ΓΡΑΜΜΙΚΗ ΚΑΙ ΣΥΝΔΥΑΣΤΙΚΗ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

ΕΡΓΑΣΙΑ 2

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:

$Z = \max(2x1 + 4x2 + x3 + x4)$

(Π1) $x1 + 3x2 + x4 \leq 8$

(Π2) $2x1 + x2 \leq 6$

(Π3) $x2 + 4x3 + x4 \leq 6$

$x1, x2, x3, x4 \geq 0$

Εδώ ορίζουμε τις εξισώσεις που είναι και οι περιορισμοί μας. Σκοπός μας είναι η μεγιστοποίηση της αντικειμενικής συνάρτησης Ζ. Στον κώδικα από κάτω ορίζουμε τους πίνακες A, b, c και με τη μέθοδο simplex, επιλύουμε το πρόβλημα.

Παρακάτω ο κώδικας:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog

# Define the objective function coefficients
c = np.array([-2, -4, -1, -1])

# Define the constraint coefficients
A = np.array([[1, 3, 0, 1], [2, 1, 0, 0], [0, 1, 4, 1]])
b = np.array([8, 6, 6])

# Define the bounds for variables (x1, x2, x3, x4 >= 0)
bounds = [(0, None)] * 4

# Solve the linear programming problem using the simplex method
result = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='simplex')

# Extract the optimal solution and objective value
optimal_solution = result.x
objective_value = result.fun

# Print the optimal solution and objective value
print("Optimal Solution:")
print("x1 =", optimal_solution[0])
print("x2 =", optimal_solution[1])
print("x3 =", optimal_solution[2])
print("x4 =", optimal_solution[3])
print("Objective Value:", -objective_value)  # Negate the objective value to maximize
```

Και το αποτέλεσμά του:

```
Optimal Solution:
x1 = 2.0
x2 = 2.0
x3 = 1.0
x4 = 0.0
Objective Value: 13.0
```

Παρατηρούμε ότι για τη βέλτιστη λύση έχουμε **x1 = 2, x2 = 2, x3 = 1 και x4 = 0**. Οι τιμές αυτές ικανοποιούν τους περιορισμούς μας και μεγιστοποιούν την συνάρτηση

Z = max(2x1 + 4x2 + x3 + x4) = 13.

Χρησιμοποιώντας τον κώδικα παρακάτω:

```
basic_variables = np.where(np.isclose(optimal_solution, 0) == False)[0]
nonbasic_variables = np.where(np.isclose(optimal_solution, 0))[0]
variable_names = ['x1', 'x2', 'x3', 'x4']

basic_variable_names = [variable_names[i] for i in basic_variables]
nonbasic_variable_names = [variable_names[i] for i in nonbasic_variables]

print("Basic Variables:", basic_variable_names)
print("Non-Basic Variables:", nonbasic_variable_names)
```

Μπορούμε να διακρίνουμε τις βασικές από τις μη-βασικές μεταβλητές, και το αποτέλεσμά του έχει ως εξής:

```
Basic Variables: ['x1', 'x2', 'x3']
Non-Basic Variables: ['x4']
```

Βασικές Μεταβλητές: x1, x2, x3

Μη-Βασικές Μεταβλητές: x4

Στον παρακάτω κώδικα:

Το μειωμένο κόστος υπολογίζεται αφαιρώντας το γινόμενο των slack μεταβλητών και των constraint coefficients από τους συντελεστές της αντικειμενικής συνάρτησης. Το μειωμένο κόστος προστίθεται στη συνέχεια στην τελευταία γραμμή του πίνακα. Δημιουργείται ο καλύτερος βασικός πίνακας, συμπεριλαμβανομένων των συντελεστών της αντικειμενικής συνάρτησης, των συντελεστών περιορισμού, των τιμών RHS και του μειωμένου κόστους.

```
# Calculate the basis indices
basis_indices = np.where(np.abs(optimal_solution) > 1e-6)[0]

# Calculate the matrix B using the basis indices
B = A[:, basis_indices]
# Calculate the inverse of matrix B
B_inv = np.linalg.inv(B)

print("\nBest Matrix B:")
print(B)

print("\nInverse of Matrix B (B^-1):")
print(B_inv)
```

Βέλτιστος βασικός πίνακας:

```
Best Matrix B:
[[1 3 0]
 [2 1 0]
 [0 1 4]]
```

```
Inverse of Matrix B (B^-1):
[[-0.2   0.6   0.  ]
 [ 0.4  -0.2   0.  ]
 [-0.1   0.05  0.25]]
```

Σε αυτόν τον κώδικα, χρησιμοποιούμε τα A και b για την ανάκτηση των συντελεστών περιορισμού και των τιμών RHS. Στη συνέχεια, εντός του βρόχου, υπολογίζουμε την τιμή του περιορισμού στο βέλτιστο σημείο λαμβάνοντας το γινόμενο τελείας των συντελεστών περιορισμού (A[i]) και του βέλτιστου σημείου (optimal_point). Συγκρίνουμε αυτή την τιμή με την αντίστοιχη τιμή RHS (rhs[i]) χρησιμοποιώντας το np.isclose για να λάβουμε υπόψη την αριθμητική ακρίβεια. Εάν οι τιμές είναι κοντά, θεωρούμε τον περιορισμό ως δεσμευτικό και προσθέτουμε τον δείκτη του στη λίστα binding_constraints. Διαφορετικά, τον θεωρούμε ως μη δεσμευτικό και προσθέτουμε τον δείκτη του στη λίστα nonbinding_constraints.

```python
# Substituting optimal point into the constraints
optimal_point = result.x
rhs = b  # Right-hand side of the inequalities

binding_constraints = []
nonbinding_constraints = []

for i in range(len(A)):
    constraint_value = np.dot(A[i], optimal_point)
    if np.isclose(constraint_value, rhs[i]):
        binding_constraints.append(i + 1)
    else:
        nonbinding_constraints.append(i + 1)

print("Binding Constraints:", binding_constraints)
print("Non-Binding Constraints:", nonbinding_constraints)
```

```
Binding Constraints: [1, 2, 3]
Non-Binding Constraints: []
```

Συνεπώς και οι 3 περιορισμοί είναι δεσμευτικοί.

Με τη χρήση της Python μπορούμε να περιγράψουμε γεωμετρικά τη βέλτιστη κορυφή:

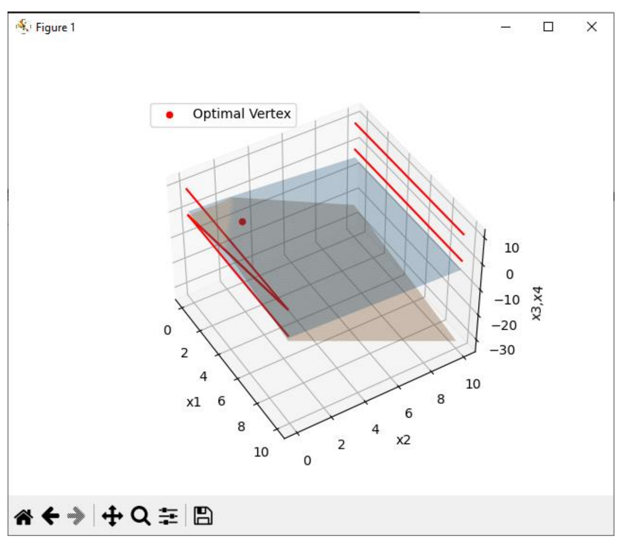
```python
# Plot the feasible region
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

x1, x2 = np.meshgrid(np.linspace(0, 10, 50), np.linspace(0, 10, 50))
x3 = np.minimum(2 - 0.5 * x2, (6 - x2 - x1) / 4)
x4 = np.minimum(8 - x1 - 3 * x2, 6 - x2 - 4 * x3)

ax.plot_surface(x1, x2, x3, alpha=0.3)
ax.plot_surface(x1, x2, x4, alpha=0.3)
ax.plot([0, 10], [0, 0], [0, 0], 'r-')
ax.plot([0, 10], [0, 0], [0, 10], 'r-')
ax.plot([0, 10], [10, 10], [0, 0], 'r-')
ax.plot([0, 10], [10, 10], [10, 10], 'r-')
ax.plot([0, 10], [0, 0], [10, 10], 'r-')
ax.plot([0, 10], [0, 0], [0, 10], 'r-')
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('x3,x4')

# Plot the optimal vertex
ax.scatter(optimal_solution[0], optimal_solution[1], optimal_solution[2], color='r', label='Optimal Vertex')
ax.legend()

plt.show()
```



Όπου η κόκκινη τελεία αποτελεί τη βέλτιστη κορυφή.

Β) Επιλέγουμε ως βασική μεταβλητή την x1 και ως μη βασική τη x4.

β) Επιλέγουμε ως βασική μεταβλητή τη x_1. Μια διαταραχή γ στον συντελεστή του x_1 δεν θα επηρεάσει τη βέλτιστη λύση αν:

$$\left(C_N^T - (C_B^T + \gamma e_k)^T B^{-1} N\right) \leq 0$$ ~> διάστημα ανοχής (για να μένει βέλτιστη)

$$\Delta z = \gamma e_k^T B^{-1} b$$ ~> μεταβολή στην αντικειμενική συνάρτηση

Έχουμε ότι: $B^{-1} = \begin{bmatrix} -0{,}2 & 0{,}6 & 0 \\ 0{,}4 & -0{,}2 & 0 \\ -0{,}1 & 0{,}05 & 0{,}2 \end{bmatrix}$ $N = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$ $b = [8\ 6\ 6]$

$B^{-1} \cdot N = \begin{bmatrix} -0{,}2 \\ 0{,}4 \\ 0{,}15 \end{bmatrix}$ $C_B^T = [2\ 4\ 1]$ $C_N^T = [1]$

Άρα έχουμε: $[1] - [2+\gamma\ \ 4\ \ 1] B^{-1} \cdot N =$

$$= (1) - (2+\gamma\ \ 4\ \ 1)\begin{pmatrix} -0{,}2 \\ 0{,}4 \\ 0{,}15 \end{pmatrix} =$$

$$= 1 - (-0{,}4 - 0{,}2\gamma + 1{,}6 + 0{,}15)$$

$$= 1 - (1{,}35 - 0{,}2\gamma) = -0{,}35 + 0{,}2\gamma \leq 0 \Leftrightarrow \boxed{\gamma \leq 1{,}75}$$

Άρα αν ο συντελεστής του x_1 είναι στο διάστημα: $[-\infty, 3{,}75]$ η βέλτιστη κορυφή δεν αλλάζει ενώ η μεταβολή στη Z:

$$\Delta z = \gamma \cdot \begin{pmatrix} 2 \\ 2 \\ 1 \end{pmatrix} = \gamma \cdot \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix}$$

- Επιλέγουμε ως μη βασική μεταβλητή τη x_4:

· Η μεταβολή που θα παρατηρήσουμε στη $\boxed{z=0}$ αφού $\boxed{x_4=0}$

Αλλά η λύση στην οποία βρισκόμαστε θα παραμένει βέλτιστη μόνο αν:

$$(1+\gamma) - (1,35) \le 0 \iff \gamma - 0,35 \le 0 \iff \boxed{\gamma \le 0,35}$$

· Έτσι αν ο συντελεστής της x_4 είναι στο διάστημα $(-\infty, 1,35]$ τότε η βέλτιστη κορυφή δεν αλλάζει και η αντικειμενική συνάρτηση παραμένει ίδια.

Μπορούμε να διαπιστώσουμε ότι η βέλτιστη λύση δεν παραμένει στην ίδια κορυφή αν η x1 διαταραχθεί κατά γ. Ενώ όσο και να διαταράξουμε τη x4 η βέλτιστη λύση θα παραμείνει στην ίδια κορυφή.

Γ) Στην περίπτωση μας και οι 3 περιορισμοί είναι δεσμευτικοί:

γ) Επιλέγουμε ως δεσμευτικό περιορισμό τον (π1). Μη δεσμευτικό δεν έχουμε σε αυτήν την περίπτωση:

- Το διάστημα ανοχής του ορίζεται από:

$$B^{-1}\cdot b + \gamma \cdot B^{-1} e_i \ge 0 \qquad B^{-1}b = \begin{pmatrix}2\\2\\1\end{pmatrix} = (2\ 2\ 1)^T$$

(π1): $x_1 + 3x_2 + x_4 \le 8 + \gamma$

$$\text{Άρα } (2\ 2\ 1)^T + \gamma (1\ 3\ 0)\begin{pmatrix}-0,2 & 0,6 & 0\\ 0,4 & -0,2 & 0\\ -0,1 & 0,05 & 0,25\end{pmatrix} \ge 0 \iff$$

$$\iff (2\ 2\ 1)^T + \gamma(1,6\ -0,2\ 0,05)^T \ge 0 \iff$$

$$\left.\begin{array}{l} 2 + 1,6\gamma \ge 0 \\ 2 - 0,2\gamma \ge 0 \\ 1 + 0,05\gamma \ge 0 \end{array}\right\} \begin{array}{l} \gamma \ge -1,25 \\ \gamma \le 10 \\ \gamma \ge -20 \end{array}$$

$$\text{Άρα } \boxed{-1,25 \le \gamma \le 10}$$

↳ Διάστημα ανοχής.

Δ) Για μη βασική μεταβλητή επιλέγουμε τη x4 και ο συντελεστής της στην αρχική συνάρτηση είναι 1.

Με τη χρήση του κώδικα παρακάτω:

```
# Check if the coefficient of the non-basic variable is already zero
if c[nonbasic_variable_index] == 0:
    print("The coefficient of the non-basic variable is already zero.")
else:
    # Find the entering constraint
    entering_constraint_index = None
    min_ratio = np.inf

    for i in range(len(A)):
        if A[i, nonbasic_variable_index] != 0:
            ratio = b[i] / A[i, nonbasic_variable_index]
            if ratio >= 0 and ratio < min_ratio:
                min_ratio = ratio
                entering_constraint_index = i

    if entering_constraint_index is None:
        print("No entering constraint found.")
    else:
        # Perform the pivot operation to convert the non-basic variable into a basic variable
        pivot_element = A[entering_constraint_index, nonbasic_variable_index]
        A[entering_constraint_index, :] /= pivot_element
        b[entering_constraint_index] /= pivot_element

        for i in range(len(A)):
            if i != entering_constraint_index:
                ratio = A[i, nonbasic_variable_index] / A[entering_constraint_index, nonbasic_
                A[i, :] -= ratio * A[entering_constraint_index, :]
                b[i] -= ratio * b[entering_constraint_index]

        # Update the objective function coefficient of the non-basic variable
        c -= c[nonbasic_variable_index] * A[entering_constraint_index, :] / pivot_element

        print("Objective function coefficients after conversion:")
        print(c)
```

Ο κώδικας επαναλαμβάνει τους περιορισμούς για να βρει τον περιορισμό εισόδου. Υπολογίζει τον λόγο της τιμής του περιορισμού (b) προς τον αντίστοιχο συντελεστή της μη βασικής μεταβλητής (A[i, nonbasic_variable_index]). Επιλέγει τον περιορισμό με τον μικρότερο θετικό λόγο ως τον εισερχόμενο περιορισμό. Εάν δεν βρεθεί περιορισμός εισόδου, αυτό σημαίνει ότι η μη βασική μεταβλητή μπορεί να εισέλθει σε οποιονδήποτε περιορισμό χωρίς να παραβιάζει τη σκοπιμότητα. Σε αυτή την περίπτωση, ο κώδικας εκτυπώνει ένα μήνυμα που υποδεικνύει ότι δεν βρέθηκε περιορισμός εισόδου. Εάν βρεθεί περιορισμός εισόδου, ο κώδικας εκτελεί την πράξη περιστροφής. Διαιρεί τη γραμμή περιστροφής (εισερχόμενος περιορισμός) με το στοιχείο περιστροφής (ο συντελεστής της μη βασικής μεταβλητής στον εισερχόμενο περιορισμό). Αυτό καθιστά το στοιχείο περιστροφής ίσο με 1. Στη συνέχεια, ο κώδικας ενημερώνει τις υπόλοιπες γραμμές αφαιρώντας πολλαπλάσια της γραμμής περιστροφής για να εξαλείψει τους μη μηδενικούς συντελεστές της μη βασικής μεταβλητής στις άλλες γραμμές. Αυτό εξασφαλίζει ότι η μη βασική μεταβλητή γίνεται μηδέν σε όλους τους άλλους περιορισμούς. Τέλος, ο κώδικας ενημερώνει τους συντελεστές της αντικειμενικής συνάρτησης. Αφαιρεί το γινόμενο του συντελεστή της μη βασικής μεταβλητής στην αντικειμενική συνάρτηση και της αντίστοιχης γραμμής της στήλης περιστροφής διαιρούμενο με το στοιχείο περιστροφής. Αυτό εξασφαλίζει ότι ο συντελεστής της μη βασικής μεταβλητής γίνεται μηδέν, καθιστώντας την βασική μεταβλητή.

Το αποτέλεσμα μας δείχνει ότι ο συντελεστής της x4 θα γίνει 0 στο Objective function.

```
Objective function coefficients after conversion:
[-2. -3.  3.  0.]
```

Άσκηση 2.

A)

Ασκηση 2

$\max.(3x_1 - 2x_2 - 5x_3 + 7x_4 + 8x_5)$ οταν,

$x_2 - x_3 + 3x_4 - 4x_5 = -6$ (1)

$2x_1 + 3x_2 - 3x_3 - x_4 \geq 2$ (2)

$x_1 + 2x_3 - 2x_4 \leq -8$ (3)

$-2 \leq x_1 \leq 10$ (4)

$5 \leq x_2 \leq 25$ (5)

$x_3, x_4 \geq 0$, $x_5 \in \mathbb{R}$

→ Πρέπει πρώτα να μετατρέψουμε το πρόβλημα σε κανονικό. Πρώτιστον:

• Ο περιορισμός (2) έχει "≥", οπότε χρειάζεται μετατροπή:

$(2) \xrightarrow{(-1)} \boxed{-2x_1 - 3x_2 + 3x_3 + x_4 \leq -2}$ (2)

• Ο περιορισμός (1) έχει "=", οπότε χρειάζεται μετατροπή:

$(1) \rightarrow x_2 - x_3 + 3x_4 - 4x_5 \leq -6$ (1.1)

$\rightarrow -x_2 + x_3 - 3x_4 + 4x_5 \leq 6$ (1.2)

• Οι περιορισμοί (4) και (5) έχουν ως εξής:

$-x_1 \leq 2$ (4.1)

$x_1 \leq 10$ (4.2)

$-x_2 \leq -5$ (5.1)

$x_2 \leq 25$ (5.2)

· Συνεπώς το κανονικό πρωτεύον πρόβλημα έχει ως εξής:

$\max.(3x_1 - 2x_2 - 5x_3 + 7x_4 + 8x_5)$, όταν

$$x_2 - x_3 + 3x_4 - 4x_5 \leq -6 \quad (1.1)$$
$$-x_2 + x_3 - 3x_4 + 4x_5 \leq 6 \quad (1.2)$$
$$-2x_1 - 3x_2 + 3x_3 + x_4 \leq -2 \quad (2)$$
$$x_1 + 2x_3 - 2x_4 \leq -5 \quad (3)$$
$$-x_1 \leq 2 \quad (4.1)$$
$$x_1 \leq 10 \quad (4.2)$$
$$-x_2 \leq -5 \quad (5.1)$$
$$x_2 \leq 25 \quad (5.2)$$

$\Downarrow$ Μετατροπή σε δυϊκό

$\min.(-6y_1 + 6y_2 - 2y_3 - 5y_4 + 2y_5 + 10y_6 - 5y_7 + 25y_8)$, όταν

$$-2y_3 + y_4 - y_5 + y_6 \geq 3$$
$$y_1 - y_2 - 3y_3 - y_7 + y_8 \geq -2$$
$$-y_1 + y_2 + 3y_3 + 2y_4 \geq -5$$
$$3y_1 - 3y_2 + y_3 - 2y_4 \geq 7$$
$$-4y_1 + 4y_2 \geq 8$$

B) Χρησιμοποιώντας τον παρακάτω κώδικα:

```python
# Define the objective function coefficients
c = np.array([3, -2, -5, 7, 8, 0, 0, 0, 0, 0, 0, 0])

# Define the constraint coefficients matrix
A = np.array([
    [0, 1, -1, 3, -4, 0, 0, 0, 0, 0, 0, 1],
    [-2, -3, 3, 1, 0, 1, 0, 0, 0, 0, 0, 0],
    [1, 0, 2, -2, 0, 0, 1, 0, 0, 0, 0, 0],
    [-1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
    [0, -1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0]
])
# Define the key variables
key_variables = [1, 3, 4, 7, 8, 9, 10]

# Define the right-hand side of constraints
b = np.array([-6, -2, -5, 2, 10, -5, 25])

# Extract the columns corresponding to the key variables
B = A[:, key_variables]
# Print the basic table
print("Basic Table:")
print(B)

# Define the non-key variables
non_key_variables = [i for i in range(A.shape[1]) if i not in key_variables]

# Extract the columns corresponding to the non-key variables
BC = A[:, non_key_variables]

# Print the complement matrix BC
print("Complement Matrix (BC):")
print(BC)
```

Οι συντελεστές της αντικειμενικής συνάρτησης c αντιπροσωπεύουν τους συντελεστές των μεταβλητών της αντικειμενικής συνάρτησης που πρέπει να μεγιστοποιηθεί.

Ο πίνακας συντελεστών περιορισμών A αντιπροσωπεύει τους συντελεστές των μεταβλητών στους περιορισμούς. Κάθε γραμμή αντιστοιχεί σε έναν περιορισμό και κάθε στήλη αντιστοιχεί σε μια μεταβλητή. Οι βασικές μεταβλητές ορίζονται ως ένας κατάλογος δεικτών που αντιπροσωπεύουν τις στήλες των βασικών μεταβλητών στον πίνακα συντελεστών A. Η δεξιά πλευρά των περιορισμών b αντιπροσωπεύει τις τιμές στη δεξιά πλευρά των περιορισμών ανισότητας. Ο κώδικας εξάγει τις στήλες που αντιστοιχούν στις μεταβλητές-κλειδιά από τον πίνακα συντελεστών A για να σχηματίσει τον βασικό πίνακα B.

Ο κώδικας εξάγει τις στήλες που αντιστοιχούν στις μη βασικές μεταβλητές από τον πίνακα συντελεστών A για να σχηματίσει τον συμπληρωματικό πίνακα BC. Τα κατώτερα και ανώτερα όρια για τις μεταβλητές ορίζονται με τη χρήση του καταλόγου x_bounds. Σε αυτή την περίπτωση, όλες οι μεταβλητές έχουν κατώτερο όριο το 0 και κανένα ανώτερο όριο (χωρίς όρια). Το πρόβλημα γραμμικού προγραμματισμού επιλύεται με τη χρήση της συνάρτησης linprog από τη scipy.optimize. Οι συντελεστές της αντικειμενικής συνάρτησης (c), ο πίνακας συντελεστών (A), η δεξιά πλευρά των περιορισμών (b) και τα όρια των μεταβλητών (x_bounds) παρέχονται ως είσοδοι στη συνάρτηση. Ο κώδικας ελέγχει αν το πρόβλημα επιλύθηκε επιτυχώς, ελέγχοντας το χαρακτηριστικό success του αντικειμένου result. Εάν το πρόβλημα επιλυθεί επιτυχώς, ο κώδικας εκτυπώνει την τιμή της βέλτιστης λύσης και στη συνέχεια εκτυπώνει τις τιμές των πρωταρχικών μεταβλητών.

- **Ο βασικός πίνακας B για βασικές μεταβλητές: x2,x4,x5,x8.x9,x10,x11**

```
Basic Table:
[[ 1  3 -4  0  0  0  0]
 [-3  1  0  0  0  0  0]
 [ 0 -2  0  0  0  0  0]
 [ 0  0  0  1  0  0  0]
 [ 0  0  0  0  1  0  0]
 [-1  0  0  0  0  1  0]
 [ 1  0  0  0  0  0  1]]
```

- **Ο συμπληρωματικός πίνακας B^C για το δυικό πρόβλημα.**

```
# Compute the complementary matrix
complementary_matrix = np.identity(B.shape[0]) - B
```

```
Complementary Matrix:
[[ 0. -3.  4.  0.  0.  0.  0.]
 [ 3.  0.  0.  0.  0.  0.  0.]
 [ 0.  2.  1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.]
 [ 1.  0.  0.  0.  0.  0.  0.]
 [-1.  0.  0.  0.  0.  0.  0.]]
```

Με χρήση του παρακάτω κώδικα:

```
# Solve the linear programming problem
result = linprog(c, A_ub=A, b_ub=b, bounds=x_bounds, method='simplex')

# Check if the problem is successfully solved
if result.success:
    print("Optimal solution found!")
    print("Objective function value:", result.fun)

    # Print the primal solution
    print("Primal solution:")
    for i, variable in enumerate(result.x):
        print("x", i+1, "=", variable)

else:
    print("Problem could not be solved. Status:", result.message)
```

Έχουμε το παρακάτω αποτέλεσμα:

```
Optimal solution found!
Objective function value: 44.5
Primal solution:
x 1 = 0.0
x 2 = 5.0
x 3 = 0.0
x 4 = 2.5
x 5 = 4.625
x 6 = 10.5
x 7 = 0.0
x 8 = 2.0
x 9 = 10.0
x 10 = 0.0
x 11 = 20.0
x 12 = 0.0
```

x1 = 0,x2 = 5,x3 =0,x4 = 2.5,x5 = 4.6,x6 = 10.6,

x7 = 0,x8 = 2,x9 = 10, x10 = 0, x11 = 10, x12 = 10

Χρησιμοποιώντας τον παρακάτω κώδικα επιλύουμε το δυικό πρόβλημα:

```python
from pulp import *

# Create a minimization problem instance
prob = LpProblem("Dual Problem", LpMinimize)

# Define the variables
y1 = LpVariable("y1", lowBound=0)
y2 = LpVariable("y2", lowBound=0)
y3 = LpVariable("y3", lowBound=0)
y4 = LpVariable("y4", lowBound=0)
y5 = LpVariable("y5", lowBound=0)
y6 = LpVariable("y6", lowBound=0)
y7 = LpVariable("y7", lowBound=0)
y8 = LpVariable("y8", lowBound=0)

# Set the objective function (negation of the original coefficients)
prob += -6*y1 + 6*y2 - 2*y3 - 5*y4 + 2*y5 + 10*y6 - 5*y7 + 25*y8

# Add the constraints
prob += -2*y3 + y4 - y5 + y6 >= 3
prob += y1 - y2 - 3*y3 - y7 + y8 >= -2
prob += -y1 + y2 + 3*y3 + 2*y4 >= -5
prob += 3*y1 - 3*y2 + y3 - 2*y4 >= 7
prob += -4*y1 + 4*y2 >= 8

# Solve the problem
prob.solve()

# Print the optimal solution
print("Optimal Solution:")
for variable in prob.variables():
    print(f"{variable.name}: {variable.varValue}")

# Print the optimal objective value (negation of the original objective value)
print("Optimal Objective Value:")
print(value(prob.objective))
```

Και το αποτέλεσμα έχει ως εξής:

```
Optimal Solution:
y1: 0.0
y2: 2.0
y3: 13.0
y4: 0.0
y5: 0.0
y6: 29.0
y7: 0.0
y8: 39.0
Optimal Objective Value:
1251.0
```

Τέλος αν ο B είναι ένας βέλτιστος πρώτος πίνακας βάσης από ένα τυπικό LO μοντέλο, τότε $y^* = (B^{-1})^T * C_{BI}$

• Ο περιορισμός (2) γίνεται: $-2x_1 - 3x_2 + 3x_3 + x_4 \le -2$ (2)

Εξετάζουμε το Θεώρημα Ασθενούς Δυϊκότητας:

Όπου αν x^0 και y^0 είναι εφικτές λύσεις τότε θα ισχύει:

$$c^T \cdot x^0 \le b^T \cdot y^0$$

$$\begin{pmatrix} 3 & -2 & -5 & 7 & 8 \ldots \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \le \begin{pmatrix} -6 & 6 & -2 & -5 & 2 & 10 & -5 & 25 \end{pmatrix} \begin{pmatrix} y_1 \\ \vdots \\ y_9 \end{pmatrix}$$

$$\boxed{44.5 \le 1251}$$

που ισχύει άρα ισχύει το θεώρημα ασθενούς δυϊκότητας

Γ) Χρησιμοποιώντας την βιβλιοθήκη Pulp και τον παρακάτω κώδικα:

```python
from pulp import *

# Create a maximization problem instance
prob = LpProblem("Primal Problem", LpMaximize)

# Define the variables
x1 = LpVariable("x1", lowBound=-2, upBound=10)
x2 = LpVariable("x2", lowBound=5, upBound=25)
x3 = LpVariable("x3", lowBound=0)
x4 = LpVariable("x4", lowBound=0)
x5 = LpVariable("x5")

# Set the objective function
prob += 3*x1 - 2*x2 - 5*x3 + 7*x4 + 8*x5

# Add the constraints
prob += x2 - x3 + 3*x4 - 4*x5 == -6
prob += 2*x1 + 3*x2 - 3*x3 - x4 >= 2
prob += x1 + 2*x3 - 2*x4 <= -5

# Solve the problem
prob.solve()

# Print the status of the solution
print("Status:", LpStatus[prob.status])

# Print the optimal solution
print("Optimal Solution:")
for variable in prob.variables():
    print(f"{variable.name}: {variable.varValue}")

# Print the optimal objective value
print("Optimal Objective Value:")
print(value(prob.objective))
```

Το αποτέλεσμα του πρωτεύοντος προβλήματος έχει ως εξής:

```
Status: Optimal
Optimal Solution:
x1: 10.0
x2: 25.0
x3: 0.0
x4: 93.0
x5: 77.5
Optimal Objective Value:
1251.0
```

Η επίλυση του αντίστοιχου δυικού του βρέθηκε παραπάνω. Εξετάζουμε αν ισχύει το θεώρημα του ισχυρής δυικότητας:

$c^T x^0 = b^T y^0$ Το οποίο ισχύει αφού:

[3 -2 -5 7 8] * [10 25 0 93 77.5]^T = [-6 6 -2 -5 2 10 -5 25] * [0 2 13 0 0 29 0 39]

1251 = 1251.

Άσκηση 3.

Για τη μοντελοποίηση του προβλήματος προγραμματισμού ανθρώπινων πόρων, μπορούμε να χρησιμοποιήσουμε ακέραιο γραμμικό προγραμματισμό (ILP) με μεταβλητές απόφασης που αντιπροσωπεύουν τον αριθμό των εργαζομένων που θα απασχοληθούν σε κάθε βάρδια. Ο στόχος είναι η ελαχιστοποίηση του συνολικού ημερήσιου κόστους της εταιρείας σε μισθούς για το συγκεκριμένο τμήμα.

Έστω x_i ο αριθμός των εργαζομένων που θα απασχοληθούν στη βάρδια κατά την ώρα i όπου i = 1,2,3,4,5…,10. **Δηλαδή i = 1 -> 06:00 – 08:00, i=2 ->08:00-10:00 κ.ο.κ.**

Οι περιορισμοί είναι οι εξής:

Ο συνολικός αριθμός των εργαζομένων σε κάθε δίωρη ώρα πρέπει να πληροί ή να υπερβαίνει τον ελάχιστο απαιτούμενο αριθμό:

x1 >= 48, x2 >= 79, x3 >= 65, x4>= 87, x5>= 64, x6>= 73, x7 >= 82, x8>= 43, x9 >= 52

, x10>= 15

Η αντικειμενική συνάρτηση είναι απαρτίζεται από τον ελάχιστο αριθμό των εργαζομένων σε κάθε βάρδια επί το κόστος της βάρδιας τους:

$Min(Z) = 170*(x_1 + x_2 + x_3 + x_4) + 160*(x_2 + x_3 + x_4 + x_5) + 175*(x_4 + x_5 + x_6 + x_7) + 180*(x_6 + x_7 + x_8 + x_9) + 195*(x_9 + x_{10})$

Ο αριθμός των εργαζομένων σε κάθε βάρδια πρέπει να είναι μη αρνητικός:

xij >= 0 για όλα τα i,j.

Χρησιμοποιώντας τον παρακάτω κώδικα:

```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.")
```

Το αποτέλεσμα έχει ως εξής:

```
Status: Optimal
Total Cost: $94770.0
Shift 1: 48 employees
Shift 2: 295 employees
Shift 3: 155 employees
Shift 4: 52 employees
Shift 5: 58 employees
```

Άσκηση 4.

Το δυικό πρόβλημα έχει ως εξής:

Το πρόβλημα γίνεται:

$$\min.\left(6y_1+4y_2+2y_3+y_4\right), \text{ όταν}$$

$$y_1+y_2+y_3+y_4 \geq 2.$$

$$-y_1-y_2+y_4 \geq -3.$$

$$2y_1-2y_2+y_3-y_4 \geq 1.$$

$$-2y_1+2y_2-y_3+y_4 \geq -1$$

$$y_1, y_2, y_3 \geq 1.$$

Ξέρουμε ότι η x^0 είναι βέλτιστη, αν υπάρχει εφικτή λύση y^0 τ.ω:

$$c^T \cdot x^0 = b^T y^0 \quad \Rightarrow$$

$$\begin{pmatrix}1 & 2 & 1 & -3 & 1 & 1 & -1\end{pmatrix} \cdot \begin{pmatrix}7\\0\\9/2\\0\\3\\0\\1/2\end{pmatrix} = \begin{pmatrix}6 & 4 & 2 & 1\end{pmatrix}\begin{pmatrix}y_1\\y_2\\y_3\\y_4\end{pmatrix} \Leftrightarrow$$

$$\boxed{12 = 6y_1+4y_2+2y_3+y_4 \quad (1)}$$

Συνεπώς αν υπάρχει y^0 που ικανοποιεί την (1), τότε η λύση είναι βέλτιστη

Βλέποντας την (1) μπορούμε εύκολα να εντοπίσουμε τη λύση y_1 [illegible] $y = (1, 1, 1, 0)$.

$$\begin{matrix}\downarrow & \downarrow & \downarrow & \downarrow\\ y_1 & y_2 & y_3 & y_4\end{matrix}$$

• Η οποία ικανοποιεί <u>και</u> τους περιορισμούς του δυικού προβλήματος. Συνεπώς η αρχική λύση $x = (7, 0, 9/2, 0, 3, 0, 1/2)$ είναι <u>βέλτιστη</u>

Για επιβεβαίωση μπορούμε να προσδιορίσουμε αν η λύση x = (7, 0, 5/2, 0, 3, 0, 1/2) είναι βέλτιστη για το συγκεκριμένο πρόβλημα γραμμικού προγραμματισμού, χρησιμοποιώντας τον αλγόριθμο Simplex όπως φαίνεται παρακάτω:

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

# Define the objective function coefficients
c = np.array([-1, -2, -1, 3, -1, -1, 1])

# Define the constraint coefficients matrix
A = np.array([
    [1, 1, 0, -1, 0, 2, -2],
    [0, 1, 0, -1, 1, -2, 2],
    [0, 1, 1, 0, 0, 1, -1],
    [0, 1, 0, -1, 0, -1, 1],
])

# Define the right-hand side of constraints
b = np.array([6, 4, 2, 1])

# Define the bounds for variables
bounds = [(0, None), (0, None),(0, None), (0, None), (0, None), (0, None), (0, None)]

# Solve the linear programming problem using the simplex method
result = linprog(c, A_ub=A, b_ub=b, bounds=bounds, method='simplex')

# Check if the primal problem is successfully solved
if result.success:
    print("Primal problem solved successfully!")
    print("Objective function value:", -result.fun)

    # Print the primal solution
    print("Primal solution:")
    for i, variable in enumerate(result.x):
        print("x", i+1, "=", variable)
else:
    print("Primal problem could not be solved. Status:", result.message)

```

Και το αποτέλεσμα έχει ως εξής:

```
Primal problem solved successfully!
Objective function value: 12.0
Primal solution:
x 1 = 8.0
x 2 = 0.0
x 3 = 3.0
x 4 = 0.0
x 5 = 2.0
x 6 = 0.0
x 7 = 1.0
```

Όπως φαίνεται ο αλγόριθμος μας βρίσκει μια άλλη λύση η οποία ικανοποιεί τους περιορισμούς και έχει το ίδιο αποτέλεσμα με τη δική μας στην αντικειμενική συνάρτηση.

Άσκηση 5.

Αυτός ο κώδικας υλοποιεί τον αλγόριθμο Branch & Bound χρησιμοποιώντας μια ουρά προτεραιότητας (priority_queue) για την αποθήκευση των κόμβων. Οι κόμβοι ταξινομούνται με βάση το κέρδος τους και σε κάθε επανάληψη επιλέγεται ο κόμβος με το μεγαλύτερο κέρδος για διακλάδωση.

Η κλάση Node αναπαριστά κάθε κόμβο στο δέντρο αναζήτησης και παρακολουθεί το επίπεδο, το κέρδος, το βάρος και τη λίστα include. Η μέθοδος __lt__ παρακάμπτεται για να καθορίσει τη σειρά των κόμβων στην ουρά προτεραιότητας με βάση το κέρδος τους. Ο κώδικας της φαίνεται παρακάτω:

```python
class Node:
    def __init__(self, level, profit, weight, include):
        self.level = level
        self.profit = profit
        self.weight = weight
        self.include = include

    def __lt__(self, other):
        return self.profit > other.profit  # Sort nodes by profit (higher is better)
```

Η συνάρτηση branch_and_bound_backpack εκτελεί τον αλγόριθμο Branch & Bound. Αρχικοποιεί την ουρά προτεραιότητας με τον κόμβο-ρίζα και συνεχίζει μέχρι η ουρά να αδειάσει. Σε κάθε βήμα, ανοίγει τον κόμβο με το υψηλότερο κέρδος, διακλαδίζεται συμπεριλαμβάνοντας ή αποκλείοντας το επόμενο στοιχείο και προσθέτει τους νέους κόμβους στην ουρά προτεραιότητας. Ο κώδικας της συνάρτησης παρακάτω:

```python
def branch_and_bound_backpack(weights, profits, capacity):
    n = len(weights)
    best_solution = [0] * n
    root = Node(-1, 0, 0, [])
    priority_queue = []
    heapq.heappush(priority_queue, root)
    max_profit = 0

    while priority_queue:
        node = heapq.heappop(priority_queue)

        level = node.level + 1
        weight = node.weight
        profit = node.profit

        if level == n:
            if profit > max_profit:
                max_profit = profit
                best_solution = node.include
            continue

        # Include the next item
        new_weight = weight + weights[level]
        new_profit = profit + profits[level]
        if new_weight <= capacity:
            include = node.include[:]
            include.append(1)
            heapq.heappush(priority_queue, Node(level, new_profit, new_weight, include))

        # Exclude the next item
        include = node.include[:]
        include.append(0)
        heapq.heappush(priority_queue, Node(level, profit, weight, include))

    return best_solution, max_profit
```

Τέλος, η συνάρτηση επιστρέφει την καλύτερη λύση (μια λίστα από 0 και 1 που δείχνει αν κάθε στοιχείο περιλαμβάνεται) και το μέγιστο κέρδος.

```
Best Solution: [0, 0, 0, 1, 1, 0, 1]
Max Profit: 217
```

Με βάση τα δεδομένα του προβλήματος ορίζουμε τις μεταβλητές weights,profits,capacity για να λύσουμε το πρόβλημα του σακιδίου πλάτης χρησιμοποιώντας τον αλγόριθμο Branch & Bound με προσέγγιση Best First.

```python
# Problem data
weights = [3, 4, 3, 3, 15, 13, 16]
profits = [12, 12, 9, 15, 90, 26, 112]
capacity = 35

# Solve the problem
best_solution, max_profit = branch_and_bound_backpack(weights, profits, capacity)

# Print the result
print("Best Solution:", best_solution)
print("Max Profit:", max_profit)
```

Άσκηση 6.

Α) Για να μοντελοποιήσουμε το συγκεκριμένο πρόβλημα χρησιμοποιώντας ακέραιο γραμμικό προγραμματισμό, πρέπει να ορίσουμε τις μεταβλητές απόφασης, την αντικειμενική συνάρτηση και τους περιορισμούς.

Ας συμβολίσουμε τις μεταβλητές απόφασης ως εξής:

xj: Δυαδική μεταβλητή που δείχνει αν επιλέγεται το επενδυτικό πρόγραμμα j. Δηλαδή για **xj = 1** σημαίνει ότι το πρόγραμμα επιλέχθηκε ενώ για **xj = 0** όχι.

Τώρα, ας μοντελοποιήσουμε το πρόβλημα:

Αντικειμενική συνάρτηση:

$Z = P1*x1 + P2*x2 + P3*x3 + P4*x4 + P5*x5 + P6*x6 + P7*x7 + P8*x8 + P9*x9 + P10*x10$

(Όπου Pj το κέρδος για το j πρόγραμμα που επιλέχθηκε)

Περιορισμοί:

Η συνολική επένδυση δεν πρέπει να υπερβαίνει το διαθέσιμο ποσό Q:

- $C1x1 + C2x2 + C3x3 + C4x4 + C5x5 + C6x6 + C7x7 + C8x8 + C9x9 + C10x10 \leq Q$ (Όπου Cj το ποσό της επένδυσης στο j πρόγραμμα)

Τα επενδυτικά προγράμματα 3 και 4 είναι αμοιβαία αποκλεισμένα:

- $x3 + x4 \leq 1$

Τα επενδυτικά προγράμματα 5 και 6 είναι αμοιβαία αποκλεισμένα:

- $x5 + x6 \leq 1$

Τα επενδυτικά προγράμματα 5 και 6 απαιτούν επένδυση είτε στο 3 είτε στο 4:

- $x5 + x6 \leq x3 + x4$

Η επιχείρηση πρέπει να επενδύσει σε τουλάχιστον δύο και το πολύ σε τέσσερα από τα προγράμματα 1, 2, 7, 8, 9, 10:

- $2 \leq x1 + x2 + x7 + x8 + x9 + x10 \leq 4$

Οι μεταβλητές απόφασης είναι δυαδικές:

$xj \in \{0, 1\}$ για $j = 1, 2, ..., 10$

Αυτή η διατύπωση ακέραιου γραμμικού προγραμματισμού αναπαριστά το πρόβλημα, όπου η αντικειμενική συνάρτηση μεγιστοποιεί το συνολικό μακροπρόθεσμο κέρδος με βάση τους δεδομένους περιορισμούς.

B) Δίνουμε τιμές στις παραμέτρους Pj, Cj και Q:

```python
# Define the parameters
P = [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000]  # Long-term profit for each program
C = [50, 100, 150, 200, 250, 300, 350, 400, 450, 500]  # Investment required for each program
Q = 1000  # Total available investment
```

Και χρησιμοποιώντας τον παρακάτω κώδικα:

```python
# Create the LP problem
prob = LpProblem("Investment_Decisions", LpMaximize)

# Create the decision variables
x = LpVariable.dicts("x", range(1, 11), cat='Binary')  # x1 to x10

# Set the objective function
prob += lpSum(P[j - 1] * x[j] for j in range(1, 11))

# Add the constraints
prob += lpSum(C[j - 1] * x[j] for j in range(1, 11)) <= Q  # Total investment constraint
prob += x[3] + x[4] <= 1  # Mutual exclusivity of programs 3 and 4
prob += x[5] + x[6] <= 1  # Mutual exclusivity of programs 5 and 6
prob += x[5] + x[6] <= x[3] + x[4]  # Dependency between programs 5 or 6 and 3 or 4
prob += 2 <= lpSum(x[j] for j in [1, 2, 7, 8, 9, 10]) <= 4  # Minimum and maximum program selection

# Solve the problem
prob.solve()

# Print the optimal solution and the selected programs
print("Optimal Solution:")
print("Objective Value (Overall long-term profit):", value(prob.objective))
print("Selected Programs:")
for j in range(1, 11):
    if value(x[j]) == 1:
        print("Program", j)
```

Ορίσαμε τις παραμέτρους: μακροπρόθεσμο κέρδος (P), απαιτούμενη επένδυση (C) και συνολική διαθέσιμη επένδυση (Q) για κάθε πρόγραμμα.

Δημιουργήσαμε το αντικείμενο του προβλήματος LP.

Δημιουργήσαμε δυαδικές μεταβλητές απόφασης για κάθε πρόγραμμα.

Ορίσαμε την αντικειμενική συνάρτηση για τη μεγιστοποίηση του συνολικού μακροπρόθεσμου κέρδους.

Προσθέσαμε περιορισμούς: συνολική επένδυση, αμοιβαία αποκλειστικότητα, εξάρτηση και επιλογή ελάχιστου/μέγιστου προγράμματος.

Έχουμε το παρακάτω αποτέλεσμα:

Όπως παρατηρούμε επιλέγονται τα προγράμματα 1,2,8,9 με συνολικό κόστος επένδυσης 1000€.

```
Optimal Solution:
Objective Value (Overall long-term profit): 2000.0
Selected Programs:
Program 1
Program 2
Program 8
Program 9
Selected Programs and Total Cost:
Program 1 - Total Cost: 50
Program 2 - Total Cost: 100
Program 8 - Total Cost: 400
Program 9 - Total Cost: 450
Total Cost of Selected Programs: 1000
```