2025

Γραμμική & Συνδυαστική Βελτιστοποίηση

Εργασία #1

ΓΕΡΟΝΤΑΣ ΝΙΚΟΛΑΟΣ
1092813

Πίνακας περιεχομένων

Παράρτημα Πηγαίος Κώδικας:

Άσκηση 1

Θεωρήστε το πρόβλημα γραμμικού προγραμματισμού:

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

όταν

$$\begin{aligned} (\Pi 1) \quad & 6x_1 + 3x_2 \geq 12 \\ (\Pi 2) \quad & 4x_1 + 8x_2 \geq 16 \\ (\Pi 3) \quad & 6x_1 + 5x_2 \leq 30 \\ (\Pi 4) \quad & 6x_1 + 7x_2 \leq 36 \\ & x_1, x_2 \geq 0 \end{aligned}$$

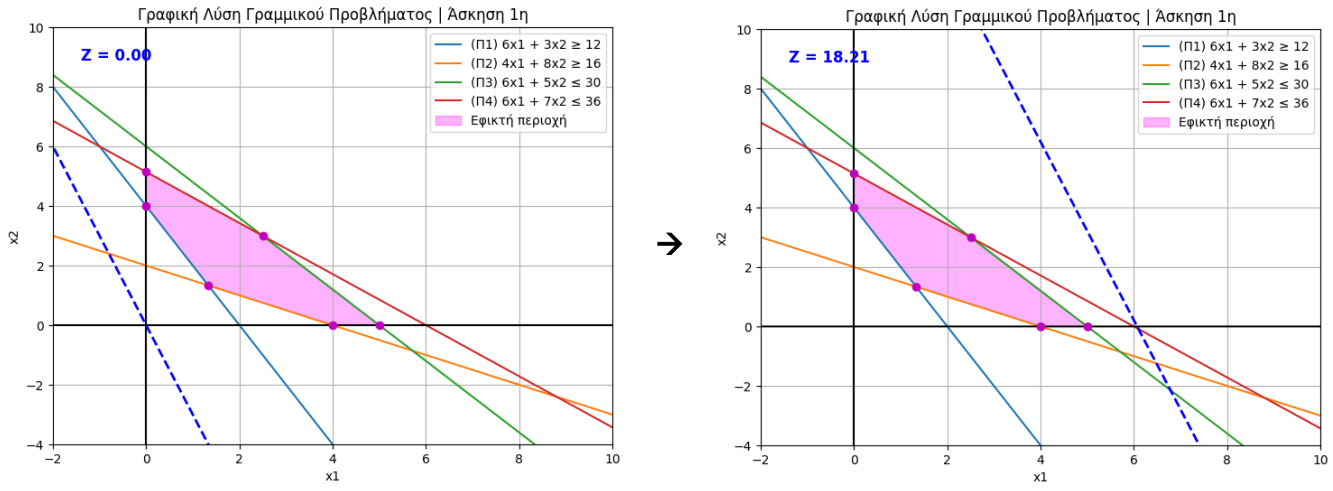
(α) Να παραστήσετε γραφικά την εφικτή περιοχή του προβλήματος καθώς και όλες τις κορυφές της. Περιγράψτε τη μορφή της εφικτής περιοχής. Με γραφικό τρόπο βρείτε τη βέλτιστη κορυφή του προβλήματος, εάν υπάρχει.

(β) Αν η αντικειμενική συνάρτηση του παραπάνω προβλήματος είναι $\min Z = x_1 + c_2 x_2$, ποιο είναι το εύρος τιμών που θα μπορούσε να πάρει το c_2 έτσι ώστε η βέλτιστη λύση να βρίσκεται στην τομή των ευθειών που ορίζουν οι περιορισμοί Π1 και Π2;

(γ) Αν στο παραπάνω πρόβλημα μεγιστοποίησης η αντικειμενική συνάρτηση ήταν $Z = c_1 x_1 + c_2 x_2$ βρείτε τις σχετικές τιμές των c_1 και c_2 έτσι ώστε η βέλτιστη λύση να βρίσκεται στην τομή των ευθειών που ορίζουν οι περιορισμοί Π3 και Π4.

Λύση:

α)



Λόγω του animation, παρατίθεται και σχετικό βίντεο με την εκτέλεση του κώδικα!
YT - Video - animation #1

```
Terminal:

Κορυφές [εφικτές λύσεις]:
  (1.3333333333333333, 1.3333333333333333) -> Z = 5.333333333333333
  (0.0, 4.0) -> Z = 4.0
  (4.0, 0.0) -> Z = 12.0
  (2.5, 3.0) -> Z = 10.5
  (5.0, 0.0) -> Z = 15.0
  (0.0, 5.142857142857143) -> Z = 5.142857142857143

Max Z = 14.98 - Καθαρά γραφική επίλυση [Animation]

Τερματισμός animation...
```

Από την παραπάνω γραφική, παρατηρούμε ότι η εφικτή περιοχή είναι ένα κυρτό πολύγωνο που περικλείεται από τις γραμμές που σχηματίζουν οι περιορισμοί: Π1, Π2 ,Π3 ,Π4 και οι άξονες (περιορισμοί προσήμου).

Παράλληλα, ο κώδικας βρίσκει γραφικά ότι η βέλτιστη κορυφή του προβλήματος είναι το σημείο (5, 0) με Z = 15!

Η μικρή απόκλιση στην τιμή του Max Z (14.98 αντί για 15) οφείλεται στην αριθμητική προσέγγιση που χρησιμοποιείται στο animation!

β)

Με αντικειμενική συνάρτηση τώρα ποια την σχέση:

$$\boxed{\min Z = x_1 + c_2 \cdot x_2},$$

απαιτούμε:

$$\mathbf{Z\left(\frac{4}{3},\frac{4}{3}\right) < Z(οποιασδήποτε\ άλλης\ κορυφής)}$$

δηλαδή, η τιμή της αντικειμενικής συνάρτησης στο σημείο τομής των Π1 και Π2 να είναι η μικρότερη μεταξύ όλων των εφικτών κορυφών!

Οπότε, πρέπει:

$$Z\left(\frac{4}{3},\frac{4}{3}\right) < Z(0,4) \Leftrightarrow \frac{4}{3} + c_2 \cdot \frac{4}{3} < 0 + c_2 \cdot 4 \Leftrightarrow \boxed{c_2 > \frac{1}{2}}$$

$$Z\left(\frac{4}{3},\frac{4}{3}\right) < Z(4,0) \Leftrightarrow \frac{4}{3} + c_2 \cdot \frac{4}{3} < 4 + c_2 \cdot 0 \Leftrightarrow \boxed{c_2 < 2}$$

$$Z\left(\frac{4}{3},\frac{4}{3}\right) < Z(2.5,3) \Leftrightarrow \frac{4}{3} + c_2 \cdot \frac{4}{3} < 2.5 + c_2 \cdot 3 \Leftrightarrow \boxed{c_2 > -0.7}$$

$$Z\left(\frac{4}{3},\frac{4}{3}\right) < Z(5,0) \Leftrightarrow \frac{4}{3} + c_2 \cdot \frac{4}{3} < 5 + c_2 \cdot 0 \Leftrightarrow \boxed{c_2 < \frac{11}{4}}$$

$$Z\left(\frac{4}{3},\frac{4}{3}\right) < Z(0,5.143) \Leftrightarrow \frac{4}{3} + c_2 \cdot \frac{4}{3} < 0 + c_2 \cdot 5.143 \Leftrightarrow \boxed{c_2 \gtrsim 0.35}$$

Επομένως:

$$\boxed{\mathbf{c_2 \in (0.5, 2)}}$$

γ)

Ακολουθώντας παρόμοια λογική με εκείνη του ερωτήματος (β), εξετάζουμε πλέον την περίπτωση όπου η αντικειμενική συνάρτηση έχει τη μορφή:

$$\boxed{\max Z = c_1 \cdot x_1 + c_2 \cdot x_2},$$

απαιτούμε:

$$\mathbf{Z(2.5, 3) > Z(οποιασδήποτε\ άλλης\ κορυφής)}$$

Άρα:

$$Z(2.5, 3) > Z\left(\frac{4}{3}, \frac{4}{3}\right) \Leftrightarrow c_1 \cdot 2.5 + c_2 \cdot 3 > c_1 \cdot \frac{4}{3} + c_2 \cdot \frac{4}{3} \Leftrightarrow \boxed{c_1 \cdot \frac{7}{6} > c_2 \cdot \left(-\frac{5}{3}\right)}$$

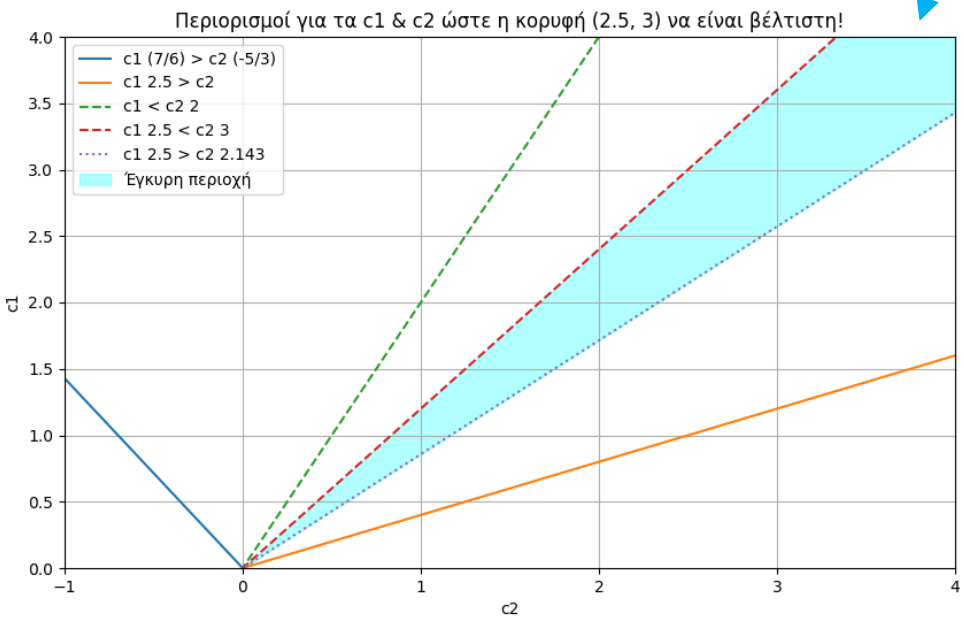
$$Z(2.5, 3) > Z(0, 4) \Leftrightarrow c_1 \cdot 2.5 + c_2 \cdot 3 > c_1 \cdot 0 + c_2 \cdot 4 \Leftrightarrow \boxed{c_1 \cdot 2.5 > c_2}$$

$$Z(2.5, 3) > Z(4, 0) \Leftrightarrow c_1 \cdot 2.5 + c_2 \cdot 3 > c_1 \cdot 4 + c_2 \cdot 0 \Leftrightarrow \boxed{c_1 < c_2 \cdot 2}$$

$$\mathbf{Z(2.5, 3) > Z(5, 0) \Leftrightarrow c_1 \cdot 2.5 + c_2 \cdot 3 > c_1 \cdot 5 + c_2 \cdot 0 \Leftrightarrow \boxed{c_1 \cdot 2.5 < c_2 \cdot 3}}$$

$$\mathbf{Z(2.5, 3) > Z(0, 5.143) \Leftrightarrow c_1 \cdot 2.5 + c_2 \cdot 3 > c_1 \cdot 0 + c_2 \cdot 5.143 \Leftrightarrow \boxed{c_1 \cdot 2.5 > c_2 \cdot 2.143}}$$

Επομένως:



Δηλαδή:

$$c_2 \cdot 2.143 < c_1 \cdot 2.5 < c_2 \cdot 3 \Leftrightarrow \boxed{\mathbf{0.8572 < \frac{c_1}{c_2} < 1.2}}$$

Κώδικας 1ης Άσκησης

Άσκηση 2

Ένας γιατρός σχεδιάζει το θεραπευτικό σχήμα ακτινοβολιών που πρόκειται να εφαρμόσει σε ασθενή με όγκο στην ουροδόχο κύστη. Ο γιατρός και οι συνεργάτες του καθορίζουν αρχικά τις δέσμες ακτινοβολίας που θα εφαρμόσουν στον ασθενή (δηλαδή το ακριβές σημείο του σώματος που θα ακτινοβοληθεί και τη γωνία με την οποία θα εφαρμοστεί η ακτινοβολία) έτσι ώστε να επηρεαστούν όσο το δυνατόν λιγότερα όργανα και στη συνέχεια θα πρέπει να ορίσει την ένταση τους (σε kilorads). Η συνολική ακτινοβολία που τελικά απορροφά η κάθε περιοχή του σώματος του ασθενούς είναι το άθροισμα της ακτινοβολίας που απορροφά από την κάθε ξεχωριστή δέσμη. Ο παρακάτω πίνακας δίνει τα ποσοστά ακτινοβολίας που κατά μέσο όρο εκτιμάται ότι απορροφούν οι διαφορετικού τύπου περιοχές του σώματος στην πάσχουσα περιοχή του συγκεκριμένου ασθενούς για τις 2 δέσμες ακτινοβολίας που ο γιατρός επέλεξε.

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

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

(β) Λύστε το πρόβλημα με γραφικό τρόπο.

Λύση:

α)

Έστω $\mathbf{x_i}$ **= η ένταση της δέσμης ακτινοβολίας i (σε kilorads)**, που πρέπει να οριστεί από τον γιατρό και τους συνεργάτες του. *Μεταβλητές απόφασης*

Έτσι, βάση της εκφώνησης και του πίνακα που μας δίνεται, προκύπτει:

$$\mathbf{max}\{-(\mathbf{0.4}\cdot\mathbf{x_1}+\mathbf{0.5}\cdot\mathbf{x_2})\}$$ *Αντικειμενική συνάρτηση*

Στόχος μας είναι να περιοριστεί η απορρόφηση ραδιενέργειας από την υγιή περιοχή!

όταν:

Ευαίσθητοι ιστοί $\quad \mathbf{0.3}\cdot\mathbf{x_1}+\mathbf{0.1}\cdot\mathbf{x_2}\leq\mathbf{2.7}$

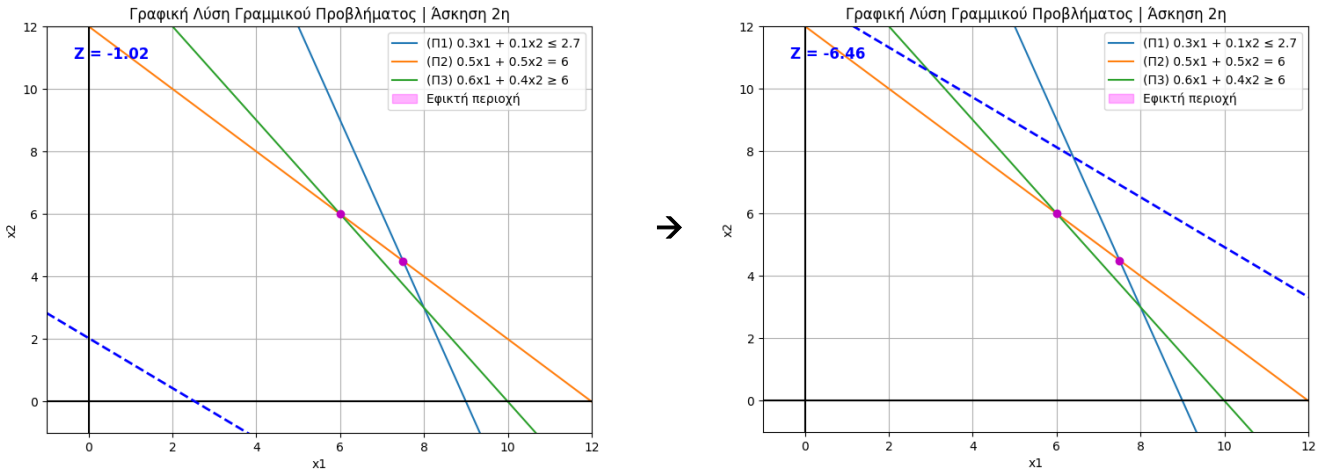
Περιοχή όγκου $\quad \mathbf{0.5}\cdot\mathbf{x_1}+\mathbf{0.5}\cdot\mathbf{x_2}=\mathbf{6}$

Κέντρο μάζας όγκου $\quad \mathbf{0.6}\cdot\mathbf{x_1}+\mathbf{0.4}\cdot\mathbf{x_2}\geq\mathbf{6}$

Περιορισμοί

$$\mathbf{x_1},\mathbf{x_2}\geq\mathbf{0}$$ *Περιορισμοί προσήμου*

β)



✓ **Παρατηρούμε ότι η εφικτή περιοχή περιορίζεται σε ένα ευθύγραμμο τμήμα!**

Λόγω του animation, παρατίθεται και σχετικό βίντεο με την εκτέλεση του κώδικα!
YT - Video - animation #2

Terminal:

```
Κορυφές [εφικτές λύσεις]:
  (7.5, 4.5) -> Z = -5.25
  (6.0, 6.0) -> Z = -5.4

Max Z = -5.24 - Καθαρά γραφική επίλυση [Animation]

Τερματισμός animation...
```

Ο κώδικας βρίσκει γραφικά ότι η βέλτιστη κορυφή του προβλήματος είναι το σημείο (7.5, 4.5) με Z = 5.25!

Η μικρή απόκλιση στην τιμή του Max Z (-5.24 αντί για -5.25) οφείλεται στην αριθμητική προσέγγιση που χρησιμοποιείται στο animation!

Κώδικας 2ης Άσκησης

Άσκηση 3

Μία επιχείρηση παράγει 3 τύπους ζωοτροφής για 3 διαφορετικά είδη ζώων (αγελάδες-τύπου I, πρόβατα-τύπου II και κοτόπουλα-τύπου III). Για την παραγωγή τους χρησιμοποιεί ως πρώτες ύλες: καλαμπόκι, ασβεστόλιθο, σόγια, και ιχθυάλευρα. Οι πρώτες ύλες περιέχουν βιταμίνες, πρωτεΐνες, ασβέστιο, και λίπος που είναι σημαντικά θρεπτικά συστατικά για τα ζώα. Οι ποσότητες (σε τυπικές μονάδες μέτρησης) των θρεπτικών συστατικών ανά kg πρώτης ύλης δίνεται στον Πίνακα 1.

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

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

Η επιχείρηση έχει αναλάβει την παραγωγή 12, 8, και 9 τόνων ζωοτροφής αντίστοιχα για τους 3 τύπους. Έχει όμως στη διάθεσή της περιορισμένες ποσότητες πρώτων υλών και δεν μπορεί να αναπληρώσει άμεσα καμία από αυτές. Συγκεκριμένα διαθέτει 9 τόνους καλαμπόκι, 12 τόνους ασβεστόλιθο, 5 τόνους σόγια, και 6 τόνους ιχθυάλευρα. Η τιμή ανά kg για τις πρώτες ύλες είναι 0.20, 0.12, 0.24, και 0.12, αντίστοιχα. Επιπλέον, η επιχείρηση θα πρέπει να λάβει υπόψη τις προδιαγραφές όπως ορίζονται για τις ζωοτροφές που θα παράγει και συνοψίζονται στον Πίνακα 2.

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

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

Μοντελοποιήστε το συγκεκριμένο πρόβλημα ως πρόβλημα γραμμικού προγραμματισμού με στόχο την ελαχιστοποίηση του κόστους παραγωγής για την επιχείρηση. Ορίστε και περιγράψτε με ακρίβεια τις μεταβλητές απόφασης και διαμορφώστε κατάλληλα την αντικειμενική συνάρτηση και όλους τους περιορισμούς του προβλήματος. (**Σημ.** Η άσκηση δεν ζητάει τη λύση του προβλήματος, μόνο τη μοντελοποίησή του.)

Λύση:

🖬 Μεταβλητές απόφασης:

Έστω $\mathbf{x_{ij}}$ **= η ποσότητα της πρώτης ύλης i (σε Kg) που χρησιμοποιείται για την παρασκευή της ζωοτροφής τύπου j.**

✍ Αντικειμενική συνάρτηση:

Έτσι, βάση της εκφώνησης, προκύπτει:

$$\min\left\{\sum_{j=1}^{3}\left(0.2\cdot x_{1,j}+0.12\cdot x_{2,j}+0.24\cdot x_{3,j}+0.12\cdot x_{4,j}\right)\right\}$$

Στόχος μας είναι η ελαχιστοποίηση του κόστους παραγωγής για την επιχείρηση!

✓ Περιορισμοί:

- Οφείλει να παράγει:

$$x_{1,1}+x_{2,1}+x_{3,1}+x_{4,1}=12000\ \{Kg\}$$
$$x_{1,2}+x_{2,2}+x_{3,2}+x_{4,2}=8000\ \{Kg\}$$
$$x_{1,3}+x_{2,3}+x_{3,3}+x_{4,3}=9000\ \{Kg\}$$

- με τις περιορισμένες ποσότητες πρώτων υλών:

$$x_{1,1}+x_{1,2}+x_{1,3}\le 9000\ \{Kg\}$$
$$x_{2,1}+x_{2,2}+x_{2,3}\le 12000\ \{Kg\}$$
$$x_{3,1}+x_{3,2}+x_{3,3}\le 5000\ \{Kg\}$$
$$x_{4,1}+x_{4,2}+x_{4,3}\le 6000\ \{Kg\}$$

- και τις εξής απαιτήσεις σε θρεπτικά συστατικά:

$$6\cdot 12000 \le 8\cdot x_{1,1}+6\cdot x_{2,1}+10\cdot x_{3,1}+4\cdot x_{4,1}$$
$$6\cdot 12000 \le 10\cdot x_{1,1}+5\cdot x_{2,1}+12\cdot x_{3,1}+8\cdot x_{4,1}$$
$$7\cdot 12000 \le 6\cdot x_{1,1}+10\cdot x_{2,1}+6\cdot x_{3,1}+6\cdot x_{4,1}$$
$$4\cdot 12000 \le 8\cdot x_{1,1}+6\cdot x_{2,1}+6\cdot x_{3,1}+9\cdot x_{4,1} \le 8\cdot 12000$$

&

$$6\cdot 8000 \le 8\cdot x_{1,2}+6\cdot x_{2,2}+10\cdot x_{3,2}+4\cdot x_{4,2}$$
$$6\cdot 8000 \le 10\cdot x_{1,2}+5\cdot x_{2,2}+12\cdot x_{3,2}+8\cdot x_{4,2}$$
$$6\cdot 8000 \le 6\cdot x_{1,2}+10\cdot x_{2,2}+6\cdot x_{3,2}+6\cdot x_{4,2}$$
$$4\cdot 8000 \le 8\cdot x_{1,2}+6\cdot x_{2,2}+6\cdot x_{3,2}+9\cdot x_{4,2} \le 6\cdot 8000$$

&

$$4\cdot 9000 \le 8\cdot x_{1,3}+6\cdot x_{2,3}+10\cdot x_{3,3}+4\cdot x_{4,3} \le 6\cdot 9000$$
$$6\cdot 9000 \le 10\cdot x_{1,3}+5\cdot x_{2,3}+12\cdot x_{3,3}+8\cdot x_{4,3}$$
$$6\cdot 9000 \le 6\cdot x_{1,3}+10\cdot x_{2,3}+6\cdot x_{3,3}+6\cdot x_{4,3}$$
$$4\cdot 9000 \le 8\cdot x_{1,3}+6\cdot x_{2,3}+6\cdot x_{3,3}+9\cdot x_{4,3} \le 5\cdot 9000$$

± Περιορισμοί προσήμου:

$$x_{i,j}\ge 0,\ \forall i,j$$

Άσκηση 4

Εξετάστε ως προς την κυρτότητα τα σύνολα:

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

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

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

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

Για κάθε σύνολο είτε αποδείξτε ότι είναι κυρτό είτε δώστε αντιπαράδειγμα για να δείξετε ότι δεν είναι κυρτό.

Λύση:

α)

Αντιπαράδειγμα

Έστω:

$$P_1 = (-4, 0)\ \&\ P_2 = (2, 0)$$

2 σημεία του δοθέντος συνόλου, καθώς:

$$P_{1_{x_1}}^2 + P_{1_{x_2}}^2 = (-4)^2 + 0^2 = 16 > 3$$

$$P_{2_{x_1}}^2 + P_{2_{x_2}}^2 = 2^2 + 0^2 = 4 > 3$$

Για λ = 0.5, δηλαδή το μέσο του ευθύγραμμου τμήματος που ενώνει τα 2 σημεία, παρατηρούμε ότι ισχύει:

$$M = 0.5 \cdot P_1 + (1 - 0.5) \cdot P_2 = 0.5 \cdot (-4, 0) + 0.5 \cdot (2, 0) = (-1, 0)$$

και

$$\mathbf{M_{x_1}^2 + M_{x_2}^2 = (-1)^2 + 0^2 = 1 < 3}$$

(δηλαδή, δεν ανήκει στο δοθέν σύνολο!)

β)

✓ Έχουμε το σύνολο:

$$S = \{\, \vec{x} \in \mathbb{R}^3 \mid x_1 + 2x_2 \leq 1, x_1 - 2x_3 \leq 2 \,\}$$

Έστω:

$$\vec{a}, \vec{b} \epsilon\, S$$

δηλαδή:

$$\boxed{\mathbf{a_1 + 2a_2 \leq 1, a_1 - 2a_3 \leq 2}}\ \&\ \boxed{\mathbf{b_1 + 2b_2 \leq 1, b_1 - 2b_3 \leq 2}}\ (\mathbf{1})$$

Ισχύει:

$$\vec{z} = \lambda\vec{a} + (1 - \lambda)\vec{b} = \begin{bmatrix} \lambda a_1 + (1 - \lambda) b_1 \\ \lambda a_2 + (1 - \lambda) b_2 \\ \lambda a_3 + (1 - \lambda) b_3 \end{bmatrix}$$

Έτσι, <u>υπολογίζουμε ότι:</u>

$$z_1 + 2z_2 = [\lambda a_1 + (1-\lambda)b_1] + 2[\lambda a_2 + (1-\lambda)b_2] = \lambda(a_1 + 2a_2) + (1-\lambda)(b_1 + 2b_2) \overset{(1)}{\Rightarrow}$$
$$\overset{(1)}{\Rightarrow} z_1 + 2z_2 \leq \lambda + (1-\lambda) \Leftrightarrow \boxed{\mathbf{z_1 + 2z_2 \leq 1}}$$

και

$$z_1 - 2z_3 = [\lambda a_1 + (1-\lambda)b_1] - 2[\lambda a_3 + (1-\lambda)b_3] = \lambda(a_1 - 2a_3) + (1-\lambda)(b_1 - 2b_3) \overset{(1)}{\Rightarrow}$$
$$\overset{(1)}{\Rightarrow} z_1 - 2z_3 \leq \lambda \cdot 2 + (1-\lambda) \cdot 2 \Leftrightarrow \boxed{\mathbf{z_1 - 2z_3 \leq 2}}$$

<u>Επομένως:</u>

$$\boxed{\vec{\mathbf{z}} \in \mathbf{S}}$$

γ)

✓ Έχουμε το σύνολο:

$$S = \{ \vec{x} \in \mathbb{R}^3 \mid x_2 \geq x_1^2, x_1 + 2x_2 + x_3 \leq 4 \}$$

<u>Έστω:</u>

$$\vec{a}, \vec{b} \epsilon\, S$$

δηλαδή:

$$\boxed{\mathbf{a_2 \geq a_1^2, a_1 + 2a_2 + a_3 \leq 4}} \,\&\, \boxed{\mathbf{b_2 \geq b_1^2, b_1 + 2b_2 + b_3 \leq 4}} \,(\mathbf{1})$$

<u>Ισχύει:</u>

$$\vec{z} = \lambda\vec{a} + (1-\lambda)\vec{b} = \begin{bmatrix} \lambda a_1 + (1-\lambda)b_1 \\ \lambda a_2 + (1-\lambda)b_2 \\ \lambda a_3 + (1-\lambda)b_3 \end{bmatrix}$$

Έτσι, <u>υπολογίζουμε ότι:</u>

$$z_2 - z_1^2 = [\lambda a_2 + (1-\lambda)b_2] - [\lambda a_1 + (1-\lambda)b_1]^2 =$$
$$= \lambda a_2 + (1-\lambda)b_2 - [(\lambda a_1)^2 + 2 \cdot \lambda a_1 \cdot (1-\lambda)b_1 + (1-\lambda)^2 b_1^2] \overset{(1)}{\Rightarrow}$$
$$\overset{(1)}{\Rightarrow} z_2 - z_1^2 \geq \lambda a_1^2 + (1-\lambda)b_1^2 - (\lambda a_1)^2 - 2 \cdot \lambda a_1 \cdot (1-\lambda)b_1 - (1-\lambda)^2 b_1^2 =$$
$$= \lambda(1-\lambda)a_1^2 + \big(1 - (1-\lambda)\big)(1-\lambda)b_1^2 - 2 \cdot \lambda a_1 \cdot (1-\lambda)b_1 =$$
$$= \lambda(1-\lambda)(a_1^2 + b_1^2 - 2a_1 b_1) \Leftrightarrow$$
$$\Leftrightarrow \boxed{\mathbf{z_2 - z_1^2 \geq \lambda(1-\lambda)(a_1 - b_1)^2 > 0}}$$

και

$$z_1 + 2z_2 + z_3 = [\lambda a_1 + (1-\lambda)b_1] + 2[\lambda a_2 + (1-\lambda)b_2] + [\lambda a_3 + (1-\lambda)b_3]$$
$$= \lambda(a_1 + 2a_2 + a_3) + (1-\lambda)(b_1 + 2b_2 + b_3) \overset{(1)}{\Rightarrow}$$
$$\overset{(1)}{\Rightarrow} z_1 + 2z_2 + z_3 \leq \lambda \cdot 4 + (1-\lambda) \cdot 4 \Leftrightarrow \boxed{\mathbf{z_1 + 2z_2 + z_3 \leq 4}}$$

<u>Επομένως:</u>

$$\boxed{\vec{\mathbf{z}} \in \mathbf{S}}$$

δ)

Αντιπαράδειγμα

Έστω:

$$\mathrm{P}_1 = (0, -1, 1) \;\&\; \mathrm{P}_2 = (0, 1, 1)$$

2 σημεία του δοθέντος συνόλου, καθώς:

$$\mathrm{P}_{1_{x_3}} = \left|\mathrm{P}_{1_{x_2}}\right| = 1$$

$$\mathrm{P}_{2_{x_3}} = \left|\mathrm{P}_{2_{x_2}}\right| = 1$$

Για λ = 0.5, δηλαδή το μέσο του ευθύγραμμου τμήματος που ενώνει τα 2 σημεία, παρατηρούμε ότι ισχύει:

$$\mathrm{M} = 0.5 \cdot \mathrm{P}_1 + (1 - 0.5) \cdot \mathrm{P}_2 = 0.5 \cdot (0, -1, 1) + 0.5 \cdot (0, 1, 1) = (0, 0, 1)$$

και

$$\mathbf{M_{x_3} = 1 \neq \left|M_{x_2}\right| = 0}$$

(δηλαδή, δεν ανήκει στο δοθέν σύνολο!)

Άσκηση 5

Δίνεται το πρόβλημα γραμμικού προγραμματισμού:

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

όταν

$$\begin{aligned} x_1 + x_2 &\geq 10 \\ x_2 + x_3 &\geq 15 \\ x_1 + x_3 &\geq 12 \\ 20x_1 + 10x_2 + 15x_3 &\leq 300 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

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

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

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

Λύση:

α)

Διαπιστώνουμε ότι καμία από τις εφικτές κορυφές δεν είναι εκφυλισμένη!

Terminal:

```
Κορυφές [Εφικτές: + | Εκφυλισμένες: Δ | Μη εφικτές: -]:
 + (3.5, 6.5, 8.5) -> Z = -94.5
 + (5.0, 5.0, 10.0) -> Z = -105.0
 - (0.0, 10.0, 5.0) -> Z = -70.0
 - (10.0, 0.0, 15.0) -> Z = -140.0
 - (-5.0, 15.0, 0.0) -> Z = -35.0
 - (-4.0, 14.0, 16.0) -> Z = -102.0
 + (0.0, 10.0, 12.0) -> Z = -98.0
 - (10.0, 0.0, 2.0) -> Z = -88.0
 - (12.0, -2.0, 0.0) -> Z = -86.0
 + (0.0, 10.0, 13.333333333333334) -> Z = -103.33333333333334
 - (10.0, 0.0, 6.666666666666667) -> Z = -106.66666666666667
 - (20.0, -10.0, 0.0) -> Z = -110.0
 - (0.0, 10.0, 0.0) -> Z = -50.0
 - (10.0, 0.0, 0.0) -> Z = -80.0
 + (6.0, 9.0, 6.0) -> Z = -117.0
 - (0.0, 3.0, 12.0) -> Z = -63.0
 - (-3.0, 0.0, 15.0) -> Z = -36.0
 - (12.0, 15.0, 0.0) -> Z = -171.0
 - (0.0, -15.0, 30.0) -> Z = -45.0
 - (3.75, 0.0, 15.0) -> Z = -90.0
 - (7.5, 15.0, 0.0) -> Z = -135.0
 - (0.0, 0.0, 15.0) -> Z = -60.0
 - (0.0, 15.0, 0.0) -> Z = -75.0
 + (0.0, 12.0, 12.0) -> Z = -108.0
 - (24.0, 0.0, -12.0) -> Z = -144.0
 - (12.0, 6.0, 0.0) -> Z = -126.0
 - (0.0, 0.0, 12.0) -> Z = -48.0
 - (12.0, 0.0, 0.0) -> Z = -96.0
 - (0.0, 0.0, 20.0) -> Z = -80.0
 - (0.0, 30.0, 0.0) -> Z = -150.0
 - (15.0, 0.0, 0.0) -> Z = -120.0
 - (0.0, 0.0, 0.0) -> Z = -0.0
```

β)

Terminal:

```
+ Βάση: {x1, x2, x3, x4} =>   x1=   6.000, x2=   9.000, x3=   6.000, x4=   5.000 | Z = -117.000
- Βάση: {x1, x2, x3, x5} =>   x1=  -4.000, x2=  14.000, x3=  16.000, x5=  15.000 | Z = -102.000
+ Βάση: {x1, x2, x3, x6} =>   x1=   5.000, x2=   5.000, x3=  10.000, x6=   3.000 | Z = -105.000
+ Βάση: {x1, x2, x3, x7} =>   x1=   3.500, x2=   6.500, x3=   8.500, x7=  37.500 | Z =  -94.500
- Βάση: {x1, x2, x4, x5} =>   x1=  12.000, x2=   6.000, x4=   8.000, x5=  -9.000 | Z = -126.000
- Βάση: {x1, x2, x4, x6} =>   x1=   7.500, x2=  15.000, x4=  12.500, x6=  -4.500 | Z = -135.000
- Βάση: {x1, x2, x4, x7} =>   x1=  12.000, x2=  15.000, x4=  17.000, x7= -90.000 | Z = -171.000
- Βάση: {x1, x2, x5, x6} =>   x1=  20.000, x2= -10.000, x5= -25.000, x6=   8.000 | Z = -110.000
- Βάση: {x1, x2, x5, x7} =>   x1=  12.000, x2=  -2.000, x5= -17.000, x7=  80.000 | Z =  -86.000
- Βάση: {x1, x2, x6, x7} =>   x1=  -5.000, x2=  15.000, x6= -17.000, x7= 250.000 | Z =  -35.000
- Βάση: {x1, x3, x4, x5} =>   x1=  24.000, x3= -12.000, x4=  14.000, x5= -27.000 | Z = -144.000
- Βάση: {x1, x3, x4, x6} =>   x1=   3.750, x3=  15.000, x4=  -6.250, x6=   6.750 | Z =  -90.000
- Βάση: {x1, x3, x4, x7} =>   x1=  -3.000, x3=  15.000, x4= -13.000, x7= 135.000 | Z =  -36.000
- Βάση: {x1, x3, x5, x6} =>   x1=  10.000, x3=   6.667, x5=  -8.333, x6=   4.667 | Z = -106.667
- Βάση: {x1, x3, x5, x7} =>   x1=  10.000, x3=   2.000, x5= -13.000, x7=  70.000 | Z =  -88.000
- Βάση: {x1, x3, x6, x7} =>   x1=  10.000, x3=  15.000, x6=  13.000, x7=-125.000 | Z = -140.000
- Βάση: {x1, x4, x5, x6} =>   x1=  15.000, x4=   5.000, x5= -15.000, x6=   3.000 | Z = -120.000
- Βάση: {x1, x4, x5, x7} =>   x1=  12.000, x4=   2.000, x5= -15.000, x7=  60.000 | Z =  -96.000
- Βάση: {x1, x5, x6, x7} =>   x1=  10.000, x5= -15.000, x6=  -2.000, x7= 100.000 | Z =  -80.000
+ Βάση: {x2, x3, x4, x5} =>   x2=  12.000, x3=  12.000, x4=   2.000, x5=   9.000 | Z = -108.000
- Βάση: {x2, x3, x4, x6} =>   x2= -15.000, x3=  30.000, x4= -25.000, x6=  18.000 | Z =  -45.000
- Βάση: {x2, x3, x4, x7} =>   x2=   3.000, x3=  12.000, x4=  -7.000, x7=  90.000 | Z =  -63.000
+ Βάση: {x2, x3, x5, x6} =>   x2=  10.000, x3=  13.333, x5=   8.333, x6=   1.333 | Z = -103.333
+ Βάση: {x2, x3, x5, x7} =>   x2=  10.000, x3=  12.000, x5=   7.000, x7=  20.000 | Z =  -98.000
- Βάση: {x2, x3, x6, x7} =>   x2=  10.000, x3=   5.000, x6=  -7.000, x7= 125.000 | Z =  -70.000
- Βάση: {x2, x4, x5, x6} =>   x2=  30.000, x4=  20.000, x5=  15.000, x6= -12.000 | Z = -150.000
- Βάση: {x2, x4, x6, x7} =>   x2=  15.000, x4=   5.000, x6= -12.000, x7= 150.000 | Z =  -75.000
- Βάση: {x2, x5, x6, x7} =>   x2=  10.000, x5=  -5.000, x6= -12.000, x7= 200.000 | Z =  -50.000
- Βάση: {x3, x4, x5, x6} =>   x3=  20.000, x4= -10.000, x5=   5.000, x6=   8.000 | Z =  -80.000
- Βάση: {x3, x4, x5, x7} =>   x3=  12.000, x4= -10.000, x5=  -3.000, x7= 120.000 | Z =  -48.000
- Βάση: {x3, x4, x6, x7} =>   x3=  15.000, x4= -10.000, x6=   3.000, x7=  75.000 | Z =  -60.000
- Βάση: {x4, x5, x6, x7} =>   x4= -10.000, x5= -15.000, x6= -12.000, x7= 300.000 | Z =    0.000

Σύνολο βασικών [εφικτών και μη] λύσεων: 32
```

Όπως και στην περίπτωση/ερώτημα (α), δεν παρατηρείται καμία εκφυλισμένη βασική λύση, δηλαδή σε όλες τις εφικτές βασικές λύσεις οι βασικές μεταβλητές είναι αυστηρά θετικές!

γ)

Οι βασικές εφικτές λύσεις που υπολογίστηκαν στο (β) ερώτημα ταυτίζονται πλήρως με τις κορυφές που προέκυψαν στο (α). Συνεπώς, επιβεβαιώνεται ότι κάθε κορυφή του πολυτόπου αντιστοιχεί σε μία βασική λύση του συστήματος {*Σελ. 39 / 70 - 01. Εισαγωγή στον Γραμμικό Προγραμματισμό - Αλγόριθμος Simplex*}!

✓ Ανάμεσά τους, η βέλτιστη λύση/κορυφή είναι:

$$\boxed{(x_1, x_2, x_3) = (3.5, 6.5, 8.5), Z = 94.5}$$

η οποία αντιστοιχεί στη **βασική λύση με βάση: {x1, x2, x3, x7}**

Κώδικας 5ης Άσκησης

Άσκηση 6

Δίνεται το πρόβλημα γραμμικού προγραμματισμού:

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

όταν

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

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

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

Λύση:

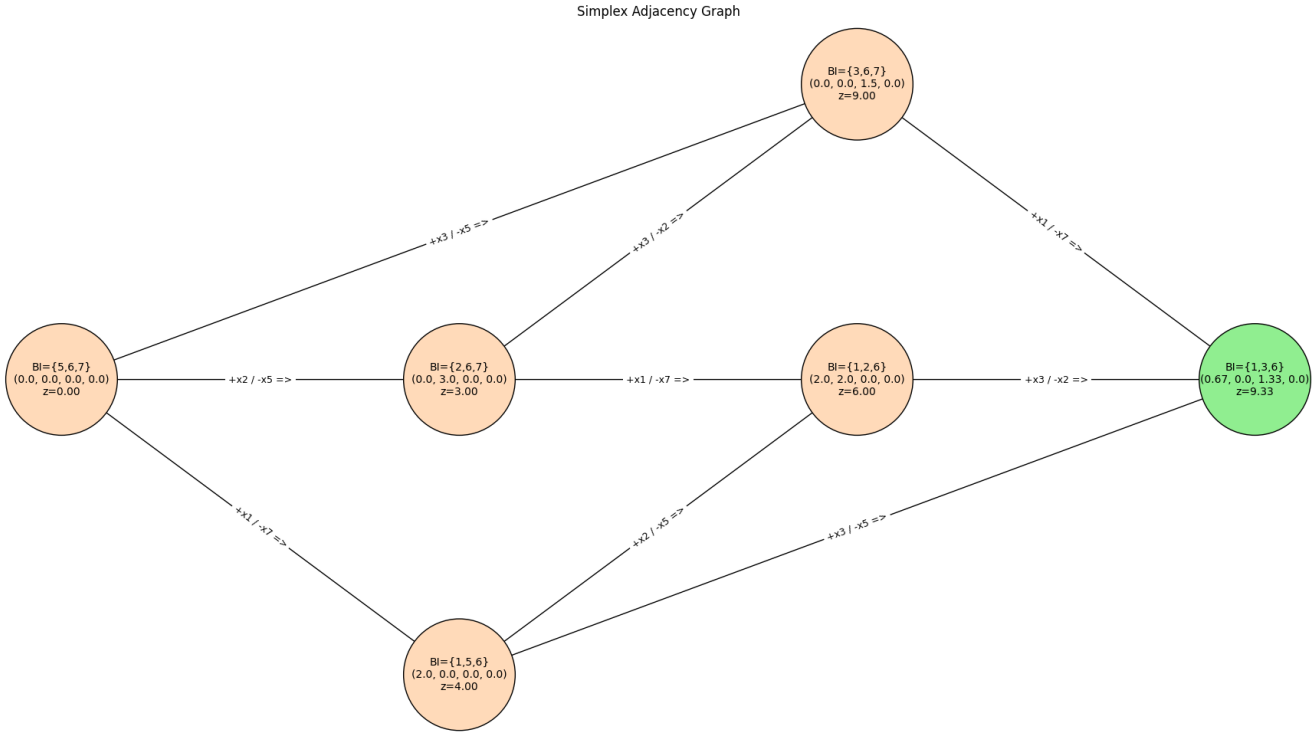
α)

Ακολουθεί η έξοδος του κώδικα στο terminal:

```
 Βήμα 0 |     x1 |     x2 |     x3 |     x4 |     x5 |     x6 |     x7 |      b
-------------------------------------------------------------------------------
     -Z |      2 |      1 |      6 |     -4 |      0 |      0 |      0 |      0
-------------------------------------------------------------------------------
     x5 |      1 |      2 |      4 |     -1 |      1 |      0 |      0 |      6
     x6 |      2 |      3 |     -1 |      1 |      0 |      1 |      0 |     12
     x7 |      1 |      0 |      1 |      1 |      0 |      0 |      1 |      2

Επιλογή της 1ης μεταβλητής με Z > 0 ως εισερχόμενη:
z =     2 | x1  <-- Εισερχόμενη βασική μεταβλητή

Κριτήριο ελαχίστου λόγου:
---------------------------------------
Βασική     |   xB[i] |    d[i] |   Λόγος
---------------------------------------
x5         |       6 |       1 |       6
x6         |      12 |       2 |       6
x7         |       2 |       1 |       2  <-- ελάχιστος
Εξερχόμενη μεταβλητή: x7
```

```
 Βήμα 1 |     x1 |     x2 |     x3 |     x4 |     x5 |     x6 |     x7 |      b
-------------------------------------------------------------------------------
     -Z |      0 |      1 |      4 |     -6 |      0 |      0 |     -2 |     -4
-------------------------------------------------------------------------------
     x5 |      0 |      2 |      3 |     -2 |      1 |      0 |     -1 |      4
     x6 |      0 |      3 |     -3 |     -1 |      0 |      1 |     -2 |      8
     x1 |      1 |      0 |      1 |      1 |      0 |      0 |      1 |      2

Επιλογή της 1ης μεταβλητής με Z > 0 ως εισερχόμενη:
z =     0 | x1
z =     1 | x2  <-- Εισερχόμενη βασική μεταβλητή

Κριτήριο ελαχίστου λόγου:
---------------------------------------
Βασική     |   xB[i] |    d[i] |   Λόγος
---------------------------------------
x5         |       4 |       2 |       2  <-- ελάχιστος
x6         |       8 |       3 |     8/3
x1         |       2 |       0 |       -
Εξερχόμενη μεταβλητή: x5

 Βήμα 2 |     x1 |     x2 |     x3 |     x4 |     x5 |     x6 |     x7 |      b
-------------------------------------------------------------------------------
     -Z |      0 |      0 |    5/2 |     -5 |   -1/2 |      0 |   -3/2 |     -6
-------------------------------------------------------------------------------
     x2 |      0 |      1 |    3/2 |     -1 |    1/2 |      0 |   -1/2 |      2
     x6 |      0 |      0 |  -15/2 |      2 |   -3/2 |      1 |   -1/2 |      2
     x1 |      1 |      0 |      1 |      1 |      0 |      0 |      1 |      2

Επιλογή της 1ης μεταβλητής με Z > 0 ως εισερχόμενη:
z =     0 | x1
z =     0 | x2
z =   5/2 | x3  <-- Εισερχόμενη βασική μεταβλητή

Κριτήριο ελαχίστου λόγου:
---------------------------------------
Βασική     |   xB[i] |    d[i] |   Λόγος
---------------------------------------
x2         |       2 |     3/2 |     4/3  <-- ελάχιστος
x6         |       2 |   -15/2 |       -
x1         |       2 |       1 |       2
Εξερχόμενη μεταβλητή: x2

 Βήμα 3 |     x1 |     x2 |     x3 |     x4 |     x5 |     x6 |     x7 |      b
-------------------------------------------------------------------------------
     -Z |      0 |   -5/3 |      0 |  -10/3 |   -4/3 |      0 |   -2/3 |  -28/3
-------------------------------------------------------------------------------
     x3 |      0 |    2/3 |      1 |   -2/3 |    1/3 |      0 |   -1/3 |    4/3
     x6 |      0 |      5 |      0 |     -3 |      1 |      1 |     -3 |     12
     x1 |      1 |   -2/3 |      0 |    5/3 |   -1/3 |      0 |    4/3 |    2/3

Η λύση είναι βέλτιστη!

x1 = 2/3, x2 = 0, x3 = 4/3, x4 = 0, x5 = 0, x6 = 12, x7 = 0 με Z = 28/3
```

β)



Κώδικας 6ης Άσκησης

Κώδικας 1ης Άσκησης

```
# Άσκηση 1η
                                                                    lin_prog_code_HW1_1.py
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from sympy.solvers import solve
from sympy import Symbol
from sympy import lambdify
from itertools import combinations

def p1(x1: Symbol, x2: Symbol) -> float:
    return 6*x1 + 3*x2 - 12;

def p2(x1: Symbol, x2: Symbol) -> float:
    return 4*x1 + 8*x2 - 16;

def p3(x1: Symbol, x2: Symbol) -> float:
    return 6*x1 + 5*x2 - 30;

def p4(x1: Symbol, x2: Symbol) -> float:
    return 6*x1 + 7*x2 - 36;

# Αντικειμενική συνάρτηση [Z]
def Z(x1: Symbol, x2: Symbol) -> float:
    return 3*x1 + x2;

def main():
    (x1_var, x2_var) = (Symbol('x1'), Symbol('x2'))

    # Οι συναρτήσεις των περιορισμών μας
    constraint_functions = [
        p1(x1_var, x2_var), # ≥ 0
        p2(x1_var, x2_var), # ≥ 0
        p3(x1_var, x2_var), # ≤ 0
        p4(x1_var, x2_var), # ≤ 0
        x1_var, x2_var      # ≥ 0
    ]

    inequality_signs = ['>=', '>=', '<=', '<=', '>=', '>=']
    compiled_constraints = [
        (lambdify((x1_var, x2_var), expr), sign)
        for (expr, sign) in zip(constraint_functions, inequality_signs)
    ] # lambdify: Converts symbolic expressions into Python functions!
```

```
    def check_restrictions(x1_val, x2_val, epsilon = 1e-8):
        # epsilon tolerance for floating point comparisons!!!!!
        for (function, sign) in compiled_constraints:
            value = function(x1_val, x2_val) # Γυρνάει float

            if sign == '>=' and value < -epsilon:
                break;
            elif sign == '<=' and value > epsilon:
                break;
        else:
            return True;

        return False;

    # Συνδυασμοί, ανά 2 συναρτήσεων περιορισμών, για τις τομές/κορυφές
    constraint_foos_combo = list(combinations(constraint_functions, 2))

    # Υπολογισμός ΟΛΩΝ των σημείων τομής
    intersection_points = []
    for (cf1, cf2) in constraint_foos_combo:
        sol = solve((cf1, cf2), (x1_var, x2_var))
        if sol: # Ώστε να μην προκύψει IndexError!
            intersection_points.append((float(sol[x1_var]), float(sol[x2_var])))

    # Φιλτράρουμε τα σημεία που δεν ικανοποιούν όλους τους περιορισμούς
    feasible_points = []
    for (candidate_x1, candidate_x2) in intersection_points:
        if check_restrictions(candidate_x1, candidate_x2):
            feasible_points.append((candidate_x1, candidate_x2))

    # Βρίσκουμε το μέγιστο της Z συνάρτησης
    best_value = float('-inf')
    for (candidate_x1, candidate_x2) in feasible_points:
        val = Z(candidate_x1, candidate_x2)
        if val > best_value:
            best_value = val

    print('Κορυφές [εφικτές λύσεις]:')
    for (x1_val, x2_val) in feasible_points:
        print(f'  {(x1_val, x2_val)} -> Z = {Z(x1_val, x2_val)}')
    print() # Καλύτερη αισθητική

    # Γραφική παράσταση
    x_vals = np.linspace(-10, 10, 400)
    plt.figure(figsize = (8, 6))
```

```
    # Γραμμές περιορισμών
    graphic_representation = [
        ((12 - 6*x_vals) / 3, '(Π1) 6x1 + 3x2 ≥ 12'),
        ((16 - 4*x_vals) / 8, '(Π2) 4x1 + 8x2 ≥ 16'),
        ((30 - 6*x_vals) / 5, '(Π3) 6x1 + 5x2 ≤ 30'),
        ((36 - 6*x_vals) / 7, '(Π4) 6x1 + 7x2 ≤ 36')
    ]
    for (y_vals, label) in graphic_representation:
        plt.plot(x_vals, y_vals, label = label)
    plt.axhline(0, color = 'black')
    plt.axvline(0, color = 'black')

    # Εφικτά σημεία στην γραφική παράσταση
    for (x1_val, x2_val) in feasible_points:
        plt.plot(x1_val, x2_val, 'mo')

    # Περίπου σαν Graham Scan για να βρούμε την εφικτή περιοχή!
    feasible_np = np.array(feasible_points)
    center = np.mean(feasible_np, axis = 0)
    sorted_points = sorted(
        feasible_points,
        key = lambda point: np.arctan2(
            point[1] - center[1], point[0] - center[0]
        )
    )
    x_sorted = [point[0] for point in sorted_points]
    y_sorted = [point[1] for point in sorted_points]
    plt.fill(x_sorted, y_sorted, color = 'magenta',
             alpha = 0.3, label = 'Εφικτή περιοχή')

    plt.xlim(-2, 10)
    plt.ylim(-4, 10)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.legend()
    plt.title('Γραφική Λύση Γραμμικού Προβλήματος | Άσκηση 1η')
    plt.grid(True)
```

```
# ----- Animation setup -----

z_line, = plt.plot(
    [], [], linestyle = '--',
    color = 'blue', linewidth = 2
)
z_label = plt.text(
    0.05, 0.95, '', transform = plt.gca().transAxes,
    color = 'blue', fontweight = 'bold', fontsize = 12,
    verticalalignment = 'top'
)

total_frames = 140 # Το animation μέχρι να φτάσει το max Z!
total_animation_frames = total_frames + 30 # + 30 για να συνεχίσει λίγο ακόμα

# ----- Προϋπολογισμός με NumPy για smooth animation -----
t_vals = np.linspace(0, total_animation_frames / total_frames, total_animation_frames)
precomputed_z_vals = t_vals * best_value
precomputed_y_vals = precomputed_z_vals[:, None] - 3*x_vals[None, :]
precomputed_is_inside = np.array([
    any(check_restrictions(x, y_) for x, y_ in zip(x_vals, y))
    for y in precomputed_y_vals
])

was_inside = [False]
has_printed = [False]
def animate_z_line(frame):
    current_z = precomputed_z_vals[frame]
    y_vals = precomputed_y_vals[frame]
    is_inside = precomputed_is_inside[frame]

    z_line.set_data(x_vals, y_vals)

    if was_inside[0] and not is_inside and not has_printed[0]:
        print(f'Max Z = {current_z:.2f} - Καθαρά γραφική επίλυση [Animation]')
        has_printed[0] = True
    was_inside[0] = is_inside

    z_label.set_text(f'Z = {current_z:.2f}')

    if frame == total_animation_frames - 1:
        print('\nΤερματισμός animation...')
        ani.event_source.stop()

    return z_line, z_label;
```

```
    ani = animation.FuncAnimation(
        plt.gcf(),
        animate_z_line,
        frames = total_animation_frames,
        interval = 20,
        blit = True # Only redraw the parts of the figure
    )               # that have changed between frames!

    plt.show()

    return;

if __name__ == '__main__':
    main()

# References:
# https://stackoverflow.com/questions/17576508/python-matplotlib-drawing-linear-inequality-functions
```

valid_region_c1_c2_plot.py

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

def main():
    # Δημιουργούμε πίνακα τιμών του c2
    c2_vals = np.linspace(-1, 4, 500)

    # Υπολογίζουμε τις αντίστοιχες τιμές για κάθε ανισότητα:
    bound1 = (6/7) * c2_vals * (-5/3) # κατώτερο όριο
    bound2 = c2_vals / 2.5            # κατώτερο όριο
    bound3 = c2_vals * 2              # ανώτερο όριο
    bound4 = c2_vals * (3/2.5)        # ανώτερο όριο
    bound5 = c2_vals * (2.143/2.5)    # κατώτερο όριο

    # Υπολογίζουμε τα κατώτερα και ανώτερα όρια
    lower_bound = np.maximum.reduce([bound1, bound2, bound5])
    upper_bound = np.minimum(bound3, bound4)

    # Σχεδίαση
    plt.figure(figsize = (10, 6))
    plt.plot(c2_vals, bound1, label = 'c1 (7/6) > c2 (-5/3)')
    plt.plot(c2_vals, bound2, label = 'c1 2.5 > c2')
    plt.plot(c2_vals, bound3, label = 'c1 < c2 2', linestyle = '--')
    plt.plot(c2_vals, bound4, label = 'c1 2.5 < c2 3', linestyle = '--')
    plt.plot(c2_vals, bound5, label = 'c1 2.5 > c2 2.143', linestyle = ':')

    plt.fill_between(c2_vals, lower_bound, upper_bound,
                     where = lower_bound < upper_bound,
                     color = 'cyan', alpha = 0.3, label = 'Έγκυρη περιοχή')
```

```
    plt.xlabel('c2')
    plt.ylabel('c1')
    plt.title('Περιορισμοί για τα c1 & c2 ώστε η κορυφή (2.5, 3) να είναι βέλτιστη!')
    plt.grid(True)
    plt.legend()
    plt.ylim(0, 4)
    plt.xlim(-1, 4)

    plt.show()

    return;

if __name__ == '__main__':
    main()
```

Κώδικας 2ης Άσκησης

```
# Άσκηση 2η
                                                                    lin_prog_code_HW1_2.py
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from sympy.solvers import solve
from sympy import Symbol
from sympy import lambdify
from itertools import combinations

def p1(x1: Symbol, x2: Symbol) -> float:
    return 0.3*x1 + 0.1*x2 - 2.7;

def p2(x1: Symbol, x2: Symbol) -> float:
    return 0.5*x1 + 0.5*x2 - 6;

def p3(x1: Symbol, x2: Symbol) -> float:
    return 0.6*x1 + 0.4*x2 - 6;

# Αντικειμενική συνάρτηση [Z]
def Z(x1: Symbol, x2: Symbol) -> float:
    return -(0.4*x1 + 0.5*x2);

def main():
    (x1_var, x2_var) = (Symbol('x1'), Symbol('x2'))

    # Οι συναρτήσεις των περιορισμών μας
    constraint_functions = [
        p1(x1_var, x2_var), # ≤ 0
        p2(x1_var, x2_var), # = 0
        p3(x1_var, x2_var), # ≥ 0
        x1_var, x2_var      # ≥ 0
    ]

    inequality_signs = ['<=', '=', '>=', '>=', '>=']
    compiled_constraints = [
        (lambdify((x1_var, x2_var), expr), sign)
        for (expr, sign) in zip(constraint_functions, inequality_signs)
    ] # lambdify: Converts symbolic expressions into Python functions!
```

```
    def check_restrictions(x1_val, x2_val, epsilon = 1e-8):
        # epsilon tolerance for floating point comparisons!!!!!
        for (function, sign) in compiled_constraints:
            value = function(x1_val, x2_val) # Γυρνάει float

            if sign == '>=' and value < -epsilon:
                break;
            elif sign == '<=' and value > epsilon:
                break;
            elif sign == '=' and not np.isclose(value, 0, atol = epsilon):
                break;
        else:
            return True;

        return False;

    # Συνδυασμοί, ανά 2 συναρτήσεων περιορισμών, για τις τομές/κορυφές
    constraint_foos_combo = list(combinations(constraint_functions, 2))

    # Υπολογισμός ΟΛΩΝ των σημείων τομής
    intersection_points = []
    for (cf1, cf2) in constraint_foos_combo:
        sol = solve((cf1, cf2), (x1_var, x2_var))
        if sol: # Ώστε να μην προκύψει IndexError!
            intersection_points.append((float(sol[x1_var]), float(sol[x2_var])))

    # Φιλτράρουμε τα σημεία που δεν ικανοποιούν όλους τους περιορισμούς
    feasible_points = []
    for (candidate_x1, candidate_x2) in intersection_points:
        if check_restrictions(candidate_x1, candidate_x2):
            feasible_points.append((candidate_x1, candidate_x2))

    # Βρίσκουμε το μέγιστο της Z συνάρτησης
    best_value = float('-inf')
    for (candidate_x1, candidate_x2) in feasible_points:
        val = Z(candidate_x1, candidate_x2)
        if val > best_value:
            best_value = val

    print('Κορυφές [εφικτές λύσεις]:')
    for (x1_val, x2_val) in feasible_points:
        print(f'  {(x1_val, x2_val)} -> Z = {Z(x1_val, x2_val)}')
    print() # Καλύτερη αισθητική

    # Γραφική παράσταση
    x_vals = np.linspace(-1, 12, 400)
    plt.figure(figsize = (8, 6))
```

```
# Γραμμές περιορισμών
graphic_representation = [
    ((2.7 - 0.3*x_vals) / 0.1, '(Π1) 0.3x1 + 0.1x2 ≤ 2.7'),
    ((6   - 0.5*x_vals) / 0.5, '(Π2) 0.5x1 + 0.5x2 = 6'),
    ((6   - 0.6*x_vals) / 0.4, '(Π3) 0.6x1 + 0.4x2 ≥ 6')
]
for (y_vals, label) in graphic_representation:
    plt.plot(x_vals, y_vals, label = label)
plt.axhline(0, color = 'black')
plt.axvline(0, color = 'black')

# Εφικτά σημεία στην γραφική παράσταση
for (x1_val, x2_val) in feasible_points:
    plt.plot(x1_val, x2_val, 'mo')

# Περίπου σαν Graham Scan για να βρούμε την εφικτή περιοχή!
feasible_np = np.array(feasible_points)
center = np.mean(feasible_np, axis = 0)
sorted_points = sorted(
    feasible_points,
    key = lambda point: np.arctan2(
        point[1] - center[1], point[0] - center[0]
    )
)
x_sorted = [point[0] for point in sorted_points]
y_sorted = [point[1] for point in sorted_points]
plt.fill(x_sorted, y_sorted, color = 'magenta',
         alpha = 0.3, label = 'Εφικτή περιοχή')

plt.xlim(-1, 12)
plt.ylim(-1, 12)
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.title('Γραφική Λύση Γραμμικού Προβλήματος | Άσκηση 2η')
plt.grid(True)

# ----- Animation setup -----

z_line, = plt.plot(
    [], [], linestyle = '--',
    color = 'blue', linewidth = 2
)
z_label = plt.text(
    0.05, 0.95, '', transform = plt.gca().transAxes,
    color = 'blue', fontweight = 'bold', fontsize = 12,
    verticalalignment = 'top'
)

total_frames = 130 # Το animation μέχρι να φτάσει το max Z!
total_animation_frames = total_frames + 30 # + 30 για να συνεχίσει λίγο ακόμα
```

```
    # ----- Προϋπολογισμός με NumPy για smooth animation -----
    t_vals = np.linspace(0, total_animation_frames / total_frames, total_animation_frames)
    precomputed_z_vals = t_vals * best_value
    precomputed_y_vals = -(precomputed_z_vals[:, None] + 0.4*x_vals[None, :]) / 0.5
    precomputed_is_inside = np.array([
        any(check_restrictions(x, y_, 1e-2) for x, y_ in zip(x_vals, y))
        for y in precomputed_y_vals
    ])

    was_inside = [False]
    has_printed = [False]
    def animate_z_line(frame):
        current_z = precomputed_z_vals[frame]
        y_vals = precomputed_y_vals[frame]
        is_inside = precomputed_is_inside[frame]

        z_line.set_data(x_vals, y_vals)

        if not was_inside[0] and is_inside and not has_printed[0]:
            print(f'Max Z = {current_z:.2f} - Καθαρά γραφική επίλυση [Animation]')
            has_printed[0] = True
        was_inside[0] = is_inside

        z_label.set_text(f'Z = {current_z:.2f}')

        if frame == total_animation_frames - 1:
            print('\nΤερματισμός animation...')
            ani.event_source.stop()

        return z_line, z_label;

    ani = animation.FuncAnimation(
        plt.gcf(),
        animate_z_line,
        frames = total_animation_frames,
        interval = 20,
        blit = True # Only redraw the parts of the figure
    )               # that have changed between frames!

    plt.show()

    return;

if __name__ == '__main__':
    main()
```

Κώδικας 5ης Άσκησης

```python
# Άσκηση 5η - α
                                                    lin_prog_code_HW1_5a.py
import numpy as np
from sympy.solvers import solve
from sympy import Symbol
from sympy import lambdify
from itertools import combinations

def p1(x1: Symbol, x2: Symbol, x3: Symbol) -> float:
    return x1 + x2 - 10;

def p2(x1: Symbol, x2: Symbol, x3: Symbol) -> float:
    return x2 + x3 - 15;

def p3(x1: Symbol, x2: Symbol, x3: Symbol) -> float:
    return x1 + x3 - 12;

def p4(x1: Symbol, x2: Symbol, x3: Symbol) -> float:
    return 20*x1 + 10*x2 + 15*x3 - 300;

# Αντικειμενική συνάρτηση [Z]
def Z(x1: Symbol, x2: Symbol, x3: Symbol) -> float:
    return -(8*x1 + 5*x2 + 4*x3);

def main():
    x1_var = Symbol('x1')
    x2_var = Symbol('x2')
    x3_var = Symbol('x3')

    # Οι συναρτήσεις των περιορισμών μας
    constraint_functions = [
        p1(x1_var, x2_var, x3_var), # ≥ 0
        p2(x1_var, x2_var, x3_var), # ≥ 0
        p3(x1_var, x2_var, x3_var), # ≥ 0
        p4(x1_var, x2_var, x3_var), # ≤ 0
        x1_var, x2_var, x3_var      # ≥ 0
    ]

    inequality_signs = ['>=', '>=', '>=', '<=', '>=', '>=', '>=']
    compiled_constraints = [
        (lambdify((x1_var, x2_var, x3_var), expr), sign)
        for (expr, sign) in zip(constraint_functions, inequality_signs)
    ] # lambdify: Converts symbolic expressions into Python functions!
```

```
def check_restrictions(x1_val, x2_val, x3_val, epsilon = 1e-8):
    # epsilon tolerance for floating point comparisons!!!!!
    for (function, sign) in compiled_constraints:
        value = function(x1_val, x2_val, x3_val) # Γυρνάει float

        if sign == '>=' and value < -epsilon:
            break;
        elif sign == '<=' and value > epsilon:
            break;
    else:
        return True;

    return False;

# Συνάρτηση για να ελέγξουμε αν η κορυφή είναι εκφυλισμένη
def is_degenerate(x1, x2, x3, epsilon = 1e-8):
    active_constraints = 0
    for (f, sign) in compiled_constraints:
        val = f(x1, x2, x3)
        if sign in ['>=', '<='] and np.isclose(val, 0, atol = epsilon):
            active_constraints += 1
        elif sign == '=' and np.isclose(val, 0, atol=epsilon):
            active_constraints += 1

    return active_constraints > 3;

# Συνδυασμοί, ανά 2 συναρτήσεων περιορισμών, για τις τομές/κορυφές
constraint_foos_combo = list(combinations(constraint_functions, 3))

# Υπολογισμός ΟΛΩΝ των σημείων τομής
intersection_points = []
for (cf1, cf2, cf3) in constraint_foos_combo:
    sol = solve((cf1, cf2, cf3), (x1_var, x2_var, x3_var))
    if sol: # Ώστε να μην προκύψει IndexError!
        intersection_points.append(
            (float(sol[x1_var]), float(sol[x2_var]), float(sol[x3_var]))
        )

# Φιλτράρουμε τα σημεία που δεν ικανοποιούν όλους τους περιορισμούς
feasible_points = []
for (candidate_x1, candidate_x2, candidate_x3) in intersection_points:
    if check_restrictions(candidate_x1, candidate_x2, candidate_x3):
        feasible_points.append((candidate_x1, candidate_x2, candidate_x3))

# Βρίσκουμε το μέγιστο της Z συνάρτησης
best_value = float('-inf')
for (candidate_x1, candidate_x2, candidate_x3) in feasible_points:
    val = Z(candidate_x1, candidate_x2, candidate_x3)
    if val > best_value:
        best_value = val
```

```
    print('Κορυφές [Εφικτές: + | Εκφυλισμένες: Δ | Μη εφικτές: -]:')
    for (x1_val, x2_val, x3_val) in intersection_points:
        temp = '-'
        if (x1_val, x2_val, x3_val) in feasible_points:
            temp = '+'
            if is_degenerate(x1_val, x2_val, x3_val):
                temp = 'Δ' # Εκφυλισμένη κορυφή!
        print(f' {temp} {(x1_val, x2_val, x3_val)} -> Z = {Z(x1_val, x2_val, x3_val)}')
    print() # Καλύτερη αισθητική

    return;

if __name__ == '__main__':
    main()
```

```
# Άσκηση 5η - β
                                                        lin_prog_code_HW1_5b.py
from sympy import symbols, Matrix, N
from itertools import combinations

def main():
    # Βασικές μεταβλητές x1, x2, x3 και μεταβλητές χαλάρωσης x4, x5, x6, x7
    (x1, x2, x3, x4, x5, x6, x7) = symbols('x1 x2 x3 x4 x5 x6 x7')
    all_vars = [x1, x2, x3, x4, x5, x6, x7]

    # Δημιουργία του πίνακα A (4x7), σύμφωνα με τις ισοδυναμίες:
    #  1)   x1 +   x2        >=  10 =>   x1 +   x2 -        x4          =  10
    #  2)         x2 +   x3 >=  15 =>         x2 +   x3 -    x5       =  15
    #  3)   x1 +          x3 >=  12 =>   x1 +          x3 -       x6    =  12
    #  4) 20x1 + 10x2 + 15x3 <= 300 => 20x1 + 10x2 + 15x3 +          x7 = 300
    A = Matrix([
        [ 1,  1,  0, -1,  0,  0,  0],
        [ 0,  1,  1,  0, -1,  0,  0],
        [ 1,  0,  1,  0,  0, -1,  0],
        [20, 10, 15,  0,  0,  0,  1]
    ])
    b = Matrix([10, 15, 12, 300])

    # Συντελεστές αντικειμενικής συνάρτησης Z = 8x1 + 5x2 + 4x3
    c = Matrix([8, 5, 4, 0, 0, 0, 0])

    # Θα επιλέξουμε 4 στήλες από τις 7 (όσες και οι εξισώσεις), σχηματίζοντας υποπίνακα B
    # Σελ. 31 / 70 - 01. Εισαγωγή στον Γραμμικό Προγραμματισμό - Αλγόριθμος Simplex
    m = 4 # Αριθμός περιορισμών
    count_bfs = 0

    # Συνάρτηση για τον υπολογισμό της τιμής της αντικειμενικής συνάρτησης
    def objective_value(x_full):
        return -sum(c[i] * x_full[i] for i in range(len(all_vars)));
```

```
    # Ξεκινάμε τον έλεγχο όλων των συνδυασμών 4 στηλών από 7
    for basis_cols in combinations(range(len(all_vars)), m):
        # Φτιάχνουμε τον 4x4 υποπίνακα
        B = A[:, basis_cols] # Όλες οι γραμμές, μόνο οι στήλες των βάσεων

        try:
            # Επίλυση του συστήματος B * xB = b
            xB = B.LUsolve(b)
        except:
            # Αν ο πίνακας είναι μη αντιστρέψιμος, προχωράμε στον επόμενο!
            continue;

        # Δημιουργία του full_x με 7 θέσεις (για x1..x7), ενημερώνουμε μόνο τις θέσεις βάσης
        full_x = [0.0] * len(all_vars)
        for (i, col_idx) in enumerate(basis_cols):
            full_x[col_idx] = float(N(xB[i]))

        # Υπολογισμός Z
        Z_val = objective_value(full_x)

        # Ελεγχος εκφυλισμού
        # Σελ. 45 / 70 - 01. Εισαγωγή στον Γραμμικό Προγραμματισμό - Αλγόριθμος Simplex
        # εναλλακτικά, μετρώντας πόσες είναι 0
        basic_indices = list(basis_cols) # indices των βασικών μεταβλητών
        degenerate = any(abs(full_x[idx]) < 1e-9 for idx in basic_indices) # Οριακά 0!

        # Τύπωση αποτελέσματος
        temp = 'Δ' if degenerate else '+' # + => εφικτή & Δ => εκφυλισμένη

        # Έλεγχος εφικτότητας: όλες οι x >= 0
        if any(val < -1e-9 for val in full_x):
            temp = '-' # - => μη εφικτή

        var_names = [str(all_vars[i]) for i in basis_cols]
        basis_str = '{' + ', '.join(var_names) + '}'
        x_vals_aligned = ', '.join(f"{v}={full_x[idx]:>8.3f}"
                                   for v, idx in zip(var_names, basis_cols))
        print(f"{temp} Βάση: {basis_str} =>   {x_vals_aligned} | Z = {Z_val:8.3f}")

        count_bfs += 1

    print(f"\nΣύνολο βασικών [εφικτών και μη] λύσεων: {count_bfs}")

    return;

if __name__ == '__main__':
    main()
```

Κώδικας 6ης Άσκησης

```
# Άσκηση 6η - α
                                                        lin_prog_code_HW1_6a.py
from sympy import symbols, Matrix, nsimplify

def print_tableau(iteration, basic_vars, tableau, xB_values, z_row, Z_val):
    # Global μεταβλητές: all_vars, var_indices

    col_width = 6
    header = [f"Βήμα {iteration}"] + [str(v) for v in all_vars] + ["b"]

    # Επικεφαλίδα
    print(" " + " | ".join(f"{name:>{col_width}}" for name in header))
    print("-" * ((col_width + 3) * len(header) - 1)) # διαχωριστική γραμμή

    # Γραμμή -Z
    z_line = ["-Z"] + [str(v) for v in z_row] + [str(Z_val)]
    print(" " + " | ".join(f"{val:>{col_width}}" for val in z_line))

    print("-" * ((col_width + 3) * len(header) - 1)) # διαχωριστική γραμμή

    # Γραμμές βασικών μεταβλητών
    for (i, v) in enumerate(basic_vars):
        row_vals = [tableau[i, var_indices[var]] for var in all_vars]
        row = [str(v)] + [str(val) for val in row_vals] + [str(xB_values[i])]

        line = " " + " | ".join(f"{val:>{col_width}}" for val in row)
        if xB_values[i].is_zero:
            line += " <-- Εκφυλισμένη"
        print(line)

    return;

def main():
    # Βασικές μεταβλητές x1, x2, x3, x4 και μεταβλητές χαλάρωσης x5, x6, x7
    (x1, x2, x3, x4, x5, x6, x7) = symbols('x1 x2 x3 x4 x5 x6 x7')
    global all_vars # Δημιουργία καθολής μεταβλητής για όλες τις μεταβλητές!
    all_vars = [x1, x2, x3, x4, x5, x6, x7]

    # Δημιουργία του πίνακα A (3x7), σύμφωνα με τις ισοδυναμίες:
    #  1)  x1 + 2x2 + 4x3 - x4 ≤  6 =>  x1 + 2x2 + 4x3 - x4 + x5       =  6
    #  2) 2x1 + 3x2 -  x3 + x4 ≤ 12 => 2x1 + 3x2 -  x3 + x4 +    x6    = 12
    #  3)  x1 +        x3 + x4 ≤  2 =>  x1 +        x3 + x4 +       x7 =  2
    A = Matrix([
        [1, 2,  4, -1, 1, 0, 0],
        [2, 3, -1,  1, 0, 1, 0],
        [1, 0,  1,  1, 0, 0, 1]
    ])
    b = Matrix([6, 12, 2])
```

```
# Συντελεστές αντικειμενικής συνάρτησης Z = 2x1 + x2 + 6x3 - 4x4
c = Matrix([2, 1, 6, -4, 0, 0, 0])

# Αρχική βάση: x5, x6, x7 | ### Βήμα - Αρχικοποίηση
'''Σελ. 47=>49 / 70 - 01. Εισαγωγή στον Γραμμικό Προγραμματισμό
                        - Αλγόριθμος Simplex'''
basic_vars    = [x5, x6, x7]
nonbasic_vars = [x1, x2, x3, x4]

# Δείκτες μεταβλητών
global var_indices # Δημιουργία καθολικής μεταβλητής για τους δείκτες των μεταβλητών!
var_indices = {v: i for (i, v) in enumerate(all_vars)}

iteration = 0
while True:
    ### Βήμα - Απαλοιφή Gauss
    B = A[:, [var_indices[v] for v in basic_vars]]
    B_inv = B.inv()

    tableau = B_inv * A
    xB_values = B_inv * b
    if any(val < 0 for val in xB_values):
        print('Δεν υπάρχει αρχικά εφικτή λύση: B⁻¹b < 0!')
        return;

    # Υπολογισμός της γραμμής Z
    c_B = Matrix([c[var_indices[v]] for v in basic_vars])

    z_row = []
    for v in all_vars:
        if v in basic_vars:
            z_row.append(0)
        else:
            Aj = A[:, var_indices[v]]
            zj = c[var_indices[v]] - (c_B.T * B_inv * Aj)[0, 0]
            z_row.append(zj)

    Z_val = -(Matrix([c[var_indices[v]] for v in basic_vars]).T * xB_values)[0, 0]
    print_tableau(iteration, basic_vars, tableau, xB_values, z_row, Z_val)

    # Έλεγχος βέλτιστης λύσης
    if all(z <= 0 for z in z_row):
        print('\nΗ λύση είναι βέλτιστη!')
        break;
```

```
        ### Βήμα - Επιλογή νέας βασικής μεταβλητής
        entering_idx = None
        print(f'\nΕπιλογή της 1ης μεταβλητής με Z > 0 ως εισερχόμενη:')
        for i in range(len(z_row)):
            print(f'z = {str(z_row[i]):>5} | {all_vars[i]}', end = ' ')
            if z_row[i] > 0:
                entering_idx = i
                print(' <-- Εισερχόμενη βασική μεταβλητή')
                break;
            print() # Καλύτερη αισθητική

        entering_var = nonbasic_vars[entering_idx]

        ### Βήμα - Κριτήριο ελαχίστου λόγου
        d = B_inv * A[:, var_indices[entering_var]]
        ratios = []
        for i in range(len(d)):
            if d[i] > 0:
                ratios.append(xB_values[i] / d[i])
            else:
                ratios.append(float('inf'))

        if all(r == float('inf') for r in ratios):
            print("Το πρόβλημα δεν είναι φραγμένο!")
            return;

        min_ratio = min(ratios)
        exiting_idx = ratios.index(min_ratio)
        exiting_var = basic_vars[exiting_idx]

        # Εμφάνιση του πίνακα με τους λόγους
        print("\nΚριτήριο ελαχίστου λόγου:")
        print("-" * 40)
        print(f"{'Βασική':<10} | {'xB[i]':>7} | {'d[i]':>7} | {'Λόγος':>7}")
        print("-" * 40)
        for i, v in enumerate(basic_vars):
            xb = xB_values[i]
            di = d[i]
            ratio_str = "-" if di <= 0 else f"{xb/di}"
            selected = "  <-- ελάχιστος" if i == exiting_idx else ""
            print(f"{str(v):<10} | {str(xb):>7} | {str(di):>7} | {ratio_str:>7}{selected}")

        print(f"Εξερχόμενη μεταβλητή: {exiting_var}\n")

        ### Βήμα - Ανταλλαγή μεταβλητών
        basic_vars[exiting_idx] = entering_var
        nonbasic_vars[entering_idx] = exiting_var
        iteration += 1
```

```
    # --- Εκτύπωση τελικής λύσης ---
    full_solution = [0.0] * len(all_vars)
    for (i, var) in enumerate(basic_vars):
        full_solution[var_indices[var]] = float(xB_values[i])

    print() # Καλύτερη αισθητική
    Z_val = sum(c[i] * full_solution[i] for i in range(len(all_vars)))
    for (i, val) in enumerate(full_solution):
        val_rat = nsimplify(val)
        print(f"x{i+1} = {val_rat}", end=", " if i < len(full_solution) - 1 else " ")
    Z_rat = nsimplify(Z_val)
    print(f"με Z = {Z_rat}\n")

    return;

if __name__ == '__main__':
    main()
```

```
# Άσκηση 6η - β
                                                    lin_prog_code_HW1_6b.py

import matplotlib.pyplot as plt
import networkx as nx
from sympy import Matrix, symbols
from copy import deepcopy
from collections import deque

# Δημιουργία μεταβλητών
(x1, x2, x3, x4, x5, x6, x7) = symbols('x1 x2 x3 x4 x5 x6 x7')
all_vars = [x1, x2, x3, x4, x5, x6, x7]
var_indices = {v: i for i, v in enumerate(all_vars)}

# Πίνακας περιορισμών και αντικειμενική συνάρτηση
A = Matrix([
    [1, 2,  4, -1, 1, 0, 0],
    [2, 3, -1,  1, 0, 1, 0],
    [1, 0,  1,  1, 0, 0, 1]
])
b = Matrix([6, 12, 2])
c = Matrix([2, 1, 6, -4, 0, 0, 0])

# Αρχικές μεταβλητές
initial_basic = [x5, x6, x7]
initial_nonbasic = [x1, x2, x3, x4]
```

```
# Γράφος
G = nx.DiGraph()
node_labels = {}
edge_labels = {}
node_id_map = {}
z_values = {}
node_count = 0

queue = deque()
queue.append((initial_basic, initial_nonbasic))
visited = set()

while queue:
    basic_vars, nonbasic_vars = queue.popleft()
    basis_key = tuple(sorted(str(v) for v in basic_vars))
    if basis_key in visited:
        continue;
    visited.add(basis_key)

    B = A[:, [var_indices[v] for v in basic_vars]]
    try:
        B_inv = B.inv()
        xB = B_inv * b
        if any(val < 0 for val in xB):
            continue;
    except:
        continue;

    # Υπολογισμός Z και z_row
    c_B = Matrix([c[var_indices[v]] for v in basic_vars])
    Z_val = float((c_B.T * xB)[0])
    z_row = []
    for v in all_vars:
        if v in basic_vars:
            z_row.append(0)
        else:
            Aj = A[:, var_indices[v]]
            zj = c[var_indices[v]] - (c_B.T * B_inv * Aj)[0, 0]
            z_row.append(zj)

    # Κόμβος γράφου
    basis_set = frozenset(basic_vars)
    if basis_set not in node_id_map:
        node_name = f"v{node_count}"
        node_id_map[basis_set] = node_name

        # Πλήρης λύση και εμφάνιση μόνο x1-x4 (ταξινομημένα)
        full_solution = [0.0] * len(all_vars)
        for i, v in enumerate(basic_vars):
            full_solution[var_indices[v]] = float(xB[i])
        values = tuple(round(full_solution[i], 2) for i in range(4))
```

```
            label = f"BI={{{','.join(str(v)[-1] for v in sorted(basic_vars, key=lambda v: int(str(v)[1:])))}}}\n"
            label += f"{values}\nz={Z_val:.2f}"
            node_labels[node_name] = label
            G.add_node(node_name)
            current_node = node_name
            node_count += 1
        else:
            current_node = node_id_map[basis_set]

        z_values[basis_set] = Z_val

        if all(z <= 0 for z in z_row):
            continue;

        # Εξέταση κάθε εισερχόμενης με z > 0
        for entering_idx, zj in enumerate(z_row):
            if zj <= 0:
                continue;

            entering_var = all_vars[entering_idx]
            if entering_var in basic_vars:
                continue;

            d = B_inv * A[:, var_indices[entering_var]]
            ratios = []
            for i in range(len(d)):
                if d[i] > 0:
                    ratios.append(xB[i] / d[i])
                else:
                    ratios.append(float('inf'))

            if all(r == float('inf') for r in ratios):
                continue;

            min_ratio = min(ratios)
            for i, r in enumerate(ratios):
                if abs(r - min_ratio) < 1e-8:
                    exiting_var = basic_vars[i]

                    new_basic = basic_vars.copy()
                    new_nonbasic = nonbasic_vars.copy()
                    new_basic[i] = entering_var
                    new_nonbasic[new_nonbasic.index(entering_var)] = exiting_var
```

```
                new_key = frozenset(new_basic)
                B_new = A[:, [var_indices[v] for v in new_basic]]
                try:
                    B_new_inv = B_new.inv()
                    xB_new = B_new_inv * b
                    if any(x < 0 for x in xB_new):
                        continue;
                except:
                    continue;

                if new_key not in node_id_map:
                    node_name = f"v{node_count}"
                    node_id_map[new_key] = node_name
                    z_val_new = float((Matrix([c[var_indices[v]] for v in new_basic]).T * xB_new)[0])

                    full_solution = [0.0] * len(all_vars)
                    for i2, v in enumerate(new_basic):
                        full_solution[var_indices[v]] = float(xB_new[i2])
                    values = tuple(round(full_solution[i], 2) for i in range(4))

                    label = f"BI={{{','.join(str(v)[-1] for v in sorted(new_basic, key=lambda v: int(str(v)[1:])))}}}\n"
                    label += f"{values}\nz={z_val_new:.2f}"
                    node_labels[node_name] = label
                    G.add_node(node_name)
                    target_node = node_name
                    node_count += 1
                else:
                    target_node = node_id_map[new_key]

                G.add_edge(current_node, target_node)
                edge_labels[(current_node, target_node)] = f"+{str(entering_var)} / -{str(exiting_var)} =>"

                queue.append((new_basic, new_nonbasic))

# Ζωγραφική!!!
best_node = max(z_values, key=z_values.get)
best_node_name = node_id_map[best_node]
node_colors = ["lightgreen" if node == best_node_name else "peachpuff" for node in
G.nodes()]

plt.figure(figsize=(16, 12))

layout = {
    'v0': (0, 0),
    'v3': (2, 1),
    'v2': (1, 0),
    'v1': (1, -1),
    'v4': (2, 0),
    'v5': (3, 0)
}
```

```
nx.draw_networkx_nodes(G, layout, node_color = node_colors,
                       edgecolors = "black", node_size = 11000)
nx.draw_networkx_edges(G, layout)
nx.draw_networkx_labels(G, layout, labels = node_labels, font_size = 10)
nx.draw_networkx_edge_labels(G, layout, edge_labels = edge_labels, font_size = 9)

plt.title("Simplex Adjacency Graph")
plt.axis("off")
plt.tight_layout()
plt.show()
```