x2 >= 5")
plt.plot(x1, y4, label="x1 - 2x2 <= 1")
```

```python
# Find all possible vertices
A = np.array([[-2, -1], [-1, -2], [1, -2]])
b = np.array([-4, -5, 1])

vertices = []
for i in range(len(A)):
    for j in range(i+1, len(A)):
        try:
            sub_A = A[[i, j], :]
            sub_b = b[[i, j]]
            x = np.linalg.solve(sub_A, sub_b)
            # evaluate the constraints for the current solution
            y = x[1]
            z1 = x[0]
            con1 = y >= 0
            con2 = 2*y >= z1-1
            con3 = 2*y >= 5-z1
            con4 = y >= 4-2*z1

            # check if all constraints are satisfied
            if con1 and con2 and con3 and con4:
                print("Solution for pair", i+1, "and", j+1, ":", x, "satisfies the constraints.")
                vertices.append(x)
        except np.linalg.LinAlgError:
            continue
```

Και το αποτέλεσμά του: Η γκρι περιοχή αποτελεί την feasible area και τα κόκκινα σημεία αποτελούν τις κορυφές της εφικτής περιοχής.



Β) Αντικειμενική συνάρτηση: Χρησιμοποιώντας τον κώδικα παρακάτω και απλώς αλλάζοντας την αντικειμενική συνάρτηση έχουμε τα εξής:

```
64  vertices = np.array(vertices)
65  x_values = vertices[:, 0]
66  y_values = vertices[:, 1]
67  z_values = 2*x_values - 5*y_values # Objective Function
68  max_z_index = np.argmax(z_values)
69  max_z_vertex = vertices[max_z_index]
70  max_z_value = np.max(z_values)
71  print("The vertex that maximizes the function is", max_z_vertex,"and the value is",max_z_value)
```

(i) max Z = 2x1 - 5x2

Για να μεγιστοποιήσουμε το Z = 2x1 - 5x2, πρέπει να βρούμε το σημείο στο όριο της εφικτής περιοχής που μεγιστοποιούν την τιμή του Z. Στη συνέχεια σχεδιάζουμε την ευθεία με την εξίσωση 2x1 - 5x2 = k, όπου k είναι μια σταθερά, και βρίσκουμε την τομή της με τις οριακές γραμμές της εφικτής περιοχής. Η βέλτιστη λύση (ή οι βέλτιστες λύσεις) θα είναι τα σημεία τομής που μεγιστοποιούν την τιμή του Z.

Για να επιλύσουμε τη μέγιστη τιμή του Z, πρέπει να βρούμε το σημείο στο όριο που τέμνει(ουν) την ευθεία 2x1 - 5x2 = k και έχει(ουν) τη μεγαλύτερη τιμή του Z. Δοκιμάζοντας διαφορετικές τιμές του k, βρίσκουμε ότι η μέγιστη τιμή του Z είναι 2 στο σημείο (2, 0,5). Επομένως, η βέλτιστη λύση είναι **x1 = 3, x2 = 1 και Z = 1**.

```
Solution for pair 1 and 2 : [1. 2.] satisfies the constraints.
Solution for pair 2 and 3 : [3. 1.] satisfies the constraints.
The vertex that maximizes the function is [3. 1.] and the value is 1.0
```

(ii) max Z = 2x1 - 4x2

Για να μεγιστοποιήσουμε το Z = 2x1 - 4x2, ακολουθούμε την ίδια διαδικασία όπως στο (i), αλλά με διαφορετική γραμμή αντικειμενικής συνάρτησης. Η κλίση της ευθείας είναι -1/2. Δοκιμάζοντας διαφορετικές τιμές του k, διαπιστώνουμε ότι η μέγιστη τιμή του Z είναι 2 στο σημείο (3, 1). Επομένως, η βέλτιστη λύση είναι **x1 = 3, x2 = 1 και Z = 2**.

```
Solution for pair 1 and 2 : [1. 2.] satisfies the constraints.
Solution for pair 2 and 3 : [3. 1.] satisfies the constraints.
The vertex that maximizes the function is [3. 1.] and the value is 2.0
```

(iii) max Z = 2x1 - 3x2

Για να μεγιστοποιήσουμε το Z = 2x1 - 3x2, ακολουθούμε την ίδια διαδικασία όπως στα (i) και (ii), αλλά με διαφορετική γραμμή αντικειμενικής συνάρτησης. Δοκιμάζοντας διαφορετικές τιμές του k, διαπιστώνουμε ότι η ευθεία όσο μεγαλώνουμε την τιμή του k, παραμένει στην εφικτή περιοχή συνεπώς δεν μπορούμε να ορίσουμε την τιμή της βέλτιστης λύσης. Το πρόβλημα **δεν έχει βέλτιστη λύση**.

Άσκηση 2.

Το πρόβλημα απαιτεί την εύρεση των βέλτιστων ποσοτήτων δημητριακών G1 και G2 για την παρασκευή του προϊόντος σνακ με χαμηλά λιπαρά. Ο στόχος είναι να ελαχιστοποιηθεί το κόστος παραγωγής μιας μονάδας του προϊόντος, ενώ παράλληλα να πληρούνται οι διατροφικές προδιαγραφές. Έστω x1 και x2 οι ποσότητες των δημητριακών G1 και G2, αντίστοιχα, που χρησιμοποιούνται για την παραγωγή μιας μονάδας του προϊόντος. Τότε, το μοντέλο γραμμικού προγραμματισμού για το πρόβλημα είναι:

A) Αντικειμενική Συνάρτηση $\min(Z) = 6x_1 + 7.5x_2$ (κόστος παραγωγής μιας μονάδας)

Υπό την προϋπόθεση ότι:

```
# define the constraints
con1 = y >= 0
con2 = 4.5*y >= 5.1-(6*x1)
con3 = 9*y <= 8.4-(6*x1)
con4 = 9*y <= 10.8-(12*x1)
```

6x1 + 4,5x2 >= 5,1 (ελάχιστη απαίτηση ινών)

6x1 + 9x2 <= 8,4 (μέγιστη απαίτηση λίπους)

12x1 + 9x2 <= 10,8 (μέγιστη απαίτηση πρωτεϊνών)

x1, x2 >= 0 (μη αρνητικός περιορισμός)

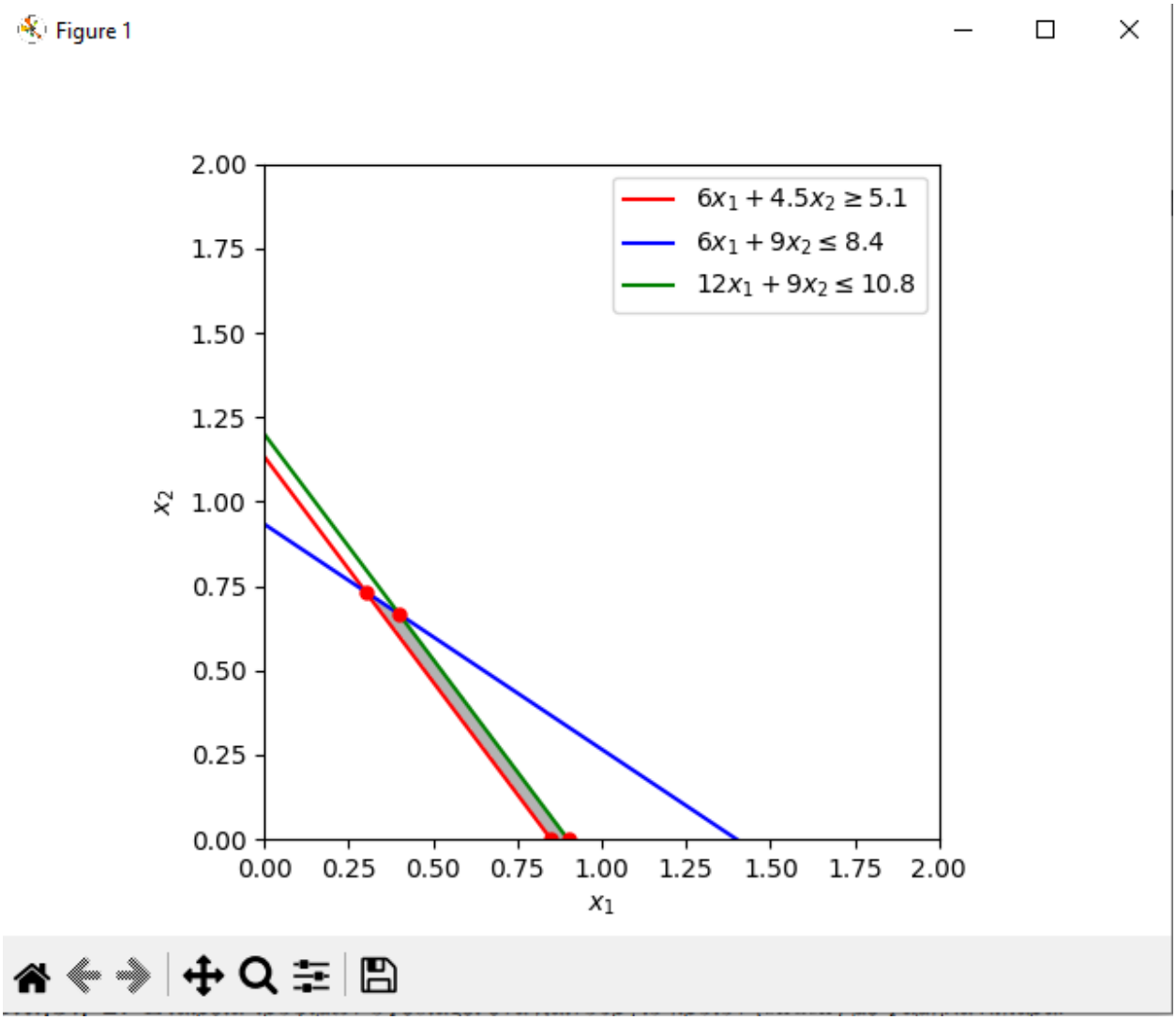
Ο πρώτος περιορισμός διασφαλίζει ότι το προϊόν πληροί την ελάχιστη απαίτηση σε φυτικές ίνες, ενώ ο δεύτερος και ο τρίτος περιορισμός διασφαλίζουν ότι το προϊόν δεν υπερβαίνει τις μέγιστες απαιτήσεις σε λίπος και πρωτεΐνη, αντίστοιχα. Ο μη αρνητικός περιορισμός εξασφαλίζει ότι οι ποσότητες δημητριακών που χρησιμοποιούνται είναι μη αρνητικές.

B) Για τη γραφική επίλυση του προβλήματος, σχεδιάζουμε τους περιορισμούς σε ένα δισδιάστατο γράφημα με x1 στον οριζόντιο άξονα και x2 στον κατακόρυφο άξονα. Η εφικτή περιοχή είναι η περιοχή που ικανοποιεί όλους τους περιορισμούς. Στον παρακάτω κώδικα φαίνεται ο ορισμός του προβλήματος και των περιορισμών και βρίσκονται οι κορυφές της εφικτής περιοχής.

```
# Finding the vertices of the feasible area
A = np.array([[-6, -4.5], [6, 9], [12, 9], [-1, 0], [0, -1]])
b = np.array([-5.1, 8.4, 10.8, 0, 0])


vertices = []
for i in range(len(A)):
    for j in range(i+1, len(A)):
      try:
        sub_A = A[[i, j], :]
        sub_b = b[[i, j]]
        x = np.linalg.solve(sub_A, sub_b)
        # evaluate the constraints for the current solution
        y = x[1]
        z1 = x[0]
        con1 = y >= 0
        con2 = 4.5*y >= 5.1-(6*z1)
        con3 = 9*y <= 8.4-(6*z1)
        con4 = 9*y <= 10.8-(12*z1)

        # check if all constraints are satisfied
        if con1 and con2 and con3 and con4:
            print("Solution for pair", i+1, "and", j+1, ":", x, "satisfies the constraints.")
            vertices.append(x)
      except np.linalg.LinAlgError:
        continue
```

Η βέλτιστη λύση είναι μια κορυφή της εφικτής περιοχής που ελαχιστοποιεί την αντικειμενική συνάρτηση.

Η εφικτή περιοχή είναι το σκιασμένο πολύγωνο που οριοθετείτε από τους τρεις περιορισμούς. Και οι κόκκινες κουκκίδες αποτελούν τις κορυφές της εφικτής περιοχής.



Έπειτα κοιτάμε αν κάποια από τις κορυφές και ποια αποτελεί τη βέλτιστη λύση δηλαδή αυτή που ελαχιστοποιεί τη συνάρτηση Z. **Παρακάτω φαίνεται ο κώδικας:**

```
63  vertices = np.array(vertices)
64  x_values = vertices[:, 0]
65  y_values = vertices[:, 1]
66  z_values = 6*x_values + 7.5*y_values
67  min_z_index = np.argmin(z_values)
68  min_z_vertex = vertices[min_z_index]
69  min_z_value = np.min(z_values)
70  print("The vertex that minimizes the function is", min_z_vertex,"and the value is",min_z_value)
```

και το αποτέλεσμα του:

```
Solution for pair 1 and 2 : [0.3        0.73333333] satisfies the constraints.
Solution for pair 1 and 5 : [ 0.85 -0.  ] satisfies the constraints.
Solution for pair 2 and 3 : [0.4        0.66666667] satisfies the constraints.
Solution for pair 3 and 5 : [ 0.9 -0. ] satisfies the constraints.
The vertex that minimizes the function is [ 0.85 -0.  ] and the value is 5.1
```

Επομένως, η βέλτιστη λύση είναι να χρησιμοποιηθούν **0.85 μονάδες δημητριακών G1** και **0 μονάδες δημητριακών G2** για την παραγωγή μιας μονάδας του προϊόντος σνακ με χαμηλά λιπαρά. Το κόστος παραγωγής μιας μονάδας με τη χρήση αυτού του μείγματος είναι **6*(0.85) + 7.5*(0) = 5.1 χρηματικές μονάδες**. Αυτή η λύση ικανοποιεί όλες τις διατροφικές απαιτήσεις και είναι ο πιο οικονομικός τρόπος παραγωγής του προϊόντος.

Γ) Συμπερασματικά, το μοντέλο γραμμικού προγραμματισμού παρέχει μια βέλτιστη λύση για την παραγωγή του προϊόντος σνακ χαμηλών λιπαρών χρησιμοποιώντας τα δημητριακά G1 και G2. Η γραφική λύση δείχνει ότι ο οικονομικότερος τρόπος παραγωγής του προϊόντος είναι η χρήση 0.85 μονάδων δημητριακών G1 και 0 μονάδων δημητριακών G2. Η λύση αυτή ικανοποιεί όλες τις διατροφικές απαιτήσεις και ελαχιστοποιεί το κόστος παραγωγής.

Άσκηση 3.

Για να μοντελοποιήσουμε το συγκεκριμένο πρόβλημα με τη χρήση γραμμικού προγραμματισμού, πρέπει να ορίσουμε τις μεταβλητές απόφασης, την αντικειμενική συνάρτηση και τους περιορισμούς.

Μεταβλητές απόφασης:

Έστω xij ο αριθμός των εργαζομένων που έχουν οριστεί να εργαστούν στη βάρδια i κατά την ώρα j, όπου i = 1,2,3,4,5 και j = 1,2,...,10.

Αντικειμενική συνάρτηση:

Ο στόχος είναι η ελαχιστοποίηση του συνολικού ημερήσιου κόστους της εταιρείας σε μισθούς, το οποίο μπορεί να παρασταθεί ως εξής:

Min(Z) = 170(x11 + x12 + x13 + x14) // shift 1

+ 160(x22 + x23 + x24 + x25)// shift 2

+ 175(x34 + x35 + x36 + x37) // shift 3

+ 180(x46 + x47 + x48 + x49) // shift 4

+ 195(x49 + x4_10) // shift 5

(Μεταβλητές όπως την τιμή ας πούμε x14..κλπ παίρνουν τιμή 0 αφού δεν γίνεται κάποιος που δουλεύει την 7η περίοδο ώρας δηλαδή 18:00-20:00 ,να ανήκει στη βάρδια 1 που είναι από τις 06:00-14:00. Προστίθονται λόγω ευκολίας στον προγραμματισμό του προβλήματος.)

Περιορισμοί:

Ο αριθμός των εργαζομένων που αντιστοιχούν σε κάθε ώρα πρέπει να είναι μεγαλύτερος ή ίσος με τον ελάχιστο απαιτούμενο αριθμό εργαζομένων για τη συγκεκριμένη ώρα. Οι περιορισμοί βγαίνουν ανάλογα με το σε ποια ώρα δύνανται να δουλέψει ο κάθε υπάλληλος**. Πχ κανένας από τις άλλες βάρδιες πέρα από την 1 δεν μπορεί να δουλέψει στη χρονική περίοδο 6:00-8:00 καθώς δεν του το επιτρέπει η βάρδια του**:

x11 >= 48 // 6:00 - 8:00

x12 + x22 >= 79 // 8:00 - 10:00

x13 + x23 >= 65 // 10:00 - 12:00 (Η τρίτη βάρδια ξεκινάει στις 12 άρα από εκει και πέρα δεν συμπεριλαμβάνονται οι άλλες βάρδιες)

x14 + x24 + x34 >= 87 // 12:00 - 14:00

x25 + x35 >= 64 // 14:00 - 16:00

x36 + x46 >= 73 // 16:00 - 18:00

x37 + x47 >= 82 // 18:00 - 20:00

x48 + x58 >= 43 // 20:00 - 22:00

x49 + x59 >= 52 // 22:00 - 24:00

x5_10 >= 15 // 24:00 - 6:00

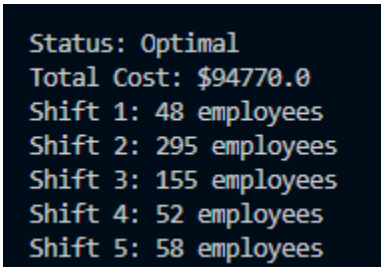
Όλες οι μεταβλητές απόφασης πρέπει να είναι μη αρνητικές:

xij >= 0, για όλα τα i και j

Ο κώδικας παρακάτω:

```python
# Import PuLP modeler functions
from pulp import *

# Create the model
model = LpProblem(name="Employee Scheduling Problem", sense=LpMinimize)

# Define decision variables
shifts = range(1, 6)
hours = range(1, 11)
x = LpVariable.dicts("x", (shifts, hours), lowBound=0, cat='Integer')

# Define objective function
model += 170 * (x[1][1] + x[1][2] + x[1][3] + x[1][4]) \
         + 160 * (x[2][2] + x[2][3] + x[2][4] + x[2][5]) \
         + 175 * (x[3][4] + x[3][5] + x[3][6] + x[3][7]) \
         + 180 * (x[4][6] + x[4][7] + x[4][8] + x[4][9]) \
         + 195 * (x[5][9] + x[5][10]), "Total Cost"

# Define constraints
for h in hours:
    model += lpSum([x[s][h] for s in shifts]) >= 1, f"At least 1 employee must work hour {h}"

# Define constraints
model += x[1][1] >= 48, "Time-Period 1"
model += x[1][2] + x[2][2] >= 79, "Time-Period 2"
model += x[1][3] + x[2][3] >= 65, "Time-Period 3"
model += x[1][4] + x[2][4] + x[3][4] >= 87, "Time-Period 4"
model += x[2][5] + x[3][5] >= 64, "Time-Period 5"
model += x[3][6] + x[4][6] >= 73, "Time-Period 6"
model += x[3][7] + x[4][7] >= 82, "Time-Period 7"
model += x[4][8] + x[5][8] >= 43, "Time-Period 8"
model += x[4][9] + x[5][9] >= 52, "Time-Period 9"
model += x[5][10] >= 15, "Time-Period 10"

# Solve the model
status = model.solve()

# Print the status of the solution
print(f"Status: {LpStatus[status]}")

if status == 1:
    # Print the total cost
    print(f"Total Cost: ${value(model.objective)}")

    # Print the optimal schedule
    for s in shifts:
        print(f"Shift {s}: {int(sum([x[s][h].varValue for h in hours]))} employees")
else:
    print("No feasible solution found.")
```

Και το αποτέλεσμα του:



Στο αποτέλεσμα φαίνεται το τελικό ημερήσιο κόστος της εταιρείας και οι υπάλληλοι σε κάθε βάρδια.

Άσκηση 4.

(Π1) Έστω x και y δύο σημεία του X και έστω $0 \leq \lambda \leq 1$ πραγματικός αριθμός. Θέλουμε να δείξουμε ότι το λx + (1-λ)y ανήκει στο X. Επειδή τα X1 και X2 είναι κυρτά σύνολα, έχουμε τα x και y στα X1 και X2 αντίστοιχα. Έτσι, λόγω της κυρτότητας των X1 και X2, έχουμε το λx + (1-λ)y στα X1 και X2. Επομένως, το λx + (1-λ)y βρίσκεται στην τομή X των X1 και X2. Εφόσον το λx + (1-λ)y είναι ένας κυρτός συνδυασμός των x και y, το X είναι ένα κυρτό σύνολο.

(Π2) Θέλουμε να δείξουμε ότι για δύο οποιαδήποτε σημεία (x1,y1) και (x2,y2) στο Ω, οποιοσδήποτε κυρτός συνδυασμός τους ανήκει επίσης στο Ω. Έστω (x1,y1) και (x2,y2) δύο σημεία στο Ω, δηλαδή $x1^2+y1^2 \leq 1$ και $x2^2+y2^2 \leq 1$. Έστω $0 \leq \lambda \leq 1$ πραγματικός αριθμός και θεωρούμε τον κυρτό συνδυασμό (λx1 + (1-λ)x2, λy1 + (1-λ)y2).

Σύμφωνα με τις ιδιότητες των πραγματικών αριθμών, έχουμε:

$(\lambda x1 + (1-\lambda)x2)^2 + (\lambda y1 + (1-\lambda)y2)^2$

$= \lambda(x1^2 + y1^2) + (1-\lambda)(x2^2 + y2^2) + 2\lambda(1-\lambda)(x1x2 + y1y2)$

Δεδομένου ότι $x1^2 + y1^2 \leq 1$ και $x2^2 + y2^2 \leq 1$, έχουμε:

$(\lambda x1 + (1-\lambda)x2)^2 + (\lambda y1 + (1-\lambda)y2)^2 \leq \lambda + (1-\lambda) = 1$

Έτσι, κάθε κυρτός συνδυασμός δύο σημείων στο Ω ανήκει επίσης στο Ω, γεγονός που δείχνει ότι το Ω είναι κυρτό σύνολο.

(Π3) Θα αποδείξουμε την πρόταση με επαγωγή στο m. Για m=2, έστω x^1 και x^2 δύο σημεία στο X, και έστω $0 \leq \lambda \leq 1$ ένας πραγματικός αριθμός. Τότε, ο κυρτός συνδυασμός λx^1 + (1-λ)x^2 ανήκει στο X σύμφωνα με τον ορισμό της κυρτότητας.

Ας υποθέσουμε τώρα ότι η δήλωση είναι αληθής για οποιαδήποτε m-1 σημεία στο X, και έστω x^1, x^2, ... , x^m είναι m σημεία στο X. Θέλουμε να δείξουμε ότι οποιοσδήποτε κυρτός συνδυασμός αυτών των σημείων ανήκει επίσης στο X.

Έστω $0 \leq \lambda \leq 1$ πραγματικός αριθμός και έστω x = λx^1 + (1-λ)x^2 + y, όπου y = (1-λ)λ^-1(x^1 - x^2) είναι γραμμικός συνδυασμός του (x^1 - x^2). Τότε, μπορούμε να ξαναγράψουμε το x ως εξής:

x = (1-λ)λ^-1x^1 + λ^-1(λx^2 + (1-λ)x^1 + y)

Δεδομένου ότι ο πρώτος όρος (1-λ)λ^-1x^1 και ο δεύτερος όρος λ^-1(λx^2 + (1-λ)x^1 + y) είναι και οι δύο κυρτοί συνδυασμοί m-1 σημείων στο X (δηλαδή των x^1 και y στον πρώτο όρο και των x^2, x^1 και y στον δεύτερο όρο), ανήκουν στο X σύμφωνα με την επαγωγική υπόθεση. Εφόσον το X είναι ένα κυρτό σύνολο, ο κυρτός συνδυασμός τους x ανήκει επίσης στο X. **Επομένως, κάθε κυρτός συνδυασμός των x^1, x^2, . . . , x^m ανήκει στο X, γεγονός που ολοκληρώνει την επαγωγική απόδειξη.**

Άσκηση 5.

(α) Για να βρούμε τις κορυφές του πολυτόπου των εφικτών λύσεων, πρέπει να βρούμε τις τομές των τριών περιορισμών ανισότητας. Ορίζουμε τους πίνακες και τους περιορισμούς μας και έπειτα βρίσκουμε όλες τις πιθανές κορυφές που δημιουργούνται από τις τομές των υπερεπιπέδων. Έπειτα ελέγχοντας τις περνώντας τες από τους περιορισμούς βρίσκουμε τις εφικτές λύσεις.

min (Z) = $12x1 - 10x2 - 30x3$

$-3x1 + 2x2 + 8x3 \leq 17$

$-x1 + x2 + 3x3 \leq 9$

$-2x1 + x2 + 8x3 \leq 16$

$x1, x2, x3 \geq 0$

Παρακάτω ο κώδικας:

```
# Finding the vertices of the feasible area
A = np.array([[-3, 2, 8], [-1, 1, 3], [-2, 1, 8], [-1, 0, 0], [0, -1, 0], [0, 0, -1]])
b = np.array([17, 9, 16, 0, 0, 0])

vertices = []
for i in range(len(A)):
    for j in range(i+1, len(A)):
      for k in range(j+1, len(A)):
        try:
          sub_A = A[[i, j, k], :]
          sub_b = b[[i, j, k]]
          x = np.linalg.solve(sub_A, sub_b)
          # evaluate the constraints for the current solution
          x1 = x[0]
          x2 = x[1]
          x3 = x[2]
          con1 = -3*x1 + 2*x2 + 8*x3 <= 17
          con2 = -x1 + x2 + 3*x3 <= 9
          con3 = -2*x1 + x2 + 8*x3 <= 16
          con4 = x1 >= 0
          con5 = x2 >= 0
          con6 = x3 >= 0

          # check if all constraints are satisfied
          if con1 and con2 and con3 and con4 and con5 and con6:
            print("Solution for pair", i+1, "and", j+1, ":", x, "satisfies the constraints.")
            vertices.append(x)
          else:
            print("Solution for pair", i+1, "and", j+1, ":", x, "does not satisfy the constraints.")
        except np.linalg.LinAlgError:
          continue
```

Και τα αποτελέσματά του:

```
Solution for pair 1 and 2 and 3 : [6.33333333 7.33333333 2.66666667] satisfies the constraints.
Solution for pair 1 and 2 and 4 : [-0.  10.5 -0.5] does not satisfy the constraints.
Solution for pair 1 and 2 and 5 : [21. -0. 10.] does not satisfy the constraints.
Solution for pair 1 and 2 and 6 : [ 1. 10. -0.] satisfies the constraints.
Solution for pair 1 and 3 and 4 : [-5.92118946e-16  1.00000000e+00  1.87500000e+00] does not satisfy the constraints.
Solution for pair 1 and 3 and 5 : [-1.   -0.    1.75] does not satisfy the constraints.
Solution for pair 1 and 3 and 6 : [-15. -14.  -0.] does not satisfy the constraints.
Solution for pair 1 and 4 and 5 : [-0.    -0.     2.125] does not satisfy the constraints.
Solution for pair 1 and 4 and 6 : [-0.   8.5 -0. ] satisfies the constraints.
Solution for pair 1 and 5 and 6 : [-5.66666667 -0.         -0.        ] does not satisfy the constraints.
Solution for pair 2 and 3 and 4 : [-4.4408921e-16  4.8000000e+00  1.4000000e+00] does not satisfy the constraints.
Solution for pair 2 and 3 and 5 : [-12.  -0.  -1.] does not satisfy the constraints.
Solution for pair 2 and 3 and 6 : [-7.  2. -0.] does not satisfy the constraints.
Solution for pair 2 and 4 and 5 : [-0. -0.  3.] does not satisfy the constraints.
Solution for pair 2 and 4 and 6 : [-0.  9. -0.] does not satisfy the constraints.
Solution for pair 2 and 5 and 6 : [-9. -0. -0.] does not satisfy the constraints.
Solution for pair 3 and 4 and 5 : [-0. -0.  2.] satisfies the constraints.
Solution for pair 3 and 4 and 6 : [-0. 16. -0.] does not satisfy the constraints.
Solution for pair 3 and 5 and 6 : [-8. -0. -0.] does not satisfy the constraints.
Solution for pair 4 and 5 and 6 : [-0. -0. -0.] satisfies the constraints.
The vertex that minimizes the function is [ 1. 10. -0.] and the value is -88.0
```

Μια κορυφή ονομάζεται εκφυλισμένη αν υπάρχει τουλάχιστον ένα μηδέν στο $B^{-1}b$ ονομάζεται εκφυλισμένη.

```
The degenerate solutions are
 [array([ 1., 10., -0.]), array([-0. ,  8.5, -0. ]), array([-0., -0.,  2.]), array([-0., -0., -0.])]
```

(β) Για να προσθέσουμε μεταβλητές χαλάρωσης στο σύστημα ανισοτήτων, εισάγουμε χαλαρές μεταβλητές x_4, x_5, x_6 και ξαναγράφουμε τις ανισότητες ως ισότητες:

$-3x1 + 2x2 + 8x3 + x_4 <= 17$

$-x1 + x2 + 3x3 + x_5 <= 9$

$-2x1 + x2 + 8x3 + x_6 <= 16$

Ορίζουμε το πρόβλημα και βρίσκουμε την βέλτιστη λύση:

```python
# Create the problem variable
prob = LpProblem("LP Problem", LpMinimize)

# Create the decision variables
x1 = LpVariable("x1", lowBound=0)
x2 = LpVariable("x2", lowBound=0)
x3 = LpVariable("x3", lowBound=0)
x4 = LpVariable("x4", lowBound=0)
x5 = LpVariable("x5", lowBound=0)
x6 = LpVariable("x6", lowBound=0)

# Set the objective function
prob += 12*x1 - 10*x2 - 30*x3, "Z"

# Add the constraints
prob += -3*x1 + 2*x2 + 8*x3 + x4 == 17
prob += -1*x1 + 1*x2 + 3*x3 + x5 == 9
prob += -2*x1 + 1*x2 + 8*x3 + x6 == 16

# Solve the problem
prob.solve()

# If an optimal solution is found, print the optimal value and decision variables
if LpStatus[prob.status] == "Optimal":
    print("Optimal Value of Z =", value(prob.objective))
    for v in prob.variables():
        print(v.name, "=", v.varValue)
```

Έπειτα βρίσκουμε όλες τις βασικές λύσεις του μη ομογενούς συστήματος:

```
# Define the matrix A and the vector b
A = np.array([[-3, 2, 8, 1, 0, 0],
              [-1, 1, 3, 0, 1, 0],
              [-2, 1, 8, 0, 0, 1]])
b = np.array([17, 9, 16])

# Find all possible sets of basic variables
basic_var_indices = [set(indices) for indices in itertools.combinations(range(A.shape[1]), 3)]

# Find the basic solutions for each set of basic variables
basic_solutions = []
for indices in basic_var_indices:
    A_basic = A[:, list(indices)]
    x_basic = np.linalg.solve(A_basic, b)
    x = np.zeros(A.shape[1])
    x[list(indices)] = x_basic
    basic_solutions.append(x)

# Print the basic solutions
for x in basic_solutions:
    print(x)
```

Και το αποτέλεσμα έχει ως εξής:

```
Basic solutions:
 [ 0.  0.  0. 17.  9. 16.]
[6.33333333 7.33333333 2.66666667 0.         0.         0.        ]
[-7.  2.  0. -8.  0.  0.]
[-15. -14.   0.   0.   8.   0.]
[ 1. 10.  0.  0.  0.  8.]
[-12.   0.  -1. -11.   0.   0.]
[-1.    0.    1.75  0.    2.75  0.  ]
[ 21.   0.  10.   0.   0. -22.]
[-8.  0.  0. -7.  1.  0.]
[ -9.   0.   0. -10.   0.  -2.]
[-5.66666667  0.          0.          0.          3.33333333  4.66666667]
[ 0.   4.8  1.4 -3.8  0.   0. ]
[0.    1.    1.875 0.    2.375 0.   ]
[ 0.  10.5 -0.5  0.   0.   9.5]
[  0.  16.   0. -15.  -7.   0.]
[ 0.  9.  0. -1.  0.  7.]
[0.  8.5 0.  0.  0.5 7.5]
[0. 0. 2. 1. 3. 0.]
[ 0.  0.  3. -7.  0. -8.]
[ 0.     0.     2.125  0.     2.625 -1.   ]
[ 0.  0.  0. 17.  9. 16.]
```

Όπου το array έχει ως εξής: [x1 x2 x3 x4 x5 x6]

Τέλος εντοπίζουμε τις εκφυλισμένες λύσεις χρησιμοποιώντας τον παρακάτω κώδικα:

```python
# Find the degenerate basic solutions
deg_solutions = []
for sol in basic_solutions:
    if 0 in sol:
        deg_solutions.append(sol)

# Print the degenerate basic solutions
if len(deg_solutions) > 0:
    print("Degenerate basic solutions:\n")
    for solution in deg_solutions:
      print(np.around(solution, 2), end="\n\n")

else:
    print("There are no degenerate basic solutions.")
```

Και το αποτέλεσμα έχει ως εξής:

```
Degenerate basic solutions:

[6.33 7.33 2.67 0.   0.   0.  ]
[-7.  2.  0. -8.  0.  0.]
[-15. -14.   0.   0.   8.   0.]
[ 1. 10.  0.  0.  0.  8.]
[-12.   0.  -1. -11.   0.   0.]
[-1.    0.    1.75  0.    2.75  0.  ]
[ 21.   0.  10.   0.   0. -22.]
[-8.  0.  0. -7.  1.  0.]
[ -9.   0.   0. -10.   0.  -2.]
[-5.67  0.    0.    0.    3.33  4.67]
[ 0.   4.8  1.4 -3.8  0.   0. ]
[0.   1.   1.88 0.   2.38 0.  ]
[ 0.  10.5 -0.5  0.   0.   9.5]
[  0.  16.   0. -15.  -7.   0.]
[ 0.  9.  0. -1.  0.  7.]
[0.  8.5 0.  0.  0.5 7.5]
[0. 0. 2. 1. 3. 0.]
[ 0.  0.  3. -7.  0. -8.]
[ 0.    0.    2.12  0.    2.62 -1.  ]
[ 0.  0.  0. 17.  9. 16.]
```

(γ) Ταιριάζοντας τις βασικές λύσεις του ερωτήματος (β) με τις κορυφές του ερωτήματος (α), έχουμε χρησιμοποιώντας τον παρακάτω κώδικα:

```python
# Round the basic solutions to 8 decimal places
basic_solutions_rounded = np.around(basic_solutions, 8)

# Define the solutions to match
solutions_to_match = np.array([[6.33333333, 7.33333333, 2.66666667],
                               [1., 10., -0.],
                               [-0., 8.5, -0.],
                               [-0., -0., 2.],
                               [-0., -0., -0.]])

# Match the basic solutions with the solutions to match
for i, solution in enumerate(basic_solutions_rounded):
    for j, solution_to_match in enumerate(solutions_to_match):
        try:
            if np.array_equal(solution[:3], solution_to_match):
                print(f"Basic solution {i+1} matches solution {j+1}\n")
                print("Basic solution:", solution)
                print("Solution to match:", solution_to_match,"\n")
                break
        except ValueError:
            continue
```

Και τα αποτελέσματα του φαίνονται παρακάτω:

```
Basic solution 1 matches vertex 1

Basic solution: [6.33333333 7.33333333 2.66666667 0.         0.         0.        ]
Vertex to match: [6.33333333 7.33333333 2.66666667]

Basic solution 4 matches vertex 2

Basic solution: [ 1. 10.  0.  0.  0.  8.]
Vertex to match: [ 1. 10. -0.]

Basic solution 16 matches vertex 3

Basic solution: [0.  8.5 0.  0.  0.5 7.5]
Vertex to match: [-0.   8.5 -0. ]

Basic solution 17 matches vertex 4

Basic solution: [0. 0. 2. 1. 3. 0.]
Vertex to match: [-0. -0.  2.]

Basic solution 20 matches vertex 5

Basic solution: [ 0.  0.  0. 17.  9. 16.]
Vertex to match: [-0. -0. -0.]
```

Και η βέλτιστη λύση που μειώνει στο ελάχιστο την αντικειμενική μας συνάρτηση είναι:

```
Optimal Value of Z = -88.0
x1 = 1.0
x2 = 10.0
x3 = 0.0
x4 = 0.0
x5 = 0.0
x6 = 8.0
```

Άσκηση 6.

A) min (Z) = −x1 − 4x2 − 5x3

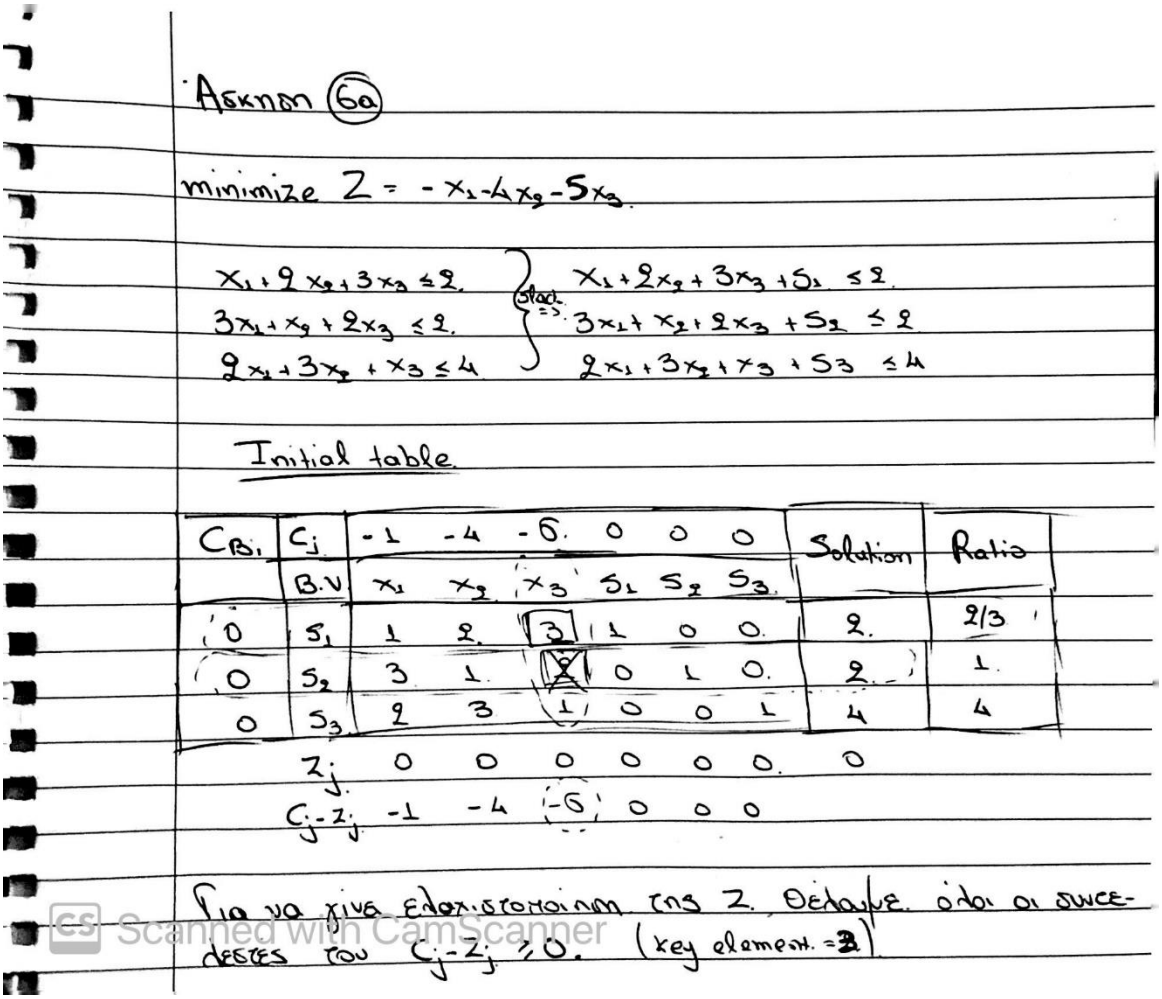
όταν

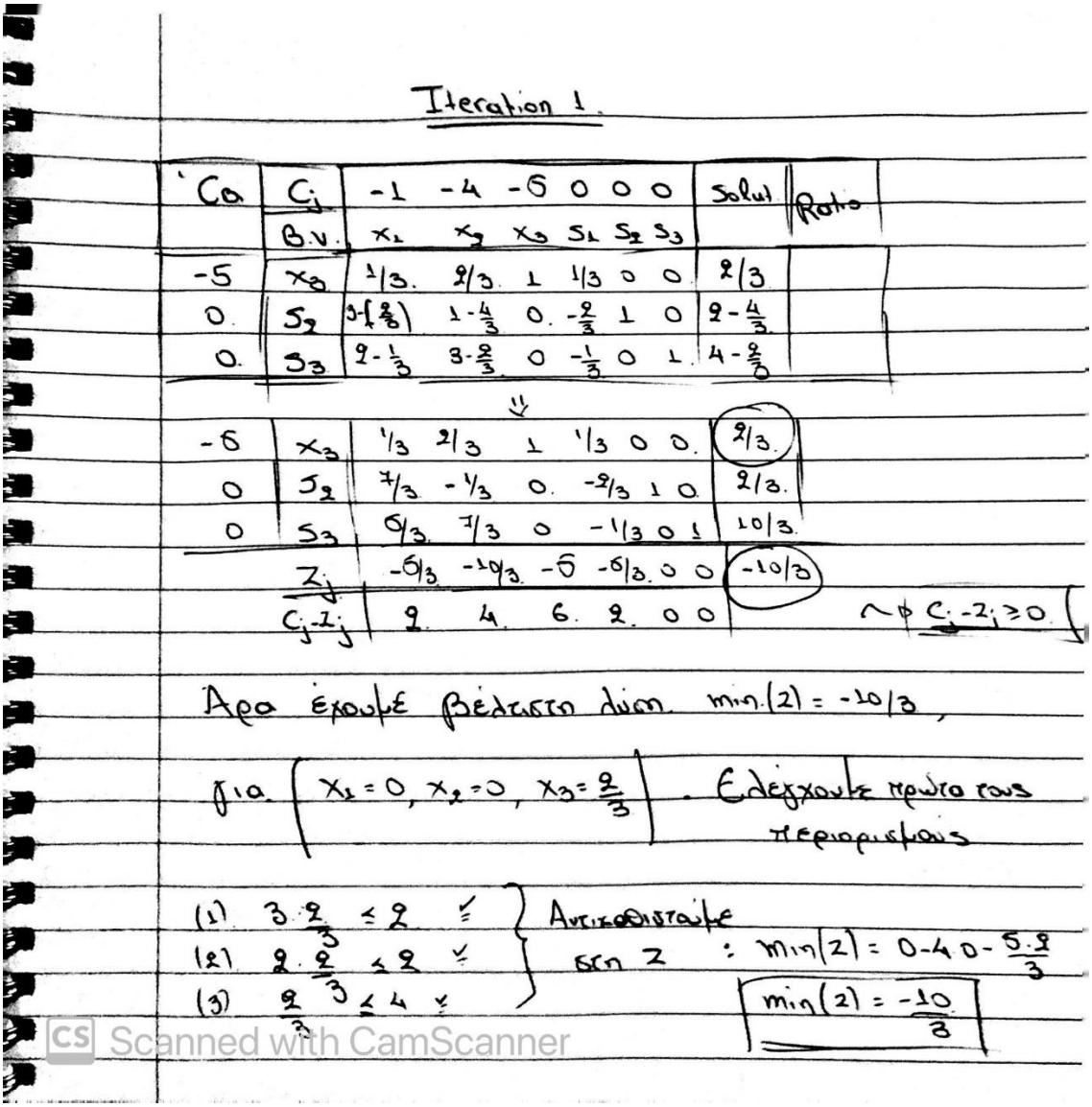
- x1 + 2x2 + 3x3 ≤ 2
- 3x1 + x2 + 2x3 ≤ 2
- 2x1 + 3x2 + x3 ≤ 4
- x1, x2, x3 ≥ 0

Η μέθοδος simplex ενημερώνει επαναληπτικά τη βάση εισάγοντας μια κατάλληλη μεταβλητή και αφήνοντας μια υπάρχουσα μεταβλητή σε κάθε επανάληψη μέχρι να βρεθεί η βέλτιστη λύση. Η έξοδος παρέχει πληροφορίες σχετικά με την πρόοδο της μεθόδου simplex σε κάθε επανάληψη, συμπεριλαμβανομένης της τρέχουσας βάσης, των συντελεστών της αντικειμενικής συνάρτησης για τις μεταβλητές που βρίσκονται και δεν βρίσκονται στη βάση, καθώς και της βέλτιστης λύσης και της αντικειμενικής τιμής στο τέλος.

Η φόρμουλα για τις νέες τιμές βρίσκεται από:

Old value – [corresponding key column value X corresponding new row value]





B) Παρακάτω φαίνεται ο κώδικας για την πραγματοποίηση μιας επανάληψης της μεθόδου Simplex:

```python
def simplex_iteration(tableau):
    # Find the pivot column and row
    pivot_col = min(range(len(tableau[0]) - 1), key=lambda i: tableau[0][i])
    ratios = [(i, tableau[i][-1] / tableau[i][pivot_col]) for i in range(1, len(tableau))]
    pivot_row = min(filter(lambda x: x[1] >= 0, ratios), key=lambda x: x[1])[0]

    # Update the pivot element
    pivot = tableau[pivot_row][pivot_col]
    pivot_row_vals = [val / pivot for val in tableau[pivot_row]]
    for i, row in enumerate(tableau):
        if i == pivot_row:
            continue
        factor = row[pivot_col] / pivot
        tableau[i] = [row[j] - factor * pivot_row_vals[j] for j in range(len(row))]

    # Update the basis
    tableau[pivot_row] = pivot_row_vals
    basis = [pivot_col] + [row.index(1) for row in tableau[1:]]

    return tableau, basis

def print_tableau(tableau):
    for row in tableau:
        for item in row:
            print("{:7.2f}".format(item), end="")
        print()
```

Παρακάτω το αποτέλεσμα της 1ης επανάληψης:

```
Initial tableau:
  -1.00  -4.00  -5.00   0.00   0.00   0.00   0.00
   1.00   2.00   3.00   1.00   0.00   0.00   2.00
   3.00   1.00   2.00   0.00   1.00   0.00   2.00
   2.00   3.00   1.00   0.00   0.00   1.00   4.00

Updated tableau:
  -0.44  -2.89  -3.33   0.56   0.00   0.00   1.11
   0.33   0.67   1.00   0.33   0.00   0.00   0.67
   2.78   0.56   1.33  -0.22   1.00   0.00   1.56
   1.89   2.78   0.67  -0.11   0.00   1.00   3.78

Optimal value: -1.1111111111111112
Optimal solution: [0, 0, 0.6666666666666666, 0, 1.5555555555555556, 3.7777777777777777]
```

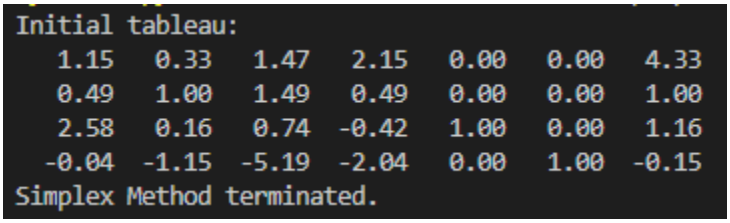
Παρακάτω το αποτέλεσμα της 2ης επανάληψης:

```
Initial tableau:
  -0.44  -2.89  -3.33   0.56   0.00   0.00   1.11
   0.33   0.67   1.00   0.33   0.00   0.00   0.67
   2.78   0.56   1.33  -0.22   1.00   0.00   1.56
   1.89   2.78   0.67  -0.11   0.00   1.00   3.78

Updated tableau:
   0.66  -0.66   0.00   1.66   0.00   0.00   3.34
   0.33   0.67   1.00   0.33   0.00   0.00   0.67
   2.34  -0.33   0.00  -0.66   1.00   0.00   0.67
   1.67   2.33   0.00  -0.33   0.00   1.00   3.33

Optimal value: -3.3411
Optimal solution: [0, 0, 0.67, 0, 0.6688999999999999, 3.3310999999999997]
```

Παρακάτω το αποτέλεσμα της 3ης επανάληψης:

```
Initial tableau:
   0.66  -0.66   0.00   1.66   0.00   0.00   3.34
   0.33   0.67   1.00   0.33   0.00   0.00   0.67
   2.34  -0.33   0.00  -0.66   1.00   0.00   0.67
   1.67   2.33   0.00  -0.33   0.00   1.00   3.33

Updated tableau:
   1.15   0.33   1.47   2.15   0.00   0.00   4.33
   0.49   1.00   1.49   0.49   0.00   0.00   1.00
   2.58   0.16   0.74  -0.42   1.00   0.00   1.16
  -0.04  -1.15  -5.19  -2.04   0.00   1.00  -0.15

Optimal value: -4.325074626865671
Optimal solution: [0, 1.0, 0, 0, 1.1625373134328358, -0.14761194029850744]
```

Σε κάθε επανάληψη απλά αλλάζουμε τον “Initial tableau”, και παίρνουμε τον “Updated tableau”. Όταν φτάσουμε σε Unbound tableau η μέθοδος τερματίζεται:



Στο πρόβλημά μας, οι μεταβλητές είναι οι x1, x2 και x3. Η αντικειμενική συνάρτηση είναι η ελαχιστοποίηση της τιμής -x1 - 4x2 - 5x3. Οι περιορισμοί είναι οι εξής:

$x1 + 2x2 + 3x3 \leq 2$

$3x1 + x2 + 2x3 \leq 2$

$2x1 + 3x2 + x3 \leq 4$

$x1, x2, x3 \geq 0$

Το γράφημα γειτνίασης Simplex αναπαριστά την εφικτή περιοχή του προβλήματος, η οποία είναι το σύνολο όλων των σημείων που ικανοποιούν τους περιορισμούς. Το γράφημα έχει επτά κόμβους, οι οποίοι αντιπροσωπεύουν τις κορυφές της εφικτής περιοχής. Οι κόμβοι αυτοί επισημαίνονται ως x1, x2, x3, s1, s2, s3 και x1x2x3s1s2s3.

Ο κόμβος x1x2x3s1s2s3 αντιπροσωπεύει την αρχική εφικτή λύση, η οποία αποτελεί την αφετηρία του αλγορίθμου simplex. Οι άλλοι κόμβοι αντιπροσωπεύουν τις κορυφές της εφικτής περιοχής, οι οποίες είναι τα σημεία τομής των γραμμών περιορισμών. Οι κόμβοι s1, s2 και s3 αντιπροσωπεύουν τις χαλαρές μεταβλητές, οι οποίες εισάγονται για τη μετατροπή των περιορισμών ανισότητας σε περιορισμούς ισότητας.

Οι ακμές του γραφήματος αντιπροσωπεύουν τους περιορισμούς που συνδέουν τις κορυφές. Ο γράφος έχει επτά ακμές, οι οποίες επισημαίνονται ανάλογα με τους περιορισμούς που αντιπροσωπεύουν. Ο αλγόριθμος μετακινείται από τη μία κορυφή στην άλλη κατά μήκος των ακμών του γραφήματος μέχρι να φτάσει στη βέλτιστη λύση. Η βέλτιστη λύση είναι η κορυφή με τη χαμηλότερη τιμή της αντικειμενικής συνάρτησης.

Με την οπτικοποίηση του γραφήματος γειτνίασης Simplex, μπορούμε να δούμε την ακολουθία των κορυφών και ακμών που επισκέπτεται ο αλγόριθμος Simplex καθώς κινείται από την αρχική εφικτή λύση προς τη βέλτιστη λύση.

```python
import networkx as nx
import matplotlib.pyplot as plt

# Define the graph
G = nx.DiGraph()

# Add nodes
G.add_node("x1x2x3s1s2s3")

# Add edges
G.add_edges_from([("x1x2x3s1s2s3", "x1s1"), ("x1x2x3s1s2s3", "x2"), ("x1x2x3s1s2s3", "x3"), ("x1s1", "s2"), ("x2", "s3"), ("x3", "s1")])

# Define node positions
pos = {"x1x2x3s1s2s3": (0, 0), "x1s1": (1, 1), "x2": (0, 2), "x3": (2, 2), "s2": (2, 1), "s3": (1, 0), "s1": (2, 0)}

# Draw the graph
nx.draw_networkx(G, pos, node_color='lightblue', node_size=1500, font_size=18, font_color='black', font_weight='bold')
plt.axis('off')
plt.show()
```