ΠΑΝΕΠΙΣΤΗΜΙΟ ΜΑΚΕΔΟΝΙΑΣ
ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
ΤΜΗΜΑΤΟΣ ΕΦΑΡΜΟΣΜΕΝΗΣ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ SIMPLEX ΓΙΑ ΤΥΧΑΙΑ ΓΡΑΜΜΙΚΑ ΠΡΟΒΛΗΜΑΤΑ ΣΕ ΡΥΤΗΟΝ

Διπλωματική Εργασία

της

Λάμπρογλου Αικατερίνης

Θεσσαλονίκη, 6/2019

ΥΛΟΠΟΙΗΣΗ ΑΛΓΟΡΙΘΜΟΥ SIMPLEX ΓΙΑ ΤΥΧΑΙΑ ΓΡΑΜΜΙΚΑ ΠΡΟΒΛΗΜΑΤΑ ΣΕ ΡΥΤΗΟΝ

Λάμπρογλου Αικατερίνη

Πτυχίο Φυσικής Α.Π.Θ. 2013

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 την ηη/μμ/εεεε

Σαμαράς Νικόλαος	Ρεφανίδης Ιωάννης	Σιφαλέρας Άγγελος
.....	.....	.....

Λάμπρογλου Αικατερίνη

.....

Περίληψη

Ο αλγόριθμος simplex είναι ο πρώτος και πιο χρησιμοποιημένος αλγόριθμος για την επίλυση γραμμικών προβλημάτων. Τα πραγματικά προβλήματα είναι κατά βάση αραιά. Στην παρούσα εργασία υλοποιείται ο αναθεωρημένος αλγόριθμος simplex σε γλώσσα προγραμματισμού python και γίνεται μια προσπάθεια μελέτης της επίδοσης του αλγόριθμου σε τυχαία πυκνά γραμμικά προβλήματα.

Λέξεις Κλειδιά: Αναθεωρημένος αλγόριθμος simplex, Python, Τυχαία Πυκνά Γραμμικά Προβλήματα

Abstract

The simplex algorithm is the first and most used algorithm for solving linear problems. The present thesis implement the revised simplex algorithm with programming language python and tries to study its efficiency in random dense linear problems.

Keywords: Revised Simplex, Python, Randomly Generated Dense Linear Problems

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Πινάκων

Λίστα Κώδικα

Κεφάλαιο 1

Εισαγωγή

1.1 Γραμμικός προγραμματισμός

Ο Γραμμικός Προγραμματισμός είναι μία τεχνική για τον προσδιορισμό της καλύτερης ή άριστης λύσης (όπως μεγιστοποίηση του κέρδους ή ελαχιστοποίηση του κόστους) σε ένα πρόβλημα με τη χρήση ενός μαθηματικού μοντέλου. Το αποτέλεσμα αυτό πρέπει να είναι συμβατό με ένα πεπερασμένο σύνολο γραμμικών ανισοτήτων.

Από οικονομικής σκοπιάς, ο γραμμικός προγραμματισμός είναι μια μέθοδος που ασχολείται με το πρόβλημα της κατανομής των περιορισμένων πόρων ενός συστήματος κατά τον καλύτερο δυνατό τρόπο. Προβλήματα αυτής της μορφής είναι η κατανομή του εργατικού δυναμικού, του τεχνολογικού εξοπλισμού και των πρώτων υλών σε διάφορες παραγωγικές διαδικασίες, η κατανομή κεφαλαίου σε διάφορα επενδυτικά προγράμματα, ο καθορισμός του συστήματος διανομής σε αποστολές προϊόντων και ο προγραμματισμός σχολικών λεωφορείων. Θεωρείται σαν μια από τις σπουδαιότερες μαθηματικές ανακαλύψεις του εικοστού αιώνα και αποτελεί ένα μοντέλο ευρείας χρήσης σε ποικίλους τομείς όπως των μαθηματικών, των οικονομικών, της γεωργίας, των επιχειρήσεων και των βιομηχανιών.

Ιστορικά ο γραμμικός προγραμματισμός ξεκίνησε να αναπτύσσεται κατά τη διάρκεια του δευτέρου παγκοσμίου πολέμου όταν ο σοβιετικός οικονομολόγος Leonid Kantorovich διαμόρφωσε ένα πρόβλημα γραμμικού προγραμματισμού και πρότεινε τρόπο επίλυσής του. Ήταν μία μέθοδος για τη βέλτιστη κατανομή των συμμαχικών στρατευμάτων και των εφοδίων στην Ευρώπη, ώστε να ελαχιστοποιηθεί ο χρόνος πρόσβασης στα πεδία των μαχών, να βελτιωθεί ο εφοδιασμός και να γίνει αποδοτικότερος ο συντονισμός και η διοίκηση των συμμαχικών δυνάμεων. Την ίδια περίοδο ο δανο-αμερικανός οικονομολόγος T.C. Koopmans μοντελοποίησε ένα κλασικό οικονομικό πρόβλημα ως γραμμικό πρόβλημα. Το 1941 ο Frank Lauren Hitchcock μοντελοποίησε επίσης ένα πρόβλημα μεταφοράς σε γραμμικό.

Στα έτη 1946-1947, ανεξάρτητα από τους υπόλοιπους, ο George B. Dantzig κατασκεύασε το γενικό πλαίσιο γραμμικού προγραμματισμού και ανακάλυψε μια μέθοδο επίλυσής του. Στόχος του ήταν να λύσει το οργανωτικό πρόβλημα της αμερικανικής πολεμικής αεροπορίας. Πρότεινε ότι οι αλληλοσυσχετίσεις μεταξύ των δραστηριοτήτων ενός μεγάλου οργανισμού μπορεί να θεωρηθούν ως ένα μοντέλο γραμμικού προβλήματος

στο οποίο η βέλτιστη λύση μπορεί να βρεθεί με την ελαχιστοποίηση μιας μοναδικής γραμμικής συνάρτησης. Η πρώτη παρουσίαση της μεθόδου του, του αλγόριθμου simplex,δεν έπεισε για την αποτελεσματικότητά της. Δεν έγινε όμως το ίδιο κατά την δεύτερη παρουσίαση. Η επιστημονική κοινότητα αποδέχτηκε την ανακάλυψή του και ξεκίνησε η ανάπτυξη μιας νέας επιστήμης. Η μέθοδός του παραμένει μέχρι και σήμερα, μια από τις πιο αποτελεσματικές για μια μεγάλη πλειοψηφία πρακτικών προβλημάτων. Αν και διάφορες παραλλαγές της μεθόδου έχουν αναπτυχθεί, καμία από αυτές δεν έχει πολυωνυμική πολυπλοκότητα. Παρόλο που ο αλγόριθμος μπορεί να απαιτήσει υπολογιστική προσπάθεια που αυξάνεται εκθετικά με το χρόνο, είναι αποτελεσματικός στην πράξη.

Ο πρώτος αλγόριθμος πολυωνυμικού χρόνου, ο ελλειψοειδής αλγόριθμος, εφευρέθηκε το 1979 από τον Khachian. Η απόδοση του όμως είναι κακή για τα πρακτικά προβλήματα συγκρινόμενα με τη μέθοδο simplex. Το 1984 ο Karmakar παρουσίασε ένα νέο αλγόριθμο εσωτερικών σημείων, που αποδείχθηκε να είναι ο πρώτος αποδοτικός αλγόριθμος πολυωνυμικού χρόνου για επίλυση γραμμικών προβλημάτων. Για τα πρακτικά προβλήματα στην πραγματική ζωή, η μέθοδος εσωτερικών σημείων, ξεπερνά τη μέθοδο simplex για τα πολύ μεγάλης κλίμακας προβλήματα. Για τα μεσαία ή μικρού μεγέθους προβλήματα όμως η μέθοδος simplex λειτουργεί συχνά καλύτερα, μιας και απαιτεί πολύ λιγότερη υπολογιστική προσπάθεια. Επιπλέον, ο αλγόριθμος simplex έχει τη δυνατότητα να υπολογίσει ένα αρχικό σημείο ξεκινήματος πολύ πιο εύκολα από ότι οι μέθοδοι εσωτερικού σημείου.

Το 1991 ο Παπαρρίζος ανέπτυξε τον πρώτο αλγόριθμο εξωτερικού σημείου τύπου simplex για το πρόβλημα της αντιστοίχισης. Το 1993, γενίκευσε τη μέθοδό του για το γενικό γραμμικό πρόβλημα με την ανάπτυξη ενός δυϊκού αλγόριθμου. Η μέθοδός του, πραγματοποιεί λιγότερες επαναλήψεις και οδηγεί σε μικρότερο χρόνο από ότι ο simplex σε εύρεση μιας εφικτής λύσης.

Ο αναθεωρημένος αλγόριθμος simplex είναι μια παραλλαγή του simplex που εφαρμόζει μια διαφορετική μέθοδο ανανέωσης των δεδομένων σε κάθε επανάληψη. Χρησιμοποιεί κάθε φορά τα αρχικά δεδομένα και όχι αυτά της προηγούμενης επανάληψης. Είναι πιο αποδοτικός υπολογιστικά από τον αλγόριθμο simplex, ειδικά για μεγάλα και αραιά προβλήματα.

1.2 Η γλώσσα Python

Η γλώσσα προγραμματισμού Python δημιουργήθηκε από τον Ολλανδό Guido van Rossum το 1990 και κυκλοφόρησε για πρώτη φορά το 1991. Θεωρείται διάδοχος της γλώσσας προγραμματισμού ABC, η οποία υπήρξε η βασική πηγή έμπνευσης για την δημιουργία της. Αρχικά χρησιμοποιήθηκε ως γλώσσα σεναρίων για το κατανεμημένο λειτουργικό σύστημα Amoeba. Το όνομά της οφείλεται στη δημοφιλή κωμική σειρά Monty Python's Flying Circus του BBC της Μεγάλης Βρετανίας και δεν σχετίζεται με το γνωστό φίδι πύθωνας.

Είναι μια διερμηνευόμενη, ισχυρή, δυναμική, αποδοτική, επεκτάσιμη, γενικού σκοπού και υψηλού επιπέδου γλώσσα προγραμματισμού. Η φιλοσοφία της ενθαρρύνει την απλότητα και αναγνωσιμότητα του κώδικα και διαθέτει πληθώρα έτοιμων βιβλιοθηκών που μπορούν να χρησιμοποιηθούν εύκολα και άμεσα. Η ανάπτυξη των εφαρμογών και του παραγόμενου κώδικα μέσω της Python ενθαρρύνεται να είναι όσο πιο απλή γίνεται. Τα προγράμματα της είναι συμπαγή, ευανάγνωστα και γράφονται και συντηρούνται γρηγορότερα σε σχέση με άλλες δημοφιλείς γλώσσες προγραμματισμού όπως οι C, C++ και Java.

Για την μετατροπή του προγράμματος από γλώσσα προγραμματισμού σε γλώσσα μηχανής υπάρχουν δύο είδη προγραμμάτων. Οι διερμηνευτές (interpreters),οι οποίοι μετατρέπουν γραμμή προς γραμμή τον πηγαίο κώδικα του προγράμματος σε γλώσσα μηχανής και τον εκτελούν άμεσα και οι μεταγλωττιστές (compilers),οι οποίοι μετατρέπουν όλο το πρόγραμμα σε γλώσσα μηχανής, το αποθηκεύουν στο δίσκο και εν συνεχεία το εκτελούν. Η Python χρησιμοποιεί εικονική μηχανή (virtual machine) για την εκτέλεση του κώδικα, στην οποία αυτός μετατρέπεται σε μια ενδιάμεση γλώσσα (bytecode) πριν μετατραπεί σε γλώσσα μηχανής. Η τελική μορφή του κώδικα που έχει μεταφραστεί και μπορεί πλέον να εκτελεστεί ονομάζεται αντικειμενικός κώδικας (object code).

Η Python διαθέτει αποδοτικές δομές δεδομένων υψηλού επιπέδου και υποστηρίζει μια απλή αλλά αποτελεσματική προσέγγιση στον αντικειμενοστρεφή προγραμματισμό. Υποστηρίζει όμως και άλλες προγραμματιστικές προσεγγίσεις όπως είναι ο διαδικαστικός και ο συναρτησιακός προγραμματισμός. Μπορεί να χρησιμοποιηθεί για τη δημιουργία σεναρίων εντολών ή ακόμα για τη γρήγορη ανάπτυξη ολοκληρωμένων εφαρμογών σε διάφορες περιοχές ενδιαφέροντος.

Οι βιβλιοθήκες της γλώσσας καλύπτουν ένα πολύ μεγάλο εύρος αναγκών και μπορούν να επεκταθούν με νέα τμήματα γραμμένα σε C/C++. Η ίδια η γλώσσα είναι

επεκτάσιμη καθώς μόνο ένα βασικό σύνολό της αποτελεί τον πυρήνα της ενώ τα υπόλοιπα είναι αρθρώματα (modules) που επεκτείνουν την λειτουργικότητά της. Το γεγονός επίσης ότι είναι γλώσσα ανοικτού λογισμικού της παρέχει το πλεονέκτημα να μην μένει ποτέ στάσιμη και να παρακολουθεί της εξελίξεις.

Είναι μια γλώσσα που έχει την δυνατότητα να παίζει σε όλες τις κύριες πλατφόρμες των λειτουργικών συστημάτων (Windows, Linux/Unix, OS/2, Mac, Amiga). Μπορεί επίσης να συνδυαστεί με άλλες γλώσσες προγραμματισμού όπως η Java, όπου μέσω της Jython μπορούμε να χρησιμοποιήσουμε βιβλιοθήκες της Java. Ένας άλλος συνδυασμός είναι με τις γλώσσες C/C++ όπου μπορεί να γραφεί κώδικας σε C/C++ και στη συνέχεια να κατασκευαστούν αρθρώματα (modules) τα οποία θα ενσωματωθούν σε κώδικα Python.

Κατά την δημιουργία αντικειμένων, η γλώσσα Python έχει το πλεονέκτημα να διαχειρίζεται αυτόματα τη μνήμη. Αντιλαμβάνεται επίσης πότε το ίδιο αντικείμενο αναφέρεται πάνω από μια φορές και με αυτό τον τρόπο δεν το αποθηκεύει στη μνήμη αν δεν χρειάζεται. Η τεχνική αυτή που χρησιμοποιεί η γλώσσα ονομάζεται μέτρηση αναφορών (reference counting).

Στα βασικά χαρακτηριστικά της γλώσσας περιλαμβάνεται το γεγονός ότι σχεδόν τα πάντα στην Python είναι αντικείμενα, καθένα με τις ιδιότητες και τις μεθόδους του. Οι συναρτήσεις, οι γεννήτορες, οι εξαιρέσεις, οι μεταβλητές και πολλά άλλα είναι αντικείμενα στην Python. Ένα άλλο βασικό χαρακτηριστικό της γλώσσας είναι ότι κάθε μπλοκ κώδικα καθορίζεται από τη στοίχισή του. Κατά αυτό τον τρόπο, είναι υποχρεωτικό να ενσωματώνεται στον τρόπο γραφής κώδικα, μια συνέπεια ως προς τον καθορισμό της στοίχισής του.

Ανάμεσα στους σημαντικότερους χρήστες της συγκαταλέγονται οι Google, Nasa, Yahoo, Μεγάλα Πανεπιστήμια (όπως MIT, Stanford κτλ) και σχεδόν όλες οι διανομές Linux. Ο κύριος λόγος για τον οποίο χρησιμοποιούν την Python είναι γιατί παρέχει την δυνατότητα να επικεντρωθεί κάποιος σε αυτό που θέλει να γράψει και όχι στις ιδιαιτερότητες της γλώσσας.

Πέρα όμως από την πληθώρα όλων των παραπάνω θετικών χαρακτηριστικών που έχει η γλώσσα Python, έχει και μειονεκτήματα. Ο χρόνος εκτέλεσης των προγραμμάτων της μπορεί να μην είναι τόσο γρήγορος όσο σε άλλες γλώσσες όπως η C και η C++. Αυτό βέβαια πολλές φορές αντισταθμίζεται από την εξοικονόμηση χρόνου που προκύπτει όταν μια εφαρμογή αναπτύσσεται σε Python. Ένα επιπλέον μειονέκτημα της γλώσσας είναι ότι δεν είναι τόσο αποδοτική για την δημιουργία ενός νέου Λειτουργικού Συστήματος.

1.3 Διάρθρωση της εργασίας

Στο δεύτερο κεφάλαιο παρουσιάζεται αναλυτικά ο τρόπος λειτουργίας του αλγόριθμου simplex, του αναθεωρημένου αλγόριθμου simplex και οι μέθοδοι που χρησιμοποιούνται για την δημιουργία αρχικής εφικτής βάσης, την επιλογή της εισερχόμενης και εξερχόμενης μεταβλητής και την αντιστροφή της βάσης. Στο τρίτο κεφάλαιο παρουσιάζεται ο αλγόριθμος όπως υλοποιήθηκε στη γλώσσα προγραμματισμού python και η εφαρμογή του σε τυχαία πυκνά προβλήματα. Στο τέταρτο κεφάλαιο παρουσιάζεται ο λύτης γραμμικών προβλημάτων της γλώσσας προγραμματισμού Python, η συνάρτηση linprog η οποία εφαρμόζεται σε τυχαία βέλτιστα προβλήματα.. Στο πέμπτο και τελευταίο κεφάλαιο παρουσιάζονται τα υπολογιστικά αποτελέσματα.

Κεφάλαιο 2

Αλγόριθμος Simplex

2.1 Το γραμμικό πρόβλημα

Η γενική μορφή ενός γραμμικού προβλήματος με μαθηματικούς όρους και συμβολισμούς μπορεί να διατυπωθεί ως εξής

$$\min\backslash\max\ z = c_1x_1 + c_2x_2 + \ldots\ldots\ldots + c_nx_n$$

$$\mu.\pi. \quad \alpha_{11}x_1 + \alpha_{12}x_2 + \ldots\ldots\ldots + \alpha_1x_n \quad \otimes \quad b_1$$

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

$$\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots\ldots \quad \ldots\ldots\ldots$$

$$\alpha_{m1} + \alpha_{m2}x_2 + \ldots\ldots\ldots + \alpha_{mn}x_n \quad \otimes \quad b_m$$

$$x_j \geq 0 \ \ (j=1,2,\ldots,n)$$

Οι συντελεστές c_j , α_{ij}, b_i (i= 1,2,…m) είναι γνωστοί πραγματικοί αριθμοί που αποτελούν τα δεδομένα του προβλήματος. Ο συμβολισμός $\otimes$ αντιστοιχεί σε ένα από τα μαθηματικά σύμβολα $\leq$, $\geq$ και =.Οι μεταβλητές x_1 ,x_2,….x_n είναι οι άγνωστες μεταβλητές του προβλήματος και ονομάζονται μεταβλητές απόφασης (decision variables).Οι μεταβλητές αυτές υπόκεινται στους λεγόμενους φυσικούς περιορισμούς (natural constraints) μη αρνητικότητας (non negativity).Η συνάρτηση z ονομάζεται αντικειμενική συνάρτηση (objective function) και οι τιμές της αντικειμενικές τιμές (objective values). Όταν σε ένα γραμμικό πρόβλημα ζητείται να βρεθεί το ελάχιστο το πρόβλημα ονομάζεται ελαχιστοποίησης (minimization problem), ενώ όταν ζητείται το μέγιστο το πρόβλημα ονομάζεται μεγιστοποίησης (maximization problem).

Από μαθηματική άποψη δεν υπάρχει ουσιαστική διαφορά μεταξύ των δύο προβλημάτων διότι ισχύει:

$$\max \ z = \min \ -z$$

Άρα ένα πρόβλημα μεγιστοποίησης (ελαχιστοποίησης) μπορεί να μετατραπεί σε ένα πρόβλημα ελαχιστοποίησης (μεγιστοποίησης) πολλαπλασιάζοντας τους συντελεστές της αντικειμενικής συνάρτησης με -1. Μετά τη βελτιστοποίηση του νέου προβλήματος, η βέλτιστη τιμή της αντικειμενικής συνάρτησης του παλιού είναι -1 φορές η βέλτιστη τιμή του νέου προβλήματος.

Το ίδιο πρόβλημα εκφρασμένο με μήτρες και διανύσματα δίνεται από τη μορφή:

$$\min \backslash \max \qquad z = c^Tx$$

$$\text{μ.π.} \qquad Ax \otimes b$$
$$x \geq 0$$

όπου $A \in \Re^{mxn}$, c, $x \in \Re^{n}$, $b \in \Re^{m}$, T σημαίνει αναστροφή και $\otimes$ ένα διάνυσμα διαστάσεων mx1 ή την πιο σύντομη μορφή :

$$\min\backslash\max\{ c^{T}x : Ax \otimes b, x \geq 0\}$$

Τα σημεία που ικανοποιούν όλους τους περιορισμούς ονομάζονται εφικτά σημεία (feasible points) ή εφικτές λύσεις (feasible solutions). Το σύνολο των εφικτών σημείων ορίζει την εφικτή περιοχή (feasible region) του προβλήματος. Αν η εφικτή περιοχή είναι το κενό σύνολο, το πρόβλημα είναι αδύνατο ή μη εφικτό (infeasible).

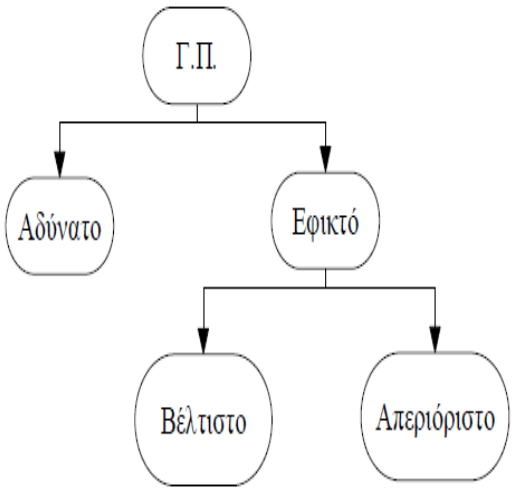
Ένα εφικτό σημείο x ενός προβλήματος είναι βέλτιστο (optimal) αν για κάθε εφικτό σημείο y ισχύει:

$$c^{T}x \leq c^{T}y \text{ (min)} \quad \text{ή} \quad c^{T}x \geq c^{T}y \text{ (max)}$$

Ένα πρόβλημα είναι βέλτιστο όταν έχει τουλάχιστον ένα βέλτιστο σημείο. Η τιμή της αντικειμενικής συνάρτησης στο βέλτιστο σημείο ονομάζεται βέλτιστη τιμή (optimum value).

Ένα εφικτό γραμμικό πρόβλημα που δεν είναι βέλτιστο, είναι απεριόριστο (unbounded). Στα απεριόριστα προβλήματα υπάρχει ακολουθία εφικτών σημείων $\{x_1, x_2, \ldots\}$ τέτοια ώστε η ακολουθία των αντίστοιχων αντικειμενικών τιμών να τείνει στο άπειρο (- ∞ για min, +∞ για max)

Συνοπτικά ένα γραμμικό πρόβλημα είναι αδύνατο ή εφικτό. Αν είναι εφικτό, είναι είτε βέλτιστο είτε απεριόριστο (Σχήμα 2.1.1)



Σχήμα2.1.1 Λύση του Γραμμικού προβλήματος

Ένα γραμμικό πρόβλημα βρίσκεται σε κανονική μορφή (canonical form) όταν περιλαμβάνει μόνο ανισοτικούς περιορισμούς, ενώ όταν περιλαμβάνει μόνο ισοτικούς περιορισμούς βρίσκεται σε τυπική μορφή (standard form).

Όλα τα γραμμικά προβλήματα μπορούν να μετασχηματιστούν από την κανονική σε τυπική μορφή με εισαγωγή χαλαρών μεταβλητών (slack variables).

Πιο αναλυτικά, ένας ανισοτικός περιορισμός της μορφής

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots\ldots\ldots + \alpha_n x_n \leq b$$

μπορεί να μετατραπεί σε ισότητα με την προσθήκη μιας νέας μεταβλητής x_{n+1} στο αριστερό της μέλος. Η προηγούμενη ανισότητα γίνεται ισοδύναμη με το σύστημα

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots\ldots\ldots + \alpha_n x_n + x_{n+1} = b$$
$$x_{n+1} \geq 0$$

Η νέα μη αρνητική μεταβλητή ονομάζεται ελλειματική (deficit)

Ομοίως ένας ανισοτικός περιορισμός της μορφής

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots\ldots\ldots + \alpha_n x_n \geq b$$

μπορεί να μετατραπεί σε ισότητα με την αφαίρεση μιας νέας μεταβλητής x_{n+1} στο αριστερό της μέλος. Η προηγούμενη ανισότητα γίνεται ισοδύναμη με το σύστημα

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots\ldots\ldots + \alpha_n x_n - x_{n+1} = b$$
$$x_{n+1} \geq 0$$

Η νέα μη αρνητική μεταβλητή ονομάζεται πλεονασματική (surplus).

2.2 Χαρακτηριστικά αλγόριθμων τύπου simplex

Ένα γραμμικό πρόβλημα της μορφής:

$$\min\backslash\max\{ c^T x : Ax \otimes b, x \geq 0\}$$

βρίσκεται στην κανονική μορφή γιατί περιλαμβάνει ισοτικούς και ανισοτικούς περιορισμούς. Για να μπορεί να επιλυθεί με έναν αλγόριθμο τύπου simplex, το πρόβλημα πρέπει να μετατραπεί στην τυποποιημένη μορφή. Η τυποποιημένη μορφή του προβλήματος είναι:

$$\min\backslash\max\{ c^T x : Ax + s = b, s \geq 0\}$$

όπου $A \in \Re^{mxn}$, $c, x \in \Re^n$, $b \in \Re^m$ και $s \in \Re^m$.

Ας θεωρήσουμε στη συνέχεια το ακόλουθο γραμμικό πρόβλημα:

$$\min \{ c^T x : Ax = b, x \geq 0\}$$

Το δυϊκό πρόβλημά του είναι:

$$\max \{ w^T x : Aw + s = b, s \geq 0\}$$

Τα δύο παραπάνω προβλήματα παρέχουν χρήσιμες πληροφορίες το ένα για το άλλο. Εάν στο πρωτεύον πρόβλημα η τιμή της αντικειμενικής ελαχιστοποιείται, στο δυϊκό μεγιστοποιείται. Όλες οι πρωτεύουσες και δυϊκές μεταβλητές είναι μη αρνητικές. Κάθε περιορισμός σε ένα πρόβλημα αντιστοιχεί σε μια μεταβλητή του άλλου (και αντίστροφα).Τα στοιχεία της δεξιάς πλευράς των περιορισμών σε ένα πρόβλημα είναι οι αντίστοιχοι συντελεστές της αντικειμενικής συνάρτησης στο άλλο πρόβλημα. Ο πίνακας των σταθερών συντελεστών για ένα πρόβλημα είναι ο αντίστροφος του πίνακα σταθερών συντελεστών για το άλλο πρόβλημα.

Επιστρέφοντας στο αρχικό πρόβλημα μπορούμε να διαχωρίσουμε τη μήτρα συντελεστών A (coefficient matrix) ως

$$A = [B\ N]$$

Με βάση αυτό το διαχωρισμό το γραμμικό πρόβλημα μπορεί να γραφεί ως

$$Bx_B + Nx_N = b$$

το οποίο απλοποιείται περαιτέρω στο σύστημα

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

και λύνοντας ως προς x_B με άγνωστο το x_N παίρνουμε

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

Για $x_N = 0$ η εξίσωση καταλήγει

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

ή

$$x = \begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1} b \\ 0 \end{bmatrix}$$

Η λύση αυτή ονομάζεται βασική λύση (basic solution), το διάνυσμα x_B καλείται διάνυσμα των βασικών μεταβλητών και το x_N καλείται διάνυσμα των μη βασικών μεταβλητών. Η μήτρα B, ονομάζεται βασική μήτρα (basic matrix). Το σύνολο δεικτών (set of indices) B είναι βάση (basis) αν η αντίστοιχη μήτρα B διαστάσεων mxm είναι βαθμού m, αν δηλαδή είναι αντιστρέψιμη (invertible). Οι στήλες της μήτρας A, οι οποίες συμβολίζονται με A_j, j=1, …,n, που ανήκουν στο σύνολο δεικτών B ονομάζονται βασικές μεταβλητές (basic variables). Οι υπόλοιπες στήλες της A ονομάζονται μη βασικές μεταβλητές (non basic variables) και ανήκουν στο σύνολο δεικτών N. Η διαμέριση της μήτρας A ονομάζεται βασική διαμέριση (basic portion). Μια βασική λύση είναι εφικτή αν ισχύει $x_B \geq 0$ και $x_N = 0$.

Με βάση το διαχωρισμό της μήτρας συντελεστών A τα διανύσματα κόστους χωρίζονται σε βασικά και μη βασικά μέρη ως εξής

$$c=\begin{bmatrix} c_B \\ c_N \end{bmatrix}$$

Έτσι η αντικειμενική συνάρτηση γράφεται

$$z = c_B^T x_B \quad + \quad c_N^T x_N$$

Αντικαθιστώντας την τιμή του x_B η παραπάνω εξίσωση γίνεται

$$z = c_B^T (B^{-1} b - B^{-1} N x_N) + c_N^T x_N$$

$$z = c_B^T B^{-1} b - (c_B^T B^{-1} N + c_N^T) x_N$$

Συνεπώς το z ισούται με μια σταθερά $c_B^T B^{-1} b$ μείον τον όρο $(c_B^T B^{-1} N + c_N^T) x_N$. Αν θέσουμε $x_N=0$ η z μετατρέπεται σε $z = c_B^T B^{-1} b$ η οποία είναι η τιμή της αντικειμενικής συνάρτησης που αντιστοιχεί στην τρέχουσα βασική λύση. Συγκεντρωτικά η λύση ενός ακραίου σημείου μπορεί να παρασταθεί σε κανονική μορφή ως εξής

$$z = c_B^T B^{-1} b - (c_B^T B^{-1} N + c_N^T) x_N$$

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

ενώ η αντίστοιχη βασική εφικτή λύση δίνεται από τις εξισώσεις

$$z = c_B^T B^{-1} b$$

$$x=\begin{bmatrix} x_B \\ x_N \end{bmatrix} = \begin{bmatrix} B^{-1} b \\ 0 \end{bmatrix}$$

Η λύση του δυϊκού προβλήματος που αντιστοιχεί στη βάση B δίνεται από τη σχέση :

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

όπου w είναι πολλαπλασιαστές simplex (simplex multipliers) και υπολογίζονται από τη σχέση:

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

Η αντίστοιχη βάση B ονομάζεται δυϊκά εφικτή αν ισχύει $s \geq 0$. Ισχύει επίσης ότι $s_B = 0$.

Μεταξύ των λύσεων x του πρωτεύοντος προβλήματος και s του δυϊκού ισχύει ότι μια βασική λύση είναι βέλτιστη αν $x_B \geq 0$ και $s_N \geq 0$.

2.3 Αλγόριθμοι τύπου simplex

Κάθε αλγόριθμος τύπου simplex κατασκευάζει μια πεπερασμένη ακολουθία βασικών λύσεων. Αν η αντίστοιχη βασική λύση (x_B , x_N) είναι βέλτιστη, οι υπολογισμοί

σταματούν. Διαφορετικά γίνεται προσπάθεια να βρεθούν δύο δείκτες $k \in B$ και $l \in N$. Αν η προσπάθεια αυτή δεν είναι επιτυχής, οι υπολογισμοί σταματούν. Σε περίπτωση που δεν μπορεί να βρεθεί ο δείκτης k το πρόβλημα είναι απεριόριστο. Αν βρεθούν και οι δύο δείκτες, αλλάζουν σύνολα δεικτών. Ο δείκτης k από βασικός γίνεται μη βασικός και ο δείκτης l από μη βασικός γίνεται βασικός. Οι σχέσεις που περιγράφουν την αλλαγή αυτή είναι οι εξής:

$$B_{new} \leftarrow B \setminus \{k\} \cup \{l\}$$

και

$$N_{new} \leftarrow N \setminus \{l\} \cup \{k\}$$

Κατασκευάζεται έτσι μία νέα βάση και η διαδικασία επαναλαμβάνεται. Η διαδικασία εναλλαγής των δεικτών (k, l) ονομάζεται περιστροφή. Η μεταβλητή x_l που μπαίνει στη βάση ονομάζεται εισερχόμενη μεταβλητή, ενώ η x_k που αφήνει τη βάση ονομάζεται εξερχόμενη μεταβλητή.

Ο δείκτης της εισερχόμενης μεταβλητής βρίσκεται στη θέση t του συνόλου N, ενώ ο δείκτης της εξερχόμενης μεταβλητής βρίσκεται στη θέση r του συνόλου B. Ισχύουν δηλαδή οι σχέσεις:

$$N(t) = l \qquad \text{και} \qquad B(r) = k$$

Η ανανέωση των συνόλων B και N γίνεται θέτοντας:

$$N(t) = k \qquad \text{και} \qquad B(r) = l$$

Δύο βάσεις είναι γειτονικές αν διαφέρουν μόνο ως προς ένα δείκτη. Κατά την διαδικασία δημιουργίας νέας βάσης, είναι πιθανό η νέα αυτή βάση να είναι ίδια με κάποια άλλη που είχε δημιουργηθεί σε προηγούμενη επανάληψη. Το φαινόμενο αυτό ονομάζεται κύκλωση.

Όλοι οι αλγόριθμοι τύπου simplex αποτελούνται από τα παρακάτω τέσσερα βήματα:

Βήμα 0: (Αρχικοποίηση-Initialization). Υπολογισμός όλων των μεταβλητών που είναι απαραίτητες για να αρχίσει να δουλεύει ο αλγόριθμος.

Βήμα 1:(Έλεγχος βελτιστότητας-Optimality test). Πραγματοποίηση ελέγχου βελτιστότητας. Αν το τρέχον βασικό σημείο είναι βέλτιστο οι υπολογισμοί σταματούν και ο αλγόριθμος τερματίζεται.

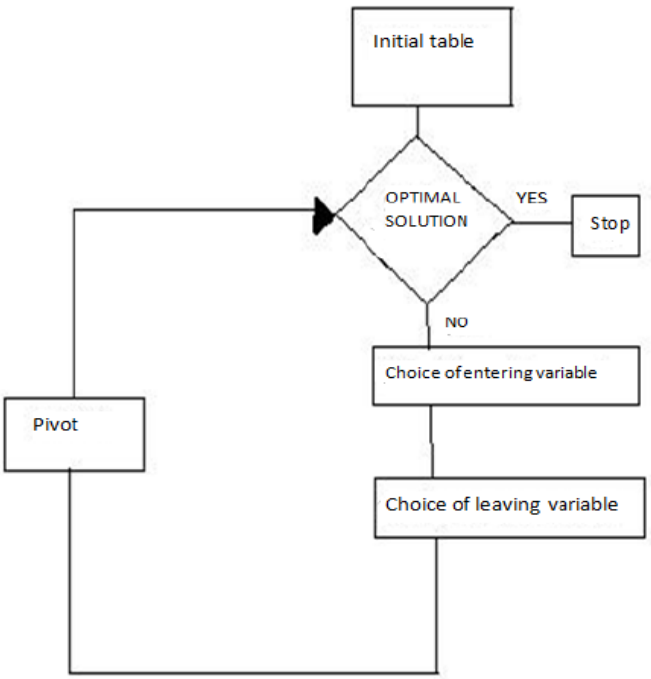
Βήμα 2: (Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής-choice of entering and leaving variable). Η επιλογή αυτή μπορεί να γίνει εφαρμόζοντας διαφορετικούς κανόνες περιστροφής. Γεωμετρικά, η περιστροφή αυτή

σημαίνει μετακίνηση από ένα βασικό σημείο σε ένα άλλο με καλύτερη αντικειμενική τιμή.

Βήμα 3: (Περιστροφή-Pivoting). Υπολογισμός της νέας βάσης, των τιμών των βασικών μεταβλητών και των δυϊκών χαλαρών μεταβλητών.

Μετά την εκτέλεση του βήματος 3, ο αλγόριθμος επιστρέφει στο βήμα 1 και επαναλαμβάνει την διαδικασία ώσπου να εντοπιστεί αν υπάρχει βέλτιστη λύση. Για το λόγο αυτό οι αλγόριθμοί τύπου simplex ονομάζονται και επαναληπτικοί αλγόριθμοι. Το βήμα 0 εκτελείται μόνο μία φορά κατά την έναρξη του αλγόριθμου.

Σε μορφή διαγράμματος ο αλγόριθμος είναι:



Σχήμα 2.3.1 Διάγραμμα ροής αλγόριθμου simplex

Παράδειγμα

Έστω το παρακάτω γραμμικό πρόβλημα

$$\max \quad 2x_1 + 3x_2$$

$$\text{μ.π.} \quad x_1 - 2x_2 \leq 4$$

$$2x_1 + x_2 \leq 18$$

$$+ x_2 \leq 10$$

Μετατρέπουμε το πρόβλημα στην τυποποιημένη μορφή

$$\begin{aligned} \max \quad & 2x_1 + 3\,x_2 \\ \mu.\pi. \quad & x_1 - 2x_2 + x_3 = 4 \\ & 2x_1 + x_2 + x_4 = 18 \\ & x_2 + x_5 = 10 \\ & x_1, x_2, x_3, x_4, x_5 \geq 0 \end{aligned}$$

Επανάληψη 1

Βήμα 0 (Αρχικοποίηση)

Από το αρχικό πρόβλημα φαίνεται ότι τα βασικά εφικτά σημεία είναι τα $x_1 = 0$, $x_2 = 0$, $x_3 = 4$, $x_4 = 18$, $x_5 = 10$. Κατασκευάζουμε έτσι το αρχικό simplex ταμπλό

	z	x_1	x_2	x_3	x_4	x_5	Δεξιό μέρος
z	1	-2	-3	0	0	0	0
x_3	0	1	-2	1	0	0	4
x_4	0	2	1	0	1	0	18
x_5	0	0	1	0	0	1	10

Βήμα 1.Επιλογή εισερχόμενης μεταβλητής

$$a_{0s} = \min\{ a_{01}, a_{02} \} = \min\{ -2,-3\} = -3 = a_{02}$$

Επειδή s= 2 η μεταβλητή x_2 μπαίνει στη βάση.

Βήμα 2. Επιλογή εξερχόμενης μεταβλητής. Θέσε I={2,3}.

$$\frac{b_r}{a_{r2}} = \min\left\{\frac{b_2}{a_{22}}, \frac{b_3}{a_{32}}\right\} = \min\left\{\frac{18}{1}, \frac{10}{1}\right\} = 10$$

Άρα r = 3 και η μεταβλητή x_5 βγαίνει από τη βάση.

Βήμα 3 (Περιστροφή)

Το επόμενο ταμπλό με το στοιχείο περιστροφής a_{32} είναι

	z	x_1	x_2	x_3	x_4	x_5	Δεξιό μέρος
z	1	-2	0	0	0	3	30
x_3	0	1	0	1	0	2	4
x_4	0	2	0	0	1	-1	8
x_2	0	0	1	0	0	1	10

Επειδή $a_{01} < 0$ επαναλαμβάνουμε την παραπάνω διαδικασία

Επανάληψη 2

Βήμα 1.Επιλογή εισερχόμενης μεταβλητής

$$a_{0s} = \min\{ a_{01}, a_{02} \} = \min\{ -2,0\} = -2 = a_{01}$$

Επειδή s= 1 η μεταβλητή x_1 μπαίνει στη βάση.

Βήμα 2. Επιλογή εξερχόμενης μεταβλητής. Θέσε I={1,2}.

$$\frac{b_r}{a_{r1}} = \min\left\{\frac{b_1}{a_{11}}, \frac{b_2}{a_{21}}\right\} = \min\left\{\frac{4}{1}, \frac{8}{2}\right\} = 4$$

Άρα r = 2 και η μεταβλητή x_4 βγαίνει από τη βάση.

Βήμα 3 (Περιστροφή)

Το επόμενο ταμπλό με το στοιχείο περιστροφής a_{21} είναι

	z	x_1	x_2	x_3	x_4	x_5	Δεξιό μέρος
z	1	0	0	0	1	2	38
x_3	0	0	0	1	-1/2	5/2	20
x_1	0	1	0	0	1/2	-1/2	4
x_2	0	0	1	0	0	1	10

Επειδή $a_{0j} \geq 0$ όπου j= 1,2,..,5 στην αντικειμενική γραμμή stop. Η λύση είναι βέλτιστη και είναι η x_1 =4, x_2 = 10, x_3 = 20, x_4 =0 και x_5 =0. Η βέλτιστη τιμή της αντικειμενικής συνάρτησης είναι z= 38 .

2.4 Αναθεωρημένος αλγόριθμος simplex

Έστω ότι υπάρχει μια εφικτή βάση B του γραμμικού προβλήματος. Ο αναθεωρημένος αλγόριθμος simplex κατασκευάζει σε κάθε επανάληψη το παρακάτω ταμπλό που ονομάζεται αναθεωρημένο ταμπλό simplex.

Basis Inversion	RHS
$w^T = c_B^T B^{-1}$	$z= w^T b$
B^{-1}	$x_B = B^{-1}b$

Ο αλγόριθμος ξεκινάει τους υπολογισμούς με το τμήμα αντιστροφής της βάσης έχοντας στη γραμμή κόστους μηδενικά $w^T = c_B^T B^{-1}=0$ και στις υπόλοιπες γραμμές τη μοναδιαία μήτρα. Κατά την διάρκεια εκτέλεσης του αλγόριθμου , οι υπολογισμοί που πραγματοποιούνται είναι οι παρακάτω:

- Τιμές βασικών μεταβλητών: $x_B=B^{-1}b$
- Τιμή αντικειμενικής συνάρτησης: $z= w^T b$
- Στήλη περιστροφής: $h_l=B^{-1} A_{.l}$
- Δυϊκές χαλαρές μεταβλητές: $s_N= c_N^T - c_B^T B^{-1}N$

Σε κάθε επανάληψη επιλέγεται μία εισερχόμενη x_i και μία εξερχόμενη x_j μεταβλητή. Οι δείκτες i,j ονομάζονται επιλέξιμοι. Σε περίπτωση που υπάρχουν περισσότεροι του ενός επιλέξιμοι δείκτες, παρουσιάζεται το φαινόμενο του δεσμού. Αν γίνει τυχαία επιλογή ενός δείκτη ως δείκτη περιστροφής, τότε ο αλγόριθμος μπορεί να κάνει κύκλωση και να μην τερματίσει.

Υπάρχουν πολλές στρατηγικές για τη σωστή επιλογή των δεικτών. Κάποιες από αυτές αναφέρονται παρακάτω.

Ο κανόνας του Dantizg

Ο κανόνας του Dantzig ονομάζεται και κανόνας ελαχίστου στοιχείου (maximum coefficient rule). Το κριτήριο σύμφωνα με το οποίο επιλέγεται η εισερχόμενη μεταβλητή είναι το εξής:

$$s_l = \min\{ s_j : s_j < 0 \wedge j \in N\}$$

Η εισερχόμενη μεταβλητή είναι η s_l με $l= N(t)$. Δεν υπάρχει ιδιαίτερος προσδιορισμός της εξερχόμενης μεταβλητής. Ο κανόνας αυτός κάνει κύκλωση αλλά έχει αποδειχθεί αποτελεσματικός στην πράξη. Επιλέγει σαν εισερχόμενη τη μη βασική μεταβλητή που προκαλεί τη μεγαλύτερη ελάττωση της τιμής της αντικειμενικής συνάρτησης ανά μονάδα αύξησης της εισερχόμενης μεταβλητής. Το μειονέκτημα της είναι ότι απαιτεί αυξημένη υπολογιστική εργασία. Η πολυπλοκότητα του κανόνα του Dantizg είναι εκθετική στην χειρότερη περίπτωση.

Ο κανόνας της μέγιστης βελτίωσης (maximum improvement rule)

Σαν εισερχόμενη μεταβλητή επιλέγεται αυτή που φέρει τη μεγαλύτερη βελτίωση στην τιμή της αντικειμενικής συνάρτησης σε σχέση με κάθε άλλη αποδεκτή μεταβλητή. Έχει το πλεονέκτημα ότι μειώνει τον αριθμό των επαναλήψεων του αλγόριθμου αλλά και το μειονέκτημα ότι απαιτεί μεγάλο υπολογιστικό χρόνο σε κάθε επανάληψη. Η πολυπλοκότητα της χειρότερης περίπτωσης για τη μέθοδο αυτή είναι εκθετική.

Ο κανόνας του Bland

Ονομάζεται και κανόνας του ελαχίστου δείκτη. Επιλέγει ως δείκτη περιστροφής εκείνον ο οποίος έχει τη μικρότερη τιμή τόσο για την εισερχόμενη, όσο και για την εξερχόμενη μεταβλητή. Η μέθοδος αυτή αποφεύγει την κύκλωση και τερματίζει πάντα μετά από πεπερασμένο αριθμό επαναλήψεων. Παρουσιάζει όμως το φαινόμενο της παγίωσης (stalling) σύμφωνα με το οποίο ο αλγόριθμος δεν μπορεί να προχωρήσει σε καλύτερη λύση σε κάθε επανάληψη. Το φαινόμενο αυτό έχει σαν συνέπεια την αύξηση του αριθμού επαναλήψεων του αλγόριθμου γεγονός που τον καθιστά μη αποτελεσματικό στην πράξη.

Ο κανόνας της στοίβας (stack rule)

Ο κανόνας της στοίβας χρησιμοποιεί δύο στοίβες, μια για τις βασικές μεταβλητές και μια για τις μη βασικές μεταβλητές. Η επιλογή των μεταβλητών γίνεται από την κορυφή της μίας στοίβας. Αυτές που επιλέχθηκαν, τοποθετούνται στην κορυφή της άλλης στοίβας σύμφωνα με την βασική πειθαρχία LIFO (Last In First Out). Ο κανόνας αυτός δεν παρουσιάζει κύκλωση αλλά δεν είναι αποτελεσματικός σε πρακτικά προβλήματα.

Ο κανόνας της ουράς(queue rule)

Ο κανόνας αυτός χρησιμοποιεί όπως και ο κανόνας της στοίβας δύο στοίβες, μια για τις βασικές μεταβλητές και μια για τις μη βασικές μεταβλητές. Η επιλογή των μεταβλητών γίνεται από την κορυφή της μίας στοίβας. Αυτές που επιλέχθηκαν, τοποθετούνται στην ουρά της άλλης στοίβας σύμφωνα με την βασική πειθαρχία FIFO (First In First Out).

Ο κανόνας της μερικής αποτίμησης (partial pricing rule)

Σύμφωνα με τον κανόνα αυτό το σύνολο των μη βασικών μεταβλητών χωρίζεται σε ισοπλήθη υποσύνολα. Γίνεται υπολογισμός των s_j του πρώτου υποσυνόλου. Αν υπάρχουν αρνητικά s_j επιλέγεται κάποια εισερχόμενη μεταβλητή από τις μεταβλητές του πρώτου υποσυνόλου. Διαφορετικά προχωράμε στο δεύτερο υποσύνολο και επαναλαμβάνουμε την διαδικασία κοκ. Το πλεονέκτημα της μεθόδου είναι λιγότεροι υπολογισμοί σε κάθε επανάληψη.

Ο κανόνας της πιο κατηφορικής ακμής (steepest edge rule)

Ο κανόνας αυτός βασίζεται στη γεωμετρία του γραμμικού προβλήματος. Μια κατεύθυνση κίνησης αντιστοιχεί σε αύξηση μιας μη βασικής μεταβλητής. Από όλες τις δυνατές κατευθύνσεις βελτίωσης επιλέγεται εκείνη η οποία σχηματίζει τη μικρότερη γωνία με το διάνυσμα –c. Οδηγεί στη βέλτιστη λύση με λιγότερες επαναλήψεις από οποιαδήποτε άλλη μέθοδο, αλλά έχει τις μεγαλύτερες υπολογιστικές απαιτήσεις ανά επανάληψη.

Για την επιλογή της εξερχόμενης μεταβλητής χρησιμοποιείται ο έλεγχος ελαχίστου λόγου που υπολογίζεται από την ακόλουθη σχέση:

$$x_k = x_{B[r]} = \frac{(B^{-1}b)_i}{h_{rl}} = \min\left\{\frac{(B^{-1}b)_i}{h_{rl}} : h_{rl} > 0 \ \wedge i = 1,2,..,m\right\}$$

όπου το $h = B^{-1} A_{.l}$. Η εξερχόμενη μεταβλητή είναι η x_k με k= B(r). Σε περίπτωση που δεν υπάρχει $h_{rl} > 0$ το γραμμικό πρόβλημα είναι απεριόριστο. Ο αριθμός r ονομάζεται δείκτης περιστροφής και το στοιχείο h_{rl} στοιχείο περιστροφής. Στην περίπτωση που το τρέχον σημείο είναι εκφυλισμένο ισχύει $x_k = 0$ και δεν υπάρχει βελτίωση στην αντικειμενική συνάρτηση σε σχέση με την προηγούμενη επανάληψη. Εάν ο έλεγχος ελαχίστου λόγου προσδιορίζει περισσότερους από έναν επιλέξιμους δείκτες, επιλέγεται ως δείκτης περιστροφής εκείνος ο οποίος έχει τη μικρότερη τιμή, δηλαδή: $k = \min\{k_1,k_2,\ldots,k_p\}$ με $p \leq m$.

Όλοι οι αλγόριθμοι τύπου simplex έχουν το μειονέκτημα ότι απαιτούν πολύ χρόνο για τον υπολογισμό της αντιστροφής της βάσης. Το 70% με 80% του χρόνου εκτέλεσης του αλγόριθμου, αναλώνεται σε αυτό τον υπολογισμό. Όσο αυξάνονται οι

διαστάσεις ενός προβλήματος τόσο μεγαλώνει και ο χρόνος αυτός. Για το λόγο αυτό έχουν αναπτυχθεί διάφορες τεχνικές, με στόχο τη βελτίωση του χρόνου υπολογισμού της αντιστροφής της βάσης. Κάτωθι παρουσιάζονται δύο από αυτές.

Μορφή γινομένου του αντιστρόφου (Product Form of the Inverse)

Μια μήτρα Ε ονομάζεται στοιχειώδης (elementary) αν διαφέρει από τη μοναδιαία μήτρα Ι κατά μια μόνο στήλη. Έστω ότι η μήτρα Ε διαφέρει από την Ι στη στήλη j, τότε συμβολίζεται:

$$E=\begin{bmatrix} 1 \dots & a_{1j} & \dots 0 \\ \vdots & a_{ij} & \vdots \\ 0 \dots & a_{mj} & \dots 1 \end{bmatrix}$$

Η αντιστροφή της μήτρας Ε είναι η μοναδιαία όπου η στήλη j έχει αντικατασταθεί από την στήλη $-a_{ij}/ a_{jj}$ για $i \neq j$ και από $1/ a_{jj}$ για i=j. Δηλαδή η E^{-1} είναι η εξής:

$$E^{-1}=\begin{bmatrix} 1 \dots & -a_{1j}/a_{jj} & \dots 0 \\ \vdots & 1/a_{jj} & \vdots \\ 0 \dots & -a_{mj}/a_{jj} & \dots 1 \end{bmatrix}$$

Η νέα αντιστροφή B^{-1} της Β υπολογίζεται σε κάθε επανάληψη από το γινόμενο $E^{-1}\ B_{old}^{-1}$, δηλαδή :

$$B_{new}^{-1} = E^{-1}\ B_{old}^{-1}$$

Μπορούμε να παρατηρήσουμε ότι ο πίνακας E^{-1} είναι αραιός και σε κάθε επανάληψη περιέχει 2m-1 μη μηδενικά στοιχεία. Αν μετρήσουμε μόνο τους πολλαπλασιασμούς και τις διαιρέσεις ανάμεσα σε μη μηδενικούς αριθμούς καταλήγουμε ότι η πολυπλοκότητα της σχέσης $B_{new}^{-1} = E^{-1}\ B_{old}^{-1}$ είναι $\Theta(m^2)$.

Παράδειγμα

Έστω $B_{old}^{-1} = \begin{matrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix}$

Η στήλη που θα εισαχθεί θα είναι η $h_l = \begin{matrix} -3 \\ 2 \\ 1 \end{matrix}$

και θα εισαχθεί στη θέση 2. Τότε η αντίστροφη μήτρα eta είναι:

$$E^{-1} = \begin{matrix} 1 & 3/2 & 0 \\ 0 & 1/2 & 0 \\ 0 & -1/2 & 1 \end{matrix}$$

και ο νέος B^{-1} είναι:

$$B_{new}^{-1} = E^{-1}\, B_{old}^{-1} = \begin{matrix} 1 & 3/2 & 0 \\ 0 & 1/2 & 0 \\ 0 & -1/2 & 1 \end{matrix} * \begin{matrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{matrix} = \begin{matrix} -1 & 3/2 & 0 \\ 0 & 1/2 & 0 \\ 0 & -1/2 & 1 \end{matrix}$$

Τροποποίηση μορφής γινομένου του αντιστρόφου

Σύμφωνα με τη μέθοδο αυτή η τρέχουσα αντίστροφη βάση B_{new}^{-1} μπορεί να υπολογιστεί από την προηγούμενη αντίστροφη βάση B_{old}^{-1} με ένα εξωτερικό γινόμενο δύο διανυσμάτων και μια πρόσθεση πινάκων ως εξής

$$B_{new}^{-1} = \overline{B_{r.}^{-1}} + v \otimes B_{r.}^{-1}$$

όπου $\overline{B_{r.}^{-1}}$ είναι ο πίνακας B_{old}^{-1} κάνοντας την r γραμμή μηδέν, $B_{r.}^{-1}$ είναι η r γραμμή του αντιστρόφου της βάσης και

$$v = \left[-\frac{h_{11}}{h_{r1}} \dots \quad \frac{1}{h_{r1}} \dots \quad -\frac{h_{m1}}{h_{r1}} \right]^T$$

Έτσι έχουμε

$$B_{new}^{-1} = \begin{matrix} b_{11} & \cdots & b_{1m} \\ \dots & \dots & \dots \\ 0 & 0 & 0 \\ \dots & \dots & \dots \\ b_{m1} & \cdots & b_{mm} \end{matrix} + \begin{matrix} -\frac{h_{11}}{h_{r1}} \\ \dots \\ \frac{1}{h_{r1}} \\ \dots \\ -\frac{h_{m1}}{h_{r1}} \end{matrix} * [b_{r1} \dots b_{rr} \dots b_{rm}]$$

Το εξωτερικό γινόμενο απαιτεί m^2 πολλαπλασιασμούς και η πρόσθεση των δύο πινάκων απαιτεί m^2 προσθέσεις. Το συνολικό κόστος της μεθόδου είναι $2m^2$ πράξεις. Άρα η πολυπλοκότητα είναι $\Theta(m^2)$.

Η τροποποιημένη αυτή μέθοδος είναι πιο γρήγορη από την μορφή γινομένου του αντιστρόφου.

Ο αναθεωρημένος αλγόριθμος simplex έχει το πλεονέκτημα ότι απαιτείται η αποθήκευση μόνο των διανυσμάτων εκείνων που σχηματίζουν τις στοιχειώδεις μήτρες και όχι ολόκληρες οι μήτρες.

Τα βήματα του αναθεωρημένου αλγόριθμου simplex είναι τα εξής:

Βήμα 0: (Αρχικοποίηση). Ξεκίνα με μία εφικτή βασική διαμέριση (B,N). Υπολόγισε τη μήτρα B^{-1} και τα διανύσματα x_B , s_N και w.

Βήμα 1:(Έλεγχος βελτιστότητας) . Αν $s_N \geq 0$, STOP. Το πρόβλημα είναι βέλτιστο.

Βήμα 2: (Επιλογή εισερχόμενης και εξερχόμενης μεταβλητής).α) Επέλεξε την εισερχόμενη με τον κανόνα του Dantzig. Η μεταβλητή x_l είναι εισερχόμενη, όπου l=N(t). β) Υπολόγισε τη στήλη περιστροφής h_l. Αν ισχύει $h_l < 0$,STOP, το πρόβλημα είναι απεριόριστο. Διαφορετικά, επέλεξε την εξερχόμενη μεταβλητή με τον έλεγχο ελαχίστου λόγου. Η μεταβλητή x_k είναι εξερχόμενη, όπου $x_{B[r]} = x_k$.

Βήμα 3: (Περιστροφή). Ανανέωσε τα σύνολα δεικτών B και N. Θέσε N(t) = k και B(r)=l. Υπολόγισε τη νέα αντιστροφή της βάσης , B_{new}^{-1} και τα διανύσματα x_B και s_N. Πήγαινε στο βήμα 1.

Παράδειγμα

Έστω το κάτωθι γραμμικό πρόβλημα σε κανονική μορφή:

min $-3x_1 - 2x_2$

μ.π. $2x_1 + 5x_2 + x_3 = 14$

$-x_1 + 2x_2 + x_4 = 4$

Επανάληψη 1

Βήμα 0

Η αρχική διαμέριση είναι B=[3 4] και N=[1 2]

$B = \begin{matrix} 1 & 0 \\ 0 & 1 \end{matrix} = B^{-1}$ $N = \begin{matrix} 2 & 5 \\ -1 & 2 \end{matrix}$

$c_B^T = [\,0\ 0\,]$ $c_N^T = [\,-3\ -2]$

$x_B = B^{-1}b = \begin{matrix} 14 \\ 4 \end{matrix}$

$w^T = c_B^T B^{-1} = [0\ 0]$

$s_N = c_N^T - c_B^T B^{-1} N = [-3\ -2]$

Βήμα 1

Επειδή $s_N < 0$ ο αλγόριθμος δεν σταματά.

Βήμα 2

α) Επιλογή εισερχόμενης

Σύμφωνα με τον κανόνα ελαχίστου στοιχείου επιλέγεται ως εισερχόμενη μεταβλητή η $l=N(t)=N(1)= x_1$.

β) Επιλογή εξερχόμενης

Υπολογίζεται το διάνυσμα h_l

$h_l = B^{-1} A_{.l} = \begin{matrix} 2 \\ -1 \end{matrix}$

Επειδή $h_l > 0$ ο αλγόριθμος δε σταματά και σύμφωνα με το τεστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή

$$x_k = x_{B[r]} = \min\left\{\frac{x_{B[1]}}{h_{r1}}, \frac{x_{B[2]}}{h_{r2}}\right\} = \min\{7,x\}=7$$

Επομένως r=1 και $k=B(r)=B(1)= x_3$

Βήμα 3

Η νέα βάση είναι B=[1 4] και N=[3 2]

$$B = \begin{matrix} 2 & 0 \\ -1 & 1 \end{matrix} \qquad N = \begin{matrix} 1 & 5 \\ 0 & 2 \end{matrix}$$

και τα διανύσματα $c_B^T = [-3\ 0]$ $c_N^T = [0\ -2]$

Υπολογίζεται η νέα αντιστροφή της βάσης

$B^{-1} = \begin{matrix} 1/2 & 0 \\ 1/2 & 1 \end{matrix}$

και τα διανύσματα

$x_B = B^{-1}b = \begin{matrix} 7 \\ 11 \end{matrix}$

$w^T = c_B^T B^{-1} = [-3/2 \;\; 0]$

$s_N = c_N^T - c_B^T B^{-1} N = [3/2 \;\; 11/2]$

Επανάληψη 2

Βήμα 1

Επειδή $s_N > 0$ ο αλγόριθμος σταματά. Η βέλτιστη βασική διαμέριση είναι

B=[1 4] N=[3 2]

Η βέλτιστη λύση είναι:

$x^T = [x_1, x_2, x_3, x_4] = [7\ 0\ 0\ 11]$

Η βέλτιστη τιμή της αντικειμενικής συνάρτησης είναι: $z = w^T b = -21$

2.5 Εύρεση αρχικής εφικτής βάσης

Ο αλγόριθμος που παρουσιάστηκε στην προηγούμενη παράγραφο, προϋποθέτει ότι το γραμμικό πρόβλημα που θα επιλυθεί έχει μια αρχική εφικτή βάση. Στα γενικά γραμμικά προβλήματα μια τέτοια λύση δεν είναι πάντα γνωστή. Σε περίπτωση που δεν δύναται στο γραμμικό πρόβλημα να υπάρχει μια αρχική λύση, χρησιμοποιείται είτε η μέθοδος των δύο φάσεων (two phase method) είτε η μέθοδος του μεγάλου M (big-M method).

2.5.1 Μέθοδος 2 φάσεων

Στη μέθοδο των δύο φάσεων με μία τεχνητή μεταβλητή (two phase method with one artificial variable) ο αλγόριθμος εκτελείται σε δύο φάσεις. Στην πρώτη επιλύεται ένα τροποποιημένο, του αρχικού, γραμμικό πρόβλημα, ενώ στην δεύτερη φάση, επιλύεται το

αρχικό γραμμικό πρόβλημα αν στη φάση ένα έχει βρεθεί ένα αρχικό βασικό εφικτό σημείο.

Το τροποποιημένο πρόβλημα της φάσης ένα κατασκευάζεται ως εξής. Εισάγεται μια νέα μεταβλητή x_{n+1} η οποία ονομάζεται τεχνητή (artificial) με συντελεστές d= - B e, όπου e ένα διάνυσμα διαστάσεων mx1 με όλες τις συνιστώσες του μονάδες. Το πρόβλημα που επιλύεται έχει τη μορφή:

$$\begin{aligned} &\min && x_{n+1} \\ &\text{μ.π.} && Ax + dx_{n+1} = b \\ & && x , x_{n+1} \geq 0 \end{aligned}$$

Αρχικά πραγματοποιείται μια περιστροφή έτσι ώστε η τεχνητή μεταβλητή x_{n+1} να εισέλθει στη βάση. Ως εξερχόμενη μεταβλητή επιλέγεται εκείνη με τη μικρότερη τιμή δεξιού μέρους σύμφωνα με τη σχέση:

$$x_k = x_{B[r]} = \min\{x_{B[i]} : i=1,2,\ldots,m\}$$

και γίνεται ανανέωση των συνόλων των δεικτών σύμφωνα με τις σχέσεις:

$$B_{new} \leftarrow B \setminus \{k\} \cup \{n+1\}$$

και

$$N_{new} \leftarrow N \setminus \{n+1\} \cup \{k\}$$

Η στήλη περιστροφής είναι αυτή που αντιστοιχεί στην τεχνητή μεταβλητή και υπολογίζεται από τη σχέση:

$$h_{n+1}=B^{-1} A_{.n+1}$$

Εν συνεχεία, υπολογίζονται η νέα μήτρα B^{-1} και τα διανύσματα x_B , s_N, από τη νέα βασική διαμέριση που προέκυψε. Η περιστροφή αυτή που εισάγει την μεταβλητή x_{n+1} στη βάση και εξάγει την x_k, εγγυάται την κατασκευή ενός εφικτού σημείου για το τροποποιημένο πρόβλημα. Ο αλγόριθμος αρχίζει και εκτελείται, όπως στη φάση δύο. Αυτό που διαφοροποιείται μόνο, είναι η συνθήκη τερματισμού. Σταματάει όταν η τεχνητή μεταβλητή εξέλθει από την βάση ή όταν βρεθεί μια βέλτιστη λύση στο τροποποιημένο πρόβλημα.

Στην πρώτη περίπτωση ο αλγόριθμος μεταβαίνει στη φάση δύο χρησιμοποιώντας ως διαμέριση (B,N) αυτή που έχει δημιουργηθεί από τη φάση ένα, αφαιρώντας όμως την τεχνητή μεταβλητή από τη μήτρα N και επαναφέροντας τους αρχικούς συντελεστές κόστους.

Στη δεύτερη περίπτωση απαιτείται ένας επιπλέον έλεγχος. Αν το $x_{n+1} > 0$, τότε το πρόβλημα είναι αδύνατο. Αν το $x_{n+1} = 0$, εκτελείται μια ακόμα περιστροφή για να

βγει η τεχνητή μεταβλητή από τη βάση και να εισέλθει ο αλγόριθμος στη φάση δύο με τον τρόπο που αναφέρθηκε στην πρώτη περίπτωση.

Η παραπάνω διαδικασία συνοψίζεται στα κάτωθι βήματα:

Βήμα 0: (Έλεγχος εφικτότητας). Ξεκίνα με μία εφικτή βασική διαμέριση (B,N). Υπολόγισε τη μήτρα B^{-1} και τα διανύσματα x_B , s_N. Αν ισχύει $x_B=B^{-1}b\geq 0$ πήγαινε στο βήμα 3. Διαφορετικά υπολόγισε τους συντελεστές της μεταβλητής x_{n+1} .

Βήμα 1:(Περιστροφή) . Επέλεξε την μεταβλητή x_{n+1} ως εισερχόμενη και αυτή με το μικρότερο δεξιό μέρος ως εξερχόμενη. Ανανέωσε τα σύνολα δεικτών B και N. Υπολόγισε τη στήλη περιστροφής, την αντιστροφή της βάσης B^{-1} και τα x_B , s_N.

Βήμα 2: (Φάση I). Εφάρμοσε τον αναθεωρημένο αλγόριθμο simplex για την νέα βασική διαμέριση (B,N). Αν $x_{n+1} \in B$ και $x_{n+1} > 0$,STOP, το πρόβλημα είναι αδύνατο. Αν $x_{n+1} \in B$ και $x_{n+1} = 0$,κάνε μια ακόμα περιστροφή. Διαφορετικά, πήγαινε στο βήμα 3.

Βήμα 3: (Φάση II). Χρησιμοποιώντας τη βασική διαμέριση που προέκυψε από τη φάση I εκτέλεσε τον αναθεωρημένο αλγόριθμο simplex για το αρχικό γραμμικό πρόβλημα.

Παράδειγμα

Έστω το κάτωθι γραμμικό πρόβλημα σε κανονική μορφή:

$$\min \quad x_2 + 10\,x_3$$

$$\text{μ.π.} \quad -2x_1 - x_2 + 4\,x_3 + x_4 = -4$$

$$3x_1 + x_2 - x_3 - x_5 = 5$$

Η αρχική διαμέριση είναι B=[4 5] και N=[1 2 3]

$$B^{-1} = \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$$

$$x_B=B^{-1}b= \begin{matrix} -4 \\ -5 \end{matrix}$$

Επειδή $x_B < 0$ η λύση αυτή δεν είναι εφικτή. Προσθέτουμε μια τεχνητή μεταβλητή x_6 και υπολογίζουμε τους συντελεστές της

$$d = -B\,e = -\begin{matrix}1 & 0\\ 0 & -1\end{matrix} * \begin{matrix}1\\ 1\end{matrix} = \begin{matrix}-1\\ 1\end{matrix}$$

Το νέο προς επίλυση πρόβλημα είναι το εξής:

min x_6

μ.π. $-2x_1 - x_2 + 4x_3 + x_4 - x_6 = -4$

$3x_1 + x_2 - x_3 - x_5 + x_6 = 5$

Φάση I-Επανάληψη 1

Βήμα 0

min $\{x_B\} = -5$, άρα r =2

B= [4 6] N=[1 2 3 5]

$c_B^T = [0\ 1]$ $c_N^T = [0\ 0\ 0\ 0]$

$$B = \begin{matrix}1 & -1\\ 0 & 1\end{matrix} \qquad N = \begin{matrix}-2 & -1 & 4 & 0\\ 3 & 1 & -1 & -1\end{matrix}$$

$$B^{-1} = \begin{matrix}1 & 1\\ 0 & 1\end{matrix}$$

$$x_B = B^{-1}b = \begin{matrix}1\\ 5\end{matrix}$$

$w^T = c_B^T B^{-1} = [0\ 1]$

$s_N = c_N^T - c_B^T B^{-1} N = [-3\ -1\ 1\ 1]$

Βήμα 1

Επειδή $s_N < 0$ ο αλγόριθμος δεν σταματά.

Βήμα 2

α) Επιλογή εισερχόμενης

Σύμφωνα με τον κανόνα ελαχίστου στοιχείου επιλέγεται ως εισερχόμενη μεταβλητή η l=N(t)=N(1)= x_1 .

β)Επιλογή εξερχόμενης

Υπολογίζεται το διάνυσμα h_l

$h_l=B^{-1} A_{.l}= \begin{matrix} 1 \\ 3 \end{matrix}$

Επειδή $h_l > 0$ ο αλγόριθμος δε σταματά και σύμφωνα με το τεστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή

$x_k = x_{B[r]} = \min\left\{\frac{x_{B[1]}}{h_{r1}}, \frac{x_{B[2]}}{h_{r2}}\right\} = \min\{1,5/3\}=1$

Επομένως r=1 και k=B(r)=B(1)= x_4

Βήμα 3

Η νέα βάση είναι B=[1 6] και N=[4 2 3 5]

$B = \begin{matrix} -2 & -1 \\ 3 & 1 \end{matrix}$ $N= \begin{matrix} 1 & -1 & 4 & 0 \\ 0 & 1 & -1 & -1 \end{matrix}$

και τα διανύσματα c_B^T = [0 1] c_N^T = [0 0 0 0]

Υπολογίζεται η νέα αντιστροφή της βάσης

$B^{-1} = \begin{matrix} 1 & 1 \\ -3 & 2 \end{matrix}$

και τα διανύσματα

$x_B=B^{-1}b= \begin{matrix} 1 \\ 2 \end{matrix}$

$w^T = c_B^T B^{-1} = [-3\ -2]$

$s_N = c_N^T - c_B^T B^{-1} N = [3\ -1\ 10\ -2]$

Επανάληψη 2

Βήμα 1

Επειδή $s_N < 0$ ο αλγόριθμος δεν σταματά.

Βήμα 2

α)Επιλογή εισερχόμενης

Σύμφωνα με τον κανόνα ελαχίστου στοιχείου επιλέγεται ως εισερχόμενη μεταβλητή η $l=N(t)=N(4)= x_5$.

β)Επιλογή εξερχόμενης

Υπολογίζεται το διάνυσμα h_l

$h_5=B^{-1} A_{.5}= \begin{matrix} -1 \\ 2 \end{matrix}$

Επειδή $h_5 > 0$ ο αλγόριθμος δε σταματά και σύμφωνα με το τεστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή

$$x_k = x_{B[r]} = \min\left\{\frac{x_{B[1]}}{h_{r1}}, \frac{x_{B[2]}}{h_{r2}}\right\} = \min\{x,1\}=1$$

Επομένως r=2 και $k=B(r)=B(2)= x_6$

Εξερχόμενη μεταβλητή είναι η τεχνητή, άρα περνάμε στη φάση δύο

Φάση ΙΙ-Επανάληψη 3

Βήμα 0

B= [1 5] N= [4 2 3]

$$B = \begin{matrix} -2 & 0 \\ 3 & 1 \end{matrix} \qquad N= \begin{matrix} 1 & -1 & 4 \\ 0 & 1 & -1 \end{matrix}$$

$c_B^T = [\ 0\ 0]$ $c_N^T = [\ 0\ 1\ 10]$

$$B^{-1} = \begin{matrix} -1/2 & 0 \\ -3/2 & -2 \end{matrix}$$

$x_B = B^{-1}b = \begin{matrix} 2 \\ 1 \end{matrix}$

$w^T = c_B^T B^{-1} = [0\ 0]$

$s_N = c_N^T - c_B^T B^{-1} N = [0\ 1\ 10]$

Βήμα 1

Επειδή $s_N \geq 0$ ο αλγόριθμος σταματά. Η τρέχουσα λύση είναι βέλτιστη.

2.5.2 Μέθοδος μεγάλου Μ

Στη μέθοδο του μεγάλου Μ δεν λύνονται δύο διαφορετικά προβλήματα αλλά ένα τροποποιημένο του αρχικού πρόβλημα. Το πρόβλημα αυτό που ονομάζεται πρόβλημα του μεγάλου Μ (big M problem) έχει τις ίδιες μεταβλητές και τους ίδιους περιορισμός με το πρόβλημα της φάσης Ι. Η αντικειμενική συνάρτηση όμως αποτελείται από δύο όρους. ο ένας εκ των οποίων είναι η αντικειμενική συνάρτηση του αρχικού προβλήματος ενώ ο άλλος είναι η τεχνητή μεταβλητή πολλαπλασιασμένη με ένα πάρα πολύ μεγάλο αριθμό Μ της τάξης του 10^{20} ή 10^{30}. Η τιμή του Μ μπορεί να υπολογιστεί από τα δεδομένα του προβλήματος.

Από τη λύση του τροποποιημένου προβλήματος μπορούμε να εξάγουμε συμπεράσματα για το αρχικό πρόβλημα. Πιο συγκεκριμένα, αν το πρόβλημα του μεγάλου Μ είναι απεριόριστο, τότε το αρχικό πρόβλημα είναι είτε απεριόριστο είτε αδύνατο. Αν η τιμή της τεχνητής μεταβλητής στο πρόβλημα του μεγάλου Μ είναι μηδέν και αυτό είναι βέλτιστο, τότε το αρχικό πρόβλημα είναι επίσης βέλτιστο.

Το πρόβλημα που επιλύεται στη μέθοδο του μεγάλου Μ έχει τη μορφή:

$$\begin{aligned} \min \quad & c^T x + M x_{n+1} \\ \text{μ.π.} \quad & Ax + d x_{n+1} = b \\ & x, x_{n+1} \geq 0 \end{aligned}$$

Παράδειγμα

Έστω το κάτωθι γραμμικό πρόβλημα σε κανονική μορφή:

$$\begin{aligned} \min \quad & x_2 + 10 x_3 \\ \text{μ.π.} \quad & -2x_1 - x_2 + 4 x_3 + x_4 = -4 \end{aligned}$$

$$3x_1 + x_2 - x_3 - x_5 = 5$$

Για M=100 το πρόβλημα παίρνει τη μορφή

$$\begin{array}{lllllll} \min & & x_2 + 10x_3 & & +100x_6 & & \\ \mu.\pi. & -2x_1 - x_2 + 4x_3 + x_4 & & - & x_6 & = -4 \\ & 3x_1 + x_2 - x_3 & - x_5 & + & x_6 & = 5 \end{array}$$

Επανάληψη 1

Η πρώτη εφικτή βάση κατασκευάζεται όπως και στη φάση Ι, δηλαδή

B= [4 6] N=[1 2 3 5]

$c_B^T = [0\ 100]$ $c_N^T = [0\ 1\ 10\ 0]$

$$B = \begin{matrix} 1 & -1 \\ 0 & 1 \end{matrix} \qquad N = \begin{matrix} -2 & -1 & 4 & 0 \\ 3 & 1 & -1 & -1 \end{matrix}$$

$$B^{-1} = \begin{matrix} 1 & 1 \\ 0 & 1 \end{matrix}$$

$w^T = c_B^T B^{-1} = [0\ 100]$

$s_N = c_N^T - c_B^T B^{-1} N = [-300\ -99\ 110\ 100]$

Επειδή $s_N < 0$ ο αλγόριθμος δεν σταματά. Σύμφωνα με τον κανόνα ελαχίστου στοιχείου επιλέγεται ως εισερχόμενη μεταβλητή η l=N(t)=N(1)= x_1

Υπολογίζεται τώρα το διάνυσμα h_l

$$h_l = B^{-1} A_{.l} = \begin{matrix} 1 \\ 3 \end{matrix}$$

Επειδή $h_l > 0$ ο αλγόριθμος δε σταματά και σύμφωνα με το τεστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή

$$x_k = x_{B[r]} = \min\left\{\frac{x_{B[1]}}{h_{r1}}, \frac{x_{B[2]}}{h_{r2}}\right\} = \min\{1,5/3\}=1$$

Επομένως r=1 και k=B(r)=B(1)= x_4

Η νέα βάση είναι B=[1 6] και N=[4 2 3 5]

$$B = \begin{matrix} -2 & -1 \\ 3 & 1 \end{matrix} \qquad N = \begin{matrix} 1 & -1 & 4 & 0 \\ 0 & 1 & -1 & -1 \end{matrix}$$

και τα διανύσματα c_B^T = [0 100] c_N^T = [0 1 10 0]

Υπολογίζεται η νέα αντιστροφή της βάσης

$$B^{-1} = \begin{matrix} 1 & 1 \\ -3 & 2 \end{matrix}$$

και τα διανύσματα

$w^T = c_B^T B^{-1}$ = [-300 -200]

$s_N = c_N^T - c_B^T B^{-1} N$= [300 -99 1010 -200]

Επανάληψη 2

Επειδή $s_N < 0$ ο αλγόριθμος δεν σταματά. Σύμφωνα με τον κανόνα ελαχίστου στοιχείου επιλέγεται ως εισερχόμενη μεταβλητή η l=N(t)=N(4)= x_5 .

Υπολογίζεται τώρα το διάνυσμα h_l

$$h_5 = B^{-1} A_{.5} = \begin{matrix} -1 \\ 2 \end{matrix}$$

Επειδή $h_5 > 0$ ο αλγόριθμος δε σταματά και σύμφωνα με το τεστ ελαχίστου λόγου επιλέγεται η εξερχόμενη μεταβλητή

$$x_k = x_{B[r]} = \min\left\{\frac{x_{B[1]}}{h_{r1}}, \frac{x_{B[2]}}{h_{r2}}\right\} = \min\{-1,1\} = 1$$

Επομένως r=2 και k=B(r)=B(2)= x_6

Η νέα βάση είναι B=[1 5] και N=[4 2 3 6]

$$B = \begin{matrix} -2 & 0 \\ 3 & -1 \end{matrix} \qquad N = \begin{matrix} 1 & -1 & 4 & -1 \\ 0 & 1 & -1 & 1 \end{matrix}$$

και τα διανύσματα $c_B^T = [\,0\ 0]$ $c_N^T = [\,0\ 1\ 10\ 100]$

Υπολογίζεται η νέα αντιστροφή της βάσης

$$B^{-1} = \begin{matrix} -1/2 & 0 \\ -3/2 & -1 \end{matrix}$$

και τα διανύσματα

$w^T = c_B^T B^{-1} = [0\ 0]$

$s_N = c_N^T - c_B^T B^{-1} N = [0\ 1\ 10\ 100]$

Επανάληψη 3

Επειδή $s_N > 0$ το τρέχον σημείο είναι βέλτιστο και οι υπολογισμοί σταματούν. Επειδή η τιμή της τεχνητής μεταβλητής στη βέλτιστη λύση είναι μηδέν, το αρχικό πρόβλημα είναι βέλτιστο.

2.6 Σύγκριση simplex και revised simplex

Ο αλγόριθμος simplex απαιτεί την αποθήκευση ενός διανύσματος διαστάσεων (m+1)x(n+1), την ίδια στιγμή που ο αναθεωρημένος αλγόριθμος simplex αποθηκεύει ένα διάνυσμα διαστάσεων (m+1)x(m+1). Όταν ο αριθμός των μεταβλητών n είναι σημαντικά μεγαλύτερος από τον αριθμό των περιορισμών, τότε ο αποθηκευτικός χώρος που καταλαμβάνει στη μνήμη είναι πολύ μεγαλύτερος.

Πίνακας 2.6.1 Σύγκριση πράξεων σε simplex και revised simplex

Πράξεις	Simplex	Revised Simplex
Πολλαπλασιασμοί	m(n-1)+n+1	m(n+2)+1
Προσθέσεις	m(n-m+1)	m(n+1)
Σύνολο	n(2m+1)-m(2m-1)+1	m(2n+3)+1

Πίνακας 2.6.2 Απαίτηση μνήμης σε simplex και revised simplex

	Simplex	Revised Simplex
Μνήμη	O(mn)	$O(m^2)$
Καλύτερη περίπτωση	O(mn)	$O(m^2)$
Χειρότερη περίπτωση	O(mn)	O(mn)

Παρατηρούμε ότι ο αναθεωρημένος αλγόριθμος απαιτεί λίγο περισσότερες πράξεις από ότι ο simplex. Όμως στην πλειοψηφία τους τα γραμμικά προβλήματα είναι ιδιαίτερα αραιά (πυκνότητα της τάξης του 5% και κάτω), με αποτέλεσμα ο αναθεωρημένος αλγόριθμος simplex να μπορεί να αποφύγει πράξεις με μηδενικά.

Κεφάλαιο 3

Μεθοδολογία

3.1 Υλοποίηση Αναθεωρημένου Αλγόριθμου Simplex

Για την υλοποίηση του αλγόριθμου χρησιμοποιήθηκε η γλώσσα προγραμματισμού Python και πιο συγκεκριμένα το περιβάλλον IDLE της έκδοσης 3.5. Οι βιβλιοθήκες που χρησιμοποιήθηκαν είναι οι: numpy, pandas και time. Τα αρχεία που δημιουργήθηκαν είναι τα: func που περιέχει βασικές συναρτήσεις που χρησιμοποιούνται στον αλγόριθμο, data_trans που περιέχει τη συνάρτηση που μετατρέπει το πρόβλημα σε κανονική μορφή και revised_simplex που έχει τον κυρίως αλγόριθμο.

Ο αλγόριθμος είναι δύο φάσεων με μια τεχνητή μεταβλητή και εφαρμόστηκε σε τυχαία πυκνά προβλήματα που δημιουργήθηκαν με γεννήτρια τυχαίων αριθμών. Η επιλογή της εισερχόμενης μεταβλητής έγινε με τον κανόνα του Dantzig. Σε περίπτωση δεσμού γίνεται τυχαία επιλογή εισερχόμενης. Για την αντιστροφή της βάσης εφαρμόστηκε η μορφή γινομένου του αντιστρόφου. Παρακάτω παρουσιάζεται αναλυτικά ο κώδικας όπως υλοποιήθηκε.

```
import numpy as np
import pandas as pd
from func import *
from data_trans import *
import time
from dense_problems import *

stat=[]

n,m,LP,eqinlu,Alu,clu,blu,min_card,den=random_lp()
for i in range(LP):
    flag='feasible'
    Iter=0
    Cpu_time=0
    Cpu_Eta=0
    A,c,b,bounds,obj=optimalrandom(n,m,eqinlu,Alu,clu,blu,min_card,den)
```

```
start=time.time()
if flag=='feasible':
    A,c,b,total_var,extra_var=standard_form(A,c,b,bounds,obj)
    #names for columns in c_rev and A:
    col_names=['x'+str(i+1) for i in range(total_var)]
    c_rev=pd.DataFrame([c],columns=col_names)
    A=pd.DataFrame(A,columns=col_names)

    Iter=1
    #creating tables B and N
    break_point=total_var-extra_var
    B=A.iloc[:,break_point:total_var:1].copy()
    N=A.iloc[:,0:break_point:1].copy()
    B_inv=pd.DataFrame(np.linalg.inv(B),columns=B.columns.tolist())
    x_B=comp_x_B(B_inv,b,B)
    while (x_B<0).any():
        #phase_I
        print('phase_I')
        c_new=c_rev.copy()
        c_new.iloc[0,:]=0
        art_var='x'+str(total_var+1)
        c_new[art_var]=[1]
        e=[1 for i in range(len(A))]
        d=-B @ np.array(e).T
        N['x'+str(total_var+1)]=d
        A['x'+str(total_var+1)]=d
        t=N.columns.get_loc(art_var)
        l=N.columns[t]
        #print('The incoming variable is:',l)
        r=x_B.values.argmin()
        k=B.columns[r]
        #print('The outcoming variable is:',k)
        N[l]=A[k]
```

```
            N=N.rename(columns={l:k})
            B[k]=A[l]
            B=B.rename(columns={k:l})
            B_inv=pd.DataFrame(np.linalg.inv(B),columns=B.columns.tolist())
            x_B=comp_x_B(B_inv,b,B)
            while (x_B.index==art_var).any():
                c_B_T=comp_c_B_T(B,c_new)
                w_T=comp_w_T(c_B_T,B_inv)
                c_N_T=comp_c_N_T(N,c_new)
                s_N=comp_s_N(c_N_T,w_T,N)
                if np.any(s_N<0):
                    t=np.argmin(s_N)
                    l=N.columns[t]
                    #print('The incoming variable is:',l)
                    h_l=comp_h_l(B_inv,A,l)
                    if (h_l>0).any():
                        r=comp_r_I(x_B,h_l,art_var,A,B)
                        k=B.columns[r]
                        #print('The outcoming variable is:',k)
                        N[l]=A[k]
                        N=N.rename(columns={l:k})
                        B[k]=A[l]
                        B=B.rename(columns={k:l})
                        start1=time.time()
                        if Iter%100==0:

B_inv=pd.DataFrame(np.linalg.inv(B),columns=B.columns.tolist())
                        else:
                            E_inv=create_eta_inv(B,r,h_l)
                            B_inv=pd.DataFrame(E_inv @ B_inv)
                        end1=time.time()
                        Eta_time_I=end1-start1
                        Cpu_Eta+=Eta_time_I
```

```
                    x_B=comp_x_B(B_inv,b,B)
                    Iter +=1
                else:
                    flag='unbounded'
                    break
            else:
                #print('Since s_N >0, the algorithm stop')
                break
        if (x_B.index==art_var).any():
            if x_B[art_var]!=0:
                flag='infeasible'
        else:
            del N[art_var]
            del A[art_var]

if flag=='feasible':
    #phase_II
    print('phase_II')
    while (x_B>=0).all():
        c_B_T=comp_c_B_T(B,c_rev)
        w_T=comp_w_T(c_B_T,B_inv)
        c_N_T=comp_c_N_T(N,c_rev)
        s_N=comp_s_N(c_N_T,w_T,N)
        if np.any(s_N<0):
            #print ('Since s_N <0, the algorithm does not stop')
            #print('Phase a: Choosing incoming variable')
            t=np.argmin(s_N)
            l=N.columns[t]
            #print('The incoming variable is:',l)
            #print('Phase b: Choosing outcoming variable')
            h_l=comp_h_l(B_inv,A,l)
            if (h_l>0).any():
                r=comp_r(x_B,h_l,A,B)
```

```
                    k=B.columns[r]
                    #print('The outcoming variable is:',k)
                    N[l]=A[k]
                    N=N.rename(columns={l:k})
                    B[k]=A[l]
                    B=B.rename(columns={k:l})
                    start2=time.time()
                    if Iter%100==0:

B_inv=pd.DataFrame(np.linalg.inv(B),columns=B.columns.tolist())
                    else:
                        E_inv=create_eta_inv(B,r,h_l)
                        B_inv=pd.DataFrame(E_inv @ B_inv)
                    end2=time.time()
                    Eta_time_II=end2-start2
                    Cpu_Eta+=Eta_time_II
                    x_B=comp_x_B(B_inv,b,B)
                    Iter +=1
                else:
                    flag='unbounded'
                    break
            else:
                #print('Since s_N >0, the algorithm stop')
                break
    end=time.time()
    Cpu_time=(end-start)
    if flag=='infeasible':
        print('the problem is',flag)
    else:
        if flag=='unbounded':
            print('The problem is',flag)
        else:
            print('The problem is feasible')
```

```
        z=w_T @b
        z=z[0][0]
        print('Number of iterations:',Iter)
        print('The optimal solution is:','\n',x_B)
        if obj=='min':
            z=z
        else:
            z=-z
        print('The optimal solution of the objective function is z=', z)

    print(' Cpu time of algorithm:',Cpu_time)
    stat.append([i+1,obj,flag,Iter,Cpu_Eta,Cpu_time])
stat=pd.DataFrame(stat,columns=['LP_n','Obj','Solution','Iter_t','Cpu_Inv','Cpu'])
#print(stat)
```

Λίστα Κώδικα 1. Revised Simplex 2 Phase

Μέσα στο αρχείο func περιέχονται οι συναρτήσεις :

Για τον υπολογισμό των c_B^T και c_N^T δημιουργήθηκαν οι συναρτήσεις

```
        def comp_c_B_T(B,c_rev):
            '''gives the values of the reverse DataFrame of c for the variables of B'''
            c_B=c_rev[B.columns]
            return c_B

        def comp_c_N_T(N,c_rev):
            '''gives the values of the reverse DataFrame of c for the variables of N'''
            c_N=c_rev[N.columns]
            return c_N
```

Λίστα Κώδικα 2. Υπολογισμός c_B^T, c_N^T

Για τον υπολογισμό του x_B δημιουργήθηκε η παρακάτω συνάρτηση. Χρησιμοποιήθηκε ανοχή της τάξης του 10^{-8}.

```
def comp_x_B(B_inv,b,B):
    '''computes the DataFrame xB=B_inv*b'''
    tol=1/10**8
    xB=B_inv @ np.array(b)
    xB=xB.squeeze()
    row_names=[i for i in B.columns]
    xB.index=row_names
    xB[abs(xB)<=tol]=0
    return xB
```

Λίστα Κώδικα 3. Υπολογισμός x_B

Για τον υπολογισμό του w^T χρησιμοποιήθηκε η συνάρτηση

```
def comp_w_T(c_B_rev,B_inv):
    '''computes the product c_B_rev*B_inv'''
    wT=np.array(c_B_rev)*np.matrix(B_inv)
    return wT
```

Λίστα Κώδικα 4. Υπολογισμός w^T

Για τον υπολογισμό του s_N χρησιμοποιήθηκε η συνάρτηση στην οποία περιλαμβάνεται ανοχή της τάξης του 10^{-8}

```
def comp_s_N(c_N_rev,w_T,N):
    '''computes the math expression c_N_rev - c_B_rev * B_inv * N'''
    tol=1/10**8
    s_n=np.array(c_N_T) - w_T*(np.matrix(N))
    s_n[abs(s_n)<=tol]=0
    return s_n
```

Λίστα Κώδικα 5. Υπολογισμός s_N

Ο υπολογισμός της στήλης περιστροφής γίνεται από τη συνάρτηση

```
def comp_h_l(B_inv,A,l):
    '''computes the value of h_l'''
    return B_inv @ np.array(A[l])
```

Λίστα Κώδικα 6. Υπολογισμός h_l

Για τον έλεγχο ελαχίστου λόγου και την επιλογή της εξερχόμενης μεταβλητής δημιουργήθηκε η παρακάτω συνάρτηση, η οποία σπάει τους δεσμούς σε περίπτωση που υπάρχουν

```
def comp_r(x_B,h_l,A,B):
    '''finds the position of exporting variable exp_var and breaks the tie'''
    x_k=(x_B/h_l.values).replace((np.inf, -np.inf),(0,0))
    q=x_k.index[x_k==min(x_k[x_k>0])]
    q=q.tolist()
    if len(q)==1:
        exp_var=B.columns.get_loc(q[0])
    else:
        j=min([A.columns.get_loc(i) for i in q])
        s=A.columns[j]
        exp_var=B.columns.get_loc(s)
    return exp_var
```

Λίστα Κώδικα 7. Υπολογισμός εξερχόμενης μεταβλητής

Για τη φάση Ι η παραπάνω συνάρτηση είναι ελαφρώς τροποποιημένη, δίνοντας προτεραιότητα εξόδου στην τεχνητή μεταβλητή σε περίπτωση που υπάρχει δεσμός και σε αυτόν περιλαμβάνεται η τεχνητή μεταβλητή

```
def comp_r_I(x_B,h_l,art_var,A,B):
    '''finds the position of exporting variable exp_var and breaks the tie'''
    x_k=(x_B/h_l.values).replace((np.inf, -np.inf),(0,0))
    q=x_k.index[x_k==min(x_k[x_k>0])]
```

```
    q=q.tolist()
    if len(q)==1:
        exp_var=B.columns.get_loc(q[0])
    else:
        if art_var in q:
            exp_var=B.columns.get_loc(art_var)
        else:
            j=min([A.columns.get_loc(i) for i in q])
            s=A.columns[j]
            exp_var=B.columns.get_loc(s)
    return exp_var
```

Λίστα Κώδικα 8. Υπολογισμός εξερχόμενης μεταβλητής στη φάση 1

Για την εύρεση του αντιστρόφου χρησιμοποιήθηκε η μορφή γινομένου του αντιστρόφου. Για αποφυγή υπολογιστικών λαθών κάθε 100 επαναλήψεις η αντίστροφη μήτρα B^{-1} υπολογίζεται από την αρχή. Ο υπολογισμός της αντίστροφης μήτρας eta πραγματοποιήθηκε με την παρακάτω συνάρτηση

```
def create_eta_inv(B,r,h_l):
    '''creates the inverse eta matrix'''
    I=np.eye(len(B))
    E=np.matrix(I)
    a=-h_l.values/h_l[r]
    a[r]=1/h_l[r]
    a_rev=a.reshape(len(a),1)
    E[:,r]=a_rev
    eta_inv=E
    return eta_inv
```

Λίστα Κώδικα 9. Υπολογισμός Αντίστροφης μήτρας Eta

Για την εισαγωγή των χαλαρών μεταβλητών στο πρόβλημα, την μετατροπή της συνάρτησης από max σε min υλοποιήθηκε η συνάρτηση που υπάρχει στο αρχείο data_trans και παρουσιάζεται παρακάτω:

```
        def standard_form(A,c,b,bounds,obj):
            '''Creates the standard form of the original problem importing the slack variables'''
            extra_var=len(A)
            l_in=[]
            var_col=0
            for i,j in enumerate(bounds):
                for k,m in enumerate(A):
                    if i==k:
                        if j==['>=']:
                            l_in.append(m      +var_col*[0]+[-1]+(extra_var-var_col-1)*[0])
                            var_col+=1
                        elif j==['<=']:
                            l_in.append(m+  var_col*[0]  +  [1]+(extra_var-var_col-1)*[0])
                            var_col+=1
            v,s=[],[]
            for i, j in enumerate(b):
                for k,m in enumerate(l_in):
                    if i==k:
                        if j[0]<0:
                            j[0]=-j[0]
                            v.append(j)
                            m=[-1*x for x in m]
                            s.append(m)
                        else:
                            v.append(j)
                            s.append(m)
            A=s
            b=v
            c=c+extra_var*[0]
```

```
if obj=='max':
    c=[-1*i for i in c]
else:
    c=c
total_var=len(c)
return A,c,b,total_var,extra_var
```

Λίστα Κώδικα 10. Μετατροπή προβλήματος σε κανονική μορφή

3.2 Δημιουργία τυχαίων προβλημάτων

Για την δημιουργία των τυχαίων πυκνών προβλημάτων που χρησιμοποιήθηκαν στη μελέτη του αλγόριθμου υλοποιήθηκαν οι συναρτήσεις που υπάρχουν στο αρχείο dense_problems. Η πρώτη αφορά την εισαγωγή των χαρακτηριστικών που θέλουμε να έχουν τα προς δημιουργία προβλήματα και η δεύτερη τα δημιουργεί. Απαραίτητη ήταν η εισαγωγή της ενσωματωμένης στην Python συνάρτησης rand από το module scipy.parse για την δημιουργία της τυχαίας μήτρας A και της συνάρτησης floor από την βιβλιοθήκη math για την αλλαγή των τιμών του δεξιού μέρους των ανισοτήτων, έτσι ώστε τα προβλήματα που δημιουργούνται να είναι βέλτιστα.

Τα τυχαία προβλήματα είναι είτε μεγιστοποίησης είτε ελαχιστοποίησης και έχουν όλα πυκνότητα 100%. Τα ορίσματα n και m καθορίζουν τις διαστάσεις κάθε προβλήματος, πιο συγκεκριμένα το n είναι το πλήθος των περιορισμών και το m το πλήθος των μεταβλητών. Το εύρος τιμών για τη μήτρα A και τα διανύσματα c και b δίνετε στον παρακάτω πίνακα

Πίνακας 3.1. Εύρος τιμών τυχαίων προβλημάτων

A	-1000	1000
c	-400	900
b	10	100

Τα προβλήματα είναι σε κανονική μορφή, δηλαδή περιλαμβάνουν μόνο ανισοτικούς περιορισμούς της μορφής $\geq$ και $\leq$. Δεν γίνεται αποθήκευση των προβλημάτων . Οι συναρτήσεις δημιουργίας τους έχουν ενσωματωθεί στον κυρίως αλγόριθμο όπου ορίζεται και ο αριθμός των προβλημάτων που κατασκευάζονται σε κάθε διάσταση. Για το λόγο αυτό έχει δοθεί συγκεκριμένη τιμή σπόρου (seed) έτσι ώστε να

υπάρχει η δυνατότητα να επαναληφθεί η κατασκευή των ίδιων προβλημάτων. Κάτωθι παρουσιάζονται οι συναρτήσεις δημιουργίας τυχαίων προβλημάτων

```
def random_lp():
    seed_number=int(input('Please give the seed number 100:',))
    np.random.seed(seed_number)
    n=int(input('Please give the number of rows:',))
    m=int(input('Please give the number of columns:',))
    LP=int(input('Please give the number of lp problems for creation:',))
    eqinlu=input('Please give the range for eqin vector (-1 <=,1 >=):',)
    Alu=input('Please give the range for A matrix:',)
    clu=input('Please give the range for c vector:',)
    blu=input('Please give the range for b vector:',)
    den=input('Please give the density of the problem:',)
    min_card=int(input('Define the type of the objective function: 0 min,1 max, 2
random:',))
    eqinlu=eqinlu.split()
    Alu=Alu.split()
    clu=clu.split()
    blu=blu.split()
    return n,m,LP,eqinlu,Alu,clu,blu,min_card,den
```

Λίστα Κώδικα 11. Συνάρτηση εισαγωγής δεδομένων

```
def optimalrandom(n,m,eqinlu,Alu,clu,blu,min_card,den):
    Al=int(Alu[0])
    Au=int(Alu[1])
    A=np.round((Au-Al+1)*rand(n,m,format='coo',density=float(den)))
    A=A.toarray()
    A=np.where(A!=0,A+Al,A)
    A=A.tolist()
    cl=int(clu[0])
    cu=int(clu[1])
    c=np.round((cu-cl+1)*np.random.rand(1,m)) + cl
```

```
ca=c.tolist()
c=[i for i in ca[0]]
bl=int(blu[0])
bu=int(blu[1])
b=np.round((bu-bl+1)*np.random.rand(n,1))+bl
b=b.tolist()
bounds=[]
for i in range(n):
    re=np.random.choice(eqinlu)
    if re=='-1':
        bounds.append(['<='])
        b[i]=[floor((8000-200+1)*np.random.rand()) + 200]
    else:
        bounds.append(['>='])
if min_card==2:
    s=[0,1]
    min_card=np.random.choice(s)
if min_card==0:
    min_card='min'
elif min_card==1:
    min_card='max'
obj=min_card
return A,c,b,bounds,obj
```

Λίστα Κώδικα 12. Δημιουργία τυχαίων προβλημάτων

Κεφάλαιο 4

Ο solver της Python

4.1 Η συνάρτηση linprog

Η Python διαθέτει μια έτοιμη συνάρτηση για την επίλυση γραμμικών προβλημάτων την scipy.optimize.linprog( ). Η συνάρτηση αυτή επιλύει προβλήματα της μορφής

Minimize: c^T * x

s.t.: A_ub * x <= b_ub

A_eq * x == b_eq

Τα ορίσματα που δέχεται η συνάρτηση αυτή είναι τα εξής:

- Διάνυσμα c

  Είναι οι συντελεστές των μεταβλητών της αντικειμενικής συνάρτησης
- Μήτρα A_ub

  Περιλαμβάνει τους συντελεστές των μεταβλητών των ανισοτικών περιορισμών
- Διάνυσμα b_ub

  Είναι οι τιμές του δεξιού μέρους των ανισοτικών περιορισμών
- Μήτρα A_eq

  Περιλαμβάνει τους συντελεστές των μεταβλητών των ισοτικών περιορισμών
- Διάνυσμα b_eq

  Είναι οι τιμές του δεξιού μέρους των ισοτικών περιορισμών
- Ακολουθία bounds

  Ορίζει τα όρια τιμών κάθε μεταβλητής του προβλήματος. Όταν μια μεταβλητή δεν έχει κατώτατο ή ανώτατο όριο χρησιμοποιείται η τιμή None. Η εξ ορισμού τιμή του ορίσματος είναι (0, None) το οποίο υποδηλώνει ότι όλες οι μεταβλητές είναι μη αρνητικές. Εάν δοθεί μόνο ένα ζεύγος τιμών ορίων, αυτό θα εφαρμοστεί σε όλες τις μεταβλητές.
- Μέθοδος

Ορίζει ποία μέθοδος θα εφαρμοστεί για την επίλυση του προβλήματος. Υπάρχουν δύο διαθέσιμες μέθοδοι: εσωτερικών σημείων και simplex. Εξ ορισμού χρησιμοποιείται η μέθοδος simplex.

- Συνάρτηση επανάκλησης.

  Μπορεί να υπάρχει ως όρισμα μόνο στη μέθοδο simplex. Εάν υπάρχει καλείται σε κάθε επανάληψη. Πρέπει να έχει την υπογραφή ``callback(xk, **kwargs)`` όπου το xk είναι το διάνυσμα της τρέχουσας λύσης και το kwargs είναι ένα λεξικό το οποίο περιλαμβάνει: το τρέχον ταμπλό simplex, τον τρέχον αριθμό επανάληψης, το τρέχον ζεύγος γραμμής και στήλης περιστροφής που θα χρησιμοποιηθεί στην επόμενη επανάληψη, την φάση στην οποία βρίσκεται ο αλγόριθμος (φάση I ή φάση II) και τη βάση B.

- Επιλογές

  Ένα λεξικό το οποίο μπορεί να περιλαμβάνει τις παρακάτω γενικές επιλογές: Μέγιστο αριθμό επαναλήψεων και μια Boolean έκφραση που αν πάρει τιμή True θα εμφανίζει μήνυμα.

Οι τιμές που επιστρέφει η συνάρτηση αναφέρονται παρακάτω:

- x

  Διάνυσμα x. Περιλαμβάνει τη βέλτιστη λύση.

- fun

  Περιλαμβάνει τη βέλτιστη τιμή της αντικειμενικής συνάρτησης.

- slack

  Διάνυσμα χαλαρών μεταβλητών. Κάθε χαλαρή μεταβλητή αντιστοιχεί σε έναν ανισοτικό περιορισμό. Αν η χαλαρή μεταβλητή έχει τιμή μηδέν, ο αντίστοιχος περιορισμός είναι ενεργός.

- success

  Boolean έκφραση. Επιστρέφει True αν ο αλγόριθμος κατάφερε να βρει μια βέλτιστη λύση

- status

  Μια ακέραια τιμή που αντιπροσωπεύει την κατάσταση τερματισμού του αλγόριθμου. Η τιμή αυτή μπορεί να είναι μια από τις παρακάτω:

  0: Ο αλγόριθμος έλυσε το πρόβλημα βελτιστοποίησης

1: Ο αριθμός των επαναλήψεων έφτασε στο ανώτατο όριο

2: Το πρόβλημα φαίνεται να είναι αδύνατο

3: Το πρόβλημα φαίνεται να είναι απεριόριστο

- nit

  Ο αριθμός των επαναλήψεων που εκτελέστηκαν

- str

  Ένα μήνυμα για την κατάσταση τερματισμού του αλγόριθμου

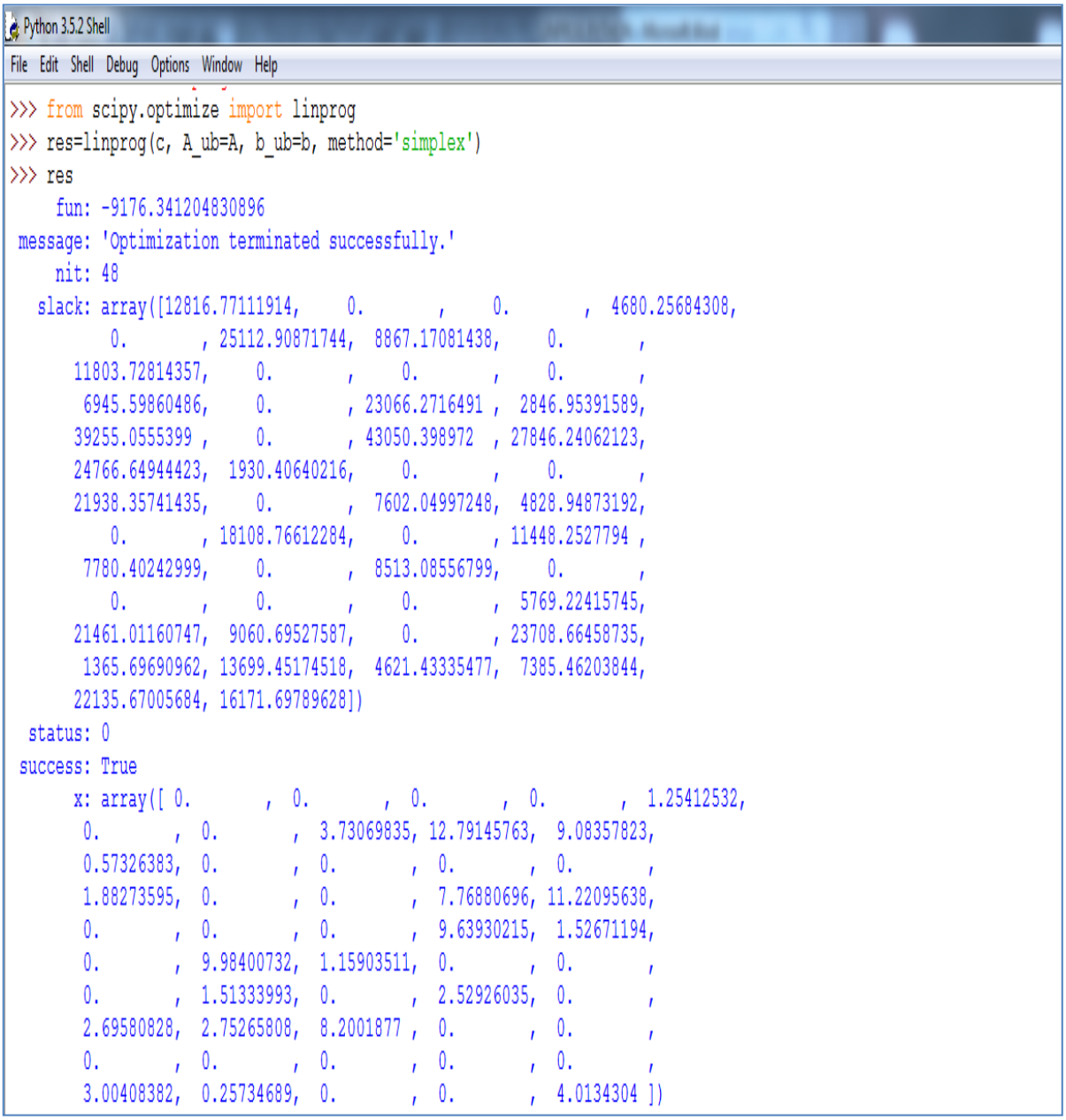
4.2 Εφαρμογή solver της Python

Για να εφαρμόσουμε την έτοιμη συνάρτηση της python για την επίλυση γραμμικών προβλημάτων δημιουργήθηκαν δύο τυχαία βέλτιστα προβλήματα διαστάσεων 50x50 και 100x100 με χρήση των συναρτήσεων που παρουσιάστηκαν στο προηγούμενο κεφάλαιο. Τα προβλήματα αυτά είναι της μορφής:

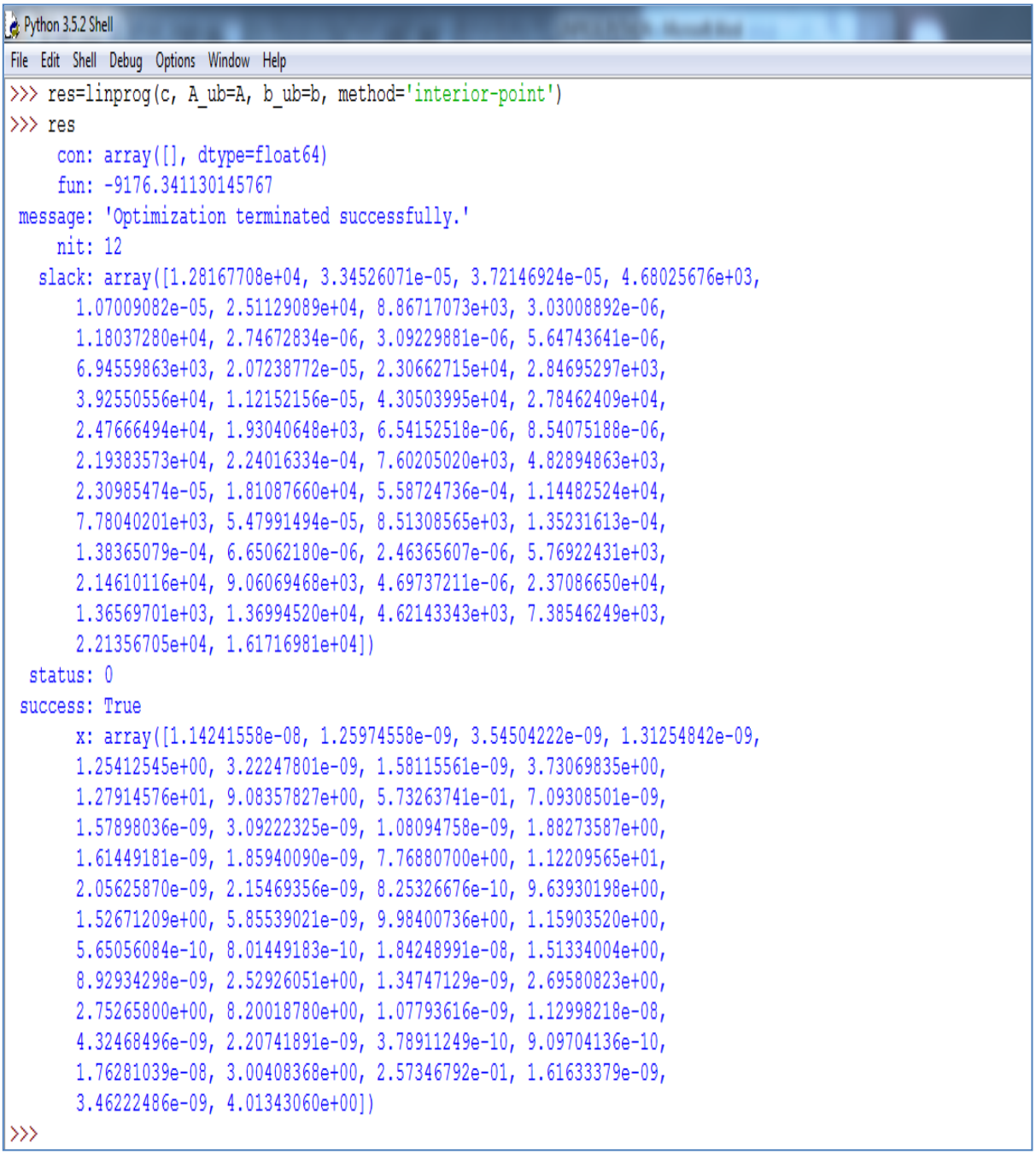
$$\begin{aligned} \min \quad & z = c^T x \\ \text{μ.π.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

με εύρος τιμών για τα A, c και b τις: A= -1000 1000, c= -400 900 και b= 10 100 και η πυκνότητα τους είναι 100%.

Για το πρόβλημα διαστάσεων 50x50 τα αποτελέσματα που προέκυψαν με χρήση της linprog και εφαρμόζοντας και τις δύο διαθέσιμες μεθόδους είναι:

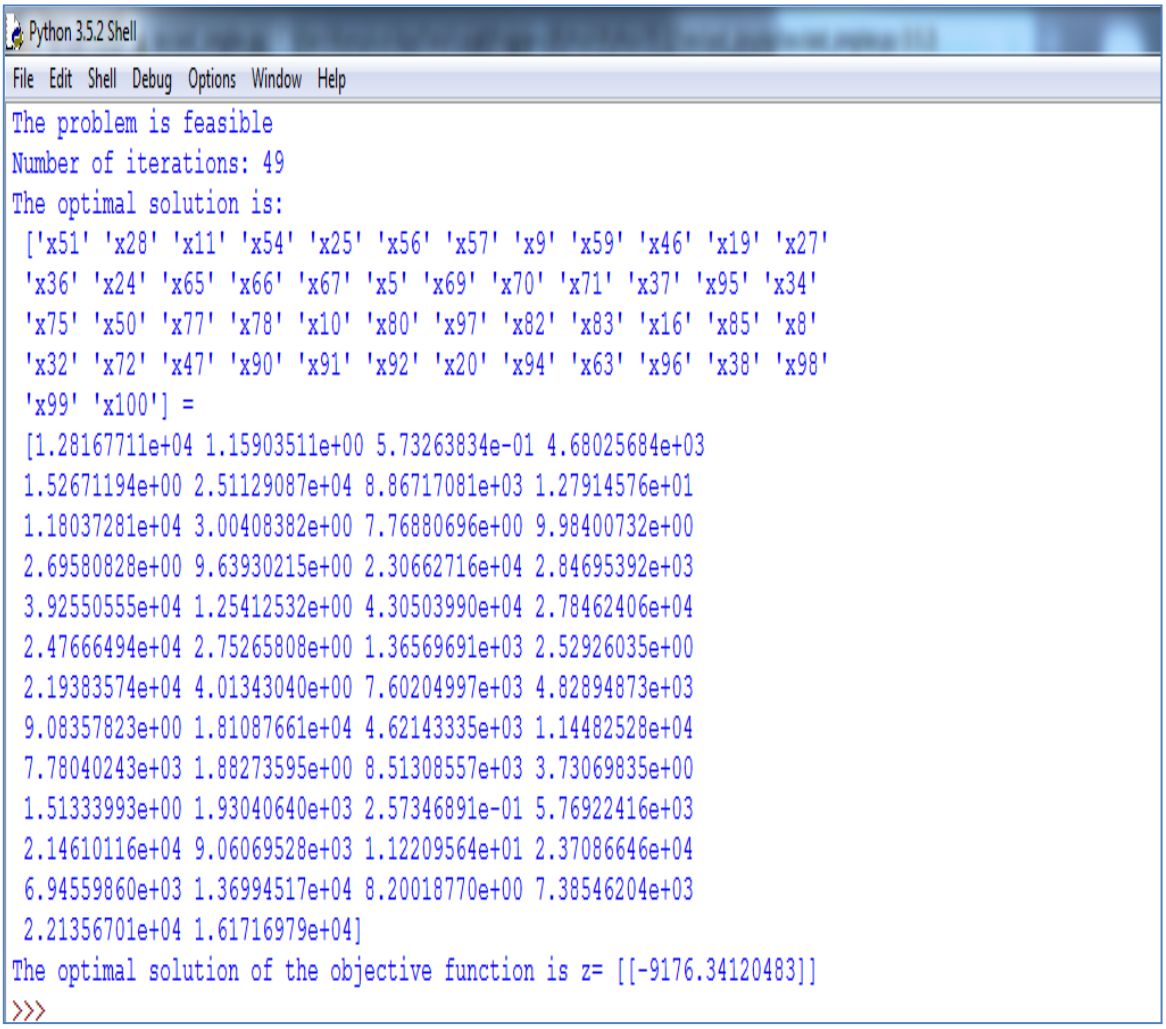


Σχήμα 4.2.1 linprog(method=simplex) για πρόβλημα διαστάσεων 50x50



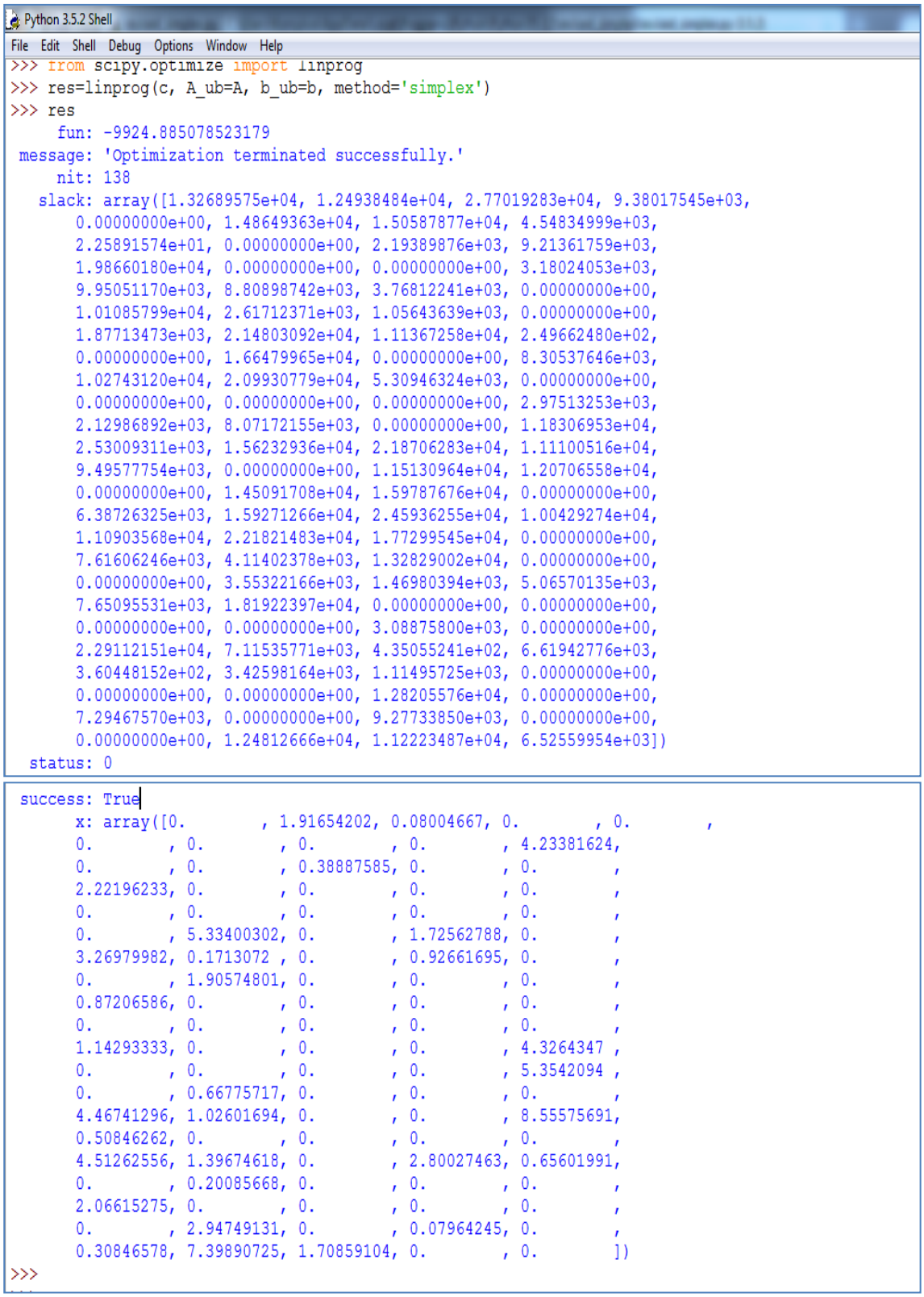
Σχήμα 4.2.2 linprog(method=interior point) για πρόβλημα διαστάσεων 50x50

Το ίδιο πρόβλημα με χρήση του υλοποιημένου στην παρούσα εργασία αλγόριθμου δίνει



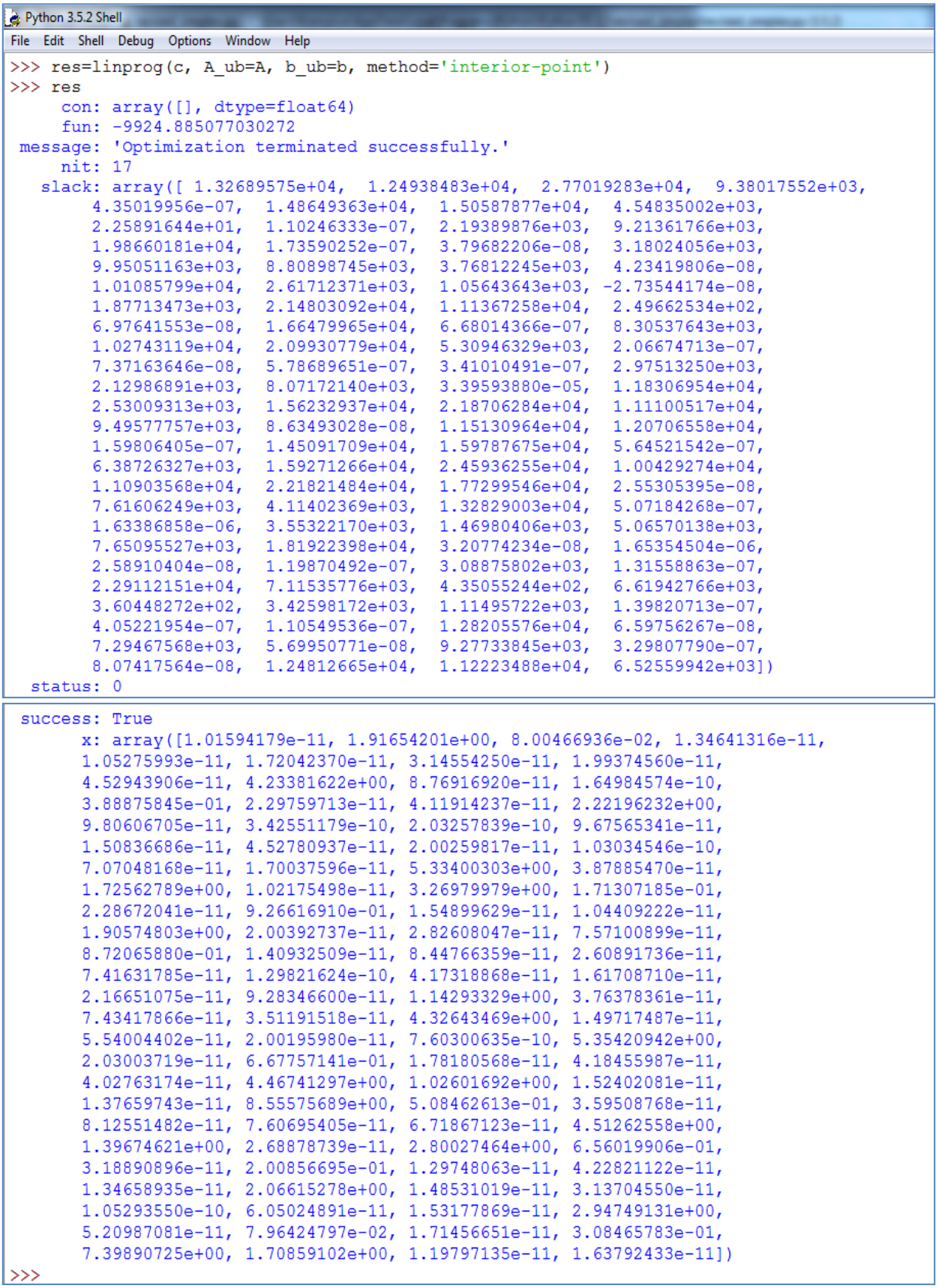
Σχήμα 4.2.3 Revised simplex για το πρόβλημα διαστάσεων 50x50

Για το πρόβλημα διαστάσεων 100x100 τα αποτελέσματα που προέκυψαν με χρήση της linprog και εφαρμόζοντας και τις δύο διαθέσιμες μεθόδους είναι:



```
Python 3.5.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>> from scipy.optimize import linprog
>>> res=linprog(c, A_ub=A, b_ub=b, method='simplex')
>>> res
     fun: -9924.885078523179
 message: 'Optimization terminated successfully.'
     nit: 138
   slack: array([1.32689575e+04, 1.24938484e+04, 2.77019283e+04, 9.38017545e+03,
       0.00000000e+00, 1.48649363e+04, 1.50587877e+04, 4.54834999e+03,
       2.25891574e+01, 0.00000000e+00, 2.19389876e+03, 9.21361759e+03,
       1.98660180e+04, 0.00000000e+00, 0.00000000e+00, 3.18024053e+03,
       9.95051170e+03, 8.80898742e+03, 3.76812241e+03, 0.00000000e+00,
       1.01085799e+04, 2.61712371e+03, 1.05643639e+03, 0.00000000e+00,
       1.87713473e+03, 2.14803092e+04, 1.11367258e+04, 2.49662480e+02,
       0.00000000e+00, 1.66479965e+04, 0.00000000e+00, 8.30537646e+03,
       1.02743120e+04, 2.09930779e+04, 5.30946324e+03, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 2.97513253e+03,
       2.12986892e+03, 8.07172155e+03, 0.00000000e+00, 1.18306953e+04,
       2.53009311e+03, 1.56232936e+04, 2.18706283e+04, 1.11100516e+04,
       9.49577754e+03, 0.00000000e+00, 1.15130964e+04, 1.20706558e+04,
       0.00000000e+00, 1.45091708e+04, 1.59787676e+04, 0.00000000e+00,
       6.38726325e+03, 1.59271266e+04, 2.45936255e+04, 1.00429274e+04,
       1.10903568e+04, 2.21821483e+04, 1.77299545e+04, 0.00000000e+00,
       7.61606246e+03, 4.11402378e+03, 1.32829002e+04, 0.00000000e+00,
       0.00000000e+00, 3.55322166e+03, 1.46980394e+03, 5.06570135e+03,
       7.65095531e+03, 1.81922397e+04, 0.00000000e+00, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 3.08875800e+03, 0.00000000e+00,
       2.29112151e+04, 7.11535771e+03, 4.35055241e+02, 6.61942776e+03,
       3.60448152e+02, 3.42598164e+03, 1.11495725e+03, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.28205576e+04, 0.00000000e+00,
       7.29467570e+03, 0.00000000e+00, 9.27733850e+03, 0.00000000e+00,
       0.00000000e+00, 1.24812666e+04, 1.12223487e+04, 6.52559954e+03])
  status: 0
 success: True
       x: array([0.        , 1.91654202, 0.08004667, 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 4.23381624,
       0.        , 0.        , 0.38887585, 0.        , 0.        ,
       2.22196233, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 5.33400302, 0.        , 1.72562788, 0.        ,
       3.26979982, 0.1713072 , 0.        , 0.92661695, 0.        ,
       0.        , 1.90574801, 0.        , 0.        , 0.        ,
       0.87206586, 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       1.14293333, 0.        , 0.        , 0.        , 4.3264347 ,
       0.        , 0.        , 0.        , 0.        , 5.3542094 ,
       0.        , 0.66775717, 0.        , 0.        , 0.        ,
       4.46741296, 1.02601694, 0.        , 0.        , 8.55575691,
       0.50846262, 0.        , 0.        , 0.        , 0.        ,
       4.51262556, 1.39674618, 0.        , 2.80027463, 0.65601991,
       0.        , 0.20085668, 0.        , 0.        , 0.        ,
       2.06615275, 0.        , 0.        , 0.        , 0.        ,
       0.        , 2.94749131, 0.        , 0.07964245, 0.        ,
       0.30846578, 7.39890725, 1.70859104, 0.        , 0.        ])
>>>
```

Σχήμα 4.2.4 linprog(method=simplex) για πρόβλημα διαστάσεων 100x100



Σχήμα 4.2.5 linprog(method=simplex) για πρόβλημα διαστάσεων 100x100

Το ίδιο πρόβλημα με χρήση του υλοποιημένου στην παρούσα εργασία αλγόριθμου δίνει

Σχήμα 4.2.6 Revised simplex για το πρόβλημα διαστάσεων 100x100

Κεφάλαιο 5

Υπολογιστική μελέτη

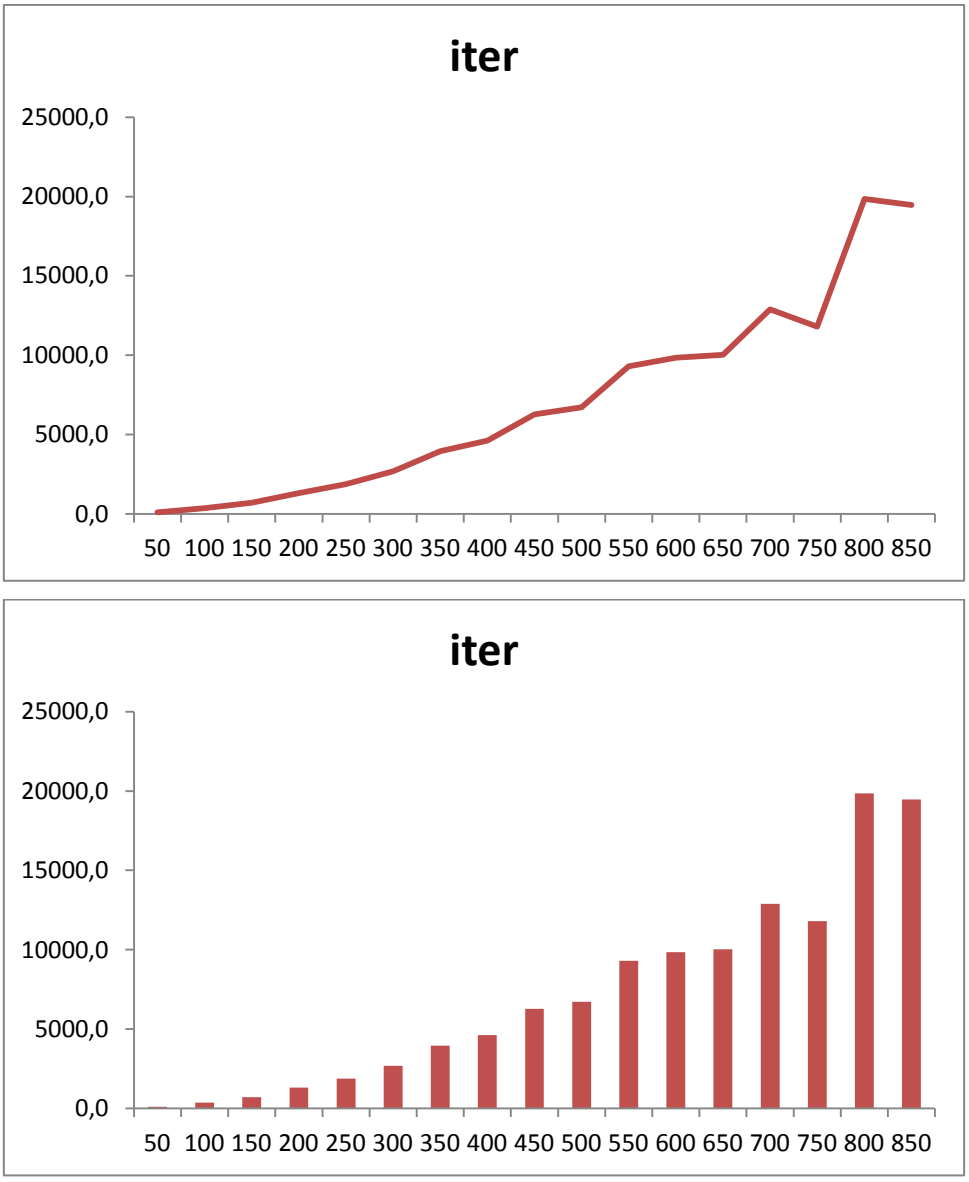
5.1 Αποτελέσματα

Το πρόγραμμα εκτελέστηκε σε τυχαία προβλήματα διαστάσεων από 50x50 έως 850x850 με βήμα 50. Σε κάθε διάσταση λύθηκαν 10 βέλτιστα προβλήματα και εξήχθησαν οι μέσοι όροι των μετρήσεων. Τα πειράματα έγιναν σε PC με 2,16GHz Intel Celeron επεξεργαστή, 4GB RAM και με λειτουργικό σύστημα Windows 7. Στον πίνακα 5.1.1 παρουσιάζονται τα αποτελέσματα που προέκυψαν από την εφαρμογή του αλγόριθμου. Η πρώτη στήλη δίνει τις διαστάσεις του προβλήματος, η δεύτερη των αριθμό επαναλήψεων, η τρίτη την απαίτηση του υπολογιστικού χρόνου για την αντιστροφή της βάσης, η τέταρτη τον ολικό χρόνο εκτέλεσης του αλγόριθμου, η πέμπτη το ποσοστό λόγου της αντιστροφής της βάσης προς τον ολικό χρόνο εκτέλεσης και η τελευταία το συντελεστή αύξησης του χρόνου.

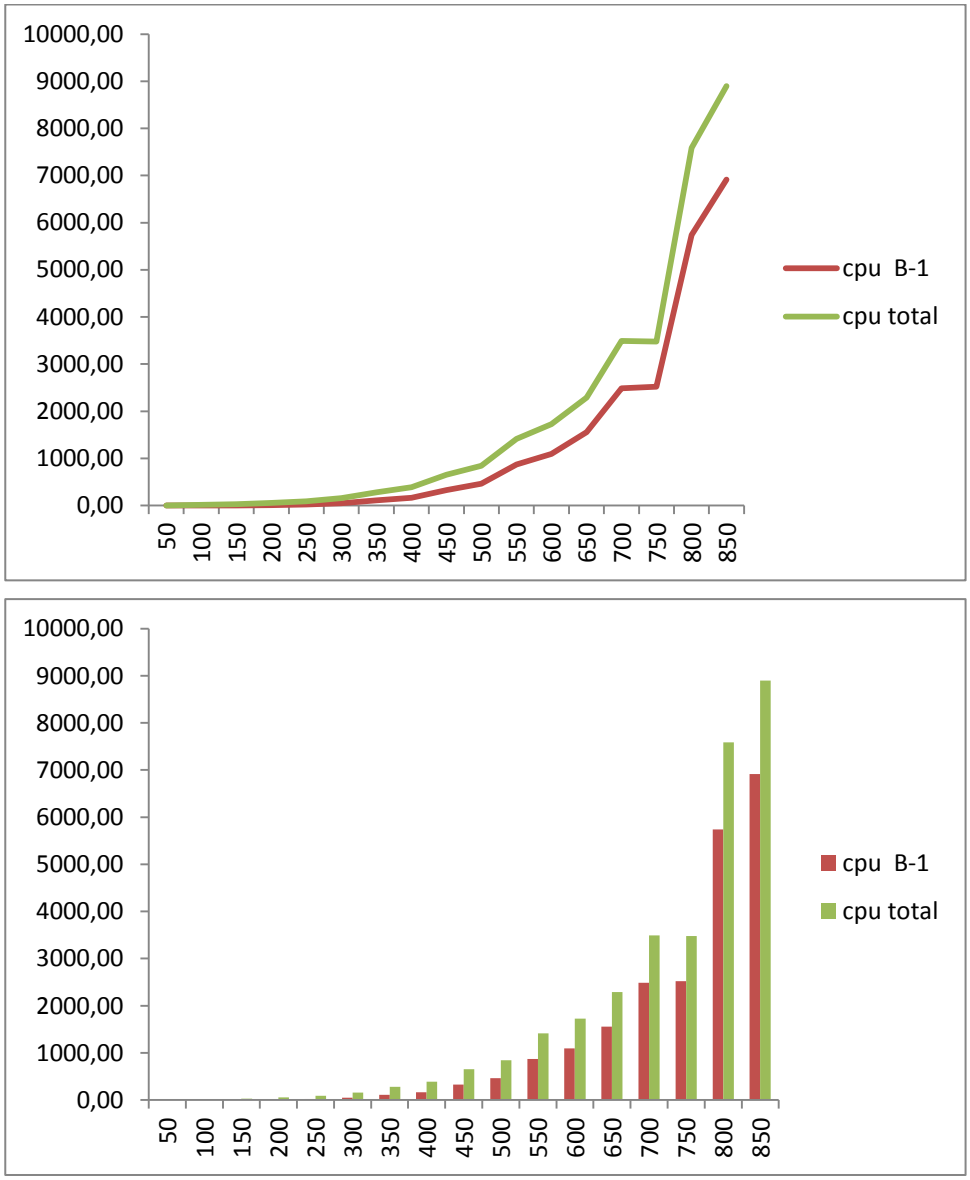
Πίνακας 5.1.1 Υπολογιστικά αποτελέσματα αλγόριθμου

nxn	iter	cpu B^{-1}	cpu total	cpu(B^{-1} / total) (%)	Αύξηση cpu
50x50	107,7	0,13	3,48	3,64	1,00
100x100	356,2	0,85	14,32	5,91	4,12
150x150	703,4	2,53	26,10	9,69	7,51
200x200	1321,0	8,73	55,25	15,79	15,90
250x250	1886,5	20,44	89,96	22,73	25,88
300x300	2676,7	50,95	160,20	31,80	46,09
350x350	3954,3	106,25	279,53	38,01	80,42
400x400	4626,8	166,03	387,47	42,85	111,48
450x450	6275,3	325,17	654,24	49,70	188,23
500x500	6727,3	464,68	843,66	55,08	242,73
550x550	9302,7	868,28	1414,69	61,38	407,03
600x600	9836,8	1093,08	1725,37	63,35	496,41
650x650	10026,5	1555,97	2289,74	67,95	658,79
700x700	12888,1	2486,32	3489,20	71,26	1003,89
750x750	11795,4	2519,64	3474,88	72,50	999,77
800x800	19843,1	5742,14	7585,11	75,70	2182,34
850x850	19471,6	6916,08	8899,50	77,71	2560,50

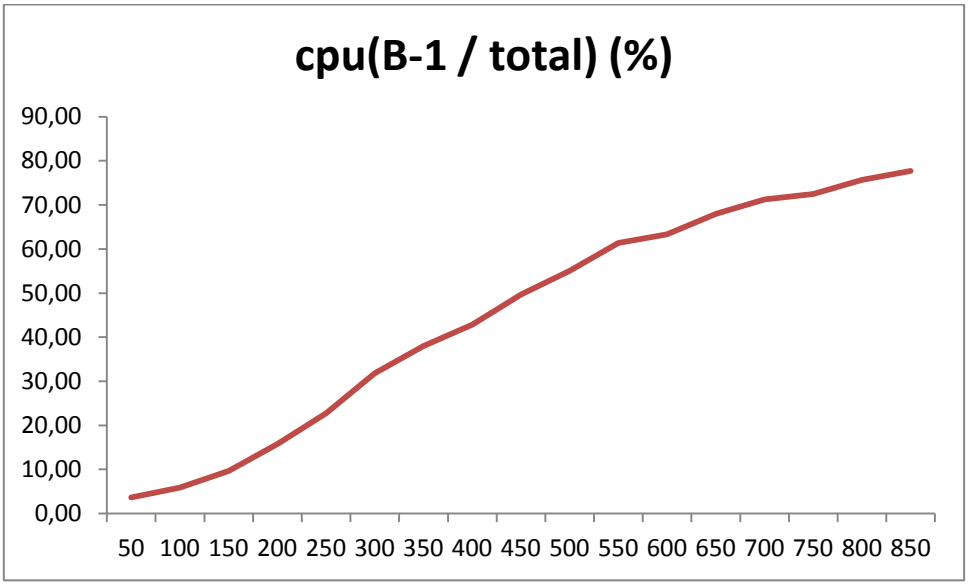
Διαγραμματικά τα παραπάνω αποτελέσματα παρουσιάζονται κάτωθι.



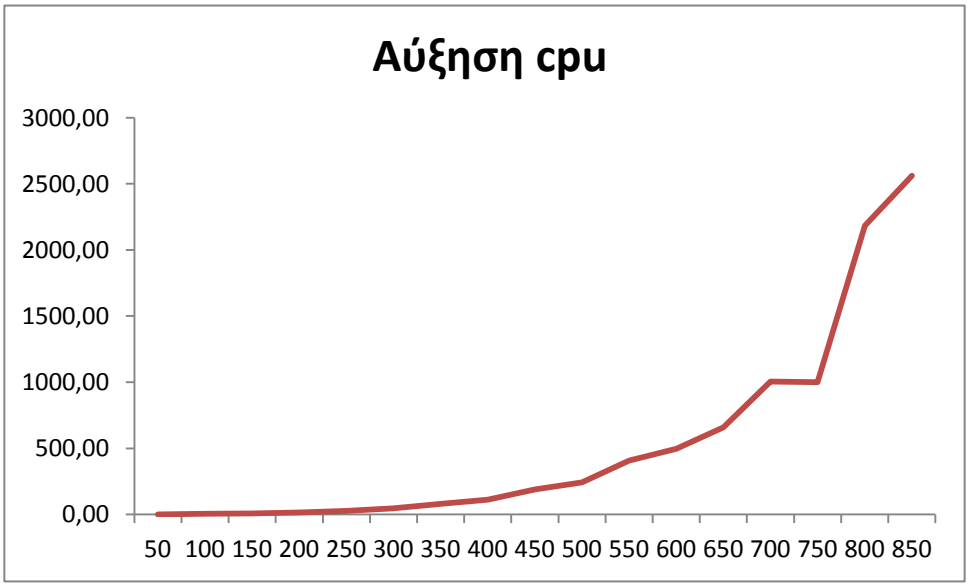
Σχήμα 5.1.1 Αριθμός επαναλήψεων ανά nxn



Σχήμα 5.1.2 Total cpu και cpu B^{-1}



Σχήμα 5.1.3 Λόγος (%) cpu B^{-1} προς total cpu



Σχήμα 5.1.4 Αύξηση χρόνου εκτέλεσης ανά διάσταση προβλήματος

Από τα παραπάνω παρατηρούμε ότι καθώς αυξάνουν οι διαστάσεις του προβλήματος , ο χρόνος που απαιτείται για τον υπολογισμό της αντιστροφής της βάσης καταλαμβάνει όλο και μεγαλύτερο μέρος του συνολικού χρόνου επίλυσης. Επίσης ο υπολογιστικός χρόνος επίλυσης οδηγείται σταδιακά σε εκθετική αύξηση. Στα πυκνά προβλήματα τα οποία μελετήσαμε, δεν γίνεται εκμετάλλευση των μηδενικών στοιχείων της μήτρας Α λόγω απουσίας αυτών. Έτσι ο υπολογιστικός χρόνος αυξάνει με ταχύτατο

ρυθμό. Για παράδειγμα τα προβλήματα διαστάσεων 700x700 απαιτούν 1000 φορές περισσότερο χρόνο για να λυθούν από τα προβλήματα διαστάσεων 50x50 ενώ τα προβλήματα διαστάσεων 850x850 απαιτούν 2500 φορές το χρόνο των προβλημάτων 50x50.

5.2 Σύνοψη και συμπεράσματα

Ο αναθεωρημένος αλγόριθμος simplex είναι ο πλέον χρησιμοποιημένος για επίλυση γραμμικών προβλημάτων. Στην πλειοψηφία τους αυτά είναι αραιά και ο αλγόριθμος εκμεταλλεύεται αυτή την ιδιότητα. Παρόλο που τα πυκνά προβλήματα είναι σπάνια, συναντώνται συχνά σε κάποιες σημαντικές εφαρμογές όπως η κατηγοριοποίηση κειμένων, ο ψηφιακός σχεδιασμός φίλτρων και η επεξεργασία εικόνας. Για αυτά τα προβλήματα η αύξηση του χρόνου επίλυσής τους συναρτήσει της αύξησης των διαστάσεων τους είναι ιδιαίτερα μεγάλη με αποτέλεσμα ο αναθεωρημένος αλγόριθμος simplex να μην είναι αποδοτικός.

Για το λόγο αυτό και έχουν δημιουργηθεί άλλοι αλγόριθμοι όπως για παράδειγμα ο αλγόριθμος εξωτερικών σημείων τύπου simplex ο οποίος έχει αποδειχθεί ότι είναι αποδοτικότερος από τον αλγόριθμο simplex σε τυχαία πυκνά προβλήματα αλλά παρουσιάζει το μειονέκτημα της κατασκευής καλής κατεύθυνσης μετακίνησης.

Μια μελλοντική μελέτη θα ήταν χρήσιμο να περιλαμβάνει προβλήματα μεγαλύτερων διαστάσεων και παράλληλη υλοποίηση του αλγόριθμου .

Αναφορικά με την υλοποίηση του αλγόριθμου στη γλώσσα προγραμματισμού Python προέκυψαν τα εξής στοιχεία. Οι βιβλιοθήκες της γλώσσας αποτελούν ένα σημαντικό εργαλείο που έχει για τη διευκόλυνση της συγγραφής κώδικα ποικίλων εφαρμογών. Η ίδια η γλώσσα βοηθάει στην απλή γραφή κώδικα. Το μειονέκτημά της είναι ότι τα δεδομένα μπορούν να απεικονιστούν με διαφορετικές μορφές (όπως λίστες, διανύσματα, πίνακες) και χρειάζεται η μετατροπή από την μια μορφή στην άλλη.

Βιβλιογραφία

Badr M.E. (2006). *Παράλληλος Προγραμματισμός Αλγόριθμων για Προβλήματα Γραμμικού Προγραμματισμού*. Phd στην Εφαρμοσμένη Πληροφορική, Σχολή Οικονομικών και Κοινωνικών Επιστημών, Πανεπιστήμιο Μακεδονίας

Dantzig B.G.(1963). *Linear Programming and Extensions*, Princeton, NJ: Princeton University Press

Downey B.A. *Think Python 2-How to Think Like a Computer Scientist*, 2nd Edition. Available at:< www.thinkpython2.com> (2015). Green Tea Press

Guttag V.J. (2013*). Introduction to Computation and Programming Using Python*, Expanded Edition, MIT press, Cambridge, Massachusetts

Hetland L.M.(2010). *Python Algorithms-Mastering Basic Algorithms in the Python Language*, Apress

Langtagen P.H.(2012). *A Primer on Scientific Programming with Python*, third edition, Springer

Paparrizos K., Samaras N., Sifaleras A. (2015). *Exterior Point Simplex-type Algorithms for Linear and Network Optimization Problems*. Annals of Operations Research 229(1):607-633

Paparrizos K., Samaras N., Stefanides G. (2003*). An efficient simplex type algorithm for sparse and dense linear programs*. European Journal of Operation Research 148(2):323-334

Pilgrim M., *Dive Into Python 3*, Apress. Available at:< www.diveinto.org/python3/>

Ploskas N.,Samaras N. (2014). *Pivoting Rules for the Revised Simplex Algorithm*. Yogoslav Journal of Operations Research 24 (2014).Number 3, 321-332

Severance R.C. *Python for Everybody-Exploring Data Using Python 3*. Available at: <www.py4e.com >(2016)

Varoquaux G.,Gouillart E.,Vahtras O. *Scipy Lecture Notes*. Available at: <www.scipy-lectures.org> (2015)

Αγγελιδάκης Α.Ν. *Εισαγωγή στον προγραμματισμό με την Python*. Διαθέσιμο: <http://aggelid.mysch.gr/pythonbook> Ά έκδοση (2015).

Λεβεντέας Δ. *Οδηγός Python Μέσω Παραδειγμάτων*. Διαθέσιμο: <www.openbook.gr.> Ομάδα TasPython (2010)

Σαμαράς Ν. (2001). *Υπολογιστικές Βελτιώσεις και Αποτελεσματική Υλοποίηση Περιστροφικών Αλγόριθμων Δύο Δρόμων*. Phd στην Εφαρμοσμένη Πληροφορική, Σχολή Οικονομικών και Κοινωνικών Επιστημών, Πανεπιστήμιο Μακεδονίας