ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΑΝΑΘΕΩΡΗΜΕΝΟΥ ΑΛΓΟΡΙΘΜΟΥ SIMPLEX (REVISED SIMPLEX ALGORITHM) ΣΕ PYTHON

Διπλωματική Εργασία

του

Τσερπισνού Γεωργίου

Θεσσαλονίκη, 06/2019

ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΑΝΑΘΕΩΡΗΜΕΝΟΥ ΑΛΓΟΡΙΘΜΟΥ SIMPLEX (REVISED SIMPLEX ALGORITHM) ΣΕ PYTHON

Τσερπισνός Γεώργιος

Διπλωματική Εργασία

υποβαλλόμενη για τη μερική εκπλήρωση των απαιτήσεων του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΤΙΤΛΟΥ ΣΠΟΥΔΩΝ ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

Επιβλέπων Καθηγητής

Σαμαράς Νικόλαος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Σαμαράς Νικόλαος	Χρήστου-Βαρσακέλης Δημήτριος	Σιφαλέρας Άγγελος
.....	.....	.....

Τσερπισνός Γεώργιος

.....

Περίληψη

Η μέθοδος Simplex αποτελεί την πιο γνωστή και διαδεδομένη μέθοδος για την επίλυση προβλημάτων γραμμικού προγραμματισμού. Ο αναθεωρημένος αλγόριθμος Simplex αποτελεί μία βελτιστοποιημένη παραλλαγή της βασικής μεθόδου Simplex, η οποία επιτρέπει τον γρηγορότερο υπολογισμό των παραμέτρων του προβλήματος, κάνοντας χρήση των υπολογισμών που έχουν ήδη εκτελεστεί σε κάποια επανάληψη του αλγορίθμου. Υπάρχουν πολλές παραλλαγές του αλγορίθμου Simplex, οι οποίες έχουν υλοποιηθεί σε διάφορες γλώσσες προγραμματισμού και πολλές υπολογιστικές μελέτες που έχουν πραγματοποιηθεί και αφορούν την επίδοση των αλγορίθμων αυτών.

Στην παρούσα διπλωματική εργασία, γίνεται η ανάπτυξη κάποιων αλγορίθμων τύπου Simplex σε γλώσσα προγραμματισμού Python, για την επίλυση του γενικού γραμμικού προβλήματος. Για το σκοπό αυτό, αναπτύσσεται ένα σύνολο συναρτήσεων, οι οποίες επιτρέπουν τη δημιουργία και την ανάκτηση γραμμικών προβλημάτων που είναι αποθηκευμένα σε μορφή mps και στη συνέχεια την επίλυσή τους με εφαρμογή των αλγορίθμων. Στη συνέχεια, πραγματοποιείται μια εκτενής υπολογιστική μελέτη για τη μέτρηση της υπολογιστικής συμπεριφοράς των αλγορίθμων αυτών, μέσω της επίλυσης προβλημάτων διαφορετικών διαστάσεων και πυκνοτήτων, τα οποία παράγονται αυτόματα με τη βοήθεια γεννήτριας τυχαίων αριθμών. Τα αποτελέσματα των μελετών αυτών, παρουσιάζονται με γραφικό τρόπο και στη συνέχεια αναλύονται τα συμπεράσματα τα οποία προκύπτουν.

Λέξεις Κλειδιά: Αλγόριθμοι τύπου Simplex, αναθεωρημένος αλγόριθμος Simplex, υλοποίηση αλγορίθμων τύπου Simplex σε Python, υπολογιστική μελέτη αλγορίθμων τύπου Simplex, Δυϊκός αλγόριθμος Simplex

Abstract

Simplex algorithm is the most famous and widespread algorithm for the solution of linear programming problems. The revised Simplex algorithm is an optimized modification of the basic Simplex algorithm, that allows the faster computation of the parameters of a linear problem, by using the already executed computations performed during the algorithm's iterations. There are many variations of the Simplex algorithm, implemented in various programming languages, and plenty of computational studies about the performance of these algorithms.

Some Simplex-type algorithms have been implemented in Python, as part of this thesis, for the solution of the general linear problem. For that purpose, a number of functions have been designed that allow the creation and the retrieval of a linear problem in mps format, so that they can later be solved by using the implemented algorithms. Thereafter, an extensive computational study is carried out for the measurement of the computational behavior of these algorithms, for different problem dimensions and densities, that are produced automatically by using random number generators. The results deduced by this analysis are depicted in a graphical way and the conclusions produced are presented.

Keywords: Simplex-type algorithms, revised Simplex algorithm, implementation of Simplex-type algorithms in Python, computational study of Simplex-type algorithms, Dual Simplex algorithm

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

1 Εισαγωγή

1.1 Πρόβλημα – Σημαντικότητα του θέματος

Ο γραμμικός προγραμματισμός (linear programming) ασχολείται με το σύνολο των τεχνικών που ασχολούνται με την μαθηματική βελτιστοποίηση μίας γραμμικής συνάρτησης, η οποία λέγεται αντικειμενική συνάρτηση, με τέτοιο τρόπο ώστε οι μεταβλητές που περιλαμβάνει να ικανοποιούν ένα δοθέν σύνολο περιορισμών. Η πιο γνωστή και διαδεδομένη μέθοδος για την επίλυση προβλημάτων γραμμικού προγραμματισμού, είναι η μέθοδος Simplex.

Ένας αλγόριθμος τύπου Simplex αποτελείται από ένα σύνολο προκαθορισμένων βημάτων, στα οποία γίνονται κάποιοι επαναλαμβανόμενοι υπολογισμοί, ξεκινώντας από ένα αρχικό σημείο της περιοχής των εφικτών λύσεων του προβλήματος, μέχρι να βρεθεί η βέλτιστη λύση για το πρόβλημα αυτό. Η μέθοδος αυτή, αναπτύχθηκε αρχικά από τον George Dantzig το 1947, αλλά στη συνέχεια έχουν αναπτυχθεί πολλές παραλλαγές και τροποποιήσεις της μεθόδου, καθώς και διάφορες υλοποιήσεις σε γνωστές γλώσσες προγραμματισμού. Με τη χρήση αλγορίθμων τύπου Simplex μπορεί να επιτευχθεί η γρήγορη επίλυση προβλημάτων βελτιστοποίησης τα οποία συναντώνται πολύ συχνά σε διάφορους τομείς, όπως της οικονομίας, της παραγωγικής διαδικασίας, της κατανομής πόρων, της διοίκησης προσωπικού κτλ. Θεωρούνται μάλιστα οι αλγόριθμοι αυτοί, κατάλληλοι σε περιπτώσεις που έχουμε μεγάλο αριθμό μεταβλητών και περιορισμών στους οποίους υπόκεινται οι μεταβλητές αυτές. Είναι πολύ σημαντικό επομένως, να υπάρχουν γρήγορες υλοποιήσεις των αλγορίθμων τύπου Simplex, σε διαδεδομένες και σύγχρονες γλώσσες προγραμματισμού, όπως η γλώσσα προγραμματισμού Python, η οποία είναι πολύ διαδεδομένη κατά τα τελευταία χρόνια.

1.2 Σκοπός – Στόχοι

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη κάποιων αλγορίθμων τύπου Simplex σε γλώσσα προγραμματισμού Python, για την επίλυση γενικών γραμμικών προβλημάτων. Στους στόχους της εργασίας περιλαμβάνεται η ανάπτυξη ενός συνόλου ειδικών συναρτήσεων οι οποίες, πέρα από την επίλυση γραμμικών προβλημάτων, επιτρέπουν την ανάγνωση γραμμικών προβλημάτων που είναι αποθηκευμένα σε μορφή mps, καθώς και η παραγωγή τυχαίων προβλημάτων τα οποία θα μπορούν επίσης να αποθηκεύονται στη μορφή αυτή. Επιπλέον, θα πραγματοποιηθεί μια εκτενής υπολογιστική μελέτη για τη μέτρηση της υπολογιστικής συμπεριφοράς των αλγορίθμων που θα αναπτυχθούν και θα γίνει παρουσίαση των αποτελεσμάτων με γραφικό τρόπο.

1.3 Διάρθρωση της μελέτης

Στο κεφάλαιο 1 της διπλωματικής εργασίας παρουσιάζεται το πρόβλημα το οποίο μελετά τη εργασία και περιγράφονται ο σκοπός και οι στόχοι της εργασίας.

Στο κεφάλαιο 2 γίνεται μία περιγραφή του θεωρητικού υποβάθρου του προβλήματος και παρουσιάζεται η έρευνα που έχει πραγματοποιηθεί στον χώρο αυτόν. Περιγράφονται επίσης τα βήματα από τα οποία αποτελούνται οι διάφορες παραλλαγές των αλγορίθμων τύπου Simplex, προβλήματα τα οποία μπορούν να προκύψουν κατά την επίλυση ενός γραμμικού προβλήματος, τρόποι εύρεσης μίας αρχικής εφικτής λύσης, καθώς και οι τεχνικές κλιμάκωσης που θα μπορούσαν να χρησιμοποιηθούν κατά την επίλυση των προβλημάτων.

Στη συνέχεια, στο κεφάλαιο 3, παρουσιάζεται η υλοποίηση κάποιων αλγορίθμων τύπου Simplex σε γλώσσα προγραμματισμού Python, δίνοντας έμφαση στον αναθεωρημένο αλγόριθμο Simplex. Περιγράφονται επίσης, διάφορες άλλες βοηθητικές συναρτήσεις για τη δημιουργία γραμμικών προβλημάτων, την ανάκτησή τους από αρχεία μορφής mps, την αποθήκευσή τους στη μορφή αυτή κ.λπ.

Στο κεφάλαιο 4, γίνεται μία υπολογιστική μελέτη των αλγορίθμων που αναπτύχθηκαν στο προηγούμενο κεφάλαιο. Η μελέτη αφορά προβλήματα που δημιουργήθηκαν με

τυχαίο τρόπο, με τη βοήθεια ειδικών συναρτήσεων σε γλώσσα Python οι οποίες υλοποιήθηκαν. Για τα προβλήματα αυτά, καταγράφεται ο μέσος όρος του πλήθους των επαναλήψεων που απαιτούνται για την επίλυσή τους, όπως και ο αντίστοιχος χρόνος. Τα αποτελέσματα παρουσιάζονται με γραφικό τρόπο με τη βοήθεια των κατάλληλων γραφημάτων.

Τέλος, στο κεφάλαιο 5 παρουσιάζονται τα συμπεράσματα τα οποία προκύπτουν από την υλοποίηση των αλγορίθμων και την υπολογιστική μελέτη που ακολούθησε. Επίσης, γίνονται κάποιες προτάσεις για μελλοντικές επεκτάσεις των αλγορίθμων που αναπτύχθηκαν και περαιτέρω έρευνα.

2 Θεωρητικό Υπόβαθρο και Βιβλιογραφική Επισκόπηση

2.1 Το γενικό γραμμικό πρόβλημα

Ως γραμμικός προγραμματισμός (linear programming), θα μπορούσε να οριστεί, το σύνολο των τεχνικών για την επίλυση προβλημάτων βελτιστοποίησης, ώστε να βρεθεί η μέγιστη ή η ελάχιστη τιμή μιας αντικειμενικής συνάρτησης, της οποίας οι μεταβλητές υπόκεινται σε ένα πεπερασμένο σύνολο περιορισμών.

Ένα πρόβλημα γραμμικού προγραμματισμού, στην γενική του μορφή μπορεί να έχει την πιο κάτω μορφή:

$$\min (\text{ή } \max)\ z = c^T x$$

$$\text{μ.π.} \quad Ax \oplus b$$

$$x \geq 0$$

όπου

- μ.π. αποτελεί συντομογραφία της έκφρασης “με περιορισμούς”.
- min αποτελεί συντομογραφία της έκφρασης “να βρεθεί το ελάχιστο της συνάρτησης” .
- ο x είναι πίνακας διάστασης nx1 περιέχει τις μεταβλητές απόφασης.
- ο πίνακας A είναι ένας πίνακας πραγματικών αριθμών, διάστασης mxn, ο οποίος περιέχει τους συντελεστές που έχουν οι μεταβλητές στους περιορισμούς.
- ο πίνακας c είναι διάστασης nx1 και περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- ο πίνακας b είναι διάστασης mx1 και περιέχει τους σταθερούς όρους στους περιορισμούς.
- n είναι το πλήθος των μεταβλητών.
- m είναι το πλήθος των περιορισμών.

- το σύμβολο $\oplus$ δηλώνει ότι τον τύπο της ανισότητας που υπάρχει σε έναν περιορισμό και μπορεί να έχει μία από τις επόμενες μορφές: $=, \geq$ ή $\leq$.
- z είναι η τιμή της αντικειμενικής συνάρτησης που προκύπτει από τον πολλαπλασιασμό πινάκων $c^T x$.

Αντί για πρόβλημα ελαχιστοποίησης (minimization problem), θα μπορούσαμε να έχουμε πρόβλημα μεγιστοποίησης (maximization problem), οπότε στην περίπτωση αυτή χρησιμοποιείται ο συμβολισμός max, αντί του συμβολισμού min, ώστε να δηλώσει την έκφραση "να βρεθεί το μέγιστο της συνάρτησης". Η πιο πάνω μορφή ονομάζεται **γενική μορφή** του προβλήματος γραμμικού προγραμματισμού. Αντίστοιχα, ένα πρόβλημα γραμμικού προγραμματισμού, σε **κανονική ή ανισοτική μορφή** (canonical form), έχει την πιο κάτω μορφή:

$$\min (\text{ή } \max)\ z = c^T x$$

$$\text{μ.π.} \quad Ax \otimes b$$

$$x \geq 0$$

όπου το σύμβολο $\otimes$ δηλώνει έναν από τους επόμενους τελεστές: $\geq$ ή $\leq$.

Η αναλυτική μορφή για το πρόβλημα αυτό, μπορεί να γραφεί ως εξής:

$$\min (\text{ή } \max)\ c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

μ.π.

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_nx_n \otimes b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n \otimes b_2$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n \otimes b_m$$

$$x_j \geq 0, \quad (j = 1, \ldots, n)$$

Η **τυποποιημένη ή ισοτική μορφή** (standard form) του γραμμικού προβλήματος είναι της μορφής:

$$\min (\text{ή } \max)\ z = c^T x$$

$$\text{μ.π.} \quad Ax = b$$

$$x \geq 0$$

ή πιο αναλυτικά

$$\min (\text{ή } \max)\ c_1x_1 + c_2x_2 + \ldots + c_nx_n$$

μ.π.

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_nx_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$

……

$$a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m$$

$$x_j \geq 0, \quad (j = 1, \ldots, n)$$

Ένα πρόβλημα μεγιστοποίησης μπορεί να μετατραπεί σε ένα πρόβλημα ελαχιστοποίησης, πολλαπλασιάζοντας τους συντελεστές της αντικειμενικής συνάρτησης με το -1. Μετά τη βελτιστοποίηση του νέου προβλήματος, για να βρούμε τη βέλτιστη λύση του αρχικού προβλήματος, αρκεί να πολλαπλασιάσουμε τη λύση που βρέθηκε με το -1. Όμοια, πολλαπλασιάζοντας με το -1 μπορούμε να μετατρέψουμε ένα πρόβλημα ελαχιστοποίησης σε πρόβλημα μεγιστοποίησης. Δηλαδή μπορούμε να πούμε ότι ισχύει ο πιο κάτω τύπος:

$$\max \sum_{i=1}^{n} c_i x_i = -\min \sum_{i=1}^{n} -c_i x_i$$

2.1.1 Επίλυση του Γραμμικού Προβλήματος

Το σύνολο των τιμών των μεταβλητών απόφασης οι οποίες ικανοποιούν όλους τους περιορισμούς, λέμε ότι αποτελούν την **εφικτή περιοχή** (feasible region) για το πρόβλημα. Για παράδειγμα, για το πιο κάτω πρόβλημα με 2 μεταβλητές απόφασης:

$$\max x_1 + x_2$$

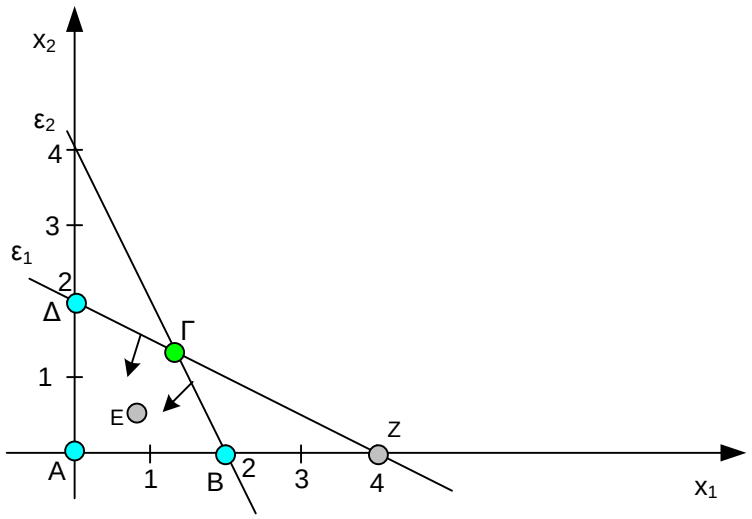
μ.π.

$$2x_1 + x_2 \leq 4$$

$$x_1 + 2x_2 \leq 4$$

$$x_1 \geq 0, \ x_2 \geq 0$$

Έχουμε την εφικτή περιοχή που ορίζεται από τα σημεία Α, Β, Γ και Δ του σχήματος στην Εικόνα 2-1 . Η εφικτή αυτή περιοχή καθορίζεται από τις ευθείες ε_1 και ε_2, οι οποίες αντιστοιχούν στους περιορισμούς του προβλήματος. Είναι σίγουρο ότι η βέλτιστη λύση βρίσκεται σε μία από τις κορυφές της εφικτής περιοχής. Εάν 2 κορυφές αντιστοιχούν σε βέλτιστες λύσεις, τότε υπάρχουν άπειρες λύσεις, αφού και όλα τα σημεία ανάμεσα στις κορυφές αυτές είναι βέλτιστα. Στην περίπτωσή μας, η μέγιστη λύση που ψάχνουμε αντιστοιχεί στο σημείο Γ του σχήματος.

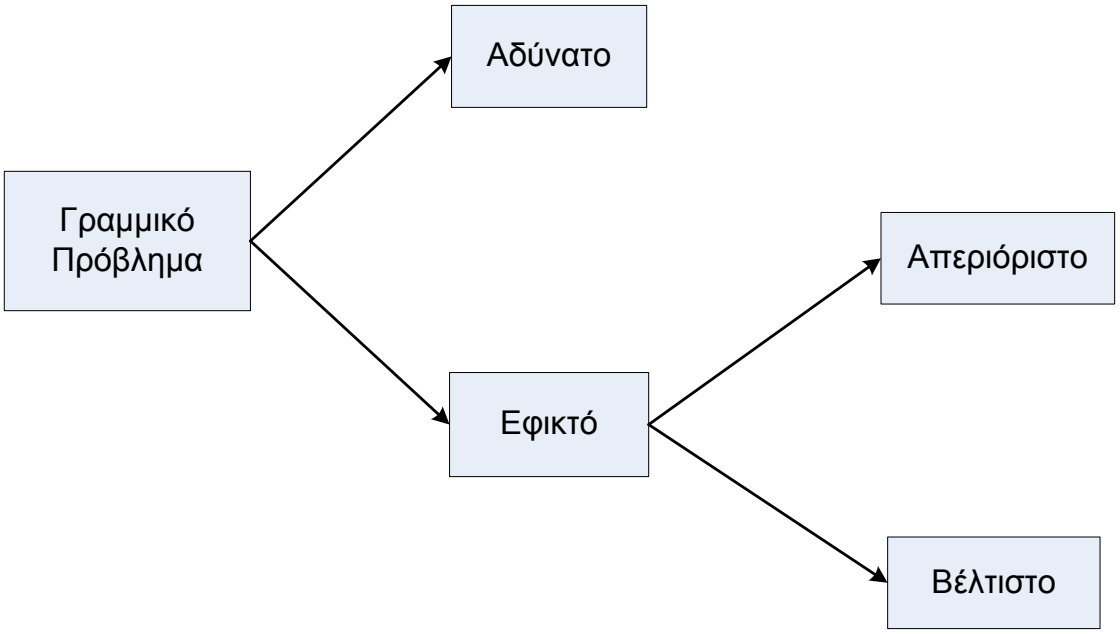


Εικόνα 2-1: Γραφική αναπαράσταση γραμμικού προβλήματος

Για την επίλυση των προβλημάτων γραμμικού προγραμματισμού έχουν εφαρμοστεί διάφοροι αλγόριθμοι επίλυσής τους. Οι αλγόριθμοι αυτοί ξεκινούν από μία αρχική εφικτή λύση του προβλήματος και προσπαθούν να φτάσουν στη βέλτιστη λύση, τροποποιώντας κάθε φορά την τρέχουσα λύση. Η μετάβαση αυτή, από μία λύση σε μία άλλη λέγεται και περιστροφή (pivot). Οι αλγόριθμοι αυτοί μπορούν να χωριστούν σε κάποιες βασικές κατηγορίες, όπως φαίνεται πιο κάτω:

- Αλγόριθμοι τύπου Simplex (Simplex-type algorithms). Οι αλγόριθμοι αυτοί ξεκινούν από την αρχική λύση και σε κάθε επανάληψη μπορούν να κινηθούν από κορυφή σε κορυφή της εφικτής περιοχής, μέχρι να φτάσουν στη βέλτιστη λύση (Dantzig 1948). Στο παράδειγμα της Εικόνα 2-1, ένας αλγόριθμος τύπου Simplex μπορεί να κινηθεί από την κορυφή Α στην κορυφή Β και από εκεί στην κορυφή Γ που αποτελεί τη βέλτιστη λύση.

- Αλγόριθμοι εσωτερικών σημείων (interior point algorithms), οι οποίοι μπορούν να κινηθούν στο εσωτερικό της εφικτής περιοχής ώστε να φτάσουν στη βέλτιστη λύση (Karmarkar 1984). Στο πιο πάνω παράδειγμα, θα μπορούσε ένας αλγόριθμος να πάει από το σημείο Α στο σημείο Ε και από εκεί στη βέλτιστη λύση στο σημείο Γ.
- Αλγόριθμοι εξωτερικών σημείων (exterior point algorithms), οι οποίοι έχουν το δικαίωμα να κινηθούν και έξω από την εφικτή περιοχή, κάτι που ίσως μπορεί να τους οδηγήσει στη γρηγορότερη εύρεση της βέλτιστης λύσης (Paparrizos 1993). Στο παράδειγμά μας, ένας αλγόριθμος εξωτερικών σημείων θα μπορούσε να πάει από την κορυφή Α στην κορυφή Z και από εκεί στην κορυφή Γ που αποτελεί τη βέλτιστη λύση του προβλήματος.



Εικόνα 2-2: Κατηγορίες γραμμικών προβλημάτων ανάλογα με την ύπαρξη βέλτιστων λύσεων

Ως μέτρο σύγκρισης της απόδοσης δύο αλγορίθμων, μπορεί να χρησιμοποιηθεί το πλήθος των επαναλήψεων που απαιτούνται για να βρεθεί η βέλτιστη λύση, καθώς επίσης και ο χρόνος που χρειάζεται για να φτάσει ο αλγόριθμος στη λύση αυτή. Ο χρόνος για την εύρεση της βέλτιστης λύσης, δεν είναι ανάλογος του αριθμού των επαναλήψεων που χρειάζονται, αφού ανάλογα με το είδος του αλγορίθμου, μία επανάληψη μπορεί να έχει διαφορετικές χρονικές απαιτήσεις.

Ανάλογα με την ύπαρξη ή όχι βέλτιστων λύσεων, ένα γραμμικό πρόβλημα μπορεί να ταξινομηθεί σε μία από τις πιο κάτω κατηγορίες (Εικόνα 2-2):

- **Αδύνατο ή μη εφικτό** (infeasible). Στα προβλήματα αυτά δεν υπάρχει κανένα σημείο, δηλαδή καμία ανάθεση τιμών στις μεταβλητές απόφασης, που να ικανοποιεί όλους τους περιορισμούς.
- **Εφικτό** (feasible). Στα προβλήματα αυτά υπάρχουν σημεία που ικανοποιούν όλους τους περιορισμούς. Με άλλα λόγια η εφικτή περιοχή δεν είναι κενή. Χωρίζονται στις πιο κάτω υποκατηγορίες:
    - **Βέλτιστο** (optimal). Είναι τα προβλήματα για τα οποία υπάρχει μία ή περισσότερες βέλτιστες λύσεις, δηλαδή μπορεί να βρεθεί το μέγιστο ή το ελάχιστο, ανάλογα με το πρόβλημα που επιλύουμε.
    - **Απεριόριστο** (unbounded). Πρόκειται για προβλήματα που δεν έχουν βέλτιστη λύση αφού η αντικειμενική συνάρτηση τείνει προς το +∞, αν ψάχνουμε για μέγιστο ή τείνει προς το -∞, αν ψάχνουμε για ελάχιστο.

2.1.2 Ελλειμματικές και πλεονασματικές μεταβλητές

Ένα γραμμικό πρόβλημα σε κανονική μορφή, μπορεί να μετατραπεί σε τυποποιημένη μορφή, εάν εισάγουμε νέες μεταβλητές ώστε να μετατρέψουμε τις ανισότητες των περιορισμών σε ισότητες. Οι νέες αυτές μεταβλητές ονομάζονται **χαλαρές μεταβλητές** (slack variables).

Μία ανισότητα της μορφής:

$$\sum_{j=1}^{m} \alpha_{ij} x_j \leq b_i$$

μπορεί να μετατραπεί σε ισότητα, με τον πιο κάτω τρόπο:

$$\sum_{j=1}^{m} \alpha_{ij} x_j + s_i = b_i$$

Έχουμε δηλαδή την εισαγωγή μίας νέας χαλαρής μεταβλητής $s_i \geq 0$, η οποία ονομάζεται και **ελλειμματική μεταβλητή** (deficit variable).

Όμοια, μία ανισότητα της μορφής:

$$\sum_{j=1}^{m} \alpha_{ij} x_j \geq b_i$$

μπορεί να μετατραπεί σε ισότητα, με τον πιο κάτω τρόπο:

$$\sum_{j=1}^{m} \alpha_{ij} x_j - s_i = b_i$$

Η μεταβλητή αυτή $s_i \geq 0$, στην περίπτωση αυτή ονομάζεται και **πλεονασματική μεταβλητή** (surplus variable).

2.1.3 Εφαρμογές του Γραμμικού Προγραμματισμού

Ο γραμμικός προγραμματισμός έχει αποδειχθεί ότι είναι ένα πολύ χρήσιμο και θεμελιώδες μαθηματικό πρότυπο προγραμματισμού, το οποίο έχει μάλιστα άμεση εφαρμογή στην επίλυση πολλών προβλημάτων της πραγματικής ζωής αλλά και σε πολλούς άλλους τομείς των επιστημών (Murty 1983). Η χρησιμότητα των μεθόδων που έχουν αναπτυχθεί για την επίλυση γραμμικών προβλημάτων, έχει καταδειχθεί κατά τη διάρκεια των τελευταίων πενήντα και πλέον ετών. Έχουν αναπτυχθεί εφαρμογές σε τομείς όπως ο προγραμματισμός μεταφορών, η επιλογή χαρτοφυλακίων και η ανάλυση παραγωγικότητας (Park 1999). Υπάρχουν πάρα πολλές εφαρμογές των μεθόδων που έχουν αναπτυχθεί στα πλαίσια του γραμμικού προγραμματισμού, όπως οι εφαρμογές που φαίνονται πιο κάτω:

- Στρατιωτικές Εφαρμογές
- Βιομηχανικές Εφαρμογές
- Εφαρμογές υπολογιστών και επικοινωνιών
- Οικονομικές Εφαρμογές
- Μαθηματικές και στατιστικές εφαρμογές

2.2 Το Δυικό πρόβλημα

Για κάθε πρόβλημα γραμμικού προγραμματισμού, υπάρχει ένα ισοδύναμο πρόβλημα, το οποίο αναφέρεται ως **δυϊκό** (dual) πρόβλημα. Εάν το αρχικό πρόβλημα είναι πρόβλημα μεγιστοποίησης, τότε το δυϊκό του είναι πρόβλημα ελαχιστοποίησης και αντίστροφα. Τα δύο αυτά προβλήματα μπορούμε να πούμε ότι παρέχουν χρήσιμες πληροφορίες το ένα για το άλλο.

Η μετατροπή ενός προβλήματος στη δυϊκή του μορφή, μπορεί να γίνει με τη βοήθεια των κανόνων που εμφανίζονται στον πιο κάτω πίνακα (Πίνακας 2-1):

Πίνακας 2-1: Μετατροπή ενός προβλήματος στο δυϊκό του

Ελαχιστοποίηση	↔	**Μεγιστοποίηση**
min $c^T x$	↔	max $b^T w$
Περιορισμός =	↔	Ελεύθερη μεταβλητή
Περιορισμός ≥	↔	Μεταβλητή ≥ 0
Περιορισμός ≤	↔	Μεταβλητή ≤ 0
Ελεύθερη μεταβλητή	↔	Περιορισμός =
Μεταβλητή ≥ 0	↔	Περιορισμός ≤
Μεταβλητή ≤ 0	↔	Περιορισμός ≥

Για παράδειγμα, εάν έχουμε το παρακάτω γραμμικό πρόβλημα:

$$\min 5x_1 + 7x_2 + 2x_3 + 8x_4$$

μ.π.

$$x_1 - 2x_2 - 2x_3 + 3x_4 \leq 5$$

$$5x_1 - 2x_2 - x_3 + 5x_4 \leq 3$$

$$3x_1 - 2x_2 + 3x_3 - 3x_4 \leq -1$$

$$- x_2 + 3x_3 \leq -1$$

$$x_1, x_2, x_3, x_4 \geq 0$$

Εάν εφαρμόσουμε τους κανόνες του πίνακα 2-1, τότε θα προκύψει το δυϊκό του πρόβλημα:

$\max 5w_1 + 3w_2 - w_3 - w_4$

μ.π.

$w_1 + 5w_2 + 3w_3 \leq 5$

$-2w_1 - 2w_2 - 2w_3 - w_4 \leq 7$

$-2w_1 - w_2 + 3w_3 + 3w_4 \leq 2$

$3w_1 + 5w_2 - 3w_3 \leq 8$

$w_1, w_2, w_3, w_4 \leq 0$

Δηλαδή, στην νέα αντικειμενική συνάρτηση έχουμε ως συντελεστές τα στοιχεία του πίνακα $b = [5, 3, -1, -1]^T$. Στους περιορισμούς έχουμε ως συντελεστές τα στοιχεία του ανάστροφου πίνακα A^T του A. Επίσης, ως σταθεροί όροι στους περιορισμούς, εμφανίζονται τα στοιχεία του $c = [5, 7, 2, 8]^T$. Επειδή στο αρχικό πρόβλημα όλες οι ελεύθερες μεταβλητές είχαν τον περιορισμό ≥0, στο δυϊκό πρόβλημα όλοι οι περιορισμοί έχουν το ≤. Τέλος, επειδή στους περιορισμούς του αρχικού προβλήματος είχαμε το σύμβολο ≤, στο δυϊκό πρόβλημα για όλες τις νέες δυϊκές μεταβλητές, έχουμε τον περιορισμό να είναι ≤0.

Δηλαδή, κάθε περιορισμός σε ένα πρόβλημα αντιστοιχεί σε μια μεταβλητή του δυϊκού του προβλήματος και αντίστροφα. Εάν η αντικειμενική συνάρτηση ενός προβλήματος μεγιστοποιείται, τότε η αντικειμενική συνάρτηση του δυϊκού του προβλήματος ελαχιστοποιείται και αντίστροφα. Προφανώς, το δυϊκό του δυϊκού προβλήματος, είναι το αρχικό πρόβλημα.

Έχει αποδειχθεί ότι ισχύει πάντοτε η πιο κάτω ανισότητα:

$$c^T x \geq b^T w$$

και μάλιστα αναφέρεται ως ασθενές δυϊκό θεώρημα (weak duality theorem) (Chvatal 1983). Δηλαδή, η τιμή της αντικειμενικής συνάρτησης για το αρχικό πρόβλημα ελαχιστοποίησης, αποτελεί άνω φράγμα της τιμής της αντικειμενικής συνάρτησης του αντίστοιχου προβλήματος μεγιστοποίησης. Γενικά, εάν ένα πρωτεύον πρόβλημα έχει μια βέλτιστη λύση, τότε και το δυϊκό του πρόβλημα έχει επίσης μια βέλτιστη λύση και οι δύο

αυτές βέλτιστες τιμές των δύο προβλημάτων συμπίπτουν (ισχυρό δυϊκό θεώρημα). Η διαφορά μάλιστα ανάμεσα στις τιμές των δύο αντικειμενικών συναρτήσεων αναφέρεται και ως χάσμα δυϊκότητας (duality gap). Στον Πίνακας 2-2 εμφανίζεται η σχέση που έχει ένα πρόβλημα, ως προς την επιλυσιμότητά του, σε σχέση με το δυϊκό του. Αν ένα πρόβλημα είναι βέλτιστο, τότε είναι βέλτιστο και το δυϊκό του. Αν όμως ένα πρόβλημα είναι αδύνατο, τότε το δυϊκό του είναι είτε αδύνατο είτε απεριόριστο. Τέλος, αν ένα πρόβλημα είναι απεριόριστο, τότε το δυϊκό του είναι αδύνατο.

Πίνακας 2-2: Κατηγοριοποίηση πρωτεύοντος και δυϊκού προβλήματος

		Δυικό πρόβλημα		
		Βέλτιστο	Αδύνατο	Απεριόριστο
Πρωτεύον πρόβλημα	Βέλτιστο	√		
	Αδύνατο		√	√
	Απεριόριστο		√	

Για την επίλυση ενός προβλήματος επομένως, αντί να επιλύσουμε το αρχικό πρόβλημα, μπορούμε να δημιουργήσουμε το αντίστοιχο δυϊκό του και να επιλύσουμε αυτό. Ένας αλγόριθμος ο οποίος για να λύσει ένα πρόβλημα, το μετατρέπει στη δυϊκή του μορφή και επιλύει αυτή τη μορφή, αναφέρεται ως δυϊκός.

2.3 Αλγόριθμοι Γραμμικού Προγραμματισμού

2.3.1 Σχετική έρευνα

Η αρχή του Γραμμικού Προγραμματισμού χρονολογείται στα μέσα του προηγούμενου αιώνα, με την εφεύρεση της μεθόδου Simplex από τον Αμερικάνο επιστήμονα George Dantzig, ο οποίος ανέπτυξε τη μέθοδο αυτή για να επιλύσει προβλήματα προγραμματισμού στην Πολεμική Αεροπορία των Η.Π.Α. (Dantzig 1948). Η μέθοδος

Simplex εξακολουθεί ακόμη και σήμερα να παραμένει μία από τις πιο αποτελεσματικές μεθόδους για μια μεγάλη ποικιλία πρακτικών προβλημάτων.

Υπάρχει μεγάλη έρευνα που έχει πραγματοποιηθεί στον τομέα αυτόν και έχουν αναπτυχθεί διάφορες παραλλαγές της μεθόδου Simplex. Καμία από τις μεθόδους αυτές δεν έχει επιτύχει ακόμη πολυωνυμική πολυπλοκότητα χρόνου αφού μπορεί να απαιτηθεί υπολογιστική προσπάθεια η οποία να αυξάνεται εκθετικά σε σχέση με το πλήθος των δεδομένων του προβλήματος (Terlaky & Zhang 1993). Από θεωρητική άποψη επομένως, οι κλασικοί αλγόριθμοι τύπου Simplex, δεν θεωρούνται ιδιαίτερα αποτελεσματικοί, όμως στην πράξη πολύ συχνά επιδεικνύουν μεγάλη αποτελεσματικότητα και χρησιμότητα.

Ο πρώτος αλγόριθμος ο οποίος επέδειξε υπολογιστική πολυπλοκότητα χειρότερης περίπτωσης πολυωνυμικού χρόνου ως προς το μέγεθος προβλήματος, ήταν ένας αλγόριθμος εσωτερικών σημείων, που ονομάστηκε ελλειψοειδής αλγόριθμος (ellipsoid algorithm). Ο αλγόριθμο αυτός αναπτύχθηκε από τον Leonid Khachiyan το 1979 και έχει πολυπλοκότητα χρόνου τάξης $O(n^4L)$, όπου n είναι το πλήθος των μεταβλητών και L το μήκος της κωδικοποίησής τους σε δυαδικά ψηφία (Khachiyan 1979). Σε πρακτικό βέβαια επίπεδο, η απόδοση του αλγορίθμου αυτού τις περισσότερες φορές αποδεικνύεται κακή σε σχέση με την κλασική μέθοδο simplex. Παρ' όλα αυτά, ο ελλειψοειδής αλγόριθμος άνοιξε νέους ορίζοντες για την ανάπτυξη αποδοτικότερων αλγορίθμων εσωτερικών σημείων.

Το 1984, ο N. Karmarkar παρουσίασε έναν νέο αλγόριθμο εσωτερικών σημείων πολυωνυμικού χρόνου για προβλήματα Γραμμικού Προγραμματισμού, με πολυπλοκότητα της τάξης του $O(n^{3,5}L)$ (Karmarkar 1984). Σύμφωνα με το κλασικό άρθρο του Karmarkar, ο αλγόριθμος αυτός στην πράξη επιδεικνύει καλύτερα αποτελέσματα σε σχέση με τον αλγόριθμο Simplex. Από τότε, διάφορες παραλλαγές μεθόδων εσωτερικών σημείων έχουν αναπτυχθεί (Roos et al. 1997 & Wright 1997). Οι μέθοδοι εσωτερικών σημείων λειτουργούν πολύ καλά για διάφορα πρακτικά προβλήματα της πραγματικής ζωής και έχουν καλύτερη απόδοση σε σχέση με τη μέθοδο simplex, για προβλήματα μεγάλης κλίμακας (Lustig et al. 1994). Όμως, για μεσαία ή μικρού μεγέθους προβλήματα ή για κάποιες κατηγορίες προβλημάτων όπως η Δικτυακή βελτιστοποίηση, συχνά η μέθοδος Simplex έχει καλύτερη απόδοση σε σχέση με τις μεθόδους εσωτερικών σημείων. Επιπλέον, για τις μεθόδους εσωτερικών σημείων δεν

είναι εύκολος ο εντοπισμός μίας καλής αρχικής βασικής λύσης, ώστε να αποτελέσει το σημείο εκκίνησης για τον αλγόριθμο.

Εκτός από τους αλγόριθμους εσωτερικών σημείων, έκαναν την εμφάνισή τους και αλγόριθμοι των εξωτερικών σημείων. Ο πρώτος αλγόριθμος αυτής της μορφής, αναπτύχθηκε το 1991 από τον Κ. Παπαρρίζο για το πρόβλημα αντιστοίχισης (Paparrizos 1991). Αργότερα, ο Κ. Παπαρρίζος γενίκευσε την μέθοδο εξωτερικών σημείων έτσι ώστε να εφαρμόζεται στο γενικό γραμμικό πρόβλημα, αναπτύσσοντας έναν κατάλληλο αλγόριθμο δυϊκού τύπου (Paparrizos 1993).

Στον Πίνακας 2-3, εμφανίζονται οι κύριες υπολογιστικές βελτιώσεις για τους αλγόριθμους τύπου Simplex (Samaras 2001). Στον Πίνακας 2-4 εμφανίζεται η αντίστοιχη εξέλιξη για τους αλγόριθμους εσωτερικών σημείων (Samaras 2001).

Πίνακας 2-3: Υπολογιστικές βελτιώσεις αλγορίθμων τύπου Simplex

Μεθοδολογία	**Συγγραφέας**	**Χρονολογία**
Tableau μορφή	G. Dantzig	1947
Αναθεωρημένη μορφή	G. Dantzig W. Orchad-Hays P. Wolfe	1953-54
Παραγοντοποίηση Βάσης	H. Markowitz L. Beale E. Hellerman D. Rarick	1954 1971 1971 1972
Τεχνικές ανανέωσης αραιών μητρών	H. Bartels, H. Golub H. Forrest, A. Tomlin K. Reid	1969 1972 1976
Τεχνικές περιστροφής	J. Harris D. Goldfarb, K. Reid H. Forrest, D. Goldfarb	1973 1977 1992
Προλυτικές διαδικασίες	L. Brearly, G. Mitra, P. Williams	1975

Πίνακας 2-4: Ανάπτυξη αλγορίθμων εσωτερικών σημείων

Θεωρητική ανάλυση	**Συγγραφέας**	**Χρονολογία**
Αλγόριθμος λογαριθμικών φραγμών Μέθοδος των κέντρων Αφινικός Κλιμακωτός Αλγόριθμος	K. Frisch P. Huard I. Dikin	1955 1967 1974
Αλγόριθμος του Karmarkar	N. Karmarkar	1984
Ανάκτηση βέλτιστης λύσης	Y. Ye, M. Kojima	1987
Αλγόριθμος ιχνιλάτης	M. Kojima S. Mizuno A. Yoshise	1989
Ανέφικτοι Αλγόριθμοι	I. Lustig	1990
Αλγόριθμοι ελάττωσης δυναμικού	Y. Ye	1991
Θετικά ημιορισμένος προγραμματισμός	Y. Nesterov A. Nemirowski	1993

2.3.2 Αλγόριθμοι τύπου Simplex

Ένας αλγόριθμος γραμμικού προγραμματισμού τύπου Simplex, χρησιμοποιεί σε κάθε βήμα του, ένα υποσύνολο του συνόλου των μεταβλητών ως βάση. Για να αποτελεί ένα τέτοιο σύνολο B βάση του προβλήματος, θα πρέπει ο πίνακας A_B, ο οποίος προκύπτει από τον πίνακα A των συντελεστών των περιορισμών όταν κρατήσουμε μόνο τις αντίστοιχες στήλες, να είναι αντιστρέψιμος. Ισοδύναμα θα πρέπει ο πίνακας A_B, να είναι τάξης m, όπου m το πλήθος των περιορισμών. Οι υπόλοιπες μεταβλητές θεωρούμε ότι είναι μη βασικές και ανήκουν στον σύνολο N. Ένας αλγόριθμος τύπου Simplex κατασκευάζει με επαναληπτικό τρόπο μια πεπερασμένη ακολουθία τέτοιων βασικών λύσεων και εάν είναι η αντίστοιχη βασική λύση (x_B, x_N) βέλτιστη, ο αλγόριθμος σταματάει. Σε διαφορετική περίπτωση, ο αλγόριθμος βρίσκει δύο δείκτες $k \in B$ και $l \in N$ και οι αντίστοιχες μεταβλητές στις θέσεις k και l εναλλάσσουν τους ρόλους τους. Με άλλα λόγια, ο δείκτης k από βασικός γίνεται μη βασικός και αντίστροφα ο δείκτης l από μη βασικός γίνεται βασικός. Έχουμε δηλαδή τη διαδικασία της περιστροφής (pivoting)

ώστε να κατασκευαστεί η νέα βάση. Ο δείκτης l εισέρχεται στη νέα βάση και η αντίστοιχη μεταβλητή x_l ονομάζεται εισερχόμενη (entering), ενώ ο δείκτης k εξέρχεται από την βάση και η αντίστοιχη μεταβλητή x_k λέγεται εξερχόμενη (leaving). Η διαδικασία επαναλαμβάνεται μέχρις ότου βρεθεί η βέλτιστη λύση ή δεν είναι δυνατόν να βρεθούν οι κατάλληλοι δείκτες k και l.

Γενικά, δύο διαφορετικές βάσεις οι οποίες διαφέρουν μόνο σε έναν δείκτη, ονομάζονται *γειτονικές (adjacent) βάσεις*. Οι αλγόριθμοι τύπου simplex λειτουργούν κατασκευάζοντας μία πεπερασμένη ακολουθία γειτονικών βάσεων. Τα βήματα που ακολουθούν γενικότερα όλοι οι αλγόριθμοι τύπου Simplex εμφανίζονται αναλυτικά στον Πίνακας 2-5.

Πίνακας 2-5: Βήματα αλγορίθμων τύπου Simplex

Βήμα-0: Αρχικοποίηση (**Initialization**).

Υπολογισμός όλων των τιμών των μεταβλητών που είναι απαραίτητες για να αρχίσει να εκτελείται ο αλγόριθμος.

Βήμα-1: Έλεγχος βελτιστότητας (Test of Optimality).

Ελέγχεται αν το τρέχον βασικό σημείο αποτελεί τη βέλτιστη λύση. Αν είναι βέλτιστο, τότε ο αλγόριθμος σταματάει.

Βήμα-2: Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής.

Με κάποιο κανόνα περιστροφής επιλέγεται η εισερχόμενη και η εξερχόμενη μεταβλητή. Αν δεν μπορεί να γίνει η επιλογή αυτή, τότε το πρόβλημα είναι απεριόριστο.

Βήμα-3: Περιστροφή (Pivoting).

Στο βήμα αυτό υπολογίζεται η νέα βάση, οι τιμές των βασικών μεταβλητών καθώς και οι δυϊκές μεταβλητές, όπως και οι δυϊκές χαλαρές μεταβλητές. Η διαδικασία επαναλαμβάνεται από το Βήμα-1.

2.4 Πρωτεύων Αλγόριθμος Simplex

Ο πρωτεύων αλγόριθμος Simplex (primal simplex algorithm) αποτελεί τον πρώτο αλγόριθμο που δημιουργήθηκε για την επίλυση του Γραμμικού Προβλήματος. Ο αλγόριθμος χρησιμοποιεί κάποιες από τις μεταβλητές απόφασης ως βασικές μεταβλητές, ενώ τις υπόλοιπες τις θεωρεί ως μη βασικές μεταβλητές. Με τον τρόπο αυτό οι μεταβλητές χωρίζονται σε δύο σύνολα, το σύνολο B των βασικών μεταβλητών που περιέχει m μεταβλητές και το σύνολο N των μη βασικών μεταβλητών με n-m μέλη. Οι αντίστοιχοι υποπίνακες του πίνακα A των περιορισμών, συμβολίζονται με A_B και A_N και έχουν διαστάσεις mxm και (n-m)xm, αντίστοιχα. Ο πίνακας A_B πρέπει να είναι αντιστρέψιμος σε κάθε επανάληψη του αλγορίθμου. Η βασική λύση που αντιστοιχεί στη διαμέριση των μεταβλητών σε βασικές και μη βασικές συμβολίζεται με $x = [x_B \; x_N]$ και ισχύει:

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

$$x_N = 0$$

Η πιο πάνω λύση είναι εφικτή αν ικανοποιεί όλους τους περιορισμούς του προβλήματος, δηλαδή, εάν ισχύει $x_B \geq 0$.

Η λύση για το δυϊκό πρόβλημα, δίνεται από την σχέση

$$s = c - A^T w$$

όπου για τις δυϊκές μεταβλητές w έχουμε

$$w^T = {c_B}^T B^{-1}$$

Η βάση B θεωρείται ότι είναι δυϊκά εφικτή αν ισχυεί $s \geq 0$.

Μια βασική λύση είναι βέλτιστη αν ισχύει $x_B \geq 0$ και $s_N \geq 0$ και στην περίπτωση αυτή όλοι οι υπολογισμοί σταματούν, αφού έχει βρεθεί η βέλτιστη λύση. Σε διαφορετική περίπτωση, θα πρέπει να επιλεγεί μία στήλη από τον πίνακα A_B και μία από τον A_N, ώστε να γίνει ανταλλαγή του ρόλου τους, εκτελώντας έτσι μία περιστροφή. Αν δεν είναι δυνατόν να βρεθεί μεταβλητή να εξαχθεί από τη βάση B, τότε το πρόβλημα είναι απεριόριστο και οι ο αλγόριθμος σταματάει. Αν k ο δείκτης της εξερχόμενης μεταβλητής και l ο δείκτης της εισερχόμενης μεταβλητής, τότε έχουμε:

$$\mathrm{B_{new}} = \mathrm{B} - \{\mathrm{k}\} \cup \{\mathrm{l}\}$$

$$\mathrm{N_{new}} = \mathrm{N} - \{\mathrm{l}\} \cup \{\mathrm{k}\}$$

Δηλαδή κατά την περιστροφή (pivoting), η βασική μεταβλητή x_k είναι η εξερχόμενη μεταβλητή και θα γίνει μη βασική, ενώ αντίθετα η μη βασική x_l θεωρείται η εισερχόμενη μεταβλητή και θα γίνει βασική.

Πίνακας 2-6: Ο Πρωτεύων αλγόριθμος Simplex

Βήμα-0: Αρχικοποίηση (**Initialization**).

(α) Ξεκινάμε με μία εφικτή βασική διαμέριση (B, N)

(β) Υπολογίζουμε τις μήτρες και τα διανύσματα: c_B, c_N, A_B^{-1}, $x_B = A_B^{-1}b$, $w^T = c_B^T A_B^{-1}$, $s_N^T = c_N^T - w^T A_N$.

Βήμα-1: Έλεγχος βελτιστότητας (Test of Optimality).

Υπολογίζουμε το σύνολο

$$J = \{j: j \in N \text{ και } s_j < 0\}$$

Αν είναι $J = \varnothing$ τότε έχει βρεθεί η βέλτιστη λύση και ο αλγόριθμος σταματάει.

Βήμα-2: Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής.

(α) **Επιλογή εισερχόμενης μεταβλητής**. Με κάποιο κανόνα περιστροφής επιλέγεται ένα στοιχείο του J, που αντιστοιχεί στη θέση t του συνόλου N και προσδιορίζει την εισερχόμενη μεταβλητή x_l. Δηλαδή $l = N(t)$.

(β) **Επιλογή εξερχόμενης μεταβλητής**. Υπολογίζουμε τον mx1 πίνακα

$$h_l = (A_B)^{-1} a_l$$

όπου a_l η l στήλη του πίνακα A και το σύνολο

$$I_+ = \{i: 1 \le i \le m \text{ και } h_{il} > 0\}$$

Αν είναι $I_+ = \varnothing$ τότε το πρόβλημα είναι **απεριόριστο** και ο αλγόριθμος σταματάει.

Έλεγχος ελαχίστου λόγου:

Διαφορετικά επιλέγεται δείκτης r που αντιστοιχεί στον ελάχιστο λόγο:

$$\frac{(A_B{}^{-1}b)_r}{h_{ir}} = \min\{\frac{(A_B{}^{-1}b)_i}{h_{il}} : i \in I_+\}$$

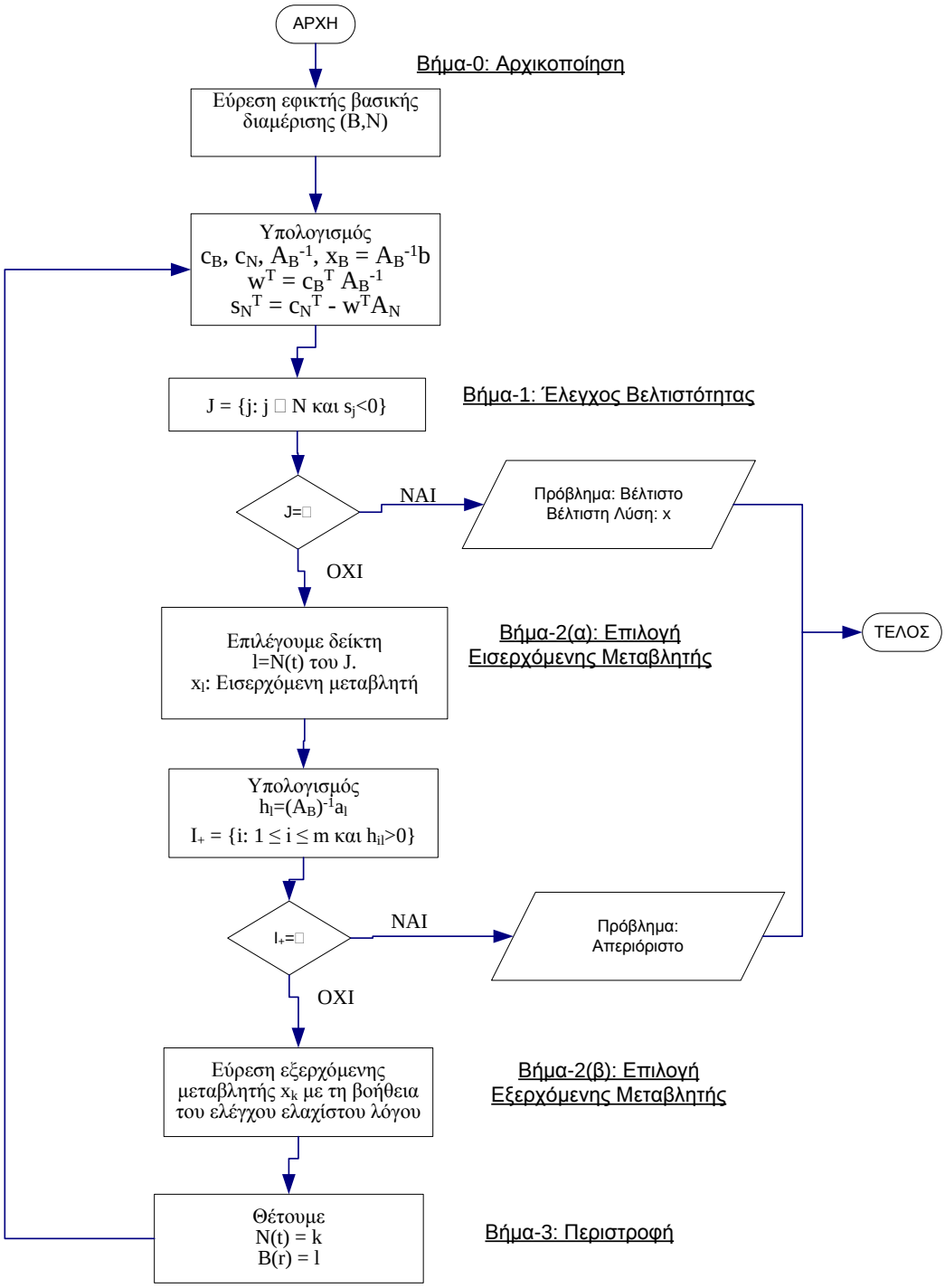
Ο δείκτης r του ελάχιστου λόγου, προσδιορίζει τη θέση στο σύνολο B για την εξερχόμενη μεταβλητή x_k. Δηλαδή $k = B(r)$.

Βήμα-3: Περιστροφή (Pivoting).

Εναλλάσσεται ο ρόλος των μεταβλητών x_l και x_k, θέτοντας $N(t) = k$ και $B(r) = l$ για να υπολογιστεί η νέα βασική διαμέριση.

Ο αλγόριθμος συνεχίζεται από το Βήμα-0 (β).

Σε κάθε επανάληψη, ο πρωτεύον αλγόριθμος Simplex, και γενικότερα οι αλγόριθμοι τύπου simplex, εναλλάσσουν μια μεταβλητή του συνόλου βασικών μεταβλητών B με μία από το σύνολο μη βασικών μεταβλητών N. Στον Πίνακας 2-6 εμφανίζονται τα αναλυτικά βήματα του πρωτεύοντος αλγόριθμου Simplex. Στην Εικόνα 2-3 εμφανίζεται ένα διάγραμμα ροής που περιγράφει τη λειτουργία του πρωτεύοντος αλγορίθμου Simplex.



Εικόνα 2-3: Διάγραμμα ροής για τον Πρωτεύοντα αλγόριθμο Simpex

2.4.1 Εφαρμογή Πρωτεύοντος Αλγορίθμου Simplex

Στην παράγραφο αυτή γίνεται περιγραφή του Πρωτεύοντος αλγορίθμου Simplex (primal simplex algorithm) δίνοντας τα βήματα επίλυσης ενός συγκεκριμένου Γραμμικού Προβλήματος. Τα βήματα του αλγορίθμου παρουσιάζονται αναλυτικά, για κάθε

επανάληψη, μέχρι να φτάσουμε στη βέλτιστη λύση. Θεωρούμε το πιο κάτω πρόβλημα Γραμμικού Προγραμματισμού:

max z = $x_1 + x_2$

μ.π. $2x_1 + x_2 \leq 4$

$x_1 + 2x_2 \leq 4$

$x_1, x_2 \geq 0$

Το πρόβλημα μετασχηματίζεται ως πρόβλημα ελαχιστοποίησης, όπως περιγράφτηκε στις προηγούμενες παραγράφους, και παίρνει την πιο κάτω τυποποιημένη μορφή, αφού πρώτα προστεθούν νέες μεταβλητές x_3 και x_4:

-min $-x_1 - x_2$

μ.π. $2x_1 + x_2 + x_3 = 4$

$x_1 + 2x_2 + x_4 = 4$

$x_1, x_2, x_3, x_4 \geq 0$

Οπότε,

$$A = \begin{bmatrix} 2 & 1 & 1 & 0 \\ 1 & 2 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 4 \\ 4 \end{bmatrix}, c = \begin{bmatrix} -1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

Θεωρούμε τη διαμέριση (B, N) όπου B=(3,4) και N=(1,2), δηλαδή θεωρούμε ότι οι μεταβλητές x_3 και x_4 είναι βασικές μεταβλητές (**Βήμα-0(α)**). Στην ουσία, ο αλγόριθμος ξεκινάει από την κορυφή A του σχήματος στην Εικόνα 2-4, δηλαδή την αρχή των αξόνων O(0,0), αφού θεωρούμε ότι οι μη βασικές μεταβλητές x_1 και x_2 έχουν την τιμή 0.

Επομένως κάνουμε τους υπολογισμούς (**Βήμα-0(β)**)

$$A_B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, A_N = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, c_B = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$

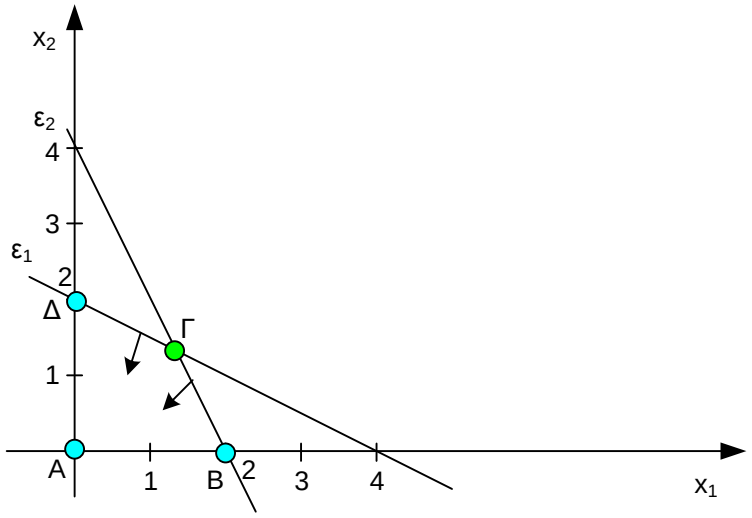
$$A_B{}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$w^T = c_B{}^T A_B{}^{-1} = [0 \quad 0] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [0 \quad 0]$$

(*Δυϊκές μεταβλητές*)

$$s_N{}^T = c_N{}^T - w^T A_N = [-1 \quad -1] - [0 \quad 0]\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = [-1 \quad -1]$$

(*Δυϊκές χαλαρές μεταβλητές*)



Εικόνα 2-4: Γραφική παράσταση Πρωτεύοντος Αλγορίθμου Simplex

Επειδή υπάρχουν αρνητικές δυϊκές χαλαρές μεταβλητές, το σύνολο J δεν είναι κενό. Επομένως, με βάση τον έλεγχο βελτιστότητας (**Βήμα-1**), θα πρέπει ο αλγόριθμος να συνεχίσει την εκτέλεσή του.

Το σύνολο J ισούται με J = {1, 2} αφού $s_1 = -1 < 0$ και $s_2 = -1 < 0$.

Επιλέγουμε τη μεταβλητή x_1 ως εισερχόμενη, δηλαδή l=1 (**Βήμα-2(α)**).

Στην συνέχεια γίνεται ο έλεγχος ελαχίστου λόγου:

$$x_B = A_B{}^{-1} b = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix}$$

$$h_l = A_B{}^{-1} a_l = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

Άρα,

$$I_+ = \{1, 2\}$$

αφού $h_{1l} = 2 > 0$ και $h_{2l} = 1 > 0$.

Έχουμε

$$min\left\{\frac{4}{2}, \frac{4}{1}\right\} = \frac{4}{2}$$

Έχουμε το ελάχιστο στη θέση r=1 και άρα η εξερχόμενη μεταβλητή θα είναι η x_3 (**Βήμα-2(β)**). Επομένως, εκτελείται η περιστροφή (**Βήμα-3**) και για την επόμενη επανάληψη θα έχουμε B = {1, 4} και N = {2, 3}. Στην ουσία έχουμε ως βασική λύση την κορυφή B της Εικόνα 2-4.

Επανάληψη-2

Επαναλαμβάνονται και πάλι οι υπολογισμοί (**Βήμα-0(β)**):

$$A_B = \begin{bmatrix} 2 & 0 \\ 1 & 1 \end{bmatrix}, A_N = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}, c_B = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

$$A_B{}^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$w^T = c_B{}^T A_B{}^{-1} = [-1 \quad 0] \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & 0 \end{bmatrix}$$

$$s_N{}^T = c_N{}^T - w^T A_N = [-1 \quad 0] - \begin{bmatrix} -\frac{1}{2} & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

Με βάση τον έλεγχο βελτιστότητας (**Βήμα-1**), το σύνολο J ισούται με J = {1} αφού είναι $s_1 < 0$ και επομένως ο αλγόριθμος συνεχίζει. Η μεταβλητή x_2 που αντιστοιχεί στη θέση 1 του συνόλου N={2, 3} είναι η εισερχόμενη. Δηλαδή l=2 (**Βήμα-2(α)**).
Στην συνέχεια γίνεται ο έλεγχος ελαχίστου λόγου:

$$x_B = A_B{}^{-1} b = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$h_l = A_B{}^{-1} a_l = \begin{bmatrix} \frac{1}{2} & 0 \\ -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 3/2 \end{bmatrix}$$

Οπότε,

$$I_+ = \{1, 2\}$$

αφού $h_{1l} = 1/2 > 0$ και $h_{2l} = 3/2 > 0$.

Έχουμε

$$min\left\{\frac{2}{\frac{1}{2}}, \frac{2}{\frac{3}{2}}\right\} = min\left\{4, \frac{4}{3}\right\} = \frac{4}{3}$$

Έχουμε το ελάχιστο στη θέση r = 2 και άρα η εξερχόμενη μεταβλητή θα είναι η x_4 (**Βήμα-2(β)**). Επομένως, εκτελείται η περιστροφή (**Βήμα-3**) και για την επόμενη επανάληψη θα έχουμε B = {1, 2} και N = {3, 4}, δηλαδή έχουμε ως βασική λύση την κορυφή Γ της Εικόνα 2-4.

Επανάληψη-3

Επαναλαμβάνονται και πάλι οι υπολογισμοί (**Βήμα-0(β)**):

$$A_B = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, A_N = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, c_B = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, c_N = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$A_B{}^{-1} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 4/3 \\ 4/3 \end{bmatrix}$$

$$w^T = c_B{}^T A_B{}^{-1} = \begin{bmatrix} -1 & -1 \end{bmatrix} \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix} = \begin{bmatrix} -\frac{1}{3} & -\frac{1}{3} \end{bmatrix}$$

$$s_N{}^T = c_N{}^T - w^T A_N = \begin{bmatrix} 0 & 0 \end{bmatrix} - \begin{bmatrix} -\frac{1}{3} & -\frac{1}{3} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} \end{bmatrix}$$

Με βάση τον έλεγχο βελτιστότητας (**Βήμα-1**), το σύνολο J ισούται με J = ∅ αφού είναι $s_1 \geq 0$ και $s_2 \geq 0$. Επομένως ο αλγόριθμος σταματάει αφού έχει βρεθεί η βέλτιστη λύση η οποία είναι:

$$x_1 = \frac{4}{3}, x_2 = \frac{4}{3},\ x_3 = 0,\ x_4 = 0$$

με τιμή αντικειμενικής συνάρτησης

$$z = x_1 + x_2 = \frac{4}{3} + \frac{4}{3} = \frac{8}{3}$$

η οποία αντιστοιχεί στη μέγιστη τιμή που μπορεί να έχουμε για την αντικειμενική συνάρτηση στο αρχικό πρόβλημα Γραμμικού Προγραμματισμού.

2.5 Αναθεωρημένος Αλγόριθμος Simplex

Ο αναθεωρημένος αλγόριθμος Simplex (revised simplex algorithm) αποτελεί μία παραλλαγή της μεθόδου simplex με τη βοήθεια της οποίας δεν χρειάζεται σε κάθε επανάληψη του αλγορίθμου ο υπολογισμός της αντίστροφης μήτρας A_B^{-1} από την αρχή. Ο υπολογισμός αυτός είναι εξαιρετικά χρονοβόρος και η υπολογιστική προσπάθεια μπορεί να φτάσει στο 80% του συνολικού χρόνου εκτέλεσης του αλγορίθμου. Η αναθεωρημένη μέθοδος Simplex υπολογίζει την αντίστροφη μήτρα A_B^{-1} μόνο κατά την πρώτη επανάληψη και στις επόμενες επαναλήψεις υπολογίζει τον πίνακα με πιο αποδοτικό τρόπο, κάνοντας χρήση των αποτελεσμάτων που έχουν υπολογιστεί στη διάρκεια της επανάληψης. Επομένως, η αναθεωρημένη μέθοδος simplex είναι υπολογιστικά αρκετά πιο αποδοτική μέθοδος, ειδικά για τα μεγάλα και αραιά προβλήματα, αφού ο υπολογισμός μίας αντίστροφης μήτρας, αποτελεί μία επίπονη διαδικασία.

Για τον υπολογισμό της μήτρας A_B^{-1}, ο αναθεωρημένος αλγόριθμος Simplex χρησιμοποιεί τον τύπο:

$$(A_B^{-1})_{new} = E^{-1} A_B^{-1}$$

όπου $(A_B^{-1})_{new}$ είναι ο νέος αντίστροφος πίνακας που θα χρησιμοποιηθεί κατά την επόμενη επανάληψη και ο πίνακας E^{-1} έχει τη μορφή που φαίνεται πιο κάτω, όπου ο πίνακας h_l, έχει τη μορφή που υπολογίζεται από το (Βήμα-2(β)) του πρωτεύοντος αλγορίθμου Simplex (Πίνακας 2-6):

$$E^{-1} = \begin{bmatrix} 1 & 0 & \cdots & -\frac{h_{1l}}{h_r} & \dots & 0 \\ 0 & 1 & \cdots & -\frac{h_{2l}}{h_r} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & \frac{1}{h_r} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -\frac{h_{ml}}{h_r} & \dots & 1 \end{bmatrix}$$

Πίνακας 2-7: Ο Αναθεωρημένος αλγόριθμος Simplex

Βήμα-0: Αρχικοποίηση (**Initialization**).

(α) Ξεκινάμε με μία εφικτή βασική διαμέριση (B, N).

Υπολογίζουμε την αντίστροφή μήτρα A_B^{-1}

(β) Υπολογίζουμε τις μήτρες: c_B, c_N, $x_B = A_B^{-1}b$, $w^T = c_B^T A_B^{-1}$, $s_N^T = c_N^T - w^T A_N$.

Βήμα-1: Έλεγχος βελτιστότητας (Test of Optimality).

Υπολογίζουμε το σύνολο

$$J = \{j: j \in N \text{ και } s_j < 0\}$$

Αν είναι $J = \varnothing$ τότε έχει βρεθεί η βέλτιστη λύση και ο αλγόριθμος σταματάει.

Βήμα-2: Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής.

(α) **Επιλογή εισερχόμενης μεταβλητής**. Με κάποιο κανόνα περιστροφής επιλέγεται ένα στοιχείο του J, που αντιστοιχεί στη θέση t του συνόλου N και προσδιορίζει την εισερχόμενη μεταβλητή x_l. Δηλαδή l = N(t).

(β) **Επιλογή εξερχόμενης μεταβλητής**. Υπολογίζουμε τον mx1 πίνακα

$$h_l=(A_B)^{-1}a_l$$

όπου a_l η l στήλη του πίνακα A και το σύνολο

$$I_+ = \{i: 1 \le i \le m \text{ και } h_{il}>0\}$$

Αν είναι $I_+ = \varnothing$ τότε το πρόβλημα είναι **απεριόριστο** και ο αλγόριθμος σταματάει.

Έλεγχος ελαχίστου λόγου:

Διαφορετικά επιλέγεται δείκτης r που αντιστοιχεί στον ελάχιστο λόγο:

$$\frac{(A_B{}^{-1}b)_r}{h_{ir}} = \min\{\frac{(A_B{}^{-1}b)_i}{h_{il}} : i \in I_+\}$$

Ο δείκτης r του ελάχιστου λόγου, προσδιορίζει τη θέση στο σύνολο B για την εξερχόμενη μεταβλητή x_k. Δηλαδή k = B(r).

Βήμα-3: Περιστροφή (Pivoting).

Εναλλάσσεται ο ρόλος των μεταβλητών x_l και x_k, θέτοντας N(t) = k και B(r)=l.

Ανανέωση τιμής αντίστροφης μήτρας A_B^{-1}:

$$(A_B^{-1})_{new} = E^{-1} A_B^{-1}$$

Ο αλγόριθμος συνεχίζεται από το Βήμα-0(β).

Ο αναθεωρημένος αλγόριθμος Simplex έχει επομένως τη μορφή που φαίνεται στον Πίνακας 2-7. Παρόλο που ο αναθεωρημένος αλγόριθμος Simplex είναι υπολογιστικά καλύτερος σε σχέση με τον κλασικό αλγόριθμο Simplex, οι δύο αλγόριθμοι παραμένουν μαθηματικά ισοδύναμοι, σε σχέση με την τάξη πολυπλοκότητας όπου ανήκουν. Στη συνέχεια παρουσιάζεται η δυϊκή μορφή του αλγόριθμου Simplex.

2.6 Δυϊκός Αλγόριθμος Simplex

Ο Δυϊκός αλγόριθμος Simplex (dual simplex algorithm) μοιάζει στην λειτουργία του με τον Πρωτεύοντα αλγόριθμο Simplex ως προς τη λειτουργία του. Η διαφορά είναι ότι ο Δυϊκός αλγόριθμος Simplex ξεκινά από μία δυϊκά εφικτή αρχική διαμέριση του συνόλου των μεταβλητών και υπολογίζει πρώτα την εξερχόμενη και μετά την εισερχόμενη μεταβλητή. Τα βήματα του αλγόριθμου φαίνονται στον Πίνακας 2-8. Στην Εικόνα 2-5 εμφανίζεται ένα διάγραμμα ροής για τον Δυϊκό αλγόριθμο Simplex.

Πίνακας 2-8: Ο Δυϊκός αλγόριθμος Simplex

Βήμα-0: Αρχικοποίηση (**Initialization**).

(α) Ξεκινάμε με μία δυϊκά εφικτή βασική διαμέριση (B, N)

(β) Υπολογίζουμε τις μήτρες: c_B, c_N, A_B^{-1}, $x_B = A_B^{-1}b$, $w^T = c_B^T A_B^{-1}$, $s_N^T = c_N^T - w^T A_N$.

Βήμα-1: Έλεγχος βελτιστότητας (Test of Optimality).

Αν είναι $x_B \geq 0$ τότε έχει βρεθεί η βέλτιστη λύση και ο αλγόριθμος σταματάει.

Βήμα-2: Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής.

(α) **Επιλογή εξερχόμενης μεταβλητής**. Επιλέγεται μία θέση r του συνόλου B που αντιστοιχεί στη βασική μεταβλητή x_k με $x_k<0$ η οποία προσδιορίζει την εξερχόμενη μεταβλητή x_k. Δηλαδή k = B(r).

(β**) Επιλογή εισερχόμενης μεταβλητής**. Υπολογίζουμε τον mx1 πίνακα

$$H_r = (A_B)^{-1}{}_r A_N$$

όπου $=(A_B)^{-1}{}_r$ η r γραμμή του πίνακα $=(A_B)^{-1}$

Αν είναι $H_{rN} \geq 0$ τότε το δυϊκό πρόβλημα είναι απεριόριστο, ενώ το πρωτεύον πρόβλημα είναι **αδύνατο**.

Διαφορετικά επιλέγεται η μη βασική μεταβλητή x_l για την οποία:

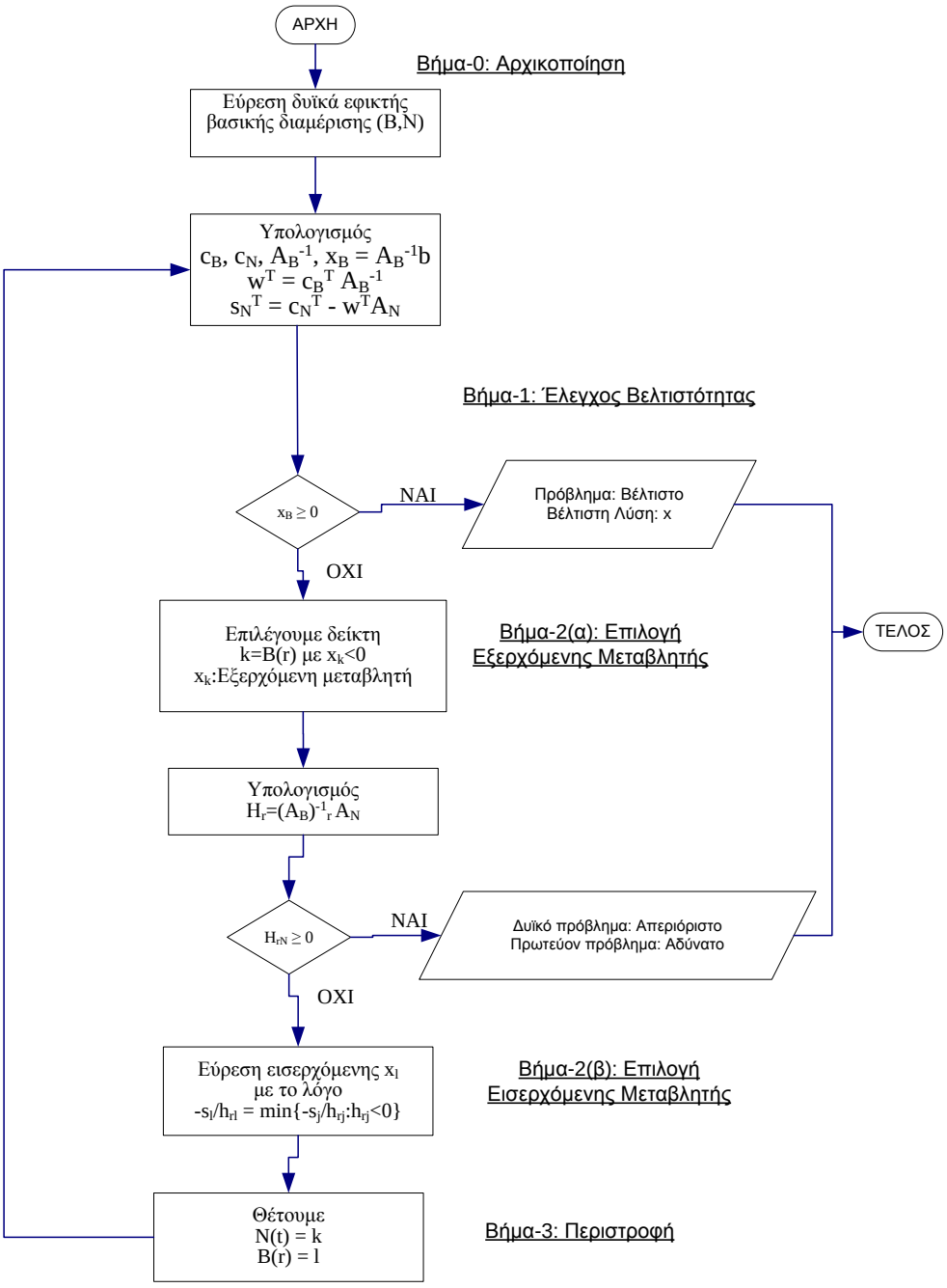
$$\frac{-s_l}{h_{rl}} = \min\{\frac{-s_j}{h_{rj}} : h_{rj} < 0\}$$

Η μεταβλητή αυτή είναι η εισερχόμενη και θεωρούμε l = N(t).

Βήμα-3: Περιστροφή (Pivoting).

Εναλλάσσεται ο ρόλος των μεταβλητών x_l και x_k, θέτοντας N(t) = k και B(r)=l.

Ο αλγόριθμος συνεχίζεται από το Βήμα-0(β).



Εικόνα 2-5:Διάγραμμα ροής για τον Δυϊκό αλγόριθμο Simpex

2.6.1 Εφαρμογή Δυϊκού Αλγορίθμου **Simplex**

Στην παράγραφο αυτή γίνεται περιγραφή του Δυϊκού αλγορίθμου Simplex (dual simplex algorithm) δίνοντας τα βήματα επίλυσης ενός συγκεκριμένου Γραμμικού Προβλήματος. Τα βήματα του αλγορίθμου παρουσιάζονται αναλυτικά, για κάθε επανάληψη, μέχρι να φτάσουμε στη βέλτιστη λύση.

Θεωρούμε το πιο κάτω πρόβλημα Γραμμικού Προγραμματισμού:

$$\min z = 5x_1 + 7x_2 + 2x_3 + 8x_4$$

$$\text{μ.π.} \quad x_1 - 2x_2 - 2x_3 + 3x_4 \leq 5$$

$$5x_1 - 2x_2 - x_3 + 5x_4 \leq 3$$

$$3x_1 - 2x_2 + 3x_3 - 3x_4 \leq -1$$

$$-x_2 + 3x_3 \leq -1$$

$$x_1, x_2, x_3, x_4 \geq 0$$

Το πρόβλημα παίρνει την πιο κάτω τυποποιημένη μορφή, αφού πρώτα προστεθούν νέες μεταβλητές x_5, x_6, x_7 και x_8:

$$\min z = 5x_1 + 7x_2 + 2x_3 + 8x_4$$

$$\text{μ.π.} \quad x_1 - 2x_2 - 2x_3 + 3x_4 + x_5 = 5$$

$$5x_1 - 2x_2 - x_3 + 5x_4 + x_6 = 3$$

$$3x_1 - 2x_2 + 3x_3 - 3x_4 + x_7 = -1$$

$$-x_2 + 3x_3 + x_8 = -1$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8 \geq 0$$

Άρα,

$$A = \begin{bmatrix} 1 & -2 & -2 & 3 & 1 & 0 & 0 & 0 \\ 5 & -2 & -1 & 5 & 0 & 1 & 0 & 0 \\ 3 & -2 & 3 & -3 & 0 & 0 & 1 & 0 \\ 0 & -1 & 3 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, b = \begin{bmatrix} 5 \\ 3 \\ -1 \\ -1 \end{bmatrix}, c = \begin{bmatrix} 5 \\ 7 \\ 2 \\ 8 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Θεωρούμε τη διαμέριση (B, N) όπου B=(5, 6, 7, 8) και N=(1, 2, 3, 4), δηλαδή θεωρούμε ότι οι μεταβλητές x_5, x_6, x_7 και x_8 είναι βασικές μεταβλητές (**Βήμα-0(α)**) και κάνουμε τους υπολογισμούς (**Βήμα-0(β)**):

$$A_B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_N = \begin{bmatrix} 1 & -2 & -2 & 3 \\ 5 & -2 & -1 & 5 \\ 3 & -2 & 3 & -3 \\ 0 & -1 & 3 & 0 \end{bmatrix}, c_B = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} 5 \\ 7 \\ 2 \\ 8 \end{bmatrix}$$

$$A_B{}^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 5 \\ 3 \\ -1 \\ -1 \end{bmatrix}$$

$$w^T = c_B{}^T A_B{}^{-1} = [0 \quad 0 \quad 0 \quad 0] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = [0 \quad 0 \quad 0 \quad 0]$$

$$s_N{}^T = c_N{}^T - w^T A_N = [5 \quad 7 \quad 2 \quad 8] - [0 \quad 0 \quad 0 \quad 0] \begin{bmatrix} 1 & -2 & -2 & 3 \\ 5 & -2 & -1 & 5 \\ 3 & -2 & 3 & -3 \\ 0 & -1 & 3 & 0 \end{bmatrix}$$

$$= [5 \quad 7 \quad 2 \quad 8]$$

Η λύση x είναι μη εφικτή επειδή υπάρχουν αρνητικές τιμές στον πίνακα x_B. Αντίθετα, η λύση είναι δυϊκά εφικτή αφού είναι $s \geq 0$. Αν η λύση ήταν εφικτή τόσο για το πρωτεύον όσο και για το δυϊκό πρόβλημα, τότε ο αλγόριθμος θα σταματούσε αφού θα είχε βρεθεί η βέλτιστη λύση.

Επιλέγουμε την μεταβλητή x_7, για την οποία ισχύει $x_7<0$, ως εξερχόμενη μεταβλητή. Δηλαδή k=7 και r=3 (**Βήμα-2(α)**).

Οπότε,

$$H_3 = (A_B^{-1})_3 A_N = [0 \quad 0 \quad 1 \quad 0] \begin{bmatrix} 1 & -2 & -2 & 3 \\ 5 & -2 & -1 & 5 \\ 3 & -2 & 3 & -3 \\ 0 & -1 & 3 & 0 \end{bmatrix} = [3 \quad -2 \quad 3 \quad -3]$$

Υπολογίζουμε το ελάχιστο των λόγων:

$$min\left\{\frac{-7}{-2}, \frac{-8}{-3}\right\} = \frac{8}{3}$$

Άρα εισερχόμενη μεταβλητή είναι η x_4, η οποία είναι στη θέση 4 του συνόλου N. Δηλαδή έχουμε l=4 και t=4 (**Βήμα-2(β)**).

Εκτελώντας την περιστροφή, προκύπτει ότι έχουμε τις βασικές μεταβλητές B={4, 5, 6, 8} και τις μη βασικές μεταβλητές N={1, 2, 3, 7} (**Βήμα-3**).

Επανάληψη-2

Επαναλαμβάνονται και πάλι οι υπολογισμοί (**Βήμα-0(β)**):

$$A_B = \begin{bmatrix} 3 & 1 & 0 & 0 \\ 5 & 0 & 1 & 0 \\ -3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, A_N = \begin{bmatrix} 1 & -2 & -2 & 0 \\ 5 & -2 & -1 & 0 \\ 3 & -2 & 3 & 1 \\ 0 & -1 & 3 & 0 \end{bmatrix}, c_B = \begin{bmatrix} 8 \\ 0 \\ 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} 5 \\ 7 \\ 2 \\ 0 \end{bmatrix}$$

$$A_B{}^{-1} = \begin{bmatrix} 0 & 0 & -\frac{1}{3} & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & \frac{5}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} 0 & 0 & -\frac{1}{3} & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & \frac{5}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 4 \\ 4/3 \\ -1 \end{bmatrix}$$

Η λύση x είναι μη εφικτή λύση επειδή υπάρχουν αρνητικές τιμές στον πίνακα x_B.

$$w^T = c_B{}^T A_B{}^{-1} = [8 \quad 0 \quad 0 \quad 0] \begin{bmatrix} 0 & 0 & -\frac{1}{3} & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & \frac{5}{3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\frac{8}{3} & 0 \end{bmatrix}$$

$$s_N{}^T = c_N{}^T - w^T A_N = [5 \quad 7 \quad 2 \quad 0] - \begin{bmatrix} 0 & 0 & -\frac{8}{3} & 0 \end{bmatrix} \begin{bmatrix} 1 & -2 & -2 & 0 \\ 5 & -2 & -1 & 0 \\ 3 & -2 & 3 & 1 \\ 0 & -1 & 3 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 13 & \frac{5}{3} & 10 & \frac{8}{3} \end{bmatrix}$$

Η μεταβλητή x_8 επιλέγεται ως εξερχόμενη μεταβλητή με r=4.

Οπότε,

$$H_4 = (A_B^{-1})_4 \, A_N = [0 \quad 0 \quad 0 \quad 1] \begin{bmatrix} 1 & -2 & -2 & 0 \\ 5 & -2 & -1 & 0 \\ 3 & -2 & 3 & 1 \\ 0 & -1 & 3 & 0 \end{bmatrix} = [0 \quad -1 \quad 3 \quad 0]$$

Υπολογίζουμε το ελάχιστο των λόγων:

$$min \left\{ \begin{matrix} -\frac{5}{3} \\ -1 \end{matrix} \right\} = \frac{5}{3}$$

Οπότε, εισερχόμενη μεταβλητή είναι η x_2, η οποία είναι στη θέση 2 του συνόλου N. Δηλαδή έχουμε l=2 και t=2 (**Βήμα-2(β)**).

Εκτελώντας την περιστροφή, προκύπτει ότι έχουμε τις βασικές μεταβλητές B={2, 4, 5, 6} και τις μη βασικές μεταβλητές N={1, 3, 7, 8} (**Βήμα-3**).

Επανάληψη-3

Επαναλαμβάνονται και πάλι οι υπολογισμοί (**Βήμα-0(β)**):

$$A_B = \begin{bmatrix} -2 & 3 & 1 & 0 \\ -2 & 5 & 0 & 1 \\ -2 & -3 & 0 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}, A_N = \begin{bmatrix} 1 & -2 & 0 & 0 \\ 5 & -1 & 0 & 0 \\ 3 & 3 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix}, c_B = \begin{bmatrix} 7 \\ 8 \\ 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} 5 \\ 2 \\ 0 \\ 0 \end{bmatrix}$$

$$A_B{}^{-1} = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1/3 & 2/3 \\ 1 & 0 & 1 & -4 \\ 0 & 1 & 5/3 & -16/3 \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1/3 & 2/3 \\ 1 & 0 & 1 & -4 \\ 0 & 1 & 5/3 & -16/3 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ -1 \\ -1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \\ -1/3 \\ 8 \\ 20/3 \end{bmatrix}$$

Η λύση x είναι μη εφικτή λύση επειδή υπάρχουν αρνητικές τιμές στον πίνακα x_B.

$$w^T = c_B{}^T A_B{}^{-1} = [7 \quad 8 \quad 0 \quad 0] \begin{bmatrix} 0 & 0 & 0 & -1 \\ 0 & 0 & -1/3 & 2/3 \\ 1 & 0 & 1 & -4 \\ 0 & 1 & 5/3 & -16/3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\frac{8}{3} & -\frac{5}{3} \end{bmatrix}$$

$$s_N{}^T = c_N{}^T - w^T A_N = [5 \quad 2 \quad 0 \quad 0] - \begin{bmatrix} 0 & 0 & -\frac{8}{3} & -\frac{5}{3} \end{bmatrix} \begin{bmatrix} 1 & -2 & 0 & 0 \\ 5 & -1 & 0 & 0 \\ 3 & 3 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 13 & 15 & \frac{8}{3} & \frac{5}{3} \end{bmatrix}$$

Η μεταβλητή x_4 επιλέγεται ως εξερχόμενη μεταβλητή με r=2.

Άρα,

$$H_2 = (A_B^{-1})_2 \, A_N = \begin{bmatrix} 0 & 0 & -\frac{1}{3} & \frac{2}{3} \end{bmatrix} \begin{bmatrix} 1 & -2 & 0 & 0 \\ 5 & -1 & 0 & 0 \\ 3 & 3 & 1 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -\frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

Υπολογίζουμε το ελάχιστο των λόγων:

$$min \left\{ \frac{-13}{-1}, \frac{-\frac{8}{3}}{\frac{-1}{3}} \right\} = 8$$

Άρα εισερχόμενη μεταβλητή είναι η x_7, η οποία είναι στη θέση 3 του συνόλου N. Δηλαδή έχουμε l=7 και t=3 (**Βήμα-2(β)**).

Εκτελώντας την περιστροφή, προκύπτει ότι έχουμε τις βασικές μεταβλητές B={2, 5, 6, 7} και τις μη βασικές μεταβλητές N={1, 3, 4, 8} (**Βήμα-3**).

Επανάληψη-4

Επαναλαμβάνονται και πάλι οι υπολογισμοί (**Βήμα-0(β)**):

$$A_B = \begin{bmatrix} -2 & 1 & 0 & 0 \\ -2 & 0 & 1 & 0 \\ -2 & 0 & 0 & 1 \\ -1 & 0 & 0 & 0 \end{bmatrix}, A_N = \begin{bmatrix} 1 & -2 & 3 & 0 \\ 5 & -1 & 5 & 0 \\ 3 & 3 & -3 & 0 \\ 0 & 3 & 0 & 1 \end{bmatrix}, c_B = \begin{bmatrix} 7 \\ 0 \\ 0 \\ 0 \end{bmatrix}, c_N = \begin{bmatrix} 5 \\ 2 \\ 8 \\ 0 \end{bmatrix}$$

$$A_B{}^{-1} = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -2 \end{bmatrix}, x_B = A_B{}^{-1} b = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & -2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & -2 \end{bmatrix} \begin{bmatrix} 5 \\ 3 \\ -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 7 \\ 5 \\ 1 \end{bmatrix}$$

Η λύση x είναι εφικτή λύση επειδή δεν υπάρχουν αρνητικές τιμές στον πίνακα x_B, άρα η λύση που έχουμε είναι η βέλτιστη λύση για $x_2 = 1$, $x_5 = 7$, $x_6 = 5$, $x_7 = 1$ και $x_1 = x_3 = x_4 = x_8 = 0$ με $z = 5\,x_1 + 7\,x_2 + 2\,x_3 + 8\,x_4 = 5 * 0 + 7 * 1 + 2 * 0 + 8 * 0 = 7$.

Η παραπάνω λύση αντιστοιχεί στο ελάχιστο για το αρχικό πρόβλημα και στο μέγιστο για το δυϊκό του πρόβλημα.

2.7 Προβλήματα κατά την εκτέλεση αλγορίθμων τύπου Simplex

Κατά την εκτέλεση των επαναλήψεων ενός αλγορίθμου τύπου Simplex, είναι δυνατόν να δημιουργηθούν κάποια προβλήματα, όπως:

- Η τιμή της αντικειμενικής συνάρτησης μπορεί να μη βελτιωθεί μετά από την εκτέλεση μίας περιστροφής. Εμφανίζεται δηλαδή το φαινόμενο της **στασιμότητας** (**stalling**).
- Μπορεί ο αλγόριθμος να πέσει σε ένα ατέρμονο βρόχο, όπου μετά από τις αλλαγές βάσης να επανέρχεται σε μία βάση η οποία έχει εξεταστεί κατά ένα προηγούμενο βήμα. Με άλλα λόγια μπορεί να εμφανιστεί ένα φαινόμενο το οποίο είναι γνωστό ως **κύκλωση (cycling)**.

Τα πιο πάνω φαινόμενα είναι δυνατόν να εμφανιστούν όταν προκύπτουν λύσεις, οι οποίες καλούνται **εκφυλισμένες (degenerate)**. Σε μία εκφυλισμένη λύση, μία ή περισσότερες από τις βασικές μεταβλητές, έχουν τιμή ίση με μηδέν. Υπάρχουν διάφοροι ερευνητές οι οποίοι έχουν μελετήσει μορφές προβλημάτων που παρουσιάζουν κύκλωση (Beale 1955, Marshall & Suurballe 1969).

Ένας αλγόριθμος τύπου Simplex μπορεί να εμφανίσει κύκλωση μόνο όταν κατασκευάζονται εκφυλισμένα σημεία, αφού τότε μπορεί να έχουμε παραπάνω από μία επιλογές για την εξερχόμενη μεταβλητή. Εάν δεν υπάρχουν φαινόμενα εκφυλισμού, μία μέθοδος τύπου Simplex, είναι σίγουρο ότι θα σταματήσει μετά από ένα πεπερασμένο αριθμό βημάτων. Αυτό προκύπτει από το γεγονός ότι υπάρχει πεπερασμένος αριθμός κορυφών οι οποίες δημιουργούνται από τους περιορισμούς του προβλήματος στον χώρο R^n. Ο τρόπος με τον οποίο γίνεται επιλογή των μεταβλητών με βάση τις οποίες εκτελείται η περιστροφή, μπορεί να βοηθήσει στον περιορισμό των πιο πάνω φαινομένων. Έχουν εμφανιστεί έτσι διάφοροι κανόνες περιστροφής, όπως περιγράφονται στην πιο κάτω παράγραφο.

2.8 Κανόνες Περιστροφής

Η επιλογή της εισερχόμενης και της εξερχόμενης μεταβλητής, μπορεί να γίνει με διαφορετικούς τρόπους, ανάλογα με τον κανόνα περιστροφής που χρησιμοποιείται κάθε

φορά. Υπάρχουν διάφοροι κανόνες περιστροφής, οι βασικότεροι από τους οποίους εμφανίζονται πιο κάτω:

- **Κανόνας του Dantzig** ή κανόνας του μέγιστου συντελεστή (maximum coefficient rule).

  Ο κανόνας του Dantzig καθορίζει τον τρόπο με τον οποίο γίνεται η επιλογή της εισερχόμενης μεταβλητής, ενώ δεν υπάρχει κάποιος συγκεκριμένος προσδιορισμός που να αφορά τον τρόπο επιλογής της εξερχόμενης μεταβλητής. Ως εισερχόμενη μεταβλητή x_l ορίζεται σε κάθε επανάληψη, η μεταβλητή που αντιστοιχεί στην ελάχιστη τιμή για την πλεονασματική μεταβλητή s, δηλαδή

  $$s_l = \min\{s_j \mid s_j<0 \}$$

  Αποτελεί τον παλαιότερο κανόνα που χρησιμοποιήθηκε και, παρόλο που στην χειρότερη περίπτωση παρουσιάζει εκθετική πολυπλοκότητα, θεωρείται πολύ αποτελεσματικός στην πράξη και εύκολος στην υλοποίηση, όμως δεν εξασφαλίζει την αποφυγή της κύκλωσης (Klee & Minty 1972).

- **Κανόνας του Bland** ή κανόνας του ελαχίστου δείκτη (least index rule).

  Σύμφωνα με τον κανόνα του Bland, τόσο για κατά την επιλογή της εισερχόμενης μεταβλητής, όσο και για την επιλογή της εξερχόμενης μεταβλητής, επιλέγεται αυτή που έχει τον μικρότερο δείκτη. Ο κανόνας του Bland βοηθάει στην αποφυγή του φαινομένου της κύκλωσης, δηλαδή είναι σίγουρο ότι όταν χρησιμοποιείται ο κανόνας αυτός, ο αλγόριθμος θα τερματίσει σε πεπερασμένο αριθμό βημάτων. Όμως, δεν αποφεύγονται φαινόμενα στασιμότητας (stalling), δηλαδή μη βελτίωσης της αντικειμενικής συνάρτησης μετά από διαδοχικές επαναλήψεις (Bland 1977). Στην πράξη συχνά ο κανόνας δεν είναι πολύ αποτελεσματικός αφού απαιτούνται περισσότερες επαναλήψεις σε σχέση με άλλους κανόνες.

- **Κανόνας της Στοίβας** (stack rule).

  Κατά την εφαρμογή της μεθόδου αυτής, διατηρείται μία στοίβα για τις βασικές μεταβλητές και μία άλλη στοίβα για τις μη βασικές μεταβλητές. Η επιλογή της εισερχόμενης μεταβλητής γίνεται με βάση τη μη βασική μεταβλητή που βρίσκεται στο υψηλότερο σημείο της στοίβας. Παρόμοια γίνεται και η επιλογή της εξερχόμενης μεταβλητής από τη στοίβα των βασικών μεταβλητών. Εφαρμόζεται δηλαδή η αρχή LIFO (Last In First Out). Ο κανόνας αυτός δεν παρουσιάζει φαινόμενα κύκλωσης και είναι πιο αποτελεσματικός από τον κανόνα του Bland.

- **Κανόνας της Ουράς** (queue rule).

  Είναι παρόμοιος με τον κανόνα της Στοίβας που περιγράφτηκε πιο πάνω, με τη διαφορά ότι χρησιμοποιείται η αρχή FIFO (First In First Out).

- **Μέθοδος της λιγότερο πρόσφατης θεώρησης** (least recently considered method).

  Σύμφωνα με τη μέθοδο αυτή, σε κάθε επανάληψη, επιλέγεται ως εισερχόμενη μεταβλητή x_l, αυτή η οποία έχει το μικρότερο δείκτη που είναι μεγαλύτερος από τον δείκτη που επιλέχτηκε κατά την προηγούμενη επανάληψη (Zadeh 1980). Η υλοποίηση τη μεθόδου δεν είναι δύσκολη και αποφεύγει τα φαινόμενα της στασιμότητας. Η πολυπλοκότητα μάλιστα της μεθόδου θεωρείται ότι είναι πολυωνυμική, χωρίς όμως να έχει πλήρως αποδεχθεί (Gartner 1995).

- **Κανόνας τη μέγιστης βελτίωσης**. (maximum improvement rule)

  Με βάση τον κανόνα αυτόν, από όλες τις επιλογές που υπάρχουν για την εισερχόμενη μεταβλητή, επιλέγεται αυτή η οποία επιφέρει τη μεγαλύτερη βελτίωση στην τιμή της αντικειμενικής συνάρτησης (Klee & Minty 1972). Με τον τρόπο αυτόν, μπορεί να βρεθεί η βέλτιστη λύση σε λιγότερες επαναλήψεις, όμως από την άλλη μεριά για κάθε επανάληψη απαιτείται μεγάλη υπολογιστική προσπάθεια. Η πολυπλοκότητα χρόνου χειρότερης περίπτωσης για τη μέθοδο αυτή θεωρείται εκθετική (Gartner 1995).

- **Κανόνας της μερικής αποτίμησης**.(partial pricing rule)

  Στην περίπτωση αυτή, για την επιλογή της εισερχόμενης μεταβλητής, οι μη βασικές μεταβλητές ομαδοποιούνται σε σύνολα, τα οποία εξετάζονται διαδοχικά κατά τις επαναλήψεις του αλγορίθμου Simplex. Το κέρδος είναι ότι σε κάθε επανάληψη απαιτούνται λιγότεροι υπολογισμοί, ενώ οι ομάδες μπορούν να δημιουργούνται είτε με δυναμικό τρόπο κατά τη διάρκεια των επαναλήψεων, είτε με στατικό τρόπο στο ξεκίνημα του αλγορίθμου (Benichou et al 1977).

- **Κανόνας της περισσότερο απότομης μεταβολής** (steepest edge rule).

  Ως εισερχόμενη επιλέγεται η μεταβλητή που επιφέρει την πιο απότομη μεταβολή στην τιμή της αντικειμενική συνάρτησης ανά μονάδα απόστασης, με βάση τη γεωμετρία του προβλήματος. Για τον προσδιορισμό όμως της μεταβλητής που επιφέρει την μέγιστη μεταβολή, υπάρχουν μεγάλες υπολογιστικές απαιτήσεις και για τον λόγο αυτό, μπορούν να χρησιμοποιηθούν προσεγγιστικές μέθοδοι

(Swietanowski 1998, Vanderbei 2001). Ο κανόνας αυτός θεωρείται κατάλληλος για παράλληλες υλοποιήσεις (Thomadakis & Jyh-Charn Liu 1996).

2.9 Εύρεση αρχικής εφικτής λύσης

Οι αλγόριθμοι τύπου Simplex για να λειτουργήσουν, απαιτούν την χρήση μίας αρχικής βασικής λύσης. Κάποιες φορές η εύρεση μίας αρχικής βασικής εφικτής λύσης είναι εύκολη. Αυτό συμβαίνει σε περιπτώσεις όπου έχουμε μόνο ανισοτικούς περιορισμούς και εισάγουμε m νέες μεταβλητές για την μετατροπή του προβλήματος σε τυποποιημένη μορφή. Στις περιπτώσεις αυτές, ο πίνακας Α περιέχει ως υποπίνακα, τον μοναδιαίο πίνακα mxm και για την επιλογή μίας αρχικής εφικτής λύσης, είναι αρκετό να επιλέξουμε τις μεταβλητές που αντιστοιχούν στις στήλες του υποπίνακα αυτού. Ωστόσο, στη γενική περίπτωση, η εύρεση μιας τέτοιας αρχικής λύσης δεν είναι εύκολη, ώστε να ξεκινήσει η εφαρμογή του αλγορίθμου. Για το λόγο αυτό, έχουν αναπτυχθεί δύο τεχνικές για την εύρεση μίας αρχικής βασικής εφικτής λύσης. Οι τεχνικές αυτές είναι η μέθοδος των δύο φάσεων και η μέθοδος του μεγάλου Μ, οι οποίες περιγράφονται στις επόμενες παραγράφους.

2.9.1 Η μέθοδος των δύο φάσεων

Η μέθοδος των δύο φάσεων (two phase method) ονομάζεται έτσι γιατί κατά την εκτέλεσή της περιλαμβάνει τις πιο κάτω δύο φάσεις:

- **Φάση Ι**. Κατά τη φάση αυτή της μεθόδου, αρχικά επιλύεται ένα τροποποιημένο γραμμικό πρόβλημα. Το νέο αυτό γραμμικό πρόβλημα προκύπτει από την προσθήκη κάποιων τεχνητών μεταβλητών και η αντικειμενική συνάρτηση, για το τροποποιημένο αυτό πρόβλημα, περιέχει το άθροισμα των τεχνητών μεταβλητών. Από την επίλυση του τροποποιημένου προβλήματος, προκύπτει μία αρχική βασική εφικτή λύση για το αρχικό πρόβλημα. Αν η ελάχιστη τιμή για την αντικειμενική συνάρτηση είναι 0, τότε περνάμε στην δεύτερη φάση για την εύρεση της βέλτιστης λύσης του αρχικού προβλήματος. Σε διαφορετική περίπτωση, το αρχικό πρόβλημα είναι αδύνατο.

- **Φάση ΙΙ**. Στη δεύτερη φάση της μεθόδου, χρησιμοποιείται η λύση που εντοπίστηκε κατά την προηγούμενη φάση, για την επίλυση του αρχικού γραμμικού προβλήματος.

Έστω, για παράδειγμα, το πρόβλημα γραμμικού προγραμματισμού που έχει την πιο κάτω μορφή:

$$\min z = -3x_1 + 2x_2 + x_3$$

$$\text{μ.π.} \quad x_1 - 3x_2 + x_3 \leq 5$$

$$-2x_1 + x_2 + 2x_3 \geq 3$$

$$-2x_1 + x_3 = 1$$

$$x_1, x_2, x_3 \geq 0$$

Το πρόβλημα παίρνει την πιο κάτω τυποποιημένη μορφή, αφού πρώτα προστεθούν νέες μεταβλητές x_4 και x_5:

$$\min z = -3x_1 + 2x_2 + x_3$$

$$\text{μ.π.} \quad x_1 - 3x_2 + x_3 + x_4 = 5$$

$$-2x_1 + x_2 + 2x_3 - x_5 = 3$$

$$-2x_1 + x_3 = 1$$

$$x_1, x_2, x_3, x_4, x_5 \geq 0$$

Στην τυποποιημένη μορφή του προβλήματος, ο πίνακας Α δεν περιέχει τον μοναδιαίο πίνακα 3x3. Για αυτόν τον λόγο, εφαρμόζοντας τη μέθοδο των δύο φάσεων, εισάγουμε δύο νέες τεχνητές μεταβλητές, x_6 και x_7 και κατά την φάση Ι επιλύουμε το πιο κάτω πρόβλημα:

$$\min r = x_6 + x_7$$

$$\text{μ.π.} \quad x_1 - 3x_2 + x_3 + x_4 = 5$$

$$-2x_1 + x_2 + 2x_3 - x_5 + x_6 = 3$$

$$-2x_1 + x_3 + x_7 = 1$$

$$x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0$$

Για το πρόβλημα αυτό μπορούμε να διαπιστώσουμε ότι υπάρχει αρχική βασική εφικτή λύση, αρκεί να επιλέξουμε ως βασικές τις μεταβλητές x_4, x_6 και x_7, οι συντελεστές των οποίων σχηματίζουν τον μοναδιαίο πίνακα 3x3. Το πρόβλημα αυτό μπορεί να επιλυθεί με τον αναθεωρημένο αλγόριθμο Simplex ή οποιαδήποτε άλλη παραλλαγή της μεθόδου Simplex.

Προφανώς ισχύει $r \geq 0$ και αν για τη βέλτιστη λύση για το πρόβλημα της πρώτης φάσης, ισχύει r=0, τότε στη συνέχεια, κατά τη δεύτερη φάση της μεθόδου, αρκεί να συνεχίζουμε την επίλυση του προβλήματος από το σημείο που προέκυψε από τη δεύτερη φάση. Με άλλα λόγια, η δεύτερη φάση της μεθόδου, θεωρεί ως αρχική βασική λύση, εκείνη τη λύση που προέκυψε από την πρώτη φάση, ενώ σαν αντικειμενική συνάρτηση θεωρείται η αντικειμενική συνάρτηση του αρχικού προβλήματος.

2.9.2 Η μέθοδος του μεγάλου M

Η μέθοδος του μεγάλου M (big M method), αποτελεί μία εναλλακτική μέθοδο για την εύρεση μίας αρχικής βασικής εφικτής λύσης. Και αυτή η μέθοδος δημιουργεί ένα νέο τροποποιημένο γραμμικό πρόβλημα προσθέτοντας τεχνητές μεταβλητές, αλλά έχει το πλεονέκτημα ότι δεν χρειάζεται να επιλύσει δύο διαφορετικά προβλήματα, όπως κάνει η μέθοδος των δύο φάσεων. Αντίθετα, επιλύει ένα μόνο τροποποιημένο γραμμικό πρόβλημα κάνοντας χρήση ενός πολύ μεγάλου αριθμού M, η τιμή του οποίου θεωρητικά προσεγγίζει το άπειρο, αλλά στην πράξη επιλέγεται ώστε να είναι αρκετά μεγάλη για τα δεδομένα του προβλήματος. Συνηθισμένες τιμές για το M είναι οι $M=10^{20}$ ή $M=10^{30}$. Στη συνέχεια, μετά τη λύση του τροποποιημένου προβλήματος, θα πρέπει να εξεταστεί η λύση που έχει βρεθεί. Αν το νέο πρόβλημα είναι βέλτιστο, τότε αν η τιμή των τεχνητών μεταβλητών είναι 0, το αρχικό πρόβλημα είναι επίσης βέλτιστο και έχει βρεθεί μία βέλτιστη λύση. Σε διαφορετική περίπτωση, δηλαδή αν στη βέλτιστη λύση υπάρχουν τεχνητές μεταβλητές με μη μηδενικές τιμές, τότε το αρχικό πρόβλημα είναι αδύνατο. Στην περίπτωση που το τροποποιημένο πρόβλημα είναι απεριόριστο, τότε το αρχικό πρόβλημα θα είναι είτε απεριόριστο, είτε αδύνατο.

Ουσιαστικά, η μέθοδος του μεγάλου M συνδυάζει τις δύο φάσεις της προηγούμενης μεθόδου σε μία. Η νέα μορφή του προβλήματος, είναι παρόμοια με την τροποποιημένη

μορφή του προβλήματος της φάσης Ι της προηγούμενης μεθόδου, με διαφορετική όμως αντικειμενική συνάρτηση. Για το παράδειγμα δηλαδή της προηγούμενης παραγράφου, θεωρούμε το πρόβλημα:

$$\min z = -3x_1 + 2x_2 + x_3 + M x_6 + M x_7$$

$$\begin{aligned} \text{μ.π.} \quad & x_1 - 3x_2 + x_3 + x_4 = 5 \\ & -2x_1 + x_2 + 2x_3 - x_5 + x_6 = 3 \\ & -2x_1 + x_3 + x_7 = 1 \\ & x_1, x_2, x_3, x_4, x_5, x_6, x_7 \geq 0 \end{aligned}$$

Στη συνέχεια, αρκεί να επιλύσουμε το νέο αυτό τροποποιημένο πρόβλημα και ανάλογα με τη λύση που προκύπτει, μπορούμε να βγάλουμε συμπεράσματα για τη λύση του αρχικού προβλήματος, όπως περιγράφτηκε πιο πάνω.

2.10 Τεχνικές κλιμάκωσης

Συχνά, κατά την επίλυση Γραμμικών Προβλημάτων, συναντάμε πολύ μεγάλους πίνακες, στους οποίους υπάρχει πολύ μεγάλη απόκλιση ανάμεσα στις απόλυτες τιμές των αριθμών των πινάκων αυτών. Το γεγονός αυτό μπορεί να προκαλέσει προβλήματα ως προς την ακρίβεια των λύσεων που προκύπτουν μετά την εφαρμογή των αλγορίθμων τύπου Simplex. Τα προβλήματα αυτά μπορούν να ξεπεραστούν, εάν έχουμε στοιχεία στον πίνακα τα οποία δεν διαφέρουν σημαντικά ως προς την απόλυτη τιμή τους. Αυτό ακριβώς επιτυγχάνουν οι τεχνικές κλιμάκωσης (scaling techniques). Οι πρώτες τεχνικές κλιμάκωσης προτάθηκαν από τον Tomlin (Tomlin 1975).

2.10.1 Η τεχνική της ισορρόπησης

Μία βασική τεχνική κλιμάκωσης είναι η τεχνική της ισορρόπησης (equilibration technique), η οποία κάνει χρήση του μέγιστου, κατ' απόλυτη τιμή, στοιχείου της κάθε στήλης του πίνακα Α. Εάν δηλαδή θεωρήσουμε το στοιχείο

$$colmax(j) = \max\left\{\frac{1}{|\alpha_{ij}|} : i = 1,2,\dots,m\right\}, j = 1,2,\dots,n$$

στη συνέχεια, τα στοιχεία της κάθε στήλης, καθώς και τα στοιχεία του πίνακα c, πολλαπλασιάζονται με τον αντίστροφο της μέγιστης αυτής τιμής, δηλαδή πολλαπλασιάζονται με τον αριθμό

$$colmax(j)$$

Με τον τρόπο αυτό, τα στοιχεία της κάθε στήλης του πίνακα A, περιέχουν τιμές ανάμεσα στο -1 και στο +1. Εάν υπάρχουν γραμμές του πίνακα που δεν περιέχουν τον αριθμό +1 ή τον αριθμό -1, τότε υπολογίζεται το μέγιστο, κατ' απόλυτη τιμή, στοιχείο της γραμμής αυτής. Δηλαδή υπολογίζουμε τον αριθμό:

$$rowmax(i) = \max\left\{\frac{1}{|\alpha_{ij}|} : j = 1,2,\dots,n\right\}, i = 1,2,\dots,m$$

Στη συνέχεια, πολλαπλασιάζονται τα στοιχεία των γραμμών αυτών, καθώς και τα αντίστοιχα στοιχεία του πίνακα b, με τον αριθμό

$$rowmax(i)$$

Έτσι, επιτυγχάνουμε την ύπαρξη σε κάθε γραμμή του πίνακα A, ενός τουλάχιστον στοιχείου με απόλυτη τιμή ίση με τη μονάδα.

Η βέλτιστη λύση ενός κλιμακωμένου Γραμμικού Προβλήματος, διαφέρει από τη βέλτιστη λύση του αρχικού προβλήματος. Η μετατροπή από την κλιμακωμένη βέλτιστη λύση x σε αυτήν του αρχικού προβλήματος, γίνεται με τις πράξεις:

$$x' = x \otimes colmulti^T$$

και

$$x'' = x' \otimes rowmulti^T$$

όπου colmulti και rowmulti τα διανύσματα με τα μέγιστα των στηλών και των γραμμών του πίνακα A, αντίστοιχα. Με το σύμβολο $\otimes$, θεωρούμε ότι γίνεται πολλαπλασιασμός στοιχείο προς στοιχείο των αντίστοιχων διανυσμάτων. Το διάνυσμα x" που προκύπτει αποτελεί τη λύση για το αρχικό πρόβλημα Γραμμικού Προγραμματισμού. Η βέλτιστη τιμή της αντικειμενικής συνάρτησης είναι ίδια και στα δύο προβλήματα, στο κλιμακωμένο και στο αρχικό πρόβλημα.

Για παράδειγμα, αν θεωρήσουμε το πρόβλημα Γραμμικού Προγραμματισμού, που αντιστοιχεί στους πίνακες

$$A = \begin{bmatrix} 100 & 90 & -80 & -20 \\ 80 & -80 & 40 & 40 \\ -1 & -100 & 20 & 10 \\ 2 & 3 & 40 & 30 \end{bmatrix}$$

$$c = \begin{bmatrix} 200 \\ -100 \\ 160 \\ -80 \end{bmatrix} \quad b = \begin{bmatrix} 10 \\ 5 \\ 20 \\ -10 \end{bmatrix}$$

Οπότε,

$$colmulti = \begin{bmatrix} \frac{1}{100} & \frac{1}{100} & \frac{1}{80} & \frac{1}{40} \end{bmatrix}$$

και προκύπτει

$$A = \begin{bmatrix} 1 & 0{,}9 & -1 & -0{,}5 \\ 0{,}8 & -0{,}8 & 0{,}5 & 1 \\ -0{,}01 & -1 & 0{,}25 & 0{,}25 \\ 0{,}02 & 0{,}03 & 0{,}5 & 0{,}75 \end{bmatrix} \quad c = \begin{bmatrix} 2 \\ -1 \\ 2 \\ -2 \end{bmatrix}$$

Επειδή η τέταρτη γραμμή δεν περιέχει στοιχείο με απόλυτη τιμή τη μονάδα, θα πρέπει να βρεθεί το μέγιστο της γραμμής αυτής, κατ' απόλυτη τιμή, το οποίο ισούται με 0,75. Στη συνέχεια, πολλαπλασιάζουμε τα στοιχεία της γραμμής, καθώς και το τέταρτο στοιχείο του b, με τον αντίστροφο του μεγίστου αυτού, δηλαδή με το 1/0,75 = 1,33.

Επομένως έχουμε:

$$A = \begin{bmatrix} 1 & 0{,}9 & -1 & -0{,}5 \\ 0{,}8 & -0{,}8 & 0{,}5 & 1 \\ -0{,}01 & -1 & 0{,}25 & 0{,}25 \\ 0{,}026 & 0{,}04 & 0{,}66 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 10 \\ 5 \\ 20 \\ -13{,}33 \end{bmatrix}$$

Δηλαδή θεωρήσαμε ότι είναι

$$rowmulti = \begin{bmatrix} 1 & 1 & 1 & \frac{1}{0{,}75} \end{bmatrix}$$

Στη συνέχεια, αρκεί να επιλύσουμε το νέο κλιμακωμένο πρόβλημα, ώστε να βρεθεί η λύση του αρχικού προβλήματος, με τους τύπους που δόθηκαν πιο πάνω.

2.10.2 Η τεχνική του Γεωμετρικού Μέσου

Η τεχνική του γεωμετρικού μέσου (geometric mean) χρησιμοποιεί το μεγαλύτερο κατά απόλυτη τιμή στοιχείο της κάθε στήλης του πίνακα Α, όπως επίσης και το μικρότερο κατά απόλυτη τιμή στοιχείο της κάθε στήλης. Έχουμε δηλαδή τα στοιχεία:

$$colmax(j) = \max\{|\alpha_{ij}|: i = 1,2,\dots,m\}, j = 1,2,\dots,n$$

και

$$colmin(j) = \min\{|\alpha_{ij}|: i = 1,2,\dots,m\}, j = 1,2,\dots,n$$

Με τη βοήθεια των τιμών αυτών υπολογίζεται η ποσότητα

$$\mathrm{mc(j)} = \sqrt{colmax(j)\, colmin(j)}$$

και πολλαπλασιάζονται όλα τα στοιχεία της στήλης j με την τιμή $\frac{1}{mc(j)}$.

Με τον τρόπο αυτό, η μέγιστη τιμή των στοιχείων της στήλης j είναι

$$\beta(j) = \pm\frac{\sqrt{colmax(j)}\sqrt{colmin(j)}}{colmin(j)}$$

Η ίδια διαδικασία επαναλαμβάνεται για τις γραμμές του πίνακα Α και υπολογίζονται τα στοιχεία:

$$rowmax(j) = \max\{|\alpha_{ij}|: j = 1,2,\dots,n\}, i = 1,2,\dots,m$$

$$rowmin(j) = \min\{|\alpha_{ij}|: j = 1,2,\dots,n\}, i = 1,2,\dots,m$$

και

$$\mathrm{mr(i)} = \sqrt{rowmax(i)\, rowmin(i)}$$

Τα στοιχεία της i γραμμής πολλαπλασιάζονται με την τιμή του $\frac{1}{mc(j)}$ ώστε το μέγιστο στοιχείο να είναι

$$\beta(i) = \pm\frac{\sqrt{rowmax(i)}\sqrt{rowmin(i)}}{rowmin(i)}$$

Με τον τρόπο αυτό ολοκληρώνεται η κλιμάκωση του προβλήματος.

3 Υλοποίηση Αλγορίθμων τύπου Simplex σε Python

3.1 Η γλώσσα προγραμματισμού Python

Η γλώσσα προγραμματισμού Python είναι μια γλώσσα γενικού σκοπού, υψηλού επιπέδου, η οποία αναπτύχθηκε αρχικά από τον Ολλανδό Guido Van Rossum. Η αρχική έκδοση της γλώσσας, κυκλοφόρησε το 1991, αποτελώντας στην ουσία διάδοχο της γλώσσας ABC (Python, 2018). Η γλώσσα κέντρισε το ενδιαφέρον των προγραμματιστών, λόγω της απλότητάς της και του ιδιαίτερα αναγνώσιμου κώδικα που μπορεί να δημιουργηθεί μέσω της γλώσσας. Τα προγράμματα σε Python έχουν συχνά αρκετά λιγότερες εντολές σε σχέση με άλλες δημοφιλείς ανταγωνιστικές γλώσσες προγραμματισμού. Το όνομα της γλώσσας προήλθε από το όνομα της γνωστής ομάδας Άγγλων κωμικών με το όνομα Monty Python.

Η έκδοση 2.0 της γλώσσας κυκλοφόρησε τον Οκτώβριο του 2000, προσφέροντας πολλές και σημαντικές νέες δυνατότητες σε σχέση με την προηγούμενη έκδοση. Δημιουργήθηκε μία πολύ δυναμική κοινότητα προγραμματιστών Python που επικοινωνούσαν μέσω του Διαδικτύου και βοήθησε πολύ σημαντικά στην βελτίωση και εξέλιξη της γλώσσας. Τον Δεκέμβριο του 2008, μετά από μία αρκετά μεγάλη περίοδο ζυμώσεων, εμφανίστηκε η έκδοση 3.0 της γλώσσας, η οποία όμως έχει κάποια σημεία ασυμβατότητας σε σχέση με τις προηγούμενες εκδόσεις. Λόγω της ασυμβατότητας αυτής και του μεγάλου όγκου υπαρχόντων προγραμμάτων, ενώ αρχικά είχε οριοθετηθεί ο Δεκέμβριος του 2015, ως το ορόσημο για την υποστήριξη της έκδοσης 2.x της γλώσσας, στη συνέχεια ανακοινώθηκε παράταση μέχρι το 2020, εξαιτίας του ότι είναι πολύ δύσκολο να μετατραπούν όλα τα προγράμματα σε εκδόσεις 2.x, ώστε να είναι συμβατά με την έκδοση 3.0 και πάνω. Αυτή τη στιγμή, από τον Ιούνιο του 2018, η τελευταία έκδοση της γλώσσας Python είναι η έκδοση 3.7 (Python, 2018).

Η γλώσσα προγραμματισμού Python αποτελεί ανοιχτό λογισμικό (open source) το οποίο διατίθεται με άδεια PSFL (Python Software Foundation License), που είναι συμβατή με την άδεια GPL (GNU General Public License). Η άδεια PSFL επιτρέπει τη διανομή παραγώγων της γλώσσας από οποιονδήποτε φορέα ή ιδιώτη, χωρίς να έχει αυτός την υποχρέωση για διανομή του πηγαίου κώδικα του παραγώγου της γλώσσας. Ο μη κερδοσκοπικός οργανισμός Python Software Foundation (PSF), διαχειρίζεται τα διάφορα θέματα που αφορούν στη σχεδίαση και στην εξέλιξη της γλώσσας. Ως ηγέτης της

Pyhton, θεωρούνταν μέχρι τον Ιούλιο του 2018 οπότε κοινοποίησε παραίτηση, ο δημιουργός της γλώσσας, Guido Van Rossum, κατέχοντας τον άτυπο τίτλο του Benevolent Dictator For Life (BDFL).

Με τη βοήθεια της γλώσσας Python μπορούν να αναπτυχθούν προγράμματα με την ανάπτυξη τεχνικών τόσο διαδικαστικού προγραμματισμού, όσο και αντικειμενοστραφούς ή ακόμη και συναρτησιακού προγραμματισμού. Η γλώσσα προσφέρει δυναμικό έλεγχο του τύπου των δεδομένων του προγράμματος, αφού δεν αποδίδεται συγκεκριμένος τύπος στις μεταβλητές αλλά μόνο στα αντικείμενα. Οι μεταβλητές δηλαδή δεν είναι υποχρεωτικό να δηλωθούν ως προς τον τύπο τους, προτού χρησιμοποιηθούν. Αυτό δεν σημαίνει ότι δεν υπάρχει καθόλου έλεγχος των τύπων των δεδομένων. Κατά τη στιγμή της εκτέλεσης των διάφορων πράξεων, γίνεται έλεγχος της συμβατότητας των δεδομένων που συμμετέχουν στις πράξεις και στην περίπτωση μη συμβατότητας παράγεται το κατάλληλο σφάλμα. Η όλη διαδικασία όμως, σε αντίθεση με άλλες γλώσσες προγραμματισμού, γίνεται δυναμικά.

Τα προγράμματα που αναπτύσσονται με τη γλώσσα Python, είναι κατά κανόνα εύκολα αναγνώσιμα, αφού η γλώσσα επιβάλει την υποχρεωτική εφαρμογή εσοχών για την ομαδοποίηση των εντολών, σε αντίθεση με τις περισσότερες γλώσσες οι οποίες χρησιμοποιούν άγκιστρα ή κάποιον άλλον τρόπο σήμανσης της αρχής και του τέλους μίας ομάδα εντολών. Υποστηρίζεται επίσης και αυτόματη διαχείρισης μνήμης, ώστε να επιτυγχάνεται η αυτόματη απελευθέρωση του χώρου που έχει πάψει να χρησιμοποιείται από μία εφαρμογή.

Μεγάλο μέρος της λειτουργικότητας της γλώσσας Python, βασίζεται στην ύπαρξη μεγάλης ποικιλίας βιβλιοθηκών που μπορούν να χρησιμοποιηθούν από τις εφαρμογές που αναπτύσσονται. Υπάρχει πολύ μεγάλη πληθώρα έτοιμων βιβλιοθηκών οι οποίες δίνουν τη δυνατότητα της επαναχρησιμοποίησης τμημάτων του κώδικα, τα οποία έχουν αναπτυχθεί και ελεγχθεί από άλλους προγραμματιστές. Η εκτέλεση των εντολών python και η κλήση των κατάλληλων βιβλιοθηκών, γίνεται από ειδικούς διερμηνευτές (interpreters) που έχουν αναπτυχθεί για τη γλώσσα. Η Python διαθέτει διερμηνευτές για όλα τα δημοφιλή λειτουργικά συστήματα τα οποία επιτρέπουν την εύκολη και άμεση εκτέλεση των προγραμμάτων. Με τον τρόπο αυτό επιτυγχάνεται η μεταφερσιμότητα (portability) των προγραμμάτων Python σε διαφορετικές υπολογιστικές πλατφόρμες. Υπάρχουν επίσης εργαλεία, όπως το Py2exe και το Pyinstaller, τα οποία επιτρέπουν τη

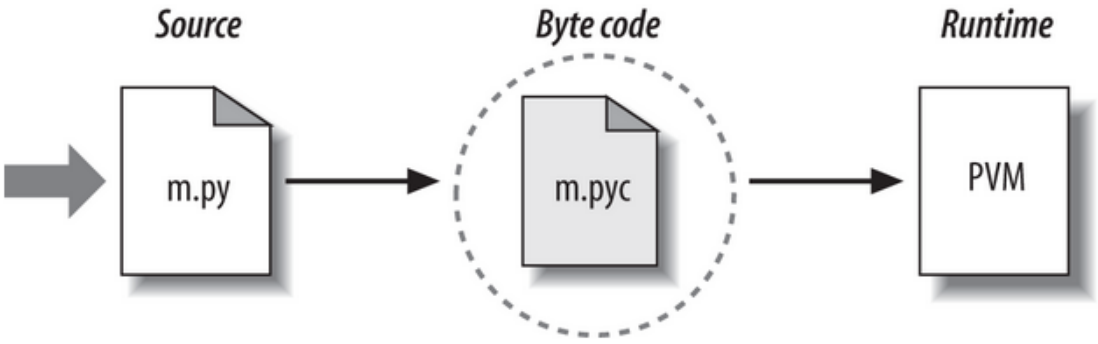
δημιουργία αυτόνομου εκτελέσιμου κώδικα, παράγοντας έτσι γρήγορα προγράμματα τα οποία μπορούν να εκτελεστούν χωρίς την ύπαρξη κάποιου διερμηνευτή ή κάποιας εικονικής μηχανής (Py2exe 2018, PyInstaller 2018).

Η Python αποτελεί πλέον μία από τις πιο δημοφιλείς γλώσσες προγραμματισμού, αφού χρησιμοποιείται πάρα πολύ, τόσο από προγραμματιστές, όσο και στον χώρο της εκπαίδευσης για την εκμάθηση των βασικών αρχών προγραμματισμού υπολογιστών. Στην Εικόνα 3-1 εμφανίζονται οι 10 πιο δημοφιλείς γλώσσες προγραμματισμού για τον Αύγουστο του 2018, όπως αυτό προκύπτει από την κατάταξη του δείκτη TIOBE (TIOBE, 2018). Οι τιμές του δείκτη TIOBE ανανεώνονται κάθε μήνα με βάση τις πληροφορίες που προκύπτουν από εκατοντάδες πηγές, οι οποίες μελετούν τη χρήση των διάφορων γλωσσών προγραμματισμού. Η Python μάλιστα παρουσιάζει έντονα αυξητικές τάσεις στην κατάταξη αυτή. Σύμφωνα με έρευνα που πραγματοποιήθηκε το 2005 η πιο δημοφιλής γλώσσα στα Πανεπιστήμια των Ηνωμένων Πολιτειών για τα εισαγωγικά μαθήματα προγραμματισμού ήταν η Java, την εποχή εκείνη, με ποσοστό 60%. Το 2018, η δημοφιλέστερη γλώσσα για τους ίδιους σκοπούς είναι η Python με ποσοστό πάνω από 70% (TIOBE, 2018). Βλέπουμε λοιπόν, ότι υπάρχει μία μεγάλη δυναμική γύρω από τη γλώσσα Python και μπορούμε να συμπεράνουμε ότι η γλώσσα θα βρίσκεται πολύ υψηλά στις προτιμήσεις των προγραμματιστών και των πανεπιστημιακών ιδρυμάτων και για τα επόμενα χρόνια.

Aug 2018	Aug 2017	Change	Programming Language	Ratings	Change
1	1		Java	16.881%	+3.92%
2	2		C	14.966%	+8.49%
3	3		C++	7.471%	+1.92%
4	5	▲	Python	6.992%	+3.30%
5	6	▲	Visual Basic .NET	4.762%	+2.19%
6	4	▼	C#	3.541%	-0.65%
7	7		PHP	2.925%	+0.63%
8	8		JavaScript	2.411%	+0.31%
9	-	⇈	SQL	2.316%	+2.32%
10	14	⇈	Assembly language	1.409%	-0.40%

Εικόνα 3-1: Οι 10 πιο δημοφιλείς γλώσσες προγραμματισμού σύμφωνα με τον δείκτη TIOBE

Ο πηγαίος κώδικας (source code) ενός προγράμματος που έχει γραφεί σε Python, κατά κανόνα βρίσκεται αποθηκευμένος σε ένα ή περισσότερα αρχεία τα οποία έχουν επέκταση .py. Ο κώδικας αυτός μεταγλωττίζεται στον κατάλληλο ενδιάμεσο κώδικα σε μορφή bytecodes, που αποθηκεύεται σε αρχεία με επέκταση .pyc. Η εκτέλεση του ενδιάμεσου αυτού κώδικα γίνεται με τη βοήθεια του κατάλληλου διερμηνευτή (interpreter) κάνοντας χρήση της εγκατεστημένης εικονικής μηχανής Python (Python Virtual Machine – PVM) στην πλατφόρμα που χρησιμοποιείται. Έτσι, με τον τρόπο αυτό, επιτυγχάνεται η μεταφερσιμότητα του κώδικα, ώστε να υπάρχει η δυνατότητα να μπορεί να εκτελεστεί σε διαφορετικές πλατφόρμες. Στην Εικόνα 3-2 εμφανίζεται η διαδικασία μεταγλώττισης και εκτέλεσης μίας εφαρμογής που έχει αναπτυχθεί σε γλώσσα Python.



Εικόνα 3-2: Μεταγλώττιση και εκτέλεση εφαρμογής Python

Η CPython έχει υλοποιηθεί σε C, με βάση το πρότυπο C89, και αποτελεί την πιο δημοφιλή έκδοση της γλώσσας Python. Είναι διαθέσιμη για διάφορα περιβάλλοντα, όπως Windows, Linux και MacOS και διανέμεται μαζί με βιβλιοθήκες έτοιμου κώδικα που έχει γραφεί σε C και Python. Εκτός από τις υλοποιήσεις οι οποίες βασίζονται σε διερμηνευτή, όπως η CPython, υπάρχουν άλλες υλοποιήσεις της γλώσσας οι οποίες χρησιμοποιούν δυναμική μεταγλώττιση του κώδικα (Just In Time compiler - JIT), όπως η PyPy. Αυτές οι υλοποιήσεις συνήθως επιτρέπουν την ανάπτυξη γρηγορότερων εφαρμογών σε Python, αφού ο κώδικας είναι ήδη μεταγλωττισμένος. Άλλες υλοποιήσεις, όπως η Jython, δίνουν τη δυνατότητα για την εκτέλεση του κώδικα μέσω της εικονικής μηχανής της Java (Java Virtual Machine - JVM). Υπάρχουν επίσης εκδόσεις, όπως η IronPython που έχει γραφεί σε C# με στόχο τη χρήση των βιβλιοθηκών .NET Framework και η Stackless Python η οποία μπορεί να χρησιμοποιηθεί για προγραμματισμό με μικρονήματα (Python, 2018).

3.1.1 Βιβλιοθήκες της Python

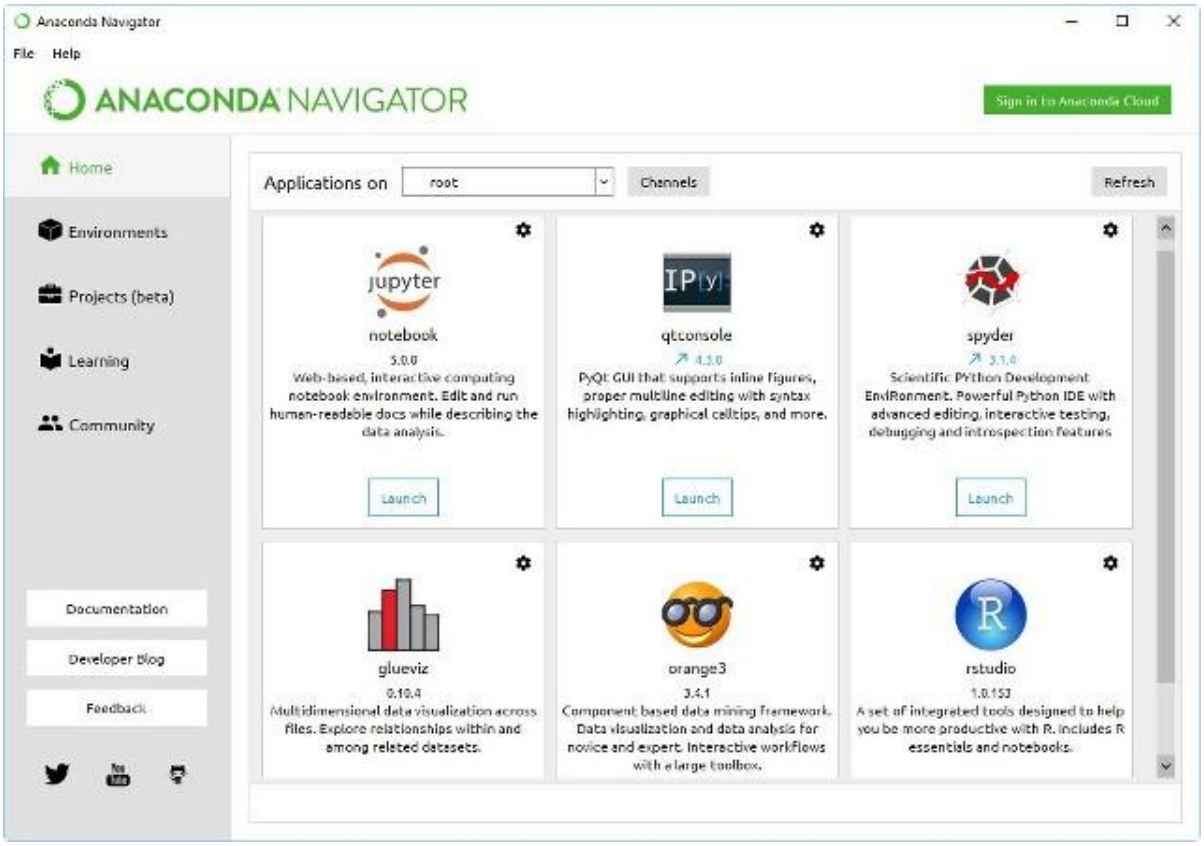
Η λειτουργικότητα και οι δυνατότητες της γλώσσας Python αυξάνονται σε τεράστιο βαθμό εξαιτίας της πληθώρας βιβλιοθηκών οι οποίες έχουν αναπτυχθεί και μπορούν να χρησιμοποιηθούν κατά την ανάπτυξη προγραμμάτων. Ουσιαστικά, μία βιβλιοθήκη είναι μία συλλογή συναρτήσεων και κλάσεων που μπορούν να επαναχρησιμοποιηθούν από οποιονδήποτε προγραμματιστή, ώστε να είναι δυνατή η εύκολη και χωρίς σφάλματα ανάπτυξη εφαρμογών. Η χρήση των βιβλιοθηκών είναι πολύ εύκολη, αρκεί να γίνει η εισαγωγή τους στο πρόγραμμα με τη βοήθεια κατάλληλων δηλώσεων import.

Η βασική βιβλιοθήκη της Python (Python Standard Library) περιλαμβάνει πολλές από τις συνηθισμένες βασικές λειτουργίες που χρειάζονται οι εφαρμογές Python, όπως η ανάγνωση και η εγγραφή αρχείων, ο χειρισμός συμβολοσειρών, σφαλμάτων (exceptions) κ.τ.λ. Πολλές από αυτές τις μεθόδους είναι γραμμένες σε γλώσσα C, για λόγους ταχύτητας, ενώ άλλες έχουν αναπτυχθεί σε Python. Εκτός από τη βασική βιβλιοθήκη, υπάρχει ένας πάρα πολύ μεγάλος αριθμός άλλων βιβλιοθηκών, για διάφορες πιο πολύπλοκες διαδικασίες, όπως η σύνδεση με κάποια Βάση Δεδομένων, η δημιουργία γραφικής διεπαφής (Graphical User Interface - GUI), η συλλογή και ανάλυση δεδομένων από το Διαδίκτυο κτλ.

Η βιβλιοθήκη NumPy, δίνει τη δυνατότητα σε ένα πρόγραμμα Python να κάνει χρήση μονοδιάστατων ή πολυδιάστατων πινάκων, καθώς και υψηλού επιπέδου μαθηματικές συναρτήσεις, όπως για παράδειγμα η εύρεση του αντιστρόφου πίνακα. Η γλώσσα Python δεν δίνει από μόνη της τη δυνατότητα χρήσης πινάκων, αφού χρησιμοποιεί μόνο λίστες. Με τη χρήση του NumPy όμως είναι δυνατόν να χρησιμοποιηθούν και οι κλασικοί πίνακες που συναντάμε σε άλλες γλώσσες προγραμματισμού (NumPy, 2018).

3.1.2 Διανομή Anaconda

Το Anaconda είναι μία ελεύθερη και ανοικτού λογισμικού διανομή της γλώσσας Python και της γλώσσας R, που περιλαμβάνει κυρίως βιβλιοθήκες που μπορούν να χρησιμοποιηθούν για ανάλυση δεδομένων και μηχανική μάθηση. Ο στόχος της διανομής είναι η απλοποίηση της διαχείρισης των πακέτων που απαιτούνται κατά την χρήση και εφαρμογή της Python σε παρόμοια επιστημονικά πεδία. Η διαχείριση των πακέτων γίνεται από ένα σύστημα που ονομάζεται conda. Η διανομή Anaconda χρησιμοποιείται από περισσότερους από 6 εκατομμύρια χρήστες παγκοσμίως και περιλαμβάνει περισσότερες από 1000 δημοφιλείς βιβλιοθήκες οι οποίες μπορούν να χρησιμοποιηθούν σε περιβάλλον Windows, Linux και MacOS (Anaconda, 2018). Περιλαμβάνεται μάλιστα και διαχειριστής Anaconda Navigator, μέσω του οποίου μπορεί να γίνει με γραφικό τρόπο η διαχείριση και εκτέλεση των συμπεριλαμβανόμενων πακέτων (Εικόνα 3-3).



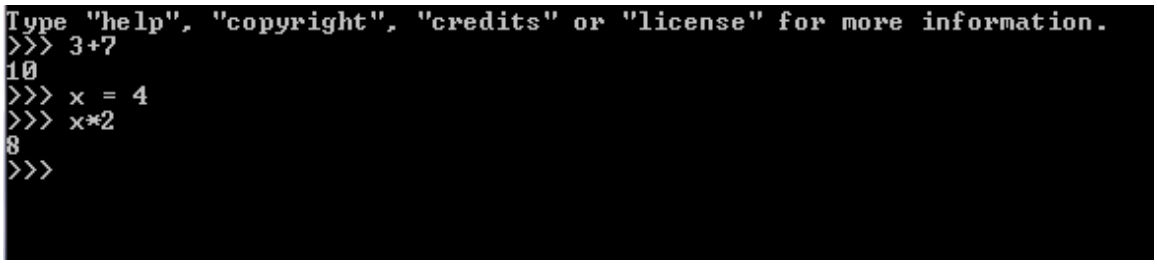
Εικόνα 3-3: Ο Anaconda Navigator

3.2 Περιβάλλον προγραμματισμού Python

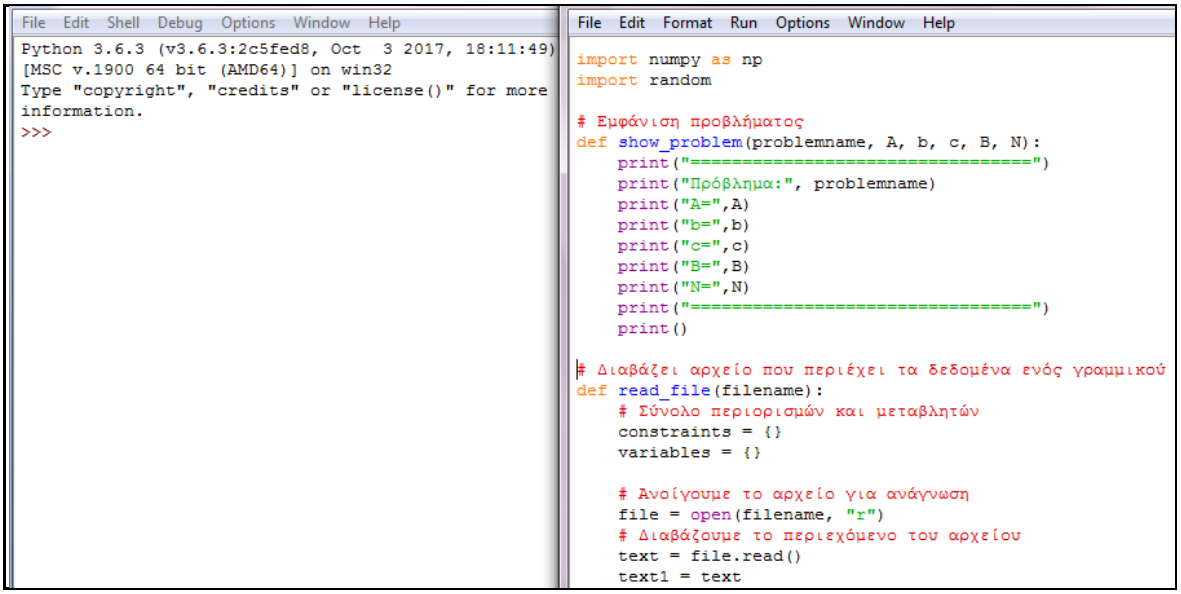
Μία εφαρμογή σε γλώσσα Python, μπορεί να πραγματοποιηθεί με βοήθεια ενός απλού κειμενογράφου (editor), ενώ η εκτέλεση και η δοκιμή του προγράμματος μπορεί να γίνει με την εκτέλεση απλών εντολών που εισάγονται από τη γραμμή εντολών του υπολογιστή. Επιπρόσθετα, διαμέσω του κελύφους (shell) της Python, μπορούμε να δώσουμε εντολές προς τον διερμηνευτή της Python, όπως φαίνεται στην Εικόνα 3-4: Το κέλυφος της Python. Παρ' όλα αυτά, για να είναι ο προγραμματισμός σε Python περισσότερα παραγωγικός, καλό είναι να γίνεται χρήση του κατάλληλου περιβάλλοντος (Integrated Development Environments – IDE).

Το περιβάλλον του IDLE, εγκαθίσταται συνήθως μαζί με τη γλώσσα Python και προσφέρει ένα ολοκληρωμένο περιβάλλον για την εύκολη δημιουργία, εκτέλεση και έλεγχο προγραμμάτων Python (Εικόνα 3-5). Υπάρχουν επίσης διάφορα άλλα ολοκληρωμένα περιβάλλοντα προγραμματισμού για Python, τα οποία προσφέρουν

περισσότερες δυνατότητες και διευκολύνσεις σε σχέση με το IDLE. Κάποια από αυτά τα περιβάλλοντα περιγράφονται σύντομα στις επόμενες παραγράφους. Το περιβάλλον προγραμματισμού το οποίο χρησιμοποιήθηκε στα πλαίσια της παρούσας εργασίας ήταν το περιβάλλον του Spyder (Spyder, 2018).



Εικόνα 3-4: Το κέλυφος της Python

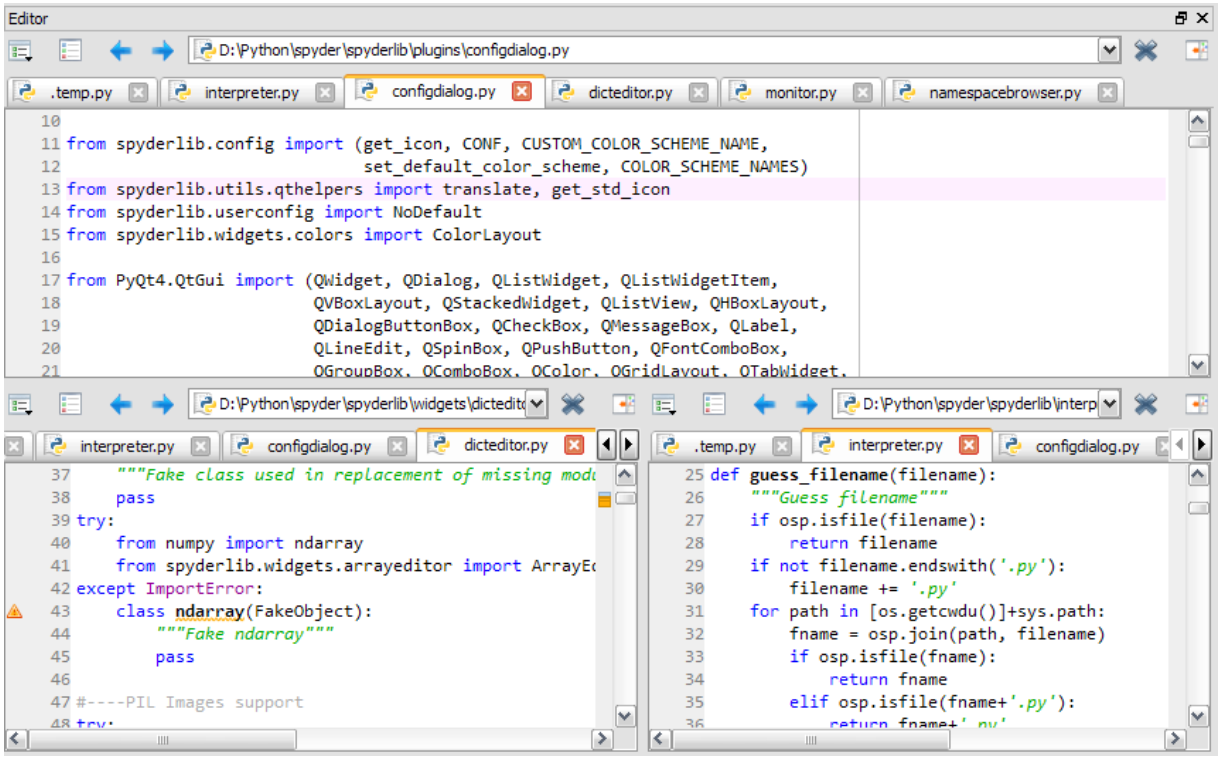


Εικόνα 3-5: Το περιβάλλον προγραμματισμού IDLE

3.2.1 Spyder

Το Spyder (Scientific PYthon Development EnviRonment) αποτελεί ένα πάρα πολύ εύχρηστο περιβάλλον IDE για την ανάπτυξη εφαρμογών Python. Το Spyder διαθέτει έναν κειμενογράφο με πολλές δυνατότητες, ο οποίος προσφέρει αυτόματο χρωματισμό

του κώδικα, έλεγχο και αποσφαλμάτωσης (debugging) του προγράμματος, αυτόματη συμπλήρωση κώδικα κ.τ.λ. (Εικόνα 3-6). Επιπλέον, το Spyder διαθέτει τον διερμηνευτή iPython για την άμεση εκτέλεση εντολών και προσφέρει επίσης πλήρη συνεργασία με δημοφιλείς βιβλιοθήκες, όπως η NumPy για την εκτέλεση Μαθηματικών πράξεων, η SciPy για την επεξεργασία σήματος και εικόνας, η MatplotLib για τη δημιουργία γραφημάτων κ.τ.λ. Το λογισμικό Spyder είναι ανοικτού κώδικα (open source) και συμπεριλαμβάνεται στη διανομή Anaconda, η οποία προσφέρει πακέτα και βιβλιοθήκες για την ανάλυση δεδομένων και τη μηχανική μάθηση (Spyder, 2018).

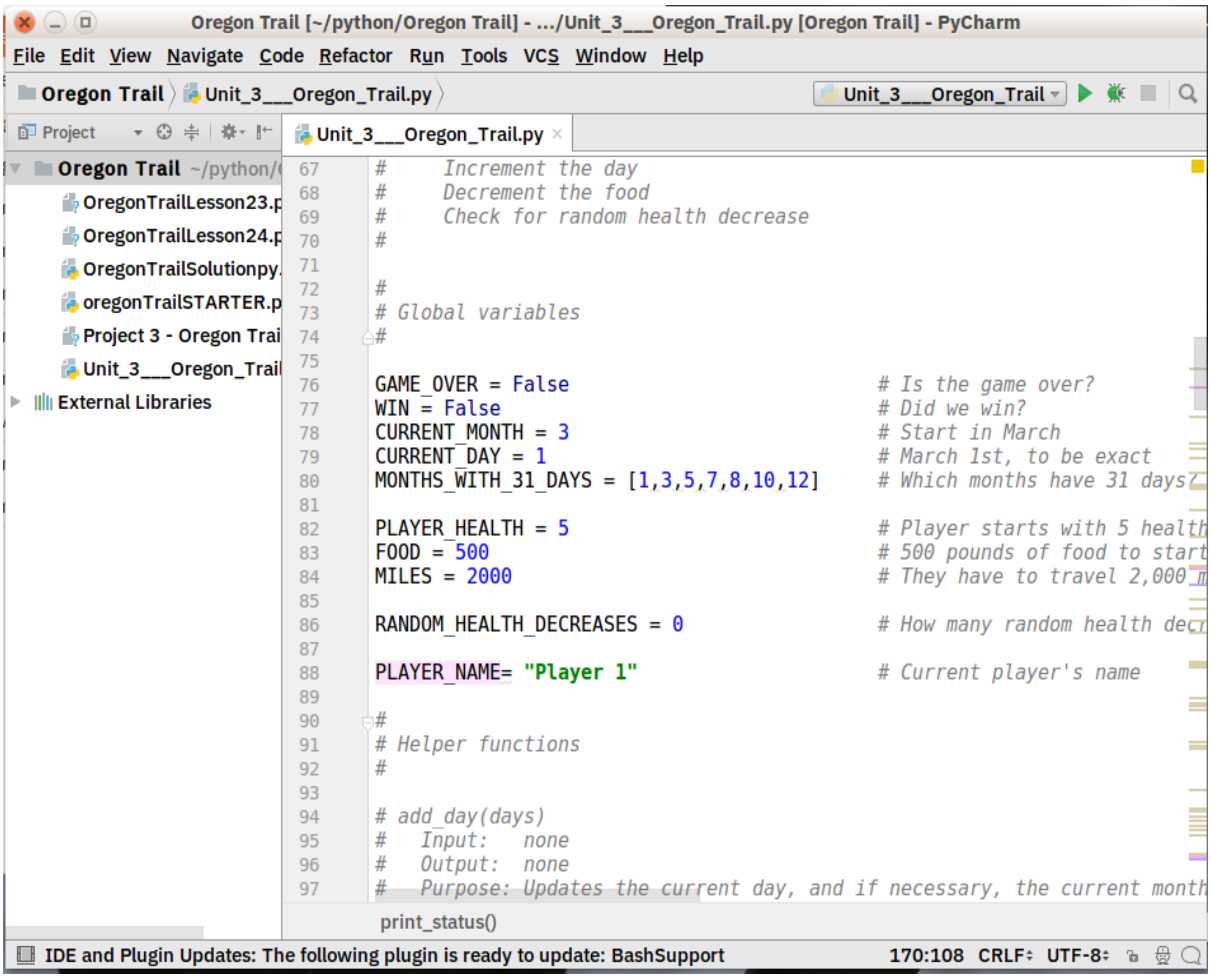


Εικόνα 3-6: Το περιβάλλον προγραμματισμού Spyder

3.2.2 PyCharm

Το περιβάλλον του PyCharm αποτελεί ένα από τα καλύτερα και πιο δημοφιλή IDE για τον προγραμματισμό σε γλώσσα Python. Διατίθεται σε περιβάλλον Windows, Linux ή MacOS (Εικόνα 3-7), αποκλειστικά και μόνο για τον προγραμματισμό Python, αφού δεν

υποστηρίζει άλλες γλώσσες. Το πρόγραμμα είναι διαθέσιμο για επαγγελματική χρήση για την οποία υπάρχει ειδική έκδοση που διαθέτει επιπλέον χαρακτηριστικά με πληρωμή, όμως υπάρχει και η δωρεάν ανοικτού λογισμικού έκδοση, η οποία δεν απαιτεί κάποια πληρωμή (PyCharm, 2018).

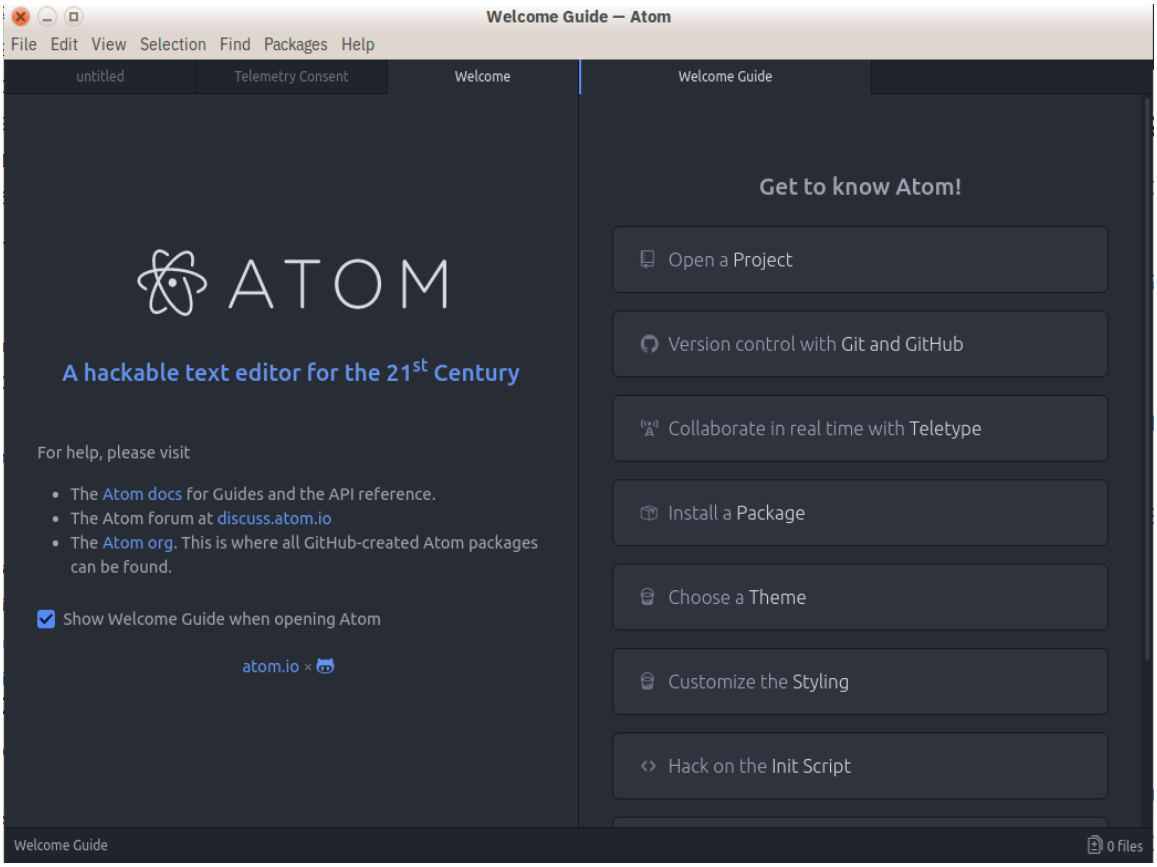


Εικόνα 3-7: Το περιβάλλον του PyCharm

3.2.3 Atom

Η εφαρμογή Atom αποτελεί έναν πάρα πολύ ισχυρό κειμενογράφο, ο οποίος μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών σε πληθώρα γλωσσών προγραμματισμού, ανάμεσα στις οποίες συμπεριλαμβάνεται και η Python. Η εφαρμογή προσφέρει ένα όμορφο περιβάλλον προγραμματισμού (Εικόνα 3-8) και έχει αναπτυχθεί με χρήση του

electron, το οποίο βασίζεται στις γλώσσες που χρησιμοποιούνται για τον προγραμματισμό Ιστού, όπως η Javascript, η HTML και η CSS. Το γεγονός αυτό, εξασφαλίζει τη φορητότητα της εφαρμογής, ώστε να μπορεί να εγκατασταθεί και να εκτελεστεί σε οποιαδήποτε πλατφόρμα. Ο κειμενογράφος του Atom φιλοξενείται στο Github και διαθέτει μία τεράστια κοινότητα στο Διαδίκτυο, με τη βοήθεια της οποίας έχουν δημιουργηθεί πάρα πολλά πρόσθετα που μπορούν να αυξήσουν τη λειτουργικότητα του προγράμματος (Atom, 2018).

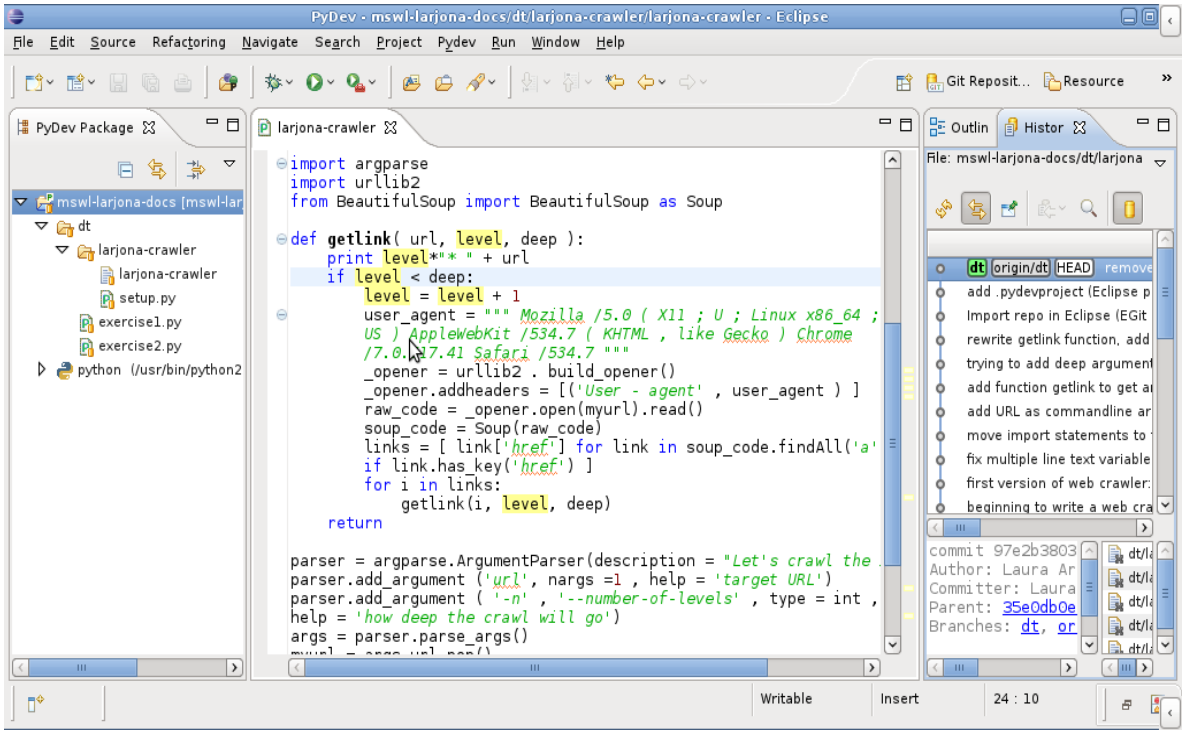


Εικόνα 3-8: Ο κειμενογράφος του Atom

3.2.4 Eclipse

Το περιβάλλον του Eclipse αποτελεί ένα πολύ δημοφιλές και πολύ ισχυρό περιβάλλον προγραμματισμού. Το Eclipse χρησιμοποιείται κυρίως για τον προγραμματισμό εφαρμογών σε γλώσσα Java και σε γλώσσα C++ (Eclipse, 2018). Με την εγκατάσταση

όμως ενός πρόσθετου με το όνομα PyDev, μπορεί το περιβάλλον να χρησιμοποιηθεί και για τον προγραμματισμό εφαρμογών και σε γλώσσα Python, προσφέροντας παρόμοια λειτουργικότητα και δυνατότητες (Εικόνα 3-9). Υποστηρίζει μάλιστα και τη δημιουργία έργων Django, ώστε να είναι δυνατή η ανάπτυξη ιστότοπων με τη βοήθεια της γλώσσας Python (PyDev, 2018).



Εικόνα 3-9: Το περιβάλλον του Eclipse

3.3 Υλοποίηση Αλγορίθμων τύπου Simplex με τη γλώσσα Python

Στην παράγραφο αυτή, θα παρουσιαστεί ο τρόπος υλοποίησης κάποιων αλγορίθμων τύπου Simplex που αναπτύχθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας. Κατά την ανάπτυξη των αλγορίθμων αυτών, αναπτύχθηκαν διάφορες συναρτήσεις, οι οποίες σχηματίζουν πακέτα, έτσι ώστε να μπορούν να επαναχρησιμοποιηθούν από διαφορετικά προγράμματα. Για παράδειγμα, οι συναρτήσεις που αφορούν κάποιες γενικές λειτουργίες γραμμικών προβλημάτων, τοποθετήθηκαν στο πακέτο *linear_problem*, ενώ αυτές που αφορούν τις μεθόδους τύπου Simplex, τοποθετήθηκαν στο πακέτο με το όνομα *simplex_method*. Για τις πράξεις ανάμεσα σε πίνακες, γίνεται εκτενής χρήση της βιβλιοθήκης *numpy* της Python, η οποία περιλαμβάνει μαθηματικές και αριθμητικές λειτουργίες.

3.3.1 Αναθεωρημένος Αλγόριθμος Simplex

Ο αναθεωρημένος αλγόριθμος Simplex υλοποιείται με την ανάπτυξη μίας συνάρτησης σε γλώσσα Python, με το όνομα *simplex_revised*, η οποία εμφανίζεται πιο κάτω:

```
def simpex_revised(problemname, A, b, c, m,
                           n, doScaling, type, M):
    # Καλούμε τη μέθοδο του μεγάλου M για τον προσδιορισμό
    # της αρχικής βάσης
    (art_vars, A, c, n, B, N) = bigM_method(M, A, c, m, n)
    # Κάνουμε κλιμάκωση αν χρειάζεται
    if doScaling:
        (A,b,c,colmulti,rowmulti)=scaling(A, b, c, m, n, type)
    # Αρχικοποίηση (βήμα 0)
    initialization(True, A, b, c, m, n, B, N)
    # Μετρητής επαναλήψεων
    iterations = 1
    while True:
        J = find_J()
        if J==[]:
            (is_optimal,values)=check_optimal(art_vars,m,n,B, N)
            # Αν έχουμε κλιμάκωση υπολογίζουμε τις τιμές των x
            if doScaling:
                for i in range(len(xB)):
                    xB[i] = xB[i] / colmulti[B[i]-1]
                    cB[i] = cB[i] * colmulti[B[i]-1]
            z=round(np.matmul(cB,xB),2)
            state = "Optimal"
            break
        else:
```

```
            # Επιλογή εισερχόμενης μεταβλητής
            l=choose_incoming(J)
            # Υπολογισμός συνόλου I+
            Iplus = find_Iplus(A, N, l, b)
            if Iplus==[]:
                state = "unbounded"
                z = None
                break
            else:
                # Υπολογισμός εξερχόμενης μεταβλητής
                (r, pos) = choose_outcoming(B, Iplus)
                # Εφαρμογή pivot
                pivot(B, N, l, r)
                iterations += 1
                initialization(False,  A, b, c, m, n, B, N)
    return (iterations, state, z)
```

Η συνάρτηση *simplex_revised*, δέχεται διάφορες παραμέτρους, οι οποίες περιγράφονται πιο κάτω:

- **problemname**, το όνομα του προβλήματος.
- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών.
- **b**, ο πίνακας διάστασης mx1 που περιέχει τους σταθερούς όρους στους περιορισμούς.
- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.
- **doScaling**, μεταβλητή λογικού τύπου (boolean), η οποία όταν είναι ίση με True, τότε κάνει τον αλγόριθμο να χρησιμοποιεί τεχνικές κλιμάκωσης.
- **type**, ο τύπος της τεχνικής κλιμάκωσης που χρησιμοποιείται, για την περίπτωση που το όρισμα doScaling έχει την τιμή True. Εάν η παράμετρος type έχει την τιμή 1, τότε χρησιμοποιείται η τεχνική της ισορρόπησης, ενώ εάν έχει την τιμή 2, τότε χρησιμοποιείται γεωμετρική κλιμάκωση.
- **M,** η μεγάλη τιμή που θα πρέπει να χρησιμοποιηθεί κατά την εφαρμογή της μεθόδου του μεγάλου M, για την εύρεση μίας αρχικής βασικής εφικτής λύσης, από όπου θα ξεκινήσει ο αλγόριθμος για την εύρεση της βέλτιστης λύσης.

Η συνάρτηση *simplex_revised*, επιστρέφει τα πιο κάτω αποτελέσματα:

- **iterations**, που είναι ο αριθμός των επαναλήψεων, μέχρι να τερματιστεί ο αλγόριθμος.
- **state**, το οποίο καθορίζει τον τύπο του προβλήματος, εάν δηλαδή είναι βέλτιστο (optimal) ή αν είναι απεριόριστο (unbounded).
- **z**, η τιμή της αντικειμενικής συνάρτησης που αντιστοιχεί στη βέλτιστη λύση που υπολογίστηκε από τον αλγόριθμο, εάν το πρόβλημα είναι βέλτιστο.

Η συνάρτηση *simplex_revised*, η οποία υλοποιεί τον αναθεωρημένο αλγόριθμο Simplex, κάνει χρήση διάφορων άλλων συναρτήσεων, οι οποίες μπορούν να χρησιμοποιηθούν και από τις υλοποιήσεις άλλων αλγορίθμων τύπου Simplex. Τέτοιες συναρτήσεις είναι οι εξής:

- **initialization**, για την αρχικοποίηση των παραμέτρων του προβλήματος και την εκτέλεση των βασικών πράξεων που χρειάζονται για την εκκίνηση της εκτέλεσης κάθε επανάληψης.
- **bigM_method**, για την υλοποίηση της μεθόδου του μεγάλου M και την εύρεση μίας αρχικής βασικής εφικτής λύσης.
- **scaling**, για την υλοποίηση των τεχνικών κλιμάκωσης, ανάλογα με την τιμή της παραμέτρου doScaling και της παραμέτρου type.
- **check_optimal**, για τον έλεγχο της βελτιστότητας της τρέχουσας λύσης. Αν η συνάρτηση αυτή επιστρέψει την τιμή True, τότε αυτό σημαίνει ότι η τρέχουσα λύση είναι βέλτιστη και ο αλγόριθμος σταματάει.
- **find_Iplus**, για την εύρεση του συνόλου I_+ που χρησιμοποιεί ο αναθεωρημένος αλγόριθμος Simplex. Αν το σύνολο αυτό είναι κενό, τότε το πρόβλημα είναι απεριόριστο.
- **choose_incoming**, για την εύρεση της εισερχόμενης μεταβλητής στο σύνολο B των βασικών μεταβλητών.
- **choose_outcoming**, για την εύρεση της εξερχόμενης μεταβλητής από το σύνολο B των βασικών μεταβλητών.

- **pivot**, για την εφαρμογή της περιστροφής με βάση την εισερχόμενη και την εξερχόμενη μεταβλητή που επιλέχτηκε από τις προηγούμενες συναρτήσεις.

Κάποιες από τις συναρτήσεις αυτές περιγράφονται συνοπτικά στη συνέχεια.

3.3.2 Δυϊκός Αλγόριθμος Simplex

Ο δυϊκός αλγόριθμος Simplex υλοποιείται σε γλώσσα Python, με την ανάπτυξη μίας συνάρτησης με το όνομα *dual_simplex*, η οποία εμφανίζεται πιο κάτω:

```
def dual_simplex(problemname, A, b, c, m, n):
    B=np.array([5, 6, 7, 8])
    N=np.array([1, 2, 3, 4])
    linear_problem.show_problem(problemname, A, b, c, B, N)
    # Αρχικοποίηση (βήμα 0)
    initialization(True, A, b, c, m, n, B, N)
    # Μετρητής επαναλήψεων
    iterations = 1
    show_iteration(iterations)
    while True:
        # Βήμα 1 (Έλεγχος βελτιστότητας)
        (is_optimal, r, values) = check_optimal_dual(m, n, B, N)
        if is_optimal == True:
            show_opt_values(is_optimal, values, B, N, n, cB, xB)
            break
        # Βήμα 2α
        k = B[r]
        # Εξερχόμενη μεταβλητή: k
        (t,l) = find_incoming_dual(A, B, r, N)
        # Εισερχόμενη μεταβλητή: l
        # Περιστροφή
        pivot(B, N, t, r)
        iterations += 1
        initialization(True,  A, b, c, m, n, B, N)
        show_iteration(iterations)
```

Η συνάρτηση *dual_simplex,* δέχεται διάφορες παραμέτρους, οι οποίες περιγράφονται πιο κάτω:

- **problemname**, το όνομα του προβλήματος
- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών
- **b**, ο πίνακας διάστασης mx1 που περιέχει τους σταθερούς όρους στους περιορισμούς.
- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.

Όμοια με την περίπτωση του αναθεωρημένου αλγόριθμου Simplex, η συνάρτηση *dual_simplex,* χρησιμοποιεί άλλες συναρτήσεις ώστε να επιτελέσει τις απαραίτητες λειτουργίες για την εφαρμογή του αλγορίθμου, όπως:

- **initialization**, για την αρχικοποίηση των παραμέτρων του προβλήματος και την εκτέλεση των βασικών πράξεων που χρειάζονται.
- **show_iteration**, για την εμφάνιση στην οθόνη των δεδομένων μίας επανάληψης του αλγορίθμου.
- **check_optimal_dual**, για τον έλεγχο εάν η τρέχουσα λύση είναι βέλτιστη.
- **show_opt_values**, για την εμφάνιση στην οθόνη της βέλτιστης λύσης.
- **find_incoming_dual**, για την εύρεση της εισερχόμενης μεταβλητής που θα χρησιμοποιήσει ο αλγόριθμος.

3.3.3 Υλοποίηση της μεθόδου του μεγάλου Μ

Για την εύρεση της αρχικής βασικής εφικτής λύσης που απαιτείται κατά την ανάπτυξη των αλγορίθμων Simplex, υλοποιήθηκε σε Python η μέθοδος του μεγάλου Μ. Ο κώδικας για την υλοποίηση της μεθόδου εμφανίζεται παρακάτω:

```
def bigM_method(M, A, c, m, n):
    # Λίστα με τις τεχνητές μεταβλητές που προστίθενται κατά την
    # εφαρμογή της μεθόδου του μεγάλου Μ
    art_vars = []
    # Αρχικά η βάση είναι κενή
    B = np.array([])
    # Ελέγχουμε τις μοναδιαίες στήλες για τιμές
    # του pos=0,1,2,...
    for pos in range(0,m):
        (A, c, B, new_var) = check_I(pos, M, A, c, m, n, B)
        if new_var != -1:
            art_vars.append(new_var)
            n = new_var
    # Η βάση που έχουμε με βάση τις μοναδιαίες
    B = B.astype(int)
    # Λίστα με τις μεταβλητές που δεν υπάρχουν στο B
    N = np.array([])
    for i in range(n):
        # Εάν μία μεταβλητή δεν υπάρχει στο B τότε
        # την προσθέτουμε στο N, δηλαδή είναι μη βασική
        if i+1 not in B:
            N = np.append(N, [i+1])
    N = N.astype(int)
    return (art_vars, A, c, n, B, N)
```

Η συνάρτηση *bigM_method*, δέχεται τις πιο κάτω παραμέτρους:

- **M**, η τιμή ενός μεγάλου αριθμού που θα χρησιμοποιηθεί από τη μέθοδο.
- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών.
- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.

Η συνάρτηση, αφού κάνει τους κατάλληλους υπολογισμούς, επιστρέφει τα εξής αποτελέσματα:

- **art_vars**, η λίστα με τις τεχνητές μεταβλητές που δημιουργούνται κατά την εφαρμογή της μεθόδου.
- **A**, η νέα μορφή του πίνακα A.
- **c**, η νέα τιμή του πίνακα c.
- **n**, η νέα τιμή του πλήθους μεταβλητών n.
- **B**, το σύνολο των βασικών μεταβλητών.
- **N**, το σύνολο των μη βασικών μεταβλητών.

Με βάση τις τιμές που επιστρέφει η συνάρτηση *bigM_method*, μπορεί είτε ο πρωτεύων αλγόριθμος Simplex, είτε ο αναθεωρημένος αλγόριθμος Simplex, να προχωρήσει στην εφαρμογή του αλγορίθμου, για την εύρεση της βέλτιστης λύσης.

3.3.4 Υλοποίηση τεχνικών κλιμάκωσης

Για την εφαρμογή των τεχνικών κλιμάκωση, αναπτύχθηκε ο πιο κάτω κώδικας σε γλώσσα Python:

```
def scaling(A, b, c, m, n, type):
    if type == 1:   # Κλιμάκωση ισορρόπησης
        colmulti = np.amax(np.absolute(A), axis=0)
    elif type == 2: # Κλιμάκωση γεωμετρικού μέσου
        # Εύρεση του μεγαλύτερου κατ' απόλυτη
        # τιμή της κάθε στήλης
        colmax = np.amax(np.absolute(A), axis=0)
        # Εύρεση του μικρότερου μη μηδενικού κατ' απόλυτη
        # τιμή κάθε στήλης
        colmin = np.zeros(n)
        colmulti = np.zeros(n)
        for j in range(n):
            min = colmax[j]
            for i in range(m):
                if A[i][j] != 0 and abs(A[i][j])<min:
                    min = abs(A[i][j])
            colmin[j] = min
            colmulti[j] = math.sqrt(colmax[j] * colmin[j])
    x=1./colmulti
    for i in range(m):
        c[i] = c[i] * x[i]
        for j in range(n):
```

```
            A[i][j] = A[i][j] * x[j]
    if type == 1:
        rowmulti = np.amax(np.absolute(A), axis=1)
    elif type == 2:
        # Εύρεση του μεγαλύτερου κατ' απόλυτη τιμή
        # της κάθε γραμμής
        rowmax = np.amax(np.absolute(A), axis=1)
        # Εύρεση του μικρότερου μη μηδενικού κατ' απόλυτη
        # τιμή κάθε γραμμής
        rowmin = np.zeros(m)
        rowmulti = np.zeros(m)
        for i in range(m):
            min = rowmax[i]
            for j in range(n):
                if A[i][j] != 0 and abs(A[i][j])<min:
                    min = abs(A[i][j])
            rowmin[i] = min
            rowmulti[i] = math.sqrt(rowmax[i] * rowmin[i])
    x=1./rowmulti
    for i in range(m):
        b[i] = b[i] * x[i]
        for j in range(n):
            A[i][j]= A[i][j] * x[i]
    return (A, b, c, colmulti, rowmulti)
```

Η συνάρτηση *scaling*, δέχεται τις πιο κάτω παραμέτρους:

- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών.
- **b**, ο πίνακας διάστασης mx1 που περιέχει τους σταθερούς όρους στους περιορισμούς.
- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.
- **type**, ο τύπος της τεχνικής κλιμάκωσης. Εάν είναι type=1 τότε χρησιμοποιείται η τεχνική της ισορρόπησης, ενώ εάν είναι type=2, τότε χρησιμοποιείται η τεχνική του γεωμετρικού μέσου.

Η συνάρτηση επιστρέφει τα παρακάτω αποτελέσματα:

- **A**, η νέα τιμή του πίνακα A μετά από τις πράξεις της κλιμάκωσης.
- **b**, η νέα τιμή του πίνακα b μετά από τις πράξεις της κλιμάκωσης.
- **c**, η νέα τιμή του πίνακα c μετά από τις πράξεις της κλιμάκωσης.
- **colmulti**, η λίστα με τους πολλαπλασιαστές των στηλών κατά την εφαρμογή της μεθόδου.

- **rowmulti**, η λίστα με τους πολλαπλασιαστές των γραμμών κατά την εφαρμογή της μεθόδου.

Οι νέες τιμές για τους πίνακες A, b και c, χρησιμοποιούνται στη συνέχεια κατά την εφαρμογή της μεθόδου Simplex για την εύρεση της βέλτιστης λύσης. Στη συνέχεια, χρησιμοποιούνται οι πολλαπλασιαστές των γραμμών και των στηλών, για τον υπολογισμό της λύσης του αρχικού προβλήματος.

4 Υπολογιστική Μελέτη

4.1 Μετροπρογράμματα

Όταν δημιουργούνται νέοι αλγόριθμοι ή νεότερες υλοποιήσεις ενός υπάρχοντος αλγορίθμου, συχνά επιθυμούμε να συγκρίνουμε την επίδοσή τους, σε σχέση με άλλους αλγορίθμους ή άλλες υλοποιήσεις των αλγορίθμων αυτών. Με άλλα λόγια, πραγματοποιείται συχνά μία υπολογιστική μελέτη για τη σύγκριση διαφορετικών αλγορίθμων ή διαφορετικών υλοποιήσεων. Η σύγκριση αυτή γίνεται συνήθως με βάση δύο κριτήρια:

- τον αριθμό επαναλήψεων των αλγορίθμων μέχρι να βρεθεί η βέλτιστη λύση του προβλήματος, εάν τέτοια λύση υπάρχει.
- τον χρόνο CPU που απαιτείται για την ολοκλήρωση του αλγορίθμου.

Για να γίνει αυτό όμως, θα πρέπει τα προγράμματα να εκτελεστούν ως προς το ίδιο σύνολο δεδομένων (test data sets). Με τον όρο μετροπρογράμματα (benchmarks) αναφερόμαστε στα προβλήματα ή δεδομένα τα οποία χρησιμοποιούνται κατά τη διαδικασία της σύγκρισης διαφορετικών αλγορίθμων μεταξύ τους. Τα δεδομένα αποθηκεύονται σε μία συγκεκριμένη μορφή ώστε να μπορούν να διαβαστούν από διαφορετικά προγράμματα. Για την περίπτωση των προβλημάτων γραμμικού προγραμματισμού, συχνά χρησιμοποιείται η αποθήκευση των δεδομένων σε μορφή αρχείων mps.

Τα πεδία σε ένα αρχείο μορφής mps, δεν διαχωρίζονται μεταξύ τους με βάση κάποιο διαχωριστικό (separator) αλλά ξεκινούν από συγκεκριμένες στήλες της κάθε γραμμής του αρχείου. Πιο συγκεκριμένα, τα πεδία των δεδομένων ξεκινούν στις στήλες 1, 5, 15, 25, 40 και 50 της κάθε γραμμής του αρχείου. Επιπρόσθετα, ένα αρχείο mps, διαχωρίζεται σε συγκεκριμένους τομείς, ανάλογα με το είδος των δεδομένων που αποθηκεύονται. Έτσι, διακρίνουμε τους πιο κάτω τομείς-τμήματα (Gay 1985):

- **NAME**, το οποίο περιλαμβάνει το όνομα του προβλήματος. Η αποθήκευση του ξεκινάει από τη στήλη 15.
- **ROWS**, όπου περιέχεται ο τύπος του κάθε περιορισμού, δηλαδή αν πρόκειται για ανισότητα, ισότητα ή αν πρόκειται για την αντικειμενική συνάρτηση. Τα

σύμβολα που μπορούν να χρησιμοποιηθούν στον τομέα αυτόν, είναι αυτά τα οποία περιέχει ο Πίνακας 4-1. Τα ονόματα των περιορισμών είναι αποθηκευμένα στις στήλες 5 μέχρι 12 και το είδος τους στις στήλες 2 μέχρι 3.

- **COLUMNS**, στο οποίο έχουμε τα ονόματα των μεταβλητών, μαζί με τις μη μηδενικές τιμές τους, τόσο στην αντικειμενική συνάρτηση, όσο και στους περιορισμούς. Κάθε γραμμή αυτού του τμήματος περιλαμβάνει το όνομα της μεταβλητής στις στήλες 5 μέχρι 12, το όνομα του περιορισμού στις στήλες 15μέχρι 22 και την τιμή του συντελεστή $\alpha_{ij}(c_j)$ στις στήλες 25 μέχρι 36. Σε περίπτωση που μια μεταβλητή έχει περισσότερα του ενός μη μηδενικά στοιχεία, μπορούν να χρησιμοποιηθούν για την αποθήκευσή τους οι στήλες 40 μέχρι 47 για το όνομα του περιορισμού και οι στήλες 50 μέχρι 61 για την τιμή του συντελεστή. Η χρησιμοποίηση των στηλών 40 μέχρι 47 και 50 μέχρι 61 είναι προαιρετική (optional).
- **RHS**, το οποίο περιλαμβάνει τους σταθερούς όρους που αντιστοιχούν στο δεξιό μέρος των περιορισμών. Στις στήλες 5 μέχρι 12 είναι αποθηκευμένο το όνομα του δεξιού μέρους, στις στήλες 15 μέχρι 22 το όνομα του περιορισμού και στις στήλες 25 μέχρι 36 οι μη μηδενικές τιμές του δεξιού μέρους.
- **RANGES**, το οποίο είναι προαιρετικό και χρησιμοποιείται για τις περιπτώσεις όπου θέλουμε για κάποιους περιορισμούς, ώστε να προσδιορίσουμε ένα χαμηλότερο και ένα υψηλότερο όριο για την τιμή που μπορούν να έχουν. Η μορφή αποθήκευσης των τιμών είναι όμοια με τα τμήματα COLUMNS και RHS εκτός από τις στήλες 5 μέχρι 12 που αποθηκεύονται τα ονόματα των περιοχών.
- **BOUNDS**, το οποίο είναι επίσης προαιρετικό και αναφέρεται στα όρια τιμών που μπορεί να έχουν οι μεταβλητές. Εάν δεν υπάρχει τμήμα BOUNDS, τότε θεωρούμε, εξ' ορισμού, ότι οι επιτρεπτές για τις μεταβλητές x_i, είναι $x_i \geq 0$. Στις στήλες 2 μέχρι 3 αποθηκεύεται το είδος του ορίου (LO, UP, FR, FX, MI, BV), στις στήλες 5 μέχρι 12 το όνομα του ορίου, στις στήλες 15 μέχρι 22 το όνομα της μεταβλητής και στις στήλες 25 μέχρι 36 η τιμή του ορίου.
- **ENDATA**, το οποίο υποδηλώνει το τέλος των δεδομένων του γραμμικού προβλήματος.

Πίνακας 4-1: Σύμβολα που χρησιμοποιούνται σε ένα αρχείο mps

Τύπος	**Σύμβολο**	**Επεξήγηση**
=	E	Έχουμε περιορισμό ισότητας
≤	L	Έχουμε περιορισμό όπου θεωρούμε ότι υπάρχει το σύμβολο μικρότερο ή ίσο
≥	G	Έχουμε περιορισμό όπου θεωρούμε ότι υπάρχει το σύμβολο μεγαλύτερο ή ίσο
Αντικειμενική συνάρτηση	N	Οι αντίστοιχες τιμές αναφέρονται στην αντικειμενική συνάρτηση

Σε ένα αρχείο δεδομένων, μορφής mps, επιτρέπεται να υπάρχουν και γραμμές με σχόλια, οι οποίες θα πρέπει να ξεκινούν με τον χαρακτήρα «*». Στην Εικόνα *4-1* εμφανίζεται η δομή ενός μετροπρογράμματος σε μορφή mps.

α/α πεδίου	1	2	3	4	5	6
στήλες	2-3	5-12	15-22	25-36	40-47	50-61
	NAME		Όνομα προβλήματος			
	ROWS					
	Είδος περιορισμού	Όνομα περιορισμού				
	COLUMNS					
		Όνομα μεταβλητής	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	RHS					
		Όνομα δεξιού μέρους	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	RANGES (προαιρετικό)					
		Όνομα περιοχής	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	BOUNDS					
	Είδος ορίου	Όνομα ορίου	Όνομα περιορισμού	Τιμή	Όνομα περιορισμού	Τιμή
	ENDATA					

Εικόνα 4-1: Δομή ενός μετροπρογράμματος

Στα πλαίσια της παρούσας διπλωματικής εργασίας, αναπτύχθηκε συνάρτηση, σε γλώσσα προγραμματισμού Python, η οποία διαβάζει τα δεδομένα ενός γραμμικού προβλήματος, από ένα αρχείο μορφής mps και τα αποθηκεύει σε μορφή πινάκων, οι οποίοι είναι κατάλληλη για την εφαρμογή ενός αλγόριθμου τύπου Simplex, ώστε να βρεθεί η βέλτιστη λύση για το πρόβλημα αυτό. Ο κώδικας για τη λειτουργία αυτή, εμφανίζεται πιο κάτω:

```
# Συνάρτηση που διαβάζει αρχείο που περιέχει τα δεδομένα ενός
# γραμμικού προβλήματος σε μορφή mps
def read_file(filename):
    # Σύνολο περιορισμών και μεταβλητών
    constraints = {}
    variables = {}
    # Ανοίγουμε το αρχείο για ανάγνωση
    file = open(filename, "r")
    # Διαβάζουμε το περιεχόμενο του αρχείου
    text = file.read()
    text1 = text
    # Διαβάζουμε την πρώτη γραμμή
    [line, text] = text.split('\n',1)
    # Διαβάζουμε το όνομα του προβλήματος
    problemname = line[14:]
    i=0
    var_pos=0
    list_coef = []
    # Παραλείπουμε τις γραμμές μέχρι την γραμμή "COLUMNS"
    # όπου υπάρχουν οι μεταβλητές του προβλήματος
    while line != "COLUMNS":
        # Διαβάζουμε την επόμενη γραμμή
        [line, text] = text.split('\n',1)
    # Διαβάζουμε την πρώτη γραμμή της περιοχής COLUMNS
    [line, text] = text.split('\n',1)
    # Διαβάζουμε τους περιορισμούς μέχρι να συναντήσουμε RHS
    while line != "RHS":
        # Διαβάζουμε το όνομα της μεταβλητής
        variable_name = line[4:12].strip()
        # Διαβάζουμε το όνομα του περιορισμού όπου
        # εμφανίζεται η μεταβλητή
        constraint_name = line[14:22].strip()
        # Διαβάζουμε το συντελεστή της μεταβλητής
        variable_coef = line[24:36].strip()
        # Αν η μεταβλητή δεν έχει ξαναδιαβαστεί την εισάγουμε
        # στο σύνολο μεταβλητών variables
        if variable_name not in variables:
```

```
            variables[variable_name] = var_pos
            # προσθήκη στη λίστα των συντελεστών
            list_coef.append([(constraint_name,
                                    variable_coef)])
            var_pos += 1
        else:   # Αν υπάρχει ήδη η μεταβλητή
            # Παίρνουμε τη θέση της μεταβλητής
            pos = variables[variable_name]
            # προσθήκη στη λίστα των συντελεστών
            list_coef[pos].append((constraint_name,
                                     variable_coef))
        # Διαβάζουμε το υπόλοιπο της γραμμής
        constraint_name = line[39:47].strip()
        variable_coef = line[49:61].strip()
        # Αν το υπόλοιπο της γραμμής δεν είναι κενό
        if constraint_name != '':
            pos = variables[variable_name]
            list_coef[pos].append((constraint_name,
                                       variable_coef))
        [line, text] = text.split('\n',1)
    # Διαβάζουμε από την αρχή ώστε να πάρουμε τους περιορισμούς του προβλήματος
    text = text1
    # Διαβάζουμε τις πρώτες γραμμές
    [line, text] = text.split('\n',1)
    [line, text] = text.split('\n',1)
    [line, text] = text.split('\n',1)
    # Διαβάζουμε τους περιορισμούς
    while line != "COLUMNS":
        # Ο τύπος του περιορισμού
        constraint_type = line[1:3].strip()
        # Το όνομα του περιορισμού
        constraint_name = line[4:12].strip()
        # περίπτωση ισότητας
        if constraint_type == 'E':
            constraints[constraint_name] = (i,constraint_type)
            i = i+1
```

```
        # περίπτωση μικρότερο ή ίσο
        elif constraint_type == 'L':
            constraints[constraint_name] = (i,constraint_type)
            i = i+1
            variable_name = "NEWVAR"+str(var_pos)
            variables[variable_name] = var_pos
            var_pos += 1
            variable_coef = 1
            list_coef.append([(constraint_name,
                                variable_coef)])
        # περίπτωση μεγαλύτερο ή ίσο
        elif constraint_type == 'G':
            constraints[constraint_name] = (i,constraint_type)
            i = i+1
            variable_name = "NEWVAR"+str(var_pos)
            variables[variable_name] = var_pos
            var_pos += 1
            variable_coef = -1
            list_coef.append([(constraint_name,
                                  variable_coef)])
        else:
            constraints[constraint_name] = (-1,'N')
        # Διαβάζουμε την επόμενη γραμμή
        [line, text] = text.split('\n',1)

    # Πλήθος περιορισμών
    m = len(constraints)-1
    # Πλήθος μεταβλητών
    n=len(variables)
    # Δημιουργούμε τους πίνακες του προβλήματος
    # Αρχικά στον πίνακα βάζουμε μηδενικά
    A = np.zeros((m,n))
    b = np.zeros(m)
    c = np.zeros(n)
    # Για όλες τις μεταβλητές
    for v in variables.keys():
        # Η στήλη της μεταβλητής στον πίνακα A
```

```
        j = variables[v]
        # Η λίστα με τους συντελεστές της μεταβλητής
        L=list_coef[j]
        # Για όλα τα στοιχεία της λίστας
        for t in L:
            # Ο περιορισμός
            con = constraints[t[0]]
            # Βρίσκουμε σε ποια γραμμή είναι ο περιορισμός
            i=con[0]
            # Αν έχουμε περιορισμό
            if i != -1:
                # Ο συντελεστής
                A[i,j] = t[1]
            else: # Αν έχουμε αντικειμενική συνάρτηση
                c[j] = t[1]
    [line, text] = text.split('\n',1)
    while line != "RHS":
        [line, text] = text.split('\n',1)
    [line, text] = text.split('\n',1)
    rhs = {}
    i=0
    list_coef=[]
    while line != "ENDATA":
        # Όνομα του RHS
        rhs_name = line[4:12].strip()
        # Όνομα περιορισμού
        constraint_name = line[14:22].strip()
        # Τιμή του συντελεστή
        rhs_coef = line[24:36].strip()
        if rhs_name not in rhs:
            rhs[rhs_name] = i
            # Προσθήκη στη λίστα
            list_coef.append([(constraint_name, rhs_coef)])
            i = i+1
        else:
            pos = rhs[rhs_name]
            # Προσθήκη στη λίστα
```

```
            list_coef[pos].append((constraint_name, rhs_coef))
        constraint_name = line[39:47].strip()
        rhs_coef = line[49:61].strip()
        if constraint_name != '':
            pos = rhs[rhs_name]
            list_coef[pos].append((constraint_name, rhs_coef))
        [line, text] = text.split('\n',1)
    # Εισάγουμε τις τιμές των συντελεστών στον πίνακα b
    for v in rhs.keys():
        L=list_coef[rhs[v]]
        for u in L:
            con = u[0]
            i = constraints[con][0]
            if i!=-1:
                b[i] = u[1]
                if b[i] < 0:
                    constr_type= list( constraints.values())
                                                        [i+1][1]
                    if constr_type == 'L':
                        for k in range(n):
                            A[i, k] = -A[i, k]
                        b[i] = -b[i]
    return (problemname, A, b, c, m, n)
```

Η συνάρτηση *read_file*, δέχεται τις πιο κάτω παραμέτρους:

- **filename**, που αντιστοιχεί στο όνομα του αρχείου το οποίο περιέχει τα δεδομένα μορφής mps.

Η συνάρτηση επιστρέφει τα πιο κάτω αποτελέσματα:

- **problemname**, το όνομα του προβλήματος, όπως αυτό είναι αποθηκευμένο στο αρχείο mps.
- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών.
- **b**, ο πίνακας διάστασης mx1 που περιέχει τους σταθερούς όρους στους περιορισμούς.

- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.

Με παρόμοιο τρόπο μπορεί να γίνει και η αποθήκευση ενός προβλήματος σε μορφή mps. Ο κώδικας για την αποθήκευση αυτή περιέχεται στη συνάρτηση *store_problem*, ο κώδικας της οποίας φαίνεται πιο κάτω:

```
def store_problem(problemname, A, b, c, m, n):
    filename = problemname + ".mps"
     # Ανοίγουμε το αρχείο για εγγραφή
    file = open(filename, "w")
    file.write('%-14s%s\n' % ("NAME", problemname))
    file.write("ROWS\n")
    file.write(" N  OBJ\n")
    for i in range(m):
        file.write(" E  CONSTR")
        file.write(str(i+1))
        file.write("\n")
    file.write("COLUMNS\n")
    for i in range(n):
        file.write('%4s%s%-9s%-8s%+14s' % (" ", "X",
                                str(i+1), "OBJ", c[i]))
        file.write('%s%-8s%s%+11s\n' % ("   ","CONSTR1","   ",
                                A[0][i]))
        for k in range(int((m-1)/2)):
            file.write('%4s%s%-9s%-8s%+14s' % (" ", "X",
                  str(i+1), "CONSTR"+str(2*(k+1)), A[2*k+1][i]))
            file.write('%s%-8s%s%+11s\n' %
              ("   ","CONSTR"+str(2*(k+1)+1),"   ", A[2*k+2][i]))
        if m%2 == 0:
            file.write('%4s%s%-9s%-8s%+14s\n' % (" ", "X",
               str(i+1), "CONSTR"+str(m), A[m-1][i]))
    file.write("RHS\n")
    for k in range(int(m/2)):
        file.write('%4s%-10s%-8s%+14s' % (" ", "RHS1",
```

```
                "CONSTR"+str(2*k+1), b[2*k]))
        file.write('%s%-8s%s%+11s\n' %
                ("   ","CONSTR"+str(2*k+2),"   ", b[2*k+1]))
    if m%2 == 1:
        file.write('%4s%-10s%-8s%+14s\n' % (" ", "RHS1",
                "CONSTR"+str(m), b[m-1]))
    file.write("ENDATA")
    file.close()
```

Η συνάρτηση δέχεται τις πιο κάτω παραμέτρους:

- **problemname**, το όνομα του προβλήματος, όπως αυτό θα είναι αποθηκευμένο στο αρχείο mps.
- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών.
- **b**, ο πίνακας διάστασης mx1 που περιέχει τους σταθερούς όρους στους περιορισμούς.
- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.

4.1.1 Παραδείγματα γραμμικών προβλημάτων σε μορφή mps

Στην παράγραφο αυτή παρουσιάζονται κάποια γραμμικά προβλήματα και ο τρόπος αποθήκευσής τους σε μορφή mps. Τα προβλήματα αυτά επιλύθηκαν κατά τη δοκιμή του προγράμματος, έτσι ώστε να εξασφαλιστεί ότι προκύπτουν τα σωστά αποτελέσματα.

Παράδειγμα 1

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα.

$\min -x_1 - x_2$

μ.π.

$$2x_1 + x_2 \leq 4$$

$$x_1 + 2x_2 \leq 4$$

$$x_1 \geq 0,\ x_2 \geq 0$$

Το πρόβλημα αυτό είναι βέλτιστο.

Στην Εικόνα 4-2 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM1
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
COLUMNS
    X1        OBJ              -1.   CONSTR1           2.
    X1        CONSTR2           1.
    X2        OBJ              -1.   CONSTR1           1.
    X2        CONSTR2           2.
RHS
    RHS1      CONSTR1           4.   CONSTR2           4.
ENDATA
```

Εικόνα 4-2: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 1

<u>Παράδειγμα 2</u>

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα:

$$\min 5x_1 + 7x_2 + 2x_3 + 8x_4$$

μ.π.

$$x_1 - 2x_2 - 2x_3 + 3x_4 \leq 5$$

$$5x_1 - 2x_2 - x_3 + 5x_4 \leq 3$$

$$3x_1 - 2x_2 + 3x_3 - 3x_4 \leq -1$$

$$-x_2 + 3x_3 \leq -1$$

$x_1 \geq 0$, $x_2 \geq 0$, $x_3 \geq 0$, $x_4 \geq 0$

Το πρόβλημα αυτό είναι βέλτιστο.

Στην Εικόνα 4-3 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM2
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
 L  CONSTR3
 L  CONSTR4
COLUMNS
    X1        OBJ                5.   CONSTR1            1.
    X1        CONSTR2            5.   CONSTR3            3.
    X1        CONSTR4            0.
    X2        OBJ                7.   CONSTR1           -2.
    X2        CONSTR2           -2.   CONSTR3           -2.
    X2        CONSTR4           -1.
    X3        OBJ                2.   CONSTR1           -2.
    X3        CONSTR2           -1.   CONSTR3            3.
    X3        CONSTR4            3.
    X4        OBJ                8.   CONSTR1            3.
    X4        CONSTR2            5.   CONSTR3           -3.
    X4        CONSTR4            0.
RHS
    RHS1      CONSTR1            5.   CONSTR2            3.
    RHS1      CONSTR3           -1.   CONSTR4           -1.
ENDATA
```

Εικόνα 4-3: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 2

Παράδειγμα 3

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα, όπου εμφανίζονται διαφορετικοί τελεστές σύγκρισης στους περιορισμούς (≤, = και ≥):

min $x_1 + x_2 - 4x_3$

μ.π.

$$x_1 + x_2 + 2x_3 \leq 9$$

$$x_1 + x_2 - x_3 = 2$$

$$-x_1 + x_2 + x_3 \geq 4$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$$

Το πρόβλημα αυτό είναι βέλτιστο.

Στην Εικόνα 4-4 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM3
ROWS
 N  OBJ
 L  CONSTR1
 E  CONSTR2
 G  CONSTR3
COLUMNS
    X1        OBJ              1.   CONSTR1           1.
    X1        CONSTR2          1.   CONSTR3          -1.
    X2        OBJ              1.   CONSTR1           1.
    X2        CONSTR2          1.   CONSTR3           1.
    X3        OBJ             -4.   CONSTR1           2.
    X3        CONSTR2         -1.   CONSTR3           1.
RHS
    RHS1      CONSTR1          9.   CONSTR2           2.
    RHS1      CONSTR3          4.
ENDATA
```

Εικόνα 4-4: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 3

Παράδειγμα 3-1

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα, το οποίο είναι παρόμοιο με το πρόβλημα του προηγούμενου παραδείγματος, με τη διαφορά ότι χρησιμοποιείται μόνο ο τελεστής ανισότητας ≤ (μικρότερο ή ίσο) στους περιορισμούς.

$$\min x_1 + x_2 - 4x_3$$

μ.π.

$$x_1 + x_2 + 2x_3 \leq 9$$

$$x_1 + x_2 - x_3 \leq 2$$

$$-x_1 + x_2 + x_3 \leq 4$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0$$

Το πρόβλημα αυτό είναι βέλτιστο.

Στην Εικόνα 4-5 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM3_1
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
 L  CONSTR3
COLUMNS
    X1        OBJ                1.   CONSTR1           1.
    X1        CONSTR2            1.   CONSTR3          -1.
    X2        OBJ                1.   CONSTR1           1.
    X2        CONSTR2            1.   CONSTR3           1.
    X3        OBJ               -4.   CONSTR1           2.
    X3        CONSTR2           -1.   CONSTR3           1.
RHS
    RHS1      CONSTR1            9.   CONSTR2           2.
    RHS1      CONSTR3            4.
ENDATA
```

Εικόνα 4-5: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 3-1

Παράδειγμα 4

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα.

$$\min 200x_1 - 100x_2 + 160x_3 - 80x_4$$

μ.π.

$$100x_1 + 90x_2 - 80x_3 - 20x_4 \leq 10$$

$$80x_1 - 80x_2 + 40x_3 + 40x_4 \leq 5$$

$$-x_1 - 100x_2 + 20x_3 + 10x_4 \leq 20$$

$$2x_1 + 3x_2 + 40x_3 + 30x_4 \leq -10$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Αυτό το πρόβλημα είναι αδύνατο.

Στην Εικόνα 4-6 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM4
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
 L  CONSTR3
 L  CONSTR4
COLUMNS
    X1        OBJ             200.   CONSTR1          100.
    X1        CONSTR2          80.   CONSTR3           -1.
    X1        CONSTR4           2.
    X2        OBJ            -100.   CONSTR1           90.
    X2        CONSTR2         -80.   CONSTR3         -100.
    X2        CONSTR4           3.
    X3        OBJ             160.   CONSTR1          -80.
    X3        CONSTR2          40.   CONSTR3           20.
    X3        CONSTR4          40.
    X4        OBJ             -80.   CONSTR1          -20.
    X4        CONSTR2          40.   CONSTR3           10.
    X4        CONSTR4          30.
RHS
    RHS1      CONSTR1          10.   CONSTR2            5.
    RHS1      CONSTR3          20.   CONSTR4          -10.
ENDATA
```

Εικόνα 4-6: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 4

Παράδειγμα 5

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα, το οποίο είναι παρόμοιο με το πρόβλημα του προηγούμενου παραδείγματος, με τη διαφορά στον σταθερό όρο του τελευταίου περιορισμού. Αυτό έχει σαν αποτέλεσμα το πρόβλημα να είναι βέλτιστο.

$$\min 200x_1 - 100x_2 + 160x_3 - 80x_4$$

μ.π.

$$100x_1 + 90x_2 - 80x_3 - 20x_4 \leq 10$$

$$80x_1 - 80x_2 + 40x_3 + 40x_4 \leq 5$$

$$-x_1 - 100x_2 + 20x_3 + 10x_4 \leq 20$$

$$2x_1 + 3x_2 + 40x_3 + 30x_4 \leq 10$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Στην Εικόνα 4-7 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM5
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
 L  CONSTR3
 L  CONSTR4
COLUMNS
    X1        OBJ             200.   CONSTR1          100.
    X1        CONSTR2          80.   CONSTR3           -1.
    X1        CONSTR4           2.
    X2        OBJ            -100.   CONSTR1           90.
    X2        CONSTR2         -80.   CONSTR3         -100.
    X2        CONSTR4           3.
    X3        OBJ             160.   CONSTR1          -80.
    X3        CONSTR2          40.   CONSTR3           20.
    X3        CONSTR4          40.
    X4        OBJ             -80.   CONSTR1          -20.
    X4        CONSTR2          40.   CONSTR3           10.
    X4        CONSTR4          30.
RHS
    RHS1      CONSTR1          10.   CONSTR2            5.
    RHS1      CONSTR3          20.   CONSTR4           10.
ENDATA
```

Εικόνα 4-7: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 5

Παράδειγμα 6

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα, έχουμε ένα παρόμοιο πρόβλημα, το οποίο είναι επίσης αδύνατο, αλλά χρησιμοποιούνται τελεστές ισότητας στους περιορισμούς:

$$\min 200x_1 - 100x_2 + 160x_3 - 80x_4$$

μ.π.

$$100x_1 + 90x_2 - 80x_3 - 20x_4 = 10$$

$$80x_1 - 80x_2 + 40x_3 + 40x_4 = 5$$

$$-x_1 - 100x_2 + 20x_3 + 10x_4 = 20$$

$$2x_1 + 3x_2 + 40x_3 + 30x_4 = 10$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Στην Εικόνα 4-8 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM6
ROWS
 N  OBJ
 E  CONSTR1
 E  CONSTR2
 E  CONSTR3
 E  CONSTR4
COLUMNS
    X1        OBJ             200.   CONSTR1          100.
    X1        CONSTR2          80.   CONSTR3           -1.
    X1        CONSTR4           2.
    X2        OBJ            -100.   CONSTR1           90.
    X2        CONSTR2         -80.   CONSTR3         -100.
    X2        CONSTR4           3.
    X3        OBJ             160.   CONSTR1          -80.
    X3        CONSTR2          40.   CONSTR3           20.
    X3        CONSTR4          40.
    X4        OBJ             -80.   CONSTR1          -20.
    X4        CONSTR2          40.   CONSTR3           10.
    X4        CONSTR4          30.
RHS
    RHS1      CONSTR1          10.   CONSTR2            5.
    RHS1      CONSTR3          20.   CONSTR4           10.
ENDATA
```

Εικόνα 4-8: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 6

Παράδειγμα 7

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα:

$$\min \; -10x_1 + 30x_2 - 20x_3 + 50x_4$$

μ.π.

$$100x_1 - 5x_2 + 3x_3 + 200x_4 \leq 10$$

$$20x_1 - 100x_2 + 4x_3 + 8x_4 \leq 20$$

$$12x_1 - 10x_2 + 10x_3 + 4x_4 \leq 30$$

$$-8x_1 + 80x_2 + 4x_3 + 14x_4 \leq 40$$

$x_1 \geq 0, x_2 \geq 0, x_3 \geq 0, x_4 \geq 0$

Το πρόβλημα αυτό είναι βέλτιστο.

Στην Εικόνα 4-9 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM7
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
 L  CONSTR3
 L  CONSTR4
COLUMNS
    X1        OBJ              -10.   CONSTR1          100.
    X1        CONSTR2           20.   CONSTR3           12.
    X1        CONSTR4           -8.
    X2        OBJ               30.   CONSTR1           -5.
    X2        CONSTR2         -100.   CONSTR3          -10.
    X2        CONSTR4           80.
    X3        OBJ              -20.   CONSTR1            3.
    X3        CONSTR2            4.   CONSTR3           10.
    X3        CONSTR4            4.
    X4        OBJ               50.   CONSTR1          200.
    X4        CONSTR2            8.   CONSTR3            4.
    X4        CONSTR4           14.
RHS
    RHS1      CONSTR1           10.   CONSTR2           20.
    RHS1      CONSTR3           30.   CONSTR4           40.
ENDATA
```

Εικόνα 4-9: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 7

Παράδειγμα 8

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα.

$$\min \ 25x_1 + 30x_2 + 3x_3 + 5x_4$$

μ.π.

$$-10x_1 + 5x_2 - 3x_3 - 20x_4 \leq -1$$

$$-3x_1 - 10x_2 + 4x_3 - 8x_4 \leq -2$$

$$12x_1 - 10x_2 + 10x_3 - 4x_4 \leq -3$$

$$8x_1 + 80x_2 + 4x_3 + 14x_4 \leq -4$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Το πρόβλημα αυτό είναι αδύνατο.

Στην Εικόνα 4-10 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM8
ROWS
 N  OBJ
 L  CONSTR1
 L  CONSTR2
 L  CONSTR3
 L  CONSTR4
COLUMNS
    X1        OBJ              25.   CONSTR1          -10.
    X1        CONSTR2          -3.   CONSTR3           12.
    X1        CONSTR4           8.
    X2        OBJ              30.   CONSTR1            5.
    X2        CONSTR2         -10.   CONSTR3          -10.
    X2        CONSTR4          80.
    X3        OBJ               3.   CONSTR1           -3.
    X3        CONSTR2           4.   CONSTR3           10.
    X3        CONSTR4           4.
    X4        OBJ               5.   CONSTR1          -20.
    X4        CONSTR2          -8.   CONSTR3           -4.
    X4        CONSTR4          14.
RHS
    RHS1      CONSTR1          -1.   CONSTR2           -2.
    RHS1      CONSTR3          -3.   CONSTR4           -4.
ENDATA
```

Εικόνα 4-10: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 8

Παράδειγμα 9

Θεωρούμε το πιο κάτω γραμμικό πρόβλημα.

$\min \; -x_3 - 2x_4$

μ.π.

$$x_1 + x_3 + x_4 = 5$$

$$-x_2 - 2x_3 - 3x_4 = -3$$

$$x_1 + x_3 + x_4 = 1$$

$$-x_2 - 2x_3 - 3x_4 = 2$$

$$x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0$$

Το πρόβλημα αυτό είναι αδύνατο.

Στην Εικόνα 4-11 παρουσιάζεται η μορφή του πιο πάνω προβλήματος, όπως αυτό αποθηκεύεται σε μορφή mps.

```
NAME          PROBLEM9
ROWS
 N  OBJ
 E  CONSTR1
 E  CONSTR2
 E  CONSTR3
 E  CONSTR4
COLUMNS
    X1        OBJ              0.   CONSTR1           1.
    X1        CONSTR2          0.   CONSTR3           1.
    X1        CONSTR4          0.
    X2        OBJ              0.   CONSTR1           0.
    X2        CONSTR2         -1.   CONSTR3           0.
    X2        CONSTR4         -1.
    X3        OBJ             -1.   CONSTR1           1.
    X3        CONSTR2         -2.   CONSTR3           1.
    X3        CONSTR4         -2.
    X4        OBJ             -2.   CONSTR1           1.
    X4        CONSTR2         -3.   CONSTR3           1.
    X4        CONSTR4         -3.
RHS
    RHS1      CONSTR1          5.   CONSTR2          -3.
    RHS1      CONSTR3          1.   CONSTR4           2.
ENDATA
```

Εικόνα 4-11: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 9

Παράδειγμα 10

Στην Εικόνα 4-12 παρουσιάζεται ένα γραμμικό πρόβλημα, όπως αυτό αποθηκεύεται σε μορφή mps.

Το πρόβλημα αυτό είναι βέλτιστο.

```
NAME          AFIRO
ROWS
 E  R09
 E  R10
 L  X05
 L  X21
 E  R12
 E  R13
 L  X17
 L  X18
 L  X19
 L  X20
 E  R19
 E  R20
 L  X27
 L  X44
 E  R22
 E  R23
 L  X40
 L  X41
 L  X42
```

```
 L  X43
 L  X45
 L  X46
 L  X47
 L  X48
 L  X49
 L  X50
 L  X51
 N  COST
COLUMNS
    X01       X48          .301   R09          -1.
    X01       R10         -1.06   X05           1.
    X02       X21          -1.    R09           1.
    X02       COST          -.4
    X03       X46          -1.    R09           1.
    X04       X50           1.    R10           1.
    X06       X49          .301    R12          -1.
    X06       R13         -1.06    X17           1.
    X07       X49          .313    R12          -1.
    X07       R13         -1.06    X18           1.
    X08       X49          .313    R12          -1.
    X08       R13          -.96    X19           1.
    X09       X49          .326    R12          -1.
    X09       R13          -.86    X20           1.
    X10       X45         2.364    X17          -1.
```

X11	X45	2.386	X18	-1.
X12	X45	2.408	X19	-1.
X13	X45	2.429	X20	-1.
X14	X21	1.4	R12	1.
X14	COST	-.32		
X15	X47	-1.	R12	1.
X16	X51	1.	R13	1.
X22	X46	.109	R19	-1.
X22	R20	-.43	X27	1.
X23	X44	-1.	R19	1.
X23	COST	-.6		
X24	X48	-1.	R19	1.
X25	X45	-1.	R19	1.
X26	X50	1.	R20	1.
X28	X47	.109	R22	-.43
X28	R23	1.	X40	1.
X29	X47	.108	R22	-.43
X29	R23	1.	X41	1.
X30	X47	.108	R22	-.39
X30	R23	1.	X42	1.
X31	X47	.107	R22	-.37
X31	R23	1.	X43	1.
X32	X45	2.191	X40	-1.
X33	X45	2.219	X41	-1.
X34	X45	2.249	X42	-1.

```
    X35       X45          2.279   X43          -1.
    X36       X44           1.4    R23          -1.
    X36       COST          -.48
    X37       X49          -1.     R23           1.
    X38       X51           1.     R22           1.
    X39       R23           1.     COST          10.
RHS
    B         X50          310.    X51          300.
    B         X05           80.    X17           80.
    B         X27          500.    R23           44.
    B         X40          500.
ENDATA
```

Εικόνα 4-12: Γραμμικό πρόβλημα σε μορφή mps - Παράδειγμα 10

4.2 Στατιστικά εκτέλεσης των προγραμμάτων

4.2.1 Δημιουργία τυχαίων προβλημάτων

Για τη δημιουργία των τυχαίων προβλημάτων, δημιουργήθηκε μία συνάρτηση, με το όνομα *generate_random_problem*, η οποία δημιουργεί τυχαία προβλήματα, τα οποία αναλαμβάνει στη συνέχεια να επιλύσει ο αναθεωρημένος αλγόριθμος Simplex που αναπτύχθηκε. Ο κώδικας της συνάρτησης φαίνεται πιο κάτω:

```
def generate_random_problem(m, n, minA, maxA, minb,
                            maxb, minc, maxc, density):
    problemname = "RANDOM_PROBLEM"
    # Δημιουργία πίνακα A
    list1=[]
```

```
        for i in range(0,m):
            row=[]
            for j in range(0,n):
                r2 = random.random()
                if r2<density:
                    r = (maxA-minA+1)*random.random()+minA
                else:
                    r=0
                row.append(r)
            list1.append(row)
        A = np.array(list1)
        # Δημιουργία του πίνακα b
        list2=[]
        for i in range(0,m):
            r = (maxb-minb+1)*random.random()+minb
            list2.append(int(r))
        b = np.array(list2)
        # Δημιουργία του πίνακα c
        list3=[]
        for i in range(0,n):
            r = (maxc-minc+1)*random.random()+minc
            list3.append(int(r))
        c = np.array(list3)
        for i in range(m):
            col = [0 for j in range(m)]
            col[i] = 1
            # print("column=",col)
            A = np.c_[A, np.asarray(col).transpose()]
```

```
            c = np.append(c, [0])

        n += m

    return (problemname, A, b, c, m, n)
```

Η συνάρτηση *generate_random_problem*, δέχεται τις πιο κάτω παραμέτρους:

- **m**, ο αριθμός των περιορισμών που επιθυμούμε να έχουμε στο πρόβλημα.
- **n**, ο αριθμός των μεταβλητών που επιθυμούμε να έχουμε στο πρόβλημα.
- **minA**, η ελάχιστη τιμή που θέλουμε να έχει ένας συντελεστής στον πίνακα A.
- **maxA**, η μέγιστη τιμή που θέλουμε να έχει ένας συντελεστής στον πίνακα A.
- **minb**, η ελάχιστη τιμή που θέλουμε να έχει ένας συντελεστής στον πίνακα b.
- **maxb**, η μέγιστη τιμή που θέλουμε να έχει ένας συντελεστής στον πίνακα b.
- **minc**, η ελάχιστη τιμή που θέλουμε να έχει ένας συντελεστής στον πίνακα c.
- **maxc**, η μέγιστη τιμή που θέλουμε να έχει ένας συντελεστής στον πίνακα c.
- **density**, μία τιμή μεταξύ του 0 και του 1 η οποία υποδηλώνει την πυκνότητα του προβλήματος που θέλουμε να έχουμε. Η πυκνότητα δίνει το ποσοστό των στοιχείων του πίνακα A τα οποία είναι διαφορετικά από 0. Εάν θέλουμε να μην υπάρχει περιορισμός πυκνότητας, τότε ως τιμή της παραμέτρου density, θεωρούμε τη μονάδα.

Η συνάρτηση επιστρέφει τα πιο κάτω αποτελέσματα:

- **problemname**, το όνομα του προβλήματος.
- **A**, ο πίνακας διάστασης mxn, των συντελεστών των περιορισμών.
- **b**, ο πίνακας διάστασης mx1 που περιέχει τους σταθερούς όρους στους περιορισμούς.
- **c**, ο πίνακας διάστασης nx1 που περιέχει τους συντελεστές των μεταβλητών στην αντικειμενική συνάρτηση.
- **m**, το πλήθος των περιορισμών.
- **n**, το πλήθος των μεταβλητών.

4.2.2 Πειράματα και συλλογή αποτελεσμάτων

Για τη δημιουργία των τυχαίων προβλημάτων και στη συνέχεια την επίλυσή τους, χρησιμοποιήθηκε το πρόγραμμα που εμφανίζεται πιο κάτω:

```
import linear_problem
import Simplex_method
import time
minA = -100
maxA = 1000
minb = 10
maxb = 10000
minc = -300
maxc = 900
iterations = 10
file = open("results.txt", "w")
for k in range(50,1001, 50):
    sum_iterations = 0
    sum_time = 0.0
    for i in range(iterations):
        print("Δημιουργία προβλήματος: ",k,"x",k)
        (problemname, A, b, c, m, n) =
           linear_problem.generate_random_problem(k, k,
                      minA, maxA, minb, maxb, minc, maxc, 1)
                print("Εκκίνηση επίλυσης προβλήματος...")
        time1 = time.time()
        (niter, state, z) = Simplex_method.simpex_revised
                 (problemname, A, b, c, m, n, False, 2, 10**6)
        time2 = time.time()
```

```
        time_diff = time2-time1
        sum_iterations += niter
        sum_time += time_diff
        print("Τερματισμός επίλυσης προβλήματος...
         \nΚατάσταση:", state," Αριθμός επαναλήψεων:", niter,
         " Ελάχιστη τιμή:", z," Διάρκεια:",time_diff, "\n")
    avg_iterations = sum_iterations/iterations
    avg_time = sum_time/iterations
file.write(str(m)+"\t"+str(m)+"\t"+str(avg_iterations)+"\t"+st
r(avg_time)+"\n")
```

Το πιο πάνω πρόγραμμα, χρησιμοποιεί τη συνάρτηση *generate_random_problem*, με τιμή παραμέτρου density ίση με την μονάδα. Με άλλα λόγια, αρχικά δεν μας ενδιαφέρει να ελέγξουμε την επίδοση του αλγορίθμου με βάση την πυκνότητα του προβλήματος, παρά μόνο με βάση τη διάσταση του προβλήματος.

Ως όρια των τιμών για τις παραγόμενες παραμέτρους του προβλήματος, χρησιμοποιήθηκαν οι τιμές που εμφανίζονται στον Πίνακας 4-2.

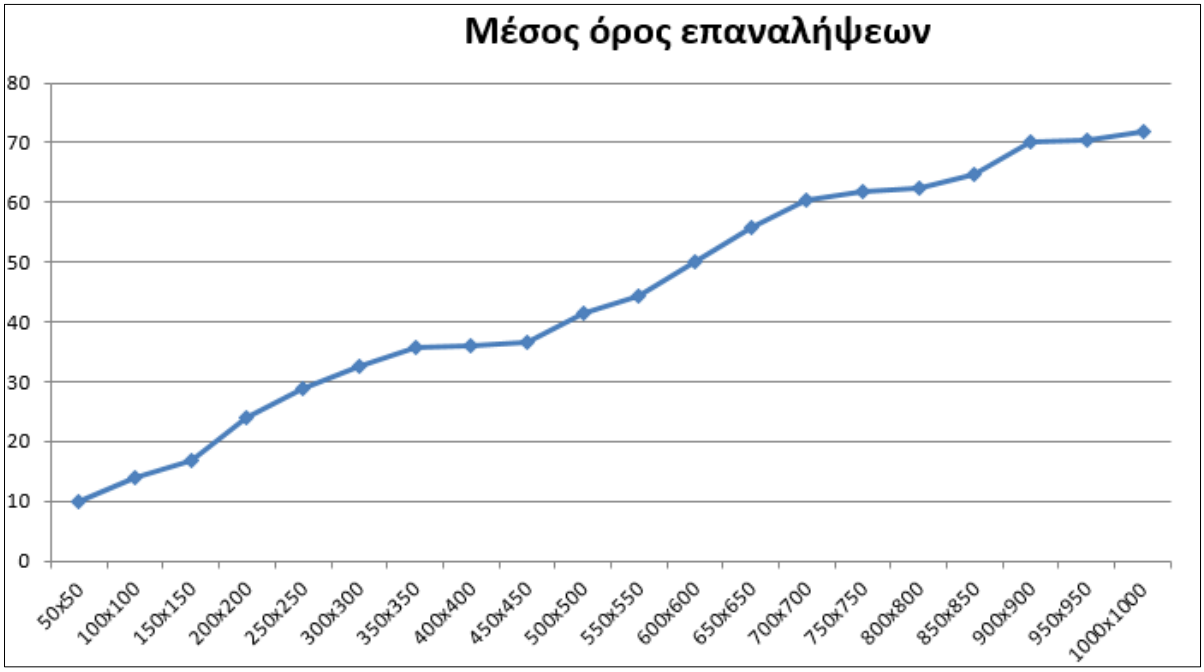
Πίνακας 4-2: Όρια τιμών για τις παραμέτρους των τυχαίων προβλημάτων

	Ελάχιστη τιμή	**Μέγιστη τιμή**
Συντελεστής πίνακα A	-100	1000
Συντελεστής πίνακα b	10	10000
Συντελεστής πίνακα c	-300	900

Τα προβλήματα που παράγονται χωρίζονται σε κατηγορίες με βάση τις διαστάσεις τους και ξεκινούν από την κατηγορία προβλημάτων διάστασης 50x50, δηλαδή με 50 περιορισμούς και 50 μεταβλητές, στη συνέχεια παράγονται προβλήματα διάστασης 100x100, 150x150 κ.ο.κ., μέχρι την κατηγορία προβλημάτων διάστασης1000x1000. Για

κάθε κατηγορία προβλημάτων, δημιουργήθηκαν 10 διαφορετικά προβλήματα με τυχαίο τρόπο και στη συνέχεια καταμετρήθηκε ο χρόνος που απαιτείται για την επίλυσή τους, καθώς και το πλήθος των επαναλήψεων που χρειάζονται.

Στο γράφημα που εμφανίζεται στην Εικόνα 4-13, εμφανίζεται ο μέσος όρος επαναλήψεων που απαιτούνται για κάθε κατηγορία προβλημάτων, όπως αυτά ταξινομήθηκαν με βάση τη διάστασή τους. Στο γράφημα που εμφανίζεται στην Εικόνα 4-14 εμφανίζεται ο αντίστοιχος χρόνος σε δευτερόλεπτα για τις ίδιες κατηγορίες προβλημάτων. Παρατηρούμε ότι όσο αυξάνεται η διάσταση του προβλήματος, έχουμε και αύξηση του αριθμού των επαναλήψεων που απαιτούνται, κάτι που είναι αναμενόμενο, αφού έχουμε μεγαλύτερους πίνακες. Όμοια, όσο αυξάνεται η διάσταση του προβλήματος, αυξάνεται και ο χρόνος που απαιτείται για την επίλυσή του. Βλέπουμε όμως ότι η αύξηση του χρόνου έχει μεγαλύτερη κλίση και εμφανίζεται να αυξάνει εκθετικά. Αυτό είναι επίσης αναμενόμενο, αφού ο χειρισμός μεγαλύτερων πινάκων και η εκτέλεση πράξεων, απαιτεί πολύ περισσότερο χρόνο, όταν έχουμε πίνακες μεγαλύτερης διάστασης.



Εικόνα 4-13: Μέσος όρος επαναλήψεων για κάθε κατηγορία προβλημάτων με βάση τη διάστασή τους

Εικόνα 4-14: Μέσος χρόνος εκτέλεσης για κάθε κατηγορία προβλημάτων με βάση τη διάστασή τους

4.3 Σύγκριση προβλημάτων με βάση την πυκνότητά τους

Στην παράγραφο αυτή έχουμε την παραγωγή τυχαίων προβλημάτων διαφορετικής πυκνότητας. Η πυκνότητα ενός πίνακα καθορίζεται ως το ποσοστό των μηδενικών που εμφανίζονται στον πίνακα αυτόν. Θεωρούμε ότι για ένα γραμμικό πρόβλημα πυκνότητας 0.1, το 90% των τιμών που έχουμε στον πίνακα A είναι μηδενικά. Δηλαδή όσο μεγαλύτερη πυκνότητα έχει ένα πρόβλημα, τόσο λιγότερα μηδενικά έχει.

Για την παραγωγή και επίλυση γραμμικών προβλημάτων διαφορετικών πυκνοτήτων χρησιμοποιήθηκε ο κώδικας που εμφανίζεται πιο κάτω:

```
import linear_problem
import Simplex_method
import time
minA = -100
maxA = 1000
```

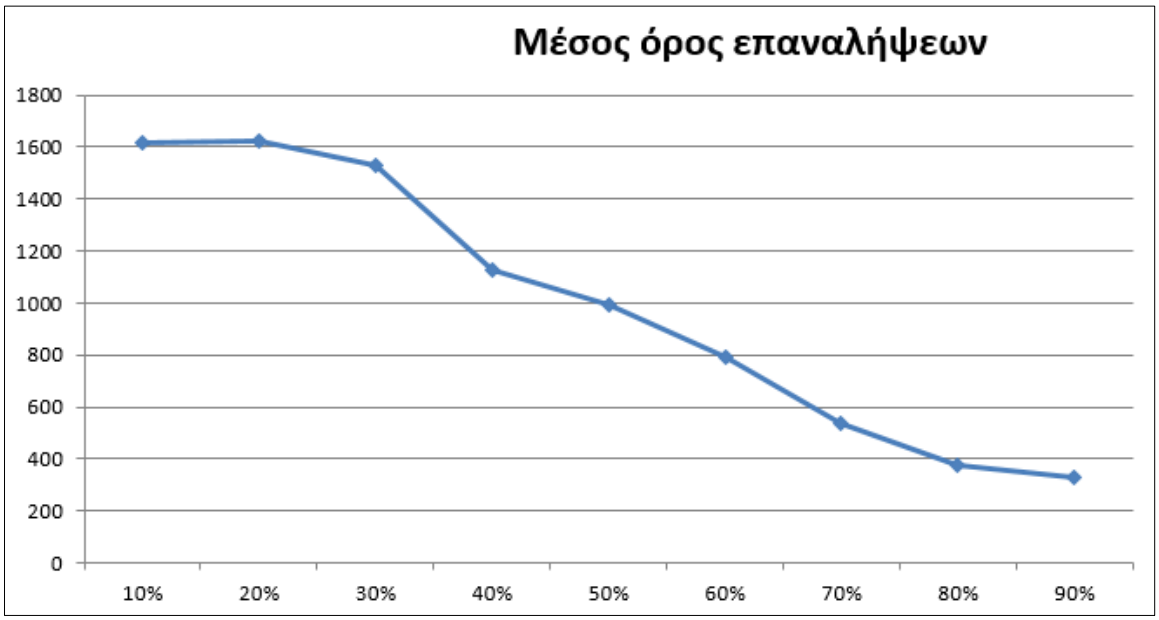
```
minb = 10
maxb = 10000
minc = -300
maxc = 900
# Διάσταση του προβλήματος
m=100
n=100
iterations = 20
file = open("results_dens.txt", "w")
for k in range(10,101,10):
    sum_iterations = 0
    sum_time = 0.0
    for i in range(iterations):
        print("Δημιουργία προβλήματος πυκνότητας: ",k,"%")
        (problemname, A, b, c, m, n)=linear_problem.generate_random_problem(m,
                                        n, minA, maxA, minb, maxb, minc, maxc,k/100)
        print("Εκκίνηση επίλυσης προβλήματος...")
        time1 = time.time()
        (niter, state, z) = Simplex_method.simpex_revised(problemname, A, b, c, m,
                                                        n, False, 2, 10**6)
        time2 = time.time()
        time_diff = time2-time1
        sum_iterations += niter
        sum_time += time_diff
        print("Τερματισμός επίλυσης προβλήματος... \nΚατάσταση:", state,"
            Αριθμός επαναλήψεων:", niter, " Ελάχιστη τιμή:", z," Διάρκεια:",
```

```
                                                    time_diff, "\n")
        avg_iterations = sum_iterations/iterations
        avg_time = sum_time/iterations
        file.write(str(m)+"\t"+str(m)+"\t"+str(avg_iterations)+"\t"+str(avg_time)+"\n")
```

Για την παραγωγή των τυχαίων προβλημάτων, χρησιμοποιήθηκε και πάλι η συνάρτηση *generate_random_problem*, με διαφορετικές όμως τιμές στην παράμετρο πυκνότητα. Για τον σκοπό αυτό έχουμε μία επαναληπτική διαδικασία, κατά την οποία καθορίζονται τιμές πυκνότητας από 0.1 έως 1 με βήμα 0.1, δηλαδή από 10% έως 100%, αυξάνοντας κάθε φορά κατά 10%. Για κάθε μία από τις κατηγορίες αυτές προβλημάτων, δημιουργήθηκαν με τυχαίο τρόπο 10 προβλήματα διάστασης 10x10 και στη συνέχεια καταγράφηκε ο μέσος όρος επαναλήψεων, καθώς και ο μέσος χρόνος που απαιτείται για την επίλυση τους.

Στο γράφημα που εμφανίζεται στην Εικόνα 4-15, εμφανίζεται ο μέσος όρος επαναλήψεων που απαιτούνται για κάθε κατηγορία προβλημάτων, όπως αυτά ταξινομήθηκαν με βάση την πυκνότητά τους. Επίσης, στο γράφημα που εμφανίζεται στην Εικόνα 4-16 εμφανίζεται ο αντίστοιχος χρόνος σε δευτερόλεπτα για τις ίδιες κατηγορίες προβλημάτων.

Παρατηρούμε ότι για τα πυκνότερα προβλήματα, επειδή υπάρχουν λιγότερα μηδενικά, απαιτείται μεγαλύτερος χρόνος για την εκτέλεσή τους. Ο λόγος είναι ότι οι πράξεις ανάμεσα στους πίνακες είναι περισσότερο χρονοβόρες. Το ίδιο όμως δεν φαίνεται να ισχύει για τον μέσο όρο των επαναλήψεων. Στην περίπτωση που έχουμε περισσότερα μηδενικά, μπορούμε να παρατηρήσουμε ότι έχουμε περισσότερες επαναλήψεις, κάτι που μπορεί να οφείλεται στο γεγονός ότι παράγονται εκφυλισμένες λύσεις.



Εικόνα 4-15: Μέσος αριθμός επαναλήψεων για κάθε κατηγορία προβλημάτων με βάση την πυκνότητά τους

Εικόνα 4-16: Μέσος χρόνος εκτέλεσης για κάθε κατηγορία προβλημάτων με βάση την πυκνότητά τους

5 Επίλογος

5.1 Σύνοψη και συμπεράσματα

Στην παρούσα εργασία παρουσιάστηκε η ανάπτυξη κάποιων αλγορίθμων τύπου Simplex σε γλώσσα προγραμματισμού Python, για την επίλυση γενικών γραμμικών προβλημάτων. Για το λόγο αυτό αναπτύχθηκε ένα σύνολο από συναρτήσεις, σε γλώσσα προγραμματισμού Python, οι οποίες επιτελούν διάφορες λειτουργίες, όπως την ανάγνωση και επίλυση γραμμικών προβλημάτων τα οποία είναι αποθηκευμένα σε μορφή mps, καθώς επίσης και τη δημιουργία και αποθήκευση σε μορφή mps τυχαίων γραμμικών προβλημάτων ανάλογα με τις παραμέτρους που δίνονται από το χρήστη. Στη συνέχεια πραγματοποιήθηκε υπολογιστική μελέτη, σκοπός της οποίας ήταν η παρατήρηση της υπολογιστικής συμπεριφοράς των αλγορίθμων αυτών. Τα αποτελέσματα της υπολογιστής μελέτης παρουσιάστηκαν με τη βοήθεια των κατάλληλων γραφημάτων.

Από την εκπόνηση της εργασίας αυτής, μπορούν να προκύψουν διάφορα χρήσιμα συμπεράσματα, όπως αυτά που φαίνονται πιο κάτω:

- Η γλώσσα προγραμματισμού Python αποτελεί ένα πολύ ισχυρό εργαλείο το οποίο μπορεί να χρησιμοποιηθεί για την ανάπτυξη προγραμμάτων και βιβλιοθηκών τα οποία αφορούν στην επίλυση γραμμικών προβλημάτων και στην υλοποίηση αλγορίθμων τύπου Simplex.
- Η Python προσφέρει πολλές έτοιμες βιβλιοθήκες, οι οποίες μπορούν να βοηθήσουν στην ανάπτυξη προγραμμάτων που αφορούν αλγορίθμους τύπου Simplex και επιπλέον, επιτρέπει τη δημιουργία επαναχρησιμοποιήσιμων συναρτήσεων, οι οποίες μπορούν να αποτελέσουν βιβλιοθήκες για την ανάπτυξη άλλων προγραμμάτων που επιλύουν παρόμοια προβλήματα.
- Ο αριθμός των επαναλήψεων που χρειάζονται για την επίλυση ενός γραμμικού προβλήματος, αυξάνεται όσο μεγαλώνει η διάσταση του προβλήματος.
- Όσο αυξάνεται η διάσταση ενός γραμμικού προβλήματος, τόσο αυξάνεται και ο απαιτούμενος χρόνος για την επίλυσή του και μάλιστα, η αύξηση αυτή έχει μεγαλύτερη επίδραση σε σχέση με την αύξηση του αριθμού επαναλήψεων.

Φαίνεται μάλιστα να αυξάνει εκθετικά, σε σχέση με τη διάσταση του προβλήματος.

- Όσο αυξάνεται η πυκνότητα ενός προβλήματος, δηλαδή όσο μειώνεται το ποσοστό των μηδενικών που υπάρχουν στον πίνακα των συντελεστών του προβλήματος, τόσο απαιτείται μεγαλύτερος χρόνος για την επίλυση του προβλήματος αυτού. Ο λόγος είναι ότι οι πράξεις ανάμεσα στους πίνακες είναι περισσότερο χρονοβόρες.
- Αντίθετα, όσο αυξάνεται η πυκνότητα ενός προβλήματος, τόσο φαίνεται να μειώνεται ο μέσος όρος των επαναλήψεων που απαιτούνται για την επίλυση του προβλήματος, εξαιτίας του φαινομένου των εκφυλισμένων λύσεων που φαίνεται να προκύπτουν.

5.2 Μελλοντικές Επεκτάσεις

Το πεδίο έρευνας στον χώρο του γραμμικού προγραμματισμού και την λειτουργία των αλγορίθμων τύπου Simplex, είναι ιδιαίτερα μεγάλο και υπάρχουν διάφορα σημεία στα οποία θα μπορούσε η μελέτη που πραγματοποιήθηκε, στα πλαίσια της εργασίας αυτής, να επεκταθεί μελλοντικά. Μερικές προτάσεις για μελλοντική επέκταση του συστήματος είναι οι πιο κάτω:

- Υλοποίηση επιπλέον αλγορίθμων τύπου Simplex με τη βοήθεια της γλώσσας προγραμματισμού Python και προσθήκη στην βιβλιοθήκη που αναπτύχθηκε, όπως για παράδειγμα υλοποίηση αλγορίθμων εξωτερικών σημείων.
- Σύγκριση των διαφορετικών αλγορίθμων μεταξύ τους, ως προς τον αριθμό των επαναλήψεων και τον απαιτούμενο χρόνο που απαιτούν, για την επίλυση προβλημάτων διαφόρων διαστάσεων.
- Ανάπτυξη γραφικού περιβάλλοντος για την δημιουργία και την επίλυση γραμμικών προβλημάτων με φιλικό τρόπο προς τον τελικό χρήστη, για παράδειγμα με τη βοήθεια της τεχνολογίας Django, που επιτρέπει την λειτουργία προγραμμάτων μέσω του Παγκόσμιου Ιστού.

- Εκτενέστερη υπολογιστική μελέτη για περισσότερες κατηγορίες προβλημάτων και εμφάνιση των αποτελεσμάτων.

6 Βιβλιογραφία

Anaconda, 2018. *Anaconda Website* [Online]. Διαθέσιμο στο: https://www.anaconda.com/. [Πρόσβαση 21 Οκτωβρίου 2018].

Atom, 2018. *Atom IDE Website* [Online]. Διαθέσιμο στο: https://ide.atom.io/. [Πρόσβαση 23 Οκτωβρίου 2018].

Beale E. M. L., 1955. *Cycling in the Dual Simplex Algorithm*, Naval Research Logistics Quarterly 2(4):269 – 275.

Benichou, M., Gautier, J.M., Hentges, G., Ribiere, G., 1977. *The efficient solution of*

large-scale linear programming problems. Mathematical Programming, 13, pp. 280–322.

Bland G. R., 1977. *New finite pivoting rules for the Simplex method*, Mathematics of Operational Research, Vol. 2, pp. 103-107.

Chvatal V., 1983. *Linear Programming*, W.H.Freeman & Co Ltd, ISBN 978-0716711957.

Dantzig, B.G., 1948. *Programming in a linear structure*, Comptroller, US Air Force, Washington, D.C.

Dantzig, B.G., 1963. *Linear programming and extensions*, PrincetonUniversity Press, Princeton.

Eclipse, 2018. *Eclipse, The Platform for Open Innovation and Collabo*ration [Online]. Διαθέσιμο στο: https://www.eclipse.org/. [Πρόσβαση 23 Οκτωβρίου 2018].

Gartner B., 1995. *Randomized optimization of Simplex-type method*, PhD Thesis, Freien University, Berlin, Department of Informatics and Mathematics.

Gay, D.M., 1985. *Electronic mail distribution of linear programming test problems*, Mathematical Programming Society COAL Newsletter, Vol. 13, pp. 10-12.

Guttag, J. V., 2016. *Introduction to Computation and Programming Using Python: With Application to Understanding Data*. MIT Press. ISBN 978-0-262-52962-4.

Karmarkar, K.N., 1984. *A new polynomial time algorithm for linear programming*, Combinatorica, vol. 4, pp. 373-395.

Khachiyan,L., 1979. *A polynomial algorithm in linear programming*, Soviet Math. Doklady, 20 (1), pp. 191-194.

Klee V., Minty G., 1972. *How good is the Simplex algorithm?*, In Inequalities III, Shisha, O. (Ed.), Academic Press, New York, pp. 159-179.

Lustig, J.I., Marsten, J.M., Shanno, D.F., 1994. *Interior point methods for linear programming computational state of the art*, ORSA Journal on Computing, Vol. 6, pp. 1-14.

Marshall K. T., Suurballe J. W., 1969. *A note on cycling in the simplex method*, Naval Research Logistics Quarterly 16(1).

Millman, K. and Aivazis, M., 2011. *Python for Scientists and Engineers. Computing in Science & Engineering*. 13. 9 - 12. 10.1109/MCSE.2011.36.

Murty, K.G., 1983. *Linear Programming*. Wiley, New York.

Netlib, 2018. *Netlib Website* [Online]. Διαθέσιμο στο: http://www.netlib.org/lp/data/. [Πρόσβαση 24 Οκτωβρίου 2018].

NumPy, 2018. *NumPy Website* [Online]. Διαθέσιμο στο: http://www.numpy.org/. [Πρόσβαση 20 Οκτωβρίου 2018].

Oliphant, T., 2007. *Python for Scientific Computing. Computing in Science & Engineering*. 9. 10-20. 10.1109/MCSE.2007.58.

Paparrizos, K., 1988. *A non-dual signature method for assignment problems and a generalization of the dual simplex method for transportation problems*, RAIRO-Operations Research, Vol. 22(3), pp. 269-289.

Paparrizos, K., 1990. *A generalization of an exterior point simplex algorithm for linearprogramming problems,* Technical Report, University of Macedonia.

Paparrizos, K., 1991. *An infeasible (Exterior Point) simplex algorithm for assignment problems,* Mathematical Programming, Vol. 51, pp. 45-54.

Paparrizos, K.,1993. *An exterior point simplex algorithm for (general) linear programming problems*, Annals of Operations Research, Vol. 47, pp. 497-508.

Paparrizos, K., 1996. *A new primal and dual pivoting rule for the simplex algorithm*, Proceedings of SYMOPIS ’96, pp. 448-453.

Paparrizos, K., 1996. *Exterior point simplex algorithms, simple and short proof of correctness*. Proceedings of SYMOPIS ’96, pp. 13-18.

Paparrizos K., 1997. *Pivoting algorithms generating two paths*. Presented in ISMP ’97, Lausanne, EPFL Switzerland, Abstracts: 207.

Paparrizos, K. Stephanides, G., Samaras, N., 2001. *Improved criteria for identifying optimal basic and nonbasic variables in LP*, Journal of Computational Analysis and Applications, Vol. 3(1), pp. 75-82.

Paparrizos, K., Samaras, N., Stephanides, G., 2001. *A new efficient primal dual simplex algorithm*, accepted for publication in Computers and operations Research.

Paparrizos, K., Samaras, N., Stephanides, G., 2000. *An efficient simplex type algorithm for sparse and dense linear programs*, accepted for publication in European Journal of Operational Research.

Paparrizos, K., Samaras, N., Tsiplidis, K., 2001. *Pivoting algorithms for (LP) generating two paths, Encyclopedia of Optimization*, Pardalos, P., Floudas, C. (Eds.), Kluwer Academic Publishers, Vol. 4, pp. 302-306.

Paparrizos, K., Samaras, Stephanides G., 2003. *An efficient simplex type algorithm for sparse and dense linear programs*, European Journal of Operational Research 148, 323-334.

Paparrizos, K., 1999. *Linear programming, algorithms and applications*, ZYGOS Publications, Thessaloniki, ISBN: 960-8065-13-5 (In Greek).

Paparrizos, K., 2009. *Linear programming, an approach in Matlab*, ZYGOS Publications, Thessaloniki, ISBN: 978-960-8065-67-3 (In Greek)

Park, S., 1999. *Linear Programming*. Minyong-sa, Seoul.

Py2exe, 2018. *Py2exe Website* [Online]. Διαθέσιμο στο: http://www.py2exe.org/. [Πρόσβαση 15 Οκτωβρίου 2018].

PyCharm, 2018. *PyCharm, The Python IDE for Professional Developers* [Online]. Διαθέσιμο στο: https://www.jetbrains.com/pycharm/. [Πρόσβαση 22 Οκτωβρίου 2018].

PyDev, 2018. *PyDev Website* [Online]. Διαθέσιμο στο: http://www.pydev.org/. [Πρόσβαση 23 Οκτωβρίου 2018].

PyInstaller, 2018. *PyInstaller Website* [Online]. Διαθέσιμο στο: https://www.pyinstaller.org/. [Πρόσβαση 15 Οκτωβρίου 2018].

Python, 2018. *Python Programming Language* [Online]. Διαθέσιμο στο: https://en.wikipedia.org/wiki/Python_(programming_language). [Πρόσβαση 14 Οκτωβρίου 2018].

Python, 2018. *Python Software Foundation*. Διαθέσιμο στο: https://www.python.org/. [Πρόσβαση 20 Σεπτεμβρίου 2018].

Roos, C., Terlaky, T., Vial, J-Ph., 1997. *Theory and algorithms for linear optimization, an interior point approach*, J. Wiley & Sons, Inc., New York, NY.

Samaras, N., 2001. *Computational Improvement and Efficient Implementation of Two Paths Pivoting Algorithms*, Ph.D. Thesis, University of Macedonia, Department of Applied Informatics, (in Greek).

Spyder, 2018. *Spyder, The Scientific Python Development Environment* [Online]. Διαθέσιμο στο: https://www.spyder-ide.org/. [Πρόσβαση 22 Οκτωβρίου 2018].

Swietanowski, A., 1998. *A new Steepest Edge approximation for simplex method for Linear Programming*, Computation Optimization and Application, 10(3), pp. 271-281.

Terlaky, T., Zhang, S., 1993. *Pivot rules for linear programming: a survey on recent theoretical developments*. Annals of Operations Research, 46, 203–233.

TIOBE, 2018. *TIOBE Index for August 2018 [Online]*. Διαθέσιμο στο: https://tiobe.com/tiobe-index/. [Πρόσβαση 3 Αυγούστου 2018].

Thomadakis, M. E., Jyh-Charn Liu, J., 1996. *An efficient Steepest Edge algorithm for SIMD Computers*, in Proc. of International Conference on Supercomputing, pp. 286-293.

Tomlin, J.A., 1975. *On scaling linear programming problems*. In: Balinski M.L., Hellerman E. (eds) Computational Practice in Mathematical Programming. Mathematical Programming Studies, vol 4. Springer, Berlin, Heidelberg.

Vanderbei, R. J., 2001. *Linear Programming: Foundations and Extensions*, 2nd edition, Kluwer Academic Publishers.

Wagner, H. M., 1957. *A comparison of the original and the revised simplex methods*, Operational Research, Vol. 5(3).

Wright, J.S., 1997. *Primal-Dual Interior Point Methods*, SIAM Press.

Zadeh, N., 1980. *What is the worst case behavior of the Simplex algorithm?*, Technical report, Stanford University, Department of Operations Research.